Weight Combine and α-Scaling for Robust PTQ on VGG16

1. Motivation: Why Weight Combine?

- Post-Training Quantization (PTQ) uses a small bit-width (e.g., 4-bit) for weights and activations.
- A single symmetric quantization range must cover:
  - A few large values (outliers),
  - And many small but important values.
- If the dynamic range is not well aligned:
  - Large weights dominate the scale,
  - Small weights collapse to 0 or ± 1 after quantization,
  - Leading to noticeable accuracy drop or high MSE.
- **Goal of weight combine / α-scaling:**
  - Re-parameterize each layer so that the effective weight distribution is more quantization-friendly,
  - While keeping the floating-point function as close as possible to the original VGG16.

2. Conv–BN Fusion as a Pre-Step

We first fuse each convolution + batch normalization pair into an effective linear layer.

- Original layer in VGG16:

$$z = \mathrm{Conv}(x; W) \quad \Rightarrow \quad y = \mathrm{BN}(z; \mu, \sigma^2, \gamma, \beta).$$

- After standard BN fusion, we get:

$$W_{\text{fused}} = \gamma \frac{W}{\sqrt{\sigma^2 + \epsilon}}, \qquad b_{\text{fused}} = \beta - \gamma \frac{\mu}{\sqrt{\sigma^2 + \epsilon}}.$$

- The layer becomes:

$$y = \mathrm{Conv}(x; W_{\text{fused}}, b_{\text{fused}}).$$

- All later quantization and scaling steps operate on $(W_{\text{fused}}, b_{\text{fused}})$.

3. Weight Combine via α-Scaling

We introduce a scalar parameter α to re-parameterize the fused weight.

3.1 Conceptual Re-parameterization

- Start from the fused layer:
$$y = xW_{\text{fused}} + b_{\text{fused}}.$$
- Introduce a scalar $\alpha > 0$ and define:
$$\widehat{W} = \alpha\, W_{\text{fused}}, \qquad \widehat{b} = \alpha\, b_{\text{fused}}.$$
- If we scale the next layer (or normalization) correspondingly, this can be seen as a **re-parameterization** that keeps the overall function almost unchanged, but changes the distribution seen by quantization.

3.2 Quantization-Oriented View

In practice, we use α as a **quantization knob**:

- Define a quantizer $\mathcal{Q}(\cdot)$, e.g., symmetric 4-bit:
$$W_{\text{int}} = \text{round}\Big(\frac{W}{s_w}\Big), \quad W_q = s_w\, W_{\text{int}}.$$
- Instead of quantizing W_{fused} directly, we quantize the scaled version:
$$W_q^{(\alpha)} = \frac{1}{\alpha}\, \mathcal{Q}\big(\alpha\, W_{\text{fused}}\big).$$
- The effective float weight is still in the same scale as W_{fused}, but the quantization grid is now controlled by α.

Key idea:

- α changes where the quantization levels sit relative to the real-valued weights.
- By choosing a good α, we can:
  - Reduce the maximum relative error on large weights,
  - Avoid collapsing too many small weights to zero,
  - Improve the MSE between quantized and full-precision outputs.

4. Application to VGG16 and 8×8 Hardware Mapping

- Backbone: **VGG16 on CIFAR-10**, with a modified "Part1" including an 8×8 squeezed convolution layer.
- Hardware target: an 8×8 systolic array with 4-bit weights and 4-bit activations.
- In this 8-in / 8-out convolution layer:
  - We fuse Conv+BN to obtain $(W_{\text{fused}}, b_{\text{fused}})$.
  - Apply weight combine (α-scaling) on the fused kernel before quantization.
  - Use this quantization-friendly kernel to generate integer input / weight / psum files for the RTL core.
- In our project, we successfully implemented this weight combine scheme on VGG16 and integrated it into the 8×8 systolic-array testbench.

5. Key Takeaways (Poster Bullets)

- **Weight combine** uses a scalar α to re-parameterize Conv+BN weights and make them more quantization-friendly in VGG16.
- We fuse Conv+BN first, then apply α-scaling and 4-bit quantization on the fused kernel of the 8×8 target layer.
- The method is:
  - Simple to implement in PyTorch,
  - Easy to integrate with our 8×8 systolic hardware flow,
  - Complementary to Hadamard-based outlier smoothing and other PTQ techniques.
- We have **successfully applied** this pipeline on a VGG16 model and used the resulting integer data to drive our RTL core testbench.