Steering Angle Control of Self Driving Car by CNN Network and Image Processing

August 3, 2020

1 Motivation

The automobile industry is heading towards end to end self driving cars. Self-driving cars are a way to reduce road accident and congestion. It will also automate the entire driving sector. Presently, some countries are implementing self-driving cars for short-distance travel like from plane to the airport, trip to the garden etc.

In this project, we are concerned with one of the most critical portions of the self-driving car - steering angle control of a car based on vision. We have implemented the same using CNNs and image processing. For the purpose of simulations we used the udacity simulator of a car. In our project, we assumed that the road is flat and doesn't have any slops. This allowed us to train our model by just three cameras.

2 The main goal of the project

The Goal of our project is to define the model which works satisfactory for the flat road surface. We train the model and validate on the simulator.

The udacity simulator gives us the access to test our model by using the python interface. (The link for the this simulator is given in the next section). In this simulator there are two tracks - one with flat road and another one with hilly road. We used the first track to validate our model.

The basic requirement of the any self-driving car is to reduce the human interference. So our goal is to reduce the human interference to drive the car in the middle of the road. In the literature they define this as 'autonomy' for the measurement of the human interference. Below is the equation for that

$$autonomy = (1 - \frac{numberofinterventions * X}{elapsedtime[seconds]}) * 100$$

In this equation X = time required by the human to drive car to centre lane(second), Number of intervention = no. of times human has to inter-

fere(second). elapsed time= total driving time(second). We wish to use this as metric to judge the performance of our model.

3 Related literature

1. Udacity self-driving car simulator
2. End to End Learning for Self-Driving Cars

4 Approach and different models

In this project we trained different models for predicting the steering angle for image taken by the center camera placed in front of the car. Training data consist of the three images taken by the left,centre and the right cameras which are positioned in front of the car as shown in below fig.1

Figure 1: Left, centre and right image generated by the simulator for training

During training we used all three images to train our models. We added some offset in the steering angle for the left and the right images as shown in the fig. 2. We normalised image intensities to values between -1 to 1.

After training our model we only have centre camera as output of the model as shown in fig.3. In paper they tested their model in real world for this they send output by the wire interface, but in our case we used the python code provide by the Udacity to test our model in the simulator. We did change the code according to the our model to do pre-process image. To control the speed we used Proportional Integral (PI) controller. PI controller works by taking a variable as input (in our case speed) and tries to reduce the error of the variable compared to a constant set point value. We used a set point of 10 miles-per-hour(mph), we had to use a low value because the real-time computation associated with calculation of steering angle was high for our hardware. We also had to implement a low pass filter at the output to avoid sudden changes in the output steering angle.

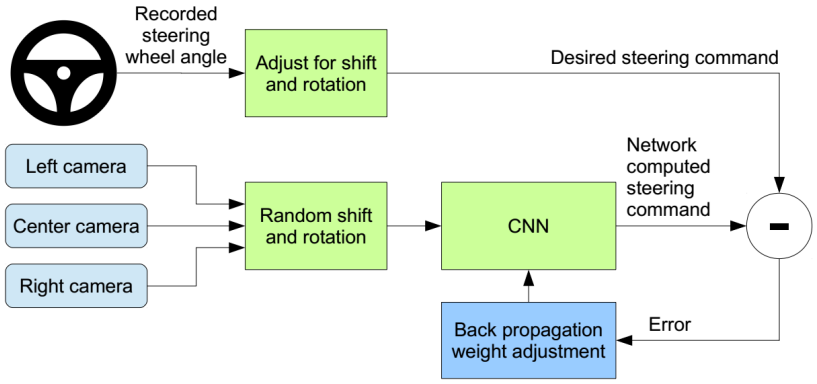

Figure 2: shift for the image taken by the right and left camera during training (Source: Taken from [2])

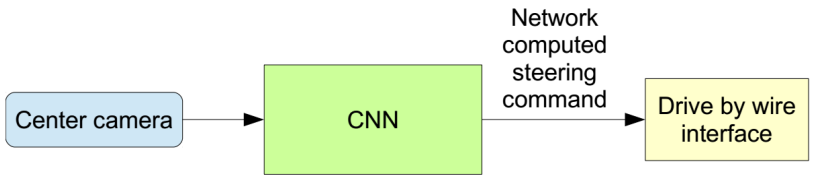

Figure 3: During test image of only centre camera is used(Source: Taken from [2])

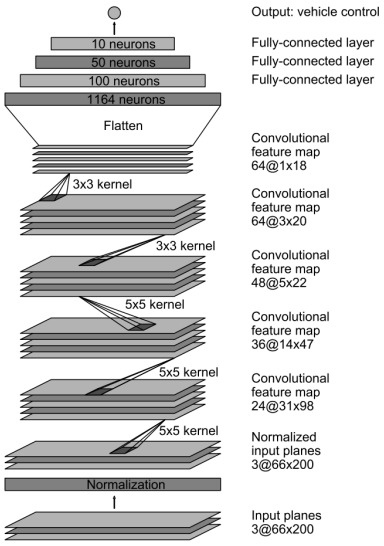

Figure 4: During test image of only centre camera is used (Source: Taken from [2])

Description different models we tried is given below.

4.1 CNN approach

Initially we tried to implement the model as given in the paper (shown in the fig.4). Below are the observations made during training and validation, and steps taken to resolve the problems:

1. The number of parameters for this model are in the range of the 30 million. As our training data set is too small(15000 images) we weren't able to train it well.

2. Our image size was different from the literature images size. We did change the convolution layers to get the same output matrix as the literature. This model doesn't work as the network is very dense.

3. We trained the model with reduced size of the hidden layers. This worked very low speed (less than 5 mhp).

4. We also tried to train this model by the [1/steering angle] as mention in the literature. But this model failed to converge even after.

5. We removed the unnecessary data by cropping the images to remove the upper part of the image not containing the road (as shown in the fig.5). For this approach we used a model with less number of layers than in the literature. Despite that it gave us better result than above approaches.

Figure 5: Left, centre and right image generated by the simulator for training

4.2 Image preprocessing and a feedforward Neural Network approach

In this we did edge detection by the preprocessing the image as shown in the fig.6. We removed all the Convolution layers and trained the model by only the NN. Observation made during training and steps followed to resolve the problem are as follows:

1. We trained only NN using preprocessed image. This model didn't work as good as the preprocessed image had a lot of unnecessary data.

2. To solve the above problem we croped the image for the road as shown in the fig.7 and train the simple NN network. This model worked well for the road with marking. However, it failed to predict the steering angle on the section of the road where no marking (eg on the bridge). The reason for this is that the preprocessed image of the bridge lacked the data for the lane or the direction. This model works for even high speed (greater than 10 mhp) as number of the hidden layer is very low.

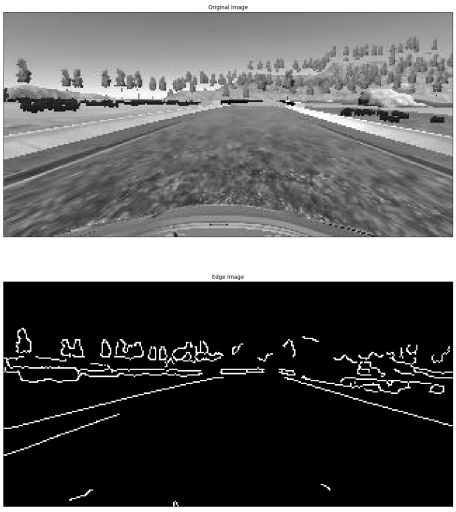

Figure 6: During test image of only centre camera is used

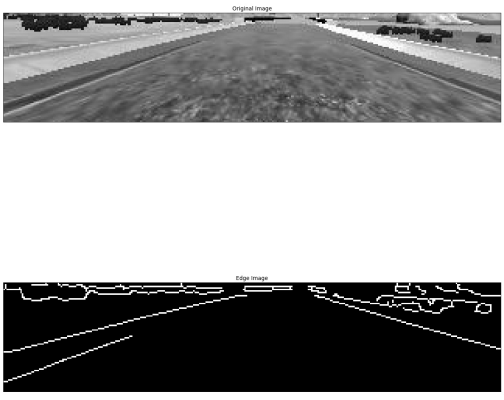

Figure 7: During test image of only centre camera is used

4.3 Image preprocessing and CNN approach

We trained different CNN model with preprocessed and the cropped image. Description of the different approach is given below.

1. As the first trail we used only one CNN layer with one layer preprocessed edge detected layer image (gray scale). It gave good results even during the high speed as this model has lower number parameter. The autonomy for this model was as high as 90%.

2. We appended the edge detected layer to original RGB image and made the input into 4 layer image and trained a dense CNN. This didn't work well as we had very less training data. This may be the best approach for someone having very high calculation power.

5 Description of the code

Average length of the code is around 180 lines. We used PYTORCH package for the making CNN and NN. For image processing and data acquisition, we used the CV2 package and CSV package.

As an activation function instead of ReLU, We used ELU activation because of its following advantages over ReLU.

1. ELU becomes smooth slowly until its output equal to $-\alpha$ whereas RELU sharply changes value near zero.

2. Unlike ReLU, ELU can produce negative outputs.

Below the function value for the given input x and function graph shown in fig.8.

$$ELU(x) = \begin{cases} \alpha(exp(x) - 1) & if\ x \leqslant 0, \\ x & if\ x \geqslant 0. \end{cases}$$

Average training time for the different model is given bellow. We trained and validate our code on system which consistant of Intel(R) Xeon(R) CPU X5680 @ 3.33GHz, 12GB ram,2GB graphic card(GM107 [GeForce GTX 750 Ti]).

1. CNN approach : dense model takes 6 hours to train, light model takes around 3 hours to train.

2. Image preprocessing and a NN approach: This model takes around 2 hours to train.

3. Image preprocessing and CNN approach: Single convolution layer CNN takes around 2 hours to train while dense CNN takes 6 hours to train.

here is the link for the our codes.

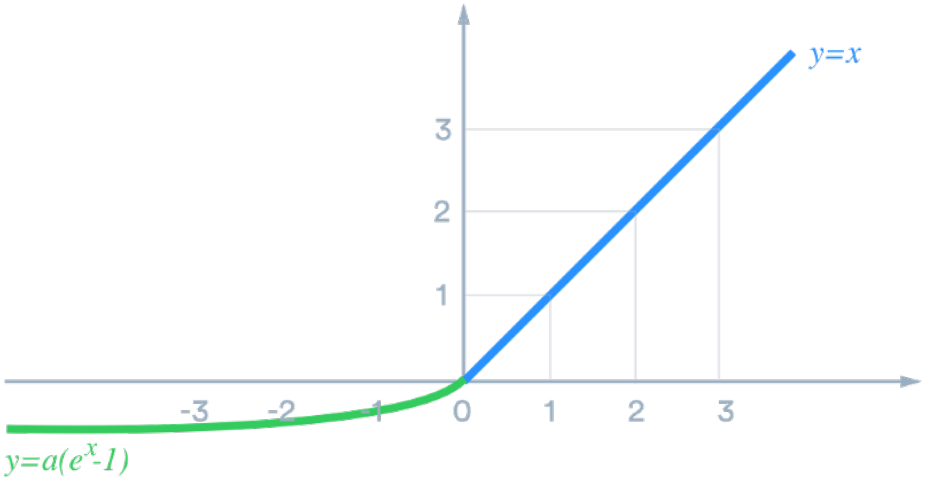

Figure 8: Function graph of ELU