Spike-Based Anytime Perception

Matthew Dutson, Yin Li, Mohit Gupta
University of Wisconsin–Madison
{dutson,yin.li,mgupta37}@wisc.org

Abstract

*In many emerging computer vision applications, it is critical to adhere to stringent latency and power constraints. The current neural network paradigm of frame-based, floating-point inference is often ill-suited to these resource-constrained applications. Spike-based perception – enabled by spiking neural networks (SNNs) – is one promising alternative. Unlike conventional neural networks (ANNs), spiking networks exhibit smooth tradeoffs between latency, power, and accuracy. SNNs are the archetype of an "anytime algorithm" whose accuracy improves smoothly over time. This property allows SNNs to adapt their computational investment in response to changing resource constraints. Unfortunately, mainstream algorithms for training SNNs (*i.e.*, those based on ANN-to-SNN conversion) tend to produce models that are inefficient in practice. To mitigate this problem, we propose a set of principled optimizations that reduce latency and power consumption by 1–2 orders of magnitude in converted SNNs. These optimizations leverage a set of novel efficiency metrics designed for anytime algorithms. We also develop a state-of-the-art simulator, SaRNN, which can simulate SNNs using commodity GPU hardware and neuromorphic platforms. We hope that the proposed optimizations, metrics, and tools will facilitate the future development of spike-based vision systems.*

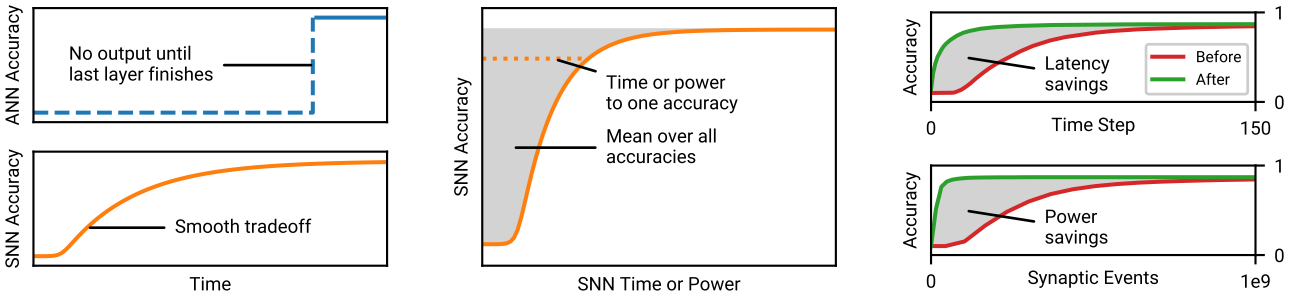
1. Introduction

Over the past decade, the computer vision community has largely embraced an "accuracy first" philosophy. For example, "state-of-the-art" usually implies achieving the highest task accuracy. However, as deep learning has continued to mature, new performance axes have begun to emerge. This trend is driven by applications (embodied perception, autonomous navigation, AR/VR) where latency and power consumption are as important as accuracy.

Conventional feed-forward neural networks (ANNs) are often ill-suited to these time- and power-constrained applications. Not only do ANNs have high computation costs – requiring billions of operations for a single inference [6] – but their computation cost also is also *inflexible*. Regardless of the runtime constraints or available computational resources, a given ANN takes a fixed amount of time and energy to return a result. In general, ANNs cannot adapt their resource investment in response to changing circumstances. In this paper, we consider *spike-based anytime perception*, an alternative approach that supports real-time modulation of the tradeoffs between accuracy, time, and power. The primary enabling component is *spiking neural networks* (SNNs), a class of brain-inspired models where neurons exchange information via temporal sequences of discrete spikes [39]. Each spike denotes an event, and information is encoded in the frequency and timing of spikes.

Vision, Fast and Slow. SNNs, through time-distributed, lightweight computation, achieve *pseudo-instantaneous information processing* [18]. Their smooth accuracy-latency tradeoff curve makes them the archetype of an "anytime algorithm" (Figure 1a) [66, 4]. Such hierarchical, *fast and slow* reasoning is thought to be a hallmark of human decision making [30], and could pave the way for a new class of dynamic control algorithms that identify optimal operating points at runtime, at the time-granularity of individual spikes. For instance, a robot navigating a dynamic environment often needs to make *fast* decisions (*e.g.*, obstacle avoidance). Such situations require short reaction times often inaccessible in ANN-based processing. Deep SNNs provide early (albeit potentially less precise) estimates. These estimates improve when given more processing time, so they can also be used for *slow*, more deliberate processes such as long-term path planning.

Minimizing Latency and Power. The discrete nature of SNN spikes results in non-differentiable network dynamics, making training SNNs a challenge. To date, the most successful approach – in terms of accuracy on challenging, large-scale datasets like ImageNet – is ANN conversion [44, 7]. Unfortunately, SNNs trained using ANN conversion often have high latency and power costs [45]. To mitigate this problem, we propose a set of optimizations for converted SNNs. These optimizations target all three phases of the model's life cycle: ANN training, SNN conversion, and SNN inference. Our methods reduce latency



(a) **Vision, Fast and Slow** (b) **Pareto Metrics** (c) **Minimizing Latency and Power**

Figure 1: **Spike-Based Anytime Perception. (a)** Unlike conventional feed-forward neural networks (ANNs), spiking neural networks (SNNs) give smooth tradeoffs between accuracy, latency, and power. This property makes them capable of "fast and slow" anytime perception. **(b)** We propose novel metrics for latency and power that measure the overall shape of the accuracy-time and accuracy-power tradeoff curves. **(c)** We leverage these metrics, along with insights into the mechanics of SNNs, to achieve significant reductions in latency and power consumption. The red and green curves show accuracy as a function of time or synaptic events, before and after applying our optimizations to a CIFAR-10 model (see section 7).

and power in converted SNNs by 1–2 orders of magnitude. We summarize our proposed optimizations below.

- **ANN Training: Sparse Representations.** Because they compute in a sparse, event-based manner, *SNNs naturally exploit weight and activation sparsity*. By altering the model representation to improve sparsity, we substantially reduce the number of synaptic events and, consequently, the model's power consumption.
- **SNN Conversion: Firing Rate Scaling.** We observe that SNN firing rates can be scaled *at the neuron level* without altering the network representation. We use an off-the-shelf optimizer to fine-tune firing rates for improved accuracy, latency, and power consumption.
- **SNN Inference: High-Entropy Initialization.** SNNs display transient dynamics where firing rates (and thus, accuracy) evolve over time before achieving steady-state behavior. We observe that *transient dynamics can be controlled by varying the model initialization*. Careful initialization can dramatically reduce latency.

Summary of Contributions. This paper presents three primary technical contributions. (1) We propose *Pareto metrics*, novel performance metrics that integrate the entire accuracy-time or accuracy-power tradeoff curve. See Figure 1b and section 4. Pareto metrics are more numerically stable than past threshold-based metrics, which allows them to be used not only for evaluation but also as tools for model optimization. (2) We use these metrics, combined with several insights into the unique dynamics of SNNs, to design a family of optimizations for improving the latency and power of ANN-converted SNNs. Rather than improving the performance of the SNN at just one operating point, these strategies simultaneously improve the entire tradeoff curve. See Figure 1c and section 5. (3) We implement SaRNN, a high-performance simulator for SNNs. SaRNN outperforms the next fastest simulator, SNN-TB [50], by an order of magnitude. This speed allows new lines of research, such as large-scale SNN optimization. SaRNN also supports mapping to neuromorphic platforms such as SpiNNaker [19, 48] through the PyNN API [14]. See section 6. We will make the SaRNN code freely available.

Scope and Limitations. This work considers the task of image classification on individual frames. An important next step is to evaluate our methods on time-varying video data and to consider other tasks (*e.g.*, object detection, semantic segmentation, and tracking).

SNNs are still a nascent technology, while ANNs and GPUs have been optimized over decades. As such, we do not attempt any watt-for-watt or second-for-second comparisons between ANNs and SNNs. Instead, our goal is to explore a promising new technological paradigm and make progress toward eventual wide-scale use. We hope that the proposed methods, coupled with ongoing advances in neuromorphic architectures[1] and the emergence of applications requiring perception with strict latency, computational and power budgets, will lower the entry barrier into SNN research and spur further research on spike-based perception.

2. Related Work

Efficient Architectures and Pruning. Many ANN architectures are designed with efficiency in mind. These include MobiletNet, SqueezeNet, and ShuffleNet [24, 27, 65] for classification, and SSD, YOLO, and RetinaNet [38, 47, 37] for object detection. Other algorithms apply pruning to reduce the number of weights and arithmetic operations

[1]The last decade has seen significant developments in the design and manufacture of neuromorphic platforms. Recent examples include TrueNorth (IBM) [2], Loihi (Intel) [13], SpiNNaker (The University of Manchester) [19], and BrainScaleS (Heidelberg University) [53].

[33, 22, 21, 35]. In general, these approaches are complementary to ours; an SNN derived from an efficient ANN architecture will inherit the efficiency of the ANN. For example, we use the MobileNet [24] architecture in section 7, which further lowers the power consumption of the optimized SNN by reducing the number of model symapses.

Network Quantization. Binary neural networks, in which weights or activations take binary values, have some similarities to SNNs [11, 46]. There are also models which use ternary, integer, and reduced-precision floating-point arithmetic [57, 26]. Like SNNs, these models may require custom-tailored hardware for maximum efficiency.

Adaptive Inference. There are several methods which, like SNNs, adaptively modify the inference cost in response to resource constraints or perceived difficulty. Some use variations of an adaptive computation graph [58, 61, 56, 62]; others adaptively resize the inputs and feature maps [9, 63]. Unlike these methods, SNNs provide adaptive inference *for free*. SNNs do not require the designer to modify the underlying network architecture or make a specific set of assumptions about end-user constraints.

Spiking Neural Networks. Because a spike is a non-differentiable impulse, standard backpropagation [51] cannot be used to train SNNs. Many types of SNN training algorithms have been proposed to circumvent this problem. Some methods perform backpropagation in the spiking domain – by using differentiable proxies [5, 34, 59, 41, 60] or by modeling neuron state dependencies [64]. Other algorithms emulate the local, unsupervised learning that occurs in biological brains [40, 3, 55, 42].

We adopt the *ANN conversion* method for SNN training [45, 44, 7, 16, 43, 25, 50, 54]. The essential idea is to train an ANN, then copy its weights to an SNN with the same architecture. ANN conversion has several key advantages. Compared to other training methods, it achieves higher accuracy on challenging, large-scale datasets like ImageNet [52, 45]. It can be applied to modern, deep architectures like ResNet [23, 54]. Its neuron model (NL-IAF, section 3) is computationally simple and does not have hand-tuned parameters (*e.g.*, a refractory period).

Despite these advantages, SNNs trained via ANN conversion often require many timesteps and spikes to converge to an accurate solution. Several previous papers seek to address this problem. Rueckauer *et al*. [50] propose several useful optimizations, including improvements to the weight normalization algorithm and post-spike reset mechanism. Han *et al*. [20] propose a similar post-spike reset mechanism to [50]. We consider these approaches as a baseline; they correspond to the "before" condition in our experiments (*e.g.*, Table 1). Sengupta *et al*. [54] and Deng and Gu [15] propose weight scaling algorithms (Spike-Norm and threshold balancing, respectively). We do not perform a direct comparison against these methods because they are *complementary* to ours. The methods we propose in subsection 5.1 and subsection 5.3 are independent of the weight scaling algorithm. The optimization of subsection 5.2 can be initialized with any starting point (*e.g.* the result of Spike-Norm) and will always perform *at least as well* as that starting point. See Figure 3b.

3. Background

We adopt the ANN conversion approach of [50]. There are three steps to this process. (1) Absorb any batch normalization transforms into the model weights. (2) Apply data-based weight normalization. This involves passing training data through the ANN and, at each layer, scaling the weights and biases so that most activations lie in the range $[0, 1]$ (the range of possible firing rates). See subsection 5.2 for further details on this step. (3) Copy ANN layers to the SNN, discarding any ReLU activations. We refer the reader to [50] for a more detailed description of the above steps. Note that our treatment of the output layer is slightly different from [50]; see the supplementary material for details.

NL-IAF Neuron Model. ANN neurons with ReLU activations are converted to SNN neurons which obey the non-leaky integrate and fire (NL-IAF) model. NL-IAF neurons operate in discrete time steps. Let t be the current timestep, j be the neuron index, and $V_{j,t}$ be the neuron membrane potential. The neuron fires when the potential exceeds a threshold (which we assume to be 1). Let the variable $\Theta_{j,t}$ indicate whether neuron j fires a spike at time t. Then,

$$\Theta_{j,t} = \begin{cases} 1 & \text{if } V_{j,t} \geq 1 \\ 0 & \text{else,} \end{cases} \tag{1}$$

After a spike, the membrane potential is reset by subtraction [50]. So, if $I_{j,t}$ is the incoming current at time t,

$$V_{j,t} = V_{j,t-1} - \Theta_{j,t-1} + I_{j,t}. \tag{2}$$

Let $\boldsymbol{s}_{j,t}$ be a binary vector (1 for a spike, 0 otherwise) of incoming spikes at time t, containing one entry for each synapse. Let $\boldsymbol{w}_j$ be a vector of synaptic weights and let b_j be the neuron bias. Then

$$I_{j,t} = \boldsymbol{s}_{j,t} \cdot \boldsymbol{w}_j + b_j. \tag{3}$$

In converted SNNs, NL-IAF neurons encode their activations as an average firing rate. For example, a neuron that fires on four out of ten timesteps has activation 0.4.

4. Pareto Metrics

In this section, we define novel metrics for latency and power consumption. Consider an anytime algorithm with a smooth accuracy versus time curve as shown in Figure 1b. One way to measure latency is as the number of time steps

required for accuracy to cross a threshold [43]. However, there are two problems with a threshold-based metric. First, it is sensitive to the choice of threshold – due to the asymptotic nature of accuracy convergence. For example, a network with asymptotic accuracy $80\,\%$ may take only 100 time steps to reach $79\,\%$ accuracy, but another 1000 steps to reach $79.5\,\%$. Second, a threshold metric does not vary smoothly with the network dynamics. A model can take 8 or 9 steps to cross a threshold, but not 8.5. This makes it difficult to use a threshold metric to fine-tune the network parameters because the metric value may not change in response to small parameter perturbations.

Instead of operating on a single predetermined accuracy, we propose metrics that compute the average time and power over *all possible accuracies*. Consider a monotonically increasing accuracy-time curve. Now, consider a horizontal line between the y-axis and one point along this curve (Figure 1b). The length of this line gives the time required to reach one specific accuracy. By extension, the *area* between the y-axis and the curve gives the average time (up to a constant) required to reach all accuracies. Note that we can equivalently (and more conveniently) compute the area left of the curve as the area *above* the curve (assuming an upper boundary equal to the maximum accuracy). With this, we propose the following metrics for latency and power:

Pareto Latency. Let $a_1, a_2, \ldots, a_T$ be the instantaneous accuracy over T inference time steps, and let a_{max} be the asymptotic accuracy. We define *Pareto latency* as

$$P_l = \sum_{t=1}^{T} (a_{\text{max}} - a_t). \tag{4}$$

Pareto Power. We define an analogous metric for power consumption. Instead of considering accuracy versus time, this metric computes the area above an accuracy versus synaptic events curve. Note that there is a subtle distinction between a spike and a synaptic event; when a neuron spikes, one synaptic event is triggered for each of its outgoing synapses. Let $e_1, e_2, \ldots, e_T$ be the number of synaptic events at each time step. We define *Pareto power* as

$$P_p = \sum_{t=1}^{T} e_t \, (a_{\text{max}} - a_t). \tag{5}$$

Pareto latency and power are defined such that they can be applied to a broad class of anytime algorithms – beyond just ANN-converted SNNs. For example, they could be used with SNNs with alternate neuron models or training strategies. Although we define them in terms of accuracy, these metrics could easily be reformulated using any performance measure (*e.g.*, the mAP metric for object detection). Note that, in the specific case of ANN-converted SNNs, we replace a_{max} (the asymptotic accuracy) with a_{ANN}, the accuracy of the ANN before conversion.

5. Minimizing Latency and Power

In this section, we develop methods for improving Pareto latency and power while maintaining high accuracy. There are three stages in the life cycle of a converted SNN: (1) ANN training, (2) SNN conversion, and (3) SNN inference. Past work on converted SNNs has focused mostly on stage 2 (specifically, weight scaling). We advocate a more holistic approach that considers all three phases. We show that there is significant untapped potential in optimizing ANN training and SNN inference dynamics.

5.1. ANN Training: Sparse Representations

Because SNNs compute in an event-based manner, sparsity in an SNN translates directly to reduced power consumption. We consider two sources of sparsity: activations and weights. If there is high activation sparsity, more neurons have zero activations and do not spike. If there is high weight sparsity, each spike triggers fewer synaptic events.

Conversion involves scaling the weights and activations of the network (subsection 5.2). While this does change the *values* of the weights and activations, it does not change their *sparsity* (zero multiplied by any scalar is zero). Therefore, *conversion preserves the sparsity of the ANN*. We leverage this observation by fine-tuning the ANN with activation and weight sparsity penalties L_a and L_w.

Activation Sparsity. We use batch normalization (BN), a common component in modern ANNs, in our formulation of L_a [28]. The BN transform is defined such that its output distribution D has mean β and variance γ^2, where β and γ are learned parameters. Because β controls the mean activation, decreasing its value increases the number of negative activations. Assuming the BN is followed by a ReLU function (a common design pattern), these negative activations are then truncated to zero (Figure 2a). During conversion, the linear batch normalization transform is absorbed into the weights of the preceding layer [50]. As a result, each weighted–BN–ReLU layer group collapses to a single NL-IAF layer in the SNN. Therefore, sparsity in the ReLU output translates to sparsity in the NL-IAF output.

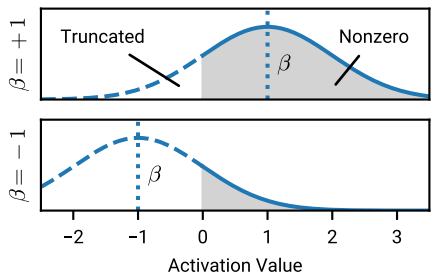
Assume there is a ReLU activation after the BN layer. The expected fraction of activations *not* truncated to zero is

$$\int_0^\infty D(x; \beta, \gamma)\, dx. \tag{6}$$

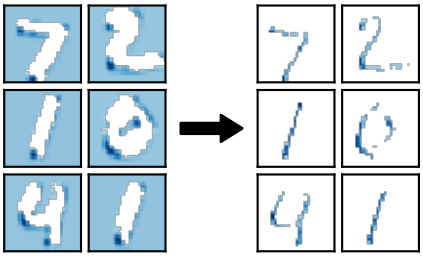
Fixing $\gamma = 1$ (which does not reduce representational power with ReLU activations) and assuming D is a normal distribution, this integral simplifies to

$$\frac{1}{2}\left(\text{erf}\left(\frac{\beta}{\sqrt{2}}\right) + 1\right). \tag{7}$$

As this quantity decreases, sparsity increases.



(a) **Batch Normalization for Activation Sparsity.** Decreasing the value of β leads to improved sparsity.



(b) **Sparse Feature Maps.** Experimental sparsity improvements on a convolutional MNIST model.

Figure 2: **ANN Training: Sparse Representations**

Let n_k be the number of neurons corresponding to the k^{th} model β value. In the BN after a convolution, n_k is the number of pixels per channel; in the BN after a fully-connected layer, n_k is 1. We define

$$L_a = \frac{1}{\sum_k n_k} \sum_k \frac{n_k}{2} \left(\text{erf}\left(\frac{\beta_k}{\sqrt{2}} \right) + 1 \right). \tag{8}$$

L_a is proportional to the expected number of nonzero activations across the entire network. By adding a training penalty on L_a, we encourage activation sparsity. To our knowledge, this formulation of L_a using batch normalization represents a new general method for improving activation sparsity, and likely has applications beyond SNNs.

Weight Sparsity. For L_w we use an L1 (lasso) penalty to encourage weight sparsity. Some weights (*i.e.*, those in a convolution) are repeated over many neurons, and we scale the weight penalty terms accordingly. Let m_k be the number of neurons corresponding to the k^{th} weight w_k. In a convolution, m_k is the number of pixels per channel; in a fully-connected layer, m_k is 1. We define

$$L_w = \frac{1}{\sum_k m_k} \sum_k m_k |w_k|. \tag{9}$$

After training, we remove all weights with magnitude less than some $\epsilon \approx 10^{-4}$. Note that this strategy is similar to conventional weight pruning; the primary difference is the use of the scaling terms m_k.

Sparsity Fine-Tuning. We apply the above loss terms during an ANN fine-tuning phase *before* SNN conversion. We first train an ANN normally or load pre-trained weights. We then train for additional epochs with a modified loss. Let L_{ANN} be the loss used during initial ANN training (*e.g.*, a cross-entropy loss). We define the modified loss as

$$L'_{\text{ANN}} = L_{\text{ANN}} + \lambda_a L_a + \lambda_w L_w. \tag{10}$$

Minimizing this loss encourages an ANN representation with fewer nonzero activations and weights. λ_a and λ_s are tradeoff-scaling parameters; see the supplementary material for details on how we chose their values.

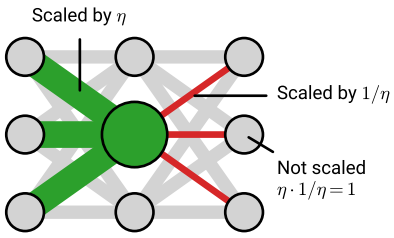
5.2. SNN Conversion: Firing Rate Scaling

A spiking NL-IAF neuron can only fire once per time step, so its maximum activation is 1. In contrast, a ReLU neuron can have activation $[0, \infty)$. Data-based normalization [16, 50] – applied during ANN to SNN conversion – re-scales the ANN weights so that most activations lie in the range $[0, 1]$. Careful scaling is vital for achieving accurate and efficient models. Neurons with low firing rates may take many time steps to send their first spike, while neurons with firing rates above 1 saturate and encode an incorrect value. More complex and subtle behaviors arise in networks of many interacting neurons.

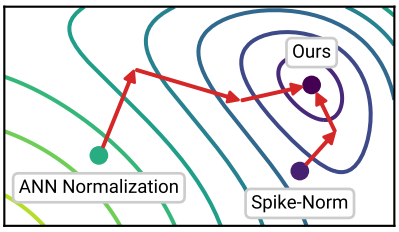
Neuron-Level Scaling. Existing methods for data-based normalization jointly scale all activations in a layer [16, 50, 54]. They assume that scaling at a finer granularity would destroy the original ANN representation. We show that this assumption is questionable, that indeed it is possible to *scale activations for individual neurons* without altering the ANN representation. Neuron-level scaling is more flexible and is especially useful when there are significant variations in activation distributions within a layer. In this case, layer-level scaling will "over-suppress" neurons with large mean activations and "under-excite" neurons with small mean activations. Neuron-level scaling resolves this issue by scaling each neuron based on its specific activation distribution.

To scale the activation of neuron j by a factor η_j, we multiply its incoming weights and bias by η_j. We then *isolate the change* by multiplying the neuron's outgoing weights by $1/\eta_j$. See Figure 3a. The net result of this isolated scaling is that only the activation of neuron j changes. In this way, we preserve the ANN representation and output.

There are two important details to consider. First, because of weight sharing, "neuron-level scaling" can only be done at the channel level in convolutions. Because different pixels in the same channel have similar activation distributions, this is not a significant problem. Second, the $1/\eta_j$ isolation only works if layer $i + 1$ has adjustable weights. This requirement is not met in unweighted pooling layers



(a) **Neuron-Level Scaling.** It is possible to scale the firing rates of an individual neuron by making targeted adjustments to its incoming and outgoing weights.



(b) **Formal Optimization.** Formal optimization over $H = \{\eta_j\}$ gives a value of L_{SNN} at least as low as existing heuristic methods [50, 54].

Figure 3: **SNN Conversion: Firing Rate Scaling**

and the output layer. We solve this problem for average pooling layers by replacing them with equivalent depthwise convolutions. We use layer-level scaling before max pooling and the model output.

Formal Optimization. Existing methods for weight normalization [16, 50, 54, 15] are, in some sense, heuristics for estimating the optimal activation scaling factors. Denote the set of all factors in a network $H = \{\eta_j\}$ as the *scaling set*. Instead of using heuristics, we *formally optimize the loss function over the scaling set*. This approach allows us to explicitly tune tradeoffs between accuracy, latency, and power by adjusting the loss function. In contrast, heuristic methods can only tune these tradeoffs indirectly by adjusting low-level parameters of the scaling algorithm. Furthermore, assuming we initialize with the scaling set produced by some heuristic method, formal optimization is guaranteed to perform *at least as well* as the heuristic (Figure 3b).

We optimize the following loss function:

$$L_{\text{SNN}}(H) = \lambda_e M_e(H) + \lambda_l P_l(H) + \lambda_p P_p(H). \quad (11)$$

P_l and P_p are the Pareto latency and power. M_e is the minimum of the SNN error $1 - a_t$ over all time steps (like P_l and P_p, it could easily be formulated using a metric other than accuracy). The λ are tradeoff parameters for accuracy (λ_e), latency (λ_l), and power (λ_p). See the supplementary material for details on how we select the λ values. We evaluate L_{SNN} by simulating the SNN over the training dataset, thereby capturing all the subtle dynamics of the model.

Because spikes are not differentiable, we use an off-the-shelf derivative-free optimizer (DFO) to minimize L_{SNN}. DFO algorithms estimate the shape of the function by querying objective function values, and do not require computing gradients. We use the Subplex DFO algorithm [49] as implemented in the NLopt package [29]. We achieve the best results with a three-phase optimization: (1) model-level scaling with one global η, (2) layer-level scaling with one η per layer, (3) and neuron-level scaling with one η per neuron. Subsequent phases requires more optimizer iterations due to the increasing dimensionality of the search space.

5.3. SNN Inference: High-Entropy Initialization

In this section, we consider improvements to SNN inference. Specifically, we examine the choice of initial neuron membrane potential V_0. The equation for $V_{j,t}$ (Equation 2) is defined recursively in terms of $V_{j,t-1}$, but it does not specify a starting value $V_{j,0}$. Unlike firing rate scaling (subsection 5.2), there is little existing work on neuron initialization, with most authors simply setting $V_{j,0} = 0$. Through both theoretical analysis and experiments, we show that this is a sub-optimal choice.

Theoretical Analysis. We now show that *a zero initialization represents a state of artificially low entropy*. Consider the subset of neurons with nonzero firing rates during an inference. Define $\bar{V}_t$ as the mean of $V_{j,t}$ over this subset. For $t > 0$, firing neurons will have $V_{j,t}$ throughout the range $[0, 1]$, and therefore for $\bar{V}_t > 0$. In other words, it is unlikely that a model would spontaneously arrive in a state where $\bar{V}_t \approx 0$. Yet, $\bar{V}_t = 0$ is the state we force the model into when initializing globally with $V_{j,0} = 0$.

What, then, is a more natural value for $V_{j,0}$? Consider a neuron j with constant incoming current $I_j \in (0, 1)$ and $V_{j,0} \in [0, 1]$. Using the NL-IAF equations, we can derive

$$V_{j,t} = V_{j,0} + I_j t - \lfloor V_{j,0} + I_j t \rfloor. \quad (12)$$

We claim that the most natural initialization is the mean of this $V_{j,t}$ over an infinite number of time steps and all possible $I_j \in (0, 1)$. That is, the value of the quantity

$$\int_0^1 \lim_{T \to \infty} \left(\frac{1}{T} \sum_{t=0}^{T-1} (V_{j,0} + I_j t - \lfloor V_{j,0} + I_j t \rfloor) \right) dI_j. \quad (13)$$

Numerical evaluation shows that this expression has a value of 0.5 for all values $V_{j,0} \in [0, 1]$.

We refer to an initialization with $V_{j,0} = 0.5$ as *high-entropy initialization*. In Figure 4 we show experimentally that a change of initialization from 0 to 0.5 leads to a dramatic reduction in latency. Neurons initialized with $V_{j,0} = 0.5$ converge more quickly to their steady-state firing rates. This speedup has a cascading effect on latency as

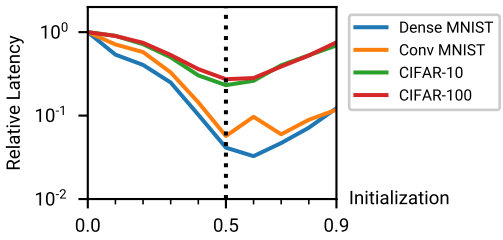


Figure 4: **High-Entropy Initialization.** A high-entropy initialization of $V_0 = 0.5$ gives much lower latency than the naive initialization $V_0 = 0$. See section 7 for model details. The y-axis units are "Pareto latency relative to a model with $V_0 = 0$." Results are averages over the entire test dataset.

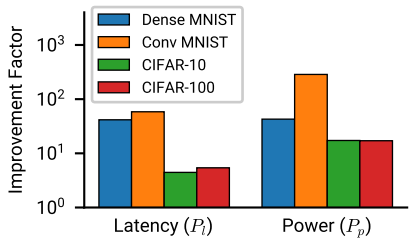


Figure 5: **Results on MNIST and CIFAR.** The combined effect of applying our methods to several models. The y-axis units are "factor of improvement in P_l and P_p relative to an unoptimized model." See supplementary for detailed results. Results are averages over the entire test dataset.

we move deeper in the network because layer $i + 1$ cannot fire accurately until the output of layer i has mostly converged to the correct value.

Optimization of Initialization. So far, we have assumed a single $V_{j,0}$ for all neurons j. However, it is possible to vary the initialization over individual neurons. Varied initialization may be beneficial, for example, if one layer has a large bias that causes it to fire prematurely. Motivated by this, we *extend the optimization of subsection 5.2 to include initialization*. Define V_0 (without subscript j) as the set $\{V_{j,0}\}$. We use the same loss function L_{SNN}, but express it as a function a function of V_0, $L_{\text{SNN}}(H, V_0)$. At each of the three optimization phases, we double the dimensionality of the search space, adding one value of $V_{j,0}$ for each value of η_j. We initialize with $V_{j,0} = 0.5$ for all j.

6. SNN Simulation

Without a high-performance SNN simulator, the formal optimization described in subsection 5.2 and subsection 5.3 takes an impractical amount of time. We found that existing SNN simulators – both general-purpose simulators (NEURON [8]) and those specialized for converted ANNs (SNN-TB [50]) – were not up to the task. We implemented a new simulator, SaRNN, which far outperforms existing simulators on converted ANNs. SaRNN (**S**piking **a**s **R**ecurrent **N**eural **N**etwork) implements SNNs as TensorFlow RNNs [1]. TensorFlow compiles the RNN into a static computation graph, preventing inefficient calls to the Python interpreter. We fuse both spiking simulation and operation counting (for evaluating P_p) into the RNN update loop. See the supplementary material for more details.

MPI [10] (Message Passing Interface) is a widely-used standard for distributed computing. SaRNN supports MPI through the mpi4py Python package [12], allowing simulation and optimization to be divided between many compute nodes. SaRNN also allows simulation with other backends (including SpiNNaker [19, 48]) through the PyNN API [14]. Source code will be made publicly available.

7. Experiments

Results on MNIST and CIFAR. We train four ANNs: dense MNIST [32], convolutional MNIST, and convolutional CIFAR-10/100 [31]. See the supplementary material for details on architectures and training. In general, we prefer simple, generalizable training (*e.g.*, SGD and minimal data augmentation) over maximum accuracy. Even so, our models achieve competitive accuracy: $98.36\,\%$ and $99.55\,\%$ on MNIST, $89.39\,\%$ on CIFAR-10, and $85.15\,\%$/$85.15\,\%$ top-1/top-5 on CIFAR-100.

After initial ANN training we perform sparsity fine-tuning, then convert the models to SNNs and optimize $L_{\text{SNN}}(H, V_0)$. We run the three optimization phases (global, layer, and neuron) for 100, 1000, and 10 000 iterations, respectively. We then compare the final models against unmodified baselines (models without the optimizations of section 5). We simulate the MNIST models for 50 time steps and the CIFAR models for 1000 time steps. Figure 5 shows the improvement factors for P_l and P_p. Latency is reduced by approximately one order of magnitude, and power is reduced by approximately two orders of magnitude. Although we only explicitly optimize P_l and P_p, we also see substantial improvements on traditional threshold-based metrics; see supplementary for these results.

Addition and Ablation. To understand the relative contributions of the three techniques in section 5, we individually add (to the baseline) and ablate (from the final model) the three techniques in section 5. We use the convolutional MNIST model. See Figure 6 for results. We make two observations. First, on its own, sparsity fine-tuning (component 1) decreases power but increases latency. However, the drop in latency vanishes when sparsity fine-tuning is combined with optimization over $L_{\text{SNN}}(H, V_0)$ (components 2 and 3). This indicates a complementary relationship, with optimization mitigating any potential downsides of sparsity fine-tuning. Second, although component 2 (firing rate scaling) individually causes an improvement over the baseline, it has little effect when used along with components 1 and

Table 1: **Results on ImageNet.** We observe significant improvements in both P_p and peak accuracy.

Metric	Before	After
$\max\{a_t\}$	$41.59\,\%$	$49.68\,\%$
P_l	1568	416
P_p	1.02×10^{10}	1.38×10^{9}

Table 2: **Results on SpiNNaker.** Due to limitatins of the PyNN API, P_p is expressed here in terms of spiking events.

Metric	Before	After
$\max\{a_t\}$	$84.00\,\%$	$97.80\,\%$
P_l	31.7	9.54
P_p	4.17×10^{3}	1.01×10^{3}

3. We suspect that optimized initialization (component 3) is "picking up the slack" when component 2 is removed.

Results on ImageNet. To show the scalability of the proposed techniques and SaRNN to modern, large-scale datasets, we apply our methods to an ImageNet [52] model. We use the MobileNet architecture due to its high efficiency and relatively simple architecture [24]. We use an input resolution of 160×160. See the supplementary material for training details – as before, we aim for reproducibility and generalizability over maximum accuracy. Our ANN achieves $49.09\,\%$/$74.92\,\%$ top-1/top-5 accuracy.

Due to the larger image and model sizes, SNN simulation on ImageNet takes considerably longer than on MNIST or CIFAR. To reduce optimization time, we eliminate the neuron-level optimization phase and shorten the layer-level phase from 1000 to 500 iterations. Instead of simulating the entire training dataset on each iteration, we use a 1000-item subset. With these changes, SNN optimization takes only four days (less time than is required for ANN training).

See Table 1 for SNN evaluation results. We simulate the model for 5000 time steps. Like [54], we report results on the entire ImageNet validation set. Note that the peak accuracy of the final SNN is *higher* than the original ANN.

Results on SpiNNaker. To show that our results generalize beyond simulation to real neuromorphic platforms, we evaluate our methods on SpiNNaker hardware [19, 48]. The SpiNNaker neuron model has three differences from NL-IAF. (1) Neurons reset to zero after spiking instead of by subtraction. (2) Spikes take one time step to traverse a synapse. (3) Neurons have a refractory period of one time step. We implement these behaviors in SaRNN and optimize under these dynamics, thereby ensuring that our improvements generalize to SpiNNaker.

Table 2 shows results for the dense MNIST model. We simulate the model for 100 time steps. Because of the added delays for synaptic transmission, it takes some minimum time (about six time steps) for a signal to propagate from the input to the output. We suspect that this is the reason for the smaller improvement in P_l compared to the results in Figure 5. The low "before" accuracy is caused by the zero reset and refratory period described in the previous paragraph. Our optimization strategies recover an accuracy close to that of the original ANN.

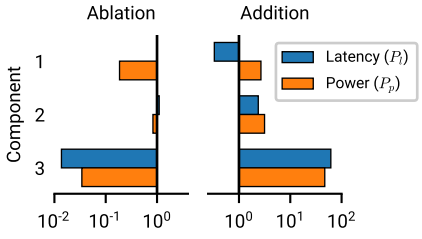


Figure 6: **Ablation and Addition.** The effect of each proposed optimization on the convolutional MNIST model. Components 1, 2, and 3 correspond to subsection 5.1, subsection 5.2, and subsection 5.3, respectively.

8. Discussion

Limitations and Future Work. This paper considers the task of image classification. However, our metrics, optimizations, and simulator are compatible with other tasks given minor modifications (*i.e.*, replacing accuracy in P_l and P_p with another metric). In the future, we hope to show results on other tasks like object detection.

The experiments in this paper involve static, single-frame inputs. However, one promising application of SNNs is in event-based processing (with event-based [36] or single-photon [17] sensors). SNNs allow inference to be simultaneous with event-based image data collection. In this case, there are two time scales to consider: one for input events and another for SNN updates. We would likely need *two* latency metrics, one for *input latency* and another for *computational latency*. The methods in this paper reduce computational latency but do not consider input latency.

Neuromorphic Hardware. Limiting the SaRNN simulator to a single, computationally efficient model type helps it achieve high performance. We believe designers should consider a similar approach for neuromorphic architectures. Current neuromorphic platforms like Intel's Loihi support a broad range of model types and biologically realistic training mechanisms [13]. While this may be useful for researchers, it results in high transistor and power costs. We hope to eventually see the emergence of simpler, more efficient architectures specialized for NL-IAF models.

Acknowledgments. This research was supported by NSF CAREER Award 1943149.

References

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, Brian Taba, Michael Beakes, Bernard Brezzo, Jente B. Kuang, Rajit Manohar, William P. Risk, Bryan Jackson, and Dharmendra S. Modha. TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10):1537–1557, Oct. 2015.

[3] Guo-qiang Bi and Mu-ming Poo. Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of Neuroscience*, 18(24):10464–10472, 1998.

[4] Mark Boddy and Thomas Dean. Decision-theoretic deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67(2):245–286, 1994.

[5] Sander M. Bohte, Joost N. Kok, and Han La Poutré. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48(1):17–37, Oct. 2002.

[6] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *arXiv:1605.07678 [cs]*, Apr. 2017.

[7] Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113(1):54–66, May 2015.

[8] Nicholas T. Carnevale and Michael L. Hines. *The NEURON Book*. Cambridge University Press, Jan. 2006.

[9] Ting-Wu Chin, Ruizhou Ding, and Diana Marculescu. AdaScale: Towards real-time video object detection using adaptive scaling. *arXiv:1902.02910 [cs]*, Feb. 2019.

[10] Lyndon Clarke, Ian Glendinning, and Rolf Hempel. The MPI message passing interface standard. In Karsten M. Decker and René M. Rehmann, editors, *Programming Environments for Massively Parallel Distributed Systems*, Monte Verità, pages 213–218, Basel, 1994. Birkhäuser.

[11] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. BinaryConnect: Training deep neural networks with binary weights during propagations. *Advances in Neural Information Processing Systems*, 28:3123–3131, 2015.

[12] Lisandro Dalcín, Rodrigo Paz, and Mario Storti. MPI for Python. *Journal of Parallel and Distributed Computing*, 65(9):1108–1115, Sept. 2005.

[13] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sir Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yunyun Liao, Chit-Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steven McCoy, Arnab Paul, Jonathan Tse, Gurugahanathan Venkataramanan, Yi-Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, Jan. 2018.

[14] Andrew P. Davison, Daniel Brüderle, Jochen M. Eppler, Jens Kremkow, Eilif Muller, Dejan Pecevski, Laurent Perrinet, and Pierre Yger. PyNN: A common interface for neuronal network simulators. *Frontiers in Neuroinformatics*, 2, 2009.

[15] Shikuang Deng and Shi Gu. Optimal conversion of conventional artificial neural networks to spiking neural networks. In *International Conference on Learning Representations*, Feb. 2022.

[16] Peter U. Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2015.

[17] Neale A. W. Dutton, Istvan Gyongy, Luca Parmesan, Salvatore Gnecchi, Neil Calder, Bruce R. Rae, Sara Pellegrini, Linsay A. Grant, and Robert K. Henderson. A SPAD-based QVGA image sensor for single-photon counting and quanta imaging. *IEEE Transactions on Electron Devices*, 63(1):189–196, Jan. 2016.

[18] Clément Farabet, Rafael Paz, Jose Perez-Carrasco, Carlos Zamarreño, Alejandro Linares-Barranco, Yann LeCun, Eugenio Culurciello, Teresa Serrano-Gotarredona, and Bernabe Linares-Barranco. Comparison between frame-constrained fix-pixel-value and frame-free spiking-dynamic-pixel ConvNets for visual processing. *Frontiers in Neuroscience*, 6, 2012.

[19] Steve B. Furber, Francesco Galluppi, Steve Temple, and Luis A. Plana. The SpiNNaker project. *Proceedings of the IEEE*, 102(5):652–665, May 2014.

[20] Bing Han, Gopalakrishnan Srinivasan, and Kaushik 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.

[21] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in Neural Information Processing Systems*, 28:1135–1143, 2015.

[22] Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in Neural Information Processing Systems*, 5:164–171, 1992.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[24] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861 [cs]*, Apr. 2017.

[25] Eric Hunsberger and Chris Eliasmith. Training spiking deep networks for neuromorphic hardware. *arXiv:1611.05141 [cs]*, 2016.

[26] Kyuyeon Hwang and Wonyong Sung. Fixed-point feedforward deep neural network design using weights +1, 0, and -1. In *Workshop on Signal Processing Systems (SiPS)*, pages 1–6, Oct. 2014.

[27] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. *arXiv:1602.07360 [cs]*, Nov. 2016.

[28] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167 [cs]*, Mar. 2015.

[29] Steven G. Johnson. The NLopt nonlinear optimization package.

[30] Daniel Kahneman. *Thinking, Fast and Slow*. Macmillan, Oct. 2011.

[31] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[32] Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[33] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in Neural Information Processing Systems*, 2:598–605, 1989.

[34] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, 10, 2016.

[35] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient ConvNets. *arXiv:1608.08710 [cs]*, Mar. 2017.

[36] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. A 128x128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor. *IEEE Journal of Solid-State Circuits*, 43(2):566–576, Feb. 2008.

[37] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. In *International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[38] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot MultiBox detector. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *European Conference on Computer Vision (ECCV)*, Lecture Notes in Computer Science, pages 21–37. Springer International Publishing, 2016.

[39] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, Dec. 1997.

[40] Henry Markram, Joachim Lübke, Michael Frotscher, and Bert Sakmann. Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science*, 275(5297):213–215, Jan. 1997.

[41] Hesham Mostafa. Supervised learning based on temporal coding in spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(7):3227–3235, July 2018.

[42] Emre O. Neftci, Charles Augustine, Somnath Paul, and Georgios Detorakis. Event-driven random back-propagation: Enabling neuromorphic deep learning machines. *Frontiers in Neuroscience*, 11, 2017.

[43] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Learning to be efficient: Algorithms for training low-latency, low-compute deep spiking neural networks. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, SAC '16, pages 293–298, New York, NY, USA, Apr. 2016. Association for Computing Machinery.

[44] José Antonio Pérez-Carrasco, Bo Zhao, Carmen Serrano, Begoña Acha, Teresa Serrano-Gotarredona, Shouchun Chen, and Bernabé Linares-Barranco. Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence Processing–Application to feedforward ConvNets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(11):2706–2719, Nov. 2013.

[45] Michael Pfeiffer and Thomas Pfeil. Deep learning with spiking neurons: Opportunities and challenges. *Frontiers in Neuroscience*, 12, 2018.

[46] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet classification using binary convolutional neural networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *European Conference on Computer Vision (ECCV)*, Lecture Notes in Computer Science, pages 525–542. Springer International Publishing, 2016.

[47] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[48] Oliver Rhodes, Petruţ A. Bogdan, Christian Brenninkmeijer, Simon Davidson, Donal Fellows, Andrew Gait, David R. Lester, Mantas Mikaitis, Luis A. Plana, Andrew G. D. Rowley, Alan B. Stokes, and Steve B. Furber. sPyNNaker: A software package for running PyNN simulations on SpiNNaker. *Frontiers in Neuroscience*, 12, 2018.

[49] Thomas Harvey Rowan. *Functional Stability Analysis of Numerical Algorithms*. PhD thesis, The University of Texas at Austin, 1991.

[50] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, 11, 2017.

[51] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct. 1986.

[52] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[53] Johannes Schemmel, Daniel Brüderle, Andreas Grübl, Matthias Hock, Karlheinz Meier, and Sebastian Millner. A

wafer-scale neuromorphic hardware system for large-scale neural modeling. In *International Symposium on Circuits and Systems (ISCAS)*, pages 1947–1950, May 2010.

[54] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: VGG and residual architectures. *Frontiers in Neuroscience*, 13, 2019.

[55] Sen Song, Kenneth D. Miller, and L. F. Abbott. Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nature Neuroscience*, 3(9):919–926, Sept. 2000.

[56] Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. BranchyNet: Fast inference via early exiting from deep neural networks. In *23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469, Dec. 2016.

[57] Vincent Vanhoucke, Andrew Senior, and Mark Z. Mao. Improving the speed of neural networks on CPUs. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS*, 2011.

[58] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *European Conference on Computer Vision (ECCV)*, pages 3–18, 2018.

[59] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in Neuroscience*, 12, 2018.

[60] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, Yuan Xie, and Luping 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.

[61] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S. Davis, Kristen Grauman, and Rogerio Feris. BlockDrop: Dynamic inference paths in residual networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8817–8826, 2018.

[62] Ran Xu, Chen-lin Zhang, Pengcheng Wang, Jayoung Lee, Subrata Mitra, Somali Chaterji, Yin Li, and Saurabh Bagchi. ApproxDet: Content and contention-aware approximate object detection for mobiles. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, SenSys '20, pages 449–462, New York, NY, USA, Nov. 2020. Association for Computing Machinery.

[63] Le Yang, Yizeng Han, Xi Chen, Shiji Song, Jifeng Dai, and Gao Huang. Resolution adaptive networks for efficient inference. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2369–2378, 2020.

[64] Wenrui Zhang and Peng Li. Temporal spike sequence learning via backpropagation for deep spiking neural networks. In *Advances in Neural Information Processing Systems*, volume 33, pages 12022–12033. Curran Associates, Inc., 2020.

[65] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6848–6856, 2018.

[66] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–73, Mar. 1996.