Building an end-to-end Deep Learning Model utilizing Convolutional Neural Networks for self driving cars to test in Udacity's simulation environment

Lakshmikar Reddy Polamreddy Yeshiva University
lakshmikarpolamreddy@gmail.com

Abstract

An attempt has been made to build an end-to-end deep learning neural network model that will allow the car to drive on its own in a simulating environment created by Udacity. This platform has been used to generate training data required for model training. Instead of using NVIDIA model architecture, we have developed our own CNN model and trained it. After the training, this CNN model was used to drive the car in the simulator. We observed that our model has shown decent performance but it did not perform better than that of NVIDIA model. As part of future investigation work, we will refine the model in terms of design and will apply data augmentation techniques for better performance. In addition to this, we would like to explore the option of using various pre-trained models to compare the performance.

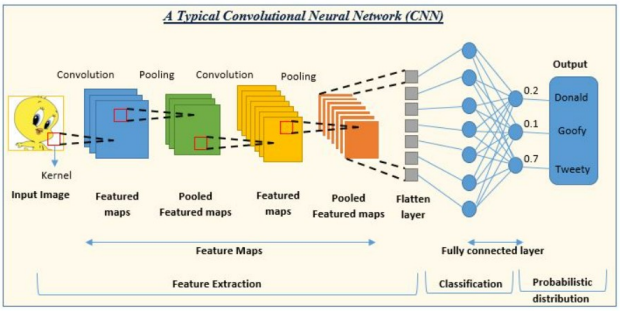
1. Introduction

WHO(World Health Organization) report [5] says that approximately 1.3 million people die each year as a result of road traffic crashes. Most of these accidents do occur because of mistakes committed by humans due to several factors like recklessness, drunken driving and biological limitations of humans. One of the best ways to overcome this dangerous situation is by implementing self driving technologies in the vehicles.

We can exploit various machine learning algorithms to drive a vehicle without human intervention. Using these algorithms, end-end machine learning models can be built and trained on huge sets of available data in the form of images generated by sensors and cameras. Then, these trained models will drive the vehicles in such a way to minimise accidents. As we have not yet collected the real time data, we have made use of simulation environment developed by Udacity for its nano degree program on self driving cars. We have utilized this platform to generate training data and to test the performance of our model.

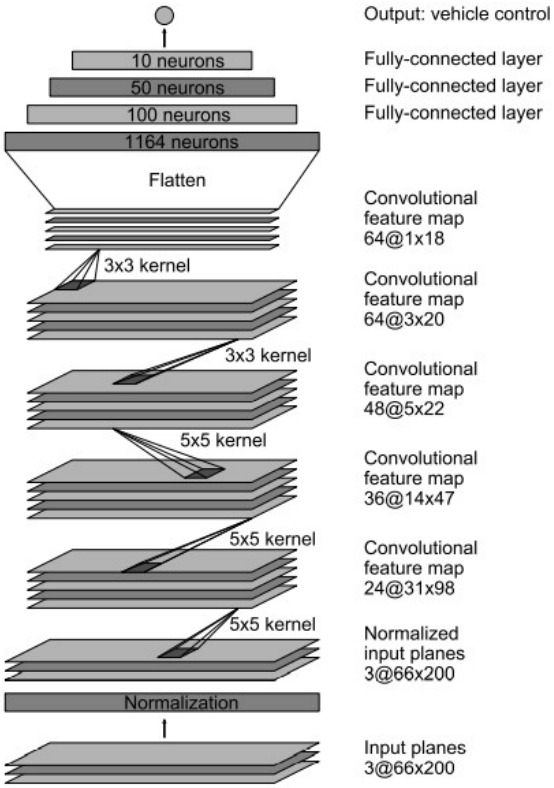
End to End learning in the context of AI and ML is a technique where the model learns all the steps between the initial input phase and the final output result. This is a deep learning process where all of the different parts are simultaneously trained instead of sequentially. An example of end-to-end learning in the context of ML is self-driving cars. Their systems are trained to automatically learn and process information using a convolutional neural network (CNN). In this case, systems use previously provided human input as guidance to complete tasks.

A Convolutional Neural Network [4] is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other. The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area. A typical CNN looks like as shown below.

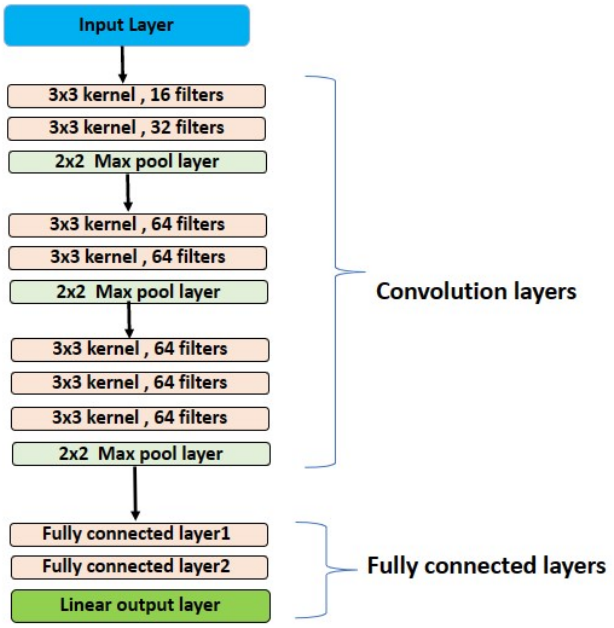


1.1. Model Architecture

Below is the model architecture developed by NVIDIA research team for an end-to-end learning of self driving cars



Below is the model architecture that we have used in this project. It consists of 7 convolution layers, 3 pooling layers, 2 fully connected layers and 1 output layer.



The table below shows the comparison of various model parameters between NVIDIA architecture and our model used for this project

Parameter	NVIDIA Model	Our Model
Number of Convolution layers	5	7
Number of Polling layers	0	3
Number of fully connected layers	4	3
Convolution Filter size	5x5 for the first 3 layers and 3x3x for the next 2 layers	3x3 for all the layers
Activation function	ELU	RELU

2. Related Work

We have gained decent knowledge in Image processing, Image classification, Object detection and Image Segmentation from the lectures offered by Professor Youshan Zhang [6] at Yeshiva University as part of our MS Program in AI. My earlier projects on classification of cow teats images, Object detection of cowstall numbers under his supervision have given good experience to work on this self driving project. Instead of using existing models, I learnt the art of building my own CNN models from him.

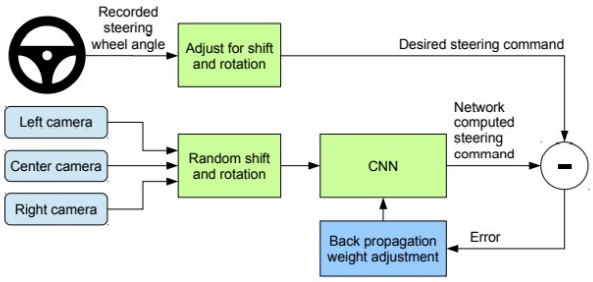
Regarding set-up of the Udacity simulator, an article by Mr. Naoki [3] has provided elaborate information on the prerequisites. He has shown that we need these three environments for this - Unity Game Engine, Git LFS and Udacity Self-Driving Car Simulator Github. Unity is a game development environment in which Udacity self-driving car simulator is built.

Unity Editor

He has very well explained the installation process of the simulator.

We were inspired by the CNN model developed by NVIDIA research team [1] for end-to-end deep learning for self driving cars. In a new automotive application, they have used convolutional neural networks (CNNs) to map the raw pixels from a front-facing camera to the steering commands for a self-driving car. This power-

ful end-to-end approach means that with minimum training data from humans, the system learns to steer, with or without lane markings, on both local roads and highways. The system can also operate in areas with unclear visual guidance such as parking lots or unpaved roads. The block diagram of their training system is shown below.



We have also referred to the approach followed by Zhenye-Na [7] for training the model for self driving vehicles using Udacity's simulation environment to generate training data and to test the model. In this case also, NVIDIA architecture has been used to train the model.

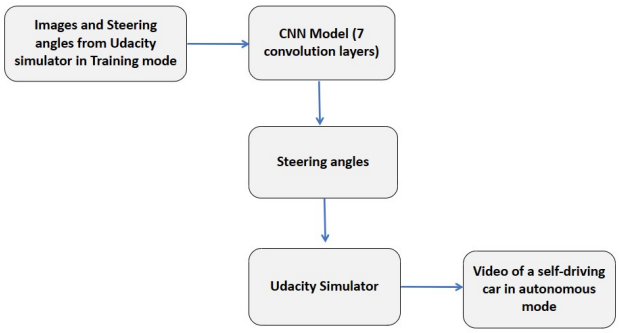
We have also referred to this paper on Reasearch Gate - "Self-Driving Car Steering Angle Prediction Based On Deep Neural Network An Example Of CarND Udacity Simulator" by a team from Orel State University, Russia [2]. They have explored the possibility of using obtained images from the emulator for training deep neural networks for prediction of steering angle and explored various architectures of convolutional neural networks in order to obtain good results with a minimum number of parameters. They have used below model architecture for their project and they have presented good results.

Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 66, 200, 3)	0
conv2d_1 (Conv2D)	(None, 31, 98, 24)	1824
conv2d_2 (Conv2D)	(None, 14, 47, 36)	21636
conv2d_3 (Conv2D)	(None, 5, 22, 48)	43248
conv2d_4 (Conv2D)	(None, 3, 20, 64)	27712
conv2d_5 (Conv2D)	(None, 1, 18, 64)	36928
dropout_1 (Dropout)	(None, 1, 18, 64)	0
flatten_1 (Flatten)	(None, 1152)	0
dense_1 (Dense)	(None, 100)	115300
dense_2 (Dense)	(None, 50)	5050
dense_3 (Dense)	(None, 10)	510
dense_4 (Dense)	(None, 1)	11

Total params: 252,219
Trainable params: 252,219
Non-trainable params: 0

3. Methods

The block diagram shown below highlights the overall approach and major steps of this project work.



Firstly, the images and corresponding steering angles collected from the simulator in training mode is passed onto the CNN model using data loaders of pytorch library. Secondly, the CNN model is trained for sufficient number of epochs till the convergence in loss is achieved. Thirdly, the outputs from the CNN model i.e. steering angles are passed onto the simulator in autonomous mode. Immediately, the car drives on its own on the selected track.

Now, we would like to briefly explain here about how to transfer outputs from the model to the simulator. Save the weights from the trained model as model.h5 file. Then, save model.py file and drive.py file in the same folder and execute this command in the python terminal - python drive.py model.h5 runs1/, where model.h5 is the weights file, drive.py is the python file that connects the weights to the simulator and passes the steering angles to it, model.py is the python file containing the our CNN model, runs1 is the folder name where the images from the front camera will be saved during self driving of the car

As the model is expected to predict the steering angle, we have considered MSE loss and this Regression loss is calculated like below.

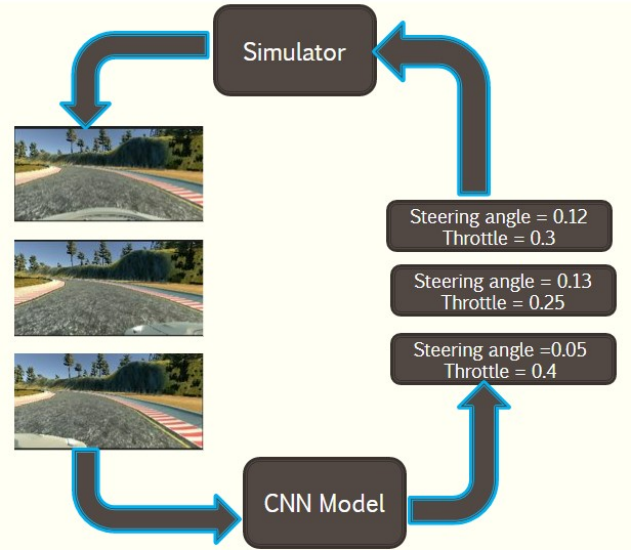
$$\text{Regression Loss} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2 \tag{1}$$

where y_i is the actual steering angle and $\hat{y_i}$ is the predicted steering angle

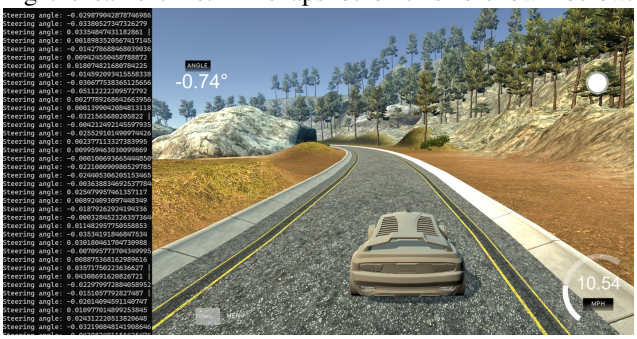
4. Results

The block diagram shown below highlights the testing approach of the model to drive the car in autonomous mode in the simulation environment. The training images obtained from the Udacity simulator are fed into the CNN model to train it. After training, the model will predict steering angle and the throttle

and steering values are passed onto the simulator in autonomous mode. Then the car drives on int own.



After running this script on the terminal - python drive.py model.h5 runs1/, we will notice the steering angles on the terminal and also car moving in the simulator window during the same time. A snapshot of this is shown below.



After training my model for 50 epochs, the training loss has reduced from 0.98 to 0.26 and the validation loss has reduced from 0.62 to 0.30. But, the loss seems to have not converged in 50 epochs. Due to more number of training images, we could not run for more epochs in limited time. In future, we will train this model for more epochs till the loss gets converged.

Below code snippet from python notebook shows the actual steering angles and predicted steering angles by our model for 25 test images taken at random. In addition to this, we have also calculated mean squared error for these test images and is also displayed below.

	Actual Angles	Predicted Angles
0	0.40	0.301617
1	-0.40	-0.245365
2	0.40	0.240596
3	-0.65	-0.482386
4	-0.40	0.075575
5	-0.40	-0.013554
6	0.40	0.323536
7	-0.40	-0.038933
8	0.40	0.325402
9	-0.40	-0.136571
10	0.35	0.262535
11	-0.40	0.250613
12	-0.40	-0.139748
13	0.40	0.120795
14	-0.40	-0.116133
15	-0.40	-0.365213
16	0.40	0.272383
17	-0.40	-0.466356
18	-0.95	-0.249330
19	0.40	0.275791
20	-0.40	-0.373646
21	0.40	0.222595
22	0.40	0.020054
23	0.40	0.278333
24	-0.40	-0.187514

MSE: 0.0838297

The above results indicate that there is more difference between the actual steering angles and the predicted values and so, model performance has to be improved so that this difference will become negligible.

In the runs1 folder mentioned above, images taken by the front camera during autonomous mode are have been captured. Below samples have been captured at different instants of time.

The first image above has been captured as soon as the car started to drive in autonomous mode. The second and the third images have been captured after some time. If we observe the position of the car in these images, the car which was initially at the center of the track has moved towards the right side with time. This indicates that if car runs for more time with our model, it may go off the track. So, the model has be be trained for more epochs with more images so that it learns better and ensure the car drives in the center of the track throughout the lap.

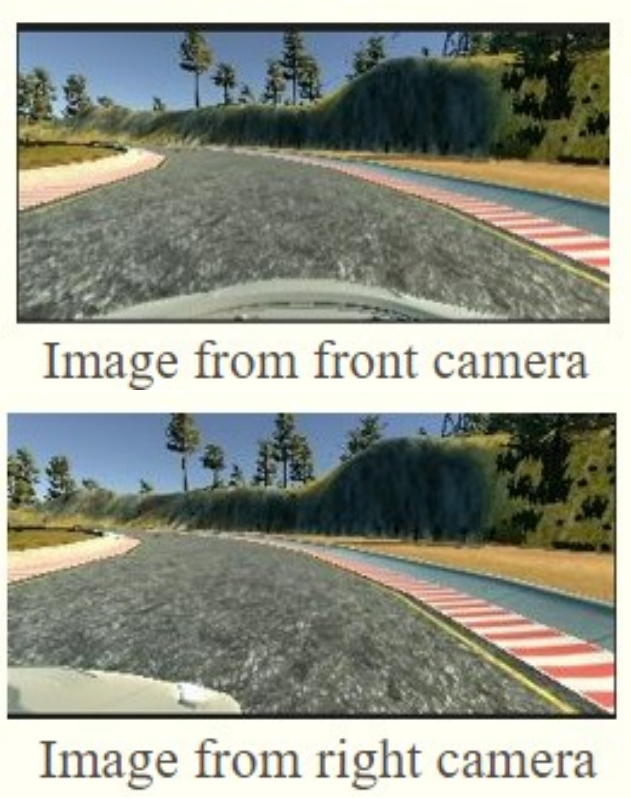
4.1. Datasets

Udacity simulation environment has been used to generate training data. It has 2 different tracks that can be used for training and testing. A sample picture of this environment is shown below.

After selecting either track1 or track2 and then clicking on the training mode, we will see below screen.

As soon as we click the recording button, this creates a folder of images taken from center, right and left cameras at each instant of time. A sample of the images captured at one particular instant of time is shown below.

Image from left camera



In addition to this, it generates datalog.csv file containing 7 columns. First 3 columns contain the path to the images of front, right and left cameras. Column 4 shows the steering angle values - zero corresponds to straight direction, positive value indicates right turn and negative value indicates left turn. Column 5, 6 and 7 indicate acceleration, deceleration and speed of the vehicle respectively.

	A	B	C	D	E	F	G
1	Desktop\tı	Desktop\tı	Desktop\tı	0	0	0	1.06E-05
2	Desktop\tı	Desktop\tı	Desktop\tı	0	0	0	7.36E-06
3	Desktop\tı	Desktop\tı	Desktop\tı	0	0	0	3.47E-06
4	Desktop\tı	Desktop\tı	Desktop\tı	0	0.048016	0	0.002267
5	Desktop\tı	Desktop\tı	Desktop\tı	0	0.281203	0	0.175589
6	Desktop\tı	Desktop\tı	Desktop\tı	0	0.091236	0	0.287728
7	Desktop\tı	Desktop\tı	Desktop\tı	0	0.129511	0	0.38219
8	Desktop\tı	Desktop\tı	Desktop\tı	0	0.321048	0	0.527709
9	Desktop\tı	Desktop\tı	Desktop\tı	0	0.213252	0	0.813831
10	Desktop\tı	Desktop\tı	Desktop\tı	0	0.340305	0	1.020443
11	Desktop\tı	Desktop\tı	Desktop\tı	0	0.580681	0	1.459831
12	Desktop\tı	Desktop\tı	Desktop\tı	0	0.814329	0	2.145293
13	Desktop\tı	Desktop\tı	Desktop\tı	0	1	0	2.820537
14	Desktop\tı	Desktop\tı	Desktop\tı	0	0.848136	0	3.706357
15	Desktop\tı	Desktop\tı	Desktop\tı	0	0.682637	0	4.227759
16	Desktop\tı	Desktop\tı	Desktop\tı	0	0.425183	0	4.708716
17	Desktop\tı	Desktop\tı	Desktop\tı	0	0.299628	0	4.929266
18	Desktop\tı	Desktop\tı	Desktop\tı	0	0.551199	0	5.322563

In this project, we have used the training data available on Kaggle platform [7]. It consists of 97.3K images generated from both the tracks 1 and 2. Out of these, 20 percentage has been used for validation purpose. 70x320x3 is the size the images generated in the simulator.

5. Discussion

Below snapshots code snippet from our python notebook are corresponding to our CNN model written in pytorch library.

```
# Creating model

class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
        # Input image size = 70x320x3

        self.conv1 = nn.Conv2d(3, 16, 3)  # 68x318
        self.conv2 = nn.Conv2d(16, 32, 3) # 66x316
        self.pool1 = nn.MaxPool2d(2, 2) # 33x158

        self.conv3 = nn.Conv2d(32, 64, 3) # 31x156
        self.conv4 = nn.Conv2d(64, 64, 3) # 29x154
        self.pool2 = nn.MaxPool2d(2, 2) # 14x77

        self.conv5 = nn.Conv2d(64, 64, 3) # 12x75
        self.conv6 = nn.Conv2d(64, 64, 3) # 10x73
        self.conv7 = nn.Conv2d(64, 64, 3) # 8x71
        self.pool3 = nn.MaxPool2d(2, 2) # 4x35


        self.fc1 = nn.Linear(4*35*64, 128)
        self.dropout1 = nn.Dropout2d(0.25)
        self.fc2 = nn.Linear(128, 64)
        self.dropout2 = nn.Dropout2d(0.5)
        self.fc3 = nn.Linear(64, 1)
```

```
def forward(self, x):

    x = F.relu(self.conv1(x))
    x = self.pool1(F.relu(self.conv2(x)))
    x = F.relu(self.conv3(x))
    x = self.pool2(F.relu(self.conv4(x)))
    x = F.relu(self.conv5(x))
    x = F.relu(self.conv6(x))
    x = self.pool3(F.relu(self.conv7(x)))

    x = x.view(-1, 4*35*64)
    x = F.relu(self.fc1(x))
    #print(x.shape)
    x = self.dropout1(x)
    x = F.relu(self.fc2(x))
    x = self.dropout2(x)
    x = self.fc3(x)
    return x
```

We have designed model in such a way that in the first two blocks, the 2 convolution layers are followed by a Max pooling layer. In the third block, we have used 3 convolution layers followed by a Max pooling layer. The

fourth block consists of 3 fully connected layers with the last being the output layer.

Max pooling layer has been used to reduce the height and width of the images. The other alternative to this is the usage of average pooling layer. In the fully connected block, we have applied dropout to 2 layers to avoid overfitting. We have applied ReLU activation function for all the layers, except the output layer, to fire neurons. The other suitable alternative to ReLU is ELU activation function. In our case, the output is just one linear value(steering angle)

Regarding loss function, we have used MSE loss function as the output is just one value and the loss is measured by calculating the difference between the actual and the predicted steering angle values. Accordingly, weights will be adjusted during back propagation and this process continues based on the number of epochs.

As part of Data Augmentation process, we have normalized the data to avoid saturation so that gradients will work better for the model training. We have also performed random rotation on the images and cropped the images as well. The snapshot below shows an example of original and cropped images. this has to be done for better performance of the model.

Original image

Cropped image

As part of generating the training data from the simulator, one has to take care, while driving manually, to ensure the car always move in the center of the track. Otherwise, the training data will be erroneous and reduce the model performance during predictions. In addition to this, a large training data is required for better learning of the model in terms of turns on the track and surrounding environment.

6. Conclusion

We started this project first by studying the NVIDIA architecture and then built our own CNN model to predict the steering angles with high accuracy at every instant so that car will will not go off track in Udacity simulation environment. In this project, we have observed our model has shown decent performance to drive the car for shorter times. There is a great scope to further improve the performance by means of better data augmentation techniques. Also, the model has to be trained for more number of epochs to improve learning. Going forward, we would like to build more complex models with many convolution layers and test them for their performance. Not only this but also we would like to use a combination of pre-trained models and test their performance. Along with predicting steering angles, we would like to predict acceleration and speed of the car. The other ambitious activity for future work is to build and train the model with images from real environment.

References

[1] B. F. L. J. U. M. K. Z. Mariusz Bojarski, Ben Firner and D. D. Testa. End-to-end deep learning for self-driving cars. 2016. 2

[2] V. V. I. M.V. Smolyakov, A.I. Frolov. Self-driving car steering angle prediction based on deep neural network an example of carnd udacity simulator. 2018. 3

[3] Naoki. Introduction to udacity self-driving car simulator. 2017. 2

[4] S. Saha. A comprehensive guide to convolutional neural networks — the eli5 way. 2018. 1

[5] WHO. Road traffic injuries. 2022. 1

[6] Y. Zhang. Assistant professor at yeshiva university. 2022. 2

[7] Zhenye-Na. End-to-end learning for self-driving cars — udacity's simulation env. 2019. 3, 6