공학박사 학위논문

Enhanced Neural Architecture Search and Applications

향상된 신경망 구조 탐색과 응용

2022년 2월

서울대학교 대학원

전기·정보공학부

나 병 국

Enhanced Neural Architecture Search and Applications

지도교수 윤 성 로

이 논문을 공학박사 학위논문으로 제출함

2022년 2월

서울대학교 대학원

전기·정보공학부

나 병 국

나병국의 박사 학위논문을 인준함

2022년 2월

위 원 장 _____ (인)

부위원장 _____ (인)

위 원 _____ (인)

위 원 _____ (인)

위 원 _____ (인)

Abstract

Deep learning with deep neural networks (DNNs) has become one of the most popular choice for realizing artificial intelligence systems. To apply the advanced technology of deep learning to various fields, it is necessary not only to nurture deep learning experts, but also to enable non-experts to gain DNNs with high-level performance. As an approach based on automatic machine learning, neural architecture search (NAS), which can automatically obtain a DNN architecture yielding the promising performance in the target domain, is garnering considerable interest as a practical alternative to the manual design for DNNs. Precise execution of NAS requires the tremendous search cost, but the weight-sharing makes NAS estimate the performance of DNNs, thereby significantly reducing the search cost. Based on the weight-sharing, diverse NAS methods have been proposed and produced results that exceed the performance of hand-crafted DNNs in various fields. This dissertation contains methods and substantial results on three essential research topics in NAS, including how to reduce the search cost, how to increase search performance by improving search spaces and search algorithms, and how to facilitate NAS for domains that overlook the architectural importance.

The first research is to reduce the search cost of NAS methods without search performance degradation. The significant computational search cost of NAS is a hindrance to researchers who want to employ NAS to their domains. To address this issue, we introduce proxy data, *i.e.*, a representative subset of the target data, for the acceleration of NAS methods. Even though data selection has been used across various fields in deep learning, our evaluation for existing data selection methods

on NAS benchmarks reveals that they are not suitable for NAS and a new data selection method is necessary. Based on in-depth analysis on proxy data constructed using these selection methods through data entropy, we propose a novel proxy data selection method tailored for NAS. To empirically demonstrate the effectiveness, we conduct thorough experiments across diverse datasets, search spaces, and NAS algorithms. Consequently, NAS algorithms with the proposed data selection method discover architectures that are competitive with those obtained using the entire dataset. It significantly reduces the search cost: executing DARTS with the proposed data selection method requires only 40 minutes on CIFAR-10 and 7.5 hours on ImageNet with a single GPU. Furthermore, as the inverse approach from the conventional NAS, when the architecture searched on our proxy data of ImageNet is transferred to the smaller datasets, *i.e.*, CIFAR-10, a test error of 2.4% is yielded.

In the second research, we improve the search performance of differentiable NAS working on a cell-based search space by refining the search space. Through continuous relaxation on a super-network cell, differentiable NAS methods can approximately evaluate various cells during the search process. When a cell is derived from an optimized super-network cell, a number of the evaluated cells, which are not included in the conventional cell-based search space but may achieve a high performance, are inevitably discarded by the conventional argmax selection rule. To overcome the limitation of the conventional cell derivation, we extend the search space and propose a top-k survival rule to derive a cell belonging in the extended search space. Meanwhile, we observe that existing NAS methods experience a significant decrease in search performance when coupled with the top-k survival rule. To properly explore the extended search space, we propose BtNAS that individually models the operation strengths as continuous random variables following Beta distributions. BtNAS approximates these true posterior distributions using the variational Bayes

method, and the parameters of the variational Beta distributions are trained through gradient-based optimization and the pathwise derivative estimator. BtNAS derives cells that are competitive to those recently reported by differentiable NAS methods and yield networks with fewer parameters. In addition, we adjust operation strengths to make all intermediate node in the super-network cell to be activated, and then achieve a state-of-the-art test error of 2.3% on CIFAR-10.

The last research is to exploit NAS to search for an energy-efficient spiking neural network (SNNs). SNNs, which mimic information transmission in brains with neurons and synapses, have received considerable attention. SNNs can efficiently process spatio-temporal information through event-driven computations with discrete and sparse spikes. Most previous studies focused on training methods to improve the performance and energy-efficiency of SNNs, and the architectural effect regarding SNNs was rarely studied. To facilitate NAS for SNNs, we propose a spike-aware neural architecture search framework, named AutoSNN. We first construct a two-level search space for SNNs by identifying which architectural factors are more suitable for SNNs than for conventional DNNs. Afterwards, to estimate the performance and the spike generation of candidate architectures in the search space, AutoSNN adopts the one-shot super-network training scheme based on weight-sharing and the evolutionary search algorithm. The architecture fitness value used for the architecture evaluation is specifically designed for considering both the accuracy and number of spikes during the search process. An SNN searched by AutoSNN outperforms handcrafted SNNs, in terms of both accuracy and energy-efficiency. We also demonstrate the superiority of AutoSNN on various datasets, including neuromorphic datasets.

Throughout this dissertation, we propose methods for developing an efficient and effective NAS framework and show a strategy to utilize NAS according to the three important research topics. To validate the proposed methods, we provide a significant

amount of experimental results, thereby demonstrating that the architecture choice must be considered to maximize the performance of deep learning. Therefore, owing to the approaches introduced in this dissertation, it is expected that NAS will be actively employed and researchers from diverse fields can easily obtain promising results by adopting deep learning with NAS.

keywords: Deep Neural Network, Neural Architecture Search, Data Sampling, Cell-based Neural Network, Spiking Neural Network

student number: 2014-21622

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, deep learning has continually produced promising results in a wide range of domains of artificial intelligence (AI) applications, including but not limited to computer vision [34, 53, 90, 109, 126, 152], speech recognition [2], natural language processing [21, 25, 129], and reinforcement learning [113, 119]. AI models based on deep learning have achieved human-level intelligence and significantly contributed to improving the quality of our daily lives by doing labor-intensive work for human, such as robotics [74] and autonomous driving system [45]. Experiencing the great stride of deep learning, industrial and academic communities have tried to apply deep learning to their business areas and research fields. In particular, deep learning have been actively studied in the field of computer vision to exploit visual information everywhere, and numerous applications based on deep learning exist: image classification [34, 53, 126], object detection [90, 109, 152], semantic segmentation [54, 101], image style transfer [39, 92], super-resolution [68, 78], reconstruction of damaged images [88], restoration of old images [130], image generation [43, 56], and so on.

The reason why deep learning has been in the spotlight and limitlessly extended

to various applications is because not only deep learning algorithms tailored for each application have emerged, but also deep neural networks (DNNs) can effectively learn and process visual representation contained in images. The development of DNNs started with LeNet [77] that was proposed in 1998 and contained convolutional layers to consider the locality in images, and afterwards, diverse architectures [34, 53, 58, 59, 76, 112, 120, 123] were devised by deep learning researchers. The strategy of designing a module and stacking it to construct an entire neural network increased the design efficiency of researchers [120, 123]. Residual connections significantly improved the performance by allowing DNNs to be extremely deep (*e.g.*, with more than 100 layers) and stably trained [53]. For mobile devices with constrained computation power, inverted bottleneck blocks were introduced, resulting in higher performance with fewer parameters of DNNs and fewer number of floating point operations (FLOPs) [112]. In the past year, DNNs based on attention modules [129], which help to reduce the inductive bias of DNNs caused by human prior and enable self-learning of visual information contained in image data, have received considerable interest [34]. It is not an exaggeration to claim that the efforts of deep learning experts over 30 years for numerous trial-and-errors with the significant amount of computation have made the momentous prosperity of DNNs.

However, figuring out which DNN is best-suited for a target task among these DNNs or newly designing a better DNN can become a stumbling block to applying deep learning to diverse areas. Because the rich know-how of the experts and the enormous amount of computation for the architecture evaluation are required. Furthermore, it is necessary to find training hyperparameters with consideration of DNNs and tasks. Therefore, this two-fold burden of obtaining AI models based on deep learning for applications is a challenging problem that we must overcome.

As an approach to tackle this problem, automatic machine learning (AutoML)

helps researchers produce AI models with high performance, even if they lack knowledge about machine learning. In the spirit of AutoML, for deep learning, hyperparameter optimization (HPO) and neural architecture search (NAS) have been introduced. These techniques not only make it easier for engineers unfamiliar with deep learning to utilize deep learning, but also can increase the research efficiency of deep learning researchers. HPO [36, 72, 147] allows the high performance of a DNN to be obtained in relatively short time by automatically searching for optimal hyperparameters for training the DNN without deeply considering the DNN and the application. Also, without the knowledge of experts, NAS [35, 139] automatically finds a DNN architecture suitable for the application. Recently, an interesting approach to combine both HPO and NAS techniques has been presented [22, 46]. As DNNs searched by modern NAS algorithms outperform hand-crafted DNNs in different types of tasks [9, 29, 47, 66, 132, 143, 154], NAS is garnering considerable interest as a practical alternative to manual architecture design.

However, precisely executing NAS requires to train each candidate architecture to convergence for evaluation, inducing the tremendous computational search cost and hindering NAS from its practical usage [107, 108, 159, 160]. To remedy this issue, weight-sharing [105] was proposed, approximately estimating the performances of the architectures at convergence. In particular, for an image classification task on CIFAR-10 [75], the search cost of NAS has been remarkably reduced from tens of thousands of GPU hours to less than one GPU day; thus, most of the recent NAS methods have been developed under the paradigm of weight-sharing. Unfortunately, NAS methods based on weight-sharing have experienced a gap between the true performances of architectures at convergence and their estimated performances [139]. Even though NAS has been actively studied to increase search quality while lowering search cost, plenty of room for improvement remains to be generally used in

various fields based on deep learning.

This dissertation comprehensively covers three important research topics in NAS, including how to reduce the search cost, how to increase search performance by improving search spaces and search algorithms, and how to facilitate NAS for domains that overlook the architectural importance. In Chapter 3, to reduce the search cost, we introduce to use proxy data, which is a representative subset of data, and suggest a novel proxy data selection method suitable for NAS. In Chapter 4, to improve search performance of NAS working on a conventional cell-based search space [89], we extend the cell-based search space and propose a promising search algorithm and a cell derivation method. In Chapter 5, to exploit NAS to design energy-efficient spiking neural networks (SNNs), we construct a search space considering properties of SNNs and propose a search algorithm that effectively finds an architecture, which achieves higher performance and better energy-efficiency than conventional SNN architectures.

Even though weight-sharing [105] enables NAS algorithms to be completed less than one GPU day, it still be restricted to small-scale datasets such as CIFAR-10 [75]. In Chapter 3, to further reduce the search cost of NAS, we introduce a proxy setting to use a subset of the target data, named proxy data. Data selection is widely used across various fields in deep learning [10, 20, 44, 117], but thorough studies regarding data selection do not exist in NAS. Using benchmarks for NAS, we evaluate existing data selection methods, and to minimize the size of proxy data, the substantial results reveal the necessity for a selection method, designed specifically for NAS. Subsequently, based on the analysis for the relationship between search performances and data entropy distributions in proxy data, we propose a novel selection method that chooses primarily low- and high-entropy data. We demonstrate that NAS algorithms with the proposed selection method achieve their own search performances, which

can be obtained using the entire data, with the minimal size of proxy data, compared to conventional selection methods. In addition, searching on ImageNet [111] is completed in 7.5 GPU hours when incorporating the proposed selection into DARTS [89]. The searched architecture achieves the top-1 test error of 24.6% and 2.4% on ImageNet and CIFAR-10, respectively. This chapter is based on the following paper:

- Byunggook Na, Jisoo Mok, Hyeokjun Choe, Sungroh Yoon, "Accelerating Neural Architecture Search via Proxy Data," in *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

In Chapter 4, we discuss approaches to improve search performance of differentiable NAS, which is based on a conventional cell-based search space, referred to as the DARTS cell space [89, 139]. The DARTS cell space is encoded by a super-network cell with operation strengths, and through continuous relaxation [89], the operation strengths can be trained using gradient-based optimization in search process. Once the search process ends, following a conventional cell derivation, only cells that fit the DARTS cell stereotype can be derived from an optimized super-network cell. However, during the search process, various cells that do not belong in DARTS cell space can be approximately evaluated. To consider such cells, we introduce an extension of the search space, called ExtCell space, where cells have multiple operations between two nodes, different from DARTS cells. In addition, we formulate NAS problem as global competition among all the candidate operations within a super-network cell, and suggest a new cell derivation method, called the top-k survival rule, which selects operations with the top-k strengths within the super-network cell. To effectively explore the ExtCell space, we propose BtNAS that individually optimizes the operation strengths, which are modeled by continuous random variable following beta distributions. Through variational Bayes method [6, 71] to approximate these true posterior distributions, BtNAS effectively optimizes parameters of

variational beta distributions, which are compatible with the top-k survival rule. As a result, ExtCells searched by BtNAS achieve competitive performance with fewer parameters, compared to the state-of-the-arts DARTS cells. Additionally, we adjust operation strengths to make all intermediate node activated, and then find a cell with a state-of-the-art test error of 2.3% on CIFAR-10. This chapter is based on the following paper:

- Byunggook Na, Jisoo Mok, Hyeokjun Choe, Sungroh Yoon, "BtNAS: Differentiable Neural Architecture Search with Beta Distribution," (under review).

In Chapter 5, inspired by the success of NAS for DNNs, we exploit NAS to search for energy-efficient SNNs, whose properties are different from those of DNNs. SNNs, which mimic the brain's information processing system [93], transmit information through sparse and binary spikes, which enable event-driven computing; on neuromorphic chips [23, 97], energy consumption occurs when asynchronously firing spikes. Even though training methods, which aim to improve the performance and reduce the spike generation of an SNN, have been studied [37, 55, 63, 79, 99, 138, 156], the architectural perspective was rarely discussed. We observe that even when the same training method is used, the accuracy and number of spikes differ significantly depending on the SNN architecture. Hence, as the first attempt to exploiting NAS for SNN design, we propose a spike-aware NAS framework, named AutoSNN. To construct a search space tailored for energy-efficient SNNs, we introduce a two-level search space that consists of a macro-level backbone architecture and micro-level candidate spiking blocks. Based on weight-sharing, AutoSNN consecutively performs two procedures: training a super-network using a direct training method for SNNs and then searching for an architecture, which is best-suited for the spike-aware target objective, in the manner based on an evolutionary algorithm. AutoSNN obtains an SNN that outperforms conventional hand-crafted SNNs across various datasets in

terms of both the accuracy and number of spikes. This chapter is based on the following paper:

- Byunggook Na, Jisoo Mok, Seongsik Park, Dongjin Lee, Hyeokjun Choe, Sungroh Yoon, "AutoSNN: Towards Energy-Efficient Spiking Neural Networks," *arXiv preprint*, arXiv:2201.12738, 2022 (under review).

The remainder of this dissertation is organized as follows: Chapter 2 provides background for this dissertation, Chapters 3-5 present the proposed methods and substantial results, and in Chapter 6, we conclude this dissertation and suggest future research for NAS.

Chapter 2

Background

2.1 Neural Architecture Search

A number of highly experienced deep learning researchers are required to harness the power of DNNs, because designing the optimal neural network is a heuristic and labor-intensive process. Recently, to remedy the financial and physical cost for hand-crafting DNN architectures, NAS has emerged. NAS aims to automatically find architectures with high performance from a set of architectures, and thus can reduce the human cost of designing and testing hundreds of DNNs. Because DNNs searched by modern NAS algorithms outperform hand-crafted neural networks in different types of tasks [9, 29, 38, 42, 47, 66, 132, 143, 154], NAS is garnering considerable interest as a practical alternative to manual architecture design.

A general NAS formulation consists of two basic components: a search space and a search algorithm [35]. Before providing the detailed explanation for these two components, we formally describe a problem setting for NAS. A search space, denoted by $\mathcal{A}$, is a set of architectures that would be candidates for a given task, *i.e.*, a dataset and an objective function. We can obtain performance of an architecture $A \in \mathcal{A}$,

such as classification accuracy, by training A equipped with trainable weights w; the trained A is denoted by $\langle A, w^*(A)\rangle$ [139]. Given a dataset, which consists of training (D_{train}) , validation (D_{val}), and testing data (D_{test}), the NAS is to search for an optimal architecture A^* yielding the highest performance on D_{test}, which is formulated as:

$$A^* = \underset{A \in \mathcal{A}}{\operatorname{argmax}} \operatorname{Eval}(\langle A, w^*(A)\rangle; D_{\text{test}}), \tag{2.1}$$

where Eval() is the evaluation function, such as calculating classification accuracy, of an input architecture using D_{test}. However, as in usual scenarios of machine learning, D_{test} is not available. We need to utilize D_{train} and D_{val} to estimate the performance of trained architectures during the search process. Therefore, Eq. 2.1 is rewritten as a bi-level optimization, which is:

$$\begin{aligned} A^* &= \underset{A \in \mathcal{A}}{\operatorname{argmax}} \operatorname{Eval}(\langle A, w^*(A)\rangle; D_{\text{val}}) \\ \text{s.t.} \quad w^*(A) &= \underset{w}{\operatorname{argmin}}\, \mathcal{L}(w(A); D_{\text{train}}), \end{aligned} \tag{2.2}$$

where $\mathcal{L}$ is the loss function, which enables A to be trained on D_{train}, such as cross-entropy loss function used for image classification task [98]. Now, we provide detailed explanation for two components of NAS, a search space and a search algorithm.

2.1.1 Search Space

Cell-based Search Space

A cell-based search space consists of DNNs that repeatedly stack cell architectures in bi-chain-styled manner, where each cell receives input feature maps from two consecutive, previous cells, as shown in 2.1. Various cell-based search spaces [89, 105, 108, 160] were studied, but recently, they have been trimmed to improve the qual-

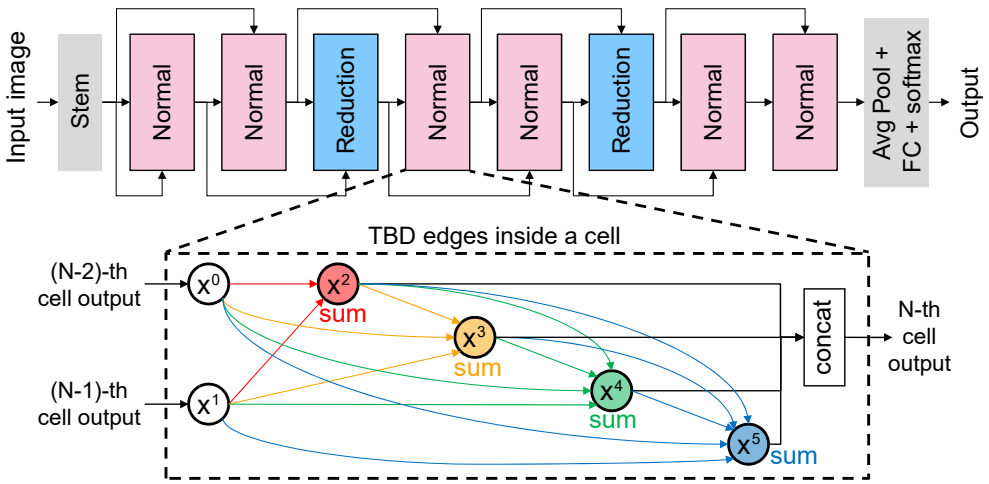


Figure 2.1: A macro architecture and a cell structure for the cell-based search space, where TBD is the abbreviation of to-be-determined that means a placeholder to be assigned by an operation.

ity of the search space by removing candidate operations that are rarely selected, and then consolidated into a single search space, referred to as DARTS cell space [89]. To construct a cell architecture, eight candidate operations are usually used as follows:

- zeroize (none operation)
- skip connection (identity operation)
- 3×3 separable convolution (×2)
- 5×5 separable convolution (×2)
- 3×3 max pooling
- 3×3 average pooling
- 3×3 dilated separable convolution
- 5×5 dilated separable convolution

In the DARTS cell space, an element is a DARTS cell pair consisting of two types of cells: a normal cell and a reduction cell. A normal cell preserves the spatial resolution of the input feature map and a reduction cell halves the spatial resolution and doubles the number of channels. NAS algorithms on the DARTS cell space focus on searching for an optimal cell architecture instead of designing the entire network. Hence, it is enabled to use a proxy network for the search process, which is shallower than a target network. Conventionally, a target network includes 18 normal cells and two reduction cells, whereas a proxy network for the search process includes six

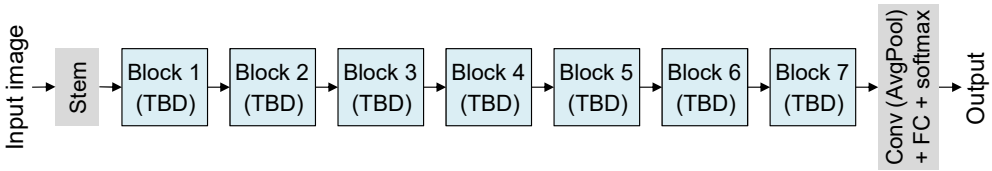


Figure 2.2: A macro architecture for the MobileNet search space, where TBD is the abbreviation of to-be-determined that means a placeholder to be assigned by an operation.

normal cells and two reduction cells. From now on, for convenience, we refer to a DARTS cell pair as a DARTS cell by omitting "pair".

A DARTS cell is represented in the form of a directed acyclic graph (DAG) with N nodes and E edges. The DARTS cell has two input nodes, which are the feature maps generated by two previous cells; thus, within a cell, there are $N-2$ intermediate nodes and corresponding output feature maps. The feature map at each node is denoted by x^i, where $i \in \{0, 1\}$ for the input nodes and $i \in \{2, ..., N-1\}$ for the intermediate nodes. Each edge (i, j) is associated with operation $o^{(i,j)}$, one of the candidate operations from the predefined candidate operation set $\mathcal{O}$. The feature map x^j at intermediate node j is computed by summing the outputs of all activated incoming edges, *i.e.*, $x^j = \sum_{i<j} o^{(i,j)}(x^i)$, where $j \in \{2, ..., N-1\}$. In the DARTS cell, the number of activated incoming edges is typically set to two. The output of a cell is defined as a depthwise concatenation of all intermediate nodes, *i.e.*, $\mathsf{concat}(x^2, ..., x^{N-1})$.

MobileNet Search Space

MobileNet search space [8, 127, 136] is also mainly studied in NAS community. Motivated by MobileNet [57, 112], a network contains layers in chain-styled manner, as shown in Figure 2.2; the network is often referred to as a macro architecture. In the macro architecture, the layer will be determined by selecting the most suitable

candidate operation after the completion of a search process. Candidate operations are based on the inverted bottleneck block that is a main component block in MobileNetV2 [112]. The inverted bottlenecks include various design choices, for example, the kernel size, the expansion ratio, the usage of squeeze-and-excitation module [58], and so on. Hence, a set of candidate operations is changeable according to how many design choices are considered to define the search space. Conventionally, the kernel size K and the expansion ratio E range in $\{3, 5, 7\}$ and $\{1, 2, 3, 4, 5, 6\}$, respectively. In most of NAS algorithms on the MobileNet search space, the search space is constructed by including a skip connection and six types of inverted bottleneck blocks, *e.g.*, $K \in \{3, 5, 7\}, E \in \{3, 6\}$ or $K \in \{3, 5\}, E \in \{3, 4, 6\}$.

2.1.2 Search Algorithm

Early NAS methods adopted a strategy to sample and evaluate architectures, each one of which needed to be trained until convergence. These methods were based on reinforcement learning [159, 160] or evolutionary algorithm [107, 108], and often required thousands of GPU hours to search for an architecture. In reinforcement learning, a controller is trained and samples candidate architectures during the search process, which includes training and evaluation of these architectures. At the end of the search process, the trained controller samples multiple potential architectures, and then the best one is selected. In NAS based on evolutionary algorithms, performances of architectures in a population, which is updated during the search process, gradually increase. After the evolutionary search ends, the best one from the population is selected.

Naturally, various methods have been developed to decrease the tremendous search cost [35, 139]. To investigate a reliable proxy setting for architecture evaluation, EcoNAS [157] suggested to use three reduction factors (lower channel dimension

of architectures, lower image resolution, and fewer training epochs) and proposed a hierarchical proxy to improve the accuracy of neural network performance estimation under such more reduced computational setting. Also, using smaller dataset can be adopted to reduce the search cost (Chapter 3 in this dissertation). Apart from the proxy-based approach, most of the recent search algorithms are based on the weight-sharing technique [105], resulting in the significant reduction in search cost: approximately less than one GPU day.

In NAS algorithms based on the weight-sharing, the search space $\mathcal{A}$ is encoded in a super-network $\mathcal{S}(W)$, where W denotes the weights in $\mathcal{S}$. W is shared across candidate architectures, which correspond to sub-networks of $\mathcal{S}$. The weight-sharing enables candidate architectures to train only a single mini-batch, when the architectures are sampled from the super-network during search process; thus, this approach is referred to as weight-sharing one-shot NAS. The objective of bi-level optimization for weight-sharing NAS is then formulated as:

$$\begin{aligned} A^* &= \underset{A \in \mathcal{S}}{\operatorname{argmax}} \, \mathrm{Eval}(\langle A, W^*(\mathcal{S}) \rangle; D_{\mathrm{val}}) \\ \text{s.t.} \quad W^*(\mathcal{S}) &= \underset{w}{\operatorname{argmin}} \, \mathcal{L}(W(\mathcal{S}); D_{\mathrm{train}}). \end{aligned} \tag{2.3}$$

Through the weight-sharing, we can estimate performance of an architecture, which is sampled from the trained super-network with copying the trained weights. Even though the accurate performance is hardly approximated, it is worthy to adopt the weight-sharing because of significant search cost reduction caused by this technique. Also, it is important to obtain high rank correlation between the estimated performance of architectures and their true performance on D_{val}, rather than to accurately estimate their true performance. Therefore, based on Eq. 2.3, diverse NAS algorithms have been proposed and are mainly categorized into two approaches: heuristic

and differentiable search.

Heuristic Search Algorithm based on One-shot Weight-sharing

For a heuristic search approach with one-shot weight-sharing, various search algorithms have been proposed such as reinforcement learning (RL) algorithms [3, 105, 127] and evolutionary algorithm (EA) [9, 48, 84, 104, 146, 151, 155, 157]. RL-based NAS algorithms train a controller and a super-network, and thereby the trained controller samples desirable architectures from the trained super-network. Among a number of architectures (*e.g.*, 100) sampled by the trained controller, the architecture with the highest performance on D_{val} is selected. EA-based NAS algorithms consist of two procedures: training a super-network with an architecture sampling strategy and searching for an optimal architecture using EAs. During training a super-network, architectures are uniformly sampled [9, 48, 84, 155], and thus all candidate architectures in $\mathcal{A}$ can be evenly trained. In the second procedure, given the trained super-network $\mathcal{S}(W^*)$, architectures inherit weights from $\mathcal{S}(W^*)$ and are evaluated without additional training. Using the trained super-network as an estimator, EA-based search algorithms can find an optimal architecture from population of architectures, which is updated during the second procedure. To improve the reliability of the evaluation process, sampling strategies for super-network training have been diversified: for example, exploitation-exploration strategy with greedy path filtering [146], novelty-based sampling [151], prioritized path distillation [104], or a sampling strategy based on Monte-Carlo tree search [122].

Differentiable Search Algorithm based on One-shot Weight-sharing

In NAS algorithms based on differentiable search approach, trainable architecture parameters are employed into a super-network. The architecture parameters corre-

spond to operation strengths of candidate operations. The differentiable NAS is also composed of two procedures: super-network training (*i.e.*, optimizing both weights of operations and architecture parameters) and derivation of an optimal architecture from the trained architecture parameters.

During the first procedure, the search algorithm must progressively learn the operation strengths, that is, how promising each candidate operation is at a to-be-determined (TBD) block within the macro architecture of MobileNet search space or a TBD edge within the super-network cell architecture of DARTS cell space. To enable a gradient-based optimization of the operation strength, DARTS [89] continuously relaxed a discrete search space (*i.e.*, a set of architectures) by introducing continuous operation strengths to the super-network. Upon continuous relaxation, each TBD block of the super-network can be expressed as a mixed-operation: $\hat{o}(x) = \sum_{o_k \in \mathcal{O}} \theta_k o_k(x)$, where $\mathcal{O}$ and $\theta \in \mathbb{R}^{|\mathcal{O}|}$ denote the candidate operation set and the operation strengths of candidate operations on a TBD block, respectively. The search objective of the differentiable NAS methods with the mixed-operations is equivalent to the optimization of θ to properly model the true operation strengths. Because operation strengths lie in a continuous space, it is possible to jointly optimize the network weights w and the operation strengths θ through a bi-level optimization:

$$\begin{aligned} \theta^* &= \underset{\theta}{\operatorname{argmin}}\, L(\theta, w^*(\theta); D_{\text{val}}) \\ \text{s.t.} \quad w^*(\theta) &= \underset{w}{\operatorname{argmin}}\, L(\theta, w; D_{\text{train}}). \end{aligned} \tag{2.4}$$

Based on how the operation strengths are obtained, the differentiable NAS methods can be divided into deterministic and stochastic approaches. In the former, deterministic functions such as softmax [16, 19, 52, 89, 133, 142, 144, 148] or sigmoid functions [18] are utilized to obtain the continuous operation strengths from architec-

ture parameters; thus, gradients of architecture parameters can be directly obtained through backpropagation. To improve the search reliability of DARTS [89], plagued with problems, such as dominance of skip connection, the operation strengths can be perturbed by random or adversarial noises [14], which provide stochasticity to the search process.

On the contrary, the differentiable NAS methods based on stochastic approach model operation strengths as random variables that can be sampled from some distribution. Most stochastic methods [4, 8, 11, 31, 134, 140, 158] sample discrete operation strengths using probability distributions. To enable gradient-based optimization, they usually adopt the reparameterization trick with the Gumbel technique, as presented in GDAS (Gumbel-softmax) [31] or PR-DARTS (Gumbel-sigmoid) [158]. To improve the search efficiency of GDAS, which samples one-hot encoded operation strength, DATA [11] uses the ensemble Gumbel-softmax function for sampling multi-hot encoded operation strengths. DrNAS [15] stochastically samples the continuous operation strengths from a Dirichlet distribution.

Once the super-network training ends, an optimal architecture can be derived from the optimized super-network with continuous θ^*. Only one operation is allowed for each TBD block, and thus the derivation is based on the argmax function to select the most preferred operation: $o = \mathrm{argmax}_{o_k \in \mathcal{O}} \, \theta_k^*$.

2.1.3 NAS Benchmark

NAS algorithms suffer from a lack of reproducibility, and thus, a fair comparison of NAS algorithms is challenging. Moreover, the extreme computational cost of evaluating NAS algorithms makes repeated testing difficult, in spite of using a proxy setting for NAS. To alleviate these issues in NAS research and provide a fair comparison of search performance, various benchmarks have been proposed on image classifica-

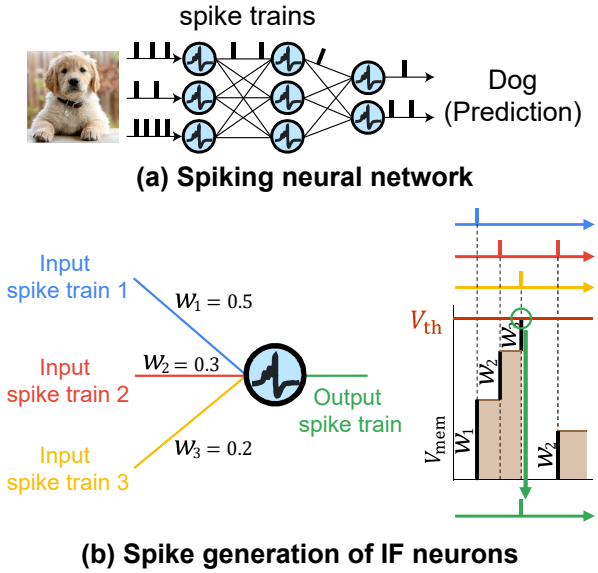


Figure 2.3: (a) A spiking neural network architecture, where information is transmitted through spike trains and (b) spike generation of integrate-and-fire (IF) neurons.

tion task [32, 33, 80, 106, 122, 145, 149], speech recognition task [95], and natural language processing task [73]. These NAS benchmarks provide a comprehensive architecture database whose element is a pair of an architecture and its performance, *e.g.*, accuracies on D_{val} and D_{test}, latency, FLOPs, and so on. In addition, several studies [32, 33, 80, 95, 149] offer an easy-to-implement NAS platform that contains existing NAS algorithms. Performance and information of architectures, which are searched by NAS algorithms, can be obtained by querying them to the architecture database without training the architectures. Therefore, for researchers in NAS community, it is preferable to use the benchmarks as the testing platform for an unbiased assessment of their own search algorithms against other existing NAS algorithms.

2.2 Spiking Neural Networks

SNNs have attracted significant interest as a solution to improve energy-efficiency of DNNs. SNNs are designed to mimic the brain's information processing systems [93].

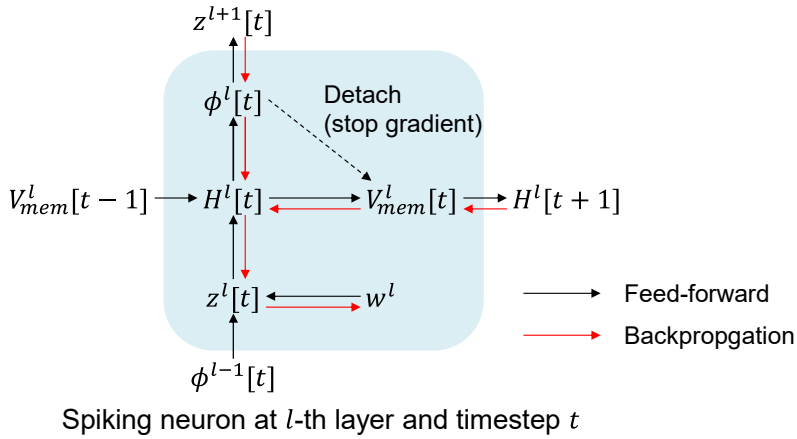


Figure 2.4: Spiking neuron overview: feed-forward and backpropagation flows.

Neurons and synapses in brains correspond to spiking neurons and connections between neurons to transmit information in SNNs, respectively. In SNNs, as depicted in Figure 2.3(a), the spiking neurons transmit information through a spike train, which is a set of sparse and binary spikes and enables event-driven computing. Figure 2.3(b) illustrates that spiking neurons in SNNs have a property of integrate-and-fire, which means input information is accumulated in the neurons and then output spikes are generated when the accumulated information is higher than a threshold. Because of such sparsity of spikes and event-drive computing, SNNs can be the next generation of neural networks to improve the energy-efficiency of DNNs.

Different from a neuron in DNNs, a spiking neuron in SNNs integrates presynaptic inputs from the previous layer into an internal state called membrane potential; this integration is considered as an addition operation, while neurons in DNNs perform linear operations of matrices such as multiplication and addition. When the membrane potential integrated over time exceeds a certain threshold value, the neuron fires a spike that will be transmitted to the next layer, and thus the spiking neurons transmit information through the generated binary spike trains. Among several spiking neuron models, the leaky integrate-and-fire (LIF) neuron is simple yet widely

used due to its effectiveness [41]. The dynamics of the LIF neuron at timestep t are as follows:

$$\tau_{\text{decay}} \frac{\partial V_{\text{mem}}(t)}{\partial t} = -(V_{\text{mem}}(t) - V_{\text{reset}}) + z(t), \tag{2.5}$$

where V_{mem} is the membrane potential of a neuron and V_{reset} is the value of V_{mem} after firing a spike. τ_{decay} is a membrane time constant that controls the decay of V_{mem} and z is the pre-synaptic inputs. For effective simulations with GPUs, the dynamics are transformed into an iterative and discrete-time form. Hence, Eq. 2.5 can be reformulated as follows:

$$z^l[t] = \sum_i w_i^l \phi_i^{l-1}[t] + b^l, \tag{2.6}$$

$$H^l[t] = V_{\text{mem}}^l[t-1] + \frac{1}{\tau_{\text{decay}}}(-(V_{\text{mem}}^l[t-1] - V_{\text{reset}}) + z^l[t]), \tag{2.7}$$

$$\phi^l[t] = \Theta(H^l[t] - V_{\text{th}}), \tag{2.8}$$

$$V_{\text{mem}}^l[t] = H^l[t](1 - \phi^l[t]) + V_{\text{reset}}\phi^l[t], \tag{2.9}$$

where superscript l indicates the layer index; for convenience, following is the explanation by omitting the superscript. The value of membrane potential at timestep t is divided into two states, which represent the values before and after determining whether a spike is generated or not, denoted by $H[t]$ and $V_{\text{mem}}[t]$, respectively. $z[t]$ is a sum of post-synaptic potential of neurons that are obtained by adding w and b as in Eq. 2.6, where $\phi[t]$ is a binary spike at timestep t, w is the synaptic weight, and b is the bias. The spike firing, *i.e.*, $\phi[t] = 1$, is determined using $\Theta(x)$, the Heaviside step function that outputs 1 if $x \geq 0$ or 0 otherwise, and V_{th} is the threshold voltage for firing. To improve performance of SNNs, such as higher accuracy and less timesteps, Fang *et al.* proposed the parametric LIF (PLIF) neuron [37]. In PLIF, the hyperparameter $\frac{1}{\tau_{\text{decay}}}$ is replaced by a trainable decay function $\sigma(\alpha)$, which is a

sigmoid function $1/(1+\exp(-\alpha))$ with a trainable parameter α. In Chapter 5 of this dissertation, we used PLIF neurons for energy-efficient SNNs.

2.2.1 Optimization for SNNs

To obtain a high-performing SNN, researchers have proposed various training methods that can be clustered into two approaches: indirect and direct training. In indirect training approach [28, 49, 65, 69, 102, 103, 110, 116], after training the DNN, the weights are transferred to the corresponding SNN, such that the firing patterns of the spiking neurons are encoded to approximate the activation values of the DNNs; this approach is referred to as DNN-to-SNN conversion. The approximation of the activation values is realized by various neural coding methods, which provide schemes for encoding information into a spike train and decoding information from the spike train. Even though these converted SNNs achieve accuracies comparable to those of DNNs, they heavily rely on the performance of trained DNNs and require a significant amount of timesteps which lead to energy inefficiency, compared with SNNs obtained by direct training methods.

Direct training methods aim to optimize synaptic weight w of every spiking neuron in SNNs. As an approach to directly optimize SNNs, unsupervised learning methods based on the spike-timing-dependent plasticity method [27, 51, 64, 121] were introduced. Inspired from Hebbian learning rule [121], the weights are adjusted according to the relative timing of presynaptic and postsynaptic spikes. However, the unsupervised learning methods were restricted to shallow networks and yielded limited performance for SNNs.

Another approach of direct training is supervised learning based on a backpropagation algorithm [7]. For backpropagation, a surrogate gradient function was used to approximate the gradients in non-differentiable spiking activities [55, 62, 63, 67,

79, 99, 137, 138, 153]. In recent studies, supervised learning has been effective for deep SNNs and yielded high accuracies with few timesteps and sparse generation of spikes [37, 156]. However, the research community for SNNs has solely focused on training methods as a means to improve the performance and energy-efficiency in SNNs. In Chapter 5 in this dissertation, we adopt a supervised learning approach for training and evaluating SNNs.

Training Framework based on a Supervised Learning Approach

In this section, we explain a training process of SNNs under the supervised learning. Assuming a classification task with C classes, a loss function L is defined as:

$$L(\mathbf{o}, \mathbf{y}) = L(\frac{1}{T}\sum_{t=0}^{T-1}\phi^{\mathrm{FC}}[t], \mathbf{y}), \tag{2.10}$$

where T is the timesteps, $\mathbf{y}$ is a target label, and $\mathbf{o} \in \mathbb{R}^C$ is a predicted output, which is calculated by averaging the number of spikes generated by the last fully-connected (FC) layer over T.

As shown in Figure 2.4, spikes in SNNs propagate through both spatial domain (from a lower layer to a higher layer) and temporal domain (from a previous timestep to a later timestep). Hence, derivatives should be considered in the both perspectives. In this section, we provide derivatives of components in a spiking neuron at the l-th layer and timestep t; these derivatives are highlighted by red lines in Figure 2.4. w^l is shared across the total timesteps T, and thus the derivative with respect to w^l can be obtained according to the chain rule as follows:

$$\frac{\partial L}{\partial w^l} = \sum_{t=0}^{T-1}\frac{\partial L}{\partial H^l[t]}\frac{\partial H^l[t]}{\partial z^l[t]}\frac{\partial z^l[t]}{\partial w^l}. \tag{2.11}$$

From Eqs. 2.6 and 2.7, the second and third terms in the right-hand side of Eq. 2.11

can be induced as follows:

$$\frac{\partial z^l[t]}{\partial w^l} = \phi^{l-1}[t], \qquad \frac{\partial H^l[t]}{\partial z^l[t]} = \frac{1}{\tau_{\text{decay}}}. \tag{2.12}$$

By applying the chain rule to the first term, $\frac{\partial L}{\partial H^l[t]}$ is written as:

$$\frac{\partial L}{\partial H^l[t]} = \frac{\partial L}{\partial \phi^l[t]}\frac{\partial \phi^l[t]}{\partial H^l[t]} + \frac{\partial L}{\partial V^l_{\text{mem}}[t]}\frac{\partial V^l_{\text{mem}}[t]}{\partial H^l[t]}. \tag{2.13}$$

The first two terms and the last two terms are related to the spatial and temporal domains, respectively. In the spatial domain, $\frac{\partial L}{\partial \phi^l[t]}$ is obtained by using the spatial gradient back-propagated from the $(l+1)$-th layer as follows:

$$\frac{\partial L}{\partial \phi^l[t]} = \frac{\partial L}{\partial z^{l+1}[t]}\frac{\partial z^{l+1}[t]}{\partial \phi^l[t]} = \frac{\partial L}{\partial z^{l+1}[t]}w^{l+1}. \tag{2.14}$$

Note that SNNs have the non-differentiable property due to the spike firing $\Theta(x)$. Approximation for the derivative of a spike, *i.e.*, $\Theta'(x)$, is necessary to optimize SNNs by gradient-based training. In this dissertation, we approximate $\Theta'(x) = \frac{1}{1+x^2}$ by employing the inverse tangent function for $\Theta(x) = \arctan(x)$. Therefore, $\frac{\partial \phi^l[t]}{\partial H^l[t]}$ is calculated using the approximation and Eq. 2.8:

$$\frac{\partial \phi^l[t]}{\partial H^l[t]} = \Theta'(H^l[t] - V_{\text{th}}). \tag{2.15}$$

The derivatives in the temporal domain of Eq. 2.13 are also induced from Eq. 2.7 and Eq. 2.8 as follows:

$$\frac{\partial L}{\partial V^l_{\text{mem}}[t]} = \frac{\partial L}{\partial H^l[t+1]}\frac{\partial H^l[t+1]}{\partial V^l_{\text{mem}}[t]} = \frac{\partial L}{\partial H^l[t+1]}(1 - \frac{1}{\tau_{\text{decay}}}), \tag{2.16}$$

$$\frac{\partial V^l_{\text{mem}}[t]}{\partial H^l[t]} = 1 - \phi^l[t] - H^l[t]\frac{\partial \phi^l[t]}{\partial H^l[t]}. \tag{2.17}$$

Chapter 3

Accelerating Neural Architecture Search through Proxy Data

3.1 Introduction

NAS aims to reduce the human cost of designing and testing hundreds of neural architectures. In early studies pertaining to NAS [107, 108, 159, 160], however, enormous computational overhead occurred, thereby requiring a significant amount of computing resources to execute search algorithms. To reduce the search cost, most recently developed NAS algorithms employ weight-sharing on a super-network and/or differentiable approach to optimize the architecture parameters lying on continuous space [139]. These techniques enable a more approximate yet faster evaluation of candidate neural architecture performance, thereby significantly reducing the cost of NAS.

In this chapter, to further reduce the search cost of NAS, we introduce an approach to incorporate proxy data, *i.e.*, a representative subset of the target data, with NAS algorithms; Figure 3.1 illustrates our approach applying to NAS algorithms that

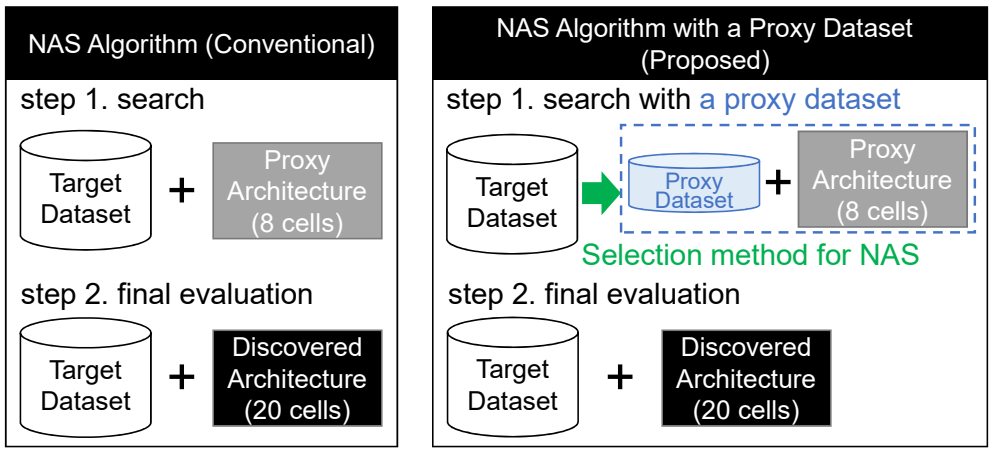


Figure 3.1: Overview: comparison between conventional NAS approach and a proposed NAS approach using proxy data, which is a representative sub-dataset sampled from a target dataset.

explore a cell-based search space. We hypothesize that it is unnecessary to execute NAS algorithms with the entire target data and there is redundant information in the entire data used during search. We are motivated by research based on data selection, which is widely used across various fields in deep learning, such as active learning [20, 117] and curriculum learning [10, 44]. For instance, given a trained model, the core-set selection methods used in active learning aim to select the training data from a large unlabeled dataset to label the selected data with minimum labeling costs, resulting in the effective reduction of the computational cost. However, comprehensive studies regarding the problem of data selection for NAS do not exist. Developing an appropriate selection for NAS is important because NAS algorithms could benefit from the significant reduction in the search cost.

To begin with, we revisit five existing data selection methods and assess whether NAS algorithms can maintain their search performance with proxy data selected by these selection methods. While the substantial results that are obtained by NAS-Bench-1shot1 [149] provide strong empirical support for our hypothesis, they also

reveal the necessity for a new, improved selection method, designed specifically for NAS. Subsequently, we analyze the relationship between the search performances and properties of different selection methods based on the entropy [118] of examples in the proxy data. Based on our analysis, the characteristics of a selection method that renders proxy data effective in preserving the search performance of NAS algorithms are identified. When the selection method chooses primarily low-entropy examples, a competitive architecture is discovered with the resulting proxy data, even when the size of the proxy data is small. To achieve the search performance obtained using the entire data, it is important to include additional high-entropy examples in the proxy data.

Accordingly, we propose a new selection method that prefers examples in the tail ends of the data entropy distribution. The selection method can be implemented in a deterministic or probabilistic manner. For the former, we adopt the composition ratio between low- and high-entropy examples, such that the examples from the opposite ends of the entropy distribution are selected. For the probabilistic manner, we suggest three sampling probabilities that satisfy the identified characteristics of the proxy data effective for NAS. Using NAS-Bench-1shot1, we demonstrate the superiority of the proposed selection method to existing selection methods and show that the search performance is preserved even when using only 1.5K (3%) of training examples selected from CIFAR-10.

We further demonstrate that the proposed selection method can be applied universally across various differentiable NAS algorithms and four benchmark datasets for image classification. The NAS algorithms with the proposed selection discover competitive neural architectures in a significantly reduced search time. For example, executing DARTS [89] using our selection method requires only **40 GPU minutes** on a single GeForce RTX 2080ti GPU. Owing to the reduction in search cost, search-

ing on ImageNet can be completed in **7.5 GPU hours** on a single Tesla V100 GPU when incorporating the proposed selection into DARTS. The searched architecture achieves a test error of 24.6%, surpassing the performance of the architecture, which is discovered by DARTS on CIFAR-10 and then transferred to ImageNet. In addition, when this architecture is evaluated on CIFAR-10, it achieves a top-1 test error of **2.4%**, demonstrating state-of-the-art performance on CIFAR-10 among recent NAS algorithms. This indicates that the architectures discovered by NAS algorithms with proxy data selected from a large-scale dataset are highly transferable to smaller datasets. The contributions of this chapter are as follows:

- We provide substantial experimental results conducted on NAS-Bench-1shot1 to demonstrate that the existing selection is inappropriate for NAS.
- By identifying the characteristics of effective proxy data, we propose a novel selection method and two approaches for implementing it.
- We demonstrate the efficacy of the proposed selection for NAS and its general applicability to various NAS algorithms and datasets, which empirically support that the field of NAS can significantly benefit from the reduced search cost afforded using the proposed proxy data selection.

3.2 Exploration Study

Motivated by the need for an algorithm-agnostic approach to reduce the search cost, we introduce *proxy data*, a representative subset of the target data. In the spirits of the core-set selection, we hypothesize that neural networks discovered by NAS algorithms using proxy data can be competitive to those searched using the entire target data. If the search performance deteriorates when using proxy data, the NAS approach based proxy data is meaningless despite the search cost reduction; thus, it is

important to maintain the search performance of when using the entire dataset, which we call the original search performance. To investigate factors which of proxy data are critical for maintaining the original search performance, we first extensively evaluate search performance using various proxy data constructed by existing selection methods.

3.2.1 Exploration Setting

NAS-Bench-1shot1 [149] is used as the primary testing platform to observe the effect of the proxy data on the search performance of three NAS algorithms on CIFAR-10: DARTS [89], ENAS [105], and GDAS [31]. NAS-Bench-1shot1 [149] is built on the search space and the architecture database offered by NAS-Bench-101 [145], which contains 423K 'architecture and its CIFAR-10 classification accuracy' pairs. To construct proxy data of size k, where we set $k \in \{1.5, 2.5, 5, 10, 15\}$K, k examples among 50K training examples of CIFAR-10 [75] are selected using selection methods. The selected examples are segregated into two parts: one for updating weight parameters and the other for updating architecture parameters (DARTS, GDAS) or a controller (ENAS). For a fair comparison, the same hyperparameter settings as those offered in NAS-Bench-1shot1 are used for all the tested NAS algorithms. To avoid cherry-picking results, we execute the search process five times with different seeds and report the average performance.

The five selection methods utilized in this investigation are: random, entropy top-k, entropy bottom-k, forgetting events, and k-center. Proxy data constructed using these five selections are denoted by D_{random}, D_{top}, D_{bottom}, D_{forget}, and D_{center}, correspondingly. We utilize ResNet-20 [53] as an auxiliary model used for selection methods. Herein, we provide a description of each selection method.

Random selection (D_{random})

Random selection, as the name suggests, samples examples uniformly; this selection function randomly samples k examples according to the uniform distribution. By definition, D_{random} must inherit the characteristics of target data such as data entropy distribution.

Top-k (bottom-k) entropy-based selection (D_{top} and D_{bottom})

These two selection functions [117] are based on the entropy of the predictive distribution of examples, abbreviated by the entropy of examples. To construct D_{top} and D_{bottom}, we calculate the entropy of all examples using a widely-known neural network (*e.g.*, ResNet [53]) pretrained on the target data. The entropy $f_{\text{entropy}}(x)$ of example x using a pretrained model M is calculated as follows:

$$f_{\text{entropy}}(x; M) = -\sum_{\tilde{y}} P(\tilde{y}|x; M) \log P(\tilde{y}|x; M), \tag{3.1}$$

where $\tilde{\boldsymbol{y}} = M(x)$ indicates the predictive distribution of x with respect to M, $\tilde{\boldsymbol{y}} \in R^d$, and d is the output dimension of M; that is, $\tilde{\boldsymbol{y}}$ is the input of softmax in a classifier. Then, after ranking the examples according to their entropy values in a descending order, the top-k examples are selected. Entropy bottom-k selection performs the opposite.

Forgetting events-based selection (D_{forget})

Forgetting events [128] of example x indicate the number of times that x is misclassified after having been correctly classified during the training of a neural network. Before constructing D_{forget}, we train a widely-known neural network from scratch and keep track of the number of forgetting events per each example. After training

the neural network, the examples are ranked based on the number of forgetting events in descending order, and then top-k examples are chosen from the ranked list.

Greedy k-center selection (D_{center})

Intuitively speaking, the k-center problem aims to select k center examples, such that the maximum distance between a given example and its nearest center is minimized [115]. The formal definition of the k-center problem is given as:

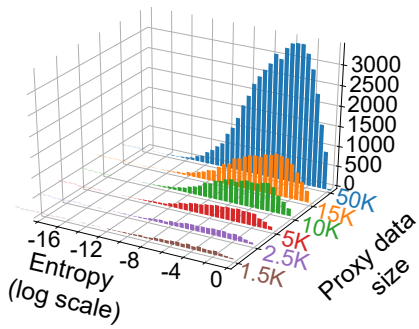
$$\min_{s^1:|s^1|\leq k} \max_{i} \min_{j\in s^1 \bigcup s^0} \Delta(x_i, x_j) \tag{3.2}$$

where s^0 is the initial set of examples and s^1 is the newly selected set of examples. To construct D_{center}, we first extract features from each example by using a pretrained neural network. Afterwards, given the features of examples, we select k number of examples according to a greedy k-center algorithm [115].

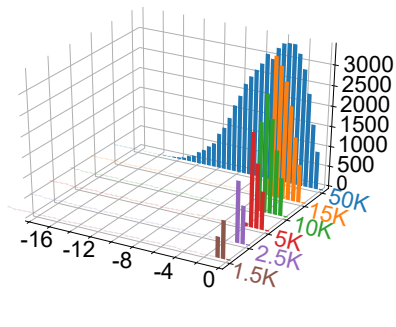
3.2.2 Observations and Analysis

We analyze different proxy data to identify the most significant factor that contributes to the search performance competitive to the original performance using data entropy, f_{entropy}. Data entropy is a typically used metric to quantify the difficulty of an example; furthermore, it is used as the defining property of proxy data in this chapter. The entropy is calculated using Eq. 3.1 with the pretrained ResNet-20. Figure 3.2 shows the histograms of data entropy of all the evaluated proxy data and the entire CIFAR-10 dataset in log scale. The search results of DARTS are shown in Figure 3.3, and those of GDAS and ENAS are provided in Figure 3.4 and Figure 3.5, respectively. In this analysis, we mainly focus on the search results of DARTS.

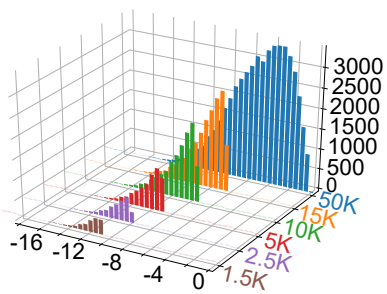
As k changes, two interesting phenomena are observed. First, for $k \leq 2.5$K ex-



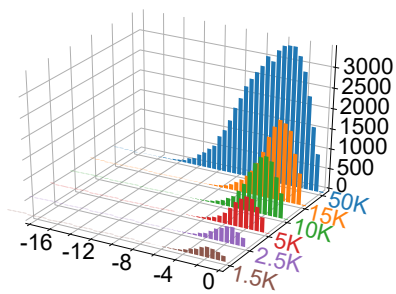
(a) Random selection



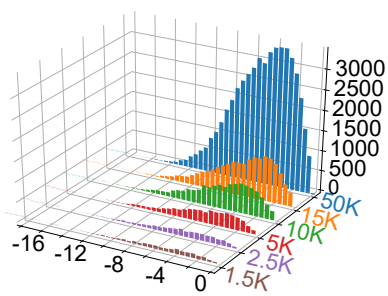
(b) Entropy-based top-*k* selection



(c) Entropy-based bottom-*k* selection



(d) Forgetting events-based selection



(e) *k*-center selection

Figure 3.2: Histograms of data entropy in log scale, where the data is selected by existing selection methods. Blue histograms indicate the entropy distributions of 50K training examples of CIFAR-10 and the others indicate those of various proxy data constructed using existing selection methods.

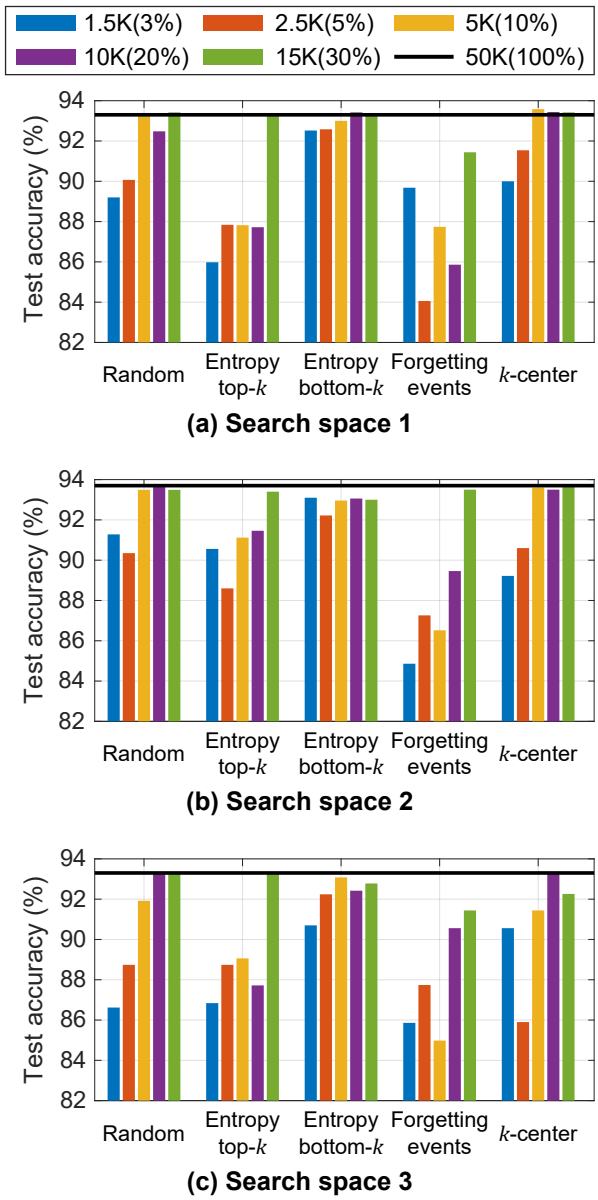


Figure 3.3: Search performance (CIFAR-10 test accuracy) on NAS-Bench-1shot1 (DARTS) with various proxy data.

amples, searching with D_{bottom} consistently yields a search performance closer to that of DARTS with the entire data, *i.e.*, the original performance. On the contrary, when searching with D_{top} and D_{forget}, which have large proportions of high-entropy examples, DARTS struggles to search for performative neural networks in the architecture database. Second, as k increases above 5K, searching with most proxy data results in a rapid increase in the resulting search performance; in the case of D_{bottom}, however, the improvement is less prominent, and the original performance is hardly achieved.

We speculate that the search performance differences come from how difficult examples in proxy data are and how to compose the proxy data. We use the entropy of each example within proxy data as a metric to quantify the difficulty of the proxy data. The difficulty can be estimated by data entropy which reflects uncertainty to a certain model (*i.e.*, a pretrained ResNet-20); the lower entropy values examples have, and the easier examples are. In this chapter, the difficulty of proxy data can be considered as the average of data entropy; $D_{\text{bottom}} < D_{\text{random}} \simeq D_{\text{center}} < D_{\text{forget}} \simeq D_{\text{top}}$.

As shown in Figure 3.2(a)-(e), the composition of D_{bottom} differs significantly from those of the other proxy data. When $k \leq 2.5$K, D_{bottom}, which achieves a more competitive search performance than other proxy data, contains a significantly larger number of easy examples. It suggests that to construct proxy data with a number of easy examples is effective for minimizing the size of desirable proxy data. However, by definition, D_{bottom} is restricted to only select comparatively easy examples. As k increases, the number of difficult examples in the other proxy data increases simultaneously, but by definition, that number for D_{bottom} stays low. The search performance of D_{bottom} does not reach the original performance, indicating that adding easy examples hardly contributes to raise search performance when easy examples are already selected.

Meanwhile, D_{random} (or D_{center}) gradually includes easy, middle, and difficult examples, and most additional examples in D_{forget} (or D_{top}) are difficult. It appears that NAS with the proxy data, which appropriately includes middle and difficult examples, can achieve the original search performance when k is sufficiently large; the appropriate value of k differs for each selection. Comprehensively, based on the observed correlations in the search performance and the compositions in the proxy data, we deduce that selection methods for NAS satisfy the following characteristics:

• When a small number of examples are selected, easy examples are more likely to discover a relatively competitive architecture than difficult examples.

• When easy examples are already selected, adding middle and difficult examples enables the original search performance to be achieved.

Discussion on GDAS and ENAS

The results of GDAS are shown in Figure 3.4. Overall patterns observed in GDAS are similar to those observed in ENAS, confirming again the validity of our analysis on the characteristics of proxy dataset suitable for NAS. As shown in Figure 3.5, the search results of ENAS are different from those of DARTS and GDAS. In most subset configurations, the search performance using the proxy data is higher than that using the entire dataset. It implies that ENAS with our proxy dataset approach can yield higher search performance than the original search performance. However, correlation between the search performance and the proxy data is unclear, and the search performance appears to be less stable compared to DARTS and GDAS. Given a more stable ENAS-based search algorithm, more reliable observations may be drawn from the experiments.

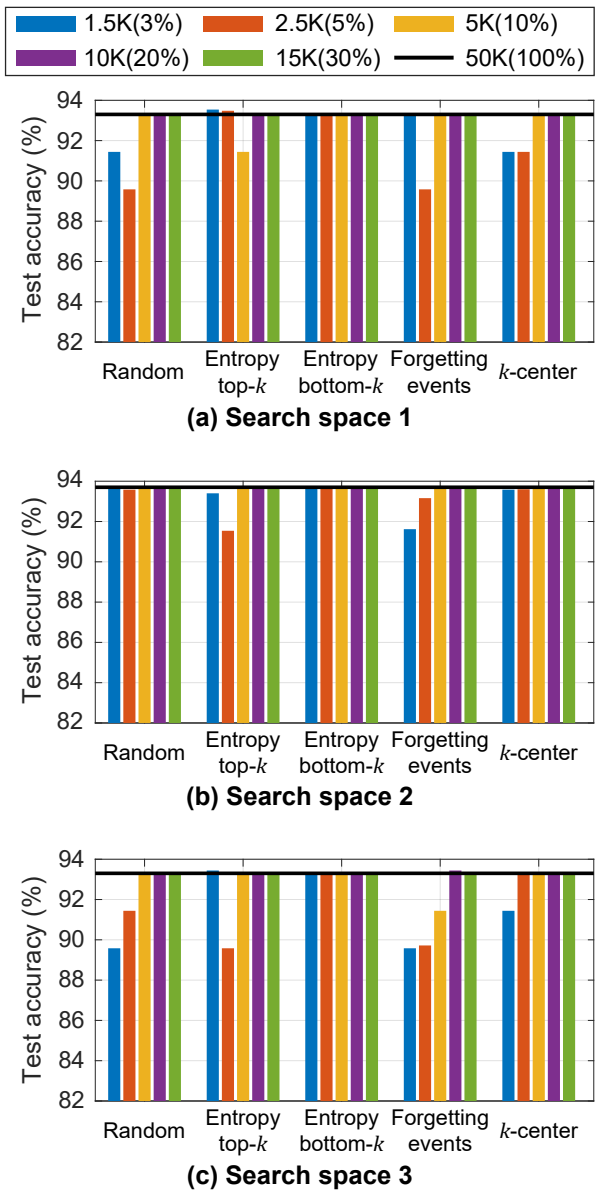


Figure 3.4: Search performance (CIFAR-10 test accuracy) on NAS-Bench-1shot1 (GDAS) with various proxy data.

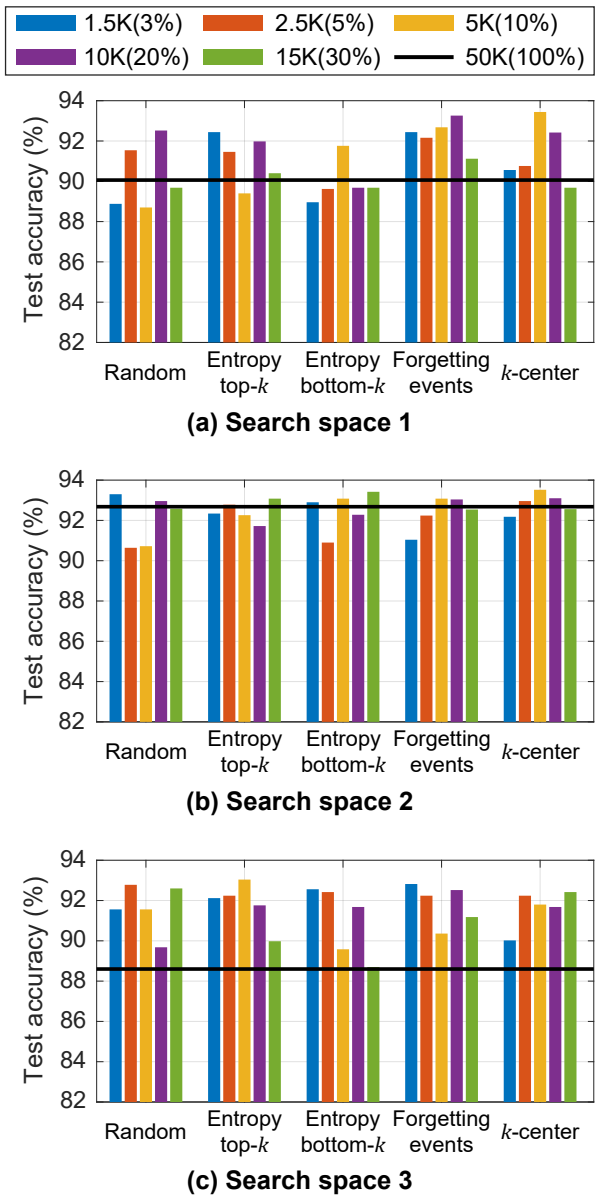


Figure 3.5: Search performance (CIFAR-10 test accuracy) on NAS-Bench-1shot1 (ENAS) with various proxy data.

3.3 Proposed Selection Method

Figure 3.3 shows that the size of the smallest effective proxy data obtained using the existing selection methods is 5K. Although random selection may be considered a strong baseline selection method, its performance deteriorates significantly when $k \leq 2.5$K. In addition, it is noticeable that D_{bottom} with $k \leq 2.5$K achieves the better search performance than the other proxy data. Therefore, to further minimize the size of the proxy data, we propose a new selection method that weighs on examples belonging to both-sided tailed distribution in the data entropy. It is intuitive that the increase in difficult examples provides more information to NAS with easy examples than additional middle examples. As shown in Figure 3.3, this is supported by results of D_{bottom} with $k = 15$K, which includes a large number of easy examples and a small number of middle examples.

Herein, we suggest two methods for implementing the proposed selection method: deterministic and probabilistic methods. For deterministic selection, we introduce the composition ratio parameter β of low-entropy examples. With respect to data entropy, bottom-βk examples and top-$(1-\beta)k$ examples are selected, where $0 < \beta < 1$. For probabilistic selection, the probability distribution of examples should be designed to satisfy the two aforementioned characteristics. Utilizing histogram information, which can be obtained using a pretrained auxiliary model (*e.g.*, ResNet-20), we design and evaluate three probabilities, denoted by P_1, P_2, and P_3, for probabilistic selection.

Let h_x denote a bin where example x belongs in data entropy histogram H and $|h_x|$ denote the height of h_x, *i.e.*, the number of examples in h_x. The three probabilities are defined as follows:

$$P_{\{1,2,3\}}(x; H) = \text{norm}(W_{\{1,2,3\}}(h_x; H)/|h_x|), \tag{3.3}$$

where $\mathrm{norm}()$ normalizes the inside term such that $\sum_{x \in D} P_{\{1,2,3\}}(x; H) = 1$ for target data D. In the inside term, selection weights denoted by $W_{\{1,2,3\}}(h_x; H)$ are defined as follows:

$$W_1(h_x; H) = \frac{\max_{h' \in H} |h'| - |h_x| + 1}{\sum_{h'' \in H} \max_{h' \in H} |h'| - |h''| + 1}, \tag{3.4}$$

$$W_2(h_x; H) = \frac{1}{\text{the number of bins in } H}, \tag{3.5}$$

$$W_3(h_x; H) = \frac{1/|h_x|}{\sum_{h'' \in H} 1/|h''|}. \tag{3.6}$$

In Eq. 3.3 with $W_2(h_x; H)$, which places equal weights on all bins, examples from tail-ends of H are more likely to be selected. W_1 penalizes sampling middle examples by using the difference between height of h_x and the maximum height of the bin near the center in the histogram H. W_3 is defined as the inverse of height of h_x. In summary, W_1 and W_3 further penalize selection of middle examples.

For evaluation on NAS-Bench-1shot1, we execute the proposed deterministic and probabilistic selections using 10 different seeds. Among the deterministic selections with $\beta = \{0.9, 0.8, 0.7, 0.6, 0.5\}$, the search performance with $\beta = 0.9$ is the best. For probabilistic selection, we quantify $|h_x|$ based on a data entropy histogram of CIFAR-10, which is the blue histogram in Figure 3.2.

As shown in Figure 3.6, the deterministic selection with $\beta = 0.9$ and the probabilistic selection based on $P_1(x; H)$ achieve the best search performance in the deterministic and probabilistic selections, respectively. In particular, in search space 1, the $P_1(x; H)$-based probabilistic selection achieves the original performance with only $k = 1.5$K examples. Although the deterministic selection with $\beta = 0.9$ achieves a competitive performance as well, finding the optimal β is nontrivial because the optimal β can be dependent on the target data or pretrained auxiliary models. By contrast,

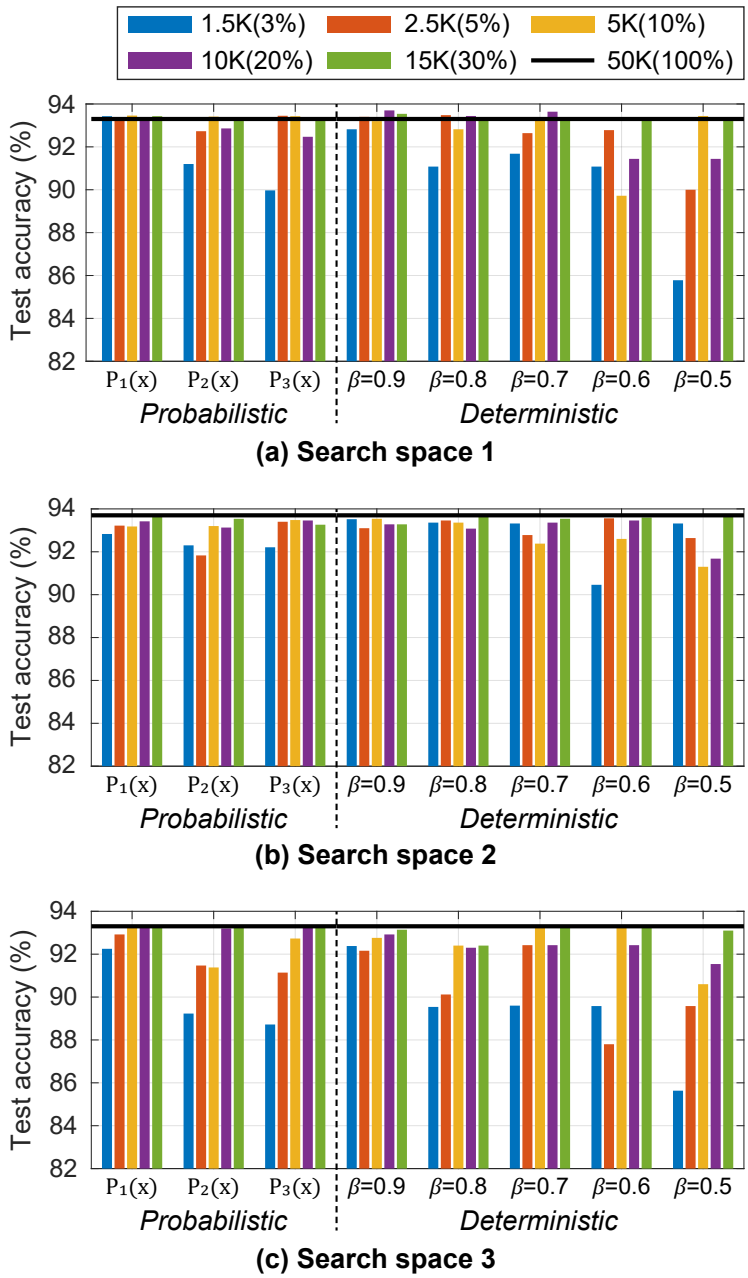


Figure 3.6: Search performance on NAS-Bench-1shot1 (DARTS) using the proposed methods. $P_{\{1,2,3\}}(x)$ are sampling probabilities used in the proposed probabilistic selection, and β denotes the composition ratio parameter of low-entropy examples in the proxy data.

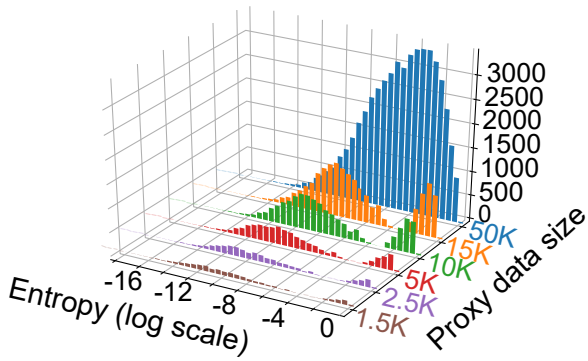


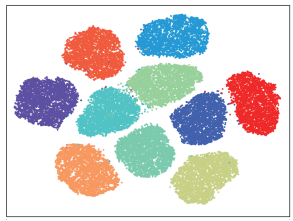
Figure 3.7: Entropy histograms of proxy data that is sampled by the proposed stochastic method with P_1.

probabilistic selection does not require additional hyperparameters; as such, an exhaustive hyperparameter search is not necessary for selecting proxy data. Therefore, we set the $P_1(x; H)$-based probabilistic selection as our main method for the remainder of this chapter.

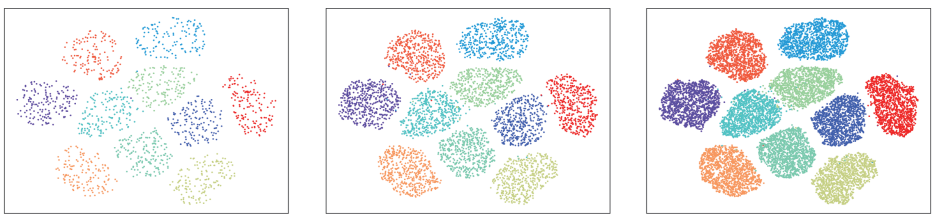
Figure 3.7 shows that the entropy histograms of proxy data selected by the proposed selection method. Compared with the other existing selections (please refer back to Figure 3.3 and Figure 3.6), the proposed proxy data selection method demonstrates better search performance.

3.3.1 Discussion Regarding Efficacy of Proposed Selection

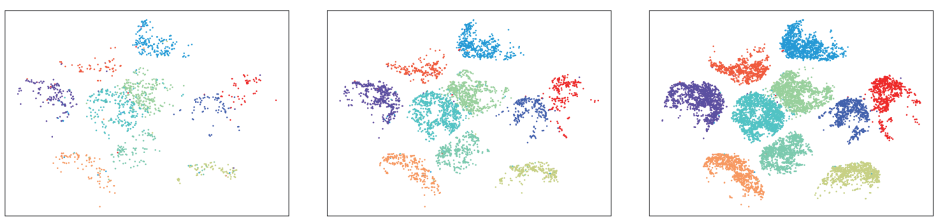
In this section, the factor contributing to the effectiveness of the proposed selection method particularly for NAS is discussed. Many differentible NAS algorithms focus on training a super-network [139]. When a super-network is trained to fit only the easier examples, it will naturally converge faster than when it is attempting to fit difficult examples. The side effect of this phenomenon is that the gradient of the loss will become small only after a few epochs of training [10] and hence will no longer backpropagate useful signals for the super-network. Therefore, when deriving an architecture from such super-network, it is likely that the resulting architecture will



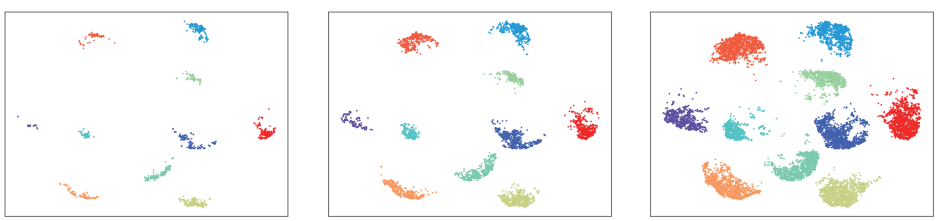
(a) All examples in CIFAR-10



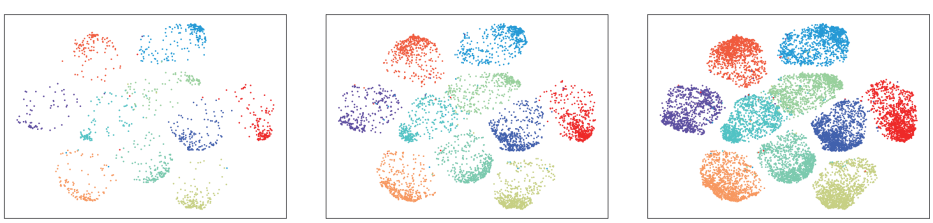
(b) Random selection



(c) Entropy-based top-*k* selection



(d) Entropy-based bottom-*k* selection



(e) Proposed probabilistic selection

Figure 3.8: Visualization of t-SNE: for each selection method, (left) 1.5K (3%), (middle) 5K (10%), (right) 10K (20%) examples selected from 50K training examples of CIFAR-10.

have limited generalization capacity to difficult examples. Using difficult examples allows the super-network to learn more meaningful information, which is difficult to be obtained from easy examples. Using the t-SNE [94] visualization of different proxy data, we can speculate that the missing information from the easy examples is related to the decision boundaries obtained from the pretrained auxiliary network and the dataset. ResNet-20 is used to extract features from different proxy data of CIFAR-10 and the corresponding t-SNE visualization results are shown in Figure 3.8. The easy examples tend to be distant from the decision boundaries, unlike the difficult ones. Therefore, for a super-network to learn such decision boundaries, proxy data with difficult examples is required.

Meanwhile, if proxy data is primarily comprised of difficult examples, the stable training of a super-network may be hindered, which is consistent with the results of D_{top} in Figure 3.3. This issue can be resolved using a sufficient number of easy examples. Consequently, the proposed selection method that satisfies the characteristics identified in Section 3.2 yields a super-network that is similar to that trained with the entire dataset while the size of the proxy data is minimized.

3.4 Experiments and Results

3.4.1 Experimental Settings

We used the pretrained ResNet-50 in Pytorch model zoo for ImageNet [111], and trained the three models for CIFAR-10 [75], CIFAR-100 [75], and SVHN [100]; the training took 0.015, 0.033, and 0.022 GPU days, where the additional cost is negligible compared with the search cost of existing NAS methods. We prepared proxy data using log-scaled data entropy histograms in Figure 3.9. Although the data entropy distributions of CIFAR-100 and ImageNet show different patterns than those

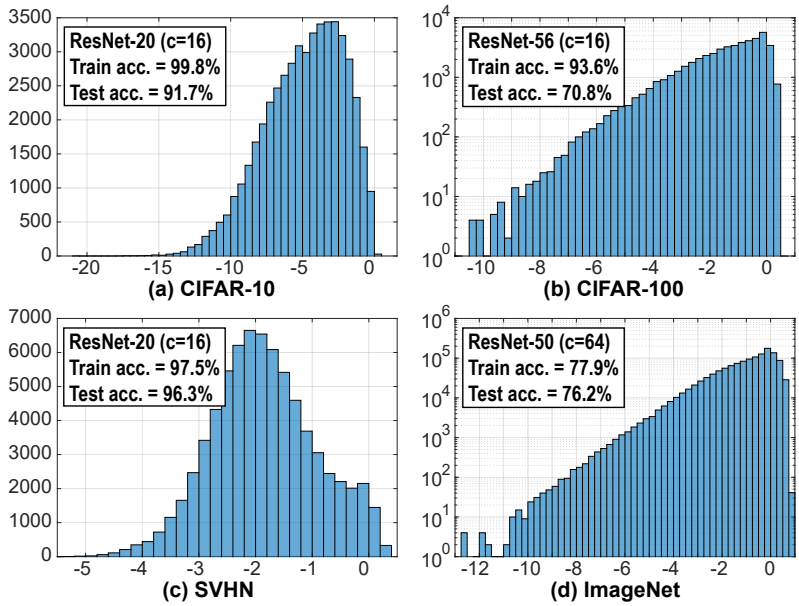


Figure 3.9: Histograms of data entropy (log scale in x-axis). In each histogram, **c** in the box located at the top-left of each figure indicates the number of filters in the first convolutional layer in each ResNet. The histograms from (a) to (d) are obtained by setting a bin width as 0.5, 0.25, 0.2, and 0.25, respectively.

of CIFAR-10 and SVHN, our experimental results consistently indicate that the proposed proxy data selection is valid for CIFAR-100 and ImageNet, as it is for the other two datasets.

In Section 3.4.2, before applying our proposed proxy data selection method to existing NAS methods, we further conduct experiments using three algorithms of NAS-Bench-1shot1: DARTS, GDAS, and ENAS. Then, Section 3.4.3 provides the robustness of the proposed selection method to hyperparameter changes. For all the experiments conducted on NAS-Bench-1shot1, we used a single NVIDIA GeForce RTX 2080ti GPU. In the remainders of Sections 3.4 and Section 3.5, the searching results using the existing NAS methods based on the cell-based search space with the proxy data is presented. For the evaluation using existing NAS methods, we used two types of NVIDIA GPUs: Tesla V100 for searching and training neu-

ral architectures on ImageNet, and GeForce RTX 2080ti for the remaining datasets. The code for execution of NAS algorithms with proxy data is publicly available at https://github.com/nabk89/NAS-with-Proxy-data. Detailed search and retraining processes we adopted are described as follows.

Search

On CIFAR-10, CIFAR-100, and SVHN, using a single NVIDIA GeForce RTX 2080ti GPU, we execute search process of differentiable NAS algorithms following protocols reported in each NAS algorithm. These protocols are commonly based on DARTS [89]. The super-network is constructed by stacking eight cells (six normal cells and two reduction cells) including mixed operations proposed by DARTS, after a 33 convolution-based stem layer with an initial channel of 16. Search process is executed for 50 epochs, with a batch size of 64 (exceptionally, 224 in PC-DARTS [142]). Weight parameters of the super-network are optimized by momentum, with an initial learning rate of 0.1, a momentum of 0.9, and a weight decay of 0.0003, where the learning rate is annealed down to zero with a cosine schedule [91]. Architecture parameters α are optimized by the Adam optimizer [70], with a fixed learning rate of 0.0006, a momentum of (0.5, 0.999), and a weight decay of 0.001. During search process, the standard data preprocessing and augmentation techniques were used: the channel normalization, the central padding of images to 4040 and then random cropping back to 3232, random horizontal flipping.

For ImageNet, we execute the search process on a single Tesla V100 GPU. The super-network is nearly identical to the aforementioned super-network with slight modifications. To fit the ImageNet search process on GPU memory, stem layers in the super-network consist of three 3×3 convolution layers of stride 2, and result in that the input data resolution of the first cell is 28×28, which is reduced from 224×224.

The batch size is set as 256 and the other hyperparamters are not changed.

Retraining for a searched architecture

To train a searched architecture using CIFAR-10, we also follow the retraining settings provided in DARTS [89]. The network is constructed by stacking 20 cells (18 normal cells and two reduction cells, each type of which shares the same architecture discovered in search process) after a 3×3 convolution-based stem layer with an initial channel of 36, including an auxiliary loss. The network is trained from scratch for 600 epochs using training examples, with a batch size of 96; several architectures, which cannot use the batch size of 96 due to memory size, are trained with a lower batch size. The momentum SGD optimizer is used, with an initial learning rate of 0.025 following cosine scheduled annealing, a momentum of 0.9, a weight decay of 0.0003, and a norm gradient clipping at 5. In addition to the data augmentation techniques used in the search process, cutout [26] is additionally used, and drop-path with a rate of 0.2 is used for regularization.

For ImageNet, we used two V100 GPUs to train the network, which is constructed by stacking 14 cells (12 normal cells and two reduction cells, each type of which shares the same architecture discovered in search process) after the stem layers consisting of three 3×3 convolution layers of stride 2, with an initial channel of 48 and an auxiliary loss. The network is trained from scratch for 250 epochs using training 1.28M examples of ImageNet, with batch size of 640. The momentum SGD optimizer is used, with an initial learning rate of 0.5 which is decayed down to zero linearly, a momentum of 0.9, a weight decay of 0.00003, and a norm gradient clipping at 5. During the first five epochs, the learning rate warm-up is applied. Label smoothing [125] with a rate of 0.1 is used to enhance the training.

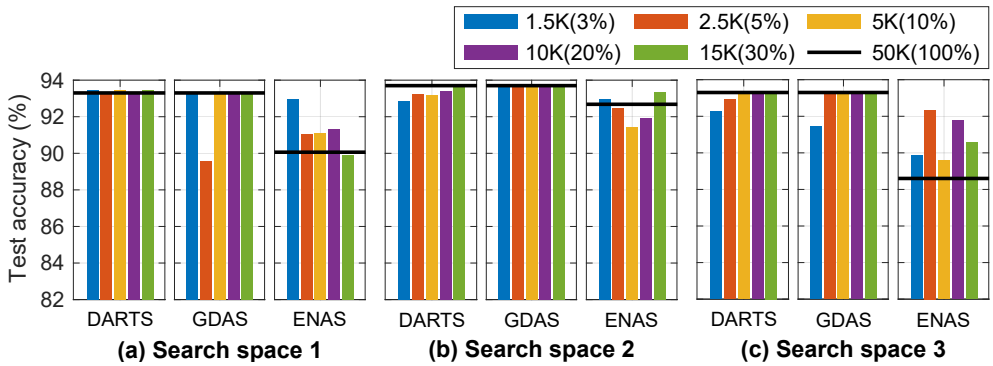


Figure 3.10: Search performance of DARTS, GDAS, and ENAS on NAS-Bench-1shot1 using the proposed methods.

3.4.2 NAS-Bench-1shot1

We evaluate three NAS algorithms with the proposed probabilistic selection method, and the results are visualized in Figure 3.10. In DARTS and GDAS, our approach using proxy data is effective to achieve their original search performance (black line in each figure of Figure 3.10). The search performance of ENAS appears to be relatively unstable, as in the cases of the other proxy data. Nevertheless, it is also effective to experience the same and even better search performance than the original one.

3.4.3 Hyperparameter Sensitivity

We analyze the sensitivity to the bin width in the proposed probabilistic method. As shown in Figure 3.7, the default bin width used for the CIFAR-10 histogram is 0.5. For the investigation, we sample additional two proxy data from CIFAR-10, by setting the width to be 0.25 and 1. Figure 3.11 shows the resulting histograms, whose general pattern is unaffected by change in the bin width. In addition, Figure 3.12 reveals that the search performance of our proposed method is hardly sensitive to the choice of a bin width on all the search spaces.

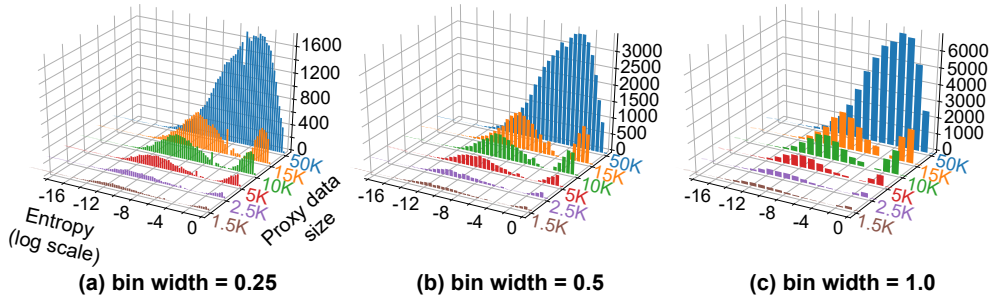


Figure 3.11: Histograms of data entropy in log scale, where the data is selected by the proposed probabilistic selection methods with three bin widths. Blue histograms indicate the entropy distributions of 50K training examples of CIFAR-10.

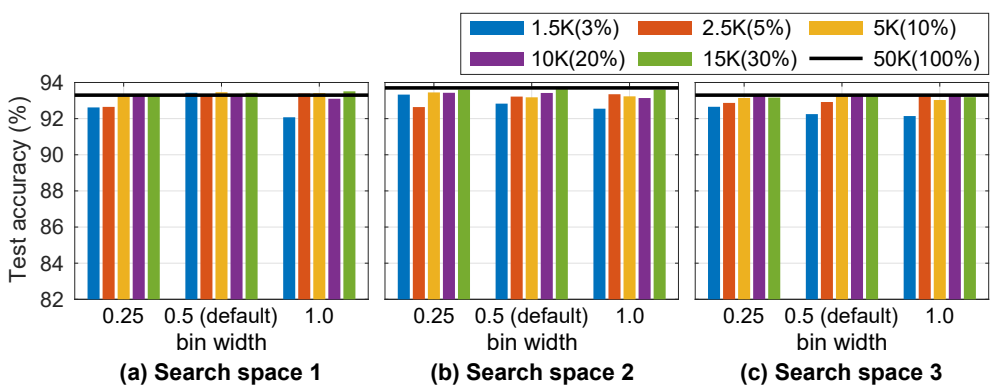


Figure 3.12: Search performance on NAS-Bench-1shot1 (DARTS) using the proposed probabilistic selection method, given data entropy histograms with three bin widths.

Table 3.1: Evaluation (test error (%)) of DARTS with varying sizes of proxy data of CIFAR-10 and ImageNet: random selection vs. the proposed probabilistic selection.

Selection	**CIFAR-10**					**ImageNet**
	5K	**10K**	**15K**	**20K**	**25K**	**128K**
Random selection	3.21	2.95	2.99	2.72	3.22	25.2
Proposed probabilistic selection	2.94	2.92	2.88	2.78	2.76	24.6

3.4.4 Comparison with Random Selection

Now, behind the evaluation on NAS-Bench-1shot1, we provide results of search algorithms working on the cell-based search space. Based on Section 3.2, it is apparent that random selection is an effective, reasonable baseline. Hence, we compare the proposed selection method with the random selection in the cell-based search space using DARTS [89] with CIFAR-10 and ImageNet.

As shown in Table 3.1, on CIFAR-10, searching with the proposed selection method is usually superior to that using random selection. Furthermore, the search performance with the random selection fluctuates with varying sizes of proxy data. The result of searching with 128K training examples chosen from ImageNet by the proposed selection, is also superior to that of random selection.

3.4.5 Applicability to NAS Algorithms

Recently, various differentiable NAS algorithms based on the cell-based search space and DARTS [89] have been proposed [139]. We apply the proposed proxy data selection to the recently proposed NAS algorithms: DARTS [89], PC-DARTS [142], SDARTS [14], SGAS [81], and EcoDARTS that is a DARTS-based variant of EcoNAS [157], respectively.

Table 3.2 reveals that all of the tested NAS algorithms achieve the comparable performance to their respective original search performance. While on CIFAR-10,

Table 3.2: Evaluation of various NAS methods used on cell-based search space using proposed proxy data selection.

NAS algorithm	Base (entire dataset)		Proposed (proxy data)	
	Test error (%)	Cost*	Test error (%)	Cost*
CIFAR-10 (Base: 50K, Proposed: 5K)				
DARTS [89]	3.00	0.26	2.94	0.03
PC-DARTS [142]	2.67	0.08	2.91	0.01
EcoDARTS-c4r2 [157]	2.80	0.23	2.81	0.02
SDARTS-RS [14]	2.67	0.23	2.83	0.03
SGAS-Cri.1 [81]	2.66	0.19	2.72	0.02
ImageNet (Base: 1.28M, Proposed: 128K)				
DARTS [89]	26.7	-	24.6	0.32
PC-DARTS [142]	24.2	3.8†	24.3	0.26

*Search cost is GPU days and single 2080ti GPU and V100 GPU are used for searching on CIFAR-10 and ImageNet, respectively.
†Authors of PC-DARTS reported that search process on ImageNet required 11.5 hours with eight V100 GPUs, *i.e.*, 3.8 GPU days.

PC-DARTS with the proposed selection method experiences a slight decrease in performance, on ImageNet, it successfully achieves the original search performance with significantly reduced search cost. We observe that the search cost decreases proportionally to the size of proxy data; if 10% of target data is used as a proxy data, the search cost reduces by approximately 10 times.

None of the existing NAS algorithms on the cell-based search space searched directly on ImageNet, with the exception of PC-DARTS. To perform the direct search on ImageNet, we incorporate the proposed selection with PC-DARTS and DARTS. DARTS with the proposed selection discovers a better architecture than the original DARTS. The original PC-DARTS searched on ImageNet with 12.5% of examples randomly sampled from the dataset, where 10% and 2.5% of examples are used for updating the weights of operations and the architecture parameters, respectively. Xu *et al.* [142] reported that the search process of PC-DARTS required 3.8 GPU days, *i.e.*, 11.5 hours with eight V100 GPUs, with a batch size of 1024; we speculate that

Table 3.3: Evaluation using proposed proxy data with $k \in \{1.5, 2.5, 5\}$K.

NAS algorithm	**Test error (%)**		
	1.5K (3%)	2.5K (5%)	5K (10%)
DARTS [89]	3.02	2.63	2.94
SDARTS-RS [14]	3.29	3.05	2.83
SGAS-Cri.1 [81]	2.76	2.75	2.72

the parallel execution on the eight GPUs resulted in a non-negligible overhead. By contrast, in our experiments, the search process of PC-DARTS was executed with the proxy data that consists of 10% of examples constructed using the proposed selection the dataset is evenly divided, unlike the PC-DARTS setting. PC-DARTS based on our approach can discover the competitive architecture using a single V100 GPU with a batch size of 256 in approximately 0.26 GPU days, *i.e.*, 14.6 times less cost than that of the original PC-DARTS.

We further evaluate the validity of proposed proxy data, which is significantly small to be used in real NAS algorithms such as $k = \{1.5, 2.5\}$K, using three cell-based differentiable NAS algorithms: DARTS, SDARTS, and SGAS. Table 3.3 reveals that the performances of cells searched by SDARTS decrease as k becomes smaller, while the performance gap is relatively small in DARTS and SGAS. SDARTS employs a perturbation-based approach that architecture parameters α is perturbed by adding some sampled noise during training the super-network cell, but DARTS and SGAS optimize α in the deterministic approach. We conjecture that a sampling approach for training α needs diverse examples for obtaining stable search results and thus does not take an advantage of using such significantly small proxy data.

3.4.6 Applicability to Datasets

To further demonstrate the general applicability of the proposed selection method to datasets, we test it on CIFAR-10, CIFAR-100, and SVHN in four restricted different

Table 3.4: Evaluation (test error (%)) in four restricted cell-based search spaces and three datasets.

NAS algorithm	Data	Search space	Base 100%	Proposed 10%	Proposed 20%
DARTS [89]	CIFAR-10	S1	3.84	3.60	2.96
		S2	4.85	3.54	3.46
		S3	3.34	2.71	2.72
		S4	7.20	6.60	5.82
	CIFAR-100	S1	29.46	26.41	28.79
		S2	26.05	21.65	22.66
		S3	28.90	22.10	23.51
		S4	22.85	98.91	25.74
	SVHN	S1	4.58	3.12	4.11
		S2	3.53	2.81	3.03
		S3	3.41	2.77	3.11
		S4	3.05	3.06	2.42
RobustDARTS (L2) [148]	CIFAR-10	S1	2.78	2.79	2.86
		S2	3.31	3.33	2.98
		S3	2.51	2.74	2.80
		S4	3.56	3.41	3.43
	CIFAR-100	S1	24.25	26.13	23.67
		S2	22.24	22.21	21.39
		S3	23.99	21.71	22.31
		S4	21.94	27.83	21.10
	SVHN	S1	4.79	2.46	2.60
		S2	2.51	2.45	2.49
		S3	2.48	2.53	2.42
		S4	2.50	5.16	2.62

search spaces [148]. These search spaces were modified from the cell-based search space by reducing the types of candidate operations (**S1**-**S3**) and inserting harmful noise operation (**S4**); a set of candidate operations for each search space is as follows.

- **S1:** Candidate operations are top-2 operations identified using DARTS search process, and thus differs in all edges.
- **S2:** Candidate operations are $\{3\times3$ separable convolution $(\times2)$, skip connection$\}$.
- **S3:** Candidate operations are $\{3\times3$ separable convolution $(\times2)$, skip connection, zeroize$\}$.
- **S4:** Candidate operations are $\{3\times3$ separable convolution $(\times2)$, noise$\}$. The noise operation generates a random Gaussian noise $\mathcal{N}(0, 1)$ whose size is identical to the input. The noise operation must not be included in a searched architecture, and thus the purpose of the insertion of the noise operation is to evaluate the search robustness of NAS algorithms.

We use DARTS and RobustDARTS [148] for the evaluation of the applicability to datasets. Following the experimental protocols in RobustDARTS, the weight decay factors for DARTS and RobustDARTS (L2) during the search process are set to be 0.0003 and 0.0243, respectively.

As shown in Table 3.4, most results of the two NAS algorithms using the proposed selection method are within a reasonable range of the original search performance. However, when DARTS is executed on **S4** with 10% of examples from CIFAR-100, a significant search failure occurs. This failure is caused because noise operations in **S4** occupy most of the edges in the cell structure after the search process. Note that the noise operation is intended for inducing failure in DARTS [148] and is generally not used in practice; thus, this result appears to be caused by the

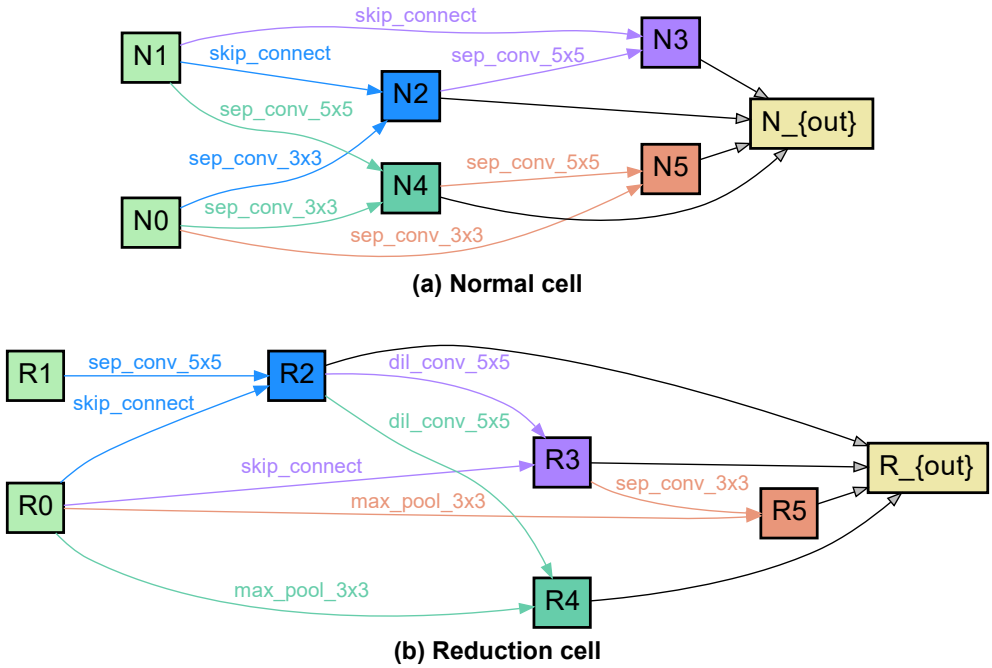


Figure 3.13: Normal and reduction cells searched by DARTS with ImageNet proxy data (10%) sampled by the proposed probabilistic selection method.

limitation of DARTS. Nevertheless, the original search performance on CIFAR-100 can be obtained when using 20% of examples.

3.5 Discussion

3.5.1 Inverse Transferability

Typically, in most NAS algorithms, the transferability of architectures discovered using CIFAR-10 is demonstrated by their performance on ImageNet. Using DARTS with the proposed selection method, the computational cost of searching with ImageNet is reduced by $\frac{1}{10}$, *i.e.*, 0.32 GPU days. The resulting search time on the proxy data of ImageNet is similar to those of other NAS algorithms on the entire CIFAR-10. Therefore, granted the similar amount of search cost for fair comparison, architec-

tures discovered on ImageNet using the proposed selection method can be transferred and evaluated on CIFAR-10, which is the inverse way from conventional studies.

The architecture searched on ImageNet using the proposed selection method is visualized in Figure 3.13. This architecture yields a test error of **2.4%** on CIFAR-10. It is noteworthy that we do not utilize additional techniques introduced in recent studies of NAS, and that the architecture above is discovered only by executing DARTS on the proxy data of a large-scale dataset. We speculate that the use of ImageNet provides DARTS with more helpful visual representations than CIFAR-10. We refer to such an approach of transferring an architecture from a large-scale dataset to a smaller dataset as *inverse transfer*. Our study reveals that the search cost on a large-scale dataset can be reasonably low, and then the inverse transfer of an architecture can provide new directions for NAS research.

3.5.2 Combination of Easy and Moderate Examples

To demonstrate that our approach is superior to using the combination of easy and middle examples (instead of difficult examples) selected in a deterministic manner, we provide NAS-Bench-1shot1 results obtained by using such combination. For $\beta = \{0.9, 0.8\}$, bottom-βk examples are selected as easy examples, and then $(1 - \beta)k$ examples are randomly selected from the remaining subset. As shown in Figure 3.14, the search performances of these settings are lower than those of both probabilistic and deterministic selection methods that we propose. Based on our discussion in Section 3.3.1, we believe that this happens because middle examples sampled by random selection contain less meaningful information for NAS than difficult ones.

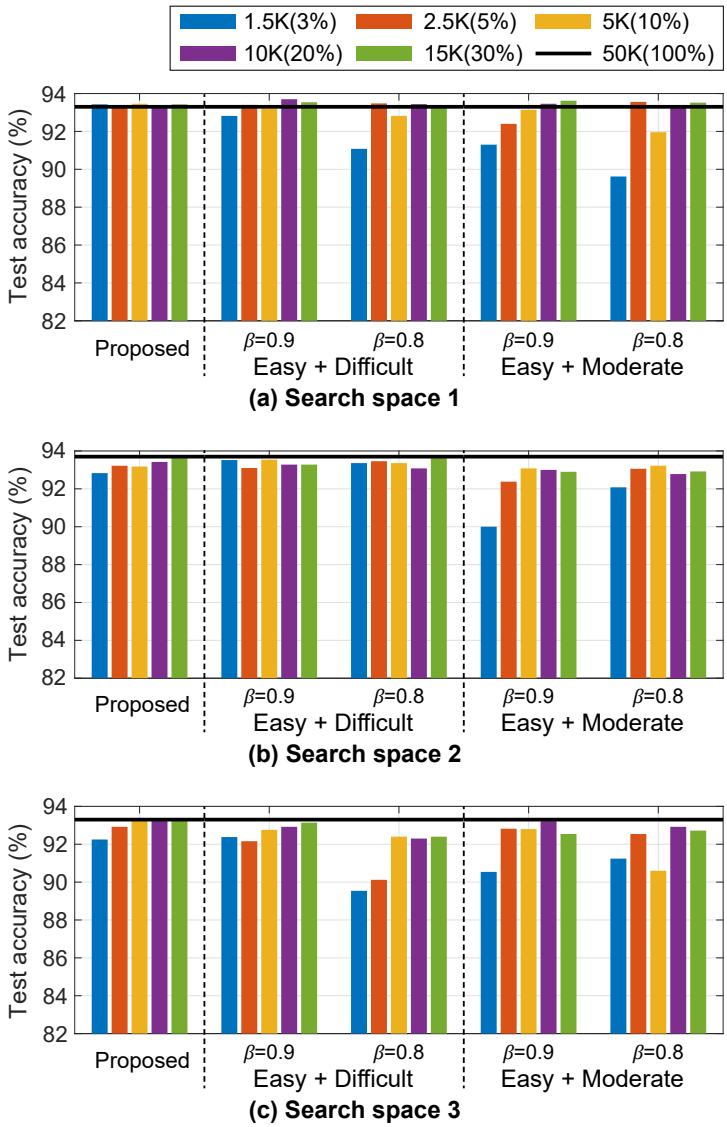


Figure 3.14: Search performance on NAS-Bench-1shot1 (DARTS) using proxy data that consists of easy and moderate examples, which are selected by the entropy-based bottom-k selection and random selection, respectively.

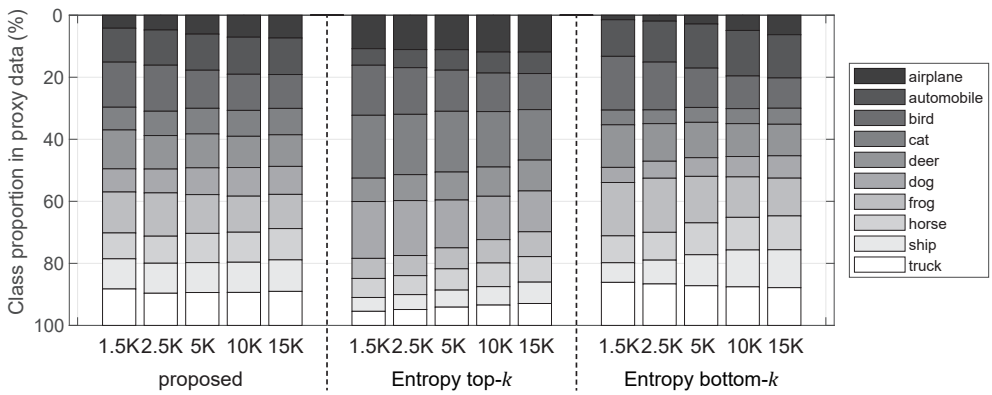


Figure 3.15: Class proportion in all the proxy data.

3.5.3 Class Balance in Proxy Data

Figure 3.15 presents class proportion of three selection methods: the proposed selection, entropy-based top-k and bottom-k selection. The class proportion of these selection methods are skewed; in entropy-based selection methods, the higher skewness is observed. To investigate effects of such class skewness in search performance, we constructed class-balanced proxy data. For each class, examples are selected according to the same algorithm of the respective selection methods.

The results on NAS-Bench-1shot1 (DARTS) are presented in Figure 3.16. In D_{bottom}, the class balance has little effect on search performance regardless of sizes. In contrast, in D_{top} and the proposed proxy data, the class balance has negative effect on search performance. Therefore, we deduce that entropy distribution of examples in proxy data is more important than the class balance to maintain the search performance of NAS algorithms.

3.5.4 Optimization Method of Pretrained Models

Recently, self-supervised learning has received considerable interest, because it can relieve labeled data which requires the manual effort to annotate labels of data. Our

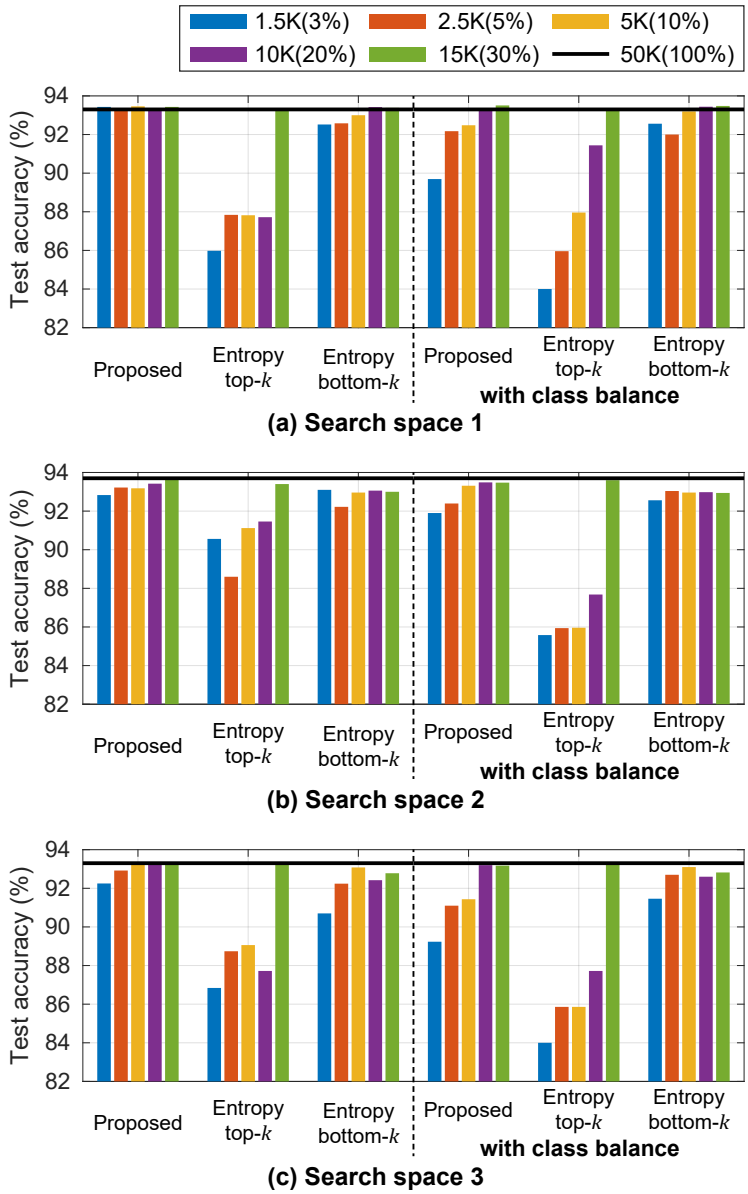


Figure 3.16: Search performance on NAS-Bench-1shot1 (DARTS) with taking class balance into consideration.

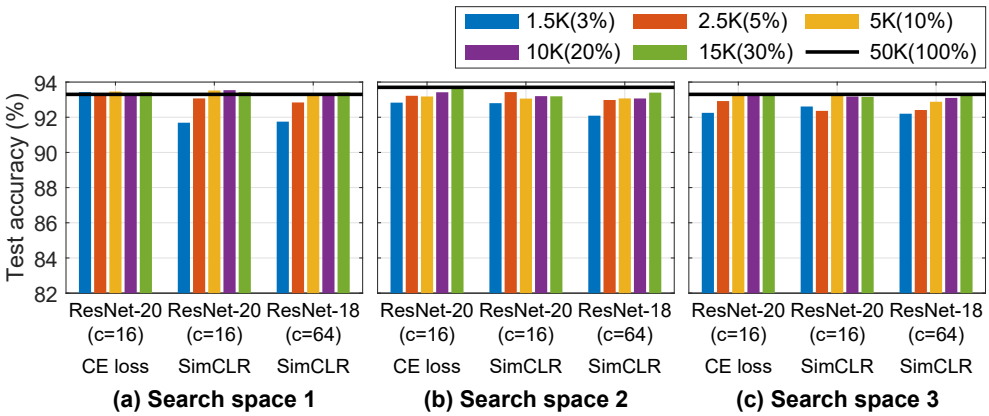


Figure 3.17: Search performance on NAS-Bench-1shot1 (DARTS) using auxiliary networks, which are pretrained by SimCLR [12] to calculate data entropy.

proxy data selection method is based on the visual representations learned by the supervised learning method using cross-entropy loss for a classification task with labeled data. We further conduct experiments to investigate the effect of the optimization methods of pretrained auxiliary models used for calculating data entropy. Using SimCLR [12], one of the well-known contrastive self-supervised learning methods, we pretrain two networks with CIFAR-10: ResNet-20 with an initial channel of 16 (which is the same architecture used for the other experiments) and ResNet-18 with an initial channel of 64. In Eq. 3.1 in Section 3.2, the data entropy calculation needs the input of softmax in a classifier. Hence, we train an additional fully connected layer while freezing the pretrained networks; test accuracies of ResNet-20 and ResNet-18 are 68.3% and 85.9%, respectively.

Figure 3.17 reveals that our proxy data selection method is hardly influenced by the optimization strategy of the pretrained auxiliary networks. For $k = 1.5$K in search space 1, when using data entropy calculated by two SimCLR-based pretrained models, our method experiences approximately performance drop by 1.7%; nevertheless, this result is higher than that of the other existing selection methods. Because the

harsh sampling condition, *i.e.*, using only 3% of target data, may be not adopted in practical, the evaluation results on NAS-Bench-1shot1 with models based on self-supervised learning indicate the model-agnostic property of our method.

3.6 Summary

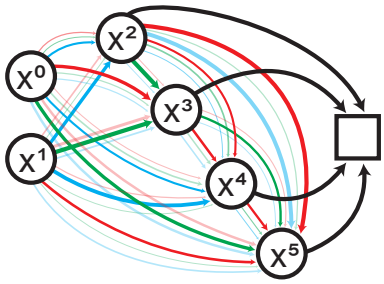
In this chapter, for the first time in NAS research, we introduced proxy data for accelerating NAS algorithms without sacrificing search performance. After evaluating existing selection methods on NAS-Bench-1shot1, we obtained the following insights regarding proxy data suitable for NAS: 1) when a small number of examples are selected, easy examples contribute more to stabilizing the search performance than difficult examples, and 2) when using a sufficient number of easy examples, adding difficult examples is crucial for achieving the original search performance. Based on our findings, we proposed a novel selection method for NAS, which prefers examples in tail-end of entropy distribution of the target data. We also presented its two implementations, *i.e.*, deterministic and probabilistic selection methods. We thoroughly demonstrated the NAS acceleration and applicability of the proposed probabilistic selection method on various datasets and NAS algorithms. Notably, a direct search on ImageNet was completed in 7.5 GPU hours, suggesting that the inverse transfer approach is valid. We expect other studies on NAS to benefit from the significant reduction in the search cost through the use of proxy data.

Chapter 4

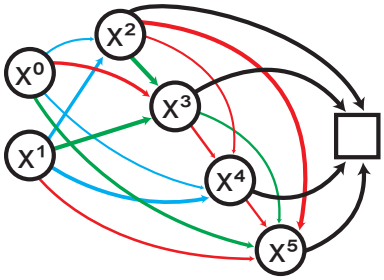
Improving Neural Architecture Search through Global Competition

4.1 Introduction

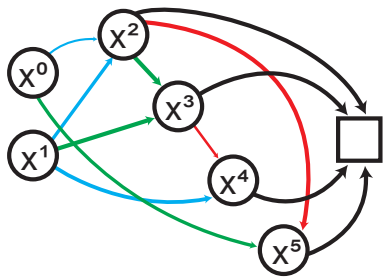
A general NAS formulation consists of two basic components: a search space and a search algorithm [139]. A cell-based search space, referred to as the DARTS cell space [89], which is popularly used across existing NAS methods for vision tasks [139], has been thoroughly studied using diversified search algorithms built upon weight-sharing [105] and differentiable approaches with continuous relaxation [89]. This family of differentiable NAS methods utilizes a super-network cell that is continuously relaxed by employing a set of operation strengths. Through continuous relaxation, various cells are approximately evaluated during the search process, and the operation strengths are optimized via gradient-based optimization. Once the search process ends, it is necessary to derive a cell from the optimized super-network with



(a) Super-network cell with continuous operation strengths

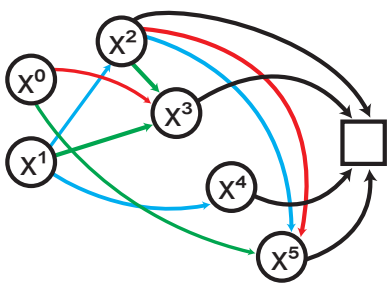


b-1) argmax selection rule



b-2) heuristic post-processing

(b) Conventional cell derivation (DARTS cell)



global top-k survival rule

(c) Proposed cell derivation (ExtCell)

Figure 4.1: Overview of cell derivations. (a) A super-network cell with optimized operation strengths, where the width of arrows represents the strength magnitude of the corresponding operations and candidate operations differ from each other by the color of arrows. (b) Derivation of a DARTS cell from the super-network cell through the argmax selection rule (b-1) and heuristic post-processing (b-2). (c) Derivation of an ExtCell from the super-network cell through the proposed global top-k survival rule, where $k = 8$ in this example.

continuous operation strengths using a selection rule.

Bounded by the definition of the DARTS cell space, most methods adopt the argmax selection rule to select the operation with the maximum strength between two nodes in the cell, and they require an additional heuristic post-processing step to leave two incoming edges for each node (Figure 4.1(b)). The conventional NAS working on the DARTS cell space can thus be interpreted as local competition among candidate operations for occupying an edge inside a cell. Although the continuous relaxation of the discrete search space enables differentiable NAS methods to simultaneously evaluate a diverse set of cells that have multiple operations in an edge, the argmax selection rule inevitably discards many of the cells that do not fit into the DARTS cell space. In this chapter, to explore the discarded cells, we propose to extend the definition of the DARTS cell space to enable multiple operations between two nodes to be selected, and refer to the new search space as the *ExtCell space*. We demonstrate that the quality of the ExtCell space is comparable to that of the DARTS cell space by using a recently developed search space quality quantification tool [106].

Because the argmax selection rule adopted in the conventional NAS algorithms allows only one operation to be activated between two nodes, it is incapable of deriving a cell that belongs in the ExtCell space. Therefore, we introduce a new selection scheme, which we call the top-k survival rule, to differentiable NAS. Given the optimized operation strengths, the top-k survival rule selects operations with top-k strengths from all operations optimized in the super-network cell (Figure 4.1(c)); with this new selection rule, NAS can be interpreted as a global competition problem.

For the operation survival problem, we propose BtNAS, a novel differentiable search algorithm that individually optimizes operation strengths of the super-network cell. BtNAS aims to find the true posterior distributions of the operation strengths. Each operation strength is a continuous random variable that follows a Beta distri-

bution, whose sample ranges from 0 to 1. Because these true posterior distributions are intractable, we utilize the variational Bayes method to approximate them through variational Beta distributions [6, 71]. Because α and β cannot be directly optimized due to the sampling process, to enable gradient-based optimization with respect to α and β, we employ pathwise derivative estimator [60]. In BtNAS, a stochastic sampling of continuous operation strengths makes the search process reliable by preventing multi-model forgetting [5, 150] and smoothing the loss landscape with respect to operation strengths [14, 15].

When existing NAS methods are coupled with the top-k survival rule, we observe that they derive inferior ExtCells with much lower accuracies than the cells obtained by the same NAS methods from the DARTS cell space. This empirical result necessitates the development of a search algorithm that can work in conjunction with the top-k survival rule on the ExtCell space. We demonstrate that BtNAS derives ExtCells competitive to recently reported DARTS cells, whose performance has been improved over time through various search algorithms. Also, compared with the DARTS cells, in the ExtCells, the number of operations in the normal cell decreases, while that in the reduction cell increases, resulting in a smaller network. As an approach to improve cell derivation, we adjust operation strengths to make all intermediate node activated, and obtain a remarkable result, *i.e.*, test error of 2.3% on CIFAR-10. The contributions of this chapter are summarized as follows:

- We introduce the ExtCell space extended from the DARTS cell space to reduce the number of discarded cells that are simultaneously evaluated through continuous relaxation in differentiable NAS. To derive an ExtCell from the optimized super-network cell, we propose the top-k survival rule.

- To effectively explore the ExtCell space, we propose BtNAS that individually

models the operation strengths using variational Beta distributions and approximates these true posterior distributions using the variational Bayes method and the gradient-based optimization.

- BtNAS derives ExtCells with state-of-the-art performance and fewer parameters, indicating that a non-negligible number of desirable cells do exist outside the DARTS cell space.

4.2 Proposed Search Space: ExtCell Space

Differentiable NAS methods utilizing mixed-operations during the search process approximately evaluate diverse cells that can be derived from the super-network cell. However, by the argmax selection rule, cells that do not belong to the DARTS cell space but may yield desirable performance are discarded without further consideration. If the DARTS cell space is expanded to include more cell structures that do not match the DARTS cell definition, novel cell structures can be discovered. In this section, we first introduce an extension of the DARTS cell space, and we subsequently discuss the quality of the extended search space; the DARTS cell space is defined in Section 2.1.1. Finally, instead of using the argmax selection rule, we propose to use an appropriate selection rule to derive such cells from the super-network cell; this rule is named the top-k survival rule.

4.2.1 ExtCell Space

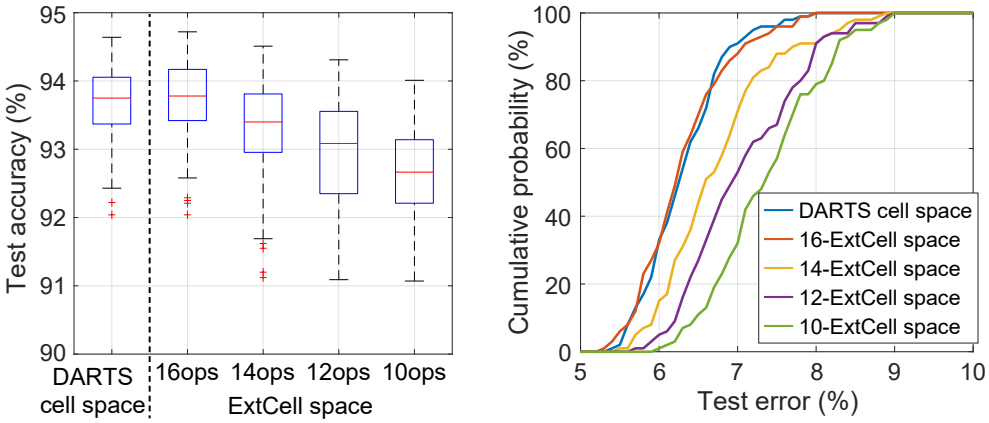
In this chapter, we propose to extend the conventional DARTS cell space to include cells, which are not fitted into the DARTS cell. The main difference from the DARTS cell is the number of operations that are activated between two nodes. While only a single operation can be activated at most between nodes i and j in a DARTS cell, it is

possible for multiple operations to be activated in an ExtCell. In other words, when candidate operations are considered as individual edges, $|\mathcal{O}|$ edges can exist between two nodes rather than only one edge. For example, as shown in Figure 4.1(c), two edges exist between nodes 2 and 5, *i.e.*, x^2 and x^5.

Changing the number of activated operations between two nodes to be variable affects three additional components: 1) the number of incoming edges for intermediate node j, 2) the number of activated intermediate nodes, and 3) the number of operations in a cell pair[1] (*i.e.*, normal and reduction cells). For each intermediate node in a DARTS cell, there must exist two incoming edges, *i.e.*, (i_1, j) and (i_2, j) for node j, where $i_1 \neq i_2$, $i_1 < j$, $i_2 < j$, and $j = \{2, ..., N-1\}$. Accordingly, all the intermediate nodes are activated and connected to the output node of a DARTS cell, and both normal and reduction DARTS cells have eight operations; the total number of operations in the DARTS cell pair is 16.

Unlike a DARTS cell, an ExtCell does not fix the number of incoming edges to two, and thus, each of normal and reduction ExtCells can have a different number of operations. Because up to $2E|\mathcal{O}|$ operations can exist in the ExtCell pair, we introduce the concept of operation budgeting. Operation budgeting sets the total number of operations in the ExtCell pair to k; k-ExtCell denotes an ExtCell with k operations. For example, as illustrated in Figure 4.1(c), given k=16 like a DARTS cell, it is possible for an ExtCell to each have a different number of operations, *i.e.*, a normal ExtCell with fewer operations than eight and a reduction ExtCell with more operations. Lastly, there can exist inactivated intermediate nodes that do not have incoming edges and thus are not connected to the output of an ExtCell. Note that the DARTS cell space is a subset of the ExtCell space by definition.

[1] In fact, as the target to be searched, most cell-based NAS studies simply refer to a cell pair as a cell.



(a) Test accuracy distribution **(b) Error empirical distribution**

Figure 4.2: Distribution of performance of architectures sampled from DARTS cell space and $\{16, 14, 12, 10\}$-ExtCell space: (a) test accuracy (%) distribution and (b) accumulated test error (%) distribution [106].

4.2.2 ExtCell Space Quality Analysis

Introduction of the ExtCell space raises the following question:

- Can an ExtCell that does not belong to the DARTS cell space achieve a high performance?

To answer this question, we evaluate the qualities of the DARTS cell and ExtCell spaces using the two metrics: 1) the test accuracies of randomly sampled cells from these search spaces and 2) the error empirical distribution function (EDF) [106] of these sampled cells. The EDF quantifies the search space quality through the percentage of cell-based networks whose test errors are equal to or less than a certain threshold. The closer the EDF curve is to the top-left corner, the higher the quality of the search space.

We follow the experimental protocol that was proposed to quantify the quality of a search space [106]. First, we randomly generate 100 cell pairs for the DARTS cell and ExtCell spaces. For the ExtCell space, we explore various values of k for k-Extcell

by setting $k \in \{10, 12, 14, 16\}$. A total of 500 unique cells are sampled from these spaces and there are no duplicates between the generated cells. Networks stacking the sampled cells are trained on CIFAR-10 for 200 epochs. The networks are configured to have a resolution of 24, eight cells (six normal and two reduction cells), and an initial channel of 16. As for the other hyperparameters and training techniques such as cutout data augmentation [26], we follow those of the typical retraining process (see Section 4.4). We use the same training settings across all sampled cells to guarantee a fair comparison of the search spaces. Even though this network configuration does not exactly match that of the retraining process (*i.e.*, a resolution of 32, 20 cells, and an initial channel of 36), the use of a proxy network for such evaluation experiments is known to provide meaningful and reliable insights as long as the training setting is kept consistent across all the sampled architectures [30, 32, 50, 73, 80, 95, 106, 126, 145, 149, 157].

The evaluation results are provided in Figure 4.2, where (a) shows box plots of the test accuracies and (b) shows the corresponding EDF plots. It appears that the quality of the 16-ExtCell space is comparable to that of the DARTS cell space. Also, some k-ExtCells with $k < 16$ yield competitive test accuracies, although the qualities of the k-ExtCell spaces monotonically decrease at average as the value of k decreases. This implies that with the help of a suitable search algorithm, performative ExtCells with $k < 16$ can be discovered.

We further investigate how performance the sampled ExtCells can achieve in the retraining setting. We choose the randomly generated k-ExtCells with $k \in \{10, 12, 14, 16\}$, which achieve the top-2 performance in the aforementioned proxy setting. It is reasonable to assume that these ExtCells are searched by the proxy-based random search, which selects two ExtCells with the top-2 performance among the 100 ExtCells randomly sampled.

Table 4.1: Test accuracies of randomly generated ExtCells on CIFAR-10.

ExtCell (# of ops)	Params. (M)	Test accuracy (%)
Random-a (10)	2.8	97.20±0.11
Random-b (10)	3.3	97.25±0.04
Random-c (12)	3.0	97.32±0.11
Random-d (12)	3.5	97.04±0.08
Random-e (14)	3.3	97.27±0.06
Random-f (14)	3.4	97.52±0.05
Random-g (16)	4.1	97.41±0.07
Random-h (16)	4.4	97.33±0.03

The test accuracies of the chosen eight ExtCells on CIFAR-10 are presented in Table 4.1. Performance drop of these ExtCells is not large, compared to the performance of the DARTS cells that were previously searched by recently developed cell-based NAS methods; please see Section 4.4 for the details. In particular, despite the proxy-based random search, Random-f achieves the performance close to the state-of-the-art. These results answer the aforementioned question; that is, ExtCells can achieve high performance. It suggests that a search algorithm suitable to explore the ExtCell space is needed to search for ExtCells that achieve higher performance and have smaller parameter size.

4.2.3 ExtCell Derivation through Top-k Survival Rule

In differentiable NAS using mixed operations in the super-network cell, the optimized continuous operation strengths θ^* must be discretized to derive a cell from the super-network cell. Conventional NAS methods with argmax selection are inherently incapable of deriving an ExtCell because the search process is equivalent to solely searching for the top-1 operation between two nodes. Hence, the selection rule is accordingly based on the argmax function to select the most preferred operation: $o^{(i,j)} = \mathrm{argmax}_{o_k \in \mathcal{O}} \, \theta_k^{*(i,j)}$. Such approach reflects a local competition among can-

didate operations. Between two nodes, the operation with the maximum operation strength is chosen, resulting in a cell with 14 edges when $N = 6$, as shown in Figure 4.1 ((a) $\rightarrow$ (b)-left); the total number of chosen operations are 28 in a cell pair. Then, a heuristic post-processing step to select eight edges among the 14 edges is required to generate a DARTS cell (from the left one to the right one in Figure 4.1(b)). Since there exists no separate metric to quantify edge strengths, the heuristic leaves edges with top-2 operation strengths among the remaining incoming edges for each intermediate node, such that both normal and reduction cells keep eight edges.

When assuming that operation o_A with the second highest operation strength in an edge is more desirable choice than operation o_B with top-1 strength in another edge, the argmax selection rule cannot select operation o_A by definition. If o_A is selected, this cell is an ExtCell. To enable to select o_A, all the candidate operations within the super-network cell should be fairly competed. Therefore, to derive k-ExtCells from the super-network cell, we propose the top-k survival rule that selects operations with top-k strengths among $2E|\mathcal{O}|$ candidate operations in normal and reduction ExtCells. The new rule selects operations with top-k strengths among $2E|\mathcal{O}|$ candidate operations in normal and reduction ExtCells. Hence, the number of operations in normal and reduction ExtCells can be different, which fits into the definition of the ExtCell. Note that the cells derived by the top-k survival rule do not need any post-processing. This approach reflects a global competition problem that is newly formulated for the differentiable NAS exploring ExtCell space.

4.3 Proposed Search Method: BtNAS

We propose a novel differentiable NAS algorithm, named BtNAS, that is compatible with the global competition problem in the ExtCell space. It was recently re-

ported that deterministic operation strengths are not reliable metrics for the final cell derivation [81, 131]. In addition, stochastic approaches that sample the operation strengths or perturb them encourage further exploration of the search space and improve the search reliability by smoothing the loss landscape with respect to operation strengths [14, 15, 158]. Therefore, we adopt a stochastic approach that samples continuous operation strengths θ, which individually follow Beta distributions with positive distribution parameters: $Beta(\alpha, \beta)$, where $\alpha > 1$ and $\beta > 1$. The Bernoulli distribution may also be a viable choice of distribution to model θ, but the discrete sampling of the operation strengths may cause multi-model forgetting, which significantly degrades the search reliability [5, 150]. Instead, we utilize the continuous Beta distribution to sample the operation strengths from [0, 1], where a sample closer to 1 indicates a higher operation strength.

We formulate the objective of BtNAS using the variational Bayes method [6, 71]. In theory, we want to find the true posterior distributions of operation strengths θ given the target dataset D and the super-network weights w: $p(\theta|w, D)$. However, because the posterior distributions are intractable to be obtained, we employ the variational Bayesian approach to optimize an approximation of the true posterior distribution. We introduce a variational Beta distribution with two trainable distribution parameters α and β over θ: $q_{\alpha,\beta}(\theta)$, where $\theta \sim Beta(\alpha, \beta)$. Note that the Beta distribution is defined separately for every operation, *i.e.*, $q_{\alpha_{(i,j,o_k)},\beta_{(i,j,o_k)}}(\theta_{(i,j,o_k)})$ for operation $o_k^{(i,j)}$, where i and j are nodes and $o_k \in \mathcal{O}$, and we omit the subscript (i, j, o_k) for simplicity.

The KL divergence from the variational distribution $q_{\alpha,\beta}(\theta)$ to the true posterior $p(\theta|w, D)$ is defined as:

$$D_{KL}[q_{\alpha,\beta}(\theta)||p(\theta|w, D)] = \mathbb{E}_{\theta\sim q_{\alpha,\beta}}[\log q_{\alpha,\beta}(\theta) - \log p(\theta|w, D)]. \tag{4.1}$$

Using Bayes' rule, the right-hand side of Eq. 4.1 can be re-written as:

$$\mathbb{E}_{\theta\sim q_{\alpha,\beta}}[\log q_{\alpha,\beta}(\theta) - \log p(D|\theta, w) - \log p(\theta|w) + \log p(D|w)]. \tag{4.2}$$

Re-arranging Eqs. 4.1 and 4.2 yields:

$$\begin{aligned} &\log p(D|w) - D_{KL}[q_{\alpha,\beta}(\theta)||p(\theta|w, D)] \\ &= \mathbb{E}_{\theta\sim q_{\alpha,\beta}}[\log p(D|\theta, w) - (\log q_{\alpha,\beta}(\theta) - \log p(\theta|w))] \\ &= \mathbb{E}_{\theta\sim q_{\alpha,\beta}}[\log p(D|\theta, w)] - D_{KL}[q_{\alpha,\beta}(\theta)||p(\theta|w)]. \end{aligned} \tag{4.3}$$

The left-hand side of Eq. 4.3 is referred to as the evidence lower bound (ELBO). Because $\log p(D|w)$ is a constant with respect to θ, the KL divergence from the variational distribution $q_{\alpha,\beta}(\theta)$ to the true posterior $p(\theta|w, D)$, which is the second term in the left-hand side of Eq. 4.3, can be minimized by maximizing the ELBO. According to Eq. 4.3, the ELBO is equivalent to the expected log likelihood and the negative KL divergence from the variational distribution $q_{\alpha,\beta}(\theta)$ to the prior distribution $p(\theta|w)$. Therefore, instead of maximizing the ELBO, we flip the sign of the right-hand side of Eq. 4.3 and minimize the resulting expression. The final search objective to find the optimal approximation is formulated as:

$$\max_{\alpha,\beta} \text{ ELBO} = \min_{\alpha,\beta} \mathbb{E}_{\theta\sim q_{\alpha,\beta}}[-\log p(D|\theta, w)] + D_{KL}[q_{\alpha,\beta}(\theta)||p(\theta|w)]. \tag{4.4}$$

The first term on the right-hand side of Eq. 4.4 is the negative log likelihood loss, which is a commonly used loss function, *e.g.*, cross-entropy loss in classification. Despite the sampling, the distribution parameters α and β can be trained via gradient-based optimization by employing the pathwise derivative estimator [60]; the detailed explanation is provided in Section 4.3.1.

Now, as in differentiable NAS, the bi-level optimization problem of BtNAS that incorporates the search objective obtained through variational inference is formulated as:

$$\begin{aligned} &\min_{\alpha,\beta} \mathbb{E}_{\theta\sim q_{\alpha,\beta}}[L(\theta, w^*(\theta); D_{\text{val}})] + \lambda D_{KL}[q_{\alpha,\beta}(\theta)||p(\theta|w^*(\theta))] \\ &\text{s.t. } w^*(\theta) = \underset{w}{\text{argmin}}\, L(\theta, w; D_{\text{tr}}), \end{aligned} \tag{4.5}$$

where λ is a coefficient that controls the strength of the KL divergence from $q_{\alpha,\beta}(\theta)$ to a prior $p(\theta|w^*(\theta))$. This KL divergence term can be interpreted as regularizing the suboptimal premature states during the search process. In this chapter, to encourage the exploration of various cell architectures, we set the prior as $Beta(\alpha = 1, \beta = 1)$, *i.e.*, a uniform distribution $U(0, 1)$.

To derive a final cell in BtNAS, the expectation of the Beta distribution is used as the operation strength: $\mathbb{E}_{\theta\sim q_{\alpha,\beta}}[\theta] = \frac{\alpha}{(\alpha+\beta)}$ per operation $o_k^{(i,j)}$. When the expected value is closer to 1, the corresponding operation is more likely to be selected by the top-k survival rule.

4.3.1 Pathwise Derivative Estimator for Beta Distribution

To optimize the distribution parameters of Beta distributions via gradient-based optimization, we need to be able to compute their gradients. However, the gradients with respect to the distribution parameters cannot be directly calculated due to the sampling process, and the reparameterization trick that is commonly employed in the Gumbel-technique also cannot be applied to the Beta distribution [60]. Therefore, we employ the pathwise derivative estimator for the Beta distribution [60], and obtain

the approximated gradients as:

$$\frac{d\theta}{d\alpha} = -\frac{\frac{\partial F_{\mathrm{Beta}}}{\partial \alpha}(\theta|\alpha,\beta)}{f_{\mathrm{Beta}}(\theta|\alpha,\beta)}, \quad \frac{d\theta}{d\beta} = \frac{\frac{\partial F_{\mathrm{Beta}}}{\partial \beta}(1-\theta|\beta,\alpha)}{f_{\mathrm{Beta}}(1-\theta|\beta,\alpha)}. \tag{4.6}$$

In Eq. 4.6, F_{Beta} and f_{Beta} denote the cumulative density function and the probability density function of a Beta distribution, respectively. These functions are defined as:

$$F_{\mathrm{Beta}}(\theta|\alpha,\beta) = \frac{B(\theta;\alpha,\beta)}{B(\alpha,\beta)}, \quad f_{\mathrm{Beta}}(\theta|\alpha,\beta) = \frac{1}{B(\alpha,\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1}, \tag{4.7}$$

where $B(\theta;\alpha,\beta)$ and $B(\alpha,\beta)$ are the incomplete beta function and the beta function, respectively. The gradients of $F_{\mathrm{Beta}}(\theta|\alpha,\beta)$ can be obtained by a numerical approximation provided by Jankowiak and Obermeyer [60]. For the pathwise derivative estimator, we use the differentiation tool provided in PyTorch [114].

4.3.2 Discussion with Previous Studies

Among the previous studies, two studies can be related with our approach that searches for cells different from DARTS cells: FairDARTS [18] and Bonsai-Net [40]. FairDARTS [18] is based on the deterministic approach on the differentiable NAS method. FairDARTS employed the operation-wise sigmoid function to obtain operation strengths of the super-network cell. This operation-wise calculation for operation strengths may be relevant to our approach, but the main objective of FairDARTS was to eliminate the dominance of skip connection, which suppresses other candidate operations within an edge when using the softmax function. In addition, after the search process, FairDARTS filters out candidate operations, whose operation strengths are below a certain threshold such as 0.85, and then selects the top-1 operation among the remaining candidate operations for each edge. Hence, the cells searched by FairDARTS are unable to have multiple operations between two nodes; *i.e.*, these cells are sub-cells

of DARTS cells.

Bonsai-Net [40], which are based on a memory-aware differentiable pruner, tried to search for the cell types similar to ExtCells as well as the connectivity between all cells in the entire network. Even though it appears that both Bonsai-Net and BtNAS aim to extend the definition of a DARTS cell, there exist three notable limitations of Bonsai-Net compared to BtNAS as follows. 1) Bonsai-Net requires a much higher search cost (3.3 GPU days on a single 1080ti GPU) than BtNAS (0.2 GPU days on a single 2080ti GPU). 2) Due to GPU memory limitation, Bonsai-Net can only be executed on networks with 8 cells, and no analysis on the quality of this search space was provided. 3) Such an approach to search for the connectivity between cells in the entire network lacks scalability. While the network searched by Bonsai-Net is restricted to only 8 cells, cell-based NAS methods including BtNAS can construct a network consisting of a variable number of cells.

4.4 Experiments and Results

For the evaluation, as in Section 3.4.1, we used two types of NVIDIA GPUs: Tesla V100 for searching and training neural architectures on ImageNet, and GeForce RTX 2080ti for the remaining datasets. Experimental settings for search and retraining processes are also maintained.

4.4.1 Evaluation on NAS Algorithms with ExtCell Derivation

To evaluate DARTS [89], FairDARTS [18], and PR-DARTS [158] on the ExtCell space, we incorporate the ExtCell derivation based on top-k survival rule with these NAS algorithms. DARTS is the most widely benchmarked differentiable NAS method that deterministically obtains the operation strengths through the softmax function of

Table 4.2: Test accuracies (%) of ExtCells derived by the top-k survival rule from the super-network cells optimized by existing NAS methods and BtNAS on CIFAR-10.

NAS algorithm	k=10	k=12	k=14	k=16
DARTS	96.06	96.04	96.36	95.65
DARTS-ELU	94.89	93.35	94.66	93.75
PR-DARTS	91.21	92.88	92.88	93.40
BtNAS (proposed)	**97.45**	**97.38**	**97.55**	**97.43**

the architecture parameters for each edge; however, the softmax function applied to each edge is more appropriate for the local competition problem than the global competition problem. Hence, to remove the local competition effect of the softmax function, we further evaluate a variant of DARTS named DARTS-ELU, which obtains positive operation strengths through the exponential linear unit (ELU) with $+1$ added on each operation strength, where $\mathsf{ELU}(x) = x$ if $x > 0$, $e^x - 1$ otherwise. FairDARTS and PR-DARTS both replace the softmax function in DARTS with another modeling approach to individually consider the operation strengths. In FairDARTS, the operation strengths are deterministically computed using the sigmoid function, and in PR-DARTS, they are discretely sampled from $\{0, 1\}$ through the Gumbel-sigmoid function with temperature scaling. We investigate whether FairDARTS and PR-DARTS can be compatible with the global competition problem by eliminating the local competition effect.

Following the default protocols provided by these methods, we optimize the operation strengths θ within the super-network, and visualize the search results through heatmaps of θ^* in Figure 4.3. As shown in Table 4.2, ExtCells derived from θ^* by the top-k survival rule with $k \in \{10, 12, 14, 16\}$ are evaluated on CIFAR-10 by obtaining the test accuracies after the typical retraining process.

As shown in Figure 4.3, ExtCells derived by DARTS and DARTS-ELU tend to include more skip connection and pooling operations than those from the other two

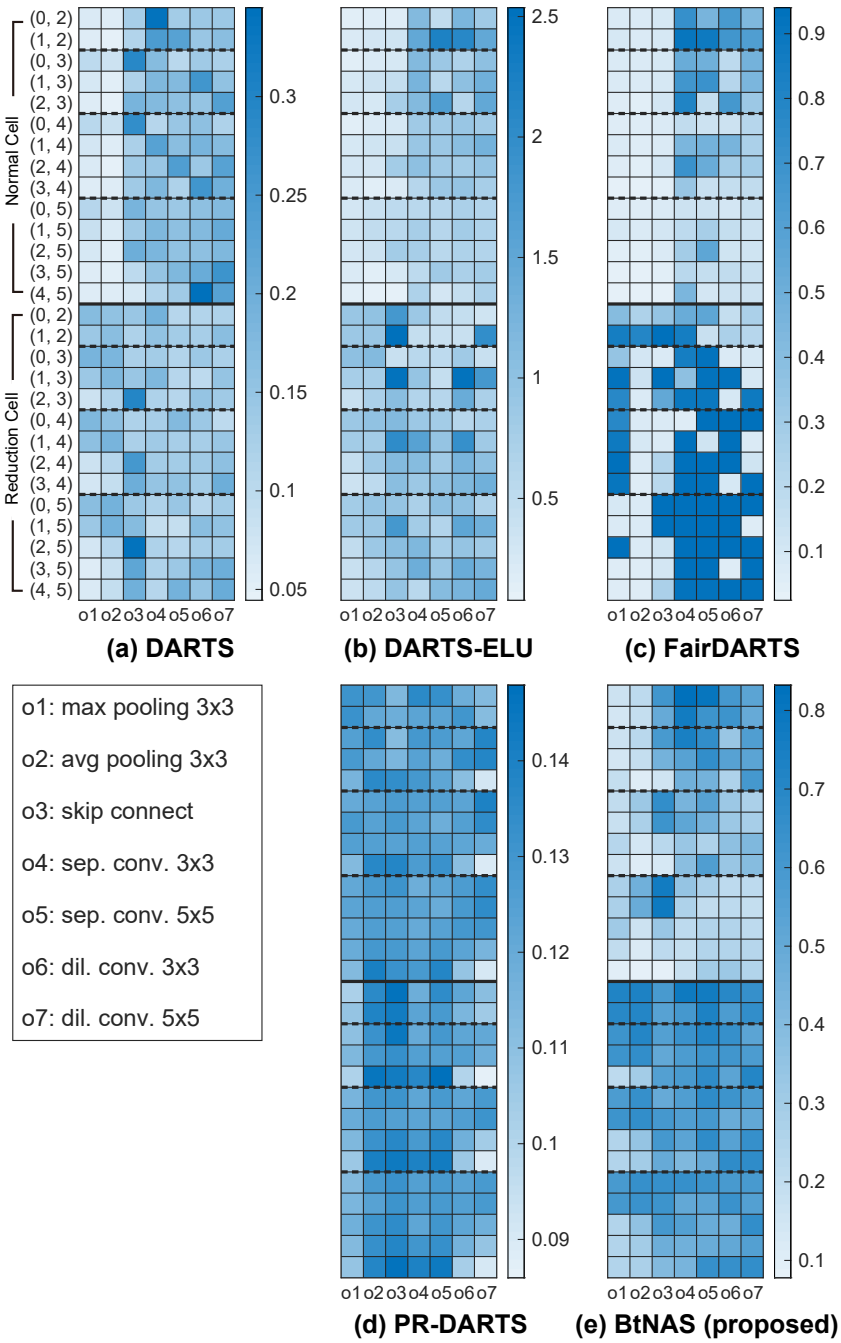


Figure 4.3: Operation strengths in super-network cells optimized by various NAS algorithms.

methods. The dominance of parameter-free operations, particularly skip connection, has been reported to lead to the relatively low search performance of DARTS [18, 131, 148, 158]. Contrary to our expectation, FairDARTS and PR-DARTS also experience a significant decrease in search performance. The low search performance of PR-DARTS could be the result of the use of the softmax function to obtain continuous θ^* when deriving the final cell. Furthermore, as shown in Figure 4.3(d), the differences among all the operation strengths are much smaller than those of the other methods; such differences seem too small to rank the operations from strong to weak operations. In FairDARTS, operations within the top-40 strengths are in the reduction cell, and consequently, a normal cell cannot be constructed when $k \leq 16$. This may be an unintentional side effect of the zero-one loss of FairDARTS, which drives operation strengths closer to zero or one, and this effect becomes more conspicuous in the reduction cell. In summary, despite the individual consideration of the operations, DARTS-ELU, FairDARTS, and PR-DARTS fail to search for desirable ExtCells. Unlike these existing NAS algorithms, for all k, BtNAS successfully searched for competitive ExtCells whose performances are close to state-of-the-art in cell-based search space, demonstrating that BtNAS is a tailored NAS algorithm for ExtCell space.

4.4.2 Effects of Regularization Coefficient λ

We investigated the effects of coefficient λ for the KL divergence term in Eq. 4.5 using different $\lambda \in \{0.01, 0.001, 0.0001\}$. As described in Section 4.3, a uniform distribution $U(0, 1)$ is used for a prior $p(\theta|w^*(\theta))$ in Eq. 4.5. Hence, as λ decreases, the exploration effect also decreases during the search process, *i.e.*, the optimization of the super-network cell.

As shown in Table 4.3, ExtCells searched by BtNAS with $\lambda = 0.0001$ experi-

Table 4.3: Test accuracies (%) of ExtCells derived by BtNAS with different λ on CIFAR-10.

λ	k=10	k=12	k=14	k=16
0.01	2.82±0.07	2.65±0.16	2.62±0.03	2.66±0.06
0.001	2.58±0.03	2.64±0.02	2.52±0.05	2.62±0.04
0.0001	3.61±0.05	3.70±0.13	3.75±0.12	3.77±0.21

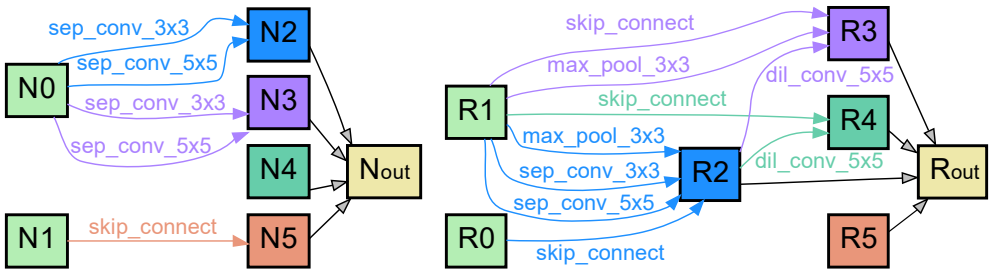


Figure 4.4: ExtCells searched by BtNAS with top-k survival rule and $k = 14$.

ence a significant performance drop, even though such performances are higher than ExtCells searched by other NAS algorithms as presented in Table 4.2. In the ExtCells of BtNAS with $\lambda = 0.0001$, most of operations selected by top-k survival rule exist in the reduction cells; for example, for $k = 16$, the normal and reduction cells have 2 and 14 operations, respectively. When $\lambda = \{0.01, 0.001\}$, BtNAS yields similar performance, and obtains the best ExtCells when $\lambda = 0.001$. Hence, $\lambda = 0.001$ was used for BtNAS by default.

4.4.3 Comparison with DARTS Cells of Existing NAS Methods

We searched for ExtCells on CIFAR-10 [75] using BtNAS and trained them three times with different seeds, and calculated the average of the test errors. Table 4.4 reveals that, with the reasonable search cost, BtNAS can find ExtCells, whose performance are comparable to that of DARTS cells searched by the state-of-the-art NAS methods. The neural network constructed by stacking the searched ExtCells

Table 4.4: Comparison of cell architectures searched by various NAS methods on CIFAR-10.

Cell architecture searched by NAS	**Test error (%)**	**Params. (M)**	**FLOPs (M)**	**Peak mem. (MB)**	**Cost (GPU days)**
DARTS (1st) [89]	3.00±0.14	3.2	511	270	0.2‡
SNAS (moderate) [140]	2.85±0.02	2.8	448	255	1.5
GDAS [31]	2.93	3.4	529	270	0.2
PDARTS [16]	2.5	3.4	543	276	0.3
PC-DARTS [142]	2.67±0.07	3.6	568	277	0.1‡
EcoDARTS-c4r2 [157]	2.80	4.5	-	-	0.2‡
SDARTS-RS [14]	2.67±0.03	3.6	573	281	0.2‡
SGAS-Cri.1 [81]	2.66±0.24	3.7	595	285	0.2‡
FairDARTS [18]	2.82†±0.14	2.9±0.4	-	-	0.2‡
(FairDARTS-d)	2.70†±0.07	3.4	543	272	
PR-DARTS [158]	2.50†±0.10	3.4	540	276	0.2‡
DARTS+PT [131]	2.61±0.08	3.0	-	-	0.8
DrNAS [15]	2.59†±0.10	4.1	648	294	0.6
EnTranNAS-DST [144]	2.56†±0.06	3.2	520	271	0.1
BtNAS (k=14), Avg	2.52±0.05	2.8	454	214	0.2‡
BtNAS (k=14), Best	2.45				

†denotes our retraining results, while the other values in the first group of rows are reported in original papers.
‡denotes the search cost with a GeForce RTX 2080 Ti GPU. The other values are reported in original papers.

has fewer parameters than most DARTS cells including the state-of-the-art DARTS cells. Compared to SNAS [140] and FairDARTS [18], the performance of BtNAS is much higher, while the network sizes are similar. Figure 4.4 shows that the normal and reduction ExtCells searched by BtNAS with $k = 14$ include five and nine operations, respectively. In addition, because a network that is used for the evaluation consists of 18 normal cells and two reduction cells, the networks stacking our ExtCells have fewer parameters compared to DARTS cell-based networks. Lastly, the networks based on the searched ExtCells can be efficiently executed; the number of floating-point operations (FLOPs) and the peak memory usage of the searched ExtCells are the lowest.

Table 4.5: Comparison with various neural networks on ImageNet under the mobile setting with <600M FLOPs.

Cell architecture searched by NAS		**Test error (%)**		**Params. (M)**	**FLOPs (M)**	**Search Cost (GPU days)**
		top-1	**top-5**			
DARTS (2nd order) [89]		26.7	8.7	4.7	538	1.1
SNAS (mild) [140]		27.3	9.2	4.3	522	1.5
GDAS [31]		26.0	8.5	5.3	545	0.2
PC-DARTS [142]		24.2	7.3	5.3	591	3.8†
SDARTS-ADV [14]		25.2	7.8	4.8	540	1.3
FairDARTS-A [18]		26.3	8.3	3.6	401	0.4
PR-DARTS [158]		24.1	7.3	5.0	552	0.2
BtNAS	k=14	26.9	9.0	4.0	467	0.2
	k=16	26.8	8.8	4.3	499	0.2
BtNAS+IMN	k=16	26.6	8.6	3.4	377	0.7†
(with ImageNet proxy data)	k=20	25.8	8.5	4.5	482	0.7†
BtNAS+IMN+ActAll (with ImageNet proxy data)	k=14	26.5	9.0	4.5	485	0.7†

†denotes searching on ImageNet, otherwise searching on CIFAR-10 or CIFAR-100 and then transferring to ImageNet.

Table 4.5 presents the performances of the architectures searched by various NAS methods in the ImageNet mobile setting, which indicates a network with <600M FLOPs and a resolution of 224. ExtCells searched by BtNAS on CIFAR-10 achieve a comparable performance to DARTS and SNAS (under the similar network size). We further search for ExtCells using proxy data of ImageNet, which is introduced in Chapter 3, and refer to these ExtCells as BtNAS+IMN. BtNAS+IMN obtains higher performance than BtNAS, and when k=16, has fewer parameters than BtNAS. Compared with FairDARTS-A, the ExtCells of BtNAS+IMN with k=16 also achieve a comparable performance while the size and FLOPs are similar.

Table 4.6: Evaluation of BtNAS with all activated intermediate nodes on CIFAR-10.

Search algorithm	k=10	k=12	k=14	k=16
BtNAS	2.58±0.03	2.64±0.02	2.52±0.05	2.62±0.04
BtNAS (Best)	2.55	2.62	2.45	2.57
BtNAS+ActAll	2.54±0.14	2.56±0.05	2.46±0.13	2.55±0.06
BtNAS+ActAll (Best)	**2.39**	**2.51**	**2.31**	**2.50**

4.5 Discussion

4.5.1 Guarantee for Activating All Intermediate Nodes

To fortify search performance of BtNAS, we refine the cell derivation to make all intermediate nodes activated; this approach is abbreviated by ActAll. To guarantee that every intermediate node has at least one incoming edge that is activated, we normalize the operation strengths used for cell derivation per intermediate node. In node x^j, operation strengths of its incoming operations, denoted by $\theta_{(i,j)}$ where $i < j$, are normalized by the maximum operation strengths, *i.e.*, $\texttt{max}(\theta_{(i,j)})$ for all i. Hence, at least one operation strength in each node is 1; that is, there are $N - 2$ number of operations whose strengths are 1 within the super-network cell, where $N - 2$ is the number of intermediate nodes. The results are presented in Table 4.6. BtNAS+ActAll improves the search performance of BtNAS in average, indicating that activating all nodes can help to derive a promising ExtCell. It is remarkable that BtNAS+ActAll with $k = 14$ achieves a test error of 2.31%, which is the state-of-the-art performance.

4.5.2 Search Robustness

To analyze the robustness of BtNAS, we evaluate it across three datasets (CIFAR-10, CIFAR-100 [75], and SVHN [100]) and two out of the four restricted search spaces (S2 and S4) [148], which are described in Section 3.4.6. Figure 4.5 demonstrates that the patterns of the optimized operation strengths are consistent across datasets in

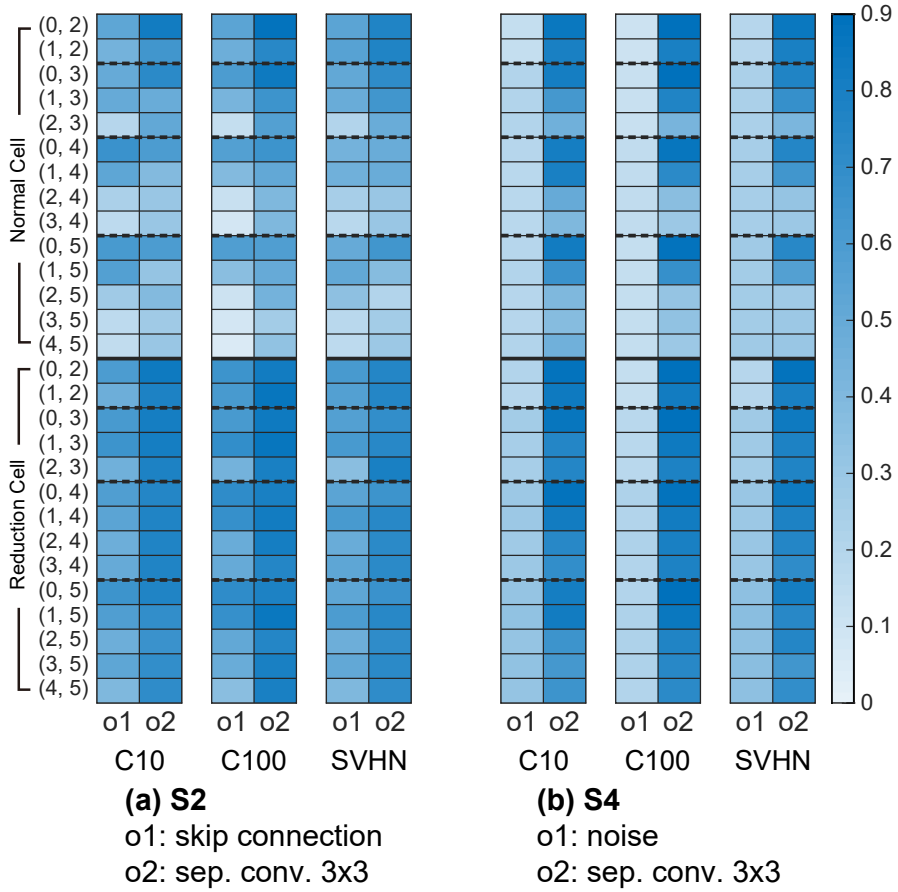


Figure 4.5: Operation strengths in super-network cells, which encode two restricted search spaces [148], optimized by BtNAS.

Table 4.7: Evaluation (test error (%)) across datasets and restricted search spaces, S2 and S4 [148].

NAS algorithm	CIFAR-10		CIFAR-100		SVHN	
	S2	S4	S2	S4	S2	S4
DARTS [89]	4.85	7.20	26.05	22.85	3.53	3.05
RobustDARTS (L2) [148]	3.31	3.56	22.44	21.94	2.51	2.50
BtNAS	3.16	2.66	23.49	22.12	2.45	2.39

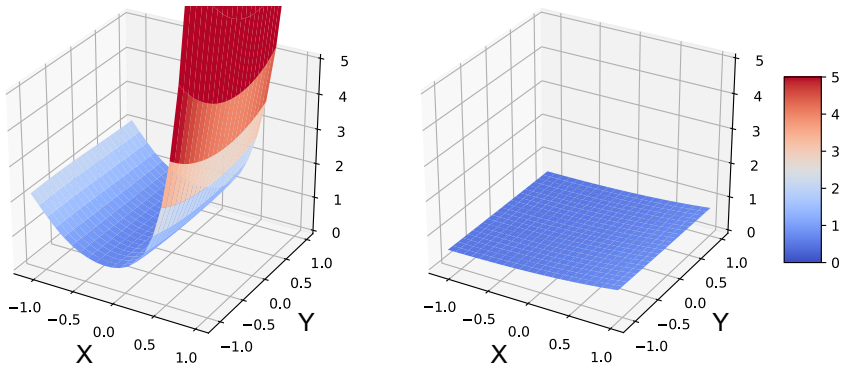


Figure 4.6: Loss landscapes with respect to operation strengths θ after the search process on CIFAR-10. The x-axis is the gradient direction $\nabla_\theta L$, and the y-axis is the random orthogonal direction.

each space. In S2, sep. conv. 3x3 is more likely to be selected than skip connection, indicating that BtNAS avoids the problem of skip connection domination due to its stochasticity [14, 15] and elimination of local competition [18]. Notably, in S4, noise in every edge has smaller operation strength than sep. conv. 3x3, and consequently, BtNAS generates noise-free ExtCells. These results demonstrate the robustness of BtNAS. Also, Table 4.7 shows that after retraining, ExtCells searched by BtNAS achieve competitive performances to DARTS cells.

4.5.3 Loss Landscape Smoothness

Based on the research on cell-based NAS [14, 15, 148], we analyze why BtNAS can discover competitive ExtCells. It has been reported that larger architectural eigenvalues leading to sharp local minima declined the generalization performance of searched architectures and smoothing the loss landscape with respect to architecture parameters (*i.e.*, operation strengths in our context) made the search results reliable [14, 148]. Given the trained super-network cell with a steep loss landscape, a small perturbation in the operation strengths causes a significant performance drop in

the cell derivation. They demonstrate that smoothing the loss landscape by injecting stochasticity to the search process can relieve the dominance of skip connection.

Analogously, we speculate that the stochastic nature of BtNAS, which samples continuous operation strengths from Beta distributions, contributes to smoothing its loss landscape. To validate this speculation, we plot the loss landscapes of BtNAS and FairDARTS with respect to the operation strengths, following previous studies [14, 82]. In Figure 4.6, the x-axis is the gradient direction $\nabla_\theta L$, and the y-axis is the random orthogonal direction. Figure 4.6 empirically supports that the stochastic sampling of continuous operation strengths does indeed lead to a smoother landscape. In contrast, FairDARTS, which deterministically calculates operation strengths using the sigmoid function, yields a much steeper landscape than BtNAS. It leads to the significant failure of FairDARTS to search for ExtCells, as described in Section 4.4.1. Even though FairDARTS discovered performative DARTS cells, additional post-processing described in Section 4.3.2 was necessarily employed to address the low reliability of the trained operation strengths when deriving DARTS cells.

Meanwhile, Chen *et al.* [15], who considered the NAS based on the local competition using Dirichlet distribution, presented a theoretical result that the upper-level objective of NAS (*i.e.*, optimization of trainable parameters of Dirichlet distribution within the super-network cell) has the implicit regularization effect. Because the Beta distribution that we adopt is considered as the 2-variate of the Dirichlet distribution, *i.e.*, $\hat{\theta} \sim Dir(\gamma)$, where $\hat{\theta} = (\theta, 1-\theta)$ and $\gamma = (\alpha, \beta)$, BtNAS can seamlessly enjoy this theoretical result. Therefore, during the search process of BtNAS, the increase in the maximum eigenvalue of Hessian matrix with respect to operation strengths can be suppressed, resulting in the smooth loss landscape.

4.6 Summary

In this chapter, to reduce the number of discarded cells that are evaluated during the search process of differentiable NAS working on the cell-based search space but can achieve a high performance, we introduced an extension of the DARTS cell space and demonstrated the quality of the ExtCell space. To derive cells that are not restricted to the DARTS space, we treated NAS as a global competition problem, and accordingly proposed BtNAS, which models operation strengths individually as continuous random variables following Beta distributions. In BtNAS, the true posterior distributions of operation strengths are approximated using the variational Bayes method, the parameters of variational Beta distributions are optimized through the pathwise derivative estimator. Compared with state-of-the-art DARTS cells, the ExtCells searched by BtNAS achieved competitive performance and yielded networks with fewer parameters. We further analyzed the effectiveness and search robustness of BtNAS through substantial experimental results. This research corroborates that expanding the search space can enhance the performance of cell-based NAS. We hope that our work encourages subsequent research on search algorithms for the global competition problem on the cell-based search spaces.

Chapter 5

Neural Architecture Search for Efficient Spiking Neural Networks

5.1 Introduction

SNNs are the next generation of neural networks inspired by the brain's information processing systems [93]. The neurons in SNNs transmit information through sparse and binary spikes, which enable event-driven computing. Hence, SNNs have the potential to improve the energy-efficiency of artificial intelligence systems, significantly. The size and number of spikes of SNNs have a great impact on the performance of neuromorphic architectures [23, 97], which support neuromorphic computing. Most neuromorphic chips adopt network-on-chip architectures with neuromorphic cores, and SNNs are mapped to these multiple cores. The large number of spikes causes spike congestion between the cores, thereby significantly increasing the communication overhead and degrading the performance [24]. Therefore, when designing SNNs for real-world applications, their efficiency must be considered along with accuracy.

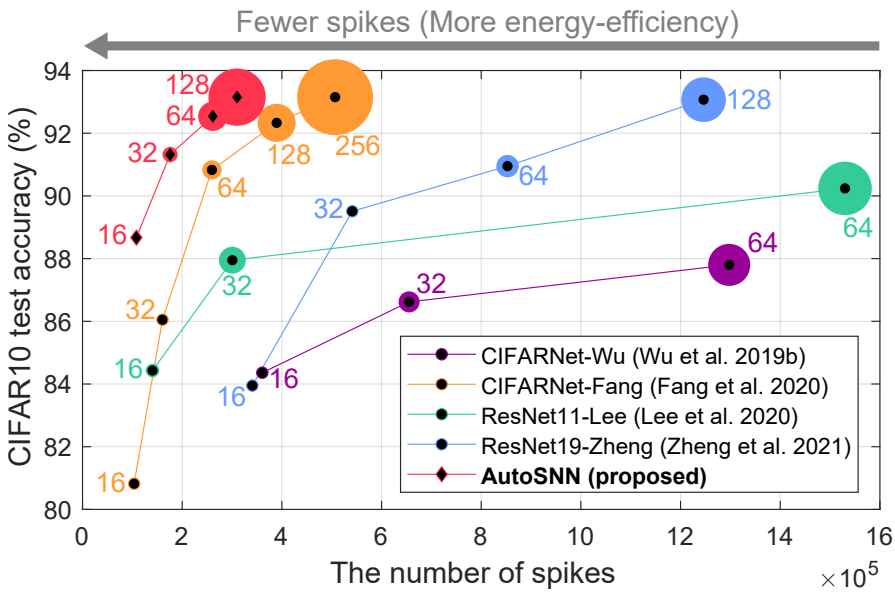


Figure 5.1: The number of spikes and CIFAR-10 accuracy of various SNN architectures. Black circle- and diamond-shaped markers denote hand-crafted and automatically-designed SNN architectures, respectively. The size of a colored circle is proportionate to the model size and the number next to the circle indicates the initial channel of each SNN architecture.

Previous studies [37, 55, 63, 79, 99, 138, 156] had focused on improving the accuracy and inference latency (timesteps) of deep SNNs by adopting gradient-based training algorithms, as in DNNs. Without architectural consideration, they used the conventional architectures used in DNNs, such as VGGNet [120] and ResNet [53]. However, even when the same training method and protocol (including the number of timesteps) are used, the number of spikes generated by an SNN differs significantly depending on its architectural design, as shown in Figure 5.1.

NAS aims to automatically find architectures with high performance from a search space. Depending on the purpose of the NAS algorithm, various evaluation metrics are defined, either using the accuracy alone or considering both the latency and FLOPs of the architectures [8, 9, 127, 136]. Across various applications [17, 29, 47, 61, 66, 139, 143, 154], NAS successfully searched for a DNN architecture that is

best-suited for the target objective.

Inspired by the success of NAS for DNNs, we propose a spike-aware NAS framework, named *AutoSNN*, to design energy-efficient SNNs. To construct an expressive search space, we introduce a two-level search space that consists of a macro-level backbone architecture and micro-level candidate spiking blocks. Through the experimental analysis of diverse architectural variations, we identify design choices that are appropriate for efficient SNNs. Our analysis shows that the preferred design choices for SNNs and DNNs differ noticeably. For instance, the inverted bottleneck block, introduced in MobileNetV2 [112] to improve the computation efficiency of DNNs, seriously degrades the efficiency of SNNs by generating a large number of spikes.

To efficiently explore the proposed search space, we also propose a search algorithm based on the one-shot weight-sharing approach of NAS [3, 9, 48, 84, 143, 155]. The AutoSNN search algorithm consists of two procedures. First, AutoSNN trains a super-network whose sub-networks share a single set of weights, by uniformly sampling an architecture from the super-network at each iteration to evenly optimize the architectures in the search space. Once the super-network is trained, an evolutionary search algorithm is executed to find the SNN with the highest evaluation metric, which we call fitness. To enable a spike-aware search, we define the fitness to reflect both the accuracy and number of spikes.

AutoSNN obtains efficient SNNs that outperform hand-crafted SNNs in terms of both the accuracy and number of spikes, as shown in Figure 5.1. The superiority of our SNNs is consistently observed across various datasets including neuromorphic datasets. Additionally, we provide substantial ablation results to demonstrate the effectiveness of AutoSNN. In particular, when our search algorithm is executed on a DNN search space, the search performance slightly deteriorates, emphasizing the importance of executing NAS directly on an SNN search space. The contributions of

this chapter are summarized as follows:

- To the best our knowledge, this is the first study to investigate the effect of architectural components on the accuracy and energy-efficiency of deep SNNs.
- We propose a spike-aware NAS framework, named AutoSNN, and find efficient SNNs that outperform conventional hand-crafted architectures.
- We demonstrate the effectiveness of AutoSNN through extensive experimental results and evaluation on various datasets.

5.2 Proposed Search Space

5.2.1 Search Space

When applying NAS, it is important to build an expressive search space that reflects the diversity of design choices. The MobileNet-based [8, 127, 136] and the cell-based search spaces [31, 89, 160] are the two most popular search spaces that form the basis of various task-specific search spaces. Inspired by the MobileNet-based search space, we propose a two-level search space design for SNNs: a macro-level backbone architecture and a micro-level to-be-determined (TBD) block design. The macro-level backbone architecture, depicted in Figure 5.2(a), defines the placement of TBD blocks, as well as other design principles, such as the choice of a down-sampling layer and the number of initial channels. In the architecture, normal blocks preserve the spatial resolution of the input feature map, and down-sampling (DS) blocks halve the spatial resolution. The first 3x3 convolution layer with spiking neurons works as a stem layer. We use a 2x2 max pooling operation with a stride of 2 as a down-sampling layer, and consequently, the channels of the TBD blocks are doubled after each down-sampling layer. We employ a voting layer that receives spikes from the

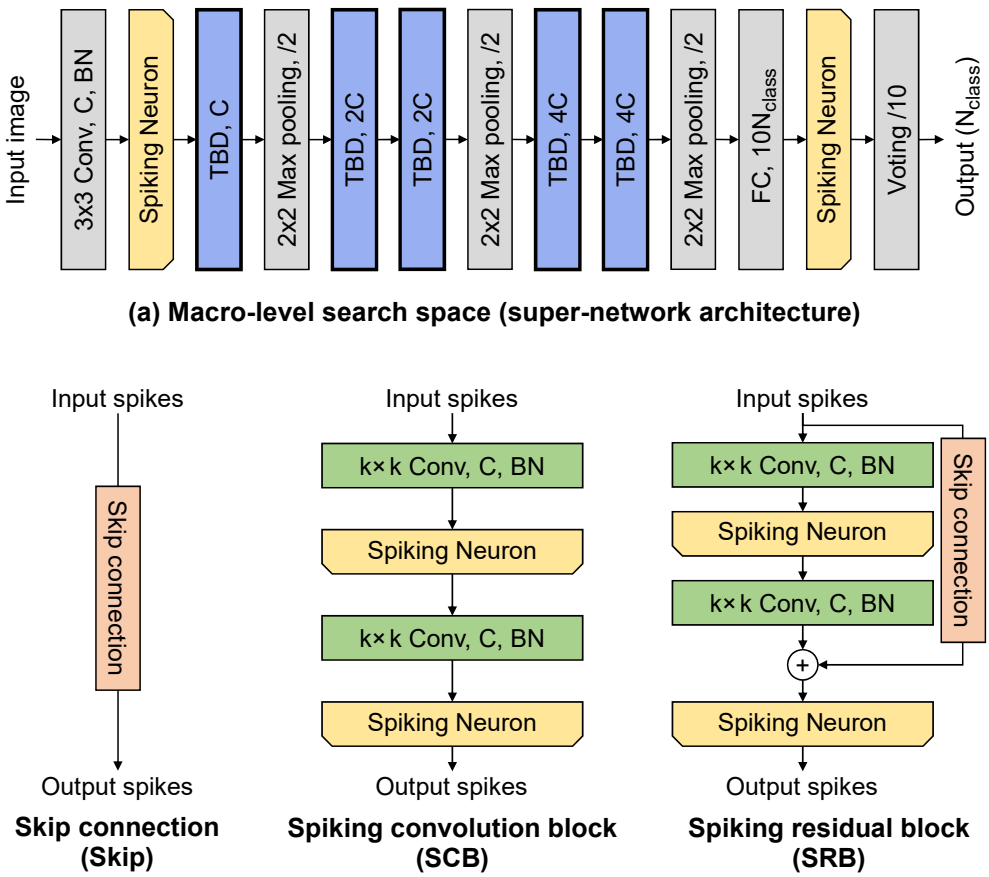


Figure 5.2: Proposed SNN search space at macro- and micro-level. (a) Structure of the proposed macro-level search space. Candidate blocks are assigned to TBD blocks. (b) Spiking blocks for the proposed micro-level search space. Convolution layer, channels, and batch normalization are denoted by Conv, C, and BN, respectively.

Table 5.1: Evaluation of different design choices for the SNN search space on CIFAR-10. All TBD blocks in each macro architecture are filled with `SCB_k3`.

Macro-level search space	GAP	Normal	Down-sample	Acc.(%)	Spikes
Proposed	✘	TBD	MaxPool	86.93	154K
Variant 1	✔	TBD	MaxPool	85.05	168K
Variant 2	✘	TBD	TBD	87.94	222K
Variant 3	✘	TBD	AvgPool	79.59	293K

spiking neurons after a fully connected (FC) layer to produce robust classification results [37]. The voting layer is implemented by a 1D average pooling layer with a kernel size of K and a stride of K; in this work, we set $K = 10$, as in the previous work [37].

The micro-level search space defines the set of candidate blocks, from which the TBD block in Figure 5.2(a) can be chosen. Based on block architectures that are often utilized for DNN search spaces, we propose three candidate blocks: skip connection, spiking convolution block (SCB), and spiking residual block (SRB). The proposed candidate blocks are depicted in Figure 5.2(b). With skip connection, denoted by `skip`, the search algorithm can choose to omit computation in certain TBD blocks. Both SCB and SRB consist of two convolution layers, but the skip connection exists only in the SRB. Although SRB had been used in recent studies [79, 156], its kernel size was limited to 3. In this chapter, we use SCBs and SRBs with kernel sizes $k \in \{3, 5\}$; these blocks are denoted by `SCB_k3`, `SCB_k5`, `SRB_k3` and `SRB_k5`, respectively.

5.2.2 Discussion on Design Choices for the SNN Search Space

In this section, we discuss how the proposed search space design is more suitable for SNNs than some of the design choices that are conventionally used for DNNs. At the macro-level, the proposed search space differs from the conventional DNN search

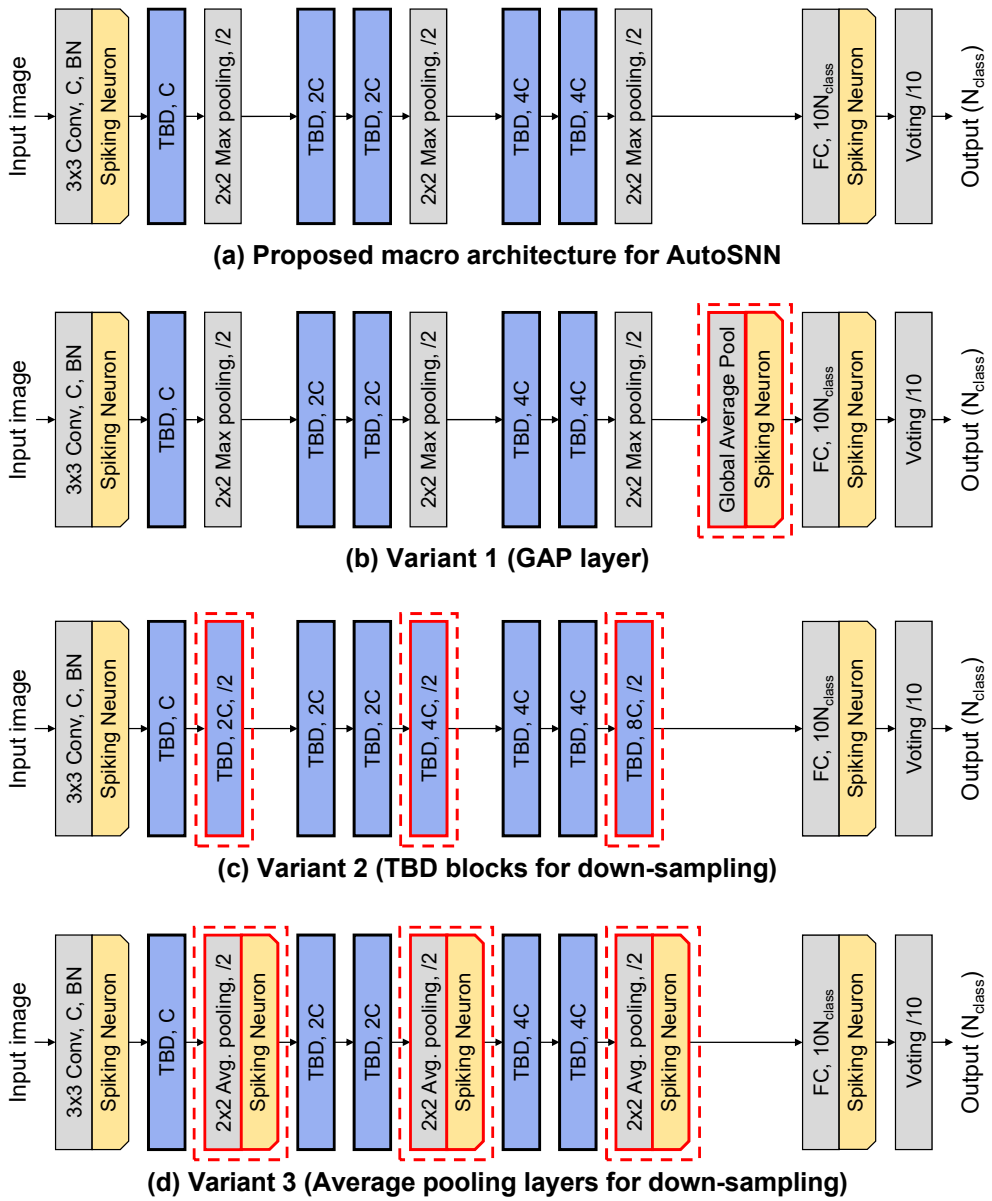
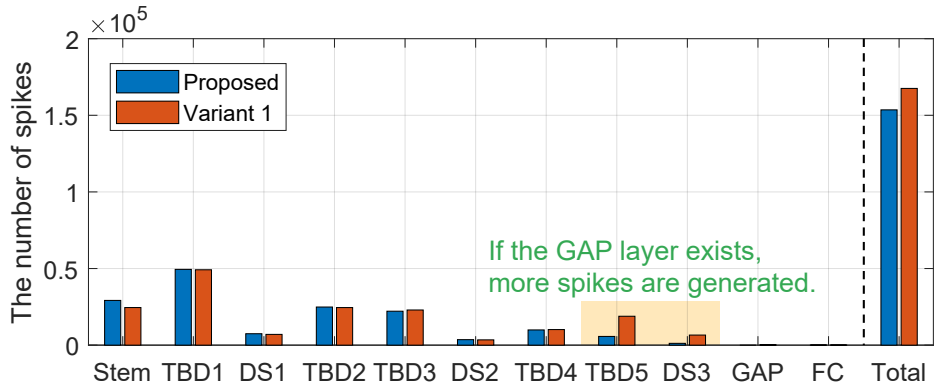


Figure 5.3: The macro architectures of the proposed macro-level search space (a) and three variants (b-d). The red dotted boxes indicate the change from the proposed macro architecture.

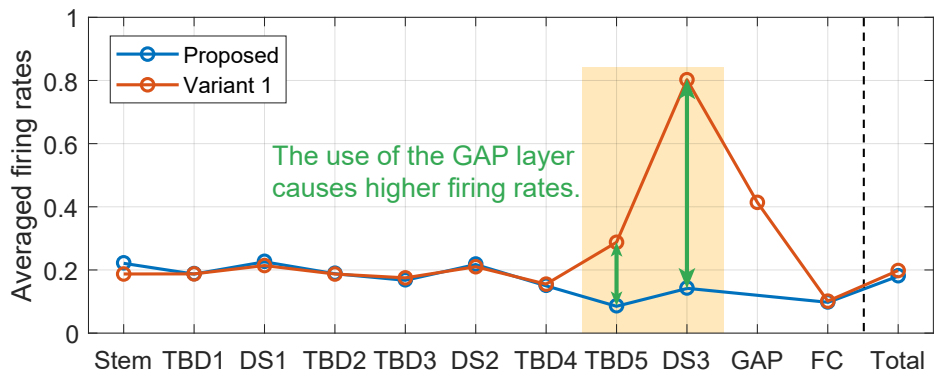
spaces in two aspects: 1) it does not have a global average pooling (GAP) layer, and 2) it is shallower and only utilizes a max pooling layer for down-sampling. To investigate the effects of these aspects, we compare the proposed search space with its variants, which are visualized in Figure 5.3. The experimental results obtained using `SCB_k3` for all TBD blocks in macro architectures are presented in Table 5.1. Following are detailed discussions, accompanied by the analysis of layer-wise spike generation pattern.

Usage of Global Average Pooling (GAP)

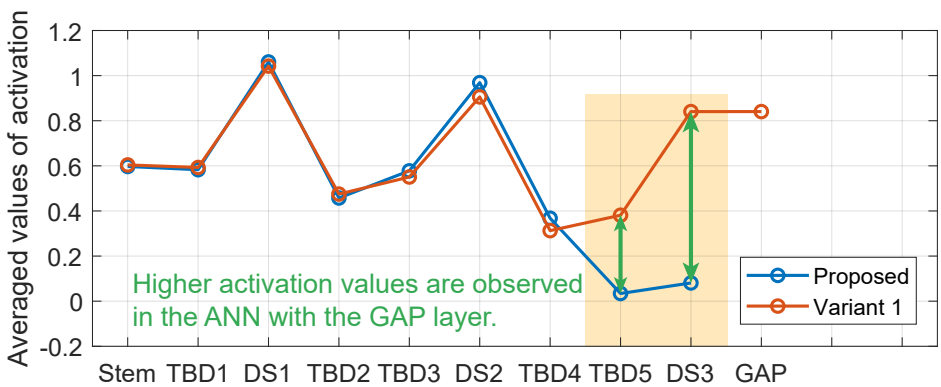
For Variant 1, we introduce a GAP layer [85] with spiking neurons to our macro architecture (before the FC layer), such that the number of parameters of the FC layer is reduced as in DNNs [53, 89, 112, 124]. Compared to our macro architecture, the accuracy significantly decreases, while an increased number of spikes are generated. As shown in Figure 5.4(a), in Variant 1 with `SCB_k3`, the number of spikes of the last TBD block (TBD5) and the last max pooling layer (DS3) before the GAP layer significantly increases when compared with the proposed macro architecture. In terms of the averaged firing rates per layer for 8 timesteps (Figure 5.4(b)), the firing rates of TBD5 and DS3 significantly affects the number of generated spikes, because the remaining layers of the proposed macro architecture and Variant 1 have similar firing patterns. DS3 and the GAP layer have $4\times4\times4C$ and $1\times1\times4C$ spiking neurons, respectively, where C is the initial channel of the architecture. Hence, in SNN_1 and SNN_2, the input feature map sizes of the FC layer are $4\times4\times4C$ and $1\times1\times4C$, respectively. Consequently, the use of the GAP layer reduces the number of spiking neurons that transmit spike-based information to the FC layer. To compensate for the information reduction caused by the GAP layer, the number of spikes and the firing rates of TBD5 and DS3 of SNN_2 significantly increase. Nevertheless, a non-



(a) The number of generated spikes



(b) Firing rates averaged over test data



(c) The activation proportion

Figure 5.4: Proposed macro architecture vs. Variant 1. (a) The number of the generated spikes. (b) Firing rates averaged over test data for 8 timesteps. (c) The activation values in feature map of each layer in the DNN versions of macro architectures that replace spiking neurons with ReLU activation functions, averaged over test data.

negligible amount of information reduction occurs because of the reduced number of spiking neurons, leading to the observed accuracy drop.

When we replace spiking neurons with ReLU activation functions in these two architectures (*i.e.*, DNN architectures) and train them, analogous phenomena are observed. As shown in Figure 5.4(c), for TBD5 and DS3, there are considerable differences in the average activation values between the two architectures. In TBD5 and DS3, the increase in the average activation values for ANNs and average firing rates for SNNs may be caused by an architectural property related to the GAP layer.

Meanwhile, when the GAP layer is used, the firing rates in the SNN and the activation values in the DNN behave differently. In DNNs, the averaged activation values of the GAP layer do not differ from those of DS3 because the GAP layer simply averages the activation values of DS3. However, in SNN, the firing rates of the GAP layer are lower than those of DS3. When membrane potential V_{mem} in the spiking neurons of the GAP layer does not exceed the threshold voltage V_{th}, the remaining membrane potential cannot be transmitted to the following FC layer, resulting in the information reduction. Table 5.1 reveals that for SNNs, the accuracy of Variant 1 is lower than that of the proposed macro architecture. On the contrary, for DNNs, the accuracies of both the proposed macro architecture and Variant 1 are approximately 90.5%, indicating that the information reduction caused by the GAP layer does not occur. Therefore, unlike DNNs, when the GAP layer is used in SNN, not only the total number of spikes increases, but also information reduction occurs, suggesting that the GAP layer is an unsuitable design choice for efficient SNNs.

Usage of TBD Blocks or Average Pooling Layers for Down-sampling

In previous studies, different types of layers have been employed as down-sampling layers: max pooling layer [37], convolutional layer [156], and average pooling layer

[138, 79], which are used in the proposed macro architecture, Variant 2, and Variant 3, respectively[1]. Thus, we investigate which down-sampling layer can lead to energy-efficient SNNs. As shown in Table 5.1, Variant 2 with `SCB_k3` achieves a 1% higher accuracy than the proposed macro architecture, because the use of trainable spiking blocks instead of max pooling layers increases the model capacity. However, the number of spikes considerably increases by approximately 44%. Variant 3 using average pooling layers experiences a significant drop in accuracy and also generates significantly increased spikes. The information reduction discussed in the previous section is also likely to occur in average pooling layers. Furthermore, we observed a large variance among the training results of Variant 3 with different seeds; this implies that the use of average pooling layers leads to unstable training. Hence, the use of trainable spiking blocks or average pooling layers is discouraged for the purpose of down-sampling.

We further compare the spike patterns of the proposed macro architecture and Variant 2. In Figure 5.5, the number of spikes in Variant 2 increases not only in the down-sampling layers (*i.e.*, DS1, DS2, and DS3) but also in their preceding layers (*i.e.*, TBD1, TBD3, and TBD5). The difference in the down-sampling layers is mainly caused by the difference between the size of feature map of the max pooling layer and the number of spiking neurons in `SCB_k3`. Thus, two convolutional layers with spiking neurons in `SCB_k3` generate more spikes, even though the firing rates of max pooling and `SCB_k3` are similar. We now lay down the potential reason behind the increase in the firing rates in the preceding layers. If a single input spike exists at least for the 2×2 kernel of the max pooling layer, this spike can be transmitted as the output spike. Hence, the max pooling layers transmit information through the spikes more efficiently than `SCB_k3`, and the preceding layers can generate fewer

[1]The kernel sizes of these pooling layers are 2×2.

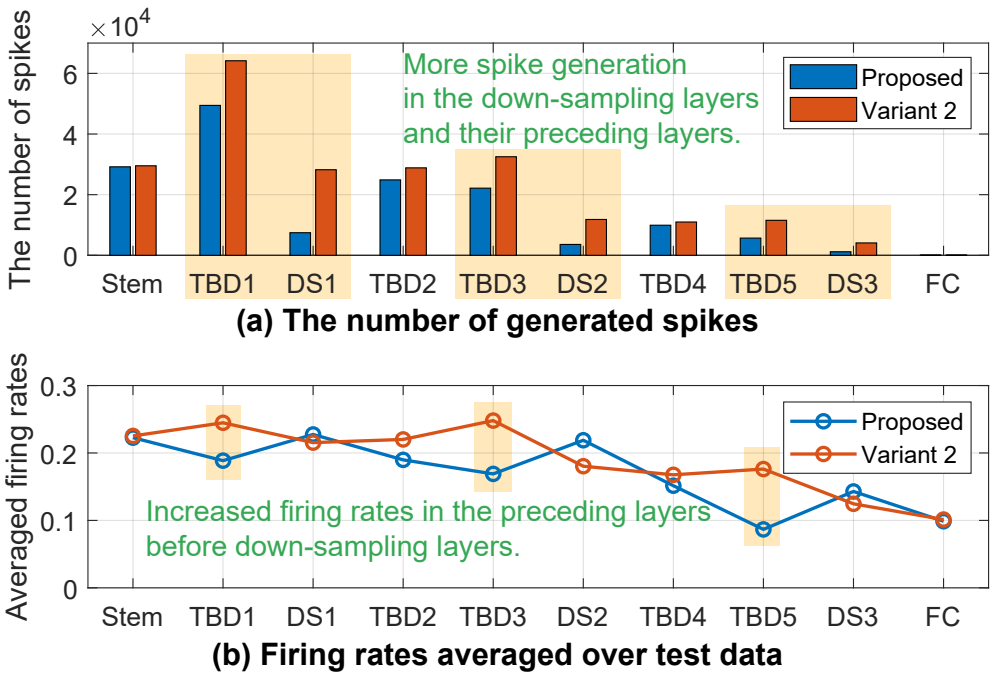


Figure 5.5: The number of spikes generated by the proposed macro architecture and Variants 2 with `SCB_k3`, and their firing rates averaged over test data for 8 timesteps.

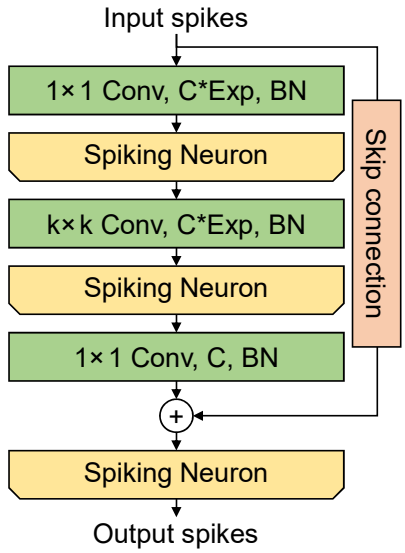


Figure 5.6: Spiking inverted bottleneck block (SIB).

Table 5.2: Evaluation of different design choices for the SNN search space on CIFAR-10. Architectures consist of a single type of spiking block.

All TBDs	GAP	Normal	Down-sample	Acc.(%)	Spikes
SCB_k3	✘	TBD	MaxPool	86.93	154K
SRB_k3				87.54	146K
SIB_k3_e1				81.07	243K
SIB_k3_e3				88.45	374K

spikes without loss of information. As a result, for energy-efficient SNNs, it is more desirable to use the max pooling layer for the down-sampling layers than the trainable spiking blocks.

Usage of Spiking Inverted Bottleneck Block (SIB)

At the micro-level, we additionally consider a spiking inverted bottleneck block (SIB), inspired by the inverted bottleneck structure of MobileNetV2 [112], which is illustrated in Figure 5.6. The inverted bottleneck structure can reduce the number of parameters and FLOPs in DNNs and is thus widely used to search for DNNs suitable to mobile devices [8, 9, 127, 136]. The design choices for SIB include kernel sizes

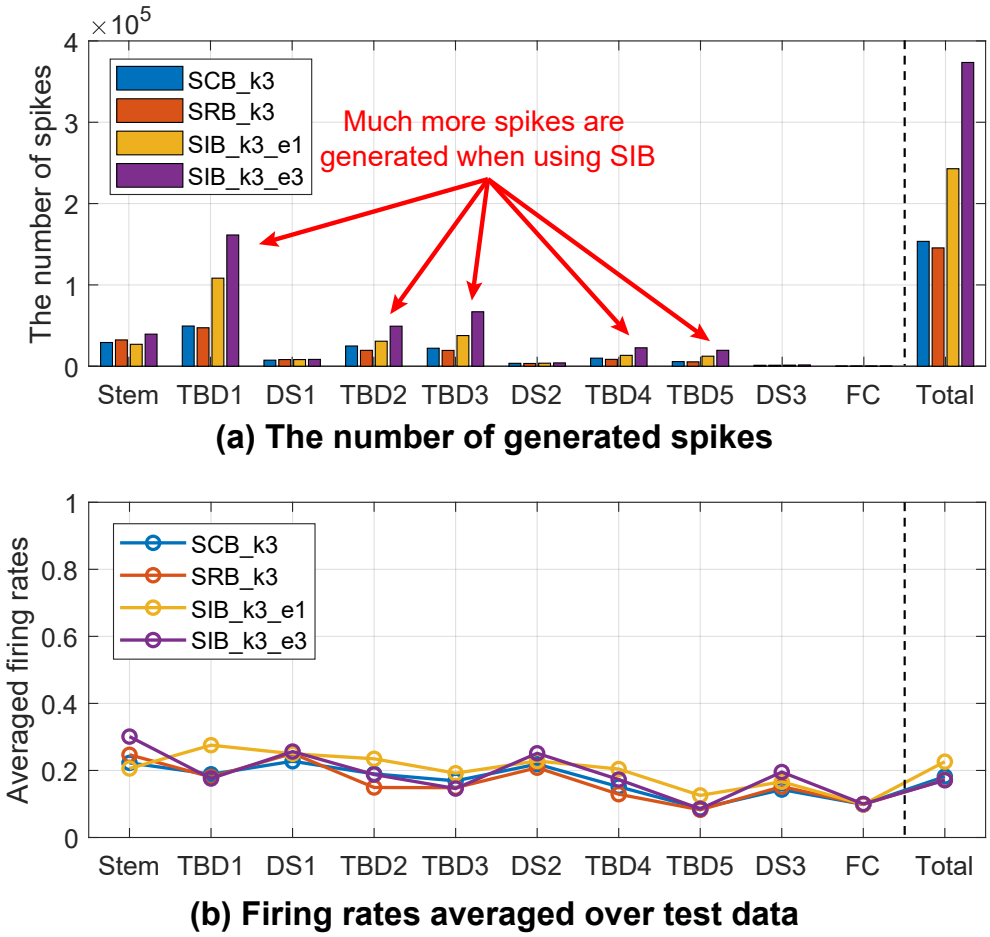


Figure 5.7: The number of spikes generated by architectures whose TBD blocks are filled with a single type of spiking block, and their firing rates averaged over test data for 8 timesteps.

$k \in \{3, 5\}$ and expansion ratios $e \in \{1, 3\}$. For simplicity, we denote SIB with $k = 3$ and $e = \{1, 3\}$ as `SIB_k3_e1` and `SIB_k3_e3`, respectively.

Table 5.2 presents the results of training SNNs, whose TBD blocks are filled with a single type of block with a kernel size of 3. SIB is significantly less desirable for efficient SNNs than SCB and SRB, even though the SNN with `SIB_k3_e3` improves the accuracy. The SNNs with `SIB_k3_e1` and `SIB_k3_e3` generate approximately 1.6x and 2.4x more spikes than that with `SCB_k3`, respectively.

The increase in the number of spikes is caused by architectural characteristics. Figure 5.7(a) reveals that all TBD blocks affect such increase in the number of spikes. As shown in Figure 5.7(b), the firing rates of these four blocks are similar, and thus the results are mainly caused by the differences in the number of spiking neurons of the spiking blocks. The number of neurons of each block is calculated as follows: $2hwc$ for `SCB_k3` (or `SRB_k3`), $hw(2c_{\text{in}} + c_{\text{out}})$ for `SIB_k3_e1`, and $hw(6c_{\text{in}} + c_{\text{out}})$ for `SIB_k3_e3`, where hw is a resolution of its input data, c_{in} is the number of channels of its input data, and c_{out} is the number of channels of its output data. Using the above equations, we obtain the number of spiking neurons of the proposed macro architectures with `SCB_k3`, `SIB_k3_e1`, and `SIB_k3_e3`: approximately $6.4HWC$, $8.0HWC$, and $16.5HWC$, respectively, where HW is a resolution of an image and C is an initial channel number (*e.g.*, 16 in the proposed macro architecture). By applying the total firing rates (0.18, 0.23, and 0.17, respectively, as shown in Figure 5.7(b)), we can calculate the approximated number of spikes: $1.17HWC$, $1.84HWC$, and $2.81HWC$, respectively. Hence, we can estimate that SNNs with `SIB_k3_e1` and `SIB_k3_e3` generate approximately 1.6x ($\simeq 1.84/1.17$) and 2.4x ($\simeq 2.81/1.17$) more spikes than that with `SCB_k3`; By comparing with the results in the lower group of Table 5.2, it is confirmed that the empirical results and our calculation results are consistent.

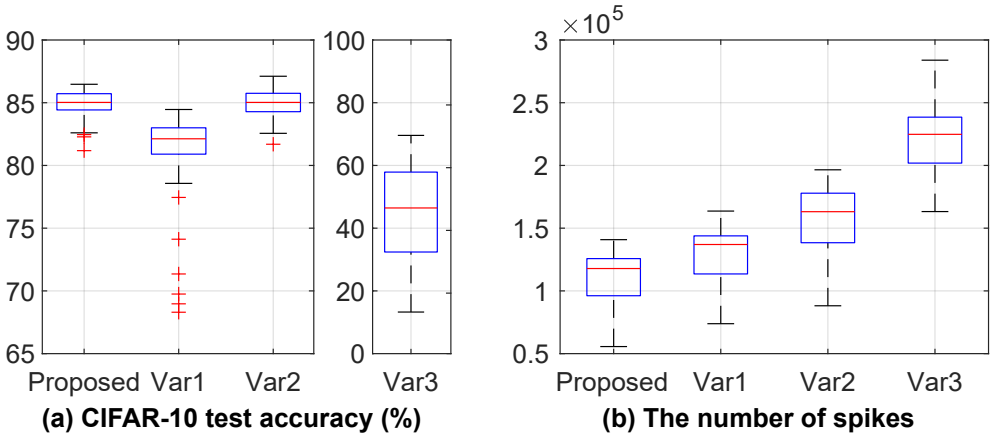


Figure 5.8: Search space quality comparison between the proposed search space and Variant $\{1, 2, 3\}$-based search spaces.

Once again, our experiments indicate that SIB, which has favorable properties for DNNs, is not suitable for efficient SNNs. Therefore, to find efficient SNNs, it is undesirable to include SIBs with a large number of spiking neurons in the micro search space, and five types of spiking blocks - `Skip`, `SCB_k3`, `SCB_k5`, `SRB_k3`, and `SRB_k5` - are used as candidate blocks in AutoSNN.

5.2.3 Search Space Quality Comparison

Herein, we evaluate the quality of the proposed search space in terms of the accuracy and number of spikes. We compare our proposed search space with three search spaces that consist of SNN architectures based on Variant $\{1, 2, 3\}$. For each search space, we generate 100 architectures by randomly choosing blocks from the predefined five candidate blocks to fill TBD blocks. All the generated SNN architectures were trained on CIFAR-10 for 120 epochs using a direct training method [37].

In Figure 5.8, the search space quality comparison empirically validates that the proposed search space is of higher quality than its variations in terms of the accuracy and number of spikes. This result is consistent with the analysis presented in

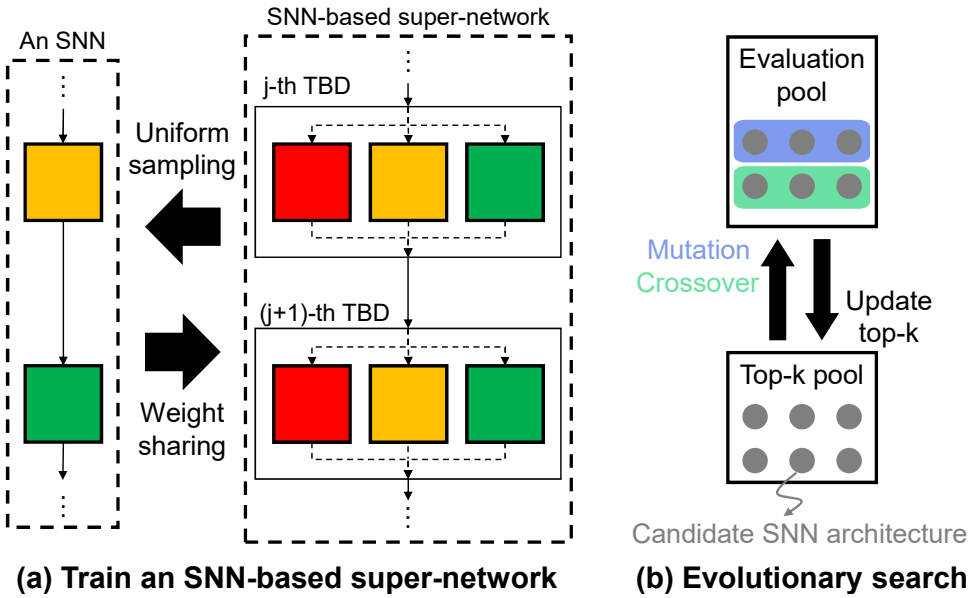


Figure 5.9: Search pipeline of AutoSNN that consists of two consecutive procedures: (a) training the stochastic super-network with uniform sampling of architectures and weight-sharing technique [105] and (b) spike-aware evolutionary search. The colored blocks in (a) represent candidate spiking blocks.

Section 5.2.2. The proposed search space includes architectures with higher accuracy and fewer spikes on average than those in the Variant 1-based search space. A similar but more distinctive pattern is observed in the Variant 3-based search space. Once again, this indicates that the use of the GAP layer and average pooling layers is inappropriate for finding performative and energy-efficient SNNs. In the Variant 2-based search space, the accuracy is comparable but the average number of spikes increases by approximately $\times 1.4$ over the proposed search space. Hence, the excessive use of spiking blocks with trainable parameters (*i.e.*, SCB and SRB) also needs to be avoided.

5.3 Proposed Search Method

In general, SNNs with timesteps take longer to train than DNNs. Thus, reducing the evaluation cost of each candidate architecture becomes even more important in NAS for SNNs. To address this challenge, we utilize the one-shot weight-sharing approach and propose AutoSNN, a spike-aware NAS framework, to find efficient SNNs from the proposed search space. As illustrated in Figure 5.9, the AutoSNN pipeline consists of two procedures: training of the super-network and evolutionary search.

5.3.1 Super-network Training

In one-shot weight-sharing NAS, sampling an architecture from the search space is equivalent to sampling a single path from the super-network as shown in Figure 5.9(a). To evenly train spiking blocks in the super-network, we adopt the single-path uniform sampling [3, 48, 84, 143, 155]. Based on the supervised training method of SNN [37], each sampled architecture is trained on a single mini-batch from training data. According to the weight-sharing, the updated weights of the architectures are shared with other architectures in the stochastic super-network. Afterwards, given the trained super-network $\mathcal{S}(W^*)$, all the architectures in the search space can be evaluated using validation data without additional training, because all sub-networks inherit their weights from W^*. Through the second procedure based on an evolutionary search in AutoSNN, the architecture with the highest fitness value will be discovered.

5.3.2 Spike-aware Evolutionary Search

To enable a spike-aware search, we first define the architecture fitness $F(A)$ as

Algorithm 1 Evolutionary search algorithm of AutoSNN.

Input: Trained super-network $S(W^*)$, validation data D_{val}

Parameter: Fitness coefficient λ, max round $\mathcal{T}$, mutation ratio ρ, the number of architectures in evolutionary way p_m and p_c, top-k pool size k, evaluation pool size p

Output: SNN A^* with the highest fitness

$\text{P}_{\text{eval}} = \text{P}_{\text{top}} = \emptyset$
for r = $1 : \mathcal{T}$ **do**
  if r == 1 **then**
    P_{eval} = RandomSample(p)
  else
    P_1 = Mutation(P_{top}, p_m, ρ)
    P_2 = Crossover(P_{top}, p_c)
    $\text{P}_{\text{eval}} = \text{P}_1 \cup \text{P}_2$
    if Size(P_{eval}) $< p$ **then**
      P_3 = RandomSample(p - Size(P_{eval}))
      $\text{P}_{\text{eval}} = \text{P}_{\text{eval}} \cup \text{P}_3$
    end if
  end if
  fitness values = Evaluate($S(W^*)$, D_{val}, P_{eval}, λ)
  P_{top} = UpdateTop$k$(P_{top}, P_{eval}, fitness values)
end for
return Top-1 SNN architecture A^* in P_{top}

follows:

$$F(A) = Accuracy \times (N/N_{\text{avg}})^{\lambda}, \tag{5.1}$$

where N is the number of spikes generated by architecture A, N_{avg} is the average number of spikes across architectures sampled during training of the super-network, and λ is the coefficient that controls the influence of the spike-aware term. Note that N is computed using validation data in the evaluation, whereas N_{avg} is computed using training data in the training of the super-network. The number of spikes cannot be available at DNN search space, *i.e.*, a super-network encoding candidate DNN architectures. To minimize the number of spikes generated by the searched SNNs while maximizing $F(A)$, we set $\lambda < 0$. As $|\lambda|$ increases, the SNNs discovered by AutoSNN are expected to generate fewer spikes.

Based on the obtained fitness value of each candidate architecture, AutoSNN explores the proposed search space and finds the architecture with the highest fitness value through an evolutionary search algorithm. Throughout the search process, depicted by Figure 5.9(b), AutoSNN maintains two population pools: the top-k population pool P_{top} with size k and the temporary evaluation population pool P_{eval} with size p. Along with Algorithm 1, we provide a detailed explanation of AutoSNN.

• **Preparation (lines 3-13)** First, P_{eval} is prepared with p architectures, which are randomly sampled from the search space or generated through mutation and crossover. We denote the number of architectures generated by mutation and crossover as p_m and p_c, respectively. In the first round, all p architectures are randomly sampled. For mutation, a parent architecture M is sampled from P_{top}, and each block in M is stochastically mutated with a mutation ratio ρ. For crossover, parent architectures M_1 and M_2 are sampled from P_{top}. The offspring architecture is generated by stacking the first X blocks in M_1 and the last $(5 - X)$ blocks in M_2; the value of X

is randomly sampled from $\{1, 2, 3, 4\}$. If an architecture that has already been evaluated is generated through mutation or crossover, it is not joined into the evaluation pool. As the search proceeds, and the architectures in P_{top} start to remain unchanged, it may become difficult to obtain a new architecture through crossover. When no new architectures are obtained through mutation or crossover, we fill P_{eval} by randomly sampling architectures.

• **Evaluation and P_{top} Update (lines 14 and 15)** All architectures in P_{eval} are evaluated based on their fitness values calculated using D_{val}. To update P_{top}, the top-k architectures are selected from P_{eval} and P_{top} of the previous round, based on their fitness values.

After repeating the aforementioned processes for $\mathcal{T}$ iterations, AutoSNN obtains architecture A^* with the highest fitness value from P_{top}. For all experiments, we set the parameters used in Algorithm 1 as follows: $\mathcal{T} = 10$, $\rho = 0.2$, $p_m = 10$, $p_c = 10$, $k = 10$, and $p = 20$.

Because AutoSNN decouples the training and search processes, it can search for an optimal SNN architecture for every different neuromorphic hardware architecture by simply changing λ. Similar to a previous study based on the one-shot NAS approach [9], AutoSNN can reuse the trained super-network and execute the second procedure alone. Therefore, our search algorithm based on two separate procedures is a practical and effective method for obtaining efficient SNNs.

5.4 Experiments and Results

5.4.1 Experimental Settings

AutoSNN and all the experiments are implemented using SpikingJelly[2]. We evaluate the SNNs searched by AutoSNN on two types of datasets: static datasets (CIFAR-10 [75], CIFAR-100 [75], SVHN [100], and Tiny-ImageNet-200[3]) and neuronmorphic datasets (CIFAR10-DVS [83] and DVS128-Gesture [1]). Details regarding these datasets are provided in Section 5.4.2.

During the dataset is divided into 8:2 for training data and validation data. An identical training setting is used to train the super-network and the searched SNNs: we used Adam [70] optimizer with a learning rate of 0.001 and cutout data augmentation [26] to train them for 600 epochs on a single NVIDIA GeForce RTX 2080ti GPU. For all architectures, we use PLIF neurons [37] with $V_{\mathrm{th}} = 0$, $V_{\mathrm{reset}} = 0$, 8 timesteps, and an initial τ of 2.

Super-network Training

To train the super-network, we employed the Adam optimizer [70] with a fixed learning rate of 0.001 and a momentum of (0.9, 0.999). The super-network was trained for 600 epochs with a batch size of 96. During the training, we applied three conventional data preprocessing techniques: the channel normalization, the central padding of images to 40×40 and then random cropping back to 32×32, and random horizontal flipping. This setting was also applied to train the super-network that encodes DNN search space; the related experimental result is explained in Section 5.5.2.

[2]https://github.com/fangwei123456/spikingjelly

[3]https://www.kaggle.com/akash2sharma/tiny-imagenet

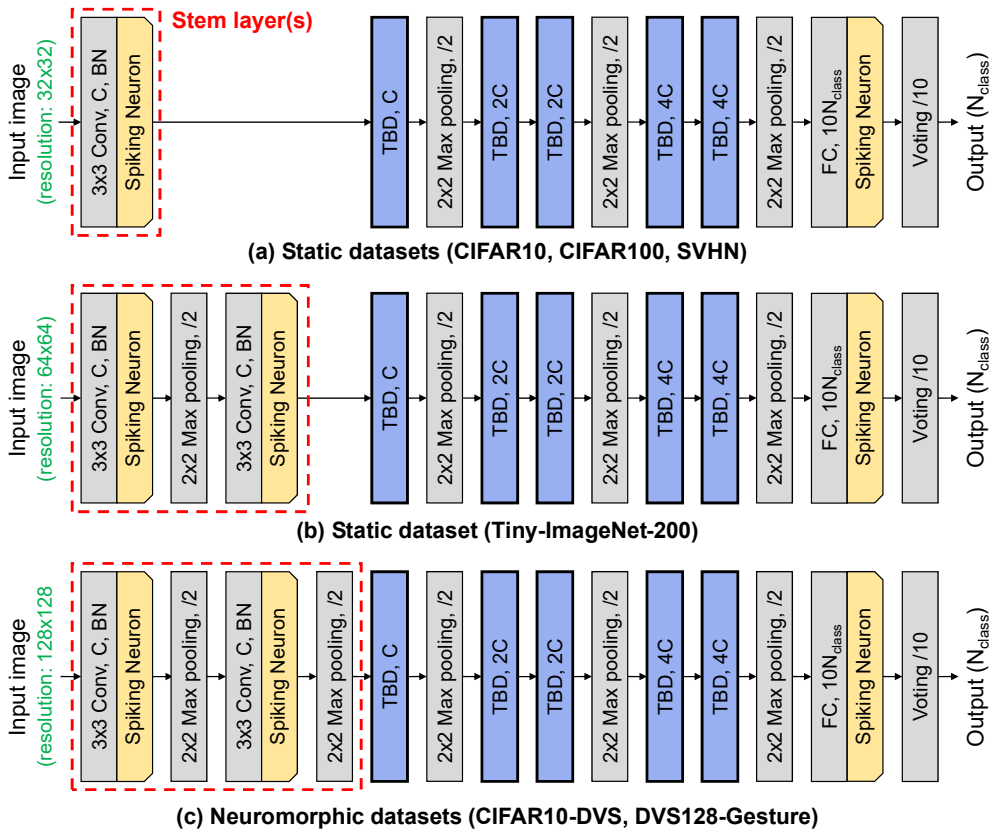


Figure 5.10: The macro architectures used for various datasets: (a) CIFAR-10, CIFAR-100, and SVHN with a resolution of 32×32, (b) Tiny-ImageNet-200 with a resolution of 64×64, and (c) CIFAR10-DVS and DVS128-Gesture with a resolution of 128×128. To reduce the resolution of image into 32×32, architectures for Tiny-ImageNet-200 and neuromorphic datasets include additional layers in stem layers.

Searched SNN Training

All the SNNs including architectures searched by AutoSNN and conventional architectures were trained from scratch for 600 epochs with a batch size of 96; some SNNs, where the batch size of 96 was not executable due to the memory size, were trained with a smaller batch size. We also used the Adam optimizer [70] with a fixed learning rate of 0.001 and a momentum of (0.9, 0.999). For the static datasets, we additionally applied cutout data augmentation with length 16 [26], along with the three conventional data augmentation techniques applied in the super-network training. When training Tiny-ImageNet-200, a max pooling and a 3×3 convolution layer are sequentially added into the stem layer of our macro architecture backbone, as shown in Figure 5.10(b). For the neuromorphic datasets with a resolution of 128×128, we constructed deeper stem layer in order to reduce the resolution from 128×128 to 32×32 such that the subsequent layers searched by AutoSNN can process the data, as shown in Figure 5.10(c).

5.4.2 Dataset Description

Static Datasets

CIFAR-10, CIFAR-100, and SVHN include images with a resolution of 32×32 and 3 channels (RGB channels). An image corresponds to one static frame with pixel values, and thus we refer to these datasets as the static datasets. CIFAR-10 and CIFAR-100 are composed of 50,000 training data and 10,000 test data, while SVHN has approximately 73K images for training and 26K images for testing. Tiny-ImageNet-200 includes images with a resolution of 64×64 and 3 channels, where the images are sampled from ImageNet [111] and downsized from 224×224 to 64×64. 100K images and 2.5K images are used for training and testing, respectively. These datasets

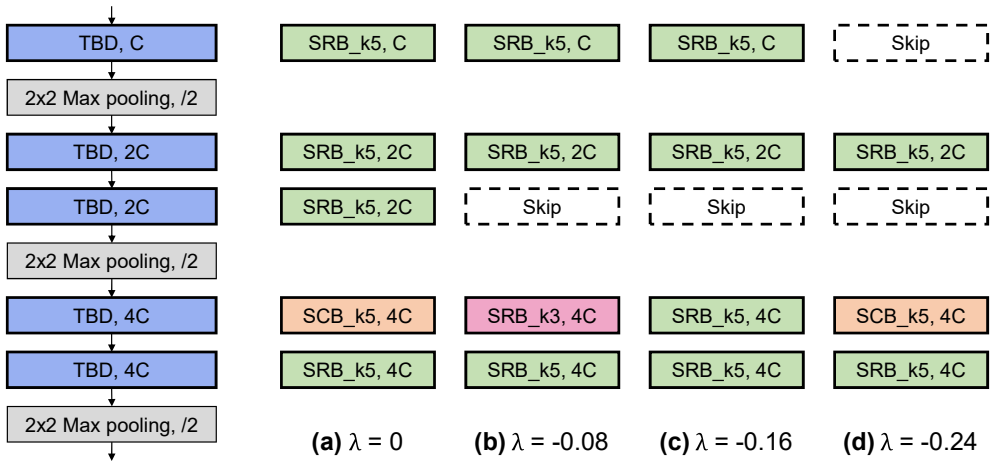


Figure 5.11: Architectures searched by AutoSNN with different λ on the proposed search space.

are used in the classification task: 10 classes for CIFAR-10 and SVHN, 100 classes for CIFAR-100, and 200 classes for Tiny-ImageNet-200.

Neuromorphic Datasets

We evaluate SNN architectures on the neuromorphic datasets, which include data with a format of event stream, referred to as a spike train in our context. The datasets are collected by using a dynamic vision sensor (DVS), which outputs 128×128 images with 2 channels. For CIFAR10-DVS [83], 10,000 images from CIFAR-10 are converted into the spike trains, and there are 1,000 images per class. We divide the dataset into two parts: 9,000 images for the training data and 1,000 images for the test data. DVS128-Gesture [1] includes 1,342 training data and 288 test data with 11 classes of hand gestures.

Table 5.3: Evaluation on CIFAR-10 with different λ varying from 0 to -0.24.

Fitness	Params (M) $\downarrow$	Test accuracy (%) $\uparrow$	Spikes $\downarrow$
$\lambda = 0$	0.53	88.69	127K
$\lambda = -0.08$	0.42	88.67	108K
$\lambda = -0.16$	0.52	88.46	106K
$\lambda = -0.24$	0.50	86.58	54K

5.4.3 Searching Results with Different λ of Spike-aware Fitness

To show the effect of λ of the fitness value on the search result of AutoSNN, we execute AutoSNN with varying values of λ and report the results in Table 5.3; the architecture searched under each setting is visualized in Figure 5.11. Note that on a single 2080ti GPU, the search cost is approximately 7 GPU hours: 6 hours 48 minutes for training a super-network and 8 minutes for executing the evolutionary search. Varying the value of λ to execute the spike-aware evolutionary search incurs a negligible additional cost (8 minutes).

In Figure 5.11, distinctive differences among SNN architectures searched with different λ are observed. For $\lambda = 0$, indicating that only accuracy is considered during executing the evolutionary search algorithm, the searched architecture includes five spiking blocks with trainable parameters, and thus has the highest model complexity among the searched architectures. For $\lambda = -0.08$ and -0.16, the third TBD in the architectures is `Skip`. As presented in Table 5.3, these architectures experience a slight drop in accuracy while reducing approximately 20K spikes, compared to the architecture for $\lambda = 0$. In the architecture searched with $\lambda = -0.24$, the first and third TBD blocks are filled with `Skip`. The spikes generated by this architecture are much fewer than the other architectures, but the accuracy decreases by approximately 2%p. These searching results confirm that λ functions according to our intent: to adjust the accuracy-efficiency trade-off in the searched SNN. Hence, in the other

Table 5.4: Comparison with architectures on CIFAR-10.

Architecture	An initial channel C	Params (M) ↓	Test accuracy (%) ↑	Spikes ↓
CIFARNet-Wu [138]	16	0.71	84.36	361K
	32	2.83	86.62	655K
	64	11.28	87.80	1298K
	†128	45.05	‡90.53	-
CIFARNet-Fang [37]	16	0.16	80.82	104K
	32	0.60	86.05	160K
	64	2.34	90.83	260K
	128	9.23	92.33	290K
	†256	36.72	93.15	507K
ResNet11-Lee [79]	16	1.17	84.43	140K
	32	4.60	87.95	301K
	†64	18.30	‡90.24	‡1530K
ResNet19-Zheng [156]	16	0.23	83.95	341K
	32	0.93	89.51	541K
	64	3.68	90.95	853K
	†128	14.69	93.07	1246K
AutoSNN (proposed)	16	0.42	88.67	108K
	32	1.46	91.32	176K
	64	5.44	92.54	261K
	128	20.92	93.15	310K

† denotes the initial channel used in previous studies.
‡ denotes reported values in original papers, which could not reproduce in our training setting.

experiments, we use $\lambda = -0.08$ for AutoSNN by default.

Furthermore, we observe that the architectures searched on the proposed SNN search space prefer spiking blocks with a kernel size of 5 (*i.e.*, `SCB_k5` and `SRB_k5`). It would be also interesting to investigate architectural properties related to such preference of SNNs.

5.4.4 Comparison with Existing SNNs

During the search process in AutoSNN, an initial channel of 16 is used for the macro architecture. Hand-crafted SNNs used in previous studies have a noticeably larger number of initial channels; † in Table 5.4 indicates the default initial channel used in each study. Hence, for a fair comparison, we increase the number of initial channels in the searched SNN architectures and decrease that in other SNNs by a factor of 2. We train all of the compared networks under the same training setting. The training results on CIFAR-10 are provided in Table 5.4 and the visual summary of the quantitative results is presented in Figure 5.1.

The experimental results do indeed support that AutoSNN successfully searches for an efficient SNN. The SNN searched by AutoSNN with $\lambda = -0.08$ lies on the Pareto front in the region of accuracy and the number of spikes; refer back to Figure 5.1. Accuracy-wise, CIFARNet-Wu [138] and CIFARNet-Fang [37] are comparable to AutoSNN, but their architecture size, measured in the number of parameters, is significantly larger than that of AutoSNN. While ResNet19-Zheng [156] also achieves a competitive performance, it generates an exceedingly large number of spikes; for instance, ResNet19-Zheng with 128 channels generates $4\times$ more spikes than AutoSNN with 128 channels.

Figure 5.1 and Table 5.4 present that AutoSNN discovers the more energy-efficient SNN than hand-crafted architectures used in previous studies. In Figure 5.12, the superiority of AutoSNN is also observed even when training SNNs with different timesteps, 4 and 16. This demonstrates that the hand-crafted SNNs have undesirable architectural components and the NAS-based approach can be essential to improve energy efficiency.

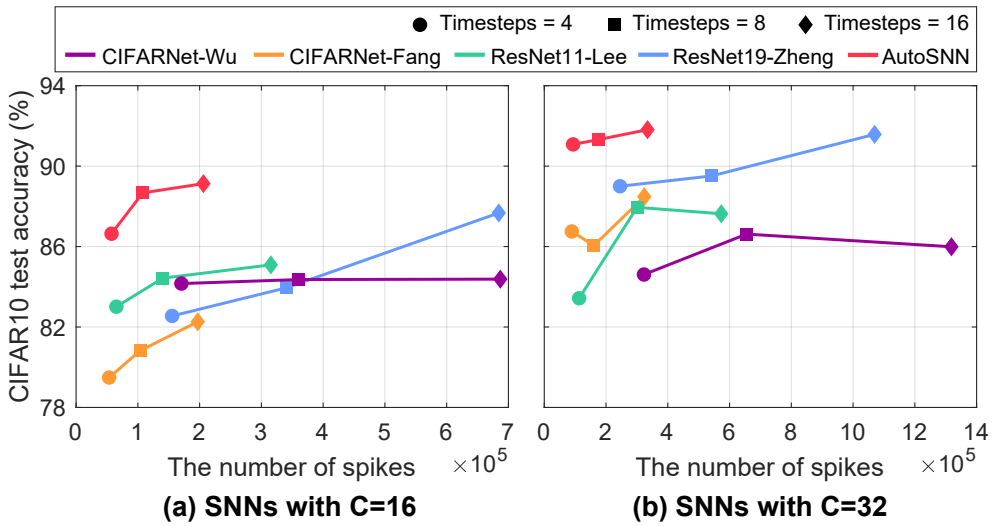


Figure 5.12: The number of spikes vs. CIFAR10 accuracy of various SNN architectures trained with different timesteps $\in \{4, 8, 16\}$.

5.4.5 Comparison on Different Datasets

Following the standard practice in NAS, we transfer the SNN architecture searched by AutoSNN to various datasets, which are different from the dataset used in the search process, to further evaluate the generalization performance of the searched architecture. For CIFAR-100 and SVHN, an initial channel of the architecture is set to 64 to account for the increased complexity of these datasets. For the two neuromorphic datasets, CIFAR10-DVS and DVS128-Gesture, we add additional layers to the stem layer of the macro architecture in AutoSNN, which is visualized in Figure 5.10. Table 5.5 clearly reveals that AutoSNN is more efficient (*i.e.*, lower number of generated spikes) and achieves higher accuracy than other hand-crafted architectures across all datasets.

We also evaluate the generalization performance using Tiny-ImageNet-200, and compare CIFARNet-Fang [37] that shows the most desirable curve of efficiency-accuracy in Figure 5.1 with an exception of AutoSNN. For all the different initial

Table 5.5: Comparison with recent studies on different datasets.

Data	SNN Architecture	C	Test accuracy (%) ↑	Spikes ↓
CIFAR-100	Fang *et al.*2020	256	66.83	716K
	AutoSNN	64	69.16	326K
SVHN	Fang *et al.*2020	256	91.38	462K
	AutoSNN	64	91.74	215K
CIFAR10-DVS	Wu *et al.*2019b	128	‡60.50	-
	Fang *et al.*2020	128	69.10	4521K
	Zheng *et al.*2021	64	66.10	1550K
	AutoSNN†	16	72.50	1269K
DVS128-Gesture	He *et al.*2020	64	‡93.40	-
	Kaiser *et al.*2020	64	‡95.54	-
	Fang *et al.*2020	128	95.49	1459K
	Zheng *et al.*2021	64	96.53	1667K
	AutoSNN†	16	96.53	423K

† denotes architectures with additional layers in the stem layer for neuromorphic datasets.
‡ denotes reported values in original papers.

Table 5.6: Comparison using Tiny-ImageNet-200.

SNN Architecture	C	Test accuracy (%) ↑	Spikes ↓
CIFARNet-Fang†	32	37.73	396K
	64	40.75	701K
	128	42.59	993K
	256	45.43	1724K
AutoSNN†	16	39.13	240K
	32	44.77	419K
	64	46.79	680K

† denotes architectures with additional layers in the stem layer for Tiny-ImageNet-200.

Table 5.7: Ablation study results of AutoSNN on CIFAR-10. WS is a shorthand for weight-sharing.

Search algorithm	Test accuracy (%) ↑	Spikes ↓
Random sampling (10 architectures)	86.97±1.06	123K±29K
WS + random search		
$\lambda = 0$	88.40	132K
$\lambda = -0.08$	88.10	133K
WS + spike-aware evolutionary search (AutoSNN)		
$\lambda = 0$	88.69	127K
$\lambda = -0.08$	88.67	108K

channels, the results in Table 5.6 again demonstrate the superiority of the searched architecture compared to CIFARNet-Fang. These results support that AutoSNN is not limited to a certain dataset and the searched architecture can be utilized in other vision tasks.

5.5 Discussion

5.5.1 Effectiveness of Components in AutoSNN

We conduct ablation study to validate the effectiveness of the two main components in AutoSNN: the super-network training based on weight-sharing and the spike-aware evolutionary search algorithm. First, we train 10 architectures randomly sampled from the proposed SNN search space (Random sampling) and report the average accuracy in Table 5.7. Second, using a super-network trained by the first procedure of AutoSNN, we randomly sample a total of 200 architectures from the super-network and obtain the top-1 architecture by evaluating them according to the proposed fitness value (WS + random search). The number of evaluated architectures (*i.e.*, 200) is the same search budget in the spike-aware evolutionary search. In Table 5.7, we report the search results of WS + random search for two different values of λ. All

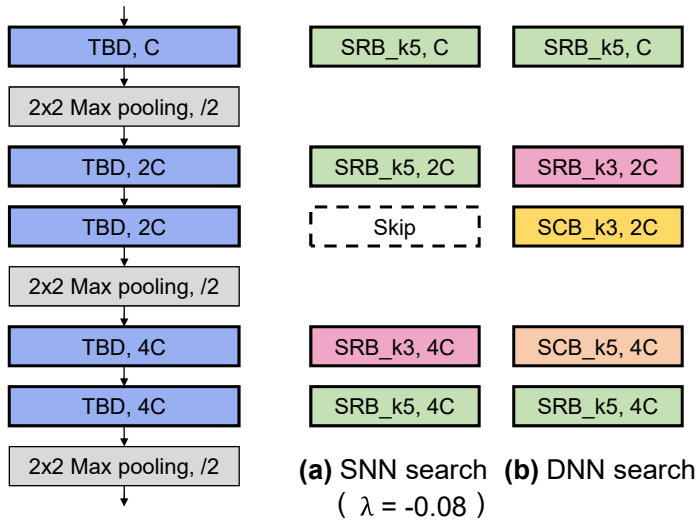


Figure 5.13: (a) Architecture searched by AutoSNN from the proposed SNN search space and (b) architecture searched by two procedures: training a DNN version of super-network and an evolutionary search with λ=0.

Table 5.8: Search results from DNN and SNN search spaces.

Search space ($\lambda = 0$)	Test accuracy (%) ↑	Spikes ↓
DNN search space	88.02	134K
SNN search space (AutoSNN)	88.69	127K

the experiments are conducted on CIFAR-10, using the same experimental settings as those described in Section 5.4.1.

As shown in Table 5.7, SNNs searched by WS + random search yield higher accuracy than the average accuracy of random sampling. This suggests that the weight-sharing strategy, one of the most popular proxy techniques in NAS for DNNs, is also valid in exploring the SNN search space. Applying the spike-aware evolutionary search further improves the searching results, solidifying the necessity of spike-awareness in AutoSNN.

Table 5.9: Evaluation for AutoSNN on enlarged search spaces (C=16).

Search space	Architecture	Test acc. (%)	Spikes
Using TBD blocks instead of max pooling layers (Variant 2)	`SCB_k3` in all TBDs	87.94	222K
	`SRB_k3` in all TBDs	89.22	221K
	AutoSNN ($\lambda = -0.04$)	89.05	170K
	AutoSNN ($\lambda = -0.08$)	87.92	65K
Adding a TBD block before each max pooling layer	`SCB_k3` in all TBDs	87.04	230K
	`SRB_k3` in all TBDs	88.69	228K
	AutoSNN ($\lambda = -0.08$)	88.60	143K
	AutoSNN ($\lambda = -0.16$)	87.29	60K

5.5.2 DNN Search based on AutoSNN

We investigate the result of executing AutoSNN procedures on a DNN search space. To construct a DNN search space, we remove spiking neurons from our macro architecture and the spiking blocks. Because no spikes are generated in the DNN search space, the search procedures only consider the accuracy, which is equivalent to setting $\lambda = 0$ in the fitness; this approach cannot reflect the distinct property of SNNs such as spike generation. Training a DNN-based super-network and searching the top-1 architecture based on our evolutionary algorithm require approximately 1.3 GPU hours. After searching, the searched DNN architecture is converted to an SNN by adding spiking neurons, and then trained following the same settings in Section 5.4.1.

As shown in Table 5.8, the architecture searched by AutoSNN with $\lambda = 0$ achieves higher accuracy and generates fewer spikes than that searched from the DNN search space. Figure 5.13 presents the architecture from the DNN search space. We conjecture that this discrepancy in search results occurs because the DNN super-network cannot reflect properties of SNNs such as neural dynamics of spiking neurons or information transmission in spike trains.

5.5.3 Validity on Enlarged Search Space

The proposed SNN search space consists of 3,125 architectures (5^5; 5 candidate blocks and 5 TBD blocks). Even if this search space is smaller than that of NAS for DNNs, it is computationally intractable to manually evaluate all of them without NAS. Assuming the average training cost of a single SNN is 7 hours, it would take 1,400 hours to evaluate 200 architectures without AutoSNN; 200 architectures are the same architecture budget of AutoSNN. In contrast, the search cost of AutoSNN for evaluating 200 architectures is merely 7 hours, showing the efficiency of NAS.

Meanwhile, AutoSNN is also effective to search for desirable architectures in enlarged search spaces. We construct two search spaces, where a macro architecture includes 8 TBD blocks as described in Table 5.9. One is Variant 2, where max pooling layers of our macro architecture are replaced by TBD blocks, and the other is a new variant where TBD blocks are added before each max pooling layer of our macro architecture. The size of both of these search spaces is increased to 390,625 (5^8). Table 5.9 reveals that AutoSNN successfully searches for energy-efficient SNNs in both of enlarged search spaces.

5.6 Summary

For energy-efficient artificial intelligence, it is essential to design SNNs that have minimal spike generation and yield competitive performance. In this study, we proposed a spike-aware fitness and AutoSNN, the first spike-aware NAS framework to effectively search for such SNNs from the energy-efficient search space that we defined. To define the search space, we analyzed the effects of the architecture components on the accuracy and number of spikes. Based on our findings, we suggested excluding the GAP layer and employing the max pooling layers as down-sampling

layers in SNNs. From the search space that consists of SNN architectures satisfying these design choices, AutoSNN successfully discovered the SNN architecture that is most performative and energy-efficient compared with various architectures used in previous studies. Our results highlighted the importance of architectural configurations in the SNN domain. We anticipate that this study will inspire further research into automatic design of energy-efficient SNN architectures.

Chapter 6

Conclusion

Throughout this dissertation, we have comprehensively covered topics being actively studied in NAS, including how to reduce the search cost, how to improve a search space and a search algorithm, and how to apply NAS to research fields where architectural consideration is important but overlooked. In this chapter, we summarize our contributions regarding these topics before concluding with a discussion of future research directions of NAS including our methods.

6.1 Dissertation Summary

In Chapter 3, we introduced proxy data, which is a representative subset of dataset, to further reduce the search cost of NAS algorithms. We observed that existing data sampling methods used for coreset selection and active learning were insufficient to be used for NAS, because they did not maintain the search performance, which can be obtained using the entire target data. We analyzed the searching results on NAS benchmarks with various proxy data, and thereby obtained the two insights on characteristics of proxy data suitable for NAS. One is that when a small number of examples are selected, easy examples with low entropy contribute more to stabiliz-

ing the search performance than difficult examples. The other is that when proxy data includes a sufficient number of easy examples, adding difficult examples is crucial for achieving the original search performance. Therefore, to satisfy these characteristics, we proposed a novel selection method that prefers examples belonging in tail-end of entropy distribution of the target data, *i.e.*, a combination of easy and difficult examples in the target data. We implemented the proposed method in two ways: mixing easy and difficult examples with a fixed composition ratio and sampling examples using a probability that reflects such data preference. The sampling method was more applicable to various datasets than the deterministic method with a fixed composition ratio, because the optimal composition ratio suitable for NAS can be different in the datasets. We thoroughly demonstrated the NAS acceleration and applicability of the proposed probabilistic selection method on various datasets and NAS algorithms. A direct search on cell-based search space using ImageNet with our proposed sampling method was completed in 7.5 GPU hours, which is similar to the search cost when using all the examples of CIFAR-10. Remarkably, the searched architecture achieved the state-of-the-art performance without any modification of the NAS algorithm, implying that searching on proxy data of large-scale datasets and then transferring searched architectures to small-scale datasets, referred to as inverse transfer, is valid. With the proposed sampling method, we can effectively execute NAS algorithms on various datasets with the significant search cost.

In Chapter 4, we considered the conventional cell-based search space, referred to as DARTS cell space. With a technique to use mixed operations in a super-network, differentiable NAS methods on the DARTS cell space approximately evaluate diverse cell structures, which can be derived from the super-network cell but are not included in the DARTS cell space. To define these cell structures, we introduce an extension of DARTS cell space, called ExtCell space. Through evaluation using a proxy network,

we observed that the quality of ExtCell spaces (*i.e.*, performance distribution of cells) is comparable to the DARTS cell space. To derive ExtCells from the super-network cell, we treated NAS as a global survival problem of all the candidate operations and proposed a top-k survival rule, where k is operation budget. Conventional NAS approach is based on a local competition among candidate operations at TBD edges; that is, even if operations at a TBD edge denoted by e are more promising than the operation that is selected at another TBD edge, only one operation can be selected at e. On the contrary, through the top-k survival rule, these more promising operations of e can be selected. Intuitively, in NAS for the global survival problem, operation strengths should be individually modeled. However, we observed that conventional differentiable NAS methods were not compatible with the ExtCell derivation based on the top-k survival rule. To address the issue, we proposed BtNAS, which models operation strengths individually as continuous random variables following Beta distributions. In BtNAS, the true posterior distributions of operation strengths are approximated using the variational Bayes method, the parameters of variational Beta distributions are optimized through the pathwise derivative estimator. Compared with state-of-the-art DARTS cells, the ExtCells searched by BtNAS achieved competitive performance and yielded networks with fewer parameters. We evaluated search robustness of BtNAS and analyzed it in terms of loss landscape smoothness, which was leaded by stochasticity of BtNAS. Furthermore, to improve search performance of BtNAS, we added a cell derivation policy that ensures all intermediate nodes to be activated by normalizing operation strengths at each node, and thereby BtNAS achieved the state-of-the-art search performance. Through this chapter, we demonstrated that it is beneficial to search for cell structures that have been overlooked in conventional NAS algorithms using a super-network cell.

In Chapter 5, we seamlessly applied NAS approach to search for energy-efficient

SNNs. Using neuromorphic chips, SNNs can work with low energy consumption because of event-driven computation. In SNNs, information is transmitted between spiking neurons through a spike train, which is correlated to energy consumption in SNNs. It is critical to optimize SNNs to maximize performance with minimal spike generation. To this end, however, only training methods for SNNs have been studied without any consideration on suitability of architectures that have been conventionally used for SNNs. In this chapter, we demonstrated that it is also important to select architectures to improve the performance and reduce the spike generation of SNNs. We proposed AutoSNN, the first spike-aware NAS framework consisting of a two-level search space tailored to efficient SNNs and a spike-aware evolutionary search algorithm based on one-shot weight-sharing NAS. To construct the search space, we empirically evaluated design choices, which are adopted in NAS methods for DNNs, such as using a global average pooling and inverted bottleneck blocks as candidate operations. The evaluation confirms that SNNs have different dynamics and preferred design choices from DNNs. Following the one-shot weight-sharing NAS, we implemented two procedures of a search algorithm of AutoSNN. First, a supernetwork is trained using a direct training method that offers more energy-efficiency than indirect training methods. Then, an evolutionary algorithm is executed with a spike-aware fitness to evaluate architectures during the search process. Hence, AutoSNN could effectively consider both the accuracy and number of generated spikes of architectures, and meaningfully improved the performance and energy-efficiency of SNNs by finding a desirable architecture. Thorough experimental results across various datasets also supported the effectiveness of AutoSNN. In this chapter, our results highlighted the importance of architectural configurations on the accuracy and efficiency of SNNs, and AutoSNN has unleashed the unrealized potential of NAS in the context of SNNs.

6.2 Suggestions for Future Research

6.2.1 More Reliable Method to Accelerate Search Algorithms

We introduced an effective approach to accelerate search algorithms by reducing the size of target datasets, *i.e.*, using proxy data. Compared to original search performance that was obtained using the entire data, NAS algorithms with the proposed sampling method for proxy data maintained their search performance. Nevertheless, operation compositions in searched architectures were different; for example, DARTS with the entire data was prone to find cell structures consisting of multiple skip connection operations [148], but cells searched by DARTS with our sampled proxy data mainly included operations equipped with trainable parameters, such as sep. conv. and dil. conv. operations. It would be meaningful to theoretically analyze and study on how the data entropy distribution used in the search process affects the searched architectures. Also, to generally apply to various NAS algorithms, an acceleration method with high reliability is required, in which searching results are also maintained. Developing a more reliable acceleration method, which makes the search performance of the search algorithms similar as well as the searching results, will greatly contribute to expanding the base of NAS and applying to larger-scale tasks in reality.

Recently, several studies [13, 86, 96, 141] have introduced an interesting approach to enable NAS to be completed without training a super-network or any architecture; this is referred to as zero-cost NAS. The approach is based on the assumption that expressivity and trainability of architectures are quantified and thus their performances (*e.g.*, test accuracy) can be estimated. Meanwhile, White *et al.* [135] evaluated the reliability of NAS without training and reported the limitation. Improving the reliability of the zero-cost NAS is a promising direction for future work.

6.2.2 Automatic Search Space Design

In Chapters 4 and 5, we highlighted the importance of search spaces in NAS for improving search quality and applying NAS to various applications. Each application has different preferred design choices for DNNs, and thus it is necessary to define a search space suitable for each application. Although it is less computationally expensive than designing a proper DNN for applications, constructing an appropriate search space still requires manual effort, such as search spaces to search for generative adversarial networks [42] and graph neural networks [38]. Aiming to minimize the manual effort, automatic design of a search space will have significant impact on deep learning community.

To this end, we need to move towards constructing a framework, which consists of a search space library including a variety of architecture types and search space selection methods. For the search space selection, it will be essential to analyze a search space and quantify the quality of a search space. Also, search space shrinking, which has been employed as a technique to improve search performance and efficiency [15, 16, 87, 106], is aligned with the future direction for the automatic search space design. In line of this approach, Radosavovic *et al.* [106] presented a pioneer study with promising results. They prepared a general search space, which is constructed by combinations of various design choices such as width and depth of networks. Then, using the search space quality evaluation technique, the general search space is gradually shrunk and becomes suitable for the given task. To enable NAS to be a general solution with deep learning, the research for automatically designing a search space suitable for an application will need to be continually progressed.

Bibliography

[1] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, et al. A low power, fully event-based gesture recognition system. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[2] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning*, 2016.

[3] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, 2018.

[4] Gabriel Bender, Hanxiao Liu, Bo Chen, Grace Chu, Shuyang Cheng, Pieter-Jan Kindermans, and Quoc V Le. Can weight sharing outperform random architecture search? an investigation with tunas. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[5] Yassine Benyahia, Kaicheng Yu, Kamil Bennani Smires, Martin Jaggi, An-

thony C. Davison, Mathieu Salzmann, and Claudiu Musat. Overcoming multi-model forgetting. In *International Conference on Machine Learning*, 2019.

[6] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112 (518):859–877, 2017.

[7] Sander M Bohte, Joost N Kok, and Han La Poutre. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48(1-4): 17–37, 2002.

[8] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

[9] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020.

[10] Haw-Shiuan Chang, Erik Learned-Miller, and Andrew McCallum. Active bias: Training more accurate neural networks by emphasizing high variance samples. In *Advances in Neural Information Processing Systems*, 2017.

[11] Jianlong Chang, xinbang zhang, Yiwen Guo, Gaofeng Meng, Shiming Xiang, and Chunhong Pan. Data: Differentiable architecture approximation. In *Advances in Neural Information Processing Systems*, 2019.

[12] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*, 2020.

[13] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. In *International Conference on Learning Representations*, 2020.

[14] Xiangning Chen and Cho-Jui Hsieh. Stabilizing differentiable architecture search via perturbation-based regularization. In *International Conference on Machine Learning*, 2020.

[15] Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. DrNAS: Dirichlet neural architecture search. In *International Conference on Learning Representations*, 2021.

[16] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *IEEE/CVF International Conference on Computer Vision*, 2019.

[17] Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Xinyu Xiao, and Jian Sun. Detnas: Backbone search for object detection. In *Advances in Neural Information Processing Systems*, 2019.

[18] Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. FairDARTS: Eliminating unfair advantages in differentiable architecture search. In *European Conference on Computer Vision*, 2020.

[19] Xiangxiang Chu, Xiaoxing Wang, Bo Zhang, Shun Lu, Xiaolin Wei, and Junchi Yan. Darts-: Robustly stepping out of performance collapse without indicators. In *International Conference on Learning Representations*, 2021.

[20] Cody Coleman, Christopher Yeh, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. Se-

lection via proxy: Efficient data selection for deep learning. In *International Conference on Learning Representations*, 2020.

[21] Raj Dabre, Chenhui Chu, and Anoop Kunchukuttan. A survey of multilingual neural machine translation. *ACM Computing Surveys*, 53(5):1–38, 2020.

[22] Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Peter Vajda, et al. Fbnetv3: Joint architecture-recipe search using predictor pretraining. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[23] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

[24] Mike Davies, Andreas Wild, Garrick Orchard, Yulia Sandamirskaya, Gabriel A Fonseca Guerra, Prasad Joshi, Philipp Plank, and Sumedh R Risbud. Advancing neuromorphic computing with loihi: A survey of results and outlook. *IEEE*, 109(5):911–934, 2021.

[25] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *The Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2019.

[26] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[27] Peter U Diehl and Matthew Cook. Unsupervised learning of digit recognition

using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, 9:99, 2015.

[28] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *International Joint Conference on Neural Networks*, 2015.

[29] Mingyu Ding, Xiaochen Lian, Linjie Yang, Peng Wang, Xiaojie Jin, Zhiwu Lu, and Ping Luo. Hr-nas: Searching efficient high-resolution neural architectures with lightweight transformers. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[30] Piotr Dollár, Mannat Singh, and Ross Girshick. Fast and accurate model scaling. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[31] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[32] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020.

[33] Xuanyi Dong, Lu Liu, Katarzyna Musial, and Bogdan Gabrys. NATS-Bench: Benchmarking nas algorithms for architecture topology and size. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2021.

[34] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer,

Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[35] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[36] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, 2018.

[37] Wei Fang, Zhaofei Yu, Yanqi Chen, Timothée Masquelier, Tiejun Huang, and Yonghong Tian. Incorporating learnable membrane time constant to enhance learning of spiking neural networks. *arXiv preprint arXiv:2007.05785*, 2020.

[38] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. Graph neural architecture search. In *International Joint Conference on Artificial Intelligence*, 2020.

[39] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[40] Rob Geada, Dennis Prangle, and Andrew Stephen McGough. Bonsai-net: One-shot neural architecture search via differentiable pruners. *arXiv preprint arXiv:2006.09264*, 2020.

[41] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge University Press, 2002.

[42] Xinyu Gong, Shiyu Chang, Yifan Jiang, and Zhangyang Wang. Autogan: Neural architecture search for generative adversarial networks. In *IEEE/CVF International Conference on Computer Vision*, 2019.

[43] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems*, 2014.

[44] Alex Graves, Marc G. Bellemare, Jacob Menick, Rémi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *International Conference on Machine Learning*, 2017.

[45] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, 2020.

[46] Julia Guerrero-Viu, Sven Hauns, Sergio Izquierdo, Guilherme Miotto, Simon Schrodi, Andre Biedenkapp, Thomas Elsken, Difan Deng, Marius Lindauer, and Frank Hutter. Bag of baselines for multi-objective joint neural architecture search and hyperparameter optimization. *arXiv preprint arXiv:2105.01015*, 2021.

[47] Jianyuan Guo, Kai Han, Yunhe Wang, Chao Zhang, Zhaohui Yang, Han Wu, Xinghao Chen, and Chang Xu. Hit-detector: Hierarchical trinity architecture search for object detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[48] 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 *European Conference on Computer Vision*, 2020.

[49] Bing Han, Gopalakrishnan Srinivasan, and Kaushik Roy. Rmp-snn: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, June 2020.

[50] Kai Han, Yunhe Wang, Qiulin Zhang, Wei Zhang, Chunjing Xu, and Tong Zhang. Model rubik's cube: Twisting resolution, depth and width for tinynets. In *Advances in Neural Information Processing Systems*, 2020.

[51] Yunzhe Hao, Xuhui Huang, Meng Dong, and Bo Xu. A biologically plausible supervised learning method for spiking neural networks using the symmetric stdp rule. *Neural Networks*, 121:387–395, 2020.

[52] Chaoyang He, Haishan Ye, Li Shen, and Tong Zhang. Milenas: Efficient neural architecture search via mixed-level reformulation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[53] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2016.

[54] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *IEEE International Conference on Computer Vision*, 2017.

[55] Weihua He, YuJie Wu, Lei Deng, Guoqi Li, Haoyu Wang, Yang Tian, Wei Ding, Wenhui Wang, and Yuan Xie. Comparing snns and rnns on neuromorphic vision datasets: similarities and differences. *Neural Networks*, 132:108–120, 2020.

[56] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, 2020.

[57] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[58] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[59] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017.

[60] Martin Jankowiak and Fritz Obermeyer. Pathwise derivatives beyond the reparameterization trick. In *International Conference on Machine Learning*, 2018.

[61] Chenhan Jiang, Hang Xu, Wei Zhang, Xiaodan Liang, and Zhenguo Li. Sp-nas: Serial-to-parallel backbone search for object detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[62] Yingyezhe Jin, Wenrui Zhang, and Peng Li. Hybrid macro/micro level backpropagation for training deep spiking neural networks. In *Advances in Neural Information Processing Systems*, 2018.

[63] Jacques Kaiser, Hesham Mostafa, and Emre Neftci. Synaptic plasticity dynamics for deep continuous local learning (decolle). *Frontiers in Neuroscience*, 14: 424, 2020.

[64] Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, Simon J Thorpe, and Timothée Masquelier. Stdp-based spiking deep convolutional neural networks for object recognition. *Neural Networks*, 99:56–67, 2018.

[65] Jaehyun Kim, Heesu Kim, Subin Huh, Jinho Lee, and Kiyoung Choi. Deep neural networks with weighted spikes. *Neurocomputing*, 311:373–386, 2018.

[66] Jihwan Kim, Jisung Wang, Sangki Kim, and Yeha Lee. Evolved speech-transformer: Applying neural architecture search to end-to-end automatic speech recognition. In *INTERSPEECH*, 2020.

[67] Jinseok Kim, Kyungsu Kim, and Jae-Joon Kim. Unifying activation-and timing-based learning rules for spiking neural networks. In *Advances in Neural Information Processing Systems*, 2020.

[68] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[69] Seijoon Kim, Seongsik Park, Byunggook Na, and Sungroh Yoon. Spiking-yolo: Spiking neural network for energy-efficient object detection. In *AAAI Conference on Artificial Intelligence*, 2020.

[70] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[71] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

[72] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. In *Artificial Intelligence and Statistics*, 2017.

[73] Nikita Klyuchnikov, Ilya Trofimov, Ekaterina Artemova, Mikhail Salnikov, Maxim Fedorov, and Evgeny Burnaev. Nas-bench-nlp: neural architec-

ture search benchmark for natural language processing. *arXiv preprint arXiv:2006.07116*, 2020.

[74] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11): 1238–1274, 2013.

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

[76] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 2012.

[77] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *IEEE*, 86(11):2278–2324, 1998.

[78] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[79] Chankyu Lee, Syed Shakib Sarwar, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. Enabling spike-based backpropagation for training deep neural network architectures. *Frontiers in Neuroscience*, 14:119, 2020.

[80] Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, and Yingyan Lin. Hw-nas-bench: Hardware-

aware neural architecture search benchmark. In *International Conference on Learning Representations*, 2021.

[81] Guohao Li, Guocheng Qian, Itzel C Delgadillo, Matthias Muller, Ali Thabet, and Bernard Ghanem. Sgas: Sequential greedy architecture search. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[82] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, 2018.

[83] Hongmin Li, Hanchao Liu, Xiangyang Ji, Guoqi Li, and Luping Shi. Cifar10-dvs: an event-stream dataset for object classification. *Frontiers in Neuroscience*, 11:309, 2017.

[84] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*, 2020.

[85] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. In *International Conference on Learning Representations*, 2014.

[86] Ming Lin, Pichao Wang, Zhenhong Sun, Hesen Chen, Xiuyu Sun, Qi Qian, Hao Li, and Rong Jin. Zen-nas: A zero-shot nas for high-performance image recognition. In *IEEE/CVF International Conference on Computer Vision*, 2021.

[87] 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 *European Conference on Computer Vision*, 2018.

[88] Guilin Liu, Fitsum A Reda, Kevin J Shih, Ting-Chun Wang, Andrew Tao, and

Bryan Catanzaro. Image inpainting for irregular holes using partial convolutions. In *European Conference on Computer Vision*, 2018.

[89] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

[90] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European Conference on Computer Vision*, 2016.

[91] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

[92] Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. Deep photo style transfer. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[93] Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997.

[94] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9, 2008.

[95] Abhinav Mehrotra, Alberto Gil CP Ramos, Sourav Bhattacharya, Łukasz Dudziak, Ravichander Vipperla, Thomas Chau, Mohamed S Abdelfattah, Samin Ishtiaq, and Nicholas Donald Lane. Nas-bench-asr: Reproducible neural architecture search for speech recognition. In *International Conference on Learning Representations*, 2021.

[96] Joe Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. Neural architecture search without training. In *International Conference on Machine Learning*, 2021.

[97] Paul Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[98] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT Press, 2012.

[99] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36 (6):51–63, 2019.

[100] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NeurIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[101] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *IEEE International Conference on Computer Vision*, 2015.

[102] Seongsik Park, Seijoon Kim, Hyeokjun Choe, and Sungroh Yoon. Fast and efficient information transmission with burst spikes in deep spiking neural networks. In *ACM/IEEE Design Automation Conference (DAC)*, 2019.

[103] Seongsik Park, Seijoon Kim, Byunggook Na, and Sungroh Yoon. T2fsnn: deep spiking neural networks with time-to-first-spike coding. In *ACM/IEEE Design Automation Conference (DAC)*, 2020.

[104] Houwen Peng, Hao Du, Hongyuan Yu, Qi Li, Jing Liao, and Jianlong Fu. Cream of the crop: Distilling prioritized paths for one-shot neural architecture search. In *Advances in Neural Information Processing Systems*, 2020.

[105] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning*, 2018.

[106] Ilija Radosavovic, Justin Johnson, Saining Xie, Wan-Yen Lo, and Piotr Dollár. On network design spaces for visual recognition. In *IEEE/CVF International Conference on Computer Vision*, 2019.

[107] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, 2017.

[108] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence*, 2019.

[109] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[110] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. Conversion of continuous-valued deep networks to efficient

event-driven networks for image classification. *Frontiers in Neuroscience*, 11: 682, 2017.

[111] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[112] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[113] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[114] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems*, 2015.

[115] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations*, 2018.

[116] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in Neuroscience*, 13:95, 2019.

[117] B. Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[118] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[119] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[120] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[121] Sen Song, Kenneth D Miller, and Larry F Abbott. Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nature Neuroscience*, 3(9):919–926, 2000.

[122] Xiu Su, Tao Huang, Yanxi Li, Shan You, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. Prioritized architecture sampling with monto-carlo tree search. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[123] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2015.

[124] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[125] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[126] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, 2019.

[127] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[128] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. An empirical study of example forgetting during deep neural network learning. In *International Conference on Learning Representations*, 2019.

[129] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[130] Ziyu Wan, Bo Zhang, Dongdong Chen, Pan Zhang, Dong Chen, Jing Liao, and Fang Wen. Bringing old photos back to life. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[131] Ruochen Wang, Minhao Cheng, Xiangning Chen, Xiaocheng Tang, and Cho-Jui Hsieh. Rethinking architecture selection in differentiable nas. In *International Conference on Learning Representations*, 2021.

[132] Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, Hanrui Wang, Yujun Lin, and Song Han. Apq: Joint search for network architecture, pruning and quantization policy. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[133] Xiaoxing Wang, Chao Xue, Junchi Yan, Xiaokang Yang, Yonggang Hu, and Kewei Sun. Mergenas: Merge operations into one for differentiable architecture search. In *International Joint Conference on Artificial Intelligence*, 2020.

[134] Yaoming Wang, Wenrui Dai, Chenglin Li, Junni Zou, and Hongkai Xiong. Sivdnas: Semi-implicit variational dropout for hierarchical one-shot neural architecture search. In *International Joint Conference on Artificial Intelligence*, 2020.

[135] Colin White, Arber Zela, Binxin Ru, Yang Liu, and Frank Hutter. How powerful are performance predictors in neural architecture search? *arXiv preprint arXiv:2104.01177*, 2021.

[136] 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 *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[137] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in Neuroscience*, 12:331, 2018.

[138] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, Yuan Xie, and Luping Shi. Direct

training for spiking neural networks: Faster, larger, better. In *AAAI Conference on Artificial Intelligence*, 2019.

[139] Lingxi Xie, Xin Chen, Kaifeng Bi, Longhui Wei, Yuhui Xu, Zhengsu Chen, Lanfei Wang, An Xiao, Jianlong Chang, Xiaopeng Zhang, and Qi Tian. Weight-sharing neural architecture search: A battle to shrink the optimization gap. *arXiv preprint arXiv:1808.05377*, 2020.

[140] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: Stochastic neural architecture search. In *International Conference on Learning Representations*, 2019.

[141] Jingjing Xu, Liang Zhao, Junyang Lin, Rundong Gao, Xu Sun, and Hongxia Yang. Knas: Green neural architecture search. In *International Conference on Machine Learning*, 2021.

[142] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2020.

[143] Bin Yan, Houwen Peng, Kan Wu, Dong Wang, Jianlong Fu, and Huchuan Lu. Lighttrack: Finding lightweight neural networks for object tracking via one-shot architecture search. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[144] Yibo Yang, Shan You, Hongyang Li, Fei Wang, Chen Qian, and Zhouchen Lin. Towards improving the consistency, efficiency, and flexibility of differentiable neural architecture search. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[145] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, 2019.

[146] Shan You, Tao Huang, Mingmin Yang, Fei Wang, Chen Qian, and Changshui Zhang. Greedynas: Towards fast one-shot nas with greedy supernet. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[147] Tong Yu and Hong Zhu. Hyper-parameter optimization: A review of algorithms and applications. *arXiv preprint arXiv:2003.05689*, 2020.

[148] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*, 2020.

[149] Arber Zela, Julien Siems, and Frank Hutter. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. In *International Conference on Learning Representations*, 2020.

[150] Miao Zhang, Huiqi Li, Shirui Pan, Xiaojun Chang, and Steven Su. Overcoming multi-model forgetting in one-shot nas with diversity maximization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[151] Miao Zhang, Huiqi Li, Shirui Pan, Taoping Liu, and Steven W Su. One-shot neural architecture search via novelty driven sampling. In *International Joint Conference on Artificial Intelligence*, 2020.

[152] Shifeng Zhang, Longyin Wen, Xiao Bian, Zhen Lei, and Stan Z Li. Single-shot refinement neural network for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[153] Wenrui Zhang and Peng Li. Temporal spike sequence learning via backpropagation for deep spiking neural networks. In *Advances in Neural Information Processing Systems*, 2020.

[154] Xiong Zhang, Hongmin Xu, Hong Mo, Jianchao Tan, Cheng Yang, Lei Wang, and Wenqi Ren. Dcnas: Densely connected neural architecture search for semantic image segmentation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[155] Xuanyang Zhang, Pengfei Hou, Xiangyu Zhang, and Jian Sun. Neural architecture search with random labels. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[156] Hanle Zheng, Yujie Wu, Lei Deng, Yifan Hu, and Guoqi Li. Going deeper with directly-trained larger spiking neural networks. In *AAAI Conference on Artificial Intelligence*, 2021.

[157] Dongzhan Zhou, Xinchi Zhou, Wenwei Zhang, Chen Change Loy, Shuai Yi, Xuesen Zhang, and Wanli Ouyang. Econas: Finding proxies for economical neural architecture search. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[158] Pan Zhou, Caiming Xiong, Richard Socher, and Steven Hoi. Theory-inspired path-regularized differential network architecture search. In *Advances in Neural Information Processing Systems*, 2020.

[159] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

[160] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning

transferable architectures for scalable image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

초 록

딥러닝과 심층 신경망은 인공지능 시스템을 실현하기 위한 가장 대중적인 선택지 중 하나가 되었다. 고도화된 딥러닝 기술을 다양한 분야에 적용하기 위해서는 이런 딥러닝 전문가의 양성 뿐만 아니라 비전문가들이 높은 수준의 성능을 지닌 심층 신경망을 확보할 수 있도록 하는 기술이 필요하다. 자동화 기계 학습에 기반한 방식으로써, 어떤 분야에서 준수한 성능을 내는 심층 신경망을 자동으로 얻을 수 있는 신경망 구조 탐색 기술이 심층 신경망 수동 설계의 실용적인 대안으로 큰 관심을 받고 있다. 정확한 탐색을 위해서는 굉장히 많은 탐색 비용이 필요하나, 가중치 공유 방법을 통해 심층 신경망의 성능을 추정할 수 있게 되어 탐색 비용이 크게 절감되었다. 가중치 공유 방법을 기반으로 다양한 신경망 구조 탐색 방법론이 제안되었고, 여러 분야에서 사람이 만든 심층 신경망의 성능을 뛰어넘는 결과를 만들어 내고 있다. 본 학위논문은 신경망 구조 탐색의 중요한 연구 주제인 어떻게 탐색 비용을 절감하는 지, 어떻게 탐색 공간과 탐색 알고리즘을 개선하여 탐색 성능을 향상시킬 지, 신경망 구조적 중요성이 간과되고 있는 분야에 신경망 구조 탐색을 어떻게 적용하는 지에 대한 방법론과 연구 결과물을 포함하고 있다.

첫번째 연구는 탐색 성능 저하 없이 신경망 구조 탐색 방법론들의 탐색 비용을 절감하는 것이다. 신경망 구조 탐색을 활용하고 싶은 연구자들에게 신경망 구조 탐색의 상당한 탐색 비용은 장애물이 된다. 이 문제를 해결하고자, 우리는 신경망 구조 탐색 가속화를 위해 주어진 데이터셋의 대표적 데이터로 이루어진 부분 집합인 프록시 데이터를 도입한다. 데이터 선택 방식은 다양한 딥러닝 분야에서 활용되어 왔지만, 신경망 구조 탐색 벤치마크 상에서의 기존 데이터 선택 방법들에 대한 평가를 통해 이들이 신경망 구조 탐색에 적합하지 않다는 것과 새로운 데이터 선택 방법이 필요하다라는 것을 보였다. 이 선택 방법들로 만든 프록시 데이터를 데이터

엔트로피를 통해 분석한 결과를 토대로, 우리는 신경망 구조 탐색에 적합한 새로운 프록시 데이터 선택 방법을 제안한다. 실효성을 보이기 위해 다양한 데이터셋과 탐색 공간들, 신경망 구조 탐색 방법들을 교차하여 철저히 실험을 하였다. 그 결과, 제안한 데이터 선택 방법을 적용한 신경망 구조 탐색 알고리즘은 전체 데이터셋을 사용했을 때 얻은 신경망의 성능에 비견하는 성능을 가진 신경망을 발견하였다. 탐색 비용을 크게 절감하였으며, 제안한 방법이 적용된 DARTS는 단일 GPU를 사용했을 때 CIFAR-10 상에서는 40분, ImageNet 상에서는 7.5시간을 소요하였다. 또한, 기존 방식과는 정반대로, 우리의 ImageNet 프록시 데이터 상에서 찾은 구조를 더 작은 데이터셋인 CIFAR-10에서 전이 평가했을 시, 최신 성능에 가까운 2.4% 테스트 오차율을 보였다.

두번째 연구에서 우리는 셀 기반 탐색 공간을 개량함으로써 이 탐색 공간 상에서 동작하는 미분가능한 신경망 구조 탐색의 성능을 개선한다. 슈퍼 신경망 셀 상에 적용된 연속적 완화를 통해 미분가능한 신경망 구조 탐색은 탐색 과정동안 다양한 셀 구조를 근사적으로 평가한다. 최적화된 슈퍼 신경망 셀로부터 셀 구조를 도출할 때 사용되는 최댓값을 기준으로 한 선택 규칙으로 인하여, 기존 셀 기반 탐색 공간에 포함되지 않으나 높은 성능을 낼 수 있는 많은 셀들이 불가피하게 전혀 선택되지 못한다. 이러한 기존 셀 도출 방식의 한계를 극복하기 위해, 우리는 탐색 공간을 확장시키고 이 확장된 탐색 공간에 존재하는 셀 구조를 도출할 수 있도록 상위-k 생존 규칙을 제안한다. 한편, 상위-k 생존 규칙이 적용된 기존 신경망 탐색 방법들에서는 큰 탐색 성능 저하가 일어나는 것을 관측하였다. 우리는 이 확장된 탐색 공간을 올바르게 탐색하기 위한 방법으로써, 베타 분포를 따르는 연속적 확률 변수로 후보 연산기들의 강도를 모델링하하는 BtNAS를 제안한다. BtNAS는 변분 베이즈 방법을 이용하여 연산기 강도의 참 사후 분포를 근사하며, 변분 베타 분포의 파라미터는 기울기 기반 최적화 방법과 편도함수 추정 기법을 통해 학습된다. 결과적으로 미분가능한 신경망 구조 탐색 방법들이 최근 발표한 셀 구조들에 비견되는 성능을 보이면서 더 적은 파라미터를 지닌 신경망을 생성하는 셀을 도

출하였다. 또한, 슈퍼 신경망 내의 모든 중간 노드가 활성화되도록 연산기 강도를 조정하였으며, 이를 통해 CIFAR-10 상에서 2.3%의 최신 성능의 테스트 오차율을 달성하였다.

마지막 연구는 에너지 효율적인 스파이킹 신경망을 찾기 위한 신경망 구조 탐색을 활용하는 방안에 관한 것이다. 뉴런과 시냅스로 이루어진 뇌 내의 정보 전달 방식을 모방한 스파이킹 신경망은 큰 관심을 받아왔다. 스파이킹 신경망은 이산적이고 드물게 발생하는 스파이크에 의한 이벤트 기반 연산 방식을 통해 공간적, 시간적 정보를 효율적으로 처리할 수 있다. 대부분의 이전 연구들은 스파이킹 신경망의 성능과 에너지 효율성을 개선하기 위한 학습 방법론에 집중하였으며, 스파이킹 신경망에 관한 구조적 효과에 대해서는 거의 연구되지 않았다. 우리는 스파이킹 신경망을 위한 신경망 구조 탐색을 활용하기 위해, AutoSNN 이라는 스파이크를 고려한 신경망 구조 탐색 프레임워크를 제안한다. 먼저 기존 심층 신경망보다 스파이킹 신경망에 적합한 구조적 요소가 무엇인지 알아낸 뒤, 스파이킹 신경망을 위한 두 단계 탐색 공간을 구축한다. 그 뒤, 탐색 공간 내의 후보 구조들의 성능과 스파이크 생성을 추정하도록 가중치 공유 기반 원샷 슈퍼 신경망 학습 방식과 진화 탐색 알고리즘을 채택한다. 구조 평가에 사용되는 구조 적합도는 탐색 과정에서 정확도와 스파이크 수를 고려하도록 고안한다. AutoSNN이 찾은 스파이킹 신경망은 정확도와 에너지 효율성 측면에서 사람이 만든 스파이킹 신경망을 능가하였다. 또한, 우리는 뉴로모픽 데이터셋을 포함한 다양한 데이터셋에서 AutoSNN의 우월성을 보였다.

본 학위논문을 통해 중요한 세 연구 주제들에 따라 우리는 효율적이며 효과적인 신경망 구조 탐색 프레임워크를 개발하기 위한 방법들을 제안하고 신경망 구조 탐색을 이용하는 전략을 보인다. 제안한 방법들을 검증하기 위한 상당량의 실험적 결과를 제시하여, 딥러닝의 성능을 최대화하기 위해서는 구조 선택이 고려되어야함을 보였다. 따라서 본 학위논문에서 소개된 접근 방식들을 통해 신경망 구조 탐색이 적극 기용되고 다양한 분야의 연구자들이 신경망 구조 탐색과 함께

딥러닝을 이용하여 유망한 결과를 쉽게 얻을 수 있을 것이라 예상한다.

주요어: 심층 신경망, 신경망 구조 탐색, 데이터 샘플링, 셀 기반 신경망, 스파이킹 신경망

학번: 2014-21622