End to End learning by predicting steering angle values for autonomous vehicles

Amey Kulkarni[1]

***Abstract*— It is imperative for a self-driving car to know how much should it turn by processing the sensor information. This report considers only camera as the sensor and predicts the steering angle of the vehicle based on the input from the camera. The camera images are processed and fed to the Convolution neural network along with the steering angle labels. The Udacity dataset [1] is used for this purpose. The left, right and center camera data is extracted along with the steering angle values. These images are taken at 20fps and the steering angles are generated at the sampling rate of 50. The timestamps of the steering angles is matched with its nearest image timestamp and this correlation is used for training. The training is done using custom made Convolutional Neural Network.**

I. INTRODUCTION

Steering angle prediction is a critical task in the self-driving vehicle industry. Accuracy of this task affects the safety and performance of the vehicle. Deep learning techniques have become the most favourable choice for performing this activity. For getting great accuracy in the prediction, we need to have a great dataset and intricate processing on the dataset is mandatory. The Udacity steering angle dataset[1] is used for training in this paper. This dataset is obtained in the form of a rosbag file and it contains about 15k images and 38k steering angle values. Accurate correlation between these two messages are established and used for training.

CNNs are popular neural network architectures which are used in this paper to train the data. Different architectures of CNNs are possible. The architecture developed by NVIDIA gives fabulous accuracy and hence it will be a worthwhile experience to study it and implement it. The details of this network are explained in [5] and as per this suggested network, currently the center camera images are trained with the steering angle values.

II. METHODOLOGY

A. Feature Engineering

The images cannot be directly used for training in their raw form. The best part about the developed network is that not a lot of preprocessing of images is required. Unlike the previous project done in this course which was to predict the steering angle values using MLP,this specific network requires minimal preprocessing. In this case, the image is taken as RGB and the top half of the image is cropped. The remaining images is scaled down to 66*200 size as the network works on this inputs. The images and steering angle values are ormalized before training and the reason for this will be explained in the results section.

Since we are using CNNs, we need not make the image into a row vector. The images is stored as a numpy array. [3] gave intense help to process the images using the opensource framework OpenCV.
Fig.1 shows the input raw RGB image. Fig.2 shows the second step which is the cropped image and Fig.3 shows the resized image used for training.

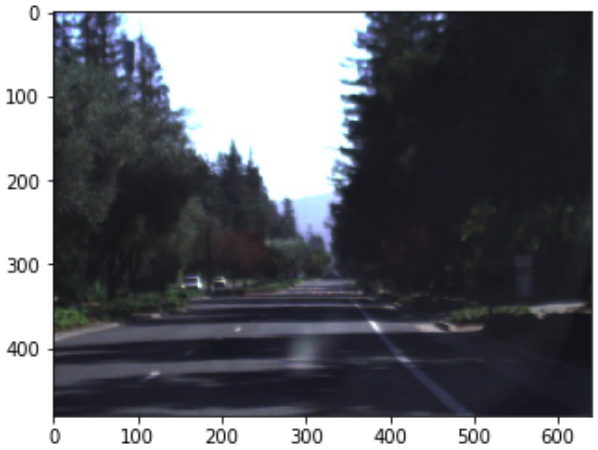

Fig. 1. Input image- 1st step

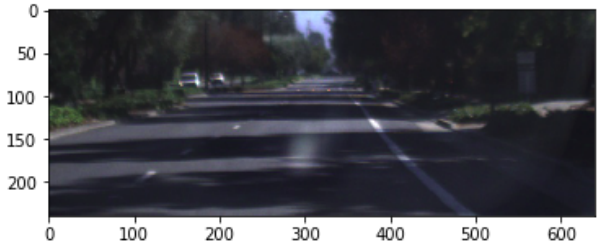

Fig. 2. cropped image- 2nd step

It is also observed that there are a lot of steering values which are near 0. About 30k values out of the 38k values are near 0. This can be detrimental to the training. Hence a histogram is made to check the number of 0 values and then some zero values are removed randomly. The two histograms are shown below.

[1]A. Kulkarni, is a graduate students in the Robotics Program of Worcester Polytechnic Institute, Worcester, MA 77005 USA.

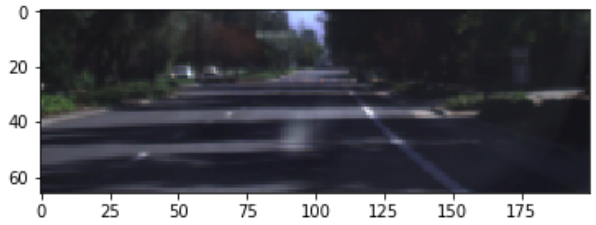

Fig. 3. Resized image-3rd step

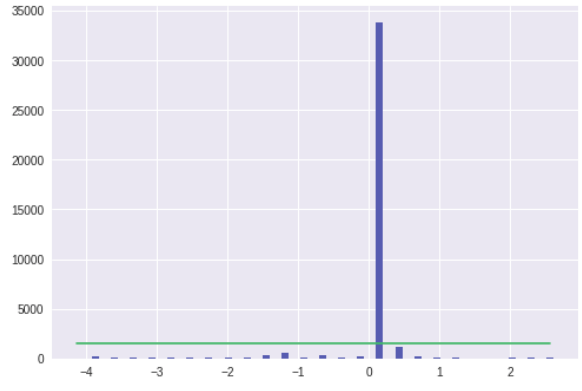

Fig. 4. Histogram of raw data

III. MODEL ARCHITECTURE

The popular NVIDIA model is used for this regression analysis. The model was developed by NVIDIA for this specific purpose of predicting the steering angle values by taking visual inputs from the front camera. The model is a Convolutional neural network with 5 convolution layers and 3 fully connected layers are added to it to facilitate regression. Another layer at the start is the BatchNormalization layer which basically normalizes the incoming data to make it more regular or easy for training. Results show that using the batch normalization layer gives more independence to each layer for training. Batch normalization reduces the amount by which the hidden unit values shift. This improves the training speed. The layer itself is untrainable.

The inputs to the network is a 66(row)*200(col) RGB image. Fig.6 shows this architecture.Fig.7. shows the layers and the output dimensions as obtained from the program used for completing this assignment for the AI for AV course at WPI.

IV. RESULTS

The training data and the testing data are separated in the 80:20 ratio. The training data was further randomly separated into the training data and the validation data in 90:10 ratio. The training data had 4435 observations, the validation data had 493 observations and the test data had 1232 observations in all. The test data was kept completely away while training. While training, the model was tried on the training dataset. The training model hyper-parameters were modified to give better results in the validation data. The final model was then tried on the test data and the

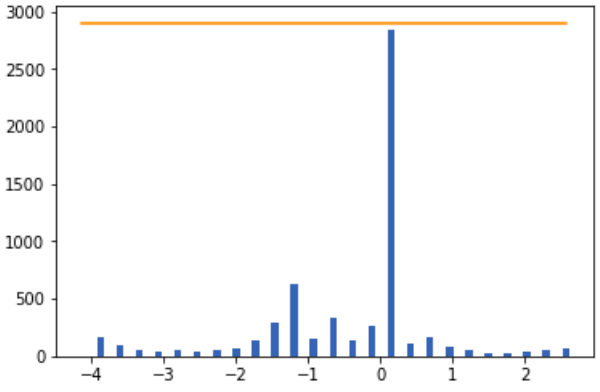

Fig. 5. Histogram of processed data

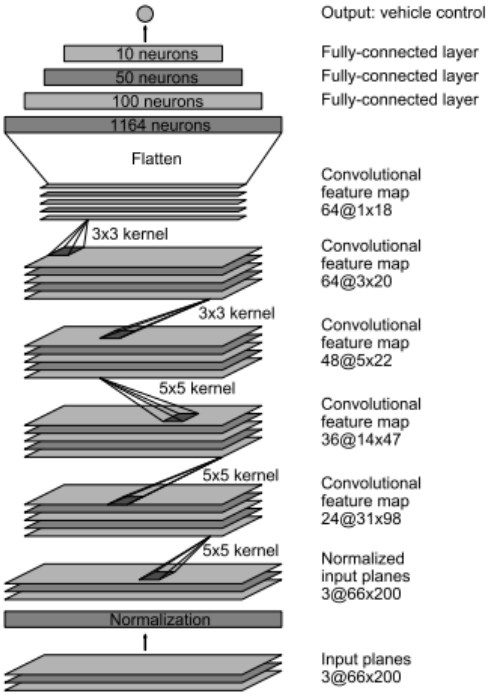

Fig. 6. NVIDIA model architecture

results are reported. All the layers are activated with relu activation function except the last layer which is activated using the tanh function.The tanh function gives a scaled output only in the range -1 to 1. Hence, this implies that all the inputs should also be scaled within this range for the training to perform correctly. The ADAM optimizer with mean square error as the loss criteria is used. The results of some of the tried combinations of learning rates trained for 30 epochs are shown below.

Learning rate	Training loss	RMSE
0.01	0.058	2.18
0.001	0.057	4.1
0.0001	8e-4	0.4

Layer (type)	Output Shape	Param #
batch_normalization_15 (Batc	(None, 66, 200, 3)	12
conv2d_67 (Conv2D)	(None, 31, 98, 24)	1824
conv2d_68 (Conv2D)	(None, 14, 47, 36)	21636
conv2d_69 (Conv2D)	(None, 5, 22, 48)	43248
conv2d_70 (Conv2D)	(None, 3, 20, 64)	27712
conv2d_71 (Conv2D)	(None, 1, 18, 64)	36928
flatten_14 (Flatten)	(None, 1152)	0
dense_66 (Dense)	(None, 1164)	1342092
dense_67 (Dense)	(None, 100)	116500
dense_68 (Dense)	(None, 50)	5050
dense_69 (Dense)	(None, 10)	510
dense_70 (Dense)	(None, 1)	11

Total params: 1,595,523
Trainable params: 1,595,511
Non-trainable params: 12

Fig. 7. NVIDIA model layers and output dimensions with trainable paramters mentioned

The results show that the predictions at the data extremities can be improved. The reason for lower performance is that the data for these specific training labels is very less. Most of the data for steering angles very close to 0 degrees. The dataset is tried to improve in this assignment by removing some unnecessary values but it can be further improved by data augmentation and actually building a new diverse dataset.A video of the network giving real time inference results is added to the submission folder. The google colab folders used for this project are also submitted with this report. The google colab files and the dataset can be found at the following link- https://drive.google.com/open?id=1IUK3H5O7VNg2QGxeg6vBtlPGMde9SGkv The best prediction obtained was using a learning rate of 0.0001 which made the training loss to go to 8e-4 and the validation loss went down to 0.02. The regression chart for this specific training session is shown in Fig.8.

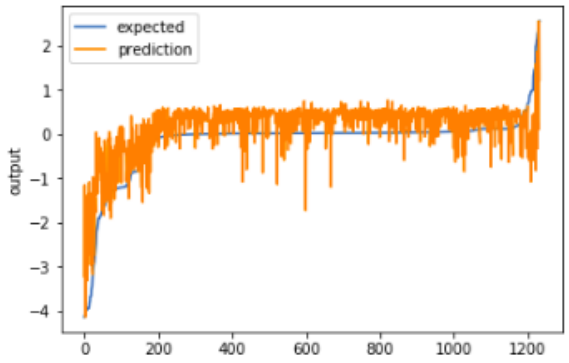

Fig. 8. Regression chart for learning rate of 0.0001

Just for the sake of comparison and to show the greatness of the NVIDIA architecture or power of CNNs, the results of the previously trained MLP framework and the current CNN framework is shown in Table 1.

Model Architecture	RMSE	epochs
MLP(500,250,125,64,32,16)(%)	0.683	33
NVIDIA CNN(5conv, 3FC)(%)	0.4	30

It is concluded that the CNN architecture is better than the MLP and can easily be improved by improving the dataset and performing more epochs.

V. Future Work

There is always a room for improvement and the results show that we can still improve the predictions from our models. Changes can be incorporated in variety of ways. We can combine other sensor data to get better predictions. All the three camera inputs can be used for training. Apart from the data, other model structures can be tried with different framework architectures.Also , this architecture can be improved by adding dropout and maxpooling layers.

VI. CONCLUSIONS

The Udacity dataset was used to input images to the model and predict the steering angle of the vehicle. since the sampling rate for both the image extraction and the steering angle is different, the correlation between data and labels is made using the comparison between their time stamps. Fully connected layers were attached to the CNN to achieve the regression results. Most of the times, CNN was used for classification and this project gave a great insight into how to use CNNs for regression.

References

[1]https://github.com/udacity/self-driving-car/tree/master/datasets
[2] http://deeplearning.net/tutorial/mlp.html
[3]https://docs.opencv.org/3.4/
[4]Petar Penkov1, Vinay Sriram, and James Ye, ”Applying Techniques in Supervised Deep Learning to Steering Angle Prediction in Autonomous Vehicles”
[5]Mariusz Bojarski et.al, ”End to End Learning for Self-Driving Cars”, arXiv:1604.07316v1 [cs.CV] 25 Apr 2016