IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 6, 2021, accepted September 20, 2021, date of publication September 29, 2021,
date of current version October 7, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3116418

Network Compression via Mixed Precision Quantization Using a Multi-Layer Perceptron for the Bit-Width Allocation

EFSTATHIA SOUFLERI, (Graduate Student Member, IEEE), AND KAUSHIK ROY, (Fellow, IEEE)
School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907, USA

Corresponding author: Efstathia Soufleri (esoufler@purdue.edu)

This work was supported in part by the Center for Brain Inspired Computing (C-BRIC), one of the six centers in Joint University Microelectronics Program (JUMP), in part by the Semiconductor Research Corporation (SRC) Program sponsored by Defense Advanced Research Projects Agency (DARPA), in part by the Semiconductor Research Corporation, in part by the National Science Foundation, in part by the Intel Corporation, in part by the Department of Defense (DoD) Vannevar Bush Fellowship, in part by the U.S. Army Research Laboratory, and in part by the U.K. Ministry of Defence under Agreement W911NF-16-3-0001.

ABSTRACT Deep Neural Networks (DNNs) are a powerful tool for solving complex tasks in many application domains. The high performance of DNNs demands significant computational resources, which might not always be available. Network quantization with mixed-precision across the layers can alleviate this high demand. However, determining layer-wise optimal bit-widths is non-trivial, as the search space is exponential. This article proposes a novel technique for allocating layer-wise bit-widths for a DNN using a multi-layer perceptron (MLP). The Kullback-Leibler (KL) divergence of the softmax outputs between the quantized and full precision network is used as the metric to quantify the quantization quality. We explore the relationship between the KL-divergence and the network size, and from our experiments observe that more aggressive quantization leads to higher divergence, and vice versa. The MLP is trained with layer-wise bit-widths as labels and their corresponding KL-divergence as the input. The MLP training set, i.e. the pairs of the layer-wise bit-widths and their corresponding KL-divergence, is collected using a Monte Carlo sampling of the exponential search space. We introduce a penalty term in the loss to ensure that the MLP learns to predict bit-widths resulting in the smallest network size. We show that the layer-wise bit-width predictions from the trained MLP result in reduced network size without degrading accuracy while achieving better or comparable results with SOTA work but with less computational overhead. Our method achieves up to 6x, 4x, 4x compression on VGG16, ResNet50, and GoogLeNet respectively, with no accuracy drop compared to the original full precision pretrained model, on the ImageNet dataset.

INDEX TERMS Deep learning, efficient deep learning, neural networks, network compression, quantization, mixed-precision, multi-layer perceptron.

I. INTRODUCTION

Deep Neural Networks are state-of-the-art solutions for solving a plethora of complex tasks, varying from image classification, object detection, and voice recognition tasks to communication and networking applications [1], [2]. These networks require tremendous computational resources, which might not always be available in resource-constrained devices, to achieve competitive performance on these tasks. A promising solution is the quantization of the weights of the network. This process trains or converts a full precision network into limited precision while trying to maintain performance.

There exist a great number of works in literature that train a neural network from scratch with limited precision and achieve high compression rates [3]–[5]. However, training in discrete space is challenging, and the convergence is slow [6]. To address this issue, emerging works convert a full precision network into its quantized version [6], [7]. Such techniques are commonly referred to as post-training quantization. Selecting the precision for each layer is a non-trivial problem as the search space is exponential in size. For a network with L layers and m possible bit-width choices for each layer, we have m^L possible choices.

The associate editor coordinating the review of this manuscript and approving it for publication was Thomas Canhao Xu.

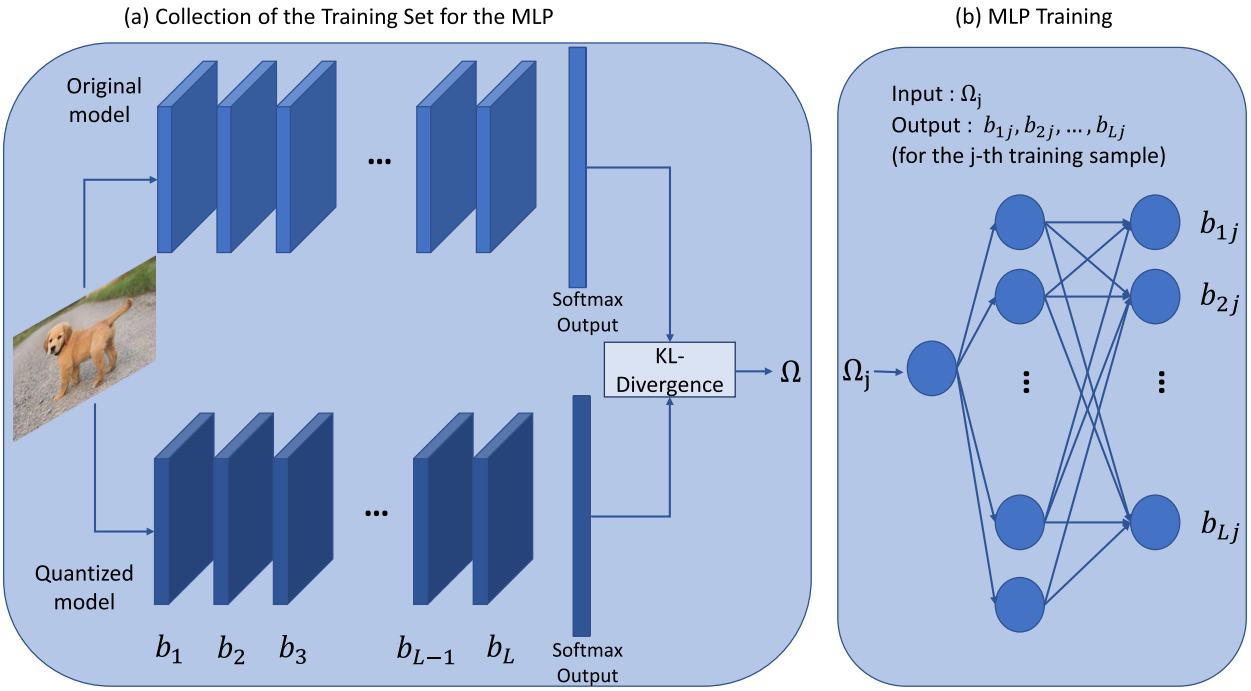

FIGURE 1. Illustration of the proposed framework: (a) the collection of the training set for the MLP. Each layer of the network is quantized with different precision b_i, $i = 1, .., L$, where L is the number of layers in the network. The KL-divergence Ω between the softmax output of the original full precision and the quantized model is computed. This process is repeated S times to collect the entire dataset for the MLP. (b) The MLP is trained using the collected dataset. The input to the MLP is the KL-divergence Ω_j and the output is the bit-width across the layers of the network, for each of the training samples j, j = 1,…,S. The MLP has L output neurons, which is equal to the number of layers of the network intended to be quantized.

Many prior works that avoid the exponential search problem have suggested heuristics to allocate the mixed-precision bit-widths or using a proxy to formulate a solvable optimization problem [8], which adds extra computational overhead. Such methods deploy Bayesian optimization [9], evolutionary search algorithms, [10], [11] calculation of the layer-wise quantization error [8] and adversarial noise computation [7]. We provide more details for the methods mentioned above in section II. Our work, presented in this article, belongs in the post-training quantization category that uses different bit precision across the network's layers and reduces the computational overhead by using a multi-layer perceptron (MLP) model for determining the mixed-precision bit-widths.

In this article, we study the impact of weight quantization on the performance of neural networks. We observe that more aggressive quantization leads to higher divergence between the output of the full precision and the quantized network, and vice versa. Based on this observation, we propose a novel approach to mixed-precision bit-width allocation using a Multi-Layer-Perceptron (MLP) model trained using the Kullback-Leibler (KL) divergence between the softmax output of the full precision and the quantized network. The KL-divergence is used as a metric to determine the performance divergence between the two networks. The network intended to be compressed is quantized with multiple mixed-precision bit-width configurations determined using Monte Carlo sampling of the search space. For each sample, the KL-divergence at the output is computed. Pairs of the bit-widths (vector) and the KL-divergence are collected, and they constitute the training set for the MLP. The MLP is trained to predict the bit-width configuration (output), given as input the desired KL-divergence. Since multiple bit-width configurations exist for a given KL-divergence, we introduce a network size penalty to the MLP loss to ensure the network learns to predict configurations leading to the smallest network size. Figure 1 illustrates the proposed framework. We evaluate our framework on the ImageNet dataset [12], which is a highly complex dataset for image classification problems, using three different deep neural network architectures, namely VGG16 [13], ResNet50 [14] and GoogLeNet [15]. The experimental results indicate that our method achieves 8x network compression with at most 0.7% accuracy drop across all the evaluated network architectures.

The rest of the article is organized as follows: the related works are summarized in Section II. Section III presents the impact of quantization on the network and describes the implementation details of the proposed framework: training set collection for the MLP, MLP training, and bit-width allocation across the layers of the network. Section IV reports the experimental evaluation of the proposed method, compares it with state-of-the-art works and analyzes the computational overhead. Finally, section V concludes the article and describes future work.

II. RELATED WORK

In the domain of network quantization, there exist two main approaches [16]: Quantization-Aware-Training (QAT) and Post-Training Quantization (PTQ). QAT refer to the case of

either training a network from scratch with limited precision or fine-tuning for a few epochs with limited precision after network quantization. PTQ refers to the case of quantizing a pretrained full precision network without retraining it. Note, that methods included in PTQ category can be data-free or may require a small calibration set, which is readily available [16]. It is worth mentioning that, both QAT and PTQ categories can include methods that allocate mixed-precision weights across the layers of the network. As far as QAT is concerned, there is a plethora of works that trains highly compressed networks [5], [9], [17]–[20]. In [20], the authors model the quantization problem as a discrete constrained optimization problem which is solved using the Alternating Direction Method of Multipliers (ADMM) during training. Reference [19] proposes incremental weight partitioning into two groups, group-wise quantization and retraining. Authors in [18] introduce a quantization method to reduce this loss by learning a symmetric code-book for particular weight subgroups. Furthermore, [17] proposes a quantization scheme that allows inference to be carried out using integer-only arithmetic. In [5] the authors suggest a method that uses low bit-width gradients along with quantized weights and activations during training. Authors in [9] start with a pretrained full precision network, deploy Bayesian optimization to allocate the bit precision across the layers of the network, and then fine-tune the network with limited precision for a few epochs. An example of extremely compressed networks are the binary and ternary networks [3], [4], [21]–[23]. However, as mentioned earlier, QAT leads to slow network convergence, and optimization is more difficult in the discrete space [6]. Moreover, the fine-tuning step after quantization can be impractical in many real scenarios, where there is no time to retrain the network after quantization, as the network needs to be deployed immediately [24]. For example, in online learning the network needs to be trained on new data and then deployed immediately.

The second approach (PTQ), including our work, has emerged to address the above challenges. Authors in [6], [25] suggest quantizing the network using equal bit-width across all the layers by using the signal-to-quantization-noise-ratio (SQNR) and the floating and fixed point error probability, respectively. This approach avoids searching the exponential solution space, however, results in sub-optimal bit-width allocation as each layer might have a different impact on the network's performance. Thus, researchers have suggested using mixed-precision quantization across the layers of the network. These works commonly propose using a proxy to formulate a solvable optimization problem for bit-width allocation. In [10], [11], the authors use an evolutionary-algorithm based method to allocate the bit-widths. Authors in [8] formulate an optimization problem based on layer-wise quantization errors and they solve it using Lagrangian multipliers. The work in [7] suggests the use of adversarial noise to formulate the optimization problem which is solved by using the Karush-Kuhn-Tucker (KKT) conditions. Further, research has emerged [26] that specializes the quantization policy for the different hardware architectures. Different from [26], our work makes no assumption about the underlying hardware. Note, that our proposal belongs to the data-free PTQ techniques with mixed-precision allocation across the network's layers. Table 1 lists and summarizes the different methods discussed in this section.

TABLE 1. Summary of the related work methods.

	Method's Name	Method's Main Distinctive Characteristic
QAT	TBSQ[20]	Alternating Direction Method of Multipliers
	INQ[19]	Weight Partition and Group-wise Quantization
	SYQ[18]	Learning of a Symmetric Code-book
	IAOI[17]	Integer-only Arithmetic
	DoReFa[5]	Low Bit-width Gradients during training
	CLIP-Q[9]	Bayesian Optimization
	BWN[3]	Binary Network
	BC[4]	Binary Network
	BNN[23]	Binary Network
	TWN[21]	Ternary Network
	FGQ[22]	Ternary Network
PTQ	AGNP[6]	Floating and Fixed Point Error Probability
	FPQ[25]	SQNR
	EMQ[10]	Evolutionary Search Algorithm
	EvoQ[11]	Evolutionary Search Algorithm
	OBA[8]	Lagrange Multipliers
	AQ[7]	KKT conditions
	HAQ[26]	Hardware-based Quantization Policy

Related to network quantization, in the domain of network compression, there are techniques like pruning, that can be applied after or before quantization to compress the network further. These techniques are orthogonal to our proposal and the other post-training techniques, and can be implemented on top of them for further network compression. For instance, Han *et al.* [27] suggest pruning and Huffman coding after quantization. Generally, pruning works [28]–[31] propose the removal of network weights based on some metric (for example the absolute values of the weights, etc.). In this way, the size of the network can be reduced and the network can be more easily deployed in resource-constrained scenarios.

III. MULTI-LAYER PERCEPTRON FOR LAYER-WISE BIT-WIDTH ALLOCATION

A. CHALLENGES OF QUANTIZATION

The bit-width allocation across the layers of the network affects its performance and size. When a quantized network is presented with an input it results in errors or deviations from a full precision network. This is a result of errors from quantization accumulating at each layer and reflecting it at the output of the network. Different bit-width combinations lead to different impacts on performance and size. Since the search space of all the possible combinations of bit-widths is exponential, finding the optimal solution is non-trivial. The problem is further complicated by the fact that two different bit-width combinations may result in similar deviations (many to one mapping) in network performance.

Addressing the challenge of quantifying deviations can be done in two ways either by combining layer-wise errors into a single metric or by using a cumulative error metric at the network output. We capture the cumulative impact of the different bit-width combinations on the network's output through our proposed method. Further, the proposed method accounts for the many to one problem. If two bit-width combinations have the same impact on the network's performance, our method selects the bit-width with the smallest network size, and it is verified experimentally in section IV-C.

B. MULTI-LAYER PERCEPTRON

In this section, we present a novel technique to allocate a layer-wise mixed-precision bit-width for quantizing a deep neural network. The proposed method uses a Multi-Layer-Perceptron (MLP) where the input to the MLP is the KL-divergence between the softmax output of the full precision and the quantized network and the output is the bit-width configuration for the compressed network. Our methodology consists of three main steps: sampling the search space to create a custom training set for the MLP, MLP model training, and prediction of the bit-width configuration for the compressed network.

1) NOTATION

Let M be a full precision L layered deep neural net, let θ symbolize the parameters of a trained full precision network, and $\tilde{\theta}$ the parameters of a quantized network. The input (image) to the network M is symbolized as x.

2) TRAINING SET COLLECTION FOR THE MLP

In this section, we describe how the training set for the MLP is collected. Let S be the size of the training set T for the MLP. Each sample in the training set T is a tuple of input and label. The label is a bit-width configuration B_j and the input is the corresponding KL-divergence Ω_j between the full precision network and quantized network whose bit-width configuration is B_j, where j denotes in j^{th} sample in T.

The bit-width configuration $B_j = (b_{ij})$ is a vector of size L where $i \in \{1, .., L\}$, $j \in \{1, .., S\}$ and L is the number of layers of the network M. Each element $b_{ij} \in B_j$ represents the bit-width for the corresponding layer of the network M. The bit-width configuration B_j is sampled from the search space using a Monte Carlo sampling with an additional constraint of $b_{ij} \in [b_{min}, b_{max}]$. If b_{ij} is extremely low i.e. $b_{ij} < b_{min}$, this leads to high KL-divergence for most samples in T. Similarly, high b_{ij} i.e. $b_{ij} > b_{max}$ results in negligible KL-divergence. Therefore, b_{min} and b_{max} are parameters that are determined experimentally for each network. This is done by quantizing all the layers with the same bit-width and evaluating the inference accuracy. The values of b_{min} and b_{max} are chosen such that b_{min} is the maximum bit-width that leads to an accuracy drop 50% or greater when compared to the full precision network and b_{max} is chosen such that it is the minimum bit-width that maintains the accuracy of the full precision network. Note, that a straightforward selection for b_{max} could be the precision of the original pretrained model (i.e. $b_{max} = 32$ bits). However, the network can be quantized to 16 bits or sometimes lower bit-widths, depending on the model, without accuracy degradation. Thus, we select b_{max} as mentioned previously, in order to avoid quantizing the network with more bits than needed and at the same time achieving smaller model size. Furthermore, using $b_{max} < 32$ bits reduces the search space during the Monte Carlo sampling. It is worth mentioning that the above process of using the same bit-width across all the layers happens once to obtain the b_{min} and b_{max}, and eventually each layer of the network M is quantized with different bit-widths (mixed-precision bit-width allocation).

For each sampled bit-width configuration $B_j = (b_{ij})$ with $j \in \{1, 2, .., S\}$ and $i \in \{1, .., L\}$, we quantize the full precision network M with the B_j and the KL-divergence Ω_j at the output is computed. This process results in a training set T with S samples of the form $T = \{(\Omega_1, B_1), (\Omega_2, B_2), .., (\Omega_S, B_S)\}$, which is used to train the MLP. Similarly, we collect the testing set for the MLP model.

We use the KL-divergence between the full precision and the quantized network softmax output as a metric to quantify the divergence between the two networks. The KL-divergence (Ω) is calculated as the average of N images from the training set:

$$\Omega = \frac{1}{N} \sum_{i=1}^{N} KL(M(\theta, x_i), M(\tilde{\theta}, x_i)) \tag{1}$$

Note, that we are using the softmax of the model's output in order to obtain the probability distribution of predicting each class. This probability vector is used for the KL-divergence computation. By definition, the KL-divergence is a measure of how one probability distribution is different from a second reference probability distribution [32].

3) MLP TRAINING PROCESS

The MLP training is similar to the training of a typical supervised learning task that seeks to minimize the empirical risk:

$$\min_{\theta} \mathcal{L}(\theta) = \frac{1}{S} \sum_{j=1}^{S} f(MLP(\theta; \Omega_j), B_j) \tag{2}$$

where $f(\cdot, \cdot)$ is the loss function (typically mean squared error or cross-entropy loss) and S is the number of samples in the training set.

However, the standard empirical risk described in Equation 2 does not account for the many to one mapping problem described in subsection III-A. The many to one mapping problem, i.e. the problem when two different bit-width configurations result in the same KL divergence, can be accounted for by ensuring that the MLP learns the configuration that results in smallest network size. To ensure that the MLP learns the smallest network size, we need to introduce a penalty parameter in Equation 2. The modified empirical risk of the MLP includes the size of the quantized

network in the loss and it is given by Equation 3.

$$\min_{\theta} \mathcal{L}(\theta) = \frac{1}{S}\sum_{j=1}^{S} f(MLP(\theta; \Omega_j), B_j) + \frac{\beta}{S}\sum_{j=1}^{S}\sum_{i=1}^{L} b_{ij} * p_i \quad (3)$$

where $f(\cdot, \cdot)$ is the loss function (typically mean squared error or cross-entropy loss), S is the number of samples in the training set, β is a scalar and p_i is the number of parameters of the i^{th} layer of the network M. The product $b_{ij} * p_i$ corresponds to the size of the i^{th} layer. The trained MLP predicts the different bit-widths across the layers of the M network (MLP output), given the desired KL-divergence (MLP input). The desired KL-divergence is user-defined. Algorithm 1 summarizes the steps for the bit-width allocation across the layers of a network M.

Algorithm 1: Bit-Width Allocation

Input: Full Precision network M with L layers, desired KL-divergence

Output: Bit-width

1 Sample Collection:

for $j \leftarrow 1$ **to** S **do**

  Generate the random bit-width B_j

  Quantize M with B_j

  Compute the KL-divergence Ω_j for N training images

  Store the pair (Ω_j, B_j)

end

2 Train MLP with the S samples

3 Feed the desired KL-divergence to the MLP and obtain the bit-width configuration prediction $B = (b_1, \ldots, b_L)$ for M

IV. EXPERIMENTAL EVALUATION

A. BIT-WIDTH ALLOCATION

In this section, we deploy our framework to compress the VGG16 [13], the ResNet50 [14], and the GoogLeNet [15] network architectures trained on the ImageNet dataset [12]. The proposed compression framework uses an MLP which consists of two fully connected layers with 200 neurons in the hidden layer. The number of output neurons is equal to the number of layers of the network to be quantized. For example, for a VGG16 model, which has 16 layers, the MLP will have 16 output neurons. The MLP is trained using empirical risk minimization given by Equation 3 and we use mean squared error as the loss function f. For all the models, we use an MLP training set of size $S = 1200$ to train the MLP, by quantizing the model with Monte Carlo sampled bit-widths and computing the KL-divergences between the full precision and the quantized model. For the KL-divergence computation, we observe that $N = 50$ (refer to Equation 1) training images are a sufficient number that captures the KL-divergence between the full precision and quantized model. Note, that for the KL-divergence calculation we do not use the testing images but training images. The activations of the network are set to 8-bit precision for all the simulations. The proposed methodology is implemented using PyTorch [33].

B. QUANTIZATION SCHEME

In the quantization process, a key component is the choice of the quantization function. In essence, the quantization function takes real values represented in floating-point and maps them to a lower precision range. A popular choice for the quantization function is defined as follows [34]:

$$Q = \left\lfloor \frac{r}{s} \right\rfloor - z \quad (4)$$

where Q is the quantization operator, r is the real-valued input, s is the scale factor and z is the zero-point. The $\lfloor\ \rfloor$ operator is the floor operator that maps a real value to an integer through a rounding operation. This method of quantization is known as uniform quantization, as the resulting quantized values (quantization levels) are uniformly spaced. The scaling factor s divides a given range of real values r into a number of partitions and it is defined using the following formula:

$$s = \frac{\beta - \alpha}{2^b - 1} \quad (5)$$

where $[\alpha, \beta]$ denotes the clipping range, i.e. a bounded range that we are clipping the real values to, and b is the quantization bit-width. Thus, in order to determine the scaling factor, the clipping range $[\alpha, \beta]$ should be defined first. For the uniform asymmetric quantization method, the clipping range is not symmetric with respect to the origin. A popular choice for the clipping range is to use the minimum/maximum value of the signal ($\alpha = r_{min}$ and $\beta = r_{max}$). For the uniform symmetric quantization scheme, the clipping range should be symmetric to the origin and therefore $\alpha = -\beta$. A popular choice is based on the min/max values of the signal: $-\alpha = \beta = max(|r_{max}|, |r_{min}|)$. In this work, we use uniform symmetric weight quantization and uniform asymmetric activation quantization for our experiments. We use the ReLU activation function that always results in non-negative values. This causes an imbalance and therefore asymmetric is preferred over symmetric quantization for quantizing the activations. Furthermore, per-tensor quantization of weights and activations is applied, meaning that we use a single set of quantization parameters (quantizer) per tensor.

C. KL-DIVERGENCE

This section studies the relationship between the size of the network and the KL-divergence between the full precision and quantized network output. Figure 2 (a) is a plot of network size versus KL-divergence and visualizes a subset of the training set T that was used to train the MLP in order to make the bit-width configuration predictions for the VGG16 [13] network architecture. Each sample in the training set is depicted as a blue dot. The x-axis represents the resulting size of the VGG16 network when quantized with a

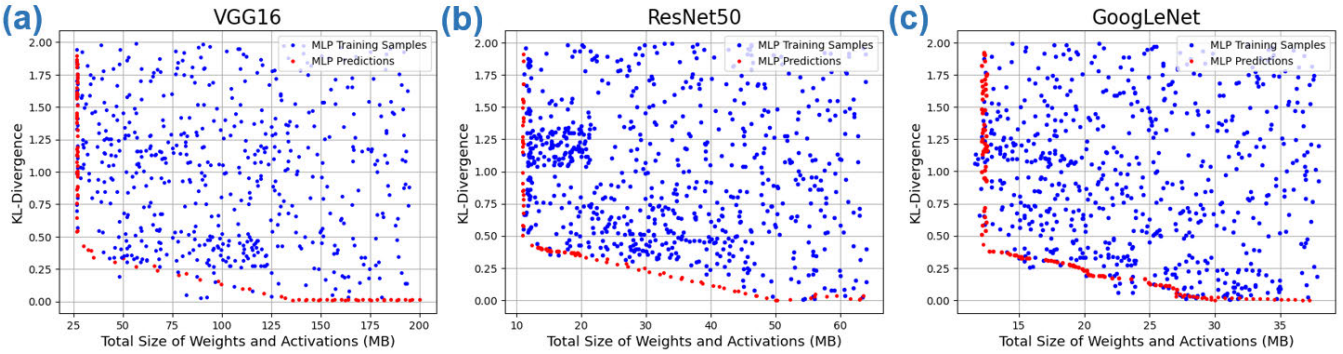

FIGURE 2. The KL-divergence versus the total size of weights and activations (MB) of: (a) VGG16 [13], (b) ResNet50 [14], and (c) GoogLeNet [14] on ImageNet [12]. The x-axis represents the resulting size of the GoogLeNet network when quantized with a bit-width configuration $B = (b_1, \ldots, b_L)$ from the training set T and the y-axis is the corresponding KL-divergence from T. The blue dots represent the samples from the MLP training set and the red dots represent the MLP's predictions.

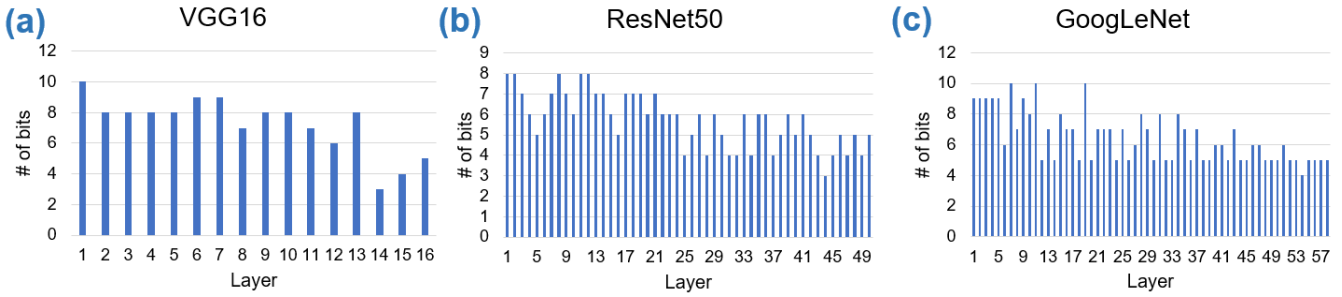

FIGURE 3. The bit-width allocation across the layers of: (a) VGG16 [13], (b) ResNet50 [14], and (c) GoogLeNet [14] for KL-divergence = 0.1.

bit-width configuration $B = (b_1, \ldots, b_L)$ from the training set T and the y-axis is the corresponding KL-divergence from T. We observe that two different bit-width configurations, consequently two different network sizes, may result in the same KL-divergence. This is because the order of the bit-widths b_{ij} in the bit-width configuration B plays a significant role, that is some layers of the network are more sensitive to quantization than others, which is leveraged with mixed-precision bit-width allocation.

The proposed training method with the aid of the added penalty described in Equation 3 forces the MLP to learn and predict the bit-width configuration that results in the smallest network size. We verify this by plotting the MLP predicted sizes in Figure 2 (a). The MLP predictions for various KL-divergence values are marked with red dots. The KL-divergence values are input to the MLP and the predicted bit-width configurations $B = (b_1, \ldots, b_L)$ is used to compute and plot the resulting network size on the x-axis. We observe from Figure 2 (a) that beyond a certain network size the KL-divergence value saturates near zero. This is expected because in this regime the network is not aggressively quantized and therefore its output does not diverge a lot compared to the original model.

In a similar manner to Figure 2 (a), we plot the training samples T with blue dots and the MLP predictions with red dots for ResNet50, and GoogLeNet architectures in Figures 2 (b) and 2 (c), respectively. We make similar observations for ResNet50 and GoogLeNet architectures, i.e. the MLP trained for the ResNet50 and GoogLeNet architectures learns the bit-width configuration leading to the smallest network size.

D. BIT-WIDTH ALLOCATION ANALYSIS

This section presents an analysis of the bit-width allocation across the network's layers obtained using our proposed methodology. Firstly, we select KL-divergence to be equal to 0.1 making the divergence between the full precision and the quantized network is negligible. Figure 3 (a) illustrates the bit-width allocation for VGG16 that is obtained from the MLP prediction when the input KL-divergence is equal to 0.1. Next, we repeat the above process, but we use KL-divergence equal to 1.5 (Figure 4 (a)). In this case, the output of the full precision network significantly divergences from the quantized network. From the bar graphs in Figures 3 (a) and 4 (a), we observe that the network is more aggressively quantized when KL-divergence is 1.5 compared to KL-divergence of 0.1. This result is in line with our expectation since aggressive quantization introduces more errors and thus the output of the full precision and the quantized model diverges more.

Moreover, we observe that most of the initial layers have higher bit-width than the later ones. Our finding

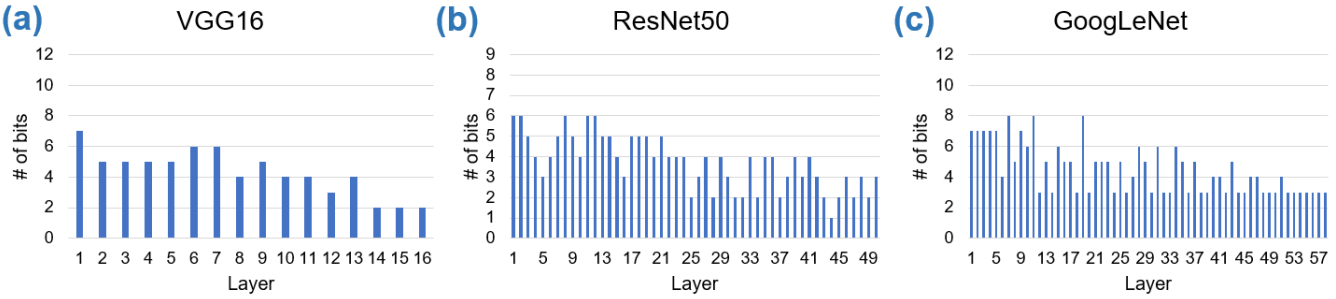

FIGURE 4. The bit-width allocation across the layers of: (a) VGG16 [13], (b) ResNet50 [14], and (c) GoogLeNet [14] for KL-divergence = 1.5.

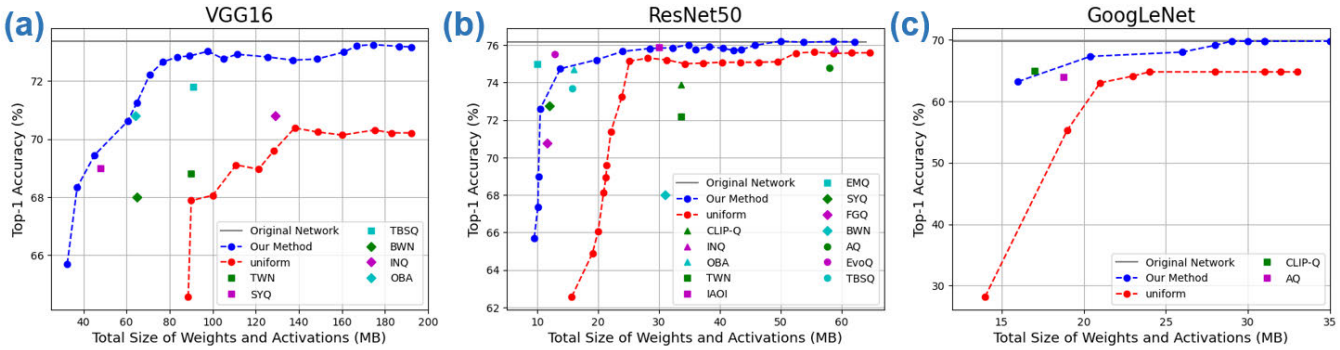

FIGURE 5. The results of state-of-the-art methods on: (a) VGG16 [13], (b) ResNet50 [14], and (c) GoogLeNet [14] over ImageNet [12] dataset.

is corroborated by previous work that states that trained networks are more sensitive to their initial layer weights [35]. For ResNet50 and GoogLeNet, we perform a similar analysis as VGG16. The resulting bit-width is depicted in Figures 3 (b), 4 (b), and 3 (c), 4 (c), respectively. Note, that similar observations as VGG16, also hold for ResNet50 and GoogLeNet.

E. COMPRESSION RESULTS AND SOTA COMPARISON

In this section we present the compression results that our method achieves for VGG16 [13], ResNet50 [14], and GoogLeNet [15] on ImageNet dataset [12] and we compare our results with state-of-the-art works. Top-1 inference accuracy is the metric used to quantify the performance of the network.

Our method compresses VGG16 [13] up to 6x when compared to the full precision model (32-bit precision weights and activations) with no accuracy drop (see Figure 5 (a)). On ResNet50 and GoogLeNet [14], our method achieves up to 4x compression compared to the full precision model (32-bit precision weights and activations) while maintaining the inference accuracy (see Figures 5 (b) and 5 (c)).

We compare the results from our experiments with the following state-of-the-art works, including both 1) QAT methods: Binarized Weight Network (BWN) [3], Ternary Weight Network (TWN) [21], Incremental Network Quantization (INQ) [19], Fine-grained Quantization (FGQ) [22], Two-bit Shift Quantization (TBSQ) [20], Integer Arithmetic-only Inference (IAOI) [17], Compression Learning by In-parallel Quantization (CLIP-Q) [9], Symmetric Quantization (SYQ) [18], and 2) PTQ methods: Adaptive Quantization (AQ) [7], Evolutionary quantization of neural networks with mixed-precision (EMQ) [10], Optimizing the Bit Allocation for Network Compression (OBA) [8] and Mixed Precision Quantization of DNNs via Sensitivity Guided Evolutionary Search (EvoQ) [11].

Our work demonstrates better performance than uniform bit-width allocation (visualized with red plot) on VGG16, ResNet50 and GoogLeNet, illustrated in Figures 5 (a), 5 (b), and 5 (c). This is because uniform quantization treats all the layers equally while mixed-precision bit-width allocation captures the impact that each layer has to the network's output more effectively and efficiently. Our method either outperforms or achieves comparable results to state-of-the-art works across different networks architectures, as illustrated in Figures 5 (a), 5 (b), and 5 (c). Our method performs better than all the SOTA works and has performance comparable with SYQ and OBA on VGG16, FGQ, SYQ, IAOI, INQ and OBA on ResNet50, and CLIP-Q on GoogLeNet. EvoQ and EMQ perform better than our method on ResNet50. However, both methods are based on evolutionary search algorithms that evaluate the fitness function multiple times, and also they require a calibration set to perform feature adjustments. This adds extra computational complexity, and we report the exact computational overhead numbers in Section IV-F. Note, that our method achieves high compression results with less

computational overhead than SOTA works, as it is analyzed in the next section.

F. ANALYSIS OF COMPUTATIONAL OVERHEAD

In this section, we will analyze the computational overhead of our method and we will compare it with other quantization methods that require retraining and/or use other forms of optimization. All the experiments were conducted on a system with a Nvidia GTX 1080ti GPU.

The computational overhead is reported in terms of effort factor (ρ). Effort factor is defined as the ratio of the number of FLOPs required to compute the bit-width allocation using a particular method to the number of FLOPs required for one training epoch over the entire training images of the dataset, in our case ImageNet. The effort factor (ρ) is given by Equation 6.

$$\rho = \frac{\text{\# of FLOPs of an allocation method}}{\text{\# of FLOPs (forward + backward pass)} * I} \tag{6}$$

where I is the number of images of the training set. For ImageNet dataset, the training set consists of 1.2 million images.

The number of FLOPs required for a training epoch is considered to be three times the number of FLOPs required for a forward-pass [36]. Note, that the number of FLOPs required to compute the bit-width allocation of a method, depends on two parameters: 1) if the method adds a few fine-tuning epochs at the end of bit-width allocation, 2) if the proxy that it used needs additional statistics, for example the layer-wise error over a few training images, to formulate the optimization problem.

The computational overhead of our method arises from the following: 1) the MLP training and, 2) the Monte Carlo sampling and creation of the custom dataset T. For all the experiments, we use a 2-layer MLP with 200 neurons in the hidden layer. As far as MLP training overhead is concerned, we observe that one training epoch requires a forward and backward pass of MLP over the samples of T. VGG16, ResNet50 and GoogLeNet have 16, 50 and 22 layers, respectively, and a training epoch requires a forward and backward pass over the 1.2 million training images of ImageNet. As the MLP size is significantly smaller than the size of VGG16/ResNet50/GoogLeNet and T is substantially smaller than the 1.2 million training images of ImageNet; the computational overhead for the MLP training is minimal compared to one training epoch of VGG16/ResNet50/GoogLeNet.

Regarding the computational overhead due to sample collection for the custom dataset T, we observe that this process requires $S * N$ forward passes on VGG16/ResNet50/ GoogLeNet, where S is the number of training samples of T and N is the number of training images used for the KL-divergence computation. No backward pass on VGG16/ResNet50/GoogLeNet is required. This computational overhead is significantly smaller than a training epoch for VGG16/ResNet50/GoogLeNet, because the number of forward passes ($S * N$) is smaller than the number of forward passes on the entire ImageNet, and no backward pass is required. Therefore, our method has minimal computational overhead ($\rho = 0.0016$x) compared to a training epoch of VGG16/ResNet50/ GoogLeNet on ImageNet.

TABLE 2. Comparison of the computational overhead of PTQ methods.

Method	Effort Factor (ρ)
OBA [8]	1x
EMQ [10]	0.3x
EvoQ [11]	0.0022x
This work	0.0016x

We compare our computational overhead with other PTQ work [9]–[11], [17], [20]. We do not compare the computational overhead with works belonging to QAT methods [3], [17]–[22], as all the PTQ work, including ours, starts with a pretrained full precision network. Thus, the QAT methods will have higher computation overhead and it is not a fair comparison. Table 2 summarizes the comparison results. We observe that among the PTQ methods, our work adds the lowest computation overhead. In Table 2, we have not included the comparison with [7]. This is because it was non-trivial to compute the FLOPs for this work. However, this method requires the computation of dataset-dependent terms whose calculations needs 6 hours for the ResNet50 network, as stated in their paper [7]. One training epoch for ResNet50 requires 55 minutes under the same underlying hardware, which translates to a $\rho = 6.54$x. Thus, [7] is more time-consuming and computationally intensive than one training epoch and therefore more computationally intense compared to our method as well.

V. CONCLUSION AND FUTURE WORK

In this article, we proposed a novel simple-yet-effective bit-width allocation method for network compression that uses an MLP model. We create a custom dataset consisting of pairs of bit-width configurations and KL-divergence to train an MLP model. For a desired KL-divergence, which is a proxy for network size, the MLP model predicts the bit-width configuration that the network should be quantized with. The proposed method has little computational overhead compared to other state-of-the-art techniques and achieves up to 6x, 4x, 4x compression on VGG16, ResNet50, and GoogLeNet respectively with no accuracy degradation compared to the original pretrained full precision network.

Today, there is a trend to move computation from the cloud to the edge. Sensor systems embedded in different devices are an example of such an application [37]–[39]. The edge devices, however, do not have the computational resources that are available on the cloud. To deploy a neural model on the edge it should be compressed to achieve energy efficiency. Our proposal can be used as a low-cost inference method for deploying neural models on edge devices. Moreover, note that in this article our proposal is evaluated on the image classification task. However, this could be applied to any model that performs a supervised task. Evaluating our proposed

methodology on other machine learning tasks would be an interesting direction for exploration in the future.

REFERENCES

[1] T. O'Shea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Trans. Cogn. Commun. Netw.*, vol. 3, no. 4, pp. 563–575, Dec. 2017.

[2] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 2, pp. 445–458, Feb. 2019.

[3] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 525–542.

[4] M. Courbariaux, Y. Bengio, and J. P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 3123–3131.

[5] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2016, *arXiv:1606.06160*. [Online]. Available: http://arxiv.org/abs/1606.06160

[6] C. Sakr, Y. Kim, and N. Shanbhag, "Analytical guarantees on numerical precision of deep neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3007–3016.

[7] Y. Zhou, S.-M. Moosavi-Dezfooli, N.-M. Cheung, and P. Frossard, "Adaptive quantization for deep neural network," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, 2018, pp. 1–9.

[8] W. Zhe, J. Lin, V. Chandrasekhar, and B. Girod, "Optimizing the bit allocation for compression of weights and activations of deep neural networks," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2019, pp. 3826–3830.

[9] F. Tung and G. Mori, "CLIP-Q: Deep network compression learning by in-parallel pruning-quantization," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 7873–7882.

[10] Z. Liu, X. Zhang, S. Wang, S. Ma, and W. Gao, "Evolutionary quantization of neural networks with mixed-precision," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Jun. 2021, pp. 2785–2789.

[11] Y. Yuan, C. Chen, X. Hu, and S. Peng, "EvoQ: Mixed precision quantization of DNNs via sensitivity guided evolutionary search," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2020, pp. 1–8.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.

[13] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: http://arxiv.org/abs/1409.1556

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[15] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.

[16] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A white paper on neural network quantization," 2021, *arXiv:2106.08295*. [Online]. Available: http://arxiv.org/abs/2106.08295

[17] 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. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713.

[18] J. Faraone, N. Fraser, M. Blott, and P. H. W. Leong, "SYQ: Learning symmetric quantization for efficient deep neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4300–4309.

[19] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless CNNs with low-precision weights," 2017, *arXiv:1702.03044*. [Online]. Available: http://arxiv.org/abs/1702.03044

[20] C. Leng, Z. Dou, H. Li, S. Zhu, and R. Jin, "Extremely low bit neural network: Squeeze the last bit out with ADMM," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, 2018, pp. 1–8.

[21] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," 2016, *arXiv:1605.04711*. [Online]. Available: http://arxiv.org/abs/1605.04711

[22] N. Mellempudi, A. Kundu, D. Mudigere, D. Das, B. Kaul, and P. Dubey, "Ternary neural networks with fine-grained quantization," 2017, *arXiv:1705.01462*. [Online]. Available: http://arxiv.org/abs/1705.01462

[23] B. Martinez, J. Yang, A. Bulat, and G. Tzimiropoulos, "Training binary neural networks with real-to-binary convolutions," 2020, *arXiv:2003.11535*. [Online]. Available: http://arxiv.org/abs/2003.11535

[24] Y. Cai, Z. Yao, Z. Dong, A. Gholami, M. W. Mahoney, and K. Keutzer, "ZeroQ: A novel zero shot quantization framework," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 13169–13178.

[25] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2849–2858.

[26] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-aware automated quantization with mixed precision," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 8612–8620.

[27] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*. [Online]. Available: http://arxiv.org/abs/1510.00149

[28] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *Proc. IEEE Int. Conf. Neural Netw.*, Mar. 1993, pp. 293–299.

[29] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient ConvNets," 2016, *arXiv:1608.08710*. [Online]. Available: http://arxiv.org/abs/1608.08710

[30] I. Garg, P. Panda, and K. Roy, "A low effort approach to structured CNN design using PCA," *IEEE Access*, vol. 8, pp. 1347–1360, 2020.

[31] S. Roy, P. Panda, G. Srinivasan, and A. Raghunathan, "Pruning filters while training for efficiently optimizing deep learning networks," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2020, pp. 1–7.

[32] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, no. 1, pp. 79–86, 1951.

[33] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," *Adv. Neural Inf. Process. Syst.*, vol. 32, pp. 8026–8037, 2019.

[34] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," 2021, *arXiv:2103.13630*. [Online]. Available: http://arxiv.org/abs/2103.13630

[35] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein, "On the expressive power of deep neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2847–2854.

[36] A. Ankit, I. E. Hajj, S. R. Chalamalasetti, S. Agarwal, M. Marinella, M. Foltin, J. P. Strachan, D. Milojicic, W.-M. Hwu, and K. Roy, "PANTHER: A programmable architecture for neural network training harnessing energy-efficient ReRAM," *IEEE Trans. Comput.*, vol. 69, no. 8, pp. 1128–1142, Aug. 2020.

[37] H. Darvishi, D. Ciuonzo, E. R. Eide, and P. S. Rossi, "Sensor-fault detection, isolation and accommodation for digital twins via modular data-driven architecture," *IEEE Sensors J.*, vol. 21, no. 4, pp. 4827–4838, Feb. 2021.

[38] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.

[39] Z. Chang, S. Liu, X. Xiong, Z. Cai, and G. Tu, "A survey of recent advances in edge-computing-powered artificial intelligence of things," *IEEE Internet Things J.*, vol. 8, no. 18, pp. 13849–13875, Sep. 2021.

EFSTATHIA SOUFLERI (Graduate Student Member, IEEE) received the bachelor's degree from the Department of Mathematics, National and Kapodistrian University of Athens, Greece, in 2016, and the master's degree (Hons.) in computer science from the University of Thessaly, Greece, in 2017. She is currently pursuing the Ph.D. degree in electrical and computer engineering with Purdue University, West Lafayette, IN, USA, under the guidance of Prof. Kaushik Roy.

She is also a Research Assistant with the Nanoelectronics Research Laboratory (NRL). Her primary research interests include the field of neural network compression and neuro-inspired algorithms for efficient learning. She was awarded a scholarship from the Greek National Foundation for academic excellence during her studies.

KAUSHIK ROY (Fellow, IEEE) received the B.Tech. degree in electronics and electrical communications engineering from IIT Kharagpur, Kharagpur, India, and the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Illinois at Urbana–Champaign, in 1990.

He was with the Semiconductor Process and Design Center, Texas Instruments, Dallas, where he worked on FPGA architecture development and low-power circuit design. He joined the Faculty of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, USA, in 1993, where he is currently working as Edward G. Tiedemann, Jr. Distinguished Professor. He was working as Purdue University Faculty Scholar, from 1998 to 2003. He was a Research Visionary Board Member of Motorola Labs, in 2002, and held a position of M. Gandhi Distinguished Visiting Faculty with IIT Bombay and the Global Foundries Visiting Chair with the National University of Singapore. He is also the Director of the Center for Brain-Inspired Computing (C-BRIC) funded by SRC/DARPA. He has published more than 700 articles in refereed journals and conferences, holds 25 patents, and supervised 85 Ph.D. dissertations, and has coauthored two books *Low Power CMOS VLSI Design* (John Wiley & McGraw Hill). His research interests include neuromorphic and emerging computing models, neuro-mimetic devices, spintronics, device-circuit algorithm co-design for nano-scale silicon, non-silicon technologies, and low-power electronics. He received the National Science Foundation Career Development Award, in 1995, the IBM Faculty Partnership Award, the ATT/Lucent Foundation Award, the 2005 SRC Technical Excellence Award, the SRC Inventors Award, Purdue College of Engineering Research Excellence Award, Humboldt Research Award, in 2010, the 2010 IEEE Circuits and Systems Society Technical Achievement Award (Charles Desoer Award), the Distinguished Alumnus Award from IIT Kharagpur, Fulbright-Nehru Distinguished Chair, the DoD Vannevar Bush Faculty Fellow, from 2014 to 2019, Semiconductor Research Corporation Aristotle Award, in 2015, the 2020 Arden Bement, Jr. Award, the Highest Research Award given by Purdue University in pure and applied science and engineering, and the Best Paper Awards at 1997 International Test Conference, the IEEE 2000 International Symposium on Quality of IC Design, the 2003 IEEE Latin American Test Workshop, the 2003 IEEE Nano, the 2004 IEEE International Conference on Computer Design, the 2006 IEEE/ACM International Symposium on Low Power Electronics and Design, and the 2005 IEEE Circuits and System Society Outstanding Young Author Award (Chris Kim), the 2006 IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS Best Paper Award, the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design Best Paper Award, and the 2013 IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS Best Paper Award. He was a Guest Editor for Special Issue on Low-Power VLSI in *IEEE Design and Test*, in 1994, the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, in June 2000, the *IEE Proceedings–Computers and Digital Techniques*, in July 2002, and the IEEE JOURNAL ON EMERGING AND SELECTED TOPICS IN CIRCUITS AND SYSTEMS, in 2011. He has been in the Editorial Board of *IEEE Design and Test*, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, and the IEEE TRANSACTIONS ON ELECTRON DEVICES.