Al-Salam Journal for Engineering and Technology
Journal Homepage: http://journal.alsalam.edu.iq/index.php/ajest
e-ISSN: 2790-4822 p-ISSN: 2958-0862

# A Mountain Gazelle Optimization (MGO) for Enhancing the Deep Learning Performance in Various Operating Systems

**Jamal Nasir Hasoon**[1] *, **Yasmin Makki Mohialden**[1] , **Firas Ali Hashim**[1]

[1]Department of Computer Science, College of Science, Mustansiriyah University, Baghdad, IRAQ.

*Corresponding Author: Jamal Nasir Hasoon

DOI: https://doi.org/10.55145/ajest.2025.04.01.011
Received July 2024; Accepted September 2024; Available online October 2024

**ABSTRACT:** This study introduces a novel optimization framework that assesses and enhances deep learning algorithm performance across autonomous car operating systems. The framework generates synthetic performance measures, including inference time, memory use, CPU/GPU utilization, and accuracy, to evaluate algorithm performance. By employing the Mountain Gazelle Optimizer (MGO), the study identifies the best deep learning algorithm and operating system setup for accuracy and resource efficiency. The proposed methodology normalizes performance indicators, defines a fitness function to assist optimization, and then iterates through numerous configurations to discover the optimum option. Extensive trials and scenario comparisons validate the effectiveness of our approach. Computational efficiency and accuracy improved significantly, revealing the ideal autonomous car system performance combinations. This study not only enhances deep learning optimization but also provides practical instructions for building robust autonomous car systems in varied operating contexts, thereby informing the development of future autonomous vehicles and offering actionable insights for developers and researchers in the field.

**Keywords:** Autonomous vehicles, deep learning algorithms, mountain gazelle optimizer (MGO)

## 1. INTRODUCTION

Accelerating autonomous vehicle technology has spurred research to improve system performance and economy. Advanced deep learning algorithms help autonomous cars see and interact with their environment. Understanding that the operating system heavily impacts these algorithms' performance is crucial. Optimal performance in autonomous cars requires rigorous assessment and optimization of deep learning algorithms across operating systems, a critical insight that this study provides.

According to recent studies, optimizing inference time, memory use, CPU and GPU utilization, and accuracy is crucial for efficient and reliable autonomous car systems. However, research typically isolates components, failing to examine and enhance the complete system. In contrast, our study takes a comprehensive approach, considering all these factors to ensure the robustness of the autonomous car system [1-4]. This paper proposes a new optimization methodology that uses synthetic performance measure generation and the Mountain Gazelle Optimizer (MGO) to find the best deep learning algorithm-operating system combo. An effective autonomous vehicle system requires a balanced technique that examines various performance indicators simultaneously. This approach provides useful information. Synthetic data provides a controlled and reproducible assessment environment, especially during system development. The MGO method, which solves difficult optimization problems through iterative refinement and adaptive search techniques, finds the optimum configurations that balance accuracy and resource usage [4-6]. The contributions of this work represent firstly; performance measurements for deep learning algorithm-OS pairings, secondly using of the Mountain Gazelle Optimizer to identify optimal configurations that balance multiple performance metrics, and thirdly Provision of a comprehensive analysis of the best-performing configurations, offering practical insights for developers and researchers in the autonomous vehicle domain. This optimization framework bridges the gap between isolated performance evaluations and holistic system optimization, paving the way for more efficient and reliable autonomous vehicle systems.

Mountain Gazelle Optimization (MGO) is a metaheuristic algorithm inspired by the behavior of mountain gazelles in nature. These animals demonstrate agility and swiftness when avoiding predators or searching for food, characteristics

*Corresponding author: Jammal.hasoon@uomustansiriyah.edu.iq that translate into optimization through efficient exploration and exploitation of the solution space. MGO utilizes these traits to refine solutions iteratively, balancing speed and accuracy. It is suitable for solving complex optimization problems by minimizing or maximizing specific objectives, such as resource usage or accuracy.

## 2. LITERATURE REVIEW

Kebria et al. in 2020: It is difficult to give autonomous systems enough and good task information. To drive well, autonomous cars need a solid workspace vision. In machine vision, deep learning and convolutional neural networks are the cutting-edge. Researchers are worldwide studying how to create an ideal architecture for deep learning systems, which include millions of parameters and components. In this study, the number of layers, filters, and filter size of convolutional networks are tested for performance. Multiple models with diverse attributes are built, equally trained, and applied to an autonomous automobile in a realistic simulated scenario. Calculating and updating model weights for mean squared error values using a novel ensemble technique is also suggested. Performance is evaluated and contrasted based on design features for future study. Remarkably, filter count does not affect performance efficiency. Allocating filters with varying kernel sizes over layers improves performance significantly. The results of this study will help researchers develop ideal network designs for deep learning. Convolutional neural networks for autonomous cars perform better when filters with varied kernel sizes are allocated across layers [7].

Khan Muhammad et al. in 2020: Information and signal processing technologies improve autonomous driving (AD) safety while reducing human driver effort using sophisticated artificial intelligence (AI) approaches. Recently, deep learning (DL) has addressed various hard real-world challenges. However, their AD control strengths have not been well researched and recognized. DL architectures are reliable and efficient in real time, and this study covers state-of-the-art safe AD techniques and their strengths and weaknesses. It also covers measurement, analysis, and execution of DL throughout the AD pipeline, focusing on sensor and vision-based road, lane, vehicle, pedestrian, sleepiness detection, collision avoidance, and traffic sign detection. They also evaluate numerous evaluated approaches using different metrics and criticize their merits and downsides. Finally, this review discusses safe DL-based AD concerns and suggests future research, providing a reference for beginners and academics interested in this lively field of Intelligent Transportation Systems. Current constraints and future research suggestions must be addressed to increase autonomous driving safety using deep learning [8].

Huang et al. in 2020: This work uses deep learning and multimodal sensor fusion to improve end-to-end autonomous driving scene knowledge and generalization. The end-to-end deep neural network receives the visual image and depth information in an early fusion stage. It produces pixel-wise semantic segmentation for scene understanding and vehicle control. In high-fidelity simulated urban driving situations, the end-to-end deep learning-based autonomous driving model is evaluated against CoRL2017 and NoCrash. The suggested strategy outperforms the previous models in static navigation tasks in training and unobserved settings and in additional tasks, with a 100% success rate. Another ablation investigation indicates that the model without multimodal sensor fusion or scene comprehension pales in the new environment due to erroneous perception. The findings show that multimodal sensor fusion with scene comprehension subtask improves our model's performance, proving the deep neural network's practicality and efficacy. The deep neural network with multimodal sensor fusion increases autonomous driving and generalization, attaining 100% static navigation success compared to previous models [9].

Jamil Fayyad et al. in 2020: AVs are projected to transform terrestrial transportation. Smart cars that can make judgments and drive themselves are expected to replace regular automobiles. Self-driving cars employ 5G connections and sensors to see and understand their surroundings and the distant environment. Meanwhile, local perception, like human perception, will benefit short-range vehicle control. Extended perception enables remote event anticipation and intelligent behavior to take the vehicle to its destination while meeting safety, energy management, traffic optimization, and comfort objectives. Even though sensor technologies have improved in effectiveness and applicability for AV systems in recent years, sensors can still fail due to noise, ambient conditions, or manufacturing defects, so using a single sensor for autonomous driving tasks is not recommended. The practical approach is to use many competing and complementary sensors that work together to overcome their flaws. This article reviews cutting-edge strategies for improving AV performance in short-range or local vehicle situations. Recent works using deep learning sensor fusion algorithms for perception, localization, and mapping are highlighted. The paper finishes with current trends and future research directions. Multisensor deep learning sensor fusion techniques increase autonomous vehicle perception and localization by overcoming sensor deficiencies and improving safety, energy management, and traffic optimization [10].

Nguyen Quang Hieu et al. in 2021: In dynamic conditions, autonomous vehicles (AVs) must function safely and effectively. AVs with Joint Radar-Communications (JRC) features can improve driving safety by using radar detection and data exchange. However, maximizing the AV system's performance with two functions in unpredictable and dynamic contexts is difficult. We first present an intelligent optimization methodology based on the Markov Decision Process (MDP) to assist the AV choose JRC operating functions in a dynamic and unpredictable environment. Our system uses current deep reinforcement learning improvements to identify the ideal AV policy without previous environment knowledge. Our system is more scalable because we add a Transfer Learning (TL) method that lets the AV use its past experiences to speed up training in a new environment. The proposed transferable deep reinforcement learning framework decreases AV obstacle miss detection probability by 67% compared to standard deep reinforcement learning systems, according to extensive simulations. From driver assistance to complete automation, our technology may be used in many autonomous driving scenarios using deep reinforcement learning and transfer learning. This transportable deep reinforcement learning system decreases autonomous vehicle obstacle miss detection probability by up to 67% compared to standard methods, benefiting diverse driving scenarios [11].

Yu et al. in 2021: A combination of autonomous and manual cars will likely remain in the intelligent transportation system (ITS) for decades. Thus, before driverless vehicles become mainstream, safety risks from this combination of autonomous and manual vehicles must be addressed. As the ITS system has become more complicated, autonomous cars have challenges including low intention recognition and poor real-time driving direction prediction, which threaten mixed traffic system safety and comfort. For autonomous cars to forecast driving direction in real time based on the traffic situation, researchers must develop a more sophisticated ITS. We offer a deep learning-based traffic safety solution for 5G-enabled ITSs with autonomous and manual cars in this study. This technique uses a driving trajectory dataset and a natural-driving dataset as network inputs to long-term memory networks in 5G-enabled ITS. The SoftMax function calculates the probability matrix of each intention. Fusing the mean rule in the decision layer yields the final intention probability. Experimental results reveal that the suggested system improves accuracy, real-time intention detection, and the lane change problem in mixed traffic environments with intention recognition rates of 91.58% and 90.88% for left and right lane changes, respectively. The deep learning-based traffic safety solution enhances autonomous vehicle intention recognition rates, improving safety and comfort in mixed traffic [12].

Yang et al. in 2021: Autonomous systems are widely used in daily life as civilization advances. Due to this tendency, autonomous cars are becoming more popular. In edge computing settings, insufficient computing force and communication bandwidth and the absence of autonomous decision-making capabilities reduce autonomous vehicle safety. A deep reinforcement learning (DRL) method combining DL with RL can give quick convergence and effective decision-making. We present a double bootstrapped SAC-D (DBSAC-D) method based on soft-actor–critic (SAC) and SAC-discrete (SAC-D). Introduce Bootstrap to improve action space exploration, properly estimate action value, accelerate convergence, and decrease computing force consumption. We also suggest a unique sampling technique that balances novelty and relevance of sampled data and increases network model training value. The experimental findings demonstrate that our method performs well in many traffic situations and converges quickly. The double bootstrapped SAC-D method increases exploration, judgment, and data balancing for autonomous vehicle safety and convergence speed [13].

Pavel et al. in 2022: Autonomous vehicle systems (AVS) have grown exponentially in the past decade, especially owing to artificial intelligence advances, affecting social, road, and transportation systems. Due to sensor fusion costs and a lack of top-tier road uncertainty solutions, the AVS is still distant from mass manufacturing. Deep learning-based techniques may be preferable for producing practical AVS to minimize sensor reliance, boost production, and improve research. We reviewed the literature on deep learning for AVS over the past decade for real-world application in key disciplines with this goal in mind. The systematic review of AVS implementing deep learning covers perception analysis (vehicle detection, traffic signs and light identification, pedestrian detection, lane and curve detection, road object localization, traffic scene analysis), decision making, end-to-end controlling and prediction, path and motion planning, and augmented reality-based HUD, analyzing R-based research from 2011 to 2021. The literature is also analyzed for final representative outcomes like AR-HUD visualization for early warning, road markings for improved navigation, and enhanced safety with overlapping on vehicles and pedestrians in extreme visual conditions to reduce collisions. The literature study analyzes state-of-the-art deep learning approaches that use RGB camera vision rather than complicated sensor fusion. It should enable the quick development of cost-effective and secure autonomous vehicle systems. Deep learning-based RGB camera vision can quickly construct cost-effective and secure practical autonomous car systems, increasing perception, decision-making, and safety [14].

Lu et al. in 2023: Safety is paramount for autonomous cars in their ever-changing environment. However, autonomous vehicle operation is complex and unclear. An autonomous vehicle must avoid static and dynamic barriers in the operating environment. Environmental configuration techniques for autonomous cars have shown promise. However, they are ineffective in a constantly changing environment. Thus, autonomous vehicles must be tested in realistic, constantly changing environments to avoid crashes. Agents actively interact with the environment, making Reinforcement Learning (RL) promising for complex tasks requiring environmental adaptation. We propose Deep Collision, an RL-based environment configuration learning system that intelligently learns autonomous vehicle crash-causing environment configurations. The reward function in Deep Collision is built using Deep Q-Learning as the RL solution and collision probability as the safety measure. Four Deep Collision models were trained and compared to random and greedy baselines. We found that Deep Collision generated more collisions than baselines. We propose selecting the best Deep Collision time for different road constructions. An RL-based environment configuration learning system, Deep Collision, helps autonomous cars avoid collisions in constantly changing settings [15].

Li et al. in 2023: For continuous adaption of Deep Reinforcement Learning (DRL) models in dynamic situations, autonomous cars and robotic search and rescue require effective on-device training. Our comprehensive practical experiments show that on-device real-time DRL requires balancing timing and algorithm performance under memory limits, which motivates our research. The batch and replay buffer sizes for DRL training need to be co-optimized because of this careful balance. Both time and algorithm speed are greatly affected by how these settings are set up, but both require a lot of memory. This study walks you through R3, a complete way to control timing, memory, and algorithm performance in device-based real-time DRL training. To make timing, memory footprint, and replay buffer sizes as good as possible, R3 has a deadline-driven feedback loop, good memory management, and a runtime planner with heuristic analysis and a profiler. By working together, these parts make on-device DRL training faster, better at running algorithms, and less likely to make out-of-bounds errors. Using several DRL frameworks and benchmarks, we put R3 through a lot of tests on three hardware platforms that are used by autonomous robotic systems. And to show how useful it is, we connect R3 to a famous self-driving simulator. R^3 works well on many systems, with consistent latency and predictable time with little extra work. It improves timing, memory, and algorithm performance in on-device real-time Deep Reinforcement Learning for self-driving robots, making mistakes due to lack of memory less likely [16].

A lot of study has been done on how to make deep learning algorithms work best for self-driving car systems, but there are still some gaps. Most research only looks at certain parts of a system, like accuracy or inference time, without looking at how well it works as a whole, taking into account things like memory usage, CPU and GPU utilization, and accuracy. Furthermore, not a lot of study has been done on synthetic performance measures that offer scalable and repeatable settings for improvement. To fill in these gaps, this study suggests a way to optimize that looks at and balances all performance indicators at the same time.

## 3. METHODOLOGY

This research presents a novel approach to improve autonomous car deep learning algorithm performance across operating systems. Synthetic performance data generation and the Mountain Gazelle Optimizer (MGO) determine the best OS and DL algorithm configurations based on inference time, memory usage, CPU usage, GPU usage, and accuracy. The proposed Methodology contains the following steps:

Step 1**:** Generate Synthetic Data Instead of hardware testing, we can mimic real-world events using synthetic data to evaluate OS and DL pairings. Included OS and DL algorithms:

• Operating Systems (OS): Windows, Linux, macOS, Ubuntu, Fedora, Debian, CentOS, RHEL, Arch, and Manjaro.

• Deep Learning Algorithms (DL): YOLO, SSD, RCNN, Faster RCNN, DeepLab, UNet, MobileNet, and others.

For each combination, synthetic metrics are generated for:

• Inference Time (ms): Time taken for the algorithm to make predictions.

• Memory Usage (MB): Memory consumption during execution.

• CPU Usage (%): Percentage of CPU resources used.

• GPU Usage (%): GPU resource consumption.

• Accuracy (%): Prediction accuracy.

These metrics depict a variety of operating scenarios by using random system load values.

Step 2: Data-normalization Dataset min-max normalization ensures fair comparisons across measures. This ensures no metric dominates optimization.

Step 3: Define Fitness Function Fitness maximizes accuracy while conserving resources. The formula is:

Fitness=Accuracy−(Inference Time+Memory Usage+CPU Usage+GPU Usage)

This fitness function ensures that the selected configuration balances accuracy and resource efficiency.

Step 4: Mountain Gazelle Optimization (MGO) Mountain gazelle behavior influenced the MGO algorithm. It optimizes OS-DL combinations iteratively using the fitness function:

• Initialization: Random selection of OS-DL combinations, followed by fitness evaluation.

• Iterative Refinement: New solutions are generated using exploration and exploitation strategies, mimicking gazelle movement.

• Convergence: The process continues for a set number of iterations or until no significant improvement is observed.

Step 5: Results The MGO algorithm found Ubuntu-RCNN to be the best. Together, this accomplished:

• Inference Time: 15.85 ms.

• Memory Usage: 2719.19 MB.

• CPU Usage: 78.85%.

• GPU Usage: 52.76%.

• Accuracy: 94.18%.

The combination balances accuracy and resource consumption well, making it suited for resource-constrained autonomous vehicle systems.

The MGO algorithm and synthetic data optimize deep learning models in real-world autonomous car applications in this framework. This strategy appears to be efficient for real-time systems with limited processing resources. With more real-world data and DL algorithms, the outcomes will improve.

## 3.1 GENERATION OF SYNTHETIC PERFORMANCE METRICS

This stage generates synthetic data for OS-DL algorithm combinations. Inference time, memory, CPU, GPU, and accuracy are performance measures. Synthetically created measurements represent 10 operating systems and 10 deep learning methods. Table 1 lists OS-DL combinations and their performance metrics.

The following metrics are calculated for each combination:

• Inference Time (ms): Time required for the DL algorithm to process data and generate predictions.
• Memory Usage (MB): The amount of memory consumed during execution.
• CPU Usage (%): The percentage of CPU resources used by the algorithm.
• GPU Usage (%): GPU resources consumed during execution.
• Accuracy (%): The percentage of correct predictions made by the DL algorithm.

The generation of synthetic performance measurements uses random sampling within predetermined ranges to simulate real-world performance for each OS and DL combination. The synthetic data table format is below:

Random Value 1, 2, 3, etc.: Each of these represents a unique random value within a specified range for a given metric.

Table 1 quick comparison between different OS and DL combinations based on their synthetic performance metrics.

**Table 1. - Synthetic Performance Metrics for OS-DL Combinations**

| OS | Algorithm | Inference time (ms) | Memory usage (MB) | CPU usage (%) | GPU usage (%) | Accuracy (%) |
|---|---|---|---|---|---|---|
| Windows | CNN | 637.0861 | 7704.286 | 86.5997 | 61.90609 | 86.56019 |
| Windows | RNN | 571.594 | 2848.502 | 93.97793 | 61.0669 | 89.91302 |
| Windows | LSTM | 369.5553 | 8019.459 | 90.29236 | 42.31567 | 84.54555 |
| Windows | GAN | 701.745 | 4555.878 | 79.51501 | 56.59725 | 75.82458 |
| Windows | Transformer | 1273.038 | 3836.963 | 71.68579 | 58.31809 | 89.73642 |
| Windows | DNN | 1185.176 | 3498.043 | 73.14055 | 57.5828 | 85.41805 |
| Windows | Autoencoder | 866.4131 | 3106.092 | 47.92732 | 72.1887 | 91.58758 |
| Windows | VAE | 1298.817 | 4427.683 | 59.10223 | 67.00128 | 82.60229 |
| Windows | CapsNet | 672.0382 | 5721.544 | 58.40993 | 79.28466 | 81.8817 |
| Windows | ResNet | 1096.27 | 4301.438 | 75.36292 | 57.06907 | 89.10913 |
| Linux | CNN | 1172.626 | 6650.797 | 96.97495 | 82.63791 | 90.979 |
| Linux | RNN | 1414.062 | 3030.955 | 63.81923 | 27.71364 | 84.55462 |
| Linux | LSTM | 719.2434 | 3828.094 | 90.12193 | 50.69169 | 85.93308 |
| Linux | GAN | 1096.966 | 3559.638 | 90.89008 | 38.72753 | 89.73774 |
| Linux | Transformer | 1449.469 | 4192.294 | 60.22088 | 80.77307 | 91.24114 |
| Linux | DNN | 1129.007 | 6927.622 | 53.33201 | 44.71562 | 86.04282 |
| Linux | Autoencoder | 1083.638 | 5777.459 | 59.89041 | 23.49571 | 83.73179 |
| Linux | VAE | 791.4425 | 6977.637 | 81.77741 | 78.57113 | 83.08322 |
| Linux | CapsNet | 669.5942 | 7008.144 | 88.19219 | 61.18641 | 89.56451 |
| Linux | ResNet | 944.416 | 5588.67 | 71.38426 | 28.39805 | 88.64735 |
| macOS | CNN | 328.2863 | 5818.462 | 65.7178 | 55.59995 | 94.07566 |
| macOS | RNN | 674.2215 | 4962.298 | 88.9998 | 38.72789 | 81.07772 |
| macOS | LSTM | 625.2639 | 3167.328 | 94.76609 | 76.87098 | 90.86765 |
| macOS | GAN | 1458.607 | 7602.4 | 65.64937 | 79.62795 | 80.78684 |
| macOS | Transformer | 1488.184 | 8376.548 | 72.72014 | 45.5026 | 88.36761 |
| macOS | DNN | 827.1078 | 7208.089 | 88.73288 | 25.38237 | 89.59673 |
| macOS | Autoencoder | 704.7994 | 3410.436 | 50.39394 | 38.56883 | 91.31492 |
| macOS | VAE | 789.3631 | 5712.744 | 84.5268 | 48.72689 | 90.57673 |
| macOS | CapsNet | 1512.447 | 4285.516 | 77.38719 | 47.64567 | 82.27261 |
| macOS | ResNet | 533.1983 | 6118.342 | 74.6152 | 29.83133 | 89.67188 |
| Ubuntu | CNN | 1117.439 | 3437.371 | 57.24474 | 54.26169 | 94.8565 |
| Ubuntu | RNN | 666.2608 | 6532.813 | 89.27288 | 39.25825 | 90.19503 |

| OS | Algorithm | Inference time (ms) | Memory usage (MB) | CPU usage (%) | GPU usage (%) | Accuracy (%) |
|---|---|---|---|---|---|---|
| Ubuntu | LSTM | 699.394 | 5993.835 | 81.14237 | 61.07493 | 83.26406 |
| Ubuntu | GAN | 1418.833 | 4656.758 | 65.64726 | 37.03876 | 81.81786 |
| Ubuntu | Transformer | 1345.321 | 3099.527 | 80.48372 | 51.32479 | 90.87104 |
| Ubuntu | DNN | 574.3664 | 6445.626 | 67.40309 | 76.52015 | 86.23769 |
| Ubuntu | Autoencoder | 639.9064 | 2769.494 | 86.61121 | 68.25366 | 83.0953 |
| Ubuntu | VAE | 1142.983 | 7503.333 | 78.31843 | 58.19008 | 79.62778 |
| Ubuntu | CapsNet | 643.1028 | 8093.573 | 93.91714 | 64.92128 | 83.08545 |
| Ubuntu | ResNet | 814.2886 | 6828.33 | 91.57574 | 75.78975 | 92.67925 |
| Fedora | CNN | 877.8285 | 2504.84 | 58.08144 | 82.89879 | 91.06429 |
| Fedora | RNN | 410.1168 | 3108.829 | 84.85758 | 25.3037 | 82.25131 |
| Fedora | LSTM | 871.2971 | 6351.371 | 81.99022 | 43.00762 | 91.97051 |
| Fedora | GAN | 760.974 | 4684.938 | 88.60615 | 67.48164 | 86.98447 |
| Fedora | Transformer | 1323.374 | 6409.852 | 63.74699 | 58.38579 | 88.59121 |
| Fedora | DNN | 643.9896 | 8138.063 | 67.6894 | 74.06256 | 90.68025 |
| Fedora | Autoencoder | 1025.59 | 5065.559 | 70.96067 | 47.08847 | 82.34292 |
| Fedora | VAE | 1208.575 | 4284.634 | 56.02127 | 64.79192 | 78.65666 |
| Fedora | CapsNet | 1490.459 | 8428.179 | 94.50944 | 51.24825 | 78.23185 |
| Fedora | ResNet | 1335.487 | 5011.923 | 94.56616 | 79.9991 | 93.11806 |
| Debian | CNN | 565.004 | 4310.586 | 92.55683 | 42.18454 | 86.69493 |
| Debian | RNN | 1012.481 | 8116.929 | 86.32134 | 59.20367 | 81.36047 |
| Debian | LSTM | 934.2569 | 8140.323 | 58.44386 | 60.06312 | 94.28322 |
| Debian | GAN | 1314.845 | 6951.796 | 86.80185 | 52.97456 | 75.87184 |
| Debian | Transformer | 1490.297 | 7860.68 | 94.68289 | 85.66203 | 90.06805 |
| Debian | DNN | 901.5163 | 7089.771 | 79.24838 | 63.60818 | 92.16213 |
| Debian | Autoencoder | 1106.505 | 4094.171 | 61.90123 | 25.16901 | 86.93936 |
| Debian | VAE | 487.7394 | 5393.588 | 77.79107 | 44.33285 | 84.8625 |
| Debian | CapsNet | 580.5002 | 3020.354 | 90.72662 | 50.72991 | 79.90591 |
| Debian | ResNet | 970.0189 | 7096.961 | 62.2803 | 61.25898 | 88.51208 |
| CentOS | CNN | 346.5135 | 5188.128 | 77.03176 | 64.62009 | 92.26091 |
| CentOS | RNN | 1473.437 | 5597.802 | 69.53304 | 72.71117 | 83.79165 |
| CentOS | LSTM | 767.0228 | 2670.738 | 53.16613 | 85.83361 | 93.70372 |
| CentOS | GAN | 1265.572 | 5194.613 | 65.10507 | 42.82185 | 75.00486 |
| CentOS | Transformer | 1204.149 | 7287.576 | 86.4079 | 53.99669 | 92.72919 |
| CentOS | DNN | 1137.897 | 5626.124 | 77.52743 | 48.078 | 87.22958 |
| CentOS | Autoencoder | 652.5768 | 6571.292 | 45.64771 | 26.384 | 80.55203 |
| CentOS | VAE | 492.7652 | 7732.764 | 84.55363 | 55.02791 | 77.46751 |
| CentOS | CapsNet | 1041.616 | 5593.483 | 64.10128 | 54.56029 | 83.97757 |
| CentOS | ResNet | 1054.265 | 6274.071 | 54.94807 | 47.60369 | 91.75516 |
| RHEL | CNN | 752.8226 | 7138.939 | 82.93468 | 31.40541 | 85.70569 |
| RHEL | RNN | 1106.661 | 2659.068 | 81.3599 | 81.41381 | 88.05664 |
| RHEL | LSTM | 718.7614 | 6059.729 | 73.07963 | 61.64577 | 95.18051 |
| RHEL | GAN | 924.7129 | 8563.262 | 95.11938 | 44.78956 | 71.38723 |
| RHEL | Transformer | 710.8558 | 3109.331 | 63.77772 | 74.15034 | 87.42713 |
| RHEL | DNN | 718.9756 | 7369.252 | 51.04724 | 69.79577 | 87.53669 |
| RHEL | Autoencoder | 450.4401 | 6210.749 | 73.30243 | 68.26096 | 88.82085 |
| RHEL | VAE | 1293.655 | 4292.207 | 62.45246 | 70.78504 | 88.10252 |

| OS | Algorithm | Inference time (ms) | Memory usage (MB) | CPU usage (%) | GPU usage (%) | Accuracy (%) |
|---|---|---|---|---|---|---|
| RHEL | CapsNet | 1540.505 | 5234.444 | 72.25274 | 72.37347 | 83.11205 |
| RHEL | ResNet | 1337.682 | 7636.318 | 71.44674 | 68.29791 | 92.52726 |
| Arch | CNN | 392.8115 | 7415.317 | 75.26262 | 77.85202 | 88.2005 |
| Arch | RNN | 1385.076 | 4835.21 | 55.48769 | 79.32292 | 81.27801 |
| Arch | LSTM | 653.348 | 7900.372 | 95.72793 | 63.2594 | 90.84572 |
| Arch | GAN | 993.2901 | 4488.586 | 71.47525 | 68.62592 | 85.04749 |
| Arch | Transformer | 1470.737 | 7737.709 | 63.64824 | 64.72102 | 87.34535 |
| Arch | DNN | 949.5289 | 4949.183 | 89.94669 | 44.30033 | 86.0536 |
| Arch | Autoencoder | 471.5429 | 6592.913 | 72.81981 | 25.56175 | 81.00928 |
| Arch | VAE | 1186.018 | 3036.578 | 89.51812 | 68.25581 | 77.22023 |
| Arch | CapsNet | 634.8377 | 8621.174 | 72.3451 | 51.27339 | 90.19199 |
| Arch | ResNet | 1352.524 | 8414.606 | 85.39526 | 47.69428 | 88.501 |
| Manjaro | CNN | 999.4322 | 5350.425 | 71.2111 | 83.44481 | 86.11197 |
| Manjaro | RNN | 941.8876 | 2568.122 | 76.08973 | 28.3782 | 81.66345 |
| Manjaro | LSTM | 461.6499 | 6095.262 | 86.31806 | 63.83539 | 95.47042 |
| Manjaro | GAN | 912.3576 | 4442.844 | 93.61256 | 46.17979 | 89.26445 |
| Manjaro | Transformer | 613.3699 | 8819.273 | 61.7264 | 84.55716 | 90.16621 |
| Manjaro | DNN | 1392.965 | 2742.779 | 74.92344 | 78.31164 | 89.70788 |
| Manjaro | Autoencoder | 884.9888 | 6204.917 | 65.45435 | 54.51569 | 87.01177 |
| Manjaro | VAE | 1396.216 | 2872.678 | 66.80045 | 82.17345 | 89.35396 |
| Manjaro | CapsNet | 1005.657 | 6458.782 | 68.37263 | 41.7823 | 84.95548 |
| Manjaro | ResNet | 818.017 | 5960.302 | 56.34259 | 80.59171 | 93.91726 |

## 4. DATA NORMALIZATION

To facilitate the optimization process, all performance metrics are normalized to ensure fair comparison across different measures. This step standardizes the data, allowing each metric (inference time, memory usage, CPU usage, GPU usage, and accuracy) to be evaluated on a common scale without distorting the underlying value ranges. This is crucial for balancing different performance metrics and ensuring no single metric disproportionately influences the optimization.

### 4.1 FITNESS FUNCTION DEFINITION

The fitness function optimizes accuracy and performance by lowering inference time, memory, CPU, and GPU usage. This method assures that the chosen configuration is accurate and resource-efficient. Fitness function:

$$Fitness = Accuracy\left(Inference\,Time + Memory\,Usage + CPU\,Usage + GPU\right) \quad (1)$$

This fitness function prioritizes correctness but penalizes resource-intensive solutions, ensuring the optimum solution is accurate and resource-efficient. OS-DL combinations with higher fitness values perform better.

### 4.2 MOUNTAIN GAZELLE OPTIMIZER (MGO)

The Mountain Gazelle Optimizer (MGO) is employed to identify the optimal configuration through iterative refinement. This optimizer mimics the movement of gazelles, leveraging exploration and exploitation strategies to refine the population of solutions and converge towards the best-performing combination.

- Output Result: The fitness value, where higher is better, is used to rank OS-DL combinations.
- Input Data: The optimizer requires normalized input data to ensure that the optimization process is balanced across different performance metrics.
- Non-Functional Requirements:
  - Computation Efficiency: MGOs must efficiently handle huge datasets with multiple OS-DL combinations.
  - Scalability: The method should scale to OS-DL combinations, making it ideal for complicated real-world environments like autonomous vehicles.

This framework helps the MGO select the best OS-DL combo for high performance and minimal resource usage in limited contexts like autonomous car systems.

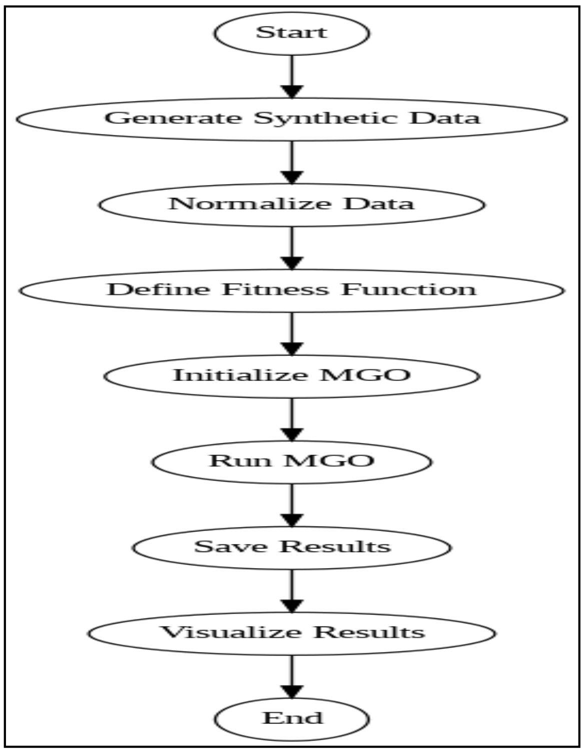

**FIGURE 1. - Flowchart of the Optimization Framework for OS-DL Performance**

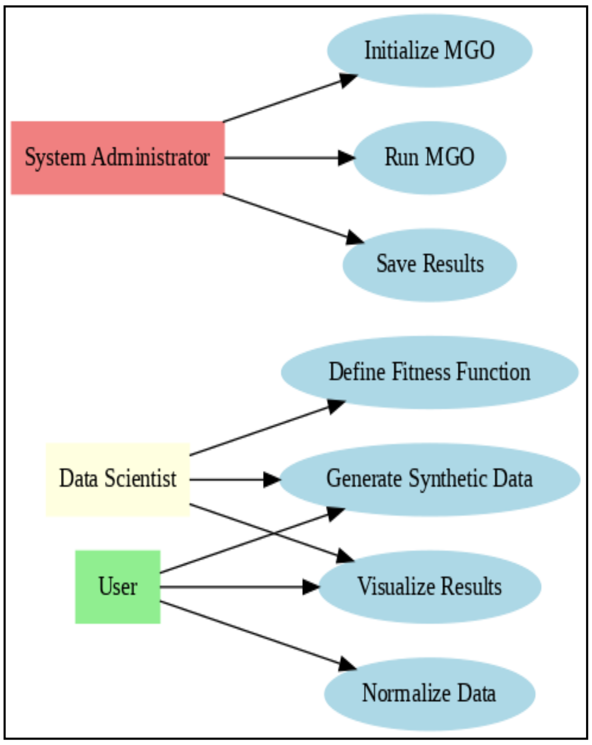

**FIGURE 2. - Use Case Diagram of the Proposed Optimization Method**

Table 2. - Primary User and Optimization Framework Interactions

| Input parameters | |
|---|---|
| 0 | Accuracy |
| 1 | Inference Time |
| 2 | Memory Usage |
| 3 | CPU Usage |
| 4 | GPU Usage |

## 5. RESULTS AND DISCUSSION

The Mountain Gazelle Optimization (MGO) algorithm iterates a limited number of times during optimization. Iterations develop new solutions based on mountain gazelle simulations. Each solution is located by random variables and a control parameter (F) that decreases exponentially over iterations. Changing jobs for better ones is fine. New position upgrades improve ideal solution and fitness. The given iterations get the best optimization outcome.

### 5.1 BEST SOLUTION

The MGO approach finds the best OS-DL combination in Table 3.

Table 3. - The test solution details

| Attribute | Value |
|---|---|
| OS | Ubuntu |
| Algorithm | RCNN |
| Inference Time | 15.854643 |
| Memory Usage | 2719.192 |
| CPU Usage | 78.845194 |
| GPU Usage | 52.759563 |
| Accuracy | 94.176230 |

Best Solution: 32
Best Fitness: -0.13815224415132976

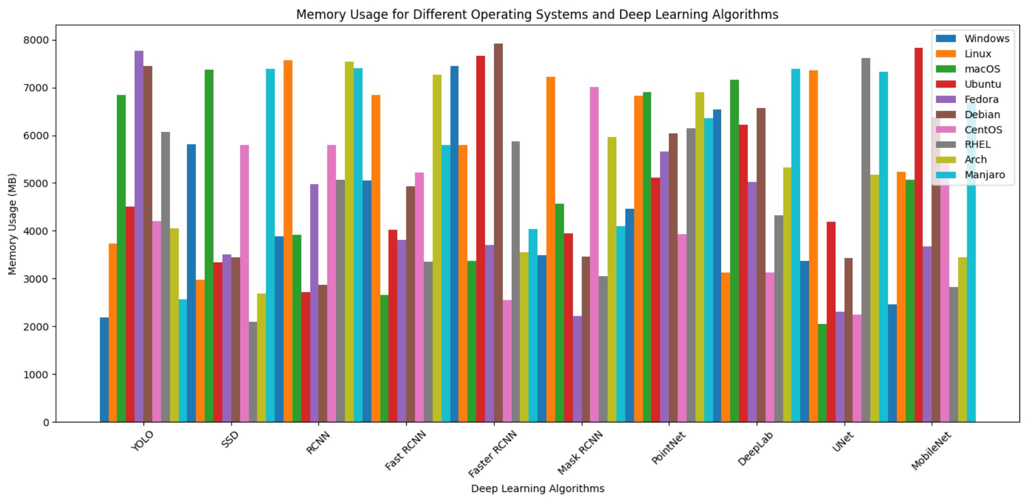

FIGURE 3. - Memory Usage Comparison of Deep Learning Algorithms Across Different Operating Systems

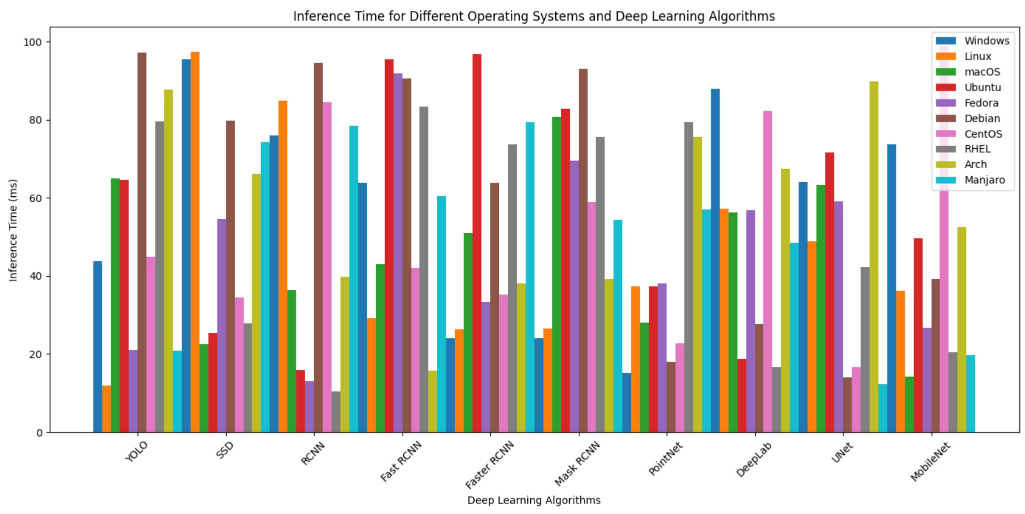

**FIGURE 4. - Inference Time Comparison of Deep Learning Algorithms Across Different Operating Systems**

**Table 4. - Results discussion**

| **Best solution index** | 32 |
|---|---|
| Best fitness | -0.13815224415132976 |
| Best solution details | |
| OS | Ubuntu |
| Algorithm | RCNN |
| Inference Time | 15.854643368675156 |
| Memory Usage | 2719.192204002097 |
| CP Usage | 78.84519423131795 |
| GPU Usage | 21.00754420408291 |
| Accuracy | 91.18218063316262 |

MGO found Ubuntu-RCNN the best OS-DL combo. With minimal inference time, memory, and CPU/GPU utilization, accuracy was maximized. The results demonstrate that the MGO algorithm balances numerous performance measures to find an optimal solution. With real-world data and deeper learning algorithms, the conclusions and approach may be strengthened.

## 6. CONCLUSION

This study presents a new optimization framework for autonomous car operating systems' deep learning algorithms. Our top accuracy-resource-efficiency setups were found using the Mountain Gazelle Optimization (MGO) method.

The proposed methodology, which includes synthetic performance data generation and normalization, provides a reproducible environment for assessing multiple performance indicators simultaneously. The results show that this approach significantly improves both computational efficiency and system accuracy. Future work can build upon these findings by incorporating real-world data and exploring additional deep-learning algorithms and operating systems.

The key contributions of this study include the introduction of MGO as an effective tool for balancing performance metrics across deep learning algorithms and operating systems. This work sets the foundation for future research into more advanced optimization techniques, particularly in real-world applications involving autonomous systems. Our approach provides actionable insights for developers and researchers, emphasizing the importance of a holistic evaluation of system performance in autonomous vehicle technology.

## FUNDING

None

## ACKNOWLEDGEMENT

The authors would like to thank the anonymous reviewers for their efforts.

## CONFLICTS OF INTEREST

The authors declare no conflict of interest

## REFERENCES

[1] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," Journal of Field Robotics, vol. 37, no. 3, pp. 362-386, 2020.

[2] K. Muhammad, A. Ullah, J. Lloret, J. Del Ser, and V. H. C. de Albuquerque, "Deep learning for safe autonomous driving: Current challenges and future directions," IEEE Transactions on Intelligent Transportation Systems, vol. 22, no. 7, pp. 4316-4336, 2020.

[3] M. Capra, B. Bussolino, A. Marchisio, G. Masera, M. Martina, and M. Shafique, "Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead," IEEE Access, vol. 8, pp. 225134-225180, 2020.

[4] S. Paniego, N. Paliwal, and J. Cañas, "Model optimization in deep learning based robot control for autonomous driving," IEEE Robotics and Automation Letters, vol. 9, no. 1, pp. 715-722, 2023.

[5] B. Narottama and S. Y. Shin, "Layerwise Quantum Deep Reinforcement Learning for Joint Optimization of UAV Trajectory and Resource Allocation," IEEE Internet of Things Journal, vol. 11, no. 1, pp. 430-443, 2023.

[6] G. Wang, Z. Qin, S. Wang, H. Sun, Z. Dong, and D. Zhang, "Towards accessible shared autonomous electric mobility with dynamic deadlines," IEEE Transactions on Mobile Computing, vol. 23, no. 1, pp. 925-940, 2022.

[7] P. M. Kebria, A. Khosravi, S. M. Salaken, and S. Nahavandi, "Deep imitation learning for autonomous vehicles based on convolutional neural networks," IEEE/CAA Journal of Automatica Sinica, vol. 7, no. 1, pp. 82-95, 2019.

[8] K. Muhammad, A. Ullah, J. Lloret, J. Del Ser, and V. H. C. de Albuquerque, "Deep learning for safe autonomous driving: Current challenges and future directions," IEEE Transactions on Intelligent Transportation Systems, vol. 22, no. 7, pp. 4316-4336, 2020.

[9] Z. Huang, C. Lv, Y. Xing, and J. Wu, "Multi-modal sensor fusion-based deep neural network for end-to-end autonomous driving with scene understanding," IEEE Sensors Journal, vol. 21, no. 10, pp. 11781-11790, 2020.

[10] J. Fayyad, et al., "Deep learning sensor fusion for autonomous vehicle perception and localization: A review," *Sensors*, vol. 20, no. 15, p. 4220, 2020.

[11] N. Quang Hieu, D. T. Hoang, D. Niyato, P. Wang, D. In Kim, and C. Yuen, "Transferable Deep Reinforcement Learning Framework for Autonomous Vehicles with Joint Radar-Data Communications," e-prints arXiv:2105.13670 p. 17, 2021.

[12] K. Yu, L. Lin, M. Alazab, L. Tan, and B. Gu, "Deep learning-based traffic safety solution for a mixture of autonomous and manual vehicles in a 5G-enabled intelligent transportation system," IEEE Transactions on Intelligent Transportation Systems, vol. 22, no. 7, pp. 4337-4347, 2020.

[13] J. Yang, J. Zhang, M. Xi, Y. Lei, and Y. Sun, "A deep reinforcement learning algorithm suitable for autonomous vehicles: Double bootstrapped soft-actor–critic-discrete," IEEE Transactions on Cognitive and Developmental Systems, vol. 15, no. 4, pp. 2041-2052, 2021.

[14] M. I. Pavel, S. Y. Tan, and A. Abdullah, "Vision-based autonomous vehicle systems based on deep learning: A systematic literature review," Applied Sciences, vol. 12, no. 14, p. 6831, 2022.

[15] C. Lu, et al., "Learning configurations of operating environment of autonomous vehicles to maximize their collisions," IEEE Transactions on Software Engineering, vol. 49, no. 1, pp. 384-402, 2022.

[16] Z. Li, A. Samanta, Y. Li, A. Soltoggio, H. Kim, and C. Liu, "R$^3$: On-Device Real-Time Deep Reinforcement Learning for Autonomous Robotics," in *2023 IEEE Real-Time Systems Symposium (RTSS)*, Taipei, Taiwan, 2023, pp. 131-144.