Bits-Ensemble: Towards Light-Weight Robust Deep Ensemble by Bits-Sharing

Yufei Cui[1], Shangyu Wu[2], Qiao Li[3*], Antoni B. Chan[2], Tei-Wei Kuo[4], Chun Jason Xue[2]
[1]School of Computer Science, McGill University, [2]Department of Computer Science, City University of Hong Kong, [3]School of Informatics, Xiamen University, [4]Department of Computer Science and Information Engineering, National Taiwan University

***Abstract*—Robustness and uncertainty estimation are crucial to the safety of deep neural networks deployed on the edge. The deep ensemble model, composed of a set of individual deep neural networks (namely members), has strong performance in accuracy, uncertainty estimation, and robustness to out-of-distribution data and adversarial attacks. However, the storage and memory consumption increases linearly with the number of members within an ensemble. Previous works focus on selecting better members, layer-wise low-rank approximation of ensemble parameters and designing partial ensemble model for reducing the ensemble size, thus lowering storage and memory consumption. In this work, we pay attention to the quantization of the ensemble, which serves as the last mile of network deployment.**

We propose a differentiable and parallelizable bit sharing scheme that allows the members to share the less significant bits of parameters, without hurting the performance, leaving alone the more significant bits. The intuition is that, numerically, more significant bits (e.g., the bit for the sign) are more useful in distinguishing a member from other members. For real deployment of the bit-sharing scheme, we further propose an efficient encoding-decoding scheme with minimal storage overhead. Experimental results show that, BitsEnsemble reduces the storage size of ensemble for over 22×, with only 0.36× increase in training latency, and no sacrifice of inference latency. The code is available in https://github.com/ralphc1212/bitsensemble.

***Index Terms*—Deep ensemble, edge computing, neural network quantization, Bits-Ensemble**

I. Introduction

Safety and Robustness are crucial for the deep neural networks (DNNs) deployed on the edge side. The potential hazards may comes from the nature of over-confidence in DNNs [3], [5], [12], [13], [15], [21], [32], data domain shift [27], [31] and adversarial attack [6], [14], [25], [39] in real world. The predictions and predictive distributions of DNNs are used to make decisions in important applications on edge, e.g., self-driving car [2] or medical diagnoses from imaging [8]. A misclassification caused by domain shift or adversarial attack, associated with high predictive confidence, might cause catastrophic consequences. For example, the first failure of self-driving car is caused by the perception system was confused the white side of a trailer for bright sky [21]. A person wearing an adversarial T-shirt in different poses can stay undetected by a deep object detector with high success rate [38].

The deep ensemble model has strong performance in accuracy, uncertainty (confidence) calibration, robustness to data shift and adversarial attack [23], [28], [39]. It contains a set of deep neural networks (namely members) that jointly make decisions for each input data. The reason an ensemble performs well could be explained from a loss landscape perspective [9]. Specifically, different members fall into different local optima, thus capture a global uncertainty instead of a local uncertainty around one local optima. Empirically, adding members would increase the ensemble performance in accuracy and robustness [13], [23].

However, the computation operations and storage increase linearly with the size of ensemble, which makes an ensemble hard to fit in resource-limited edge devices. Previously, for mitigating this issue, researchers and practitioners study averaging the members' parameters within an ensemble to generate one powerful model [11], [17], [26]. However, except for Bayesian model averaging, these methods are highly non-explainable. The over-confidence issue could not be resolved as a single deterministic network is generated. Recent works explore carefully designing an ensemble with moderate size using evolutionary search algorithm [40], selecting active members with sparse gates [29], fitting the ensemble in one DNN [37] and combining Bayesian neural network with the ensemble [7]. There is no much attention paid on the developing *network quantization* [16], [19] scheme for a deep ensemble.

We focus on this last mile of deploying the ensemble models on the edge devices. Instead of directly applying the existing quantization scheme to the deep ensemble, we exploit the *numerics similarities* between the quantization bits. The intuition is that the less significant bits could be less influential to the diversity and performance of ensemble members, while they take a large proportion of the quantization bits. The quantization scheme with low bit-width [18], [33] chooses to discard the less significant bits or prune them progressively during training. We choose to cluster the similar less significant bits between ensembles and derive the representation bits for a cluster. The representation bits for less significant bits are still maintained for refining the quantization of members within its cluster, namely *bits-sharing*. The idea of bits-sharing is described in Fig. 1.

A diversity analysis is conducted to validate the idea of bits-sharing and shows a potential in saving quantization bits.

Manuscript received April 07, 2022; revised June 11, 2022; accepted July 05, 2022. This article was presented at the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES) 2022 and appeared as part of the ESWEEK-TCAD special issue.
*Corresponding author: Qiao Li.

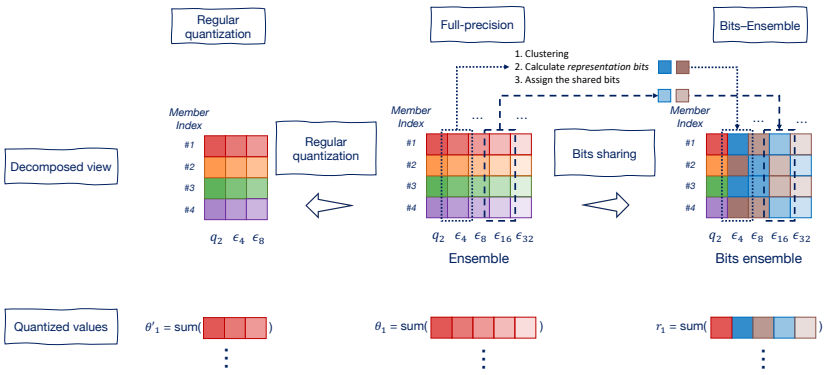

Fig. 1. The description of concepts in BitsEnsemble. The left ensemble uses the decomposition in Eq. 5. $[\theta_1, \theta_2, \theta_3, \theta_4]$ are the full-precision scalar weights for different members. Different colors represent bases or residual errors from different neural network members. The blue and brown color represents the clustered residual errors with the same representation bits. $[r_1, r_2, r_3, r_4]$ are the bits-shared and quantized scalar weights for different members. $[\theta'_1, \theta'_2, \theta'_3, \theta'_4]$ are the quantized scalar weights by pruning the less significant bits.

However, the clustering introduces hyper-parameters to tune, which is prevalent in the modern clustering algorithm, and is inefficient to be executed for training under the context of deep ensemble. The complexity and feasibility are studies in the analysis. This poses two challenges for the design of bits-sharing: differentiability and parallelizability.

In this paper, we propose a full solution to address the two problems, containing a set of new differentiable and parallelizable operands. Each operand is a replacement of the non-differentiable and non-parallelizable of its 1-dimensional counterpart. Note that some novel operands are general enough and could be potentially useful to other applications, e.g., the RBF kernel-based one-hot encoding scheme. The differentiability allows the hyper-parameter to directly learn from data, thus searching and tuning for hyper-parameter is no longer required. The parallelizability allows efficient training of the deep ensemble.

In addition, we propose a low-rank decomposition based network layer, leveraging the bits sharing technique, for further reducing the deep ensemble size. For realizing the storage and delivery of deep ensemble on edge device, we design an efficient encoding and decoding scheme for bits-sharing.

Experimental results with VGG and ResNet show that BitsEnsemble reduces the storage size of ensemble for over 22×, with only 0.36× increase in training latency, and no sacrifice of inference latency. The performance is also improved over the quantized state-of-the-art ensemble, due to the diversity induced by the learnable clustering hyper-parameters.

The proposed technique is named BitsEnsemble. To the best of our knowledge, this is the first work that exploits saving bits for the quantization of extensively deployed deep ensemble model.

II. Preliminary and Analysis

In this section, we present the scheme of sharing quantization bits of ensemble models. First, the preliminaries about ensemble and quantization are reviewed. Second, the basics of problem in bits-sharing is presented. Third, an analysis on real dataset with deep ensemble is provided, to motivate this work.

TABLE I
List of symbols.

Symbol	Description
s	Step size for quantization
$[\alpha, \beta]$	Upper bound β and lower bound α for quantization
x	Input for different functions
q_2	Base quantization bits
ϵ_4	Residual error at 4 bit level, encoded with 2 bits, et cetera
$\mathcal{E}$	Residual error matrix
ϵ'	sorted residual errors
θ	A (quantized) weight scalar
$\mathbf{\Theta}$	Quantized weight matrix
$\bar{\mathbf{\Theta}}$	Quantized weight matrix by bits sharing
k	Index of a member
K	Ensemble size
$\mathbf{\Psi}_k$	All parameters of the k-th member
$f_k(\cdot; \mathbf{\Psi}_k)$	The k-th member
m	Index of a bin within $[0, 1]$
M	Number of bins
Λ	Scalar clustering parameter
δ	Distance scalar of adjacent elements in sorted residual errors
$\mathbf{\Delta}$	Distance matrix of adjacent elements in sorted residual errors
p	Binary variable indicating a partitioning
$\mathbf{p}$	Vector of binary variable indicating a partitioning
$\mathbf{P}$	Matrix form of $\mathbf{p}$
$\mathbf{q}$	Cummulative summation of $\mathbf{p}$
I	Index matrix of sorting
T	One-hot encoded index matrix of sorting
$\mathbf{W}$	Weight matrix of a layer
π	Index for the merged weight dimension
Π	Merged weight dimension
$\mathcal{L}$	Training objective
λ	Temperature for compare function
$\mathbf{C}$	Clustering matrix
$\mathbf{M}$	Matrix of representation bits
$\mathbf{R}'$	Output of bits sharing in sorted order
$\mathbf{R}$	Output of bits sharing
$\mathcal{K}_x$	RBF kernel at center x
D_{in}	Input dimension or number of input channels
D_{out}	Output dimension or number of output channels
κ	Convolutional kernel size
$\mathbf{H}$	Input activation of a layer
l	Layer index
$\mathbf{\Omega}$	Shared matrix in BatchEnsemble
$\mathbf{S}_k$	k-the rank-1 matrix in BatchEnsemble
$[\mathbf{s}_k^{\text{out}}, \mathbf{s}_k^{\text{in}}]$	A pair of vector for $\mathbf{S}_k$
$\mathbf{U}$	Low-rank matrix in BitsEnsemble for input dimension
$\mathbf{V}$	Low-rank matrix in BitsEnsemble for output dimension

A. Preliminaries

a) Quantization: The weights and activation quantization of a single deterministic neural network have been extensively explored [4], [20], [30], [35]. One branch is to design a quan-

tization function that applies to the weights and activations for generating the discrete values. However, due to the complex design of quantization functions, the deployment of some quantized networks is obscure.

The additive power-of-two quantization bit-width [19], [34] allows the multiplication to be calculated by bit-shift, which is compatible with modern mobile hardware. Consider the input θ of a quantization function in the range $[\alpha, \beta]$ is uniformly quantized with the bit-width b:

$$\theta_q = s\lfloor\frac{\theta}{s}\rceil, \quad s = \frac{\beta-\alpha}{2^b-1} \tag{1}$$

where the integer generated by $\lfloor\frac{\theta}{s}\rceil$ could be represented in binary bits.

Consider the quantization of θ with $b = 2$:

$$\theta_2 = s_2\lfloor\frac{\theta}{s_2}\rceil, \quad s_2 = \frac{\beta-\alpha}{2^b-1} \tag{2}$$

The residual error is $\theta - \theta_2$, which could be further quantized by the lower-level bits, say the next two bits. Then,

$$\epsilon_4 = s_4\lfloor\frac{\theta-\theta_2}{s_4}\rceil, \quad s_4 = \frac{s_2}{2^2+1} \tag{3}$$

Note that ϵ_4 is quantized by 2 bits instead of 4. The key feature in this quantization scheme is that, by addition, $\theta_4 = \theta_2 + \epsilon_4$, the quantized tensor θ_4 could be obtained, which has an effective bit width of $b = 4$ with a step size of $s_4 = \frac{\beta-\alpha}{2^4-1}$. In other words, the high precision quantization (θ_4, 4 bits) could be obtained by summation of *base* quantization (θ_2, 2 bits) and *residual error* quantization (ϵ_4, $4-2=2$ bits). This property of addition could be extended to any-bit quantization, e.g.,

$$\theta = q_2 + \epsilon_4 + \epsilon_8 + \epsilon_{16} + \epsilon_{32}, \tag{4}$$

where each quantized residual error could be obtained recursively with the example approach. In Bayesian Bits [34], a probabilistic training method is proposed for automatically determining the quantization bits, by adding a binary gate for each residual error. ABS [24] uses a loss-aware objective for training the binary gates of residual errors.

b) Ensemble: The deep ensemble model is a set of deep neural networks (DNNs), $\{f_k(\cdot;\Psi_k)\}_{k=1}^K$, where K is the ensemble size and Ψ_k represents the parameters for k-th member. Each DNN is randomly initialized and trained with the whole training datasets $\mathcal{D}_{\text{tr}}$ individually. For a testing data x^* in testing set $\mathcal{D}_{\text{te}}$, each DNN predicts it's own results $f_k(x^*;\Psi_k)$. All prediction results are aggregated to generate the final prediction y^*. General aggregation methods includes majority voting, averaging and weighted averaging.

The naive way for ensemble quantization is to quantize the ensemble by power-of-bits quantization functions. By allowing the adaptive bit-width, each member could be reduced to a moderate size such that the storage and memory consumption could be saved.

In this work, we propose to exploit the numerical similarity of quantization bits among different ensembles. The similar quantization bits could be clustered into a group and represented with the shared bits, namely *representation bits*. The intuition is that, the base quantization bits, e.g., x_2, determine the main value that the a member weight should be based on. The high level quantization bits, namely the less significant bits, only determine the residual quantization errors, thus refine the base values. Clustering the similar less significant bits would introduce trivial impact on the network diversity, thus approximates the full-precision ensemble performance better.

B. Basics of BitsEnsemble

The members in an ensemble could be different in structure. However, heterogenous structures cause a large overhead for parallelization. In this work, we study the ensemble with the same structure but different initializations, which was shown to perform well in the literature.

We present the basic setup which is generally applicable for the ensembles. Say the ensemble size (the number of members) is K. A weight scalar at some position has K corresponding scalars for different members. Each scalar could be decomposed into weights using the decomposition in Eq. 2 and Eq. 3.

$$\begin{aligned}\theta^1 &= q_2^1 + \epsilon_4^1 + \epsilon_8^1 + \epsilon_{16}^1 + \epsilon_{32}^1,\\ &\ldots,\\ \theta^K &= q_2^K + \epsilon_4^K + \epsilon_8^K + \epsilon_{16}^K + \epsilon_{32}^K\end{aligned} \tag{5}$$

We named it the decomposed view. As the base bits are crucial for the value, we only consider clustering the residual quantization errors ϵ's. The objective is to find N cluster centers that covers all values, with minimal number of cluster centers and low cluster sizes. For example, in the bit-level 4, $[\epsilon_4^1, \epsilon_4^2, \ldots, \epsilon_4^K]$ are clustered and N cluster centers are calculated. The cluster centers are taken as the *representations bits* to replace other bits. As fewer bits are used across member 1 to member K, the major methodology is named *bits-sharing*.

C. Motivation of bit-sharing

1) Setups and metrics: Before diving into the methodology, an empirical analysis is conducted on quantization on a trained ensemble, namely post-quantization, to motivate the bits-sharing scheme. The correlation of weights in the ensemble is first analyzed. The accuracy, calibration and ensemble diversity are evaluated for validating our main idea. We note that, this example might not have the best performance under this setting, but is for providing the insight for bits-sharing.

We introduce the following metrics for evaluating the ensemble performance.

- Accuracy: the accuracy evaluated with averaged outputs of an ensemble.
- Expected calibration error (ECE): the calibration errors by dividing the predicted probability range [0, 1] into bins $\{B_m\}_{m=1}^M$ and taking average of the calibration over bins.

$$\text{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{n} |\text{acc}(B_m) - \text{conf}(B_m)|, \tag{6}$$

where n is the number of data points, M is the number of bins. This metric is important for evaluating if a learning model is *over-confident*. A perfect calibration means the averaged prediction confidence is equivalent to the overall accuracy, which is reached when $\text{ECE} = 0$.

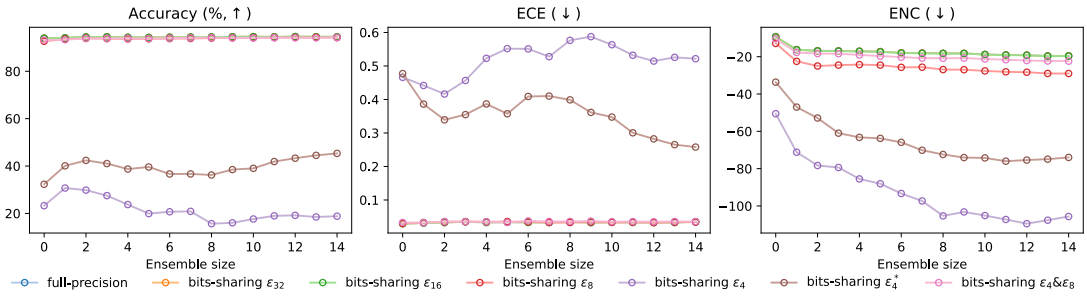

Fig. 2. The performance of deep ensemble under different member size and bits-sharing scheme in analysis. The first four bits-sharing schemes set Λ to be the median of bit distances. $\text{bits}-\text{sharing}\,\epsilon_4^*$ uses a tuned smaller Λ. $\text{bits}-\text{sharing}\,\epsilon_4\&\epsilon_8$ is at the two levels with two tuned Λ's.

TABLE II
THE PERCENTAGE OF SAVED BITS UNDER DIFFERENT BITS-SHARING (BS) SCHEME.

Name	FP	BS ϵ_{32}	BS ϵ_{16}	BS ϵ_8	BS ϵ_4	BS ϵ_4^*	BS $\epsilon_4\&\epsilon_8$
Saved bits in total (% ↑)	0	46.7	73.4	86.7	93.3	90.4	90.1
Saved bits at its level (% ↑)	0	93.3	93.5	93.3	93.3	47.5	79.2

- Expected negative correlation (ENC): The negative correlation (NC) is a metric to evaluate the diversity according to the model output.

$$\begin{aligned}\text{NC} &= \text{div}(f_k(\mathbf{x}); f_{1:K}) \\ &= (f_k(\mathbf{x}) - \bar{f}(\mathbf{x}))[\sum_{j\neq k}(f_j(\mathbf{x}) - \bar{f}(\mathbf{x}))]), \end{aligned} \quad (7)$$

where $\bar{f}(\mathbf{x}) = \frac{1}{K}\sum_{k=1}^{K} f_k(\mathbf{x})$. The ENC is the expected NC over all members. Previous works use it for enhancing the diversity of boosted models. Here, it could be a portable metric for evaluating the diversity in deep ensemble. A diverse ensemble tends to explore more local optima in the loss landscape, thus could be potential to perform well.

The accuracy and ECE dominate the ENC metric as they are directly related to the performance.

The experiments are conducted on the hand-written digits dataset with 10 classes. A full-precision ensemble of size 16 is trained first. Each member in the full-precision ensemble is a neural network with structure 784-128-10 and Relu activation. Each network is trained individually on the training set. Fig. 2 shows the performance with increased ensemble size. Under full precision setup, the ensemble accuracy increases from 92.1% to 94.6% by increasing the ensemble size. ECE decreases from 0.035 to 0.029, while ENC decreases from -9.25 to -19.54.

2) Correlation of weights: Afterwards, we perform the decomposition presented in Eq. 5 for quantizing the trained parameters and bits-sharing.

We empirically motivates the idea of clustering bits at the same location by the similarity of weights. The analysis is performed on a ensemble of 2-layer neural network trained on the MNIST dataset. The first metric is cosine similarities of parameters of different members.

$$\text{Cos}(\mathbf{w_1}, \mathbf{w_2}) = \frac{\mathbf{w_1^T w_2}}{||\mathbf{w_1}||\,||\mathbf{w_2}||} \quad (8)$$

$$\text{Corr} = \frac{K^2 - K}{2}\sum_{i=1}^{K-1}\sum_{j=i}^{K}\text{Cos}(\mathbf{w_i}, \mathbf{w_j}) \quad (9)$$

$[\mathbf{w_1}, \ldots, \mathbf{w_K}]$ indicates the parameter vectors of different members. The reason of using cosine similarity is to obtain the correlations of consequent clustering scheme in high-dimensional space. The second metric is based on variance.

$$\text{disp} = \text{mean}(\text{variance}([\mathbf{w_1}, \ldots, \mathbf{w_K}]^\mathbf{T})) \quad (10)$$

where disp means dispersion. A lower disp means the elements are more correlated. The inside variance is taken along the second dimension of the matrix $[\mathbf{w_1}, \ldots, \mathbf{w_K}]^\mathbf{T}$ and generates a vector of variance. For example, the first entry of this vector is the variance over $[w_{11}, w_{21}, \ldots, w_{K1}]$. The outside mean is taken over this vector. This metric is to show the numeric similarity of different members at the same location. A smaller variance shows the a higher correlation.

There are four types of parameters in the tested network: layer-1-weight, layer-1-bias, layer-2-weight and layer-2-bias. The weight matrix is flattened as a vector for calculating the correlation. For comparison, we perturb the parameters by reversing the entry locations of half of $\mathbf{w_i}$'s. Furthermore, we randomly perturb the parameters by shuffling the entries' location of each $\mathbf{w_i}$'s. The random perturbations are executed for 3 times and the average results are presented.

TABLE III
THE WEIGHT CORRELATION OF THE TRAINED ENSEMBLE. NOR. (NORMAL), REV. (REVERSE), RAN. (RANDOM)

	Corr ($\times 10^{-2}$)			disp ($\times 10^{-3}$)		
Name	Nor.	Rev.	Ran.	Nor.	Rev.	Ran.
layer-1-weight	**1.2**	1.0	0.2	**4.5**	4.5	4.7
layer-1-bias	**6.7**	6.3	5.8	**5.0**	5.1	5.5
layer-2-weight	**4.1**	3.6	1.0	**22.2**	22.4	23.0
layer-2-bias	**55.6**	46.6	7.4	**2.8**	4.1	6.9

The results show that among different members, high correlations are observed in the same location. Furthermore, exhaustively exploiting similarity of weights at different locations might find parameters that has higher numerical similarity, but introduces an indexing matrix to remember which elements be clustered and stored, for each parameter matrix. This contradicts with our idea of saving storage space with the bits-sharing scheme.

3) A smart sharing scheme saves bits: We propose an unsupervised clustering algorithm with an adjustable cluster distance Λ presented in Algo. 1, which is a vanilla version of

bits-sharing. Λ is a hyper-parameter for thresholding the gap of the sorted residual errors. Whenever the algorithm meets a gap less than Λ, the buffered residual errors are put in one cluster and the representation is calculated (line 10 - line 15). By controlling the value of Λ, we could determine how sensitive and frequent the algorithm make a clustering operation. By properly tuning Λ, the center distance and the optimal cluster size could be determined. In this example, we tune Λ to be the median of all possible distances between bits. The bits-sharing is conducted on each bit level first. For example, at the ϵ_8 bit level, all ϵ_{32}'s and ϵ_{16}'s are dropped, we only cluster ϵ_8's. Fig. 2 shows the performance with different bits-sharing scheme. Tab. II shows the corresponding percentage of saved bits. It could be concluded that bits-sharing at level ϵ_{32} and ϵ_{16} performs similarly with the full precision ensemble under all metrics. It is worth to note that, bits-sharing at ϵ_8 provides an improvement of ENC when maintaining high accuracy and ECE. However, when more significant bits (ϵ_4) are shared, the performance drops significantly. Although ENC is improved due to the large numerical difference, it could not provide an evidence for better performance due to the large drops of accuracy and ECE. Under such case, increasing the ensemble size could not provide performance gain.

Algorithm 1 1-dimensional bits-sharing

Require: $[\epsilon_1, \ldots, \epsilon_K]^T$, scalar or vector Λ
1: initialize a list cluster of size K
2: counter = 0, buffer = 0, milestone = 1
3: $([\epsilon'_1, \ldots, \epsilon'_K]^T$, I) = sort($[\epsilon_1, \ldots, \epsilon_K]^T$)
4: $[\delta_1, \ldots, \delta_K]^T = [\epsilon'_2, \ldots, \epsilon'_K]^T - [\epsilon'_1, \ldots, \epsilon'_{K-1}]^T$
5: $\mathbf{p} = \text{compare}([\delta_1, \ldots, \delta_{K-1}]^T, \Lambda, >)$
6: $\mathbf{p} = [p_1, \ldots, p_{K-1}, 1]^T$
7: **for** i from 1 to K **do**
8: buffer += ϵ^k
9: counter += 1
10: **if** p_i == 1 **then**
11: avg = buffer / counter
12: cluster[milestone : i] = avg
13: milestone = i + 1
14: buffer = 0
15: **end if**
16: **end for**
17: result = map_back(cluster, index)
18: **Return** result

Then, we set Λ to be the mean of all possible distances between bits to obtain bits-sharing ϵ_4^*. The brown line (bits-sharing ϵ_4^*) in Fig. 2 shows the performance is improved but still far from the others. The percentage of saved bits at this level is also reduced to 47.5%. For finding a better setup, a bits-sharing at two levels, ϵ_4 and ϵ_8, is executed. The Λ's are explored by grid search. Under this evaluation, the performance is improved to a comparable level with full precision ensemble. The saved bits at each level is also improved by around 31.7% compared with bits-sharing ϵ_4^*.

We could draw the conclusion that a properly tuned clustering distance, thus clustering centers for each weight, is useful for the bits-sharing scheme. However, how to determine the clustering centers with general clustering algorithms is difficult. Following the notation in Sec.II.A, for a deep neural networks, Ψ_k consists of parameters from different layers $\Psi_k = [\mathbf{W}_l]_l$, where l is the index of a layer. Clustering the residual errors at each bit level is a classical NP-hard K-center problem. For example, for the 4-bit level, $[\epsilon_4^1, \ldots, \epsilon_4^\Pi]$, the problem is to find the K cluster centers that covers all values, with minimal number of cluster centers and low cluster size. As the modern neural network has at least millions of parameters, solving the problem with non-parallelizable heuristics is both inefficient and less effective.

From another perspective, if a search algorithm is adopted. The search space for the clustering of an deep ensemble is

$$4 \cdot 2^K \cdot \sum_{k=1}^{K} |\Psi_k|, \tag{11}$$

where $|\Psi_k|$ represents the number of parameters in a DNN member. The number 4 means there are 4 bit levels. 2^K is the number of possible combinations. Each search would be associated with a forward pass of the whole ensemble. As the size of neural network is notoriously large (e.g. ResNet50 has over 23 million parameters), it is inefficient to directly searching for the best parameters for the bits clustering.

Furthermore, the analysis uses post-training quantization for an easy exposition. A post-training quantization for large-scale neural network might introduce relatively large quantization error.

This drives us to explore a fully learnable bits-sharing scheme, for efficiently optimizing the clusters of bits and prediction objective jointly.

III. Parallelizable and differentiable bits-sharing

The core algorithm of our bits-sharing is based on the numerical distance of sorted elements. The vanilla algorithm is shown in Algo. 1. The idea is to first sort the residual errors at some bit level, then perform a partitioning operation according to the gap of adjacent elements. The effect of sorting is to make the similar numerics aligned closely. Then a simple comparison operation could generate the clustering.

$$\begin{aligned} &\mathbf{p} = \text{compare}(\delta, \Lambda, >) \\ &p_i = 1, \quad if\ \delta_i > \Lambda_i, \quad else\ p_i = 0 \end{aligned} \tag{12}$$

A brief elaboration of the algorithm is as follows: At line 3, the residual errors are sorted with the indices corresponding to the original location returned. Λ's are calculated as the gaps of adjacent elements in the sorted vector. The gaps are then compared with the elements in Λ which determines if a partitioning should take effect. Lines 7-16 denote that the mean of elements partitioned in a cluster is calculated as the representation bits, which are re-assigned to these elements then. Line 17 maps the updated elements to the original location before sorting.

The algorithm complexity of comparing and re-assigning values is $\mathcal{O}(K)$, while that of whole algorithm is bounded by the sorting algorithm. However, comparing with the other clustering-based algorithms, sorting is of lower time complexity and could be efficiently parallelized for all parameter

values. More importantly, this vanilla bits-sharing scheme makes it possible to build learnable bits-sharing scheme, which will discussed in the following subsections.

To make the vanilla Algo. 1 work for the whole neural network efficiently, the following key points and challenges arise:

- The whole algorithm should be parallelizable for all elements in neural network tensors in order to be efficiently executed in an element-wise fashion.
- The algorithm should provide useful gradients for Λ, such that the bits-sharing scheme could learn from data instead of being searched or tuned manually.

The reason why element-wise execution is preferred than clustering high dimensional vectors or matrices is two-fold: 1) besides 1-dimensional numerical distance, there is no direct correlation between the multi-dimensional representation of residual errors; 2) high-dimensional clustering is inefficient to execute during training of neural network; 3) it is obscure to design a learnable clustering without introducing much hyperparameters.

A software solution for parallelizing is using library that supports functions like for-loop and the condition function, e.g., Jax [10]. However, the library could not provide useful gradients as the gradients are not properly defined. Meanwhile, the intermediate variable incurs large overhead for optimizing the GPU memory usage.

In this section, we propose a package of solutions for solve the two challenges. Each solution contains a novel technique which is potentially useful to more fundamental sub-problems that Algo. 1 induces.

A. General setups

We consider a weight matrix of an neural network layer $\mathbf{W} \in \mathbb{R}^{K\times\Pi}$, where K is the ensemble size, Π is the product of input and output dimension or the product of (output channel, input channel, kernel size, kernel size) for convolutional layer. $\mathbf{W}$ could be recursively quantized in a parallelizable fashion and written in Eq. 5.

$$\begin{aligned} \mathbf{\Theta} &= \text{quant}(\mathbf{W}) \\ \theta_{k\pi} &= q_2^{k\pi} + \epsilon_4^{k\pi} + \epsilon_8^{k\pi} + \epsilon_{16}^{k\pi} + \epsilon_{32}^{k\pi} \end{aligned} \tag{13}$$

The objective is designing a replacement of Algo. 1 to generate $\bar{\mathbf{\Theta}}$ that is parallelizable over $[\theta_{:1}, \ldots, \theta_{:K}]$ and has useful gradient $\frac{\mathrm{d}}{\mathrm{d}\Lambda}\mathcal{L}$, where $\mathcal{L}$ is the training objective. Θ' is supposed to have shared bits which has reduced bits usage compared with $\mathbf{\Theta}$.

The presented solutions assume a particular bit-level, e.g., $\mathcal{E}_4 = [\epsilon_4^{k\pi}]^{k\pi}$, similar to Algo. 1, and follow the execution order in Algo. 1. The key computation modules are highlighted with ***bold italic font*** in the paper.

B. Compare function: $\text{compare}(\delta, \Lambda, >)$

Sorting is a well-explored algorithm with mature parallel computing software support. Letting the gradient $\frac{\mathrm{d}}{\mathrm{d}\epsilon}\epsilon'$ pass sorting is straightforward as $\epsilon \to \epsilon'$ is a one-to-one mapping memorized by the computation flow. Therefore, line 3-4 are already parallelizable and differentiable for input matrix $\mathcal{E}$.

Assume we obtained a matrix $\Delta = [\delta_{k\pi}]_{k\pi}$ that represents the difference of the bits at each weight dimension π, we use the following function to replace Eq. 14 for the ***compare*** function,

$$p = g(\delta, \Lambda, \lambda) = \frac{1}{1 + \frac{1}{\lambda}\exp(\delta - \Lambda)} \tag{14}$$

Note that $g(\cdot)$ is based on the sigmoid function with a temperature λ.

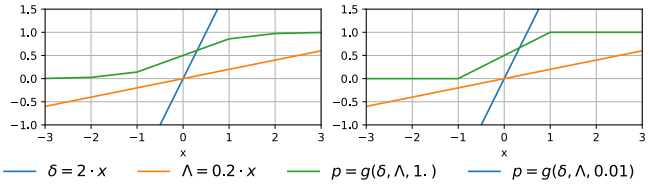

Fig. 3. The differentiable compare function under different temperature. x is a dummy variable for constructing δ and Λ.

It could observed that the designed function g realize the compare function with soft relaxation. When $\delta > \Lambda$, the value gradually changes to 1. Otherwise, it approaches to 0. When λ is large, g generates a smoother function that increases slowly. Otherwise, a hard function with steeper change is generated, with less approximation error to an exact step function.

As all the operands in Eq. 2 is differentiable, the gradient $\frac{\mathrm{d}}{\mathrm{d}\Lambda}g(\delta, \Lambda, \lambda) = -\frac{1}{\lambda(1+\frac{1}{\lambda}e^{(\delta-\Lambda)})^2}$ is calculated in closed form. As all operands have the vectorized implementation with modern computation library, the compare operation could be fully parallelizd for the matrix Δ to generate $\mathbf{P} = [\mathbf{p}_1, \ldots, \mathbf{p}_\Pi]$. Instead of paddding 1 to the tail of $\mathbf{p}_\pi$ like Algo. 1, we ***pad 0*** as the first element of $\mathbf{p}_\pi$. The functionality will be illustrated in Sec. III-C.

As the function might generate a value between $[0, 1]$ for $p_{k\pi}$, one option is to use a rounding function for $p_{k\pi}$, namely ***Dif_round***. Estimating the gradient of rounding could use the straight-through-estimator [1], which is generally adopted in the quantization network.

C. Clustering and re-assigning

Assume we calculated the matrix $\mathbf{P} = [p_{k\pi}]_{k\pi}$ of 0 and 1, representing the partitioning of clusters at a each position, the next step is to execute the partition and calculate the representation in each cluster. The 1-dimensional counterpart is line 7-16 in Algo. 1.

The idea of parallelizing the clustering is first use $\mathbf{P}$ to construct another matrix $\mathbf{Q}$, where each cluster has the same value. To this end, we use a cummulative summation for each column $\mathbf{p}_\pi$, namely ***cummulative_sum***. For example, we assume there a $\mathbf{p}_\pi = [0, 1, 1, 0]$, which means the π-th column has the information of partitioning at the the second and third member weight. $\mathbf{q}_\pi = \text{cummulative\_sum}(\mathbf{p}_\pi) = [0, 1, 2, 2]$, with each value exactly represents the cluster index number. As cummulative_sum is highly parallelizable, it could be executed for matrix $\mathbf{P}$ in a column-wise fashion.

We use the mean of residual errors within each cluster as the representation of the cluster. If we could obtain a one-hot encoded clustering matrix $\mathbf{C}$ for $\mathbf{P}$, the clustering and weight re-assigning could be done by matrix operation. Assume the

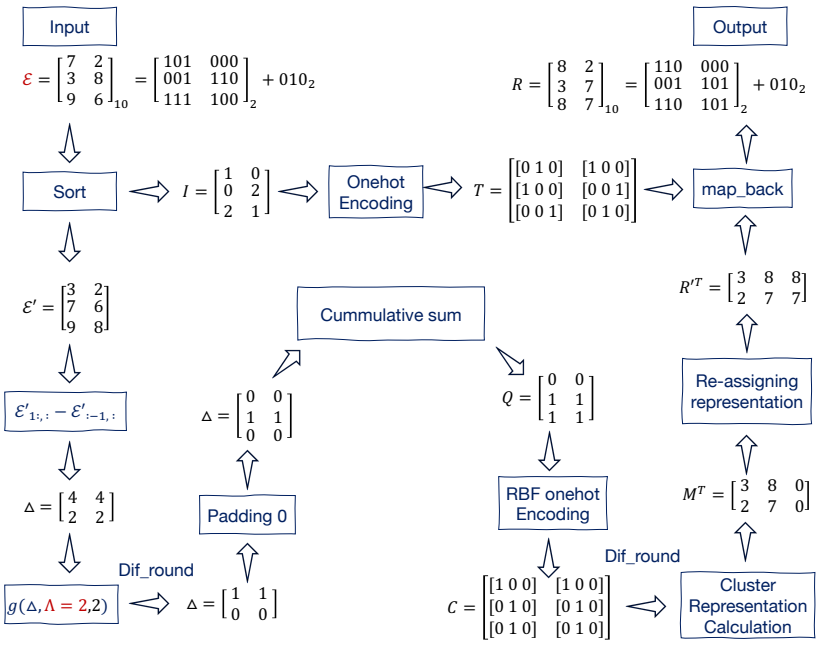

Fig. 4. An example of parallelizable and differentiable bits-sharing. For simplicity, we use integer as the residual errors at a certain bit level. The key computation modules are highlighted with ***bold italic font*** in Sec. III. The output matrix R is an approximation to input matrix $\mathcal{E}$ of residual errors, determined by the learnable clustering parameter Λ. The gradients $\frac{\mathrm{d}}{\mathrm{d}\Lambda}R$ and $\frac{\mathrm{d}}{\mathrm{d}\mathcal{E}}R$ could be computed by general computation library. The footnotes for the input and output mean decimal system.

clustering matrix is obtained, $\mathbf{C} \in \{0,1\}^{K\times\Pi\times K}$. The ***cluster representation calculation*** is done by

$$\mathbf{M} = [m_{\pi j}]_{\pi j} = \frac{\sum_k \epsilon_{k\pi} c_{k\pi j}}{\sum_k c_{k\pi j}}, \tag{15}$$

where $m_{\pi j}$ is the elements in $\mathbf{M}$ and $c_{k\pi j}$ is the elements in $\mathbf{C}$. $\mathbf{M} \in \mathbb{R}^{K\times\Pi}$ is contains the representation bits in each cluster.

$$\mathbf{R}' = r'_{k\pi} = \sum_j c_{k\pi j} m_{\pi j} \tag{16}$$

This step is to ***re-assign*** the corresponding ***representation*** values to elements within a cluster. $\mathbf{R}'$ is the final approximation to $\mathcal{E}$.

However, the problem is, the regular one-hot encoding $\mathbf{C} = \mathrm{onehot}(\mathbf{P})$ is not differentiable. For addressing this problem, we propose an ***RBF kernel-based one-hot encoding*** scheme. We place a radial basis function (RBF) kernel at each possible index of cluster.

$$\mathcal{K}_x(x') = \mathcal{K}(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right) \tag{17}$$

Using the above example, $\mathbf{p}_\pi = [0,1,1,0]$ and $\mathbf{q}_\pi = [0,1,2,2]$, the kernel functions should be place at the index $[0,1,2,3]$ as there are four possible positions. Thus, the kernels $[\mathcal{K}_{x=0}(x'), \ldots, \mathcal{K}_{x=3}(x')]$ centered at each index center scalar should be used.

Each $q_{k\pi}$ in $\mathbf{q}_\pi$ will be placed as the input x' of kernels $[\mathcal{K}_{x=0}(x'), \ldots, \mathcal{K}_{x=3}(x')]$, generating a one-hot encoded vector. For example, $[\mathcal{K}_{x=0}(q_{3\pi}=2), \ldots, \mathcal{K}_{x=3}(q_{3\pi}=2)] \approx [0,0,1,0]$. The reason is the proposed kernel only generates 1 when $x \approx x'$. The sensitivity could be controlled by setting an empirical value for the standard deviation σ.

In this way, the clustering matrix $\mathbf{C}$ could be obtained efficiently, with the pre-defined RBF kernels. This process could be fully parallelized using the modern computation library. As all operations are differentiable, the calculation of RBF one-hot encoding provides useful gradients for the whole training process.

D. Map function: map_back

As mentioned in Sec. III-B, sorting is differentiable and efficient to execute. However, the sorted and clustered residual errors should be mapped to the original location according to the index provided by sort using ***map_back***. This process is inefficient if executed one by one. Similar to the idea in clustering matrix $\mathbf{C}$, we propose a one-hot encoded index map $\mathbf{T}$ for the index.

$$\mathbf{T} = [t_{k\pi j}]_{k\pi j} = \mathrm{onehot}(\mathrm{index}) \tag{18}$$

$$\mathbf{R}^T = [r_{\pi j}]_{\pi j} = \sum_k r'_{k\pi} t_{k\pi j}, \tag{19}$$

where $\mathbf{R} \in \mathbb{R}^{K\times\Pi}$ is the final results that provided by the bits-sharing algorithm.

E. Whole workflow

An example of the whole workflow is shown in Fig. 4. We could conclude that combining the solutions provides an efficient and learnable bits-sharing scheme.

The bits-sharing results at different quantization levels are aggregated by summation, in the similar form denoted in Eq. 13.

$$\bar{\mathbf{\Theta}} = \mathbf{R}_{\mathcal{E}_2} + \mathbf{R}_{\mathcal{E}_4} + \mathbf{R}_{\mathcal{E}_8} \cdots \tag{20}$$

It is worthy to note that, although the bits-sharing scheme should be repetitively executed for different bit levels, in practice, it is only required to execute twice ($\mathbf{R}_{\mathcal{E}_4}$ and $\mathbf{R}_{\mathcal{E}_8}$) for a weight matrix. The reason is that higher levels could

be eliminated as they have subtle affection on the ensemble performance. We keep the bits at base level $\mathbf{R}_{\mathcal{E}_2}$ not shared, as these numeric are important for the ensemble diversity.

The middle results and dummy parameters including Λ will be discarded once the training is finished. The clustering matrix $\mathbf{C}$ and index matrix $\mathbf{I}$ will be aggregated and transformed into a codebook with a low storage cost (see Sec. V).

IV. Details for BitsEnsemble

The proposed bits-sharing scheme is generally applicable to all deep ensembles, as long as they share the same architecture. For persuing a practically useful ensemble with moderate size for embedded devices, we fit the deep ensemble in one neural network.

Traditionally, the ensemble should be repetitively executed or it requires large amount of memory for parallelization. One way to fit an ensemble to one neural network is to split network computation workflow into several independent branches. Each branch is feed with different training data, such that the training process of a branch does not interfere with others. During inference time, the same data is feed into different branches to obtain the ensemble output. In this way, the network could perform the inference in one pass instead of repetitively executing all networks in an ensemble. This setting was explored in BatchEnsemble [37] for improving the inference efficiency with full-precision weights. Our implementation of the network execution workflow is similar to the ensemble. We present the detailed execution flow in the appendix.[1]

We present a new method for further slimming the ensemble network: Decomposition of weight matrices. Suppose we obtained the quantized results generated by bits-sharing, $\bar{\mathbf{\Theta}} = \text{bits\_sharing}(\mathbf{W})$.

Keeping a full rank weight matrix like $\bar{\mathbf{\Theta}} \in \mathbb{R}^{K\times D_{\text{out}}\times D_{\text{in}}}$ is beneficial for the diversity of ensemble. However, from the storage perspective, it is highly inefficient as the network size is K-fold greater than a regular neural network. BatchEnsemble [37] uses a collection of rank-1 matrix $\mathbf{S}_k = \mathbf{s}_k^{\text{out}}\mathbf{s}_k^{\text{in}T}$, where $\mathbf{s}_k^{\text{out}} \in \mathbb{R}^{D_{\text{out}}\times 1}$ and $\mathbf{s}_k^{\text{in}} \in \mathbb{R}^{D_{\text{in}}\times 1}$ for the k-th members. There is a shared matrix $\mathbf{\Omega} \in \mathbb{R}^{D_{\text{out}}\times D_{\text{in}}}$, for generating the weight matrix by $\mathbf{W} = [\mathbf{W}_k]_k$ and $\mathbf{W}_k = \mathbf{S}_k \circ \mathbf{\Omega}$. More discussions of the comparisons will be presented in Sec. VI-B and Sec. VII.

Note that although BitsEnsemble is effective to execute the bits-sharing for $[\mathbf{s}_k^{\text{out}}]_k$ and $[\mathbf{s}_k^{\text{in}}]_k$, which are owned by the members, the benefits is not significant as the rank-1 decomposed vectors only take part of less than 20% of the network weights. Over 80% of the network weights belongs to $\mathbf{\Omega}$, which is shared across the members.

To take a step further on saving quantization bits, we propose to use the following low-rank decomposition for the network weights, before bits-sharing:

$$w^{kd_{\text{out}}d_{\text{in}}} = \sum_{\nu=1}^{\nu=\mu} u^{kd_{\text{out}}\nu} v^{kd_{\text{in}}\nu}, \tag{21}$$

[1]https://anonymous.4open.science/r/bits_ensemble-C6A1/

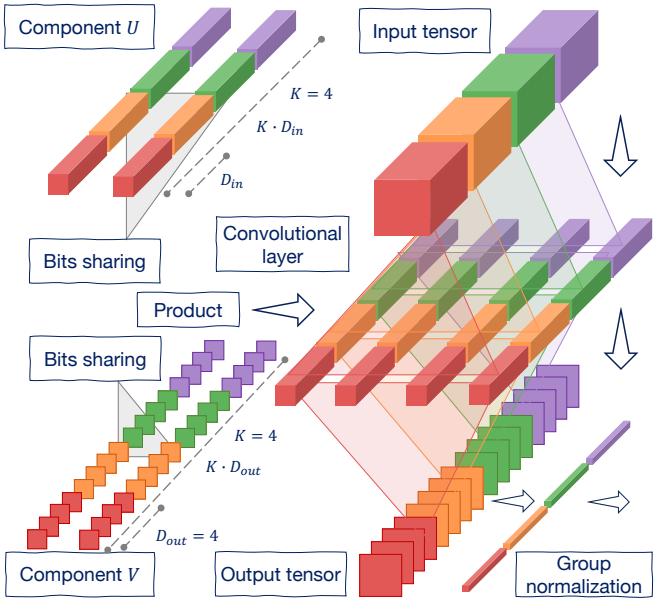

Fig. 5. The computation workflow with weight matrix decomposition, convolution and batch normalization in branch. Note that different colors denote the different members. The ensemble size $K = 4$ and the number of output channels $D_{\text{out}} = 4$ in this example. Similar workflow applies to fully connected layers, which does not have the $\kappa \times \kappa$ kernel dimension.

where $\mathbf{W} = [w^{kd_{\text{out}}d_{\text{in}}}]^{kd_{\text{out}}d_{\text{in}}} \in \mathbb{R}^{K\times D_{\text{out}}\times D_{\text{in}}}$,$\mathbf{U} = [u^{kd_{\text{out}}\nu}]^{kd_{\text{out}}\nu} \in \mathbb{R}^{K\times D_{\text{out}}\times\mu}$ and $\mathbf{V} = [v^{kd_{\text{in}}\nu}]^{kd_{\text{in}}\nu} \in \mathbb{R}^{K\times D_{\text{in}}\times\mu}$ for fully connected layers. For convolutional layers, the two components $\mathbf{U} \in \mathbb{R}^{K\times D_{\text{out}}\times\mu\times\kappa\times\kappa}$ and $\mathbf{V} \in \mathbb{R}^{K\times D_{\text{in}}\times\mu\times\kappa\times\kappa}$. The benefits of the decomposition is that, the two matrices are associated with each member, thus bits-sharing could be applied to for all components of $\mathbf{W}$, for maximally reducing the bits usage. Thus, the computation of quantized matrix becomes

$$\bar{\mathbf{\Theta}} = \text{bits\_sharing}(\mathbf{U})\text{bits\_sharing}(\mathbf{V})^T \tag{22}$$

A example of the computation workflow of the weight matrix decomposition, convolution and batch normalization in branch is shown in Fig. 5.

V. Ensemble encoding / decoding methods

With our bit-sharing scheme, the quantization bits at each quantization level could be virtually clustered. There is still a large gap how these network parameters could be efficiently stored for the storage and delivery. A proper data organization paired up with the corresponding encoding and decoding scheme is required for generating dense representations on real devices. The overhead includes the bit-sharing scheme itself, as well as metadata of the bit tensors. We design a encoding scheme for storing the bits with minimal overhead. The decoding scheme is also presented.

a) Storage layout: The quantization bits are stored in binary files, which contains a sequence of blocks. Each block consists of three kinds of bits.

- *Metadata bits.* The metadata bits indicate the bit-width of stored network parameters. It is possible that a lot of quantized network parameters have the same bit-width. Instead of storing the bit-width of each parameter,

storing the same bit-width and the number of consecutive parameters following that bit-width can be more effective to save the storage overhead of metadata.

- *Pattern bits.* The pattern bits indicate the sharing pattern between ensemble members. The sharing pattern is represented by a fixed bit-width number $code$, which indicates the order among all possible patterns. The bit-width of the $code$ is computed as the 2-based logarithm of the number of all possible patterns. It is easy to compute the number of patterns by enumeration for a small number of members. While for a large number of members, the number of patterns is the sum of the second-type Stirling Numbers [22].
- *Parameter bits.* The parameter bits are the quantization bits of network parameters. As sharing bits are for different members, for each network parameter, the quantization bits first contains the base quantization bits then the shared less significant bits.

b) Encoding/Decoding Scheme: To encode the quantization bits into a binary file, we use bit shifting to encapsulate quantization bits into few unsigned integers. For example, to encode a 2-bit parameter, a 4-bit parameter, and a 8-bit parameter into two 8-bit unsigned integers, the encoding scheme first shift all high two bits of each parameter into one integer, leaving 2 bits to receive other bits. Then, the encoding scheme shifts the bits of the residual error into the integer. Since the remaining bits of the integer are not enough for the bits of two residual errors, the encoding scheme stores the 4-bit parameter's residual error bits in the previous integer, use another integer to store the residual error bits of the 8-bit parameter. If they share the residual error bits, the encoding scheme only store once. Finally, the encoding scheme stores the final residual error of the 8-bit parameter the following bits of the integer. This can be implemented by an array of unsigned integers called bit cache and the corresponding read, write, and flush operations on the bit cache.

VI. Experiments

This section presents the experimental evaluation for the proposed BitsEnsemble scheme and the corresponding encoding and decoding scheme.

A. Experimental setup

For evaluating the performance, we train a VGG11 network and a ResNet18 neural network on Cifar10 dataset. The BitsEnsemble technique is applied to all the layers in the two networks. For comparison, the full rank network described in Sec. IV and BatchEnsemble [37] are adopted. Full rank uses a weight matrix $\mathbf{W} \in \mathbb{R}^{KD_{\text{out}} \times D_{\text{in}} \times \kappa \times \kappa}$ for the convolutional layer and $\mathbf{W} \in \mathbb{R}^{KD_{\text{out}} \times D_{\text{in}}}$ for the fully connected layer. BatchEnsemble uses a group of weight matrix for each layer $\{[\mathbf{s}_k^{\text{out}}]_{k=1}^K, [\mathbf{s}_k^{\text{in}}]_{k=1}^K, \mathbf{\Omega}\}$, where $\mathbf{s}_k^{\text{out}} \in \mathbb{R}^{D_{\text{out}} \times 1}$ and $\mathbf{s}_k^{\text{in}} \in \mathbb{R}^{D_{\text{in}} \times 1}$. For convolutional layer, $\mathbf{\Omega} \in \mathbb{R}^{D_{\text{out}} \times D_{\text{in}} \times \kappa \times \kappa}$, while for fully connected layer, $\mathbf{\Omega} \in \mathbb{R}^{D_{\text{out}} \times D_{\text{in}}}$. The two techniques use a standard power-of-2 8-bit quantization during training, without pruning or merging any bit, namely Full Rank (8 bit) and BatchEnsemble (8 bit). For fairness, our BitsEnsemble starts training with the maximal available bit level 8, that means the bits-sharing is only executed on $\mathcal{E}_4$ and $\mathcal{E}_8$, while $\mathcal{E}_{16}$ and $\mathcal{E}_{32}$ are discarded. The ensemble size is set to 4 as suggested in BatchEnsemble. As there is no approximation and bits loss on Full Rank (8 bit), its performance is supposed to be the upper bound of BitsEnsemble. We also apply the state-of-the-art BayesianBits [34] that use power-of-two quantization for quantizing the full rank ensemble. We report the full-precision results of the full rank ensemble for reference, namely Full Rank (full precision). Note that, activation quantization is an orthogonal technique to this work. Applying activation quantization to the ensemble has the same effect on the four evaluated deep ensemble.

For BitsEnsemble, we use a shared Λ over various dimensions, for introducing less overhead of parameters. For both types of layers, Λ is a vector of size D_{out} for $\mathbf{U}$ and a vector of size D_{in} for $\mathbf{V}$. The initial value for Λ is set to 0.2 for all entries in the two vectors.

A standard training procedure is adopted. We use a SGD optimizer with initial learning rate 0.01, weight decay 10^{-5} and momentum 0.9. The batch size for each member in the ensemble is 256, thus the total batch size is 1024. There is no trick like mix-up is adopted except for the warm-up learning rate scheduler, for the three techniques. The aim is to let the quantization networks converges faster. In the first 20 epochs, the learning rate is gradually increased to the initial learning rate. Then, it decreases by a factor 0.1 with a period 40 epochs. The implementation is on PyTorch and the training process is conducted on an GeForce RTX 3090 GPU.

The encoding and decoding workflow is evaluated for simulating the case that: 1) a server or an edge server encodes the updated ensemble to a piece of dense code; 2) the code is distributed to an edge device; 3) the edge device decodes the ensemble for real deployment. The implementation is on C/C++ standard library. The encoding stage is finished on a desktop with Intel Core i7-10700 CPU and 64GB memory. The decoding stage is evaluated on a Raspberry 4 device with Cortex-A72 CPU and 4GB memory.

B. Performance of BitsEnsemble

For evaluating the performance, we use the accuracy, ECE and ENC mentioned in Sec. II-C, on the testing dataset. The efficiency of training and inference are considered, as well as the storage. For the training and inference latency, we test the time within single iteration. For the model size, we present the parameter size stored by PyTorch without optimization. The model size with our encoding scheme is presented in the next section.

In terms of performance, BitsEnsemble outperforms the full-rank ensemble and BatchEnsemble on most performance metrics on the quantization tasks. On VGG11 network, BitsEnsemble has better accuracy than Full-rank ensemble (8 bit) and BatchEnsemble (8 bit) and approaches the accuracy of full-rank VGG11. BitsEnsemble provides the best expected calibration error compared with the others and has better diversity than Full-rank ensemble (8 bit) and BatchEnsemble (8 bit). On ResNet18, BitsEnsemble outperforms the other

TABLE IV
THE COMPARISONS OF PERFORMANCE.
ALL MODEL SIZES HERE ARE STORED IN *Full Precision* WITH PYTORCH IMPLEMENTATION WITHOUT OPTIMIZED ENCODING.

Name	Accuracy (%)	ECE (↓)	ENC (↓)	Training time (s) per iter	Inference time (s) per iter	Model size (MB)
Full-rank VGG11 (full-precision)	89.97	0.06	-8.12	1.02	0.91	227
Full-rank VGG11 (8 bit)	87.66	0.06	-6.15	1.07	0.90	227
Full-rank VGG11 (BayesianBits)	88.10	0.05	-7.71	1.20	0.90	227
BatchEnsemble VGG11 (8 bit)	85.31	0.08	-5.42	1.12	0.90	36
BitsEnsemble VGG11	89.78	0.05	-7.61	1.30	0.91	40
Full-rank Res18 (full-precision)	91.10	0.05	-9.31	1.53	1.12	173
Full-rank Res18 (8 bit)	90.34	0.05	-8.71	1.61	1.10	173
Full-rank Res18 (BayesianBits)	90.19	0.06	-9.67	1.73	1.11	173
BatchEnsemble Res18 (8 bit)	87.04	0.06	-4.63	1.49	1.10	117
BitsEnsemble Res18	91.41	0.05	-10.36	1.99	1.08	48

three schemes in accuracy and diversity, providing similar calibration performance for estimating the predictive confidence.

The phenomenon that BitsEnsemble outperforms Full-rank ensemble (8 bit) could tell the bits-sharing scheme and low-rank decomposition are not only helpful in reducing the ensemble size, but provides useful regularization for the training of quantized ensemble network. Combining the residual errors is not harmful to the diversity of ensemble, but provides improvement for it. This could be attributed to the learnable parameter Λ, as it could be flexibly adjusted to obtain better performance. The effectiveness of gradient $\frac{\mathrm{d}}{\mathrm{d}\Lambda}R$ is then validated.

The comparison with BatchEnsemble shows that directly quantizing the rank-1 decomposed matrix in BatchEnsemble could be harmful to the performance. The reason might be that the quantization errors of the different components are accumulated by multiple production as each component $\mathbf{W}_k$ is obtained by $(\mathbf{s}_k^{\text{out}}\mathbf{s}_k^{\text{in}\,T}) \circ \mathbf{\Omega}$. Specifically, the quantization is over $[\mathbf{s}_k^{\text{out}}]_{k=1}^K$, $[\mathbf{s}_k^{\text{in}}]_{k=1}^K$ and $\mathbf{\Omega}$. Quantization of each vector or matrix would introduce errors which is then magnified by the process to obtain $\mathbf{W}$. It is meaningless to quantize the middle results $S_k = \mathbf{s}_k^{\text{out}}\mathbf{s}_k^{\text{in}\,T}$, as storing $[S_k]_{k=1}^K$ would increase the storage cost by K fold. Then, the rank-1 decomposition in BatchEnsemble has no effect in reducing the model size. Therefore, this validates the necessity of our design in BitsEnsemble, which provides high performance ensemble with a small size.

In terms of efficiency, compared with other schemes, the training time of BitsEnsemble is increased by at most 27.5% and 36% for the two networks. This value is acceptable as the bits-sharing scheme should be executed twice (for $\mathcal{E}_4$ and $\mathcal{E}_8$). At inference time, BitsEnsemble has similar performance with the others, as the structures for different ensembles are similar in inference time.

In terms of storage, the full precision results show that the decomposition in BitsEnsemble is useful in reducing the model size. By properly adjusting the latent dimension μ, it could reach a comparable model size as the rank-1 BatchEnsemble on VGG11. On ResNet18, the reduction of model size is from 2.43× to 3.6×.

The next sub-section shows that the model size of a deep ensemble could be further reduced by leveraging the clustering information provided by bits-sharing, provided by our encoding scheme.

C. Performance of encoding and decoding

To validate the performance of encoding/decoding with our bit-sharing scheme on real devices, we evaluate the encoding/decoding time and the encoded model size. The encoding time is measured on the server side while the decoding time is measured on the edge side, such as the Raspberry 4 device. We collect the encoding/decoding time of each layer in VGG11. For the encoded file size, we collect the results of ours, Full-rank ensemble (full-precision) and Full-rank ensemble (8 bit). Besides, we also collect the storage overhead of different kinds of bits.

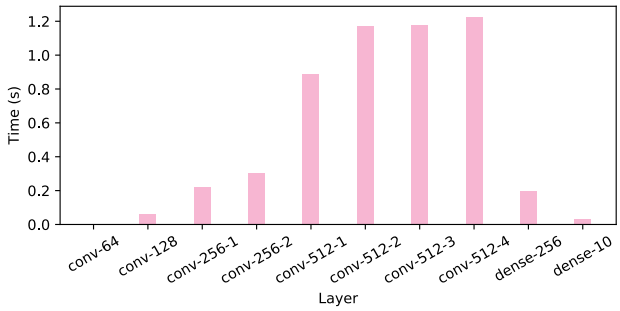

Fig. 6. The decoding time of each layer in VGG11 on the Raspberry 4 device.

In terms of decoding efficiency, BitsEnsemble only takes 5.25 seconds to decode the quantized VGG11 on the Raspberry 4 device. Figure 6 shows the decoding time of each layer in VGG11 on the edge device. The main decoding time is spent on the last four convolutional layers, due to the larger number of parameters of these layers. Besides, we can observe that the difference of decoding time between layers is less than about 1.2 seconds. Since the decoding of each layer is independent, the decoding efficiency can be further improved with a simple scheduling.

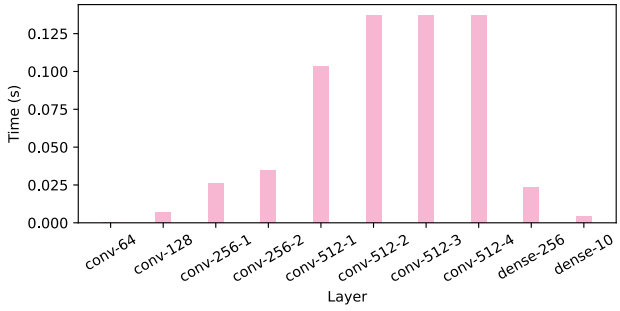

Fig. 7. The encoding time of each layer in VGG11 on the server.

Although the encoding stage is generally performed on the server side with powerful computation resources, we also

collect the total encoding time and the encoding time of each layer to show the encoding efficiency. The total encoding time of the quantized VGG11 on the server side is 0.61 seconds. Figure 7 shows the encoding time of each layer in VGG11 on the server side. The encoding time of layers follows the same rule that encoding those layers with larger number of parameters is more time-consuming.

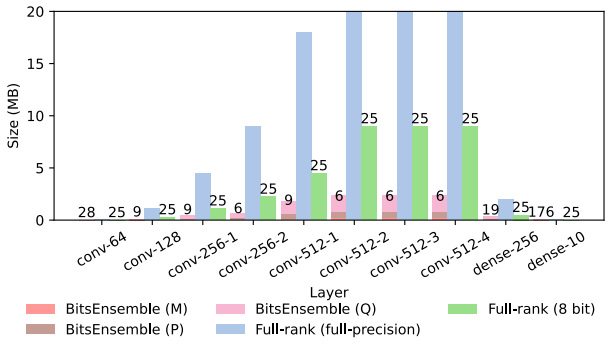

Fig. 8. The encoded size of each layer in VGG11. The numbers denotes the size of BitsEnsemble and Full rank (8 bit) in percentage (%) normalized by the absolute size of Full rank (full precision).

To verify the efficacy of the proposed encoding scheme in Sec. V, we also collect the encoded model size and the encoded layer size in VGG11. The total encoded model size of VGG11 is about *11 MB*, which is much smaller than the result (40 MB) in the Tab. IV. The reason is that the proposed encoding scheme only stores the valid bits of each variable. For sharing bits, the encoding scheme only stores once for saving more bits, while BitsEnsemble stores multiple copies for computation efficiency. During the decoding, those copies can also be restored based on the metadata and pattern bits. Compared to Full-rank (full-precision) and Full-rank (8 bit), our encoding scheme reduces the model size by 92.60% and 70.39%, respectively. This also mainly benefits from the bit-sharing scheme.

As shown in the Fig. 8, the numbers represent the ratio of the layer size compared to the Full-rank (full-precision). For the last three convolutional layers, the layer size of Full-rank (full-precision) is greater than 20MB. From the layer size results, our encoding scheme significantly reduces the layer size, especially for the layers with large number of parameters, e.g., the convolutional layer with channel size 512. For the first layer and the last layer, our encoding scheme produce more bits that the other comparisons. This is due to the larger overhead introduced in the encoding scheme. However, the layer size of the first layer and the last layer is quite small, e.g., only 7.63 KB for the first layer with our encoding scheme, thus the overhead of these two layers can be ignored.

BitsEnsemble (M), BitsEnsemble (P), BitsEnsemble (Q) in the Fig. 8 represents the corresponding number of metadata bits, pattern bits, and quantization bits. In our encoding scheme, the overhead of metadata bits and pattern bits are 10.02% and 23.61% of each layer size on average. Our encoding scheme adopts the fixed bit-width number to store the pattern bits, thus leading to a higher overhead than metadata bits. However, adding all overheads, our encoding scheme can still outperform than other schemes.

VII. Related work

We present the comparisons with BatchEnsemble and its related technique in this section. We refer the readers to Sec. I and Sec. II, for the related techniques for quantization and ensemble.

As illustrated in Sec. IV, BatchEnsemble [37] is a full precision ensemble. It fits an ensemble in one neural network by splitting multiple sub-networks as the members of an ensemble. The significant contribution of BatchEnsemble is the rank-1 structure introduced to the weight matrix. For one layer, the parameters contain $\{[\mathbf{s}_k^{\text{out}}]_{k=1}^K, [\mathbf{s}_k^{\text{in}}]_{k=1}^K, \mathbf{\Omega}\}$. A member k is associated with a pair of vectors $\{\mathbf{s}_k^{\text{out}}, \mathbf{s}_k^{\text{in}}\}$, which could be used to generate the rank-1 matrix $\mathbf{S}_k = \mathbf{s}_k^{\text{out}} \mathbf{s}_k^{\text{in}^T}$. $\mathbf{\Omega}$ is shared across all members. The weight matrix for member k is obtained by $\mathbf{W}_k = \mathbf{S}_k \circ \mathbf{\Omega}$.

As illustrated in Sec. VI-B, a regular quantizaton scheme is not suitable for quantizing BatchEnsemble. The reason is applying quantization to $\{\mathbf{s}_k^{\text{out}}, \mathbf{s}_k^{\text{in}}\}$ would cause quantization errors. The errors would be magnified during the complex process of obtaining $\mathbf{S}_k$ and $\mathbf{W}_k$, which seriously damages the performance of quantized ensemble under even standard quantization. Directly quantizing $\mathbf{S}_k$ might produce a high rank matrix to store, which introduce $K\times$ more storage consumption.

By comparison, BitsEnsemble provides a robust and simple $\mathbf{W} = \mathbf{U}\mathbf{V}^T$ decomposition of the matrix. The quantization errors on $\mathbf{U}$ and $\mathbf{V}$ won't be significantly magnified as only one tensor computation is used. Also, both $\mathbf{U}$ and $\mathbf{V}$ contain components from K members. Our bits-sharing technique could be fully leveraged for saving bits.

Dusenberry et al. [7] extends the BatchEnsemble by placing uncertainty over the weights to obtain a scalable Bayesian neural network. It would be interesting to study Bayesian counterpart of BitsEnsemble and its corresponding quantization scheme. Wen et al. [36] use data augmentation to enhance the performance of BatchEnsemble. Note that this is an orthogonal technique to this work, as the same tricks could be applied to BitsEnsemble for better performance.

VIII. Conclusion

In this paper, we propose to cluster the quantization residual errors among members within a deep ensemble to save bits. An analysis performed on real dataset shows, the clustering of quantization bits could significantly reduce the ensemble size. To obtain a decent clustering requires exhaustively searching for the best hyper-parameter Λ and repetitively executing the cluster algorithm for different parameters. Both are computationally prohibitive. We propose a generic bits-sharing scheme, for making the whole clustering algorithm differentiable and parallelizable. Differentiable bits-sharing could enable Λ to be learned from data, thus no searching is required. Parallelizable bits-sharing could leverage the modern computation library for high training efficiency. The bits-sharing scheme is universally applicable to ensemble with members in the same structure. We further design our ensemble model for pursuing a smaller model size. An encoding and decoding scheme is further proposed for realizing the virtually shared bits for storage.

REFERENCES

[1] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[2] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[3] Y. Cui, Z. Liu, Q. Li, A. B. Chan, and C. J. Xue. Bayesian nested neural networks for uncertainty calibration and adaptive compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2392–2401, 2021.

[4] Y. Cui, Z. Liu, W. Yao, Q. Li, A. B. Chan, T.-w. Kuo, and C. J. Xue. Fully nested neural network for adaptive compression and quantization. In *IJCAI*, pages 2080–2087, 2020.

[5] Y. Cui, W. Yao, Q. Li, A. B. Chan, and C. J. Xue. Accelerating monte carlo bayesian prediction via approximating predictive uncertainty over the simplex. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[6] D. Danks. How adversarial attacks could destabilize military ai systems. *IEEE Spectrum: Technology, Engineering, and Science News (https://spectrum. ieee. org/automaton/artificialintelligence/embedded-ai/adversarial-attacks-and-aisystems)*, 2020.

[7] M. Dusenberry, G. Jerfel, Y. Wen, Y. Ma, J. Snoek, K. Heller, B. Lakshminarayanan, and D. Tran. Efficient and scalable bayesian neural nets with rank-1 factors. In *International conference on machine learning*, pages 2782–2792. PMLR, 2020.

[8] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *nature*, 542(7639):115–118, 2017.

[9] S. Fort, H. Hu, and B. Lakshminarayanan. Deep ensembles: A loss landscape perspective. *arXiv preprint arXiv:1912.02757*, 2019.

[10] R. Frostig, M. J. Johnson, and C. Leary. Compiling machine learning programs via high-level tracing. *Systems for Machine Learning*, pages 23–24, 2018.

[11] H. Fu, C. Li, X. Liu, J. Gao, A. Celikyilmaz, and L. Carin. Cyclical annealing schedule: A simple approach to mitigating kl vanishing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 240–250, 2019.

[12] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[13] J. Gawlikowski, C. R. N. Tassi, M. Ali, J. Lee, M. Humt, J. Feng, A. Kruspe, R. Triebel, P. Jung, R. Roscher, et al. A survey of uncertainty in deep neural networks. *arXiv preprint arXiv:2107.03342*, 2021.

[14] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[15] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330. PMLR, 2017.

[16] L. Hou and J. T. Kwok. Loss-aware weight quantization of deep networks. In *International Conference on Learning Representations*, 2018.

[17] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*, 2017.

[18] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016.

[19] 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 *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[20] Q. Jin, L. Yang, and Z. Liao. Adabits: Neural network quantization with adaptive bit-widths. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2146–2156, 2020.

[21] A. Kendall and Y. Gal. What uncertainties do we need in bayesian deep learning for computer vision? *Advances in neural information processing systems*, 30, 2017.

[22] D. E. Knuth. Fundamental algorithms. 1973.

[23] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

[24] J. Liu, B. Zhuang, P. Chen, Y. Guo, C. Shen, J. Cai, and M. Tan. Lbs: Loss-aware bit sharing for automatic model compression. *arXiv preprint arXiv:2101.04935*, 2021.

[25] Z. Liu, C. Yufei, and A. B. Chan. Improve generalization and robustness of neural networks via weight scale shifting invariant regularizations. In *ICML 2021 Workshop on Adversarial Machine Learning*, 2021.

[26] W. J. Maddox, P. Izmailov, T. Garipov, D. P. Vetrov, and A. G. Wilson. A simple baseline for bayesian uncertainty in deep learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[27] Y. Ovadia, E. Fertig, J. Ren, Z. Nado, D. Sculley, S. Nowozin, J. Dillon, B. Lakshminarayanan, and J. Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. *Advances in neural information processing systems*, 32, 2019.

[28] T. Pang, K. Xu, C. Du, N. Chen, and J. Zhu. Improving adversarial robustness via promoting ensemble diversity. In *International Conference on Machine Learning*, pages 4970–4979. PMLR, 2019.

[29] S. Papi, E. Trentin, R. Gretter, M. Matassoni, and D. Falavigna. Mixtures of deep neural experts for automated speech scoring. In *INTERSPEECH*, 2020.

[30] E. Park, J. Ahn, and S. Yoo. Weighted-entropy-based quantization for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5456–5464, 2017.

[31] J. Quiñonero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence. *Dataset shift in machine learning*. Mit Press, 2008.

[32] M. Raghu, K. Blumer, R. Sayres, Z. Obermeyer, B. Kleinberg, S. Mullainathan, and J. Kleinberg. Direct uncertainty prediction for medical second opinions. In *International Conference on Machine Learning*, pages 5281–5290. PMLR, 2019.

[33] M. Rusci, A. Capotondi, and L. Benini. Memory-driven mixed low precision quantization for enabling deep network inference on microcontrollers. *Proceedings of Machine Learning and Systems*, 2:326–335, 2020.

[34] M. Van Baalen, C. Louizos, M. Nagel, R. A. Amjad, Y. Wang, T. Blankevoort, and M. Welling. Bayesian bits: Unifying quantization and pruning. *Advances in neural information processing systems*, 33:5741–5752, 2020.

[35] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8612–8620, 2019.

[36] Y. Wen, G. Jerfel, R. Muller, M. W. Dusenberry, J. Snoek, B. Lakshminarayanan, and D. Tran. Improving calibration of batchensemble with data augmentation. *TWorkshop on Uncertainty and Ro-Bustness in Deep Learning*, 2020.

[37] Y. Wen, D. Tran, and J. Ba. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. *arXiv preprint arXiv:2002.06715*, 2020.

[38] K. Xu, G. Zhang, S. Liu, Q. Fan, M. Sun, H. Chen, P.-Y. Chen, Y. Wang, and X. Lin. Adversarial t-shirt! evading person detectors in a physical world. In *European conference on computer vision*, pages 665–681. Springer, 2020.

[39] H. Yang, J. Zhang, H. Dong, N. Inkawhich, A. Gardner, A. Touchet, W. Wilkes, H. Berry, and H. Li. Dverge: diversifying vulnerabilities for enhanced robust generation of ensembles. *Advances in Neural Information Processing Systems*, 33:5505–5515, 2020.

[40] S. Zaidi, A. Zela, T. Elsken, C. C. Holmes, F. Hutter, and Y. Teh. Neural ensemble search for uncertainty estimation and dataset shift. *Advances in Neural Information Processing Systems*, 34, 2021.