Near Lossless Transfer Learning for Spiking Neural Networks

Zhanglu Yan*, Jun Zhou*, Weng-Fai Wong
Department of Computer Science, National University of Singapore
{zhangluyan, zhoujun, wongwf}@comp.nus.edu.sg

Abstract

Spiking neural networks (SNNs) significantly reduce energy consumption by replacing weight multiplications with additions. This makes SNNs suitable for energy-constrained platforms. However, due to its discrete activation, training of SNNs remains a challenge. A popular approach is to first train an equivalent CNN using traditional backpropagation, and then transfer the weights to the intended SNN. Unfortunately, this often results in significant accuracy loss, especially in deeper networks. In this paper, we propose *CQ training* (Clamped and Quantized training), an SNN-compatible CNN training algorithm with clamp and quantization that achieves near-zero conversion accuracy loss. Essentially, CNN training in CQ training accounts for certain SNN characteristics. Using a 7 layer VGG-$*$ and a 21 layer VGG-19, running on the CIFAR-10 dataset, we achieved 94.16% and 93.44% accuracy in the respective equivalent SNNs. It outperforms other existing comparable works that we know of. We also demonstrate the low-precision weight compatibility for the VGG-19 structure. Without retraining, an accuracy of 93.43% and 92.82% using quantized 9-bit and 8-bit weights, respectively, was achieved. The framework was developed in PyTorch and is publicly available.[1]

Introduction

Spiking neural networks (SNNs) differs from traditional artificial neural networks such as the highly successful convolutional neural networks (CNNs) in that the activation data is transmitted among layers as sequences of binary spikes that are the result of some firing rules, instead of floating-point or fixed-point numbers. A key advantage of doing so is the elimination of expensive multiplications, making SNNs promising for resource-constrained devices like those used in edge AI. Unfortunately, their discrete activations render them non-differentiable, and hence unsuitable for the traditional back propagation training algorithms that relies on the chain rule applied to various partial derivatives. There are several ways around this problem. The first is to use training algorithms that are specific to SNNs (Wu et al. 2019b; Taherkhani et al. 2019). Many of these are based on *spike-timing-dependent plasticity* (Vigneron and Martinet 2020). However, to date, such approaches only seems to work for shallow networks, and simple datasets. Another approach is *tandem training* where forward passes are made on a CNN model and its twin SNN model (Wu et al. 2019a). On a backward training pass, errors and gradients from the SNN model are used to drive the learning in the CNN twin. The updated weights in the CNN is then transferred to the SNN for the next forward pass. Good accuracy for up to 14 layers on the DVS-CIFAR10 dataset was achieved (Wu et al. 2019a). However, this method requires both the CNN and SNN models during training, and hence cost more in terms of resources. The final approach, and the approach taken by this paper, is to train an equivalent CNN using the standard back propagation methods, then transfer the weights to the SNN. However, if done naïvely, there is a significant drop in accuracy. This brings us to the main contribution of this paper. Essentially, we propose a series of extensions to the basic back propagation framework that accounts for the idiosyncrasies of the SNN that the weights are intended for. The end result is what we called the *CQ training* ('clamped and quantized') algorithm. After CQ training, the weights of the CNN can be directly transferred over to the equivalent SNN with nearly no loss in accuracy. We have verified this for up to 21 layers of a CNN (VGG-19) on a complex series of dataset. The contributions of this paper are:

- We introduce a near lossless transfer learning method *CQ training* for spiking neural networks, especially deep SNNs. Activation clamping and quantization are adapted for each convolutional layer during the SNN-compatible CNN training, which significantly reduces the approximation error and deepens SNNs layers. Both the forward and backward propagation have been updated in this paper.
- A comprehensive discussion on SNN transfer learning problems is made in this paper. This includes new input spiking generation methods, low-precision weight exploration (weight quantization), weighted sum with batch normalization, and approximation reduction study.
- We introduce a fast and easy-to-program framework for CQ back propagation training, weight transfer, and SNN testing. This is evaluated on large SNNs (equivalent of VGG-19) running the MNIST, CIFAR-10, CIFAR-100

*Both authors contributed equally to this work.

[1]https://github.com/zhoujuncc1/shenjingcat

datasets. We show near lossless accuracy compared to their CNN counterparts, while using relatively small time window sizes compared to the state-of-the-art.

This paper is organized as follows. First, the background of SNN is introduced in comparison with CNN. Then, the details of CQ training framework is presented including clamp and quantization in Section . In Section , we present the near-lossless performance of our methods on deep SNN, with both full and low-precision weight. We also analyse the effectiveness of CQ training in reducing the approximate error with experiments. We then discussed insights from our study and reviewed and compared the existing works with our method in Sections -, followed by a conclusion.

Background

CNN. Convolution neural networks are sparse, equivariance neural networks (Liu et al. 2017) with continuous input and differentiable back propagation. In a layer l, the weighted sum of inputs $\boldsymbol{x}^{l-1}$ and weights $\boldsymbol{w}^l$ is computed in the CNN neurons. After adding a bias $\boldsymbol{b}^l$, an activation function f is then applied to produce the output activation $\boldsymbol{x}^l$. Mathematically, the activation of a neuron i is:

$$x_i^l = f(\sum_{j=0}^{n}(w_{i,j}^l \cdot x_j^{l-1}) + b_i^l) \tag{1}$$

SNN. While CNNs transmit activation data among layers ($\boldsymbol{x}^l$) as real numbers, SNN neurons receive series of binary inputs as *spike trains*. Formally, $\boldsymbol{s}_i^l$ is a spike train of neuron i at layer l where $\boldsymbol{s}_i^l(t) \in \{0, 1\}$ is a spike at time t. Unlike CNNs, the weighted sum is merely a summation since the weights are binary.

The Integrate-and-Fire (IF) Model. In this work, we adopt the widely used IF model to map between ANN and SNN activation. At each time step t, a neuron i computes the weighted sum of the received input spikes $\boldsymbol{s}_j^{l-1}(t)$ and the corresponding weights, and 'integrates' (sums) them as the membrane potential V_i. Whenever V_i exceeds a predefined threshold θ_i, the neuron fires a spike of '1', and decreases V_i by θ_i. Otherwise, it outputs '0'. Formally,

$$V_i^l(t) = V_i^l(t-1) + \sum_j (w_{i,j}^l \cdot \boldsymbol{s}_j^{l-1}(t) + b_i); \tag{2}$$

$$\boldsymbol{s}_i^l(t) = \begin{cases} 1, & V_i^l(t) \geq \theta_i \\ 0, & \text{otherwise} \end{cases}; \tag{3}$$

$$V_i^l(t) = \begin{cases} V_i^l(t) - \theta_i, & V_i^l(t) > \theta_i \\ V_i^l(t), & \text{otherwise} \end{cases}. \tag{4}$$

Spike Rate Encoding. In an SNN, trains (sequences) of spikes are used by neutrons to communicate with each other. They can be characterized as discrete events within a fixed time window. In the rate-based neural activity models, the firing rate is defined in a limit that involves infinite spikes. One way would be to divide the number of spikes by the duration of the time window. (Brette 2015)

For instance, $x_i = \frac{1}{3}$ is converted into a spike train of $[0, 0, 1, 0, 0, 1, ..., 0, 0, 1]$, a '1' spike out of every three spikes in the discrete spike train. $x_i = 1$ would then correspond to a spike train of all ones, and $x_i = 0$ is an all-zero spike train. These are then fed to the input layer.

Approach

In our CQ training framework, we constrain the CNN training in its input, weighted sum and activation so that its inference is nearly equivalent to the SNN. Therefore, the learned weights of the CNN can be transferred to its SNN counterpart with minimum approximation error. As shown in Figure 1, our the forward pass of the CNN in CQ training corresponds to the SNN in terms of input, weighted sum with batch normalization and activation.

1. **Input:** An SNN input spike train $\mathbf{s}_j$ of length T can only represent the spike rate $\frac{|\mathbf{s}_j|}{T} \in \mathbb{T}$ where $\mathbb{T}$ is the discrete set of values $\{0, \frac{1}{T}, \frac{2}{T}, \ldots, 1\}$, and $|\boldsymbol{s}_j|$ is the number of spikes in the spike train. Hence, we first *normalize* the CNN input data to $[0, 1]$, and then do a *quantization*.
2. **Weighted sum with batch normalization:** With the quantized input, weighted sum of CNN and SNN are nearly equivalent as $\sum w_{i,j}x_j + b_i \approx \sum w_{i,j}\frac{|\mathbf{s}_j|}{T} + b_i$. Batch normalization can also be transferred to the SNN via application on the trained weight.
3. **Activation:** In the same spirit of what is done to the input, CQ training also transforms the CNN activation to $\mathbb{T}$ by *clamping* and *quantizing* it so that it can be equivalently represented by the output spike train $\mathbf{s}_i^{l+1}$.

The pseudo code of this constrained training for one iteration is in Algorithm 1.

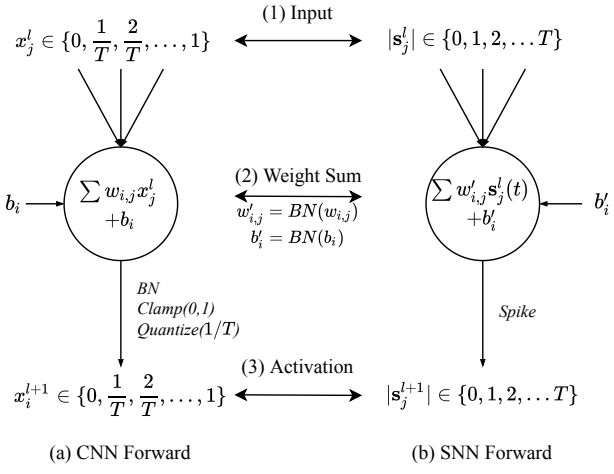

Figure 1: Correspondence of CNN and SNN forward pass.

Algorithm 1 One iteration of CQ training.

Input: Normalized and quantized input data $\boldsymbol{X}$; Label Y; Initialized layers $\{l_1, \ldots, l_N\}$;
Output: Trained layers $\{l_1, \ldots, l_N\}$.

Forward (Inference):
$\boldsymbol{x}^{l_0} = \boldsymbol{X}$
for $i = 1$ to N **do**
  $\boldsymbol{x}^{l_i} = l_i(\boldsymbol{x}^{l_{i-1}})$
  $\boldsymbol{x}^{l_i} = \textbf{BN}(\boldsymbol{x}^{l_i})$
  $\boldsymbol{x}^{l_i} = \textbf{Clamp}(\boldsymbol{x}^{l_i})$ // Equation 10
  $\boldsymbol{x}^{l_i} = \textbf{Quantize}(\boldsymbol{x}^{l_i})$ // Equation 5
end for

Backward (Training):
The back propagation with clamp and quantization are done according to Equations 13-14.

Input Normalization and Quantization

In rate encoding, a spike train of length T can only represent the spike rate in $\mathbb{T}$. Therefore, we first normalize the input data to $[0, 1]$ and then quantize them to $\mathbb{T}$. For example, each pixel of CIFAR-10 dataset is an integer in $[0, 255]$. This can be normalized to $[0, 1]$ simply by dividing the value by 255.

Quantization to a length T is performed by

$$Q_T(x) = \frac{\lfloor x \cdot T \rfloor}{T}. \tag{5}$$

We used the floor $\lfloor \cdot \rfloor$ function in quantization because in the IF model, a spike is only generated when it exceeds the threshold. Figure 2 illustrates the output of $Q_T(x)$ with $x \in [0, 1]$ and $T = 10$.

Gradient of $\mathbf{Q_T(x)}$. Quantization is applied both to input data and the activations of layers. Hence, we need to consider their gradients for back propagation. As shown in Figure 2, $Q_T(x)$ is not differentiable at the turning points and $Q'_T(x) = 0$ elsewhere. Therefore, for back propagation, we use the method of *straight-through estimator* (Bengio, Léonard, and Courville 2013), and approximate its gradient simply as

$$Q'_T(x) \approx 1. \tag{6}$$

Weighted Sum with Batch Normalization

Batch normalization is crucial in the training of deep CNN as it reduces the internal covariate shift (Ioffe and Szegedy 2015). Without batch normalization, training is difficult to converge, requiring lower learning rates and careful parameter initialization. In CNN training, batch normalization is performed to the activation of CNN layers $\boldsymbol{x}^l$ as follows:

$$\text{BN}_x(\boldsymbol{x}^l) = \gamma \frac{\boldsymbol{x}^l - \mu}{\sqrt{\sigma_B^2 + \epsilon}} + \beta \tag{7}$$

where μ and σ are the mean and variance of $\boldsymbol{x}$ over a *mini-batch* and γ and β are two trainable parameters.

In SNN, batch normalization cannot be applied to the binary spike trains, therefore, we apply it to the trained weight when transferring them to the SNN (Rueckauer et al. 2016). For each neuron i at a convolution layer l with batch normalization, we transfer the weights $\boldsymbol{W}^l$ and bias $\boldsymbol{b}^l$ as

$$\text{BN}_w(\boldsymbol{W}_i^l) = \frac{\gamma_i^l}{\sqrt{(\sigma_i^l)^2 + \epsilon}} \boldsymbol{W}_i^l \tag{8}$$

$$\text{BN}_w(\boldsymbol{b}_i^l) = \frac{\gamma_i^l}{\sqrt{(\sigma_i^l)^2 + \epsilon}} (\boldsymbol{b}_i^l - \mu_i^l) + \beta_i^l \tag{9}$$

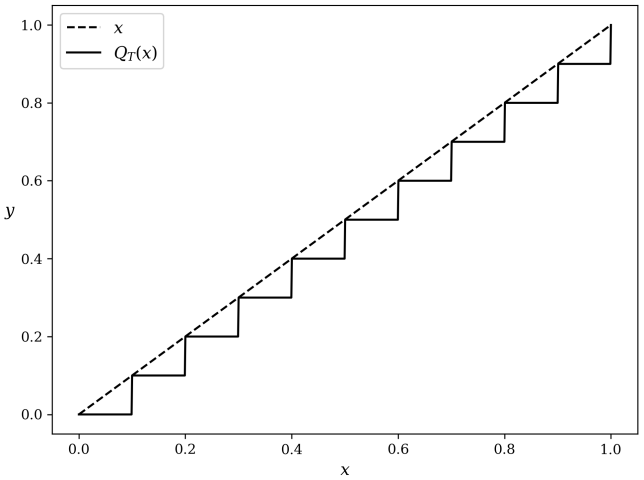

Figure 2: Figure of the quantization function $Q_T(x)$.

Activation Clamping and Quantization

ReLU is a popular activation function in CNNs, which makes its activation $\boldsymbol{x}^l \in \mathbb{R}_{\geq 0}$. However, as we discussed in the beginning of the section, the spike rate of a spike train is in a discrete set $\mathbb{T} \subset [0, 1]$. Therefore, we apply clamping and quantization as the activation function to restrict the activation to $\mathbb{T}$.

Clamping Clamping is first applied to the activation to restrict them to $[0, 1]$, and is defined as follows:

$$C(x) = \begin{cases} 0, & x < 0 \\ x, & 0 \leq x \leq 1 \\ 1, & x > 1. \end{cases} \tag{10}$$

Just like ReLU, the gradient (first derivative) of $C(x)$ is computed as

$$C'(x) = \begin{cases} 0, & x < 0 \\ 1, & 0 \leq x \leq 1 \\ 0, & x > 1. \end{cases} \tag{11}$$

Quantization As in the case of input data, we need to quantize the CNN activation to the set of spike rates $\mathbb{T}$ over length T with the function $Q_T(x)$ as defined in Equation 5.

Back Propagation

In the conventional CNN training, the operation at each layer l is given by

$$\boldsymbol{x}^l = (\boldsymbol{W}^l \boldsymbol{x}^{l-1})$$

and the gradients for back propagation are computed as

$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{w}^l} = (\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{x}^l})^T \cdot \boldsymbol{x}^{l-1};$$
$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{x}^{l-1}} = (\boldsymbol{W}^l)^T \cdot \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{x}^l}$$

where gradient of ReLU and batch normalization are omitted as they remain the same in our method.

In our CQ training, the forward operation is

$$\boldsymbol{x}^l = C \circ Q \circ (\boldsymbol{W}^l \boldsymbol{x}^{l-1}) \tag{12}$$

where '$\circ$' is the composition of functions. Therefore the gradients for back propagation is

$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{w}^l} = Q' \cdot C' \cdot \boldsymbol{x}^{l-1} \cdot (\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{x}^l})^T \tag{13}$$
$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{x}^{l-1}} = Q' \cdot C' \cdot (\boldsymbol{W}^l)^T \cdot \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{x}^l} \tag{14}$$

We have shown in Equations 6 and 11 that C' and Q' are either 0 or 1. Therefore, back propagation in CQ training is just the conventional back propagation in training CNNs.

Workflow of CQ Training

Based on the analysis above, the specific conversion methods and the forward propagation for our converted SNN are shown in Figure 3. The whole process has four steps:

1. A CNN is trained with ReLU activation, average pooling layer, and batch normalization.
2. The trained CNN is retrained with the ReLU function replaced by the C (Equation 10) and Q (Equation 5) functions (shown as 'CQ' in Figure 3). This retraining of the original training CNN network is key to our framework.
3. Batch normalization is then applied to the trained weights and biases using Equations 8 and 9 before transferring them to the corresponding SNN.
4. The SNN is deployed for inference.

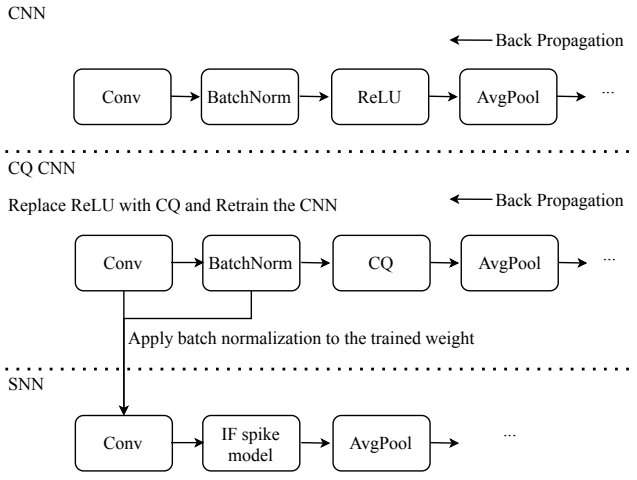

Figure 3: Summary of CQ training.

CQ Training and Approximation Error

According to Equation 2, in a SNN with the integrate-and-fire model, the number of spikes rate of a neuron $|\boldsymbol{s}_i|$ with respect to its input is

$$|\boldsymbol{s}_i^l| = \lfloor \frac{\sum_j w_{i,j}^l |\boldsymbol{s}_j^{l-1}| + b_i T}{\theta_i} \rfloor = \frac{\sum_j w_{i,j}^l |\boldsymbol{s}_j^{l-1}| + b_i T}{\theta_i} - r_i^l \tag{15}$$

where the residual

$$r_i^l = \frac{\sum_j w_{i,j}^l |\boldsymbol{s}_j^{l-1}| + b_i T}{\theta_i} - \lfloor \frac{\sum_j w_{i,j}^l |\boldsymbol{s}_j^{l-1}| + b_i T}{\theta_i} \rfloor$$

For convenience, we remove the bias b_i in this analysis without loss of generality. Expanding the recursion, the number of spikes in the last layer L is then

$$|\boldsymbol{s}_{i_L}^L| = \sum_{i_{L-1}} w_{i_L,i_{L-1}}^L \cdot \sum_{i_{L-2}} w_{i_{L-1},i_{L-2}}^{L-1} \cdots \sum_{i_0} w_{i_1,i_0}^1 |\boldsymbol{s}_{i_0}^0| - \varepsilon_{i_L}'^L \tag{16}$$

where

$$\begin{aligned} \varepsilon_{i_L}'^L = r_{i_L}^L &+ \sum_{i_{L-1}} w_{i_L,i_{L-1}}^L \cdot (r_{i_{L-1}}^{L-1} \\ &+ \sum_{i_{L-2}} w_{i_{L-1},i_{L-2}}^{L-1} \cdot (r_{i_{L-2}}^{L-2} \\ &+ \cdots + \sum_{i_0} w_{i_2,i_1}^2 r_{i_1}^1)) \cdots)) \end{aligned} \tag{17}$$

and $\boldsymbol{s}_{i_0}^0$ is the input spike train.

In conventional CNN, expanding the recursive Equation 1, the output of last layer is

$$x_{i_L}^L = \sum_{i_{L-1}} w_{i_L,i_{L-1}}^L \cdot \sum_{i_{L-2}} w_{i_{L-1},i_{L-2}}^{L-1} \cdots \sum_{i_0} w_{i_1,i_0}^1 x_{i_0}^0 \tag{18}$$

From above Equations 16-18, assuming the input spike train without error is $|\boldsymbol{s}_{i_0}^0| = x_{i_0}^0 T$, the approximation error is

$$\varepsilon_{i_L}^L = \frac{|\boldsymbol{s}_{i_L}^L|}{T} - x_{i_L}^L = \frac{\varepsilon_{i_L}'^L}{T} \tag{19}$$

The error will accumulate over the layers and cause loss in accuracy.

In CQ training, by quantization, a similar residual term $r' = x - Q(x)$ is introduced to the forward pass so that the activation becomes

$$x_i^l = Q(\sum_{j=0}^n w_{i,j}^l \cdot x_j^{l-1} + b_i^l) = \sum_{j=0}^n w_{i,j}^l \cdot x_j^{l-1} + b_i^l - r_i'^l \tag{20}$$

Omitting the expansion of equations, and taking $\theta_i^l = 1$, it is easy to derive that $r_i'^l \approx r_i^l$. Hence, in theory, the approximation error $\varepsilon_{i_L}^L \approx 0$.

In conclusion, the CQ training minimize the approximation error of SNN to near zero. We conducted series of experiments in Section to shows the it successfully mitigated the accuracy loss due to accumulated approximation error.

Experiment

Experiment Setup

We have implemented CQ training in CUDA-accelerated PyTorch version 1.6.0. The experiments were performed on a Intel Xeon E5-2680 server with 256GB DRAM and a Tesla P100 GPU, running 64-bit Linux 4.15.0.

Training. We experimented with standard deep CNN structures like VGG-11, 13, 16 and 19, and trained them with our CQ training framework. As rate-encoded SNN does not support max-pooling, we replaced these layers with average pooling. In addition, we also added a new model based on VGG, labelled VGG-$*$, that has fewer parameters, but shows better overall baseline accuracy for CIFAR-10 and CIFAR-100. We trained the networks using the Adam optimizer with an adaptive learning rate for 100$\sim$150 epochs until they converged. The network structure we used in this work is summarized in Table 1.

SNN Inference. In SNN inference, we use the 'integrate-and-fire' neuron model, and set all thresholds to 1. We adjusted the length of spike trains T to achieve the best accuracy. In the experiments, T is between 100 to 1000. The input data was normalized to $[0, 1]$ before being used to produce input spike trains as depicted in Algorithm 2.

Algorithm 2 Input encoding algorithm.

Input: Input data $x \in [0, 1]$.
Output: Spike train $\boldsymbol{s}$ of length T.

```
Membrane voltage V = 0
for i = 0 to T − 1 do
    V = V + x
    if V >= 1 then
        s_i = 1
        V = V − 1
    else
        s_i = 0
    end if
end for
```

LeNet*	32, 32, C, 64, 64, C, 128
VGG-11	64, A, 128, A, 256, 256, A, 512, 512, A, 512, 512, A
VGG-13	64, 64, A, 128, 128, A, 256, 256, A, 512, 512, A, 512, 512, A
VGG-16	64, 64, A, 128, 128, A, 256, 256, 256, A, 512, 512, 512, A, 512, 512, 512, A
VGG-19	64, 64, C, 128, 128, C, 256, 256, 256, 256, C, 512, 512, 512, 512, C, 512, 512, 512, 512, C
VGG-$*$	128, 128, A, 256, 256, A, 512, 512, A, 1024, A

Table 1: Summary of network structures.

The last dense classifier layer in all the networks are omitted in the table. 'A' stands for an average pooling layer. 'C' stands for a stride-2 convolutional layer used for down-sampling.

Accuracy Benchmark

We tested our methods on different networks structures and datasets, and the results are summarized in Table 2. The length of the spike trains T were set between 100 to 1000 for LeNet*, VGG-11/*, VGG-13 and VGG-16/19 respectively.

For comparison, we define the *accuracy loss in conversion* (using method A), Δ_A, to be

$$\Delta_A = \frac{\text{Acc. of baseline model} - \text{Acc. using } A}{\text{Acc. of baseline model}} \tag{21}$$

Note that Δ_A is signed. In the rare instance where the converted model is more accurate than the baseline model, then Δ_A (the 'loss') will be negative.

MNIST. MNIST consists of 60,000 28×28 grayscale images of handwritten digits from 0 to 9. 50,000 for training and 10,000 for testing. LeNet is a model trained to do the classification for the MNIST dataset. With data augmentation, the accuracy we achieve for SNN with a LeNet-type network structure is 99.39% and the conversion accuracy loss (Δ_{SNN}) is only 0.05%.

CIFAR-10. CIFAR-10 dataset consists of 60,000 32×32 RGB images of 10 classes, 6000 per class. They are split to 50,000 training images and 10,000 test images. We trained VGG-11, 13, 16 and 19 for this dataset with data augmentation and the conversion accuracy loss compared to CNN is very small and the CQ trained VGG-16 SNN achieved an accuracy of 92.48%, 0.08% accuracy drop (Δ_{SNN}) compared with the baseline model, with a time step of 600.

CIFAR-100. The CIFAR-100 has the same structure as CIFAR-10 but with labels for 100 classes (Krizhevsky, Nair, and Hinton 2009). We built a 7-layers VGG-like CNN (VGG-$*$ in Table 1) for this dataset, and achieved accuracy of 71.52% with a time step of 200. The conversion accuracy (conversion accuracy) loss is 0.4%.

Low-Precision Weight

The elimination of multiplications makes SNN suitable for energy-constraint platforms. On such platforms, low bit-width fixed point weights are more often used than 32-bit floating point numbers. Therefore, we tested the accuracy of our method with low precision weights.

Dataset	Network structure	Acc. of CNN baseline	Acc. of SNN	Δ_{SNN}
MNIST	LeNet*	99.44%	99.39%	0.05%
CIFAR-10	VGG-11	82.13%	82.09%	0.05%
	VGG-13	86.10%	86.08%	0.02%
	VGG-16	92.55%	92.48%	0.08%
	VGG-19	93.50%	93.44%	0.06%
	VGG-$*$	94.20%	94.16%	0.04%
CIFAR-100	VGG-$*$	71.84%	71.52%	0.4%

Table 2: Accuracy from CNNs to SNNs.

Bitwidth	Baseline (32 bits)	11	10	9	8
Acc(%)	93.44	93.44	93.42	93.43	92.82
Loss(%)	-	0	0.02	0.01	0.62

Table 3: Accuracy of CQ trained VGG-19 SNN on CIFAR-10 using quantized weights.

Weight Quantization We used the following n-bit fixed point number $w[n-1:0]$ to represent weights:

- $w[n-1]$ is the sign bit;
- $w[n-2:0]$ are the fraction bits;
- The integral part left of the binary point is always 0.

Therefore, with n bits, we can represent the weights

$$w \in \{\pm\frac{k}{2^{n-1}} | k \in \mathbb{N}_{<2^{n-1}}\}$$

where $\mathbb{N}_{<2^{n-1}}$ is the set of natural numbers less than 2^{n-1}.

We quantized the trained weights by first normalizing to the representable range $[\frac{1}{2^{n-1}}-1, 1-\frac{1}{2^{n-1}}]$, and then quantization to the fixed-point numbers with the following:

$$\boldsymbol{w_1} = (1-\frac{1}{2^{n-1}}) \cdot \frac{\boldsymbol{w}}{\max(|\boldsymbol{w}|)};$$

$$\boldsymbol{w_Q} = Q_{(2^{n-1}-1)}(\boldsymbol{w_1}).$$

Note that as we normalize the weight in each layer, the threshold must also be normalized by the same factor.

Accuracy of CIFAR-10 We quantize the trained weight of VGG-19 for CIFAR-10 in the range of 8 to 11 bits, and tested the accuracy loss. Table 3 shows the accuracy of SNN with weights of different bitwidths and 32-bit floating point as baseline. Quantizating to 9 and 8-bit weights does not result in much accuracy loss. It is a mere 0.01% and 0.62%, respectively. However, there is significant accuracy loss if the weights are 7-bit or lower.

Going Deep

As we discussed in Equation 19, the approximation error accumulates through the CNN layers, causing significant accuracy loss when converting deep CNN to SNN. Hence many have dismissed SNN as not suitable for deep neural network, limiting their usefulness for real applications. We performed experiments to explore the accuracy loss from the approximation error in deep SNN, and the effectiveness of CQ training in reducing this error. We trained deep CNNs with different numbers of convolution layers for the simple MNIST dataset and tested their accuracy loss when converted to SNN. Table 4 shows the accuracy achieved with CQ training with that of naïvely transferring the CNN weights over to the SNN. Without CQ training, the conversion loss becomes significant and increases rapidly from 11 convolution layers onward. In contrast, with CQ training, the accuracy loss stays very low even up to 21 convolution layers. This shows the effectiveness of CQ training in reducing the approximation error when converting a CNN to a SNN.

# Conv Layer	Δ_{CQ}	$\Delta_{\text{naïve}}$
10	0.05%	1.77%
11	0.09%	26.8%
12	0.05%	54.7%
13	-0.13%	88.7%
14	0.12%	88.7%
15	0.03%	89.76%
16	0.16%	88.67%
17	0.14%	88.57%
18	0.1%	89.76%
19	0.21%	86.82%
20	0.23%	85.07%
21	0.01%	88.65%

Table 4: Conversion accuracy loss with/without CQ training.

Discussion

Input Spike Generation Methods. In rate encoding, there are different methods to generate the spike train whose spiking rate represents the input data. Some works sample the spike train from Poisson or Bernoulli distribution (Diehl et al. 2015; Rueckauer et al. 2017). However, the randomness may cause sampling error leading to accuracy loss. Therefore, in this work, we used a spike layer to produce a deterministic spike train from the original floating-point inputs. As shown in Algorithm 2, each normalized input $x \in [0, 1]$ is repeatedly fed into a spike layer of threshold 1.0 for T time steps. The spike layer then output a deterministic spike train of evenly distributed xT spikes. Compared to a spike train generated using the random distribution method, our method gives 8.33% higher accuracy.

Limitation: Average Pooling. Max pooling has been shown to be effective in state-of-the-art CNN training as it naturally pick the prominent features among the inputs(Chollet 2017). However, it is difficult to implement max pooling on rate encoded SNN because the max spike rate can only be determined after receiving the entire spike train, and the integrate-and-fire model is inherently sequential. This limits how much rate encoded SNN can do compared to the best CNN performance, even though CQ training provides near lossless conversion. Chen et al. (2018) implemented max pooling by stalling the whole spike train at the pooling layer, then choosing the one with highest spike rate. However, this severely increases the latency from $O(T)$ to $O(nT)$ where n is the number of pooling layers, compared to a pipelined sequential SNN. A convolutional layer with a stride of 2 can be used to simulate pooling.

ResNet Block. We have also applied CQ training on the basic and bottleneck block of ResNet introduced in (He et al. 2016).With the CQ training, unlike (Hu et al. 2018). no compensation for normalisation is needed. The accuracy loss Δ_{SNN} for basic block and bottleneck block are 0.05% and 0.1%, respectively.

Dataset	Model	Method	Accuracy	Architecture	Δ_{SNN}	Time Steps
CIFAR-10	(Rathi et al. 2020)	Hybrid Training	92.02 %	VGG-16	N/A	200
	(Wu et al. 2019b)	Spike-based BP	90.53%	5 Conv	N/A	12
	(Rueckauer et al. 2017)	Pretrained SNN	90.85%	4Conv, 2Linear	1.15%	N/A
	(Hunsberger and Eliasmith 2015)	Pretrained SNN	82.95%	2Conv, 2Linear	2.83%	6000
	(Cao, Chen, and Khosla 2015)	Pretrained SNN	77.43%	3Conv, 2Linear	2.14%	400
	(Sengupta et al. 2019)	Pretrained SNN	91.55%	VGG-16	0.26%	2500
	This Work	CQ trained SNN	**92.48%**	VGG-16	0.08%	600
	This Work	CQ trained SNN	**93.44%**	VGG-19	0.06%	1000
	This Work	CQ trained SNN	**94.16%**	VGG-$*$	0.04%	600
CIFAR-100	(Hu et al. 2018)	Pretrained SNN	68.56%	ResNet-44	2.31%	350
	This Work	CQ trained SNN	**71.52%**	VGG-$*$	0.4%	200
	This Work	CQ trained SNN	**71.84%**	VGG-$*$	0%	300

Table 5: Comparison with state-of-the-art SNNs on the CIFAR-10/100 dataset.

Related Work

In the state of the art, there are mainly three directions of SNN learning – unsupervised learning, spike-based back propagation and CNN-to-SNN conversion.

By its neuromorphic nature, SNN training can use bio-inspired unsupervised learning rules such as the Hebbian learning rule (Hebb 2005) and spiking timing dependent plasticity (STDP) (Bi and Poo 1998). These methods have been shown to work for shallow networks and simple datasets, but not for deep neural networks.

The main challenge of supervised learning in SNN is the non-differentiability of spikes. Most spike-based back propagation methods attempted to solve this issue by either approximating the spiking function as differentiable functions (Huh and Sejnowski 2018; Shrestha and Orchard 2018) or defining a surrogate gradient to it (Neftci, Mostafa, and Zenke 2019; Bellec et al. 2018). Again, most of these works are limited to shallow networks and simple datasets. For example, SLAYER (Shrestha and Orchard 2018) only trained up to 3-layer CNNs on MNIST and DVS-based datasets. DECOLLE (Kaiser, Mostafa, and Neftci 2020) trained up to 4-layer CNNs on similar datasets. Computation cost is another challenge for spike-based back propagation. Though SNN inference is done by addition, for the sake of back-propagation in these algorithms, the forward pass in training still has to use matrix multiplications.

CNN-to-SNN conversion does not suffer from non-differentiability and high computation cost, and is widely used in SNN learning. Sengupta et al. (2019) and Lee et al. (2020) train ANNs without biases. They claimed that the bias term in SNN has an indirect effect on the threshold voltage that makes finding and tuning the threshold difficult. During CNN training, dropout was used as a regularizer instead of batch normalization in the absence of bias (Srivastava et al. 2014). However, inspired by Rueckauer et al. (2016), we have implemented bias as a constant charge to the membrane potential in each time step, and batch normalization as a transformation to the trained weight.

By the addition of clamping and quantization, the CQ training framework trains a SNN-compatible CNN structure using the traditional CNN backpropagation algorithm, while achieving both high accuracy, and near zero approximation error. We compare our results with the state-of-the-art works (Rueckauer et al. 2017; Cao, Chen, and Khosla 2015; Sengupta et al. 2019; Zhang et al. 2019) in Table 5 on both CIFAR-10 and CIFAR-100 datasets. We achieved not only higher accuracy than the existing works, but also significantly lower the relative accuracy loss compared to CNN counterparts: 0.06% for SNN (VGG-19) and 0.08% for SNN (VGG-16), comparing with 1.15% in Rueckauer et al. (2017), 2.14% in Cao, Chen, and Khosla (2015), and 0.83% in Zhang et al. (2019). Besides higher accuracy, our method also uses shorter spike trains, which significantly improves the throughput of the SNN. For example, our VGG-16/19 use spike trains of 600/1000 times steps, smaller than the 6,000 and 2,500 used in Hunsberger and Eliasmith (2015) and Sengupta et al. (2019), respectively, even though their networks are much shallower. Using a VGG-like network with 7 convolution layers, we achieved 71.84% accuracy, higher than the 68.56% achieved by Hu et al. (2018) which used a much deeper ResNet-44 SNN.

Conclusion

In this paper, we introduce CQ training, a CNN learning strategy that will allow for the transfer of trained weights from a CNN over to its equivalent SNN with nearly no loss . Through clamping and quantization, the discrete nature of a SNN spike train is account for in the training process. Our experiments show that CQ training yields deeper SNNs with nearly no accuracy loss compared to their CNN counterparts, outperforming the state of the art. Although the number of layers in such an SNN is still very modest compared to the largest CNNs, we believe that it has reached a level where it is now feasible to implement a relatively sophisticated CNN as an SNN for practical inference tasks on edge AI devices, and reap the area and energy benefits of the multiplier-less SNNs.

Acknowledgments

This research is supported in part by programmatic grant no. A1687b0033 from the Singapore government's Research, Innovation and Enterprise 2020 plan (Advanced Manufacturing and Engineering domain).

References

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. In *Advances in Neural Information Processing Systems*, 787–797.

Bengio, Y.; Léonard, N.; and Courville, A. 2013. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation.

Bi, G.-q.; and Poo, M.-m. 1998. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of neuroscience* 18(24): 10464–10472.

Brette, R. 2015. Philosophy of the spike: rate-based vs. spike-based theories of the brain. *Frontiers in systems neuroscience* 9: 151.

Cao, Y.; Chen, Y.; and Khosla, D. 2015. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision* 113(1): 54–66.

Chen, R.; Ma, H.; Guo, P.; Xie, S.; Li, P.; and Wang, D. 2018. Low Latency Spiking ConvNets with Restricted Output Training and False Spike Inhibition. In *2018 International Joint Conference on Neural Networks (IJCNN)*, 1–8.

Chollet, F. 2017. Deep Learning with Python.

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)*, 1–8. IEEE.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hebb, D. O. 2005. *The organization of behavior: A neuropsychological theory*. Psychology Press.

Hu, Y.; Tang, H.; Wang, Y.; and Pan, G. 2018. Spiking deep residual network. *arXiv preprint arXiv:1805.01352* .

Huh, D.; and Sejnowski, T. J. 2018. Gradient descent for spiking neural networks. In *Advances in Neural Information Processing Systems*, 1433–1443.

Hunsberger, E.; and Eliasmith, C. 2015. Spiking deep networks with LIF neurons. *arXiv preprint arXiv:1510.08829* .

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

Kaiser, J.; Mostafa, H.; and Neftci, E. 2020. Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE). *Frontiers in Neuroscience* 14: 424.

Krizhevsky, A.; Nair, V.; and Hinton, G. 2009. Cifar-10 and cifar-100 datasets. *URl: https://www. cs. toronto. edu/kriz/cifar. html* 6: 1.

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

Liu, W.; Wang, Z.; Liu, X.; Zeng, N.; Liu, Y.; and Alsaadi, F. E. 2017. A survey of deep neural network architectures and their applications. *Neurocomputing* 234: 11–26.

Neftci, E. O.; Mostafa, H.; and Zenke, F. 2019. Surrogate gradient learning in spiking neural networks. *IEEE Signal Processing Magazine* 36: 61–63.

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* .

Rueckauer, B.; Lungu, I.-A.; Hu, Y.; and Pfeiffer, M. 2016. Theory and tools for the conversion of analog to spiking convolutional neural networks. *arXiv preprint arXiv:1612.04052* .

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. *Frontiers in neuroscience* 11: 682.

Sengupta, A.; Ye, Y.; Wang, R.; Liu, C.; and Roy, K. 2019. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in neuroscience* 13: 95.

Shrestha, S. B.; and Orchard, G. 2018. Slayer: Spike layer error reassignment in time. In *Advances in Neural Information Processing Systems*, 1412–1421.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* 15(1): 1929–1958.

Taherkhani, A.; Belatreche, A.; Li, Y.; Cosma, G.; Maguire, L.; and Mcginnity, T. 2019. A review of learning in biologically plausible spiking neural networks. *Neural Networks* 122. doi:10.1016/j.neunet.2019.09.036.

Vigneron, A.; and Martinet, J. 2020. A critical survey of STDP in Spiking Neural Networks for Pattern Recognition (Preprint) doi:10.13140/RG.2.2.24086.09287.

Wu, J.; Chua, Y.; Zhang, M.; Li, G.; Li, H.; and Tan, K. C. 2019a. A Tandem Learning Rule for Effective Training and Rapid Inference of Deep Spiking Neural Networks.

Wu, Y.; Deng, L.; Li, G.; Zhu, J.; Xie, Y.; and Shi, L. 2019b. Direct training for spiking neural networks: Faster, larger, better. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 1311–1318.

Zhang, L.; Zhou, S.; Zhi, T.; Du, Z.; and Chen, Y. 2019. Tdsnn: From deep neural networks to deep spike neural networks with temporal-coding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 1319–1326.