Advanced Topics in Machine Learning - PA3

Ahmad Ashraf [1] Sarim Malik [2]

Abstract

This paper examines three major neural network compression methods: pruning, quantization, and knowledge distillation; on VGG-11 models trained on CIFAR-10 and CIFAR-100. Both structured and unstructured pruning significantly reduced model parameters with minimal accuracy loss, revealing the higher sensitivity of early convolutional layers. Quantization-Aware Training consistently outperformed Post-Training Quantization, especially at lower precisions, while mixed-precision approaches achieved strong compression-accuracy trade-offs. Supplementary analysis and visualizations confirmed that compressed models preserved key feature representations. Overall, this study highlights effective strategies for building efficient and deployable neural networks across diverse hardware settings.

Our codebase with complete experiments can be found here: https://github.com/ATML-AshrafxSarim/PA3 and backup: https://github.com/s-malix21/atml-fall2025.

1. Introduction

The rapid growth of neural networks in both complexity and depth has enabled them to achieve remarkable success across a wide range of tasks. However, their deployment on resource-constrained platforms such as edge devices, smartphones, and embedded systems remains a major challenge. Modern networks often contain millions of parameters and require significant storage and computational overhead, making them unsuitable for real-time or low-power applications. For instance, the VGG-11 network contains approximately 130 million parameters, requiring nearly 500 megabytes of storage and a substantial number of floating-point operations for a single forward pass. Over recent years, substantial research has focused on optimizing neural networks to deliver comparable performance while reducing storage demands and improving inference speed.

In this study, we focus on three major families of model compression techniques: Pruning, Quantization, and Knowledge Distillation, where each method offers distinct mechanisms for efficiency and present unique trade-offs between model size, accuracy, and computational cost.

Our experiments aim to address the following key questions:

- How convenient and feasible are these methods to implement, and under what circumstances does one compression approach outperform the others?
- What are the advantages and drawbacks of each method in practical settings?
- How effective is each approach under specific hardware and precision limitations?

Through this paper, we compare the effectiveness of state-of-the-art compression strategies, examine how they achieve efficiency, and explain the underlying mechanisms that allow models to retain performance while operating within constrained environments.

2. Methodology

2.1. Pruning

2.1.1. Model Architecture, Setup, and Datasets

The experiments used the standard VGG-11 architecture from the Han Lab 6.5940 specification, consisting of eight convolutional layers with Batch Normalization and ReLU activations. The channel configuration followed $[64, 128, \mathtt{M}, 256, 256, \mathtt{M}, 512, 512, \mathtt{M}, 512, 512, \mathtt{M}]$, where M indicates max pooling. The classifier employed global average pooling followed by three fully connected layers $(512 \to 512 \to 10)$ with dropout $(p = 0.5)$ and ReLU activations.

CIFAR-10 was used for all experiments (as per the Han lab config), comprising 50k training and 10k test images across ten classes. Data augmentation included random cropping (padding = 4) and horizontal flipping, with per-channel normalization, with the respective μ and σ computed over the training set.

Training used SGD with learning rate 0.01, momentum 0.9, and weight decay 5×10^{-4}, optimized for 20 epochs with batch size 128. The objective minimized the cross-entropy loss:

$$\mathcal{L}_{CE} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \log(\hat{y}_{i,c})$$

where $y_{i,c}$ and $\hat{y}_{i,c}$ denote the true and predicted probabilities for class c.

All experiments ran on a CUDA-enabled GPU with automatic device detection and CPU fallback, ensuring reproducible and efficient training across pruning setups.

2.1.2. Unstructured Pruning Methodology

Unstructured pruning followed a magnitude-based approach, removing individual weights with low L2-norm values. For each layer, weights below the pruning threshold

$$\tau = \text{quantile}(|W|, s)$$

were set to zero, where $s \in [0, 1]$ denotes the target sparsity fraction. This procedure was applied independently per layer to achieve the desired sparsity profile.

To assess pruning sensitivity, sparsity ratios $s \in \{0.0, 0.2, 0.4, 0.6, 0.8\}$ were tested across layers, allowing visualization of accuracy degradation and weight distribution changes. Weight histograms before and after pruning confirmed the expected spike at zero, reflecting removed connections while retaining the original distribution shape for remaining weights.

Based on the sensitivity findings, differentiated sparsity levels were assigned across layers—lower sparsity for early convolutional layers (0.1–0.3) and higher for later and fully connected layers (up to 0.8)—targeting roughly 70% overall network sparsity. After pruning, models were fine-tuned for 5 epochs using a reduced learning rate of 0.001 to recover accuracy.

2.1.3. Structured Pruning Methodology

Structured pruning operated at the channel level, removing entire convolutional filters based on their L2-norm importance scores, defined as

$$I_c = \|W_c\|_2,$$

where W_c denotes the weights associated with channel c. Channels with the lowest importance were pruned to achieve target sparsity while maintaining functional integrity.

To preserve consistency across layers, the pruning of output channels in layer l was coupled with the removal of corresponding input channels in layer $l + 1$. Batch Normalization parameters, including weights, biases, and running statistics, were updated accordingly to match the reduced dimensionality.

Layer-wise sensitivity was evaluated using the same sparsity levels as in the unstructured setup, but applied to channel removal rather than individual weights. Layer-specific pruning ratios were then configured to achieve roughly 70% overall parameter reduction, with conservative pruning applied to early layers and higher pruning in deeper ones. Structural constraints were enforced to prevent collapse, ensuring that the final convolutional layer and classifier input dimensions remained intact.

2.1.4. Comparative Analysis and Grad-CAM Evaluation

Comparative evaluation was conducted across the baseline, unstructured-pruned, and structured-pruned models using several quantitative and qualitative metrics. Performance was measured in terms of classification accuracy, model size (MB), parameter count, inference time, FLOPs, and sparsity. For providing a direct measure of model compression efficiency, actual sparsity was computed as

$$\text{sparsity} = 1 - \frac{\text{non-zero parameters}}{\text{total parameters}},$$

Grad-CAM visualizations were generated using the final convolutional layer to assess how pruning affected spatial feature localization. These class activation maps enabled comparison of attention consistency across model variants, revealing whether pruned networks preserved the discriminative focus of the baseline. Inference time was averaged over 50 synchronized GPU batches, excluding warm-up iterations to ensure stable measurement. Finally, hardware efficiency was analyzed through comparisons of memory footprint, computational cost, and observed speedups, highlighting the differing deployment advantages of each pruning method.

2.1.5. Robustness and Sensitivity Characterization

Robustness evaluation examined how each pruning method responded to increasing compression levels. Validation accuracy was tracked against target sparsity levels $\{0.0, 0.2, 0.4, 0.6, 0.8\}$ to form accuracy–sparsity curves for both structured and unstructured pruning. Extreme sparsity experiments (85%–97%) were conducted to identify collapse points where network performance degraded sharply.

Layer-wise sensitivity was further analyzed by measuring the accuracy drop associated with pruning each layer independently, distinguishing between critical and redundant layers. Fine-tuning recovery potential was assessed by retraining pruned models under fixed hyperparameters to evaluate post-pruning convergence stability. A cross-method comparison at matched compression ratios provided insight into trade-offs between accuracy retention, robustness, and hardware efficiency.

2.2. Quantization

2.2.1. Dataset and Model Selection

For all quantization experiments, we use a pretrained VGG-11 CNN initialized with ImageNet weights. The model is then usually fine-tuned on the full CIFAR-100 dataset using standard data augmentation, which includes only random horizontal flipping and resizing images to 224×224. The final linear layer of the model is replaced with a new classifier suitable for CIFAR-100.

2.2.2. Post Training Quantization vs. Quantization Aware Training

We perform weight-only quantization for both Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT) at four different precision levels: FP16, BF16, INT8, and INT4. We use the FP32 model as the baseline. The models are evaluated based on their top-1 classification accuracy and compressed model size to measure the trade-off between compression and performance.

Post Training Quantization (PTQ). We fine-tune the VGG-11 model on the CIFAR-100 dataset for five epochs using the Adam optimizer with default hyperparameters and a learning rate of 1e-4. After fine-tuning, we perform Post-Training Quantization (PTQ) using custom quantizers and dequantizers that replace the model's layers with quantized counterparts at the desired bitwidths. During inference, these quantized layers are dequantized back to float32 to ensure compatibility and correct model execution. This custom implementation is necessary due to compatibility issues between PyTorch's built-in quantization libraries and the hardware and software environment provided by Google Colab.

Quantization Aware Training (QAT). We initialize the VGG-11 model with pretrained ImageNet weights and prepare it for quantization-aware training by replacing its standard convolutional and linear layers with custom QAT-compatible layers that incorporate fake quantization modules. These modules simulate the numerical effects of quantization and dequantization during the forward pass, enabling the model to learn within the quantization space dynamically throughout training. The learning rate and number of epochs are adjusted based on the quantization bitwidth, as summarized in Table 1. After training, the model is converted into its quantized form to achieve compression. It should be noted that during inference, the model dequantizes back to float32 to maintain compatibility.

Table 1. QAT Hyperparameter Settings by Bitwidth

Bitwidth	**Learning Rate**	**Maximum Epochs**
FP16 / BF16	5e-5	5
INT8	5e-5	8
INT4	3e-5	10

2.2.3. Scaling Laws

We compute the relationship between model accuracy and the total number of model bits to evaluate how performance scales with decreasing numerical precision. This analysis follows the principle of scaling laws, which describe how model performance tends to follow a power-law or logarithmic decay as model capacity or precision is reduced. We plot this for both PTQ and QAT to assess how our models adhere to this scaling behavior and perform further analysis.

2.2.4. Mixed Precision Quantization Analysis

We explore the concept of mixed-precision quantization, in which layers of critical importance are maintained at higher precision (e.g., FP16), while less sensitive layers are quantized to lower bitwidths such as INT8 or INT4. The finetuned vanilla VGG-11 model serves as a common baseline across all configurations. Layer precision assignments can be determined either arbitrarily or through informed selection criteria, such as activation variance or Hessian trace analysis.

We evaluate three configurations to compare their performance in terms of accuracy, model compression, and inference efficiency. The first configuration serves as the baseline, where all layers are quantized to INT4. The second configuration follows an arbitrary mixed-precision scheme, retaining higher precision for the first and last layers while quantizing all others to lower precision. The third configuration employs an adaptive selection strategy based on Hessian traces and activation variances, where the top 30% of the most sensitive layers are preserved in high precision, and the remaining layers are quantized. For this experiment, high-precision layers correspond to FP16, while low-precision layers correspond to INT4 quantization.

We further evaluate the impact of mixed-precision quantization by analyzing the quantization error using Mean Square Error (MSE) and Signal-to-Quantization-Noise Ratio (SQNR) metrics. These metrics provide quantitative insights into how closely the quantized model's activations and weights approximate their full-precision counterparts. Experiments are conducted for lower-precision settings of INT8 and INT4, while the high-precision layers are retained at FP16. The results highlight how the degree of precision reduction influences numerical fidelity, helping to identify an optimal trade-off between compression efficiency and representational accuracy.

2.2.5. Effect of Outliers on Quantization Performance

For the purpose of this experiment, we perform quantization on both activations and weights. We assess the effect of outliers, particularly in activations that produce extremely large values and force quantizers to allocate a wider dynamic range, which can introduce additional noise. We evaluate the efficacy of two outlier-handling strategies: the first is a baseline using standard uniform quantization (with no adjustment for outliers), and the second applies clipping to various percentiles (99% and 99.5%) to limit the influence of extreme values. We begin by fine-tuning a fresh VGG-11 model on CIFAR-100 for five epochs and then apply these strategies at different bitwidths, namely INT8 and INT4, to examine how clipping affects quantization performance across precision levels.

Furthermore, we collect activation statistics from each layer to perform a detailed outlier analysis and identify which layers are most susceptible to extreme activation values.

2.3. Task 3; TBD

3. Results

3.1. Pruning

3.1.1. Unstructured Pruning Results

The baseline VGG-11 model achieved 85.1% accuracy on CIFAR-10 with 9.76M parameters and a model size of 37.2 MB. The baseline weight histogram followed the expected bell-curve pattern, with most weights centered near zero. After unstructured pruning, a sharp spike appeared at zero, representing pruned connections, while the remaining weights retained a similar distribution.

Layer-wise sensitivity analysis across sparsity levels (0.0 - 0.8) showed clear differences between early and deeper layers. The first convolutional layer (`features.0`) was highly sensitive, with accuracy dropping from 85.1% to 36.5% at 80% sparsity. In contrast, later convolutional layers (like `features.22` eg) remained robust, sustaining above 84% accuracy even at high sparsity (Figure 7). FC layers showed comparable stability with minimal degradation as shown in Table 9.

Guided by this analysis, custom sparsity ratios were assigned, increasing gradually from early to deeper layers: convolutional layers $[0.1, 0.3, 0.5, 0.6, 0.65, 0.7, 0.75, 0.8]$ and fully connected layers $[0.8, 0.8, 0.5]$. This configuration targeted roughly 70% overall sparsity while preserving essential early-layer representations.

The final pruning achieved 72% actual sparsity. Accuracy before fine-tuning was 83.8%, a which was a really small 1.3% drop from baseline. After 5 epochs of fine-tuning, accuracy recovered to 89.3%. This improvement suggests that pruning acted as a regularizer, reducing redundancy and improving generalization. The layer-wise sparsity results confirmed that early layers were largely preserved, while later layers were pruned more aggressively (Table 9).

3.1.2. Structured Pruning Results

The structured pruning approach validated the L2-norm channel importance method, effectively removing less critical channels while preserving key feature extraction. Coordinating pruning across layers required careful updating of dependent input channels and associated BatchNorm parameters.

Layer-wise sensitivity analysis revealed sharp contrasts across depths (as shown in Table 10). As shown in Figure 5, early layers were highly sensitive: the first convolutional layer (features.0) dropped to 59.2% accuracy at 60% channel removal, while the second (features.4) collapsed to 23.8%. Table 10 shows that deeper layers showed better tolerance; features.15 maintained 78.0% accuracy under similar pruning.

Based on these results, conservative ratios were used for early layers and progressively larger ones for deeper layers: [0.1, 0.25, 0.4, 0.45, 0.5, 0.55, 0.7], leaving fully connected layers intact. This configuration achieved a 71.4% parameter reduction, compressing the model from 9.76M to 2.79M parameters and from 37.2 MB to 10.6 MB, while retaining structural consistency.

Post-pruning weight distributions indicated the retained channels preserved their original variance and activation ranges, confirming that pruning removed redundancy rather than distorting learned representations. Initial accuracy dropped to 22.8% before fine-tuning, reflecting the stronger coupling between channels in structured pruning. After fine-tuning, the pruned model reached 87.6% accuracy, a 2.5% drop from baseline but higher recovery relative to unstructured pruning given the stronger compression.

3.1.3. Comparative Analysis and Robustness Evaluation

As shown in Figure 1, Grad-CAM visualizations showed strong consistency across all models, each correctly identifying key features in test images. For cat samples, activations centered on the head region, while ship samples consistently focused on the upper structure. This indicates that both pruning methods preserved critical feature representations and maintained robust attention despite compression.

Performance metrics in Table 2 reveal clear trade-offs between pruning types. The unstructured model kept the same parameter count (9.76M) but reached 72% sparsity and improved accuracy (89.3% vs. 85.1%). This gain stems from

Table 2. Primary Comparative Study between Baseline, Unstructured, and Structured Pruned Models

Model	Accuracy (%)	Params	Sparsity (%)	Size (MB)	Inf. Time (ms)	Std (ms)	FLOPs (M)	Acc. Drop (%)
Baseline	85.09	9,756,426	0.00	37.24	10.79	0.81	153.29	–
Unstructured (71.96% sparsity)	89.25	9,756,426	71.96	37.24	12.19	1.61	153.29	-4.16
Structured (71.45% param reduction)	87.60	2,785,206	0.00	10.64	7.64	0.48	53.24	-2.51

Table 3. Robustness Analysis at Extreme Sparsity Levels

Method	Target (%)	Actual / Reduction (%)	Acc. (%)	Collapse Point
Unstructured	85	84.9	11.25	∼85% sparsity
Structured	70	81.0	10.00	∼70% pruning

pruning's regularization effect, which removes redundant connections and improves generalization (Kuzmin et al., 2024). Fine-tuning further helped the model reach a better minimum.

Structured pruning, by removing entire channels, reduced parameters to 2.79M and achieved 87.6% accuracy. While slightly lower than the unstructured variant, it offered much higher efficiency: inference time dropped to 7.6 ms from 10.8 ms in the baseline, whereas unstructured pruning rose to 12.2 ms due to dense zero storage. Structured pruning thus delivered real hardware gains, while unstructured pruning offered logical sparsity only.

Hardware metrics confirmed structured pruning's practical benefits, showing a 65% FLOPs reduction (53.2M vs. 153.3M) and 71% smaller memory footprint, leading to genuine speedups on standard hardware. Unstructured pruning, in contrast, maintained the same FLOPs and memory usage since zeros remain stored densely.

Accuracy–sparsity trends showed that unstructured pruning held above 80% accuracy up to 80% sparsity, while structured pruning dropped more sharply beyond 60% channel removal. Excessive compression led to collapse near 95–97% sparsity for unstructured pruning and 75–80% for structured pruning, showing the structural limits of each approach (Table 3).

Across both methods, early layers were most sensitive to pruning. Fine-tuning proved highly effective: structured accuracy recovered from 22.8% to 87.6%, and unstructured from 83.8% to 89.3%, confirming that pruned ntworks can retrain to match or exceed baseline performance.

3.2. Quantization

3.2.1. PTQ vs. QAT

Table 4 shows that QAT consistently outperforms PTQ across all bitwidths. This improvement is expected since QAT learns quantization parameters during training, reducing quantization error. The gains are modest for FP16, BF16, and INT8 but significant for INT4, where QAT recovers most of the lost accuracy.

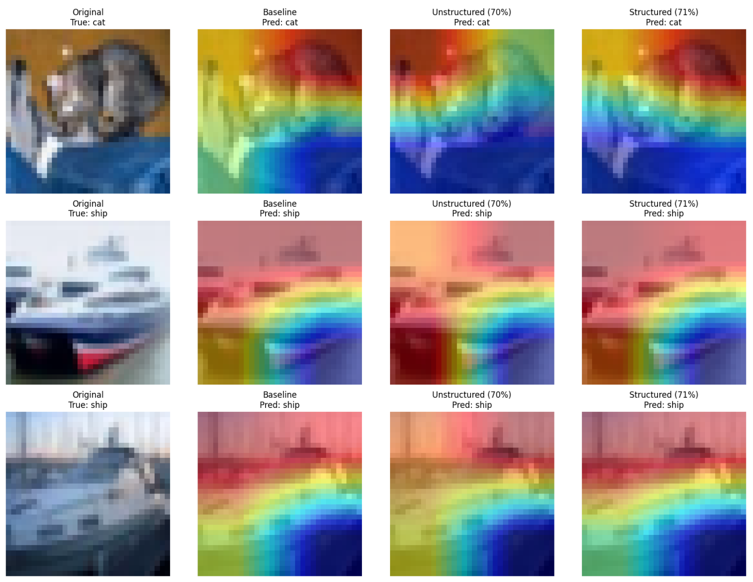

Figure 1. Grad-CAM visualizations comparing baseline, unstructured pruned, and structured pruned VGG-11 models, illustrating similar attention regions despite high sparsity.

However, QAT requires more training to achieve these results. For instance, FP16 and BF16 reach their best accuracy after 5 epochs, INT8 after 8 epochs, and INT4 after 10 epochs. Despite this extra effort, QAT not only recovers lost accuracy but sometimes even surpasses the FP32 baseline, likely due to the regularization effect of quantization noise. What is also interesting is that QAT consistently outperforms the baseline accuracy of (72.93%) suggesting that perturbations induced by QAT serve as a regularizer, aiding the model in its training.

Both methods still achieve the expected model compression (2× for FP16/BF16, 4× for INT8, and 8× for INT4). While the performance remains consistent on all bidwidths, there is a sharp decrease when performing PTQ on INT4, suggesting that INT8 might strike as the sweet spot in terms of balance between model size and performance.

3.2.2. Scaling Laws

Figure 2 illustrates the relationship between model accuracy and total number of model bits for both PTQ and QAT. The PTQ curve demonstrates a clear scaling law behavior, following a logarithmic decay consistent with a power-law

Table 4. Comparison of PTQ and QAT across different bitwidths. QAT achieves higher accuracy at the cost of additional training effort.

Bitwidth	PTQ Acc (%)	QAT Acc (%)	QAT Gain (%)	PTQ Size (MB)	QAT Size (MB)
FP32 (Baseline)	72.94	72.94	+0.00	492.77	492.77
FP16	72.93	77.03	+4.10	246.38	246.38
BF16	72.94	77.09	+4.15	246.38	246.38
INT8	72.69	77.76	+5.07	123.27	123.27
INT4	62.53	76.94	+14.41	61.67	61.67

trend. As the bitwidth decreases, accuracy declines gradually at first but drops sharply at extremely low precisions. This indicates that the quantization error grows nonlinearly as model capacity is reduced, which aligns with the theoretical scaling law observed in studies.

In contrast, QAT consistently outperforms PTQ across all bitwidths, shifting the entire scaling curve upward. The improvements are modest at higher precisions (e.g., FP16 and BF16) but become substantially more pronounced at lower bitwidths. Specifically, at INT4 precision, QAT achieves an accuracy increase of approximately 14.4%, effectively mitigating the degradation seen in PTQ and demonstrating that training-aware adaptation enables the model to maintain high performance even under extreme quantization.

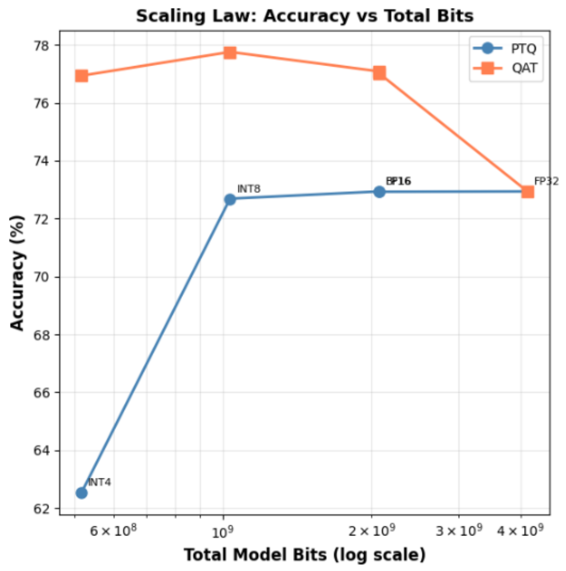

Figure 2. Scaling law analysis of accuracy vs. model size in bits for both PTQ and QAT. PTQ clearly follows a logarithmic decay trend, consistent with the power-law relationship between quantization precision and performance degradation. In contrast, QAT deviates from this trend due to its ability to adaptively learn quantization parameters during training, effectively mitigating the sharp loss in accuracy at lower bitwidths.

3.2.3. Mixed Precision Quantization Analysis

Table 5. Mixed Precision Quantization Results for VGG-11 on CIFAR-100

Configuration	Accuracy (%)	Acc Drop (%)	Size (MB)
FP32 Baseline	70.14	+0.00	492.77
Uniform INT4	59.32	-10.82	61.67
Simple Mixed (First+Last FP16)	68.32	-1.82	62.26
Adaptive Mixed (Top30% FP16)	59.44	-10.70	212.61

The results in Table 5 demonstrate that simple mixed-precision schemes, where only the first and last layers are preserved in higher precision (FP16), achieve a favorable balance between compression and accuracy. These layers typically handle either low-level feature extraction or high-level classification representations, both of which are more sensitive to quantization noise. Preserving their precision allows the network to maintain representational fidelity while aggressively compressing the intermediate layers. As a result, the simple mixed configuration yields minimal accuracy degradation compared to the FP32 baseline while still achieving nearly 8x compression. It must be noted that unfortunately, the variation in inference speeds could not be measured due to the customized implementations of quantization schemes.

In contrast, adaptive mixed-precision schemes require careful tuning of sensitivity thresholds, such as selecting the correct proportion of layers or variance-based criteria for precision assignment. While conceptually adaptive, such methods may not always yield consistent improvements, especially when the selection criteria fail to capture true layer-wise sensitivity. Therefore, for practical deployments, simpler schemes that selectively protect critical layers often provide more predictable and efficient trade-offs between accuracy, compression, and inference speed.

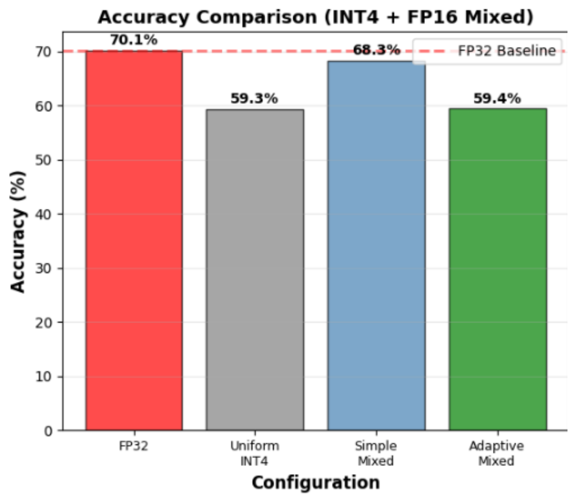

Figure 3. Accuracy comparison of various mixed precision quantization schemes.

3.2.4. Outlier Handling Strategies

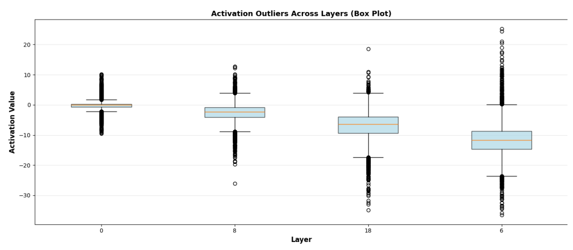

Figure 4. Visualization of activation outliers through boxplots across different layers of VGG-11.

We analyze the presence of activation and weight outliers across different layers to understand their effect on quantization quality. The results in Table 6 show that earlier convolutional layers, such as `features.0`, contain a higher proportion of outliers compared to deeper layers and the classifier. These outliers lead to large activation ranges, increasing quantization noise, especially at lower bitwidths such as INT4. Visualizations of activation value distributions are provided in the Appendix.

Table 6. Activation Outlier Statistics across major layers

Layer	Mean	Std	Abs Max	Outliers
features.0	-0.19	0.84	11.24	6.27%
features.8	-2.38	2.69	65.02	1.56%
features.18	-6.87	4.39	85.40	1.05%
classifier.6	-11.65	4.94	38.61	1.26%

Table 7. Effect of Clipping on Quantization Error and SQNR

Quantization	Clipping	Avg. MSE	Avg. SQNR (dB)
INT8	None	0.00433	38.98
INT8	99.9%	0.01088	35.09
INT4	None	1.41974	14.05
INT4	99.9%	0.90111	16.09

Table 7 compares standard uniform quantization (no clipping) with clipped quantization at the 99.9% percentile. For INT8 quantization, clipping slightly reduces SQNR and increases MSE, implying that higher-precision layers already preserve sufficient dynamic range. However, for INT4 quantization, clipping significantly reduces the average MSE and improves SQNR, showing that clipping effectively limits the influence of extreme outliers and improves representational fidelity. In summary, clipping provides clear benefits for very low bitwidth quantization (INT4), but is less useful for higher bitwidths such as INT8.

This is validated by the results of accuracy over our configurations in both INT8 and INT4, as shown in Table 8. Clipping proves highly effective for INT4 quantization, where activation and weight outliers otherwise cause severe distortion due to the limited representational range. By constraining the dynamic range (e.g., at the 99.9% percentile), the quantizer focuses on the dense distribution of values, leading to a substantial recovery in accuracy. In contrast, INT8 quantization already retains sufficient precision to represent outliers without major degradation, and excessive clipping can remove ”extremes” that might contain important information, slightly reducing performance.

Table 8. Effect of Clipping on Quantization Accuracy for INT4 and INT8

Configuration	Accuracy (%)
INT4 No Clipping	18.55
INT4 99.9% Clipping	51.05
INT4 99.5% Clipping	43.80
INT8 No Clipping	72.95
INT8 99.9% Clipping	71.23

This analysis allows us to observe how certain channels are more prone to producing large outliers, particularly due to activation functions like ReLU, which generate sparse activations with occasional large spikes. Such uneven scaling makes uniform quantization inefficient, as a few extreme values can dominate the dynamic range and increase quantization error. Therefore, effective mitigation techniques such as percentile clipping or adaptive range scaling are essential to improve quantization accuracy. In real-world scenarios, like in large models like Vision Transformers, which exhibit high activation variance across layers, such outlier-robust quantizers are crucial to ensure stable performance and reliable deployment under low-bitwidth constraints.

4. Discussion

5. Conclusion

References

Kuzmin, A., Nagel, M., van Baalen, M., Behboodi, A., and Blankevoort, T. Pruning vs quantization: Which is better? *arXiv preprint arXiv:2401.12345*, 2024. Available at https://arxiv.org/abs/2401.12345.

A. Appendix

A.1. Supplementary Material - Network Pruning

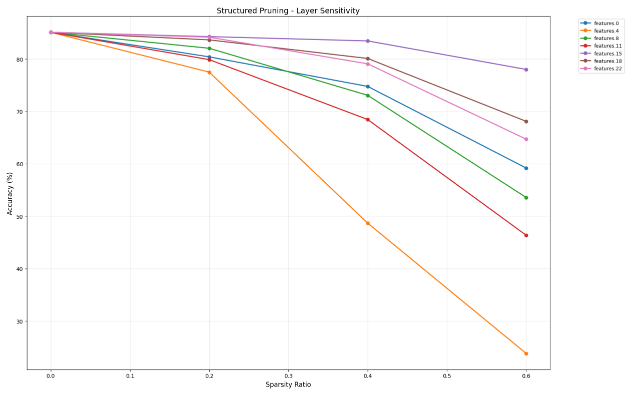

Figure 5. Layer-wise sensitivity of VGG-11 to structured pruning, showing that early convolutional layers suffer the greatest accuracy drop as sparsity increases.

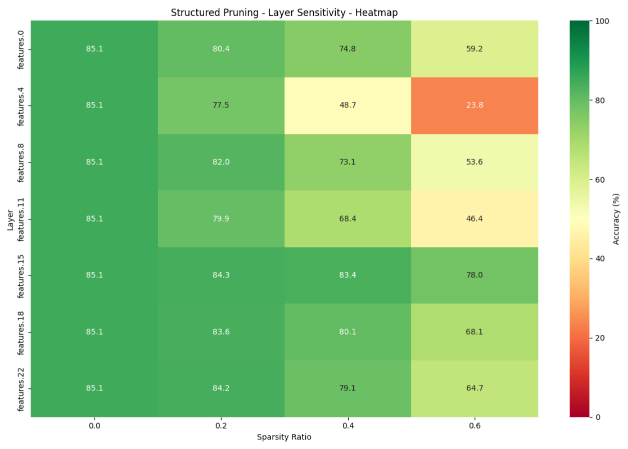

Figure 6. Heatmap of VGG-11 layer sensitivity to structured pruning, highlighting that early layers experience the most severe accuracy degradation at high sparsity ratios.

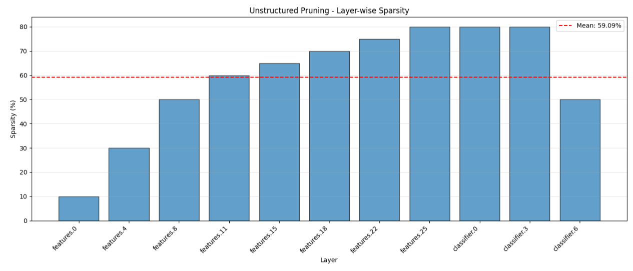

Figure 7. Layer-wise sparsity after unstructured pruning of VGG-11, showing higher sparsity in deeper layers and fully connected layers compared to early convolutional layers.

Table 9. Unstructured pruning sensitivity analysis across selected layers. Accuracy (%) is reported at increasing sparsity levels.

Layer	**0.0**	**0.2**	**0.4**	**0.6**	**0.8**
features.0	85.09	84.98	82.93	71.70	36.45
features.4	85.09	85.11	84.69	85.22	81.21
features.8	85.09	85.05	84.69	83.97	75.45
features.11	85.09	84.97	84.67	83.63	77.67
features.15	85.09	85.06	85.01	85.00	82.35
features.18	85.09	85.15	85.05	85.21	85.46
features.22	85.09	85.04	84.94	85.03	84.96
features.25	85.09	85.08	85.08	85.12	84.89
classifier.0	85.09	85.09	85.06	85.09	85.18
classifier.3	85.09	85.08	85.05	85.07	84.99
classifier.6	85.09	85.03	84.97	85.16	84.97

Table 10. Structured pruning sensitivity analysis across selected layers. Accuracy (%) is reported at increasing channel removal ratios.

Layer	**0.0**	**0.2**	**0.4**	**0.6**
features.0	85.09	80.40	74.76	59.18
features.4	85.09	77.49	48.67	23.80
features.8	85.09	82.04	73.06	53.56
features.11	85.09	79.87	68.44	46.35
features.15	85.09	84.26	83.44	78.00
features.18	85.09	83.64	80.09	68.09
features.22	85.09	84.15	79.07	64.69

A.2. Supplementary Material - Quantization

A.2.1. ACTIVATION VISUALIZATIONS

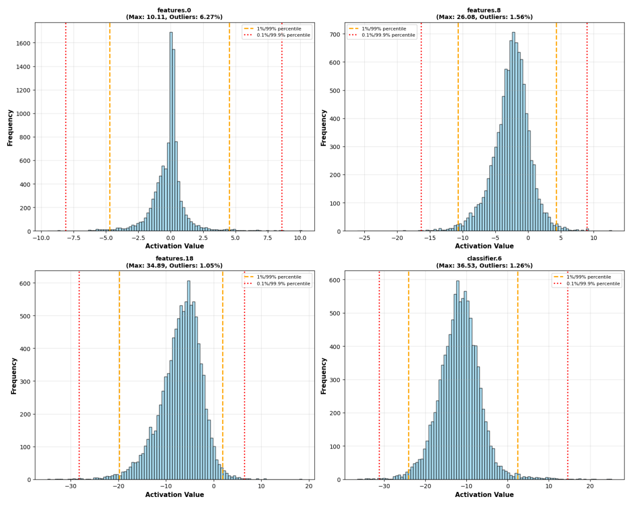

Figure 8. Visualization of activation distributions.