LATTICE QUANTIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Low bit quantization of weights in increasingly large deep convolutional neural networks (DCNNs) can be critical for their implementation in memory constrained hardware systems. Post-training quantization consists in quantizing a model without retraining, which is user-friendly, fast and data frugal. In this paper, we propose LatticeQ, a new post-training weight quantization method designed for DCNNs. Instead of the standard scalar rounding widely used in state-of-the-art quantization methods, LatticeQ uses a quantizer based on lattices - discrete algebraic structures - which we show are able to exploit the inner correlations between the model parameters. LatticeQ allows us to achieve state-of-the-art results in post-training quantization, enabling us to approach full precision accuracies for bitwidths previously not accessible to post-training quantization methods in similar experimental settings. In particular, we achieve ImageNet classification results close to full precision on the popular Resnet-18/50, with only 1% and 3% accuracy drop for the 4-bit weights and 3-bit weights model architectures respectively.

1 INTRODUCTION

Complex tasks such as image recognition on large datasets (Krizhevsky et al., 2017; He et al., 2015) and speech recognition (Hinton et al., 2012) are very efficiently solved by deep convolutional neural networks. To improve precision, deep learning models have become continually larger, more memory demanding and computationally heavier. Popular models like ResNet (He et al., 2015) use millions of parameters (Resnet-50 uses 25Mio of them, which represents a storage of almost 100MB of data). However, for embedded systems applications, storing such quantities of information is often impossible in practice. Some architectures are specifically designed to reduce this memory footprint like MobileNet (Howard et al., 2017; Sandler et al., 2018) (Mobilenet-v2 requires just 14MB of memory). But compressing existing architectures is easier than inventing new ones for specific needs. Therefore, various methods were designed to improve memory efficiency and reduce computational complexity of these models, in order to be able to use DCNNs for low power and/or low memory applications.

There are two major ways of reducing the size of a deep neural network :

- Reducing the total number of parameters, through pruning (Blalock et al., 2020), weight sharing, low-rank factorization.
- Reducing the memory footprint of its parameters and operation complexity, mainly through quantization (Krishnamoorthi, 2018).

Recently, quantization of popular models down to 4 and even 3-bit has been made possible, with little to no degradation of task performance (Esser et al., 2020; Jin et al., 2019). These breakthroughs often rely on retraining the model from scratch with stochastic gradient descent, using the full training dataset. This class of methods is known as quantization-aware training, and leads to the best performance in DCNNs quantization. However, using these methods can sometimes be impractical. The first drawback is the need for the full dataset. In some cases we only have access to a trained full precision network, and for confidentiality reasons, it may be impossible to access the full training dataset of this network. A second drawback is their practicality: training a quantized network from scratch can be extremely demanding in computation resources, last generation GPUs, which might not be available for the developer.

To overcome these constraints, several methods have been suggested that do not rely on retraining a network from scratch. This framework is called post-training quantization. It is commonly acknowledged that post-training methods do not lead to task performance as good as quantization aware training (Krishnamoorthi, 2018), especially when dealing with bitwidths 4 or lower, but they do solve the aforementioned problems, and can be critical for rapid deployment.

Inspired by Nagel et al. (2019), we propose a classification of quantization methods given their level of complexity and data requirements, which extends their own classification. We will use this framework to help the reader compare our method with those of other authors.

- **Level 1a** No data and no finetuning of the network's parameters. "As simple as an API call". The required inputs are only the model definition and its weights. (Nagel et al., 2019).
- **Level 1b** No data and no finetuning of the network's parameters. Limited data can be used to calibrate activation quantization thresholds. Per-channel quantization is widely used. (Banner et al., 2019; Zhao et al., 2019).
- **Level 2a** Some samples of application data are needed, no finetuning of the network's parameters. Data determines the quantization function (i.e. step, threshold, channel splitting, mixed precision), or updates batch normalization statistics. Per-channel and per-layer quantization coexist in the state of the art. (Choukroun et al., 2019; Liu et al., 2021).
- **Level 2b** Some samples of application data are needed, restricted finetuning pipeline for the network's parameters. Data is used for layerwise optimization (Nagel et al., 2020; Wang et al., 2020) or blockwise optimization (Li et al., 2021) to improve quantization locally. From this level on, per-layer quantization is standard.
- **Level 3** Quantization aware training. Requires the full dataset and parameter optimization, needs finetuning and hyperparameter tuning to achieve good performance.

In this paper, we introduce a new level 1b post-training quantization technique for deep convolutional neural networks, which achieves level 1b state-of-the-art classification performance on ImageNet down to 4-bit weights on Resnet architectures, with less than 1% accuracy drop compared to full precision models. We also show good performance for 3-bit quantization with 3% accuracy drop compared to full precision. Our method relies on a new quantizer that uses linear correlations between the parameters of convolution layers to minimize its error. The goal of this paper is not to show absolute best performance among all post-training quantization methods, but rather to show some improvement over the scalar quantizer is a standard and simple experimental setting.

Our contributions

1. We introduce a new quantizer, based on the structure of lattices, to allow for more flexible and adaptive quantization of weights, contrary to most other techniques that rely on uniform quantizers.
2. We explore the impact of combining parameter and activation quantization for our quantizer, to allow for faster inference and low complexity computation.
3. We provide insights and analysis of how our quantizer works, and its memory overhead.

This paper is organized as follows: section 2 presents previous work on post-training quantization as well as quantization-aware training. Section 3 analyzes the parameter correlations inside DCNNs and introduces the intuition behind our approach. In section 4, we detail our approach and compare it to existing state-of-the art methods. In section 5, we give additional results of our quantizer without enhancements as an ablation study. Finally, in section 6, we provide analysis of the characteristics of our quantizer, including its quantization error, its distribution, and its memory overhead.

2 RELATED WORK

The performance of DCNN quantization schemes depends heavily on hypotheses: availability of training data, computation time and hardware capacities. The best results are currently achieved by quantization-aware training, which is the least restrictive: unlimited access to training data, unlimited computation time and resources, heavy hyperparameter finetuning and human expertise. Recently, promising results in this area have been achieved by Esser et al. (2020), Jin et al. (2019), which both used uniform quantizers for weights and activations. Non uniform quantizers, including vector quantizers relying on clustering techniques, have also been successfully employed (Han et al., 2016; Stock et al., 2020).

Post-training quantization meets stricter specifications than quantization-aware training. Nagel et al. (2019) for example assumed that no data is available at all for quantization. Banner et al. (2019) introduced bias correction and per-channel bit allocation. Choukroun et al. (2019) specifically designed a quantizer to minimize the MSE loss of the quantization operation. Other approaches that use a few samples of data have been proposed (Nagel et al., 2020), which used data in order to learn whether to round the weights up or down. Wang et al. (2020) proceeded by bit optimization, and Liu et al. (2021) exploited multipoint quantization to approximate full precision weight vectors with a linear combination of low bit vectors.

3 PRELIMINARY OBSERVATIONS

Most post-training schemes try to optimize task performance by adapting their quantizer to better fit the typically bell-shaped density function of the model weights. Moreover, these scalar methods do not pay attention to potential correlations between these parameters. As can be seen on Figure 1, correlations may exist between the weight values of trained DCNNs.

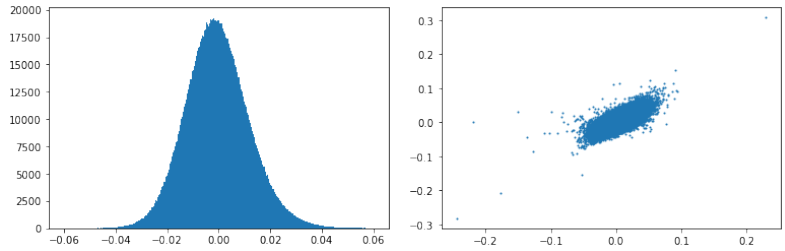

Figure 1: Left : Weight scalar distribution in layer 4.0 conv2 of an ImageNet pretrained Resnet50. Right : 2D plot of (w_1, w_2) in the same layer, where w_i is the ith coordinate of filter w.

Figure 2 is the correlation matrix of the 9 distributions of filter parameters in a 3×3 convolution layer. The first distribution is the one of weights located in the upper left corner of a filter, the second distribution is the one of weights located in the upper center of a filter, etc. Formally, we plot in line i and column j the points (w_i, w_j) for each filter $f = (w_1, ..., w_9)$ in a chosen layer. On the long diagonal, we plot the histogram of w_i. We notice that filters tend to have correlated coordinates, with quite high correlation coefficients. The same observation can be made in other 3×3 layers. From this observation, we justify the main assumption of our method: a quantizer "shaped as a parallelogram" is more data efficient than a uniform quantizer "shaped as a square".

4 METHODS

Many state-of-the-art quantization methods rely on scalar uniform quantizers, which require only a few additional parameters: step size, bitwidth, and/or threshold. Other approaches use vector quantization, which may either be scalar or multidimensional, leading to more flexible quantization sets (Han et al., 2016; Stock et al., 2020) but are often limited by the need to use codebooks. LatticeQ takes the best of both approaches. It adds a limited number of additional parameters to encode quantization bases, and offers a broader variety of possible quantization sets than scalar uniform quantizers.

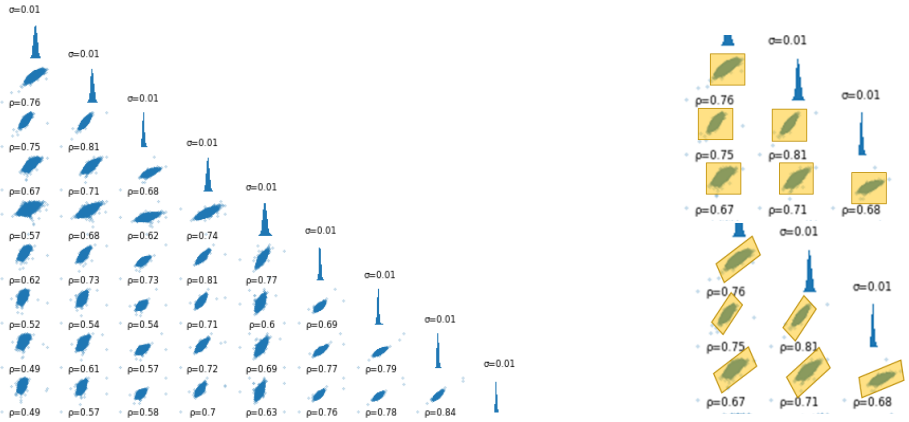

Figure 2: Left : Correlation diagram of filters in layer 4.0 conv2 (Conv2d 3×3). Right : Uniform (= square) quantization and lattice (= parallelogram) quantization

4.1 LATTICE BASED WEIGHT QUANTIZER

In order to quantize the weights, LatticeQ uses lattices, which are algebraic structures that discretize the notion of vector space (see APPENDIX A for more details). Each lattice has a basis, meaning that each point of the lattice can be written as an integer linear combination of the vectors of this basis. This integer linear combination is the encoding of our quantization. A lattice has infinite cardinality. In order to use this structure as our quantizer we need a finite number of quantization points. We truncate our lattice in the following fashion : let Λ be a lattice, and $\mathcal{B} = (\mathbf{b_i})_{1 \leq i \leq n} \in \mathbb{R}^n$ a basis of Λ. Given b the bitwidth, our quantization set is $Q = \{q \in \mathbb{R}^n, q = \sum_{i=1}^{n} \mu_i \mathbf{b_i},\ \forall i \in \{1, ..., n\},\ -2^{b-1} \leq \mu_i \leq 2^{b-1} - 1\}$.

In memory, each quantized block is represented by its coordinates in the quantization basis. Let us suppose $\mathcal{B} = (\mathbf{b_1}, \mathbf{b_2}, \mathbf{b_3}) = ((b_{1,1}, b_{1,2}, b_{1,3}), (b_{2,1}, b_{2,2}, b_{2,3}), (b_{3,1}, b_{3,2}, b_{3,3}))$ is our quantization basis (with each $b_{i,j}$ a scalar, possibly quantized with a uniform min/max quantizer), and:

$$F^q = \begin{pmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{pmatrix}$$

is a 3×3 quantized filter. It consists in 3 quantized blocks: (f_1, f_2, f_3), (f_4, f_5, f_6) and (f_7, f_8, f_9). The convolution represented by F^q is the concatenation of three 1×3 vectors:

$$F = Dequant(F^q) = \begin{pmatrix} (f_1\mathbf{b_1} + f_2\mathbf{b_2} + f_3\mathbf{b_3}) \\ (f_4\mathbf{b_1} + f_5\mathbf{b_2} + f_6\mathbf{b_3}) \\ (f_7\mathbf{b_1} + f_8\mathbf{b_2} + f_9\mathbf{b_3}) \end{pmatrix}$$

Note that if we choose $\mathcal{B}$ as a uniform scaling of an orthonormal basis ($\mathcal{B} = \lambda \times I_n$), the quantization set is exactly the one of classic scalar quantization (we call this simplified version "Cubic LatticeQ").

The quantization process for a 3×3 layer is simple: we flatten the weights and group them by blocks of 3. Then, using the quantization basis, we search for the quantization point nearest to this block. The vector found is the quantization point for this block. In practice, we can use $Dequant$ to calculate the convolution in the quantized network. However, for per-layer quantization, we propose an optimized way of dealing with the inference in a LatticeQ-quantized network in APPENDIX E.

Quantizing literally means defining a function from an infinite set to a finite set. In the scalar model, the quantization operation relies on a simple round function. In order to quantize on a lattice, we need to use a more complex algorithm. Actually, the problem of finding the closest lattice point to a real vector is known to be NP-hard (Micciancio, 1998), and is called Closest Vector Problem (CVP):

"Given $x \in \mathbb{R}^n$ and Λ a lattice of $\mathbb{R}^n$, find $\lambda \in \Lambda$ such that $d(x, \lambda) = min\{d(x, l),\ l \in \Lambda\}$."

Algorithm 1 Nearest plane algorithm

function BABAI(Basis $\mathcal{B}$, Vector t)
  $\mathcal{B}^* = GramSchmidt(\mathcal{B})$
  $b \leftarrow t$
  for $j \in \{n, ..., 1\}$ **do**
    $u_j \leftarrow \frac{<b, b_j^*>}{<b_j^*, b_j^*>}$
    $b \leftarrow b - \lfloor u_j \rceil b_j$
  end for
  return $x = \sum_{j=1}^{n} \lfloor u_j \rceil b_j = t - b$
end function

Algorithm 2 Gram-Schmidt algorithm

function GRAMSCHMIDT(Basis $\mathcal{B}$)
  $b_1^* \leftarrow b_1$
  for $j \in \{2, ..., n\}$ **do**
    $b_j^* \leftarrow b_j - \sum_{i=1}^{j-1} < b_j, b_i^* > b_i^*$
  end for
  return $\mathcal{B}^* = (b_j^*)_{1 \leq j \leq n}$
end function

In order to solve the closest vector problem, we use the nearest plane algorithm (Babai, 1986). It is fairly easy to understand and implement, computationally light, and still provides good approximations in low dimension. We detail the implementation in algorithms 1 and 2.

Now that we have an algorithm to quantize on a lattice, we want to find a good lattice. For the data free approach, we opt for a simple random search with restart as described in algorithm 3. Restarts simply consist in running the algorithm several times in a row and keeping the best result of all the runs. We look for a lattice that reduces the mean cube error loss (MCE) between the full precision weights of the layer (or channel), and their quantized version. We chose this loss rather than MSE because it performed slightly better (Table 1). We assume this is because the MCE loss gives more importance to larger weights that are far away from any quantization point compared to the MSE.

Algorithm 3 FindBasis

1: **function** FINDBASIS(Weight tensor W, Basis dimension dim, Bitwidth $bits$, Temperatures T)
2: $\mathcal{B} \leftarrow I_{dim}$
3: **for** $0 \leq s \leq |T|$ **do**
4: $\mathcal{B}' \leftarrow \mathcal{B}' + Sample(\mathcal{G}(0, \sigma = T(s), |\mathcal{B}|))$
5: **if** $MCELoss(W, W^q_{\Lambda_{\mathcal{B}'}}) < MCELoss(W, W^q_{\Lambda_{\mathcal{B}}})$ **then**
6: $\mathcal{B} \leftarrow \mathcal{B}'$
7: **end if**
8: **end for**
9: **return** $\mathcal{B}$
10: **end function**

The random transformation we use is a multivariate gaussian noise addition with standard deviation $T(s)$, where T is a sequence of temperature steps, that can be adjusted as a hyperparameter, see APPENDIX B. In order to perform low bit computations, we may want to quantize the basis itself to integer values, which we did in all our experiments, by adding a simple min/max quantizer for $\mathcal{B}'$ between line 4 and line 5 in algorithm 3.

Since the approximation factor of the nearest plane algorithm grows in square root in the basis dimension n (Babai, 1986), there is no interest in choosing high dimensional bases. Moreover, it increases the storage requirements, since an n-dimensional basis requires n^2 entries in memory. Experimentally, we notice that the most accurate dimension for the quantization basis is 3 for 3×3 layers (Table 2), and 2 for 1×1 layers.

Table 1: Comparison between MSE loss and MCE loss for Resnet-18 with weights quantized to 4-bit.

	Top-1 accuracy	
Network	MSELoss	MCELoss
Resnet-18	67.85	**67.94**

Table 2: Impact of the choice of the basis dimension in 3×3 layers on final accuracy with weights quantized to 4-bit.

	Top-1 accuracy	
Network	$dim = 3$	$dim = 9$
Resnet-18	**67.94**	61.90

4.2 ACTIVATION QUANTIZATION

As typically done in post-training scenario, we chose to perform activation quantization on one batch of data right before testing the quantized network. In real life, this can be done right before deployment on one batch of application data. We can either quantize activations per layer or per channel. In case we quantize the activations in a layer-wise fashion, the performance drops significantly below the 8-bit setting. In case we quantize the activations in a channel-wise fashion, it is possible to reach an activation precision down to 4-bit without much loss of accuracy by using a quantile trick similar to McKinstry et al. (2019). Let C be an activation channel to quantize. Let α_C be the quantile 0.9997 of C's distribution, which means $p_{c \in C}(c \leq \alpha_C) = 0.9997$. We estimate α_C by running a forward pass with one batch of data. Then we use α_C as our activation threshold for the channel C. We chose different quantiles depending on the network and activation bitwidths, see APPENDIX C.

4.3 EXPERIMENTAL SETUP

We evaluate our method on the ImageNet (Russakovsky et al., 2015) classification task. All experiments are made using PyTorch (Paszke et al., 2019), and the pretrained models used all come from the torchvision.models library. The basis is always quantized with the same bitwitdh as the activations. Based on experiments, we set bases dimension to 3 for 3×3 convolution layers, 2 for 1×1 layers and linear layers, and 1 for the first layer (since we did not notice any advantage in increasing the dimension for this particular layer). To quantize activations, we use a calibration batch of 512 images. As it is common practice in such applications, we quantize the pooling layers, first layer and last layer to 8-bit, and we do not quantize layer biases (only VGG (Simonyan & Zisserman, 2015) has biases). We report bitwidths settings and top-1 accuracy for each model tested, and we also provide the results from Banner et al. (2019) and Choukroun et al. (2019) for comparison. In part 4.4, we report the results of per-channel LatticeQ with data free enhancements in Table 3. In part 5, we report the results of per-channel LatticeQ without bias correction (refered to as our baseline) in Table 4 and the results of per-layer LatticeQ without bias correction in Table 5. Finally, in APPENDIX F, we report the results of weights-only quantization.

4.4 DATA FREE PER-CHANNEL ENHANCEMENT

Table 3: LatticeQ with bias correction on ImageNet. For 8-bits activations we use naive per-layer activation quantization. For 4-bit activations, we use our quantile per-channel approach. For weights, we use per-channel quantization and bias correction. We compare our results with results that we generated from the source code of Banner et al. (2019), using per-channel quantization and bias correction. We also compare with paper results from OMSE (Choukroun et al., 2019).

			Top-1 accuracy				
Network	Method	FP32	W4A8	W4A4	W3A8	W3A3	W2A8
Resnet-18	LatticeQ (Ours)	69.6	**68.9**	**67.3**	**66.2**	**56.9**	**38.5**
	Banner et al.	69.6	67.4	64.3	43.4	28.6	1.3
	OMSE+opt[1]	69.6	67.1				
Resnet-50	LatticeQ (Ours)	76.0	**75.3**	**70.9**	**73.1**	**55.0**	**45.6**
	Banner et al.	76.0	74.8	70.3	67.5	38.4	0.4
	OMSE+opt[1]	76.0	74.7				
VGG-16bn	LatticeQ (Ours)	73.4	**73.0**	**70.6**	**70.8**	**61.9**	**35.8**
	Banner et al.	73.4	71.6	70.4	65.9	59.7	0.1
	OMSE+opt[1]	73.4	72.3				
Densenet	LatticeQ (Ours)	74.4	71.5	**69.5**	**65.9**	**54.2**	**9.6**
	Banner et al.	74.4	70.2	63.0	53.8	18.7	0.4
	OMSE+opt[1]	74.4	**71.7**				
Mobilenet-v2	LatticeQ (Ours)	71.9	**66.5**	**46.0**	**46.9**	**0.3**	**0.3**
	Banner et al.	71.9	61.1	36.3	10.2	**0.3**	0.1

For this part, we rely on the work of Banner et al. (2019). First, we use per-channel quantization, to allow for more accurate quantization in each channel. We then add bias correction. Quantization operations tend to alter the moments (mean and variance) of the weight distribution in each channel, which is taken into account by bias correction. In our case, bias correction is also computed during the execution of $FindBasis$ for bitwidths lower than 4, so that the quantization basis found takes bias correction into account. Line 5 of $FindBasis$ is, in this case, replaced by :

if $MCELoss(W, biascorrection(W^q_{\Lambda_{\mathcal{B}'}})) < MCELoss(W, biascorrection(W^q_{\Lambda_{\mathcal{B}}}))$ **then**
  $\mathcal{B} \leftarrow \mathcal{B}'$
end if

The results in Table 3 show the substantial improvement of LatticeQ over state-of-the-art approaches. Our method outperforms Banner et al. (2019) with similar hypotheses, and even the level 2a OMSE+opt method, in almost all settings for all presented models.

As we read the Table 3 from left to right, quantization is more and more aggressive. We find that LatticeQ manages to stay within 1% of full precision model accuracy on Resnets and VGG in 4-bit weights and 8-bit activation, and within 3% in 3-bit weights and 8-bit activation. As expected, compact models like Densenet (Huang et al., 2018) and Mobilenet-V2 (Sandler et al., 2018) are less resilient to quantization than Resnets (He et al., 2015) and VGG (Simonyan & Zisserman, 2015). It is noticeable that our method reaches a top-1 accuracy of 54.2% for Densenet in 3-bit weights and 3-bit activation setting while the method proposed by Banner et al. (2019) only achieved 18.7%.

5 Ablation study

In this section, we present an ablation study which objective is twofold. First, we provide the results of our per-channel quantizer without bias correction (Table 4). It is noticeable that our results remain close to full precision accuracy on Resnets and VGG (within 2% in per-channel W4A8, and within 10% in W3A8 for instance). Bias correction increases our method's accuracy, but our baseline significantly outperforms baselines of other scalar quantization methods (Banner et al., 2019; Choukroun et al., 2019). Based on those results, we want to emphasize that lattice quantization is indeed the cornerstone of the quantization process.

Table 4: LatticeQ per-channel baseline quantization of weights and activations.

	Top-1 accuracy				
Network	FP32	W4A8	W4A4	W3A8	W3A3
Resnet-18	69.6	67.2	66.0	59.6	40.2
Resnet-50	76.0	74.6	70.6	69.3	47.6
VGG-16bn	73.4	72.4	70.4	64.3	57.0
Densenet	74.4	70.5	66.5	54.9	33.9
Mobilenet-v2	71.9	58.0	31.7	21.5	0.1

Next, we provide the results of our per-layer quantizer (Table 5). Quantizing per layer in W8A8 does not impact performance on Resnets. Although it remains challenging to quantize weights per layer under our assumptions (no calibration data for weights), we show promising accuracy that lays the foundations of potential level 2 applications.

[1]OMSE+opt (Choukroun et al., 2019), uses calibration data to setup its quantizer (level 2a).

Table 5: LatticeQ per-layer baseline quantization of weights and activations.

	Top-1 accuracy			
Network	FP32	W8A8	W4A8	W3A8
Resnet-18	69.6	69.5	59.5	23.8
Resnet-50	76.0	75.9	70.2	41.9
VGG-16bn	73.4	73.3	68.5	42.5
Densenet	74.4	71.3	59.4	11.4
Mobilenet-v2	71.9	70.6	12.9	0.2

6 ANALYSIS

6.1 LATTICE QUANTIZATION VERSUS CUBIC QUANTIZATION

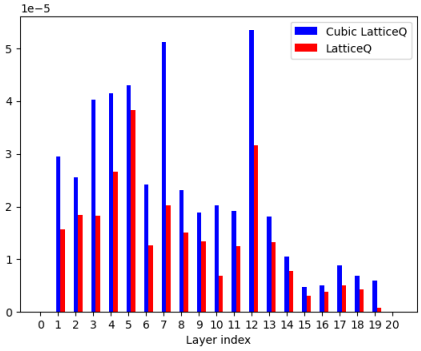

Figure 3: Resnet18 per-layer quantization error comparison between LatticeQ and Cubic LatticeQ (scalar quantization). Vertical axis is MCE.

Table 6: Comparison between baseline per-channel LatticeQ and baseline per-channel Cubic LatticeQ (scalar quantization).

Network	Method	FP32	W4A8
Resnet-18	LatticeQ	69.6	**67.2**
	Cubic LatticeQ	69.6	57.6

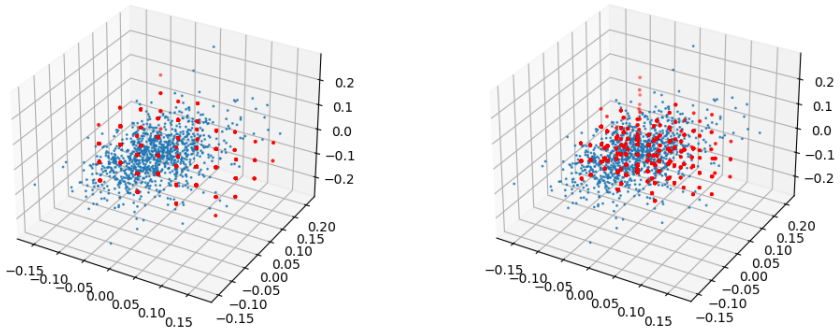

Figure 4: Left : Cubic LatticeQ quantization points (red) and 1×3 filter blocks (blue), Right : LatticeQ quantization points (red) and 1×3 filter blocks (blue). *These are the 2-bit quantization points for visualization.*

We have demonstrated experimentally the advantages, both in quantization error - as can be seen in Figure 3 - and task loss (Table 6), in using a deformable lattice for quantization rather than a cubic lattice (which is equivalent to uniform scalar quantization). This confirms our hypothesis that the inner correlations of the parameters of a neural network can be exploited for the purpose of quantization. We also show how the distribution of the quantization points indeed fits the multidimensional distribution of the network's parameters thanks to our method (Figure 4). On this figure, each full precision filter block is represented by a blue dot in the 3D space, and each quantization point of our method is represented by a red dot in the 3D space. LatticeQ increases the concentration of quantization points in the most critical areas of the filters' multidimensional distribution.

6.2 Memory overhead

In this section, we compute the memory overhead due to using our method, both in per-channel and per-layer settings. Each time a basis is used to quantize either a channel or a layer, we need to store $32 + n^2.b$ bits where n is the dimension of the basis and b is the number of quantization bits used for basis elements. 32 comes from the scaling factor. See Table 7 for a few examples among the networks we experimented with in this paper. We report the compression ratios of the full models. The compression rate penalty due to quantization bases is negligible in the per-layer setting, and always less than 1% in the per-channel setting. The compression that can be achieved by descending to lower bitwidths therefore largely offsets the memory overhead.

Table 7: Baseline LatticeQ memory cost. Scalar compression rate is the compresion rate of a scalar quantization method with the same bitwidth, such as Cubic LatticeQ.

Type	Network	W4A8 memory total	Compression rate	Scalar comp. rate
Per layer	Resnet-18	6.100 MB	13.06%	13.06%
	Resnet-50	13.78 MB	13.51%	13.51%
	Densenet	4.465 MB	14.14%	14.14%
	Mobilenet-v2	2.376 MB	17.12%	17.12%
Per channel	Resnet-18	6.158 MB	13.18%	13.10%
	Resnet-50	14.01 MB	13.74%	13.61%
	Densenet	4.555 MB	14.43%	14.27%
	Mobilenet-v2	2.547 MB	18.35%	17.60 %

7 Discussion and future work

In this paper, we introduced LatticeQ, a new post-training method which exploits the flexibility of lattice quantizers for the purpose of DCNN quantization. LatticeQ is particularly useful in cases where we want to deploy deep learning models trained in floating point precision on lightweight architectures without requiring a single training sample (which could happen for confidentiality, safety reasons, or for the sake of simplicity). We showed that our quantizer significantly outperforms the scalar quantizer for 3-bit quantization on several well-known architectures, and by up to 20% on Resnet-18, with less than 1% additional memory costs. LatticeQ does not require any finetuning or hyperparameter optimization, which makes it simple to use in practice.

Since lattice quantizers are a generalization of uniform quantizers, every uniform quantization method has a lattice extension. Therefore, we believe that lattice quantizers could have a high potential under other experimental hypotheses, like quantization-aware training. Using limited calibration data in order to perform parameter optimization after quantization could also improve performance, which we leave for future work.

8 REPRODUCIBILITY STATEMENT

We want to make sure our results are reproducible. As suggested in the author guide, we will make a comment directed to the reviewers and area chairs and put a link to an anonymous repository to submit our code. Due to the randomness of our optimization strategy and the choice of calibration data, results may slightly vary. We mitigated this issue by adding restarts to our random search.

REFERENCES

L Babai. Nearest lattice point problem. 1986.

Ron Banner, Yury Nahshan, Elad Hoffer, and Daniel Soudry. Post-training 4-bit quantization of convolution networks for rapid-deployment. *NeurIPS*, 2019.

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the State of Neural Network Pruning? 2020.

Yoni Choukroun, Eli Kravchik, Fan Yang, and Pavel Kisilev. Low-bit Quantization of Neural Networks for Efficient Inference. *arXiv:1902.06822*, 2019.

Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned Step Size Quantization. *arXiv:1902.08153*, 2020.

Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv:1510.00149*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385*, 2015.

Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*, 2012.

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861*, 2017.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. *arXiv:1608.06993*, 2018.

Qing Jin, Linjie Yang, and Zhenyu Liao. Towards Efficient Training for Neural Network Quantization. *arXiv:1912.10207*, 2019.

Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv:1806.08342*, 2018.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 2017.

Lenstra. Lattices. In *Algorithmic number theory*, volume 44 of *MSRI publications*. 2008.

Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. BRECQ: Pushing the Limit of Post-Training Quantization by Block Reconstruction. *ICLR*, 2021.

Xingchao Liu, Mao Ye, Dengyong Zhou, and Qiang Liu. Post-training Quantization with Multiple Points: Mixed Precision without Mixed Precision. *AAAI*, 2021.

Jeffrey L. McKinstry, Steven K. Esser, Rathinakumar Appuswamy, Deepika Bablani, John V. Arthur, Izzet B. Yildiz, and Dharmendra S. Modha. Discovering Low-Precision Networks Close to Full-Precision Networks for Efficient Embedded Inference. *arXiv:1809.04191*, 2019.

Daniele Micciancio. On the Hardness of the Shortest Vector Problem. 1998.

Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-Free Quantization Through Weight Equalization and Bias Correction. *ICCV*, 2019.

Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? Adaptive rounding for post-training quantization. PMLR, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *NeurIPS*, 2019.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *arXiv:1409.0575*, 2015.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. IEEE, 2018.

Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556*, 2015.

Pierre Stock, Armand Joulin, Remi Gribonval, Benjamin Graham, and Herve Jegou. And the bit goes down: Revisiting the quantization of neural networks. *ICLR*, 2020.

Peisong Wang, Qiang Chen, Xiangyu He, and Jian Cheng. Towards Accurate Post-training Network Quantization via Bit-Split and Stitching. *ICML*, 2020.

Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Christopher De Sa, and Zhiru Zhang. Improving Neural Network Quantization without Retraining using Outlier Channel Splitting. *PMLR*, 2019.

APPENDIX A

Lattices

The following part is a very synthetic introduction to the (very rich) theory of lattices. It states only what the reader needs to know to understand how our quantization method works. If the reader wants to know more about lattices, they are welcome to start with Lenstra (2008).

Definition 1

A lattice Λ of $\mathbb{R}^n$ is a discrete additive subgroup of $\mathbb{R}^n$, such that $span(\Lambda) = \mathbb{R}^n$, where $span(\Lambda)$ is the set of linear combinations of the vectors of Λ.

Example

$\mathbb{Z}^n$ is a lattice. It is called the cubic lattice.

Property 1

Let Λ be a lattice of $\mathbb{R}^n$, there is a sequence $\mathcal{B}$ of cardinality n of lattice vectors such that any vector that belongs to Λ can be uniquely expressed as an integer linear combination of the elements of $\mathcal{B}$. Such a sequence is called basis of Λ.

Examples

- $((1,0),(0,1))$ is a basis of $\mathbb{Z}^2$
- $((1,0),(1,1))$ is a basis of $\mathbb{Z}^2$
- $(\delta_{i,i})_{1\leq i\leq n}$ is a basis of $\mathbb{Z}^n$

Observation

The knowledge of a basis of a lattice Λ is sufficient to know every point of the lattice. Given that we want to use this type of structure to quantize a weight distribution, this observation allows us to avoid keeping the whole lattice stored as a codebook in memory.

Important observation

We shall emphasize that the uniform scalar quantization scheme widely used for neural networks quantization is nothing else but a cubic lattice quantization scheme. Therefore, lattice quantization is nothing else but a generalization of scalar techniques.

APPENDIX B

Heuristic details

This paragraph is dedicated to providing the details of our heuristic FindBasis (algorithm 3). Let $sc = \frac{1}{2^{b-1}}$ where b is the bitwidth of the channel, or layer, to be quantized. Basis $\mathcal{B}$ is initialized as $\frac{sc}{10^4} * I_n$. T consists in a sequence of 800 steps at each of these deviation values : $[\frac{sc}{10^4}, sc, \frac{sc}{2}, \frac{sc}{3}, \frac{sc}{5}, \frac{sc}{7}, \frac{sc}{9}, \frac{sc}{15}, \frac{sc}{30}]$. We apply this algorithm with 5 restarts and keep the best basis at each restart. Note that it is possible to reduce the number of steps and hence shrink computation time.

APPENDIX C

Quantile tables

Table 8: Quantiles chosen for Resnet and Densenet activation quantization.

Bitwidth	1	2	3	4	5	6	7	8
Quantile	0.97	0.992	0.9991	0.9997	0.9998	0.99995	0.99999	1

Table 9: Quantiles chosen for VGG activation quantization.

Bitwidth	3	4	8
Quantile	0.9999	0.9999	1

Table 10: Quantiles chosen for Mobilenet activation quantization.

Bitwidth	3	4	8
Quantile	0.986	0.998	1

APPENDIX D

Intuition on vector quantization

We know for a fact that correlated distributions share mutual information, in the sense of Shannon information theory. Shannon's mutual information between continuous random variables X and Y is calculated as a double integral :

$$I(X;Y) = \int_{\mathcal{Y}} \int_{\mathcal{X}} p(X=x, Y=y) log_2(\frac{p(X=x, Y=y)}{p(X=x)p(Y=y)})\, dx\, dy \tag{1}$$

If X and Y are independant, $I(X;Y) = 0$, they share no mutual information. When $X = Y$, $I(X;Y) = H(X) = H(Y)$. Now, we model the pairs of correlated weights of the layer we want to quantize with (w_1, w_2) which follows a bivariate normal distribution with location $\mu = (0, 0)$ and covariance matrix :

$$\Sigma = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix} \tag{2}$$

where σ_1, σ_2 are the standard deviations of w_1 and w_2 and ρ their correlation coefficient. This yields:

$$I(w_1; w_2) = -\frac{1}{2} log_2(1 - \rho^2) \tag{3}$$

Therefore, the more w_1 and w_2 are correlated, the more information they share, which means that an independent coding of w_1 and w_2 scalars is suboptimal, because of information redundancy. This is the idea behind our approach and behind vector quantization in general.

APPENDIX E

Optimized inference for LatticeQ networks

In this section, we suppose that we have a 3×3 convolution layer Λ quantized using our per-layer method, with in input channels and out output channels. We want to compute a forward pass through this layer. Let $(C_i^{in})_{1 \leq i \leq in}$ the set of input channels and $(C_j^{out})_{1 \leq j \leq out}$ the set of output channels. We want to compute C_j^{out} for each j. Let W^q the tensor of quantized weights. Let the quantization basis $\mathcal{B} = (\mathbf{b_1}, \mathbf{b_2}, \mathbf{b_3}, \mathbf{b_4}, \mathbf{b_5}, \mathbf{b_6}, \mathbf{b_7}, \mathbf{b_8}, \mathbf{b_9})$. Note that in this notation, our basis has dimension 9, which is not the choice we made in the paper. This is not a big deal, since we can choose to fill coordinates with zeros: $\mathbf{b_1} = (b_{1,1}, b_{1,2}, b_{1,3}, 0, 0, 0, 0, 0, 0)$, $\mathbf{b_4} = (0, 0, 0, b_{1,1}, b_{1,2}, b_{1,3}, 0, 0, 0)$, $\mathbf{b_7} = (0, 0, 0, 0, 0, 0, b_{1,1}, b_{1,2}, b_{1,3})$. In this manner, our 3D bases can be expanded in 9D bases.

Usually, in a full precision network, C_j^{out} is computed as :

$$C_j^{out} = \sum_{i=1}^{in} Conv(W_{i,j}; C_i^{in}) \tag{4}$$

In our case, we change the order of operations :

$$C_j^{out} = \sum_{k=1}^{9} \sum_{i=1}^{in} W_{i,j,k}^q Conv(\mathbf{b_k}; C_i^{in}) \tag{5}$$

Since $Conv(\mathbf{b_k}; C_i^{in})$ does not depend on the output channel, we can start by computing $Conv(\mathbf{b_1}; C_i^{in})$ for each i (in convolutions). $\mathbf{b_1}$ is uniformly quantized using a simple min/max quantizer, therefore we can use low-bit operators to compute these convolutions. Then, we only need to multiply the result by $W_{i,j,1}^q$ (which is an integer) for each j, and store the result in the corresponding output channel. Then, we reiterate the process with $\mathbf{b_2}$, $\mathbf{b_3}$, etc.

With this method, the complexity of the forward pass through Λ is $9 \times in$ low-bit convolutions and $9 \times in \times out$ scalar multiplications and additions.

APPENDIX F

Weight-only quantization

Table 11: LatticeQ per-channel weight-only quantization with bias correction.

	Top-1 accuracy			
Network	FP32	W4	W3	W2
Resnet-18	69.6	69.0	66.7	41.7
Resnet-50	76.0	75.5	73.6	44.3
Densenet	74.4	73.2	68.9	6.4
Mobilenet-v2	71.9	66.7	48.9	0.3