Neural Networks 121 (2020) 294–307

Contents lists available at ScienceDirect

Neural Networks

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

Rethinking the performance comparison between SNNS and ANNS

Lei Deng [a,b,1], Yujie Wu [a,1], Xing Hu [b,*], Ling Liang [b], Yufei Ding [c], Guoqi Li [a,*], Guangshe Zhao [d], Peng Li [b], Yuan Xie [b]

[a] Department of Precision Instrument, Center for Brain Inspired Computing Research, Tsinghua University, Beijing 100084, China
[b] Department of Electrical and Computer Engineering, University of California, Santa Barbara,, CA 93106, USA
[c] Department of Computer Science, University of California, Santa Barbara,, CA 93106, USA
[d] School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China

ARTICLE INFO

Article history:
Available online 19 September 2019

Keywords:
Spiking neural networks
Artificial neural networks
Deep learning
Neuromorphic computing
Benchmark

ABSTRACT

Artificial neural networks (ANNs), a popular path towards artificial intelligence, have experienced remarkable success via mature models, various benchmarks, open-source datasets, and powerful computing platforms. Spiking neural networks (SNNs), a category of promising models to mimic the neuronal dynamics of the brain, have gained much attention for brain inspired computing and been widely deployed on neuromorphic devices. However, for a long time, there are ongoing debates and skepticisms about the value of SNNs in practical applications. Except for the low power attribute benefit from the spike-driven processing, SNNs usually perform worse than ANNs especially in terms of the application accuracy. Recently, researchers attempt to address this issue by borrowing learning methodologies from ANNs, such as backpropagation, to train high-accuracy SNN models. The rapid progress in this domain continuously produces amazing results with ever-increasing network size, whose growing path seems similar to the development of deep learning. Although these ways endow SNNs the capability to approach the accuracy of ANNs, the natural superiorities of SNNs and the way to outperform ANNs are potentially lost due to the use of ANN-oriented workloads and simplistic evaluation metrics.

In this paper, we take the visual recognition task as a case study to answer the questions of "*what workloads are ideal for SNNs and how to evaluate SNNs makes sense*". We design a series of contrast tests using different types of datasets (ANN-oriented and SNN-oriented), diverse processing models, signal conversion methods, and learning algorithms. We propose comprehensive metrics on the application accuracy and the cost of memory & compute to evaluate these models, and conduct extensive experiments. We evidence the fact that on ANN-oriented workloads, SNNs fail to beat their ANN counterparts; while on SNN-oriented workloads, SNNs can fully perform better. We further demonstrate that in SNNs there exists a trade-off between the application accuracy and the execution cost, which will be affected by the simulation time window and firing threshold. Based on these abundant analyses, we recommend the most suitable model for each scenario. To the best of our knowledge, this is the first work using systematical comparisons to explicitly reveal that the straightforward workload porting from ANNs to SNNs is unwise although many works are doing so and a comprehensive evaluation indeed matters. Finally, we highlight the urgent need to build a benchmarking framework for SNNs with broader tasks, datasets, and metrics.

1. Introduction

Artificial neural networks (ANNs) (LeCun, Bengio, & Hinton, 2015) are able to learn high-level features from large amount of input data via the deep hierarchy. This powerful representation brings amazing successes in a myriad of artificial intelligence (AI) applications. For example, researchers report multi-layered perceptron (MLP) or convolutional neural networks (CNNs)-based image recognition (He, Zhang, Ren, & Sun, 2016), speech recognition (Abdel-Hamid et al., 2014), language processing (Hu, Lu, Li, & Chen, 2014; Young, Hazarika, Poria, & Cambria, 2018), object detection (Redmon & Farhadi, 2017), solar radiation estimation (Jahani & Mohammadi, 2018), medical diagnosis (Esteva et al., 2017), game playing (Silver et al., 2016), etc., recurrent neural networks (RNNs)-based speech recognition (Lam et al., 2019), language processing (Ghaeini et al., 2018), state control

* Corresponding authors.
E-mail addresses: leideng@ucsb.edu (L. Deng), wu-yj16@mails.tsinghua.edu.cn (Y. Wu), xinghu@ucsb.edu (X. Hu), lingliang@ucsb.edu (L. Liang), yufeiding@cs.ucsb.edu (Y. Ding), liguoqi@mail.tsinghua.edu.cn (G. Li), zhaogs@xjtu.edu.cn (G. Zhao), lip@ucsb.edu (P. Li), yuanxie@ucsb.edu (Y. Xie).

[1] Equal contribution.

https://doi.org/10.1016/j.neunet.2019.09.005
0893-6080/

(Graves et al., 2016), etc., and sometimes the combination of CNNs and RNNs (Caglayan & Burak Can, 2018; Zhang, Bai, & Zhu, 2019; Zoph, Vasudevan, Shlens, & Le, 2018). Besides various models and learning algorithms, the big data resources (e.g. ImageNet dataset (Deng et al., 2009) for image recognition) and high-performance computing platforms (e.g. GPU) further boost the development of ANNs. The above successes motivate numerous researches on ANN-specific accelerators (Chen et al., 2014; Chen, Krishna, Emer, & Sze, 2017; Jouppi et al., 2017; Yin et al., 2017).

Spiking neural networks (SNNs) (Ghosh-Dastidar & Adeli, 2009; Maass, 1997) closely mimic the behaviors of biological neural circuits. They operate with continuous spatio-temporal dynamics and event-driven firing activities (0-nothing or 1-spike event). Due to the asynchronous spiking mechanism, SNNs have shown advantages in event-based scenarios such as optical flow estimation (Haessig, Cassidy, Alvarez, Benosman, & Orchard, 2017), spike pattern recognition (Wu et al., 2019), and SLAM (Vidal, Rebecq, Horstschaefer, & Scaramuzza, 2018). Besides, they are also promising in addressing some interesting problems, for instance, probabilistic inference (Maass, 2014), heuristically solving NP-hard problem (Jonke, Habenschuss, & Maass, 2016) or quickly solving optimization problem (Davies et al., 2018), sparse representation (Shi, Liu, Wang, Li, & Gu, 2017), and robotics (Hwu, Isbell, Oros, & Krichmar, 2017). Furthermore, SNNs are widely deployed in neuromorphic devices for brain inspired computing (Davies et al., 2018; Furber, Galluppi, Temple, & Plana, 2014; Merolla et al., 2014; Shi et al., 2015). Whereas, there are ongoing debates for a long time about the practical value of SNNs as a computational tool in both AI and neuromorphic computing communities (Davies et al., 2018), especially when compared to ANNs. These skepticisms slow down the development of neuromorphic computing during the past few years, which is upstaged by the rapid progress of deep learning. Researchers attempt to fundamentally mitigate this issue by strengthening SNNs from means such as training algorithm design.

Unlike the mature and effective training algorithms such as error backpropagation (BP) for ANNs, one of the most difficulties in SNN study is the hardness of training caused by the complex dynamics and non-differentiable spike activities. The case still remains challenging even if we use the simple leaky-integrate-and-fire (LIF) (Gerstner, Kistler, Naud, & Paninski, 2014) neuron model. To enhance the application accuracy of SNNs, the conventional spike timing dependent plasticity (STDP) unsupervised learning rule can be improved such as by adding lateral inhibition and adaptive threshold (Diehl & Cook, 2015) or reward mechanism (Mozafari, Ganjtabesh, Nowzari-Dalini, Thorpe, & Masquelier, 2018). Some other works pre-train adapted ANNs and convert them to SNNs (Cao, Chen, & Khosla, 2015; Diehl et al., 2015; Hu, Tang, Wang, & Pan, 2018; Hunsberger & Eliasmith, 2016; Rueckauer, Lungu, Hu, Pfeiffer, & Liu, 2017; Sengupta, Ye, Wang, Liu, & Roy, 2019). The adaptions in ANNs usually include removing biases, using ReLU activation function (or its variant), changing max pooling to average pooling, etc., to enhance the compatibility with SNN models. The conversion from ANNs to SNNs often induces weight/activation normalization, threshold tuning, sampling error compensation, etc., to maintain the accuracy. Recently, the supervised BP learning in ANNs is borrowed to train accurate SNNs directly (Jin, Li, & Zhang, 2018; Lee, Delbruck, & Pfeiffer, 2016; Pengjie Gu & Tang, 2019; Shrestha & Orchard, 2018; Wu, Deng, Li, Zhu, & Shi, 2018; Wu et al., 2019). The gradients can propagate along only the spatial direction by aggregating spikes in the temporal dimension when performing BP, or propagate along both temporal and spatial dimensions by directly calculating the derivates of membrane potential and spike activity at each time step. The combination of BP and STDP learning also exists, such as applying STDP update after the BP update at each training iteration (Tavanaei & Maida, 2017) or applying BP fine-tuning after an STDP pre-training (Lee, Panda, Srinivasan, & Roy, 2018). In a nutshell, with the above efforts, SNNs are gradually approaching the ANN-level application accuracy in visual recognition tasks.

Due to the lack of specialized benchmark workloads for SNNs, plenty of works directly port the testing workloads from the ANN domain to verify SNN models (Cao et al., 2015; Diehl & Cook, 2015; Diehl et al., 2015; Hu et al., 2018; Hunsberger & Eliasmith, 2016; Jin et al., 2018; Lee et al., 2016, 2018; Mozafari et al., 2018; Rueckauer et al., 2017; Sengupta et al., 2019; Tavanaei & Maida, 2017; Wu et al., 2018). For example, the image datasets for ANN validation are simply converted to the spike version for SNN training and testing. This seems reasonable because the input data are encoded in spikes when SNNs run. Whereas, the original datasets for ANNs are just static images which cannot make full use of the spatio-temporal superiorities of SNNs even if they are converted to spike patterns. Furthermore, the application accuracy is still the prime evaluation metric, but it is known that our brain usually performs worse than current AI machines in terms of absolute recognition accuracy. This reflects that we need more comprehensive and fair metrics to evaluate the brain inspired SNNs. In a nutshell, due to the inappropriate workloads and evaluation metrics, current SNNs fail to beat ANNs. Therefore, there comes up two open questions that "*what workloads are ideal for SNNs and how to evaluate SNNs makes sense*".

In this paper, we try to answer the above questions and take the visual recognition task as a case study. In Hunsberger and Eliasmith (2016) that converts pre-trained ANNs with soft-LIF neurons and noisy activities to SNNs, the efficiencies of SNNs and ANNs are defined and compared. It is one of the few works that consider the metric beyond accuracy in the SNN algorithm community and study the trade-off between accuracy and efficiency. We extend this path to a deeper rethinking via more systematical modeling methodologies and experimental analyses. Specifically, through designing a series of contrast tests using datasets in different domains, diverse processing models, signal conversion methods, and learning algorithms, we compare the performance of ANNs and SNNs (with rate coding). We propose comprehensive metrics on the application accuracy, memory cost, and compute cost to evaluate these models and conduct extensive analyses. We evidence the fact that SNNs fail to beat ANNs at the same network scale on ANN-oriented workloads with respect to the accuracy but hold the potential for efficient processing; while on SNN-oriented workloads, SNNs can fully perform better. We further demonstrate that the change of simulation time window and firing threshold will produce a trade-off between the application accuracy and the execution cost. Based on these analyses, we recommend the most suitable model for each scenario. The major contributions of this work are summarized as follows:

- We propose the much-needed comprehensive evaluation metrics for SNNs and ANNs by taking the trade-off between application accuracy and execution cost into account. We conduct extensive experiments and analyses with diverse benchmark datasets, processing models, signal conversions, and learning algorithms. Various comparison methodologies and visualization means are also demonstrated.
- Based on the above results, we recommend the best model for each workload, which provides insightful modeling guidances.
- We point out that directly porting workloads from ANNs to SNNs is inappropriate, at least unwise, although many works are doing so. We further highlight that to create more SNN-oriented datasets and build a benchmarking framework with broader tasks is urgent for the SNN community. This shall shed light on the future opportunities of SNN research.

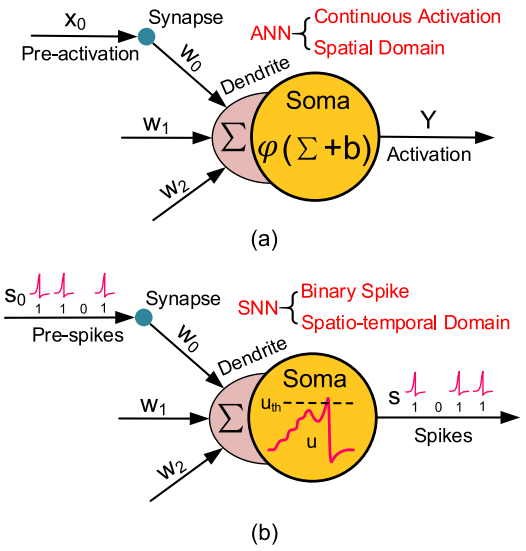

Fig. 1. Basic neuron model in (a) ANNs and (b) SNNs.

The rest of this paper is organized as follows: Section 2 introduces some preliminaries of ANNs and SNNs, as well as typical network topologies and benchmark datasets; Section 3 systematically proposes signal conversion methods, testing workloads, training algorithms, and evaluation metrics; The experimental setup and result analyses along with visualizations are provided in Section 4; Finally, Section 5 concludes and discusses the paper.

2. Preliminaries

In this section, we give the preliminary knowledge of ANNs and SNNs for visual recognition, including neuron models, network topologies, and benchmark datasets.

Neurons are the basic compute units, which are wired by abundant synapses to form a neural network. If we treat the neural network as a graph, each neuron and synapse can be viewed as a node and edge, respectively. As aforementioned, there are two categories of neural networks: ANNs and SNNs, which will be explained in detail as follows.

2.1. Artificial neural networks

Fig. 1(a) depicts the model of a typical artificial neuron. The compute process is governed by

$$y = \varphi(b + \sum_j x_j w_j) \tag{1}$$

where x, y, w, and b are input activation, output activation, synaptic weight, and bias, respectively, and j is the index of input neuron. $\varphi(\cdot)$ is a nonlinear activation function, e.g. $\varphi(x) = ReLU(x) = max(x, 0)$. Neurons in ANNs communicate with each other using activations coded in high-precision and continuous values, and only propagate information in the spatial domain (i.e. layer by layer). From the above equation, it can be seen that the multiply-and-accumulate (MAC) of inputs and weights is the major operation in ANNs.

2.2. Spiking neural networks

Fig. 1(b) shows a typical spiking neuron, which has a similar structure but different behavior compared to the ANN neuron. By contrast, spiking neurons communicate through spike trains coded in binary events rather than the continuous activations in ANNs. The dendrites integrate the input spikes and the soma consequently conducts nonlinear transformation to produce the output spike train. This behavior is usually modeled by the popular LIF model (Gerstner et al., 2014), described as

$$\begin{cases} \tau \frac{du(t)}{dt} = -[u(t) - u_{r_1}] + \sum_j w_j \sum_{t_j^k \in S_j^{T_w}} K(t - t_j^k) \\ \begin{cases} s(t) = 1 \ \& \ u(t) = u_{r_2}, \ \text{if } u(t) \geq u_{th} \\ s(t) = 0, \ \text{if } u(t) < u_{th} \end{cases} \end{cases} \tag{2}$$

where (t) denotes the time step, τ is a time constant, and u and s are the membrane potential and output spike, respectively. u_{r_1} and u_{r_2} are the resting potential and reset potential, respectively. w_j is the synaptic weight from the jth input neuron, t_j^k is the time when the kth spike of the jth input neuron fires within the integration time window of T_w (a spike sequence of $S_j^{T_w}$ in total), and $K(\cdot)$ is a kernel function describing the time decay effect. u_{th} is the firing threshold that determines whether to fire a spike or not. Besides the LIF model, there also exist other neuron models in SNNs, such as the model of Hodgkin and Huxley (1952) or Izhikevich (2003). However, due to the higher complexity they are not widely used in practical SNN models.

Different from ANNs, SNNs represent information in spike patterns and each spiking neuron experiences rich dynamic behaviors. Specifically, besides the information propagation in the spatial domain, the current state is tightly affected by the past history in the temporal domain. Therefore, SNNs usually have more temporal versatility but lower precision compared to ANNs mainly with spatial propagation and continuous activations. Since a spike only fires when the membrane potential exceeds a threshold, the entire spike signals are often sparse and the compute can be event driven (only enabled when a spike input arrives). Furthermore, because the spike is binary, i.e. 0 or 1, the costly multiplication between the input and weight can be removed if the integration time window T_w equals to 1 (see Section 3.5). For above reasons, SNNs can usually achieve lower power consumption compared to ANNs with intensive computation.

2.3. Typical network topologies

The basic layer topologies used to build neural networks are fully-connected (FC) layer, recurrent layer, and convolutional (Conv) layer (as well as pooling layer). The corresponding networks are named as multi-layered perceptron (MLP), RNNs, and CNNs, respectively. MLP and RNNs only include stacked FC layers with or without recurrent connections in each layer as shown in Fig. 2(a) and (b), respectively. For CNNs shown in Fig. 2(c), they directly target the processing of 2D features instead of the 1D ones in MLP and RNNs. Each neuron in the convolutional (Conv) layer only receives inputs from a local receptive field (RF) across all feature maps (FMs) in previous layer. The basic calculation of each neuron is the same as Eq. (1) or (2). In addition, CNNs also use the pooling layer to down sample the size of each FM individually via outputting the maximum (i.e. max pooling) or average (i.e. average pooling) value of each RF, and use the FC layer for final classification.

2.4. Benchmark datasets

Fig. 3 illustrates two different types of benchmark datasets we used for visual recognition. The upper two rows are sampled from MNIST (LeCun, Bottou, Bengio, & Haffner, 1998) and CIFAR10 (Krizhevsky & Hinton, 2009) datasets. Because they are frame-based static images and widely used in ANNs, we call

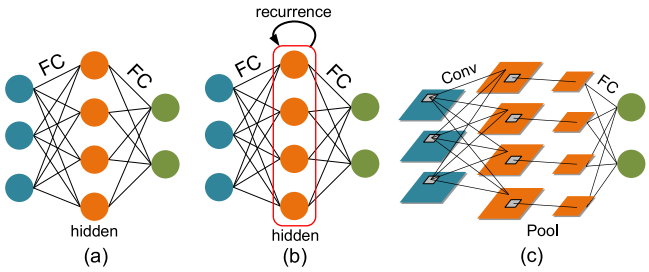

Fig. 2. Network topologies: (a) MLP; (b) RNN; (c) CNN.

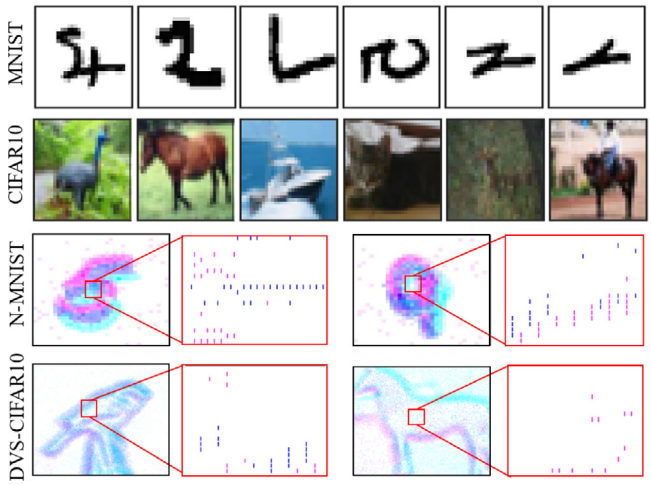

Fig. 3. Illustration of benchmark datasets for visual recognition, including ANN-oriented datasets (MNIST, CIFAR10) and SNN-oriented datasets (N-MNIST, DVS-CIFAR10). Red or blue color denotes the On or Off event, respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

them ANN-oriented datasets. The lower two rows are sampled from N-MNIST (Orchard, Jayawant, Cohen, & Thakor, 2015) and DVS-CIFAR10 (Li, Liu, Ji, Li, & Shi, 2017). The data format is spike event which is converted from the above static datasets through scanning each image by using dynamic vision sensor (DVS) (Lichtsteiner, Posch, & Delbruck, 2008). Besides the similar spatial information as ANN-oriented datasets, it contains more dynamic temporal information and the spike events are naturally compatible with the signal format in SNNs, therefore we call them SNN-oriented datasets.

For the ANN-oriented datasets, MNIST is comprised of a training set with 60,000 labeled hand-written digits, and a testing set of other 10,000 labeled digits. Each digit sample is a 28×28 grayscale image. CIFAR10 contains a training set with 50,000 labeled training images with natural and manufactured objects in their environment, and other 10,000 testing images. Each sample is a $32 \times 32 \times 3$ RGB image.

For the SNN-oriented datasets, DVS can scan the original static image along given directions and collect the generated spike trains triggered by the intensity change of each pixel. Since it has two change manners (increase or decrease), DVS produces two channels of spike events, named as On and Off event (red and blue color in Fig. 3, respectively). Hence, DVS converts each image into $row \times col \times 2 \times T$ spike pattern, where T is the recording time length. Due to the relative shift during the movement, the resulting images usually have a larger size than original images. In this way, N-MNIST converts the original MNIST into its spike version. It also contains 60,000 training samples and 10 000 testing samples and each sample in N-MNIST is a spatio-temporal

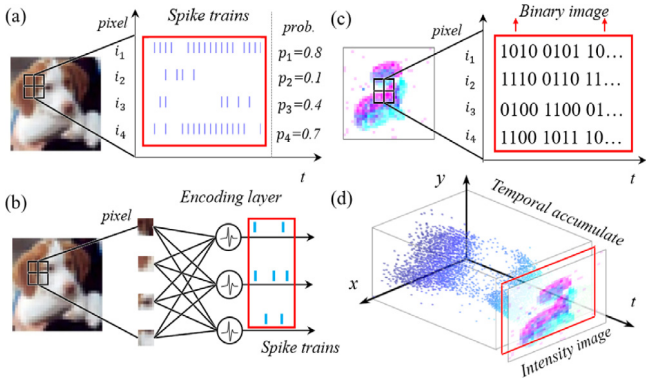

Fig. 4. Illustration of data signal conversion: (a) image to spike pattern by probabilistic sampling; (b) image to spike pattern by encoding layer; (c) spike pattern to binary image; (d) spike pattern to intensity image.

spike pattern with size of $34 \times 34 \times 2 \times T$. DVS-CIFAR10 converts only 10 000 images from the original CIFAR10 datasets, in which each class has 1000 spike patterns with size of $128 \times 128 \times 2 \times T$. Because of the much smaller data volume than the original CIFAR10, we randomly select 9000 images for training and the other 1000 for testing.

3. Benchmarking methodology

In this section, we first introduce the signal conversion between the ANN and SNN domains. Then we design six benchmark models along with the training algorithms for contrast tests and propose three evaluation metrics for consequent model assessment and comparison.

3.1. Data signal conversion

In general, ANNs receive frame-based images in real value while SNNs receive event-driven spike signals. Thus, sometimes it is necessary to transform the same data resource into a different form for the processing in the other domain. Here we take the visual recognition task as a case study and primarily introduce four signal conversion methods as follows.

Image to spike pattern. Since real-valued signals of pixel intensity are not suitable for spike-based SNNs, the conversion to spike trains is needed when testing SNN models over ANN-oriented datasets. One of the prevalent strategies is the probabilistic sampling. At every time step, it samples the original pixel intensity (usually normalized to [0, 1]) into a binary value, wherein the probability of being 1 (firing a spike) equals to the intensity value. The sampling follows a specific probability distribution such as Bernoulli distribution or Poisson distribution. For example, the i_1 neuron in Fig. 4(a), corresponding to the pixel on the upper left with a normalized intensity of 0.8, produces a binary spike train following the Bernoulli distribution $B(0.8, T)$. Here T is a given time window for sampling.

The above element-wise sampling usually suffers from precision loss in the case of short time window (see Section 4.2). To circumvent this problem, modern works (Esser et al., 2016) add an encoding layer to generate spike signals globally, as shown in Fig. 4(b). Each neuron in this layer receives intensity values of multiple pixels as the input (i.e. the dendrite works in ANN mode with normal input-weight MAC operations) while produces spikes as the output (i.e. the soma works in SNN mode with LIF dynamics). Although the encoding layer is an ANN–SNN hybrid layer rather than a complete SNN layer like following layers in the network, its weights are trainable because our training

method for SNNs in this paper is also BP-compatible for ANNs (see Section 3.4). Since the number of neurons can be flexibly customized and the parameters are trainable, it can adapt to the overall optimization problem for better accuracy.

Spike pattern to image. Similarly, to test ANN models over SNN-oriented datasets, we need to convert spike patterns into frame-based images. There exist two possible output formats: (i) binary image with 0/1 pixels; (ii) intensity image with real-valued pixels. An intuitive strategy for converting a spike pattern into binary images is to first expand the spike trains from all neuron locations along the temporal dimension. Then, as Fig. 4(c) illustrates, the expanded 2D spike pattern (location index v.s. time) can be directly viewed as a binary image (every spike event represents the pixel intensity being 1, otherwise the pixel intensity is 0). The original recording time length T is usually long which might result in a too big image size. So the spike events along the temporal direction need to be aggregated or sampled periodically with a moderate sliding time window (e.g. every 1 ms). Here the aggregation means the resulting spike event is 1 if it has spikes in the window, otherwise it is 0. Even if so, the unfolding of 2D locations (the left side in Fig. 4(c)) to an 1D location vector (Y-axis of the right side in Fig. 4(c)) also produces a big image size.

For the conversion into intensity images, a temporal accumulation of the spike events (counting spike numbers) is required. Fig. 4(d) depicts the accumulation process of spike trains within 100 ms. The accumulated spike numbers will be normalized to be pixels with a proper intensity value. Due to the relative motion and intrinsic noise of DVS, the resulting image is often blurred and its edge features are obscure. This conversion is only allowed by a strong assumption that each spike location should not move away its beginning location as t evolves, otherwise it will severely harm the quality of resulting image.

3.2. ANN-oriented workloads

Here the "ANN-oriented workloads" means the target is to recognize the images in frame-based datasets (e.g. MNIST and CIFAR10), which is widely used in the ANN field. To process this type of workloads, we introduce three benchmark models. As shown in Fig. 5(a), the most straightforward solution is the natural ANN with "ANN training & ANN inference". It fully trains the network in the ANN mode (Eq. (1)) with intensity images and then does consequent inference in the same domain. The training follows the most widely used BP algorithm in the ANN field.

Besides, many works in the SNN community also use these datasets to test the performance of SNNs (Diehl et al., 2015; Hu et al., 2018; Lee et al., 2016; Sengupta et al., 2019; Wu et al., 2018, 2019). Since in these cases the network works in the SNN mode, the input data need to be converted from frame-based images to spike events, as illustrated in Fig. 4(a) or (b). After the signal conversion, we further provide two modeling branches. One is shown in Fig. 5(b), where an ANN is first trained using BP algorithm on the original image dataset, and then the pre-trained ANN is adapted to its SNN counterpart with the same structure but different neuron model. This converted SNN receives the dataset variant with spike events in the inference phase. The other is the enforced SNN as depicted in Fig. 5(c), where an SNN model is directly trained from scratch on the converted spike datasets and tested in the same domain. As aforementioned in Section 1, the emerging BP-inspired supervised algorithms for training SNNs usually present better accuracy than the unsupervised ones. Actually they use different learning methodologies. For example, both synaptic weights and axonal delays are trainable variables under the error backpropagation framework in Shrestha and Orchard (2018), the gradients at every location and time are directly

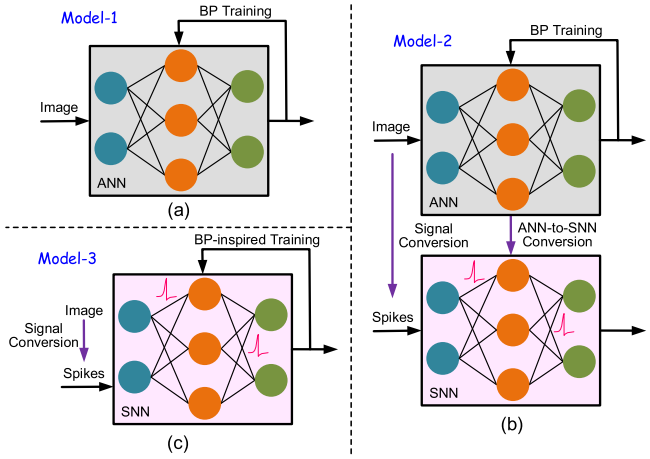

Fig. 5. Model configuration on ANN-oriented datasets: (a) Model-1, natural ANN with ANN training and ANN inference; (b) Model-2, converted SNN with ANN training and SNN inference; (c) Model-3, enforced SNN with SNN training and SNN inference.

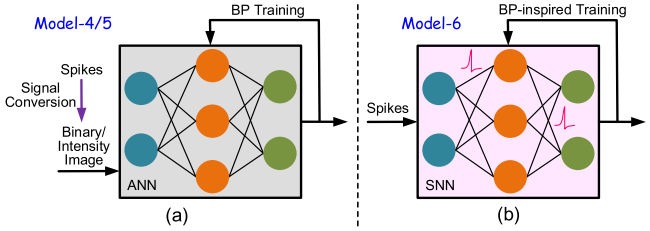

Fig. 6. Model configuration on SNN-oriented datasets: (a) Model-4 or Model-5, enforced binary ANN or enforced intensity ANN using converted binary or intensity images, respectively, for ANN training and ANN inference; (b) Model-6, natural SNN with SNN training and SNN inference.

derived using spatio-temporal backpropagation (STBP) method in Wu et al. (2018) and Wu et al. (2019), while a temporal based loss function is further proposed to solve the spatio-temporal credit assignment problem in Pengjie Gu and Tang (2019). Considering the open-source codes in Pytorch, we select the STBP for the direct training of SNNs. In summary, the three benchmark models for ANN-oriented workloads are denoted as Model-1/2/3 for clarity. Note that we use the rate coding scheme in the SNN domain throughout this paper, which is because of its higher accuracy indicated by previous work.

3.3. SNN-oriented workloads

Here the "SNN-oriented workloads" means the target is to recognize the images in frame-free spike datasets (e.g. N-MNIST and DVS-CIFAR10), which is widely used in the SNN field. Like the above ANN-oriented workloads, here we also introduce three benchmark models. The first two work in the ANN mode (see Fig. 6(a)), where the input data are converted into images. The models are trained using the regular BP algorithm for ANNs and tested also in the ANN domain. There are two ways for the data conversion from spike events to images: (i) binary images by directly receiving the unfolded spike pattern; (ii) intensity images by compressing the spike pattern along the temporal dimension, which are illustrated in Fig. 4(c) and (d), respectively. Fig. 6 gives another benchmark model that works in the natural SNN mode. The network is directly trained and tested using the original spike dataset, and the learning algorithm is still BP-inspired as above in Model-3. Similarly, the three benchmark models for SNN-oriented workloads are denoted as Model-4/5/6.

3.4. Training algorithms

In the above section, we mentioned two training algorithms for the six benchmark models: (i) BP for ANNs; (ii) BP-inspired variant for SNNs. The BP algorithm is widely used in the training of ANN. It can be simply described as

$$\begin{cases} \frac{\partial L}{\partial y_i^n} = \sum_j \frac{\partial L}{\partial y_j^{n+1}} \varphi_j^{'n+1} w_{ji}^{n+1} \\ \nabla w_{ji}^n = \frac{\partial L}{\partial y_j^{n+1}} \varphi_j^{'n+1} y_i^n, \quad \nabla b_j^n = \frac{\partial L}{\partial y_j^{n+1}} \varphi_j^{'n+1} \end{cases} \tag{3}$$

where i and j are the neuron index in previous layer n and current layer $(n+1)$, respectively, $\varphi_j^{'n+1}$ is the activation derivative of the jth neuron in layer $(n+1)$, and L is the loss function for optimization. Usually, we can simply use mean square error (MSE) as the cost to be minimized, i.e. let $L = \frac{1}{2}\|\boldsymbol{Y}^{label} - \boldsymbol{Y}\|_2^2$. Here $\boldsymbol{Y}$ and $\boldsymbol{Y}^{label}$ are the actual output and labeled ground truth, respectively.

As aforementioned, because of the superior accuracy and open-source codes, we select STBP (Wu et al., 2018, 2019) to train our SNNs. It is based on an iterative version of the original LIF model in Eq. (2). Specifically, it yields

$$\begin{cases} u_i^{t+1,n+1} & = e^{-\frac{dt}{\tau}} u_i^{t,n+1}(1 - o_i^{t,n+1}) + \sum_j w_{ij}^n o_j^{t+1,n} \\ o_i^{t+1,n+1} & = f(u_i^{t+1,n+1} - u_{th}) \end{cases} \tag{4}$$

where dt is the length of simulation time step, o denotes the neuronal spike output, t and n are time step and layer index, respectively. $e^{-\frac{dt}{\tau}}$ reflects the decay effect of the membrane potential. $f(\cdot)$ is a step function, which satisfies $f(x) = 1$ when $x \geq 0$, otherwise $f(x) = 0$. This iterative LIF format incorporates all the behaviors in the original neuron model including integration, firing, and reset. Note that here we set $u_{r_1} = u_{r_2} = 0$, $T_w = 1$, and $K(\cdot) \equiv 1$ in the original LIF model for simplicity. Given the iterative LIF model, the gradient propagates along both the temporal and spatial dimensions, and the parameter update can be derived accordingly as follows

$$\begin{cases} \frac{\partial L}{\partial o_i^{t,n}} = \sum_j \frac{\partial L}{\partial o_j^{t,n+1}} \frac{\partial o_j^{t,n+1}}{\partial o_i^{t,n}} + \frac{\partial L}{\partial o_i^{t+1,n}} \frac{\partial o_i^{t+1,n}}{\partial o_i^{t,n}}, \\ \frac{\partial L}{\partial u_i^{t,n}} = \frac{\partial L}{\partial o_i^{t,n}} \frac{\partial o_i^{t,n}}{\partial u_i^{t,n}} + \frac{\partial L}{\partial o_i^{t+1,n}} \frac{\partial o_i^{t+1,n}}{\partial u_i^{t,n}}, \\ \nabla w_{ji}^n = \sum_{t=1}^T \frac{\partial L}{\partial u_j^{t,n+1}} o_i^{t,n}. \end{cases} \tag{5}$$

To solve the non-differentiable problem, Wu et al. (2018) introduces an auxiliary function to approximate the derivative of step function $f(\cdot)$ when calculating $\frac{\partial o}{\partial u}$. It satisfies

$$\frac{\partial o}{\partial u} \approx \frac{1}{a} sign(|u - u_{th}| < \frac{a}{2}) \tag{6}$$

where the parameter a determines the gradient width. We have $sign(x) = 1$ when $x > 0$ and $sign(x) = 0$ when $x = 0$. The loss function L measures the discrepancy between the ground truth and the average firing rate of the last layer N within a given time window T. Also, it can be defined as the format of MSE like that in ANNs, i.e. $L = \|\boldsymbol{Y}^{label} - \frac{1}{T}\sum_{t=1}^T \boldsymbol{O}^{t,N}\|_2^2$.

3.5. Evaluation metrics

As well known, the SNNs-based brain usually cannot beat the current ANNs-based AI system in terms of the absolute recognition accuracy, while the true brain performs better on other metrics, such as operating efficiency. Whereas, in most recent works, recognition accuracy is still a mainstream metric to judge which model (ANNs or SNNs) is better, especially in the algorithm research. This is not fair enough since ANNs and SNNs have very different characteristics. For example, the data precision in ANNs are much higher than SNNs, which makes ANNs easier to gain better recognition accuracy at the same network size. All these indicate more comprehensive metrics are needed for model evaluation. Besides the common accuracy comparison, here we further introduce memory and compute cost as complementary evaluation metrics. Note that this is just a beginning, and more insightful metrics are expected in the future (see discussions in Section 5.2).

Recognition accuracy. Here we use the top-1 accuracy. In ANNs, this accuracy means the percentage of the correctly recognized samples. If the label class is the same with the one predicted by the model with the maximum activation value, the recognition is correct for the current sample. In SNNs, we first count the fire rate (number of spikes) of the output neurons during a given time window T for rate coding, then determine the predicted class $\hat{C}$ according to the fire rate values. The consequent accuracy calculation is the same as that in ANNs. A well-accepted decoding method can be defined as follows

$$\hat{C} = \underset{i}{\text{argmax}}\{\frac{1}{T}\sum_{t=1}^T o_i^{t,N}\} \tag{7}$$

where $o_i^{t,N}$ denotes the spike output of the ith neuron in the last layer N at the tth time step.

Notation: Before stepping into the introduction of memory and compute costs, we make a clarification that in this paper we consider costs in only the inference phase. On one hand, the training of SNNs under Eq. (5) with spatio-temporal gradient propagation paths is more complicated than that of the inference phase. The complication will weaken the readability of this work since it is already complicated due to various signal conversions, modeling methods, workloads, and evaluation metrics. On the other hand, most neuromorphic devices supporting SNNs perform only the inference phase. Therefore, it is better to focus on the inference phase that can be quickly applied in current hardware platforms.

Memory cost. Usually, the memory footprint matters a lot when deploying models on embedded devices. In ANNs, the memory cost includes the weight memory and activation memory. The overhead of the activation function is ignored but it should be counted if it is realized using look up table. In SNNs, the memory cost includes the weight memory, membrane potential memory, and spike memory. Other parameters such as the fire threshold u_{th} and time constant τ are negligible since they can be shared by all neurons in the same layer or in the whole network. The spike memory overhead only occurs when spike fires. In summary, the memory cost can be calculated by

$$\begin{cases} \text{ANN: } M = M_w + M_a \\ \text{SNN: } M = M_w + M_p + M_s \end{cases} \tag{8}$$

where M_w, M_a, M_p, and M_s denote the memory cost for weights, activations, membrane potentials, and spikes, respectively. Compared to the static values of M_w, M_a, and M_p that are determined by the network structure, M_s is dynamically determined by the maximum number of spike events at a certain time step. Here for clarity, all the data are assumed to be stored in 32 bits except for the 1-bit spike data. Note that, in models with data quantization (Banner, Hubara, Hoffer, & Soudry, 2018; Deng, Jiao, Pei, Wu, & Li, 2018) or on specialized devices (Davies et al., 2018; Jouppi et al., 2017; Merolla et al., 2014), the bitwidth can be reduced to save the memory cost. For some compressed models via matrix/tensor decomposition (Novikov, Podoprikhin, Osokin, & Vetrov, 2015) or sparsification (Li, Kadav, Durdanovic, Samet, & Graf, 2016; Liu et al., 2018) techniques, the memory cost can also be reduced. However, these are out of the scope of this paper.

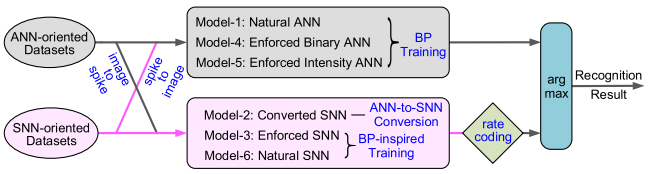

Fig. 7. Illustration of experimental flowchart.

Compute cost. The compute overhead is critical for the running latency and energy consumption. In ANNs, the compute cost is mainly determined by the MAC operations in Eq. (1), which is widely used in ANN accelerators (Chen et al., 2014, 2017). In SNNs, the major compute overhead is the spike-driven input integration in Eq. (2). Two differences from ANNs should be emphasized: (i) the costly multiplication can be removed due to the binary spike inputs if $T_w = 1$ and $K(\cdot) \equiv 1$ actually we satisfy these for simplicity, then the dendrite integration becomes multiplierless as $\sum_j w_j \cdot s_j = \sum_{j'} w_{j'}$ ($if\ s_{j'} = 1$); (ii) the integration is event-driven that implies no computation occurs if no spike received. The compute cost can be calculated by

$$\begin{cases} \text{ANN: } C = C_{mul} + C_{add} \\ \text{SNN: } C = C_{add} \end{cases} \tag{9}$$

where C_{mul} and C_{add} are the compute cost for multiplications and additions, respectively.

The compute overheads in soma (such as the activation function in ANNs and the membrane potential update and firing activity in SNNs) are ignored, which is a common way in neural network devices (Akopyan et al., 2015; Jouppi et al., 2017). Note that in SNNs, the C_{add} takes all the addition operations into account during the entire rate coding period (e.g. T), which is proportional to the total number of spike events. Similar to the memory cost evaluation, the data quantization can replace the complex high-precision MACs to efficient low-precision ones and the compression techniques (e.g. decomposition and sparsification) can additionally reduce the amount of operations. Whereas, they are not the focus of this paper as a starting point to establish a comprehensive evaluation framework.

4. Experimental results

4.1. Experimental setup

In our experiments, we comprehensively evaluate the performance of several ANN and SNN models for visual recognition over different types of benchmark workloads (ANN-oriented and SNN-oriented). As a starting work, we discuss more on the results of MLP and (plain) CNNs for simplicity and just provide the results of other models (e.g. RNNs and temporal CNNs) on SNN-oriented workloads at the end. On ANN-oriented workloads, we evaluate the performance of Model-1/2/3 on MNIST and CIFAR10 datasets. Due to the incapability of MLP in processing larger-scale CIFAR10, we just show the CNN results on this dataset. On SNN-oriented workloads, we evaluate the performance of Model-4/5/6 on the neuromorphic version of above datasets, i.e. N-MNIST and DVS-CIFAR10. Note that why we mainly compare the results across Model 1–6 implemented by ourselves is because we can easily control many factors (e.g. network structure and size, training techniques, learning hyper-parameters, etc.) to guarantee the fairness. Otherwise, it will be challenging to present a fair comparison with others due to the modeling diversity in this field.

The overall experimental flowchart is presented in Fig. 7, and the main network structures are given in Table 1. Besides, the parameter configurations for ANN-oriented and SNN-oriented workloads are provided in Tables 2 and 3, respectively. Since these workloads usually have different encoding formats for input conversion, we just show the parameter configurations of hidden layers for clarity. In all Conv layers, we set the padding value to 1. Due to the large input size of Model-4, we set the stride value to 2 there, and set the stride value to 1 in other models. In all SNN models (Model-2/3/6), each unit in the pooling layer is an independent neuron, which guarantees that the outputs of pooling layer are still in the spike format. This pooling configuration is different from previous work (Wu et al., 2018, 2019). In Model 3/6 we use the complete LIF neuron model described in Eq. (4), while in Model-2 we use the IF neuron model (without the leakage item in LIF) that is a popular choice in previous converted SNNs (Diehl et al., 2015; Sengupta et al., 2019).

We implement all the models in Pytorch framework. On MNIST and N-MNIST datasets, we adopt Adam (adaptive moment estimation (Kingma & Ba, 2014)) optimizer with default parameter setting ($\alpha = 0.0001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$); while on CIFAR10 and DVS-CIFAR10 datasets, we use SGD (stochastic gradient descent) optimizer with initial learning rate $r = 0.1$ and momentum 0.9, wherein r decays by 10x every 35 training epochs.

Table 1
Configuration of network structures.

ANN-oriented testing workloads (Model-1/2/3)	
MNIST	MLP: Input-512-10 CNN: Input-32C3-AP2-32C3-AP2-128FC-10
CIFAR10	CNN: Input-64C3-AP2-128C3-128C3-AP2-256FC-10
SNN-oriented testing workloads (Model-4/5/6)	
N-MNIST	MLP: Input-512-512-10 CNN: Input-64C3-128C3-AP2-128C3-AP2-256FC-10
DVS-CIFAR10	CNN: Input-64C3-AP2-128C3-AP2-256FC-10

Note: nC3—Conv layer with n output FMs and 3×3 kernel size, AP2—Average pooling layer with 2×2 kernel size, FC—FC layer.

Table 2
Parameter setting on ANN-oriented workloads.

Parameters	Descriptions	Model	MNIST	CIFAR10
Max epoch	–	1, 2, 3	150	150
Batch size	–	1, 2, 3	50	50
T	Simulation time window	2	500	1000
		3	20	15
u_{th}	Firing threshold	2	2	2.5
		3	0.5	0.3
τ	Decay factor	3	0.25	0.3
a	Gradient width	3	0.5	0.5

Table 3
Parameter setting on SNN-oriented workloads.

Parameters	Descriptions	Model	N-MNIST	DVS-CIFAR10
Max epoch	–	4, 5, 6	200	200
Batch size	–	4, 5, 6	50	50
T	Simulation time window	6	15	10
u_{th}	Firing threshold	6	0.3	0.25
τ	Decay factor	6	0.25	0.3
a	Gradient width	6	0.25	0.25

4.2. Accuracy analysis

Table 4 lists the accuracy results of different models on ANN-oriented workloads. On MNIST, the straightforward natural ANNs (Model-1) can achieve the best accuracy, i.e. 98.60% for MLP and 99.31% for CNN, because they are naturally compatible with the frame-based dataset. The accuracy scores of converted SNNs

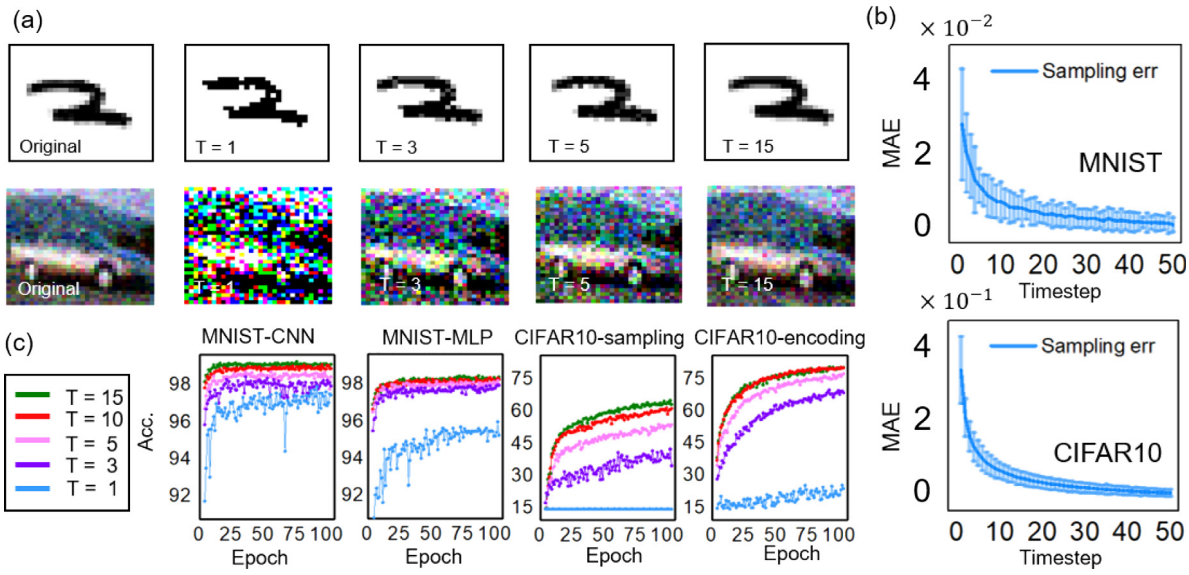

Fig. 8. Sensitivity analysis for the simulation time window T on ANN-oriented datasets: (a) sampled images; (b) mean absolute error (MAE) curves; (c) training curves of enforced SNNs.

Table 4
Accuracy on ANN-oriented workloads.

Dataset	Network	Model	Accuracy
MNIST	MLP	Model-1 (Natural ANN)	98.60%
		Model-2 (Converted SNN)[a]	98.51%
		Model-3 (Enforced SNN)[a]	98.41%
	CNN	Model-1 (Natural ANN)	99.31%
		Model-2 (Converted SNN)[a]	99.07%
		Model-3 (Enforced SNN)[a]	99.22%
CIFAR10	CNN	Model-1 (Natural ANN)	78.16%
		Model-2 (Converted SNN)[a]	76.81%
		Model-3 (Enforced SNN)[a]	63.19%
		Model-3 (Enforced SNN)[b]	74.23%

Note:
[a]Refers to the model using probabilistic sampling for input signal conversion.
[b]Refers to the model using encoding layer for input signal conversion.

based on pre-trained ANNs (Model-2) are comparable but slightly worse than those of natural ANNs. The enforced SNNs (Model-3) trained by BP-inspired algorithm on the converted spike datasets achieve the worst accuracy. Note that although we also observe that the enforced SNN can occasionally surpass the converted SNN (e.g. on the CNN structure), it has never been better than the natural ANN.

The small gap between different models on MNIST might be due to the reason that it is a simple task. To this end, we continue to test on a larger dataset, i.e. CIFAR10. In this case, the accuracy differences become more obvious, which follows 'natural ANN (Model-1) ⇒ converted SNN (Model-2) ⇒ enforced SNN (Model-3)' from the best to the worst (i.e. 78.16% ⇒ 76.81% ⇒ 63.19%). Even though the accuracy of enforced SNN can be improved to 74.23% by using an extra encoding layer (Fig. 4(b)) for signal conversion, it still cannot beat the natural ANN.

From Table 4, you can find evidence that the converted and enforced SNNs cannot beat natural ANNs. Now, we try to reveal the underlying reason for this accuracy gap. Fig. 8(a) presents the information loss when converting the original image to spike signal for consequent SNN processing (in Model-2 & Model-3). Here we take the probabilistic sampling in Fig. 4(a) as an example. We first count the total number of spikes produced by probabilistic sampling of each pixel during the simulation time window T, and then visualize the spike numbers after being normalized as images. As T increases, the images become clearer which indicates less precision loss in signal conversion. However, a too large T will cause long simulation time. In our experiments, we set $T = 15$, however, the precision loss for signal conversion still exists. Fig. 8(b) depicts the mean absolute errors (MAE) between the original pixel value and normalized spike number as well as the MAE variances due to the nondeterministic sampling. Finally, we study how T affects the training convergence, which is shown by the training curves under different T configuration. Usually, the longer T produces a faster convergence speed and better final accuracy, which is mainly due to the smaller precision loss during input signal conversion. In summary, the signal conversion from images to spikes causes information loss and makes the SNN models lose the chance to outperform the natural ANN on ANN-oriented workloads. Furthermore, the length of simulation time window T significantly affects the model convergence.

Next, let us analyze why the converted SNN also loses accuracy compared to the natural ANN. The converted SNN is based on a pre-trained ANN model. In order to be compatible with the final IF model after conversion, the pre-training of ANN usually adds some constraints, termed as constrained ANN. The constraints include removing bias, allowing only *ReLU* activation function, changing max pooling to average pooling, and so on Diehl et al. (2015) and Hunsberger and Eliasmith (2016). These constraints usually cause accuracy loss as shown in Fig. 9(a). Then the conversion from the pre-trained ANN to the converted SNN will suffer from further precision loss during the aforementioned signal conversion and the newly incurred parameter adaption such as weight/activation normalization and threshold tuning. On CIFAR10, the accuracy loss for model conversion is significantly larger than that on small MNIST. The influence of simulation time window T is shown in Fig. 9(b), wherein the smaller T produces significantly larger accuracy loss. Therefore, the converted SNN usually requires a much longer T to recover the accuracy. In summary, due to the constrained pre-training and lossy model conversion, the converted SNN cannot outperform the natural ANN. More and more recent works benchmark the SNN performance using converted SNNs on ANN-oriented datasets (Diehl et al., 2015; Hu et al., 2018; Sengupta et al., 2019). Although this way demonstrates that SNNs can still work well on these ANN-oriented workloads, it also makes SNNs lose the opportunity to be the winner.

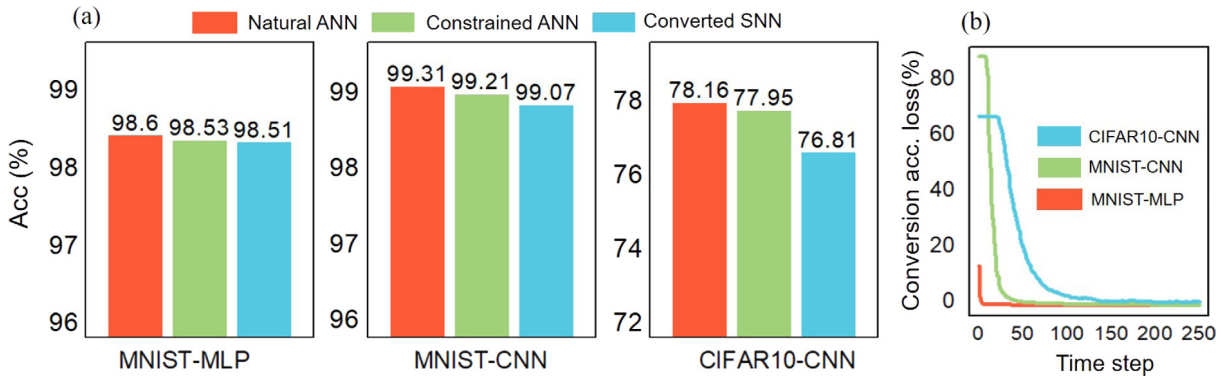

Fig. 9. Accuracy analysis for the converted SNNs on ANN-oriented datasets: (a) testing accuracy comparison among the natural ANN, constrained ANN, and converted SNN; (b) conversion accuracy loss curve recording the accuracy gap between the natural ANN and the converted SNN as time step increases.

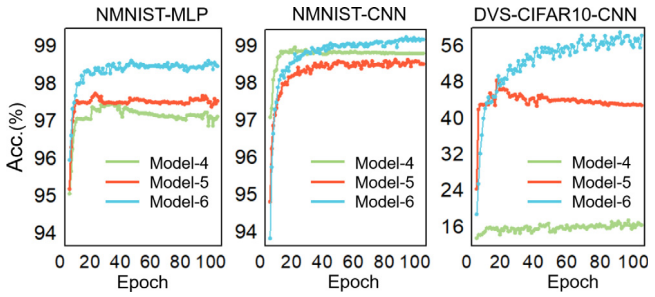

Fig. 10. Training curves of the enforced binary ANN (Model 4), enforced intensity ANN (Model 5), and natural SNN (Model 6) on different datasets.

Table 5
Accuracy on SNN-oriented workloads.

Dataset	Network	Model	Accuracy
N-MNIST	MLP	Model-4 (Enforced Binary ANN)	97.24%
		Model-5 (Enforced Intensity ANN)	97.63%
		Model-6 (Natural SNN)	98.61%
	CNN	Model-4 (Enforced Binary ANN)	99.08%
		Model-5 (Enforced Intensity ANN)	98.63%
		Model-6 (Natural SNN)	99.42%
DVS-CIFAR10	CNN	Model-4 (Enforced Binary ANN)	17.60%
		Model-5 (Enforced Intensity ANN)	51.10%
		Model-6 (Natural SNN)	60.30%

Table 5 lists the accuracy results of different models on SNN-oriented workloads. The natural SNNs (Model-6) can always achieve the best accuracy (98.61% for MLP on N-MNIST, 99.42% for CNN on N-MNIST, and 60.30% for CNN on DVS-CIFAR10) on these networks and datasets, which owes to the natural compatibility with frame-free spike events. In the enforced ANNs, the intensity image-based ANN (Model-5) usually performs better than the binary image-based one (Model-4) except for one occasional case of the CNN on N-MNIST. The reason is similar to that on ANN-oriented workloads, because the relative simplicity of N-MNIST narrows the accuracy gaps between different models. The corresponding training curves of all the models in Table 5 are presented in Fig. 10. Overall, on these SNN-oriented workloads, the natural SNNs are able to converge better than other ones converted to process in ANN domain.

We attempt to explain the accuracy results in a comprehensive way. Fig. 11 visualizes the different inputs injected to the three models (Model-4/5/6) on SNN-oriented workloads. For the first two models, the spike events are converted to binary images or intensity images according to the signal conversion methods in Fig. 4(c) or (d), respectively. Since Model-4 directly receives each spike pattern (0/1) as a binary image, the features are not intuitive and are too sparse as shown in Fig. 11(a). Differently, Model-5 accumulates the spikes for each pixel along the

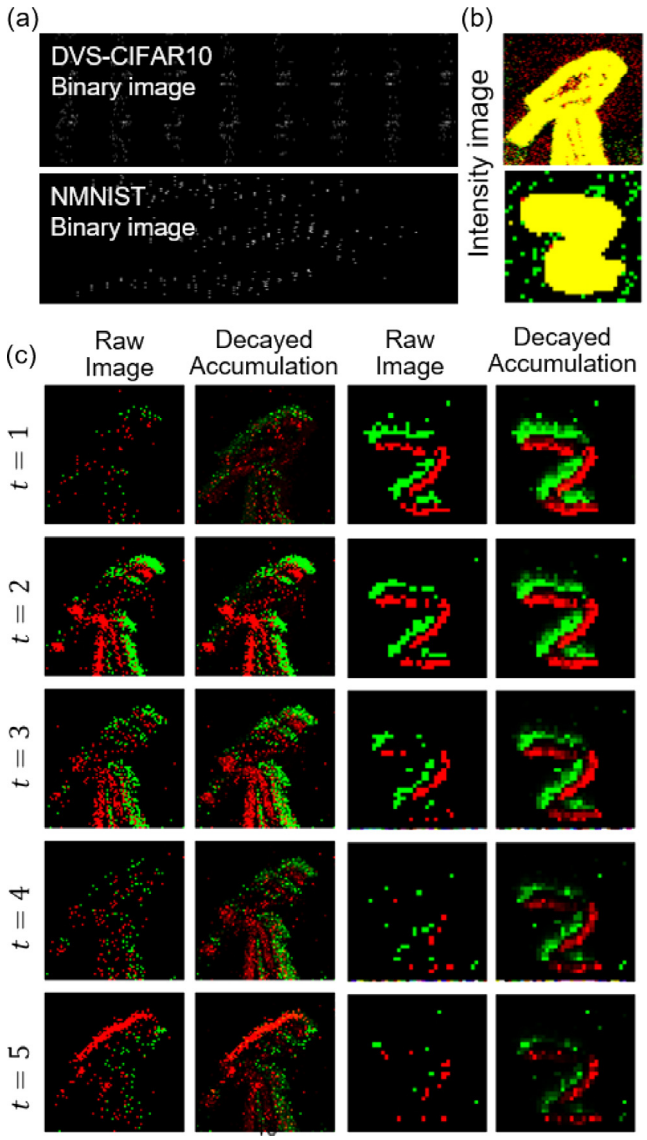

Fig. 11. Visualization of the input data carried by (a) binary images for the enforced binary ANN (Model-4), (b) intensity images for the enforced intensity ANN (Model-5), and (c) raw & accumulated (with decay) spike patterns for the natural SNN (Model-6).

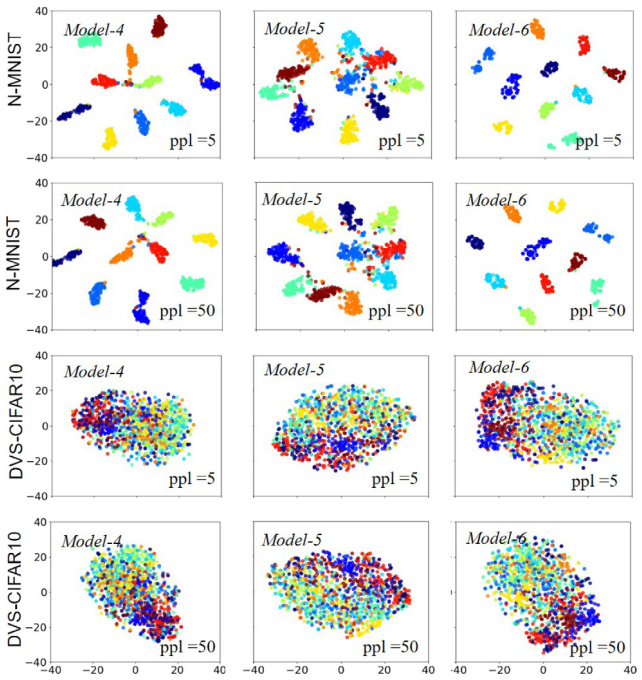

Fig. 12. 2D embedding visualization by t-SNE (Matten & Hinton, 2008) for the activations (in ANNs) or the firing rates (in SNNs) of the last hidden layer in the enforced binary ANN (Model 4), enforced intensity ANN (Model 5), and natural SNN (Model 6), with only CNN structures for simplicity. Here we use different perplexity values and datasets.

temporal dimension and then normalizes the spike number to be an intensity value. As Fig. 11(b) depicts, the object contour can be kept to a great extent, however, the detailed sharp features are blurred. In contrast to the above two frame-based models, Model-6 naturally receives the spike events. The raw spike pattern at every time step (actually it still has to aggregate the spike events of every 3 ms to narrow the total simulation time length) is visualized in the odd columns of Fig. 11(c). Recalling Eq. (4), the membrane potential in SNNs actually accumulated all the previous inputs with a temporal decay factor if we ignore the potential reset after firing for simplicity. To mimic the equivalent inputs finally injected onto the membrane potential, we plot the decayed accumulation of previous spike patterns at every time step and show them in the even columns of Fig. 11(c). It can be seen that the natural SNN (Model-6) is well capable of remaining the detail features contained in the spike events, which is one reason to explain why it can perform better on SNN-oriented workloads.

Besides the visualization of input signals, we further visualize the activations (in ANNs) and the firing rates (in SNNs) to compare the recognition capability of these three models. Fig. 12 provides the 2D embedding results generated by a nonlinear dimensionality reduction algorithm, t-distributed stochastic neighbor embedding (t-SNE) (Matten & Hinton, 2008). It can project the high-dimensional embedding down to a low-dimensional one for data visualization. Due to the sensitivity of t-SNE to the setting of perplexity (ppl) value, we adopt two ppl values (ppl = 5, ppl = 50) on each dataset according to the suggested range by Matten and Hinton (2008). In this way, we project the 256-dimensional activations or firing rates of the last hidden layer down to two dimensions. Points with the same color have the same label and with different color belong to different classes. Therefore, more concentrated points of the same color and more divergent points of different color reflect better classification performance of the model. Although different ppl setting indeed causes pattern change, the relative clustering effects between three models remain consistent as ppl value varies. On N-MNIST, three models are all able to project data from different classes into different point clusters. Compared with Model-4 and Model-5, Model-6 can attract the points within each class more compact and push classes farther, which indicates that Model-6 performs the best. On DVS-CIFAR10, all models present large cluster overlaps. This implies the incapability to achieve comparable accuracy scores like those on the simple N-MNIST task. It is obvious that the clustering effect from Model-4 to Model-6 is gradually strengthened. Combining the results on N-MNIST and DVS-CIFAR10, it confirms the superiority of Model-6 on SNN-oriented workloads, which is consistent with the results in Table 5. Note that the classification performance of models has been already demonstrated in Tables 4 and 5, and the t-SNE simulation is just an auxiliary way of visualization for better understanding (not the focus of this work). This is the reason why we only show the t-SNE results for Table 5 with Model 4/5/6 on SNN-oriented datasets and do not repeat the same experiments for Table 4 with Model 1/2/3.

4.3. Cost analysis

In this subsection, we count the memory and compute costs for all models in Tables 4 and 5. The cost calculation is according to Eqs. (8)–(9). The results are shown in Table 6, wherein the cost values are the average results across all testing samples during inference.

Memory cost analysis. For all ANN models, the memory cost mainly includes the weight memory and activation memory. Here we omit some possible intermediate variables (e.g. partial sums) for simplicity, since they are related to the hardware architecture design (e.g. dataflow mapping). We also ignore the memory cost for bias because it occupies only a small fraction of the total memory especially in CNNs. On ANN-oriented workloads, the memory cost of SNNs is comparable with that of ANNs. Specifically, the amount of membrane potentials in SNNs is the same with that of activations in ANNs, but the extra spikes of SNNs still consume memory. Fortunately, the spike is in the binary format, i.e. only one bit, the memory cost of which is negligible. On SNN-oriented workloads, similar conclusions can be drawn, except for the Model-4 (enforced binary ANN). The input size of Model-4 is much larger than Model-5/6 (see Fig. 4), which significantly increases the feature map size and memory/compute cost. Note that the memory cost of ANNs will be increased if we consider the bias memory, especially in the case of MLP networks.

Compute cost analysis. Different from the memory cost determined by the time step with the maximum memory consumption, the compute cost takes the total operations across all time steps within the entire simulation time window (T) into account. Two major conclusions are observed: (1) All SNNs can remove the costly multiplication operations (under the mentioned condition of $T_w = 1$) which is very helpful for power and energy reduction; (2) The number of addition operations in SNNs might exceed that of ANNs or at least in the same level because SNNs require a processing duration, i.e. T, to accomplish a practical task under the rate coding scheme. The first conclusion does not include Model-3 with encoding layer, whose first layer works in the ANN–SNN hybrid mode, leading to extra multiplications and a bit more additions. The second conclusion is a little counter-intuitive against the common sense that the firing activity of SNNs is very sparse, i.e. should be low-cost. It is correct that the spike events are very sparse at each time step, which results in the low-power characteristic of SNN-based devices. However, the compute cost here is determined by the total activities across all time steps, which is more close to the energy consumption rather than power.

Table 6
Memory and compute cost on all workloads.

Model	Network	#W	#A/P	#S	M (bits)	#Add	#Mul
MNIST ($units = 10^6$)							
Model-1 (Natural ANN)	MLP	0.4065	0.0005	0	13.0256	0.4065	0.4065
Model-2 (Converted SNN)[a]	MLP	0.4065	0.0005	0.0000	13.0256	24.325	0
Model-3 (Enforced SNN)[a]	MLP	0.4065	0.0005	0.0001	13.0257	1.202	0
Model-1 (Natural ANN)	CNN	0.2115	0.0393	0	8.0259	2.2655	2.2420
Model-2 (Converted SNN)[a]	CNN	0.2115	0.0393	0.0058	8.0317	35.9342	0
Model-3 (Enforced SNN)[a]	CNN	0.2115	0.0393	0.0097	8.0356	7.9147	0
CIFAR10 ($units = 10^6$)							
Model-1 (Natural ANN)	CNN	2.3226	0.1559	0	79.3120	60.4923	60.4923
Model-2 (Converted SNN)[a]	CNN	2.3226	0.1559	0.0312	79.3432	1580.4851	0
Model-3 (Enforced SNN)[a]	CNN	2.3226	0.1559	0.0236	79.3356	127.4985	0
Model-3 (Enforced SNN)[b]	CNN	2.3226	0.1559	0.0243	79.3375	152.9086	26.5421
N-MNIST ($units = 10^6$)							
Model-4 (Enforced Binary ANN)	MLP	118.6417	0.0010	0	3796.5661	118.6417	118.6417
Model-5 (Enforced Intensity ANN)	MLP	1.4510	0.0010	0	46.464	1.4510	1.4510
Model-6 (Natural SNN)	MLP	1.4510	0.0010	0.0003	46.4655	1.4419	0
Model-4 (Enforced Binary ANN)	CNN	7.3022	6.1312	0	429.8688	1235.1050	1233.6164
Model-5 (Enforced Intensity ANN)	CNN	2.3220	0.3044	0	84.0448	131.4607	131.3210
Model-6 (Natural SNN)	CNN	2.3220	0.3044	0.0092	84.054	104.1391	0
DVS-CIFAR10 ($units = 10^6$)							
Model-4 (Enforced Binary ANN)	CNN	43.3306	7.9026	0	1639.4624	490.5503	485.8083
Model-5 (Enforced Intensity ANN)	CNN	3.3542	0.2106	0	114.0736	37.9949	37.8666
Model-6 (Natural SNN)	CNN	3.3542	0.2106	0.0194	114.093	23.142	0

Note: #—amount, W—weight, A—activation, P—membrane potential, S—spike, M—memory (total), Add—addition, Mul—multiplication. Mul is much more costly than Add in hardware implementation.
[a]Refers to the model using probabilistic sampling for input signal conversion.
[b]Refers to the model using encoding layer for input signal conversion.

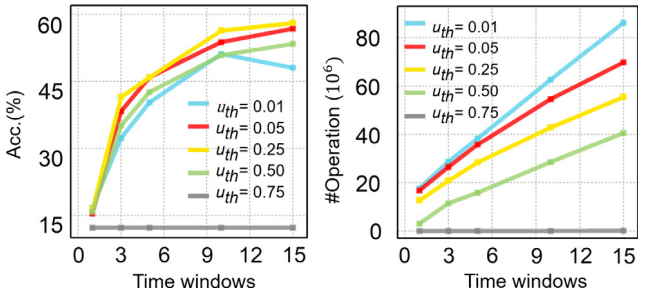

Fig. 13. The trade-off between accuracy and compute cost of Model-6 over DVS-CIFAR10 dataset, influenced by the simulation time window T and firing threshold u_{th}.

On ANN-oriented workloads, the amount of addition operations of SNNs is higher than that of ANNs due to the mentioned time window for rate coding. Specifically, we find Model-3 of enforced SNN consumes less additions than Model-2 of converted SNN. This is because Model-2 requires longer time window (up to 66.7x, see Table 2) to recover the accuracy loss after model conversion. On SNN-oriented workloads, Model-4 of enforced binary ANN consumes much higher compute cost due to the larger feature size as mentioned earlier. Model-6 of natural SNN removes the costly multiplications; moreover, it consumes lower additions than Model-5 of enforced intensity ANN, which presents an opposed observation compared to the case of Model-1/2/3. This indicates that the spike train of neuromorphic datasets (e.g. N-MNIST and DVS-CIFAR10) is much sparser than that of converted ANN-oriented datasets (e.g. MNIST and CIFAR10). Therefore, the SNN models on neuromorphic datasets can suppress the extra cost caused by the rate coding window.

From the above analyses, we know that the firing activities in SNNs will greatly affect the accuracy and compute cost. Thus we further conduct sensitivity analysis shown in Fig. 13, taking Model-6 over DVS-CIFAR10 dataset as an example. We omit the memory cost since it is only determined by the maximum amount of spike events at a certain time step rather than the total spikes across the entire time window. Two factors, i.e. T and u_{th}, are taken into account. We can get two observations: (1) Apparently, a larger T can usually bring higher accuracy but causes more compute cost; (2) A properly larger u_{th} could improve both the accuracy and compute cost simultaneously. A too small u_{th} will produce much more spikes that not only increases the compute cost but also decreases the model accuracy due to lower pattern discriminability. Although a too large u_{th} could sparsify the spikes significantly, it will compromise the network's expression ability leading to lower accuracy.

4.4. Additional comparison

Besides SNNs, both RNNs and temporal CNNs in the neural network family are also able to process temporal information. Therefore, here we provide an additional comparison of RNNs, temporal CNNs, and SNNs on N-MNIST and DVS-CIFAR10 datasets, as shown in Tables 7–8. We implement plain RNN, long short term memory (LSTM), and temporal CNN models, which are in non-spiking version, as well as SNNs in spiking version on these two datasets. We also reference other reported results. All the network structures are detailed to help understanding.

First, our SNN results are advanced compared with previously reported references, benefit from our selection of superior training algorithm and signal conversion method. The minor accuracy degradation compared to the recent work (Wu et al., 2019) is because we use a smaller network and do not adopt the neuron normalization and voting-based classification techniques. We would like to emphasize that the focus of this work is to give a comprehensive benchmarking methodology rather than beating prior work, so we abandon the possible optimization techniques. Our LSTM result is consistent with prior work (Liu et al., 2018), which also reflects the reliability of our implementation.

Second, regarding the comparison between SNNs and RNN (LSTM)/temporal CNN models, it seems that the natural SNN

Table 7
Accuracy comparison of RNNs, temporal CNNs, and SNNs on N-MNIST dataset.

Model	Structure	N-MNIST
LSTM (Liu et al., 2018)	Input-110–10	97.05%
Phased-LSTM (Liu et al., 2018)	Input-110–10	97.38%
RNN (this work)	Input-128–10	96.08%
LSTM (this work)	Input-128–10	97.13%
Natural SNN (this work)	Input-128–10	98.52%
Natural SNN (Wu et al., 2019)	InputL-128C3-128C3-AP2 -128C3-256C3-AP2-1024FC-10	99.53%
Temporal CNN	InputL-64C3-128C3-AP2 -128C3-AP2-256FC-10	99.47%
Natural SNN (this work)	InputL-64C3-128C3-AP2 -128C3-AP2-256FC-10	99.42%

Note: In temporal CNN, the convolution and pooling here are 3D operations. Here C3 refers to $3 \times 3 \times 3$ convolution, and AP2 refers to $2 \times 2 \times 2$ average pooling.

Table 8
Accuracy comparison of RNNs, temporal CNNs, and SNNs on DVS-CIFAR10 dataset.

Model	Structure	DVS-CIFAR10
RNN (this work)	Input-128–10	23.8%
LSTM (this work)	Input-128–10	28.7%
Natural SNN (this work)	Input-128–10	32.1%
Natural SNN (Wu et al., 2019)	InputL-128C3-128C3-AP2 -128C3-256C3-AP2-1024FC-10	60.5%
Temporal CNN	InputL-64C3-AP2- 128C3-AP2-256FC-10	53.7%
Natural SNN (this work)	InputL-64C3-AP2- 128C3-AP2-256FC-10	60.3%

Note: In temporal CNN, the convolution and pooling here are 3D operations. Here C3 refers to $3 \times 3 \times 3$ convolution, and AP2 refers to $2 \times 2 \times 2$ average pooling.

model (Model-6) has greater potential to perform better. Although our natural SNN performs slightly worse than the temporal CNN on N-MNIST, we should note that the latter has much more weight parameters due to the 3D convolution. We try our best to explain the advantages of SNNs on neuromorphic workloads as follows: (1) The temporal integration of historical information can naturally maintain the detail features of sparse data better (see Fig. 11); (2) The leakage and thresholded firing mechanism in each neuron can synergistically help denoise redundant information and thus remain useful features (Evangelos et al., 2015; Gutig, 2016).

5. Conclusion and discussion

5.1. Brief summary

In this work, we design well-rounded experiments to answer the questions of "what workloads are ideal for SNNs and how to evaluate SNNs makes sense". Specifically, we conduct extensive workload analyses using different benchmark datasets (ANN-oriented and SNN-oriented), processing models (ANNs and SNNs), signal conversion methods (images⇔spikes), and learning algorithms (direct and indirect supervised training). Comprehensive evaluation metrics are proposed in consideration of the trade-off between the application accuracy and the memory/compute cost.

Through a variety of comparisons, visualizations, and sensitivity studies, we give the following modeling insights:

- On simple ANN-oriented workloads (e.g. MNIST), Model-3 (enforced SNN) is a better choice with acceptable accuracy and less compute cost (without costly multiplications and with slightly more additions).
- On more complicated ANN-oriented workloads (e.g. CIFAR10), Model-1 (natural ANN) is preferred to maintain the model accuracy. Although it needs multiplications, the amount of additions is the least one.
- On SNN-oriented workloads, Model-6 (natural SNN) is the best selection with both higher accuracy and lower compute cost.

The converted SNNs on one hand compromise the accuracy compared to their original ANN counterparts due to the extra constraints during ANN pre-training and lossy signal conversion & parameter adaption during SNN conversion; on the other hand also lose efficiency compared to the enforced SNNs due to the much longer time window for accuracy recovery. In SNN modeling, the sparse activities of SNN-oriented datasets significantly benefit the reduction of compute cost, and a properly larger firing threshold can improve both the accuracy and compute cost. In addition, we for the first time point out in the following discussions that the direct porting of ANNs workloads for the evaluation of SNN is unwise although many works are doing so, and a benchmarking framework for SNNs to cover broader tasks, datasets and metrics is urgently needed.

5.2. Future opportunities

Coding schemes and learning algorithms. The application accuracy and execution cost of SNNs are mainly determined by the coding scheme and learning algorithm. Studying new coding schemes beyond the rate coding is an attractive topic to narrow the time window (i.e. towards faster response and lower compute cost) without compromising the expressive power (i.e. maintaining the accuracy). Powerful learning algorithms are also required to further explore the potential of these coding schemes. Moreover, modifying the simple MSE loss function is also a promising way to further improve the network classification performance. All these directions are helpful for training SNNs with less compute cost or extending current models to larger scale, e.g. on ImageNet-level tasks (Deng et al., 2009).

Datasets, benchmarks, metrics, and applications. In this work, we highlight that it is unwise to intuitively test SNNs

using ANN-oriented benchmark datasets. More aggressively, it is interesting to ask a question whether current SNN-oriented datasets are neuromorphic enough, i.e. whether they can help fully explore the characteristics of SNNs. In Iyer, Chua, and Li (2018), the authors claim that SNNs fail to outperform ANNs even if on neuromorphic datasets (e.g. N-MNIST) and the temporal information in current neuromorphic datasets is not important. We find that their comparison is a little unfair since the network size of the ANNs they use is larger than the SNN baseline. But they still open an interesting question for the neuromorphic community: what is and how to build a good neuromorphic dataset? Despite that there exist different types of generation method for neuromorphic datasets (Li et al., 2017; Mueggler, Rebecq, Gallego, Delbruck, & Scaramuzza, 2017; Orchard et al., 2015; Serrano-Gotarredona & Linares-Barranco, 2015), scanning static image datasets into the spiking version using a DVS-like device is still the current common practice. However, it might not be the best way to exploit the spatio-temporal processing capability of SNNs.

Second, most of the existing benchmarks are based on visual tasks, especially the image recognition. They are insufficient to showcase the capability of SNNs in processing temporal or undeterministic information. We need a broader set of benchmark tasks for SNNs to drive research innovations. Besides visual recognition applications, processing auditory and language information with intrinsic temporal properties or finding fast solution of optimization problems that do not require high solution accuracy, for example, might be well-suited for SNNs. As another example, object detection (Binas, Neil, Liu, & Delbruck, 2017; Zhu, Yuan, Chaney, & Daniilidis, 2018) in high-volume videos is promising to benefit much from the event-drive paradigm of SNNs towards low cost. Furthermore, more evaluation metrics beyond the proposed recognition accuracy and memory/compute cost are also needed. For instance, how to assess the capability of temporal association, memorization capacity, fault tolerance, and practical running efficiency on devices are all interesting.

Furthermore, recognizing images is just one out of a huge number of tasks the brain can perform. Extensive efforts to explore more functionality potentials of SNNs should be conducted. One promising direction is to study the network dynamics from a more global angle. Each neuron in SNNs is a dynamic model with firing activity that distinguishes from ANNs, which can bring rich response behaviors in the temporal domain. For example, an SNN could memorize multiple scenarios encoded in metastable states and can output the corresponding response trajectory under different stimuli. Even if for one stimulus, changing the sequence of input spikes can also produce a totally different response. Therefore, how to accurately control the response trajectory via external stimuli is interesting for the exploration of memorization behavior. Overall, SNNs have a huge behavior space that merits in-depth analysis in the future. In this paper we just provide several points to start a discussion. More investigations are expected, requiring efforts from the entire community.

Acknowledgments

The work was partially supported by National Science Foundation (Grant No. 1725447), Tsinghua University Initiative Scientific Research Program, Tsinghua-Foshan Innovation Special Fund (TFISF), and National Science Foundation of China (Grant No. 61876215).

References

Abdel-Hamid, O., Mohamed, A.-r., Jiang, H., Deng, L., Penn, G., & Yu, D. (2014). Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *22*(10), 1533–1545.

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., et al. (2015). Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *34*(10), 1537–1557.

Banner, R., Hubara, I., Hoffer, E., & Soudry, D. (2018). Scalable methods for 8-bit training of neural networks, arXiv preprint arXiv:1805.11046.

Binas, J., Neil, D., Liu, S. C., & Delbruck, T. (2017). Ddd17: End-to-end davis driving dataset.

Caglayan, A., & Burak Can, A. (2018). Exploiting multi-layer features using a cnn-rnn approach for rgb-d object recognition, In: Proceedings of the european conference on computer vision (ECCV).

Cao, Y., Chen, Y., & Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, *113*(1), 54–66.

Chen, T., Du, Z., Sun, N., Wang, J., Wu, C., Chen, Y., et al. (2014). Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ACM sigplan notices, Vol. 49* (pp. 269–284). ACM.

Chen, Y.-H., Krishna, T., Emer, J. S., & Sze, V. (2017). Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, *52*(1), 127–138.

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

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *Computer vision and pattern recognition, 2009. CVPR 2009. IEEE conference on* (pp. 248–255). IEEE.

Deng, L., Jiao, P., Pei, J., Wu, Z., & Li, G. (2018). Gxnor-net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework. *Neural Networks*, *100*, 49–58.

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

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., & Pfeiffer, M. (2015). Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *Neural networks (IJCNN), 2015 international joint conference on* (pp. 1–8). IEEE.

Esser, S. K., Merolla, P. A., Arthur, J. V., Cassidy, A. S., Appuswamy, R., Andreopoulos, A., et al. (2016). Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences*, *113*(41), 11441–11446.

Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., et al. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, *542*(7639), 115.

Evangelos, S., Daniel, N., Michael, P., Francesco, G., Furber, S. B., & Shih-Chii, L. (2015). Robustness of spiking deep belief networks to noise and reduced bit precision of neuro-inspired hardware platforms. *Frontiers in Neuroscience*, *9*.

Furber, S. B., Galluppi, F., Temple, S., & Plana, L. A. (2014). The spinnaker project. *Proceedings of the IEEE*, *102*(5), 652–665.

Gerstner, W., Kistler, W. M., Naud, R., & Paninski, L. (2014). *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press.

Ghaeini, R., Hasan, S. A., Datla, V., Liu, J., Lee, K., Qadir, A., et al. (2018). Dr-bilstm: Dependent reading bidirectional lstm for natural language inference, arXiv preprint arXiv:1802.05577.

Ghosh-Dastidar, S., & Adeli, H. (2009). Spiking neural networks. *International Journal of Neural Systems*, *19*(04), 295–308.

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., et al. (2016). Hybrid computing using a neural network with dynamic external memory. *Nature*, *538*(7626), 471.

Gutig, R. (2016). Spiking neurons can discover predictive features by aggregate-label learning. *Science*, *351*(6277), aab4113.

Haessig, G., Cassidy, A., Alvarez, R., Benosman, R., & Orchard, G. (2017). Spiking optical flow for event-based sensors using ibm's truenorth neurosynaptic system.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition, In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778.

Hodgkin, A. L., & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, *117*(4), 500–544.

Hu, B., Lu, Z., Li, H., & Chen, Q. (2014). Convolutional neural network architectures for matching natural language sentences, In: Advances in neural information processing systems, pp. 2042–2050.

Hu, Y., Tang, H., Wang, Y., & Pan, G. (2018). Spiking deep residual network, arXiv preprint arXiv:1805.01352.

Hunsberger, E., & Eliasmith, C. (2016). Training spiking deep networks for neuromorphic hardware.

Hwu, T., Isbell, J., Oros, N., & Krichmar, J. (2017). A self-driving robot using deep convolutional neural networks on neuromorphic hardware. In *2017 international joint conference on neural networks (IJCNN)* (pp. 635–641). IEEE.

Iyer, L. R., Chua, Y., & Li, H. (2018). Is neuromorphic mnist neuromorphic? analyzing the discriminative power of neuromorphic datasets in the time domain, arXiv preprint arXiv:1807.01013.
Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, *14*(6), 1569–1572.
Jahani, B., & Mohammadi, B. (2018). A comparison between the application of empirical and ann methods for estimation of daily global solar radiation in iran. *Theoretical and Applied Climatology*, 1–13.
Jin, Y., Li, P., & Zhang, W. (2018). Hybrid macro/micro level backpropagation for training deep spiking neural networks, arXiv preprint arXiv:1805.07866.
Jonke, Z., Habenschuss, S., & Maass, W. (2016). Solving constraint satisfaction problems with networks of spiking neurons. *Frontiers in Neuroscience*, *10*, 118.
Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., et al. (2017). In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture* (pp. 1–12). ACM.
Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *Computer Science*.
Krizhevsky, A., & Hinton, G. (2009). *Learning multiple layers of features from tiny images*. Citeseer.
Lam, M. W., Chen, X., Hu, S., Yu, J., Liu, X., & Meng, H. (2019). Gaussian process lstm recurrent neural network language models for speech recognition. In *ICASSP 2019-2019 IEEE international conference on acoustics, speech and signal processing (ICASSP)* (pp. 7235–7239). IEEE.
LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436.
LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.
Lee, J. H., Delbruck, T., & Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, *10*, 508.
Lee, C., Panda, P., Srinivasan, G., & Roy, K. (2018). Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning. *Frontiers in Neuroscience*, *12*.
Li, H., Kadav, A., Durdanovic, I., Samet, H., & Graf, H. P. (2016). Pruning filters for efficient convnets, arXiv preprint arXiv:1608.08710.
Li, H., Liu, H., Ji, X., Li, G., & Shi, L. (2017). Cifar10-dvs: An event-stream dataset for object classification. *Frontiers in Neuroscience*, *11*.
Lichtsteiner, P., Posch, C., & Delbruck, T. (2008). A 128× 128 120 db 15 μs latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, *43*(2), 566–576.
Liu, L., Deng, L., Hu, X., Zhu, M., Li, G., Ding, Y., et al. (2018). Dynamic sparse graph for efficient deep learning, arXiv preprint arXiv:1810.00859.
Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, *10*(9), 1659–1671.
Maass, W. (2014). Noise as a resource for computation and learning in networks of spiking neurons. *Proceedings of the IEEE*, *102*(5), 860–880.
Matten, L. v. d., & Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research (JMLR)*, *9*, 2579–2605.
Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, *345*(6197), 668–673.
Mozafari, M., Ganjtabesh, M., Nowzari-Dalini, A., Thorpe, S. J., & Masquelier, T. (2018). Combining stdp and reward-modulated stdp in deep convolutional spiking neural networks for digit recognition arXiv preprint arXiv:1804.00227.
Mueggler, E., Rebecq, H., Gallego, G., Delbruck, T., & Scaramuzza, D. (2017). The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam. *International Journal of Robotics Research*, (49).
Novikov, A., Podoprikhin, D., Osokin, A., & Vetrov, D. P. (2015). Tensorizing neural networks, In: Advances in neural information processing systems, pp. 442–450.
Orchard, G., Jayawant, A., Cohen, G. K., & Thakor, N. (2015). Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, *9*(178).
Pengjie Gu, G. P., & Tang, H. (2019). Stca: Spatio-temporal credit assignment with delayed feedback in deep spiking neural networks, In: International joint conferences on artificial intelligence.
Redmon, J., & Farhadi, A. (2017). Yolo9000: better, faster, stronger, arXiv preprint.
Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., & Liu, S.-C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, *11*, 682.
Sengupta, A., Ye, Y., Wang, R., Liu, C., & Roy, K. (2019). Going deeper in spiking neural networks: VGG and residual architectures. *Frontiers in Neuroscience*, *13*.
Serrano-Gotarredona, T., & Linares-Barranco, B. (2015). Poker-dvs and mnist-dvs. their history, how they were made, and other details. *Frontiers in Neuroscience*, *9*(437).
Shi, G., Liu, Z., Wang, X., Li, C. T., & Gu, X. (2017). Object-dependent sparse representation for extracellular spike detection. *Neurocomputing*, *266*, 674–686.
Shi, L., Pei, J., Deng, N., Wang, D., Deng, L., Wang, Y., et al. (2015). Development of a neuromorphic computing system, In: 2015 IEEE international electron devices meeting (IEDM), pp. 4.3.1–4.3.4.
Shrestha, S. B., & Orchard, G. (2018). Slayer: Spike layer error reassignment in time, In: Advances in neural information processing systems, pp. 1412–1421.
Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489.
Tavanaei, A., & Maida, A. S. (2017). Bp-stdp: Approximating backpropagation using spike timing dependent plasticity, arXiv preprint arXiv:1711.04214.
Vidal, A. R., Rebecq, H., Horstschaefer, T., & Scaramuzza, D. (2018). Ultimate slam? combining events, images, and imu for robust visual slam in hdr and high speed scenarios. *IEEE Robotics & Automation Letters*, *3*(2) 994–1001.
Wu, Y., Deng, L., Li, G., Zhu, J., & Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in Neuroscience*, *12*.
Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., & Shi, L. (2019). Direct training for spiking neural networks: Faster, larger, better, In: Proceedings of the AAAI conference on artificial intelligence, Vol. 33, pp. 1311–1318.
Yin, S., Ouyang, P., Tang, S., Tu, F., Li, X., Zheng, S., et al. (2017). A high energy efficient reconfigurable hybrid neural network processor for deep learning applications. *IEEE Journal of Solid-State Circuits*, *53*(4), 968–982.
Young, T., Hazarika, D., Poria, S., & Cambria, E. (2018). Recent trends in deep learning based natural language processing. *IEEE Computational Intelligence Magazine*, *13*(3), 55–75.
Zhang, W., Bai, Z., & Zhu, Y. (2019). An improved approach based on cnn-rnns for mathematical expression recognition. In *Proceedings of the 2019 4th international conference on multimedia systems and signal processing* (pp. 57–61). ACM.
Zhu, A. Z., Yuan, L., Chaney, K., & Daniilidis, K. (2018). Ev-flownet: Self-supervised optical flow estimation for event-based cameras.
Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition, In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 8697–8710.