DOI: 10.54254/2755-2721/2025.PO25967

Model Optimization Techniques for Embedded Artificial Intelligence

Tianxu Hou

Nanjing University of Posts and Telecommunications, Nanjing, China
spherica_cherry@qq.com

Abstract. Deep neural networks (DNNs) are playing an important role in various areas including computer vision (CV) and natural language processing. This paper comprehensively analyzes model optimization techniques for deploying deep neural networks on resource-constrained embedded systems. We evaluate three core paradigms—pruning, quantization, and dynamic inference—focusing on their efficacy in balancing computational efficiency, memory footprint, and accuracy retention. For each technique, we conduct dedicated experiments spanning representative architectures including ResNet, VGG, Inception, and MobileNet variants to evaluate accuracy-FLOPS trade-offs. We also discuss and compare the practical deployment metrics for optimization techniques. Finally, we emphasize promising future directions.

Keywords: Embedded AI, Model Optimization, Pruning, Quantization, Dynamic Inference.

1. Introduction

The rapid proliferation of deep learning (DL) has revolutionized artificial intelligence, enabling breakthroughs in computer vision [1], natural language processing, and autonomous systems. Convolutional Neural Networks (CNNs), such as AlexNet and VGG, pioneered large-scale image recognition but demanded substantial computational resources, limiting their deployment in resource-constrained environments. Subsequent architectures like ResNet [1] introduced residual learning to mitigate vanishing gradients in deeper networks, while DenseNet [2] enhanced feature reuse through dense connectivity. These "big" models achieved state-of-the-art accuracy but incurred high parameter counts and computational costs (e.g., DenseNet161: 28.68M parameters, 7.79G operations [2]), rendering them impractical for embedded platforms.

In response, lightweight "little" models like MobileNetV2 [3] and ShuffleNet V2 [4] emerged, optimizing for efficiency through inverted residuals, channel shuffling, and aggressive parameter reduction. For instance, ShuffleNet V2 x0.5 utilizes only 1.37M parameters and 42.52M operations [4]. However, Table 1 reveals a critical trade-off: while these models reduce compute intensity, they exhibit severe accuracy degradation under lower-precision arithmetic—essential for embedded hardware acceleration. ShuffleNet variants suffer catastrophic drops (>65%) in fp16/int8 precision (e.g., ShuffleNet V2 x1.5: 80.91% fp32 vs. 10.03% fp16 [4]), contrasting sharply with robust models like ResNet50 (merely 0.24% int8 loss [2]).

Embedded AI deployments—drones, mobile robots, and smartphones—face stringent constraints: limited memory, energy budgets, and real-time latency requirements. High-parameter models (e.g., ResNet152: 60.19M parameters [2]) exhaust memory bandwidth, while intensive computations (e.g., Dense-Net201: 4.34G ops [1]) drain batteries and violate latency thresholds. Lower-precision inference (fp16/int8) alleviates these issues but amplifies accuracy volatility, as Table 1 quantifies. Such instability jeopardizes safety-critical applications; a drone's navigation system failing due to quantization errors could have dire consequences.

Table 1. Accuray and computation intensity analysis in terms of "big" and "little" DNNs. Int8 activations are calibrated using the test dataset with entropy based method. The accuracy variation is recorded in the parethesis for fp16 and int8, and marked bold if the degradation is larger than 1%

Model Param/Comp	Densenet 121 [1] (7.98M/2.87G)	Densenet 161 [1] (28.68M/7.79G)	Densenet 169 [1] (14.15M/3.40G)	Densenet 201 [1] (20.01M/4.34G)	Resnet50 [2] (25.56M/4.11G)	Resnet101 [2] (44.55M/7.83G)	Resnet152 [2] (60.19M/11.56G)
Acc.fp32	89.63	90.4	89.84	90.04	89.06	88.68	89.25
Acc.fp16	89.60 (0)	90.38 (-0.02)	89.85 (+0.01)	90.05 (+0.01)	89.04 (-0.02)	88.67 (-0.01)	89.25 (0)
Acc.int8	88.86 (-0.77)	89.92 (-0.48)	88.52 (-1.32)	87.39 (-2.65)	88.82 (-0.24)	88.57 (-0.11)	88.23 (-1.02)
Model Param/Comp	resnext50_32x4d [5] (25.03M/4.26G)	resnext101_32x8d [5] (88.79M/16.48G)	mobilenet_v2 [3] (3.51M/314.13M)	shufflenet_v2_x0_5 [4] (1.37M/42.52M)	shufflenet_v2_x1_0 [4] (2.28M/148.81M)	shufflenet_v2_x1_5 [4] (3.50M/301.29M)	shufflenet_v2_x2_0 [4] (7.39M/590.74M)
Acc.fp32	89.09	89.76	84.16	75.14	79.41	80.91	81.15
Acc.fp16	89.08 (-0.01)	89.76 (0)	84.14 (-0.02)	8.99 (-66.15)	8.99 (-70.42)	10.03 (-70.88)	9.86 (-71.29)
Acc.int8	89.07 (-0.02)	89.75 (-0.01)	82.67 (-1.49)	9.47 (-65.67)	9.26 (-70.15)	10.00 (-70.91)	9.93 (-71.22)

To bridge this gap, three key optimization paradigms have evolved:

- Pruning

Pruning is a fundamental model compression technique that eliminates redundant parameters or structural components from neural networks, significantly reducing computational complexity and memory footprint [6]. The objective of Model Pruning is to remove nodes and connections that contribute very little to the model's output.The pruned model can be significantly smaller than the full model, making it faster and much cheaper for inference,while using less energy.

- Quantization

Quantization reduces the bit-width of weights and activations in deep neural networks (DNNs), enabling efficient deployment on resource-constrained edge de-vices. It maps full-precision (FP32) values to low-bit integers (e.g., INT8), significantly compressing model size and accelerating inference.

- Dynamic inference

Different from conventional deep learning models with fixed computational graphs and parameters during inference, dynamic networks can adjust their structures and parameters in response to varying inputs, resulting in significant advantages regarding accuracy, computational efficiency, adaptability and more [7].

This work analyzes these techniques, evaluating their efficacy in mitigating the embedded deployment challenges highlighted by Table 1. We dissect algorithmic principles, trade-offs, and real-world applicability, setting the stage for a comprehensive discussion on optimizing DL for the embedded frontier.

2. Model optimization techniques

2.1. Pruning

Pruning is a critical model optimization technique aimed at enhancing the efficiency of neural networks while minimizing any degradation in quality. The primary objective is to maximize efficiency quantified through metrics such as FLOPs and compression ratio with the least possible reduction in accuracy [8]. As shown in Figure 1, which plots the accuracy of a pruned network against compression ratio, a compression ratio of 2 results in only a marginal accuracy drop. Beyond this point, higher compression ratios incur severe accuracy penalties. This demonstrates that pruning can achieve substantial efficiency improvements, such as reduced computational cost and model size, with negligible impact on performance, aligning well with the goal of embedded AI systems where resource constraints are paramount.

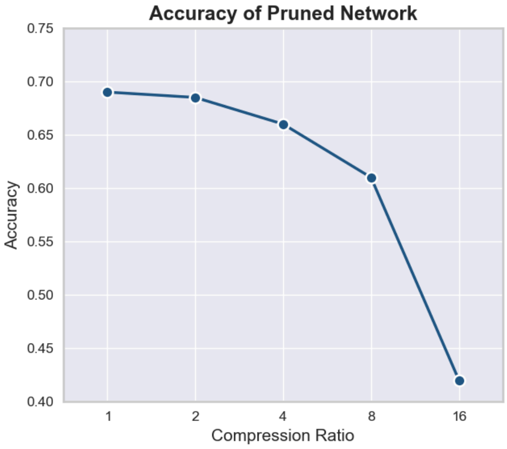


Figure 1. Accuracy of pruned network

Pruning techniques can be broadly classified into hardware-amenable and hardware-oblivious approaches, based on their compatibility with existing hardware platforms. Hardware-amenable pruning refers to methods that can be directly executed on general-purpose hardware like GPUs, whereas hardware-oblivious pruning necessitates specialized accelerators due to irregular memory access patterns. The distinction largely depends on the granularity and manner of parameter removal, as depicted in Figure 2 [9]. For hardware-oblivious solutions, such as fine-grained pruning, key advantages include a highly controllable compression rate, ease of implementation based on parameter ranking, and the potential to achieve extremely compact models. However, these methods suffer from significant drawbacks: they require specialized hardware and software libraries for deployment, exhibit unpredictable accuracy drops, and often involve tedious, aimless fine-tuning processes to recover performance, making them less practical for standard embedded environments without custom infrastructure [10-13].

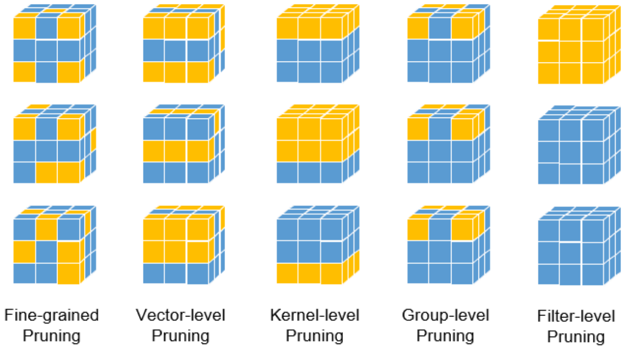


Figure 2. Pruning methods of different granularities

The application of pruning varies significantly across hardware architectures, with structured pruning formally denoting hardware-amenable approaches and unstructured pruning representing hardware-oblivious ones. Structured pruning, which removes entire neurons or filters in a regular pattern, is optimized for deployment on GPUs, leveraging their parallel processing capabilities [14-17]. In contrast, unstructured pruning, characterized by irregular weight removal, is suited for specialized hardware like ASICs and FPGAs, as visualized in Figure 3. This figure highlights how structured pruning focuses on pruned neurons for GPU execution, while unstructured pruning targets pruned synapses for implementation on accelerators, ensuring that embedded AI systems can achieve tailored efficiency gains based on the target hardware platform.

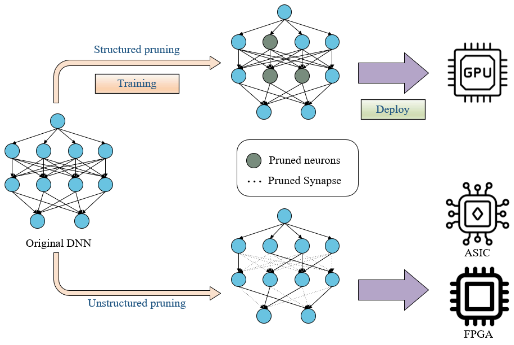


Figure 3. Pruning in hardware architecture

2.2. Quantization

Quantization, a fundamental model optimization technique, reduces the precision of neural network parameters and activations from floating-point representations (e.g., 32-bit) to lower-bit integers (e.g., 8-bit or 4-bit). This approach significantly accelerates on-device inference for embedded AI systems by minimizing computational latency, reducing energy consumption, and decreasing storage requirements—crucial for resource-constrained devices like mobiles and IoT sensors. It also enables efficient deployment on specialized hardware and GPUs by simplifying arithmetic operations. While the core goal is maintaining high accuracy alongside substantial efficiency gains in inference time, power usage, and model size, quantization methods vary: data-free quantization transforms weights/activations using only model-derived statistics (e.g., min-max ranges), offering simplicity but potential accuracy loss; calibration-based methods use small datasets to adjust quantization parameters (e.g., via entropy minimization), improving accuracy at the cost of dataset dependence;

and finetune-based quantization incorporates error simulation during training (e.g., using straight-through estimators), allowing weight adjustments for robustness but increasing training complexity [18-22].

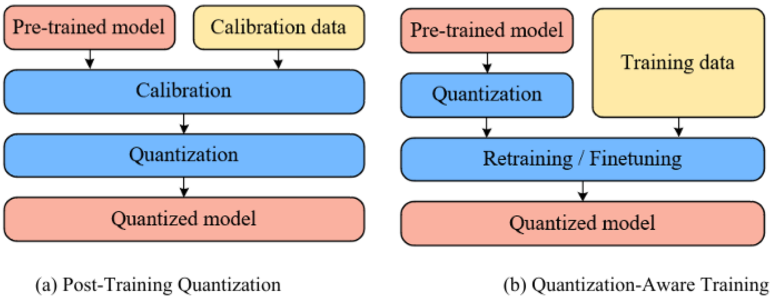


Figure 4. Llustration of PTQ and QAT

Quantization is further categorized by application stage: post-training quantization (PTQ) and quantization-aware training (QAT), with distinct workflows shown in Figure 4 (a) and Figure 4 (b) respectively. PTQ is applied after full-precision training, converting weights and activations directly to low precision; it typically uses a small calibration dataset to determine quantization parameters (e.g., via min-max or entropy methods) before final quantization [23-26]. This approach is computationally efficient, avoiding retraining overhead, but may incur accuracy loss due to unmodeled quantization effects during initial training. In contrast, QAT integrates quantization simulation during fine-tuning or retraining: starting from a pretrained model and using training data, it inserts fake quantization operators during the forward pass to mimic precision loss while computing gradients in full precision to iteratively adjust weights via backpropagation [27]. This co-optimization significantly enhances robustness to quantization, yielding higher accuracy than PTQ at the cost of substantial training resources and time [28,29]. Consequently, PTQ suits scenarios prioritizing rapid deployment with acceptable accuracy trade-offs, while QAT is preferred for applications demanding maximal precision in re-source-constrained systems.

2.3. Dynamic inference

Dynamic inference, also referred to as dynamic networks, is a model optimization technique that selectively skips redundant computations during inference based on the characteristics of input data, thereby enabling adaptive and efficient processing. This approach, often termed "dynamic inference," reduces computational overhead, latency, and energy consumption which critical for embedded AI systems with limited resources by avoiding unnecessary operations for simpler inputs while maintaining accuracy for complex ones. Dynamic inference represents a paradigm shift from static computational graphs by enabling neural networks to adapt their inference pathways in real-time based on input characteristics. This approach optimizes computational efficiency while maintaining accuracy through three primary mechanisms: sample-wise, spatial-wise, and temporal-wise adaptation.

2.3.1. Sample-wise dynamic inference

Sample-wise methods allocate computation per input sample, Dynamically adjusting network depth or width. The most common dynamic architecture is the early-exit network (Fig. 5), which has multiple intermediate classifiers attached to various internal layers [30-32]. The forward propagation can be halted when adequate confidence is achieved or when specific criteria are met at an internal classifier.

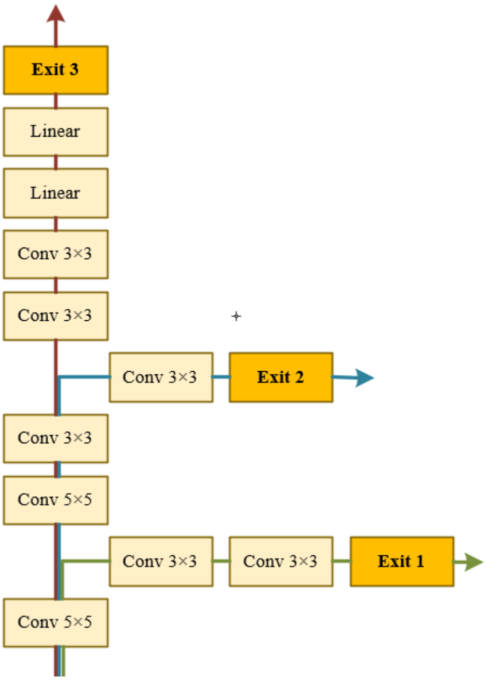
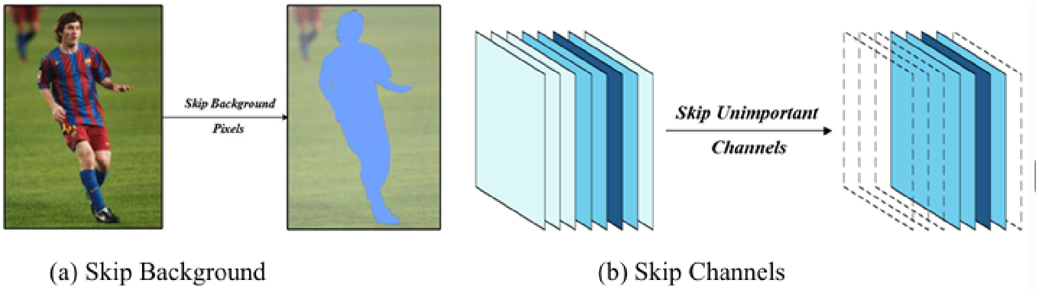


Figure 5. Early-exit

2.3.2. Spatial-wise dynamic inference

Spatial adaptation focuses on pixel or region-level efficiency. In convolutional layers, dynamic inference can be applied spatially by skipping computations on background pixels that contribute minimally to the output [33-35].For instance, Figure 6(a) demonstrates this spatial skipping, where non-salient regions are bypassed during convolution, focusing computation only on relevant areas to accelerate processing. Methods like Dynamic Convolutions [36-38] skip computations on less informative pixels using predicted spatial masks. For example, entropy-based masking achieves 2.5× speedup in semantic segmentation by processing only 40% of pixels [39].

Similarly, in the channel dimension, as shown in Figure 6(b), dynamic inference skips entire channels with low impact on the final result, such as those with negligible activation magnitudes, to reduce the number of operations per layer [40].



(a) Skip Background

(b) Skip Channels

Figure 6. Skip background and channels

These approaches can be combined for enhanced efficiency. Figure 7 depicts a unified method named Dynamic Dual Gating [41], a dynamic computing method, to reduce the model complexity at run-time. For each convolutional block, dual gating identifies the informative features along two separate dimensions, spatial and channel. Specifically, the spatial gating module estimates which areas are essential, and the channel gating module predicts the salient channels that contribute more to the results. Then the computation of both unimportant regions and irrelevant channels can be skipped dynamically during inference.

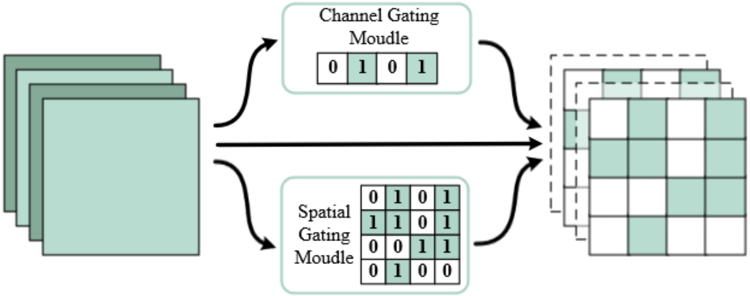


Figure 7. Ilustration of dual gating

2.3.3. Temporal-wise dynamic inference

Temporal-wise dynamic neural networks can unevenly allocate computation along the temporal dimension for sequential data, such as videos or time series data [42]. In the case of streaming data, such as videos, there is typically high correlation among nearby frames. Consequently, the dynamic focus on specific key frames is a crucial characteristic for deep learning models to reduce redundant computation [43].Furthermore, temporal-wise and spatial-wise adaptive computation could be implemented simultaneously to achieve higher efficiency [44].

3. Discussion

3.1. Experimental configuration

All experiments were rigorously designed to evaluate optimization techniques under resource-constrained conditions. The CIFAR-10 dataset, comprising 60,000 32×32 RGB images across 10 classes, served as the primary testbed for quantization and pruning analyses. For dynamic inference validation, we utilized the ImageNet-1k benchmark with 1.28 million training images spanning 1,000 object categories. Hardware execution was conducted on an NVIDIA TU102 GPU (24GB GDDR6 VRAM, 4,608 CUDA cores), leveraging TensorRT 8.6 for accelerated FP16/INT8 operations and cuDNN 8.9 for optimized kernel execution.

Prior to training, standardized preprocessing pipelines were implemented. For CIFAR-10, input images underwent normalization with per-channel mean and standard deviation, supplemented by random horizontal flipping and 32×32 center cropping. ImageNet samples were resized to 256×256 resolution followed by 224×224 center cropping. Quantization deployments employed TensorRT's post-training quantization workflow, where activation ranges were calibrated using 1,000 randomly selected test images through entropy minimization. Pruning experiments adopted gradual magnitude-based scheduling during fine-tuning, initializing from pre-trained FP32 checkpoints and applying sparsity ramping from 0% to 60% over 10 epochs.

3.2. Comparative analysis of individual optimization techniques

Our experimental evaluation systematically examines three core model optimization paradigms—quantization, pruning, and dynamic inference—across diverse neural architectures and datasets. The results demonstrate significant efficiency gains while highlighting nuanced trade-offs between computational performance and accuracy retention.

3.2.1. Quantization efficacy

As quantified in Table 2, INT8 precision delivers substantial inference acceleration across all models and batch sizes. For in-stance, ResNet50 achieves a 3.85× speedup at batch size 128, while MobileNet v1 attains a 6.21× improvement. This acceleration stems from reduced memory bandwidth requirements and streamlined integer operations, particularly beneficial for embedded hardware. Crucially, Table 3 reveals minimal accuracy degradation for most architectures: ResNet and VGG variants exhibit drops of <0.3%, and Inception v4 maintains near-identical accuracy. However, MobileNet v1 experiences a pronounced 1.55% decline, underscoring its sensitivity to precision reduction due to architectural constraints. Collectively, INT8 quantization balances high throughput with negligible accuracy loss for robust models but necessitates careful calibration for lightweight networks.

Table 2. Model training speed

Image/s	Batch size 1			Batch size 8			Batch size 128		
	FP32	FP16	Int 8	FP32	FP16	Int 8	FP32	FP16	Int 8
MobileNet v1	1509	2889	3762	2455	7430	13493	2718	8247	16885
MobileNet v2	1082	1618	2060	2267	5307	9016	2761	6431	12652
ResNet50 (v1.5)	298	617	1051	500	2045	3625	580	2475	4609
VGG-16	153	403	415	197	816	1269	236	915	1889
VGG-19	124	358	384	158	673	1101	187	749	1552
Inception v3	156	371	616	350	1318	2228	385	1507	2560
Inception v4	76	226	335	173	768	1219	186	853	1339
ResNet101	84	208	297	200	716	1253	233	899	1724

Table 3. Accuracy comparison of 8-bit quantized inference results

Model	FP32 Acc.(%)	Int 8 Acc.(%)	Acc. Drop(%)
MobileNet v1	71.01	69.46	1.55
MobileNet v2	74.08	73.85	0.23
NASNet (large)	82.72	82.66	0.06
NASNet (mobile)	73.97	73.4	0.57
ResNet50 (v1.5)	76.51	76.28	0.23
ResNet50 (v2)	76.37	76.22	0.15
ResNet152 (v1.5)	78.22	77.95	0.27
ResNet152 (v2)	78.45	78.15	0.30
VGG-16	70.89	70.82	0.07
VGG-19	71.01	70.85	0.16
Inception v3	77.99	77.85	0.14
Inception v4	80.19	80.16	0.03

3.2.2. Pruning advantages

Structured pruning via QAT yields compelling efficiency gains, as evidenced in Table 4. VGG-19 reduces FLOPs by 56% with only a 0.04% accuracy drop, while ResNet-50 achieves 54% FLOPs reduction at a 0.52% accuracy cost. This efficiency arises from eliminating redundant filters without disrupting critical feature pathways. Notably, MobileNet v2 shows limited FLOPs reduction despite aggressive pruning, indicating diminishing returns in highly optimized architectures. Pruning’s primary strength lies in its hardware compatibility: structured sparsity aligns with parallel processing units, enabling deployment without specialized accelerators.

Table 4. Comparison of different model pruning methods

Model	Baseline	Pruned		
	Acc.(%)	Acc.(%)	Acc.Drop(%)	FLOPs Reduction
VGG-16	73.63	73.54	0.09	41%
VGG-19	73.60	73.56	0.04	56%
ResNet-34	73.31	72.57	0.74	49%
ResNet-50	76.15	75.63	0.52	54%
MobileNet v2	72.0	71.80	0.20	28%

3.2.3. Dynamic inference flexibility

Table 5 highlights the adaptability of dynamic methods. DGNet excels for ResNet-50, reducing FLOPs by 57.2% while slightly improving accuracy. In contrast, FPGM and DynConv incur >1.3% accuracy penalties for similar FLOPs reductions. For MobileNet-v2, DGNet achieves 44% FLOPs savings with minimal degradation, outperforming MetaPruning. This variability underscores that dynamic inference’s efficacy hinges on sophisticated gating mechanisms—spatial-channel dual gating optimally skips redundant computations, whereas simpler policies struggle with accuracy preservation.

Table 5. Comparison of accelerated ResNet and MobileNet-V2 on ImageNet

Model	Method	Top 1 accuracy(%)			FLOPs
		Baseline	Accelerated	Acc↓	
ResNet-18	FPGM [45]	70.28	68.41	1.87	1.05E9(41.8%↓)
	LCCL [34]	69.98	66.33	3.65	1.23E9(34.6%↓)
	CGNet [46]	69.20	68.80	0.40	0.98E9(48.2%↓)
	DynConv [37]	69.76	66.97	2.79	1.08E9(41.5%↓)
	DGNet [41]	69.76	70.12	-0.36	9.54E8(49.4%↓)
ResNet-34	FPGM [45]	73.92	72.63	1.29	2.17E9(41.1%↓)
	LCCL [34]	73.42	72.99	0.43	2.78E9(24.8%↓)
	CGNet [46]	72.40	71.30	1.10	1.83E9(50.5%↓)
	DynConv [37]	73.31	71.75	1.56	2.01E9(44.1%↓)
	DGNet [41]	73.31	73.01	0.30	1.50E9(59.3%↓)
ResNet-50	FPGM [45]	76.15	74.83	1.32	1.91E9(53.5%↓)
	Hrank [40]	76.15	74.98	1.17	2.30E9(41.3%↓)
	ConvNet-AIG [32]	76.13	75.25	0.88	2.56E9(32.6%↓)
	DynConv [37]	76.13	74.40	1.73	2.25E9(42.4%↓)
	DGNet [41]	76.13	76.41	-0.28	1.65E9(57.2%↓)
MobileNet-v2	MetaPruning [47]	71.88	71.20	0.68	2.17E8(22.5%↓)
	DGC [48]	72.00	70.70	1.30	2.45E8(18.3%↓)
	DGNet [41]	71.88	71.62	0.26	1.60E8(44.0%↓)

3.3. Cross-technique trade-offs and deployment implications

The interplay between optimization techniques reveals critical design considerations for embedded systems, synthesized in Table 6.

Quantization dominates in memory reduction and hardware acceleration potential but risks instability in lightweight models. Pruning offers moderate memory savings yet excels in FLOPs reduction with better generalization but demands iterative retraining.

For latency-critical edge devices, INT8 quantization is preferable; for energy-constrained systems where FLOPs directly correlate with power draw, pruning is superior.

Dynamic Inference as a hybrid solution, uniquely adapts computational cost per input, achieving 30–60% FLOPs reduction with near-zero accuracy loss. However, its runtime overhead and dependency on programmable logic limit deployment on fixed-function hardware. It is ideal for heterogeneous inputs but less suited for uniform data streams.

Consequently, no single optimization technique universally dominates, as each addresses distinct aspects of efficiency: quantization maximizes hardware utilization by leveraging low-precision arithmetic, pruning enhances computational efficiency through targeted sparsity, and dynamic inference optimizes re-source allocation by adapting to real-time demands. Therefore, for holistic embedded deployment, we recommend a context-dependent strategy: prioritizing quantization-first for platforms with dedicated hardware integer acceleration support to exploit its latency and energy benefits; employing structured pruning for GPU-accelerated platforms where its predictable sparsity patterns align well with parallel architectures; and adopting dynamic inference in scenarios characterized by significant workload variability to dynamically conserve resources during periods of lower demand, thereby optimizing the overall system efficiency.

Table 6. Comparison of compression techniques

Technique	Primary Goal	Advantages	Disadvantages	Memory Reduction	Computation Cost Reduction	Accuracy Degradation	Hardware Adaptability	Edge Suitability
Dynamic Inference	Adapt computational pathways based on input characteristics	Preserves accuracy via input-adaptive processing; Reduces redundant computation; Enables real-time resource allocation	Runtime policy overhead; Vulnerability to adversarial samples; Complex training requirements	Low	Moderate to High (30-60%)	Very Low to Low (Δ<0.5 %)	Moderate (requires programmable logic)	Moderate to High
Quantization	Reduce numerical precision	High hardware acceleration; Significant memory/computation savings; Native support on edge accelerators	Catastrophic failure in lightweight models; Calibration complexity; Precision-dependent accuracy instability	High to Very High (2-8×)	Moderate to Very High	Very Low (QAT) to High (PTQ)	Very High	Very High
Pruning	Eliminate redundant parameters	Improved generalization; Compatible with structured hardware; Reduced energy consumption	Unstructured sparsity underutilizes hardware; Iterative retraining overhead; Architecture-dependent efficacy	Moderate (structured) to Very High (unstructured)	Low to Very High	Very Low to Medium	Low (unstructured) to High (structured)	Moderate to High

4. Conclusion

This study comprehensively evaluates three pivotal model optimization paradigms—quantization, pruning, and dynamic inference—for deploying deep neural networks on resource-constrained embedded systems. Through rigorous experimentation across diverse architectures and datasets, we validate that each technique uniquely addresses the trilemma of computational efficiency, memory footprint, and accuracy retention. Quantization emerges as the foremost solution for memory compression and hardware acceleration. However, its vulnerability in lightweight architectures necessitates careful calibration, underscoring the critical role of model robustness in low-precision deployment.

Pruning, particularly structured variants via QAT, demonstrates exceptional efficacy in computational reduction. Its hard-ware compatibility with GPUs positions it as an ideal candidate for energy-constrained systems where FLOPs directly correlate with power consumption. Conversely, dynamic inference techniques like DGNet offer adaptive efficiency, dynamically reducing computation by 30–60% for variable-input scenarios while occasionally enhancing accuracy, albeit at the cost of runtime overhead and dependency on programmable logic.

Crucially, no single technique universally dominates; instead, their synergistic integration—such as quantized sparse models or hardware-aware dynamic policies—represents the most promising path toward ultra-efficient embedded AI. Future work must prioritize co-designing optimization techniques with lightweight model architectures to mitigate precision sensitivity. Cross-technique interoperability is under-explored. While quantization and pruning individually excel in memory and computation re-duction, their combined application often triggers cascading errors. Emerging frameworks must establish unified optimization methods.

References

[1] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, 770-778.

[2] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition , 4700-4708.

[3] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition , 4510-4520.

[4] Ma, N., Zhang, X., Zheng, H. T., & Sun, J. (2018). Shufflenet v2: Practical guidelines for efficient cnn architecture design. In Proceedings of the European conference on computer vision (ECCV) , 116-131.

[5] Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2017). Aggregated residual transformations for deep neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, 1492-1500.

[6] LeCun, Y., Denker, J., & Solla, S. (1989). Optimal brain damage. Advances in neural information processing systems, 2.

[7] Han, Y., Huang, G., Song, S., Yang, L., Wang, H., & Wang, Y. (2021). Dynamic neural networks: A survey. IEEE transactions on pattern analysis and machine intelligence, 44(11), 7436-7456.

[8] Li, Y., Gu, S., Mayer, C., Gool, L. V., & Timofte, R. (2020). Group sparsity: The hinge between filter pruning and decomposition for network compression. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition , 8018-8027.

[9] Mao, H., Han, S., Pool, J., Li, W., Liu, X., Wang, Y., & Dally, W. J. (2017). Exploring the regularity of sparse structure in convolutional neural networks. arXiv preprint arXiv: 1705.08922.

[10] Wen, W., Wu, C., Wang, Y., Chen, Y., & Li, H. (2016). Learning structured sparsity in deep neural networks. Advances in neural information processing systems, 29.

[11] Luo, J. H., Wu, J., & Lin, W. (2017). Thinet: A filter level pruning method for deep neural network compression. In Proceedings of the IEEE international conference on computer vision , 5058-5066.

[12] Zhu, J., Zhao, Y., & Pei, J. (2021). Progressive kernel pruning based on the information mapping sparse index for CNN compression. IEEE Access, 9, 10974-10987.

[13] Polyak, A., & Wolf, L. (2015). Channel-level acceleration of deep face representations. IEEE Access, 3, 2163-2175.

[14] Anwar, S., Hwang, K., & Sung, W. (2017). Structured pruning of deep convolutional neural networks. ACM Journal on Emerging Technologies in Computing Systems (JETC), 13(3), 1-18.

[15] Zhou, A., Ma, Y., Zhu, J., Liu, J., Zhang, Z., Yuan, K., ... & Li, H. (2021). Learning n: m fine-grained structured sparse neural networks from scratch. arXiv preprint arXiv: 2102.04010.

[16] Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., & Zhang, C. (2017). Learning efficient convolutional networks through network slimming. In Proceedings of the IEEE international conference on computer vision, 2736-2744.

[17] He, Y., Lin, J., Liu, Z., Wang, H., Li, L. J., & Han, S. (2018). Amc: Automl for model compression and acceleration on mobile devices. In Proceedings of the European conference on computer vision (ECCV) , 784-800.

[18] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., ... & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE conference on computer vision and pattern recognition , 2704-2713.

[19] Reed, J., DeVito, Z., He, H., Ussery, A., & Ansel, J. (2022). torch. fx: Practical program capture and transformation for deep learning in python. Proceedings of Machine Learning and Systems, 4, 638-651.

[20] Siddegowda, S., Fournarakis, M., Nagel, M., Blankevoort, T., Patel, C., & Khobare, A. (2022). Neural network quantization with ai model efficiency toolkit (aimet). arXiv preprint arXiv: 2201.08442.

[21] Hubara, I., Nahshan, Y., Hanani, Y., Banner, R., & Soudry, D. (2021, July). Accurate post training quantization with small calibration sets. In International Conference on Machine Learning , 4466-4475.

[22] He, X., & Cheng, J. (2018). Learning compression from limited unlabeled data. In Proceedings of the European Conference on Computer Vision (ECCV) , 752-769.

[23] Hubara, I., Nahshan, Y., Hanani, Y., Banner, R., & Soudry, D. (2020). Improving post training neural quantization: Layer-wise calibration and integer programming. arXiv preprint arXiv: 2006.10518.

[24] Choukroun, Y., Kravchik, E., Yang, F., & Kisilev, P. (2019, October). Low-bit quantization of neural networks for efficient inference. In 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW) , 3009-3018.

[25] Fang, J., Shafiee, A., Abdel-Aziz, H., Thorsley, D., Georgiadis, G., & Hassoun, J. H. (2020). Post-training piecewise linear quantization for deep neural networks. In Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16 , 69-86.

[26] Kullback, S., & Leibler, R. A. (1951). On information and sufficiency. The annals of mathematical statistics, 22(1), 79-86.

[27] Choi, J., Wang, Z., Venkataramani, S., Chuang, P. I. J., Srinivasan, V., & Gopalakrishnan, K. (2018). Pact: Parameterized clipping activation for quantized neural networks. arXiv preprint arXiv: 1805.06085.

[28] Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., & Zou, Y. (2016). Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint arXiv: 1606.06160.

[29] Esser, S. K., McKinstry, J. L., Bablani, D., Appuswamy, R., & Modha, D. S. (2019). Learned step size quantization. arXiv preprint arXiv: 1902.08153.

[30] Graves, A. (2016). Adaptive computation time for recurrent neural networks. arXiv preprint arXiv: 1603.08983.

[31] Wang, X., Yu, F., Dou, Z. Y., Darrell, T., & Gonzalez, J. E. (2018). Skipnet: Learning dynamic routing in convolutional networks. In Proceedings of the European conference on computer vision (ECCV) , 409-424.

[32] Veit, A., & Belongie, S. (2018). Convolutional networks with adaptive inference graphs. In Proceedings of the European conference on computer vision (ECCV), 3-18.

[33] Ren, M., Pokrovsky, A., Yang, B., & Urtasun, R. (2018). Sbnet: Sparse blocks network for fast inference. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition , 8711-8720.

[34] Dong, X., Huang, J., Yang, Y., & Yan, S. (2017). More is less: A more complicated network with less inference complexity. In Proceedings of the IEEE conference on computer vision and pattern recognition, 5840-5848.

[35] Cao, S., Ma, L., Xiao, W., Zhang, C., Liu, Y., Zhang, L., ... & Yang, Z. (2019). Seernet: Predicting convolutional neural network feature-map sparsity through low-bit quantization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition , 11216-11225.

[36] Kong, S., & Fowlkes, C. (2019, January). Pixel-wise attentional gating for scene parsing. In 2019 IEEE Winter Conference on Applications of Computer Vision (WACV) , 1024-1033.

[37] Verelst, T., & Tuytelaars, T. (2020). Dynamic convolutions: Exploiting spatial sparsity for faster inference. In Proceedings of the ieee/cvf conference on computer vision and pattern recognition , 2320-2329.

[38] Xie, Z., Zhang, Z., Zhu, X., Huang, G., & Lin, S. (2020). Spatially adaptive inference with stochastic feature sampling and interpolation. In Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16 , 531-548.

[39] Xie, Z., Zhang, Z., Zhu, X., Huang, G., & Lin, S. (2020). Spatially adaptive inference with stochastic feature sampling and interpolation. In Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16 , 531-548.

[40] Lin, M., Ji, R., Wang, Y., Zhang, Y., Zhang, B., Tian, Y., & Shao, L. (2020). Hrank: Filter pruning using high-rank feature map. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition , 1529-1538.

[41] Li, F., Li, G., He, X., & Cheng, J. (2021). Dynamic dual gating neural networks. In Proceedings of the IEEE/CVF International Conference on Computer Vision , 5330-5339.

[42] Campos, V., Jou, B., Giró-i-Nieto, X., Torres, J., & Chang, S. F. (2017). Skip rnn: Learning to skip state updates in recurrent neural networks. arXiv preprint arXiv: 1708.06834.

[43] Hansen, C., Hansen, C., Alstrup, S., Simonsen, J. G., & Lioma, C. (2019). Neural speed reading with structural-jump-lstm. arXiv preprint arXiv: 1904.00761.

[44] Tao, J., Thakker, U., Dasika, G., & Beu, J. (2019, November). Skipping rnn state updates without retraining the original model. In Proceedings of the 1st workshop on machine learning on edge in sensor systems, 31-36.

[45] He, Y., Liu, P., Wang, Z., Hu, Z., & Yang, Y. (2019). Filter pruning via geometric median for deep convolutional neural networks acceleration. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition , 4340-4349.

[46] Hua, W., Zhou, Y., De Sa, C. M., Zhang, Z., & Suh, G. E. (2019). Channel gating neural networks. Advances in neural information processing systems, 32.

[47] Liu, Z., Mu, H., Zhang, X., Guo, Z., Yang, X., Cheng, K. T., & Sun, J. (2019). Metapruning: Meta learning for automatic neural network channel pruning. In Proceedings of the IEEE/CVF international conference on computer vision , 3296-3305.

[48] Su, Z., Fang, L., Kang, W., Hu, D., Pietikäinen, M., & Liu, L. (2020). Dynamic group convolution for accelerating convolutional neural networks. In Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VI 16 , 138-155.