Progressive Layer-based Compression for Convolutional Spiking Neural Network

Hammouda Elbez, Mazdak Fatahi

Progressive Layer-based Compression for Convolutional Spiking Neural Network

Hammouda Elbez [1,*] **and Mazdak Fatahi** [1]

[1] *Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL - F-59000 Lille, France*

Correspondence*:
Hammouda Elbez
hammouda.elbez@univ-lille.fr

ABSTRACT

Spiking neural networks (SNNs) have attracted interest in recent years due to their low energy consumption and the increasing need for more power in real-life machine learning applications. Having those bio-inspired networks on neuromorphic hardware for extra-low energy consumption is another exciting aspect of this technology. Furthermore, many works discuss the improvement of SNNs in terms of performance and hardware implementation. This paper presents a progressive layer-based compression approach applied to convolutional spiking neural networks trained either with Spike Time Dependent Plasticity (STDP) or Surrogate Gradient (SG). Moreover, we study the effect of this approach when used with SpiNNaker. This approach, inspired by neuroplasticity, produces highly compressed networks (up to 90% compression rate per layer) while preserving most of the network performance, as shown by experimental results on MNIST, FMNIST, Caltech face/motorbike, and CIFAR-10 datasets.

Keywords: Spiking Neural Network, Neuromorphic Computing, Compression, STDP, Surrogate Gradient, SpiNNaker

1 INTRODUCTION

In recent years, the use of neural networks in real-life applications has risen significantly due to the progress in the field. However, the current progress increased the complexity of the used models, resulting in more resource-hungry models that the Von Neumann architecture cannot guarantee. Spiking neural networks (SNNs) are considered a promising alternative to overcome Moore's law limitation, rise to the energy demands of modern network models, and provide a bio-inspired solution for lower energy consumption. SNN is inspired by brain functionality and uses spikes to communicate, guarantee low energy consumption, and the ability to process natural signals. Deploying SNNs using neuromorphic hardware is another promising way to have those benefits on more optimized architectures, making it possible to use such technology with energy-constrained applications.

Nowadays, complex and deep architectures are often necessary for better accuracy regarding neural network performance, which explains the rising complexity of the recent models, such as MobileNetV2 (Sandler et al., 2018), ResNet152 (He et al., 2016), and GPT-3 (Brown et al., 2020). Moreover, in SNN, increasing the size of the network helps improve performance and process complex data. As we can see in the existing works, some of them raised the number of neurons in the network to get a better accuracy. (Diehl and Cook, 2015), and other works preferred to use multiple layers, each with a group of neurons to get a better performance (Lee et al., 2016; Diehl et al., 2015; Kheradpisheh et al.,

2018). However, the number of neurons increases using large networks, and so does the internal activity and complexity. Therefore, it is more challenging to analyze the network behavior, especially with the use of spikes for communication and the asynchronous nature. Moreover, the increase in the network size implies an increase in the required resources to run, which will prevent their deployment on hardware using technologies like memristive crossbars (Merolla et al., 2011; Strukov et al., 2008), SpiNNaker (Furber et al., 2014), or Loihi (Davies et al., 2018).

In terms of hardware implementation, one of the techniques that we can use to overcome the complexity issue is pruning. Pruning compresses the network by reducing one or multiple network components, which results in a smaller size that can fit the hardware-limited resources and decreases computational operations. Pruning is inspired by the activity of the human brain in the early stages (Huttenlocher, 1979; Cun et al., 1990; Hassibi et al., 1994), where the brain losses neurons during the process of learning. The changes in synapse strength (known as neuroplasticity) happen not only in the early stages but throughout a person's lifespan. In biology, we have synaptotrophins and synaptotoxins responsible for synapse creation and elimination respectively (Sanes and Lichtman, 1999). We can identify three types of pruning when working with neural networks: Filter pruning (He et al., 2019; Huang et al., 2018; Li et al., 2016), weights pruning (Carreira-Perpinan and Idelbayev, 2018; Liu et al., 2018), and neuron-based pruning (Yu et al., 2018).

In the case of spiking neural networks, we can use pruning to reduce the size of the network. Shi et al. (Shi et al., 2019) presented a pruning method for SNNs on emerging non-volatile memory (eNVM) devices by exploiting the output firing characteristics of neurons. This technique is used during training and can maintain 90% classification accuracy on MNIST with up to 75% of the network pruned. Cho et al. (Cho et al., 2019) applied a distance-based pruning on a CMOS SNN chip, which decreases spikes activity by 52%. Rathi et al. (Rathi et al., 2019) combined weight quantization and pruning during the learning phase. Using a two-layer SNN of 6400 neurons and a static pruning threshold, they obtain a highly compressed network able to preserve a good performance, which is based on the Spike Time Dependent Plasticity (STDP) learning rule (Bi and Poo, 1998). In the work of Chen et al. (Chen et al., 2018), the authors used a three-phase prune process. The first two involve removing quiet neurons, and the third one concerns the removal of weak synapses. They used the prune operation as a part of the CNN to SNN conversion to reduce computational operations by 85%. Finally, Saunders et al. (Saunders et al., 2019) used a two-layer network of 900 neurons, and applied pruning once after the learning phase. Therefore, removing half of the synapses while preserving 90% network accuracy. For more deep spiking neural networks, we usually train using a global learning rule such as Surrogate Gradient (Neftci et al., 2019)or local learning rule like the Spike Time Dependent Plasticity (STDP). However, the pruning mechanism remains the same. Chen et al. (Chen et al., 2021) proposed a gradient rewiring technique (Grad R), an algorithm for learning weights and connectivity in a deep spiking neural network. As a result, the authors minimized the loss in terms of performance. Furthermore, they revealed a remarkable structure refining capability in SNN since they had a 3.5% loss in accuracy when using 0.73% connectivity. Nguyen et al. (Nguyen et al., 2021) presented connection pruning applied to a deep SNN, which is trained using STDP on FPGA. The approach consists of two stages: dynamic pruning during the on-chip learning and post-learning pruning after each layer. Using a weight update history value, the author calculated it using a proposed formula and compared it to a predefined threshold to prune. As a result, they achieved 2.1x speed-up and 64% energy saving during the on-chip learning. In the work of Faghihi et al. (Faghihi et al., 2022), a synaptic pruning-based SNN was presented, which uses a modified learning rule combined with a synaptic pruning method. Moreover, the prune operation is based on a defined threshold μ, resulting in a sparse neural connection between two layers that uses a few-shot-based classification method.

By looking at the literature, we can see that the existing works focus on when to prune (at the end or during network activity), what to prune (neurons or synapses), and how to treat the pruned element (hard-pruning or soft-pruning). Our contribution is the proposition of a novel technique for pruning threshold selection which is dynamic as opposed to existing works. The new threshold depends on the pruning rate of the previous prune operation. This work extends the previous work (Elbez et al., 2022) applied only on shallow networks to Convolutional Spiking Neural Networks (CSNN) by using a layer-based progressive pruning to get highly compressed layers. The compression rate increases when going more profound in the network (up to 98% compression rate). Moreover, the network performance is preserved compared to the baseline in the best-case scenario or records less than 3% accuracy loss in the worst-case scenario. We evaluated the efficacy of this approach by applying it to MNIST, FMNIST, Caltech face/motorbike, and CIFAR-10 datasets and analyzing the result when used with SpiNNaker.

We can resume our contribution in six points: 1) the extension of the progressive pruning and weight reinforcement techniques for convolutional spiking neural networks by adapting the first formula for multi-epoch training. 2) the application of progressive compression on networks trained with Spike Time Dependent Plasticity (STDP) or Surrogate Gradient (SG). 3) the study of the maximum pruning threshold value (α) effect on the network performance and compression rate. 4) proposing a layer-based version of this approach for more compression by setting the initial alpha value based on the depth of the actual layer. 5) testing this approach for the first time on SpiNNaker by deploying the compressed and baseline network on the board and estimating the reduced energy. 6) evaluating this technique on multiple datasets instead of only MNIST, which is the case in the previous work. Moreover, we share an opensource repository that contains the required code to reproduce the experiments using the csnn-simulator, Norse, and SpiNNaker [1]

2 MATERIALS AND METHODES

We observe the benefit of compressing a spiking neural network when we want to implement it on hardware. Due to limited resources, reducing the elements of a neural network can enable the deployment of more extensive networks, which is impossible without compression. Moreover, by reducing the network size, we also reduce the resources needed for the deployment. This section describes our approach, the network topology, the different datasets used in the experiments, and the SpiNNaker board.

2.1 The Progressive Compression

Progressive Compression involves two processes: Progressive Pruning (PP) and Dynamic Synaptic Weight Reinforcement (DSWR). Progressive Pruning (PP) eliminates connections between neurons from one layer to another. We perform this operation after each batch (group of inputs) during the training phase (Elbez et al., 2022). Using a dynamic pruning threshold $T_n, n \in \mathbb{N}$, which we calculate using equation (1).

$$T_{n+1} = T_n + \alpha * (C_{r_n}/C_n) \qquad n \in \mathbb{N} \tag{1}$$

α is a constant representing the initial threshold. T_n and T_{n+1} are the old and new threshold for the next batch, respectively. C_n represents the total number of synapses, and C_{r_n} the remaining synapses between the two layers at batch n. In (Elbez et al., 2022), the authors applied this approach to single-layer neural networks using the MNIST dataset, which proved effective in compressing the network. Therefore, in this work, we will extend this work and study the effect of this approach on convolutional spiking neural networks trained with STDP or SG.

[1] https://archive.softwareheritage.org/swh:1:rev:d8ead5b48684e309a58ceba04664b4849e9ae5c2

Dynamic Synaptic Weight Reinforcement (DSWR) is another process combined with the pruning operation, which concerns the maintained synapses after pruning. We can see this process in biology and the human brain as a part of the synaptogenesis process (Sanes and Lichtman, 1999). Moreover, by reinforcing the preserved synapses, we speed up their convergence toward one feature. The equation used to determine the amount of reinforcement depends on the currently calculated threshold, and it is done based on equation (2).

$$W_{n+1} = W_n + \beta * T_n, \qquad n \in \mathbb{N}, \quad W \in [0, 1] \tag{2}$$

β is a constant we define based on experiments. W_n and W_{n+1} are the concerned connection's current and new weights, respectively. In our work, for STDP-based networks, we keep the two constants α and β the same as in (Elbez et al., 2022) (α = 0.05 and β = 0.1). Those values were fixed using Pareto front multiobjective optimization (Deb, 2011) based on network accuracy and compression rate.

When applying the two formulas on a single-layer network, we trained for one epoch and used it after each batch of 10k inputs (in the case of MNIST). However, for deeper networks, we usually train for multiple epochs, and following the same method will cause the pruning threshold to be updated numerous times and destroy the network connectivity. Therefore, instead of compressing after each batch, we would compress after each epoch and add a constraint on the max possible pruning threshold value. If we reach this value, we do not update the threshold value. This approach is represented in Figure 1, and as a result, equation (3) represents the adapted version of the Progressive Pruning formula for STDP-based networks.

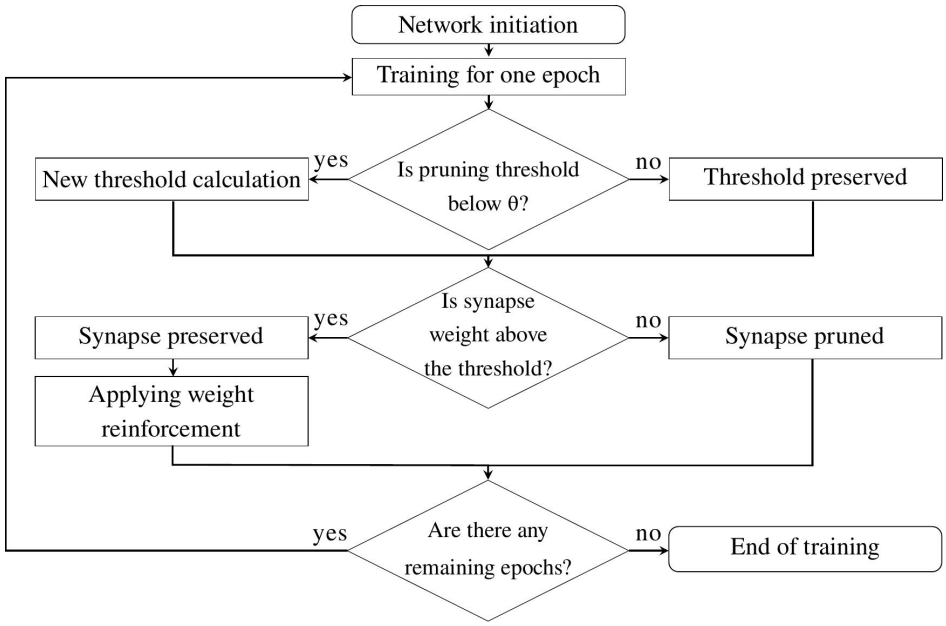

Figure 1. Compression flowchart for Convolutional SNN

$$T_{e+1} = T_e + \alpha * (C_{r_e}/C_e) \qquad e \in \mathbb{N}, \quad T_{e+1} \leq \theta \tag{3}$$

θ is the maximum pruning threshold, which is used to decide if we will compute a new threshold or not (θ = 0.3 in this work). Moreover, the reinforcement part stays the same, and we apply it after each prune operation.

In the case of SG-Based networks, since the weights in the network are not all positive, which is the case in STDP-Based ones. We need to update the Progressive Pruning formula in a way it can also support negative weights, otherwise, the network will be destroyed by setting all negative weights to zero. From equation (3), we will create two formulas (equation (4)) with θ^+ and θ^-, which represent the positive maximum pruning threshold and negative one, respectively. Positive and negative initial threshold ($\alpha^+ = 0.005$ and $\alpha^- = -0.005$). Finally, T_e^+ and T_e^- represent the positive and negative threshold values.

$$\begin{aligned} T_{e+1}^+ &= T_e^+ + \alpha^+ * (C_{r_e}/C_e) \qquad e \in \mathbb{N}, \quad T_{e+1}^+ \leq \theta^+ \\ T_{e+1}^- &= T_e^- + \alpha^- * (C_{r_e}/C_e) \qquad e \in \mathbb{N}, \quad T_{e+1}^- \geq \theta^- \end{aligned} \tag{4}$$

Since the range of the weights in the SG-Based networks using Norse is not similar, the selection of the positive θ^+ and negative θ^+ maximum pruning threshold is computed using equation 5. If we apply this equation on a network with positive weights $W \in [0, 1]$, we will get $\theta^+ = \theta = 0.3$, which is the initial threshold used with STDP-Based networks.

$$\begin{aligned} \theta^+ &= \theta * \max(W_L) \\ \theta^- &= \theta * \min(W_L) \end{aligned} \tag{5}$$

For applying the Dynamic Synaptic Weight Reinforcement (DSWR) with SG-Based networks, we also have $\beta^+ = 0.1$ and $\beta^- = -0.1$, and the reinforcement is applied based on equation (6).

$$\begin{aligned} W_{n+1}^+ &= W_n^+ + \beta^+ * T_e^+, \qquad W > 0 \\ W_{n+1}^- &= W_n^- + \beta^- * T_e^-, \qquad W < 0 \end{aligned} \tag{6}$$

Figure 2 shows how the threshold value changes during training (20 epochs) using our approach in both cases (STDP-Based and SG-Based). Moreover, we can also follow the evolution of the compression rate. After a couple of epochs, we can see in the case of the STDP-Based network (Figure 2 (A)) that the threshold value is stable (around 0.32) due to $\theta = 0.3$. Nevertheless, the compression keeps increasing due to the learning and the reinforcement applied in the network. Since we have two pruning thresholds for the SG-Based network, we can see in Figure 2 (B) how the thresholds change while the compression increases. After a couple of epochs, the two thresholds became stable (around 0.06 and -0.06) since $\theta^+ = 0.0595$ and $\theta^- = -0.0599$.

2.2 Network Topology

Our experiments use two approaches for training SNN: STDP and SG. This section presents the network topology used for each of the two approaches.

STDP-based networks:

For the STDP-based networks we use the same topology presented by Falez et al. (Falez et al., 2019), which is composed of multiple pairs of convolutional and max pooling layers. Those pairs of layers are followed by a dense layer and a support vector machine (SVM) for classification and decision-making.

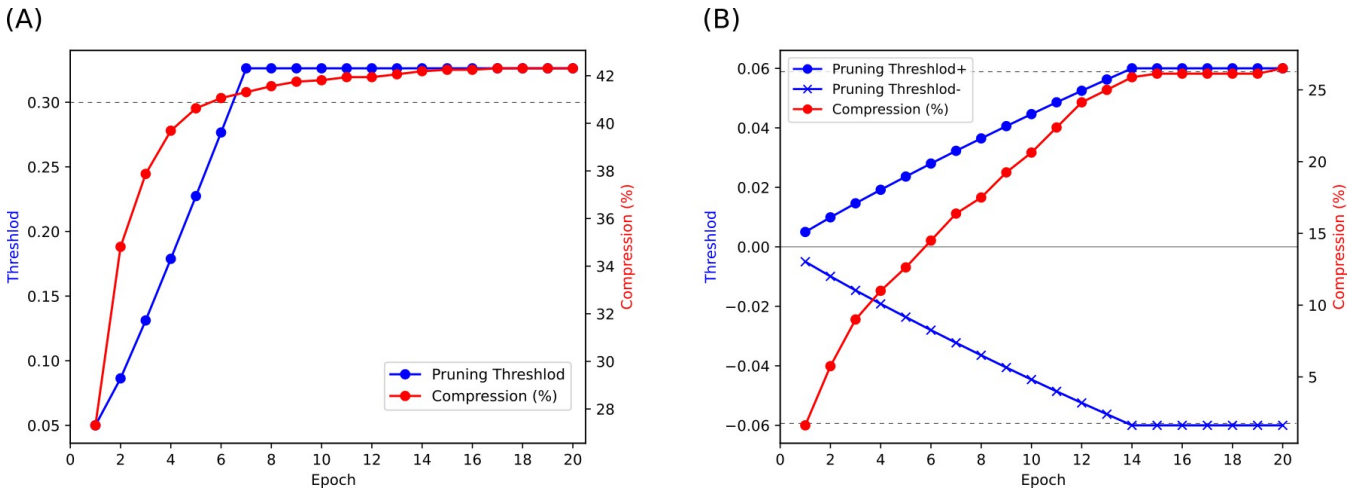

Figure 2. Compression rate and pruning threshold evolution for (A) STDP-Based network and (B) SG-Based network

Moreover, we use latency coding (Rullen and Thorpe, 2001) for handling spikes in the network. Finally, as preprocessing, we apply a difference-of-Gaussian (DoG) filter on the inputs to simulate on-center/off-center cells. Network topology is presented in Figure 3 (A). For the simulation, we use csnn-simulator (Falez, 2019), a C++-based open-source simulator [2].

We use integrate-and-fire (IF) neurons (Burkitt, 2006) in the different layers of the model. IF neuron model adds the input spikes into the membrane potential $v(t)$ until a threshold $v_{th}(t)$ is reached, resulting in the neuron firing and sending an output spike. Then, the membrane potential is reset to a defined value (0 in this work). This neuron model is represented by the following equation (7) (Falez, 2019).

$$C_m \frac{\partial v(t)}{\partial t} = \sum_{i \epsilon \varepsilon} v_i f_s(t - t_i), \qquad f_s(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \qquad (7)$$
$$v(t) = v_r \quad when \quad v(t) \geq v_{th}(t)$$

C_m represents the membrane capacitance, and v_i is the spike voltage of the i-th spike. Also, ε and f_s represent the set of incoming spikes and the kernel of spikes, respectively. Finally, t_i is the timestamp of the i-th spike.

What makes the neural network useful is being able to learn. In our work, we use the Spike Time Dependent Plasticity (STDP) learning rule, where the synaptic weight update depends on the spike time from neurons on both ends of the synapse (pre-neuron and post-neuron). The STDP we use is defined in equation (8).

$$\Delta_w = \begin{cases} \eta_w e^{-\frac{t_{\text{pre}} - t_{\text{post}}}{\tau_{\text{STDP}}}}, & \text{if } t_{\text{pre}} \leq t_{\text{post}} \\ -\eta_w e^{-\frac{t_{\text{pre}} - t_{\text{post}}}{\tau_{\text{STDP}}}}, & \text{otherwise} \end{cases} \qquad (8)$$

τ_{STDP} is the time constant for the STDP learning window, and η_w is the learning rate. t_{pre} and t_{post} represent the spiking time of pre-synaptic and post-synaptic neuron, respectively. We can see from the equation that

[2] https://archive.softwareheritage.org/swh:1:dir:fa5f52f0d1c769c1d4ebc691fd2dea61d3bbce5f

the update on the synaptic weight Δ_w can be either positive or negative, depending on which spike came first. In this work, the synaptic weights are between 0 and 1.

For the multi-layer network to work properly, two additional mechanisms are used. First, the Winner-takes-all (WTA) represents an inhibition mechanism to prevent neuron domination when learning and prevent multiple neurons in one layer from learning the same feature, which improves the network performance. Besides WTA, we need to add a homeostasis mechanism. Since we are using multi-layer SNN we will use the same neuron threshold adaptation technique presented in (Falez, 2019), which trains the neurons to fire at a given time t_{obj} to maintain the homeostasis. This technique is applied each time a neuron fires or gets inhibited. Every neuron's threshold is updated, so its firing time converges toward t_{obj}. The neuron threshold adaptation is presented by equations (9) and (10).

$$\begin{aligned} \Delta_{th}^{1} &= -\eta_{th}(t - t_{obj}) \\ \Delta_{th}^{2} &= \begin{cases} \eta_{th}, & \text{if } t_i = \min(t_0, ..., t_N) \\ -\frac{\eta_{th}}{l_d(n)}, & \text{otherwise} \end{cases} \end{aligned} \tag{9}$$

$$v_{th}(t) = max(th_{min}, v_{th}(t-1) + \Delta_{th}^{1} + \Delta_{th}^{2}) \tag{10}$$

η_{th} represents the threshold learning rate, and l_d is the number of neurons in competition in the layer. Furthermore, t and t_i are the spike timestamp of the neuron and the firing time of neuron i, respectively. Finally, th_{min} is the minimum possible neuron threshold value.

In our experiments, the different hyperparameters we used are:

1. Difference-of-Gaussian: DoG_{in} = 1.0, DoG_{out} = 4.0,DoG_{size} = 7.0
2. STDP: η_w = 0.1, τ_{STDP} = 0.1
3. Neuron Threshold Adaptation: η_{th} = 1.0, th_{min} = 1.0, $t_{obj^{CIFAR\text{-}10}}$= 0.95, $t_{obj^{MNIST}}$= $t_{obj^{FMNIST}}$= 0.75, $t_{obj^{Face/Motor}}$= 0.80

Moreover, we use default parameters for the SVM part of the network, which delivers good performance.

Surrogate Gradient-based networks:

For the Surrogate Gradient-based experiments, we use Norse (Pehle and Pedersen, 2021) for the simulation. Norse is a Python library that expands PyTorch with primitives for bio-inspired neural components, which allows us to train multilayer spiking neural networks using Surrogate Gradient.

We used an architecture similar to the STDP-based network (multiple pairs of convolutional and max pooling layers). However, the only difference is replacing the SVM part with a LILinearCell, which consists of a group of cells for a leaky-integrator (LI) with an additional linear weighting. We can see in Figure 3 (B) that when using Norse, each convolution or Dense layer is followed by a LIFCell, which consists of a group of leaky integrate-and-fire (LIF) neurons to process the output of that layer before going to the next one. Moreover, we use latency coding (Rullen and Thorpe, 2001) (SpikeLatencyLIFEncoder) before introducing the input to the network.

For training the network, we use the SuperSpike method (Zenke and Ganguli, 2018), which is a voltage-based global learning rule that can be interpreted as a nonlinear Hebbian three-factor rule. The learning

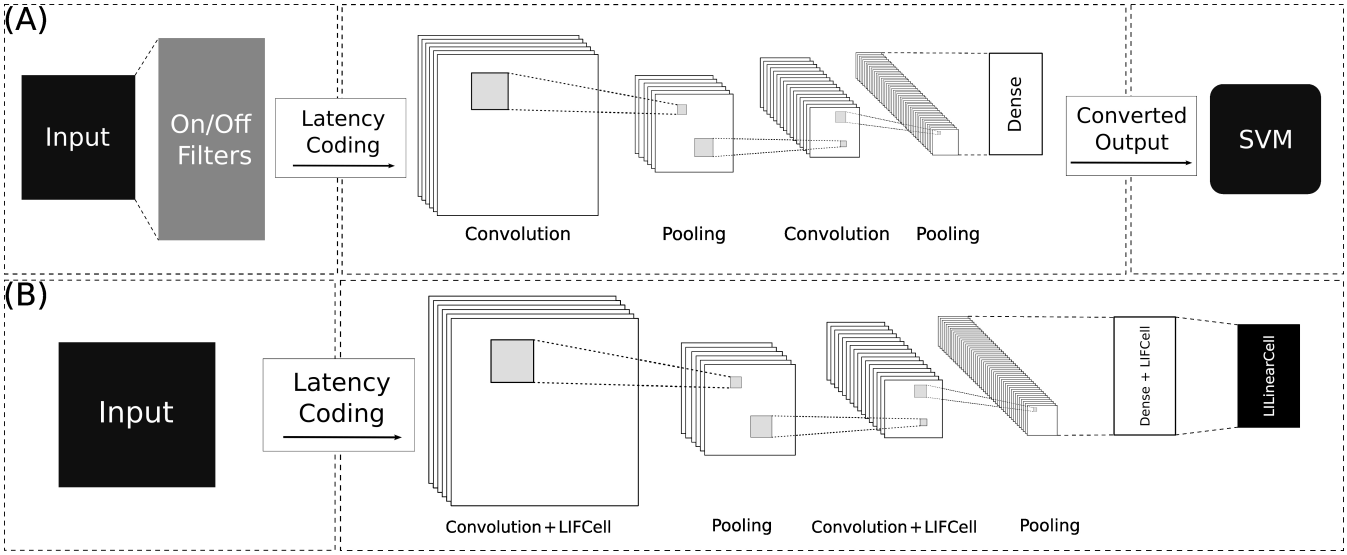

Figure 3. The network topology for (A) STDP-based networks (B) Surrogate Gradient-based networks

rule is defined in equation 11.

$$\Delta w_{ij}^{k} = r_{ij} \int_{t_k}^{t_{k+1}} \underbrace{e_i(s)}_{\text{Error signal}} \; \alpha * \left(\underbrace{\sigma^{'}(U_i(s))}_{\text{Post}} \underbrace{(\epsilon * S_j)(s)}_{\text{Pre}} \right) ds. \tag{11}$$

t_k is the spike time (k = 1,2,...). The error signal represents the third factor in this rule, with $e_i(s) = \alpha * (\hat{S}_i - S_i)$. S_i is the spike activity of pre-synaptic neuron i, $\hat{S}_i$ represents the target spike train for a given stimulus, and α is a normalized smooth temporal convolution kernel. r_{ij} is the learning rate for synapse ij. $\sigma^{'}(U_i(s))$ represents the derivative of a continuous auxiliary function σ of the membrane potential $U_i(s)$, and ϵ is the postsynaptic potential (PSP) shape.

For the SG-based experiments, the different hyperparameters we used are:

1. Latency coding: T = 35
2. LIFCell: V_{th} = 0.25
3. SuperSpike: α = 80, Optimizer = Adam, Learning rate = 0.001

2.3 Datasets

Using convolutional SNN, we can test the compression effect on the network using different datasets. In our experiments, we use MNIST (Lecun et al., 1998), composed of 28x28 pixel images of handwritten digits with labels from 0 to 9. MNIST contains 60,000 training images and 10,000 test images. FMNIST (Xiao et al., 2017) is similar to MNIST in terms of type, data dimensions, and dataset size. However, FMNIST contains clothes with greyscale images instead of handwritten digits. Caltech face/motorbike (Kheradpisheh et al., 2018) contains modified data from Caltech-100 by using only two classes (Face/Motor). Caltech face/motorbike images are converted to greyscale and resized to 160 pixels in height while preserving the aspect ratio, and it contains 400 train and 396 test images. Finally, CIFAR-10 (Krizhevsky and Hinton, 2009) consists of colored images composed of 32x32 pixel images of objects with ten classes. CIFAR-10 contains 50000 train images and 10000 test images.

2.4 SpiNNaker Board

Modeling large neural networks on Von Neumann architecture requires a lot of computing resources and power consumption (Sharp et al., 2012). SpiNNaker (Painkras et al., 2013) is one of the neuromorphic architectures (Basu et al., 2022) that was proposed to overcome the limitations and provide the requirements for spiking neural networks.

SpiNNaker is a biologically inspired, massively parallel computing system optimized for modeling and simulating large-scale real-time networks. In this work, we use the SpiNN-5 (SpiNNaker 103) board (Painkras et al., 2013) (Furber et al., 2013), which consists of 48 SpiNNaker chips. Each chip contains 18 ARM cores with a 32 kB ITCM (instruction tightly coupled memory) and a 64 kB DTCM (data tightly coupled memory) per core. Moreover, a 128 MB SDRAM is shared between the 18 cores. To imitate the high connectivity of the brain, the cores are interconnected by an asynchronous Network-on-Chip (NoC) through a multicast packet-routing mechanism. In addition, SpiNN-5 uses three Xilinx Spartan-6 FPGAs for high-speed serial links.

A 100 MB Ethernet controller handles the connection between the SpiNNaker board and the computer. We use it to load data to the SpiNNaker memory to perform a real-time simulation. Furthermore, the sPyNNaker (Rhodes et al., 2018) is a software package used to define models in PyNN script (Davison et al., 2009) and translates models into a suitable form for SpiNNaker.

3 RESULTS AND DISCUSSION

This section describes our experiments on the image classification task and presents the results using the csnn-simulator and Norse.

3.1 STDP-based networks

In the case of STDP-based networks, training is done layerwise by training each layer for 100 epochs. For the network architecture, the number of layers used varies from one dataset to another, but they all use SVM

for decision-making. Therefore, we use one convolutional layer for CIFAR-10, while two convolutional-max pooling layers and one dense layer are used with MNIST, FMNIST, and Caltech face/motorbike. More details about the different architectures are presented in Table 1.

Table 1. The architectures used in the experiments of STDP-based networks

Dataset	Parameters	Architecture				
		Conv1	Pool1	Conv2	Pool2	Fc1
CIFAR-10	filters (w, h, n)	(5, 5, 128)	—	—	—	—
	padding (w, h)	(0, 0)	—	—	—	—
	stride (w, h)	(1, 1)	—	—	—	—
MNIST & FMNIST	filters (w, h, n)	(5, 5, 32)	(2, 2)	(5, 5, 128)	(2, 2)	(4, 4, 1024)
	padding (w, h)	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)
	stride (w, h)	(1, 1)	(2, 2)	(1, 1)	(2, 2)	(1, 1)
Face/Motor	filters (w, h, n)	(5, 5, 32)	(7, 7)	(17, 17, 64)	(5, 5)	(5, 5, 128)
	padding (w, h)	(5/2, 5/2)	(7/2, 7/2)	(17/2, 17/2)	(5/2, 5/2)	(5/2, 5/2)
	stride (w, h)	(1, 1)	(6, 6)	(1, 1)	(5, 5)	(1, 1)

Note: w = width, h = height, n = number

In Table 2, we can see the effect of compressing the network using our approach on different datasets. We observe, in particular, the network classification rate, the compression rate on each layer, the duration of the training and testing phase, number of spikes per layer, and number of synaptic updates per layer. Finally, the comparison is made between the compressed network and the baseline (the same model without compression).

Table 2. Accuracy, compression, and layers activity (spikes & synaptic updates) for STDP-based networks

		Accuracy $\pm$ std	Compression/layer	Simulation time	Spikes/layer	Synaptic updates/layer
CIFAR-10	baseline	54.63 ± 0.37	—	2:21:34	$5\text{x}10^6$	$7.5\text{x}10^8$
	compressed	53.45 ± 0.28	19.91	2:13:33	$5\text{x}10^6$	$1.6\text{x}10^7$
Face/Motor	baseline	98.43 ± 0.10	—	0:41:51	39996|40000|40000	$1999810\|3.69\text{x}10^8\|6.40\text{x}10^7$
	compressed	87.82 ± 0.49	44.12|46.00|45.77	0:36:31	39995|40000|40000	$429385\|5.16\text{x}10^7\|8.22\text{x}10^6$
MNIST	baseline	98.18 ± 0.07	—	2:29:56	$3982386\|5954200\|6\text{x}10^6$	$1.99\text{x}10^8\|4.76\text{x}10^9\|1.22\text{x}10^{10}$
	compressed	96.55 ± 0.11	42.43|27.17|13.16	2:48:16	$3988197\|5950494\|6\text{x}10^6$	$4.13\text{x}10^7\|5.67\text{x}10^7\|1.24\text{x}10^8$
FMNIST	baseline	84.65 ± 0.21	—	3:50:05	$5604907\|5999531\|6\text{x}10^6$	$2.80\text{x}10^8\|4.79\text{x}10^9\|1.22\text{x}10^{10}$
	compressed	83.50 ± 0.31	50.25|27.17|13.17	3:52:29	$5609543\|5999693\|6\text{x}10^6$	$1.35\text{x}10^7\|5.51\text{x}10^7\|1.24\text{x}10^8$

By analyzing Table 2, we can see that in terms of the STDP-based networks performance, we have a small loss when compressing on some datasets and a considerable one on Face/Motor. Moreover, for the compression rate, we can observe that compression is not that high (19%) for a single-layer network (CIFAR-10). On the other hand, for multi-layer networks, we can see the compression rate for each layer. However, it is interesting that despite using different datasets, the compression rate is higher at the first layer, decreases when going deeper (MNIST & FMNIST), and is more stable in the case of Face/Motor. Furthermore, we can see a slight change in the simulation time when we use compression due to applying it during the training and the SVM (not concerned with compression) training part, which is time-consuming. Finally, for the layer activity (spikes & synaptic updates), we do not see a difference in spikes activity in

the last layer. However, some compressed layers have more activity than the baseline, which differs from what we expected. Furthermore, for the synaptic updates, we can see an apparent decrease in the activity of all layers of the compressed network, which is expected due to the compression.

Table 3. Max pruning threshold θ effect on the accuracy and compression for STDP-based networks

		Max pruning threshold θ				
		0.3	0.4	0.5	0.6	0.7
CIFAR-10	Accuracy $\pm$ std	53.45 $\pm$ 0.28	53.01 $\pm$ 0.33	53.49 $\pm$ 0.28	53.58 $\pm$ 0.45	**53.71 $\pm$ 0.41**
	Compression/layer	19.91	19.80	19.97	19.61	**19.91**
Face/Motor	Accuracy $\pm$ std	87.82 $\pm$ 0.49	89.29 $\pm$ 0.30	89.79 $\pm$ 0.61	90.50 $\pm$ 0.47	**90.95 $\pm$ 0.37**
	Compression/layer	44.12|46.00|45.77	41.56|47.67|47.26	42.93|48.73|47.49	41.62|49.41|47.24	**42.87|49.85|47.56**
MNIST	Accuracy $\pm$ std	96.55 $\pm$ 0.11	96.64 $\pm$ 0.19	**96.88 $\pm$ 0.19**	96.78 $\pm$ 0.18	96.80$\pm$ 0.34
	Compression/layer	42.43|27.17|13.16	42.12|27.22|13.16	**41.43|27.22|13.17**	41.75|27.18|13.16	43.00|27.29|13.16
FMNIST	Accuracy $\pm$ std	83.50 $\pm$ 0.31	83.41 $\pm$ 0.11	83.44 $\pm$ 0.23	83.46 $\pm$ 0.19	**83.61 $\pm$ 0.40**
	Compression/layer	50.25|27.17|13.17	50.68|27.25|13.17	50.56|27.17|13.17	50.31|27.23|13.10	**50.18|27.25|13.17**

In Table 2, we set the max pruning threshold $\theta = 0.3$ for the experiments. The selection of θ can impact the network compression since it defines the highest possible pruning threshold value, and if we set it too high ($\theta = 1$), the compression will destroy the network. Moreover, the value of θ may vary from one application to another and from one dataset to another. In Table 3, we explore the effect of increasing θ up to 0.7 on the network performance and compression rate. The experiments are applied using the same network architectures and the same datasets.

In Table 3, we can see that by increasing the maximum pruning threshold value θ, the compression rate is stable with a small increase in the network performance, which is visible on all the datasets. Therefore, for the STDP-based networks, we do not see a clear improvement in compression by increasing the maximum pruning threshold value θ. Regarding the network performance, we can see that the best-recorded accuracy across the experiments does not always use a specific θ value and changes from one dataset to another.

To understand why the compression rate is low on the STDP-based networks even when increasing the max pruning threshold and the accuracy is lower, we need to take a look at the threshold mechanism used in the network. The threshold mechanism is essential in the network for learning, and it helps the network maintain activity in the different layers by updating the threshold of the neurons. Therefore, any other mechanism that may affect the spike activity in the network may negatively impact the network by changing the neuron threshold, the synaptic weights, or the input value. In our work, the progressive compression, when applied, will reduce the number of synapses and reinforce the remaining synapses, affecting the synaptic weights and input value and conflicting with the existing threshold adaptation mechanism, which explains the low compression rate in the STDP-based networks, and the loss in accuracy. Moreover, in Table 2, we noticed that the spikes activity increases in the compressed network. Such an increase is now justified due to the threshold mechanisms reducing the threshold as a reaction to the compression.

In Figure 4 (A), we can see how the thresholds of neurons behave when we apply the compression. Therefore, it is visible that when the compression is applied (the red dotted line), there is a visible change in the thresholds, which is the threshold adaptation mechanism reaction that we mentioned before. In Figure 4 (B), we can see how the synaptic weights of one neuron are being updated due to the compression and the threshold adaptation. As we can see in the figure, the weights are initially between zero and one. However, once we compress the network after each epoch, we can see how the weights are being removed (going to zero), and at the same time, the other weights are going to one. This behavior remains the same

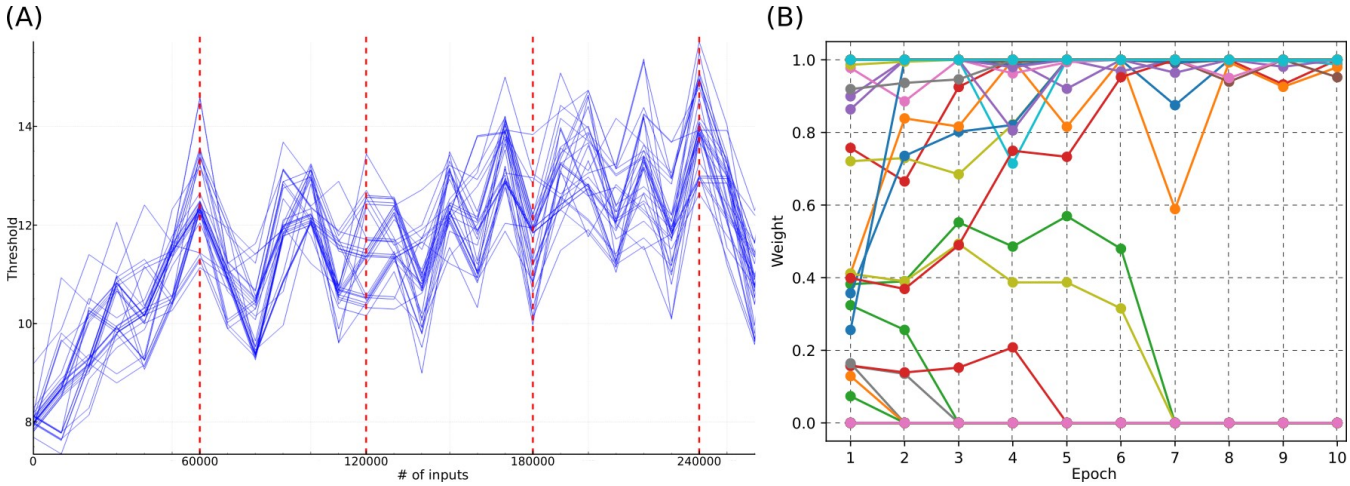

Figure 4. The threshold adaptation and progressive compression impact on (A) the thresholds of the neurons and (B) single neuron weights activity

even when we increase the maximum pruning threshold value θ, which also explains the low compression rate.

3.2 Surrogate Gradient-based networks

In the Surrogate Gradient-based networks, we train the network for 100 epochs. We use a similar architecture with parameters similar to STDP-based networks, with one additional dense layer to replace the SVM. Regarding datasets, we use MNIST, FMNIST, CIFAR-10, and Caltech face/motorbike. Table 4 presents more details about the used architectures.

Table 4. The architectures used in the experiments of SG-based networks

Dataset	Parameters	Architecture					
		Conv1	Pool1	Conv2	Pool2	Fc1	Fc2
MNIST, FMNIST, CIFAR-10	filters (w, h, n)	(5, 5, 32)	(2, 2)	(5, 5, 128)	(2, 2)	1024	10
	padding (w, h)	(0, 0)	(0, 0)	(0, 0)	(0, 0)	—	—
	stride (w, h)	(1, 1)	(2, 2)	(1, 1)	(2, 2)	—	—
Face/Motor	filters (w, h, n)	(5, 5, 32)	(7, 7)	(17, 17, 64)	(5, 5)	128	2
	padding (w, h)	(3, 3)	(3, 3)	(9, 9)	(2, 2)	—	—
	stride (w, h)	(1, 1)	(6, 6)	(1, 1)	(5, 5)	—	—

Note: w = width, h = height, n = number

Table 5 shows the compression effect on different datasets using SG-based networks. We observe, in particular, the network classification rate, the compression rate on each layer, the simulation time, and the number of trainable parameters. Moreover, we can compare the compressed network ($\theta = 0.3$) and the baseline based on those points.

We can see in Table 5 a slight decrease in the network accuracy when compressing. Compared to the baseline, we can see that for CIFAR-10 and FMNIST, we have a more significant loss (2%) and a minor loss for Face/Motor and MNIST. Moreover, in terms of the compression rate, we can see that compression varies from one layer to another, getting bigger once we move more profoundly in the network, which is observed in all the datasets. Furthermore, we can see that we have a slightly short simulation time in the

Table 5. Accuracy, compression, and layers activity (spikes & synaptic updates) for SG-based networks

		Accuracy ± std	Compression/layer	Simulation time	Trainable params
CIFAR-10	baseline	59.87 ± 0.45	—	1:30:08	3,391,840
	compressed	57.82 ± 0.16	33.42|55.36|72.71|67.65	1:22:23	952,210
Face/Motor	baseline	97.60 ± 1.11	—	0:30:28	1,036,320
	compressed	96.17 ± 2.94	26.88|55.92|51.14|61.56	0:26:32	461,144
MNIST	baseline	99.05 ± 0.13	—	0:55:14	2,210,592
	compressed	98.69 ± 0.14	41.25|85.31|73.31|66.22	1:08:13	1,008,487
FMNIST	baseline	85.63 ± 0.37	—	0:55:46	2,210,592
	compressed	83.21 ± 0.47	28.25|55.83|60.32|64.02	1:19:17	901,983

case of some datasets (CIFAR-10 and Face/Motor). However, for MNIST and FMNIST, we have a longer simulation time. Finally, for the trainable parameters, we can see that we have a considerable decrease in trainable parameters for all the datasets due to the compression.

Table 6. Positive and negative pruning threshold effect on the accuracy and compression for SG-based networks, by updating θ value

		θ value				
		0.3	0.4	0.5	0.6	0.7
CIFAR-10	Accuracy ± std	**57.82 ± 0.16**	56.91 ± 1.06	57.18 ± 0.67	57.32 ± 0.76	56.49 ± 0.71
	Compression/layer	**33.42|55.36|72.71|67.65**	41.50|61.51|72.32|72.44	48.13|66.73|80.06|75.25	52.96|70.78|79.91|76.71	58.38|78.01|85.24|77.92
Face/Motor	Accuracy ± std	96.17 ± 2.94	**97.93 ± 0.83**	96.63 ± 1.06	96.92 ± 1.12	97.13 ± 1.34
	Compression/layer	26.88|55.92|51.14|61.56	**39.63|58.11|51.32|64.45**	43.50|87.68|52.82|68.75	50.00|82.02|66.66|69.84	50.25|80.64|75.68|69.69
MNIST	Accuracy ± std	**98.69 ± 0.14**	98.69 ± 0.08	98.61 ± 0.25	98.51 ± 0.41	98.64 ± 0.21
	Compression/layer	**41.25|85.31|73.31|66.22**	42.63|84.04|67.05|69.60	41.88|63.15|65.17|76.68	44.25|68.54|73.26|79.71	48.13|73.40|78.65|80.06
FMNIST	Accuracy ± std	83.21 ± 0.47	83.18 ± 0.96	83.65 ± 0.69	83.17 ± 0.66	**83.66 ± 0.56**
	Compression/layer	28.25|55.83|60.32|64.02	35.75|61.52|59.88|72.34	39.38|64.97|68.91|77.41	43.63|88.88|78.77|81.36	**48.50|77.48|81.29|80.23**

In Table 6, we explore the effect of increasing/decreasing the maximum/minimum pruning threshold value θ^+ and θ^- by increasing the value of θ (used in equation 5) up to 0.7 on the SG-based network performance and compression rate. Moreover, we apply the experiments using the same network architecture.

We can see from Table 6 that by increasing the value of θ, the compression rate is growing, and some layers are compressed more than 80%. Furthermore, we can see that the compression rate increases when going more profound in the network, which is also the case in STDP-based networks. In terms of the network accuracy, we can see a slight improvement in some cases (Face/Motor and FMNIST) and a small decrease (less than 2%) in others (CIFAR-10 and MNIST). Therefore, for the SG-based networks, we see a clear compression improvement by increasing the θ value. Finally, we can see that the best-recorded accuracy across the experiments does not use a specific θ value. The selection of this value may also depend on the used dataset and architecture.

In Table 7, we compare our work (STDP-based and SG-based) with existing works regarding accuracy and compression rate. Although in our work, we focus on providing a pruning technique that reduces the loss in performance and does not improve the state-of-the-art (SOTA) performance, we can see that the network's performance depends on the training mechanism and the network architecture. Our work reports accuracy close to SOTA with some datasets (MNIST and Face/Motor) and worst in others (FMNIST and CIFAR-10), which may be due to the small size of the network or the lack of hyperparameters tuning. Moreover, let's compare the two types of networks used in our work. We can see that SG-based networks do better in network performance and compression rate than STDP-based networks. Finally, We can see

Table 7. Accuracy and compression compared to existing works

		Training	Architecture	Accuracy ± std	Pruning technique	Compression
CIFAR-10	Our work	STDP + SVM	1 conv layer	53.71 ± 0.41	PP & DSWR	19.91
	Our work	Surrogate Gradient	2 conv-pool + 2 fc layer	57.82 ± 0.16	PP & DSWR	57.29
	(Deng et al., 2021)	Surrogate Gradient	7 conv + 2 fc layer	89.15	ADMM-based	50.00
	(Chen et al., 2021)	Surrogate Gradient	6 conv + 2 fc layer	92.54	Grad R	71.59
Face/Motor	(Nguyen et al., 2021)	STDP + SVM	3 conv-pool layer	95.70	Static threshold	92.83
	Our work	STDP + SVM	2 conv-pool + 1 fc layer	90.95 ± 0.37	PP & DSWR	46.76
	Our work	Surrogate Gradient	2 conv-pool + 2 fc layer	97.93 ± 0.83	PP & DSWR	53.38
	(Zhang et al., 2022)	Back-propagation	2 conv-pool + 3 fc layer	99.50	—	—
MNIST	Our work	STDP + SVM	2 conv-pool + 1 fc layer	96.88 ± 0.19	PP & DSWR	27.27
	Our work	Surrogate Gradient	2 conv-pool + 2 fc layer	98.69 ± 0.14	PP & DSWR	66.52
	(Chen et al., 2021)	Surrogate Gradient	2 fc layer	98.59	Grad R	74.29
	(Diehl et al., 2015)	ANN-SNN Conversion	2 conv-pool + 1 fc layer	99.14	—	—
FMNIST	Our work	STDP + SVM	2 conv-pool + 1 fc layer	83.61 ± 0.40	PP & DSWR	30.20
	Our work	Surrogate Gradient	2 conv-pool + 2 fc layer	83.66 ± 0.56	PP & DSWR	71.88
	(Ranjan et al., 2020)	Back-propagation	2 conv + 1 pool + 2 fc layer	89.00	—	—
	(Zhang et al., 2022)	Back-propagation	2 conv-pool + 3 fc layer	90.1	—	—

that different pruning techniques have been used in the existing works, and most of them report a highly compressed network.

3.3 Layer-based Progressive Compression

From the experimental results that we presented in Table 2, Table 3, Table 5, and Table 6 we can see that in the case of multi-layer SNN, we get a higher compression rate in the deeper layers compared to the first one. Moreover, in the previous experiments, we used the same α value (α = 0.05 for STDP-based networks, and $\alpha^+ = 0.005$, $\alpha^- = -0.005$ for SG-based networks), which represents the initial pruning threshold value across all layers. Therefore, we test in this section a layer-based progressive compression by studying the effect of having an increasing α when going more deep in the network on the performance and compression rate. In our experiments, We increase the α value (α = 0.05 for STDP-based networks, and $\alpha^+ = 0.005$, $\alpha^- = -0.005$ for SG-based networks) each time we go to the next layer in the network, and we test on the same multi-layer architectures used with MNIST, FMNIST, and Caltech face/motorbike. We can see the evolution of the pruning threshold on three different network layers in Figure 5. For the STDP-based networks (Figure 5 (A)), we can see that the pruning threshold value does not increase after crossing the θ value set to 0.7 for all layers, and the time required to cross the threshold rises when going deeper in the network. Moreover, the last possible pruning threshold value equals or exceeds θ. For the SG-based networks (Figure 5 (B)), we can see that for each layer, we have two thresholds, the maximum pruning threshold θ is different from one layer to another due to the weights range being different from one layer to another. Moreover, we can see that the first layer is the first to cross θ, while the last layer threshold is still increasing, which is the opposite of the STDP-based networks.

In Table 8, we run the same experiments using a fixed α and layer-based α for ten times, and we record the results in terms of compression per layer and network performance.

We can see in Table 8 that using a layer-based α allows a higher compression rate compared to a fixed α in both cases (STDP-based and SG-based). Moreover, the performance is maintained and slightly improved in the case of STDP-based networks. On the other hand, a slight loss in accuracy is recorded for the SG-based networks (around 2%), which can be due to the high rate of compression recorded in some

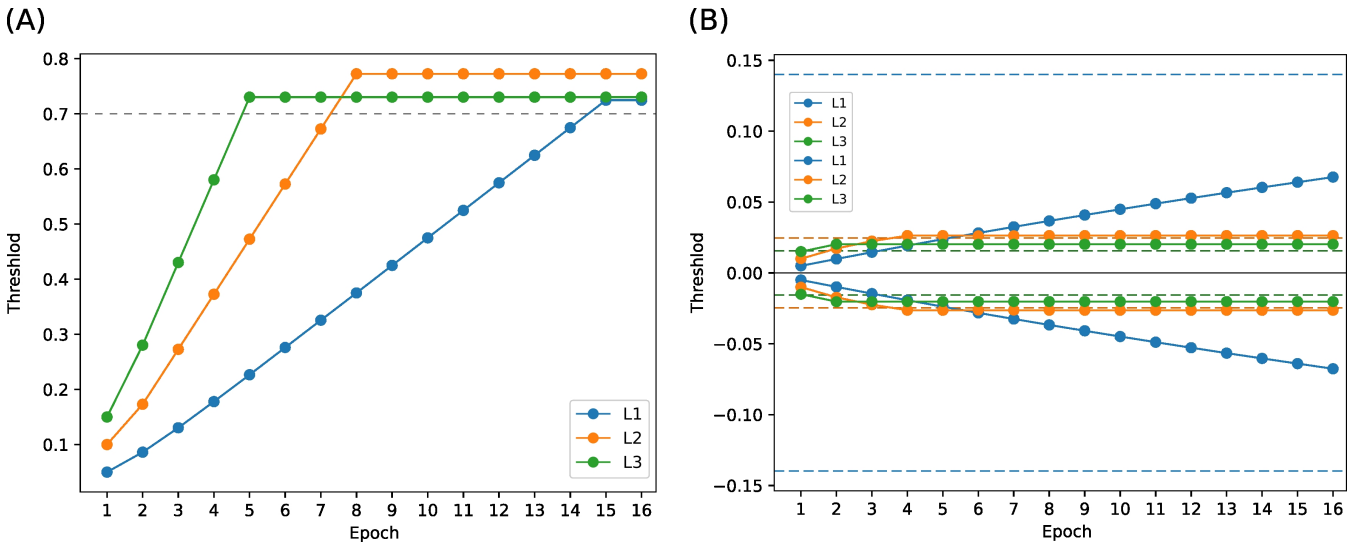

Figure 5. Layer-based pruning threshold activity using convolutional SNN for (A) STDP-based networks and (B) SG-based networks

Table 8. Layer-based compression compared to having a fixed α

			Accuracy ± std	Compression/layer
STDP-Based	Face/Motor	Fixed α	90.95 ± 0.37	42.87|49.85|47.56
		Layer-based α	91.56 ± 0.30	42.87|50.04|50.16
	MNIST	Fixed α	96.80 ± 0.34	43.00|27.29|13.16
		Layer-based α	97.26 ± 0.18	43.00|27.30|13.17
	FMNIST	Fixed α	83.61 ± 0.40	50.18|27.25|13.17
		Layer-based α	83.76 ± 0.24	50.25|27.24|13.16
SG-Based	Face/Motor	Fixed α	97.13 ± 1.34	50.25|80.64|75.68|69.69
		Layer-based α	94.82 ± 2.99	53.75|98.45|90.76|87.81
	MNIST	Fixed α	98.64 ± 0.21	48.13|73.40|78.65|80.06
		Layer-based α	97.99 ± 1.70	75.00|80.69|95.01|95.57
	FMNIST	Fixed α	83.66 ± 0.56	48.50|77.48|81.29|80.23
		Layer-based α	82.24 ± 1.64	50.25|82.03|93.29|91.05

internal layers. Finally, we can see for the STDP-based networks that the compression did not increase even with a layer-based α due to the issue we mentioned in Section 3.1.

3.4 Compressed Network On The SpiNNaker Board

To evaluate the effectiveness of the proposed approach in neuromorphic implementation, we transfer the learned weights of a baseline and compressed STDP-based network from csnn-simulator to SpiNNaker to observe the network activity in both cases. We use a network of two fully-connected layers of 50 and 128 neurons, which we train for 10 epochs, and We use the MNIST dataset for this experiment.

The trained weights are transferred to the PyNN model without additional adaptation (neuron model or other hyperparameters) compared to the original network used in csnn-simulator. Moreover, it is worth mentioning that for the sake of simplicity, the transfer learning to SpiNNaker concerns only the synapses

in this work. Therefore, using the same neuron type and looking for an optimized configuration and hyperparameters will be discussed in future work.

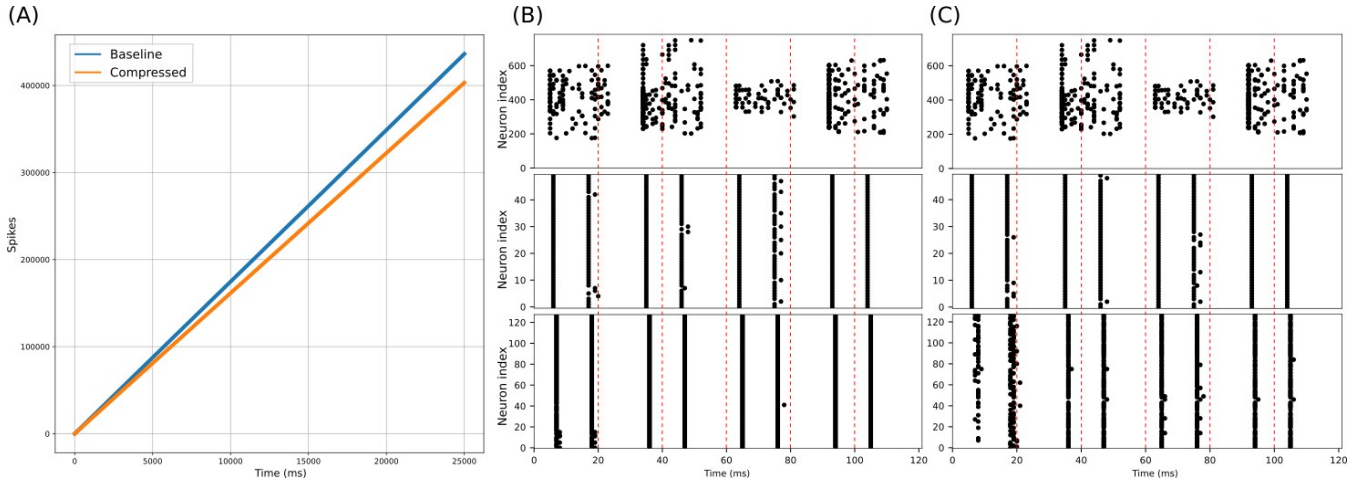

Figure 6. (A) spikes accumulated activity in the output layer. (B) and (C) represent spikes per layer for four inputs in the case of baseline and compressed STDP-based network, respectively

In Figure 6 (A), we compare the accumulated spikes for the baseline and the compressed model for 25 second. As a result, we can see how the gap between the two use cases starts growing when moving forward in the simulation. Moreover, in Figure 6 (B) and (C), we can check spikes activity during 120 ms (four digits) for the two layers of the network. Therefore, we see a difference in spikes activity in the two use cases, with a drop in the spikes activity of the output layer for the compressed network (especially the first input). However, in the first layer, the difference between the two use cases is not very visible. Hence, Figure 6 shows the effect of the compression approach on the network activity compared to the baseline when using the SpiNNaker board.

Table 9. Spikes activity and energy estimation on SpiNNaker

	# of Spikes (L1)	# of Spikes (L2)	Energy (J)
Baseline	$999,786$	$2,559,907$	$28.48 \times e^{-3}$
Compressed	$999,827$	$2,158,839$	$25.27 \times e^{-3}$

In Table 9, we compare the number of spikes per layer for the baseline and the compressed network using the MNIST test set (10k digits) and report the estimated energy in both cases. In terms of spikes activity, since spikes generation depends on many factors but essentially on synapses for propagation, we observe a drop in the number of spikes in second layer (almost 16%) when compressed. Moreover, regarding the energy consumption of SpiNNaker, based on literature (Painkras et al., 2013; Stromatias et al., 2014; Sugiarto et al., 2016; van Albada et al., 2018; Stromatias et al., 2013), a significant fraction of the total power for different stages of simulation is spent on the idle mode. Moreover, the reported energy per synaptic event for LIF neurons equals 8 nJ. Therefore, compressing saves approximately $3.2 \times e^{-3}$ Joule, which means 11.068 uW less power consumption (for simulation time = 290 s).

4 CONCLUSION

This paper presents the progressive compression for convolutional spiking neural networks, which we train using STDP+SVM or Surrogate gradient. The proposed approach, an extension of the PP & DSWR for shallow networks, is tested with complex architecture on a classification task with multiple datasets. We also test the resulting network on the SpiNNaker board by transferring the final weights. Using this approach, we got an average layer compression of more than 70% in some datasets when using SG-based networks, with some layers highly compressed than others (more than 80%). Moreover, we discuss the low compression rate recorded when using STDP-based networks due to the combination of the threshold adaptation mechanism and the progressive compression, which did not help the network maintain a reasonable classification and compression rate.

Furthermore, the layer-based approach discussed in this work provides extra compression (up to 98%) without a significant loss in the network performance (less than 3%). For some datasets, we record a tiny improvement in the network performance. Finally, the tests we conducted on the SpiNNaker board by analyzing the two use cases (baseline and compressed) show a noticeable decrease in the spikes activity when we apply the compression, which will allow the implementation of bigger models in a resource-constrained architecture.

Regarding compression in neural networks, we can use different techniques targeting synapses and other network components. Therefore, the work we presented, which concerns the synapses, can easily be combined with other methods (neuron compression, weight quantization, etc.) to improve the compression even more. Finally, as future works, a detailed parameters exploration for the different parameters in the model or the two formulas (α & β) can improve the compression, testing this approach on networks trained with other surrogate gradient methods, more complex datasets, and different tasks (other than image classification). For SpiNNaker, a more profound analysis of the compression effect and parameters exploration is needed when training onboard, with a clear report on the energy and the performance of the resulting network.

CONFLICT OF INTEREST STATEMENT

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

AUTHOR CONTRIBUTIONS

H.E. and M.F. contributed to formulating the study. H.E. conducted the experiments on csnn-simulator and Norse. M.F. implemented the networks on the SpiNNaker board. In addition, both contributed to writing the paper.

FUNDING

This work was partially funded by the European CHIST-ERA APROVIS3D project, and the Luxant-ANVI industrial chair (I-Site and Metropole Européene de Lille).

ACKNOWLEDGMENTS

This work was supported in part by IRCICA (Univ. Lille, CNRS, USR 3380 – IRCICA, F-59000 Lille, France) under the Bioinspired Project. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

REFERENCES

Basu, A., Deng, L., Frenkel, C., and Zhang, X. (2022). Spiking neural network integrated circuits: A review of trends and future directions. In *2022 IEEE Custom Integrated Circuits Conference (CICC)*. 1–8. doi:10.1109/CICC53496.2022.9772783

Bi, G.-q. and Poo, M.-m. (1998). Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type. *Journal of Neuroscience* 18, 10464–10472. doi:10.1523/JNEUROSCI.18-24-10464.1998

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., et al. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems* (Curran Associates, Inc.), vol. 33, 1877–1901

Burkitt, A. N. (2006). A review of the integrate-and-fire neuron model: I. homogeneous synaptic input. *Biological cybernetics* 95, 1–19. doi:10.1007/s00422-006-0068-6

Carreira-Perpinan, M. A. and Idelbayev, Y. (2018). "learning-compression" algorithms for neural net pruning. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8532–8541. doi:10.1109/CVPR.2018.00890

Chen, R., Ma, H., Xie, S., Guo, P., Li, P., and Wang, D. (2018). Fast and efficient deep sparse multi-strength spiking neural networks with dynamic pruning. In *2018 International Joint Conference on Neural Networks (IJCNN)*. 1–8. doi:10.1109/IJCNN.2018.8489339

Chen, Y., Yu, Z., Fang, W., Huang, T., and Tian, Y. (2021). Pruning of deep spiking neural networks through gradient rewiring. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21* (International Joint Conferences on Artificial Intelligence Organization), 1713–1721

Cho, S.-G., Beigné, E., and Zhang, Z. (2019). A 2048-neuron spiking neural network accelerator with neuro-inspired pruning and asynchronous network on chip in 40nm cmos. In *2019 IEEE Custom Integrated Circuits Conference (CICC)*. 1–4. doi:10.1109/CICC.2019.8780116

Cun, Y. L., Denker, J. S., and Solla, S. A. (1990). Optimal brain damage. In *Advances in neural information processing systems 2* (San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.). 598–605

Davies, M., Srinivasa, N., Lin, T., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro* 38, 82–99. doi:10.1109/MM.2018.112130359

Davison, A., Brüderle, D., Eppler, J., Kremkow, J., Muller, E., Pecevski, D., et al. (2009). Pynn: a common interface for neuronal network simulators. *Frontiers in Neuroinformatics* 2. doi:10.3389/neuro.11.011.2008

Deb, K. (2011). Multi-objective optimisation using evolutionary algorithms: An introduction. In *Multi-objective Evolutionary Optimisation for Product Design and Manufacturing* (Springer London). 3–34. doi:10.1007/978-0-85729-652-8_1

Deng, L., Wu, Y., Hu, Y., Liang, L., Li, G., Hu, X., et al. (2021). Comprehensive snn compression using admm optimization and activity regularization. *IEEE Transactions on Neural Networks and Learning Systems* PP, 1–15. doi:10.1109/TNNLS.2021.3109064

Diehl, P. and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience* 9. doi:10.3389/fncom.2015.00099

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., and Pfeiffer, M. (2015). Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks (IJCNN)*. 1–8. doi:10.1109/IJCNN.2015.7280696

Elbez, H., Benhaoua, M. K., Devienne, P., and Boulet, P. (2022). Progressive compression and weight reinforcement for spiking neural networks. *Concurrency and Computation: Practice and Experience* 34. doi:10.1002/cpe.6891

Faghihi, F., Alashwal, H., and Moustafa, A. A. (2022). A synaptic pruning-based spiking neural network for hand-written digits classification. *Frontiers in Artificial Intelligence* 5. doi:10.3389/frai.2022.680165

Falez, P. (2019). *Improving Spiking Neural Networks Trained with Spike Timing Dependent Plasticity for Image Recognition*. Theses, Université de Lille

Falez, P., Tirilly, P., Marius Bilasco, I., Devienne, P., and Boulet, P. (2019). Multi-layered spiking neural network with target timestamp threshold adaptation and stdp. In *2019 International Joint Conference on Neural Networks (IJCNN)*. 1–8. doi:10.1109/IJCNN.2019.8852346

Furber, S. B., Galluppi, F., Temple, S., and Plana, L. A. (2014). The SpiNNaker Project. *Proceedings of the IEEE* 102, 652–665. doi:10.1109/JPROC.2014.2304638

Furber, S. B., Lester, D. R., Plana, L. A., Garside, J. D., Painkras, E., Temple, S., et al. (2013). Overview of the SpiNNaker System Architecture. *IEEE Transactions on Computers* 62, 2454–2467. doi:10.1109/TC.2012.142

Hassibi, B., Stork, D. G., and Wolff, G. (1994). Optimal Brain Surgeon: Extensions and performance comparisons. In *Advances in Neural Information Processing Systems 6* (Morgan-Kaufmann). 263–270

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. doi:10.1109/CVPR.2016.90

He, Y., Liu, P., Wang, Z., Hu, Z., and Yang, Y. (2019). Filter pruning via geometric median for deep convolutional neural networks acceleration. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE), 4335–4344

Huang, Q., Zhou, K., You, S., and Neumann, U. (2018). Learning to prune filters in convolutional neural networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 709–718. doi:10.1109/WACV.2018.00083

Huttenlocher, P. R. (1979). Synaptic density in human frontal cortex - developmental changes and effects of aging. *Brain Research* 163, 195–205. doi:10.1016/0006-8993(79)90349-4

Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., and Masquelier, T. (2018). STDP-based spiking deep convolutional neural networks for object recognition. *Neural Networks* 99, 56–67. doi:10.1016/j.neunet.2017.12.005

Krizhevsky, A. and Hinton, G. (2009). *Learning multiple layers of features from tiny images*. Tech. rep., University of Toronto, Toronto, Ontario

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 2278–2324. doi:10.1109/5.726791

Lee, J. H., Delbruck, T., and Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience* 10. doi:10.3389/fnins.2016.00508

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. (2016). Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710* doi:10.48550/ARXIV.1608.08710

Liu, Z., Xu, J., Peng, X., and Xiong, R. (2018). Frequency-domain dynamic pruning for convolutional neural networks. In *Advances in Neural Information Processing Systems* (Curran Associates, Inc.), vol. 31, 1051–1061

Merolla, P., Arthur, J., Akopyan, F., Imam, N., Manohar, R., and Modha, D. S. (2011). A digital neurosynaptic core using embedded crossbar memory with 45pj per spike in 45nm. In *2011 IEEE Custom Integrated Circuits Conference (CICC)*. 1–4. doi:10.1109/CICC.2011.6055294

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine* 36, 51–63. doi:10.1109/MSP.2019.2931595

Nguyen, T. N. N., Veeravalli, B., and Fong, X. (2021). Connection pruning for deep spiking neural networks with on-chip learning. In *International Conference on Neuromorphic Systems 2021* (New York, NY, USA: Association for Computing Machinery), ICONS 2021. doi:10.1145/3477145.3477157

Painkras, E., Plana, L. A., Garside, J., Temple, S., Galluppi, F., Patterson, C., et al. (2013). Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation. *IEEE Journal of Solid-State Circuits* 48, 1943–1953. doi:10.1109/JSSC.2013.2259038

[Dataset] Pehle, C. and Pedersen, J. E. (2021). Norse - A deep learning library for spiking neural networks. doi:10.5281/zenodo.4422025. Documentation: https://norse.ai/docs/

Ranjan, J. A. K., Sigamani, T., and Barnabas, J. (2020). A novel and efficient classifier using spiking neural network. *The Journal of Supercomputing* 76, 6545–6560. doi:10.1007/s11227-019-02881-y

Rathi, N., Panda, P., and Roy, K. (2019). STDP-Based Pruning of Connections and Weight Quantization in Spiking Neural Networks for Energy-Efficient Recognition. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 668–677. doi:10.1109/TCAD.2018.2819366

Rhodes, O., Bogdan, P. A., Brenninkmeijer, C., Davidson, S., Fellows, D., Gait, A., et al. (2018). spynnaker: A software package for running pynn simulations on spinnaker. *Frontiers in Neuroscience* 12. doi:10.3389/fnins.2018.00816

Rullen, R. V. and Thorpe, S. J. (2001). Rate coding versus temporal order coding: What the retinal ganglion cells tell the visual cortex. *Neural Computation* 13, 1255–1283. doi:10.1162/08997660152002852

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (Los Alamitos, CA, USA: IEEE Computer Society), 4510–4520. doi:10.1109/CVPR.2018.00474

Sanes, J. R. and Lichtman, J. W. (1999). Development of the vertebrate neuromuscular junction. *Annual Review of Neuroscience* 22, 389–442. doi:10.1146/annurev.neuro.22.1.389

Saunders, D. J., Patel, D., Hazan, H., Siegelmann, H. T., and Kozma, R. (2019). Locally connected spiking neural networks for unsupervised feature learning. *Neural Networks* 119, 332–340. doi:10.1016/j.neunet.2019.08.016

Sharp, T., Galluppi, F., Rast, A., and Furber, S. (2012). Power-efficient simulation of detailed cortical microcircuits on spinnaker. *Journal of neuroscience methods* 210, 110–118. doi:10.1016/j.jneumeth.2012.03.001

Shi, Y., Nguyen, L., Oh, S., Liu, X., and Kuzum, D. (2019). A Soft-Pruning Method Applied During Training of Spiking Neural Networks for In-memory Computing Applications. *Frontiers in Neuroscience* 13. doi:10.3389/fnins.2019.00405

Stromatias, E., Galluppi, F., Patterson, C., and Furber, S. (2013). Power analysis of large-scale, real-time neural networks on spinnaker. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*. 1–8. doi:10.1109/IJCNN.2013.6706927

Stromatias, E., Patterson, C., and Furber, S. (2014). Optimising the overall power usage on the spinnaker neuromimetic platform. In *2014 International Joint Conference on Neural Networks (IJCNN)*. 4280–4287. doi:10.1109/IJCNN.2014.6889837

Strukov, D. B., Snider, G. S., Stewart, D. R., and Williams, R. S. (2008). The missing memristor found. *Nature* 453, 80–83. doi:10.1038/nature06932

Sugiarto, I., Liu, G., Davidson, S., Plana, L. A., and Furber, S. B. (2016). High performance computing on spinnaker neuromorphic platform: A case study for energy efficient image processing. In *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*. 1–8. doi:10.1109/PCCC.2016.7820645

van Albada, S. J., Rowley, A. G., Senk, J., Hopkins, M., Schmidt, M., Stokes, A. B., et al. (2018). Performance comparison of the digital neuromorphic hardware spinnaker and the neural network simulation software nest for a full-scale cortical microcircuit model. *Frontiers in Neuroscience* 12. doi:10.3389/fnins.2018.00291

[Dataset] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. doi:10.48550/ARXIV.1708.07747

Yu, R., Li, A., Chen, C., Lai, J., Morariu, V. I., Han, X., et al. (2018). NISP: pruning networks using neuron importance score propagation. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. IEEE (IEEE Computer Society), 9194–9203

Zenke, F. and Ganguli, S. (2018). SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks. *Neural Computation* 30, 1514–1541. doi:10.1162/neco_a_01086

Zhang, M., Wang, J., Wu, J., Belatreche, A., Amornpaisannon, B., Zhang, Z., et al. (2022). Rectified linear postsynaptic potential function for backpropagation in deep spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems* 33, 1947–1958. doi:10.1109/TNNLS.2021.3110991