EasyQuant: Post-training Quantization via Scale Optimization

Di Wu[1], Qi Tan[2], Yongle Zhao[1], Ming Zhang[1], Ying Fu[3], and Debing Zhang[1]

[1] DeepGlint, Beijing, China
{diwu, yonglezhao, mingzhang, debingzhang}@deepglint.com
[2] OPEN AI LAB, Shanghai, China
qtang@openailab.com
[3] Beijing Institute of Technology, Beijing, China
fuying@bit.edu.cn

Abstract. The 8 bits quantization has been widely applied to accelerate network inference in various deep learning applications. There are two kinds of quantization methods, training-based quantization and post-training quantization. Training-based approach suffers from a cumbersome training process, while post-training quantization may lead to unacceptable accuracy drop. In this paper, we present an efficient and simple post-training method via scale optimization, named EasyQuant (EQ), that could obtain comparable accuracy with the training-based method. Specifically, we first alternately optimize scales of weights and activations for all layers target at convolutional outputs to further obtain the high quantization precision. Then, we lower down bit width to INT7 both for weights and activations, and adopt INT16 intermediate storage and integer Winograd convolution implementation to accelerate inference. Experimental results on various computer vision tasks show that EQ outperforms the TensorRT method and can achieve near INT8 accuracy in 7 bits width post-training.

Keywords: Post-training quantization, scale optimization, INT7 inference, ARM deployment

1 Introduction

Deep convolutional neural networks (CNN) have made considerable success in various computer vision tasks, including classification, detection, and recognition [15, 13, 30, 12, 20, 8]. However, it is not trivial to deploy CNN on computation-constrained devices, due to the huge computing power required by these models. Quantization is an essential technique to reduces CNN models' memory footprint and the amount of computation [11].

Low bit representation, e.g. 8 bits width or lower, typically leads to accuracy lost compared with float point 32 (FP32) models [16, 19, 21]. Although training-based methods could achieve higher accuracy compared with post-training approaches, they suffer from some drawbacks in real applications. For example,

training a quantized neural network is a time-consuming work, and it needs expert experience to guide the whole training process, which significantly influences the success of the work. Besides, in some scenarios, entire training data is not available to deploy the quantization model.

In this paper, we introduce an efficient and simple post-training quantization method via effectively optimizing the scales of weights and activations. The proposed scale optimization method is named EasyQuant (EQ). Specifically, we first formulate the quantized convolutional process as an optimization problem target at maximizing the cosine similarity between FP32 and INT8 outputs. This problem is solved by searching weights and activations scales alternately. For entire network optimization, we sequentially optimize scales layer by layer, and greedily obtain the optimal quantization scales for each layer. The scales of weights and activations are jointly optimized in each layer, and their scales of the next layer are optimized based on the quantized results of the previous layers. Besides, we adopt INT16 intermediate storage and integer Winograd algorithm to improve the inference latency on real hardware in context of 7 bits width. Finally, we verify our approach in different bits width settings on common computer vision tasks, including classification, detection, and recognition.

In summary, our main contributions are that

- We present a scale optimization method for the post-training quantization, which alternately searches weights and activations scales target, and can obtain comparable accuracy with the training-based quantization method.
- We implement the proposed post-training quantization method to a more efficient INT7 quantization inference framework, which improve the usage efficiency of intermediate INT16 storage.
- Extensive experiments on various computer vision tasks demonstrate that our scale optimization approach can achieve effective INT8 post-training quantization and near INT8 precision in the context of 7 bits width without finetuning. Besides, we implement and test the proposed EQ INT7 inference on real ARM platforms.

2 Related Work

Most works on quantization can be roughly divided into two categories, i.e., training-based quantization and post-training quantization. Training-based quantization usually applies sophisticated designs to train a low bit integer model from scratch or finetune a pre-trained FP32 model [18, 36, 38]. Post-training quantization often transforms a pre-trained network from float point to integer range with few data to calibrate applied scales [33, 2, 5]. In the following, we review the related work in detail, and discuss the applicability of different quantization methods on real hardware devices.

2.1 Training-based Quantization

Early works on training-based quantization often learn a quantization network at a more limited bit width (e.g. under 2 bits) [6, 27, 37]. These methods often

suffer from a huge accuracy drop due to the more limited bit width. Most recent works focus on higher bit width quantization to obtain the similar precision with that from FP32 models [22, 36, 4, 18]. Mishra *et al.* [22] proposed wide reduced-precision networks to overcome the accuracy drop by increasing the number of filters and obtain better precision in 4 bits width. Zhou *et al.* [36] proposed to incrementally quantize part of weights of network to reduce the training difficulties involved by quantization in 5 bits width. In [18, 4], they both adopted the optimizing quantization thresholds by training associated with task loss to guide the training process in 4 bits width. These works all attempt to train less than 8 bits width models from scratch, which are difficult to obtain the similar accuracy with the 8 bits width models. Besides, they need specific hardware and software to work, for most of applied devices only support general INT8 quantized models. Thus, training-based quantization in lower bits width (4 bits width) is seldom adopted in nowaday industry applications.

Another type of training-based method is quantization aware training (QAT) [19, 26]. [19] proposed the QAT method as a supplementary approach to regain some lost accuracy induced by INT8 quantization. QAT simulated the quantization noise in conventional training processes and trained the models with normal methods in float point 32 range, which were always used to finetune from an FP32 model [19]. Jain *et al.* [26] improved QAT by making the threshold trainable in the regular training process, which could be seen as a training-based scale optimization method. Since these works do not apply sophisticated designs on training process, they can mainly deal with 8 bits width quantization and scarcely consider the effect from the intermediate storage. Besides, training a quantized model from scratch has high time complexity and needs the expert experience of both target tasks domain and quantization domain, especially in more complex tasks such as objection detection and face recognition.

2.2 Post-training Quantization

Due to the drawbacks mentioned above, INT8 post-training quantization becomes the major trend in most real quantization applications. Researches on this field include TensorRT (TRT) from Nvidia (Migacz, 2017) [21] and Tensorflow Lite from Google [19]. TRT [21] adopted Kullback-Leibler divergence (KLD) minimization to calibrate quantization thresholds for activations, and utilized the maximum absolute values as thresholds for weights quantization. Tensorflow Lite [19] utilized the maximum absolute values as thresholds for activations, and joined a per-channel quantization method with the maximum absolute values as thresholds to quantize the weight. These two methods quantize the activation and weight scales according to either simple maximum absolute values or statistical characters, which still suffer unaccepted performance drop in some pre-trained networks.

Later, Yoni Choukroun *et al.* [5] improved the quantization method by treating each layer quantization process as a constrained optimization process solved by alternate golden section search. The whole search process is very time-consuming due to the large search space of quantized tensors. Banner *et al.* [2] optimized

the thresholds for activations by theoretically deriving the optimal clipping values. The analytical expressions were based on strict assumptions of activation distribution, which were rarely held in real models. In this work, we jointly optimize each layers' scales of both weights and activations, and target for reducing the quantization effect of convolutional output. Our method is more robust for various models' situations for not requiring specific hardware and further assumption.

2.3 Industrial applicability

One of the significant benefits of quantization techniques is that it could reduce the inference latency on edge devices, which have limited computation power. However, in the literature of quantization, seldom of works discuss the applicability of quantization methods, and it is non-trivial in real quantization deployment. In order to reduce inference latency in general edge device, e.g. ARM CPU, quantization method provides both quantized weights and activations for convolution operation in inference time. Some approaches, e.g. [32, 24, 6] which only quantize weights to fixed point, are hard to adopted to accelerate the real inference process. Besides, some methods [27, 33, 5] indeed quantize both weights and activations to fixed point, but they usually need particular hardware or software to facilitate the implementation of quantized inference. This hinders the wide usage of these approaches. Our method quantizes both weights and activations without the need of specialized hardware. Furthermore, we propose an instruction-level optimization on INT7 quantization inference to accelerate normal INT8 inference which could easily be deployed in general hardware, e.g. the real ARM platforms.

3 The Proposed Method

In this section, we first formulate the linear quantization process. Then, the proposed scale optimization method is introduced in detail. Moreover, the designs of the INT7 post-training inference are discussed.

3.1 Linear Quantization Formulation

The linear quantization process could be denoted as function $Q(X, S)$, where $X \in \mathbf{R}$ is a tensor and S is a positive real number scale factor. Quantized results $Q(X, S) \in \mathbf{Z}_b$, where $\mathbf{Z}_b$ is the b bits width integer domain. Linear quantization function $Q(X, S)$ includes three sub-processes, i.e., scaled, round and clipped. The linear quantization formulation of input tensor X and scale factor S could be represented as

$$Q(X, S) = Clip(Round(X \cdot S)), \tag{1}$$

where *Round* means that the scaled input tensors are rounded to integers using ceiling rounding, and "$\cdot$" denotes element-wise product. In different implementations of linear quantization, it can adopt different types of rounding (round,

ceil, or floor). *Clip* denotes that elements in the tensor that exceed the ranges of the quantized domain are clipped.

Let us define a quantized L-layer neural network as $\{A_l, W_l, S_l\}_{l=1}^{L}$. A_l, W_l, and S_l are the l-th layer input activation, weight, and quantization scale factors in FP32 range, respectively. Specifically, quantization scale factors (S_l) contain two parts. The scale number for the input activations in the l-th layer is denoted as S_l^a. The scale number for the weights in the l-th layer is denoted as S_l^w. S_l^a is a non-negative real number applied for each elements in feature maps. S_l^w is a non-negative real number for input weights. For the convenience of discussion, we illustrate the proposed method in per-layer quantization scheme. For per-channel quantization scheme, each filter in W_l should have independent scale number. Besides, we denote the l-th output feature map of the pre-trained FP32 model as O_l and its corresponding quantized inference output feature map as $\hat{O}_l$.

Therefore, the whole linear quantization forward convolution and dequant operation in the l-th layer can be described as

$$\hat{O}_l = \frac{Q(A_l, S_l^a) * Q(W_l, S_l^w)}{S_l^a \cdot S_l^w}. \tag{2}$$

where $*$ denotes convolution operation. The original output of the l-th layer can be expressed as

$$O_l = A_l * W_l. \tag{3}$$

From Equation (2), it can be seen that the scale factors actually control the thresholds clipped in quantization process, which affects the cosine similarity of convolutional results between original output feature map (O_l) and quantization inference feature map ($\hat{O}_l$) to a great extent. Therefore, our target function focuses on optimizing the scale factors for both weights (S_l^w) and activations (S_l^a) and improving the similarity between O_l and $\hat{O}_l$.

3.2 Scale Optimization

Quantization process of a neural network model could be divided into each layer, where weights and activations are quantized respectively and prepared for convolutional operation. The quantization of a convolutional layer is shown in Figure 1.

The overall Information is passed and transformed from the first layer of the neural network to the end. It introduces inevitable noise into the final outputs of the neural network. Most post-training works [21, 31, 1] use the Kullback-Leibler divergence (KLD) method to calculate scale factors of activation for each layer. Typically, they use around 1000 calibration data to approximate the input activation distribution of each layer [21]. For the scales of weights in each layer, they usually use the absolute maximum as thresholds to determine scales because of the bigger weights values always dominate the results. They separately optimize the scale for each activation and does not optimize weight scales, which easily results in error accumulation. It also ignores the fact that the optimization of

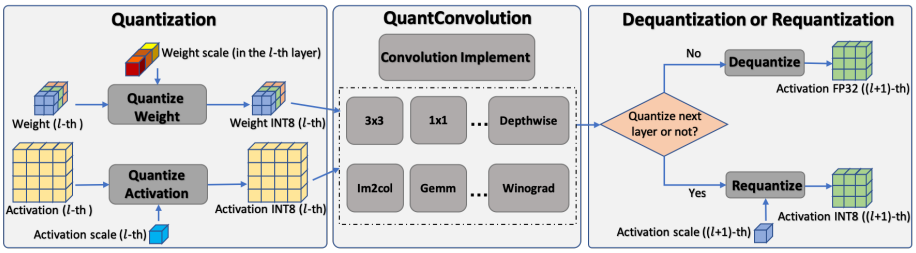


Fig. 1: The quantization process of a convolutional layer. The whole process contains three parts: quantization of input activations and weights, convolutional operation implementation and requant/dequant operation.

similarity between original and quantized distribution, which can not guarantee the promotion of the similarity between original and quantized convolutional outputs. Furthermore, these approaches mainly design INT8 quantization on activations and weights, and need INT32 bits to save all the intermediate results in the CNN inference process. This takes more computational time and limits their application fields.

To solve this problem, we present a simple but efficient scale optimization method, which jointly optimize both scales of activation and weights targeting at the loss on the similarity between original and quantized convolutional outputs. Moreover, we also propose INT7 quantization inference to further accelerate traditional INT8 quantization and deploy it on real ARM platforms.

Optimal Scale for Each Layer According to Equation (2), we optimize scale factors $\{S_l\}_{l=1}^{L}$ for a pre-trained convolutional neural network $\{A_l, W_l\}_{l=1}^{L}$, where activations $\{A_l\}_{l=1}^{L}$ are generated from a given calibration dataset D with N samples. It is much less than the common training dataset. In the l-th layer of the neural network, our approach can be expressed to maximize output feature map cosine similarity

$$\begin{aligned} \max_{S_l} \quad & \frac{1}{N}\sum_{i=1}^{N}\cos\left(O_l^i, \hat{O}_l^i\right), \\ \text{s.t.} \quad & S_l \in \mathbf{R}^+. \end{aligned} \tag{4}$$

where S_l scales the activations and weights to the fixed bits width range. Generally, the larger the scale is, the more high value elements will be saturated to the maximum of the quantized domain. While the smaller the scale is, the more low value elements will be rounded to zeros.

We adopt alternating optimization method here to solve this problem in two folds. First, S_l^a is fixed, and solve S_l^w for weight scales adjustment. Second, S_l^w is fixed, and solve S_l^a to finetune activations scales. S_l^w and S_l^a are alternately

optimized until $\cos(O_l^i, \hat{O}_l^i)$ converges or excesses the time limitation. Here, for fast convergence, S_l^w and S_l^a are initialized in terms of the maximum of weigths or activations respectively. For search space of S_l^w and S_l^a, we linearly divide interval of $[\alpha S_l, \beta S_l]$ into n candidate options and conduct a simple search strategy on them. In experiments, hyper-parameters α, β and n are robust for various tasks with $\alpha = 0.5$, $\beta = 2$ and $n = 100$. More advanced search method could be applied to search the candidate scales, while we find that in experiments, simple search strategy is more robust for the irregular fluctuation of target function. Therefore, these strategy with reasonable initialization is applied in our optimization process to solve the problem. When optimizing weight scales in per-channel quantization scheme, where S_l^w is a collection of c (the number of filters in this layer) dimension, we can adjust each kernel's independent scale in parallel in one search process.

Optimal Scale for the Whole Network In the previous section, we present how to optimize one layer activation scale S_l^a and weight scale S_l^w with optimal scale in a layer. We apply our approach layer by layer sequentially for a whole convolutional neural network.

Algorithm 1: Scale optimization for the whole convolutional neural network

Input: model weights set $\{W_l\}_{l=1}^L$, model original input activation set and output set$\{A_l, O_l\}_{l=1}^L$ generated by calibration D
Output: Optimal scales for L layers $\{S_l\}_{l=1}^L$
Data: Calibration dataset D with N samples

Initialize $\{S_l\}_{l=1}^L$ for L layers
while *Convergence or excess time limitation* **do**
  for $l = 1 : L$ **do**
    for S_{lk}^w *in interval* $[\alpha S_l^w, \beta S_l^w]$ **do**
      record S_{lk}^{w*} with maximum $\frac{1}{N}\sum_{i\in\mathbb{D}} \cos(O_l^i, \hat{O}_l^i)$
  Fix $\{S_l^{w*}\}_{l=1}^L$ optimize $\{S_l^a\}_{l=1}^L$ and update A_{l+1} with $\hat{A}_{l+1}$
  for $l = 1 : L$ **do**
    for S_{lk}^a *in interval* $[\alpha S_l^a, \beta S_l^a]$ **do**
      record S_{lk}^{a*} with maximum $\frac{1}{N}\sum_{i\in\mathbb{D}} \cos(O_l^i, \hat{O}_l^i)$
return *Optimal searched scales for L layers* $\{S_l\}_{l=1}^L$

For each layer, input activation $\hat{A}_l$ is obtained from the current model, where all former layers are quantized and optimized. The output feature map O_l^i is collected from the original model without quantization. The benifits of adopting this greedy strategy is that (a) dividing the optimization of entire networks to subproblems helps reduce the huge search space of optimization, and (b) the optimization of current layer O_l^i is taking accumulated noise from all former layers

into consideration. Above all, for an L layers convolutional neural network, we adopt our proposed approach sequentially to obtain the optimal scales $\{S_l\}_{l=1}^{L}$. The proposed scales optimization for post-training layers in the integral model quantization is summarized in Algorithm 1.

3.3 INT7 Post-training Inference

Here, we implement our efficient designs on INT7 post-training inference. Detailed explanations are also discussed here to give more insight on the significance of 7 bits width.

Quantization relies heavily on the characteristic of hardware in order to take advantage of low bits inference. In regular convolutional calculations, there are many matrix multiply-add operations which could be implemented by Signed Vector Multiply-Add Long instruction (SMLAL) and Signed Add and Accumulate Long Pairwise instruction (SADALP) in ARM NEON instruction sets. SMLAL instruction multiply and add 8-bit elements to produces 16-bit results while SADALP instruction adds two adjacent 16-bit results into 32-bit accumulators [17]. Using these two instructions, we could efficiently implement convolutional process on ARM architecture. The data flow mentioned above is described in Figure 2.

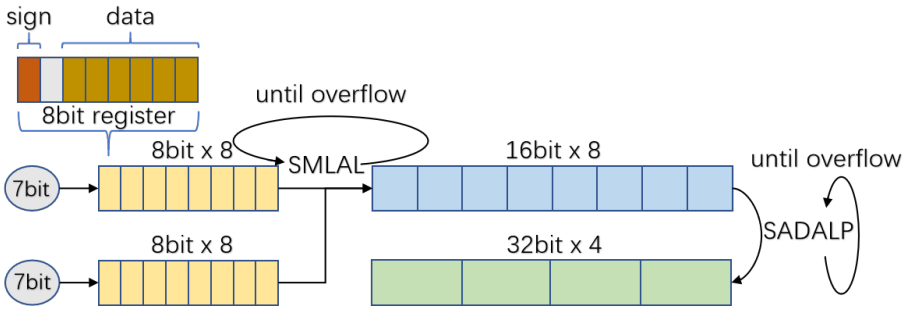


Fig. 2: Proposed INT7 inference data flow in ARM architecture. Two main NEON instructions applied in implementation of convolution are SMLAL and SADALP. In case of overflow, 7 bits inference could do 8 times SMLAL instruction (2 times for 8 bits) safely before adding intermediate 16 bits results to 32 bits register by SADALP.

In INT8 quantization, the safe solution is to use 32-bit register to store the intermediate variable. Nevertheless, before ARM V8.2-A architecture in Cortex-A processor, there is no instruction to store the multiplication result of two 8-bit register numbers into a 32-bit register. Therefore, the general solution uses SMLAL first to multiply-add 8-bit elements and produces 16-bit results, and then uses SADALP to add two adjacent 16-bit results into 32-bit accumulators

[17]. For regular convolution operation, 8 bits (8 bits signed integer) inference could only make $\left\lfloor \frac{2^{15}-1}{(2^7-1)^2} \right\rfloor = 2$ times SMLAL operations without any overflow which is ineffective [31]. Our proposed 7 bits (7 bits signed integer) inference could do $\left\lfloor \frac{2^{15}-1}{(2^6-1)^2} \right\rfloor = 8$ times SMLAL without overflow which is more effective compared with INT8 inference. It can make more SMLAL operations before adding them to 32-bit accumulators compared with 8 bits inference. In general, INT7 post-training method could make more use of CPU efficiency [17] , which is important in industrial application.

4 Experimental Results

In this section, we first describe the setting applied in our following experiments, including the method compared in our experiments, quantization implementation details and hyper-parameters in our EQ. Then, we conduct experiments on different bit-width settings, including most common INT8 bits width, proposed INT7 bits width, and less than 7 bits width.

4.1 Settings

Our method belongs to post-training approach due to no requirement of re-training. To make fair comparison, we compare our EQ with the post-training approach TRT [21, 25, 10] [4]. 1000 samples are used for calibration data on TRT method in all experiments.

In our experiments, we alternately optimize weights and activations for one round and use 50 random samples to optimize the scales targeting at convolutional outputs. For the number of samples, we have tested our method that more samples give almost same results in 8 bit width and can provide a little better results in 7 bit width. For example, there are about $0.1\% \sim 0.15\%$ gain of MobileNet v1 on ImageNet2012 with INT7 quantization, by using 1000 samples instead of 50. Considering the much time consuming of 1000 samples and limited computation resource we have, we use 50 samples on all the experiments, which have outperformed TRT method with 1000 samples. And we search 100 values uniformly distributed between $[0.5S_l, 2S_l]$ for each scales in one layer.

According to implementations, all methods are tested on the open-source quantization framework NCNN [31]. Given n bit quantization, we constrain the absolute maximum values under $2^n - 1$, which saturates the values beyond the threshold. In TRT approach, for activation scales, 1000 samples randomly drawn from the respective training set are used to generate the activations scales $\{S_l\}_{l=1}^{L}$.

[4] In the post-training methods, the TRT approach is widely adopted for optimizing activation scales in real industrial deployment, for most inference frameworks utilize the TRT method to optimize activation scales [21, 31, 1]. In particular, the TRT method compared in our experiments refer to current version, which include per-channel quantization

4.2 INT8 Post-training Quantization

8 bits post-training quantization is adopted widely, when deploying the convolutional model in a general edge device or specific server optimized engine [21, 10, 31, 16]. Here, we test our method in context of 8 bits quantization. concretely, we compare our EQ with widely used TRT [21, 25, 10] and quantization aware training (QAT) [19] approach. We re-implement TRT quantization scheme with NCNN framework based on their open resources [31, 21, 25]. For QAT comparison, we directly adopt the results in [19] for reference.

Table 1: Top 1 classification accuracy (%) on ImageNet2012 validation dataset for different convolutional models in context of both INT8 and INT7 post-training quantization. Bold results show the better results between our EQ and TRT.

Models	FP32	INT8		INT7	
		TRT	EQ	TRT	EQ
SqueezeNetV1.1 [15]	56.56	56.24	**56.28**	54.88	**56.08**
MobileNetV1 [13]	69.33	68.74	**68.84**	66.97	**68.26**
VGG16 [30]	70.97	70.95	**70.97**	70.92	**70.96**
ResNet50 [12]	75.20	75.04	**75.13**	72.78	**75.04**

We verify the effectiveness of our method among different tasks, including image classification (ImageNet2012)[5], object detection (VOC2007), and face recognition (seven stardard). Performances are evaluated on ImageNet 2012 validation dataset [28], Pascal VOC object detection 2007 test dataset [9] and seven common face recognition datasets [14, 23, 35, 29, 34, 3]. For model architectures, more Computationally efficient backbone MobileNet V1 [13] are chosen among all the tasks. Meanwhile, other classical models e.g. SqueezeNetV1.1 [15], ResNet50 [12] and VGG16 [30] are also tested in our experiments.

For 8 bits quantization, the results of the classification on ImageNet2012 [7], detection on VOC2007, and seven standard face recognition tasks are shown in Tables 1, 2 and 3, respectively. It can be seen that our EQ outperforms TRT method on per-channel INT8 quantization across all three kinds of tested tasks and all the convolutional neural network architectures. For example, for MobileNet V1 backbone models, EQ could further gain 0.1 %, 1.6% and average 0.37 % precision compared with TRT respectively. It implies, in some complex task, e.g. object detection, EQ could reduce more precision loss compared with TRT method.

[5] Most previous quantization methods [6, 27, 2, 36, 33, 5, 19] only test the performance on image classification tasks, e.g. on the ImageNet2012 dataset.

Table 2: Object detection on VOC2007 task for SSD [20] models with backbone SqueezeNet and MobileNet V1. Mean average precision (mAP) is evaluated in FP32, TRT and our EQ (both in INT8 and INT7 post-training quantization).

Models	FP32	INT8		INT7	
		TRT	EQ	TRT	EQ
SqueezeNet-SSD	62.00	61.45	**62.05**	60.01	**61.62**
MobileNet-SSD	72.04	69.79	**71.39**	63.88	**68.79**

Table 3: Verification performance (%) for InsightFace [8] model MobileFaceNet on seven most common validation datasets. Comparisons are on TRT and EQ with INT8 and INT7 quantization.

Test dataset	FP32	INT8		INT7	
		TRT	EQ	TRT	EQ
lfw	99.45	99.36	**99.48**	99.28	**99.36**
agedb_30	95.78	95.23	**95.38**	95.03	**95.73**
calfw	95.05	94.76	**94.88**	**94.75**	94.68
cfp_ff	99.50	99.50	**99.61**	99.44	**99.60**
cfp_fp	89.77	89.17	**90.04**	88.47	**89.87**
cplfw	86.45	85.58	**86.03**	85.91	**86.76**
vgg2_fp	90.64	89.70	**90.50**	89.64	**90.44**

We also compare our method with more complex QAT approach in 8 bit width. The results of MobileNetV1 and ResNet50 models on ImageNet classification are shown in Table 4. The results show that EQ could get competitive precision compared with more complex QAT method in context of INT8 quantization. It is noticeable that for ResNet50 EQ could even outperform QAT method.

Table 4: Top 1 classification accuracy (%) on ImageNet2012 validation dataset for MobileNetV1 and ResNet50 with EQ INT8 method and QAT approach. Bold results show the better results between EQ and QAT.

Methods	MobileNetV1		ResNet50	
	FP32	INT8	FP32	INT8
EQ	69.33	68.84	75.20	**75.13**
QAT [19]	70.90	**70.70**	75.20	75.00

4.3 INT7 Post-training Quantization

Both requiring activations and weights constrained in 7 bits keep less information from the original model and will add more quantization errors into models. Therefore, it require more advanced quantization scheme in INT7 post-training quantization. In this section, we test our method with TRT in context of 7 bits width. Besides, we also verify the latency of our proposed INT7 quantization inference on real hardware. The experiments are conducted on the same three tasks and respective models with INT8 post-training quantization.

Table 5: The latency (ms) performance on RK3399, whose inside is a 1.5 GHz 64-bit Quad-core ARM Cortex-A53. #k means k threads.

Models	TRT-INT8(#1)	EQ-INT7(#1)	TRT-INT8(#4)	EQ-INT7(#4)
SqueezeNetV1.1	180	**120**	66	**44**
MobileNetV1	234	**189**	65	**57**
VGG16	3326	**2873**	1423	**1252**
ResNet50	1264	**993**	415	**300**

The results of these experiments are shown in Tables 1, 2, and 3. In these INT7 quantization experiments, TRT method suffers a sharp accuracy drop compared with FP32 model. For the classification on ImageNet2012 and detection on VOC2007, the accuracy lost is even more significant in some models, e.g. MobileNet V1 (2.36% and 8.16% in ImageNet2012 and VOC2007, respectively). Our EQ shows substantial superiority that it can obtain much better precision on all the models across three tasks (largest gap occurred in MobileNet V1 1.06% and 3.25% in ImageNet2012 and VOC2007 respectively). Our EQ performs much better compared with TRT method in INT7 quantization, and our proposed method could still achieve near FP32 accuracy in 7 bits width.

As we point out in section 3.3, INT7 quantization reduces 2 bits storage for activations and weights, and give solid supports for Int16 intermediate storage. Its inference could be more efficient compared with INT8 inference. In our EQ INT7 quantization, the summation of 8 multiplication results could be saved directly into INT16 without overflow. Using INT16 registers is not only much faster than using INT32 registers, but also efficiently reduces the amount of memory access which is also very important. Concretely, about 20%-33% computational cost could be saved on various ARM platforms. We test the INT7 post-training inference latency based on RK3399 [6]. The results are provided in Tables 5 and 6. It can be seen that our proposed INT7 inference scheme has less latency on general edge devices.

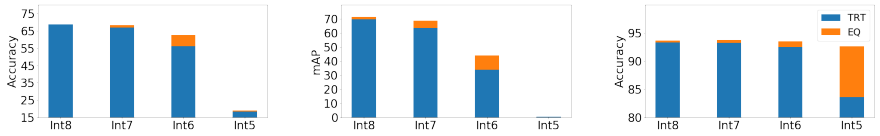
[6] RK3399 is a low power, high performance processor based on ARM architecture

Table 6: The latency (ms) performance on RK3399, which inside is a 1.8 GHz 64-bit Dual-core ARM Cortex-A72. #k means k threads.

Models	TRT-INT8(#1)	EQ-INT7(#1)	TRT-INT8(#2)	EQ-INT7(#2)
SqueezeNetV1.1	79	**57**	54	**37**
MobileNetV1	105	**84**	56	**46**
VGG16	1659	**1385**	1034	**849**
ResNet50	559	**463**	338	**262**

4.4 Comparison on Less than 7 Bits Width

We also conduct experiments with less than 7 bits width on classification, detection and face recognition. Taking MobileNet V1 based model as an example, the results are shown in Figure 3. For some tasks (classification on ImageNet2012 and detection on VOC2007), when the bits width is constrained to less than 7 bits, the precision dropped sharply on these tasks, as shown in Figure 3(a) and 3(b). In particular, on the VOC2007 tasks, the mean average precision (mAP) drops sharply when the bits width is constrained to less than 7 bits. The visu-



(a) (b) (c)

Fig. 3: Comparison between TRT and EQ in less than 7 bits width on classification, detection and Face recognition. (a) Top 1 classification accuracy (%) for MobileNet V1 models tested on ImageNet2012 validation dataset of different bit widths. (b) mAP (%) for SSD models with MobileNet V1 backbone on VOC2007 test dataset of different bit widths. (c) Mean verification accuracy (%) for InsightFace model MobileFaceNet on seven test datasets of different bit widths.

alizations of 6 bits width MobileNet SSD model quantized by TRT method and our EQ approach are shown in Figure 4. It can be seen that our EQ is still better than TRT method.

However, In face recognition task, It can be seen that even lower than 7 bits, our EQ still outperforms TRT method in all bits width we tested, especially, EQ with 6 bits width on insight face model still keeps near FP32 precision on seven face recognition datasets, as shown in Figure 3(c). In real application of post-

Fig. 4: Visualizations (Mobilenet V1 SSD model) of different types of detection errors in 6 bits quantization. (a) False negative of detection in single object scenario. (b) False negative of detection in multiple objects scenario. (c) Inaccurate bounding box regression. (d) Wrong type of detection.

training quantization, one could choose the appropriate bits width to balance the acceleration of deployment and lost precision introduced by quantization.

5 Conclusion

In this paper, we present a scale optimization based method to boost post-training quantization both from the perspective of preserved quantization precision and deployment latency. Our proposed INT7 quantization inference does not rely on any specific framework and could be applied to any linear post-training scheme to both increase the inference speed and accuracy. It benefits real industrial INT8 post-training quantization without complex quantization aware finetuning. Experiments show that our proposed method can obtain better precision of quantized models in various tasks and convolutional architectures. By designing Int16 intermediate storage and integer Winograd algorithm, we can further improve the inference speed with less precision decreasing compared with TRT methods on real hardware platform.

References

1. Alibaba: MNN: a lightweight deep neural network inference engine. https://github.com/alibaba/MNN (2019)
2. Banner, R., Nahshan, Y., Soudry, D.: Post-training 4-bit quantization of convolution networks for rapid-deployment. arXiv preprint arXiv:1810.05723 (2018)
3. Cao, Q., Shen, L., Xie, W., Parkhi, O.M., Zisserman, A.: Vggface2: A dataset for recognising faces across pose and age. In: Proceedings of IEEE International Conference on Automatic Face Gesture Recognition. pp. 67–74 (2018)
4. Choi, J., Wang, Z., Venkataramani, S., Chuang, P.I.J., Srinivasan, V., Gopalakrishnan, K.: Pact: Parameterized clipping activation for quantized neural networks. arXiv preprint arXiv:1805.06085 (2018)

5. Choukroun, Y., Kravchik, E., Yang, F., Kisilev, P.: Low-bit quantization of neural networks for efficient inference. arXiv preprint arXiv:1902.06822 (2019)
6. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. arXiv preprint arXiv:1602.02830 (2016)
7. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Li, F.F.: Imagenet: A large-scale hierarchical image database. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 248–255 (2009)
8. Deng, J., Guo, J., Xue, N., Zafeiriou, S.: Arcface: Additive angular margin loss for deep face recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4690–4699 (2019)
9. Everingham, M., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A.: The pascal visual object classes challenge. International journal of Computer Vision **88**(2), 303–338 (2010)
10. Google: Tensorflow lite guide. https://www.tensorflow.org/lite/guide
11. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149 (2015)
12. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 770–778 (2016)
13. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
14. Huang, G.B., Mattar, M., Berg, T., Learned-Miller, E.: Labeled faces in the wild: A database forstudying face recognition in unconstrained environments. arXiv preprint arXiv:1708.08197 (2017)
15. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$ 0.5mb model size. arXiv preprint arXiv:1602.07360 (2016)
16. Intel: Introduction to low-precision 8-bit integer computations. https://intel.github.io/mkl-dnn/ex_int8_simplenet.html
17. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D.: 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)
18. Jung, S., Son, C., Lee, S., Son, J., Han, J.J., Kwak, Y., Hwang, S.J., Choi, C.: Learning to quantize deep networks by optimizing quantization intervals with task loss. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4350–4359 (2019)
19. Krishnamoorthi, R.: Quantizing deep convolutional networks for efficient inference: A whitepaper. arXiv preprint arXiv:1806.08342 (2018)
20. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: Ssd: Single shot multibox detector. In: Proceedings of European Conference on Computer Vision. pp. 21–37 (2016)
21. Migacz, S.: 8-bit inference with tensorrt. In: Proceedings of Conference on GPU Technology. p. 7 (2017)
22. Mishra, A., Nurvitadhi, E., Cook, J.J., Marr, D.: Wrpn: wide reduced-precision networks. arXiv preprint arXiv:1709.01134 (2017)

23. Moschoglou, S., Papaioannou, A., Sagonas, C., Deng, J., Kotsia, I., Zafeiriou, S.: Agedb: the first manually collected, in-the-wild age database. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. pp. 51–59 (2017)
24. Nayak, P., Zhang, D., Chai, S.: Bit efficient quantization for deep neural networks. arXiv preprint arXiv:1910.04877 (2019)
25. nvidia: Nvidia tensorrt. https://developer.nvidia.com/tensorrt
26. Rajagopal, V., Ramasamy, C.K., Vishnoi, A., Gadde, R.N., Miniskar, N.R., Pasupuleti, S.K.: Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks. arXiv preprint arXiv:1903.08066 (2019)
27. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: Imagenet classification using binary convolutional neural networks. In: Proceedings of European Conference on Computer Vision. pp. 525–542 (2016)
28. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: Imagenet large scale visual recognition challenge. International journal of Computer Vision **115**(3), 211–252 (2015)
29. Sengupta, S., Chen, J.C., Castillo, C., Patel, V.M., Chellappa, R., Jacobs, D.W.: Frontal to profile face verification in the wild. In: Proceedings of IEEE Winter Conference on Applications of Computer Vision. pp. 1–9 (2016)
30. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
31. Tencent: NCNN: a high-performance neural network inference framework optimized for the mobile platform. https://github.com/Tencent/ncnn (2017)
32. Zhang, T., Zhu, L., Zhao, Q., Shin, K.: Neural networks weights quantization: Target none-retraining ternary (tnt). arXiv preprint arXiv:1912.09236 (2019)
33. Zhao, R., Hu, Y., Dotzel, J., De Sa, C., Zhang, Z.: Improving neural network quantization without retraining using outlier channel splitting. In: Proceedings of International Conference on Machine Learning. pp. 7543–7552 (2019)
34. Zheng, T., Deng, W.: Cross-pose lfw: A database for studying cross-pose face recognition in unconstrained environments. Technical Report **5** (2018)
35. Zheng, T., Deng, W., Hu, J.: Cross-age lfw: A database for studying cross-age face recognition in unconstrained environments. arXiv preprint arXiv:1708.08197 (2017)
36. Zhou, A., Yao, A., Guo, Y., Xu, L., Chen, Y.: Incremental network quantization: Towards lossless cnns with low-precision weights. In: Proceedings of International Conference on Learning Representations (2017)
37. Zhu, C., Han, S., Mao, H., Dally, W.J.: Trained ternary quantization. In: Proceedings of International Conference on Learning Representations (2017)
38. Zhuang, B., Shen, C., Tan, M., Liu, L., Reid, I.: Towards effective low-bitwidth convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7920–7928 (2018)