TEAS: EXPLOITING SPIKING ACTIVITY FOR TEMPORAL-WISE ADAPTIVE SPIKING NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Spiking neural networks (SNNs) are energy-efficient alternatives to commonly used deep artificial neural networks (ANNs). However, their sequential computation pattern over multiple time steps makes processing latency a significant hindrance to deployment. In existing SNNs deployed on time-driven hardware, all layers generate and receive spikes in a synchronized manner, forcing them to share the same time steps. This often leads to considerable time redundancy in the spike sequences and considerable repetitive processing. Motivated by the effectiveness of dynamic neural networks for boosting efficiency, we propose a temporal-wise adaptive SNN, namely TEAS, in which each layer is configured with independent number of time steps to fully exploit the potential of SNNs. Specifically, given an SNN, the number of time steps of each layer is configured according to its contribution to the final performance of the whole network. Then, we exploit the temporal transforming module to produce a dynamic policy that can adapt the temporal information dynamically during inference. The adaptive configuration generating process also enables the trading-off between model complexity and accuracy. Extensive experiments on a variety of challenging datasets demonstrate that our method provides significant savings in energy efficiency and processing latency under similar accuracy outperforming the existing state-of-the-art methods.

1 INTRODUCTION

Artificial Neural Networks (ANNs) have achieved phenomenal successes in multiple domains, such as object recognition, natural language processing, self-driving, etc Srinivasan & Roy (2019); Deng et al. (2020). However, rapid evolution of this field brings about a momentum of dramatic growth in computation. State-of-the-art ANN models with billions of parameters consumes such a huge amount of computation that renders on-device inference challenging. For instance, surprisingly powerful GPT-3 Brown et al. (2020), an autoregressive language model with 175 billion parameters, consume 740,000 GFLOPS (Floating Point Operations) of computation for a single inference. Such situation stimulates the demand for deploying more power-efficient neural networks in real-world applications Rathi et al. (2021).

Recently, extensive efforts have been attracted by spiking neural networks (SNNs) due to their low-power and biologically plausible nature. Different from ANNs, SNNs transmit information with spike trains, which extends in the additional temporal dimension and contains only sparse binary spike events. This leads to more power-efficient computation by substituting expensive multiplication with addition operations in neuromorphic hardware Lee et al. (2021).

In fact, the inference process for SNNs is quite different from that of ANNs: ANNs infer in a single-shot forward pass, whereas SNNs require computation over multiple time steps. The requirement of the multi-shot forward pass in SNNs causes repetitive spike generation across the time window, resulting in higher inference latency and energy consumption. Currently, the information transmitted in SNNs is conventionally encoded as an identical-sized time dimension, that is, the globally unified time steps allocation Chen et al. (2021); Deng & Gu (2021). An excessively small time window will lose important information, resulting in a significant accuracy loss. To achieve better perfor-

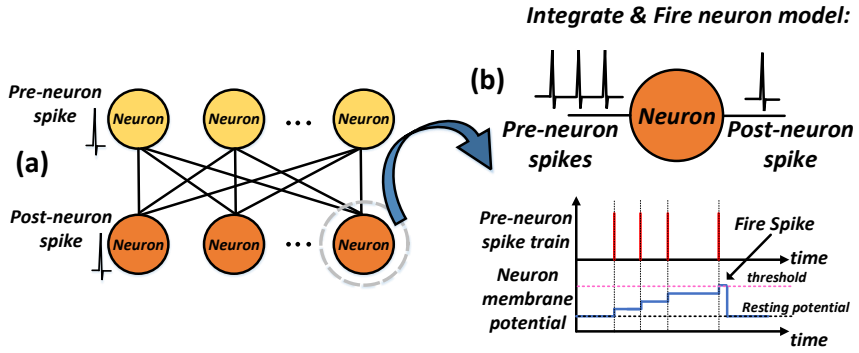


Figure 1: Illustration of Linear-Integrate and Fire (IF) neuron dynamics. (a) A typical fully connected neural network architecture for SNN; (b) The membrane potential of IF neuron increases with the accumulation of weighted spikes.

mance, SNNs generally employ a large number of time steps for each layer, which imposes three challenges to edge deployment under the current time-driven synchronized mechanism, as pointed out in Narayanan et al. (2020); Singh et al. (2020): (i) long latency that places hurdles for real-time applications; (ii) low energy efficiency induced by massive spike accumulation operations across time steps; (iii) expensive overhead of memory storage and access for the intermediate values. As a result, reducing the number of time steps while maintaining performance is a critical research problem in SNNs. However, the existing SNNs fail to dynamically transform the temporal dimension in the inference, so that these works end up missing the chance in further reducing the power and improving accuracy.

Therefore, we are motivated to apply different time steps of the spike train for layers with different spiking activities to achieve SNNs with adaptive temporal information. Although adaptive temporal information of SNNs looks trivial and handy at the first glance, we need to address two critical challenges: 1) how to efficiently determine what time steps to use per layer, and 2) given layer-wise time steps, how can we flexibly and efficiently convert the spike train between the adjacent layers into various time steps at runtime, without loss of information.

The insight enabling this idea is that the average spiking rate varies greatly among different layers, so the best temporal configuration for each layer can be pre-determined. To this end, we develop a novel adaptive spike train conversion methodology, termed as TEAS. In the offline phase, TEAS takes a description of SNN layers and explores the space of possible temporal configurations based on the spiking activity to construct an SNN with the best per-layer configurations, while minimizing the accuracy loss. Then, in the execution phase, temporal configuration enforces the SNNs to dynamically optimize processing latency and energy efficiency.

In summary, this paper makes the following contributions:

- We discover that the spiking activities in layers are highly correlated with the SNN accuracy, termed as vulnerable layers. This observation implies that applying the high-fidelity temporal dimension on the vulnerable layers will reserve the SNN accuracy, while allowing the salient information to flow the invulnerable layers can boost the performance.
- We propose TEAS to enable adaptive temporal dimensions in the SNN, a new dynamic spike train conversion methodology. It is the first approach that adjusts informative features in the temporal dimension at run-time to improve the computation efficiency, enabling trade-off the accuracy and computation more flexibly.
- We evaluate the performance of TEAS with deep network architectures on MNIST, CIFAR-10/100, and ImageNet datasets. The proposed TEAS exceeds state-of-the-art accuracy on all tested datasets, using fewer time steps than other SNNs with unified time windows.

2 Preliminaries and Motivation

Spiking Neural Network. The spiking neuron is the fundamental computing unit of SNNs. Fig. 1 illustrates the execution process of SNN neurons. An IF neuron receives train of spikes, over certain time window. Input spikes (‘0’ or ‘1’ spike signals) from pre-synaptic neurons multiply the synaptic weight, and consequently, the products accumulate to the potential of the post-synaptic neuron Roy

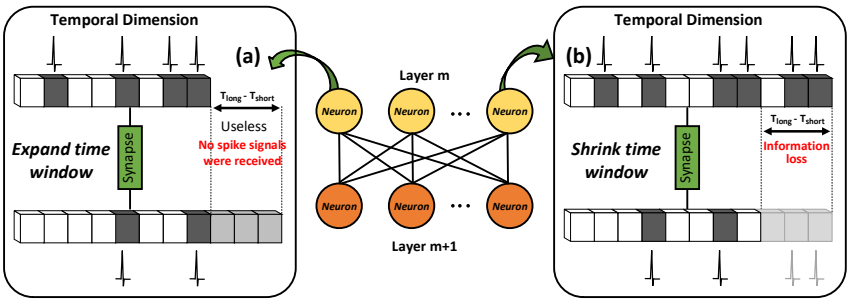


Figure 2: An example of temporal alignment. (a) Expanding the time window; (b) Shrinking the time window.

et al. (2019); Tavanaei & Maida (2019). The spiking neuron dynamics are described by:

$$\mathbf{V}(t+1) = \mathbf{V}(t) + \sum_{i \in \{i | X_i(t)=1\}} w_i \tag{1}$$

wherein, $\mathbf{V}(t)$ is the neuron membrane potential, $X_i(t) = 1$ means that a spike arrives at time step t from the i-th pre-synaptic neuron, and w_i is the weight associated with these inputs. The neuron *fires* a spike when its membrane potential reaches the firing threshold V_{th}, and then resets membrane potential to the reset potential V_{reset}.

We refer to rate-encoded and temporally-encoded SNNs as SNN-R and SNN-T respectively. Rate-encoding converts an input pixel value into the spike train consisting of multiple spikes, representing the average number of spikes fired by the neuron in a given time window Han et al. (2020). In contrast to rate-encoding, temporal-encoding represents information through the latency to the first spike of the corresponding spike train over a given time window, which converts an input pixel value into the spike train consisting of a single spike Roy et al. (2019).

Limitations of previous works. However, these previous SNNs are developed based on the temporal alignment for the network (i.e., all the layers of the SNN share the unified time window), where we believe such SNNs may be redundant and inefficient.

Suppose the previous layer uses a short time window T_{short}, thus the transmitted information is stuck at the last time step (shown in Fig. 2(a)). Not only will the information out of the time window not be transmitted, but also the information already transmitted from the earlier time steps (i.e., accumulated membrane potential, but not yet fired spikes) will be lost together. Even if the following layer uses a long time window T_{long}, the informative features from the previous layers are only terminated at T_{short}, while $T_{short}+1 \sim T_{long}$ is useless without any information. In that case, directly shortening the time window is extremely easy to cause the degradation of SNN accuracy. Fig. 2(b) shows an example of shrinking the time window. In effect, it causes the information loss resulted from ignoring the information transmitted beyond the time window of the post-neuron. Therefore, the unified temporal dimension incurs redundant overhead for reserving better SNN accuracy, leading to significant latency and energy consumption opposite to the original purpose.

Why we need temporal-wise adaptive SNNs. The key to achieving efficient SNNs in trading-off the processing efficiency (i.e., the temporal dimension) and accuracy is to produce lean temporal-wise informative features matching each layer, thus enhancing the overall performance of the SNN without compromising the representation power of networks. It is noteworthy that the adaptive time window poses a bigger challenge for SNNs since it requires aligning the unequal temporal dimensions between different layers in the information propagation Roy et al. (2019); Deng et al. (2021).

3 Approach

In this section, we first study the existence of vulnerable layers that affect the SNN performance and to what degree, and then we present the adaptive conversion employed in the temporal dimension. Finally, we describe the optimization procedure and implementation details for the proposed TEAS.

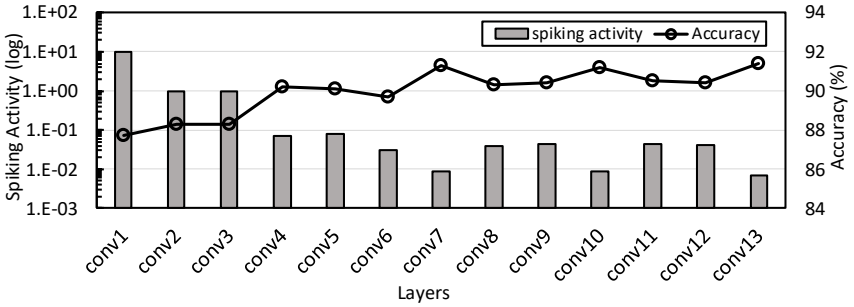


Figure 3: Spiking activity of each layer and accuracy of SNN at different layers bearing the input noise ($u = 0.1$ that represents the magnitude of noise). The SNN with VGG-16 architecture trained on CIFAR-10.

3.1 Identifying Vulnerable Layers in the SNN

Because shortening the time steps of the spike train will exert an impact on the SNN accuracy as revealed by prior studies Fang et al. (2021); Han & Roy (2020); Han et al. (2020), two questions naturally arise: 1) whether different layers have different degrees of sensitivity to the reduced time steps and 2) how to utilize the difference to maintain the accuracy while shortening the time step.

To study the two questions in this section, we first verify that the spike train of layers does affect the accuracy, and they are termed as vulnerable layers. With just a little noise added, these vulnerable layers will have a great impact on the prediction accuracy. This means these layers need more temporal information to precisely deliver information to ensure the overall SNN accuracy. In addition, we discover that these layers tend to have higher spiking activity (in Eq. (2)). As a result, vulnerable layers should be carefully identified and allocated more temporal information to reserve more detailed characteristics for the SNN, while there is no such need for other invulnerable ones. A natural implication is that we should take advantage of the invulnerable to speed up the SNN by differentiating time steps on different layers with almost no accuracy loss.

$$\#Act(l) = \frac{\#\text{fired spikes}}{\#\text{time steps}} \tag{2}$$

Fig. 3 gives the results, where the curve labeled circle represents network accuracy with noise being added only to the certain layer, while bar represents the spiking activity (i.e., the average number of spikes over the time window) of the layer. From the plot, a critical observation can be made: when we add noise to the certain layer, the curve drops more rapidly at the layer with larger spiking activity. This observation clearly manifests that different layers may have different impacts on the final accuracy, and it is a popular observation across different networks we have investigated. In other words, we can retain the SNN accuracy for a given dataset by restricting the informative impact in the temporal dimension (equivalent to noise) for the vulnerable layers, while time steps for the invulnerable layers can be relaxed.

3.2 Adaptive Conversion in the Temporal Dimension

Based on this observation, we need to address the following question: how to conduct the spike train scaling at runtime? Layers with different magnitudes of vulnerability may introduce spike trains with different time steps. This intra-layer conversion in the temporal dimension needs to be efficient to reduce the neuron computation workload and benefit the inference performance on the fly. The overview of the proposed adaptive conversion as shown in 4.

According to the method to translate the information contained in a spike train, SNNs can be categorized into two classes Roy et al. (2019). SNN-Rs sort spike trains according to the number of spikes, and spike trains with more spikes are regarded as carrying stronger information. SNN-Ts sort spike trains according to the firing time of the only spike in the series, and an earlier spike means that the spike train is carrying stronger information.

When the number of time steps of two layers are the same, then we can directly apply the output spike train of the former layer as the input spike train of the later layer. However, if the number of time steps of two layers are different, denoted as T^l and T^{l+1} respectively, then we have found a transformation from a spike train of length T^l into a spike train of length T^{l+1}. In the context of

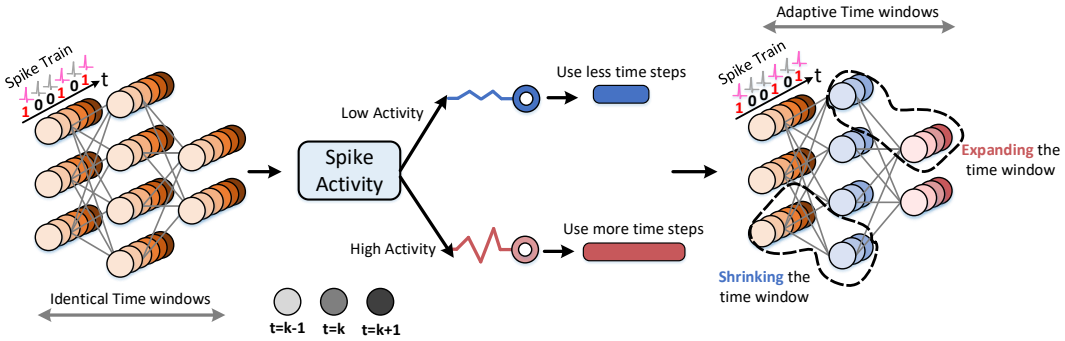


Figure 4: Overview of the proposed adaptive conversion in the temporal dimension. SNNs are routed to use different time steps among layers regarding their spike activity. The brighter circle indicates the time step closer to the first time step in the spike train.

SNN-R or SNN-T, we only need to find a transformation between firing times or firing numbers, i.e. f: $\{0, 1, \cdots, T^l\} \rightarrow \{0, 1, \cdots, T^{l+1}\}$.

An intuitive solution is linear mapping, i.e. $f(x) = \texttt{round}(\frac{T^{l+1}}{T^l}x)$. However, such conversion does not work when we want to expand the number of time steps, $T^{l+1} > T^l$. Since the input information has only $T^l + 1$ levels of difference, the output information has at most $T^l + 1$ levels of difference, failing to fulfill the object of increasing the resolution from T^l to T^{l+1}.

Inspired by the fact that the membrane potential at the firing time step also reflects the strength of the spike, a larger firing potential represents a stronger information, we separate the integer spike time or spike count information into continuous domain $[0, T^l]$ to better distinguish the order of each spike. Combing the relationship between the firing time or spike count with the strength of the information, we have

$$\begin{aligned} \tilde{x}_T &= x + \frac{V_{th}}{V_{fire}} \\ \tilde{x}_R &= x + 1 - \frac{V_{th}}{\overline{V}_{fire}} \end{aligned} \tag{3}$$

where, V_{th} is the threshold voltage. x_T and x_R is the the firing time in SNN-T and the spike count in SNN-R while $\tilde{x}_T$ and $\tilde{x}_R$ are new values in modified SNN, $\overline{V}_{fire}$ is the firing potential for the only spike in SNN-R, and V_{fire} is the average firing potential for all spikes in SNN-T. After applying the adjustment, we can apply the linear mapping to decide the input spike time to the next layer. For SNN-T, the input spike train to the next layer would a series with only one spike at time $\texttt{round}(\frac{T^{l+1}}{T^l}x_T)$. For SNN-R, the result input spike train would be a series with spikes at the first $\texttt{round}(\frac{T^{l+1}}{T^l}x_R)$ time steps.

3.3 Training Process

The key idea here is that training SNNs makes networks adaptive to the various representation in the varying temporal dimension. To train SNNs, spatio-temporal back-propagation (STBP) is used to perform gradients computation across the time window Mirsadeghi et al. (2021); Liu et al. (2021); Fang et al. (2021). Specifically, during the training, synaptic weights can be updated as:

$$\begin{aligned} W_l &= W_l - \eta \Delta W_l \\ where \quad \Delta W_l &= \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial O_l^t} \frac{\partial O_l^t}{\partial U_l^t} \frac{\partial U_l^t}{\partial W_l} \end{aligned} \tag{4}$$

Here, O_l^t and U_l^t are output spikes and membrane potential at time step t for l-th layer, respectively. Since the neuron generate the output spike only if the membrane potential exceeds the firing threshold, results in non-differentiability. To deal with this problem, we follow Roy et al. (2019); Comsa et al. (2020) and introduce an approximate gradient:

$$\frac{\partial O_l^t}{\partial U_l^t} = \max\{0, 1 - |\frac{U_l^t}{V_{th}}|\} \tag{5}$$

Initialization is an important issue during the training. Without proper initialization, SNNs varying the changes in the temporal dimension usually converge to a suboptimal state, where the temporal information is not allocated properly among the layers leads to network accuracy cannot being restored to the optimal one. Here, we allocate the time window to each layer of the pre-trained SNN based on the spiking activity at the beginning, where the shortened time windows is constrained to a target scaling ratio. Specifically, the layer with the highest spiking activity maintains the initial time window, and the other layers are shortened according to their spiking activities.

Objective of TEAS training. To encourage fewer time steps of layers, we introduce a regularization term into the training loss. This guides the average processing latency of the dynamic model to a specific target overall scaling ratio. The regularization can be formulated as:

$$\ell_r = \sum_{l=1}^{L} \frac{\sum_{l=1}^{L} \#Act(l) \times \#Para(l)}{\#Act(l)} \tag{6}$$

where the spiking activity $\#Act(l)$ reflects the magnitude of temporal information reduction tolerated by the layer l, and the number of parameters $\#Para(l)$ reflects the overall processing overhead as defined in Deng et al. (2021); Wang et al. (2019). The cross-entropy loss of the SNNs is:

$$\mathcal{L}_{CE} = -\sum_{i} y_i \log\left(\frac{e^{u_i^T}}{\sum_{k=1}^{N} e^{u_k^T}}\right) \tag{7}$$

where u_i corresponds to the firing rate or firing time of the output layer (i.e. the final (L-th) layer) across the simulation time window, y indicates ground truth, T is the number of time steps, and N is the number of categories. Therefore, the total loss function is the combination of cross-entropy loss and regularization:

$$\mathcal{L} = \mathcal{L}_{CE} + \lambda \ell_r \tag{8}$$

where ℓ_r combines the average spike rate and the number of parameters and λ is a penalty coefficient of the network to balance the entropy and regularization term. The loss function in Eq. (8) achieves the layer-wise adjustment by imposing a weaker regularization to the more vulnerable layers. The reason that we use the average spike rate combined with parameters is to unify the exploration of λ setting across different networks. By introducing this regularization item, we can further refine the firing activities of an SNN model, resulting in a decreased time window.

Temporal Adjustment. After back-propagation across all the time steps, we will adjust the temporal dimension of each layer. Specifically, we first count the spiking activity for each layer during the forward pass. Then, we check the time window T_l and remove the time steps based on the spiking activity (lower spiking activity is better for shortening the time window).

The regularization strength allocated to each layer will vary with the temporal dimension, as described in Eq. (8). The adjustment procedure of the temporal dimension will be executed periodically at intervals of several training epochs during the training process. At the end of the TEAS training, we perform a final adjustment of the temporal dimension in order to obtain the final allocation of the time window for each layer of the SNN. After the temporal dimensions are determined (i.e., the time windows of each layer are no longer changed), we further perform fine-tuning to improve the network accuracy of the SNN with adjusted temporal dimensions. Meanwhile, we find it would be more efficient in reaching higher accuracy by starting from a pre-trained model than training from scratch.

The training process is sensitive to parameter tuning; we observe that employing smaller tuning magnitudes for parameters usually yields superior performance in practice. In practice, we first initialize the time step for each layer based on the spiking activity and then gradually adjust the temporal information guided by the information strength until the SNN reaches its objective. We only update some of the most active layers during this process each iteration.

4 RESULTS

4.1 EXPERIMENTAL SETUP

Dataset and Network Architectures. In this work, we evaluate our TEAS on the standard visual object recognition benchmarks, namely the MNIST LeCun (1998), CIFAR-10, CIFAR-100 Krizhevsky

Table 1: Choice of hyperparameters.

Parameters	Descriptions	Configurations
N_p	# Epochs for pretraining	150
N_r	# Epochs for retraining	30
Batch Size	-	$16 \sim 64$
V_{th}	Threshold Potential	$0.7 \sim 10$
V_{reset}	Rest Potential	0.0
decay coefficient	Adam Decay	0.99
lr	Learning Rate	5e-4/1e-4

et al. (2014) and ImageNet datasets Deng et al. (2009). MNIST contains 70,000 grayscale images in 10 categories, of which 60,000 images are used for training and 10,000 images are for testing. CIFAR-10 includes 60k 32×32 RGB images evenly sampled from 10 categories, with 50k and 10k samples for training and test respectively. CIFAR-100 has the same configuration as CIFAR-10, except it contains images from 100 categories. ImageNet dataset contains 1.2M training images divided into 1000 distinct categories. To demonstrate the generalizability of our scheme, we evaluate the performance of our scheme on ANN converted SNNs and directly trained SNNs. For the MNIST dataset, we consider a shallow fully-connected neural network architecture (Input-Flatten-800-IF-10-IF) similar to previous works (i.e., directly trained SNNs). On the CIFAR-10/-100 and ImageNet datasets, we use a convolutional SNN (i.e., VGG-16 and ResNet network architecture), which is identical to previous works (i.e., ANN converted SNNs). To facilitate the learning process, we use the Adam optimizer. The choice of all hyperparameters is shown in Tab. 1. Note that, all the experiments are performed on a 4-way NVIDIA Tesla V100 under the framework of Pytorch Paszke et al. (2019).

Table 2: Classification Accuracy (%) on MNIST, CIFAR10, CIFAR100, and ImageNet.

SNN	Structure	Dataset	ANN Acc.	SNN Acc.	Ori. T	Reduced Acc.	Reduced T
Directly Trained SNN Liu et al. (2021)	2 FC	MNIST	-	98.1	16	98.1	6.7
Directly Trained SNN Deng et al. (2021)	LeNet	MNIST	-	99.07	10	99.05	7.1
Directly Trained SNN Liu et al. (2021)	7 Conv, 2 FC	CIFAR-10	-	91.31	16	91.31	6.2
Directly Trained SNN Fang et al. (2021)	6 Conv, 2 FC	CIFAR-10	-	92.84	8	92.80	5.3
ANN-Converted SNN Han & Roy (2020)	ResNet-20	CIFAR-10	91.47	91.42	2048	91.43	45.5
ANN-Converted SNN Han & Roy (2020)	VGG-16	CIFAR-10	93.63	93.63	2048	93.66	40.5
ANN-Converted SNN Han & Roy (2020)	ResNet-20	CIFAR-100	68.72	68.18	2048	68.15	78.2
ANN-Converted SNN Han & Roy (2020)	VGG-16	CIFAR-100	71.22	70.97	2048	70.95	71.3
ANN-Converted SNN Han & Roy (2020)	ResNet-34	ImageNet	70.64	69.93	4096	69.87	83.1
ANN-Converted SNN Han & Roy (2020)	VGG-16	ImageNet	73.49	73.46	2560	73.41	76.9

4.2 Result Evaluation

Inference performance. Tab. 2 shows the performance of our proposed TEAS for all four datasets. The results show that our method can achieve state-of-the-art results in temporal-adaptive SNNs. Specifically, for ANN converted SNNs with the VGG-16 network structure, we reduce the averaged time steps to 40.5, 71.3, and 77.3 on CIFAR-10, CIFAR-100, and ImageNet, respectively, achieving 0.03% accuracy loss on average, where the maximum accuracy loss is 0.05%. The results of the layer-wise time steps of the SNN with the VGG-16/ResNet-20 network structure under our TEAS is shown in Fig. 5. It can be observed that the maximum time steps among the layer in SNNs are 108 and 72 in VGG-16 and ResNet-20, respectively. TEAS can easily find better tradeoff points with both accuracy and the size of temporal dimension.

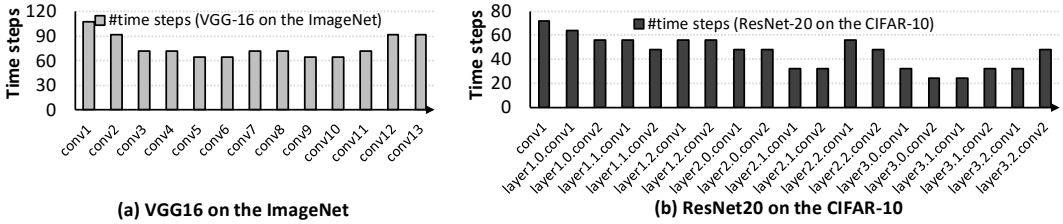


Figure 5: Layer-wise time steps of SNNs with VGG16 network structure on ImageNet (a) and ResNet20 network structure on CIFAR-10 (b).

Comparison to other competing methods. Tab. 3 provides a performance comparison between the proposed method and previous works, from which we can find that the proposed method outperforms previous works. The results show that the proposed TEAS method outperforms all the

Table 3: Comparion of our work with other SNN models on CIFAR-10, CIFAR-100 and ImageNet datasets

Method	Structure	ANN-SNN Conversion (T=2500)	Temporal Comp.	Acc.(%)	Time Steps
		CIFAR-10			
Deng et al. (2021)	11 layer CNN	-	312.5×	89.53	8
this work	**7 Conv, 2 FC**	**-**	403.2×	**91.31**	**6.2**
Kundu et al. (2021)	VGG16	92.48%	25×	91.29	100
Rathi et al. (2020)	VGG16	92.48%	25×	90.2	100
Rathi et al. (2020)	ResNet20	92.94%	10×	91.12	250
Garg et al. (2021)	VGG9	-	52×	89.94	48
Park et al. (2020)	VGG16	92.48%	3.7×	91.40	680
this work	**VGG16**	**92.48%**	54.9×	**91.43**	**45.5**
this work	**ResNet20**	**92.94%**	61.7×	**93.66**	**40.5**
		CIFAR-100			
Kundu et al. (2021)	VGG11	70.94%	20.8×	64.98	120
Rathi et al. (2020)	VGG11	70.94%	20×	65.52	125
Garg et al. (2021)	VGG11	-	52×	68.30	48
Park et al. (2020)	VGG16	70.97%	3.7×	68.80	680
this work	**VGG16**	**70.97%**	35.1×	**70.95**	**71.3**
		ImageNet			
Rathi et al. (2020)	VGG16	68.12%	10×	56.87	250
Rathi et al. (2020)	ResNet34	65.1%	10×	62.73	250
this work	**VGG16**	**73.46%**	32.5×	**73.41**	**76.9**
this work	**ResNet34**	**69.93%**	30.1×	**69.87**	**83.1**

competitors and achieves state-of-the-art temporal compression rate without significant accuracy loss. Compared to previous work, our TEAS adopts spiking activity-guided time step allocation to adjust the temporal information adaptively that could realize both high accuracy and efficiency. For example, regarding ResNet-20 on the CIFAR-10, TEAS achieves $61.7\times$ time steps reduction with nearly no accuracy loss compared to the original SNN (2500 time steps). For CIFAR-100, TEAS algorithm shows a 0.02% accuracy loss compared to the original SNN and a significant 6.3% accuracy improvement over the state-of-art temporal compression method Kundu et al. (2021) with 40% fewer time steps. While for the more complex task ImageNet, the adaptive time window among layers makes our TEAS more robust for preserving SNN accuracy than the SNN with the small windows Rathi et al. (2020). Specifically, TEAS achieves 4.9% accuracy improvement with 67% fewer time steps over the method in Rathi et al. (2020). This is because existing efforts to reduce temporal information are unified over all layers, which are too aggressive to achieve good accuracy. The original SNN pursues better performance while ignoring time steps; the compression method for the temporal dimension pursues the time steps to be small while ignoring the performance; our TEAS can achieve a balanced complexity-accuracy trade-off by encouraging the layers with the high and low spike activity to use the large and small number of time steps, respectively. These results suggest that there exists extremely high redundancy in the temporal dimension of SNNs.

Effect of retraining. As stated above, we propose to apply layer-wise spiking activity-aware adjustment on the regularization term to penalize more on smaller layers during the training. Fig. 6 compares the TEAS scheme and the SNN (ResNet-20 network structure) performance achieved without retraining or performing the TEAS retraining or without such a regularization term. All the hyperparameters are kept the same. We observe that SNNs without retraining (black line in Fig. 6) are sensitive to the temporal dimension, resulting in a drop in accuracy. On the other hand, SNN recovers the accuracy with the adaptive adjustment with retraining. Meanwhile, from the plot,

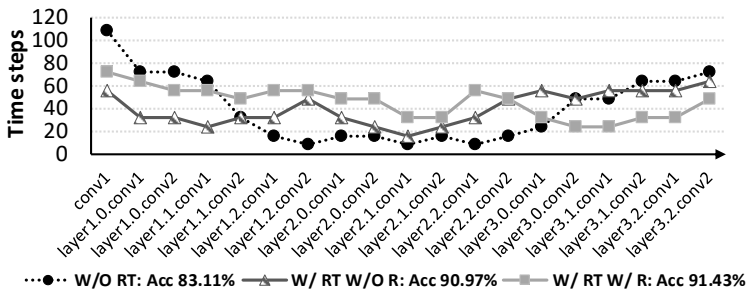


Figure 6: Layer-wise time steps comparison under TEAS schemes achieved with or without retraining. W/O RT, W/ RT W/O R and W/ RT W/ R indicate without retraining, retraining without regularization term and retraining with regularization term, respectively.

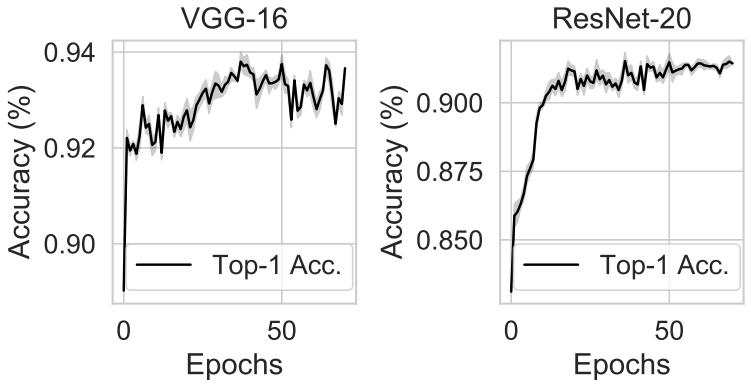
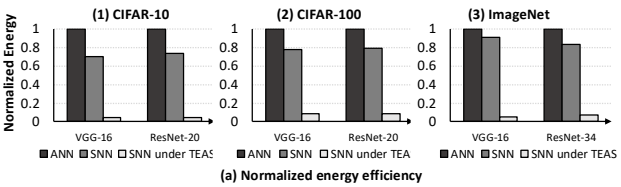


Figure 7: The accuracy (Top-1) evolution curve versus the number of epochs under TEAS. Regions in shadow indicates the error band w.r.t 5 trials.

training without the term will lead to over-penalization on earlier layers with higher spiking activities (vulnerable layers), while later layers with fewer spiking activities (invulnerable layers) are not compressed enough.

Before retraining, the parameters of the SNNs are set to be the corresponding values of the trained SNNs. We find it more efficient to reach convergence by starting from a pre-trained model than training from scratch. Fig. 7 shows the image classification accuracy of VGG-16 and ResNet-20 models with averaged time steps 40.5 and 45.5, as the retraining epochs increase. The results show that the SNN model can recover the original performance with 50 epoch.



Operation	Operand Format	Energy (pJ)	Relative costs
Adder	8-bit INT	0.03	1.0
	32-bit INT	0.1	3.3
Multiplier	8-bit INT	0.2	6.7
	32-bit INT	3.1	103.3
MAC (ANNs)	8-bit INT	0.23	7.7
	32-bit INT	3.2	106.7
AC (SNNs)	8-bit INT	0.03	1.0
	32-bit INT	0.1	3.3

(b) Estimated energy costs of operations in ANNs and SNNs for integer data format under 45nm process.

Figure 8: (a) Comparison of ANN, SNN and SNN under TEAS in terms of normalized energy efficiency with VGG and ResNet network structure on CIFAR-10 (1), CIFAR-100 (2) and ImageNet (3). (b) Estimated energy costs of operations in ANNs and SNNs for integer data format under 45nm process.

Computation efficiency of SNNs. SNNs are well-known to have higher energy efficiency compared to ANNs. Therefore, we compare the energy consumption of ANNs and SNNs. Here, we compute the energy based on the number of spikes generated by all layers. Fig. 8(a) illustrates the energy consumption for ANNs, SNNs and SNNs under TEAS with VGG16 and ResNet network structure on three datasets, where the energy is normalized to that of the ANN. Specifically, for VGG-16, the SNN under TEAS provides $11.5\times$ higher energy efficiency compared to ANN and SNN with the identical temporal dimension. This efficiency mainly comes from TEAS, which makes the temporal dimension assigned by SNN to each layer appropriate and without waste. Since the addition operation consumes significantly less energy than multiplication (see Fig. 8(b)), the SNNs are significantly more energy-efficient. However, since the energy consumption of ANN depends on FLOPs, while the fired spikes dominate that of SNN, making the actual energy efficiency gain of SNN limited by the fact that SNN requires multiple inferences. The time steps of SNN with our scheme are substantially reduced, the energy efficiency can reach up to $22.9\times$ on average, as opposed to ANN models having similar parameters.

5 Conclusion

Our proposed TEAS is the very first work for dynamic conversion on the temporal dimension of SNNs. TEAS sheds early lights on why the control of the temporal dimension for SNNs needs more attention. Our method uses spiking activities of layers, which can be easily integrated and trained in an end-to-end fashion, to guide the temporal dimension optimization. Then, we propose temporal conversion to skip the redundant temporal dimension at run-time. We demonstrate through extensive experiments and analyses that the redundancy in the temporal dimension of SNNs is extremely severe than anticipated.

REFERENCES

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, et al. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020.

Yanqi Chen, Zhaofei Yu, Wei Fang, Tiejun Huang, and Yonghong Tian. Pruning of deep spiking neural networks through gradient rewiring. *IJCAI*, 2021.

Iulia M Comsa, Thomas Fischbacher, et al. Temporal coding in spiking neural networks with alpha synaptic function. In *ICASSP*, pp. 8529–8533. IEEE, 2020.

Jia Deng, Wei Dong, Richard Socher, et al. Imagenet: A large-scale hierarchical image database. In *CVPR*, pp. 248–255. Ieee, 2009.

Lei Deng, Yujie Wu, Xing Hu, et al. Rethinking the performance comparison between snns and anns. *Neural Networks*, 121:294–307, 2020.

Lei Deng, Yujie Wu, Yifan Hu, Ling Liang, Guoqi Li, Xing Hu, Yufei Ding, Peng Li, and Yuan Xie. Comprehensive snn compression using admm optimization and activity regularization. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

Shikuang Deng and Shi Gu. Optimal conversion of conventional artificial neural networks to spiking neural networks. In *ICLR*, 2021.

Wei Fang, Zhaofei Yu, Yanqi Chen, Timothee Masquelier, Tiejun Huang, and Yonghong Tian. Incorporating learnable membrane time constant to enhance learning of spiking neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2661–2671, 2021.

Isha Garg, Sayeed Shafayet Chowdhury, and Kaushik Roy. Dct-snn: Using dct to distribute spatial information over time for low-latency spiking neural networks. In *ICCV*, pp. 4671–4680, 2021.

Bing Han and Kaushik Roy. Deep spiking neural network: Energy efficiency through time based coding. In *ECCV*, pp. 388–404, 2020.

Bing Han, Gopalakrishnan Srinivasan, and Kaushik Roy. Rmp-snn: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13558–13567, 2020.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55:5, 2014.

Souvik Kundu, Gourav Datta, Massoud Pedram, and Peter A Beerel. Spike-thrift: Towards energy-efficient deep spiking neural networks by limiting spiking activity via attention-guided compression. In *WACV*, pp. 3953–3962, 2021.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Hunjun Lee, Chanmyeong Kim, Yujin Chung, and Jangwoo Kim. Neuroengine: A hardware-based event-driven simulation system for advanced brain-inspired computing. In *ASPLOS*, pp. 975–989, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383172. doi: 10.1145/3445814.3446738. URL https://doi.org/10.1145/3445814.3446738.

Fangxin Liu, Wenbo Zhao, Yongbiao Chen, et al. Sstdp: Supervised spike timing dependent plasticity for efficient spiking neural network training. *Frontiers in Neuroscience*, 15, 2021.

Maryam Mirsadeghi, Majid Shalchian, Saeed Reza Kheradpisheh, and Timothée Masquelier. Stidi-bp: Spike time displacement based error backpropagation in multilayer spiking neural networks. *Neurocomputing*, 427:131–140, 2021.

Surya Narayanan, Karl Taht, Rajeev Balasubramonian, Edouard Giacomin, and Pierre-Emmanuel Gaillardon. Spinalflow: an architecture and dataflow tailored for spiking neural networks. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 349–362. IEEE, 2020.

Seongsik Park, Seijoon Kim, Byunggook Na, and Sungroh Yoon. T2fsnn: deep spiking neural networks with time-to-first-spike coding. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6. IEEE, 2020.

Adam Paszke, Sam Gross, Francisco Massa, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Nitin Rathi, Gopalakrishnan Srinivasan, Priyadarshini Panda, and Kaushik Roy. Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. In *ICLR*, 2020.

Nitin Rathi, Amogh Agrawal, Chankyu Lee, Adarsh Kumar Kosta, and Kaushik Roy. Exploring spike-based learning for neuromorphic computing: Prospects and perspectives. In *DATE*, pp. 902–907. IEEE, 2021.

Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, 2019.

Sonali Singh, Anup Sarma, Nicholas Jao, et al. Nebula: a neuromorphic spin-based ultra-low power architecture for snns and anns. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 363–376. IEEE, 2020.

Gopalakrishnan Srinivasan and Kaushik Roy. Restocnet: Residual stochastic binary convolutional spiking neural network for memory-efficient neuromorphic computing. *Frontiers in neuroscience*, 13:189, 2019.

Amirhossein Tavanaei and Anthony Maida. Bp-stdp: Approximating backpropagation using spike timing dependent plasticity. *Neurocomputing*, 330:39–47, 2019.

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *CVPR*, pp. 8612–8620, 2019.