DANCE: Resource-Efficient Neural Architecture Search with Data-Aware and Continuous Adaptation

Maolin Wang[1], **Tianshuo Wei**[1], **Sheng Zhang**[1], **Ruocheng Guo**[2,*],
Wanyu Wang[1,*], **Shanshan Ye**[3,*], **Lixin Zou**[4], **Xuetao Wei**[5], **Xiangyu Zhao**[1]
[1]City University of Hong Kong
[2]Independent Researcher
[3]Australian Artificial Intelligence Institute, University of Technology Sydney
[4]Wuhan University
[5]Southern University of Science and Technology
rguo.asu@gmail.com, wanyuwang4-c@cityu.edu.hk, shanshan.ye@student.uts.edu.au

Abstract

Neural Architecture Search (NAS) has emerged as a powerful approach for automating neural network design. However, existing NAS methods face critical limitations in real-world deployments: architectures lack adaptability across scenarios, each deployment context requires costly separate searches, and performance consistency across diverse platforms remains challenging. We propose DANCE (Dynamic Architectures with Neural Continuous Evolution), which reformulates architecture search as a continuous evolution problem through learning distributions over architectural components. DANCE introduces three key innovations: a continuous architecture distribution enabling smooth adaptation, a unified architecture space with learned selection gates for efficient sampling, and a multi-stage training strategy for effective deployment optimization. Extensive experiments across five datasets demonstrate DANCE's effectiveness. Our method consistently outperforms state-of-the-art NAS approaches in terms of accuracy while significantly reducing search costs. Under varying computational constraints, DANCE maintains robust performance while smoothly adapting architectures to different hardware requirements. The code and appendix can be found at https://github.com/Applied-Machine-Learning-Lab/DANCE.

1 Introduction

Neural Architecture Search (NAS) has revolutionized deep neural network design by automating the architecture optimization process [Chen *et al.*, 2022]. Despite its recent success, existing NAS methods face significant challenges in effectively modeling three key aspects: architecture adaptability [Zhao, 2022], deployment efficiency [Zhu *et al.*, 2023], and performance consistency across diverse scenarios [Ren *et al.*, 2021; Zhang *et al.*, 2023]. While current approaches can address certain aspects, they often fail to comprehensively tackle all these challenges simultaneously [Cai *et al.*, 2019].

In real-world deployment scenarios, neural architectures naturally exhibit three key characteristics: (1) **Compute Constraints**: architectures need to adapt to varying computational constraints (e.g., an architecture suitable for GPU servers may be impractical on mobile devices) [Howard, 2017; Liu *et al.*, 2020; Zhao *et al.*, 2021b], (2) **Varied Requirements**: deployment scenarios vary significantly in their requirements (e.g., real-time applications demand low latency while offline tasks prioritize accuracy) [Wang *et al.*, 2019; Zhu *et al.*, 2023], and (3) **Performance Variance**: model performance patterns often differ across deployment contexts (e.g., models optimized for one domain may underperform in others) [Tan and Le, 2019; Gao *et al.*, 2023]. These observations highlight the critical need for dynamic and adaptable neural architectures that can efficiently handle diverse deployment scenarios while maintaining robust performance [Zhao, 2022].

Based on these observations, neural architecture search systems need to address three fundamental challenges that require innovative solutions: First, architecture flexibility needs to be effectively modeled. Current discrete search methods lack the ability to smoothly adapt architectures as deployment requirements change, necessitating expensive re-search for each scenario [Zoph, 2016; Zhao, 2022; Zhu *et al.*, 2023; Lin *et al.*, 2022]. Second, deployment scenarios exhibit diverse computational constraints. While recent methods have shown promise in handling specific deployment contexts [Wu *et al.*, 2019; Liu *et al.*, 2020], they struggle to maintain consistent performance across varying scenarios due to their rigid architecture designs [Gao *et al.*, 2023; Li *et al.*, 2023; Song *et al.*, 2022]. Third, architecture search naturally involves multiple competing objectives. Despite their advantages in finding optimal architectures, existing methods typically process these objectives uniformly, failing to capture the complex trade-offs between performance metrics [Elsken *et al.*, 2019; Zhao *et al.*, 2021a; Zhang *et al.*, 2023; Jin *et al.*, 2021]. These limitations of current approaches are particularly evident in practice: Pure evolutionary methods demonstrate sub-optimal

*Corresponding authors.

efficiency in large-scale deployments [Real *et al.*, 2019; Liu *et al.*, 2024], gradient-based approaches face significant scalability challenges [El Bsat *et al.*, 2017], and while one-shot methods offer improved search efficiency [DONG *et al.*, 2023; Zhaok *et al.*, 2021], they not only show limited transfer capability but also lack mechanisms to effectively capture diverse deployment requirements [Bender *et al.*, 2018; Chen *et al.*, 2022; Li *et al.*, 2023]. This creates a clear need for a unified solution that can simultaneously address architecture adaptability and deployment efficiency while maintaining consistent performance.

To address these challenges, we propose DANCE (**D**ynamic **A**rchitectures with **N**eural **C**ontinuous **E**volution), a novel framework that synergistically combines continuous architecture evolution and distribution learning through a unified optimization approach. Unlike previous methods that focus on block-level components, DANCE operates directly on feature dimensions, providing a unified perspective that bridges the gap between NAS and pruning. The continuous evolution mechanism enables smooth architecture adaptation across different deployment scenarios, while the distribution learning component efficiently captures and models deployment patterns. By integrating data-aware dynamic selection and importance-based sampling at the feature level, this unified framework effectively leverages both architectural flexibility and deployment efficiency while maintaining computational tractability through selective sampling and adaptive optimization strategies.

Our main contributions are summarized as follows:

- We propose DANCE, a novel neural architecture search framework that learns continuous distributions over architectural components. This enables efficient architecture adaptation under varying computational budgets while maintaining high performance.
- We design a unified architecture space with dynamic select gates that integrate batch-level features and layer-wise importance metrics. This mechanism enables flexible component selection and smooth architecture adaptation across different hardware constraints and deployment contexts.
- Extensive experiments on five image datasets demonstrate that DANCE achieves consistent accuracy improvements over state-of-the-art methods while significantly reducing search costs. The results validate its effectiveness for practical applications with diverse computational requirements and resource constraints.

2 Methodology

Neural architecture search (NAS) has emerged as a promising approach for automating deep neural network design. However, current NAS methods face two key challenges in real-world deployments: (1) the static architectures lack flexibility to adapt to varying computational constraints, and (2) separate expensive searches are required for each deployment scenario, leading to prohibitive costs. To address these limitations, we propose DANCE (Dynamic Architectures with Neural Continuous Evolution), which reformulates architecture search as a continuous evolution problem. Instead of finding a single fixed architecture, DANCE learns a distribution over architectural components that can be efficiently sampled and adapted to specific deployment requirements. This enables rapid architecture derivation while maintaining performance across diverse scenarios.

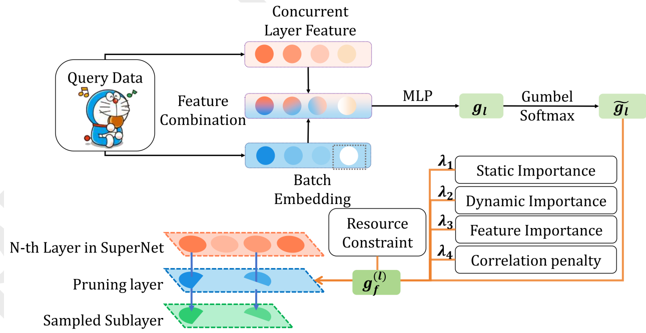

Figure 1: Our layer-wise dynamic select gate mechanism. The framework combines concurrent layer features and batch embeddings through feature combination, applies gumbel-softmax for exploration, and generates final gates through sampling under resource constraints. The gate is also guided by four importance metrics evaluating static, dynamic, feature based, and correlation significance.

2.1 Problem Formulation

Traditional NAS methods typically operate in a discrete search space, optimizing architectures under the fixed resource constraints:

$$\mathcal{A}^* = \arg\min_{\mathcal{A}} \mathcal{L}(\mathcal{A}) \text{ s.t. } \mathrm{Cost}(\mathcal{A}) \leq C \tag{1}$$

where $\mathcal{A}$ denotes a candidate architecture and C specifies the computational budget constraint. This discrete optimization approach presents several challenges. First, it requires running separate searches for different deployment scenarios, leading to substantial computational overhead. Second, the discrete nature of the search space makes it difficult to adapt architectures smoothly as requirements change. Third, there is no guarantee of consistency between the selected components across different layers, resulting in suboptimal results.

2.2 Unified Architecture Space Design

Building upon the limitations of traditional NAS methods, DANCE introduces a unified architecture space design that accommodates diverse deployment requirements while significantly enhancing search efficiency. As shown in Figure 1, for each N-th SuperNet layer, we introduce an **data-aware dynamic select gate** to dynamically generate a pruning layer for component selection.

The key insight is learning a continuous distribution over architectural components that factorizes across layers while maintaining dependencies:

$$p(\mathcal{A}|\mathcal{D}, C) = \prod_{l=1}^{L} p(g_l|\mathcal{F}_l, \mathcal{D}, C) \tag{2}$$

This formulation expresses the probability of selecting a specific architecture $\mathcal{A}$ as a product of layer-wise selection

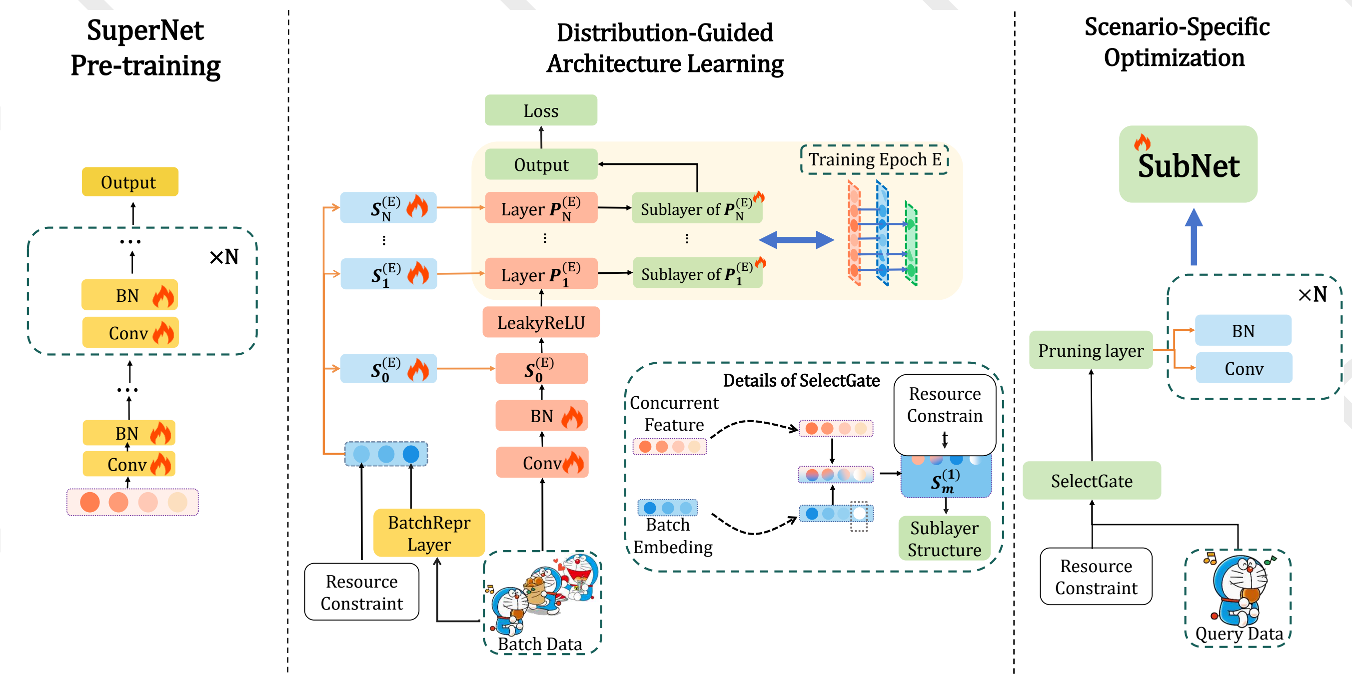

Figure 2: Overview of DANCE's three-stage training process. Stage 1 pre-trains a SuperNet with N stacked layers as backbone. Stage 2 employs distribution-guided learning where SelectGate generates architecture masks based on batch features, layer features and constraints through dual-loop optimization. Stage 3 optimizes scenario-specific SubNets by inheriting SuperNet weights and fine-tuning under deployment requirements. This enables continuous adaptation while maintaining performance across diverse scenarios.

probabilities, where each layer's selection depends on the layer features $\mathcal{F}_l$, input data batch $\mathcal{D}$, and resource constraints C_l (where C_l represents the number of parameters to retain in layer l). Specifically, we implement this through a data-aware dynamic selection mechanism:

$$g_{\mathrm{f}}^{(l)} = \mathrm{SelectGate}(\mathcal{D}, \mathcal{F}_l, C_l), \quad l \in [1, L] \tag{3}$$

For each n-th layer in the supernet, the SelectGate system combines concurrent layer feature and batch embedding through feature combination, which then passes through gumbel-softmax to generate select gate for layer selection.

This process can be formulated as:

$$\begin{aligned} \mathrm{FeatureComb}_l &= \mathrm{Combine}(\mathcal{D}, \mathcal{F}_l) \\ g_l &= \mathrm{MLP}(\mathrm{FeatureComb}_l) \end{aligned}$$

where $\mathrm{FeatureComb}_l$ is the combined feature and g_l is the final select gate output. Here MLP is a two-layer neural network with ReLU activation in between, which learns to map the combined features to selection probabilities. Given the layer-wise resource budget C_l, the selection gate mechanism employs a combination of soft and stochastic selection to determine the final gate value.

$$\begin{aligned} \epsilon &\sim \mathrm{Gumbel}(0, 1) \\ \tilde{g}_l &= \mathrm{Softmax}((g_l + \epsilon)/\tau) \\ p_l &= \mathrm{Softmax}(\tilde{g}_l \cdot \mathrm{Score}(\mathcal{A}_l)) \\ g_{\mathrm{f}}^{(l)} &= \mathrm{BernoulliSample}(p_l, C_l) \end{aligned}$$

Here, ϵ is sampled from a $Gumbel(0, 1)$ distribution to introduce randomness in the softmax operation, τ is the temperature parameter controlling the sharpness of the softmax distribution, $\tilde{g}_l$ represents the soft selection probabilities after gumbel-softmax. p_l normalizes the importance-weighted probabilities through softmax to ensure valid sampling probabilities. The BernoulliSample function performs stochastic binary sampling where components are sampled according to their probabilities while ensuring exactly C_l components are selected, maintaining the resource constraint while introducing controlled randomness in architecture exploration. $\mathrm{Score}(\mathcal{A}_l)$ comprehensively evaluates component importance by considering static importance, dynamic importance, feature importance, and correlation penalty metrics. The resulting $g_{\mathrm{f}}^{(l)}$ determines which components in the layer are retained or pruned, effectively generating a sublayer structure. Detailed design and mathematical equations of $\mathrm{score}(\mathcal{A}_l)$ are provided in **Appendix Sec.A**.

2.3 Multi-Stage Training Process

To effectively learn and utilize the unified architecture space, DANCE employs a three-stage training process (as shown in Figure 2) that progressively optimizes both the architectural distribution and model parameters at the same time.

First Stage: SuperNet Pre-training

We first construct and pre-train a SuperNet containing N stacked layers as the backbone architecture. For each layer, we incorporate multiple parallel branches with different operators and channel dimensions to provide architectural flexibility. The SuperNet training process involves sampling different paths and optimizing them jointly to learn robust and transferable feature representations. To maintain stability during training, we employ progressive path dropping and knowledge distillation between different architectural configurations. Additionally, we introduce architecture-aware regularization to encourage diversity in the learned features across different components. This pre-training stage establishes a strong foundation that can effectively generalize across varying computational environments while ensuring consistent feature extraction capabilities. The pre-trained SuperNet serves as the starting point for subsequent distribution-guided architecture adaptation.

Second Stage: Distribution-Guided Learning

For each layer l, in each batch iteration t, our SelectGate takes three key inputs: the batch-level distribution features $\mathcal{D}_t$, concurrent layer features $\mathcal{F}_l$, and layer-wise resource constraints C_l, generating binary masks $g_{\mathrm{f}}^{(l,t)}$ to indicate components:

$$g_{\mathrm{f}}^{(l,t)} = \mathrm{SelectGate}(\mathcal{D}_t, \mathcal{F}_t^l, C_l)$$

The training process employs a dual-loop optimization strategy that enables comprehensive architecture exploration while ensuring stable performance. In the outer loop across epochs E, the network progressively refines both its model weights and SelectGate parameters.

In each batch iteration t, we sample T different architectures using SelectGate to explore the architectural space while adhering to resource constraints. These sampled architectures are used to train the network weights, with SelectGate parameters being updated based on their performance feedback. This inner loop process enables thorough architecture exploration with efficient parameter optimization.

Specifically, for each batch iteration t, we first obtain T different architecture samples $\{\mathcal{A}_i^{(t)}\}_{i=1}^T$ following SelectGate's probability distribution under given resource constraints C. We then update both the network weights θ and SelectGate parameters ϕ using the training feedback from these sampled architectures. This dual-loop mechanism achieves both efficient architecture exploration and effective parameter optimization while satisfying deployment constraints.

Third Stage: Scenario-Specific Optimization

In the final stage, DANCE optimizes SubNets for specific deployment scenarios. Given the input query data $\mathcal{D}$ and resource constraints C, we obtain the optimal architecture $\mathcal{A}^*$ by inheriting weights from the SuperNet $\mathcal{W}$ and selecting features at each layer:

$$\begin{aligned} g^{(l)} &= \mathrm{SelectGate}(\mathcal{D}, \mathcal{F}_l, C_l), \quad l \in [1, L] \\ \mathcal{A}^* &= \{g_l \odot \mathcal{W}_l\}_{l=1}^L \end{aligned} \tag{4}$$

where g_l is the selection gate output at layer l based on distribution $\mathcal{D}$ and features $\mathcal{F}_l$, and $\mathcal{W}_l$ represents the weights from layer l of the SuperNet. The final architecture $\mathcal{A}^*$ is constructed by applying these pruning layers $g^{(l)}$ to the SuperNet weights. We then fine-tune this architecture through a multi-objective optimization process task performance: $\min_\theta \mathcal{L}_{\mathrm{total}}(\mathcal{A}^*) = \mathcal{L}_{\mathrm{task}}$. Through this carefully designed three-stage process, DANCE achieves continuous adaptation to varying computational constraints and data distributions, resource-efficient architecture search guided by distribution awareness, robust performance across diverse deployment scenarios, and end-to-end trainability without discrete sampling or complex optimization procedures.

Algorithm 1 Three-Stage Training Process

Input: Training data $\mathcal{D}$, Resource constraints C, Initial parameters θ
Output: Optimized SubNet $\mathcal{A}^*$

```
// Stage 1: SuperNet Pre-training
for each training iteration do
    Sample batch data from D
    Update SuperNet parameters θ with standard training
end for
// Stage 2: Distribution-Guided Learning
for each batch iteration t do
    Extract features: F_l^t = BatchRepr_l(D_t)
    for each layer l do
        g_f^(l,t) = SelectGate(D_t, F_l^t, C_l)
    end for
    Update parameters with sampled architectures
end for
// Stage 3: Scenario-Specific Optimization
for deployment scenario do
    Sample batch data from D
    Generate architecture via SelectGate
    A* = {g_f^(l) ⊙ W_l}_{l=1}^L
    Fine-tune A* by minimizing L_total
end for
return A*
```

2.4 Framework Analysis

Given a network with L layers and maximum width W per layer, processing a batch of size B, DANCE achieves $\mathcal{O}(TBL(W + |C| + P))$ computational complexity, where T is the number of training iterations, $|C|$ is the dimension of constraint embeddings, and P denotes the number of trainable parameters in the network. Through distribution learning and continuous evolution, DANCE enables efficient architecture sampling at $\mathcal{O}(1)$ cost per scenario with the learned distribution $p(\mathcal{A}|\mathcal{D}, C)$. While traditional NAS methods require separate searches for each deployment scenario leading to substantial computational overhead, DANCE learns a unified distribution that enables efficient architecture sampling for new scenarios. Compared to methods that directly evaluate architectures without training, DANCE's distribution learning captures both data characteristics and deployment constraints through $\mathcal{F}_l(\mathcal{D})$ for more accurate architecture generation. The discrete nature of traditional search spaces makes it difficult to adapt architectures smoothly as requirements change, but DANCE's continuous evolution through the selection gate g_l enables smooth architecture adaptation via distribution sampling. Unlike methods requiring manual tuning with separate optimizations per scenario, DANCE automates this through the comprehensive scoring mechanism $\mathrm{Score}(\mathcal{A})$ that maintains consistency between selected components across layers. The continuous evolution through the learned distribution $p(\mathcal{A}|\mathcal{D}, C)$ enables dynamic adaptation as constraints change, making DANCE particularly effective for deployment with varying computational requirements.

3 Experiment

To demonstrate the effectiveness of our proposed DANCE, we conduct extensive experiments on five image datasets.

3.1 Datasets and Backbones

To systematically evaluate DANCE's adaptive capabilities and validate its technical innovations, we conduct comprehensive experiments across five datasets using two classic backbone architectures: The evaluation framework employs CIFAR-10 [Krizhevsky *et al.*, 2009] and CIFAR-100 [Krizhevsky *et al.*, 2009] as foundational benchmarks to establish baseline architectural distributions. Three challenging fine-grained datasets—Stanford Cars [Kramberger and Potočnik, 2020], CUB-200-2011 [Wah *et al.*, 2011], and Food-101 [Bossard *et al.*, 2014]-are used to rigorously test the precision of dynamic selection mechanisms and architectural sampling under high-resolution visual discrimination requirements. These datasets test DANCE on several dimensions, including resolution scalability, architecture efficiency, and task complexity progression (basic categorization to fine-grained differentiation). It demonstrates how DANCE can adapt to constantly changing task requirements while balancing resource constraints. For more details about datasets and backbones, please refer to **Appendix Sec. B**.

3.2 Evaluation Metrics

We establish a rigorous evaluation framework to assess the effectiveness of our proposed pruning method across multiple dimensions of performance and efficiency. The primary performance indicator is Top-1 accuracy measured across five datasets (CIFAR-10/100, Food-101, Stanford Cars, CUB-200), complemented by performance retention ratios relative to pretrained models. To quantify computational efficiency, we track both parameter reduction metrics and FLOPs measurements, analyzing pruning granularity through per-layer parameter changes and channel configuration modifications across network stages. We also performing detailed ablation studies to examine the contributions of static, dynamic, and feature importance components. The framework incorporates cross-platform evaluation across different model scales (ResNet-18 [He *et al.*, 2016], VGG-16 [Simonyan, 2014]) and progressive assessment under varying resource constraints (target sparsity from 0.5 to 0.9), enabling comprehensive analysis of the accuracy-efficiency trade-off. The stability and adaptability of our method are further validated through fine-grained resource constraint variations (0.1 to 0.5 with 0.02 step size), providing insights into performance consistency across diverse deployment scenarios. This comprehensive evaluation methodology enables us to rigorously validate our method's capability in achieving optimal balance between model performance and resource efficiency while maintaining robust adaptability.

3.3 Baselines

To demonstrate the effectiveness and efficiency of our approach, we compare our approach with different state-of-the-art NAS methods. OFA [Cai *et al.*, 2019] uses progressive shrinking to train a supernet that supports diverse architectural configurations, enabling flexible deployment under different resource constraints. ProxylessNAS [Cai *et al.*, 2018b] directly trains specialized architectures on target hardware while addressing resource constraints through gradient-based optimization. RecNAS [Peng *et al.*, 2022] employs recursive channel pruning to find efficient architectures, achieving a better trade-off between accuracy and model size. SPOS [Guo *et al.*, 2020] adopts a single-path one-shot architecture to train a supernet, allowing efficient architecture sampling and evaluation. FixMatch [Sohn *et al.*, 2020] leverages a semi-supervised learning [Zhu, 2005] strategy to improve model performance with limited labeled data through consistency regularization. The key difference in our approach is the integration of dynamic channel selection and correlation-aware pruning, which enables more effective identification of redundant parameters while maintaining model accuracy across different sparsity levels. We also implement a Score-based Pruning baseline that directly applies our architecture evaluation metric Score($\mathcal{A}$) (defined in **Appendix Sec.A** and **Sec 2.2**) for the direct parameter.

3.4 Implementation Details

When implementing the training process, we adopt a two-stage approach with carefully selected hyperparameters. Stage 1(Pre-training)runs for epochs with frozen SelectGate modules, while Stage 2 (Distribution-Guided Architecture Learning) continues for epochs with all components activated. The learning rates are selected from [0.0001, 0.0005, 0.001] for different components. We employ the AdamW optimizer with OneCycleLR scheduler using 30% warm-up period and early stopping patience of 15. All components are implemented using PyTorch and monitored through comprehensive metrics, including accuracy, loss components, and gate statistics. The backbone network uses ResNet-18 and VGG-16 architecture with customized SelectGate modules integrated at different layers. Training is performed on standard GPU hardware with automatic mixed precision (AMP) enabled for efficiency. For more details about datasets and backbones, please refer to **Appendix Sec. C**.

3.5 Overall Performance Comparison

As shown in Table 1, we observe three remarkable phenomena across five datasets. First, our Stage 2 results achieve strong performance through direct sampling without any retraining - notably outperforming several baseline methods that require full fine-tuning. For instance, on CIFAR-100, our Stage 2 sampling achieves 70.92% accuracy, surpassing RecNAS (66.50%) and SPOS (60.76%). Second, with minimal parameters (2.1M-3.0M for ResNet-18), our method maintains competitive accuracy across diverse tasks. Third, while retraining further boosts performance, the strong Stage 2 results already demonstrate the effectiveness of our approach.

These phenomena stem from our distribution-guided architecture learning strategy. Unlike traditional methods that

Network	Method	Food-101		Stanford Cars		CUB-200		CIFAR-100		CIFAR-10	
		Params	Acc	Params	Acc	Params	Acc	Params	Acc	Params	Acc
ResNet18	Original (Pretrained)	11.69	85.01	11.69	88.54	11.69	76.87	11.69	79.58	11.69	93.31
	OFA-Width 0.85	8.54	70.89	9.39	72.13	8.08	61.10	8.99	63.03	9.49	77.77
	OFA-Width 1.0	11.70	72.60	11.27	79.23	11.28	66.60	11.17	66.70	11.17	78.23
	FixMatch	8.89	75.90	9.01	79.02	9.02	65.89	8.93	72.69	8.93	85.78
	RecNAS	8.80	77.50	7.90	72.39	8.20	66.11	8.30	66.50	2.70	84.17
	SPOS	7.30	73.42	7.20	70.48	6.90	60.54	7.20	60.76	6.40	**89.12**
	Ours (Stage 2)	2.50	75.41	2.80	76.02	3.00	64.68	2.50	70.92	2.10	78.48
	Ours (Retrained)	**2.50**	**83.80***	**2.80**	**79.93***	**3.00**	**66.90***	**2.50**	**78.37***	**2.10**	87.19
VGG16	Original (Pretrained)	37.78	78.62	37.78	81.89	37.78	74.02	37.78	75.54	37.78	93.87
	OFA-Width 0.85	22.56	50.05	**23.33**	64.43	**22.04**	57.03	**22.57**	59.36	**21.34**	65.91
	OFA-Width 1.0	37.88	56.54	37.70	63.15	37.69	59.93	37.70	60.07	37.25	66.22
	FixMatch	30.22	58.89	30.22	68.12	28.99	60.92	30.43	68.30	30.22	81.78
	RecNAS	32.11	50.64	32.09	56.43	31.40	55.29	33.65	58.08	33.65	**83.80**
	SPOS	24.80	50.82	24.30	50.80	24.80	51.70	25.50	52.60	22.50	72.19
	Ours (Stage 2)	21.57	60.02	26.12	72.80	25.96	69.76	26.06	73.72	25.70	76.18
	Ours (Retrained)	**21.57**	**62.20***	26.12	**76.62***	25.96	**72.83***	26.06	**74.54***	25.70	81.32

Table 1: Performance Comparison of Different Pruning Methods. The best results are in **bold** with marker * for statistical significance better than the second best except the original one ($p < 0.05$).

Ablation Setting	Resource Constraint 0.9	0.8	0.7	0.6	0.5
Default	67.81	65.56	61.67	48.79	22.53
Only Static Importance	51.37	49.87	42.76	28.61	13.17
Only Dynamic Importance	39.90	36.76	28.14	14.08	10.58
Only Feature Importance	45.21	41.96	32.42	16.13	14.10
Only Correlation Penalty	44.56	47.95	35.42	18.76	10.00

Table 2: Ablation experiment on items of combined importance when pruning supernet.

require separate, expensive fine-tuning for each deployment scenario, our SelectGate mechanism learns to directly generate effective architectures by comprehensively evaluating both the static and dynamic importance of components. The data-aware nature of our approach enables the SelectGate to capture task-specific requirements and generate suitable architectures through simple sampling from the learned distribution $p(\mathcal{A}|\mathcal{D}, C)$. The exceptional Stage 2 performance validates our core insight of reformulating architecture search as a continuous evolution problem. Our approach enables rapid architecture derivation through direct sampling while maintaining strong performance by learning a distribution over architectural components rather than searching for fixed architectures. This represents a significant advantage over existing methods that rely heavily on costly fine-tuning procedures.

3.6 Effectiveness of Distribution-Guided Learning vs Simple Score-Based Sampling

As shown in Figure 3, we observe several key advantages of DANCE over score-based pruning, despite both methods utilizing the same importance scoring mechanism. First, DANCE achieves substantially better efficiency-accuracy trade-off, with a much steeper slope in the accuracy-FLOPs curve. While score-based pruning requires 1200-1500M FLOPs to reach 60% accuracy, DANCE achieves similar or better accuracy with only 400-500M FLOPs, demonstrating superior efficiency. Second, score-based pruning shows scattered and inconsistent performance (blue dots widely spread),

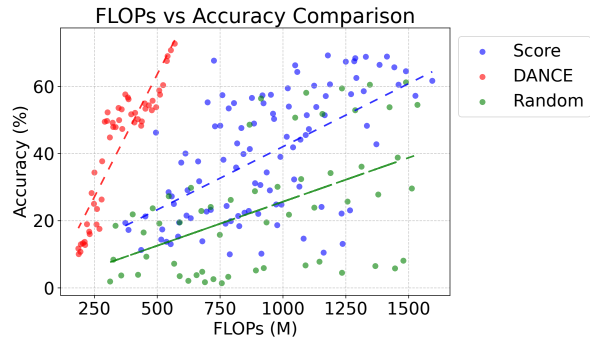

Figure 3: Distribution of FLOPs vs. Accuracy across different pruning strategies. DANCE achieves higher accuracy with lower FLOPs compared to score-based pruning, demonstrating a better efficiency-accuracy trade-off. Random architecture selection on SuperNEt, which randomly samples network architectures, performs significantly worse. The dashed lines show the general trends.

while DANCE demonstrates stable and predictable behavior with a clear trend (red dots closely following the trend line). Random architecture selection on SuperNet performs significantly worse, validating the importance of structured pruning strategies. These phenomena validate that merely using importance scores for selection is insufficient. Although both methods leverage the same scoring mechanism, our SelectGate's distribution-guided learning strategy enables more effective architecture generation by learning to combine and utilize the importance scores through the continuous distribution $p(\mathcal{A}|\mathcal{D}, C)$. The substantial performance gap demonstrates that the key advantage comes from our distribution-guided learning framework rather than the scoring mechanism itself.

To further validate this finding, we conduct experiments on sub-networks with different constraints. As shown in Figure 4, DANCE consistently outperforms score-based pruning across different scales. This consistent superior performance

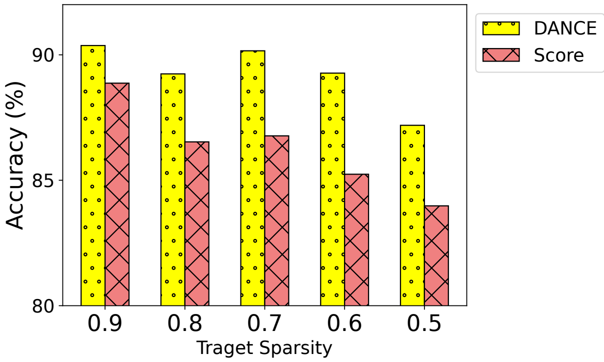

Figure 4: Performance comparison between DANCE and Score-based Purning under different sub-network constraints. Our method consistently demonstrates better performance across different scales, validating the effectiveness of the distribution-guided search.

validates the effectiveness of our distribution-guided learning framework, showing its ability to maintain high accuracy while satisfying various computational constraints.

3.7 Ablation Study

Our sampling strategy incorporates four essential and complementary dimensions of importance metrics. Static importance measures the inherent significance of nodes by reflecting their fundamental structural contributions, while dynamic importance captures runtime activation patterns that reveal operational importance during inference. Feature importance evaluates the discriminative power of different features to preserve critical information, and the correlation penalty balances these metrics by reducing redundancy for efficient sampling. As shown in Table 2, using only a single importance metric leads to substantial performance drops across all resource constraints, validating that these dimensions must work in synergy to achieve optimal sampling effectiveness. Hyperparameter analysis can be found at **Appendix Sec . D**.

4 Related Work

Neural Architecture Search. Neural Architecture Search (NAS) aims to automate the discovery of optimal neural network architectures [Elsken *et al.*, 2019]. Early NAS methods relied on reinforcement learning or evolutionary algorithms to explore discrete architecture spaces, but suffered from prohibitive computational costs due to training each candidate architecture from scratch [Zoph, 2016]. One-shot NAS methods [Pham *et al.*, 2018] improved efficiency by training a supernet containing all possible architectures and sharing weights between candidates. While this reduced search costs, these methods still struggle with effectively adapting architectures across different deployment scenarios due to their discrete nature, as they primarily focus on block-level architecture components rather than feature dimensions.

Recent works have explored more flexible approaches. Progressive NAS [Liu *et al.*, 2018] gradually increases architecture complexity during search but lacks mechanisms for handling varying computational constraints. Differentiable NAS methods [Wang *et al.*, 2021] enable end-to-end architecture optimization but face challenges in maintaining performance consistency across diverse deployment contexts. Although these advances have improved search efficiency, they do not fully address the need for dynamic architecture adaptation under varying computational requirements.

DANCE demonstrates consistent accuracy gains and reduced search costs across diverse datasets, proving its effectiveness for practical applications with varying computational demands and resource limitations.

Efficient Architecture Design. The growing demand for deploying deep neural networks across different computational environments has sparked interest in efficient architecture design [Cai *et al.*, 2018a]. Traditional network pruning approaches focus on removing redundant parameters or channels based on importance metrics [Han *et al.*, 2015], while NAS methods typically search over predefined block-level components. Our work bridges this gap by reformulating both pruning and NAS from a unified feature dimension perspective, enabling more flexible architecture adaptation.

More recent approaches have focused on automated, efficient architecture design. Resource-aware NAS methods [Yang and Sun, 2021] explicitly incorporate hardware constraints during search but struggle with smooth architecture adaptation as requirements change. While these methods have shown promise for specific deployment contexts, they lack mechanisms for effectively modeling the complex trade-offs between multiple competing objectives across diverse scenarios. Additionally, their block-based search spaces limit the granularity of architecture optimization compared to our feature-dimension based approach.

Our DANCE framework addresses these limitations by unifying NAS and pruning through continuous evolution over feature dimensions rather than discrete blocks. This novel perspective enables more efficient and flexible architecture adaptation while maintaining performance consistency across diverse deployment scenarios.

5 Conclusion

This paper presents DANCE, a novel framework that reformulates neural architecture search through continuous evolution over feature dimensions rather than traditional block-level components. Our approach uniquely bridges the gap between NAS and pruning, providing a unified perspective for flexible architecture adaptation across diverse deployment scenarios. Through extensive experiments, we demonstrate that DANCE successfully addresses several key limitations of existing methods by enabling smooth architecture adaptation, effectively handling diverse computational constraints, and efficiently balancing multiple competing objectives during the search process. Our results show that DANCE achieves superior accuracy compared to state-of-the-art methods while significantly reducing search costs across various real-world deployment scenarios. These findings validate the effectiveness of viewing architecture search through the lens of feature-dimension optimization and continuous evolution, suggesting a promising direction for future research in automated neural architecture design.

Acknowledgements

This research was partially supported by Research Impact Fund (No.R1015-23), Collaborative Research Fund (No.C1043-24GF), Huawei (Huawei Innovation Research Program, Huawei Fellowship), Tencent (CCF-Tencent Open Fund, Tencent Rhino-Bird Focused Research Program), Alibaba (CCF-Alimama Tech Kangaroo Fund No. 2024002), Ant Group (CCF-Ant Research Fund), and Kuaishou.

Contribution Statement

Maolin Wang and Tianshuo Wei conceptualized the research idea, designed the methodology, and conducted the primary experiments. Maolin Wang led the software implementation, manuscript writing and data analysis. Tianshuo Wei contributed to software implementation and validation. Ruocheng Guo (corresponding author) provided critical insights on the experimental design and data interpretation. Wanyu Wang (corresponding author) supervised the research direction and provided overall guidance. Shanshan Ye (corresponding author) contributed to the methodology verification and oversaw the experimental validation. Sheng Zhang assisted with data collection and analysis, and contributed to manuscript preparation. Lixin Zou assisted with data analysis and visualization. Xuetao Wei provided valuable feedback on the manuscript and contributed to the discussion section. Xiangyu Zhao supervised the project, secured funding, and provided critical revision of the manuscript. All authors reviewed and approved the final version of the manuscript.

References

[Bender *et al.*, 2018] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *Proc. of ICML*, pages 550–559, 2018.

[Bossard *et al.*, 2014] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101–mining discriminative components with random forests. In *Proc. of ECCV*, pages 446–461, 2014.

[Cai *et al.*, 2018a] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *Proc. of AAAI*, 2018.

[Cai *et al.*, 2018b] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[Cai *et al.*, 2019] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

[Chen *et al.*, 2022] Bo Chen, Xiangyu Zhao, Yejing Wang, Wenqi Fan, Huifeng Guo, and Ruiming Tang. Automated machine learning for deep recommender systems: A survey. *arXiv preprint arXiv:2204.01390*, 2022.

[DONG *et al.*, 2023] Pei-jie DONG, Xin NIU, Zi-mian WEI, and Xue-hui CHEN. Review of one-shot neural architecture search. *Computer Engineering & Science*, page 191, 2023.

[El Bsat *et al.*, 2017] Salam El Bsat, Haitham Bou Ammar, and Matthew Taylor. Scalable multitask policy gradient reinforcement learning. In *Proc. of AAAI*, 2017.

[Elsken *et al.*, 2019] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, pages 1–21, 2019.

[Gao *et al.*, 2023] Jingtong Gao, Xiangyu Zhao, Bo Chen, Fan Yan, Huifeng Guo, and Ruiming Tang. Autotransfer: Instance transfer for cross-domain recommendations. In *Proc. of SIGIR*, pages 1478–1487, 2023.

[Guo *et al.*, 2020] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *Proc. of ECCV*, pages 544–560, 2020.

[Han *et al.*, 2015] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Proc. of NeurIPS*, 2015.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. of CVPR*, pages 770–778, 2016.

[Howard, 2017] Andrew G Howard. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[Jin *et al.*, 2021] Wei Jin, Xiaorui Liu, Xiangyu Zhao, Yao Ma, Neil Shah, and Jiliang Tang. Automated self-supervised learning for graphs. *arXiv preprint arXiv:2106.05470*, 2021.

[Kramberger and Potočnik, 2020] Tin Kramberger and Božidar Potočnik. Lsun-stanford car dataset: enhancing large-scale car image datasets using deep learning for usage in gan training. *Applied Sciences*, page 4913, 2020.

[Krizhevsky *et al.*, 2009] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[Li *et al.*, 2023] Muyang Li, Zijian Zhang, Xiangyu Zhao, Wanyu Wang, Minghao Zhao, Runze Wu, and Ruocheng Guo. Automlp: Automated mlp for sequential recommendations. In *Proceedings of the ACM web conference 2023*, pages 1190–1198, 2023.

[Lin *et al.*, 2022] Weilin Lin, Xiangyu Zhao, Yejing Wang, Tong Xu, and Xian Wu. Adafs: Adaptive feature selection in deep recommender system. In *Proc. of KDD*, pages 3309–3317, 2022.

[Liu *et al.*, 2018] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proc. of ECCV*, pages 19–34, 2018.

[Liu *et al.*, 2020] Haochen Liu, Xiangyu Zhao, Chong Wang, Xiaobing Liu, and Jiliang Tang. Automated embedding size search in deep recommender systems. In *Proc. of SIGIR*, pages 2307–2316, 2020.

[Liu *et al.*, 2024] Ziru Liu, Kecheng Chen, Fengyi Song, Bo Chen, Xiangyu Zhao, Huifeng Guo, and Ruiming Tang. Autoassign+: Automatic shared embedding assignment in streaming recommendation. *Knowledge and Information Systems*, pages 89–113, 2024.

[Peng *et al.*, 2022] Cheng Peng, Yangyang Li, Ronghua Shang, and Licheng Jiao. Recnas: Resource-constrained neural architecture search based on differentiable annealing and dynamic pruning. *IEEE Transactions on Neural Networks and Learning Systems*, pages 2805–2819, 2022.

[Pham *et al.*, 2018] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *Proc. of ICML*, pages 4095–4104, 2018.

[Real *et al.*, 2019] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proc. of AAAI*, pages 4780–4789, 2019.

[Ren *et al.*, 2021] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys (CSUR)*, pages 1–34, 2021.

[Simonyan, 2014] Karen Simonyan. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[Sohn *et al.*, 2020] Kihyuk Sohn, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin A Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *Proc. of NeurIPS*, pages 596–608, 2020.

[Song *et al.*, 2022] Fengyi Song, Bo Chen, Xiangyu Zhao, Huifeng Guo, and Ruiming Tang. Autoassign: Automatic shared embedding assignment in streaming recommendation. In *Proc. of ICDM*, pages 458–467, 2022.

[Tan and Le, 2019] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proc. of ICML*, pages 6105–6114, 2019.

[Wah *et al.*, 2011] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[Wang *et al.*, 2019] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proc. of CVPR*, pages 8612–8620, 2019.

[Wang *et al.*, 2021] Ruochen Wang, Minhao Cheng, Xiangning Chen, Xiaocheng Tang, and Cho-Jui Hsieh. Rethinking architecture selection in differentiable nas. *arXiv preprint arXiv:2108.04392*, 2021.

[Wu *et al.*, 2019] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proc. of CVPR*, pages 10734–10742, 2019.

[Yang and Sun, 2021] Zhao Yang and Qingshuang Sun. Efficient resource-aware neural architecture search with dynamic adaptive network sampling. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2021.

[Zhang *et al.*, 2023] Zijian Zhang, Xiangyu Zhao, Hao Miao, Chunxu Zhang, Hongwei Zhao, and Junbo Zhang. Autostl: Automated spatio-temporal multi-task learning. In *Proc. of AAAI*, pages 4902–4910, 2023.

[Zhao *et al.*, 2021a] Xiangyu Zhao, Haochen Liu, Wenqi Fan, Hui Liu, Jiliang Tang, and Chong Wang. Autoloss: Automated loss function search in recommendations. In *Proc. of KDD*, pages 3959–3967, 2021.

[Zhao *et al.*, 2021b] Xiangyu Zhao, Haochen Liu, Hui Liu, Jiliang Tang, Weiwei Guo, Jun Shi, Sida Wang, Huiji Gao, and Bo Long. Autodim: Field-aware embedding dimension searchin recommender systems. In *Proc. of WWW*, pages 3015–3022, 2021.

[Zhao, 2022] Xiangyu Zhao. Adaptive and automated deep recommender systems. *ACM SIGWEB Newsletter*, pages 1–4, 2022.

[Zhaok *et al.*, 2021] Xiangyu Zhaok, Haochen Liu, Wenqi Fan, Hui Liu, Jiliang Tang, Chong Wang, Ming Chen, Xudong Zheng, Xiaobing Liu, and Xiwang Yang. Autoemb: Automated embedding dimensionality search in streaming recommendations. In *Proc. of ICDM*, pages 896–905, 2021.

[Zhu *et al.*, 2023] Chenxu Zhu, Bo Chen, Huifeng Guo, Hang Xu, Xiangyang Li, Xiangyu Zhao, Weinan Zhang, Yong Yu, and Ruiming Tang. Autogen: An automated dynamic model generation framework for recommender system. In *Proc. of WSDM*, pages 598–606, 2023.

[Zhu, 2005] Xiaojin Jerry Zhu. Semi-supervised learning literature survey. 2005.

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