Hindawi
Modelling and Simulation in Engineering
Volume 2022, Article ID 5716820, 11 pages
https://doi.org/10.1155/2022/5716820

Research Article

A Lightweight Convolutional Neural Network to Predict Steering Angle for Autonomous Driving Using CARLA Simulator

Imtiaz Ul Hassan,[1] Huma Zia,[2] H. Sundus Fatima,[1] Syed Adnan Yusuf,[3] and Muhammad Khurram[1]

[1]*Smart City NCAI, NED University of Engineering and Technology, Karachi, Pakistan*
[2]*College of Engineering, Abu Dhabi University, Abu Dhabi, UAE*
[3]*Research and Development Department, Intelexica Pvt. Ltd., Southampton SO152RZ, UK*

Correspondence should be addressed to Huma Zia; huma.zia@adu.ac.ae

Received 31 March 2022; Accepted 29 July 2022; Published 23 August 2022

Academic Editor: Noé López Perrusquia

End-to-end learning for autonomous driving uses a convolutional neural network (CNN) to predict the steering angle from a raw image input. Most of the solutions available for end-to-end autonomous driving are computationally too expensive, which increases the inference of autonomous driving in real time. Therefore, in this paper, CNN architecture has been trained which is lightweight and achieves comparable results to Nvidia's PilotNet. The data used to train and evaluate the network is collected from the Car Learning to Act (CARLA) simulator. To evaluate the proposed architecture, the MSE (mean squared error) is used as the performance metric. Results of the experiment shows that the proposed model is 4x lighter than Nvidia's PilotNet in term of parameters but still attains comparable results to PilotNet. The proposed model has achieved 5.1×10^{-4} MSE on testing data while PilotNet MSE was 4.7×10^{-4}.

1. Introduction

Research on autonomous vehicles (AV) dates back to the last century when the prototype of a radio-controlled autonomous vehicle was designed in the 1920s [1]. Since that day, automotive engineers, scientists, and researchers have done extensive research in AV's. Moreover, this research has gotten a lot of attention due to the recent development in computing power and artificial intelligence. Furthermore, automotive companies and academic institutions like Tesla, Apple, Nissan, Audi, Stanford University, Carnegie Mellon University, and MIT are playing critical roles in the research on AV [2]. The three AV competitions organized by the Defense Advanced Research Project Agency (DARPA) in 2004, 2005, and 2007, respectively, marked the beginning of a new era in AV research [3]. As in the second competition, 4 out of 23 finalists successfully completed 132 miles of road passing through mountains, with tunnels and sharp left and right turns [3]. Likewise, in the Third DARPA competition, 6 out of 11 finalists completed a 60-mile route in the urban environment following traffic rules. Subsequently, the current autonomous driving system used by most of the DARPA challenge participants is based on a modular approach that consists of sensor inputs, environment perception module, path planning module, and control module as shown in Figure 1. Sensors like cameras, Lidar, and radar are used to sense the world around them.

The perception module receives sensor input and uses it to make sense of the world around the vehicle. The path planning module uses the information passed by the perception module to generate way-points to be followed. Consequently, the control module ensures these way-points are followed by creating control values, i.e., throttle and steering.

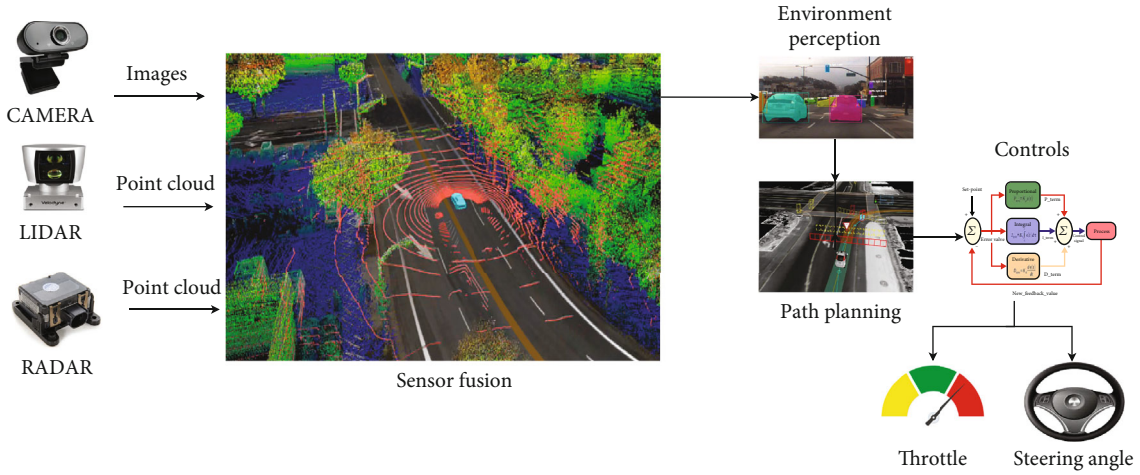

Figure 1: Traditional approach toward AV.

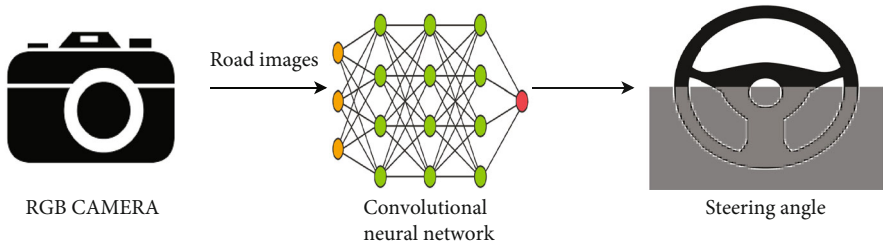

Figure 2: End-to-end learning for AV.

In addition to that, another approach toward autonomous driving is known as end-to-end learning, which trains a CNN on raw red green blue (RGB) images to predict the steering angle required to keep the vehicle on the road as depicted in Figure 2. Not only are RGB cameras cheap but they also provide rich information about the environment [4]. The end-to-end learning model learns the input features from the road images taken from the RGB camera, which can generate steering commands to drive the car without human interference.

Overall, most of the work done in the domain of end-to-end learning relies on deep learning models that are computationally expensive and designed to run high processing graphical processing units (GPUs) [4, 5]. To save the computational resources and keep the overall cost of AV low, lightweight architecture is required, which can run on low computing devices with minimum inference time. As a result, this work focuses on building lightweight architecture to achieve the task of end-to-end learning by predicting the steering values from RGB images. The MSE of the proposed model was 5.1×10^{-4}. The proposed architecture has achieved almost the same results as Nvidia's PilotNet [5] but is 4 times lighter in terms of parameters. The data to train and evaluate the architecture was obtained from different towns of CARLA simulators under other weather conditions using CARLA-ROS-Bridge. The rest of the paper has been organized as follows. Section 2 represents a literature review of research in the field of end-to-end learning. Section 3 provides implementation details. In contrast, Section 4 discusses the results, followed by a conclusion in Section 5.

2. Literature Review

The availability of high-power GPU's and bulk amount of data generated across the internet has led to significant development in robust deep learning algorithms. CNNs is one of the deep learning models that are used on visual imagery. Meanwhile, the ImageNet scale Visual Recognition Challenge [6] turned out to be a milestone for computer vision as this competition resulted in the development of state-of-the-art CNN architectures, like AlexNet [7], VGG16 [8], and GoogleNet [9], to name a few. Henceforth, convolutional networks have been used in many sectors, including healthcare, agriculture, robotics, and automotive industry. Moreover, recently, deep learning has achieved astonishing development like automated machine learning [10], playing the game of Go and Chess [11, 12] and end-to-end learning [13, 14].

End-to-end learning is aimed at solving complex systems by training a single deep learning model, a CNN to be specific, bypassing intermediate layers. There has been extensive research done in the domain of end-to-end learning for AV. The subject of autonomous driving requires intensive study and research. It has been observed that different modality approaches have been implemented at advancement levels. The first work was done by Pomerleau. The concept was

initiated based on the concept of the neural network three-layer architecture with a single convolution layer which allowed vehicles to trace and track the vehicle movement in a particular direction [15], who used a simple three-layer neural network architecture with a single hidden layer which was able to find directions for the vehicle to follow [16] and used a six-layered CNN, which was trained to control the robot using radio signals and attained a max speed of 2 m/s. Similarly, for saving the computational resources and keeping the overall cost of AV low, lightweight architecture is required, which can run on low computing devices with minimum inference time. Most of the solutions available for end-to-end autonomous driving are computationally too expensive, which increases the inference of autonomous driving in real time. As a result, this work focuses on building lightweight architecture to achieve the task of end-to-end learning by predicting the steering values from RGB images. The DARPA autonomous vehicle (DAVE) was based on end-to-end learning. Moreover, in the simulation of physics, rendering of the output sensors is also controlled by the server side. On the other hand, the client side is responsible for the client side initialization of scenarios and control actors. CARLA provides a model of every sensor which is used by AVs including Lidar, radar, camera, and IMU. The parameters of all these sensors can be configured. CARLA also provides the flexibility of creating custom sensors, DAVE used a CNN with six layers to predict the necessary direction to be taken by the vehicle [17]. DAVE was able to drive around an alley full of junk, avoiding obstacles.

Recently, Nvidia [5] has developed a CNN architecture PilotNet, which was trained on RGB images using a left, right, and central camera. The PilotNet successfully achieved lane-following in simple real-world scenarios. PilotNet was able to drive on road with and without lane markings. Moreover, it was also able to drive on unpaved roads and even on parking lots.

[18] modified the approach taken by PilotNet and added throttle prediction to the system. [19] fed image input from and front-facing camera as well as turning information i.e., indicator signals at junctions, to a CNN named 100 DriveNet. CARLA also provides the functionality to change the weather and time of day. ROS bag files to store the road images for every timestamp which were published to CARLA Ego vehicle RGB topic while the corresponding steering values were published under CARLA Ego vehicle Status topic. CARLA basically provides a model of every sensor which is used by AVs including Lidar, radar, camera, and IMU. The parameters of all these sensors can be configured. DriveNet was evaluated in the real world and was able to avoid obstacles dynamically

Following PilotNet, Chen et al. [4] used the comma.ai dataset to train a CNN to predict the steering angle, which was implemented using Caffe [20]. The model was evaluated using mean squared error, and the testing MSE was 2.42 degrees. [21] applied different techniques, including 3D CNN, LSTM, and ResNet, with a minimum RMSE of 0.00709 achieved by ResNet50.

Moreover, the network was also successful in taking turns at intersections. The demonstrations design trained on deep neural networks specifically for the ease of drivers, i.e., the self-driving for the eradication of the human dependency for training has indicated pronounced effects through the training and learning of the models with the certainty in route following and road hindrance avoidance [22], whereas the policies based on which the training has been conducted imitated learning techniques that were uncontrollable at the time of testing. Imitation learning is basically providing promising outcomes on the approach of training [23]. The results of the testing have indicated that the implemented policy for driving has a continuous response to the commands given by the navigator. The experiment was undertaken on the three-dimensional simulators of urban driving. However, the study has not addressed the usage of natural language for the guidance of the human in the autonomous vehicle, because the technique which was implemented is based on the vision driving of the robotic vehicle [24]. Proposing a lightweight model with low computational and memory requirements is quite a challenging task [22–25]. Overall, most of the solutions presented in the literature are computationally expensive, requiring high computing devices like GPUs and TPU's. For this reason, we have propose a lightweight convolutional network for end-to-end autonomous driving.

We have used the CARLA simulator [26] to collect the data as it represents the real world most closely as other simulators with available urban layouts, buildings, and vehicles. ROS [27] is an open-source collection of software frameworks that are used for robot software management. Integrating CARLA with the ROS environment provides reproducibility in the real world without any significant modifications.

3. Implementation Details

This section provides a brief overview regarding the implementation of lightweight CNN for end-to-end AV. In the details about the training setup, CARLA simulator environment is discussed which is followed by data collection and data preprocessing details. Finally, we present the architectural details of the proposed model followed by the discussion about PilotNet.

3.1. End-to-End Driving Training Setup. In end-to-end learning, the whole driving task is treated as one single instead of breaking into different modules, unlike the traditional method where the task is explicitly decomposed into different modules. A neural network (NN) is trained using input images from a front-facing camera. The NN instinctively learns the necessary road features and their relation with the steering angle provided for the training dataset. We mounted the RGB camera available in CARLA and used CARLA-ROS-Bridge in order to extract the steer values and the RGB images. These images and steer values were used to train a CNN. The MSE is used to evaluate the model on test images which the model has never seen before. The training setup can be seen in Figure 3.

The working of CARLA is based on the server-client-based system. Based on the unreal engine the server side controls the environment (town maps, weather, etc.) and actors (cars, pedestrians, and traffic lights). Moreover, in the simulation of physics, rendering of the output sensors are also controlled by the server side. On the other hand, the client side is responsible for the client side initialization of scenarios and control

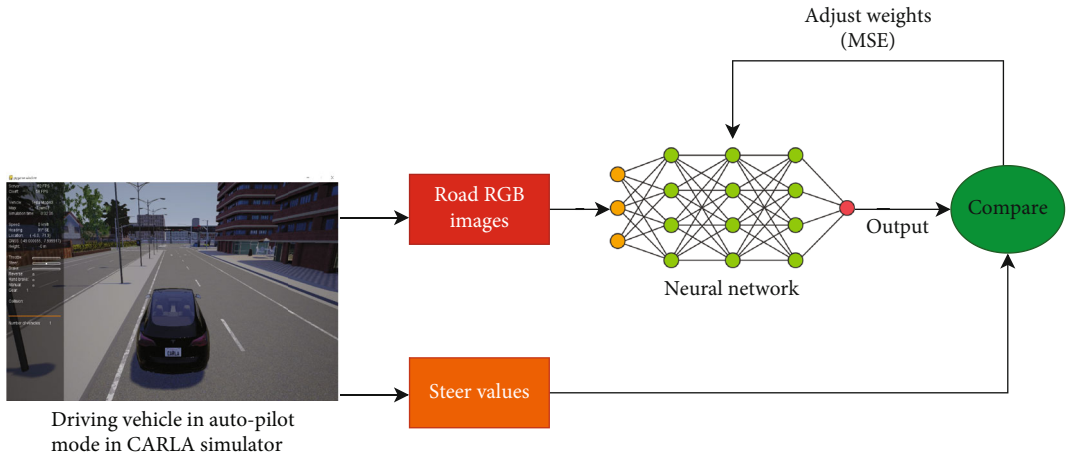

Figure 3: End-to-end training setup.

Figure 4: Four different towns of CARLA.

actors. CARLA provides a model of every sensor which is used by AVs including Lidar, radar, camera, and IMU. The parameters of all these sensors can be configured. Moreover, CARLA also provides the flexibility of creating custom sensors. There are different towns available in CARLA, and we have used four of them to collect data Figure 4. Furthermore, CARLA also provides the functionality to change the weather and time of day. Different weather conditions of the same road are provided in Figure 5.

We used ROS bag files to store the road images for every timestamp which were published to CARLA Ego vehicle RGB topic while the corresponding steering values were published under CARLA Ego vehicle status topic. The images and steer values were extracted from ROS bag files using python script. The images were rescaled from 800×600 to 220×220.110. The dataset contained a total of 172982 which were resampled to 86491 images with corresponding steering angles.

3.2. Data Preprocessing. Two versions of the dataset with and without the region of interest (ROI) were prepared. The dimensions of images with ROI cropped were $110 \times 220 \times 3$, while the original images had dimensions of $220 \times 220 \times 3$. Images with and without ROI cropped can be seen in Figure 6. Next, all the images were normalized using the min-max scaling Equation (1), a normalization technique. Normalization ensures that every image has the same distribution, which helps in the faster convergence of neural networks.

$$x_{\text{norm}} = \frac{x - x_{\text{min}}}{255}. \tag{1}$$

Clear Noon

Wet Noon

Hard Rain Sunset

Cloudy Sunset

Figure 5: Same road under 4 different weather conditions.

3.3. Data Splitting. After normalization, we randomly split the input data into 64% training, 16% validation, and 20% testing, respectively. The validation data is used to tune hyperparameters, i.e., the number hidden layers and the number of neurons in those layers. Moreover, validation data is also used to decide when to stop the training.

3.4. Network Architecture Overview. The proposed lightweight architecture was trained and tested with different layers, kernel sizes, loss functions, and activation functions. The aim was to choose an optimal architecture that achieves high accuracy and is lightweight. After many experiments, the proposed architecture was chosen. Figure 7 represents the architectural details of the proposed model. The model consists of 4 convolutional layers, each followed by a max-pooling layer. The last max-pooling layer is followed by a flatten layer and two fully connected layers. The final fully connected layer serves as a controller. The first layer of the model consists of a 2D convolutional layer with 64 filters each of size 3×3. This layer serves as a feature extractor from

Without ROI cropped Roi Cropped

Figure 6: Cropping region of Interest.

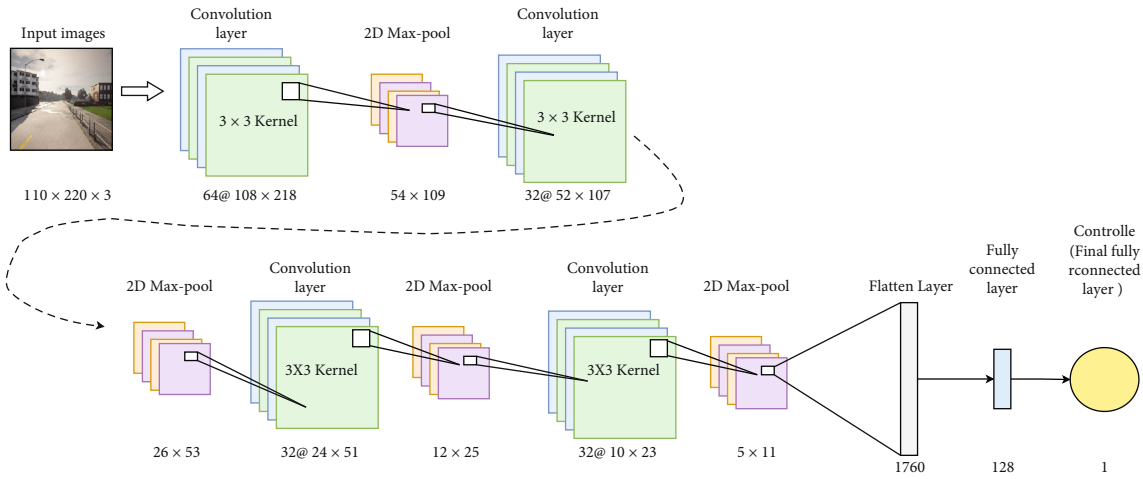

Figure 7: Network architecture.

the raw input images. Equation (2) [28] represents the 2D convolution performed by this layer.

$$s(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n), \tag{2}$$

where $s(i,j)$ represents the output of a convolutional layer, which is a 2D feature map tensor. The dimensions of the feature map, i.e., the output of 2D convolution can be calculated using Equations (3) and (4) [28], respectively.

$$W\text{out} = W - F + 2PS + 1, \tag{3}$$

$$H\text{out} = H - F + 2PS + 1, \tag{4}$$

where W and H represent the height and width of the input to the convolutional layer F the kernel/filter size, P padding, and S strides. In our case, W and H were 110 and 220, respectively, which is the dimension of input pics after cropping the region of interest. The padding value P was set to 0, stride S was 1, and filter F was 3. Hence width $Wout$ and Height $Hout$ of the output layer were 108 and 218, respectively, which can be calculated from Equations (3) and (4). The number of filters used in this layer was 64. Equation (5) has been used to find the total number of trainable parameters for the convolutional layer, where n and m represent the height and width of the filter used in the convolutional layer and l represent the depth of input dimension which is 3 for the first layer as the image contains 3 channels. K represents the number of filters used which is 64. So, the total number of trainable parameters for the convolution layer calculated using Equation (5) [28] was 1792.

Relu was used as an activation function for this layer. The Relu function shown in Equation (6) is not only computationally cheaper than the tan h and Sigmoid functions but also unaffected by the gradient vanishing effects in Sigmoid and tan h due to activation nonlinearities [29].

$$N\text{parameters} = (n * m * l + 1)k, \tag{5}$$

$$\text{Relu}(x) = \max(0, x). \tag{6}$$

The convolutional layer was followed by a max-pooling layer with a 2×2 dimension. This layer reduces the output

Table 1: Results of training with and without ROI cropped.

Data	Validation MSE	Testing MSE	Trainable parameters
With ROI cropped	5.3×10^{-4}	5.1×10^{-4}	264289
Without ROI cropped	5.5×10^{-4}	5.7×10^{-4}	534625

of the convolutional layer by half so the resultant dimension was 54×10^9. The max-pooling layer reduces the computational complexity by reducing the number of training parameters while retaining spatial information. This layer is followed by another 2D convolutional layer with a 3×3 filter size with a total number of 32 filters. This layer has 18464 trainable parameters. The input dimension of this layer is $52 \times 107 \times 32$. This layer was followed by the second max-pooling layer of 2×2 reducing the output to $26 \times 53 \times 32$. This layer is followed by two other convolutional layers. Both of these convolutional layers are followed by max-pooling layers of size 2×2. Both of these convolutional layers have 9248 trainable parameters.

The last max-pooling layer was followed by a flattened layer which converts the two-dimensional feature map into a one-dimensional vector. The flattened layer is followed by a fully connected layer with 128 neurons. The number of parameters in this layer is 225408. This was followed by the final fully connected layer with a single neuron having 129 trainable parameters. As this layer acts as the controller and predicts the steering values which lie between -1 and+1, therefore, two activation functions, i.e., linear function and hyperbolic tangent $\tan h$ Equation (7) [30] were chosen. The linear function's output ranges from (−, +), while $\tan h$ has a range over (−1,+1).

$$\tanh(x) = \frac{2}{1 + 2e^{-2x}} - 1. \tag{7}$$

Mean squared error 8 and mean absolute error (MAE) 9 [31] were used with Adam optimizer to train the architecture separately with different combinations of the activation layer. MAE is not affected by outliers, while MSE penalizes large errors significantly more than small ones.

$$\begin{aligned} \text{MSE} &= \frac{1}{n}\sum_{0}^{n-1}(y - \bar{y})^2. \\ \text{MAE} &= \frac{1}{n}\sum_{0}^{n-1}|y - \bar{y}|. \end{aligned} \tag{8}$$

4. Results and Discussion

The proposed model was implemented in Keras. The Kaggle kernel was used which provides the Nvidia P100 GPU with 16 GB memory. The input data contained 86491 images each of size $220 \times 220 \times 3$ with taking up 7.8 GB of space. After cropping out the ROI, the resolution was reduced to $110 \times 220 \times 3$ taking up 3.5 GB of hard disk space.

4.1. Training with and without Cropping ROI. Cropping ROI not only decreased the space requirements of the input data but also decreased the number of trainable parameters required for training. Furthermore, the mean squared error was also decreased as ROI cropping removed noise from the input data which was caused by the building and sky. The number of trainable parameters for the model with ROI cropped images is 264289 while the model without ROI cropped images was 534625 which is double the number of parameters of the ROI cropped model. Also from Table 1, it is evident that both validation and testing MSE for ROI cropped images is better than the one without ROI cropped. The validation loss of training for both datasets can be seen in Figure 8. The training was stopped using Keras early stopping call backs as we can see that after the 10 epochs that the validation loss for model trained on data without ROI cropped increases which suggests that the model is overfitting. On the other hand, the validation loss for the model with ROI cropped images goes on decreasing which suggests the model learns better on this data.

4.2. Training with Different Loss and Activation Functions. The given architecture was selected after extensive trial and error methods Once the architecture was selected, we experimented with different loss and different activation functions for the final fully connected layer which outputs the steering values. The results obtained after the combination of different activation and loss functions can be seen in Table 2. As evident from the table, that architecture with linear activation and MSE loss function gives the best results. In the next section, we compare it to other pretrained models. The validation means squared error for training under different combinations of activation and loss function can be seen in Figure 9. The model with the lowest validation MSE was saved using Keras checkpoints.

4.3. Proposed Model Comparison with PilotNet. The computational cost of a neural network depends on the number of parameters it has. Multilayer perceptrons consist of fully connected layers in which every neuron in one layer is connected to every neuron in the succeeding layer. However, this is not the case with CNNs where each neuron is only connected to its neighbor neurons from the previous layers. Moreover, the same set of weights is shared by every neuron.

Hence, the computational complexity of a convolutional layer depends upon the number of computations done by the filter elements. Therefore, to have a quantitative comparison of computational complexity between PilotNet Figure 10 and the proposed network, the total number of trainable parameters were compared. Moreover, the total number of parameters depends upon the number of convolutional layers,

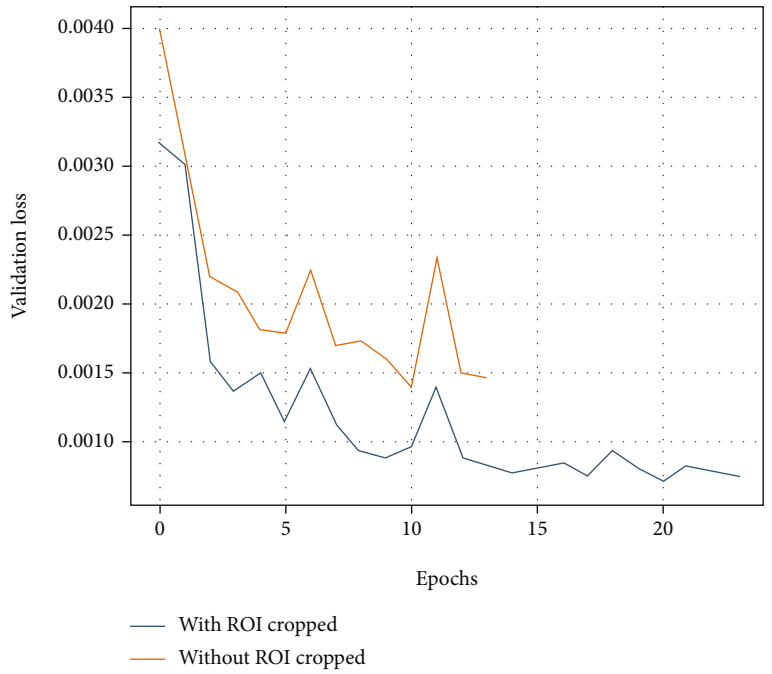

Figure 8: Validation loss for models trained with and without ROI cropping.

Table 2: Testing and validation loss under different loss and activation functions.

Loss and activation function	Testing MSE	Validation MSE
MSE loss and linear activation	5.1×10^{-4}	5.4×10^{-4}
MSE loss and tan$h$ activation	6.4×10^{-4}	6.6×10^{-4}
MAE loss and linear activation	8.7×10^{-4}	8.6×10^{-4}
MAE loss and tan$h$ activation	6.6×10^{-4}	5.9×10^{-4}

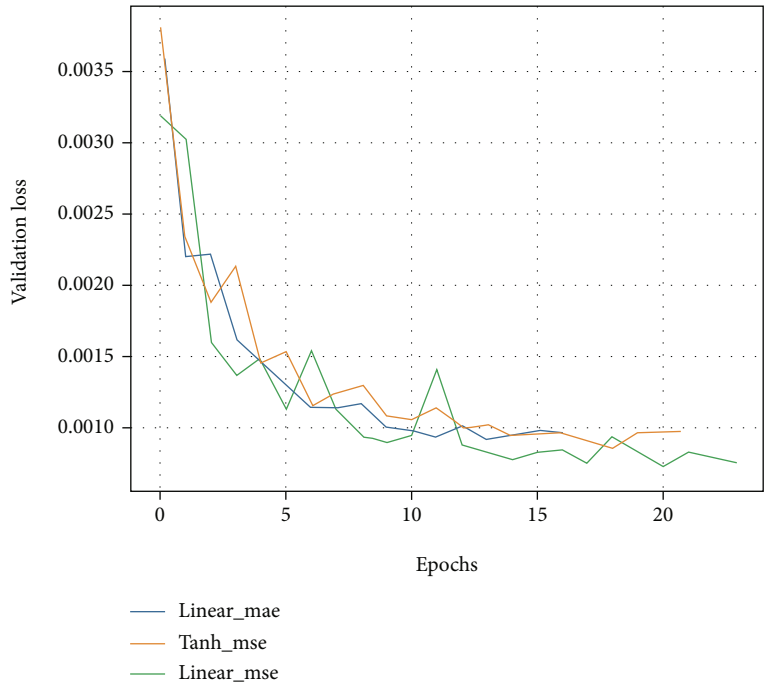

Figure 9: Validation for different loss and activation functions.

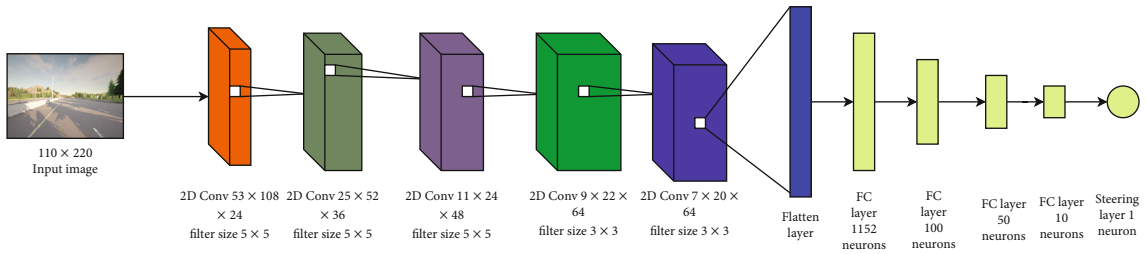

Figure 10: Modified PilotNet architecture.

Table 3: Test mean squared error and parameters comparison between proposed architecture and PilotNet.

Model	Testing MSE	Parameters	Inference time (sec)
PilotNet	4.7×10^{-4}	1.0575291×10^{7}	0.042
Proposed architecture	5.1×10^{-4}	2.64289×10^{5}	0.036

Table 4: Test root mean squared of proposed model on various weather conditions.

Weather	Testing MSE
ClearNoon	7.3×10^{-4}
CloudyNoon	5.9×10^{-4}
WetNoon	4.7×10^{-4}
WetNoon	6.5×10^{-4}
MidRainyNoon	8.3×10^{-4}
HardRainNoon	4.2×10^{-4}
SoftRainNoon	4.4×10^{-4}
ClearSunset	5.7×10^{-4}
CloudySunset	3.6×10^{-4}
WetSunset	3.69×10^{-4}
WetCloudySunset	4.7×10^{-4}
MidRainSunset	6.5×10^{-4}
HardRainSunset	3.3×10^{-4}
SoftRainSunset	4.7×10^{-4}

size of filters, number of max-pooling layers, and number of neurons in fully connected layers. Table 3 represents a comparison between the proposed network and PilotNet in terms of testing mean squared error, a total number of parameters, and inference time taken by the model on a single image. Comparing the total number of parameters, we can see that the proposed architecture has 4 times fewer parameters than PilotNet. Also, the proposed model takes 0.036 sec per image while PilotNet takes 0.042 sec per image. But there is a slight trade-off of MSE as PilotNet achieves 0.00004 less mean squared error on testing data.

4.4. Checking Model Performance on Different Weather Conditions. To check the robustness of our model, a dataset has been collected by driving the vehicle through the same route under all different 14 weather conditions. Each dataset consists of 1000 images. The results RMSE (root mean squared error) for different weather conditions can be seen in Table 4. From the table, it is notable that the RMS value for those sunsets is slightly better than the weather condition with noon.

5. Conclusions and Future Work

End-to-end learning for AV develops a relationship between input road images and output steering angles to control a vehicle. Most of the available solutions for end-to-end learning are based on deep learning models which have a huge number of parameters, hence making them computationally expensive. Moreover, the underlying hardware needed to deploy such a solution also cost more. Thus, in this paper, we have proposed a lightweight CNN architecture for end-to-end AV. CARLA simulator was used to acquire the training data. The data was preprocessed, and the region of interest was cropped. The proposed model was chosen after trying different combinations of loss functions and activation functions. Finally, the proposed model was compared with PilotNet, which is a state-of-the-art model for end-to-end AV. The model achieved relative mean squared error to PilotNet while being four computationally less expensive. The proposed model was 4 times lighter in terms of parameter as compared to the state-of-the-art model PilotNet. The inference time taken by the proposed model is also less as compared to PilotNet. In future work, we would implement the same model on real images. Moreover, we will also add another target variable which is acceleration/throttle. The model will also be given directional commands as input so it can take decision at intersections.

Data Availability

Data will be made available when requested.

Conflicts of Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We would like to acknowledge CARLA, an open simulator for urban driving, and ROS, an open-source robotics middleware suite for sharing their resources to be able to produce such research for autonomous systems that can be tested and evaluated in the simulation environment and can be easily reproduced into the real-world environment. We would like to thank the National Centre of Artificial Intelligence (NCAI) and NEDUET, Karachi, for providing the necessary resources to conduct this research. This study is supported by the Smart City NCAI, NED University of Engineering and Technology and Pakistan Abu Dhabi University, Abu Dhabi, UAE.

References

[1] M. Bojarski, P. Yeres, A. Choromanska et al., "Explaining how a deep neural network trained with end-to-end learning steers a car," http://arxiv.org/abs/1704.07911.

[2] C. Badue, R. Guidolini, R. V. Carneiro et al., "Self-driving cars: a survey," http://arxiv.org/abs/1901.04407.

[3] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*, vol. 56, Springer, 2009.

[4] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "DeepDriving: learning affordance for direct perception in autonomous driving," in *Proceedings of the IEEE international conference on computer vision*, pp. 2722–2730, 2015.

[5] M. Bojarski, D. Del Testa, D. Dworakowski et al., "End to end learning for self-driving cars," http://arxiv.org/abs/1604.07316.

[6] O. Russakovsky, J. Deng, H. Su et al., "ImageNet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for largescale image recognition," http://arxiv.org/abs/1409.1556.

[9] C. Szegedy, W. Liu, Y. Jia et al., "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[10] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

[11] D. Silver, J. Schrittwieser, K. Simonyan et al., "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[12] D. Silver, T. Hubert, J. Schrittwieser et al., "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," http://arxiv.org/abs/1712.01815.

[13] D. Amodei, S. Ananthanarayanan, R. Anubhai et al., "Deep speech 2: end-to-end speech recognition in English and Mandarin," in *International conference on machine learning, PMLR*, pp. 173–182, 2016.

[14] Z. Chen, T. Zhang, and C. Ouyang, "End-to-end airplane detection using transfer learning in remote sensing images," *Remote Sensing*, vol. 10, no. 1, p. 139, 2018.

[15] D. A. Pomerleau, "ALVINN: an autonomous land vehicle in a neural network," *Advances in neural information processing systems*, vol. 1, 1988.

[16] U. Muller, J. Ben, E. Cosatto, B. Flepp, and Y. Cun, "Off-road obstacle avoidance through end-to-end learning," *Advances in neural information processing systems*, vol. 18, 2005.

[17] Y. Lecun, E. Cosatto, J. Ben, U. Muller, and B. Flepp, *DAVE: autonomous off-road vehicle control using end-to-end learning*, DARPA-IPTO Final Report., 2004.

[18] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-toend driving via conditional imitation learning," in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 4693–4700, 2018.

[19] C. Hubschneider, A. Bauer, M. Weber, and J. M. Zöllner, "Adding navigation to the equation: turning decisions for end-to-end vehicle control," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–8, 2017.

[20] Y. Jia, E. Shelhamer, J. Donahue et al., "Caffe: convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, pp. 675–678, 2014.

[21] S. Du, H. Guo, and A. Simpson, "Self-driving car steering angle prediction based on image recognition," 2019, http://arxiv.org/abs/1912.05440.

[22] F. Iandola and K. Keutzer, "Keynote: small neural nets are beautiful: enabling embedded systems with small deep-neural-network architectures," in *2017 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pp. 1–10, 2017.

[23] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics JMLR Workshop and Conference Proceedings*, pp. 249–256, 2010.

[24] J. Kocić, N. Jovičić, and V. Drndarević, "An end-to-end deep neural network for autonomous driving designed for embedded automotive platforms," *Sensors*, vol. 19, no. 9, p. 2064, 2019.

[25] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: a tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[26] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: an open urban driving simulator," in *Conference on robot learning, PMLR*, pp. 1–16, 2017.

[27] M. Quigley, B. Gerkey, K. Conley et al., "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, vol. 3, p. 5, Kobe, Japan, 2009.

[28] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International conference on engineering and technology (ICET)*, pp. 1–6, 2017.

[29] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings*, pp. 315–323, 2011.

[30] B. Karlik and A. V. Olgac, "Performance analysis of various activation functions in generalized mlp architectures of neural networks," *International Journal of Artificial Intelligence and Expert Systems*, vol. 1, no. 4, pp. 111–122, 2011.

[31] A. Ghosh, H. Kumar, and P. Sastry, "Robust loss functions under label noise for deep neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, 2017no. 1.