Spiking Approximations of the MaxPooling Operation in Deep SNNs

Ramashish Gaurav  
Systems Design Engineering  
University of Waterloo, Canada  
Email: rgaurav@uwaterloo.ca

Bryan Tripp  
Systems Design Engineering  
University of Waterloo, Canada  
Email: bptripp@uwaterloo.ca

Apurva Narayan  
Department of Computer Science  
University of British Columbia, Canada  
Email: apurva.narayan@ubc.ca

Abstract—Spiking Neural Networks (SNNs) are an emerging domain of biologically inspired neural networks that have shown promise for low-power AI. A number of methods exist for building deep SNNs, with Artificial Neural Network (ANN)-to-SNN conversion being highly successful. MaxPooling layers in Convolutional Neural Networks (CNNs) are an integral component to downsample the intermediate feature maps and introduce translational invariance, but the absence of their hardware-friendly spiking equivalents limits such CNNs’ conversion to deep SNNs. In this paper, we present two hardware-friendly methods to implement MaxPooling in deep SNNs, thus facilitating easy conversion of CNNs with MaxPooling layers to SNNs. In a first, we also execute SNNs with spiking-MaxPooling layers on Intel’s Loihi neuromorphic hardware (with MNIST, FMNIST, & CIFAR10 dataset); thus, showing the feasibility of our approach.

I. INTRODUCTION

Artificial Neural Networks (ANNs) have established themselves as the de-facto tool for a variety of Artificial Intelligence (AI) tasks. And the flagship performance of CNNs for image recognition/classification has remained unparalleled so far. However, their limitations in the aspects of energy consumption, robustness against noisy inputs, etc. has attracted interest in developing their spiking counterparts. Spiking Neural Networks (SNNs) offer a promise of low power AI and have shown to be more robust against noisy inputs [1], perturbations to the weights [2], and adversarial attacks [3], [4]. Out of a number of ways to build SNNs [5], [6], [7], the ANN-to-SNN conversion method has been highly effective for building deep SNNs. In this method, one first trains an ANN with standard rate neurons (e.g. ReLU) and then replaces those rate neurons with spiking neurons, along with the other required modifications of weights [8], etc. For our work, we consider this ANN-to-SNN conversion paradigm to build deep SNNs.

MaxPooling in CNNs is a common method to downsample the intermediate feature maps obtained from the Convolutional layers. One can also use AveragePooling or Strided Convolution to downsample the feature maps; however, the choice of the pooling method is contextual [9], and MaxPooling is generally found to give better performance. That said, [9] also found that “depending on the data and features, either max or average pooling may perform best”. Several architectures e.g. ResNet-50 (-152) [10], Xception [11], EfficientNet [12] which form the backbone of different methods to achieve SoTA results on the ImageNet, use a mix of Max and AveragePooling layers (or GlobalMax/Average Pooling layers).

However, the conversion of CNNs with MaxPooling layers to SNNs is a convoluted process. MaxPooling in SNNs has been a long-standing problem; only a handful of approaches exist for the same. [13] present three approaches for MaxPooling in SNNs; where they design a pooling gate to monitor the spiking neurons’ activities (in a pooling window) and dynamically connect one of the neurons to the output neuron based on their criteria for the maximally firing neuron. Such a gating mechanism is leveraged by [14] too, where they employ a finite impulse response filter to control the gating function and do MaxPooling in SNNs; [15] too use the same. Few other works [16], [17], [18], [19], [20] leverage a Time-To-First-Spike based temporal Winner Take All (WTA) mechanism (or its variants) to do MaxPooling in SNNs; where the earliest occurring spike in a pooling area is sent to the next layer, with rest of the neurons (in the pooling area) reset to blocked. Similarly, [21] employ a lateral inhibition based method to do MaxPooling in SNNs. In another novel approach, [22] select a neuron (in the pooling region) with the highest membrane potential to output the spikes; they do a soft reset (i.e. reset by subtraction) of the firing neuron’s membrane potential. They also propose a hardware architecture for their method of MaxPooling. However, none of the above methods have been evaluated on a specialised neuromorphic hardware (e.g. Loihi [23]), with the exception of WTA (but on FPGA [18]).

The conversion of CNNs with AveragePooling layers to SNNs is trivial, as the AveragePooling can be modeled as convolution operation in the SNNs. Thus, a number of works replace the MaxPooling layers with AveragePooling layers [24], [8], [25], [26], [27], [28], [29], [30] or with Strided Convolutional layers [31], [32] in their network; often leading to weaker ANNs [14]. Overall, this dearth of spiking-MaxPooling approaches in SNNs and the neuromorphic hardware-unfriendliness of the existing ones motivated us to build spiking equivalents of the MaxPooling operation which can be entirely deployed on a neuromorphic hardware (e.g. Loihi). Our contributions are outlined below:

- We propose two methods to do MaxPooling in SNNs, and evaluate them on the Loihi neuromorphic chip
- In a first, we also deploy deep SNNs with MaxPooling layers on the Loihi boards with one of our methods
We next present our Methods of spiking-MaxPooling, followed by the Experiments & Results section, and a Discussion on the result analysis and adaptability of our methods.

II. Methods

In this section we present our proposed methods of spiking-MaxPooling. We start with each method’s elemental details, followed by their Proof-of-Concept Demonstration on the Loihi chip. Henceforth, otherwise stated, all the instances of “neurons” are Integrate & Fire (IF) spiking neurons.

A. Method 1: MAX join-Op

MAX join-Op (MJOP) based spiking-MaxPooling leverages the low level NxCore APIs made available through the NxSDK tool (to program the Loihi chips) by Intel; thus, this method is Loihi hardware dependent. A single Loihi chip consists of 128 Neuro-Cores, where each Neuro-Core implements 1024 Single Compartment (SC) spiking units. Each compartment can be simulated as an individual neuron or can become part of a Multi-Compartment (MC) neuron. Each MC neuron is a binary tree of single compartments, where each node (a compartment) can have at most two child nodes/dendrites (also compartments). Note that a MC neuron cannot span across two or more Neuro-Cores; rather is limited to just one Neuro-Core. Thus, a MC neuron can have a maximum of 1024 SC. Through each connection between the compartments (in a MC neuron), the two state variables: current \(U\) and voltage \(V\) can flow. Thus, a compartment can communicate either its \(U\) or \(V\) to its parent compartment. The parent compartment incorporates the state variables from its dendrites with its own \(U\) by following a join operation defined for its dendritic connections. Note that, an external \(U\) can be injected to each compartment (including the root compartment) in a MC neuron, thereby enabling them to spike (provided their \(V\) reaches the threshold). Also, note that the neurons (MC or SC) in a Loihi chip communicate via spikes only; they cannot exchange \(U\) or \(V\) directly, which falls in line with the biological neurons.

Fig. 1 shows an example of a MC neuron communicating its spikes to a SC neuron. The MC neuron has a binary tree structure, with the root node (i.e. soma compartment \(C\)) having two child leaf nodes (dendrite compartments \(A\) & \(B\)). In Loihi, each compartment’s voltage at time-step \(t\) i.e. \(X.V[t]\) (\(X\) can be a soma or its dendrites) is updated by the following rule:

\[
X.V[t] = X.V[t-1] \times (1 - decay) + X.dV[t]
\]

where the value of \(X.dV[t]\) is found by applying the specific join operation (join-Op) e.g. ADD, MAX, MIN, etc. over its current \(X.U[t]\) and the input state variables from its dendrites. We leverage this MC neuron creation functionality and the MAX join-Op for realizing spiking-MaxPooling (on Loihi boards) in deep SNNs. We next explain the MAX join-Op in the context of the MC neuron in Fig. 1. For the same, we first define few terms: \(X.U[t]\) and \(X.bias\) denote the input current and bias current (respectively) of a compartment \(X\). \(X.U'[t]\) is the sum of \(X.U[t]\) and \(X.bias\). \(A[t]\) and \(B[t]\) is the output of the child dendrite \(A\) (and \(B\)), which can either be \(A.V[t]\) or \(A.U'[t]\) (and \(B.V[t]\) or \(B.U'[t]\)) depending upon which state variable we want to work with; we choose \(U\). NxSDK defines the MAX join-Op as (\(X\) being \(C\) here):

\[
C.dV[t] = \text{max}(C.U'[t], A[t], B[t])
\]

We expand and simplify the Eq. 2 by assuming \(X.bias = 0\) for \(X \in \{C, A, B\}\) and setting the child compartments \(A\) and \(B\) to output current instead of \(V\) (to its parent \(C\)). Thus, \(A[t] = A.U'[t]\) and \(B[t] = B.U'[t]\). Eq. 2 simplifies as:

\[
C.dV[t] = \text{max}(C.U'[t], A[t], B[t])
\]

\[
= \text{max}(C.U[t] + C.bias, A[t], B[t])
\]

\[
= \text{max}(C.U[t], A.U'[t], B.U'[t])
\]

\[
= \text{max}(C.U[t], A.U[t], B.U[t])
\]

Thus, in MJOP case, the root compartment (i.e. soma \(C\)’s voltage dynamics is governed by the maximum of the input currents to it (from external source and its dendrites \(A\) & \(B\)).

1) MJOP Net for MaxPooling: : In a conventional CNN architecture, MaxPooling is done over the activations of the preceding Convolutional layer rate-neurons. In an SNN, we can represent those real-valued activations (of rate-neurons) by passing the corresponding spiking-neurons’ spike-trains
through a low-pass filter (also known as filtering/synapsing). In other words, the synapsed spikes i.e. the post-synaptic current $U$ represents the activation. We leverage this characteristic of the SNNs and feed the individual currents $U_i$ (in a pooling window) to a MAX join-Op configured MC neuron. Note that the number of compartments in the MC neuron should be equal to the size of the pooling window.

For a $2 \times 2$ MaxPooling window, we construct a MC neuron with 4 compartments as shown in the Fig. 2. The outgoing spikes from each neuron in a pooling window induce a post-synaptic current $U$ in the respective individual compartments (bias current of each compartment is set to 0). Each of the compartments (except the root/soma) is set to communicate its $U$ to its parent. The leaf node/compartment 4 upon receiving the post-synaptic current updates its $V$ and communicates the received $U$ to its parent 3. Compartment 3 then computes the MAX of the current from its child 4 and the incoming post-synaptic current, updates its $V$, and communicates the resulting maximum $U$ to its parent 2. Compartments 2 and 1 repeat this same process; except that the compartment 1 has no parent to communicate the MAX $U$ so far. The soma compartment 1 then spikes at a rate corresponding to the maximum input post-synaptic $U$ depending on its configuration, instead of outputting the max $U$. Note that in this network, MAX join-Op is executed over two arguments instead of three.

2) Proof-of-Concept Demonstration: In MJOP Net, a running MAX of input currents $U_i$ is maintained, which is finally fed to the root/soma compartment. Mathematically:

$$U_{\text{out}} = F(G(\text{max}(U_1, \text{max}(U_2, \text{max}(U_3, U_4)))))$$

(7)

where $U_i$ is the input current to compartment $i$, $G$ is the non-linear dynamical function of $U$ governing the voltage dynamics (thus, the spiking output) of soma, $F$ is the synaptic filter applied on the spike outputs of soma. Since the soma does not communicate the computed max current, rather its spikes to the next neuron (if connected), there arises a need to properly scale the synapped spikes i.e. $U_{\text{out}}$ to match “True Max $U$” ($= \text{max}(U_1, U_2, U_3, U_4)$ computed without spiking neurons on a non-neuromorphic hardware). We next execute the MJOP Net (in Fig. 2) on the Loihi chip. The example MJOP net consists of a MAX join-Op configured MC neuron of 4 compartments receiving periodic spiking inputs (spike amplitude 1, with time-periods of 10, 8, 4, and 6 - a possible case when receiving the spike outputs of pooled neurons in a preceding Convolutional layer in an SNN). We set a probe on soma (compartment 1) and filter its output spikes. In Fig. 3a we see that the “Scaled $U_{\text{out}}$” matches the “True Max $U$” closely; the “Average $U$” is lower than the estimated max. Note that the MJOP Net is deployable on Loihi only (not on GPUs), because it is built on the NxCore APIs native to Loihi.

B. Method 2: Absolute Value based Associative Max

The representation of the real-valued activations of the rate-neurons as currents in SNNs inspires our second method as well. The Absolute Value based Associative Max (AVAM) method of spiking-MaxPooling is hardware independent and leverages the following two properties of the max function.

$$\max(a, b) = \frac{a + b}{2} + \frac{|a - b|}{2}$$

(8)

$$\max(a_1, a_2, a_3, \cdots, a_n) = \max(\max(a_1, a_2), \cdots, \max(a_{n-1}, a_n))$$

(9)

where $a_1, a_2 \in \mathbb{R}, n \in \mathbb{N}$; Eq. 9 holds due to the Associative Property of $\max()$. In Eq. 8, the average term $\frac{a + b}{2}$ (a linear operation) can be easily calculated via the connection weights from the neurons representing those values (i.e. $a$ and $b$); the challenge is to calculate the non-linear absolute value function i.e. $|.|$. One can use the Neural Engineering Framework (NEF) principles [33] to estimate the $|.|$ function by employing a network of Ensembles of neurons. However, the number of neurons in each Ensemble can be large (100s or more); thus, this method is not scalable. For the same reasons, a direct calculation of the $\max()$ using NEF principles is not desirable.

1) Estimation of $|.|$ function: Therefore, we rather take a unique approach of using the tuning curves to estimate the $|.|$ function. Tuning curves characterize the activation (i.e. firing rate) profile of neurons with respect to the input stimulus. In NEF, these curves depend on the neuron type and properties (e.g. max firing rate $\phi$, representational radius $r$, etc.). Upon configuring an Ensemble of two IF neurons properly (e.g. $\phi = 200$Hz, $r = 2.5$), one can obtain desirable tuning curves as shown in the Fig. 4a, which resembles the plot of the $|.|$ function; we leverage the same to estimate the absolute value of a signed scalar input. In our example (Fig. 4a), “Neuron 1” (and “Neuron 2”) is tuned to fire at $\phi = 200$Hz when $2.5$ (and $-2.5$) is fed to the Ensemble. Thus, irrespective of the sign of the input values $\in [-r, \cdots, r]$, we receive a positive firing rate from either neuron. Therefore, for an input $r$ (or $-r$), we can filter the spiking output from the corresponding neuron (the other outputs 0) to obtain $\phi$ and scale it with $\frac{r}{\phi}$ (i.e. $\phi \times \frac{r}{\phi}$) to estimate $|r|$. Note that for such a system of two neurons, one needs to pre-determine a max firing rate $\phi$ and the approximate representational radius $r$. If the magnitude of the input value is lesser (or larger) than $r$, then this system produces a noisier (and saturated) output.

![Fig. 3. “MJOP Net $U_{\text{out}}$” is the synapsed/filtered spiking output from the soma. It is scaled by 1.1 (to obtain “Scaled $U_{\text{out}}$”) to match the “True Max $U$” = $\max(U_1, U_2, U_3, U_4)$ closely. “AVAM Net $U_{\text{out}}$” are the outputs from Node 0 (in Fig. 5) for different radii $r$. For $r = 0.25$ and 0.20, the $U_{\text{out}}$ matches the “True Max $U$” = $\max(U_1, U_2, U_3, U_4)$ closely.](image-url)
2) Estimation of $max(a, b)$: The above method of estimating the $\lvert \cdot \rvert$ function can be incorporated with the average term calculation to construct a network as shown in the Fig. 4b to estimate the $max(a, b)$. In Fig. 4b, Nodes $A$ and $B$ represent and output the values $a$ and $b$ respectively as currents. They are directly connected to the Node $O$ with a connection weight of $1/2$ each. Thus, their scaled output $i.e. a/2$ and $b/2$ gets summed up at Node $O$ to result in $(a + b)/2$. Nodes $A$ and $B$ are also connected to an Ensemble of two Neurons, 1 and 2 (whose tuning curves are similar to that in Fig. 4a and their representational radius $r \approx (\lvert a - b \rvert)/2$ with a connection weight of $1/2$ and $-1/2$ respectively, such that the sum $(a - b)/2$ fed to the Ensemble. Note that $r$ can be heuristically set without knowing the actual values of $a$ and $b$ (shown later). Depending on the sign of the sum $(a - b)/2$, either the Neuron 1 or Neuron 2 spikes at a frequency $\phi$ (the other outputs 0). Therefore, after filtering the spike outputs to obtain $\phi$ and scaling it with $\frac{\phi}{2} (= \phi \times \frac{1}{2})$ through the weighted connection to the Node $O$, $r$ is sent. The Node $O$ then finally accumulates the inputs, i.e. $\frac{a+b}{2} + r \approx \frac{a+b}{2} + \frac{|a-b|}{2}$ which is approximately equal to the $max(a, b)$ and relays the same.

3) Proof-of-Concept Demonstration: For a $2 \times 2$ MaxPooling window, we can compute the $max(a_1, a_2, a_3, a_4)$ as $max(max(a_1, a_2), max(a_3, a_4))$ using the Associative Property of $max()$. We therefore construct an example AVAM Net, shown in Fig. 5 with four spiking inputs (firing time-period $= 10, 8, 4, 6$) connected to the individual Nodes. Note that the filtered spikes i.e. currents $U_i$ are being fed to the Nodes (here $a_i = U_i$) and they relay the same to the next connected components. In this hierarchical network, $max(a_1, a_2)$ and $max(a_3, a_4)$ gets estimated at the Nodes $P$ and $Q$ respectively. They forward the same and Node $O$ finally estimates the $max(max(a_1, a_2), max(a_3, a_4))$. Three instances of this AVAM Net were executed on the Loihi chip with IF neurons’ $\phi$ fixed to 500Hz and $r \in [0.20, 0.25, 0.30]$. In each instance, all the neurons in each Ensemble had the same $\phi$ and $r$. Each instance’s estimated $max(U_1, U_2, U_3, U_4)$ i.e. “AVAM Net $U_{out}$” (for a corresponding $r$) is shown in the Fig. 3b. We see that $r$’s value around 0.25 (for a fixed $\phi$) does a fair job of approximating the “True Max $U$”; as well as, the estimated max $U$ are higher than the Average $U$.

One should note that ideally, the output current from a MaxPooling window (of spiking neurons) should be the current due to the maximally firing neuron (i.e. “Exact Max $U$”). However, the MJOP and AVAM spiking-MaxPooling methods compute the instantaneous max of all incoming currents (in a pooling window) at each time-step, which is not equivalent to the “Exact Max $U$”. It is possible that the instantaneous max of currents could be higher than the “exact max” when a slower spiking neuron fires recently than the maximally firing neuron. This holds true with “True Max U” as well; however, we defined it as $max()$ of currents in the spirit of MaxPooling in ANNs. Therefore, our methods are approximations of the ideal MaxPooling in SNNs.

C. Heuristics for scale (MJOP) and radius (AVAM)

As seen in the Proof-of-Concept Demonstration sections, one needs to properly $scale$ the $U_{out}$ in case of MJOP Net and set the radius parameter in AVAM Net to correctly approximate the “True Max $U$”. For a fixed configuration of the compartments/neurons in the MJOP and AVAM Net, the value of $scale$ and radius depends on the group of inputs $U_i$. Recollect that in the MJOP Net, the soma compartment spikes at a rate corresponding to the maximum input $U_i$, and the output $U_{out}$ needs to be scaled accordingly. And in the AVAM Net, the radius should be heuristically chosen to be equal to $|a - b|/2$ for estimating the $max(a, b)$ (where $a$ and $b$ are the inputs $U_i$). The inputs $U_i$ in turn depend on the periodicity (or the Inter-Spike Interval (ISI)) of the incoming individual spike trains. We note here that the maximum and minimum value of $U_i$ can be 1 (with spike amplitude = 1, ISI = 1) and 0 (with the corresponding neuron not spiking at all) respectively. This implies that the maximum value of the difference of two $U_i$s can be 1, i.e. radius $\leq 1$ always. This holds true for the radius value of the Ensemble neurons further in the AVAM Net hierarchy. Moreover, many or all of the neurons in a pooling window may spike with ISI $> 1$, which would further lower down the radius value.
For our MJOP & AVAM Nets, the efficacy of our spiking-MaxPooling methods shows the true and estimated $U$ are $\leq$ datasets (normalized MaxPooling operation via our proposed methods of spiking-MaxPooling) properly tuned to account for the firing-rate quantization boards. While training the models we ensure that they are NengoLoihi library for deploying the SNNs on the Loihi spiking network (of IF spiking neurons). We use the NengoDL (ReLU) based model and then convert it to a ReLU layer (preceding the MaxPooling layer) for a dataset; although, analysing the ISI distribution of the neurons in a model’s Conv layer are flattened and mapped to a Neuro-Core each. Each architecture. The individual channels of the preceding Conv layer’s channels.

### III. EXPERIMENTS & RESULTS

We next describe the experiments conducted with the MJOP and AVAM methods of spiking-MaxPooling. In accordance with the ANN-to-SNN conversion paradigm, we first train a rate-neuron (ReLU) based model and then convert it to a spiking network (of IF spiking neurons). We use the NengoDL library [34] for training, conversion, and inference; and the NengoLoihi library for deploying the SNNs on the Loihi boards. While training the models we ensure that they are properly tuned to account for the firing-rate quantization of IF spiking neurons. In the converted SNNs, we do the MaxPooling operation via our proposed methods of spiking-MaxPooling Nets. We use the MNIST, FMNIST, and CIFAR10 datasets (normalized $[-1,1]$) and conduct experiments with 3 different architectures (Fig. 7). For simplicity, we fix the MaxPooling window to $2 \times 2$ size in all the architectures; also, $2 \times 2$ window is one of the most common pooling windows.

#### A. Common settings in MJOP and AVAM methods

For our proposed spiking-MaxPooling methods, each Conv-channel is flattened and the $2 \times 2$ pooling window inputs are grouped in sizes of 4, and passed next to the layer of MJOP Nets or AVAM Nets depending on the choice of spiking-MaxPooling. The outputs from the individual Nets are collected in a channel wise manner and passed to the next Conv layer. While flattening the channels and passing the pooled inputs to the spiking-MaxPooling layer, and subsequently while collecting the outputs, one has to properly arrange the inputs and outputs to preserve the $2 \times 2$ MaxPooling layout. Note that the MJOP and AVAM methods are compatible with any ordering of the preceding Conv layer’s channels.

#### B. Model Details

The first Conv layer in each architecture (in Fig. 7) has a kernel size of $(1,1)$ which acts a pixel-value to spike converter. For training each architecture, we use the Adam optimizer with a learning rate of $1e^{-3}$ and a decay of $1e^{-4}$, and use the Categorical Crossentropy loss function with logits. For MNIST, Architectures 1, 2, and 3 were trained for 8 epochs each; for CIFAR10, 1 and 2 were trained for 64 epochs each, and 3 for 164 epochs; and for FMNIST, 1 and 2 were trained for 24 epochs each, and 3 for 64 epochs. The training was done on the NVIDIA Tesla P100 GPUs. During inference with SNNs, the individual test images (irrespective of the dataset) were presented for 50, 60, and 120 time-steps to the Architectures 1, 2, and 3 respectively. Code is public 1.

#### C. SNNs with MJOP Net based spiking-MaxPooling

Since MJOP method is Loihi dependent (and not supported on the GPUs), we execute the converted SNNs right on the Loihi boards in inference mode. Fig. 8 shows an example of how MJOP spiking-MaxPooling can be done in any arbitrary architecture. The individual channels of the preceding Conv layer are flattened and mapped to a Neuro-Core each. Each Neuro-Core has an Ensemble of MAX join-Op configured MC neurons (with 4 compartments each as shown in Fig. 2) to do the $2 \times 2$ MaxPooling. The outputs (quartered in length) from each Ensemble deployed on the individual Neuro-Cores are collected and reshaped as individual channels, and then fed to the next Conv layer. This process is repeated for any additional MaxPooling layers in the network. While deploying the architectures on the Loihi boards, the first Conv layer and the last “Output” layer are executed Off-Chip; rest of the layers run On-Chip. Inference is done over the entirety of each dataset for varying scale values. Due to the Loihi hardware resource constraints and no support for same padding (in Conv layers) in NengoLoihi (1.1.0.dev0), we were unable to execute Architecture 3 on the Loihi boards. Table I shows the accuracy results for Architectures 1 and 2 - both run on Loihi.

---

1https://github.com/R-Gaurav/SpikingMaxPooling
Fig. 7. CNN Architectures. “Conv2D, x” ⇒ “x” number of filters in the Conv layer; “Dropout, x” ⇒ “x” dropout probability. In each architecture, Conv layer strides = (1, 1), kernel size = (3, 3) (except the first Conv layer with kernel size = (1, 1)); 2 × 2 MaxPooling window; 128 neurons in Dense layer; no activation in the Output layer; no bias in layers except Dense. MaxPool layers in all architectures have no padding; Conv layers too have no padding except in Architecture 3. For experiments with MNIST & FMNIST, the colored blocks in Architecture 3 are removed to account for their smaller image size than CIFAR10. During the inference phase with the SNNs obtained after the conversion of ANNs, the “Dropout, x” layers are removed.

Fig. 8. MAX join-op based MaxPooling. For a MaxPool layer in an architecture, the preceding Conv layer’s channels are each flattened and mapped to a single Neuro-Core. Each Neuro-Core has an Ensemble of MJOI Net configured MC neurons. Post execution of MJOI Nets on the Neuro-Cores, the flattened vectors are reshaped to channels and passed to the next Conv layer.

| Architecture 1 | Architecture 2 | Architecture 3 |
|----------------|----------------|----------------|
| NSR TMS MAS MJOP AVAM | NSR TMS MAS MJOP AVAM | NSR TMS MAS MJOP AVAM |
| CIFAR10 | 60.2 | 60.6 | 48.7 | 55.0 | 55.1 | 81.2 | 86.0 | 80.6 | 0.4 | 60.5 | 60.7 | 59.9 | 65.3 | 64.9 | 38.8 | 55.7 | 81.2 | 26.3 | 64.1 | 66.0 | 64.1 | 83.7 | 72.8 | 69.6 | 28.7 | 82.7 | 82.8 | 71.3 |
| MNIST | 98.8 | 98.8 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 | 98.2 |
| FMNIST | 91.0 | 90.5 | 87.3 | 88.0 | 88.1 | 88.0 | 80.2 | 90.2 | 90.3 | 89.4 | 89.9 | 89.5 | 79.5 | 86.4 | 83.5 | 89.0 | 89.2 | 89.1 | 88.3 | 93.4 | 93.2 | 91.6 | 93.2 | 93.3 | 93.2 | 93.3 |

**TABLE I**

Accuracy Results (%). **NSR:** NON-SPIKING RELU ANN’s results; **TMS:** TRUE-MAX U SNN’s results; **MAS:** MAX-TO-AVG SNN’s results; **MJOI & AVAM:** SPIKING-MAXPOOLING SNN’s results. For CIFAR10 $s_1$, $s_2$, $s_3$ (scale) values are 1.0, 2.0, 5.0; for MNIST they are 1.0, 1.2, 2.0; and for FMNIST they are 1.0, 1.5, 2.0 resp. $R_s$, $R_h$, $R_c$ (radius) values vary the datasets are 0.20, 0.25, 0.30, 1.0 resp. As expected, $R_d$ = 1.0 results in accuracy loss, since difference of $U_d$ is not always ≈ 1.0; thus, poor approximation of max $U_d$. Note that the SNNs do not necessarily outperform the ANNs because they are obtained after the ANN-to-SNN conversion. Also note that the MJOI and AVAM experiments are conducted on LOIHI and GPUs respectively, for all the architectures.
D. SNNs with AVAM Net based spiking-MaxPooling

AVAM method of spiking-MaxPooling is hardware independent. Although the individual AVAM Nets with varying radii were executed on Loihi (Fig. 3b), executing SNNs with them on the Loihi boards gets challenging due to the AVAM Net layers exceeding the maximum supported number of input and output axons on the Loihi hardware. Therefore, we execute the SNNs with AVAM method of spiking-MaxPooling on GPUs only. Similar to the case of MJOP Net based SNNs, the individual channels in the preceding Conv layer are flattened and the pooling window’s grouped values (in sizes of 4) are passed to the layer of AVAM Nets to estimate the max values; which are then collected, reshaped as channels, and forwarded next. This process is repeated for any additional MaxPooling layers in the network. In the experiments with each of the three architectures, \( \phi \) is set to 250Hz, and the radius value is varied; Table I shows the accuracy results.

IV. Discussion

A. Table I Result Analysis

To evaluate the efficacy of SNNs with our proposed methods of spiking-MaxPooling, we compare it against the baseline results obtained with the “True Max \( U \)” based SNNs (column TMS) and its ReLU based non-spiking counterpart (column NSR, with TensorFlow) - both with regular \( \text{max}() \) based MaxPooling, as well as with the SNNs where MaxPooling layers are replaced with AveragePooling layers after training (column MAS, only Ensembles and associated connections removed in AVAM Net). In case of MNIST, the MJOP based SNNs perform similar to their non-spiking counterpart and the “True Max \( U \)” based SNN across all the architectures. In case of FMNIST, the performance drop of MJOP based SNNs is noticeable; and in case of CIFAR10, they perform very poorly. One reason for the accuracy drop is the Loihi hardware constraints i.e. 8-bit quantization of the network weights and fixed-point arithmetic. We found the other reason in case of CIFAR10 to be the highly varying ISI distribution (of Conv layer neurons) across the test images – possibly due to color channels (resulting in highly variable neuron activations). This prevented the choice of a scale value to generalize well across all the test images. For MNIST (and to a large extent for FMNIST), we found the ISI distribution of the Conv layer neurons to be light-tailed and mostly similar across the test images. More details on the ISI distribution here \(^2\). In a separate experiment, setting two different scale values (2 and 1.5) for the corresponding MaxPooling layers in Architecture 2 for FMNIST did not improve the results. Overall, although the MJOP based SNNs were successfully deployed on Loihi, the MJOP spiking-MaxPooling seems suboptimal due to its poor generalizability. For its optimal performance, the maximally firing neuron in each pooling window (in a preceding Conv layer) should have (nearly) same ISI for a consistent effect of the scale value. On the other hand, AVAM based SNNs perform at par with their non-spiking counterpart and the “True Max \( U \)” based SNN across all the three datasets and architectures. An important distinction between the MJOP & AVAM methods is that the MJOP method estimates the “True Max \( U \)” indirectly by scaling the \( U_{\text{out}} \) (note that this \( U_{\text{out}} \) corresponds to the maximum input \( U_i \)), whereas the AVAM method estimates the “True Max \( U \)” directly from the group of \( U_i \), which makes the AVAM spiking-MaxPooling more robust and effective; and its optimal performance with the same set of radius values across all the three datasets and architectures promises its generalizability. It is interesting to note that in some cases, AVAM based SNNs perform better than their non-spiking counterpart and/or “True Max \( U \)” based SNNs. Also, the SNNs with AveragePooling layers (column MAS) perform poorly compared to all other networks.

B. Adapting MJOP & AVAM spiking-MaxPooling

Our proposed methods of spiking-MaxPooling are suitable for the rate-based SNNs which represent activations as currents (i.e. filtered/synapsed spikes). SNNs which do not filter the spike trains and work directly on binary spikes [8], [35, . . . ], cannot adapt our proposed methods; this holds true for Time-To-First-Spike based SNNs as well. With respect to the scalability of the MJOP and AVAM Net against the size of pooled inputs, it is linear. For an \( r \times c \) MaxPooling window (where \( r, c \in \mathbb{N} \)), the number of compartments/neurons required in the MJOP and AVAM Net is \( r \times c \) and \( 2 \times (r \times c) - 2 \) respectively. However, with the increase in the number of neurons in the hierarchical AVAM Net, the estimated max output may get noisier; although, this doesn’t hold true for MJOP based method. AVAM Net based spiking-MaxPooling is deployable on any neuromorphic hardware that supports weighted connections and spiking neurons. In our MJOP based SNNs experiments, each channel of a Conv layer (prior to a MaxPooling layer) was small enough in dimensions such that the flattened vector was of size \( \leq 1024 \), thus easily mapped to a Neuro-Core. If the channel’s dimensions are sufficiently large and the flattened vector’s size is \( > 1024 \), then one needs to spatially split the channel and properly map it to more than one Neuro-Core such that no pooling window (thus the corresponding MC neuron) spans across two or more Neuro-Cores, as Loihi restricts the creation of a MC neuron on one Neuro-Core only. However, such a procedure need not be followed with the AVAM based SNNs (if deployed on GPUs).

V. Conclusion

To our best knowledge, this work is a first to present two different hardware-friendly methods of spiking-MaxPooling operation in SNNs, with both their evaluation on Loihi. In a first, we also deployed SNNs with MaxPooling layers (via MJOP method) on the Loihi boards. Note that our goal was not to outperform the SoTA results on the experimented datasets, rather to show the efficacy of our hardware-friendly spiking-MaxPooling methods, which we do so by achieving comparable results with the regular MaxPooling based ANNs (column NSR in Table I). For the appropriate choice of the tunable scale and radius values in MJOP & AVAM Net respectively,
we also presented a heuristic method. The Proof-of-Concept Demonstrations of both the methods on Loihi show that our work makes successful strides towards providing a solution to the MaxPooling problem in SNNs. This also opens up avenues for building hardware ready SNNs with MaxPooling layers without compromising on the quality (otherwise due to replacing MaxPooling with AveragePooling). One immediate future direction of our work is to enable the deployment of AVAM based SNNs on a neuromorphic hardware. Energy consumption comparison between MJOP and AVAM could be a next step too. What about estimating the radius of the AVAM Net from the pooled activations of the original ANN? Finally, we hope that our work encourages efforts towards developing other hardware-friendly methods of spiking-MaxPooling.

ACKNOWLEDGMENT

We would like to thank Eric Hunsberger and Xuan Choo (from ABR \(^3\)) for their help with the NengoLoihi and NxCore details to deploy our spiking-MaxPooling methods on Loihi. We thank the anonymous reviewers for their insightful comments and suggestions to extend this work. We also thank Intel for providing cloud access to the Loihi hardware. This work is supported by the NSERC Grant Award AWD-013766.

REFERENCES

[1] E. Stromatias, D. Neil, M. Pfeiffer, F. Galluppi, S. B. Furber, and S.-C. Liu, “Robustness of spiking deep belief networks to noise and reduced bit precision of neuro-inspired hardware platforms,” Frontiers in neuroscience, vol. 9, p. 222, 2015.

[2] C. Li, R. Chen, C. Moutafis, and S. Furber, “Robustness to noisy synaptic weights in spiking neural networks,” in 2020 International Joint Conference on Neural Networks (IJCNN), 2020, pp. 1–8.

[3] S. Sharmin, P. Panda, S. S. Sarwar, C. Lee, W. Ponghiran, and K. Roy, “A comprehensive analysis on adversarial robustness of spiking neural networks,” in 2019 International Joint Conference on Neural Networks.

[4] S. Sharmin, N. Rathi, P. Panda, and K. Roy, “Inherent adversarial robustness of deep spiking neural networks: Effects of discrete input encoding and non-linear activations,” in European Conference on Computer Vision. Springer, 2020, pp. 399–414.

[5] M. Pfeiffer and T. Pfeil, “Deep learning with spiking neurons: Opportunities and challenges,” Frontiers in Neuroscience, vol. 12, p. 774, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2018.00774

[6] M. Zhang, J. Wang, J. Wu, A. Belatreche, B. Amornpaisarn, Z. Zhang, V. P. K. Miriyala, H. Qu, Y. Chua, T. E. Carlson et al., “Rectified linear postsynaptic potential function for backpropagation in deep spiking neural networks,” IEEE Transactions on Neural Networks and Learning Systems, 2021.

[7] M. Zhang, J. Wu, Y. Chua, X. Luo, Z. Pan, D. Liu, and H. Li, “Mpd-al: an efficient membrane potential driven aggregate-label learning algorithm for spiking neurons,” in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, no. 01, 2019, pp. 1327–1334.

[8] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, “Going deeper in spiking neural networks: Vgg and residual architectures,” Frontiers in neuroscience, vol. 13, p. 95, 2019.

[9] Y.-L. Bourjot, J. Ponce, and Y. LeCun, “A theoretical analysis of feature pooling in visual recognition,” in Proceedings of the 27th international conference on machine learning (ICML-10), 2010, pp. 111–118.

[10] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[11] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 1251–1258.

[12] M. Tan and Q. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” in International Conference on Machine Learning. PMLR, 2019, pp. 6105–6114.

[13] Y. Hu and M. Pfeiffer, “Max-pooling operations in deep spiking neural networks,” 2016.

[14] B. Rueckauer, L.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, “Conversion of continuous-valued deep networks to efficient event-driven networks for image classification,” Frontiers in neuroscience, 2017.

[15] S. Kim, S. Park, B. Na, and S. Yoon, “Spiking-yolo: Spiking neural network for energy-efficient object detection,” in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, no. 07, 2020.

[16] T. Masquelier and J. S. Thorpe, “Unsupervised learning of visual features through spike timing dependent plasticity,” PLoS computational biology, vol. 3, no. 2, p. e31, 2007.

[17] B. Zhao, S. Chen, and H. Tang, “Bio-inspired categorization using event-driven feature extraction and spike-based learning,” in 2014 International Joint Conference on Neural Networks (IJCNN), 2014.

[18] G. Orchard, C. Meyer, R. Etienne-Cummings, C. Posch, N. Thakor, and R. Benosman, “Hirst: A temporal approach to object recognition,” IEEE transactions on pattern analysis and machine intelligence, 2015.

[19] J. Li, W. Hu, Y. Yuan, H. Hao, and T. Fang, “Bio-inspired deep spiking neural network for image classification,” in International Conference on Neural Information Processing. Springer, 2017, pp. 294–304.

[20] M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini, and T. Masquelier, “Spyketorch: Efficient simulation of convolutional spiking neural networks with at most one spike per neuron,” Frontiers in neuroscience, vol. 13, p. 625, 2019.

[21] Z. Lin, J. Shen, D. Ma, and J. Meng, “Quantisation and pooling method for low-inference-latency spiking neural networks,” Electronics Letters, vol. 53, no. 20, pp. 1347–1348, 2017.

[22] D.-A. Nguyen, X.-T. Tran, K. N. Dang, and F. Iacopi, “A lightweight max-pooling method and architecture for deep spiking convolutional neural networks,” in 2020 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS). IEEE, 2020, pp. 209–212.

[23] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain et al., “Loihi: A neuromorphic manycore processor with on-chip learning,” IEEE Micro, 2018.

[24] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, “Spatio-temporal backpropagation for training high-performance spiking neural networks,” Frontiers in neuroscience, vol. 12, p. 331, 2018.

[25] X. Cheng, Y. Hao, J. Xu, and B. Xu, “Lisnn: Improving spiking neural networks with lateral interactions for robust object recognition,” in IJCAI, 2020, pp. 1519–1525.

[26] I. Garg, S. S. Chowdhury, and K. Roy, “Det-snn: Using dct to distribute spatial information over time for learning low-latency spiking neural networks,” arXiv preprint arXiv:2010.01795, 2020.

[27] A. S. Kucik and G. Meoni, “Investigating spiking neural networks for energy-efficient on-board ai applications: a case study in land cover and land use classification,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 2020–2030.

[28] S. Kundu, G. Datta, M. Pedram, and P. A. Beerler, “Spike-thrift: Towards energy-efficient deep spiking neural networks by limiting spiking activity via attention-guided compression,” in Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2021.

[29] Z. Yan, J. Zhou, and W.-F. Wong, “Near lossless transfer learning for spiking neural networks,” in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, no. 12, 2021, pp. 10577–10584.

[30] X. Huang, E. Jones, S. Zhang, S. Xie, S. Furber, Y. Goulermas, E. Marsden, I. Baistow, S. Mitra, and A. Hamilton, “An fpga implementation of convolutional spiking neural networks for radiosotope identification,” in 2021 IEEE International Symposium on Circuits and Systems (ISCAS).

[31] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch et al., “Convolutional networks for fast, energy-efficient neuromorphic computing,” Proceedings of the national academy of sciences, 2016.

[32] K. Patel, E. Hunsberger, S. Batir, and C. Eliasmith, “A spiking neural network for image segmentation,” arXiv:2106.08921, 2021.

[33] C. Eliasmith and C. H. Anderson, Neural engineering: Computation, representation, and dynamics in neurobiological systems. MIT press.

[34] D. Rasmussen, “Nengodl: Combining deep learning and neuromorphic modelling methods,” Neuroinformatics, vol. 17, no. 4, 2019.

[35] B. Han and K. Roy, “Deep spiking neural network: Energy efficiency through time based coding,” in Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings.