Data Driven Threshold and Potential Initialization for Spiking Neural Networks

Velibor Bojković^{1,*}, **Srinivas Anumasa**^{1,*}, **Giulia De Masi**^{2,3}, **Bin Gu**^{4,1,†} **Huan Xiong**^{5,1,†}
^{1} Mohamed bin Zayed University of Artificial Intelligence, UAE
^{2} ARRC, Technology Innovation Institute, UAE
^{3} BioRobotics Institute, Sant'Anna School of Advanced Studies Pisa, Italy
^{4} School of Artificial Intelligence, Jilin University, China
^{5} Harbin Institute of Technology, China
†Correspondence to: huan.xiong.math@gmail.com, jsgubin@gmail.com. *Equal Contributions

Abstract

Spiking neural networks (SNNs) present an increasingly popular alternative to artificial neural networks (ANNs), due to their energy and time efficiency when deployed on neuromorphic hardware. However, due to their discrete and highly non-differentiable nature, training SNNs is a challenging task and remains an active area of research. Some of the most prominent ways to train SNNs are based on ANN-to-SNN conversion where an SNN model is initialized with parameters from the corresponding, pre-trained ANN model. SNN models trained through ANN-to-SNN conversion or hybrid training show state of the art performance among SNNs on many machine learning tasks, comparable to those of ANNs. However, the top performing models need high latency or tailored ANNs to perform well, and in general are not using the full information available from ANNs. In this work, we propose novel method to initialize SNN's thresholds and initial membrane potential after ANN-to-SNN conversion, using distributions of ANN's activation values. We provide a theoretical framework for feature distribution-based conversion error, providing theoretical results on optimal membrane initialization and thresholds which minimize this error, as well as a practical algorithm for finding these optimal values. We test our method, both as a stand-alone ANN-to-SNN conversion and in combination with other methods, and show state of the art results on high-dimensional datasets such as CIFAR10, CIFAR100 and ImageNet and various architectures. Our code is available at https://github.com/srinuvaasu/data_driven_init

Proceedings of the 27th International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238. Copyright 2024 by the author(s).

1 INTRODUCTION

Spiking neural networks (SNNs) are dubbed third generation neural networks [Maass, 1997]. Inspired and designed by how biological neurons process and share information [McCulloch and Pitts, 1943, Hodgkin and Huxley, 1952, Izhikevich, 2003], they have seen increasing popularity and success in the past decade, due to their promising energy efficiency. Unlike in artificial neural networks (ANNs) [Braspenning et al., 1995], neurons in an SNN model communicate through a stream of spikes, emulating the communication between neurons in biological brain which is through electric pulses. An SNN neuron accumulates the incoming spikes in the form of membrane potential, and emits a spike only when the potential reaches a threshold, making the processing of information event-driven and binary. On the other side, processing information in ANNs involves floating point operations, which on large scale results in an energy inefficient deep learning models [Roy et al., 2019]. The advantage of SNN models is further emphasized through the recent advancements in production of neuromorphic chips [Pei et al., 2019, DeBole et al., 2019], which are dedicated to supporting and embedding SNN models into hardware in an energy efficient way. The invention of neuromorphic hardware gave a new boost in popularity of SNNs, challenging the traditional neural networks in different domains, like object detection [Kim et al., 2020a, Cheng et al., 2020], object tracking [Yang et al., 2019], video reconstruction [Zhu et al., 2022], generative models [Kamata et al., 2022], just to name a few.

The ever-growing popularity of SNNs would not be possible without efficient ways to train them. There are two standard approaches to training an SNN model: direct training (supervised and unsupervised) and ANN-to-SNN conversion based approaches. In supervised direct training [Wu et al., 2018, Neftci et al., 2019, Zenke and Vogels, 2021, Mukhoty et al., 2023, Anumasa et al., 2024], an SNN model with randomly initialized weights is trained via backpropagation where the non-differentiability of spikes is overcome by using surrogate gradients or by taking one-sided derivatives. The unsupervised direct training [Diehl and Cook, 2015] is biologically inspired and uses local learning rules and timing of spikes to update the weights. In particular, unsupervised methods are hardware friendly, but SNN models thus trained still lag in performance compared to supervised or ANN-to-SNN conversion based methods. In the ANN-to-SNN conversion approach [Cao et al., 2015, Diehl et al., 2015, Deng and Gu, 2021], an already trained ANN model is used to initialize the corresponding SNN model through a series of operations. In general, these operations include copying the weights (sometimes normalized in a suitable way), replacing the ANN neurons and activation functions with SNN neurons and their dynamics, to replicate the output and performance of ANNs. Moreover, such obtained SNN models may be further tuned to increase their performance.

Although both these approaches have shown impressive results on computer vision tasks in recent years, they come with their own drawbacks. For example, supervised direct training requires a lot of computational and energy resources due to the time-recursive nature of SNNs and the fact that training still has to be performed on standard GPUs which are not adapted to the spiking nature of SNNs. On the other hand, ANN-to-SNN conversion based methods, in general, suffer from long latency needed to approximate ANN performance [Li et al., 2021a, Rathi et al., 2019, Han et al., 2020], or they need tailored trained ANN models with non-standard activation functions to optimize the conversion [Bu et al., 2021, Deng and Gu, 2021].

Drawing our motivation from the fact that present ANN-to-SNN methods do not fully exploit ANN models to initialize SNN models, we propose a new way to initialize SNN thresholds and initial membrane potential in SNNs, further exploiting the corresponding ANN model. Namely, thhe SNN model weights are initialized with the weights of the ANN model while the spiking neuron membrane thresholds for each layer are set by inferring the knowledge from the statistics of the features in each layer of the ANN model. To the best of our knowledge, this is the first time that the full statistics of ANN layer's outputs have been exploited in the SNN setting.

Our contributions in this work can be summarized as follows:

- We formulate and study activation-distribution based, layer-wise conversion error. The conversion error is expressed in dependence of the probability measure (corresponding to the distribution of ANN activation values), and for latency $T = 1$ we characterize optimal potential and membrane initialization in terms of this probability measure, while in higher latency we show an intrinsic relation between these two optimal values.
- Based on our theoretical findings, we propose practical algorithms to find optimal values of thresholds and potential initialization in each layer using the knowledge inferred from the ANN activation values.
- We show through experiments that our models show state-of-the-art performance on several popular benchmarks such as CIFAR10, CIFAR100, ImageNet and deep architectures such as VGG16, ResNet18, ResNet19 and CIFARNet.

2 RELATED WORK

Training SNNs through ANN-to-SNN conversion is an alternative to direct training of SNNs achieving top performance in supervised learning problems. First works in this direction, [Diehl et al., 2015] and [Cao et al., 2015] outline the general procedure of ANN-to-SNN conversion, while [Diehl et al., 2015] introduces weight and threshold balancing, i.e. weight normalization and threshold initialization based on the maximal activation values of layers of ANN (further explored in [Sengupta et al., 2019]). Other improvements of the method came with [Rueckauer et al., 2017] where a reset-by-subtraction (also known as soft-reset as in [Han et al., 2020]) in SNN neurons is used to eliminate information loss during membrane potential resetting. More generally, [Rueckauer et al., 2016] provides a theoretical justification for the ANN-to-SNN procedure, explicitly describing the equations underlying conversion (summarized above), but also suggesting the 99.9th percentile method to initialize SNN neuron thresholds to further reduce the conversion error. The weight and threshold normalization is also used in [Kim et al., 2020b] (where it is applied channel-wise) and in [Li et al., 2022], where authors introduced dynamics of bistable SNN neurons adapted for ANN-to-SNN conversion using temporal encoding. In [Li et al., 2022] the authors also study the part of the conversion error coming from the properties of spike trains (thus, not directly related to the conversion process). We point out that all the work mentioned suffers from long latency SNNs need to achieve near ANN accuracy.

In an attempt to reduce the SNN latency, other methods have been proposed that do not necessarily follow the general procedure. For example, [Rathi et al., 2019] uses a hybrid method (conversion for initialization of weights and thresholds as described before, followed by direct training). In a similar way, [Li et al., 2021a] proposes weights and threshold calibration after the conversion, as well as a grid search for the threshold initialization for SNNs during the conversion process. In [Deng and Gu, 2021] authors study in detail the conversion error. Further, a shift of the bias while converting ANNs to SNNs is proposed as well as a truncated ReLU activation while training ANNs, to reduce the conversion error. In [Bu et al., 2021] a new activation function for ANNs has been proposed which reduces the conversion error, while threshold initialization is learned during the training of ANNs. In a continuation of this work [Hao et al., 2023], authors further improve on the accuracy of the method through sample-based, layer-wise membrane potential initialization, but increasing latency proportional to the number of activation layers in ANNs. Although, [Deng and Gu, 2021] and [Bu et al., 2021] achieve high SNN accuracy through ANN-to-SNN conversion, these methods suffer from the need to train the corresponding ANN models using their respective proposed activation functions.

When it comes to threshold initialization, all the above methods either use the maximal activation values layer-wise to derive the corresponding membrane threshold or the threshold is learned during the training.

3 BACKGROUND

We use integrate-and-fire (IF) neurons in SNNs, which are widely used in the setting of ANN-to-SNN conversion. At every time step, each neuron receives spikes from the previous layers weighted with the network weights, and accumulates membrane potential. If the potential reaches the membrane threshold, the neuron fires a spike (weighted by the threshold value) and resets by subtracting the membrane threshold [Rueckauer et al., 2016]. The process is repeated at discrete time steps throughout the simulation time.

To describe the neuronal dynamics and ANN-to-SNN conversion process precisely, we introduce some notations. We will use $\mathbf{v}[t]$ to denote the membrane potential (voltage) at time step t, $\mathbf{s}[t]$ will denote a binary variable, 1 or 0, according to whether there is a spike at time step t or not, $\mathbf{W}$ will denote the network weight while V_{th} stands for the membrane threshold. The superscripts will stand for the layer index while the subscripts will denote the neuron index in a given layer. Finally, T will stand for the simulation time. With this notation, at each time step t the membrane voltage is updated accordingly as

$$\begin{aligned} \mathbf{v}[t] &= \mathbf{v}[t-1] + \mathbf{W}V_{\text{th}}^{(-1)}\mathbf{s}^{(-1)}[t] - V_{\text{th}}\mathbf{s}[t-1], \\ \mathbf{s}[t] &= H(\mathbf{v}[t] - V_{\text{th}}), \end{aligned} \quad (1)$$

where H is the Heaviside function.

3.1 General ANN-to-SNN conversion procedure

We follow the general conversion procedure outlined in [Diehl et al., 2015] and [Cao et al., 2015], by copying ANN weights to SNN. Furthermore, we use constant rate encoding of inputs, where an input value is converted into a length T spike train by repeating its value T times. This generalizes to deeper layers as a real value is encoded into a spike train via firing rate, where the number of spikes during the simulation time averaged over T, approximates the value.

In more detail, let us consider equation (1). Summing up all the equations for $t = 1, \ldots, T$, initializing the voltage with $\mathbf{v}^{(l)}[0]$ (which has the same value denoted by $v^{(l)}[0]$ for each neuron in the layer l) and averaging over T, we obtain

$$V_{\text{th}}^{(l)} \frac{\sum_{t=1}^{T} \mathbf{s}^{(l)}[t]}{T} = \mathbf{W}^{(l)} V_{\text{th}}^{(l-1)} \frac{\sum_{t=1}^{T} \mathbf{s}^{(l-1)}[t]}{T} \quad (2)$$

$$+ \frac{\mathbf{v}^{(l)}[T] - \mathbf{v}^{(l)}[0]}{T}. \quad (3)$$

On the ANN side of things, a passage between the layers takes the form

$$a^{(l)} = \mathcal{A}^{(l)}(\mathbf{W}^{(l)} a^{(l-1)}), \quad (4)$$

where $\mathcal{A}^{(l)}$ is the activation function. In ANN-to-SNN conversion process, the most commonly used choice for $\mathcal{A}$ is the ReLU function, due to its simplicity and non-negative output which relates well to the properties of IF neurons. We adopt this choice so that, comparing equations (2) and (4), it becomes clear that the activation value is approximated with the averaged output of the corresponding SNN neuron (firing rate) [Rueckauer et al., 2016]

$$a_i^{(l)} \approx V_{\text{th}}^{(l)} \frac{\sum_{t=1}^{T} \mathbf{s}_i^{(l)}[t]}{T}. \quad (5)$$

3.2 Conversion error

The conversion error between the outputs of an ANN and the corresponding SNN is an intricate function of the layer-wise approximations coming from (5) and accumulated error from the previous layers. Namely, for a given input sample, the error at a particular layer l (accumulated sum of differences at neuronal level) depends on the conversion errors made at all the previous layers as well as the error made when approximating ANN outputs with firing rate of SNN neurons. For a detailed discussion, one may refer to [Deng and Gu, 2021].

When dealing with layer-wise conversion, which measures how well the firing rate approximates the ANN ReLU output, one assumes that both the ANN and SNN layers receive the same input: $a^{(l-1)} = V_{\text{th}}^{(l-1)} \frac{\sum_{t=1}^{T} \mathbf{s}^{(l-1)}[t]}{T}$. Then, the conversion error made in the layer l for the output $a^{(l)}$ follows from the expression (3) above and is given by

$$\mathcal{C}(a^{(l)}) := \sum_{i=1}^{N^{(l)}} \Big(\frac{\mathbf{v}_i^{(l)}[T] - \mathbf{v}^{(l)}[0]}{T} \Big), \tag{6}$$

where $N^{(l)}$ is the number of neurons in the layer. From the previous expression it becomes evident the importance of the potential initialization $\mathbf{v}^{(l)}[0]$ and the membrane threshold V_{th} during the conversion. Both parameters have to be set in a way to minimize (the absolute value of) the overall conversion error, when $a^{(l)}$ ranges through the activation values.

For the threshold parameter $V_{\text{th}}^{(l)}$, classically one takes $M^{(l)} := \max\{a_i^{(l)} \mid a^{(l)} \text{ is an activation vector vector}\}$ [Diehl et al., 2015, Sengupta et al., 2019], usually after the weight normalization, or $V_{\text{th}}^{(l)}$ is taken to be some fraction of $M^{(l)}$ [Rueckauer et al., 2016]. A different approach is taken in [Bu et al., 2021] where $V_{\text{th}}^{(l)}$ is learned during necessary training of ANN. In Section 4 we propose to choose $V_{\text{th}}^{(l)}$ based on the *distribution* of activation values $a^{(l)}$ showing how it optimizes the conversion error (6).

For the voltage initialization $\mathbf{v}^{(l)}[0]$, both [Deng and Gu, 2021] and [Bu et al., 2021] use the intuition coming from Figure 1. where one can notice that the conversion error minimizes when slightly shifting the ReLU function (through the bias) or using some carefully chosen nonzero $\mathbf{v}^{(l)}[0]$. The former paper proposes shifting of the ReLU while the latter suggests taking $\mathbf{v}^{(l)}[0] = \frac{1}{2}V_{\text{th}}^{(l)}$, proving that this particular value minimizes the expectation of the error under the assumption (not explicitly stated in the paper) that the activation values follow a *uniform distribution*, or, more precisely, the activation values are *linear*. The reader can refer to Figure 1 A. for the graphical representation of the conversion error under these assumptions.

We revisit the term $\mathbf{v}^{(l)}[0]$ in Section 4 providing both practical and theoretical results on how to find its optimal value.

4 PROPOSED APPROACH

We propose an ANN-to-SNN conversion method of training SNNs, where not only the weights of the ANN model are used, but also the distribution of its layer-wise activation values. We show that these distributions can be used to find optimal membrane potential initialization and threshold, which play prominent role in minimizing the conversion error.

4.1 Activation values and threshold initialization

The feature distributions of layers in ANNs, when a representative set of samples of training dataset are passed to the network, are far from being regular/uniform[1]. In Figure 5 we show an example of a distribution and a plot of sorted features for ResNet18 ANN model (for a VGG16 model the corresponding figures can be found in the supplementary material). One can get a glimpse at the highly non-linear structure of the activation values. We also note here that, due to the high number of activation values ($\sim 10^6 - 10^8$) and their density, the obtained distributions and sorted arrays appear, at least visually, as continuous functions, but nonetheless they are discrete. However, at least as far as the theoretical results are concerned, discrete and continuous cases should be treated in a unified manner.

With this in mind, we revisit the steps of Sections 3.

4.1.1 Layer-wise conversion error

Generally speaking, layer-wise conversion error is a measure of difference between outputs at a particular layer of an ANN model and the corresponding SNN model.

We consider a layer l of ANN model with ReLU activation, and the corresponding SNN model initialized with weights of ANN. Let $a^{(l)} := a^{(l)}(x)$ be an output of ANN layer l and let $\tilde{a}^{(l)} := \tilde{a}^{(l)}(x) = V_{\text{th}}^{(l)} \frac{\sum_{t=1}^{T} \mathbf{s}_i^{(l)}(x)[t]}{T}$ be the corresponding output in SNN model (equation (5)) (here x is an input sample, so both outputs should be seen as functions of the input sample).

Let $\underline{\lambda}^{(l)}$ be the distribution of activation vectors of ANN and let $\underline{\Lambda}^{(l)}$ be the associated probability measure.

[1] Since the set of all activation values is discrete, by regularity we mean that the sorted activation values do not follow a linear pattern, or that they do not form an arithmetic sequence. We will keep this terminology throughout the paper

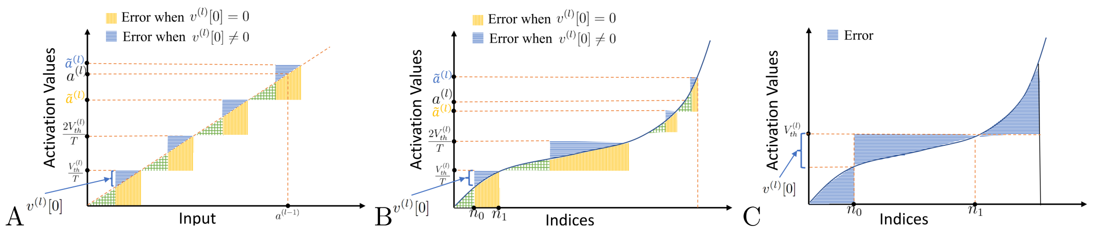


Figure 1: The plots show conversion error under different assumptions (green regions correspond to overlapping of the blue and yellow regions): A. The error obtained when assuming that the activation values are uniformly distributed; B. The error obtained taking into account activation values' distribution. The activation values are assumed to be sorted; C. The error made in one time step.

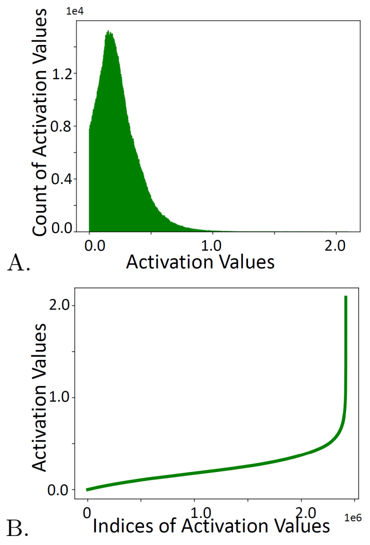


Figure 2: The feature distributions and plots of sorted feature values extracted at second ReLU layer of ResNet18 architecure using CIFAR10 dataset: The plots suggest that even discarding the outlier feature values, the remaining features are far from being uniformly distributed.

Then, the (signed) conversion error takes the form

$$\underline{\mathcal{C}}^{(l)}=\int\left(a^{(l)}-\tilde{a}^{(l)}\right)^{\mathrm{T}}\cdot J_{M^{(l)},1}d\underline{\Lambda}^{(l)}, \quad (7)$$

where $J_{M^{(l)},1}$ is an $M^{(l)}$-dimensional column vector of ones. In the case of a discrete (finite) distribution, the previous expression specifies to:

$$\underline{\mathcal{C}}^{(l)}=\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{M^{(l)}}\left((a_j^{(l)})_i-(\tilde{a}_j^{(l)})_i\right),$$

where N is the number of samples (counted with multiplicities) in distribution $\lambda^{(l)}$.

In this full generality, studying (7) and optimizing it in terms of parameters $V_{\text{th}}^{(l)}$ and $v^{(l)}[0]$ becomes a rather difficult challenge. To make things more tractable, we look for parameters $V_{\text{th}}^{(l)}$ and $v^{(l)}[0]$ that only depend on layers and not on a particular neuron in a layer, hence, we will consider a 1-dimensional distribution $\lambda^{(l)}$ (and its corresponding probability measure $\Lambda^{(l)}$) which is a distribution of all the activation values of all ANN neurons in the layer l, regarded independently of each other. Furthermore, similarly as in [Deng and Gu, 2021], we consider layer-wise conversion error without dependence on the accumulated error from the previous layers, which amounts to assuming that the SNN receives at layer l the same inputs $a^{(l-1)}$ as ANN, repeated at each time step $t=1,\ldots,T$ (see Section 3).

In particular, we consider: $a\sim\lambda^{(l)}$, $\tilde{a}:=V_{\text{th}}^{(l)}\frac{\sum_{t=1}^{T}s[t]}{T}$, where $s[t]$ is binary (0 or 1) output of the IF neuron (with potential threshold $V_{\text{th}}^{(l)}$ and membrane initialization $v^{(l)}[0]$), at a time step t, when the input a is coming at each time step. The conversion error (7) is then simplified to

$$\mathcal{C}^{(l)}=\mathcal{C}^{(l)}(V_{\text{th}}^{(l)},v^{(l)}[0],T):=\int(a-\tilde{a})d\Lambda. \quad (8)$$

The previous expression does naturally correspond to the signed error made during conversion. More generally, one may consider the more informative functions

$$\mathcal{C}_m^{(l)}=\mathcal{C}_n^{(l)}(V_{\text{th}}^{(l)},v^{(l)}[0],T):=\int|a-\tilde{a}|^m d\Lambda, \quad (9)$$

where m is a positive integer (for example, $m=2$ corresponds to the mean square conversion error).

4.1.2 Main results

As was already seen in equation (6), the parameters $V_{\text{th}}^{(l)}$ and $v^{(l)}[0]$ play a prominent role in minimizing the conversion error. As they are the only parameters in (8), our problem is to find

$$\underset{V_{\text{th}}^{(l)},v^{(l)}[0]}{\operatorname{argmin}}\,\mathcal{C}_m(V_{\text{th}}^{(l)},v^{(l)}[0],T), \quad (10)$$

where $\mathcal{C}_m(V_{\text{th}}^{(l)}, v^{(l)}[0], T)$ is as in equation (9). We note that the previous is a non-convex optimization problem and we refer to Figure 1 B. for a visual insight into $\mathcal{C}_1$.

We split the situation in two cases, depending on the latency, $T = 1$ and $T > 1$. In the former case, we provide a complete characterization of the optimal values for $V_{\text{th}}^{(l)}$ and $v^{(l)}[0]$, while in the latter, we provide enough theoretical understanding to provide practical algorithms for ANN-to-SNN conversion.

Case $T = 1$. In the first theorem that follows we consider continuous distribution λ, as it is easier to track the notation and it motivates the second result.

Theorem 4.1. *Suppose that $\lambda^{(l)}$ is a continuous distribution with support $[0, M]$ and let $\Lambda^{(l)}$ be the corresponding CDF. Then, there exists $x_0 \leq x_1 \in (0,1)$ such that $V_{\text{th}}^{(l)} = \left(\Lambda^{(l)}\right)^{-1}(x_1)$, $v^{(l)}[0] = \left(\Lambda^{(l)}\right)^{-1}(x_1) - \left(\Lambda^{(l)}\right)^{-1}(x_0)$, is a solution to* (10)*. Moreover,*

$$v^{(l)}[0] = \frac{1}{2}V_{\text{th}}^{(l)} \quad \text{and} \quad x_1 = \frac{1}{2}(1 + x_0).$$

In the discrete case, we have the following analogue.

Theorem 4.2. *Let $\left(a(n)\right)_{n=1}^{N}$ be the non-decreasing sequence of all the activation values, and let $V_{\text{th}}^{(l)}$ and $v^{(l)}[0]$ be a solution to optimization problem* (10)*. Then, there exist indices n_0 and n_1 such that $V_{\text{th}}^{(l)} = a(n_1)$ and $v^{(l)}[0] = a(n_1) - a(n_0)$.*

Moreover, n_0 is such that $|a(n_0) - \frac{1}{2}a(n_1)|$ is minimal, while $n_1 = n_0 + \left\lfloor \frac{1}{2}(N - n_0) \right\rfloor$.

Both of these results are proved in the supplementary material, but for the idea one can refer to Figure 3.2 and notice, for example, that by increasing the index n_1, the error increases or decreases, depending on the values of $N - n_1$ and $n_1 - n_0$. Similarly, the error increases or decreases with increasing n_0 depending on how far $a(n_0)$ is from $\frac{1}{2}a(n_1)$.

Although both results have their merits, the second theorem allows for an $\mathcal{O}(N)$ algorithm to find optimal $V_{\text{th}}^{(l)}$ and $v^{(l)}[0]$.

Case $T > 1$. This case presents a challenge in itself even for small latencies T. Nonetheless, the relation between $V_{\text{th}}^{(l)}$ and $v^{(l)}[0]$ that we present next and prove in the supplementary material, allows for the $\mathcal{O}(N^2 \log(N))$ algorithm, which is still useful in practice.

Theorem 4.3. *Keeping the assumptions and notation of Theorem 7.1, we have that $v^{(l)}[0] = \frac{1}{2}V_{\text{th}}^{(l)}$.*

Theorem 4.4. *Keeping the assumptions and notation of Theorem 7.2, we have that $|a(n_0) - \frac{1}{2}a(n_1)|$ is minimal.*

One may note that in the case of a uniform distribution λ, our theorem Theorem 4.3 recovers the main finding of [Deng and Gu, 2021].

4.2 Algorithms

Theorems 7.2 and Theorem 4.4 are what allows us to find optimal values for $V_{\text{th}}^{(l)}$ and $v^{(l)}[0]$ in terms of T and distributions of activation values λ. We present two algorithms, for the above cases $T = 1$ and $T > 2$. In both cases, the distribution λ is "given" in terms of sorted array of the ANN activation values. More details on both algorithms are given in the supplementary material.

In the first algorithm, E is the conversion error when using the current index i_0 to calculate the inner and outer threshold, while E_min and corresponding i_{min} are used to keep track of the optimal values.

Algorithm 1 Algorithm for computing optimal $V_{\text{th}}^{(l)}$ and $v^{(l)}[0]$ for $\mathcal{C}_1(V_{\text{th}}^{(l)}, v^{(l)}[0])$ and $T = 1$

```
Input: Sorted array of positive activation values $a$, of length $N$
Output: $V_{\text{th}}^{(l)}$ and $v^{(l)}[0]$
Initialize: $i_{min}, i_0 := 1$, $i_1 := i_{min} + (N - i_{min})//2$,
    $E, E_{\min} := \sum_{i=1}^{i_0-1} a[i] + a[i_1](N - i_0) - \sum_{i=i_0}^{i_1-1} a[i] + \sum_{i=i_1+1}^{N} a[i]$
for $i_0 = 2, \dots, N-2$ do:
    if $N - i_0 \mod 2 = 0$ do:
        $E += 2a[i_0] - a[i_1]$
        if $E < E_{\min}$:
            $E_{\min} := E$, $i_{min} := i_0$, $i_1 := i_{min} + (N - i_{min})//2$
    else:
        $E += 2a[i_0] - a[i_1] - 2a[i_1 + 1]$
         $+(N - i_0)(a[i_1 + 1] - a[i_1])$, $i_1 += 1$
        if $E < E_{\min}$:
            $E_{\min} := e$, $i_{min} := i_0$, $i_1 := i_{min} + (N - i_{min})//2$
return $a[i_1]$, $a[i_1] - a[i_{min}]$
```

For the second algorithm, given the index i of the sorted array of activation values a, we denote by $E(i)$ the conversion error obtained when using $a[i]$ as a threshold and $\frac{a[i]}{2}$ as the initial value of the membrane potential.

4.3 SNN calibration

Once the parameters of the model are initialized in the first part of the training, the model is already performing better in low latency than some of the state-of-the-art models. However, the true power of the proposed initialization lies in the fact that the converted model only needs a few epochs of training to drastically improve its performance (Figure 5).

The training is performed in a latency-increasing fash-

Algorithm 2 Algorithm for computing optimal $V_{\text{th}}^{(l)}$ and $v^{(l)}[0]$ for $\mathcal{C}_1(V_{\text{th}}^{(l)}, v^{(l)}[0])$ and $T > 1$

```
Input: Sorted array of positive activation values $a$, of length $N$
Output: $V_{\text{th}}^{(l)}$ and $v^{(l)}[0]$
Initialize: $N_{start}, N_{end}$,
        $ind_{min} = N_{start}, E_{min} := E(N_{start})$
for $i = N_{start} + 1, \ldots, N_{end}$ do:
    Calculate $E(i)$ ($\sim N\log(N)$ computational steps)
    if $E(i) < E_{\min}$:
        $E_{\min} := E$, $i_{min} := i$
return $a[i_{min}]$, $\frac{1}{2}a[i_{min}]$
```

ion for both computational and performance reasons (see [Deng et al., 2021]). More precisely, first initialized SNN model is trained for $T = 1$. After this, SNN model following the same architecture and the same potential and threshold initialization, but using the weights of $T = 1$ SNN model is trained for $T = 2$ and so on. The advantages of this way of training are shown experimentally in the following section.

5 EXPERIMENTS

We assess the effectiveness of our proposed method for classifying high-dimensional datasets such as CIFAR10, CIFAR100 [Krizhevsky and Hinton, 2009] and ImageNet [Russakovsky et al., 2015]. We train on these datasets on deep learning architectures such as VGG16, ResNet18, ResNet19 and CIFARNet. More details of experimental setup can be found in the supplementary. We compare our approach with the state-of-the-art ANN-SNN conversion methods (using ReLU as an activation in ANN models), but also in combination with other ANN-to-SNN conversion methods using auxiliary, ReLU -like activation functions. Finally, we also test our proposed method in combination with direct training, and the results can be found in the supplementary material.

5.1 ANN-to-SNN conversion: ReLU ANN activation

Firstly, we compare our method with SOTA methods based on ANN-to-SNN conversion, where ANN models use ReLU activation functions. on CIFAR10 and CIFAR100 datasets. For the baselines, we report the earliest reported accuracy until latency $T = 128$. For our method, we report the results for latencies from $T = 2$ until $T = 128$ (successive powers of 2). The results are presented in Table 3.

We emphasize that our method is able to obtain reasonable accuracies early on (especially in the case of CIFAR10) which we attribute to the method, as the thresholds are adapted to the dataset itself. Except for the few cases our method performs better than all the baselines in all the latencices (the exceptions are methods AP [Li et al., 2021a] and SNM+NN [Wang et al., 2022] for the case VGG16 and CIFAR100 (but we point to somewhat higher ANN accuracy of their models), and AP [Li et al., 2021a] for VGG16 and CIFAR10 for latency $T = 64$, but again with higher ANN accuracy of the baseline). Additionally, our method of initialization significantly improves the performance of SNM+NN model, as shown by Ours+SNM in the Table3.

5.2 Data driven initialization in combination with other methods

We test our method in combination with other methods used to train SNNs. We divide the baselines in two categories: ANN-to-SNN method, where ANN models use auxiliary activation functions, and direct training SNN methods, where one trains SNN models, either after ANN-to-SNN converison, either directly from scratch.

For the first category, we use our method in combination with Opt method [Deng and Gu, 2021] and QCFS method [Bu et al., 2021], where in the former case a truncated ReLU activation function is used instead of ReLU to train ANN models, while in the latter, a stairs-like function is used instead of ReLU . Both of the activation functions are designed to facilitate ANN-to-SNN conversion (that is, to reduce the conversion error). The first method does it by eliminating outliers from the distribution of activation values while the second one simplifies the distribution itself.

However, when using our method of initialization of SNNs from ANNs trained with QCFS method (stairs activation function), we recover the thresholds that are intrinsic to this method (related to the step size of the activation function, and depending on the latency). In particular, the performance of our method is identical to the one of QCFS, hence we do not report these results here (the details are in the supplementary material).

On the other side, we compare our method Opt [Deng and Gu, 2021]. We used their available pretrained models with truncated ReLU activation function, and we use our method to initialize the corresponding SNN. In Table 2, we compare the two methods' resulting SNN accuracies for different latencies. We once again conclude that our method vastly outperforms the baseline for the low latency, while in the top latency $T = 128$ the performances are comparable.

Finally, we tested our method in combination with direct training in ultra-low latency. Namely, the converted SNN models are further trained for few epochs,

Table 1: Comparison of our method with SOTA ANN-to-SNN conversion methods with ReLU ANN activation, on CIFAR10 and CIFAR100 datasets

Method	Architecture	ANN acc.	T =2	T =4	T =8	T =16	T =32	T =64	T =128
CIFAR100									
RMP [Han et al., 2020]	ResNet20	68.72	-	-	-	-	27.64	46.91	57.69
TSC [Han and Roy, 2020]	ResNet20	68.72	-	-	-	-	-	-	58.42
AP [Li et al., 2021a]	ResNet20	77.16	-	-	-	-	76.32	77.29	77.73
SNM+NN [Wang et al., 2022]	ResNet18	78.26	-	-	-	-	74.48	77.59	77.97
Ours	ResNet18	79.87	18.25	47.88	66.21	74.58	77.15	78.85	79.69
Ours	ResNet19	81.28	20.97	44.15	65.23	76.30	79.84	80.63	80.97
RMP [Han et al., 2020]	VGG16	71.22	-	-	-	-	-	-	63.76
TSC [Han and Roy, 2020]	VGG16	71.22	-	-	-	-	-	-	69.86
AP [Li et al., 2021a]	VGG16	77.89	-	-	-	-	73.55	76.64	77.40
SNM+NN [Wang et al., 2022]	VGG16	74.13	-	-	-	-	71.8	73.69	73.95
Ours	VGG16	75.49	42.24	51.21	53.65	57.12	61.61	70.44	73.82
CIFAR10									
RMP [Han et al., 2020]	ResNet20	91.47	-	-	-	-	-	-	87.60
TSC [Han and Roy, 2020]	ResNet20	91.47	-	-	-	-	-	69.38	88.57
AP [Li et al., 2021a]	ResNet20	95.46	-	-	-	-	94.78	95.30	95.42
RateNorm [Ding et al., 2021]	ResNet18	93.06	-	-	-	-	83.95	91.96	93.27
SNM+NN [Wang et al., 2022]	ResNet18	95.39	-	-	-	-	94.03	94.01	95.19
Ours	ResNet18	96.80	56.64	81.24	90.90	94.17	95.66	96.39	96.56
Ours	ResNet19	97.24	66.39	82.47	92.80	95.44	96.41	96.83	96.98
HT [Rathi et al., 2019]	VGG16	92.81	-	-	-	-	-	91.13	
RMP [Han et al., 2020]	VGG16	93.63	-	-	-	-	60.30	90.35	92.41
TSC [Han and Roy, 2020]	VGG16	93.63	-	-	-	-	-	92.79	93.27
AP [Li et al., 2021a]	VGG16	95.72	-	-	-	-	93.71	95.14	95.65
RateNorm [Ding et al., 2021]	VGG16	92.20	-	-	-	-	85.40	91.15	92.51
SNM+NN [Wang et al., 2022]	VGG16	94.09	-	-	-	-	93.43	94.07	94.07
Ours	VGG16	95.89	65.62	77.63	87.99	91.27	94.24	94.95	95.64
Ours+SNM	VGG16	95.89	70.29	85.44	92.57	94.64	95.25	95.57	95.53

Table 2: Data driven initialization in combination with ANN models with truncated ReLU on CIFAR100 and CIFAR10 datasets.

Method	Architecture	ANN acc.	T =2	T =4	T =8	T =16	T =32	T =64	T =128
CIFAR100									
Opt [Deng and Gu, 2021]	ResNet20	69.81	1.00	1.00	1.00	29.13	67.23	69.27	**69.72**
Ours	ResNet20	69.81	**48.13**	**58.61**	**63.86**	**67.42**	**69.03**	**69.61**	69.67
Opt [Deng and Gu, 2021]	VGG16	70.38	1	1	1	5.85	58.24	70.15	**70.46**
Ours	VGG16	70.38	**51.25**	**60.25**	**66.25**	**68.73**	**69.91**	**70.40**	70.37
CIFAR10									
Opt [Deng and Gu, 2021]	ResNet20	93.25	10.00	10.00	10.00	11.22	72.88	92.84	93.1
Ours	ResNet20	93.25	**75.00**	**88.14**	**91.32**	**92.60**	**92.92**	**93.15**	**93.26**
Opt [Deng and Gu, 2021]	VGG16	92.18	10.00	10.00	10.00	15.03	92.08	**92.33**	**92.30**
Ours	VGG16	92.18	**84.50**	**90.13**	**91.89**	**92.14**	**92.26**	92.22	92.20

the idea being that the additional training helps to adjust the weights to compensate for the accumulated error and inability of the SNN model to capture the ANN information in few time-steps. The details of this simple calibration procedure are presented in the supplementary material, along with the report on performance of such obtained SNN models, where we obtain many SOTA results in ultra-low latency.

6 CONCLUSION

In this work, we propose a novel approach for the ANN-to-SNN conversion where we argue that an already trained ANN model can be utilized not only for initializing the weights of the SNN model, but also for determining optimal initial neuron membrane potential and neuron membrane threshold values at different layers. We provide a theoretical framework for the distribution-based conversion error, and several results that characterize the inquired optimal values. Based on our theoretical findings, we propose practical algorithms to find the optimal values, and through experimental evaluations, we demonstrate that the converted SNN models can produce results that could surpass the baselines. Additionally, we also showed that our approach can be easily integrated with a tailored ANN model trained with a specific activation function.

We believe that our method can be further improved by applying it channel-wise instead of layer-wise, where the ANN activation value distributions are collected for every channel.

Acknowledgements

This work is part of the research project "ENERGY-BASED PROBING FOR SPIKING NEURAL NETWORKS" performed at Mohamed bin Zayed University of Artificial Intelligence (MBZUAI), in collaboration with Technology Innovation Institute (TII) (Contract No. TII/ARRC/2073/2021).

References

S. Anumasa, B. Mukhoty, V. Bojkovic, G. De Masi, H. Xiong, and B. Gu. Enhancing training of spiking neural networks with stochastic latency. In *Proceedings of the AAAI conference on artificial intelligence*, 2024.

L. Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

P. J. Braspenning, F. Thuijsman, and A. J. M. M. Weijters. *Artificial neural networks: an introduction to ANN theory and practice*, volume 931. Springer Science & Business Media, 1995.

T. Bu, W. Fang, J. Ding, P. Dai, Z. Yu, and T. Huang. Optimal ANN-SNN conversion for high-accuracy and ultra-low-latency spiking neural networks. In *International Conference on Learning Representations*, 2021.

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

X. Cheng, Y. Hao, J. Xu, and B. Xu. Lisnn: Improving spiking neural networks with lateral interactions for robust object recognition. In *IJCAI*, pages 1519–1525, 2020.

M. V. DeBole, B. Taba, A. Amir, F. Akopyan, A. Andreopoulos, W. P. Risk, J. Kusnitz, C. O. Otero, T. K. Nayak, R. Appuswamy, et al. Truenorth: Accelerating from zero to 64 million neurons in 10 years. *Computer*, 52(5):20–29, 2019.

S. Deng and S. Gu. Optimal conversion of conventional artificial neural networks to spiking neural networks. *International Conference on Learning Representations*, 2021.

S. Deng, Y. Li, S. Zhang, and S. Gu. Temporal efficient training of spiking neural network via gradient reweighting. In *International Conference on Learning Representations*, 2021.

P. U. Diehl and M. Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in computational neuroscience*, 9:99, 2015.

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)*, pages 1–8. ieee, 2015.

J. Ding, Z. Yu, Y. Tian, and T. Huang. Optimal ann-snn conversion for fast and accurate inference in deep spiking neural networks. *In International Joint Conference on Artificial Intelligence*, pages 2328–2336, 2021.

C. Duan, J. Ding, S. Chen, Z. Yu, and T. Huang. Temporal effective batch normalization in spiking neural networks. *Advances in Neural Information Processing Systems*, 35:34377–34390, 2022.

W. Fang, Z. Yu, Y. Chen, T. Huang, T. Masquelier, and Y. Tian. Deep residual learning in spiking neural networks. *Advances in Neural Information Processing Systems*, 34:21056–21069, 2021.

B. Han and K. Roy. Deep spiking neural network: Energy efficiency through time based coding. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part X*, pages 388–404. Springer, 2020.

B. Han, G. Srinivasan, and K. Roy. Rmp-snn: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13558–13567, 2020.

Z. Hao, J. Ding, T. Bu, T. Huang, and Z. Yu. Bridging the gap between ANNs and SNNs by calibrating offset spikes. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=PFbzoWZyZRX`.

A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500, 1952.

E. M. Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003.

H. Kamata, Y. Mukuta, and T. Harada. Fully spiking variational autoencoder. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 7059–7067, 2022.

S. Kim, S. Park, B. Na, and S. Yoon. Spiking-yolo: spiking neural network for energy-efficient object detection. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 11270–11277, 2020a.

S. Kim, S. Park, B. Na, and S. Yoon. Spiking-yolo: Spiking neural network for energy-efficient object detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(07):11270–11277, Apr. 2020b. doi: 10.1609/aaai.v34i07.6787. URL `https://ojs.aaai.org/index.php/AAAI/article/view/6787`.

A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *https://www.cs.toronto.edu/ kriz/cifar.html*, 2009.

Y. Li, S. Deng, X. Dong, R. Gong, and S. Gu. A free lunch from ann: Towards efficient, accurate spiking neural networks calibration. In *International Conference on Machine Learning*, pages 6316–6325. PMLR, 2021a.

Y. Li, Y. Guo, S. Zhang, S. Deng, Y. Hai, and S. Gu. Differentiable spike: Rethinking gradient-descent for training spiking neural networks. *Advances in Neural Information Processing Systems*, 34:23426–23439, 2021b.

Y. Li, D. Zhao, and Y. Zeng. Bsnn: Towards faster and better conversion of artificial neural networks to spiking neural networks with bistable neurons. *Frontiers in Neuroscience*, 16, 2022. ISSN 1662-453X. doi: 10.3389/fnins.2022.991851. URL `https://www.frontiersin.org/articles/10.3389/fnins.2022.991851`.

I. Loshchilov and F. Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2016.

W. Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(97)00011-7. URL `https://www.sciencedirect.com/science/article/pii/S0893608097000117`.

W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

B. Mukhoty, V. Bojkovic, W. de Vazelhes, X. Zhao, G. De Masi, H. Xiong, and B. Gu. Direct training of snn using local zeroth order method. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 18994–19014. Curran Associates, Inc., 2023.

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

J. Pei, L. Deng, S. Song, M. Zhao, Y. Zhang, S. Wu, G. Wang, Z. Zou, Z. Wu, W. He, et al. Towards artificial general intelligence with hybrid tianjic chip architecture. *Nature*, 572(7767):106–111, 2019.

N. Qiao, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, and G. Indiveri. A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses. *Frontiers in neuroscience*, 9:141, 2015.

N. Rathi and K. Roy. Diet-snn: A low-latency spiking neural network with direct input encoding and leakage and threshold optimization. *IEEE Transactions on Neural Networks and Learning Systems*, 34(6):3174–3182, 2023. doi: 10.1109/TNNLS.2021.3111897.

N. Rathi, G. Srinivasan, P. Panda, and K. Roy. Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. *International Conference on Learning Representations*, 2019.

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

B. Rueckauer, I.-A. Lungu, Y. Hu, and M. Pfeiffer. Theory and tools for the conversion of analog to spiking convolutional neural networks. *ArXiv*, abs/1612.04052, 2016. URL `https://api.semanticscholar.org/CorpusID:17743384`.

B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in neuroscience*, 11:682, 2017.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

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

Y. Wang, M. Zhang, Y. Chen, and H. Qu. Signed neuron with memory: Towards simple, accurate and high-efficient ann-snn conversion. In *International Joint Conference on Artificial Intelligence*, 2022.

Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in neuroscience*, 12:331, 2018.

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

Y. Yang, W. Zhang, and P. Li. Backpropagated neighborhood aggregation for accurate training of spiking neural networks. In *International Conference on Machine Learning*, pages 11852–11862. PMLR, 2021.

Z. Yang, Y. Wu, G. Wang, Y. Yang, G. Li, L. Deng, J. Zhu, and L. Shi. Dashnet: A hybrid artificial and spiking neural network for high-speed object tracking. *ArXiv*, abs/1909.12942, 2019. URL `https://api.semanticscholar.org/CorpusID:203593170`.

X. Yao, F. Li, Z. Mo, and J. Cheng. Glif: A unified gated leaky integrate-and-fire neuron for spiking neural networks. *Advances in Neural Information Processing Systems*, 35:32160–32171, 2022.

F. Zenke and T. P. Vogels. The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. *Neural computation*, 33(4):899–925, 2021.

W. Zhang and P. Li. Temporal spike sequence learning via backpropagation for deep spiking neural networks. *Advances in Neural Information Processing Systems*, 33:12022–12033, 2020.

H. Zheng, Y. Wu, L. Deng, Y. Hu, and G. Li. Going deeper with directly-trained larger spiking neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11062–11070, 2021.

L. Zhu, X. Wang, Y. Chang, J. Li, T. Huang, and Y. Tian. Event-based video reconstruction via potential-assisted spiking neural network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3594–3604, 2022.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]
2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]
3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Not Applicable]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]
5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Supplementary

7 PROOFS OF THEORETICAL RESULTS FROM SECTION 4.1.2 AND FURTHER DISCUSSION

7.1 Proofs

Theorem 7.1. *Suppose that $\lambda^{(l)}$ is a continuous distribution with support $[0, M]$ and let $\Lambda^{(l)}$ be the corresponding CDF. Then, there exists $x_0 \leq x_1 \in (0,1)$ such that $V_{\text{th}}^{(l)} = \left(\Lambda^{(l)}\right)^{-1}(x_1)$, $v^{(l)}[0] = \left(\Lambda^{(l)}\right)^{-1}(x_1) - \left(\Lambda^{(l)}\right)^{-1}(x_0)$, is a solution to Equation **10**(Main paper). Moreover,*

$$v^{(l)}[0] = \frac{1}{2}V_{\text{th}}^{(l)} \quad \text{and} \quad x_1 = \frac{1}{2}(1 + x_0).$$

Proof. Clearly, both $V_{\text{th}}^{(l)}$ and $v^{(l)}[0]$ have to take values in $[0, M]$, where M is the maximum of the support of $\lambda^{(l)}$. Since $(\Lambda^{(l)})^{-1}$ is continuous and surjectively maps onto $[0, M]$, the first part of the theorem follows. As for the next part, we can express the conversion error as (see Figure 3)

$$\mathcal{C}_m^{(l)} = \int_0^{x_0} \left((\Lambda^{(l)})^{-1}(x)\right)^m dx \tag{11}$$

$$+ \int_{x_0}^{x_1} \left((\Lambda^{(l)})^{-1}(x_1) - (\Lambda^{(l)})^{-1}(x)\right)^m dx \tag{12}$$

$$+ \int_{x_1}^{1} \left((\Lambda^{(l)})^{-1}(x) - (\Lambda^{(l)})^{-1}(x_1)\right)^m dx. \tag{13}$$

Let us fix x_1 and look for the x_0 which minimizes the previous expression, which is equivalent to minimizing the sum of the expressions (11) and (12). Taking the derivative of this sum with respect to x_0 (using Leibniz integral rule), we obtain that it is equal to

$$\left((\Lambda^{(l)})^{-1}(x_0)\right)^m - \left((\Lambda^{(l)})^{-1}(x_1) - (\Lambda^{(l)})^{-1}(x_0)\right)^m.$$

This last expression is 0 precisely when $(\Lambda^{(l)})^{-1}(x_0) = \frac{1}{2}(\Lambda^{(l)})^{-1}(x_1)$.

Similarly, let us fix x_0 and look for x_1 which minimizes the error. This in turn is equivalent to minimizing the sum of (12) and (13). We can rewrite the sum as

$$\int_{(\Lambda^{(l)})^{-1}(x_0)}^{(\Lambda^{(l)})^{-1}(x_1)} \left(\Lambda^{(l)}(x) - x_0\right)^m dx + \int_{(\Lambda^{(l)})^{-1}(x_1)}^{(\Lambda^{(l)})^{-1}(1)} \left((\Lambda^{(l)})^{-1}(1) - \Lambda^{(l)}(x)\right)^m dx.$$

Again taking the derivative with respect to x_1 and equating it with 0 we obtain that $x_1 = \frac{1}{2}(1 + x_0)$. □

Theorem 7.2. *Let $\left(a(n)\right)_{n=1}^N$ be the non-decreasing sequence of all the activation values, and let $V_{\text{th}}^{(l)}$ and $v^{(l)}[0]$ be a solution to optimization problem Equation **10**(Main paper). Then, there exist indices n_0 and n_1 such that $V_{\text{th}}^{(l)} = a(n_1)$ and $v^{(l)}[0] = a(n_1) - a(n_0)$.*

Moreover, n_0 is such that $|a(n_0) - \frac{1}{2}a(n_1)|$ is minimal, while $n_1 = n_0 + \left\lfloor \frac{1}{2}(N - n_0) \right\rfloor$.

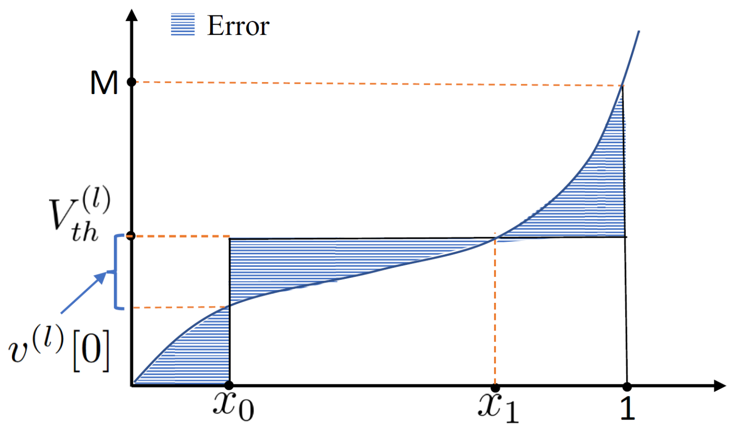


Figure 3: Plot of the inverse CDF function $\left(\Lambda^{(l)}\right)^{-1}(x)$.

Proof. Suppose that $v^{(l)}[0]$ and $V_{\text{th}}^{(l)}$ is the solution of the problem Equation **10**(Main paper), and put $v := V_{\text{th}}^{(l)} - v^{(l)}[0]$. Then,

$$\mathcal{C}_m^{(l)} = \sum_{a(i)<v} \left(a(i)\right)^m + \sum_{\substack{a(i)\geq v\\ a(i)\leq V_{\text{th}}^{(l)}}} \left(V_{\text{th}}^{(l)} - a(i)\right)^m + \sum_{a(i)\geq V_{\text{th}}^{(l)}} \left(a(i) - V_{\text{th}}^{(l)}\right)^m. \tag{14}$$

Since the points $a(i)$ take values in a discrete set, one can see that we can take $v' := \max\{a(i) \mid a(i) < v\}$ which still minimizes the problem Equation **10**(Main paper).

Now, if

$$\#\{a(i) \mid a_i \geq V_t^{(l)}h\} > \#\{a(i) \mid a(i) \geq v, a(i) \leq V_{\text{th}}^{(l)}\},$$

then moving $V_{\text{th}}^{(l)}$ to $\min\{a(i) \mid a(i) \geq V_{\text{th}}^{(l)}\}$ will further reduce the error, while if

$$\#\{a(i) \mid a_i \geq V_t^{(l)}h\} \leq \#\{a(i) \mid a(i) \geq v, a(i) \leq V_{\text{th}}^{(l)}\},$$

then moving $V_{\text{th}}^{(l)}$ to $\max\{a(i) \mid a(i) \leq V_{\text{th}}^{(l)}\}$ will reduce the error. This finishes the proofs of the statement related to $V_{\text{th}}^{(l)}$. Finally, one can notice that the taking v such that $|\frac{1}{2}V_{\text{th}}^{(l)} - v|$ is minimal, makes the sum of first two terms in (14), hence the overall error. □

Theorems **4.3** and **4.4** in main paper are proved in the same fashion as the corresponding statements of theorems 7.1 and 7.2, respectively.

7.2 Further discussions

In this section, we provide some further insight into our results, as well as a discussion concerning results in higher latency. We provide some questions that the research community may find interesting in order to further push the methods of this paper.

We consider the situation from two aspects, theoretical and practical. Theoretical one deals with insights about general distributions (continuous or not) and the minimization problem that we stated in the paper, while the practical aspect is concerned with efficient ways to solve the minimization problems on discrete distributions.

Theoretical considerations: To say a bit more, let λ be our distribution, Λ the corresponding CDF and let $\Theta = \Lambda^{-1}$ (This is the function we used in the proofs of our main results). So, the function Θ is defined on the segment $[0, 1]$ and we assume that $\Theta(1) = M < \infty$, as this is the case of interest. Now let T be the latency and V_{th} the threshold used for that particular latency. Note that the optimal initial membrane potential is $\frac{1}{2}V_{th}$ as follows from Theorem 4.3 in our paper. Before writing the precise conversion error formula, we refer to Figure 1 (b) (The indices in this figure correspond to the segment [0,1], the vertical axes contains the image of function Θ.

The conversion error will be the area of green and blue curved triangles). Then, let us denote by, for $i = 0, \dots, T$ the set $A_i = \{\xi \mid i \cdot V_{th} \le \Theta(\xi) \le i \cdot V_{th} + \frac{1}{2}V_{th}\}$. The region A_i contains those values of ξ for which $\Theta(\xi)$ is approximated with $i \cdot V_{th}$ (green triangles and their x-axis support in Figure 1. B in the article). Similarly, let us denote for $i = 1, \dots, T$, by $B_i := \{\xi \mid i \cdot V_{th} - \frac{1}{2}V_{th} \le \Theta(\xi) \le i \cdot V_{th}\}$ and note that B_i contains precisely those values of ξ for which $\Theta(\xi)$ is approximated by $i \cdot V_{th}$ (blue triangles and their x-axis support in Figure 1. B in the article). Note that regions A_i and B_i partition the segment [0,1] and may overlap only at endpoints. With this notation, the conversion error can be written as

$$\mathcal{C}_m(V_{th}, T) = \sum_{i=0}^{T} \int_{A_i} (\Theta(\xi) - i \cdot V_{th})^m \, d\xi + \sum_{i=1}^{T} \int_{B_i} (i \cdot V_{th} - \Theta(\xi))^m \, d\xi. \tag{15}$$

The peculiarity of the latency $T = 1$ lies in the fact that only three regions appear A_0, B_1 and A_1, which means the whole situation can be described with two points in the segment $[0, 1]$ (these two points are basically the endpoints of the segment B_1. Their images under the Θ map are precisely $v[0]$ and V_{th}). When considering the solution of the minimization problem - find V_{th} such that $\mathcal{C}_m(V_{th}, T)$ is minimal-, because of the simplicity of the situation, one is able to determine further relations between the two points of interest (in our paper these two points are x_0 (and $\Theta(x_0) = \frac{1}{2}V_{th}$) and x_1 (and $\Theta(x_1) = V_{th}$). Besides the relation $\Theta(x_0) = \frac{1}{2}\Theta(x_1)$ we are able to note also the relation $x_1 = \frac{1}{2}(1 + x_0)$. In particular, this is sufficient to have an efficient implementation algorithm in the case of discrete distributions.

In this situation, we may still pose the following question

1. Do the two relations provided in Theorem 4.1 determine *the* solution of the minimization problem? We expect the answer to be no. In fact, it is not clear to us whether the solution of the minimization problem is unique or not.

When considering higher latency $T \ge 2$, one may notice the significant complication of the situation in that there are $2T + 1$ regions to consider which are determined with $2T$ points. We are not able to provide any insight in this general situation beyond our theorem 4.3. More precisely, when considering the solution of the minimization problem, on the vertical axis we have the relation $v[0] = \frac{1}{2}V_{th}$. But, we are not able to determine any relation that would hold among the points on the x-axis (if we denote these points by $x_0, x_1, x_2, \dots$, note that we have $v[0] = \Theta(x_0) = \frac{1}{2}V_{th} = \frac{1}{2}\Theta(x_1)$, $\frac{3}{2}V_{th} = \Theta(x_2)$ and so on...).

2. In this general situation, and for V_{th} that is the solution of the minimization problem, do we have any relation among the points $x_0, x_1, \dots$? When λ is a uniform distribution, that is when Λ and Θ are linear functions on their domain, this problem can be tackled with standard Lagrange multipliers techniques (at least for $m = 1$ as all the integrals involved can be explicitly solved) and as you may suspect, we recover the solution which is the most intuitive one.

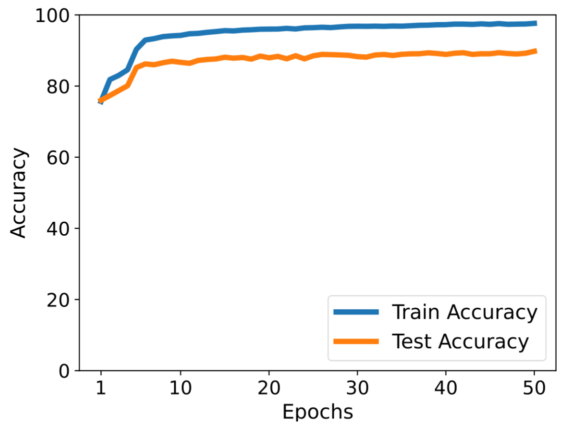
In a little shift of perspective, one could somewhat simplify the situation with assuming that $v[0] = 0$ and assume that we are approximating the function Θ from "bellow" (meaning each value $\Theta(\xi)$ is approximated with some $i \cdot V_{th} < \Theta(\xi)$). Note that the integrands in the above equation all have the same sign so the situation somewhat simplifies.

3. The same as question 2. Again as before, in the situation of uniform distributions we can answer this question rather straightforwardly.

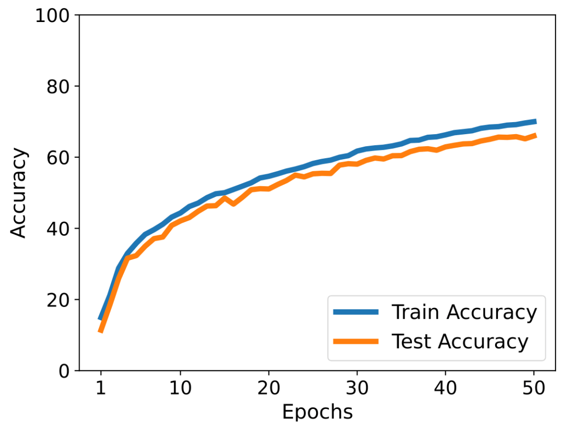
Finally, the above questions all fall under the umbrella of a very simple, but abstract one:

4. How to approximate arbitrary distributions with discrete ones having fixed support, so that the error of the approximation (however we define it) is minimal?

Practical considerations: When considering situation of discrete distributions coming from activation values of ANN models, ideally one would like to have an efficient algorithm that receives as input the list of activation values, and outputs the optimal threshold for a particular latency T. This is in particular important as the sets



(a)



(b)

Figure 4: Training and test accuracy plots of the VGG16 SNN models with T=1 on CIFAR10 (no augmentation) dataset, initialized with VGG16 ANN weights. Figure 4(a) shows the performance of SNN model over epochs with threshold values initialized with our approach. Figure 4(b) shows the performance of SNN model over epochs with threshold values initialized using the max activation value computed at the corresponding ReLU layer.

of activation values can be of the size anywhere to 10^8 or more. When we are in situation $T = 1$, the relation among the points x_0 and x_1 (when translated to indices of the list) allow us to have such an algorithm which scales linearly with the size of the input list. Because of the the lack of such relations in higher latency, we were not able to find a corresponding efficient algorithm. The algorithm for higher latency implemented in this article scales quadratically with the number of activation values, hence it is impractical when they are more than 10^6 in number. Hence the question:

5. Find an efficient algorithm that solves the minimization problem in higher latency.

8 ADDITIONAL DISCUSSION ON EXPERIMENTS

8.1 Experimental Setup

We trained all the models for 300 epochs, the initial learning rate for CIFAR10 dataset is 0.1 and for CIFAR100 dataset is 0.02. The stochastic gradient descent(SGD) [Bottou, 2012] optimizer is used to learn the weights of the model; the weight decay is set to 5e-4 and the momentum parameter is set to 0.1. Similar to [Li et al., 2021a, Bu et al., 2021, Hao et al., 2023], we adopt data-augmentation techniques for CIFAR10 and CIFAR100 when training for VGG16, ResNet18 and ResNet19 architectures to further improve the performance of the models. Over epochs, the learning rate is adjusted using the cosine decay scheduler [Loshchilov and Hutter, 2016]. After the models are trained, we extract the activation values at each ReLU layer. We use 3000 data points for CIFAR10, CIFAR100 and 500 datapoints for ImageNet from the training data for computing the membrane threshold and potential initialization values for each layer. NVIDIA's A100 is used to train all the models.

For $T = 1$ latency, we used the full activation values array obtained from the samples passed, while for $T > 1$ we reduced the number of elements by taking every 10th or every 100th element of the array, so that in the end we end up with $\sim 10^6$ elements in the array. SNN model is then obtained by copying the weights and biases of the ANN model, and replacing the ReLU layer with a Spike layer. The thresholds and membrane initialization are set to the values which are computed using the activation values at different layers, for $T = 1$ we used the first algorithm from the main paper, while for other latencies we used the second algorithm.

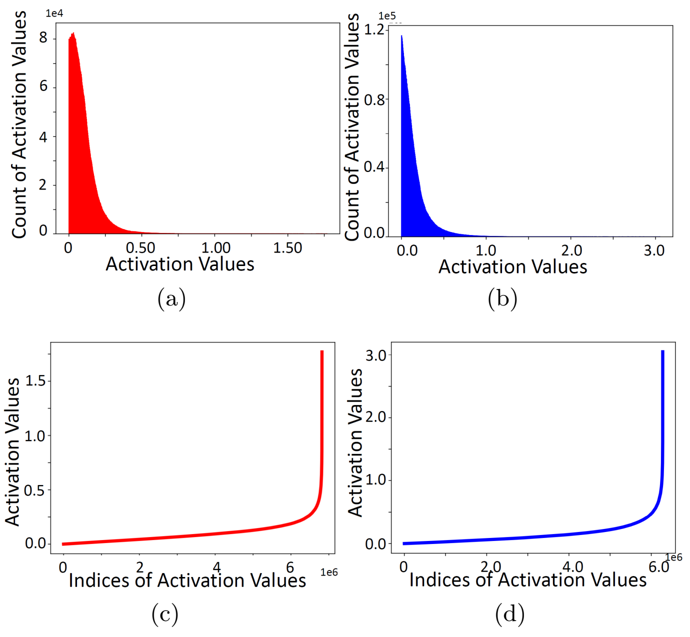


Figure 5: The feature distributions and plots of sorted feature values extracted at second ReLU layer of different architectures using CIFAR10 dataset: 5(a) and 5(c) ResNet18,5(b) and 5(d) VGG16 architectures. The plots suggest that even discarding the outlier feature values, the remaining features are far from being uniformly distributed.

8.2 Data driven initialization in spiking neural networks and direct training

The SNN models initialized with our method, need higher latency in order to be comparable with the corresponding ANN models, if the ANN models are . This is due to the fact that the float point outputs of the ANN activations cannot be matched with few spikes emanated in lower latency. Nonetheless, even in lower latency they are able to capture substantial information about the ANN model, which does not necessarily show in the immediate performance after the conversion, but rather in the potential obtained after training for a few epochs, which we explain next.

To start with, we compared training for 50 epochs of two SNN models, both having latency $T = 1$. Both of the models have been obtained by ANN-to-SNN conversion, from a pretrained VGG16 ANN model on CIFAR10 dataset. However, one model has been initialized with our method, while for the other we use the maximum activation method to set up the threshold and initial membrane potential. With our approach just after conversion the test accuracy was 40%, and after training (or better say fine-tuning) just for few epochs the accuracy reached 85%. On the other side, the alternative model even after 50 epochs did not reach the accuracy of our model at 10 epochs. Figure 4 contains the details of this experiment.

In conclusion, this experiment demonstrates the effectiveness of our proposed approach for initializing the threshold and initial membrane potential values with optimal values. To further use this property of our method in lower latency, we devised a simple training procedure - calibration - to use our method of conversion in combination with direct training of SNNs.

8.2.1 SNN-Calibration

In general, we start by training the converted SNN model with $T = 1$ for 50 epochs using surrogate method [2] (with a learning rate of $5e^{-3}$). We obtain the SNN models with higher latency ($T = 2, 3, 4$) after training for additional 20 epochs. The weights of the SNN model with latency T are initialized with the weights of already trained SNN model with $T = 1$ latency.

[2] $\frac{d\mathbf{s}^{(l)}[t]}{d\mathbf{v}^{(l)}[t]} = sign(|\mathbf{v}^{(l)}[t] - V_{\text{th}}^{(l)}| < a)$

8.2.2 Results for CIFAR10, CIFAR100 and ImageNet datasets

Table 3 shows the performance of the proposed model compared with state-of-the-art ANN-to-SNN conversion based and directly trained models for CIFAR100 and ImageNet dataset. For the baselines, we report the accuracy achieved for the reported latency closest to ours. For ultra-low latency values ($T \leq 4$), for all the architectures our proposed model exhibits exceptional performance in terms of SNN accuracy. For the complex dataset such as ImageNet with 1000 classes, our approach achieved a test accuracy of 69.28 with latency as low as $T = 2$ for VGG16 architecture, the accuracy achieved by our model is higher than all the baselines. (Although the baseline QCFSC [Hao et al., 2023] reported their test accuracies for $T = 1$, they require few additional post-processing steps(ρ) for each input at each layer, resulting in latency needed to obtain the predictions which is approximately ρL, where L is the number of activation layers in the model. This is the latency that we report in the tables, using their reported values for ρ).

Table 3: Comparison with the state-of-the-art direct training methods on different datasets. For the baselines, we report the accuracy achieved for the reported latency closest to ours or the best reported.

Dataset	Methods	Model	Architecture	T	Accuracy
CIFAR100	ANN2SNN	RMP [Han et al., 2020]	ResNet20	2048	67.82
	ANN2SNN	Opt [Deng and Gu, 2021]	ResNet20	512	72.34
	ANN2SNN	QCFS [Bu et al., 2021]	ResNet18	4	75.64
				2	70.79
	ANN2SNN	QCFS [Bu et al., 2021]	VGG16	4	69.62
				2	63.79
	Hybrid training	HC[Rathi et al., 2019]	VGG-11	125	67.87
	Direct-Training	Dspike [Li et al., 2021b]	ResNet-18	6	74.24
				4	73.35
				2	71.68
	Direct-Training	Diet-SNN[Rathi and Roy, 2023]	ResNet-20	5	64.07
	Direct-Training	STBP[Zheng et al., 2021]	ResNet-19	6	71.12
				4	70.86
				2	69.41
	Direct-Training	TET[Deng et al., 2021]	ResNet-19	6	74.72
				4	74.47
				2	72.87
	Direct-Training	TEBN[Duan et al., 2022]	ResNet-19	6	78.76
				4	78.71
				2	**78.07**
	Hybrid training	**Ours**	ResNet-19	4	**79.58**
				2	**77.83**
				1	**74.63**
	Hybrid-Training	**Ours**	ResNet18	4	**76.32**
				2	**74.37**
				1	**72.35**
	Hybrid-Training	**Ours**	VGG16	4	**72.58**
				2	**71.28**
				1	**68.43**
ImageNet	Hybrid-Training	HC[Rathi et al., 2019]	ResNet-34	250	61.48
	ANN2SNN	QCFS [Bu et al., 2021]	VGG16	32	68.47
	ANN2SNN	QCFSC [Hao et al., 2023]	VGG16	≈ 8	63.84
	Direct-Training	STBP-tdBN[Zheng et al., 2021]	Spiking-ResNet-34	6	63.72
	Direct-Training	SEW[Fang et al., 2021]	SEW-ResNet	4	67.04
	Direct-Training	TET[Deng et al., 2021]	SEW-ResNet34	4	68.00
	Direct-Training	TEBN[Duan et al., 2022]	SEW-ResNet34	4	68.28
	Direct-Training	Dspike[Li et al., 2021b]	VGG-16	5	71.24
	Direct-Training	GLIF[Yao et al., 2022]	ResNet-34	4	67.52
	Hybrid-Training	**Ours**	VGG16	2	**69.28**
	Hybrid-Training	**Ours**	VGG16	3	**70.15**

Table 4 provides the performance of the proposed approach with different baselines, both ANN-2-SNN approaches and direct training approaches. When compared to ANN-2-SNN models our approach has a better test accuracy both for VGG16 and ResNet18. When compared with the direct training models our approach with ResNet19, CIFARNet and AlexNet architectures shows a better performance in terms of SNN test accuracy at ultra low latency values.

Table 5 provides train accuracies of different datasets trained on different architectures. For ImageNet on VGG16 we downloaded already trained model[3]. If we compare the ANN accuracies from 5 and SNN test accuracies in Tables 3 and 4, the loss in test accuracy after the conversion is only around $2-3\%$. With this simple and effective approach a trained ANN model can be converted to SNN model following few calibration steps. It saves lot of time in training an SNN model from scratch when there is need to train a model for a huge dataset such as ImageNet.

Table 4: Comparison with the state-of-the-art direct training methods on CIFAR10 dataset.

Dataset	Methods	Model	Architecture	T	Accuracy
	ANN2SNN	RMP[Han et al., 2020]	VGG16	32	60.30
	ANN2SNN	Opt [Deng and Gu, 2021]	VGG16	32	76.24
	ANN2SNN	QCFS [Bu et al., 2021]	ResNet18	4	93.83
				2	91.75
	ANN2SNN	TSC[Han and Roy, 2020]	VGG16	64	92.79
	ANN2SNN	SNNC-AP[Li et al., 2021a]	VGG16	32	93.71
	ANN2SNN	QCFS [Bu et al., 2021]	VGG16	4	93.96
				2	91.18
CIFAR10	Hybrid Traning	HC[Rathi et al., 2019]	ResNet-20	250	92.22
	Direct-Training	Diet-SNN[Rathi and Roy, 2023]	ResNet-20	10	92.54
	Direct-Training	Diet-SNN[Rathi and Roy, 2023]	CIFARNet	5	91.54
	Direct-Training	STBP[Wu et al., 2018]	CIFARNet	12	89.83
	Direct-Training	STBP NeuNorm[Wu et al., 2019]	CIFARNet	12	90.53
	Direct-Training	BAAT[Yang et al., 2021]	AlexNet	5	91.76
	Direct-Training	TSSL-BP[Zhang and Li, 2020]	CIFARNet	5	91.41
	Direct-Training	STBP[Zheng et al., 2021]	ResNet-19	6	93.16
				4	92.92
				2	92.34
	Direct-Training	TET[Deng et al., 2021]	ResNet-19	6	94.50
				4	94.44
				2	94.16
	Direct-Training	TEBN[Duan et al., 2022]	ResNet-19	6	95.60
				4	95.58
				2	95.45
	Hybrid-Training	**Ours**	ResNet-19	4	**96.04**
				2	**95.55**
				1	**94.86**
	Hybrid-Training	**Ours**	CIFARNet	2	**92.50**
	Hybrid-Training	**Ours**	AlexNet	2	**92.59**
	Hybrid-Training	**Ours**	ResNet18	4	**94.19**
				2	**93.32**
				1	**93.00**
	Hybrid-Training	**Ours**	VGG16	4	**93.96**
				2	**92.97**
				1	**92.45**

8.3 Algorithms

In the main article, we presented two algorithms that are used in the experiments. Here, we describe the procedure behind the second second algorithm, which we used for finding optimal threshold initial membrane potential in latency $T > 2$.

Namely, the input of the algorithm is a sorted array a of feature values (ReLU activation values of an ANN

[3]https://pytorch.org/hub/pytorch_vision_vgg/

model at a particular layer without zeros) of length N, and desired latency T. We start by going through the indices $(N_{start}, \dots, N_{end})$, where N_{start} and N_{end} are chosen indices from 1 to N (in the experiments, we used $N_{start} = N//2$ and $N_{end} = N - 5$). Then, for each $i = N_{start}, \dots, N_{end}$:

1. We find the the T indices in the array that correspond to the values $j \cdot a[i]/T$, $j = 1, \dots, T$. Since the array is discrete, we are looking for the closest values. This step approximately takes $\log N$ computational steps.

2. We calculate the conversion error $E(i)$ which is the error obtained if we were to use $a[i]/T$ and $\frac{1}{2}a[i]/T$ for the threshold and initial membrane potential and when passing the elements of the array through the spiking neuron (the average firing rate multiplied with the threshold is then the approximation for the value. See the article for the details). This error is rather straightforward to calculate, and it requires one passing through the array, so approximately N computational steps.

3. Finally, we record that index i for which the error $E(i)$ was minimal, and use the corresponding values for initial membrane potential and threshold during the inference.

Table 5: ANN accuracies of CIFAR10,CIFAR100 and ImageNet datasets on different Architectures

Dataset	**Architecture**	**ANN-Test-Accuracy**
CIFAR10	VGG16	95.89
	ResNet18	96.80
	CIFARNet	93.99
	ResNet19	97.25
	AlexNet	93.09
CIFAR100	VGG16	75.49
	ResNet18	79.87
	ResNet19	81.28
ImageNet	VGG16	73.27

9 ENERGY ESTIMATION

To estimate the energy efficiency of our models, we used formula from [Rathi and Roy, 2023], Section 5 (at each layer one records the averege spikes per neuron during the latency when a sample image is presented. This number is then multiplied by the number of operations in that particular layer, and all the numbers are summed up over layers. The result estimates the number of synaptic operations (SOP) of the network when classifying the image). We use the energy values for FLOP and SOP reported in [Qiao et al., 2015]

For $T = 1$ the average energy consumption of our model VGG16, with CIFAR100 is 0.0064 mJ. For the ANN model following the same architecture, the energy consumption is 7.85 mJ. When comparing with baselines, for the QCFS model, for the latency $T = 2$ (the earliest latency for which they report the energy), the energy consumption is 0.007 mJ. However, we note that their accuracy for this latency is 63.79 while ours (for $T = 1$) is 68.43. The energy consumption for higher latency is then approximately proportional to the latency itself.