Exploration and Comparative Analysis of Hyperparameter Optimization(HPO) Algorithms

Yingtao Zhang

This experiment focuses on the exploration of hyperparameter optimization (HPO) algorithms, covering the definition of search space and other HPO-related concepts. Implements and compares the performance of multiple HPO representative algorithms on MNIST, including Grid Search, Random Search, Genetic Algorithm, Bayesian Optimization, and Successive Halving. The experiment summarizes the advantages and limitations of each algorithm by evaluating the performance of each algorithm in terms of trend, efficiency, accuracy, and stability. The results show that Grid Search can completely traverse the search space, but the resource consumption is huge; Random Search is simple to implement and suitable for resource-limited situations; Genetic Algorithm combines crossover and mutation mechanisms to help avoid local minimum; Bayesian Optimization obtains the highest validation accuracy through sampling and objective function fitting in continuous space; Successive Halving achieves high resource utilization by stage elimination, which is suitable for fixed resource situation. In addition, this experiment points out the impact of the hyperparameters of the HPO algorithm itself on the performance of the algorithm, emphasizing that hyperparameter optimization still requires a certain knowledge foundation. Overall, this study verifies the value of various HPO algorithms.

1. INTRODUCTION

Deep learning is a machine learning method based on the structure and algorithm of artificial neural networks. The artificial neural network is a simulation of the human brain. Computers can learn from data and make predictions or decisions using artificial neural networks. A good deep learning model has properties of stability, fast convergence, high accuracy, and good generalization. A bad deep learning model has properties of slow convergence, overfitting, or underfitting. Learners and researchers using deep learning usually optimize deep learning models by adjusting hyperparameters. Hyperparameters refer to those parameters that are not updated by gradients during model training (Yu & Zhu, 2020) [1].

Finding the optimal hyperparameters for a model is a task that requires a lot of time and computing resources and is very difficult for beginners. Therefore, automated machine learning(Hutter et al., 2019)[4] came into being. Hyperparameter optimization (HPO) is a core technology of AutoML, and it is very important. HPO algorithms can help deep learning researchers and learners who do not have enough experience to quickly deploy models, which can save a lot of time and computing resources.[4]

This paper is an exploration of HPO algorithms, with the goal of understanding HPO techniques. Before discussing exploration and experiments, we should first clarify what the problem is and what is the representation of this problem. This paper explores HPO techniques, so the questions here are: How do various HPO algorithms work? How do HPO algorithms find the optimal solution? What is the performance of the optimal model found by the HPO algorithm? The explanation of the HPO problem's representation is more detailed in the next section.

2. SEARCH SPACE

The search space can be defined as the set of all possible parameter combinations in the search and optimization problem, subject to constraints, variable ranges, or prior knowledge (Boyd & Vandenberghe, 2004; Bergstra & Bengio, 2012). [2] [3] The search space constitutes the representation of HPO and represents all the parameter combinations that we can use during the optimization. The search space is the core component of HPO. A properly defined search space can avoid wasting computational resources. In algorithms like Grid Search, a large search space will lead to the curse of dimensionality. Therefore, how to save computational resources is the main problem that HPO algorithms want to solve. [3]

2.1. Components of the search space

As mentioned in the introduction of this section, the search space represents the combinations of parame-

ters that we can use in the optimization. Hyperparameters can be divided into four categories based on the properties of hyperparameters: model hyperparameters, training hyperparameters, regularization hyperparameters, and data-related hyperparameters (Jatender Kumar, 2024) [5].

In this section, I'll give a quick overview of the categories of the hyperparameters. The selection of hyperparameters in this experiment mainly covers three aspects: model hyperparameters, training hyperparameters, and regularization hyperparameters. Data-dependent hyperparameters are not selected here because this paper uses a relatively simple dataset, and no data augmentation-related techniques are used.

2.1.1. Model hyperparameters

Model hyperparameters are the hyperparameters related to the model structure, which usually appear in the constructor of the model and directly determine the type, complexity and characteristics of the model.[5] The model hyperparameters include the type of network layer, the number of network layers, the number of neurons in each layer, the size of the convolution kernel and other settings, such as whether to use the activation function and the type of activation function.[5]

Hyperparameter	Description
Layer type	Defines layer types
Number of layers	Controls model depth
Units per layer	Adjusts layer width
Activation function	Adds non-linearity

TABLE 1: Model Hyperparameters Overview[5].

2.1.2. Training hyperparameters

Training hyperparameters are the hyperparameters related to the model training behavior, which are usually set in the training function of the model, and directly determine the convergence speed and stability of the model.[5] The training hyperparameters include learning rate, batch size, and number of epochs...[5] The learning rate determines the step size of the model weight update. The batch size determines the amount of data in each iteration. The learning rate and batch size together determine the training convergence efficiency of the model.[5].The number of epochs indicates how many times the entire training set is traversed during the training process, directly affecting the convergence degree and generalization performance of the model.[5]

Hyperparameter	Description
Learning rate	Step size for updates
Batch size	Data per iteration
Number of epochs	Training passes

TABLE 2: Training Hyperparameters Overview[5].

2.1.3. Regularization hyperparameters

Regularization hyperparameters are used to prevent the model from overfitting, depending on the regularization technique, which is often present in the model architecture or optimizer settings, and directly determines the generalization ability of the model.[5] Regularization hyperparameters include Dropout rate and weight decay (L2 regularization). [5]Dropout reduces the risk of overfitting by randomly ignoring some neurons. Weight decay reduces the complexity of the model by suppressing over-weighted neurons.[5]

Hyperparameter	Description
Dropout rate	Prevents overfitting
L2 regularization	Penalizes large weights

TABLE 3: Regularization Hyperparameters Overview[5]

2.1.4. Data-related hyperparameters

Data-related hyperparameters are the hyperparameters related to the input data preprocessing of the model, which usually appear in the data processing stage. Data-related hyperparameters will impact the generalization performance and data adaptability of the model and usually increase the training time.[5] Data-related hyperparameters include data augmentation methods (e.g., transformations, noise injection, erasing), normalization, standardization, etc.[5] Data augmentation can improve the model's robustness by increasing the training data's diversity. [5]

Hyperparameter	Description
Data augmentation	Enhances data diversity
Normalization	Scales data
Standardization	Centers data

TABLE 4: Data-related Hyperparameters Overview[5].

2.2. Feasible Region of the Search Space

In this experiment, I did not include data-related hyperparameters in the search space. Mainly because of the following two reasons. First of all, the dataset that I used in this experiment is MNIST. This is a handwritten digit image data set containing the numbers 0-9. This is a relatively simple image dataset. In general,

a good model performance can be obtained using this dataset without additional data augmentation methods. Adding data-related hyperparameter data augmentation methods has a limited effect on improving the model.

More importantly, data-dependent hyperparameters are not selected to avoid the disaster of dimensionality caused by searching too large a space. To give a simple example, suppose I choose 10 hyperparameters to define the dimensionality of the search space, with each hyperparameter having 10 discrete values. Then, this search space can produce 10^{10} combinations. If each model can be trained in only 10 seconds, then it will take 10^{11} seconds, or 100 billion seconds, to complete a grid search. In other words, it will only take 3170 years to complete such a grid search in the search space. Therefore, in this experiment, we only focus on hyperparameters in modelling, training, and regularization to balance computational resources with experimental results.

2.2.1. Model hyperparameters selection

In the selection of the number of model layers, I referred to the experimental results of LeCun et al. in the LeNet-5 network. LeNet-5 achieved good performance on the MNIST dataset with a simple architecture, which shows that fewer layers are sufficient to achieve good results on MNIST. [6] Therefore, in this experiment, we only use fully connected layers and limit the number of layers between 1 and 4.

In the selection of the number of units per layer, I referred to the model experiment results in the deep learning course materials. The experimental results show that using 32 or 64 units as the number of units per layer can achieve very good results on the MNIST dataset. Therefore, the number of neurons per layer in this experiment is set in the range of 2 to 128. Setting a lower boundary (2) helps to establish a simple model structure in order to evaluate the performance of the model with a lower complexity. The purpose of such a setting is to cover model structures from simple to higher complexity in order to observe the performance of models of different complexity. In the selection of activation function, I used the three most commonly used activation functions: ReLU, tanh and sigmoid.

Parameter	Values/Range	Type
Number of Layers	1, 2, 3, 4	Discrete
Units per Layer	2 to 128	Cont./Disc
Activation	ReLU, Tanh, Sigmoid	Discrete

TABLE 5: Model Hyperparameters

2.2.2. Training hyperparameters selection

In the selection of the learning rate, I still refer to the course materials of deep learning. Overall, the reasonable learning rate interval is generally between 0.001 and 0.1. In this experiment, I set the learning rate range to 0.0001 to 0.3, including not only the intervals commonly used for stable convergence (such as 0.001 to 0.1) but also smaller and larger values. The reason for this setting is to cover models with different performances. A smaller learning rate helps to observe the performance of models that converge slowly, while a larger learning rate may cause gradient explosion, which may prevent some models from converging. This range setting ensures that the experiment covers good models and bad models.

In the selection of batch size, I still refer to the course materials of deep learning. In this experiment, the batch size range is set between 16 and 128. Smaller batch sizes (such as 16 and 32) can provide stable and frequent weight updates. Larger batch sizes (such as 128 and 256) can improve computational efficiency. This setting ensures that the experiment covers different convergencies.

Parameter	Values/Range	Type
Learning Rate	0.0001 to 0.3 (log scale)	Cont./Disc
Batch Size	16, 32, 64, 128, 256	Discrete

TABLE 6: Training Hyperparameters

2.2.3. Regularization hyperparameters selection

In the selection of regularization hyperparameters, the range of dropout rate and L2 regularization strength is set. For the dropout rate, I chose a continuous range of 0.05 to 0.6 to cover different levels from low to high. The L2 regularization strength range is set between 10^{-7} and 10^{-2}, and using a logarithmic scale to finely control the regularization effect.

Parameter	Values/Range	Type
Dropout Rate	0.05 to 0.6	Cont./Disc
L2 Strength	1e-7 to 1e-2 (log scale)	Cont./Disc

TABLE 7: Regularization Hyperparameters

3. EXPERIMENTAL DESIGN

3.1. Dataset Description

This experiment uses a slice of the MNIST dataset, which contains 3000 training images and 500 testing images. Each image is a grayscale image composed of 28x28 pixels, representing the numbers 0-9. This experiment did not use any data preprocessing technology, and the performance of each HPO algorithm is evaluated on the original images.

3.2. Experimental steps

3.2.1. Search space construction

First, according to the description of the search space in Section 2, define the search space of all

hyperparameters. All hyperparameter combinations used by HPO algorithms are extracted from this search space. Different types of hyperparameter values can be generated according to different HPO algorithms. For example, discrete hyperparameters are set for Grid Search. Some hyperparameters set for random search are randomly extracted from continuous range.

3.2.2. Modelling and training function
I created model-building and model-training functions. For any hyperparameter combination generated from the search space, the model builder and model trainer can extract the required hyperparameters and automatically build and train the model. All training is based on the dataset defined in Section 3.1. The training epochs are fixed at 10. Training conditions are consistent for all HPO algorithms.

3.2.3. HPO algorithms setting
For each HPO algorithm in the experiment, a unique function is created. Each algorithm trains up to 1,000 models, records the model metrics of each training, including training and test loss, training and test accuracy, and training time, and stores them in Google Drive in real-time.

3.2.4. Special Settings for Grid Search
In the Grid Search algorithm, I set 15,360 hyperparameter combinations for model creation to ensure that the value range of each hyperparameter covers more of the search space, including its upper and lower bounds. If 1,000 combinations are used, only 2-3 values can be assigned for each hyperparameter, which will make search space to be too sparse.

Hyperparameter	**Values**
Number of Layers	1, 2, 3, 4
Units per Layer	2, 44, 86, 128
Activation Function	relu, tanh, sigmoid
Learning Rate	$1e-4$, $1.4e-3$, $2e-2$, $3e-1$
Batch Size	16, 32, 64, 128, 256
Dropout Rate	0.05, 0.23, 0.42, 0.60
L2 Strength	$1e-7$, $4.6e-6$, $2.2e-4$, $1e-2$

TABLE 8: Discrete Values for Each Hyperparameter in Grid Search

3.3. Evaluation Metrics

To comprehensively evaluate the performance of each HPO algorithm, this experiment uses four metrics.

3.3.1. Speed of finding the optimal model
This metric is used to evaluate the efficiency of each algorithm in finding the optimal model within the same computing resources. Comparing the number of searches of each HPO algorithm can reflect the efficiency of the algorithm.

3.3.2. Performance of the optimal model
The optimal model found by each HPO algorithm in the experiment is different. Comparing the best model performance of each method can measure the difference in the quality of their optimal model.

3.3.3. Optimization Path of HPO Algorithms
Comparing the accuracy trends of different model combinations generated by the HPO algorithm during the hyperparameter search process, can reveal the different approaches to finding the optimal solution.

3.3.4. Stability
By observing whether the HPO algorithm can maintain a consistent level of result in multiple runs, to guarantee the effectiveness and robustness of the HPO algorithm.

4. COMPARATIVE ANALYSIS OF HPO ALGORITHMS

In this section, I will first introduce grid search and random search. The experiment uses these two basic algorithms as baseline algorithms to evaluate other HPO algorithms.

HPO algorithms can be divided into three categories: methods based on evolutionary theory, methods based on Bayesian optimization, and methods based on incremental search.[4] In the experiment, I will introduce, evaluate and analyze one representative algorithm from each of these three categories.

4.1. Baseline Algorithms: Grid Search and Random Search

4.1.1. Grid Search
Grid search is the most basic method in hyperparameter optimization.[1] It searches for the optimal solution by traversing all permutations and combinations of hyperparameters in the search space. If the computing resources are sufficient, theoretically, grid search can ensure that the global optimal hyperparameter combination is found.[1] However, this process is often limited by the cost of time and computing resources, because the hyperparameter space is usually very large.[1]

In addition, the types of hyperparameters can be discrete or continuous. The example provided in Section 2.2 on the feasible domain of the search space demonstrates that, even in a discrete hyperparameter space, a limited number of hyperparameters with only a few possible values for each can still produce a huge number of combinations, and the computing resources consumed by traversing all the resulting permutations and combinations are unaffordable.

Although grid search is simple and can theoretically find the optimal solution, it cannot be used in high-dimensional hyperparameter spaces.

4.1.2. Result of Grid Search

In the Grid Search of this experiment, a total of 15,360 hyperparameter combinations were tried. Figure 1 shows the overall distribution of these combinations, more than half of the models have a test accuracy of over 80%. This shows that the search space is properly defined, covering good and bad hyperparamers combinations. providing a comprehensive performance distribution.

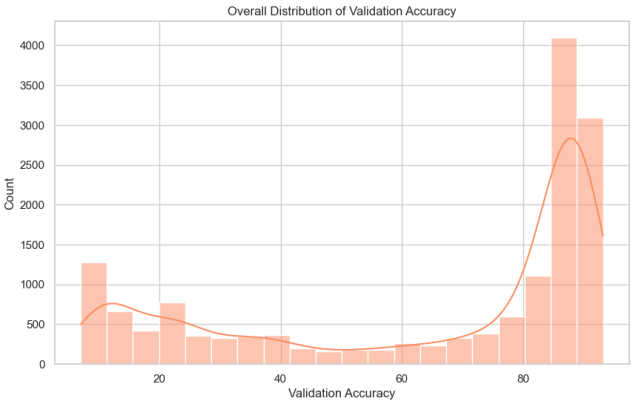

FIGURE 1: Overall Accuracy Distribution of Grid Search

Figure 2 shows an increase in units per layer generally leads to higher accuracy. However, the difference in the effect of 86 and 128 units is not obvious, indicating that when units per layer reach a certain number, the effect of continuing to increase the number of units on performance gradually weakens, and even reaches a bottleneck.

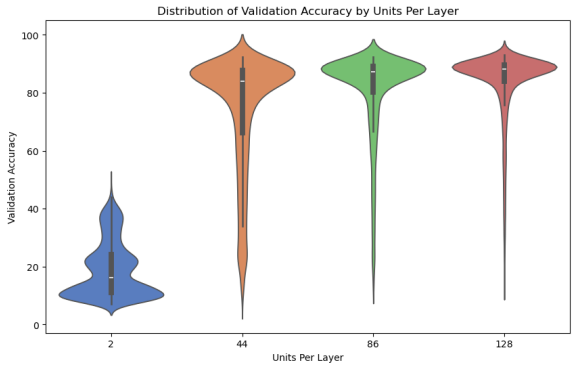

FIGURE 2: Accuracy Distribution by Units Per Layer

Figure 3 shows the effect of number of layers on validation accuracy, and the results show that models with lower layers have higher accuracy.

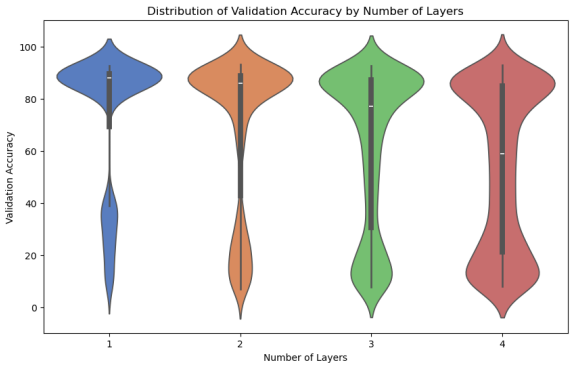

FIGURE 3: Accuracy Distribution by Number of Layers

Further, the heat maps in Figures 4 and 5 show the distribution of hyperparameter combinations on validation accuracy. The model combination with 1 layer and 128 units contains more high accuracy (¿90%) models, and achieves the highest average accuracy in the test.

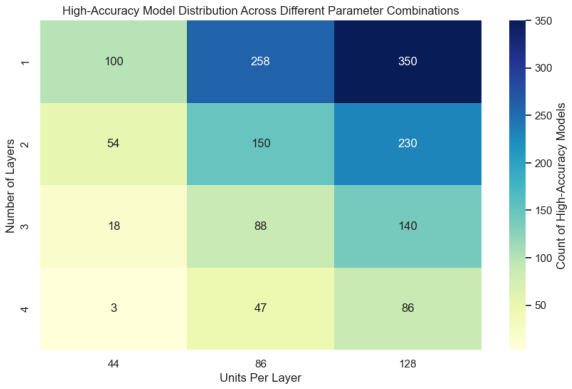

FIGURE 4: Parameter Combination Heatmap for High-Accuracy Model

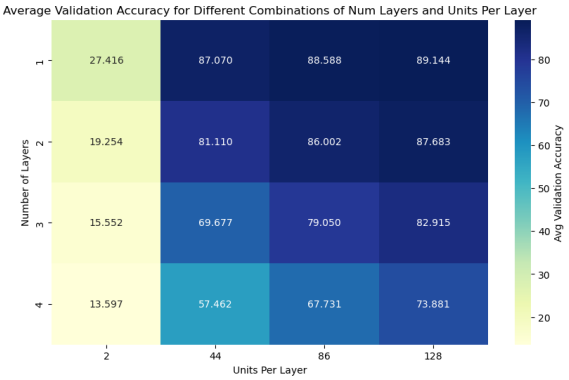

FIGURE 5: Parameter Combination Heatmap for All Models

This does not mean that increasing the number of layers will lead to worse training results. Models with higher numbers of layers usually require longer training

time and more data to capture complex features. However, since the number of training epochs is fixed to 10 in this experiment, this may limit the performance of deep models. Therefore, it is reasonable to speculate that the performance of deep models may be improved if more epochs or larger amounts of data are provided. The results in Figure 6 further support this view. In general, we can conclude that models with fewer layers are simpler and faster to train.

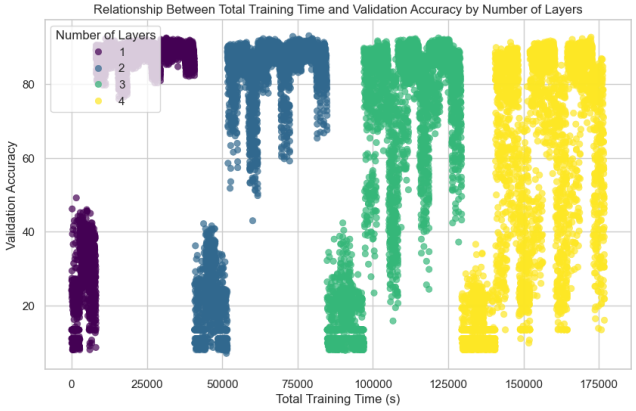

FIGURE 6: Validation Accuracy Distribution by Model Layer Count

4.1.3. Random Search

Random search[3] is an intuitive HPO algorithm. Unlike grid search, random search does not traverse all possible hyperparameter combinations, but randomly extracts values from the value space of each hyperparameter to form different hyperparameter combinations.[3] The number of random searches can be predefined, and each sampling will produce a new hyperparameter combination.[3] Although random search may seem imprecise, it is often more efficient than grid search in high-dimensional space.[1] Random search can usually find relatively good performance combinations within a certain number of attempts,[3] which is a nice choice in limited resources situations.

4.1.4. Result of Random Search

From Figure 7, we can tell that Random Search performs well in these 1000 samples, with more than half of the models achieving an accuracy of more than 80%. Among them, the number of models with an accuracy of more than 90% is greater than the number of models with an accuracy between 80% and 90%.

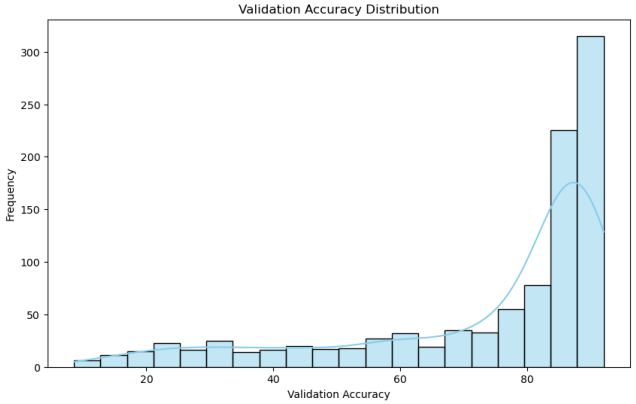

FIGURE 7: Overall Accuracy Distribution of Random Search

This result shows that Random Search can quickly explore good solutions in large-scale hyperparameter combinations, which is different from the performance of Grid Search. Grid Search is highly possible to skip global optimal due to fixed-interval sampling in the discrete value space, thereby missing some potential high-performance combinations.[3] This phenomenon is also consistent with the layout mentioned in the study by Bergstra and Bengio (2012).

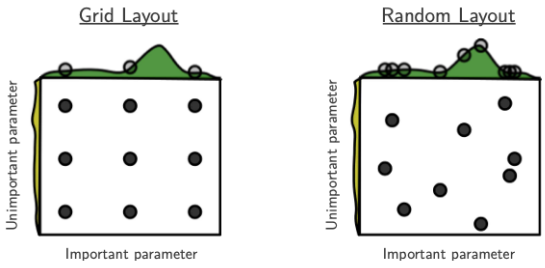

FIGURE 8: Grid & Random layout(Bergstra & Bengio 2012

In Figure 9, a similar phenomenon to that of Grid Search also appears in the results of Random Search: the effect is better when the model has fewer layers.

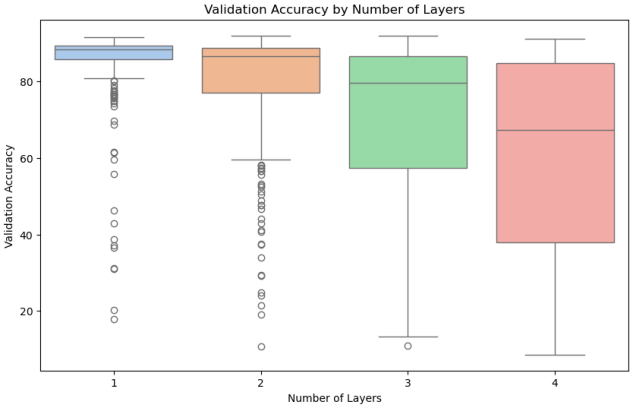

FIGURE 9: Accuracy Distribution by Number of Layers

From Figure 10, it can be seen that the performance of the tanh activation function is significantly better than the other two activation functions.

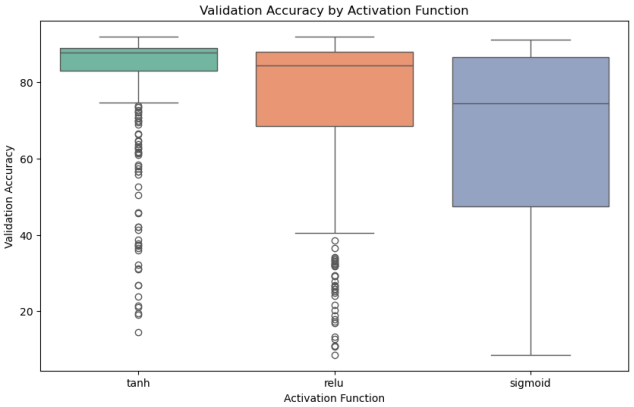

FIGURE 10: Accuracy Distribution by Activation Function

4.2. Method Based on Evolutionary Theory: Genetic algorithm

4.2.1. Genetic algorithm

Genetic algorithm is an optimization algorithm based on evolutionary theory and Darwin's theory of natural selection, which is inspired by human reproduction and inheritance mechanism (Massachusetts Institute of Technology, 2021). [8] In the human reproductive process, germ cells pass on the chromosome fragment combination of the parent to the next generation through meiosis.[8] During this process, chromosomes crossover, recombination and mutate, thereby generating genetic diversity. This mutation may bring positive or negative effects, individuals with stronger adaptability are more likely to be preserved in evolution. The genetic algorithm simulates the reproduction and evolution mechanism and gradually approaches the optimal solution using four core steps: evaluation, selection, crossover mutation, and iteration. [8]

- Evaluation: First, set an initial population containing different hyperparameter combinations. Perform fitness evaluation on each hyperparameter combination in the population and calculate the model's performance. Models with higher performance mean that their hyperparameter combinations are better. This step provides a clue for the selection operations.
- Selection: According to the evaluation results, the hyperparameter combinations with poor adaptability are eliminated, and only the hyperparameter combinations with high adaptability are retained to enter the next stage. This step ensures the overall quality of the next population.
- Crossover and mutation: New hyperparameter combinations are generated through crossover and mutation operations in the retained hyperparameter combinations. The crossover operation randomly reorganizes the values of the retained hyperparameter combinations. The mutation operation randomly adjusts the values of some hyperparameters (randomly re-extracted from the search space). The crossover and mutation operations retain the excellent hyperparameters of the previous generation, and increase the genetic diversity of the population. The new hyperparameter combinations generated by the crossover and mutation operation are used as the initial population for the next iteration.
- Iteration: The genetic algorithm repeats the process of evaluation, elimination, crossover and mutation through multiple rounds of iterations. The overall adaptability of the population is improved from generation to generation and gradually approaches the optimal solution.

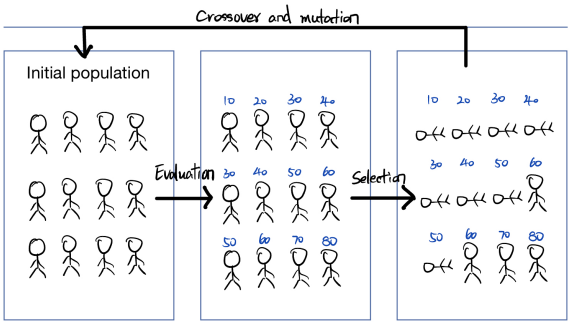

FIGURE 11: Process of Genetic algorithm

4.2.2. Result of Genetic algorithm

In this experiment, there are 20 generations, and each generation contains 50 models, so a total of 1,000 models are trained. From the distribution in Figure 12, we can see that the models in the second generation have achieved a low validation loss, and the loss performance is very stable in subsequent generations. This shows that Genetic Search can

converge quickly and effectively find high-quality hyperparameter combinations. In five independent tests, Genetic Search showed similar convergence speed and stability.

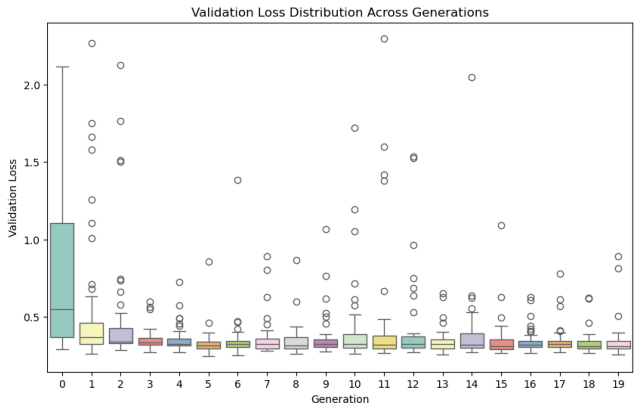

FIGURE 12: Validation Loss Distribution by generation

4.3. Method Based on Bayesian Optimization

4.3.1. Bayesian Optimization

In this section, I will give a brief introduction to the Bayesian optimization algorithm. Bayesian optimization and gradient descent have similarities in their target: both hope to find the optimal solution for a loss function. The difference is that gradient descent relies on the differentiability of the loss function to calculate the gradient, while Bayesian optimization does not require the gradient information. Bayesian optimization uses a surrogate model and sampling function(acquisition function) to explore the search space.[1] Specifically,The process of Bayesian optimization can be divided into the following four steps:

- Initialization: Randomly sample several points in the search space as a sample set, and calculate the values of these points.[1]
- Surrogate model: The surrogate model approximates the true objective function by using the sample set and the values of the sampling points. The surrogate model needs to provide the approximated objective function and estimate the uncertainty of the approximated objective function.[1] (Because the behaviour of the true objective function between two adjacent sampling points is uncertain). Gaussian process(Rasmussen & Williams 2006)[11] is one of the commonly used surrogate models in Bayesian optimization (Feurer, Springenberg & Hutter 2015),[10] which can provide the mean and confidence interval (upper and lower confidence) of the objective function in the unsampled area.[1] Also, the choice of surrogate model is not unique.
- Sampling function: The goal of the sampling function is to select the area with the potential optimal solution[1] and determine the next sampling point based on the information provided by the surrogate model.[1] Common sampling functions include Probability of Improvement (PI), Expected Improvement (EI) and Upper Confidence Bound (UCB)(Snoek, Larochelle & Adams 2012).[12] Different sampling function determines the next sampling point by different logic.
- Iteration: Adding the new sampling point to the sampling set, and repeat the steps from 2 to 4.

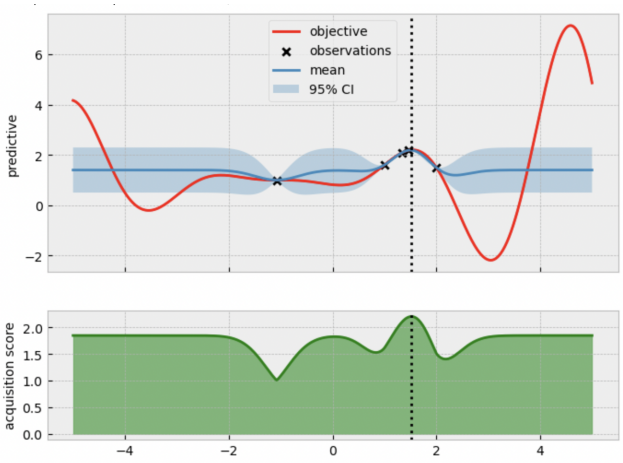

FIGURE 13: Bayesian Optimization(Nguyen, Q. 2024)[9]

4.3.2. Result of Bayesian Optimization

In this experiment, I used GaussianProcessRegressor in sklearn as the surrogate model and Expected Improvement as the sampling function. The initial sampling points were 10. According to Figure 14, we can see that in Bayesian optimization, the accuracy of most models is between 80-100, and only a few are in the remaining area. This proves the effectiveness of Bayesian optimization, and also proves that Bayesian optimization can quickly lock the possible range of the optimal solution by fitting the objective function and explore within this range.

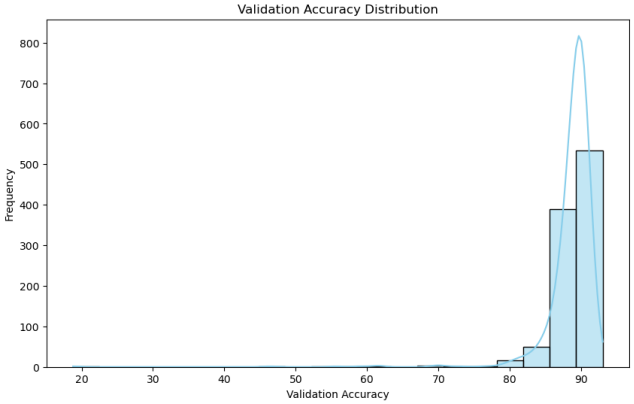

FIGURE 14: Overall Accuracy Distribution of Bayesian Optimization

From Figure 15, we can see that more than two-thirds of the high-accuracy models still have only one layer, which once again confirms my previous guess.

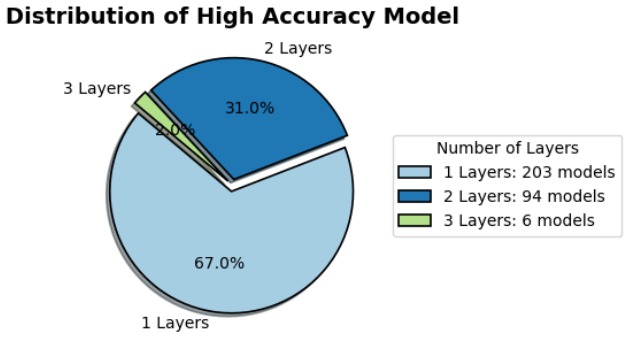

FIGURE 15: Overall Accuracy Distribution of High Accuracy Model

4.4. Methods Based on Incremental Search

4.4.1. Successive Halving

Successive Halving is the basis of incremental search algorithms. Its core idea is to allocate limited resources to those hyperparameters that perform well. The implementation method of Successive Halving is also relatively simple, mainly divided into four parts:

- Resource setting: Set a fixed amount of total resources, which can be training time, or number of epochs.
- Initial resource allocation: Define the number of models in the initial stage and the amount of resources allocated to each model in the initial stage. Usually, the initial stage will contain more models, and each model will be allocated fewer resources in order to quickly evaluate its initial performance.
- Elimination:Train all initial models and sort them based on their performance. Keep the model parameters with better performance and eliminate half of the ones with worse performance.
- Resource reallocation:Continue to train the models retained in the previous round and allocate more resources to them for more accurate evaluation. Repeat steps 3 and 4, eliminate poorly performing models round by round, and focus resources on the possible optimal hyperparameter combination until all resoures are used.

In Successive Halving, half of the models are eliminated and resource allocation is doubled in each round.[1] Successive Halving is very suitable for finding the optimal hyperparameter combination when resources are limited, avoiding waste of resources.

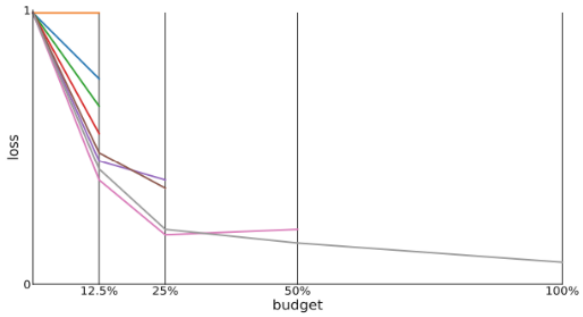

FIGURE 16: Budget allocation with successive halving (Bissuel, 2018)

4.4.2. Result of Successive Halving

In this experiment, I allocated a fixed total resource of 10,000 epochs, aligning with the total training epochs used by other hyperparameter optimization (HPO) algorithms for comparison. Rather than strictly following the standard Successive Halving approach, I implemented a tailored distribution across six stages to maintain comparable training resources with other HPO algorithms. The stages are structured as follows:

- Stage 1: 1,500 models, each with 2 epochs
- Stage 2: 750 models, each with 4 epochs
- Stage 3: 375 models, each with 6 epochs
- Stage 4: 187 models, each with 8 epochs
- Stage 5: 20 models, each with 10 epochs
- Stage 6: 2 models, each with 27 epochs

This setup used the 10,000-epoch budget precisely, enabling a fair comparison with other algorithms by using similar training resources. Additionally, a final stage with 27 epochs was included to retain the unique characteristics of resourses allocate algorithms. From Figure 17, we can see that the accuracy is increasing at different stages, which also proves the effectiveness of the SF algorithm.

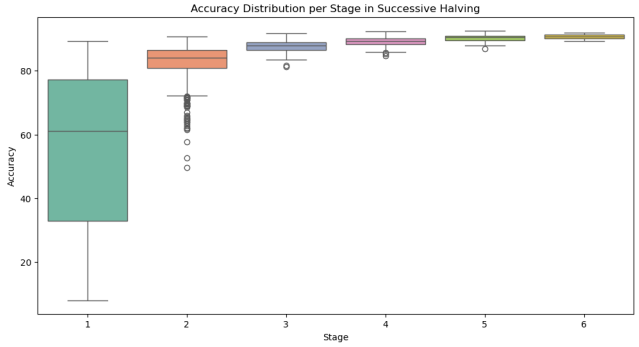

FIGURE 17: Accuracy Distribution of Different Stage

4.5. Comparative Analysis

In this comparison, we selected the data of the first 12,500 seconds for each HPO algorithm for analysis. Grid Search uses more resources than other algorithms, it appears in the line graph, but its results are not included in the comparison. I sampled the line graph with 50 data points as a sampling range to facilitate trend observation. The optimal model and reasonable model (the model validation accuracy over 90%) of each HPO algorithm are also marked in the line graph.

4.5.1. Trend Analysis

By observing the line graph of each algorithm, we can see the characteristics and search behaviours:

- Grid Search: The data of Grid Search only shows the initial part, so it is not included in the comparison.
- Random Search: The results of Random Search are relatively random. This is consistent with the characteristics of Random Search. Random search has no fixed pattern, and the performance of each model is unpredictable. In a reasonable search space, high-quality solutions can be found through multiple attempts.
- Genetic Algorithm: Genetic Algorithm shows small fluctuations but maintains a high accuracy trend overall. This is consistent with the characteristics of the Genetic Algorithm: eliminating low-quality combinations and trying different hyperparameter combinations through crossover and mutation mechanisms. It makes sure the algorithm explores the search space within a high accuracy range, and still maintains the possibility to try mutated combinations to avoid the local minimum.
- Bayesian Optimization: Bayesian Optimization occasionally has large fluctuations while maintaining high accuracy. This might be the reason that it attempts to adjust the fitting curve of the objective function by locating the position of low-accuracy combinations.
- Successive Halving: Successive Halving has significant accuracy fluctuations in Stage 1. When the stage gets higher, the accuracy gradually increases. This is consistent with the core of its algorithm, training resources are reallocated on models with better performance.

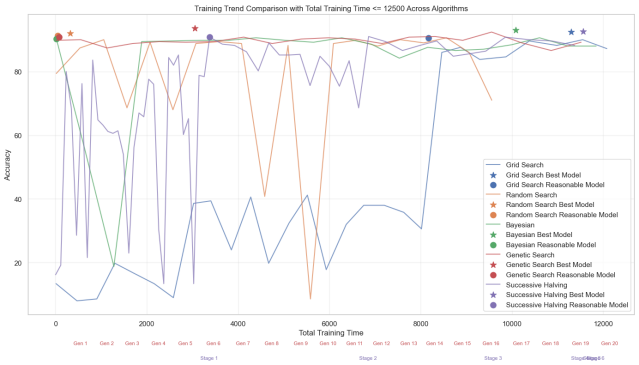

FIGURE 18: Trend of HPO algorithms

4.5.2. Speed of finding reasonable models and optimal models

This section compares the efficiency of each HPO algorithm in finding reasonable models and optimal models to reveal the search efficiency of the algorithm.

- Grid Search: Grid Search has to traverse the entire search space, the speed of Grid Search to find the optimal model depends on the traversal order. Therefore, the efficiency of Grid Search is not compared in this analysis.
- Random Search: In this experiment, Random Search has shown good efficiency and can quickly find reasonable models and optimal models in the early stage. This shows that in a properly defined search space, Random Search is an effective search method. Random Search is suitable for finding high quality solutions in resource limited situation.
- Genetic Algorithm: Since the initial population of Genetic Algorithms is generated based on random selection(the same as random search), it is highly probable that a reasonable model will be found in the early stage. In this experiment, the optimal model of Genetic Algorithms mostly appears in the middle stage, but due to the characteristics of crossover and mutation, the optimal solution can appear in any stage.
- Bayesian Optimization: The initial sample set of Bayesian Optimization is also generated based on random selection. A reasonable model will probably be found in the early stage due to the same reason. The global optimal of Bayesian Optimization can appear at any time. The fitting curve of the objective function gets more accurate as time goes on. Theoretically, it is more possible to find a better solution at the late stage. In this experiment, the optimal model of Bayesian Optimization mostly appears in the middle and late stages.

- Successive Halving: Although Successive Halving initialization is also based on random selection, it is hard to find a reasonable solution or optimal solution in the early stage due to the small amount of resource allocation. Usually, a reasonable solution appears at the beginning of the second stage because the algorithm sorts and eliminates half of the models, the first model that is trained in the second stage is the model with the highest performance in the first stage. In this experiment, the optimal solution of Successive Halving mostly appears in the late stages, which is consistent with its elimination strategy and resource reallocation mechanism.

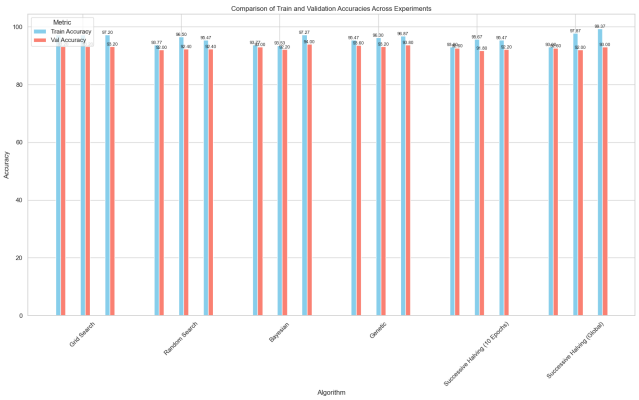

FIGURE 19: Comparison of Best Models

4.5.3. Comparison of the performance of the best models

When comparing the performance of the best models of each algorithm, we list the 10 epoch best models and the global best model of the Successive Halving algorithm separately, and compare them with the best models of other HPO algorithms to show the performance under the same computation resources.

- Grid Search: Since Grid Search only conducted one experiment, the results showed good model performance, with a training accuracy of 97.2% and a validation accuracy of 93.2
- Random Search: The results of Random Search in multiple attempts were similar, and the validation accuracy remained at around 92.4%, Random Search shows a consistent fast search ability.
- Bayesian Optimization: Bayesian Optimization achieved the highest validation accuracy in the experiment, reaching 94%. Bayesian Optimization shows a strong global search ability.
- Genetic Algorithm: The validation accuracy of the Genetic Algorithm remained above 93% in multiple attempts. Genetic Algorithm shows a stable and efficient, with the highest validation accuracy of 93.8%.
- Successive Halving: The validation accuracy of Successive Halving is relatively low, averaging around 92%. This does not mean that its algorithm is inefficient, it is related to the experimental design and data set characteristics. Due to the resource allocation strategy of Successive Halving, some combinations that may perform well with more resources are eliminated in the early stage, which may cause its optimal solution to be slightly inferior to other algorithms.

4.5.4. Stability Analysis

Stability reflects the consistency of the results in multiple attempts. By comparing the performance of each algorithm, all algorithms show good consistency in all three aspects.

5. CONCLUSION

This experiment is mainly an exploration of the HPO algorithm. It introduces the concepts related to the HPO algorithm and defines the search space. Through experimental comparison, I have a deep understanding of the efficiency and accuracy of each HPO representative algorithm, and summarize its advantages and limitations.

- Grid Search: It can fully explore the entire search space. However, it consumes a lot of resources, which is often not affordable, and because its parameters are discrete, its search space is sparse and may skip the global optimal solution.
- Random Search: It is very simple to implement and is very suitable for situations where you want a reasonable solution with very few attempts. When the search space is well defined, random search is very effective.
- Genetic algorithm: It retains the advantages of random search in the initialization stage and uses the advantages of crossover and mutation mechanisms to enable it to avoid local minimum and always maintain high accuracy. The disadvantage is that when the mutation rate is not high, most models lack changes, which easily leads to many useless attempts.
- Bayesian optimization: In general, Bayesian optimization is currently the most commonly used HPO algorithm. The highest model validation accuracy in this experiment was found by BO. Because Bayesian optimization samples and fits the objective function in a continuous space, in theory, as long as the time is enough, Bayesian optimization can find the global optimal solution, but this process consumes a lot of resources.

Fortunately, Bayesian optimization can find a reasonable solution in a very short time. Manual implementation is relatively complicated, so it requires the help of automated tools.

- Successive halving: It is a basic algorithm for incremental resource allocation that is very easy to implement. It is very suitable for finding a reasonable solution when resources are fixed. In general, I think it is not as flexible and efficient as Bayesian optimization, but it is simple to implement, does not require any experience, and can save a lot of wasted resources due to useless attempts, which is suitable for beginners.

In this experiment, there is another important point worth noting: the HPO algorithm itself also has hyperparameters that need to be adjusted. For example, in the Genetic algorithm, the mutation rate, the number of initial populations, and the number of iterations (generations) will significantly affect the effect of the algorithm; Bayesian optimization depends on the sampling strategy and the parameters of the surrogate model. It is worth noting that the hyperparameters of these algorithms are difficult to further optimize through HPO nesting, because doing so will cause the computing resources to grow exponentially, and it is unaffordable.

Therefore, even if we can simplify the process of finding hyperparameters when using the HPO algorithm, full automation is still challenging. HPO can not completely replace human knowledge and experience.HPO can help to simplify complex problems into more feasible problems. This process enables developers and learners to find reasonable solutions with fewer attempts, saving resources while also lowering the threshold for hyperparameter optimization.

6. REFERENCE

[1] Yu, T., & Zhu, H. (2020). Hyper-parameter optimization: A review of algorithms and applications. arXiv preprint arXiv:2003.05689.

[2] Boyd, S., & Vandenberghe, L. (2004). Convex Optimization. Cambridge University Press.

[3] Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of machine learning research, 13(2).

[4] Hutter, F., Kotthoff, L., & Vanschoren, J. (Eds.). (2019). Automated Machine Learning: Methods, Systems, Challenges. Springer.

[5] Jatender Kumar. (2024). Hyperparameters in Deep Learning: A Comprehensive Review. International Journal of Intelligent Systems and Applications in Engineering, 12(4), 4015 –. Retrieved from https://www.ijisae.org/index.php/IJISAE/article/view/6967

[6] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[7] Goldberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning. Addison-Wesley.

[8] Massachusetts Institute of Technology. (2021). Introduction to Artificial Intelligence. MIT OpenCourseWare. https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/

[9] Nguyen, Q. (2024). Bayesian Optimization in Action. Manning Publications.

[10] Feurer, M., Springenberg, J., & Hutter, F. (2015, February). Initializing bayesian hyperparameter optimization via meta-learning. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 29, No. 1).

[11] Rasmussen, C. E., & Williams, C. K. I. (2006). Gaussian Processes for Machine Learning. The MIT Press.

[12] Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. Advances in neural information processing systems, 25.

[13] Zoph, B. (2016). Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578.

[14] Elsken, T., Metzen, J. H., & Hutter, F. (2019). Neural architecture search: A survey. Journal of Machine Learning Research, 20(55), 1-21.

[15] Sharma, A., & Kumar, D. (2023, November). Hyperparameter Optimization in CNN: A Review. In 2023 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS) (pp. 237-242). IEEE.

[16] Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. Advances in neural information processing systems, 25.

[17] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2018). Hyperband: A novel bandit-based approach to hyperparameter optimization. Journal of Machine Learning Research, 18(185), 1-52.