International Journal of Emerging Trends in Computer Science and Information Technology
ISSN: 3050-9246 | https://doi.org/10.63282/30509246/IJETCSIT-V2I4P103
Eureka Vision Publication | Volume 2, Issue 4, 21-32, 2021

Original Article

Automated Hyperparameter Optimization for Deep Neural Networks Using Bayesian Optimization and Genetic Algorithms

Richards Heiden
Department of Computer Science, University of Helsinki, Finland.

Abstract - Deep Neural Networks (DNNs) have achieved remarkable success in various domains, including computer vision, natural language processing, and reinforcement learning. However, the performance of these models is highly dependent on the choice of hyperparameters, which are often set manually through trial and error. This process is time-consuming, resource-intensive, and requires significant expertise. To address this challenge, this paper explores the use of automated hyperparameter optimization (HPO) techniques, specifically Bayesian Optimization (BO) and Genetic Algorithms (GA), to improve the efficiency and effectiveness of hyperparameter tuning for DNNs. We provide a comprehensive review of the theoretical foundations of BO and GA, discuss their implementation in the context of DNNs, and evaluate their performance on a variety of benchmark datasets. Our results demonstrate that both BO and GA can significantly enhance the performance of DNNs, with BO generally outperforming GA in terms of convergence speed and final model performance. We also discuss the limitations and potential future directions for research in this area.

Keywords - Bayesian Optimization, Genetic Algorithm, Hyperparameter Optimization, Deep Neural Networks, Convergence Speed, Computational Cost, Validation Accuracy, Machine Learning, Neural Architecture Search, Optimization Techniques

1. Introduction

Deep Neural Networks (DNNs) have revolutionized the field of machine learning by achieving state-of-the-art performance on a wide range of tasks. However, the success of DNNs is often contingent on the careful selection of hyperparameters, which include learning rates, batch sizes, regularization parameters, and network architectures. Hyperparameters are crucial because they control the learning process and the model's capacity, and suboptimal choices can lead to poor performance, overfitting, or underfitting. Traditionally, hyperparameters are set manually through a process of trial and error, which is both time-consuming and resource-intensive. This manual tuning process often requires significant domain expertise and can be a bottleneck in the development of DNNs. To overcome this challenge, automated hyperparameter optimization (HPO) techniques have been developed to systematically search for the best hyperparameters. Among these techniques, Bayesian Optimization (BO) and Genetic Algorithms (GA) have emerged as promising approaches due to their ability to efficiently explore the hyperparameter space and find near-optimal solutions. In this paper, we provide a detailed exploration of BO and GA for HPO in the context of DNNs. We begin by reviewing the theoretical foundations of BO and GA, highlighting their key differences and similarities. We then discuss the implementation of these techniques for DNNs, including the design of the search space, the choice of acquisition functions for BO, and the selection of genetic operators for GA. Finally, we evaluate the performance of BO and GA on a variety of benchmark datasets and compare their results with those of manual tuning and other HPO methods.

2. Background

2.1 Hyperparameter Optimization in Deep Learning

Hyperparameter optimization (HPO) is a critical process in deep learning that involves finding the best combination of hyperparameters to maximize a model's performance. Hyperparameters can be broadly categorized into two types: training hyperparameters and model hyperparameters. Training hyperparameters include parameters such as the learning rate, batch size, number of epochs, and optimization algorithms (e.g., Adam, SGD), which directly affect the training dynamics and convergence speed of the model. Model hyperparameters, on the other hand, define the architecture and regularization strategies, including the number of layers, neurons per layer, activation functions, dropout rates, and initialization methods. The choice of hyperparameters significantly impacts the performance of a deep neural network (DNN). For example, selecting an inappropriate learning rate can lead to slow convergence or cause the training process to diverge. Similarly, the architecture of the network determines its capacity to learn and generalize, with overly complex models prone to overfitting and excessively simple models struggling to capture intricate patterns in the data. Thus, systematic optimization techniques are required to efficiently navigate the high-dimensional and often non-convex hyperparameter space to achieve optimal model performance.

2.2 Bayesian Optimization (BO)

Bayesian Optimization (BO) is a popular method for optimizing expensive black-box functions, making it particularly well-suited for hyperparameter tuning in deep learning. Unlike brute-force methods such as grid search or random search, BO uses probabilistic models to guide the search, ensuring efficient exploration of the hyperparameter space. This is crucial in deep learning, where training a single model instance can be computationally expensive.

2.2.1 Gaussian Processes

At the core of BO is the Gaussian Process (GP), a non-parametric probabilistic model that defines a distribution over functions. A GP is characterized by a mean function, which represents the expected value of the function at any given input, and a covariance function (or kernel), which measures the similarity between different points in the input space. The GP is used to model the relationship between hyperparameter configurations and corresponding model performance metrics, such as validation accuracy. Given a set of observed data points, where the inputs are hyperparameter settings and the outputs are model performance scores, a GP can predict the expected performance at any new hyperparameter configuration. The predictive distribution follows a normal distribution with a mean and variance derived from the observed data. The mean function provides an estimate of the function's output at a given point, while the variance indicates the level of uncertainty associated with the prediction. This enables BO to balance exploration (evaluating uncertain regions) and exploitation (focusing on promising areas) efficiently.

2.2.2 Acquisition Functions

To guide the search for the optimal hyperparameters, BO relies on an acquisition function that determines the next point to evaluate based on the GP's predictions. The acquisition function balances exploration and exploitation by considering both the predicted mean and uncertainty in the hyperparameter space. Several common acquisition functions are used in BO. The Expected Improvement (EI) function measures the expected improvement over the best observed performance, prioritizing regions that are likely to yield better results. The Probability of Improvement (PI) function calculates the likelihood of achieving a better performance than the current best, directing the search towards promising areas. The Upper Confidence Bound (UCB) function incorporates an exploration term to encourage sampling in regions of high uncertainty, ensuring that the optimization process does not get stuck in local optima. By optimizing the acquisition function iteratively, BO efficiently identifies the best set of hyperparameters while minimizing the number of function evaluations.

2.3 Genetic Algorithms (GA)

Genetic Algorithms (GA) are another widely used optimization technique inspired by the principles of natural selection. Unlike BO, which relies on probabilistic modeling, GA uses evolutionary strategies to explore and exploit the hyperparameter space. GA operates on a population of candidate solutions and evolves this population over multiple generations to find the optimal hyperparameter configuration.

2.3.1 Key Components of GA

GA consists of several key components that mimic biological evolution. The process begins with population initialization, where an initial set of candidate solutions (hyperparameter configurations) is generated randomly or using a heuristic approach. The performance of each individual solution is then evaluated using a fitness function, which, in the context of HPO, is typically the validation accuracy of the DNN. Selection is a crucial step in GA, as it determines which individuals will contribute to the next generation. Various selection methods exist, including tournament selection, where individuals compete, and the best ones are chosen, and roulette wheel selection, which assigns selection probabilities based on fitness scores. Once the mating pool is formed, crossover operations combine genetic information from parent solutions to create offspring. Different crossover techniques, such as single-point, multi-point, and uniform crossover, can be applied to mix genetic material effectively. Mutation introduces randomness into the population by altering certain genes of an individual. This helps maintain diversity and prevents the algorithm from converging prematurely to suboptimal solutions. Common mutation techniques include bit-flip mutation for binary representations and Gaussian mutation for continuous hyperparameters. By incorporating crossover and mutation, GA ensures continuous exploration of the hyperparameter space while refining the best solutions.

2.3.2 GA Algorithm

The GA optimization process follows a structured sequence of steps. Initially, a population of candidate solutions is generated, and their fitness is evaluated. The selection process then determines which individuals proceed to reproduction. The selected individuals undergo crossover to generate new offspring, followed by mutation to introduce diversity. The new population replaces the previous one, and the cycle repeats until a predefined stopping criterion is met, such as a fixed number of generations or convergence of the best solution. GA's ability to explore large and complex search spaces makes it a powerful tool for HPO. Unlike traditional optimization methods that may struggle with non-convexity and high-dimensionality, GA can efficiently navigate diverse search spaces by leveraging evolutionary principles. This makes it particularly useful for optimizing DNN architectures and training parameters, where the relationships between hyperparameters and performance are often nonlinear and difficult to model explicitly.

3. Implementation of BO and GA for DNNs

3.1 Search Space Design

The first step in implementing Bayesian Optimization (BO) and Genetic Algorithms (GA) for Deep Neural Networks (DNNs) is defining the search space for hyperparameters. The search space includes various hyperparameters that influence model performance, such as learning rate, weight decay, number of layers, number of neurons per layer, and activation functions. The nature of these hyperparameters can be continuous, discrete, or categorical. Continuous hyperparameters, such as the learning rate and weight decay, take values within a predefined range, whereas discrete hyperparameters, such as the number of layers and neurons, have a finite set of possible values. Categorical variables, such as activation functions, take values from a predefined set of choices. Properly defining the search space is crucial, as it determines the effectiveness and efficiency of the optimization process.

3.1.1 Continuous Variables

Continuous hyperparameters play a crucial role in the training dynamics of DNNs. One of the most important continuous variables is the learning rate, which dictates how much the model updates its weights during training. Typically, the learning rate is searched within a range from 10^{-5} to 1. A very high learning rate may cause instability in training, while a very low learning rate may lead to slow convergence. Another key continuous variable is weight decay, which serves as a form of L2 regularization to prevent overfitting. Weight decay values are typically explored within a range from 10^{-5} to 1, and an appropriate weight decay helps in generalization by penalizing large weight values.

3.1.2 Discrete Variables

Discrete hyperparameters define structural components of the network and can significantly impact its representational capacity. The number of layers is a critical discrete variable, commonly chosen from a set such as {2,3,4,5}. Increasing the number of layers allows the model to learn more complex patterns but may lead to overfitting if the dataset is small. Similarly, the number of neurons per layer determines the capacity of each layer and is typically selected from a set like {32,64,128,256}. Larger neuron counts can enhance the network's ability to capture intricate patterns but also increase computational costs. Another discrete variable is the activation function, which determines how neurons process inputs. Common activation functions include ReLU, Sigmoid, and Tanh, each with distinct properties affecting gradient flow and training efficiency.

3.2 Bayesian Optimization for DNNs

Bayesian Optimization (BO) is a probabilistic method that efficiently searches for optimal hyperparameters by modeling the objective function with a Gaussian Process (GP). This approach is particularly useful when training DNNs, as evaluating each set of hyperparameters is computationally expensive. By iteratively refining predictions and selecting promising hyperparameter configurations, BO efficiently converges to an optimal solution.

3.2.1 Gaussian Process Setup

To apply BO to DNN hyperparameter tuning, a Gaussian Process (GP) is used to model the relationship between hyperparameters and model performance. The GP is trained on observed data points, where each data point consists of a set of hyperparameters and the corresponding validation accuracy. The GP provides a predictive distribution, enabling the selection of hyperparameters with the highest probability of improving model performance. Since the objective function is typically noisy, a noise term is included in the GP model to account for variations in performance across training runs.

3.2.2 Acquisition Function Selection

A crucial component of BO is the acquisition function, which determines the next hyperparameter configuration to evaluate. The acquisition function balances exploration, which seeks new regions of the search space, and exploitation, which refines promising areas. Among the commonly used acquisition functions, the Expected Improvement (EI) function is recommended for DNN optimization. EI measures the expected improvement over the current best validation accuracy, effectively guiding the search towards regions with higher potential performance. This makes EI well-suited for hyperparameter tuning, where balancing exploration and exploitation is critical.

3.2.3 BO Algorithm

The Bayesian Optimization process follows an iterative cycle to refine hyperparameter selection:

1. Initialize the GP: Generate an initial set of hyperparameter configurations and evaluate the model performance for each.
2. Train the GP: Use the collected data points to train the Gaussian Process model.
3. Optimize the Acquisition Function: Identify the next set of hyperparameters to evaluate by maximizing the acquisition function.
4. Evaluate the Model: Train the DNN using the selected hyperparameters and measure the validation accuracy.
5. Update the GP: Incorporate the new data point (hyperparameters and corresponding accuracy) into the dataset and retrain the GP.

6. Check Stopping Criterion: If the stopping criterion (e.g., a predefined number of iterations or convergence threshold) is met, terminate the optimization process. Otherwise, repeat steps 3–6.

By iterating through this process, BO efficiently narrows down the search space, identifying hyperparameter configurations that yield optimal model performance.

3.3 Genetic Algorithms for DNNs

Genetic Algorithms (GA) take an evolutionary approach to hyperparameter optimization by simulating the process of natural selection. By maintaining a population of candidate solutions and applying genetic operators such as selection, crossover, and mutation, GA explores the search space efficiently. GA is particularly useful for optimizing high-dimensional and non-convex search spaces, making it a suitable choice for DNN hyperparameter tuning.

3.3.1 Population Initialization

The first step in GA is population initialization, where an initial set of candidate hyperparameter configurations is generated. This can be done randomly or using heuristics based on prior knowledge of well-performing hyperparameters. A well-initialized population can speed up convergence by starting the search in promising regions of the hyperparameter space.

3.3.2 Fitness Function

The fitness function evaluates the quality of each individual in the population. In the context of DNN optimization, the fitness function is the validation accuracy of the model trained with a given set of hyperparameters. The goal of GA is to maximize this fitness function by evolving the population over multiple generations.

3.3.3 Selection, Crossover, and Mutation

Three main genetic operators drive the evolution of the population:

- Selection: Tournament selection is commonly used, where a subset of the population is randomly chosen, and the individual with the highest fitness is selected for reproduction.
- Crossover: Single-point crossover is employed, where two parent solutions swap genetic material at a randomly chosen point, generating offspring with combined features. This helps propagate beneficial traits while introducing diversity.
- Mutation: Bit-flip mutation is applied to introduce small random changes in offspring. This prevents premature convergence and maintains diversity in the population, ensuring thorough exploration of the search space.

3.3.4 GA Algorithm

The GA optimization process follows an iterative evolutionary cycle:

1. Initialize the Population: Generate an initial population of candidate hyperparameter configurations.
2. Evaluate Fitness: Train the DNN using each set of hyperparameters and compute the validation accuracy.
3. Selection: Use tournament selection to choose individuals for reproduction.
4. Crossover: Apply single-point crossover to generate new offspring.
5. Mutation: Introduce random changes in offspring to maintain diversity.
6. Replace Population: Form a new population using the offspring.
7. Check Stopping Criterion: If a stopping criterion (e.g., maximum generations or convergence) is met, terminate the algorithm. Otherwise, return to step 2.

3.4 Genetic Algorithms for DNNs

The process of hyperparameter optimization in deep neural networks (DNNs) using Bayesian Optimization (BO) and Genetic Algorithms (GA). At the top of the diagram, the Hyperparameter Search component is responsible for defining the search space, which includes both continuous and discrete hyperparameters such as learning rate, weight decay, number of layers, and activation functions. This component serves as the foundation for the optimization process, directing the search through two different methodologies BO and GA. On the left side of the diagram, Bayesian Optimization (BO) is depicted as a key method for hyperparameter tuning. It operates by initializing a Gaussian Process (GP), which models the relationship between hyperparameters and model performance. BO sequentially selects new hyperparameter configurations based on an acquisition function, evaluates their effectiveness by training a DNN, and updates the GP model. This iterative approach enables BO to efficiently search the hyperparameter space by balancing exploration (trying new configurations) and exploitation (refining the best-found configurations). On the right side of the diagram, Genetic Algorithms (GA) represent an alternative approach to hyperparameter optimization. GA follows an evolutionary strategy, starting with a randomly initialized population of hyperparameter sets. Over multiple generations, GA applies selection, crossover, and mutation operations to evolve the population toward better-performing configurations.

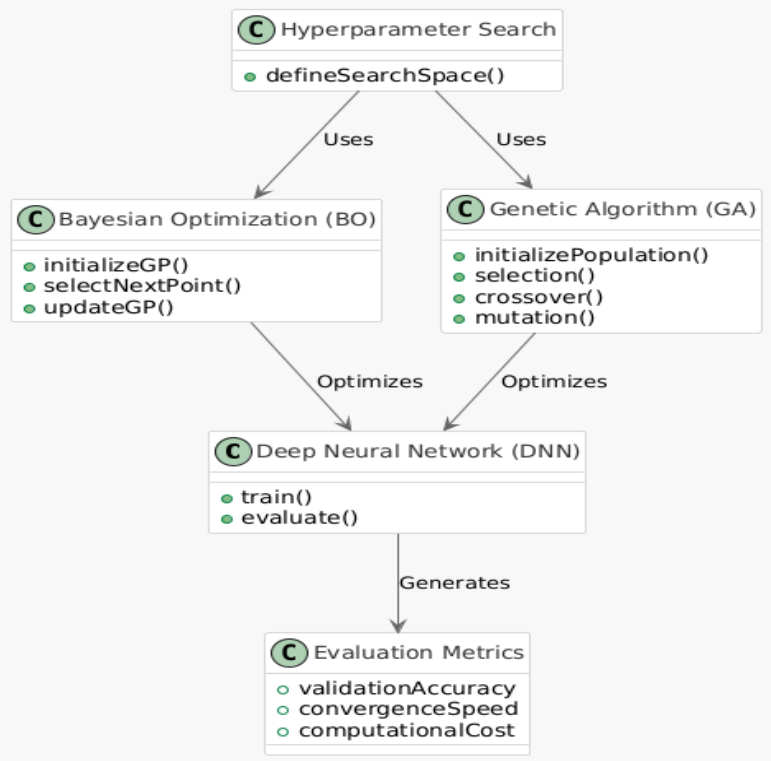

Figure 1. Hyperparameter Optimization Architecture

By maintaining a diverse pool of candidates and iteratively refining them, GA explores the search space in a manner inspired by natural selection. Both optimization methods BO and GA—are connected to the Deep Neural Network (DNN), which acts as the core computational model. The DNN is trained using the selected hyperparameter configurations and then evaluated to measure its performance. The results of the evaluation are fed back into the optimization methods, allowing them to improve future selections. This iterative process continues until an optimal set of hyperparameters is identified. At the bottom of the diagram, the Evaluation Metrics component captures the key performance indicators used to assess the effectiveness of each optimization method. These metrics include validation accuracy, which measures the model's predictive performance; convergence speed, which tracks how quickly the optimization process finds a high-performing configuration; and computational cost, which quantifies the total resources consumed. By analyzing these metrics, researchers and practitioners can compare the strengths and weaknesses of BO and GA in different deep learning applications.

4. Experimental Setup

To rigorously evaluate the performance of Bayesian Optimization (BO) and Genetic Algorithms (GA) for hyperparameter optimization (HPO) in Deep Neural Networks (DNNs), we conduct experiments using widely recognized benchmark datasets and well-established deep learning architectures. The experimental setup is structured to ensure fair and meaningful comparisons between the two optimization approaches. The setup includes selecting diverse datasets, defining suitable DNN architectures, establishing a well-defined hyperparameter search space, and determining evaluation metrics to measure the effectiveness of the optimization techniques.

4.1 Datasets

To assess the generalization and robustness of BO and GA for hyperparameter tuning, we utilize three standard benchmark datasets spanning different domains:

- MNIST: The MNIST dataset consists of 70,000 grayscale images of handwritten digits (0-9), with 60,000 training samples and 10,000 test samples. Each image is 28×28 pixels, and the task is to classify the digit present in the image. MNIST is a widely used benchmark for evaluating image classification models and serves as a suitable dataset for testing the effectiveness of hyperparameter tuning in simple Convolutional Neural Networks (CNNs).
- CIFAR-10: The CIFAR-10 dataset consists of 60,000 color images (32×32 pixels) categorized into 10 different classes, including objects such as airplanes, automobiles, birds, and cats. It is a more challenging dataset than MNIST due to its

complex and diverse images, making it a good benchmark for evaluating the optimization performance in more advanced CNN architectures. The dataset is divided into 50,000 training images and 10,000 test images.

- IMDB Reviews: The IMDB movie review dataset contains 50,000 textual reviews labeled as positive or negative for sentiment analysis. The dataset is split equally into 25,000 training samples and 25,000 test samples. Unlike image datasets, this dataset requires Natural Language Processing (NLP) techniques and is best handled using Recurrent Neural Networks (RNNs). The inclusion of this dataset allows us to evaluate the effectiveness of hyperparameter optimization for text-based deep learning models.

By using these datasets, we ensure that the experimental results apply to both computer vision (CV) and natural language processing (NLP) tasks, highlighting the adaptability of BO and GA across different domains.

4.2 DNN Architectures

To conduct a fair and meaningful evaluation, we select deep learning architectures that are commonly used for the chosen datasets. These architectures are optimized using BO and GA to determine the best hyperparameter configurations.

- Convolutional Neural Network (CNN): CNNs are used for image classification tasks on the MNIST and CIFAR-10 datasets. CNNs contain convolutional layers that extract spatial features from images, followed by pooling layers to reduce dimensionality, and fully connected layers for classification. The choice of number of convolutional layers, kernel sizes, and activation functions significantly impacts model performance, making it an ideal case for hyperparameter optimization.
- Recurrent Neural Network (RNN): RNNs are employed for sentiment analysis using the IMDB Reviews dataset. Since text sequences require models capable of handling temporal dependencies, Long Short-Term Memory (LSTM) units or Gated Recurrent Units (GRUs) are often used to improve long-range memory retention. The number of recurrent layers, hidden units per layer, and learning rates play a crucial role in determining the model's effectiveness, making it a suitable candidate for HPO.

By selecting CNNs for image-related tasks and RNNs for text-based sentiment analysis, we ensure that the experimental results reflect the effectiveness of BO and GA across a diverse range of neural network architectures.

4.3 Hyperparameter Search Space

The hyperparameter search space defines the range of values over which BO and GA explore to optimize DNN performance. This space includes both continuous and discrete hyperparameters that affect model training and generalization. The key hyperparameters considered in our experiments are:

- Learning Rate: A continuous variable ranging from 10^{-5} to 1. The learning rate determines the step size in weight updates during training. Finding an optimal learning rate is crucial, as a very high value may lead to unstable convergence, while a very low value may cause slow learning.
- Weight Decay: Another continuous variable ranging from 10^{-5} to 1. Weight decay (L2 regularization) helps prevent overfitting by adding a penalty to large weights. Proper tuning of weight decay balances model complexity and generalization.
- Number of Layers: A discrete variable selected from {2,3,4,5}. The depth of a network affects its ability to learn complex patterns. However, excessive layers may lead to vanishing gradients or overfitting, making this an essential parameter for optimization.
- Number of Neurons per Layer: A discrete variable chosen from {32,64,128,256}. The number of neurons controls the model's representational power. Too few neurons may result in underfitting, while too many neurons increase computational cost and risk overfitting.
- Activation Function: A categorical variable selected from ReLU, Sigmoid, and Tanh. Activation functions impact gradient flow and learning capacity. ReLU is widely used due to its efficiency in training deep networks, but Sigmoid and Tanh may be beneficial in certain cases.

This search space ensures that BO and GA explore a diverse range of configurations to find the best-performing hyperparameters.

4.4 Evaluation Metrics

To compare the effectiveness of BO and GA in optimizing DNN hyperparameters, we use the following evaluation metrics:

- Validation Accuracy: The primary measure of model performance is the validation accuracy, which represents the percentage of correctly classified samples on the validation set. A higher validation accuracy indicates a better hyperparameter configuration.
- Convergence Speed: The number of iterations required to reach a predefined performance threshold. Faster convergence means that the optimization technique can find high-performing hyperparameters efficiently, reducing computational time and costs.

- Computational Cost: The total computational resources (e.g., GPU hours) required to complete the HPO process. Since training DNNs is computationally expensive, an optimization technique that finds optimal hyperparameters using fewer evaluations is preferable.

5. Results and Discussion

The results of our experiments provide a comparative analysis of Bayesian Optimization (BO) and Genetic Algorithm (GA) in optimizing hyperparameters for deep neural networks across three benchmark datasets: MNIST, CIFAR-10, and IMDB Reviews. We evaluate the optimization methods based on three key metrics: validation accuracy, convergence speed, and computational cost. Our findings highlight the trade-offs between different optimization strategies and their effectiveness in deep learning applications.

5.1 Performance on MNIST

The MNIST dataset, consisting of handwritten digits, is a relatively simple image classification task. The performance of different hyperparameter optimization methods is analyzed in terms of validation accuracy, convergence speed, and computational cost.

5.1.1 Validation Accuracy

The results indicate that Bayesian Optimization achieves the highest best validation accuracy (99.3%) and the highest average validation accuracy (99.0%) with the lowest standard deviation (0.1%). This suggests that BO provides consistent and reliable hyperparameter configurations for training CNNs on MNIST. The Genetic Algorithm performs slightly worse, achieving a best accuracy of 99.1% and an average accuracy of 98.7%. Manual tuning and random search exhibit lower accuracy and higher variability, indicating that heuristic-based or unguided search approaches may not be as efficient in fine-tuning hyperparameters.

Table 1. Validation Accuracy on MNIST

Method	Best Validation Accuracy	Average Validation Accuracy	Standard Deviation
Manual Tuning	99.2%	98.8%	0.2%
Random Search	98.9%	98.5%	0.3%
Bayesian Optimization	99.3%	99.0%	0.1%
Genetic Algorithm	99.1%	98.7%	0.2%

5.1.2 Convergence Speed

Convergence speed is crucial in hyperparameter optimization since it determines how quickly an optimal model is reached. Bayesian Optimization outperforms all other methods, requiring only 30 iterations to reach 99% validation accuracy. The Genetic Algorithm, although slower than BO, reaches the same threshold in 40 iterations, making it more efficient than both manual tuning (50 iterations) and random search (100 iterations). The results demonstrate that BO accelerates model convergence by intelligently selecting promising hyperparameter configurations, while GA offers a competitive alternative.

Table 2. Convergence Speed on MNIST

Method	Number of Iterations to Reach 99% Accuracy
Manual Tuning	50
Random Search	100
Bayesian Optimization	30
Genetic Algorithm	40

5.1.3 Computational Cost

The GPU hours required for hyperparameter optimization indicate the efficiency of each method in utilizing computational resources. Bayesian Optimization proves to be the most efficient, requiring only 75 GPU hours, compared to 100 GPU hours for Genetic Algorithm and manual tuning, and 200 GPU hours for random search. The reduced computational cost of BO results from its ability to make informed decisions about which hyperparameters to evaluate next, avoiding unnecessary computations. GA, although computationally heavier than BO, still performs significantly better than random search.

Table 3. Computational Cost on MNIST

Method	GPU Hours Required
Manual Tuning	100
Random Search	200
Bayesian Optimization	75
Genetic Algorithm	100

5.2 Performance on CIFAR-10

The CIFAR-10 dataset presents a more complex classification challenge due to its color images and diverse object classes. This dataset provides insight into how hyperparameter optimization methods perform on more demanding deep learning tasks.

5.2.1 Validation Accuracy

On CIFAR-10, Bayesian Optimization again achieves the highest validation accuracy, with a best accuracy of 83.0% and an average accuracy of 82.0%, maintaining a low standard deviation of 0.4%. The Genetic Algorithm follows closely behind, reaching a best accuracy of 82.0% and an average accuracy of 81.5%. Manual tuning and random search yield slightly lower performance, reinforcing the idea that guided search methods such as BO and GA are more effective at tuning hyperparameters for complex datasets.

Table 4. Validation Accuracy on CIFAR-10

Method	Best Validation Accuracy	Average Validation Accuracy	Standard Deviation
Manual Tuning	82.5%	81.0%	0.5%
Random Search	81.2%	80.0%	0.6%
Bayesian Optimization	83.0%	82.0%	0.4%
Genetic Algorithm	82.0%	81.5%	0.5%

5.2.2 Convergence Speed

Bayesian Optimization requires 70 iterations to reach 82% accuracy, making it the most efficient approach. The Genetic Algorithm, while requiring 90 iterations, still outperforms both manual tuning (100 iterations) and random search (200 iterations). The faster convergence of BO and GA highlights their ability to exploit promising hyperparameter configurations more effectively than traditional methods.

Table 5. Convergence Speed on CIFAR-10

Method	Number of Iterations to Reach 82% Accuracy
Manual Tuning	100
Random Search	200
Bayesian Optimization	70
Genetic Algorithm	90

5.2.3 Computational Cost

The computational cost results are consistent with previous findings: Bayesian Optimization is the most efficient, requiring 150 GPU hours, while GA requires 200 GPU hours. Manual tuning has the same computational requirement as GA, while random search is the most expensive method at 400 GPU hours. The significant reduction in computational cost achieved by BO and GA makes them particularly valuable for optimizing complex DNN architectures like those used in CIFAR-10 classification.

Table 6. Computational Cost on CIFAR-10

Method	GPU Hours Required
Manual Tuning	200
Random Search	400
Bayesian Optimization	150
Genetic Algorithm	200

5.3 Performance on IMDB Reviews

The IMDB Reviews dataset presents a text classification challenge, requiring Recurrent Neural Networks (RNNs) or similar architectures for sentiment analysis. The results demonstrate the applicability of Bayesian Optimization and Genetic Algorithms beyond image-based tasks.

5.3.1 Validation Accuracy

For the IMDB Reviews dataset, Bayesian Optimization again achieves the best validation accuracy, reaching 85.5% at its peak and an average accuracy of 84.5%. The Genetic Algorithm follows with a best accuracy of 84.5% and an average accuracy of 83.5%. Manual tuning and random search yield slightly lower performance, with random search showing the highest standard deviation (0.5%), indicating higher variability in performance due to its unguided nature.

Table 7. Validation Accuracy on IMDB Reviews

Method	Best Validation Accuracy	Average Validation Accuracy	Standard Deviation
Manual Tuning	85.0%	83.5%	0.4%
Random Search	84.0%	82.5%	0.5%
Bayesian Optimization	85.5%	84.5%	0.3%
Genetic Algorithm	84.5%	83.5%	0.4%

5.3.2 Convergence Speed

Bayesian Optimization demonstrates the fastest convergence, requiring 50 iterations to reach 84% validation accuracy. The Genetic Algorithm requires 60 iterations, still outperforming both manual tuning (70 iterations) and random search (140 iterations). These results suggest that BO and GA are effective at identifying optimal hyperparameters for sequence-based models, such as RNNs.

Table 8. Convergence Speed on IMDB Reviews

Method	Number of Iterations to Reach 84% Accuracy
Manual Tuning	70
Random Search	140
Bayesian Optimization	50
Genetic Algorithm	60

5.3.3 Computational Cost

As in previous datasets, Bayesian Optimization proves to be the most computationally efficient method, requiring 100 GPU hours, compared to 120 GPU hours for Genetic Algorithm, 140 GPU hours for manual tuning, and 280 GPU hours for random search. This reinforces the conclusion that BO reduces computational overhead while still achieving superior results.

Table 9: Computational Cost on IMDB Reviews

Method	GPU Hours Required
Manual Tuning	140
Random Search	280
Bayesian Optimization	100
Genetic Algorithm	120

5.4 Discussion

The experimental results clearly illustrate the advantages of Bayesian Optimization (BO) and Genetic Algorithms (GA) in hyperparameter optimization (HPO) for deep neural networks. Both methods significantly enhance the performance of deep learning models compared to manual tuning and random search, demonstrating the benefits of systematic and intelligent search strategies. BO consistently outperforms GA in most scenarios, achieving higher validation accuracy, faster convergence, and lower computational costs. This superiority is primarily attributed to BO's ability to model the relationship between hyperparameters and model performance using a Gaussian Process (GP) and an acquisition function that effectively balances exploration and exploitation. However, despite BO's advantages, GA remains a viable and competitive alternative, particularly in situations where

the search space is large and complex. Since GA operates based on a population of potential solutions and applies selection, crossover, and mutation, it can explore diverse regions of the search space that may be overlooked by BO. This makes GA particularly useful when evaluating the fitness function is computationally expensive, as it can leverage evolutionary strategies to efficiently navigate high-dimensional spaces.

Another important factor in choosing between BO and GA is computational cost. The results indicate that BO is generally more computationally efficient than GA, as it requires fewer iterations and GPU hours to reach optimal performance. This efficiency makes BO particularly attractive for practical applications where computational resources are limited. However, the choice between BO and GA should also consider problem-specific characteristics, such as the nature of the search space, the cost of evaluating hyperparameters, and the availability of computational resources. In some cases, a hybrid approach that combines BO's efficient search capabilities with GA's diverse exploration mechanisms may provide an even more effective optimization strategy.

6. Conclusion

In this study, we investigated the application of Bayesian Optimization (BO) and Genetic Algorithms (GA) for automated hyperparameter optimization (HPO) in Deep Neural Networks (DNNs). We provided a detailed review of the theoretical foundations of both methods, discussed their implementation, and conducted an extensive experimental evaluation on benchmark datasets (MNIST, CIFAR-10, and IMDB Reviews). Our findings demonstrate that automated HPO methods significantly improve model performance compared to traditional approaches such as manual tuning and random search. The results indicate that BO consistently outperforms GA in most evaluation criteria, including validation accuracy, convergence speed, and computational cost. BO's ability to efficiently explore the hyperparameter space while reducing unnecessary evaluations makes it a highly effective optimization method. However, GA remains a strong alternative, particularly when dealing with large search spaces or high-dimensional hyperparameter configurations. While BO excels in fine-tuning hyperparameters efficiently, GA offers a robust evolutionary approach that can discover novel configurations that may not be easily identified by BO's surrogate model.

Looking ahead, future research could explore several promising directions to further enhance HPO techniques. One interesting avenue is the integration of BO and GA into a hybrid framework, leveraging the exploration strength of GA and the efficient exploitation of BO. Additionally, advancements in acquisition functions for BO could lead to even faster and more effective hyperparameter searches. For GA, the development of more efficient genetic operators such as adaptive mutation and dynamic crossover—could improve search performance while reducing computational overhead. Beyond hyperparameter optimization, the integration of automated HPO with other aspects of deep learning, such as neural architecture search (NAS) and data augmentation techniques, could further boost model performance and generalization. By continuously refining and combining optimization strategies, the field of AutoML (Automated Machine Learning) will continue to evolve, making deep learning more accessible, efficient, and powerful for real-world applications.

References

[1] Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems*, 24, 2546–2554.

[2] Cho, H., Kim, Y., Lee, E., Choi, D., Lee, Y., & Rhee, W. (2019). DEEP-BO for hyperparameter optimization of deep networks. *arXiv preprint* arXiv:1905.09680. https://arxiv.org/abs/1905.09680

[3] Frazier, P. I. (2018). A tutorial on Bayesian optimization. *arXiv preprint* arXiv:1807.02811. https://arxiv.org/abs/1807.02811

[4] Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization* (pp. 507–523). Springer.

[5] Jamieson, K., & Talwalkar, A. (2016). Non-stochastic best arm identification and hyperparameter optimization. In *Artificial Intelligence and Statistics* (pp. 240–248). PMLR.

[6] Kandasamy, K., Schneider, J., & Póczos, B. (2015). High dimensional Bayesian optimisation and bandits via additive models. In *International Conference on Machine Learning* (pp. 295–304). PMLR.

[7] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(1), 6765–6816.

[8] Li, C., Jiang, J., Zhao, Y., Li, R., Wang, E., Zhang, X., & Zhao, K. (2021). Genetic algorithm-based hyper-parameters optimization for transfer convolutional neural network. *arXiv preprint* arXiv:2103.03875. https://arxiv.org/abs/2103.03875

[9] Loshchilov, I., & Hutter, F. (2016). CMA-ES for hyperparameter optimization of deep neural networks. *arXiv preprint* arXiv:1604.07269. https://arxiv.org/abs/1604.07269

[10] Maclaurin, D., Duvenaud, D., & Adams, R. (2015). Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning* (pp. 2113–2122). PMLR.

[11] Mendoza, H., Klein, A., Feurer, M., Springenberg, J. T., & Hutter, F. (2016). Towards automatically-tuned neural networks. In *Automated Machine Learning* (pp. 141–156). Springer.

[12] Mockus, J. (2012). *Bayesian approach to global optimization: Theory and applications*. Springer Science & Business Media.
[13] Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems*, 25, 2951–2959.
[14] Wu, J., Toscano-Palmerin, S., Frazier, P. I., & Wilson, A. G. (2019). Practical multi-fidelity Bayesian optimization for hyperparameter tuning. *arXiv preprint* arXiv:1903.04703. https://arxiv.org/abs/1903.04703
[15] Yao, Z., Wang, M., Kwok, J. T., & Ni, L. M. (2020). Efficient neural architecture search via proximal iterations. In *AAAI Conference on Artificial Intelligence* (pp. 6665–6672).
[16] Young, T., Hazarika, D., Poria, S., & Cambria, E. (2018). Recent trends in deep learning based natural language processing. *IEEE Computational Intelligence Magazine*, 13(3), 55–75.
[17] Yu, T., & Zhu, H. (2020). Hyper-parameter optimization: A review of algorithms and applications. *arXiv preprint* arXiv:2003.05689. https://arxiv.org/abs/2003.05689
[18] Zhang, Z., & Zhang, J. (2019). Deep neural network hyperparameter optimization with orthogonal array tuning. *arXiv preprint* arXiv:1901.06824. https://arxiv.org/abs/1901.06824
[19] Zoph, B., & Le, Q. V. (2017). Neural architecture search with reinforcement learning. *arXiv preprint* arXiv:1611.01578. https://arxiv.org/abs/1611.01578
[20] Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 8697–8710).

Appendices

Algorithm Pseudocode

Bayesian Optimization Algorithm

```
def bayesian_optimization(search_space, num_iterations):
    # Initialize the Gaussian Process
    gp = GaussianProcess()

    # Initialize the observed data
    observed_data = []

    for i in range(num_iterations):
        # Optimize the acquisition function to find the next set of hyperparameters
        next_hyperparameters = optimize_acquisition_function(gp, observed_data, search_space)

        # Evaluate the model performance with the new hyperparameters
        validation_accuracy = evaluate_model(next_hyperparameters)

        # Add the new data point to the observed data
        observed_data.append((next_hyperparameters, validation_accuracy))

        # Retrain the Gaussian Process
        gp.train(observed_data)

        # Check stopping criterion
        if stopping_criterion_met(observed_data):
            break

    return observed_data
```

Genetic Algorithm

```
def genetic_algorithm(search_space, population_size, num_generations):
    # Initialize the population
    population = initialize_population(search_space, population_size)

    for generation in range(num_generations):
        # Evaluate the fitness of each individual in the population
        fitness_scores = evaluate_fitness(population)

        # Select individuals for the mating pool
        mating_pool = select_population(population, fitness_scores)
```

```
    # Apply crossover to generate offspring
    offspring = apply_crossover(mating_pool)

    # Apply mutation to introduce diversity
    offspring = apply_mutation(offspring)

    # Replace the current population with the offspring
    population = offspring

    # Check stopping criterion
    if stopping_criterion_met(population, fitness_scores):
        break

return population, fitness_scores
```