2024 25th International Symposium on Quality Electronic Design (ISQED) | 979-8-3503-0927-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISQED60706.2024.10528706

Unleashing Energy-Efficiency: Neural Architecture Search without Training for Spiking Neural Networks on Loihi Chip

Shiya Liu, Yang Yi
Bradley Department of Electrical and Computing Engineering,
Virginia Tech, Blacksburg, Virginia, 24061, USA
E-mail: {shiyal, yangyi8}@vt.edu

***Abstract*—Spiking neural networks (SNNs) offer energy-efficient computation due to their high-sparsity activation and event-driven nature. However, existing SNN designs often utilize suboptimal artificial neural network (ANN)-like architectures for binary sequence processing. Moreover, improving accuracy often leads to higher computational complexity, making it difficult to deploy SNNs on resource-constrained devices. Furthermore, SNN architectures tailored for GPUs may not fully exploit the energy-efficient capabilities of SNN models. To address these limitations, we present a novel neural architecture search (NAS) algorithm that merges recent advancements in ANNs and focuses on enhancing SNN architectures specifically for the Loihi chip. The Loihi chip is a neuromorphic computing chip designed to emulate the brain's neural networks, with particular strength in event-driven SNNs, making it an energy-efficient alternative to GPUs. Our algorithm efficiently selects an optimal architecture by leveraging gradients induced at initialization across diverse data samples, eliminating the requirement for training. We propose to design a search space that aligns with the chip's capabilities, taking into account its support for integer-only inference and the lack of advanced operators such as backward and shortcut connections. Experimental results on two image classification benchmarks demonstrate the superiority of our SNN models, which achieve comparable accuracy to state-of-the-art architectures while significantly reducing energy consumption per image and minimizing model size. Our approach paves the way for energy-efficient SNN designs on the Loihi chip, unlocking the full potential of SNNs for real-world applications.**

***Index Terms*—Artificial Neural Networks, Neural Architecture Search, Spiking Neural Networks**

I. Introduction

Spiking neural networks (SNNs) have emerged as a promising approach for achieving low-power intelligence [1]. SNNs employ sparse and asynchronous discrete events for communication between neurons, allowing for efficient hardware implementation. The event-based computation enables SNNs to perform energy-efficient inference on resource-constrained devices. Among the available hardware implementations, Loihi is a neuromorphic computing chip that stands out due to its ability to leverage asynchronous event-driven parallel computations to improve the speed and power efficiency of SNN models [2]. This innovative architecture of Loihi can benefit future applications that require real-time processing and energy efficiency. Therefore, the objective of this paper is to design accurate and efficient SNNs on the Loihi platform.

In numerous applications, a better neural architecture design tends to result in a considerable improvement in accuracy [3, 4, 5, 6, 7, 8, 9, 10]. However, it is observed that such enhancement in accuracy usually comes at the expense of increased computational complexity, which makes it more challenging to implement SNNs on devices with limited computational resources. The process of designing an accurate and efficient neural architecture for such resource-constrained devices is challenging due to the vast design space involved. To elaborate, let's take a 10-layer spiking convolutional neural network (CNN) as an example. Assume each layer of the network, can choose a different kernel size and a filter number from a set of $\{1, 3, 5\}$ and $\{32, 64, 128\}$, respectively. Even with these simplified design choices and shallow layers, the design space for this network contains approximately $(3 \times 3)^{10} \approx 3.5 \times 10^9$ possible architectures.

Neural architecture search (NAS) [11, 12, 13, 14, 15] is a technique to automate the design of neural networks. In recent years, NAS outperforms manually designed neural architectures in several tasks [11, 12, 13, 14, 15, 16]. However, standard NAS algorithms are computation-intensive and require training a supernet, which includes all architecture candidates [11, 12, 13, 14]. Due to the considerably slower training process of SNNs compared to ANNs, the aforementioned NAS approaches are not directly applicable to SNNs. Nonetheless, recent studies [17, 18, 19] have proposed efficient NAS approaches for ANNs that search for the best architecture from initialized networks without the need for any training. Inspired by the findings of [18], which suggest that network architectures with high representation power at initialization tend to achieve higher validation accuracy, researchers in [10] introduce a NAS approach for SNNs. This approach selects the SNN architecture capable of representing diverse spike activation patterns across various data samples. However, the method employs a complicated search space, large layer size, and utilizes advanced operators such as backward and shortcut connections [20, 21], rendering the architectures unsuitable

This work was supported in part by the U.S. National Science Foundation (NSF) under Grant CCF-1750450, Grant ECCS-1731928, Grant ECCS-2128594, Grant ECCS-2314813, and Grant CCF-1937487.

for resource-constrained devices. Additionally, the algorithm discovers optimal architectures for Graphics Processing Unit (GPU) hardware platforms, which may not be well-suited for efficiently executing SNNs.

In this paper, we propose a novel NAS algorithm for SNNs on the Loihi chip, without the need for training. We address the challenges of finding optimal SNN architectures on the Loihi chip by combining recent NAS approaches for ANNs [19]. Our algorithm selects an optimal architecture based on gradients induced by the architecture at initialization across different data samples without the need for training the architecture. Gradients are widely recognized as crucial for optimization in neural networks [22, 23, 24]. The authors in [19, 25] present a theoretical analysis to demonstrate that the gram matrix of gradients, which quantifies the dot product between any two gradient vectors, plays a crucial role in determining the convergence outcome of neural architectures. In our work, we conduct comprehensive experiments and exploit the gram matrix of gradients to evaluate randomly initialized architectures in place of downstream training. The larger the gram matrix, the higher the convergence rate of an SNN. According to our experimental results, the gram matrix of gradients exhibits a strong correlation with the model's validation accuracy, with a Spearman coefficient of 0.59 ($P \ll 0.001$). In terms of design space, we create a layer-wise search space and exclude advanced operators such as backward and short connections, as they are not supported by the Loihi chip. In our approach, we randomly sample 1000 architectures from the search space and pick the top 20 architectures with the largest mean gram matrix of gradients as candidates. Then, we train these 20 architectures to select the best one. Our experimental results on two image classification benchmarks demonstrate that our SNN models achieve comparable accuracy to state-of-the-art architectures with significantly lower energy consumption per image and a much smaller model size. The significance of our work is in the development of a novel NAS algorithm for finding accurate and efficient SNN architectures on the Loihi chip, without the need for training to evaluate candidate architectures. Our contributions are summarized below.

- This paper presents a novel NAS algorithm for SNNs on the Loihi chip, which selects an optimal architecture using gradients induced by the architecture at initialization across various data samples without training the architecture. Our work is significant because the proposed NAS algorithm enables us to identify accurate and efficient SNN architectures without the need for training to evaluate candidate architectures.
- To accommodate the capabilities of the Loihi chip, a layer-wise search space is created that excludes advanced operators, such as backward and short connections.
- The results of our experiments on two image classification benchmarks indicate that our searched SNN models achieve comparable accuracy to state-of-the-art architectures while exhibiting significantly lower energy consumption per image and a much smaller model size.

II. Background

A. Spiking Neural Networks

Both ANNs and SNNs draw inspiration from the brain's structure and functioning. However, ANNs diverge from the brain in their neural computations. One of the most prominent differences lies in the communication mechanism between neurons. In the brain, neurons communicate with each other through a series of spike trains, which are action potentials that are sparsely distributed in time and have uniform amplitude [1]. On the other hand, ANNs employ continuous real numbers to convey information between neurons. This fundamental distinction has led to the emergence of SNNs. In SNNs, information is transmitted through event-driven firing activities, where information is represented using 1-bit spikes.

Despite the potential for energy efficiency improvements in ANNs, SNNs offer a unique opportunity in this regard, as spike events are temporally sparse. Additionally, SNNs have the inherent ability to detect temporal properties of information transmission that are observed in biological neural systems. Previous research has demonstrated the reliability of the precise timing of each spike for multiple regions of the brain, indicating its significance in neural coding [26].

B. Neural Architecture Search

The primary objective of NAS is to replace neural networks that are manually designed with architectures that are learned through automated methods. In the early stages of NAS algorithms, reinforcement learning [12, 13] or evolutionary algorithms [27] were used. However, these approaches necessitated training the searched architecture from scratch for every search iteration, resulting in significant computational demands. To tackle this issue, weight-sharing techniques have been proposed [11, 14, 15]. These techniques train the supernet, which encompasses all architecture candidates, only once. For instance, Darts [15] simultaneously optimizes the network parameters and the significance of each architecture candidate. Also, ENAS is an efficient NAS algorithm and it reduces search cost by sharing parameters among child models [11]. The idea of sharing parameters among child models is inspired by transfer learning and multi-task learning that parameters learned by a model in a task can be exploited by another model on another task. The weight-sharing methods obviate the need for training the architecture from scratch at each search iteration, leading to significantly improved efficiency in contrast to earlier NAS algorithms. In the current research, considerable attention has been paid to improving the efficiency of the NAS technique [28, 29, 30]. This is due to the increasing size of datasets and network architectures. An approach involves the implementation of NAS without training, where the networks are not trained during the search phase [18, 19]. Such an approach can markedly reduce the computational burden involved in searching for optimal architecture.

III. Methodology

This section begins with an introduction to the gram matrix of gradients approach for assessing randomly generated neural

architectures. Subsequently, we outline our proposed NAS algorithm for SNNs. Finally, we present the search space utilized in our NAS algorithm.

A. Gram Matrix of Gradients

This section presents the gram matrix of gradients (GMG) as a way of evaluating the effectiveness of neural network architectures without training. The significance of gradients induced by neural networks is well-established in determining both convergence and generalization outcomes [22, 23, 24, 31].

we examine a multi-layer fully-connected neural network comprised of N layers. Specifically, we make the assumption that the i-th layer of the network produces an output defined as follows:

$$y^{(i)} = \sigma(w^{(i)} y^{(i-1)}), \tag{1}$$

where $w^{(i)}$ is the weight matrix and σ represents the activation function. The gradient of the i-th layer with respect to $w^{(i)}$ is defined as [19],

$$\frac{\partial L}{\partial w^{(i)}} = \langle \frac{\partial L}{\partial y^{(N)}}, y^{(i-1)} \left(\prod_{k=i+1}^{N} J^{(k)} w^{(k)} \right) J^{(i)} \rangle, \tag{2}$$

where L and $\frac{\partial L}{\partial y^{(N)}}$ denotes the loss function and the derivative of the last layer's output, respectively. $J^{(k)}$ is the diagonal matrix where the main diagonal consists of the derivatives of the activation with respect to the inputs in the k-th layer. $J^{(k)}$ can be represented as [19],

$$J^{(k)} = diag \left(\sigma'(w_1^{(k)} y^{(k-1)}), ..., \sigma'(w_d^{(k)} y^{(k-1)}) \right), \tag{3}$$

where d is defined as the dimension of the activation, and w_d^k as the d-th row in matrix w^k. The equation highlights that the gradients are dependent on the design of the neural network, which includes various factors such as network depth, activation function, initialization, among others. It is worth noting that the gradients can vary significantly based on the different architectures, leading to varying outcomes.

The authors in [19, 25] present a theoretical analysis that demonstrates the connection between the architecture of a neural network and its convergence behavior. According to their findings, the GMG, which quantifies the dot product between any two gradient vectors, plays a crucial role in determining the convergence outcome. As such, the GMG of a given neural network architecture can serve as a useful indicator for evaluating its downstream performance. Denoting the GMG as $H(t)$ and t as the t-th iteration, the following inequation holds [19]:

$$||y^* - y^{(N)}(t)||_2^2 \leq e^{-\lambda_{min}(H(t))t} ||y^* - y^{(N)}(0)||_2^2. \tag{4}$$

The upper bound of the loss is determined by $\lambda_{min}(H(t))$ and a higher $\lambda_{min}(H(t))$ corresponds to lower training losses. Additionally, since $H(t)$ is a symmetrical matrix, the Frobenius norm of $H(t)$ can be used to bounds $\lambda_{min}(H(t))$ as,

$$\lambda_{min}(H(t)) \leq \sqrt{\sum_i |\lambda_i|^2} = ||H(t)||_F, \tag{5}$$

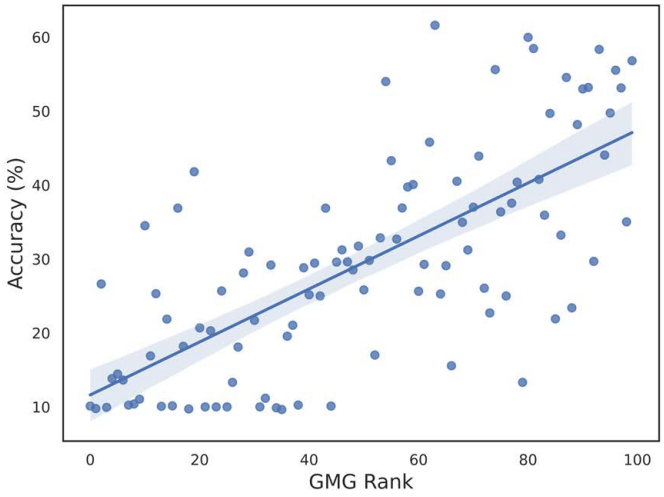

Fig. 1. Relationship between GMG rank and validation accuracy on the CIFAR-10 dataset. We generate 100 architectures and rank them by GMG values in descending order.

where λ_i is the eigenvalue of $H(t)$.

This theoretical analysis [19, 25] highlights that the Frobenius norm of a GMG provides an upper bound for the convergence rate. Each element in a GMG represents the dot product of two gradient vectors. This dot product is influenced by the gradient values observed across various data samples. The gradient values determine the optimization step size. If the gradient values become exceedingly small, it can lead to premature termination of training and reduce the convergence rate. A higher Frobenius norm value indicates a faster convergence rate. To summarize, the theoretical results emphasize that the GMG is a crucial and all-encompassing metric for evaluating the quality of a neural network's architecture. According to our experimental results on the CIFAR-10 dataset [32], a strong correlation was observed between the GMG metric and the validation accuracy of the model, as evidenced by a Spearman coefficient of 0.59 ($P \ll 0.001$). Fig. 1 shows the relationship between GMG metric and validation accuracy.

B. Neural Architecture Search without Training

In this study, we propose to employ the GMG to assess randomly-initialized SNNs. The approach involves utilizing the mean value of GMG of a given neural architecture as a useful indicator for evaluating its downstream performance. Specifically, we define GMG as an H matrix, where each entry (i, j) in the matrix represents the dot-product of two gradient vectors. We further obtain the gradient score g by computing the mean value of the elements in the H matrix.

$$g = \frac{1}{M^2} \sum_{i=1}^{M} \sum_{j=1}^{M} \left(\frac{\partial y_i^{(N)}}{\partial w} \right) \left(\frac{\partial y_j^{(N)}}{\partial w} \right)^T, \tag{6}$$

where N is the number of layers and M is the number of input samples.

For the implementation, we calculate the gradient score for every layer and aggregate them. To handle the issue of excessively long gradient vectors, we only select 100 weight parameters from each layer. The complete process can be represented as follows:

$$g = \frac{1}{NM^2} \sum_{n=1}^{N} \sum_{i=1}^{M} \sum_{j=1}^{M} \left(\frac{\partial y_i^{(N)}}{\partial w^n} \right) \left(\frac{\partial y_j^{(N)}}{\partial w^n} \right)^T, \quad (7)$$

where w^n is the weight parameters from the n-th layer.

The proposed NAS without training algorithm is presented in Algorithm 1. The algorithm begins by generating a total of I possible architectures at random from the search space S. For each architecture, we compute the gradient score based on Eq. 7. Then we choose the top-k architectures with the highest scores as potential candidates. Subsequently, these candidates are trained from scratch, and their performance is evaluated on a validation set, with a focus on the validation accuracy achieved at 30 epochs to minimize computational costs. To compute Eq. 7, we adopt the following algorithmic procedure. We start by dividing the n gradients for each parameter into two vectors, each containing $n/2$ items. Subsequently, we compute the dot-product between these two vectors. Ultimately, we select m parameters from each layer's GMG and compute the mean dot-product, which serves as the final gradient score.

Algorithm 1: Proposed NAS algorithm for SNN

Input: A training set $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$;
A validation set $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$;
A search space S
Output: The best SNN architecture A_{best}
specify max number of architectures I;
initialize candidates set C;
for $iteration \leftarrow 0$ **to** I **do**
  Sample an architecture A from search space S;
  Compute gradient score g using Eq. 7;
  C.add(A, g);
end
Sort candidates in C by gradient score in descending order;
Pick the top-k architectures from the sorted list;
Train the top-k architectures;
Evaluate these architectures on validation dataset;
Return the best architecture A_{best};

C. The Search Space

Loihi is a neuromorphic computing chip that stands out due to its ability to leverage asynchronous event-driven parallel computations to improve the speed and power efficiency of SNN models [2]. This innovative architecture of Loihi can benefit future applications that require real-time processing and energy efficiency. Nonetheless, the chip is constrained by its limited support for advanced operators, such as attention, backward, and shortcut connections, and only permits integer-only inference. Consequently, we formulated a search space that aligns with the chip's capabilities. To reduce the search computation cost of the proposed NAS, the layer-wise search space is adopted. Each cell in the search space is a spiking convolution with different kernel sizes and groups. Each convolution can choose a kernel size from the set $\{3, 5, 7\}$. Also, each convolution can pick a group from the set$\{1, 2, 4\}$. Large group sizes can reduce the number of parameters and increase hardware efficiency. The neuron we used is the sigma-delta neuron [33, 34].

There are 9 cells in the search space. Table I shows the list of cells in the search space. In Table I, the Neuron Type represents the type of neuron used in convolutions. Sigma-delta represents the sigma-delta neuron.

TABLE I
LIST OF CELLS IN THE SEARCH SPACE (NEURON TYPE REPRESENTS THE TYPE OF NEURON USED IN CONVOLUTIONS. SIGMA-DELTA REPRESENTS THE SIGMA-DELTA NEURON

Cell	Kernel	Groups	Neuron Type
1	3	1	Sigma-delta
2	3	2	Sigma-delta
3	3	4	Sigma-delta
4	5	1	Sigma-delta
5	5	2	Sigma-delta
6	5	4	Sigma-delta
7	7	1	Sigma-delta
8	7	2	Sigma-delta
9	7	4	Sigma-delta

The macro architecture is shown in Table II. We do not use any advanced operators such as backward and shortcut connections in our neural architectures. The first layer is the spike encoding layer, responsible for the direct conversion of a floating-point image into spikes [3, 35]. We employ the direct encoding technique [3, 35], wherein the input image is passed through the first layer for a duration of T time steps. There are 9 layers that need to be searched and each layer has 9 options. Therefore, there are $9^9 \approx 3.9 \times 10^8$ possible architectures. It is very time-consuming if an exhaustive search algorithm is adopted to find the optimal architecture.

TABLE II
MACRO-ARCHITECTURE

Input Dimension	Layer Type	Output Channel	Kernel Stride
$32 \times 32 \times 3$	3x3 Conv	96	1
$16 \times 16 \times 96$	To be Searched	96	2
$16 \times 16 \times 96$	To be Searched	96	1
$16 \times 16 \times 96$	To be Searched	96	1
$8 \times 8 \times 96$	To be Searched	192	2
$8 \times 8 \times 192$	To be Searched	192	1
$8 \times 8 \times 192$	To be Searched	192	1
$4 \times 4 \times 192$	To be Searched	256	2
$4 \times 4 \times 256$	To be Searched	256	1
$4 \times 4 \times 256$	To be Searched	256	1
4096	Dense	128	1
128	Dense	10	N/A

TABLE III
ACCURACY, ENERGY CONSUMPTION PER IMAGE, AND MODEL SIZE COMPARISON ON THE CIFAR-10 DATASET BETWEEN DIFFERENT MODELS (M IN THE NUMBER OF PARAMS MEANS MILLION.)

Network	Training Method	Number of Params	Energy/Image(J)	Energy Consumption	Accuracy(%)
Wu *et al.* [3]	Surrogate Gradient	44.5M	16.97	94.22X	90.53
Rathi *et al.* [4]	Hybrid	39.9M	18.32	101.78X	90.50
Deng *et al.* [6]	ANN-SNN Conversion	35.6M	6.98	38.78X	92.42
Li *et al.* [7]	ANN-SNN Conversion	14.7M	6.69	37.16X	92.98
Fang *et al.* [8]	Surrogate Gradient	36.7M	13.66	75.89X	93.50
Wu *et al.* [9]	Tandem Learning	44.5M	16.97	94.22X	89.04
Kim *et al.* [10]	Surrogate Gradient	19.5M	4.43	24.61X	93.12
Model-A (Our on GPU)	Surrogate Gradient	5.5M	0.82	4.56X	91.81
Model-A (Our on Loihi)	Surrogate Gradient	5.5M	0.18	1.00X	91.42

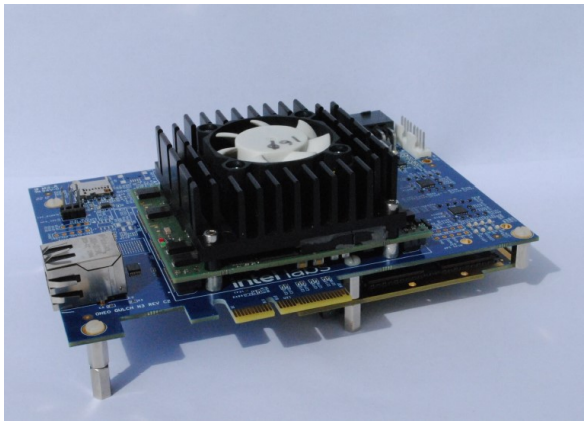

Fig. 2. The Loihi 2 board [37]

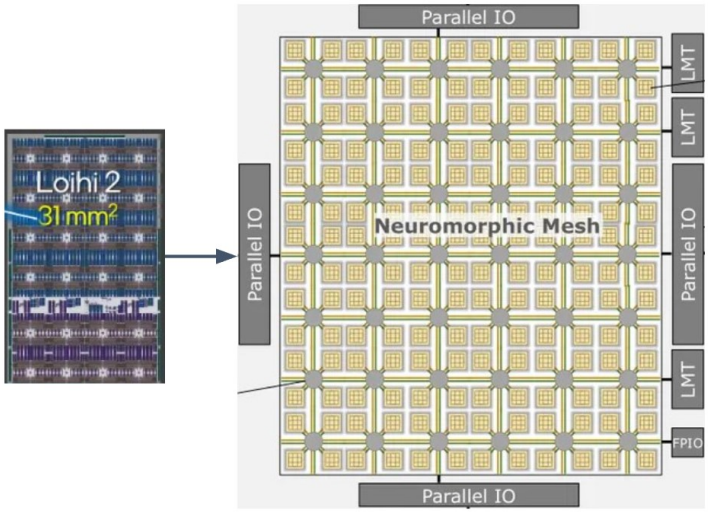

Fig. 3. The Loihi 2 chip [38]

IV. EXPERIMENTAL RESULTS

We implement the proposed NAS using python and PyTorch [36]. All the GPU experiments are conducted on the NVIDIA GeForce RTX 2080 hardware platform. All Loihi experiments are conducted on Loihi 2. The Loihi 2 board is shown in Fig. 2 [37]. Loihi is a neuromorphic research test chip designed by Intel Labs. It uses asynchronous event-driven fine-grained parallel computations to improve the speed and the power efficiency of SNN models [2]. It consists of 128 "neurocores" and each "neurocore" has 1M neurons. The chip is fabricated on Intel 4 EUV process [38]. The details of the chip are summarized in Fig. 3.

In our experiments, the CIFAR-10 [32] and Fashion-MNIST [39] datasets are used. An image in the CIFAR10 dataset has a shape of $32 \times 32 \times 3$. The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The Fashion-MNIST dataset consists of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. The accuracy, energy consumption per image, and the number of model parameters are compared. The delta unit threshold and the surrogate gradient relaxation in the sigma-delta neuron are set to 0.1 and 0.5, respectively.

We use the following abbreviations to represent different models in our experiments.

- **"Model-A":** Model-A is the model that was searched, trained, and tested on the CIFAR-10 dataset.
- **"Model-B":** Model-B is the model that was searched, trained, and tested on the Fashion-MNIST dataset.
- **"Model-C":** Model-C is the model that was searched on the CIFAR-10 dataset but trained and tested on the Fashion-MNIST dataset. It is used to evaluate if the architecture can be transferred to other tasks.

TABLE IV
ACCURACY COMPARISON ON THE FASHION-MNIST DATASET BETWEEN DIFFERENT MODELS

Network	Training Method	Accuracy(%)
Wu *et al.* [3]	Surrogate Gradient	91.97
Rathi *et al.* [4]	Hybrid	92.11
Deng *et al.* [6]	ANN-SNN Conversion	93.95
Li *et al.* [7]	ANN-SNN Conversion	94.12
Fang *et al.* [8]	Surrogate Gradient	94.38
Kim *et al.* [10]	Surrogate Gradient	94.46
Chen *et al.* [40]	Surrogate Gradient	92.07
Model-B (Ours)	Surrogate Gradient	93.77
Model-C (Ours)	Surrogate Gradient	93.52

A. Training Setup

We generate 1000 neural architectures and pick the top 20 architectures with the highest GMG scores. Then, we train these 20 neural architectures with the following learning algorithm. We use Adam learning algorithm [41], and the optimizer parameters are learning rate=0.005, beta1=0.9, beta2=0.999, epsilon=1e-07. We add two learning scheduler. The first scheduler is a warmup scheduler, the learning rate increases linearly from 0.001 to 0.005 in 5 epochs. Then, it is followed by a step scheduler and the learning rate is multiplied by 0.1 every 100 epochs. The training epoch is 150.

B. Performance Comparison

1) CIFAR-10 Dataset: Table III presents a comparison of accuracy, energy consumption per image, and model size among various implementations. We set the batch size of inference for both GPU and Loihi at 1 since Loihi only supports a batch size of 1.

Table III reveals that our searched model on Loihi yields comparable accuracy to state-of-the-art architectures, such as Wu et al. [3], Rathi et al. [4], Deng et al. [6], Li et al. [7], Fang et al. [8], Wu et al. [9], and Kim et al. [10]. However, our model surpasses the others in terms of energy consumption per image and model size. Specifically, the energy consumption per image of our SNN model on Loihi is significantly lower than the other models, with energy reductions of 94.22X, 101.78X, 38.78X, 37.16X, 75.89X, 94.22X, and 24.61X compared to models in Wu et al. [3], Rathi et al. [4], Deng et al. [6], Li et al. [7], Fang et al. [8], Wu et al. [9], and Kim et al. [10], respectively. Also, the energy consumption per image of our SNN model on GPU is approximately 20.5X, 22.1X, 8.5X, 8.1X, 16.6X, 20.5X, and 5.4X lower than these models, respectively. Furthermore, our model's number of parameters is considerably smaller than these models, with reductions of 8.1X, 7.3X, 6.5X, 2.7X, 6.7X, 8.1X, and 3.5X, respectively.

2) Fashion-MNIST Dataset: To investigate the dependency of our NAS algorithm on the dataset, we conducted an additional experiment on the Fashion-MNIST dataset. Specifically, we searched for the optimal architecture on the CIFAR-10 dataset and trained and tested the searched architecture on the Fashion-MNIST dataset. The obtained results are presented in Table IV, which includes the accuracy of Model-B, searched on the Fashion-MNIST dataset, and Model-C, searched on the CIFAR-10 dataset. Our results demonstrate that both models achieve accuracy that is comparable to state-of-the-art architectures, including Wu *et al.* [3], Rathi *et al.* [4], Deng *et al.* [6], Li *et al.* [7], Fang *et al.* [8], Kim *et al.* [10], and Chen *et al.* [40]. Moreover, the negligible accuracy difference between Model-B and Model-C confirms the transferability of the searched architectures across diverse datasets.

V. Conclusion

This paper proposes a NAS algorithm that aims to design efficient and accurate SNN models running on the Loihi chip. Our proposed NAS algorithm employs the GMG metric to evaluate randomly-initialized architectures, obviating the need for downstream training. Given that the Loihi chip supports only integer-only inference and lacks support for advanced operators like backward and shortcut connections, we designed a layer-wise search space that aligns with the chip's capabilities. On two image classification benchmarks, our searched SNN models achieve comparable accuracy to state-of-the-art architectures while consuming significantly lower energy per image and having a much smaller model size.

References

[1] Samanwoy Ghosh-Dastidar and Hojjat Adeli. "Spiking neural networks". In: *International journal of neural systems* 19.04 (2009), pp. 295–308.

[2] Mike Davies et al. "Loihi: A neuromorphic manycore processor with on-chip learning". In: *IEEE Micro* 38.1 (2018), pp. 82–99.

[3] Yujie Wu et al. "Direct training for spiking neural networks: Faster, larger, better". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 1311–1318.

[4] Nitin Rathi et al. "Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation". In: *arXiv preprint arXiv:2005.01807* (2020).

[5] Hanle Zheng et al. "Going deeper with directly-trained larger spiking neural networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 12. 2021, pp. 11062–11070.

[6] Shikuang Deng and Shi Gu. "Optimal conversion of conventional artificial neural networks to spiking neural networks". In: *arXiv preprint arXiv:2103.00476* (2021).

[7] Yuhang Li et al. "Converting Artificial Neural Networks to Spiking Neural Networks via Parameter Calibration". In: *arXiv preprint arXiv:2205.10121* (2022).

[8] Wei Fang et al. "Incorporating learnable membrane time constant to enhance learning of spiking neural networks". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 2661–2671.

[9] Jibin Wu et al. "A tandem learning rule for effective training and rapid inference of deep spiking neural networks". In: *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[10] Youngeun Kim et al. "Neural architecture search for spiking neural networks". In: *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXIV*. Springer. 2022, pp. 36–56.

[11] Hieu Pham et al. "Efficient neural architecture search via parameters sharing". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4095–4104.

[12] Barret Zoph and Quoc V Le. "Neural architecture search with reinforcement learning". In: *arXiv preprint arXiv:1611.01578* (2016).

[13] Dimitrios Stamoulis et al. “Single-path nas: Designing hardware-efficient convnets in less than 4 hours”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2019, pp. 481–497.

[14] Bichen Wu et al. “Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 10734–10742.

[15] Hanxiao Liu, Karen Simonyan, and Yiming Yang. “DARTS: Differentiable Architecture Search”. In: *International Conference on Learning Representations*. 2018.

[16] Mingxing Tan and Quoc Le. “Efficientnetv2: Smaller models and faster training”. In: *International conference on machine learning*. PMLR. 2021, pp. 10096–10106.

[17] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. “Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective”. In: *arXiv preprint arXiv:2102.11535* (2021).

[18] Joe Mellor et al. “Neural architecture search without training”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 7588–7598.

[19] Jingjing Xu et al. “KNAS: green neural architecture search”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 11613–11625.

[20] Guillaume Bellec et al. “A solution to the learning dilemma for recurrent networks of spiking neurons”. In: *Nature communications* 11.1 (2020), p. 3625.

[21] Wenrui Zhang and Peng Li. “Spike-train level backpropagation for training deep recurrent spiking neural networks”. In: *Advances in neural information processing systems* 32 (2019).

[22] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training recurrent neural networks”. In: *International conference on machine learning*. Pmlr. 2013, pp. 1310–1318.

[23] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[24] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[25] Arthur Jacot, Franck Gabriel, and Clément Hongler. “Neural tangent kernel: Convergence and generalization in neural networks”. In: *Advances in neural information processing systems* 31 (2018).

[26] Wyeth Bair and Christof Koch. “Temporal precision of spike trains in extrastriate cortex of the behaving macaque monkey”. In: *Neural computation* 8.6 (1996), pp. 1185–1202.

[27] Yuqiao Liu et al. “A survey on evolutionary neural architecture search”. In: *IEEE transactions on neural networks and learning systems* (2021).

[28] Mohamed S Abdelfattah et al. “Zero-cost proxies for lightweight nas”. In: *arXiv preprint arXiv:2101.08134* (2021).

[29] Zhicheng Yan et al. “Fp-nas: Fast probabilistic neural architecture search”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 15139–15148.

[30] Tien-Ju Yang, Yi-Lun Liao, and Vivienne Sze. “Netadaptv2: Efficient neural architecture search with fast super-network training and architecture optimization”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 2402–2411.

[31] Kun Yuan, Qing Ling, and Wotao Yin. “On the convergence of decentralized gradient descent”. In: *SIAM Journal on Optimization* 26.3 (2016), pp. 1835–1854.

[32] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. “The cifar-10 dataset”. In: *online: http://www. cs. toronto. edu/kriz/cifar. html* 55 (2014), p. 5.

[33] Davide Zambrano and Sander M Bohte. “Fast and efficient asynchronous neural computation with adapting spiking neural networks”. In: *arXiv preprint arXiv:1609.02053* (2016).

[34] Manu V Nair and Giacomo Indiveri. “An ultra-low power sigma-delta neuron circuit”. In: *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2019, pp. 1–5.

[35] Wenrui Zhang and Peng Li. “Temporal spike sequence learning via backpropagation for deep spiking neural networks”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 12022–12033.

[36] Adam Paszke et al. “Pytorch: An imperative style, high-performance deep learning library”. In: *arXiv preprint arXiv:1912.01703* (2019).

[37] Garrick Orchard et al. “Efficient neuromorphic signal processing with loihi 2”. In: *2021 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE. 2021, pp. 254–259.

[38] *Intel Introduces 2nd Gen Neuromorphic Research Chip: Loihi 2 on Intel 4 EUV Process*. https://fuse.wikichip.org/news/6383/intel-introduces-2nd-gen-neuromorphic-research-chip-loihi-2-on-intel-4-euv-process/. Accessed: 2023-01-30.

[39] Han Xiao, Kashif Rasul, and Roland Vollgraf. “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms”. In: *arXiv preprint arXiv:1708.07747* (2017).

[40] Xiang Cheng et al. “LISNN: Improving spiking neural networks with lateral interactions for robust object recognition.” In: *IJCAI*. 2020, pp. 1519–1525.

[41] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).