PROJECT 3: BEHAVIORAL CLONING

Files Submitted for Review includes:
model.py
drive.py (no changes)
model.h5
writeup report
video.mp4

Behavioural cloning is one of the most challenging and satisfying projects so far in the Self Driving Car Nanodegree program. The project demands to successfully make the car in the simulator run in autonomous mode using the model designed by the student. The simulator has 2 modes, one with controls and other being autonomous. The control mode can be used for making a dataset by driving the car in the simulator. The car is fitted with 3 cameras, centre, left and right. The inputs we are interested are the 3 camera outputs and the steering wheel angle measurements.
For this project I have used the provided sample driving data (https://d17h27t6h515a5.cloudfront.net/topher/2016/December/584f6edd_data/data.zip) provided by Udacity. The sample data contains driving behavioural information of car while driving in both clockwise anti clockwise direction. The data is taken for 4 loops and contains 8,036 data points. Driving the car in both directions generalises the driving behavioural data. A sample image from the dataset is given in figure 1.

Quality of Code

- The model provided was successfully operated the car in simulator in autonomous mode without collision
- The model.py uses python generator with batch size of 28 to generate data for training, hence saves memory. Also, comments are included in the code of model.py.

Figure 1: Sample front camera image

Reading the data:

The data was read from the driving_log.csv file provided with the dataset. Pandas is used to read the csv with its header data. We read only the first 4 columns, which contains information, centre, left and right camera image locations and steering angles respectively. The first few rows of the file is shown in table 1.

Table 1: driving_log.csv sample data

	center	left	steering	throttle	reverse	speed
center	left	right	steering	throttle	brake	speed
IMG/center_2016_12_01_13_30_48_287.jpg	IMG/left_2016_12_01_13_30_48_287.jpg	IMG/right_2016_12_01_13_30_48_287.jpg	0	0	0	22.14829
IMG/center_2016_12_01_13_30_48_404.jpg	IMG/left_2016_12_01_13_30_48_404.jpg	IMG/right_2016_12_01_13_30_48_404.jpg	0	0	0	21.87963
IMG/center_2016_12_01_13_31_12_937.jpg	IMG/left_2016_12_01_13_31_12_937.jpg	IMG/right_2016_12_01_13_31_12_937.jpg	0	0	0	1.453011
IMG/center_2016_12_01_13_31_13_037.jpg	IMG/left_2016_12_01_13_31_13_037.jpg	IMG/right_2016_12_01_13_31_13_037.jpg	0	0	0	1.438419

The camera images are the inputs and the steering angles are the corresponding desired outputs fed to the CNN network to create an autonomous driving model. But when we analyse the steering angle closely we can understand that the steering angles data consists of mostly zero angle. This makes the dataset highly imbalanced and will cause the model to predict zeros more frequently, which is undesirable. The histogram showing distribution of steering angles in data set can be seen in figure 2. Method used to reduce the effect of this drawback will be discussed in the coming sections.

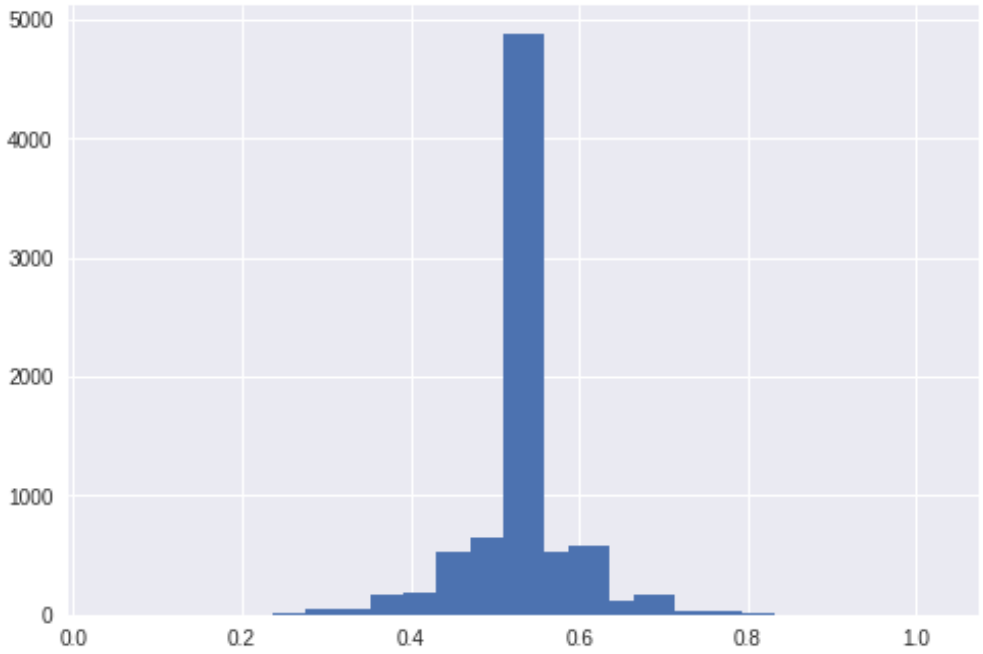

Figure 2: Histogram of steering angles

Reading Images and Pre-processing

Images are read by iterating through each line of csv file and making use of the image location stored under image heads. Apart from using only the centre image the left and right camera images are used, along with their steering angle measurements. Random

corrections are made to the measurement [1] to make the dataset more balanced. Reading of image is also random thus shuffling the images effectively. Moreover, roughly half of the images are horizontally flipped to generalise the dataset further which will help to reduce overfitting. Example of a horizontally flipped image is shown in figure 3.

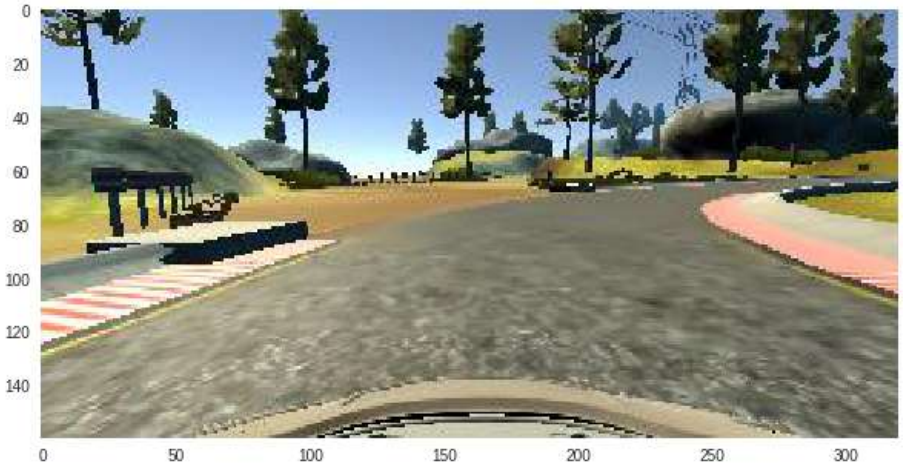

Figure 3: Horizontally flipped image

Model Architecture and Training

The model discussed here is based on the model by comma.ai's research[2]. I have implemented the NVIDIA [3] model also, but the modified comma.ai model seems to reduce the cost within very less number of epochs. The actual model has 3 convolutional layers, 2 dense layers and a total of 3,345,009 parameters. The layers are shown in the table 1.

Table 2: Model by comma.AI

Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 160, 320, 3)	0
cropping2d_1 (Cropping2D)	(None, 65, 320, 3)	0
conv2d_1 (Conv2D)	(None, 17, 80, 16)	3088
elu_1 (ELU)	(None, 17, 80, 16)	0
conv2d_2 (Conv2D)	(None, 9, 40, 32)	12832
elu_2 (ELU)	(None, 9, 40, 32)	0
conv2d_3 (Conv2D)	(None, 5, 20, 64)	51264
flatten_1 (Flatten)	(None, 6400)	0
dropout_1 (Dropout)	(None, 6400)	0
elu_3 (ELU)	(None, 6400)	0
dense_1 (Dense)	(None, 512)	3277312
dropout_2 (Dropout)	(None, 512)	0
elu_4 (ELU)	(None, 512)	0
dense_2 (Dense)	(None, 1)	513
Total params: 3,345,009		
Trainable params: 3,345,009		
Non-trainable params: 0		

The modified model consists of 5 convolutional layers and 2 dense layers. The convolutional layers reduced the number of parameters to 496,865. The layer architecture and details such as filter size, number of filter, etc can be seen in the table 2.

Table 3: Modified implemented model

Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 160, 320, 3)	0
cropping2d_1 (Cropping2D)	(None, 65, 320, 3)	0
conv2d_1 (Conv2D)	(None, 17, 80, 16)	3088
elu_1 (ELU)	(None, 17, 80, 16)	0
conv2d_2 (Conv2D)	(None, 9, 40, 32)	12832
elu_2 (ELU)	(None, 9, 40, 32)	0
conv2d_3 (Conv2D)	(None, 5, 20, 48)	38448
elu_3 (ELU)	(None, 5, 20, 48)	0
conv2d_4 (Conv2D)	(None, 3, 10, 64)	76864
elu_4 (ELU)	(None, 3, 10, 64)	0
conv2d_5 (Conv2D)	(None, 2, 5, 64)	36928
flatten_1 (Flatten)	(None, 640)	0
dropout_1 (Dropout)	(None, 640)	0
elu_5 (ELU)	(None, 640)	0
dense_1 (Dense)	(None, 512)	328192
dropout_2 (Dropout)	(None, 512)	0
elu_6 (ELU)	(None, 512)	0
dense_2 (Dense)	(None, 1)	513

Total params: 496,865
Trainable params: 496,865
Non-trainable params: 0

Function for normalizing the image is given as input to the lambda layer of keras, before supplying it to the cropping layer, were the image is cropped horizontally 70 and 25 pixels from top and bottom. The cropped image is then fed into 4 convolution layers having same filter size 5x5 and strides 2x2 and varying filter sizes of 16, 32, 48 and 64 respectively. The last convolutional layer has 64 filters of 3x3 size whose output is flattened. Dropouts of 0.2 and 0.5 are added after flatten layer and first dense layer respectively. Activation function used is ELU, the loss function is MSE and optimizer is ADAM.

I used mode.fit_generator with a batch size of 28 for training the network, so that all the data need not be saved in the workspace at a time. The generator function is used for reading and pre-processing the images in batches when called by the **model.fit_generator**. The model displayed satisfactory performance to create a successful mode to run the car in simulated mode.

Conclusion

The model discussed in the previous section was saved as model.h5 for running the car in simulator**. python drive.py model.h5** command was used in the terminal to run the model and was successfully connected with the car in simulator. The car was able to navigate in autonomous mode without making any erroneous decision. Later python drive.py model.h5 run1 command was used to save the continuous images of autonomous navigation of car in the simulator. **python video.py run1** command was used to make a video from the images at 60 frames per second. The video and the images are saved in the workspace for evaluation.

References

1.End-to-end learning for self-driving cars. https://navoshta.com/end-to-end-deep-learning/

2. Learning a Driving Simulator. https://arxiv.org/pdf/1608.01230.pdf

3. End to End Learning for Self-Driving Cars. https://arxiv.org/pdf/1604.07316v1.pdf