frontiers in Neuroscience

ORIGINAL RESEARCH
published: 09 December 2021
doi: 10.3389/fnins.2021.773954

Revisiting Batch Normalization for Training Low-Latency Deep Spiking Neural Networks From Scratch

Youngeun Kim and Priyadarshini Panda*

Department of Electrical Engineering, Yale University, New Haven, CT, United States

Spiking Neural Networks (SNNs) have recently emerged as an alternative to deep learning owing to sparse, asynchronous and binary event (or spike) driven processing, that can yield huge energy efficiency benefits on neuromorphic hardware. However, SNNs convey temporally-varying spike activation through time that is likely to induce a large variation of forward activation and backward gradients, resulting in unstable training. To address this training issue in SNNs, we revisit Batch Normalization (BN) and propose a temporal Batch Normalization Through Time (BNTT) technique. Different from previous BN techniques with SNNs, we find that varying the BN parameters at every time-step allows the model to learn the time-varying input distribution better. Specifically, our proposed BNTT decouples the parameters in a BNTT layer along the time axis to capture the temporal dynamics of spikes. We demonstrate BNTT on CIFAR-10, CIFAR-100, Tiny-ImageNet, event-driven DVS-CIFAR10 datasets, and Sequential MNIST and show near state-of-the-art performance. We conduct comprehensive analysis on the temporal characteristic of BNTT and showcase interesting benefits toward robustness against random and adversarial noise. Further, by monitoring the learnt parameters of BNTT, we find that we can do temporal early exit. That is, we can reduce the inference latency by $\sim 5-20$ time-steps from the original training latency. The code has been released at https://github.com/Intelligent-Computing-Lab-Yale/BNTT-Batch-Normalization-Through-Time.

Keywords: spiking neural network, batch normalization, image recognition, event-based processing, energy-efficient deep learning

Edited by:
Emre O. Neftci,
University of California, Irvine,
United States

Reviewed by:
Jason Eshraghian,
University of Michigan, United States
Elena Cerezuela,
Sevilla University, Spain
Yujie Wu,
Tsinghua University, China

****Correspondence:***
Youngeun Kim
youngeun.kim@yale.edu

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *10 September 2021*
Accepted: *08 November 2021*
Published: *09 December 2021*

Citation:
Kim Y and Panda P (2021) Revisiting Batch Normalization for Training Low-Latency Deep Spiking Neural Networks From Scratch. Front. Neurosci. 15:773954. doi: 10.3389/fnins.2021.773954

1. INTRODUCTION

Artificial Neural Networks (ANNs) have shown state-of-the-art performance across various computer vision tasks. Nonetheless, huge energy consumption incurred for implementing ANNs on conventional von-Neumann hardware limits their usage in low-power and resource-constrained Internet of Things (IoT) environment, such as mobile phones, drones among others. In the context of low-power machine intelligence, Spiking Neural Networks (SNNs) have received considerable attention in the recent past (Cao et al., 2015; Diehl and Cook, 2015; Roy et al., 2019; Comsa et al., 2020; Panda et al., 2020). Inspired by biological neuronal mechanisms, SNNs process visual information with discrete spikes or events over multiple time-steps. Recent works have shown that the event-driven behavior of SNNs can be implemented on emerging neuromorphic hardware to yield 1–2 order of magnitude energy efficiency over ANNs (Akopyan et al., 2015; Davies et al., 2018).

Despite the energy efficiency benefits, SNNs have still not been widely adopted due to inherent training challenges. The training issue arises from the non-differentiable characteristic of a spiking neuron, generally, Integrate-and-Fire (IF) type (Burkitt, 2006), that makes SNNs incompatible with gradient descent training.

To address the training issue of SNNs, several methods, such as, *Conversion* and *Surrogate Gradient Descent* have been proposed. In ANN-SNN conversion (Diehl et al., 2015; Rueckauer et al., 2017; Sengupta et al., 2019; Han et al., 2020), off-the-shelf trained ANNs are converted to SNNs using normalization methods to transfer ReLU activation to IF spiking activity. The advantage here is that training happens in the ANN domain leveraging widely used machine learning frameworks like, PyTorch, that yield short training time and can be applied to complex datasets. But the ANN-SNN conversion method requires large number of time-steps ($\sim 500 - 1,000$) for inference to yield competitive accuracy, which significantly increases the latency and energy consumption of the SNN. On the other hand, directly training SNNs with a surrogate gradient function (Wu et al., 2018; Neftci et al., 2019; Lee et al., 2020) exploits temporal dynamics of spikes, resulting in lesser number of time-steps ($\sim 100 - 150$). However, the discrepancy between forward spike activation function and backward surrogate gradient function during backpropagation restricts the training capability. Therefore, naive SNNs without additional optimization techniques are difficult to be trained on large-scale datasets (e.g., CIFAR-100 and Tiny-ImageNet). Recently, a hybrid method (Rathi et al., 2020) that combines the conversion method and the surrogate gradient-based method shows state-of-the-art performance at reasonable latency ($\sim$ 250 time-steps). However, the hybrid method incurs sequential processes, i.e., training ANN from scratch, conversion of ANN to SNN, and training SNNs using surrogate gradient descent, that increases the total computation cost to obtain the final SNN model. Overall, training high-accuracy and low-latency SNNs from scratch still remains an open problem.

In this paper, we investigate the temporal characteristics of Batch Normalization (BN) for more advanced SNN training. The BN layer (Ioffe and Szegedy, 2015) has been used extensively in deep learning to accelerate the training process of ANNs. It is well known that BN reduces internal covariate shift (or soothing optimization landscape Santurkar et al., 2018) mitigating the problem of exploding/vanishing gradients. In SNN literature, there are a few recent works that leverage BN layers during training and have shown competitive performance for image classification tasks with low latency. Ledinauskas et al. (2020) use a standard BN layer and show the scalability of SNNs toward deep architectures with BN layers. Fang et al. (2020) propose a learnable membrane time constant with a standard BN layer. Zheng et al. (2020) present the advantage of scaling BN parameter according to the neuronal firing threshold. Even though the previous BN approaches show performance/latency improvement, we assert that there is need to explore the advantage of BN in the temporal dimension since SNNs convey information through time. The previous BN works with SNNs use a single BN parameter across all time-steps. We are essentially motivated by the question, *Can a single learnable parameter in the BN layer learn the temporal characteristics of the input spikes that vary across different time-steps?*

Different from previous works, we highlight the importance of temporal characterization of BN technique. To this end, we propose a new SNN-crafted batch normalization layer called Batch Normalization Through Time (BNTT) that decouples the parameters in the BN layer across different time-steps. BNTT is implemented as an additional layer in SNNs and is trained with surrogate gradient backpropagation. To investigate the effect of our BNTT, we compare the statistics of spike activity of BNTT with previous approaches: Conversion (Sengupta et al., 2019) and standard Surrogate Gradient Descent (Neftci et al., 2019), as shown in **Figure 1**. Interestingly, different from the conversion method and surrogate gradient method (without BNTT) that maintain reasonable spike activity during the entire time period across different layers, spike activity of layers trained with BNTT follows a gaussian-like trend. BNTT imposes a variation in spiking across different layers, wherein, each layer's activity peaks in a particular time-step range and then decreases. Moreover, the peaks for early layers occur at initial time-steps and latter layers peak at later time-steps. This phenomenon implies that learnable parameters in BNTT enable the networks to pass the visual information temporally from shallow to deeper layers in an effective manner.

The newly observed characteristics of BNTT brings several advantages. First, similar to BN, the BNTT layer enables SNNs to be trained stably from scratch even for large-scale datasets. Second, learnable parameters in BNTT enable SNNs to be trained with low latency ($\sim 25 - 50$ time-steps) and impose optimum spike activity across different layers for low-energy inference. Finally, the distribution of the BNTT learnable parameter (i.e., γ) is a good representation of the temporal dynamics of spikes. Hence, relying on the observation that low γ value induces low spike activity and vice-versa, we further propose a temporal early exit algorithm. Here, an SNN can predict at an earlier time-step and does not need to wait till the end of the time period to make a prediction.

In summary, our key contributions are as follows: (i) We explore the temporal characteristics of BN for SNNs and propose a temporally adaptive BN approach, called BNTT. (ii) BNTT allows SNNs to be implemented in a low-latency and low-energy environment. (iii) We further propose a temporal early exit algorithm at inference time by monitoring the learnable parameters in BNTT. (iv) To ascertain that BNTT captures the temporal characteristics of SNNs, we mathematically show that proposed BNTT has similar effect as controlling the firing threshold of the spiking neuron at every time step during inference.

2. BATCH NORMALIZATION

Batch Normalization (BN) reduces the internal covariate shift (or variation of loss landscape Santurkar et al., 2018) caused by the distribution change of input signal, which is a known problem of deep neural networks (Ioffe and Szegedy, 2015). Instead of calculating the statistics of total dataset, the intermediate

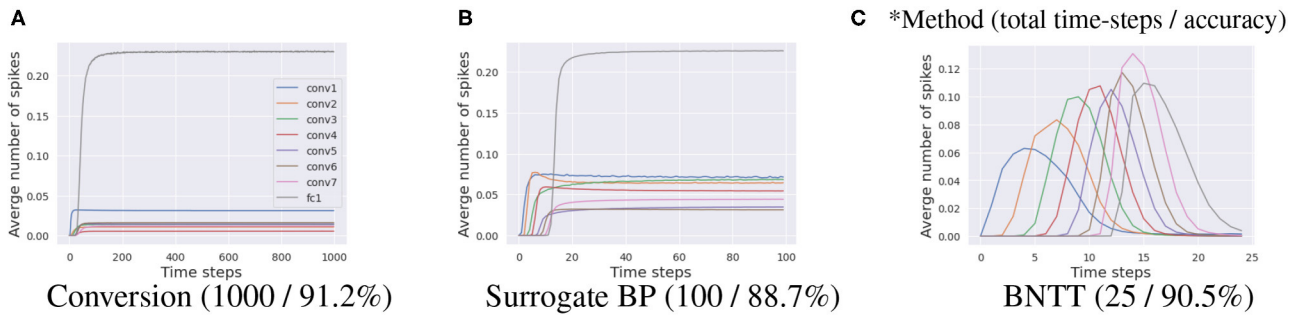

FIGURE 1 | Visualization of the average number of spikes in each layer with respect to time-steps. Compared to **(A)** ANN-SNN conversion and **(B)** surrogate gradient-based backpropagation, our **(C)** BNTT captures the temporal dynamics of spike activation with learnable parameters, enabling low-latency (i.e., small time-steps) and low-energy (i.e., less number of spikes) training. All experiments are conducted on CIFAR-10 with VGG9.

representations are standardized with a mini-batch to reduce the computation complexity. Given a mini-batch $\mathcal{B} = \{x_{1,\dots,m}\}$, the BN layer computes the mean and variance of the mini-batch as:

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{b=1}^{m} x_b; \qquad \sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{b=1}^{m}(x_b - \mu_{\mathcal{B}})^2. \tag{1}$$

Then, the input features in the mini-batch are normalized with calculated statistics as:

$$\widehat{x}_b = \frac{x_b - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}, \tag{2}$$

where, ϵ is a small constant for numerical stability. To further improve the representation capability of the layer, learnable parameters γ and β are used to transform the input features that can be formulated as $BN(x_i) = \gamma\widehat{x}_i + \beta$. At inference time, BN uses the running average of mean and variance obtained from training. In this work, different from the static BN, we explore the temporal characteristics of BN with SNNs by enabling temporally-varying parameters in BN.

3. METHODOLOGY

3.1. Spiking Neural Networks

Different from conventional ANNs, SNNs transmit information using binary spike trains. To leverage the temporal spike information, Leaky-Integrate-and-Fire (LIF) model (Dayan and Abbott, 2001) is widely used to emulate neuronal functionality in SNNs, which can be formulated as a differential equation:

$$\tau_m \frac{dU_m}{dt} = -U_m + RI(t), \tag{3}$$

where, U_m represents the membrane potential of the neuron that characterizes the internal state of the neuron, τ_m is the time constant of membrane potential decay. Also, R and $I(t)$ denote the input resistance and the input current at time t, respectively. Following the previous work (Wu et al., 2019), we convert this continuous dynamic equation into a discrete equation for digital simulation. For a single post-synaptic neuron i, we can represent the membrane potential u_i^t at time-step t as:

$$u_i^t = \lambda u_i^{t-1} + \sum_j w_{ij} o_j^t. \tag{4}$$

Here, j is the index of a pre-synaptic neuron, λ is a leak factor with value less than 1, o_j is the binary spike activation, and w_{ij} is the weight of the connection between pre- and post-neurons. From Equation (4), the membrane potential of a neuron decreases due to leak and increases due to the weighted sum of incoming input spikes.

If the membrane potential u exceeds a pre-defined firing threshold θ, the LIF neuron i generates a binary spike output o_i. After that, we perform a soft reset, where the membrane potential u_i is reset by reducing its value by the threshold θ. Compared to a hard reset (resetting the membrane potential u_i to zero after neuron i spikes), the soft reset minimizes information loss by maintaining the residual voltage and carrying it forward to the next time step, thereby achieving better performance (Han et al., 2020). **Figure 2A** illustrates the membrane potential dynamics of a LIF neuron.

For the output layer, we discard the thresholding functionality so that neurons do not generate any spikes. We allow the output neurons to accumulate the spikes over all time-steps by fixing the leak parameter (λ in Equation 4) as one. This enables the output layer to compute probability distribution after softmax function without information loss. As with ANNs, the number of output neurons in SNNs is identical to the number of classes C in the dataset. From the accumulated membrane potential, we can define the cross-entropy loss for SNNs as:

$$L = -\sum_i y_i log(\frac{e^{u_i^T}}{\sum_{k=1}^{C} e^{u_k^T}}), \tag{5}$$

where, y is the ground-truth label, and T represents the total number of time-steps. Then, the weights of all layers are updated by backpropagating the loss value with gradient descent.

To compute the gradients of each layer l, we use back-propagation through time (BPTT), which accumulates the gradients over all time-steps (Wu et al., 2018; Neftci et al., 2019).

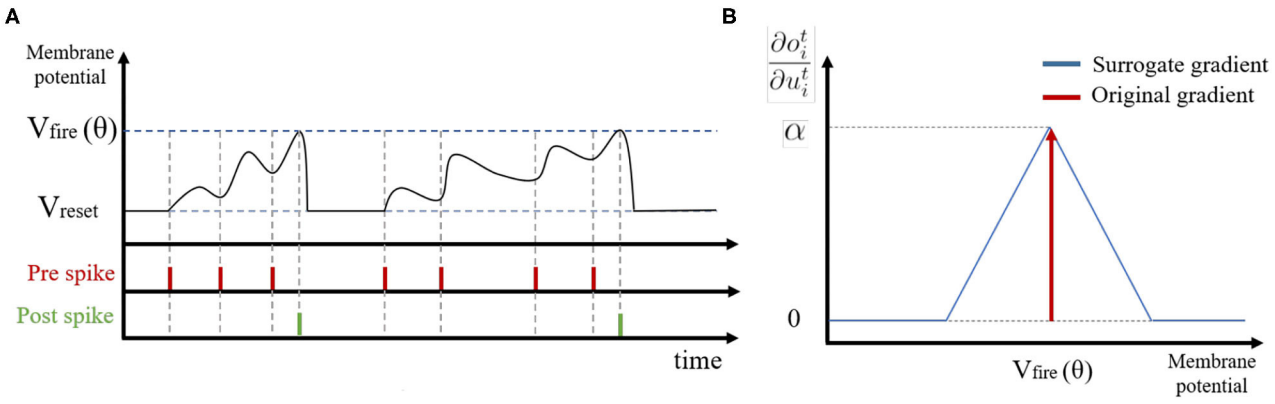

FIGURE 2 | (A) Illustration of spike activities in Leaky-Integrate-and-Fire neurons. **(B)** The approximated gradient value with respect to the membrane potential.

These approaches can be implemented with auto-differentiation tools, such as PyTorch (Paszke et al., 2017), that enable backpropagation on the unrolled network. To this end, we compute the loss function at time-step T and use gradient descent optimization. Mathematically, we can define the accumulated gradients at the layer l by chain rule as:

$$\frac{\partial L}{\partial W_l} = \begin{cases} \sum_t \left(\frac{\partial L}{\partial O_l^t}\frac{\partial O_l^t}{\partial U_l^t} + \frac{\partial L}{\partial U_l^{t+1}}\frac{\partial U_l^{t+1}}{\partial U_l^t}\right)\frac{\partial U_l^t}{\partial W_l}, & \text{if } l = \text{hidden layer} \\ \sum_t \frac{\partial L}{\partial U_l^T}\frac{\partial U_l^T}{\partial W_l}. & \text{if } l = \text{output layer} \end{cases} \tag{6}$$

Here, O_l and U_l are output spikes and membrane potential at layer l, respectively. For the output layer, we get the derivative of the loss L with respect to the membrane potential u_i^T at final time-step T:

$$\frac{\partial L}{\partial u_i^T} = \frac{e^{u_i^T}}{\sum_{k=1}^{C} e^{u_k^T}} - y_i. \tag{7}$$

This derivative function is continuous and differentiable for all possible membrane potential values. On the other hand, LIF neurons in hidden layers generate spike output only if the membrane potential u_i^t exceeds the firing threshold, leading to non-differentiability. To deal with this problem, we introduce an approximate gradient (**Figure 2B**):

$$\frac{\partial o_i^t}{\partial u_i^t} = \alpha \max\{0, 1 - |\frac{u_i^t - \theta}{\theta}|\}, \tag{8}$$

where, α is a damping factor for back-propagated gradients. Note, a large α value causes unstable training as gradients are summed over all time-steps. Hence, we set α to 0.3. Overall, we update the network parameters at the layer l based on the gradient value (Equation 6) as $W_l = W_l - \eta \Delta W_l$.

3.2. Batch Normalization Through Time (BNTT)

In this work, we present a new temporally-variant Batch Normalization for accelerating SNN training. We first visualize the distribution of the input signal of standard BN at layer 5 in VGG9 SNN with surrogate-gradients based training (**Figure 3**). The results show that the input signal to the BN layer varies with time. Therefore, we assert that if we enable temporal flexibility to BN parameters (e.g., global mean μ, global variation σ, and learnable parameter γ), the representation power of the networks might be improved.

To this end, we vary the internal parameters in a BN layer through time, that we define as, BNTT. Similar to the digital simulation of LIF neuron across different time-steps, one BNTT layer is expanded temporally with a local learning parameter associated with each time-step. This allows the BNTT layer to capture temporal statistics (see section 3.3 for mathematical analysis). The proposed BNTT layer is easily applied to SNNs by inserting the layer after convolutional/linear operations as:

$$\begin{aligned} u_i^t &= \lambda u_i^{t-1} + BNTT_{\gamma^t}(\sum_j w_{ij} o_j^t) \\ &= \lambda u_i^{t-1} + \gamma_i^t(\frac{\sum_j w_{ij} o_j^t - \mu_i^t}{\sqrt{(\sigma_i^t)^2 + \epsilon}}). \end{aligned} \tag{9}$$

During the training process, we compute the mean μ_i^t and variance σ_i^t from the samples in a mini-batch $\mathcal{B}$ for each time step t, as shown in **Algorithm 1**. Note, for each time-step t, we apply an exponential moving average to approximate global mean $\bar{\mu}_i^t$ and variance $\bar{\sigma}_i^t$ over training iterations. These global statistics are used to normalize the test data at inference. Also, we do not utilize β as in conventional BN, since it adds redundant voltage to the membrane potential of SNNs.

Adding the BNTT layer to LIF neurons changes the gradient calculation for backpropagation. Given that $x_i^t = \sum_j w_{ij} o_j^t$ is an input signal to the BNTT layer, we can calculate the gradient value passed through lower layers by the BNTT layer as:

$$\frac{\partial L}{\partial x_b^t} = \frac{1}{m\sqrt{(\sigma^t)^2 + \epsilon}}\left(m\frac{\partial L}{\partial \hat{x}_b^t} - \sum_{k=1}^{m}\frac{\partial L}{\partial \hat{x}_k^t} - \hat{x}_b^t\sum_{k=1}^{m}\frac{\partial L}{\partial \hat{x}_k^t}\hat{x}_k^t\right). \tag{10}$$

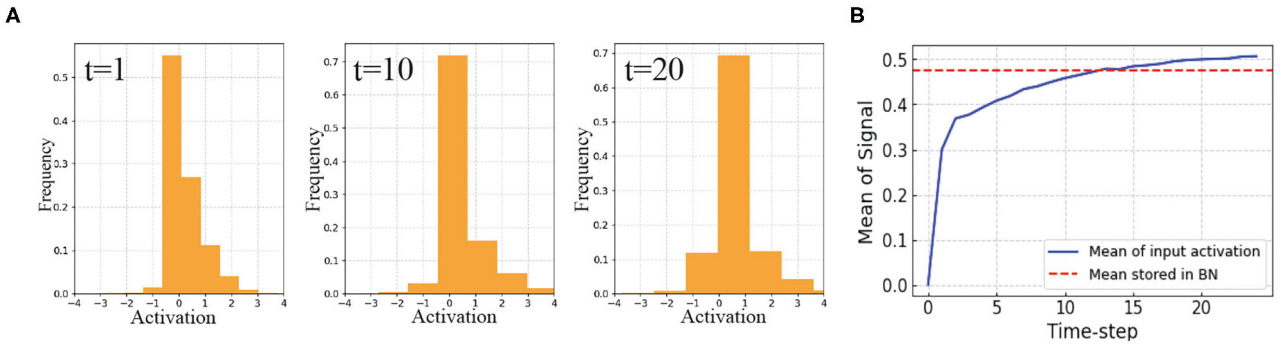

FIGURE 3 | SNNs with standard BN: **(A)** Distributions of the input activation of BN at time-step 1, 10, and 20. **(B)** While the mean of input activation varies with time, stored mean in standard BN layer has constant value at inference. This will create discrepancy and inhibit the BN layer to learn well. This suggests a temporally varying BN technique.

Here, we omit a neuron index i for simplicity. Also, m and b denote the batch size and batch index (see **Supplementary Material A** for more detail). Thus, for every time-step t, gradients are calculated based on the time-specific statistics of input signals. This allows the networks to take into account temporal dynamics for training weight connections. Moreover, a learnable parameter γ is updated to restore the representation power of the batch normalized signal. Since we use different γ^t values across all time-steps, γ^t finds an optimum over each time-step for efficient inference. We update gamma $\gamma^t = \gamma^t - \eta\Delta\gamma^t$ where:

$$\Delta\gamma^t = \frac{\partial L}{\partial \gamma^t} = \frac{\partial L}{\partial u^t}\frac{\partial u^t}{\partial \gamma^t} = \sum_{k=1}^{m}\frac{\partial L}{\partial u_k^t}\hat{x}_k^t. \tag{11}$$

3.3. Mathematical Analysis

In this section, we discuss the connections between BNTT and the firing threshold of a LIF neuron. Specifically, we formally prove that using BNTT has a similar effect as varying the firing threshold over different time-steps, thereby ascertaining that BNTT captures temporal characteristics in SNNs. Recall that BNTT normalizes the input signal using stored approximated global average $\bar{\mu}_i^t$ and standard deviation $(\bar{\sigma}_i^t)^2$ at inference. From Equation (9), we can calculate a membrane potential at time-step $t = 1$, given that initial membrane potential u_i^0 has a zero value:

$$\begin{aligned} u_i^1 &= \gamma_i^1\left(\frac{\sum_j w_{ij}o_j^1 - \bar{\mu}_i^1}{\sqrt{(\bar{\sigma}_i^1)^2+\epsilon}}\right) \\ &\approx \frac{\gamma_i^1}{\sqrt{(\bar{\sigma}_i^1)^2+\epsilon}}\sum_j w_{ij}o_j^1 = \frac{\gamma_i^1}{\sqrt{(\bar{\sigma}_i^1)^2+\epsilon}}\tilde{u}_i^1. \end{aligned} \tag{12}$$

Here, we assume $\bar{\mu}_i^1$ can be neglected with small signal approximation due to the spike sparsity in SNNs, and $\tilde{u}_i^1 = \sum_j w_{ij}o_j^1$ is membrane potential at time-step $t = 1$ without BNTT (obtained from Equation 4). We can observe that the membrane potential with BNTT is proportional to the membrane potential without BNTT at $t = 1$. For time-step $t > 1$, we should take into account the membrane potential from the previous time-step, which is multiplied by leak λ. To this end, by substituting (Equation 12) in the BNTT equation (Equation 9), we can formulate the membrane potential at $t = 2$ as:

$$\begin{aligned} u_i^2 &\approx \lambda u_i^1 + \frac{\gamma_i^2}{\sqrt{(\sigma_i^2)^2+\epsilon}}\sum_j w_{ij}o_j^2 \\ &= \left(\frac{\lambda\gamma_i^1}{\sqrt{(\sigma_i^1)^2+\epsilon}}\right)\tilde{u}_i^1 + \frac{\gamma_i^2}{\sqrt{(\sigma_i^2)^2+\epsilon}}\sum_j w_{ij}o_j^2 \\ &\approx \frac{\gamma_i^2}{\sqrt{(\sigma_i^2)^2+\epsilon}}\{\lambda\tilde{u}_i^1 + \sum_j w_{ij}o_j^2\} = \frac{\gamma_i^2}{\sqrt{(\sigma_i^2)^2+\epsilon}}\tilde{u}_i^2. \end{aligned} \tag{13}$$

In the third line, the learnable parameter γ_i^t and σ_i^t have similar values in adjacent time intervals ($t = 1, 2$) because of continuous time property. Hence, we can approximate γ_i^1 and σ_i^1 as γ_i^2 and σ_i^2, respectively. Finally, we can extend the equation of BNTT to the time-step t:

$$u_i^t \approx \frac{\gamma_i^t}{\sqrt{(\sigma_i^t)^2+\epsilon}}\tilde{u}_i^t. \tag{14}$$

Considering that a neuron produces an output spike activation whenever the membrane potential $\tilde{u}_i^t$ exceeds the pre-defined firing threshold θ, the spike firing condition with BNTT can be represented $u_i^t \geq \theta$. Comparing with the threshold of a neuron without BNTT, we can reformulate the firing condition as:

$$\tilde{u}_i^t \geq \frac{\sqrt{(\sigma_i^t)^2+\epsilon}}{\gamma_i^t}\theta. \tag{15}$$

Thus, we can infer that using a BNTT layer changes the firing threshold value by $\sqrt{(\sigma_i^t)^2+\epsilon}/\gamma_i^t$ at every time-step. In practice, BNTT results in an optimum γ during training that improves the representation power, producing better performance and

Algorithm 1: BNTT layer

Input: mini-batch $\mathcal{B}$ at time step t ($x^t_{\{1...m\}}$), learnable parameter (γ^t), update factor (α)
Output: $\{y^t = \text{BNTT}_{\gamma^t}(x^t)\}$

$\mu^t \leftarrow \frac{1}{m}\sum_{b=1}^{m} x_b^t$
$(\sigma^t)^2 \leftarrow \frac{1}{m}\sum_{b=1}^{m}(x_b^t - \mu^t)^2$
$\hat{x}^t = \frac{x^t - \mu^t}{\sqrt{(\sigma^t)^2 + \epsilon}}$
$y^t \leftarrow \gamma^t \hat{x}^t \equiv \text{BNTT}_{\gamma^t}(x^t)$
% Exponential moving average
$\bar{\mu}^t \leftarrow (1-\alpha)\bar{\mu}^t + \alpha\mu^t$
$\bar{\sigma}^t \leftarrow (1-\alpha)\bar{\sigma}^t + \alpha\sigma^t$

Algorithm 2: Training process with BNTT

Input: mini-batch (X); label set (Y); max_timestep (T)
Output: updated network weights

for $i \leftarrow 1$ to *max_iter* **do**
  fetch a mini batch X
  for $t \leftarrow 1$ to T **do**
    O $\leftarrow$ PoissonGenerator(X)
    for $l \leftarrow 1$ to $L-1$ **do**
      $(O_l^t, U_l^t) \leftarrow (\lambda, U_l^{t-1}, \text{BNTT}_{\gamma^t}(W_l, O_{l-1}^{t-1}))$
    end for
    % For the final layer L, stack the voltage
    $U_L^t \leftarrow (U_L^{t-1}, \text{BNTT}_{\gamma^t}(W_l, O_{L-1}^{t-1}))$
  end for
  % Calculate the loss and back-propagation
  $L \leftarrow (U_L^T, Y)$
end for

low-latency SNNs.This observation allows us to consider the advantages of time-varying learnable parameters in SNNs. This implication is in line with previous work (Han et al., 2020), which insists that manipulating the firing threshold improves the performance and latency of the ANN-SNN conversion method. However, Han et al. change the threshold value in a heuristic way without any optimization process and fix the threshold value across all time-steps. On the other hand, our BNTT yields time-specific γ^t which can be optimized via back-propagation.

3.4. Early Exit Algorithm

The main objective of early exit is to reduce the latency during inference (Panda et al., 2016; Teerapittayanon et al., 2016). Most previous methods (Wu et al., 2018; Sengupta et al., 2019; Han et al., 2020; Lee et al., 2020; Rathi et al., 2020) accumulate output spikes till the end of the time-sequence, at inference, since all layers generate spikes across all time-steps as shown in **Figures 1A,B**. On the other hand, learnable parameters in BNTT manipulate the spike activity of each layer to produce a peak value, which falls again (a gaussian-like trend), as shown in **Figure 1C**. This phenomenon shows that SNNs using BNTT convey little information at the end of spike trains.

Inspired by this observation, we propose a temporal early exit algorithm based on the value of γ^t. From Equation (15), we know that a low γ^t value increases the firing threshold, resulting in low spike activity. A high γ^t value, in contrast, induces more spike activity. It is worth mentioning that $(\sigma_i^t)^2$ shows similar values across all time-steps and therefore we only focus on γ^t. Given that the intensity of spike activity is proportional to γ^t, we can infer that spikes will hardly contribute to the classification result once γ^t values across every layer drop to a minimum value. Therefore, we measure the average of γ^t values in each layer l at every time-step, and terminate the inference when γ^t value in every layer is below a pre-determined threshold. For example, as shown in **Figure 4**, we observe that all averaged γ^t values are lower than threshold 0.1 after $t > 20$. Therefore, we define the early exit time at $t = 20$. Note that we can determine the optimum time-step for early exit before forward propagation without any additional computation. In summary, the temporal early exit method enables us to find the earliest time-step during inference that ensures integration of crucial information, in turn reducing the inference latency without significant loss of accuracy.

3.5. Overall Optimization

Algorithm 2 summarizes the whole training process of SNNs with BNTT. Our proposed BNTT acts as a regularizer, unlike previous methods (Lee et al., 2016, 2020; Sengupta et al., 2019; Rathi et al., 2020) that use dropout to perform regularization. Our training scheme is based on widely used rate coding where the spike generator produces a Poisson spike train (see **Supplementary Material B**) for each pixel in the image with frequency proportional to the pixel intensity (Roy et al., 2019). For all layers, the weighted sum of the input signal is passed through a BNTT layer and then is accumulated in the membrane potential. If the membrane potential exceeds the firing threshold, the neuron generates an output spike. For last layer, we accumulate the input voltage over all time-steps without leak, that we feed to a softmax layer to output a probability distribution. Then, we calculate a cross-entropy loss function and gradients for weight of each layer with the approximate gradient function. During the training phase, a BNTT layer computes the time-dependent statistics (i.e., μ^t and σ^t) and stores the moving-average global mean and variance. At inference, we first define the early exit time-step based on the value of γ in BNTT. Then, the networks classify the test input (note, test data normalized with pre-computed global $\bar{\mu}^t, \bar{\sigma}^t$ BNTT statistics) based on the accumulated output voltage at the pre-computed early exit time-step.

4. EXPERIMENTS

In this section, we carry out comprehensive experiments on public classification datasets. We first compare our BNTT with previous SNNs training methods. Then, we quantitatively and qualitatively demonstrate the effectiveness of our proposed BNTT.

4.1. Experimental Setup

We evaluate our method on three static datasets (i.e., CIFAR-10, CIFAR-100, Tiny-ImageNet), one neuromophic dataset (i.e., DVS-CIFAR10), and one sequential dataset (i.e., Sequential

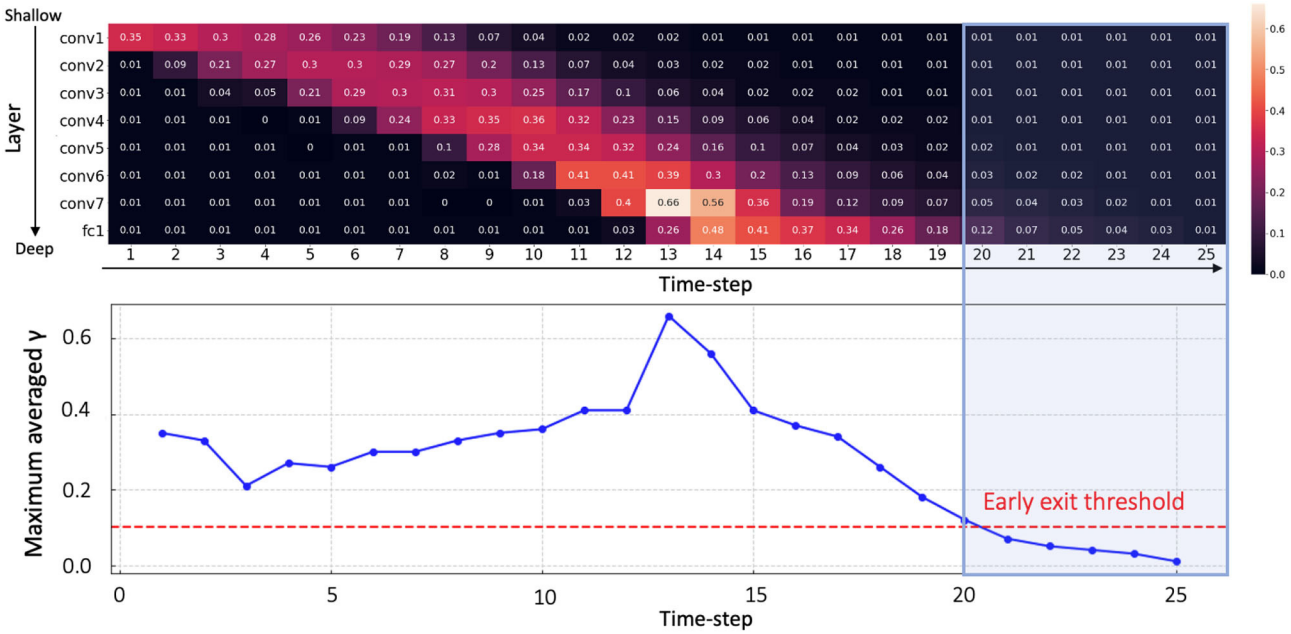

FIGURE 4 | The average value of γ^t at each layer over all time-steps (upper panel). Maximum averaged γ^t for each time-step (lower panel). Early exit time can be calculated as $t = 20$ since γ^t values at every layer have lower value than threshold 0.1 after time-step 20 (blue shaded area). Here, we use a VGG9 architecture on CIFAR-10.

MNIST). **CIFAR-10** (Krizhevsky and Hinton, 2009) consists of 60,000 images (50,000 for training/10,000 for testing) with 10 categories. All images are RGB color images whose size are 32 × 32. **CIFAR-100** has the same configuration as CIFAR-10, except it contains images from 100 categories. **Tiny-ImageNet** is the modified subset of the original ImageNet dataset. Here, there are 200 different classes of ImageNet dataset (Deng et al., 2009), with 100,000 training and 10,000 validation images. The resolution of the images is 64×64 pixels. **DVS-CIFAR10** (Li et al., 2017) has the same configuration as CIFAR-10. This discrete event-stream dataset is collected by moving the event-driven camera. We follow the similar data pre-processing protocol and a network architecture used in previous work (Wu et al., 2019) (details in **Supplementary Material C**). **Sequential MNIST** (Le et al., 2015) is the variant of MNIST (LeCun et al., 1998). Instead of showing the whole image to the networks, this dataset presents each pixel in an image pixel by pixel. Our implementation is based on Pytorch (Paszke et al., 2017). We train the networks with standard SGD with momentum 0.9, weight decay 0.0005 and also apply random crop and horizontal flip to input images. The base learning rate is set to 0.3 and we use step-wise learning rate scheduling with a decay factor 10 at 50, 70, and 90% of the total number of epochs. Here, we set the total number of epochs to 120, 240, 90, and 60 for CIFAR-10, CIFAR-100, Tiny-ImageNet, and DVS-CIFAR10, respectively.

4.2. Comparison With Previous Methods

On public datasets, we compare our proposed BNTT method with previous rate-coding based SNN training methods, including ANN-SNN conversion (Cao et al., 2015; Sengupta et al., 2019; Han et al., 2020), surrogate gradient back-propagation (Lee et al., 2020), and hybrid (Rathi et al., 2020) methods. From **Table 1**, we can observe some advantages and disadvantages of each training method. The ANN-SNN conversion method performs better than the surrogate gradient method across all datasets. However, they require large number of time-steps for training and testing, which is energy-inefficient and impractical in a real-time application. The hybrid method aims to resolve this high-latency problem, but it still requires over hundreds of time-steps. The surrogate gradient method (denoted as *Baseline*) suffers from poor optimization and hence cannot be scaled to larger datasets such as CIFAR-100 and Tiny-ImageNet. The results show that the performance improvement of SNN models is because of BNTT, and not because of applying the loss to the membrane potential which can improve the performance of SNNs (Eshraghian et al., 2021). Using standard BN with surrogate gradient training (i.e., *Baseline + standard BN*) improves the optimization capability of SNNs enabling us to train deep SNNs for complex datasets, however, there is performance degradation. Increasing the number of time-steps to $> 100 - 150$ does improve the performance, but that would also lead to increased computation. Our BNTT is based on the surrogate gradient method (i.e., Baseline + BNTT), and it enables SNNs to achieve high performance even for more complicated datasets. At the same time, we reduce the latency due to the inclusion of learnable parameters and temporal statistics in the BNTT layer. As a result, BNTT can be trained with 25 time-steps on a simple CIFAR-10 dataset, while preserving state-of-the-art accuracy. For CIFAR-100, we achieve about 40× and 2× faster inference speed compared to the conversion

TABLE 1 | Classification accuracy (%) on CIFAR-10, CIFAR-100, and Tiny-ImageNet.

	Dataset	Training method	Architecture	Time-steps	Accuracy (%)
Cao et al. (2015)	CIFAR-10	ANN-SNN Conversion	3Conv, 2Linear	400	77.4
Sengupta et al. (2019)	CIFAR-10	ANN-SNN Conversion	VGG16	2500	91.5
Lee et al. (2020)	CIFAR-10	Surrogate Gradient	VGG9	100	90.4
Rathi et al. (2020)	CIFAR-10	Hybrid	VGG16	200	92.0
Han et al. (2020)	CIFAR-10	ANN-SNN Conversion	VGG16	2048	93.6
Baseline	CIFAR-10	Surrogate Gradient	VGG9	100	88.7
Baseline + standard BN	CIFAR-10	Surrogate Gradient	VGG9	25	84.3
Baseline + BNTT (ours)	CIFAR-10	Surrogate Gradient	VGG9	25	90.5
Baseline + BNTT + Early Exit (ours)	CIFAR-10	Surrogate Gradient	VGG9	20	90.3
Sengupta et al. (2019)	CIFAR-100	ANN-SNN Conversion	VGG16	2500	70.9
Rathi et al. (2020)	CIFAR-100	Hybrid	VGG16	125	67.8
Han et al. (2020)	CIFAR-100	ANN-SNN Conversion	VGG16	2048	70.9
Baseline	CIFAR-100	Surrogate Gradient	VGG11	n/a	n/a
Baseline + standard BN	CIFAR-100	Surrogate Gradient	VGG11	50	43.0
Baseline + BNTT (ours)	CIFAR-100	Surrogate Gradient	VGG11	50	66.6
Baseline + BNTT + Early Exit (ours)	CIFAR-100	Surrogate Gradient	VGG11	30	65.8
Sengupta et al. (2019)	Tiny-ImageNet	ANN-SNN Conversion	VGG11	2500	54.2
Baseline	Tiny-ImageNet	Surrogate Gradient	VGG11	n/a	n/a
Baseline + standard BN	Tiny-ImageNet	Surrogate Gradient	VGG11	30	32.7
Baseline + BNTT (ours)	Tiny-ImageNet	Surrogate Gradient	VGG11	30	57.8
Baseline + BNTT + Early Exit (ours)	Tiny-ImageNet	Surrogate Gradient	VGG11	25	56.8

methods and the hybrid method, respectively. Interestingly, for Tiny-ImageNet, BNTT achieves better performance and shorter latency compared to previous conversion method. Note that ANN with VGG11 architecture used for ANN-SNN conversion achieves 56.3% accuracy. Moreover, using an early exit algorithm further reduces the latency by $\sim$ 20%, which enables the networks to be implemented with lower-latency and energy-efficiency. It is worth mentioning that surrogate gradient method without BNTT (*Baseline* in **Table 1**) only converges on CIFAR-10. For neuromorphic DVS-CIFAR10 dataset (**Table 2**), using BNTT improves the stability of training compared to a surrogate gradient baseline, and achieves state-of-the-art performance. These results show that our BNTT technique is very effective on event-driven data and hence well-suited for neuromorphic applications. We also compare the performance of BNTT with previous works on Sequential MNIST in **Table 3**. Here, we use 3-layer SNN architecture: FC(1,256)-FC(256,256)-FC(256,10). Without BNTT, *Baseline* has difficulty in capturing the sequential pattern of input data, resulting in low performance. Adding BNTT to *Baseline* enhances the training capability of SNNs, resulting in a slightly better performance than the state-of-the-art (Bellec et al., 2018).

TABLE 2 | Classification accuracy (%) on DVS-CIFAR10.

Method	Type	Accuracy (%)
Orchard et al. (2015)	Random forest	31.0
Lagorce et al. (2016)	HOTS	27.1
Sironi et al. (2018)	HAT	52.4
Sironi et al. (2018)	Gabor-SNN	24.5
Wu et al. (2019)	Surrogate gradient	60.5
Baseline	Surrogate gradient	n/a
Baseline + BNTT (ours)	Surrogate gradient	63.2

TABLE 3 | Classification accuracy (%) on sequential MNIST.

Method	Accuracy (%)
LIF (Bellec et al., 2018)	63.3
LSNN (Bellec et al., 2018)	93.7
DEEP R LSNN (Bellec et al., 2018)	96.4
Baseline	36.2
Baseline + BNTT (ours)	96.6

4.3. Comparison With the Previous BN Techniques for SNNs

We compare our temporal BNTT technique with the previous BN approaches for SNN in **Table 4**. The approaches with the standard BN (Fang et al., 2020; Ledinauskas et al., 2020) do not show scalability to complicated datasets such as CIFAR-100 and Tiny-ImageNet. Compared to this, our approach enables training SNNs with low latency on such datasets. Zheng et al. (2020) show the advantage of scaling BN parameter according to the firing threshold, which shows good performance for large-scale

TABLE 4 | Comparison between different BN techniques for SNNs.

	Method	CIFAR-10	CIFAR-100	Tiny-imageNet	ImageNet
Ledinauskas et al. (2020)	Standard BN	90.2	58.5	-	-
Fang et al. (2020)	Standard BN	93.5	-	-	-
Zheng et al. (2020)	Threshold-dependent BN	93.2	-	-	67.1
BNTT (ours)	Temporal BN	90.5	66.6	57.8	-

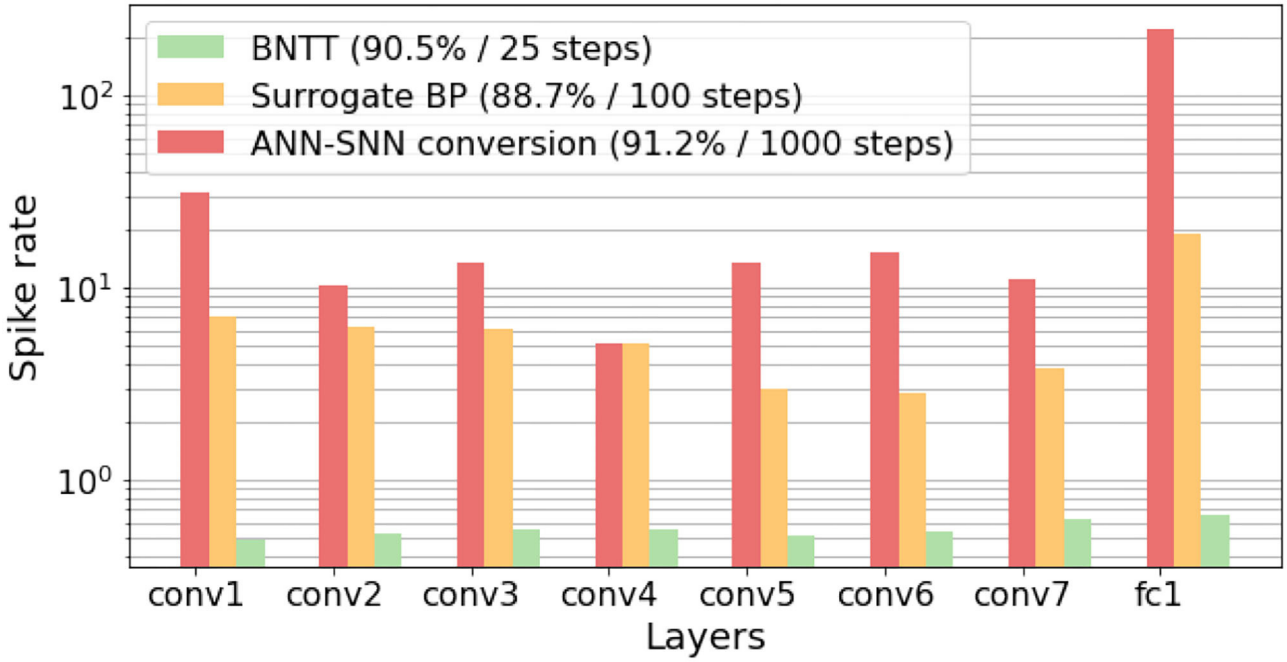

FIGURE 5 | Visualization layer-wise spike activity (log scale) in VGG9 on CIFAR-10 dataset.

datasets, including ImageNet. Our objective is to study the effect of BN in temporal domain, not enhance the capability of BN itself, which is different from their approach. Combining these two orthogonal approaches in order to achieve further performance gain can be a good topic for future work.

4.4. Spike Activity Analysis

We compare the layer-wise spiking activities of our BNTT with two widely-used methods, i.e., ANN-SNN conversion method (Sengupta et al., 2019) and surrogate gradient method (without BNTT) (Neftci et al., 2019). Specifically, we calculate the spike rate of each layer l, which can be defined as the total number of spikes at layer l over total time-steps T divided by the number of neurons in layer l (see **Supplementary Material D** for the equation of spike rate). In **Figure 5**, converted SNNs show a high spike rate for every layer as they forward spike trains through a larger number of time-steps compared to other methods. Even though the surrogate gradient method uses less number of time-steps, it still requires nearly hundreds of spikes for each layer. Compared to these methods, we can observe that BNTT significantly improves the spike sparsity across all layers. In addition, we conduct further energy comparison on Neuromorphic architecture in **Supplementary Material E**.

4.5. Analysis on Learnable Parameters in BNTT

The key observation of our work is the change of γ across time-steps. To analyze the distribution of the learnable parameters in our BNTT, we visualize the histogram of γ in conv1, conv4, and conv7 layers in VGG9 as shown in **Figure 6**. Interestingly, all layers show different temporal evolution of gamma distributions. For example, conv1 has high γ values at the initial time-steps which decrease as time goes on. On the other hand, starting from small values, the γ values in conv4 and conv7 layers peak at $t = 9$ and $t = 13$, respectively, and then shrink to zero at later time-steps. Notably, the peak time is delayed as the layer goes deeper, implying that the visual information is passed through the network sequentially over a period of time similar to **Figure 1C**. This gaussian-like trend with rise and fall of γ across different time-steps can support the explanation of overall low spike activity compared to other methods (**Figure 5**).

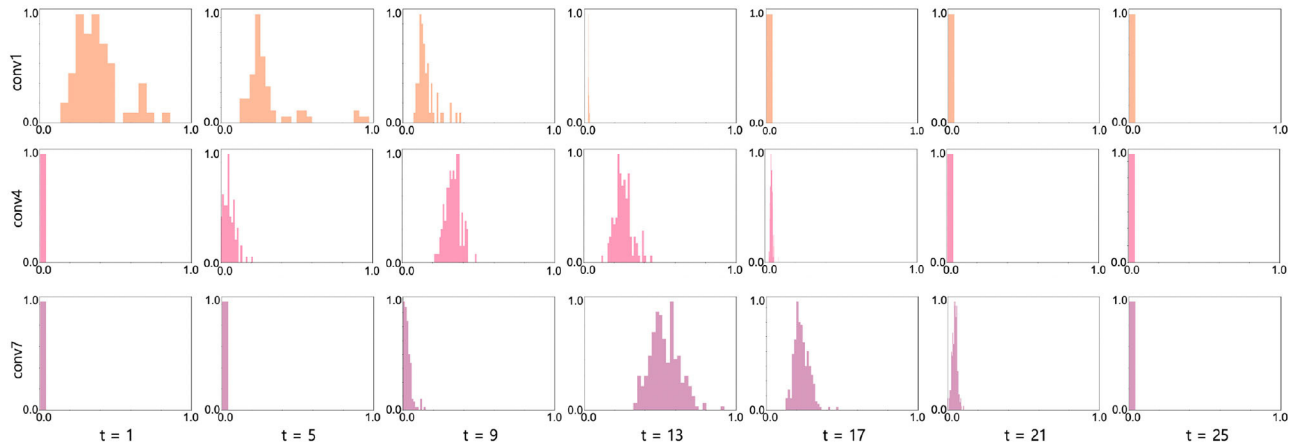

FIGURE 6 | Histogram visualization (x axis: γ value, y axis: frequency) at conv1 (row1), conv4 (row2), and conv7 (row3) layers in VGG9 across all time-steps. We normalize the frequency into range [0, 1] for better visualization. The experiments are conducted on CIFAR-10 with 25 time-steps.

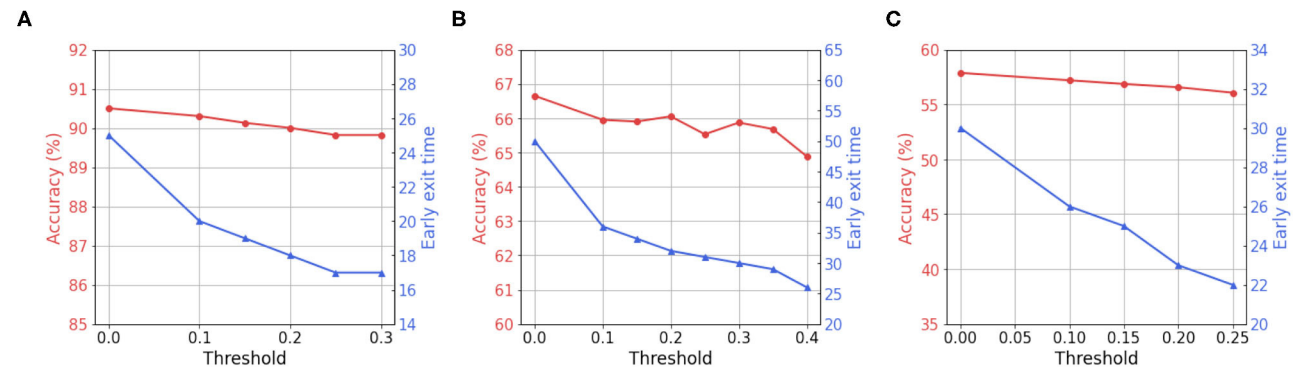

FIGURE 7 | Visualization of accuracy and early exit time with respect to the threshold value for γ. **(A)** CIFAR-10. **(B)** CIFAR-100. **(C)** Tiny-ImageNet.

4.6. Analysis on Early Exit

Recall that we measure the average of γ values in each layer at every time-step, and stop the inference when all γ values in every layer is lower than a predetermined threshold. To further investigate this, we vary the predetermined threshold and show the accuracy and exit time T_{exit} trend. As shown in **Figure 7**, we observe that high threshold enables the networks to infer at earlier time-steps. Although we use less number time-steps during inference, the accuracy drops marginally. This implies that BNTT rarely sends crucial information at the end of spike train (see **Figure 1C**). Note that the temporal evolution of learnable parameter γ with our BNTT allows us to exploit the early exit algorithm that yields a huge advantage in terms of reduced latency at inference. Such strategy has not been proposed or explored in any prior works that have mainly focused on reducing the number of time-steps during training without effectively using temporal statistics.

4.7. Analysis on Robustness

Finally, we highlight the advantage of BNTT in terms of the robustness to noisy input. To investigate the effect of our BNTT for robustness, we evaluate the performance change in the SNNs as we feed in inputs with varying levels of noise. We generate the noisy input by adding Gaussian noise $(0, \sigma^2)$ to the clean input image. From **Figure 8A**, we observe the following: (i) The accuracy of conversion method degrades considerably for $\sigma > 0.4$. (ii) Compared to ANNs, SNNs trained with surrogate gradient back-propagation shows better performance at higher noise intensity. Still, they suffer from large accuracy drops in presence of noisy inputs. (iii) BNTT achieves significantly higher performance than the other methods across all noise intensities. This is because using BNTT decreases the overall number of time-steps which is a crucial contributing factor toward robustness (Sharmin et al., 2020). These results imply that, in addition to low-latency and energy-efficiency, our BNTT method also offers improved robustness for suitably implementing SNNs in a real-world scenario.

In order to further validate the robustness of BNTT, we conduct experiments on adversarial inputs. We use FGSM (Goodfellow et al., 2014) to generate adversarial samples for ANN. For a given image x, we compute the loss function $\mathcal{L}(x, y)$ with the ground truth label y. The objective of FGSM attack is to

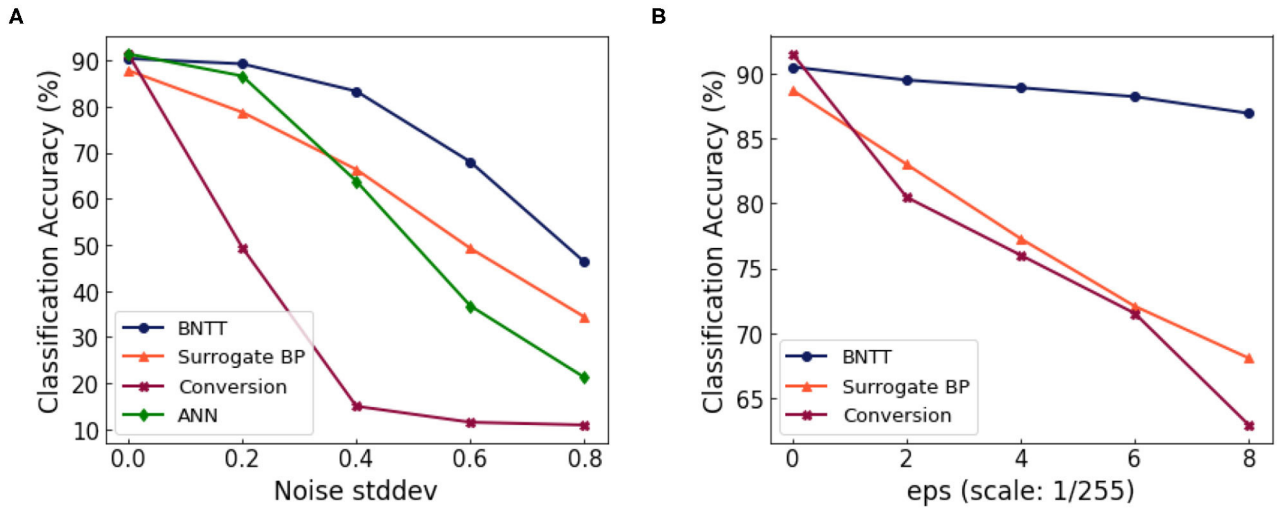

FIGURE 8 | (A) Performance change with respect to the standard deviation of the Gaussian noise. **(B)** Performance change with respect to the attack intensity (ϵ, denoted in x-axis) of the FGSM attack.

change the pixel intensity of the input image that maximizes the cost function:

$$x_{adv} = x + \epsilon \times sign(\nabla_x \mathcal{L}(x, y)). \tag{16}$$

We call x_{adv} as "adversarial sample." Here, ϵ denotes the strength of the attack. To conduct the FGSM attack for SNN, we use the SNN-crafted FGSM method proposed in Sharmin et al. (2020). In **Figure 8B**, we show the classification performance for varying intensities of FGSM attack. The SNN approaches (e.g., BNTT and Surrogate BP) show more robustness than ANN due to the temporal dynamics and stochastic neuronal functionality. We highlight that our proposed BNTT shows much higher robustness compared to others. Thus, we assert that BNTT improves robustness of SNNs in addition to energy efficiency and latency.

4.8. Comparison With Layer Norm

Layer Normalization (LN) (Ba et al., 2016) is an optimization method for recurrent neural networks (RNNs). The authors asserted that directly applying BN layers is hardly applicable since RNNs vary with the length of the input sequence. To this end, an LN layer calculates the mean and the variance for every single layer. As SNNs also take time-sequence data as input, we compare our BNTT with Layer Normalization in **Table 5**. For all experiments, we use a VGG9 architecture. Also, we set a base learning rate to 0.3 and we use step-wise learning rate scheduling as described in section 4.1. The results show that BNTT is more suitable structure to capture the temporal dynamics of Poisson encoded spikes.

TABLE 5 | Comparison with layer normalization on CIFAR-10 dataset.

Method	Acc (%)
Layer normalization (Ba et al., 2016)	75.4
BNTT	90.5

5. CONCLUSION

In this paper, we revisit the batch normalization technique and propose a novel mechanism for training low-latency, energy-efficient, robust, and accurate SNNs from scratch. Our key idea is to investigate the temporal characteristics of Batch Normalization (BN) with time-specific learnable parameters and statistics. Note, BN is known as an effective way of addressing vanishing/exploding gradients problem in ANNs. We discover that optimizing time-dependent learnable parameters γ captures the temporally varying input distribution so that it stabilizes the backward gradients during training and enables better learning of SNN representations. Our experiments reveal interesting benefits of BNTT for temporal early exit during inference as well as sturdy robustness against adversarial attacks. As previous SNN-based BN works (Fang et al., 2020; Ledinauskas et al., 2020; Zheng et al., 2020), this work showcases the importance of incorporating dynamic time-dependent parameters during surrogate gradient-based training to enable large-scale SNN implementations. By showing the importance of addressing the unstable gradient problem in SNN, we suggest future direction for better SNN training. Today, SNNs have few advanced optimization techniques (such as, weight initialization, skip connection that are common in ANN optimization suite) for addressing such issues. Our proposed BNTT can be considered to

be one SNN-crafted optimization technique that can relieve the gradient problem, resulting in performance improvement. We hope this work fosters future work on advanced SNN optimization.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/**Supplementary Material**, further inquiries can be directed to the corresponding author.

AUTHOR CONTRIBUTIONS

YK and PP conceived the work and contributed to the writing of the manuscript. YK carried out experiments. Both authors contributed to the article and approved the submitted version.

FUNDING

This work was supported in part by the Center for Brain-inspired Computing (C-BRIC) which is a JUMP center sponsored by DARPA and SRC, the National Science Foundation (Grant#1947826), the Technology Innovation Institute, Abu Dhabi and the Amazon Research Award.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2021.773954/full#supplementary-material

REFERENCES

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., et al. (2015). Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Trans. Comput. Aided Design Integr. Circ. Syst.* 34, 1537–1557. doi: 10.1109/TCAD.2015.2474396

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint* arXiv:1607.06450.

Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., and Maass, W. (2018). Long short-term memory and learning-to-learn in networks of spiking neurons. *arXiv preprint* arXiv:1803.09574.

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

Cao, Y., Chen, Y., and Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *Int. J. Comput. Vis.* 113, 54–66. doi: 10.1007/s11263-014-0788-3

Comsa, I. M., Fischbacher, T., Potempa, K., Gesmundo, A., Versari, L., and Alakuijala, J. (2020). "Temporal coding in spiking neural networks with alpha synaptic function," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (Barcelona: IEEE), 8529–8533.

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Dayan, P., and Abbott, L. F. (2001). *Theoretical Neuroscience*. vol. 806.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). "Imagenet: a large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition* (Miami, FL: IEEE), 248–255.

Diehl, P. U., and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* 9:99. doi: 10.3389/fncom.2015.00099

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

Eshraghian, J. K., Ward, M., Neftci, E., Wang, X., Lenz, G., Dwivedi, G., et al. (2021). Training spiking neural networks using lessons from deep learning. *arXiv preprint* arXiv:2109.12894.

Fang, W., Yu, Z., Chen, Y., Masquelier, T., Huang, T., and Tian, Y. (2020). Incorporating learnable membrane time constant to enhance learning of spiking neural networks. *arXiv preprint* arXiv:2007.05785.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint* arXiv:1412.6572.

Han, B., Srinivasan, G., and Roy, K. (2020). "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* (IEEE), 13558–13567.

Ioffe, S., and Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift. *arXiv preprint* arXiv:1502.03167.

Krizhevsky, A., and Hinton, G. (2009). Learning multiple layers of features from tiny images.

Lagorce, X., Orchard, G., Galluppi, F., Shi, B. E., and Benosman, R. B. (2016). Hots: a hierarchy of event-based time-surfaces for pattern recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* 39, 1346–1359. doi: 10.1109/TPAMI.2016.2574707

Le, Q. V., Jaitly, N., and Hinton, G. E. (2015). A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint* arXiv:1504.00941.

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

Ledinauskas, E., Ruseckas, J., Juršėnas, A., and Buračas, G. (2020). Training deep spiking neural networks. *arXiv preprint* arXiv:2006.04436.

Lee, C., Sarwar, S. S., Panda, P., Srinivasan, G., and Roy, K. (2020). Enabling spike-based backpropagation for training deep neural network architectures. *Front. Neurosci.* 14:119. doi: 10.3389/fnins.2020.00119

Lee, J. H., Delbruck, T., and Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Front. Neurosci.* 10:508. doi: 10.3389/fnins.2016.00508

Li, H., Liu, H., Ji, X., Li, G., and Shi, L. (2017). Cifar10-dvs: an event-stream dataset for object classification. *Front. Neurosci.* 11:309. doi: 10.3389/fnins.2017.00309

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks. *IEEE Signal. Process. Mag.* 36, 61–63. doi: 10.1109/MSP.2019.2931595

Orchard, G., Meyer, C., Etienne-Cummings, R., Posch, C., Thakor, N., and Benosman, R. (2015). Hfirst: a temporal approach to object recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* 37, 2028–2040. doi: 10.1109/TPAMI.2015.2392947

Panda, P., Aketi, S. A., and Roy, K. (2020). Toward scalable, efficient, and accurate deep spiking neural networks with backward residual connections, stochastic softmax, and hybridization. *Front. Neurosci.* 14:653. doi: 10.3389/fnins.2020.00653

Panda, P., Sengupta, A., and Roy, K. (2016). "Conditional deep learning for energy-efficient and enhanced pattern recognition," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)* (Dresden: IEEE), 475–480.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., et al. (2017). "Automatic differentiation in pytorch," in *NIPS-W*. Long Beach, CA.

Rathi, N., Srinivasan, G., Panda, P., and Roy, K. (2020). Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. *arXiv preprint* arXiv:2005.01807.

Roy, K., Jaiswal, A., and Panda, P. (2019). Towards spike-based machine intelligence with neuromorphic computing. *Nature* 575, 607–617. doi: 10.1038/s41586-019-1677-2

Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., and Liu, S.-C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Front. Neurosci.* 11:682. doi: 10.3389/fnins.2017.00682

Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. (2018). "How does batch normalization help optimization?" in *Advances in Neural Information Processing Systems* (Montreal, CA), 2483–2493.

Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. (2019). Going deeper in spiking neural networks: Vgg and residual architectures. *Front. Neurosci.* 13:95. doi: 10.3389/fnins.2019.00095

Sharmin, S., Rathi, N., Panda, P., and Roy, K. (2020). Inherent adversarial robustness of deep spiking neural networks: effects of discrete input encoding and non-linear activations. *arXiv preprint* arXiv:2003.10399. doi: 10.1007/978-3-030-58526-6_24

Sironi, A., Brambilla, M., Bourdis, N., Lagorce, X., and Benosman, R. (2018). "Hats: histograms of averaged time surfaces for robust event-based object classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Salt Lake City, UT: IEEE), 1731–1740.

Teerapittayanon, S., McDanel, B., and Kung, H.-T. (2016). "Branchynet: fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)* (Cancun: IEEE), 2464–2469.

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* 12:331. doi: 10.3389/fnins.2018.00331

Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., and Shi, L. (2019). Direct training for spiking neural networks: faster, larger, better. *Proc. AAAI Conf. Artif. Intell.* 33, 1311–1318. doi: 10.1609/aaai.v33i01.33011311

Zheng, H., Wu, Y., Deng, L., Hu, Y., and Li, G. (2020). Going deeper with directly-trained larger spiking neural networks. *arXiv preprint* arXiv:2011.05280.

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

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.