Udacity Self Driving Car Nanodegree –

Project: Behavioral Cloning

By:

Saurabh Sohoni

Udacity profile: <https://profiles.udacity.com/p/u183059>

Contents

1. Introduction 2

2. Project Goals 3

3. Implementation of Project Rubric Points 3

3.1 Submission of code and output files 3

3.2 Functional Python code 3

3.3 Model Architecture and Training Strategy 4

3.4 Low level analysis of network and training 5

4. References 8

# Introduction

This document acts in support with code and output files created in order to implement ‘Behavioral Cloning’ project. In this project, driving behavior of a human is imitated by an autonomous system to drive a car around the Udacity simulator track.

# Project Goals

Following were the goals of this project:

1. Use a simulator to record driving behavior of a human on sample round track. The recorded data consists of images taken from cameras mounted on the hood of the car. A steering angle measurement is captured for corresponding image.
2. Build and create a convolutional neural network (referred as CNN, hereafter) which leverages use of deep learning methodologies for training a model on captured data set. Implementation this CNN in Keras.
3. Predict steering angle for a new image passed to the CNN with the help of model created from training data.
4. Test the model to drive the car in ‘Autonomous mode’ of simulator ensuring that the car doesn’t leave any part of the track.
5. Summarize the approach and results in a report

# Implementation of Project Rubric Points

Following section lists down various rubric points for this project and also details out the implementation strategy followed:

## Submission of code and output files

The submission includes following files:

1. model.py – This python file contains code to build CNN and train the model of recorded data set.
2. model.h5 – This file contains the Keras model of built by using CNN implemented in model.py.
3. drive.py – This python file contains code for running the simulator in autonomous mode. Methods in this file take the image and the trained model and predicts steering angle. This steering angle is then passed to the simulator to control steering of the car
4. video.mp4 – Video recording of the car successfully running one lap on the track provided in the simulator. The simulation was run in 512 x 384 resolution with graphics quality set to fastest.
5. writeup\_report.pdf – Current file summarizing about the project goals and strategies implemented.

## Functional Python code

1. With the help of trained model saved in model.h5 and drive.py, car in the Udacity simulator track was driven in autonomous mode around the selected track by using following command in the terminal:

python drive.py model.h5

1. CNN model was built in model.py. The python code was divided into modules by devising different methods for different functionality. It also followed the following coding approach:
   * Import of external libraries
   * Declaration and initialization of global variables
   * Declaration of Utility methods
   * Declaration of methods used in data pre-processing
   * Definition of Keras generator and CNN model architecture
   * main() method used for training the model on recorded data set and exporting it to model.h5

All methods were provided with code comment blocks to brief on the processing done inside the method.

## Model Architecture and Training Strategy

Model published by the team at NVIDIA Corporation in the paper ‘End to End Learning for Self-Driving Cars’ was perfect for training a car to drive itself. The layers used in this model are shown below:

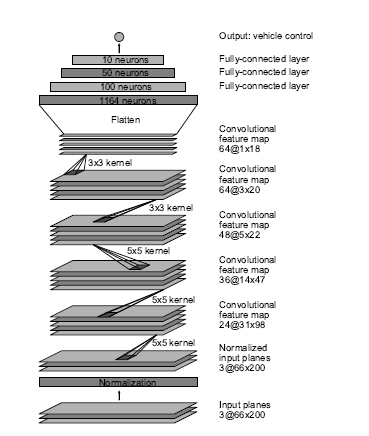


Figure 1: Model architecture proposed by team at NVIDIA. Image from original paper published under the name ‘End to End Learning for Self-Driving Cars’

The architecture used in this project is a clone of the one described above. The dimensions for image vector at the input layer was different (160 x 80 for this project) than the one used by team at NVIDIA. This is due to image re-sizing applied in one of the pre-processing techniques.

The architecture was modified to include dropout layers and the final architecture devised for this project had following features:

1. Input layer accepting images of size 160 x 80 x 3 (3 planes for colored images).
2. Batch Normalization layer just after Input layer to normalize the dataset.
3. Three convolutional layers with 5 x 5 filter kernel, each layer followed by a ReLU (Rectified Linear Units) activation to introduce non-linearity.
4. Two convolutional layers with 3 x 3 filter kernel, each layer followed by a ReLU (Rectified Linear Units) activation.
5. Vector space was flattened to run fully connected layers here after.
6. Two fully connected layers, each followed by a ReLU activation and a dropout layer with 50% as keep probability.
7. This problem being a regression problem, last fully connected layer with 1 x 1 output predicting the steering angle measurement was employed at the output node.

The next section explains functionality of each layer in detail.

## Low level analysis of network and training

This section details out key features of different layers used in the model architecture.

#### **3.4.1 Data Capture:**

Udacity had provided a dataset comprising of roughly 24000 images corresponding for little more than one lap of manual driving. This dataset was enhanced with left and right lane recovery data captured specifically to bring back the car to center of lane recovering from any unexpected deviation. Also, the car seemed to wobble when approaching the bridge for a couple of times and was fixed with more data recorded around the bridge. After summing up, a total of roughly 35,000 images were available just from training samples.

#### **3.4.2 Data Pre-processing:**

Following pre-processing techniques were applied before feeding the data to CNN:

1. Cropping of images to discard features in top portion of image:

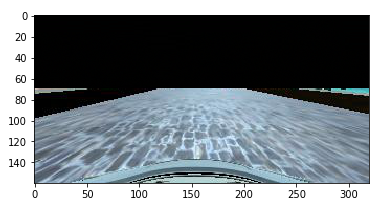
An example is shown below:

Figure 2b: Image with top portion sliced to remove background interference [2b]

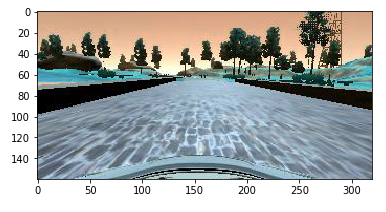


Figure 2a: Original image captured by center camera [2a]

Since lanes line lie in roughly bottom 55% of the image in vertical axis, upper portion was removed to avoid interference of scenic features in training of model. The upper cropped portion was replaced with black colored pixels.

1. Image resizing:

All cropped images were resized from 320 x 160 to 160 x 80 (dimensions are width x height) as shown below:

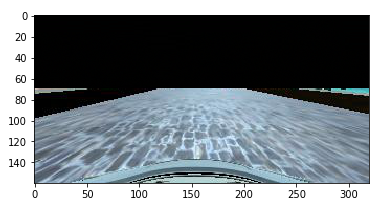
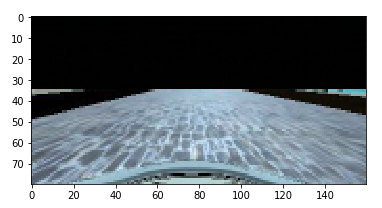


Figure 3b: Resized image (160 x 80) [2c]

Figure 4a: Original image (320 x 160) [2b]

As evitable by just looking at the images, most of the important features were retained and images with smaller dimensions are faster to train with a CNN architecture.

1. Vertical flipping of images:

The track in Udacity simulator was circular and curving towards the left. This introduced a left bias in training data, and also would have introduced a bias in predictions. To avoid this, all images were flipped horizontally. This also helped in doubling the training dataset. An example is shown below:

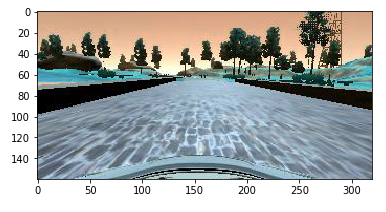


Figure 5b: Vertically flipped image [2d]

Figure 6a: Original training image [2a]

#### **3.4.3 Using Left, Center and Right camera images:**

Udacity simulator captured three images each from left camera, center camera and right camera respectively for car driven in each frame of simulation. But the steering angles measurements was only for the image corresponding to one captured by center camera. To make use of left and right camera images, a steering threshold of 0.25 was employed as per calculations given below:

* Range of steering data: -1 to 1 with positive values corresponding to right turn and negative to left turn.
* For left camera image, 0.25 was added to the steering angle of center image to force car to turn more to the right.
* For right camera images, 0.25 was subtracted from the steering angle of center image to force car to turn more the left.
* The steering values obtained were multiplied by -1 when image was flipped vertically

Hence for each frame of simulation, 3 images were provided by the simulator and 3 images were obtained after flipping. This helped in increasing the size of training data set even more.

#### **Key features of model architecture**

Following were the key features of various layers employed in the CNN architecture:

1. Batch Normalization of Input: This ensured the data set is balanced and that the number of samples for a particular scenario does not add bias in the prediction from CNN
2. 5 x 5 convolutional layer: Three layers with 5 x 5 kernel were employed to pick features such as lane lines, boundary of track, information about terrain, etc.
3. 3 x 3 convolutional layer: Two layers with 3 x 3 kernel were employed to pick detailed features such as edges of track, markings on lane lines, etc.
4. Fully connected layers: Three fully connected layers (including output layer) merged the features captured by convolutional layers and iterated over smaller number of samples to further reduce the cross entropy.
5. ReLU activation: ReLU activation introduced non-linearity in the network
6. Dropout after fully connected layers: Dropout layers ensured that the model doesn’t over fit to the training data and that it generalizes easily to any new image during testing.

#### **Training and Validation dataset:**

Training images were split into two dataset namely training and validation with validation set containing 20% of samples. This ensured that the network does not over fit to the training data.

#### **Learning rate and number of epochs:**

For training, Adam optimizer was used. Adam optimizer is an adaptive optimizer which decreases the learning rate as the number of epochs increases. This ensures that the model doesn’t get stuck in local minima and network loss is decreased gradually. Training was carried out over 4 epochs in total. For each epoch, the data was randomly shuffled. After the 4th epoch, the validation loss was close to 0.025.

# References

1. [1] Mariusz Bojarski et al: ‘End to End Learning for Self-Driving Cars’. Cornell University Library, Computer Vision and Pattern Recognition. arxiv.org/abs/1604.07316
2. [2] Image references:
   1. /examples/training-image.jpg
   2. /examples/cropped-image.jpg
   3. /examples/resized-image.jpg
   4. /examples/flipped-image.jpg