Mixed Precision Neural Architecture Search for Energy Efficient Deep Learning

Chengyue Gong*[1], Zixuan Jiang*[2], Dilin Wang[1], Yibo Lin[2], Qiang Liu[1], and David Z. Pan[2]

[1]CS Department, The University of Texas at Austin
[2]ECE Department, The University of Texas at Austin
chengyuegong@gmail.com; {zixuan, yibolin}@utexas.edu; {dilin, lqiang}@cs.utexas.edu; dpan@ece.utexas.edu

***Abstract*—Large scale deep neural networks (DNNs) have achieved remarkable successes in various artificial intelligence applications. However, high computational complexity and energy costs of DNNs impede their deployment on edge devices with a limited energy budget. Two major approaches have been investigated for learning compact and energy-efficient DNNs. Neural architecture search (NAS) enables the design automation of neural network structures to achieve both high accuracy and energy efficiency. The other one, model quantization, leverages low-precision representation and arithmetic to trade off efficiency against accuracy. Although NAS and quantization are both critical components of the DNN design closure, limited research considered them collaboratively.**

In this paper, we propose a new methodology to perform end-to-end joint optimization over the neural architecture and quantization space. Our approach searches for the optimal combinations of architectures and precisions (bit-widths) to directly optimize both the prediction accuracy and hardware energy consumption. Our framework improves and automatizes the flow across neural architecture design and hardware deployment. Experimental results demonstrate that our proposed approach achieves better energy efficiency than advanced quantization approaches and efficiency-aware NAS methods on CIFAR-100 and ImageNet. We study different search and quantization policies, and offer insights for both neural architecture and hardware designs.

I. Introduction

Deep neural networks (DNNs) have revolutionized many areas of machine intelligence and enabled superhuman accuracy for many different tasks. While most of the inferences currently reside in the cloud, it is increasingly desirable to deploy the trained DNNs to edge devices, such as mobile phones and wearable devices, due to privacy, security, and latency concerns or limitations in communication bandwidth. The increasing gaps between complicated DNNs and hardware implementations deteriorate the edge inference [1]. Hence, growing demand in small-size and energy efficient DNNs is motivated to meet the limited area and energy budget of edge applications.

Model quantization. Research has shown that there are redundancies in both the number of weights and the number of bit representations (or precisions) of weights and Arithmetic in DNNs [2]. Neural networks can be compressed using weight clustering and quantization to reduce the computation complexity with negligible loss of accuracy. In the quantization approach, full precision floating point representation is replaced by lower fixed point precision or even binary representation to achieve a significant reduction in computation as well as energy consumption. Learning layer-wise bitwidths has been proposed recently [3; 4] to explore the search space of layer-wise quantization policies.

Neural Architecture Search. Another major approach to learning energy efficient deep networks is by designing efficiency-friendly neural architectures [5; 6; 7; 8]. Recently, researches show that automatically designing neural architectures by efficiency-aware NAS [5; 9; 10] can bring more benefits than hand-crafted design. Following NAS framework, compact and efficient architectures are found by either searching from scratch [5; 9], or pruning or distilling from well trained large neural networks [2; 11; 12].

However, the existing works are focusing on either of the two methods aforementioned while neglecting the potential of their interaction and joint optimization for further improvement. Intuitively, the optimal choices of bit-widths and architectures are correlated. For instance, in MobileNets [13], one should retain more bits for the bottleneck layers (the expansion and projection layer) which encode the model's intermediate inputs and outputs using small 1×1 convolutional filters; on the other hand, one may consider using fewer bits in the depthwise convolution layers because empirically, they are often over-parameterized, memory bounded and are less sensitive than bottleneck layers. The combination of NAS and quantization contributes to the final accuracy and energy efficiency of neural architectures in a complex and interleaved manner. Therefore, a synergistic co-optimization of NAS and quantization is likely to allow us to learn more hardware efficient networks.

In this paper, we propose to learn more hardware-efficient deep networks by co-optimizing both precision and NAS. Compared with the hand-crafted heuristic design which often falls in sub-optimal results, our method can achieve better solutions. Our algorithm leverages a differentiable neural architecture search method with Gumbel-Softmax to directly search the optimal combinations of precision and architectures, to minimize both the accuracy and the energy cost, estimated from a physical simulator. We conduct extensive studies on two widely-adopted image classification benchmarks, CIFAR-100 [14] and ImageNet2012 [15], on which significant improvements are obtained. The contributions of this paper are highlighted as below.

- Our approach yields more energy efficient deep learning by co-optimizing the neural network architectures and quantization policies that assign different precision to different blocks of the network. To our best knowledge, this paper is the first work to explore the end-to-end co-optimization of NAS and mixed precision quantization.
- We develop a framework to effectively search the new solution space. We train a quantized DNN during the search, and adopt the REINFORCE algorithm for the non-differentiable energy oracle from hardware simulators.
- Our experimental results show that the co-optimized architectures and the bitwidths settings can achieve lower error rate and less energy consumption on CIFAR-100 and ImageNet2012 tasks than strong baseline approaches. Specifically, in ImageNet2012, we can reduce 63% energy with almost no loss of top-1 accuracy, compared with 8-bit MobileNetV2.

The rest of the paper is organized as follows. Section II introduces the related research in this area; Section III explains the detailed algorithms and discusses the proposed NAS methodology; Section IV demonstrates the results; Section V concludes the paper.

$*$ indicates equal contributions.

II. Related Work

In this section, we first introduce a general topic, hardware-aware machine learning. Then we discuss two related topics: searching hardware-friendly neural network architectures and quantization.

A. Hardware-aware Machine Learning

With the huge success of deep learning, there is an increasing demand for pushing it to the edge. However, DNNs are quickly evolving towards deeper and more complicated architectures for higher accuracy [1]. Hence, it is energy hungry and less run-time efficient on the edge inference, raising the necessity of hardware-aware machine learning design.

Besides the hardware-aware compression and quantization, researchers also pay attention to the design of efficient neural architectures for hardware-aware machine learning [16].

Some efficient neural architectures have been designed handcrafted, e.g. **MobileNet** [13], **ShuffleNet** [17]. Others are developed automatically by neural architecture search methods [5; 9].

B. Neural Architecture Search (NAS)

NAS has demonstrated superior performance on many challenging applications, such as image classification [7], object detection [9], natural language processing [6], to name a few. Most pioneer works in NAS [6; 7; 8; 18; 19] focused on searching novel architectures such that the task-oriented performance (usually accuracy) is optimized without taking hardware performance into consideration. Some of the previous works [10; 19; 20] try to find efficient network architectures, but mainly focus on the latency on GPU and CPU.

Some NAS related works have also been done on searching efficient neural networks for edge devices. **FBNet** [9] focuses on NAS with less latency on mobile devices, which relaxes the non-differentiable discrete space to differentiable continuous space using Gumbel-Softmax [21]. **ProxylessNAS** [5] also focuses on NAS for given mobile devices, where the binarized parameters are trained based on **BinaryConnect** [22]. Both works use MobileNetV2 block [13] as the base search unit, which is proved energy efficient and low-latency on edge devices.

Weight pruning can be viewed as a type of NAS, which discovers a small neural network from an over-parameterized neural network [23]. Weight pruning leverages the inherent redundancy in the number of weights, thereby achieving effective model compression with negligible accuracy loss. However, its irregular sparsity and complicated indexing scheme may induce overhead in hardware implementation and require careful optimization[2; 11; 12]. In this work, we focus on other schemes that are more friendly to hardware implementation, but our methodology can be extended to include pruning as well.

C. Quantization

Model quantization methods remove the redundancies in the neural networks and bit representations, and they are able to reduce the computational complexity significantly. The quantized models offer the potential of remarkable memory and computation efficiency, while achieving the accuracy of their full-precision counterparts [24; 25]. Moreover, weight quantization, especially equal-distance quantization, is more hardware friendly than weight pruning methods.

The critical point for quantization is to keep the balance of task-specific performance and hardware-related metrics.

Some work tries to train a binary neural network, which can extremely save energy, latency, and other hardware-related metrics at the cost of severe degradation on accuracy. **BinaryConnect** and **XNOR Net** binarize all the weights and activations [22; 26]. However, it causes an extreme loss on accuracy. For instance, applying **XNOR Net** on AlexNet [27] achieves 44.27% top-1 accuracy on ImageNet2012, which is far worse than the full-precision accuracy 56.6%. Other works focus on training low-precision neural networks, which can keep the balance between efficiency and accuracy better than binary neural networks. **Deep Compression** quantizes the network weights to reduce the model size by rule-based strategies [24]. Jacob et al. and Banner et al. train neural networks with 8-bit precision [28; 29], with straight through estimator and range batch normalization.

However, once the neural network becomes deeper, the search space of layer-wise bitwidth increases exponentially, which makes it infeasible to rely on hand-crafted strategies. The heuristic layer-wise bitwidths are believed to be sub-optimal [3], and cannot keep the balance between accuracy and efficiency. Recent works try to search layer-wise bitwidths for a pre-trained model with a particular architecture. **HAQ** [3] searches layer-wise bitwidths for MobileNet using DDPG [30], and adds a fine-tune process with few iterations after quantization. Guo et al. develops a evolutionary algorithm to search layer-wise bitwidths [10]. **Mixed Precision Quantization** [4] and **Stochastic Layer-Wise Precision** [31] search with Gumbel-Softmax [21] are not tested on some efficient neural architectures, e.g. MobileNet [13], ShuffleNet [17]. Applied on a pre-trained model, all these works can achieve better accuracy than one single bit-width and heuristic layer-wise bitwidths. These empirical results show that automatically searching the layer-wise bitwidth is helpful for deep neural network quantization.

III. Algorithm

In this section, we introduce the general form of our algorithm and present our joint NAS and mixed precision quantization framework.

A. Energy Constrained NAS

Figure 1 shows the overall flow of our model. We aim at searching novel energy efficient neural architectures. Our method can be viewed as a *controller* that interacts with a *task environment* (e.g. image classification task) and a *hardware environment* (e.g. physical energy simulator). The goal of the controller is to discover novel neural architectures that minimize the task-related loss while satisfying some energy constraints. To be concrete, our training objective can be written as follows,

$$\begin{aligned} &\min_{\theta} \mathbb{E}_{\alpha\sim\pi_\theta}\Big[\mathcal{L}(f_{w^*}(\alpha);\ \mathcal{D}^{val})\Big], \\ \text{s.t.}\quad &w^* = \arg\min_{w} \mathcal{L}(f_w(\alpha);\ \mathcal{D}^{trn}, \alpha), \\ &\mathbb{E}_{\alpha\sim\pi_\theta} J(\alpha) < c. \end{aligned} \tag{1}$$

The NAS controller π_θ generates *samples* $\{\alpha\}$ that are different configurations of neural networks. Given a network configuration α, $f_w(\alpha)$ defines a deep neural network model associated with model weights $w(\alpha)$. In the case of image classification, network f_w could be the AlexNet [27] that maps an image to a probability distribution over predicted output classes. $\mathcal{L}(\cdot)$ stands for the task-dependent loss (e.g., cross entropy for image classification) and $J(\alpha)$ measures the energy cost of the network initialized by a configuration α. The controller is trained to find the best internal parameters θ so that it will generate architectures that achieve the minimum expected task-specific loss $\mathcal{L}(f_{w^*}(\alpha); \mathcal{D}^{val})$ evaluated on a validation set $\mathcal{D}^{val}$, where the network weights $w^*(\alpha)$ are obtained by minimizing the loss function on a training set $\mathcal{D}^{trn}$. Meanwhile, the expected energy

consumption is constrained to be smaller than a threshold c to promote energy efficient architecture search.

We first relax the energy constrained objective (1) as follows,

$$\min_{\theta} \mathbb{E}_{\alpha\sim\pi_\theta}\Big[\mathcal{L}(f_{w^*}(\alpha);\ \mathcal{D}^{val})\Big] + \lambda\Big[\mathbb{E}_{\alpha\sim\pi_\theta}[J(\alpha)] - c\Big]_+ \quad (2)$$

$$\text{s.t.}\quad w^* = \arg\min_{w} \mathcal{L}(f_w(\alpha);\ \mathcal{D}^{trn}, \alpha), \quad (3)$$

with $[x]_+ = \max(x, 0)$ and λ a hyper-parameter, which only penalizes the objective function when the energy constraint is violated.

However, it is still infeasible to solve the optimization problem (2) correctly. The computational difficulties arise in two-fold. First, the optimal model weights $w^*(\alpha)$ depend on model configuration α, an inner loop minimization is required whenever the control parameter θ is changed. Second, in general, energy-based objectives $J(\alpha)$ are non-differentiable, which prohibits the use of efficient back-propagation.

To solve the aforementioned challenges, motivated by DARTS [32], we approximate $w^*(\alpha)$ with one step gradient descent, specifically,

$$w'(\alpha) = w(\alpha) - \epsilon\nabla_w \mathcal{L}(f_w(\alpha); \mathcal{D}^{trn}, \alpha), \quad (4)$$

with ϵ as the step size. Without considering the energy related terms, our objective function is reduced to

$$\min_{\theta} \mathbb{E}_{\alpha\sim\pi_\theta}\Big[\mathcal{L}(f_{w'}(\alpha);\ \mathcal{D}^{val})\Big].$$

We then apply gradient descent for optimization, such that the model parameters θ are updated as follows:

$$\theta \leftarrow \theta - \beta\nabla_\theta \mathbb{E}_{\alpha\sim\pi_\theta}\Big[\mathcal{L}(f_{w'}(\alpha);\ \mathcal{D}^{val})\Big] \quad (5)$$

where β is the step size. In practice, the expected gradient is approximated using Monte Carlo samples $\{\alpha_i\}_{i=1}^m$ drawn from π_θ, combining with Equation (4), we can approximate Equation (5) as follows,

$$\theta \leftarrow \theta - \beta\nabla_\theta\Big[\frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(f_{w-\epsilon\nabla_w\mathcal{L}(f_w(\alpha_i);D^{trn})}; \mathcal{D}^{val})\Big]. \quad (6)$$

To deal with the non-differentiable energy measures, we use the REINFORCE algorithm [33]. The expected gradient can be computed as follows

$$\begin{aligned}\nabla_\theta \mathbb{E}_{\alpha\sim\pi_\theta} J(\alpha) &= \nabla_\theta \int J(\alpha)\pi_\theta(\alpha)d\alpha \\ &= \int J(\alpha)\nabla_\theta\pi_\theta(\alpha)d\alpha = \int J(\alpha)\pi_\theta(\alpha)\nabla_\theta \log\pi_\theta d\alpha \\ &= \mathbb{E}_{\alpha\sim\pi_\theta}\Big[J(\alpha)\nabla_\theta\log\pi_\theta\Big] \approx \frac{1}{k}\sum_{i=1}^{k} J(\alpha_i)\nabla_\theta\log\pi_\theta(\alpha_i). \quad (7)\end{aligned}$$

The full algorithm is outlined in Algorithm 1.

B. Mixed Precision Architecture Search Space

In order to search an energy efficient neural architecture and have a fair comparison with previous NAS algorithms [9; 10], we adopt the commonly used MobileNetV2 (MB) block [13] as the base search unit, which is demonstrated in Figure 2. Each MB block consists of two bottleneck layers (expansion and projection layer) and one depthwise convolution layer. First, the 1×1 expansion convolution layer expands the number of channels by a factor of m before the data goes into the depthwise convolution. Second, the depthwise convolution applies $k \times k$ filters to its input while keeping the number of output channels the same. Finally, a 1×1 projection convolution layer squeezes the network in order to match the initial number of channels. We allow each MB block with a filter size $k \in \{3, 5, 7\}$ and an expansion ratio $m \in \{1, 3, 6\}$.

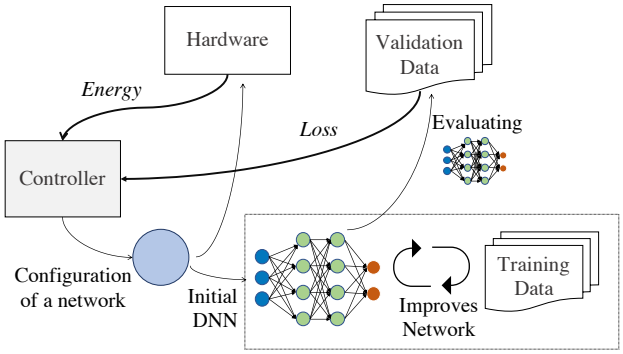

Fig. 1. Illustration of our energy aware neural architecture search framework.

We assume the final neural network architecture is hierarchical that stacks of a certain number of MB blocks. In order to search the number of MB blocks, we add a zero operation (skip connection, marked as s in Figure 2.) into the search space. The entire MB block will be skipped if the skip operation is selected (s: true).

In addition, we augment our NAS search space to allow each MB block to choose the quantization precision adaptively. Note that in a MB block the depthwise convolution layer normally contains more parameters than the bottleneck layers, and thus more energy hungry. We propose to assign two different bitwidths for each MB block, and search the optimal precision choices for the bottleneck layers (b_1) and the depthwise layer (b_2), respectively. And $b_1, b_2 \in \{2, 4, 6, 8\}$ bits. Table I shows the macro-architecture of our search space.

Input Shape	Block and Bit-width	#channels	stride	n
$224 \times 224 \times 3$	Conv3 $\times$ 3, 8bit	32	2	1
$112 \times 112 \times 32$		16	2	1
$56 \times 56 \times 16$		24	1	2
$56 \times 56 \times 24$		32	2	4
$28 \times 28 \times 32$	MB(s, m, k, b_1, b_2)	64	2	4
$14 \times 14 \times 64$		128	1	4
$14 \times 14 \times 128$		160	2	5
$7 \times 7 \times 160$		256	1	2
$7 \times 7 \times 256$	Conv1 $\times$ 1, 8bit	1280	1	1
$7 \times 7 \times 1280$	Pooling & FC, 8bit	-	1	1

TABLE I
THIS TABLE SHOWS THE MACRO-ARCHITECTURE OF THE SEARCH SPACE. AND n DENOTES THE NUMBER OF REPEATED MB LAYERS WITH THE SAME NUMBER OF CHANNELS. (s, m, k, b_1, b_2) STANDS FOR MB BLOCK CONFIGURATIONS.

C. Search Algorithm

We are now ready to show the representation of the network configuration α. Let $\eta_\ell = [s^\ell, m^\ell, k^\ell, b_1^\ell, b_2^\ell]$ be the discrete variables that associated with the MB block in the ℓ-th layer. A network configuration α is a stack of N MB blocks as a vector of $[\eta_1, ..., \eta_N] \in \mathbb{R}^{5N}$, where N denotes the number of MB blocks. Each element in α can be considered as a specific operation. Without loss of generality, denote $o(x)$ as one of the operator defined in α, which takes K possible values. E.g., $o(x)$ could be a depthwise convolution layer, in this case, all possible filter size choices are $\{3 \times 3, 5 \times 5, 7 \times 7\}$ and $K = 3$. Given the input x, $o(x)$ maps x to its corresponding output.

To learn the controller, it remains to show the representation of π_θ and the gradient of the validation loss $\nabla_\theta \mathbb{E}_{\alpha\sim\pi_\theta}[\mathcal{L}(f_{w'}(\alpha), \mathcal{D}^{val})]$

Algorithm 1 Energy aware neural architecture search

repeat
  Sample minibatch of network configurations $\{\alpha_i\}_{i=1}^{k}$ from the controller π_θ
  Update network models $\{w(\alpha_i)\}$ by minimizing the training loss following Eqn. 4.
  Update the controller parameters θ following Eqn. (6) and (7).
until Converge

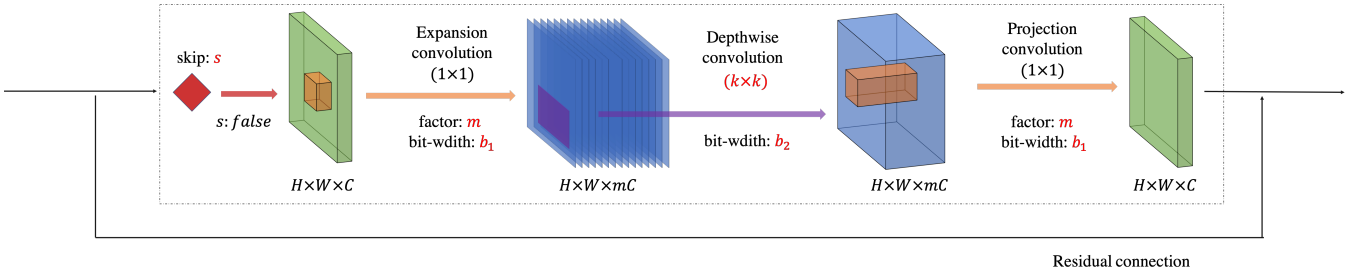

Fig. 2. An illustration of the structure of a MB search unit. m denotes the expand ratio, k denotes the kernel size, s denotes whether skipping this block, and b_1, b_2 denotes the layer-wise bitwidths.

w.r.t. θ defined in Eqn. 6. For each operator $o(x)$, we represent discrete valued architecture choices as a one-hot vector $d \in \mathbb{R}^K$, such that $d_j \in \{0, 1\}$ and $\sum_{j=1}^{K} d_j = 1$. In this way, the corresponding output of applying operator o with representation d is $\sum_{j=1}^{K} d_j o_j(x)$, where $o_j(x)$ denotes the forward operation with j-th candidate in the search space (e.g., o_1 could be 3×3 depthwise convolution). We parametrize $p(d_j = 1) = \exp(\phi_j)/\sum_{i=1}^{K} \exp(\phi_i)$, where $\{\phi_j\}$ are trainable parameters that belongs to the controller parameters θ. It is problematic to calculate the gradients directly due to the inability to back-propagate through categorical samples. We thus replace the non-differentiable network configuration samples with continuous differentiable samples from a Gumbel-Softmax distribution [21]. That is, we replace one-hot representation d as a continuous vector such that $d_j \geq 0, \sum_{j=1}^{K} d_j = 1$, which can be trained using standard back-propagation. For the rest, we follow the settings for Gumbel-softmax sampling used in FBNet [9].

Training with Quantized Weights. In our approach, only the quantized values of the weights and activations are used in all forward operations. In order to learn quantized weights, we follow the linear quantization schemes suggested by [2; 3]. Specifically, for a layer with $n \times d$ dimensional input $\boldsymbol{x} = (x_1, ..., x_n), x_i \in \mathbb{R}^d$, we quantize each x_i linearly into b bits:

$$\text{Quantize}(x_i, b) = \text{round}(\text{clamp}(x_i, c)/s) \times s,$$

where clamp truncates all values into the range of $[-c, c]$, with $c = \max\{|x_i|\}$. And the scaling factor s is defined as $s = c/(2^{b-1} - 1)$.

IV. Experimental Results

We search architectures and mixed quantization precision for each layer on a proxy task, tiny ImageNet [1]. Next, we train the discovered architectures on two image classification benchmarks (CIFAR-100 and ImageNet) from scratch. We show that our method achieves competitive (or better) accuracy compared with state-of-the-art deep neural architectures and simultaneously yields significant smaller model size and lower energy consumption.

Low precision neural network baselines are quantized and then finetuned using the standard compression methods proposed in [24] unless otherwise specified.

[1]https://tiny-imagenet.herokuapp.com/

A. NAS

Energy Modeling. We use the simulator for Bit Fusion [2] [34] to obtain the hardware-related performance metrics. Bit Fusion employs a 2D systolic array of bit-level processing elements that dynamically fuse to match the bitwidth of every single layer. For a fair comparison, we exactly follow the setting described in HAQ [3] for energy estimation. All the energy consumption discussed below are for the inference with a batch of 16 images.

Settings. The tiny ImageNet dataset consists of 200 classes. Each class contains 500 training images, 50 validation images, and 50 test images, respectively. For preprocessing, we resize the image to the size of 224×224. We use a batch size of 64 and adopt label smoothing for training. Our algorithm is trained for a fixed 60 epochs during the architecture search process. We run our search program on one NVIDIA Tesla P100 GPU, which takes five days.

Results. Note that we jointly optimize the task-oriented loss and energy constrains, the trade-off between these two terms are controlled by a hyperparameter λ (defined in Equation 2). Large λ values lead to smaller energy efficient models while less accurate on the prediction task; vice versa. We evaluate two different settings $\lambda = 0.1, 0.01$, which result in one smaller and one bigger neural architecture (see Figure 3), respectively. We can see from Figure 3 that depthwise convolution layers (more parameters) are allocated with fewer bitwidths than bottleneck convolution layers (fewer parameters), as also observed in HAQ [3]. In addition, we notice that bottleneck layers with small expand ratios (fewer parameters) tend to have larger bitwidths to preserve good accuracy.

B. Experiments on ImageNet

Dataset. The ILSVRC 2012 classification dataset [27] consists of 1.2 million training images, and 50,000 validation images, with 1,000 classes. We resize the image size to 224×224, and adopt the standard data augment scheme (mirroring and shifting) for training images.

Settings. For ImageNet, we set the batch size to be 64, and fix the number of the filters in the first convolution layer to be 32. We use stochastic gradient descent with an initial learning rate 10^{-4} and apply cosine learning rate annealing scheduling [39]. We also use label smoothing [40] ($\alpha = 0.1$), mixup [37] ($\alpha = 0.2$

[2]https://github.com/hsharma35/bitfusion

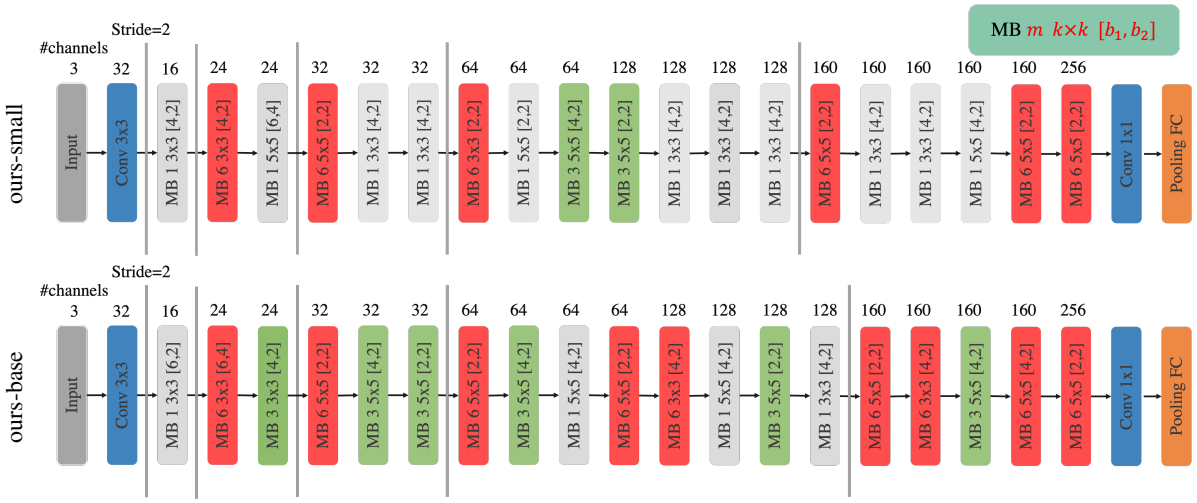

Fig. 3. Two energy efficient deep neural architectures found by our method. "MB m $k \times$ k $[b_1, b_2]$ " represents a mobile inverted bottleneck convolution layer, for which m stands for the expansion ratio, $k \times k$ is the filter size for the the depthwise convolution layer, b_1 and b_2 represents the bit-width precision for the bottleneck layers (expansion and projection convolution layers) and the depthwise layer, respectively.

Model	Precision	Top-1 Error (%)	Top-5 Error (%)	Model Size (Bytes)	Energy (mJ)	Latency (ms)
VGG-16 [35]	8-bit	29.10	7.40	138.00M	753.11	838.03
ResNet-50 [36]	8-bit	24.70	5.30	25.50M	557.46	591.17
MobileNetV2 [3]	8-bit	28.19	9.75	3.40M	29.01	73.85
FBNet-B [9]	8-bit	26.84	8.97	4.50M	34.65	83.91
FBNet-B [9]	3-bit	36.29	15.38	1.68M	13.47	27.93
HAQ-small [3]	*mixed*	33.01	12.67	1.70M	12.85	32.10
ours-small	*mixed*	**31.62**	**11.56**	**1.44M**	**8.91**	**21.19**
HAQ-base [3]	*mixed*	29.10	10.09	2.12M	16.31	40.21
Ours-base	*mixed*	**28.23**	**9.94**	**2.06M**	**10.85**	**24.71**

TABLE II
RESULTS ON THE IMAGENET2012 DATASET.

Model	Precision	Model Size (Bytes)	Error (%)	Energy (mJ)	Latency (ms)
DenseNet-BC-190 + Mixup [37]	8-bit	26.0M	17.02	125.74	247.21
ENAS + Cutout [38]	8-bit	4.6M	16.58	25.93	49.78
NAO + Cutout [18]	8-bit	10.8M	**15.87**	37.20	75.92
MobileNetV2	8-bit	2.5M	21.85	5.09	13.13
FBNet-B [9]	8-bit	2.8M	21.36	8.27	17.48
HAQ-small [3]	*mixed*	0.6M	22.93	1.17	2.75
ours-small	*mixed*	0.6M	**22.16**	**1.00**	**2.53**
HAQ-base [3]	*mixed*	0.8M	21.89	1.53	3.31
ours-base	*mixed*	0.8M	**21.27**	**1.21**	**2.98**

TABLE III
RESULTS ON THE CIFAR-100 DATASET.

) for data augmentation, and clip the gradient to the range of $[-5, 5]$. We replace the vanilla batch normalization layer with 8-bit range normalization following [29], and train our discovered neural architectures for 120 epochs on the training set.

Results. The performance of our models (denoted as *ours-small* and *ours-base*) is reported in Table II along with other state-of-the-art approaches. We report the top-1 and top-5 classification error on the validation set. In addition to the energy consumption, we also show the latency and model size off all evaluated approaches. As shown in Table II, our discovered models can extremely reduce the energy while achieving on par (better) accuracy compared to strong baseline approaches.

In the first block, we can see that classic DNNs are most energy hungry. For example, VGG-16 [35] costs 753.11mJ energy along with 838.03ms latency, ResNet-50 [36] costs 557.46mJ energy along with 591.17ms latency. The energy consumption of these two models is about 50× higher than our models. To have a fair comparison with HAQ [3] with similar level of energy consumption, we compare with two variants of HAQ, *HAQ-small* and *HAQ-base*: *HAQ-small* denotes the energy-conserving setting with more aggressive quantization strategies; *HAQ-base* denotes the setting favors better accuracy and thus larger models. Compared to *HAQ-small*, *ours-small* reduces the top-1 error rate from 33.01% to 31.62% and reduces energy by 3.94mJ. *Ours-base* also achieves lower top-1 error rate (30.60% → 28.23%) and leads to about 3× lower energy consumption (30.60mJ→ 10.85mJ) than *HAQ-Base*.

Compared to 8-bit MobileNetV2, *ours-base* achieves 35% reduction on model size and 63% reduction on energy, respectively.

Compared with 3-bit FBNet-B [9], which has similar energy cost to *ours-base*, we can improve the top-1 error rate from 36.29% to 28.23%. Besides, our discovered models can be fitted into on-chip SRAM cache (5pJ per access under 45nm CMOS technology) rather than off-chip DRAM memory (640pJ per access under 45nm CMOS technology). Shown in Table II, the model size of *ours-small* is 1.44MB, which could be accommodated into on-chip SRAM cache.

C. Experiments on CIFAR-100

Dataset. CIFAR-100 [3] consists of images drawn from 100 classes. The training and test sets contain 50,000 and 10,000 images respectively. We adopt a standard data argumentation scheme (mirroring and shifting) that is widely used for this dataset.

Settings. We set the batch size to be 128, set the learning rate to be 0.1, remove label smoothing and remove the first two down-sample convolutions in the architecture. We follow the other settings in the ImageNet task.

Results. The main results on CIFAR-100 are shown in table III, we can see that pioneer state-of-the-art architectures, e.g. 8-bit DenseNet [41], lead to significant energy and memory cost, which is 125.74mJ and 26.0MB respectively. Recent NAS frameworks only focus on accuracy optimization that also results in architectures with notable energy consumption. For instance, 8-bit NAO [18] uses 37.20mJ; 8-bit ENAS [38] costs 25.93mJ. On the other hand, *ours-small* achieves $\times 5$ reduction in model size and more than $\times 5$ reduction in energy consumption while achieving better accuracy compared to the 8-bit MobileNetV2. Compared to HAQ, *ours-small* costs less energy than HAQ-small, meanwhile, improves the error rate from 22.93% to 22.16%. *Ours-base* can achieve 0.32mJ energy reduction than HAQ-base, and improve the error rate from 21.89% to 21.27%.

Optimization	Model Size (Bytes)	Error(%)	Energy (mJ)
NAS+Quantization (Small)	0.5M	22.34	1.04
ours-small	0.5M	**22.16**	**1.00**
NAS+Quantization (Base)	0.9M	21.58	1.28
Ours-base	0.8M	**21.27**	**1.21**

TABLE IV
COMPARISON OF STEPWISE NAS AND MODEL QUANTIZATION V.S. OUR JOINT-OPTIMIZING FRAMEWORK. THE MODELS ARE EVALUATED ON THE CIFAR-100 DATASET.

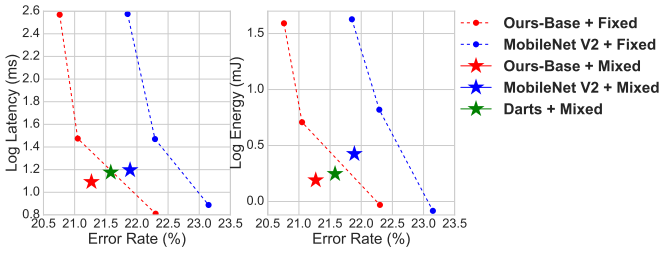

Fig. 4. Comparison of mixed precision quantization (denoted as 'mixed') v.s. constant precision quantization (denoted as 'fixed'). The dashed line shows the results for $\{8, 4, 2\}$ bitwidths (left to right), respectively.

D. Joint NAS and Adaptive Mixed Precision Quantization

We further study the importance of joint NAS and mixed precision quantization. On the CIFAR-100 dataset, we study a baseline approach by performing NAS and model quantization stepwisely. Specifically, we use DARTS [32] [4] for architecture search with the same search space defined in Table I while using 8-bits precision for all layers. Next, we perform layer-wise model quantization following HAQ. In a way similar to our approach, we set $\lambda = 0.1, 0.01$, respectively, and discover two neural models (NAS+Quantization (small) and NAS+Quantization (base)) with similar model size compared to *ours-small* and *ours-base*.

As we can see from Table IV, compared to *NAS+Quantization (small)*, *ours-small* reduces the error rate from 22.34% to 22.16% and also reduce the energy cost by around 0.05mJ. Compared to *NAS+Qunatization (base)*, *Ours-base* reduces the error rate from 21.58% to 21.27% and achieves a 5% reduction on energy consumption. These results highlights the benefits of joint NAS and mixed precision quantization as suggested by our approach.

In Figure 4, we show the advantage of adaptive mixed precision quantization versus constant precision quantization. In Figure 4, the dashed line shows the performance (latency v.s. classification error rate) of different baseline architectures with a constant 8-bit, 4-bit, and 2-bit precision, respectively. The star points show the results of HAQ and our method, that both allow flexible layer-wise bitwidths. With mixed layer-wise precision searching, both HAQ and our method discovers neural architectures yields good trade-off between accuracy and latency, and our method outperforms HAQ.

V. CONCLUSION

In this work, we propose a new methodology to perform NAS and mixed precision quantization in the extended search space with hardware performance involved in the objective function. Experimental results demonstrate that our proposed approach achieves better energy efficiency than advanced quantization approaches on CIFAR-100 and ImageNet. Our methodology facilitates the end-to-end design automation flow of neural network design and deployment, especially the edge inference.

ACKNOWLEDGEMENT

The authors would like to thank Hardik Sharma from Georgia Institute of Technology for the assistance on Bit fusion. The authors also appreciate constructive comments and discussions provided by Wuxi Li, Lemeng Wu, Keren Zhu, Mingjie Liu, Jiaqi Gu from the University of Texas at Austin.

REFERENCES

[1] X. Xu, Y. Ding, S. X. Hu, M. Niemier, J. Cong, Y. Hu, and Y. Shi, "Scaling for edge inference of deep neural networks," *Nature Electronics*, vol. 1, no. 4, p. 216, 2018.

[2] A. Ren, T. Zhang, S. Ye, J. Li, W. Xu, X. Qian, X. Lin, and Y. Wang, "Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers," in *Proc. ASPLOS*. ACM, 2019, pp. 925–938.

[3] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: hardware-aware automated quantization with mixed precision," in *Proc. CVPR*, 2019, pp. 8612–8620.

[4] B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer, "Mixed precision quantization of convnets via differentiable neural architecture search," *arXiv preprint arXiv:1812.00090*, 2018.

[5] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *Proc. ICLR*, 2019.

[3] https://www.cs.toronto.edu/~kriz/cifar.html

[4] https://github.com/quark0/darts

[6] J. Chen, K. Chen, X. Chen, X. Qiu, and X. Huang, "Exploring shared structures and hierarchies for multiple nlp tasks," *arXiv preprint arXiv:1808.07658*, 2018.

[7] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. CVPR*, 2018, pp. 8697–8710.

[8] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: stochastic neural architecture search," *CoRR*, vol. abs/1812.09926, 2018.

[9] 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," *arXiv preprint arXiv:1812.03443*, 2018.

[10] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, "Single path one-shot neural architecture search with uniform sampling," *arXiv preprint arXiv:1904.00420*, 2019.

[11] C. Ding, S. Liao, Y. Wang, Z. Li, N. Liu, Y. Zhuo, C. Wang, X. Qian, Y. Bai, G. Yuan, X. Ma, Y. Zhang, J. Tang, Q. Qiu, X. Lin, and B. Yuan, "Circnn: Accelerating and compressing deep neural networks using block-circulant weight matrices," in *Proc. MICRO*. ACM, 2017, pp. 395–408.

[12] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. NeurIPS*. Curran Associates Inc., 2016, pp. 2082–2090.

[13] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *CoRR*, vol. abs/1801.04381, 2018.

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

[15] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[16] D. Marculescu, D. Stamoulis, and E. Cai, "Hardware-aware machine learning: Modeling and optimization," in *Proc. ICCAD*. ACM, 2018, pp. 137:1–137:8.

[17] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proc. CVPR*, 2018, pp. 6848–6856.

[18] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," in *Proc. NeurIPS*, 2018, pp. 7827–7838.

[19] D. Stamoulis, T.-W. R. Chin, A. K. Prakash, H. Fang, S. Sajja, M. Bognar, and D. Marculescu, "Designing adaptive neural networks for energy-constrained image classification," in *Proc. ICCAD*. ACM, 2018, pp. 23:1–23:8.

[20] T. Véniat and L. Denoyer, "Learning time/memory-efficient deep architectures with budgeted super networks," in *Proc. CVPR*, 2018, pp. 3492–3500.

[21] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[22] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Proc. NeurIPS*, 2015, pp. 3123–3131.

[23] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *arXiv preprint arXiv:1810.05270*, 2018.

[24] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[25] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *CoRR*, vol. abs/1806.08342, 2018.

[26] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Proc. ECCV*. Springer, 2016, pp. 525–542.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. NeurIPS*, 2012, pp. 1097–1105.

[28] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. CVPR*, 2018, pp. 2704–2713.

[29] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, "Scalable methods for 8-bit training of neural networks," in *Proc. NeurIPS*, 2018, pp. 5151–5159.

[30] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[31] G. Lacey, G. W. Taylor, and S. Areibi, "Stochastic layer-wise precision in deep neural networks," *arXiv preprint arXiv:1807.00942*, 2018.

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

[33] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[34] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *Proc. ISCA*, June 2018, pp. 764–775.

[35] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, June 2016.

[37] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," *arXiv preprint arXiv:1710.09412*, 2017.

[38] 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.

[39] J. Xie, T. He, Z. Zhang, H. Zhang, Z. Zhang, and M. Li, "Bag of tricks for image classification with convolutional neural networks," *arXiv preprint arXiv:1812.01187*, 2018.

[40] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. CVPR*, 2016, pp. 2818–2826.

[41] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. CVPR*, 2017, pp. 4700–4708.