Temporal Pattern Coding in Deep Spiking Neural Networks

Rueckauer, Bodo ; Liu, Shih-Chii

DOI: https://doi.org/10.1109/ijcnn52387.2021.9533837

Accepted for publication at the 2021 International Joint Conference on Neural Networks (IJCNN)

Temporal Pattern Coding in Deep Spiking Neural Networks

Bodo Rueckauer and Shih-Chii Liu
Institute of Neuroinformatics
University of Zurich and ETH Zurich
Zurich, Switzerland
rbodo,shih@ini.uzh.ch

***Abstract*—Deep Artificial Neural Networks (ANNs) employ a simplified analog neuron model that mimics the rate transfer function of integrate-and-fire neurons. In Spiking Neural Networks (SNNs), the predominant information transmission method is based on rate codes. This code is inefficient from a hardware perspective because the number of transmitted spikes is proportional to the encoded analog value. Alternate codes such as temporal codes that are based on single spikes are difficult to scale up for large networks due to their sensitivity to spike timing noise. Here we present a study of an encoding scheme based on temporal spike patterns. This scheme inherits the efficiency of temporal codes but retains the robustness of rate codes. The pattern code is evaluated on MNIST, CIFAR-10, and ImageNet image classification tasks. We compare the network performance of ANNs, rate-coded SNNs, and temporal-coded SNNs, using the classification error and operation count as performance metrics. We also estimate the power consumption of the digital logic needed for the operations associated with each encoding type, and the impact of the bit precision of the weights and activations. On ImageNet, the temporal pattern code achieves up to $35\times$ reduction in the estimated power consumption compared to the rate-coded SNN, and $42\times$ compared to the ANN. The classification error of the pattern-coded SNN is increased by $< 1\%$ compared to the ANN, and decreased by 2% compared to the rate-coded SNN.**

I. Introduction

The question of neural coding - how signals are represented in the brain - has engaged neuroscientists for many decades [1]. With the revival of ANNs in the late 1980's, interest in the principles of neural coding has spread to the machine learning community and inspired pioneering work such as [2]. The subsequent success of Deep Learning (DL) was in part made possible by the increased availability of computational power from hardware in the form of Graphics Processing Units (GPUs), but the limited resources available from platforms used in real-time mobile applications continue to fuel the search for more efficient ways to process Deep Neural Networks (DNNs). If not based on practical considerations alone, this research effort may further be motivated by the realization that our brain is able to accomplish complex cognitive tasks with as little as 20 W power consumption.

SNNs have emerged as viable candidates for implementing DL architectures. Information between neurons in SNNs is typically transmitted in form of discrete events (spikes), rather than real values as in ANNs. SNNs use cheaper elementary operations than their analog counterparts (additions vs. multiply-accumulates (MACs)), and they can exploit spatial and temporal sparsity, i. e. to skip redundant computations where neurons in the network are inactive. Whereas ANNs are typically operated in a synchronous, frame-wise manner, the processing mode of SNNs is usually asynchronous and driven by changes in the input. This event-driven computing paradigm [3] is attractive for low-latency, low-power applications such as gesture recognition [4] and object detection [5].

However, operating DNNs in the spiking regime has its challenges, starting with the problem of obtaining suitable weights for the synaptic connections. Directly training the SNN using the supervised backpropagation training method of DL is difficult because of the non-differentiable nature of the spike generation mechanism. Employing surrogate gradients or a smoothed version of the activation function are promising ways to circumvent this problem [6], but direct training has not yet been shown to scale to full-scale models like VGG-16, GoogleNet, or ResNet. An alternate approach to spike-based training is conversion of an ANN, where a wealth of DL tools is available to obtain high starting accuracy on arbitrarily large models. A drawback of this approach is that there is often a drop in accuracy when porting the model into the spike domain. Many of these conversion methods employ a form of rate coding, where the average firing rate of a neuron in the SNN emulates the analog activation value of the corresponding neuron of the ANN. In a theoretical analysis, [7] showed that such a rate code can approximate the ANN to arbitrary precision, but only in the limit of a large simulation duration.

Since then, several conversion schemes have been proposed to maintain high SNN accuracy at lower computational cost. Reduced spike rates can be achieved e. g. by training the original ANN with a regularization term on the firing rates [8]. Unfortunately, low rates are more susceptible to approximation errors, which can be compensated by quantization-aware training [9], or by employing an adaptive neuron model [10]. A more rigorous approach is to forgo rate codes altogether and encode ANN activation values in single spikes or short firing patterns, where information is contained in individual spike timing. Biologically inspired temporal coding in SNNs has a long history (see e. g. [11], [12]), but only recently have these principles been applied to multi-layer networks [13]–[15].

This work was supported by the Samsung Advanced Institute of Technology, the University of Zurich and ETH Zurich.

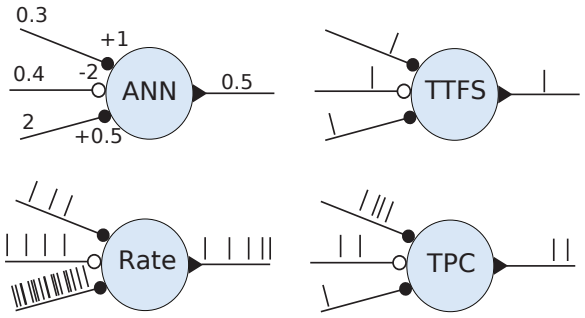


Fig. 1: Illustration of the four DNN operating modes studied in this paper.

This paper compares the accuracy and computational cost of three of these spike codes. In particular, we evaluate an encoding based on temporal firing patterns which aims to find a middle ground between the fast but brittle temporal codes and the slow rate codes. In our temporal pattern code (TPC), individual spikes carry information in their timing, but rather than relying on a single spike per neuron we allow for subsequent spikes to supplement the message. To achieve this, the analog value to be encoded is transformed into a string of bits, for instance using the binary number format. This transformation introduces the notion of time: each nonzero bit in the string represents a spike in a firing pattern. The amount of information carried by individual spikes in the pattern decreases from a spike at the position of the most-significant bit (MSB) to that of the least-significant bit (LSB).

An immediate benefit of this encoding is the ability to trade off operations and latency against accuracy: By dropping some number of LSBs, we can sacrifice accuracy to save operations and speed up inference time. Exploiting the use of low-precision parameters in deep neural networks for efficient inference has been a topic of great interest, see e. g. [16]. In contrast to low-precision ANNs, where dedicated processing elements are required to perform operations at different levels of quantization, precision in our coding scheme can be varied dynamically using the same processing elements, by reducing the number of time steps per spike sequence.

This paper contributes a direct comparison of ANN-to-SNN conversion methods with rate, latency and pattern codes, using classification error and operation count as metrics. The three encoding schemes are briefly reviewed in Sec. II. The converted SNNs are evaluated in Sec. III-A on MNIST, CIFAR-10, and ImageNet, using LeNet [17] and MobileNet [18] architectures. In Sec. III-C, the impact of low-precision ANN training is studied together with the estimated power consumption of the operations underlying each encoding. The paper concludes in Sec. IV with a discussion of the advantages and drawbacks of the three spike codes, and a comparison against related work.

II. Methods

A schematic illustration of the operation modes of analog and spiking neural networks is shown in Fig. 1. In ANNs, an input sample is processed within a single forward pass by transmitting information in form of real values. SNNs use a state variable ("membrane potential") to integrate discrete spike events over time, and generate spikes themselves when crossing a threshold. The rate-based conversion method used here relies on the principle that the firing rate of an SNN neuron approximates the analog activation value of the corresponding ANN neuron. The latency code evaluated here represents ANN activation values by the inverse time-to-first-spike (TTFS). Every neuron fires at most once; the higher the ANN activation, the faster the spike is generated. The TPC conversion uses a finite sequence of spikes to encode neuron activations. A more detailed formulation of the rate and latency code is given in [7] and [14], respectively, so the remainder of this section focuses on the pattern code.

The basic principle underlying the temporal pattern encoding is that n-bit activation values in the ANN can be represented by a sequence of n spikes in the SNN, e. g. using the binary number format. The time of a spike from a pre-synaptic neuron i determines the amount of charge that is added to the membrane potential of a post-synaptic neuron j. For instance, assume neuron i has an activation value of 9, whose binary representation is 00001001 in an $n = 8$ bit word. Thus, neuron i will fire only two spikes, the first after 5 time steps, and the second at time step 8. The membrane potential of the post-synaptic neuron j is increased by the corresponding amount of charge proportional to the synaptic weight w_{ij}. To account for the relative spike timing, the synaptic strength is modulated by a time-dependent kernel.

We formulate the dynamics of the membrane potential using the spike-response model [19]:

$$u_i(t) = \sum_{j\in\Gamma_i}\sum_{t_j\in\mathcal{F}_j} w_{ij}\epsilon_{ij}(t-t_j) + I_{\text{ext}}(t) + \sum_{t_i\in\mathcal{F}_i}\eta_i(t-t_i), \quad (1)$$

where the kernel ϵ_{ij} represents the post-synaptic potential (PSP) caused by an input spike from one of the pre-synaptic neurons in Γ_i, and the external input current $I_{\text{ext}}(t)$ is given by the bias b_i of a neuron or feature map in the ANN. The term η_i models the shape of the action potential and the refractory period, and is not used in this work. The time $t \in [0, n]$ is limited to the number of bits n, at the end of which the membrane potential is reset and a new sample (e. g. image frame) is presented for the next n time steps. This time threshold n replaces the voltage threshold that was used to trigger spikes in previous coding schemes. When reaching the time threshold, the post-synaptic neuron i fires a burst of spikes, whose times $\mathcal{F}_i = \{t_i\}$ correspond to the binary representation of the accumulated membrane potential. Such

a time threshold could be implemented via a periodically firing "clock" neuron (see Sec. IV for a discussion).

Defining the synaptic kernel ϵ as

$$\epsilon_{ij}(t - t_j) = \mathrm{H}(t - t_j) \cdot 2^{-t_j+n-1}, \tag{2}$$

where H is the Heaviside step function, we arrive at the final expression for the membrane potential:

$$u_i(t < n) = \sum_{j \in \Gamma_i} \sum_{t_j \in \mathcal{F}_j} w_{ij} \cdot \mathrm{H}(t - t_j) \cdot 2^{-t_j+n-1} + b_i. \tag{3}$$

By construction, at $t = n$, the accumulated kernel contribution $\sum^{\mathcal{F}_j} 2^{-t_j+n-1}$ of pre-synaptic neuron j equals exactly that neuron's membrane potential, $u_j(n)$. In the input layer, this value corresponds to the pixel activation a_j. Thus, membrane potentials in the first hidden layer are a perfect reflection of the corresponding activation values in the ANN[1]: $u_i(t = n) = \sum^{\Gamma_i} w_{ij} \cdot a_j + b_i = a_i$. The same argument extends iteratively to higher layers, which asserts that the encoding is loss-less if all n time steps are used during simulation. A lossy encoding is obtained by running for k fewer time steps, which is equivalent to truncating the last k bits of the binary representation. The effect on accuracy and computational cost is evaluated in Sec. III-A.

1) Notes on Reproducibility: The temporal pattern code is implemented using Python 3.7 and Tensorflow 2.2.0 and its first version has been open-sourced mid 2017 as part of the SNN toolbox [7]. This software package also includes tools for latency- and rate-based conversion, which were evaluated here. Since inference with the SNN is deterministic, each network was run once on all the samples of the respective dataset. Vertical bars around data points in the figures below (if large enough to be visible) indicate the 95% confidence interval for the classification error. Horizontal bars denote the standard deviation of the operation count when averaging over all samples in the dataset. The networks are run on a GeForce GTX 980 Ti GPU using a desktop computer with 32 GB memory running Ubuntu 18.04.

III. Results

To evaluate the performance and robustness of TPC, we measure the classification error of models trained with 32 bit weights and activations, after conversion to TPC at different values of the bitwidth n (representing the number of inference time steps per sample). The resulting error-operations tradeoff curve allows comparison against the other spike codes and the original ANN performance. We first present tradeoff results for MNIST, CIFAR-10 and ImageNet (Sec. III-A). We then study the effect of training the ANN with low-precision weights and activations. Finally, we estimate the power consumption of the low-precision add, MAC, and shift operations needed for inference (Sec. III-C, Table II).

[1]The rectifying linear unit (ReLU) activation is assumed.

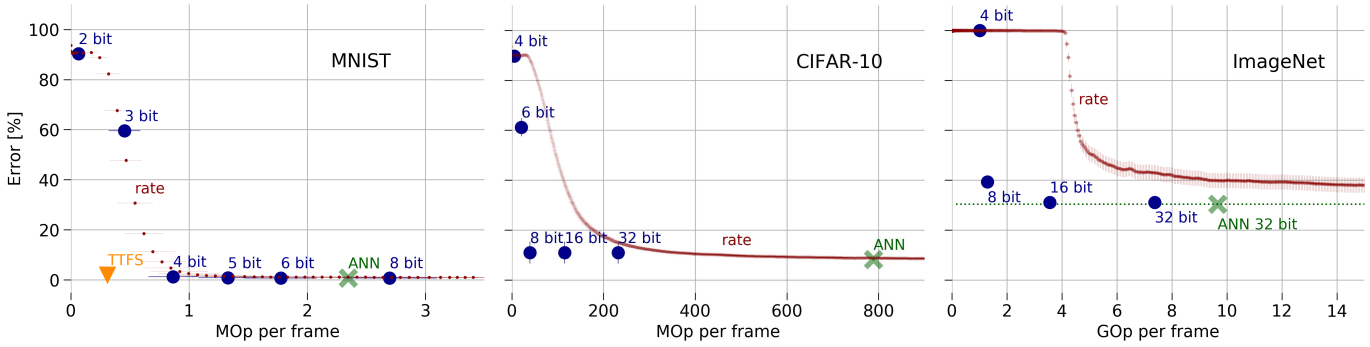
TABLE I: Comparison of classification error and computational cost between ANNs and SNNs using rate, latency (TTFS), and pattern (TPC) code. We highlight in bold the configuration that requires the fewest operations while staying within 1% of the ANN error.

Task	Encoding	Error [%]	Operations [1e6]
MNIST LeNet-5	ANN	0.72	2.35
	rate SNN	1.16	1.76
	TTFS	2.00	0.31
	TPC 2 bit	90.26	0.07
	TPC 3 bit	59.52	0.45
	TPC 4 bit	**1.26**	**0.87**
	TPC 5 bit	0.81	1.33
	TPC 6 bit	0.76	1.78
	TPC 8 bit	0.75	2.70
CIFAR-10 MobileNet-v1	ANN	8.82	789
	rate SNN	9.40	576
	TPC 4 bit	89.73	5
	TPC 6 bit	61.19	20
	TPC 8 bit	**8.82**	**39**
	TPC 16 bit	8.82	115
	TPC 32 bit	8.82	232
ImageNet MobileNet-v1	ANN	30.41	9663
	rate SNN	36.01	23812
	TPC 4 bit	100.00	1016
	TPC 8 bit	39.30	1293
	TPC 16 bit	**31.13**	**3561**
	TPC 32 bit	30.41	7372

A. Trading off Classification Error vs Operations

1) LeNet-5 on MNIST: As a proof-of-concept, we first evaluate the pattern code on the classic LeNet-5 architecture for MNIST [17]. After training the ANN in Tensorflow with standard DL procedure, the model was converted to an SNN using rate code, latency code, and pattern code presented here. As shown in Fig. 2a, both the rate and pattern converted SNNs perform similarly, with the classification error of the TPC network converging slightly faster to the error rate of the ANN. At $n = 4$ ("TPC 4 bit"), TPC requires $2\times$ fewer operations than the rate code and $2.7\times$ fewer operations than the ANN to achieve the same error rate. TTFS is another $2.6\times$ more efficient than TPC 4 bit in this experiment because only a single spike is fired per active neuron. However, this latency code suffers from unacceptable accuracy loss in the more difficult tasks studied below. The results are summarized in Table I.

2) MobileNet on CIFAR-10: For the CIFAR-10 classification [20] we choose the MobileNet-V1 [18] architecture which achieves good classification performance with comparably small parameter footprint due to the use of depthwise-separable convolutions. We trained the model using PyTorch 1.5.0, and achieved a classification error of 8.82% after 300 epochs using Adam optimizer, a batch size of 100, learning rate decay and mild data augmentation (horizontal flips, random crops). After conversion, the rate-based SNN achieves near state-of-the-art error rate for SNNs on this dataset, but requires about the same number of operations as the ANN. TPC 8 bit



(a) (b) (c)

Fig. 2: Classification error versus operation count per frame of the original ANN and the converted SNNs tested on MNIST (a), CIFAR-10 (b), and ImageNet (c). Three conversion methods are compared, based on rate (small dots), latency (triangle), and pattern encoding (large circles). For the pattern code, each data point represents an experiment where the model was run for n time steps per test sample, which is equivalent to truncating the network activations to n bit (as indicated by the label).

needs about $15\times$ fewer operations than the rate code, and $20\times$ fewer operations than the ANN at equal accuracy (c. f. Fig. 2b and Table I).

3) MobileNet on ImageNet: For the ImageNet dataset [21], we use the same architecture as for CIFAR-10, except that the resolution of the input images is higher (256×256 instead of 32×32). The ANN[2] achieves a top-1 error of 30.41% at 9.66 GOp per frame (Fig. 2c). The rate-encoded SNN fails to reach this error rate within the same computational budget.[3] TPC 32 bit reaches the same error as the ANN. Even though each analog activation value is replaced by a pattern of up to 32 spikes, the TPC model still requires $1.3\times$ fewer operations than the ANN, because TPC natively skips over zeros in the feature maps.[4] At 16 bit, TPC has to process fewer spikes, leading to $2.7\times$ fewer operations. The error increases by 0.72%, which is equivalent to the error obtained when naively rounding the ANN activations to 16 bits. Reducing run time further to $n = 8$ incurs a more substantial error increase (8.9%). This degradation can be avoided by training the ANN with 8-bit activations, which is described in more detail in Sec. III-C. The ImageNet results are summarized in Table I.

B. One-Hot Encoding

The efficiency of this pattern code can be further improved by quantizing activations to powers of 2. The binary pattern representation then becomes maximally sparse. We did not train the model ourselves to achieve equivalent accuracy at power-2 quantized activations, which has been shown in other work [24]. However, in a preliminary experiment, we found that MobileNet on ImageNet would require 550 MOp in a power-2 encoding, a reduction of $18\times$ compared to the ANN.

C. Estimated Power Consumption

One deficiency in the comparison in Sec. III-A is that we simply counted the raw number of operations, irrespective of their type. Standard ANNs use MACs operations; TPC networks require addition and bit-shift operations; rate- and latency-coded SNNs use only additions. To account for the different costs associated with each type of basic operation, we repeated the tradeoff analysis for ImageNet, this time weighting each operation with its respective power consumption in hardware. Instead of raw operation count, we then report the needed power consumption to perform the operations during inference on one frame.

This metric is calculated as follows:

$$C = \sum_{\text{op}}^{\{\text{Op types}\}} c_{\text{op}} N_{\text{op}}, \tag{4}$$

where $\{\text{Op types}\}$ is the set of operation types present in the considered models (MAC, addition, bitshift), c_{op} is the cost associated with a particular op, and N_{op} the number of times it is performed during inference of one sample.

The numbers c_{op} represent the power consumption relative to one adder. For two operands of bitwidth b_1 and b_2, the cost of addition follows $c_{\text{add}} = \max(b_1, b_2) + \frac{|b_1 - b_2|}{2}$; a MAC consumes $c_{\text{MAC}} = b_1 \cdot b_2$. To shift a b_1 bit word by b_2 possible bits would consume $\frac{2}{3} b_1$ relative to the adder. However, in our TPC models we only need to shift by 0 or 1 bits as the bitstring is processed serially, therefore $c_{\text{shift}} = \frac{b_1}{5}$. These power estimates for c_{op} were obtained using the Synopsys Design Compiler 2018 synthesis tool and a Globalfoundry 28 nm SLP process.

Since the cost of an operation depends on the bitwidth of the operands, we consider three scenarios: The first ANN is

[2]Pretrained weights are available from https://nervanasystems.github.io/distiller/model_zoo.html

[3]Inspection of the parameter distribution revealed weight and bias values several orders of magnitude larger than the standard deviation. Previous work [7] indicates that such outliers can cause erroneous spikes at the onset of the simulation while spike rates are still ramping up, which delays convergence to the desired accuracy. Removing these outliers by training the model with hard constraints or regularization would likely improve the rate-based conversion but was not within the scope of the present work.

[4]This advantage would be evened out in zero-skipping hardware [22] or algorithms [23].

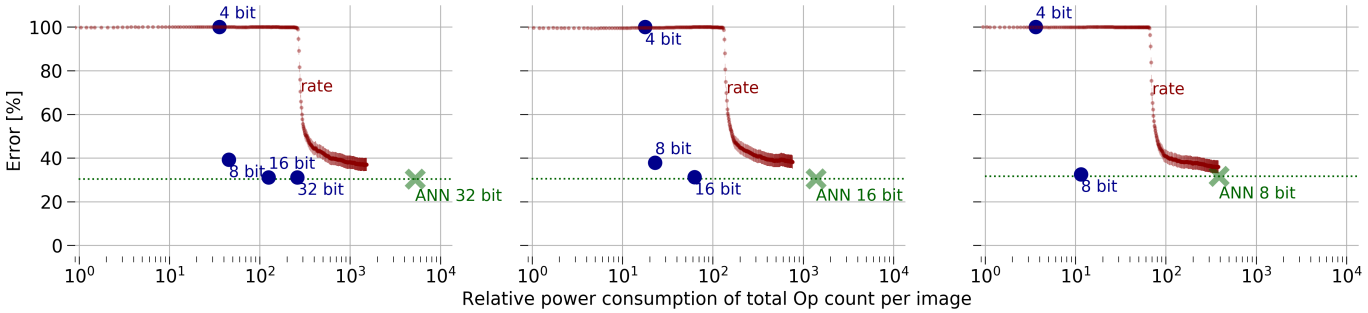


Fig. 3: Classification error versus relative power consumption of the original ANN and the converted SNNs tested on ImageNet. Each of the three panels represents the same conversion experiment but starting from an ANN trained with 32 (left), 16 (center), and 8 bit activations (right).

trained with 32-bit activations, the second with 16-bit, the third with 8-bit, using the Distiller framework [25] for quantization-aware training. The bitwidth of the weights is fixed to 8 bit in all models to match the lowest bitwidth considered for the activations. Starting with these three ANNs, we again perform the conversion to rate- and pattern-encoded SNNs and measure the tradeoff curves (Fig. 3). When converted from the 16 bit ANN, the TPC model achieves a relative reduction of power consumption up to $42\times$ with less than 1% increased error (c. f. Table II).

From the three panels in Fig. 3, we observe the following trend: As one moves from 32 bits (left plot) to 8 bits (right plot), all data points shift towards the origin, i. e. to the desired regime of lower error rate at lower power consumption. To dissect this result in more detail, consider first the error rate. In case of the ANN, quantization-aware training is able to maintain the accuracy of the 32-bit model even when going to 16 and 8 bit. In case of the rate-coded SNN, training with reduced precision appears to make the model more robust to approximation errors that are bound to occur in a network of this size. In case of the pattern-coded model, training the ANN at low precision prepares the converted SNN to handle truncation of LSBs when reducing the number of time steps during inference.

In terms of power consumption, we expect a shift to the left for two reasons: (1) The shorter bitwidth of operands results in a lower cost per operation. (2) Reduced precision is often accompanied by an increase in sparsity, which would reduce the total number of operations. However, the latter did not play a role here because we found that the sparsity did not change significantly.

While this power analysis enables a comparison of ANNs and SNNs purely based on the cost of their elementary operations, it does not take into account some important aspects of a potential implementation, for instance the cost associated with memory traffic. Reading a bit string of weights from external Dynamic Random Access Memory (DRAM) costs as much as 21 pJ per bit in an LPDDR3 memory model [26], seven times as much as a 32 bit multiplication (3.1 pJ according to [27]). In rate-coded stateful SNNs, such fetches are more frequent and less predictable than in ANNs. Because TPC-coded networks are stateless and evaluated in a layerwise fashion, they share the reduced memory traffic of ANNs. The related hardware implementation by [28] has demonstrated about $2\times$ speedup and 57% improved energy efficiency over the DaDianNao ANN accelerator [29] using models of comparable size as the MobileNets considered here. Thus, when taking into account memory traffic, we can expect the difference in power consumption between rate- and pattern-coded SNNs to widen further.

When implemented on dedicated neuromorphic hardware [30], the power consumption of rate-coded SNNs due to off-chip memory traffic is partially alleviated by storing weights and states in area-expensive Static Random Access Memory (SRAM), which lowers the power consumption for reads by two orders of magnitude compared to DRAM. Still, the large amount of spike events in rate-coded models may cause spike congestion in chip-to-chip traffic (internal results), increasing latency during inference. This potential bottleneck would be mitigated by a finite pattern code as presented here.

IV. Discussion

We present a study of encoding schemes for conversion of ANNs to SNNs. Among the three encoding types, TPC offers a number of advantages over the rate- and latency-based encodings. For one, the number of spikes in a pattern is independent of the the magnitude of activations; in rate-coded SNNs the spike count grows linearly with the ANN activation.

Spike times in TPC are log-compressed: An activation value of $x = 0.016 \approx 2^{-6}$ is represented by a spike at $t = \log_2(x) = 6$. In the TTFS latency code considered in Sec. III-A, the spike time is proportional to the inverse activation ($t \sim 1/x$); the example above would thus require 64 time steps.

In TPC, the upper limit for the required simulation duration is given by the bit precision, i. e. at most 32 time steps, which may be orders of magnitude shorter than in a standard rate code. Theoretically, an exact spike rate representation of a 32 bit value would require 4 billion time steps; in practice, tens to thousands of steps are used. With a shorter simulation duration, we can expect the overall spike count and thereby the memory traffic to be lower than in a rate code.

TABLE II: Summary of power estimates for ImageNet (power is measured relative to the cost of one adder). The factor in parenthesis represents the reduction relative to the ANN. For each ANN, we highlight in bold the converted model with lowest power estimate and classification error within 1% of the ANN.

ANN precision:	32 bit		16 bit		8 bit	
	Error [%]	Power	Error [%]	Power	Error [%]	Power
ANN	30.41	5255	30.47	1391	31.58	386
rate-SNN	36.01	1526 (3×)	37.07	764 (2×)	34.88	381 (1×)
TPC 32 bit	30.41	259 (20×)				
TPC 16 bit	**31.13**	**125** (**42**×)	**31.28**	**63** (**22**×)		
TPC 8 bit	39.30	45 (117×)	37.84	23 (60×)	**32.59**	**11** (**35**×)

With layers being processed sequentially in TPC, only the neuron state variables of the current layer need to be kept in memory. In a rate code, neuron states for the whole network need to be stored and are accessed repeatedly in an unpredictable manner.

When using the full bitwidth ($n = 32$), TPC does not involve any approximation errors as in the rate and TTFS code. Each layer in the SNN is guaranteed to reach an accurate representation of the ANN within a finite simulation time. An error-latency trade-off can be obtained by reducing the simulation duration per layer to values $T < n$. The resulting SNN classification error will be equivalent to an ANN where the $n - T$ LSBs of the activation values are clipped. We showed in Sec. III-C that the error increase can be mitigated by quantization-aware training.

Open questions remain about the proposed pattern encoding method. Unlike the asynchronous processing paradigm of SNNs, the pattern code treats one layer at a time for n time steps before moving on to the next. Thus, the pattern code sacrifices pseudo-simultaneity, i. e. the ability of the network to produce a preliminary guess at the beginning of a simulation and then refine it as more data comes in. This issue can be partially mitigated by pipelining input samples for an efficient use of the whole network.

The encoding mechanism relies on a time threshold or clock neuron to trigger spikes. However, most other coding schemes employ a similar mechanism to trigger a global time or membrane potential reset between samples in an uncorrelated input sequence, one example being the rank-order models of communication in the visual pathway [31]. Neuroscientists have suggested visual saccades or micro-saccades as possible implementation of this mechanism [32], [33].

It is not clear how real neurons would convert a given analog value into a particular spike pattern. One possibility is to combine a population code with a rank-order or latency code: Neurons in a population may respond to a stimulus with different delays; the first spikes could constitute the spike pattern (bit sequence in our TPC), with their respective order determining their efficacy on the post-synaptic neuron (c. f. [31] for a candidate for such a rank code.). Further evidence for information coding with spike patterns has been presented in [34].

The efficiency of the pattern code is based on precise spike timing. This feature poses a potential limitation, shared by other temporal coding schemes: The network accuracy is expected to be reduced in the presence of spike timing jitter in input or hidden neurons. Related to this is the question how asynchronous input from an event sensor could be encoded within the temporal pattern scheme. One solution is to integrate events within a given time window in the first layer neurons, on which the encoding can be applied as above. The network would then have to be trained on data preprocessed in the same way.

A. Comparison Against Related Work

Table III compares our results with other conversion methods that employ a temporal code. Pattern codes like the one studied here have been developed in several variants. Similar to our work here, [37] use a binary representation of activation values to generate spike patterns in the SNN. In their encoding however, the number of spikes in the pattern grows logarithmically with the magnitude of the activation value. The method induces an accuracy loss during conversion that needs to be compensated for by training the ANN with a special objective function, non-linearity, and additional hyperparameters. Their results are limited to the MNIST task. In another variant of a pattern code [15], spikes are generated asynchronously based on a time-decaying voltage threshold. Because neurons can thus fire output spikes before having received the full input pattern, this encoding results in an accuracy drop which has to be mitigated by longer run times and a transient clamping for membrane potentials. While achieving good classification error, the spike count is higher than in our method.

The work by [36] uses variable-length bursts of spikes, where threshold and effective spike weight change dynamically. The authors evaluate different combinations of spike codes for input and hidden layer neurons and demonstrate computational savings over rate code and the pattern code by [15]. However, the burst code is computationally more costly than the TPC proposed here.

In [14], the authors develop a latency code where the neuron activation is represented by the delay of a single spike. The method by [37] is similar in that they also need only a single spike per neuron, but a large activation leads to a slower rather than faster firing time. Approximation errors are reduced by preventing neurons to fire a spike until all input spikes are received. The computational cost is increased by the presence

TABLE III: Comparison of ANN to SNN conversion methods using a rate or temporal code. Results from this paper are in bold.

Dataset	Method	Neurons	Params	Error [%] ANN	SNN	Spikes	Op
MNIST	**Rate**	**8k**	**213k**	**0.72**	**1.16**	**4k**	**1.8M**
	Rate [10]	29k	760k	0.41	0.49	120k	160M
	Latency	**8k**	**213k**	**1.04**	**1.43**	**1k**	**230k**
	Latency	**1k**	**476k**	**1.50**	**1.65**	**100**	**590**
	Latency [35]	8k	213k	0.84	0.92	-	5M
	Burst [36]	23k	-	0.75	0.80	77k	-
	Pattern [37]	48k	3M	0.59	0.59	-	2.3M
	Pattern [15]	25k	30k	0.80	0.90	1M	-
	Pattern	**8k**	**213k**	**0.72**	**1.26**	**2k**	**865k**
	Pattern	**8k**	**213k**	**0.72**	**0.75**	**5k**	**2.7M**
CIFAR-10	**Rate**	**413k**	**3M**	**8.82**	**9.40**	**5M**	**576M**
	Rate [10]	182k	9M	10.34	10.33	4M	4700M
	Rate [9]	332k	2M	8.63	8.57	-	277M
	Burst [36]	281k	34M	8.59	8.59	7M	-
	Pattern [15]	118k	2M	10.90	10.90	100M	-
	Pattern [38]	475k	0.8M	7.01	7.58	0.7M	-
	Pattern	**413k**	**3M**	**8.82**	**8.82**	**2M**	**39M**
ImageNet	**Rate**	**5M**	**4M**	**30.41**	**36.01**	**75M**	**24G**
	Rate [10]	2M	-	37.02	37.11	110M	-
	Latency [35]	0.5M	80M	42.84	43.30	-	0.4G
	Latency [35]	15M	138M	30.82	29.13	-	17G
	Pattern [38]	10M	26M	24.78	24.90	14M	-
	Pattern	**5M**	**4M**	**30.41**	**31.13**	**15M**	**3.6G**
	Pattern	**5M**	**4M**	**30.41**	**30.41**	**30M**	**7.4G**

of a clock neuron. The method requires additional ANN training to adapt to a leak parameter in the SNN neuron model.

The method by [38] approximates ANN activations with a pattern-coded SNN in a layer-wise fashion. This conversion scheme introduces three additional global parameter vectors for the spiking neuron dynamics that need to be learned during ANN training. Processing of spikes in this model requires multiplication operations with these neuron parameters. In the special case of emulating ReLU nonlinearities, the extra parameters are obtained analytically, and multiplications during inference are replaced by cheaper bit-shift operations similarly to our method. Together with the TPC method presented here, the work by [38] underlines the benefits of applying pattern codes in SNNs.

In a custom CNN accelerator design, [28] replace the bit-parallel processing of the activation-weight product by a bit-serial implementation, which is one way to integrate the spike patterns in the encoding discussed here. To make up for the increased number of clock cycles needed when going from bit-parallel to bit-serial, the parallelism inherent in CNNs is exploited. The authors report roughly $2\times$ improved performance over a then state-of-the-art accelerator [29]. The proposed accelerator is 57% more energy efficient than the baseline at a cost of 32% additional area.

In the context of quantizing feature maps, [39] investigate the benefits of choosing the bitwidth on a per-layer basis. Best results are obtained with *convex quantization*, where top and bottom layers have higher bitwidth than layers in the middle. Taking a step away from heuristic quantization rules, [40] automatically determine optimal layer-wise quantization policies using reinforcement learning with feedback from the target hardware, with a cost function that accounts for model size, latency, and energy, rather than proxy metrics like floating-point operations (Flops) and memory accesses. Such layer-wise quantization would be beneficial in our TPC scheme.

To summarize the comparison in Table III, rate-coded models appear less favorable in terms of operational cost, as expected from the use of redundant spikes. The latency code [14] leads to the lowest computational cost, but is restricted to the MNIST task. Pattern codes like the one evaluated here achieve good classification errors on the more challenging tasks while requiring a number of operations comparable to sparser latency codes.

V. Conclusion

We evaluate an encoding scheme for spike signals in SNNs based on the binary number format. The proposed method addresses several shortcomings of rate- and latency-based encoding methods of SNNs, in particular poor scalability due to approximation errors, as well as the high number of redundant spikes in rate coding. The presented method was evaluated on MNIST, CIFAR-10, and ImageNet, and improves on the classification error of rate-coded SNNs by 2% while reducing the estimated cost of operations by $35\times$.

Promising directions for future work include implementation on neuromorphic hardware [41], training of the ANN with power-2 activations, and replacing the binary code with a compressing entropy code.

Acknowledgments

The authors wish to thank Alessandro Aimar for his feedback and help in estimating the power consumption.

REFERENCES

[1] W. Bialek, F. Rieke, R. R. de Ruyter van Steveninck, and D. Warland, "Reading a Neural Code," *Science*, vol. 252, no. 5014, pp. 1854–1857, 1991.

[2] W. Maass, "Networks of Spiking Neurons: The Third Generation of Neural Network Models," *Neural Netw.*, vol. 10, no. 9, pp. 1659–1671, 1997.

[3] S. C. Liu, B. Rueckauer, E. Ceolini, A. Huber, and T. Delbruck, "Event-Driven Sensing for Efficient Perception: Vision and Audition Algorithms," *IEEE Signal Process. Mag.*, vol. 36, no. 6, pp. 29–37, 2019.

[4] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, J. Kusnitz, M. Debole, S. Esser, T. Delbruck, M. Flickner, and D. Modha, "A Low Power, Fully Event-Based Gesture Recognition System," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI, USA: IEEE, Jul. 2017, pp. 7388–7397.

[5] R. Kim, Y. Li, and T. J. Sejnowski, "Simple Framework for Constructing Functional Spiking Recurrent Neural Networks," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 116, no. 45, pp. 22 811–22 820, 2019.

[6] S. B. Shrestha and G. Orchard, "SLAYER: Spike Layer Error Reassignment in Time," in *Advances in Neural Information Processing Systems*. Montreal, Canada: Curran Associates, Inc., 2018, pp. 1412–1421.

[7] 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," *Front. Neurosci.*, vol. 11, no. December, pp. 1–12, 2017.

[8] D. Neil, M. Pfeiffer, and S. C. Liu, "Learning to be Efficient: Algorithms for Training Low-Latency, Low-Compute Deep Spiking Neural Networks," in *Proceedings of the ACM Symposium on Applied Computing*, Pisa, Italy, 2016, pp. 293–298.

[9] M. Sorbaro, Q. Liu, M. Bortone, and S. Sheik, "Optimizing the Energy Consumption of Spiking Neural Networks for Neuromorphic Applications," *Front. Neurosci.*, vol. 14, p. 662, 2020.

[10] D. Zambrano, R. Nusselder, H. S. Scholte, and S. Bohte, "Efficient Computation in Adaptive Artificial Spiking Neural Networks," *Front. Neurosci.*, vol. 12, no. January, pp. 1–11, 2019.

[11] W. Maass, "Noisy Spiking Neurons with Temporal Coding have more Computational Power than Sigmoidal Neurons," in *Advances in Neural Information Processing Systems*, vol. 9, 1997, pp. 211–217.

[12] S. M. Bohte, "The Evidence for Neural Information Processing with Precise Spike-times: A Survey," *Nat. Comput.*, vol. 3, no. 2, pp. 195–206, 2004.

[13] H. Mostafa, B. U. Pedroni, S. Sheik, and G. Cauwenberghs, "Fast Classification Using Sparsely Active Spiking Networks," in *Proceedings - IEEE International Symposium on Circuits and Systems*. Baltimore, MD, USA: IEEE, 2017, p. 1.

[14] B. Rueckauer and S. C. Liu, "Conversion of Analog to Spiking Neural Networks using Sparse Temporal Coding," in *Proceedings - IEEE International Symposium on Circuits and Systems*. Florence, Italy: IEEE, 2018, pp. 8–12.

[15] J. Kim, H. Kim, S. Huh, J. Lee, and K. Choi, "Deep Neural Networks With Weighted Spikes," *Neurocomputing*, vol. 311, pp. 373–386, 2018.

[16] Y. Choukroun, E. Kravchik, F. Yang, and P. Kisilev, "Low-bit Quantization of Neural Networks for Efficient Inference," in *Proceedings - 2019 International Conference on Computer Vision Workshop, ICCVW 2019*, 2019, pp. 3009–3018.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," in *Proceedings of the IEEE*, vol. 86. IEEE, 1998, pp. 2278–2323.

[18] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *CoRR*, vol. abs/1704.04861, p. 1, 2017.

[19] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, 2014.

[20] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," University of Toronto, Tech. Rep., 2009.

[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[22] A. Aimar, H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I. A. Lungu, M. B. Milde, F. Corradi, A. Linares-Barranco, S. C. Liu, and T. Delbruck, "NullHop: A Flexible Convolutional Neural Network Accelerator Based on Sparse Representations of Feature Maps," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 3, pp. 644–656, 2019.

[23] N. Messikommer, D. Gehrig, A. Loquercio, and D. Scaramuzza, "Event-Based Asynchronous Sparse Convolutional Networks," in *Computer Vision – ECCV 2020*. Springer International Publishing, 2020, pp. 415–431.

[24] D. A. Gudovskiy and L. Rigazio, "ShiftCNN: Generalized Low-Precision Architecture for Inference of Convolutional Neural Networks," *arXiv:1706.02393*, 2017.

[25] N. Zmora, G. Jacob, L. Zlotnik, B. Elharar, and G. Novik, "Neural Network Distiller: A Python Package For DNN Compression Research," *arXiv: 1910.12232*, Oct. 2019.

[26] M. Schaffner, F. K. Gurkaynak, A. Smolic, and L. Benini, "DRAM or No-DRAM? Exploring Linear Solver Architectures for Image Domain Warping in 28 Nm CMOS," in *Proceedings -Design, Automation and Test in Europe, DATE*. Institute of Electrical and Electronics Engineers Inc., Apr. 2015, pp. 707–712.

[27] M. Horowitz, "Computing's Energy Problem (and What We Can Do About It)," in *Digest of Technical Papers - IEEE International Solid-State Circuits Conference*, vol. 57, San Francisco, CA, 2014, pp. 10–14.

[28] P. Judd, J. Albericio, and A. Moshovos, "Stripes: Bit-Serial Deep Neural Network Computing," *IEEE Comput. Archit. Lett.*, vol. 16, no. 1, pp. 80–83, 2017.

[29] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A Machine-Learning Supercomputer," in *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*. IEEE Computer Society, Jan. 2015, pp. 609–622.

[30] M. Davies, N. Srinivasa, T. H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[31] R. VanRullen and S. J. Thorpe, "Rate Coding Versus Temporal Order Coding: What the Retinal Ganglion Cells Tell the Visual Cortex," *Neural Comput.*, vol. 13, no. 6, pp. 1255–1283, 2001.

[32] R. W. Rodieck, *The First Steps in Seeing*. Oxford University Press, 1998.

[33] S. Martinez-Conde, S. L. Macknik, and D. H. Hubel, "Microsaccadic Eye Movements and Firing of Single Cells in the Striate Cortex of Macaque Monkeys," *Nat. Neurosci.*, vol. 3, no. 3, pp. 251–258, Mar. 2000.

[34] J. D. Victor, "How the Brain Uses Time to Represent and Process Visual Information," *Brain Res.*, vol. 886, no. 1-2, pp. 33–46, 2000.

[35] L. Zhang, S. Zhou, T. Zhi, Z. Du, and Y. Chen, "TDSNN: From Deep Neural Networks to Deep Spike Neural Networks with Temporal-Coding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 1319–1326.

[36] S. Park, S. Kim, H. Choe, and S. Yoon, "Fast and Efficient Information Transmission with Burst Spikes in Deep Spiking Neural Networks," in *Proceedings - Design Automation Conference*. ACM, 2019, pp. 10–15.

[37] M. Zhang, Z. Gu, N. Zheng, D. Ma, and G. Pan, "Efficient Spiking Neural Networks With Logarithmic Temporal Coding," *IEEE Access*, vol. 8, pp. 98 156–98 167, 2020.

[38] C. Stöckl and W. Maass, "Optimized Spiking Neurons Can Classify Images with High Accuracy Through Temporal Coding with Two Spikes," *Nature Machine Intelligence*, vol. 3, no. 3, pp. 230–238, Mar. 2021.

[39] E. Park, J. Ahn, and S. Yoo, "Weighted-Entropy-Based Quantization for Deep Neural Networks," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*. IEEE, Nov. 2017, pp. 7197–7205.

[40] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-aware Automated Quantization with Mixed Precision," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8604–8612.

[41] B. Rueckauer, C. Bybee, R. Goettsche, Y. Singh, J. Mishra, and A. Wild, "NxTF: An API and Compiler for Deep Spiking Neural Networks on Intel Loihi," *arXiv:2101.04261*, Jan. 2021.