HPE-DARTS: Hybrid Pruning and Proxy Evaluation in Differentiable Architecture Search

Hung-I Lin[1], Lin-Jing Kuo[2] and Sheng-De Wang[1]
[1]*Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan*
[2]*Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan*

Keywords: Deep Learning, Neural Architecture Search, Differential Neural Architecture Search, Hybrid Pruning, Proxy Evaluation.

Abstract: Neural architecture search (NAS) has emerged as a powerful methodology for automating deep neural network design, yet its high computational cost limits practical applications. We introduce Hybrid Pruning and Proxy Evaluation in Differentiable Architecture Search (HPE-DARTS), integrating soft and hard pruning with a proxy evaluation strategy to enhance efficiency. A warm-up phase stabilizes network parameters, soft pruning via NetPerfProxy accelerates iteration, and hard pruning eliminates less valuable operations to refine the search space. Experiments demonstrate HPE-DARTS reduces search time and achieves competitive accuracy, addressing the reliance on costly validation. This scalable approach offers a practical solution for resource-constrained NAS applications.

1 INTRODUCTION

Neural Architecture Search (NAS) automates neural network design, addressing the inefficiencies of manual approaches by systematically exploring a range of architectures tailored to specific tasks (Elsken et al., 2019; Ren et al., 2021). Early NAS methods, such as those by (Zoph and Le, 2017; Zoph et al., 2018; Real et al., 2019), incurred high computational costs due to repeated training for performance evaluation. To mitigate this, Differentiable ARchiTecture Search (DARTS) (Liu et al., 2019) introduced gradient-based optimization, significantly reducing costs while delivering competitive results through continuous relaxation of architecture representation for simultaneous weight and parameter optimization. Moreover, its gradient-based approach established a foundation for more efficient NAS methodologies.

Despite its advancements, DARTS faces some issues and inefficiencies. P-DARTS (Chen et al., 2019) points out an optimization gap that leads to accuracy drops when searching with a super-net and evaluating with a sub-network. The P-DARTS approach narrows the gap by progressively increasing the network depth while reducing the search space during the search stage. VP-DARTS (Feng and Wang, 2024) addresses the performance collapse caused by excessive skip connections in the final architecture by transforming the architecture search as a model pruning problem, and applying a soft pruning technique with decaying the architecture parameters of unimportant operations to prevent the direct removal of high potential operations.

However, both VP-DARTS and P-DARTS face notable limitations. VP-DARTS relies heavily on the validation dataset to assess operation importance, leading to increased search time and low efficiency. While P-DARTS enhances efficiency by progressively narrowing the search space, it discards all supernet parameters and weights each time the network depth increases, hindering overall performance. Additionally, its strategy for pruning less important operations based on architecture parameters has proven to be inaccurate in (Feng and Wang, 2024), further impacting search results.

To address these challenges, we propose Hybrid Pruning and Proxy Evaluation in Differentiable Architecture Search (HPE-DARTS). Combining soft and hard pruning, HPE-DARTS refines architecture parameters while removing unimportant operations. Additionally, we adapt Neural Architecture Search WithOut Training (NASWOT) (Mellor et al., 2021) into NetPerfProxy, enhancing efficiency and accuracy in evaluating operations within DARTS-like search spaces.

Lin, H.-I., Kuo, L.-J. and Wang, S.-D.
HPE-DARTS: Hybrid Pruning and Proxy Evaluation in Differentiable Architecture Search.
DOI: 10.5220/0013148700003890
In *Proceedings of the 17th International Conference on Agents and Artificial Intelligence (ICAART 2025) - Volume 2*, pages 252-263
ISBN: 978-989-758-737-5; ISSN: 2184-433X

The contributions of our work can be comprehensively detailed and summarized as follows:

- **Enhanced Efficiency.** We introduce a novel hybrid pruning strategy that enhances the performance and the efficiency with respective to VP-DARTS, Furthermore, we achieve comparable results to P-DARTS while reducing the search time cost.
- **Improved Evaluation Method.** We developed NetPerfProxy, which is an adaptation and modification of NASWOT, to use within DARTS-like search spaces. The results show that using NetPerfProxy to evaluate operations could achieve superior results compared with using NASWOT, making it more suitable for applying on DARTS-like search space.
- **Generalization Ability.** HPE-DARTS is useful when applying to DARTS search space, which requires a comprehensive exploration of operations and connections between nodes. Experimental results show that HPE-DARTS maintains its effectiveness across different search spaces, demonstrating generalization capabilities and providing competitive performance against state-of-the-art methods.

The paper is organized as follows: Section 2 provides a literature review, highlighting the state-of-the-art approaches and their limitations. Section 3 details the proposed methodology, including the hybrid pruning strategy and the design of NetPerfProxy. Section 4 presents experimental results and comparative analyses. Finally, Section 5 discusses conclusions and future research directions.

2 RELATED WORK

2.1 Neural Architecture Search

Traditional hand-crafted architectures like VGG (Simonyan and Zisserman, 2015) and ResNet (He et al., 2016), have demonstrated significant success across various domains, including image classification, object detection, semantic segmentation, etc. These architectures are known for their deep layers and the ability to capture complex features. However, designing these networks often requires extensive domain knowledge and a substantial amount of trial-and-error, which can be time-consuming and inefficient (Elsken et al., 2019; Ren et al., 2021; White et al., 2023). Thus, Neural Architecture Search (NAS) has gained lots of attention because it evolved from manual designs to automated systems, reducing reliance on human expertise and time costs. NAS methodologies are commonly categorized based on the search space, search strategy, and performance estimation strategy they employ (Kyriakides and Margaritis, 2020). Some earlier studies utilized reinforcement learning (RL) (Zoph and Le, 2017; Baker et al., 2017) as a search strategy to automatically generate neural architectures for image classification tasks. In contrast to developing a neural architecture gradually by using RL, other research employed the evolutionary algorithm (EA) (Xie and Yuille, 2017; Real et al., 2017) to identify optimal neural architectures from a set of potential neural architectures. However, with the increase of the NAS algorithm complexity, to achieve better results, the algorithms even need hundreds of GPU days to search for a good architecture. It is obviously a dilemma to utilize these NAS methods in practical applications.

2.2 DARTS

In order to accomplish the architecture search within a short period of time, one-shot techniques were introduced to avoid training each architecture from scratch, thus circumventing the associated computational burden. Instead of training each architecture from scratch individually to evaluate performance, one-shot approaches optimize all networks within the search space by training a single, over-parameterized ”supernet.” This supernet encompasses all possible architectures as sub-networks, allowing for simultaneous training and evaluation (Pham et al., 2018; Bender et al., 2018; Saxena and Verbeek, 2016; Xie et al., 2021). Once a supernet is trained, each architecture in the search space could be derived from the supernet by inheriting the weights to evaluate. While the supernet allows quick evaluations of all architectures, it still needs a search strategy to search for a good architecture.

Differentiable Architecture Search (DARTS) (Liu et al., 2019) employs a continuous relaxation of the discrete search space, enabling the application of gradient descent to find a good architecture. In DARTS, as depicted in Figure 1, each edge simultaneously contains all candidate operations, weighted by architecture parameters (α). The architecture parameters are optimized concurrently with the supernet's weights through gradient descent to determine the contribution of each operation. After the search, the final model is derived by choosing the operation with the highest architecture parameter on each edge and then retrained from scratch to evaluate its final performance. DARTS gained significant attention due to its simplicity and efficiency, reducing the computational cost and search time of traditional NAS methods to only a couple of GPU days.

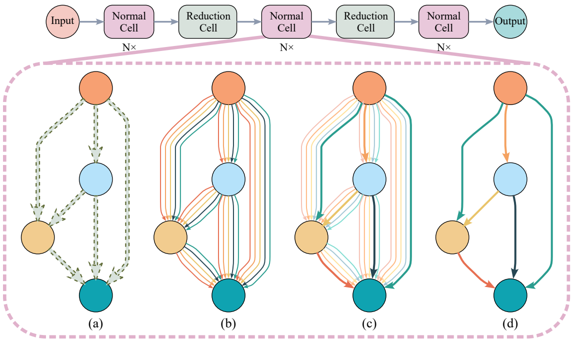

Figure 1: Stages of architecture search in DARTS (Liu et al., 2019): (a) Initial state with unknown operations on the edges (b) Applying continuous relaxation of the search by mixing candidate operations with learnable parameters on each edge. (c) Joint optimization of the mixing learnable parameters and the network weights by gradient descent. (d) The final architecture is derived by selecting the most influential operations based on the learned parameters.

Despite the advantages of using DARTS, several works (Zela et al., 2020; Wang et al., 2021; Liang et al., 2019) show that differentiable NAS tends to favor the parameter-free operations like skip connections and leads to under-performance, which may be caused by the supernet using skip connections to over-compensate the vanishing gradients problem (Chu et al., 2021). Beta-DARTS (Ye et al., 2022) proposed Beta-Decay regularization to constrain the values of network parameters after softmax from changing too much. Fair-DARTS (Chu et al., 2020) remove the softmax function on architecture parameters to set all operations independent of all others, and then apply an additional loss function which pushes the architecture parameters toward 0 or 1.

Moreover, constructing a supernet reduces the need to train lots of candidate architectures, while suffocating with the high memory consumption to train a supernet. PC-DARTS (Xu et al., 2020) randomly sample a proportion of channels to perform operation searches while bypassing the rest of the channels in a shortcut. DrNAS (Chen et al., 2021a) adopts a progressive approach by gradually increasing the fraction of channels forwarded while concurrently reducing the operation space by pruning less critical operations, and models the architecture parameter as Dirichlet distribution.

However, there is an optimization gap that leads to an accuracy drop when searching with a super-net and evaluating with a sub-network because of the inconsistency of the settings between searching and evaluating. P-DARTS (Chen et al., 2019) alleviates the gap

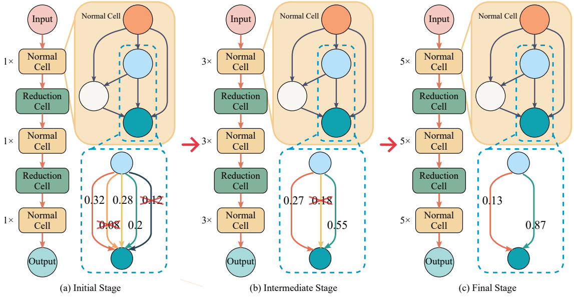

Figure 2: The overall pipeline of P-DARTS (Chen et al., 2019). (a) The initial stage presents the shallowest version of the network, incorporating all candidate operations on its edges. (b) By the intermediate stage, the network deepens and eliminates less critical operations. (c) The final stage displays a further deepened and optimized network architecture.

by progressively increasing the network depth while simultaneously reducing the search space during the search stage as illustrated in Figure 2. SGAS (Li et al., 2020) gradually replaces the mixture on each edge with the most critical operation based on the greedy Selection Criterion, which is formed by the entropy of the weights of non-zero operations and the changes of the weights within a history window, to divide the search procedure into sub-problems. VP-DARTS (Feng and Wang, 2024) formulates the architecture search as a model pruning problem as shown in Figure 3 and applies a soft pruning technique by decaying the architecture parameters α of unimportant operations, which is evaluated by temporally removing it on each edge and calculating the accuracy drop of the supernet, i.e., larger accuracy drop means the more important of the operation.

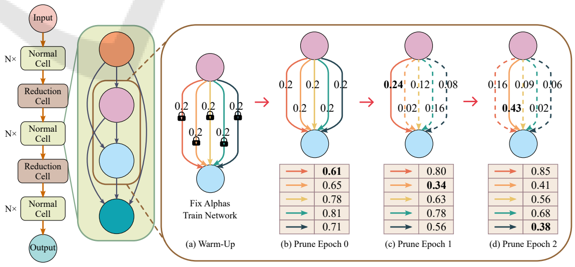

Figure 3: The overall pipeline of VP-DARTS (Feng and Wang, 2024). VP-DARTS employs a soft pruning technique that involves decaying the architecture parameters of operations that are less important. In each pruning epoch, operations are temporarily removed to assess their impact on the overall accuracy of the supernet. The process includes a warm-up stage, where architecture parameters are fixed and the network is trained, followed by subsequent pruning epochs with a soft pruning technique to prevent pruning high potential operations.

Our method builds on the foundational strategies of the soft pruning from VP-DARTS and P-DARTS's strategy of progressively shrinking the search space. By integrating soft pruning, our approach uses network performance to selectively prune operations, ensuring the most effective operations are retained, and others could regain their importance later in the search process. Simultaneously, we implement progressive search space reduction from P-DARTS as a hard pruning method, enhancing the efficiency of the search process. This hybrid approach aims to address the challenges of overfitting and computational inefficiency in NAS, yielding robust, high-performing models.

2.3 Approach of Evaluation

DARTS (Liu et al., 2019) employs a continuous relaxation of the discrete search space, constructing a supernet where each edge contains all candidate operations associated with learnable architecture parameters. Due to the continuous relaxation, DARTS makes the architecture parameters differentiable and could utilize gradient-based optimization to search for a good architecture. However, when selecting the most crucial operation on each edge, DARTS selects the operation with the highest architecture parameter which can be misleading. This is because the architecture parameters are optimized as ratios to combine the operations rather than indicating the most plausible operation.

Besides, the recent success of deep convolutional neural network (Simonyan and Zisserman, 2015; He et al., 2016) has been linked with the increasing requirement of computation resources due to over-parameterization. To address this issue, network pruning has gained a lot of attention due to its competitive performance and compatibility, as it removes the redundant parameters that do not significantly contribute to accuracy. However, identifying the redundant parameters to prune becomes a critical issue. An early method to prune networks is brute-force pruning, traversing the entire network element-wise and removing weights that do not affect accuracy. However, the brute force could be more efficient and practical; later works change to adopt l_1-norm and l_2-norm to calculate the importance of the weights or layers and prune those closer to zero. VP-DARTS (Feng and Wang, 2024) observes that the over-parameterized ”supernet” in DARTS also needs to be ”pruned” to find the final architecture with minimal accuracy drop. Therefore, VP-DARTS formulate the differentiable NAS as a model pruning problem, evaluating the importance of operations by their impact on the supernet’s validation accuracy when selectively temporally pruned. This pruning-based evaluation method improves the decision-making process in DARTS while significantly increasing the time and computational resources required due to numerous evaluations during the search process.

Recently, training-free approaches (Mellor et al., 2021; Wu et al., 2024; Wu et al., 2022) have emerged as cost-effective alternatives in Neural Architecture Search (NAS) by eliminating the need for a training process to evaluate architectural performance. NAS without training (NASWOT) (Mellor et al., 2021) introduces a scoring function that predicts architecture efficacy based on the overlap of activations among inputs without requiring any training procedure. The fundamental premise of NASWOT is that a greater disparity for different inputs at each convolutional layer indicates a higher potential for the network to learn to distinguish between those inputs effectively. For instance, as shown in Figure 4, data points i and j pass through a convolution layer in a network, and S_{ij} is referred to the number of same bits between binary codes S_i and S_j. Then, the score is defined as :

$$\mathbb{K}^C = \begin{bmatrix} L & S^C_{12} & \dots & S^C_{1N} \\ S^C_{21} & L & \dots & S^C_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ S^C_{N1} & S^C_{N2} & \dots & L \end{bmatrix}, \qquad (1)$$

$$\mathrm{NASWOT}(Net) = \log\det\Big(\sum_{\text{layer } C\in\text{Net}} \mathbb{K}^C\Big). \qquad (2)$$

That is, the logarithm determinant of the summation of the matrix formed by S_{ij} from each convolution layer C, and the L in the matrix is defined as the length of the binary codes. This innovative approach predicts scores with a high correlation with trained network performance and drastically reduces the time cost in NAS by replacing the training process. However, NASWOT was not initially designed to handle DARTS-like architectures where multiple operations compete on a single edge, each modulated by their respective architecture parameters α.

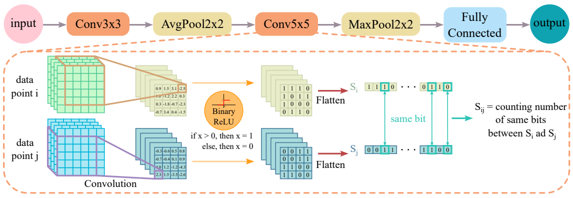

Figure 4: A simple example to illustrate the concept of NASWOT score.

3 APPROACHES

Our method enhances the Differentiable ARchiTecture Search (DARTS) framework (Liu et al., 2019) by introducing a novel hybrid pruning strategy that synergizes the strengths of soft and hard pruning to enhance both the efficiency and effectiveness of the architecture search process. We also integrate a modified version of NASWOT (Mellor et al., 2021) as NetPerfProxy, adapted for DARTS-like search spaces with multiple operations per edge. This adaptation enables rapid and effective performance predictions during the architecture search, streamlining the evaluation process.

3.1 Preliminaries

In DARTS, the search space is defined by a repeated cell structure. Each cell is modeled as a directed acyclic graph (DAG) with N nodes, where each node represents an intermediate feature map and each directed edge (i,j) between nodes i and j carries an operation $o^{(i,j)}$ that transforms the feature map $x^{(i)}$. By parameterizing the operations on the edges, each edge (i,j) in the directed acyclic graph becomes a mixed operation $\bar{o}^{(i,j)}$ that is a weighted combination of all candidate operations $o \in O$, and is thus formulated as:

$$\bar{o}^{(i,j)}(x^{(i)}) = \sum_{o \in O} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in O} \exp(\alpha_{o'}^{(i,j)})} o(x^{(i)}), \quad (3)$$

where the weights are derived from the learnable architecture parameters $\alpha_o^{(i,j)}$ using the softmax function, which normalizes these parameters across all operations and mixes the outputs of different operations on an edge.

With continuous relaxation, DARTS allows for the simultaneous optimization of the learnable architecture parameters α and the network parameters ω via gradient descent, and this bi-level optimization problem can be expressed as:

$$\begin{aligned} \min_{\alpha} \quad & \mathcal{L}_{\text{val}}(\omega^*(\alpha), \alpha) \\ \text{s.t.} \quad & \omega^*(\alpha) = \arg\min_{\omega} \mathcal{L}_{\text{train}}(\omega, \alpha). \end{aligned} \quad (4)$$

Upon completion of the search phase in DARTS, the final architecture is derived by discretizing the continuously relaxed architecture representation; that is, DARTS selects the operation with the highest weight on each edge, effectively converting the probabilistic mixture of operations into a discrete choice.

3.2 Hybrid Pruning Mechanism

Our methodology refines the DARTS framework by implementing a hybrid pruning strategy that enhances the efficiency and efficacy of the architecture search by incorporating both soft and hard pruning techniques at different phases of the search. As illustrated in Algorithm 1, the process begins with a warm-up stage with E_{warmup} epochs and progresses through S hybrid pruning stages, each involving soft and hard pruning techniques denoted by SoftP and HardP. For the hard pruning parts within these hybrid pruning stages, it is essential to pre-define a vector of constants:

$$Pop = [Pop_1, Pop_2, \cdots, Pop_S], \quad (5)$$

as hyper-parameters, where each Pop_i specifies the number of operations to be hard pruned in the i-th hybrid pruning stage. Additionally, it is crucial that the cumulative effect of the Pop values across all stages results in only one operation left on each edge after the search. Specifically, the sum of all Pop_i values, from the first to the S-th stage, must equate to $|O|-1$, where $|O|$ represents the total number of candidate operations. This ensures that exactly one operation is left on each edge after the search process:

$$\sum_{i=1}^{S} Pop_i = |O| - 1 \quad (6)$$

where O is the set of candidate operations in the search space. At each hybrid pruning stage, soft pruning decays the less important operations while allowing them to recover in the later epochs, which is described in Section 3.2.2. After soft pruning, hard pruning is applied to remove the Pop_i less important operations to reduce the search space for searching more efficiently on important operations as explained in Section 3.2.3. This iterative refinement continues until each edge of the network retains only its most vital operation, resulting in an optimized final architecture.

Moreover, to address the inefficiency of the architecture evaluation method proposed by (Feng and Wang, 2024), we adopt NAS Without Training (NASWOT) (Mellor et al., 2021), a quick performance estimation technique by assessing the connectivity and potential of the architecture without any training, and modify it as NetPerfProxy to predict better on DARTS-like search space, which will be introduced in Section 3.3.

3.2.1 Warm-up Stage

The warm-up stage serves as the foundation of our pruning strategy, inspired by techniques used in VP-DARTS (Feng and Wang, 2024). During this initial stage, the architecture parameters α are fixed to stabilize the training dynamics of the network. This allows the network weights to learn without the influence of changing architecture parameters. The fixed

Algorithm 1: Hybrid Pruning Search Strategy.

```
Require: Number of hybrid pruning stages $S$
Require: Number of warmup epochs $E_{warmup}$; Number of pruning epochs $E_{prune}$
Require: Training dataset $D_{train}$; Validation dataset $D_{val}$
Require: Numbers of pruning operations at each hybrid stage $Pop = [Pop_1, Pop_2, \cdots, Pop_S]$
Ensure: Final derived architecture: $Arch$
Create a supernet $SNet$ with weights $\omega$ and architecture parameters $\alpha$
// Warm-up Stage:
for $e = 1$ to $E_{warmup}$ do;
    Train network weights $\omega$ on $D_{train}$ with fixed architecture parameters $\alpha$
end for
// Hybrid Pruning Stages:
for $st = 1$ to $S$ do;
    $SNet$=SoftP($SNet$, $E_{warmup}$, $D_{train}$, $D_{val}$)
    $SNet$=HardP($SNet$, $Pop[st]$, $D_{val}$)
end for
Derive the final architecture $Arch$ based on Pruned supernet $SNet$
```

architecture parameters during this phase ensure that the model's focus is on learning robust feature representations before any pruning begins, setting a strong baseline for subsequent optimization steps.

3.2.2 Hybrid Pruning Stage: Soft Pruning Part

Following the warm-up, we transition to the hybrid pruning stages. In the first half part of each hybrid pruning stage, soft pruning will be applied for E_{prune}. At each epoch, influenced by the soft pruning approach in VP-DARTS (Feng and Wang, 2024), the operations on each edge of the cell are evaluated individually for their impact on the network performance. The importance of each operation is evaluated by examining the performance drop after temporarily removing the operation from the network. If network performance decreases significantly, the removed operation is quite important. After evaluating each operation on all edges, the architecture parameters α of these unimportant operations will be gradually decayed by the soft pruning decay strategy in (Feng and Wang, 2024) as shown in Equation 7:

$$\alpha_{o,i} = \alpha_{o,i-1} * (1 - \frac{i}{E_{prune}}), 0 \leq i < E_{prune}, \quad (7)$$

where i represents the current epoch, E_{prune} is the total number of pruning epochs, and o is the operation on each edge. This reduction is not immediate elimination but a down-weighting process, allowing these operations to have a decreasing influence over the model's output while still having the chance to recover in later epochs if these operations have high potential. This pruning method is less aggressive and permits the network to adapt smoothly to changes in its architecture, preserving critical operations while de-emphasizing the less important ones. After updating the architecture parameters, the supernet will be trained to refine the weights to cooperate with the updated architecture parameters.

3.2.3 Hybrid Pruning Stage: Hard Pruning Part

After soft pruning has effectively identified and down-weighted the less important operations, we implement hard pruning to eliminate them decisively. This hard pruning part is inspired by the strategies used in P-DARTS (Chen et al., 2019; Chen et al., 2021b). Similar to soft pruning, the operations on each edge of the cell are evaluated individually. After sorting operation importance, the least important *Pop* operations on each edge are completely removed from the architecture. This action significantly reduces the computational complexity and narrows the search space, enabling a more focused and efficient optimization of the remaining operations. Hard pruning acts as a conclusive refinement step, ensuring that only the most critical operations are carried forward to derive the final architecture.

3.3 Network Performance Proxy

To address the inefficiency of the traditional architecture evaluation method, we adopt NAS Without Training (NASWOT) (Mellor et al., 2021), a quick performance estimation technique by assessing the connectivity and potential of the architecture that does not require training, to use as the evaluation method. More specifically, when using the evaluation method of examining the performance drop after temporarily removing the operation from the network, we use NASWOT to test the network performance instead of using the validation dataset. Although only substituting the validating process, not training, it still could reduce the time cost, especially since the evaluation method is frequently used in soft pruning. However, the original NASWOT framework was primarily designed for simpler architectures where each edge in the network's graph represented a single operation. This model did not account for the complexities of a DARTS-like search space, where multiple candidate operations can exist on a single edge, each weighted by learnable parameters α.

Thus, we developed NetPerfProxy, a modified version of NASWOT, to effectively operate within such

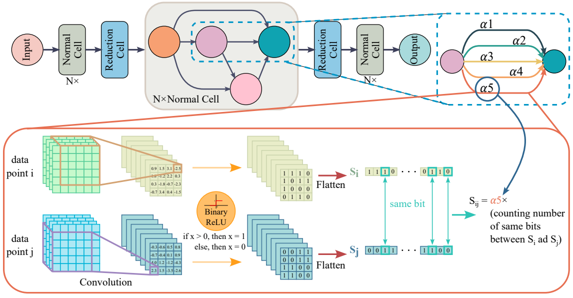

Figure 5: A demonstration of the concept of NetPerfProxy.

a search space. In order to integrate this dynamic into NASWOT's architecture evaluation, we modified the way NASWOT calculates the Hamming distance, which serves as a measure of similarity between data points in a mini-batch based on their activation patterns. Specifically, as illustrated in Figure 5, data points i and j pass through the operation with architecture parameter α_5 on the edge and form the binary code S_i and S_j that represents the data points being activated or not. When computing the Hamming distance between activation patterns S_i and S_j, the NetPerfProxy multiplies the Hamming distance $|S_i - S_j|$ by the architecture parameter α_5 associated with each operation on the edges. Thus, the score of each edge with data points i and j in a batch is defined as:

$$K^e_{ij} = \sum_{\text{operation } o \in e} |S^o_i - S^o_j| * \alpha_o, \tag{8}$$

where the score is calculated as the summation of multiplying the Hamming distance with an architecture parameter α_o of each operation o within edge e between pairs of data points. Moreover, the matrix $\mathbb{K}^e$ of size $N \times N$ is then formed by K^e_{ij} in Equation 8 on edge e where the N is the size of one mini-batch of data. The L in the matrix is the length of the binary codes.

$$\mathbb{K}^e = \begin{bmatrix} L & K^e_{12} & \dots & K^e_{1N} \\ K^e_{21} & L & \dots & K^e_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ K^e_{N1} & K^e_{N2} & \dots & L \end{bmatrix}. \tag{9}$$

Thus, the score function of NetPerfProxy can be defined as follows:

$$\text{NetPerfProxy(Net)} = \log\det\Big(\sum_{\text{edge } e \in \text{Net}} \mathbb{K}^e\Big), \tag{10}$$

where the score is the logarithm determinant of the element-wise summation of matrices $\mathbb{K}^e$ across all edges e within the network Net.

This approach ensures that operations deemed more important (i.e., those with higher architecture parameters) have a proportionally greater influence on the network's overall architecture score. This modification allows NetPerfProxy to more accurately predict the performance of complex, DARTS-like architectures where multiple operations compete on the same edge, thereby enhancing the efficiency and search results.

4 EXPERIMENTS

4.1 Results on NAS-Bench-201

4.1.1 Search Space

NAS-Bench-201 (Dong and Yang, 2020) enhances the reproducible Neural Architecture Search (NAS) landscape by offering a standardized benchmarking framework, providing fair and consistent comparisons across different NAS methods. The benchmark employs a fixed cell-based search space, where each cell is a directed acyclic graph (DAG) consisting of 4 nodes interconnected by 6 edges as shown in Figure 6. Each edge allows for an operation selected from a set of 5 candidates, including Zeroize, Skip Connection, 1x1 Convolution, 3x3 Convolution, and 3x3 Average Pooling. Consequently, this benchmark comprises a total of 15,625 unique architectures. We evaluate a search method across three distinct datasets: CIFAR-10, CIFAR-100, and ImageNet-16-120. This extensive evaluation allows for comparing the performance of various NAS methods under consistent experimental conditions.

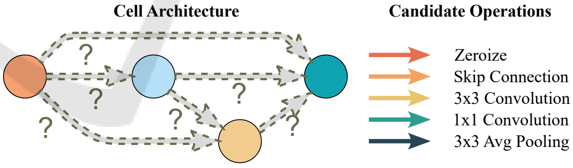

Figure 6: Overview of NAS-Bench-201 search space.

4.1.2 Implementation Details

At the beginning of our architecture search process, we initiate a warm-up stage consisting of five epochs. During this warm-up stage, the network parameters are held constant to ensure stability before any adjustments are made to the architecture. Following the initial warm-up stage, our architecture search process transitions into the hybrid pruning stages. Each hybrid pruning stage is designed to further refine the architecture by alternating between training, soft pruning, and hard pruning, allowing for gradual learning and adjustment to ensure that only the most effective operations are retained. In the first half of the hybrid pruning stages, the importance of operations during soft pruning is determined based on their impact on accuracy. For the second half of the hybrid pruning stages, we shift our strategy to incorporate our NetPerfProxy method for evaluating operation importance. We execute a hard pruning step after the five epochs of combined soft pruning and training in each hybrid pruning stage. During this step, the two least important operations are removed based on their performance impact calculated by NetPerfProxy. That is to say, each edge in the network's cell structure starts with five operations. After the first stage of hard pruning, this number is reduced to three operations per edge. Following the second stage, the number is further reduced, leaving just one operation per edge at the final stage. We conducted all experiments by searching on CIFAR-10 (Krizhevsky et al., 2009) and evaluating the performance across all datasets.

4.1.3 Search Results

The comparison results are shown in Table 1. We only searched on CIFAR-10 and used the found genotypes to query the performance of various datasets. To ensure robustness and repeatability, the results are averaged over 10 independent search runs. We could observe that HPE-DARTS demonstrates a remarkable reduction in search time, completing the process in just 0.61 hours on the CIFAR-10 dataset. This is considerably faster than other evaluated methods, including the more commonly referenced DARTS variants. For instance, the closest competitor, DrNAS, requires nearly three times as long at 1.66 hours. When compared to the original DARTS methods, HPE-DARTS operates in just about 61% and 18.7% of the time taken by the first-order and second-order versions of DARTS, respectively. This substantial decrease in search time does not come at the cost of performance; HPE-DARTS achieves competitive accuracies of 91.52 ± 0.04% on CIFAR-10 and 73.17 ± 0.34% on CIFAR-100 validation sets, closely mirroring the 'Optimal' benchmarks of 91.61% and 73.49% respectively. This efficiency provides an effective solution in situations where computational resources or time are constrained.

4.2 Results on DARTS

4.2.1 Search Space

In the Differentiable Architecture Search (DARTS) framework (Liu et al., 2019), the search space is structured around two types of cells: normal cells and reduction cells. In the network architecture, reduction cells are positioned at one-third and two-thirds of the total depth. All operations adjacent to the input nodes are implemented with a stride of two in these reduction cells,. Each cell is a small directed acyclic graph (DAG) consisting of an ordered sequence of 4 nodes, where each node can receive any of the outputs of the operation applied to the preceding nodes within the same cell as well as from the outputs of the immediate previous cell and the cell before it as shown in Figure 7. The operations on the edges are selected from a set of 8 candidate operations, resulting in a total of 8^{14} possible combinations. However, only two inputs per node are retained in the final architecture. This arrangement requires that the NAS algorithms not only determine the optimal operations to apply at each edge but also strategically select the connections between nodes.

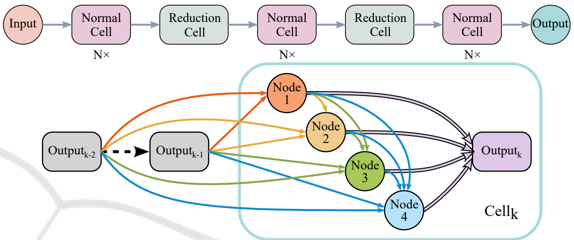

Figure 7: Detailed View of the DARTS Search Space and Cell Architecture. This diagram illustrates the overall network structure of DARTS search space and the internal configuration of a cell structure used in both Normal and Reduction Cells within the network. Each edge entering a node in the cell represents a mixture of eight candidate operations, and only two of the entering edges per node are retained in the final architecture.

4.2.2 Implementation Details

In adapting our architecture search strategy to the DARTS search space, we retain several core elements from our NAS-Bench-201 settings with necessary adjustments to fit the requirements of DARTS. On the DART search space, the search process involves one warm-up stage and three hybrid pruning stages. Throughout each of these hybrid pruning stages, the importance of operations during soft pruning is consistently evaluated using our NetPerfProxy approach. During the hard pruning part of each hybrid pruning stage, the number of operations that need to be pruned follows the structured reduction pattern inspired by P-DARTS. Initially, the network starts with eight operations on each edge. In the first hybrid pruning stage, three operations are removed, leaving five operations per edge. The second hybrid pruning stage is further reduced, with two more operations pruned to leave only three operations per edge. In the final hybrid pruning stage, two additional operations are removed, and only the single most important operation remains on each edge.

Table 1: Performance comparison on NAS-Bench-201 benchmark.

Methods	Search Cost (hours)	CIFAR-10		CIFAR-100		ImageNet16-120	
		validation(%)	test(%)	validation(%)	test(%)	validation(%)	test(%)
Non-weight sharing							
Random Search†	3.33	91.02 ± 0.34	93.73 ± 0.37	71.21 ± 1.17	71.46 ± 1.21	44.87 ± 1.25	45.06 ± 1.37
REA†(Real et al., 2019)	3.33	91.17 ± 0.31	93.94 ± 0.26	71.55 ± 0.77	71.91 ± 1.03	44.74 ± 0.93	45.24 ± 0.99
REINFORCE†(Williams, 1992)	3.33	90.07 ± 0.66	93.14 ± 0.65	69.87 ± 1.59	69.94 ± 1.51	43.30 ± 1.51	43.24 ± 1.90
BOHB†(Falkner et al., 2018)	3.33	89.43 ± 0.70	92.55 ± 0.63	68.68 ± 1.20	68.54 ± 1.40	41.64 ± 1.58	41.67 ± 1.80
Weight sharing							
RSPS (Li and Talwalkar, 2020)	0.80	81.04 ± 9.23	84.59 ± 9.61	55.95 ± 8.85	55.92 ± 8.85	30.77 ± 6.69	30.00 ± 6.51
ENAS (Pham et al., 2018)	1.51	39.80 ± 3.54	55.76 ± 3.22	14.17 ± 1.67	14.83 ± 1.54	16.17 ± 0.54	15.95 ± 0.71
GDAS (Dong and Yang, 2019b)	2.10	90.10 ± 0.17	93.44 ± 0.14	70.89 ± 0.34	70.54 ± 0.24	41.71 ± 0.95	42.05 ± 0.82
SETN (Dong and Yang, 2019a)	3.18	84.00 ± 4.34	87.20 ± 3.74	58.40 ± 7.05	58.55 ± 7.05	32.64 ± 5.32	31.91 ± 5.55
DARTS(1st) (Liu et al., 2019)	1.00	39.77 ± 0.00	54.30 ± 0.00	15.03 ± 0.00	15.61 ± 0.00	16.43 ± 0.00	16.32 ± 0.00
DARTS(2nd) (Liu et al., 2019)	3.26	39.77 ± 0.00	54.30 ± 0.00	15.03 ± 0.00	15.61 ± 0.00	16.43 ± 0.00	16.32 ± 0.00
DrNAS w/o PL‡(Chen et al., 2021a)	2.19	91.55 ± 0.00	94.36 ± 0.00	73.49 ± 0.00	73.51 ± 0.00	46.37 ± 0.00	46.34 ± 0.00
DrNAS (Chen et al., 2021a)	1.66	90.20 ± 0.00	93.76 ± 0.00	70.71 ± 0.00	71.11 ± 0.00	40.78 ± 0.00	41.44 ± 0.00
β-DARTS (Ye et al., 2022)	2.25	91.39 ± 0.17	94.13 ± 0.28	72.63 ± 0.98	72.74 ± 0.83	46.01 ± 0.35	45.59 ± 0.89
VP-DARTS (Feng and Wang, 2024)	2.06	91.28 ± 0.63	94.06 ± 0.44	72.61 ± 1.38	72.61 ± 1.40	45.80 ± 1.55	46.00 ± 1.33
HPE-DARTS	0.61	91.52 ± 0.04	94.31 ± 0.07	73.17 ± 0.34	73.24 ± 0.22	46.25 ± 0.28	46.42 ± 0.09
Optimal	-	91.61	94.37	73.49	73.51	46.77	47.31

† The search time limits to 12000 seconds
‡ Without progressive learning

4.2.3 Search Results

The experimental results are shown in Table 2. In our experimental setup, all searches were conducted on the CIFAR-10 dataset, while evaluations were extended across multiple datasets, CIFAR-10 and CIFAR-100, to assess generalizability. For robustness, each reported result is derived from the average of three independent runs with different random seeds, and we present the results as the average ± standard deviation. The result shows that HPE-DARTS achieves comparable results to the best-performing state-of-the-art methods, with an accuracy of 97.26 ± 0.14% on CIFAR-10 and 82.67 ± 0.29% on CIFAR-100. However, the search process on the CIFAR-10 dataset was completed in just 1.34 hours, requiring only 6% of the time taken by VP-DARTS, which was completed in 23.00 hours. Furthermore, the quickest among the evaluated state-of-the-art methods prior to HPE-DARTS was P-DARTS, which completed its search in 2.44 hours. HPE-DARTS completed the search process in approximately 55% of the time taken by P-DARTS, while still achieving competitive accuracies on both CIFAR-10 and CIFAR-100 datasets. These results highlight HPE-DARTS's ability to maintain competitive performance while drastically reducing the computational overhead, making it an attractive option for efficient yet effective neural architecture search.

4.3 Ablation Study

4.3.1 Number of Warm-up Epoches

The choice of setting warm-up epochs is crucial in our approach, because it allows the network parameters to stabilize before engaging in more computationally intensive pruning operations. We varied the warm-up epochs across four settings: 0, 5, 10, 15, and 20 epochs, while keeping the pruning epoch fixed at 5. For each configuration, we measured CIFAR-10 accuracy, search cost in hours, and the number of parameters of searched architecture, aiming to identify a sweet spot where the increase in accuracy justifies the additional computational expense.

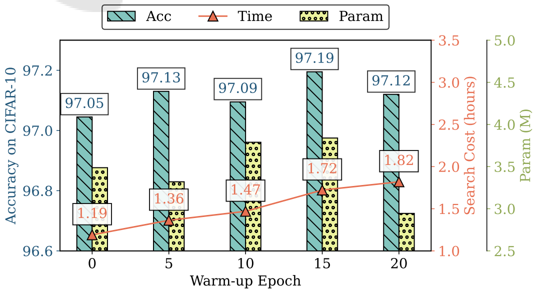

Figure 8: Comparison of HPE-DARTS using different Warm-up Epochs when Pruning Epochs set to 5. In this figure, the left blue bars represent the accuracy on CIFAR-10, and the red line stands for the search cost in hours. The number of parameters of the searched model is presented as the right green bars.

The results, as illustrated in Figure 8, clearly show that increasing the warm-up period from 0 to 5 epochs

Table 2: Performance comparison on DARTS search space.

Methods	Search Cost (hours)	CIFAR-10		CIFAR-100	
		Params(M)	Acc(%)	Params(M)	Acc(%)
DARTS(1st) (Liu et al., 2019)	3.11	3.34	97.41 ± 0.12	3.39	83.22 ± 0.45
DARTS(2nd) (Liu et al., 2019)	12.67	3.13	97.30 ± 0.10	3.18	82.64 ± 0.23
P-DARTS (Chen et al., 2019)	2.44	3.47	97.28 ± 0.13	3.52	82.53 ± 0.41
DrNAS (Chen et al., 2021a)	6.05	4.70	97.03 ± 0.15	4.75	82.66 ± 0.52
β-DARTS (Ye et al., 2022)	3.01	3.92	96.48 ± 0.62	3.97	80.66 ± 1.18
VP-DARTS (Feng and Wang, 2024)	23.00	2.44	96.92 ± 0.19	2.49	81.21 ± 0.56
HPE-DARTS	1.34	3.42	97.26 ± 0.14	3.47	82.67 ± 0.29

leads to an improvement in accuracy, from approximately 97.05% to 97.13%, representing the importance of the warm-up stage. This increase in accuracy is achieved with a modest increase in search cost from 1.19 hours to 1.36 hours, highlighting an efficient balance between computational expense and performance enhancement. While extending the warm-up period to 15 epochs achieves the highest accuracy of 97.19%, the corresponding search cost of 1.72 hours represents a larger jump in computational demand. By choosing 5 epochs, the approach efficiently manages to balance accuracy with the computational cost and complexity of the model.

4.3.2 Number of Pruning Epoches

The choice of setting pruning epochs is also important in our approach because it decides how many epochs the soft pruning technique would be applied. We varied the warm-up epochs across four settings: 5, 10, 15, and 20 epochs, while keeping the warm epoch fixed at 5 to experiment with how different pruning epochs impact both the accuracy on CIFAR-10 and the search cost in hours and trying to find which number of pruning epochs is suitable for our approach.

As shown in Figure 9, it illustrates a clear trend where extending the pruning epoch length significantly increases the computational cost, while the benefits in accuracy do not consistently increase with pruning epochs. At 5 pruning epochs, the model achieves a commendable accuracy of 97.13% on CIFAR-10. While there is a slight peak in accuracy at 10 pruning epochs (97.19%), the corresponding increase in search cost—from 1.36 hours at 5 epochs to 2.43 hours at 10 epochs—suggests a substantial rise in computational demands. Further extending the pruning epochs to 10, 15, or 20 shows either minor improvements or declines in accuracy but with disproportionate increases in search time and costs.

The increase in search time and computational resources required beyond 5 epochs does not justify the marginal gains in performance, underscoring why 5 epochs is the selected setting in our optimized approach. This choice ensures that our methodology remains computationally feasible while still providing competitive performance.

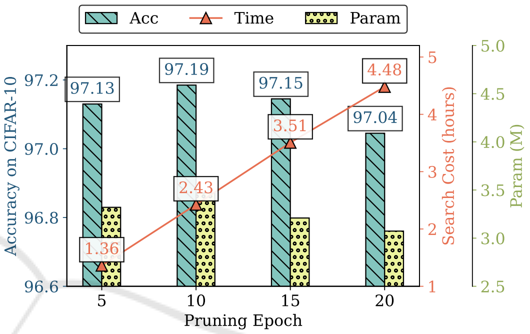

Figure 9: Comparison of HPE-DARTS using different Pruning Epochs when Warm-up Epochs set to 5. This figure illustrates the accuracy (blue bars, left y-axis), search cost in hours (red line, right y-axis), and number of parameters of the searched model (green bars, right y-axis) at pruning epochs 5, 10, 15, and 20. The search cost at 5 epochs (1.34 hours) provides an optimal balance between high accuracy (97.06%) and computational efficiency

4.3.3 Number of Stages and Operations to Hard Pruning

The list of numbers of operations influences how many less important operations would be hard pruned, which would also affect the number of hybrid pruning stages in the search and the search time of the search. In this experiment, both the warm-up epoch and pruning epoch are set to 5 and all the settings are presented in a list that represents the operations remaining at each hybrid stage.

As indicated in Table 3, the selection of the [8, 5, 3, 1] setting for hard pruning operations represents an optimal balance between search cost, model complexity, and accuracy on CIFAR-10. This setting leads to a search cost of 1.34 hours, which is considerably efficient compared to all the other settings. While the [8, 5, 3, 1] setting slightly increases the search duration compared to the [8, 4, 1] setting, which has the lowest cost of 1.05 hours, it achieves a higher accuracy of 97.26% compared to 97.14% offered by the [8, 4, 1] setting. The [8, 5, 3, 1] setting thus provides an effective trade-off between performance and efficiency for the hard pruning part of our approach.

Table 3: Comparison between using different settings to hard pruning operations.

Settings	Number of Stages	Search Cost (hours)	Params(M)	CIFAR-10 Acc(%)
[8,7,6,5,4,3,2,1]	7	2.73	2.94	97.04 ± 0.06
[8,6,4,2,1]	4	1.65	3.14	97.06 ± 0.13
[8,5,2,1]	3	1.30	3.51	97.10 ± 0.28
[8,4,1]	2	**1.05**	3.60	97.14 ± 0.01
[8,5,3,1]	3	1.34	3.52	**97.26 ± 0.14**

4.3.4 Impact of NetPerfProxy Modifications

NetPerfProxy enhances the original NASWOT framework by incorporating the architecture parameters of the operations with the NASWOT score, and this modification allows NetPerfProxy to better predict the actual performance of architectures by considering how different operations weighted by their architecture parameters influence the overall network performance. As demonstrated in Table 4, NetPerfProxy not only achieves a higher CIFAR-10 accuracy but also with a small increase in search cost compared to NASWOT, making NetPerfProxy an efficient tool to conduct a search on DARTS-like search spaces.

Table 4: Comparison between using NASWOT (Mellor et al., 2021) and NetPerfProxy on DARTS search space.

Methods	Settings	Search Cost (hours)	Params(M)	CIFAR-10 Acc(%)
HPE-DARTS	NASWOT	**1.29**	2.81	96.78 ± 0.19
	NetPerfProxy	1.34	3.52	**97.26 ± 0.14**

5 CONCLUSIONS

In this paper, we presented a NAS method called HPE-DARTS that integrates soft and hard pruning with a novel evaluation strategy, NetPerfProxy, to reduce time and resource consumption in differentiable architecture searches. HPE-DARTS accelerates the search process while maintaining competitive performance compared to state-of-the-art methods. It employs a warm-up phase to explore diverse architectural configurations, followed by iterative pruning to refine operations systematically, resulting in efficient convergence to high-performing models. NetPerfProxy enhances evaluation efficiency in DARTS-like search spaces, eliminating reliance on extensive validation. Currently tailored for convolutional neural networks, HPE-DARTS demonstrates strong potential for scalability. In the future, we will explore its application to larger datasets and complex architectures like transformers and incorporate progressive techniques to further advance neural architecture search capabilities.

REFERENCES

Baker, B., Gupta, O., Naik, N., and Raskar, R. (2017). Designing neural network architectures using reinforcement learning. In *Proceedings of International Conference on Learning Representations*.

Bender, G., Kindermans, P.-J., Zoph, B., Vasudevan, V., and Le, Q. (2018). Understanding and simplifying one-shot architecture search. In *Proceedings of International Conference on Machine Learning*, pages 550–559.

Chen, X., Wang, R., Cheng, M., Tang, X., and Hsieh, C.-J. (2021a). DrNAS: Dirichlet neural architecture search. In *Proceedings of International Conference on Learning Representations*.

Chen, X., Xie, L., Wu, J., and Tian, Q. (2019). Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of IEEE/CVF International Conference on Computer Vision*, pages 1294–1303.

Chen, X., Xie, L., Wu, J., and Tian, Q. (2021b). Progressive darts: Bridging the optimization gap for nas in the wild. *International Journal of Computer Vision*, 129:638–655.

Chu, X., Wang, X., Zhang, B., Lu, S., Wei, X., and Yan, J. (2021). DARTS-: Robustly stepping out of performance collapse without indicators. In *Proceedings of International Conference on Learning Representations*.

Chu, X., Zhou, T., Zhang, B., and Li, J. (2020). Fair DARTS: Eliminating unfair advantages in differentiable architecture search. In *Proceedings of European Conference on Computer Vision*, pages 465–480. Springer.

Dong, X. and Yang, Y. (2019a). One-shot neural architecture search via self-evaluated template network. In *Proceedings of IEEE/CVF International Conference on Computer Vision*, pages 3680–3689.

Dong, X. and Yang, Y. (2019b). Searching for a robust neural architecture in four gpu hours. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1761–1770.

Dong, X. and Yang, Y. (2020). Nas-bench-201: Extending the scope of reproducible neural architecture search. In *Proceedings of International Conference on Learning Representations*.

Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21.

Falkner, S., Klein, A., and Hutter, F. (2018). BOHB: Robust and efficient hyperparameter optimization at scale. In

Proceedings of International Conference on Machine Learning, pages 1437–1446.

Feng, T.-C. and Wang, S.-D. (2024). VP-DARTS: Validated pruning differentiable architecture search. In *Proceedings of International Conference on Agents and Artificial Intelligence*, pages 47–57.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 770–778.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

Kyriakides, G. and Margaritis, K. (2020). An introduction to neural architecture search for convolutional networks. *arXiv:2005.11074*.

Li, G., Qian, G., Delgadillo, I. C., Müller, M., Thabet, A., and Ghanem, B. (2020). Sgas: Sequential greedy architecture search. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1617–1627.

Li, L. and Talwalkar, A. (2020). Random search and reproducibility for neural architecture search. In *Proceedings of Uncertainty in Artificial Intelligence Conference*, pages 367–377.

Liang, H., Zhang, S., Sun, J., He, X., Huang, W., Zhuang, K., and Li, Z. (2019). Darts+: Improved differentiable architecture search with early stopping. *arXiv:1909.06035*.

Liu, H., Simonyan, K., and Yang, Y. (2019). DARTS: Differentiable architecture search. In *Proceedings of International Conference on Learning Representations*.

Mellor, J., Turner, J., Storkey, A., and Crowley, E. J. (2021). Neural architecture search without training. In *Proceedings of International Conference on Machine Learning*, pages 7588–7598.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. (2018). Efficient neural architecture search via parameters sharing. In *Proceedings of International Conference on Machine Learning*, pages 4095–4104.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2019). Regularized evolution for image classifier architecture search. *Proceedings of AAAI Conference on Artificial Intelligence*, 33(01):4780–4789.

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., and Kurakin, A. (2017). Large-scale evolution of image classifiers. In *Proceedings of International Conference on Machine Learning*, pages 2902–2911.

Ren, P., Xiao, Y., Chang, X., Huang, P.-y., Li, Z., Chen, X., and Wang, X. (2021). A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys*, 54(4):1–34.

Saxena, S. and Verbeek, J. (2016). Convolutional neural fabrics. In *Proceedings of Advances in Neural Information Processing Systems*.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *Proceedings of International Conference on Learning Representations*.

Wang, R., Cheng, M., Chen, X., Tang, X., and Hsieh, C.-J. (2021). Rethinking architecture selection in differentiable NAS. In *Proceedings of International Conference on Learning Representations*.

White, C., Safari, M., Sukthanker, R., Ru, B., Elsken, T., Zela, A., Dey, D., and Hutter, F. (2023). Neural architecture search: Insights from 1000 papers. *arXiv:2301.08727*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256.

Wu, M.-T., Lin, H.-I., and Tsai, C.-W. (2022). A training-free genetic neural architecture search. In *Proceedings of ACM International Conference on Intelligent Computing and Its Emerging Applications*, page 65–70.

Wu, M.-T., Lin, H.-I., and Tsai, C.-W. (2024). A training-free neural architecture search algorithm based on search economics. *IEEE Transactions on Evolutionary Computation*, 28(2):445–459.

Xie, L., Chen, X., Bi, K., Wei, L., Xu, Y., Wang, L., Chen, Z., Xiao, A., Chang, J., Zhang, X., and Tian, Q. (2021). Weight-sharing neural architecture search: A battle to shrink the optimization gap. *ACM Computing Surveys*, 54(9):1–37.

Xie, L. and Yuille, A. (2017). Genetic cnn. In *Proceedings of IEEE/CVF International Conference on Computer Vision*, pages 1388–1397.

Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G.-J., Tian, Q., and Xiong, H. (2020). PC-DARTS: Partial channel connections for memory-efficient architecture search. In *Proceedings of International Conference on Learning Representations*.

Ye, P., Li, B., Li, Y., Chen, T., Fan, J., and Ouyang, W. (2022). β-DARTS: Beta-decay regularization for differentiable architecture search. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10864–10873.

Zela, A., Elsken, T., Saikia, T., Marrakchi, Y., Brox, T., and Hutter, F. (2020). Understanding and robustifying differentiable architecture search. In *Proceedings of International Conference on Learning Representations*.

Zoph, B. and Le, Q. (2017). Neural architecture search with reinforcement learning. In *Proceedings of International Conference on Learning Representations*.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8697–8710.