Deep Learning Implementation of Autonomous Driving using Ensemble-M in Simulated Environment

Meenu Gupta (✉ gupta.meenu5@gmail.com)
Chandigarh University  https://orcid.org/0000-0001-7366-0841

Vikalp Upadhyay
Chandigarh University

Prince Kumar
Chandigarh University

Fadi Al-Turjman
Near East University: Yakin Dogu Universitesi

Research Article

Keywords: Autonomous Driving, CNN, ImageNet, MobileNetV2, NVIDIA Model, Self-Driving.

DOI: https://doi.org/10.21203/rs.3.rs-483461/v1

License: ☝️ ¹ This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License
Abstract

Making Autonomous Driving a safe, feasible and better alternative is one the core problems the world is facing today. The horizon of the applications of AI and Deep Learning has changed the perspective of the human mind. Initially, what used to be thought as subtle impossible task is applicable today and that too in the feasibly efficient way. Computer vision tasks powered with highly tuned CNNs are outperforming humans in many fields. Introductory implementations of autonomous vehicle were merely achieved using raw image processing and hard programmed rule-based logic systems along with machine/deep Learning used as secondary objective handlers. With the autonomous driving method proposed by NVIDIA, the usability of CNNs is more adequate, adaptable and applicable. In this paper, we propose, the ensemble implementation of CNN based regression models for autonomous-driving. We have taken simulator generated driving view image dataset along with mapped file of steering angle in radians. After applying image pre-processing and augmentation, we have used two CNN models along with their ensemble and compared their performance as to minimize the risks of unsafe driving. We have compared Nvidia proposed CNN, MobileNet-V2 as regression model and Ensemble-M results for comparison their respective performance, MSE scores and compute time to process. In result analysis, the MobileNet-V2 model performs better in densely-featured roads and Nvidia model performs better in sparsely-featured roads whereas Ensemble-M normalizes the performance of both models and efficiently result in the least MSE score (0.0201) with highest computation time utilization.

1. Introduction

Deep learning is a sub-category of Artificial Intelligence which enables machines to achieve intelligence mathematically to take decisions. Self-driving is one of the classic applications of AI super field where a small inconsistency in building the efficient solution may lead to severe loss of capital and even life. The self-driving application is not only limited to transportation on roads to carry passenger from one node to another but also have primary potential in the world shipping market, where it is implemented on a goods carrier truck, a rover explicitly designed for particular information extraction project, automated micro delivery units [26], IOT devices [10] and video game bots. Applications of Deep Learning which initially thought to be constant in number, are now explored with a different and innovative approach to attain almost any task [31]. The key parameters which are required to implement such learning is extensive computational power and primary storage units. However, the problem of space and computational utilization is now solved with the introduction of Cloud Computing and GPU (Graphics Processing Unit) and even TPU (Tensor Processing Unit) [32].

A vehicle’s autonomy had always been a core problem in the sector. Experimental implementation of such project in the real-world scenario is not easy. Legal issues, various test cases, and situations has to be monitored closely which may or may not end up in a fatal accident causing both life and capital hazard. Therefore, for implementation of such project, simulated environment is a fare and dominant alternative. Due to the boom in the Video Game industry, the R&D in the sector of simulators have been increased [2]. There are several environment engines available that can be either directly utilized or can be transformed
and utilized as per our objective. CARLA, Unity, TORCS (The Open Racing Car Simulator), AirSim, Udacity (based on Unity) are the few open-source environments/simulators which could be utilized to test and implement the autonomous vehicles. The scope of this paper is to implement and test results of various autonomous vehicles models using Open-Source Simulator.

Earlier attempts to implement Autonomous Driving was the use of hard-core logics and using Deep/Machine Learning as the secondary objective handlers. Moreover, Reinforcement Learning as an option for autonomy uses too much computational resources which is just not feasible [13]. But in 2017, the paper of Nvidia changed the conventional approach by using a single CNN model as a regression model to attain self-driving [1]. Advantages of a CNN based model is that high-level features can be automatically considered without giving explicit instructions to the system. The only limitation of such deep-learning system is being the size of dataset used and efficient utilization of computational resources.

This paper, further classified in the different sections such as section II, discussed various researchers’ views in analyzing the implementation of self-driving via means of various techniques. Dataset used and model formulation is discussed in section III. Different matrices used are also discussed in section III. Further, the evaluation of results in terms of training and testing with regression matrices for models used is discussed in section IV. Next, this work is concluded in section V with its future scope.

2. Related Works

In [1], the authors proposed the study in which they empirically demonstrated the ability of CNNs to learn the entire driving task without manual decomposition and without marked lanes, semantic abstraction, path planning, and control (i.e., end-to-end steering control system). The study successfully resulted in the CNNs ability to operate a car in diverse weather condition including sunny, cloudy, rainy. In [2], the author described all the simulators including simulators for self-driving and implemented the simulator FODS (First Order Driving Simulator) and tested its performance based on rendering against simulators including Udacity, TORCS, Driving Interactions, OpenAI Gym (CarRacing-V0). The study resulted in the outperformance of FODS against all the 3-D and 2-D simulators based on mean and standard deviation of Rendering Steps (X1000) per minutes and achieved the score of 0.0372 for standard deviation and 3.48 for mean of rendering. In [3], the authors used 7.25 hours of driving video of 20Hz, along with the mapped dataset having steering angle, speed, GPS data to implement a CNN based system for lane keeping for self-driving cars that can automatically produce proper steering angles from captured frames.

In [4], implemented JacintoNet, a 13 layers deep CNN and used low power TDA2x SoC on an IOT device to implement end-to-end steering angle prediction for self-driving application, they also compared the performance of the trained network against the performance by VGG-16 network. In [5], the authors have explained in detail the implementation, working and application of peer-to-peer Client-Server Architecture and its application in real-world scenario. In [6], the authors have used two models including Attention Branch Network and CNN based regression model for steering and throttle for visually explaining the end-
to-end leaning for self-driving. They also combined the attention map with an FCLN that generates captions for specific regions. In [7], the authors used TORCS simulator to implement deep reinforcement learning framework using TensorFlow to control the car and get sensor values. They also used additional python library SCR (Simulated Car Racing) to access driving parameter controls and the system took 1.5 million states to converge for the desired output. In [8], the authors implemented 3 different CNN end-to-end steering control models and compared its performance with Nvidia model [1], with respect to convergence of loss and Lane Keep Time. The study resulted in Model 3 being more robust than Nvidia’s proposed CNN architecture model with lane keeping average time equals 617.3 seconds than Nvidia’s 453.7 seconds. In [9], the authors upgraded a pre-existing Unity based self-driving simulator CAIAS using Blenders and SolidWorks, added features like RBG camera, Lidar Sensor and simulated a new environment having sorts of weather conditions including rain, snow, fog, autumn and lanes including muddy, forest, and regular. Implemented Nvidia’s CNN model and scored behavior learning on parameters like obstacle avoidance, bumpy road tackle [1].

In [10], the authors did experiment on the combination of VGG-19 model for classification along with Nvidia end-to-end steering control system [1] model to detect traffic signal at the same time detecting steering angle and attained the accuracy score of 86% for signal. In [11], the authors developed a low cost IOT platform DeepPicar and trained it using Nvidia’s end-to-end steering control regression model [1]. Compared the processing time with respect to number of cores, and compared the performance of various micro computer platforms based on parameters like read-write speeds, average execution time and cost. In [12], the authors used an MPC and NMPC controllers backed up with RNN and a feed forward model for achieving the self-driving task and showcased the use of two computational modules to efficiently calculate the steering reference trajectory and RMS values respectively. In [13], the authors, showcased CNN based approach to classify the essential traffic signals and assigned them with wights to be used in reinforcement learning environment using Q-Learning algorithm. In [14], the authors studies about various parameters of reinforcement learning, presented empirical results to demonstrate disadvantages and advantages of transferring SFs between MDPs that only differ in rewards function and concluded that transferring a SF representation between tasks gives a significance boost in leaning speed.

In [15], the authors implemented autonomous-driving using Nvidia’s end-to-end steering control model [1] in Udacity CarND simulator, the study resulted in achieving the accuracy score of 78.5% on validation set. In [16], the authors experimented end-to-end autonomous driving in TORGS racing simulator with 6 different track data, used pre-trained CNN model to predict the desired steering wheel angle from continuous video stream and compared the performance based on mean-square-error between human and end-to-end controller. In [17], the authors developed a CNN via LSTM and fully connected layer based deep learning network to implement end-to-end steering control system for cooperative self-driving, they also argued for using V2V communication instead of GPS sensor data for making system more resistant to location-based errors. In [18], the authors used WPI and self-generated dataset for the sake of training heuristic ROI detection algorithm with Rttld implemented with RCNN, Yolo2, Yolo2-tiny, Yolo3, Yolo3-tiny, SSD, Faster RCNN models for traffic light detection. In [19], the author used Deepest LSTM-TinyPilotNet
for implementing end-to-end steering control system for autonomous vehicle, and calculated the error with respect to the center of the road and also achieved the root-mean-square value of 0.0912 radians. In [20], the authors used DRIVETM PX2 computer to implement a CNN facilitating end-to-end steering control system for a self-driving car and compared the results based on steering wheel angles deviation pre and post training and displayed number of lane departures in real-time scenario. In [21], the authors experimented with 3D structure-based regression methods instead of end-to-end learning for steering prediction to predict the failure cases for pose regression techniques. Used PoseNet, MapNet and DenseVLAD for pose regression and concluded that MapNet outperforms DenseVLAD on smaller LOOP datasets but performs poorly with larger full scene. In [23], the authors used capability CNN combined with sampling-based predictive control to predict a cost map projection in image plane and top-down view of the cost map and compared the results with GPS sensor output.

3. Method And Materials

The dataset used and the methodology used is explained in the subsequent sections.

3.1. Simulator

We used Unity based Udacity open-source simulator [2] as the base environment for implementing our self-driving car. The respective simulator makes tasks easier due to its client-server architecture [5] and easy to use API to collect and transmit data in and out of the simulator. The simulator has two modes, training and autonomous modes. While in training mode, we start the recording session to trigger collections of frames from 3 different cameras (left, center, right), based on time it also records and maps the output frames with parameters like speed, throttle, steering angle and brake. The autonomous mode uses client server architecture to communicate and transmit the data via an API pipeline, once the model trained, could use the real-time data flow stream from this mode and can transmit back the model results based on output to update the current parameters and situation of the autonomous vehicle. Figure 1 shows the used simulator for implementing self-driving.

3.2. Dataset

The dataset used for this work has been collected from Unity based Udacity self-driving simulator [2]. Our primary focus is to extract only the essential features for our deep-learning network. Images from left, center and right along with their mapping with steering angle is enough to implement our regression model, as other parameters could be adjusted using the logic programming and basic physics principles. One of the core problems with driving data collection is that it is never balanced. From the simulator we collected 12000 frames including the images and mapping from 3 different cameras. Due to the circular driving path of simulator, the graph which we get is either left or right skewed rather than being in normal distribution. A normal distribution can be equated as (1)

\[
p(x) = \frac{1}{\sigma \sqrt{2 \pi}} e^{-(x-\mu)^2/(2\sigma^2)} \tag{1}
\]
Moreover, due to the maximum inactivity of the steering angle, the data is more imbalanced and outlined near the center as seen in Fig. 2.

The solution to this problem, can either be to improve the data collection method in the simulator by developing environments which facilitates equally diverted roots and lanes or by augmenting (horizontal flipping) all the collected images and negating the output steering angle and adding them to our dataset file, which will give us, a more normally distributed curve. However, to limit the outlier on the center i.e., 0 radian and due to the maximum inactivity of the steering wheel, we created a special function based on a count or bin value of the steering wheel rotation to eradicate the outlier image dataset concentrated at center or steering inactivity. Figure 3 shows the corrected and normally distributed dataset.

### 3.3. Augmentation and Pre-processing

To increase the variety of input, the images were cropped to focus on the essential features and were randomly augmented with 50% probability including translation, zooming, and HSV adjustment. Images were converted into YUV format from RGB to make them more human visual appearing which has become the standard in implementation of autonomous driving. Figure 4 shows the image before and after pre-processing augmenting (horizontal flipping) all the collected images and negating the output steering angle and adding them to our dataset file, which will give us a more normally distributed curve. However, to limit the outlier on the 0 angles, we created a special function based on a binning technique to eradicate the unnecessary image dataset concentrated at center or steering inactivity. Figure 3 shows the corrected and normally distributed dataset.

Images were also blurred using Gaussian Blur which equates to Eq. (2).

\[
G(x, y) = \frac{1}{\sqrt{2\pi}\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \tag{2}
\]

### 3.4. Model Formulation

Peer to Peer steering control mechanism works in the manner depicted in Fig. 5. The frames captured from their respective cameras i.e., left, center and right along with its mapping steering angles in radian lying between −1 to 1, is horizontally stacked to normalize form and is fed into the deep-learning network respective to their pre-processing requirement. The CNN network computes the result and backpropagates the error value based on Mean Squared Error matrix and updates the weights and biases of the network for optimal results and better applicability of the network.

We are using two different network architectures for the implementation of a peer-to-peer steering control system for an autonomous driving vehicle including the ones from Nvidia’s [12] as shown in Fig. 6. The first 5 layers are convolutional fully connected layers followed by a dropout layer and then a flattened layer.
The network is ended using 3 layers of dense type and an output layer having ELU equated as (3) as activation function since the Elu output limits between −1 to +1 ELU holds the best interest of limits for the application. The network consists of a total of 252,219 trainable parameters.

The applicability and adaptability of MobileNet-V2 architecture is due to the bottleneck shown in Fig. 7. Due to its support for convolutional adaptability to reduce the number of features via a bottle necking, the network still is able to work in the most efficient way irrespective of reduced channels. The architecture shown in Fig. 8 has a depth of 88 layers and consists of around 3,538,984 trainable parameters. This classification network was converted into a regression-based model as shown in Fig. 7 for its adaptability for self-driving task by adding 4 additional dense layers at the end filtered via Elu activation function using Adagrad optimizer due to its lack of momentum and concentration to the features that are slow to change.

The Training was conducted into batches of 3000 each with various applied augmentation and preprocessing mentioned earlier. The dataset consists of around 24000 normally distributed and balanced frames. The results and trainability of different networks are discussed in the next section. While collecting the dataset from the simulated environment and implementing the model results in actual terrain faces a lot of ambiguous difficulties. Differences in obstacle density including crowd variations, signal variations, and lane variation and narrowing, one particular model cannot be universal. Therefore, ensemble of different models having deep layers and shallow layers is necessary for normalizing the performance output in the city and outside of the city environment. Deep layers of the model observe more features and train better in densely obstructed environment whereas shallow layered model observes less but more important/weighing features essential for the sparsely obstructed environment.

We have used the mean-square-error loss to train our model equated as (4). It is used to optimize the value of parameters used in our model. We intend to decrease the loss function with successive epochs. We have used Adam and Adagrad optimizers respectively with a learning rate = 0.001 for training our models.

\[
R(z) = \begin{cases} 
  z, & z > 0 \\
  \alpha e^x, & x \leq 0 
\end{cases} \quad (3)
\]

\[
L(Y - \hat{Y}) = \frac{1}{n} \sum_{i=1}^{n} (Y_i - \hat{Y_i})^2 \quad (4)
\]

4. Results And Discussion

In result analysis, model resulted steering angles have been compared against the actual result. Nvidia, MobileNet-V2 and Ensemble-M results are analyzed based on root-mean-square error matrices and average computation time per frame. The results were then contrasted to determine the best model. Although the model MSE are performing pretty good, we recommend using diverse but balanced data to enhance the performance using other simulators or real time driving image views. Due to the limitation of
the simulator to generate variety of data, the model is trained only on 24000 augmented frames collected from looped path. The performance and respective model the training is mentioned below.

4.1. Nvidia Model

The Nvidia model due to its limited depth and high tuning performs exceptionally well for most of the end-to-end steering angle predictions in the simulator. But for some features the predictions starts deviating from actual result. Figure 8 shows the training and validation loss (MSE) respectively during the training phase. The convolutional filter visualization of Nvidia model can be seen in Fig. 9.

4.2. MobileNet-V2 Model

There is decent convergence rate of a pretrained classification model converted into a regression model by adding a few additional support dense layers. MobileNet-V3 performs decently well for self-driving car application. Figure 10 shows the convergence of loss (MSE) over training and validation dataset during training phase and Fig. 11 shows the MobileNet-V2 model.

4.3. Ensemble-M

To make a peer-to-peer steering control system to be universally adaptable, a deeper and the model with more number of trainable parameters performs better in the terrain accommodating more features and a shallower model with less number of trainable features performs better in the environment with less frequent changes and features. An Ensemble-M approach shown in Fig. 11 normalizes the deviation (error) to the middle grounds by averaging the predicted regression results hence making the final result more converged towards the actual result. Therefore, the ensemble can be used effectively to solve this problem which equates to (5).

\[ Ensemble - M = \sum_{i=0}^{n} \frac{Y_i}{n} \quad (5) \]

Where \( Y \) is the output of the respective model, \( n \) is the total number of models used.

Table 1 shows the respective MSE scores, average computation time per frame, depth of individual model. Ensemble-M performs better than any other individual model and have attained an MSE score of 0.0201 (radians). However, the computational time per frame for NVIDIA model is less as compared with others with average compute time per frame equals 0.5087 seconds thus, verifying performance and time tradeoff in self-driving.

TABLE I

Comparison of Performance, depth and compute time
| Model        | MSE   | Depth | Average Compute Time per Frame |
|--------------|-------|-------|-------------------------------|
| Nvidia       | 0.0268| 11    | 0.5087                        |
| MobileNet-V2 | 0.0207| 88 + 5** | 0.5975                      |
| Ensemble-M   | 0.0201| 104   | 0.8514                        |

** Added layers for conversion of pre-trained model to regression model

Figure 12 shows the visualization of the performance of our models over testing dataset against actual regression results. For some parts of predicted regression, the results from Nvidia model lies near the actual result and for other parts of regression prediction, the results from MobileNet-V2 lie near the actual results. Due to the fact that a shallow model gets trained faster and is adaptable for most of the results still, at some complex points with a greater number of features its predictions do not coincide with the requirements. Hence our deeper model solves this problem by feeding the input into more deeper layers for prediction. The importance of ensemble techniques plays a significant role in limiting the predicted output. Hence, Ensemble-M makes it normalized therefore making autonomous-driving safer due to its normalized deviation.

5. Conclusion And Future Scope

This study only covered the ensemble-M approach on 2 models respectively. Future work involves the verification of the ensemble-M on more than two models for achieving the least RMS score and hence making end-to-end steering control system more feasible than what its status for implementation is today. There is a plethora of techniques to implement full-fledged autonomous driving. Methods including deep reinforcement learning, logic programming, deep learning (road segmentation, end-to-end) are a few of the methods to achieve the tasks. Irrespective of the technique used, there will always be a time-space trade-off. A deeper network tuned with correct parameters may lead to a better MSE score but at the same time, it may lead to a more time-consuming process. On the other hand, a shallow deep network may not have a good MSE score but performs comparatively enough required by the application. Since autonomous driving involves the risk of life, this trade-off has to be balanced on some middle ground. Since this project was implemented in the simulated environment, there will always be the possibility of missing features in the simulator which may have affected the performance and decision making of the model. Therefore, the development of a versatile simulator with custom feature adjustment is always a possibility. Moreover, the implementation of such a system in the real-life environment will also require enhanced security for data transmission facilitating the client-server architecture. Work on system versatility is also required since the system's adaptability is an issue that changes from country to country based on local traffic rules and variation.

Declarations

Ethical Approval: No agency or research Centre is involved in this research
Funding Statement: This research received no external funding.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

Informed Consent: The authors declare that they do not have any conflict of interest. This research does not involve any human or animal participation. All authors have checked and agreed the submission.

Authors Contributions: All authors conceived and design the study. Meenu Gupta and Vikalp upadhyay conducted the experiments, analyzed the data and wrote the paper. All authors contributed to manuscript revision. All authors approved the final version of manuscript and agree to be held accountable for the content therein.

References

1. Bojarski M, Del Testa D, Dworakowski D, Firner B, Flepp B, Goyal P, … Zieba K (2016) End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316
2. Hsieh W (2017) First order driving simulator (Doctoral dissertation, Master’s thesis, EECS Department, University of California, Berkeley)
3. Chen Z, Huang X (2017, June) End-to-end learning for lane keeping of self-driving cars. In 2017 IEEE Intelligent Vehicles Symposium (IV) (pp. 1856–1860). IEEE
4. Viswanath P, Nagori S, Mody M, Mathew M, Swami P (2018, September) End to end learning based self-driving using JacintoNet. In 2018 IEEE 8th International Conference on Consumer Electronics-Berlin (ICCE-Berlin) (pp. 1–4). IEEE
5. Oluwatosin HS (2014) Client-server model. IOSRJ Comput Eng 16(1):2278–8727
6. Mori K, Fukui H, Murase T, Hirakawa T, Yamashita T, Fujiyoshi H (2019, June) Visual explanation by attention branch network for end-to-end learning-based self-driving. In 2019 IEEE Intelligent Vehicles Symposium (IV) (pp. 1577–1582). IEEE
7. Chopra R, Roy SS (2020) End-to-End Reinforcement Learning for Self-driving Car. In: Advanced Computing and Intelligent Engineering. Springer, Singapore, pp 53–61
8. Wang Y, Liu D, Jeon H, Chu Z, Matson ET (2019) End-to-end Learning Approach for Autonomous Driving: A Convolutional Neural Network Model. In ICAART (2) (pp. 833–839)
9. Hossain S, Fayjie AR, Doukhi O, Lee DJ (2018, October) CAIAS simulator: self-driving vehicle simulator for AI research. In International Conference on Intelligent Computing & Optimization (pp. 187–195). Springer, Cham
10. Babiker MA, Elawad MA, Ahmed AH (2019, September) Convolutional Neural Network for a Self-Driving Car in a Virtual Environment. In 2019 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE) (pp. 1–6). IEEE
11. Bechtel MG, McEllhiney E, Kim M, Yun H (2018, August) Deeppicar: A low-cost deep neural network-based autonomous car. In 2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA) (pp. 11–21). IEEE

12. Garimella G, Funke J, Wang C, Kobilarov M (2017, September) Neural network modeling for steering control of an autonomous vehicle. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 2609–2615). IEEE

13. Chishti SO, Riaz S, BilalZaib M, Nauman M (2018, November) Self-driving cars using CNN and Q-learning. In 2018 IEEE 21st International Multi-Topic Conference (INMIC) (pp. 1–7). IEEE

14. Lehnert L, Tellex S, Littman ML (2017) Advantages and limitations of using successor features for transfer in reinforcement learning. arXiv preprint arXiv:1708.00102

15. Smolyakov MV, Frolov AI, Volkov VN, Stelmashchuk IV (2018, October) Self-driving car steering angle prediction based on deep neural network an example of CarND udacity simulator. In 2018 IEEE 12th International Conference on Application of Information and Communication Technologies (AICT) (pp. 1–5). IEEE

16. Qiu T, Huang Z (2019, April) Learning A Steering Decision Policy for End-to-End Control of Autonomous Vehicle. In 2019 5th International Conference on Control, Automation and Robotics (ICCAR) (pp. 347–351). IEEE

17. Valiente R, Zaman M, Ozer S, Fallah YP (2019, June) Controlling steering angle for cooperative self-driving vehicles utilizing cnn and lstm-based deep networks. In 2019 IEEE Intelligent Vehicles Symposium (IV) (pp. 2423–2428). IEEE

18. Ouyang Z, Niu J, Liu Y, Guizani M (2019) Deep CNN-based real-time traffic light detector for self-driving vehicles. IEEE transactions on Mobile Computing 19(2):300–313

19. del Egio J, Bergasa LM, Romera E, Huélamo CG, Araluze J, Barea R (2018) November). Self-driving a Car in Simulation Through a CNN. In: Workshop of Physical Agents. Springer, Cham, pp 31–43

20. Jhung J, Bae I, Moon J, Kim T, Kim J, Kim S (2018, June) End-to-end steering controller with cnn-based closed-loop feedback for autonomous vehicles. In 2018 IEEE Intelligent Vehicles Symposium (IV) (pp. 617–622). IEEE

21. Sattler T, Zhou Q, Pollefeys M, Leal-Taixe L (2019) Understanding the limitations of cnn-based absolute camera pose regression. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 3302–3312)

22. Da Conceicao MA, Degadwala S (2020) Steering Angle Prediction Based on Road Direction using Convolution Neural Network (CNN)

23. Drews P, Williams G, Goldfain B, Theodorou EA, Rehg JM (2017) Aggressive deep driving: Model predictive control with a cnn cost model. arXiv preprint arXiv:1707.05303

24. Alexeev VF, Staravoitau AI, Piskun GA, Likhacheuski DV (2018) End to end learning for a driving simulator. Доклады Белорусского государственного университета информатики и радиоэлектроники, (2 (112))

25. Urmson C (2008) Self-driving cars and the urban challenge. IEEE Intell Syst 23(2):66–68
26. Daily M, Medasani S, Behringer R, Trivedi M (2017) Self-driving cars. Computer 50(12):18–23
27. Paden B, Čap M, Yong SZ, Yershov D, Frazzoli E (2016) A survey of motion planning and control techniques for self-driving urban vehicles. IEEE Transactions on intelligent vehicles 1(1):33–55
28. Chen C, Demir E, Huang Y, Qiu R (2021) The adoption of self-driving delivery robots in last mile logistics. Transportation research part E: logistics transportation review 146:102214
29. Ni J, Chen Y, Chen Y, Zhu J, Ali D, Cao W (2020) A Survey on Theories and Applications for Self-Driving Cars Based on Deep Learning Methods. Applied Sciences 10(8):2749
30. Goodall NJ (2016) Can you program ethics into a self-driving car? IEEE Spectr 53(6):28–58
31. Zeng W, Wang S, Liao R, Chen Y, Yang B, Urtasun R (2020, August) Dsdnet: Deep structured self-driving network. In European Conference on Computer Vision (pp. 156–172). Springer, Cham
32. Ohn-Bar E, Trivedi MM (2016) Looking at humans in the age of self-driving and highly automated vehicles. IEEE Transactions on Intelligent Vehicles 1(1):90–104
33. Smith RG, Eckroth J (2017) Building AI applications: Yesterday, today, and tomorrow. AI Magazine 38(1):6–22
34. Wang YE, Wei GY, Brooks D (2019) Benchmarking tpu, gpu, and cpu platforms for deep learning. arXiv preprint arXiv:1907.10701
35. Xia H, Yang H (2018) Is last-mile delivery a ‘killer app’ for self-driving vehicles? Commun ACM 61(11):70–75
36. Olaverri-Monreal C (2020) Promoting trust in self-driving vehicles. Nature Electronics 3(6):292–294
37. Savelsbergh MW, Sol M (1995) The general pickup and delivery problem. Transportation science 29(1):17–29
38. Chen SC (2019) Multimedia for autonomous driving. IEEE MultiMedia 26(3):5–8
39. Badue C, Guidolini R, Carneiro RV, Azevedo P, Cardoso VB, Forechi A, … De Souza AF (2020) Self-driving cars: A survey. Expert Systems with Applications, 113816
40. Ndikumana A, Tran NH, Kim KT, Hong CS (2020) Deep learning based caching for self-driving cars in multi-access edge computing. IEEE Transactions on Intelligent Transportation Systems
41. Shalamov V, Filchenkov A, Shalyto A (2016, September) Genetic search of pickup and delivery problem solutions for self-driving taxi routing. In IFIP International Conference on Artificial Intelligence Applications and Innovations (pp. 348–355). Springer, Cham
42. Ali Alheeti KM, Gruubler A, McDonald-Maier K (2016) Intelligent intrusion detection of grey hole and rushing attacks in self-driving vehicular networks. Computers 5(3):16
43. Faas SM, Baumann M (2019, November) Light-based external Human Machine Interface: color evaluation for self-driving vehicle and pedestrian interaction. In Proceedings of the Human Factors and Ergonomics Society Annual Meeting (Vol. 63, No. 1, pp. 1232–1236). Sage CA: Los Angeles, CA: SAGE Publications
44. Pearah PJ (2017) Opening the Door to Self-Driving Cars: How Will This Change the Rules of the Road. J High Tech L 18:38
45. Blyth PL, Mladenovic MN, Nardi BA, Ekbia HR, Su NM (2016) Expanding the design horizon for self-driving vehicles: Distributing benefits and burdens. IEEE Technol Soc Mag 35(3):44–49
46. West DM (2016) Moving forward: self-driving vehicles in China, Europe, Japan, Korea, and the United States. Center for Technology Innovation at Brookings: Washington, DC, USA
47. Brooks R (2017) The big problem with self-driving cars is people. IEEE Spectrum: Technology, Engineering, and Science News

Figures

Figure 1

Densely Featured, Sparsely Featured and Ariel Views of Simulator

Figure 2

Raw imbalanced dataset collected from simulator having concentration on a few values due to inactivity of steering most of the time.
**Figure 3**

Balance dataset after conversion, processing, and augmentation based on normal distribution, equation (1) binning limited to 600.

**Figure 4**

Image after and before pre-processing, the conversion of image from RGB color scheme to YUV color scheme.
Figure 5

End-to-End Steering Control Mechanism.
Figure 6

Network Architecture of Nvidia End-to-End steering control CNN model [1].
Figure 7

Bottleneck Architecture of MobileNet-V2 along with the Processing for Autonomous-Driving Application

Figure 8

Training and Validation loss score per epoch for Nvidia Model.
Figure 9

Visualization of output based on convolutional filters of Nvidia Model

![Loss Graph](image)

Figure 10

Training and Validation loss based on MSE score for End-to-End Steering Control Mechanism for MobileNet-V2
Figure 11

Training and Validation loss based on MSE score for End-to-End Steering Control Mechanism for MobileNet-V2

Figure 12

Deviation comparison of Nvidia, MobileNet-V2 and Ensemble-M model against the actual result.