Evolutionary Hyperparameter Optimization to Find Lightweight CNN Models for Autonomous Steering

Devson Butani
Department of Math and Computer Science
Lawrence Technological University
Southfield, MI, USA
dbutani@ltu.edu

Ryan Kaddis
Department of Math and Computer Science
Lawrence Technological University
Southfield, MI, USA
rkaddis@ltu.edu

Giuseppe DeRose Jr.
Department of Math and Computer Science
Lawrence Technological University
Southfield, MI, USA
gderose@ltu.edu

Chan-Jin Chung
Department of Math and Computer Science
Lawrence Technological University
Southfield, MI, USA
cchung@ltu.edu

***Abstract*—This research investigates the optimization of Convolutional Neural Networks (CNNs) for autonomous steering using the (N+M) Evolutionary Strategy (ES) with the 1/5th success rule. The primary objective is to develop a lightweight CNN model 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 dataset consists of timestamped images labeled with steering angles, 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 model size while maintaining competitive predictive accuracy. The results highlight the viability of lightweight CNN architectures for real-time autonomous systems, striking a balance between computational efficiency and performance. This study not only advances research initiatives in using evolutionary strategies for autonomous driving applications but also lays the groundwork for deploying cost-effective, scalable solutions in self-driving technology.**

***Index Terms*—Evolutionary Strategy, Hyperparameter Optimization, Convolutional Neural Networks (CNN), Lightweight CNN, Autonomous Driving, Deep Learning, Real-Time Performance, Autonomous Vehicles, Cameras**

I. 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 on-board processors [**?**]. This research focuses on training and optimizing CNNs leveraging evolutionary strategies (ES) to automate the search for the best-performing and best size-reduced model configurations.

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 [**?**]. 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 deployment feasibility, particularly in resource-constrained on-board processors [**?**]. Minimizing model size while also training with limited data is the core design challenge in autonomous driving systems.

Furthermore, the performance of deep learning models is heavily reliant on the availability of large, diverse, and labeled datasets. In the context of autonomous driving, collecting and annotating such datasets for every conceivable scenario is impractical and cost-prohibitive [**?**] Even though limited data can lead to overfitting, 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 adding small datasets everyime the domain expands and utilize techniques like transfer learning and few shot learning to improve model performance over time [**?**]. While optimization techniques have been extensively explored in various domains, their application to creating lightweight and adaptable models for autonomous driving, specifically for real-time steering angle prediction, remains an active area of research.

Evolutionary Strategies (ES) offer 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 [**?**]. 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 [**?**]. 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.

A. Data Acquisition and Preprocessing

Equipment: The LTU ACTor autonomous driving platform, integrated with the Robot Operating System (ROS), was used for data collection. Sensor data, including images from the vehicle's forward-facing camera and corresponding steering angles, were recorded in rosbag files. This setup ensured synchronized visual and control data, essential for training and evaluating the models.

Environment: Data was collected along a predefined circular path: the red brick path surrounding 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 1 shows an overhead view of the data collection site.

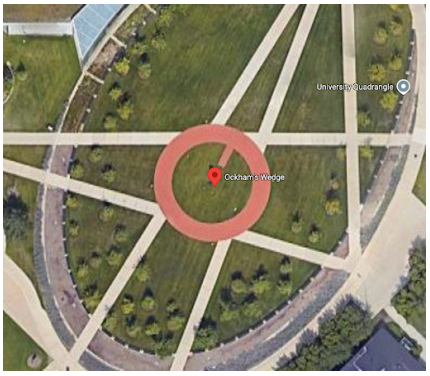

Fig. 1. Aerial view of Ockham's Wedge at Lawrence Technological University. Data was collected from the red brick circle surrounding the art piece.

Extraction and Preprocessing: A custom script was developed to extract and preprocess data from rosbag files.

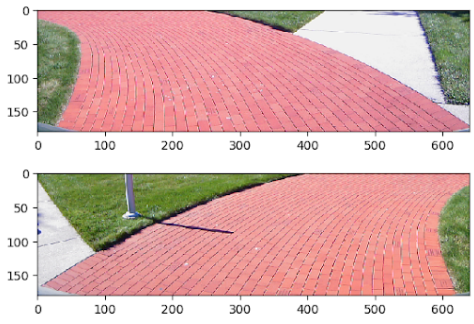

Fig. 2. Sample images after cropping and preprocessing. The extracted steering angles are 12.901 degrees (Top) and -10.099 degrees (Bottom).

Images were saved at 200 ms intervals, with filenames encoding timestamps and steering angles. Each image was cropped (fixed size and position for all images) to exclude irrelevant regions, such as the sky and surrounding buildings. The resulting dataset consisted of 2,957 images, split into 70% training (2,069), 20% validation (592), and 10% testing (296). Even though the dataset is small, it is representative of real-world driving scenarios and provides a useful benchmark for evaluating model performance.

B. Research Goals

Since there are many experiments in image to steering angle prediction that have shown quaility improvements over the years [**?**], [**?**], [**?**], the proposed ES based optimization should allow for a significant improvement in deployable performance and ability to run real-time inference on on-board compute resources. The primary goals of this research are to:

1) Develop a framework to apply the (N+M) Evolutionary Strategy (ES) with the 1/5th success rule for automated hyperparameter tuning.
2) Minimize the size of the baseline CNN model while maintaining satisfactory steering angle prediction performance.
3) Validate the real-world applicability of ES-optimized models in autonomous vehicle systems.

II. Methodology

A. 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 predicted and actual steering angles. The Mean Absolute Error (MAE) serves as a 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 both MSE and MAE is crucial for achieving precise control of the vehicle. These metrics serve as benchmarks for evaluating the efficacy of the Evolutionary Strategy (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 quickly became apparent, rendering it unsuitable for real-world driving scenarios. To address this, the PilotNET architecture, a CNN developed by NVIDIA [**?**] was adopted. PilotNET, an early milestone in autonomous steering research, demonstrated the potential of GPU-accelerated deep learning for this domain. Leveraging a proven architecture allowed for a more effective comparison of ES-optimized models against a recognized standard.

To enhance training efficiency and prevent over optimization, early stopping was implemented. 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.

B. Hyperparameter Management

Initially, the hyperparameter 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, layer units were allowed to vary from 20% to 300% of the PilotNet baseline.

TABLE I
HYPERPARAMETER SEARCH SPACES

Hyperparameter	Search Space
Layer Units	20% to 300% of PilotNet
Batch Size	1 - 32
Learning Rate	0.001 - 1.5
Activation Function	ReLU, eLU, Sigmoid, Tanh
Optimizer	Adam, SGD, RMSprop

However, preliminary experiments indicated that focusing the optimization efforts solely on layer units resulted in improved performance and reduced search time significantly. Consequently, batch size (32), learning rate (0.001), activation function (ReLU), and optimizer (Adam) were fixed to match the baseline model settings. This narrowed scope then allowed the Evolutionary Strategy (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 I.

C. Optimization Framework

CNN hyperparameters, specifically the number of filters in convolutional layers and the number of neurons in fully connected layers, are encoded as genes within an Evolutionary Strategy (ES) framework. The (N+M)-ES approach is employed, iteratively optimizing 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).

D. ES Algorithm

The Evolutionary Strategy (ES) optimization process unfolds as follows:

1) **Initialization:** Generate and train 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:

$$\hat{h} = h + \text{gauss}(\sigma * (\max(h) - \min(h))) \quad (1)$$

where $\hat{h}$ represents the mutated hyperparameter value. h is the original hyperparameter value from the parent model. $\text{gauss}(\mu, \sigma)$ is a random number drawn from a Gaussian distribution with mean μ and standard deviation σ. In this case, $\mu = 0$. 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.
3) **Training and Evaluation:** Train all offspring models and evaluate their performance based on validation loss (MSE) and accuracy (MAE).
4) **Selection:** Retain the top N models (from parents and children) based on validation performance to form the next generation's parent population. This selection process ensures that the most promising models are carried forward, driving the evolution towards improved performance. The mutation step size (σ) is dynamically adjusted using the 1/5 success rule [**?**]:

$$\sigma_{t+1} = \begin{cases} \alpha * \sigma_t & \text{success rate} > \frac{1}{5} \\ \beta * \sigma_t & \text{success rate} \leq \frac{1}{5} \end{cases} \quad (2)$$

where σ_{t+1} 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 success rate is greater than 1/5, 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.

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 VRAM ($\approx$ 80GB) 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 are determined based on the available computational resources and the complexity of the optimization problem.

E. Model Selection

Four different model architectures are proposed for determining the effect of ES on model performance and size. The first model, named "Baseline", is the same architecture established by PilotNET. [**?**] The second model, named "Optimized", contains a CNN and DNN structure that has been optimized with ES, within the search space proposed

by Table I. The third model, named "Half-Size", contains the PilotNET CNN architecture with half of the baseline number of neurons per layer, and half of the search space proposed by Table I. The fourth model, named "Quarter-Size", contains the PilotNET CNN architecture with a quarter of the baseline number of neurons per layer, and a quarter of the search space proposed by Table I. These four models allow the effect of ES on model performance with smaller model sizes to be observed.

TABLE II
MODEL DESCRIPTIONS

Model Name	Description
Baseline	PilotNET architecture, baseline for comparison.
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.

F. 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 [**?**]. The simulation allows for controlled and repeatable testing of the steering control algorithms on the red brick circular path used for training. This serves as the operational design domain for our ES-optimized models. For variance, both directions (clockwise and counterclockwise) were tested however the difference in performance was negligible.

Fig. 3. Aerial vi w of the playing field in the GazelleSim environment.

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 view.
2) **Model Inference:** The captured image is fed into the trained model, which predicts a steering angle. The predicteed steering angle represents the desired direction of the vehicle.
3) **Steering Control:** The predicted steering angle is then translated into a control command that adjusts the vehicle's trajectory within the simulation. The steering angle is converted into a control signal that influences the vehicle's direction and speed. For real-world testing, this control signal is 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.

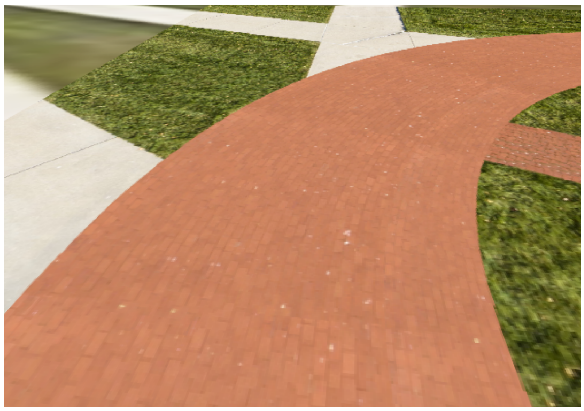

Fig. 4. Perspective view of the playing field from the GazelleSim vehicle.

III. EXPERIMENTAL RESULTS

Table III presents the units per layer of each of the four trained and optimized models. Bolded layers are dynamically optimized using ES with the 1/5 success rule while non-bolded layers are static baseline PilotNET architecture layers modified according to Table II.

TABLE III
ES-OPTIMIZED MODEL ARCHITECTURES

Model	Architecture
Baseline	CNN: 24 >36 >48 >64 >64 > DNN: 1164 >100 >50 >10 >1
Optimized	**CNN: 6 >9 >43 >28 >61 > DNN: 2328 >146 >100 >20** >1
Half-Size	CNN: 12 >18 >24 >32 >32 > **DNN: 149 >14 >20** >1
Quarter-Size	CNN: 6 >9 >12 >16 >16 > **DNN: 8 >3** >1

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 significantly lower power limit ($\approx$ 35W) than the A100 GPU used for training. Even though our models do not saturate the VRAM, the power limit and memory bandwidth

bottlenecks real-time inference latency psuedo simulating a low power onboard compute environment.

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

TABLE IV
TEST RESULTS

Test Duration & Speed	Model	Distance (m)
10 min @ 2 m/s	Baseline	1200.026
	Optimized	1200.072
	Half-Sized	1200.206
	Quarter-Sized	1200.169
60 min @ 2 m/s	Baseline	7200.026
	Optimized	7200.177
	Half-Sized	7200.053
	Quarter-Sized	7200.056
60 min @ 4 m/s	Baseline	14400.408
	Optimized	Failed
	Half-Sized	Failed
	Quarter-Sized	14400.400

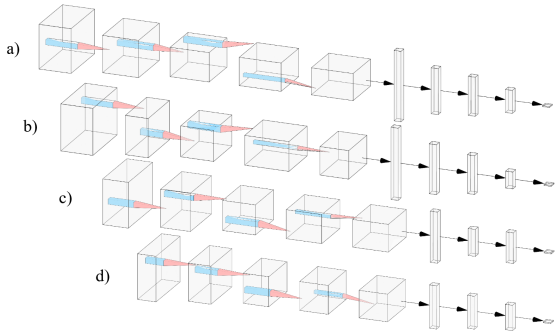

Fig. 5. Visual representation of the models by layer. a) Baseline. b) Optimized. c) Half-Size. d) Quarter-Size.

Table V details the performance and size metrics of each of the four models. The number of parameters and MSE and MAE were direct output from the Keras API. The VRAM usage was measured using the nvidia-smi command while the tests were running.

TABLE V
COMPUTE UTILIZATION

Model	Parameters	Memory (MB)	MSE	MAE
Baseline	245M	936	1.01	0.63
Optimized	250M	1050	0.49	0.41
Half-Size	61.4M	420	1.33	0.78
Quarter-Size	4.97M	146	1.88	0.88

IV. DISCUSSION

The results demonstrate the effectiveness of using evolutionary strategies (ES) with 1/5 success rule to optimize CNN architectures for steering angle prediction in autonomous driving systems and hence meets our goals. Key findings include:

Baseline Performance: 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 average error of 1.01 degrees, and a maximum error 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.

Impact of ES Optimization: The ES-optimized PilotNET model achieved the lowest error (MSE: 0.49, MAE: 0.41 degrees) while the model size increased slightly 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, their prediction accuracy relatively suffered, with the quarter-size model exhibiting the highest error (MSE: 1.88, MAE: 0.88 degrees). Even though a minor increase of error was observed in the quarter-size model, it required only $\approx 15\%$ of the baseline model's VRAM for inference all while achieving 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 tradeoff.

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. Contrastingly, the Quarter-Sized model completed the 60-minute trial with only 0.006m 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.

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 better performance or size reduction does yield practical results that improve application efficiency. In the future, we plan to:

1) Add small datasets by recording driving data on various roads across the LTU campus, including different surfaces, path geometries, and lighting conditions. Techniques like few shot learning and meta continual learning can be explored to further improve model performance.
2) Add Recurrent Neural Network layers to train models based on the past steering inputs. This may improve model stability and smoothness.
3) Deploy the optimized CNN models on the LTU ACTor autonomous driving platform and compare their performance in real-world scenarios.
4) Evaluate these optimized models on low-power compute platforms, such as NVIDIA Jetson or Raspberry Pi, to assess scalability of using ES based optimization for cost-effective systems in autonomous vehicle applications.