Supplementary Material for Mixed-Precision Neural Network Quantization via Learned Layer-wise Importance

Chen Tang[1*], Kai Ouyang[1*], Zhi Wang[1,4†], Yifei Zhu[2], Wen Ji[3,4], Yaowei Wang[4], and Wenwu Zhu[1]

[1] Tsinghua University
[2] Shanghai Jiao Tong University
[3] Institute of Computing Technology, Chinese Academy of Sciences
[4] Peng Cheng Laboratory

1 Experimental Setups

For layer-wise quantization, we use the bit-width options $\mathcal{B}$ ={2,3,4,5,6} for its weights and activation. For weight only quantization, we set the bit-width of activations to 8. We use the pre-trained model as initialization and keep the first and last layer at 8 bits. Based the method in Section 3.4 (main paper), we train 5 epochs for ResNet18 and MobileNet and 1 epoch for ResNet50, both with 0.01 learning rate (LR). Then, we extract the layer-wise importance indicators to apply mixed-precision search upon different constraints according to Equation. 4 (main paper). The hyper-parameter α for ResNet18, ResNet50, MobileNetv1 is 3.0, 2.0, 1.0 respectively. After searching, we quantize the models with the searched policies and finetuning them 90 epochs, both using the cosine LR scheduler and the SDG optimizer with 0.04 LR and 2.5×10^{-5} weight-decay, the first 5 epochs are used for warm-up.

2 Details of One-time Training for Importance Derivation

According to Section 4.3 of the main paper, the parameters θ of the whole network, including weights and all importance indicators, at training step t are updated through

$$\theta_t = \theta_{t-1} - \eta \times g_t \tag{1}$$

where η is the learning rate and

$$g_t = \frac{\partial}{\partial \theta_t} \underbrace{\left(\mathbb{E}_{\mathcal{S} \sim \Gamma(\mathcal{A})} \left[\mathcal{L}(f(\mathbf{x}; \mathcal{S}, \theta_t^{(\mathcal{S})}), \mathbf{y}) \right] + \mathcal{R}(\theta) \right)}_{\text{ideal loss function } \Psi(\cdot;\theta)}, \tag{2}$$

where $\mathcal{A}$ is the search space, $\mathcal{S}$ is the mixed-precision quantization policy (*i.e.,* the bit-width of each layer), $\Gamma(\mathcal{A})$ is the prior distribution of $\mathcal{S} \in \mathcal{A}$, $\theta_t^{(\mathcal{S})}$ is the

* Equal contribution: {tc20, oyk20}@mails.tsinghua.edu.cn
† Corresponding author: wangzhi@sz.tsinghua.edu.cn

parameters of policy $\mathcal{S}$ (*i.e.,* quantized weights and the importance indicators of $\mathcal{S}$), $\mathcal{R}(\theta)$ is the regularization term (*e.g.,* L2 regularization), $\mathcal{L}(\cdot)$ is the task loss (*e.g.,* cross entropy). We call $\Psi(\cdot;\theta)$ as the *ideal loss function* due to the expectation is time prohibitive to solve in training, however, it can be approximated by the Monte-Carlo Sampling. That is,

$$\nabla_{\theta_t}\mathbb{E}_{\mathcal{S}\sim\Gamma(\mathcal{A})}\left[\mathcal{L}(f(\mathbf{x};\mathcal{S},\theta_t^{(\mathcal{S})}),\mathbf{y})\right]=\frac{1}{K}\sum_{\mathcal{S}_k\sim\Gamma(\mathcal{A})}^{K}\left[\nabla_{\theta_t^{(\mathcal{S}_k)}}\left(\mathcal{L}(f(\mathbf{x};\mathcal{S}_k,\theta_t^{(\mathcal{S}_k)}),\mathbf{y})\right)\right]. \tag{3}$$

Here, we use uniform distribution as our prior distribution since its simplicity and effectiveness in our experiments, thus $\theta_t^{(\mathcal{S}_k)}$ is the parameters of uniformly sampled policy $\mathcal{S}_k$. Also, it is obvious that there are several key policies that seriously affect other policies. These key policies occur when the bit-widths of

Algorithm 1: One-time Training for Importance Derivation

Input: Full-precision Network $\boldsymbol{f}$; Bit-list, for example, [6, 5, 4, 3, 2]; Learning rate η.
Output: A trained network with importance indicators for each layer.

Initialize each layer's importance indicators of each *bit-width* in Bit-list;
for *t=1,...,T* **do**
  Get batch of input data $\mathbf{x}$ and label $\mathbf{y}$;
  Clear gradients of weights and importance indicators, *optimizer.zerograd()*;
  $n = length$(Bit-list);
  for *i=0,...,n-1* **do**
    /* Compute gradient of n key policies in this For loop. */
    Get the biw-witdh $\mathbf{b}$ = Bit-list[i];
    Switch $\boldsymbol{f}$'s each layer to bit-width policy $\mathbf{b}$;
    Get the parameters $\theta_t^{(\mathcal{S}_i)}$ of i-th policy $\mathcal{S}_i$;
    Compute outputs under i-th policy $\hat{y} = f(x;\mathcal{S}_i,\theta_t^{(\mathcal{S}_i)})$;
    Compute loss, $loss = \mathcal{L}(\hat{y}, y)$;
    Compute gradient through BP, $\nabla_{\theta_t^{(\mathcal{S}_i)}} = loss.backward()$;
  end
  /* Compute the gradient of uniformly sampled policy. */
  A bit-width is selected independently and randomly for each layer;
  Get the parameters $\theta_t^{(\mathcal{S}_{n+1})}$ of $(n+1)$-th policy $\mathcal{S}_{n+1}$; // Notice that $\mathcal{S}_{n+1}$ is a uniformly sampled policy.
  Compute outputs under n-th policy $\hat{y} = f(x;\mathcal{S}_{n+1},\theta_t^{(\mathcal{S}_{n+1})})$;
  Compute loss, $loss = \mathcal{L}(\hat{y}, y)$;
  Compute gradient through BP, $\nabla_{\theta_t^{(\mathcal{S}_{n+1})}} = loss.backward()$;
  /* Use joint gradient to update the whole network. */
  Aggregate above gradient to get g_t;
  Update $\boldsymbol{f}$'s parameters $\theta_t = \theta_{t-1} - \eta * g_t$
end

each layer are equal. For example, when optional bit-widths $\mathcal{B} = \{b_0, ..., b_{n-1}\}$, there are n key policies corresponding to the bit-width of each layer equal to $b_0, ..., b_{n-1}$. It is indispensable to ensure these key policies are sampled at each training step, because their bit-widths construct the bit-width options $\mathcal{B}$.

Therefore, at training step t, we first compute the gradient of n key policies. Then, we uniformly sample K policies and compute their gradient to make sure different bit-widths in different layers can communicate with each other. Inspired by one-shot NAS [3] and due to we already sample n policies, we set $K = 1$. Finally, we aggregate the above gradient to update the parameters of whole network.

Given all this, we build our importance indicators training algorithm. The pseudocode of our importance indicators training method is shown in Alg. 1.

3 Additional Comparisons

We add HAWQ [2] and DiffQ [1] as additional comparisons, both accuracy results in Tab. 1 come from the papers directly. Since the DiffQ and HAWQ pipelines contain search and fine-tuning time and are hard to measure the search time only, we measure the total time costs through their official training settings as the efficiency results. Please notice that DiffQ do not quantize the activations, and our method can quantize the activations to low bits while providing higher accuracy/compress rate and about 3× speedup. Red column: higher is better; Blue column: lower is better.

Table 1. Accuracy results. "W-b" represents the bit-width of weights. "A-b" represents the bit-width of activations. "MP" represents the mixed-precision quantization is supported. "FP" represents the full-precision (float 32) is used. "Top-1 Quant" represents the top-1 accuracy of quantized model. "Top-1 Drop" represents the top-1 accuracy degradation.

Method	W-b	A-b	Top1 Quant	Top1 Drop	Model Size
HAWQ	MP	MP	75.48%	1.91%	7.96MB
DiffQ	MP	FP	76.3%	0.8%	8.8MB
Ours	MP	MP	76.9%	0.6%	7.97MB

Table 2. Efficiency results. We measure the total time consumption instead of searching time only here.

Method	Total Costs (Search + Fine-tuning, GPU-hours)
HAWQ	540
DiffQ	613
Ours	191

4 Bit-width Assignment Visualization

We visualize the bit-width assignment for MobileNet and ResNet50 in Fig. 1 to understand the behavior of our method. We can observe that the top layers tend to be assigned more bit-widths due to their extraction of low-level features. In particular, in MobileNet, DW-convs are assigned higher bit-width due to their sensitivity to quantization. Thus, we can conclude that our method does achieve not only better performance and but also a reasonable bit-width assignment through the learned importance indicators.

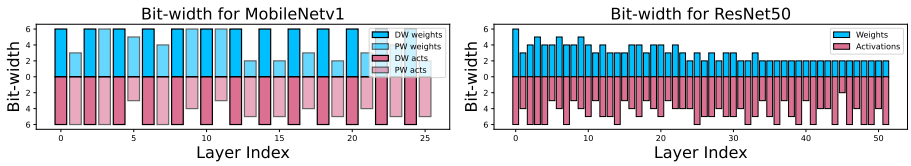

Figure 1. Bit-width for MobileNet and ResNet50.

References

1. Défossez, A., Adi, Y., Synnaeve, G.: Differentiable model compression via pseudo quantization noise. arXiv preprint arXiv:2104.09987 (2021)
2. Dong, Z., Yao, Z., Gholami, A., Mahoney, M.W., Keutzer, K.: Hawq: Hessian aware quantization of neural networks with mixed-precision. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 293–302 (2019)
3. Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., Sun, J.: Single path one-shot neural architecture search with uniform sampling. In: European Conference on Computer Vision. pp. 544–560. Springer (2020)