[Supplementary Materials] Neural Architecture Search for Spiking Neural Networks

Youngeun Kim, Yuhang Li, Hyoungseob Park, Yeshwanth Venkatesha, and Priyadarshini Panda

Department of Electrical Engineering
Yale University
New Haven, CT, USA
{youngeun.kim, yuhang.li, hyoungseob.park, yeshwanth.venkatesha, priya.panda}@yale.edu

A. Code Implementation

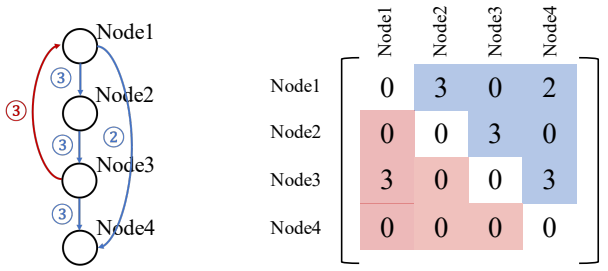
Code is available at Github URL. In Algorithm 1, we provide the pseudo code for the overall flow of our search algorithm.

Cell representation. Like NAS-Bench101 [13], we represent a cell with a $N \times N$ matrix where N is the number of nodes. We use such matrix representation in our code implementation. Fig. 1 shows a 4-node example. Each element of the matrix represents the type of operations (shown in Fig. 5 of the main manuscript). A cell always has a forward connection between Node 1 and Node 4 (the type of operation is searched by NAS algorithm) for promoting the fast spike propagation with a small number of timesteps.

Algorithm 1 main.py

```
# search optimal architecture with given N samples
def find_best_neuroncell(N, trainset):
    dict_net_score = {}
    for i in range(N):
        candidateNet = RandomNetGeneration()
        score = candidateNet(trainset)
        dict_net_score[score] = candidateNet
    searchedNet = SelectMaxScoreNet(dict_net_score)
    return searchedNet

# training the founded model
for (batch, labels) in enumerate(train_loader):
    outputs = searchedNet(batch)
    loss = CrossEntropyLoss(outputs, labels)
    loss.backward()
    optimizer.step()
```



(a) Searched Architecture (b) Corresponding Connection Matrix

Fig. 1. An example of cell representation.

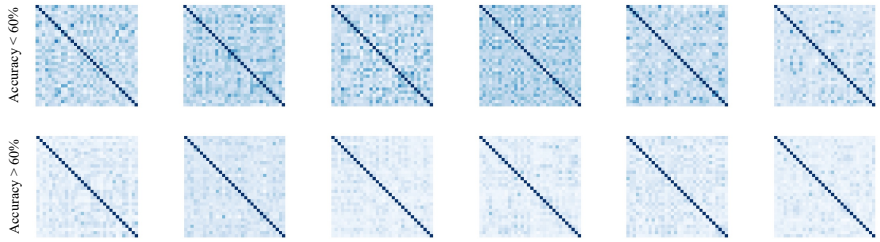


Fig. 2. Matrix Visualization (Eq. 4) of randomly sampled architectures on a CIFAR100 dataset. We use 64 mini-batch samples. **Top row**: Architectures achieves under 60% accuracy. **Bottom row**: Architectures achieves over 60% accuracy.

B. Dataset Details

In this section, we provide the details of datasets used in our experiments. **CIFAR10** [6] consists of 50,000 training samples / 10,000 testing samples with 10 categories. All images are RGB color images whose size are 32×32. **CIFAR100** has the same setting as CIFAR10, except it contains images from 100 categories. **TinyImageNet** is the modified subset of the original ImageNet dataset. Here, there are 200 different classes of ImageNet dataset [1], with 100,000 training and 10,000 validation 64×64 images.

C. Training Details with Surrogate Gradients

Based on the Cross-Entropy loss L from the prediction of the searched architecture, we calculate the gradients of each layer l based on the spikes activities. We accumulate the gradients in both spatial and time axis, which is called a spatio-temporal back-propagation (STBP) [12,9]. By chain rule, we formulate

the gradients of the weight parameters W_l at the layer l as:

$$\frac{\partial L}{\partial W_l} = \begin{cases} \sum_t (\frac{\partial L}{\partial O_l^t}\frac{\partial O_l^t}{\partial U_l^t} + \frac{\partial L}{\partial U_l^{t+1}}\frac{\partial U_l^{t+1}}{\partial U_l^t})\frac{\partial U_l^t}{\partial W_l}, & \text{if } l = \text{hidden} \\ \frac{\partial L}{\partial U_l^T}\frac{\partial U_l^T}{\partial W_l}. & \text{if } l = \text{output} \end{cases} \tag{1}$$

Here, O_l^t and U_l^t are output spikes and membrane potential at time-step t for layer l, respectively. Since we accumulate the spikes at the last layer, we can get a continuous and differentiable derivative function:

$$\frac{\partial L}{\partial u_i^T} = \frac{e^{u_i^T}}{\sum_{k=1}^{C} e^{u_k^T}} - y_i, \tag{2}$$

where y_i is class label. On the other hand, LIF neurons in hidden layers bring non-differentiability as the neuron generates a spike output whenever the membrane potential u_i^t exceeds the firing threshold. Following the previous work, we use an approximate gradient for backpropagation:

$$\frac{\partial o_i^t}{\partial u_i^t} = \frac{1}{\pi} arctan(\pi x) + \frac{1}{2}. \tag{3}$$

D. K Matrix Visualization

In our main manuscript, we introduce a kernel matrix by computing Hamming distance $d_H(\mathbf{c}_i, \mathbf{c}_j)$ between different samples i and j, which can be formulated as follows:

$$\mathbf{K}_H = \begin{pmatrix} N_A - d_{SAH}(\mathbf{c}_1, \mathbf{c}_1) & \cdots & N_A - d_{SAH}(\mathbf{c}_1, \mathbf{c}_N) \\ \vdots & \ddots & \vdots \\ N_A - d_{SAH}(\mathbf{c}_N, \mathbf{c}_1) & \cdots & N_A - d_{SAH}(\mathbf{c}_N, \mathbf{c}_N) \end{pmatrix} \tag{4}$$

Here, N_A stands for the number of LIF neurons in the given layer. Then, using this matrix, we compute the final score of the architecture candidate:

$$s = \log(\det|\sum_l \mathbf{K}_H^l|), \tag{5}$$

where, $\mathbf{K}_H^l$ is the kernel matrix at layer l. Finally, the highest-scored architecture among the candidates is selected for training. We clarify that a high score implies low off-diagonal elements of kernel matrix $\mathbf{K}_H$, which means that the activation patterns from different samples are not similar. To show this, we visualize the examples of kernel matrix in Fig. 2. We divide the matrices into two groups based on its post-training performance. The results show that architectures with low off-diagonal elements are likely to achieve higher post-training performance.

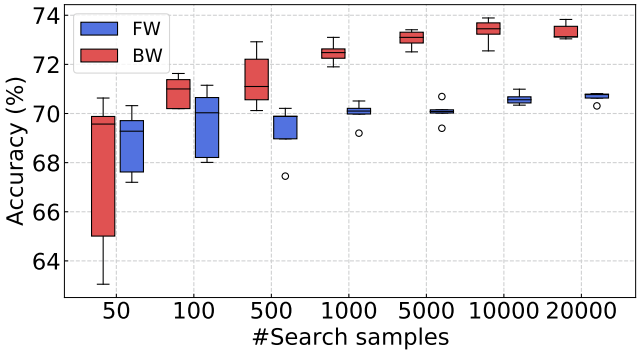


Fig. 3. Accuracy and with respect to number of search samples. For accuracy experiments, we run the same settings 5 times. We use CIFAR100 dataset.

Table 1. Memory / number of spikes on CIFAR100.

Method	Acc.(%)	Timestep	#Params(M)	#Spikes(10^6)
Han *et al.* [4] (VGG16)	70.90	2048	40.28	164.45
Rathi *et al.* [10] (VGG16)	69.67	5	40.28	0.555
Li *et al.* [7] (ResNet20)	72.33	32	11.29	0.225
SNASNet-Fw (ours)	70.20	5	20.54	0.077
SNASNet-Fw-AP4 (ours)	70.15	5	7.30	0.078
SNASNet-Fw-AP8 (ours)	69.24	5	4.15	0.077
SNASNet-Bw (ours)	73.31	5	20.62	0.092
SNASNet-Bw-AP4 (ours)	72.80	5	8.77	0.096
SNASNet-Bw-AP8 (ours)	70.18	5	5.55	0.092

E. Ablation on Number of Samples

In our experiments, we search 5000 architecture candidates from search space. Here, we provide more accuracy-#search samples configurations. In Fig. 3, the accuracy of both forward (marked as blue) and backward connections (marked as red) saturates after 5000 samples. The backward connection configuration shows higher variation as well as higher performance increase compared to that of the forward connection setting. This is because searching backward connections has larger search space than searching forward connections only.

F. Memory and Computational Efficiency

Finally, we compare the memory and computational efficiency between SNASNet and previous works [10,5,7] in Table 1. In the table, we also compare SNASNet-Fw-AP**x** and SNASNet-Bw-AP**x** where **x** is the kernel size of AvgPooling layer for the vectorize block (we use **x**=2 in our default setting). A large kernel size reduces the number of parameters in the classifier. We find the trade-off between the number of parameters in the classifier and performance. Our original

Table 2. Combining Fang *et al.* [2] with SNASNet-Bw on CIFAR10.

Method	Timestep	Accuracy
Fang *et al.* [2]	8	93.50
SNASNet-Bw	5	93.73 ± 0.32
SNASNet-Bw + Fang *et al.*[2]	5	93.92 ± 0.41

model (*i.e.*, SNASNet-Fw and SNASNet-Bw) requires more parameters than the ResNet architecture, while achieving higher accuracy with a smaller number of timesteps. However, if we use a larger kernel size such as 4 or 8, SNASNet requires fewer number of parameters. Surprisingly, compared to previous works, all SNASNet configurations show a significantly smaller number of spikes (less than $\sim 50\%$). The results demonstrate that the SNASNet founded by our search algorithm can improve not only accuracy but also efficiency.

G. Complementary Objective with respect to Prior Works

Existing state-of-the-art SNN works have proposed advanced techniques such as, normalization [14] and learning neuronal dynamics [2] among others, to achieve high-performing SNNs. The motivation of our work is to find high-performing SNNs through architecture search, which is a new line of research compared to previous works focusing on advanced training techniques. Thus, our work is complementary to previous methods. To demonstrate this, we combine our searched architecture with Fang *et al.* [2] where a trainable membrane time constant has been proposed and yields the best accuracy from previous works. In Table 2, our searched architecture is seamlessly combined with such training technique, resulting in further accuracy improvement with even lower timestep requirement. This corroborates our assertion that both careful architectural design and advanced training techniques are important for improving SNNs.

H. Applying VanRossum Distance for Metric

VanRossum distance [11] has been widely used for measuring a difference between spike trains by convolving an exponential kernel. To show the effect of such spike-crafted distance, we compare HD, SAHD, and VanRossum in Table 3. We notice that HD exhibits a lower accuracy than SAHD and VanRossum, indicating HD fails to characterize the spike activity. The VanRossum gets slightly lower results than SAHD (0.3%). We think the results of VanRossum can be further improved if we carefully tune the hyperparameter of τ, which is fixed to 1 in our experiments. However, we should also point out VanRossum is much slower compared to SAHD, mainly due to the exponential kernel function and the ℓ_2-distance between any two neurons, while the SAHD can be easily implemented by a matrix-vector multiplication. In sum, VanRossum shows a good performance but it needs careful hyperparameter tuning and suffers from higher computation cost.

Table 3. Combining Fang *et al.* [2] with SNASNet-Bw on CIFAR10.

Method (CIFAR10)	HD	SAHD (ours)	VanRossum
Time cost per model (sec)	5.98 ± 0.12	6.12 ± 0.09	44.52 ± 0.15
Accuracy (%)	90.12 ± 0.24	93.73 ± 0.32	93.44 ± 0.43

I. Comparison with NAS + ANN-SNN conversion

We compare the performance of our SNASNet with *ANN-NAS followed by ANN-SNN conversion*. We first train ANN architecture founded by two representative ANN-NAS algorithms (DARTS [8] and SPOS [3]) on CIFAR10. After that, we convert the ANNs using the recent ANN-SNN conversion algorithm [7] with various timesteps (T=16, 32, 64, 128, 256). Although the converted SNN achieves similar performance in large timesteps (T>128) compared to SNASNet, they show a huge performance drop with T=16. Our NAS approach shows better results even with less timesteps (T=5), showing the advantage of SNASNet.

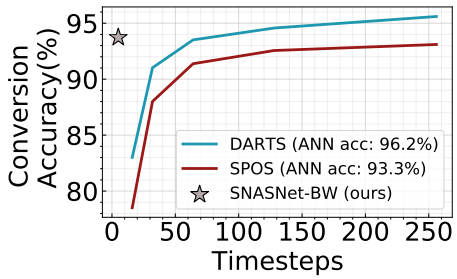


Fig. 4. Accuracy change with respect to number timesteps. We show our SNASNet-Bw and *ANN-NAS+ANN-SNN conversion*.

References

1. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255. Ieee (2009)
2. Fang, W., Yu, Z., Chen, Y., Masquelier, T., Huang, T., 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 (2021)
3. Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., Sun, J.: Single path one-shot neural architecture search with uniform sampling. In: European Conference on Computer Vision. pp. 544–560. Springer (2020)

4. Han, B., Roy, K.: Deep spiking neural network: Energy efficiency through time based coding. In: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part X 16. pp. 388–404. Springer (2020)
5. Han, B., Srinivasan, G., 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. pp. 13558–13567 (2020)
6. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
7. Li, Y., Deng, S., Dong, X., Gong, R., Gu, S.: A free lunch from ann: Towards efficient, accurate spiking neural networks calibration. arXiv preprint arXiv:2106.06984 (2021)
8. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055 (2018)
9. Neftci, E.O., Mostafa, H., Zenke, F.: Surrogate gradient learning in spiking neural networks. IEEE Signal Processing Magazine **36**, 61–63 (2019)
10. Rathi, N., Roy, K.: Diet-snn: A low-latency spiking neural network with direct input encoding and leakage and threshold optimization. IEEE Transactions on Neural Networks and Learning Systems (2021)
11. van Rossum, M.C.: A novel spike distance. Neural computation **13**(4), 751–763 (2001)
12. Wu, Y., Deng, L., Li, G., Zhu, J., Shi, L.: Spatio-temporal backpropagation for training high-performance spiking neural networks. Frontiers in neuroscience **12**, 331 (2018)
13. Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., Hutter, F.: Nas-bench-101: Towards reproducible neural architecture search. In: International Conference on Machine Learning. pp. 7105–7114. PMLR (2019)
14. Zheng, H., Wu, Y., Deng, L., Hu, Y., Li, G.: Going deeper with directly-trained larger spiking neural networks. arXiv preprint arXiv:2011.05280 (2020)