AutoSNN: Towards Energy-Efficient Spiking Neural Networks

Byunggook Na [1] [†] **Jisoo Mok** [2] **Seongsik Park** [3] **Dongjin Lee** [2] **Hyeokjun Choe** [2] **Sungroh Yoon** [2] [4]

Abstract

Spiking neural networks (SNNs) that mimic information transmission in the brain can energy-efficiently process spatio-temporal information through discrete and sparse spikes, thereby receiving considerable attention. To improve accuracy and energy efficiency of SNNs, most previous studies have focused solely on training methods, and the effect of architecture has rarely been studied. We investigate the design choices used in the previous studies in terms of the accuracy and number of spikes and figure out that they are not best-suited for SNNs. To further improve the accuracy and reduce the spikes generated by SNNs, we propose a spike-aware neural architecture search framework called AutoSNN. We define a search space consisting of architectures without undesirable design choices. To enable the spike-aware architecture search, we introduce a fitness that considers both the accuracy and number of spikes. AutoSNN successfully searches for SNN architectures that outperform hand-crafted SNNs in accuracy and energy efficiency. We thoroughly demonstrate the effectiveness of AutoSNN on various datasets including neuromorphic datasets.

1. Introduction

Spiking neural networks (SNNs) are the next generation of neural networks inspired by the brain's information processing systems (Maass, 1997). The neurons in SNNs asynchronously transmit information through sparse and binary spikes, enabling event-driven computing. Unlike conventional neural networks being executed on GPUs, energy

[†]Portions of this research were done while the author was a Ph.D. student in SNU. [1]Samsung Advanced Institute of Technology, South Korea [2]Department of Electric and Computer Engineering, Seoul National University, South Korea [3]Korea Institute of Science and Technology, South Korea [4]Interdisciplinary Program in Artificial Intelligence, Seoul National University, South Korea. Correspondence to: Sungroh Yoon <sryoon@snu.ac.kr>.
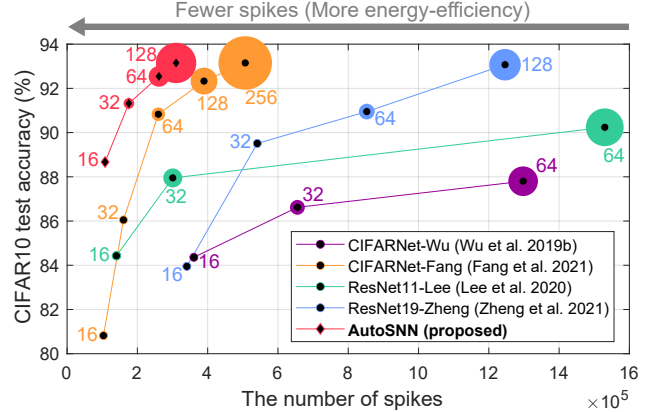
Figure 1. The number of spikes and CIFAR10 accuracy of various SNN architectures. Black circle- and diamond-shaped markers denote hand-crafted and automatically-designed SNN architectures, respectively. The size of a colored circle is proportionate to the model size and the number next to the circle indicates the initial channel of each SNN architecture. **AutoSNN discovers an energy-efficient SNN architecture that outperforms hand-crafted SNNs in terms of the accuracy and number of spikes.**

consumption occurs if a spike is generated on neuromorphic chips, which support neuromorphic computing. Hence, SNNs can significantly improve the energy efficiency of artificial intelligence systems. Most neuromorphic chips adopt network-on-chip architectures with neuromorphic cores, and SNNs are mapped to these multiple cores (Davies et al., 2018; Merolla et al., 2014). A large number of spikes cause spike congestion between the cores, thereby considerably increasing the communication overhead and energy consumption (Davies et al., 2021).Therefore, when realizing SNNs on neuromorphic chips, their energy efficiency, that is, the number of generated spikes, must be considered along with accuracy.

As a means of improving the performance of SNNs and relieving their energy consumption, previous studies only focused on training algorithms and protocols, such as reducing the timesteps required in SNNs (Neftci et al., 2019; Wu et al., 2019b; Fang et al., 2021b; Lee et al., 2020; He et al., 2020; Kaiser et al., 2020; Zheng et al., 2021). They employed conventional architectures typically used as artificial neural networks (ANNs), such as VGGNet (Simonyan &

Zisserman, 2015) and ResNet (He et al., 2016). Even though architecture modifications such as spike-element-wise residual blocks have been proposed (Kim & Panda, 2020; 2021; Fang et al., 2021a), the architectural consideration about which architecture is suitable for use as SNNs in terms of the number of generated spikes has been overlooked. As observed in Figure 1, the number of generated spikes differs significantly depending on the SNN architecture. It is necessary to investigate the design choices that affect the accuracy and spike generation.

In this study, we analyze architectural properties in terms of the accuracy and number of spikes, and identify preferable design choices for energy-efficient SNNs with minimal spikes. The use of the global average pooling layer (Lin et al., 2014) and employing layers with trainable parameters for down-sampling (He et al., 2016; Sandler et al., 2018; Liu et al., 2019) decreases the energy efficiency of SNNs, suggesting the exclusion of these design choices from architectures. To further improve the performance and energy efficiency, we adopt neural architecture search (NAS) that has emerged as an attractive alternative to hand-crafting ANN architectures. Across various applications (Xie et al., 2020; Chen et al., 2019; Jiang et al., 2020; Guo et al., 2020a; Kim et al., 2020a; Ding et al., 2021; Yan et al., 2021; Zhang et al., 2021b), NAS successfully searched for ANN architectures best-suited for the target objectives. Inspired by the success of NAS, we propose a spike-aware NAS framework, named *AutoSNN*, to design energy-efficient SNNs.

For AutoSNN, we define a search space considering both accuracy and energy efficiency and propose a spike-aware search algorithm. 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. To explore the proposed search space with a reasonable search cost, we exploit the one-shot weight-sharing approach of the NAS (Bender et al., 2018; Cai et al., 2020; Guo et al., 2020b; Li & Talwalkar, 2020; Zhang et al., 2021a; Yan et al., 2021). To estimate the accuracy and number of spikes of candidate SNN architectures, a super-network that encodes all the architectures is trained through a direct training method for SNNs. Once the super-network is trained, AutoSNN executes an evolutionary search algorithm that is proposed to find the SNN with the highest evaluation metric, which we call fitness. To enable a spike-aware search, we define new fitness that reflects the accuracy and number of spikes.

AutoSNN discovers desirable SNN architectures that outperform hand-crafted SNNs in terms of both the accuracy and number of spikes, as shown in Figure 1. The superiority of the searched SNNs is consistently observed across various datasets including neuromorphic datasets. Additionally, when our search algorithm is executed on a search

space consisting of ANNs that share the proposed macro architecture, the resulting architectures with spiking blocks experience performance deterioration, emphasizing the importance of searching in the SNN search space to consider properties of SNNs. The code of AutoSNN is available at https://github.com/nabk89/AutoSNN. Our contributions are summarized as follows:

- To the best our knowledge, this is the first study to thoroughly investigate the effect of architectural components on performance and energy efficiency of SNNs.

- We propose a spike-aware NAS framework, named AutoSNN, and discover SNNs that outperform handcrafted SNNs designed without consideration of the structures suitable for SNNs.

- We demonstrate the effectiveness of AutoSNN through substantial results and evaluations on various datasets.

2. Background and Related Work

2.1. Spiking Neural Networks

A spiking neuron in SNNs integrates synaptic inputs from the previous layer into an internal state called the membrane potential. When the membrane potential integrated over time exceeds a certain threshold value, the neuron fires a spike to the next layer, and thus the spiking neurons transmit information through binary spike trains. Among several spiking neuron models, the leaky integrate-and-fire (LIF) neuron is simple yet widely used due to its effectiveness (Gerstner & Kistler, 2002). The dynamics of the LIF neuron at timestep t are as follows:

$$\tau_{\text{decay}} \frac{\partial V_{\text{mem}}(t)}{\partial t} = -(V_{\text{mem}}(t) - V_{\text{reset}}) + z(t), \quad (1)$$

where V_{mem} is the membrane potential of a neuron and V_{reset} is the value of V_{mem} after spike firing. τ_{decay} is a membrane time constant that controls the decay of V_{mem} and z is the presynaptic inputs. For effective simulations with GPUs, Eq. 1 is re-written with an iterative and discrete-time form as follows:

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

$$H^l[t] = V_{\text{mem}}^l[t-1] +$$
$$\frac{1}{\tau_{\text{decay}}}(-(V_{\text{mem}}^l[t-1] - V_{\text{reset}}) + z^l[t]), \quad (2)$$

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

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

where superscript l indicates the layer index. The value of membrane potential at timestep t is divided into two states,

which represent the values before and after the trigger of a spike, denoted by $H[t]$ and $V_{\text{mem}}[t]$, respectively. $\phi[t]$ is a binary spike at t, w is the synaptic weight, and b is the bias. The value of $\phi[t]$ 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. In this study, we employed parametric LIF (PLIF) neurons (Fang et al., 2021b) that was proposed to improve the performance of SNNs. In PLIF neurons, $\frac{1}{\tau_{\text{decay}}}$ is replaced with a sigmoid function $1/(1 + exp(-\alpha))$ with a trainable parameter α.

To obtain a high-performing SNN, researchers have proposed various training methods that can be clustered into several approaches. In the ANN-to-SNN conversion approach (Diehl et al., 2015; Rueckauer et al., 2017; Sengupta et al., 2019; Park et al., 2019; 2020; Han et al., 2020; Kim et al., 2020b), after training an ANN, the optimized 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 ANNs. Even though these converted SNNs achieve accuracies comparable to those of ANNs, they heavily rely on the performance of trained ANNs and require a significant number of timesteps, which leads to significant spike generation and energy inefficiency.

As an approach to directly optimize SNNs, unsupervised learning methods based on the spike-timing-dependent plasticity method (Diehl & Cook, 2015) were introduced but were restricted to shallow networks and yielded limited performance. Another approach is supervised learning based on a backpropagation algorithm (Bohte et al., 2002). A surrogate gradient function was used for backpropagation to approximate the gradients in the non-differentiable spiking activities (Neftci et al., 2019; Wu et al., 2019b; Lee et al., 2020; He et al., 2020; Kaiser et al., 2020); the detailed explanation is provided in Section E. In recent studies, supervised learning has been effective for deep SNNs and yielded high accuracies with few timesteps and sparse generation of spikes (Fang et al., 2021b; Zheng et al., 2021). Therefore, we adopt a supervised learning method (Fang et al., 2021b) to obtain energy-efficient SNNs.

2.2. Neural Architecture Search

Early NAS methods (Real et al., 2017; 2019; Zoph & Le, 2017; Zoph et al., 2018) sampled and separately evaluated candidate architectures, all of which had to be trained from scratch until convergence. To reduce the enormous search cost induced by such approach, recent NAS methods have adopted the weight-sharing strategy (Pham et al., 2018). With weight-sharing, search space $\mathcal{A}$, a set of all candidate architectures, is encoded in the form of a super-network $\mathcal{S}(W)$, whose weights W are shared across all subnetworks.

Depending on how weight-sharing is incorporated into the search algorithm, NAS methods are primarily categorized into differentiable and one-shot methods. The family of differentiable NAS methods (Cai et al., 2019; Liu et al., 2019; Wu et al., 2019a; Dong & Yang, 2019) starts by constructing a continuous search space that spans the entire search space of discrete architectures. They introduced trainable architecture parameters a into a super-network. During the training of the super-network, W and a are optimized alternately, and once the training process is complete, an architecture with the largest a is selected.

Unlike differentiable NAS, one-shot weight-sharing NAS (Bender et al., 2018; Cai et al., 2020; You et al., 2020; Guo et al., 2020b; Zhang et al., 2020; Li & Talwalkar, 2020; Peng et al., 2020; Zhang et al., 2021a; Yan et al., 2021) disentangles the search process into two procedures: super-network training and architecture evaluation. During super-network training, architectures are sampled, and thus all candidate architectures in $\mathcal{A}$ can be approximately trained. Given the trained super-network $\mathcal{S}(W^*)$, architectures inherit weights from $\mathcal{S}(W^*)$ and are evaluated without additional training. Various sampling strategies for super-network training have been proposed to improve the reliability of the evaluation process, such as greedy path filtering (You et al., 2020), novelty-based sampling (Zhang et al., 2020), and prioritized path distillation (Peng et al., 2020). Because one-shot weight-sharing NAS does not have to retrain the super-network every time it needs to search for a new architecture (Cai et al., 2020), it is far more computationally efficient than differentiable NAS when deploying a diverse set of SNNs for different neuromorphic chips.

SNASNet (Kim et al., 2022), which is concurrent to AutoSNN, has been recently proposed to find a performative SNN using a NAS method without training. SNASNet only focuses on the accuracy of SNNs, while AutoSNN considers both their accuracy and energy efficiency.

3. Architectural Analysis for SNNs

The SNN architectures used in previous studies (Neftci et al., 2019; Wu et al., 2019b; Lee et al., 2020; He et al., 2020; Kaiser et al., 2020; Fang et al., 2021b; Zheng et al., 2021) originated from conventional ANNs, such as VGGNet-styled stacked convolutional layers and max pooling layers (Simonyan & Zisserman, 2015) and ResNet-styled stacked residual blocks with skip connections (He et al., 2016). These architectures were selected without considering the architectural suitability of the SNNs. This section analyzes architectural factors that affect the accuracy and spike generation of SNNs and investigates which design choices are desirable for energy-efficient SNNs.

We start by standardizing the building blocks used in previous SNN architectures (Wu et al., 2019b; Lee et al., 2020; Fang et al., 2021b; Zheng et al., 2021) into spiking convolu-

Table 1. Evaluation for different design choices on CIFAR10.

Architecture	GAP	Normal	Down-sample	Acc.(%)	Spikes
SNN_1	✗	SCB_k3	MaxPool	86.93	154K
SNN_2	✔	SCB_k3	MaxPool	85.05	168K
SNN_3	✗	SCB_k3	SCB_k3	87.94	222K
SNN_4	✗	SCB_k3	AvgPool	79.59	293K

tion block (SCB) and spiking residual block (SRB), which stem from VGGNet and ResNet, respectively. As depicted in Figure 8, both SCB and SRB consist of two convolutional layers with spiking neurons, and the SRB additionally includes a skip connection. In the SNN research field, spiking blocks with a kernel size of 3 have been widely used.

Using these spiking blocks, we discuss two architectural aspects: 1) A global average pooling (GAP) layer with spiking neurons is not suitable for SNNs, and 2) max pooling layers are best-suited for down-sampling in SNNs. We prepare four architectures, denoted by SNN_{1, 2, 3, 4} and depicted in Figure 7, consisting of to-be-determined (TBD) blocks that can be filled with spiking blocks. Based on the SNN architecture (Fang et al., 2021b), which yields the Pareto frontier curve in Figure 1 among previous studies, we construct SNN_1 and its variants, described in the following sections. In these architectures, normal blocks preserve the spatial resolution of the input feature map, and down-sampling (DS) blocks halve the spatial resolution. We employ a voting layer that receives spikes from the spiking neurons of a fully connected (FC) layer to produce robust classification results (Fang et al., 2021b). The voting layer is implemented by a 1D average pooling layer with a kernel size of K and a stride of K; in this study, we set $K = 10$, as in the previous study (Fang et al., 2021b). The experimental results of these architectures by filling the TBD blocks with SCB_k3 and SRB_k3 are provided in Table 1 and Table 8, respectively.

3.1. Use of Global Average Pooling Layer

To observe the effect of the GAP layer (Lin et al., 2014), we compare SNN_1 and SNN_2. SNN_2 includes a GAP layer with spiking neurons before the FC layer. The GAP layer is commonly used to reduce the number of parameters of the FC layer in ANNs (Szegedy et al., 2015; He et al., 2016; Sandler et al., 2018) and SNNs (Zheng et al., 2021). However, our results indicate that the GAP layer has a negative effect on both the accuracy and energy efficiency.

As presented in Table 1, the accuracy of SNN_2 is significantly lower than that of SNN_1 and even more spikes are generated. Figure 2A and B show the layerwise spike patterns in SNN_1 and SNN_2, where all the TBD blocks are SCB_k3. With the GAP layer, the number of spikes of
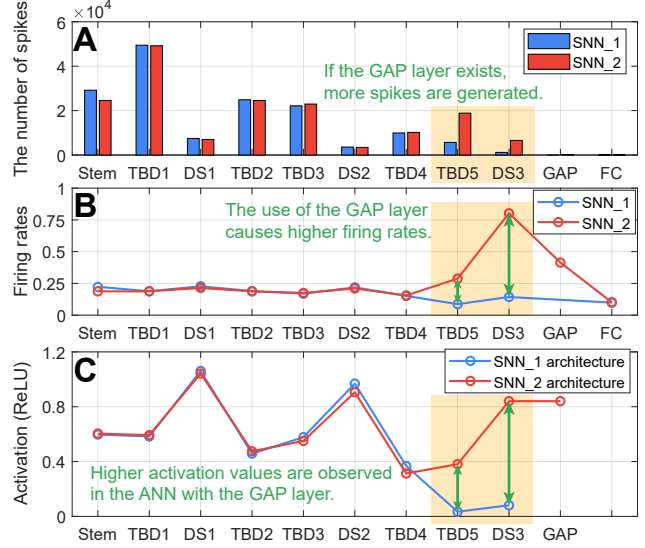

Figure 2. Layerwise patterns of architectures without and with the GAP layer, denoted by SNN_1 and SNN_2, respectively. (A, B) The number of spikes and firing rates averaged over test data for 8 timesteps. (C) The average activation values in feature map of architectures, which replace spiking neurons with ReLU activation functions, averaged over test data.

the last TBD block (TBD5) and the last max pooling layer (DS3) significantly increases. DS3 and the GAP layer have 4×4×4C and 1×1×4C 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 4×4×4C and 1×1×4C, 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-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., ANN architectures) and train them, analogous phenomena are observed. As shown in Figure 2C, 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, the firing rates in SNN_2 and the activation values in the corresponding ANN behave differently. When the 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. Consequently, as shown in Figure 2B, for SNN_2, the firing rates of the GAP layer are lower than those of DS3. This results in further information reduction, and thus the accuracy of SNN_2 is lower than that of SNN_1 (Table 1). In contrast, such information reduction does not occur in ANNs; the average activation values of the GAP layer do not differ from those of DS3, because by definition the GAP layer simply averages the activation values of DS3. This is empirically supported by the fact that the accuracies of the two ANNs corresponding to SNN_1 and SNN_2 were approximately 90.5%. Therefore, unlike ANNs, when SNNs employ the GAP layer, the number of spikes increases and the information reduction occurs, suggesting that the GAP layer is an unsuitable design choice for performative and energy-efficient SNNs.

3.2. Block Choice for Down-sampling

In previous studies, different types of layers have been employed as down-sampling layers: max pooling layer (Fang et al., 2021b), convolutional layer (Zheng et al., 2021), and average pooling layer (Wu et al., 2019b; Lee et al., 2020), which are used in SNN_1, SNN_3, and SNN_4, respectively[1]. Thus, we investigate which down-sampling layer can lead to energy-efficient SNNs. As shown in Table 1, SNN_3 with SCB_k3 achieves a 1% higher accuracy than SNN_1, 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%. SNN_4 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 SNN_4 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 SNN_1 and SNN_3. In Figure 3, the number of spikes in SNN_3 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 pool-

---

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

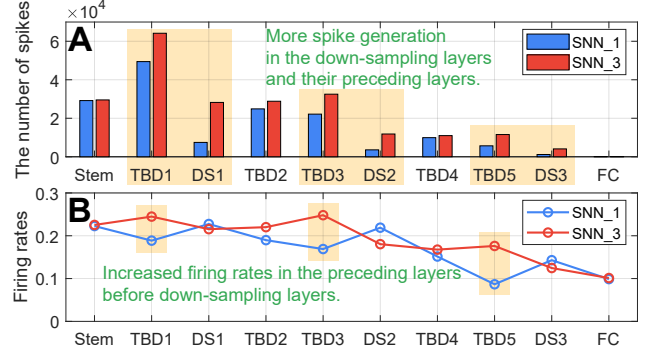*Figure 3.* Layerwise patterns of architectures that employ max pooling layers (SNN_1) and SCB_k3 (SNN_3) as down-sampling (DS) layers. (A) The number of spikes and (B) firing rates averaged over test data for 8 timesteps.

ing 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 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.

4. AutoSNN

In Section 3, we showed that excluding the GAP layer and using the max pooling layers effectively yields energy-efficient SNNs. To further increase the energy efficiency, we leverage NAS, which automatically designs an optimal architecture. We propose a spike-aware NAS framework, named AutoSNN, including both search space and search algorithm, which are described in Sections 4.1 and 4.2, respectively.

4.1. Energy-Efficient Search Space

It is important to define an expressive search space to effectively leverage NAS. To this end, we define a search space on two levels: a macro-level backbone architecture and a micro-level candidate block set. Based on the findings in Section 3, SNN_1 is exploited as the macro-level backbone architecture, abbreviated as the macro architecture. On the micro-level, we define a set of candidate spiking blocks derived from SCB and SRB with a kernel size k. With $k \in \{3, 5\}$, the set of candidate blocks consists of five blocks: skip connection (skip), SCB_k3, SCB_k5, SRB_k3 and SRB_k5. These blocks are employed for each TBD block in the macro architecture. With skip, search algorithms can choose to omit computation in certain TBD blocks. We additionally considered a spiking block motivated by MobileNet (Sandler et al., 2018), but this block was less suitable for designing energy-efficient SNNs than SCB
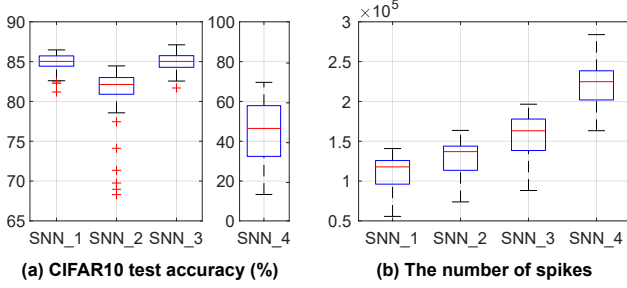
Figure 4. Search space quality comparison between SNN_1-based (proposed) and SNN_{2, 3, 4}-based search spaces.

and SRB; detailed discussion is provided in Section B.3.

Herein, we evaluate the quality of the proposed search space in terms of the accuracy and number of spikes. We compare four search spaces that consist of SNN architectures based on SNN_{1, 2, 3, 4} used in Section 3. For each search space, we generate 100 architectures by randomly choosing blocks from the predefined candidate blocks to fill TBD blocks. All the generated SNN architectures were trained on CIFAR10 for 120 epochs using a direct training method (Fang et al., 2021b).

In Figure 4, the search space quality comparison empirically validates that the proposed search space based on SNN_1 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 Section 3. The proposed search space includes architectures with higher accuracy and fewer spikes on average than those in the SNN_2-based search space. A similar but more distinctive pattern is observed in the SNN_4-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 SNN_3-based search space, the accuracy is comparable but the average number of spikes increases by approximately x1.4 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.

4.2. Spike-aware Search Algorithm

In general, according to Eq. 2 and Section E, SNNs with timesteps take longer to train than ANNs requiring feedforward and backpropagation once. Thus, reducing the search cost induced by training and evaluating candidate architectures becomes even more critical in NAS for SNNs than for ANNs. To address this challenge, we adopt a one-shot weight-sharing approach based on an evolutionary algorithm. AutoSNN consists of two consecutive procedures, which are illustrated in Figure 10: 1) training a super-network that encodes the proposed search space and 2) evaluating candidate architectures under a search bud-

get to search for an optimal architecture. In AutoSNN, the target dataset is divided into training data D_{train} and validation data D_{val}, which are used for the first and second procedures, respectively.

In the first procedure, AutoSNN trains the architectures sampled from a super-network whose subnetworks correspond to all the candidate architectures in the proposed search space. To evenly train spiking blocks in the super-network, we adopt the single-path uniform sampling (Bender et al., 2018; Li & Talwalkar, 2020; Guo et al., 2020b; Zhang et al., 2021a; Yan et al., 2021) because of its effectiveness and simplicity. Based on the supervised training method of SNN (Fang et al., 2021b), each sampled architecture is trained on a single mini-batch from D_{train}.

Given a trained super-network, to enable a spike-aware search, AutoSNN penalizes SNNs with more spikes. To evaluate SNN architectures, we define the architecture fitness $F(A)$ based on exponential discounting as follows:

$$F(A) = Accuracy \times (N/N_{\text{avg}})^{\lambda}, \quad (3)$$

where N is the number of spikes generated by architecture A, N_{avg} is the average number of spikes across architectures sampled during training the super-network, and λ is the coefficient that controls the influence of the spike-aware term. We set $\lambda < 0$ to explicitly search for architectures with lower N. As $|\lambda|$ increases, the SNNs discovered by AutoSNN are expected to generate fewer spikes. Note that other discounting functions are viable. Using a fitness function with linear discounting ($F'(A) = Acc. - \lambda'(N/N_{\text{avg}})$) led to similar results to those using our exponential discounting function.

Using the obtained fitness value of each candidate architecture, AutoSNN based on an evolutionary algorithm explores the proposed search space and finds the architecture with the highest fitness value. We briefly describe the evolutionary search algorithm; a detailed explanation is provided in Section C along with Algorithm 1. AutoSNN maintains two population pools throughout the search process: the top-k population pool P_{top} and the temporary evaluation population pool P_{eval}. First, P_{eval} is prepared by generating architectures using evolutionary techniques: mutation and crossover. For these techniques, the parent architectures are sampled from P_{top}. The architectures in P_{eval} are evaluated using the proposed spike-aware fitness and D_{val}. If the fitness values of the evaluated architectures are higher than those of the architectures in P_{top}, P_{top} is updated. These processes are repeated until B architectures are evaluated, where B is a search budget; in this study, we set B as 200. Finally, AutoSNN obtains architecture A^* with the highest fitness value from P_{top}.

Because the two procedures are decoupled, AutoSNN can discover a promising SNN architecture for every different

Table 2. Searching results with different λ on CIFAR10.

λ in fitness	0	-0.08	-0.16	-0.24
Acc (%) ↑	88.69	88.67	88.46	86.58
Spikes ↓	127K	108K	106K	54K

neuromorphic chip by simply changing λ. Similar to a previous study (Cai et al., 2020), AutoSNN can reuse a trained super-network and execute the second procedure alone. Unlike the differentiable NAS approach, where the entire search process is executed to find a single architecture, the additional search cost is negligible, as demonstrated in Section 5. Therefore, our search algorithm based on two separate procedures is a practical and effective method for obtaining energy-efficient SNNs.

5. Experiments and Discussion

5.1. Experimental Settings

We evaluated the SNNs searched by AutoSNN on two types of datasets: static datasets (CIFAR10, CIFAR100 (Krizhevsky et al., 2009), SVHN (Netzer et al., 2011), and Tiny-ImageNet-200[2]) and neuromorphic datasets (CIFAR10-DVS (Li et al., 2017) and DVS128-Gesture (Amir et al., 2017)). Details regarding these datasets are provided in Section A.2. The dataset is divided into 8:2 for D_{train} and D_{val}. We use the Adam optimizer (Kingma & Ba, 2015) with a learning rate of 0.001 and cutout data augmentation (DeVries & Taylor, 2017) to train the super-network and the searched SNNs for 600 epochs on a single NVIDIA 2080ti GPU. For all architectures, we use PLIF neurons (Fang et al., 2021b) with $V_{\text{th}} = 0$, $V_{\text{reset}} = 0$, 8 timesteps, and an initial τ of 2.

5.2. Searching with Different λ of Spike-aware Fitness

By varying $\lambda \in \{0, -0.08, -0.16, -0.24\}$, we execute AutoSNN and report the results in Table 2; the searched architectures are visualized in Figure 11. Note that on a single 2080ti GPU, the search cost is approximately 7 GPU hours: 6 h 48 min for training a super-network and 8 min for executing the evolutionary search. For each λ, it incurs a negligible additional cost (8 min).

In Figure 11, distinctive differences among the SNN architectures searched with different λ values are observed; detailed discussion is provided in Section D.1. Increasing $|\lambda|$ leads to architectures with more TBD blocks filled with Skip, thereby decreasing the number of spikes. This confirms that λ functions according to our intent to adjust the trade-off between the accuracy and number of spikes in the

---

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

Table 3. Evaluation of SNN architectures with an initial channel of C on CIFAR10. † denotes the initial channel used in previous studies. ‡ denotes reported values in original papers (i.e., not reproduced in our training settings).

SNN Architecture	C	Acc (%) ↑	Spikes ↓	Params (M)
CIFARNet-Wu	16	84.36	361K	0.71
(Wu et al., 2019b)	32	86.62	655K	2.83
	64	87.80	1298K	11.28
	†128	‡90.53	-	45.05
CIFARNet-Fang	16	80.82	104K	0.16
(Fang et al., 2021b)	32	86.05	160K	0.60
	64	90.83	260K	2.34
	128	92.33	290K	9.23
	†256	93.15	507K	36.72
ResNet11-Lee	16	84.43	140K	1.17
(Lee et al., 2020)	32	87.95	301K	4.60
	†64	‡90.24	‡1530K	18.30
ResNet19-Zheng	16	83.95	341K	0.23
(Zheng et al., 2021)	32	89.51	541K	0.93
	64	90.95	853K	3.68
	†128	93.07	1246K	14.69
AutoSNN	16	88.67	108K	0.42
(proposed)	32	91.32	176K	1.46
	64	92.54	261K	5.44
	128	93.15	310K	20.92

searched SNN. Compared to the architecture with $\lambda = 0$, the one searched with $\lambda = -0.08$ reduces approximately 20K spikes while achieving a similar accuracy. Thus, in other experiments, we use $\lambda = -0.08$ by default.

5.3. Comparison with Existing SNNs

When AutoSNN is executed, an initial channel of 16 is used for the macro architecture, but the hand-crafted SNNs used in previous studies had a noticeably larger number of initial channels denoted by † in Table 3. Hence, for a fair comparison, we increase and decrease the initial channel by a factor of two, for AutoSNN and the other SNNs, respectively. We train all SNNs under the same training settings. The evaluation results for CIFAR10 are provided in Table 3; refer to Figure 1 for the visual summary.

The experimental results confirm that AutoSNN successfully discovers an energy-efficient SNN, which lies on the Pareto front in the region of accuracy and the number of spikes. In Figure 5, 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.
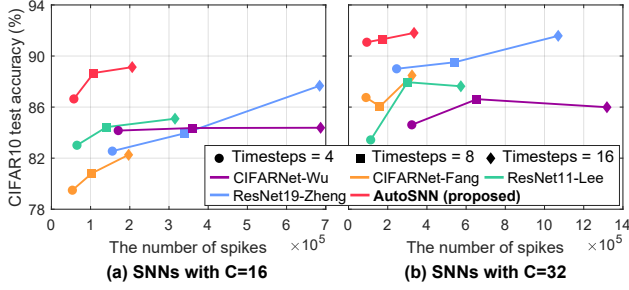
Figure 5. The number of spikes vs. CIFAR10 accuracy of various SNN architectures trained with different timesteps $\in \{4, 8, 16\}$.

Table 4. Evaluation of SNN architectures with the initial channel of C on various datasets. $\dagger$ denotes architectures with additional stem layers. $\ddagger$ denotes reported values in original papers.

Data	SNN Architecture	C	Acc (%) ↑	Spikes ↓
CIFAR100	Fang et al. 2021	256	66.83	716K
	AutoSNN	64	69.16	326K
SVHN	Fang et al. 2021	256	91.38	462K
	AutoSNN	64	91.74	215K
Tiny-Image Net-200	Fang et al. 2021	256	45.43	1724K
	AutoSNN$\dagger$	64	46.79	680K
CIFAR10 -DVS	Wu et al. 2019b	128	$\ddagger$60.50	-
	Fang et al. 2021	128	69.10	4521K
	Zheng et al. 2021	64	66.10	1550K
	AutoSNN$\dagger$	16	72.50	1269K
DVS128 -Gesture	He et al. 2020	64	$\ddagger$93.40	-
	Kaiser et al. 2020	64	$\ddagger$95.54	-
	Fang et al. 2021	128	95.49	1459K
	Zheng et al. 2021	64	96.53	1667K
	AutoSNN$\dagger$	16	96.53	423K

We further evaluate our searched SNN architecture by transferring it to various datasets. For the three static datasets, an initial channel is set to 64 to account for the increased complexity of these datasets. For datasets with a larger resolution than CIFAR10 (i.e., Tiny-ImageNet-200, CIFAR10-DVS, and DVS128-Gesture), we use macro architectures with deeper stem layers, which are visualized in Figure 6. Table 4 clearly reveals that AutoSNN achieves higher accuracy and generates fewer spikes than the other hand-crafted architectures across all datasets including both static and neuromorphic datasets.

5.4. Validity of the Search Algorithm in AutoSNN

Ablation Study for Two Procedures. Through an ablation study in Table 5, we inspect two procedures in the search algorithm of AutoSNN. First, we randomly sample and train 10 architectures (Random sampling); the cost is approximately 100 GPU hours. Second, using a trained super-network, we select the architecture with the highest

Table 5. Ablation study results of AutoSNN on CIFAR10. WS is a shorthand for weight-sharing.

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

Table 6. Evaluation for AutoSNN on enlarged search spaces with eight TBD blocks ($C = 16$).

SNN Architecture	Acc (%) ↑	Spikes ↓
Using TBD blocks instead of max pooling layers		
`SCB_k3` in all TBD blocks	87.94	222K
`SRB_k3` in all TBD blocks	89.18	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 TBD blocks	87.04	230K
`SRB_k3` in all TBD blocks	88.69	228K
AutoSNN ($\lambda = -0.08$)	88.60	143K
AutoSNN ($\lambda = -0.16$)	87.29	60K

spike-aware fitness among the 200 randomly-sampled architectures (WS + random search), which is the same search budget as AutoSNN. As shown in Table 5, SNNs searched by WS + random search yield higher accuracy than the average accuracy of 10 randomly-sampled architectures. This indicates that the weight-sharing strategy with a direct training method of SNNs is valid in the SNN domain. Applying evolutionary search further improves the search result, solidifying the effectiveness of our evolutionary search with spike-aware fitness.

Searching on Enlarged Search Spaces. AutoSNN is effective in searching for desirable architectures in enlarged search spaces. The proposed SNN search space consists of 3,125 architectures (5^5; five candidate blocks and five TBD blocks). We construct two search spaces, where the macro architectures have eight TBD blocks as described in Table 6; both of them include 390,625 architectures (5^8). One is the SNN_3-based search space, and the other consists of architectures where a TBD block is added before each max pooling layer of SNN_1. Table 6 shows the validity of the search algorithm of AutoSNN. In both search spaces, compared with architectures consisting of `SCB_k3` and `SRB_k3`, AutoSNN discovers architectures which generate significantly fewer spikes.

Table 7. Searching results on CIFAR10 from ANN and SNN search spaces, where λ in fitness is set as 0 for a fair comparison.

Search space	Acc (%) ↑	Spikes ↓
w/o spiking neurons (ANN)	88.02	134K
w/ spiking neurons (proposed)	88.69	127K

5.5. Architecture Search without Spiking Neurons

To validate the importance of considering the properties of SNNs during the search process, we execute our evolutionary search algorithm on an ANN search space, where spiking neurons in architectures are removed; instead, ReLU activation functions are used. Because spikes cannot be observed in the ANNs, the accuracy is solely used to evaluate the architectures. After finding an architecture from the ANN search space, spiking neurons are added to this architecture, which is then trained according to the settings in Section 5.1. As presented in Table 7, the SNN architecture searched by AutoSNN with $\lambda = 0$ achieves higher accuracy and generates fewer spikes than the architecture searched from the ANN search space. We conjecture that this discrepancy because training the super-network without spiking neurons cannot reflect the properties of SNNs such as spike-based neural dynamics to represent information.

6. Conclusion

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, a 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.

Acknowledgment

References

Amir, A., Taba, B., Berg, D., Melano, T., McKinstry, J., Di Nolfo, C., Nayak, T., Andreopoulos, A., Garreau, G., Mendoza, M., et al. A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Bender, G., Kindermans, P.-J., Zoph, B., Vasudevan, V., and Le, Q. Understanding and simplifying one-shot architecture search. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Bohte, S. M., Kok, J. N., and La Poutre, H. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48(1-4):17–37, 2002.

Cai, H., Zhu, L., and Han, S. Proxylessnas: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020.

Chen, Y., Yang, T., Zhang, X., Meng, G., Xiao, X., and Sun, J. Detnas: Backbone search for object detection. *Advances in Neural Information Processing Systems*, 2019.

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., Dimou, G., Joshi, P., Imam, N., Jain, S., et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

Davies, M., Wild, A., Orchard, G., Sandamirskaya, Y., Guerra, G. A. F., Joshi, P., Plank, P., and Risbud, S. R. Advancing neuromorphic computing with loihi: A survey of results and outlook. *Proceedings of the IEEE*, 109(5): 911–934, 2021.

DeVries, T. and Taylor, G. W. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

Diehl, P. U. and Cook, M. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, 9:99, 2015.

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., and Pfeiffer, M. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *International Joint Conference on Neural Networks*, 2015.

Ding, M., Lian, X., Yang, L., Wang, P., Jin, X., Lu, Z., and Luo, P. Hr-nas: Searching efficient high-resolution neural architectures with lightweight transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

Dong, X. and Yang, Y. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

Fang, W., Yu, Z., Chen, Y., Huang, T., Masquelier, T., and Tian, Y. Deep residual learning in spiking neural networks. *Advances in Neural Information Processing Systems*, 34, 2021a.

Fang, W., Yu, Z., Chen, Y., Masquelier, T., Huang, T., and Tian, Y. Incorporating learnable membrane time constant to enhance learning of spiking neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2661–2671, 2021b.

Gerstner, W. and Kistler, W. M. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

Guo, J., Han, K., Wang, Y., Zhang, C., Yang, Z., Wu, H., Chen, X., and Xu, C. Hit-detector: Hierarchical trinity architecture search for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020a.

Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., and Sun, J. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision*, 2020b.

Han, B., Srinivasan, G., and Roy, K. Rmp-snn: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, June 2020.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

He, W., Wu, Y., Deng, L., Li, G., Wang, H., Tian, Y., Ding, W., Wang, W., and Xie, Y. Comparing snns and rnns on neuromorphic vision datasets: similarities and differences. *Neural Networks*, 132:108–120, 2020.

Jiang, C., Xu, H., Zhang, W., Liang, X., and Li, Z. Sp-nas: Serial-to-parallel backbone search for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

Kaiser, J., Mostafa, H., and Neftci, E. Synaptic plasticity dynamics for deep continuous local learning (decolle). *Frontiers in Neuroscience*, 14:424, 2020.

Kim, J., Wang, J., Kim, S., and Lee, Y. Evolved speech-transformer: Applying neural architecture search to end-to-end automatic speech recognition. In *INTERSPEECH*, 2020a.

Kim, S., Park, S., Na, B., and Yoon, S. Spiking-yolo: Spiking neural network for energy-efficient object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020b.

Kim, Y. and Panda, P. Revisiting batch normalization for training low-latency deep spiking neural networks from scratch. *Frontiers in Neuroscience*, pp. 1638, 2020.

Kim, Y. and Panda, P. Optimizing deeper spiking neural networks for dynamic vision sensing. *Neural Networks*, 144:686–698, 2021.

Kim, Y., Li, Y., Park, H., Venkatesha, Y., and Panda, P. Neural architecture search for spiking neural networks. *arXiv preprint arXiv:2201.10355*, 2022.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

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

Lee, C., Sarwar, S. S., Panda, P., Srinivasan, G., and Roy, K. Enabling spike-based backpropagation for training deep neural network architectures. *Frontiers in Neuroscience*, 14:119, 2020.

Li, H., Liu, H., Ji, X., Li, G., and Shi, L. Cifar10-dvs: an event-stream dataset for object classification. *Frontiers in Neuroscience*, 11:309, 2017.

Li, L. and Talwalkar, A. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*, 2020.

Lin, M., Chen, Q., and Yan, S. Network in network. In *International Conference on Learning Representations*, 2014.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

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

Merolla, P., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., Jackson, B. L., Imam, N., Guo, C., Nakamura, Y., et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

Neftci, E. O., Mostafa, H., and Zenke, F. 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.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. In *NeurIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

Park, S., Kim, S., Choe, H., and Yoon, S. Fast and efficient information transmission with burst spikes in deep spiking neural networks. In *ACM/IEEE Design Automation Conference (DAC)*, 2019.

Park, S., Kim, S., Na, B., and Yoon, S. T2fsnn: deep spiking neural networks with time-to-first-spike coding. In *ACM/IEEE Design Automation Conference (DAC)*, 2020.

Pellegrini, T., Zimmer, R., and Masquelier, T. Low-activity supervised convolutional spiking neural networks applied to speech commands recognition. In *IEEE Spoken Language Technology Workshop (SLT)*, pp. 97–103. IEEE, 2021.

Peng, H., Du, H., Yu, H., Li, Q., Liao, J., and Fu, J. Cream of the crop: Distilling prioritized paths for one-shot neural architecture search. In *Advances in Neural Information Processing Systems*, 2020.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., and Kurakin, A. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., and Liu, S.-C. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, 11:682, 2017.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3): 211–252, 2015.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in Neuroscience*, 13:95, 2019.

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

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019a.

Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., and Shi, L. Direct training for spiking neural networks: Faster, larger, better. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019b.

Xie, L., Chen, X., Bi, K., Wei, L., Xu, Y., Chen, Z., Wang, L., Xiao, A., Chang, J., Zhang, X., and Tian, Q. Weight-sharing neural architecture search: A battle to shrink the optimization gap, 2020.

Yan, B., Peng, H., Wu, K., Wang, D., Fu, J., and Lu, H. Lighttrack: Finding lightweight neural networks for object tracking via one-shot architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

You, S., Huang, T., Yang, M., Wang, F., Qian, C., and Zhang, C. Greedynas: Towards fast one-shot nas with greedy supernet. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

Zhang, M., Li, H., Pan, S., Liu, T., and Su, S. W. One-shot neural architecture search via novelty driven sampling. In *International Joint Conference on Artificial Intelligence*, 2020.

Zhang, X., Hou, P., Zhang, X., and Sun, J. Neural architecture search with random labels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021a.

Zhang, X., Xu, H., Mo, H., Tan, J., Yang, C., Wang, L., and Ren, W. Dcnas: Densely connected neural architecture search for semantic image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021b.

Zheng, H., Wu, Y., Deng, L., Hu, Y., and Li, G. Going deeper with directly-trained larger spiking neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

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

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2018.

A. Experimental Environment

A.1. Experimental Settings

We implemented AutoSNN and all the experiments using SpikingJelly[3], and have included the codes in the supplementary materials (code.zip). The code will be made public on github after the review process.

For all the experiments, we used a single GeForce RTX 2080 Ti GPU and a supervised training method that utilizes PLIF neurons with an initial τ of 2 (i.e., $\alpha = -ln(1)$), $V_{\text{reset}} = 0$, and $V_{\text{th}} = 1$ (Fang et al., 2021b). We evaluated SNN architectures on two types of datasets: static datasets (CIFAR-10 (Krizhevsky et al., 2009), CIFAR-100 (Krizhevsky et al., 2009), SVHN (Netzer et al., 2011), and Tiny-ImageNet-200[4]) and neuronmorphic datasets (CIFAR10-DVS (Li et al., 2017) and DVS128-Gesture (Amir et al., 2017)). Details regarding these datasets are provided in Section A.2. We set timesteps as 8 and 20 for the static datasets and the neuromorphic datasets, respectively. When executing AutoSNN to search for SNNs on the proposed search space, CIFAR10 was used. The training data of CIFAR10 were divided into 8:2 for D_{train} and D_{val}, which were used to train the super-network and evaluate candidate architectures during the spike-aware evolutionary search, respectively.

A.1.1. TRAINING AN SNN-BASED SUPER-NETWORK

To train the super-network, we employed the Adam optimizer (Kingma & Ba, 2015) 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.

A.1.2. SPIKE-AWARE EVOLUTIONARY SEARCH

Once the super-network is trained, AutoSNN can evaluate candidate architectures that are sampled from the search space or generated by mutation and crossover. The architectures inherit the weights of the trained super-network. We set the search algorithm parameters as follows: a maximum round $\mathcal{T}$ of 10, a mutation probability ρ of 0.2, the number of architectures generated by mutation p_m of 10, the number of architectures generated by crossover p_c of 10, the size of the top-k architecture pool k of 10, and the size of the evaluation pool p of 20.

---

[3]https://github.com/fangwei123456/spikingjelly
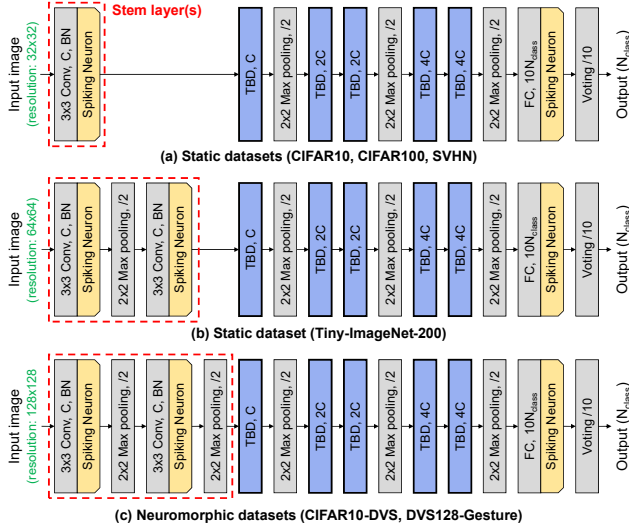[4]https://www.kaggle.com/akash2sharma/tiny-imagenet

Figure 6. 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.

A.1.3. TRAINING SNN ARCHITECTURES

All the SNNs including architectures searched by AutoSNN and conventional hand-crafted 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 (Kingma & Ba, 2015) 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 (DeVries & Taylor, 2017), 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 to reduce the resolution from 64×64 to 32×32, as shown in Figure 6(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 6(c).

A.2. Dataset Description

A.2.1. 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 (Russakovsky et al., 2015) and downsized from 224x224 to 64×64. 100K images and 2.5K images are used for training and testing, respectively. These datasets 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.

A.2.2. 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 (Li et al., 2017), 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 (Amir et al., 2017) includes 1,342 training data and 288 test data with 11 classes of hand gestures.

B. Supplemental Material of Section 3

B.1. Architecture Preparation for Analysis

In Section 3, we analyzed the architectural effects on accuracy and the number of spikes of SNNs. For the analysis, motivated by architectures used in the previous studies (Lee et al., 2020; Fang et al., 2021b; Zheng et al., 2021), we prepared architecture variations, i.e., SNN_{1, 2, 3, 4}, which are depicted in Figure 7. SNN_1 is a base architecture, SNN_2 includes the GAP layer, and SNN_3 and SNN_4 use trainable spiking blocks and average pooling layers as down-sampling layers, respectively.

B.2. Filling TBD Blocks with `SRB_k3`

Table 8 presents additional results of the architectures, where all TBD blocks are solely filled with SRB_k3. As discussed in Section 3 with Table 1, the effect of the architecture components is consistently observed. The use of the global average pooling layer and employing trainable spiking blocks or average pooling layers for down-sampling decreases the energy efficiency of SNNs, suggesting to exclude these design choices from SNN architectures.
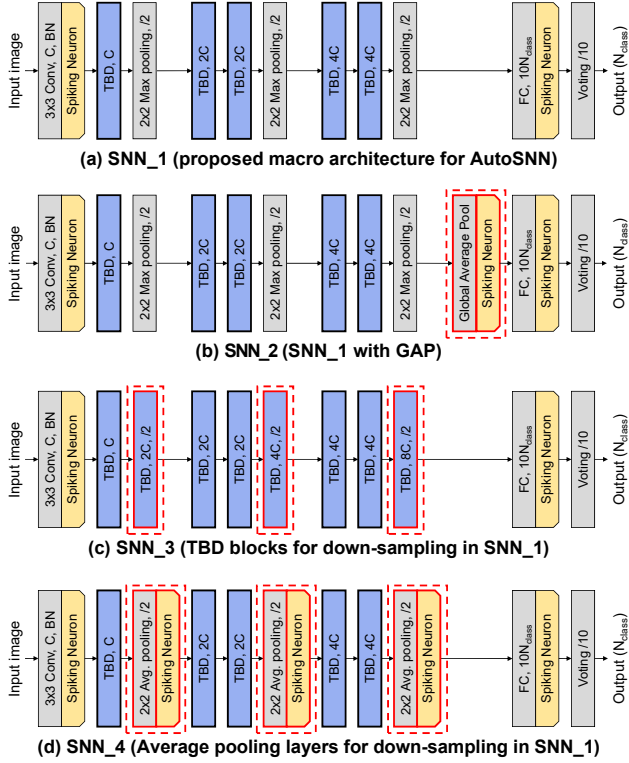
Figure 7. Architectures for analyzing the architectural effects: (a) SNN_1 and (b-d) its variants, SNN_{2, 3, 4}. The red dotted boxes indicate the change from SNN_1.

Table 8. Evaluation for different design choices on CIFAR10.

Architecture	GAP	Normal	Down-sample	Acc.(%)	Spikes
SNN_1	✗	SRB_k3	MaxPool	87.54	146K
SNN_2	✔	SRB_k3	MaxPool	85.82	168K
SNN_3	✗	SRB_k3	SRB_k3	89.18	221K
SNN_4	✗	SRB_k3	AvgPool	83.79	291K

B.3. Exploration for Energy-Efficient Candidate Blocks

In this study, we investigated which spiking blocks are suitable for designing energy-efficient SNNs. In previous studies, building blocks based on SCB and SRB have been mainly used. We newly introduced spiking inverted bottleneck block (SIB), inspired by the inverted bottleneck structure of MobileNetV2 (Sandler et al., 2018). Three trainable spiking blocks that we standardized are depicted in Figure 8. The inverted bottleneck structure can reduce the number of parameters and FLOPs in ANNs and is thus widely used to search for ANNs suitable to mobile devices (Cai et al., 2019; Wu et al., 2019a; Tan et al., 2019; Cai et al., 2020). The hardware affinity of the inverted bottleneck structure can be in line with designing SNN architectures that are realized on neuromorphic chips. The design choices for SIB include kernel size k and expansion ratio e; for simplicity,
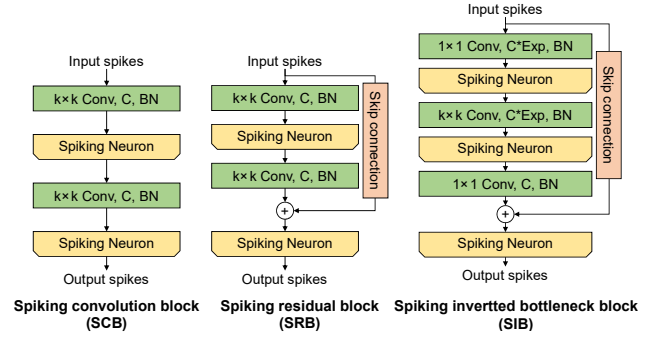

Figure 8. Spiking blocks, where a convolution layer, channels, and batch normalization are denoted by Conv, C, and BN, respectively.

Table 9. Evaluation for SNN_1-based architectures, which consist of a single type of spiking block.

Spiking block in SNN_1	Acc. (%)	Spikes	Firing rates
SCB_k3	86.93	154K	0.18
SRB_k3	87.54	146K	0.17
SIB_k3_e1	81.07	243K	0.23
SIB_k3_e3	88.45	374K	0.17

we denote SIB with $k = 3$ and $e = 3$ as SIB_k3_e3.

Three trainable spiking blocks, i.e., SCB, SRB, and SIB, are evaluated on SNN_1. Table 9 and Figure 9 provide the results when TBD blocks in SNN_1 are assigned a single type of block with a kernel size of 3. The SIB is significantly less desirable for energy-efficient SNNs than the 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. Because the firing rate of each TBD block is similar across all three blocks (the second plot in Figure 9), the difference in the number of spikes can be associated with the change in the number of spiking neurons.

The number of neurons of each block is theoretically as follows: $2hwc$ for SCB_k3 (which is identical for SRB_k3), $hw(2c_{in} + c_{out})$ for SIB_k3_e1, and $hw(6c_{in} + c_{out})$ for SIB_k3_e3, where hw is a resolution of its input feature map, and c_{in} and c_{out} are the number of channels of its input and output feature maps, respectively. Using the above equations, we obtain the number of spiking neurons of SNN_1 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 of the SNN. By applying the total firing rates (0.18, 0.23, and 0.17, respectively), we can calculate the approximated number of spikes as follows: $1.17HWC$, $1.84HWC$, and $2.81HWC$, respectively. Hence, we can
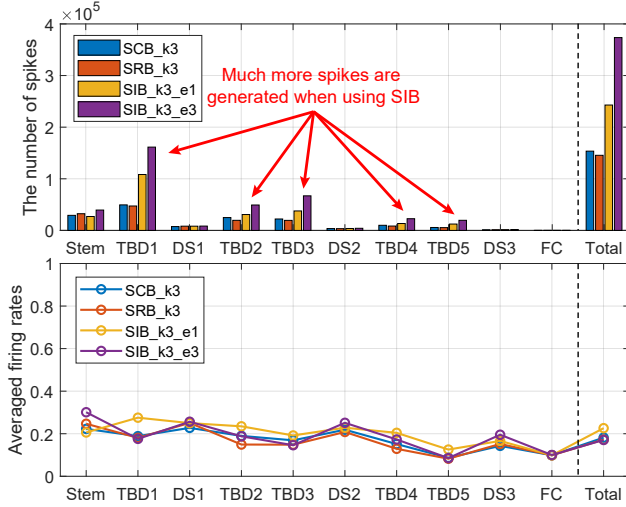
Figure 9. The number of spikes generated by architectures whose TBD blocks are assigned by a single type of spiking block, and their firing rates averaged over test data and 8 timesteps.

estimate that SNNs with SIB_k3_e1 and SIB_k3_e3 generate approximately 1.6x and 2.4x more spikes than that with SCB_k3. It is confirmed that the empirical results in Table 9 and our theoretical calculation are consistent. Therefore, to obtain energy-efficient SNNs, it is undesirable to include SIBs that have a large number of spiking neurons.

Applying spike regularization Using a method to regularize the spike generation (Pellegrini et al., 2021), we further evaluate spiking blocks with the following spike regularization term: $\frac{\lambda_{\mathrm{reg}}}{KT} \sum_t \sum_k \phi_k[t]$, where ϕ is a spike, K is the number of neurons, and T is the timesteps. This term is added to the training loss, and λ_{reg} controls the regularization strength. The results are provided in Table 10. Even with spike regularization, SIB is a less desirable choice than SCB and SRB in terms of both accuracy and spikes.

Table 10. Combining Table 9 with a regularization technique.

Spiking block	$\lambda_{\mathrm{reg}} = 1$		$\lambda_{\mathrm{reg}} = 0.1$		$\lambda_{\mathrm{reg}} = 0.01$		$\lambda_{\mathrm{reg}} = 0$	
	Acc.	Spikes	Acc.	Spikes	Acc.	Spikes	Acc.	Spikes
SCB_k3	64.36	83K	79.09	84K	86.39	124K	86.93	154K
SRB_k3	72.76	49K	83.25	70K	86.59	109K	87.54	146K
SIB_k3_e1	56.61	89K	73.54	119K	81.05	155K	81.07	243K
SIB_k3_e3	74.71	136K	84.59	186K	87.61	249K	88.45	374K

C. Search Algorithm Implementation

AutoSNN consists of two separate procedures, as illustrated in Figure 10: super-network training with a direct training method of SNNs and an evolutionary search algorithm with a spike-aware fitness. Using the proposed spike-aware fitness, AutoSNN finds the architecture with the highest fitness value through an evolutionary search algorithm. Through-
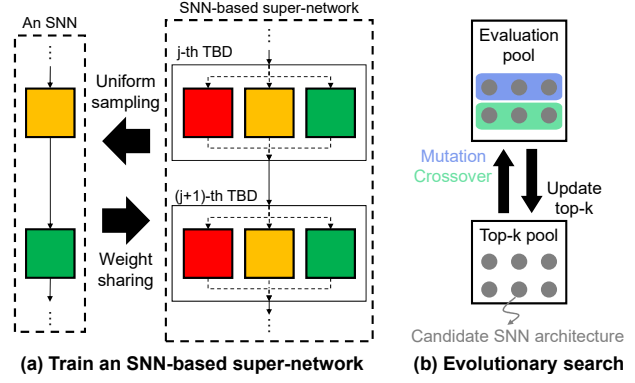

Figure 10. Two consecutive searching procedures of AutoSNN. The colored blocks in (a) represent candidate spiking blocks.

out the search process, depicted by Figure 10(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 as follows.

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 $p = 20$.

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

1: $P_{\text{eval}} = P_{\text{top}} = \emptyset$
2: **for** r = 1 : $\mathcal{T}$ **do**
3:    **if** r == 1 **then**
4:       $P_{\text{eval}} = \text{RandomSample}(p)$
5:    **else**
6:       $P_1 = \text{Mutation}(P_{\text{top}}, p_m, \rho)$
7:       $P_2 = \text{Crossover}(P_{\text{top}}, p_c)$
8:       $P_{\text{eval}} = P_1 \cup P_2$
9:       **if** $\text{Size}(P_{\text{eval}}) < p$ **then**
10:         $P_3 = \text{RandomSample}(p - \text{Size}(P_{\text{eval}}))$
11:         $P_{\text{eval}} = P_{\text{eval}} \cup P_3$
12:       **end if**
13:    **end if**
14:    fitness values = $\text{Evaluate}(S(W^*), D_{\text{val}}, P_{\text{eval}}, \lambda)$
15:    $P_{\text{top}} = \text{UpdateTop}k(P_{\text{top}}, P_{\text{eval}}, \text{fitness values})$
16: **end for**
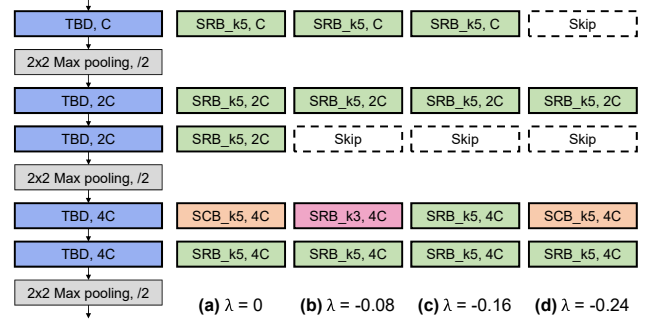17: **return** Top-1 SNN architecture A^* in P_{top}

Figure 11. SNN architectures searched by AutoSNN with different λ on the proposed search space.
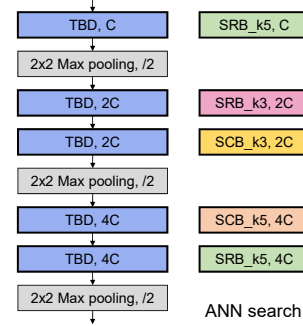

Figure 12. An architecture which is searched from the ANN search space. The candidate blocks filling TBD blocks in the architecture are inverted into the corresponding spiking blocks.

D. Supplemental Results of AutoSNN

D.1. SNN Architectures Searched with Different λ

SNN architectures searched by AutoSNN with different λ of the fitness value are visualized in Figure 11. For $\lambda = 0$, indicating that only accuracy is considered during the search process, 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 block in the architectures is filled with Skip. As shown in Table 2, 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 $\lambda = -0.24$, both the first and third TBD blocks are filled with Skip. The spikes generated by this architecture are much fewer than the other searched 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.

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.

D.2. Architecture Search without Spiking Neurons

In Section 5.5, we validate the importance of including spiking neurons in the search space. Figure 12 shows the architecture which is searched by our search algorithm on the search space without spiking neurons and then transformed into the SNN by adding spiking neurons to every selected blocks.

D.3. Comparison with Recent SNN Techniques

Here, in Table 11, we compare AutoSNN with BNTT (Kim & Panda, 2020), SALT (Kim & Panda, 2021), and SNAS-Net (Kim et al., 2022) as fairly as possible using their reported results. In BNTT and SALT, architectures were manually modified; BNTT considers time variance of batch normalization (BN) layers in VGG9, and VGG16_SBN used in SALT has a single BN layer before the last FC layer. As described in Section 2, SNASNet is concurrent to our work in terms of utilizing NAS; but, SNASNet is based on totally different approach from AutoSNN, because SNASNet employed a NAS method that does not require any training process. Compared to SALT, AutoSNN yielded higher accuracy and fewer spikes with less timesteps. BNTT reduced the number of spikes but only at the cost of accuracy. BNTT also uses a larger channel number and more timesteps

Table 11. Comparison with Recent SNN Techniques: vs. manually modified SNNs (the first row group) and vs. SNNs searched by a NAS approach (the second row group)

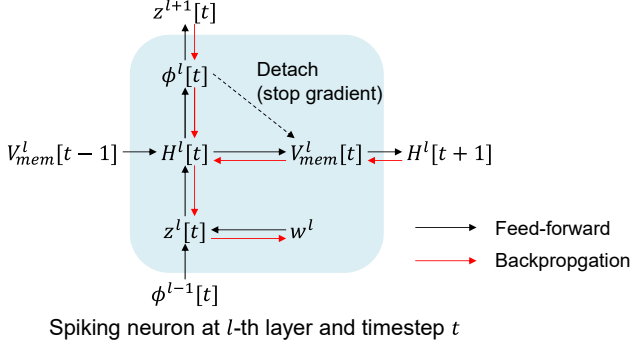| Method	Acc.(%)	Spikes	Timesteps
VGG9+BNTT ($C = 128$)	90.5	131K	25
VGG16_SBN+SALT ($C = 64$)	87.1	1500K	40
AutoSNN ($C = 16$)	88.7	108K	8
AutoSNN ($C = 32$)	91.3	176K	8
SNASNet-Fw ($C = 256$)	93.6	-	8
SNASNet-Bw ($C = 256$)	94.1	-	8
AutoSNN ($C = 128$)	93.2	310K	8

Figure 13. Dynamics of a spiking neuron: feed-forward and back-propagation flows.

than AutoSNN, both of which result in more energy consumption and longer latency. Two SNASNet architectures with $C = 256$ brought upon some increase in accuracy, but based on the results from the main paper, it is fair to expect AutoSNN with $C = 256$ will also achieve comparable accuracy.

E. Direct Training Framework for SNNs

In this section, we explain a direct training framework based on the supervised learning approach. Assuming a classification task with C classes, a loss function L is defined as:

$$L(\mathbf{o}, \mathbf{y}) = L(\frac{1}{T} \sum_{t=0}^{T-1} \phi^{\text{FC}}[t], \mathbf{y}), \quad (4)$$

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. In this study, we use the mean squared error (MSE) for L:

$$L = MSE(\mathbf{o}, \mathbf{y}) = \frac{1}{C} \sum_{i=0}^{C-1} (\frac{1}{T} \sum_{t=0}^{T-1} \phi_i^{\text{FC}}[t] - y_i)^2, \quad (5)$$

where C is the number of classes.

As shown in Figure 13, 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. 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 13. w^l is shared across T timesteps, and thus the derivative with respect to w^l can be obtained according to the chain rule as follows:

$$\frac{\partial L}{\partial w^l} = \sum_{t=0}^{T-1} \frac{\partial L}{\partial H^l[t]} \frac{\partial H^l[t]}{\partial z^l[t]} \frac{\partial z^l[t]}{\partial w^l}. \quad (6)$$

From Eq 2, the second and third terms in the right-hand side of Eq. 6 can be induced as follows:

$$\frac{\partial z^l[t]}{\partial w^l} = \phi^{l-1}[t], \qquad \frac{\partial H^l[t]}{\partial z^l[t]} = \frac{1}{\tau_{\text{decay}}}. \quad (7)$$

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

$$\frac{\partial L}{\partial H^l[t]} = \frac{\partial L}{\partial \phi^l[t]} \frac{\partial \phi^l[t]}{\partial H^l[t]} + \frac{\partial L}{\partial V_{\text{mem}}^l[t]} \frac{\partial V_{\text{mem}}^l[t]}{\partial H^l[t]}. \quad (8)$$

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:

$$\frac{\partial L}{\partial \phi^l[t]} = \frac{\partial L}{\partial z^{l+1}[t]} \frac{\partial z^{l+1}[t]}{\partial \phi^l[t]} = \frac{\partial L}{\partial z^{l+1}[t]} w^{l+1}. \quad (9)$$

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 $\phi^l[t]$ in Eq. 2:

$$\frac{\partial \phi^l[t]}{\partial H^l[t]} = \Theta'(H^l[t] - V_{\text{th}}). \quad (10)$$

The derivatives in the temporal domain of Eq. 8 are also induced from Eq. 2 as follows:

$$\frac{\partial L}{\partial V_{\text{mem}}^l[t]} = \frac{\partial L}{\partial H^l[t+1]} \frac{\partial H^l[t+1]}{\partial V_{\text{mem}}^l[t]} \quad (11)$$

$$= \frac{\partial L}{\partial H^l[t+1]} (1 - \frac{1}{\tau_{\text{decay}}}), \quad (12)$$

$$\frac{\partial V_{\text{mem}}^l[t]}{\partial H^l[t]} = 1 - \phi^l[t] - H^l[t] \frac{\partial \phi^l[t]}{\partial H^l[t]}. \quad (13)$$