A Controlled Evaluation with a Last-Stage Multiscale Variant

Zhaoyang Chen*, Tuchao Li
College of Artificial Intelligence and Big Data, Guangzhou Vocational University of Science and Technology, Guangzhou, Guangdong, China
**Corresponding Author*

Abstract: Efficiency claims for compact convolutional networks are commonly supported by parameter counts and floating-point operation estimates, although neither metric directly measures execution time. This study examines how accuracy, static complexity, and observed GPU runtime align for three established lightweight networks and one controlled architectural variant. MobileNetV2, MobileNetV3-Small, ShuffleNetV2, and a MobileNetV3 variant with a parallel multi-scale module in its final inverted-residual block were trained from scratch on CIFAR-10 and CIFAR-100. A fixed data split and training recipe were used throughout, with three independent random seeds per model and dataset, yielding 24 formal runs. ShuffleNetV2 produced the highest mean test accuracy on both datasets, reaching 93.76% on CIFAR-10 and 74.24% on CIFAR-100. MobileNetV2 was fastest in the hardware benchmark despite requiring approximately four times the FLOPs of MobileNetV3. The multi-scale intervention increased CIFAR-10 accuracy by only 0.39 percentage points while adding 45.09% more parameters and 43.89% more FLOPs. On CIFAR-100, it reduced accuracy by 4.18 percentage points and also lowered throughput. The findings show that low FLOPs should not be treated as a sufficient proxy for runtime and that a multi-scale operator can become counterproductive when its placement and fusion cost disturb an already compact representation.

Keywords: Compact Neural Networks; Hardware-aware Evaluation; Inference Latency; Depthwise Convolution; Model Benchmarking; Small-Image Classification

1. Introduction

Compact convolutional neural networks are designed to make visual recognition feasible under constrained computation and storage. Their success has produced architecture families with markedly different design choices. Modern mobile CNNs continue to use depthwise convolution as a central operation for efficient spatial processing [1]. MobileNetV2 organizes this operation through inverted residuals and linear bottlenecks [2], while MobileNetV3 combines architecture search, squeeze-and-excitation operations, and hardware-oriented nonlinearities [3]. ShuffleNet follows another route, using grouped pointwise convolution and channel shuffle [4], and ShuffleNetV2 places greater emphasis on practical execution characteristics [5].

Comparisons among these networks often reduce efficiency to one or two static quantities. Parameter count describes storage demand, and floating-point operations (FLOPs) approximate arithmetic work for a specified input. Both are useful, but neither models memory traffic, kernel-launch overhead, graph fragmentation, operator implementation, or parallel occupancy. Ma et al. demonstrated that architectures with similar FLOPs can exhibit different speeds because computation is not the only cost incurred by a device [5]. MobileNetV3 likewise incorporated latency on target mobile hardware into its design process [3]. Consequently, an architecture described as efficient in one software-hardware setting may not retain the same ranking on another.

A second issue concerns feature-enhancement modules. Multi-scale convolution is attractive because different kernel sizes expose a network to different spatial contexts. InceptionNeXt, for example, combines channel-split depthwise branches with different kernel shapes to improve spatial processing [6]. Yet the term multi-scale describes a family of operations rather than a guarantee of better representation. The result depends on where the operator is inserted, how channels are allocated, and how branch outputs are fused. Adding two full-width branches late in

a compact network may create substantial pointwise-convolution cost and may alter the information path that the original bottleneck was designed to preserve.

This work investigates these two issues in a controlled small-image setting. Three research questions guide the study:

The objective is not to establish a new state of the art. Instead, the study isolates model architecture as far as practical by holding data splits, augmentation, optimization, training length, and random-seed sets constant. It makes three contributions. First, it reports a 24-run comparison across two datasets and four model configurations. Second, it pairs static complexity with synchronized GPU timing. Third, it documents a cross-dataset reversal in the effect of a last-stage multi-scale intervention. The reversal is useful because it identifies a design condition under which additional receptive-field diversity does not compensate for extra transformation and fusion.

2. Technical Background

2.1 Lightweight Convolutional Architectures

Depthwise separable convolution factorizes spatial and cross-channel processing. A depthwise layer applies one spatial filter per input channel, after which a pointwise layer mixes channels. Xception examined this factorization as a central architectural operation, and MobileNetV2 used it to reduce the computational burden of mobile vision models. The factorization greatly lowers arithmetic count, but its performance on a particular device depends on whether the depthwise kernel and subsequent pointwise operation can use the available hardware efficiently.

MobileNetV2 reorganizes the factorization into an inverted-residual block. A narrow input is expanded with a pointwise convolution, processed spatially with a depthwise convolution, and projected back to a narrow output. The last projection is linear, avoiding an additional nonlinearity in the bottleneck representation. Residual connections are used when stride and channel dimensions permit. This structure separates the high-dimensional space used for nonlinear transformation from the low-dimensional space carried between blocks.

MobileNetV3-Small retains the inverted-residual principle but introduces a more heterogeneous graph [3]. Selected blocks use squeeze-and-excitation attention, which recalibrates channels from a pooled descriptor [7], and h-swish replaces more expensive smooth activations in parts of the network. These choices were developed with mobile latency in mind. They do not imply that MobileNetV3 will minimize latency on a desktop GPU at 32 by 32 pixels, where operator-launch and memory effects can dominate a small arithmetic workload.

ShuffleNet reduces pointwise-convolution cost with groups and restores cross-group communication through channel shuffle. ShuffleNetV2 revises the design around practical speed guidelines, including balanced channel widths, lower fragmentation, and reduced use of costly elementwise or grouped operations. GhostNet later pursued a related efficiency goal by generating part of the feature set through inexpensive transformations. Together, these studies motivate an evaluation that separates parameter efficiency, arithmetic efficiency, and measured execution efficiency.

2.2 Multi-Scale Processing and Efficiency Criteria

Multi-scale processing introduces receptive fields of different sizes within the same stage. Its implementation, however, determines both the information path and the computational burden. A channel-split design assigns different spatial operators to parallel channel groups, as in InceptionNeXt [6], whereas a full-width parallel design applies every kernel to the entire tensor. The latter exposes each channel to multiple spatial neighborhoods but duplicates spatial processing and increases the input width of the fusion layer. Placement also matters. Earlier stages retain larger spatial maps and encode local patterns, while later stages contain more channels and more class-specific representations. Consequently, a late intervention can be inexpensive in spatial dimensions yet costly in channel fusion and potentially disruptive to a mature representation.

Efficiency must therefore be evaluated along several dimensions [8]. Parameter count approximates storage requirements, FLOPs estimate arithmetic work under a specified counting convention, and measured latency or throughput describes execution on a particular hardware and software stack. FasterNet demonstrates that low FLOPs alone do not guarantee low latency on modern hardware [9]. ShuffleNetV2 similarly emphasizes memory

access and graph fragmentation in addition to arithmetic count [5]. In this study, the same principle is applied to a local architectural modification: the multi-scale branch is considered beneficial only if its accuracy effect is commensurate with its parameter, FLOP, and runtime overhead.

3. Experimental Method

3.1 Datasets and Split Control

CIFAR-10 and CIFAR-100 each contain 60,000 color images at a resolution of 32 by 32 pixels. Both provide 50,000 official training images and 10,000 official test images; CIFAR-10 contains 10 classes and CIFAR-100 contains 100. These benchmarks are widely used for controlled comparisons of compact image-classification architectures [10]. The shared image count and resolution make the pair useful for examining a change in class granularity without changing input dimensions.

The official training set was divided into 45,000 training images and 5,000 validation images using a fixed split seed of 12,345. The resulting index file was reused by every model and random seed. The official test set remained untouched during training and model selection. Training images were padded by four pixels, randomly cropped back to 32 by 32, and randomly flipped horizontally. Dataset-specific channel normalization was then applied. Validation and test images received normalization only.

3.2 Models and CIFAR Adaptation

The comparison included MobileNetV2, ShuffleNetV2 1.0x, MobileNetV3-Small, and the proposed experimental variant, denoted MS-MobileNetV3. All models were trained from random initialization. MobileNetV2 and ShuffleNetV2 used the implementations distributed with Torchvision. Their input stems were adapted to preserve spatial detail: the initial convolution stride was changed from 2 to 1 for both models, and the initial max-pooling operation was removed from ShuffleNetV2. MobileNetV3-Small used a 3 by 3 stem with stride 1. The output layer was set to 10 or 100 classes as required.

These changes define the scope of the comparison. The reported values are not official ImageNet results and should not be read as a reproduction of the original papers. They describe CIFAR-adapted implementations trained under one common recipe.

3.3 Last-Stage Multi-Scale Intervention

MS-MobileNetV3 modifies only the final inverted-residual block of MobileNetV3-Small. Let the expanded feature map entering the added module be

$$X \in \mathbb{R}^{C\times H\times W} \quad (1)$$

The two branches are

$$B_3(X) = \mathrm{ReLU}\left(\mathrm{BN}\left(\mathrm{DWConv}_{3\times 3}(X)\right)\right) \quad (2)$$

$$B_5(X) = \mathrm{ReLU}\left(\mathrm{BN}\left(\mathrm{DWConv}_{5\times 5}(X)\right)\right) \quad (3)$$

The branch outputs are concatenated and fused as

$$\mathrm{ReLU}\left(\mathrm{BN}\left(\mathrm{Conv}_{1\times 1}\left([B_3(X), B_5(X)]\right)\right)\right) \quad (4)$$

The fused feature then passes through the remaining activation, squeeze-and-excitation, and linear projection of the original block. The block-level residual connection is retained whenever its original stride and channel conditions are satisfied. Unlike the channel-split Inception depthwise design [6], both branches process all C channels and the fusion layer maps 2C inputs back to C outputs.

3.4 Training Protocol

Each model-dataset combination was run with seeds 42, 2026, and 3407. All 24 runs used 300 epochs and a batch size of 1,024. Stochastic gradient descent was configured with an initial learning rate of 0.2, momentum of 0.9, and weight decay of 5 x 10-4. The first five epochs used linear warm-up, followed by cosine decay, following the general scheduling principle formalized in SGDR [11].

The objective was cross-entropy with label smoothing of 0.1. Label smoothing was introduced as a regularization method for classification in the Inception architecture [12]. Mixup was applied with alpha=0.2, forming convex combinations of input pairs and their labels [13]. Training used automatic mixed precision. Python, NumPy, PyTorch, CUDA, and data-loader random generators were seeded for each run, and deterministic cuDNN behavior was enabled during training.

For a sample with class index y, the smoothed cross-entropy used in every run was

$$\mathcal{L}_{LS} = -(1-\varepsilon)\log p_y - \frac{\varepsilon}{K}\sum_{k\neq y}\log p_k,\ \varepsilon = 0.1 \quad (5)$$

where p_k is the predicted probability for class k and K is the number of classes. For Mixup, two training pairs were combined with λ sampled from Beta(0.2, 0.2):

$$\tilde{x} = \lambda x_i + (1-\lambda)x_j, \quad \tilde{y} = \lambda y_i + (1-\lambda)y_j \quad (6)$$

The optimized objective was the corresponding convex combination of the two smoothed losses:

$$\mathcal{L}_{mix} = \lambda \mathcal{L}_{LS}(f(\tilde{x}), y_i) + (1-\lambda)\mathcal{L}_{LS}(f(\tilde{x}), y_j) \quad (7)$$

The experiments ran on Linux with Python 3.12.3, PyTorch 2.3.0, CUDA 12.1, and one NVIDIA GeForce RTX 4090. PyTorch provided the model execution and automatic differentiation framework [14]. The best validation checkpoint was retained for each run, and the official test set was evaluated after selection.

3.5 Metrics and Runtime Procedure

Test accuracy is reported as the mean and sample standard deviation over three seeds. Trainable parameters were counted directly. FLOPs were estimated for one forward pass with a 3x32x32 input. The small number of seeds supports a descriptive comparison but not a high-powered inferential analysis. Accordingly, the study emphasizes effect direction, magnitude, and consistency across seeds rather than treating a small-sample statistical test as decisive.

Runtime was measured with the seed-42 best checkpoint for each architecture and dataset. Inputs were generated in advance and kept on the GPU, so timing excluded model loading and host-to-device transfer. The benchmark used FP32, cuDNN benchmarking, 200 warm-up iterations, and five measured repeats of 500 forward passes. CUDA events and synchronization bounded the measured region. Mean batch latency was reported for batch size 1, and images per second were reported for batch size 128.

4. Results

4.1 Accuracy Ranking Across Datasets

Table 1 reports test accuracy and static complexity. ShuffleNetV2 ranked first on both datasets. Its mean accuracy was 93.76% on CIFAR-10 and 74.24% on CIFAR-100. MobileNetV2 ranked second at 93.12% and 72.05%. MobileNetV3-Small reached 88.24% and 64.75%, while MS-MobileNetV3 reached 88.63% and 60.57%.

The standard deviation remained below 0.60 percentage points for every configuration. This level of variation does not eliminate uncertainty from having only three runs, but it shows that the broad ranking was not produced by one unstable seed. The separation between ShuffleNetV2 or MobileNetV2 and the two MobileNetV3 configurations became larger on CIFAR-100.

Table 1. Test Accuracy and Static Model Complexity

Dataset	Model	Test accuracy (%)	Parameters (M)	FLOPs (G)
CIFAR-10	MobileNetV2	93.12 ± 0.15	2.237	0.0266
CIFAR-10	ShuffleNetV2	93.76 ± 0.13	1.264	0.0473
CIFAR-10	MobileNetV3	88.24 ± 0.35	1.523	0.0063
CIFAR-10	MS-MobileNetV3	88.63 ± 0.33	2.209	0.0090
CIFAR-100	MobileNetV2	72.05 ± 0.54	2.352	0.0268
CIFAR-100	ShuffleNetV2	74.24 ± 0.37	1.356	0.0473
CIFAR-100	MobileNetV3	64.75 ± 0.59	1.615	0.0064
CIFAR-100	MS-MobileNetV3	60.57 ± 0.59	2.302	0.0091

4.2 Static Efficiency Does Not Determine GPU Speed

The results show three distinct forms of efficiency. ShuffleNetV2 had the fewest parameters and the highest accuracy, but also the largest FLOP count. MobileNetV3 had the fewest FLOPs by a wide margin but substantially lower accuracy.

The hardware results further changed the ranking. As shown in Table 2 and Figure 1, MobileNetV2 had the lowest latency and highest throughput on both datasets. For CIFAR-10, its 2.902 ms latency was about 25.4% lower than MobileNetV3's 3.893 ms latency. Its throughput was approximately 33.6% higher. The corresponding CIFAR-100 advantages were about 24.0% in latency and 29.0% in throughput. FLOPs therefore failed to predict the observed ordering. MobileNetV3 required roughly one quarter of the FLOPs of MobileNetV2, but it executed more slowly. ShuffleNetV2 provides another instructive case: it had the largest FLOP estimate, yet it ran faster than either MobileNetV3 configuration. The benchmark does not identify one isolated cause, but it is consistent with the hardware-aware view that arithmetic count omits memory access, kernel efficiency, operator fragmentation, and device utilization [5].

Table 2. FP32 Inference Performance

Dataset	Model	Batch-1 latency (ms/image)	Batch-128 throughput (images/s)
CIFAR-10	MobileNetV2	2.902 ± 0.027	42,505 ± 526
CIFAR-10	ShuffleNetV2	3.603 ± 0.008	33,753 ± 435
CIFAR-10	MobileNetV3	3.893 ± 0.028	31,813 ± 89
CIFAR-10	MS-MobileNetV3	4.099 ± 0.011	30,274 ± 78
CIFAR-100	MobileNetV2	3.025 ± 0.095	41,240 ± 476
CIFAR-100	ShuffleNetV2	3.755 ± 0.063	32,708 ± 429
CIFAR-100	MobileNetV3	3.978 ± 0.030	31,981 ± 486
CIFAR-100	MS-MobileNetV3	4.059 ± 0.030	30,607 ± 200

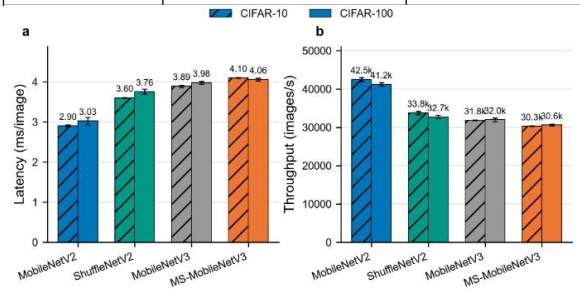

Figure 1. FP32 Inference Efficiency

4.3 The Intervention Changes Direction with Dataset Difficulty

The effect of the multi-scale module is summarized by the per-seed results. On CIFAR-10, the per-seed changes relative to MobileNetV3 were +0.53, -0.02, and +0.67 percentage points. The mean gain was 0.39 points. One run was effectively unchanged, so the result does not support a claim of uniform improvement.

On CIFAR-100, the changes were -3.94, -3.37, and -5.23 points. The mean loss was 4.18 points, and all three seeds moved in the same direction. The static cost also increased: parameters rose by 42.51% and FLOPs by 43.26%. MS-MobileNetV3 was therefore strictly worse than MobileNetV3 on CIFAR-100 under the three reported dimensions of accuracy, parameters, and FLOPs.

The runtime results reinforce this conclusion. Relative to MobileNetV3, the intervention increased CIFAR-10 batch-1 latency by approximately 5.3% and reduced batch-128 throughput by 4.8%. On CIFAR-100, latency increased by about 2.1% and throughput fell by 4.3%. The module did not provide a hardware-efficiency benefit that could offset the loss in accuracy.

5. Discussion

5.1 Accuracy and Dataset Difficulty

The ranking of the four configurations was consistent across random seeds, but the magnitude of their separation depended strongly on dataset difficulty. ShuffleNetV2 and MobileNetV2 remained close on CIFAR-10, where both exceeded 93% mean test accuracy. Their advantage became more pronounced on CIFAR-100, which distributes the same number of training images across ten times as many classes. Under this fixed training recipe, the compact MobileNetV3-Small representation was less competitive, and the additional multi-scale branch did not recover the gap. This pattern suggests that the usefulness of a structural modification cannot be inferred from one low-resolution benchmark alone. A small improvement on CIFAR-10 may reflect sufficient capacity for coarse class separation, while CIFAR-100 exposes whether the same representation supports finer inter-class discrimination.

5.4 Scope and Practical Implications

Several boundaries should be considered when applying these findings. First, CIFAR-10 and CIFAR-100 contain small 32 by 32 images, so architecture rankings may change for ImageNet-scale classification, detection, or segmentation. Second, three seeds characterize run-to-run variation but do not support high-powered statistical inference. The analysis therefore emphasizes effect magnitude and directional consistency. Third, the common training recipe improves comparability but may not be optimal for every backbone. Architecture-specific augmentation, regularization, or learning-rate tuning could change absolute accuracy. Fourth, runtime was measured only in FP32 on one GPU and excluded preprocessing and host-to-device transfer; end-to-end application latency may differ.

6. Conclusion

A controlled comparison of four lightweight CNN configurations on CIFAR-10 and CIFAR-100 produced three main findings. ShuffleNetV2

gave the highest test accuracy and the lowest parameter count. MobileNetV2 delivered the lowest latency and highest throughput. MobileNetV3 required the fewest FLOPs but was not the fastest model. These results demonstrate why static complexity and observed runtime should be reported separately.

The last-stage multi-scale intervention did not produce a favorable trade-off. Its small and seed-dependent CIFAR-10 gain required substantially more parameters and computation, while CIFAR-100 accuracy decreased for every seed. The added branch also reduced measured throughput. Multi-scale processing should therefore be evaluated as a placement- and fusion-dependent intervention rather than assumed to improve a compact backbone. For practical model selection, accuracy, storage, arithmetic cost, and target-platform runtime must be considered together.

Acknowledgments

This work was supported by the 2026 University Research Project of Guangzhou Vocational University of Science and Technology under Grant No. 2026LG29 (Research on Improved Multimodal Semantic Fusion Doodle-Based Medical Image Segmentation).

References

[1] D. Qin, C. Leichner, M. Delakis, et al., "MobileNetV4: Universal models for the mobile ecosystem," in Computer Vision - ECCV 2024. Cham, Switzerland: Springer, 2024, pp. 78-96.

[2] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition, 2018, pp. 4510-4520.

[3] A. Howard, M. Sandler, B. Chen, et al., "Searching for MobileNetV3," in Proc. IEEE/CVF Int. Conf. Computer Vision, 2019, pp. 1314-1324.

[4] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition, 2018, pp. 6848-6856.

[5] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," in Computer Vision - ECCV 2018. Cham, Switzerland: Springer, 2018, pp. 122-138.

[6] W. Yu, P. Zhou, S. Yan, and X. Wang, "InceptionNeXt: When Inception meets ConvNeXt," in Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition, 2024, pp. 5672-5683.

[7] J. Hu, L. Shen, and G. Sun, "Squeeze-and-Excitation networks," in Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition, 2018, pp. 7132-7141.

[8] P. K. A. Vasu, J. Gabriel, J. Zhu, O. Tuzel, and A. Ranjan, "MobileOne: An improved one millisecond mobile backbone," in Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition, 2023, pp. 7907-7917.

[9] J. Chen, S.-H. Kao, H. He, et al., "Run, don't walk: Chasing higher FLOPS for faster neural networks," in Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition, 2023, pp. 12021-12031.

[10] S. Zagoruyko and N. Komodakis, "Wide residual networks," in Proc. British Machine Vision Conf., 2016, pp. 87.1-87.12.

[11] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," in Int. Conf. Learning Representations, 2017.

[12] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception architecture for computer vision," in Proc. IEEE Conf. Computer Vision and Pattern Recognition, 2016, pp. 2818-2826.

[13] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," in Int. Conf. Learning Representations, 2018.

[14] J. Ansel, E. Yang, H. He, et al., "PyTorch 2: Faster machine learning through dynamic Python bytecode transformation and graph compilation," in Proc. 29th ACM Int. Conf. Architectural Support for Programming Languages and Operating Systems, vol. 2, 2024, pp. 929-947.

