Going Deeper in Spiking Neural Networks via Conversion and Hybrid Training

Gourav Datta*, Muhammad Waqas*, Akshay Manjunath*
{gdatta, waqas, kakshay}@usc.edu
Group 10, mentored by Souvik Kundu
Ming Hsieh Department of Electrical and Computer Engineering
University of Southern California
Los Angeles, California 90089, USA

***Abstract*—Spiking Neural Networks (SNNs) have become popular in recent times, owing to their high energy-efficiency in neuromorphic hardware implementations. Recent works show that an SNN inference engine can be formed by copying the weights from a trained Artificial Neural Network (ANN) and estimating the firing threshold for each layer based on the trained weights. These type of converted SNNs require a large number of time steps ($\sim$2500) to achieve competitive accuracy which again increases the energy and memory cost. Training of SNNs via spike-based backpropagation from scratch can greatly reduce the number of timesteps. However, the training effort in SNNs is two orders of magnitude higher than traditional ANN because SNN requires more resources to iterate over multiple time steps and store the membrane potential for each neuron. To address these challenges, we present two energy-efficient conversion and training techniques for deep SNNs. We first propose a conversion technique which estimates the firing thresholds as a scaled version of the maximum input received in that layer. We show such a technique is able to achieve competitive accuracy with VGG architectures on CIFAR-10 dataset. Additionally, we propose a hybrid training methodology which takes a converted SNN and use its weights as an initialization step and performs incremental spike-timing dependent backpropagation (STDB) on this carefully initialized network to obtain an SNN that converges within few epochs. STDB is performed with a novel surrogate gradient function defined using neuron's spike time. The weight update is proportional to the difference in spike timing between the current time step and the most recent time step the neuron generated an output spike. The SNNs trained with our hybrid conversion-and-STDB training perform at 10-25 times fewer time-steps and achieve similar accuracy as compared to purely converted SNNs. We perform experiments on CIFAR-10 and MNIST datasets for both VGG and LENET architectures. We achieve top-1 accuracy of 92.44% for CIFAR-10 dataset on SNN with 200 time steps, which is 10× faster compared to trained SNNs with similar accuracy.**

***Index Terms*—SNN, ANN, backpropagation**

I. Introduction

Spiking Neural Networks(SNNs) mimic a human neuron using biologically-realistic models of neurons to carry out computation [1]. Spikes or binary events drive the communication and computations in SNNs via asynchronous sweeps that offer the benefit of event driven hardware operations [2]. SNNs offer a low power alternative to conventional Artificial Neural Networks (ANNs) [3]. This makes them attractive for deployments in edge devices which are limited by power consumption and memory bandwidth.

With all their appeal for power efficiency, training SNNs still remains a challenge [4]. The discontinuous and non-differentiable nature of a spiking neuron poses difficulty to conduct gradient descent based backpropagation, and hence, SNNs tend to perform poorly compared to their non-spiking counterparts. Several works over the past few years have proposed different algorithms and learning rules for implementing deep convolutional spiking architectures for complex visual recognition tasks [4] [5] . Among all the techniques that have been explored, conversion from a trained ANN model gives the best classification accuracies [6]. The performance of such models are somewhat comparable to deep ANN performance on image recognition applications for datasets like MNIST, CIFAR and ImageNet. However, these type of converted SNNs require large number of time steps to achieve competitive accuracies leading to higher energy consumption [7].

Our main goal in this paper is to present a computationally efficient training technique for deep SNNs. We first focus on training of an ANN model, with constraints incorporated for training/inference in SNN. We present a hybrid training technique which combines ANN-SNN conversion and spike-based backpropagation that reduces the overall latency as well as decreases the training effort for convergence. We use the same weights as that obtained from a trained ANN model and estimate the firing thresholds based on the maximum activation achieved in each layer. We then train this initialized network with spike-based backpropagation for few epochs to perform inference at a reduced latency or time steps. We evaluate our hybrid approach on MNIST and CIFAR10 datasets for LeNet5 and VGG architectures.

II. Neural Spike Dynamics : Background

Leaky-Integrate-and-Fire (LIF) neurons are fundamental computational elements integral to the dynamics of Spiking Neural Networks [8]. SNNs operate using spikes, which are discrete events that take place in a temporal fashion. The occurrence of a spike represents the membrane potential of the neuron. Once a neuron reaches a certain potential, it produces a spike, and the potential of that neuron is reset. Figure 1

*All authors have equal contribution.

illustrates a basic SNN architecture with Leaky-Integrate-and-Fire (LIF) spiking dynamics.

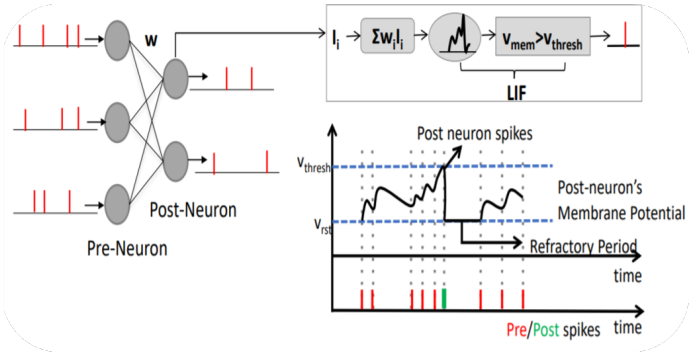

Fig. 1. A feedforward fully-connected SNN architecture with Leaky-Integrate-and-Fire (LIF) spiking dynamics.

The dynamics of the neuron's internal state are described by the neuron model as shown in Figure 2. The differential equation widely used to characterize the leaky-integrate-and-fire (LIF) neuron model is described by

$$\tau \frac{dU}{dt} = -(U - U_{rest}) + \sum_i I_i w_i \tag{1}$$

where U is the internal state of the neuron called the membrane potential and U_{rest} is the resting potential. The pre-spikes obtained from the previous layer are modulated by the synaptic weights to be integrated as the current influx in the membrane potential that decays exponentially. The membrane potential integrates the incoming spikes through the weights wi and leaks with time constant τ whenever it does not receive a spike. When the membrane potential crosses the firing threshold, the neuron generates an output spike, and the membrane potential is reduced to the reset potential. A refractory period ensues after spike generation during which the post neuron's membrane potential is not affected.

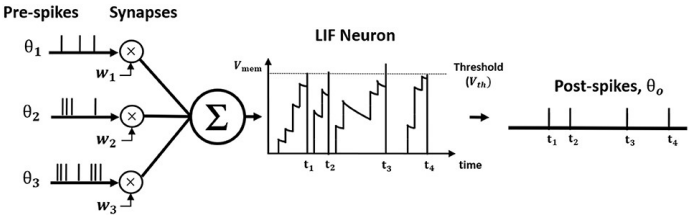

Fig. 2. Illustration of Leaky Integrate and Fire (LIF) neuron dynamics.

We modify Eq. 1 to be evaluated in the Pytorch framework. The iterative model for a single post-neuron is described by

$$u_i^t = \lambda u_i^{t-1} + \sum_j \omega_{ij} o_j^t - v o_i^{t-1} \tag{2}$$

$$o_i^{t-1} = \begin{cases} 1, & if\ u_i^{t-1} > v \\ 0, & otherwise \end{cases} \tag{3}$$

where u is the membrane potential, subscript i and j represent the post- and pre-neuron, respectively, superscript t is the time step, λ is a constant (<1) responsible for the leak in membrane potential, w is the weight connecting the pre- and post-neuron, o is the binary output spike, and v is the firing threshold potential. Here, we calculate the leak in the membrane potential from the previous time step, and then integrate the input from the previous layer and add it to the membrane potential. Next, we reduce the membrane potential by the threshold value if a spike is generated. This is known as soft reset as the membrane potential is lowered by the firing threshold compared to hard reset where the membrane potential is reduced to the reset value. Soft reset enables the spiking neuron to carry forward the excess potential above the firing threshold to the following time step, thereby minimizing information loss.

The dynamics of the neuron in the output layer is described by Eq. 4, where the leak part is removed and the neuron only integrates the input without firing. This eliminates the difficulty of defining the loss function on the membrane potential. This type of information encoding is very robust to errors because a failure to detect a few spikes does not induce a significant error on the average spike rate, and thereby on the accuracy.

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

III. ANN to SNN Conversion

The goal here is to match the input-output mapping function of the trained ANN to that of SNN. We propose to use the ANN–SNN conversion process as an initialization technique for STDB. In this work, we use the threshold balancing method that yields almost zero loss ANN to SNN conversion performance for deep VGG like architectures.

After obtaining the trained ANN, we first copy the weights from the ANN to the SNN and initialize the threshold voltages for all the layers to zero. The threshold voltages in SNN are estimated based on the ANN weights. To do so, the first step is to generate a Poisson spike train corresponding to the entire training dataset for a large simulation duration or time period. The Poisson spike train allows us to record the maximum summation of weighted spike input received by the first layer of the ANN. The threshold value for the first layer is then set to normalization factor which corresponds to the maximum neuron activation for the corresponding convolution/linear layer in SNN. After the threshold for the first layer is set, the input spike train is again fed to the network to obtain a spike-train at the first layer, which serves as the input spike-stream for the second layer of the network. This process of generating spike train and setting threshold value is repeated for all layers of the network. Note, the weights during this balancing process remain unchanged. We employ two approaches to get the value of the threshold voltages namely baseline conversion and scaled conversion. In the baseline conversion approach, we set the maximum activation for each layer to the threshold voltage. This is done for ANN – SNN conversion which involves

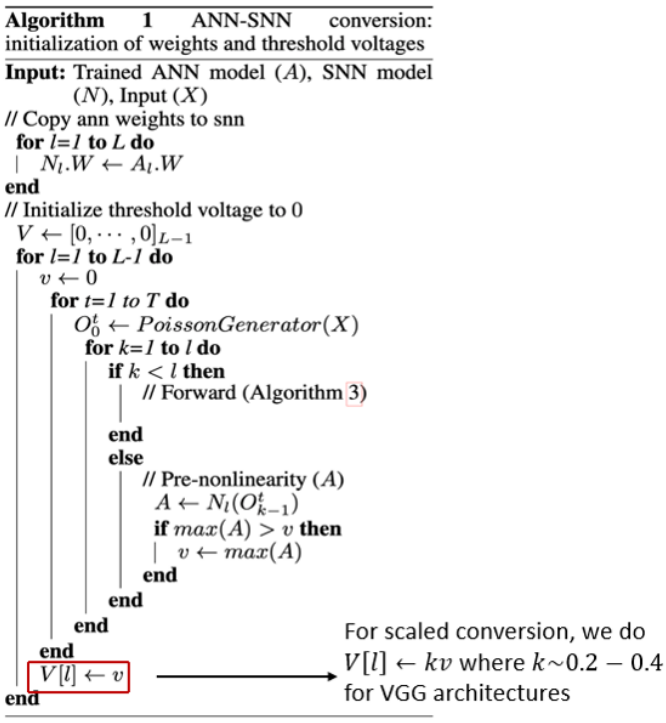

Fig. 3. ANN-SNN Conversion Algorithm illustrating the initialization of weights and threshold voltages, obtained from [3].

large time steps. In the scaled conversion approach, we scale the value of the threshold voltage at each layer by k where $k \in [0.2, 0.4]$ for VGG architectures. This enables ANN–SNN conversion with less number of time steps.

IV. Spike Timing Dependent Backpropagation (STDB)

The neuron dynamics described in (2) and (3) show that neurons are recurrent networks and each next state depends upon the previous state. So the backpropagation has to be done in temporal as well as special dimensions. In other words, the learning rule must perform both temporal as well as special credit assignment. Let us denote L as the loss function, y as the true output, p as the prediction, T as the total number of time-steps, u^T as the accumulated membrane potential of the neuron in the output layer from all time steps, and N as the number of categories in the task. The number of neurons in the output layer is same as the number of classes. The dynamical model of the neuron in the output layer is given in 5. The output neuron only integrates input without firing.

$$u_i^t = u_i^{t-1} + \sum_j \omega_{ij} o_j \tag{5}$$

Softmax layer is used on the output of the network to use cross-entropy as the loss function. The loss function is defined in (6) as the cross-entropy between the target output and the prediction.

$$L = \sum_i y_i log(p_i) \tag{6}$$

$$p_i = \frac{e^{u_i^T}}{\sum_{k=1}^{N} e^{u_k^T}} \tag{7}$$

The derivative of the loss function is given in (8).

$$\frac{\partial L}{\partial u_i^T} = p_i - y_i \tag{8}$$

The learning rule for STDB based backpropagation is derived in (9), (10), (11).

$$W_{ij} = W_{ij} - \eta \Delta W_{ij} \tag{9}$$

$$\begin{aligned} \Delta W_{ij} &= \sum_t \frac{\partial L}{\partial W_{ij}^t} = \sum_t \frac{\partial L}{\partial u_i^T} \frac{\partial u_i^T}{\partial W_{ij}^t} \\ &= \frac{\partial L}{\partial u_i^T} \sum_t \frac{\partial u_i^T}{\partial W_{ij}^t} \end{aligned} \tag{10}$$

$$\Delta W_{ij} = \sum_t \frac{\partial L}{\partial W_{ij}^t} = \sum_t \frac{\partial L}{\partial o_i^t} \frac{\partial o_i^t}{\partial u_i^t} \frac{\partial u_i^t}{\partial W_{ij}^t} \tag{11}$$

As the spiking neural network's output is not differentiable, a surrogate gradient function is used which is the continuous approximation of the real gradient.

$$\frac{\partial o_i^t}{\partial u_i^t} = \alpha e^{-\beta \partial t} \tag{12}$$

In the above equation, α and β are constant, δt is the time difference between the current time step (t) and the last time step the post-neuron generated a spike (ts). The Fig. 4 describes the algorithm [8] for training SNN using STDB based backpropagation. It takes a mini batch of inputs X, target values Y, initial weights and threshold voltages. It first performs forward propagation and then updates the weight through STDB learning rule shown in Fig. 4.

V. Network Architectures

We have used LeNet and VGG architectures for training. The traditional versions of these architectures are described below:

A. LENET Architecture

The traditional LeNet5 architecture is described in the Fig. 1 and Fig. 2. It has three convolution layers with 2X2 kernal size, two average pooling layers, two fully connected layers. Softmax is used for the output.

B. VGG Architecture

VGG16 is a famous convolution neural net (CNN) for image classification problems. It has 16 layers with approximately 138 million parameters.

Algorithm 2 Training a SNN with surrogate gradient computed with spike timing. The network is composed of L layers. The training proceeds with mini-batch size ($batch\_size$)

Input: Mini-batch of input (X) - target (Y) pairs, network model (N), initial weights (W), threshold voltage (V)

$U, S, M = InitializeNeuronParameters(X)$

// Forward propagation

for *t=1* **to** *T* **do**

  $O_0^t = PoissonGenerator(X)$

  for *l=1* **to** *L-1* **do**

    if $isintance(N_l, [Conv, Linear])$ **then**

      // accumulate the output of previous layer in U, soft reset when spike occurs

      $U_l^t = \lambda U_l^{t-1} + W_l O_{l-1}^t - V_l * O_l^{t-1}$

      // generate the output (+1) if U exceeds V

      $O_l^t = STDB(U_l^t, V_l, t)$

      // store the latest spike times for each neuron

      $S_l^t[O_l^t == 1] = t$

    end

    else if $isintance(N_l, AvgPool)$ **then**

      $O_l^t = N_l(O_{l-1}^t)$

    end

    else if $isintance(N_l, Dropout)$ **then**

      $O_l^t = O_{l-1}^t * M_l$

    end

  end

  $U_L^t = \lambda U_L^{t-1} + W_L O_{L-1}^t$

end

// Backward Propagation

Compute $\frac{\partial L}{\partial U_L}$ from the cross-entropy loss function using BPTT

for *t=T* **to** *1* **do**

  for *l=L-1* **to** *1* **do**

    Compute $\frac{\partial L}{\partial O_l^t}$ based on if N_l is linear, conv, pooling, etc.

    $\frac{\partial L}{\partial U_l^t} = \frac{\partial L}{\partial O_l^t}\frac{\partial O_l^t}{\partial U_l^t} = \frac{\partial L}{\partial O_l^t} * \alpha e^{-\beta S_l^t}$

  end

end

Fig. 4. Algorithm illustrating the SNN Training with STDB based backpropagation with proper initialization of parameters, obtained from [3].

C. Constraints on ANN Training

For efficient conversion from ANN to SNN, several constraints are proposed by researchers in literature [6], [9]. These constraints are described below:

- No Batch Normalization: Introducing bias term makes difficulty in balancing threshold voltages in SNN. So bias term cannot be used in ANN. Due to the absence of bias term, batch normalization cannot be used as regularization technique.
- Average Pooling instead of Max Pooling: SNN are binary in nature therefore max pooling cannot extract the required information during training. Therefore average pooling is used instead of Max pooling [9]. The revised architecture is shown in Fig 4.

VI. Overall Training Algorithm

Algorithm 1 illustrates the procedure to initialize the weights and estimate the thresholds of SNN. Algorithm 2 explains the procedure to initialize the different parameters

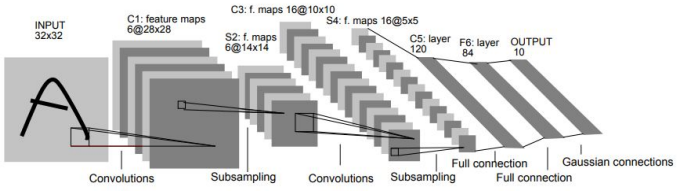

Fig. 5. Complete description of LeNet5 Architecture

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

Fig. 6. LeNet5 Architecture Summary

of the SNN like membrane potential, last spike time, etc. and train the converted SNN with STDB. The threshold voltage for all neurons in a layer (convolutional or linear) remains fixed during the training process. We initialize a mask (M) for every mini-batch of inputs in case of a dropout layer. Dropout is a widely popular regularization technique which randomly removes a certain number of inputs in order to avoid overfitting. In case of SNN, inputs are represented as a spike train and it is desirable to keep the dropout units same for the entire duration of the input. Thus, a random mask (M) is initialized (Algorithm 2) for every mini-batch and the input is element-wise multiplied with the mask to generate the output of the dropout layer [8]. The Poisson generator function outputs a Poisson spike train with frequency of spikes proportional to the inpiut pixel value. A random number is generated at every time step for each pixel in the input image, which is compared with the normalized pixel value. If the random number turns out to be less than the pixel value, an output spike is generated. This results in a Poisson spike train with rate equivalent to the pixel value if a large number of timesteps is used. The weighted sum of the input is accumulated in the membrane potential of the first convolution layer. The STDB function compares the membrane potential and the threshold of that layer to generate an output spike. The neurons that output a spike update their spike time with the current time step. The last spike time is initialized with a large negative number. This is repeated for all layers except the last layer. For the last layer, the inputs are accumulated over all time steps and passed through a softmax layer to compute the multi-class probability. The cross-entropy loss function is defined on the output of the softmax and the weights are updated by performing the temporal and spatial

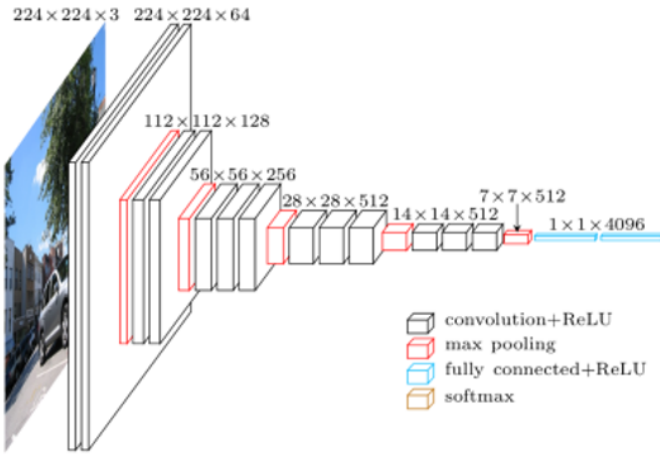

Fig. 7. Complete Description of Traditional VGG16 Architecture

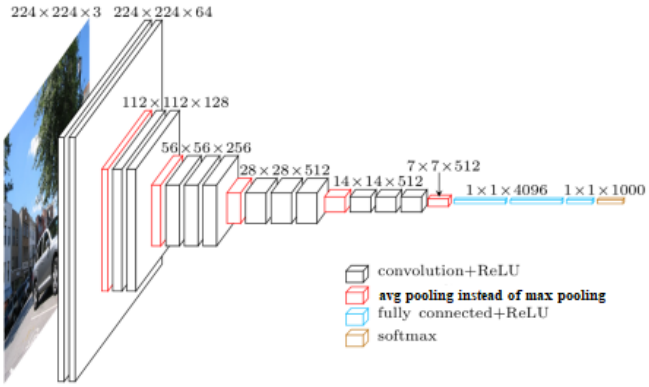

Fig. 8. Architecture of Trained VGG16 Architecture with Constraints

credit assignment according to the STDB rule.

VII. Simulations and Results

We tested the proposed training mechanism on image classification tasks from MNIST [10] and CIFAR-10 [11] datasets. We use LENET-5 and VGG architectures for classifying MNIST and CIFAR-10 datasets respectively. The results are summarized in Fig. 9.

MNIST: The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

CIFAR-10: The dataset consists of labeled 60,000 images of 10 categories divided into training (50,000) and testing (10,000) set. The images are of size 32×32 with RGB channels.

VIII. Related Work

[12] proposed a method to directly train a SNN by keeping track of the membrane potential of spiking neurons only at spike times and backpropagating the error at spike times based

TABLE I
COMPARION OF OUR WORK WITH OTHER SNN MODELS ON CIFAR10 DATASETS

Model	Training Method	Architecture	Accuracy	Timesteps
Hunsberger and Eliasmith (2015)	ANN-SNN Conversion	2-Conv, 2-Linear	82.95	6000
Cao et al. (2015)	ANN-SNN Conversion	3-Conv, 2-Linear	77.43	400
Sengupta et al. (2019)	ANN-SNN Conversion	VGG-16	91.55	2500
Lee et al. (2019)	Spiking BP	VGG-9	90.45	100
Wu et al. (2019)	Surrogate gradient	5-Conv, 2-Linear	90.53	12
This work	ANN-SNN Conversion	VGG-16	91.13	200

Architecture	ANN	ANN-SNN Conversion (T=2500)	ANN-SNN Baseline Conversion (reduced time steps)	Hybrid Training(ANN-SNN conversion+STDB)
CIFAR-10				
VGG5	87.96	86.84	85.97 (T=200)	86.79 (T=200)
			85.46 (T=100)	86.49 (T=100)
VGG9	91.37	91.11	89.96 (T=200)	90.46 (T=200)
			85.12 (T=100)	88.89 (T=100)
VGG16	92.55	92.48	91.13 (T=200)	92.03 (T=200)
			84.64 (T=100)	91.13 (T=100)
MNIST				
LENET5	98.83	98.91	98.82 (T=100)	98.88 (T=100)

Fig. 9. Classification results (Top-1) for CIFAR10 and MNIST datasets. Column-1 shows the network architecture. Column-2 shows the ANN accuracy when trained under the constraints as described in Section VI. Column-3 shows the SNN accuracy for T = 2500 when converted from a ANN with threshold balancing. Column-4 shows the performance of the same converted SNN with lower time steps and adjusted thresholds. Column-5 shows the performance after training the Column-4 network with STDB for less than 20 epochs.

on only the membrane potential. This method is not suitable for networks with sparse activity due to the 'dead neuron' problem, since no learning happens when the neurons do not spike. In our work, we need one spike for the learning to start but gradient contribution continues in later time steps. [13] derived a surrogate gradient based method on the membrane potential of a spiking neuron at a single time step only. The error was backpropagated at only one time step and only the input at that time step contributed to the gradient. This method neglects the effect of earlier spike inputs. In our approach, the error is backpropagated for every time step and the weight update is performed on the gradients summed over all the time steps. [14] proposed a gradient function similar to the one proposed in this work. They used the difference between the membrane potential and the threshold to compute the gradient compared to the difference in spike timing used in this work. The membrane potential is a continuous value whereas the spike time is an integer value bounded by the number of time steps. Therefore, gradients that depend on spike time can be pre-computed and stored in a look-up table for faster computation. They evaluated their approach on shallow architectures with two convolution layer for MNIST dataset. In this work, we trained deep SNNs with multiple stacked layers for complex calssification tasks. [15] performed backpropagation through time on SNN with a surrogate gradient defined on the membrane potential. The surrogate gradient was defined as piece-wise linear or exponential function of the membrane potential. The other surrogate gradients proposed in the literature are all computed on the membrane potential [16]. [8] approximated the neuron output as continuous low-pass filtered spike train. The authors used this approximated continuous value to perform backpropagation. Most of the works in the literature on direct training of SNN or conversion based methods have been evaluated on shallow architectures for simple classification problems. In Table I we compare our model with the models that reported accuracy on CIFAR10 dataset. [17] achieved convergence in 12 time steps by using a dedicated encoding layer to capture the input precision. It is beyond the scope of this work to compute the hardware and energy implications of such encoding layer. Our model performs better than all other models at far fewer number of time steps. Our work is mainly focused on the technique discussed in [3], with the modification that [3] does not describe any scaled conversion technique which can get rid of the expensive SNN STDB and achieve competitive accuracy based on conversion alone.

IX. CONCLUSIONS

Spiking Neural Networks (SNNs) are energy-efficient and can lead to low-power neuromorphic hardware implementations. The direct training of SNN with backpropagation is computationally expensive and slow, whereas ANN-SNN conversion suffers from high latency. To address this issue we proposed a hybrid training technique for deep SNNs. We took an SNN converted from ANN and used its weights and thresholds as initialization for spike-based backpropagation of SNN. We then performed spike-based backpropagation on this initialized network to obtain an SNN that can perform with fewer number of time steps. This training algorithm was tested on VGG and LeNet architectures to train ANN on MNIST and CIFAR10 dataset. The trained ANNs are converted to SNNs via proper thresholding. The SNNs trained with our hybrid conversion-and-STDB training perform at fewer number of time steps and achieve similar accuracy compared to purely converted SNNs. The trained SNN models have accuracy of 98.8% for LeNet on MNIST dataset and 92.44% for VGG on CIFAR10 dataset. Future works include decreasing SNN training/conversion times and scaling the proposed work to large datasets.

REFERENCES

[1] D. Soni, "Spiking neural networks, the next generation of machine learning," 2018.

[2] P. Panda, A. Aketi, and K. Roy, "Towards scalable, efficient and accurate deep spiking neural networks with backward residual connections, stochastic softmax and hybridization," 2019.

[3] N. Rathi, G. Srinivasan, P. Panda, and K. Roy, "Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation," in *International Conference on Learning Representations*, 2020.

[4] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: Opportunities and challenges," p. 774, 2018.

[5] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "Stdp-based spiking deep convolutional neural networks for object recognition," 2016.

[6] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: Vgg and residual architectures," *Frontiers in neuroscience*, vol. 13, 2019.

[7] E. Hunsberger and C. Eliasmith, "Training spiking deep networks for neuromorphic hardware," 2016.

[8] C. Lee, S. S. Sarwar, P. Panda, G. Srinivasan, and K. Roy, "Enabling spike-based backpropagation for training deep neural network architectures," *Frontiers in Neuroscience*, vol. 14, Feb 2020.

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

[10] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[11] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, 05 2012.

[12] S. M. Bohte, J. N. Kok, and H. L. Poutré, "Spikeprop: backpropagation for networks of spiking neurons," in *ESANN*, 2000.

[13] "Superspike: Supervised learning in multilayer spiking neural networks," *Neural Comput.*, vol. 30, no. 6, p. 1514–1541, Jun. 2018.

[14] S. B. Shrestha and G. Orchard, "Slayer: Spike layer error reassignment in time," in *Advances in Neural Information Processing Systems 31*, 2018, pp. 1412–1421.

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

[16] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks," *CoRR*, vol. abs/1901.09948, 2019. [Online]. Available: http://arxiv.org/abs/1901.09948

[17] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Direct training for spiking neural networks: Faster, larger, better," 2018.