Dependency-aware Differentiable Neural Architecture Search

Buang Zhang[1], Xinle Wu[2], Hao Miao[2], Chenjuan Guo[1], and Bin Yang[1(✉)]

[1] School of Data Science and Engineering, East China Normal University, China
buazhang@stu.ecnu.edu.cn, {byang,cjguo}@dase.ecnu.edu.cn
[2] Department of Computer Science, Aalborg University, Denmark
{xinlewu,haom}@cs.aau.dk

Abstract. Neural architecture search (NAS) reduces the burden of manual design by automatically building neural network architectures, among which differential NAS approaches such as DARTS, have gained popularity for the search efficiency. Despite achieving promising performance, the DARTS series methods still suffer two issues: 1) It does not *explicitly* establish dependencies between edges, potentially leading to suboptimal performance. 2) The high degree of parameter sharing results in inaccurate performance evaluations of subnets. To tackle these issues, we propose to model dependencies *explicitly* between different edges to construct a high-performance architecture distribution. Specifically, we model the architecture distribution in DARTS as a multivariate normal distribution with learnable mean vector and correlation matrix, representing the base architecture weights of each edge and the dependencies between different edges, respectively. Then, we sample architecture weights from this distribution and alternately train these learnable parameters and network weights by gradient descent. With the learned dependencies, we prune the search space dynamically to alleviate the inaccurate evaluation by only sharing weights among high-performance architectures. Besides, we identify good motifs by analyzing the learned dependencies, which guide human experts to manually design high-performance neural architectures. Extensive experiments and competitive results on multiple NAS Benchmarks demonstrate the effectiveness of our method.

Keywords: Neural Architecture Search (NAS) · Differentiable NAS · rchitecture distribution · Dependency-aware modeling

1 Introduction

As part of Automated Machine Learning (AutoML), Neural Architecture Search (NAS) has been widely used for various tasks, ranging from computer vision to time series analytics [3,12,26,27,33,53], due to its capability to quickly search for high-performance neural architectures tailored to specific tasks [9, 16]. In the early works of NAS, researchers often employ reinforcement learning, evolutionary algorithms or Bayesian optimization to find optimal architectures [4, 46, 70]. Although these methods can discover decent architectures, they require significant computational costs due to training the architecture from scratch for evaluation. Therefore, to further enhance the efficiency

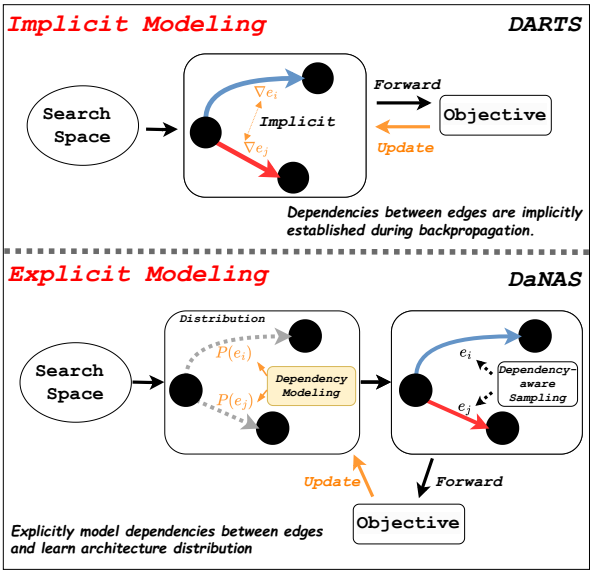


Fig. 1: Comparison between *explicit* and *implicit* dependency modeling. The upper part shows that the algorithm updates the architecture weights for each edge based on a given evaluation strategy, where dependency was *implicitly* modeled during optimization. The lower part shows that our proposed algorithm *explicitly* models dependencies between edges.

of architecture search, the weight-sharing-based search approach, also known as one-shot NAS [31], has gained popularity among researchers. Specifically, these approaches build a supernet that encompasses all possible architecture in the search space. After training the supernet once, the weights can be shared across all subnets, eliminating the need for individual weight training for each subnet and reducing the tremendous resource consumption.

Among various NAS algorithms, Differentiable Architecture Search (DARTS) [36, 51, 64] stands out as a pioneering work that transforms discrete operation selection into the weighted sum of all operations. It employs a bi-level optimization approach to alternately update network weights and architecture weights during gradient descent, reducing resource consumption while still being able to search for competitive architectures.

However, these works ignore the dependency modeling between edges. As shown in Fig. 1, when updating the architecture weights for each edge, DARTS computes the loss based on current network weights and architecture weights. Then, it updates the architecture weights for each edge to achieve better optimization. Throughout this process, dependencies between edges are only established *implicitly* during backpropagation. However, we argue that this *implicitly* modeling is insufficient to fully capture the dependencies between edges. Consequently, this inadequacy may lead to the exploration of suboptimal architectures and susceptibility to local optima. As demonstrated in [32, 54], performing *implicit* and *explicit* modeling simultaneously is advantageous

for sufficiently exploring the hidden information within the data. As the first study on edge dependencies, ITNAS [40] employs a sequential decision process to leverage the dependencies of operation choices, but it only establishes dependencies within certain parts. This sequential decision process is susceptible to the issue of error accumulation and does not model the global decision-making process.

In this paper, we propose Dependency-aware Differentiable Neural Architecture Search (DaNAS), which *explicitly* models the edge dependencies in the forward pass of training. In pursuit of this goal, we investigate the dependencies between edges within a trainable architectural distribution, which encompasses high-performing architectures. Specifically, we model the architecture distribution as a multivariate normal distribution with learnable mean vector and correlation matrix, where the mean vector represents the base architecture weights of each edge, and the correlation matrix represents the dependencies between different edges. At each optimization step, we sample the architecture weights from the distribution and train the supernet via gradient descent. Since the architecture weights of the supernet are sampled from the architecture distribution and are directly affected by the learnable correlation matrix, we are able to *explicitly* model edge dependencies during the forward pass of supernet training. In contrast to DARTS, which overlooks edge dependencies in the forward pass, our approach enforces *explicit* constraints on architecture weights. This strategy prevents the exploration of suboptimal regions within the search space, ultimately enhancing the efficacy of differentiable algorithms. With the learned dependency matrix, we dynamically prune the search space during training by a dependency-based indicator, alleviating the issue of undesired co-adaptation that arises from the high degree of weight sharing [69]. Besides, we present an analysis of the relationship between the *explicitly* defined interdependencies among edges and architectural performance. The learned dependency matrix serves as a discerning tool for uncovering *good* motifs, which provide guidance for human experts to manually design high-performance neural architectures.

In summary, our contributions can be summarized as follows.

- We propose to *explicitly* model the edge dependencies in differentiable architecture search, which is achieved by learning a well-performing architecture distribution with learnable base architecture weights and dependency matrix. We show that *explicit* modeling of edge dependencies helps capture the interactions of edges and find high-performance architectures.
- We propose a heuristic indicator to dynamically prune the search space during training to alleviate the high degree of weight sharing.
- With the learned dependencies matrix, we identify competitive motifs to help human experts to design high-performance architecture.
- Experiments on multiple benchmark datasets demonstrate the effectiveness of the proposed method.

2 Related Work

2.1 Neural Architecture Search

Early neural architecture search methods were primarily based on reinforcement learning (RL), where the agent network is trained to select better architectures. RL-based

methods [1, 71] require evaluating the test accuracy of each candidate network to determine the reward value, which necessitates training each candidate network from scratch. This approach becomes impractical for large-scale datasets like ImageNet. To address this challenge, weight-sharing NAS methods were introduced [21, 43, 65] by sharing weights among different candidate architectures, allowing for more efficient architecture search. In the branch of weight-sharing NAS, [38, 52, 59, 66] present a gradient-based approach. In these methods, both the architecture parameters and weight parameters are jointly trained. The ultimate architecture is determined by the learned architecture parameters after training.

While weight-sharing based methods have brought significant improvements in efficiency, it also introduces certain drawbacks. Research studies like RobustDARTS [64] and StacDARTS [17] have shown that the DARTS [38] algorithm tends to converge towards sharp local minima, resulting in a drastic decline in model performance. To tackle this issue, the Hessian matrix was incorporated as an early stopping indicator and applied L2 regularization. StacDARTS [17] focuses on directly optimizing the vicinity of the architecture weights to achieve a smoother curvature of validation loss. These approaches aim to alleviate the problem of sharp local minima and mitigate their adverse effects on architecture performance. From the perspective of decoding methods, DARTS-PT [51] uncovers the limitations of using architecture weights in DARTS to measure the importance of candidate operations for the supernet. Instead, the authors suggest a more comprehensive approach that considers the contribution of candidate operations to the search network. This alternative perturbation-based architecture selection method offers a new perspective and a potential solution to the architecture selection process.

2.2 Architecture Distribution Learning

Network Architecture Generator Optimization (NAGO) framework [48], proposed to shift the focus of NAS from optimizing individual architectures to optimizing architecture distribution. It leverages multi-fidelity and multi-objective Bayesian optimization (BO) to search for the optimal hyperparameters of the random network model. GraphPNAS [29] introduces a deep graph generation model based on GNN. This GNN model continuously learns the distribution of good architectures through reinforcement learning to generate good architectures. By training the architecture generator using constraint labels and structural samples, RS-NAG [25] proposes a constraint-specified generator and learns the mapping relationship between constraints and architecture distribution.

Although they all try to get a generator that captures the distribution of the network under certain conditions, all of them are time-consuming or depend on a specific search space and do not easily generalize to other settings.

2.3 Search Space Shrinkage

NSE [11] points out that while NAS algorithms have shown performance improvements on many tasks with the enhancement of search space quality and scale, some NAS algorithms face challenges when applied to large-scale search spaces. Therefore, they

iteratively train the supernet and prune the search space, removing operations if they fall below a certain threshold on their designed fitness metric.

In weight-sharing-based algorithms, training a satisfactory supernet is challenging, and the designed search space often contains a vast number of possible architectures. This places a significant burden on the performance evaluation of the supernet. Even when using unique sampling strategies to attempt to ensure fairness in evaluating the supernet for all architectures, poor architecture can still disrupt the supernet's performance evaluation [61].

Some algorithms have demonstrated that on effectively reduced search spaces, these weight-sharing-based NAS algorithms can achieve better results [8, 21, 24, 62]. For instance, P-DARTS [6] uses learnable operation weights as indicators to remove low-weight operations during the model search process. GreedyNASv2 [24] designs a filter using positive and unlabeled (PU) learning paradigm to weed out incorrect operation choices and conducts a more greedy search in the reduced space. In this paper, we design an indicator for pruning from dependency instead of the architecture itself.

3 Proposed Method

3.1 Preliminaries

Differential Architecture Search approaches utilize a continuous relaxation technique to effectively explore the discrete search space for architecture. Initially, within the DARTS search space, every edge $e^{(i,j)}$ is composed of a multitude of candidate operations, denoted as o, with each operation o being assigned a continuous architecture weights $\alpha_o^{(i,j)} \in [0,1]$. As the supernet training, edge $e^{(i,j)}$ dynamically incorporates a combination of all candidate operations, where the contribution of each operation o is weighted by its corresponding $\alpha_o^{(i,j)}$ value processed by a softmax function, defined in Eq. 1.

$$\begin{aligned} O\left(x\right)^{(i,j)} &= \sum_{o \in O} \bar{\alpha}_o^{(i,j)} o\left(x\right) \\ \bar{\alpha}_o^{(i,j)} &= \frac{\exp\left(\alpha_o^{i,j}\right)}{\sum_{o' \in \mathcal{O}} \exp\left(\alpha_{o'}^{i,j}\right)} \end{aligned} \tag{1}$$

By employing this relaxation approach, the discrete sampling of architecture is transformed into a continuous parameter update. This allows efficient updates of both the architecture weights α and the network weights w. The objective of the search is to select the α corresponding to the best-performing architecture. Based on this, we can express the update process in DARTS as a bi-level optimization :

$$\begin{aligned} &\min_{\alpha \in \Omega} \mathcal{L}_{val}\left(w^*(\alpha), \alpha\right) \\ &\text{s.t. } w^*(\alpha) = \operatorname{argmin}_w \ \mathcal{L}_{\text{train}}\left(w, \alpha\right) \end{aligned} \tag{2}$$

where Ω represents the given search space. This shows that in the DARTS algorithm, we calculate the gradient through loss to continuously optimize the architectural parameter α to find an optimal architecture in the search space, and there is no *explicit*

constraint on α. The outer-level optimization focuses on optimizing the architecture weights α, while the inner-level optimization targets the network weights w. To mitigate the impact of the inner-level optimization on search efficiency, DARTS employs the following approximate approach:

$$\begin{aligned} &\nabla_\alpha \mathcal{L}_{val}\left(w^*(\alpha), \alpha\right) \\ \approx &\nabla_\alpha \mathcal{L}_{val}\left(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha\right) \end{aligned} \tag{3}$$

3.2 Modeling Operation Dependency

In this section, we will provide an intuitive explanation of why a multivariate normal distribution can effectively establish dependencies between edges. Following that, we will provide a detailed overview of the methods presented in our paper.

To establish the dependencies between edges in the differentiable NAS algorithm, we learn a multivariate normal distribution and exploit the constraints between random variables in it. For a given multivariate normal distribution with $\mathbf{T}$ random variables $\{\mathbf{X}_i\}_{i\in\mathbf{T}}$, it is represented by the mean vector μ and the correlation matrix Σ :

$$\begin{aligned} \mu(t) &= \mathbb{E}(X_t), & t \in T \\ \Sigma(t_1, t_2) &= \kappa(X_{t_1}, X_{t_2}), & t_1, t_2 \in T \end{aligned} \tag{4}$$

where Σ is a symmetric semi-positive definite matrix and κ measures the dependence between two random variables.

The overview of the proposed mothed is shown in Fig. 2. We treat the architecture weights of each edge in the search space as random variables in a multivariate normal distribution. Without loss of generality, for a given search space Ω, we denote the i-th edge (or block) as $e_i, i \in [1, m]$, and we can choose k_i operations for it. Therefore, random variable set $\alpha = \{\alpha_i\}_{i\in\mathbf{T}}$ indicate architecture weights of all edges in the search space, where $|\mathbf{T}| = m$ represents the number of edges, $\alpha_i \in \mathbb{R}^{k_i}$ represents the weight of the each operator on edge. In the distribution, the dependency matrix $\Sigma \in \mathbb{R}^{|\mathbf{T}|\cdot|\mathbf{T}|}$ represents the dependencies between edges. Obviously, the dependency matrix Σ is symmetric, which is an easily satisfied condition. Additionally, in the mean vector $\mu \in \mathbb{R}^{|\mathbf{T}|}$, $\mu_i \in \mathbb{R}^{k_i}$ represents the i-th edge in the given search space. We can use μ and Σ to model the architecture distribution with high performance.

In order to elucidate how the relationship between edges is established, we apply the conditioning property [49] of the multivariate normal distribution, which is an important property exploited by the Gaussian Process(GP) [13, 41].

In the multivariate normal distribution established for a given search space, we consider any two non-empty sets $\mathbf{T}_u = \{i|\alpha_i \in \alpha\}$ and $\mathbf{T}_v = \{j|\alpha_j \in \alpha\}$, $\mathbf{T}_u$ and $\mathbf{T}_v$ satisfy $\mathbf{T}_u \cap \mathbf{T}_v = \emptyset$. Then $\alpha_{\mathbf{T}_u}$ and $\alpha_{\mathbf{T}_v}$ represent two disjoint subsets in α. If the dependency between edges can be explicitly established during the search process, the architecture weights of each edge will be affected by other edges. According to the conditioning property [49], we can get the relationship between $\mathbf{T}_u$ and $\mathbf{T}_v$:

$$\begin{aligned} \alpha_{\mathbf{T}_u}|\mathbf{T}_u, \mathbf{T}_v, \alpha_{\mathbf{T}_v} \sim \mathcal{N}(&\mu_{\mathbf{T}_u} + \Sigma_{\mathbf{T}_u,\mathbf{T}_v}\Sigma^{-1}_{\mathbf{T}_v,\mathbf{T}_v}\left(\alpha_{\mathbf{T}_v} - \mu_{\mathbf{T}_v}\right), \\ &\Sigma_{\mathbf{T}_u,\mathbf{T}_u} - \Sigma_{\mathbf{T}_u,\mathbf{T}_v}\Sigma^{-1}_{\mathbf{T}_v,\mathbf{T}_v}\Sigma_{\mathbf{T}_v,\mathbf{T}_u}) \end{aligned} \tag{5}$$

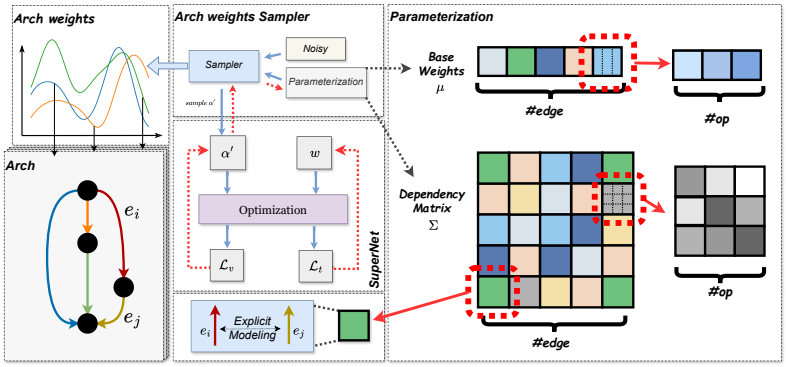


Fig. 2: Overview of proposed method. The Sampler parameters include base architecture weights μ and the dependency matrix Σ. The dependency matrix Σ models the dependencies between edges *explicitly*. The Sampler is responsible for sampling architecture (*Arch*) weights α' from the architecture distribution. Then feed it into the optimization process to calculate the training loss $\mathcal{L}_t$ and validation loss $\mathcal{L}_v$. The differentiable sampling design allows us to train the framework by gradient descent.

We regard the architecture weights α_i of each edge in the search space as a random variable to establish a multivariate normal distribution, with a learnable weight vector μ and a dependency matrix Σ. During the forward phase of training, the architecture weights α is sampled from this multivariate normal distribution. Eq. 5 proves that the distribution of each value in α will affect each other according to the learned μ and the dependency matrix Σ. This ensures that during the search we take into account dependencies between edges. In this way, α during the search process has additional constraints provided by the dependencies to guide exploration in the search space.

Therefore, our optimization goal is:

$$\begin{aligned} &\min \mathbb{E}_{\tilde{\alpha}\sim\mathcal{P}(\mu,\Sigma)}[\mathcal{L}_{val}(w^*(\tilde{\alpha}),\tilde{\alpha})] \\ &\text{s.t. } w^*(\tilde{\alpha}) = \operatorname{argmin}_w \mathcal{L}_{\text{train}}(w,\tilde{\alpha}) \end{aligned} \tag{6}$$

Compared to Eq.(2), DARTS directly models and learns the architecture weights themselves, while our method models the architecture distribution, and architecture weights are sampled from this distribution, taking into account dependencies between edges. To achieve this objective, we need to address two issues: 1) How to solve non-differentiable problems caused by the sampling process. 2) How to maintain the symmetric semi-positive definiteness of the dependency matrix Σ in the distribution during the training process.

For issue 1, we extended the reparameterization technique under a univariate normal distribution to differentiate the base architecture weights μ and correlation matrix Σ. Specifically, for the expectation of any function $f(z)$, e.g. supernet, under a univariate normal distribution $\mathcal{N}(\mu,\sigma^2)$, we can rewrite it as $\mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}[f(\mu+\sigma\epsilon)]$.

In the multivariate case, we sample for any mean vector μ and dependency matrix Σ by performing the following operations:

1. Compute the Cholesky factor (i.e., matrix square root) L of the dependency matrix Σ, such that $L^T L = \Sigma$, where L is a lower triangular matrix [19].
2. Sample a random factor $\epsilon \sim \mathcal{N}(0, I)$ from the standard normal distribution.
3. Calculate the sampled architecture weights $\alpha' \sim \mathcal{N}(\mu, \Sigma)$:

$$\alpha' = \mu + L\epsilon \tag{7}$$

4. Optimize learnable parameters in distribution and network weights alternately by gradient descent.

This reparameterization technique transfers randomness to ϵ to ensure the differentiability of the sampling process, allowing training of the distribution's parameters by gradient descent.

Furthermore, we can prove that each sampled α' possesses the desired properties, namely that the expectation of α is equal to the weight vector μ:

$$\begin{aligned} \mathbb{E}[\alpha'] &= \mu + L\mathbb{E}[\epsilon] \\ &= \mu \end{aligned} \tag{8}$$

and each operation choice in α' satisfies the constraints of our dependency matrix:

$$\begin{aligned} \Sigma_{\alpha'} &= \mathbb{E}[(\mu + L\epsilon)(\mu + L\epsilon)^T] - \mu\mu^T \\ &= \mathbb{E}[L\epsilon\epsilon^T L^T] = LIL^T \\ &= \Sigma \end{aligned} \tag{9}$$

where $\mathbb{E}[\epsilon\epsilon^T]$ is :

$$\mathbb{E}[\epsilon\epsilon^T]_{i,j} = \begin{cases} \mathbb{E}[\epsilon_i]\mathbb{E}[\epsilon_j] = 0 & , i \neq j \\ \mathbb{E}[\epsilon_i^2] = \mathbb{E}[\epsilon_j^2] = 1 & , i = j \end{cases} \tag{10}$$

Regarding issue 2, maintaining the positive definiteness of Σ during the training process is challenging. Therefore, we parameterize and learn the Cholesky factor L instead of Σ itself. By doing this, to ensure the positive definiteness of Σ according to the properties of Cholesky decomposition [19], we only need to maintain that the Cholesky factor L is a lower triangular matrix with positive diagonal entries. Ultimately, this leads to $\Sigma = LL^T$.

3.3 Dependency-Aware Search Space Pruning

To facilitate effective search space pruning during the algorithm search process, we propose a new heuristic metric based on the parameterized architecture distribution. In Eq.(8), it is clear that, under the assumption of weight-sharing based methods, the greater the weight of an operation, the more likely it is to appear in the final architecture. Therefore, we define a Selection Score (SS) for each operation choice, where $(SS_i^{(t)}) = \mu_i^{(t)}, i \in [1, |T|]$, $\mu_i^{(t)} \in \mathbb{R}^{k_i}$ represents the weight of operations on the i-th edge at the t-th training iteration. Furthermore, inspired by the fact that good architectures always contain combinations of multiple types of operations, commonly referred

Algorithm 1 Training Pipeline

Require: Parameterized distribution with base architecture weight μ and Cholesky factor of dependency matrix L, SuperNet with network weights w, maximum training iteration count M, training dataset D_{tr}, validation dataset D_{val}, warm-up step ws, pruning frequency f.

for $t = 1, ..., M$ **do**
  Sample noise vector ϵ from $\mathcal{N}(0, I)$
  Sample architecture parameters α' using Eq. (7)
  Update network weights w using gradient descent on $\mathcal{L}(w, \alpha', D_{tr})$
  if $i > ws$ **then**
    Update base architecture weight μ using gradient descent on $\mathcal{L}(w, \alpha', D_{val})$
    Update Cholesky factor of dependency matrix L using gradient descent on $\mathcal{L}(w, \alpha', D_{val})$
  end if
  if $t > ws$ and $t \mod f = 0$ **then**
    Calculate IS using Eq. (11)
    Calculate $Scores$ as $\mu + IS$
    if NOT A SINGLE ARCHITECTURE **then**
      Delete operator with minimum $Scores$
    end if
  end if
end for

to as motifs, and the components of these good motifs are highly correlated and therefore frequently occur together, we empirically demonstrate this correlation in the next section. Therefore, we define the Interaction Score (IS), where:

$$IS_i^{(t)} = -2(LL^T)_{i,i}^{(t)} + \sum_{j=1..|T|} (LL^T)_{i,j}^{(t)}, i \in [1, |T|] \tag{11}$$

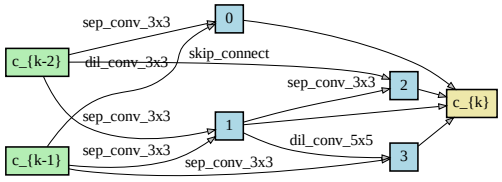
$IS_i^{(t)}$ represents the interaction strength of the i-th edge at the t-th iteration with other choices. Finally, we sum SS and IS pointwise to obtain a dependency-aware pruning indicators for each operation. We propose a search space pruning pipeline, described in Algorithm 1.

4 Experiment

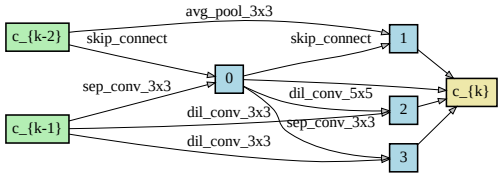
In this section, we conduct experiments on multiple NAS benchmarks to validate the effectiveness of the proposed method and compare with other state-of-the-art algorithms.

4.1 Results on NAS Benchmark

NAS-Bench-201 [15] is a tabular cell-based search space that consists of 15,625 possible architectures. Each architecture is constructed using a fixed cell-based design, where the cell comprises four nodes and six different operations. These architectures share a common outer-skeleton and are created by interleaving searched cells with residual downsampling blocks. The benchmark provides comprehensive information on the



(a) Normal cell



(b) Reduction cell

Fig. 3: Illustration of the searched normal and reduction cells by DaNAS.

learning curve and final performance of the architectures on CIFAR-10, CIFAR-100, and ImageNet datasets.

First, we train the framework using the method described in Section 3. Once training is complete, we sample several architectures and rank their performance using the weights of the trained supernet following the common settings [67]. Finally, we access *NAS-201* to obtain the final performance of the networks, thereby avoiding the need for training from scratch. Table 2 presents the performance of the proposed method on the NAS-Bench-201 search space compared to other SOTA methods. Our approach significantly surpasses the 1st and 2nd order DARTS baselines. And compared to the SOTA algorithm we achieved better performance, demonstrating the effectiveness of the algorithm. The normal and reduction cells searched by DaNAS are illustrated in Fig. 3.

4.2 Results on Open-domain Search

The common DARTS search space is widely used for evaluating different NAS methods. This search space comprises normal cells and reduction cells, each containing 4 intermediate nodes with 14 edges. Each edge is associated with 8 candidate operations. To maintain consistency with the original DARTS, our method only introduces dependencies modeling without altering any of the search settings on the DARTS search space. For evaluation purposes, we follow the same protocols as DARTS [36] for CIFAR-10 datasets, P-DARTS [7] and PC-DARTS [58] for ImageNet-1K dataset. Firstly, we performed architecture search on a proxy model to find the optimal architecture, and then we used the discovered architecture for further evaluation. It is worth noting that we only used first-order gradient estimates. The results are presented in Table 1, compared to advanced NAS algorithms, we achieve competitive and better results.

Table 1: Comparison between ours method and state-of-the-art methods on CIFAR-10. The searching cost is measured as GPU days.

Architecture	Error (%)	Params (M)	Cost(days)
DenseNet-BC [22]	3.46	25.6	-
NASNet-A [45]	2.65	3.3	1800
AmoebaNet-B [50]	2.55	2.8	3150
Hier-Evolution [35]	3.75	15.7	300
PNAS [34]	3.41	3.2	225
ENAS [44]	2.89	4.6	0.50
NAONet-WS [39]	3.53	3.1	0.40
SGAS (Cri.2) [28]	2.52	4.1	0.25
GDAS [14]	2.93	3.4	0.21
SNAS [56]	2.85±0.02	2.9	1.5
DARTS (1st) [37]	3.00±0.14	3.3	0.4
DARTS (2nd) [37]	2.76 ±0.09	3.3	1.0
P-DARTS [6]	2.81±0.14	3.5	0.3
R-DARTS(L2) [64]	2.95±0.21	-	1.6
PC-DARTS [57]	2.57±0.07	3.6	0.1
SDARTS-ADV [5]	2.61 ±0.02	3.3	1.3
DARTS+PT [51]	2.61±0.08	3.0	0.8
DARTS- [10]	2.59±0.08	3.5	0.4
DaNAS	2.53±0.06	3.7	0.30
DaNAS(*best*)	2.39	3.7	0.30

To demonstrate the transferability, we also conducted experiments on the large-scale ImageNet dataset. We trained the architecture for 250 epochs using a batch size of 1024. We utilized an SGD optimizer with a momentum of 0.9, an initial learning rate of 0.5, a weight decay of 0.0003. And other common setting following [40, 58]. As shown in Table 3, compared to other methods, DaNAS achieved competitive results

4.3 Effect of Dependency Modeling

To demonstrate the *explicit* modeling advantages and performance improvements of our approach. We designed the following experiments. We change the strength of dependencies between operations to see whether capturing more dependencies is beneficial to performance. Specifically, we vary the dependency matrix Σ in the model used to capture the dependencies between edges: 1) *FULL*: Use the complete dependency matrix for learning. 2) *MASK* 30%: Randomly mask out 30% of the values in the dependency matrix Σ. 3) *MASK* 60%: Randomly mask out 60% of the values in the dependency matrix Σ. 4) Diagonal: Does not learn any dependencies between edges. It is worth noting that this approach is essentially equivalent to vanilla DARTS [36].

We run these different ablation models multiple times on the NAS-Bench-201 search space and report the average results. Fig. 4 shows the accuracy of these four ablation models on the test set on CIFAR-10, CIFAR-100, and ImageNet16-120 as the search time runs.

We can observe that *FULL* achieves the best and most stable results. For *MASK* 30% and *MASK* 60%, compared to *FULL*, there is a missing part of modeling the de-

Table 2: Accuracy comparison on NAS-Bench-201 benchmark. Note that the proposed method only searches on CIFAR-10 dataset, but can robustly achieve new SOTA on CIFAR-10, CIFAR-100 and ImageNet16-120 datasets. Averaged on 4 independent architectures.

Methods	CIFAR-10	CIFAR-100	ImageNet16-120
DARTS(2nd) [37]	54.30±0.00	15.61±0.00	16.32±0.00
GDAS [14]	93.51±0.13	70.61±0.26	41.84±0.90
SNAS [56]	92.77±0.83	69.34±1.98	43.16±2.64
DSNAS [20]	93.08±0.13	31.01±16.38	41.07±0.09
PC-DARTS [57]	93.41±0.30	67.48±0.89	41.31±0.22
iDARTS [68]	93.58±0.32	70.83±0.48	40.89±0.68
DARTS- [10]	93.80±0.40	71.53±1.51	45.12±0.82
CDARTS [63]	94.02±0.31	71.92±1.30	45.51±0.72
DaNAS	**94.33±0.03**	**73.21±0.19**	**46.35±0.13**
Optimal	94.37	73.51	47.31

pendencies between edges in the search space. This results in the inability to provide sufficient constraints to explore the search space, making it easy to fall into local optimal solutions. Diagonal does not establish any dependencies between edges, which causes Diagonal to have a common performance collapse problem as the running time increases [65]. This proves that the dependencies between the proposed operations are advantageous for the stability of the algorithm and the improvement of performance.

4.4 Effect of Dependency-aware Search Space Pruning

We introduce pruning indicators that consider edge dependencies, allowing us to dynamically prune search space during the training of the supernet. Other works, such as P-DARTS [7], gradually remove non-competitive operations during algorithm exploration. However, P-DARTS [7] does not take into account the dependencies between edges during pruning, leading to overlooking the combination of operations on edges for enhancing architecture performance.

We compare DaNAS and its two variants P-DaNAS and O-DaNAS to demonstrate the effectiveness of the proposed pruning strategy, where P-DaNAS adopts the pruning strategy of P-DARTS [7], which does not consider edge dependencies, while O-DaNAS does not have the pruning part, and other settings are the same as DaNAS. In Table 4, we present the impact of modeling edge dependencies and pruning the search space. P-DaNAS only considers the weights of individual edges when pruning, choosing to prune operations with smaller weights. However, this approach may lead to getting stuck in local optima as it does not account for the interaction between them. Furthermore, we can observe that incorporating dependency-aware pruning into the search space further enhances the performance of the algorithm. This proves that modeling dependency between edges and pruning the search space is effective for performance improvement.

4.5 Association between Dependency and Motif

In this subsection, we demonstrate that dependencies between edges can identify useful motifs within search space, therefore can be used as an indicator to pruning search

Table 3: Comparison between DaNAS and state-of-the-art methods on ImageNet.

Architecture	Top-1 (%)	Top-5 (%)	Params (M)	FLOPs (M)	Cost(days)
DARTS(2nd) [37]	26.7	8.7	4.7	574	4
P-DARTS [6]	24.4	7.4	4.9	557	0.3
SNAS [56]	27.3	9.2	4.3	522	1.5
PC-DARTS [57]	25.1	7.8	-	571	0.3
ProxylessNAS [2]	24.9	7.5	7.1	465	8.3
GDAS [14]	26.0	8.5	5.3	581	0.21
DARTS+PT [51]	25.5	8.0	4.6	-	0.8
PVLL-NAS [30]	25.60	8.10	5.0	556	0.20
CARS-I [60]	24.80	7.50	5.1	591	0.40
MiLeNAS [18]	24.70	7.60	5.3	584	0.30
Λ-DARTS [42]	24.60	-	5.1	-	-
GeNAS [28]	24.66	7.59	5.3	600	-
SGAS(Cri.2) [28]	24.44	7.41	5.7	621	0.25
DaNAS	**24.21**	**7.11**	5.5	607	0.30

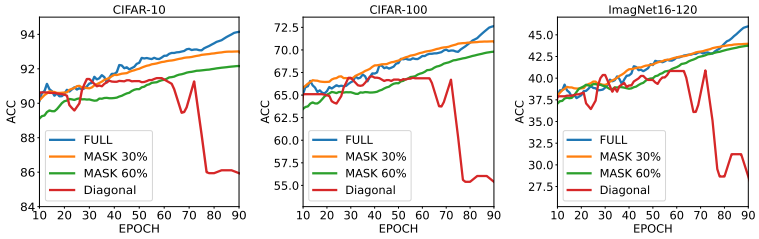
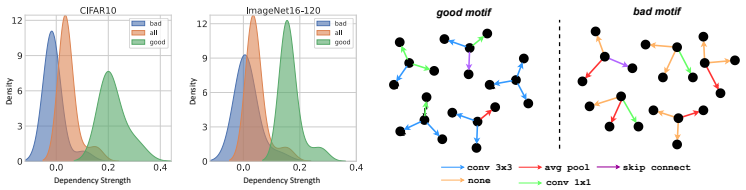


Fig. 4: Results of models with different dependency learning strengths on the NAS-Bench-201 search space.

Table 4: Ablation study on component combinations. Results are averaged over multiple runs.

Architecture	Dependency	Pruning	ACC(%)
P-DaNAS	✘	✔	92.08
O-DaNAS	✔	✘	94.25
DaNAS	✔	✔	**94.33**

space. To achieve this, for each feasible motif, we first compute the average performance across all architectures which contain at least one of it. Then, we extract the dependency strength of this motif from the learned dependency matrix. Specifically, we follow the common setup [47] to categorize motifs as either good or bad. As shown in Fig. 5(a), low-performance motifs have dependency strengths near 0, while high-performance motifs exhibit higher dependencies. This demonstrates that by establishing dependencies between edges, we can identify high-performance motifs, aiding al-



(a) Distribution of the dependency strength. **(b)** Good/Bad motifs.

Fig. 5: (a) Distribution of the dependency strength of the good/bad motif. The left figure shows statistical results on the CIFAR10 dataset, and the right figure shows statistical results on the ImageNet16-120 dataset. (b) Captured good/bad motifs in the NAS201 search space,with CIFAR-10 as the proxy task.

gorithms in exploring high-performance architectures. Additionally, based on this property, we validate the rationality behind the proposed dynamic pruning algorithm.

In Fig. 5(b), we present the good and bad motifs captured by the algorithm. Empirically, it can be observed that all motifs contain at least one convolution operation. Furthermore, in many popular NAS algorithms, we can observe a preference for the aggregation of convolution operations [21,43]. Additionally, combinations of convolution operations with different kernel sizes have also been identified. This combination of different receptive fields has led to the success of many architecture, e.g. ResNeXt [55]. Although skip operations may not provide feature extraction capabilities, their residual design facilitates network learning and convergence. Therefore, the presence of skip connections is also crucial for the success of architectures design. Another observation is the presence of *avg_ pool* operations within a motif. *avg_ pool* can be seen as convolution operations with fixed kernels. While it does not introduce non-linearity to the forward process, it still provides some local weighting capabilities and maintains fixed gradient coefficients. As a result, *avg_ pool* operations are still present in the backbones of some advanced networks [23] and this behavior can also be successfully captured by our edge dependencies.

5 Conclusion

In this paper, we introduce DaNAS, a novel approach that *explicitly* establishes dependencies between edges in the search space of neural architectures. Unlike traditional methods that rely solely on *implicit* modeling, DaNAS leverages both *implicit* and *explicit* modeling to capture more comprehensive inter-edge dependencies. Furthermore, the use of the learned dependency matrix enables dynamic pruning of the search space, addressing the issue of undesired co-adaptation. Extensive experiments on real datasets offer evidence of the effectiveness of the proposed method.

Acknowledgments

This work was partially supported by National Natural Science Foundation of China (62372179) and Independent Research Fund Denmark (8048-00038B).

References

1. Baker, B., Gupta, O., Naik, N., Raskar, R.: Designing neural network architectures using reinforcement learning. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net (2017), https://openreview.net/forum?id=S1c2cvqee
2. Cai, H., Zhu, L., Han, S.: Proxylessnas: Direct neural architecture search on target task and hardware. In: International Conference on Learning Representations (2018)
3. Campos, D., Kieu, T., Guo, C., Huang, F., Zheng, K., Yang, B., Jensen, C.S.: Unsupervised time series outlier detection with diversity-driven convolutional ensembles. Proc. VLDB Endow. **15**(3), 611–623 (2022)
4. Campos, D., Zhang, M., Yang, B., Kieu, T., Guo, C., Jensen, C.S.: Lightts: Lightweight time series classification with adaptive ensemble distillation. Proc. ACM Manag. Data **1**(2), 171:1–171:27 (2023). https://doi.org/10.1145/3589316, https://doi.org/10.1145/3589316
5. Chen, X., Hsieh, C.J.: Stabilizing differentiable architecture search via perturbation-based regularization. In: International conference on machine learning. pp. 1554–1565. PMLR (2020)
6. Chen, X., Xie, L., Wu, J., Tian, Q.: Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 1294–1303 (2019)
7. Chen, X., Xie, L., Wu, J., Tian, Q.: Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 1294–1303 (2019)
8. Chen, X., Xie, L., Wu, J., Tian, Q.: Progressive DARTS: bridging the optimization gap for NAS in the wild. Int. J. Comput. Vis. **129**(3), 638–655 (2021). https://doi.org/10.1007/S11263-020-01396-X, https://doi.org/10.1007/s11263-020-01396-x
9. Chen, Y., Yang, T., Zhang, X., Meng, G., Xiao, X., Sun, J.: Detnas: Backbone search for object detection. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada. pp. 6638–6648 (2019)
10. Chu, X., Wang, X., Zhang, B., Lu, S., Wei, X., Yan, J.: Darts-: robustly stepping out of performance collapse without indicators. arXiv preprint arXiv:2009.01027 (2020)
11. Ci, Y., Lin, C., Sun, M., Chen, B., Zhang, H., Ouyang, W.: Evolving search space for neural architecture search. In: 2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021. pp. 6639–6649. IEEE (2021). https://doi.org/10.1109/ICCV48922.2021.00659, https://doi.org/10.1109/ICCV48922.2021.00659
12. Cirstea, R.G., Yang, B., Guo, C.: Graph attention recurrent neural networks for correlated time series forecasting. In: MileTS19@KDD (2019)
13. Damianou, A., Lawrence, N.D.: Deep gaussian processes. In: Artificial intelligence and statistics. pp. 207–215. PMLR (2013)
14. Dong, X., Yang, Y.: Searching for a robust neural architecture in four gpu hours. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1761–1770 (2019)
15. Dong, X., Yang, Y.: Nas-bench-201: Extending the scope of reproducible neural architecture search. arXiv preprint arXiv:2001.00326 (2020)

16. Gong, X., Chang, S., Jiang, Y., Wang, Z.: Autogan: Neural architecture search for generative adversarial networks. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV) (Oct 2019). https://doi.org/10.1109/iccv.2019.00332
17. Guilin, L., Xing, Z., Zitong, W., Zhenguo, L., Tong, Z.: Stacnas: Towards stable and consistent optimization for differentiable neural architecture search (2019)
18. He, C., Ye, H., Shen, L., Zhang, T.: Milenas: Efficient neural architecture search via mixed-level reformulation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 11993–12002 (2020)
19. Higham, N.J.: Cholesky factorization. Wiley interdisciplinary reviews: computational statistics **1**(2), 251–254 (2009)
20. Hu, S., Xie, S., Zheng, H., Liu, C., Shi, J., Liu, X., Lin, D.: Dsnas: Direct neural architecture search without parameter retraining. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 12084–12092 (2020)
21. Hu, Y., Liang, Y., Guo, Z., Wan, R., Zhang, X., Wei, Y., Gu, Q., Sun, J.: Angle-based search space shrinking for neural architecture search. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J. (eds.) Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XIX. Lecture Notes in Computer Science, vol. 12364, pp. 119–134. Springer (2020). https://doi.org/10.1007/978-3-030-58529-7_8
22. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: CVPR (2017)
23. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
24. Huang, T., You, S., Wang, F., Qian, C., Zhang, C., Wang, X., Xu, C.: Greedynasv2: Greedier search with a greedy path filter. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022. pp. 11892–11901. IEEE (2022). https://doi.org/10.1109/CVPR52688.2022.01160, https://doi.org/10.1109/CVPR52688.2022.01160
25. Jing, K., Xu, J., Zhang, Z.: A neural architecture generator for efficient search space. Neurocomputing **486**, 189–199 (2022)
26. Kieu, T., Yang, B., Guo, C., Cirstea, R., Zhao, Y., Song, Y., Jensen, C.S.: Anomaly detection in time series with robust variational quasi-recurrent autoencoders. In: ICDE. pp. 1342–1354 (2022)
27. Kieu, T., Yang, B., Guo, C., Jensen, C.S., Zhao, Y., Huang, F., Zheng, K.: Robust and explainable autoencoders for unsupervised time series outlier detection. In: ICDE. pp. 3038–3050 (2022)
28. Li, G., Qian, G., Delgadillo, I.C., Muller, M., Thabet, A., Ghanem, B.: Sgas: Sequential greedy architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1620–1630 (2020)
29. Li, M., Liu, J.Y., Sigal, L., Liao, R.: Graphpnas: Learning distribution of good neural architectures via deep graph generative models. arXiv preprint arXiv:2211.15155 (2022)
30. Li, Y., Dong, M., Wang, Y., Xu, C.: Neural architecture search in a proxy validation loss landscape. In: International Conference on Machine Learning. pp. 5853–5862. PMLR (2020)
31. Liang, H., Zhang, S., Sun, J., He, X., Huang, W., Zhuang, K., Li, Z.: Darts+: Improved differentiable architecture search with early stopping. arXiv preprint arXiv:1909.06035 (2019)
32. Liu, A., Liu, Y., Gu, J., Qiao, Y., Dong, C.: Blind image super-resolution: A survey and beyond. IEEE Trans. Pattern Anal. Mach. Intell. **45**(5), 5461–5480 (2023). https://doi.org/10.1109/TPAMI.2022.3203009, https://doi.org/10.1109/TPAMI.2022.3203009

33. Liu, C., Chen, L., Schroff, F., Adam, H., Hua, W., Yuille, A.L., Fei-Fei, L.: Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019. pp. 82–92. Computer Vision Foundation / IEEE (2019). https://doi.org/10.1109/CVPR.2019.00017
34. Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.J., Fei-Fei, L., Yuille, A., Huang, J., Murphy, K.: Progressive neural architecture search. In: Proceedings of the European conference on computer vision (ECCV). pp. 19–34 (2018)
35. Liu, H., Simonyan, K., Vinyals, O., Fernando, C., Kavukcuoglu, K.: Hierarchical representations for efficient architecture search. arXiv preprint arXiv:1711.00436 (2017)
36. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. In: International Conference on Learning Representations (2018)
37. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055 (2018)
38. Liu, H., Simonyan, K., Yang, Y.: DARTS: differentiable architecture search. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net (2019), https://openreview.net/forum?id=S1eYHoC5FX
39. Luo, R., Tian, F., Qin, T., Chen, E., Liu, T.Y.: Neural architecture optimization. Advances in neural information processing systems **31** (2018)
40. Ma, B., Zhang, J., Xia, Y., Tao, D.: Inter-layer transition in neural architecture search. Pattern Recognit. **143**, 109697 (2023). https://doi.org/10.1016/J.PATCOG.2023.109697, https://doi.org/10.1016/j.patcog.2023.109697
41. MacKay, D.J., et al.: Introduction to gaussian processes. NATO ASI series F computer and systems sciences **168**, 133–166 (1998)
42. Movahedi, S., Adabinejad, M., Imani, A., Keshavarz, A., Dehghani, M., Shakery, A., Araabi, B.N.: $lambda$-darts: Mitigating performance collapse by harmonizing operation selection among cells. arXiv preprint arXiv:2210.07998 (2022)
43. Oymak, S., Li, M., Soltanolkotabi, M.: Generalization guarantees for neural architecture search with train-validation split. In: Meila, M., Zhang, T. (eds.) Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event. Proceedings of Machine Learning Research, vol. 139, pp. 8291–8301. PMLR (2021), http://proceedings.mlr.press/v139/oymak21a.html
44. Pham, H., Guan, M., Zoph, B., Le, Q., Dean, J.: Efficient neural architecture search via parameters sharing. In: International conference on machine learning. pp. 4095–4104. PMLR (2018)
45. Qin, X., Wang, Z.: Nasnet: A neuron attention stage-by-stage net for single image deraining. arXiv preprint arXiv:1912.03151 (2019)
46. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: Proceedings of the aaai conference on artificial intelligence. vol. 33, pp. 4780–4789 (2019)
47. Ru, B.X., Wan, X., Dong, X., Osborne, M.A.: Interpretable neural architecture search via bayesian optimisation with weisfeiler-lehman kernels. In: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net (2021), https://openreview.net/forum?id=j9Rv7qdXjd
48. Ru, R., Esperanca, P., Carlucci, F.M.: Neural architecture generator optimization. Advances in Neural Information Processing Systems **33**, 12057–12069 (2020)
49. SEEGER, M.: Gaussian processes for machine learning. International Journal of Neural Systems p. 69–106 (Apr 2004). https://doi.org/10.1142/s0129065704001899, http://dx.doi.org/10.1142/s0129065704001899

50. Shah, S.A.R., Wu, W., Lu, Q., Zhang, L., Sasidharan, S., DeMar, P., Guok, C., Macauley, J., Pouyoul, E., Kim, J., et al.: Amoebanet: An sdn-enabled network service for big data science. Journal of Network and Computer Applications **119**, 70–82 (2018)
51. Wang, R., Cheng, M., Chen, X., Tang, X., Hsieh, C.J.: Rethinking architecture selection in differentiable nas. arXiv preprint arXiv:2108.04392 (2021)
52. Wu, X., Zhang, D., Guo, C., He, C., Yang, B., Jensen, C.S.: AutoCTS: Automated correlated time series forecasting. Proc. VLDB Endow. **15**(4), 971–983 (2021). https://doi.org/10.14778/3503585.3503604, https://www.vldb.org/pvldb/vol15/p971-wu.pdf
53. Wu, X., Zhang, D., Zhang, M., Guo, C., Yang, B., Jensen, C.S.: AutoCTS+: Joint neural architecture and hyperparameter search for correlated time series forecasting. Proc. ACM Manag. Data **1**(1), 97:1–97:26 (2023). https://doi.org/10.1145/3588951, https://doi.org/10.1145/3588951
54. Xie, R., Ling, C., Wang, Y., Wang, R., Xia, F., Lin, L.: Deep feedback network for recommendation. In: Bessiere, C. (ed.) Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020. pp. 2519–2525. ijcai.org (2020). https://doi.org/10.24963/IJCAI.2020/349, https://doi.org/10.24963/ijcai.2020/349
55. Xie, S., Girshick, R.B., Dollár, P., Tu, Z., He, K.: Aggregated residual transformations for deep neural networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017. pp. 5987–5995. IEEE Computer Society (2017). https://doi.org/10.1109/CVPR.2017.634, https://doi.org/10.1109/CVPR.2017.634
56. Xie, S., Zheng, H., Liu, C., Lin, L.: Snas: stochastic neural architecture search. arXiv preprint arXiv:1812.09926 (2018)
57. Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G.J., Tian, Q., Xiong, H.: Pc-darts: Partial channel connections for memory-efficient architecture search. arXiv preprint arXiv:1907.05737 (2019)
58. Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G.J., Tian, Q., Xiong, H.: Pc-darts: Partial channel connections for memory-efficient architecture search. In: International Conference on Learning Representations (2019)
59. Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G., Tian, Q., Xiong, H.: PC-DARTS: partial channel connections for memory-efficient architecture search. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net (2020), https://openreview.net/forum?id=BJlS634tPr
60. Yang, Z., Wang, Y., Chen, X., Shi, B., Xu, C., Xu, C., Tian, Q., Xu, C.: Cars: Continuous evolution for efficient neural architecture search. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 1829–1838 (2020)
61. You, S., Huang, T., Yang, M., Wang, F., Qian, C., Zhang, C.: Greedynas: Towards fast one-shot nas with greedy supernet. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1999–2008 (2020)
62. You, S., Huang, T., Yang, M., Wang, F., Qian, C., Zhang, C.: Greedynas: Towards fast one-shot NAS with greedy supernet. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020. pp. 1996–2005. Computer Vision Foundation / IEEE (2020). https://doi.org/10.1109/CVPR42600.2020.00207
63. Yu, H., Peng, H., Huang, Y., Fu, J., Du, H., Wang, L., Ling, H.: Cyclic differentiable architecture search. IEEE Transactions on Pattern Analysis and Machine Intelligence **45**(1), 211–228 (2022)
64. Zela, A., Elsken, T., Saikia, T., Marrakchi, Y., Brox, T., Hutter, F.: Understanding and robustifying differentiable architecture search. arXiv preprint arXiv:1909.09656 (2019)

65. Zela, A., Elsken, T., Saikia, T., Marrakchi, Y., Brox, T., Hutter, F.: Understanding and robustifying differentiable architecture search. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net (2020), https://openreview.net/forum?id=H1gDNyrKDS
66. Zhang, M., Huang, W., Yang, B.: Interpreting operation selection in differentiable architecture search: A perspective from influence-directed explanations. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (eds.) NeurIPS (2022)
67. Zhang, M., Pan, S., Chang, X., Su, S., Hu, J., Haffari, G.R., Yang, B.: Balenas: Differentiable architecture search via the bayesian learning rule. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 11871–11880 (2022)
68. Zhang, M., Su, S.W., Pan, S., Chang, X., Abbasnejad, E.M., Haffari, R.: idarts: Differentiable architecture search with stochastic implicit gradients. In: International Conference on Machine Learning. pp. 12557–12566. PMLR (2021)
69. Zhao, Y., Wang, L., Tian, Y., Fonseca, R., Guo, T.: Few-shot neural architecture search. In: Meila, M., Zhang, T. (eds.) Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event. Proceedings of Machine Learning Research, vol. 139, pp. 12707–12718. PMLR (2021), http://proceedings.mlr.press/v139/zhao21d.html
70. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)
71. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018. pp. 8697–8710. Computer Vision Foundation / IEEE Computer Society (2018). https://doi.org/10.1109/CVPR.2018.00907