Hyperparameter Optimization in Binary Communication Networks for Neuromorphic Deployment

Maryam Parsa
Purdue University
West Lafayette, Indiana, USA
mparsa@purdue.edu

Catherine D. Schuman
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
schumancd@ornl.gov

Prasanna Date
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
datepa@ornl.gov

Derek C. Rose
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
rosedc@ornl.gov

Bill Kay
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
kaybw@ornl.gov

J. Parker Mitchell
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
mitchelljp1@ornl.gov

Steven R. Young
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
youngsr@ornl.gov

Ryan Dellana
Sandia National Laboratories
Albequerque, New Mexico, USA
rdellan@sandia.gov

William Severa
Sandia National Laboratories
Albequerque, New Mexico, USA
wmsever@sandia.gov

Thomas E. Potok
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
potokte@ornl.gov

Kaushik Roy
Purdue University
West Lafayette, Indiana, USA
kaushik@purdue.edu

***Abstract*—Training neural networks for neuromorphic deployment is non-trivial. There have been a variety of approaches proposed to adapt back-propagation or back-propagation-like algorithms appropriate for training. Considering that these networks often have very different performance characteristics than traditional neural networks, it is often unclear how to set either the network topology or the hyperparameters to achieve optimal performance. In this work, we introduce a Bayesian approach for optimizing the hyperparameters of an algorithm for training binary communication networks that can be deployed to neuromorphic hardware. We show that by optimizing the hyperparameters on this algorithm for each dataset, we can achieve improvements in accuracy over the previous state-of-the-art for this algorithm on each dataset (by up to 15 percent). This jump in performance continues to emphasize the potential when converting traditional neural networks to binary communication applicable to neuromorphic hardware.**

***Index Terms*—hyperparameter optimization, neural networks, Bayesian optimization, neuromorphic**

I. Introduction

Neuromorphic computing offers the promise of very low power hardware implementations of machine learning, along with potential opportunities for new ways to perform computing with a fundamentally different type of architecture [1]. Neuromorphic hardware platforms are being developed by both industry and academic groups that have largely focused on providing an implementation of traditional spiking neural networks. To date there has been relatively little focus on the development of algorithms which aim to effectively leverage neuromorphic systems for spiking networks [2].

One common class of algorithms of this type are based on traditional back-propagation-trained algorithms, such as those used for traditional neural network training but have been adapted to accommodate for neuromorphic deployment. When these algorithms are applied to networks that can be deployed on spiking neuromorphic systems, hyperparameters can have a tremendous impact on the performance of the network. This challenge is well known in traditional neural network training [3]–[5], and often these hyperparameters are determined through a combination of trial-and-error, intuition, and random search [6]. However, it is not clear how these methods should be adapted to accommodate for the changes in the training algorithm. Moreover, there are often even more

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

new hyperparameters for these adapted approaches to back-propagation-like algorithms.

One such algorithm we choose to investigate is Whetstone [7]. 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 Bayesian hyperparameter optimization [5], [8], [9] to find optimal hyperparameters for the Whetstone algorithm on four different datasets. We compare our results to the previously published Whetstone results from [7] 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. Finally, we discuss how this approach can be extended, both in future work with the Whetstone approach as well as other training approaches for Spiking Neural Networks (SNNs), and neuromorphic systems. 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.

The main contributions of this work are:

- A demonstration of the effect of hyperparameters on a training algorithm (Whetstone) that trains neural networks with binary communication.
- A Bayesian optimization approach to optimize Whetstone's hyperparameters.
- State-of-the-art results for Whetstone on four commonly used datasets.

II. Background and Related Work

We first review the various approaches used for optimizing the hyperparameters of deep learning models. Hyperparameter optimization for neural networks used to be largely governed by rules of thumb [10]. Bengio outlines some of these rules and practical guidelines for efficiently training large-scale deep neural networks [11]. Bergstra and Bengio show that random search outperforms grid search and manual search for hyperparameter optimization and has good theoretical guarantees and empirical evidence [6]. Continuing along this line of research, Bergstra *et al.* present greedy sequential algorithms for hyperparameter optimization and show that their performance is better than that of random search [12].

Bayesian-based approaches have also been used for optimizing the hyperparameters of deep neural networks. Bergstra *et al.* show that algorithms based specifically on the Gaussian process are the most call-efficient for hyperparameter optimization of deep neural networks [13]. Snoek *et al.* describe algorithms that take into consideration the variable costs of learning experiments and show that the resulting set of hyperparameters returned by these algorithms can match or even surpass human expert-level optimizations [14]. Zhang *et al.* propose a search algorithm based on Bayesian optimization while training deep convolutional neural networks on the PASCAL VOC 2007 and 2012 datasets [15]. Balaprakash *et al.* develop DeepHyper, which is a Python package that leverages the Balsam workflow and provides an interface for implementation and study of scalable hyperparameter search methods [16]. Ilievski *et al.* propose a deterministic and efficient method for hyperparameter optimization using radial basis function as the error surrogate in Bayesian-based methods called HORD, and demonstrate its effectiveness on MNIST and CIFAR-10 datasets [17].

Evolutionary optimization techniques have also been used for hyperparameter optimization in the literature. Miikkulainen *et al.* propose CoDeepNEAT, which is a method that extends the conventional neuro-evolution methods to topology, components and hyperparameters and achieves performance comparable to the best human-optimized networks [18]. Young *et al.* propose a scalable evolutionary optimization method and demonstrate its efficacy on varied datasets [19]. Shafiee *et al.* propose a genetic algorithm-like method for hyperparameter optimization, which not only achieves the state-of-the-art performance, but is also seen to use up to $48\times$ less synapses in doing so [20]. Liang *et al.* evaluate several hyperparameter optimization methods that evolve the architecture of deep neural networks and demonstrate that a synergetic approach for evolving custom routings with evolved, shared modules is very powerful, and significantly improves the state-of-the-art performance on the Omniglot character recognition domain [21]. In addition to these evolutionary optimization-based methods, reinforcement learning has also been used for hyperparameter optimization of deep neural networks.

While the above approaches catered to deep neural networks, several hyperparameter optimization methods have been used in the literature for optimizing architectures or hyperparameters specifically pertaining to neuromorphic computing. Schuman *et al.* present several approaches for encoding numerical values as spikes for spiking neural networks, hierarchically combine them to form more complex encoding schemes, and demonstrate their usability on four different applications [22]. Salt *et al.* use differential evolution (DE) and self-adaptive differential evolution algorithms (SADE) 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 [23]. Schuman *et al.* develop an evo-

lutionary optimization based training framework for spiking neural network and neuromorphic architectures, and test this approach on four datasets [24]. Kim and Kim apply a Neuro-evolutionary algorithm to optimize the hyperparameters of spiking neural networks and show that the model trained using this approach outperforms all other models [25]. Parsa *et al.* demonstrate effectiveness of Bayesian approach for hyperparameter optimization for spiking neuromorphic systems [8].

In this work we focus on Bayesian hyperparameter optimization for binary communication network for neuromorphic deployment, Whetstone [7]. The powerful and yet effective underlying mathematics of Bayesian approach, paves the way to quickly estimate an expensive objective function such as network performance.

III. Methods

In this section we briefly introduce Whetstone and Bayesian optimization approaches. The former is an approach for training binary communication networks for neuromorphic deployment, and the latter is an optimization tool for problems with black-box and expensive objective functions. Detailed description on each of these techniques can be found at [7], and [9], [26], respectively.

A. 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.

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 output 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.* [7] 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.

B. Bayesian Optimization

To systematically take the human out of the loop in finding the optimum set of hyperparameter for an expensive, black-box objective function such as training a neural networks, several approaches are introduced in the literature and already discussed in section II. Bayesian optimization is one of the primary approaches for these types of problems due to its flexible and powerful underlying mathematics [26].

As summarized by [5], [9], [26], Bayesian optimization is a sequential technique that aims at predicting the unknown objective function with limited and yet effective observations. For our hyperparameter optimization problem, the unknown objective function is the classification performance of Whetstone, and observations are the performance values (accuracies) for a set of hyperparameters in each iteration. We start the optimization process with two random initial set of hyperparameters, and for each one of them evaluate the performance of Whetstone network. This will create the first set of observations. In the Bayesian optimization technique for each iteration, we estimate a Gaussian distribution over the available observations (called the *prior* distribution, current beliefs). We update the current beliefs with a new observation and estimate the *posterior* distribution. With enough observations, the *posterior* distribution is the prediction of the unknown, expensive objective function we are optimizing.

In this search technique, the new observations are chosen based on optimizing a surrogate model, called the *acquisition function*. This function is built upon the *posterior* distribution at each iteration. There are different policies introduced in the literature to calculate this function such as improved-based, optimistic, and information-based policies. Each one of these approaches calculate the *acquisition function* to explore and exploit the search space. The maximum point of this function is the best next set of hyperparameter to observe in the next iteration. More details on Bayesian optimization can be found in [26]. In this work, we are dealing with a single objective Bayesian optimization problem [27], as we aim at finding the optimum set of hyperparameter that maximizes the Whetstone performance.

IV. 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. Details of the hyperparameters that are optimized and their corresponding ranges are given in each case study as follows.

A. Datasets

Our methods were benchmarked on four labeled image data sets commonly used to demonstrate efficacy of supervised image classification protocols. The *MNIST [28]* 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 [29]* 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 [30]* 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* [30] dataset is the same as the CIFAR-10 dataset, except with 100 classes. Each class represents an equal proportion of the total dataset.

B. Case Study One

For case study one, we select a small search space for hyperparameters given in Table I for classification task on CIFAR-100 dataset [30]. 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 this case study, the fixed hyperparameters that are not included in the optimization search and their corresponding values are given in Table IV-B.

TABLE I
Case study one: Evaluated hyperparameters

Hyperparameter	Range
Optimizer learning rate (lr)	0.0001, 1
Optimizer decay (dec)	1e-8, 1e-6
Sharpener starting epoch (sh_st)	15, 25
Sharpener duration (sh_du)	3, 7
Sharpener intermission (sh_int)	2, 5
Conv. layer 1, filter size (filter 1)	3, 7
Conv. layer 1, # of features (feat 1)	64, 128
Dense layer, # of features (dense)	256, 1024
Search space size: 256	

Hyperparameter	Value
Optimizer rho	0.9
Optimizer epsilon	1e-6
Optimizer type	Adadelta
Gaussian noise layer	Without noise
Batch normalizer conv. layers momentum	0.95
Batch normalizer dense layer momentum	0.95
Batch normalizer epsilon	1e-3
Batch normalizer center	True
Batch normalizer scale	True
Conv. layer 2, filter size	5
Conv. layer 3, filter size	3
Conv. layer 2, # of features	256
Conv. layer 3, # of features	512

In Figure 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 I.

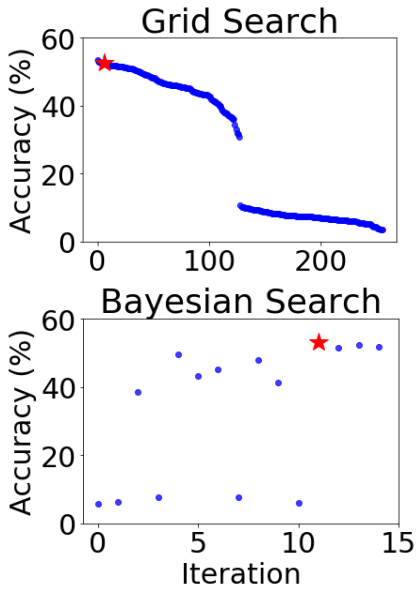

Fig. 1. Case Study One: Comparing grid search and Bayesian hyperparameter optimization for hyperparameters given in Table I with search space size of 256

TABLE II
CASE STUDY ONE: DETAILS OF BAYESIAN HYPERPARAMETER OPTIMIZATION

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**

After only 15 evaluations of Whetstone [7], 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 1, and leads to accuracy of 53.13%, which outperforms the 38% accuracy reported in Whetstone original results [7]. 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". The hyperparameter values at each iteration are given in Table II.

We also perform further analysis on the changes of hyperparameters and their effect on the final accuracy of the network. For example, with changing the sharpener starting epoch from 15 to 25, its duration from 3 to 7, and the filter size 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 II). This table also shows that some hyperparameters play a vital role on the final performance of the system, such as learning rate.

C. Case Study Two

In case study two, we increase the search space size to 398,131,200 combinations of hyperparameters shown in Table III. 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 [7] hyperparameters, and the hyperparameters that belong to the neural network architecture itself, such as filter sizes or the number of features to extract.

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. For each hyperparameter, all values in Table III are based on acceptable and reasonable ranges.

For the hyperparameters given in Table III, the performance of the hyperparameter optimization approach for Whetstone technique for four different dataset of MNIST [28], Fashion-MNIST [29], CIFAR-10 [30], and CIFAR-100 [30] as well as their corresponding optimum hyperparameter values are given in Table IV. 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 larger dastaset 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%. Table V 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 hyperpa-

TABLE III
CASE STUDY TWO: EVALUATED HYPERPARAMETERS

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

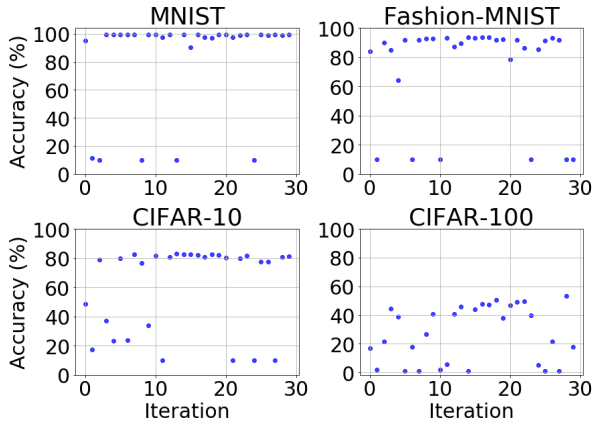

Fig. 2. Case study two: Performance value (accuracy (%) for each hyperparameter optimization search iteration for MNIST, Fashion-MNIST, CIFAR-10, and CIFAR-100 dataset with the hyperparameters given in Table III

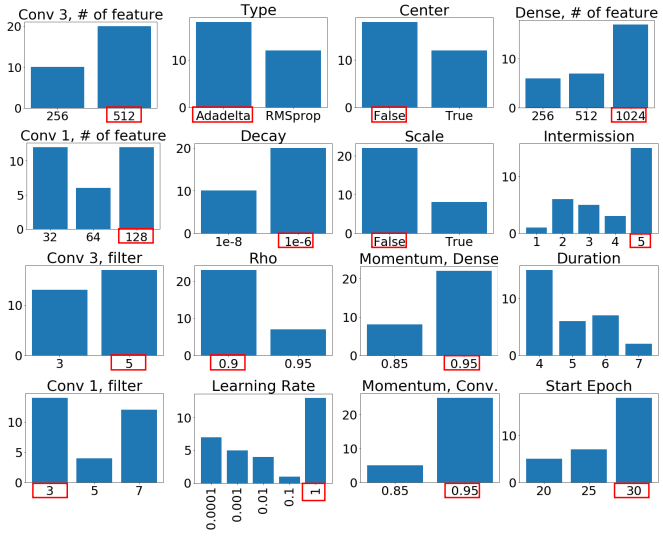

Fig. 3. Case study two: Histograms of each hyperparameter value for CIFAR-100 dataset experiment for the 30 iterations of the optimization search process

rameter optimization framework, we can drastically 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.

Figure 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 3, we show the frequency of selecting each value for some of the hyperparameters given in Table III for CIFAR-100. The x-axis 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 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.

Table VI 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 III, and the performances shown in Figure 2 for this dataset. These experiments are chosen among the 30 iterations of the Bayesian optimization search. The first three experiments in Table VI show that with quite different combinations of hyperparameters 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.

V. Discussion and Conclusion

In this work, we introduce a hyperparameter optimization approach on Whetstone for training neural networks that can be deployed to neuromorphic hardware. 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

TABLE IV
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%**

systems in the future. The best results on the different datasets are shown with different parameters in Table IV. 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 optimization approach such as MENNDL [19] 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 the MENNDL framework to better understand when sharpening is useful and hopefully discover higher performance network designs that may better leverage binarized operations.

In [47], Whetstone is deployed on SpiNNaker [48], 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 [48] or Loihi [49]. As observed in [47], 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 [5], [9] and the fitness function used within MENNDL [50] 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.

ACKNOWLEDGMENT

The research was funded in part by Center for Brain-Inspired Computing Enabling Autonomous Intelligence (C-BRIC), one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, the National Science Foundation, Intel Corporation and Vannevar Bush Faculty Fellowship.

This material is also based in part upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under contract number DE-AC05-00OR22725, and in part by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

REFERENCES

[1] J. B. Aimone, K. E. Hamilton, S. Mniszewski, L. Reeder, C. D. Schuman, and W. M. Severa, "Non-neural network applications for spiking neuromorphic hardware," in *3rd International Workshop on Post-Moore's Era Supercomputing (PMES 2018), Dallas, TX*, 2018.

[2] 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.

TABLE V
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. [31]	6-layer CNN	Temporal credit assignment for backpropagating (BP) error	99.36	-	-	-
Rueckauer et al. [32]	8-layer CNN	Offline, ANN-to-SNN conversion	99.44	-	90.85	-
Hunsberger et al. [33]	AlexNet	Offline, ANN-to-SNN conversion	99.12	-	83.54	55.13
Lee et al. [34]	ResNet-11	Spike-based backpropagating	99.59	-	90.95	-
Hao et al. [35]	3-layer FF SNN	Symmetric STDP Rule	96.73	85.31	-	-
Shrestha et al. [36]	4-layer NN	Error Modulated STDP with symmetric weights	97.3	86.1	-	-
Jin et al [37]	6-layer CNN	Direct macro/micro BP	99.49	-	-	-
Sengupta et al. [38]	VGG-16	Offline, ANN-to-SNN conversion	-	-	91.55	-
Machado et al. [39]	3-layer NatCSNN	Two-phase (unsupervised STDP, ReSuMe supervised)	-	-	84.7	-
Wu et al. [40]	CIFARNet	SNN and ANN with shared weights	-	-	91.54	-
Roy et al. [41]	VGG-9	StochSigmoid XNOR-Net	-	-	87.95	55.54
Xing et al. [42]	Inception-v4	Homeostasis-based conversion	-	-	92.49	70.4
Hu et al. [43]	ResNet-8	ANN-to-SNN conversion	99.59	-	-	-
Hu et al. [43]	ResNet-44	ANN-to-SNN conversion	-	-	91.98	68.56
Guerguiev et al. [44]	ConvNet + LIFNet	Regression discontinuity design	-	91.81	76.2	-
Thiele et al. [45]		Direct spike gradient	99.52	-	89.99	-
Wu et al. [46]	8-layer CNN	Error BP through time	-	-	90.53	-
Severa et al. [7]	VGG-like	Whetstone (Sharpened ANN)	99.53	-	84.67	-
Severa et al. [7]	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**

[3] J. Bergstra, D. Yamins, and D. D. Cox, "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms," in *Proceedings of the 12th Python in science conference*. Citeseer, 2013, pp. 13–20.

[4] J. M. Hernández-Lobato, M. A. Gelbart, R. P. Adams, M. W. Hoffman, and Z. Ghahramani, "A general framework for constrained bayesian optimization using information-based search," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 5549–5601, 2016.

[5] 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)*, 2019, pp. 1–8.

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

[7] 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, p. 86, 2019.

[8] 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.

[9] ——, "Bayesian multi-objective hyperparameter optimization for accurate, fast, and efficient neural network accelerator design," 2020, p. submitted.

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

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

[12] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in neural information processing systems*, 2011, pp. 2546–2554.

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

[14] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951–2959.

[15] Y. Zhang, K. Sohn, R. Villegas, G. Pan, and H. Lee, "Improving object detection with deep convolutional networks via bayesian optimization and structured prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 249–258.

[16] 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.

[17] 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.

[18] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy *et al.*, "Evolving deep neural networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 2019, pp. 293–312.

[19] 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*. ACM, 2017, p. 7.

[20] M. J. Shafiee, A. Mishra, and A. Wong, "Deep learning with darwin: Evolutionary synthesis of deep neural networks," *Neural Processing Letters*, vol. 48, no. 1, pp. 603–613, 2018.

[21] J. Liang, E. Meyerson, and R. Miikkulainen, "Evolutionary architecture search for deep multitask networks," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2018, pp. 466–473.

[22] 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.

[23] 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.

[24] 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.

[25] J. Kim and D.-S. Kim, "Competitive hyperparameter balancing on spiking neural network for a fast, accurate and energy-efficient inference,"

TABLE VI
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%**

in *International Symposium on Neural Networks*. Springer, 2018, pp. 44–53.

[26] 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.

[27] *Scikit-Optimize python package*, accessed January 6, 2020. [Online]. Available: https://scikit-optimize.github.io/

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

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

[30] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

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

[32] 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.

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

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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."

[41] 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.

[42] 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.

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

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

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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.