Backpropagation-based learning techniques for deep spiking neural networks: a survey

Manon Dampfhoffer, Thomas Mesquida, Alexandre Valentian, Lorena Anghel

Backpropagation-based Learning Techniques for Deep Spiking Neural Networks: A survey

Manon Dampfhoffer*†, Thomas Mesquida†, Alexandre Valentian†, Lorena Anghel*
*Univ. Grenoble Alpes, CEA, CNRS, Grenoble INP, INAC-Spintec, 38000 Grenoble, France
†Univ. Grenoble Alpes, CEA, List, F-38000 Grenoble, France
manon.dampfhoffer@cea.fr, thomas.mesquida@cea.fr, alexandre.valentian@cea.fr, lorena.anghel@grenoble-inp.fr

***Abstract*—With the adoption of smart systems, Artificial Neural Networks (ANNs) have become ubiquitous. Conventional ANN implementations have a high energy consumption, limiting their use in embedded and mobile applications. Spiking Neural Networks (SNNs) mimic the dynamics of biological neural networks by distributing information over time through binary spikes. Neuromorphic hardware has emerged to leverage the characteristics of SNNs, such as asynchronous processing and high activation sparsity. Therefore, SNNs have recently gained interest in the machine learning community as a brain-inspired alternative to ANNs for low-power applications. However, the discrete representation of the information makes the training of SNNs by backpropagation-based techniques challenging. In this survey, we review training strategies for deep SNNs targeting deep learning applications such as image processing. We start with methods based on the conversion from an ANN to a SNN and compare these with backpropagation-based techniques. We propose a new taxonomy of spiking backpropagation algorithms into three categories, namely: spatial, spatio-temporal and single-spike approaches. In addition, we analyze different strategies to improve accuracy, latency and sparsity, such as regularization methods, training hybridization and tuning of the parameters specific to the SNN neuron model. We highlight the impact of input encoding, network architecture and training strategy on the accuracy-latency trade-off. Finally, in the light of the remaining challenges for accurate and efficient SNNs solutions, we emphasize the importance of joint hardware-software co-development.**

***Index Terms*—Deep learning, spiking neural networks, brain-inspired computing, neuromorphic computing.**

I. Introduction

ARTIFICIAL Neural Networks (ANNs) are machine-learning algorithms inspired by the computations done by the brain while solving complex tasks. Deep ANNs (with more than a few hidden layers) have been successfully used in many applications such as image recognition [1], object detection [2], speech recognition [3], medical diagnosis [4], game playing [5], etc. ANNs can solve difficult tasks using a large amount of data and large network architectures. This comes at the cost of a high energy consumption [6], which limits their use in embedded and even more in mobile applications. It is known that biological neural networks in the human brain consumes very few energy while performing much complex tasks, which motivates brain-inspired computing.

Spiking Neural Networks (SNNs) are among the new generations of ANNs pushing the brain-inspired approach. In the brain, neurons use electrical pulses to transmit information through the synapses in a sparse and asynchronous manner. Similarly, the neurons of SNNs communicate through temporal events (called spikes) and follow Integrate and Fire dynamics [7], [8]. Information can be encoded sparsely in the temporal dimension, in the spike rates or in the timing of spike emissions, and processed asynchronously. SNNs are sometimes seen as Recurrent Neural Networks (RNNs) in a wide sense [9], as the state of a spiking neuron depends on its previous state, therefore naturally introducing recurrence. SNNs are also often compared to Binarized Neural Networks (BNNs), which use binarized weights and activations. However, in RNNs and BNNs the operations are executed in a synchronized manner, while in SNNs the computation can be event-driven (computation only when an input spike arrives).

SNNs aim not only at realistically emulating biological computation of the brain, but also target better energy-efficiency than ANNs when implemented in hardware. ANN computations involve matrix multiplications which can be done in parallel on GPUs. On the other hand, SNN computations can be realized efficiently on dedicated neuromorphic hardware [10], [11]. Neuromorphic computing aims at mimicking the asynchronous information processing and co-localization of memory and computing units in the brain, as opposed to the traditional von Neumann paradigm. Large-scale neuromorphic accelerators [12]–[16] and application-specific accelerators [17], [18] have been demonstrated, either for on-chip training, or for inference only. While ANNs process the analog information in a one-shot fashion using matrix multiplications, information in SNN is coded in a binary signal distributed over time. The use of binary spikes is one of the motivations for SNNs as it allows replacing costly multiply-accumulate (MAC) operations in ANNs by simpler accumulate (AC) operations, which consume less energy and occupy less area [19]. Moreover, not all neurons spike during an inference phase. Thus, this sparsity of activations decreases the number of ACs operations to be performed, which is another motivation for the use of SNNs. In addition, the asynchronous behavior of SNNs can allow faster inference in some neuromorphic hardware such as the neuromorphic asynchronous processors proposed in [20]. Note that most of the energy consumption of neural networks in specialized architectures comes from memory accesses associated with arithmetic operations rather than from the arithmetic operations themselves [21]. However, spike sparsity and event-based

This work has been partially supported by MIAI @ Grenoble Alpes, (ANR-19-P3IA-0003).

processing can help reduce the memory accesses by reducing the number of associated operations.

Various methods have been proposed to train SNNs in supervised or unsupervised manners. The Spike-Timing-Dependent Plasticity (STDP) rule is a local learning rule demonstrated by neurobiologists and is inspired by chemical mechanisms with computation capability existing in nervous system [22]. This rule makes it possible to train SNNs in an unsupervised and biologically-plausible manner and is suitable for on-chip learning due to its locality. Although it has been shown that STDP can approximate backpropagation update rules [23], [24], it is not the most appropriate rule to train deep networks with high accuracy in the context of deep learning applications. On the other hand, backpropagation-based training methods are able to accurately train deep networks. In that case, deep SNNs can be trained either directly with backpropagation, or indirectly by converting a pre-trained ANN to a SNN formalism. In this survey, we focus on training deep SNNs, and therefore we will not discuss the applications of the STDP rule, for which we refer the reader to [11], [24]–[27].

Surveys on SNNs [11], [24]–[28] have been published the past few years, addressing spiking neuron models, learning rules (unsupervised and supervised), neuromorphic hardware and applications. Motivated by the success of many recently proposed backpropagation-based training methods for deep SNNs, this survey takes a different approach by focusing on the latter. A novel taxonomy of backpropagation-based learning algorithms for the direct training of SNNs is proposed and strategies to improve these algorithms in terms of accuracy, latency and sparsity are discussed. In addition, the impact of the input encoding and network architecture on the accuracy-latency trade-off is highlighted.

The remainder of the paper is organised as follows. In Section II, the SNN model is presented, addressing the choices of neurons, synapses and coding strategy. In Section III, the different strategies to train deep SNNs from conversion methods to backpropagation-based methods are reviewed. In Section IV, strategies to improve the training of deep SNNs are discussed. Targeting accurate and energy-efficient hardware inference, the accuracy-latency trade-off is studied in Section V. Finally, promising solutions and remaining challenges for SNNs are summarized in Section VI.

II. The Spiking Neural Networks model

A. Neurons and synapses

The basic ANNs and SNNs units in a neural networks are shown in Fig.1. In ANNs, the output of a neuron is a function defined as:

$$y_i = \varphi(\sum_j x_j w_{ij} + b_i) \tag{1}$$

where y_i is the output activation of neuron i, b_i is the bias of neuron i, x_j is the input activation from presynaptic neuron j, and w_{ij} is the synaptic weight between neurons i and j. φ is an activation function, such as the Rectified linear unit (ReLU). While in ANNs the information propagates synchronously on a layer-by-layer basis, the information processing in SNNs is asynchronous and in a depth-first manner. Indeed, neurons in a layer fire spikes without waiting for other neurons in the same layer to fire. Due to the temporal dynamics of the neurons, SNNs operate in the spatio-temporal domain, while standard ANNs operate only in the spatial domain.

The most popular neuron model for SNNs is the Leaky Integrate and Fire model (LIF) [7], [8]. More complex neuron models exist [29], [30], but have not yet demonstrated superior performance than the simple LIF model for deep learning applications. In the LIF model, the membrane potential $V_i(t)$ of the neuron i is described as :

$$\begin{cases} \lambda \frac{\mathrm{d}V_i}{\mathrm{d}t} = -V_i + \sum_j w_{ij} \sum_k \epsilon(t - t_{jk}) \\ V_i(t) = V_{reset}, \text{ if } V_i(t) \geq \theta_i \end{cases} \tag{2}$$

where λ is the membrane time constant, w_{ij} is the synaptic weight from neuron j to i, $\epsilon(.)$ is the synaptic kernel, t_{jk} is the k^{th} spike of the input neuron j, V_{reset} is the reset membrane potential and θ_i is the membrane potential threshold. It is also possible to add a bias in the SNN model, for example by setting $V_i(0) = b_i$. Similar to biological neurons, the neuron integrates the weighted input spikes into its membrane potential. When the latter reaches its threshold, the neuron fires an output spike and the membrane potential is reset. Depending on the chosen membrane time constant, the Integrate and Fire neuron can also be non-leaky (IF). The difference is that the membrane potential of the IF neuron is constant in time between two spikes, while in the LIF neuron the membrane potential decays over time. The synapse model is defined by the kernel function $\epsilon(.)$, corresponding to the response of the neuron membrane potential to the presynaptic spike. The synapse can be instantaneous (Dirac kernel function) or continuous (e.g. linear, exponential, or alpha kernel functions). Neurons and synapses are shown in Fig.1b. The inference phase of SNNs is usually discretized in timesteps, each timestep corresponding to one forward pass in the network. In this case, an iterative version of the LIF model is used, as shown in [31]. The number of timesteps is generally used to estimate the future latency of the SNN inference in hardware.

B. Coding strategy

Contrary to ANNs, SNNs use the temporal dimension to code the information. Indeed, while ANNs use static real-valued activations per neuron per inference, SNNs use one or several binary spikes for which the temporal information is used to code the information. Therefore, the coding strategy refers to how the temporal information in the spikes is taken into account. There are several coding strategies based on the spike rate, timing, rank, phase, etc. (for a recent review see [32]), but the majority of works in deep SNNs use either the spike rate or the spike timing. The rate coding strategy uses several spikes to represent one unit of information while in temporal coding, the information is carried by individual spike times. Note that, in order to be efficient in neuromorphic hardware, the coding strategy should use a minimum number of spikes, as the energy consumption is strongly correlated to the spiking activity. Moreover, the coding strategy affects the

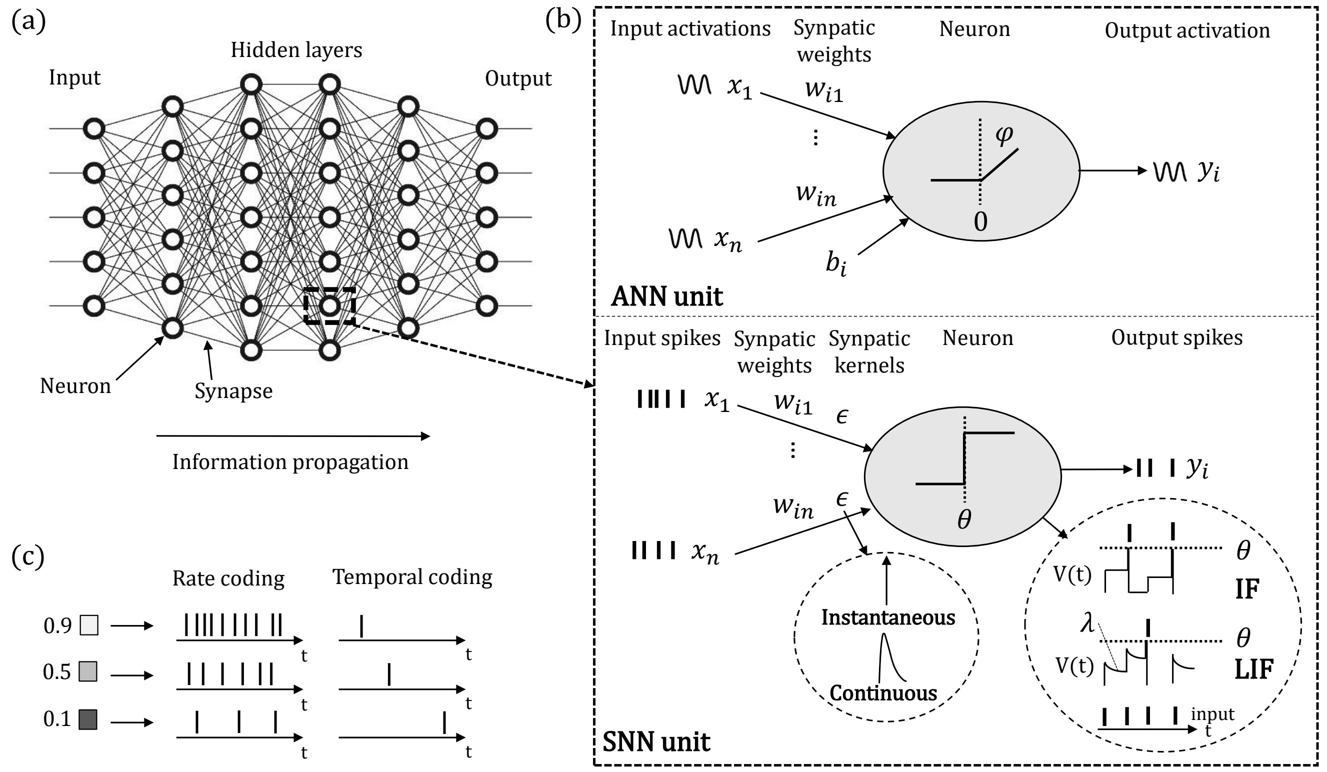

Fig. 1. (a) Feedforward fully-connected neural network. (b) ANN and SNN neuron and synapse models. (c) Input encoding: example of pixel-to-spike conversion with a rate coding or temporal (latency) coding.

behavior of the entire system, the characteristics of the neurons and synapses, and the efficiency of the learning process.

The coding strategy must consider both the encoding of the input to the network and the decoding of the output. Indeed, to process real-valued data, such as pixels for images, these values can be converted into spikes in order to be processed further by the SNN. Note that neuromorphic sensors can provide data already in the form of spikes which can be fed directly to the SNN without pre-processing. Fig.1c represents a typical pixel-to-spike conversion in rate and time. The rate-based strategy matches each pixel intensity with a firing rate, using a probabilistic sampling (generally Poisson) to generate the spike trains: the higher the pixel value, the higher the firing rate of the corresponding input. A simple time-based strategy, also called latency coding, consists in associating the pixel intensity with the latency of a single spike. In that case, the latency is inversely proportional to the pixel intensity: earlier spikes encode higher values and later spikes encode lower values. Decoding the output in a classification task consists in determining the most activated neuron in the output layer, each neuron being associated with a class. With rate coding, this is done by using the highest spike rate or membrane potential value. With temporal coding, a popular solution called Time-To-First-Spike (TTFS) uses the first spike fired by a neuron.

Temporal codes are sparse and can have a lower latency (e.g. when the TTFS decoding is used). However, temporal coding requires high temporal resolution because each spike carries important information, which may be difficult to implement efficiently in neuromorphic hardware [33]. Moreover, the analogy with ANNs is more straightforward with rate coding, and therefore the latter is easier to use for supervised training, either with conversion or direct training approaches.

III. Training deep SNNs

Learning is the process by which the weights (i.e. the values of the synapses) of the neural network are determined such that the network performs a specific task. Most deep learning applications use supervised learning, meaning that the expected outputs are known (each data is associated to a label). A cost metric is defined as a function of the error between the desired and actual output. The learning process consists of tuning the values of the synaptic weights such that the cost function is minimized. In ANNs, the backpropagation algorithm is used to calculate the gradients of the cost function with respect to each synaptic weight in order to perform synaptic weights updates. The calculation starts from the last layer of the network and proceeds backwards layer by layer. This survey focuses on supervised strategies inspired by the backpropagation algorithm for training deep SNNs. Applying backpropagation to SNNs is challenging due to the nature of the neuron activation function (step function) which is not differentiable, thus the error can not be backpropagated correctly. Different strategies have been proposed to mitigate this problem, such as approximating the derivative with a surrogate gradient [9] (see Fig. 3), or directly differentiating the spike times [34]. An alternative solution is to convert a trained ANN to a SNN formalism, also called indirect training or ANN-to-SNN conversion, which bypasses the training difficulty of SNNs.

A. ANN-to-SNN conversion

The ANN-to-SNN conversion is an indirect training strategy consisting in training an ANN and then mapping the trained weights to a SNN, assuming equivalence of the SNN computing units to the ANN ones. The ANN must be trained under constraints to fit the SNN model, such as removing biases of neurons (which are usually not represented in the SNN model) and batch normalization layers (which rely on biases). Then, either the thresholds of the spiking neurons or the weights are normalized equivalently, so that the transfer function (input-output mapping) of the SNN unit matches the transfer function of the ANN unit [35], [36].

1) Conversion with rate coding: Rate coding is a straightforward approach to conversion, in which the firing rate of neurons is considered equivalent to the analog output (activation value) of the ANN neuron [10], [37]. However, the conversion process results in errors in some cases, for instance when the ANN activation is too high and can not be accurately represented by the spike rate given a fixed simulation duration. An effective data-based weight normalization, consisting in rescaling the weights in each layer according to the maximum ANN activation in the corresponding layer within the training set, is presented in [35] to mitigate this problem. Another solution is proposed in [36], which balances the thresholds in each layer according to the maximum SNN activation. The difference with the method in [35] is that SNN statistics are used instead of ANN statistics to determine the normalization, which leads to more accurate results. They report high accuracy for a converted SNN with VGG-16 network architecture, such as 91.55% on CIFAR-10 and 69.96% on ImageNet using 2500 inference timesteps. The accuracy can be further improved using a soft reset mechanism instead of a hard reset [38]. The soft reset consists in subtracting the threshold value from the membrane potential after the neuron fires a spike, instead of setting it to the reset potential value. The residual membrane potential above the threshold is thus kept for the next spike, which reduces the loss in the spiking quantization process. Their method yields a near lossless conversion, showing 93.63% accuracy on CIFAR-10 and 73.09% on ImageNet with a VGG-16 architecture using 2048 and 4096 timesteps, respectively. Similar results are shown with ResNet-20 and ResNet-34 on CIFAR-10 and ImageNet achieving 91.36% and 69.89% accuracy, respectively.

2) Conversion with temporal coding: Another approach to conversion is based on temporal coding. This approach is attractive because the number of spikes emitted can be decreased drastically, thus further reducing the energy consumption. This is first proposed in [19] using the equivalence between the activation value of the ANN unit and the inverse of the spike time of the SNN unit. In [39] an accuracy as high as the one obtained with rate coding [38] is demonstrated on CIFAR-10 and Imagenet using VGG-16 and ResNet architectures, with at most two spikes per neuron. They propose a novel temporal coding with one positive and one negative spike per neuron. In addition, they introduce a threshold balancing method to improve the accuracy while using fewer timesteps compared to the rate-based conversion of [38]. Furthermore, the paper [33] introduces a temporal code associated with a new spiking neuron model using $logN$ different values of spike times to transmit integers between 1 and N in order to reduce the required temporal resolution. They achieve 83.57% accuracy on ImageNet with the EfficientNet-B7 architecture (75.10% with ResNet-50), with on average less than 2 spikes per neuron per inference. However, the proposed neuron model is complex, requiring additional parameters and additional state functions computed at each timestep, and might be costly to implement in neuromorphic hardware.

3) Limitations: The ANN-to-SNN conversion results in SNNs with accuracy close to the accuracy of the original ANN. Although originally designed for rate coding, the conversion process is compatible with many coding strategies. In fact, conversion with temporal coding is a promising way to increase the spike sparsity, but it might be less robust to noise in hardware implementation. However, both coding strategies require hundreds to thousands of inference timesteps, leading to a much higher latency and a degraded energy efficiency [28]. The long inference time required to achieve high accuracy is inherent to the equivalence chosen for the conversion. Indeed, the ReLU activation function approximates the firing rate of the IF model only if the SNN inference is discretized with a sufficient number of timesteps. However, recent improvements in conversion methods can mitigate this problem. For instance, the authors in [40] propose to replace the standard ReLU in the ANN model by the quantization clip-floor-shift activation function, which better approximates the firing rate of the IF model. Indeed, by clipping and flooring, they suppress the conversion errors due to large activations in the ANN and the discretization of the SNN activation (resp.), if the quantization step is properly chosen according to the time discretization of the SNN inference. Therefore, they reduce the conversion error between the ANN and SNN model and make it possible to obtain SNNs with a much smaller number of timesteps. For instance, on ImageNet with ResNet-34, they achieve 69.37% (resp. 72.35%) accuracy with 32 (resp. 64) timesteps. However, there is still a gap between the accuracy of the ANN and the converted SNN due to unevenness errors, which result from the fact that a different order of input spikes produces a different output [40]. Besides, the conversion process does not allow the optimization of the temporal dynamics of the SNN, contrary to direct training approaches [41].

B. Backpropagation-based learning algorithms for deep SNNs

The direct training of SNNs with backpropagation was partly motivated by the objective of reducing the SNN latency. In order to apply backpropagation to SNNs, different strategies are used to circumvent the problem of the non-differentiability of the spiking activation function, such as approximating the derivative with a surrogate gradient [9], or directly differentiating the spike times [34]. The surrogate gradient technique consists in approximating the neuron's activation function (which is usually a step function) by a differentiable function during the backward pass, to enable informative gradients to backpropagate through the layers (see Fig. 3). The approach was first introduced by [42], [43] with the straight-through estimator used to train quantized neural networks. Furthermore, the SNN inference being discretized in timesteps, there are several strategies to apply backpropagation to SNNs (see Fig.2). We classify these methods into three categories. (1) The spatial approach uses accumulated quantities which are retrieved at the end of the inference phase (such as the spike count or membrane potential value) to serve as an activation value for each neuron. Then, backpropagation can be applied on these quantities. This strategy does not take into account the SNN temporal dynamics (i.e. the precise timing and order of the spikes and temporal components of the SNN model). (2) The spatio-temporal approach considers each timestep of the inference phase by using backpropagation through time (BPTT), which was originally used to train RNNs. The BPTT consists in applying the backpropagation method on an unrolled SNN network, where all the timesteps are simultaneously considered (one neuron unrolled for T timesteps will be instantiated T times in this virtual network). This method thus considers both temporal and spatial dynamics of SNN, and therefore takes into account the temporal dependencies associated with LIF neurons and continuous synapses. (3) The single-spike approach uses only one spike per neuron (using a latency temporal coding), and computes directly the spike time of each neuron during the inference phase. Then, the backpropagation algorithm can be applied using the spike time of each neuron as its activation value.

1) Spatial approaches: Spatial approaches, such as [44]–[46], use a rate-coding strategy with IF neurons and instantaneous synapses and do not consider temporal dependencies in the gradient computation. They consist in approximating the SNN forward pass during the training in order to obtain a lighter backpropagation, only in the spatial domain, as in ANN training. The spatial approach resembles the ANN-to-SNN conversion with rate coding, however the training targets are different. In conversion methods, the ANN is trained under constraints and then, there is a conversion procedure to transfer the ANN trained weights to the SNN. On the other hand, in the spatial approach, the SNN is directly trained, but viewed as an ANN, and thus can be trained in a similar way using accumulated quantities during the SNN forward pass. Hence, contrary to the ANN-to-SNN conversion, there is only one network and no conversion procedure.

For instance, the authors in [44] consider the membrane potential without the spiking discontinuities, using low-pass filtered spike signals. Therefore the signal is continuous (considering the spiking activity as noise) and backpropagation can be applied on it. They use an architecture with lateral inhibitions and neurons with a refractory periods. To keep the performance stable, error normalization, threshold regularization and a novel exponential weight regularization are used to reach 99.31% accuracy on MNIST with a 4-layer CNN architecture. In [46] the authors take the spike count of neurons as a surrogate for gradient backpropagation. They define the equivalent of the neuron activation as the sum of its spikes

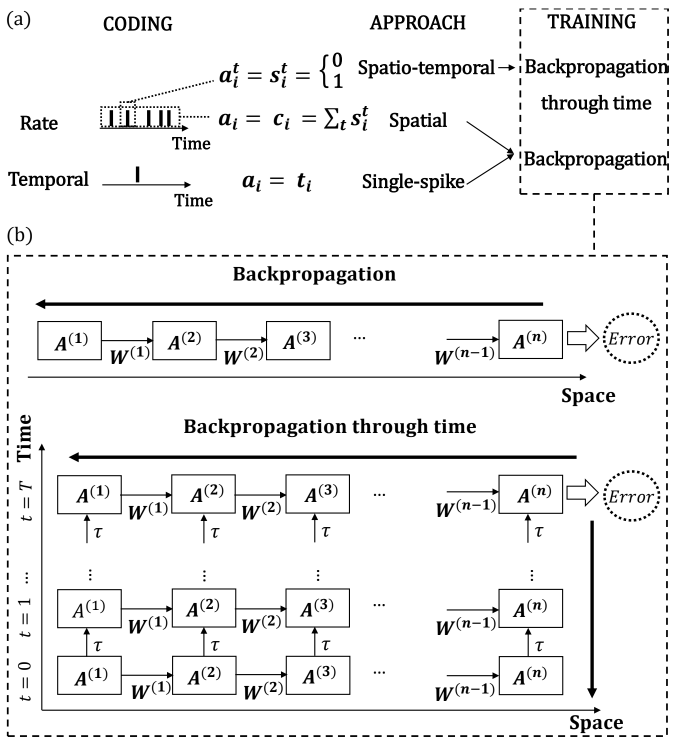

Fig. 2. (a) Backpropagation-based learning algorithms. Spatial and spatio-temporal approaches use a rate coding while the single-spike approach use a temporal (latency) coding. The spatio-temporal approach considers the activation of each neuron at each timestep a_i^t, corresponding to the emission or not of a spike s_i^t. The backpropagation through time is used to backpropagate the error in both space and time dimensions. On the other hand, the spatial and single-spike approaches consider for each neuron a single activation a_i for the forward pass, which can correspond to the spike count c_i for the former or the timing of the unique spike emitted by the neuron t_i for the latter. Therefore, the backpropagation is used to backpropagate the error only in the space dimension. (b) Backpropagation and backprogation through time training. Notations: n number of layers, T number of timesteps used in the SNN inference, $A^{(l)}$ activation of neurons in layer l, $W^{(l)}$ weight vector from layer l to $l+1$, τ membrane potential and postsynaptic potential update (for LIF neurons and continuous synapses).

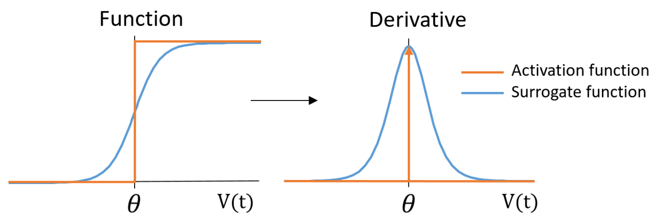

Fig. 3. The activation function of neurons has derivative equals to zero everywhere except in θ where it is infinite. Therefore, the derivative of a surrogate function is used to compute the gradients during the backward pass.

produced during the simulation time [46]:

$$a_i^n = \sum_{t=1}^{T} s_i^{t,n} \tag{3}$$

with a_i^n the equivalent activation of neuron i from layer n, and $s_i^{t,n}$ a spike produced by this neuron at timestep t. Therefore, the aggregated input current of a neuron can be expressed as [46]:

$$z_i^n = \theta \sum_j w_{ij}^{n-1} a_j^{n-1} + b_i^n \tag{4}$$

They use a cross-entropy loss, as it is commonly used for classification tasks:

$$E(a_i^N, y_i) = -log(\frac{exp(a_i^N)}{\sum_k exp(a_k^N)}) \tag{5}$$

with a_j^N the output spike count of neuron j in the last layer and y_j the target one-hot label for this neuron. Then, the error derivative with respect to weights and biases is computed using standard backpropagation. For instance, in the hidden layer, the error derivative with respect to the weights is computed using the chain rule [46]:

$$\frac{\partial E}{\partial w_{ij}^{n-1}} = \frac{\partial E}{\partial a_i^n}\frac{\partial a_i^n}{\partial z_i^n}\frac{\partial z_i^n}{\partial w_{ij}^{n-1}} \tag{6}$$

with $\frac{\partial a_i^n}{\partial z_i^n} = \frac{1}{\theta}.(z_i^n > 0)$. They achieve 99.26% accuracy on MNIST with a 3-layer CNN architecture. However, they argue that considering only the spike count generates a quantization error, as the surplus membrane potential of spiking neurons is not taken into account, which could become a problem for deeper neural networks. Furthermore, the paper [45] shows that the backpropagation phase can also be realized with spikes by considering the error in a discrete form. Therefore, the same hardware infrastructure can be used for both inference and learning, which makes it attractive for on-chip learning. Moreover, they show that the spike discretization error can be reduced to zero by adding some constraints on the ANN. Therefore, they can perform the offline training directly with the equivalent ANN. They demonstrate 89.99% accuracy on CIFAR-10 with a 8-layers VGG architecture. However, because they perform training and inference with the equivalent ANN, they do not indicate the number of inference timesteps which would be used for the inference with the SNN.

Overall, the spatial approach is purely rate-based and therefore is the closest to ANNs. This approach was not used in many works as it appears to not benefit from the spatio-temporal dynamics of SNNs. However, in [46] the authors argue that spatial rules are less complex than spatio-temporal rules and can be more efficient in terms of computation and memory due to the use of backpropagation instead of BPTT. Moreover, the most recent ANN-to-SNN conversion techniques resemble the spatial SNN training. Indeed, they convert a SNN from an ANN that matches more closely the SNN such that the differences between the two models decrease and the conversion becomes more straightforward [40].

2) Spatio-temporal approaches: Spatio-temporal approaches, such as those described in [31], [47]–[58], also use a rate-coding strategy, but propagate the gradient both in spatial and temporal dimensions using the BPTT. They usually have temporal components, such as in the neurons (LIF), synapses (continuous), or in the loss function. The majority of works use rate-coded loss functions (based on the output firing rate or output membrane potential) and LIF neurons with instantaneous synapses.

Paper [31] proposes a spatio-temporal backpropagation for SNN based on an iterative LIF model and approximated the non-differentiable spiking activity with a surrogate gradient. Their iterative LIF obtained by solving the differential equation of the LIF is defined as [31]:

$$\begin{cases} u_i^{t+1,n} = u_i^{t,n} f(o_i^{t,n}) + x_i^{t+1,n} + b_i^n \\ x_i^{t+1,n} = \sum_{j=1}^{l(n-1)} w_{ij}^n o_j^{t+1,n-1} \\ o_i^{t+1,n} = g(u_i^{t+1,n}) \\ f(x) = \tau exp(-x/\tau) \\ g(x) = 1 \text{ if } x \geq \theta \text{ , 0 otherwise.} \end{cases} \tag{7}$$

with $u_i^{t,n}$, $o_i^{t,n}$, $x_i^{t,n}$, respectively the membrane potential, output, and input current of neuron i in layer n at time t. They use a loss based on the mean square error for all samples of the batch (of size S) between the target vector (y) and the sum of the output vector of neurons in the last layer during the simulation time T [31]:

$$L = \frac{1}{2S}\sum_{s=1}^{S} ||y_s - \frac{1}{T}\sum_{t=1}^{T} o_s^{t,N}||_2^2 \tag{8}$$

The derivative of the loss with respects to the membrane potential and output spikes is needed to compute the error derivative with respect to weights and biases. These values are obtained by unrolling the network in time and space. For instance, in the hidden layers at a given timestep, the loss derivative with respect to output spikes depends on the output spikes of the next layer (space) and the output spikes at the next timestep (time). Similarly, the loss derivative with respect to the membrane potential depends on the output spikes produced by this neuron (space) and the output spikes at the next timestep (time) [31]:

$$\begin{aligned} \frac{\partial L}{\partial o_i^{t,n}} &= \sum_{j=1}^{l(n+1)} \frac{\partial L}{\partial o_j^{t,n+1}}\frac{\partial o_j^{t,n+1}}{\partial o_i^{t,n}} + \frac{\partial L}{\partial o_i^{t+1,n}}\frac{\partial o_i^{t+1,n}}{\partial o_i^{t,n}} \\ &= \sum_{j=1}^{l(n+1)} \frac{\partial L}{\partial o_j^{t,n+1}}\frac{\partial g}{\partial u_i^{t,n+1}} w_{ij} + \frac{\partial L}{\partial o_i^{t+1,n}}\frac{\partial g}{\partial u_i^{t+1,n}} u_i^{t,n}\frac{\partial f}{\partial o_i^{t,n}} \end{aligned} \tag{9}$$

$$\begin{aligned} \frac{\partial L}{\partial u_i^{t,n}} &= \frac{\partial L}{\partial o_i^{t,n}}\frac{\partial o_i^{t,n}}{\partial u_i^{t,n}} + \frac{\partial L}{\partial o_i^{t+1,n}}\frac{\partial o_i^{t+1,n}}{\partial u_i^{t,n}} \\ &= \frac{\partial L}{\partial o_i^{t,n}}\frac{\partial g}{\partial u_i^{t,n}} + \frac{\partial L}{\partial o_i^{t+1,n}}\frac{\partial g}{\partial u_i^{t+1,n}} f(o_i^{t,n}) \end{aligned} \tag{10}$$

All the terms can be derived using the equations (7)-(8), except the derivative of the spiking activation function ($\frac{\partial g}{\partial u}$), which must be approximated by a surrogate gradient. They demonstrate 99.42% accuracy on MNIST with a 4-layer CNN architecture. The authors in [53] follow the approach of [44] by considering the membrane potential without the spiking discontinuities, but with a leak in the neuron model. They achieve 90.95% accuracy on CIFAR-10 with a ResNet-11 using 100 timesteps for inference with on average 1.53x10^6 spikes per inference, which leads to x15 efficiency gain with respect to MAC and AC operations (using the energy values given in [21]) compared to the ANN version. In [58], they do not use temporal components in the neuron and synapse models, but use a temporal loss inspired by the van Rossum distance. Therefore, they minimize the distance between the desired and actual output spike train.

The use of surrogate gradient to approximate the non-differentiable behavior has the effect of smoothing the spiking activity. This leads to an inconsistency between the computed gradient and loss, which degrades the accuracy [49], [54]. Therefore, the authors in [49] introduce a backpropagation at the spike train level with a decoupled macro factor accounting for the rate-coded loss function and a micro factor to incorporate temporal dependencies at the spike train level. They show that this decomposition improves the precision of the temporal learning. In the same spirit, in [54], the derivative of the error is decomposed in two factors, one accounting for the inter-neuron dependencies and one accounting for the intra-neuron dependencies. They use LIF neurons with continuous exponential synapses and consider incremental changes in the postsynaptic potential with regards to the membrane potential of a neuron to compute the gradients. Moreover, they use a temporal loss function analogous to the van Rossum distance. They proposed a warm-up mechanism with a sigmoid instead of a step function as activation function to enable backpropagation when there is no spike. They improved the temporal learning compared to [49] and scaled to CIFAR-10, while reducing the inference latency and showing high spike sparsity with on average 0.49 spike per neuron per inference.

In summary, the spatio-temporal approaches consider spatial and temporal dependencies using the BPTT. This is relevant to take into account the temporal components of the model (such as in the neurons, synapses and loss function) in the backpropagation. Therefore, they can benefit from the full spatio-temporal dimension of the SNN dynamics.

3) Single-spike approaches: Single-spike approaches, such as [34], [59]–[64], circumvent the non-differentiability problem when applying backpropagation in SNNs by directly differentiating the spike times. These methods have been applied to image processing, the spatial information contained in the images being directly converted to the spike timing using a temporal (latency) coding.

The single-spike approach was introduced very early with the SpikeProp learning rule [65]. SpikeProp defines the firing time of neurons as a function of their membrane potential and thus approximates their derivative using the changes of the membrane potential around the firing time. Later, the authors in [34] demonstrated that by using single-spike IF neurons with exponential synapses, the differential equation of the neuron membrane potential has a simple solution. Indeed, they define the IF neurons and exponential synapses as [34]:

$$\frac{dV^j_{mem}(t)}{dt} = \sum_i w_{ji} \sum_r \epsilon(t - t^r_i) \tag{11}$$

with t^r_i the time of the rth spike from neuron i and ϵ the synaptic current kernel given by [34]:

$$\epsilon(x) = exp(-\frac{x}{\tau_{syn}}) \text{ if } x \geq 0 \text{ , 0 otherwise.} \tag{12}$$

with τ_{syn} the synaptic time constant. Then, assuming a neuron will spike at time t_{out}, they integrate the membrane potential for $t < t_{out}$, which allows to obtain the output spike time of a neuron (t_{out}) as a function of the spike times of its presynaptic neurons [34]:

$$exp(t_{out}) = \frac{\sum_{i \in C} w_i exp(t_i)}{\sum_{i \in C} w_i - \theta} \tag{13}$$

$C = \{i : t_i < t_{out}\}$ is the causal set of the neuron, containing the input spikes that had arrived before its output spike and thus have influenced the output spike. Due to this analytic input-output relation, the spike times can be computed directly without simulating the whole network dynamics (i.e. computing membrane potentials at each timestep). Therefore, there is no need to do a BPTT, but a direct backpropagation only on the spike times is possible. A particularity of this algorithm is that, due to the form of the analytical relation, for a given neuron, the backpropagation rule only applies to the presynaptic neurons in its causal set. Furthermore, backpropagation is not performed on the neurons that do not spike. Therefore, the authors in [34] apply weight regularization to push neurons to spike, by forcing its presynaptic sum of weights to be superior to its threshold. Using a change of variable $exp(t_x) \longrightarrow z_x$, the derivative of the output spike time with respect to the weight and with respect to an input spike time is computed [34]:

$$\frac{dz_{out}}{dw_p} = \frac{z_p - z_{out}}{\sum_{i \in C} w_i - \theta} \text{ if } p \in C, \text{ 0 otherwise.} \tag{14}$$

$$\frac{dz_{out}}{dz_p} = \frac{w_p}{\sum_{i \in C} w_i - \theta} \text{ if } p \in C, \text{ 0 otherwise.} \tag{15}$$

The loss function for the output spike times vector of the output neurons $\boldsymbol{z}^L$ and a target class index g is defined as [34]:

$$L(g, z^L) = -ln \frac{exp(-z^L[g])}{\sum_i exp(-z^L[i])} \tag{16}$$

The loss function takes the negative of the spike times (in the z-domain) to encourage the neurons corresponding to the correct class to fire earlier. They achieve 97.2% accuracy on MNIST with a 784-800-10 fully-connected network.

Papers [59] and [60] take their inspiration from [34] but use IF neurons and synapses with alpha synaptic kernel, and LIF neurons and synapses with dual exponential kernel, respectively. Their models are more biologically plausible, but the differential equations have complex solutions. Performance in hardware is demonstrated in [60] by implementing the algorithm in BrainScaleS-2. They yield 95.9% accuracy with $25 \mu J$ per classification on MNIST (16x16 images) with a 256-128-10 fully-connected network.

Another work [61] uses IF neurons with linear synapses, taking inspiration from the ReLU units used in ANNs, demonstrating 99.2% accuracy on MNIST with a 5-layer CNN architecture. They argue that the linear synapse alleviates the problem of exploding gradient and dead neurons compared to the alpha synapse (and more generally to a decaying kernel function). Indeed, the derivative of the linear synapse is never close to zero, and there is no leak in the model therefore neurons are more likely to spike. The authors in [62] also use linear synapses and added a temporal penalty term to stabilize the timing of the output spike instead of the penalty on the weights used by [34]. This allows to control the output spike times of the network and therefore to tune the inference latency. Furthermore, considering analog VLSI implementations, they show that their SNN dynamics can be mapped to circuits with analog resistive memory. Using numerical simulations, the robustness of their model to device variations (when taken into account during the training) was demonstrated for a small fully connected network on the MNIST task. Besides, the authors in [63] simplify the model of [34] by using instantaneous synapses with IF neurons. However, because the synapse is not continuous, they must approximate the derivative of the spike time with regards to the membrane potential, and so the gradients are not exact. They obtain 97.4% accuracy on MNIST with a 784-400-10 fully-connected network.

Finally, the paper [64] extends the work of [34] by proposing an efficient way of computing the spike times taking advantage of parallel tensor computations in deep learning frameworks running on GPUs, thus speeding-up the offline training. They achieved 92.68% and 68.8% accuracy on CIFAR-10 and ImageNet using VGG-16 and GoogleNet architectures, respectively. Targeting efficient inference, they show that their method is robust to weight quantization. Indeed, they yield 90.93% accuracy for CIFAR-10 with 2-bits quantization, and 65.2% (resp. 60.0%) for ImageNet with 4-bit (resp. 2-bit) quantization. In addition, they report high sparsity of activations, with on average 0.62 (resp 0.56) spike per neuron per inference on CIFAR-10 (resp. ImageNet). However, as they directly compute the spike times without simulating the SNN dynamics, they do not discretize the inference phase with timesteps. Therefore, the number of timesteps which would be used to simulate the SNN inference with the desired temporal resolution in hardware is unknown.

In summary, single-spike approaches have the advantage of using at most one spike per neuron and thus appear promising for energy-efficient hardware implementations. Moreover, using the analytical input-output relation allows us to compute exact gradients instead of using approximate gradients, which is better for the optimization process. However, the backpropagation is only performed on presynaptic neurons which have spiked before the given neuron. Therefore, fewer synaptic weights are updated per iteration, which could lead to a slower training convergence. Moreover, encouraging neurons to fire during training improves the learning [34], [64] but can limit the sparsity reached. In addition, while the single-spike approach is adapted to static input data, it is hardly compatible with dynamically changing input data, as neurons can fire only once [24].

4) Limitations: Considering the spatial and single-spike approaches, only the works [45], [64] have shown scalability to CIFAR-10, and ImageNet for the latter. The spatio-temporal approach is the only approach considering spatial and temporal dimension in the backpropagation, which can lead to a better optimization of the SNN in terms of accuracy and latency. Moreover, spatio-temporal approaches are compatible with dynamically changing input data, such as neuromorphic data. However, the computational and memory cost of training with BPTT is huge, as this method requires storing the activations and computing the gradient at all timesteps. This is a major obstacle to the direct training of SNNs. For instance, training VGG-16 on CIFAR-10 for one epoch of BPTT (using 100 timesteps) with the method of [66] takes 78 min and 9.36 GB of GPU memory (using Nvidia GeForce RTX 2080 Ti TU102 GPU with 11GB memory). For comparison, the VGG-16 ANN training of one epoch requires only 0.57 min and 1.47 GB, which is x137 less time and x6 less memory.

In addition, SNN training is still challenging. Most of the works use an approximate derivative for the spiking activity, therefore there is an inconsistency between the model which is optimized with the backpropagation and the actual SNN model. These approximation errors could accumulate through the layers. Moreover, [58] argued that, for SNNs trained with BPTT, reducing the number of timesteps is crucial, not only to reduce the latency and energy consumption, but also to improve the training convergence. Indeed, with the same network architecture, the unrolled SNN in the BPTT is much bigger compared to the ANN trained with backpropagation. Therefore, similar to RNNs, the vanishing and exploding gradients problem is increased. These problems prevent SNNs from scaling to very deep architectures. Hence, in addition to an increased offline training cost, the accuracy of SNNs trained with backpropagation is still limited and lags behind conversion approaches.

IV. Optimizing further deep SNNs

Recent works have proposed additional strategies to improve the supervised training of SNNs with backpropagation, such as transferring ANNs techniques to SNNs, improving the encoding and decoding, using wider network architectures, hybridization between training strategies, and tuning the parameters specific to the SNN model. This methods have demonstrated significant improvements regarding the accuracy, latency and spike sparsity. The main results of backpropagation-based methods (spatial, spatio-temporal and single-spike methods) on static and neuromorphic vision datasets are summarized in Tables I and II, while showing the effect of using the improvements described in the following subsections.

A. Adapting successful ANN techniques to SNNs

ANNs training is more mature and already benefits from successful techniques developed in the past few years. Therefore, taking inspiration from these techniques could be a solution to improve SNNs accuracy. For instance, regularization is a powerful ANN technique to decrease the overfitting occurring during the training process and thus improving the generalization of the network. One regularization method especially designed for SNN is proposed in [51]. This neuron normalization method (NeuNorm) consists in using auxiliary neurons at each convolutional layer. These auxialiary neurons compute a weighted summation of spike counts of the other neurons to be fed to the next layer in addition to the raw spike signals. This additional signal is used to balance the

TABLE I
COMPARISON OF BACKPROPAGATION-BASED ALGORITHMS ON STATIC VISION DATASETS

Learning strategy	Paper	Coding	Neurons + synapses	Architecture	Regularization method	Additional training strategy	Timesteps	Acc. (%)
CIFAR-10								
Spatial	[45]	rate	IF + instantaneous	VGG-8	/	/	/	89.99
Spatio-temporal	[51]	rate	LIF + instantaneous	VGG-8	neuron normalization, dropout	encoding layer, voting layer	12	90.53
	[53]	rate	LIF + instantaneous	ResNet-11	dropout	/	100	90.95
	[54]	rate	LIF + exponential	VGG-8	/	encoding layer	5	91.41
	[58]	rate	IF + instantaneous	ResNet-11	batch normalization, dropout	surrogate gradient tuning	20	90.20
	[56]	rate	LIF + instantaneous	VGG-8	batch normalization, dropout	encoding layer, voting layer	8	93.50
	[67]	rate	LIF + instantaneous	VGG-9	batch normalization	/	25	90.50
	[57]	rate	LIF + instantaneous	ResNet-19	batch normalization	encoding layer, voting layer	6	93.16
Single-spike	[64]	time	IF + exponential	VGG-16	weight regularization	/	/	92.68
CIFAR-100								
Spatio-temporal	[58]	rate	IF + instantaneous	ResNet-50	batch normalization, dropout	surrogate gradient tuning	40	58.5
	[67]	rate	LIF + instantaneous	VGG-11	batch normalization	/	30	65.8
ImageNet								
Spatio-temporal	[57]	rate	LIF + instantaneous	ResNet-34	batch normalization	encoding layer, voting layer	6	67.05
	[68]	rate	LIF + instantaneous	ResNet-152	batch normalization	encoding layer	4	69.26
Single-spike	[64]	time	IF + exponential	GoogLeNet	weight regularization	/	/	68.8

TABLE II
COMPARISON OF BACKPROPAGATION-BASED ALGORITHMS ON NEUROMORPHIC VISION DATASETS

Learning strategy	Paper	Coding	Neurons + synapses	Architecture	Regularization method	Additional training strategy	Timesteps	Acc. (%)
CIFAR-10-DVS								
Spatio-temporal	[51]	rate	LIF + instantaneous	VGG-5	neuron normalization, dropout	encoding layer, voting layer	20	60.5
	[56]	rate	LIF + instantaneous	VGG-6	batch normalization, dropout	encoding layer, voting layer	20	74.8
	[67]	rate	LIF + instantaneous	VGG-7	batch normalization	/	20	63.2
	[57]	rate	LIF + instantaneous	ResNet-19	batch normalization	encoding layer, voting layer	10	67.8
DVSGesture								
Spatio-temporal	[55]	rate	LIF + continuous	5-layer CNN	/	synapse kernel optimization	/	96.09
	[56]	rate	LIF + instantaneous	VGG-7	batch normalization, dropout	encoding layer, voting layer	20	97.57
	[57]	rate	LIF + instantaneous	ResNet-17	batch normalization	encoding layer, voting layer	40	96.87

input current received by neurons and leads to an increased accuracy. However, this normalization step requires additional multiplications as both the spike counts and the associated weights are real-valued quantities.

In the ANN domain, dropout [69] is a simple yet effective regularization technique transferable to SNNs [51], [53], [56], [58], [66], [70]. It consists in disconnecting some units of a layer with a given probability during the training to avoid the network relying too much on certain connections. For example, [53] introduced dropout for SNN trained with BPTT by keeping the same random subset of dropped units at each timestep. Batch Normalization (BN) [71] is a another powerful regularization technique widely used to train deep ANNs. It consists in rescaling the activations of a layer, and learning this scaling per batch, in order to maintain the variance of the activations throughout the network, which leads to better convergence. Several works propose to transfer the BN technique to the SNN training [57], [58], [67]. The authors in [67] show that standard BN should not be applied directly to SNNs, because it considers the timesteps all at once. Instead, they propose a BN ”through time” to decouple the parameters of the BN across the timesteps. They show that their BPTT method could converge on CIFAR-100 and TinyImageNet datasets, which was not the case without the BN. Furthermore, they report a decreased number of spikes per inference by one order of magnitude compared to the BPTT without BN and an increased robustness to noisy inputs. They showed x9 efficiency gain compared to the ANN version in terms of AC/MAC operations using the energy values of [21]. In addition, [57] propose a threshold-dependent spatio-temporal BN that normalizes the variance of the inputs to the threshold. By adding this BN to the BPTT algorithm described in [31] (further improved by the neuron normalization in [51]), they demonstrate scalability to deep residual networks with high accuracy while using fewer timesteps. In addition, they report sparse spiking activity (less than 2 spikes per neuron per inference on average). Therefore, it seems that BN enables the convergence of deeper networks while potentially reducing the spiking activity.

Scalability of SNNs can also be improved using optimized network architectures. For instance, the ResNet architecture can alleviate the gradient vanishing problem by adding residual shortcut connections, which enables the effective training of deeper networks [72]. Papers [53], [57], [58], [68] achieve high accuracy using ResNet architectures and regularization techniques. For instance, [58] and [57] show scalibility up to a 50-layer ResNet on CIFAR-100 and ImageNet, respectively. However, the accuracy gap with respect to ANN for the same architecture is still high. On ImageNet, the SNN ResNet-50 in [57] yields 64.88% accuracy while the ANN ResNet-50 achieves 76.13% [73], both being trained with BN. However, [68] proposes a novel implementation of the identity mapping in spiking ResNet, which can mitigate the problem of SNNs scaling with depth. Indeed, they are able to scale to very deep residual networks while increasing the accuracy with depth. For instance, they reach 69.26% accuracy on ImageNet with ResNet-152 using only 4 timesteps.

B. Improving encoding and decoding

Input encoding and output decoding of the network seriously impact the SNN accuracy and latency [10], [51], [74], [75]. Therefore, recent works proposed alternative coding methods.

1) Encoding: It is known that the higher the number of timesteps used in the inference phase, the higher the precision of the encoding and in turn, the higher the network accuracy. However, this induces an increased inference latency. Therefore, reducing the number of timesteps while preserving the accuracy is challenging. The authors in [74] point out that Poisson spike generation (the most common encoding scheme used in both spatial and spatio-temporal approaches) is inefficient, because it relies only on spike rate to encode information and does not make use of the temporal dimension. They propose a Discrete Cosine Transform which encodes distinct information at each timesteps, by decomposing the input signal into a basis of spectral components. Therefore, the latency is defined as the number of vectors in the basis, starting from coarse to fine grain resolution to maximize the accuracy while reducing the latency. However, this encoding requires two matrix multiplications in the pre-processing step.

Another solution to improve the encoding, which can be applied to real-valued signals from non-spiking datasets (such as image pixels), consists in using an encoding layer, also called direct input encoding, to do the conversion from real values to spikes [10], [51], [54], [56], [57], [70], [75], [76] (see Fig.4b). This method consists in directly feeding the real values to the first layer at each timestep, without discretization of the input, the discretization process with the spikes being done in the first layer. Such encoding layer is thus a hybrid ANN-SNN layer, as the synapses perform real-valued input-weight multiplications but the neurons are spiking units. The authors in [75] report that by using the encoding layer, a small VGG-5 trained with BPTT on CIFAR-10 could achieve 74.23% accuracy instead of 63.19% with a standard probabilistic sampling. Moreover, it seems that, as it improves the accuracy, the number of timesteps can be reduced without losing in accuracy. Indeed, it appears that since the inputs are encoded with full precision in the first layer, a lower precision can be used in the subsequent layers while maintaining a high accuracy. They show that the VGG-5 with the encoding layer can achieve already the same accuracy than the one obtained without the encoding layer by using only 3 timesteps instead of 15 [75]. Besides, other spatio-temporal approaches [51], [54], [56], [57] demonstrate accuracy above 90% on CIFAR-10 with 5-12 timesteps, while usually the same approach with Poisson spike generation requires about a hundred timesteps for accurate inference [53]. Similar conclusions are derived in the recent study [77] comparing the rate coding and the encoding layer strategies. Note that this hybrid layer must be supported in the neuromorphic hardware used for inference, or computed at the interface between the data and neuromorphic hardware.

2) Decoding: A simple way to increase the precision of the output layer is to apply the loss function on the high precision membrane potential of the output neurons instead of their spike rate [53], [66], [70]. Alternatively, population decoding can be used to improve the robustness of the classification when using the output spike rates [51], [56], [57]. In this decoding scheme, each group of output neurons represents a class, and the choice is made based on a voting strategy. The drawback is to use more neurons in the output layer and therefore more parameters. Besides, improving the softmax layer can also improve the accuracy by a few percents. For instance, [78] show about 2 to 3% accuracy improvement and a sparser spiking activity, by using a stochastic softmax instead of the standard softmax classifier. The stochastic aspect of this softmax can be considered as a regularization technique.

C. Wide network architectures

Using wider network architectures, by increasing the number of neurons per layer, improves the accuracy when the number of timesteps is constrained. For instance, the works [53], [57], [58], [66], [70] achieving low latency with high accuracy on CIFAR-10 use very large ResNet architectures (11 to 18M parameters), following the architecture designed in the ResNet paper [72] for the ImageNet task instead of CIFAR-10. Indeed, ResNet architectures for CIFAR-10 are originally very small architectures, such as the ResNet-20 with only 0.27M parameters, and yet are able to achieve good accuracy on CIFAR-10 (91.25%). On ImageNet, [57] report that by doubling the number of filters per convolutional layers in the ResNet-34 architecture, they increase the accuracy from 63.72% to 67.05%. Interestingly, when they compare the SNN with the ResNet-50 and ResNet-34 original architectures, the ResNet-50 yields better accuracy (64.88%), but not compared to the large ResNet-34. This shows that, in SNNs, from a certain depth, increasing the width of the network is more beneficial compared to further increasing the depth. Indeed, the number of neurons is increased, which can be considered equivalent to increasing the precision of a smaller number of neurons, without degrading the backpropagation phase. On the other hand, increasing the depth of the network makes the backpropagation more difficult due to accumulation of errors (for instance due to the surrogate gradient training) through the layers. This is on par with the work [79] proposing to use wider architectures to improve the accuracy of reduced-precision ANNs (with quantized weights and activations). Indeed, SNNs with few timesteps behave similarly to ANNs with strongly-quantized activations.

D. Training hybridization

Hybrid training approaches have also been proposed to further reduce the offline training and hardware inference costs

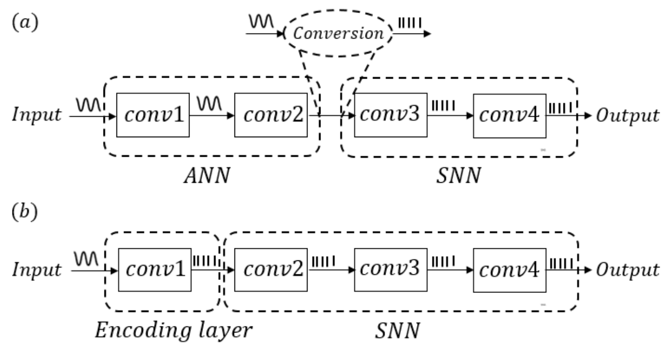

Fig. 4. (a) Example of ANN-SNN hybridization. Here, the first convolutions are done in ANN mode (MACs with real-valued inputs) and the last convolutions are performed in SNN mode (ACs with binary spikes). A conversion from analog values to spikes is performed between ANN and SNN layers. (b) Encoding layer. The first layer is hybrid ANN-SNN, as synapses perform MAC operations between weights and real-valued inputs, but neurons are spiking. Therefore, the conversion from analog values to spikes is performed directly by this layer.

while preserving the accuracy.

1) ANN-SNN network hybridization: Mixing ANN and SNN layers is one strategy to improve the accuracy (see Fig. 4a). For instance, the authors in [78] use a network with ANN layers at the inputs to improve the encoding accuracy, and SNN layers at the output, the whole network being trained with backpropagation (spatio-temporal for SNN layers). This approach demonstrates benefits for the CIFAR-10 classification task, for which the hybrid version yields 84.98% accuracy (+1.65% compared to the full ANN) using 25 timesteps with a VGG-9, showing x4 efficiency in terms of MAC/AC operations (using the energy values in [80]) compared to the ANN. However, the benefits are smaller on the Imagenet dataset (x1.3 efficiency), as they could not use more than two spiking layers in the VGG-13 architecture to achieve satisfactory accuracy (-2% compared to the full the ANN).

2) Tandem learning: Another strategy proposed in [76] is to couple each SNN layer with an ANN layer with weight sharing. In the training phase, the inference is performed by the SNN and the obtained spike counts are used as activation values by the ANN to perform the backpropagation. Therefore, the offline training phase is accelerated and needs less memory because the backpropagation is done on the ANN (thus removing the need for BPTT). The obtained SNN yields 90.98% accuracy on CIFAR-10 with 8 timesteps (7-layer VGG architecture), and 50.22% on ImageNet with 10 timesteps (AlexNet architecture). In comparison, the ANN versions achieve 91.77% and 57.55%, respectively. In addition, they report a sparse spiking activity (less than 0.4 spike per neuron per inference on the CIFAR-10 task), and thus up to x20 energy-efficiency compared to the ANN in terms of MAC/AC operations (using the energy values of [80]). Note that the temporal information can not be used in the backpropagation as the training is performed in the ANN domain.

3) Conversion and direct training hybridization: A hybrid approach between conversion and supervised direct training, thus mixing direct and indirect training, is proposed in [66]. Indeed, the ANN-to-SNN conversion yields very good accuracy but at the cost of a high number of inference timesteps, while SNN supervised training yields a lower number of timesteps but the spatio-temporal training with BPTT is expensive. Taking the best of both worlds, they used ANN-to-SNN conversion as a pretraining and further applied SNN fine-tuning with a spatio-temporal learning rule. For instance with a VGG-16 architecture on CIFAR-10, after the ANN pretraining (250 training epochs), the SNN converged with 20 training epochs, showing the effectiveness of the pretraining. Therefore, the total training duration is reduced to 28 hours (using Nvidia GeForce RTX 2080 Ti TU102 GPU with 11GB memory) compared to 325 hours with SNN training from scratch. However, the memory requirements for training are not reduced, as the SNN still requires training with BPTT. In addition, the latency is decreased while yielding high accuracy. For instance, on ImageNet with 250 timesteps, a VGG-16 (resp. ResNet-34) with hybrid training achieves 65.19% (resp. 61.48%) accuracy, while with the same number of timesteps the converted SNN without the fine tuning only achieves 62.73% (resp. 56.87%).

E. Leveraging the specificity of the SNN model

The methods to improve the supervised training proposed in the previous sections are mostly inspired by the ANN training. However, the SNN model have some specificity that do not exist in the ANN version, such as the threshold parameter, or time constants for LIF neurons and continuous synapses. This section shows that, by making efficient use of the rich dynamics of the SNN model, further benefits can be expected in terms of accuracy and efficiency (latency and sparsity).

1) Neuron's leak and threshold: Typical parameters of spiking neurons such as leak (for LIF neurons) and threshold are usually defined as hyperparameters and not considered in the training. However, neuron's leak and threshold are important parameters determining the SNN behavior. For a given set of weights, the threshold determines how much the input neurons must spike in order for the neuron to spike. The leak parameter controls how close to each other input spikes must be for a temporal coincidence to be detected. Thus, tuning leak and threshold parameters with backpropagation can allow a better optimization of the SNN model. For instance, the authors in [56] propose to learn the leak parameters and reported higher accuracy than other spiking approaches on neuromorphic datasets CIFAR-10-DVS and DVSGesture (see Table II). In [70], both leak and threshold are learned with the hybrid conversion pretraining and supervised fine-tuning method of [66]. They show an accuracy improvement compared to learning only the synaptic weights of about 1% on CIFAR-10 and up to 5% on CIFAR-100 and ImageNet, as well as a lower spike rate (average number of spikes per neuron per inference). The effect of tuning the thresholds when the weights are learned is surprising, as adjusting either the weights or the thresholds (from pre-trained weights) is usually considered equivalent in conversion methods. However, they demonstrate that, with iso-accuracy (using VGG-16 on CIFAR-10), tuning the thresholds leads to a reduction in timesteps from 25 to 15 and in spike rate from 1.94 to 1.47. Tuning the leaks reduces even more the number of timesteps from 15 to 5 and the spike sparsity from 1.47 to 0.39. The threshold and leak parameters are shared between neurons in a layer, thus the number of added parameters is negligible compared to the total number of parameters in the model.

2) Synapse dynamics: The choice of the synaptic kernel function, when continuous synapses (rather than instantaneous) are used, is another parameter to explore in order to optimize the SNN model. For instance, in [55], the authors propose a SNN with LIF neurons where the synapses are described as second order infinite impulse response filters, allowing to model various types of kernel (e.g. instantaneous, exponential, alpha, dual-exponential). The coefficients of the synapse filter are jointly learned with the synaptic weights using BPTT. They achieved 96.09% accuracy on DVSGesture.

3) Surrogate gradient: Finally, the spatial and spatio-temporal approaches use a surrogate gradient to approximate the derivative of the spiking activity. The derivative of sigmoid functions is often used as a surrogate, as the sigmoid function can be seen as a smooth approximation of the step function (see Fig. 3), but other surrogate derivatives, such as exponential or piece-wise linear functions, can also be used [9]. While the training performance is robust to the shape of the surrogate function, it is strongly affected by its scale [81]. For instance, paper [58] shows that, by tuning the scale of the surrogate gradient function, the variance of the gradients can be preserved through the layers, avoiding exploding and vanishing gradients. Therefore, by optimizing the width of the surrogate gradient function (and additionally using BN), they can scale to a ResNet-50 architecture, achieving 58.5% accuracy on CIFAR-100 and 81.2% accuracy on Imagenette (a subset of ImageNet), with 40 and 20 timesteps for evaluation, respectively (10 timesteps are used for training).

V. Impact of input encoding, training and network architecture width on the accuracy-latency trade-off

Finally, as SNNs distribute the information through binary events over time, there is an inherent trade-off between accuracy and latency (and thus the energy cost). Indeed, the latency directly impacts the accuracy, as the precision of the coding depends on the number of timesteps used for inference [35], [36], [38]. In order for SNNs to replace ANNs for efficient inference on neuromorphic hardware, the accuracy-latency trade-off should be carefully considered. In particular, parameters such as the coding, training strategies, and network architecture width, impact this trade-off. State-of-the-art approaches in conversion, direct training with BPTT, and hybrid training, are compared in terms of accuracy-latency trade-off with regards to these parameters in Table III.

First, the training strategy appears to have an effect on the accuracy-latency trade-off. For instance, the conversion with temporal switch coding proposed in [39] yields better results than the conversion with rate coding of [38] when a reduced number of timesteps is used (256). This is explained by the use of a better threshold balancing. Moreover, fine-tuning with BPTT after the rate-coded conversion in [66] improves the accuracy by 1% on CIFAR-10 and by 3% (resp. 5%) on ImageNet when using a VGG (resp. ResNet) architectures with the same number of timesteps (250). Adding further the fine-tuning of the threshold and leak improves also the accuracy by 1% (resp. 5%) on CIFAR-10 with ResNet architecture (resp. ImageNet with VGG architecture) when using the same number of timesteps (5) [70].

Considering the network architecture, as mentioned in section IV-C, increasing the width can improve the accuracy-latency trade-off. For instance, we observe that the gap between the SNN and ANN accuracy decreases as much as the architecture width increases. This explains the bigger differences between SNN and ANN accuracy on ImageNet with ResNet-34 architecture (21M parameters) than with VGG-16 (138M parameters). For instance, 69.00% accuracy is achieved with 5 timesteps using VGG-16 in [70], which is close to the ANN accuracy (71.59% without BN). The best accuracy among SNNs for a 34-layer ResNet is 67.05% with the spatio-temporal approach (trained with BN) of [57] by using a wide ResNet ($\approx$85M parameters). The problem is mitigated by [68] with a modified implementation of the spiking ResNet achieving 67.04% accuracy with 4 timesteps using the original ResNet-34 architecture. However, both SNNs are still far from the ANN ResNet-34 accuracy (73.31% with BN). Similarly in conversion approaches [38], [39], using the ResNet-34 architecture on ImageNet with a reduced number of timesteps (256 vs 4096), the degradation in accuracy is larger than the one observed for VGG-16. This highlights the accuracy-size trade-off in SNNs: the loss in accuracy due to the quantization of information (which is further increased by reducing the number of inference timesteps) is compensated by increasing the number of neurons in each layer of the network.

In addition, the encoding layer impacts a lot the accuracy-latency trade-off for both hybrid conversion and direct training approaches. Indeed, when compared to the same architecture (ResNet-20 large or VGG-16) with the same training methodology (i.e. conversion with fine-tuning), the encoding layer can allow to reduce the number of timesteps from 250 to 5 with only 1 to 2% accuracy loss [70]. In addition, direct training approaches yielding high accuracy with very few timesteps (5 to 12), such as [51], [54], [56], [57], use the encoding layer. The paper [70] studies the effect of the input encoding and leak and threshold optimization on both the spike rate and latency for the hybrid conversion with fine-tuning approach (with VGG-16 on CIFAR-10). Using an encoding layer instead of probabilistic sampling induces a significant improvement in the latency (from 150 to 25 timesteps)and in the spike rate (from 26 to 1.94).

Hence, it seems that the training strategy, the network architecture width and the use of an encoding layer impact the accuracy-latency trade-off. In particular, the use of wide architectures and encoding layer seems to be the key to compensate for the quantization process inherent to the spike coding with a limited number of timesteps and thus to reach the best accuracy-latency trade-off. However, they bring additional costs, such as using MACs operations for the encoding

TABLE III
IMPACT OF INPUT ENCODING, TRAINING AND NETWORK ARCHITECTURE WIDTH ON THE ACCURACY-LATENCY TRADE-OFF

Paper	Architecture	Encoding layer	Training	Timesteps	Acc. (%)
CIFAR-10					
[38]	ResNet-20	✗	conversion (rate)	2048	91.36
[38]	ResNet-20	✗	conversion (rate)	256	89.37
[39]	ResNet-20	✗	conversion (time)	2048	91.42
[39]	ResNet-20	✗	conversion (time)	256	90.10
[40]	ResNet-20	✓	conversion (rate)	16	91.62
[66]	ResNet-20 (L)	✗	conversion (rate)	250	91.12
[66]	ResNet-20 (L)	✗	conversion (rate) + backpropagation	250	92.22
[70]	ResNet-20 (L)	✓	conversion (rate) + backpropagation	5	90.29
[70]	ResNet-20 (L)	✓	conversion (rate) + backpropagation (+ leak & threshold tuning)	5	91.78
[57]	ResNet-19 (L)	✓	backpropagation	6	93.16
[53]	ResNet-11 (L)	✗	backpropagation	100	90.95
[58]	ResNet-11 (L)	✗	backpropagation (+ batch normalization + surrogate gradient tuning)	20	90.20
[72]	ResNet-20	/	ANN (+ batch normalization)	/	91.25
[70]	ResNet-20 (L)	/	ANN	/	92.79
ImageNet					
[38]	ResNet-34	✗	conversion (rate)	4096	69.89
[38]	ResNet-34	✗	conversion (rate)	256	≈20
[39]	ResNet-34	✗	conversion (time)	4096	69.93
[39]	ResNet-34	✗	conversion (time)	256	55.65
[40]	ResNet-34	✓	conversion (rate)	64	72.35
[66]	ResNet-34 (M)	✗	conversion (rate)	250	56.87
[66]	ResNet-34 (M)	✗	conversion (rate) + backpropagation	250	61.48
[57]	ResNet-34	✓	backpropagation (+ batch normalization)	6	63.72
[57]	ResNet-34 (L)	✓	backpropagation (+ batch normalization)	6	67.05
[68]	ResNet-34	✓	backpropagation (+ batch normalization)	4	67.04
[73]	ResNet-34	/	ANN (+ batch normalization)	/	73.31
[38]	VGG-16	✗	conversion (rate)	4096	73.09
[38]	VGG-16	✗	conversion (rate)	256	48.32
[39]	VGG-16	✗	conversion (time)	2560	73.46
[39]	VGG-16	✗	conversion (time)	256	69.71
[40]	VGG-16	✓	conversion (rate)	64	72.85
[66]	VGG-16	✗	conversion (rate)	250	62.73
[66]	VGG-16	✗	conversion (rate) + backpropagation	250	65.19
[70]	VGG-16	✓	conversion (rate) + backpropagation	5	64.32
[70]	VGG-16	✓	conversion (rate) + backpropagation (+ leak & threshold tuning)	5	69.00
[73]	VGG-16	/	ANN (+ batch normalization)	/	73.36

Number of parameters of the architectures (estimated according to the details given in the associated papers): ResNet-20: 0.27M. ResNet-20 (L): 11M. ResNet-19 (L): 13M. ResNet-11 (L): 18M. ResNet-34: 21M. ResNet-34 (M): 22M. ResNet-34 (L): 85M. VGG-16: 138M.

layer, and increasing the area and memory requirements for large architectures compared to smaller one.

VI. CONCLUSION

Compared to ANNs, the SNN model is more complex and requires to choose the neuron, synapse and coding strategies. These choices are intertwined and impact the learning rule and the system efficiency. Related to SNN training, the conversion and direct training backpropagation-based approaches have the potential to replace ANNs for accurate and energy-efficient hardware inference on typical deep learning tasks such as image classification. From the conversion approaches, the temporal switch coding [38] is a promising strategy to increase the spike sparsity. Moreover, it is important to use ANN model which are closest to SNN models to minimize the conversion errors [40]. From the direct training methods, the spatio-temporal approaches have demonstrated the best results. Only two algorithms [45], [64] from the spatial and single-approaches were shown to scale, but they lack information related to the number of timesteps required for inference. Thus, further work is needed to evaluate the performance of those approaches compared to the spatio-temporal ones.

Conversion methods suffer from high latency while direct training spatio-temporal approaches show an increased offline training cost and a degraded accuracy. Thus, to reach ANN capabilities, further methods to improve SNN training and inference are necessary, such as applying regularization techniques during training and tuning parameters specific to SNNs.

Moreover, feeding directly real values at the inputs with an encoding layer instead of performing spike conversion improves both SNN accuracy and latency. In the same spirit, the hybrid ANN-SNN approach mixing ANN and SNN layers also has a potential to improve the accuracy-latency trade-off and should be further explored. The hybrid training approaches such as conversion pre-training and supervised fine-tuning in [66], [70] or tandem ANN-SNN learning [76], can improve the accuracy, latency and spike sparsity. In addition, they reduce the cost of the offline training due to the use of ANN training, which was a hurdle to direct training approaches. Furthermore, wide network architectures allow reducing the accuracy loss when the number of timesteps is limited. This shows the importance of finding network architectures suitable for SNNs, which can be done using Neural Architecture Search [82]. Finally, the accuracy-latency trade-off in SNN should be carefully considered. For instance, the use of an encoding layer and wide architectures can significantly improve this trade-off.

In the interests of comparing the different training algorithms, we have focused on image classifications tasks. However, SNNs may offer greater benefits in other applications, but these remain to be fully explored [28]. For instance, some works have studied SNNs for speech processing [50], [83], biomedical applications [84]–[89] or in the context of privacy and security scenarios [90]–[92] including federated learning [93], [94]. These works show a broad range of applications where SNNs could bring both high accuracy and energy-efficient solutions.

The sparsity of activations and event-based processing result in efficient SNNs in neuromorphic hardware. However, note that, although the sparsity and the number of timesteps provide initial elements of comparison at the algorithmic level, they are not sufficient to accurately evaluate the efficiency of an algorithm implemented in hardware. For instance, the energy consumption and the latency of the SNN hardware inference also depends on the complexity of the neuron model and the network architecture. Moreover, the characteristics of SNNs may be particularly suited to the constraints of memristor-based neural networks accelerators. Indeed, activation sparsity and information coding through time could mitigate the current limitations of this emerging hardware, such as heat dissipation, device variability or limited variable precision [95]. Hence, we emphasize the importance of a hardware-software co-design strategy in order to enable highly energy-efficient solutions.

References

[1] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Toronto, Ontario, Tech. Rep. 0, 2009.

[2] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587.

[3] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[4] A. Esteva *et al.*, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, pp. 115–118, 2017.

[5] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[6] D. Li, X. Chen, M. Becchi, and Z. Zong, "Evaluating the energy efficiency of deep convolutional neural networks on cpus and gpus," in *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, 2016, pp. 477–484.

[7] L. Lapicque, "Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization," *Journal of Physiol Pathol Générale*, no. 9, pp. 620–635, 1907.

[8] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, 2014.

[9] E. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks," *IEEE Signal Processing Magazine*, vol. 36, pp. 51–63, 11 2019.

[10] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification," *Frontiers in Neuroscience*, vol. 11, p. 682, 2017.

[11] M. Pfeiffer and T. Pfeil, "Deep Learning With Spiking Neurons: Opportunities and Challenges," *Frontiers in Neuroscience*, vol. 12, 2018.

[12] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[13] P. A. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[14] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker Project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.

[15] J. Schemmel, D. Briiderle, A. Griibl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. IEEE, 2010, pp. 1947–1950.

[16] B. V. Benjamin *et al.*, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014.

[17] S. Yin *et al.*, "Algorithm and hardware design of discrete-time spiking neural networks based on back propagation with binary activations," in *2017 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, 2017, pp. 1–5.

[18] H. Mostafa, B. U. Pedroni, S. Sheik, and G. Cauwenberghs, "Fast classification using sparsely active spiking networks," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017, pp. 1–4.

[19] B. Rueckauer and S. Liu, "Conversion of analog to spiking neural networks using sparse temporal coding," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–5.

[20] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A Scalable Multicore Architecture With Heterogeneous Memory Structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs)," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, no. 1, pp. 106–122, Feb. 2018.

[21] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 10–14.

[22] Y. Dan and M.-M. Poo, "Spike timing-dependent plasticity: From synapse to perception," *Physiological Reviews*, vol. 86, no. 3, pp. 1033–1048, 2006.

[23] A. Tavanaei and A. Maida, "BP-STDP: Approximating backpropagation using spike timing dependent plasticity," *Neurocomputing*, vol. 330, pp. 39–47, 2019.

[24] J. K. Eshraghian *et al.*, "Training Spiking Neural Networks Using Lessons From Deep Learning," *arXiv:2109.12894 [cs]*, Jan. 2022.

[25] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. S. Maida, "Deep learning in spiking neural networks," *Neural Networks*, vol. 111, pp. 47–63, 2019.

[26] M. Bouvier *et al.*, "Spiking neural networks hardware implementations and challenges: A survey," *J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 2, pp. 1–35, 2019.

[27] C. S. Han and K. M. Lee, "A Survey on Spiking Neural Networks," *International Journal of Fuzzy Logic and Intelligent Systems*, vol. 21, no. 4, pp. 317–337, 12 2021.

[28] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, Nov. 2019.

[29] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of Physiology*, vol. 117, no. 4, pp. 500–544, 1952.

[30] E. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

[31] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-Temporal Backpropagation for Training High-performance Spiking Neural Networks," *Frontiers in Neuroscience*, vol. 12, p. 331, May 2018.

[32] D. Auge, J. Hille, E. Mueller, and A. Knoll, "A survey of encoding techniques for signal processing in spiking neural networks," *Neural Process. Lett.*, vol. 53, no. 6, p. 4693–4710, dec 2021.

[33] C. Stöckl and W. Maass, "Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes," *Nature Machine Intelligence*, vol. 3, 03 2021.

[34] H. Mostafa, "Supervised learning based on temporal coding in spiking neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 7, pp. 3227–3235, 2018.

[35] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2015, pp. 1–8.

[36] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: VGG and residual architectures," *Frontiers in Neuroscience*, vol. 13, p. 95, 2019.

[37] Y. Cao, Y. Chen, and D. Khosla, "Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, May 2015.

[38] B. Han, G. Srinivasan, and K. Roy, "Rmp-snn: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13 555–13 564, 2020.

[39] B. Han and K. Roy, "Deep spiking neural network: Energy efficiency through time based coding," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Springer International Publishing, 2020, pp. 388–404.

[40] T. Bu, W. Fang, J. Ding, P. Dai, Z. Yu, and T. Huang, "Optimal ANN-SNN conversion for high-accuracy and ultra-low-latency spiking neural networks," in *International Conference on Learning Representations*, 2022.

[41] N. Rathi, A. Agrawal, C. Lee, A. K. Kosta, and K. Roy, "Exploring spike-based learning for neuromorphic computing: Prospects and perspectives," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 902–907.

[42] G. Hinton, "Neural networks for machine learning, coursera," *Coursera, video lectures*, 2012.

[43] Y. Bengio, N. Leonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv:1308.3432 [cs.LG]*, 2013.

[44] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training Deep Spiking Neural Networks Using Backpropagation," *Frontiers in Neuroscience*, vol. 10, 2016.

[45] J. C. Thiele, O. Bichler, and A. Dupret, "SpikeGrad: An ANN-equivalent Computation Model for Implementing Backpropagation with Spikes," *arXiv:1906.00851 [cs]*, Jun. 2019.

[46] J. Wu, Y. Chua, M. Zhang, Q. Yang, G. Li, and H. Li, "Deep spiking neural network with spike count based learning rule," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–6.

[47] S. B. Shrestha and G. Orchard, "SLAYER: Spike layer error reassignment in time," in *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[48] F. Zenke and S. Ganguli, "SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks," *Neural computation*, vol. 30, no. 6, pp. 1514–1541, Jun. 2018.

[49] Y. Jin, W. Zhang, and P. Li, "Hybrid macro/micro level backpropagation for training deep spiking neural networks," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18, Dec. 2018, pp. 7005–7015.

[50] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, "Long short-term memory and learning-to-learn in networks of spiking neurons," *arXiv:1803.09574 [cs, q-bio]*, Dec. 2018.

[51] Y. Wu, L. Deng, G. Li, J. Zhu, Y. Xie, and L. Shi, "Direct training for spiking neural networks: Faster, larger, better," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 1, pp. 1311–1318, Jul. 2019.

[52] W. Zhang and P. Li, "Spike-train level backpropagation for training deep recurrent spiking neural networks," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

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

[54] W. Zhang and P. Li, "Temporal spike sequence learning via backpropagation for deep spiking neural networks," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 12 022–12 033.

[55] H. Fang, A. Shrestha, Z. Zhao, and Q. Qiu, "Exploiting neuron and synapse filter dynamics in spatial temporal learning of deep spiking neural network," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, Jul. 2020, pp. 2799–2806.

[56] W. Fang, Z. Yu, Y. Chen, T. Masquelier, T. Huang, and Y. Tian, "Incorporating learnable membrane time constant to enhance learning of spiking neural networks," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 2641–2651.

[57] H. Zheng, Y. Wu, L. Deng, Y. Hu, and G. Li, "Going deeper with directly-trained larger spiking neural networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, pp. 11 062–11 070, May 2021.

[58] E. Ledinauskas, J. Ruseckas, A. Juršėnas, and G. Buračas, "Training deep spiking neural networks," Jun. 2020. [Online]. Available: http://arxiv.org/abs/2006.04436

[59] I.-M. Comşa, K. Potempa, L. Versari, T. Fischbacher, A. Gesmundo, and J. Alakuijala, "Temporal coding in spiking neural networks with alpha synaptic function: Learning with backpropagation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 10, pp. 5939–5952, 2022.

[60] J. Göltz *et al.*, "Fast and deep neuromorphic learning with first-spike coding," in *Proceedings of the Neuro-inspired Computational Elements Workshop*, ser. NICE '20, Mar. 2020, pp. 1–3.

[61] M. Zhang *et al.*, "Spike-timing-dependent back propagation in deep spiking neural networks," Mar. 2020. [Online]. Available: http://arxiv.org/abs/2003.11837

[62] Y. Sakemi, K. Morino, T. Morie, and K. Aihara, "A supervised learning algorithm for multilayer spiking neural networks based on temporal coding toward energy-efficient vlsi processor design," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 1, pp. 394–408, 2023.

[63] S. R. Kheradpisheh and T. Masquelier, "S4nn: temporal backpropagation for spiking neural networks with one spike per neuron," vol. 30, no. 6, p. 2050027, 2020. [Online]. Available: http://arxiv.org/abs/1910.09495

[64] S. Zhou, X. Li, Y. Chen, S. T. Chandrasekaran, and A. Sanyal, "Temporal-coded deep spiking neural network with easy training and robust performance," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, pp. 11 143–11 151, May 2021.

[65] S. M. Bohte, J. N. Kok, and H. La Poutré, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1, pp. 17–37, 2002.

[66] N. Rathi, G. Srinivasan, P. Panda, and K. Roy, "Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation," 2020. [Online]. Available: http://arxiv.org/abs/2005.01807

[67] Y. Kim and P. Panda, "Revisiting Batch Normalization for Training Low-Latency Deep Spiking Neural Networks From Scratch," *Frontiers in Neuroscience*, vol. 15, 2021.

[68] W. Fang, Z. Yu, Y. Chen, T. Huang, T. Masquelier, and Y. Tian, "Deep Residual Learning in Spiking Neural Networks," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 21 056–21 069.

[69] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, p. 1929–1958, Jan. 2014.

[70] N. Rathi and K. Roy, "Diet-snn: A low-latency spiking neural network with direct input encoding and leakage and threshold optimization," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–9, 2021.

[71] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15, 2015, p. 448–456.

[72] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[73] Pytorch torchvision models. [Online]. Available: https://pytorch.org/vision/stable/models.html

[74] I. Garg, S. S. Chowdhury, and K. Roy, “DCT-SNN: Using DCT to distribute spatial information over time for learning low-latency spiking neural networks,” Oct. 2020. [Online]. Available: http://arxiv.org/abs/2010.01795

[75] L. Deng *et al.*, “Rethinking the performance comparison between SNNS and ANNS,” *Neural Networks*, vol. 121, pp. 294–307, Jan. 2020.

[76] J. Wu, Y. Chua, M. Zhang, G. Li, H. Li, and K. C. Tan, “A tandem learning rule for effective training and rapid inference of deep spiking neural networks,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 1, pp. 446–460, 2023.

[77] Y. Kim, H. Park, A. Moitra, A. Bhattacharjee, Y. Venkatesha, and P. Panda, “Rate coding or direct coding: Which one is better for accurate, robust, and energy-efficient spiking neural networks?” in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 71–75.

[78] P. Panda, S. A. Aketi, and K. Roy, “Toward scalable, efficient, and accurate deep spiking neural networks with backward residual connections, stochastic softmax, and hybridization,” *Frontiers in Neuroscience*, vol. 14, 2020.

[79] A. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr, “WRPN: Wide reduced-precision networks,” Sep. 2017. [Online]. Available: http://arxiv.org/abs/1709.01134

[80] S. Han, J. Pool, J. Tran, and W. J. Dally, “Learning both weights and connections for efficient neural networks,” in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS’15, 2015, p. 1135–1143.

[81] F. Zenke and T. P. Vogels, “The Remarkable Robustness of Surrogate Gradient Learning for Instilling Complex Function in Spiking Neural Networks,” *Neural Computation*, vol. 33, no. 4, pp. 899–925, 03 2021.

[82] Y. Kim, Y. Li, H. Park, Y. Venkatesha, and P. Panda, “Neural Architecture Search for Spiking Neural Networks,” in *Computer Vision – ECCV 2022*, S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, and T. Hassner, Eds. Cham: Springer Nature Switzerland, 2022, pp. 36–56.

[83] J. Wu, E. Yılmaz, M. Zhang, H. Li, and K. Tan, “Deep Spiking Neural Networks for Large Vocabulary Automatic Speech Recognition,” *Frontiers in Neuroscience*, vol. 14, Mar. 2020.

[84] N. K. Kasabov, “NeuCube: a spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data,” *Neural Networks: The Official Journal of the International Neural Network Society*, vol. 52, pp. 62–76, Apr. 2014.

[85] Y. Luo *et al.*, “EEG-Based Emotion Classification Using Spiking Neural Networks,” *IEEE Access*, vol. 8, pp. 46 007–46 016, 2020.

[86] E. Ceolini *et al.*, “Hand-gesture recognition based on EMG and event-based camera sensor fusion: A benchmark in neuromorphic computing,” *Frontiers in Neuroscience*, vol. 14, 2020.

[87] M. R. Azghadi *et al.*, “Hardware implementation of deep network accelerators towards healthcare and biomedical applications,” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 14, no. 6, pp. 1138–1159, 2020.

[88] G. Zhan *et al.*, “Applications of spiking neural network in brain computer interface,” in *9th International Winter Conference on Brain-Computer Interface (BCI)*, 2021, pp. 1–6.

[89] K. Kumarasinghe, N. Kasabov, and D. Taylor, “Brain-inspired spiking neural networks for decoding and understanding muscle activity and kinematics from electroencephalography signals during hand movements,” *Scientific Reports*, vol. 11, no. 1, p. 2486, Dec. 2021.

[90] S. Sharmin, N. Rathi, P. Panda, and K. Roy, “Inherent Adversarial Robustness of Deep Spiking Neural Networks: Effects of Discrete Input Encoding and Non-linear Activations,” in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Springer International Publishing, 2020, pp. 399–414.

[91] Y. Kim and P. Panda, “Visual explanations from spiking neural networks using inter-spike intervals,” *Scientific Reports*, vol. 11, no. 1, p. 19037, Sep. 2021.

[92] Y. Kim, Y. Venkatesha, and P. Panda, “PrivateSNN: Privacy-Preserving Spiking Neural Networks,” in *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*. AAAI Press, 2022, pp. 1192–1200.

[93] N. Skatchkovsky, H. Jang, and O. Simeone, “Federated neuromorphic learning of spiking neural networks for low-power edge intelligence,” in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 8524–8528.

[94] Y. Venkatesha, Y. Kim, L. Tassiulas, and P. Panda, “Federated learning with spiking neural networks,” *IEEE Transactions on Signal Processing*, vol. 69, pp. 6183–6194, 2021.

[95] J. K. Eshraghian, X. Wang, and W. D. Lu, “Memristor-based binarized spiking neural networks: Challenges and applications,” *IEEE Nanotechnology Magazine*, vol. 16, no. 2, pp. 14–23, 2022.

Manon Dampfhoffer received the M.Sc. degree in Informatics and Applied Mathematics from the Grenoble Institute of Technology, France, in 2019. She is currently working toward the Ph.D. degree at Univ. Grenoble Alpes, France, in the Systems-on-Chip and Advanced Technologies (LSTA) laboratory at CEA LIST and Spintec laboratory. Her current research focuses on models and algorithms for implementing energy-efficient Spiking Neural Networks on neuromorphic hardware at the edge.

Thomas Mesquida joined CEA in 2019, after a PhD in microelectronics. His past research included information encoding within Spiking Neural Network, with a focus on inter-spike interval, its hardware implementations and applications. He is currently pursuing the development lightweight spiking and hybrid neural network implementations for embedded applications, combining memory technology, information encoding and learning method for quantized networks.

Alexandre Valentian joined CEA LETI in 2005, after an MSc and a PhD in microelectronics. His past research activities included design technology co-optimization, promoting the FDSOI technology (notably through his participation in the SOI Academy), 2.5D/3D integration technologies and non-volatile memory technology. He is currently pursuing the development of bio-inspired circuits for AI, combining memory technology, information encoding and dedicated learning methods. Since 2020, he heads the Systems-on-Chip and Advanced Technologies (LSTA) laboratory at CEA LIST. Dr Valentian has authored or co-authored 80 conference and journal papers.

Lorena Anghel received the Ph.D. degree *(cum laude)* from Grenoble INP in 2000. From 2016 to 2020, she was the Vice President of Grenoble INP, in charge of industrial relationships, where she is currently the Scientific Director. She is also a Full Professor with Grenoble INP and a member of the Research Staff of the Spintec Laboratory. She has published more than 130 publications in international conferences and symposia. She has supervised 24 Ph.D. students. Her research interests include hardware design and test of neural networks, on-line testing, fault tolerance, and reliable design and verification. She was a recipient of several best paper and outstanding paper awards. She had fulfilled positions, such as the General Chair and the Program Chair for many prestigious IEEE conferences including the IEEE VTS, the IEEE ETS, and the IEEE On-Line Test Symposium.