BAYESIAN-BASED MULTI-OBJECTIVE HYPERPARAMETER OPTIMIZATION FOR ACCURATE, FAST, AND EFFICIENT NEUROMORPHIC SYSTEM DESIGNS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Maryam Parsa

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2020

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Prof. Kaushik Roy, Chair
School of Electrical and Computer Engineering

Prof. Anand Raghunathan
School of Electrical and Computer Engineering

Prof. Vijay Raghunathan
School of Electrical and Computer Engineering

Prof. Saeed Mohammadi
School of Electrical and Computer Engineering

Dr. Catherine D. Schuman
Oak Ridge National Laboratory

Approved by:

Prof. Dimitrios Peroulis
Head of the School Graduate Program

To Hannah, My Sweet Little Angel..

"Everything you can imagine is real"

ACKNOWLEDGMENTS

I would like to acknowledge Intel Corporation and Semiconductor Research Corporation (SRC) for their support of four years of my PhD studies under INTEL/SRCEA PhD Fellowship, Oak Ridge National Laboratory (ORNL) for supporting this journey for a year under ORNL/ASTRO program, and also Center for Brain Inspired Computing (C-BRIC), a Joint University Microelectronics program (JUMP) center sponsored by DARPA.

Some people leave footprints in your life forever, some people are so humble that when you talk with them you feel they are your closest friends, some people are so strong that no matter how stressed you are you feel relieved after talking with them, some people have no limits in their ideas, imaginations, support, and encouragement, some people are the best and caring advisor, teacher, role model, friend, and mentor, all at the same time. Prof. Kaushik Roy is among those, and I'm so fortunate to call him my PhD advisor. I'm truly grateful for all his advise, feedback, guidance, and being there for all of us no matter what time of the day and what day of the week is. All I can hope for is to make him proud of being my advisor someday.

I would like to thank my PhD committee member, my ORNL mentor, my dear Katie, Dr. Catherine Schuman, who is the best that could have happened to my career. I would like to thank her for helping me not only frame my research path and ideas, but also develop professionally by facilitating collaborations within and beyond ORNL. She is a fabulous researcher, supportive mentor, kind friend, and in a word a gem.

I also like to express my sincere gratitude to the rest of my PhD committee, Prof. Anand Raghunathan, Prof. Vijay Raghunathan, and Prof. Saeed Mohammadi for their invaluable feedback throughout my PhD journey. I'm also thankful to all my Purdue family, specially Prof. Ali Shakouri who is the best listener, and a marvelous

mentor, Prof. Ganesh Subbarayan, Prof. Shreyas Sen, Prof. Abhronil Sengupta, Prof. Priyadarshini Panda, Aayush Ankit, Nitin Rathi, Mustafa Ali, and Nicole Piegza. Special thanks to all my ORNL family, Parker Mitchell, Dr. Shruti Kulkarni, Daniel Elbrecht, Dr. Steven Young, Dr. Travis Johnston, Dr. Bill Kay, Dr. Prasanna Date, Dr. Derek Rose, Dr. Robert Patton, and Dr. Thomas Potok.

I wouldn't have been in a place I am right now without amazing guidance, help, and support from my phenomenal mentors at Intel Corporation, and University of Ottawa (uOttawa). I would like to thank Katherine Hoopman (Intel), with no doubt, she helped me to be the best version of me, and Prof. Mustapha Yagoub (uOttawa) who has been by my side since August 2011, the first day I left my home country. I will never forget what he told me the last time I met him before leaving uOttawa, "no matter what happens, and what you do, never ever forget your smile!".

Words cannot express how grateful I am to my parents, Maman Gita and Baba Parviz, to their devotion, unconditional support, sacrifices, and love. They are strength, backbone, and pillars of my life. I would also love to remind my beloved mother-in-law, Maman Sonia, who is the strongest woman I ever met, who I still cannot believe is not among us anymore, who I miss every single day that passes by. I'm also thankful to my father-in-law, Baba Farrokh, who is the symbol of patience to me.

I would like to thank my siblings, Moti and Mina, my brother-in-laws, Ali, Kavian, and Kamyar, and my beautiful niece, Melissa, for listening to me, and making my life meaningful, and cheerful. I'm so blessed to have every single one of you. I would like to specially mention my little Mina, who is mature enough to be on my side during the days I needed her the most, and who is young enough to cherish Hannah as if they are the same age. Also thank you Kavian to be the best brother I have ever wanted in my life! I also like to specially thank Chirine, who is my favorite and a friend that became family, and Yasaman, my person, who I can talk with for hours and nothing at all, and who I can say everything to with just a look.

I cannot thank enough my soulmate, Amir Koushyar, for his beyond words devotion, love, and care, who went above and beyond to make my PhD journey possible. Thank you for always having my back, and being the most phenomenal person in the universe. Finally, thanks to our magnificent little Hannah who has been the meaning of life, joy, and happiness to us. Because of you we laugh, and we dare to dream more than we ever have!

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

ABSTRACT

Parsa, Maryam Ph.D., Purdue University, December 2020. Bayesian-based Multi-Objective Hyperparameter Optimization for Accurate, Fast, and Efficient Neuromorphic System Designs. Major Professor: Professor Kaushik Roy.

Neuromorphic systems promise a novel alternative to the standard von-Neumann architectures that are computationally expensive for analyzing big data, and are not efficient for learning and inference. This novel generation of computing aims at "mimicking" the human brain based on deploying neural networks on event-driven hardware architectures. A key bottleneck in designing such brain-inspired architectures is the complexity of co-optimizing the algorithm's speed and accuracy along with the hardware's performance and energy efficiency. This complexity stems from numerous intrinsic hyperparameters in both software and hardware that need to be optimized for an optimum design.

In this work, we present a versatile hierarchical pseudo agent-based multi-objective hyperparameter optimization approach for automatically tuning the hyperparameters of several training algorithms (such as traditional artificial neural networks (ANN), and evolutionary-based, binary, back-propagation-based, and conversion-based techniques in spiking neural networks (SNNs)) on digital and mixed-signal neural accelerators. By utilizing the proposed hyperparameter optimization approach we achieve improved performance over the previous state-of-the-art on those training algorithms and close some of the performance gaps that exist between SNNs and standard deep learning architectures.

We demonstrate $> 2\%$ improvement in accuracy and more than $5X$ reduction in the training/inference time for a back-propagation-based SNN algorithm on the dynamic vision sensor (DVS) gesture dataset. In the case of ANN-SNN conversion-

based techniques, we demonstrate 30% reduction in time-steps while surpassing the accuracy of state-of-the-art networks on an image classification dataset (CIFAR10) on a simpler and shallower architecture. Further, our analysis shows that in some cases even a seemingly minor change in hyperparameters may change the accuracy of these networks by 5-6X. From the application perspective, we show that the optimum set of hyperparameters might drastically improve the performance (52% to 71% for Pole-Balance control application). In addition, we demonstrate resiliency of different input/output encoding, training neural network, or the underlying accelerator modules in a neuromorphic system to the changes of the hyperparameters.

1. INTRODUCTION

Advances in computing engines and graphic processing units (GPUs) as well as massively produced data from smart devices, social media, and internet, create an immense opportunity for machine learning and in particular deep neural networks (DNN) to solve tasks such as recognition and classification. DNNs are computationally expensive, and require substantial resources. Therefore, their computation is either carried out in the cloud, or in a neuromorphic computing system through domain-specific energy-efficient accelerators built with CMOS [1–4], or speculatively on resistive crossbars [5–7] and spintronics [8] based technologies to boost the performance and speed of DNNs.

Spiking neuromorphic system is an alternative computing platform that takes direct inspiration from biology in how information is processed. These biologically-inspired computing platforms not only offer tremendous energy efficiency for computing in resource-constrained environments such as mobile and edge devices, but also extend the ability to solve challenging machine learning problems due to their massive connectivity of synthetic neurons and synapses [9].

The in-memory computing capability of neuromorphic systems proposes a promising alternative or complement to von Neumann architectures that suffer from the low bandwidth between CPU and memory, also known as the von Neumann bottleneck [10]. In addition, the brain-like structure of spiking neuromorphic systems is suitable for on-line, real-time learning for certain tasks such as smart healthcare diagnosis on edge devices, special purpose applications on drones, and robotics.

Designing a high-performance neuromorphic computing system is reliant on not only maximizing accuracy, and speed of training or inference of the neural network, but also minimizing energy and area requirements of the underlying hardware. Therefore, the algorithm-hardware co-design is an indispensable step toward empowering

high-performance neuromorphic computing. The optimum design for a neuromorphic system (non-spiking or spiking) is highly dependent on the selection of the inherent hyperparameters (HPs) that belong to the algorithm, underlying hardware, the application, and in the case of spiking neuromorphic systems, the input/output encoding schemes.

In the deep learning community, for traditional artificial neural networks (ANNs), hyperparameter and network architecture decisions are often made by choosing an "off-the-shelf" network architecture and then relying on manual tuning (often based on the user's intuition) to customize the model's hyperparameters for a particular application. With ANNs, the community has had decades to build up "intuition" on how to make these decisions, though even that community often relies on optimization approaches to help make those decisions on non-standard problems [11]. However, in a non-spiking neuromorphic system, optimizing the ANN performance without considering the strong correlation between its HPs and the corresponding hardware specific parameters results in a sub-optimal and inefficient hardware architecture. For an ANN, the HPs include, but are not limited to, the number of hidden layers, kernel sizes, the choice of optimizer and non-linearity function. In addition, examples of hardware specific HPs are memory bandwidth, and pipelining in CMOS technologies [12], and the number of bits, the number of crossbars and the crossbar sizes in memristive crossbar accelerators [7].

In the spiking domain, the input/output information is received and generated in the form of spikes over time. Additionally, network dynamics include a notion of time in how the information is processed, which is often in the form of delays on the synapses or axons. Due to these differences with ANNs, spiking neural networks (SNNs) require adaptations to existing training algorithms or entirely new training approaches in order to train the networks to effectively perform new tasks. Liquid state machines [13], evolutionary-based algorithms [14], backpropagation-based [15, 16] and ANN-SNN conversion-based techniques [17, 18] are among the commonly

used training algorithms for SNNs. Similar to ANNs, the network architecture and hyperparameters of the model for SNNs must be defined before training begins.

For SNNs, a similar approach to determine HPs and network architectures is often taken. That is, the same "off-the-shelf" hyperparameters and network architectures for ANNs are chosen and then manually tuned. Unlike ANNs however, there is limited "community intuition" to help guide the manual tuning of these parameters. Additionally, as SNNs have fundamentally different computational characteristics than ANNs, there is no guarantee that HPs that behave well on ANNs will also behave well on SNNs. In fact, SNNs often fail to achieve the same level of accuracy as ANNs on tasks such as image classification, but it is not clear whether that difference in performance is due to the computational characteristics of SNNs or the algorithms that are training them, or if it is due to the lack of customization of hyperparameters and network architectures for SNNs.

Selection of HPs is critical for the design of accurate neuromorphic systems; however, there are often other considerations, such as optimizing speed of processing or energy efficiency, when utilizing custom hardware. In spiking neuromorphic computing systems, HPs and neural architecture decisions can have a significant impact on the network latency–the time required for processing of a single input spike across all the layers. A larger latency leads to an increased inference time, and in turn, compromises the energy efficiency of SNN architectures. Therefore, it is often necessary to take these factors into account when optimizing the HPs for the best network accuracy, which requires solving a multi-objective optimization problem.

In this work we propose a novel optimization framework built upon agent-based modeling and hierarchical Bayesian optimization techniques to obtain the optimum set of HPs for any neuromorphic system design. This generic framework is not only suitable for both non-spiking or spiking neural networks, but also handles different types of hardware (CMOS [19–21] or beyond-CMOS [7]). Bayesian optimization is a powerful tool for finding the optimal point of objective functions that are unknown and expensive to evaluate [22]. However, for problems with more than one objective

function Bayesian-only techniques are mathematically complex, and suffer from high dimensionality limitations in parameter-heavy models [23]. Other approaches such as Neural Architecture Search (NAS, [24]) also require massive computational resources. These factors were the driving forces to search for alternative algorithms to find the optimal set of hyperparameters.

Our proposed approach, Hierarchical Pseudo Agent-based Bayesian Optimization (Hierarchical-PABO [25–29]), is built upon using a supervisor agent correlating the results of isolated Bayesian estimations for each of the objective functions. The agent creates an extra set of Bayesian estimator focusing only on finding the Pareto frontier. The hierarchy of Bayesian optimizers enables predicting the Pareto frontier for complex problems regardless of the number of objective functions. The Pareto frontier is a set that consists of solutions in which no other is superior in optimizing objective functions (i.e. performance matrices such as maximizing the neural network's accuracy and speed of training/inference as well as minimizing the energy and area requirements of the underlying hardware). In other words, each member of the Pareto set is not dominated by other members of the set, where the dominance is defined as: The vector $\vec{a}$ dominates vector $\vec{b}$ notated as $\vec{a} \succ \vec{b}$ or $\vec{b} \prec \vec{a}$, iff $\forall i; f_i(a) \leq f_i(b)$ where f_i is the i-th objective function [30].

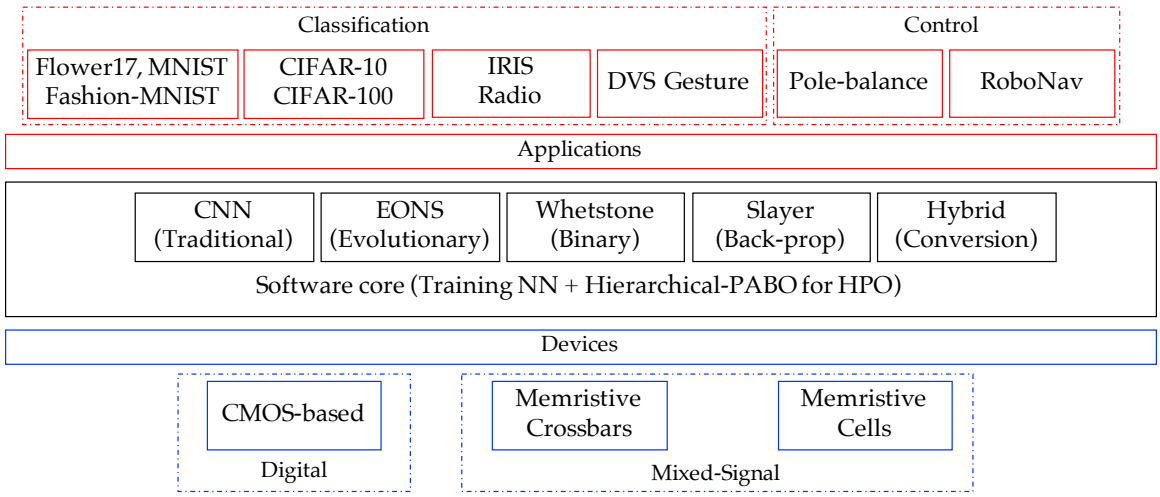

Fig. 1.1.: An overview of the Hierarchical-PABO framework

Figure 1.1 demonstrates an overview of the Hierarchical-PABO framework for a multi-objective optimization problem. The software core trains a neural network and optimizes the HPs using Hierarchical-PABO. We validated the approach on traditional convolutional neural network (CNN) on AlexNet [31] and VGG architectures [32], EONS [14] as an evolutionary-based spiking training algorithm, WHETSTONE [16] as a binary network approach, SLAYER [15] which is a backpropagation-based technique, and HYBRID [17] as a modified ANN-SNN conversion approach. We chose three different underlying hardware, digital (CMOS-based DANNA2 [19]), and mixed signal (memristive crossbar, PUMA [7], and memristive cells, mrDANNA [33]). We also considered several benchmarking applications such as image classification tasks (Flower17 [34], MNIST [35], Fashion-MNIST [36], CIFAR10 [37], CIFAR100 [37], IRIS [38], satellite radio signal [39], and DVS Gesture dataset [40]) as well as control tasks (canonical pole balancing [41], and autonomous robotic navigation [42]).

We demonstrate that by utilizing Hierarchical-PABO for neuromorphic computing system designs (ANN-based and SNN-based) we automatically discover optimum hyperparameters that outperform the network accuracy of the previous state-of-the-art results for all algorithms shown in Figure 1.1. Moreover, we show how this hyperparameter optimization (HPO) approach can include additional objectives beyond accuracy (e.g., minimizing training/inference time, energy and area requirements of the underlying hardware) and demonstrate that Hierarchical-PABO can find hyperparameters that produce models that simultaneously optimize several objectives. For example, we observe $> 2\%$ improvement in accuracy and more than $5\times$ reduction in the training/inference time for the SLAYER [15] algorithm on the DVS Gesture dataset. In addition, in the case of the HYBRID [17] technique, we demonstrate 30% reduction in time-steps while surpassing the accuracy of state-of-the-art networks on CIFAR10 on simpler VGG13 architecture, which we would expect to be more energy efficient. Our analysis further clarifies the significance of the present work by highlighting cases where even a seemingly minor change in hyperparameters can drastically change the performance of the network (by $5-6\times$). The speed and accuracy of

the framework enables designers to perform sensitivity analyses on hyperparameters to determine the resiliency of the system to the changes of the hyperparameters.

The salient features of our work are summarized below:

- In [25], we introduced PABO, which was the first step toward designing the Hierarchical PABO. In the case of PABO, there is no hierarchy of Bayesian estimators, rather the supervisor agent decides for the search direction in favor of the Pareto region, without any Bayesian estimator. By turning off the extra set of Bayesian estimator that is used to predict the Pareto frontier, Hierarchical-PABO reduces to PABO. We tested PABO on both AlexNet and VGG19 architectures on a memristive crossbar accelerator (PUMA [7]). Using PABO, we estimated sets of hyperparameters that belong to the Pareto region of a multi-objective optimization problem, where the objectives were maximizing the neural network's accuracy and minimizing the energy consumption of the underlying hardware. Compared to grid search, random search, and evolutionary-based hyperparameter optimization approaches (NSGA-II [43]), PABO obtains superior performance both in terms of accuracy and computational time (predicting the Pareto region at least 100x faster compared to the NSGA-II).

- Our work [26] is, to the best of our knowledge, the first in the literature that a hyperparameter optimization technique for spiking neuromorphic computing system was proposed and the effects of different types of hyperparameters on the overall performance of the system were analyzed. We not only discovered an optimum set of hyperparameters to maximize accuracy of an SNN, but also performed sensitivity analysis on spiking neuromorphic system hyperparameters, and discussed the strategic role of some sets of hyperparameters on the system's final performance. In addition, we demonstrated that hyperparameters of a resilient training framework for spiking neuromorphic systems such as EONS [14] have the least impact on the final performance of the system compared to the input encoding or hardware-specific hyperparameters.

- In [27], we showed that an optimum set of hyperparameters drastically increases the performance of Whetstone [16] as a binary neural network approach that can be deployed to neuromorphic hardware. We also observed that the best hyperparameters found for different datasets differ across the datasets, indicating the importance of specifically optimizing hyperparameters for each new problem when converting to binary communication. In [44], Whetstone is deployed on SpiNNaker [45], with slight drop in accuracy due to issues with input/output encoding. Here, we optimized the network using Whetstone, but we do not map the resulting networks to a neuromorphic hardware implementation, such as SpiNNaker [45] or Loihi [20]. As observed in [44], several other hyperparameters such as input/output encoding, different network topologies and training parameters will have an effect on this mapping performance. In the future, we plan to include how the network performs on real neuromorphic hardware as part of our training objectives in the hyperparameter and network architecture optimization process.

- In [28], we introduced Hierarchical-PABO as a novel approach that, with its simple yet effective underlying mathematics, is able to predict a Pareto frontier of a multi-objective hyperparameter optimization for both non-spiking and spiking neural network systems with only few evaluations. We defined sets of hyperparameters and estimated a Pareto region for three-objective optimization problem (performance, energy, and network size). This framework also paves the way to further analyze and study sensitivity and resiliency of the system due to the changes of the hyperparameters. The main current limitation of Hierarchical-PABO is scalability and ability to parallelize the approach. The goal of Hierarchical-PABO is predicting the Pareto region for a search space with reasonable ranges for the hyperparameters and with only few evaluations; it is not designed to compete with all NAS-based approaches that search the entire search space with massive computational resource requirements. However, improving scalability of Hierarchical-PABO paves the way for incorporating the technique in different frameworks with multiple layers of optimization problems and hyperparameters.

- In [29], we illustrated an approach for Hierarchical-PABO that has been successfully applied to two distinct SNN training algorithms, SLAYER [15] and HYBRID [17] with the goal of simultaneously optimizing SNN's accuracy and latency (the time required for processing of a single input spike across all the layers). Optimizing the latter further improves the practical usability of these algorithms. For the SLAYER [15] algorithm on the DVS Gesture dataset [40], we demonstrated that this approach achieved state-of-the-art results by increasing the Top-1 accuracy from 94.13% to 96.2%. In addition, we showed that with a multi-objective hyperparameter optimization approach, we are able to reduce network latency (training/inference times) by 5× while obtaining comparable accuracy.

  Using the proposed hierarchical Bayesian optimization, that contains a single-objective Bayesian approach for hyperparameter optimization of the ANN and an agent-based multi-objective Bayesian approach for hyperparameter optimization of the SNN, we optimized and trained networks that outperform the previous state-of-the-art HYRBID [17] training SNN results on the CIFAR10 and CIFAR100 dataset with VGG and RESNET architectures in terms of accuracy with more than 40% reduction in network latency (time steps). We demonstrated that the proposed approach can discover hyperparameters for simpler architectures that achieve higher accuracy and lower latency than previously published results. Both the reduction in architecture size and network latency have significant implications for energy efficiency of these architectures. For example, we demonstrated the results for CIFAR10 on VGG9 with improved accuracy compared to a much deeper and more energy-consumptive VGG16, and with 30% reduction in inference time.

Through these numerous examples, we also achieve one of the key goals of this work, which is to help close the gap in performance between ANNs and SNNs in resource-constrained environments without compromising the practicality of utilizing SNNs.

1.1 Contributions

We made the following contributions:

1. **A novel optimization framework based on hierarchical Bayesian optimization and agent-based modeling, suitable for both non-spiking and spiking neuromorphic systems**. With simple yet effective underlying mathematics, Hierarchical-PABO estimates the Pareto region for multi-objective hyperparameter optimization problems with few evaluations.

2. **One of the first techniques in the literature for co-designing software-hardware that is not limited to the number of objectives to optimize (network performance, energy consumption, size, speed of inference, etc.)**. Based on our knowledge, our proposed technique is one of the first techniques in the literature that simplifies the mathematical complexity of exclusive Bayesian approaches for multi-objective optimization. We do this by adding a supervisor agent and performing Bayesian optimization in different levels. This paves the way to effectively optimize more than two objective functions.

3. **Generic framework extendable to various artificial and spiking neural networks and the underlying digital, analog, or mixed-signal accelerators**. We tested our framework using various training techniques on several classification and control applications with both digital and mixed-signal accelerators as the underlying hardware. We were able to estimate the Pareto frontier regardless of the number of performance matrices, size of the search space, training algorithm, type of application or hardware.

4. **Superior performance in terms of accuracy and computational speed in finding the Pareto region compared to the state-of-the-art Genetic Algorithm (GA) optimization approach** (in scenarios where GA-based optimizations were available for comparison, [43]). Please see [25] for details of this contribution.

5. **Hierarchical-PABO closes the gap in performance between ANNs and SNNs for resource-constrained environments without compromising the practicality of utilizing SNNs**.

2. BACKGROUND AND RELATED WORK

In the era of the exigent need to design energy efficient neuromorphic systems for resource-constrained environments such as mobile edge devices, several approaches have been proposed in the literature to reduce the massive energy requirement of these systems. For artificial neural networks (ANNs), these techniques span from simplifying models, such as pruning and quantization [24, 46–48], to designing energy efficient architectures [49–52], and neural architecture search (NAS) [24]. In spiking neuromorphic domain, these include different training algorithms such as evolutionary optimization [14, 53], modified backpropagation techniques [15, 54, 55], binary communication [56], and hybrid approach [17] while deploying them on neuromorphic hardware such as [57, 58]. In this section, we briefly review the literature on each of these methods and continue with the added complexity of co-designing algorithm and hardware for neuromorphic systems. We then present the contribution of our work (Hierarchical-PABO) and how we fill the existing gap in a generic approach of co-designing software and hardware in the literature.

To reduce the energy requirement of neural network architectures, there have been a variety of model simplification techniques proposed by [46], and continued with [24, 47], and [48]. Each of these techniques focus on simplifying the neural network with different approaches of pruning, quantization, learning the connections, and leveraging sparsity. Designing energy-efficient architectures are also well-studied in the literature with flattened Convolutional Neural Network (CNN) [49], factorized CNN [50], conditional CNN [51, 59], and staged-conditional CNN [52]. More recently, compact structures such as MobileNets [60] and ShuffleNet [61] are also introduced and are specifically designed for mobile devices. Although both approaches of model simplification and efficient architecture design demonstrate promising results in reducing the energy requirements of neural networks, they do not necessarily yield to

the optimum designs for energy efficient accelerators. This is mainly due to the fact that they only locally search the space. In addition, layers with more parameters do not necessarily consume more energy [23, 62].

Spiking neural networks (SNNs) have great algorithmic promise as an energy-efficient machine learning technique, but training and learning in SNNs have proved to be difficult with the existing approaches. A common learning mechanism for SNNs is synaptic plasticity, such as spike-timing dependent plasticity (STDP) [63,64], but the utility of these approaches has been relatively limited. Another approach is evolutionary algorithms [14], which have the advantage that they can design all aspects of the network (structure and parameters) and are flexible with respect to applications, but can be slow to train.

Adapting existing backpropagation methods to work with SNNs is a widely used approach for training SNNs. These include training a traditional artificial neural network and then developing a mapping to an SNN [18, 65–68], adapting the training procedure to accommodate spiking neurons or binary activations [56], or changing the training procedure to leverage timing in the SNN [15, 69]. There are several key issues with these backpropagation-based approaches. First, by utilizing existing training approaches without much adaptation, it is not clear that SNNs will be able to establish an advantage over existing approaches. Second, defining the appropriate neural architecture and hyperparameters of these approaches is difficult and can require a tremendous amount of human effort. Third, to achieve comparable results with their ANN counterparts, these types of training algorithms require large training/inference time (time-steps), which negates many of the underlying benefits of spike-based approaches.

Several stakeholders play a role in designing a high-performance neuromorphic system, such as neural network itself (ANN or SNN), underlying hardware (CMOS-based on Beyond-CMOS), and in the case of spiking neuromorphic system, input/output encoding modules to encode the real-world data to spikes and vice-versa.

Different training algorithms, both in the non-spiking and spiking domains, have several HPs that have to be set and that can potentially significantly affect performance of the algorithm, such as ANN-specific HPs (kernel sizes, optimizer type, learning rate, etc), crossover and mutation rate for genetic algorithm approaches, number of neurons in spiking reservoir computing, and back propagation parameters and network structure in deep SNNs [18, 68]. In addition to algorithmic HPs, neuromorphic hardware also have HPs that can be set as part of a design process. These HPs include the number of required input/output neurons (sensors and actuators in the hardware), the range or resolutions of synaptic or neuronal delays and weights. Each of these HPs can play a role in the performance and energy requirements of the neuromorphic computing system. There have also been a variety of approaches proposed for converting data into spikes and some training or learning algorithms rely on a particular type of encoding to function properly. Rate-based and temporal-based are two of the most popular approaches for input encoding ([70]). Other approaches, including binning, have also been proposed to allow for higher resolution input values to be encoded over a shorter time period ([71]). These different encoding approaches require one or more HPs that have to be defined for the problem. Examples include the number of bins and spikes per bins for binning-base, and Poisson rate, lateral inhibition and homeostasis for temporal-based encoding. HPs for these modules should not only be optimized for their performances, but also be co-optimized to obtain the maximum algorithm-hardware performance (at least maximum accuracy, minimum energy and area requirement, and maximum speed of training/inference). Of course, there will be other possibilities for performance matrices such as sparsity, resiliency, and robustness.

We first review different hyperparameter optimization (HPO) techniques that are proposed in the literature for single objective optimization (neural network accuracy only) for both non-spiking and spiking domains, and then review the hardware-aware HPO techniques that is required in the neuromorphic computing platform.

HPO for neural networks used to be largely governed by rules of thumb [72]. Examples of these rules and practical guidelines for efficiently training large-scale deep neural networks are given in [73]. In addition, it is shown that random search outperforms grid search and manual search for HPO and has good theoretical guarantees and empirical evidence [74]. Continuing along this line of research, another approach is greedy sequential algorithms, which have shown promising results compared to random search [75].

Bayesian-based approaches have also been used for optimizing the hyperparameters of deep neural networks. It is shown that algorithms based specifically on the Gaussian process are the most call-efficient for hyperparameter optimization of deep neural networks [76]. DeepHyper [77] is a Python package that leverages the Balsam workflow and provides an interface for implementation and study of scalable hyperparameter search methods. In addition, HORD [78] is a deterministic and efficient method for hyperparameter optimization using radial basis function as the error surrogate in Bayesian-based methods, and its effectiveness is shown on MNIST and CIFAR-10 datasets.

To achieve higher performance and avoid human driven optimization, significant effort has been placed on automating architecture selection. A powerful method for obtaining the best performance ANN architecture designs is Neural Architecture Search (NAS) [24, 79]. The objective of these techniques is to automate architectural engineering to discover a network design which provides maximum performance [80–84]. NAS was started by Google Brain [24] to find an optimal neural architecture by searching for architectural building blocks on a small dataset and then transferring the block to larger ones. NAS was a starting point for a series of NAS-based approaches in recent years [85–87]. Reinforcement learning NAS [88] has also been used for HPO of deep neural networks. This approach suggests architectures with significantly less trainable parameters, shorter training times, and accuracies matching or surpassing the state-of-the-art models used on cancer dataset [89]. All of these works were

proposed to design a neural network with optimum performance, regardless of the energy requirement of the underlying neural accelerator.

Hardware-aware neural architecture designs can be categorized in three domains of multi-layer co-optimization [90], hardware-aware NAS [91–95], and Bayesian-based hyperparameter optimization [12,96,97]. Each one of these approaches have their pros and cons. While defining an optimum neural architecture with energy-efficient hardware in mind, the multi-layer co-optimization approach cannot easily be extended to generic platforms. Hardware-aware NAS techniques are time consuming and require substantial resources, and Bayesian-based methods are not well-suited for parameter-heavy models [23].

While the above approaches catered to deep neural networks, NAS approaches have been much less common in the realm of SNNs, where learning algorithms are still in their infancy. Differential evolution (DE) and self-adaptive differential evolution algorithms (SADE) is proposed by [98] to optimize the parameter space of synaptic plasticity and membrane adaptivity learning mechanisms in the lobula giant movement detector (LGMD) neuron that is driven by a dynamic vision sensor (DVS) camera. A neuro-evolutionary algorithm to optimize the hyperparameters of spiking neural networks is given at [99] and shown that the model trained using this approach outperforms all other models. In general; however, HPO and NAS approaches specifically for SNNs and spiking neuromorphic systems have been largely unexplored.

In Hierarchical-PABO [25–28] we propose a novel hardware-aware approach with minimum mathematical complexity suitable for both non-spiking and spiking neuromorphic computing systems. This framework is based on hierarchical Bayesian optimization and agent-based modeling. Using a set of Bayesian estimators in different levels and correlating them with a supervisor agent, we overcome the drawbacks of exclusive Bayesian approaches available in the literature. In addition, with the need to optimize several performance matrices in any neuromorphic computing system, the number of objective functions that Hierarchical-PABO is optimizing simultaneously is flexible.

3. METHODOLOGY

In order to systematically take the human knowledge out of the loop in selecting the optimum set of hyperparameters for a neuromorphic computing system (and in general any artificial intelligence-based platform), we chose Bayesian optimization as the core of our approach. In this section, we first review the basic mathematics of Bayesian modeling and Bayesian optimization for single objective optimization (SOO) problems [22, 26, 27]. We then present PABO (Pseudo Agent-based Bayesian Optimization) [25] for multi objective hyperparameter (MOO) problems, and finally we add a hierarchy to PABO design (Hierarchical-PABO: Hierarchical Pseudo Agent-based Bayesian Optimization), to improve efficiency and speed of predicting the Pareto region for MOO [28].

3.1 An Introduction to Bayesian Optimization

Bayesian optimization is a powerful tool for finding the optimum point of objective functions that are unknown and expensive to evaluate [100]. The problem of finding a global optimizer for an unknown objective function is formulated in Equation 3.1.

$$x^* = \underset{x \in X}{\operatorname{argmax}}\, f(x) \tag{3.1}$$

where X is the entire design space, and f is the black-box objective function without simple closed form. As summarized by [22], in a sequential manner, we search for the best location x_{n+1} to observe y_{n+1} point in order to estimate f. After N iterations, the algorithm suggests the best estimation of the black-box function f. This sequential approach is based on building a prior estimation over possible objective functions, and then iteratively re-estimating the prior model using the observations

from updating the Bayesian posterior model. The posterior representations are the updated knowledge on the objective function we are trying to optimize. We explore the search space by leveraging the inherent uncertainty of the posterior model and mathematically introducing a surrogate model, called the acquisition function α_n. The maximum point of this function is the next candidate point to observe (x_{n+1}) and guides the search direction toward the true representation of the objective function. The efficiency of Bayesian approach to estimate the global optimizer for the expensive black-box function with fewer evaluations relies on the ability of Bayesian technique to learn from prior belief on the problem and direct the observations by trading off exploration and exploitation of the design space.

In the context of neuromorphic computing, x is the system's hyperparameters such as inherent hyperparameters for different input/output encoding schemes, or population size or optimizer choice for various training techniques. Hardware-specific hyperparameters are also another choice for parameter x. Function f is the black-box objective function, such as accuracy of the network, energy or area requirements of the system, and speed of inference, for stochastic observations of y. A summary of the Bayesian approach is illustrated in the Figure 3.1. (See [75, 100, 101] for detailed tutorials.)

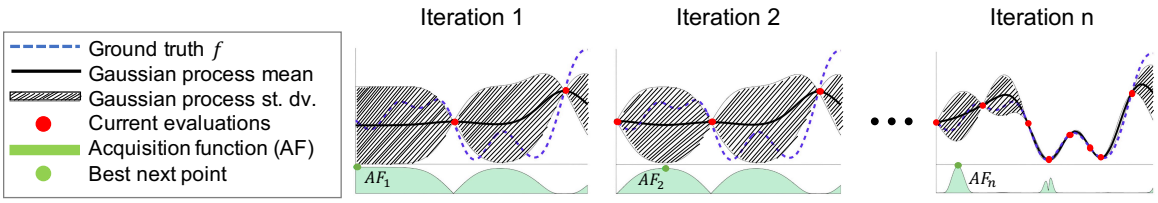

Fig. 3.1.: Summary of single objective Bayesian optimization. Reproduced with permission from [25]

In Figure 3.1, we are estimating an unknown objective function, ground truth f. We only have two observations (likelihood model) in iteration one (red dots). We first build our prior distribution (current belief) based on these observations using

Gaussian processes. The Gaussian distribution is shown with mean and standard deviation, solid black line, and highlighted dashed area, respectively. A surrogate model, acquisition function, is estimated for this posterior distribution, which is shown as the highlighted green function. The maximum point of the acquisition function (green dot) is the best next point to observe in the next iteration. As the new points are added to the observations in different iterations, the standard deviations, and therefore the uncertainty of estimating the ground truth function, is reduced. Each observation requires evaluating an unknown, expensive objective function. The ability of the Bayesian technique in predicting this function (ground truth in Figure 3.1) with few evaluations, speeds up the process of finding the optimum set of hyperparameters with minimum computational resources.

For configuring the Gaussian process, the covariance function is a positive definite kernel that specifies the similarity between points of observations. There are different methods to estimate this kernel function based on the smoothness, noise level and periodicity of the ground truth. In our experimental setup, we selected the Matern kernel function with smoothness value of 1.5. This particular kernel is selected due to the intrinsic stochastic nature, and noise level of our problem. Once we estimate the posterior distribution based on the likelihood model and the prior distribution, we build an acquisition function to guide the search direction. This acquisition function defines whether to search the space where the uncertainty is high (explore) or sample at locations where the model predicts high objectives (exploit). There are different methods to calculate this surrogate model such as improved-based, optimistic, and information-based policies [100, 102–108]. The choice of the method to use directly impacts the speed of convergence to the ground truth in Bayesian search. We chose *"expected improvement"* approach for the acquisition function. This selection does not impact the effectiveness or performance of our approach; rather, it only impacts the speed of searching the hyperparameter space and avoid trapping in local minima. (More details in selecting kernel or acquisition function can be found in [22]).

3.2 PABO: Pseudo Agent-based Bayesian Optimization

Figure 3.2a summarizes the PABO search process. The framework starts with selecting observations (at least two) from the design space. The design space is a set containing all possible HP combinations. The observations are the performance matrices values for a set of HP. These observations are then passed to separate Bayesian estimators for each performance metric. In this figure, for example, performance of the neural network (in terms of accuracy), energy usage, and size requirements of the underlying neural accelerator are the objective functions and performance matrices we would like to optimize. GP stands for Gaussian Process, and AF stands for Acquisition Function. For each objective, a Gaussian distribution is estimated followed by a surrogate model (acquisition function, AF). In this step, for each objective function, the optimum point of AF is the best HP to observe in the next iteration regardless of the search direction for other objectives. The process is then followed by a supervisor agent that evaluates the impacts of output HPs on the other posterior models and decides which HPs it must pass along. This agent decides on the set of HPs to evaluate at each step, the direction of the search process, and when to stop the technique. With a supervisor agent we reduce the complexity of the joint optimization problem, which in turn speeds up the algorithm to obtain the Pareto frontier compared to the state-of-the-art methods. Such capability is further beneficial for solving multi-objective problems with more than two objective functions.

In Figure 3.2b the estimated correlated posterior Gaussian distributions are shown for this example which is a three-objective optimization problem. This figures shows how each observation for isolated Bayesian estimators is helping the search direction toward the Pareto region of the problem using the supervisor agent. Throughout the process, the supervisor agent guides the search process to the Pareto frontier region and speeds up the procedure without adding extra complexity to the underlying mathematics.

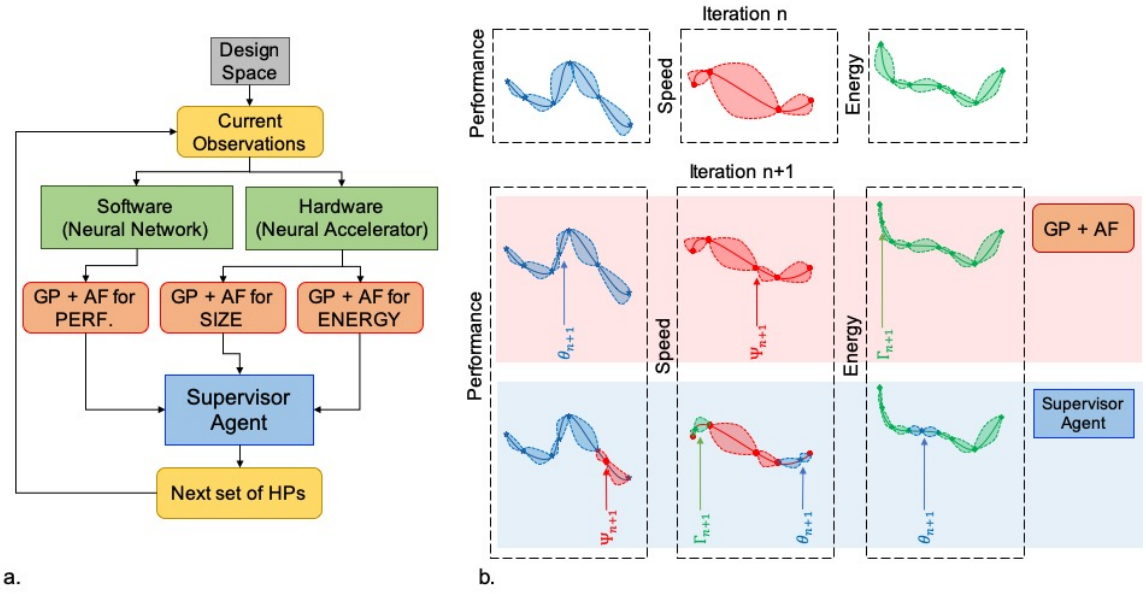

Fig. 3.2.: a. Overview of PABO framework, b. Estimated correlated posterior Gaussian distributions for multi-objective Bayesian optimization problem using PABO

3.3 Hierarchical-PABO: Hierarchical Pseudo Agent-based Bayesian Optimization

Hierarchical-PABO (Hierarchical Pseudo Agent-based Bayesian Optimization) is an ultra-efficient Bayesian-based optimization framework to find an optimum set of hyperparameters for designing an accurate neural network while minimizing the energy consumption and area requirement of the underlying hardware.

Figure 3.3 summarizes the Hierarchical-PABO framework. We randomly select two hyperparameter (HP) combinations from the design space. In the first level, these current observations are used to build Bayesian estimation posterior distributions for each objective function separately. We then define the acquisition function for each posterior model. The optimum point of these acquisition functions are the best next point (HP combination) to evaluate for their corresponding objective function. In the second level, the supervisor agent level, the process starts with all current observations (set of HP combinations) and the candidate HP combination that led to

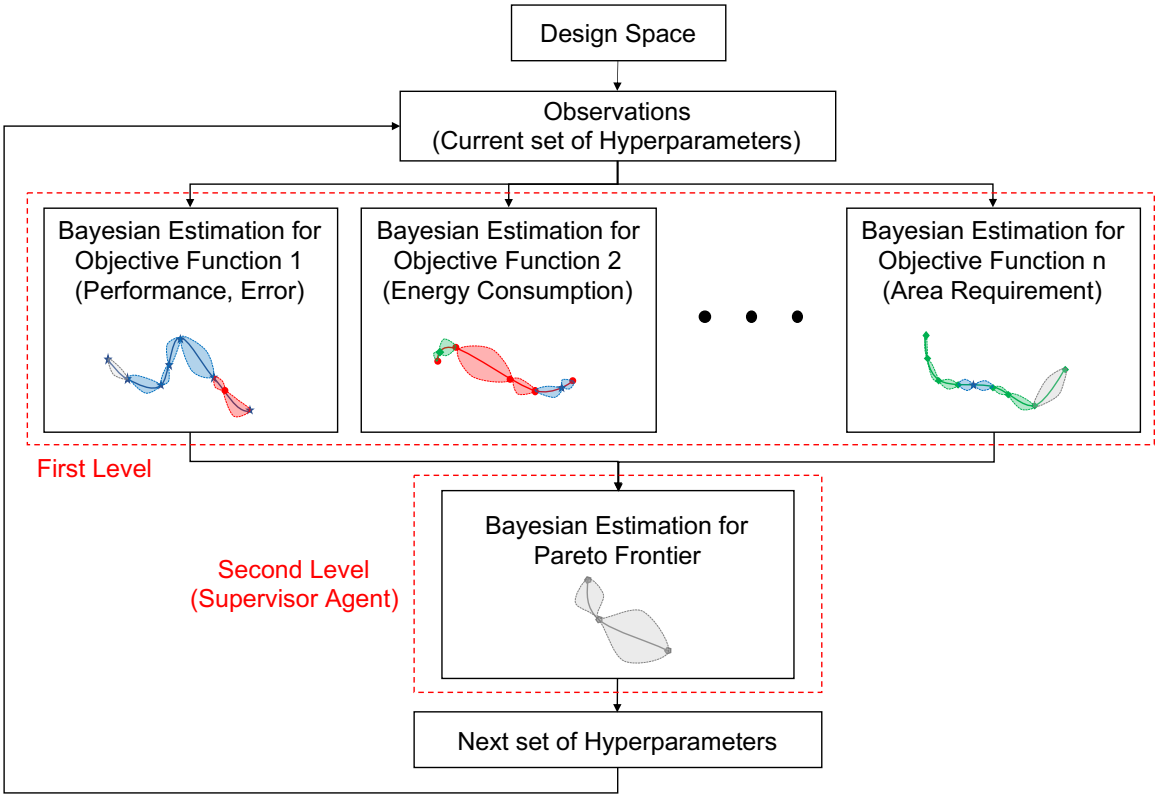

Fig. 3.3.: Overview of Hierarchical-PABO framework

the optimum value of the acquisition functions in the previous iteration. For these observations, we estimate an intermediate Pareto frontier function using a Gaussian distribution. This is calculated based on the observation points (on the Pareto front set), as well as a score calculated based on L1-norm of these points after being normalized. Therefore, a corresponding surrogate model (acquisition function) for this Gaussian distribution explores and exploits the search space with the goal of estimating the current intermediate Pareto function. The next best observation for this Pareto is then added to the observations for each Bayesian estimator. With this technique, we force the Bayesian approach to add extra observations that help in minimizing the current intermediate Pareto function. This function is updated iteratively and moved toward the actual Pareto region of the problem.

In Hierarchical-PABO, the Pareto Bayesian estimator in the second level plays a vital role in correlating the Bayesian estimators for each objective function in the first level. However, to speed up the search process, the supervisor agent might turn off this Pareto Bayesian estimator. If this extra Bayesian estimator is turned off, the supervisor agent takes HP combinations taken from optimum point of the acquisition function for each objective and only allow those that are in favor of moving toward the Pareto region.

Algorithm 1 illustrates the pseudo-code of Hierarchical-PABO framework. In this triple-objective optimization algorithm the black-box objective functions are shown with f_{perf}, f_{eng}, and f_{size}. f_{perf} is the performance of the neural network (ie. error), f_{eng} is the energy consumption of the underlying neural accelerator, and f_{size} is a proxy for area requirement of the design.

In iteration n, we have observations from the isolated Bayesian estimators for the objective functions. Among all these observations, we select and store those points that belong to the Pareto frontier (i.e. the HP points coming from any of the D_{perf}, D_{eng}, or D_{size} that are non-dominated), and their corresponding score vector (i.e. the vector containing the results of evaluating performance, energy, and size for that specific HP). Please note that this vector is not limited in size and can be adjusted based on the number of objective functions. These non-dominated points for this iteration, create D_{IntPar} set. In this step, assume that you would like to estimate a completely new function using Bayesian optimization (intermediate Pareto function). Bayesian optimization helps in estimating black-box functions with sets of observations. In the second level, this black-box function is a intermediate function that changes in every iteration as we learn more about the isolated Bayesian estimators in the first level. To build a posterior model for this intermediate function, we require a likelihood model (i.e. our observations) and a prior model. Observations are non-dominated HPs stored in the D_{IntPar} set. The prior Gaussian distribution model uses these observations along with a score dedicated for each observation. In Hierarchical-PABO we use a normalized summation of the score vectors for each HP,

and in this way, we represent a single score for each non-dominated point. We estimate a Gaussian distribution for these non-dominated HPs (From D_{IntPar}) and their corresponding scores IntPar_n, calculate an acquisition function ($AF_n(\tilde{\text{IntPar}})$), and optimize it. The optimum point of this acquisition function is the new HP that helps moving the current Pareto to the corner. This new HP is then added to all isolated Bayesian estimators in the first level and help with improving those estimations. By repeating this process, we move the intermediate function in the second level closer to the corner and therefore actual Pareto region of the problem.

In the Hierarchical-PABO framework, there are two different stopping criteria. One is after a predefined number of iterations in the Hierarchical-PABO process, and the other one is when the new observations (new set of hyperparameter) does not improve the Bayesian estimation. This happens when the surrogate model (acquisition function) converges to zero and the optimum point of this acquisition function cannot suggest a new set of hyperparameter that helps in exploring and exploiting the search space.

Algorithm 1 Hierarchical-PABO (for triple-objective optimization: performance, energy, size)

Notations: AF: Acquisition Function; p-norm: $||.||_p$; HP: hyperparameter; n: iteration number; O: Observations; β: estimated Pareto front set; $I_K := \{1, 2, ..., K\}$

Inputs: Three objective functions (performance, energy and size): f_{perf}, f_{eng}, f_{size}
HP_{all}: The set containing all possible combinations of hyperparameters (HPs)
Initial training datasets: θ, Γ, Ψ

Initialize: $n \leftarrow 1$
$flag \leftarrow True$
$\theta_n = \Gamma_n = \Psi_n = \{hp1, hp2\}$, where set $\{hp1, hp2\}$ is randomly selected from HP_{all}
$O_n(\theta) \equiv [f_{\text{perf}(\theta_n)}, f_{\text{eng}(\theta_n)}, f_{\text{size}(\theta_n)}]$
$D_{\text{perf}} = \emptyset$: Set for storing all selected HPs for estimating f_{perf}
$D_{\text{eng}} = \emptyset$: Set for storing all selected HPs for estimating f_{eng}
$D_{\text{size}} = \emptyset$: Set for storing all selected HPs for estimating f_{size}
$D_{\text{IntPar}} = \emptyset$: Set for storing all selected HPs for estimating IntPar (intermediate Pareto front)

========== **LEVEL 1** ==========

$D_{\text{perf}} = D_{\text{perf}} \cup \theta_n$, $D_{\text{eng}} = D_{\text{eng}} \cup \Gamma_n$, $D_{\text{size}} = D_{\text{size}} \cup \Psi_n$.
Posterior Gaussian distributions:
$\tilde{f_{\text{perf}}} = p(\tilde{f_{\text{perf}}}|(f_{\text{perf}}, D_{\text{perf}}))$, $\widetilde{f_{\text{eng}}} = p(\tilde{f_{\text{eng}}}|(f_{\text{eng}}, D_{\text{eng}}))$, $\tilde{f_{\text{size}}} = p(\tilde{f_{\text{size}}}|(f_{\text{size}}, D_{\text{size}}))$
while $flag$ do
    Calculate $AF_n(\tilde{f_{\text{perf}}})$, $AF_n(\tilde{f_{\text{eng}}})$, $AF_n(\tilde{f_{\text{size}}})$
    $\theta_{n+1} = \underset{HP_{\text{all}}}{\text{argmax}}\, AF_n(\tilde{f_{\text{perf}}})$, $\Gamma_{n+1} = \underset{HP_{\text{all}}}{\text{argmax}}\, AF_n(\tilde{f_{\text{eng}}})$, $\Psi_{n+1} = \underset{HP_{\text{all}}}{\text{argmax}}\, AF_n(\tilde{f_{\text{size}}})$
    if $\theta_{n+1} = \theta_n$, and $\Gamma_{n+1} = \Gamma_n$, and $\Psi_{n+1} = \Psi_n$:
        $flag \leftarrow False$
    else:
        $D_{\text{perf}} = D_{\text{perf}} \cup \theta_{n+1}$, $D_{\text{eng}} = D_{\text{eng}} \cup \Gamma_{n+1}$, $D_{\text{size}} = D_{\text{size}} \cup \Psi_{n+1}$.
        Evaluate $O_{n+1}(\theta)$, $O_{n+1}(\Gamma)$, $O_{n+1}(\Psi)$

========== **LEVEL 2** ==========

        calculate $\beta_n = \{\forall\, i \in I_K |\quad \beta_n^i\}$
            (where $O_n(\beta)$ are non-dominant points. Please note K maybe different in each iteration)
        $D_{\text{IntPar}} = D_{\text{IntPar}} \cup \beta_n$
        calculate $O_{n,norm}(\beta)$, (by normalizing each element of $O_n(\beta)$ to $[0, 1]$)
        $\text{IntPar}_n = \{\forall\, i \in I_K |\quad \text{IntPar}_n^i = ||O_{n,norm}^i||_1\}$
        $\tilde{\text{IntPar}}_n = p(\tilde{\text{IntPar}}_n|(\text{IntPar}, D_{\text{IntPar}}))$
        Calculate $AF_n(\tilde{\text{IntPar}})$
        $\beta_{n+1} = \underset{HP_{\text{all}}}{\text{argmax}}\, AF_n(\tilde{\text{IntPar}})$ (Next best data set to move the current Pareto the corner)
        $D_{\text{perf}} = D_{\text{perf}} \cup \beta_{n+1}$, $D_{\text{eng}} = D_{\text{eng}} \cup \beta_{n+1}$, $D_{\text{size}} = D_{\text{size}} \cup \beta_{n+1}$.
        Evaluate $O_{n+1}(\beta)$
        $n \leftarrow n + 1$
        update $\tilde{f_{\text{perf}}}$, $\tilde{f_{\text{eng}}}$, $\tilde{f_{\text{size}}}$, $\tilde{\text{IntPar}}$.

4. PABO FOR NON-SPIKING NEUROMORPHIC SYSTEMS

In this chapter we use PABO as hyperparameter optimization framework to maximize accuracy of a traditional ANN while simultaneously minimizing the energy requirements of a memristive crossbar-based underlying accelerator. The experiment and its corresponding results are published in [25]. Details of the neural network architectures, an overview on the underlying accelerator, and how to estimate an abstract energy consumption for this accelerator are given in this section. This is then followed by the experimental setup, and results for three different case studies.

4.1 Artificial Neural Network Architecture

Throughout this chapter we present artificial neural network (ANN) architecture with the following notation: $-$ for dividing layers, c for convolution layers, p for pooling layers, and fc for fully connected layers. For example $128 \times 128 \times 3 - 12c5 - 2p - 10o$ is a four-layer ANN with $128 \times 128 \times 2$ input followed by 12 convolution filters with size 5, a 2×2 pooling layer, and finally 10 output neurons. Details of the AlexNet [31], and VGG19 [32], architectures for flower17 [34] and CIFAR-10 [37] datasets are given in Table 4.1, respectively. This table only shows a sample architecture for AlexNet, and VGG19, and the architectures are modified based on the hyperparameters given in the experimental setup for ANN.

4.2 Baseline Accelerator Overview

This section provides an overview of a memristive crossbar-based accelerator, shown in Figure 4.1. Typical memristive accelerators employ a spatial architecture,

Table 4.1.: Details of ANN architectures used in the PABO for non-spiking neuromorphic systems experiments

Name	Architecture
AlexNet	$227 \times 227 \times 3 - 96c5 - p3 - 256c3 - p3 - 384c3 - 384c3 - 256c3 - p3 - 4096fc$ $-4096fc - 17fc$
VGG19	$32 \times 32 \times 2 - 64c3 - 64c3 - p2 - 128c3 - 128c3 - p2 - 256c3 - 256c3 - 256c3$ $-256c3 - p2 - 512c3 - 512c3 - 512c3 - 512c3 - p2 - 4096fc - 4096fc - 1000fc$

where the DNN is executed by mapping the model across the on-chip crossbar storage in a spatial manner [7]. This is because the memristive devices have high storage density, but are limited by the high write cost. Consequently, the high storage density enables mapping DNNs spatially in practical die sizes while alleviating the high write cost which would be required if a crossbar was reused for different parts of the model in a time-multiplexed fashion. At the lowest level, N Matrix Vector Multiplication Units (MVMUs) are grouped into a single core. Each MVMU is composed of multiple crossbars and performs a 16-bit 128×128 matrix-vector multiplication. Note that multiple crossbars are needed to store high precision data required for DNN inference, since typical memristive crossbars store low-precision data such as 2-bits [5, 7]. At the next level, M cores are grouped into a single tile with access to a shared memory, which enables data movement between cores (inter and intra tile). At the highest level, T tiles are connected via a network-on-chip that enables data movement between tiles within a single node. For large scale applications, multiple nodes can be connected using suitable chip-to-chip interconnect.

4.3 PUMA Energy Consumption

We use an abstract energy consumption model to evaluate the efficiency for PABO, where we consider the energy consumption of the MVMUs only. First, the abstract model enables evaluating the impact of hyperparameter optimization while isolating the benefits obtained from microarchitectural techniques. This isolation enables widespread applicability where DNNs optimized with PABO can be executed over

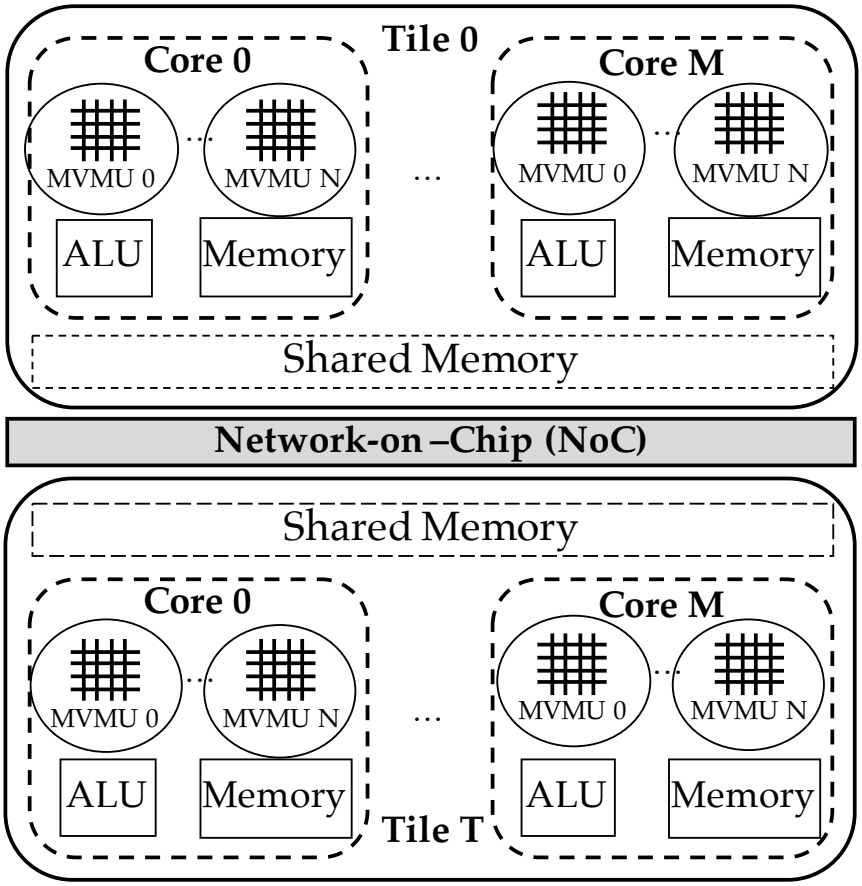

Fig. 4.1.: High-level overview of PUMA [7] hybrid accelerator architecture

a wide range of memristive accelerators, where each accelerator may be leveraging different dataflow, compute to control granularity, etc. Second, while a typical memristive accelerator expends significant energy in shared memory, network on chip and chip-chip interconnect due to the data movements in a spatial architecture, reducing the number of MVMU operations typically reduces the total energy consumption commensurately [109].

A layer (fully connected or convolution layer) is partitioned into smaller blocks of size N×N to fit a MVMU (sized N×N). Each layer will map across multiple MVMUs that may span multiple cores and multiple tiles (see Figure 4.1). Further, a MVMU may be used multiple times (once) for an input in a convolution layer (fully connected layer) due to weight-sharing. Hence, the number of MVMU operations required to

execute an inference of deep neural network will depend on the several HPs such as the number of layers, the number of extracted feature in each convolution layers, and the kernel sizes in the network architecture (Equations 4.1, and 4.2).

$$num\_xbar\_c_i = d_i \times d_i \times \lceil \frac{nc_i \times k_i \times k_i}{xs} \rceil \times \lceil \frac{nc_{i+1}}{xs} \rceil \tag{4.1}$$

$$num\_xbar\_f_i = \lceil \frac{nf_i}{xs} \rceil \times \lceil \frac{nf_{i+1}}{xs} \rceil \tag{4.2}$$

In these equations, $num\_xbar\_c_i$, and $num\_xbar\_f_i$ are number of crossbars for the i_{th} convolution layer and the fully connected layers, respectively. d_i is the dimension of the output, nc_i is the number of input features for convolution layer i. Similarly, nf_i is the number of input features for the fully connected layer i. k_i is the kernel size in i_{th} convolution layer, and xs is the crossbar size. The term d_i in Equation 4.1 is for inherent weight-sharing property of convolution layers.

Typically, each memristive operation is followed by vector linear, vector non-linear and data movement operations [7]. Consequently, the number of MVMU operations is proportional to the overall energy consumption and can be used as a metric of computational cost on hardware. We calculate the total energy consumption in each convolution and fully connected layer based on the number of crossbar operations using the following equations. In our selected memristive crossbar accelerator, a 16-bit (inputs and weights) crossbar operation (size 128×128) consumes $\simeq$44nJ energy. epx is the energy per matrix vector multiplication operation. The sum of energy consumption for all the convolution and fully connected layers is then used to calculate the total energy consumption of the memristive crossbar accelerator (Equation 4.3).

$$tot\_eng\_t = (\sum_i num\_xbar\_c_i + \sum_i num\_xbar\_f_i) \times epx \tag{4.3}$$

In this experiment, we used Equations 4.1 through 4.3 to calculate the total hardware energy consumption for each combination of HP.

Table 4.2.: Evaluated parameters for three different case studies for using PABO on ANN with PUMA as underlying hardware. ANN's accuracy, and PUMA's energy consumption were the two objectives we optimized in these case studies

	Case study one	Case study two		Case study three
Dropout	0.4, 0.5	0.5	Dropout, Layer 1	0.3, 0.4
Learning Rate	0.001	0.001, 0.01	Learning Rate	0.01, 0.1
Momentum	0.85, 0.9, 0.95	-	Learning Rate Decay	$1e-6$, $1e-4$
Optimizer	Momentum	Momentum, Adam	Weight Decay	0.0005, 0.05
# of FC Layers	2, 3	2, 3	Kernel Size, Layer 6	3, 5
# of Conv. Layers	4, 5	3, 4, 5	Kernel Size, Layer 7	3, 5
Kernel Size, Layer 1	5, 7	3, 5, 7	Kernel Size, Layer 8	3, 5
Kernel Size, Layer 2	3, 5	3, 5	Kernel Size, Layer 9	3, 5, 7
Kernel Size, Layer 3	3, 5		# of Features, Layer 1	64, 128
Kernel Size, Layer 4	3	3, 5	# of Features, Layer 2	128, 256
			# of Features, Layer 4	256, 512
Architecture	AlexNet	AlexNet		VGG19
Neural Accelerator	PUMA	PUMA		PUMA
Dataset	Flower17	Flower17		CIFAR10
Search Space	192	288		3072

4.4 Experimental Setup and Results

We performed several case studies for different types of hyperparameters, including the number of layers, kernel sizes, number of features to extract in each layer, and also the values for learning rate, momentum, and dropout.

Table 4.2 shows a summary of the selected ranges for the hyperparameters (HPs) for three different case studies. All these cases are studied with PUMA [7] as the underlying hardware. Case study one is designed with a small search space of size 192 HPs. We begin with the small search space size in order to estimate the actual Pareto frontier of the problem with a grid search technique and to compare the PABO result with other state-of-the-art approaches. Case study two is included to capture the effects of different types of HPs in the analysis, and case study three is a more realistic experiment with VGG19 as the chosen architecture on CIFAR10 dataset.

4.4.1 Single-Objective Optimization

Before presenting the joint optimization results, it is imperative to answer the question: Why one cannot rely on single objective optimization to minimize the hardware energy consumption with maximum neural network accuracy? Figure 4.2 demonstrates the limitations of using independent single objective HP optimization techniques to separately design a neural network with optimum performance and a hardware with minimum energy requirements.

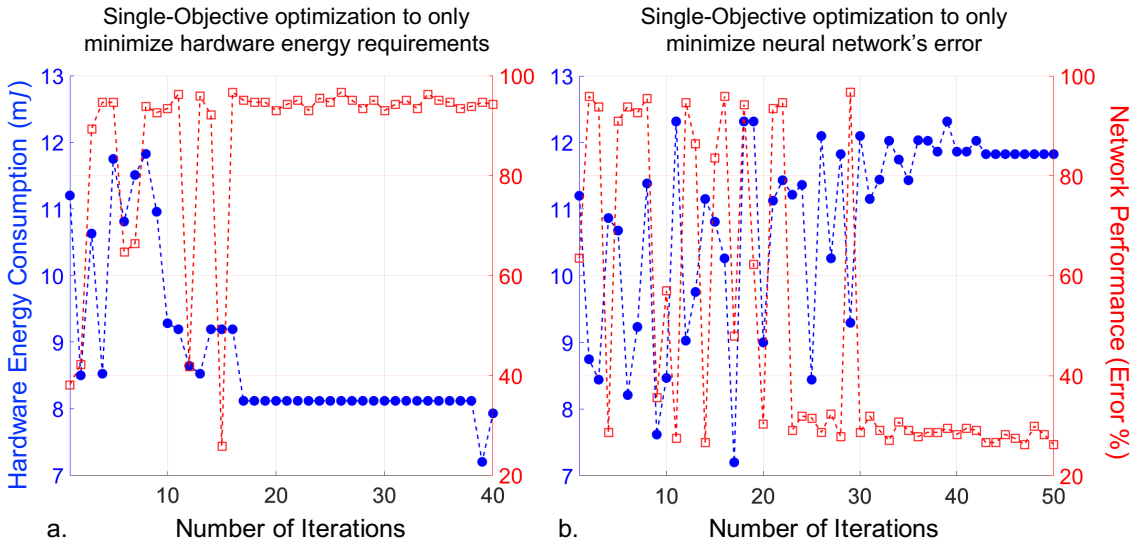

Fig. 4.2.: PABO for ANN on PUMA for case study two, single-objective optimization results for Table 4.2, obtained using SKOPT [89] python Bayesian optimization package. a. Optimizing HPs for hardware energy consumption only. b. Optimizing HPs for ANN's performance only.

Figure 4.2a shows that designing an energy efficient hardware without optimizing the network accuracy leads to significant decrease in the network performance (large error). The selected HP set is reported after 40 evaluations of hardware energy consumption. For this HP, the minimum energy is ∼7.8mJ, while the DNN's error is ∼92%. Similarly, in Figure 4.2b the network performance, in terms of reducing error, is optimized without considering the energy consumption of the underlying hardware. The inefficient hardware design is evident as the reported minimum error region occurs

at high hardware energy consumption. Both of these results are undesirable, and are the main reasons to seek a multi-objective approach to find HPs that minimizes DNN's error while designing an energy-efficient hardware. We used SKOPT [89] python package to solve these single-objective Bayesian optimization for AlexNet on Flower17 dataset with HPs given in Table 4.2, case study 2.

4.4.2 Multi-Objective Optimization (PABO)

We used the proposed PABO algorithm to find the optimum ANN accuracy while minimizing the underlying memristive crossbar accelerator energy consumption on three case study. In case study 1, we used AlexNet network with the Flower17 dataset with a small search space of 192 HPs. The range of HPs are provided in Table 1. In this case, we intentionally selected a small search space, so that we can estimate the actual Pareto frontier using the grid search method. Figure 4.3 shows the results for case study 1. In this figure, PABO's result compared with grid search, random search, and state-of-the-art NSGA-II (Non-Dominated Sorting Genetic Algorithm) [43]. Red triangles, blue dots, black squares and gray crosses correspond to PABO, random search, NSGA-II, and grid search, respectively. Each point in the figure corresponds to one evaluation of the noted techniques.

With only 17 evaluations (out of 192 possible sets of HPs), PABO estimates the Pareto frontier (red dash line in Figure 4.3) for the HPs within $\sim$1-2% percent of the actual Pareto set (gray line in Figure 4.3) obtained using the grid search method. Compared to the NSGA-II approach, PABO not only estimates Pareto frontier more accurately, but is also 92$\times$ faster. A comparison between the execution time for different techniques is shown in Figure 4.4.

In Table 4.3, to further illustrate the impact of HPs, we summarized them for the points A, B, C and D that are shown in Figure 4.3. Point A belongs to the Pareto frontier of the network at which we obtained the optimum DNN performance and hardware energy requirement. Point B corresponds to an HP set with minimized

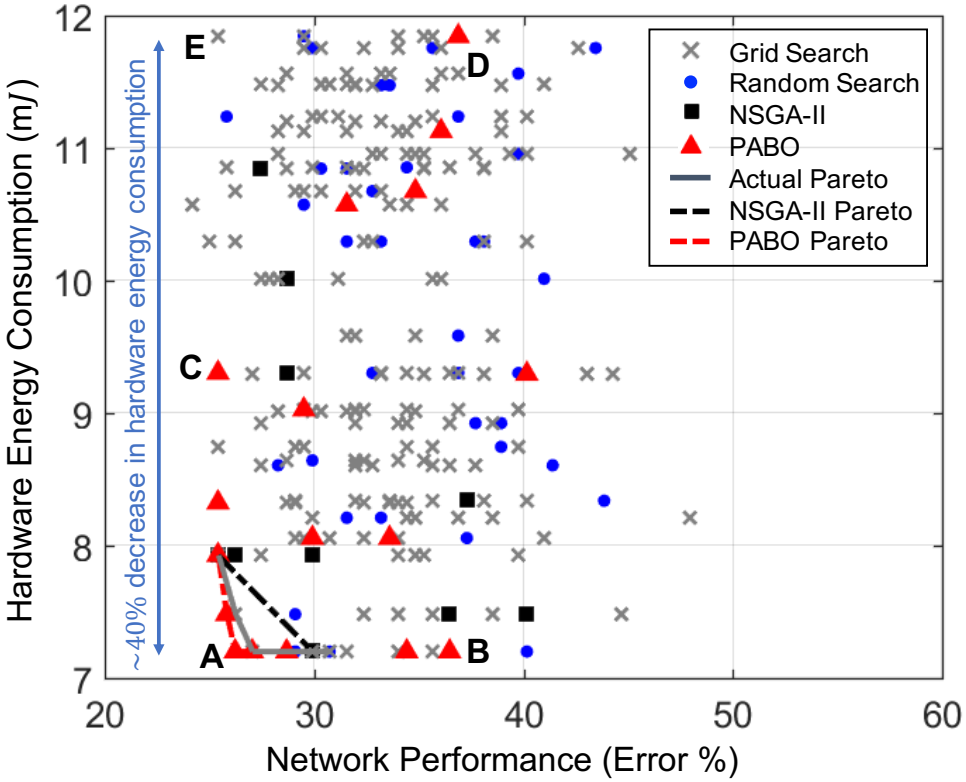

Fig. 4.3.: PABO for ANN on PUMA for case study one: AlexNet on Flower17 dataset with 192 possible set of HPs. Comparison between grid search for all HP combinations (grey cross), random search with evaluating 40 different sets of HPs (blue dots), NSGA-II with population size of 10 and maximum generation of 50 (black squares), and PABO (red triangles). The red dashed line, gray line and the black dashed line are the Pareto frontiers obtained by PABO, grid search and NSGA-II approaches.

energy requirement for hardware, while producing a sub-optimal DNN design. At point C, HPs result in minimum DNN error but with an inefficient hardware design, and the corresponding HP at point D neither optimizes the DNN performance nor hardware energy consumption. It is clear from Table 4.3, that using a joint optimization approach is indispensable for optimal design of both the DNN and the hardware. Moreover, in this case study, selecting HPs given in point A (from Table 4.3) results in up to 40% decrease in energy requirements for the memristive crossbar accelerator compared to the case where DNN is not optimized for the hardware architecture design (point E shown in Figure 4.3).

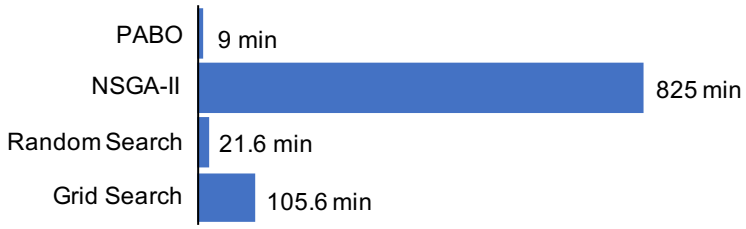

Fig. 4.4.: PABO for ANN on PUMA for case study one: execution time for PABO is 92x faster than state-of-the-art NSGA-II technique on one Nvidia GeForce RTX 2080 Ti TU102 GPU with 11 GB of memory.

Table 4.3.: PABO for ANN on PUMA for case study one, hyperparameter analysis

Hyperparameter	**A**	**B**	**C**	**D**
Dropout	0.5	0.4	0.5	0.4
Learning rate	0.001	0.001	0.001	0.001
Momentum	0.95	0.85	0.95	0.9
Batch size	64	64	64	64
# of FC layers	2	2	2	3
# of conv. layers	4	4	5	5
Kernel size, layer 1	5	5	7	7
Kernel size, layer 2	3	3	3	5
Kernel size, layer 3	3	3	5	5
Kernel size, layer 4	3	3	3	3

Figures 4.5 and 4.6 show the results for two additional case studies with more realistic choices of HPs. Details of the HP selections are given in Table 4.2. Figure 4.5 is a case study with 6912 choices of HP combinations on AlexNet architecture with Flower17 dataset. PABO results are shown in red triangles and compared with random search with blue dots and NSGA-II with black squares. Random search is performed with 40 evaluations, NSGA-II had population size of 20 with maximum generation of 100. PABO approximates the Pareto frontier with only 33 function evaluations (where each evaluation corresponds to training the DNN and computing the hardware energy consumption). Compared to 6000 function evaluations of NSGA-II technique, this leads to 183× faster execution time. Note that the results obtained for NSGA-II are the best we could get with the same computational resources we

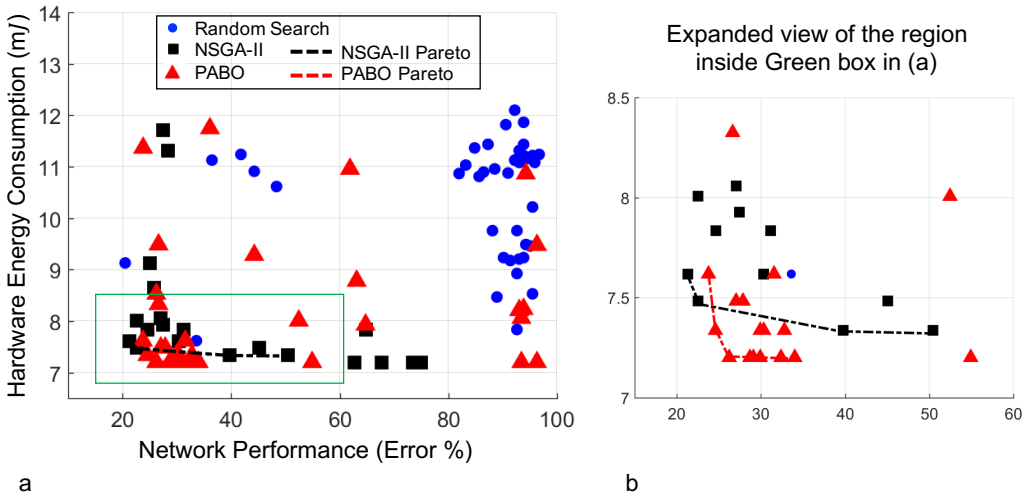

Fig. 4.5.: PABO for ANN on PUMA for case study two: a. Comparison between PABO, NSGA-II, Grid Search and Random Search for AlexNet on Flower17 dataset with 6912 different set of HP combination. b. The expanded view of the region inside the green box in panel (a).

Fig. 4.6.: PABO for ANN on PUMA for case study three: VGG19 network on CIFAR-10 dataset with 3072 different set of HP combination. Other methods cannot be run on this case study due to significant computational requirements.

used to perform PABO and other methods. We may improve the NSGA-II's results using supercomputers and significantly longer hours of computation.

In Figure 4.6, we used the VGG19 network on CIFAR-10 data with 3072 HPs combinations listed in Table 4.2. Since the network is significantly larger than AlexNet, it is computationally prohibiting to perform NSGA-II using our computational resources. On the other hand, PABO was able to estimate the Pareto frontier with only 22 evaluations.

4.5 Discussions

We proposed a novel pseudo agent-based multi-objective hyperparameter optimization technique, deemed PABO, that can maximize the neural network performance while minimizing the energy requirements of the underlying hardware. PABO uses Bayesian optimization with Gaussian processes and acquisition function along with a supervisor agent, to estimate the Pareto frontier that shows the optimum HP sets for maximum neural network performance and minimum hardware energy consumption [25].

We tested PABO on both AlexNet [31], and VGG19 [32] neural network with an underlying memristive crossbar accelerator [7], and compared it with other algorithms. Superior performance of our method both in terms of accuracy and computational time was demonstrated. 100x increase in computational speed compared to the NSGA-II algorithm was demonstrated. It is important to note that PABO is not limited to a specific neural network or hardware architecture, and can be applied to multiple (more than two) black-box functions corresponding to different design requirements.

5. HIERARCHICAL-PABO FOR EVOLUTIONARY-BASED SPIKING NEUROMORPHIC SYSTEMS

In this chapter we use Hierarchical-PABO as hyperparameter optimization framework for an evolutionary-based training algorithm (EONS [14]), using two different underlying hardware (DANNA2 [19], and mrDANNA [33]) on several control and classification tasks. Pole-balance [41, 110], and RoboNav [42] were the two selected control applications. Pole-balance is a control benchmark in engineering which involves a pole connected to a cart through a joint that allows single axis movement. The goal of this control application is to keep the pole from falling by moving the cart either direction. RoboNav is an autonomous navigation system for robotic applications and is meant to be deployed on a specific robot [42]. We also used the Iris [38] and Radio [39] datasets for classification tasks. The former is a multivariate dataset of 50 samples from each of three species of the Iris flower, and the latter is a satellite radio signal classification problem.

We demonstrate the effectiveness of Hierarchical-PABO first as single-objective optimization problem to maximize accuracy of EONS only and then we continue with three-objective optimization problem of maximizing accuracy and minimizing network size and energy requirement of the underlying hardware. In these experiments for the Kernel function in Hierarchical-PABO, we chose *Matern* covariance function with the Kernel function shown in Equation 5.1.

$$C_\nu(d) = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu} \frac{d}{\rho} \right)^\nu K_\nu \left(\sqrt{2\nu} \frac{d}{\rho} \right) \tag{5.1}$$

where d is a distance function, Γ is the gamma function, K_ν is the modified Bessel function of the second kind, ρ and ν are positive parameters. For this paper,

we found out the fastest most accurate HP optimization results are obtained when $\rho = 1$, and $\nu = 1.5$. We selected Matern kernel function [111] due to smooth Gaussian distribution estimate it provides. This results are summarized in [27, 28]. Details of the EONS training algorithm, the input encoding module, an overview on the underlying accelerators, and how to estimate an abstract energy consumption for a accelerator is given in this section. This is then followed by the experimental setup and results for several case studies.

5.1 Introduction

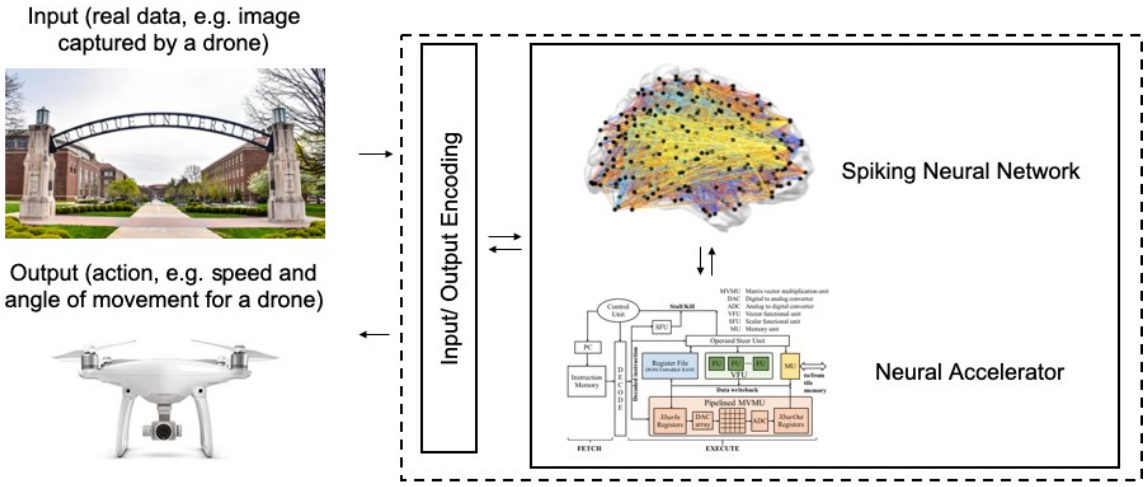

Fig. 5.1.: High-level Overview of a Spiking Neuromorphic Computing System

As shown in Figure 5.1, a spiking neuromorphic computing systems consists of two key building blocks, namely a spiking neural network (SNN) and an underlying neuromorphic hardware. There are different techniques available to train an SNN [14, 16, 18]. In this section we only focus on EONS [14]. As for the spike-based neuromorphic systems there have been various CMOS-based [19–21] and beyond-CMOS [7] neuromorphic accelerators introduced in literature. Each of these building blocks have inherent hyperparameters that directly affect the final performance of the system, such as accuracy, energy efficiency, inference time, and network size. While

one set of hyperparameters might satisfy one or all performance metrics, a minor change can drastically alter the resulting performance.

Neuromorphic devices and architectures run only on spikes; however, most of the time the real-world applications communicate their states through values, and only accept values as their input. There are several input/output coding schemes such as direct, binning [71], rate coding [112], and temporal coding [113]. In this work we considered a combination of several coding techniques that can handle a wide range of data within a short time period for real-time online applications (specially for edge devices) [71].

We consider various types of hyperparameters ranging from input encoding to SNN's training and neuromorphic hardware implementation. We perform sensitivity analysis on various hyperparameter sets and demonstrate how critical some sets of hyperparameters are, directly impacting the performance of the system. We further analyze different input encoding schemes and show how combining multiple schemes might boost the performance of the system.

5.1.1 EONS: Evolutionary Optimization for Neuromorphic Systems

EONS trains a graph representation of a spiking neural network for a neuromorphic hardware system using Genetic Algorithms (GA). This graph representation of a neuromorphic network generates an initial population of random graphs based on the application inputs and hardware specifics. Next generations are produced by duplicating, merging, mutation or crossover on current generations based on their fitness values. The resulting generations are converted to spiking neural networks, and the new performance values are calculated. The process repeats until the desired fitness is reached or terminated due to the running time [14, 114]. Neurons and synapses are the building blocks of this graph and are inherent to the device characteristics. Synaptic weights and delays ranges, neurons thresholds, as well as their leaky rates or

plasticity parameters are defined by the neuromorphic hardware and are also targeted in our proposed optimization framework.

In this work we use TENNLab's framework [114] for input encoding, EONS, and neuromorphic hardware hyperparameter (HP) optimization to find the optimum set of HPs that maximizes the overall performance of the system. We considered the fitness value (accuracy), number of synapses (size), and neuromorphic hardware energy consumption as the performance matrices of the system; however, this can be easily modified to any other metric such as inference time.

5.1.2 Input/Output Coding Module

The TENNLab framework facilitates value to spike encoding and vice-versa through an encoding/decoding module. This module accepts different state-of-the-art input/output coding schemes such as direct, binning [71], rate coding [112], and temporal coding [113]. In this work we considered a combination of several coding techniques that can handle a wide range of data within a short time period for real-time online applications (specially for edge devices) [71].

- **Binning**. Encoding an input with single spike in a single time step, through creating multiple neurons and spiking on a particular neuron based on the value of the input.
- **Spike-count**. Converting input values to a number of spikes at a fixed rate.
- **Charge-injection**. Injecting a specific charge value into a neuron rather than a fixed magnitude spike.
- **Combined coding schemes**. Combining all above techniques in a hierarchy by introducing three different inter-bin functions of *simple, flip-flop, and triangle*. *Simple* inter-bin function linearly maps the values between the preassigned charge values for each bin, whereas in *flip-flop*, this mapping is only for odd-numbered bins and then flips for even-numbered ones. This function enables continuous mapping

across the bins. Finally, the *triangle* inter-bin function creates overlapped bins to encourage smooth mapping.

Details of the above input/output coding schemes are given in [71]. Each of these methods have inherent hyperparameters that directly impact the performance of the system. Here our main focus is finding the optimum set of input encoding hyperparameters to maximize the performance of spiking neuromorphic system for a specific application and hardware.

5.1.3 Neuromorphic Hardware

We use two different neuromorphic implementations that are already deployed in the TENNLab framework, a fully digital neuromorphic processor, DANNA2 [19], and a memristive mixed-signal neuromorphic processor, mrDANNA, [33]. DANNA2 is a fully digital programmable device with integrate-and-fire neurons and synapses that can be deployed either on an FPGA or a custom chip, and mrDANNA is a mixed analog-digital programmable device with metal-oxide memristors.

We use mrDANNA for the case studies where we would like to minimize energy requirement of the underlying neuromorphic hardware. Table 5.1 summarizes the energy estimate per spike for this neuromorphic device. mrDANNA is a synchronous neuromorphic architecture and is simulated in a discrete event simulation. Events in the simulation include accumulations, fires, and learning. The energy estimates for each event type are given in Table 5.1 and we track how many of each type of event occurs in the simulation and sum up the energies. If no event is occurring on a neuron or synapse in a clock cycle, that neuron or synapse is "idle", but still performing some operations that contribute to idle cost. We use these energy estimates to estimate the overall energy cost of running on a particular application.

Table 5.1.: Energy estimate per spike for mrDANNA

	Accumulation	Fire	Learning	Idle
Neuron	$9.81pJ$	$12.5pJ$	-	$7.2pJ$
Synapse	$1.45pJ$	-	$2.58pJ$	$0.07pJ$

Table 5.2.: Evaluated parameters for six different case studies for using Hierarchical-PABO on EONS with DANNA2 and mrDANNA as underlying hardware.

Hyperparameters	Case Study 1	Case Study 2	Case Studies 3,4	Case Studies 5,6
b_k	1, 2, 4, 8	2, ..., 8	2, 4, 8	2, 4, 8, 10, 12
p_k	1, 2, 4, 8	1, ..., 12	4, 8	2, 4, 8, 10, 12
$[c_k, C_k]$	[0,0.5],[0,1], [0.25,0.5], [0.25,1], [0.5,0.5],[1,1]	[0,0.5],[0,1], [0.25,0.5],[0.25,1], [0.5,0.5],[1,1]	[0,1], [0.5,0.5], [1,1]	[0,0.5],[0,1], [0.25,0.5], [0.25,1], [0.5,0.5],[1,1]
Function	Simple Flip-flop Triangle	Simple Flip-flop Triangle	Simple Flip-flop	Simple Flip-flop Triangle
Interval	1	1, ..., 5	0, 1	0, 1, 2
Population size	1000	600, 800, 1000, 1200, 1500, 2000	10, 100, 500	10, 100, 500, 700
Mutation rate	0.9	0.6, 0.7, 0.8, 0.9	0.2, 0.6, 0.9	0.2, 0.6, 0.9
Crossover rate	0.5	0.3, 0.4, 0.5, 0.6, 0.7	0.3, 0.5, 0.9	0.3, 0.5, 0.9
Synaptic weight	[-255,255]	[-127,127],[-255, 255] [-511, 511],[-1023, 1023]	-	-
Neuron threshold	[0,1023]	255, 511, 1023	-	-
Synaptic delay	127	15, 31, 63, 127, 255	-	-
Neural Accelerator	DANNA2	DANNA2	mrDANNA	mrDANNA
Application	Pole-balance	Pole-balance	3: IRIS, 4:Radio	5: IRIS, 6: Radio
Search Space	**240**	**54,432,000**	**1458**	**35,640**
Objective	**Accuracy**	**Accuracy**	**Accuracy Energy Size**	**Accuracy Energy Size**

5.2 Experimental Setup and Results

Table 5.2 shows a summary of the selected ranges for the hyperparameters (HPs) for case studies in SNN domain using EONS as the underlying training algorithm. In this table, b_k, p_k, $[c_k, C_k]$, *function*, and *interval* are from the input encoding module, *population size*, *mutation rate*, and *crossover rate* are for EONS evolutionary-based training algorithm, and *synaptic weight*, *neuron threshold*, and *synaptic delay* belong to the underlying neuromorphic hardware. The input encoding hyperparameters include several approaches such as *binning-based*, using b_k as the number of bins required

for each input values, *spike-count* with p_k as the maximum number of spikes to encode a single input value, *charge-value* with $[c_k, C_k]$ on injecting a specific charge to fire a neuron, *function* on how to map the values to spikes, and *interval* to define the interval between pulses. For more details on each of these hyperparameters please refer to [26, 71].

We first show the importance of hyperparameter optimization for spiking neuromorphic systems by only focusing on single-objective optimization (performance of the system on the task) problem, where grid search results are already available by [71]. We then continue with Hierarchical-PABO (H-PABO) results for a three-objective optimization problem (performance, energy, and network size).

5.2.1 Single-Objective Optimization

While H-PABO is generally aimed for multi-objective problems, it can easily be reduced to a single-objective optimization by setting objective functions to one. This is the case for case studies one and two in Table 5.2, where we are only optimizing a single objective function that is the accuracy of the neural network.

For the first case study, we start with finding the optimum set of HP for the input encoding hyperparameters of an SNN. For this problem, a grid search technique has previously been applied for all HP combination settings and thus, the optimum HP set is known [71]. In Table 5.2, case study one shows the possible values for different HPs. The ranges are all based on reasonable and acceptable values for each of the HPs and the combinations that do not make sense are removed from the search space.

Figure 5.2 shows box plot figures with interquartile ranges. The grid search result is produced and published by [71] and shown in Figure 5.2a. For each one of the 240 combinations of the hyperparameters, the network accuracy is calculated and evaluated for 100 times. In Figure 5.2b, we used H-PABO for the same experiment, and with only 40 hyperparameter combinations, each repeated for 10 times, we are able to predict not only the exact optimum set of hyperparameter, but also predict

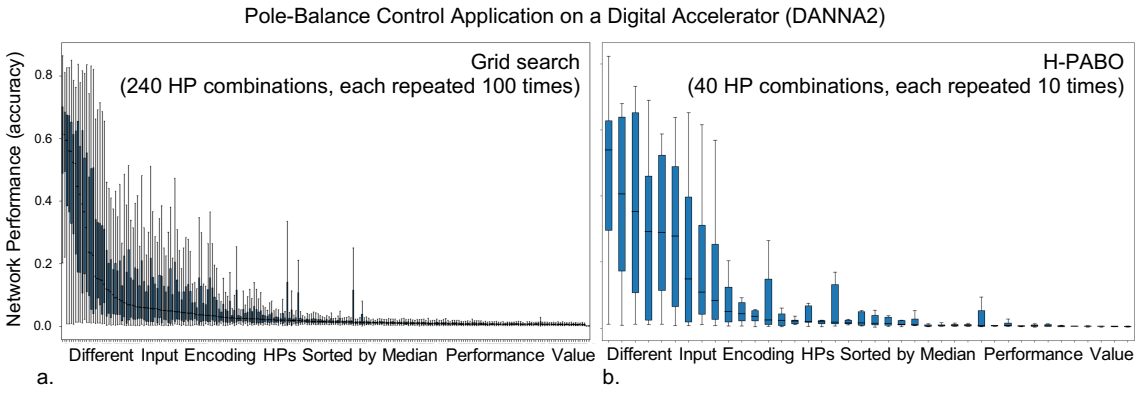

Fig. 5.2.: Hierarchical-PABO for EONS on DANNA2 for case study one: Comparing grid search with HP optimization for problem with HP combinations shown in Table 5.2, case study one. a. Grid search: 100 runs for each of the valid 240 different HP sets according to [71]. b. Bayesian-based HP optimization: 10 runs for selected 40 HP combinations. Both techniques report the same optimum HP set ($b_k = 2$, $p_k = 8$, $charge = [0, 0.5]$, $function = flip - flop$) with median fitness value of 52% (Reproduced with permission from [26]).

the same trend in the network accuracy changes for different hyperparameter combinations [26]. In this case study the optimum hyperparameter combination leads to median value of 52%.

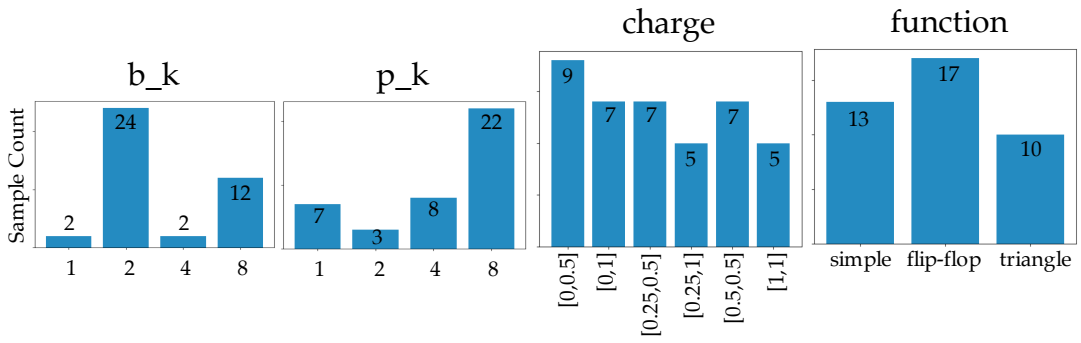

Fig. 5.3.: Hierarchical-PABO for EONS on DANNA2 for case study one: Histogram of the HP combinations for 40 evaluations.

The iterative Bayesian-based HP optimization finds the optimum set of HPs by exploring and exploiting the search space. This means that it not only maintains the HP combinations that creates the highest fitness values, but also explores the search

space to avoid trapping in local minima. The frequency of selecting the value for each hyperparameter for case study one is shown in Figure 5.3. The hyperparameter value with maximum number of calls is consistent with the optimum HP set defined by the optimization technique ($b_k = 2, p_k = 8, charge = [0, 0.5], function = flip - flop$).

In Table 5.2 case study two, we considered three types of HP combinations: input encoding related, EONS, or hardware-related hyperparameters. In this experiment the total number of hyperparameter combinations is $54,432,000$. This shows how drastically the number of hyperparameter sets increase in real problems where there are multiple HPs involved in different modules, frameworks, algorithm, and architectures. In Figure 5.4 we plotted the median fitness value for 50 HP combinations sets (50 iterations of Bayesian optimization) each repeated for 100 times. It is evident from the figure that the method ensures obtaining the optimum HP set through exploring the search space by evaluating HP combinations with low fitness values that were chosen outside the predicted range of optimum HPs, while performing exploitation by maintaining the predicted optimum HPs in overall higher fitness value domains.

Please see the discussion Section 5.3 for further analysis and observations on this results.

5.2.2 Multi-Objective Optimization

To validate Hierarchical-PABO technique for multi-objective hyperparameter optimization problems in SNN domain, we focus on classification application with IRIS [38], and Radio [101] dataset on both digital [19] and mixed-signal memristive [33] neuromorphic devices using EONS [14] as the core training algorithm. The summary of the case studies three to six, and their corresponding HP ranges are given in Table 5.2.

Figure 5.5 demonstrates the Hierarchical-PABO (H-PABO) results using EONS training algorithm on IRIS classification dataset with a mixed-signal underlying hardware (mrDANNA [33]). Figure 5.5a shows the H-PABO results compared to grid

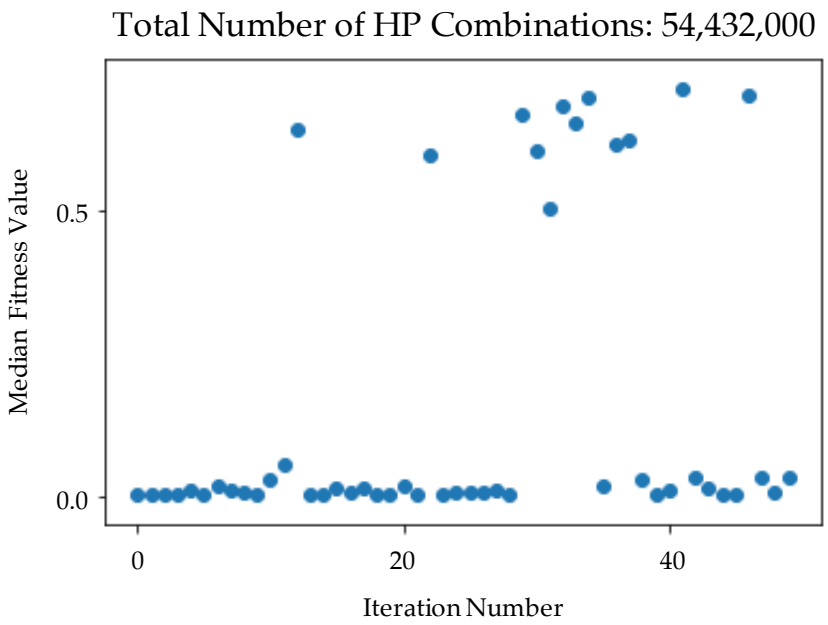

Fig. 5.4.: Hierarchical-PABO for EONS on DANNA2 for case study two: Median fitness value after 100 runs with 50 different HP evaluations for the parameters given in Table 5.2. The optimum set of HP combination in this case study is shown in Table 5.4 with median fitness value of 70.99%

search for the case study three given in Table 5.2 with 1458 different sets of HPs. Each point in the three-dimension figure represents network performance, hardware energy consumption, and number of required synapses for a set of HP combination. The number of required synapses increases as the color becomes lighter. The grid search results show that most of the time the energy consumption increases as the number of synapses increase (the top left region of Figure 5.5a). However, we might also have a larger network with more inhibitory synapses, for example, that would have less activity and thus less energy than a smaller network (top right region). The triangles are the H-PABO search points, and as expected, all different regions of the search space are explored with H-PABO. The H-PABO Pareto points are shown with squares. These points are calculated once the H-PABO search process is completed and are the H-PABO search points that belong to the Pareto frontier. As shown

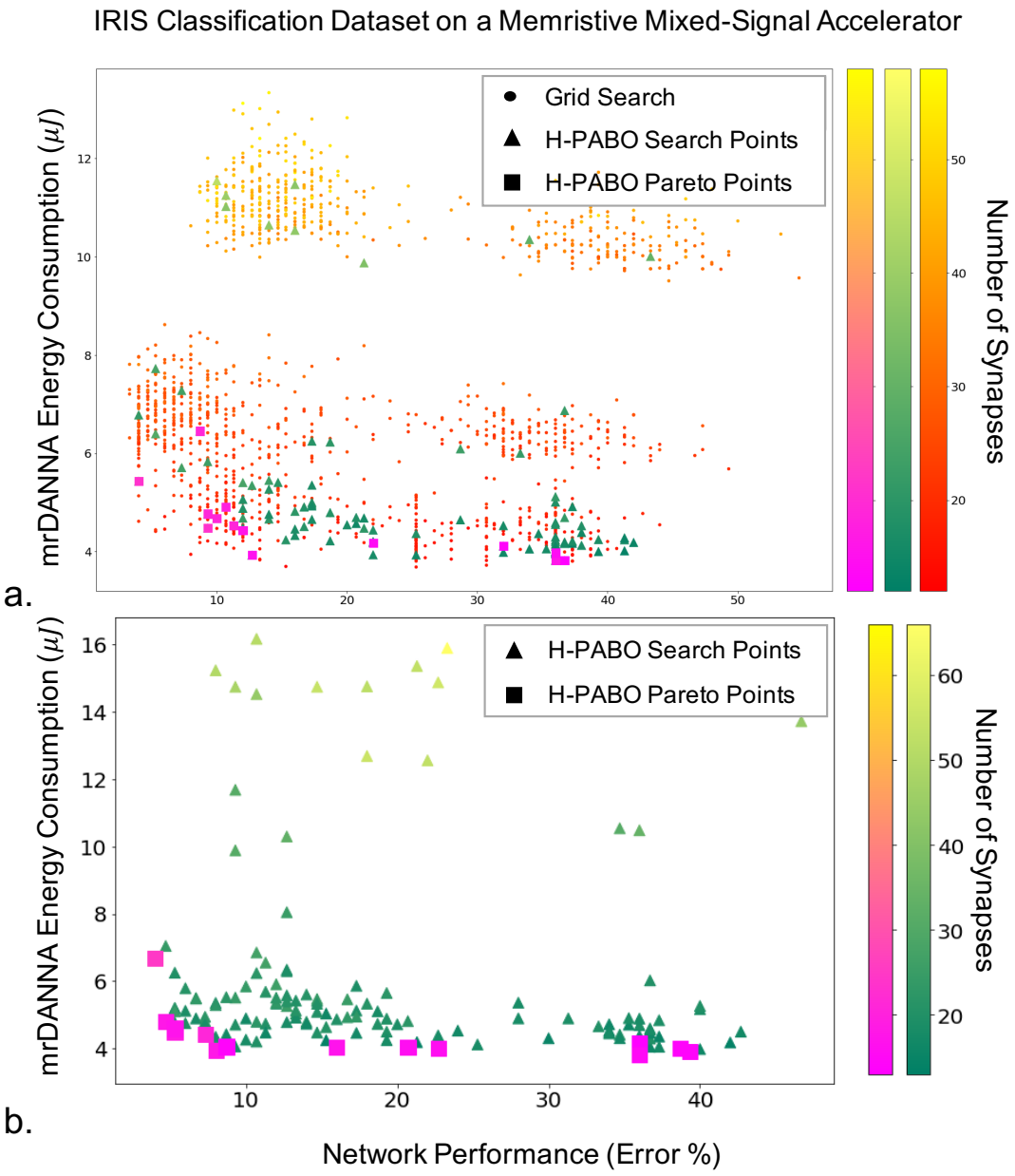

Fig. 5.5.: Hierarchical-PABO for EONS on mrDANNA for three-objective hyper-parameter optimization (network performance, hardware energy consumption, and number of synapses) for Iris classification dataset on mrDANNA with HP search space of a. 1458, case study three, b. 35640, case study five in Table 5.2

in Figure 5.5a this calculated Pareto frontier is within close proximity to the actual Pareto frontier of the problem.

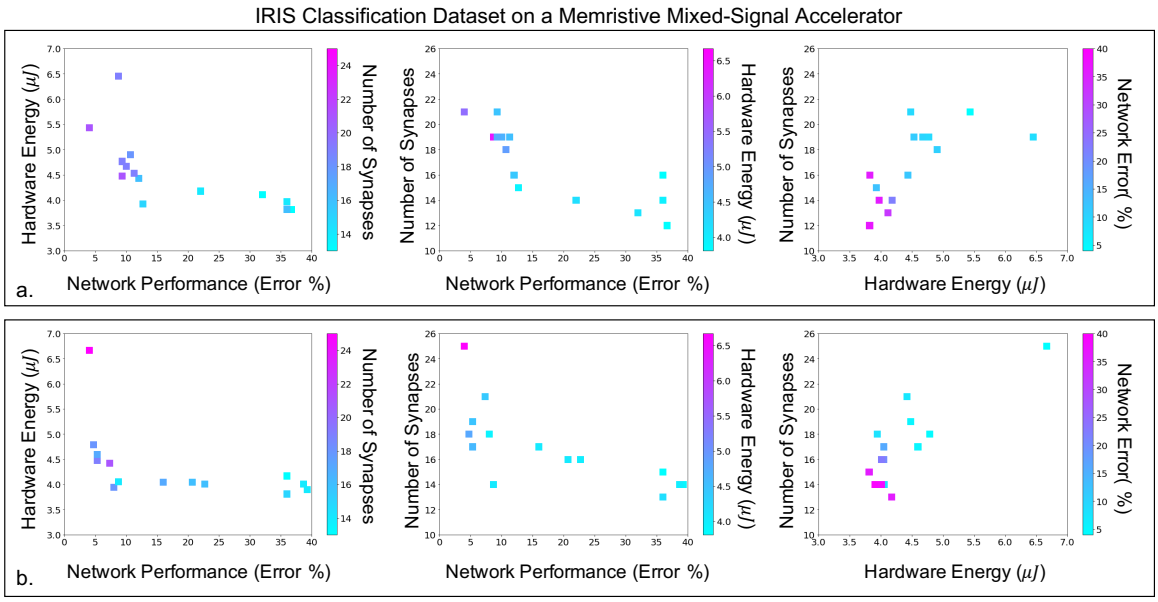

Fig. 5.6.: Hierarchical-PABO for EONS on mrDANNA: Comparing three-dimensional results, pairwise for a. case study three with search space size 1458, b. case study five with search space size 35640.

Figure 5.5b shows the H-PABO results for case study five in Table 5.2 for the HP search space of 35640 different HP combinations. Once again, we see that all regions of the search space are explored by the H-PABO approach, but that the majority of the H-PABO points are evaluated are in the region of interest and near the H-PABO Pareto front. In this case, H-PABO was able to find well-performing networks with desired characteristics (low energy consumption and relatively few synapses) with significantly fewer evaluates than what would be required for a full grid search of 35640 points. It is also worth noting that by optimizing over the additional HPs, the H-PABO approach is able to find well-performing networks with better characteristics than the networks found simply optimizing over the smaller set of HPs (shown in Figure 5.5b).

Figures 5.6 shows the H-PABO results from Figure 5.5, but splits the results into three different pairwise comparison plots, for each case study, to show how the different objectives play off of each other. The third objective is also shown in each

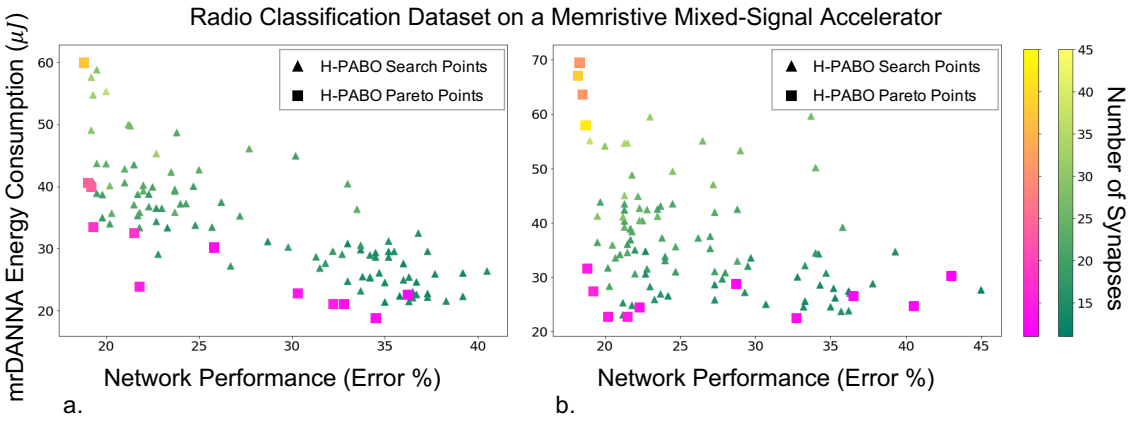

Fig. 5.7.: Hierarchical-PABO for EONS on mrDANNA for three-objective hyper-parameter optimization (network performance, hardware energy consumption, and number of synapses) for case studies four and six in Table 5.2, Radio classification dataset on mrDANNA, a. search space size 1458, b. search space size 35640

Table 5.3.: Observation Three. Hierarchical-PABO for EONS. Evaluated parameters for best and worst networks for isolated HP optimization analysis

	Best Network	**Worst Network**
Input Encoding HPs	$b_k = 2$ $p_k = 8$ $[c_k, C_k] = [0, 0.5]$ function = flip-flop Interval = 1	$b_k = 8$ $p_k = 1$ $[c_k, C_k] = [0.5, 0.5]$ function = simple Interval = 1
EONS HPs	Population size = 1000 Mutation rate = 0.9 Crossover rate = 0.6	Population size = 1000 Mutation rate = 0.6 Crossover rate = 0.3
Hardware HPs	Synaptic Weights = $[-511, 511]$ Neuron Thresholds = $[0, 1023]$ Synaptic Delay = 127	Synaptic Weights = $[-1023, 1023]$ Neuron Thresholds = $[0, 255]$ Synaptic Delay = 15

plot through the color of the squares. With these plots, we can see the different Pareto fronts for each of the pairwise objectives. For example, in the network performance vs. hardware energy plots, we can see that there are trade-offs in energy usage in order to achieve lower error (and similarly for network performance vs. number of synapses). However, the number of synapses and energy usage are relatively correlated, such that fewer synapses typically corresponds to a lower energy value.

Figure 5.7 gives the results for case studies four and six, in which the H-PABO approach is applied to HP optimization for the Radio classification dataset on the memristive mixed-signal system (mrDANNA). The two case studies look at the same HP combination sets as the Iris dataset and correspond to 1458 and 35640 combinations, respectively. As shown in the figure, H-PABO once again explores the space of potential solutions but is able to find a Pareto front in relatively few evaluations. Again, similar to the result for the Iris dataset, by expanding our HP set to the 35640 potential HP combinations, H-PABO is able to achieve overall better performing networks (lower error and energy and fewer synapses required), and in general moving the Pareto front closer to the desired region.

5.3 Discussions

Figure 5.8 demonstrates a partial dependence plot for non-categorical hyperparameters in case study four. Inter-bin function types and population size are considered as categorical HPs and therefore not shown in this plot. The black dots are the set of parameters we have evaluated and the red dot is the best parameter we found. In these plots colors are the surrogate model built by the Gaussian Process. Light regions are the best (highest fitness values) while the darker ones are the worst. To make each of these partial dependence plots, we make a grid for a set of two parameters. We then calculate the surrogate model with fix values for those two parameters while generating random values for all other parameters. We repeat the process and average the fitness values and plot the color map. These plots are only used for extra analysis on the optimization search process and are not deterministic due to the inherent variability of the built surrogate models for Gaussian distribution with random parameter selections. In this figure for some HPs the counter plots show the convergence toward the optimum values. For example for p_k versus *synaptic_delay* partial dependence plot it is clear that the higher the value for both parameters is the better performance. However for all partial dependence plot for parameter b_k, as

Table 5.4.: Hierarchical-PABO for EONS. Sensitivity analysis for SOO

	Hyperparameters	Experiment 1	Experiment 2	Experiment 3	Experiment 4
Input Encoding HPs	b_k	2	2	2	2
	p_k	8	12	8	8
	charge	$[0, 0.5]$	$[0, 0.5]$	$[0, 0.5]$	$[0, 0.5]$
	function	flip-flop	flip-flop	flip-flop	flip-flop
	interval	1	5	1	2
EONS HPs	population size	1000	1500	400	1000
	mutation rate	0.9	0.9	0.9	0.9
	crossover rate	0.5	0.4	0.5	0.7
Accelerator HPs	synp weight	$[-255, 255]$	$[-127, 127]$	$[-255, 255]$	-
	neuron thrshld	$[0, 1023]$	$[0, 1023]$	$[0, 1023]$	-
	synp delay	127	255	15	
Neuromorphic System Performance		52%	70.99%	50%	53%

long as the optimum set stays in any of the light regions, it does not matter what combination we choose for the next iteration.

Different hyperparameters have different impact on the final performance of the neuromorphic system. Figure 5.9 demonstrates how critical it is to select the optimum set of input encoding and hardware hyperparameters to obtain the maximum fitness value. In both cases the worst HP combination lead to almost 0% fitness value. However, EONS hyperparameters such as crossover and mutation rates have less impact on the final performance of the system. This shows resiliency of EONS framework. Details of the HP sets in each network is given in Table 5.3.

In Table 5.4, a sensitivity analysis is performed for Hierarchical-PABO single objective optimization (SOO) for different classification applications (Pole-balance for Experiments 1 and 2, and RoboNav for Experiments 3 and 4) on two different neural accelerators (i.e. DANNA2 [19] for Experiments 1 to 3, and mrDANNA [33] for Experiment 4). These experiments show how sensitive is pole-balance control application to the changes of hyperparameters. If we only change few hyperparameters (all in reasonable ranges), the resulting accuracy will change from 52% to 70.99% (comparing experiments 1 and 2 in Table 5.4). Based on these experiments, RoboNav appears to be less sensitive to changes in hyperparameters and architectures, but more extensive experiments may be required in order to understand the full impact on this particular application.

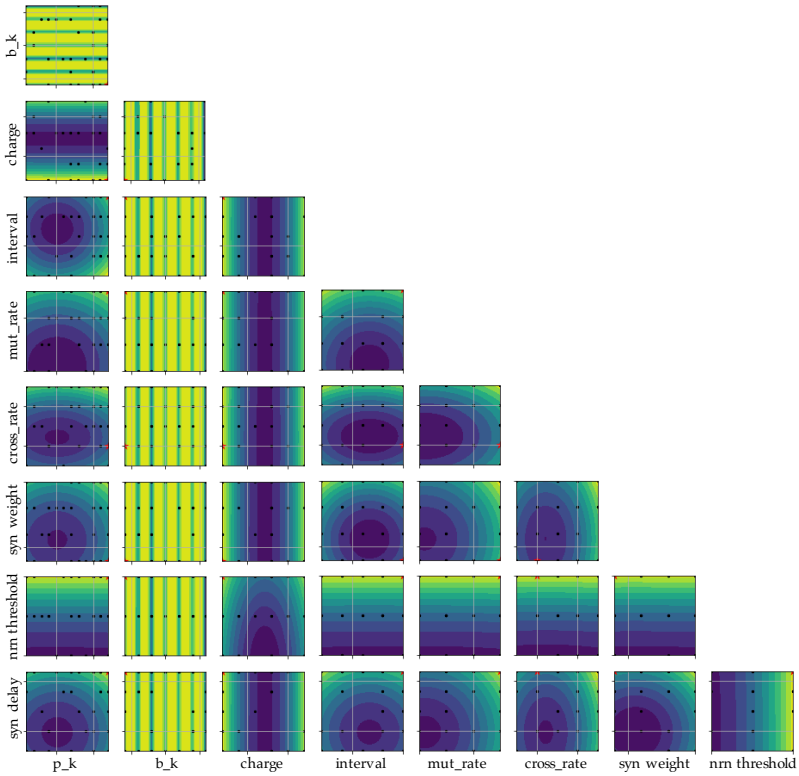

Fig. 5.8.: Hierarchical-PABO for EONS. Partial dependence plot for case study two in Table 5.2

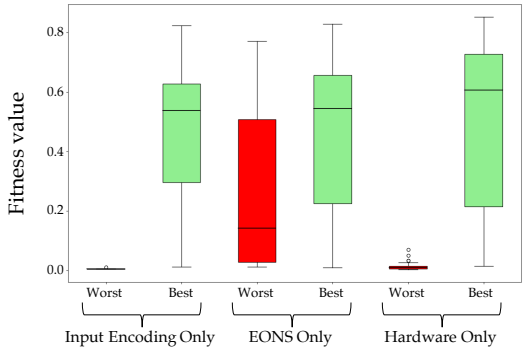

Fig. 5.9.: Hierarchical-PABO for EONS. Sensitivity analysis on different types of HPs. Comparing the fitness values for best and worst HP combinations

Neuromorphic computing systems provide a potential solution for energy efficient machine learning. However, there are many aspects of spiking neuromorphic computing systems that are not well understood, including how to input information into

the spiking neuromorphic computing system, how to train associated spiking neural networks for these systems, and hardware details themselves. In this work, we demonstrate a Bayesian-based hyperparameter optimization approach for neuromorphic computing systems. We show that this approach can discover the appropriate hyperparameters for input encoding for neuromorphic systems in many fewer iterations than a grid search. We also show that selecting the appropriate hyperparameters can have a tremendous impact on application performance.

6. HIERARCHICAL-PABO FOR BINARY NEUROMORPHIC SYSTEMS

In this section we introduce Whetstone [56], as binary neuromorphic system, to our Hierarchical-PABO framework with the goal of pushing its limit to the full capacity with optimized hyperparameters. After a brief introduction, we describe the Whetstone [56] training algorithm, and then continue with the experimental setup and results. Results of this work are published in [27].

6.1 Introduction

Whetstone trains networks that have binary communication, which are amenable for mapping onto spiking neuromorphic hardware. In this approach, neural networks are trained initially with differentiable activation functions (e.g., sigmoidal or bounded rectified linear units), but over the course of gradient descent optimization, the activation functions are slowly "sharpened" to non-differentiable threshold functions. This approach not only has all of the hyperparameters associated with traditional neural network or deep learning network training, but also additional hyperparameters of its own, for example, associated with how sharpening occurs over the course of the algorithm. As we will show below, these hyperparameters can have a significant effect on the performance of the algorithm, but it is not clear what hyperparameters to use for a given dataset *a priori*.

In this work, we apply Hierarchical-PABO for single-objective optimization problem (accuracy only) to find optimal hyperparameters for the Whetstone algorithm on four different datasets. We compare our results to the previously published Whetstone results from [56] and show that by tuning the hyperparameters for each dataset we can achieve significantly better performance, up to a 15% improvement in accuracy

in some cases. We compare the best performing hyperparameters for each dataset, and study the sensitivity of the final performance on the changes of hyperparameters. These results represent, not just an improvement over state-of-the-art, but also an indication that off-the-shelf spiking algorithms may be significantly improved by optimization via this Bayesian approach.

6.2 Whetstone

Whetstone utilizes bounded rectified linear units (bRELUs) and sigmoidal units that are modified during training to approach binarized step-functions. The approach aims to gradually modify the activation function so as to minimally otherwise disrupt network training. Due to the sensitivity of backpropagation to zeroed activations, this sharpening and thus binary conversion process was found to be more stable when applied layer-by-layer on a schedule and in the direction of input layer to output layer. This *scheduled-sharpening* involves several hyperparameters, such as the epoch to start the sharpening, duration of sharpening, and number of epochs to wait before starting the next scheduled sharpening (intermission). To avoid a fully manual schedule with additional hyperparameters, Whetstone's authors introduced an *adaptive-sharpening* scheduler that monitors loss after each training epoch and decides to resume or pause sharpening dependent on the relative change in training loss(fixed to 15% in [16]). If sharpening is currently on, the rate of sharpening is dependent on a duration hyperparameter but is adjusted gradually over each batch in the epoch. Initial sharpening epoch and sharpening pause duration are additional hyperparameters to be chosen.

Whetstone also attempts to mitigate a condition which occurs in bRELUs and sigmoidal nodes that stop responding and produce zero outputs regardless of input. The authors note that this condition happens in non-binarized networks as well but hypothesize that the sharpening process can increase occurrence odds. To alleviate this problem, Whetstone networks typically use redundant output encodings as out-

put targets. To produce output for loss computation, Whetstone uses a softmax over a population encoding (neuron distribution key generated or specified at network initialization) that allows for n-hot encoding of targets while output neurons can contribute to more than one class. This also enables the use of a cross-entropy loss function (common to many neural network classification tasks), which the authors found to be more effective than a direct mean squared error vector loss.

Severa *et al.* [56] also demonstrate the effects of architecture hyperparameters such as number of convolution layers and filter sizes on the overall performance of Whetstone for four different dataset. Their results for these various hyperparameters were consistent with the intuition that deeper networks perform better for spiking networks. However, they did not perform any comprehensive hyperparameter optimization. Additional instability was noted in relation to the choice of optimizer used during training, with Adam optimized networks' performance being especially sensitive to initial conditions. For the choice of optimizer, they show that Adadelta and RMSprop are more reliable compared to Adam. Batch normalization was further found to improve stability during training. The sensitivity of Whetstone approach on various hyperparameters such as the choice of optimizer or batch normalization layer, differentiates the hyperparameter optimization approach for this binary communication from traditional artificial neural network training. This leads to a research question on which hyperparameter optimization technique is suitable for non-traditional networks such as Whetstone.

In this work, we only focus on *scheduled-sharpening* due to the stability and consistency of the results obtained with this scheduler. In our hyperparameter optimization search, we considered three main hyperparameters involved in this technique: sharpener starting epoch ("sh_st"), duration ("sh_du"), and intermission ("sh_int"). For each case study, detailed of the ranges for each of these hyperparameters is given in the following section.

6.3 Experimental Setup and Results

We validate our Bayesian hyperparameter optimization approach across several datasets, hyperparameter combinations and case studies. In using Whetstone, there are a variety of sets of hyperparameters that can be optimized. Here we focus on the following hyperparameter sets: optimizer parameters, noise parameters, batch normalization parameters, Whetstone sharpener parameters, and CNN architecture parameters. The Whetstone's scheduled sharpener sharpens layers one at a time in sequential order. The "start epoch" hyperparameter is the epoch on which it begins sharpening the first layer. The "duration" is how many epochs it takes to sharpen each layer, and the "intermission" is how many epochs it waits after sharpening a layer before beginning sharpening of the next layer. Details of the hyperparameters that are optimized and their corresponding ranges are given in each case study as follows.

Our methods were benchmarked on four labeled image data sets commonly used to demonstrate efficacy of supervised image classification protocols. The *MNIST [35]* dataset consists of gray-scale images of handwritten single digits, each 28×28 pixels. There are 10 classes, one for each number $0 - 9$, and the data is split in to a training set of 60000 images and a test set of 10000 images. The *Fashion-MNIST [36]* dataset consists of gray-scale images of miscellaneous clothing items (shirts, pants, shoes, etc.), each 28×28 pixels. There are 10 classes, one for each type of item, and the data is split in to a training set of 60000 images and a test set of 10000 images. The Fashion-MNIST dataset is designed to be a drop in replacement for the MNIST dataset, with the only difference being the items which are classified. The *CIFAR-10 [37]* dataset consists of color images of miscellaneous items (dogs, airplanes, birds, ships, etc.), each 32×32 pixels. There are 10 classes, one for each type of item, and the data is split in to a training set of 50000 images and a test set of 10000 images. The *CIFAR-100* [37] dataset is the same as the CIFAR-10 dataset, except with 100 classes. Each class represents an equal proportion of the total dataset.

Table 6.1.: Hierarchical-PABO for Whetstone: Evaluated hyperparameters

Hyperparameter		Case Study One	Case Study Two
Optimizer	Learning rate	0.0001, 1	0.0001, 0.001, 0.01, 0.1, 1
	Rho	0.9	0.9, 0.95
	Epsilon	1e-6	1e-8, 1e-6
	Decay	1e-8, 1e-6	1e-8, 1e-6
	Type	Adadelta	Adadelta, RMSprop
Noise	Standard deviation	-	0.2, 0.3
	Location	Without noise	Without noise, After 1st dense
Batch Normalizer	Momentum, conv.	0.95	0.85, 0.95
	Momentum, dense	0.95	0.85, 0.95
	Epsilon	1e-3	1e-3, 1e-2
	Center	True	False, True
	Scale	True	False, True
Sharpener Schedule	Start Epoch	15, 25	20, 25, 30
	Duration	3, 7	4, 5, 6, 7
	Intermission	2, 5	1, 2, 3, 4, 5
CNN Architecture	Conv. layer 1, filter size	3, 7	3, 5, 7
	Conv. layer 2, filter size	5	3, 5
	Conv. layer 3, filter size	3	3, 5
	Conv. layer 1, # of features	64, 128	32, 64, 128
	Conv. layer 2, # of features	256	64, 128, 256
	Conv. layer 3, # of features	512	256, 512
	Dense layer, # of features	256, 1024	256, 512, 1024
Search space size		**256**	**398,131,200**

In this experiment, we designed two different case studies with the hyperparameters given in Table 6.1. For case study one, we select a small search space for classification task on CIFAR-100 dataset [37]. This limited search space is helpful in validating the results through comparing the optimum hyperparameters from the optimization technique and the grid search approach. The grid search approach is evaluating the network for all possible combinations of the hyperparameters.

In Figure 6.1, for CIFAR-100 dataset, the grid search results are compared with the results from the Bayesian hyperparameter search. The hyperparameter ranges are given in Table 6.1, case study one. After only 15 evaluations of Whetstone [56], the Bayesian hyperparameter search finds the almost optimum combination of hyperparameters that the grid search predicts after 256 evaluations. This optimal point for the Bayesian search, $(l\_r = 1, dec = 1e-6, sh\_st = 25, sh\_du = 7, sh\_int = 2, filter1 = 3, feat1 = 128, dense = 1024)$, is shown in red star in Figure 6.1, and leads to accu-

racy of 53.13%, which outperforms the 38% accuracy reported in Whetstone original results [56]. The optimum hyperparameter set for the grid search is ($l\_r = 1, dec = 1e-6, sh\_st = 25, sh\_du = 3, sh\_int = 5, filter1 = 3, feat1 = 128, dense = 1024$) with accuracy of 53.34%. These two points predict almost the same classification accuracy and only differ in two hyperparameters of "duration of sharpening", and "sharpening intermission".

In case study two, we increase the search space size to 398,131,200 combinations of hyperparameters shown in Table 6.1. In this scenario we consider various hyperparameter types ranging from optimizer hyperparameters, to Gaussian noise, or batch normalization layers. In addition we also include the Whetstone scheduled sharpening [56] hyperparameters, and the hyperparameters that belong to the neural network architecture itself, such as filter sizes or the number of features to extract.

For the hyperparameters given in Table 6.1, the performance of the hyperparameter optimization approach for Whetstone technique for four different dataset of MNIST [35], Fashion-MNIST [36], CIFAR-10 [37], and CIFAR-100 [37] as well as their corresponding optimum hyperparameter values are given in Table 6.2. For each dataset, the Whetstone network is trained for 50 epochs and the hyperparameter optimization search evaluated the network for 30 different hyperparameter sets. The Whetstone performance once its hyperparameters are optimized is increased from 99.53% to 99.6% for MNIST, and from 93.2% to 93.68% for Fashion-MNIST dataset. This improved performance is more noticeable for more complex dataset such as CIFAR-10 and CIFAR-100. For the former, the accuracy is increased from 79% to 84.36, and for the latter it is improved from 38% to 53.42%.

Figure 6.2 demonstrates the exploration and exploitation capability of Bayesian optimization technique in finding the optimum set of hyperparameter for each dataset. Starting from two random sets of hyperparameters, the search technique not only exploits and leverages the sets of hyperparameters with decent performance, but also explores the search space. In Figure 6.3, we show the frequency of selecting each value for some of the hyperparameters given in Table 6.2 for CIFAR-100. The x-axis

Table 6.2.: Hierarchical-PABO for Whetstone, case study two: Optimized hyperparameters and their corresponding classification accuracies for different dataset

Dataset		MNIST	Fashion-MNIST	CIFAR-10	CIFAR-100
Optimizer Hyperparameters	Learning Rate	0.001	0.001	0.001	1
	Rho	0.95	0.9	0.9	0.9
	Epsilon	1e-6	1e-6	1e-8	1e-6
	Decay	1e-8	1e-6	1e-6	1e-6
	Type	RMSprop	RMSprop	RMSprop	Adadelta
Noise Layer Hyperparameters	Standard deviation	-	0.2	-	-
	Location	No Noise	After 1st Dense	No Noise	No Noise
Batch Normalizer Hyperparameters	Momentum, conv.	0.95	0.95	0.85	0.95
	Momentum, dense	0.95	0.85	0.95	0.95
	Epsilon	1e-2	1e-2	1e-3	1e-3
	Center	False	False	True	False
	Scale	False	False	False	False
Whetstone Sharpener Schedule Hyperparameters	Start Epoch	30	20	30	30
	Duration	6	4	4	4
	Intermission	4	5	2	5
CNN Architecture Hyperparameters	Conv. layer 1, filter size	7	3	3	3
	Conv. layer 2, filter size	5	3	5	5
	Conv. layer 3, filter size	3	5	5	5
	Conv. layer 1, # of features	128	128	64	128
	Conv. layer 2, # of features	128	128	256	256
	Conv. layer 3, # of features	256	512	512	512
	Dense layer, # of features	256	512	512	1024
Accuracy		**99.6%**	**93.68%**	**83%**	**53.42%**

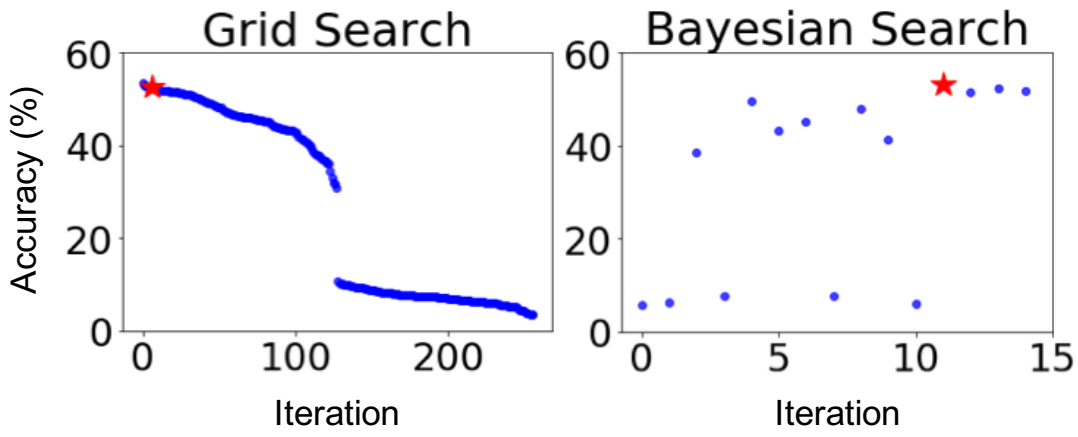

Fig. 6.1.: Hierarchical-PABO for Whetstone case study one: Comparing grid search and Bayesian hyperparameter optimization for hyperparameters given in Table 6.1 with search space size of 256

is the choice of hyperparameter and the y-axis is the number of times that a specific choice is called within the 30 evaluations in the Bayesian optimization search. The optimum hyperparameter values are highlighted in red rectangles in the figure. This

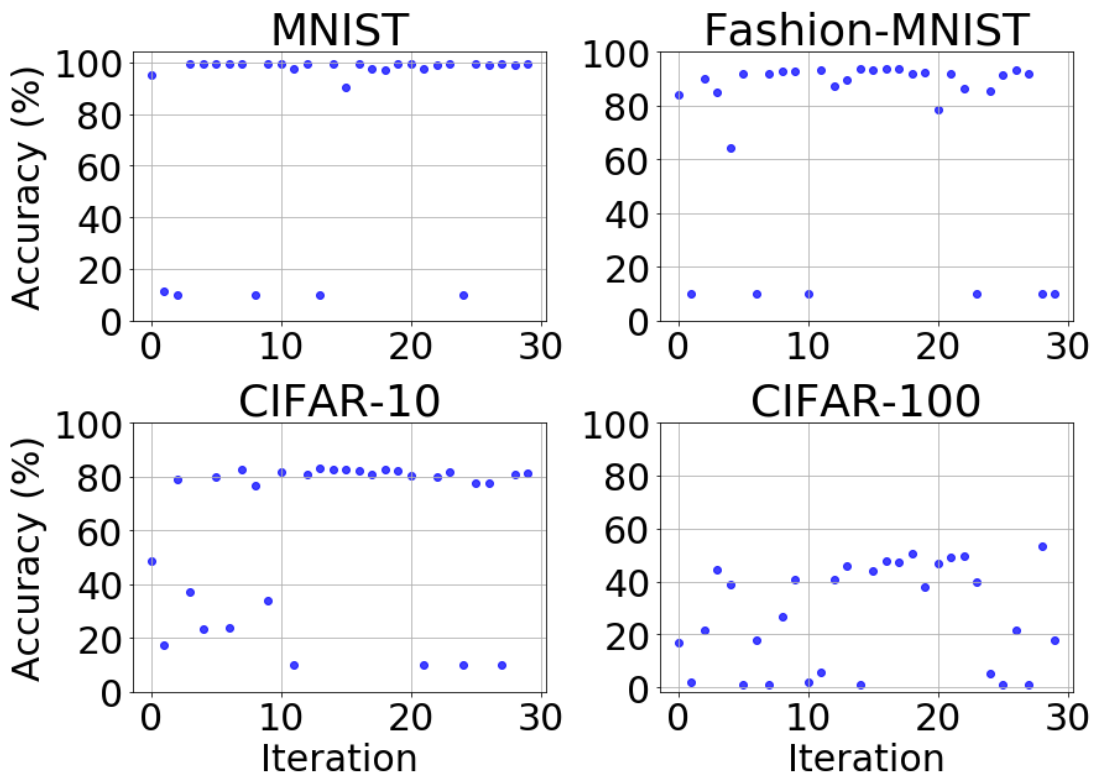

Fig. 6.2.: Hierarchical-PABO for Whetstone case study two: Performance value (accuracy (%) for each hyperparameter optimization search iteration for MNIST, Fashion-MNIST, CIFAR-10, and CIFAR-100 dataset with optimum hyperparameters given in Table 6.2

also shows that after 30 iterations for searching the optimum hyperparameter set, the Bayesian framework not only leans toward the optimum values by selecting them most, but also tries all possible hyperparameter values to avoid trapping in any local minimum.

6.4 Discussions

We perform further analysis on the changes of hyperparameters and their effect on the final accuracy of the network. The hyperparameter values at each iteration in case study one in Table 6.1 are given in Table 6.3. For example, with changing the sharpener starting epoch from 15 to 25, its duration from 3 to 7, and the filter size

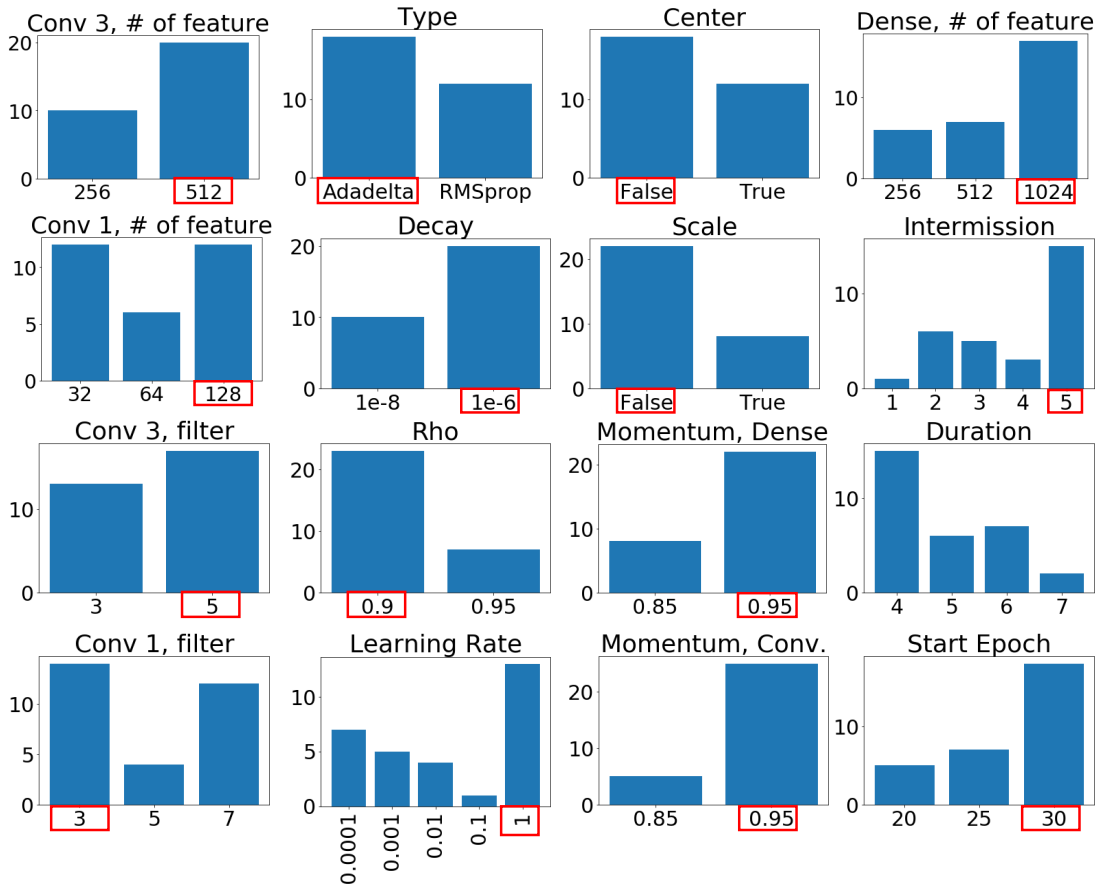

Fig. 6.3.: Hierarchical-PABO for Whetstone, case study two: Histograms of each hyperparameter value for CIFAR-100 dataset experiment for the 30 iterations of the optimization search process

in the first convolution layer from 7 to 3, we are able to improve the final accuracy from 38.65% to 53.13% (iteration 3 versus iteration 13 in Table 6.3). This table also shows that some hyperparameters play a vital role on the final performance of the system, such as learning rate.

Table 6.4 gives a comprehensive sensitivity analysis on changing hyperparameter values and observing the final performance of the spiking neural network for CIFAR-100 dataset with the hyperparameter values given in Table 6.1 in case study two, and the performances shown in Figure 6.2 for this dataset. These experiments are chosen among the 30 iterations of the Bayesian optimization search. The first three experiments in Table 6.4 show that with quite different combinations of hyperparam-

Table 6.3.: Hierarchical-PABO for Whetstone, case study one: Details of the Bayesian search direction

HPs	Iter 1	Iter 2	Iter 3	Iter 4	Iter 5	Iter 6	Iter 7	Iter 8	Iter 9	Iter 10	Iter 11	**Iter 12**	Iter 13	Iter 14	Iter 15
lr	1e-4	1e-4	1	1e-4	1	1	1	1e-4	1	1	1e-4	**1**	1	1	1
dec	1e-8	1e-6	1e-6	1e-8	1e-6	1e-6	1e-6	1e-6	1e-6	1e-6	1e-6	**1e-6**	1e-8	1e-6	1e-6
sh_st	25	25	15	15	25	25	25	25	25	25	25	**25**	25	25	25
sh_dur	3	7	3	3	7	7	3	3	7	3	7	**7**	7	7	7
sh_int	2	2	2	5	5	5	5	2	2	5	2	**2**	5	5	2
filter 1	3	7	7	3	3	7	7	3	7	7	3	**3**	3	3	3
feat 1	64	128	128	64	64	64	64	128	128	64	128	**128**	64	128	64
dense	256	256	1024	1024	256	256	1024	1024	1024	256	256	**1024**	1024	1024	1024
Acc (%)	**5.61**	**6.37**	**38.65**	**7.69**	**49.69**	**43.4**	**45.19**	**7.59**	**47.84**	**41.34**	**6.11**	**53.13**	**51.54**	**52.38**	**51.69**

eters we are getting almost zero improvement in the classification performance. This also intuitively shows that when the performance is not acceptable, the Bayesian approach drastically changes the hyperparameters to find the areas in the search space with better accuracies. In experiment four, the hyperparameter combination leads to an acceptable classification performance of 44.21%. From this point forward, the changes in the hyperparameter values are less aggressive to leverage the decent performance (only two hyperparameter values are changed from experiment four to five). In experiment six, optimizer hyperparameter type and the corresponding learning rate are changed; however, the final performance is within the same range compared to experiment five. This shows that different sets of hyperparameters might lead to similar classification performances. This indicates that this problem is well-suited for multi-objective hyperparameter optimization problems, where we might achieve similar performance while minimizing energy or area consumption. Experiments seven and eight demonstrate the exploration aspect of our optimization approach, meaning that although we already know an acceptable values for the hyperparameters, we also explore other areas of the search space to see if we can further improve the performance or not.

Table 6.5 shows a comparison between the Spiking Neural Network (SNN) classification accuracies on MNIST, Fashion-MNIST, CIFAR-10, and CIFAR-100 dataset for state-of-the-art models and network architectures in the literature. The purpose of this work is not obtaining the best accuracy for each dataset; instead, our goal is to show that with an effective hyperparameter optimization framework, we can drasti-

Table 6.4.: Hierarchical-PABO for Whetstone, case study two. Sensitivity Analysis: Comparing CIFAR-100 classification accuracy for different experiments

CIFAR-100	**Exp. 1**	**Exp. 2**	**Exp.3**	**Exp. 4**	**Exp. 5**	**Exp. 6**	**Exp. 7**	**Exp. 8**	**Exp. 9**
Optimizer Learning Rate	0.0001	1	0.1	0.0001	0.0001	1	1	0.001	1
Optimizer Rho	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9
Optimizer Epsilon	1e-8	1e-6	1e-8	1e-8	1e-8	1e-8	1e-8	1e-6	1e-6
Optimizer Decay	1e-8	1e-8	1e-6	1e-6	1e-6	1e-6	1e-6	1e-8	1e-6
Optimizer Type	Adadelta	RMSprop	RMSprop	RMSprop	RMSProp	Adadelta	RMSprop	Adadelta	Adadelta
Noise Standard deviation	0.2	0.3	-	-	-	-	0.3	0.3	-
Noise Location	1st Dense	1st Dense	No Noise	No Noise	No Noise	No Noise	1st Dense	1st Dense	No Noise
Batch Norm. Momentum, conv.	0.95	0.85	0.95	0.95	0.95	0.95	0.95	0.95	0.95
Batch Norm. Momentum, dense	0.85	0.95	0.95	0.85	0.95	0.95	0.95	0.95	0.95
Batch Norm. Epsilon	1e-3	1e-2	1e-2	1e-2	1e-2	1e-2	1e-3	1e-3	1e-3
Batch Norm. Center	True	True	False	False	False	False	False	False	False
Batch Norm. Scale	True	True	True	True	False	False	False	False	False
Sharpener Start Epoch	25	20	25	30	30	30	20	25	30
Sharpener Duration	7	6	4	4	4	4	4	4	4
Sharpener Intermission	2	1	5	5	5	5	5	5	5
Conv. layer 1, filter size	7	5	3	3	3	3	3	3	3
Conv. layer 2, filter size	5	5	5	5	5	5	5	5	5
Conv. layer 3, filter size	3	3	5	5	5	5	5	5	5
Conv. layer 1, # of features	64	128	32	32	32	128	128	128	128
Conv. layer 2, # of features	256	128	64	64	64	64	128	256	256
Conv. layer 3, # of features	512	512	256	512	512	512	512	256	512
Dense layer, # of features	256	1024	256	1024	1024	1024	1024	1024	1024
Accuracy	**1.96%**	**1.01%**	**1.01%**	**44.21%**	**46.07%**	**48%**	**1.01%**	**21.73%**	**53.42%**

cally improve a performance of a model with only few evaluations. It is worth noting that the networks that achieve higher accuracy in this table are often significantly more complicated than the network structure we use, in terms of the architecture and input encoding techniques for SNNs. By allowing for more complex network structures, we expect that comparable accuracies can be achieved.

In this work, we show that by optimizing the hyperparameters associated with Whetstone we increase the performance over the previous state-of-the-art for this algorithm. From our results, we see that the choice of hyperparameters (even among reasonable choices) can have a tremendous effect on the performance of Whetstone. We also observe that the best hyperparameters found for each dataset differ across the datasets, indicating the importance of specifically optimizing hyperparameters for each new problem when converting to binary communication. We perform some small network architecture optimizations in this work. In particular, we optimize the filter size and number of features for each of the three convolutional layers, as well as the number of features for the dense layer. We are limiting our search to a fixed maximum network depth to deploy it on embedded systems in the future. The best

Table 6.5.: Comparison of the SNN classification accuracies on MNIST, Fashion-MNIST, CIFAR-10, and CIFAR-100 dataset

Model	Network Architecture	Method	Accuracy (%)			
			MNIST	Fashion MNIST	CIFAR-10	CIFAR-100
Shrestha et al. [15]	6-layer CNN	Temporal credit assignment for backpropagating (BP) error	99.36	-	-	-
Rueckauer et al. [115]	8-layer CNN	Offline, ANN-to-SNN conversion	99.44	-	90.85	-
Hunsberger et al. [116]	AlexNet	Offline, ANN-to-SNN conversion	99.12	-	83.54	55.13
Lee et al. [117]	ResNet-11	Spike-based backpropagating	99.59	-	90.95	-
Hao et al. [118]	3-layer FF SNN	Symmetric STDP Rule	96.73	85.31	-	-
Shrestha et al. [119]	4-layer NN	Error Modulated STDP with symmetric weights	97.3	86.1	-	-
Jin et al [120]	6-layer CNN	Direct macro/micro BP	99.49	-	-	-
Sengupta et al. [18]	VGG-16	Offline, ANN-to-SNN conversion	-	-	91.55	-
Machado et al. [121]	3-layer NatCSNN	Two-phase (unsupervised STDP, ReSuMe supervised)	-	-	84.7	-
Wu et al. [122]	CIFARNet	SNN and ANN with shared weights	-	-	91.54	-
Roy et al. [123]	VGG-9	StochSigmoid XNOR-Net	-	-	87.95	55.54
Xing et al. [124]	Inception-v4	Homeostasis-based conversion	-	-	92.49	70.4
Hu et al. [125]	ResNet-8	ANN-to-SNN conversion	99.59	-	-	-
Hu et al. [125]	ResNet-44	ANN-to-SNN conversion	-	-	91.98	68.56
Guerguiev et al. [126]	ConvNet + LIFNet	Regression discontinuity design	-	91.81	76.2	-
Thiele et al. [127]		Direct spike gradient	99.52	-	89.99	-
Wu et al. [128]	8-layer CNN	Error BP through time	-	-	90.53	-
Severa et al. [56]	VGG-like	Whetstone (Sharpened ANN)	99.53	-	84.67	-
Severa et al. [56]	6-layer CNN	Whetstone (Sharpened ANN)	99.53	93.2	79	38
Hyperparameter Optimized Whetstone (this work)	**6-layer CNN**	**Bayesian hyperparameter optimized Whetstone**	**99.6**	**93.68**	**84.36**	**53.42**

results on the different datasets are shown with different parameters in Table 6.2. We anticipate that further optimizing the network architecture will be able to improve the performance of Whetstone on different datasets. In future work, we plan to use an evolutionary optimization approach such as MENNDL [84] added to Hierarchical-PABO search to further optimize the architecture (the number and type of layers) of these networks. Whetstone's simple modifications to neural network design should allow us to search for topologies including sharpening activations within this joint Bayesian, Genetic Algorithm framework to better understand when sharpening is useful and hopefully discover higher performance network designs that may better leverage binarized operations.

In [44], Whetstone is deployed on SpiNNaker [45], with slight drop in accuracy due to issues with input/output encoding. Here, we optimize the network using Whetstone, but we do not map the resulting networks to a neuromorphic hardware implementation, such as SpiNNaker [45] or Loihi [20]. As observed in [44], several other hyperparameters such as input/output encoding, different network topologies

and training parameters will have an effect on this mapping performance. In the future, we plan to include how the network performs on real neuromorphic hardware as part of our training objectives in the hyperparameter and network architecture optimization process.

Finally, as we consider mapping onto real neuromorphic hardware, there are often other important performance considerations beyond accuracy on the task at hand. For example, size, area, and energy efficiency are often important considerations for real deployments of neuromorphic systems. As such, it is important to train with those objectives in mind. In previous work, we have extended the Bayesian optimization approach [25, 28] and the fitness function used within MENNDL [129] to incorporate multiple objectives. In future work, we plan to apply this approach to the Whetstone algorithm in order to optimize networks that are both more accurate, but also more efficient.

7. HIERARCHICAL-PABO FOR BACKPROPAGATION-BASED SPIKING NEUROMORPHIC SYSTEMS

In this chapter we focus on a backpropagation-based training algorithm for spiking neural networks, Spike Layer Error Reassignment in Time or SLAYER [15]. The goal is to define optimum sets of hyperparameters that maximize SLAYER's accuracy and minimize the required time for training this SNN [29]. We first introduce SLAYER and then continue with the experimental setup and results.

7.1 Introduction

The SLAYER approach, introduced in [15], overcomes the issue of non-differentiability of the spiking function by introducing a backpropagation approach that uses a temporal credit assignment policy to backpropagate errors. In this experiment we have illustrated an approach for Hierarchical-PABO that has been successfully applied to SLAYER [15]. We have shown that we can utilize this approach to simultaneously optimize multiple objectives, including accuracy and network latency. Optimizing the latter further improves the practical usability of these algorithms.

For the SLAYER [15] algorithm on the DVS Gesture dataset, we have demonstrated that this approach has achieved state-of-the-art results by increasing the Top-1 accuracy from 94.13% to 96.2%. In addition, we have shown that with Hierarchical-PABO approach, we are able to reduce network latency (training/inference times) by 5× while obtaining comparable accuracy.

7.2 SLAYER

As already mentioned, SLAYER is a backpropagation-based training algorithm for spiking neural networks, that uses temporal credit assignment policy for training weights and delays. They treat the spiking neuron as probabilistic and observe the conditions in which the spike state changes. The derivative is well approximated by the probability density function of the spike state change [15].

In addition to typical hyperparameters of convolutional neural networks and optimization approaches, the SLAYER [15] algorithm itself has a variety of hyperparameters (HPs) to be optimized. The first set of parameters are concerned with how the neurons function in the network. SLAYER utilizes a Spike Response Model (SRM) neuron, which takes a signal processing approach to represent the behavior of a spiking neuron. Several different types of neuron models (e.g., leaky integrate-and-fire neurons) can be implemented using SRMs by changing the parameters. The particular hyperparameters of the neuron model are as follows:

- Neuron threshold (theta): If a neuron's state (or membrane potential) exceeds this value, the neuron will generate a spike.

- Spike Response time constant (tauSr): The spike response time constant determines the rate of decay for the spike response kernel. This effectively acts as a membrane time constant for the neuron.

- Refractory period time constant (tauRef): The refractory period time constant determines the refractory behavior of the neuron following a spike.

- Neuron refractory response scaling factor (scaleRef): This value indicates how to scale the neuron's response when the neuron is in its refractory period. It is specified relative to the neuron threshold value (theta).

- Spike function derivative time constant (tauRho) and spike function derivative scale factor (scaleRho): These values are used in formulating an approximation of the

derivative of the spike function. Both values are scaling factors in a probability density function for the change in state of the spiking neuron. More details are available in the original paper on SLAYER [15].

Additional HPs for SLAYER are related to the spiking neural network (SNN) simulation, as well as information about the input data. Those parameters are:

- Time length of sample (tSample): This is how long the SNN simulation is run for each data sample. This value is specified in ms.
- Sampling time (Ts): The specifies the size of the time step in the SNN simulation. This value is specified in ms.

Finally, SLAYER requires that both correct and incorrect output neurons fire over the course of training to alleviate the "dead neuron" problem that is common in backpropagation-based SNN training approaches. Thus, SLAYER also requires extra set of HPs to specify how many times the "true" neuron should spike (sCountTrue) and how many times the "false" neurons should spike (sCount_div). Please see the original SLAYER paper [15] and the SLAYER source code for PyTorch[1] for full information about the SLAYER algorithm and how it is implemented. The original SLAYER approach has the ability to learn both synaptic weights and axonal delays, but in the current software release (in Python, utilizing pytorch), SLAYER only learns the synaptic weights of the SNN.

7.3 Experimental Setup and Results

We use Hierarchical-PABO for defining sets of HPs that maximize SLAYER's accuracy with the minimum training time per epoch (as a proxy for the network latency). The HPs on the Pareto front not only lead to SNN performances (both in terms of accuracy and speed) that surpass those obtained in [15], but also show a trade-off in designing accurate and fast networks. Such information can be used by a

[1] `https://github.com/bamsumit/slayerPytorch`

skilled designer to tune their networks according to their requirements. For example, a network designer might choose to reduce accuracy by a few hundredths in exchange for doubled speed. All results are obtained with NVIDIA Tesla V100 GPUs with 16GB memory.

7.4 Results

In the first case study for SLAYER, we apply Hierarchical-PABO on the DVS Gesture dataset [40] to optimize SLAYER's accuracy and training/inference time. We intentionally design a small search space (with only 512 different HP combinations), so that we can obtain a ground truth for the actual Pareto region of the problem by running a grid search algorithm. We then compare the actual Pareto region of the problem with the one estimated using our Hierarchical-PABO technique. We repeat each HP combination five times to account for the inherent stochasticity involved in designing neural networks. The evaluated and fixed HPs for this case study are given in Table 7.1. All evaluated HPs are in reasonable and acceptable ranges for the SLAYER algorithm. In Table 7.1, *Arch* refers to the SLAYER architecture.. Architecture 1 ("$Arch = 1$") refers to a ten layer SNN with spiking delays after the convolution and fully connected layers, architecture 2 ("$Arch = 2$") refers to the original eight layer SNN from [15], and architecture 3 ("$Arch = 3$") is a ten layer SNN without any spiking delays. Details of these architectures are as follows:

- $Arch = 1$: input/pool1/conv1/delay1/pool2/conv2/delay2/pool3/conv3/delay3/ pool4/fc1/delay4/fc2.
- $Arch = 2$: input/pool1/conv1/delay1/pool2/conv2/ delay2/pool3/fc1/delay3/fc2
- $Arch = 3$: input/pool1/conv1/pool2/conv2/pool3/ conv3/ pool4/fc1/fc2.

In this notation, "pool" refers to pooling layer, "conv" is convolution layer, and "fc" is a fully connected layer. In addition, in Table 7.2 filters 1 to 3 refer to filter sizes in convolution layers 1 to 3 in the architectures described above.

Table 7.1.: Hierarchical-PABO for SLAYER, case study one. Fixed and evaluated HPs with their corresponding values, as well as details of HPs for maximum accuracy (Point A) and minimum time (Point B)

Fixed Hyperparameters		Evaluated Hyperparameters		Point A	Point B
tSample	1200	Ts	10, 14	10	14
theta	10	sCountTrue	200, 150	200	200
sCountFalse	20	tauRho	0.1, 1	1	1
tauSr	2 × Ts	scaleRho	0.5, 1	0.5	0.5
tauRef	Ts	Optimizer	Adam, Nadam	Adam	Adam
scaleRef	2	[Arch.,Filter1,2,3]	[3,7,5,3], [1,7,3,3], [3,3,5,3], [2,5,3,3]	[3,7,5,3]	[2,5,3,3]
Learning Rate	0.01	# of Feature in FC	256, 512	512	512
# Feature 1,2,3	[20,32,64]	Batch Size	4, 8	4	8
Search Space Size: 512				95.113% 73.17sec	91.35% 29.19sec

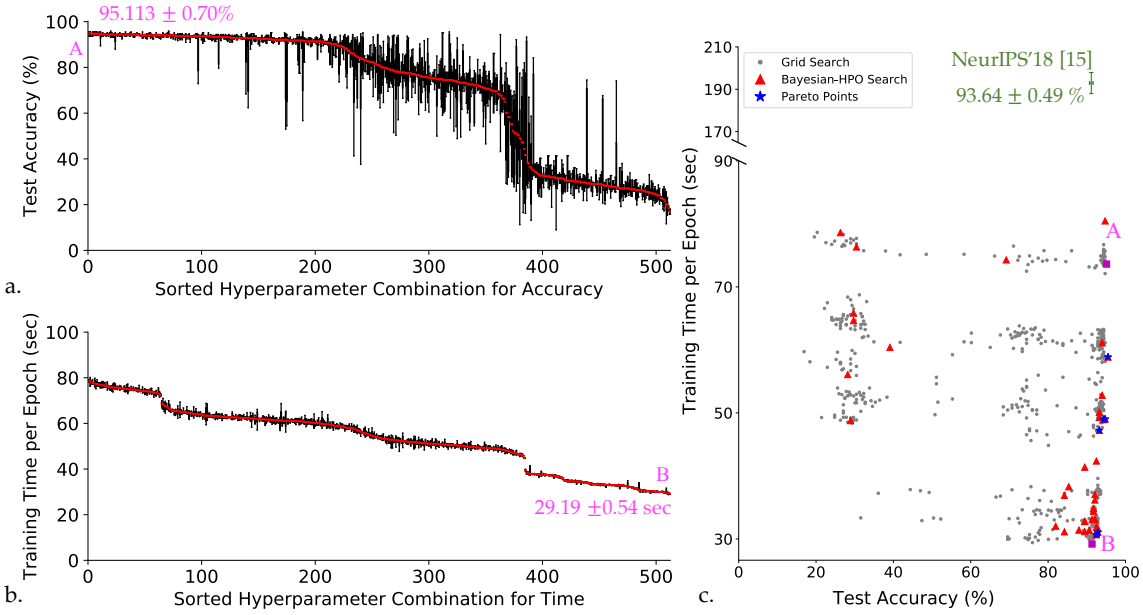

Fig. 7.1.: Hierarchical-PABO for SLAYER, case study one. Grid search results are shown in, (a) for accuracy; and (b) for training time per epoch; for all possible HP combinations based on the HPs given in Table 7.1. Please note, the x-axis in (a) and (b) are NOT the same, as the combinations are sorted in descending order according to accuracy in (a) and time in (b). (c) Pareto front obtained using Hierarchical-PABO framework with only 15 iterations. The green data point, shows the best result from [15]. Points A, and B, refer to maximum accuracy and minimum training time per epoch, respectively. The corresponding HPs for these two points are given in Table 7.1.

Figure 7.1a and Figure 7.1b show how SLAYER's accuracy and training time per epoch change for the hyperparameter combinations given in Table 7.1. Each data point on these two panels are a boxplot with interquartile ranges that are obtained after 5 repeats of the same run with the same set of hyperparameter. The test accuracy drastically changes from 95.113% to 16.91% if only sCountTrue changes from 200 to 150, tauRho from 1 to 0.1, scaleRho from 0.5 to 1, and filter size in the first convolution layer from 7 to 3. We emphasize that all these hyperparameters are chosen within a reasonable range typically used by the designers of such networks. Please note that HP combinations are sorted for test accuracy, and for training time per epoch individually on the x-axis of Figure 7.1a and b, respectively; and therefore, the HP indices are different in the two panels in the figure.

Figure 7.1c demonstrates the Hierarchical-PABO search points and estimated Pareto points after 15 iterations of the Bayesian algorithm. The Pareto points (shown as blue stars) are within close proximity to the actual Pareto frontier of the problem (obtained from grid search shown as grey dots). The optimizer search points (shown in red squares) are the Bayesian estimators observations and show how the entire search space is explored and exploited. The accuracy shown as the *NeurIPS'18* point in the figure is the accuracy reported in [15]. It should be noted that the authors of that work have provided us with the hyperparameters that lead to the reported accuracy. We used those HPs to obtain the training time per epoch for the network on the same system on which all other runs were carried out. With only 15 iterations of our Bayesian HPO framework, we obtained sets of hyperparameters that surpass results given by [15] both in terms of accuracy and speed of training as shown in Figure 7.1c. The co-optimized Pareto HP points show better or comparable accuracy while reducing the training time per epoch by as much as 5×. In all data points shown in Figure 7.1, the number of epochs is fixed to 40.

Table 7.2 gives details of the HPs that led to the Pareto points shown with blue stars in Figure 7.1c. For each HP combination, the median value for accuracy and training time per epoch after five evaluations of SLAYER is given in Table 7.2. All of

Table 7.2.: Hierarchical-PABO for SLAYER, case study one on DVS Gesture dataset using SLAYER. Hyperparameter combinations for Pareto points shown with blue stars in Figure 1. Search space size: 512

Evaluated Hyperparameter	**Point 1**	**Point 2**	**Point 3**	**Point 4**	**Point 5**	**Point 6**
Ts	10	14	14	14	10	10
sCountTrue	200	200	200	200	200	200
tauRho	1.0	1.0	1.0	1.0	1.0	1.0
scaleRho	0.5	1.0	1.0	1.0	0.5	0.5
Optimizer	Adam	Adam	Adam	Adam	Nadam	Nadam
[Arch.,Filter1,2,3]	[2,5,3,3]	[3,7,5,3]	[3,7,5,3]	[3,3,5,3]	[3,7,5,3]	[2,5,3,3]
# of Feature in FC	256	256	512	512	256	256
Batch Size	8	8	8	4	8	8
Accuracy (%)	92.86%	94.74%	94.36%	93.23%	95.49%	92.48%
Training Time per Epoch (sec)	31.06	49.02	48.92	47.22	58.86	30.67

these points belong to the Pareto frontier of a multi-objective optimization problem; therefore, none of them can be dominated by any other one for both accuracy and time. However, with only a 0.75% drop in median accuracy, we can drop the median training time by 9.84 seconds per epoch (point 5 vs. point 2 in Table 7.2).

As indicated in Table 7.1, in this case study, tSample is fixed to 1200, and Ts is limited to 10 or 14. The ratio of tSample to Ts is the number of discrete time steps to evaluate which is a primary factor for training time. However, other HPs also play an important role in training time such as tauSR which helps determine the spike response kernel. The response kernel must be convolved with the spiking signal for each layer, so a larger kernel results in additional computation. In the grid search results shown in Figure 7.1, to have a manageable search space size, tSample/Ts is either 120, or 85. These factors combined with all other HPs given in Table 1, create a grid search with only 512 HP combinations and a training time per epoch that varies from 79 seconds to 29 seconds. In problems where the hyperparameter combinations are on the order of millions and billions, a simple change in one HP can create orders of magnitude shifts in training time. In the *NeurIPS'18* [15] point shown in Figure 7.1c, tSample is 1450, and theta is 1.0. This change, combined with the values for other HPs, led to a non-optimal design with almost 5× higher training time.

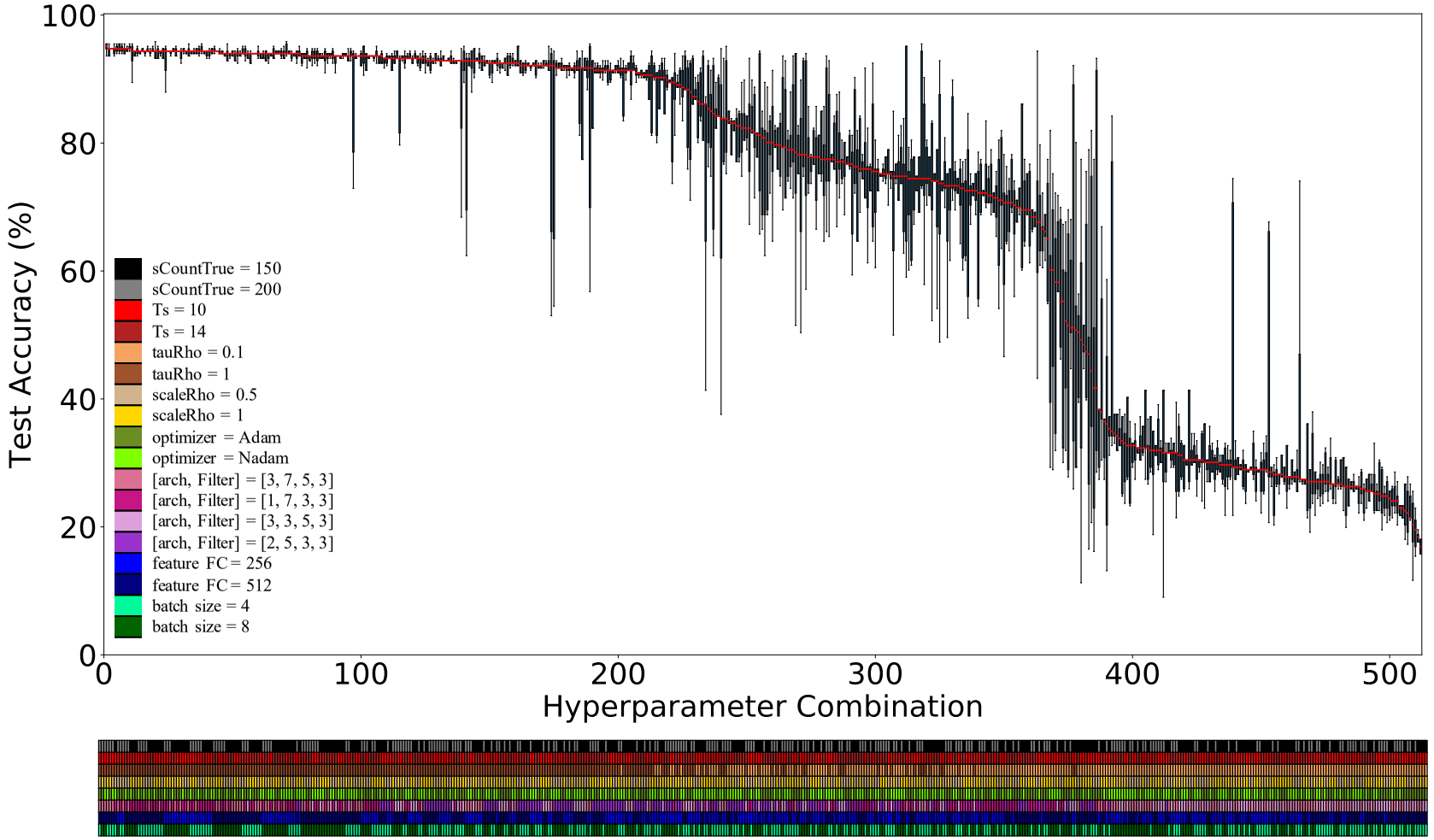

Fig. 7.2.: Hierarchical-PABO for SLAYER, case study one. Grid search results for Accuracy with their corresponding HPs for all possible combinations of HPs given in Table 1

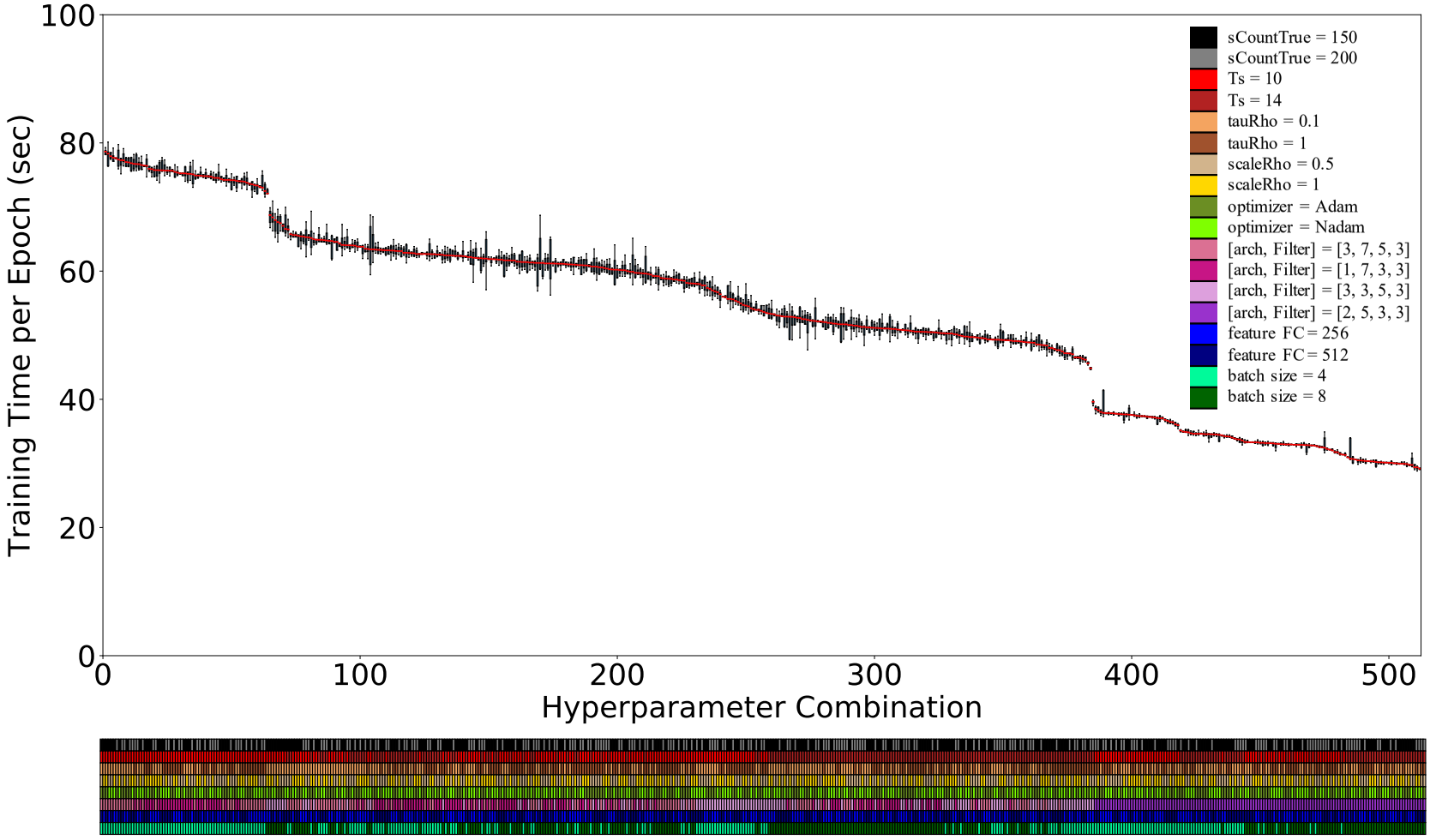

Fig. 7.3.: Hierarchical-PABO for SLAYER, case study one. Grid search results for Training Time with their corresponding HPs for all possible combinations of HPs given in Table 1

Table 7.3.: Hierarchical-PABO for SLAYER. Hyperparameter combination for the optimum accuracy of 96.421% on DVS gesture dataset

Optimum HP	**Value**	**Optimum HP**	**Value**
tSample	1200	tauRho	1
theta	10	scaleRho	0.5
sCountTrue	200	Learning Rate	0.01
sCountFalse	20	Optimizer	Adam
Ts	10	Batch Size	4
tauSr	20	[Arch.,Filter1,2,3]	[3,3,5,3]
tauRef	10	# Feature 1,2,3	[20,32,64]
scaleRef	2	# of Feature in FC	512

Test Accuracy: 96.421%
Number of Epochs: 100
Number of Bayesian Iterations: 40
Number of Repeats: 5
Training/Test split: 70%/30%
Performance Metric: Accuracy-only

Figures 7.2, and 7.3 demonstrate the changes in accuracy and training time per epoch for different HPs, respectively. All 512 HP combinations given in Table 7.1 are ranked from best to worst for accuracy (in Figure 7.2), and training time per epoch (in Figure 7.2). The box plots show the variation across 5 evaluations for each HP set. The median value is shown with red line. The color blocks for each figure represents HP combination that correspond to the given accuracy, and training time per epoch (for Figures 7.2, and 7.3, respectively). These figures show the drastic difference that HP combinations have on the SLAYER performance (both in terms of accuracy and time). In addition, the color blocks for each set of HP show that a combination of HPs affects the final performance, not only one specific HP. This further demonstrates how critical it is to perform hyperparameter optimization when designing any neural accelerator system.

In addition, we used our Hierarchical-PABO for optimizing accuracy only, and obtained 96.241% Top-1 accuracy compared to the Top-1 accuracy of 94.13% reported by [15]. This optimum HP combination was found after 40 iterations of Bayesian optimizer with maximum epoch count of 100, and it is given in Table 7.3.

To further investigate efficiency of our Hierarchical-PABO approach, in case study two on SLAYER, we increase the search space size to 14,929,920 different HP combinations. Figure 7.4 shows that after only 40 iterations of the Hierarchical-PABO search the Pareto frontier is estimated (which is shown in blue stars). The hyperparameter ranges for this example are given in Table 7.4. The four Pareto points

Table 7.4.: Hierarchical-PABO for SLAYER, case study two on DVS Gesture dataset. Search space size: 14,929,920. Results are shown in Figure 7.4

HP	Range	HP	Range	HP	Range
tSample	1000, 1200, 1400, 1600, 2000	tauRho	0.1, 0.5, 1	# Feature 1	20
theta	10, 12	scaleRho	0.1, 0.5, 1	# Feature 2	32, 64
sCountTrue	100, 150, 200, 250	tauRef	Ts/1, Ts/2, Ts/3	# Feature 3	64, 128
Ts	10, 16	scaleRef	1, 2, 3	# of Feature in FC	256, 512
sCountFalse	sCountTrue/5, sCountTrue/10, sCountTrue/15	[Arch.,Filter1,2,3]	[2, 5, 3, 5], [3, 7, 5, 3], [1, 7, 3, 3], [2, 5, 3, 3], [2, 3, 5, 3], [3, 3, 5, 3], [3, 7, 3, 3], [3, 5, 3, 3]	Optimizer	Adam, Nadam
tauSr	Ts × 1, Ts × 2, Ts × 3	Learning Rate	0.001, 0.01	Batch Size	8

shown with blue stars in Figure 7.4 and their corresponding HPs are summarized in Table 7.5. This study shows that for a case study with almost 15 million different HP combinations, we are able to find optimum HPs with acceptable ranges for both accuracy and training time with only 40 iterations of the Hierarchical-PABO approach.

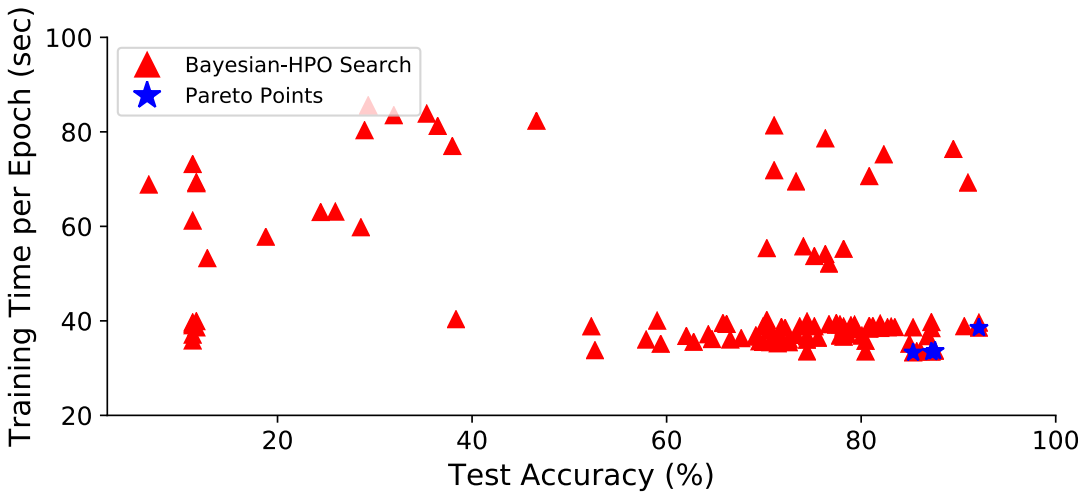

Fig. 7.4.: Hierarchical-PABO for SLAYER, case study two on DVS Gesture dataset. Search space size: 14,929,920 with HP ranges given in Table 7.5

Table 7.5.: Hierarchical-PABO for SLAYER, case study two. Pareto Points and the Corresponding HPs for HP ranges given in Table 7.4, and results shown in Figure 7.4

HP [tSample, theta, sCountTrue, sCountFalse, Ts, tauSr, tauRef, scaleRef, tauRho, scaleRho, LR, optimizer, [Arch, Filter1,2,3], Feature 1, Feature 2, Feature 3, Feature FC, Batch-size]	[Accuracy, Training time per epoch]
[1400, 12, 200, 40, 10, 20, 10, 1, 0.5, 1.0, 0.01, Adam, 5, 20, 32, 64, 256, 8]	92.10%, 38.51 sec
[1000, 10, 100, 20, 10, 10, 20, 1, 1.0, 0.1, 0.01, Nadam, 1, 20, 32, 64, 256, 8]	87.22%, 33.43 sec
[1000, 10, 100, 20, 10, 20, 20, 1, 1.0, 0.5, 0.01, Nadam, 1, 20, 32, 64, 256, 8]	87.60%, 33.62 sec
[1000, 10, 100, 20, 10, 20, 20, 2, 1.0, 0.1, 0.01, Nadam, 1, 20, 32, 64, 256, 8]	85.34%, 33.28 sec

Table 7.6.: Comparison of the results of our Hierarchical-PABO framework with other SNN models on DVS Gesture dataset [40]

Work	Network Type	Accuracy (%)
Kaiser eRBP [130]	Feed Forward SNN, No Convolutions	92.7
DECOLLE [131]	Deep SNN, Online Learning	95.54
Fang IIR [132]	Deep SNN, Neuron+Synapse Filters	96.09
SLAYER [15]	Deep SNN	93.64
This Work	**HP Optimized SLAYER**	**96.241**

7.5 Discussions

In Table 7.6 we compare recent results in the literature with our results on DVS Gesture dataset [40] with an HP optimized SLAYER. In this table we only show a sample of results for our multi-objective HPO. There are several Pareto points discussed in the paper that when compared to other results in the literature are better in terms of accuracy, time, or both.

In this work, we adopted a novel hierarchical-PABO approach, and applied it to a distinct SNN training algorithm, SLAYER. The proposed multi-objective method was utilized to simultaneously optimize multiple objectives, including accuracy and network latency. Optimizing the latter further improves the practical usability of these algorithms. The estimated hyperparameter sets from the proposed method results in more accurate and faster networks compared to the state-of-the-art approaches. Through several case studies, we demonstrated cutting-edge results that can be leveraged by a skilled designer in the field to design lightweight, energy efficient and accurate architectures.

8. HIERARCHICAL-PABO FOR CONVERSION-BASED SPIKING NEUROMORPHIC SYSTEMS

The HYBRID approach proposed by [17] relies on converting an already trained ANN to an SNN with appropriate threshold-balancing, and then fine-tuning with a spike-based backpropagation approach. The technique proposes a computationally efficient algorithm with reduced latency and time-steps [17]. In this chapter, we first briefly review HYBRID approach, and then present the experimental setup and results for using our Hierarchical-PABO framework to push HYBRID limits to its full capacity. The goal is to find optimum set of HPs that maximize performance of HYBRID training algorithm in terms of accuracy and speed of training/inference [29].

8.1 HYBRID

HYBRID [17] proposes a hybrid training approach for deep SNNs on VGG [32] and ResNet [133] architectures for image classification tasks [17]. The HYBRID technique combines the two previously proposed methods of supervised training in SNNs: ANN-SNN conversion [18,66], and surrogate gradient-based backpropagation in SNN [134]. The objective of the HYBRID approach is to reduce the inference latency associated with the conversion methods, and at the same time enable the training of deep SNNs that is otherwise difficult to perform with only backpropagation methods due to high training time (wall-clock time).

In this method, an ANN (with ReLU neurons) is trained with standard backpropagation and then converted to an SNN (with integrate-and-fire (IF) neurons) with appropriate threshold-balancing [18]. The threshold-balancing computes the threshold for each layer as the maximum pre-activation value for that layer. This converted SNN that already achieves decent accuracy ($3\% - 15\%$ less than ANN) is further

fine-tuned (to improve the accuracy) with spike-based backpropagation to capture the temporal information from the spatiotemporal inputs. Please see the original HYBRID paper [17] and the HYBRID source code for PyTorch[1] for full information about the HYBRID algorithm and how it is implemented.

8.2 Experimental Setup and Results

As the HYBRID technique is designed based on ANN-SNN conversion, if the accuracy of the underlying ANN is maximized, the overall performance of the HYBRID approach increases significantly. Therefore in this experiment, we use the **two-level Hierarchical-PABO (H-PABO) technique** and demonstrate how the HYBRID approach is further improved, in terms of accuracy and time-steps, once both ANN and SNN's HPs are optimized. In the first level of the two-level H-PABO technique, we use the framework as a single-objective optimizer (SOO) to find an optimum set of HPs that maximizes the performance of the underlying ANN. In the second level, we use the Bayesian HPO as a multi-objective hyperparameter optimizer (MOO) that optimizes both accuracy of the trained SNN and the latency in terms of time-steps.

For the ANN, we choose the optimizer, learning rate, learning rate change schedule, weight decay term (L2 regularization), momentum, and dropout as the hyperparameters. Additionally, for the SNN, we select time-steps, leak, and scaling factor (used to reduce the thresholds at lower time-steps) as the hyperparameters. All the ANN hyperparameters are also considered for the SNN during the spike-based backpropagation. The hierarchical-PABO finds the sets of HPs that belong to the Pareto region of the problem. The HPs on the Pareto front not only lead to SNN performances (both in terms of accuracy and speed) that surpass those obtained in [17], but also show a trade-off in designing accurate and fast networks. All results are obtained with NVIDIA Tesla V100 GPUs with 16GB memory.

[1] `https://github.com/nitin-rathi/hybrid-snn-conversion`

Table 8.1.: Hierarchical-PABO for HYBRID. Summary of results for the two-level hierarchical Bayesian HPO compared with results from [17]

		ANN		SNN	
Dataset	Architecture	ICLR'20 [17] Accuracy	This work, SOO Accuracy	ICLR'20 [17] [Accuracy, Time-step]	This Work, MOO: Sample Pareto Points [Accuracy, Time-step] (# of epochs)
CIFAR10	VGG5	87.88%	90.66%	[86.91%,75]	[90.00%,70], [89.32%,60], [88.53%,40] (30)
	VGG9	91.45%	92.42%	[90.54%,100]	[91.53%,90], [91.32%, 80], [90.71%, 60] (30)
	VGG13	-	93.6%	-	[91.45%,80], [89.81%, 60], [88.00%, 40] (30)
	VGG16	92.81%	93.9%	[91.13%, 100]	[91.47%,120], [90.04%,70] (5)
	RESNET20	93.15%	93.53%	[92.22%, 250]	[90.49%,80], [90.24%,60] (5)
CIFAR100	VGG11	71.21%	69.29%	[67.87%, 125]	[63.87%, 60](5)
	VGG16	-	72.35%	-	[62.70%,80],[60.90%,60] (5)
	RESNET20	-	64.56%	-	[63.10%,80], [61.74%,60] (30)

Table 8.2.: Hierarchical-PABO for HYBRID. HPs for the Pareto points given in Table 8.1

Dataset	Architecture	[Accuracy (%), Time-step]: Pareto HP [LR, LR_red, LR_int, Weight-decay,Time-step, Leak, Scaling-factor, Dropout] (# of epochs)
CIFAR10	VGG5	[90.00%,70]: [1e-5, 5, [0.4, 0.65, 0.85], 5e-5, 70, 1.0, 0.5, 0.2] (30) [89.32%,60]: [1e-5, 10, [0.4, 0.65, 0.85], 5e-5, 60, 1.0, 0.5, 0.2] (30) [88.53%,40]: [1e-5, 5, [0.6, 0.8, 0.9], 5e-5, 40, 1.0, 0.5, 0.2] (30)
	VGG9	[91.53%,90]: [1e-5, 5, [0.4, 0.65, 0.85], 5e-5, 90, 1.0, 0.5, 0.2] (30) [91.15%, 70]: [1e-5, 5, [0.6, 0.8, 0.9], 5e-5, 70, 1.0, 0.5, 0.2] (30) [90.38%, 50]: [1e-5, 10, [0.6, 0.8, 0.9], 5e-5, 50, 1.0, 0.5, 0.2] (30)
	VGG13	[91.45%,80]: [5e-4, 5, [0.6, 0.8, 0.9], 5e-5, 80, 1.0, 0.5, 0.2] (30) [89.81%, 60]: [1e-5, 5, [0.6, 0.8, 0.9], 5e-5, 60, 1.0, 0.5, 0.2] (30) [88.00%, 40]: [1e-4, 10, [0.5, 0.7, 0.9], 1e-4, 40, 1.0, 0.5, 0.3] (30)
	VGG16	[91.47%, 120]: [1e-5, 2, [0.6, 0.8], 5e-4, 120, 0.95, 0.4, 0.0] (5) [90.04%, 70]: [5e-4, 10, [0.6, 0.8], 1e-4, 70, 0.95, 0.5, 0.3] (5)
	RESNET20	[90.49%, 80]: [1e-4, 10, [0.6, 0.8], 1e-5, 80, 1.0, 0.5, 0.0] (5) [90.24%,60]: [5e-4, 5, [0.4, 0.6], 1e-6, 60, 1.0, 0.5, 0.0] (5)
CIFAR100	VGG11	[63.87%, 60]: [1e-4, 10, [0.6, 0.8, 0.9], 5e-4, 60, 1.0, 0.5, 0.3] (5)
	VGG16	[62.70%, 80]: [1e-4, 2, [0.6, 0.8], 5e-4, 80, 1.0, 0.4, 0.1] (5) [60.90%,60]: [5e-4, 5, [0.6, 0.8], 5e-5, 60, 0.99, 0.5, 0.0] (5)
	RESNET20	[63.10%,80]: [5e-5, 10, [0.4, 0.6], 5e-5, 80, 1.0, 0.5, 0.2] (30) [61.74%,60]: [5e-5, 8, [0.6, 0.8], 5e-4, 60, 1.0, 0.5, 0.2] (5)

In this experimental setup, we use the HYBRID technique [17] on image classification datasets of CIFAR10 and CIFAR100 [37] using VGG [32] and RESNET [133] architectures.

Table 8.1 shows a summary of the results on both the ANN and SNN compared with [17], where available. For CIFAR10 on VGG architectures without Hierarchical-PABO reported by HYBRID [17], the highest accuracy and lowest time-steps found were 91.13% and 100, respectively, on the VGG16 architecture. In this work, however,

there are numerous Pareto points that surpass the previous state-of-the-art results both in terms of accuracy and time-step. For example, this approach discovered a set of hyperparameters that led to a Pareto point with higher accuracy of 91.45%, a lower time-step of 80, and the simpler architecture of VGG9.

Table 8.2 gives details of the HPs for the Pareto points shown in Table 8.1 for CIFAR-10 and CIFAR-100 datasets using different VGG and ResNet architectures. In this table, "LR" represents Learning Rate, "LR_red" refers to how much to reduce LR, and "LR_int" gives the location where LR is reduced. Figure 8.1 demonstrates how the Hierarchical-PABO searches for the Pareto frontier of the multi-objective optimization problem. The distribution of red triangles in the figure show the exploration and exploitation of the search space. After 15 iterations of the Bayesian estimators, the Pareto frontier is estimated, which is shown in blue stars in Figure 8.1.

As discussed in [17], with the increase in the spiking activity, the energy efficiency of the SNN decreases. Thus, although deeper networks might increase the training accuracy, an increase in the average number of spikes per layer, as well as increasing the number of layers in the network, will cause a decrease in the energy efficiency of the network. Here, we are able to show that by optimizing hyperparameters, we can achieve similar accuracy with smaller (and hence more efficient) SNNs.

Table 8.3 gives details of HPs and their ranges for each single-objective optimization run for the underlying ANN. With a search space size of $12,096$, we searched for the optimum HP that maximizes the underlying ANN performance (in terms of test accuracy) for all VGG and ResNet architectures used in these experiments on CIFAR-10 and CIFAR-100 datasets. Figure 8.2 is a demonstration of how the hierarchical Bayesian HPO searches for the optimum HP for CIFAR-10 for the VGG16 architecture. This figure shows as the number of iterations increases in the Bayesian estimator, the underlying ANN accuracy is optimized. Similar trends are seen for all other architectures on CIFAR-10 and CIFAR-100. The batch size is fixed to 64 and the maximum epoch count for all experiments was 300.

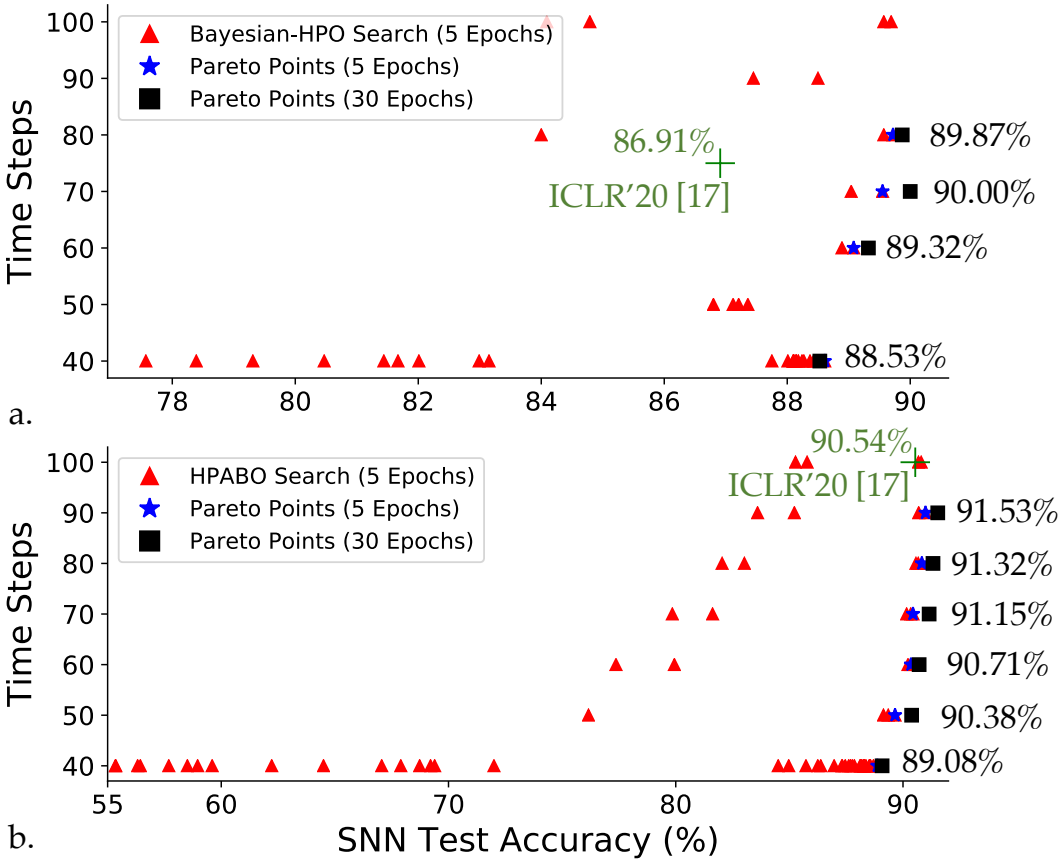

Fig. 8.1.: Hierarchical-PABO for HYBRID. CIFAR10 on VGG5. Architectures with better performances are shown in the Pareto frontier. These designs improve the accuracy by 2-4%, with 50% reduce in time-steps.

Table 8.3.: Hierarchical-PABO for HYBRID. Single-objective optimization (SOO): Search space size: 12,096 for SOO on Underlying ANN

Hyperparameter	Range
LR	0.1, 0.07, 0.03, 0.01, 0.007, 0.003, 0.001
LR_red	5, 10
LR_int	[0.4, 0.6, 0.8], [0.5, 0.7, 0.9], [0.6, 0.8, 0.9]
Optimizer	SGD, Adam, Adagrad, Adamax, Adadelta, RMSprop
Weight-decay	1e-5, 5e-5, 1e-4, 5e-4
Momentum	0.8, 0.85, 0.9, 0.95
Dropout	0.2, 0.3, 0.4

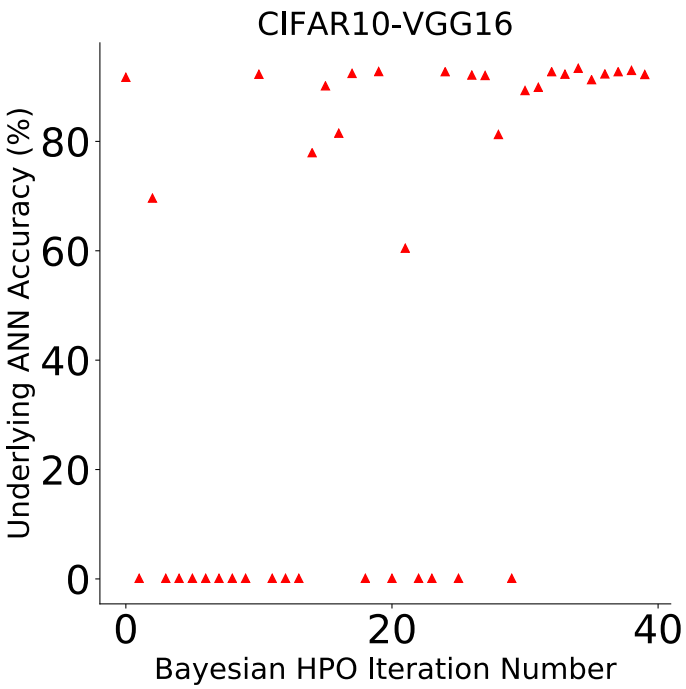

Fig. 8.2.: Hierarchical-PABO for HYBRID, Single-objective optimization. Search space size: 12,096 for SOO on Underlying ANNs with HP ranges given in Table 8.3

Figures 8.3 and 8.4 show the Pareto frontier of the multi-objective optimization problem (accuracy and time-steps) for several architectures on CIFAR-10 and CIFAR-100 image classifications. The corresponding HPs for each single Pareto point are given in Tables 8.4, and 8.5. For all of these examples the optimizer is fixed to Adam.

8.3 Discussions

In Table 8.6 we compare recent results in the literature with our results on CIFAR-10 with an HP optimized HYBRID. In this table we only show a sample of results for our multi-objective HPO. There are several Pareto points discussed in the paper that when compared to other results in the literature are better in terms of accuracy, time, or both.

Using the proposed novel two-level Hierarchical-PABO approach, that contains a single-objective Bayesian approach for hyperparameter optimization of the ANN and

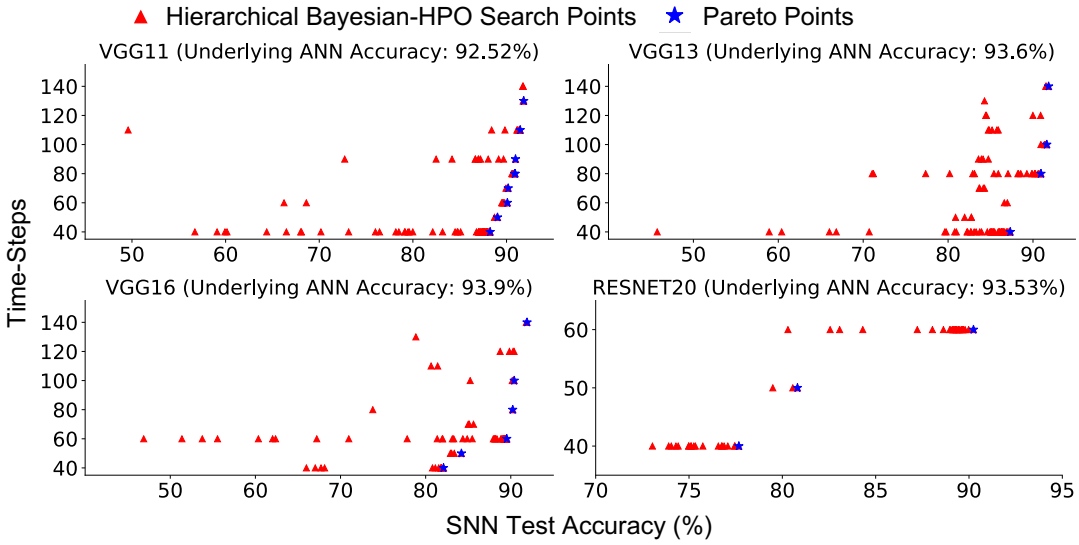

Fig. 8.3.: Hierarchical-PABO for HYBRID. CIFAR10 Image Classification on VGG and RESNET architectures

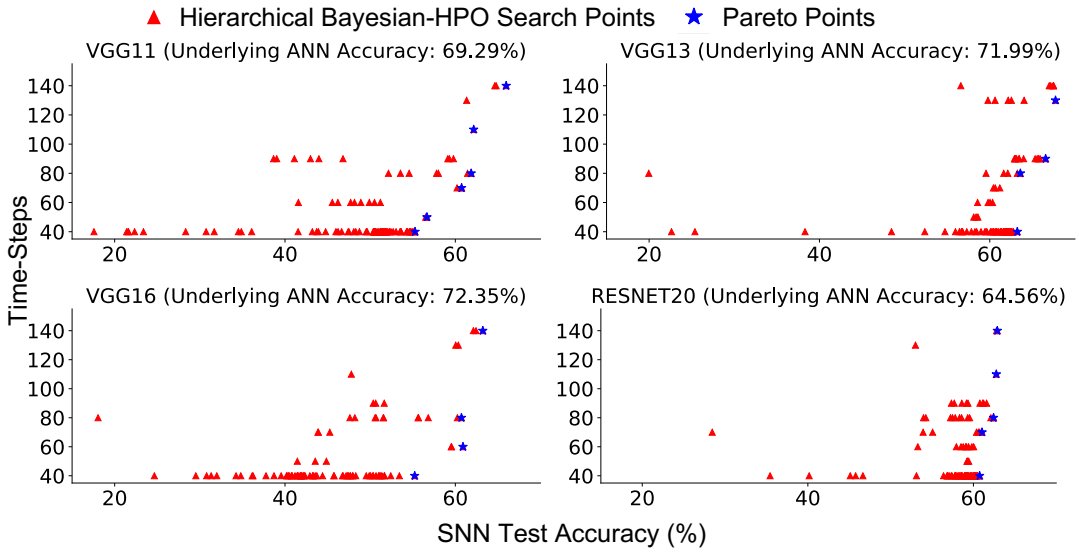

Fig. 8.4.: Hierarchical-PABO for HYBRID. CIFAR100 Image Classification on VGG and RESNET architectures

Table 8.4.: Hierarchical-PABO for HYBRID. Details of the HPs for the Pareto Points in Figure 8.3

Architecture	Pareto HP = [LR, LR_red, LR_int, Weight-decay, Time-step, Leak, Scaling-factor, Dropout]	Accuracy	Time-step
CIFAR10			
VGG11	[1e-5, 10, [0.6, 0.8], 1e-5, 40, 1.0, 0.5, 0.0]	88.25%	40
	[1e-5, 15, [0.4, 0.6], 1e-5, 50, 1.0, 0.5, 0.0]	89.04%	50
	[1e-5, 10, [0.6, 0.8], 1e-6, 60, 1.0, 0.5, 0.1]	90.09%	60
	[1e-5, 5, [0.4, 0.6], 1e-5, 70, 1.0, 0.5, 0.0]	90.18%	70
	[1e-5, 10, [0.6, 0.8], 5e-5, 80, 1.0, 0.5, 0.3]	90.92%	80
	[1e-5, 5, [0.4, 0.6], 1e-5, 90, 1.0, 0.5, 0.0]	90.96%	90
	[1e-5, 5, [0.6, 0.8], 1e-5, 110, 0.99, 0.5, 0.0]	91.46%	110
	[1e-5, 15, [0.6, 0.8], 1e-5, 130, 1.0, 0.6, 0.5]	91.84%	130
VGG13	[5e-4, 5, [0.4, 0.6], 5e-4, 40, 0.99, 0.4, 0.1]	87.32%	40
	[1e-5, 8, [0.4, 0.6], 5e-4, 80, 1.0, 0.6, 0.4]	90.94%	80
	[1e-4, 5, [0.6, 0.8], 5e-5, 100, 0.95, 0.4, 0.3]	91.63%	100
	[1e-5, 10, [0.4, 0.6], 5e-5, 140, 1.0, 0.7, 0.0]	91.84%	140
VGG16	[1e-5, 5, [0.4, 0.6], 1e-5, 40, 1.0, 0.4, 0.1]	82.1%	40
	[1e-5, 5, [0.4, 0.6], 1e-5, 50, 1.0, 0.4, 0.0]	84.2%	50
	[1e-5, 5, [0.4, 0.6], 1e-5, 60, 1.0, 0.5, 0.0]	89.53%	60
	[5e-4, 8, [0.6, 0.8], 1e-4, 80, 0.95, 0.4, 0.0]	90.23%	80
	[1e-5, 5, [0.4, 0.6], 5e-4, 100, 0.99, 0.5, 0.1]	90.4%	100
	[1e-5, 10, [0.4, 0.6], 5e-5, 140, 1.0, 0.7, 0.0]	91.91%	140
RESNET20	[1e-5, 5, [0.6, 0.8], 1e-6, 40, 1.0, 0.5, 0.2]	77.67%	40
	[1e-5, 5, [0.4, 0.6], 1e-5, 50, 1.0, 0.5, 0.0]	80.81%	50
	[5e-4, 5, [0.4, 0.6], 1e-6, 60, 1.0, 0.5, 0.0]	90.24%	60

an agent-based multi-objective Bayesian approach for hyperparameter optimization of the SNN, we have optimized and trained networks that outperform the previous state-of-the-art HYRBID [17] training SNN results on the CIFAR10 and CIFAR100 dataset with VGG and RESNET architectures in terms of accuracy with more than 40% reduction in network latency (time steps). In addition, we demonstrate that the proposed approach can discover hyperparameters for simpler architectures that achieve higher accuracy and lower latency than previously published results. Both the reduction in architecture size and network latency have significant implications for energy efficiency of these architectures. For example, we demonstrate the results for CIFAR10 on VGG9 with improved accuracy compared to a much deeper and more energy-consumptive VGG16, and with 30% reduction in inference time. Finally, by

Table 8.5.: Hierarchical-PABO for HYBRID. Details of the HPs for the Pareto Points in Figure 8.4

Architecture	Pareto HP = [LR, LR_red, LR_int, Weight-decay, Time-step, Leak, Scaling-factor, Dropout]	Accuracy	Time-step
CIFAR100			
VGG11	[1e-4, 5, [0.6, 0.8], 1e-5, 40, 0.99, 0.5, 0.0]	55.27%	40
	[1e-5, 10, [0.4, 0.6], 1e-5, 50, 1.0, 0.5, 0.0]	56.67%	50
	[1e-5, 10, [0.4, 0.6], 1e-5, 70, 1.0, 0.5, 0.2]	60.74%	70
	[1e-5, 5, [0.6, 0.8], 5e-5, 80, 0.99, 0.5, 0.1]	61.86%	80
	[1e-5, 5, [0.4, 0.6], 1e-6, 110, 0.99, 0.6, 0.2]	62.16%	110
	[1e-5, 15, [0.4, 0.6], 5e-5, 140, 1.0, 0.7, 0.0]	65.97%	140
VGG13	[5e-4, 5, [0.6, 0.8], 5e-5, 40, 1.0, 0.5, 0.1]	63.23%	40
	[1e-5, 10, [0.6, 0.8], 5e-5, 80, 0.95, 0.5, 0.3]	63.59%	80
	[5e-4, 5, [0.4, 0.6], 5e-5, 90, 0.95, 0.5, 0.0]	66.56%	90
	[1e-5, 5, [0.4, 0.6], 1e-6, 130, 1.0, 0.6, 0.4]	67.72%	130
VGG16	[5e-4, 5, [0.6, 0.8], 1e-5, 40, 1.0, 0.5, 0.0]	55.23%	40
	[5e-4, 5, [0.6, 0.8], 5e-5, 60, 0.99, 0.5, 0.0]	60.9%	60
	[1e-5, 10, [0.4, 0.6], 5e-6, 80, 1.0, 0.6, 0.2]	60.72%	80
	[1e-5, 10, [0.4, 0.6], 5e-6, 140, 1.0, 0.6, 0.2]	63.23%	140
RESNET20	[1e-4, 10, [0.4, 0.6], 5e-5, 40, 1.0, 0.5, 0.1]	60.75%	40
	[1e-5, 5, [0.6, 0.8], 1e-6, 70, 1.0, 0.5, 0.0]	61.04%	70
	[5e-5, 10, [0.4, 0.6], 5e-5, 80, 1.0, 0.5, 0.2]	52.43%	80

evaluating the performance of the proposed approach on multiple SNN algorithms on standard and non-standard datasets, we signify its versatility in optimizing the performance of SNNs. Through these numerous examples, we also achieve one of the key goals of this approach, which is to help close the gap in performance between ANNs and SNNs for resource-constrained environments without compromising the practicality of utilizing SNNs.

Table 8.6.: Comparison of the results of our Hierarchical-PABO framework with other SNN models on CIFAR10 dataset

Work	Network Type	Accuracy (%)	Time-step
CIFAR10 [37]			
SPIKE-NORM [18]	Deep CNN Conversion, VGG16	91.55	2500
Rueckauer [115]	Deep Binary Weight CNN Conversion, BinaryNet	90.85	400
Wu Direct Training [135]	Deep SNN	90.53	12
Lee Spike-Based BP [136]	Deep SNN, VGG9 (ResNet11)	90.45 (90.95)	100 (100)
HYBRID [17]	Hybrid Training, VGG16	91.13 (92.02)	100 (200)
This Work	**HP Optimized HYBRID, VGG9 (VGG13)**	**91.53 (91.45)**	**90 (80)**

9. DISCUSSION AND FUTURE WORK

In this thesis, we propose a novel multi-objective optimization framework based on hierarchical Bayesian optimization and agent-based modeling (Hierarchical-PABO). With its one of a kind structure, and simple yet effective underlying mathematics, we are able to predict a Pareto frontier of a multi-objective hyperparameter optimization for both non-spiking and spiking neural network systems with only few evaluations. This framework paves the way to further analyze and study sensitivity and resiliency of the system due to the changes of the hyperparameters. In addition, this optimization framework is compatible with various training algorithms, applications, and underlying accelerators. This include, but not limited to, convolutional neural networks in traditional deep learning, binary networks, evolutionary, and back-propagation based in spiking domain training algorithms, CMOS and beyond-CMOS accelerators, as well as control and classification applications.

The main limitation of Hierarchical-PABO is scalability and ability to parallelize the approach. The goal of Hierarchical-PABO is predicting the Pareto region for a search space with reasonable ranges for the hyperparameters and with only few evaluations and we do not want to compete with all NAS-based approaches that search the entire search space with massive computational resource requirements. However, improving scalability of Hierarchical-PABO paves the way for incorporating the technique in different frameworks with multiple layers of optimization problems and hyperparameters.

For future work, we intend to fully integrate the Hierarchical-PABO approach into the TENNLab neuromorphic framework by [114], so that it can seamlessly determine hyperparameters for the neuromorphic framework user. Within that framework, we also intend to apply this hyperparameter framework to other neuromorphic implementations that are supported and other applications, including a variety of control

applications (like those described by [137]) and other classification tasks. We also plan to investigate an implementation of Hierarchical-PABO for high-performance computers, such as Oak Ridge National Laboratory's Summit supercomputer.

9.1 Broader Impact

The overall goal of this work is to provide a framework that can automatically design models for new tasks that are more accurate and more efficient in neuromorphic hardware. There are a variety of potential positive and negative outcomes of this approach. One positive outcome is that this approach can reduce the barrier of entry in utilizing these SNN algorithms for researchers who are less familiar with SNN approaches. Another positive outcome is that this approach can be used to develop AI models that are more energy efficient in their deployment, which can potentially lead to a decrease in the energy usage and carbon footprint of AI models that are deployed for real-world applications.

We believe negative implications of our work largely stem from the amplification of negatives embodied in existing neural network methods. One such example of this would be on the tuning of model parameters to create networks that are more strongly fit to datasets which have underlying biases that enable the exploitation of underrepresented groups in society. Another concern is that efficient hyperparameter optimization techniques may lead to a loss of intuition behind model design that could reduce understanding of neural network components and functionality in a way that researchers are less likely to be equipped to understand model drawbacks as well as less likely to make future structural improvements. A strong focus on fundamentals during machine learning education may mitigate the extent of this concern. Alternatively, avoiding tedious human in the loop hyperparameter testing frees up research time to look into deeper questions for machine learning and neural network architectures.

REFERENCES

REFERENCES

[1] Y.-H. Chen and e. al., "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016, pp. 367–379.

[2] N. P. Jouppi and e. al., "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA'17. New York, NY, USA: ACM, 2017, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/3079856.3080246

[3] S. Venkataramani and e. al., "Scaledeep: A scalable compute architecture for learning and evaluating deep networks," in *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2. ACM, 2017, pp. 13–26.

[4] J. Fowers and e. al., "A configurable cloud-scale dnn processor for real-time ai," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*. IEEE Press, 2018, pp. 1–14.

[5] A. Shafiee and e. al., "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[6] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA'16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 27–39. [Online]. Available: https://doi.org/10.1109/ISCA.2016.13

[7] A. Ankit, I. E. Hajj, S. R. Chalamalasetti, G. Ndu, M. Foltin, R. S. Williams, P. Faraboschi, W.-m. W. Hwu, J. P. Strachan, K. Roy *et al.*, "Puma: A programmable ultra-efficient memristor-based accelerator for machine learning inference," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2019, pp. 715–731.

[8] S. G. Ramasubramanian and e. al., "Spindle: Spintronic deep learning engine for large-scale neuromorphic computing," in *Proceedings of the 2014 international symposium on Low power electronics and design*. ACM, 2014, pp. 15–20.

[9] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," *arXiv preprint arXiv:1705.06963*, 2017.

[10] D. Monroe, "Neuromorphic computing gets ready for the (really) big time," *Communications of the ACM*, vol. 57, no. 6, pp. 13–15, 2014.

[11] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, vol. 20, no. 55, pp. 1–21, 2019.

[12] B. Reagen and e. al., "A case for efficient accelerator design space exploration via bayesian optimization," in *Low Power Electronics and Design (ISLPED, 2017 IEEE/ACM International Symposium on*. IEEE, 2017, pp. 1–6.

[13] Y. Liu, Y. Jin, and P. Li, "Online adaptation and energy minimization for hardware recurrent spiking neural networks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 14, no. 1, pp. 1–21, 2018.

[14] 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.

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

[16] W. Severa, C. M. Vineyard, R. Dellana, S. J. Verzi, and J. B. Aimone, "Whetstone: A method for training deep artificial neural networks for binary communication," *arXiv preprint arXiv:1810.11521*, 2018.

[17] 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.

[18] 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.

[19] J. P. Mitchell, M. E. Dean, G. R. Bruer, J. S. Plank, and G. S. Rose, "Danna 2: Dynamic adaptive neural network arrays," in *Proceedings of the International Conference on Neuromorphic Systems*. ACM, 2018, p. 10.

[20] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[21] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam *et al.*, "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*, vol. 34, no. 10, pp. 1537–1557, 2015.

[22] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[23] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia *et al.*, "Chamnet: Towards efficient network design through platform-aware model adaptation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 398–11 407.

[24] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[25] M. Parsa, A. Ankit, A. Ziabari, and K. Roy, "Pabo: Pseudo agent-based multi-objective bayesian hyperparameter optimization for efficient neural accelerator design," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.

[26] M. Parsa, J. P. Mitchell, C. D. Schuman, R. M. Patton, T. E. Potok, and K. Roy, "Bayesian-based hyperparameter optimization for spiking neuromorphic systems," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 4472–4478.

[27] M. Parsa, C. D. Schuman, D. C. Rose, B. Kay, J. P. Mitchell, S. R. Young, R. Dellana, W. Severa, T. E. Potok, K. Roy *et al.*, "Hyperparameter optimization in binary communication networks for neuromorphic deployment," *arXiv preprint arXiv:2005.04171*, 2020.

[28] M. Parsa, J. P. Mitchell, C. D. Schuman, R. M. Patton, T. E. Potok, and K. Roy, "Bayesian multi-objective hyperparameter optimization for accurate, fast, and efficient neural network accelerator design," *Frontiers in Neuroscience*, vol. 14, p. 667, 2020.

[29] M. Parsa, C. Schuman, N. Rathi, A. Ziabari, D. Rose, J. P. Mitchell, T. Johnston, B. Kay, S. Young, and K. Roy, "Accurate and accelerated spiking neural networks trained with back-propagation: A bayesian hyperparameter pareto optimization approach." In Preparation, 2020.

[30] T. Okabe, Y. Jin, and B. Sendhoff, "A critical survey of performance indices for multi-objective optimisation," in *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, vol. 2. IEEE, 2003, pp. 878–885.

[31] A. Krizhevsky and e. al., "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[32] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[33] G. Chakma, M. M. Adnan, A. R. Wyer, R. Weiss, C. D. Schuman, and G. S. Rose, "Memristive mixed-signal neuromorphic systems: Energy-efficient learning at the circuit-level," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 125–136, 2017.

[34] M.-E. Nilsback and A. Zisserman, "A visual vocabulary for flower classification," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2. IEEE, 2006, pp. 1447–1454.

[35] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, vol. 2, 2010.

[36] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[37] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[38] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[39] J. J. Reynolds, J. S. Plank, C. D. Schuman, G. R. Bruer, A. W. Disney, M. E. Dean, and G. S. Rose, "A comparison of neuromorphic classification tasks," in *Proceedings of the International Conference on Neuromorphic Systems*. ACM, 2018, p. 12.

[40] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza *et al.*, "A low power, fully event-based gesture recognition system," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7243–7252.

[41] F. Gomez, J. Schmidhuber, and R. Miikkulainen, "Efficient non-linear control through neuroevolution," in *European Conference on Machine Learning*. Springer, 2006, pp. 654–662.

[42] J. P. Mitchell, G. Bruer, M. E. Dean, J. S. Plank, G. S. Rose, and C. D. Schuman, "Neon: Neuromorphic control for autonomous robotic navigation," in *2017 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*. IEEE, 2017, pp. 136–142.

[43] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[44] C. M. Vineyard, R. Dellana, J. B. Aimone, F. Rothganger, and W. M. Severa, "Low-power deep learning inference using the spinnaker neuromorphic platform," in *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*, 2019, pp. 1–7.

[45] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[46] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[47] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in neural information processing systems*, 2016, pp. 2074–2082.

[48] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "Netadapt: Platform-aware neural network adaptation for mobile applications," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 285–300.

[49] J. Jin, A. Dundar, and E. Culurciello, "Flattened convolutional neural networks for feedforward acceleration," *arXiv preprint arXiv:1412.5474*, 2014.

[50] M. Wang, B. Liu, and H. Foroosh, "Factorized convolutional neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 545–553.

[51] P. Panda, A. Sengupta, and K. Roy, "Conditional deep learning for energy-efficient and enhanced pattern recognition," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 475–480.

[52] M. Parsa, P. Panda, S. Sen, and K. Roy, "Staged inference using conditional deep learning for energy efficient real-time smart diagnosis," in *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2017, pp. 78–81.

[53] T. Bohnstingl, F. Scherr, C. Pehle, K. Meier, and W. Maass, "Neuromorphic hardware learns to learn," *Frontiers in neuroscience*, vol. 13, 2019.

[54] S. Esser, P. Merolla, J. Arthur, A. Cassidy, R. Appuswamy, A. Andreopoulos, D. Berg, J. McKinstry, T. Melano, D. Barch *et al.*, "Convolutional networks for fast, energy-efficient neuromorphic computing. 2016," *Preprint on ArXiv. http://arxiv. org/abs/1603.08270. Accessed*, vol. 27, 2016.

[55] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Backpropagation for energy-efficient neuromorphic computing," in *Advances in neural information processing systems*, 2015, pp. 1117–1125.

[56] W. Severa, C. M. Vineyard, R. Dellana, S. J. Verzi, and J. B. Aimone, "Training deep neural networks for binary communication with the whetstone method," *Nature Machine Intelligence*, vol. 1, no. 2, pp. 86–94, 2019.

[57] S. Schmitt, J. Klähn, G. Bellec, A. Grübl, M. Guettler, A. Hartel, S. Hartmann, D. Husmann, K. Husmann, S. Jeltsch *et al.*, "Neuromorphic hardware in the loop: Training a deep spiking network on the brainscales wafer-scale system," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 2227–2234.

[58] M. Koo, G. Srinivasan, Y. Shim, and K. Roy, "sbsnn: Stochastic-bits enabled binary spiking neural network with on-chip learning for energy efficient neuromorphic computing at the edge," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–10, 2020.

[59] P. Panda, A. Sengupta, and K. Roy, "Energy-efficient and improved image recognition with conditional deep learning," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, p. 33, 2017.

[60] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[61] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856.

[62] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5687–5695.

[63] P. Ferré, F. Mamalet, and S. J. Thorpe, "Unsupervised feature learning with winner-takes-all based stdp," *Frontiers in computational neuroscience*, vol. 12, p. 24, 2018.

[64] J. C. Thiele, O. Bichler, and A. Dupret, "Event-based, timescale invariant unsupervised online deep learning with stdp," *Frontiers in computational neuroscience*, vol. 12, p. 46, 2018.

[65] P. U. Diehl, G. Zarrella, A. Cassidy, B. U. Pedroni, and E. Neftci, "Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2016, pp. 1–8.

[66] 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.

[67] A. R. Voelker, D. Rasmussen, and C. Eliasmith, "A spike in performance: Training hybrid-spiking neural networks with quantized activation functions," *arXiv preprint arXiv:2002.03553*, 2020.

[68] 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.

[69] S. M. Bohte, J. N. Kok, and H. La Poutre, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1-4, pp. 17–37, 2002.

[70] Y. Yi, Y. Liao, B. Wang, X. Fu, F. Shen, H. Hou, and L. Liu, "Fpga based spike-time dependent encoder and reservoir design in neuromorphic computing processors," *Microprocessors and Microsystems*, vol. 46, pp. 175–183, 2016.

[71] C. D. Schuman, J. S. Plank, G. Bruer, and J. Anantharaj, "Non-traditional input encoding schemes for spiking neuromorphic systems," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–10.

[72] P. Date, J. A. Hendler, and C. D. Carothers, "Design index for deep neural networks," *Procedia Computer Science*, vol. 88, pp. 131–138, 2016.

[73] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 437–478.

[74] J. Bergstra and e. al., "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[75] J. S. Bergstra and e. al., "Algorithms for hyper-parameter optimization," in *Advances in neural information processing systems*, 2011, pp. 2546–2554.

[76] J. Bergstra, B. Komer, C. Eliasmith, and D. Warde-Farley, "Preliminary evaluation of hyperopt algorithms on hpolib," in *ICML workshop on AutoML*, 2014.

[77] P. Balaprakash, M. Salim, T. Uram, V. Vishwanath, and S. Wild, "Deephyper: Asynchronous hyperparameter search for deep neural networks," in *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*. IEEE, 2018, pp. 42–51.

[78] I. Ilievski, T. Akhtar, J. Feng, and C. A. Shoemaker, "Efficient hyperparameter optimization for deep learning algorithms using deterministic rbf surrogates," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[79] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[80] P. Liu, M. D. El Basha, Y. Li, Y. Xiao, P. C. Sanelli, and R. Fang, "Deep evolutionary networks with expedited genetic algorithms for medical image denoising," *Medical image analysis*, vol. 54, pp. 306–315, 2019.

[81] A. Shaw, D. Hunter, F. Landola, and S. Sidhu, "Squeezenas: Fast neural architecture search for faster semantic segmentation," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2019, pp. 0–0.

[82] Y. Weng, T. Zhou, Y. Li, and X. Qiu, "Nas-unet: Neural architecture search for medical image segmentation," *IEEE Access*, vol. 7, pp. 44 247–44 257, 2019.

[83] Y. Chen, T. Yang, X. Zhang, G. Meng, X. Xiao, and J. Sun, "Detnas: Backbone search for object detection," in *Advances in Neural Information Processing Systems*, 2019, pp. 6638–6648.

[84] S. R. Young, D. C. Rose, T. Johnston, W. T. Heller, T. P. Karnowski, T. E. Potok, R. M. Patton, G. Perdue, and J. Miller, "Evolving deep networks using hpc," in *Proceedings of the Machine Learning on HPC Environments*, 2017, pp. 1–7.

[85] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 19–34.

[86] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," *arXiv preprint arXiv:1802.03268*, 2018.

[87] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[88] P. Balaprakash, R. Egele, M. Salim, S. Wild, V. Vishwanath, F. Xia, T. Brettin, and R. Stevens, "Scalable reinforcement-learning-based neural architecture search for cancer deep learning research," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–33.

[89] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[90] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 267–278.

[91] L. L. Zhang, Y. Yang, Y. Jiang, W. Zhu, and Y. Liu, "Fast hardware-aware neural architecture search," 2019.

[92] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.

[93] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv preprint arXiv:1812.00332*, 2018.

[94] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 734–10 742.

[95] S. V. K. Srinivas, H. Nair, and V. Vidyasagar, "Hardware aware neural network architectures using fbnet," *arXiv preprint arXiv:1906.07214*, 2019.

[96] D. Stamoulis, E. Cai, D.-C. Juan, and D. Marculescu, "Hyperpower: Power- and memory-constrained hyper-parameter optimization for neural networks," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 19–24.

[97] D. Marculescu, D. Stamoulis, and E. Cai, "Hardware-aware machine learning: modeling and optimization," in *Proceedings of the International Conference on Computer-Aided Design*. ACM, 2018, p. 137.

[98] L. Salt, D. Howard, G. Indiveri, and Y. Sandamirskaya, "Differential evolution and bayesian optimisation for hyper-parameter selection in mixed-signal neuromorphic circuits applied to uav obstacle avoidance," *arXiv preprint arXiv:1704.04853*, 2017.

[99] J. Kim and D.-S. Kim, "Competitive hyperparameter balancing on spiking neural network for a fast, accurate and energy-efficient inference," in *International Symposium on Neural Networks*. Springer, 2018, pp. 44–53.

[100] E. Brochu, V. M. Cora, and N. De Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *arXiv preprint arXiv:1012.2599*, 2010.

[101] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, and K. Leyton-Brown, "Towards an empirical foundation for assessing bayesian optimization of hyperparameters," in *NIPS workshop on Bayesian Optimization in Theory and Practice*, vol. 10, 2013, p. 3.

[102] H. J. Kushner, "A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise," *Journal of Basic Engineering*, vol. 86, no. 1, pp. 97–106, 1964.

[103] A. D. Bull, "Convergence rates of efficient global optimization algorithms," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2879–2904, 2011.

[104] D. R. Jones, "A taxonomy of global optimization methods based on response surfaces," *Journal of global optimization*, vol. 21, no. 4, pp. 345–383, 2001.

[105] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global optimization*, vol. 13, no. 4, pp. 455–492, 1998.

[106] T. L. Lai and H. Robbins, "Asymptotically efficient adaptive allocation rules," *Advances in applied mathematics*, vol. 6, no. 1, pp. 4–22, 1985.

[107] S. Agrawal and N. Goyal, "Thompson sampling for contextual bandits with linear payoffs," in *International Conference on Machine Learning*, 2013, pp. 127–135.

[108] J. M. Hernández-Lobato and e. al., "Predictive entropy search for efficient global optimization of black-box functions," in *Advances in neural information processing systems*, 2014, pp. 918–926.

[109] A. Ankit, A. Sengupta, and K. Roy, "Trannsformer: Neural network transformation for memristive crossbar-based neuromorphic system design," in *Proceedings of the 36th International Conference on Computer-Aided Design*. IEEE Press, 2017, pp. 533–540.

[110] A. P. Wieland, "Evolving neural network controllers for unstable systems," in *IJCNN-91-Seattle International Joint Conference on Neural Networks*, vol. 2. IEEE, 1991, pp. 667–673.

[111] M. G. Genton, "Classes of kernels for machine learning: A statistics perspective," *J. Mach. Learn. Res.*, vol. 2, p. 299–312, Mar. 2002.

[112] D. Querlioz, O. Bichler, and C. Gamrat, "Simulation of a memristor-based spiking neural network immune to device variations," in *The 2011 International Joint Conference on Neural Networks*. IEEE, 2011, pp. 1775–1781.

[113] J. V. Arthur and K. Boahen, "Learning in silicon: Timing is everything," in *Advances in neural information processing systems*, 2006, pp. 75–82.

[114] J. S. Plank, C. D. Schuman, G. Bruer, M. E. Dean, and G. S. Rose, "The tennlab exploratory neuromorphic computing framework," *IEEE Letters of the Computer Society*, vol. 1, no. 2, pp. 17–20, 2018.

[115] 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.

[116] E. Hunsberger and C. Eliasmith, "Training spiking deep networks for neuromorphic hardware," *arXiv preprint arXiv:1611.05141*, 2016.

[117] C. Lee, S. S. Sarwar, and K. Roy, "Enabling spike-based backpropagation in state-of-the-art deep neural network architectures," *arXiv preprint arXiv:1903.06379*, 2019.

[118] Y. Hao, X. Huang, M. Dong, and B. Xu, "A biologically plausible supervised learning method for spiking neural networks using the symmetric stdp rule," *Neural Networks*, vol. 121, pp. 387–395, 2020.

[119] A. Shrestha, H. Fang, Q. Wu, and Q. Qiu, "Approximating back-propagation for a biologically plausible local learning rule in spiking neural networks," in *Proceedings of the International Conference on Neuromorphic Systems*, 2019, pp. 1–8.

[120] Y. Jin, W. Zhang, and P. Li, "Hybrid macro/micro level backpropagation for training deep spiking neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 7005–7015.

[121] P. Machado, G. Cosma, and T. M. McGinnity, "Natcsnn: A convolutional spiking neural network for recognition of objects extracted from natural images," in *International Conference on Artificial Neural Networks*. Springer, 2019, pp. 351–362.

[122] J. Wu, Y. Chua, M. Zhang, G. Li, H. Li, and K. C. Tan, "A tandem learning rule for efficient and rapid inference on deep spiking neural networks."

[123] D. Roy, I. Chakraborty, and K. Roy, "Scaling deep spiking neural networks with binary stochastic activations," in *2019 IEEE International Conference on Cognitive Computing (ICCC)*. IEEE, 2019, pp. 50–58.

[124] F. Xing, Y. Yuan, H. Huo, and T. Fang, "Homeostasis-based cnn-to-snn conversion of inception and residual architectures," in *International Conference on Neural Information Processing*. Springer, 2019, pp. 173–184.

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

[126] J. Guerguiev, K. P. Kording, and B. A. Richards, "Spike-based causal inference for weight alignment," *arXiv preprint arXiv:1910.01689*, 2019.

[127] J. C. Thiele, O. Bichler, and A. Dupret, "Spikegrad: An ann-equivalent computation model for implementing backpropagation with spikes," *arXiv preprint arXiv:1906.00851*, 2019.

[128] 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, 2019, pp. 1311–1318.

[129] S. R. Young, P. Devineni, M. Parsa, J. T. Johnston, B. Kay, R. M. Patton, C. D. Schuman, D. C. Rose, and T. E. Potok, "Evolving energy efficient convolutional neural networks," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 4479–4485.

[130] J. Kaiser, A. Friedrich, J. C. V. Tieck, D. Reichard, A. Roennau, E. Neftci, and R. Dillmann, "Embodied Neuromorphic Vision with Event-Driven Random Backpropagation," *arXiv:1904.04805 [cs]*, May 2019, arXiv: 1904.04805. [Online]. Available: http://arxiv.org/abs/1904.04805

[131] J. Kaiser, H. Mostafa, and E. Neftci, "Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE)," *Frontiers in Neuroscience*, vol. 14, May 2020. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2020.00424/full

[132] H. Fang, A. Shrestha, Z. Zhao, and Q. Qiu, "Exploiting Neuron and Synapse Filter Dynamics in Spatial Temporal Learning of Deep Spiking Neural Network," *arXiv:2003.02944 [cs, stat]*, Feb. 2020. [Online]. Available: http://arxiv.org/abs/2003.02944

[133] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

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

[135] Y. Wu, L. Deng, G. Li, J. Zhu, Y. Xie, and L. Shi, "Direct Training for Spiking Neural Networks: Faster, Larger, Better," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 1311–1318, Jul. 2019. [Online]. Available: https://aaai.org/ojs/index.php/AAAI/article/view/3929

[136] 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, Feb. 2020. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2020.00119/full

[137] J. S. Plank, C. Rizzo, K. Shahat, G. Bruer, T. Dixon, M. Goin, G. Zhao, J. Anantharaj, C. D. Schuman, M. E. Dean, G. S. Rose, N. C. Cady, and J. Van Nostrand, "The TENNLab suite of LIDAR-based control applications for recurrent, spiking, neuromorphic systems," in *44th Annual GOMACTech Conference*, Albuquerque, March 2019. [Online]. Available: http://neuromorphic.eecs.utk.edu/raw/files/publications/2019-Plank-Gomac.pdf

VITA

VITA

Maryam Parsa is PhD candidate at Center for Brian-Inspired Computing (C-BRIC), Electrical Engineering department, Purdue University under supervision of Prof. Kaushik Roy. She is a recipient of Intel/SRC PhD Fellowship for the duration of her PhD studies. Maryam also interned as a Graduate Researcher at Intel Corporation for two consecutive summers, and at ORNL as an ASTRO intern for more than a year. Her primary research interests are Bayesian optimization, hyperparameter optimization, software-hardware co-design, nanoelectronics, and neuromorphic computing.