Automated Optimization of Deep Neural Networks: Dynamic Bit-Width and Layer-Width Selection via Cluster-Based Parzen Estimation

Seyedarmin Azizi*, Mahdi Nazemi*, Arash Fayyazi, and Massoud Pedram
Department of Electrical & Computer Engineering, University of Southern California, Los Angeles, CA, USA
{seyedarm,mnazemi,fayyazi,pedram}@usc.edu

***Abstract*—Given the ever-growing complexity and computational requirements of deep learning models, it has become imperative to efficiently optimize neural network architectures. This paper presents an automated, search-based method for optimizing the bit-width and layer-width of individual neural network layers, achieving substantial reductions in the size and processing requirements of these models. Our unique approach employs Hessian-based search space pruning to discard unpromising solutions, greatly reducing the search space. We further refine the optimization process by introducing a novel, adaptive algorithm that combines k-means clustering with tree-structured Parzen estimation. This allows us to dynamically adjust a figure of merit used in tree-structured Parzen estimation, i.e., the desirability of a particular bit-width and layer-width configuration, thereby expediting the identification of optimal configurations. Through extensive experiments on benchmark datasets, we validate the efficacy of our method. More precisely, our method outperforms existing techniques by achieving an average 20% reduction in model size without sacrificing any output accuracy. It also boasts a 12× acceleration in search time compared to the most advanced search-based approaches.**

I. Introduction

Deep neural networks (DNNs) have emerged as powerful and versatile tools for tackling real-world problems across various domains, including computer vision [1]–[3] and natural language processing [4], [5]. The remarkable performance of DNNs can be attributed to their capacity to learn intricate patterns and representations from large-scale data, involving millions or even billions of parameters. However, this success comes at the expense of high compute cycles, memory footprint, I/O bandwidth, and energy consumption, resulting in an increased carbon footprint and limitations in deploying them on resource-constrained platforms.

To address these challenges, quantization and structured pruning techniques have emerged as promising strategies, offering the potential to mitigate the computational and memory burden of DNNs while maintaining satisfactory accuracy [6]–[9]. These techniques have proven effective in making DNNs more efficient, enabling their deployment on edge devices, and accelerating inference in cloud environments. However, the successful application of quantization and structured pruning heavily relies on finding the optimal bit-width and layer-width for each layer of the DNN, which presents a formidable challenge due to the exponential growth of the search space as the number of layers increases.

The challenges in optimizing the bit-width for each layer of a DNN are compounded by the inherent diversity in weight distributions across different layers (see Fig. 1) and the sensitivity of a DNN's predictions to each layer's weights. The weight distributions and sensitivity values can vary significantly, and therefore, different layers may benefit from different bit-widths to achieve maximum gains in terms of memory and computation cost savings without compromising accuracy.

Additionally, widening layers can profoundly impact the accuracy of the model [2]. Therefore, the search process should not be confined to reducing the layer-width only; it must also identify situations where widening a layer, combined with a sufficiently reduced bit-width, leads to potential accuracy gains without compromising cost savings.

The search process for optimizing the bit-width and layer-width of DNNs should extend beyond traditional metrics such as FLOPs and memory footprint when evaluating cost savings. While these metrics are useful, they may not directly reflect the real-world model performance and efficiency. Therefore, it is imperative to consider factors like latency, throughput, and energy consumption when comparing models. Hence, devising an algorithm that effectively explores the bit-width and layer-width search space while considering the diverse weight distributions and sensitivity values and the varying impact of layer-width on accuracy and real-world performance becomes critical. This necessitates an approach that can intelligently adapt the bit-width and layer-width for each layer, striking a balance between latency, throughput, or energy consumption and the preservation of model accuracy.

This paper presents a novel model-based optimization method based on tree-structured Parzen estimators (TPEs) [10] to address the challenge of simultaneously searching for the optimal bit-width and layer-width values for DNN layers. The major innovations of our optimization method are as follows.

Handling Loss Landscapes: DNNs often exhibit loss landscapes with wide flat minima that are sparsely distributed among dominant critical points (absolute narrow minima, local minima, or saddle points in the loss surface), which poses challenges for conventional optimizers. To address this, we extend vanilla TPE to have a new category of *undecided* configurations in addition to desirable and undesirable ones. Additionally, we dynamically adjust the definition of desirable and undesirable configurations of bit-widths and layer-widths before fitting surrogate distributions to observations of the objective function. This novel approach enables us to achieve comparable or better objective values compared to state-of-the-art (SOTA) optimizers while significantly reducing convergence time, typically by a factor of at least 10x.

Joint Optimization of Bit-Width and Layer-Width: Unlike

*Seyedarmin Azizi and Mahdi Nazemi contributed equally to this work.

979-8-3503-4859-0/DATE24/© 2024 EDAA

approaches that treat bit-width and layer-width independently, we search for joint optimal configurations, which enables us to discover configurations that yield higher-quality results.

Exploiting Second-Order Derivatives: In addition to leveraging the distribution of layer weights, our method incorporates the Hessian of the loss function with respect to layer weights to prune the search space when dealing with pre-trained models, resulting in enhanced efficiency. For example, instead of considering bit-widths $B = \{8, 6, 4, 3, 2\}$ in a DNN optimization problem, only $\{8, 6\}$ may be considered for a sensitive layer.

Hardware-Aware Objective Function: Our approach accounts for essential information about the target hardware that will execute the optimized DNN. It takes latency, throughput, and energy consumption models as its inputs, which, combined with model accuracy, define a composite objective function guiding the optimization process.

II. Related Work

Mixed-precision quantization (MPQ) exploits the fact that not all model parameters require the same level of precision to maintain model accuracy. Quantization-aware training techniques simulate the quantization effects during the training process, allowing the model to adapt to lower precision, while post-training quantization techniques quantize DNN models after they have been trained at full precision. The former typically achieves higher accuracy at the expense of increased time.

In sensitivity-based quantization, first- or second-order gradient statistics, or other sensitivity metrics, are employed as a proxy to layer importance, which is then used to assign the bit-width to each layer [11]–[13]. However, sensitivity-based quantization has some shortcomings. First, it does not take into account the influence of quantized weights and input activations on output activations, which are inputs to other layers of a DNN including batch normalization layers commonly found after convolutional layers. In other words, the batch statistics of batch normalization layers or weights of succeeding convolutional layers are trained with full-precision input activations, and any change in those input activations due to quantization may hamper model accuracy. Second, gradient-based methods utilize gradients calculated on the full-precision model, thus overlooking the impact of quantization on gradients, which is a significant effect when using ultra-low bit-width quantization. Third, the sensitivity values are often calculated based on the weights of layers, and thus, do not provide any insight into the proper bit-width of input activations. Sensitivity-based quantization approaches often fall short of achieving high model size compression ratios and/or maintaining accuracy, leaving ample space for further optimization.

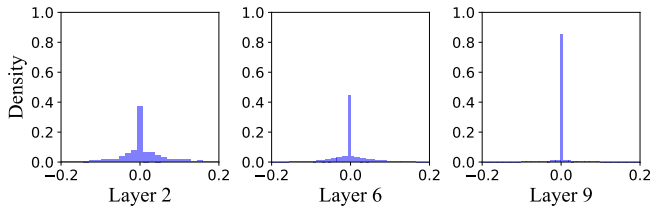

Fig. 1. Distribution of weights in three representative layers of the MobileNetV1 architecture trained on the CIFAR-100 dataset.

In reinforcement learning (RL)-based MPQ, an agent interacts with a quantized DNN environment, adjusting the bit-width configurations for various layers using RL algorithms, such as policy gradients, which provide rewards based on model accuracy and resource consumption, enabling the agent to discover optimal quantization strategies through iterative exploration and exploitation [14]–[16]. Despite their potential benefits, these RL-based techniques are confronted with a significant challenge: the considerable search time involved in the RL training process. As a consequence, achieving favorable results within specified GPU-hour constraints becomes challenging.

Differentiable search-based approaches build upon the concept of differentiable architecture search [17] and apply it to MPQ. In these approaches, a super-network is trained to replace each layer (or activation function) of a DNN, such as a convolutional layer, with parallel branches where each branch implements a quantized version of the layer (or activation function) [18]. These approaches have significant drawbacks. First, they suffer from large training times and high GPU RAM requirements due to the very large size of the super-network. Second, the distribution defined over each group of parallel branches, which is found during training, may not converge to a uni-modal distribution, rendering the selection of a single quantized version of a layer infeasible.

Lastly, sequential model-based MPQ approaches employ surrogate models that define a mapping between the search space and an objective function to guide the exploration of the search space [19]. Tree-structured Parzen estimator is a powerful tool that has shown great success in hyperparameter tuning [20]. However, its naive application to MPQ ignores the characteristics of the loss landscapes that are prevalent in DNNs. This greatly increases the search time and yields inferior objective values. Our TPE-based optimization achieves a $12\times$ average search time speedup and a 20% reduction in model size compared to [19] while preserving the model accuracy.

III. Proposed Method

This section details the three main components of our model-based bit-width and layer-width optimization framework, i.e., K-Means TPE, Hessian-based search space pruning, and hardware-aware objective function definition.

A. K-Means TPE

An integral part of the presented framework is a novel sequential model-based optimization methodology that is based on the TPE methods. The main idea behind TPE methods is to recursively partition the data space into smaller regions (nodes) and estimate the probability density function within each region separately. In particular, the TPE methods use Bayesian reasoning to propose configurations from the search space that are likely to improve an objective function as explained next. After drawing a few random configurations

$\chi = \{x^{(1)}, x^{(2)}, \ldots, x^{(k)}\}$ from the search space and observing their corresponding objective function values $Y = \{y^{(1)}, y^{(2)}, \ldots, y^{(k)}\}$, the TPE methods define a threshold $\hat{y}$ that is equal to the largest of q-quantiles of Y, and creates a surrogate distribution $l(x)$ for configurations with desirable objective function values ($y^{(i)} \geq \hat{y}$ in a maximization problem) and a surrogate distribution $g(x)$ for configurations with undesirable objective function values ($y^{(i)} < \hat{y}$.) The candidate configuration to be evaluated next is an $\tilde{x}$ that maximizes the $\frac{l(x)}{g(x)}$ ratio. The chosen $\tilde{y}$ leads to an update in $\hat{y}$, $l(x)$, and $g(x)$ until a stopping criterion is met.

In the case of stochastic gradient descent and cross-entropy loss, a slow reduction of the norm of the weights along the learning process leads to a loss landscape with wide flat minima. Although these wide minima are rare compared to the dominant critical points, they can be accessed by a large family of simple learning algorithms [21]. Due to these loss landscapes, widely different configurations from the search space may yield very close objective values. This becomes problematic when objective values of configurations from promising parts of the search space fall slightly below $\hat{y}$, which puts those configurations in $g(x)$. This effectively discourages exploring those parts of the search space, which, in turn, can yield configurations with inferior objective values.

To address this problem, one may introduce a TPE method that employs two thresholds: $\hat{y}_{\mathrm{L}}$ and $\hat{y}_{\mathrm{H}}$ and subsequently creates a first surrogate distribution $l(x)$ for configurations with desirable objective function values ($y^{(i)} \geq \hat{y}_{\mathrm{H}}$) and a second surrogate distribution $g(x)$ for configurations with undesirable objective function values ($y^{(i)} \leq \hat{y}_{\mathrm{L}}$), leaving the remaining configurations in a third *undecided* category. In practice, we have encountered challenges in finding $\hat{y}_{\mathrm{L}}$ and $\hat{y}_{\mathrm{H}}$, and it has become apparent that doing so is unnecessary. A more effective approach involves leveraging the k-means clustering technique to first construct a set of clusters, and then, using information about these clusters to differentiate between desirable (good) and undesirable (bad) distributions. Details are provided next.

The proposed k-means cluster-based TPE optimizer, which we refer to as k-means TPE, commences by selecting an initial k value (where $k \geq 3$). It then organizes the elements within Y into clusters $C_1, \ldots, C_k$ using the k-means clustering method. Subsequently, it calculates the centroid value for each cluster, determined as the mean value of the $y^{(j)}$ values assigned to that particular cluster. These clusters are then sorted in non-increasing order based on their centroid values, resulting in $C_{\pi(1)}, \ldots, C_{\pi(k)}$, with $\pi(.)$ denoting a permutation. The surrogate distributions are then obtained as follows:

$$p(x|y) = \begin{cases} l(x), & \text{if } y \in C_{\pi(1)} \\ g(x), & \text{if } y \in C_{\pi(k)} \end{cases}.$$

It is important to note that after every few iterations of the search process, we increase k and update the clusters, with the effect of tightening the criteria for being a desirable versus an undesirable configuration. This effectively implements an annealing process that initially allows for large moves in the search space to explore distant regions of the search space, and gradually reduces the move size so that the search is narrowed to nearby regions of the search space (i.e., those regions that are close to the currently found promising solutions.)

B. Hessian-Based Search Space Pruning

The size of the search space grows exponentially as the number of DNN layers increases. As a result, pruning the search space by eliminating the bit-width choices that are likely to hamper the model accuracy or cause unnecessary computations is of paramount importance due to its considerable reduction in the size of the search space. The Hessian of the loss function with respect to each layer's weights provides an excellent starting point to evaluate the criticality of the bit-width for each layer [11]. We first prove an important result.

Lemma 1. *The maximum error induced in a DNN's output by unit perturbation in a layer's parameters is bounded by the trace value of the Hessian matrix of the loss with respect to that layer's parameters.*

Proof. Assume we freeze all DNN parameters except those of a single convolutional filter or neuron in layer l. Let $\boldsymbol{w}_l$ and $\boldsymbol{w}_l^{\mathrm{q}}$ denote the parameters of layer l and their corresponding quantized values, respectively. With $\Delta\boldsymbol{w}_l = \boldsymbol{w}_l^{\mathrm{q}} - \boldsymbol{w}_l$, the Taylor's Theorem implies that the output loss may be approximated around $\boldsymbol{w}_l^{\mathrm{q}}$ as follows:

$$\mathcal{L}(\boldsymbol{w}_l^{\mathrm{q}}) \approx \mathcal{L}(\boldsymbol{w}_l) + (\Delta\boldsymbol{w}_l)^{\mathrm{T}} \nabla\mathcal{L}_{\boldsymbol{w}_l} + \frac{1}{2}(\Delta\boldsymbol{w}_l)^{\mathrm{T}} \boldsymbol{H}_{\boldsymbol{w}_l} (\Delta\boldsymbol{w}_l),$$

where $\nabla\mathcal{L}_{\boldsymbol{w}_l}$ is the gradient vector of the loss function with respect to parameters $\boldsymbol{w}$ evaluated at $\boldsymbol{w}_l$, which is nearly zero on a trained model, and $\boldsymbol{H}_{\boldsymbol{w}_l}$ denotes the Hessian matrix, whose entries are the second derivatives of the loss function with respect to parameters $\boldsymbol{w}$ evaluated at $\boldsymbol{w}_l$. We have:

$$\mathcal{L}(\boldsymbol{w}_l^{\mathrm{q}}) - \mathcal{L}(\boldsymbol{w}_l) \approx \frac{1}{2}(\Delta\boldsymbol{w}_l)^{\mathrm{T}} \boldsymbol{H}_{\boldsymbol{w}_l} (\Delta\boldsymbol{w}_l). \tag{1}$$

By writing the spectral decomposition of the Hessian matrix as $\boldsymbol{H}_{\boldsymbol{w}_l} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T$, where $\boldsymbol{U}$ is a unitary matrix and $\boldsymbol{D}$ is a diagonal matrix, and plugging it back into (1), we obtain:

$$\Delta\mathcal{L}(\boldsymbol{w}_l) \approx \frac{1}{2}(\boldsymbol{U}^{\mathrm{T}}\Delta\boldsymbol{w}_l)^{\mathrm{T}} \boldsymbol{D} (\boldsymbol{U}^{\mathrm{T}}\Delta\boldsymbol{w}_l).$$

Denoting $\boldsymbol{U}^T\Delta\boldsymbol{w}_l$ with $\boldsymbol{a}$, we have:

$$\Delta\mathcal{L}(\boldsymbol{w}_l) \approx \frac{1}{2}\boldsymbol{a}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{a} = \frac{1}{2}\sum_i a_i^2\lambda_i \leq \frac{1}{2}\max_i(a_i^2)\sum_i \lambda_i.$$

Since $\boldsymbol{U}$ is a unitary matrix and the lemma assumes unit perturbation, $\max_i(a_i^2) \leq 1$. Therefore, $\Delta\mathcal{L}(\boldsymbol{w}_l) \leq \frac{1}{2}\mathrm{Tr}(\boldsymbol{H}_{\boldsymbol{w}_l})$.

The significance of this lemma lies in its implications for training and quantizing DNNs. For example, a small trace value of the Hessian matrix indicates that the loss function's surface is relatively flat around the current parameter values, making the model less sensitive to perturbations and potentially more stable during training. On the other hand, a large trace value suggests that the loss function's surface has steeper curves, indicating higher sensitivity to parameter changes and potentially making training more challenging. Owing to the prohibitively high computational cost associated with the exact calculation of the

Hessian matrix, we utilize the Hutchinson Hessian trace approximation [22]. In practical terms, the computational expense of the trace calculation is markedly minimal, constituting less than 1% of the total GPU-hour cost.

The Hessian-based search space pruning algorithm starts by normalizing the trace of the Hessian of the loss function with respect to each layer's weight by the number of weights in that layer to find an estimate of the relative importance of layers in a DNN. It then applies k-means clustering to the normalized trace values, sorts clusters in non-increasing order of their centroid values, and assigns candidate bit-widths to the layers within each cluster according to these centroid values by assigning higher bit-widths to layers that are within clusters with larger trace values. We use the same bit-width for weights and input activations of a DNN layer for improved hardware performance. This is mainly because most accelerators are optimized for homogeneous bit-widths for their operands. Moreover, the search space is considerably smaller and easier to explore in this case. For example, when $k = 4$, possibly overlapping subsets of candidate bit-widths $B = \{8, 6, 4, 3, 2\}$ can be considered for different clusters, e.g., $B_1 = \{8, 6\}$, $B_2 = \{6, 4, 3\}$, $B_3 = \{4, 3, 2\}$, and $B_4 = \{3, 2\}$, where the bit-widths of layers within the cluster with the largest centroid value are selected from B_1, and those for the second largest centroid value are chosen from B_2, and so on. If there are L layers, the size of the search space shifts from $|B|^L$ to $|B_1|^{|cluster_1|} \times |B_2|^{|cluster_2|} \times |B_3|^{|cluster_3|} \times |B_4|^{|cluster_4|}$, representing a significant reduction. The part of the search space defined by the layer-width values is not pruned, and layer-width is always taken from the set $S = \{0.75, 0.875, 1, 1.125, 1.25\}$.

C. Hardware-Aware Objective Function

The problem addressed in this work is mathematically formulated in its general form as follows:

$$\begin{aligned} \max_{B,S} \quad & \text{accuracy}(\boldsymbol{\Theta}, \mathcal{D}, B, S) \\ \text{s.t.} \quad & \text{modelSize}(\boldsymbol{\Theta}, B, S) \leq \mu, \text{latency}(\boldsymbol{\Theta}, \mathcal{H}, B, S) \leq \tau \\ & \text{energy}(\boldsymbol{\Theta}, \mathcal{H}, B, S) \leq \epsilon, \text{throughput}(\boldsymbol{\Theta}, \mathcal{H}, B, S) \geq \pi, \end{aligned}$$

where $\boldsymbol{\Theta}$ denotes parameters of the target DNN to be optimized, $\mathcal{D}$ comprises samples from a target dataset, B and S denote the sets of candidate bit-widths and layer-widths multipliers, respectively, $\mathcal{H}$ characterizes the target hardware, μ, τ, and ϵ represent the model size, latency, and energy upper bounds, and π denotes the throughput lower bound. The optimization problem can be addressed by solving its Lagrangian dual problem, where large Lagrangian multipliers are assigned to various constraints. In practical applications, one or more of the said constraints may be relaxed. The focus of this work is on the model size and latency constraints.

We employ a hardware architecture similar to that of [23] and extend the idea of HiKonv [24], which introduces packed operations for 1D convolutions, to support 2D convolutions with arbitrary bit-widths. More specifically, our operand and operation packing approach is capable of performing two multiplications for eight- or six-bit operands, six multiplications and two additions for four- or three-bit operands, and 15 multiplications and eight additions for two-bit operands, all while only using a single DSP. Fig. 2 illustrates an example of the arrangement of four-bit activations and weights of a convolutional layer in a single DSP in two different cycles. In each cycle, six multiplications are preformed, four of which are consumed in the same cycle and two are used in the next cycle. Note that the first cycle is a special case where the result in the ten lowest significant bits, $a_0 w_2$, is invalid and disregarded.

Due to our packing, weight quantization yields linear weight size reduction as a function of the bit-width selected while latency reduction is a function of the number of operations that can be packed together. Considering the total number of layers and the number of weights/operations per layer, the overall model size reduction and speedup can be calculated:

$$\eta = \frac{\sum_{l=1} n_l \times b_l}{\sum_{l=1} n_l^b \times 16}, \quad \kappa = \frac{\sum_{l=1} n_l \times \rho(b_l)}{\sum_{l=1} n_l^b \times \rho(16)},$$

where η is the compression ratio, n_l and n_l^b denote the number of parameters of layer l in the compressed and baseline models (16-bit and uncompressed), b_l is the proposed bit-width for layer l, κ is the latency speedup, and ρ returns the packing factor which indicates how many operations we can pack and perform simultaneously using the given bit-width.

IV. Results & Discussion

This section presents the results of experiments that evaluate the effectiveness of our methodology, which involve hyperparameter tuning in addition to MPQ and layer-width scaling through neural architecture search.

A. Convergence of K-Means TPE

This section presents the speedup in convergence for three types of models and three datasets. The first experiment involves applying random forest regression to the Iris dataset. The variables that define the search space are the number of trees in the forest, the maximum depth of each tree, and the minimum number of samples required to split an internal node.

The second experiment involves training a gradient boosting classifier on the Titanic dataset. The variables that define the search space are the learning rate, number of boosting stages, maximum depth of the estimator, minimum number of samples required to split, minimum number of samples needed to be at a leaf node, and number of features to consider when looking for the best split. For the first two experiments, $n_0 = 20$, $n = 100$, $k = 4$, and $\alpha = 0.98$.

Finally, the third experiment involves MPQ and layer-width scaling of a ResNet-18 model on the CIFAR-100 dataset. For

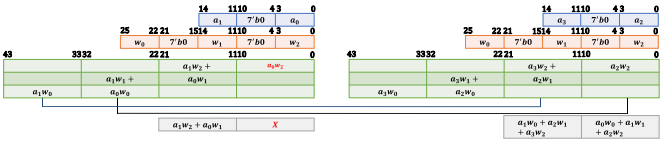

Fig. 2. Four-bit operand and operation packing. The design yields the computations required for two rows of convolutional kernels every two cycles.

this experiment, $n_0 = 40$, $n = 160$, $k = 4$, and $\alpha = 0.98$. Fig. 3 showcases the superior convergence speed of the presented k-means TPE over the traditional TPE for all experiments. It is observed that k-means TPE converges to superior or same-quality results at about two to three times fewer evaluations of proposed configurations compared to TPE.

B. Mixed-Precision Quantization and Layer-Width Scaling

We integrate our k-means TPE into the HyperOpt library [20], and use PyTorch to train and quantize the models. For all experiments, similar to [17], we only use a very small number of epochs to evaluate different configurations during the search process. More specifically, for models trained on the CIFAR datasets, the number of epochs to train during the search is set to four, and for the ImageNet dataset, this number is set to one. As shown in Table I for an exemplary model and dataset, regardless of whether different configurations are trained for 4 or 90 epochs during the search, the top-performing configurations in each case reach the same level of accuracy after they are fully trained for 90 epochs. After finding the best configuration, we train the models for 90 epochs using the Adam optimizer [25] with the weight decay set to 10^{-4}. We use the OneCycleLR learning rate schedule with a maximum learning rate of 0.01. All experiments were run on the Nvidia A6000 GPUs.

Table II compares the accuracy, model size, and latency speedup of different DNNs trained on different datasets and quantized using different approaches.

1) ImageNet: **ResNet-18 [1]** The ResNet-18 model trained with the bit-widths and layer-widths returned by the proposed search process achieves 70.8% validation accuracy with the model size reduced to 4.01MB and the end-to-end latency improved by a factor of 10.9 compared to a 16-bit fixed point (FiP-16) baseline. The trained model outperforms all prior work in terms of accuracy, compression, or both. Compared to the best prior work results for the ResNet-18 model [12], our ResNet-18 model achieves 1.1% higher accuracy with at least 9% smaller model size.

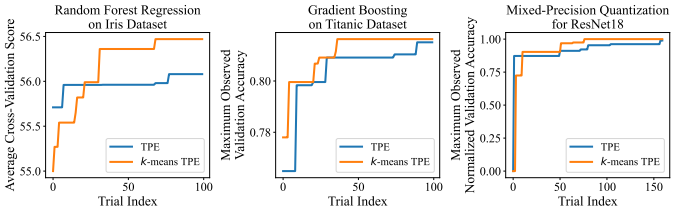

Fig. 3. Comparison of the convergence speed of TPE and k-means TPE for different ML algorithms and on Iris, Titanic, and CIFAR-100 datasets.

TABLE I
IMPACT OF THE NUMBER OF EPOCHS USED TO TRAIN EACH CONFIGURATION DURING THE SEARCH ON THE ACCURACY, MODEL SIZE, AND SPEEDUP OF THE BEST CONFIGURATION AFTER FULL TRAINING – SPEEDUP IS REPORTED COMPARED TO FIP-16.

Dataset	Architecture	Approach (Epochs per Config.)	Accuracy (%)	Model Size (MB)	Speedup
CIFAR-10	**ResNet-20**	90	91.94	0.097	10.86×
		4	91.90	0.088	11.14×

TABLE II
COMPARISON OF THE ACCURACY AND MODEL SIZE FOR DIFFERENT DATASETS AND DNN ARCHITECTURES.

b-MP denotes mixed-precision quantization with a minimum bit-width of *b*. *NR* stands for "Not Reported" by the authors.

Dataset	Architecture	Approach (W/A)	Accuracy (%)	Model Size (MB)	Speedup
ImageNet	**ResNet-18**	Baseline (FiP16/FiP16)	71.0	23.38	1.00×
		PACT [6] (3/3)	68.1	4.38	*NR*
		AutoQ [16] (4MP/4MP)	68.2	5.80	*NR*
		[12] (3MP/3MP)	69.7	4.38	*NR*
		EvoQ [13] (*NR*)	68.5	5.85	*NR*
		Ours (2MP/2MP)	**70.8**	**4.01**	10.90×
	MobileNetV2	Baseline (FiP16/FiP16)	72.6	6.8	1.00×
		PACT [6] (4/4)	61.4	*NR*	5.09×
		HAQ [14] (4MP/4MP)	67.0	*NR*	5.13×
		[26] (4/4)	71.2	1.79	*NR*
		EMQ [27]	71.0	**1.45**	*NR*
		EvoQ [13] (*NR*)	68.9	1.78	*NR*
		Ours (2MP/2MP)	**72.0**	1.50	7.74×
	ResNet-50	Baseline (FiP16/FiP16)	77.3	51.3	1.00×
		PACT [6] (3/3)	75.3	9.17	*NR*
		[26] (3/3)	75.1	9.36	*NR*
		HAQ [14] (3MP/3MP)	75.3	9.22	*NR*
		HAWQ [28] (*NR*)	75.5	7.96	*NR*
		HAWQ-V2 [11] (*NR*)	75.8	7.99	*NR*
		EMQ [27] (*NR*)	76.0	8.26	*NR*
		EvoQ [13] (*NR*)	75.5	12.81	*NR*
		[12] (3MP/4MP)	76.9	7.97	*NR*
		Ours (2MP/2MP)	76.7	**7.15**	13.56×
CIFAR-100	**ResNet-18**	Baseline (FiP16/FiP16)	76.1	23.38	1.00×
		Ours (2MP/2MP)	76.1	2.09	22.55×
	MobileNetv1	Baseline (FiP16/FiP16)	65.5	8.4	1.00×
		Ours (2MP/2MP)	66.09	1.66	11.09×
CIFAR-10	**ResNet-20**	Baseline (FiP16/FiP16)	91.5	0.54	1.00×
		ReLeQ [15]	91.38	0.101	*NR*
		Ours	**91.9**	**0.088**	11.14×

MobileNetV2 [29] The k-means TPE can compress the MobileNetV2 model to only 1.5 MB while having a mere 0.6% accuracy drop compared to a baseline floating point model. Our optimized MobileNetV2 model has only 3% higher model size compared to EMQ [27] while achieving 1% higher accuracy on the difficult ImageNet dataset.

ResNet-50 [1] Our optimized ResNet-50 model has a 7.15 MB model size while causing a 0.6% accuracy degradation. To the best of our knowledge, our method is the first to compress the network to this level with an acceptable accuracy drop. For example, the work in [12] achieves a similar accuracy drop but obtains a model with 11.4% larger model size.

2) CIFAR-100: Using the k-means TPE, we successfully compressed ResNet-18 and MobileNetV1 on CIFAR-100 datasets by factors of 11.18× and 5.06×, respectively, without any loss in accuracy. The inference latency experienced a significant improvement, as reported in Table II.

3) CIFAR-10: For the CIFAR-10 dataset, we conducted experiments with ResNet-20 and compared our results to those of ReLeQ [15]. As reported in Table II, the k-means TPE outperformed ReLeQ in terms of compressing to a very small size while preserving the output accuracy.

TABLE III
CONFIGURATIONS RETURNED BY k-MEANS TPE FOR REPRESENTATIVE DNN ARCHITECTURES

Model	Dataset	Configuration
ResNet-18	ImageNet	8, 6, 6, 4, 4, 6, 6, 4, 4, 4, 4, 2, 2, 3, 3, 2, 2, 6 1.25, 1.25, 1.25, 1.25, 1.25, 0.875, 0.875, 0.875, 0.875, 1, 1, 1, 1, 1, 1, 1, 1, 1
ResNet-20	CIFAR-10	8, 3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 3, 3, 2, 2, 3, 3, 8 1, 1, 1, 1, 1, 1, 1, 1, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75
MobileNetV1	CIFAR-100	8, 8, 8, 6, 6, 4, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 4, 4, 3, 3, 4, 4, 3, 3, 2, 2, 2 1, 1.25, 1.25, 0.875, 0.875, 0.875, 0.875, 1.125, 1.125, 0.875, 0.875, 1.25, 1.25, 1, 1, 1.125, 1.125, 1.25, 1.25, 1.25, 1.25, 1.125, 1.125, 0.75, 0.75, 1, 1, 0.875

TABLE IV
COMPARISON WITH BOMP-NAS

Dataset	Approach	Accuracy (%)	Model Size (MB)	Speedup	Search Cost (GPU-hour)
CIFAR-10	BOMP-NAS [19]	**88.67**	0.076	-	12.00
	Our ResNet-20	88.60	**0.052**	18.8×	**1.30**
CIFAR-100	BOMP-NAS [19]	75.84	4.199	-	30.00
	Our ResNet-18	**76.10**	**2.090**	**8.5**×	**2.05**

We also conducted a comparison with BOMP-NAS [19], which achieved SOTA results in terms of GPU hours of search and compression level. As shown in Table IV, our ResNet-20 model is found by expending 9.23× less search time while achieving nearly the same level of accuracy and being 31.5% smaller in size compared to BOMP-NAS. Similar gains can be observed for our ResNet-18 model. These results highlight the significant advantages of the proposed k-means TPE approach in terms of search efficiency and model compression.

Table III shows configurations found by the k-means TPE. For each model and dataset, the first and second rows of the configurations report the assigned bit-width and layer-width scaling factor for each layer, respectively.

V. CONCLUSIONS

We presented a search-based approach including a Hessian-based pruner and a tree-structured dual-threshold Parzen estimator for automatic optimization of DNNs bit-width and layer-width configurations to enable their efficient deployment. Through extensive experiments on benchmark datasets, we showed a 12× average search time speedup and a 20% reduction in model size compared to the SOTA compression techniques while preserving the model's output accuracy.

ACKNOWLEDGEMENTS

This research was sponsored in part by a grant from the National Science Foundation.

REFERENCES

[1] K. He *et al.*, "Deep residual learning for image recognition," in *Conf. on Comp. Vision and Pattern Recognition*, 2016.
[2] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *British Machine Vision Conf.*, 2016.
[3] I. O. Tolstikhin *et al.*, "MLP-Mixer: An all-MLP architecture for vision," in *Advances in Neural Information Processing Systems*, 2021.
[4] A. Vaswani *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017.
[5] J. Devlin *et al.*, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019.
[6] J. Choi *et al.*, "PACT: parameterized clipping activation for quantized neural networks," *CoRR*, vol. abs/1805.06085, 2018.
[7] S. Zhou *et al.*, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *CoRR*, vol. abs/1606.06160, 2016.
[8] H. Li *et al.*, "Pruning filters for efficient convnets," in *Int'l Conf. on Learning Representations*, 2017.
[9] J. Luo *et al.*, "ThiNet: A filter level pruning method for deep neural network compression," in *Int'l Conf. on Comp. Vision*, 2017.
[10] J. Bergstra *et al.*, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, 2011.
[11] Z. Dong *et al.*, "HAWQ-V2: Hessian aware trace-weighted quantization of neural networks," in *Advances in Neural Information Processing Systems*, 2020.
[12] C. Tang *et al.*, "Mixed-precision neural network quantization via learned layer-wise importance," in *European Conf. on Comp. Vision*, 2022.
[13] Y. Yuan *et al.*, "EvoQ: Mixed precision quantization of DNNs via sensitivity guided evolutionary search," in *Int'l Joint Conf. on Neural Networks*, 2020.
[14] K. Wang *et al.*, "HAQ: hardware-aware automated quantization with mixed precision," in *Conf. on Comp. Vision and Pattern Recog.*, 2019.
[15] A. T. Elthakeb *et al.*, "ReLeQ : A reinforcement learning approach for automatic deep quantization of neural networks," *IEEE Micro*, 2020.
[16] Q. Lou *et al.*, "AutoQ: Automated kernel-wise neural network quantization," in *Int'l Conf. on Learning Representations*, 2020.
[17] H. Liu *et al.*, "DARTS: differentiable architecture search," in *Int'l Conf. on Learning Representations*, 2019.
[18] Z. Cai and N. Vasconcelos, "Rethinking differentiable search for mixed-precision neural networks," in *Conf. on Comp. Vision and Pattern Recognition*, 2020.
[19] D. van Son *et al.*, "BOMP-NAS: Bayesian optimization mixed precision NAS," in *Design, Automation & Test in Europe Conf. & Exhibition*, 2023.
[20] J. Bergstra *et al.*, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Int'l Conf. on Machine Learning*, 2013.
[21] C. Baldassi *et al.*, "Shaping the learning landscape in neural networks around wide flat minima," *Proc. Natl. Acad. Sci. USA*, 2020.
[22] M. F. Hutchinson, "A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines," *Communications in Statistics-Simulation and Computation*, 1989.
[23] S. Basalama *et al.*, "FlexCNN: An end-to-end framework for composing CNN accelerators on FPGA," *ACM Trans. Reconfig. Technol. Syst.*, 2023.
[24] X. Liu *et al.*, "HiKonv: High throughput quantized convolution with novel bit-wise management and computation," in *Asia and South Pacific Design Automation Conf.*, 2022.
[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Int'l Conf. on Learning Representations*, 2015.
[26] S. Han *et al.*, "Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding," in *Int'l Conf. on Learning Representations*, 2016.
[27] Z. Liu *et al.*, "Evolutionary quantization of neural networks with mixed-precision," in *Int'l Conf. on Acoustics, Speech and Signal Processing*, 2021.
[28] Z. Dong *et al.*, "HAWQ: Hessian aware quantization of neural networks with mixed-precision," in *Int'l Conf. on Comp. Vision*, 2019.
[29] M. Sandler *et al.*, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Conf. on Comp. Vision and Pattern Recognition*, 2018.