Behavioral Cloning for Steering Control

Author: Aneeq Mahmood

Email: Aneeq.sdc@gmail.com

The goal of this work has been to build a convolutional neural network (CNN) in Keras which can detect driving lanes accurately by predicting the correct steering of the car. This has been achieved by using a data set which has been divided into training and validation set, and then the CNN is used to drive around a track using the simulator without leaving the road.

The individual rubric points are addressed below

Files Submitted:

Following files are submitted for this work:

drive.py : File used to use the simulator in autonomous mode

video.py: file used to make video from still images

imgaug/: A library for augmenting images from https://github.com/aleju/imgaug

images_aug.py: file containing all the helper functions

model_generator.py: main file to run the CNN

model_X.zip: zipped folder containing model_X.h5 file, which Is used to run the simulator in autonomous mode

data/driving_log.csv: a reference csv file containing addresses of images

This write-up is also part of submitted work.

Quality of Code:

The model_generator.py script is used to train and validate the CNN and is functional. The images_aug.py script contains all the helper functions and a python generator to perform data augmentation on the fly. It should be noted that this script defines the path to images and the csv file in line 19 and 20 as

current_dir = "data/IMG/"

csv_file = "data/driving_log.csv"

Hence, a directory “data” should be present in the same directory of the scripts containing the data images and the csv file. Otherwise, the values of curret_dir and csv_file should be changed to point to the right resources. If this is not done, the code will exit with an error.

The model_generator.py script will create a model_X.h5, which can be used with drive.py script (unchanged from what has been provided by Udacity) and the simulator using the following command:

Python drive.py model_X.h5

If the model_X.h5 file already exists in the directory, the code will use that existing model and weights on the training and validation dataset. The code has been commented throughout to ease understanding and clarity.

Model Architecture and Training Strategy:

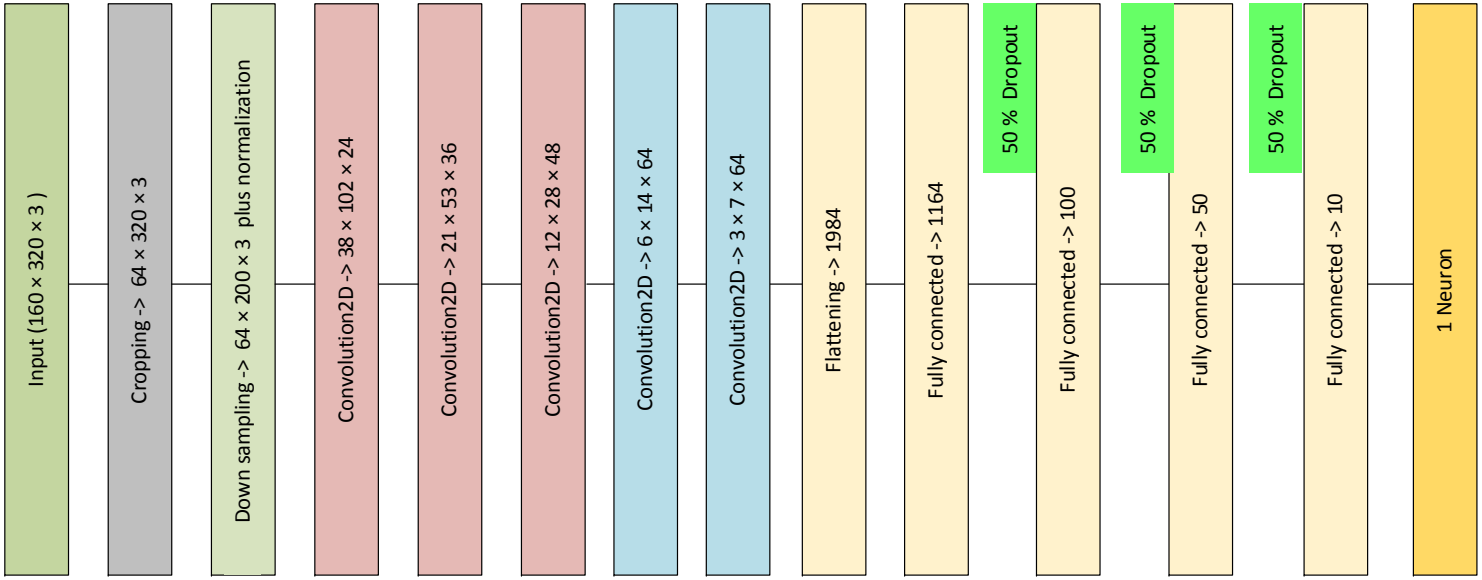
As it is a long and arduous process to find the write CNN architecture, the focus has been on using an existing CNN architecture. The LeNet architecture has been used in the previous project, which could have been a good starting point for this work, however, a big limitation of that architecture is its small image input size, which is 32 * 32. On the other hand, the smallest possible input image size from the Udacity simulator is 160*320. Down-sampling the images from the simulator to suit the LeNet architecture would have led to loss of a information. Thus, this work has primarily used the CNN architecture from [1], where the input image size has been 66*200.

As shown in the figure above (which is an image from the Udacity simulator), information at the top and bottom of the image is not helpful in detecting lanes for steering the car. Hence, if the above image is cropped from top and bottom, the resulting image will have height similar to what is required by [1]. Hence, the cropped images from the simulator will be suitable for training using the model from [1]. Moreover, this CNN model has several layers and employs filters of varying depths, which has come as a result of extensive investigations as discussed in [1]. Hence, this architecture can be used for this work as well.

The architecture used in this work is shown in the figure below. The input image is first cropped by 74 lines of pixels from top, and 20 lines from bottom, to focus the learning on the road .Afterwards the image is reduced to 66*200 and is normalized. A few other modifications has been made to the initial architecture of [1] mainly due to the lack of details. For example, whereas kernel size and strides are described in [1] for the different convolutional layers, other details such as zero-padding or type of activations are not discussed. In this work, zero-padding of 3*3 is used with Relu activations in the first three convolutional layers, and no-zero padding is used in the last two convolutional layers (with Relu) before flattening. The reason for using padding is that without it, information at the edges will be ‘dissolved’ in the first few layers. However, edges contain extremely important information such as lane marking, ledges, or dirt roads. Thus to make this information last several layers, zero padding is done in the first three convolutional layers. It should be noted that in the figure below, this zero padding is shown within the same layer. However, in the model_generator.py script, this is done in a separate layer to breakdown individual steps.

To reduce overfitting, dropout of 50% is introduced after the fully connected layers to reduce the chances of overfitting. To further limit overfitting and increase size of dataset, image augmentation is employed whose details are discussed later. To avoid tempering with learning rate, ‘Adam’ optimizer

is used. This choice has been made to reduce the hyperparameters as learning rate for the Adam optimizer can be left untouched. Also, 20% of the dataset is left aside for validation while the rest is used to train the mode.

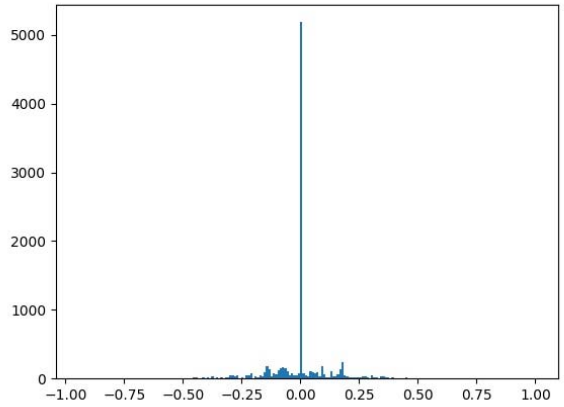


Convolution layer — Kernel = 5 × 5 , padding = 3 × 3, stride = 2, RELU activation

Convolution layer — Kernel = 3 × 3 , padding = 0, stride = 1, RELU activation

Training and Validation Documentation

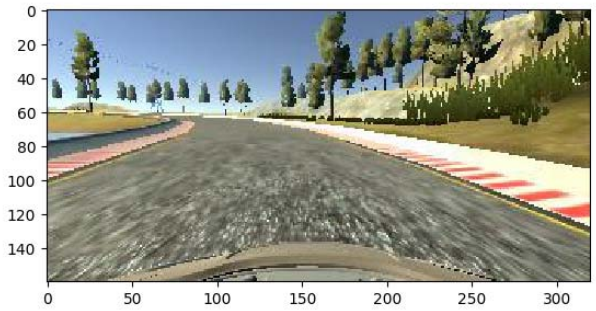
Whereas choice of the mode and model architecture are described above in detail, information about training the network is not yet described. This section deals with this issue. To train this network, an initial dataset has been provided. It has turned out that this dataset was sufficient to train the network. However, this dataset has too many steering values as zeros corresponding to driving straight. This is evident by the following histogram.

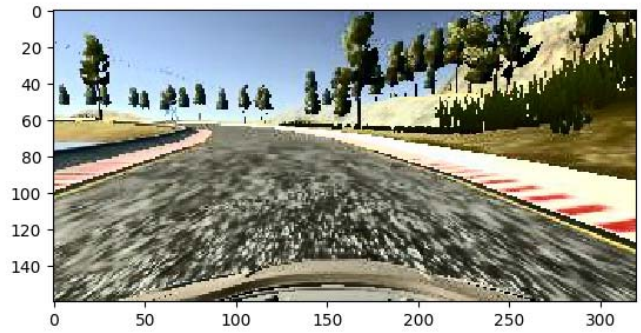


As a first step to add additional images, the simulator is run on those sections where there exist sharp turns. These are sections where there is water and dirt road in view inside the track 1 of the simulator. Afterwards, 50% of images corresponding to steering values less than 0.25 are randomly removed from the dataset. To increase the dataset, all images are also flipped and added to the original dataset.

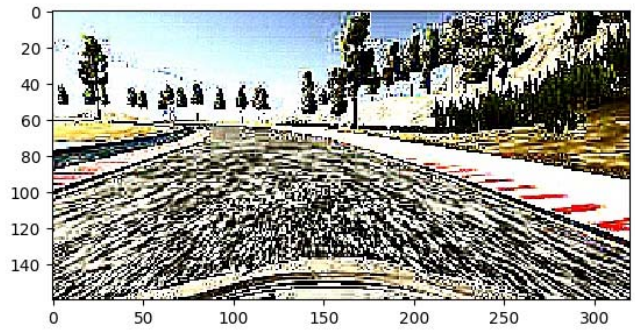
This dataset is randomly shuffled and 20% of it is set aside as data for validation. Overall, there are around 10372 images out of which 2074 images are for validation and the rest are for training. To train the model, training images are provided from a generator. Inside the generator, for a particular batch size, images are retrieved from the directory data/IMG. For a particular image, it is decided randomly to use either one of the center, left or right camera view. As the default camera view is center, hence for left or right camera view, the steering values are compensated with a value of 0.238. This value has been obtained through hit and trial and is a result of experimentation.

The image is then passed through “histogram equalization” and then through augmentation techniques. This latter augmentation is done using the library from [2]. At first, Gaussian blur and random brightness is added to an image. Afterwards the images are randomly sharpened and then contrast-normalized; the resulting images are collected as output of the generator. Following images show the original, histogram-equalized, and the final image for training, respectively.





,



This image is then cropped to remove the top 74 pixel lines and bottom 20 lines, and then resized to fit the model. The training is done for 5 epochs. The training is done over 20000 images and takes several hours. No GPU support has been present to do this training as the AWS cloud support was not available due to NVIDIA driver problems. After failing to set the environment via the AWs, the whole training and validation was done on a normal desktop computer.

Simulation

The results of the simulation for one lap are contained inside the run2.mp4 file and also the model_X.h5 can be used to see the performance of the model by opening the simulator in the autonomous model. The car remains on the road all the time. It does get very close to the road marking at the first major turn, but the wheels do not touch the lane marking. Afterwards, the car drives almost in the middle of the road and does not go off the road, hence fulfilling the project requirement.

When the same model was used on the second 'Jungle' track, the model performed poorly and the car could not tackle with sharp turns on that track. The reason can be that model is trained on the first track and a large majority of the training is done to drive in straight lines, and not enough sharp turns are present in the data set.

References

[1]https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf.

[2] https://github.com/aleju/imgaug