Behavioral Cloning in Autonomous Vehicle Using Deep Learning

Aman Agarwal*†
15bce006@nirmauni.ac.in
Computer Science and Engineering,
Nirma University
Ahmedabad, India-382481

Aditya Mishra†
15bce003@nirmauni.ac.in
Computer Science and Engineering,
Nirma University
Ahmedabad, India-382481

Priyanka Sharma
priyanka.sharma@nirmauni.ac.in
Computer Science and Engineering,
Nirma University
Ahmedabad, India-382481

Abstract

In this paper, we critically compare various deep learning model architectures for mimicking the behavior of a driver for autonomous driving. We have designed an indoor car track and prepared the Autonomous-Arena dataset from the dashcam images of a remote-controlled (RC) car. This data is used to train deep learning models and predict the steering direction of the vehicle. The data is fed into the models after several preprocessing and augmentations. We have applied Convolutional Neural Network (CNN) for the classification of images. Various other models are also used, including Recurrent Neural Network (RNN), a combination of CNN and RNN, and Residual Network (ResNet). We show that all these networks give comparable results and also highlight the challenges with the data in training deep neural networks.

Keywords: Autonomous-Arena, Autonomous Vehicle, Convolutional Neural Network.

1 Introduction

Deep learning has been a cynosure of the IT industry from the last few years. It has shown near-human results in object detection, natural language processing, dimensionality reduction, motion modeling, and many other fields. Although, it has been there for decades now, recently, the increase in the amount of data and computational power (in the form of GPU) has accelerated the use of deep learning commercially. The main reason for these extraordinary results of deep learning is its ability to map complex nonlinear functions.

Deep learning offers various approaches to different kinds of data. CNN is used for spatial data like images [1], it does so by looking at different segments of the image for essential features. RNN finds the relation between different parts of data over time and can be used for temporal (time series) data like text classification or language translation.

Many technological fields are using deep learning for one or the other problem. Similarly, autonomous driving has been an active area of research because of excellent results in pedestrian detection [2, 3], vehicle detection [4, 5], and environment perception [6]. Additionally, many groups have used deep learning for end-to-end autonomous driving [8].

Our project focuses on behavioral cloning by predicting different directions in which a car should go, given an image from a monocular camera. These images are passed through a deep learning models, that are used to detect essential features in an image and predict the direction in which the car should be maneuvered. The goal of this work is to provide proof of the concept that image data alone can be used to drive a car autonomously. The code for the project is available at this link[1].

2 Related Work

ALVINN [7] was among the very first attempts to use deep learning in the field of autonomous driving for predicting the direction of motion of the car. It followed a simple approach where image pixels were given as input, and a shallow network of three layers was used for predicting the direction. Its success depicted that neural networks could be used for this task.

Recently, NVIDIA [8] has given a full-fledged framework using end-to-end deep learning for autonomous driving. This framework shows the power of CNN to extract relevant features from video frames, and use them in relatively simple real-time scenarios such as lane following in obstacle-free environments.

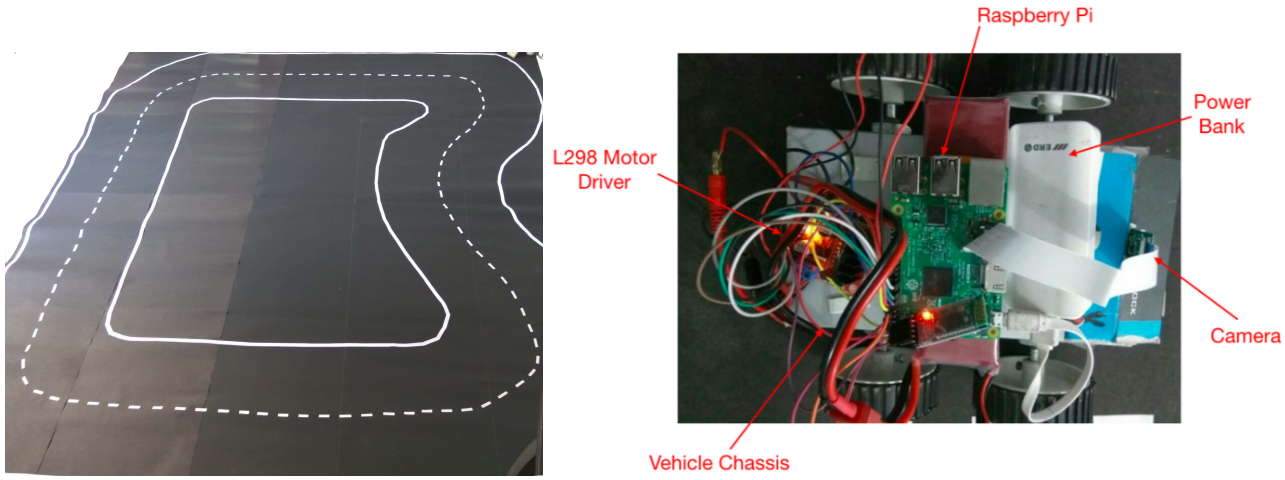
Brody Huval *et al.* [4] uses deep learning and computer vision for car and lane detection. They applied a modified version of selective search [10] and used object mask detector, as described in Szegedy *et al.* [9], for the detection of vehicles on road. For lane detection, they used the same CNN model with regression over a six-dimensional feature vector.

Paul Drews *et al.* [12] combined model predictive control (MPC) and CNN for high-speed autonomous driving. They fed the image from a monocular camera into CNN, and the cost function for MPC was predicted. This cost was then directly supplied to the MPC algorithm for online trajectory optimization.

Recently, Udacity started a challenge to build an open-source self-driving car, and many people have come up with great ideas. Greg Katz *et al.* [13] use LSTM along with CNN

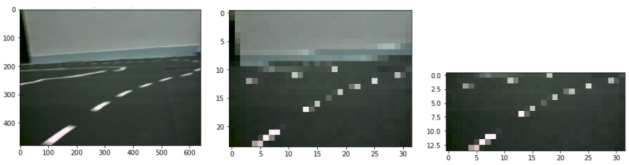
*Corresponding author.
†Equal contribution.

[1] https://github.com/amanbasu/Autonomous-Car-Prototype



(a) Indoor car track **(b)** Raspberry Pi controlled vehicle

Figure 1. (a) Circular track made using chart papers to generate the Autonomous-Arena dataset, (b) top view of the car which was driven on the track. Different components used in the car were Raspberry Pi, camera, L-298 motor driver, aluminium chassis, power bank, and 11.3V Li-Po battery.



(a) Original image **(b)** Resized image **(c)** Cropped image

Figure 2. Image preprocessing was applied to reduce the size and remove the unnecessary information from the images.

for predicting the steering angle of the car. Shuyang Du *et al.* [14] proposed two models, one with 3D CNN and LSTM for covering Spatio-temporal features [15] having residual connections, and the other with transfer learning.

Comma.ai [16] uses Variational Autoencoders for simulating driving control with classical and learned cost function using Generative Adversarial Network (GAN) for learning frame embeddings.

Ahmad El Sallab *et al.* [17] introduced Reinforcement Learning for driver control combined with RNN for information integration, which helped the system to handle a partially visible environment. It also incorporates the attention mechanism to focus on the relevant information.

3 Dataset

Autonomous-Arena dataset was generated by running the RC car on an indoor, circular track. The track was made of black chart paper having dual lanes in white. The Raspberry Pi controlled car was equipped with a camera that was mounted on the front of the car. As the car was driven on the track, the camera took images at 8 frames per second. A higher frame rate would have lead to similar frames without gaining relevant information. The track and car details can be seen in Fig. 1.

The RC car was controlled by an android app that gave signals to a bluetooth unit. The Raspberry Pi took input from the bluetooth unit and maneuvered the car according to the signal in the desired direction. There were three signals: `front` (F), `left` (L), and `right` (R) to control the car. At each image frame taken by the camera, the Bluetooth signal was registered in the memory along with the image name at that instance.

The car was driven in five sets having nearly 1000 images each and at a different time of the day in different directions to get a varied dataset. The final dataset contained almost 14,000 labeled images and can be found on Kaggle[2].

3.1 Preprocessing and Augmentation

The dataset thus obtained was preprocessed before sending it to the network. Images captured by the Raspberry Pi camera were single channeled of size 480x640 pixels. To reduce the number of pixels and thus, the training time, they were resized to 24x32 pixels. Resizing to this resolution did not cause any loss in feature since the main features in our data i.e. the track lanes, were still conspicuous.

For the car to make a decision, it should be able to see the things that are just in front of it, and not very far away [11]. The camera took the images such that nearly half of it contained meaningless information like a wall, window, table, etc., in the upper region of the picture. Consequently, all the images were cropped from above to 14x32 pixels, such

[2] https://www.kaggle.com/firstofhisname/indoor-car-track

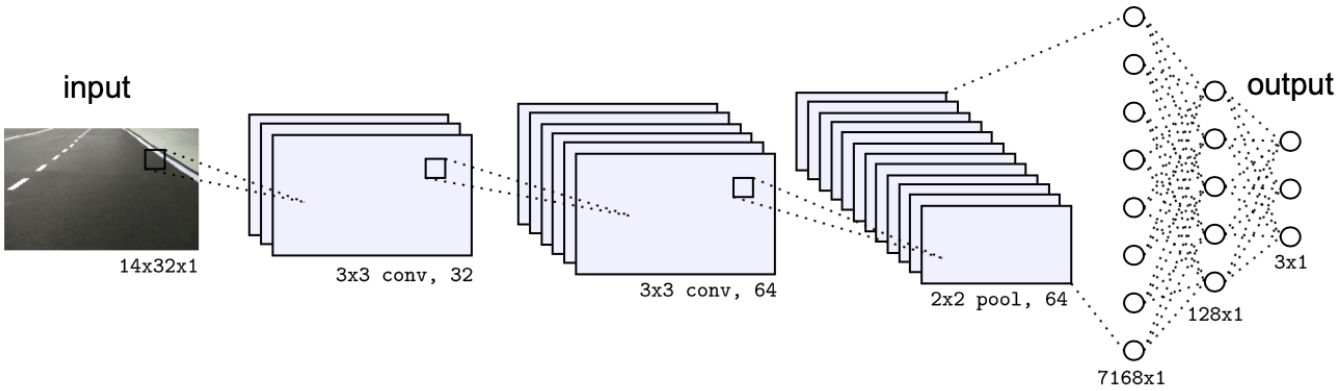


Figure 3. CNN architecture consisting of two convolution, one pooling, two fully connected and one output layer. The total trainable parameters were 918,979.

that only the track lanes were visible. The preprocessing of the image can be visualized in Fig. 2.

Another problem with the data was that it was skewed. This meant that the number of samples for each class were non-uniform. The reason for the skewness was that the track contained more straight roads than the curved ones, and moreover, the number of right and left turns were disparate. The issue was resolved by flipping the images with turns along the vertical axis, so that the number of left and right labels become equal. This also increased the training data size. To make the `front` labels equal to the one with turns, they were under-sampled during the training process.

Each pixel value of the image was between 0 and 255. As a common practice, we normalized these pixels to have a value between 0 and 1. The normalization was done to reduce the excess computation of large numeric values, which made the model converge faster.

4 Network Architecture

We trained our data on four different network architectures, the details of which are listed in the following sections.

4.1 CNN (Model 1)

The network was trained to reduce the cross-entropy error between the predicted and the actual direction the car must follow. The network architecture is shown in Fig. 3 that consists of six layers, including two convolutions, one max pool, one flattened, one fully connected, and one output layer.

The input image was fed as a 14x32x1 matrix with the pixel values normalized in the range of 0 to 1. Grayscale (one channel) images were used since the track and lanes were in black and white with the area above the horizon cropped. As a result, there were no color-dependent features in the image which the network could learn. The single-channeled image reduced the training time and also the number of parameters of the model.

The size and number of kernels to detect the features were chosen empirically. In our model, the kernel size of 3x3 was used with the stride of 1x1 in both the convolution layers. As the image dimension was low, longer strides would have missed useful features. 32 filters were used in the first convolution layer, followed by 64 in the second layer. With kernel size and stride of 2x2, the max pool layer reduced the image size to 7x16x1 after the second convolution layer. Consequentially, the flatten layer size was 7168x1 followed by one fully connected layer of size 128x1, which finally produced three outputs. Rectified linear unit (ReLU) activation function was used between the layers to add the non-linearity.

A small network was chosen since the data was obtained from a single source i.e. self-made track, so it lacked diversity. Moreover, the number of features (only track lanes) were very less.

4.2 LSTM (Model 2)

Convolutional neural networks are used to determine the spatial relationship in the data. But, we wanted to test our data for any temporal relationship. Therefore, we applied recurrent neural networks and basically LSTM. LSTMs are a special kind of RNN, that are capable of learning long-term dependencies [19]. They work tremendously well on a large variety of problems and are now widely used.

We experimented with various LSTM models with single and stacked layers. The inputs were given as an array of size 14xBx32 where B is the batch size. The value of B in our models was 64. Each image was fed into the LSTM cell one row at a time for the entire batch, and the output was obtained as a single value i.e. the class label.

The first model contained a single LSTM layer of batch size 64 with 128 cells. In the second and third sequential models,

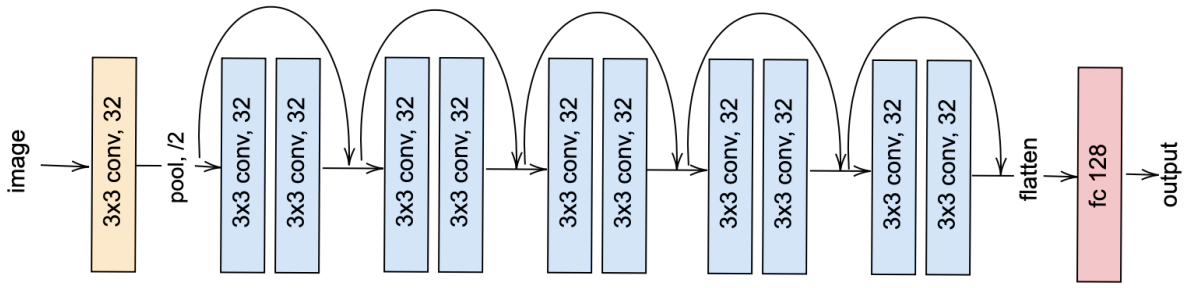


Figure 4. Residual Network architecture with six groups of convolutional layers and one group of dense (fully connected) layer. Five skip connections were made between the groups of convolutional layers.

two and three stacked LSTM layers were used respectively with a dropout of 0.8 between the layers.

4.3 CNN+RNN (Model 3)

In this model, we tried the characteristics of both CNN and RNN. CNN was used to get the low dimensional encoding of the image that was passed to the LSTM layer to get the output label.

The network architecture was nearly similar to the one given in Fig. 3 with a few minor changes. Kernel size of 5x5 was used with the stride of 2x2 in the two convolution layers with 32 filters in the first layer, and 64 in the second layer. Max pool layer with the kernel size and stride of 2x2 reduced the image size to 7x16x1 after the first layer, and to 4x8x1 after the second layer. A dropout of 0.8 was added after every max pool layer. An LSTM layer, consisting of 64 cells was added between the flatten and the fully connected layer. The output of the second max pool layer was reshaped to 4xBx8 before passing to the LSTM layer, where B is the batch size. This layer was followed by a dense layer of size 128x1, followed by the last layer giving 3 outputs.

4.4 ResNet (Model 4)

Deep neural networks are sometimes difficult to train because of the vanishing or exploding gradient problems. Skip connections in Residual Network allow us to take the activation from one layer and feed it to another layer much deeper in the neural network. We used the same intuition to make our own residual network shown in Fig. 4.

The network consists of 7 groups of different layers with all the convolution layers consisting of 32 filters with a kernel size of 3x3 and stride of 1x1. In the first group, the input is passed to the convolution layer, followed by the ReLu and max pool layer of stride 2x2. The next group consists of two convolutional layers without the pooling layer. The output of this group is added to the output of the first group, thus giving a skip connection. After applying the activation, the result is passed to the next group.

The process is repeated with all the next four similar groups. The output after the last skip connection is then flattened and passed to a fully connected layer of 128 neurons, which finally outputs the three classes.

5 Results

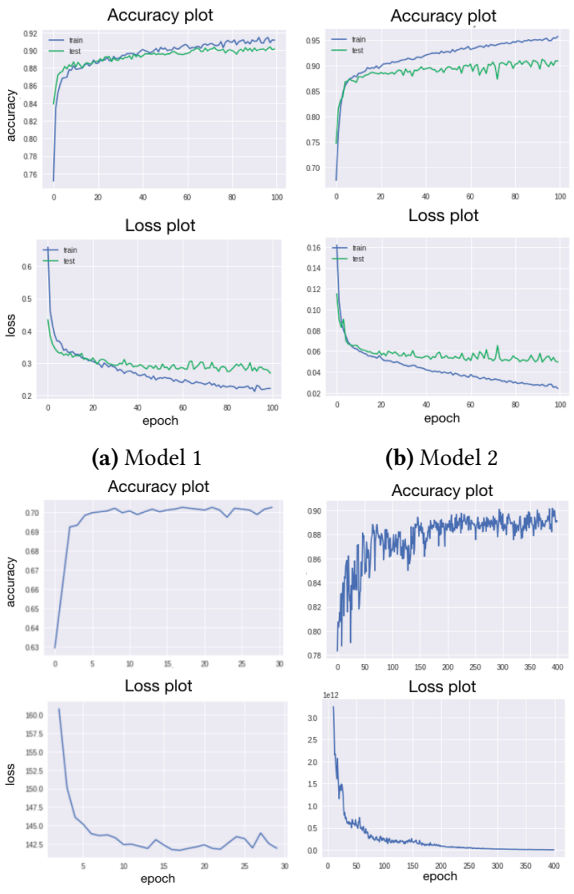
All our models were trained on the Nvidia Tesla K80 system that offers a total of 24 GB of GDDR5 on-board memory (12 GB per GPU) and supports PCI Express Gen3. The comparison of accuracy from all the models on the training and testing set is given in Table. 1.

Table 1. A comparison of different network architectures showing the number of epochs, training accuracy and testing accuracy.

Architecture		Dropout	Epochs	Accuracy	
				Train	Test
Model 1		0.7	100	89.04	88.67
		0.8	100	91.16	90.25
Model 2	1 layer	-	50	91.84	89.93
	2 layer	0.8	50	92.97	**90.39**
	3 layer	0.8	100	**95.70**	90.34
Model 3		0.8	30	70.28	69.83
Model 4		0.7	400	90.10	90.34

We experimented with a learning rate of 1e-3, 1e-4, and 1e-5 on our networks for small number of iterations and observed an effective learning rate of 1e-4 that provided the best results. Adam optimizer was used while training for regulating the learning rate [18]. The mini-batch size of 64 was kept as it performed well on our data.

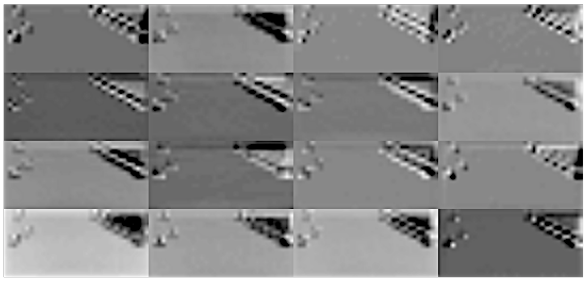
Model 1 converged very fast, considering the low-resolution images and a shallow network. Further training would have overfitted the training set. After multiple runs of the model, we observed a training accuracy of 91.16% while a testing accuracy of 90.25%. The accuracy and loss curves of the model can be seen in Fig. 5(a). The activations of different filters in the first convolutional layer can be visualized in Fig. 6. The



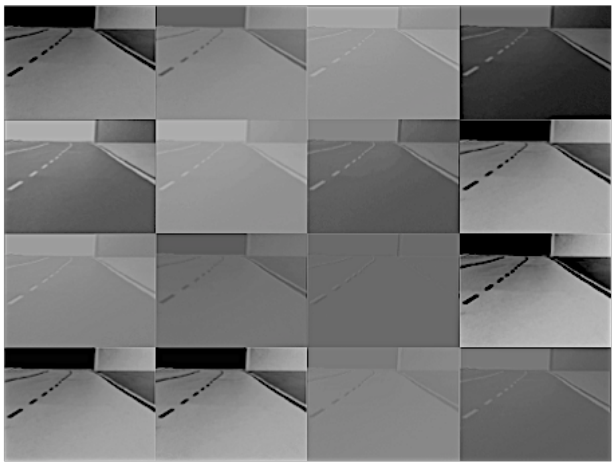
(a) Model 1 **(b)** Model 2

(c) Model 3 **(d)** Model 4

Figure 5. Accuracy and loss curves of different network architectures over number of iterations.



(a) 14x32 dimensional feature maps.



(b) 480x640 dimensional feature maps.

Figure 6. Feature maps from the first convolutional layer of Model 1 showing the efficacy of the model to detect the track lanes.

figure shows the efficiency of our model to detect the track lanes in the data.

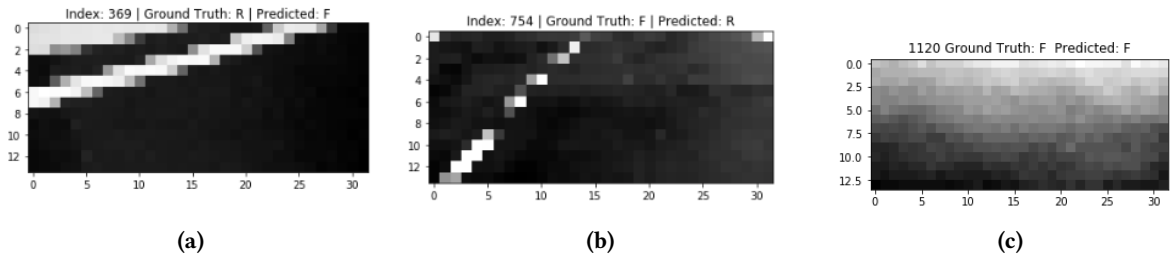
For Model 2, we experimented with single and stacked LSTMs with two and three layers. It was found that the model with a single LSTM layer gave a test accuracy of 89.93%. Whereas, the model with three stacked LSTM layers was overfitting with training accuracy of 95.70% and testing accuracy of 90.34%. Which made the model with two stacked LSTM layers better than all sequential models, providing 90.39% accuracy, which was almost the same as that given by the CNN model.

The results of Model 3 were not compelling. The accuracy on the training as well as on the testing set was around 70%, which was worse than the accuracy of other models. The intuition behind this model was that the convolutional layers were used to find the lower dimensional encoding of the input image, which was fed into the sequential model to find the temporal relationships in these encodings. One possible reason for the poor performance of this model might be that the lower dimensional encoding obtained from the convolutional layers lacked any temporal relation that the sequential model could accurately represent.

Model 4 was trained with a batch size of 128. To prevent overfitting, we applied a dropout of 0.7 between the layers. The training was stopped when the loss curve started reaching a plateau. This model was the deepest one in all our models and took a long time to converge. It was trained for almost 400 epochs, after which the loss became constant. We achieved a testing accuracy of 90.34% that was comparable with the accuracy of Model 1 and Model 2.

6 Error Analysis

Seeing the accuracy after training of all the models, it could be said that the results were not exceptional. But actually, the reason for such less accuracy could be the huge human error that was introduced in the dataset while labeling. The ambiguous interpretation of the track images also plays a significant role in the error. In the following sections, some of the reasons for this accuracy are mentioned in detail.



(a) (b) (c)

Figure 7. Examples of (a), (b) wrongly labled images (c) noisy images unsuitable for training. The major contributors to the poor performance of neural networks.

6.1 Wrong Labels

Human error played a huge role in the bad performance of all the models. The images were labeled manually, and therefore, there were high chances of human error. This can be clarified by taking the examples of some training samples. In Fig 7(a), the ground truth value of an image is given as R (`right`) while the predicted value is F (`front`). This is a case of wrong labeling. The ground truth label should have been F because the car was at a safe distance from the boundary lane and could have gone straight for a few frames, instead of taking a right turn. Which means that the model correctly predicted the label. The same mistake can be observed in Fig 7(b) where the car needed to go right instead of the front.

6.2 Noisy Image

Some of the training examples, like Fig 7(c), were not obtained properly and contained noise. The reasons leading to such unwanted noise could be the blurry or very bright images taken during the data generation phase or the errors in preprocessing the data. Hence, such images were unsuitable for training.

6.3 Ambiguous Interpretations

The track between the lanes on which the car ran was wide enough. There was a gap of around 5cm on either side of the road when the car was in between. Therefore, while labeling the data, there could be two options when the car was at a curve, either to take a turn on that frame and go straight in the next or to go straight in that frame and take a turn on the next one. Such situations led to the ambiguous interpretation of the images while labeling and influenced the error.

A point to note here is that although the accuracy of the model was low per se, the actual performance of the vehicle on the track was better. It can be reasoned that when the vehicle is on the turn, the model can predict any sequence of `front` and `left/right` labels to make a successful turn. And as long as the turn is made, it doesn't matter what this sequence was. So practically, the accuracy of the model was good enough to be deployed on the autonomous car prototype.

7 Conclusion & Future Work

This project was aimed at building a self-driving car that maneuvers itself on a track. The project was divided into three phases. The first phase included image generation that was achieved by running the car and taking the picture of the track in front of it. These images were appropriately labeled with the direction in which the car was moving to make the Autonomous-Arena dataset.

The second phase was to process the collected data and make a deep neural network to map the input images to their corresponding direction labels. Many different neural network architectures were tried on the data, and their results were compared.

And the third phase was to deploy this well-trained model on the actual car and run it autonomously. Under the test conditions, the vehicle drove well autonomously on the straight track but sometimes struggled on the turns, by going off-road.

It was observed that three of our four models were giving an accuracy of around 90%. We also saw that the convolution layers were able to detect primary features of the track, like lanes, without providing any hand-engineered features. The under-performance of the models and the reason for their high bias was found in the form of human error while labeling the data along with other minor issues with the data. We proved that only a good accuracy is not necessarily an indicator of a well trained neural network as the car drove well on the real track.

In the future, we would like to scale this approach to real cars where the data is much larger and the process more complicated. As the end-to-end approach is difficult to apply on real cars due to varied natural conditions and computational requirements, we would like to work on other sensory information that could be incorporated to make better decisions.

Acknowledgments

We would like to thank the Department of Computer Science and Engineering, Nirma University for funding our project and providing us with the necessary hardware components used in this project.

References

[1] LeCun, Yann, Léon Bottou, Yoshua Bengio and Patrick Haffner. "Gradient-based learning applied to document recognition." (1998).

[2] Tian, Yonglong, Ping Luo, Xiaogang Wang and Xiaoou Tang. "Pedestrian detection aided by deep learning semantic tasks." 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015): 5079-5087.

[3] Du, Xianzhi, Mostafa El-Khamy, Jungwon Lee and Larry Davis. "Fused DNN: A Deep Neural Network Fusion Approach to Fast and Robust Pedestrian Detection." 2017 IEEE Winter Conference on Applications of Computer Vision (WACV) (2017): 953-961.

[4] Huval, Brody, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, Fernando A. Mujica, Adam Coates and Andrew Y. Ng. "An Empirical Evaluation of Deep Learning on Highway Driving." ArXiv abs/1504.01716 (2015): n. pag.

[5] Li, Bo, Tianlei Zhang and Tian Xia. "Vehicle Detection from 3D Lidar Using Fully Convolutional Network." ArXiv abs/1608.07916 (2016): n. pag.

[6] Chen, Chenyi, Ari Seff, Alain L. Kornhauser and Jianxiong Xiao. "DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving." 2015 IEEE International Conference on Computer Vision (ICCV) (2015): 2722-2730.

[7] Pomerleau, Dean. "ALVINN: An Autonomous Land Vehicle in a Neural Network." NIPS (1988).

[8] Bojarski, Mariusz, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao and Karol Zieba. "End to End Learning for Self-Driving Cars." ArXiv abs/1604.07316 (2016): n. pag.

[9] Szegedy, Christian, Alexander Toshev and Dumitru Erhan. "Deep Neural Networks for Object Detection." NIPS (2013).

[10] Sermanet, Pierre, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus and Yann LeCun. "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks." CoRR abs/1312.6229 (2014): n. pag.

[11] Yang, Shun, Wenshuo Wang, Chencheng Liu, Weiwen Deng and J. Karl Hedrick. "Feature analysis and selection for training an end-to-end autonomous vehicle controller using deep learning approach." 2017 IEEE Intelligent Vehicles Symposium (IV) (2017): 1033-1038.

[12] Drews, Paul, Grady Williams, Brian Goldfain, Evangelos Theodorou and James M. Rehg. "Aggressive Deep Driving: Combining Convolutional Neural Networks and Model Predictive Control." CoRL (2017).

[13] Katz, Greg, Abhishek Kumar Roushan and Abhijeet Shenoi. "Supervised Learning for Autonomous Driving." (2017).

[14] Du, Shuyang, Haoli Guo and Adam Simpson. "Self-Driving Car Steering Angle Prediction Based on Image Recognition." ArXiv abs/1912.05440 (2019): n. pag.

[15] Goyal, Raghav, Samira Ebrahimi Kahou, Vincent Michalski, Joanna Materzynska, Susanne Westphal, Heuna Kim, Valentin Haenel, Ingo Fründ, Peter Yianilos, Moritz Mueller-Freitag, Florian Hoppe, Christian Thurau, Ingo Bax and Roland Memisevic. "The ?Something Something? Video Database for Learning and Evaluating Visual Common Sense." 2017 IEEE International Conference on Computer Vision (ICCV) (2017): 5843-5851.

[16] Santana, Eder and George Hotz. "Learning a Driving Simulator." ArXiv abs/1608.01230 (2016): n. pag.

[17] Sallab, Ahmad El, Mohammed Abdou, Etienne Perot and Senthil Kumar Yogamani. "Deep Reinforcement Learning framework for Autonomous Driving." ArXiv abs/1704.02532 (2017): n. pag.

[18] Diederik P. Kingma Kingma, Diederik P. and Jimmy Ba. "Adam: A Method for Stochastic Optimization." CoRR abs/1412.6980 (2015): n. pag.

[19] Hochreiter, Sepp and Jürgen Schmidhuber. "Long Short-Term Memory." Neural Computation 9 (1997): 1735-1780.