7 Appendix

7.1 Details for Problem Setups

Baselines. While our method aims to solve the QAT problem, we extensively compare our *BiTAT* against various Post-training Quantization (PTQ)- or QAT methods: BRECQ [18] is a PTQ method that considers weight dependencies using the Hessian of the task loss. DBQ [8] is a QAT method based on continuous relaxation of the quantizer function. EBConv [3] conditionally selects appropriate binarized weights based on the task information. Bi-Real Net [20] adds residual connections to propagate full-precision values, preventing information loss to activation quantization. Real-to-Bin [22] constrains a loss term at the end of each convolution to minimize the output discrepancy between the full-precision and the quantized model. LCQ [35] devises a trainable quantization function in order to reduce the quantization error. MeliusNet [2] proposes a new architecture that better propagates full-precision values throughout the network. ReActNet [21] is the state-of-the-art binary quantization method, which additionally adopts residual connections, and element-wise shift operations before/after the activation and the sign operation. Note that DBQ, LCQ, and MeliusNet keep some crucial layers of MobileNet in full-precision, leading to inefficiency at evaluation time.

Training. Following the setup from ReActNet [21], we quantize all layers' weights and activations except the initial and final layers. We use the Adam optimizer [16]. For the ImageNet experiment, a learning rate is 0.002 and 0.0002 for quantization training and the fine-tuning, respectively, with linear learning rate decay. We set batch size as 512 both the quantization phase and the fine-tuning phase is done for 5 epochs per layer. For the CIFAR-100 experiment, a learning rate is 3×10^{-4} for quantization training and the fine-tuning with linear learning rate decay. We set batch size as 800 and both the quantization and fine-tuning are done with 40 epochs per layer. For all experiments, we set $\lambda = 100$, and $\gamma = 10^{-5}$, which notes that simple choice of the hyperparameters for our regularization terms is sufficient to show impressive performance. The number of input dimension groups is set $k = 256$, applying the grouped weight correlation to layers with input dimensions smaller than k.

Inference. In deployment, the highly efficient XNOR-Bitcounting operations can be used for the convolutional layers, also used in existing neural network binarization works [5, 30, 21].

7.2 Extension to Convolutional Layers

Let us consider a convolutional layer of size $n_{out} \times n_{in} \times k \times k$, where n_{in} and n_{out} are the number of input and output channels, respectively, and k is the kernel size. We define the set $\mathcal{P}_{\boldsymbol{x}}$ as the set of all patches of size $n_{in} \times k \times k$ extracted from the training image $\boldsymbol{x}$. This patch-extracting operation is sometimes called `im2col` or `F.unfold` in PyTorch.

A convolutional layer applied to $\boldsymbol{x}$ can be thought of as a fully-connected layer individually applied to all patches in $\mathcal{P}_{\boldsymbol{x}}$ and then concatenated:

$$\boldsymbol{w} * \boldsymbol{x} = \{\mathrm{Reshape}_{(n_{in}k^2)\times(n_{out})}(\boldsymbol{w})^\top \mathrm{vec}(\boldsymbol{p})\}_{\boldsymbol{p}\in\mathcal{P}_{\boldsymbol{x}}}, \tag{14}$$

where $*$ denotes the convolution operation, $\mathrm{Reshape}_{shape}(\cdot)$ denotes the reshaping of the tensor into the specified *shape*, and $\mathrm{vec}(\cdot)$ denotes the flattening operation. Each pixel of the output feature map corresponds to a matrix multiplication between a patch and the weight matrix. Therefore, we can analogically apply the same transformation as explained in Section 3 to convolutional layers.

7.3 Details on Cross-layer Dependency

In this section, we further explain the detailed experimental setting for Figure 3. We take the standard MobileNetV2 [31] model and train it to convergence on the CIFAR-100 dataset with standard SGD with a weight decay. Then, we add noise to the same pretrained model parameters before evaluating the test accuracy based on the following two different ways:

1. **Layer-dependent noise addition.** We first compute the covariance of the input to the first layer and perform PCA using obtained covariance values to compute $\widetilde{\boldsymbol{w}}^{(1)}$ in Equation 4. Now, we add independent gaussian noise with varying scales to the top five rows of $\widetilde{\boldsymbol{w}}^{(1)}$. Next, we sequentially repeat the process to the consecutive layers, and after that, we evaluate

METHOD	Accuracy (%)	Q_{orig}	Q_{ours}
REACTNET [21]	65.51 ± 0.74	**13.35**	475.94
BITAT (Ours)	**68.36 ± 0.45**	39.77	**434.92**

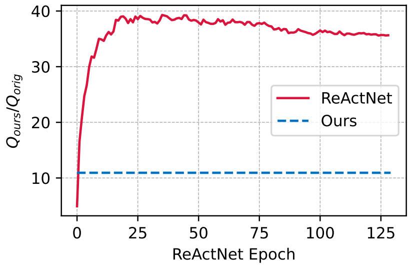

Figure 8: **Additional ablation studies. Left:** The comparison of the final Q_{orig} (Equation 2) and Q_{ours} (Equation 6) values in ReActNet and BiTAT. **Right:** The evolution of the ratio of Q_{ours} to Q_{orig} in ReActNet, in comparison to the final ratio in our BiTAT. Cross-layer dependencies not considered in both computations.

the model performance, which is shown in solid red lines. The same process is done but with the bottom five rows of each layer instead of the top five, shown in solid blue lines.

2. **Layer-independent noise addition.** Before adding noise to model parameters, we compute the covariance of the input values for all layers. Next, we perform PCA and compute $\widetilde{\boldsymbol{w}}^{(l)}$ with Equation 4 per layer using these initial covariance values. That is, a layer cannot reflect the weight change through noise addition in others layers, as different from the first approach. Independent gaussian noise with varying scales is added to the top five rows of $\widetilde{\boldsymbol{w}}^{(l)}$ for each layer, and then the performance of the model is evaluated, shown in red dashed lines. The same is done with the bottom five rows of $\widetilde{\boldsymbol{w}}^{(l)}$, shown in blue dashed lines.

7.4 Additional Analysis

This paper suggests that the proposed quantization loss on disentangled weights is a better indicator for prediction accuracy than the general quantization loss (Equation 2), which is evident in multiple validation analyses and the superior model performance in our BiTAT as described in the main text. Here, we provide the quantitative analysis to show that ReActNet [21] fails to minimize the quantization loss on disentangled weights while our BiTAT successfully does. In Figure 8 Left, we show the Q_{orig} (Equation 2) and Q_{ours} (Equation 6 w/o ℓ_1 norm) between the initial full-precision weights of the pre-trained model and the obtained binarized weights from ReActNet and BiTAT. Q_{orig} represents the naive MSE between the full-precision weights and the binarized weights. Q_{ours} represents the dependency-weighted MSE between the full-precision weights and the binarized weights. Note that, in this analysis, we obtain $\boldsymbol{s}$ and $\boldsymbol{V}$ for each layer from the initial pre-trained model by Equation 10 to compute Q_{ours} and neglect the weight dependency across different layers, which is hard to be computed analytically.

We observe that while the value of Q_{orig} is lower in ReActNet than in BiTAT, Q_{ours} is higher in ReActNet than in BiTAT. As shown in Figure 8 Right, the ratio $r = Q_{ours}/Q_{orig}$ in ReActNet (Red) drastically increases at the beginning stage and is maintained in a high degree until the completion of the quantization-aware training, compared to the r value of the model obtained by BiTAT (Blue dashed). While disregarding the first few epochs of ReActNet training, where the accuracy is very low, ReActNet's r value dominates that of BiTAT. The value slowly decreases as the ReActNet training proceeds, but never reaches the level of BiTAT, demonstrating the inefficiency of the ReActNet training procedure compared to ours.

7.5 Limitations

We consider two limitations of our work in this section. First, our BiTAT framework is built based on a sequential quantization strategy, which progressively quantizes the layers from the bottom to the top. Therefore, the training time of our algorithm depends on the number of layers in the backbone network architecture. While we have already validated the cost-efficiency of our proposed method against ReActNet using MobileNet (26 stacked layers) in Figure 7 Left, we might spend more training time quantizing all layer weights for the backbones, composed of much more layers like ResNet-1001 (1001 stacked layers). Next, our method focuses on the cross-layer weight dependency within each neural block, including a few consecutive layers. We thereby avoid the excessive computational cost of obtaining the relationship across all layers in the backbone architecture, yet we consider it a tradeoff between accurate dependency and computation budgets.