# 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 ML-related 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 with unsupervised STDP. 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) while preserving the same network performance or slightly improving it, as shown by experimental results using MNIST, FMNIST, Caltech face/motorbike, STL-10, and CIFAR-10 datasets.

**Keywords: Spiking Neural Network, Neuromorphic Computing, Compression, STDP, 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 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 architecture, 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, ResNet152, and GPT-3. 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 in 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 computation resources. Pruning is a widely used technique in machine learning, and the activity of the human brain in the early stages is the source of inspiration behind this technique (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., 2017), weights pruning (Carreira-Perpinan and Idelbayev, 2018; Liu et al., 2018), and neuron-based pruning (Yu et al., 2018).

In the case of SNNs, we can use pruning to achieve the same result. 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 the training and can maintain 90% classification accuracy on MNIST with up to 75% pruning. Cho et al. (Cho et al., 2019) applied a pruning on a CMOS SNN chip. It is used based on the distance between a group of neurons by removing the connection between them, 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. Furthermore, based on the STDP learning rule (Bi and Poo, 1998), we deactivate non-critical synapses during the application of this technique. In the work of Chen et al. (Chen et al., 2018), the authors use 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 they pruned the network only once when the network learning phase was done. 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 Learning (Neftci et al., 2019) or local learning rule (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 $\mu$, 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 progressive pruning on a Convolutional SNN to reduce synaptic connections, we get highly compressed layers (70% compression rate at least), while the compression rate increases when going more profound in the network. Moreover, the network performance may increase compared to the baseline in the best-case scenario or record less than 1% accuracy loss in the worst-case scenario. We tested the experiments on MNIST, FMNIST, Caltech face/motorbike, STL-10, and CIFAR-10 datasets and studied the effect on our approach when used with SpiNNaker.

## 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).

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

$\alpha$ 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 in two scenarios: using the same configuration as in shallow networks or modifying the equation to fit a multilayer network.

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).

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

$\beta$ 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, we keep the two constants $\alpha$ and $\beta$ the same as in (Elbez et al., 2022) ($\alpha$ = 0.05 and $\beta$ = 0.1). Those values were fixed using Pareto front multiobjective optimization 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. However, for CSNN, we usually train for multiple epochs, and following the same method will cause the 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 threshold value. If we reach this value, we don't 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 Convolutional SNN.

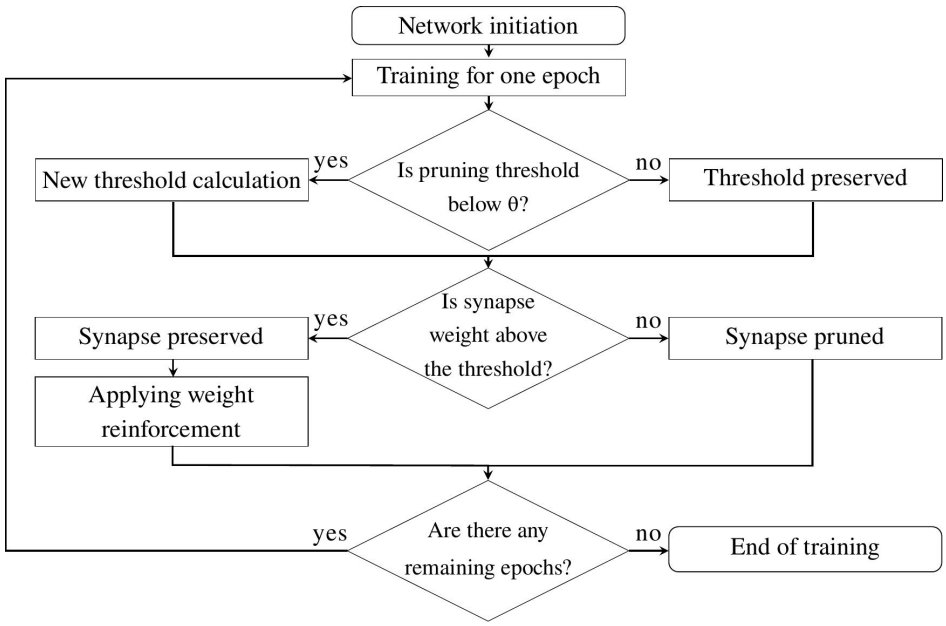


**Figure 1.** Compression flowchart for Convolutional SNN

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

$\theta$ is the max possible threshold. Since the weights in our network are between 0 and 1, we set $\theta$ = 0.3 in our work. Moreover, the reinforcement part stays the same, and we apply it after each prune operation.

In Figure 2, we can observe how the threshold value changes during training (10 epochs) using our approach of one layer during the learning phase. Moreover, we can follow the evolution of the compression rate also. After a couple of epochs, we can see that the threshold value is stable (around 0.28) due to $\theta$ = 0.3. Nevertheless, the compression keeps increasing due to the learning and the reinforcement applied in the network.

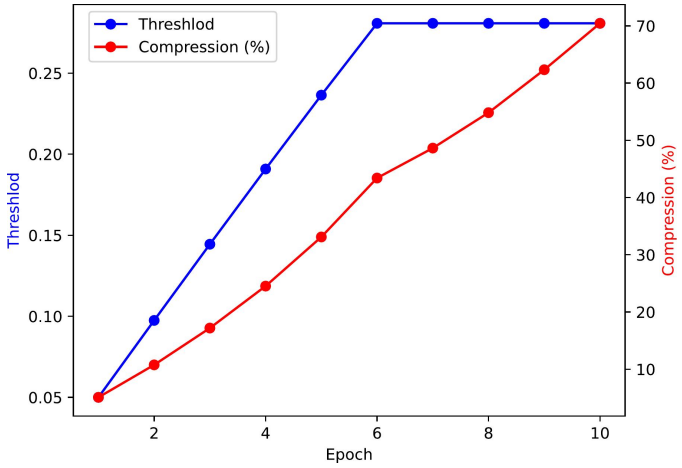


**Figure 2.** Compression rate and threshold evolution during training

## 2.2 Network Topology

In our experiments, we use the same network 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. 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. All the experiments are simulated using csnn-simulator (Falez, 2019), a C++-based open-source simulator [1].

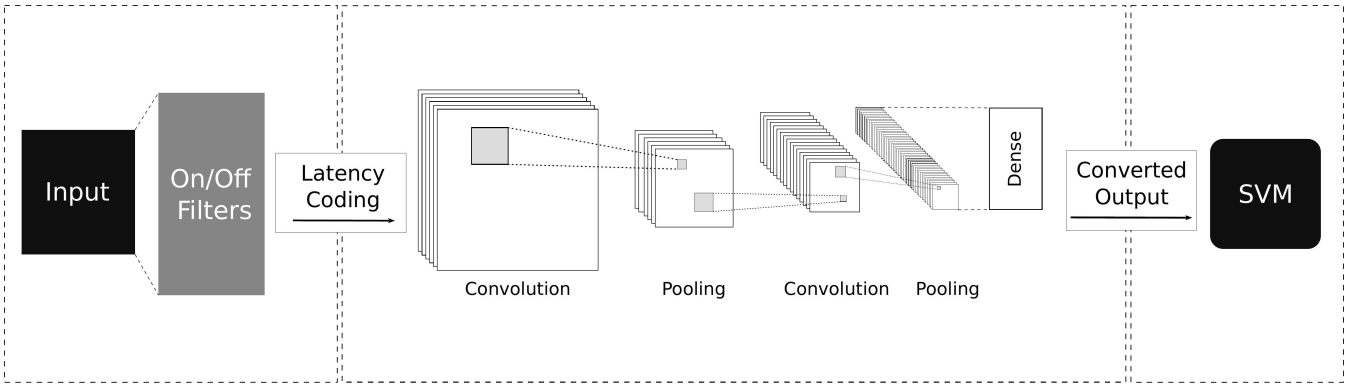


**Figure 3.** The network topology

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 (4) (Falez, 2019).

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

$$C_m \frac{\partial v(t)}{\partial t} = \sum_{i \epsilon \varepsilon} v_i f_s(t - t_i), \qquad f_s(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{4}$$
$$v(t) = v_r \quad when \quad v(t) \geq v_{th}(t)$$

$C_m$ represents the membrane capacitance, and $v_i$ is the spike voltage of the $i$-th spike. Also, $\varepsilon$ 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 (5).

$$\Delta_w = \begin{cases} \eta_w e^{-\frac{t_{\text{pre}} - t_{\text{post}}}{\tau_{\text{STDP}}}}, & \text{if } t_{\text{pre}} \leq t_{\text{post}} \\ -\eta_w e^{-\frac{t_{\text{pre}} - t_{\text{post}}}{\tau_{\text{STDP}}}}, & \text{otherwise} \end{cases} \tag{5}$$

$\tau_{\text{STDP}}$ is the time constant for the STDP learning window, and $\eta_w$ is the learning rate. $t_{\text{pre}}$ and $t_{\text{post}}$ represent the spiking time of pre-synaptic and post-synaptic neuron, respectively. We can see from the equation that the update on the synaptic weight $\Delta_w$ can be either positive or negative, depending on which spike came first. In our experiments $w\epsilon$ [0,1].

For the multilayer 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 multilayer SNN we will use the same 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 threshold adaptation is presented by equations (6) and (7).

$$\begin{aligned} \Delta^1_{\text{th}} &= -\eta_{\text{th}}(\text{t} - \text{t}_{\text{obj}}) \\ \Delta^2_{\text{th}} &= \begin{cases} \eta_{\text{th}}, & \text{if } t_i = \min(t_0, ..., t_N) \\ -\frac{\eta_{\text{th}}}{l_d(n)}, & \text{otherwise} \end{cases} \end{aligned} \tag{6}$$

$$v_{th}(t) = max(\text{th}_{\text{min}}, v_{th}(t-1) + \Delta^1_{\text{th}} + \Delta^2_{\text{th}}) \tag{7}$$

$\eta_{th}$ 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, $th_{\text{min}}$ is the minimum possible threshold value.

In Table 1, we can see the different hyperparameters used in the experiments for the DoG filter, STDP, and threshold adaptation. Some parameters are kept the same when using different datasets, whereas others differ from one dataset to another. Moreover, we use default parameters for the SVM part of the network, which delivers good performance.

**Table 1.** The hyperparameters used in the experiments

| Difference-of-Gaussian | STDP | Threshold Adaptation |
|---|---|---|
| $DoG_{in} = 1.0$ | $\eta_w = 0.1$ | $\eta_{th} = 1.0$, $th_{min} = 1.0$ |
| $DoG_{out} = 4.0$ | $\tau_{STDP} = 0.1$ | $t_{obj}^{CIFAR\text{-}10} = 0.95$ |
| $DoG_{size} = 7.0$ | | $t_{obj}^{MNIST} = t_{obj}^{FMNIST} = 0.75$ |
| | | $t_{obj}^{Face/Motor} = 0.80$, $t_{obj}^{STL\text{-}10} = 0.85$ |

### 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. Moreover, 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. Finally, we have STL-10 (Coates et al., 2011), which is inspired by CIFAR-10 and consists of colored 96x96 pixel images. STL-10 has ten classes and contains 5000 train and 8000 test images.

### 2.4 SpiNNaker Board

Modeling large neural networks on Von Neumann architecture requires computing resources and a lot of 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 (Figure 4) 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., 2012), 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

In this section, we describe our experiments on the image classification task and present the obtained results. We run each simulation ten times, then report the average results. Moreover, training is done layerwise by

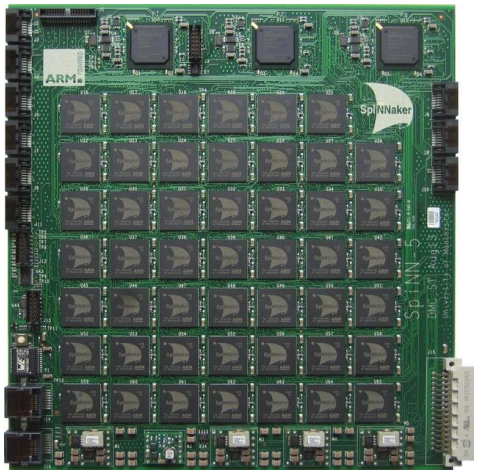


**Figure 4.** A SpiNNaker board with 48 chips (SpiNN-5)

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 STL-10 and 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 2.

**Table 2.** The used architectures in the experiments

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

In Table 3, we can see the effect of compressing the network using our approach on different datasets. We observe, in particular, the network classification rate (Acc(%) $\pm$ std), the compression rate on each layer (Comp(%)/layer), the duration of the simulation (Time), number of spikes per layer (Spikes/layer), and number of synaptic updates per layer (Synaptic updates/layer). Finally, the comparison is made between the compressed network and the baseline (the same model without compression).

By analyzing Table 3, we can see that in terms of the network performance, we don't have any significant loss when compressing, and the network can maintain the same performance on different datasets. Moreover, for the compression rate, we can observe that compression is high, starting from 72% and going up to 92% for single-layer networks (CIFAR-10 and STL-10). For multilayer networks, we can see the compression rate for each layer. However, it is interesting that despite using different datasets, the compression rate is lower at the first layer and highest at the second layer. At the same time, the dense layer is somewhere in

**Table 3.** Accuracy, compression, and layers activity (spikes & synaptic updates)

| | | Acc(%) ± std | Comp(%)/layer | Time | Spikes/layer | Synaptic updates/layer |
|---|---|---|---|---|---|---|
| CIFAR-10 | baseline | 54.63 ± 0.37 | — | 2:21:34 | 5x10$^6$ | 7.5x10$^8$ |
| | this work | 54.60 ± 0.39 | 72.69 | 2:21:31 | 5x10$^6$ | 7.5x10$^8$ |
| STL-10 | baseline | 58.61 ± 0.24 | — | 1:59:15 | 2.5x10$^5$ | 3.75x10$^7$ |
| | this work | 58.21 ± 0.36 | 92.26 | 2:05:51 | 2.5x10$^5$ | 3.75x10$^7$ |
| Face/Motor | baseline | 98.91 ± 0.57 | — | 0:41:51 | 19981\|20000\|20000 | 999075\|1.84x10$^8$\|3.2x10$^7$ |
| | this work | 98.51 ± 0.29 | 80.06\|98.72\|98.05 | 0:42:18 | 19989\|20000\|20000 | 999475\|1.84x10$^8$\|3.2x10$^7$ |
| MNIST | baseline | 98.12 ± 0.13 | — | 2:29:56 | 1995723\|2977102\|3x10$^6$ | 99786150\|2.38x10$^9$\|6.14x10$^9$ |
| | this work | 98.12 ± 0.10 | 72.81\|94.39\|87.82 | 2:48:16 | 1992944\|2974986\|3x10$^6$ | 99647240\|2.37x10$^9$\|6.14x10$^9$ |
| FMNIST | baseline | 84.32 ± 0.27 | — | 3:50:05 | 2804511\|2999767\|3x10$^6$ | 1.4x10$^8$\|2.39x10$^9$\|6.14x10$^9$ |
| | this work | 84.56 ± 0.27 | 90.56\|95.53\|91.64 | 3:52:29 | 2805825\|2999786\|3x10$^6$ | 1.4x10$^8$\|2.39x10$^9$\|6.14x10$^9$ |

between. For the simulation time, we can see a slight increase when we use compression due to applying it during the training and the fact that the SVM training part is time-consuming and included in the reported times. Moreover, SVM is not concerned with compression in this work. Finally, for the layer activity (spikes & synaptic updates), we don't see a considerable decrease in both activities when applying the compression due to the threshold adaptation used in the network, and by presenving the network activity, the network can maintain most of its performance.

**Table 4.** Max threshold $\theta$ effect on the accuracy and compression

| | | Max threshold $\theta$ | | | | |
|---|---|---|---|---|---|---|
| | | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 |
| CIFAR-10 | Acc(%) ± std | 54.31 ± 0.23 | 54.86 ± 0.58 | 54.93 ± 0.27 | 54.43 ± 0.41 | **54.96 ± 0.39** |
| | Comp(%)/layer | 72.38 | 73.00 | 72.83 | 73.39 | **74.23** |
| STL-10 | Acc(%) ± std | 58.29 ± 0.29 | 58.45 ± 0.36 | **58.47 ± 0.42** | 58.38 ± 0.40 | 58.40 ± 0.36 |
| | Comp(%)/layer | 92.30 | 92.27 | **92.70** | 92.83 | 92.66 |
| Face/Motor | Acc(%) ± std | 98.53 ± 0.46 | 98.23 ± 0.37 | **98.86 ± 0.28** | 98.53 ± 0.62 | 98.83 ± 0.34 |
| | Comp(%)/layer | 80.12\|98.06\|97.11 | 80.75\|98.25\|97.69 | **80.93\|98.39\|98.25** | 81.37\|98.42\|98.54 | 80.68\|98.63\|98.38 |
| MNIST | Acc(%) ± std | 98.16 ± 0.11 | 98.17 ± 0.08 | 98.09 ± 0.13 | **98.19 ± 0.11** | 98.16 ± 0.13 |
| | Comp(%)/layer | 71.56\|95.78\|86.43 | 72.87\|95.07\|87.32 | 70.06\|95.84\|88.00 | **74.50\|96.16\|87.62** | 72.93\|95.91\|87.93 |
| FMNIST | Acc(%) ± std | 84.59 ± 0.26 | 84.58 ± 0.24 | 84.49 ± 0.28 | 84.47 ± 0.29 | **84.66 ± 0.33** |
| | Comp(%)/layer | 90.25\|96.07\|91.31 | 90.68\|92.78\|92.52 | 91.25\|96.18\|92.96 | 89.12\|95.91\|92.83 | **89.18\|96.07\|92.81** |

In Table 3, we set the max threshold $\theta$ to 0.3 for the experiments. The selection of $\theta$ can impact the network compression since it defines the highest possible threshold value, and if we set it too high ($\theta$ = 1), the compression will destroy the network. Moreover, the value of $\theta$ may vary from one application to another and from one dataset to another. In Table 4, we explore the effect of increasing $\theta$ 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 4, we can see that by increasing the maximum threshold value $\theta$, the compression rate is also rising with a slight improvement in the network performance, which is the case with single convolutional layer networks using CIFAR-10 and STL-10. For multilayer networks, we see that the overall compression rate of the network increases when we increase the value of $\theta$. However, it does not always imply an increase in the compression rate per layer since some layers are less compressed when $\theta$ increases. Regarding the network performance, we can see that the best-recorded accuracy across the experiments does not always

use a specific $\theta$ value and changes from one dataset to another. Moreover, the difference in performance between all $\theta$ values is less than 1%.

### 3.1 Layer-based Progressive Compression

From the experimental results that we presented in Table 3 and Table 4, we can see that in the case of multilayer SNN, we always get a higher compression rate in the second and third layer compared to the first one. Moreover, in the previous experiments, we used the same $\alpha$ value ($\alpha$ = 0.05 in this work), which represents the initial threshold value across all layers. Therefore, we test in this section a layer-based progressive compression by studying the effect of having an increasing $\alpha$ when going more deep in the network on the performance and compression rate. In our experiments, We double the $\alpha$ value (starting from 0.05) when we go to the next layer in the network, and we test on the same multilayer architectures used with MNIST, FMNIST, and Caltech face/motorbike. We can see the evolution of $\alpha$ on three different network layers in Figure 5. The threshold value is not increasing after crossing the $\alpha$ value set to 0.7 in this experiment, and the last possible threshold value is equal to or greater than $\alpha$. Moreover, the deeper we go into the network, the fewer epochs are needed to cross the maximum threshold value.

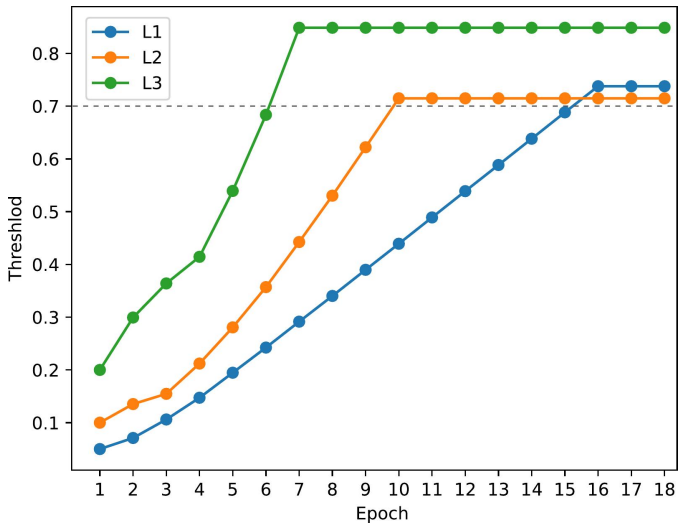


**Figure 5.** Layer-based threshold activity using convolutional SNN ($\theta$ = 0.7)

In Table 5, we run the same experiments using a fixed $\alpha$ and layer-based $\alpha$ for ten times, and we record the results in terms of compression per layer (Comp(%)/layer) and network performance (Acc(%) $\pm$ std).

**Table 5.** Layer-based compression compared to having a fixed $\alpha$

| | | Acc(%) $\pm$ std | Comp(%)/layer |
|---|---|---|---|
| Face/Motor | Fixed $\alpha$ | 98.83 $\pm$ 0.34 | 80.68\|98.63\|98.38 |
| | Layer-based $\alpha$ | 98.93 $\pm$ 0.53 | 81.12\|98.99\|99.04 |
| MNIST | Fixed $\alpha$ | 98.16 $\pm$ 0.13 | 72.93\|95.91\|87.93 |
| | Layer-based $\alpha$ | 98.17 $\pm$ 0.11 | 72.93\|96.91\|93.52 |
| FMNIST | Fixed $\alpha$ | 84.66 $\pm$ 0.33 | 89.18\|96.07\|92.81 |
| | Layer-based $\alpha$ | 84.46 $\pm$ 0.25 | 89.75\|96.75\|95.33 |

We can see in Table 5 that using a layer-based $\alpha$ allows a higher compression rate compared to a fixed $\alpha$. Moreover, the network performance is maintained, and the SVM can achieve almost the same classification rate using information from highly compressed layers.

## 3.2 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 network from the CSNN simulator to SpiNNaker in order to observe the network activity in both cases. We use a network of two fully-connected layers of 400 and 1600 neurons, which we train for 25 epochs. Moreover, the compression recorded in the compressed network is %74. 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 the 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.

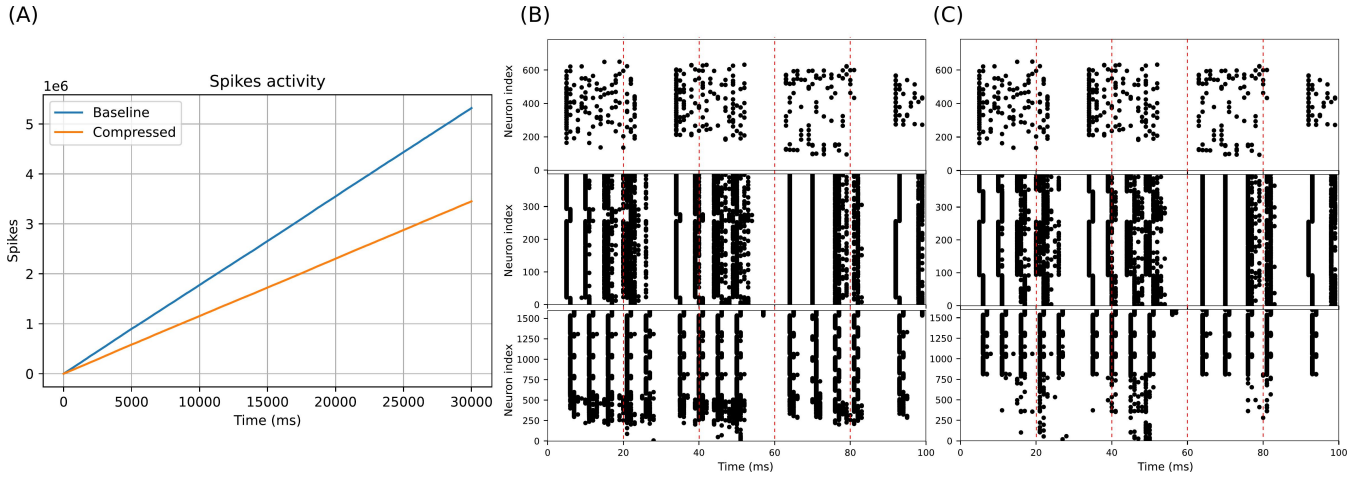


**Figure 6.** (A) spikes accumulated activity in the output layer. (B) and (C) represent spikes per layer for four inputs in the case of baseline and compressed network, respectively

In Figure 6 (A), we compare the accumulated spikes for the baseline and the compressed model using 10k inputs (30ms per input). We can see how the gap between the two use cases starts growing when moving forward in the simulation up to $2x10^6$ after 10k inputs. Moreover, in Figure 6 (B) and (C), we can check spikes activity during 100ms (almost four digits) for the two layers of the network. Therefore, we see a difference in spikes activity in the two use cases, with a noticeable drop in the spikes activity of the output layer for the compressed network. Hence, Figure 6 shows the effect of the compression approach on the network activity compared to the baseline when using the SpiNNaker board.

**Table 6.** Spikes activity and energy estimation on SpiNNaker

| | # of Spikes (L1) | # of Spikes (L2) | Energy (J) |
|---|---|---|---|
| Baseline | 15,257,058 | 51,217,781 | 0.5318 |
| Compressed | 15,384,711 | 33,531,934 | 0.3913 |

In Table 6, 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 considerable drop in the number of output spikes (almost %35) 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 8nJ. Therefore, compressing saves approximately 0.14 Joule, which means 483uW less power consumption (for simulation time = 290s).

## 4 CONCLUSION

This paper presents the progressive compression for convolutional spiking neural networks, which we train using STDP, and we use an SVM for classification. 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 80%, with some layers highly compressed than others (more than 90%). Moreover, the layer-based approach discussed in this work provides extra compression without a significant loss in the network performance (less than 1%). 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.

In terms of 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 ($\alpha$ & $\beta$) can improve the compression. Moreover, testing this approach on other scenarios like networks trained with global learning rules such as Surrogate Gradient Learning, more complex datasets, and different tasks. 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 the CSNN simulator, and 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.

## 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)* (IEEE), 1–8

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

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

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* (IEEE), 8532–8541

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)* (IJCNN), 1–8

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*, ed. Z.-H. Zhou (International Joint Conferences on Artificial Intelligence Organization), 1713–1721

Cho, S., 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)* (IEEE), 1–4

Coates, A., Ng, A., and Lee, H. (2011). An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, eds. G. Gordon, D. Dunson, and M. Dudík (Fort Lauderdale, FL, USA: PMLR), vol. 15 of *Proceedings of Machine Learning Research*, 215–223

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. P., Brüderle, D., Eppler, J. M., Kremkow, J., Muller, E., Pecevski, D., et al. (2009). Pynn: a common interface for neuronal network simulators. *Frontiers in neuroinformatics* 2, 11

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

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)* (IEEE), 1–8. ISSN: 2161-4393

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. (2012). Overview of the spinnaker system architecture. *IEEE transactions on computers* 62, 2454–2467

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

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)* (IEEE), 709–718

Huttenlocher, P. R. (1979). Synaptic density in human frontal cortex - developmental changes and effects of aging. *Brain Research* 163, 195–205

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. 0, 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

Lee, J. H., Delbruck, T., and Pfeiffer, M. (2016). Training Deep Spiking Neural Networks Using Backpropagation. *Frontiers in Neuroscience* 10

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. (2017). Pruning filters for efficient convnets. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net (OpenReview.net)

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 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, eds. S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (NeurIPS 2018), 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)* (IEEE), 1–4. ISSN: 0886-5930

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

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* (ACM). 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

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

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, 816

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

Sanes, J. R. and Lichtman, J. W. (1999). Development of the vertebrate neuromuscular junction. *Annual Review of Neuroscience* 22, 389–442

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

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

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

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)* (IEEE), 1–8

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)* (IEEE), 4280–4287

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

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)* (IEEE), 1–8

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, 291

[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