A Quantitative Exploration of Collaborative Pruning and Approximation Towards Energy-Efficient Deep Neural Networks

Xin He, *Member, IEEE,* Wenyan Lu, Ke Liu, Guihai Yan, *Member, IEEE,* and Xuan Zhang, *Member, IEEE*

***Abstract*—To reduce the excess energy footprint of deep neural networks (DNNs), diverse solutions have been proposed across the system stacks. For instance, algorithm-level network pruning techniques remove insignificant connections to simplify the NN models, whereas architecture-level techniques leverage approximate units to perform computations in a cost-efficient manner. However, the collaborations across different optimization layers are rarely explored. We observe that with careful characterization, cross-stack techniques can be leveraged to further reduce the energy consumption of DNNs with minimal accuracy impact compared with employing a single technique. Specifically, in this paper, we bridge the gap between network pruning and approximate multiplication by proposing a collaborative solution to optimize computational power. We first characterize the error resilience and energy consumption of DNNs in each layer and apply layer-wise network pruning. Based on the pruned model, we propose an incremental approximation and retraining scheme to apply approximate multiplication. The experimental results from three datasets demonstrate that the proposed cross-stack approach achieves 26.23% and 29.97% energy reduction compared with the approximation-only and pruning-only method with only 1.39% accuracy degradation compared to the optimal accuracy of the original networks on average.**

***Index Terms*—Neural network, Energy efficient computing, Network pruning, Approximate computing.**

I. Introduction

DEEP neural networks (DNNs) are biologically inspired machine learning models that have been successfully demonstrated to deliver superior performance in a wide range of tasks, including image classification, object detection, machine translation, and so on. The success of DNNs can be attributed to innovations across the computing system stacks: to achieve higher accuracy, large-scale networks are created along with more advanced training algorithms and an ever-increasing volume of sample data. To speed up NN training deployment, powerful parallel computing engines (e.g., GPUs) are designed to accelerate computationally-intensive mathematical operations. Despite the fast pace of performance improvement, when it comes to edge devices with stringent power budgets, deploying advanced DNNs remains a challenge because of their excess computational energy and memory footprint[1].

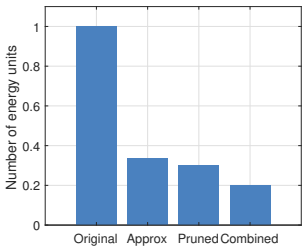
To deal with the high energy consumption, a growing body of techniques has been explored. At the algorithm level, DNNs are simplified by proposing concise network models (e.g. ResNet for ImageNet) or pruning insignificant connections of a pre-trained models [2]. At the hardware front, efficiency-driven designs have been conducted at the architecture-, circuit-, and device-levels: customized accelerators aim at higher energy efficiency [3], [4]; approximate circuits trade accuracy for energy efficiency [5], [6]; and emerging technologies (e.g. resistive memory crossbar) provide in-memory computing substrates for NN operations [7].

Despite the advance achieved at the individual optimization levels, these isolated techniques may lead to suboptimal energy saving from a whole system perspective, which calls for a synergistic and collaborative approach across the different system stacks. For example, an aggressively pruned DNNs may be able to further take advantage of the less accurate but more energy efficient arithmetic operators to carry out tasks with minimal accuracy loss. Fig. 1a shows the computational energy of a typical CNN model for *CIFAR10* dataset and its energy-optimized counterparts are estimated and compared. The computational energy is estimated based on the amount of multiply-accumulation (MAC) operations and its energy consumption dumped from Synopsys EDA tools. And the analytical model is detailed in Section III. Without loss of generality, we adopt an existing approximate multiplier, and its configuration is detailed in Section IV. All computational energies are normalized by the energy consumption from the *Original* CNN. The *Approx* case indicates the energy-intensive multiplication operations in the networks are performed by aggressive approximate multipliers (detailed in Section IV), the *Pruned* represents the case when the unimportant/unnecessary weights are pruned (i.e. around 70% of weights on average in both convolutional and fully connected layers), and the *Combined* represents a simple joint optimization of approximate multiplication and network pruning where 50% of weights are pruned and the rest are used with approximate multiplications in forward propagation. As illustrated in the figure, although stand-alone techniques could save computational energy by 66.7% and 70.1% compared to the original network, their synergistic combination could further improve the energy savings to 80.0%. This collaborative approach, however, has to be conducted with care, for naive implementations could easily

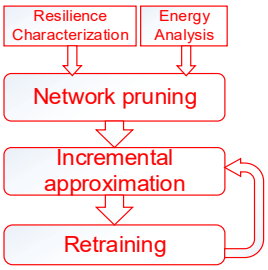
X. He is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, 48109 USA e-mail: ({xinhe}@umich.edu). This work was done when X. He was a postdoctoral research associate at Washington University in St. Louis.

L. Ke and X. Zhang are with the Department of Electrical and Systems Engineering, Washington University in St. Louis, St. Louis, MO, 63130 USA e-mail: ({xin.he, xuan.zhang}@wustl.edu).

W. Lu and G. Yan are with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, 100190, and the University of Chinese Academy of Sciences. e-mail: ({luwenyan, yan})@ict.ac.cn



(a) Comparisons of computational energy consumption between the original network and optimized networks by approximate computing, network pruning, and the combined scheme



(b) Framework of proposed cross-stack approach: 1) Network characterization to determine pruning factor, 2) Network pruning, 3) Incremental approximation and retraining

Fig. 1: Motivation and framework of proposed cross-stack approach

degrade the energy efficiency instead of improving it. For example, aggressively pruning 70% of total weights and naively applying approximation for the remaining weights fails to meet the accuracy requirement. Therefore, the energy-optimized *Combined* case prunes less weights (50%) and leaves some headroom for approximation. In this case, while the amount of pruned weights are reduced, the complementary effect of approximate multiplication and network pruning outperforms using any stand-alone technique. Note that in all three energy-optimized cases, the maximal accuracy degradation is limited to be less than 1%.

The above example motivates the exploration of cross-stack approach that has the potential to offer superior energy efficiency. However, finding an optimal solution is nontrivial, as a naive implementation could degrade the energy consumption. We first need to carefully characterize accuracy-energy tradeoffs in DNNs. This is because the overall proportion of pruning and approximation determines the inference accuracy and energy consumption. For example, with aggressive pruning, the network may fail to meet a certain accuracy requirement with approximate multiplication, whereas with conservative pruning, the energy consumption may not be optimal even after applying approximation. What complicates the exploration further is the sensitivity of individual weight to pruning and approximation is hard to characterize, because weights in different layers have different contributions to the energy footprint. Another problem is naively applying approximation and retraining leads to degraded accuracy because the noise from approximation reduces the effectiveness of parameter updating during training.

In this paper, we investigate a collaborative cross-stack approach leveraging both network pruning and approximate computing. We perform pruning sensitivity and energy consumption analysis in a layer-wise manner to guide network pruning. With the knowledge of layer resilience and power consumption, three different pruning modes (conservative, intermediate and aggressive) are derived for later collaboration with approximate computing. To complement network pruning and leverage approximate computing with minimal accuracy impact, we then develop an incremental network approximation and retraining scheme that can work on top of the pruned model. Specifically, the approximation multiplications for network weights are incrementally applied, so the network weights that are not currently involved in approximate computing can still be precisely updated to compensate for possible accuracy loss. We are able to obtain a network tailored for approximate computing with high accuracy after the proposed incremental approximation and retraining steps. The entirety of our proposed framework is illustrated in Fig. 1b.

II. Background

A. Neural network pruning

Despite their superior performance, DNNs require a large number of parameters to carry out a task. For example, VGG-16 uses 138MB parameters to perform the ImageNet task. With the large number of parameters, parameter redundancy has been identified as commonly existed in DNNs [8], [9]. The redundancy allows the network pruning and quantization techniques to simplify the DNNs with minimum accuracy impact. Early approaches like Optimal Brain Damage [10] prune the network based on the Hessian of the loss function to network weights. However, the complexity of the Hessian matrix requires high computation efforts. Recently, Han et al. propose a magnitude-based pruning and retraining scheme to preserve the original accuracy[2]. Inspired by Han's work, Dynamic Network Surgery (DNS) [11] allows a portion of network connections to be pruned or spliced dynamically during the retraining process, which exhibits superior accuracy to prior works. In this work, we adopt DNS to perform network pruning.

B. Approximate computing

Approximate computing is a promising technique for energy optimization. Due to the intrinsic noise tolerance of DNNs, approximate multiplications and fuzzy storage have been explored to improve their energy efficiency. In this paper, we focus on the computational energy optimization since it contributes to a significant portion of energy in NN hardware. Without loss of generality, to assess the impact of approximate multiplication, we adopt a recently-proposed unbiased approximate multiplier for weight-activation multiplication [12]. This design explores the tradeoff between precision and computing efficiency which is similar to changing the effective widths k_0 and k_1 for two operands, i.e., with a smaller k configuration, the approximate multiplier gains higher energy efficiency at the cost of increased noise. Mathematically, the approximation is performed as follows: firstly, the positions of the leading non-zero bit and the radix point of the two operands are detected. Then the following k_0 and k_1 bits are extracted and then updated: from MSBs to LSBs the first $k_0 - 1$ and $k_1 - 1$ bits are kept intact while the k_0th and k_1th bits are set to one. Then the k_0 and k_1 bits are multiplied and shifted with the summed radixes to obtain the approximated result.

III. Framework of the Cross-Stack Solution

A. Neural network resilience and energy characterization

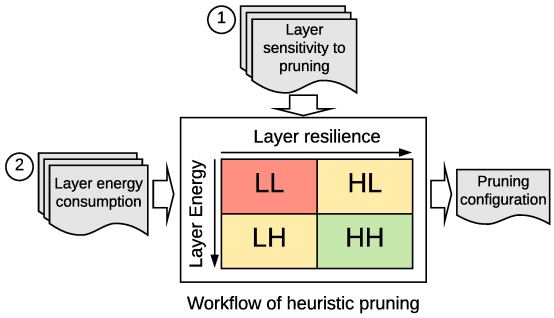
In our cross-stack framework, network pruning can be viewed as an extreme energy-efficient but error-prone version of approximate computing. We explore performing the network pruning step before the approximation step. As we briefly discussed in the introduction, the knowledge of both resilience of weights and the energy consumption of multiplications and accumulations with the weights are needed to determine the optimal network pruning configuration. The decision-making overview is illustrated in Fig. 2a. First, layer wise sensitivity analysis is performed to study the general knowledge of error-tolerance capability of DNNs. Second, computation energy estimation is used to quantify the potential energy saving from network pruning. With the observation and energy estimation, we categorize the layers of a DNN into four categories: high resilience high energy (HH), high resilience low energy (HL), low resilience high energy (LH) and low resilience low energy (LL). We select the HH layers for higher pruning rate while selecting HL and LH layers for lower pruning rate because it guarantees maximum energy saving at the cost of little accuracy impact. Note that this simple decision making heuristic is conducted offline.

For resilience analysis, existing techniques fail to faithfully and fully characterize the resilience for DNNs. Representative works like [5], [6] leverage neuron-level criticality analysis by measuring the first-order/second-order derivatives of output loss to neuron activations through backpropagation. However, this measure of sensitivity only works for small weight perturbations and fails to characterize larger deviations from pruning and approximation. Aiming to reduce the number of weights, Han et al. [2] adopts a light-weight method by ranking all the weights of a pretrained model according to their magnitudes and prunes the weights with small magnitude. This global pruning method results in minimal network parameters instead of energy consumption.

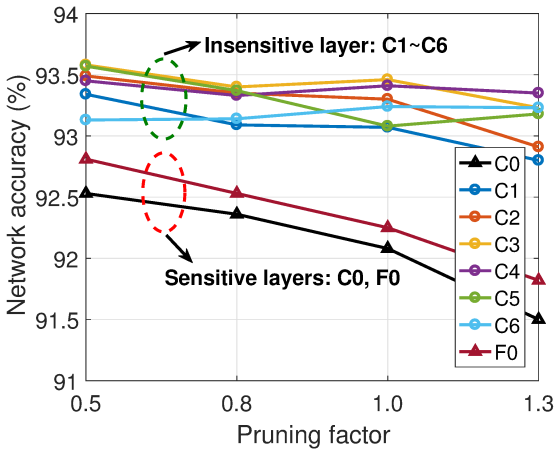
We perform resilience characterization layer-wise since it requires less efforts to explore the design space than weight-level or neuron-level methods, and the insights from high-level characterization are more stable and applicable to different tasks. Specifically, we systemically evaluate the layer-wise resilience by applying different pruning rates to a layer and compare the output accuracy. The pruning threshold for a layer is correlated to $Mean(|W|) + \eta \times Std(|W|)$ [11]. Here, W is the set of weights in a layer, and absolute values for the weights are used for mean and standard deviation calculation. By changing the pruning factor η, different pruning rates can be applied. In this exploration, the pruning factors η are selected from $0.5, 0.8, 1.0$ and 1.3, which correspond to $68 \sim 87\%$ pruning rate for a layer. And we only prune a single layer at a time for characterization purposes. Note that after weight pruning, the NNs are retrained using a dynamic pruning method [11] for enough epochs (e.g., 5) to fine-tune the remaining weights and recover from pruning.

Fig. 2b shows the classification accuracy under different pruning factors across different layers of a CNN for *CIFAR10*. At a similar pruning factor, later deeper layers (except the fully connected layers) always exhibit higher resilience to pruning compared to early shallower layers, as the network accuracy is more stable at different pruning factors for $C1 \sim C6$ layers and the accuracy degrades significantly with increasing pruning factors for $C0, F0$ layers. We speculate that the reasons are two folds. Firstly, early convolutional layers extract fundamental features which are used by later layers to form high-level features. Discarding these fundamental features severely degrades network functionality. Secondly, the early convolutional (CONV) layer ($C0$) and final fully connected (FC) layer ($F0$) have a smaller amount of weights which carry more information per weight.

In order to select the best candidates for pruning, we expect maximum energy saving at the cost of little accuracy impact. Hence besides the resilience characterization, we estimate the energy consumption of layers so that resilient layers with high energy consumption could be identified and prioritized for high pruning rate. To simplify the estimation, we model the energy consumption of a layer in an hardware-implementation-independent manner. We treat the energy consumption E_{MAC} of a single multiply-accumulation (MAC) operation as the ba-



(a) Layer-wise sensitivity analysis and layer computational energy estimation are first performed. And high energy and high resilience layers are selected



(b) Layer-wise sensitivity analysis of accuracy to pruning rate on CIFAR10: For each layer, different pruning factors are applied while other layers stay intact

Fig. 2: Upper: Overview of proposed heuristic decision making process, Lower: layer sensitivity analysis of a representative DNN

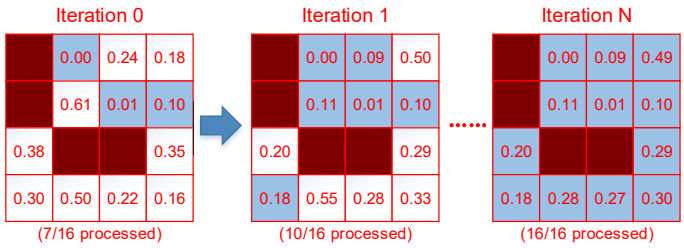


Fig. 3: Incremental approximation scheme: in each iteration, a fixed percentage of weights of low approximation error is picked for approximate computing while the rests are fine-tuned for error recovery. Grey squares indicate pruned weights, Blue squares mean weights used with approximate computing while the white squares are the weights for fine-tuning

sic energy unit. Thus the energy consumption of a CONV layer and an FC layer are $N_o \times N_i \times K_W \times K_H \times W_o \times H_o \times E_{MAC}$ and $N_o \times N_i \times E_{MAC}$, respectively. In the CONV layer, N_o and N_i are the numbers of output feature maps (OFMs) and input feature maps, K_W and K_H are the width and height of kernels, while W_o and H_o are width and height of OFMs. In the FC layer, N_o and N_i are the numbers of output and input neurons. For *CIFAR10*, the normalized layer energy consumptions for seven CONV layers and one FC layer are $[\mathbf{0.023}, 1.000, 0.500, 1.000, 0.500, 1.000, 0.125, \mathbf{0.000}]$. We find the energy consumption of the first CONV layer $C0$ and last FC layer $F0$ contribute little (i.e. less to 0.6%) to total energy. This is because N_i is small for the $C0$ layer and computations used by $FC0$ layer only involve inner-product operations. The low energy consumption nature of $C0$ and $FC0$ makes them less receptive to pruning. Besides, sensitivity analysis above also reveals that it is undesirable to prune the $C0$ and $F0$ layers from an accuracy perspective. These insights can also be applied for other DNNs because of their structural similarity.

Hence our pruning strategy is to prune the middle layers of large energy footprint while leaving the first and last layers intact. For a thorough exploration, we adopt three different pruning configurations: conservative, intermediate and aggressive pruning modes. In the conservative pruning, the scaling factor vector for the middle layers is $0.5, 0.5, 0.5, 0.5, 0.5, 0.5$ in which the value 0.5 is selected to leave accuracy headrooms for approximate computing. And in the aggressive pruning, the factor vector is $0.8, 0.8, 0.8, 0.8, 0.8, 0.8$, which is obtained by selecting the maximum scaling factors within allowed accuracy degradation. In the intermediate pruning, the scaling factors of the early layers (i.e. two layers in this paper) in the middle layers are set to 0.5. These three pruned models are then fed to the proposed incremental approximation scheme. We applied this strategy for all datasets used in the experiment.

B. Incremental approximation and retraining scheme

In this section, we present an incremental approximate and retraining scheme on top of the pruned models to fully exploit the potential of energy saving from approximate multiplication and suppress the accuracy loss. Instead of directly incorporating the entire network weights to approximate multiplication, in the proposed scheme, the pruned models are incrementally trained under approximate multiplication, as shown in Fig. 3. In each iteration, three major steps are applied to each layer: weight partitioning, group-wise approximation, and network retraining/fine-tuning. Weight partitioning first divides the unprocessed weights into two groups in a complementary way. The weights of the first group undergo approximate multiplication during the network forward propagation, while the weights in the second group are updated normally in the retraining step to compensate for the accuracy loss from approximate computing. Once the three steps are finished in an iteration, the same process is repeatedly performed on the second group of weights until all network weights are processed by this incremental approximation scheme. Mathematically, weight partition in a layer is represented as:

$$\begin{aligned} &G^{(1)} \cup G^{(2)} \cup G^{(Pruned/processed)} = W \\ &G^{(1)} \cap G^{(2)} = \emptyset, G^{(1)} \cap G^{(Pruned/processed)} = \emptyset \\ &G^{(2)} \cap G^{(Pruned/processed)} = \emptyset \end{aligned} \tag{1}$$

In these equations, $G^{(1)}$ means the first group of weights used in approximate multiplications while $G^{(2)}$ indicates the second group used to compensate for the accuracy loss from approximation in the current iteration. The group $G^{(Pruned)}$ contains pruned (zero-valued) and processed weights in this layer. The pruned weights are fixed while more and more weights get processed along iterations of incremental approximation. Specifically, to partition the group, we rank the weights based on their absolute approximation error, and weights with small approximation error are put into $G^{(1)}$ for approximation. Note that we use approximation error of weights to guide the application of approximate multiplication because other operands (the input activations of DNNs) are input-dependent and highly sensitive. This also reflects the insights from preliminary exploration that the approximate multipliers used should not compromise the precision of the activations too much.

A novel retraining/fine-tuning algorithm for approximate multiplication is also developed that differs from conventional training algorithm. The conventional training seeks to reduce the loss function:

$$E_{network} = E(W) + \gamma R(W) \tag{2}$$

where $E(W)$ is the network output loss, $R(W)$ is the regularization loss and γ is a control coefficient for regularization. In our retraining algorithm for approximate multiplication, the loss function is transformed to:

$$E_{network} = E(\boldsymbol{amul}(W)) + \gamma R(W) \tag{3}$$

The difference is that the functionality of approximation multipliers (**amul**) are incorporated in training. So the two input operands (weights and input activations) to approximate multiplications are changed accordingly. In this way, the problem of reducing network loss can be handled by the stochastic gradient descent (SGD) optimization simultaneously. In the incremental retraining scheme, only weights from the compensation group $G^{(2)}$ can be updated while the gradients for $G^{(1)}$ and $G^{(Pruned/processed)}$ are masked with zero to

Algorithm 1 Incremental network retraining algorithm for approximate computing

Input: the training data $Samples$, the pruned DNN model W, number of iterations N_{iter}

Output: the pruned model with optimized for approximate computing

1: Initializations: $G^{(2)} = W^{(Remaining)}$, $G^{(1)} = \emptyset$, $G^{(Pruned/processed)} = W^{(Pruned)}$, current iteration $i = 1$
2: **for** current iteration $i \leq N_{iter}$ **do**
3: Rank the weights in $G^{(2)}$ with the approximation error
4: $\eta = 100 * 1/(N_{iter} - i + 1)$ percent weights in $G^{(2)}$ need to be approximated in this iteration
5: Remove η percent weights of smallest approximation error from $G^{(2)}$ to form $G^{(1)}$ for approximate multiplication.
6: Perform forward propagation to obtain network loss, then conduct back-propagation and SGD to update weights in $G^{(2)}$
7: $G^{(Pruned/processed)} = G^{(Pruned/processed)} \cup G^{(1)}$
8: $i+ = 1$

prevent updating. The detailed algorithm for the proposed incremental approximation is listed in Algorithm 1. In this paper, we use four iterations in the incremental approximation and retraining scheme. Empirically, four iterations are enough to get a satisfactory accuracy. With more iterations, the return on the accuracy improvement is diminishing and small.

IV. EXPERIMENTAL METHODOLOGY

NN accelerator architecture. To evaluate the energy efficiency improvement from approximate computing, we implement and customize a data-driven NN accelerator architecture named "FlexFlow" proposed in earlier work for approximate computing [3]. FlexFlow stores weights and activations in two separate buffers and employs parallel processing engines (PE) for computation. Each PE consists of a DRUM approximate multiplier [12], an adder, a neuron local memory, a weight local memory, and a controller. The approximate multiplier in proposed approach and approximation-only approach, we set $K = 1$ for weights and $K = 5$ for activations (denoted as $k(1, 5)$) since the input-dependent nature of activations makes them hard to characterize. For the pruning-only approach, the multiplier is set to high accuracy mode, $k(1, 5)$. Note that FlewFlow is mainly used as a faithful tool to evaluate the energy saving of the proposed approach, and we do not claim the high energy efficiency of FlexFlow itself since it is not customized for pruned networks.

Energy evaluation flow. To evaluate the energy improvement from approximate multiplication, we implement the approximate accelerator using Verilog and then synthesize the design using the Synopsys Design Compiler with the TSMC 65nm library. The energy results are gathered using Synopsys PrimeTime.

Training tool and Dataset. We implement the layer-wise pruning and incremental approximation algorithm and inference simulator with PyTorch deep learning suite. Representative datasets *CIFAR10*, *CIFAR100*, *STL-10* and *ImageNet12* are used in the experiment and detailed in Table I. For *ImageNet12*, its archived training set and test set are used while for other datasets, 80% of samples are used for training and the remaining 20% samples are for testing.

V. EXPERIMENTAL RESULTS

To evaluate the accuracy improvement, we first compare the accuracy between the proposed cross-stack approach and the state-of-the-art baseline which combining pruning and approximation. For a fair comparison, dynamic network surgery are used for pruning while same approximate computing configuration and same pruning rate is applied across the two approaches. The cross-stack approach performs the dynamic network surgery step for pruning and the incremental approximation step consecutively while the baseline leverages dynamic network surgery, approximate multiplication, and fine-tuning simultaneously. With the knowledge of network resilience and energy consumption, in this experiment, three carefully-selected pruning modes, conservative (CON), intermediate (INTER), and aggressive (AGG) modes which are detailed in Section III are leveraged for exploration and evaluation. The accuracy results for the three datasets are shown in Fig. 4. In general, the proposed cross-stack approach achieves on average 0.8730% higher accuracy over the baseline for three datasets in all pruning modes. Even though 0.8730% higher accuracy does not seem a large improvement at first glance, this improvement is sufficiently large to bridge the gaps from the pruned and approximated models to the ideal models. The average accuracy loss to the ideal case is reduced from 2.0120% with the baseline method to 1.139% by using the proposed approach. Besides, the cross-stack approach shows better resilience to pruning. Compared with the baseline, the accuracy improves by 0.3234%, 0.7709% and 1.5246% in the conservative, intermediate and aggressive modes, correspondingly. Note that in this experiment, the resulting pruning rate and approximation configurations are the same, so are the energy consumption of the two approaches.

Secondly, we evaluate the computational energy consumption of approximation-only, pruning-only, and the proposed collaborative cross-stack approach generated DNNs on FlexFlow accelerator. In the approximation-only case, the computations are carried out by the aggressive approximate multiplier working at $k(1, 5)$ mode, similar to the proposed approach. In the pruning-only case, the DNNs are retrained by DNS approach with 80% weights pruned on average while the approximate multiplier working a high accuracy $k(5, 5)$ mode. In the proposed case, the cross-stack approach applies the incremental approximation technique on the aggressively pruned (AGG) model, which shows only 1.39% accuracy degradation to the ideal networks on average. The results of energy consumption for three datasets are shown in Fig. 5. The results show that the proposed approach effectively reduces the energy consumption by 26.23% and 29.97% compared with the approximation-only and pruning-only method, correspondingly. To summarize, the proposed cross-stack approach makes it possible to further reduce the energy consumption of DNNs with minimal (e.g., 1.39%) accuracy loss. Note that *CIFAR10*

TABLE I: Datasets and the corresponding topologies of DNNs

Datasets	Topology of DNNs (in terms of feature maps)	Ideal accu	Source
CIFAR10	Regular network: Inputs: $3 \times 32 \times 32$, $128 \times 32 \times 32$, $128 \times 32 \times 32$, $256 \times 16 \times 16$, $256 \times 16 \times 16$, $512 \times 8 \times 8$, $512 \times 8 \times 8$, $1024 \times 2 \times 2$, Outputs: 10	93.57%	UToronto
	Shallow network: Inputs: $3 \times 32 \times 32$, $256 \times 32 \times 32$, $256 \times 32 \times 32$, $512 \times 16 \times 16$, $1024 \times 8 \times 8$, $2048 \times 2 \times 2$, Outputs: 10	87.74%	UToronto
CIFAR100	Regular network: Inputs: $3 \times 32 \times 32$, $128 \times 32 \times 32$, $128 \times 32 \times 32$, $256 \times 16 \times 16$, $256 \times 16 \times 16$, $512 \times 8 \times 8$, $512 \times 8 \times 8$, $1024 \times 2 \times 2$, Outputs: 100	73.92%	UToronto
	Shallow network: Inputs: $3 \times 32 \times 32$, $256 \times 32 \times 32$, $256 \times 32 \times 32$, $512 \times 16 \times 16$, $1024 \times 8 \times 8$, $2048 \times 2 \times 2$, Outputs: 100	66.71%	UToronto
STL-10	Regular network: Inputs: $3 \times 96 \times 96$, $32 \times 96 \times 96$, $64 \times 48 \times 48$, $128 \times 24 \times 24$, $128 \times 12 \times 12$, $256 \times 4 \times 4$, $256 \times 2 \times 2$, Outputs: 10	76.10%	Stanford
	Shallow network: Inputs:$3 \times 96 \times 96$, $64 \times 96 \times 96$, $128 \times 48 \times 48$, $256 \times 24 \times 24$, $512 \times 12 \times 12$, $512 \times 8 \times 8$, Outputs: 10	71.26%	Stanford
ImageNet12	Regular network: Inputs: $3 \times 224 \times 224$, $64 \times 55 \times 55$, $192 \times 27 \times 27$, $384 \times 13 \times 13$, $256 \times 13 \times 13$, $256 \times 6 \times 6$, $4096 \times 1 \times 1$, $4096 \times 1 \times 1$, Outputs: 1000	77.42%	Stanford

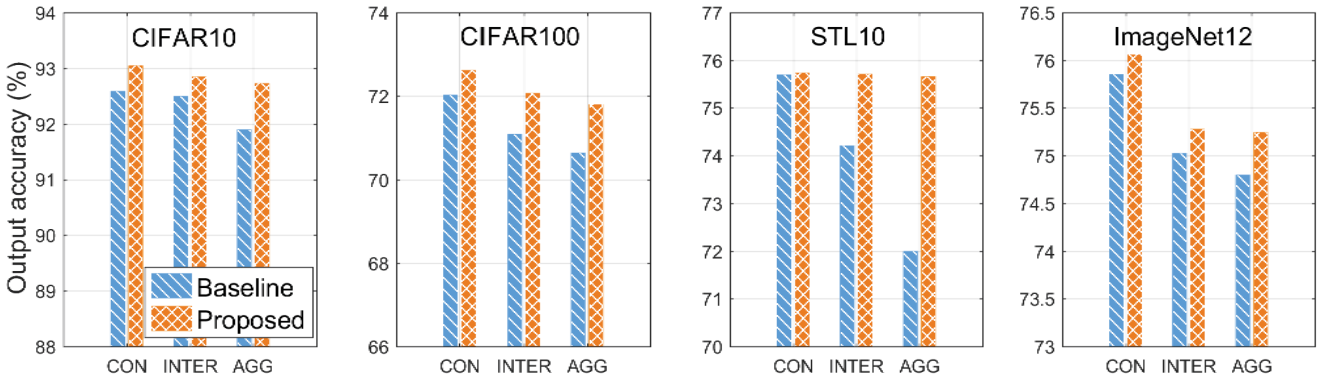


Fig. 4: Network output accuracy using proposed cross-stack approach and the baseline (i.e., DNS) at three different pruning modes for three representative datasets

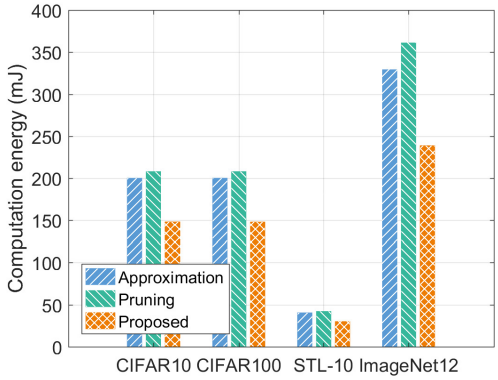


Fig. 5: Energy consumption of DNNs under approximation-only, pruning-only and proposed cross-stack approach

and *CIFAR100* have similar energy profiles since the network topologies used are very similar and the only difference lies in the last FC layer and the number of outputs.

Furthermore, in order to demonstrate the effectiveness of the "collaboration" between pruning and approximation, we compare *the aggressive pruning-only scheme*, *the iterative incremental approximation-only scheme* and *the proposed cross-stack* in terms of output accuracy and energy consumption under more diverse network models. Specifically, *the aggressive pruning-only scheme* leverages high pruning factor (i.e., 1.3 which indicates a pruning rate between 80% and 87%) for intermediate layers following the observation in Section III, which will fully exploit the energy saving under the aggressive pruning factor. And *the iterative incremental approximation scheme* adopts proposed incremental pruning which would take advantages of the error-tolerance capability of DNNs. The results are reported in Table II. As shown in Table II, the proposed cross-stack approach outperforms *the aggressive pruning scheme* and *the incremental approximation-only scheme* by 26.59% and 24.64% in terms of energy consumption on average, respectively. Besides the reduced energy consumption, the proposed cross-stack solution exhibits both 1.30% higher accuracy with *the aggressive pruning scheme*, and 0.91% higher accuracy with the *the iterative incremental approximation scheme*.

VI. Conclusion and Future Work

To address the high energy consumption in modern deep neural networks, diverse techniques from different system stacks have been proposed. In this work, we aim to bridge the gap across the stacks. Specifically, we propose a cross-stack approach where network pruning and approximate computing collaborate. To guide network pruning, we explore the pruning sensitivity and energy consumption at the layer level. Through this exploration, a general insight is gained: the first convolution layer and fully connected layers are less robust and consume less energy compared with other layers, which makes them less appealing to pruning. Meanwhile, three different pruning modes are derived to explore the collaboration with

TABLE II: Computational energy and accuracy comparison between *aggressive pruning-only scheme*, *incremental approximation-only scheme* and *the proposed cross-stack approach* with more diverse network models

Network models	Scheme	Accuracy(%)	Energy(mJ)
Cifar10 (Regular)	Pruning	91.77	208.2380
	Incremental	92.59	200.3728
	Proposed	92.54	142.7912
Cifar10 (Shallow)	Pruning	86.80	429.5200
	Incremental	86.57	413.0913
	Proposed	87.36	354.7709
Cifar100 (Regular)	Pruning	69.18	208.2845
	Incremental	71.25	200.3967
	Proposed	71.64	142.6540
Cifar100 (Shallow)	Pruning	65.16	429.6400
	Incremental	65.91	413.1149
	Proposed	66.10	347.1929
STL-10 (Regular)	Pruning	73.72	42.3805
	Incremental	74.39	40.6913
	Proposed	74.29	29.2383
STL-10 (Shallow)	Pruning	69.55	205.4496
	Incremental	71.00	227.3043
	Proposed	71.15	160.7291
ImageNet12 (AlexNet)	Pruning	73.84	361.3896
	Incremental	74.01	329.4823
	Proposed	75.31	239.0736

approximate computing. With the pruned model, we propose an incremental approximation scheme which incrementally incorporate approximate computing to a changing group of network weights and retraining the remaining weights to compensate for the accuracy loss. The experimental results across three representative datasets show the proposed cross-stack approach makes it possible to further reduce the energy consumption of DNNs with minimal accuracy loss. Besides, the proposed approach exhibits better resilience (higher accuracy than the baseline optimization method) when incorporating approximate computing in the aggressively pruned DNNs.

Our work represents an initial step in the direction of cross-stack frameworks. Future directions for improved DNN energy efficiency that extend this work may include: 1) with more flexible approximate hardware (e.g. accuracy-configurable approximate multiplier), approximate computing can be conducted in a dynamically reconfigurable manner during runtime, which could be leveraged to adaptively tradeoff between energy and accuracy; 2) since the memory consumption can also be reduced by fuzzy memorization, building the synergy between approximate multiplication and fuzzy storage could further reduce the energy consumption of deep neural networks.

ACKNOWLEDGMENT

This work is partially supported by NSF award CNS-1739643 and the National Natural Science Foundation of China under Grant Nos. 61872336, 61572470, and in part by Youth Innovation Promotion Association, CAS under grant No.Y404441000.

REFERENCES

[1] Z. Xu, Z. Qin, F. Yu, C. Liu, and X. Chen, "Direct: Resource-aware dynamic model reconfiguration for convolutional neural network in mobile systems," in *Proceedings of the International Symposium on Low Power Electronics and Design*. ACM, 2018, p. 37.

[2] 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.

[3] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 553–564.

[4] S. Pal, J. Beaumont, D.-H. Park, A. Amarnath, S. Feng, C. Chakrabarti, H.-S. Kim, D. Blaauw, T. Mudge, and R. Dreslinski, "Outerspace: An outer product based sparse matrix multiplication accelerator," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 724–736.

[5] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, "Approxann: an approximate computing framework for artificial neural network," in *Design, Automation and Test in Europe Conference Exhibition (DATE)*, 2015, pp. 701–706.

[6] X. He, L. Ke, W. Lu, G. Yan, and X. Zhang, "Axtrain: Hardware-oriented neural network training for approximate inference," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2018.

[7] L. X. et al, "Switched by input: Power efficient structure for rram-based convolutional neural network," in *Design Automation Conference (DAC)*, 2016, p. 125.

[8] M. Denil, B. Shakibi, L. Dinh, N. De Freitas *et al.*, "Predicting parameters in deep learning," in *Advances in neural information processing systems*, 2013, pp. 2148–2156.

[9] L. Liu, L. Deng, X. Hu, M. Zhu, G. Li, Y. Ding, and Y. Xie, "Dynamic sparse graph for efficient deep learning," *International Conference on Learning Representations(ICLR)*, 2019.

[10] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in neural information processing systems*, 1990, pp. 598–605.

[11] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Advances In Neural Information Processing Systems*, 2016, pp. 1379–1387.

[12] S. R. S Hashemi, R Bahar, "Drum: A dynamic range unbiased multiplier for approximate applications," in *International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 418–425.

Xin He (M17) is a postdoctoral research fellow in University of Michigan, Ann Arbor. He received the the PhD degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2017. His research interests include application specific acceleration, deep learning, neural network accelerator, and approximate computing. He is a member of the IEEE.

Wenyan Lu is an assistant professor at the State Key Lab. of Computer Architecture, Institute of Computing Technology (ICT), Chinese Academy of Science (CAS). He received his Ph.D. degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2019. His research interests include computer architecture and domain-specific accelerator.

Liu Ke is a Ph.D student in the Electrical and Systems Engineering department at Washington University in St. Louis. Her current research interest lies in design automation and hierarchical modeling of custom machine learning and artificial intelligence accelerators.

Guihai Yan (M10) is a Professor with the Institute of Computing Technology, Chinese Academy of Sciences. He received the Ph.D. degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences, Beijing, China, in 2011. His research interests include computer architecture, domain-specific computing, and intelligent systems. He is IEEE/ACM/CCF member.

Dr. Xuan Zhang (S08, M15) is an Assistant Professor at Washington University in St. Louis. She received her Ph.D. degree in Electrical and Computer Engineering from Cornell University. Her research interest include hardware/software co-optimization, design automation, and hardware acceleration. She is a member of IEEE and ACM.