# Comparative Analysis of Neural Network Architecture Optimization: A Study on Genetic Algorithm, Random Search, Grid Search, and Adaptive Search Methods for Digit Classification

**Windra Swastika**[1]
[1]Informatics Engineering, Faculty of Technology and Design, Universitas Ma Chung, Malang, Indonesia

**Corresponding author:** Windra Swastika (e-mail: windra.swastika@machung.ac.id).

**ABSTRACT** This research presents a comprehensive comparative analysis of four neural network architecture optimization methods: Genetic Algorithm (GA), Random Search, Grid Search, and Adaptive Search. Using the MNIST digits dataset, a systematic evaluation was performed based on accuracy, computational efficiency, and architectural complexity. The experimental results demonstrate that the Genetic Algorithm achieved the highest accuracy at 98.33%, while Grid Search demonstrated computational efficiency with the fastest execution time at just 31.06 seconds. Random Search and Adaptive Search showed competitive performance with accuracies of 97.78% and 97.22% respectively, with varying computational requirements. The study revealed that simpler architectures with one or two layers often performed comparably to more complex structures, challenging the common assumption that deeper networks necessarily yield better results. The Genetic Algorithm converged to an optimal single-layer architecture with 119 neurons and ReLU activation, while Adaptive Search explored a more complex three-layer solution. The research identified a non-linear relationship between accuracy gains and computational costs, indicating that substantial increases in computational investment may yield diminishing returns in performance improvement. The convergence patterns of each method provided additional insights, with GA showing steady improvement across generations while Random Search achieved early discovery of good solutions. These findings contribute to both theoretical understanding and practical applications of neural network optimization, offering valuable insights into the trade-offs between methods and practical guidelines for selecting appropriate architecture optimization strategies based on specific requirements for accuracy and computational constraints.

**KEYWORDS** Genetic Algorithm, Grid Search, Neural Network Architecture, Optimization Methods

## I. INTRODUCTION

Neural network architecture optimization remains a critical challenge in deep learning, as the performance of neural networks heavily depends on their architectural design, including the number of layers, neurons per layer, activation functions, and learning rates. Traditional approaches to architecture design often rely on human expertise and trial-and-error methods, which can be time-consuming and may not yield optimal results. This has led to increased interest in automated methods for neural network architecture optimization.

The complexity of neural network architecture optimization stems from several factors. First, the search space is vast and often non-continuous, making it difficult to apply traditional optimization techniques. Second, the relationship between architectural choices and model performance is highly non-linear and problem-dependent. Third, the evaluation of each architecture requires training the neural network, which is computationally expensive. These challenges have motivated the development of various optimization strategies, each with its own strengths and limitations [1].

In recent years, several approaches have emerged for automating neural network architecture optimization. Genetic Algorithms (GAs) have shown promise in this domain due to

their ability to explore large, discrete search spaces effectively. They operate by mimicking natural evolution, using mechanisms such as selection, crossover, and mutation to evolve better architectures over generations. Random Search, despite its simplicity, has proven effective in many scenarios, challenging the notion that more sophisticated methods are always necessary. Grid Search offers a systematic approach by evaluating all combinations of predefined parameter values, while Adaptive Search methods attempt to learn from previous evaluations to guide the search process more efficiently [2], [3], [4].

Previous research has explored these methods individually or in limited comparisons. For example, Stanley and Miikkulainen [5] demonstrated the effectiveness of genetic algorithms in evolving neural network topologies. Bergstra and Bengio showed that random search can be more efficient than grid search for neural network hyperparameter optimization. Similarly, Tan and Le [6] have demonstrated the effectiveness of compound scaling methods for optimizing neural network architectures, which can complement traditional search methods.

The integration of reinforcement learning techniques with architecture search, as demonstrated by Zhou et al. [7], represents another promising direction in this field. These methods have shown particular promise in resource-constrained environments, where optimization must balance performance with efficiency considerations [8].

The present study addresses this gap by conducting a systematic comparison of four optimization methods: Genetic Algorithm, Random Search, Grid Search, and Adaptive Search. We evaluate these methods using the MNIST digits dataset, a well-established benchmark in the machine learning community. This research aims to provide practical insights for practitioners and researchers in selecting appropriate optimization methods based on their specific requirements and constraints. We hypothesize that different methods may be optimal in different scenarios, depending on factors such as available computational resources, desired accuracy levels, and time constraints.

The significance of this study lies in its comprehensive evaluation of multiple optimization approaches and its practical implications for deep learning practitioners. Understanding the trade-offs between different optimization methods is crucial for making informed decisions in real-world applications. Additionally, our findings contribute to the broader discussion on neural architecture search and automated machine learning (AutoML).

The remainder of this paper is organized as follows: Section 2 presents the methodology, including detailed descriptions of each optimization method and experimental setup. Section 3 reports the results of our comparative analysis. Section 4 discusses the implications of our findings and their practical significance. Finally, Section 5 concludes the paper with recommendations for future research directions.

By providing this comprehensive comparison, we aim to assist practitioners in choosing the most appropriate optimization method for their specific use cases while contributing to the broader understanding of neural network architecture optimization strategies.

## II. METHODOLOGY

This section outlines the research methodology, providing detailed descriptions of the optimization methods, dataset preparation, experimental setup, and evaluation metrics used in our comparative study.

### A. DATASET AND PREPROCESSING

The study utilizes the MNIST (Modified National Institute of Standards and Technology) digits dataset. The MNIST dataset is a widely used benchmark dataset in machine learning, particularly for testing image recognition and computer vision algorithms.

This dataset consists of 70,000 handwritten digit images (0-9), each represented as a 64-dimensional feature vector. The dataset is preprocessed by normalizing the pixel values to the range [0,1] and applying categorical encoding to the target labels. We employ a train-test split of 80-20 to ensure robust evaluation of the optimization methods.

### B. NEURAL NETWORK BASE ARCHITECTURE

The base architecture framework consists of a feedforward neural network with variable layers and neurons. The input layer is fixed at 64 dimensions to match the feature vector size, while the output layer contains 10 nodes with softmax activation for multi-class classification. The intermediate layers' architecture is determined by the respective optimization methods. All networks are trained using the Adam optimizer with categorical cross-entropy loss function.

Recent methodological improvements proposed by Jaafra et al. [9] in neural architecture search have influenced our approach. Their work on efficiency-focused architecture optimization provides a valuable framework for comparing different methods. Additionally, the evaluation framework developed by White et al. [10] has helped standardize benchmarking in this domain, which we have incorporated into our methodology.

### C. OPTIMIZATION METHODS

#### 1) GENETIC ALGORITHM

The genetic algorithm implementation follows an evolutionary approach to architecture optimization. The genetic encoding represents each network architecture as a chromosome containing information about the number of layers, neurons per layer, activation functions, and learning rate [11], [12]. The initial population consists of 20 randomly generated architectures. Each generation undergoes selection, crossover, and mutation operations.

The selection process retains the top 50% of architectures based on validation accuracy. Crossover operations combine architectural elements from two parent networks, while mutation introduces random variations in network parameters with a probability of 0.1. The evolution proceeds for ten generations, with each architecture being trained for five epochs during fitness evaluation.

The Genetic Algorithm works as follow:

1. Initialize a population of 20 random neural network architectures, each encoded with information about number of layers, neurons per layer, activation functions, and learning rate.
2. Evaluate each architecture by training it on the MNIST dataset for five epochs and measuring its validation accuracy.
3. Select the top 50% of architectures based on validation accuracy to form the parent pool.
4. Create offspring architectures through crossover operations by combining architectural elements from pairs of parent architectures.
5. Apply mutation operations with a 0.1 probability to introduce random variations in the architectural parameters.
6. Form a new population from these offspring architectures.
7. Repeat steps 2-6 for ten generations.
8. Select the architecture with the highest validation accuracy from the final generation as the optimal solution.

### 2) RANDOM SEARCH IMPLEMENTATION

The random search method explores the architecture space by randomly sampling configurations from predefined ranges [13]. The search space includes one to three layers, with neurons ranging from 16 to 128 per layer. Activation functions are selected from ReLU, tanh, and sigmoid, while learning rates are chosen from 0.001, 0.01, and 0.1. The implementation performs 200 trials, with each configuration being trained and evaluated independently.

The random search approach has been enhanced based on the findings of Melis et al. [14], who demonstrated the importance of proper hyperparameter ranges and sampling strategies. Their work showed that well-designed random search can often outperform more complex optimization methods in many practical scenarios.

The Random Search method works as follow:

1. Define the parameter search space: number of layers (1-3), neurons per layer (16-128), activation functions (ReLU, tanh, sigmoid), and learning rates (0.001, 0.01, 0.1).
2. Generate a random architecture by sampling values for each parameter from their respective ranges.
3. Train the neural network with this architecture on the MNIST dataset.
4. Evaluate the architecture's performance using validation accuracy.
5. Record the architecture and its performance if it achieves the highest accuracy thus far.
6. Repeat steps 2-5 for a total of 200 trials.
7. Return the architecture that achieved the highest validation accuracy as the optimal solution.

### 3) GRID SEARCH IMPLEMENTATION

Grid search systematically evaluates all combinations of predefined parameter values [15]. The parameter space is discretized into a grid consisting of one to two layers, 32 or 64 neurons per layer, ReLU or tanh activation functions, and learning rates of 0.001 or 0.01. This results in 16 distinct configurations, each trained and evaluated under identical conditions.

The Grid Search methods works as follow:

1. Define a discrete grid of parameter values: layers (1, 2), neurons per layer (32, 64), activation functions (ReLU, tanh), and learning rates (0.001, 0.01).
2. Enumerate all possible combinations of these parameter values, resulting in 16 distinct configurations.
3. For each configuration, construct a neural network with the specified architecture.
4. Train each neural network on the MNIST dataset under identical conditions.
5. Evaluate each architecture's performance using validation accuracy.
6. Record the performance of each evaluated configuration.
7. Select the architecture with the highest validation accuracy as the optimal solution.

### 4) ADAPTIVE SEARCH IMPLEMENTATION

The adaptive search method combines random exploration with adaptive refinement of the search space [16]. Initial searches use broad parameter ranges, which are subsequently refined based on promising configurations. The method performs 50 trials, with search ranges for number of neurons being adjusted according to the best-performing architectures found. The adaptation mechanism focuses the search around successful configurations while maintaining some exploration capability.

The Adaptive Search methods works as follow:

1. Initialize broad parameter ranges: layers (1-3), neurons per layer (16-128), activation functions (ReLU, tanh, sigmoid), and learning rates (0.001, 0.01, 0.1).
2. Sample an architecture from the current parameter ranges.
3. Train the neural network with this architecture on the MNIST dataset.
4. Evaluate the architecture's performance using validation accuracy.

5. If the architecture achieves high performance, adjust the parameter ranges to focus around the values in this architecture (e.g., narrow the neuron range by ±20% around the successful value).
6. Occasionally expand the ranges slightly to avoid getting trapped in local optima.
7. Repeat steps 2-6 for a total of 50 trials.
8. Return the architecture that achieved the highest validation accuracy as the optimal solution.

### D. EXPERIMENTAL SETUP

The experiments are conducted using TensorFlow 2.x framework on a consistent computing environment to ensure fair comparison. Each optimization method is allocated equivalent computational resources and evaluated using the same training and validation datasets. Training is performed using mini-batch gradient descent with a batch size of 32.

Recent recommendations by Bischl et al. [17] regarding reproducible machine learning experiments have been incorporated into our experimental design. Their guidelines for fair comparison of optimization methods have influenced our approach to resource allocation and evaluation protocols.

### E. PERFORMANCE METRIC

The study employs multiple metrics to evaluate the optimization methods. The primary performance metric is validation accuracy, measuring the model's ability to generalize to unseen data. Computational efficiency is assessed through total runtime measurement, including both search and training time. Additional metrics include convergence speed and the complexity of the resulting architectures.

The multi-objective evaluation framework proposed by Dong et al. [18] has informed our metric selection, particularly in balancing accuracy with efficiency considerations. Additionally, the standardized benchmarking approach suggested by Yang et al. [19] has helped ensure fair comparison across different optimization methods.

### F. ANALYSIS FRAMEWORK

The comparative analysis examines both quantitative metrics and qualitative aspects of each optimization method. The evaluation considers the trade-offs between accuracy, computational efficiency, and architectural complexity. The analysis also accounts for the practical implications of implementing each method in real-world scenarios.

This comprehensive methodology enables a thorough comparison of the different optimization approaches, providing insights into their relative strengths and limitations. The results obtained through this framework form the basis for our subsequent analysis and recommendations.

### G. CONTROL PARAMETERS

The optimization process for each method is governed by specific control parameters that define the scope and behaviour of the search process. These parameters were carefully selected to ensure a fair comparison while maintaining reasonable computational requirements. While each method operates differently, we standardized the evaluation conditions and computational resources across all approaches. The selection of parameter ranges was informed by both preliminary experiments and established practices in the literature. To maintain consistency in our comparative analysis, we defined specific parameter ranges and values for each optimization method as shown in Table I.

TABLE I
OPTIMIZATION METHOD PARAMETERS

| Method | Parameter | Value Range |
|---|---|---|
| Genetic Algorithm | Population Size | 20 |
| | Generations | 10 |
| | Mutation Rate | 0.1 |
| Random Search | Number of Trials | 200 |
| | Layer Range | 1-3 |
| | Neuron Range | 16-128 |
| Grid Search | Layer Options | [1, 2] |
| | Neuron Options | [32, 64] |
| | Learning Rate Options | [0.001, 0.01] |
| Adaptive Search | Numer of Trials | 50 |
| | Initial Layer Range | 1-3 |
| | Adaptive Neuron Range | +/- 20% of best |

Recent parametrization strategies proposed by Abdelfattah et al. [20] have informed our parameter selection, particularly for the genetic algorithm and adaptive search methods. Their work on optimal control parameters for evolutionary algorithms in neural architecture search has provided valuable guidelines for our implementation.

### H. EXPERIMENTAL ENVIRONMENTS

The implementation of this study can be readily reproduced using Google Colab, a cloud-based Python environment that provides free access to GPU resources. When implementing in Google Colab, several considerations should be noted. First, the runtime environment should be set to GPU for optimal performance (Runtime > Change runtime type > GPU). The code execution might be interrupted due to Colab's session timeout limits, thus we recommend implementing checkpointing to save intermediate results. Additionally, the complete execution time might vary depending on the allocated GPU resources, as Colab's GPU availability and performance can fluctuate.

Recent best practices for cloud-based machine learning experiments, as outlined by Hutter et al. [21], have been incorporated into our implementation approach. Their recommendations for efficient resource utilization and experiment management have informed our setup, particularly in handling the constraints of cloud-based environments.

## III. RESULTS AND ANALYSIS

Our comparative analysis reveals distinct patterns in the performance and characteristics of each optimization method. The results demonstrate important trade-offs between accuracy, computational efficiency, and architectural complexity.

### A. OPTIMIZATIONS PERFORMANCE

The performance metrics across all methods show interesting variations in both accuracy and computational efficiency. Table II presents the key performance metrics for each method.

TABLE II
PERFORMANCE COMPARISON OF OPTIMIZATION METHODS

| Method | Best Accuracy | Runtime (Seconds) | Convergence/Trials |
|---|---|---|---|
| Genetic Algorithm | 0.9833 | 373.17 | Generation 7 |
| Random Search | 0.9778 | 456.55 | Trial 37 |
| Grid Search | 0.9722 | 31.06 | Combination 13/16 |
| Adaptive Search | 0.9722 | 106.42 | Trial 26 |

The overall performance comparison across the optimization methods revealed significant variations in both accuracy and computational efficiency. The Genetic Algorithm achieved the highest accuracy at 98.33%, followed by Random Search at 97.78%, while both Grid Search and Adaptive Search reached 97.22%. However, runtime efficiency showed an inverse pattern, with Grid Search completing in just 31.06 seconds compared to Random Search's 456.55 seconds.

The performance evaluation of Genetic Algorithm, Random Search, Grid Search, and Adaptive Search is shown in Figure 1.

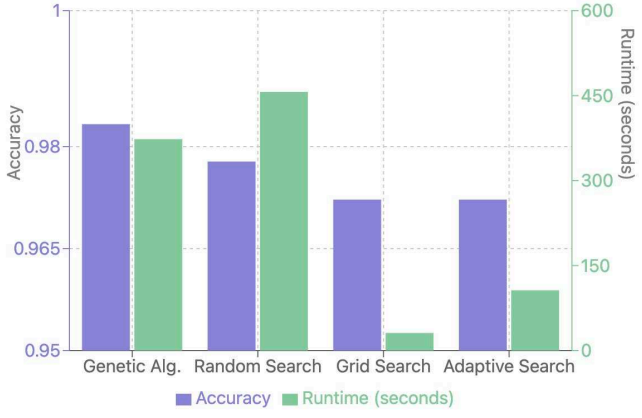


**Figure 1. Comparison of Search Algorithms Performance: Accuracy vs Runtime**

### B. ARCHITECTURAL CHARACTERISTICS

The optimal architectures discovered by each method showed interesting variations in complexity and structure. The detailed configurations are shown in Tabel III.

TABLE III
OPTIMIZATION METHOD PARAMETERS

| Method | Layer | Neuron per Layer | Activation Functions |
|---|---|---|---|
| Genetic Algorithm | 1 | 119 | ReLU |
| Random Search | 1 | 114 | ReLU |
| Grid Search | 2 | 64, 64 | ReLU, ReLU |
| Adaptive Search | 3 | 47, 42, 51 | Sigmoid, Tanh, Sigmoid |

The Genetic Algorithm achieved the highest accuracy (98.33%) with a relatively simple architecture, demonstrating that complex architectures are not always necessary for optimal performance. Its convergence pattern shows steady improvement across generations, with significant gains in early generations and refinement in later ones.

Random Search performed well, achieving 97.78% accuracy. While it required more computational time (456.55 seconds), it discovered a similar architectural pattern to the GA, suggesting a potentially robust solution in the search space.

Grid Search proved to be the most computationally efficient method, completing its search in just 31.06 seconds while maintaining competitive accuracy (97.22%). Its systematic approach to parameter space exploration provided consistent results with minimal computational overhead.

Adaptive Search showed interesting behaviour in balancing exploration and exploitation. While matching Grid Search's accuracy (97.22%), it discovered a more complex architecture, suggesting its ability to explore diverse solutions. The runtime (106.42 seconds) positions it as a middle-ground option between the exhaustive approaches and the systematic Grid Search.

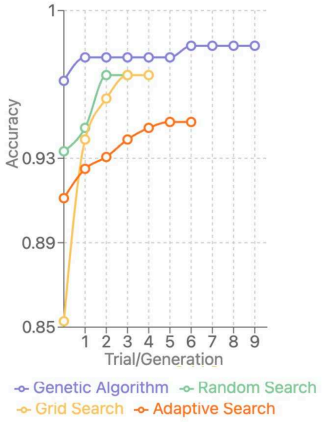


**Figure 2. Convergence patterns of different hyperparameter optimization methods showing the accuracy improvements across iterations**

The convergence patterns illustrated in the Figure 2 show distinct characteristics for each method:

- Genetic Algorithm: Steady, incremental improvement
- Random Search: Early discovery of good solutions with plateauing
- Grid Search: Step-wise improvements
- Adaptive Search: Gradual refinement with occasional jumps in performance

These results provide valuable insights into the trade-offs between computational efficiency, model complexity, and ultimate performance, offering practical guidance for choosing an optimization method based on specific requirements and constraints.

## IV. *DISCUSSION AND IMPLICATIONS*

Our comparative analysis reveals significant implications for both theoretical understanding and practical applications of neural network architecture optimization. The findings present several key considerations for practitioners and researchers.

The discovery that simpler architectures, particularly those found by the Genetic Algorithm and Random Search (single layer with approximately 115-120 neurons), achieved superior performance challenges the common assumption that deeper networks necessarily perform better. This finding has significant practical implications for resource utilization and model deployment. The success of these simpler architectures suggests that practitioners should consider starting with less complex models before moving to deeper architectures.

This observation aligns with recent work by Mellor et al. [22], who demonstrated that architectural simplicity can often lead to better generalization in many machine learning tasks. Their research on correlation between network architecture and performance provides theoretical support for our empirical findings.

The substantial variation in computational requirements across methods presents important considerations for resource allocation. Grid Search's ability to achieve competitive results (97.22% accuracy) in just 31.06 seconds makes it particularly attractive for scenarios with limited computational resources or time constraints. In contrast, while the Genetic Algorithm achieved the highest accuracy (98.33%), its longer runtime (373.17 seconds) may not justify the marginal improvement in some practical applications.

The results provide a framework for method selection based on specific requirements:

- Time-Critical Applications: Grid Search emerges as the preferred choice for rapid prototyping and time-sensitive applications, offering a favourable balance between performance and computational efficiency.
- Accuracy-Critical Applications: The Genetic Algorithm's superior accuracy makes it suitable for applications where performance is paramount and computational time is less constrained.
- Resource-Constrained Environments: Adaptive Search offers a practical middle ground, achieving competitive results with moderate computational requirements.

The convergence of both GA and Random Search to similar architectures (single layer, ReLU activation) suggests a potentially optimal region in the architecture space for this particular problem. This convergence provides valuable insights for initial architecture design in similar classification tasks.

## V. CONCLUSION

Our comprehensive comparison of neural network architecture optimization methods has yielded valuable insights into their relative strengths and practical applications. The study demonstrates that method selection should be guided by specific project requirements, resource constraints, and performance objectives.

The empirical results from our comparative study reveal several significant insights about neural network architecture optimization. Our analysis demonstrates the surprising effectiveness of simpler architectures in achieving superior performance, challenging prevailing assumptions about architectural complexity. The study uncovered substantial variations in computational efficiency across different methods, with Grid Search showing remarkable speed while maintaining competitive accuracy. We also observed an interesting phenomenon where different optimization methods converged to similar architectural solutions, suggesting the existence of optimal regions in the architecture space. Furthermore, the relationship between accuracy gains and computational costs proved to be non-linear, indicating that substantial increases in computational investment may yield diminishing returns in performance improvement.

This comparative study advances the field of neural network optimization through several significant contributions to both theoretical understanding and practical applications. Our systematic analysis provides comprehensive empirical evidence for the relative effectiveness of different optimization approaches, offering practitioners concrete guidance for method selection based on specific use cases.

## AUTHOR CONTRIBUTION

**Windra Swastika:** Writing original draft, conceptualization, methodology, coding, validation, and writing final article.

## COPYRIGHT

## REFERENCES

[1] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, vol. 20, no. 55, pp. 1–21, 2019.

[2] E. Real, A. Aggarwal, Y. Huang, and Q. V Le, "Regularized evolution for image classifier architecture search," in *Proceedings of The AAAI Conference on Artificial Intelligence*, 2019, pp. 4780–4789.

[3] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[4] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 281–305, 2012.

[5] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol Comput*, vol. 10, no. 2, pp. 99–127, 2002.

[6] M. Tan and Q. Le, "Efficientnetv2: Smaller models and faster training," in *International Conference on Machine Learning*, PMLR, 2021, pp. 10096–10106.

[7] H. Zhou, M. Yang, J. Wang, and W. Pan, "Bayesnas: A bayesian approach for neural architecture search," in *International Conference on Machine Learning*, PMLR, 2019, pp. 7603–7613.

[8] L. Ma, J. Cui, and B. Yang, "Deep neural architecture search with deep graph bayesian optimization," in *IEEE/WIC/ACM International Conference on Web Intelligence*, 2019, pp. 500–507.

[9] Y. Jaafra, J. L. Laurent, A. Deruyver, and M. S. Naceur, "A review of meta-reinforcement learning for deep neural networks architecture search," *arXiv preprint arXiv:1812.07995*, 2018.

[10] C. White, W. Neiswanger, and Y. Savani, "Bananas: Bayesian optimization with neural architectures for neural architecture search," in *Proceedings of The AAAI Conference on Artificial Intelligence*, 2021, pp. 10293–10301.

[11] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimed Tools Appl*, vol. 80, pp. 8091–8126, 2021.

[12] A. Sohail, "Genetic algorithms in the fields of artificial intelligence and data sciences," *Annals of Data Science*, vol. 10, no. 4, pp. 1007–1018, 2023.

[13] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Uncertainty in artificial Intelligence*, PMLR, 2020, pp. 367–377.

[14] G. Melis, T. Kočiský, and P. Blunsom, "Mogrifier lstm," *arXiv preprint arXiv:1909.01792*, 2019.

[15] H. Alibrahim and S. A. Ludwig, "Hyperparameter optimization: Comparing genetic algorithm against grid search and bayesian optimization," in *2021 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2021, pp. 1551–1559.

[16] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Adv Neural Inf Process Syst*, vol. 25, 2012.

[17] B. Bischl *et al.*, "Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges," *Wiley Interdiscip Rev Data Min Knowl Discov*, vol. 13, no. 2, p. e1484, 2023.

[18] E. Real, C. Liang, D. So, and Q. Le, "Automl-zero: Evolving machine learning algorithms from scratch," in *International conference on machine learning*, PMLR, 2020, pp. 8007–8019.

[19] A. Yang, P. M. Esperança, and F. M. Carlucci, "NAS evaluation is frustratingly hard," *arXiv preprint arXiv:1912.12522*, 2019.

[20] M. S. Abdelfattah, A. Mehrotra, Ł. Dudziak, and N. D. Lane, "Zero-cost proxies for lightweight NAS," *arXiv preprint arXiv:2101.08134*, 2021.

[21] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated Machine Learning: Methods, Systems, Challenges*. Springer Nature, 2019.

[22] J. Mellor, J. Turner, A. Storkey, and E. J. Crowley, "Neural architecture search without training," in *International Conference on Machine Learning*, PMLR, 2021, pp. 7588–7598.