On the Trade-offs Between Neural Network Compression Techniques

Muhammad Usman [1] **Qasim Ayub** [1] **Usman Ahad** [1]

Abstract

Model compression remains one of the key challenges in the deployment of highly effective models, including deep neural networks, on resource-constrained devices. Compression techniques, including Pruning, Quantization, and Knowledge Distillation, aim to reduce model size and improve efficiency while maintaining accuracy. In this paper, we study the advantages and trade-offs of these methods and their applications. We analyze unstructured and structured Pruning strategies, demonstrating the impact of sparsity levels and fine-tuning on model performance and inference times. We evaluate quantization approaches, including post-training and quantization-aware training, investigating how mixed precision and outlier mitigation approaches preserve accuracy. Lastly, we examine Knowledge Distillation (KD) techniques, reviewing how different teacher-student configurations and feature matching methods influence student model generalization. Through comprehensive experiments on vision benchmarks, we provide considerations in the practicality of compressed models and selection of compression strategies to achieve optimal trade-offs between accuracy, size, and computational efficiency. The code for this paper can be found here.

1. Introduction

Deploying deep neural networks on resource-constrained devices requires careful management of model size, computational cost, and inference efficiency. Model compression techniques provide effective solutions to these challenges, primarily through pruning, quantization, and knowledge distillation (KD).

Pruning reduces the number of parameters by removing redundant weights or entire neurons/filters. Unstructured pruning offers fine-grained control but can be hardware-inefficient, whereas structured pruning improves hardware utilization at the potential cost of flexibility.

Quantization lowers the numerical precision of weights and activations, converting 32-bit floating-point values to formats such as int8 or int4, thereby reducing memory footprint and accelerating inference. Post-Training Quantization (PTQ) is straightforward but may degrade performance at very low bit-widths, while Quantization-Aware Training (QAT) enables the model to adapt to precision constraints, albeit with additional computational overhead.

Knowledge Distillation allows a smaller student model to learn from a larger teacher model through softened output probabilities or intermediate feature representations. Techniques such as Logit Matching, Hint-based Distillation, and Contrastive Representation Distillation each offer distinct benefits, including improved generalization, transfer of invariances, or richer inter-class relational information. However, KD also introduces considerations such as teacher selection, training stability, and alignment with the student's architecture.

In this study, we aim to evaluate these three compression approaches in a holistic manner, comparing their relative strengths and limitations across different scenarios. Our analysis considers multiple dimensions: the convenience and practicality of implementing each method, potential advantages and drawbacks in real-world deployment, and their effectiveness under constrained computational budgets. Additionally, we examine whether these techniques can be combined, exploring potential synergies, such as pruning followed by quantization or distillation combined with quantization, to achieve enhanced efficiency without sacrificing performance.

2. Pruning

2.1. Methodology

In Pruning (LeCun et al., 1989), model parameters are set to zero values by applying a mask M.

$$\tilde{\theta} = M \odot \theta$$

[1]Department of Computer Science, LUMS, Lahore, Pakistan. Correspondence to: Muhammad Usman <27100046@lums.edu.pk>, Qasim Ayub <27100168@lums.edu.pk>, Usman Ahad <27100041@lums.edu.pk>.

where $M_i = 0$ indicates a pruned parameter. Pruning is optimized when

$$\min_{\theta,M} \; E[\mathcal{L}(f_{\tilde{\theta}}(X), Y)] \text{ such that } \|M\|_0 \leq (1-a)\|\theta\|_0$$

where $\mathcal{L}$ is the loss function and $\|M\|_0$ enforces a given sparsity ratio a.

The target sparsity and choice of parameters to prune can dictate model performance. This choice involves grouping parameters and ranking their importance. In unstructured pruning, the groups are individual parameters θ_{ij} and in structured pruning, parameters of the same model substructure form groups.

We investigate the application of structured and unstructured pruning in model compression by applying pruning to the VGG-11 architecture (Simonyan & Zisserman, 2014). Using the CIFAR-100 dataset (Krizhevsky, 2009), we assess the effects of pruning on model performance, size, and inference time.

First, we form a baseline by finetuning a VGG-11 ImageNet model without pruning. We crop images to 224-pixel resolution and normalize using ImageNet values. Following previous literature (Han et al., 2015), we use Stochastic Gradient Descent (SGD) with batch size of 128, weight decay of 0.0005, momentum of 0.9, and cosine-annealing learning rate schedule with initial learning rate of 0.04 to train for 10 epochs.

To implement unstructured pruning, we conduct a sensitivity scan by individually pruning the baseline model layers to determine its impact on model performance [8]. Here, we rank parameters based on their absolute value and prune the lowest ranking parameters. In order to achieve 70% overall sparsity, We allocate pruning percentages proportionally to individual layers based on their robustness to increased sparsity, parameter count, and depth. We further finetune this model for 5 epochs following the same settings as above and also prune the finetuned model to re-achieve 70% sparsity. We evaluate three unstructured pruning variants: the pruned baseline, the finetuned-after-pruning model, and the pruned–finetuned model.

For structured pruning, we implement channel pruning with 30% convolutional sparsity by grouping output channel parameters for each convolutional layer of the network. Output channels are ranked by their L2 norms, and the lowest-ranked ones are pruned by removing both the selected channels and their corresponding inputs in the next layer. As in the unstructured version, we conduct a sensitivity scan [9], allocate proportional pruning percentages, and finetune the pruned model for 5 epochs. We evalaute the pruned baseline and the finetuned-after-pruning models.

Table 1. Test accuracy (%) across pruning strategies. 'FT' denotes fine-tuning.

Baseline	**Unstructured**	**+FT**	**+FT+Re-prune**	**Structured**	**+FT**
74.33	72.73	72.64	72.28	54.93	71.64

2.2. Results

Performance

Table 1 shows that unstructured pruning with 70% overall sparsity results in only a 1.6% decrease on the strong baseline performance of 74.33%, demonstrating that a large fraction of parameters can be removed with a minor performance drop. This indicates that the parameters removed contributed very little the discriminability of the model, which is explained by our choice of ranking. As we prune parameters with the lowest magnitudes, we select parameters which are the closest, or equal, to zero. This explains why unstructured pruning with high sparsity sacrifices little performance. In our experiment, finetuning the non-structurally pruned model does not increase performance, which remains similar at 72.64%, although the sparsity is lost. Literature proves that repeated finetuning and repruning is effective at restoring accuracy (Han et al., 2015), but our results show that this is not always the case. This could be attributed to strong baseline performance or sub-optimal finetuning hyperparamater selection in our experiment.

Structured pruning shows a substantial decrease in pruned-baseline performance with accuracy of 53.93% at 30% convolutional channel sparsity (Table 1). This suggests that removing entire channels impacts performance more significantly than removing individual parameters, even though fewer total parameters are removed in our experiment. Although we prune channels with the lowest L2 norms, these may contain important parameters. Because structured pruning removes entire filters and convolutional kernels, it can eliminate key components that the network depends on for accurate discrimination. In our experiment, this loss is largely recovered by finetuning the pruned model, which increases accuracy to 71.64%, 2.69% lower than the baseline model. This finetuning allows the model to relearn lost representations within the smaller, pruned architecture, proving that structured pruning can be applied effectively with a small performance loss.

Attention

The Grad-CAM visualizations (Selvaraju et al., 2017) of the final convolutional layers of each model show the effect of pruning on attention (Figure 1). For the first image (true class: lizard), the pruned models retain the attention patterns of the baseline model, focusing on the limbs of the lizard as discriminating information. This contributes to correct classification by each model, including the non-finetuned-

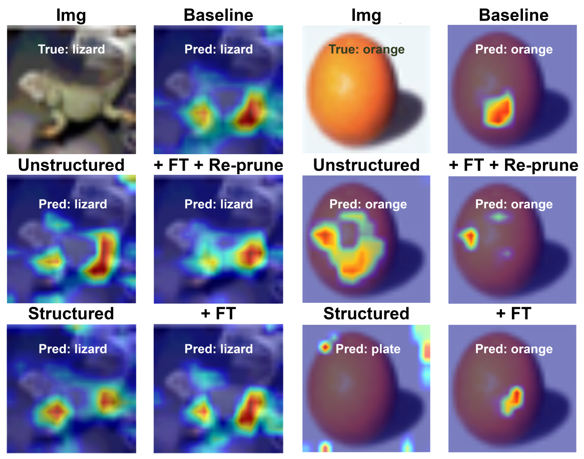

Figure 1. Grad-CAM visualization (final convolutional layer) and prediction comparison across pruning strategies

Table 2. Model size and per-input inference time

	Baseline	**Unstructured**	**Structured**
Size (MiB)	492.77	147.86	476.01
CPU (ms)	73.15	73.33	57.62
CUDA (ms)	5.94	5.60	4.20

pruned models. The attention map from the non-finetuned structured model resembles the baseline less so than that of its unstructured counterpart, showing that channel-pruning disrupts discriminative representations more so than unstructured pruning. This disruption is recovered by finetuning, as the Structured + FT model shows very high similarity to the baseline. The second example (true class: orange) shows that pruning can drastically affect attention patterns for a given layer. The unstructured model shows enhanced activation on the class object compared to the baseline, highlighting that this method can reduce noise in the feature maps, allowing remaining neurons to produce clearer, more focused activations. After channel pruning, the model no longer focuses on the class object, leading to a misprediction. This shows that there can be significantly different changes to a model depending on pruning strategies. We again show that finetuning is able to recover from the effects of structured pruning, leading to more class-object attention and a correct prediction.

Compression

Table 2 shows that unstructured pruning achieves a substantial reduction in model size from 492.77MiB to 147.86MiB, reflecting the sparsity (70%) of individual parameters, but does not show an improvement in CPU or GPU inference time. This can be attributed to the fact that the sparse weight matrices with randomly distributed zeros do not allow for more efficient memory access or computation than the original non-sparse matrices.

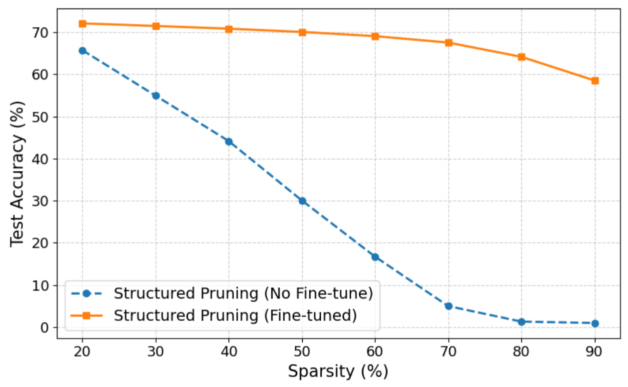

Figure 2. Fine-tuning slows the rate of accuracy decrease due to higher sparsity in structured pruning.

Structured pruning with 30% conolutional channel pruning reduces model size by only 3.6% (Table 2) because the convolutional layers of VGG-11 do not contain the majority of the parameters. The main advantage of structured pruning is in inference time, especially on CPU, where it achieves a 21.23% lower time over the baseline model. This increased efficency is the result of more efficient computation and memory usage due to removed channels and highlights the practicability of using structured pruning where compute is limited. As a whole, the compression and efficiency results of our experiment show the trade-off between decreased model size and increased efficiency depending on the choice of pruning strategy.

Target Sparsity Selection

Figure 2 shows that the performance of structured pruning without finetuning is inversely proportional to the level of convolutional sparsity, with higher sparsity removing more critical filters and causing greater accuracy degradation. As more channels are removed, the network's ability to discriminate between classes decreases with equal proportion. Once more, finetuning the pruned model supresses this effect. With finetuning, accuracy stays above 70% until 50% sparsity. Surprisingly, the model retains 58%+ accuracy at 90% sparsity, highlighting the robstuness of the model. Remarkably, only 10% of the original convolutional channels are sufficient for the network to retain significant predictive capability. These results highlight the destructive nature of structured pruning when applied directly to a baseline model, emphasizing the necessity and effectiveness of finetuning to restore performance.

2.3. Discussion

Our experiments demonstrate that both pruning strategies are effective for model compression but each comes with distinct trade-offs. Unstructured pruning achieves very high sparsity with minimal performance loss, making it suitable for scenarios where storage is the primary constraint. However, this approach fails to reduce inference times, making it more suitable to settings with higher compute available. Structured pruning, on the other hand, does lead to faster inference and more predictable memory usage but does not achieve the highest sparsity levels without higher performance loss.

Future literature should continue to consider both approaches and provide a framework for hybrid approaches which target high-performance, high-sparsity pruning with minimal storage and compute requirements. It may also be beneficial to apply specific pruning techniques to certain scenarios, such as structured pruning in edge devices, and validate the extent to which inference times can be reduced on machines with limited compute power.

3. Quantization

3.1. Methodology

This section explores model compression through weight quantization at various bit widths, namely *FP16*, *BF16*, *INT8*, and *INT4*. We use the VGG11 architecture pretrained on *ImageNet1K_V1* weights and fine-tune it for 5 epochs. The fine-tuning process uses the Cross-Entropy loss and the SGD optimizer with a base learning rate of *0.04*, weight decay of *0.0005*, and momentum of *0.9*. A *CosineAnnealingLR* scheduler with $\eta_{\min} = 0.00005$ is also used to dynamically adjust the learning rate across epochs.

3.1.1. Post-Training Quantization vs. Quantization-Aware Training

We evaluate quantized models using *Post-Training Quantization (PTQ)* applied to the fine-tuned model. We then perform *Quantization-Aware Training (QAT)* on pretrained model (without fine-tuning) to determine whether training with simulated quantization effects can recover PTQ performance loss.

For most experiments, the `torchao` library is used with weight-only quantization. Since hardware kernels are not optimized for *int4*, we use the *INT4CPULayout* implementation in *torchao*. Additionally, we perform manual quantization to simulate *int4* within the *int8* domain to increase performance. This is done by scaling linear layer weights as:

$$q = \text{clip}\Big(\text{round}\Big(\frac{x}{s}\Big), q_{\min}, q_{\max}\Big), \quad \hat{x} = s \cdot q,$$

where

$$s = \frac{\max(|x|)}{q_{\max}}, \qquad q_{\min} = -2^{b-1}, \qquad q_{\max} = 2^{b-1} - 1.$$

3.1.2. Mixed Precision Quantization

To optimize efficiency while preserving accuracy, we apply *mixed-precision quantization*, assigning different precisions to different layers. Sensitive layers are kept at higher precision (e.g., *FP32*), while less sensitive ones are quantized to lower bit-widths (e.g., *INT8*). Uniform *INT8* quantization using `torchao` will serve as the baseline for this part. We evaluate two approaches: (i) a simple mixed-precision scheme and (ii) an adaptive scheme guided by activation statistics.

In the **Simple Mixed-Precision** setup, convolutional and final classifier layers (`classifier[6]`) are kept in *FP32*, while intermediate fully connected layers (`classifier[0,3]`) are quantized to *INT8* using weight-only quantization. This design preserves critical layers in full precision while compressing intermediate ones, balancing accuracy and storage. Accuracy, latency, and model size are recorded for comparison.

In the **Adaptive Mixed-Precision** configuration, per-layer precision is assigned using activation variance:

$$\sigma_\ell^2 = \text{Var}(a_\ell),$$

where higher variance indicates greater sensitivity. We keep more sensitive layers in higher precision while quantizing the others. Using the 30th and 70th percentile thresholds ($p_{\text{low}}, p_{\text{high}}$), layers are quantized as:

$$\sigma_\ell^2 < p_{\text{low}} \Rightarrow \text{INT8}$$

$$p_{\text{low}} \le \sigma_\ell^2 < p_{\text{high}} \Rightarrow \text{FP16}$$

$$\sigma_\ell^2 \ge p_{\text{high}} \Rightarrow \text{FP32}.$$

Convolutional layers remain in *FP32* due to their typically high variance. We use a *FP16Wrapper* module in quantization to correctly cast between *FP16* and *FP32* during inference to mitigate unexpected errors. This adaptive approach preserves accuracy in sensitive layers while reducing overall size. Finally, we evaluate accuracy, latency, and storage for the model.

3.1.3. Analyzing Outliers

We next examine how outliers in activations and weights affect quantized model performance. Outliers stretch the quantization range, wasting bit precision on rare magnitudes and increasing quantization error for common values. We compare two schemes: (i) standard uniform quantization (no clipping) and (ii) percentile-based clipped quantization.

Activation distributions are collected using forward hooks during inference on a subset of test data:

$$a_\ell = \text{ReLU}(x_\ell), \quad \text{for selected layers } \ell.$$

Table 3. Inference time and model size for post-training quantized VGG11 on CIFAR-100.

Quantization Method	Inference Time (s)	Model Size (MB)
Baseline (FP32)	53.7	492.8
Float16	44.2	246.4
BFloat16	83.9	246.4
Int8	51.0	149.7
Int4	49.7	92.4

To manage memory, a random subset of activation values per layer is sampled. Visualizing these activations reveals the effect of outliers.

The two quantization schemes are defined as:

(i) **Standard Uniform Quantization:** No clipping is applied (standard *INT8* quantization via `torchao`).

(ii) **Clipped Quantization:** To remove outliers, activations and weights are clipped to a fixed percentile (e.g., 99.9%) before quantization:

$$s = \frac{x_{\text{clip}}}{q_{\max}}, \quad x_{\text{clip}} = \text{clip}(x, -P_{99.9}, P_{99.9}),$$

where $P_{99.9}$ is the 99.9^{th} percentile of the tensor magnitude.

Both are implemented for *INT8* weight-only quantization. The resulting models are evaluated for classification accuracy, size, and numerical stability. Finally, we plot histograms of the activation distribution before and after clipping to visualize how clipping removes extreme values in activations and enhances quantization effectiveness.

3.2. Results

After fine-tuning the VGG11 model for 5 epochs, we achieve a test accuracy of 71.01%, which serves as the baseline.

3.2.1. Post-Training Quantization vs. Quantization-Aware Training

As shown in Table 4, accuracy decreases as we go from FP16 to BF16 and further to lower bit widths (INT8 and INT4). Model sizes and inference times are summarized in Table 3.

QAT proved less stable than PTQ after *FP32* fine-tuning, with test accuracy fluctuating between 57–60%. No QAT model surpassed PTQ performance, even after twice as many epochs. Table 5 summarizes the results for this part.

3.2.2. Mixed Precision

As shown in Table 6, mixed-precision quantization improves accuracy with slight computational overhead. The simple scheme slightly increases accuracy and size, while the adaptive variant achieves full baseline accuracy (71.01%) by allocating higher precision to sensitive layers, at the expense of greater inference time and memory.

Table 4. Post-Training Quantization (PTQ) results for VGG11 fine-tuned on CIFAR-100 (Top-1 test accuracy).

Method	Precision / Format	Accuracy (%)
Baseline (FP32)	32-bit float	71.01
4*PTQ	Float16	67.55
	BFloat16	67.45
	Int8	67.20
	Int4	67.07

Table 5. Results for Quantization-Aware Training (QAT) for 10 epochs using pretrained VGG11 (IMAGENET1K_V1).

Method	Precision / Format	Accuracy (%)
Baseline (FP32)	32-bit float	71.01
3*QAT on Pretrained	Float16	59.12
	BFloat16	58.93
	Int8ActivationsInt4Weights	59.43

Table 6. Comparison of mixed-precision quantization schemes on VGG11 (CIFAR-100).

Configuration	Accuracy (%)	Inference Time (s)	Model Size (MB)
Baseline (INT8)	67.20	51.0	149.7
Simple Mixed Precision	67.77	63.7	150.88
Adaptive Mixed Precision	71.01	323.4	248.80

3.2.3. Analyzing Outliers

Percentile-based clipping improves INT8 quantization performance by reducing the influence of extreme activations. As shown in Table 7, applying 99.9th percentile clipping leads to a 3.96% accuracy gain over standard INT8 quantization. Clipping removes rare large activations, improving resolution for common values. Figures 3 and 4 visualize the activation distributions before and after clipping.

Table 7. Effect of percentile-based clipping on INT8 quantization performance.

Method	Accuracy (%)	Improvement (%)
INT8 (standard)	67.20	-
INT8 (with clipping, 99.9th perc.)	71.16	+3.96

3.3. Analysis

3.3.1. PTQ vs QAT

Our results reveal a clear trade-off between the simplicity of *Post-Training Quantization (PTQ)* and the complexity of *Quantization-Aware Training (QAT)*. PTQ, as a straightforward one-shot method, provided a robust and reliable baseline, with only a small and consistent accuracy drop across

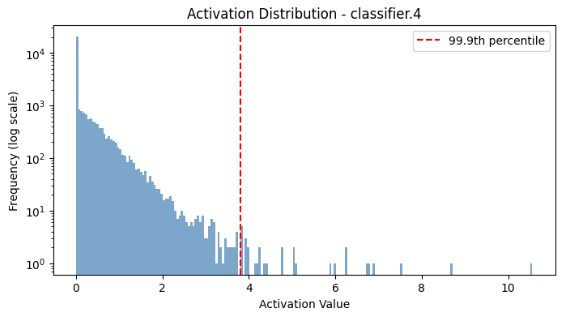

Figure 3. Activation histogram before clipping.

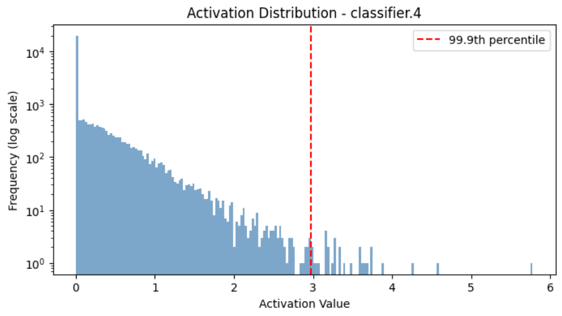

Figure 4. Activation histogram after clipping.

all bit-widths. Starting from the *FP32* baseline accuracy of 71.01%, both 16-bit formats (*FP16* and *BF16*) achieved $\approx$ 67.5%, a drop of about 3.5%. More aggressive quantization let to further drop in accruacy to achieve 67.20% for *INT8* (3.81% drop) and 67.07% for *INT4* (3.94% drop), Table 4. This drop is explained by loss of information when we quantize to lower precisions.

In contrast, QAT, designed to mitigate such losses, failed to recover performance in our setup. Despite training for 10 epochs (twice the baseline fine-tuning duration), QAT proved unstable, with accuracy fluctuating between 57% and 60%. The best-performing QAT model reached only 59.43%. This suggests that fine-tuning in higher precision provides greater training stability. Since QAT models underperformed relative to PTQ even after longer training, it requires more training epochs and compute to reach comparable stability. The worst PTQ result (67.07% for *INT4*) still exceeded the best QAT result (59.43%) by 7.6%. While further experimentation with more complex models and larger datasets could lead to behaviour consistent with theory, our results show PTQ to be more effective and efficient under the tested conditions.

3.3.2. Scaling Law Analysis

We next analyze how model performance scales with reduced bit precision. The scaling behavior of PTQ exhibits diminishing accuracy losses as precision decreases.

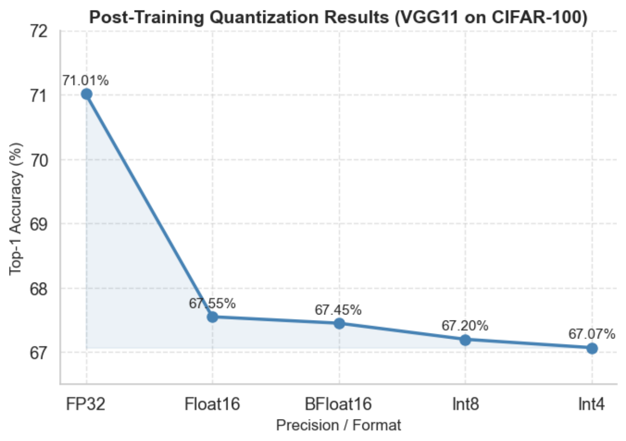

Figure 5. PTQ accuracy versus precision on CIFAR-100 test dataset.

As shown in Figure 5, the PTQ accuracy curve remains nearly flat across lower bit-widths. The accuracy drop from Float16 (67.55%) to Int8 (67.20%) is merely 0.35%, and from Int8 to Int4 (67.07%) only 0.13%. This minimal degradation, despite a large size reduction (from 149.7 MB to 92.4 MB), strongly supports recent scaling law findings. Our results align with the theory, which suggests that 4-bit quantization is nearly optimal in balancing model size and accuracy (Dettmers & Zettlemoyer, 2023). Even in the vision domain, our 4-bit PTQ retained almost all 8-bit performance.

Theoretically, QAT should offer benefits at lower precisions where PTQ degrades. However, our findings contradict this expectation, QAT consistently underperformed at every bit-width, likely due to training setup and instability that masked its potential benefits. Further training and more rigorous experiments are required to prove ineffectiveness of QAT.

3.3.3. Mixed Precision

We now evaluate mixed-precision quantization, which seeks to balance efficiency and accuracy by assigning different precisions to layers based on their sensitivity. The main challenge lies in the exponentially large search space for layer-wise precision assignments (Dong et al., 2019). Our analysis compares three configurations: (i) uniform *INT8* quantization, (ii) a simple mixed-precision scheme preserving the first and last layers at full precision, and (iii) an adaptive mixed-precision scheme using activation variance as a sensitivity measure. The adaptive approach approximates more advanced methods that use second-order metrics, such as the Hessian trace (Dong et al., 2019).

The trade-offs for each configuration are detailed in Table 6. The **uniform INT8** baseline achieved the smallest size (149.7MB) and fastest inference (51.0s) but at reduced accuracy (67.20%). The **simple mixed-precision** variant slightly improved accuracy to 67.77% with minimal latency and size overhead. In contrast, the **adaptive mixed-precision** scheme fully recovered baseline accuracy (71.01%) by selectively assigning *INT8*, *FP16*, and *FP32* formats based on layer variance. However, this came at a significant cost: model size increased to 248.8MB and latency to 323.4s. Despite this, it remained more storage-efficient than the baseline *FP32* model while achieving same accuracy.

The simple mixed precision method assumes that the first and last layers are most critical. This proved to be partially incorrect, as the resulting accuracy gains were marginal. The adaptive variant, by contrast, correctly identified and preserved precision in high-variance layers, explaining its accuracy recovery. While starting and ending layers were kept in high precision in this method too, one intermediate layer was also moved to a medium precision to gain accuracy. However, this improvement did not reflect a true efficiency gain: it effectively reverted a large, sensitive layer to higher precision, negating much of the quantization benefit. The high latency likely arises from frequent type casting between *FP16* and *FP32*. These findings emphasize that quantization performance depends critically on accurate sensitivity estimation rather than random layer selection.

3.3.4. Outlier Impact and Mitigation

Outliers, pose a major challenge for low-bit quantization. They force the quantizer to allocate a wide dynamic range for rare magnitudes, which increases quantization noise for common values and degrades performance. Theory suggests inaccurate activation clipping ranges are a key cause of such degradation in low-bit models (He et al., 2022).

We compare two *INT8* quantization schemes: (i) standard uniform quantization (no clipping) and (ii) percentile-based clipping, truncating tensors to the 99.9^{th} percentile before quantization. The standard *INT8* model achieved 67.20% accuracy, nearly 4% below the full-precision baseline. Applying 99.9^{th} percentile clipping restored accuracy to 71.16%, fully recovering the $+3.96\%$ loss (Table 7). Clipped model overperformed the baseline model which shows how outliers can have extreme effects even a higher precisions. Quantization to *INT8* using clipping proved to be the most effective technique in this experiment keeping the latency and model size of true *INT8* model as mentioned in Table 3 while keeping the highest accuracy.

Figures 3 and 4 visualize this effect. The pre-clipping histogram shows a long-tailed distribution dominated by small values but stretched by outliers. After clipping, these extreme values are removed and the tail of the distribution is more compressed, allowing the quantizer to allocate bits more effectively to the dense region of the distribution. Quantization perturbs intermediate feature distributions, and this mismatch which is worsened by outliers, can reduce model accuracy (He et al., 2022). Our clipping strategy succeeded by minimizing such disturbances at the output of ReLU layers, thus stabilizing subsequent normalization layers. Overall, this experiment shows that outlier-robust quantization methods, even as simple as percentile clipping, can yield substantial accuracy recovery.

3.3.5. Overall Analysis

Overall, our results align with theoretical expectations: quantization reduces model size and can maintain competitive accuracy with proper calibration. However, little to no latency improvements were noticed for most methods because hardware acceleration is typically optimized for *FP32* Therefore, our analysis prioritizes accuracy and model size as the main evaluation metrics, as inference time can vary significantly across hardware platforms e.g. CUDA might be able to reduce latency for FP16 but MPS might lead to same latency even with smaller model.

Quantization is a simple and fast method to reduce model size and in our experiments techniques like clipping even led to an increase in accuracy over the baseline model. This shows that quantization can effectively reduce model size and at times even increase the performance. Quantization, especially with PTQ, is much faster and less complex than knowledge distillation and pruning, which makes it most suitable for quick deployment if compute and training is a major issue.

4. Knowledge Distillation

4.1. Methodology

In this study, we investigate Knowledge Distillation (KD) techniques for transferring knowledge from a larger teacher model to a smaller student model. We examine multiple KD approaches, including Logit Matching (LM), Hint-based Distillation, and Contrastive Representation Distillation (CRD), and compare their effectiveness against a student model trained independently from scratch.

4.1.1. Teacher Models

We employed several teacher models to study different aspects of KD:

Standard Teacher A VGG-16 network pretrained on ImageNet was finetuned on CIFAR-100 for 200 epochs using standard augmentations. Training augmentations included random cropping to 32×32 with padding of 4, random horizontal flipping, and normalization with mean $\mu = [0.5071, 0.4867, 0.4408]$ and standard deviation $\sigma = [0.2675, 0.2565, 0.2761]$. Optimization used SGD with learning rate $\eta = 0.01$, momentum 0.9, weight decay 5×10^{-4}, and a cosine annealing scheduler with $T_{\max} = 200$.

KL Divergence vs. Accuracy in Hint-based KD. Although the Hint-based student achieves the highest validation accuracy, what is unexpected is its KL divergence with the teacher is also the highest. This occurs because Hint-based KD optimizes intermediate feature representations rather than directly matching the teacher's output probabilities. As a result, the student learns high-quality features that improve top-1 classification, but the full probability distribution, particularly the non-target class probabilities, may differ substantially from the teacher. KL divergence is sensitive to these differences in minor probabilities, whereas accuracy only reflects whether the predicted top class is correct. Hence, high accuracy can coexist with a high KL divergence when the student replicates the teacher's decision-making effectively but does not match the full output distribution.

Color Jitter Teacher To investigate invariance to color transformations, we applied color jitter augmentation with brightness, contrast, saturation, and hue perturbations of $(0.4, 0.4, 0.4, 0.1)$ during training. The remaining training protocol mirrored that of the standard teacher.

Larger Teacher A VGG-19 model from chenyaofo/pytorch-cifar-models was used to examine the effect of teacher capacity. It was finetuned on CIFAR-100 under the same protocol as the VGG-16 teacher.

4.1.2. Student Models

All student models were VGG-11 networks trained from scratch for 50 epochs with batch size 128. SGD optimization was used with learning rate $\eta = 0.01$, momentum 0.9, and weight decay 5×10^{-4}. Standard CIFAR-100 augmentations were applied uniformly.

Logit Matching and Variants The student was trained to mimic the softened logits of the teacher with temperature $T = 4.0$ and distillation weight $\alpha_{\text{KD}} = 0.9$. Label Smoothing Regularization (LSR) with $\epsilon = 0.1$ was applied to prevent overconfidence. Decoupled Knowledge Distillation (DKD) separated the learning of target and non-target classes with weights $\alpha = 1.0$ and $\beta = 8.0$, respectively, to optimize knowledge transfer more effectively.

Hint-Based Distillation Hint-based KD leverages intermediate teacher features. The teacher hint layer was `features[16]` of VGG-16 and the student hint layer was `features[8]` of VGG-11. Training occurred in two stages:

1. **Stage 1**: Student features were mapped via a learnable regressor to align with teacher hints. The MSE loss was computed after adaptive pooling to match spatial dimensions.
2. **Stage 2**: Student logits were trained with a combination of cross-entropy and KL-divergence-based KD loss:

$$\mathcal{L} = \alpha_{\text{KD}}\mathcal{L}_{\text{KD}} + (1 - \alpha_{\text{KD}})\mathcal{L}_{\text{CE}}, \quad \alpha_{\text{KD}} = 0.9. \quad (1)$$

Contrastive Representation Distillation (CRD) CRD aligns student and teacher representations via a contrastive loss. Features from the last convolutional layers were projected into a shared embedding space:

$$\text{proj} : \mathbb{R}^{512} \to \mathbb{R}^{256} \to \text{ReLU} \to \mathbb{R}^{128}.$$

CRD used a contrastive temperature $\tau = 0.07$, contrastive weight $\alpha_{\text{CRD}} = 0.8$, KD weight $\alpha_{\text{KD}} = 0.5$, and 4096 negative samples per batch. All students were trained with standard augmentations, except for the CRD-Color variant where the teacher was trained with color jitter while the student received standard augmentations to study ambient color invariance transfer.

4.1.3. Training Protocol

All student models were trained under consistent augmentation policies to ensure fair comparisons. SGD optimization with learning rate $\eta = 0.01$, momentum 0.9, and weight decay 5×10^{-4} was used uniformly. For Hint-based and CRD methods, additional projection or regressor layers were applied to align intermediate or latent representations between teacher and student.

4.2. Results

Teacher Models. The validation accuracies of the teacher models on CIFAR-100 are summarized in Table 8.

Table 8. Validation Accuracy (%) of Teacher Models on CIFAR-100

Teacher Model	Validation Accuracy (%)
VGG-16	72.24
VGG-19	73.87
VGG-16 with Color Jitter	71.77

Student Models. The student models trained under various KD methods, along with the independent student, achieved the validation accuracies reported in Table 9.

Table 9. Validation Accuracy (%) of Student Models on CIFAR-100

Student Model	Validation Accuracy (%)
Independent (SI)	59.19
Logit Matching (LM)	63.35
LM with VGG-19 Teacher	61.24
Label Smoothing Regularization (LSR)	60.63
Decoupled Knowledge Distillation (DKD)	66.11
Hint-based KD	66.99
Contrastive Representation Distillation (CRD)	62.40
CRD with Color-Jittered Teacher (True Val)	63.59
CRD with Color-Jittered Teacher (Color-Jittered Val)	55.44

Probability Distribution Alignment. The KL-divergence between the teacher and student probability distributions is shown in Figure 6.

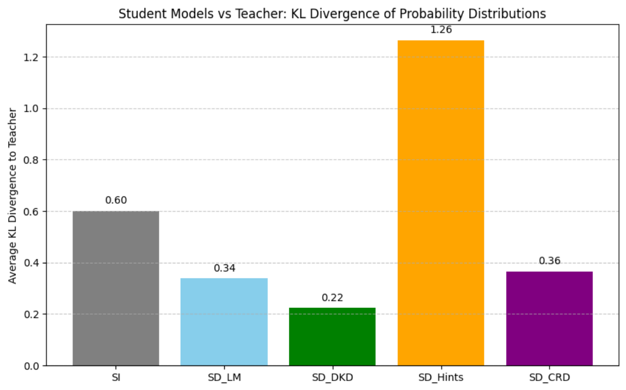

Figure 6. Average KL-divergence between teacher and student probability distributions. Lower values indicate closer alignment with the teacher.

Localization Knowledge Transfer. GradCAM visualizations comparing teacher and student attention are presented in Figure 7. The top row shows teacher visualizations, and the bottom row shows student visualizations. Each column corresponds to a different student model in the order: independent, basic logit matching, LM, DKD, CRD, and Hint-based KD. The figure spans both columns for full-width presentation.

5. Discussion

Our experiments highlight several important observations regarding the efficacy of different knowledge distillation (KD) methods in compressing knowledge from a teacher to a student model. We discuss these results in terms of student accuracy, probability distribution alignment, attention localization, invariance properties, and teacher characteristics.

Performance and Probability Alignment. Across all KD methods, the order of student validation accuracy was: Hint-based KD (66.99%) $>$ Decoupled Knowledge Distillation (DKD, 66.11%) $>$ Logit Matching (LM, 63.35%) $>$ CRD (62.4%) $>$ Label Smoothing Regularization (LSR, 60.63%) $>$ Independent student (SI, 59.19%). DKD achieved the best performance primarily because it separates the target and non-target components of the teacher's probability distribution (Zhao et al., 2022). This separation preserves richer inter-class relationships in the non-target classes, which standard KL-divergence based methods often fail to capture, as the target class probability dominates and minor probabilities are ignored (Hinton et al., 2015). LM followed closely as it directly mimics the teacher's soft labels, which already encode optimized inter-class relationships. LSR performed worst among KD methods because the softening is applied arbitrarily via a fixed formula, rather than reflecting the teacher's learned class relationships (Sun et al., 2021). Nevertheless, all KD-based methods improved the alignment with the teacher over the independent student, as reflected in lower KL divergence values.

Hint-based KD. Interestingly, hint-based KD exhibited the worst GradCAM alignment with the teacher, even worse than the independent student (Romero et al., 2014). While the method is stable to train due to its staged feature-matching approach, it can be computationally intensive and time-consuming. The poorer attention alignment may result from the intermediate feature regression focusing on Euclidean feature distances, which does not directly enforce attention similarity at the final decision layer. This highlights a trade-off between training stability and effective localization knowledge transfer.

Contrastive Representation Distillation (CRD). CRD demonstrated noticeable improvement in student performance when the teacher was trained with color jitter augmentations (Tian et al., 2020). Training the student on the normal dataset while distilling features from a color-augmented teacher likely introduced some color invariance and reduced color bias. This suggests that CRD may be particularly beneficial in scenarios where invariance to input transformations (e.g., color, lighting) is desirable, or when students need to generalize beyond the exact distribution seen during training.

Teacher Size and Strictness. Using a larger teacher (VGG-19) for LM distillation resulted in slightly worse student accuracy compared to VGG-16. This is likely because VGG-19 produces more confident predictions, compressing non-target class probabilities towards zero, thereby limiting the inter-class information available for the student to learn. This demonstrates that a ”lenient” teacher that provides richer probability distributions can sometimes be more effective than a strictly confident teacher in KD settings.

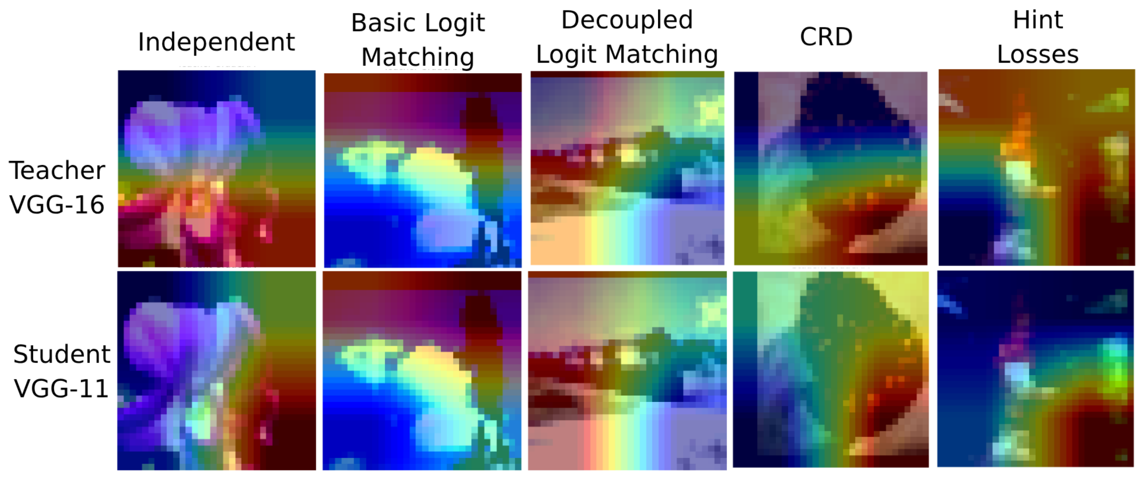

Figure 7. GradCAM visualizations comparing teacher and student attention. Top row: teacher; bottom row: students trained under different KD methods.

General Observations. Overall, the results were mostly in line with expectations from the literature (Hinton et al., 2015; Zhao et al., 2022; Tian et al., 2020). CRD underperformed slightly, potentially due to limited training epochs and fixed hyperparameters kept consistent for fairness. Unlike simply reducing the size of the teacher network, which can significantly degrade its predictive performance, Knowledge Distillation (KD) allows a smaller student to leverage the full representational power of a large, well-trained teacher. Compared to pruning or quantization, KD offers the advantage of improving student generalization and potentially imparting invariance properties or richer inter-class relationships from the teacher without modifying the architecture or reducing representational capacity. While pruning and quantization reduce model size and computation, they do not inherently transfer knowledge from a larger model. KD thus provides complementary benefits, particularly in scenarios where model interpretability, robustness, and alignment with teacher behavior are critical.

6. Conclusion

Our study highlights the strengths and limitations of *quantization*, *pruning*, and *knowledge distillation* as model compression techniques. *Quantization* provides the most hardware-friendly and deployment-ready optimization by reducing numerical precision, leading to significant reductions in memory and latency with minimal compute. It is particularly effective in real-time or edge scenarios where inference efficiency and low energy consumption are critical. However, extremely low-bit quantization may lead to representational loss, especially in sensitive layers. *Pruning*, on the other hand, exploits the overparameterization of deep networks by removing redundant weights or channels. While it can deliver substantial sparsity and speedups when supported by sparse computation kernels, identifying an optimal pruning mask, the winning lottery ticket, remains a complex and often compute intensive process. Finally, *knowledge distillation* compresses models by transferring knowledge from a high-capacity teacher to a lightweight student, preserving accuracy better than other methods, but it has an additional computational overhead. Therefore, to reach the same accuracy as other methods, KD requires more training. KD provides more control over compression process where most important things can be learnt from the teacher, leading to better generalization.

In practice, quantization is best suited for deployment on resource-constrained or specialized hardware; pruning is advantageous when fine-grained control over sparsity or interpretability is desired; and knowledge distillation excels in scenarios prioritizing accuracy preservation under tight model size constraints. Collectively, these strategies, from low-level numerical compression to high-level knowledge transfer, can be combined for scalable model compression and deployment.

References

Dettmers, T. and Zettlemoyer, L. The case for 4-bit precision: k-bit inference scaling laws. *arXiv preprint arXiv:2212.09720v2 [cs.LG]*, 2023.

Dong, Z., Yao, Z., Cai, Y., Arfeen, D., Gholami, A., Mahoney, M. W., and Keutzer, K. Hawq-v2: Hessian aware trace-weighted quantization of neural networks. *arXiv preprint arXiv:1911.03852v1 [cs.CV]*, 2019.

Han, S., Pool, J., Tran, J., and Dally, W. J. Learning both weights and connections for efficient neural networks. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

He, Y., Zhang, L., Wu, W., and Zhou, H. Data-free quantization with accurate activation clipping and adaptive batch normalization. *arXiv preprint arXiv:2204.04215v2 [cs.LG]*, 2022.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. Technical Report.

LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In *Advances in Neural Information Processing Systems*, volume 2, pp. 598–605. Morgan Kaufmann, 1989.

Romero, A., Ballas, N., Ebrahimi Kahou, S., Chassang, A., Gatta, C., and Bengio, Y. Fitnets: Hints for thin deep nets. In *arXiv preprint arXiv:1412.6550*, 2014.

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 618–626, 2017.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Sun, S., Ren, W., Li, J., Wang, R., and Cao, X. Logit standardization in knowledge distillation. In *arXiv preprint arXiv:2106.02240*, 2021.

Tian, Y., Krishnan, D., and Isola, P. Contrastive representation distillation. In *arXiv preprint arXiv:2010.04187*, 2020.

Zhao, B., Cui, Q., Song, R., Qiu, Y., and Liang, J. Decoupled knowledge distillation. In *arXiv preprint arXiv:2106.08478*, 2022.

Appendix

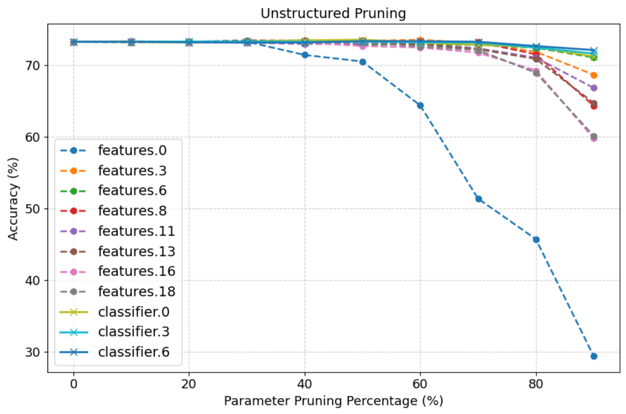

Figure 8. Sensitivity Analysis for Unstructured Pruning.

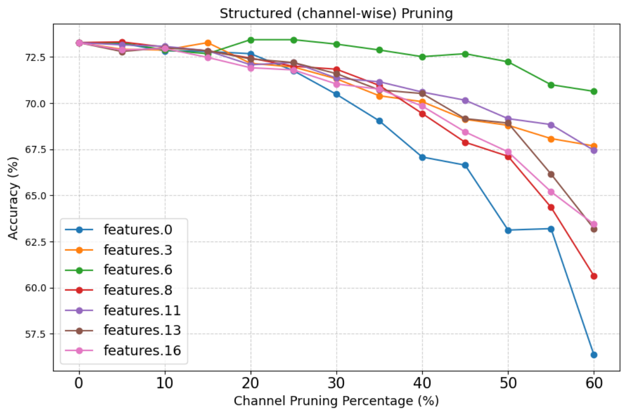

Figure 9. Sensitivity Analysis for Structured Pruning.