Neuroevolution Guided Hybrid Spiking Neural Network Training

Sen Lu, Abhronil Sengupta
School of Electrical Engineering and Computer Science
The Pennsylvania State University
University Park, PA 16802, USA
Email: sengupta@psu.edu

***Abstract*—Neuromorphic computing algorithms based on Spiking Neural Networks (SNNs) are evolving to be a disruptive technology driving machine learning research. The overarching goal of this work is to develop a structured algorithmic framework for SNN training that optimizes unique SNN-specific properties like neuron spiking threshold using neuroevolution as a feedback strategy. We provide extensive results for this hybrid bio-inspired training strategy and show that such a feedback-based learning approach leads to explainable neuromorphic systems that adapt to the specific underlying application. Our analysis reveals 53.8%, 28.8% and 28.2% latency improvement for the neuroevolution-based SNN training strategy on CIFAR-10, CIFAR-100 and ImageNet datasets respectively in contrast to state-of-the-art conversion based approaches. The proposed algorithm can be easily extended to other application domains like image classification in presence of adversarial attacks where 43.2% and 27.9% latency improvements were observed on CIFAR-10 and CIFAR-100 datasets respectively.**

***Index Terms*—Spiking Neural Networks, Neuroevolution, Adversarial Attack, Neuromorphic Computing**

I. Introduction

Spiking Neural Network (SNN) based next-generation brain-inspired computational paradigms are emerging to be a disruptive technology driving machine learning research due to its unique temporal, event-driven behavior. SNN computing models are driven by the fact that biological neurons process information temporally and the computation is triggered by sparse events or spikes transmitted from fan-in neurons. Recent work has demonstrated that event-driven SNNs can result in a significant reduction of power consumption and communication overhead in hardware implementations of Artificial Intelligence (AI) platforms by *exploiting 'dynamic' sparsity in neural activations* [1], [2]. In addition to event-driven computing in the network itself, such a computing framework is an ideal fit for the emerging market of low-power, low-latency event-driven sensors [3] that capture spatio-temporal information in the spiking domain. Such an end-to-end pipeline across the stack from sensors to the hardware and computational primitives enables us to truly leverage advantages from event-driven computation and communication.

While the true potential of SNNs is expected to be demonstrated on spatio-temporal application drivers triggered by sparse events [3], [4] by leveraging its temporal processing capability [5], [6], significant research has been also performed to establish its near-term efficacy on standard static recognition tasks [7]–[13], routinely performed by conventional deep learning methods (referred to as Analog Neural Networks (ANNs) [14], hereafter). The vast majority of works in this domain have focused on rate encoding frameworks [14] where the operation of the ANN is distributed as binary information over time in the SNN, resulting in a significant reduction of peak power consumption [15]. To achieve supervised SNN training, two competing approaches are usually adopted:

(i) ANN-SNN conversion: In this scenario, an ANN is trained with specific constraints [7], [8], [12], [14] and subsequently converted to an SNN for event-driven inference on neuromorphic hardware. The conversion process is enabled by the equivalence of Rectified Linear Unit (ReLU) functionality of ANN neurons to the operation of an Integrate-Fire (IF) spiking neuron. The method takes advantage of standard ANN backpropagation techniques like stochastic gradient descent but is limited by the baseline ANN accuracy. Recent works have been directed at minimizing the accuracy loss during the conversion process [12], [13] and have reported competitive accuracies in large-scale machine learning tasks.

(ii) Direct SNN training: Direct SNN training by adopting backpropagation through time (BPTT) has also proven successful recently, albeit in simpler image classification tasks. Gradient descent is usually performed in SNNs by approximating the spiking neuron operation by surrogate gradients to avoid the discontinuity in the neuron transfer function due to discrete spiking behavior [5], [6]. While SNN training from scratch would probably benefit from temporal processing in neuromorphic chips, current near-term GPU-enabled training suffers from limited scalability due to exploding memory requirements for BPTT.

Relative advantages and disadvantages of the two approaches are still being explored. Initial work has suggested that direct SNN training from scratch [16] or a hybrid method of fine-tuning an ANN-SNN converted network for a few epochs through BPTT [10] can significantly reduce the SNN inference latency. However, recent approaches have shown that significant latency reduction can be also achieved through simple design-time and run-time optimizations in the ANN-SNN conversion process as well [12]. This is also intuitive since the application drivers for image classification tasks are static and do not involve temporal information. Also, gradient descent is utilized to minimize the classification

error in the rate encoding framework for both scenarios and not the inference latency. The ANN-SNN conversion process essentially abstracts the SNN operation in a time-averaged fashion during the training process without exploiting precise timing information for gradient descent.

This paper is an attempt to develop a structured algorithmic framework for the ANN-SNN conversion process. The key parameter driving the event-driven temporal behavior of neurons in the SNN is the neuron threshold. A higher neuron threshold is useful for distinguishability of temporal evidence integration [8] and therefore translates to higher accuracy. A higher threshold also causes the neurons to spike less frequently thereby increasing the spiking sparsity at the cost of increased latency. Inference latency (impacting delay) and sparsity of the spike train (impacting power) are key metrics governing the energy efficiency (delay $\times$ power) of SNNs implemented on neuromorphic hardware. Hence, we can abstract the SNN network performance (accuracy/latency/power/energy) to be a function of the neuron thresholds in each layer of the network. It is worth mentioning here that all neurons in a particular layer have the same threshold to ensure consistent rate encoded information in each layer. Previous works have mainly optimized neuron thresholds to maximize accuracy [8] or adopt a sub-optimal heuristic choice for all thresholds in the network to reduce inference latency with minimal accuracy drop [12]. However, different layers' thresholds of a network may have varying non-linear impact on the SNN efficiency metric and a holistic singular choice for the entire network may not be optimal. Further, the thresholds may also need to be retuned for different efficiency metrics of choice. Driven by this observation, we propose a hybrid training framework where a converted SNN is optimized in tandem with a neuroevolutionary algorithm. Once an ANN with appropriate constraints for conversion has been trained, we optimize the layerwise threshold using a neuroevolutionary algorithm. Neuroevolution optimized neural networks is a growing topic of interest [17] guided by the notion that biological brains are an outcome of natural evolution. It is worth mentioning here that our proposal is not specific to the optimizer. We chose evolutionary algorithms due to their simple gradient-free operation, parallelizability and ability to outperform reinforcement learning algorithms at scale [17], [18]. The neuroevolution process considers a space of possible candidate solutions (defined by a set of layerwise neuron thresholds) and evaluates a cost function (latency, accuracy, among others) by evaluating the candidate SNN on a subset of the training set through the evolution generations. The additional computing overhead due to the hybrid approach is therefore driven by relatively cheaper SNN inference runs. The main contributions of the paper can be summarized as:

(i) We present a simple automated framework to optimize an SNN by a hybrid training process that does not suffer from computationally expensive operations like BPTT.

(ii) We evaluate our approach with regards to SNN inference latency improvements on static image classification tasks and adversarial attack scenarios (CIFAR-10, CIFAR-100 and ImageNet datasets). The framework can be easily extended to involve hardware aware constraints as well like peak power or energy consumption in specific layers.

(iii) We present design insights to interpret the optimized SNN thresholds. For image classification and adversarial attack scenarios, we obtain an interpretable understanding of the need for layerwise SNN optimization.

II. Related Works

Hybrid SNN training: Prior work has considered hybrid SNN training approaches [9], [10], [16], [19]. Relevant to our approach, Ref. [9] considered training an ANN-SNN converted spiking network through BPTT for a few epochs to improve on inference latency. However, during the second stage of BPTT training, gradient descent was performed not only on the weights, but also on the neuron thresholds and additional leak parameters that were introduced in this second training stage. The requirement of joint optimization of weights and thresholds may not be necessary since the ratio of the two governs the spiking neuron behavior [8]. Further, optimization of additional leak parameters adds to the computational burden of SNN BPTT. It is also unclear whether the superior SNN performance is attributed primarily to a fine-tuned optimized threshold or whether time-based information in training also plays a role.

In contrast, our algorithm adopts a simplistic approach of fine-tuning only the SNN thresholds to optimize the metric of choice using neuroevolutionary algorithms. Neuroevolutionary algorithms are easily parallelizable and the search parameter space in our scenario is bounded by the number of network layers and hence is not computationally expensive. The search process also involves evaluation of the cost function which is equivalent to the relatively cheaper SNN inference simulations and does not suffer from the explosive computational requirements of BPTT. The work also aims to serve as a benchmark for static image classification tasks to address the question of whether BPTT training from scratch or fine-tuning adds significantly to the training process. We provide results to substantiate that conversion techniques might produce competitive SNNs in application drivers not exploiting temporal information.

Neuroevolution in SNN training: Evolutionary algorithms have been used for training SNNs [20]–[22] where the computational unit (neuron/synapse) parameters and network architectures have been optimized. A variety of techniques like EONS [20] and HyperNEAT [21] algorithms have been used to train the networks from scratch. However, the techniques have been primarily evaluated on simple machine learning tasks. Hence, the scalability of the approaches remains unclear. Our work considers a hybridized approach where a supervised conventional SNN is optimized with a neuroevolutionary algorithm depending on the cost function of choice (accuracy, latency, among others), thereby leveraging the scalability of gradient descent approaches.

Significance driven layerwise optimizations: There have been a plethora of works recently in the deep learning com-

munity on layerwise optimizations of different parameters based on their significance to a relevant cost function. For instance, bit widths of weights and activations per layer have been optimized from computation requirement perspective [23]–[27]. Distinct from prior methods, this work explores significance-driven layerwise optimizations for SNN training.

III. Preliminaries

A. Spiking Neural Networks

Let us first consider the algorithmic formulation underpinning ANN-SNN conversion [7], [13]. In T timesteps, for an N-layer SNN converted by copying the weights W_n from an ANN (where $n \in N$), suppose that a particular neuron in the n-th layer at the t-th timestep has membrane potential denoted by V_n^t. When the membrane potential is greater than the threshold Vth_n, the neuron is reset by subtracting Vth_n from the potential. The membrane potential dynamics of the subtractive IF neurons in response to the input signal x_n^t for the n-th layer can be expressed as the following:

$$V_n^{t+1} = V_n^t + W_n * x_n^t - Vth_n * \mathbb{1}_{V_n^t > Vth_n} \tag{1}$$

where, $\mathbb{1}_{V_n^t > Vth_n}$ is an indicator function defined as:

$$\mathbb{1}_{V_n^t > Vth_n} \rightarrow \{0,1\} = \begin{cases} 1 & \text{if } V_n^t > Vth_n \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

As the neuron accumulates spikes over time, assuming $V_n^0 = 0$, the membrane potential for a particular neuron of the n-th layer can be expressed as:

$$V_n^T = W_n * \sum_{t=0}^{T} x_n^t - Vth_n * \sum_{t=0}^{T} \mathbb{1}_{V_n^t > Vth_n} \tag{3}$$

In the rate encoding framework, the average magnitude of the input spikes over T timesteps, $\hat{x}_n = \sum_{t=0}^{T} x_n^t / T$, represents the equivalent SNN input activation for the n-th layer. Simplifying Eqn. 3,

$$\frac{V_n^T}{T} = W_n * \hat{x}_n - \frac{Vth_n}{T} * \sum_{t=0}^{T} \mathbb{1}_{V_n^t > Vth_n} \tag{4}$$

The average input spikes to the $(n+1)$-th layer, $\hat{x}_{n+1}$, is the summed indicator function $\sum_{t=0}^{T} \mathbb{1}_{V_n^t > Vth_n}$. Hence Eqn. 4 can be rearranged as:

$$\hat{x}_{n+1} = \frac{W_n * \hat{x}_n}{Vth_n / T} - \frac{V_n^T}{Vth_n} \tag{5}$$

Assuming that the remaining V_n^T will be less than the threshold Vth_n and will not result in a spike, the neuron transfer function can be formulated with a clipping function as the following [13]:

$$\hat{x}_{n+1} = \frac{1}{T} * clip\left(\left\lfloor \frac{W_n * \hat{x}_n}{Vth_n / T} \right\rfloor, 0, T\right) \tag{6}$$

where, a clipping function $clip(a, b, c)$ restricts the value a to be minimally b or maximally c, and does not affect a's value when $b \leq a \leq c$. As shown in Eqn. 6, the output of a layer is critically dependent on the threshold Vth of the layer and is a bit discretized version of the ReLU functionality, thereby enabling ANN-SNN conversion. It is worth mentioning that this simplification of neuron transfer function may differ slightly from the actual network simulation due to positive and negative membrane potential cancellations [13] or multiple neuron fan-in [8].

B. Differential Evolution Algorithm

Differential Evolution (DE) is a parallel direct search method that optimizes a solution iteratively through evolving candidate solutions. Unlike other optimization techniques such as gradient descent that requires the problem to be differentiable, DE can be applied to noisy and discrete problems. DE starts with a population P of initial candidate solutions (randomly initialized or normally distributed around the preliminary solution). In each iteration, the existing candidates are mutated and evaluated by a cost function, and the best ones become members of the next generation. The evolution of new solutions is achieved by two operations, namely 'mutation' and 'crossover'.

(i) Mutation in DE algorithm refers to adding the weighted difference between two candidates to the third. The mutation process to obtain the i-th vector $\vec{v}_{g+1}$ at generation $g+1$ is given by:

$$\vec{v}_{i,g+1} = \vec{x}_{r1,g} + M \times (\vec{x}_{r2,g} - \vec{x}_{r3,g}) \tag{7}$$

where, $\vec{x}_{r1,g}$ is the $r1$-th vector of generation g, $r1, r2, r3 \in \{1, 2, .., P\}$ are random indices in the population. $M \in [0, 2]$ is a real-valued hyper-parameter controlling the extent of mutation in differential variation.

(ii) Crossover adds diversity by creating a trial vector $\vec{u}_{i,g+1}$ with problem dimension D at generation $g+1$:

$$\vec{u}_{i,g+1} = (u_{i,g+1}(1), u_{i,g+1}(2), ..., u_{i,g+1}(D)) \tag{8}$$

in which,

$$u_{i,g+1}(j) = \begin{cases} v_{i,g+1}(j) & \text{if } (rand(j) \leq C) \text{ or } j = randInd(\vec{v}_{i,g+1}) \\ x_{i,g}(j) & \text{otherwise} \end{cases} \tag{9}$$

where, $j \in 1, 2, ..., D$, $u_{i,g+1}(j)$ is the j-th element of the trial vector $\vec{u}_{i,g+1}$, $rand(j)$ is a real-valued uniform random number generator (RNG) outcome with the range [0, 1] evaluated at j-th time; C is another real-valued hyper-parameter that controls the extent of inheritance from the mutant vector $\vec{v}_i$ in the trial vector $\vec{u}_i$. $randInd(\vec{v}_{i,g+1})$ randomly selects an index from the given vector's dimension $1, 2, ..., D$ and the condition after 'or' enforces that there is at least one element from $\vec{v}_i$. The candidate solution $\vec{u}_{i,g+1}$ will be evaluated against $\vec{x}_{i,g}$ on the same cost function and the one with the lower cost will be selected as the member of $(g+1)$-th generation. Considering that the DE solution takes G generations to converge, the total number of function evaluations (*nfe*) during the optimization process is therefore given by:

$$nfe = G \times P \tag{10}$$

In this work, we used the DE implementation by a Python-based toolkit 'SciPy' [28], which is based on the algorithm outlined in Ref. [29].

IV. Neuroevolution Guided Hybrid SNN Training Algorithm

As discussed previously, our proposed neuroevolution optimized SNN models are trained using a hybrid approach - **(i)** standard ANN-SNN conversion [12] followed by **(ii)** DE optimization of SNN neuron thresholds. The DE optimization is driven by a cost-function evaluated on randomly chosen subsets from the training set. The random shuffling of the sub-dataset adds a regularization effect to the training process. Next, we discuss our cost-function formulation for handling the accuracy-latency tradeoff in standard image classification tasks. We utilize a similar approach for adversarial attack scenarios on the same dataset and show that the thresholds adapt to the new cost-function, thereby showing the flexibility of the approach. Finally, we also provide insights to explain the optimal threshold choice.

A. Latency-Accuracy Tradeoff Driven Optimization and Interpretibility

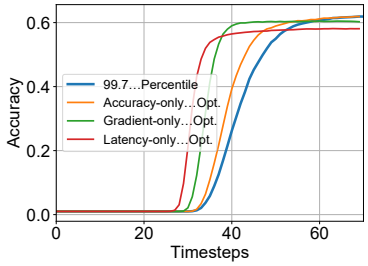


Fig. 1. Impact of various components of the cost function on the accuracy-latency tradeoff for VGG-15 model on CIFAR-100 dataset.

Our multi-objective DE cost-function consists of weighted factors to optimize the latency of the SNN along with the final accuracy. In particular, the latency is abstracted by the timestep at which it reaches the highest gradient in the accuracy-time variation function. The resulting costs are scaled by hyperparameters (α, β and γ) and then linearly summed up. To summarize, the cost function is as follows:

$$Cost = \alpha \times J + \beta \times (1 - max(\nabla)) + \gamma \times (1 - Acc[T]) \quad (11)$$

where, J is $\underset{t}{\text{argmax}}\{\nabla Acc(t)\}$, the timestep at which the SNN reaches the highest gradient in accuracy $max(\nabla)$ with respect to time. The maximal gradient magnitude is also added to the cost function to guide solutions toward models with sharper accuracy-timestep transitions such that latency required to reach a specific accuracy is minimized. We observed that this was critical to achieving the latency-accuracy tradeoff. Finally, the cost function also includes $Acc[T]$, the final accuracy of the model at timestep T (a sufficiently long time window for inference) where $Acc[]$ is a function of accuracy against time.

It is worth reiterating here that the accuracy is evaluated over randomly chosen subsets of the training set for each candidate solution. The impact of each individual component in the cost function is depicted in Fig. 1.

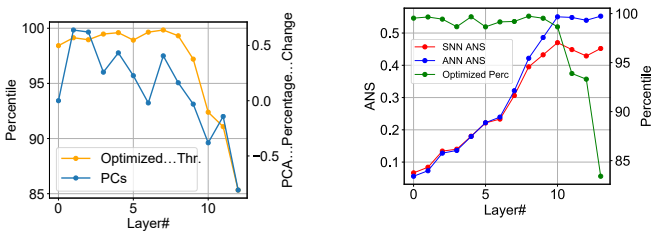


Fig. 2. The thresholds are expressed as the percentile of the maximum ANN activations. Both the figures are plotting one of the best solutions in their respective scenarios. (a) The optimized threshold shows a similar general trend as the principal components. (b) Blue and red: layerwise ANS value (left vertical axis) of the ANN and the converted + optimized SNN respectively. ANS is significantly reduced after optimization. Green: The optimized threshold (right vertical axis) shows drastic reduction after layer 10 corresponding to the layers where the ANS metric is significantly reduced.

Fig. 1(a) depicts the optimized threshold (expressed as a percentile of the maximum ANN activation [12]: higher percentile values translate to higher threshold) as a function of layer number for a typical run. In order to attain an understanding for the importance of layerwise neuron threshold optimization, we hypothesized that this might be correlated to the significance of a particular layer toward its prediction capability. For this purpose, we used Principal Component Analysis (PCA) - one of the prominent tools that can be used to quantify a neural network layer's significance [30]. In short, PCA can be thought of as an orthogonal transformation that maps uncorrelated variables in the input data points and forms a basis vector set that maximizes the variance in different directions. Generally, neural network models project the input into higher dimensions as layers get deeper with the goal of achieving linear separability at the final output layer. Therefore, the calculation of the Principal Components (PCs) of each layer's feature map is able to quantify the projective ability of each layer and thus its significance.

We performed PCA on the feature maps before the non-linear activation to examine the redundancy in every layer as the dimension increases. To explain the first, say $R\%$, of the variance in the feature map of the layer, only a number of the PCs, denoted by k, are needed. Given an activation map $\mathbb{P}$ with dimension $B \times H \times W \times F$, where B is the mini-batch size, H and W are height and width of the filter respectively, and F is the number of filters, it is first flattened to 1D in the first 3 dimensions. This makes the activation $\mathbb{Q}$ a 2D matrix with dimension $K \times F$ where $K = B \times H \times W$. Singular value decomposition is applied to $\mathbb{Q}^T\mathbb{Q}$ to obtain L eigenvectors v_i and eigenvalues λ_i. The total variance P_{var} is given by:

$$P_{var} = \sum_{i=1}^{L} \sigma_{ii}^2 \quad (12)$$

Function `CalculateCost` ($Acc[], \alpha, \beta, \gamma$) **:**
  Data:
  $Acc[t]$: A list of size T, where t = {1, 2, ... , T}
  $\alpha, \beta, \gamma \in \mathbb{R}$
  Result: *cost*
  $\nabla[] \leftarrow \frac{\partial(Acc)}{\partial t}$ `// Gradient of accuracy function`
  `/* Use Eqn. 11 to evaluate the cost */`
  $cost \leftarrow \alpha \times \underset{t}{\text{argmax}}\{\nabla[t]\} + \beta \times (1 - max\{\nabla\}) + \gamma \times (1 - Acc[T])$
  return $cost$

Function `DE` ($SNN, Thresholds[], TH, std, P, M, C, G, \alpha, \beta, \gamma$) **:**
  Data:
  SNN: Base SNN model
  $Thresholds[]$: A 2D array of shape $P * N$, where N is the number of layers, P is the population size
  TH: Base threshold values at 99.7 percentile
  std: Desired standard deviation of the initial population
  M: Mutation factor
  C: Crossover factor
  G: Number of maximum generations
  $\alpha, \beta, \gamma \in \mathbb{R}$: Scaling factors of the cost function
  Result: *bestThr*: The threshold with lowest evaluation score
  $Thresholds[] \leftarrow$ `Initialize` (*TH, P, std*) s.t. $Thresholds[, n] \leftarrow X \sim \mathcal{N}(TH[n], std^2)$
  Randomly select B samples from the training set and create mini-dataset S
  $Acc[] \leftarrow 0$ `// length of` T
  bestCost $\leftarrow$ 0, bestThr $\leftarrow \phi$ `// Variables for tracking`
  for $g \leftarrow 1$ **to** G **do**
    for $p \leftarrow 1$ **to** P **do**
      `/* Generates new candidate thresholds after applying Eqn. 7 - 9 */`
      $NewThreshold[] \leftarrow$ `Evolution`$(Thresholds[], p, M, C)$
      `/* Initialize SNN with the new thresholds for evaluation */`
      SNN.`Update` ($Threshold$)
      for $samplebatch$ **in** S **do**
        `/* SNN inference */`
        $Acc[] \leftarrow$ ***SNN***$(samplebatch, T)$
      end
      `/* Apply Eqn. 11 */`
      $cost \leftarrow$ ***CalculateCost***$(Acc[], \alpha, \beta, \gamma)$
      if *bestCost*>*cost* **then**
        $bestCost \leftarrow cost$
        $bestThr \leftarrow NewThreshold[]$
    end
  end
  return $bestThr$

Algorithm 1: DE Guided Hybrid SNN Training Algorithm

The significance of component λ_i would be simply $\frac{\lambda_i}{P_{var}}$. The first k principal components explain variance of a threshold value R:

$$R = \frac{\sum_{i=1}^{k} \lambda_i^2}{\sum_{i=1}^{L} \lambda_i^2} \tag{13}$$

The ratio R is used as a threshold for the algorithm to calculate the first k PCs and k suggests the number of significant components required after removing the redundancy in $\mathbb{Q}$. After obtaining k PCs for every layer in the SNN to explain a fixed threshold of $R\%$ variance (99.9% in our case), we interpreted a layer's significance to be proportional to the percentage increase in the number of PCs in comparison to the previous layer, i.e. the layer contributes significantly to the transformation of the input data provided to it by the previous layer. The percentage layerwise changes in PCs are plotted in Fig. 1(a), and interestingly the general trend matches with the variation of layerwise optimized neuronal thresholds. This is explainable since a higher spiking threshold allows more time for evidence integration, thereby improving SNN accuracy by ensuring more significant layers perform more accurate computations.

B. Adversarial Attack Driven Optimization and Interpretability

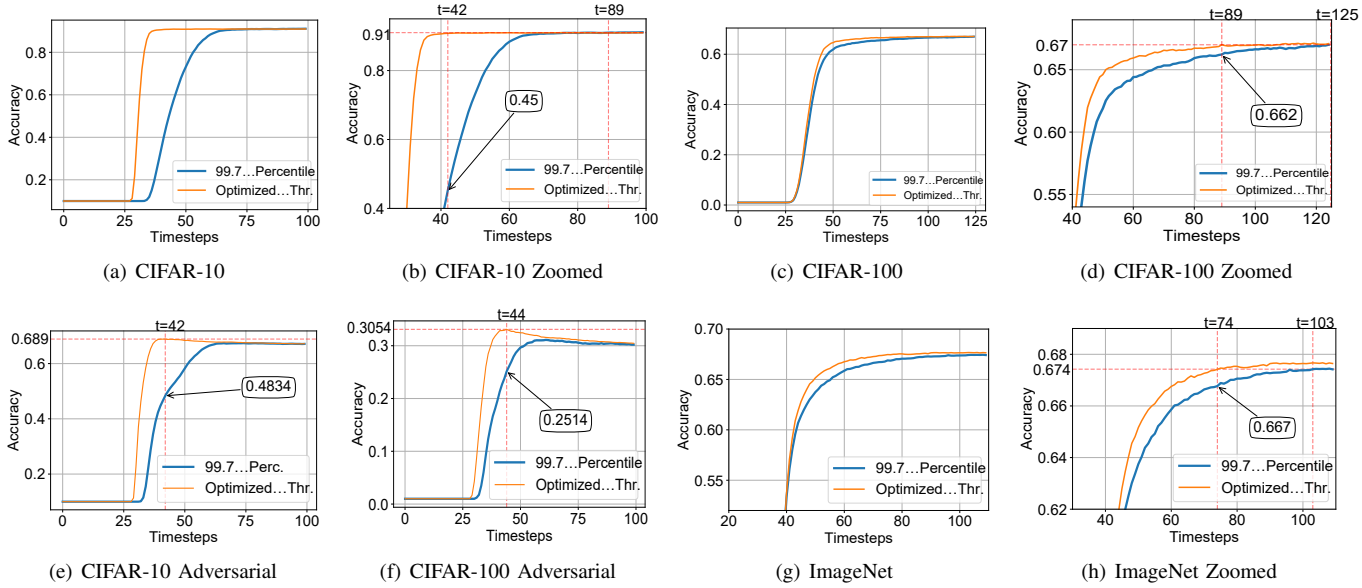
Next, we show that neuron threshold optimization is not application agnostic, thereby requiring the need for a cross-stack optimization. To substantiate our motivation, we consider SNN adversarial attack scenarios. Adversarial attack in neural networks refers to malicious attempts to mislead the model prediction. Since neural networks are proven to be vulnerable in such attacks [31], it becomes a non-trivial task to optimize the model for adversarial scenarios. While there are a plethora of adversarial attack algorithms [32], we used the vanilla version of the Fast Gradient Sign Method (FGSM) attack as a proof of concept for our optimization method's adversarial robustness. Details in the adversarial setup and implementation will be discussed in the next section.

We applied our neuroevolutionary guided SNN training strategy in this case but optimized for adversarial accuracy-latency tradeoff. Fig. 1(b) depicts the optimized threshold (expressed as a percentile of the maximum ANN activation) as a function of layer number for a typical run. However, the trend shows a slightly different distribution of thresholds as compared to the normal accuracy scenario. We observe that the deeper layers exhibits a similar downward trend of thresholds but this occurs only after layer 10 in the adversarial scenario whereas the network optimized for normal accuracy shows this trend much before (explained by % changes in PCs, as discussed in the previous subsection).

To explain this trend, we used Adversarial Noise Sensitivity (ANS), $\mathbb{A}_\delta$, as a metric for measuring layerwise perturbation in neural networks [27]. It is defined as the error ratio between a particular layer's perturbed adversarial activation and the

TABLE I
ALGORITHM HYPERPARAMETERS FOR VARIOUS DATASETS. *TRAINING COST COMPUTED USING EQN. 17

Dataset	α	β	γ	Initial Perc.	std	Population Size P	No. of Generations G	nfe	Training Cost*
Image Classification									
CIFAR-10	1	10	500	99.7	0.15	25	25.9	647.5	38.85
CIFAR-100(VGG15)	1	40	20	99.7	0.15	25	26.55	663.75	39.825
CIFAR-100(VGG11)	0.7	60	200	99.7	0.13	35	7.8	312	16.38
ImageNet	1	70	110	99.8	0.13	20	6.08	121.66	0.475
Image Classification with Adversarial Attack									
CIFAR-10	1	2	60	99.7	0.25	25	21.94	548.5	13.2
CIFAR-100	1	2	60	99.7	0.25	25	24.22	605.5	14.5



(a) CIFAR-10 (b) CIFAR-10 Zoomed (c) CIFAR-100 (d) CIFAR-100 Zoomed

(e) CIFAR-10 Adversarial (f) CIFAR-100 Adversarial (g) ImageNet (h) ImageNet Zoomed

Fig. 3. Accuracy vs timesteps for neuroevolutionary optimized SNN against homogeneously normalized (using 99.7 percentile of maximum activation [12]) SNN on CIFAR-10 dataset. Iso-time and iso-accuracy comparison are denoted by dotted-red line and textboxes.

unperturbed original activation and can be expressed by the following equation:

$$\mathbb{A}_{\delta,\mathrm{n}} = \frac{||a^n_{adv} - a^n||_2}{||a^n||_2} \tag{14}$$

where, a^n is the activation map of the n-th layer and the subscript adv denotes the same activation with adversarial input. In summary, the higher the ANS value of a particular layer, the higher is the sensitivity to noise of that layer. In other words, the layers with high ANS values will perform worse than the layers with low ANS values under the same degree of adversarial attack. In the SNN case, we use the cumulative spikes as the activation:

$$\mathbb{A}_{\delta,\mathrm{n}} = \frac{||\sum_{t=1}^{T} x^n_{adv,t} - \sum_{t=1}^{T} x^n_t||_2}{||\sum_{t=1}^{T} x^n_t||_2} \tag{15}$$

where, x^n_t is the n-th layer's spike at timestep t and adv still denotes the adversarial version; T is the total duration of inference. We can observe from Fig. 1(b) that the highest ANS values start from layer 10, which incidentally correlates with the trend of layerwise optimized neural thresholds.

To understand the relationship between neuron threshold and noise sensitivity, one needs to consider the activation discretization caused by the firing threshold. As shown in Eqn. 6, the output of a layer is critically dependent on the threshold Vth of the layer and is a bit discretized version of the ReLU functionality. Thus, the SNN neuron activation representation can be considered to be discretized due to the spiking behavior. When the threshold is lower in the denominator of Eqn. 6, there will be more discrete states and vice versa. Therefore lower firing threshold should relate to layers with higher noise sensitivity since reduced precision/discretization results in minimizing the adversarial perturbation [33], [34]. To summarize, in adversarial scenarios, the optimal set of thresholds attributes low thresholds to high ANS layers to increase discretization to resist the effect of adversarial noise.

V. EXPERIMENTS AND RESULTS

A. Datasets and Implementation

We evaluated our proposal on the CIFAR-10, CIFAR-100 [43] and the large-scale ImageNet [44] dataset. CIFAR-10 and CIFAR-100 datasets consist of 10 and 100 classes respectively. They include 60,000 32 $\times$ 32 colored images partitioned into

TABLE II
PERFORMANCE BENCHMARKING OF OUR PROPOSAL AGAINST PRIOR WORKS

Reference	Method	Architecture	SNN Accuracy	Timesteps
CIFAR10				
[35]	ANN-SNN	2C, 2L	82.95%	6000
[8]	ANN-SNN	VGG16	91.55%	2500
[36]	Phase-coding	VGG16	91.2%	1500
[37]	Burst-coding	VGG16	91.4%	1125
[38]	Time-Till-First-Spike	VGG16	91.40%	680
[7]	ANN-SNN	4 Conv, 2 FC	90.85%	400
[39]	ANN-SNN	3C,2L	77.43%	400
[10]	Hybrid	VGG16	92.02%	200
[16]	Backprop	VGG9	90.45%	100
[12]	ANN-SNN	VGG15	91.03%	91
This work	**Neuroevolutionary SNNs**	**VGG15**	**91.05%**	**42**
CIFAR100				
[36]	Phase-coding	VGG16	68.6%	8950
[37]	Burst-coding	VGG16	68.77%	3100
[40]	ANN-SNN	VGG16	70.09%	768
[38]	Time-Till-First-Spike	VGG16	68.8%	680
[10]	Hybrid	VGG11	67.87%	125
[12]	ANN-SNN	VGG11	67.00%	125
This work	**Neuroevolutionary SNNs**	**VGG11**	**67.00%**	**89**
ImageNet				
[8]	ANN-SNN	VGG16	69.96%	2500
[40]	ANN-SNN	VGG16	71.34%	768
[7]	ANN-SNN	VGG16	49.61%	400
[10]	Hybrid	VGG16	65.19%	250
[12]	ANN-SNN	VGG15	67.40%	103
This work	**Neuroevolutionary SNNs**	**VGG15**	**67.40%**	**74**

TABLE III
PERFORMANCE BENCHMARKING OF OUR PROPOSAL AGAINST PRIOR WORKS FOR SNN ADVERSARIAL ATTACKS. ALL FGSM ARE WHITE-BOX ATTACKS AND USE $\epsilon = 8/255$.

Reference	Attack	Method	Architecture	ANN	SNN	Timesteps
CIFAR10						
[41]	FGSM	Backprop	ResNet20	1.8%	3.8%	200
[41]	FGSM	Backprop	VGG5	10.4%	15.0%	100
[42]	FGSM	Backprop	VGG9	61.7%	51.6%	70
[12]	FGSM	ANN-SNN	VGG15	67.42%	67.40%	74
This work	**FGSM**	**Neuroevolutionary SNNs**	**VGG15**	**67.42%**	**68.9%**	**42**
CIFAR100						
[41]	FGSM	Backprop	VGG11	17.1%	15.5%	200
[12]	FGSM	ANN-SNN	VGG15	30.54%	31.11%	61
This work	**FGSM**	**Neuroevolutionary SNNs**	**VGG15**	**30.54%**	**33.1%**	**44**

50,000 and 10,000 training and testing images respectively. ImageNet 2012 is a much more challenging dataset with 1000 object categories that include 1.28 million images for training and 50,000 images for validation. The ImageNet images are randomly cropped into 224×224 pixels before being fed into the network. All images are normalized with zero mean and unit variance and shuffled during the training and DE optimization phase. The ANN models are pretrained VGG15 architectures based on constraints described in our prior work [12]. All experiments are conducted in 'PyTorch' framework using 'BindsNet' toolbox [45] with the 'SciPy' toolbox providing efficient DE algorithm implementation.

For the adversarial attack scenario, we used FGSM as a white box attack where the model parameters and network structure are fully available to the attacker. It utilizes the gradient of the original input and then perturbs it to create an adversarial version that maximizes the loss. This perturbation process can be summarized as:

$$\hat{X} = X + \epsilon \times sign(\nabla_X L(w, X, y)) \tag{16}$$

where, $\hat{X}$ is the perturbed image, X is the original input image, ϵ is the hyper-parameter to adjust the extent of perturbation, $\nabla_X L(w, X, y)$ is the gradient of the loss L given model parameter w, input X and label y, $sign()$ operation provides the direction of the gradient (in terms of '1's and '-1's). In our case, we adopted $\epsilon = 8/255$, commonly used in other works. Further details can be found in Ref. [46].

B. Results

The specific hyperparameter settings for our algorithm are specified in Table I for the various datasets and applications. For the DE algorithm, we used a typical setting of the mutation

rate (M is a random number between 0.5 and 1.5) and crossover rate (C = 0.7). It is worth reiterating here that only the training set is utilized during the neuroevolutionary optimization process. Considering that the DE algorithm is initialized with P particles and takes G generations to converge (measured by averaging over 20 runs), the excess overhead of running our hybrid training technique is tabulated as "Training Cost" in Table I and is computed in terms of the training set size by:

$$Training\ Cost = (E \times G \times P)/D_{train} \tag{17}$$

where, E is the total number of images used for cost function evaluation per particle per generation and D_{train} is the total number of images in the training set. Table I illustrates the advantage of our proposed algorithm in terms of scalability. The number of evaluation images required for the optimization process is primarily determined by the dimensionality of the optimization space rather than the size of the training set of the dataset. Hence, the "Training Cost" reduces significantly for complex datasets like ImageNet. This is in stark contrast to BPTT guided hybrid training approaches where backpropagation based gradient updates will require significantly large training datasets.

The performance of our proposed hybrid SNN training technique for CIFAR-10, CIFAR-100 and ImageNet datasets are depicted in Fig. 3 including adversarial attack scenarios. Significant latency improvement is consistently observed in all cases in contrast to a uniform percentile-based threshold optimization scheme. Iso-accuracy and iso-latency improvements for latency and accuracy respectively are also provided. A detailed comparison of the performance of our algorithm against prior work is provided in Tables II-III.

C. Comparison Against Backpropagation Through Time (BPTT) Fine-Tuning

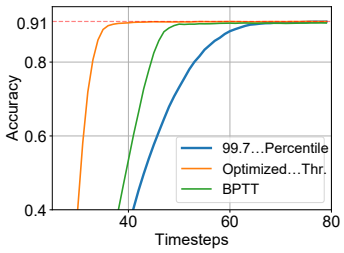
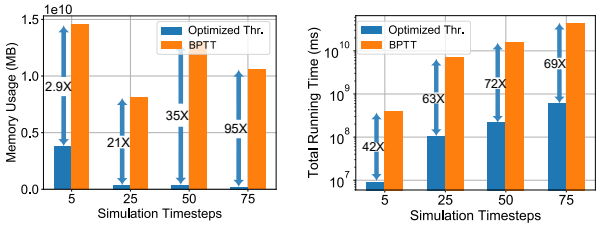


Fig. 4. Performance of VGG15 model on the CIFAR-10 dataset based on various training techniques - (blue) ANN-SNN conversion: 99.7 Percentile, (orange) Hybrid neuroevolutionary approach: Optimized Thr. and (green) BPTT: Hybrid training with backpropagation through time.

As mentioned before, our work is most relevant to hybrid SNN training approaches where the network is fine-tuned using BPTT after conversion [9], [10]. While the computational overhead is significantly higher in BPTT based approaches, another important difference between the two approaches lies in the absence of any temporal information in our neuroevolutionary optimization process. In order to benchmark the performance of the two hybrid training techniques, we performed BPTT fine-tuning from the same initialized converted SNN model as used in our neuroevolutionary algorithm. For BPTT, the network layers are unfolded at each timestep for IF operations. The BPTT method uses surrogate gradient for IF neurons [47]:

$$\frac{\partial p_i^t}{\partial V_i^t} = \gamma max\{0, 1 - |V_i(t)|\} \tag{18}$$

where, p is the output spike train, $V_i(t)$ is the normalized membrane potential voltage of neuron i at timestep t. γ is a hyper-parameter to dampen the error which is set to 0.15 in our case. For our experiments, we used a pre-trained VGG15 model on CIFAR-10 dataset, initialized with 99.7 percentile thresholds for the IF neuron layers. The BPTT algorithm was run for 25 epochs and the network was unrolled over 70 timesteps. However, as shown in Fig. 4, while the hybrid BPTT training performed better than a simple conversion approach, it was outperformed by our proposed hybrid neuroevolutionary approach. While recent versions of hybrid BPTT training [9] have reported only 5 timesteps as SNN latency, it is probably attributed to performing gradient descent on additional introduced parameters like neuron leak. Further, latency is re-defined to exclude intrinsic delay of an SNN where the neuron in each layer spikes at the current timestep instead of the next, and therefore eliminates the intrinsic layerwise SNN delay. While this is a simple method to reduce SNN latency, it may potentially have limitations in neuromorphic chip designs in terms of spike routing or parallel spike processing capability. It is worth mentioning here that additional optimizations like learnable membrane time constants [9], [48], network architectures like Residual networks [49], conversion error calibration techniques [13], [50], hybrid spike encoding [51] are complementary to the current proposal and can be augmented in the algorithm to further minimize the inference latency. Tables II-III therefore includes primarily basic SNN architectures based on IF nodes without any additional optimizations to substantiate the importance and interpretability of the need for layerwise threshold optimization. The dimensionality of the optimization algorithm can be easily expanded to incorporate additional optimization parameters like membrane potential leak, spike encoding rate, among others.



(a) Memory Usage (b) Expected Running Time

Fig. 5. Memory usage and running time comparison of hybrid neuroevolutionary and BPTT based approaches of VGG-15 model on ImageNet dataset, with maximum batch-size (21, 2, 2, 1) for (5, 25, 50, 75) timesteps respectively.

To quantitatively substantiate the computational benefit of our proposed hybrid neuroevolutionary training approach against BPTT based methods, we also report the memory usage and running time of the two methodologies on the ImageNet dataset in Fig. 5. The memory usage was profiled and the extrapolated running time for our proposed neuroevolutionary algorithm is calculated as:

$$Total\ Running\ Time = \bar{\tau} \times Training\ Cost \times \frac{D_{train}}{B} \quad (19)$$

where, $\bar{\tau}$ is the average running time per batch (averaged over 20 batches), "Training Cost" is calculated from Eqn. 17 and B is the batch-size. The average running time $\bar{\tau}$ is used to minimize the fluctuations caused by external processes. The "Training Cost" of BPTT was considered to be 20 epochs as reported in prior literature [10]. It is worth mentioning here that unlike our proposed algorithm, BPTT is heavily memory-constrained for large scale datasets like ImageNet even for 5 timesteps, as shown in Fig. 5. Our algorithms were run on Nvidia Tesla V100 16GB GPUs where we had to limit the batch-size for the BPTT based hybrid training approach due to memory limit. The batch-sizes of Fig. 5 are chosen based on the maximum memory capacity of the BPTT based approach and iso-batch-size comparison is performed with the neuroevolutionary method. As illustrated in the plot, the situation worsens significantly with increasing timesteps due to drastic increase in gradient trace information. As shown in Fig. 5, our proposed neuroevolutionary method requires $2.9\times$ less memory and $42\times$ less running time than BPTT based framework even for 5 timesteps used for SNN simulation. It is worth mentioning here that iso-timestep based comparison may not be valid for further optimized SNN algorithms like BPTT with membrane potential leak [9], [48] and therefore require further benchmarking.

VI. Conclusions

In conclusion, the work explores a neuroevolution-based hybrid SNN training strategy that optimizes SNN specific parameters like neuron spiking threshold after the conversion process. While significantly outperforming state-of-the-art approaches in terms of accuracy-latency tradeoffs in image classification tasks including adversarial attack scenarios, the work highlights the need for significance-driven layerwise SNN optimization schemes leading to explainable SNNs. We also highlight that the work outperforms computationally expensive BPTT based fine-tuning approaches since temporal information may not be relevant in static image classification tasks. Future exploration into application drivers with temporal information [4], [52] or temporal spike encoding schemes [53], [54] is expected to truly leverage the full potential of BPTT based SNN training strategies.

Acknowledgments

The work was supported in part by the National Science Foundation grants CCF #1955815, BCS #2031632 and ECCS #2028213 and by Oracle Cloud credits and related resources provided by the Oracle for Research program.

References

[1] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[2] M. Davies, N. Srinivasa, T. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. 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.

[3] G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis *et al.*, "Event-based vision: A survey," *arXiv preprint arXiv:1904.08405*, 2019.

[4] K. Mahapatra, A. Sengupta, and N. R. Chaudhuri, "Power system disturbance classification with online event-driven neuromorphic computing," *IEEE Transactions on Smart Grid*, 2020.

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

[6] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks," *IEEE Signal Processing Magazine*, vol. 36, pp. 61–63, 2019.

[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," *Frontiers in neuroscience*, vol. 11, p. 682, 2017.

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

[9] N. Rathi and K. Roy, "DIET-SNN: Direct input encoding with leakage and threshold optimization in deep spiking neural networks," *ArXiv*, vol. abs/2008.03658, 2020.

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

[11] Y. Wu, L. Deng, G. Li, J. Zhu, Y. Xie, and L. Shi, "Direct training for spiking neural networks: Faster, larger, better," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 1311–1318.

[12] S. Lu and A. Sengupta, "Exploring the connection between binary and spiking neural networks," *Frontiers in neuroscience*, vol. 14, 2020.

[13] S. Deng and S. Gu, "Optimal conversion of conventional artificial neural networks to spiking neural networks," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=FZ1oTwcXchK

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

[15] S. Singh, A. Sarma, N. Jao, A. Pattnaik, S. Lu, K. Yang, A. Sengupta, V. Narayanan, and C. R. Das, "NEBULA: a neuromorphic spin-based ultra-low power architecture for SNNs and ANNs," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 363–376.

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

[17] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, no. 1, pp. 24–35, 2019.

[18] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," *arXiv preprint arXiv:1712.06567*, 2017.

[19] C. Lee, P. Panda, G. Srinivasan, and K. Roy, "Training deep spiking convolutional neural networks with STDP-based unsupervised pre-training followed by supervised fine-tuning," *Frontiers in neuroscience*, vol. 12, p. 435, 2018.

[20] C. D. Schuman, J. P. Mitchell, R. M. Patton, T. E. Potok, and J. S. Plank, "Evolutionary optimization for neuromorphic systems," in *Proceedings of the Neuro-inspired Computational Elements Workshop*, 2020, pp. 1–9.

[21] D. Elbrecht and C. Schuman, "Neuroevolution of spiking neural networks using compositional pattern producing networks," in *International Conference on Neuromorphic Systems 2020*, 2020, pp. 1–5.

[22] C. D. Schuman, J. S. Plank, A. Disney, and J. Reynolds, "An evolutionary optimization framework for neural networks and neuromorphic architectures," in *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2016, pp. 145–154.

[23] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8612–8620.

[24] I. Chakraborty, D. Roy, I. Garg, A. Ankit, and K. Roy, "Constructing energy-efficient mixed-precision neural networks through principal component analysis for edge intelligence," *Nature Machine Intelligence*, vol. 2, no. 1, pp. 43–55, 2020.

[25] I. Garg, P. Panda, and K. Roy, "A low effort approach to structured CNN design using PCA," *IEEE Access*, vol. PP, pp. 1–1, 12 2019.

[26] M. F. F. Khan, M. M. Kamani, M. Mahdavi, and V. Narayanan, "Learning to quantize deep neural networks: A competitive-collaborative approach," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[27] P. Panda, "QUANOS: adversarial noise sensitivity driven hybrid quantization of neural networks," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2020, pp. 187–192.

[28] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright *et al.*, "SciPy 1.0: fundamental algorithms for scientific computing in python," *Nature methods*, vol. 17, no. 3, pp. 261–272, 2020.

[29] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[30] I. Chakraborty, D. Roy, I. Garg, A. Ankit, and K. Roy, "Constructing energy-efficient mixed-precision neural networks through principal component analysis for edge intelligence," *Nature Machine Intelligence*, vol. 2, pp. 1–13, 01 2020.

[31] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[32] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, "Adversarial attacks and defences: A survey," *arXiv preprint arXiv:1810.00069*, 2018.

[33] A. S. Rakin, J. Yi, B. Gong, and D. Fan, "Defend deep neural networks against adversarial examples via fixed and dynamic quantized activation functions," *arXiv preprint arXiv:1807.06714*, 2018.

[34] S. Sen, B. Ravindran, and A. Raghunathan, "EMPIR: Ensembles of mixed precision deep networks for increased robustness against adversarial attacks," *arXiv preprint arXiv:2004.10162*, 2020.

[35] E. Hunsberger and C. Eliasmith, "Spiking deep networks with LIF neurons," *arXiv preprint arXiv:1510.08829*, 2015.

[36] J. Kim, H. Kim, S. Huh, J. Lee, and K. Choi, "Deep neural networks with weighted spikes," *Neurocomputing*, vol. 311, pp. 373–386, 2018.

[37] S. Park, S. Kim, H. Choe, and S. Yoon, "Fast and efficient information transmission with burst spikes in deep spiking neural networks," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[38] S. Park, S. Kim, B. Na, and S. Yoon, "T2FSNN: Deep spiking neural networks with time-to-first-spike coding," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[39] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, 2015.

[40] B. Han, G. Srinivasan, and K. Roy, "RMP-SNN: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network," 06 2020, pp. 13 555–13 564.

[41] S. Sharmin, N. Rathi, P. Panda, and K. Roy, "Inherent adversarial robustness of deep spiking neural networks: Effects of discrete input encoding and non-linear activations," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, pp. 399–414.

[42] S. Sharmin, P. Panda, S. S. Sarwar, C. Lee, W. Ponghiran, and K. Roy, "A comprehensive analysis on adversarial robustness of spiking neural networks," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8.

[43] A. Krizhevsky, V. Nair, and G. Hinton, "CIFAR-100 (canadian institute for advanced research)." [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[44] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[45] H. Hazan, D. J. Saunders, H. Khan, D. Patel, D. T. Sanghavi, H. T. Siegelmann, and R. Kozma, "BindsNET: A machine learning-oriented spiking neural networks library in python," *Frontiers in Neuroinformatics*, vol. 12, Dec 2018. [Online]. Available: http://dx.doi.org/10.3389/fninf.2018.00089

[46] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6572

[47] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, "Long short-term memory and learning-to-learn in networks of spiking neurons," ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 795–805.

[48] W. Fang, Z. Yu, Y. Chen, T. Masquelier, T. Huang, and Y. Tian, "Incorporating learnable membrane time constant to enhance learning of spiking neural networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 2661–2671.

[49] W. Fang, Z. Yu, Y. Chen, T. Huang, T. Masquelier, and Y. Tian, "Deep residual learning in spiking neural networks," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[50] Y. Li, S. Deng, X. Dong, R. Gong, and S. Gu, "A free lunch from ANN: Towards efficient, accurate spiking neural networks calibration," in *International Conference on Machine Learning*. PMLR, 2021, pp. 6316–6325.

[51] G. Datta, S. Kundu, and P. A. Beerel, "Training energy-efficient deep spiking neural networks with single-spike hybrid input encoding," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[52] S. Singh, A. Sarma, S. Lu, A. Sengupta, V. Narayanan, and C. R. Das, "Gesture-SNN: Co-optimizing accuracy, latency and energy of snns for neuromorphic vision sensors," in *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2021, pp. 1–6.

[53] K. Yang and A. Sengupta, "Stochastic magnetoelectric neuron for temporal information encoding," *Applied Physics Letters*, vol. 116, no. 4, p. 043701, 2020.

[54] B. Petro, N. Kasabov, and R. M. Kiss, "Selection and optimization of temporal spike encoding methods for spiking neural networks," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 2, pp. 358–370, 2019.