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 also been used to generate training data required for model training. It is a standard practice to make use of NVIDIA model architecture to predict steering angles which are further fed to the simulator. However, this paper explores the use of various available pre-trained models and also a simple CNN model built on our own for this use case. When we compared the performance of all these models, it has been observed that NVIDIA model performed better than all of these pre-trained models and our own CNN model. The performance of these models have been measured based on the time duration of the car for which it drives without going off-track.

1. Introduction

WHO(World Health Organization) report [9] 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 [7] 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.

1.1. Model Architectures

Firstly, we have used the model architecture developed by NVIDIA research team with 5 concolutional layers for an end-to-end learning of self driving cars to measure its performance. They have developed it

Secondly, we have built a simple CNN model architecture and trained it. It consists of 7 convolution layers, 3 pooling layers, 2 fully connected layers and 1 output layer.

In addition to these two model architectures, this project has utilized the below listed pre-trained models to check the performance of the self-driving car in the simualtion environment - Alexnet, Googlenet, Resnext50, Squeezenet, Densenet201, Efficientnet, Nasnetalarge, Resnet101 and Xception

2. Related Work

A paper on Autonomous vehicles: A study of implementation and security [2] looked at the pros and cons of implementation of autonomous vehicles. It has discussed about the benifits such as safety, congestion and traffic management, adoption of autonomous vehicle technology by vari-

ous sectors like mining, freight transportation and the military industry. It has mentioned that valid and accurate sensor data is critical for the successful router planning, emergency maneuvers and route calculations. Any tampering or manipulation of the data generated and transmitted by these can have disastrous consequences, as human lives are at stake here. Security attacks on various sensors and onboard cameras have exposed several vulnerabilities of the autonomous vehicles. The wireless connectivity exposes the vehicles to DoS and malware attacks. This paper has concluded that, for efficient implementation, a secure wireless technology which encompasses the highest level of privacy and security is critical.

A paper on Key Points Estimation and Point Instance Segmentation Approach for Lane Detection [3]has focused on Lane detection, which can localize the drivable area on a road, is a major perception technique. There are many ways to recognize lanes, but most techniques utilize traffic line detection. In this paper, they proposed a traffic line detection method called Point Instance Network (PINet); the method is based on the key points estimation and instance segmentation approach. The PINet includes several stacked hourglass networks that are trained simultaneously. They casted a clustering problem of the predicted key points as an instance segmentation problem; the PINet can be trained regardless of the number of the traffic lines. The PINet achieves competitive accuracy and false positive on the TuSimple and Culane datasets, popular public datasets for lane detection. This paper has also established that PINet show better performance than other methods in difficult light conditions such as night, shadow, and dazzling light. However, PINet has limitations when local occlusions or unclear traffic lines exist. They have shown by ablation study that the knowledge distillation method improves the performance of the clipped short network. As a result, they have observed that the clipped short network's performance is close to that of the whole network's performance.

A paper on Virtual to Real Reinforcement Learning for Autonomous Driving [6] proposed a novel realistic translation network to make model trained in virtual environment be workable in real world. The proposed network can convert non-realistic virtual image input into a realistic one with similar scene structure. Given realistic frames as input, driving policy trained by reinforcement learning can nicely adapt to real world driving. Their experiments have shown that their proposed virtual to real (VR) reinforcement learning (RL) worked pretty well. They concluded that by using synthetic real images as training data in reinforcement learning, the agent generalizes better in a real environment than pure training with virtual data or training with domain randomization.

A paper on YOLOP: You Only Look Once for Panoptic Driving Perception [10] foused on a high-precision and real-time perception system can assist the vehicle in making the reasonable decision while driving. They presented a panoptic driving perception network (YOLOP) to perform traffic object detection, drivable area segmentation and lane detection simultaneously. It is composed of one encoder for feature extraction and three decoders to handle the specific tasks. Their model performed extremely well on the challenging BDD100K dataset, achieving state-of-the-art on all three tasks in terms of accuracy and speed.

A paper on Stereo R-CNN based 3D Object Detection for Autonomous Driving [4] proposed a 3D object detection method for autonomous driving by fully exploiting the sparse and dense, semantic and geometry information in stereo imagery. Their method, called Stereo R-CNN, extends Faster R-CNN for stereo inputs to simultaneously detect and associate object in left and right images. They added extra branches after stereo Region Proposal Network (RPN) to predict sparse keypoints, viewpoints, and object dimensions, which are combined with 2D left-right boxes to calculate a coarse1 3D object bounding box. They then recovered the accurate 3D bounding box by a region-based photometric alignment using left and right RoIs. Their method does not require depth input and 3D position supervision, however, outperformed all existing fully supervised image-based methods. Experiments on the challenging KITTI dataset have shown that their method outperforms the state-of-the-art stereo based method by around 30 percent AP on both 3D detection and 3D localization tasks.

Regarding set-up of the Udacity simulator, an article by Mr. Naoki [5] 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. He has very well explained the installation process of the simulator. The Figure 1 shown below corresponds to Unity environment.

Unity Editor

Figure 1: Unity Editor

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 powerful 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 in Figure 2.

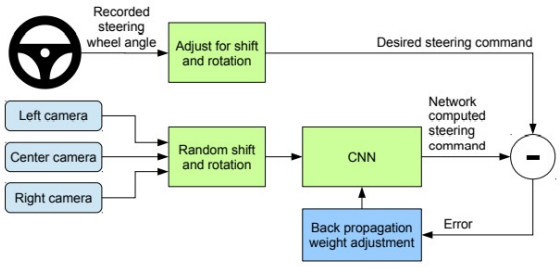

Figure 2: Block diagram of NVIDIA training system

We have also referred to the approach followed by Zhenye-Na [11] 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 [8]. 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.

3. Methods

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 in the simulator. This overall approach is shown in Figure 3.

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,

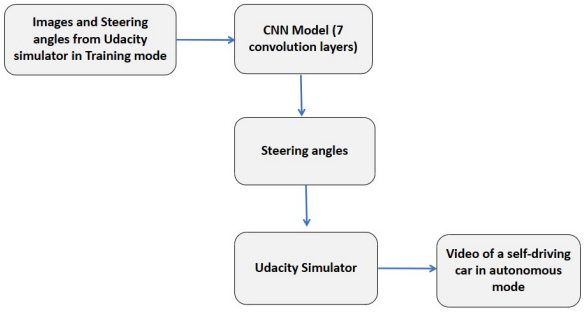

Figure 3: Block diagram of overall approach

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

We have used the following pre-trained models to predict steering angles - Alexnet, Googlenet, Resnext50, Squeezenet, Densenet201, Efficientnet, Nasnetalarge, Resnet101 and Xception. As we are predicting just one value, we have changed the number of output classes to '1' in the output layer of all these models.

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

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

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

4. Results

The block diagram shown in Figure 4 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 in Figure 5.

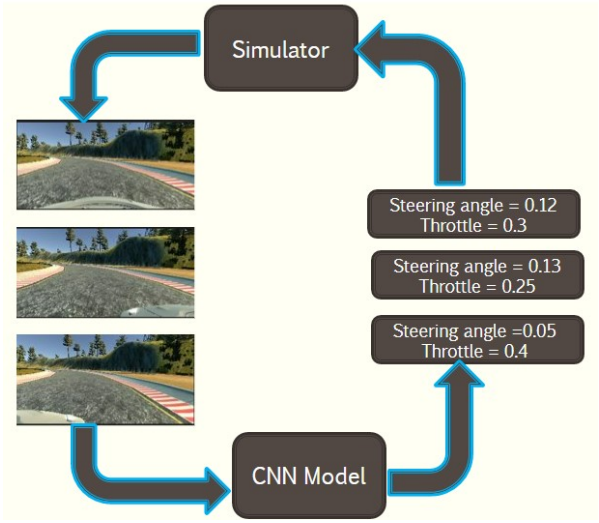

Figure 4: Block diagram of testing approach in the simulator

Figure 5: Snapshot of steering angles and simulator windows

After training my model for 50 epochs, we have obtained below results shown in Figure 6 and Figure 7. Figure 6 shows the actual steering angles and predicted steering angles by all the models for 22 test images taken at random. In addition to this, we have also calculated mean squared error and the run time of the car with 48 fps(frames per second) on the track in the simulator before going off the track and is also displayed in the Figure 7.

In the runs1 folder mentioned above, images taken by the front camera during autonomous mode are have been captured. Samples shown in Figure 8 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

NVIDIA model		Our own CNN model		Alexnet		Googlenet		Mobilenet		Resnext50		Squeezenet	
Actuals	Predicted	Actuals	Predicted	Actuals	Predicted	Actuals	Predicted	Actuals	Predicted	Actuals	Predicted	Actuals	Predicted
0.000000	-0.164012	0.400000	0.390185	-0.200000	-0.413029	0.450000	0.454702	0.400000	0.312478	0.350000	-0.059730	-0.400000	0.000000
0.400000	0.633807	-0.400000	-0.201689	-0.400000	-0.212711	0.400000	0.311127	0.400000	0.268185	-0.400000	-0.504598	0.400000	0.402782
0.400000	-0.313648	0.400000	0.405525	0.150000	0.113537	-0.050000	-0.080268	-0.400000	-0.413642	0.400000	-0.055548	0.050000	0.000000
-0.400000	-0.305965	0.400000	0.346989	-0.400000	-0.552834	-0.400000	-0.172279	0.350000	0.364034	-0.400000	-0.426258	0.400000	0.457157
-0.400000	-0.568068	0.400000	0.243563	0.337404	0.331305	-0.400000	-0.064548	0.400000	0.128022	0.285519	0.081358	-0.400000	0.000000
0.400000	0.444104	-0.400000	-0.267561	0.400000	0.325581	-0.600000	-0.698352	-0.400000	-0.288486	0.400000	0.687338	0.285334	0.128831
-0.400000	-0.338435	-0.800000	-0.702031	-0.400000	-0.345265	0.500000	0.063031	0.600000	0.473940	-0.600000	-0.533170	0.600000	0.000000
0.245230	0.271731	0.400000	0.675449	0.350000	0.599421	-0.400000	-0.090100	-0.400000	-0.305959	0.400000	0.132041	0.400000	0.000433
0.400000	0.349102	0.550000	0.744039	0.400000	0.141696	-0.400000	-0.261200	0.350000	0.115864	0.350000	0.211284	0.304019	0.266565
0.400000	0.126524	-0.600000	-0.667463	-0.400000	-0.337530	0.150000	0.370031	-0.400000	-0.404758	0.700000	0.376021	-0.050000	0.066786
-0.400000	-0.355197	0.400000	0.273889	-0.400000	-0.317810	0.150000	0.215809	-0.400000	-0.177160	-0.702466	-0.347825	1.400000	1.313614
-0.400000	-0.337081	-0.400000	-0.397824	0.100000	0.044452	-0.400000	0.084145	-0.400000	-0.396702	-0.400000	-0.350153	-0.161804	0.068480
-0.537208	-0.497585	-0.400000	-0.304498	0.400000	0.286877	-0.650000	-0.405436	-0.400000	-0.093751	-0.050000	-0.263364	0.400000	0.334642
0.400000	0.446516	0.400000	0.336916	0.600000	0.518862	0.400000	0.281209	-0.400000	-0.364283	-0.600000	-0.404902	-0.400000	0.000000
0.400000	0.389895	-0.400000	-0.210289	0.550000	0.562296	-1.400000	-0.993652	0.150000	0.181673	-0.061557	-0.053517	0.400000	0.670454
-0.400000	-0.443279	-0.400000	-0.422980	0.200000	0.195487	-0.400000	-0.029574	-0.350000	-0.353578	-0.200000	-0.064314	0.200000	0.000000
-0.120642	-0.111900	0.400000	0.682997	0.400000	0.331961	0.400000	0.595860	0.400000	0.340024	0.700000	0.689830	0.400000	0.214673
0.300000	0.065128	-0.050000	-0.110995	-0.200000	-0.395128	0.400000	0.463194	-0.400000	-0.058386	-0.400000	-0.330922	0.400000	0.435587
0.400000	0.370176	-0.400000	-0.224604	0.954509	0.782850	0.400000	0.067759	0.349666	0.196666	-0.400000	-0.270940	-0.322511	0.000000
-0.400000	-0.383971	-0.300000	-0.341821	0.800000	0.608431	-0.400000	0.029909	-0.750000	-0.768130	0.100000	0.007140	-0.050000	0.000000
1.050000	0.658469	0.162953	0.282436	0.400000	0.383005	-0.225049	-0.072989	0.400000	0.217836	0.150000	-0.067900	0.000000	0.000000
0.150000	0.315054	0.650000	0.910603	-0.400000	-0.367347	0.400000	0.129276	0.600000	0.320074	-0.400000	-0.265528	0.400000	0.397356
-0.200515	-0.003046	0.450000	0.374912	-0.400000	-0.315321	-0.400000	0.015914	-0.600000	-0.648385	-0.750000	-0.540185	-0.400000	0.000000

Figure 6: Actual angles vs Predicted angles

Model	MSE	Run time (in secs) on track
NVIDIA model	0.021631496	90
Our own CNN model	0.020940233	62
Alexnet	0.017101737	50
Googlenet	0.07649981	50
Mobilenet	0.025872601	8
Resnext50	0.04703514	3
Squeezenet	0.066304624	12
Densenet201	0.023140233	3
Nasnetalarge	0.04562012	4
Resnet101	0.026940233	14
Xception	0.036470223	3

Figure 7: MSE and Run time

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 on the track, it may go off the track. So, the model has be be trained for more epochs and with data augmentation techniques 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 in Figure 9.

After selecting either track1 or track2 and then clicking on the training mode, we will see the screen shown in Figure 10.

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 in Figure 11, Figure 12 and Figure 13.

In addition to these images, it generates drivinglog.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

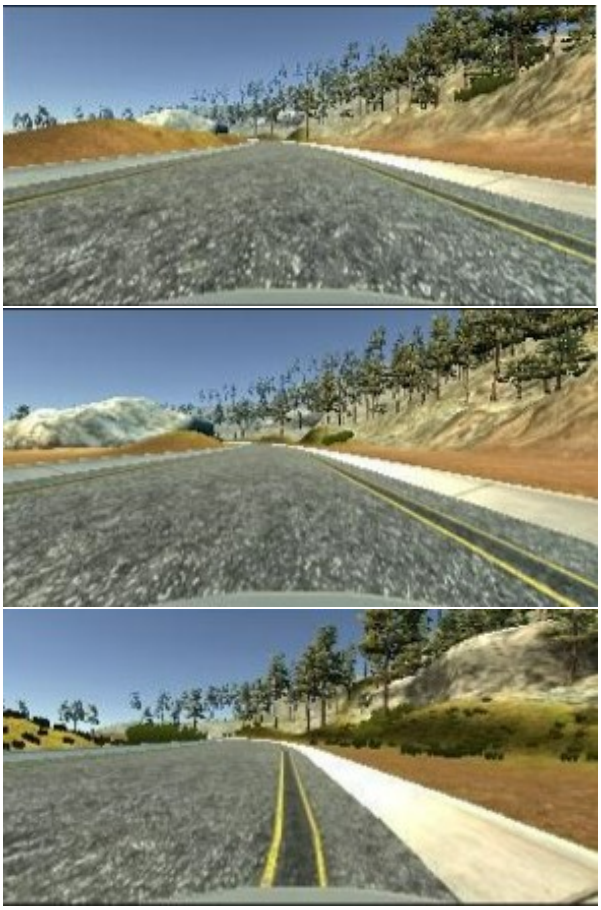

Figure 8: Sample images captured during autonomous mode

Figure 9: Udacity simulator envoronment

Figure 10: Training mode in simulator

Figure 11: Image from left camera

respectively. A snapshot of this csv file is shown in Figure 14.

In this project, we have generated 33096 images from the simulator and combined this with the training data of 97330 images available on Kaggle platform [11]. Out of these, 20 percentage of images have been used for validation purpose. 70x320x3 is the size the images generated in the simulator.

5. Discussion

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)

Figure 12: Image from front camera

Figure 13: Image from right camera

	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

Figure 14: Drivinglog.csv file

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 snapshots shown in Figure 15 and Figure 16 are an example of original and cropped images. This is required for better performance of the model.

Figure 15: Original image

Figure 16: 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. We also have observed that none of the pre-trained models preformed better than these 2 simple models. There is a great scope to further improve the performance by means of better data augmentation techniques. Going forward, we would like to analyze why the performance of all these complex pre-trained models is very poor. My future work will focus on building our own CNN model in such a way that it performs better than the NVIDIA model. The other ambitious activity for our future work is to build and train the model with images from real environment.

References

[1] M. Bojarski, B. Firner, B. Flepp, L. Jackel, U. Muller, K. Zieba, and D. D. Testa. End-to-end deep learning for self-driving cars. *arxiv.org/pdf/1604.07316v1*, 2016. 3

[2] F. Khan, R. L. Kumar, S. Kadry, Y. Nam, and M. N. Meqdad. Autonomous vehicles: A study of implementation and security. *International Journal of Electrical and Computer Engineering 11(4):3013-3021*, 2021. 1

[3] Y. Ko, Y. Lee, S. Azam, F. Munir, M. Jeon, and W. Pedrycz. Key points estimation and point instance segmentation approach for lane detection. *arxiv.org/pdf/2002.06604v4*, 2020. 2

[4] P. Li, X. Chen, and S. Shen. Stereo r-cnn based 3d object detection for autonomous driving. *arxiv.org/pdf/1902.09738v2*, 2019. 2

[5] Naoki. Introduction to udacity self-driving car simulator. *GitHub*, 2017. 2

[6] X. Pan, Y. You, Z. Wang, and C. Lu. Virtual to real reinforcement learning for autonomous driving. *arxiv.org/pdf/1704.03952v4*, 2017. 2

[7] S. Saha. A comprehensive guide to convolutional neural networks — the eli5 way. *towardsdatascience*, 2018. 1

[8] M. V. Smolyakov, A. I. Frolov, V. N. Volkov, and I. V. Stelmashchuk. Self-driving car steering angle prediction based on deep neural network an example of carnd udacity simulator. *2018 IEEE 12th International Conference on Application of Information and Communication Technologies (AICT*, 2018. 3

[9] WHO. Road traffic injuries. *WHO report*, 2022. 1

[10] D. Wu, M. Liao, W. Zhang, X. Wang, X. Bai, W. Cheng, and W. Liu. Yolop: You only look once for panoptic driving perception. *arxiv.org/pdf/2108.11250v7*, 2022. 2

[11] Zhenye-Na. End-to-end learning for self-driving cars — udacity's simulation env. *Kaggle*, 2019. 3, 5