HOYER REGULARIZER IS ALL YOU NEED FOR ULTRA LOW-LATENCY SPIKING NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Spiking Neural networks (SNN) have emerged as an attractive spatio-temporal computing paradigm for a wide range of low-power vision tasks. However, state-of-the-art (SOTA) SNN models either incur multiple time steps which hinder their deployment in real-time use cases or increase the training complexity significantly. To mitigate this concern, we present a training framework (from scratch) for one-time-step SNNs that uses a novel variant of the recently proposed Hoyer regularizer. We estimate the threshold of each SNN layer as the Hoyer extremum of a clipped version of its activation map, where the clipping threshold is trained using gradient descent with our Hoyer regularizer. This approach not only downscales the value of the trainable threshold, thereby emitting a large number of spikes for weight update with limited number of iterations (due to only one time step) but also shifts the membrane potential values away from the threshold, thereby mitigating the effect of noise that can degrade the SNN accuracy. Our approach outperforms existing spiking, binary, and adder neural networks in terms of accuracy-FLOPs trade-off for complex image recognition tasks. Downstream experiments on object detection also demonstrate the efficacy of our approach. Codes will be made available.

1 INTRODUCTION & RELATED WORKS

Due to its high activation sparsity and use of cheaper accumulates (AC) instead of energy-expensive multiply-and-accumulates (MAC), SNNs have emerged as a promising low-power alternative to compute- and memory-expensive deep neural networks (DNN) Indiveri et al. (2011); Pfeiffer et al. (2018); Cao et al. (2015). Because SNNs receive and transmit information via spikes, analog inputs have to be encoded with a sequence of spikes using techniques such as rate coding Diehl et al. (2016), temporal coding Comsa et al. (2020), and rank-order coding Kheradpisheh et al. (2018). Recently, a new form of encoding called direct encoding Rathi et al. (2020a) has gained significant traction that directly feeds the analog pixel values in the first convolutional layer, and thereby, emits spikes only in the subsequent layers. This helps reduce the number of time steps needed to achieve SOTA accuracy by an order of magnitude. Although the first layer now requires MACs, the overhead is negligible for deep convolutional architectures Rathi et al. (2020a). Because we focus on SNNs with reduced time steps, we adopt direct encoding in this work.

In addition to accommodating various forms of spike encoding, supervised training algorithms for SNNs have overcome various roadblocks associated with the discontinuous spike activation function Lee et al. (2016); Kim et al. (2020). Moreover, SNNs can be converted from DNNs with low error by approximating the activation value of ReLU neurons with the firing rate of spiking neurons Sengupta et al. (2019). SNNs trained using DNN-to-SNN conversion, coupled with supervised training, have been able to perform similar to SOTA DNNs Rathi et al. (2020b;a) in image recognition tasks. However, most of these efforts require multiple time steps which increases training and inference costs compared to non-spiking counterparts. The training effort is high because backpropagation must integrate the gradients over an SNN that is unrolled once for each time step Panda et al. (2020). Moreover, the multiple forward passes result in an increased number of spikes, which degrades the SNN's energy-efficiency, both during training and inference, and possibly offsets the compute advantage of the ACs. The multiple time steps also increase the inference complexity because of the need for input encoding logic and the increased latency associated with requiring one forward pass per time step. A recent SNN work Chowdhury et al. (2021) reduced the number of time steps

to unity, however it required CNN pre-training, followed by iterative SNN training from 5 to 1 time steps, significantly increasing the training complexity, particularly for ImageNet-level tasks.

Our Contributions. We propose a novel training framework for one-time-step SNNs that does not require any non-spiking DNN pre-training. The resulting SNNs are more compute-efficient than existing multi-time-step SNNs and have test accuracy similar to iso-architecture non-spiking DNNs. Our framework is based on a novel application of the Hoyer regularizer for the SNN membrane potential tensor and a novel Hoyer spike layer. More specifically, our spike layer threshold is training-input-dependent and set to the Hoyer extremum of a clipped version of the membrane potential tensor, where the clipping threshold (existing SNNs use this as the threshold) is trained using gradient descent with our Hoyer regularizer. In this way, compared to SOTA one-time-step non-iteratively trained SNNs, our threshold increases the rate of weight updates and our Hoyer regularizer shifts the membrane potential distribution away from this threshold, improving convergence.

We consistently surpass the accuracies obtained by SOTA one-time-step SNNs Chowdhury et al. (2021) on diverse image recognition datasets with different convolutional architectures, while reducing the average training time by $\sim 19\times$. Compared to binary neural networks (BNN) and adder neural network (AddNN) models, our SNN models yield similar test accuracy with a $\sim 5.5\times$ reduction in the floating point operations (FLOPs) count, thanks to the extreme sparsity enabled by our training framework. Downstream tasks on object detection also demonstrate that our approach surpasses the test mAP of existing BNNs and SNNs.

2 Preliminaries

2.1 LIF Model

Leaky Integrate and Fire (LIF) is one of the most popular spiking neuron models that is used to train and infer SNNs. One of its variants without the reset can be illustrated as Rathi et al. (2020a):

$$\boldsymbol{u}_l^t = \lambda \boldsymbol{u}_l^{t-1} + \boldsymbol{w}_l \boldsymbol{o}_{l-1}^t \qquad \boldsymbol{o}_l^{t-1} = \begin{cases} 1, & \text{if } \boldsymbol{u}_l^{t-1} \geq v_l^{th}; \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where $\boldsymbol{u}_l^t$ represents the membrane potential tensor of the lth layer at the tth time step, λ is a leak factor that varies between 0 and 1, w_l is the weight connecting layers $(l-1)$ and l, $\boldsymbol{o}_{l-1}^t$ is the spike output of the $(l-1)^{th}$ layer at the tth time step, v_l^{th} is the threshold that is kept constant for layer l.

2.2 Hoyer Regularizer

Based on the interplay between L1 and L2 norms, a new measure of sparsity was first introduced in Hoyer (2004), based on which reference Yang et al. (2020) proposed a new regularizer, termed the Hoyer regularizer for the trainable weights that was incorporated into the loss term to train DNNs. We adopt the same form of Hoyer regularizer for the membrane potential to train our SNN models as $H(\boldsymbol{u}_l) = \left(\frac{\|\boldsymbol{u}_l\|_1}{\|\boldsymbol{u}_l\|_2}\right)^2$ Kurtz et al. (2020). Here, $\|\boldsymbol{u}_l\|_i$ represents the Li norm of the tensor $\boldsymbol{u}_l$, $|\boldsymbol{u}_l|$ represents the element-wise absolute of the tensor $\boldsymbol{u}_l$, and the superscript t for time step is omitted for simplicity. Compared to the L1 and L2 regularizers, the Hoyer regularizer has scale-invariance (similar to L0 regularizer). It is also differentiable almost everywhere as shown in equation 2.

$$\frac{\partial H(\boldsymbol{u}_l)}{\partial \boldsymbol{u}_l} = 2sign(\boldsymbol{u}_l)\frac{\|\boldsymbol{u}_l\|_1}{\|\boldsymbol{u}_l\|_2^4}(\|\boldsymbol{u}_l\|_2^2 - \|\boldsymbol{u}_l\|_1|\boldsymbol{u}_l|) \tag{2}$$

Letting the gradient $\frac{\partial H(\boldsymbol{u}_l)}{\partial u_l} = 0$, we estimate the value of the Hoyer extremum as $H_e(\boldsymbol{u}_l) = \frac{\|\boldsymbol{u}_l\|_2^2}{\|\boldsymbol{u}_l\|_1}$.

This extremum is actually the minimum, because the second derivative is greater than zero for any value of the output element. Training with the Hoyer regularizer can effectively help push the activation values that are larger than the extremum ($\boldsymbol{u}_l > H_e(\boldsymbol{u}_l)$) even larger and those that are smaller than the extremum ($\boldsymbol{u}_l < H_e(\boldsymbol{u}_l)$) even smaller.

3 Proposed Training Framework

Our approach is inspired by the fact that Hoyer regularizers can shift the pre-activation distributions away from the Hoyer extremum in a non-spiking DNN Yang et al. (2020). Our approach is based on the observation that ensuring this extremum is the SNN threshold would shift the distribution

of membrane potentials around the threshold value, reducing noise and thereby improving convergence. To achieve this goal for one-time-step SNNs we present a novel *Hoyer spike layer* that sets the threshold based upon a *Hoyer regularized training process*, as described below.

3.1 HOYER SPIKE LAYER

In this work, we adopt a variant of the popular Leaky Integrate and Fire (LIF) representation that is illustrated in Eq. 1 to model the spiking neuron. Since we target one-time-step SNNs, u_l^{t-1} and λ can be dropped in equation 1, and the LIF variant can be simplified to

$$\boldsymbol{u}_l = \boldsymbol{w}_l \boldsymbol{o}_{l-1} \qquad \boldsymbol{z}_l = \frac{\boldsymbol{u}_l}{v_l^{th}} \qquad \boldsymbol{o}_l = \begin{cases} 1, & \text{if } \boldsymbol{z}_l \geq 1; \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

where $\boldsymbol{z}_l$ denotes the normalized membrane potential. Such a neuron model with a unit step activation function is difficult to optimize even with the recently proposed surrogate gradient descent techniques for multi-time-step SNNs Panda et al. (2020); Panda & Roy (2016), which either approximate the spiking neuron functionality with a continuous differentiable model or use surrogate gradients to approximate real gradients. This is because the average number of spikes with only one time step is too low to adjust the weights sufficiently using gradient descent with only one iteration available per input. This is because if a neuron does not emit a spike, the pre-synaptic weights connected to it cannot be updated. Therefore, it is crucial to reduce the value of the threshold to generate enough spikes for better network convergence. Note that a sufficiently low value of threshold can generate a spike for every neuron, but that would yield random outputs in the final classifier layer. Previous works Datta et al. (2021); Rathi et al. (2020a) show that the number of SNN time steps can be reduced by training the threshold term v_l^{th} using gradient descent. However, our experiments indicate that, for one time step SNNs, this approach still yields thresholds that produce significant drops in accuracy. In contrast, we propose to dynamically down-scale the threshold (see Fig. 1(a)) based on the membrane potential tensor using our proposed form of the Hoyer regularizer. In particular, we clip the membrane potential tensor corresponding to each convolutional layer to the trainable threshold v_l^{th} obtained from the gradient descent with our Hoyer loss, as detailed later in Eq. 12. Unlike existing approaches Datta & Beerel (2022); Rathi et al. (2020a) that require v_l^{th} to be initialized from a pre-trained non-spiking model, our approach can be used to train SNNs from scratch with a Kaiming uniform initialization He et al. (2015) for both the weights and thresholds. In particular, the down-scaled threshold value for each layer is computed as the Hoyer extremum of the clipped membrane potential tensor, as shown in Fig. 1(a) and mathematically defined as follows.

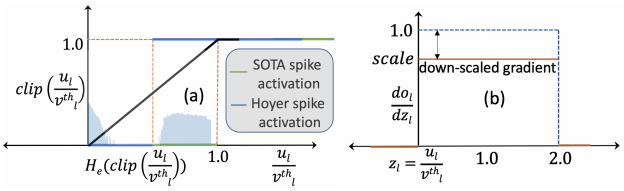

Figure 1: (a) Comparison of our Hoyer spike activation function with existing activation functions where the blue distribution denotes the shifting of the membrane potential away from the threshold using Hoyer regularized training, (b) Proposed derivative of our Hoyer activation function.

$$\boldsymbol{z}_l^{clip} = \begin{cases} 1, & \text{if } \boldsymbol{z}_l > 1 \\ \boldsymbol{z}_l, & \text{if } 0 \leq \boldsymbol{z}_l \leq 1 \\ 0, & \text{if } \boldsymbol{z}_l < 0 \end{cases} \qquad \boldsymbol{o}_l = h_s(\boldsymbol{z}_l) = \begin{cases} 1, & \text{if } \boldsymbol{z}_l \geq H_e(\boldsymbol{z}_l^{clip}); \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

Note that our threshold $H_e(\boldsymbol{z}_l^{clip})$ is indeed less than the trainable threshold v_l^{th} used in earlier works Datta & Beerel (2022); Rathi et al. (2020a) for any output, and the proof is shown in Appendix A.1.

Moreover, we observe that the Hoyer extremum in each layer changes only slightly during the later stages of training, which indicates that it is most likely an inherent attribute of the dataset and model architecture. As a result, in order to estimate the threshold during inference, we adopt an approach similar to batch normalization (BN), i.e., we calculate the exponential average of the Hoyer extremums during training, and use the same value during inference. This alleviates the need to recompute the Hoyer extremums for each input during inference.

3.2 HOYER REGULARIZED TRAINING

The loss function (L_{total}) of our proposed approach is shown below in Eq. 5.

$$L_{total} = L_{CE} + L_H = L_{CE} + \lambda_H \sum_{l=1}^{L-1} H(\boldsymbol{u}_l) \tag{5}$$

where L_{CE} denotes the cross-entropy loss calculated on the softmax output of the last layer L, and L_H represents the Hoyer regularizer calculated on the output of each convolutional and fully-connected layer, except the last layer. The weight update for the last layer is computed as

$$\Delta W_L = \frac{\partial L_{CE}}{\partial \boldsymbol{w}_L} + \frac{\partial L_H}{\partial \boldsymbol{w}_L} = \frac{\partial L_{CE}}{\partial \boldsymbol{u}_L}\frac{\partial \boldsymbol{u}_L}{\partial \boldsymbol{w}_L} + \frac{\partial L_H}{\partial \boldsymbol{u}_L}\frac{\partial \boldsymbol{u}_L}{\partial \boldsymbol{w}_L} = (\boldsymbol{s}-\boldsymbol{y})\boldsymbol{o}_{L-1} + \frac{\partial H(\boldsymbol{u}_L)}{\partial \boldsymbol{u}_L}\boldsymbol{o}_{L-1} \tag{6}$$

$$\frac{\partial L_{CE}}{\partial \boldsymbol{o}_{L-1}} = \frac{\partial L_{CE}}{\partial \boldsymbol{u}_L}\frac{\partial \boldsymbol{u}_L}{\partial \boldsymbol{o}_{L-1}} = (\boldsymbol{s}-\boldsymbol{y})\boldsymbol{w}_L \tag{7}$$

where $\boldsymbol{s}$ denotes the output softmax tensor, i.e., $s_i = \frac{e^{u_L^i}}{\sum_{k=1}^{N} u_L^k}$ where u_L^i and u_L^k denote the ith and k^{th} elements of the membrane potential of the last layer L, and N denotes the number of classes. Note that $\boldsymbol{y}$ denotes the one-hot encoded tensor of the true label, and $\frac{\partial H(\boldsymbol{u}_L)}{\partial \boldsymbol{u}_L}$ is computed using Eq. 2. The last layer does not have any threshold and hence does not emit any spike.

For a hidden layer l, the weight update is computed as

$$\Delta W_l = \frac{\partial L_{CE}}{\partial \boldsymbol{w}_l} + \frac{\partial L_H}{\partial \boldsymbol{w}_l} = \frac{\partial L_{CE}}{\partial \boldsymbol{o}_l}\frac{\partial \boldsymbol{o}_l}{\partial \boldsymbol{z}_l}\frac{\partial \boldsymbol{z}_l}{\partial \boldsymbol{u}_l}\frac{\partial \boldsymbol{u}_l}{\partial \boldsymbol{w}_l} + \frac{\partial L_H}{\partial \boldsymbol{u}_l}\frac{\partial \boldsymbol{u}_l}{\partial \boldsymbol{w}_l} = \frac{\partial L_{CE}}{\partial \boldsymbol{o}_l}\frac{\partial \boldsymbol{o}_l}{\partial \boldsymbol{z}_l}\frac{1}{v_l^{th}}\boldsymbol{o}_{l-1} + \frac{\partial L_H}{\partial \boldsymbol{u}_l}\boldsymbol{o}_{l-1} \tag{8}$$

where $\frac{\partial L_H}{\partial \boldsymbol{u}_l}$ can be computed as

$$\frac{\partial L_H}{\partial \boldsymbol{u}_l} = \frac{\partial L_H}{\partial \boldsymbol{u}_{l+1}}\frac{\partial \boldsymbol{u}_{l+1}}{\partial \boldsymbol{o}_l}\frac{\partial \boldsymbol{o}_l}{\partial \boldsymbol{z}_l}\frac{\partial \boldsymbol{z}_l}{\partial \boldsymbol{u}_l} + \frac{\partial H(\boldsymbol{u}_l)}{\partial \boldsymbol{u}_l} = \frac{\partial L_H}{\partial \boldsymbol{u}_{l+1}}\boldsymbol{w}_{l+1}\frac{\partial \boldsymbol{o}_l}{\partial \boldsymbol{z}_l}\frac{1}{v_l^{th}} + \frac{\partial H(\boldsymbol{u}_l)}{\partial \boldsymbol{u}_l} \tag{9}$$

where $\frac{\partial L_H}{\partial \boldsymbol{u}_{l+1}}$ is the gradient backpropagated from the $(l+1)^{th}$ layer, that is iteratively computed from the last layer L (see Eqs. 7 and 10). Note that for any hidden layer l, there are two gradients that contribute to the Hoyer loss with respect to the potential $\boldsymbol{u}_l$; one is from the subsequent layer $(l{+}1)$ and the other is directly from its Hoyer regularizer. Similarly, $\frac{\partial L_{CE}}{\partial \boldsymbol{o}_l}$ is computed iteratively, starting from the penultimate layer $(L{-}1)$ defined in Eq. 7, as follows.

$$\frac{\partial L_{CE}}{\partial \boldsymbol{o}_l} = \frac{\partial L_{CE}}{\partial \boldsymbol{o}_{l+1}}\frac{\partial \boldsymbol{o}_{l+1}}{\partial \boldsymbol{z}_{l+1}}\frac{\partial \boldsymbol{z}_{l+1}}{\partial \boldsymbol{u}_{l+1}}\frac{\partial \boldsymbol{u}_{l+1}}{\partial \boldsymbol{o}_l} = \frac{\partial L_{CE}}{\partial \boldsymbol{o}_{l+1}}\frac{\partial \boldsymbol{o}_{l+1}}{\partial \boldsymbol{z}_{l+1}}\frac{1}{v_l^{th}}\boldsymbol{w}_{l+1} \tag{10}$$

All the derivatives in Eq. 8-11 can be computed by Pytorch autograd, except the spike derivative $\frac{\partial \boldsymbol{o}_l}{\partial \boldsymbol{z}_l}$, whose gradient is zero almost everywhere and undefined at $o_l{=}0$. We extend the existing idea of surrogate gradient Neftci et al. (2019) to compute this derivative for one-time-step SNNs with Hoyer spike layers, as illustrated in Fig. 1(b) and mathematically defined as follows.

$$\frac{\partial \boldsymbol{o}_l}{\partial \boldsymbol{z}_l} = \begin{cases} \textit{scale}\times 1 & \text{if } 0 < \boldsymbol{z}_l < 2 \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

where *scale* denotes a hyperparameter that controls the dampening of the gradient. Finally, the threshold update for the hidden layer l is computed as

$$\Delta v_l^{th} = \frac{\partial L_{CE}}{\partial v_l^{th}} + \frac{\partial L_H}{\partial v_l^{th}} = \frac{\partial L_{CE}}{\partial \boldsymbol{o}_l}\frac{\partial \boldsymbol{o}_l}{\partial \boldsymbol{z}_l}\frac{\partial \boldsymbol{z}_l}{\partial v_l^{th}} + \frac{\partial L_H}{\partial v_l^{th}} = \frac{\partial L_{CE}}{\partial \boldsymbol{o}_l}\frac{\partial \boldsymbol{o}_l}{\partial \boldsymbol{z}_l}\frac{-\boldsymbol{u}_l}{(v_l^{th})^2} + \frac{\partial L_H}{\partial \boldsymbol{u}_{l+1}}\frac{\partial \boldsymbol{u}_{l+1}}{\partial v_l^{th}} \tag{12}$$

where $\frac{\partial \boldsymbol{u}_{l+1}}{\partial v_l^{th}}$ is computed as

$$\frac{\partial \boldsymbol{u}_{l+1}}{\partial v_l^{th}} = \frac{\partial \boldsymbol{u}_{l+1}}{\partial \boldsymbol{o}_l}\cdot\frac{\partial \boldsymbol{o}_l}{\partial v_l^{th}} = \boldsymbol{w}_{l+1}\cdot\frac{\partial \boldsymbol{o}_l}{\partial \boldsymbol{z}_l}\cdot\frac{-\boldsymbol{u}_l}{(v_l^{th})^2} \tag{13}$$

Note that we use this v_{th}^l, that is updated in each iteration, to estimate the threshold value in our spiking model using Eq. 4.

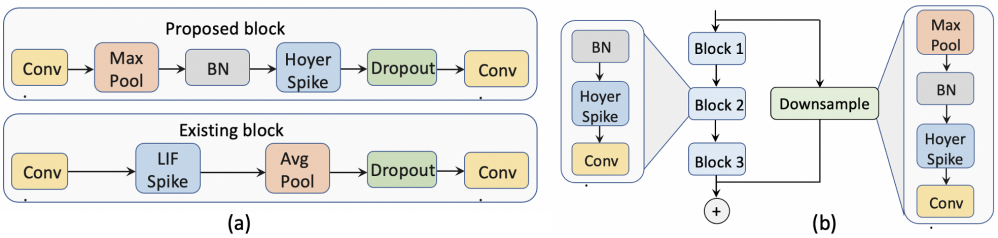

Figure 2: Spiking network architectures corresponding to (a) VGG and (b) ResNet based models.

3.3 Network Structure

We propose a series of network architectural modifications of existing SNNs Datta & Beerel (2022); Chowdhury et al. (2021); Rathi et al. (2020a) for our one-time-step models. As shown in Fig. 2(a), for the VGG variant, we use the max pooling layer immediately after the convolutional layer that is common in many BNN architectures Rastegari et al. (2016), and introduce the BN layer after max pooling. Note that unlike existing multi-time-step SNN architectures that avoid BN due to difficulty in convergence, we observe that BN helps increase the test accuracy with one time step. In contrast, for the ResNet variants, inspired by Liu et al. (2018), we observe models with shortcuts that bypass every block can also further improve the performance of the SNN. We also observe that the sequence of BN layer, Hoyer spike layer, and convolution layer outperforms the original bottleneck in ResNet. More details are shown in Fig. 2(b).

3.4 Possible Training Strategies

Based on existing SNN literature, we hypothesize a couple of training strategies that can used to train one-time-step SNNs, other than our proposed approach.

Table 1: Accuracies from different strategies to train one-step SNNs

Training Strategies	Pretrained DNN(%)	Acc. (%)	Sparsity (%)
Pre-trained+fine-tuning	93.15	91.39	23.56
Iterative training (N=10)	93.25	92.68	10.22
Iterative Training (N=20)	92.68	92.24	9.54
Proposed Training	-	**93.13**	22.57

Pre-trained DNN, followed by SNN fine-tuning. Similar to the hybrid training proposed in Rathi et al. (2020b), we pre-train a non-spiking DNN model, and copy its weights to the SNN model. Initialized with these weights, we train a one-time-step SNN with normal cross-entropy loss.

Iteratively convert ReLU neurons to spiking neurons. First we train a DNN model which uses the relu function with threshold as the activation function, then we iteratively reduce the number of the ReLU neurons whose output activation values are multi-bit. Specifically, we first force the neurons with values in the top N percentile to spike (set the output be 1), and those with bottom N percentile percent to die (set the output be 0), and gradually increase the N until there is a significant drop of accuracy or all neuron outputs are either 1 or 0.

Proposed training from scratch. With our proposed Hoyer spike layer and Hoyer loss, we train a SNN model from scratch.

Our results with these training strategies are shown in Table 1, which indicates that it is difficult for training strategies that involve pre-training and fine-tuning to approach the accuracy of non-spiking models with one time step. One possible reason for this might be the difference in the distribution of the pre-activation values between the DNN and SNN models Datta & Beerel (2022). It is also intuitive to obtain a one-time-step SNN model by iteratively reducing the proportion of the ReLU neurons from a pretrained full-precision DNN model. However, our results indicate that this method also fails to generate enough spikes at one time step required to yield close to SOTA accuracy. Finally, with our network structure modifications to existing SNN works, our Hoyer spike layer and our Hoyer regularizer, we can train a one-time-step SNN model with SOTA accuracy from scratch.

4 Experimental Results

Datasets & Models: Similar to existing SNN works Rathi et al. (2020b;a), we perform object recognition experiments on CIFAR10 Krizhevsky et al. (2009) and ImageNet Deng et al. (2009)

Table 2: Comparison of the test accuracy of our one-time-step SNN models with the non-spiking DNN models for object recognition. Model* indicates that we remove the first max pooling layer.

Network	dataset	DNN Top 1(%)	SNN Top 1 (%)	SNN Top 5 (%)	SNN Sparsity (%)
VGG16	CIFAR10	94.10	93.44	97.88	21.87
ResNet18	CIFAR10	93.34	91.48	97.34	25.83
ResNet18*	CIFAR10	94.28	93.67	97.98	16.12
ResNet20	CIFAR10	93.18	92.38	97.63	23.69
ResNet34*	CIFAR10	94.68	93.47	97.86	16.04
ResNet50*	CIFAR10	94.90	93.00	97.86	17.79
VGG16	ImageNet	70.08	68.00	78.75	24.48
ResNet50	ImageNet	73.12	66.32	77.06	23.89

dataset using VGG16 Simonyan & Zisserman (2014) and several variants of ResNet He et al. (2016) architectures. For object detection, we use the MMDetection framework Chen et al. (2019) with PASCAL VOC2007 and VOC2014 Everingham et al. (2010) as training dataset, and benchmark our SNN models and the baselines on the VOC2007 test dataset. We use the Faster RCNN Ren et al. (2015) and RetinaNet Lin et al. (2017) framework, and substitute the original backbone with our SNN models that are pretrained on ImageNet1K.

Object Recognition Results: For training the recognition models, we use the Adam Kingma & Ba (2014) optimizer for VGG16, and use SGD optimizer for ResNet models. As shown in Table 2, we obtain the SOTA accuracy of 93.44% on CIFAR10 with VGG16 with only one time step; the accuracy of our ResNet-based SNN models on ImageNet also surpasses the existing works. On ImageNet, we obtain a 68.00% top-1 accuracy with VGG16 which is only $\sim 2\%$ lower compared to the non-spiking counterpart. All our SNN models yield a sparsity of $\sim 25\%$ or lower on both CIFAR10 and ImageNet, which is significantly lower compared to the existing multi-time-step SNN models as shown in Fig. 3.

Object Detection Results: For object detection on VOC2007, we compare the performance obtained by our spiking models with non-spiking DNNs and BNNs in Table 3. For two-stage architectures, such as Faster R-CNN, the mAP of our one-time-step SNN models surpass the existing BNNs by $>0.6\%$[1]. For one-stage architectures, such as RetinaNet (chosen because of its SOTA performance), our one-time-step SNN models with a ResNet50 backbone yields a mAP of 70.5% (highest among existing BNN, SNN, AddNNs). Note that our spiking VGG and ResNet-based backbones lead to a significant drop in mAP with the YOLO framework that is more compatible with the DarkNet backbone (even existing DarkNet-based SNNs lead to very low mAP with YOLO as shown in Table 3). However, our models suffer $5.8-6.8\%$ drop in mAP compared to the non-spiking DNNs which may be due to the significant sparsity and loss in precision.

Table 3: Comparison of our one-time-step SNN models with non-spiking DNN, BNN, and multi-step SNN counterparts on VOC2007 test dataset.

Framework	Backbone	mAP(%)
Faster R-CNN	Original ResNet50	79.5
Faster R-CNN	Bi-Real Liu et al. (2018)	65.7
Faster R-CNN	ReActNet Liu et al. (2020)	73.1
Faster R-CNN	Our spiking ResNet50	73.7
Retinanet	Original ResNet50	77.3
Retinanet	SNN ResNet50 (ours)	70.5
YOLO	SNN DarkNet Kim et al. (2019)	53.01
SSD	BNN VGG16 Wang et al. (2020b)	66.0

Accuracy Comparison: We compare our results with various SOTA ultra low-latency SNNs for image recognition tasks in Table 4. Our one-time-step SNNs yield better test accuracy compared to all the existing works for both VGG and ResNet architectures, with significantly lower inference latency. The only exception for the latency reduction is the one-time-step SNN proposed in Chowdhury et al. (2021), however, it increases the training time significantly as illustrated later in Fig. 3. Other works that have training complexity similar or worse than ours, such as Datta & Beerel (2022) yields 1.78% lower accuracy with a $2\times$ more number of time steps. Furthermore, Zheng et al. (2021) achieves 0.41% lower accuracy compared to our method with $6\times$ more number of time steps. Across both CIFAR10 and ImageNet, our proposed training framework demonstrates 2-

[1]We were unable to find existing SNN works for two-stage object detection architectures.

Table 4: Comparison of our one-time-step SNN models to existing low-latency counterparts. SGD and hybrid denote surrogate gradient descent and pre-trained DNN followed by SNN fine-tuning respectively. (qC, dL) denotes an architecture with q convolutional and d linear layers.

Reference	Training	Architecture	Acc. (%)	Time steps
Dataset : CIFAR10				
Deng et al. (2021)	DNN-SNN conversion	VGG16	92.29	16
Wu et al. (2019)	SGD	5C, 2L	90.53	12
Kundu et al. (2021)	Hybrid	VGG16	92.74	10
Wu et al. (2021)	Tandem Learning	5C, 2L	90.98	8
Bu et al. (2022a)	DNN-SNN coonversion	VGG16	90.96	8
Zhang & Li (2020)	SGD	5C, 2L	91.41	5
Rathi et al. (2020a)	Hybrid	VGG16	92.70	5
Zheng et al. (2021)	STBP-tdBN	ResNet19	93.16	6
Datta & Beerel (2022)	Hybrid	VGG16	91.79	2
Bu et al. (2022b)	DNN-SNN conversion	VGG16	91.18	2
Chowdhury et al. (2021)	Hybrid	VGG16	93.05	1
Chowdhury et al. (2021)	Hybrid	ResNet20	91.10	1
This work	**Adam+Hoyer Reg.**	**VGG16**	**93.44**	**1**
Dataset : ImageNet				
Li et al. (2021b)	DNN-SNN conversion	VGG16	63.64	32
Bu et al. (2022b)	DNN-SNN conversion	ResNet34	59.35	16
Wu et al. (2021)	Tandem Learning	AlexNet	50.22	12
Rathi et al. (2020a)	Hybrid	VGG16	69.00	5
Zheng et al. (2021)	STBP-tdBN	ResNet34	67.05	6
Chowdhury et al. (2021)	Hybrid	VGG16	67.71	1
This work	**Adam+Hoyer Reg.**	**VGG16**	**68.00**	**1**

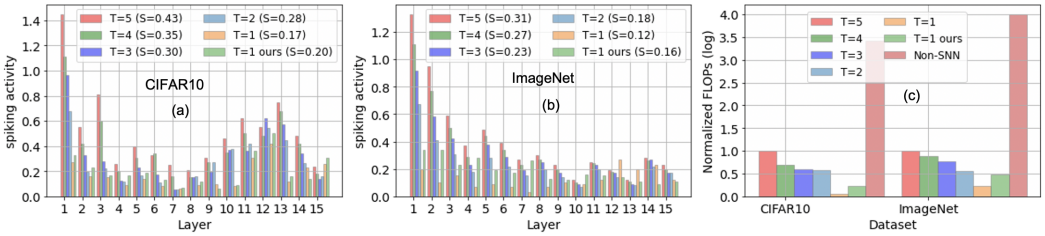

Figure 3: Layerwise spiking activities for a VGG16 across time steps ranging from 5 to 1 (average spiking activity denoted as S in parenthesis) representing existing low-latency SNNs including our work on (a) CIFAR10, (b) ImageNet, (c) Comparison of compute energy of between SNNs with different time steps and non-spiking DNNs.

$32\times$ improvement in inference latency with similar or worse training complexity compared to other works while yielding better test accuracy. Table 4 also demonstrates that the DNN-SNN conversion approaches require more time steps compared to our approach at worse test accuracies. Though they do not involve SNN training, the DNN pre-training requires identical training time per epoch as our one-step SNNs shown in Fig. 4.

Inference Efficiency: We compare the energy-efficiency of our one-time-step SNNs with non-spiking DNNs and existing multi-time-step SNNs in Fig. 3. The compute-efficiency of SNNs stems from two factors:- 1) sparsity, that reduces the number of floating point operations compared to non-spiking DNNs according to $SNN_l^{flops} = S_l \times DNN_l^{flops}$ Chowdhury et al. (2021), where S_l denotes the average number of spikes per neuron per inference over all timesteps in layer l. 2) Use of only AC (0.9pJ) operations that are each $5.1\times$ lower compared to each MAC (4.6pJ) operation in 45nm CMOS technology Horowitz (2014) for floating-point representation. Hence, the total compute energy of a multi-time-step SNN (SNN_{CE}) and non-spiking DNN (DNN_{CE}) can be estimated as

$$SNN_{CE}{=}DNN_1^{flops}{*}4.6{+}\sum_{l=2}^{L} S_l * DNN_l^{flops}{*}0.9 \quad DNN_{CE}{=}\sum_{l=1}^{L} DNN_l^{flops}{*}4.6 \tag{14}$$

because the direct encoded SNN receives analog input in the first layer (l=1) without any sparsity Chowdhury et al. (2021); Datta & Beerel (2022); Rathi et al. (2020a). We compare the layer-wise spiking activities that represent the sparsity S_l for time steps ranging from 5 to 1 in Fig. 3 that represent existing low-latency SNN works, including our work. Note, the spike rates decrease significantly with time step reduction from 5 to 1, leading to considerably lower FLOPs in our one-time-step SNNs. These lower FLOPs, coupled with the $5.1\times$ reduction for AC operations leads to a $24.34\times$ and $34.21\times$ reduction in energy on CIFAR10 and ImageNet respectively with VGG16. Though we focus on compute energies for our comparison, multi-time-step SNNs also incur a large number of memory accesses as the membrane potentials and weights need to be fetched from and read to the on-/off-chip memory for each time step. Our one-time-step models can avoid these repetitive read/write operations as it does involve any *state* and lead to a $\sim T\times$ reduction in the number of memory accesses compared to a T-time-step SNN model. Our one-time-step SNN models may also incur lower memory cost even compared to non-spiking DNNs, with the support of zero gating logic that can leverage the sparsity in digital DNN accelerators. However, the exact savings in memory cost will depend on the data reuse and underlying hardware Chen et al. (2016).

Training & Inference Time Requirements: Because SOTA SNNs require iteration over multiple time steps and storage of the membrane potentials for each neuron, their training and inference time can be substantially higher than their DNN counterparts. However, reducing their latency to 1 time step can bridge this gap significantly, as shown in Figure 4. On average, our low-latency, one-time-step SNNs represent a $2.38\times$ and $2.33\times$ reduction in training and inference time per epoch respectively, compared to the multi-time-step training approaches Datta & Beerel (2022); Rathi et al. (2020a) which represents the SOTA in latency (without iterative training as done in Chowdhury et al. (2021)), with iso-batch and hardware conditions. Compared to the existing one-time-step SNNs Chowdhury et al. (2021), we yield a $19\times$ and $1.25\times$ reduction in training and inference time. Such significant savings in training time, that translates to power savings in big data centers, can potentially reduce AI's environmental impact.

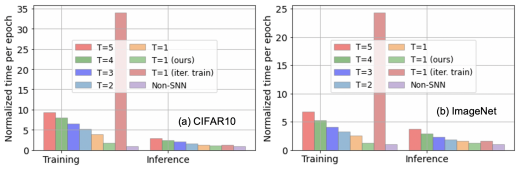

Figure 4: Normalized training and inference time per epoch with iso-batch (256) and hardware (RTX 3090 with 24 GB memory) conditions for (a) CIFAR10 and (b) ImageNet with VGG16.

Ablation Studies: We conduct ablation studies to analyze the contribution of each technique in our proposed approach. For fairness, we train all the ablated models on CIFAR10 dataset for 400 epochs, and use Adam as the optimizer, with 0.0001 as the initial learning rate. Our results are shown in Table 5, where the model without Hoyer spike layer indicates that we set the threshold as v_l^{th} similar to existing works Datta & Beerel (2022); Rathi et al. (2020a) rather than our proposed Hoyer extremum. With VGG16, our optimal network modifications lead to a 1.9% increase in accuracy. Furthermore, adding only the Hoyer regularizer leads to negligible accuracy and sparsity improvements. This might be because the regularizer alone may not be able to sufficiently down-scale the threshold for optimal convergence with one time step. However, with our Hoyer spike layer, the accuracy improves by 2.68% to 93.13% while also yielding a 2.09% increase in sparsity. We observe a similar trend for our network modifications and Hoyer spike layer with ResNet18. However, Hoyer regularizer substantially reduces the sparsity from 27.62% to 20.50%, while also negligibly reducing the accuracy. In summary, while our network modifications significantly increase the test accuracy compared to the SOTA SNN training with one time step, the combination of our Hoyer regularizer and Hoyer spike layer yield the SOTA SNN performance.

Table 5: Ablation study of the different methods in our proposed training framework.

Arch.	Network Structure	Hoyer Reg.	Hoyer Spike	Acc. (%)	Sparsity (%)
VGG16	×	×	×	88.42	15.62
VGG16	✓	×	×	90.33	20.43
VGG16	✓	✓	×	90.45	20.48
VGG16	✓	✓	✓	**93.13**	22.57
ResNet18	×	×	×	87.41	22.78
ResNet18	✓	×	×	91.08	27.62
ResNet18	✓	✓	×	90.95	20.50
ResNet18	✓	✓	✓	**91.48**	25.83

Effect on Quantization: In order to further improve the compute-efficiency of our one-time-step SNNs, we perform quantization-aware training of the weights in our models to $2-6$ bits. This transforms the full-precision ACs to $2-6$ bit ACs, thereby leading to a $4.8-15.2$ reduction in compute energy as obtained from FPGA simulations. Note that we only quantize the convolutional layers, as quantizing the linear layers lead to a noticeable drop in accuracy. From the results shown in Table 6, when quantized to 6 bits, our one-time-step VGG-based SNN incur a negligible accuracy drop of only 0.02%. Even with 2-bit quantization, our model can yield an accuracy of 92.34% with any special modification, while still yielding a sparsity of $\sim 22\%$.

Table 6: Accuracies of weight quantized one-time-step SNN models based on VGG16 where FP is 32-bit floating point.

Bits	Acc. (%)	Sparsity (%)	CE (mJ)
FP	93.13	22.57	29742.3
6	93.11	22.46	61.9
4	92.84	21.39	39.4
2	92.34	22.68	21.6

Comparison with AddNNs & BNNs: We compare the accuracy of our one-time-step SNN models with recently proposed AddNN models Chen et al. (2020) that also removes multiplications for increased energy-efficiency in Table 7. With the VGG16 architecture, on CIFAR10, we obtain 0.6% lower accuracy, while on ImageNet, we obtain 1.0% higher accuracy. Moreover, unlike SNNs, AddNNs do not involve any sparsity, and hence, consume $\sim 5.5\times$ more energy compared to our SNN models on average across both CIFAR10 and ImageNet (see Table 7). We also compare our SNN models with SOTA BNNs in Table 7 that replaces the costly MAC operations with cheaper pop-count counterparts, thanks to the binary weights and activations. Note that there are two kinds of BNNs, one that has 0 and 1 as two states (uni-polar), and the other that has 1 and -1 (bi-polar) as two states. Most recent BNN models that have yielded accuracy almost similar to the full-precision DNNs fall under the latter category. However, they can not yield any sparsity (even full-precision DNNs yield sparsity due to ReLU) unlike the uni-polar BNNs and SNNs. Both our full-precision and quantized one-time-step SNN models yield accuracies higher than bi-polar BNNs at iso-architectures on both CIFAR10 and ImageNet. Additionally, our quantized SNN models (2-6 bits) also consume $1.1-3.4\times$ lower energy compared to the bi-polar networks due to the improved trade-off between the high sparsity ($\sim 22\%$ as shown in Table 7) provided by our one-time-step SNN models, and less energy due to XOR operations compared to quantized (2-6 bit) ACs.

Table 7: Comparison of our one-time-step SNN models to AddNNs and BNNs that also incur AC-only operations for improved energy-efficiency, where CE denotes compute energy

Reference	Dataset	Acc. (%)
BNNs (CE = 73.44 mJ for CIFAR10 on VGG16		
Sakr et al. (2018)	CIFAR10	89.6
Wang et al. (2020a)	CIFAR10	90.2
Wang et al. (2020a)	ImageNet	59.7
Diffenderfer & Kailkhura (2021)	CIFAR10	91.9
AddNNs (CE = 163.582 J for CIFAR10 on VGG16)		
Chen et al. (2020)	CIFAR10	93.72
Chen et al. (2020)	ImageNet	67.0
Li et al. (2021a)	CIFAR10	91.56
Our SNNs (CE=21.6 mJ with 2-bits for CIFAR10 on VGG16)		
This work	CIFAR10	93.44
This work	ImageNet	68.00

5 Discussions & Future Impact

Existing SNN training works choose ANN-SNN conversion methods to yield high accuracy or SNN fine-tuning to yield low latency or a hybrid of both for a balanced accuracy-latency trade-off. However, none of the existing works can discard the temporal dimension completely, which can enable the deployment of SNN models in multiple real-time applications, without significantly increasing the training cost. To the best of our knowledge, our paper for the first time presents a SNN training framework from scratch involving a novel combination of a Hoyer regularizer and Hoyer spike layer for one time step. Our SNN models incur similar training time as non-spiking DNN models and achieve SOTA accuracy, outperforming the existing SNN, BNN, and AddNN models. Additionally, our models yield a $24.34-34.21\times$ reduction in energy compared to standard DNNs, that can enable the deployment of our models on power-constrained devices. However, our work can also enable cheap and real-time computer vision systems that might be susceptible to adversarial attacks. Preventing the application of this technology from abusive usages is an important and interesting area of future work.

REFERENCES

Tong Bu, Jianhao Ding, Zhaofei Yu, and Tiejun Huang. Optimized potential initialization for low-latency spiking neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(1):11–20, Jun. 2022a. doi: 10.1609/aaai.v36i1.19874. URL https://ojs.aaai.org/index.php/AAAI/article/view/19874.

Tong Bu, Wei Fang, Jianhao Ding, PENGLIN DAI, Zhaofei Yu, and Tiejun Huang. Optimal ANN-SNN conversion for high-accuracy and ultra-low-latency spiking neural networks. In *International Conference on Learning Representations*, 2022b. URL https://openreview.net/forum?id=7B3IJMM1k_M.

Yongqiang Cao et al. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113:54–66, 05 2015.

Hanting Chen, Yunhe Wang, Chunjing Xu, Boxin Shi, Chao Xu, Qi Tian, and Chang Xu. Addernet: Do we really need multiplications in deep learning? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open MMLab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019.

Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ACM SIGARCH Computer Architecture News*, volume 44, 06 2016.

Sayeed Shafayet Chowdhury, Nitin Rathi, and Kaushik Roy. One timestep is all you need: Training spiking neural networks with ultra low latency. *arXiv preprint arXiv:2110.05929*, 2021.

I. M. Comsa et al. Temporal coding in spiking neural networks with alpha synaptic function. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 1, pp. 8529–8533, 2020.

Gourav Datta and Peter A. Beerel. Can deep neural networks be converted to ultra low-latency spiking neural networks? In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, volume 1, pp. 718–723, 2022. doi: 10.23919/DATE54114.2022.9774704.

Gourav Datta et al. Training energy-efficient deep spiking neural networks with single-spike hybrid input encoding. In *2021 International Joint Conference on Neural Networks (IJCNN)*, volume 1, pp. 1–8, 2021. doi: 10.1109/IJCNN52387.2021.9534306.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Shikuang Deng et al. Optimal conversion of conventional artificial neural networks to spiking neural networks. In *International Conference on Learning Representations*, 2021.

Peter U Diehl et al. Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pp. 1–8. IEEE, 2016.

James Diffenderfer and Bhavya Kailkhura. Multi-prize lottery ticket hypothesis: Finding accurate binary neural networks by pruning a randomly weighted network. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=U_mat0b9iv.

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv preprint arXiv:1502.01852*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

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

Patrik O Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of machine learning research*, 5(9), 2004.

Giacomo Indiveri et al. Frontiers in neuromorphic engineering. *Frontiers in Neuroscience*, 5, 2011.

Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, Simon J. Thorpe, and Timothée Masquelier. STDP-based spiking deep convolutional neural networks for object recognition. *Neural Networks*, 99:56–67, Mar 2018. ISSN 0893-6080. doi: 10.1016/j.neunet.2017.12.005. URL `http://dx.doi.org/10.1016/j.neunet.2017.12.005`.

Seijoon Kim, Seongsik Park, Byunggook Na, and Sungroh Yoon. Spiking-yolo: Spiking neural network for energy-efficient object detection, 2019.

Youngeun Kim et al. Revisiting batch normalization for training low-latency deep spiking neural networks from scratch. *arXiv preprint arXiv:2010.01729*, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Souvik Kundu et al. Towards low-latency energy-efficient deep SNNs via attention-guided compression. *arXiv preprint arXiv:2107.12445*, 2021.

Mark Kurtz, Justin Kopinsky, Rati Gelashvili, Alexander Matveev, John Carr, Michael Goin, William Leiserson, Sage Moore, Nir Shavit, and Dan Alistarh. Inducing and exploiting activation sparsity for fast inference on deep neural networks. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5533–5543. PMLR, 13–18 Jul 2020. URL `https://proceedings.mlr.press/v119/kurtz20a.html`.

Jun Haeng Lee et al. Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, 10, 2016.

Wenshuo Li, Hanting Chen, Mingqiang Huang, Xinghao Chen, Chunjing Xu, and Yunhe Wang. Winograd algorithm for addernet. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 6307–6315. PMLR, 18–24 Jul 2021a. URL `https://proceedings.mlr.press/v139/li21c.html`.

Yuhang Li et al. A free lunch from ANN: Towards efficient, accurate spiking neural networks calibration. *arXiv preprint arXiv:2106.06984*, 2021b.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002*, 2017.

Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 722–737, 2018.

Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. Reactnet: Towards precise binary neural network with generalized activation functions. In *European conference on computer vision*, pp. 143–159. Springer, 2020.

E. O. Neftci, H. Mostafa, and F. Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019.

Priyadarshini Panda and Kaushik Roy. Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition. *arXiv preprint arXiv:1602.01510*, 2016.

Priyadarshini Panda et al. Toward scalable, efficient, and accurate deep spiking neural networks with backward residual connections, stochastic softmax, and hybridization. *Frontiers in Neuroscience*, 14, 2020.

Michael Pfeiffer et al. Deep learning with spiking neurons: Opportunities and challenges. *Frontiers in Neuroscience*, 12:774, 2018.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pp. 525–542. Springer, 2016.

Nitin Rathi et al. DIET-SNN: Direct input encoding with leakage and threshold optimization in deep spiking neural networks. *arXiv preprint arXiv:2008.03658*, 2020a.

Nitin Rathi et al. Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. *arXiv preprint arXiv:2005.01807*, 2020b.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

Charbel Sakr, Jungwook Choi, Zhuo Wang, Kailash Gopalakrishnan, and Naresh Shanbhag. True gradient-based training of deep binary activated neural networks via continuous binarization. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 1, pp. 2346–2350, 2018. doi: 10.1109/ICASSP.2018.8461456.

Abhronil Sengupta et al. Going deeper in spiking neural networks: VGG and residual architectures. *Frontiers in Neuroscience*, 13:95, 2019.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Peisong Wang, Xiangyu He, Gang Li, Tianli Zhao, and Jian Cheng. Sparsity-inducing binarized neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34 (07):12192–12199, Apr. 2020a. URL https://ojs.aaai.org/index.php/AAAI/article/view/6900.

Ziwei Wang, Ziyi Wu, Jiwen Lu, and Jie Zhou. Bidet: An efficient binarized object detector. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020b.

Jibin Wu, Yansong Chua, Malu Zhang, Guoqi Li, Haizhou Li, and Kay Chen Tan. A tandem learning rule for effective training and rapid inference of deep spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 1(1):1–15, 2021. doi: 10.1109/TNNLS.2021.3095724.

Yujie Wu et al. Direct training for spiking neural networks: Faster, larger, better. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1311–1318, 2019.

Huanrui Yang, Wei Wen, and Hai Li. Deephoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures. In *International Conference on Learning Representations*, 2020.

Wenrui Zhang and Peng Li. Temporal spike sequence learning via backpropagation for deep spiking neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 12022–12033. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/8bdb5058376143fa358981954e7626b8-Paper.pdf.

Hanle Zheng, Yujie Wu, Lei Deng, Yifan Hu, and Guoqi Li. Going deeper with directly-trained larger spiking neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35 (12):11062–11070, May 2021. doi: 10.1609/aaai.v35i12.17320. URL https://ojs.aaai.org/index.php/AAAI/article/view/17320.

A APPENDIX

A.1 PROOF FOR THE DOWN-SCALING OF OUR THRESHOLD

In order to prove that our Hoyer extremum is always less than or equal to v^{th}, we first prove the Hoyer extremum of $\boldsymbol{z}_l^{clip}$ is less than or equal to 1. Let us use $\boldsymbol{c}_l$ to represent $\boldsymbol{z}_l^{clip}$, so $\forall j, 0 \leq c_l^j \leq 1$

$$H_e(\boldsymbol{c}_l) = \frac{\|\boldsymbol{c}_l\|_2^2}{\|\boldsymbol{c}_l\|_1} = \frac{\sum_j (c_l^j)^2}{\sum_j c_l^j} \leq \frac{\sum_j (c_l^j \cdot max(c_l^j))}{\sum_j c_l^j} \leq max(c_l^j)) \leq 1 \tag{15}$$

So the Hoyer extremum of $\boldsymbol{z}_l^{clip}$ is always less than or equal one, and our Hoyer extremum of every layer l which is the product of v_l^{th} and $H_e(\boldsymbol{z}_l^{clip})$ is always less than or equal v_l^{th}.

A.2 EXPERIMENTAL SETUP

For training VGG16 models, we using Adam optimizer with initial learning rate of 0.0001, weight decay of 0.0001, dropout of 0.1 and batch size of 128 in CIFAR10 for 600 epochs, and Adam optimizer with weight decay of $5e^{-6}$ and with batch size 64 in ImageNet for 180 epochs. For training ResNet models, we using SGD optimizer with initial learning rate of 0.1, weight decay of 0.0001 and batch size of 128 in CIFAR10 for 400 epochs, and Adam optimizer with weight decay of $5e^{-6}$ and with batch size 64 in ImageNet for 120 epochs. Depending on the number of epochs, we divide the learning rate by 5 at [0.6, 0.8, 0.9]* max num of epoch.

When calculating the Hoyer extremum we implement two versions, one that calculates the Hoyer extremum for the whole batch, while another that calculates it channel-wise. Our experiments show that using the channel-wise version can bring $0.1{-}0.3\%$ increase in accuracy. All the experimental results reported in this paper use this channel-wise version.

For Faster RCNN, we use SGD optimizer with initial learning rate of 0.01 for 50 epoch. For Retina, we use SGD optimizer with initial learning rate of 0.001 for 50 epoch

A.3 TRAINING ALGORITHM

Our training algorithm can be summarized as follows:

Algorithm 1: Detailed Algorithm for training our one-time-step SNN model.

Input: runEpochs, numBatches, numLayers, initial weights $\mathbf{W}$, initial thresholds $\boldsymbol{v}^{th}$, Training data $\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^{N}$, Hoyer regularizer coefficient λ_H.
Data: Hoyer extremum $H_e = 0$, Layer index $l = 0$

```
for i ← 0 to runEpochs do
    for j ← 0 to numBatches do
        output ← x
        for l ← 0 to numLayers do
            if layer_l is Hoyer Spike layer then
                u_l ← output
                L_H ← L_H + H(u_l)
                z_l ← u_l / v_l^th;                                  // Divide input by threshold
                H_e ← computeExponentialMovingAverage(H_e, H_e(clamp(z_l, min = 0, max = 1)))
                o_l[z_l ≥ H_e] ← 1, o_l[z_l < H_e] ← 0;   // Output spiking activation map
                output ← o_l
            end
            else
                output = layer_l(output)
            end
        end
        L = L_CE + λ_H * L_H
        ∂L/∂W = computeGradients(W, L)
        ∂L/∂v^th = computeGradients(v^th, L)
        updateWeightsAndThresholds(∂L/∂W, ∂L/∂v^th)
    end
    for l ← 0 numLayers do
        if layer_l is Hoyer Spike layer then
            u_l ← output
            z_l ← u_l / v_l^th
            o_l[z_l ≥ H_e] ← 1, o_l[z_l < H_e] ← 0;
            output ← o_l
        end
        else
            output = layer_l(output)
        end
    end
end
```