Fully Quantized Network for Object Detection

Rundong Li [*] [†] [‡]   Yan Wang [*] [‡]   Feng Liang [‡]   Hongwei Qin [‡]   Junjie Yan [‡]   Rui Fan [†]
[†] ShanghaiTech University   [‡] SenseTime Research
{lird, fanrui}@shanghaitech.edu.cn
{wangyan1, liangfeng, qinhongwei, yanjunjie}@sensetime.com

Abstract

Efficient neural network inference is important in a number of practical domains, such as deployment in mobile settings. An effective method for increasing inference efficiency is to use low bitwidth arithmetic, which can subsequently be accelerated using dedicated hardware. However, designing effective quantization schemes while maintaining network accuracy is challenging. In particular, current techniques face difficulty in performing fully end-to-end quantization, making use of aggressively low bitwidth regimes such as 4-bit, and applying quantized networks to complex tasks such as object detection. In this paper, we demonstrate that many of these difficulties arise because of instability during the fine-tuning stage of the quantization process, and propose several novel techniques to overcome these instabilities. We apply our techniques to produce fully quantized 4-bit detectors based on RetinaNet and Faster R-CNN, and show that these achieve state-of-the-art performance for quantized detectors. The mAP loss due to quantization using our methods is more than $3.8\times$ *less than the loss from existing methods.*

1. Introduction

State-of-the-art object detectors are based on powerful convolution neural network (CNN) architectures [26, 21, 23, 25]. While CNNs achieve remarkable accuracy, their high computational cost during inference restricts their usage on resource-limited devices such as mobile phones, smart cameras and drones.

To perform efficient inference on complex networks, several techniques have been proposed. These include improved network designs [12, 15, 32] and network search [36], network pruning [9, 8] and network quantization. Current research areas in network quantization include reducing the bitwidth of network parameters to decrease model memory usage [33, 11, 30], quantizing both network parameters and activations to accelerate specific types of layers in the network using bitwise operations [35, 31, 14, 4], and quantizing network gradients to speed up distributed training [2].

Although promising results on tasks such as image classification have been reported [31, 4, 35, 33, 24] using the aforementioned quantization techniques, using quantized networks for more complex tasks such as object detection remains a challenge. Issues faced by current methods include:

Hardware-friendly end-to-end quantization Many current quantization techniques [14, 35] focus on specific types of operations such as convolutions or matrix multiplication, while leaving other operations and network layers in full precision. This introduces two problems. The first is that critical operations such as batch normalization are either not handled [35] or ablated [29] during training, leading to a mismatch between training and evaluation behaviors, or causing training convergence difficulties. Another problem is that when deployed on real hardware, both integer and floating point arithmetic units are needed for performing network inference. This causes data exchange between different arithmetic units that may sometimes negate the speedups achieved by quantization.

Low bitwidth quantization on complex tasks Current quantization research mostly falls into two categories. The first focuses on performing aggressive bit-width compression, *e.g.* using ternary [19, 1] or even binary [13, 24] values, and applies this to relatively simple tasks such as classification, where high accuracy is not required. Another type of research uses relatively conservative quantization, *e.g.* 8 bits, but can be applied to a broader range of more complex tasks such as neural language processing [11], face attribute extraction [17] and object detection [18]. Detection is complex because given a candidate region, the detector needs to not only classify whether this region contains a target object class, but also accurately regress a bounding box if an object is convincingly detected. There have been several works on quantizing detectors to use 8 bits [17, 28]. However, as the computational complexity of multiplication grows superlin-

[*] Equal contributions. This work was done when Rundong, Yan and Feng were interns at SenseTime Research.

early in the bitwidth of the operands, 8 bit arithmetic is often still too expensive for devices with very limited resources. A natural question thus arises whether even stronger quantization such as 4-bit can be applied to accelerate detection and other complex tasks. To answer this question, we examined a carefully designed fully end-to-end quantized detector proposed in [17]. When quantizing this detector to lower than 8 bits, we discovered that the quantization-aware fine tuning process was unstable and had difficult converging. Several best practices for 8-bit fine tuning led to very poor final accuracy in the 4-bit setting. By monitoring the evolution of the model's weights and gradients during fine tuning, we found that the poor accuracy and convergence comes from instability in several sensitive operations of the quantized model. In particular, we observed that in batch normalization layers, where batch statistics are computed using aggressively quantized activations, the very small batch sizes used in detector fine tuning led to highly degraded estimates of statistical quantities. We also found that activations after batch normalization often contain outliers that decrease quantization accuracy. Finally, we found that different channels of the model weights have large differences in magnitude, so that performing layer-wise normalization introduces large inaccuracies in certain channels.

To address these problems, we propose three effective improvements to current quantization-aware fine tuning schemes:

1. Freeze batch norm statistics during quantization-aware fine tuning, and always normalize activations by the means and variances obtained after the training stage.

2. Use a small subset of the training set to calibrate activation magnitudes. Discard outlier activation values based on percentiles and clamp quantized activations and gradients.

3. Use channel-wise quantization for all parameters to reduce quantization error, at the expense of a negligible amount of additional computation.

We apply our techniques to build 4-bit versions of the one-stage RetinaNet detector and two-stage Faster R-CNN detector, using a variety of networks including ResNet-18, ResNet-50 and MobileNetV2 as backbones. We performed extensive experiments using the COCO benchmark, and detailed ablation studies to identify the effectiveness of our proposed improvements. Contributions of this work include:

1. We propose a hardware-friendly quantization scheme which does not use any floating point arithmetic operations during inference.

2. We identify several difficulties faced by current low-bitwidth detectors during fine tuning, and propose techniques to stabilize fully quantized detector fine tuning.

3. We construct and report on the performance of state-of-the-art detectors quantized to 4 bits. To our knowledge, these are the first fully quantized 4-bit object detection models that achieve acceptable accuracy loss and requires no special hardware design, and thus may be used as a baseline for future end-to-end low-bit quantization schemes on complex tasks.

2. Related works

2.1. Modern Detectors

In recent years, the dominant method for object detection has been anchor-based detection networks, including single stage detectors such as SSD [23], YOLO [25] and RetinaNet [21], and the two stage R-CNN series of detectors [7, 6, 26, 20]. In the one stage method, the features of a convolutional backbone network are fed to subnetworks for object classification and bounding box regression. In two stage methods, the first stage generates a set of object candidates with rough locations, then in the second stage these candidates are classified according to target labels and their bounding box locations are refined.

2.2. Network Quantization

Network quantization is an effective method for speeding up neural networks. Most network quantization research has focused on object classification, including BNN [13], QNN [14], XNOR-Net [24], DoReFa-Net [35], INQ [33], ELQ [34], LQ-Nets [31] etc.

In terms of quantization of object detector networks, Google proposed an 8-bit quantization method [17], which led to significant improvements in the latency-accuracy tradeoff for MobileNets [12] on both ImageNet classification [5] and COCO object detection [22]. Wei *et al* [28] quantized activations in object detection models for the purpose of knowledge transfer from large to small models.

3. Techniques for Fully Quantized Network

In this section, we introduce a set of quantization schemes, fine tuning protocols and several specific enhancements, which we together call Fully Quantized Network (FQN), allowing quantization of an object detection network which uses full precision arithmetic to one using 4-bit arithmetic, while largely retaining the accuracy of the original network. Network quantization typically consists of three stages, full precision training, quantization and fine tuning, and finally deployment of the quantized model. We empirically observe that most quantization problems arise during the fine tuning stage, and FQN focuses on this stage.

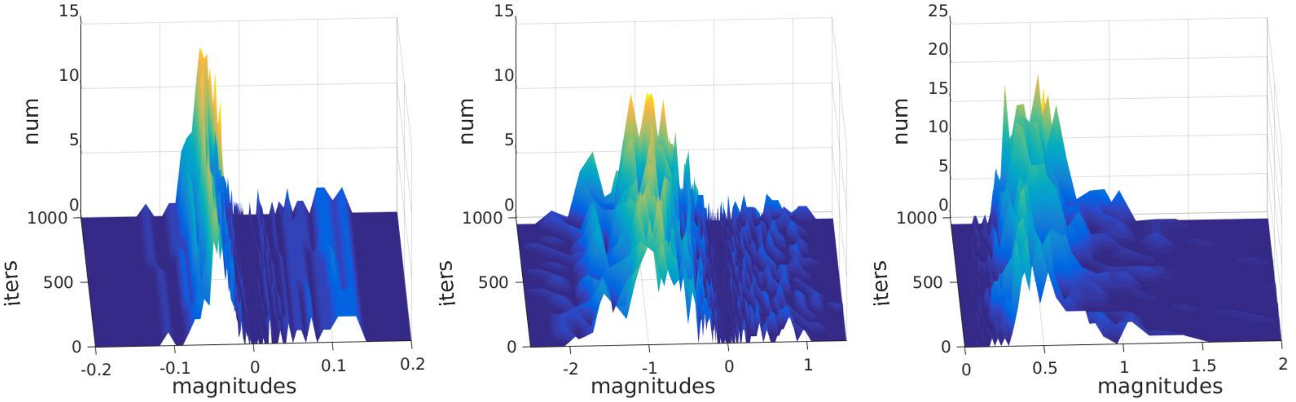

Figure 1: Histograms of affine parameter β (*left*), batch average μ_b (*middle*) and variance σ_b (*right*) at the batch norm layer "layer2.0.bn1" of a ResNet-18 RetinaNet detector during 4-bit fine tuning. Batch statistics become unstable due to small batch size and aggressively quantized activations, and thus are not used in this work.

FQN uses asymmetric uniform quantization, making it easily deployable on real world devices.

3.1. Network Quantization Process

We first review the three main steps for network quantization.

Full-precision training is performed if no trained detector is provided. During training, weights, activations and gradients are all processed in full-precision. In each batch normalization layer, an average μ_b and variance σ_b is computed for each feature channel, and then used to normalize each feature within the current minibatch. In addition, each batch normalization layer keeps track of exponential moving average (EMA) statistics μ_{EMA} and σ_{EMA}, and updates them at each forward step by μ_b and σ_b.

Quantization-aware fine tuning is performed once full-precision training is done, or if a well trained detector is initially provided. This stage consists of additional training steps, but in which forward passes operate on weights and activations that have been quantized to the same bitwidth as that to be eventually used during inference. Note that full precision copies of the weights are still maintained, and are updated by full precision gradients throughout fine tuning.

In [17], batch normalization layers normalize input features and update μ_{EMA}, σ_{EMA} with batch statistics μ_b and σ_b during fine tuning. We empirically find this harms final accuracy, and instead we propose to prevent these values from being updated during fine tuning, as discussed in §3.5.1.

Fully-quantized inference can be performed on hardware with integer arithmetic units, or simulated on GPUs using floating point operations. Given a fine tuned detector from the previous phase, batch normalization values μ_{EMA} and σ_{EMA} and affine parameters are folded into each corresponding layer's weights, to eliminate explicit normalization and scaling during inference. Activations from normalized inputs to output predictions and all weights are quantized to the target bitwidth, and no floating point operations are performed.

3.2. Uniform Quantization

Modern neural networks store weights, activations and gradients as tensors of floating point values. Quantization rounds these to a smaller set of values to reduce the number of bits used in their representation. Given a full precision tensor $\mathbf{X}^R = [x^R_{0,\ldots n-1}]$ and target bitwidth k, the quantization function $Q_k(\cdot)$ maps x^R_i to the nearest quantization point q_j:

$$\mathbf{X}^Q = Q_k(\mathbf{X}^R) \in \{q_0, q_1, \ldots q_{2^k-1}\} \tag{1}$$

We adopt a *uniform quantization* scheme in this work, where distances between adjacent quantized points are equal, so that $\mathbf{X}^Q$ can be represented as

$$\mathbf{X}^Q = \Delta(\mathbf{X}^I - z) \tag{2}$$

where Δ is distance between adjacent quantized points, $\mathbf{X}^I$ is a set of integer indices and z is the index for the bias. A quantization range $[lb, ub]$, for $lb, rb \in \mathbb{R}$ is used to determine q_j and Δ:

$$lb = q_0, ub = q_{2^k-1} \tag{3}$$

$$\Delta = \frac{ub - lb}{2^k - 1} \tag{4}$$

Once Δ computed, the indices tensor $\mathbf{X}^I$ can be computed

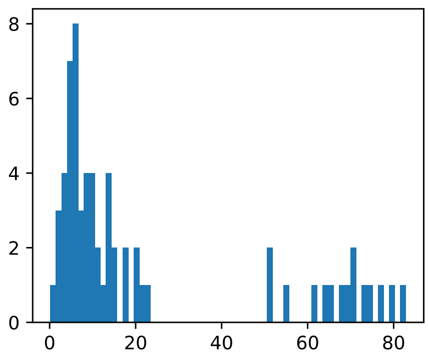

Figure 2: Histogram of channel-wise magnitude variations in weights of a ResNet-50 RetinaNet detector. The x-axis is the ratio of maximum to minimum magnitudes in different weight channels, given in units of dB. The y-axis shows the frequency of such log ratios. Note the bars on the right, indicating channels with large weight variations.

by:

$$\mathbf{X}^R = \text{clamp}(\mathbf{X}^R, lb, ub) \tag{5}$$

$$\mathbf{X}^I = \lfloor \frac{\mathbf{X}^R - lb}{\Delta} \rceil \tag{6}$$

where $\text{clamp}(x, lb, ub) = \max(\min(x, ub), lb)$ restricts the first argument to the interval spanned by the second and third arguments, and where $\lfloor \cdot \rceil$ is the round operator.

When both weights and activations are quantized during inference, expensive floating point tensor arithmetic can be replaced by efficient integer arithmetic [1]:

$$\mathbf{y} = Q_k(\mathbf{W})Q_k(\mathbf{x}) = \Delta_{\mathbf{W}}\Delta_{\mathbf{x}}(\mathbf{W}^I\mathbf{x}^I) \tag{7}$$

Since the mapping operation $Q_k(\cdot)$ is not differentiable, the straight though estimator (STE) [3] is used during network training. Note that entries outside the quantization boundaries receive no gradient:

$$\frac{\partial y}{\partial x_i^R} = \begin{cases} \frac{\partial y}{\partial x_i^Q} & \text{if } lb \leq x_i^R \leq ub \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

Unless noted, the following discussions are based on uniform quantization with bitwidth $k = 4$.

3.3. Weight Quantization

CNN based detectors are usually formed by combining convolutional and fully connected layers. Convolutional layer weights are represented by a tensor with shape $c^{out} \times c^{in} \times h^k \times w^k$, and fully connected layer weights have shape $c^{out} \times c^{in}$. For both types of weight tensors $\mathbf{W} \in \mathbb{R}^{c^{out} \times *}$ in FQN, quantization boundaries are computed along each of the c^{out} output channels:

$$lb = \min_{axis=1:\mathbf{W}.dim}(\mathbf{W}) \tag{9}$$

$$ub = \max_{axis=1:\mathbf{W}.dim}(\mathbf{W}) \tag{10}$$

Each output channel can have a different quantization boundary. By quantizing each channel independently we ensure channels with small weight ranges use finer quantization and smaller Δ values than channels with larger weight ranges. We found empirically that different channels can differ drastically in their weight ranges; for example, in ResNet-50's `layer2.0.conv1` layer weight ranges varied from 3.745×10^{-8} to 0.727. As shown in Figure 2, a considerable number of weight channels in ResNet-50 have significant magnitude variations. Using layer-wise quantization in this case would have introduced severe distortion in some of the quantized values.

3.4. Activation Quantization

Unlike most of works, we quantize all activations in FQN from the normalized input, to the final predictions feeding into anchor regression and non-maximum suppression (NMS). Activations are quantized in a layer-wise manner.

To determine quantization ranges lb and ub for activation $\mathbf{x}^l$ on layer l, [17] used exponential moving averages (EMA) with momentum M to record a smoothed minimum lb^l_{EMA} and maximum ub^l_{EMA} for $\mathbf{x}^l$. During fully quantized fine tuning, each activation's lb and ub values are assigned with these EMA statistics.

We encountered several problems when fine tuning a 4-bit detector using this method:

1. Hyperparameters such as EMA momentum M are difficult to set properly, and the final statistics lb^l_{EMA} and ub^l_{EMA} are very sensitive to these hyperparameters.

2. Normalized activations are more likely to contain outlier values. If hyperparameters are not properly set, the outliers will expand the quantization range, decrease quantization resolution and introduce quantization noise, especially on activation channels with smaller magnitudes.

3.4.1 Reducing Activation Instability

To control this instability, we develop a simple yet effective method. We randomly sample n_{cal} batches of training data to build a small calibration set, then evaluate a trained detector on the calibration set and record each layers activation values, between the γ'th and $1 - \gamma$'th percentiles, for some $0 < \gamma < 1$.

[1] We omit the bias term z for clarity.

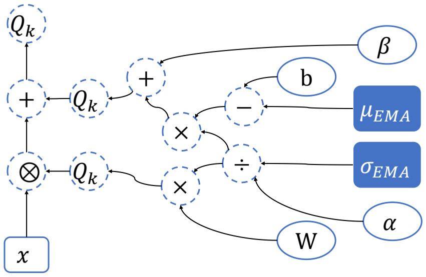

Figure 3: Stabilized batch norm folding: BN statistics μ, σ and affine parameters α, β are folded into parameters W and b of the preceding Conv or FC layers. Note that statistics μ_{EMA} and σ_{EMA} from trained full-precision BN layers are used instead of per-bath statistics from quantized activations, to address the instability during quantization-aware fine-tuning.

We empirically find using $n_{cal} = 20$ and $\gamma = 0.999$ yield good performance on all our evaluated settings. In addition, as shown in §4.2, quantization fine tuning results are largely insensitive to γ selection.

3.5. Batch Normalization Folding

During training, the batch normalization [16] layers normalize input x^l in batch b using minibatch statistics μ_b and σ_b to eliminate covariance shifting. μ_b and σ_b are also used to update EMA statistics μ_{EMA} and σ_{EMA}, smoothed by a momentum. During inference, EMA statistics and the batch normalization affine parameters α and β are folded into the previous layer's weights and biases. Folded weights are then quantized in the same way as normal weights and used in subsequent computations.

In [17], the training computation graph is modified to simulate noise introduced from the quantized folded weights:

$$W_{fold} = \frac{\alpha}{\sqrt{\sigma_b^2 + \epsilon}} W \tag{11}$$

$$b_{fold} = \frac{\alpha}{\sqrt{\sigma_b^2 + \epsilon}} (b - \mu_b) + \beta \tag{12}$$

where σ_b and μ_b are full precision values.

3.5.1 Reducing Batch Normalization Instability

As shown in Figure 1, the batch normalization parameters and batch statistics are both unstable during fine tuning. This introduces significant quantization noise when the parameters are folded into the previous layer, and this effect is magnified when using aggressive 4 bit quantization. In practice, this is one of the main obstacles to fine tuning stability.

We propose a simple solution to address this problem. Since the EMA statistics of μ and σ from a well trained model should match that of the input data, during fine tuning we replace the unstable σ_b, μ_b in (11) and (12) by the EMA statistics σ_{EMA} and μ_{EMA}, obtained during the full-precision training stage. Furthermore, we do not update the σ_{EMA}, μ_{EMA} values during fine tuning; we call this *freezing* the batch normalization values. This procedure is illustrated in Figure 3.

$$W_{fold} = \frac{\alpha}{\sqrt{\sigma_{EMA}^2 + \epsilon}} W \tag{13}$$

$$b_{fold} = \frac{\alpha}{\sqrt{\sigma_{EMA}^2 + \epsilon}} (b - \mu_{EMA}) + \beta \tag{14}$$

We show in experiments that freezing batch normalization values improves fine tuning stability and yields improved accuracy. Another benefit is that no additional computations are needed during fine tuning to calculate σ_b and μ_b. A similar technique is also used in [18], but they only freeze the batch normalization values during the last few thousand steps of training.

3.6. Implementation Details

Zero-point alignment As a standard practice when deploying quantized networks to hardware, the zero point in $\mathbf{X}^R$ should be accurately mapped to $\mathbf{X}^Q$. This alignment is critical to maintaining the quantized network's accuracy, because misalignment of the zero-point will introduce significant errors in operations such as zero-padding. We do this alignment by nudging quantization boundaries with respect to quantization resolution.

Upsampling and element-wise operations Feature pyramid networks (FPN) [20] are commonly used in modern detectors. To eliminate float point operations in evaluating FPN, we adopt two modifications: 1. All upsamplings are performed by nearest interpolation. 2. Element-wise addition is performed using the same scheme as [17] to eliminate floating point rescaling, *i.e.* rescaling of operands Δ is performed by higher precision fix-point multiplication followed by bit-shifting. $Q_k(\cdot)$ is added to addition inputs and outputs to model this behavior during fine tuning.

4. Experiments

To evaluate the proposed FQN detectors, we perform a series of experiments on the COCO detection benchmark [22]. COCO is one of the most popular object detection dataset. It is widely used to benchmark state-of-the-art

Model	Input	mAP						mAR					
		AP	$\mathrm{AP}^{0.5}$	$\mathrm{AP}^{0.75}$	AP^{S}	AP^{M}	AP^{L}	AR^{1}	AR^{10}	AR^{100}	AR^{S}	AR^{M}	AR^{L}
R50-FP32	800	0.356	0.551	0.382	0.203	0.393	0.465	0.308	0.498	0.529	0.335	0.569	0.680
R50-INT4	800	0.325	0.515	0.347	0.173	0.356	0.426	0.286	0.463	0.493	0.298	0.530	0.643
R34-FP32	800	0.348	0.538	0.371	0.192	0.381	0.460	0.306	0.493	0.523	0.319	0.564	0.686
R34-INT4	800	0.313	0.504	0.333	0.161	0.344	0.416	0.284	0.457	0.487	0.284	0.524	0.647
R18-FP32	800	0.317	0.503	0.337	0.164	0.346	0.424	0.288	0.464	0.495	0.297	0.529	0.652
R18-INT4	800	0.286	0.469	0.299	0.149	0.312	0.387	0.268	0.433	0.461	0.269	0.491	0.621
MN2-FP32	600	0.275	0.448	0.290	0.154	0.289	0.365	0.267	0.441	0.470	0.283	0.492	0.615
MN2-INT4	600	0.255	0.425	0.269	0.134	0.271	0.345	0.253	0.417	0.445	0.256	0.468	0.596

(a) RetinaNet results on COCO

Model	Input	mAP						mAR					
		AP	$\mathrm{AP}^{0.5}$	$\mathrm{AP}^{0.75}$	AP^{S}	AP^{M}	AP^{L}	AR^{1}	AR^{10}	AR^{100}	AR^{S}	AR^{M}	AR^{L}
R50-FP32	800	0.377	0.593	0.409	0.220	0.415	0.489	0.316	0.513	0.541	0.364	0.580	0.678
R50-INT4	800	0.331	0.540	0.355	0.182	0.362	0.436	0.291	0.468	0.494	0.320	0.529	0.629
R34-FP32	800	0.358	0.576	0.384	0.211	0.390	0.461	0.307	0.496	0.526	0.348	0.564	0.611
R34-INT4	800	0.318	0.529	0.339	0.176	0.344	0.422	0.284	0.460	0.486	0.306	0.520	0.634
R18-FP32	800	0.322	0.538	0.340	0.180	0.347	0.419	0.286	0.466	0.494	0.326	0.524	0.630
R18-INT4	800	0.281	0.484	0.293	0.145	0.304	0.381	0.263	0.429	0.454	0.281	0.480	0.594
MN2-FP32	600	0.290	0.497	0.295	0.160	0.307	0.390	0.272	0.439	0.465	0.280	0.502	0.610
MN2-INT4	600	0.255	0.453	0.257	0.127	0.275	0.352	0.250	0.402	0.426	0.236	0.465	0.573

(b) Faster R-CNN results on COCO

Table 1: Performance of FQN detectors on the COCO benchmark. Standard metrics including mean average precision (mAP) and mean average recall (mAR) on `coco-2017-val` are reported. Models with “-INT4” suffix are fine-tuned and evaluated in 4-bits precision. Note that models with MobileNetV2 backbone use an input size of 600 pixels due to GPU memory limitation.

object detectors because of its rich annotations and challenging scenarios. In all our experiments, results are evaluated by standard COCO metrics including average precision and average recall on the bounding box detection task. To analyze the sources of the improvements produced by our approach, we also conduct a series of ablation studies on detection tasks using as backbones ResNet [10] and MobileNets [27] in 4-bit and 8-bit settings.

Training protocol Detectors are trained on COCO the data-set `coco-2017-train` partition using full-precision before quantization. Backbones are initialized with classification models pre-trained on the ImageNet dataset. Training is performed with synchronized SGD across 16 workers. The batch size on each worker is 2. The learning rate is warmed up to 0.04, then scaled by a factor of 0.1 at 30K and 80K steps.

Quantization-aware fine-tuning is performed on the same dataset, with detector weights and activations quantized to 4-bits. This procedure uses identical settings as full-precision training, except that the learning rate is fixed to 0.004. Activation ranges are measured at the 99.9% and 0.1% percentiles on 20 randomly sampled data batches from the training set. Finetuning stops after 40K steps, and results on checkpoints with the highest $\mathrm{mAP}^{0.5:0.95}$ on the `coco-2017-val` partition are reported.

All training and evaluation images are resized so that their shorter edges are 800 pixels. Images are augmented by random horizontal flipping during training, and no evaluation augmentations are performed. Note that FQN with MobileNetV2 backbone uses an input size of 600 pixels due to GPU memory limitations.

4.1. Main Results on COCO

We apply our proposed finetuning scheme to both one stage RetinaNet detectors and two stage Faster R-CNN detectors. ResNet-18, ResNet-32, ResNet-50, and the compact MobileNetV2 are used as backbones. Results of both full-precision baselines and 4-bit FQN on the `coco-2017-val` partition are listed in Table 1.

As shown, FQN can achieve acceptable accuracy loss on different detection frameworks and backbones. The 4-bit RetinaNet detector with MobileNetV2 backbone only suffers a 2.0% mAP loss compared to its full-precision base-

#	F	P	C	AP	AP$^{0.5}$	AP$^{0.75}$
0				0.317	0.503	0.337
1				0.197	0.348	0.198
2	✓			0.235	0.402	0.245
3		✓		0.222	0.381	0.228
4			✓	0.250	0.419	0.260
5	✓	✓		0.254	0.426	0.265
6	✓		✓	0.268	0.442	0.280
7		✓	✓	0.273	0.449	0.288
8	✓	✓	✓	**0.286**	**0.469**	**0.299**

(a) Ablations on 4-bits ResNet-18 RetinaNet

#	F	P	C	AP	AP$^{0.5}$	AP$^{0.75}$
0				0.317	0.503	0.337
1				0.296	0.475	0.312
2	✓			0.299	0.481	0.314
3		✓		0.313	0.499	0.344
4			✓	0.293	0.473	0.308
5	✓	✓		0.312	0.495	0.311
6	✓		✓	0.302	0.482	0.319
7		✓	✓	0.312	0.497	0.311
8	✓	✓	✓	**0.314**	**0.498**	**0.332**

(b) Ablations on 8-bits ResNet-18 RetinaNet

Table 2: Ablation studies on the COCO benchmark. In all subtables, row 0 is the FP32 baseline, row 1 is finetuned by methods in [17], and row 8 is our proposed FQN. F indicates batch norm freezing, P indicates using percentiles for activation statistics, and C indicates using channel-wise quantization.

Precision	F	AP	AP$^{0.5}$	AP$^{0.75}$
4 bits		0.226	0.384	0.235
4 bits	✓	**0.236**	0.400	0.245
8 bits		0.299	0.479	0.316
8 bits	✓	**0.300**	0.480	0.318

Table 3: Comparisons on FreezeBN strategy on different bit-width. Checking F means freezing BN in the entire finetuning process, otherwise only freezing BN in the final 10k steps as in [18].

γ	AP	AP$^{0.5}$	AP$^{0.75}$
0.9990	0.286	0.469	0.299
0.9973	**0.289**	0.469	0.308
0.9545	0.275	0.449	0.287

Table 4: Comparison of varying percentile γ durning calibrating activation ranges.

line. 4-bit ResNet-50 incurs 3.1% mAP loss when used in the RetinaNet detector, and 4.6% mAP loss when used in the Faster R-CNN detector. Considering ResNet and MobileNets are compact and widely used, these experiments indicates the robustness and generality of FQN.

Note that for all backbones, two stage Faster R-CNN detectors always incur higher mAP loss than one stage RetinaNet detectors. One possible reason is that the number of parameters in the region proposal network (RPN) and final prediction sub-nets of Faster R-CNN is smaller than the number of parameters in the classification and bounding box regression sub-nets of RetinaNet. Another possibility is the fully connected layers in Faster R-CNN detectors are more sensitive to quantization.

4.2. Ablation Studies

We also analyzed FQN by performing a number of ablation studies using ResNet-18 with a RetinaNet detector on COCO's `coco-2017-val` partition. We choose ResNet-18 as the backbone because it is a relatively small networks, and hence more sensitive to quantization, and because ResNet-18 has received great interest from the network compression community.

FQN shares many features with [17], which is designed for 8-bit networks. We therefore also report our reproduced 4 and 8-bit results based on [17]'s method.

Results are listed in Table 2. In both subtables in Table 2, row 0 is the FP32 full precision baseline, row 1 is our reproduced results for [17]. A check in the F column indicates batch normalization statistics were frozen during the entire fine tuning process, as described in §3.5.1. A check in the P column indicates activation ranges used percentiles instead of EMA statistics, as described in §3.4.1. Finally, a check in the C column indicates network weights were quantized on a per channel basis, as described in §3.3.

Freezing BarchNorm statistics As shown in rows 1 and 2 in Table 2a and Table 2b, freezing batch normalization statistics leads to a significant increase of 3.8% on 4-bit mAP, and a small improvement of 0.3% in 8-bit. This indicates that batch normalization in low bitwidth detectors are more likely to suffer instability from low quality activation statistics.

Recent work in [18] also indicated freezing batch normalization statistics helps quantized fine tuning. However, [18] suggested freezing statistics only in the last few thousand steps of training. We compared our method with theirs on both 4 and 8-bit settings. As shown in Table 3, freezing statistics during the entire fine tuning process leads to better performance, at least in detection tasks. One possible reason for this is that detectors are fine tuned in smaller mini-batches, so quantization noise introduced in detection models batch statistics are relatively strong.

#	Method	Act. calibration	mAP
0	*FP32 baseline*	–	0.317
1	Integer-only [17]	Moving average	0.197
2	Quant whitepaper [18]	Moving average	0.226
3	DoReFa-Net [35] foldBN	Clip to $[\{-1, 0\}, +1]$	0.039
4	XNOR-Net [24] foldBN	Moving average	0.244
5		Percentile	0.267
6	**Ours**	Percentile	**0.286**

Table 5: Baseline comparisons on 4-bit ResNet-18 RetinaNet.

Percentile based activation clamping Comparing rows 1 with 3 in Table 2a and Table 2b, we find that using percentile statistics to clamp activation values improves performance in both the 4 and 8-bit settings. The mAP gains are 2.5% and 1.7%, respectively.

To analyze the efficiency and robustness of percentile based statistics, we evaluate our model (shown in row #8 in Table 2a) by varying γ. We compared the default $\gamma = 0.999$ with $\gamma = 0.9973$ and $\gamma = 0.9545$. The latter two γ values represent clamping activation values within a range of $\pm 2\sigma$ and $\pm 3\sigma$, respectively. They lie around the mean activation value, assuming activation values are stochastic and follow a Gaussian distribution. As shown in Table 4, $\gamma = 0.9973$ yields slightly better performance, though the effect is small.

Channel-wise quantization scheme Comparing rows 1 with 4 in Table 2a and Table 2b, we see that using channel-wise quantization produces a 5.3% mAP gain in 4-bit settings. This significant improvement indicates the magnitude the variation between weights is an important factor for accuracy loss in low bitwidth scenarios.

4.3. Comparison with Previous Methods

We now compare our proposed FQN method with several existing methods, including integer-only detection [17], the quantization whitepaper [18], XNOR-Net [24] and DoReFa-Net [35], applied to ResNet-18 RetinaNet in 4-bit precision. We note that XNOR-Net and DoReFa-Net perform certain operations in floating point (*e.g.* batch normalization), while all operations in FQN are quantized. Thus, for a proper comparison we quantized all operations in XNOR-Net and DoReFa-Net. We also note that [24] only specified a calibration method for 1-bit activations. In our results we report on two 4-bit calibration methods using a moving average and our proposed percentile method, as shown in rows 4 and 5 of Table 5, respectively. As can be seen, the detection accuracy (mAP) of FQN is significantly better than all the baselines.

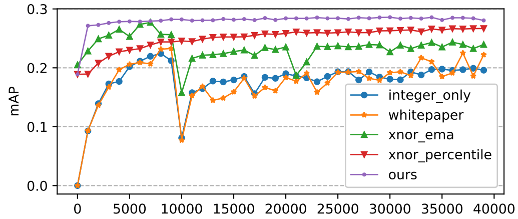

Figure 4: Fine-tuning curve for 4-bit ResNet-18 RetinaNet. The x-axis is the number of fine-tuning iterations. Note that methods which calibrate activation ranges using moving average delay activation quantization for 10K iterations, causing a trough. Also, methods using channel-wise quantization have higher starting accuracy.

FQN also improves the convergence speed compared to previous methods during quantization-aware fine-tuning. As shown in Figure 4, FQN converges much faster, the mAP of FQN nearly recovers to 0.27 after only 1K steps of fine-tuning, and is stable during the entire fine-tuning process.

5. Conclusion

In this paper, we propose FQN, a general quantization approach for low-precision, integer-only arithmetic inference. FQN supports end-to-end fully quantized training of complex object detection tasks. Compared to previous quantization methods, our approach produces a 4-bit model with performance very close to the 32-bit floating-point version, even on mobile friendly networks. We hope this approach and the observations in our experimental analysis can facilitate future quantization research and industrial vision applications on resource constrained devices.

References

[1] Hande Alemdar, Vincent Leroy, Adrien Prost-Boucle, and Frederic Petrot. Ternary neural networks for resource-efficient ai applications. *2017 International Joint Conference on Neural Networks (IJCNN)*, May 2017.

[2] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding, 2016.

[3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013.

[4] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. *arXiv preprint arXiv:1702.00953*, 2017.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[6] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[7] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[8] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns, 2016.

[9] 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.

[10] 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*, pages 770–778, 2016.

[11] Lu Hou and James T. Kwok. Loss-aware weight quantization of deep networks, 2018.

[12] 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 preprint arXiv:1704.04861*, 2017.

[13] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in neural information processing systems*, pages 4107–4115, 2016.

[14] 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.

[15] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[17] 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 *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

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

[19] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks, 2016.

[20] Tsung-Yi Lin, Piotr Dollár, Ross B Girshick, Kaiming He, Bharath Hariharan, and Serge J Belongie. Feature pyramid networks for object detection. In *CVPR*, volume 1, page 4, 2017.

[21] Tsung-Yi Lin, Priyal Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[22] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[23] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[24] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[25] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

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

[27] 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*, pages 4510–4520, 2018.

[28] Yi Wei, Xinyu Pan, Hongwei Qin, Wanli Ouyang, and Junjie Yan. Quantization mimic: Towards very tiny cnn for object detection. *Lecture Notes in Computer Science*, page 274–290, 2018.

[29] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. Training and inference with integers in deep neural networks. *arXiv preprint arXiv:1802.04680*, 2018.

[30] Penghang Yin, Shuai Zhang, Jiancheng Lyu, Stanley Osher, Yingyong Qi, and Jack Xin. Binaryrelax: A relaxation approach for training deep neural networks with quantized weights, 2018.

[31] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 365–382, 2018.

[32] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[33] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*, 2017.

[34] Aojun Zhou, Anbang Yao, Kuan Wang, and Yurong Chen. Explicit loss-error-aware quantization for low-bit deep neural networks. In *Proceedings of the IEEE Conference on*

Computer Vision and Pattern Recognition, pages 9426–9435, 2018.

[35] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[36] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2(6), 2017.