TRAINING DEEP NEURAL NETWORKS WITH JOINT QUANTIZATION AND PRUNING OF FEATURES AND WEIGHTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Quantization and pruning are widely used to reduce the inference costs of deep neural networks. In this work, we propose a framework to train deep neural networks using novel methods for uniform quantization and unstructured pruning on both the features and weights. We demonstrate that our method delivers an increased performance per memory footprint over existing state-of-the-art solutions. Using our framework, we empirically evaluate the prune-then-quantize paradigm and independence assumption across a wide range of computer vision tasks and observe the non-commutativity of quantization and pruning when applied to both features and weights.

1 INTRODUCTION

The performance of deep neural networks (DNNs) has been shown to scale with the size of both the training dataset and model architecture (Hestness et al., 2017); however, the resources required to deploy large networks for inference can be prohibitive as they often exceed the budgets of mobile or edge devices (Han et al., 2015; Gale et al., 2019). To minimize the cost of inference, quantization and pruning are widely used techniques that reduce compute and memory requirements by respectively limiting the precision of and removing elements from the computation graph of DNNs.

In this work, we propose a framework to apply novel methods for uniform quantization and unstructured pruning to both the features and weights of DNNs. The majority of state-of-the-art techniques for quantization-aware training calculate gradients using the straight-through estimator (STE), which is notoriously sensitive to weight initialization (Yin et al., 2019; Gholami et al., 2021). To account for this, we propose a modification we refer to as *delayed quantization*, in which we postpone the STE-based calculations to later training stages. When extended to generative adversarial networks (GANs) trained on image-to-image translation tasks, we observe a long-tailed distribution of features similar to the weight distribution observed by Wang et al. (2019). To minimize the impact of outliers, we introduce another modification we refer to as *saturated quantization*, in which we clip the feature values to pre-defined quantiles determined from the training distribution. Finally, we extend the unstructured weight sparsity technique proposed by Zhu & Gupta (2017) to the feature space. To our knowledge, we are the first to thoroughly evaluate the impact of unstructured feature pruning.

Quantization and pruning techniques are often considered to be independent problems (Paupamah et al., 2020; Liang et al., 2021); however, recent work has begun to study the application of both quantization and pruning in a unified scope (Han et al., 2015; Zhao et al., 2019b; Yu et al., 2020; van Baalen et al., 2020). In this work, we aim to more deeply understand the relationship between these optimizations and, in doing so, address two key problems: (1) quantization and pruning techniques are often analyzed over either discriminative or generative tasks, rarely both; (2) frameworks for joint quantization and pruning default to a "prune-then-quantize" paradigm without exploring the alternative.

Beyond simply reducing the storage required to run inference, our goal is to develop a method to jointly apply both uniform quantization and unstructured pruning to both the features and weights of DNNs during training. To this end, we introduce modifications on top of state-of-the-art quantization and pruning techniques that improve performance across a wide range of discriminative and generative tasks. In doing so, we observe a non-commutative nature when quantization and pruning are applied to both the features and weights of DNNs. Based on these results, we state *the non-commutativity hypothesis.*

The Non-Commutativity Hypothesis. *For a given deep neural network trained for a specific task, there exists an order in which pruning and quantization may be applied to optimize network performance.*

Contributions.

1. We propose a framework to train deep neural networks using novel methods for uniform quantization and unstructured pruning on both the features and weights (Section 3).
2. We demonstrate the non-commutative nature of of quantization and pruning when applied to both the features and weights of deep neural networks (Section 4.1).
3. We show that our method delivers the best network performance per memory footprint across a wide range of discriminative and generative computer vision tasks when compared to existing state-of-the-art solutions (Section 4.2).

The code for the algorithms introduced in this paper can be found at https://github.com/anonymous308/qsparse, as discussed further in Appendix A.

2 PRELIMINARIES

2.1 QUANTIZATION

Quantization is the procedure of representing a high-precision input by a discrete set of values with reduction in precision. Uniform quantization refers to the case when the ordered sequence of the discrete set has a constant step size (Pearlman, 1995), on top of which supports efficient implementation of matrix multiplication (Jacob et al., 2018).

Let N be the total number of bits used for quantization, thus the size of the discrete set will be 2^N, and d be the number of decimal bits (*i.e.*, the bits used to represent fractions to the right of the decimal), $\mathbf{x}$ be the input, then uniform quantization can be written as $Q_u(\mathbf{x}, d)$ in Eq. 1.

$$Q_u(\mathbf{x}, d) = \text{clip}(\lfloor \mathbf{x} \times 2^d \rfloor, -2^N, 2^N - 1)/2^d \tag{1}$$

To calculate gradients of uniform quantization operation, straight-through estimator (Hinton et al., 2012), as shown in Eq. 2, is usually applied for backward computation because of its superior performance on convergence speed (Hubara et al., 2017).

$$\frac{\partial Loss}{\partial \mathbf{x}} = \text{clip}(\frac{\partial Loss}{\partial Q_u(\mathbf{x}, d)}, -2^{N-d}, 2^{N-d} - 1) \tag{2}$$

For existing work on joint quantization and pruning, (Yu et al., 2020) only supports to prune and quantize weights, (van Baalen et al., 2020; Wang et al., 2020b) propose gradient-based methods to learn mixed precisions and sparsity ratios but are incapable to provide direct control over target precision and sparsity, (Wang et al., 2020a) applies neural architecture search (NAS) to discover efficient networks with existing quantization and pruning methods and is complementary with our work. None of the above methods support unstructured feature pruning.

2.2 UNSTRUCTURED PRUNING

Pruning is called structured if it imposes structures over the topology of weights or features. Wang & Zhu (2020) design a structured pruning method for weights and features by pruning weights in a specific channel-wise manner. However, structured sparsity provides a trade-off between accuracy and implementation feasibility because unstructured weight sparsity is difficult to efficiently utilize on contemporary GPUs (Gray et al., 2017). In this work, we focus on unstructured pruning because it is most flexible and produces higher compression ratio (Liu et al., 2018), and unstructured feature sparsity has the potential to provide an inherently better trade-off than unstructured and structured weight sparsity as we will discuss next.

Magnitude-based weight pruning. Magnitude-based weight pruning uses weight magnitude as a proxy to represent weight importance and set least important weights to zero following a sparsification schedule. Zhu & Gupta (2017) propose to maintain a binary mask $\mathbf{M}_{\mathbf{w},s}$ for each $\mathbf{w}$, which allows for masked weights to reactivate during training. Let s denote sparsity measured as the percentage of zero-valued elements in a tensor, (i, j) be row and column indices, the unstructured pruning operation can be written as $P_u(\mathbf{w}, s)$ in Eq. 3. $\mathbf{M}_{\mathbf{w},s}$ is calculated in Eq. 4.

$$P_u(\mathbf{w}, s) = \mathbf{w} \circ \mathbf{M}_{\mathbf{w},s} \tag{3}$$

$$\mathbf{M}_{\mathbf{w},s}(i,j) = \begin{cases} 1 & |\mathbf{w}(i,j)| \geq \text{quantile}(|\mathbf{w}|, s) \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$$\text{quantile}(\mathbf{x}, a) = a\text{-th quantile of } \mathbf{x}$$

Sparsity s is controlled and updated by a sparsification schedule at time steps: $t_p + i\Delta t_p, i \in \{1, 2, .., , n\}$, where $(t_p, \Delta t_p, n)$ are hyper parameters of the sparsification schedule and represent (starting step, frequency, steps) of pruning.

Unstructured Feature Pruning. Due to the prevalence of rectified activation functions, DNN features are inherently sparse (Glorot et al., 2011; Xu et al., 2015). Except that the sparse pattern is not pre-determined but depends on each input, thus making it difficult to utilize during inference. In this work, we refer unstructured feature sparsity to a subset of neurons gets removed from a neural network, whose corresponding computations can skipped, and feature maps can be stored with fewer memory. Hu et al. (2016) applies post-training unstructured pruning to remove neurons by taking advantage of existing feature sparse pattern caused by ReLU activation. Contrary to their methods, we propose to apply unstructured pruning on features during training with an introduced sparsity instead of only leveraging existing sparse pattern, and we study this problem in the context of quantization and weight pruning.

Compared to unstructured and structured weight sparsity, unstructured feature sparsity is more promising at providing large memory reduction since feature is usually more dominant in memory consumption than weight in DNNs (Jha et al.). Meanwhile, removed neurons can be skipped during computation because they will always returns zero, and skipping a single output convolution feature value can save an entire dot product (Dumoulin & Visin, 2016).

3 FRAMEWORK

Our framework consists of a quantization method and a pruning method. The joint pipeline is shown in Fig. 1. As the uniform quantization is symmetric around the origin, we place pruning structurally before the quantization method.

3.1 QUANTIZATION METHOD

We make two enhancements on top of the STE-based quantization in Section 2.1, as explained below.

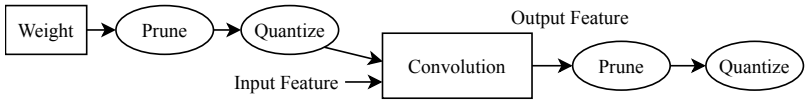

Figure 1: An example of applying joint pruning and quantization of both weight and feature on a convolution layer.

(a) MobileNet	Acc
Baseline	92.60
$Q_8(w, f)$	92.52

(b) ResNet101	mAP
Baseline	74.47
$Q_8(w, f)$	74.25

(c) ESPCN	PSNR
Baseline	32.84
$Q_8(w, f)$	32.68

(d) Pix2Pix	FID
Baseline	119.9
$Q_8(w, f)$	118.5

Table 2: Evaluation of delayed quantization on more tasks. Experiment settings and notation are explained in Section 4.

Delayed Quantization. We start by training the original full-precision network, then calculate the optimal decimal bits (d^*) by minimizing the quantization error after a given number of steps (t_q). Let $\mathbf{x}_t$ to be the quantization input at time t, which can be either feature or weight, our delayed quantization can be written as $Q_d(\mathbf{x}_t)$ in Eq. 5.

$$Q_d(\mathbf{x}_t) = \begin{cases} \mathbf{x}_t & t < t_q \\ Q_u(\mathbf{x}_t, d^*) & t \geq t_q \text{ where } d^* = \arg\min_d |Q_u(\mathbf{x}_{t_q}, d) - \mathbf{x}_{t_q}|^2 \end{cases} \tag{5}$$

In Table 1, we compare our delayed quantization against uniform quantization in Eq. 1 on CIFAR10 with MobileNetV2. Both $Q_u(\mathbf{x}_t, N-1)$ and $Q_u(\mathbf{x}_t, d^*)$ apply STE-based quantization since the start of training, while $Q_d(\mathbf{x}_t)$ delays the quantization according to Eq. 5. Xavior initialization (Glorot & Bengio, 2010) is used in all of the four experiments, which is usually the default initialization method for full precision networks. It can be clearly seen that our method provides nearly no loss in accuracy, while $Q_u(\mathbf{x}_t, N-1)$ which sets decimal bit to $N-1$ according to Jacob et al. (2018), fails to converge properly without a carefully designed weight initialization.

Quantization Method	Top-1 Acc
Baseline	92.6
$Q_d(\mathbf{x}_t)$	92.52
$Q_u(\mathbf{x}_t, N-1)$	75.83
$Q_u(\mathbf{x}_t, d^*)$	18.35

Table 1: Quantization-aware training without special initialization. Delayed Quantitation (Q_d) vs Uniform Quantization (Q_u).

Moreover, if we take the optimal decimal bits d^* from a trained network of $Q_d(\mathbf{x}_t)$ and retrain a new network from scratch with $Q_u(\mathbf{x}_t, d^*)$, the network fails to converge as shown by the last entry in Table 1. This indicates that the feature and weight distribution of neural networks undergo large shifts during training, and these large shifts may not be easily done with quantized weights and features. This distribution shift supports the favor of our delayed quantization approach because our method does not limit the network capacity of shifting its weight and feature distribution before t_q, thereby become less prone to shallow local minimums. Contrary to previous work (Jacob et al., 2018; Bhalgat et al., 2020) which uses floating point auxiliary weights for simulated quantization or finds clever weight initialization, we argue our method is simpler, more efficient for implementation and possibly more robust.

By thorough experiments in Table. 2, we show that our method achieves superior performance while accommodating initialization and training dynamics of various tasks and network structures.

Saturated Quantization. When calculating d^* in Eq. 5, we find the quantization error minimization can be biased from outliers of features in tasks like CycleGAN. In Fig. 2, it can be seen clearly the feature distribution of CycleGAN is much more long-tailed than that of MobileNetV2 for CIFAR10 classification.

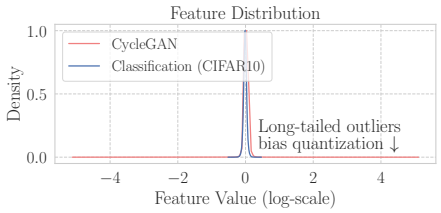

Figure 2: Feature distribution of CycleGAN, and MobileNetV2 for image classification, which displays a long-tailed nature.

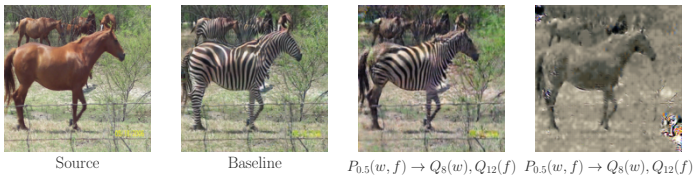

Figure 3: Generated images from CycleGAN, which can suffer from underpresentation of quantization because of outliers in features, as shown in the fourth image (rightmost).

These long tails will result in underestimation of d^* and thereby underpresentation of network features. The fourth image in Fig. 3 shows that the negative effect of underpresentation which collapses the generative quality entirely. This underpresentation issue still persists even if these long-tailed values are quantized in the logarithmic domain (Wang et al., 2019).

Contrary to (Wang et al., 2019), which designs a GAN-specific quantization method with the EM algorithm, we discover that the underpresentation issue can be simply and effectively moderated by clipping the outliers in the linear domain. Specifically, before computing the mean square quantization error as in Eq. 5, we protect quantization from outliers by clipping $\mathbf{x}_{t_q}$ with a saturation function $S_a(\mathbf{x}, q_l, q_u)$ defined in Eq. 6, where hyper parameters $q_l, q_u \in [0, 100], q_l < q_u$ are indexes of quantiles of $\mathbf{x}_{t_q}$ to denote minimum and maximum range after saturation.

$$S_a(\mathbf{x}, q_l, q_u) = \text{clip}(\mathbf{x}, \text{quantile}(\mathbf{x}, q_l), \text{quantile}(\mathbf{x}, q_u)) \tag{6}$$

With Eq. 6, we rewrite our quantization method in Eq. 7.

$$\text{Quantize}(\mathbf{x}_t) = \begin{cases} \mathbf{x}_t & t < t_q \\ Q_u(\mathbf{x}_t, d^*) & t \geq t_q \text{ where } d^* = \arg\min_d |Q_u(\mathbf{x}_{t_q}, d) - S_a(\mathbf{x}_{t_q}, q_l, q_u)|^2 \end{cases} \tag{7}$$

As shown by comparing third and fourth image in Fig. 3, the generative quality can be greatly preserved by saturated quantization. Although we find that the saturated quantization only becomes useful when quantizing the CycleGAN during our experiments, we argue this modification is simple and can be used as an add-on for our method, and therefore is easier to generalize to more scenarios.

3.2 Pruning Method

We adopt the magnitude-based weight pruning method in Section 2.2 and extend it to prune features by maintaining a binary mask $\mathbf{M}_{\mathbf{f}_l,s}$ for each feature map $\mathbf{f}_l$ and prune $\mathbf{f}_l$ as in Eq. 3. Zero values in $\mathbf{M}_{\mathbf{f}_l,s}$ mean the corresponding neurons are removed.

Contrary to weight which is fixed during inference, feature map of a DNN shows different activated regions for each input (Zhou et al., 2016). Therefore, in order to approximate the expected activation pattern without over sacrificing training efficiency, we introduce a

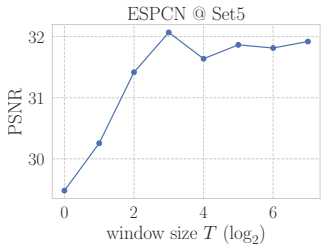

Figure 4: Evaluation of window size T for unstructured feature pruning

sliding window for unstructured feature pruning. Specifically, let $\mathbf{f}_t$ denote an input feature map at time t, T denote the size of sliding window, s denote the target sparsity, (i, j) be row and column indices, instead of calculating the binary mask as in Eq. 4, we write $\mathbf{M}_{\mathbf{f}_t,s}$ in Eq. 8.

$$\mathbf{M}_{\mathbf{f}_t,s}(i,j) = \begin{cases} 1 & \sum_{n=0}^{T-1} |\mathbf{f}_{t-n}(i,j)| \geq \text{quantile}(\sum_{n=0}^{T-1} |\mathbf{f}_{t-n}|, s) \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

In Fig. 4, we evaluate the effect of window size T against PNSR of a super resolution network (ESPCN) in Set5. A clear positive correlation can be observed between PSNR and window size T. It is also interesting to see that the benefit of increasing window size T quickly saturates as T reaches 16. Not only it indicates that we can estimate the expected activation pattern by using Eq. 8, it also aligns with the observation from (Hanin & Rolnick, 2019) that deep relu neural networks empirically show a much fewer activation patterns than their theoretical limit, which corresponds to a severe underuse of neural network potential expressivity but also serves as a supporting evidence for the feasibility of feature pruning.

4 Empirical Analysis

In this section, we empirically evaluate the *non-commutativity hypothesis* with our framework, compare our method against existing ones, and demonstrate that it supports a wide range of computer vision tasks.

We conduct experiments on **(a)** image classification with MobileNetV2 (Sandler et al., 2018) on Cifar10 (Krizhevsky et al., 2009), **(b)** Faster RCNN (Ren et al., 2015) object detection with ResNet101 (He et al., 2016) on Pascal VOC (Everingham et al., 2010), **(c)** super resolution with ESPCN (Shi et al., 2016) on Set5 (Bevilacqua et al., 2012), **(d)** generative adversarial networks, include Pix2Pix (Isola et al., 2017) on facades, and CycleGAN (Zhu et al., 2017) on horse2zebra. We implement each task on top of existing popular open-source implementations. We maintain hyper-parameters and weight initialization identical to the original settings except for training epochs and when to apply quantization & pruning. More details about experiment setups are available in appendix B.

For each task, we consider different combinations and orders of pruning and quantization, and we introduce a notation to describe each configuration succinctly. For example, $P_{0.5}(w) \rightarrow Q_8(w, f)$ means to first prune 50% weight then quantize weight and feature with 8bit, the same rule applies for others.

To fairly compare various pruning and quantization configurations under a unified scope, we introduce a metric we refer to as *performance density* (PD). We define the performance density of a model as its domain-specific performance (e.g., accuracy, PSNR, mAP) divided by the size of both parameters and feature maps. The detailed definition and unit of PD of each task can be found in Appendix B. We will continue to discuss performance density in Section 4.2.

4.1 Non-Commutativity Hypothesis

We start by assessing the previous "prune-then-quantize" paradigm (Han et al., 2015; Zhao et al., 2019b; Yu et al., 2020) and independence assumption (Paupamah et al., 2020; Liang et al., 2021) between quantization and pruning. In Table 3, we experiment the orders of $P_{0.5}(w)$ and $Q_8(w, f)$ over experiments (a), (b) and (c). Note that there could have many combinations over different quantization and pruning setups, but we choose the above ones to isolate the order of these two operations.

We find these existing ideas hold for some cases but not all, which actually softens their strengths. The $P_{0.5}(w) \rightarrow Q_8(w, f)$ performs significantly better than $Q_8(w, f) \rightarrow P_{0.5}(w)$ for (a), which means they are not independent. However, in (c) the order of $P_{0.5}(w)$ and $Q_8(w, f)$ does not make major difference and the

(a) MobileNetV2	Top-1 Acc	PD
Baseline	92.60	0.31
$P_{0.5}(w) \rightarrow Q_8(w, f)$	**92.23**	**1.43**
$Q_8(w, f) \rightarrow P_{0.5}(w)$	85.94	1.33

(b) ResNet101	mAP	PD
Baseline	74.47	0.33
$P_{0.5}(w) \rightarrow Q_8(w, f)$	**74.44**	**0.71**
$Q_8(w, f) \rightarrow P_{0.5}(w)$	74.13	0.71

(c) ESPCN	PSNR	PD
Baseline	32.84	6.22
$P_{0.5}(w) \rightarrow Q_8(w, f)$	32.51	8.36
$Q_8(w, f) \rightarrow P_{0.5}(w)$	**32.54**	**8.37**

Table 3: Empirical analysis of "prune-then-quantize" paradigm and independence assumption. Pruning and quantization are clearly not independent in (a), while "prune-then-quantize" may not be strictly required in (b), (c).

(a) MobileNetV2	Top-1 Acc	PD
Baseline	92.60	0.31
$P_{0.5}(w, f) \rightarrow Q_8(w, f)$	**91.44**	**2.49**
$Q_8(w, f) \rightarrow P_{0.5}(w, f)$	86.84	2.36

(b) ResNet101	mAP	PD
Baseline	74.47	0.33
$P_{0.5}(w, f) \rightarrow Q_8(w, f)$	**73**	**0.86**
$Q_8(w, f) \rightarrow P_{0.5}(w, f)$	70.13	0.82

(c) ESPCN	PSNR	PD
Baseline	32.84	6.22
$P_{0.5}(w, f) \rightarrow Q_8(w, f)$	31.03	8.34
$Q_8(w, f) \rightarrow P_{0.5}(w, f)$	**31.66**	**8.51**

Table 4: Switching the optimal order of pruning and quantization results in large performance degradation, which empirically follows the *non-commutativity hypothesis*. The optimal order for each task is in bold text.

difference in (b) also appears to be marginal, which indicates that these two operations are exchangeable and "prune-then-quantize" may not be strictly required.

We apply unstructured pruning on features, and results are available in Table 4. Comparing Table 3 and 4, it can be seen that feature pruning is able to provide much higher performance density by reducing memory footprint while maintaining a similar level of accuracy to their weight pruning only counterpart. More importantly, it empirically shows that there exists an optimal order between quantization and pruning for each setting, and this optimal order varies among tasks. Changing the order can result in significantly large accuracy degradation, which we conclude as the *non-commutativity hypothesis*.

Moreover, in Table 6 from Section 4.3, it shows that even Pix2Pix and CycleGAN meet the hypothesis, where the difference between $P(w, f) \rightarrow Q(w, f)$ and $Q(w, f) \rightarrow P(w, f)$ is considerably larger than the difference between $P(w) \rightarrow Q(w, f)$ and $Q(w, f) \rightarrow P(w)$.

4.2 Comparing Against Existing Methods

We compare against existing pruning and quantization methods with various network structures in image classification task on Cifar10. Results are available in Table 5, where “W/F Bits” denotes the averaged number of bits used to quantize weights or features, and “W/F Sparsity” represents the averaged sparsity in weights or features.

From Table 5, we can see that our method is the most comprehensive one by supporting both quantization and pruning over both weight and feature. Our method achieves the smallest memory footprint and highest performance density without special adjustment on bitwidth or sparsity. Contrary to methods like (Yang et al., 2020; van Baalen et al., 2020) which obtains an averaged bitwidth and sparsity, our method provides direct control over targeted bitwidth and sparsity, which can be more useful for practical scenarios.

Furthermore, we argue performance density provides a holistic view to existing pruning and quantization methods and neural architectures. For example, in Table 5, DenseNet-76 has the smallest size of weight by both 2-bit weight quantization and pruning (Achterhold et al., 2018), but its feature size is disproportionately large comparing to weight. On the contrary, the VGG16 network is commonly known as a classical neural architecture and has many redundant parameters. Therefore, pruning on VGG16 may not seem to be as impressive as in DenseNet. However, the feature size of VGG16-C is almost one tenth of that of DenseNet76. By pruning 95% of weight, (Dettmers & Zettlemoyer, 2019) achieves a much better performance density using VGG16-C than DenseNet76 in (Achterhold et al., 2018). By introducing performance density in addition to traditional metrics, we become less prone to misconceptions.

Method	Network	W Bits	F Bits	W Sparsity	F Sparsity	Baseline Acc	Accuracy	Weight (Mb)	Feature (Mb)	PD (Acc/Mb)
(Choi et al., 2016)	ResNet-32	8	–	77.8%	–	92.58	92.64 (+0.06)	37.80	131.07	0.55
(Achterhold et al., 2018)	DenseNet-76	2	–	54%	–	92.19	91.17 (-1.02)	0.68	282.53	0.32
(Liu et al., 2018)	VGG-19	–	–	80%	–	93.5	93.52 (+0.02)	128.26	38.80	0.56
	PreResNet-110	–	–	30%	–	95.04	95.06 (+0.02)	952.03	619.71	0.06
	DenseNet-100	–	–	30%	–	95.24	95.21 (-0.03)	27.11	426.74	0.21
(Zhao et al., 2019a)	DenseNet-40	–	–	59.67%	–	94.11	93.16 (-0.95)	3.23	114.87	0.79
(Xiao & Wang, 2019)	VGG-16	–	–	79%	–	93.4	91.5 (-1.9)	98.97	35.39	0.68
(Dettmers & Zettlemoyer, 2019)	VGG16-C	–	–	95%	–	93.51	93 (-0.51)	23.57	35.39	1.58
	WRN-22-8	–	–	95%	–	95.74	95.07 (-0.67)	27.46	230.69	0.37
(Yang et al., 2020)	ResNet-20	1.9	–	54%	–	91.29	91.15 (-0.14)	9.77	78.64	1.03
(van Baalen et al., 2020)	VGG-7	4.75	5.4	–	–	93.05	93.23 (+0.18)	43.85	3.27	1.98
(Paupamah et al., 2020)	MobileNet	8	8	–	–	91.31	90.59 (-0.72)	25.74	13.17	2.33
(Choi et al., 2020)	ResNet-32	8	–	87.5%	–	92.58	92.57 (-0.01)	21.28	131.07	0.61
Ours	MobileNetV2	8	8	50%	–	92.60	92.23 (-0.37)	9.19	55.20	1.43
		8	8	50%	50%		91.44 (-1.16)	**9.19**	**27.60**	**2.49**

Table 5: Comparisons with previous pruning and quantization methods on image classification with Cifar10. Our method is able to deliver the smallest memory footprint (weight + feature) and best performance density and is far more versatile.

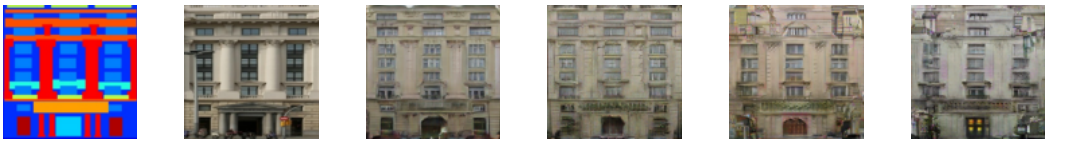

Figure 5: Examples of generated images from Pix2Pix.

Moreover, from Table 5, we observe that the ratio between sizes of feature and weight is much higher in modern network structures like MobileNet, ResNet, and especially DenseNet. On the contrary, this ratio becomes much smaller for classical network structures like VGG16. Based on these observations, we can deduce that the marginal benefit of only applying weight pruning is likely to become smaller, and unstructured feature pruning could be a promising direction as we have argued in Section 2.2.

4.3 Generative Adversarial Networks

In the field of computer vision, the majority of existing works studying quantization and pruning techniques primarily focus on discriminative tasks (*e.g.*, image classification or object detection) rather than generative tasks (*e.g.*, image-to-image translation). However, recent studies have shown impressive results when applying these techniques to generative adversarial networks (GANs). Wang et al. (2019) proposes a weight quantization scheme for GANs. Zhou et al. (2020) uses sparse regularization to improve generative quality. Here, we show our method enables to jointly quantize and prune on Pix2Pix (Isola et al., 2017) and CycleGAN (Zhu et al., 2017), whose selected generated samples are demonstrated in Fig. 5 and Fig. 6, and Fréchet inception distance (FID) (Heusel et al., 2017) are shown in Table 6.

We observe the *non-commutativity hypothesis* to hold as well in these scenario in Table 6. It can also be visually seen from Fig. 5 and 6 that $Q_8(w, f) \rightarrow P_{0.5}(w, f)$ generates higher quality images than $P_{0.5}(w, f) \rightarrow Q_8(w, f)$. We provide more samples in the supplementary.

Moreover, comparing $Q_{12}(w, f) \rightarrow P_{0.5}(w, f)$ and $P_{0.5}(w) \rightarrow Q_{12}(w, f)$ in the CycleGAN table, it is clear that feature pruning provides a significant margin on performance density because it directly reduces the large memory consumption of features in CycleGAN, while it brings smaller benefits to Pix2Pix because Pix2Pix

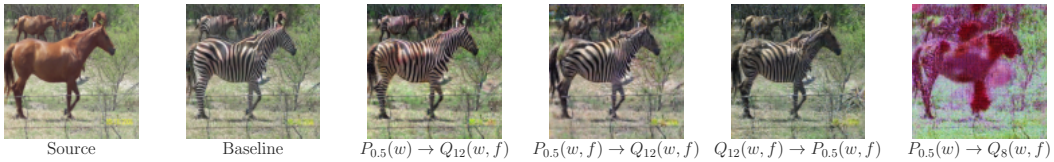

Figure 6: Examples of generated images from CycleGANs.

Pix2Pix	**FID**	**PD**
Baseline	119.9	3.67
$P_{0.5}(w) \rightarrow Q_8(w, f)$	127.1	22.51
$Q_8(w, f) \rightarrow P_{0.5}(w)$	123.5	23.16
$P_{0.5}(w, f) \rightarrow Q_8(w, f)$	154.8	22.37
$Q_8(w, f) \rightarrow P_{0.5}(w, f)$	**135.0**	**25.66**

CycleGAN	**FID**	**PD**
Baseline	67.1	3.28
$P_{0.5}(w) \rightarrow Q_{12}(w, f)$	81.8	11.22
$Q_{12}(w, f) \rightarrow P_{0.5}(w)$	83.72	10.96
$P_{0.5}(w, f) \rightarrow Q_{12}(w, f)$	100.4	16.67
$Q_{12}(w, f) \rightarrow P_{0.5}(w, f)$	**89.35**	**18.73**

Table 6: Fréchet Inception Distance (FID) of generated samples from Pix2Pix and CycleGAN to real images (smaller is better). The reciprocal of FID is used as domain-specific metric to calculate performance density.

uses a different network structure which has smaller feature maps. Moreover, we experimentally find the quantization of feature of CycleGAN needs more bitwidth to deliver good generative results.

5 Limitation and Future Work

We only evaluate our framework and findings on computer vision tasks with a limited set of neural architectures and smaller datasets. In future work, we intend to try more network structures and larger datasets, e.g. ImageNet (Russakovsky et al., 2015), and tasks not limited to computer vision.

Our empirical analysis does not go deep enough to explain the cause of the non-commutativity. We only assess the *non-commutativity hypothesis* with our framework, and the conclusions drawn upon may not be representative enough to generalize to more scenarios. In future work, we intend to study in networks and tasks with more analytical feasibility to draw insights on explaining the formation conditions of this non-commutativity phenomenon and improve understanding of the nature of unstructured feature pruning.

Our unstructured feature pruning method maintains an individual binary mask for each feature map. However, as discussed in Section 3.2, neural networks have many but not an infinite amount of different activation patterns. In future work, we intend to explore the possibility of allowing a many-to-one mapping from binary masks to feature maps and allow the mapping to be dynamically conditioned on the input, which we believe could provide another degree of freedom to the trade-off between model accuracy and inference efficiency.

6 Conclusion

We propose a framework using novel methods for uniform quantization and unstructured pruning on both the features and weights. Unlike previous work which mostly focus on weight pruning and analysis over either discriminative or generative tasks, we extend unstructured weight pruning to the feature space and thoroughly evaluate impact of applying unstructured feature pruning and demonstrate superior performance over a wide range of discriminative and generative tasks. Using our framework, we assess the "prune-then-quantize" paradigm and independence assumption between quantization and pruning, and empirically discover that there exists an order in which pruning and quantization may be applied to optimize network performance for a given DNN in a specific task, which we conclude as *non-commutativity hypothesis*.

REFERENCES

torch.nn.qat — pytorch 1.9.0 documentation. https://pytorch.org/docs/stable/torch.nn.qat.html. (Accessed on 09/01/2021).

yjn870/espcn-pytorch: Pytorch implementation of real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network (cvpr 2016). https://github.com/yjn870/ESPCN-pytorch, 4 2019. (Accessed on 10/05/2021).

kuangliu/pytorch-cifar: 95.47% on cifar10 with pytorch. https://github.com/kuangliu/pytorch-cifar, 2 2021. (Accessed on 10/05/2021).

Jan Achterhold, Jan Mathias Koehler, Anke Schmeink, and Tim Genewein. Variational network quantization. In International Conference on Learning Representations, 2018.

Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie Line Alberi-Morel. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. 2012.

Yash Bhalgat, Jinwon Lee, Markus Nagel, Tijmen Blankevoort, and Nojun Kwak. LSQ+: improving low-bit quantization through learnable offsets and better initialization. CoRR, abs/2004.09576, 2020. URL https://arxiv.org/abs/2004.09576.

Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. Towards the limit of network quantization. arXiv preprint arXiv:1612.01543, 2016.

Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. Universal deep neural network compression. IEEE Journal of Selected Topics in Signal Processing, 14(4):715–726, 2020.

Claudionor N Coelho Jr, Aki Kuusela, Hao Zhuang, Thea Aarrestad, Vladimir Loncar, Jennifer Ngadiuba, Maurizio Pierini, and Sioni Summers. Ultra low-latency, low-area inference accelerators using heterogeneous deep quantization with qkeras and hls4ml. arXiv e-prints, pp. arXiv–2006, 2020.

Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. arXiv preprint arXiv:1907.04840, 2019.

Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285, 2016.

Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. International journal of computer vision, 88(2):303–338, 2010.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. arXiv preprint arXiv:1902.09574, 2019.

Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. arXiv preprint arXiv:2103.13630, 2021.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Proceedings of the fourteenth international conference on artificial intelligence and statistics, pp. 315–323. JMLR Workshop and Conference Proceedings, 2011.

Scott Gray, Alec Radford, and Diederik P Kingma. Gpu kernels for block-sparse weights. arXiv preprint arXiv:1711.09224, 3, 2017.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149, 2015.

Boris Hanin and David Rolnick. Deep relu networks have surprisingly few activation patterns. 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778, 2016.

Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking imagenet pre-training. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 4918–4927, 2019.

Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Patwary, Mostofa Ali, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. arXiv preprint arXiv:1712.00409, 2017.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. Advances in neural information processing systems, 30, 2017.

Geoffrey Hinton, Nitsh Srivastava, and Kevin Swersky. Neural networks for machine learning. Coursera, video lectures, 264(1):2146–2153, 2012.

Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. arXiv preprint arXiv:1607.03250, 2016.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. The Journal of Machine Learning Research, 18(1):6869–6898, 2017.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1125–1134, 2017.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry 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, pp. 2704–2713, 2018.

Nandan Kumar Jha, Sparsh Mittal, and Sasikanth Avancha. Data-type aware arithmetic intensity for deep neural networks. Energy, 120:x109.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. Neurocomputing, 461:370–403, 2021.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. arXiv preprint arXiv:1810.05270, 2018.

Alessandro Pappalardo. Xilinx/brevitas. https://doi.org/10.5281/zenodo.3333552.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems, 32:8026–8037, 2019.

Kimessha Paupamah, Steven James, and Richard Klein. Quantisation and pruning for neural network compression and regularisation. In 2020 International SAUPEC/RobMech/PRASA Conference, pp. 1–6. IEEE, 2020.

W.A. Pearlman. Chapter 2 - information theory and image coding. In Hseuh-Ming Hang and John W. Woods (eds.), Handbook of Visual Communications, Telecommunications, pp. 13–71. Academic Press, San Diego, 1995. ISBN 978-0-08-091854-9. doi:https://doi.org/10.1016/B978-0-08-091854-9.50007-5. URL https://www.sciencedirect.com/science/article/pii/B9780080918549500075.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems, 28:91–99, 2015.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention, pp. 234–241. Springer, 2015.

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. International Journal of Computer Vision (IJCV), 115(3):211–252, 2015. doi:10.1007/s11263-015-0816-y.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4510–4520, 2018.

Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1874–1883, 2016.

Mart van Baalen, Christos Louizos, Markus Nagel, Rana Ali Amjad, Ying Wang, Tijmen Blankevoort, and Max Welling. Bayesian bits: Unifying quantization and pruning. arXiv preprint arXiv:2005.07093, 2020.

Peiqi Wang, Dongsheng Wang, Yu Ji, Xinfeng Xie, Haoxuan Song, XuXin Liu, Yongqiang Lyu, and Yuan Xie. Qgan: Quantized generative adversarial networks. arXiv preprint arXiv:1901.08263, 2019.

Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, Hanrui Wang, Yujun Lin, and Song Han. Apq: Joint search for network architecture, pruning and quantization policy. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2078–2087, 2020a.

Wei Wang and Liqiang Zhu. Structured feature sparsity training for convolutional neural network compression. Journal of Visual Communication and Image Representation, 71:102867, 2020.

Ying Wang, Yadong Lu, and Tijmen Blankevoort. Differentiable joint pruning and quantization for hardware efficiency. In European Conference on Computer Vision, pp. 259–277. Springer, 2020b.

Xia Xiao and Zigeng Wang. Autoprune: Automatic network pruning by regularizing auxiliary parameters. Advances in Neural Information Processing Systems 32 (NeurIPS 2019), 32, 2019.

Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. arXiv preprint arXiv:1505.00853, 2015.

Haichuan Yang, Shupeng Gui, Yuhao Zhu, and Ji Liu. Automatic neural network compression by sparsity-quantization joint learning: A constrained optimization-based approach. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2178–2188, 2020.

Jianchao Yang, John Wright, Thomas S Huang, and Yi Ma. Image super-resolution via sparse representation. IEEE transactions on image processing, 19(11):2861–2873, 2010.

Jianwei Yang, Jiasen Lu, Dhruv Batra, and Devi Parikh. A faster pytorch implementation of faster r-cnn. https://github.com/jwyang/faster-rcnn.pytorch, 2017.

Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley Osher, Yingyong Qi, and Jack Xin. Understanding straight-through estimator in training activation quantized neural nets. arXiv preprint arXiv:1903.05662, 2019.

Po-Hsiang Yu, Sih-Sian Wu, Jan P Klopp, Liang-Gee Chen, and Shao-Yi Chien. Joint pruning & quantization for extremely sparse neural networks. arXiv preprint arXiv:2010.01892, 2020.

Chenglong Zhao, Bingbing Ni, Jian Zhang, Qiwei Zhao, Wenjun Zhang, and Qi Tian. Variational convolutional neural network pruning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2780–2789, 2019a.

Yiren Zhao, Xitong Gao, Daniel Bates, Robert Mullins, and Cheng-Zhong Xu. Focused quantization for sparse cnns. Advances in Neural Information Processing Systems, 32:5584–5593, 2019b.

B. Zhou, A. Khosla, Lapedriza. A., A. Oliva, and A. Torralba. Learning Deep Features for Discriminative Localization. CVPR, 2016.

Kang Zhou, Shenghua Gao, Jun Cheng, Zaiwang Gu, Huazhu Fu, Zhi Tu, Jianlong Yang, Yitian Zhao, and Jiang Liu. Sparse-gan: Sparsity-constrained generative adversarial network for anomaly detection in retinal oct image. In 2020 IEEE 17th International Symposium on Biomedical Imaging (ISBI), pp. 1227–1231. IEEE, 2020.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In Proceedings of the IEEE international conference on computer vision, pp. 2223–2232, 2017.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. arXiv preprint arXiv:1710.01878, 2017.

A SOFTWARE LIBRARY

We release our method in a pytorch-based python library named `qsparse`. In Listing 1, we demonstrate how to use our method through `qsparse` to quantize and prune the weight and feature of a convolution layer in pytorch.

Due to the simplicity of our method, we are able to implement the software interface with a novel design. The majority of existing model compression python library only provide a minimal set of quantized or pruned layers or modules isolated from other floating-point modules e.g. `QuantiConv2d` from (Pappalardo), which means they do not have enough flexibility to support a wide range of neural network specifications (Coelho Jr et al., 2020; Qat). To improve the library flexibility, we propose a creative technique which directly transforms the weight attribute of the input layer into a pruned or quantized version during runtime. This is to say, our library is layer-agnostic and can work with any pytorch modules, as long as their parameters can be accessed from the weight attribute, which is the convention of pytorch standard library (Paszke et al., 2019).

```python
import torch.nn as nn
from qsparse import prune, quantize

# feature pruning and quantization
nn.Sequential(
    nn.Conv2d(10, 30, 3),
    prune(0.5, start=1000, interval=100, repetition=3),
    quantize(bits=8, timeout=2000)  # `timeout` is `quantize step` in Alg. 1
)

# weight pruning and quantization (layers other than `Conv2d` work as well)
qpconv = quantize(prune(nn.Conv2d(10, 30, 3), 0.5), 8)
```

Listing 1: Examples of our software interface for quantization and pruning on both weights and features

B EXPERIMENT SETUP

We implement our experiments by applying minimal code changes with our library in Appendix A on top of existing popular open-source implementations. We will release all experiment source code as examples for our library. We keep all hyper parameters identical to each original implementation except training epochs and the ones introduced by our method, namely:

- t_q: The number of epochs to delay quantization, used in Eq. 7. Here we write t_{qw}, t_{qf} to denote t_q for weight and feature.
- q_u, q_l: The indexes of quantiles to denote minimum and maximum range for saturated quantization, used in Eq. 7.
- $t_p, \Delta t_p, n$: Sparsification schedule, explained in Section 2.2. We use the same schedule for both weight and feature pruning.
- T: The window size for unstructured feature pruning in Eq. 8.

B.1 IMAGE CLASSIFICATION ON CIFAR10

We conduct experiments of this task using (kua, 2021). We use MobileNetV2 for all experiments in this task. For quantization, we quantize all the weights and feature maps including network input to 8 bit. We do not quantize the bias because bias is usually stored and computed with high precision during on-device inference (Jacob et al., 2018). For pruning, we prune all weights and feature maps except the first and last layers.

For hyper parameters, we set training epoch to 250, q_u to 0, q_l to 1, and T to 2048 in each experiment.

- For $P_{0.5}(*) \rightarrow Q_8(*)$, t_{qw}, t_{qa} are set to $230, 235$, and sparsification schedule $t_p, \Delta t_p, n$ are set to $100, 15, 4$.
- For $Q_8(*) \rightarrow P_{0.5}(*)$, t_{qw}, t_{qa} are set to $160, 170$, and sparsification schedule $t_p, \Delta t_p, n$ are set to $180, 15, 4$.
- For $Q_8(*)$, t_{qw}, t_{qa} are is set to $230, 240$.

Performance density (PD) for this task is calculated as "Accuracy $\div$ size of weight and feature combined", whose unit is Accuracy/Mb. The size of feature is calculated based on input size of (3, 32, 32).

B.2 SUPER RESOLUTION ON SET5

We conduct experiments of this task using (yjn, 2019). We use ESPCN for all experiments and train ESPCN using 91-image (Yang et al., 2010). Similar to Appendix B.1, we quantize the entire network with 8 bit and skip the pruning for first and last layer.

For hyper parameters, we set training epoch to 200, q_u to 0, q_l to 1, and T to 16 in each experiment.

- For $P_{0.5}(*) \rightarrow Q_8(*)$, t_{qw}, t_{qa} are set to $160, 170$, and sparsification schedule $t_p, \Delta t_p, n$ are set to $140, 5, 4$.
- For $Q_8(*) \rightarrow P_{0.5}(*)$, t_{qw}, t_{qa} are set to $140, 150$, and sparsification schedule $t_p, \Delta t_p, n$ are set to $155, 5, 4$.
- For $Q_8(*)$, t_{qw}, t_{qa} are is set to $140, 150$.

ESPCN is a fully convolutional network which is trained to upsample image patches of size (17, 17) to (48, 48). During testing, ESPCN is used to upsample images of size (170, 170) to (480, 480). Because of this size mismatch between training and testing, the binary mask $\mathbf{M}_{\mathbf{f}_t,s}$ from unstructured feature pruning in Eq. 8 learned during training cannot be directly applied for testing. To solve this issue, we replicate the $\mathbf{M}_{\mathbf{f}_t,s}$ along height and width axis for each feature map to match the testing size requirements.

Performance density (PD) for this task is calculated as "PSNR $\div$ $\log$(size of weight and feature combined)", whose unit is set to PSNR/log(Mb). The reason we use $\log$ here is that PSNR is a logarithmic domain metric. The size of feature is calculated based on input size of (1, 170, 170).

B.3 OBJECTION DETECTION ON PASCAL VOC

We conduct experiments of this task using (Yang et al., 2017). We use Faster RCNN with ResNet101 as backbone for all experiments. The ResNet101 is pretrained on ImageNet dataset (Russakovsky et al., 2015). We follow a similar quantization and pruning scheme to Appendix B.1 but we do not prune or quantize for the first 3 residual blocks because they are instructed to be frozen by the original paper (Ren et al., 2015). We also freeze all pre-trained batch normalization layers. However, these limitations can be overcome with group normalization or synchronized batch normalization according to (He et al., 2019). Thus, we do not regard this as our limitation.

For hyper parameters, we set training epoch to 9, q_u to 0, q_l to 1, and T to 32 in each experiment.

- For $P_{0.5}(*) \rightarrow Q_8(*)$, t_{qw}, t_{qa} are set to $7.2, 7.5$, and sparsification schedule $t_p, \Delta t_p, n$ are set to $3, 0.5, 4$.
- For $Q_8(*) \rightarrow P_{0.5}(*)$, t_{qw}, t_{qa} are set to $5.2, 5.5$, and sparsification schedule $t_p, \Delta t_p, n$ are set to $5.7, 0.5, 4$.
- For $Q_8(*)$, t_{qw}, t_{qa} are set to $5.2, 5.5$.

Since the image sizes are different between each batch during Faster RCNN training, it is impossible to obtain a fixed binary mask $\mathbf{M}_{\mathbf{f}_t,s}$ on features of convolution layers. Thus, we apply channel-wise pruning on convolution feature.

Performance density (PD) for this task is calculated as "mAP ÷ size of weight and feature combined", whose unit is set to mAP/Gb. The size of feature is calculated based on input size of (3, 480, 720).

B.4 PIX2PIX ON FACADES AND CYCLEGAN ON HORSE2ZEBRA

We conduct experiments of these two tasks using (Isola et al., 2017; Zhu et al., 2017). Both Pix2Pix and CycleGAN applies networks with encoder-decoder structure, while the Pix2Pix uses UNet (Ronneberger et al., 2015) and CycleGAN uses a variant of ResNet with deconvolution layers. The quantization and pruning scheme is similar to Appendix B.1, but we use 12bit quantization for CycleGAN features. For pruning, in addition to skipping first and last layer, the second layer and last second layer of CycleGAN, and the most inner layer in UNet of Pix2Pix are also skipped for both weight and feature pruning.

For hyper parameters, we set q_u to 0 and q_l to 1 in each Pix2Pix experiment and q_u to 0.0001 and q_l to 0.9999 in each CycleGAN experiment. Other hyper parameters are equal between Pix2Pix and CycleGAN. We set training epoch to 300, T to 128.

- For $P_{0.5}(*) \rightarrow Q_8(*)$, t_{qw}, t_{qa} are set to $280, 290$, and sparsification schedule $t_p, \Delta t_p, n$ are set to $100, 15, 4$.
- For $Q_8(*) \rightarrow P_{0.5}(*)$, t_{qw}, t_{qa} are set to $110, 120$, and sparsification schedule $t_p, \Delta t_p, n$ are set to $130, 15, 4$. T is set to 128.
- For $Q_8(*)$, t_{qw}, t_{qa} are set to $110, 120$.

Performance density (PD) for this task is calculated as "1 / FID ÷ size of weight and feature combined", whose unit is set to 1/(FID*bit). The size of feature is calculated based on input size of (3, 256, 256).