LaksNet: an end-to-end deep learning model for self-driving cars in Udacity simulator

Anonymous Authors

Anonymous Institute

Abstract. The majority of road accidents occur because of human errors, including distraction, recklessness, and drunken driving. One of the effective ways to overcome this dangerous situation is by implementing self-driving technologies in vehicles. In this paper, we focus on building an efficient deep-learning model for self-driving cars. We propose a new and simple CNN model called 'LaksNet' consisting of four convolutional layers and two fully connected layers. We conducted extensive experiments using our LaksNet model with the training data generated from the Udacity simulator. Our model outperforms many existing pre-trained ImageNet and NVIDIA models in terms of the duration of the car for which it drives without going off the track on the simulator.

Keywords: Self driving cars · Steering angles · pre-trained ImageNet models · Udacity simulator.

1 Introduction

WHO (World Health Organization) [25] reported that approximately 1.3 million people die each year as a result of road traffic crashes. To reduce the accident rate, We could exploit various machine learning algorithms to drive a vehicle without human intervention. End-to-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 as to minimize accidents. As we have not yet collected the real-time data, we have made use of the simulation environment developed by Udacity for its nano degree program on self-driving cars. We 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.

The major focus of our research is to check the performance of the NVIDIA model and to develop a model better than that of NVIDIA in terms of predicting the steering angles of a self-driving car. We have utilized the Udacity simulator for driving the car in autonomous mode on the track. We first trained the NVIDIA model with our dataset and then measured its performance. As mentioned in the results section 4.2, it could drive the car for 120 seconds without going off the track. Then, we investigated if any of the existing pre-trained ImageNet models would show better performance than that of the NVIDIA model. Finally, we developed our own model to achieve these two objectives. Firstly, our model achieves state-of-the-art performance. Secondly, we build a model with a smaller number of convolutional and fully connected layers so that it requires a smaller number of parameters for training. After conducting extensive experiments, we achieved these two objectives with our LaksNet model.

2 Related Work

Khan et al. [12] looked at the pros and cons of the implementation of autonomous vehicles. They discussed the benefits such as safety, congestion, traffic management, and the adoption of autonomous vehicle technology by various sectors like mining, freight transportation, and the military industry. Shalev et al. [21] proposed a white-box, interpretable, mathematical model for safety assurance, called Responsibility-Sensitive Safety (RSS) for self-driving cars and also the design of a system that adhered to safety assurance requirements and was scalable to millions of cars. Ko et al. [13] proposed a method for key points estimation and point instance segmentation approach for lane detection called Point Instance Network (PINet), which can localize the drivable area on the road. Pan et al. [18] proposed a novel realistic translation network to make the model trained in a virtual environment be workable in the real world. 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. Wu et al. [26] YOLOP (You Only Look Once for Panoptic Driving Perception) for a high-precision and real-time perception system that can assist the vehicle in making reasonable decisions while driving. They presented a YOLOP network to perform traffic object detection, drivable area segmentation, and lane detection simultaneously. Li et al. [15] proposed a Stereo R-CNN-based 3D object detection for autonomous driving by fully exploiting the sparse and dense, semantic, and geometry information in stereo imagery.

Bojarski et al. [3], as part of the NVIDIA research team, proposed a new CNN architecture 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. Zhenye-Na [28] also implemented NVIDIA model architecture for training the model for self-driving vehicles using Udacity's simulation environment to generate training data and test the model. In addition to this, Smolyakov et al. [22] explored various architectures of convolutional neural networks in order to obtain good results with a minimum number of parameters for predicting the steering angles to drive the car in autonomous mode in the Udacity simulation environment.

Santana and Hotz [20] investigated video prediction models based on autoencoders and RNNs. Instead of learning everything in an end-to-end way, they first trained the autoencoder with generative adversarial network-based cost functions to generate realistic-looking images of the road and then trained the RNN transition model in the embedded space. Behley et al. [1] proposed three benchmark experiments based on the KITTI dataset for (i) semantic segmentation of point clouds using a single scan, (ii) semantic segmentation using multiple past scans, and (iii) semantic scene completion to understand LIDAR sequences useful for learning the environment around a self-driving car. Vora et al. [24] proposed PointPainting: a sequential fusion method for 3D object detection in the case of self-driving cars. They mentioned that Point Painting works by projecting lidar points into the output of an image-only semantic segmentation network and appending the class scores to each point. The appended (painted) point cloud can then be fed to any lidar-only method. Accurate depth estimation is a key prerequisite in many robotics tasks, including autonomous driving. Guizilini [7] proposed a novel self-supervised monocular depth estimation method combining geometry with a new deep network, PackNet, learned only from unlabeled monocular videos. They leveraged the novel symmetrical packing and unpacking blocks to jointly learn to compress and decompress detail-preserving representations using 3D convolutions. Bertoni et al. [2] proposed an approach to fundamentally tackle the ill-posed problem of 3D human localization from monocular RGB images for autonomous driving. They addressed the ambiguity in the task by predicting confidence intervals through a loss function based on the Laplace distribution. Gu et al. [6] proposed an approach to learning a long short-term memory (LSTM)-based model for imitating the behavior of Waymo's self-driving model. The proposed model was evaluated based on Mean Absolute Error (MAE) and the experimental results have shown that the LSTM model outperformed several baseline models in driving action prediction. Image Segmentation has been an active field of research as it has a wide range of applications, ranging from automated disease detection to self-driving cars. Jadon [11] summarized some of the well-known loss functions widely used for Image Segmentation and listed out the cases where their usage can help in fast and better convergence of a model and they also introduced a new log-cosh dice loss function. Liao et al. [16] developed KITTI-360 for autonomous driving, the successor of the popular KITTI dataset. KITTI-360 is a suburban driving dataset that comprises richer input modalities, comprehensive semantic instance annotations, and accurate localization to facilitate research at the intersection of vision, graphics, and robotics. They also created a tool to label 3D scenes with bounding primitives and developed a model that transfers this information into the 2D image domain, resulting in over 150k images. Yang et al. [27] proposed a multi-task learning framework to predict the steering angle and speed control simultaneously in an end-to-end manner by taking previous feedback speeds and visual recordings as inputs. Moghadam and Elkaim [17] proposed multi-modal architecture that includes the environmental modeling of ego surrounding and training a deep reinforcement learning (DRL) agent that yields consistent performance in stochastic highway driving scenarios and obtained the high-level sequential commands (i.e. lane change) to send them to lower-level controllers. Deruyttere et al. [5] considered a problem in an autonomous driving setting, where a passenger requests an action that can be associated with an object found in a street scene. Their work presented the Talk2Car dataset, which was the first object referral dataset that contains commands written in natural language for self-driving cars.

3 Methods

Firstly, the images and corresponding steering angles collected from the simulator in training mode are passed onto the CNN model as inputs. Secondly, the CNN model is trained for a 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.

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

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

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

3.1 Model Architecture

Our LaksNet model consists of four convolutional layers, four max-pooling layers, two dropout layers, and two fully connected layers, with one being an output layer, as shown in Fig. 1. We use 3×3 kernels for the first three convolutional layers and 5×5 kernels for the last convolutional layer. The first fully connected layer takes 576 input parameters and gives 256 output parameters, and the second fully connected layer results in a single output because it is the final output layer. The total training parameters of this model are 274,017.

On the other hand, the model developed by NVIDIA architecture [3] consists of five convolutional layers, one dropout layer, and four fully connected layers, with one being an output layer. The total training parameters of this model are 559,419. In addition to these two model architectures, we utilized the below nine ImageNet pre-trained models to check the performance of the self-driving car in the simulation environment AlexNet [14], GoogleNet [23], MobileNetv2 [19], ResNet50 [8], SqueezeNet [10], DenseNet201 [9], NasnetLarge [29], ResNet101 [8], and Xception [4].

3.2 Implementation details

Our model is implemented based on the following configurations:

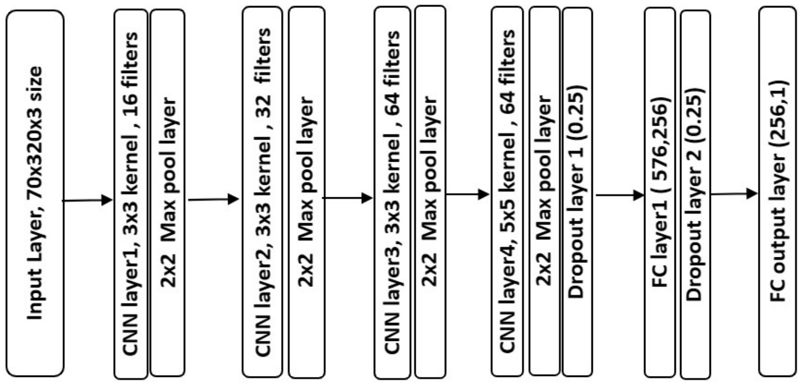

Input Layer, 70x320x3 size
CNN layer1, 3x3 kernel , 16 filters
2x2 Max pool layer
CNN layer2, 3x3 kernel , 32 filters
2x2 Max pool layer
CNN layer3, 3x3 kernel , 64 filters
2x2 Max pool layer
CNN layer4, 5x5 kernel , 64 filters
2x2 Max pool layer
Dropout layer 1 (0.25)
FC layer1 (576,256)
Dropout layer 2 (0.25)
FC output layer (256,1)

Fig. 1: LaksNet model architecture

1. Processor: 12th Gen Intel(R) Core(TM) i7-12700H 2.30 GHz
2. Installed RAM: 16.0 GB
3. System type: 64-bit operating system, x64-based processor
4. GPU: NVIDIA GeForce RTX 3060, 6.0 GB

Our model is trained based on the following hyperparameters:

1. Number of epochs: 50
2. Batch size: 32
3. Optimizer: Adam
4. Learning rate: 0.1

4 Results and Datasets

4.1 Datasets

The Udacity simulation environment has been used to generate training data. It has two different tracks that can be used for training and testing. After selecting either track1 or track2 and then clicking on the record button, a folder of images is created taken from the center, right and left cameras at each instant of time.These images contain details like a portion of the car on the track, a track with borders, and an environment outside the track. A sample of the images generated in training mode at one particular instant of time is shown in Fig. 2.

In addition to these images, the Udacity simulation generates a drivinglog.csv file that contains seven columns. The first three columns contain the path to the images of the front, right and left cameras. Column 4 shows the steering angle values - zero corresponds to the straight direction, a positive value indicates a right turn, and a negative value indicates a left turn. Columns 5, 6, and 7 indicate acceleration, deceleration, and speed of the vehicle, respectively. Some rows of this CSV file are shown in Table. 1.

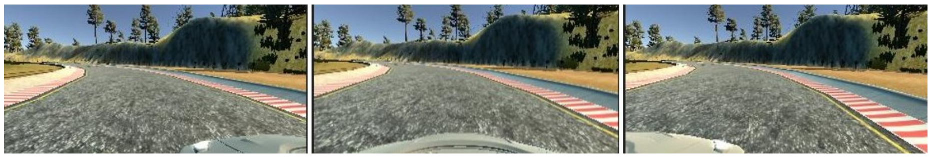


Fig. 2: Images from the Left, Front, and Right cameras

Table 1: Drivinglog.csv file generated by the simulator

Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7
center_216.jpg	left_216.jpg	right_216.jpg	0	0.3325	0	0.2858
center_316.jpg	left_316.jpg	right_316.jpg	0	0.6327	0	0.8770
center_424.jpg	left_424.jpg	right_424.jpg	-0.1215	0.9266	0	1.8474
center_535.jpg	left_535.jpg	right_535.jpg	-0.4860	1	0	3.2320
center_637.jpg	left_637.jpg	right_637.jpg	-0.1827	1	0	4.4072
center_739.jpg	left_739.jpg	right_739.jpg	0	1	0	5.5639

We generated 33096 images from the simulator and combined this with the training data of 97330 images available on the GitHub page of Zhenye-Na [28]. In addition to this, we have generated a separate set of images for validation purposes. 70×320×3 is the size of the images generated in the simulator.

4.2 Results

The block diagram shown in Fig. 3 highlights the testing approach of the model to drive the car in the autonomous mode of the simulation environment. The training images and the respective steering angles generated from the Udacity simulator are fed into the CNN model to train it. After training for a required number of epochs, the model will predict steering angle values that are passed onto the simulator. Then, we notice that the car drives on its own on the track in the simulation environment in autonomous mode.

During the test stage, we notice the steering angles on the terminal and also the car moving in the simulator window during the same time. A snapshot of this is shown in Fig. 4. We can also save the images taken by the front camera during autonomous mode.

After training each of the models, we obtained the results shown in Table. 2. Run time (in seconds) refers to the duration for which the car runs at an average speed of 10 miles per hour, without going off the track. These results highlight that the LaksNet model has shown better performance than that of NVIDIA and the other pre-trained models.Table. 3 shows the actual steering angles, predicted steering angles, and MSE (Mean Square Error) of all the models for 30 test images generated separately from the Udacity simulator. Though the MSE of the LaksNet model is more than that of the NVIDIA model, it could drive the

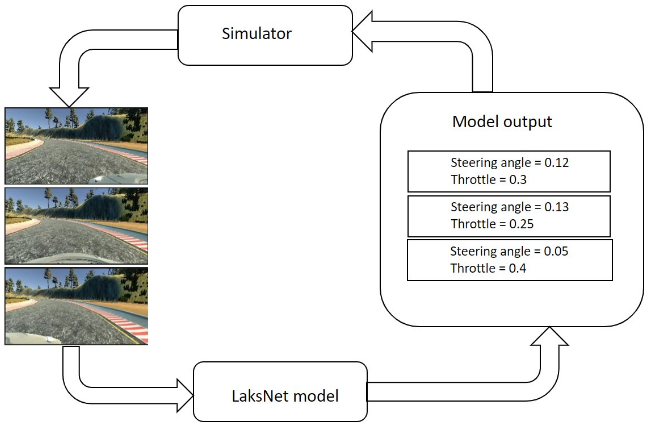
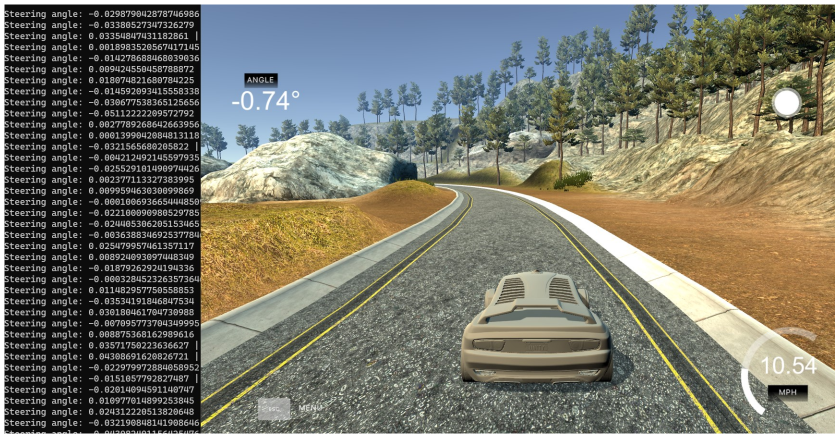


Fig. 3: Block diagram of testing approach in the simulator



```
Steering angle: -0.029879042878746986
Steering angle: -0.03380527347326279
Steering angle: 0.03354847431182861 |
Steering angle: 0.0018983520567417145
Steering angle: -0.014278688468039036
Steering angle: 0.009424550458788872
Steering angle: 0.018074821680784225
Steering angle: -0.014592093415558338
Steering angle: -0.030677538365125656
Steering angle: -0.05112222209572792
Steering angle: 0.0027789268642663956
Steering angle: 0.0001399042084813118
Steering angle: -0.0321565680205822 |
Steering angle: -0.004212492145597935
Steering angle: -0.025529101490974426
Steering angle: 0.002377113327383995
Steering angle: 0.009959463030099869
Steering angle: -0.00010069366544485
Steering angle: -0.022100090980529785
Steering angle: -0.024405306205153465
Steering angle: -0.0036388346925377784
Steering angle: 0.025479957461357117
Steering angle: 0.008924093097448349
Steering angle: -0.01879262924194336
Steering angle: -0.00032845232635736
Steering angle: 0.011482957750558853
Steering angle: -0.03534191846847534
Steering angle: 0.030180461704730988
Steering angle: -0.007095773704349995
Steering angle: 0.008875368162989616
Steering angle: 0.03571750223636627 |
Steering angle: 0.04308691620826721 |
Steering angle: -0.022979972884058952
Steering angle: -0.0151057792827487 |
Steering angle: -0.02014094591140747
Steering angle: 0.010977014899253845
Steering angle: 0.024312220513820648
Steering angle: -0.032190848141908646
```

Fig. 4: A snapshot of steering angles predicted by the model in the terminal and simulator window in autonomous mode

Table 2: Car run time(in secs) on the track in the simulator for each model

Model	Run time(in secs)
AlexNet [14]	50
GoogleNet [23]	50
MobileNet [19]	8
ResNext50 [8]	3
SqueezeNet [10]	12
DenseNet201 [9]	7
NasnetLarge [29]	4
ResNet101 [8]	14
Xception [4]	5
NVIDIA [3]	120
LaksNet	**150**

car on track for more time. This might be due to the fact that it did not overfit the data.

Table 3: Predicted steering angles and MSE by each model for 30 test images

Actual angles	**LaksNet**	NVIDIA	Alexnet	Googlenet	Mobilent	Resnext50	Squeezenet	Densenet201	Nasnetalarge	Resnet101	Xception
-0.341	-0.348	-0.079	-0.133	-0.288	-0.122	-0.040	0.000	0.053	3.315	-0.039	-0.005
0.000	-0.355	-0.082	-0.033	-0.123	0.078	-0.200	0.000	0.025	3.294	0.110	0.026
0.000	-0.363	-0.098	-0.289	-0.201	-0.100	-0.216	0.000	0.014	3.458	0.018	0.009
-0.072	-0.375	-0.186	-0.053	-0.144	-0.202	-0.121	0.000	0.002	3.507	0.001	0.084
-0.418	-0.355	-0.079	-0.035	-0.148	-0.073	-0.109	0.000	0.037	3.459	-0.073	-0.029
-0.116	-0.350	-0.094	-0.206	-0.105	-0.089	0.015	0.000	-0.006	3.563	-0.009	0.001
0.000	-0.325	-0.032	-0.225	-0.191	-0.108	-0.126	0.014	0.032	3.421	-0.190	-0.005
0.000	-0.251	-0.060	-0.169	-0.121	-0.064	-0.200	0.000	-0.032	3.511	-0.152	-0.018
0.000	-0.280	-0.030	0.001	-0.311	-0.290	-0.070	0.0229	0.032	3.576	-0.088	-0.037
0.000	-0.375	-0.036	-0.197	-0.014	-0.109	-0.112	0.025	0.014	3.652	-0.074	-0.034
0.000	-0.360	-0.032	-0.297	-0.210	-0.275	-0.138	0.000	-0.017	3.742	-0.108	-0.008
0.000	-0.383	-0.068	-0.023	-0.156	-0.091	-0.112	0.000	0.041	3.632	-0.030	0.681
0.000	-0.417	-0.024	-0.032	-0.094	-0.093	-0.318	0.000	-0.006	3.742	-0.182	0.004
-0.071	-0.464	-0.059	-0.130	-0.011	-0.101	-0.017	0.000	-0.075	3.745	0.001	0.029
-0.391	-0.395	-0.064	0.001	-0.073	-0.312	-0.124	0.000	-0.041	3.978	-0.103	-0.013
-0.279	-0.402	-0.055	-0.057	-0.123	-0.047	-0.072	0.000	0.029	3.722	-0.065	-0.049
0.000	-0.264	-0.045	-0.044	-0.038	-0.057	0.006	0.000	0.144	3.901	-0.105	-0.012
0.000	-0.219	-0.029	-0.020	-0.006	-0.071	-0.056	0.000	0.000	3.840	-0.151	0.085
0.000	-0.206	-0.049	-0.147	-0.105	-0.318	-0.071	0.000	0.066	3.818	0.015	0.003
0.000	-0.238	-0.048	-0.065	-0.123	-0.028	-0.023	0.000	0.076	3.703	-0.020	-0.001
0.000	-0.237	-0.056	-0.012	-0.119	-0.026	-0.040	0.000	0.053	3.794	0.101	-0.003
0.000	-0.280	-0.042	-0.228	-0.023	-0.039	-0.045	0.000	0.092	3.583	0.015	0.001
0.000	-0.278	-0.044	0.034	-0.069	-0.054	-0.050	0.000	0.011	3.625	0.022	-0.014
0.000	-0.251	-0.036	-0.135	-0.116	-0.038	-0.239	0.000	0.051	3.692	-0.057	0.011
0.000	-0.335	-0.053	-0.007	-0.075	-0.049	-0.231	0.000	0.083	3.771	0.015	-0.026
0.000	-0.305	-0.040	-0.184	-0.178	-0.081	-0.074	0.000	-0.026	3.762	-0.037	-0.013
0.000	-0.483	-0.051	-0.068	-0.151	-0.181	-0.096	0.000	0.088	3.651	-0.052	-0.014
0.000	-0.342	-0.051	-0.077	-0.167	-0.089	-0.092	0.000	0.116	3.557	0.026	-0.019
0.000	-0.381	-0.074	-0.244	-0.214	-0.131	-0.010	0.000	-0.027	3.518	-0.042	-0.006
0.000	-0.345	-0.083	-0.151	-0.108	-0.063	-0.038	0.000	-0.113	3.589	-0.093	-0.018
MSE	0.091	0.014	0.031	0.023	0.021	0.047	0.066	0.023	13.682	0.020	0.033

5 Discussion

As we are inspired by NVIDIA model architecture for end-to-end learning of a self-driving car, we first trained this model with our data set and noticed that the car ran properly on the track in the simulator for 120 seconds in autonomous mode and then deviated from the track. Then, we decided to check the performance of the available pre-trained models for this purpose. Surprisingly, these complex models have shown poor performance when compared to the NVIDIA model. This might be due to the fact that all these pre-trained models are designed for classification tasks to identify a particular category instead of predicting numerical values but our project is a regression problem as we are predicting continuous numerical values (steering angles). Secondly, each of the similar images can have different angles, which also confuses these complex models. Among these models, Alexnet and Googlenet have given better results when compared to the other pre-trained models because these two are simpler models in terms of the number of convolutional layers.

As the pre-trained models have not met the expectations set by the NVIDIA model, we have tried to build our own CNN models that are more competent. One of our CNN models, with just four convolutional layers and two fully connected layers, could drive the car on the track properly for 150 seconds, which is more than that of the NVIDIA model. To arrive at this model, we have built many models and checked their performances.

5.1 Hyper-parameter tuning

We first explored a CNN model with seven convolutional layers using 3×3 filters. We initially thought that adding more convolutional layers would make the model learn better and would show better results but this model could drive properly for just 20 seconds. We observe that adding more layers will extract features up to a certain extent and eventually overfit the data instead of extracting features. So, we built a few more models with five and six convolutional layers using bigger kernels of size 5×5 and 7×7. We observed that the model with five convolutional layers using 5x5 filters and another model using a combination of 7×7 and 5×5 filters have shown a major improvement in their performance by driving the car properly on the track for 90 seconds. Then, we built another model with just 3×3 convolutional layers using 7×7 filters but it could drive the car properly for just 50 seconds. Later, when we reduced the filter size to 3×3 in this model, it could drive the car properly for 120 seconds, meeting the expectations of the NVIDIA model. Therefore, we noticed that models with a smaller number of layers and using small filter sizes are giving better results in this case, as shown below in Table. 4.

In an attempt to achieve better results than that of the NVIDIA model, we have experimented with various models with 3 or 4 convolutional layers using 3×3 or 5×5 or a combination of 3×3 and 5×5 filters. Finally, our LaksNet model with four convolutional layers, using 3×3 filters for the first three layers and a 5×5 filter for the last convolutional layer, has achieved better results driving the car properly for 150 seconds. In this model, we only used two fully connected layers, with one being an output layer. When we tried this model with 3 or 4 fully connected layers, it was overfitting and did not yield better results.

Table 4: Run time of the car for various model configurations

Model configuration	Run time(in secs)
Seven layers, all 3×3 filters	20
Five layers, all 5×5 filters	90
Five layers, three 7×7 filters and two 5×5 filters	90
Three layers, all 7×7 filters	50
Three layers, all 3×3 filters	120

In the LaksNet model, we added a max pooling layer after each convolutional layer to reduce the height and width of the images. Another alternative way is the usage of an average pooling layer. In the fully connected block, we added a dropout layer after the last convolutional layer and another dropout layer after the first fully connected layer to avoid over-fitting. We applied the ReLU activation function for all the convolutional and fully connected layers, except the output layer, to fire neurons. The other suitable alternative to ReLU is the ELU activation function.

Regarding loss function, we used the 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. In addition to the data augmentation process, we normalized the data to avoid saturation so that gradients would work better for the model training. We also performed random rotations on the images and cropped the images as well. This is required for better training of the model. As part of generating the training data from the simulator, one has to take care, while driving manually, to ensure the car always moves 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

In this paper, we built a LaksNet model with only four convolutional and two fully connected layers for the autonomous driving base on the Udacity simulator. Extensive experimental results demonstrate that our proposed LaksNet model achieved state-of-the-art performance. Our future work will focus on building a model that will run the car forever without going off the track under the Udacity simulator. In addition, we would like to predict acceleration values along with steering angles and ensure that the car will always run in the middle of the track.

References

1. Behley, J., Garbade, M., Milioto, A., Quenzel, J., Behnke, S., Stachniss, C., Gall, J., Semantickitti: A dataset for semantic scene understanding of lidar sequences.

In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 9297–9307
2. Bertoni, L., Kreiss, S., Alahi, A.: Monoloco: Monocular 3d pedestrian localization and uncertainty estimation. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 6861–6871 (2019)
3. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., et al.: End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316 (2016)
4. Chollet, F.: Xception: Deep learning with depthwise separable convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1251–1258 (2017)
5. Deruyttere, T., Vandenhende, S., Grujicic, D., Van Gool, L., Moens, M.F.: Talk2car: Taking control of your self-driving car. arXiv preprint arXiv:1909.10838 (2019)
6. Gu, Z., Li, Z., Di, X., Shi, R.: An lstm-based autonomous driving model using a waymo open dataset. Applied Sciences **10**(6), 2046 (2020)
7. Guizilini, V., Ambrus, R., Pillai, S., Raventos, A., Gaidon, A.: 3d packing for self-supervised monocular depth estimation. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 2485–2494 (2020)
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778 (2016)
9. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4700–4708 (2017)
10. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. arXiv preprint arXiv:1602.07360 (2016)
11. Jadon, S.: A survey of loss functions for semantic segmentation. In: 2020 IEEE conference on computational intelligence in bioinformatics and computational biology (CIBCB). pp. 1–7. IEEE (2020)
12. Khan, F., Kumar, R.L., Kadry, S., Nam, Y., Meqdad, M.N.: Autonomous vehicles: A study of implementation and security. International Journal of Electrical & Computer Engineering (2088-8708) **11**(4) (2021)
13. Ko, Y., Lee, Y., Azam, S., Munir, F., Jeon, M., Pedrycz, W.: Key points estimation and point instance segmentation approach for lane detection. IEEE Transactions on Intelligent Transportation Systems **23**(7), 8949–8958 (2021)
14. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems. pp. 1097–1105 (2012)
15. Li, P., Chen, X., Shen, S.: Stereo r-cnn based 3d object detection for autonomous driving. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7644–7652 (2019)
16. Liao, Y., Xie, J., Geiger, A.: Kitti-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d. IEEE Transactions on Pattern Analysis and Machine Intelligence (2022)
17. Moghadam, M., Elkaim, G.H.: A hierarchical architecture for sequential decision-making in autonomous driving using deep reinforcement learning. arXiv preprint arXiv:1906.08464 (2019)
18. Pan, X., You, Y., Wang, Z., Lu, C.: Virtual to real reinforcement learning for autonomous driving. arXiv preprint arXiv:1704.03952 (2017)

19. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4510–4520 (2018)
20. Santana, E., Hotz, G.: Learning a driving simulator. arXiv preprint arXiv:1608.01230 (2016)
21. Shalev-Shwartz, S., Shammah, S., Shashua, A.: On a formal model of safe and scalable self-driving cars. arXiv preprint arXiv:1708.06374 (2017)
22. Smolyakov, M., Frolov, A., Volkov, V., Stelmashchuk, I.: 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 (2018)
23. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proc. of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1–9 (2015)
24. Vora, S., Lang, A.H., Helou, B., Beijbom, O.: Pointpainting: Sequential fusion for 3d object detection. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 4604–4612 (2020)
25. WHO: Road traffic injuries. WHO report (2022), `https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries`
26. Wu, D., Liao, M.W., Zhang, W.T., Wang, X.G., Bai, X., Cheng, W.Q., Liu, W.Y.: Yolop: You only look once for panoptic driving perception. Machine Intelligence Research pp. 1–13 (2022)
27. Yang, Z., Zhang, Y., Yu, J., Cai, J., Luo, J.: End-to-end multi-modal multi-task vehicle control for self-driving cars with visual perceptions. In: 2018 24th International Conference on Pattern Recognition (ICPR). pp. 2289–2294. IEEE (2018)
28. Zhenye-Na: End-to-end learning for self-driving cars — udacity's simulation env (2019), `https://github.com/Zhenye-Na/e2e-learning-self-driving-cars`
29. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 8697–8710 (2018)