Deep Learning for Self-Driving Cars

Yash Vardhan Sharma, Sangam Jindal, Zong-Fu Hsieh
Department of Computer Science and Engineering
Texas A&M University

***Abstract*—This paper describes an implementation of "End to End Learning for Self-Driving Cars" and some further experimental results on different hyper-parameters.**

Keywords—autonomy;

I. INTRODUCTION

Visionaries have always talked about a future where technological advancement would make life much more comfortable and safe. Self-Driving car is one such technological marvel that would make commuting easier and safer for everyone on the road. Many technology companies are working to make this a reality. This paper attempts to explain one such research from a technology company.

Nvidia Corporation published a paper titled "End to End Learning for Self-Driving Cars" in 2016. This paper described a self-driving car framework that enabled Nvidia to convert this concept of self-driving into reality. We followed the paper mentioned above to write our own implementation of the CNN model and evaluated the network on Udacity's open source simulator for self-driving cars.

II. LITERATURE REVIEW

Self-Driving cars have gained a lot of attention these days from various technology companies and automobile manufacturers in the world. A lot of research is going on in this area. Some of the latest work include "Motion Planning of Vehicle Obstacle Avoidance in Complex Traffic Scenarios", "Edge Enhanced Traffic Scene Segmentation Algorithm with Deep Neural Network" etc.

In this paper, we have tried to improve the existing research by changing the parameters and introducing new layers in the network.

III. PROBLEM FORMULATION

Algorithms need to analyze a lot of parameters to run a car. To achieve this high scale image analysis, we have convolutional neural networks (CNN). The main idea is to design a CNN that can take camera images from the car and output values for steering angle, throttle, speed and brake.

To measure accuracy of the network we have defined a measure of accuracy for the convolutional neural network. We call this measure "autonomy". It is the amount of time the car can follow the road.

IV. PROBLEM SOLUTION

The paper mentioned above details one such CNN that takes in images, steering angle, throttle, speed and brake. The car is mounted with 3 cameras that takes images on the left, right and front of the car. The other four car parameters steering angle, throttle, speed and brake are recorded by the Nvidia Drive Px hardware as described in the paper. For us, all these car parameters were recorded from the Udacity's simulator for self-driving car.

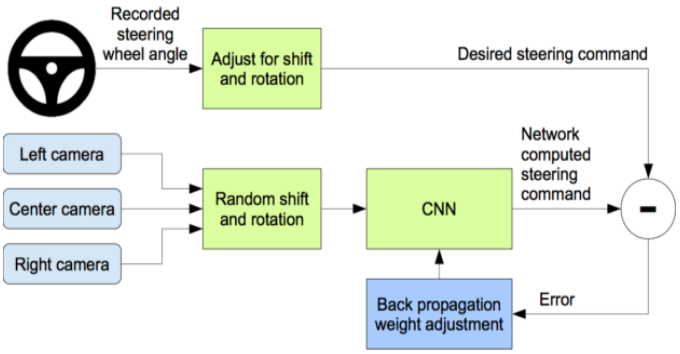

Figure 1 Training the Neural Network

This data was fed to a CNN. The CNN consists of a normalization layer, 5 convolution layers and 3 fully connected layers. The normalization layer performs hard coded normalization that helps to avoid saturation and make gradients work better. The first three convolution layers have a kernel of 5x5 and stride of 2x2 and the other 2 layers have a kernel of 3x3 with no stride. These layers help in feature extraction from the input images. All the convolution layers use Exponential Linear Unit "elu" as activation function. This activation helps to solve the vanishing gradient problem in the network.

After feature extraction from the 5 convolution layers, we have a dropout layer with dropout 0.5. This helps the network to learn more robust features and to avoid over-fitting. Next layer after dropout is flatten. All the input matrices are flattened to form a 1 dimensional array. This is done because the next layers are fully connected.

This is followed by 3 fully connected layers. The purpose of these fully-connected layer to provide inverse of steering angle. These layers function as a controller for the car. The layers have different neurons that keep on decreasing with every layer. The first layer has 100 neurons, second has 50 and the last has 1 neuron that contains the final steering angle value.

The implementation has two files model.py and drive.py. The model.py file contains source code the generates a Keras model that is of type ".h5". The CNN is implemented using Keras library. Firstly, the data is loaded in a data frame using pandas. Here we ignore other car parameters and just take in steering angle. This data is then split into test and train data using train_test_split in sklearn.

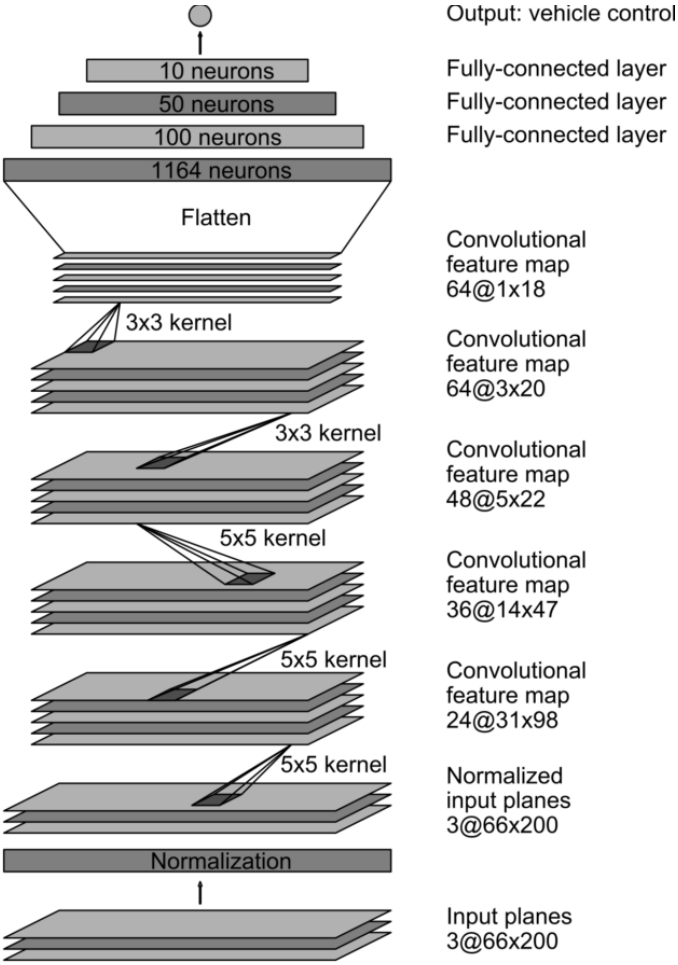

Figure 2 Convolutional Neural Network Architecture

After loading the data, the model is built using Keras functions. Once the model is built, it is compiled. In addition to compiling and fitting the model, we save checkpoints after every epoch. This gives us a list of models. The checkpoints are saved using ModelCheckpoint method with auto mode so that the most useful models are saved rather than saving all the models.

We use mean squared error as our loss function and adam optimizer for gradient descent. These parameters are passed while compiling the model.

To train the model we use the fit method in Keras, we specify training data in batch generator and testing data in the validation data parameter of the method.

The other file is drive.py that is used to send and receive data. The idea is to have a circular loop where the simulator acts as a server and our python program acts as a client. The server sends picture from the camera. The difference between training and testing is for training images from three cameras are needed while for testing only a single image from the center camera is required.

So, the simulator sends images from the center camera and the CNN predicts the steering angle and throttle for the car which is then sent to the server. All this is done using flask-socketIO.

V. DATA DESCRIPTION

Data is generated using the Udacity's simulator. The simulator has a training mode that used to drive the car on the track. This run can be recorded and it generates a csv file with 7 columns. The first 3 columns contain absolute path to the three images from left, center and right camera and the rest 4 have parameters from the car steering angle, speed, throttle and brake.

VI. RESULTS

The paper defines autonomy as the amount of time the car can run without human intervention but in the simulator, there is no way to human inputs. As mentioned above, we define autonomy as the amount of time the car can follow the road. We did various experiments using different hyper-parameters and by changing the CNN. We observed the following results.

Model	Time
CNN described in original paper	6 minutes
CNN with learning rate 10e-5	1 minute 23 seconds
CNN with learning rate 10e-3	23 second
CNN with 4 convolutional layers	2 minutes 30 seconds
CNN with Max Pooling layer after 1 convolutional layer	30 second
CNN with Max Pooling layer after 2 convolutional layers	12 minutes 30 seconds
CNN with Max Pooling layer after all convolutional layers	>20 minutes

VII. CONCLUSIONS

It has been observed that adding a layer of max pooling operation after every convolutional layer improves the performance of the network.

REFERENCES

[1] Mariusz Bojarski et al "End to End Learning for Self-Driving Cars"

[2] Flask-SocketIO https://flask-socketio.readthedocs.io/