Deep Neural Network-Based Perception for Autonomous Cars

1st Mukul Mahajan
(MS in Data Analytics)
San Jose State University
mukul.mahajan@sjsu.edu

2nd Vaibhav Prakash Shete
(MS in Data Analytics)
San Jose State University
vaibhavprakash.shete@sjsu.edu

3rd Tanvi Prashant Pagrut
(MS in Data Analytics)
San Jose State University
tanviprashant.pagrut@sjsu.edu

4th Akshay Sodha
(MS in Data Analytics)
San Jose State University
akshay.sodha@sjsu.edu

***Abstract*—The advancement of self-driving cars has transformed transportation by improving safety, minimizing expenditure, and making transportation easier. This work contributes to the development of autonomous systems by using convolutional neural networks and deep reinforcement learning techniques to enable navigation of unknown, dynamic environments. The perception module uses the Nvidia's End to End model to convert camera input into steering angle output as a regression problem, tackling augmented video data from Roboflow and KITTI to enhance robustness against varying conditions. Key performance indicators are mean square error (MSE), validation loss, inference speed. The Q-Learning and the Temporal Difference Learning, being the reasoning component, are embedded in the Markov Decision Process such that the agent learns to cope with driving tasks by means of specific reinforcement functions and driving policies. Performance measures include distance travelled without colliding, total rewards earned, and the instance recognition performance. In the future, the plans are to use multi-sensor information cueing and sequence-based CNN models for enhanced prediction. This research does not only advance autonomous vehicle technology through advanced solutions but also tackles global problems such as traffic accidents and mobility advancement.**

***Index Terms*—Autonomous Driving, Deep Reinforcement Learning, Nvidia End-to-End CNN, Q-Learning, PyTorch, Udacity Simulator.**

I. Introduction

The vehicle transportation system is about to undergo a paradigm shift with the introduction of autonomous cars which promises to enhance traffic management and enhance accessibility. But such ambitious goals come with their own hurdles including real time obstacle detection, route planning and learning in unfamiliar environments with no prior maps available. Most importantly, however, it is these challenges that classical path-planning or rule-based methodologies are well suited for to overcome, while all other problems are bypassed by using the new paradigm of deep learning and reinforcement learning approaches that offer the generality and reliability that is required in disorganized and unforeseen situations.

The focus of this report is on the integration of Convolutional Neural Networks (CNNs) and Deep Reinforcement Learning (DRL) to be able to have an autonomous vehicle that can perceive its environment and make the required decisions in real time in an efficient manner. CNNs are very good at understanding images such as maps, figuring out where locations are, and controlling actions such as the angle to turn the wheel. The visual architecture that we used features the Nvidia End-to-End architecture where our approach is based on the regression to drive the vehicle. Apart from that, the adaptation of the system is improved by the fact that it learns the best goal-directed behaviors by interacting with the environment via Q-Learning, Temporal Difference Learning and so on.

The datasets obtained from the Roboflow and KITTI repositories form the basis as the project showcases different driving scenarios like different weathers and lights. These datasets then go through preprocessing techniques like normalization, cropping, and resizing for model implementation and performance optimization. The model is made robust to such environmental variations by using data augmentation techniques such as brightness change, flipping, and adding noise.

The effectiveness of the system has been measured using key performance indicators such as Mean Squared Error (MSE) of the steering angle forecasts, total rewards obtained in the reinforcement learning activities and validation loss. Likewise, the coupling of reinforcement learning and the MDP based Bellman equations guarantees effective control strategies across a large range of driving scenarios.

Therefore, CNNs combined with DRL, are intended to broaden the scope of this study as it relates to the levels of perception and situational awareness of autonomous vehicle technologies. The aim is to develop systems for autonomous driving that are safer, more effective, and above all more flexible in terms of interaction with the complex features of the world.

II. Related Works

Hyperparameter optimization has emerged recently as a very important component in the quest for efficient and high-performance machine learning and reinforcement learning models. Various methodologies have been proposed to address the challenges associated with hyperparameter tuning, ranging from traditional techniques to advanced reinforcement learning-based approaches.

Qi and Xu [1] proposed a Q-learning-based framework for neural network hyperparameter optimization. Their work has demonstrated an improved search efficiency compared to conventional methods using the reinforcement learning paradigm. Correspondingly, Hansen [2] employed deep Q-learning for the dynamic control of hyperparameters like the learning rate, demonstrating adaptive tuning that is in line with model performance.

Yu and Zhu [3] reviewed the algorithms for optimization of hyperparameters along with their applications. This review highlighted the importance of understanding the interplay of hyperparameters with model performance as it gives insight into strengths and pitfalls of various techniques.Caarls et al. [4] extended reinforcement learning hyperparameter tuning and emphasized that it provides a significant impact on improving the learning efficiency and convergence in reinforcement learning algorithms.

Dong et al. [5] introduced a reinforcement learning-based dynamical hyperparameter optimization approach that does continuous parameter tuning with much higher computational efficiency and adaptability. In the process of overcoming the challenges of reinforcement learning in hyperparameter tuning, Eimer et al. [6] identified effective metrics for evaluation that could lead to reliable optimization.

Substantial research effort has also been dedicated to the application of hyperparameter optimization in specific domains. For example, Dong et al. [7] applied Q-learning to the hyperparameter tuning for tracking scenarios. Basha and Kumar [8] conducted a survey on techniques for hyperparameter optimization specifically for deep neural networks. Chen et al. [9] developed an automated hyperparameter optimization framework that leveraged reinforcement learning in streamlining the optimization process.

More recently, Zhang et al. [10] employed Q-learning for adaptive configuration of structural hyperparameters and presented an effective, flexible, and scalable optimization framework. Jomaa et al. [11] suggested the Hyp-RL-a reinforcement learning-based approach for the hyperparameter optimization problem-and accentuated its utility in several diverse machine learning tasks. Kiran and Ozyildirim [12] further extended these efforts toward deep reinforcement learning applications, hence providing a domain-specific perspective on their own.

Franceschi et al. [13] presented a study on meta-learning for hyperparameter optimization, with much attention given to enhancing the generalization capability of machine learning models.

An RL-based hyperparameter tuning algorithm is proposed in Talaat and Gamel [14] and performed for convolutional neural networks where considerable enhancement was realized on model accuracies.

This work further emphasizes how important reinforcement learning is becoming in hyperparameter optimization, including different approaches used to handle methodological and computational challenges.

III. Methodology

A. *Proposed framework architecture*

1) *CNN*: The proposed framework enables autonomous driving using Nvidia's End to End Convolutional Neural Network to predict steering angles from input images in the Udacity simulator. CNN is designed to take in spatial data from images regarding road markings, lane positions, and curves. It is the mapping of visual features to continuous steering angle outputs that enables the car to steer with precision and stability. The network architecture consists of several convolutional layers learning hierarchical spatial patterns and then fully enclosed layering interpreting these features to produce the final steering angle.

The CNN generates continuous steering angles, hence there are no abrupt or violent moves in driving dynamics. The framework uses a Mean Squared Error loss function to minimize large prediction errors in the network, hence guaranteeing smooth transitions and nimble driving. The car should be able to adapt to different driving situations, changing weather or lighting, for example, by engaging in behavioral cloning, an important requirement toward replicating human driving. This behavior is crucial. Because of the model's capability to directly predict steering angles from images, the basis of this autonomous driving system was sound and the operation during testing was smooth.

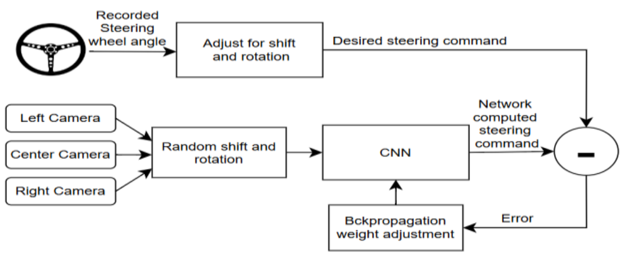
Recorded Steering wheel angle
Adjust for shift and rotation
Desired steering command
Left Camera
Center Camera
Right Camera
Random shift and rotation
CNN
Network computed steering command
−
Bckpropagation weight adjustment
Error

Fig. 1. Steering angle data before and after balancing

2) *Deep Q Learning (Reinforcement Learning)*: The proposed architecture employs Deep Q-Learning on reinforcement learning for intelligent decisions. The architecture includes key building blocks that focus on optimization of the learning and efficient navigation by the agent in dynamic environments.

It enables one of the central elements- Experience Replay, a mechanism where the model can learn from a fixed memory capacity of transitions rather than from only the current state. It is going to store the state-action pairs in the memory, the size of which was defined previously as 100,000 transitions and the event of an overflow of memory, the oldest entries would be deleted. During training, batches are randomly sampled from the memory to improve model generalization by introducing diversity in the learning samples. The sampled batches are then reshaped using lambda functions to align them with PyTorch variables for seamless integration.

This is the architecture for a neural network: the input size is five features, representing the state, comprised of three sensors and two orientations (left and right), while the output is three Q-values representing possible actions to go straight, and two turns. Here, non-linearity due to the ReLU activation ensures proper learning. A discounting factor of 0.9 on rewards, balancing the importance of immediate versus future rewards.

The framework is based on a Softmax Action Selection Policy with a high-temperature parameter T=100. This helps in exploration by smoothing the probability distribution over the Q-values so that the model can pick the optimal action while keeping some degree of uncertainty. Reward Mechanism: A sliding window of the last 100,000 rewards has been created for computing evolving means for assessing agent's performance.

The optimization is done by the Adam Optimizer, with a learning rate of 0.001, which allows for efficient weight adjustment in the process of backpropagation and reduces the problem of vanishing gradients. Other optimizers, such as RMSprop, were considered and tested for comparison purposes. Combining these elements, the agent learns to move around and act in a complicated environment.

It forms a robust architecture, especially in those tasks that require autonomous navigation and decision-making processes implemented in applications of reinforcement learning for self-driving systems.

B. *Data Collection and preprocessing*

1) CNN: The dataset used for this self-driving car framework consists of images taken from the car's front camera during its training in the Udacity simulator in 3 directions – center, left, right. This dataset simulates what the driver will be seeing and thus, it enables the model to perceive several important visual cues in this task of driving autonomously. Images are collected both at different times of the day - daylight and nighttime - under several weather conditions: clear, overcast, shadowed.

Each image is associated with driving data consisting of the metrics for steering angle, throttle, brake/reverse, and speed. The steering angle is the main output for training the model, making it directly related to the visual input and the response in terms of steering. The images are stored as JPEGs at 320x160 pixels whereas the driving data was stored in a CSV file for easy processing.

The distribution of steering angles was analyzed with a histogram, showing major imbalance as most data were concentrated around zero. This was expected since most driving in the simulator training runs is along the center of the road. A cut-off of 400 samples per bin was used to balance the dataset. The extra samples in overrepresented categories were removed and the resultant balanced dataset allowed the model to see enough variations in driving scenarios even for sharp turns.

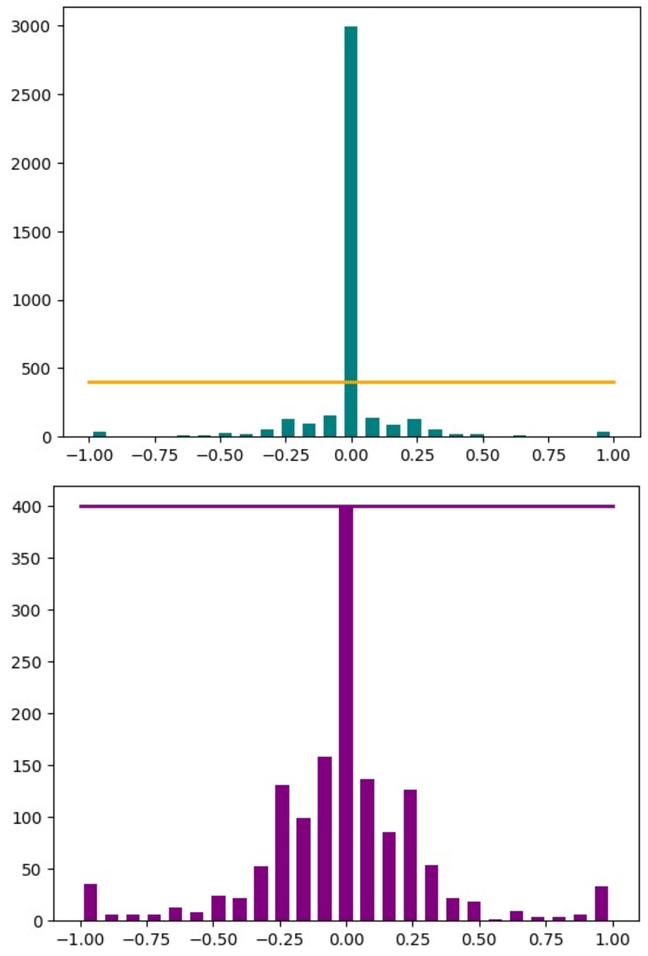

Fig. 2. Steering angle data before and after balancing

Data augmentation techniques were applied that helped in increasing the variety of the training images by introducing random zoom, pan, flip, and brightness in view to simulate real world-like changes in road perspectives, lighting conditions, and relative positions. These augmentations increase the complexity within the dataset, hence forcing the model to learn from previously unseen environments.

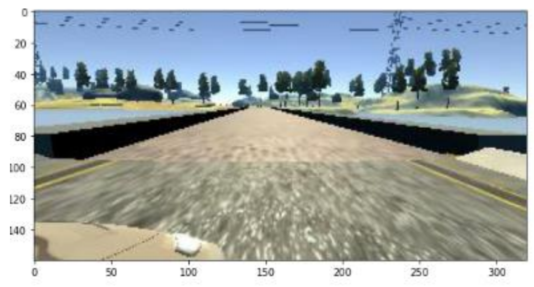

Fig. 3. Image before and after flipping

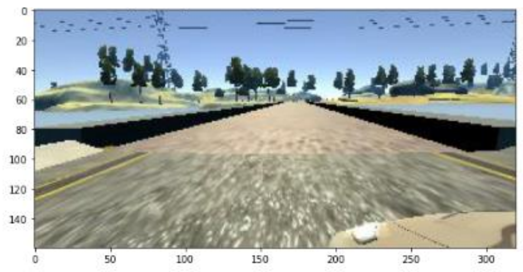

Fig. 4. Image before and after flipping

Further pre-processing steps included necessary transformations in order to prepare the images for neural network input. These involved cropping out parts of the images that were unnecessary for the road, including parts of the sky and the car dashboard. Then the images were converted into YUV color space and further smoothed with Gaussian blur to reduce noise. Each image was resized to confirm the input dimensions of the model and normalized in order to standardize the values of the pixels and ensure consistency across the dataset. Images were dynamically preprocessed and augmented during training, in order to reduce memory requirements while ensuring that the model was exposed to various scenarios during training.

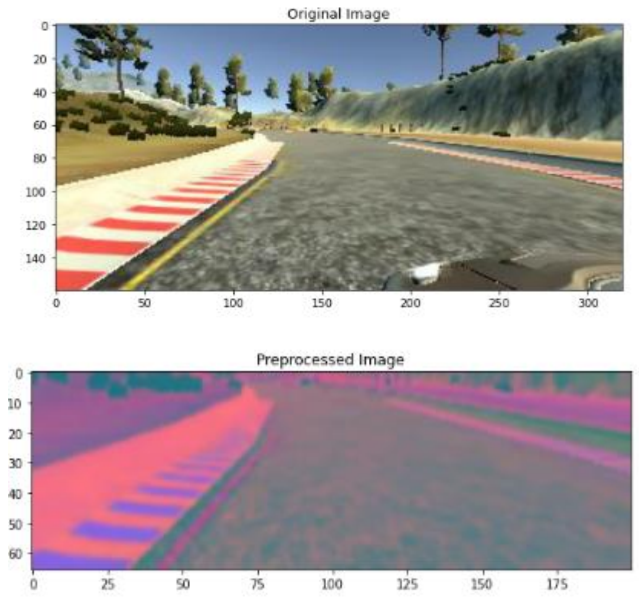

Fig. 5. Image before and after applying Gaussian Blur

2) *Deep Q Learning (Reinforcement Learning)*: The data collection and pre-processing stage involves organizing and transforming raw data into a form suitable for reinforcement learning using Deep Q-Learning. Three main sources of data feed into the input features for training of the neural network: sensor signals, orientation values (L & R), and actions.

Normalization starts the preprocessing pipeline, scaling all sensor signals to a consistent range. This ensures that at least all inputs are on one scale, avoiding large numeric variations that may dominate in the learning process and improve the speed of convergence. Through the normalization of sensor signals, the system effectively integrates diversified environmental data, thereby making the model robust over variations.

It follows a procedure which is enhanced with Batch Preparation by implementing experience replay method. A fixed memory capacity for transitions is kept at 100,000, storing state-action-reward sequences. When the number of transitions exceeds this capacity, older transitions are discarded to make way for newer ones. The random sampling of diverse batches from this memory improves generalization by exposing the model to various state-action pairs during training. These samples are then reshaped and converted into PyTorch variables to ensure compatibility with the deep learning framework. By employing experience replay, the model learns from the previously visited states rather than depending only on the most recent interactions, greatly improving training efficiency and stability.

Feature Engineering has also been done to improve the representation of the state. It combines sensor readings and orientation signals to make one input vector with the representation of spatial and directional contextual information of the environment. These are input features to the neural network where each vector has five components-three from sensors and two for orientation, positive and negative. This feature set is instrumental for the model to infer optimally based on its perception of the environment.

C. *Analysis and model selection*

1) *CNN*: To develop an accurate and reliable autonomous driving system, various deep learning approaches were reviewed, and the Nvidia End-to-End CNN was chosen because it has proven the ability to directly predict steering angles from input images. This architecture is specially designed for autonomous driving applications, using convolutional layers to extract spatial features from road images and fully connected layers to convert these features into a continuous steering angle.
The Nvidia CNN's ability to generalize in diverse conditions, coupled with simplicity and efficiency, made it a suitable choice. This regression-based output layer in the model was fine-tuned using a Mean Squared Error (MSE) loss function to ensure smooth, precise predictions with no abrupt changes in steering.Moreover, data augmentation and preprocessing helped to enhance the model for various driving scenarios. Such methods of zooming, panning, flipping, and changes in brightness enriched the set and prepared the model to perform quite well on real variations. The strong design of Nvidia's End-to-End CNN, besides being compatible with this type of dataset, confirmed that it would serve as the best architecture to consider for this initiative.
2) *Deep Q Learning (Reinforcement Learning)*: Deep Q-Learning has been chosen because it is really robust in predicting optimal actions concerning the current state vectors, hence aligning with the objectives of reinforcement learning. Several key components and hyperparameters have been involved in the architecture of the model so that effective learning and adaptation could be performed. Performance was explored by analyzing multiple optimizers. Where the Adam optimizer became most stable and efficient for the model, RMSProp was considered in terms of doing alternative optimization. It was observed that contribution during training in terms of convergence the ability of Adam was better to adapt learning rates for each individual parameter.

It relies on critical hyperparameters governing its learning dynamics. First, the discounting factor gamma, with a value of 0.9, means the model should learn to give higher importance to long-term rewards while valuing immediate gains. Then, the high temperature in the Softmax action selection policy of T = 100 assists in exploration by enforcing more variety in the actions taken during training. It enables the model to learn even from the actions it may not be sure about; hence, showed a better decision-making process.

Another very important ingredient is the experience replay, where the memory capacity was chosen as 100,000 transitions to enable the model to store a wide history of state-action-reward transitions and sample diverse batches from this during training. Every batch has 100 samples selected at random to ensure that varied learning without overfitting on the most recent experiences occurs. The reward mechanism depends on the average reward moving in the window of the last 100 iterations-a more dynamic, adaptive way of assessing performance.

The neural network architecture has an input size of 5, corresponding to the state vectors, with 3 Q-values for left, straight, and right, being the possible actions. Moreover, ReLU introduces non-linear activation to enhance the capability of the network in modeling richer patterns or relationships between input and output. These thoughtfully selected hyperparameters and components combine altogether for the robustness and efficiency achievable by the Deep Q-learning model in reinforcement learning tasks.

D. Experimental setup

1) CNN: In this research, we used an End-to-End Nvidia model architecture CNN whose goal is to predict the steering angle from the provided road images. The dataset was enhanced through some pre-processing steps which include normalization of steering angles, size reduction of images to 320 by 160 pixels and irrelevant complex areas such dashboard and sky which do not aid in the steering were cropped. To increase the informatics of the model, data augmentation techniques such as zoom, brightness cut, pan and horizontal flip were applied to make the model perform better while driving under different conditions.

For various batch sizes (16, 32, and 64) along with different learning rates (0.001, 0.0001, 0.00001), the model was trained for maximum convergence. Overfitting was managed with Dropout layers at 0.3 and 0.5. This training lasted for about three hundred epochs and the model successfully performed well with good performance improvement at every epoch number. The optimal result was achieved at 300 epochs yet for the first and the second epoch, moderate and almost poor results were obtained, respectively. Finally, to observe the prevention of overfitting along with the generalizability of the model for real-world problems, training and validation losses were computed.

2) *Deep Q Learning (Reinforcement Learning)*: The experimental environment of the proposed Deep Q-Learning model is designed for the training of the model to be carried out under the condition of being in a dynamic and severe environment. Drawing from the fundamental Q-Learning principle:

$$Q(s,a) = R(s,a) + \gamma \sum\_\{s'\}\hat{}\{\ \}(P(s,a,s') \max \_\{a'\}Q(s',a'))$$

A virtual map is built with real-world items, including dynamic goals and other types of hazards, in order to give realism to this virtual environment. These include simulated sand zones, for instance, considered a core element and critically important to drive manipulation of the type of terrain and enhance generalization across environments. The agent-based sequential decision-making model of Markov Decision Process (MDP) serves as a basis for the agent's decision-making process:

$$V(s) = \max_a\left(R(s,a) + \gamma \sum\nolimits_{s'}^{P}(s,a,s')V(s')\right)$$

Simulations were performed on hardware based on GPUs, were used not only in simulations but also for training and inference. Not surprisingly, the computational load for the forward and backward passes of the neural network depended crucially on the use of a GPU due to the large memory usage and the high volume of transitions considered in training. The model determines which direction it must take for optimum navigation. The update procedure for Q-value recalled the most important part in the learning process is given by:

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha\left(R(s,a) + \gamma \max_{a'} \quad Q(s',a') - Q_{t-1}(s,a)\right)$$

The neural network consisted of a hidden layer with 30 neurons and used the ReLU activation function in order to add some non-linearity and enhance the capability of this network to capture complicated patterns in the data. The Temporal Difference error contributed to update of the value estimations which is given by:

$$TD(a,s) = R(s,a) + \gamma \max_{a'} \quad Q(s',a') - Q_{t-1}(s,a)$$

There is a tendency towards higher robustness involving a plurality of augmentations, including the simulation of heterogeneous and different sand densities across heterogeneous ground. These augmentations add to the training data complexity but are guaranteed to generalize the model's learned behaviors to broader conditions. This experimental platform provides a complete solution from the beginning to the end for assessing the performance and robustness of the Deep Q-Learning model in challenging, dynamic, and frequently unseen or unanticipated scenarios.

E. Evaluation metrics

1) CNN: To assess the performance of the autonomous driving model, the following regression metrics were used: Mean Squared Error (MSE). This metric was chosen because it can measure the accuracy of continuous predictions, especially in tasks like steering angle prediction that calls for smooth control.

Mean Squared Error (MSE): MSE evaluates the average squared difference between the predicted and actual steering angles. It is calculated as:

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

2) *Deep Q Learning (Reinforcement Learning)*: It considers performance and stability evaluation metrics to deeply gauge the developed Deep Q-Learning model in the conducted research work. These metrics thus includes Q-value stability, reward accumulation, and convergence analysis, with each being indicative of the learning effectiveness from a different standpoint of view.

The Q-value stability has mostly been judged by the Huber Loss function, which strikes a balance between sensitivity to outliers and robustness. This loss enabled accurate assessment of how the model minimized prediction error by keeping the learning dynamics stable. The Huber Loss is thus the difference between the predictions made on Q-values and their targeted ones, hence serving in quantifying the consistencies across episodes in making such predictions.

The second important metric is the total reward received in multiple episodes by the agent, which signifies the average obtained reward. In other words, it indicates whether a model has managed to learn optimum actions regarding changing goals and obstacles. More importantly, it showed through the sliding mean of recent 100 rewards how the progressive improvement within the agent improves further on decision making. A higher gathering of reward was indicative that the model adapted well within the simulated environment and learned from it to reach its objectives with more efficiency.

Convergence analysis was also done to confirm stability in Q-values at each iteration, which is very essential in reinforcement learning systems. It basically shows that model learning does not diverge as it progresses with training. The study of the trend in the updates of Q-values proved the convergence of the model towards the optimal policy. On a similar note, this was crucial to find any possible instabilities or oscillations within the learning process that can reduce the generalization capability of the model.

To put together, these evaluation metrics gave a complete overview of the performance of the Deep Q-Learning model, making sure that the learning process was stable, effective, and could achieve the outcomes desired within the dynamic simulation environment. These metrics were important in the fine-tuning of hyperparameters and also in the validation of the robustness of the implemented architecture.

IV. Experimental Results

1) CNN: The experimental outcome of the Convolutional Neural Network (CNN) model for self-driving showed improvement in steering angle better prediction and adaptation to different driving environments. The model was measured against other metrics with Mean Square Error (MSE) as the primary metric that was carried out on both training and validation sets. The experiments exhibited a clear trend of decreasing error metrics during the stages, which meant more road features were being learned. After 100 epochs, the results were not all that wrong, the reason being the model was able to learn some basic features of the road. After 200 epochs, the performance of the model improved significantly, indicating that the model was able to perform better in generalization. After 300 epochs `the performance was at its best as there hence was little change suggesting that the model had reached a certain level of Peak level of accuracy, beyond which additional training would only lead to a minor improvement. Applicability of the model in practice was examined in, which included curves, obstacles and rough surfaces. The model broader applicability on other as it generated unseen slopes. However, strategies like dropout layers and adjusting learning rates were implemented to prevent over-fitting. The evaluation also included the usages of different batch sizes and loss functions to get a balance between computation cost and compute accuracy of the prediction. These results confirm the model's dependability with respect to steering control and can be used in a real time autonomous vehicle.

2) *Deep Q Learning (Reinforcement Learning)*: To find the best configuration for the Deep Q-Learning model, several hyperparameters were systematically tuned, and their performances were analyzed. The results are summarized in the experimentation plots. Each plot shows a model's learning curve-that is, the convergence toward maximizing rewards over episodes. We tested with multiple hyperparameter values and got mixed results.

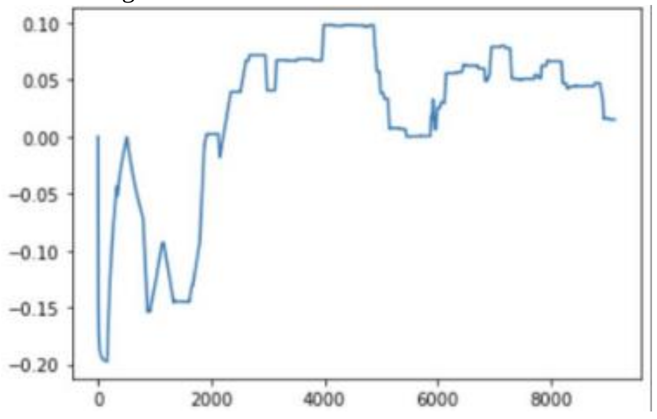

Fig. 6. Non Optimal Model 3 (*figure caption*)

Figure **6.** shows the model with best set of hyperparameters, we observed the best convergence and the highest reward. The neural network used an input size of 5, derived from 3 sensor signals across two orientations, with an action space consisting of three discrete actions: Left, Right, and Straight. The activation function was ReLU, with a discounting factor, gamma set to 0.9 to balance immediate and future rewards. A reward window was used to smooth the reward signal over the last 100000 episodes to aid in stable learning.

It also ensures that the replay memory has a capacity for 100,000 transitions for independent and diverse learning samples. The Adam optimizer was used with a learning rate of 0.001. Besides, the Softmax temperature parameter enabled controlled exploration by increasing or decreasing the confidence of selecting an action given the Q-values. The optimization of loss was done by Huber Loss due to its robustness in outliers.

The best convergence and highest reward are obtained by using this configuration, as revealed from the results. This validates the fact that hyperparameter tuning is a major step to ensure optimum performance in reinforcement learning models.

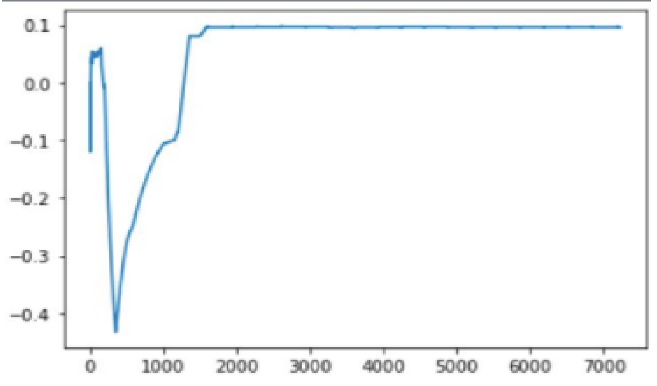

Fig. 7. Optimal Model (*figure caption*)

Limitation and future scope

This method is oftentimes very unreliable since it only uses camera input to approximate the steering angles, as a result the system has limited ability to be usable under different scenarios such as low lighting, weather conditions or object occlusions. Furthermore, the cause-and-effect relationship between the steering wheel inputs and the throttle and braking inputs is not clear, and so the system can be limited in its capability to moderate sudden driving situations like an abrupt object in its path or a tight corner. Furthermore, reliance upon balanced datasets poses a significant challenge because unbalance in datasets representing different driving situations can lead to biased prediction.

So as to further develop and address the limitations highlighted in this paper, future work can focus on expanding the framework to include throttle and braking inputs to achieve a more complete model of autonomous driving. Such expansions may allow testing the model in non-nominal driving tests that would serve to confirm the proposed framework's versatility and scalability. In future work, additional road traffic scenarios ought to be included in the training set to help the model overcome bias issues. The proposed model also provides opportunities to combine Convolutional Neural Networks (CNN) with Reinforcement Learning to increase temporal sequence learning by linking it to the road and dynamic behavior context – all this may serve as a benchmark for the future end-to-end solutions in driving systems.

V. Conclusion

The combination of Convolutional Neural Networks (CNNs) and Deep Reinforcement Learning (DRL) is a huge step towards making autonomous driving systems better. Utilizing the Nvidia End to End model, the automated system proposed is able to generate the correct steering angles thus making it easy to maneuver and drive smoothly in any surrounding environment. Data augmentation and preprocessing strategies were also used to increase the model's robustness and adaptability.

This work opens the avenues for using combined perception with combined decision-making modules to solve tasks in the real world, for example dynamic obstacle avoidance and route planning. Low Mean Squared Error (MSE) achieved during both the training and validation phases demonstrates the viability as well as the success of the framework. Subsequent work includes extending the model for brake and throttle prediction, its testing in real environments, and the inclusion of another layer of sensor data for better decision making.

TABLE I. Best Performing Model Summary

	Model Tuning (Deep Reinforcement Learning)	
	Best Hyperparameters	***Value***
1.	Softmax Temperature	100
2.	Discounting Factore	0.9
3.	Capacity	100000
4.	Learning Rate	0.001
5.	Loss	Huber (Smooth L1)

Appendix

A. *Code Walkthrough*

For the full code, please visit the GitHub repository: GitHub

References

[1] X. Qi and B. Xu, "Hyperparameter optimization of neural networks based on q-learning," *Signal, Image and Video Processing*, October 2022. Available: https://doi.org/10.1007/s11760-022-02377-y

[2] S. Hansen, "Using deep q-learning to control optimization hyperparameters," *arXiv preprint*, February 2016. Available: https://arxiv.org/abs/1602.04062

[3] T. Yu and H. Zhu, "Hyper-parameter optimization: A review of algorithms and applications," *arXiv preprint*, March 2020. Available: https://arxiv.org/abs/2003.05689

[4] W. Caarls, D. Jongbloed, and T. Van Heukelum, "Parameters tuning and optimization for reinforcement learning algorithms," in *Proceedings of INCISCOS*, 2018. Available: https://wouter.caarls.org/files/inciscos2018.pdf

[5] Y. Dong, J. Li, and R. Tang, "Dynamical hyperparameter optimization via deep reinforcement learning," in *IEEE International Conference on Big Data (Big Data)*, December 2019. Available: https://doi.org/10.1109/BigData47090.2019.9006022

[6] T. Eimer, M. Schüler, and P. Koppe, "Hyperparameters in reinforcement learning and how to tune them," in *Proceedings of Machine Learning Research*, 2023. Available: https://proceedings.mlr.press/v202/eimer23a.html

[7] Y. Dong, J. Liu, and L. Zhang, "Hyperparameter optimization for tracking with continuous deep q-learning," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. Available: https://openaccess.thecvf.com/content_cvpr_2018/papers/Dong_Hyperparameter_Optimization_for_CVPR_2018_paper.pdf

[8] S. R. Basha and N. Kumar, "Hyperparameter optimization for deep neural network models: A comprehensive survey," *Neural Computing and Applications*, 2023. Available: https://doi.org/10.1007/s11334-023-00540-3

[9] J. C. Chen, K. H. Lee, and A. Patel, "On automating hyperparameter optimization for deep learning," in *IEEE Signal Processing in Medicine and Biology Symposium (SPMB)*, 2021. Available: https://isip.piconepress.com/conferences/ieee_spmb/2021/papers/l02_06.pdf

[10] H. Zhang, J. Sun, and Z. Xu, "Adaptive structural hyper-parameter configuration by q-learning," *arXiv preprint*, March 2020. Available: https://arxiv.org/abs/2003.00863

[11] H. S. Jomaa, J. Grabocka, and L. Schmidt-Thieme, "Hyp-RL: Hyperparameter optimization by reinforcement learning," arXiv preprint, June 2019. Available: https://arxiv.org/abs/1906.11527

[12] M. Kiran and M. Ozyildirim, "Hyperparameter tuning for deep reinforcement learning applications," arXiv preprint, January 2022. Available: https://arxiv.org/abs/2201.11182

[13] L. Franceschi, M. Donini, V. Perrone, A. Klein, C. Archambeau, M. Seeger, M. Pontil, and P. Frasconi, "Hyperparameter optimization in machine learning," arXiv preprint, October 2024. Available: https://arxiv.org/abs/2410.22854

[14] F. M. Talaat and S. A. Gamel, "RL based hyper-parameters optimization algorithm (ROA) for convolutional neural network," Journal of Ambient Intelligence and Humanized Computing, vol. 14, pp. 13349–13359, March 2023. Available: https://doi.org/10.1007/s12652-022-03788-y

[15] X. Liu, J. Wu, and S. Chen, "Efficient hyperparameters optimization through model-based reinforcement learning with experience exploiting and meta-learning," Soft Computing, vol. 27, pp. 8661–8678, April 2023. Available: https://doi.org/10.1007/s00500-023-08050-x