Simulation of Autonomous Car using Deep Learning

Nidhi Gupta[1], Rishabh Kumar[2], Raju Patel[3], Raghvendra Kumar[4], Shivam Rathour[5]
[1]*Assisstant Professor, CSE Department, Raj Kumar Goel Institute of Technology*
[2,3,4,5]*Student, Raj Kumar Goel Institute of Technology*

Abstract **- In this world of rapid advancement of computer technologies, like CNN, open-cv etc., Deep learning has grown tremendously in the field of artificial intelligence and can be used to automate almost anything, that includes modern technologies. These technologies can be applied to the car so that it requires minimum interaction with driver to run on the road or we can say that to program a car in such way that it drives in self-governing mode. With help of existing simulator, we can use it to generate the enormous amount of data that includes images and csv containing car details. We train the network with generated images (left, right, center) to predict the required steer angle to keep the car on track. This approach decreases down the resolution of images to train the network very rapidly. Before sending the data to network it is preprocessed which is very much beneficial. After the preprocessing data is send to convolutional neural network in form of fixed size batches formed by random collection of images with their corresponding steering angle within the dataset generated to train and predict the steering angle as a final result. The Model achieved better performance when it is provided even more dataset. Here, we observe many Convolutional neural network architectures to obtain better performance with lesser load.**

Index Terms **- Autonomous car, CNN, Multilayer.**

I.INTRODUCTION

Autonomous cars are the cars that require no interaction with driver to run on the track. Autonomous car relies on central processing unit, Graphics processing unit and sensors to run car without the occurrence of faults.

If a person wants to go hospital or to some other place but he or she not able to drive or don't know how to drive, in this case the person will have to hire a driver which may be costly for person to afford. That situation might be unpleasant for the car owner. To avoid such type of unwanted situations autonomous car can play a key role and can be affordable for the person.

Here, with the help of simulator we are creating an autonomous car that has capability to sense its surrounding environment through the three-camera placed in front the car. And to compute the steer angle by which the car have to turn to keep itself on track. This will save the people's lives by avoiding accident like drunk and drive and it will keep car on track by avoiding running of the road With the rapid development of various types of sensors such as radar, camera systems and wireless communications, autonomous driving assistance systems have come a long way in past years. Self-driving cars or autonomous vehicles are reaching a point where they're as good, or better than human drivers. Companies like Google, Tesla, and Uber are each pushing the limits of innovation to dominate the space. The main requirements for autonomous vehicles are to reduce the incidence of accidents and traffic jams, to respect traffic regulations and to be able to drive longer distances more safely. All of these without human intervention.

Autonomous car uses the supervised learning as it takes steering angle as target value along with the images. Prediction of the steering angle can be made possible using multi-layer convolutional neural network architecture because single layer CNN cannot process the highly complexed images and cannot predict the desired target value.

The purpose of the paper is to reduce the loss function between the actual and predicted values of the target value so that that the accuracy of the model can be increased on the given large dataset generated by simulator. By generating the large amount of dataset, we can use it to train the CNN model to predicted steering angle. Our primary goal is studying the three different CNN architecture to choose well sophisticated architecture that captures and mimic the driving scenario of a driver in real life.

II. REALATED WORK

In past few years Deep learning networks are very helpful in automation filed. Convolutional neural networks (CNN) are the most important for image processing. CNN is used in enormous number of technical fields like handwriting recognition, face recognition and various application in computer vision. They are introduced by Yann LeCun, a postdoctoral computer science researcher, to create a very basic of image recognition network. CNN is very important neural network. These are used recognition, segmentation and prediction of target value.

The DARPA (Defense Advance Research Project Agent) created an autonomous driving system known as DAVE which used images captured from the two front cameras along with steering angle to train the system to drive.

The Autonomous Land Vehicle in a neural network (ALVINN) is multilayer backpropagation network developed by researchers at Carnegie Melion University. It uses images captured by camera and distance measure by laser to train the machine to predict the direction in which car should move to keep the car on track.

There are many pre-trained networks for image recognition. Among from them VGG16 have show very good results but it contains huge number of parameters, thus it requires enormous number of resources to train the machine and can lead to overfitting of data which may predict the wrong steering angles and can led the car run into an accident.

III. METHODS

A. Data gathering and simulator

We used the term 1 Udacity simulator to generate the data. This Udacity simulator is build with the help of Unity and it is open sourced for everyone to use it. we can easily generate the data by using simulator. The data is generated in form of csv log file containing state of vehicle (steer angle, brake, throttle and name of the images (left, centre, right) captured by cameras (left, centre, right) at an instance) and it also contain folder containing those images mention in CSV log file.

We collected nearly 30000 images collectively from all three cameras (left, centre, right) in 1920X1050 resolution along with vehicle data when the car is driven around the track.

Fig. 3.1.1 Centre camera image

Fig. 3.1.2 Left camera image

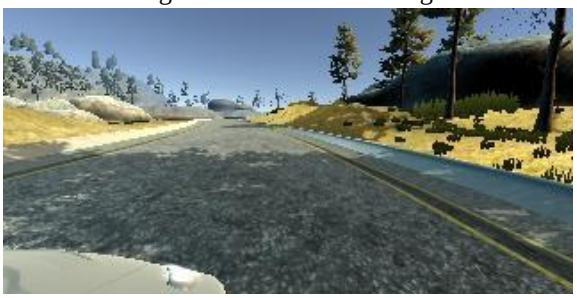

Fig. 3.1.3 Right camera image

B. Avoiding overfitting

To avoid the overfitting of the model on any value during the training we need to remove some of the data on that value. In the dataset, there are enormous number of data entries of the steering angle are very close to zero. So, we deleted some of the entries in dataset to spread the values across the range and to avoid the overfitting.

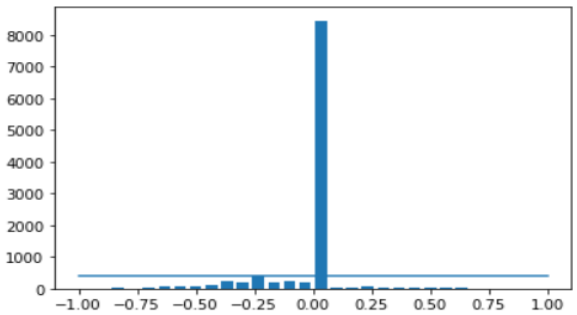

Fig. 3.2.1 Data before.

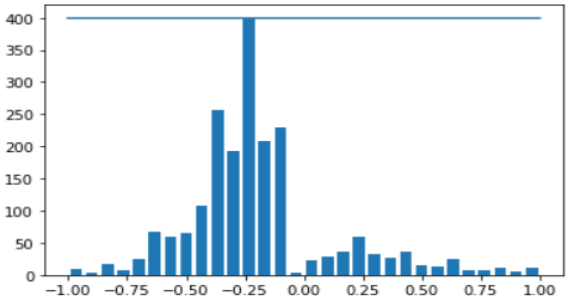

Fig 3.2.2 Data after.

C. Lowering the resolution

The images generated by the simulator are in high resolution (1400X1050), so to reduce the training time we lowered the resolution of the captured images. Thus, in this way, we have prevented the machine from training with noisy data present in the image that could affect the performance of the model. And by downsizing and cropping we remove the unwanted object present in the image like road side trees, pole and sky etc. and keeps only the main parts like track in the centre of the image.

D. Image Flip

The chosen random image is flipped horizontally in the process along with its steer angle by multiplying it by -1. After the image is chosen then it is decided with the constant probability to whether to flip the image or not. This helps to add new data to the dataset if the image is flipped so that model can be trained in every possible steering angle more accurately.

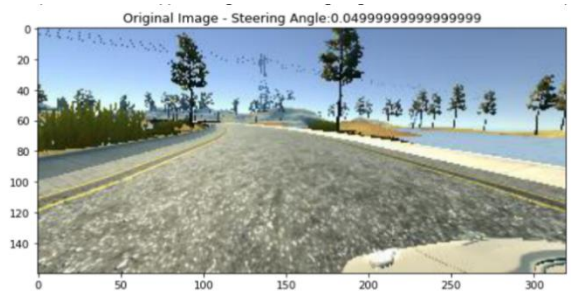

Fig. 3.4.1 Original image

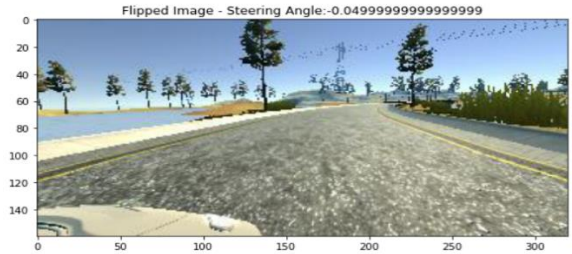

Fig. 3.4.2 Flipped image

E. Image processing

After the downsizing and cropping of the image, it is converted from RGB image to YUV image. YUV is a colour coding system that is commonly used in colour image pipeline. The main reason is that brightness is important than the image colour, so you can lower the resolution of V and U while keeping Y resolution same. Therefore, the u and v resolutions can be lowered, they are compatible with CNNs, and CNNs can be trained with the same precision many times faster than traditional RGB models. After the conversion we normalise the pixel values of whole image by dividing it by 255 and then store it into multidimensional array to feed it to network.

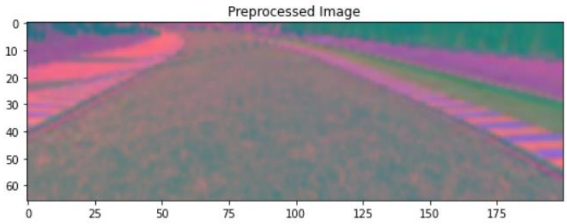

Fig 3.5 Processed image ready for input.

F. Convolutional neural network architectures

We have designed three multilayer CNN architecture with the aim of comparing these architectural models to get maximum accuracy model and to minimise the loss while training the model.

The first architecture consists of five Convolutional layers along with varying filter layers like Maxpooling, Dropout, Flatten, and Dense layer. In this last Dense layer output only one value which is considered the predicted steering angle for processed input image passed to first convolutional layer as shown below in Fig. 3.6.

Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 32, 99, 32)	896
max_pooling2d_4 (MaxPooling2	(None, 16, 49, 32)	0
dropout_8 (Dropout)	(None, 16, 49, 32)	0
conv2d_8 (Conv2D)	(None, 7, 24, 64)	18496
max_pooling2d_5 (MaxPooling2	(None, 3, 12, 64)	0
dropout_9 (Dropout)	(None, 3, 12, 64)	0
conv2d_9 (Conv2D)	(None, 1, 5, 128)	73856
dropout_10 (Dropout)	(None, 1, 5, 128)	0
flatten_2 (Flatten)	(None, 640)	0
dense_4 (Dense)	(None, 1024)	656384
dropout_11 (Dropout)	(None, 1024)	0
dense_5 (Dense)	(None, 1)	1025

Total params: 750,657
Trainable params: 750,657
Non-trainable params: 0

Fig. 3.6 Architecture of first model.

In this architectural model we use small kernel size of (3,3) because it reduces computational costs and Weight sharing that ultimately leads to lesser weights for back-propagation. And in this architecture, we use the RELU activation function.

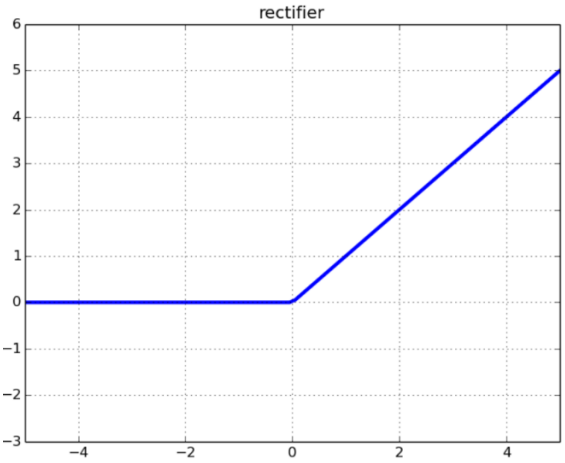

Fig 3.7 The graph of RELU activation function.

$f(x)=max(0,x)$

In the second architecture, there are four convolutional layers and also along with varying no. of filter layers Maxpooling, Dropout, Flatten, and Dense layer. The output of last Dense layer is the steering angle calculated for the input image. The architecture second of the CNN model is shown in Fig. 3.8.

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 31, 98, 24)	1824
conv2d_7 (Conv2D)	(None, 14, 47, 36)	21636
conv2d_8 (Conv2D)	(None, 5, 22, 48)	43248
conv2d_9 (Conv2D)	(None, 3, 20, 64)	27712
conv2d_10 (Conv2D)	(None, 1, 18, 64)	36928
dropout_1 (Dropout)	(None, 1, 18, 64)	0
flatten_1 (Flatten)	(None, 1152)	0
dense_4 (Dense)	(None, 100)	115300
dense_5 (Dense)	(None, 50)	5050
dense_6 (Dense)	(None, 10)	510
dense_7 (Dense)	(None, 1)	11

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

Fig. 3.8 Architecture of second model

we use the ELU activation function. And also, we took set the strides to (2,2) and kernel size to (5,5) for the first three convolutional layers and for the rest convolutional layers we set kernel size to (3,3) to increase the performance of the architecture. And expectedly the training parameter were decreased to train the model in less time.

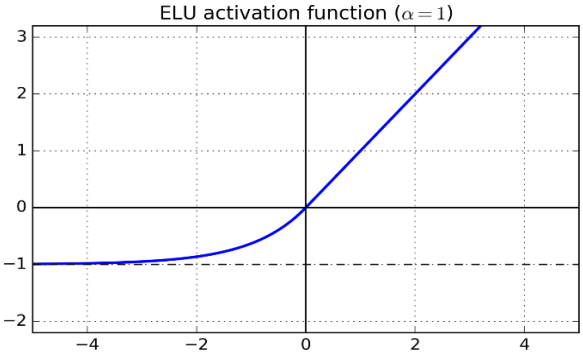

Fig 3.9 the graph of ELU function.

$$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & for\ x < 0 \\ x & for\ x \geq 0 \end{cases}$$

In the third architecture, we have Five convolutional layers, with some more regulators to increase the accuracy the architecture. In this architecture we there are large no. of trainable parameter because it extracts very large features from the images which help it to predict more accurate steer angle. In this we use the ELU activation function to predict continuous value in range of {-1, 1}.

This model should be able to predict more accurate value with the reduced loss over the epochs. This architecture takes much time to train the model.

Layer (type)	Output Shape	Param #
conv0 (Conv2D)	(None, 66, 200, 3)	12
conv1 (Conv2D)	(None, 32, 99, 32)	896
conv2 (Conv2D)	(None, 15, 49, 32)	9248
max_pooling2d (MaxPooling2D)	(None, 7, 24, 32)	0
dropout (Dropout)	(None, 7, 24, 32)	0
conv3 (Conv2D)	(None, 3, 11, 64)	18496
dropout_1 (Dropout)	(None, 3, 11, 64)	0
conv4 (Conv2D)	(None, 1, 5, 128)	73856
dropout_2 (Dropout)	(None, 1, 5, 128)	0
flatten (Flatten)	(None, 640)	0
dense (Dense)	(None, 1024)	656384
dropout_3 (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 1)	1025

Total params: 759,917
Trainable params: 759,917
Non-trainable params: 0

Fig 3.10 Architecture of third model.

G. Training and validation

By using the train_test_split function from the sklearn we split the total image dataset of 31845 images between training set and the validation set like 80%

goes to training set and remaining 20% goes to validation set randomly. We set the batch size to 300 images for training and to 150 images for validating. We set the step_per_epoch to 300 and validation_step to 150 and epoch to 30.

$$MSE = \frac{1}{n}\sum(Yi - \hat{Y}i)^2$$

Where Yi is the actual value,

$\hat{Y}i$ is the predicted value,

n defines the number of training samples.

We use Adam optimizer with learning rate of 10-4 and loss as MSE (mean squared error) which compute the loss on predicted target value and actual target value.

IV. RESULTS

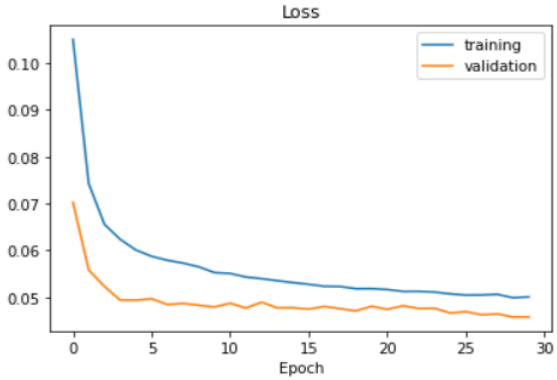

Fig. 4.1 The loss over epoch during training and validation for the first architecture.

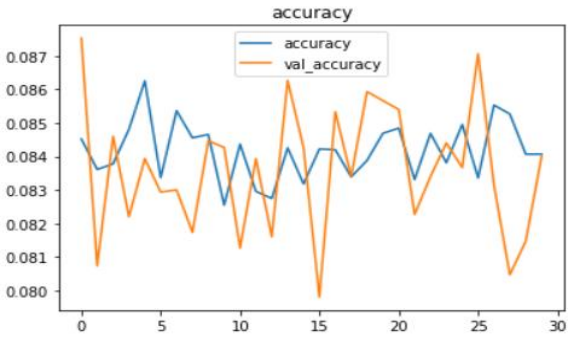

Fig. 4.4 the accuracy over epoch for second architecture.

The above fig. 4.1 shows the loss computed by MSE loss function during every epoch for the first CNN architecture of. We see the drastic decrement in the loss over the first few epochs during the training and validation. And for the rest of the remaining epochs loss decrease very slowly. Here loss at the 30th epoch is 0.045 for validation.

The fig 4.2 demonstrate the accuracy for the first CNN architecture over epochs during training and validation. This graph shows the average accuracy of 85% for training and validation.

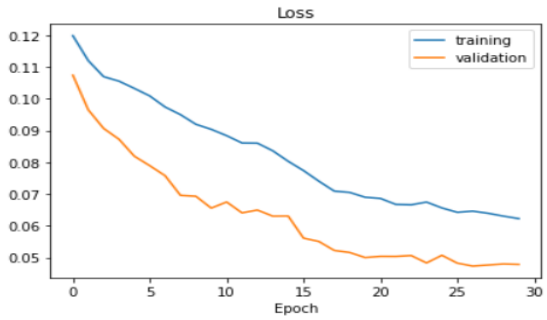

Fig. 4.3 The loss over epoch for second architecture.

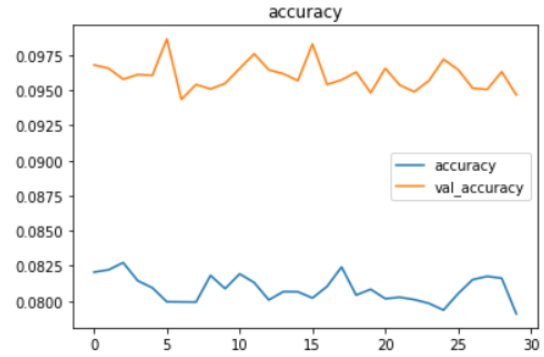

Fig. 4.4 the accuracy over epoch for second architecture.

The Fig 4.3 demonstrate the loss function value over the 30 epochs during the training and validation. We see the slight decrease in the loss over every epoch. The observed loss during the training is 0.062 and during validation is 0.047.

As shown in the Fig. 4.4 the validation accuracy 94.7% which is very high. And training accuracy is 79.1% which is comparatively very low than validation accuracy.

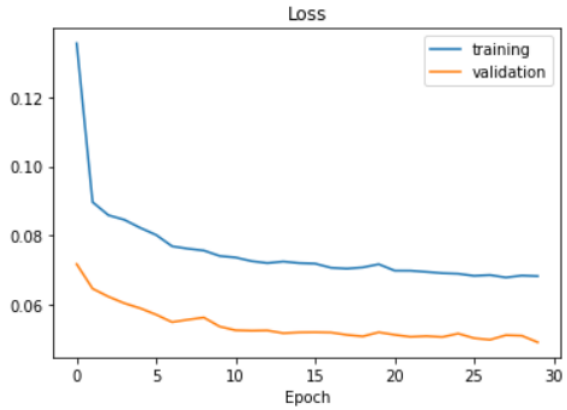

Fig. 4.5 The loss over epoch for third architecture.

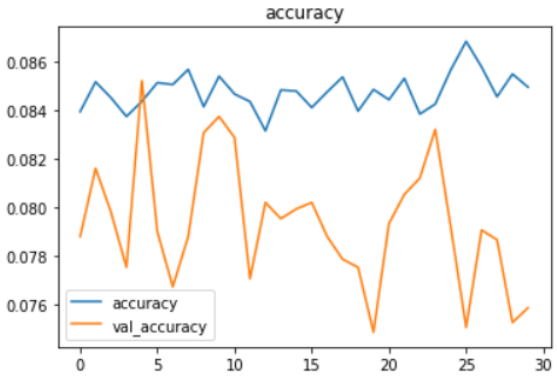

Fig. 4.6 the accuracy over epoch for third architecture.

In the Fig 4.5 for the first few epochs loss has rapidly decreased then after for remaining epochs less decrement is seen. The training loss is at 30th epoch is 0.068 and the validation loss is 0.049.

In Fig. 4.6 it is seen that validation accuracy is less than training accuracy. The validation a accuracy is 75.9% and training accuracy is 84.2%.

By analysing the facts, we see loss over the epoch is decreasing continuously which mean the model is computing more accurate target value then before. So we can say that architecture with more training parameter will perform better. Hence, neural network architecture first and neural architecture third will give better performance than architecture second because that have large number of trainable parameters.

IV. CONCLUSION

In this research paper, we explained the need of simulator to generate the data which turn out to be very effective for the model to train to compute the steering angle based on input images. The use of simulator save cost of physical equipments required to collect the real-world data. And simulator turn out to be very well efficient for training the model for prediction steering angle.

The prediction of the steering angle based on the processed captured images turn out to be fine for CNN network because it extracts the feature and find the necessary dependencies required for prediction.

In this paper, we compared the three CNN architecture to know which architecture can give better performance. Here we find out that the more parameter architecture has the better the performance will be. The first and third architecture the show the better result than second architecture. The first architecture shows the accuracy of 85% and it is more than the other two architectures. The architecture with lesser trainable parameter should be trained over many more epochs than epochs in architecture have larger training parameter to give same performance.

REFERENCES

[1] LeCun, Y., et al. DAVE: Autonomous off-road vehicle control using end-to-end learning. Technical Report DARPA-IPTO Final Report, Courant Institute/CBLL, 2004.

[2] Pomerleau, Dean A. "Alvinn: An autonomous land vehicle in a neural network." Advances in neural information processing systems. 1989.

[3] Bojarski, Mariusz, “End to end learning for self-driving cars." (2016).

[4] A. R. Diana Putri, Litasari and A. Susanto, "Comparison between colour models in automatic identification of cane sugar," 2013 IEEE International Conference on Computational Intelligence and Cybernetics Yogyakarta, 2013.

[5] “The Effectiveness of Data Augmentation in Image Classification using Deep Learning” Jason Wang, Luis Perez.

[6] Visualizing and Understanding Convolutional Networks by Matthew D. Zeiler, Rob Fergus.

[7] Dropout: A Simple Way to Prevent Neural Networks from Overfitting by Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov

[8] S. Marra, M. A. Iachino and F. C. Morabito, "Tanh-like Activation Function Implementation for High-performance Digital Neural Systems," 2006 Ph.D. Research in Microelectronics and Electronics, Otranto, 2006.

[9] “Multi-Layer Neural Network.” Unsupervised Feature Learning and Deep Learning Tutorial. Stanford University, n.d. Web.