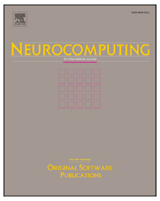
Neurocomputing 654 (2025) 131244

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom



Predictor-assisted evolutionary neural architecture search for spiking neural networks

Yuting Wang[a], Yu Xue[b,*], Wei Ding[a], Yiyu Tan[c], Peng Chen[d], Mohamed Wahib[d]

[a] School of Electronic and Information Engineering, Anhui Jianzhu University, Hefei, 230601, China
[b] School of Software, Nanjing University of Information Science and Technology, Nanjing, 210044, China
[c] Faculty of Science and Engineering, Iwate University, Morioka, 020-8551, Japan
[d] RIKEN Center for Computational Science (R-CCS), Kobe, Japan

ARTICLE INFO

Communicated by J. Liu

Keywords:
Spiking neural networks
Evolutionary neural architecture search
Performance predictor
Multi-objective optimization

ABSTRACT

Spiking neural networks (SNNs) are a biologically inspired computing paradigm that transmit information through discrete sequences of spikes and have good biological interpretability and energy efficiency. The performance of SNNs primarily depends on their architectures. However, current SNN architectures are mainly designed manually, significantly rely on expert experience and lack flexibility. In this research, we propose the PESNN, a predictor-assisted evolutionary neural architecture search for SNNs, to automatically design SNN architectures in which the entire algorithm follows the evolutionary neural architecture search (ENAS) framework. Unlike the existing search methods for SNN architectures, which still involve some manual design, we propose a variable-length coding strategy based on spatial channel adaptation for fully automatic architecture design. Since the fitness evaluation stage in the search is very computationally expensive, we design a performance predictor to assist this stage to speed up the search process. Moreover, a multi-objective optimization method is introduced to reduce the number of spikes and power consumption. Compared with other methods, the experimental results with the static datasets (CIFAR-10, CIFAR-100) and neuromorphic dataset (DVS-CIFAR-10) demonstrate that the PESNN can find the optimal network architecture with high classification accuracy and fewer spikes in fewer timesteps.

1. Introduction

Deep neural networks (DNNs) have developed rapidly and become popular tools for achieving tasks such as image classification [1,2], object detection [3–5], and semantic segmentation [6,7]. DNNs have been widely used on edge devices such as mobile phones, smart home devices, and unmanned aerial vehicles (UAVs). As the amount of data increases, the complexity of the network continues to grow. Existing computing platforms face enormous energy consumption challenges [8]. In DNNs, spiking neural networks (SNNs), as the third generation of neural networks [9], have attracted widespread attention from researchers in recent years. SNNs process information between neurons through spike sequences. Neurons are activated when an event occurs (such as membrane potential exceeding threshold). This event-driven mechanism significantly reduces energy consumption, allowing SNNs to exhibit higher energy efficiency and potential in resource-limited edge computing environments [10,11].

Many hand-designed SNN architectures have been proposed, inspired by the architecture design of traditional DNNs (e.g., VGG [12], ResNet [13], Transformer [14]). Han et al. proposed using a "soft reset" spiking neuron model called residual membrane potential (RMP) to achieve deeper high-precision, low-latency SNNs. Specifically, they used the RMP neuron model to perform almost lossless ANN to SNN conversion (A2S) of VGG and ResNet [15]. However, this approach of simply replacing the ReLU activation layer in the network with spiking neurons suffers from degradation problems and hardly achieves residual learning. Fang et al. proposed that spike-element-wise ResNet implements residual learning in deep SNNs, which can realize identity mapping well [16]. In addition, the A2S method usually requires extended encoding and cannot exploit spatio-temporal features to solve temporal tasks

* Corresponding author.
Email addresses: 202333105179@stu.ahjzu.edu.cn (Y. Wang), xueyu@nuist.edu.cn (Y. Xue), orangeding@163.com (W. Ding), tanyiyu@iwate-u.ac.jp (Y. Tan), peng.chen@a.riken.jp (P. Chen), mohamed.attia@riken.jp (M. Wahib).

https://doi.org/10.1016/j.neucom.2025.131244
Received 23 December 2024; Received in revised form 2 March 2025; Accepted 7 August 2025
Available online 13 August 2025

during training. Zheng et al. adopted a threshold-dependent batch normalization method based on spatio-temporal backpropagation (STBP), which can directly train SNNs in the spatio-temporal domain and support the training of very deep SNNs [17]. As the depth of the network increases, model degradation problems may be encountered. Feng et al. proposed an STBP method with multi-level firing and a spiking dormant-suppressed residual network (DS-ResNet) [18]. This method can realize more efficient gradient propagation and effectively alleviate the difficulty of deep network training, but it may lead to suboptimal solutions for SNNs. The current SNNs are mainly hand-designed fixed architectures, which limits the flexible applicability of SNNs in different application scenarios. In addition, designing the network architecture to solve a specific problem is a complex task for humans and usually relies on expert experience.

Neural architecture search (NAS) is a technique that automatically designs network architectures to meet specific task requirements, which can effectively solve the previously mentioned problems and has achieved remarkable success in DNNs [19–21]. Kim et al. first introduced the NAS approach to SNNs, and the proposed algorithm fully applied the temporal information in SNNs to search SNN architectures from the activation patterns at initialization [22]. Although the proposed algorithm achieved excellent performance on the image classification task dataset, it required a manually designed backbone network, only searched local cells in the network, and was challenging to find the globally optimal network architecture. In addition, they only focused on the accuracy of SNN, which makes the searched network architecture perform poorly in other indexes (e.g., number of spikes/power consumption). Na et al. proposed a spike-aware NAS framework called AutoSNN, which first trained a supernet that encoded all the architectures. After training, it executed an evolutionary search algorithm to find the optimal network architecture [23]. Man et al. used the multi-attention differentiable architecture search method to look for the best network structure for SNNs [24]. However, these methods require a pre-designed supernet, which strongly relies on professional experience. Training the supernet is time-consuming and may also suffer from performance crashes [25]. The above methods still involve the manual design of backbone networks or supernets. Therefore, it is necessary to investigate automatic design methods for SNNs that balance performance and power consumption.

In NAS, performance evaluation often requires complete training of each network architecture, resulting in deep neural network evolution consuming significant computational resources and time. For example, the automatic evolution CNN algorithm AE-CNN took 27 GPU days [26]. The neural architecture transfer algorithm proposed by Lu et al. required 59 GPU days [27]. The partial weight-sharing OSNet algorithm took 8.6 GPU days to search [28]. Some acceleration methods to reduce computational consumption have been proposed to address the inherent cost challenges of NAS, including the zero-shot method, one-shot method, and few-shot method. Among them, the zero-shot method [29] scored candidate architectures in their untrained state by using low-cost computational methods instead of expensive training in NAS. Since it did not go through the actual training process, it was prone to making misjudgments. The one-shot method [30–32] meant that the candidate architecture was trained on a single or very limited dataset, and the algorithm only needed to train a supernet. Although it saved evaluation time, it still took a long time for the supernet to reach convergence. In addition, the weights might not be appropriate for some sub-networks, resulting in poor performance or even collapse. Recently, surrogate models (or performance predictors) have been used to speed up performance evaluation [33–35]. These methods first trained a certain number of network architectures to obtain their test accuracy. These empirical data were then used to train the surrogate model. Subsequently, given an architecture, the model would predict its performance. This avoided the enormous computational overhead of training all architectures and sped up the architecture search process [36,37]. Although the surrogate model plays a better role than other acceleration methods, it still has some shortcomings. For example, it is inefficient to use random samples in the search space to train the model, and it is necessary to use high-performance architecture samples to train the model to improve the model's differentiation ability and guide the architecture search more effectively.

With the above analysis, our goal is to achieve the automatic design of SNNs using the evolutionary neural architecture search (ENAS) algorithm. First, we encode the network architecture using variable-length coding. This is because the optimal network depth for a specific task is unknown, and a fully automated design of SNNs is not possible if multiple manual attempts are made to find the optimal depth. Second, some approaches only focus on the accuracy of SNNs while ignoring the power consumption, which makes SNNs tough to deploy in practice. Therefore, we propose a multi-objective evolutionary search for SNNs. Moreover, a performance predictor is designed to assist the performance evaluation phase of ENAS. Our proposed predictor-assisted evolutionary neural architecture search algorithm for spiking neural networks is called PESNN.

The main contributions are summarized as follows:

(1) In response to the fact that the design methods of existing SNNs rely entirely or partially on manual work, a variable-length coding strategy based on spatial channel adaptation is proposed to adapt to different network structures, thereby achieving the goal of fully automatic design of SNN architectures.
(2) Considering the power consumption of SNNs for applications in edge devices, neuromorphic computing, and other fields, we introduce a multi-objective optimization strategy during the evolutionary search process to find high-performance and low-power consumption network architectures.
(3) An online performance predictor is proposed to assist the performance evaluation process in reducing the high computational overhead of the search process. The experimental results based on both static and neuromorphic datasets demonstrated that the SNN architecture searched by the PESNN algorithm showed significant improvements in performance and power consumption compared with related methods.

The rest of this paper is organized as follows. Section 2 describes the background and related work of PESNN, including the spiking neuron model, the training methods of spiking neural networks, and the neural architecture search. Section 3 presents the proposed PESNN algorithm. Section 4 provides the experimental setup, experimental results, and analysis, followed by conclusions and future works described in Section 5.

2. Background and related work

2.1. Spiking neuron model

SNNs are energy-efficient models with low latency and efficient neuromorphic computation by simulating the properties of neuronal communication in the human brain [38]. The spiking neuron is the basic computational unit in SNNs. This paper uses a Leaky Integrate-and-Fire (LIF) neuron model with low computational cost and good biological interpretability. It accomplishes information processing through membrane potential accumulation, spike firing, and membrane potential reset [8]. The dynamics of LIF neurons are modeled as follows:

$$\tau \frac{dU(t)}{dt} = -U(t) + I(t), \tag{1}$$

where τ is the membrane potential time constant, $U(t)$ denotes the membrane potential of the neuron at timestep t, and $I(t)$ represents the external input received by the neuron at timestep t. $I(t)$ is expressed as follows:

$$I(t) = \sum_{i=1}^{n} w_i s_i(t), \tag{2}$$

where n is the number of pre-neurons, w_i denotes the synaptic weight from pre-neuron i to the current neuron, and $s_i(t) = H(U(t) - U_{\text{th}})$ is the output of the pre-neuron. U_{th} is the membrane potential threshold. $H(\cdot)$ is the Heaviside function that outputs one when the input is greater than or equal to zero, and zero otherwise. This tensor with an output value of zero or one is referred to as a spike.

When the neuron releases the spike, it consumes the previously accumulated membrane potential. Therefore, there is a momentary decrease called the reset of the membrane potential. There are two ways to realize the process of resetting the membrane potential. One is the hard way in which the membrane potential will be directly set to its set value [39]. Another way is called the soft reset method, where the membrane potential decreases by the threshold value after a spike is emitted. Since the soft method can retain more temporal information [40], we choose this method to reset the membrane potential.

2.2. Training methods of SNNs

The mainstream training methods for SNNs can be divided into two categories. One is the A2S method, and the other is the direct-training surrogate gradient (SG) method. The A2S method converts pre-trained ANNs to SNNs. This typically involves converting continuous-valued neurons in ANNs to spiking neurons in SNNs and incorporating scaling operations such as weight normalization and threshold balancing [41]. Although some A2S methods have achieved comparable accuracy to ANNs [42,43], they often require more timesteps to approach the accuracy of pre-trained ANNs, increasing the inference latency of SNNs. In addition, the A2S method is difficult to make compatible with neuromorphic datasets [44]. The non-differentiable spike function in SNNs makes the traditional backpropagation algorithm incapable of direct application. To address the non-differentiability issue, Meng et al. proposed the differentiable spiking representation (DSR) method for training SNNs. This approach encodes spike trains into a weighted firing rate representation, transforming the forward computation of SNNs into a differentiable mapping, thereby avoiding the non-differentiability problem inherent in traditional training methods. Moreover, DSR does not require backpropagation through time [45]. In contrast, the SG method uses a differentiable approximate gradient to enable back-propagation to effectively optimize SNNs, and has been widely used in various domains. For instance, Su et al. first employed SG to train deep SNNs for object detection tasks [46]. Yang et al. combined the SG method to propose a new self-learning high-order information bottleneck strategy (SeLHIB), which enhances the model's ability to represent input information through the information bottleneck framework, and shows high generalization ability and robustness in the optical flow estimation task [47]. They also used SG technology to propose a high-order spike information bottleneck (HOSIB) framework to effectively improve the training efficiency of deep SNNs when facing complex and large datasets [48]. Compared to the A2S method, SG has the advantage of achieving good performance within limited timesteps. Meanwhile, the SG method can efficiently train SNNs on neuromorphic datasets [49]. Therefore, the SG method is chosen in this research.

2.3. Neural architecture search

The performance of neural networks primarily depends on their architectures [50]. Manually designed network architectures often rely on expert experience. NAS is an automated approach for designing network architectures. All possible network architectures are defined in a search space, and a search strategy is used to explore network architectures with superior performance in the search space. Mathematically, NAS can be formulated as the following optimization problem:

$$A^* = \arg\max_{A_i \in \mathcal{A}} f\left(A_i\right), \tag{3}$$

where $\mathcal{A} = \left\{A_1, A_2, \dots, A_n\right\}$ represents all possible network architectures in the search space. A_i is the candidate architecture, and $f\left(A_i\right)$ is the function that evaluates the fitness of architecture A_i. The goal of NAS is to continuously optimize Eq. (3) to find the optimal architecture.

NAS can be categorized into three types according to the search strategy: reinforcement learning-based NAS, gradient-based NAS, and evolutionary algorithm-based NAS. Zoph et al. used reinforcement learning for the search of neural architectures [51]. Specifically, a recurrent neural network was used as a controller to generate a model description of the neural network. The corresponding network architecture was then trained on a dataset, and the validation accuracy was used as a reward signal for updating the controller parameters, thus continuously improving the performance of the generated architectures. Although NAS based on reinforcement learning can search for network architectures with excellent performance, it is computationally expensive. For example, the NAS method used 800 GPUs to search for 28

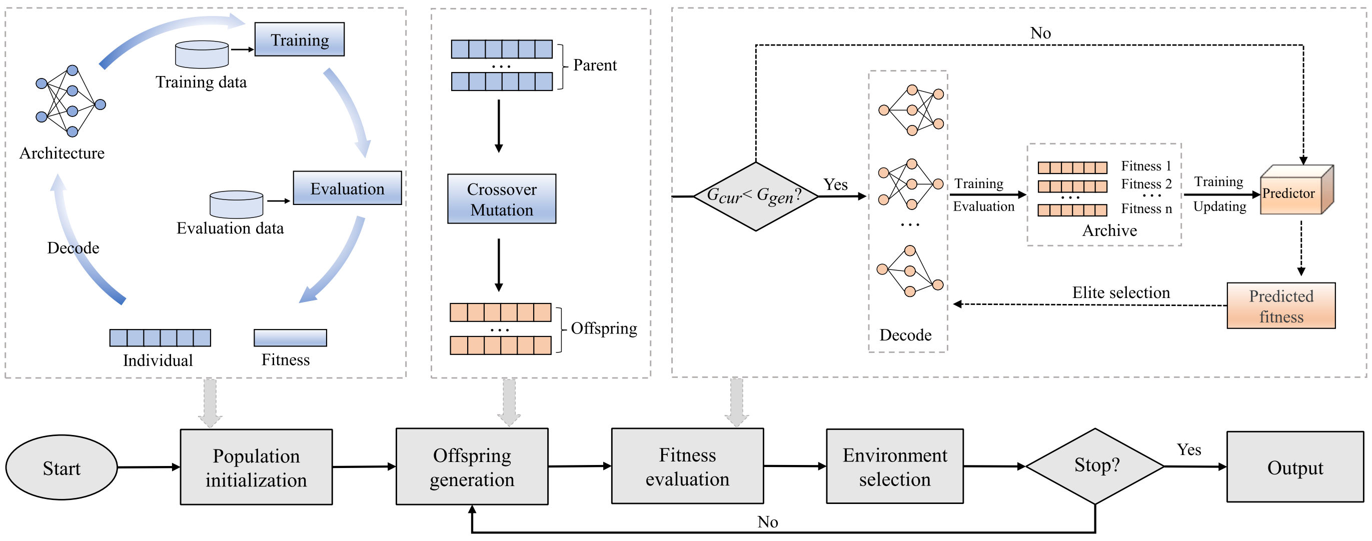


Fig. 1. The general framework of our PESNN. Following traditional ENAS, the first stage evolves G_{gen} generation individuals, and the second stage uses a predictor to speed up the evolutionary search process.

days [52]. Unlike NAS methods that apply reinforcement learning on a non-differentiable search space, differentiable architecture search [53] relaxed the search space to be continuous, allowing gradient descent algorithms to search for architectures effectively. This gradient-based NAS algorithm can achieve competitive performance using fewer computing resources [54]. However, it is prone to searching for poorly performing networks and needs to pre-design a supernet.

ENAS uses the principle of natural selection to simulate the evolutionary process to search for excellent network architectures, which can effectively balance the advantages and disadvantages of the above two techniques [50,55]. At the same time, it can effectively avoid local optimal solutions and has global search capabilities. It maintains diversity through population evolution and enhances exploration capabilities [56]. Besides, ENAS has better biological interpretability and is more compatible with SNN models inspired by neurons in the brain. Therefore, we use the ENAS method in this research.

3. Proposed method

The traditional ENAS algorithm includes population initialization, offspring generation, fitness evaluation, and environmental selection. Through continuous iterative evolution, inferior individuals are discarded while superior individuals are retained, and the individual with the best performance is usually obtained. Similarly, the overall framework of the evolutionary neural architecture search for SNN is shown in Fig. 1. In order to alleviate the time-consuming process of fully training network architectures to obtain their ground-truth performance, we use a performance predictor to assist the evaluation process in this work. The pseudo-code of PESNN algorithm is given in Algorithm 1.

3.1. Overall framework

(1) **Population initialization** (lines 1–7): First, according to the architecture encoding strategy (Section 3.2), N individuals (or architectures) are initialized as the initial population, and each individual in the population is trained and tested to obtain the individual's fitness value (Section 3.4), that is, the classification accuracy and the number of spikes. The classification accuracy directly reflects the performance of the SNN. The number of spikes is the combined firing spikes of all LIF neurons in the SNN, which corresponds to the power consumption of the SNN. Subsequently, the evaluated architecture information, including the architecture encoding and the corresponding fitness value of the architecture, is saved in $population$, and the population generations are updated simultaneously.

(2) **Offspring generation** (lines 8–9): When the termination condition (the current evolutionary generation G_{cur} is less than the total evolutionary generation G) is not satisfied, the offspring individuals are generated according to the proposed crossover and mutation operators (Section 3.3). This process is implemented using variable-length encoding based on the architecture.

(3) **Fitness evaluation** (lines 10–17): Each individual is trained to obtain the ground-truth performance, and a performance predictor is introduced to assist the evaluation process (Section 3.6). After the fitness value of the offspring individual is obtained, the architectural information is saved in $population$.

(4) **Environment selection** (lines 18–31): The NSGA-II algorithm is applied to perform non-dominated sorting on individuals to divide the population into multiple levels according to the dominance relationship, and calculate the crowding distance between individuals in each level. Then, individuals are selected in hierarchical order and in descending order of crowding distance until the predetermined population size is met, thus constructing the next generation population. The above evolutionary process is repeated until the termination condition is satisfied, and finally, an optimal SNN architecture that balances higher performance and lower power consumption is obtained.

Algorithm 1 PESNN.

```
Input: $G$: Evolutionary generations; $N$: Population size; $G_{gen}$: Evolutionary generations when using predictor.
Output: $O'_b$: Optimal SNN architecture.
1: $G_{cur} \leftarrow 0$, $population \leftarrow \{\emptyset\}$;
   // $G_{cur}$ denotes the current generation; $population$ is used to store individuals.
2: while the size of $population < N$ do
3:     $indi$.arch ← Initialize individual;
4:     $indi$.fitness ← Fitness evaluation;
5:     Add $indi$ to $population$;
6: end while
7: $G_{cur} \leftarrow G_{cur} + 1$;
8: while $G_{cur} < G$ do
9:     new individuals ← Generate offspring;
       // Using the crossover and mutation operators.
10:    for each $indi$ in new individuals do
       // Individuals not evaluated.
11:        if $G_{cur} < G_{gen}$ then
12:            $indi$.fitness ← Training;
13:        else
14:            $indi$.fitness ← Predictor-assisted evaluation;
15:        end if
16:        Add $indi$ to $population$;
             // Individuals have been evaluated.
17:    end for
18:    $next\_gen \leftarrow \emptyset$;
19:    $fronts$ ← Non-dominated sorting;
       // Perform non-dominated sorting on the $population$ and divide individuals into different layers.
20:    while size of $next\_gen < N$ do
21:        for each front $F$ in $fronts$ do
22:            $remaining \leftarrow N$ - size of $next\_gen$;
                 // Calculate the remaining space to accommodate the next generation of individuals.
23:            if size of $F \leq remaining$ then
24:                $next\_gen \leftarrow next\_gen \cup F$;
25:            else
26:                Sort $F$ in descending order by crowding distance;
27:                $next\_gen$ ← top $remaining$ individuals from $F$;
                     // Add individuals with high crowding distance to the next generation.
28:            end if
29:        end for
30:    end while
31:    $population \leftarrow next\_gen$;
32:    $G_{cur} \leftarrow G_{cur} + 1$;
33: end while
34: return $O'_b$;
   // The optimal individual achieves a good balance between performance and power consumption.
```

3.2. Search space and encoding

Inspired by the great success of ResNet in the field of deep learning, the network architecture in the search space mainly consists of a stack of basic architecture blocks. The blocks are represented as yellow rectangles with dashed boxes in Fig. 2, where each block is composed of two two-dimensional 3 ×3 convolutional layers (Conv), two batch normalization (BN) layers, two LIF neurons, and one skip connection. Since SNN conveys information through binary signals, it is different from the basic block in ResNet. Here, we use LIF neurons. The number of blocks N_{block} and the output channels $N_{channel}$ of the blocks are randomly selected

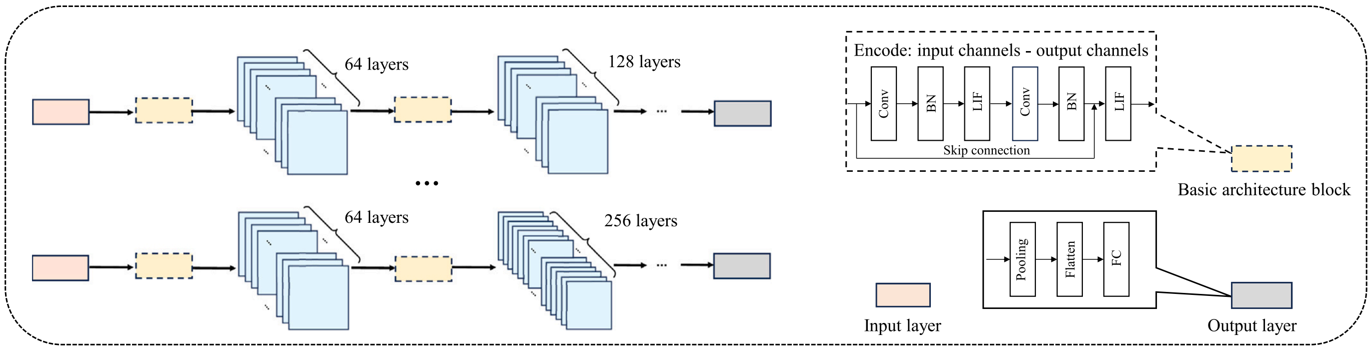


Fig. 2. Search space. The network architecture in the search space consists of three parts: input layer, searched basic architecture block, and output layer. The number of light blue rectangles represents the output channels of the basic architecture.

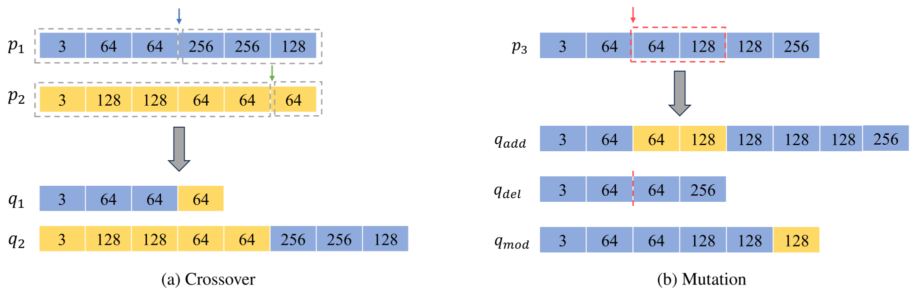

p_1 3 64 64 256 256 128
p_2 3 128 128 64 64 64
q_1 3 64 64 64
q_2 3 128 128 64 64 256 256 128
(a) Crossover
p_3 3 64 64 128 128 256
q_{add} 3 64 64 128 128 128 128 256
q_{del} 3 64 64 256
q_{mod} 3 64 64 128 128 128
(b) Mutation

Fig. 3. Generate offspring individuals using crossover and mutation operators. Each combination of two numbers represents the coding of a basic architecture block. Blue and green arrows represent crossover points, and red denotes the mutation point.

at the stage of initialization. At the same time, the constraints need to be met: $N_{block} \in [5,15]$, $N_{channel} \in [64,128, 256]$. The encoding of the architecture is decided by the number of basic architecture blocks as well as the output channels, which are dynamically changing. A variable-length encoding strategy can effectively avoid the pre-specified length required in a fixed encoding. For example, for the first individual in Fig. 2, when a three-channel image is input into the network, the input channel of the first searchable basic architecture block is three, and the output channel is 64, so the block can be encoded as "3–64". The output channel of the first basic architecture block is also used as the input channel of the second block. When the output channel of the second basic architecture block is 128, it can be encoded as "64–128". Combining the encodings can yield the encoding of the first two basic architecture blocks "3–64-64–128". The encodings of subsequent basic architecture blocks are similar. Finally, the individual encoding can be obtained.

Using a Poisson encoder to convert input images to spike values for input into the network significantly increases training and inference costs [57,58]. Similar to recent studies, we use direct encoding [59] to input decimal data into the network without converting it to spike values, which significantly reduces the timesteps required for information representation. As shown in Fig. 2, the output layer of the network architecture is composed of an average pooling layer, a flattened layer, and a fully connected layer (FC). SNNs are different from CNNs in that not only is spatial-domain information required, but also time-domain information needs to be considered. The classification task can finally be accomplished by continuously accumulating the network architecture output at T timesteps.

3.3. Offspring generation

Assume that the individuals p_1, p_2, and p_3 are composed of three basic architecture blocks, with a total of 6-bit encoding. The crossover operation involves choosing a position as the crossover point in the parent individuals p_1 and p_2, respectively, as shown by the blue arrow and the green arrow in Fig. 3(a). According to the crossover point, the parent individuals are divided into two parts, and the two child individuals, q_1 and q_2, can be obtained by exchanging the right parts of two parent individuals.

Mutation operations mainly include adding basic architecture blocks (probability 0.8), deleting blocks (probability 0.1), and changing the number of output channels of an architecture block (probability 0.1): (1) After the first basic architecture block of the parent individual p_3 (red arrow in Fig. 3(b)), a new block with output channel of 128 is added. The output channel 64 of the first block is used as the input channel of the new block. At the same time, the input channel of the second block of p_3 is updated to 128. (2) Deleting the basic architecture block is similar to adding a basic architecture block. For example, after the second block of the parent individual p_3 (red dashed box in Fig. 3(b)) is deleted, the encoding of the following basic architecture block is updated from "128–256" to "64–256". (3) The parent individual p_3 changes output channels of the last basic architecture block from 256 to 128, which generates the child individual.

3.4. Fitness evaluation

In genetic algorithms, fitness evaluation is a critical step in measuring the quality of individuals, which facilitates guiding the evolution process based on the individual evaluation results. The pseudo-code of the fitness evaluation algorithm is shown in Algorithm 2. Given the individual O_i to be evaluated (the network architecture after decoding based on string encoding), the datasets D_{train} and D_{test} for training and testing, and the number of training iterations $epochs$. Initialize the maximum classification accuracy to zero. The network architecture is trained by accumulating spike transmission and membrane potential updates for multiple timesteps on the training set D_{train}. Loss calculation and weight updates are performed at each timestep to optimize

Algorithm 2 Fitness evaluation.

```
Input: $O_i$: Unevaluated individual; $epochs$: Number of training iterations; $D_{train}$: Training set; $D_{test}$: Test set.
Output: $O_i'$: Evaluated Individual.
1: $\max\_acc \leftarrow 0$;
2: while $epochs$ not fulfilled do
3:     $O_i.spikes \leftarrow$ Individuals are trained on $D_{train}$ to obtain the number of spikes;
4:     $O_i.acc \leftarrow$ Calculate classification accuracy on $D_{test}$;
5:     if $O_i.acc > \max\_acc$ then
6:         $\max\_acc \leftarrow O_i.acc$;
7:     end if
8: end while
9: return $O_i'$;
   // Using $O_i.spikes$ and $\max\_acc$ as individual $O_i$ fitness values.
```

network performance gradually. In addition, the classification accuracy of the individual is calculated on the dataset D_{test}, and the maximum classification accuracy is continuously updated. Finally, the individual with the best fitness is output.

3.5. Multi-objective optimization

Most methods that use NAS to search for SNN architectures only consider one performance metric or use multi-objective optimization, but may tend to optimize one objective while ignoring the other. For example, the AutoSNN method [23] designs a scalar fitness function that combines accuracy and the number of spikes, where the term for the number of spikes is adjusted by an exponential decay parameter. However, this scalarization method actually simplifies the multi-objective optimization problem into a single-objective optimization problem and relies on predefined fixed weights. When the weight design is unreasonable, the optimization algorithm will first optimize the objective with a larger weight and ignore the other objective, making it difficult to dynamically adjust the conflicting relationship between the objectives. In PESNN, the conflicting relationship between goals is reflected in the fact that the improvement of accuracy is accompanied by complex neuronal activities, resulting in excessive spike emission, while a lower number of spikes usually affects the accuracy. The goal of PESNN is to find an SNN architecture with higher classification accuracy and fewer spikes by dynamically balancing the conflicting relationships, which can be naturally expressed as a multi-objective optimization problem:

$$\underset{\alpha\in\Omega}{\operatorname{argmin}} F(\alpha) = \left\{ f_{error}(\alpha), f_{spikes}(\alpha) \right\}, \tag{4}$$

where Ω is the search space containing all possible solutions (or architectures) α. $f_{error}(\alpha)$ represents the classification error of architecture α, which is obtained by training the architecture on D_{train} and then validating it on D_{test}. $f_{spikes}(\alpha)$ represents the number of spikes fired by the neurons in the architecture α during the inference phase. This formula describes the process of finding an architecture α that minimizes the classification error (*i.e.*, maximizes the classification accuracy) in the objective function $F(\alpha)$ and decreases the number of spikes.

PESNN employs a trade-off strategy to resolve conflicts between objectives in each iteration. When classification accuracy and spike reduction are not aligned, it considers both factors comprehensively to ensure that the search does not overly favor one objective. Specifically, it utilizes the non-dominated sorting and crowding distance mechanisms of NSGA-II to select excellent individuals that achieve a good balance between classification accuracy and spikes (Lines 18 to 31 of Algorithm 1). First, non-dominated sorting is applied to classify individuals into different layers. An individual is considered non-dominated if no other individual outperforms it in both classification accuracy and spikes, and it has at least one objective superior to another individual. The first Pareto front consists of all non-dominated individuals. These individuals are then removed, and the process is repeated to form subsequent fronts until all individuals are sorted. However, selecting individuals solely based on non-dominated sorting may cause population to gather in a certain area, limiting evolutionary search diversity and leading to local optima. To mitigate this, crowding distance is introduced to measure the distribution of individuals. The crowding distance of individuals at the same level is calculated, that is, they are sorted according to the classification accuracy and the number of spikes, and the normalized distance between adjacent individuals on the target is calculated. The larger the crowding distance, the fewer solutions there are around the individual, which helps maintain population diversity. By selecting individuals with high crowding distances, PESNN can explore the possibilities of different architectures while avoiding falling into local optima. Through iterative evolution, PESNN generates a set of architectures that balance higher accuracy and lower spikes. Users can choose the final architecture according to their needs, such as choosing the one with the lowest spikes. Here, PESNN selects the individual with the highest classification accuracy as the final architecture.

3.6. Performance predictor

The performance evaluation phase in the evolutionary search process requires more computational resources. To reduce the computational cost, we introduce a performance predictor that can predict the performance of candidate architectures within seconds to accelerate the evolutionary search process. Regarding the training of the performance predictor, it is primarily divided into the initial stage in the early evolution and the online training stage in the later evolution. In the initial stage, instead of randomly generating some individuals to be trained and then used as the initial training samples of the predictor, we use the individuals that have evolved through G_{gen} generations as the initial training samples to ensure that the training samples of the performance predictor contain some high-performance individuals. In the previous process, we completed the fitness evaluation of all the individuals in previous G_{gen} generations (lines 1–12 of Algorithm 1, line 16) and saved their architectural information (including architectural encoding and corresponding fitness). In order to improve the generalization ability of the performance predictor, we save the architecture information after removing duplicate information to *Archive* for training the initial performance predictor. After comparing different prediction models, we chose the random forest (RF) model owing to its high prediction accuracy and reliability.

In the later stage of evolution, the predictor is trained online. Specifically, after G_{gen} generations, the initial performance predictor is first used to predict the classification accuracy of individuals in generation G_{gen+1}. Then, the elite selection strategy selects the top 5 individuals to be trained. These individuals are merged with the parent individuals to enter the environment selection phase. Thus, all network architectures are avoided being trained, and the evolution efficiency is improved. At the same time, the trained architecture information is updated in the *Archive*. This is because the initial performance predictor is trained based on a small number of individuals in the first G_{gen} generations and may not be able to predict the performance of each network architecture in the G_{gen+1} generation accurately. Before the next prediction, the predictor will be trained using the updated *Archive*. By increasing the training samples of the predictor generation by generation, the architecture information in the *Archive* increases, and the prediction becomes more accurate. Although the process of obtaining predictor training samples requires a certain amount of computing resources, its computational overhead is acceptable compared to the entire evolutionary search process.

The orange part of Fig. 5 shows the training process of the predictor. The *KTau* coefficient measures the consistency between the prediction

Table 1
Experimental parameters.

Parameter	Value	Description
G	20	Total evolutionary generations
N	20	Population size
P_c	0.9	Probability of using the crossover operator
P_m	0.2	Probability of using the mutation operator
$epochs$	300	Number of rounds of training architecture
lr	0.001	Learning rate
bs	128	Batch size
τ	1.1	Membrane potential time constant
U_{th}	1.0	Membrane potential threshold
G_{gen}	5	Evolutionary generations when using predictor

and the ground-truth performance. The larger the value, the higher the consistency. Conversely, the prediction error is larger. For example, when the initial predictor predicts the sixth-generation individuals, the *KTau* is slightly lower, indicating that the predictor exhibits some bias in its early predictions. However, this limitation diminishes as evolution progresses.

4. Experiments

4.1. Experimental settings

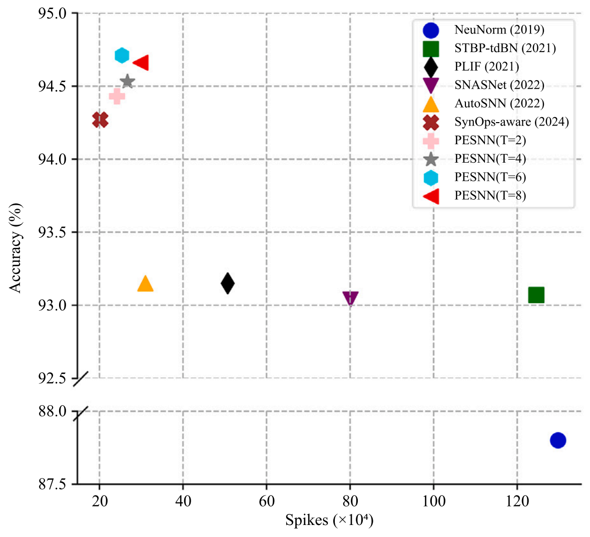
We conducted experiments on the static datasets CIFAR-10 and CIFAR-100 and the neuromorphic dataset DVS-CIFAR-10 to verify the effectiveness of PESNN. CIFAR-10 consists of 60,000 32 × 32 pixel color images, 50,000 of which are used for training and 10,000 for testing. Images in the CIFAR-10 are divided into ten categories, including airplanes, cars, birds, cats, and more. Each category has 6000 images. CIFAR-100 is similar to CIFAR-10 and also consists of 60,000 32 × 32 pixel color images, but it is divided into 100 classes. In each class, 500 images and 100 images are used for training and testing, respectively. CIFAR-10 and CIFAR-100 use the same processing methods, including normalization, random cropping with 4-pixel padding, random masking, and random horizontal flipping. DVS-CIFAR-10 is a dataset that converts CIFAR-10 frame-based images into 128 × 128 pixel event streams through a dynamic vision sensor (DVS). The dataset contains a total of 10,000 event streams, 9000 of which are used as training sets, and the remaining 1000 are used as test sets. Since the data are based on event streams and the network model needs to handle sparsity and temporal information, this dataset is more suitable for studying SNNs. The experiments were conducted on a single NVIDIA GeForce RTX 4090 using the PyTorch framework, and the SNN architecture was trained using the Adam optimizer. The parameters involved in the experiment are shown in Table 1. PESNN evolved for 20 generations, each generation containing 20 individuals. We observed that a higher mutation rate ($P_m > 0.2$) would cause the architecture to change too drastically, affecting the stability and convergence speed of the model, while a lower mutation rate ($P_m < 0.2$) would limit the search space, thereby reducing the probability of finding the optimal architecture. Therefore, we chose $P_m = 0.2$ as a compromise in the experiment. Additionally, when using the predictor, we set $G_{gen} = 5$, introducing the predictor after the 5th generation to reduce computational costs. Other key training parameters include P_c, $epochs$, learning rate and batch size. The membrane potential time constant is set to $\tau = 1.1$, and the membrane potential threshold is set to $U_{th} = 1.0$ to optimize the dynamic characteristics of the SNN.

4.2. Results on CIFAR-10

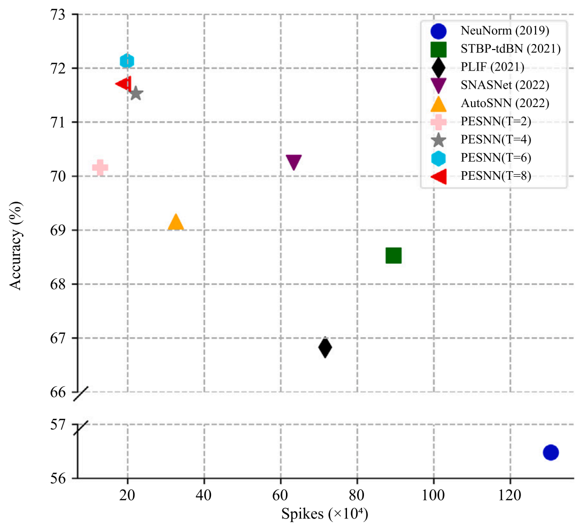
The experimental results of our method PESNN and the comparison methods on CIFAR-10 are shown in Table 2 and Fig. 4(a). From Fig. 4(a), it can be seen that PESNN outperforms most models. In Table 2, the first column in the table represents the relevant methods, and the second column is the training method of SNN architecture, including the A2S method and the surrogate gradient method. The third column represents the SNN architecture type. The table shows the classification accuracy and number of spikes of SNN architecture under different timesteps (T). The data of other algorithms are from the original papers or reproduced using the corresponding open-source code. As shown in Table 2, the classification accuracy of the SNN architecture searched by PESNN is higher than that of the manually designed SNNs. Specifically, PESNN has an accuracy higher than the methods with A2S by 0.46 %–4.49 % when the timestep of PESNN is set to 2. When the timestep of PESNN is set to 6, the accuracy is higher by 0.74 %–4.77 % compared with the other algorithms with A2S. In addition, the methods trained using A2S require a large number of timesteps, which undoubtedly results in high computational and inference latency. On the other hand, compared with the manual design method based on SG, for example, when the timestep is 8, the classification accuracy of the SNN architecture searched by PESNN

Table 2
Experimental results on CIFAR-10 and comparison with other related methods. † indicates that the author did not provide a specific method name. For the convenience of comparison, we summarized the method names based on the author's core contributions, such as optimal conversion pipeline (OCP) and optimized membrane potential initialization (OMPI).

Method	Train	Architecture	T	Accuracy (%)	Spikes (K)
DCT-SNN (2021) [62]	A2S	VGG9	48	89.94	–
Light Pipeline (2021) [63]	A2S	VGG16	32	91.82	–
OCP (2021) [64] †	A2S	ResNet20	16	92.41	–
OMPI (2022) [65] †	A2S	ResNet20	64	92.57	–
Adaptive threshold (2022) [66]	A2S	VGG19	400	93.97	–
QCFS (2022) [67]	A2S	ResNet20	32	92.24	–
Fast-SNN (2023) [68]	A2S	AlexNet	7	92.53	–
NeuNorm (2019) [60]	SG	5 Conv, 2 FC	8	87.80	1298
STBP-tdBN (2021) [17]	SG	ResNet19	8	93.07	1246
PLIF (2021) [8]	SG	6 Conv, 2 FC	8	93.15	507
TOIB (2023) [48]	SG	VGG9	–	91.30	–
OTTT (2024) [69]	SG	VGG	6	93.58	–
SNASNet (2022) [22]	SG	NAS	5	93.04	801
AutoSNN (2022) [23]	SG	NAS	8	93.15	310
SynOps-aware (2024) [61]	SG	NAS	3	94.27	202
MixedSNN (2024) [70]	SG	NAS	4	90.85	–
PESNN (Ours)	SG	NAS	2	94.43	242
PESNN (Ours)	SG	NAS	4	94.53	267
PESNN (Ours)	SG	NAS	6	94.71	254
PESNN (Ours)	SG	NAS	8	94.66	298



(a) CIFAR-10



(b) CIFAR-100

Fig. 4. Comparison visualization of PESNN with other methods on CIFAR-10 and CIFAR-100 that have an indicator of the number of spikes.

Table 3
Experimental results on CIFAR-100 and comparison with other related methods.

Method	Train	Architecture	T	Accuracy (%)	Spikes (K)
DCT-SNN (2021) [62]	A2S	VGG11	48	68.30	–
Light Pipeline (2021) [63]	A2S	VGG16	32	64.53	–
OCP (2021) [64] †	A2S	ResNet20	32	68.40	–
OMPI (2022) [65] †	A2S	ResNet20	32	67.18	–
Adaptive threshold (2022) [66]	A2S	VGG19	1100	73.58	–
QCFS (2022) [67]	A2S	ResNet20	32	68.12	–
NeuNorm (2019) [60]	SG	5 Conv, 2 FC	8	56.48	1306
STBP-tdBN (2021) [17]	SG	ResNet19	8	68.53	895
PLIF (2021) [8]	SG	6 Conv, 2 FC	8	66.83	716
TOIB (2023) [48]	SG	VGG11	–	66.80	–
OTTT (2024) [69]	SG	VGG	6	71.11	–
SNASNet (2022) [22]	SG	NAS	5	70.25	634
AutoSNN (2022) [23]	SG	NAS	8	69.16	326
MixedSNN (2024) [70]	SG	NAS	6	64.45	–
PESNN (Ours)	SG	NAS	2	70.16	128
PESNN (Ours)	SG	NAS	4	71.53	221
PESNN (Ours)	SG	NAS	6	72.13	198
PESNN (Ours)	SG	NAS	8	71.71	188

is 94.66 %, which is higher than NeuNorm which is manually designed architecture with SG [60]. It is also 1.59 % and 1.51 % higher than the SNN architectures designed by Zheng et al. [17] and Fang et al. [8], respectively. Meanwhile, these two methods produce about 2–4 times more spikes than the PESNN method. In addition to the manual design methods, we also list some typical automatic design SNN methods, which use SG to train the network architecture. Kim et al. [22] and Na et al. [23] applied NAS to SNNs and proposed SNASNet and AutoSNN methods, respectively, which are still not as effective as PESNN in terms of performance and power consumption. We conducted experiments at different timesteps and found that the performance of the network in the case of a timestep of 8 was not as good as the network with a timestep of 6, which indicates that PESNN has searched for a suboptimal architecture when the timestep is 8. In terms of the number of spikes, the spikes in the searched architectures appear to be fewer compared to most other methods. For example, the number of spikes is only 242 K when $T=2$, close to the 202 K of the SynOps-aware approach [61] at $T=3$. Although the architecture generates a slightly higher number of spikes at multiple timesteps, this discrepancy demonstrates an acceptable trade-off between spikes and the accuracy of the PESNN.

4.3. Results on CIFAR-100

The experimental results of PESNN on CIFAR-100 are shown in Table 3 and Fig. 4(b). The figure shows that PESNN has higher classification accuracy and fewer spikes than other SG-based methods. As presented in Table 3, the classification accuracy of the methods with A2S ranges from 64.53 % to 73.58 %. Although the adaptive threshold method proposed by Chen et al. [66] outperforms PESNN on CIFAR-100 in terms of accuracy, it needs 1100 timesteps to achieve similar results as PESNN with at most 8 timesteps. PESNN is generally more accurate than other SG-based methods with the same or fewer timesteps. When the timestep is 8, the classification accuracy of PESNN is 71.71 %, which is 3.18 % higher than the fixed SNN architecture manually designed by Zheng et al. [17]. Meanwhile, it is 15.23 % and 4.88 % higher than the fixed architecture searched by NeuNorm method [60] and the fixed architecture searched by PLIF method [8], respectively. It is also 2.55 % higher than the AutoSNN method [23]. When the timestep is 6, the performance is significantly better than the MixedSNN method proposed by Xie et al. [70]. We conducted experiments with timestep settings of 2, 4, 6, and 8 to further analyze the effect of timestep on the algorithm. It can be seen that the classification accuracy increases

Table 4
Experimental results on DVS-CIFAR-10 and comparison with other related methods.

Method	Train	Architecture	T	Accuracy (%)	Spikes (K)
ASF-BP (2021) [71]	SG	VGG7	12	62.50	–
STBP-tdBN (2021) [17]	SG	ResNet19	8	66.10	1550
PLIF (2021) [8]	SG	7-layer	8	69.10	4521
Tandem learning (2023) [72]	SG	CifarNet	20	65.59	–
AutoSNN (2022) [23]	SG	NAS	20	72.50	1269
PESNN (Ours)	SG	NAS	8	73.10	772
PESNN (Ours)	SG	NAS	10	72.88	799

slightly as the timestep increases. For example, the classification accuracy is 70.16 % when $T=2$ and reaches a maximum of 72.13 % when $T=6$. As the timestep is moderately increased, the network can propagate information between more neurons, and may capture more feature information to improve classification accuracy. Once the optimal network architecture is found in the case of the timestep being 6, if the timestep is increased further to 8, the accuracy decreases slightly because of redundant calculations, and the system searches for a suboptimal solution. No matter at a small timestep (e.g., $T=2$) or a big timestep (e.g., $T=8$), PESNN can maintain an accuracy rate above 70 %, which demonstrates that it has excellent adaptability and stability. In terms of the number of spikes, PESNN can achieve better classification results with a relatively small number of spikes, indicating that PESNN reduces energy consumption and can provide advantages for power-sensitive application scenarios, such as embedded devices and edge computing.

4.4. Results on DVS-CIFAR-10

We list the results of training SNNs using SG on DVS-CIFAR-10 in Table 4. It can be observed that PESNN achieves the highest classification accuracy of 73.10 % when the timestep is 8. For the other methods, the AutoSNN method achieves 72.50 %. It is the second-best result in accuracy, but it requires 20 timesteps [23]. In contrast, our method can achieve high classification accuracy using fewer timesteps, demonstrating the efficiency of the architecture in a shorter inference time, which is especially significant for dealing with real-time tasks. The number of spikes is an essential indicator for measuring the power consumption of SNNs. The spikes for PESNN at $T=8$ and $T=10$ are 772 K and 799 K, respectively, which is dramatically less than the 4521 K of PLIF [8]. This is about 6 times more than PESNN. In addition, the spikes of the SNN architecture searched by STBP-tdBN are 1550 K [17]. It is about twice as many as PESNN. This shows that PESNN is able to considerably reduce the number of spikes, thus reducing power consumption while maintaining higher performance. PESNN successfully finds the optimal balanced solution between classification accuracy and the number of spikes at a timestep of 8 by the NSGA-II multi-objective optimization method. Compared to other methods, PESNN is capable of adaptively selecting features that are conducive to reducing power consumption and improving performance. Overall, PESNN effectively improves the efficiency and adaptability of SNN architectures using multi-objective optimization in the ENAS framework.

4.5. Ablation study

Comparison of predictors: The performance predictor is an important component of PESNN. The predictor is trained using empirical data and is used to predict the accuracy of architectures. In order to select a more accurate predictor, we tested three prediction models: random forest (RF), K-nearest neighbor (KNN), and support vector regression (SVR). We conducted three sets of experiments under two timesteps, and the CIFAR-10 dataset was used to verify the prediction performance of different models. During the evolutionary search process, each model is trained using the same data. We use Kendall's tau coefficient (*KTau*) as an indicator, which measures the correlation between the true ranking and the predicted ranking of the architecture. Fig. 5 shows the change in *KTau* per generation for each prediction model. It can be seen from the figure that the *KTau* value for each model trends upwards with the increase in the number of evolutionary generations, which indicates that the training samples are increasing from generation to generation, making the predictions of the predictor more accurate. Although the *KTau* values of SVR and KNN show an overall upward trend, they are not as high or stable as RF. Among them, the performance of SVR fluctuates more, and the robustness of the prediction accuracy is slightly inferior to that of RF, which may be due to its sensitivity to changes in the training data. KNN's performance is significantly lower than that of RF and SVR, which may be because it has difficulty processing the high-dimensional features of SNN-encoded data and cannot capture complex patterns in the data. Therefore, we chose the RF model as the predictor, which can maintain high accuracy and relatively stable prediction performance.

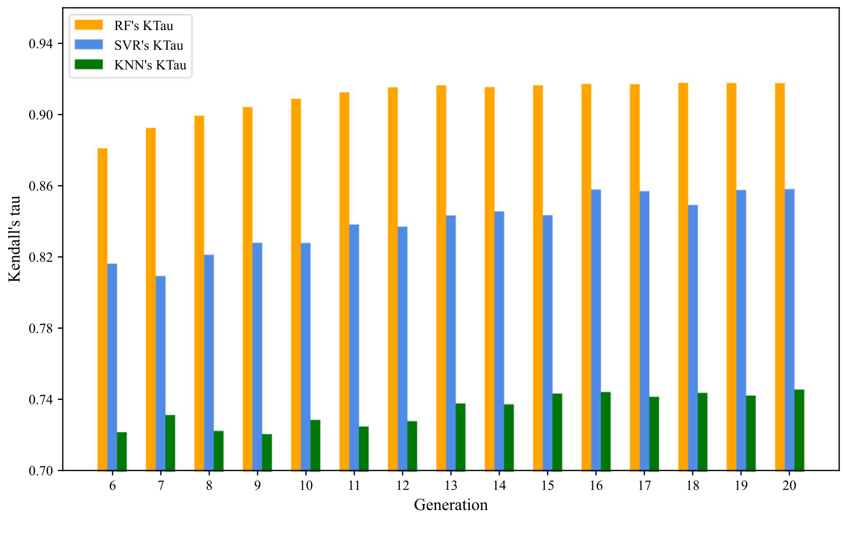


Fig. 5. Visualization of *KTau* changes of different prediction models during evolution.

Table 5
Performance comparison of the RF prediction model with different hyperparameter settings. The first column represents different G_{gen} values, while the next three columns record the average indicators of the RF model after training ten times and predicting the next generation of individuals. The last two columns represent the performance of the architecture searched during the evolution process.

G_{gen}	Training time (s)	Prediction time (s)	*KTau*	Accuracy (%)	Spikes (K)
3	0.8864	0.1354	0.7621	92.498	152
5	0.7938	0.1259	0.8877	94.432	224
7	1.1355	0.1301	0.8944	94.428	255

Selection of hyperparameters: The individuals in the pre-G_{gen} generations are evaluated and used as training samples for the initial performance predictor. The appropriate G_{gen} can effectively balance the search quality and efficiency. We set different G_{gen} values to train the initial RF prediction model under the same conditions and predict the classification accuracy of the individuals in the following generation. We also trained the predicted SNN architectures to obtain ground-truth accuracy. The Table 5 shows that the training and prediction time of the prediction model reaches the second level. This indicates that the predictor can quickly estimate the performance of architectures, and when combined with the elitism strategy to assist the performance evaluation process, it significantly accelerates the SNN architecture search. Fig. 6 illustrates the variation of the *KTau* coefficient for different G_{gen} during the evolutionary process. When G_{gen} is 3, its *KTau* value is lower than those of $G_{gen} = 5$ and $G_{gen} = 7$. Combined with the *KTau* value (0.7621) when $G_{gen} = 3$ in Table 5, it can be seen that the initial predictor has low prediction ability, which leads to misjudgment of the performance of the architecture, which in turn affects the architecture selection of subsequent generations. Although the *KTau* value tends to stabilize with the increase of generations, due to the insufficient initial prediction ability, the performance of the architecture searched under the $G_{gen} = 3$ setting is poor. Fig. 6 also shows the *KTau* curve for $G_{gen} = 5$ is close to that of $G_{gen} = 7$. Additionally, Table 5 indicates that when G_{gen} is 5, the *KTau* value reaches 0.8877, which is comparable to that of $G_{gen} = 7$. This suggests that both settings provide relatively accurate predictions in the early training stages, effectively guiding the evolutionary process. Although the final architectures found under $G_{gen} = 5$ and $G_{gen} = 7$ exhibit similar performance, setting to 7 requires evaluating approximately 20 % more network architectures, reducing search efficiency. Therefore, we set G_{gen} to 5.

5. Conclusion

This paper proposes a method for the evolutionary search of SNN architectures to solve the problems existing in manually designing SNN architectures. A real-time performance predictor is designed to assist the time-consuming performance evaluation stage. Moreover, the classification accuracy and the number of spikes are used as two objectives. The multi-objective evolutionary algorithm has been introduced to search for high-performance and low-power SNN architectures. To verify the effectiveness of PESNN, we conducted a series of experiments on three datasets: CIFAR-10, CIFAR-100, and DVS-CIFAR-10. The experimental results demonstrate that the SNN architecture searched by PESNN outperforms other advanced methods.

In this work, although we provide new ideas for the evolutionary search of SNN architectures, some problems still exist. The proposed predictor can speed up the search process, but it only focuses on the classification accuracy indicator. Furthermore, the single-objective predictor may not be better adapted to the multi-objective algorithm. In the future, we will explore a multi-objective predictor to speed up the evolutionary search process. Meanwhile, due to the event-driven nature and spike communication mechanism of SNNs, their excellent performance is difficult to fully realize on GPUs. We plan to apply PESNN to neuromorphic chips, which can more efficiently process the event-driven calculations, thereby better leveraging the low power consumption and high parallel processing advantages of SNNs.

CRediT authorship contribution statement

Yuting Wang: Visualization, Methodology, Software, Investigation, Validation, Writing - original draft. **Yu Xue:** Writing - review & editing, Project administration, Funding acquisition, Supervision. **Wei Ding:** Writing - review & editing. **Yiyu Tan:** Writing - review & editing. **Peng**

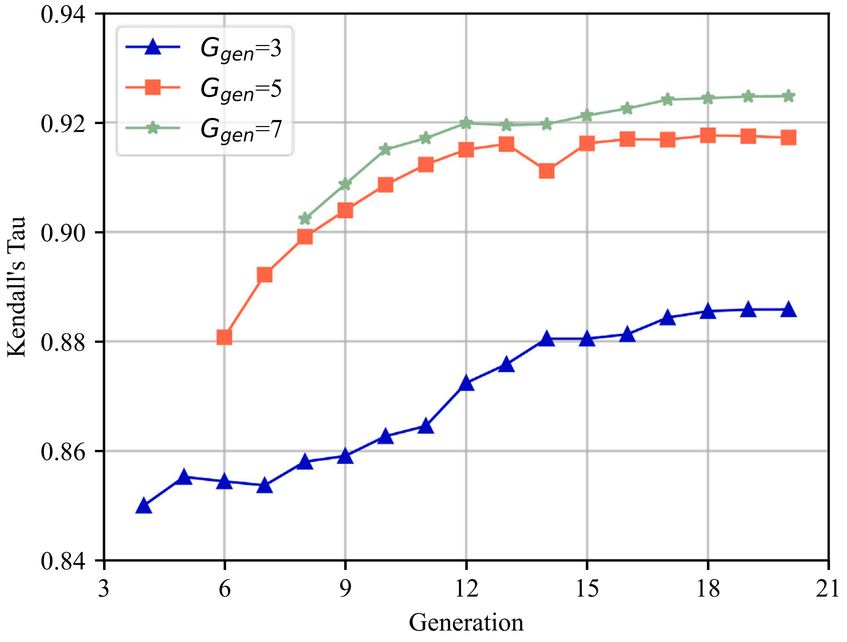


Fig. 6. Changes in *KTau* coefficient under different G_{gen} during evolution.

Chen: Writing - review & editing. **Mohamed Wahib:** Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (Nos. 62376127, 61876089, 61876185, 61902281, 61403206), by Natural Science Foundation of Jiangsu Province (No. BK20141005), by Natural Science Foundation of the Jiangsu Higher Education Institutions of China (No. 14KJB520025), and partially supported by the project of Distinguished Professors of Jiangsu Province.

Data availability

Data will be made available upon request.

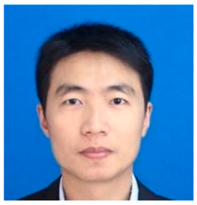
References

[1] P. Ganesan, S.K. Jagatheesaperumal, M.M. Hassan, F. Pupo, G. Fortino, Few-shot image classification using graph neural network with fine-grained feature descriptors, Neurocomputing 610 (2024) 128448.
[2] X. Zhang, Z. Xiao, X. Wu, Y. Chen, J. Zhao, Y. Hu, J. Liu, Pyramid pixel context adaption network for medical image classification with supervised contrastive learning, IEEE Trans. Neural Netw. Learn. Syst. (2024) 1–14.
[3] J. Yin, J. Shen, X. Gao, D.J. Crandall, R. Yang, Graph neural network and spatiotemporal transformer attention for 3D video object detection from point clouds, IEEE Trans. Pattern Anal. Mach. Intell. 45 (8) (2023) 9822–9835.
[4] S.-C. Huang, Q.-V. Hoang, T.-H. Le, SFA-Net: a selective features absorption network for object detection in rainy weather conditions, IEEE Trans. Neural Netw. Learn. Syst. 34 (8) (2023) 5122–5132.
[5] Q. Chen, Z. Zhang, Y. Lu, K. Fu, Q. Zhao, 3-D convolutional neural networks for RGB-D salient object detection and beyond, IEEE Trans. Neural Netw. Learn. Syst. 35 (3) (2024) 4309–4323.
[6] R. Vohra, F. Senjaliya, M. Cote, A. Dash, A.B. Albu, J. Chawarski, S. Pearce, K. Ersahin, Detecting underwater discrete scatterers in echograms with deep learning-based semantic segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2023, pp. 375–384.
[7] Y. Liu, J. Han, X. Yao, S. Khan, H. Cholakkal, R.M. Anwer, N. Liu, F.S. Khan, Bidirectional reciprocative information communication for few-shot semantic segmentation, in: International Conference on Machine Learning, ICML, 2024.
[8] W. Fang, Z. Yu, Y. Chen, T. Masquelier, T. Huang, Y. Tian, Incorporating learnable membrane time constant to enhance learning of spiking neural networks, in: Proceedings of the IEEE International Conference on Computer Vision, ICCV, 2021, pp. 2661–2671.
[9] W. Maass, Networks of spiking neurons: the third generation of neural network models, Neural Netw. 10 (9) (1997) 1659–1671.
[10] Z. Yi, J. Lian, Q. Liu, H. Zhu, D. Liang, J. Liu, Learning rules in spiking neural networks: a survey, Neurocomputing 531 (2023) 163–179.
[11] J. Shen, W. Ni, Q. Xu, H. Tang, Efficient spiking neural networks with sparse selective activation for continual learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, AAAI, vol. 38, 2024, pp. 611–619.
[12] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: International Conference on Learning Representations, ICLR, 2015.
[13] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2016, pp. 770–778.
[14] A. Vaswani, Attention is all you need, Adv. Neural Inf. Process. Syst. (2017).
[15] B. Han, G. Srinivasan, K. Roy, RMP-SNN: residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2020, pp. 13558–13567.
[16] W. Fang, Z. Yu, Y. Chen, T. Huang, T. Masquelier, Y. Tian, Deep residual learning in spiking neural networks, Adv. Neural Inf. Process. Syst. 34 (2021) 21056–21069.
[17] H. Zheng, Y. Wu, L. Deng, Y. Hu, G. Li, Going deeper with directly-trained larger spiking neural networks, in: Proceedings of the AAAI Conference on Artificial Intelligence, AAAI, vol. 35, 2021, pp. 11062–11070.
[18] L. Feng, Q. Liu, H. Tang, D. Ma, G. Pan, Multi-level firing with spiking DS-ResNet: enabling better and deeper directly-trained spiking neural networks, in: Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI, 2022, pp. 2471–2477.
[19] P. Liao, Y. Jin, W. Du, EMT-NAS: transferring architectural knowledge between tasks from different datasets, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2023, pp. 3643–3653.
[20] J. Huang, B. Xue, Y. Sun, M. Zhang, G.G. Yen, Particle swarm optimization for compact neural architecture search for image classification, IEEE Trans. Evol. Comput. 27 (5) (2022) 1298–1312.
[21] G. Yuan, B. Wang, B. Xue, M. Zhang, Particle swarm optimization for efficiently evolving deep convolutional neural networks using an autoencoder-based encoding strategy, IEEE Trans. Evol. Comput. 28 (5) (2024) 1190–1204.
[22] Y. Kim, Y. Li, H. Park, Y. Venkatesha, P. Panda, Neural architecture search for spiking neural networks, in: European Conference on Computer Vision, ECCV, 2022, pp. 36–56.
[23] B. Na, J. Mok, S. Park, D. Lee, H. Choe, S. Yoon, AutoSNN: towards energy-efficient spiking neural networks, in: International Conference on Machine Learning, ICML, vol. 162, 2022, pp. 16253–16269.
[24] Y. Man, L. Xie, S. Qiao, Y. Zhou, D. Shang, Differentiable architecture search with multi-dimensional attention for spiking neural networks, Neurocomputing 601 (2024) 128181.
[25] R. Wang, M. Cheng, X. Chen, X. Tang, C. Hsieh, Rethinking architecture selection in differentiable NAS, in: International Conference on Learning Representations, ICLR, 2021.
[26] Y. Sun, B. Xue, M. Zhang, G.G. Yen, Completely automated CNN architecture design based on blocks, IEEE Trans. Neural Netw. Learn. Syst. 31 (4) (2019) 1242–1254.
[27] Z. Lu, G. Sreekumar, E. Goodman, W. Banzhaf, K. Deb, V.N. Boddeti, Neural architecture transfer, IEEE Trans. Pattern Anal. Mach. Intell. 43 (9) (2021) 2971–2989.
[28] H. Zhang, Y. Jin, K. Hao, Evolutionary search for complete neural network architectures with partial weight sharing, IEEE Trans. Evol. Comput. 26 (5) (2022) 1072–1086.
[29] J. Mellor, J. Turner, A. Storkey, E.J. Crowley, Neural architecture search without training, in: International Conference on Machine Learning, ICML, vol. 139, 2021, pp. 7588–7598.
[30] H. Wang, C. Ge, H. Chen, X. Sun, PreNAS: preferred one-shot learning towards efficient neural architecture search, in: International Conference on Machine Learning, ICML, 2023, pp. 35642–35654.
[31] M. Zhang, X. Yu, H. Zhao, L. Ou, ShiftNAS: improving one-shot NAS via probability shift, in: Proceedings of the IEEE International Conference on Computer Vision, ICCV, 2023, pp. 5919–5928.
[32] J. Huang, B. Xue, Y. Sun, M. Zhang, G.G. Yen, Split-level evolutionary neural architecture search with elite weight inheritance, IEEE Trans. Neural Netw. Learn. Syst. 35 (10) (2024) 13523–13537.
[33] Y. Sun, H. Wang, B. Xue, Y. Jin, G.G. Yen, M. Zhang, Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor, IEEE Trans. Evol. Comput. 24 (2) (2019) 350–364.
[34] C. Wei, C. Niu, Y. Tang, Y. Wang, H. Hu, J. Liang, NPENAS: neural predictor guided evolution for neural architecture search, IEEE Trans. Neural Netw. Learn. Syst. 34 (11) (2022) 8441–8455.
[35] X. Song, X. Xie, Z. Lv, G.G. Yen, W. Ding, J. Lv, Y. Sun, Efficient evaluation methods for neural architecture search: a survey, IEEE. Trans. Artif. Intell. 5 (12) (2024) 1–21.
[36] S. Lu, Y. Hu, P. Wang, Y. Han, J. Tan, J. Li, S. Yang, J. Liu, PINAT: a permutation invariance augmented transformer for NAS predictor, in: Proceedings of the AAAI Conference on Artificial Intelligence, AAAI, vol. 37, 2023, pp. 8957–8965.
[37] J. Wu, X. Dai, D. Chen, Y. Chen, M. Liu, Y. Yu, Z. Wang, Z. Liu, M. Chen, L. Yuan, Stronger NAS with weaker predictors, Adv. Neural Inf. Process. Syst. 34 (2021) 28904–28918.
[38] Y. Zhang, L. Feng, H. Shan, L. Yang, Z. Zhu, An AER-based spiking convolution neural network system for image classification with low latency and high energy efficiency, Neurocomputing 564 (2024) 126984.
[39] W. Fang, Y. Chen, J. Ding, Z. Yu, T. Masquelier, D. Chen, L. Huang, H. Zhou, G. Li, Y. Tian, SpikingJelly: an open-source machine learning infrastructure platform for spike-based intelligence, Sci. Adv. 9 (40) (2023) eadi1480.
[40] Q. Meng, M. Xiao, S. Yan, Y. Wang, Z. Lin, Z.-Q. Luo, Towards memory- and time-efficient backpropagation for training spiking neural networks, in: Proceedings of the IEEE International Conference on Computer Vision, ICCV, 2023, pp. 6166–6176.
[41] B. Wang, J. Cao, J. Chen, S. Feng, Y. Wang, A new ANN-SNN conversion method with high accuracy, low latency and good robustness, in: Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI, 2023, pp. 3067–3075.
[42] B. Han, K. Roy, Deep spiking neural network: energy efficiency through time based coding, in: European Conference on Computer Vision, ECCV, 2020, pp. 388–404.
[43] Y. Li, S. Deng, X. Dong, R. Gong, S. Gu, A free lunch from ANN: towards efficient, accurate spiking neural networks calibration, in: International Conference on Machine Learning, ICML, 2021, pp. 6316–6325.
[44] L. Deng, Y. Wu, X. Hu, L. Liang, Y. Ding, G. Li, G. Zhao, P. Li, Y. Xie, Rethinking the performance comparison between SNNs and ANNs, Neural Netw. 121 (2020) 294–307.
[45] Q. Meng, M. Xiao, S. Yan, Y. Wang, Z. Lin, Z.-Q. Luo, Training high-performance low-latency spiking neural networks by differentiation on spike representation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2022, pp. 12434–12443.
[46] Q. Su, Y. Chou, Y. Hu, J. Li, S. Mei, Z. Zhang, G. Li, Deep directly-trained spiking neural networks for object detection, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2023, pp. 6555–6565.
[47] S. Yang, B. Linares-Barranco, Y. Wu, B. Chen, Self-supervised high-order information bottleneck learning of spiking neural network for robust event-based optical flow estimation, IEEE Trans. Pattern Anal. Mach. Intell. (2024).
[48] S. Yang, B. Chen, Effective surrogate gradient learning with high-order information bottleneck for spike-based machine intelligence, IEEE Trans. Neural Netw. Learn. Syst. 36 (1) (2025) 1734–1748.
[49] Z. Wang, R. Jiang, S. Lian, R. Yan, H. Tang, Adaptive smoothing gradient learning for spiking neural networks, in: International Conference on Machine Learning, ICML, vol. 202, 2023, pp. 35798–35816.

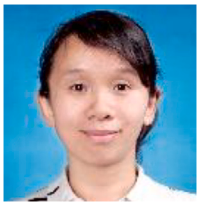
[50] Y. Liu, Y. Sun, B. Xue, M. Zhang, G.G. Yen, K.C. Tan, A survey on evolutionary neural architecture search, IEEE Trans. Neural Netw. Learn. Syst. 34 (2) (2021) 550–570.
[51] B. Zoph, Q. Le, Neural architecture search with reinforcement learning, in: International Conference on Learning Representations, ICLR, 2017.
[52] B. Zoph, V. Vasudevan, J. Shlens, Q.V. Le, Learning transferable architectures for scalable image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2018, pp. 8697–8710.
[53] H. Liu, K. Simonyan, Y. Yang, DARTS: differentiable architecture search, in: International Conference on Learning Representations, ICLR, 2019.
[54] Y. Xue, X. Han, Z. Wang, Self-adaptive weight based on dual-attention for differentiable neural architecture search, IEEE Trans. Ind. Inf. 20 (4) (2024) 6394–6403.
[55] Y. Xue, C. Chen, A. Słowik, Neural architecture search based on a multi-objective evolutionary algorithm with probability stack, IEEE Trans. Evol. Comput. 27 (4) (2023) 778–786.
[56] Y. Xue, J. Zha, M. Wahib, T. Ouyang, X. Wang, Neural architecture search via similarity adaptive guidance, Appl. Soft Comput. 162 (2024) 111821, https://doi.org/10.1016/j.asoc.2024.111821.
[57] W. Zhang, P. Li, Spike-train level backpropagation for training deep recurrent spiking neural networks, Adv. Neural Inf. Process. Syst. 32 (2019).
[58] X. Cheng, Y. Hao, J. Xu, B. Xu, LISNN: improving spiking neural networks with lateral interactions for robust object recognition, in: Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI, 2020, pp. 1519–1525.
[59] N. Rathi, K. Roy, DIET-SNN: a low-latency spiking neural network with direct input encoding and leakage and threshold optimization, IEEE Trans. Neural Netw. Learn. Syst. 34 (6) (2023) 3174–3182.
[60] Y. Wu, L. Deng, G. Li, J. Zhu, Y. Xie, L. Shi, Direct training for spiking neural networks: faster, larger, better, in: Proceedings of the AAAI Conference on Artificial Intelligence, AAAI, vol. 33, 2019, pp. 1311–1318.
[61] J. Yan, Q. Liu, M. Zhang, L. Feng, D. Ma, H. Li, G. Pan, Efficient spiking neural network design via neural architecture search, Neural Netw. 173 (2024) 106172.
[62] I. Garg, S.S. Chowdhury, K. Roy, DCT-SNN: using DCT to distribute spatial information over time for low-latency spiking neural networks, in: Proceedings of the IEEE International Conference on Computer Vision, ICCV, 2021, pp. 4671–4680.
[63] Y. Li, S. Deng, X. Dong, R. Gong, S. Gu, A free lunch from ANN: towards efficient, accurate spiking neural networks calibration, in: International Conference on Machine Learning, ICML, vol. 139, 2021, pp. 6316–6325.
[64] S. Deng, S. Gu, Optimal conversion of conventional artificial neural networks to spiking neural networks, in: International Conference on Learning Representations, ICLR, 2021.
[65] T. Bu, J. Ding, Z. Yu, T. Huang, Optimized potential initialization for low-latency spiking neural networks, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, 2022. pp. 11–20.
[66] Y. Chen, Y. Mai, R. Feng, J. Xiao, An adaptive threshold mechanism for accurate and efficient deep spiking convolutional neural networks, Neurocomputing 469 (2022) 189–197.
[67] T. Bu, W. Fang, J. Ding, P. Dai, Z. Yu, T. Huang, Optimal ANN-SNN conversion for high-accuracy and ultra-low-latency spiking neural networks, in: International Conference on Learning Representations, ICLR, 2022.
[68] Y. Hu, Q. Zheng, X. Jiang, G. Pan, Fast-SNN: fast spiking neural network by converting quantized ANN, IEEE Trans. Pattern Anal. Mach. Intell. 45 (12) (2023) 14546–14562.
[69] M. Xiao, Q. Meng, Z. Zhang, D. He, Z. Lin, Online training through time for spiking neural networks, in: Proceedings of the International Conference on Neural Information Processing Systems, NeurIPS, vol. 35, 2024, pp. 20717–20730.
[70] Z. Xie, Z. Liu, P. Chen, J. Zhang, Efficient spiking neural architecture search with mixed neuron models and variable thresholds, in: Proceedings of the International Conference on Neural Information Processing Systems, NeurIPS, 2024, pp. 466–481.
[71] H. Wu, Y. Zhang, W. Weng, Y. Zhang, Z. Xiong, Z.-J. Zha, X. Sun, F. Wu, Training spiking neural networks with accumulated spiking flow, in: Proceedings of the AAAI Conference on Artificial Intelligence, AAAI, vol. 35, 2021, pp. 10320–10328.
[72] J. Wu, Y. Chua, M. Zhang, G. Li, H. Li, K.C. Tan, A tandem learning rule for effective training and rapid inference of deep spiking neural networks, IEEE Trans. Neural Netw. Learn. Syst. 34 (1) (2023) 446–460.

Author biography

Yuting Wang is currently pursuing the master's degree in the school of electronic and information engineering with Anhui Jianzhu University, Hefei, China. Her research interests include deep learning, neural architecture search, and spiking neural network.



Yu Xue (Senior Member, IEEE) received the Ph.D. degree from the School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2013. He was a Visiting Scholar with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand, from August 2016 to August 2017. He was a Research Scholar with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, USA, from October 2017 to November 2018. He is currently a Professor with the School of Software, Nanjing University of Information Science and Technology, Nanjing. His research interests include deep learning, evolutionary computation, machine learning, computer vision, and feature map selection.



Wei Ding is a doctor in computer applications and is currently a lecturer at Anhui Jianzhu University, Hefei, China. Her research interests include computer security and computer applications.

Yiyu Tan received the B.S. degree in Electrical Engineering from Jiangnan University in 1994, M.S. and Ph.D. degrees in Electronic Engineering from Nanjing University and City University of Hong Kong in 2001 and 2006, respectively. He is currently an associate professor in the faculty of Science and Engineering, Iwate University. Before he joined Iwate University, he worked as a research scientist in the RIKEN Center for Computational Science, a post-doctoral researcher in Japan Advanced Institute of Science and Technology, a lecturer in Nanjing University, an associate professor in Nanjing University of Technology. His research interests include sound field rendering, reconfigurable systems, and hardware-based acceleration on numerical computing.

Peng Chen is a senior scientist at the RIKEN Center for Computational Science (RIKEN-CCS), Japan. He received his B.E. degree in Navigation from Dalian Maritime University, China, in 2005, followed by an M.E. degree in Traffic Information Engineering and Control from Shanghai Maritime University, China, in 2007. He earned his Ph.D. from the Tokyo Institute of Technology, Japan, in 2020. His research interests include high-performance computing (HPC), parallel computing, image processing, machine learning, and evolutionary computation.

Mohamed Wahib is currently a team leader of the "High Performance Artificial Intelligence Systems Research Team" at RIKEN Center for Computational Science (R-CCS), Kobe, Japan. Prior to that he worked as is a senior scientist at AIST/TokyoTech Open Innovation Laboratory, Tokyo, Japan. He received his Ph.D. in Computer Science in 2012 from Hokkaido University, Japan. His research interests revolve around the central topic of high-performance programming systems, in the context of HPC and AI. He is actively working on several projects including high-level frameworks for programming traditional scientific applications, as well as high-performance AI.