LAWRENCE TECHNOLOGICAL UNIVERSITY

College of Arts & Sciences

Department of Math and Computer Science

EVOLUTIONARY OPTIMIZATION TO FIND LIGHTWEIGHT MODELS FOR AUTONOMOUS STEERING

A thesis presented in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

Awarded Spring 2025

BY

DEVSON BUTANI

Advisor: Dr. Chan-Jin "CJ" Chung

Date: May 13, 2025

This thesis has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Computer Science.

Department of Math and Computer Science, College of Arts and Sciences, Lawrence Technological University

Thesis Advisor:	*Dr. Chan-Jin "CJ" Chung*
	[illegible]
Committee Member:	*Dr. Eric Martinson*
	[illegible]
Committee Member:	*Dr. Corey Bohil*
	Corey Bohil
Department Chair:	*Dr. Eric Martinson*
	[illegible]

ABSTRACT

This research investigates the optimization of Convolutional and Dense Neural Networks (CNNs and DNNs) for autonomous steering using the (N+M) Evolution Strategy (ES) with the 1/5th success rule. The primary objective is to develop a lightweight CNN model architecture capable of real-time steering angle prediction, mimicking human driving behavior on predefined paths. The ES algorithm automates hyperparameter tuning, dynamically adjusting parameters such as filter sizes and layer configurations. Data collection encompasses driving scenarios recorded via the LTU ACTor autonomous driving platform, including variations in path direction and driving style. The very small dataset consists of timestamped images labeled with steering angles and pre-processed to focus on relevant visual information. Initial experiments involve training a baseline CNN model, which is then refined using ES to significantly reduce the size of the model while maintaining competitive predictive accuracy. The results highlight the viability of Evolutionary Hyperparameter Optimization (HPO) to find lightweight neural network architectures for real-time autonomous systems, striking a balance between computational efficiency and performance. This study not only advances research initiatives on the use of evolutionary algorithms for hyperparameter optimization but also lays the foundation for the deployment of cost-effective and scalable solutions in self-driving technology.

ACKNOWLEDGEMENTS

I am profoundly thankful to my advisor, Dr. Chan Jin Chung, for his unwavering support, mentorship, and encouragement. His dedication and leadership have not only guided this research but also inspired me to strive for excellence.

I would like to express my deepest gratitude to Dr. Eric Martinson, a member of my thesis committee, for his valuable insights, guidance, and constructive feedback throughout the development of this research. His expertise has been instrumental in refining the scope and direction of this work.

Their contributions have been vital to the completion of this thesis, and I am sincerely grateful for their time and commitment.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Autonomous driving systems require robust models capable of making real-time decisions under varying conditions. A critical component of these systems is the prediction of steering angles based on camera input, which involves processing visual data effectively and efficiently on compute-limited onboard processors [1]. This research focuses on training and optimizing CNNs using Evolutionary Strategy (ES) to automate the search for the best performing and best size-reduced model configurations.

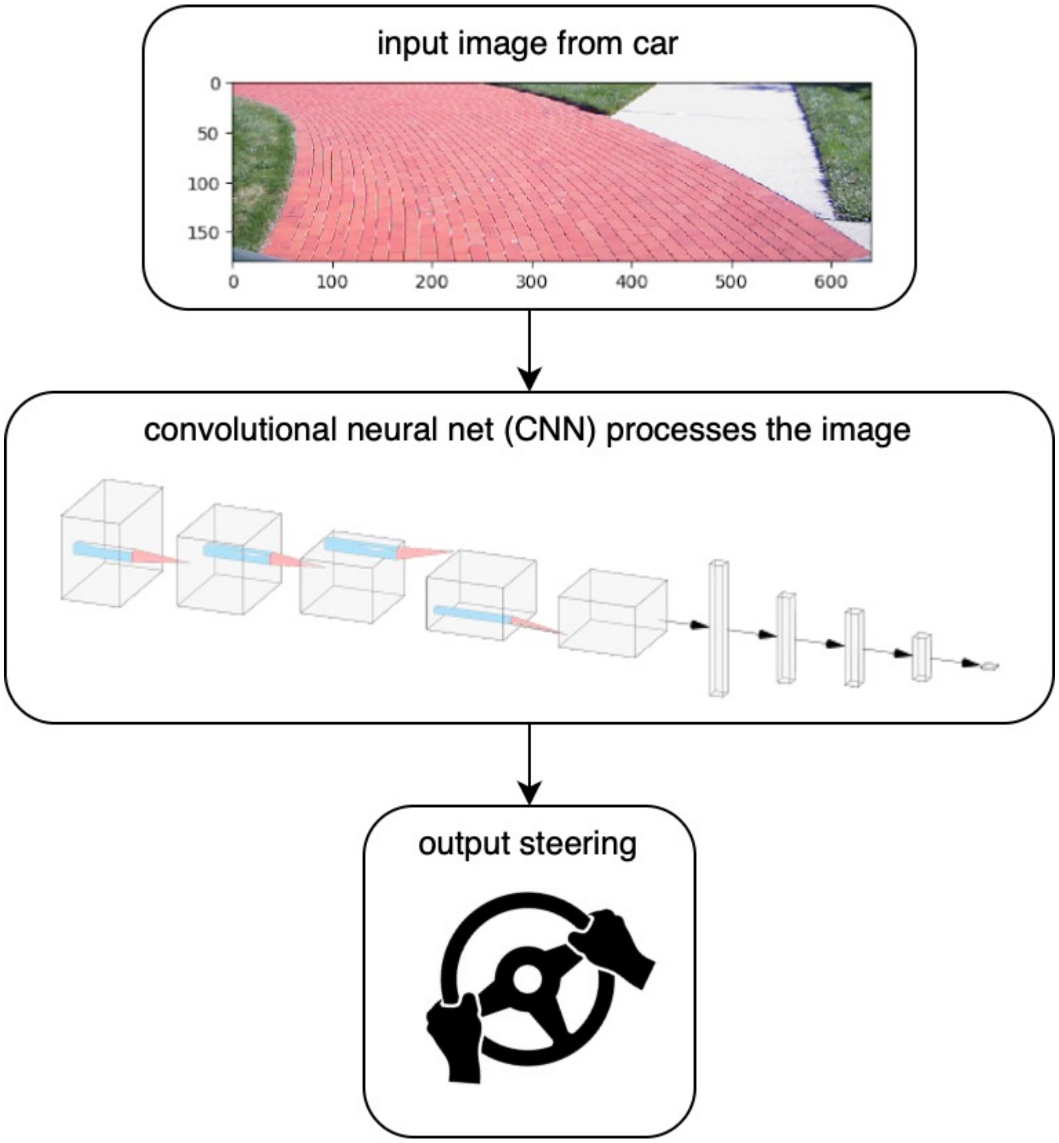


Figure 1. Problem space - end-to-end image-based steering

1.1 LITERATURE REVIEW

This research directly addresses the limitations that come with larger models and limited data. Larger models, while often achieving higher accuracy, come with increased computational cost and memory requirements, which can lead to slower inference times and higher energy consumption [2]. This poses a significant challenge for real-time applications like autonomous driving, where split-second decisions and energy efficiency are crucial. The size of a model directly impacts its feasibility for deployment, particularly in resource-constrained onboard processors [3]. Minimizing the size of the model and also training with a very small dataset is the core design challenge in autonomous driving systems.

Furthermore, the performance of deep learning models is highly dependent on the availability of large, diverse, and labeled datasets. In the context of autonomous driving, collecting and annotating such datasets for every possible scenario is impractical and cost prohibitive [4]. Although small datasets can lead to over-fitting, where the model performs well on the training data but fails to generalize to unseen scenarios or conditions, this is often not a problem in real-world applications where the operational design domain (i.e., the area of interest) can be regulated. For example, deploying an autonomous vehicle specifically for a small city or a specific set of routes. With this method, we can leverage the addition of small data sets every time the domain expands and utilize techniques like transfer learning and few shot learning to improve model performance over time [5]. Although optimization techniques have been extensively explored in various domains, their application to creating lightweight, well-generalized, and adaptable models for autonomous driving, specifically for real-time steering angle prediction, remains an active area of research.

Evolution Strategy (ES), one of the Evolutionary Algorithms (EAs) [6], offers a compelling advantage in this context. ES is a population-based optimization algorithm that leverages the collective intelligence of a population of candidate solutions to find the best-performing solution within a given search space [6]. Unlike gradient-based optimization methods, which can struggle in complex, non-differentiable, or noisy search spaces, ES algorithms are well-suited for exploring such landscapes [7].

In contrast to knowledge distillation techniques [8] that train smaller models to mimic larger pre-trained ones, our evolutionary approach directly finds the architectural hyperparameters to discover inherently efficient and small models within the search space. Compared to distillation, ES eliminates the need for large datasets and to design the two working model architectures to transfer knowledge between. This is particularly relevant to hyperparameter optimization of CNNs for autonomous driving, where the relationship between operational domain, model architecture, hyperparameters, and performance can be highly complex and non-linear.

The development of autonomous vehicles (AVs) holds immense potential for enhancing road safety, improving traffic flow, and increasing accessibility for individuals with mobility limitations [9]. However, the computational demands of traditional deep learning models pose a significant challenge. While optimization techniques have been extensively explored in various domains [10], their application to creating lightweight and adaptable models for autonomous driving, specifically for real-time steering angle prediction, remains an active area of research.

Addressing the vulnerability of autonomous driving models to adversarial attacks, Ren et al. (2024) presented two novel poisoning attacks on Federated Learning for regression tasks in autonomous driving: FLStealth and Off-Track Attack (OTA) [11]. Their work points out the critical requirement of massive datasets that need to be protected as well as bandwidth and storage costs when training large scale autonomous driving models. Even though our research is on optimizing CNN architectures using evolution strategies, it could potentially contribute to developing more resilient models by leveraging smaller datasets and smaller model sizes; using distributed cloud storage across servers or regions our models may iteratively learn over time.

Evolution Strategies (ES) offer a compelling advantage in this context. Unlike gradient-based optimization methods, which can struggle in complex, non-differentiable, or noisy search spaces, ES algorithms are well-suited for exploring such landscapes [7]. This is particularly relevant to hyperparameter optimization of CNNs for autonomous driving, where the relationship between model architecture, hyperparameters, and performance can be highly complex and non-linear. The concept of evolutionary strategies (ES) for optimizing neural networks has gained traction in recent years. Sadovsky et al. (2024) explored evolutionary approaches in the context of chloroplasts and mitochondria, highlighting the potential of adaptive algorithms in complex biological system [12]. This research suggests that ES could be effectively applied to the optimization of CNN architectures for autonomous steering, potentially leading to more efficient and adaptable models. The ability of ES to efficiently search for optimal architectures, even with limited data, makes it a powerful tool for developing lightweight and adaptable model [13]. Moreover, ES's population-based approach provides a natural way to explore diverse solutions and avoid getting trapped in local optima, increasing the likelihood of discovering architectures that generalize well to unseen scenarios [14].

Jiang et al. (2024) proposed a novel approach to steering angle prediction using behavioral cloning strategies for autonomous driving. Their work involved adapting NVIDIA's architecture and implementing the Swish activation function to train CNNs. By utilizing human driving data from both simulated and real-world environments, they achieved significant improvements in replicating human driving behavior [15]. While their approach demonstrates the potential of CNNs in steering angle prediction, Jiang et al. have not addressed the optimization of CNN architectures using evolutionary strategies.

In the realm of continual learning for autonomous driving, Zhang et al. (2024) introduced an Analytic Exemplar-Free Online Continual Learning algorithm (AEF-OCL) [16]. Their method leverages analytic continual learning principles and employs ridge regression as a classifier for features extracted by a large backbone network. Although this study focuses on classification rather than regression, Zhang et al. have not focused on addressing data imbalance and catastrophic forgetting in autonomous driving applications, which are challenges our research aims to tackle through the use of evolutionary strategies.

Addressing the challenges of model parameter identification in autonomous racing systems, Kim et al. (2023) proposed a hyperparameter optimization scheme (MI-HPO) for efficient explore-exploit strategies. Their method demonstrated more than 13 times faster convergence than traditional parameter identification methods and showed good generalization ability in unseen dynamic scenarios [17]. While this study highlights the importance of efficient hyperparameter optimization in autonomous driving applications, it does not specifically address the optimization of CNN architectures using evolutionary strategies.

1.2 Research Goals

Recent advances in image-to-steering angle prediction have demonstrated notable improvements in model quality and performance [1], [18], [19], [20] Building on this progress, the proposed Evolution Strategy (ES)-based optimization aims to significantly enhance deployable performance and enable real-time inference on embedded compute resources. This research is intended as a proof-of-concept for ES-driven hyperparameter optimization, rather than an attempt to outperform state-of-the-art architectures for autonomous steering.

It is important to note that evolutionary algorithms, including ES, are guided random search methods. Unlike purely random searches, they leverage selection and adaptation mechanisms to efficiently explore the search space, and are therefore expected to yield better results than random search alone.

The primary objectives of this research are:

1. **Framework Development:** Design and implement a framework that applies the (N+M) Evolution Strategy (ES) with the 1/5th success rule for automated hyperparameter tuning.

2. **Model Efficiency Improvement:** Reduce the size of the baseline CNN model while preserving robust steering angle prediction accuracy. Compare with the baseline model as well as random search optimized model.

3. **Real-World Validation:** Demonstrate the real-world applicability of ES-optimized models within autonomous vehicle systems.

1.3 Core Concepts

This thesis centers around several key concepts and explains how they are relevant to optimizing CNNs for steering angle prediction:

Evolutionary Algorithms (EAs): EAs are a class of optimization algorithms inspired by natural evolution, using mechanisms like selection, mutation, and recombination [17].

Equation 1. General idea of EAs

$$population^{t+1} = \mathbf{select}(\mathbf{vary}(population^{t}))$$

They are particularly effective in navigating complex, high-dimensional search spaces [21], making them suitable for optimizing neural network architectures and hyperparameters. In the context of this research, EAs provide a robust method for exploring diverse CNN architectures and finding optimal hyperparameter configurations that lead to improved steering angle prediction accuracy and model efficiency.

(N+M) Evolutionary Strategy (ES): A specific type of EA, the (N+M)-ES, maintains a population of N parent solutions and generates M offspring in each iteration [6]. The best N individuals from the combined parent and offspring pool are selected to form the next generation. This strategy balances exploration (through offspring generation) and exploitation (through selection of the fittest individuals), which is crucial for discovering CNN architectures that are both accurate and computationally lightweight for autonomous steering.

1/5th Success Rule: This rule, proposed by Rechenberg, is used to adaptively adjust the mutation strength (step size) in ES [22]. The simply states that the step-size should increase if “too many” steps are successful, indicating that the search is too local, and should decrease if “too few” steps are successful, indicating that the step-size used for sampling solutions is “too large”. It helps to dynamically balance exploration and exploitation during the optimization process. For our model, this ensures that the search for optimal CNN hyperparameters is both efficient and effective, leading to faster convergence and better model performance.

Hyperparameter Optimization (HPO): HPO is the process of finding the optimal configuration of hyperparameters for a machine-learning model, such as learning rate and batch size [25]. In this research, HPO is used to fine-tune the CNN architecture and training process, leading to improved accuracy and efficiency in steering angle prediction.

Random Search: A simple optimization method that randomly samples values over a uniform distribution of the search space and then evaluates them for performance. This process is repeated for a set number of trials or until a satisfactory result is found. It does not follow any pattern or use previous results to guide the search-each trial is independent and selected at random. Random search is often used to find good hyperparameter settings for models when the search space is large or not well understood.

2. METHODOLOGY

This research employs the (N+M)-ES with the 1/5th success rule to automatically optimize CNN architectures for the specific task of steering angle prediction in autonomous driving. This is a regression problem because it involves predicting a continuous value (steering angle). By encoding network hyperparameters (e.g., number of filters, layer sizes) as genes, the ES algorithm iteratively searches for optimal model configurations that minimize prediction error while maintaining a compact model size [26]. This is then compared random search where the same hyperparameters and search space is used to optimize the model.

Autonomous steering serves as a concrete, real-world application to demonstrate the effectiveness of the proposed optimization framework. By demonstrating our model on an autonomous driving platform, we can validate its performance in a realistic setting and showcase the tangible impact of this research. The choice of autonomous steering as an application domain is motivated by its direct relevance to road safety and its potential to significantly improve the capabilities of autonomous vehicles [27].

2.1 Data Acquisition and Pre-processing

2.1.1 Equipment

The LTU ACTor autonomous driving research platform served as the primary data acquisition system for this study. This platform integrates the Robot Operating System (ROS) Noetic framework with Ubuntu 20.04 as its underlying operating system, providing a robust and standardized environment for autonomous vehicle experimentation. Data collection was facilitated through the platform's comprehensive sensor suite, with particular emphasis on the forward-facing camera imagery and corresponding steering angle measurements.

All sensor data was systematically recorded using the ROS-native rosbag format. Rosbags function as chronologically-ordered repositories that capture the complete messaging ecosystem within the ROS network, preserving the precise temporal relationships between sensor inputs, control commands, and actuator responses. This data structure maintains critical temporal synchronization between visual perception data and control inputs, establishing the foundation for developing accurate end-to-end driving models.

The vehicle architecture incorporates a sophisticated drive-by-wire system that enables programmatic control of the vehicle's primary mechanical interfaces. This system permits direct electronic actuation of steering mechanisms and pedal positions while maintaining a seamless override capability for human operators. The drive-by-wire implementation creates a dual-control paradigm wherein manual operation remains fully accessible while simultaneously allowing software-defined control signals to be translated into physical vehicle movements. This architectural design facilitates a smooth transition between human and autonomous control modes, enabling direct deployment and validation of the optimized steering prediction models developed in this research.

The integration of these hardware and software components creates a closed-loop experimental environment where model outputs can be directly implemented as vehicle control inputs, allowing for comprehensive real-world evaluation of the steering prediction models.

2.1.2 Environment

Data collection was conducted along a carefully selected route: the circular red brick path encircling Ockham's Wedge at Lawrence Technological University. This sculptural centerpiece serves as a distinctive landmark within the university's Quadrangle. The location was strategically chosen to minimize the environmental complexities typically associated with asphalt roadways and traffic safety concerns.

The circular configuration of the path offers significant advantages for autonomous driving research. Situated within an open campus environment, the route provides excellent visibility in all directions, enabling safety drivers to maintain continuous visual awareness of their surroundings. Similarly, pedestrians can observe the research vehicle from considerable distances, substantially reducing potential safety risks associated with autonomous vehicle testing. This enhanced mutual visibility creates an ideal controlled environment for data collection while maintaining the highest safety standards.

Additionally, the consistent surface characteristics of the brick pathway provide uniform traction and visual features, eliminating many of the variables that would complicate data collection on public roadways. This controlled setting allows for more systematic evaluation of the steering prediction algorithms under development, as environmental factors remain relatively constant throughout testing sessions.

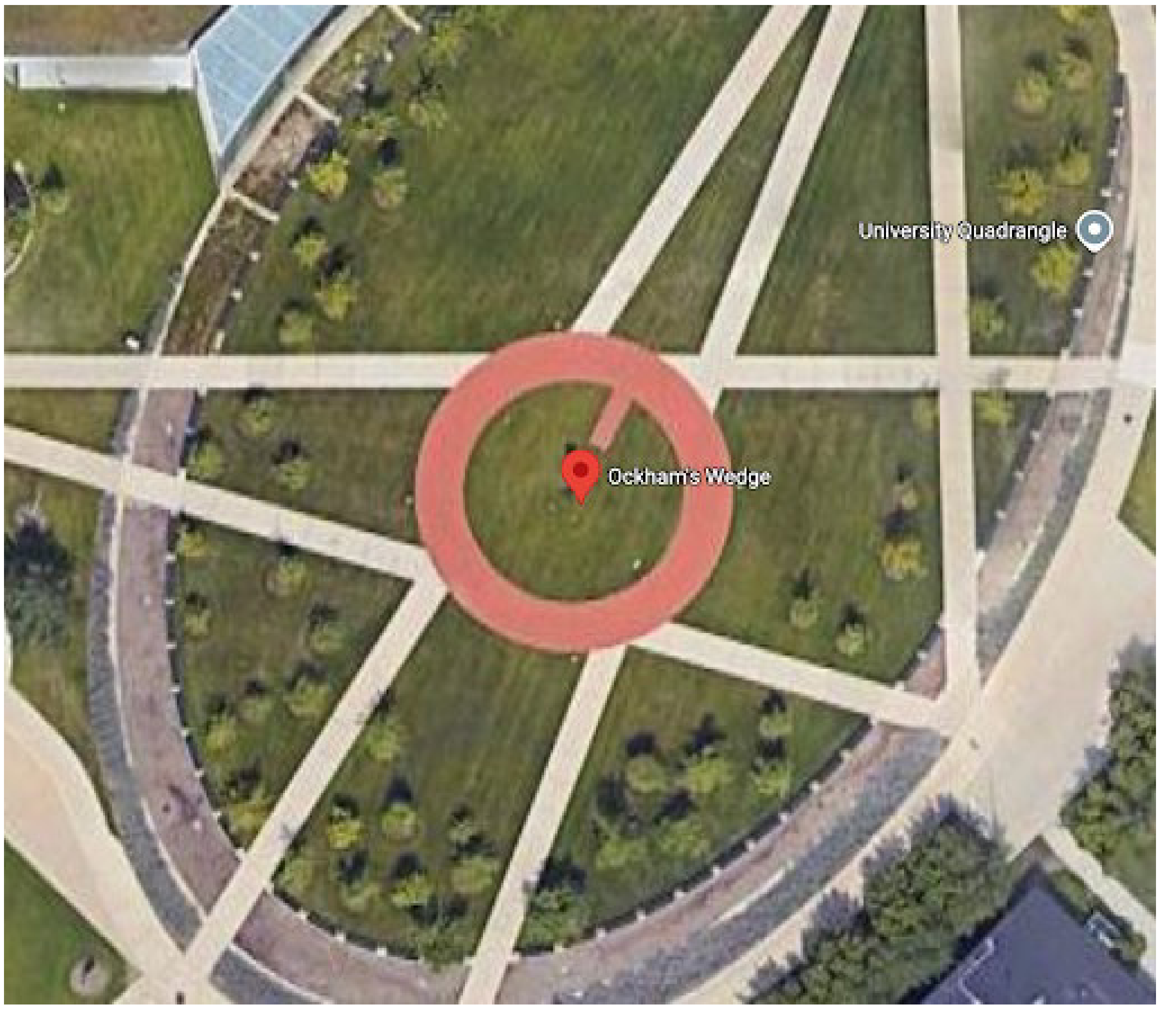


Figure 2. Ockham's Wedge at LTU

To introduce variability and enhance model robustness, driving sessions included clockwise and counterclockwise directions, smooth and zigzag maneuvers, and driving along inner and outer path edges to simulate diverse spatial alignments. Figure 2 shows an overhead view of the data collection site. Individual rosbags were collected for each of these variations to keep them separated for later use.

2.1.3 Extraction and Pre-processing

An extraction Python script was developed to process the ROS rosbag files, enabling systematic retrieval and organization of the experimental data. This utility extracted images and associated them with precise temporal identifiers and corresponding steering angles. The sampling frequency was adaptively configured based on driving dynamics: higher-frequency sampling (200ms intervals, corresponding to approximately 0.5m of vehicle travel) was employed for rosbags with zigzag maneuvers to capture rapid steering transitions, while lower-frequency sampling (up to 1000ms intervals, approximately 2m of vehicle travel) was utilized for rosbags with steady driving conditions.

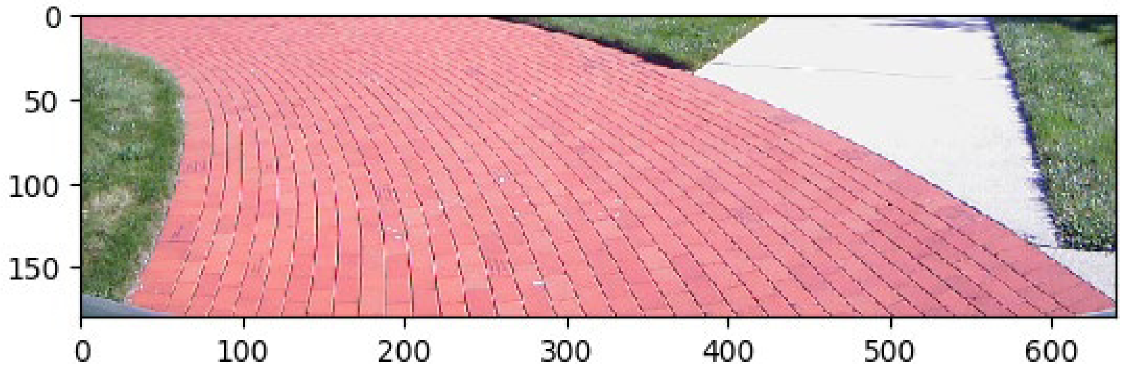
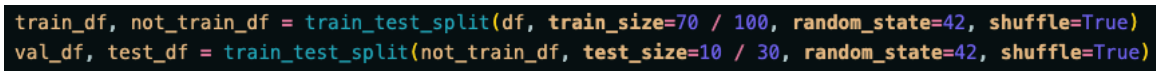


Figure 3. Example of extracted and preprocessed image

The preprocessing pipeline incorporated region-of-interest selection to eliminate non-informative visual elements. Specifically, the upper half of each image-containing predominantly sky and peripheral structures-was excluded from the input data. This dimensionality reduction focused the model's attention on the road surface and immediate surroundings, which contain the most relevant features for steering prediction. The resulting dataset comprised 2,958 images at 640×180 pixel resolution, partitioned using scikit-learn's train_test_split function with shuffled random sampling: 70% (2,069 images) allocated to training, 20% (592 images) to validation, and 10% (296 images) to testing.



```
train_df, not_train_df = train_test_split(df, train_size=70 / 100, random_state=42, shuffle=True)
val_df, test_df = train_test_split(not_train_df, test_size=10 / 30, random_state=42, shuffle=True)
```

Figure 4. Dataset random shuffle split

Despite the temporal sequencing of image acquisition, each frame was treated as an independent sample during model training. The shuffling of the dataset before splitting makes sure that each split has randomly distributed images collected from different rosbags. The variations of driving clockwise and counter clockwise around the circle path, as well as sunny and cloudy weather provide enough information about the path itself. It is important to emphasize that the primary research objective was not to develop a universally applicable autonomous driving system, but rather to demonstrate effective model size reduction and hyperparameter optimization while maintaining performance within a constrained operational domain. The deliberately limited dataset size facilitated execution of ES-based hyperparameter optimization algorithm on available computational resources, while simultaneously establishing a foundation for future research into few-shot learning techniques for domain adaptation.

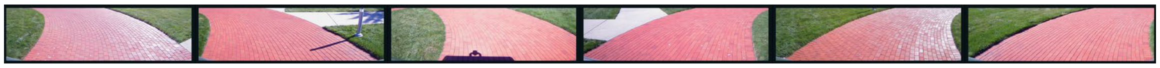


Figure 5. Example of random images picked from the train split

Data augmentation techniques were intentionally omitted to maintain a compact dataset size, simulating scenarios where computational efficiency is prioritized and incremental learning from new environments is required. This approach reflects real-world constraints where deployment on resource-constrained platforms necessitates efficient training methodologies. While the dataset's limited scope restricts generalization to diverse environmental conditions, it provides a controlled experimental framework for evaluating the efficacy of model size reduction techniques and optimization methodologies. The dataset's composition strikes a balance between representational adequacy for the specific operational domain and computational tractability for the evolutionary optimization process.

2.2 Model Training and Baseline Establishment

Models are trained using the Keras API with a PyTorch backend to leverage GPU compute capacity. The training process employs the Mean Squared Error (MSE) loss function, which quantifies the average squared difference between the predicted and actual steering angles. The Mean Absolute Error (MAE) serves as the key performance metric, indicating the average deviation in steering angle predictions in degrees. This provides a straightforward and interpretable measure of model accuracy for autonomous steering applications. Minimizing MAE is crucial to achieving precise real-time control of the vehicle. These metrics serve as benchmarks for evaluating the efficacy of ES with the 1/5 success rule in optimizing model architectures.

Initial experiments began with a single-layer CNN to establish a rudimentary baseline. However, the limited capacity of this architecture renders it unsuitable for real-world driving scenarios. To address this, the PilotNET architecture, a CNN and DNN combination developed by NVIDIA [28] as a foundation model was adopted as our baseline.

PilotNET, an early milestone in autonomous steering research, demonstrated the potential of GPU-accelerated deep learning for this domain. This simple architecture, shown in Figure 6, works as the baseline because this study is benchmarking performance improvements rather than absolute performance. Leveraging this proven architecture allowed for a more effective comparison of ES-optimized models against a recognized standard.

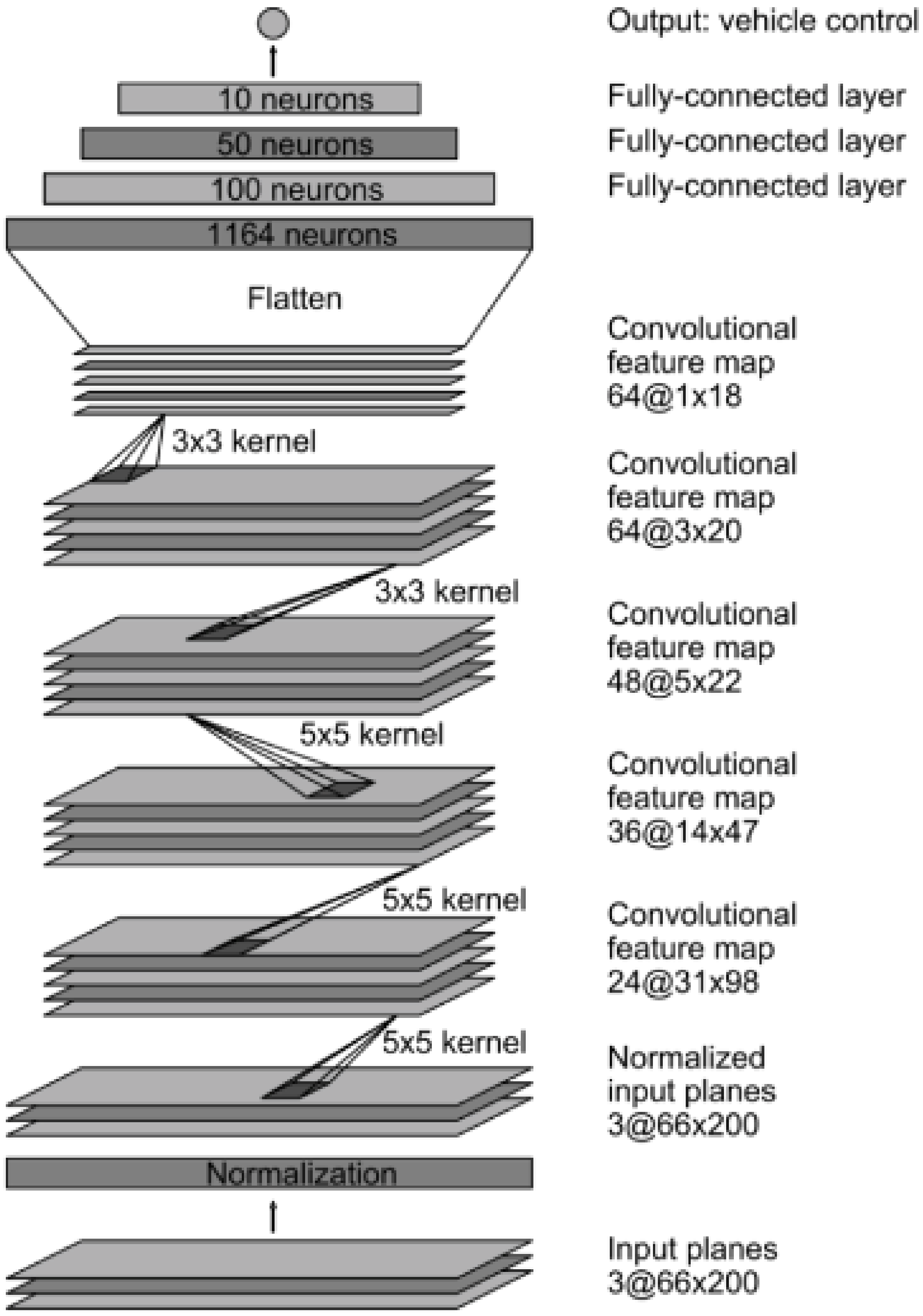
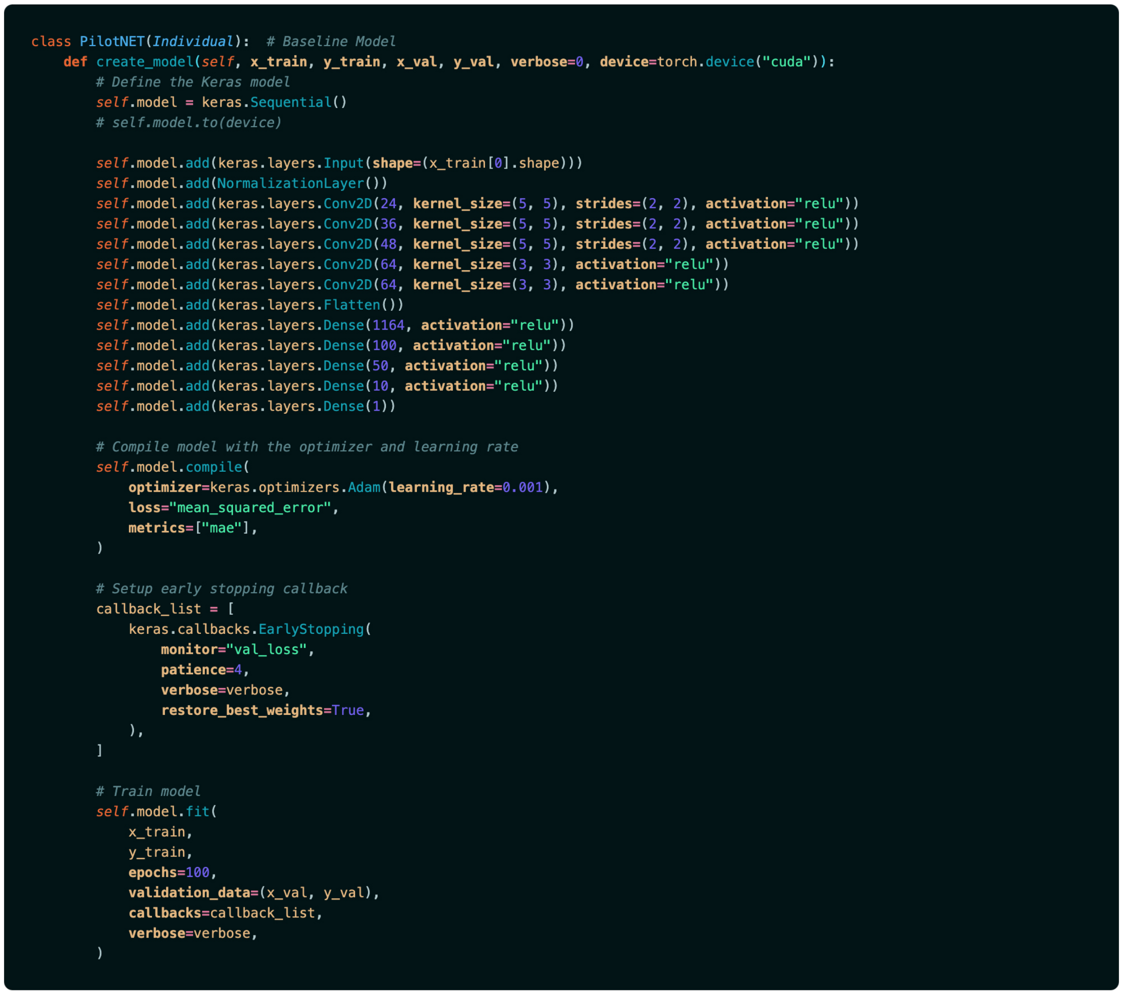


Figure 6. NVIDIA PilotNET architecture



```python
class PilotNET(Individual):  # Baseline Model
    def create_model(self, x_train, y_train, x_val, y_val, verbose=0, device=torch.device("cuda")):
        # Define the Keras model
        self.model = keras.Sequential()
        # self.model.to(device)

        self.model.add(keras.layers.Input(shape=(x_train[0].shape)))
        self.model.add(NormalizationLayer())
        self.model.add(keras.layers.Conv2D(24, kernel_size=(5, 5), strides=(2, 2), activation="relu"))
        self.model.add(keras.layers.Conv2D(36, kernel_size=(5, 5), strides=(2, 2), activation="relu"))
        self.model.add(keras.layers.Conv2D(48, kernel_size=(5, 5), strides=(2, 2), activation="relu"))
        self.model.add(keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu"))
        self.model.add(keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu"))
        self.model.add(keras.layers.Flatten())
        self.model.add(keras.layers.Dense(1164, activation="relu"))
        self.model.add(keras.layers.Dense(100, activation="relu"))
        self.model.add(keras.layers.Dense(50, activation="relu"))
        self.model.add(keras.layers.Dense(10, activation="relu"))
        self.model.add(keras.layers.Dense(1))

        # Compile model with the optimizer and learning rate
        self.model.compile(
            optimizer=keras.optimizers.Adam(learning_rate=0.001),
            loss="mean_squared_error",
            metrics=["mae"],
        )

        # Setup early stopping callback
        callback_list = [
            keras.callbacks.EarlyStopping(
                monitor="val_loss",
                patience=4,
                verbose=verbose,
                restore_best_weights=True,
            ),
        ]

        # Train model
        self.model.fit(
            x_train,
            y_train,
            epochs=100,
            validation_data=(x_val, y_val),
            callbacks=callback_list,
            verbose=verbose,
        )
```

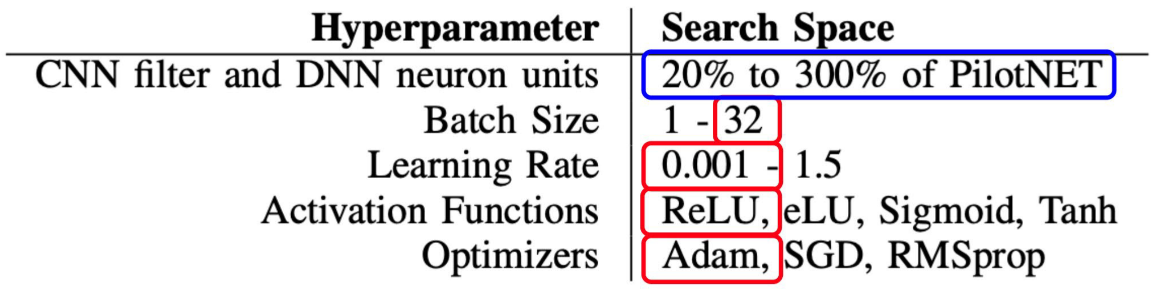
Figure 7. Keras based PilotNET baseline model

As shown in Figure 7, early stopping was implemented to improve training efficiency and prevent overfitting. Training instances are terminated if no improvement in the MSE metric is observed after four epochs. This strategy prevents the allocation of computational resources to hyperparameters that do not contribute to model performance, thereby accelerating the optimization process.

2.3 Hyperparameter Management

Initially, the hyperparameter search space included individual layer units (number of filters in convolutional layers and neurons in fully connected layers), batch sizes, learning rates, activation functions, and optimizer selection. To facilitate a comprehensive exploration of potential architectures, the layer units were allowed to vary from 20% to 300% of the PilotNET baseline.

Table 1. Hyperparameter search spaces



Hyperparameter	Search Space
CNN filter and DNN neuron units	20% to 300% of PilotNET
Batch Size	1 - 32
Learning Rate	0.001 - 1.5
Activation Functions	ReLU, eLU, Sigmoid, Tanh
Optimizers	Adam, SGD, RMSprop

However, preliminary experiments indicated that focusing optimization efforts solely on the layer units resulted in improved performance and reduced search time. Consequently, batch size (32), learning rate (0.001), activation function (ReLU), and optimizer (Adam) were fixed to match the baseline model configuration. This narrowed scope allowed ES to concentrate on the most impactful architectural parameters, leading to more efficient exploration of the design space. The ranges of the layer units explored are provided in Table 1.

2.4 Optimization Framework

CNN hyperparameters, specifically the number of filters in convolutional layers and the number of neurons in fully connected layers, are encoded as real-valued genes within the ES framework [30]. The (N+M)-ES approach iteratively optimizes hyperparameters by combining the N best-performing parent models with M offspring models generated through mutation. This approach allows for both exploitation of promising regions of the hyperparameter space (through selection of top parents) and exploration of new possibilities (through mutation of offspring).

2.5 ES Algorithm with 1/5 Success Rule

The Evolution Strategy (ES) algorithm employs the 1/5th success rule [22] to dynamically adjust the mutation step size (σ). This adaptive mechanism ensures efficient exploration of the hyperparameter space while avoiding premature convergence to local optima.

The rule is defined as follows:

Equation 2. 1/5th success rule

$$\sigma_{t+1} = \begin{cases} \alpha * \sigma_t & \textbf{success rate} > \frac{1}{5} \\ \beta * \sigma_t & \textbf{success rate} \leq \frac{1}{5} \end{cases}$$

where σ is the updated step size for the next generation. σ_t is the current step size. $\alpha > 1$ is a scaling factor to increase the step size (e.g., 1.22). $\beta < 1$ is a scaling factor to decrease the step size (e.g., 0.82). The "success rate" is defined as the fraction of offspring that outperform their parents in terms of validation performance.

If the optimization success rate is greater than 1/5th within a generational window, the step size is increased, encouraging greater exploration of the hyperparameter space. Conversely, if the success rate is less than or equal to 1/5, the step size is decreased, promoting more focused refinement of promising solutions. This adaptive mechanism ensures efficient exploration while avoiding premature convergence to local optima.

```
Algorithm 1 (N+M) Evolution Strategy with 1/5 Success Rule
Input: Hyperparameter ranges $[min(h), max(h)]$, number of parents $N$, offspring $M$, generations $G$, step size factors $\alpha > 1$, $\beta < 1$, window size $W$
Output: Best model with optimized hyperparameters
  Initialize, train and evaluate $N$ parents random parent models
  Set step size $\sigma \leftarrow$ initial value, generation $t \leftarrow 0$, positive counter $P \leftarrow 0$
  while $t < G$ and termination condition not met do
      Generate $M$ offspring:
          $\hat{h} = h + \text{gauss}(0, \sigma \cdot (max(h) - min(h)))$
      Train and evaluate offspring
      Combine parents and offspring
      Select top $N$ models as next-generation parents
      if at least one offspring outperforms parents then
          Increment $P \leftarrow P + 1$
      end if
      if $t$ mod $W == 0$ then            ▷ Adjust $\sigma$ every $W$ gen.
          Calculate success rate:
              success rate $= P/M$
          if success rate $> 1/5$ then $\sigma = \alpha \cdot \sigma$
          else $\sigma = \beta \cdot \sigma$
          end if
          Reset $P \leftarrow 0$
      end if
      Increment $t$
  end while
  Return best model from final population
```

Figure 8. (N+M) Evolution Strategy with 1/5 Success Rule

The ES optimization process with 1/5 success rule [31], [32], [33] is as follows:

1) **Initialization**: Generate, train and evaluate N parent models with random hyperparameters sampled within predefined ranges. The initial hyperparameters are

drawn from a uniform distribution within the specified ranges. The number of parents, N, is a tunable parameter that controls the diversity of the population.

2) **Child Generation**: Create M offspring by mutating parent hyperparameters according to the following equation:

Equation 3. Bounded random mutation

$$\hat{h} = h + \textbf{gauss}(0, \sigma \cdot (max(h) - min(h)))$$

Where ˆh represents the mutated hyperparameter value. h is the original hyperparameter value from the parent model. gauss(μ, σ) is a random number drawn from a Gaussian distribution with mean μ and standard deviation σ. In this case, μ = 0. σ is the mutation step size, controlling the magnitude of the mutation. max(h) and min(h) define the upper and lower bounds of the hyperparameter search space, respectively.

The term (max(h) - min(h)) normalizes the mutation step size to the range of the hyperparameter, ensuring that the mutation is proportional to the scale of the parameter search space.

3) **Training and Evaluation**: Train all offspring models and evaluate their performance based on validation loss (MSE) and accuracy (MAE). These are calculated using the test split of the dataset.

4) **Selection and Step size Adjustment**: Retain the top N models (from parents and children) based on validation performance to form the next generation's parent population. A window size of 5 generations was used to balance exploitation and exploration for this particular experiment as seen in Figure 8. This selection process ensures that the most promising models are carried forward, driving the evolution towards improved performance while also allowing exploration to nearby search space. The mutation step size σ is dynamically adjusted using the 1/5 success rule [22].

As described in Figure 6, this process repeats until the max generation count is reached or the termination condition, a solution much better than the baseline, is met; for this research max generations were limited to 100 and the termination condition was a MAE of 0.1 degrees.

2.5.1 Computational Resources

Training is performed on Lawrence Technological University's NVIDIA A100 GPU server, enabling efficient parallel training of N parent and M offspring model pools. The large 80GB VRAM (Video Random Access Memory) capacity of the A100 GPU allows for the use of larger N and M values, which increases the diversity of the population and enhances the exploration of the hyperparameter space. The specific values of N and M, shown in Table 3 are based on the availability of shared computational resources during training and the complexity of the model. While other N and M values (1 - 100) were used, Table 3 values are from the optimization runs that showed the best performance. One of the benefits of reducing model size is that it allows ES optimization with higher N and M values or free up resources for other concurrent projects.

For real-time inference that is discussed in later sections, NVIDIA RTX A2000 (8GB) was used for simulation testing and NVIDIA RTX 3060Ti (6GB) was used for testing on a real vehicle. Both these mobile GPUs are sized much larger than the model size preventing any bottlenecks from VRAM availability while running real-time inference.

2.6 Establishing Baseline Random Search Optimization

To evaluate the efficacy of the (N+M)-ES approach, a comparative analysis was conducted using Random Search Optimization as a benchmark method. This traditional technique serves as a well-established baseline for hyperparameter optimization tasks. The implementation utilized the identical hyperparameter search space defined in Table 1, which specifies the bounds for CNN and DNN neuron units. These boundaries established the constraints for uniform random sampling of architectural parameters.

The Random Search procedure was executed for 1,000 iterations – representative of (N+M)*100 generations for roughly equivalent comparison – wherein each iteration involved stochastically selecting hyperparameters from the defined distribution, constructing the corresponding neural network architecture, and evaluating its performance on the test dataset. This approach embodies a non-gradient, zero-order optimization strategy that makes no assumptions about the underlying structure of the hyperparameter landscape.

Upon completion of the search process, the algorithm outputs both the optimal hyperparameter configuration and the corresponding model that achieved the lowest Mean Absolute Error (MAE) on the test dataset. This methodology provides a probabilistic exploration of the hyperparameter space with computational efficiency comparable to the (N+M)-ES approach, thereby enabling a fair comparison between the two optimization strategies. The random nature of this search technique allows it to potentially discover high-performing configurations that might be overlooked by more structured search methods, particularly in non-convex optimization landscapes with multiple local minima.

2.7 Model Selection

Four different model architectures are proposed to determine the effect of ES on model performance and size. The first model, named "Baseline", is the same architecture established by PilotNET [28]. To compare with the traditional random search optimization method, “Optimized-RS” model is also added. The third model, named "Optimized", contains a CNN and DNN structure that has been optimized with ES, within the search space proposed in Table 1. The fourth model, named "Half-Size", contains the PilotNET CNN architecture with half of the baseline number of CNN filters per layer, and half of the DNN search space proposed by Table 1. The fifth model, named "Quarter-Size", contains the PilotNET CNN architecture with a quarter of the baseline number of CNN filters per layer, and a quarter of the DNN search space proposed by Table 1. Note that the pretrained weights of PilotNET were not used in this study. These five models allow the effect of ES on model performance with smaller model sizes and traditional methods to be observed.

Table 2. Model Descriptions

Model Name	Description
Baseline	PilotNET architecture, baseline for comparison.
Optimized-RS	Random search optimized CNN & DNN layers
Optimized	ES-optimized CNN & DNN layers.
Half-Size	Half the size of baseline CNN layers and ES-optimized DNN layers.
Quarter-Size	Quarter the size of baseline CNN layers and ES-optimized DNN layers.

2.8 Model Testing

To validate the real-time applicability of the ES-optimized models, a series of tests were conducted within a 2D simulation environment, GazelleSim [34]. The simulation allows for controlled and repeatable testing of the steering control algorithms on the red brick circular path (operational design domain) used for training. For variance, both directions (clockwise and counterclockwise) were tested; however, the observed difference in performance was negligible.

The autonomous driving testing pipeline operates as follows:

1) **Image Acquisition**: Simulated camera images are captured from the 2D environment, representing the vehicle's forward perspective.
2) **Model Inference**: The captured image is fed into the trained model, which predicts a steering angle. The predicted steering angle represents the desired direction of the vehicle.
3) **Steering Control**: The moving average of the predicted steering angle is translated into a control command that adjusts the vehicle's trajectory within the simulation. Then steering angle is converted into a control signal that influences the vehicle's direction and speed. For real-world testing, this control signal can be sent to the real vehicle's steering and throttle actuators instead.
4) **Closed-Loop Feedback**: This pipeline loops back to image acquisition, model inference, and steering control, creating a closed-loop control system that is capable of driving the vehicle in real-time using onboard compute.

2.9 Experimental Results

Table 3 presents the units per layer of each of the four trained and optimized models as well as the model optimized by random search. Layers that are dynamically optimized using ES with the 1/5 success rule are labeled in a bolded font (same for random search), while non-bolded layers are static baseline PilotNET architecture layers modified according to Table 2.

Table 3. Model architectures after ES-Optimization

Model	Architecture	N	M
Baseline	CNN: 24 → 36 → 48 → 64 → 64 → DNN: 1164 → 100 → 50 → 10 → 1	n/a	n/a
Optimized-RS	**CNN: 29 → 36 → 56 → 99 → 117 →** **DNN: 1613 → 149 → 10 → 16 →** 1	n/a	n/a
Optimized	**CNN: 6 → 9 → 43 → 28 → 61 →** **DNN: 2328 → 146 → 100 → 20 →** 1	2	4
Half-Size	CNN: 12 → 18 → 24 → 32 → 32 → **DNN: 149 → 14 → 20 →** 1	4	6
Quarter-Size	CNN: 6 → 9 → 12 → 16 → 16 → **DNN: 8 → 3 →** 1	3	6

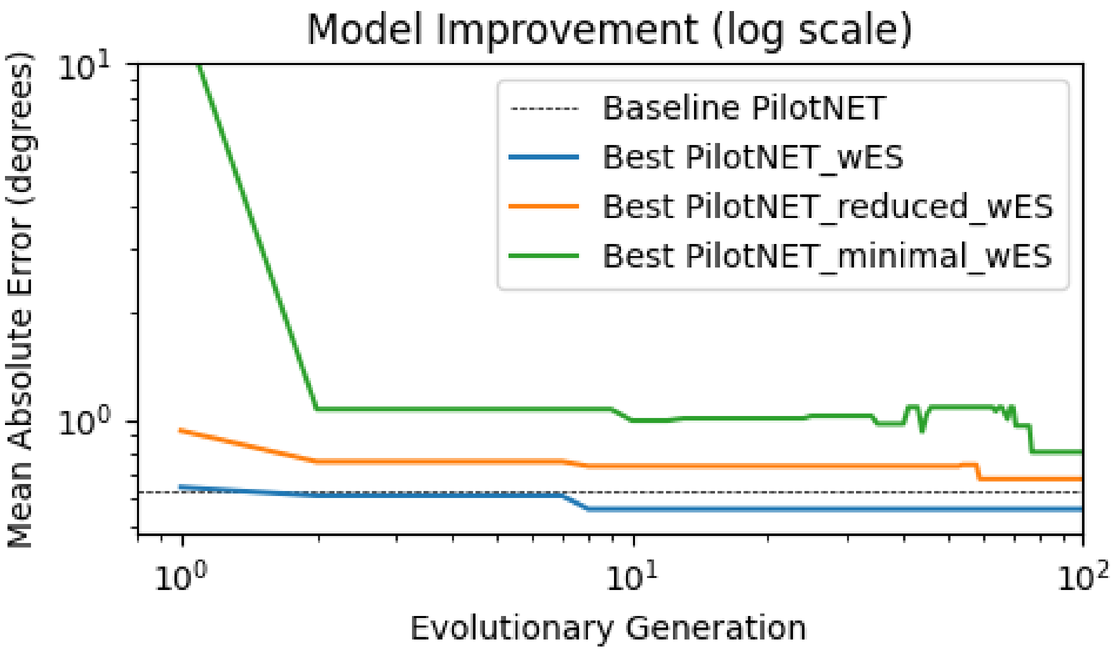


Figure 9. Model improvement over evolutionary generations (wES=Optimized, reduced_wES=Half-Size, minimal_wES=Quarter-Size)

2.9.1 Testing with GazelleSim

These optimized models were tested in the GazelleSim environment to evaluate their real-time applicability. This test was conducted on a NVIDIA RTX A2000 Laptop GPU with 8GB VRAM and a lower power limit (35W) than the A100 GPU used in training. Even though our models do not saturate the VRAM, the model architecture, power limit and memory bandwidth may bottleneck real-time inference latency, simulating a low power onboard compute environment.

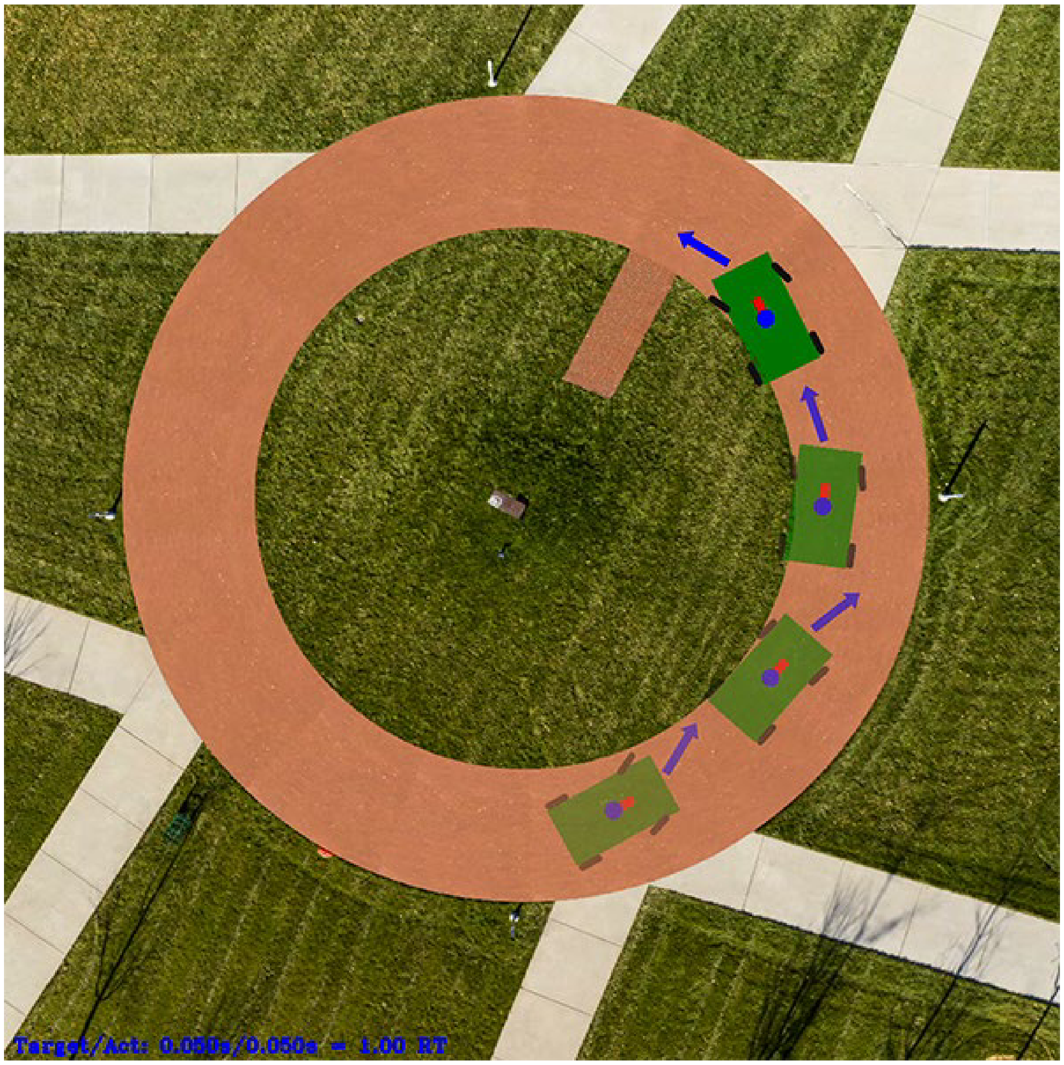


Figure 10. GazelleSim driving diagram

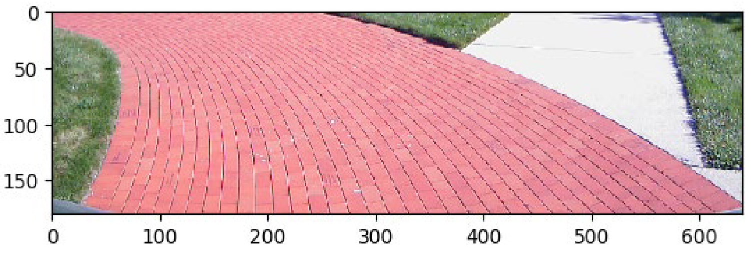


Figure 11. Example prediction results from the baseline model (Expected Angle: 12.901, Predicted Angle: 13.265, Abs Error: 0.364)

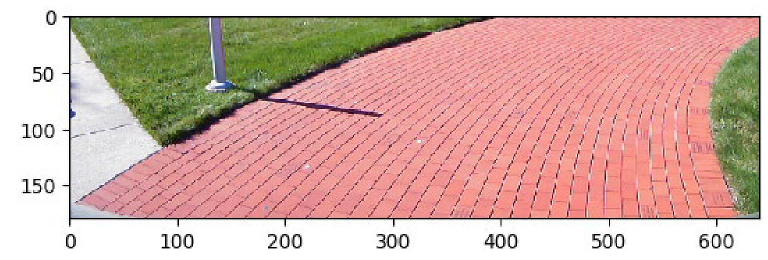


Figure 12. Example prediction results from the ES-optimized model (Expected Angle: -10.099, Prediction Angle: -9.822, Abs Error: 0.276)

The results are presented in Table 4. The speeds are not to-scale with real-world speeds due to simulator calibration; rather, the results can represent model performance as vehicle speeds increase while inference capacity is held constant.

Table 4. Test results

Test Duration & Speed	Model	Distance (m)
10 min @ 2 m/s	Baseline	1200.026
	Optimized-RS	1200.094
	Optimized	1200.072
	Half-Sized	1200.206
	Quarter-Sized	1200.169
60 min @ 2 m/s	Baseline	7200.026
	Optimized-RS	7200.047
	Optimized	7200.177
	Half-Sized	7200.053
	Quarter-Sized	7200.056
60 min @ 4 m/s	Baseline	14400.408
	Optimized-RS	14400.412
	Optimized	Did not make 1 lap
	Half-Sized	Did not make 1 lap
	Quarter-Sized	14400.400

Table 5 details the performance and size metrics of each of the four models. The number of parameters and MSE and MAE (evaluated on the test split of the dataset) were direct output from the Keras API. The VRAM usage of the inference process was measured using the *nvidia-smi* command during the trials. The inference times were averaged over 1000 predictions.

Table 5. Compute utilization

Model	Params	Memory (MB)	MSE (deg)	MAE (deg)	Inference Time (ms)
Baseline	245M	936	1.01	0.63	4.8
Optimized-RS	621M	3274	0.81	0.58	4.5
Optimized	250M	1050	0.49	0.41	4.5
Half-Size	61.4M	420	1.33	0.78	4.2
Quarter-Size	4.97M	146	1.88	0.88	4.1

2.9.2 Testing on LTU ACTor

In the context of the LTU ACTor autonomous driving platform outlined in Section 2.1.1, the drive-by-wire system facilitates the transmission of target steering angle commands to the vehicle. To deploy these models on the ACTor, a ROS package was created to handle receiving the camera stream in real-time, performing pre-processing on it (resize to 640x130 by cropping the top 50%) and then send the image to the CNN model pre-loaded in the GPU. The model then outputs the steering angle that is sent as a target angle to the vehicles drive by wire system (also running on ROS). However, the model's output exhibits noise characteristics that, despite maintaining values within acceptable margins of the expected steering angle, cause the vehicle's drive-by-wire system to continuously attempt to achieve the target angle within brief temporal intervals. From a hardware perspective, such persistent directional adjustments in the steering actuator generate significant thermal load, potentially leading to premature component failure.

To mitigate this issue, an Exponential Moving Average (EMA) filter is implemented. The EMA filter is a discrete, low-pass, infinite-impulse response filter that assigns greater weight to recent data points while exponentially discounting older measurements. This characteristic makes it particularly suitable for real-time control applications in autonomous vehicles, where responsiveness to current conditions must be balanced with signal stability.

The implementation utilizes a modest window of just 10 values, which proves sufficient to attenuate the high-frequency noise components while preserving the essential steering command information. This filtering approach effectively smooths the control signal without introducing excessive lag that might compromise vehicle maneuverability.

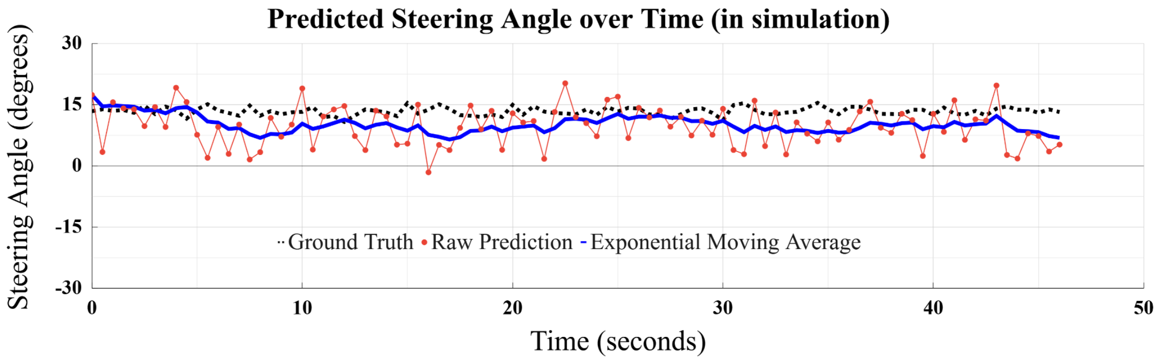


Figure 13. Real-time steering angle prediction over time using the quarter-size model

By applying this filtering technique, the system maintains steering angles closely aligned with expected values while significantly reducing the frequency and magnitude of actuator adjustments. This not only extends the operational lifespan of the steering hardware but also enhances the overall stability and predictability of the vehicle's trajectory, which is critical for reliable autonomous operation in diverse driving scenarios.

After deploying the optimized CNN models on the LTU ACTor autonomous driving platform in real-world scenarios [30], our testing shows simulation equivalent real-world performance. With all four models, the car was able to steer around the circular path without leaving the path. Results from Table 4 were close to real-world observations; models Baseline and Quarter-sized performed well while Optimized and Half-sized failed much sooner as the speed increases.

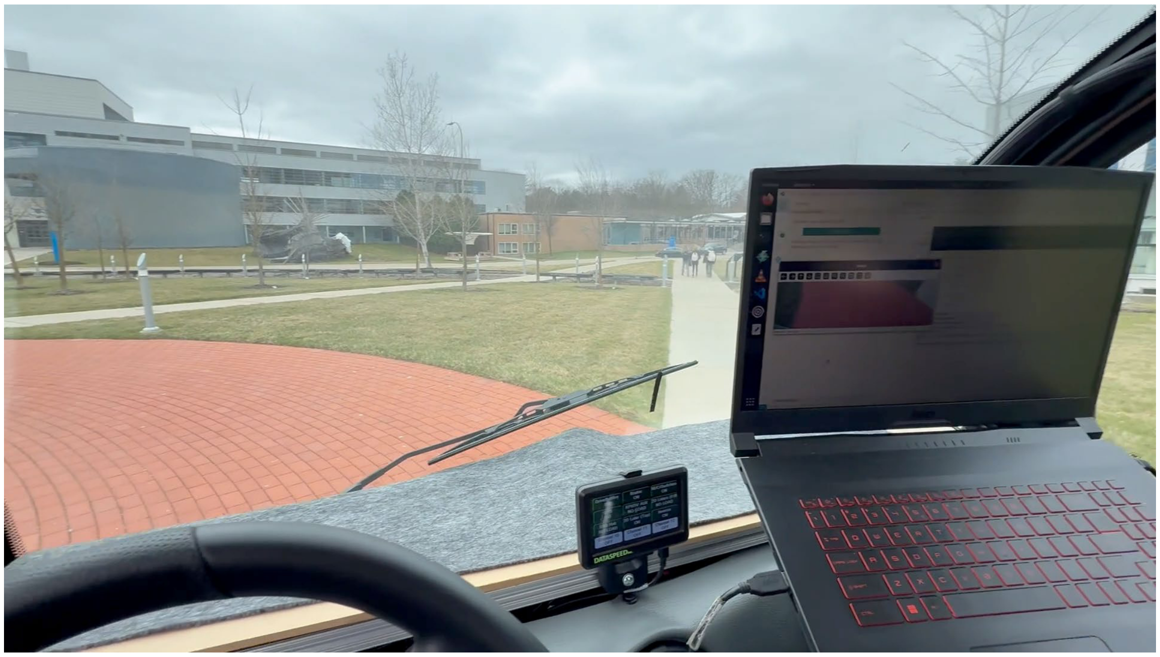


Figure 14. Real-world validation using the LTU ACTor platform

2.9.3 Discussion

Across 5-10 ES runs for each model size we observed the 1/5th rule reliably sort through hyperparameters to output a positively optimized end model for all runs. No instability or failed optimization runs occurred therefore, this proof-of-concept experiment establishes a simple statistical significance for using ES with 1/5 success rule to optimize convolutional and fully connected dense neural network layers for steering angle prediction in autonomous driving systems and hence meets our goals.

The PilotNET baseline achieved reasonable accuracy but required significant computational resources due to its large model size (245M params, 936 MB). The baseline model was able to predict steering angles with an MSE of 1.01 degrees, and a MAE of 0.63 degrees both within reasonable real-world driving accuracy. This model had no problems with real-time inference in simulations at 2 m/s and 4 m/s proving its suitability as a baseline for our study. This opens the way for architecture optimization to improve efficiency without sacrificing performance.

The Random Search optimized model, designated "Optimized-RS," demonstrated comparable predictive accuracy to the evolutionary strategy approach. However, this came at a substantial 3.5x computational cost, with the model containing over 3.2x (621 million) parameters and requiring 3,274 MB of memory-approximately 2.5 times larger than the evolutionary strategy optimized alternative. Despite its considerable resource requirements, the model achieved impressive performance metrics on the test dataset, with a Mean Squared Error of 0.81 and a Mean Absolute Error of 0.58 degrees, surpassing the baseline model's performance (MAE 0.63 degrees) benchmarks.

The Optimized-RS model maintained its effectiveness across both simulation environments and real-world driving tests at various vehicle speeds, confirming its practical applicability. This outcome validates that traditional random search optimization techniques can indeed produce high-performing models for autonomous steering prediction tasks. However, the stochastic nature of random search introduces significant variability in optimization outcomes, as the method cannot guarantee consistent convergence to optimal or near-optimal solutions across multiple optimization runs. This inherent non-determinism represents a notable limitation when compared to more directed search strategies, particularly in resource-constrained development environments where computational efficiency and reliability are paramount considerations.

The ES-optimized PilotNET model achieved the lowest error (MSE: 0.49, MAE: 0.41 degrees) while the model size increased by 5M parameters. While this shows that ES-optimization works to improve model performance, it is important to note that the reduced models (half-size and quarter-size) showed trade-offs. They consumed significantly fewer resources and their prediction accuracy decreased, with the half-size model exhibiting the highest error (MSE: 1.33, MAE: 0.78 degrees). Even though a minor increase of error was observed in the half-sized model, the quarter-size model required just 15% of the baseline model's VRAM for inference while achieving baseline equivalent performance in real-time simulations. In real-world low speed self-driving applications, a small difference in error (<1.0 degree) with a significant size reduction is a valid trade-off.

The performance-to-resource ratio achieved by the random search approach further emphasizes the value of the evolutionary strategy optimization method, which produced comparable performance with substantially reduced computational requirements. This comparison highlights the importance of considering both model performance and resource utilization when developing autonomous driving systems intended for deployment on platforms with limited computational capacity.

The 4m/s trials revealed a hard inference rate threshold, with Optimized (250M params) and Half-Sized (61.4M params) failing despite superior theoretical compute capacity. Given the close inference times in Table 5, it is clear that the bottleneck is not the model size or memory availability. It may be the case that the end-to-end image to steering pipeline in the simulator is not fast enough to handle a simulated speed of 4m/s in real-time however, a larger model like the Optimized-RS ran well at 4m/s.

The Quarter-Sized model completed the 60-minute trial with only 0.008m deviation from Baseline (14400.400m vs. 14400.408m), demonstrating ES's ability to preserve functionality during radical downsizing. Minor performance variations at 2m/s (<0.18% across models) further confirm architectural equivalence under non-saturating conditions.

A notable finding from this research is the disproportionate relationship between model size reduction and inference latency improvement in the steering prediction pipeline. Despite achieving substantial reductions in model parameters, the corresponding decrease in processing latency was relatively modest. As shown in Table 5, the latency improved from 4.8ms to 4.1ms, representing only a 14.6% reduction however, this improvement does not corelate with the model size. This raises important questions regarding the efficacy of model compression techniques for enhancing real-time performance in autonomous driving applications.

To contextualize these latency values, at vehicle speeds of 5mph and 70mph, the temporal delay translates to spatial displacements of approximately 0.01m and 0.15m, respectively. While these values fall well within acceptable parameters for the current experimental platform operating within the constrained operational design domain of a red brick path, they prompt critical consideration of deployment requirements in more complex environments.

The modest improvement in latency despite significant model size reduction suggests the presence of other major bottlenecks in the inference pipeline. These may include memory bandwidth limitations, data transfer overhead, or computational constraints unrelated to model size. The actual computation done by the model may be faster with reduced size however the improvement may be too miniscule in comparison with other bottlenecks such as memory bandwidth or the Python script's time measurement speed. This observation aligns with common knowledge in the field indicating that model size optimization techniques primarily address parameter count and memory footprint rather than necessarily yielding directly proportional improvements in inference speed. This ultimately means, latency is not a reliable comparison metric for the models in this research.

For deployment in more demanding scenarios such as public roadways with higher speeds, denser traffic patterns, and unpredictable environmental conditions, the latency requirements would likely be considerably more stringent. In such contexts, a comprehensive approach to system optimization would be necessary, potentially incorporating specialized hardware accelerators, optimized software implementations, and architectural modifications to the inference pipeline. This is where algorithms like (N+M) ES with 1/5th rule can assist by finding a solution in a complex non-linear search space.

This highlights the importance of balancing model performance and resource usage in real-world applications. Hence, our study shows that using ES with the 1/5 success rule to optimize models for either better performance or size reduction does yield practical results that improve application efficiency.

3. CONCLUSION

This research has successfully demonstrated the efficacy of the (N+M) Evolution Strategy with the 1/5th success rule for optimizing neural network architectures in the context of autonomous steering. Through systematic experimentation and comparative analysis, several significant findings have emerged that contribute to both the theoretical understanding and practical application of evolutionary hyperparameter optimization.

The primary contribution of this work is the development of an effective framework for automated hyperparameter optimization that yields significantly more compact neural network architectures while maintaining competitive performance. This research empirically validated that ES-optimized models consistently outperform their random search counterparts across multiple evaluation metrics. Specifically, when compared to traditional random search optimization across 1,000 iterations, the ES approach achieved superior steering angle prediction accuracy while requiring less memory, demonstrating its efficiency in navigating the complex hyperparameter landscape.

The comparative analysis between random search and Evolution Strategy revealed distinct advantages of the latter in terms of convergence speed and solution quality. While random search occasionally discovered high-performing configurations through probabilistic exploration, the adaptive nature of ES with the 1/5th success rule enabled more systematic traversal of the search space, leading to more consistent optimization outcomes. This structured exploration-exploitation balance proved particularly valuable when operating with limited computational resources and constrained datasets.

Furthermore, the demonstrated ability to reduce model size by 75% and 98% (Half-Size and Quarter-Size models respectively) while maintaining acceptable performance represents a significant advancement for resource-constrained autonomous systems. The Quarter-Size model, containing roughly 2% of the parameters of the baseline architecture, successfully navigated the test environment in real-time on the LTU ACTor platform, validating the practical applicability of our approach. This size reduction directly translates to lower computational requirements, reduced energy consumption, and improved inference speeds - all critical factors for embedded deployment in autonomous vehicles.

Real-world validation confirmed that the optimized models maintained robust steering prediction capabilities despite their reduced complexity. The successful deployment on the LTU ACTor platform demonstrated that these lightweight architectures are viable solutions for practical autonomous driving applications within defined operational domains. The models exhibited reliable performance across various conditions, including different driving directions and path positions, indicating effective generalization within the target environment.

In conclusion, this research establishes Evolutionary Strategies (ES) with the 1/5th success rule for hyperparameter optimization as a powerful tool for developing efficient neural network architectures for autonomous steering applications. By systematically comparing ES with random search optimization and demonstrating real-world applicability, this work contributes valuable insights to the ongoing effort to create lightweight, deployable models for autonomous systems. The ability to automatically discover optimized architectures that balance performance with computational efficiency represents an important step toward practical, widespread deployment of autonomous driving technology.

Looking ahead, this work lays a foundation for scalable, adaptive autonomous driving systems that can be incrementally improved with minimal data collection effort. Future research could expand on these results by exploring more advanced meta-learning frameworks, integrating robustness against adversarial scenarios, and validating the approach on larger and more diverse operational domains. Ultimately, the ES-driven optimization and adaptation pipeline presented here offers a practical pathway toward efficient, resilient, and continually learning autonomous vehicle control.

4. REFERENCES

[1] M. Bojarski *et al.*, “End to End Learning for Self-Driving Cars,” Apr. 25, 2016, *arXiv*: arXiv:1604.07316. doi: 10.48550/arXiv.1604.07316.

[2] T.-D. Do, M.-T. Duong, Q.-V. Dang, and M.-H. Le, “Real-Time Self-Driving Car Navigation Using Deep Neural Network,” in *2018 4th International Conference on Green Technology and Sustainable Development (GTSD)*, 2018. doi: 10.1109/GTSD.2018.8595590.

[3] A. G. Howard *et al.*, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” Apr. 17, 2017, *arXiv*: arXiv:1704.04861. doi: 10.48550/arXiv.1704.04861.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2009, pp. 248–255. doi: 10.1109/CVPR.2009.5206848.

[5] J. Snell, K. Swersky, and R. S. Zemel, “Prototypical Networks for Few-shot Learning,” Jun. 19, 2017, *arXiv*: arXiv:1703.05175. doi: 10.48550/arXiv.1703.05175.

[6] H.-G. Beyer and H.-P. Schwefel, “Evolution strategies - A comprehensive introduction,” *Nat. Comput.*, vol. 1, pp. 3–52, Mar. 2002, doi: 10.1023/A:1015059928466.

[7] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, “Evolution Strategies as a Scalable Alternative to Reinforcement Learning,” Sep. 07, 2017, *arXiv*: arXiv:1703.03864. doi: 10.48550/arXiv.1703.03864.

[8] G. Hinton, O. Vinyals, and J. Dean, “Distilling the Knowledge in a Neural Network.” 2015. [Online]. Available: https://arxiv.org/abs/1503.02531

[9] “Autonomous Vehicle Implementation Predictions: Implications for Transport Planning,” SciSpace - Paper. Accessed: Apr. 28, 2025. [Online]. Available: https://scispace.com/papers/autonomous-vehicle-implementation-predictions-implications-3cicqead0t

[10] T. Elsken, J. H. Metzen, and F. Hutter, “Neural Architecture Search: A Survey,” Apr. 26, 2019, *arXiv*: arXiv:1808.05377. doi: 10.48550/arXiv.1808.05377.

[11] S. Garg *et al.*, “Poisoning Attacks on Federated Learning for Autonomous Driving,” May 02, 2024, *arXiv*: arXiv:2405.01073. doi: 10.48550/arXiv.2405.01073.

[12] M. G. Sadovsky, “Evidence for strong co-evolution of mitochondrial and somatic genomes,” May 20, 2014, *arXiv*: arXiv:1405.5128. doi: 10.48550/arXiv.1405.5128.

[13] E. Real *et al.*, “Large-Scale Evolution of Image Classifiers,” Jun. 11, 2017, *arXiv*: arXiv:1703.01041. doi: 10.48550/arXiv.1703.01041.

[14] K. O. Stanley and R. Miikkulainen, “Evolving Neural Networks through Augmenting Topologies,” *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, Jun. 2002, doi: 10.1162/106365602320169811.

[15] S. I. Morga-Bonilla, I. Rivas-Cambero, J. Torres-Jiménez, P. Téllez-Cuevas, R. S. Núñez-Cruz, and O. V. Perez-Arista, “Behavioral Cloning Strategies in Steering Angle Prediction: Applications in Mobile

Robotics and Autonomous Driving," *World Electr. Veh. J.*, vol. 15, no. 11, Art. no. 11, Nov. 2024, doi: 10.3390/wevj15110486.

[16] H. Zhuang *et al.*, "Online Analytic Exemplar-Free Continual Learning with Large Models for Imbalanced Autonomous Driving Task," *IEEE Trans. Veh. Technol.*, vol. 74, no. 2, pp. 1949–1958, Feb. 2025, doi: 10.1109/TVT.2024.3483557.

[17] H. Seong, C. Chung, and D. H. Shim, "Model Parameter Identification via a Hyperparameter Optimization Scheme for Autonomous Racing Systems," *IEEE Control Syst. Lett.*, vol. 7, pp. 1652–1657, 2023, doi: 10.1109/LCSYS.2023.3267041.

[18] F. Faizi and A. Alsulaifanie, "Steering angle prediction via neural networks," *Indones. J. Electr. Eng. Comput. Sci.*, vol. 31, pp. 392–399, Jul. 2023, doi: 10.11591/ijeecs.v31.i1.pp1-1x.

[19] J. Jhung, I. Bae, J. Moon, T. Kim, J. Kim, and S. Kim, "End-to-End Steering Controller with CNN-based Closed-loop Feedback for Autonomous Vehicles," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, 2018, pp. 617–622. doi: 10.1109/IVS.2018.8500440.

[20] G. DeRose, A. Ramsey, J. Dombecki, N. Paul, and C.-J. Chung, "Autonomously steering vehicles along unmarked roads using low-cost sensing and computational systems," *Vehicles*, vol. 5, no. 4, pp. 1400–1422, 2023.

[21] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments-a survey," *IEEE Trans. Evol. Comput.*, vol. 9, no. 3, pp. 303–317, Jun. 2005, doi: 10.1109/TEVC.2005.846356.

[22] W. Vent, "Rechenberg, Ingo, Evolutionsstrategie — Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. 170 S. mit 36 Abb. Frommann-Holzboog-Verlag. Stuttgart 1973. Broschiert," *Feddes Repert.*, vol. 86, no. 5, pp. 337–337, 1975, doi: 10.1002/fedr.19750860506.

[23] R. Vuorio, D.-Y. Cho, D. Kim, and J. Kim, "Meta Continual Learning," Jun. 11, 2018, *arXiv*: arXiv:1806.06928. doi: 10.48550/arXiv.1806.06928.

[24] J. Kirkpatrick *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proc. Natl. Acad. Sci.*, vol. 114, no. 13, pp. 3521–3526, Mar. 2017, doi: 10.1073/pnas.1611835114.

[25] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J Mach Learn Res*, vol. 13, no. null, pp. 281–305, Feb. 2012.

[26] K. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nat. Mach. Intell.*, vol. 1, Jan. 2019, doi: 10.1038/s42256-018-0006-z.

[27] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A Survey of Deep Learning Techniques for Autonomous Driving," *J. Field Robot.*, vol. 37, no. 3, pp. 362–386, Apr. 2020, doi: 10.1002/rob.21918.

[28] M. Bojarski *et al.*, "Explaining how a deep neural network trained with end-to-end learning steers a car," *ArXiv Prepr. ArXiv170407911*, 2017.

[30] D. Butani and R. Kaddis, "LTU-Self-Drive-Sim." 2025. [Online]. Available: https://github.com/Aeolus96/LTU-Self-Drive-Sim

[31] C.-J. Chung, *Knowledge-based approaches to self-adaptation in cultural algorithms*. Wayne State University, 1997.

[32] C.-J. Chung and R. Reynolds, "Knowledge-based self-adaptation in evolutionary search," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 14, no. 01, pp. 19–33, 2000.

[33] C.-J. Chung and R. G. Reynolds, “CAEP: An evolution-based tool for real-valued function optimization using cultural algorithms,” *Int. J. Artif. Intell. Tools*, vol. 7, no. 03, pp. 239–291, 1998.

[34] G. DeRose, “GazelleSim.” 2025. [Online]. Available: https://github.com/gderose2/gazelle_sim