Evaluating Evolutionary Algorithms for Neural Network Hyperparameter Optimization

Kai Cho
Auckland University of Technology
October 2025

Abstract

Hyperparameter optimization is critical for achieving optimal performance in neural networks but often requires significant computational resources and expert intuition. This study compares three evolutionary algorithms—Genetic Algorithm (GA), Differential Evolution (DE), and Particle Swarm Optimization (PSO)—with traditional grid and random search methods for optimizing deep learning hyperparameters. Using MNIST and CIFAR-10 benchmarks, the evolutionary methods achieved 0.7–1.8 percentage-point higher test accuracy within comparable compute budgets. PSO showed the fastest early convergence but occasional premature stagnation; GA demonstrated the most consistent results across runs, while DE provided robustness against parameter sensitivity. Each optimization run required approximately 7–8 hours on consumer-grade hardware (NVIDIA GTX 1660). These results provide practical insights for choosing efficient and reliable hyperparameter optimization strategies under real-world computational constraints.

---

1. Introduction

Hyperparameter tuning remains one of the most resource-intensive aspects of modern deep learning workflows. A small change in learning rate, batch size, or optimizer configuration can yield significant differences in accuracy and convergence time. Traditional grid and random search methods offer straightforward implementations but scale poorly with increasing parameter dimensionality (Bergstra & Bengio, 2012). Evolutionary algorithms, inspired by biological evolution, have emerged as flexible alternatives capable of exploring complex, non-linear, and mixed discrete–continuous search spaces.

This research aims to evaluate and compare the performance of three prominent evolutionary algorithms—GA, DE, and PSO—against classical search methods in optimizing neural network hyperparameters. We specifically investigate (1) whether evolutionary algorithms achieve superior accuracy within fixed computational budgets, (2) how their convergence dynamics differ across datasets, and (3) what implementation challenges arise when adapting these algorithms for deep learning optimization tasks.

2. Literature Review

2.1 Background: Hyperparameter Optimization in Neural Networks

Hyperparameter optimization (HPO) plays a central role in determining the learning dynamics and generalization ability of neural networks. Key parameters—learning rate, batch size, optimizer choice, dropout rate, and network depth—strongly influence both convergence stability and ultimate accuracy. Traditionally, researchers relied on manual experimentation and heuristic tuning, but such processes are costly, non-reproducible, and depend heavily on practitioner intuition (Bergstra & Bengio, 2012).

Systematic approaches such as grid and random search offered reproducibility but quickly became infeasible as dimensionality and categorical complexity grew. Random search improved coverage efficiency yet remained uninformed about previously evaluated configurations. Bayesian optimization addressed this by using probabilistic surrogate models—typically Gaussian processes or Tree-structured Parzen Estimators (Snoek et al., 2012; Bergstra et al., 2013)—to guide exploration.

Although effective in low-dimensional continuous domains, these models struggle with discrete or hierarchical search spaces typical in deep neural networks. Consequently, researchers have turned toward metaheuristic approaches such as evolutionary algorithms (EAs), simulated annealing, and swarm intelligence. These population-based algorithms inherently parallelize well, adapt to non-differentiable objectives, and handle mixed discrete–continuous parameters.

Recent frameworks like **Google Vizier** (Golovin et al., 2017), **Optuna** (Akiba et al., 2019), and **Ray Tune** (Liaw et al., 2018) integrate evolutionary or bandit-based HPO strategies into production-scale pipelines, reflecting the growing industrial relevance of evolutionary optimization in MLOps.

2.2 Early Applications of Evolutionary Algorithms in Neural Network Optimization

The origins of evolutionary computation in neural network design date back to the 1980s and 1990s. Holland's (1975) genetic algorithm framework established the principles of selection, crossover, and mutation that later underpinned neural optimization methods. Montana and Davis (1989) first demonstrated the use of GAs for training feedforward networks by evolving connection weights directly, proving that stochastic search could replace gradient-based training in some cases.

Orive et al. (2014) conducted one of the earliest systematic studies on GA-based hyperparameter optimization, comparing crossover operators such as arithmetic, uniform, and two-point schemes. They found evolutionary initialization improved stability relative to random initialization, especially in high-dimensional datasets. Later works, such as Al-Shareef and Abbod (2010) and Huang and Wang (2006), applied GAs to accelerate

convergence or improve feature selection but were constrained by limited compute and shallow architectures.

Neuroevolution frameworks like **NEAT** (Stanley & Miikkulainen, 2002) extended evolutionary design to network topologies, while **CoDeepNEAT** (Miikkulainen et al., 2019) scaled the concept to convolutional networks—though at massive computational cost.

2.3 Recent Advances in Evolutionary Hyperparameter Optimization

In the last five years, evolutionary methods have seen renewed attention as deep learning systems grow increasingly complex. **Vincent and Jidesh (2023)** proposed hybrid Bayesian–Evolutionary AutoML frameworks (BO-DE, BO-GA, BO-CMA-ES) that achieved improved accuracy with 28% less computational cost compared to pure Bayesian search.

Raiaan et al. (2024) categorized HPO techniques into sequential model-based, metaheuristic, numerical, and statistical families, identifying EAs as uniquely suited to mixed discrete–continuous spaces. **Loshchilov and Hutter (2016)** adapted CMA-ES for non-separable hyperparameter landscapes, and **Real et al. (2019)** used Regularized Evolution to discover state-of-the-art CNN architectures—though at high computational cost.

Recent efforts emphasize **multi-objective optimization**: **Assunção et al. (2022)** applied PSO to balance accuracy and training time, while **Dutta et al. (2021)** used DE for robust tuning under noisy feedback. Industrial AutoML systems such as **AWS SageMaker** and **Google Vertex AI** now incorporate EA-inspired population search for scalable parallel training (Amazon Web Services, 2023).

However, most studies use single EAs (usually GA or PSO) without systematic cross-comparison or consistent baselines—highlighting a significant research gap this study addresses.

Table 2.1

Representative Studies on Evolutionary Hyperparameter Optimization

Study	Algorithm	Dataset	Key Findings
Orive et al. (2014)	GA	Chemical regression / ANN	Evolutionary initialization improved accuracy vs. random.
Vincent & Jidesh (2023)	Hybrid (BO+EA)	MNIST, CIFAR-10	Hybrid EA–Bayesian improved accuracy and reduced cost.
Real et al. (2019)	Regularized Evolution	ImageNet	Found architectures rivaling human designs (2000 TPU-hrs).
Assunção et al. (2022)	Multi-objective PSO	CNN benchmarks	Balanced accuracy vs. training time.

Study	Algorithm	Dataset	Key Findings
Dutta et al. (2021)	Differential Evolution	CIFAR-10	Stable tuning under noisy validation signals.
Li et al. (2023)	Benchmark Survey	Multiple	Advocated standardized, resource-aware HPO benchmarks.

2.4 Gaps and Research Motivation

Despite promising results, four persistent gaps exist in the literature:

1. **Lack of comparative analyses:** Few studies evaluate multiple EA paradigms under identical conditions (GA vs. DE vs. PSO).
2. **Weak baselines:** Comparisons to grid or random search are often missing or inconsistent.
3. **Limited integration with MLOps:** Real-world pipeline implementation (e.g., Kubernetes, Airflow) is rarely studied.
4. **Focus on initialization:** Prior works emphasize weight initialization rather than training-phase hyperparameter optimization.

This study directly addresses these issues through a controlled, reproducible benchmark of GA, DE, and PSO using modern frameworks.

2.5 Research Contribution

This research contributes by:

- Conducting the first **standardized cross-family comparison** of GA, DE, and PSO for HPO on MNIST and CIFAR-10.
- Benchmarking against **classical search baselines** (grid and random search).
- Ensuring **reproducibility** through open frameworks (PyTorch, DEAP).
- Analyzing **convergence behavior**, runtime, and robustness under realistic computational limits.

Together, these advances establish an empirical foundation for future hybrid or multi-objective EA-based HPO.

---

3. Methodology

This research used an empirical comparative approach to evaluate the performance of three evolutionary algorithms—Genetic Algorithm (GA), Differential Evolution (DE), and Particle Swarm Optimization (PSO)—against traditional grid and random search methods for neural

network hyperparameter optimization.
Two datasets, **MNIST** and **CIFAR-10**, were chosen for their contrasting complexity: MNIST offers a simple classification landscape, while CIFAR-10 introduces deeper network structures and higher training variance.

All models were implemented in **Python (3.10)** using **PyTorch 2.0** for network training and the **DEAP** framework for evolutionary operations.
The optimization space included:

- Learning rate
- Batch size
- Dropout rate
- Number of hidden layers / filters
- Optimizer type

Each evolutionary algorithm was applied to the **same parameter space and architectures** to ensure fair comparison.
To avoid overfitting, early stopping was applied based on validation loss, and test accuracy was reported from the best checkpoint.

3.1 Experimental Design

Experiments were conducted on a **Windows workstation** with an Intel i7-9700K CPU, 16 GB RAM, and an **NVIDIA GTX 1660 (6 GB)** GPU.
Each optimization run took between **6.5 and 8.5 hours**, with training accounting for roughly 94 percent of the total runtime.

Algorithm Configuration

- **GA**: Population = 20, generations = 30, crossover = 0.7, mutation = 0.2, tournament size = 3.
  During pilots, higher crossover (> 0.8) led to early convergence, while lower (< 0.5) slowed exploration.
- **DE**: Scaling factor F = 0.8, crossover CR = 0.9. One run mistakenly used F = 0.6, revealing slower convergence (~0.3 % accuracy drop).
- **PSO**: Particles = 20, velocity limits [–1.5, 1.5], inertia weight linearly decaying from 0.9 to 0.5.
  Without clamping, roughly 23 % of particles produced invalid configurations (negative learning rates or excessive batch sizes).

Model Architectures

- **MNIST**: Fully connected feed-forward network with up to 3 hidden layers (64–512 neurons), ReLU activation, and dropout 0–0.5.
- **CIFAR-10**: Convolutional network with 2–4 conv layers (32–128 filters), kernel 3×3 or 5×5, and dropout 0–0.4.

Each experiment used an **80–10–10** train/validation/test split.
To improve statistical reliability, every configuration was trained **three times** with random seeds (42, 123, 456).
During early testing, sequential seeds (1, 2, 3) gave inconsistent results, motivating this fixed-seed protocol.

---

4. Results and Discussion

Evolutionary algorithms consistently outperformed the baselines on both datasets.
PSO converged fastest—achieving 95 % of its final accuracy within 12 generations—but stagnated in 37 % of runs.
GA showed the most stable behavior, maintaining low variance across runs, while **DE** required more generations yet proved more tolerant of parameter mis-tuning.

Table 4.1 – Summary of Experimental Results

Method	Dataset	Mean Accuracy (%)	Std Dev (%)	Runtime (hrs)
GA	MNIST	97.6	0.4	7.1
DE	MNIST	97.4	0.5	8.4
PSO	MNIST	97.8	0.3	6.2
Random Search	MNIST	97.1	0.6	5.8
Grid Search	MNIST	96.9	0.2	14.2
GA	CIFAR-10	79.2	1.1	7.3
DE	CIFAR-10	78.5	0.9	8.6
PSO	CIFAR-10	78.8	1.3	6.4
Random Search	CIFAR-10	77.3	1.5	6.0
Grid Search	CIFAR-10	76.8	0.8	14.5

Statistical Analysis

Welch’s t-test ($\alpha = 0.05$, Holm–Bonferroni adjusted) confirmed that evolutionary algorithms significantly outperformed both baselines on all tests except **PSO vs GA** on CIFAR-10 ($p = 0.127$).
In total, evolutionary approaches achieved **0.7–1.8 percentage-point** gains in accuracy with 40–60 % less search cost than grid search.

A notable outlier occurred when GA achieved 82.1 % on CIFAR-10 in generation 8; replication failed (0/5 runs ± 0.5 pp), confirming it as stochastic noise.
PSO’s faster early improvement made it attractive for tight compute budgets, while GA remained more predictable for repeated experiments.
DE’s robustness suggests it could serve as a reliable optimizer in noisy or distributed environments.

---

5. Threats to Validity

Internal validity: Random initialization, seed sensitivity, and early-stopping bias were controlled through three-seed averaging and fixed splits, but residual stochasticity remains.

Construct validity: Accuracy was the sole metric; energy use, runtime, and model complexity were not jointly optimized.

External validity: Results were limited to image-classification tasks on consumer GPUs. Larger transformer-based models or distributed training might change the relative performance of these algorithms.

Implementation reliability: Memory-swap effects were observed when populations exceeded 25 individuals, increasing runtime by ≈ 45 %. CUDA out-of-memory errors required manual batch-size adjustments—practical issues often omitted in controlled benchmarks.

---

6. Conclusions and Practical Recommendations

Evolutionary algorithms provided measurable improvements over grid and random search within constrained budgets.
Key findings:

1. **Random search** remains effective for quick baselines when time is limited.
2. **PSO** delivers rapid gains early but can stall—best for small or time-limited studies.
3. **GA** offers stable, repeatable optimization and balanced exploration–exploitation.
4. **DE** is slower but resilient to hyperparameter errors and noise.

From a deployment perspective, evolutionary methods integrate well with modern MLOps tools like **Airflow** or **Kubernetes**, enabling scalable population-based experiments.
Future work should explore **adaptive population sizing** and **multi-objective variants** that jointly optimize accuracy, runtime, and energy efficiency.

References

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). *Optuna: A next-generation hyperparameter optimization framework.* Proceedings of the 25th ACM SIGKDD Conference, 2623–2631.
Amazon Web Services. (2023). *Amazon SageMaker automatic model tuning.* AWS Documentation.
Assunção, F., Lourenço, N., & Machado, P. (2022). Multi-objective particle swarm optimization for deep learning hyperparameters. *Neurocomputing, 509*, 68–81.
Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research, 13*, 281–305.

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2013). Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems, 25*, 2546–2554.
Dutta, S., Chakraborty, D., & Das, S. (2021). Differential evolution for neural-network hyperparameter tuning under uncertainty. *Applied Soft Computing, 110*, 107631.
Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., & Sculley, D. (2017). Google Vizier: A service for black-box optimization. *Proceedings of the 23rd ACM SIGKDD Conference*, 1487–1495.
Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J., & Stoica, I. (2018). Tune: A research platform for distributed model selection and training. *Proceedings of ICML*, 87, 502–514.
Loshchilov, I., & Hutter, F. (2016). CMA-ES for hyperparameter optimization of deep neural networks. *arXiv preprint arXiv:1604.07269.*
Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., & Hodjat, B. (2019). Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing* (pp. 293–312). Academic Press.
Montana, D. J., & Davis, L. (1989). Training feedforward neural networks using genetic algorithms. *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, 762–767.
Orive, D., Sorrosal, G., Borges, C., Martin, C., & Alonso-Vicario, A. (2014). Evolutionary algorithms for hyperparameter tuning on neural network models. *Proceedings of the European Modelling and Simulation Symposium*, 402–409.
Raiaan, M. A. K., et al. (2024). A systematic review of hyperparameter optimization. *Artificial Intelligence Review.*
Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2019). Regularized evolution for image classifier architecture search. *Proceedings of the AAAI Conference on Artificial Intelligence, 33*(1), 4780–4789.
Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian optimization of machine-learning algorithms. *Advances in Neural Information Processing Systems, 25*, 2951–2959.
Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation, 10*(2), 99–127.
Vincent, A. M., & Jidesh, P. (2023). An improved hyperparameter optimization framework for AutoML systems using evolutionary algorithms. *Scientific Reports, 13*, 4737.