

Article

A Re-Parameterized Lightweight Residual Attention Framework for Resource-Constrained Edge Computing

Yuze Gao ^{1,†}, Jiamin Zhu ^{2,†} , Xiaoxiao Liu ^{1,*} and Wei Wu ^{1,*}¹ Hainan Institute, Zhejiang University, Sanya 572025, China; yuzegao@zju.edu.cn² College of Computer Science and Technology, Hainan University, Haikou 570228, China; zikarixue@gmail.com

* Correspondence: xxliu616@zju.edu.cn (X.L.); wuweixu@zju.edu.cn (W.W.)

† These authors contributed equally to this work.

Abstract

Edge vision systems require convolutional neural networks (CNNs) that preserve recognition accuracy under strict storage, computation, and latency constraints. Although ResNet18 is a compact residual backbone, direct deployment on resource-constrained devices remains costly, whereas simple channel reduction weakens representation capacity. This study aims to build a deployable ResNet18-based classifier that reduces model complexity while recovering the accuracy lost during compression. We propose a lightweight framework that combines global channel scaling, a re-parameterized attention residual block, and teacher–student knowledge distillation. The proposed block uses multi-branch convolution and squeeze-and-excitation attention during training, then folds the linear branches into a single 3-by-3 convolution for inference. Experiments on CIFAR-100 show that the final model reduces parameters from 11.220 M to 2.841 M, retains comparable Top-1 accuracy (0.7579 vs. 0.7606), improves Top-5 accuracy (0.9340 vs. 0.9253), and reduces graphics processing unit (GPU) batch inference latency from 3.279 ms to 2.161 ms. Deployment on PYNQ-Z2 verifies the complete camera-based CPU-side inference workflow, with an average end-to-end latency of 421.467 ms/frame. The results indicate that residual topology preservation, re-parameterized feature enhancement, and distillation form a practical route for edge-oriented lightweight CNN deployment.

Keywords: lightweight deep learning; residual network; re-parameterization; attention mechanism; knowledge distillation; edge deployment

1. Introduction

Convolutional neural networks (CNNs) have become a core component of visual perception systems. Their deployment target, however, is shifting from cloud servers and high-end graphics processing units (GPUs) to embedded terminals, mobile devices, and edge computing platforms. This shift changes the optimization objective. In server-side settings, accuracy is often the dominant metric; in edge settings, accuracy must be achieved under limited memory, restricted computing power, constrained energy consumption, and strict inference latency. Therefore, the central scientific problem is no longer whether a deep model can recognize images but whether a structurally compact and hardware-friendly model can preserve recognition performance after substantial compression.

ResNet18 is a natural starting point for this problem because it retains the optimization advantage of residual learning while being shallower than deeper residual variants [1].



Academic Editor: Paolo Bellavista

Received: 25 May 2026

Revised: 16 June 2026

Accepted: 17 June 2026

Published: 19 June 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Nevertheless, the standard ResNet18 still contains more than 11 million trainable parameters in the CIFAR-100 configuration used in this study. Direct channel scaling can reduce the parameter count and inference cost, but it also removes feature channels and weakens the representational capacity of residual blocks. A deployable lightweight design must therefore satisfy three coupled requirements: compress the backbone, compensate for the lost representation, and avoid introducing inference-time structures that are difficult to execute on edge devices.

Related Work and Research Gap

Existing studies have addressed lightweight vision models from three main directions. The first direction designs efficient CNN architectures. MobileNet reduces computation by decomposing standard convolution into depthwise and pointwise operations [2]; ShuffleNet reduces pointwise convolution cost through grouped convolution and channel shuffle [3]; GhostNet generates additional feature maps by cheap transformations [4]; and EfficientNet jointly scales depth, width, and resolution [5]. These methods demonstrate that compact operators can reduce computational cost. However, replacing the residual backbone with a new operator family may change the optimization behavior, implementation path, and hardware performance. As a result, such methods do not directly answer how to preserve the simple residual topology of ResNet18 while making it suitable for edge deployment.

The second direction compresses existing networks by pruning, decomposition, quantization, or distillation. Pruning removes redundant filters or channels according to importance measures [6–8]; low-rank decomposition approximates heavy layers by lower-dimensional factors [9,10]; quantization reduces numerical precision [11]; and knowledge distillation transfers class or feature information from a stronger teacher to a smaller student [12,13]. These techniques reduce different forms of redundancy, but each alone leaves a gap. Structural compression can damage accuracy, distillation cannot reduce inference cost without a compact student, and aggressive operator changes may not produce practical speedup on a target device. Thus, the bottleneck is the lack of an integrated design that links structural compression, representation recovery, and deployment-form simplification.

The third direction improves training-time representation while keeping inference simple. Structural re-parameterization trains a model with multi-branch structures and converts them into a single branch for inference. ACNet strengthens convolutional kernels with asymmetric branches [14], and RepVGG shows that training-time residual-like branches can be folded into VGG-style convolutions for efficient inference [15]. Channel attention, represented by squeeze and excitation (SE), adaptively recalibrates channel responses and improves the use of feature channels [16]. Yet, existing re-parameterization studies mainly focus on improving full-width networks or designing new backbones. For a compressed ResNet18, the key unresolved question is whether re-parameterization, attention, and distillation can jointly compensate for channel reduction while preserving a deployment-friendly graph. In this setting, the role of re-parameterization is not only to improve a full-width backbone, but also to recover the representation capacity lost after channel scaling without adding multi-branch inference overhead.

Motivated by this gap, this paper proposes a lightweight ResNet18 framework for resource-constrained edge deployment. The logic is demand-driven and problem-oriented: edge deployment requires compact inference; compact inference motivates channel-scaled residual backbones; channel scaling causes representation loss; representation loss motivates re-parameterized attention blocks and distillation; and deployment constraints require folding the training-time branches into a simple inference-time structure. The main contributions are as follows:

- A topology-preserving lightweight ResNet18 framework is constructed by applying global channel scaling to the residual backbone, producing a compact student network suitable for edge inference.
- A re-parameterized attention residual block is designed to enhance the compressed network during training and to convert its linear branches into a single 3-by-3 convolution during deployment.
- A full-width re-parameterized attention ResNet18 is used as the teacher to transfer soft class-distribution information to the half-width student, recovering accuracy without adding inference overhead.
- The complete pipeline is evaluated on CIFAR-100 and deployed on a PYNQ-Z2 platform with OV5640 camera input, validating both algorithmic effectiveness and embedded implementation feasibility.

Table 1 further summarizes how the proposed framework differs from the representative research directions discussed above. Existing lightweight architectures mainly reduce cost by replacing standard convolutions or redesigning the backbone. Compression methods mainly reduce redundancy but require additional mechanisms to recover accuracy. Re-parameterization methods improve the training-inference trade-off but are not always combined with channel-scaled residual deployment. In contrast, the proposed method preserves the ResNet18 topology, enhances the compressed blocks during training, distills teacher knowledge to the student, and verifies the final model on an embedded board. This positioning makes the contribution a deployment-oriented compression and recovery framework rather than a simple aggregation of existing modules.

Table 1. Position of the proposed framework relative to representative related methods.

Method Family	Typical Examples	Residual Topology Preserved	Re-Param.	Attention	Board-Level Validation
Efficient CNN	MobileNet, ShuffleNet, GhostNet, EfficientNet	No	No	Optional	No
Compression	Pruning, low-rank, distillation	Partly	No	Optional	No
Re-param. CNN	ACNet, RepVGG	Partly	Yes	No	No
Proposed framework	Channel-scaled ResNet18	Yes	Yes	SE	PYNQ-Z2

2. Materials and Methods

2.1. Problem Formulation and Framework Overview

The goal is to obtain a student model S from a ResNet18 backbone such that S has substantially fewer parameters and lower inference latency than the baseline model B , while its classification accuracy remains close to that of B . The design is constrained by deployment: the inference graph should be composed mainly of standard convolutional operations and lightweight channel attention, rather than training-time multi-branch structures.

Figure 1 shows the baseline residual topology. Because CIFAR-100 images are only 32-by-32 pixels, the baseline uses a 3-by-3 initial convolution with stride 1 and no initial max-pooling layer. This avoids the excessive early downsampling that would be introduced by the ImageNet-oriented 7-by-7 stride-2 convolution followed by max-pooling. The backbone then contains four residual stages and a final fully connected classifier. Each stage contains two basic residual blocks. Instead of changing the number of stages or the residual topology, the proposed method compresses the channel width and then recovers representation by block-level enhancement. Figure 2 summarizes the overall route: structural compression produces a lightweight backbone, the re-parameterized attention block improves the compressed representation, and knowledge distillation transfers class-relationship information from the teacher to the student.

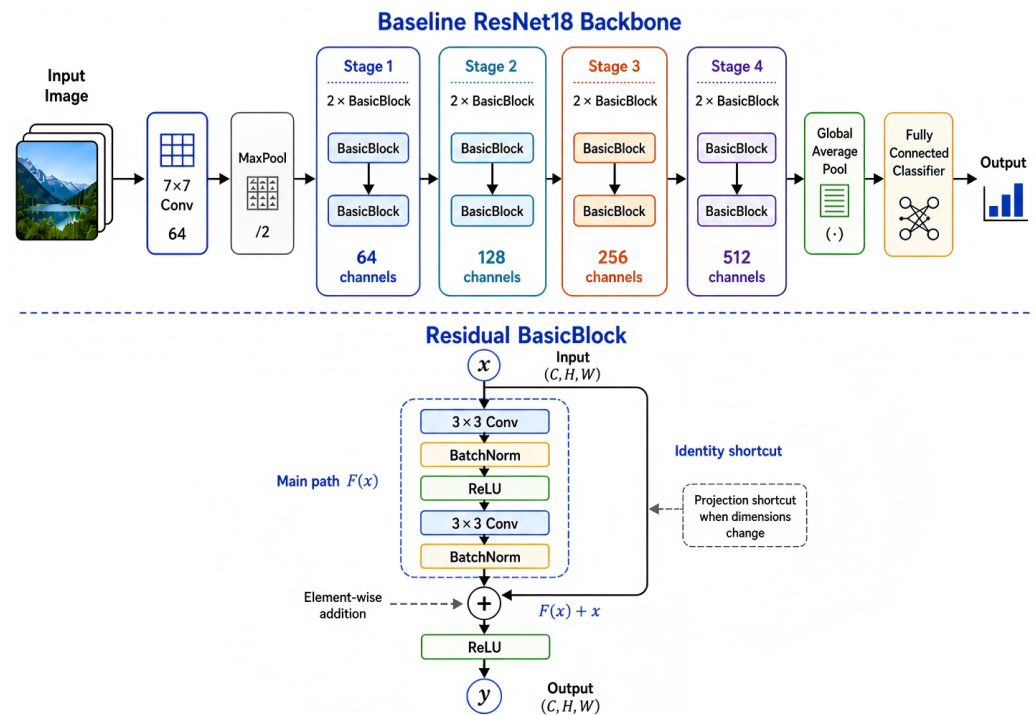


Figure 1. Baseline ResNet18 residual topology and residual basic block used as the starting point. The CIFAR-100 implementation uses a 3-by-3 initial convolution without an initial max-pooling layer.

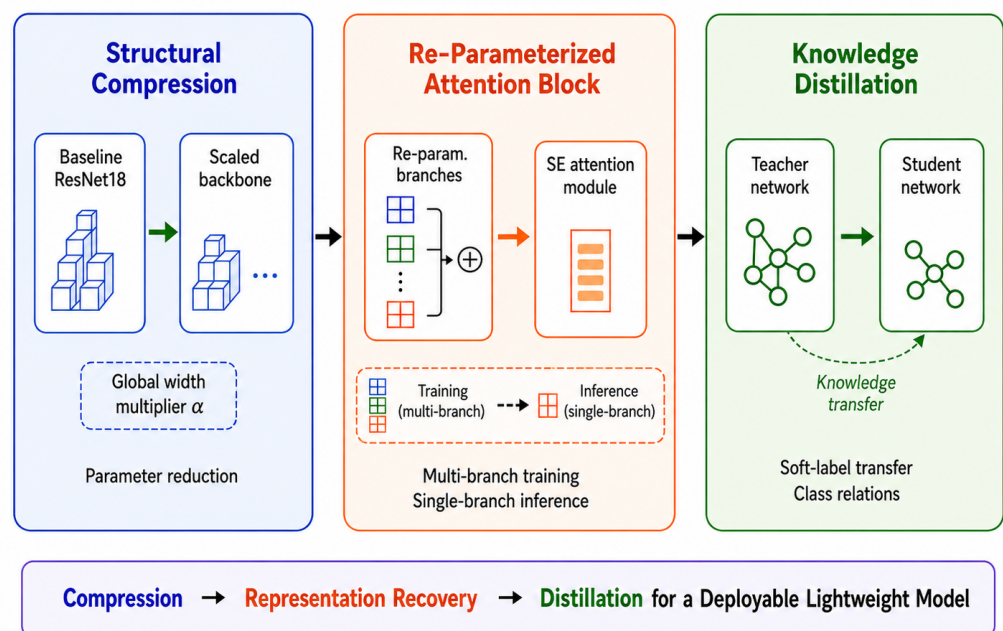


Figure 2. Overall design flow of the proposed lightweight framework.

2.2. Channel-Scaled Lightweight Backbone

Let C_l be the output channel number of the l -th convolutional layer in the original ResNet18. A global width multiplier α scales all convolutional channels as

$$C'_l = \max(1, \lfloor \alpha C_l \rfloor), \quad (1)$$

where C'_l is the compressed channel number. Kernel size, stride, residual connection form, and block number remain unchanged. Because the parameter count and floating-point operations of a standard convolution are approximately proportional to the product of

input and output channel numbers, the dominant convolutional complexity is reduced approximately according to α^2 .

This design is intentionally simple. It avoids irregular sparsity, preserves implementation regularity, and enables a controlled study of the accuracy–efficiency trade-off under different width multipliers. We evaluate $\alpha \in \{1.0, 0.7, 0.5, 0.3\}$ and select $\alpha = 0.5$ as the main student configuration because it provides strong compression while leaving enough capacity for subsequent recovery by re-parameterization and distillation.

2.3. Re-Parameterized Attention Residual Block

Channel scaling reduces the number of available feature channels, so the compressed block needs stronger training-time representation. The proposed residual block replaces a standard convolution in the residual branch with multiple parallel linear branches during training:

$$y = \sum_{i=1}^N \mathcal{F}_i(x), \quad (2)$$

where x is the input feature map and $\mathcal{F}_i(\cdot)$ denotes the i -th branch transformation. The implemented branches include a standard 3-by-3 convolution, a 1-by-1 convolution, an average-pooling-related branch, a cascaded 1-by-1 and k -by- k branch, and asymmetric 1-by-3 and 3-by-1 branches. These branches are complementary: the 3-by-3 branch captures local spatial patterns, the 1-by-1 branch improves channel mixing, the pooling-related branch introduces local statistical smoothing, the cascaded branch separates channel transformation from spatial modeling, and the asymmetric branches strengthen directional responses.

After linear branch fusion, an SE attention module recalibrates channel responses. Given an intermediate feature map $u \in \mathbb{R}^{C \times H \times W}$, global average pooling generates a channel descriptor $z \in \mathbb{R}^C$. The channel weights are computed by

$$s = \sigma(W_2 \delta(W_1 z)), \quad (3)$$

where W_1 and W_2 are the two fully connected layers, $\delta(\cdot)$ is the ReLU activation, and $\sigma(\cdot)$ is the sigmoid function. The output feature is

$$\hat{u}_c = s_c u_c. \quad (4)$$

Thus, the multi-branch structure expands the candidate feature transformations, while SE attention performs adaptive soft selection among compressed channels.

2.4. Deployment-Time Structural Folding

A multi-branch block is useful during training but inefficient during inference. Therefore, before deployment, all linear branches are folded into a single 3-by-3 convolution. In the implemented re-parameterized branches, convolution layers are configured without explicit bias before batch normalization. For the k -th output channel of a convolution followed by batch normalization, the equivalent kernel and bias are

$$\tilde{W}_k = \frac{\gamma_k}{\sqrt{\sigma_k^2 + \epsilon}} W_k, \quad \tilde{b}_k = \beta_k - \frac{\gamma_k \mu_k}{\sqrt{\sigma_k^2 + \epsilon}}, \quad (5)$$

where W_k is the convolution kernel, and γ_k , β_k , μ_k , σ_k^2 , and ϵ are the batch-normalization scale, shift, running mean, running variance, and numerical constant.

After Conv–BN fusion, all branches are mapped into a common 3-by-3 kernel space. The standard 3-by-3 branch is kept unchanged. The 1-by-1 branch is center-padded to 3-by-3. The 1-by-3 and 3-by-1 asymmetric branches are zero-padded along the missing

spatial dimension. The average-pooling-related branch is represented as a fixed local smoothing kernel with entries $1/9$ and then merged with its learnable convolution and batch normalization. For the cascaded 1-by-1 to 3-by-3 branch, the two fused convolutions are composed into a single equivalent 3-by-3 kernel, and the bias produced by the first stage is propagated through the second-stage kernel. The final deployment parameters are obtained by element-wise summation:

$$W_{\text{deploy}} = \sum_{i=1}^N \tilde{W}_i, \quad b_{\text{deploy}} = \sum_{i=1}^N \tilde{b}_i. \quad (6)$$

The nonlinear SE attention module is not folded into the convolution and remains in the inference graph. This folding makes the final student model closer to a conventional residual CNN and easier to execute on edge devices.

2.5. Knowledge Distillation

The teacher model $\mathcal{T}$ is the full-width enhanced ResNet18, and the student model $\mathcal{S}$ is the half-width lightweight network. The teacher is trained with the same optimizer, data preprocessing, and learning-rate schedule as the student models. The reported implementation follows the official CIFAR-100 split with 50,000 training images and 10,000 test images and does not introduce an additional validation subset. The main distilled student uses $\tau = 4$ and $\lambda = 0.5$, while the supplementary official-split reruns use the same temperature and $\lambda = 0.7$; these result groups are reported separately in the Results Section.

Given teacher logits z_t and student logits z_s , the softened distributions are

$$p_t = \text{softmax}(z_t/\tau), \quad p_s = \text{softmax}(z_s/\tau). \quad (7)$$

The distillation loss is the Kullback–Leibler divergence between the teacher and student softened distributions:

$$\mathcal{L}_{\text{KD}} = \text{KL}(p_t \| p_s). \quad (8)$$

The total student objective combines hard-label supervision and soft-label distillation:

$$\mathcal{L} = (1 - \lambda)\mathcal{L}_{\text{CE}}(z_s, y) + \lambda\tau^2\mathcal{L}_{\text{KD}}, \quad (9)$$

where $\mathcal{L}_{\text{CE}}$ is the cross-entropy loss, τ^2 compensates for the gradient-scale change caused by temperature scaling, and λ balances the two supervision terms. This strategy transfers class-similarity information to the compressed student without changing its deployment-time structure.

Unless otherwise stated, all CIFAR-100 models are trained for 200 epochs with a batch size of 128. Stochastic gradient descent is used with momentum 0.9, weight decay 5×10^{-4} , and an initial learning rate of 0.1. The random seed is fixed to 42, and deterministic computation settings are enabled to reduce run-to-run variation. The learning rate is linearly warmed up during the first epoch and then decayed by a factor of 0.1 at epochs 60 and 120.

2.6. Edge Deployment Design

After training, the student network is structurally folded and exported from PyTorch 2.7.1 [17] to Open Neural Network Exchange (ONNX) 1.20.1 format [18]. Figure 3 shows the deployment workflow from training to ONNX export, PYNQ-Z2 board-side loading, and OV5640 camera-based online inference. Figure 4 shows the image acquisition path. The programmable logic side receives the camera stream, converts it to an Advanced eXtensible Interface (AXI) video stream, performs Bayer-to-red-green-blue (RGB) conversion,

and writes frames into double data rate 3 (DDR3) memory through video direct memory access (VDMA). The processing system side reads frames, resizes and normalizes them to the CIFAR-100 input format, builds the input tensor, and calls ONNX Runtime 1.22.1 [19] for inference. This design separates the algorithm model from the camera pipeline, so the model can be replaced by updating the ONNX file.

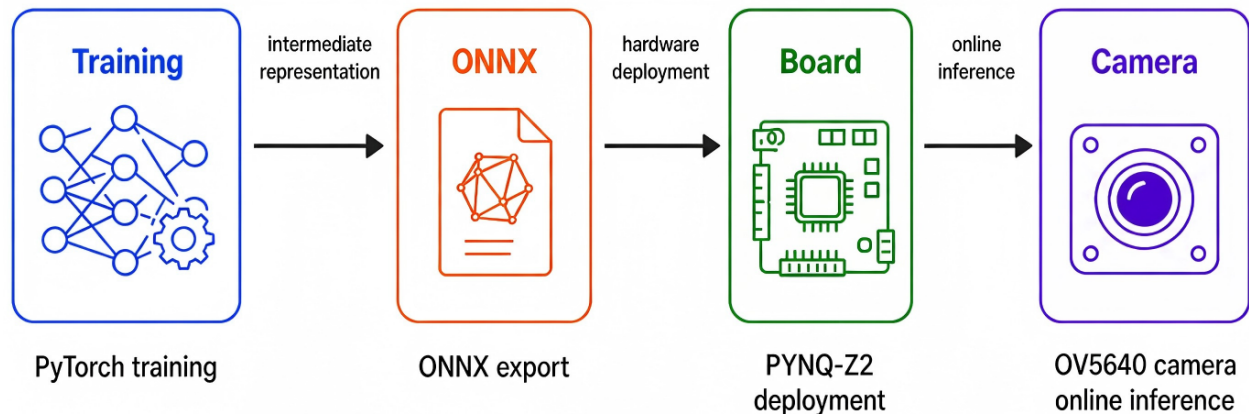


Figure 3. Deployment workflow on the PYNQ-Z2 platform.

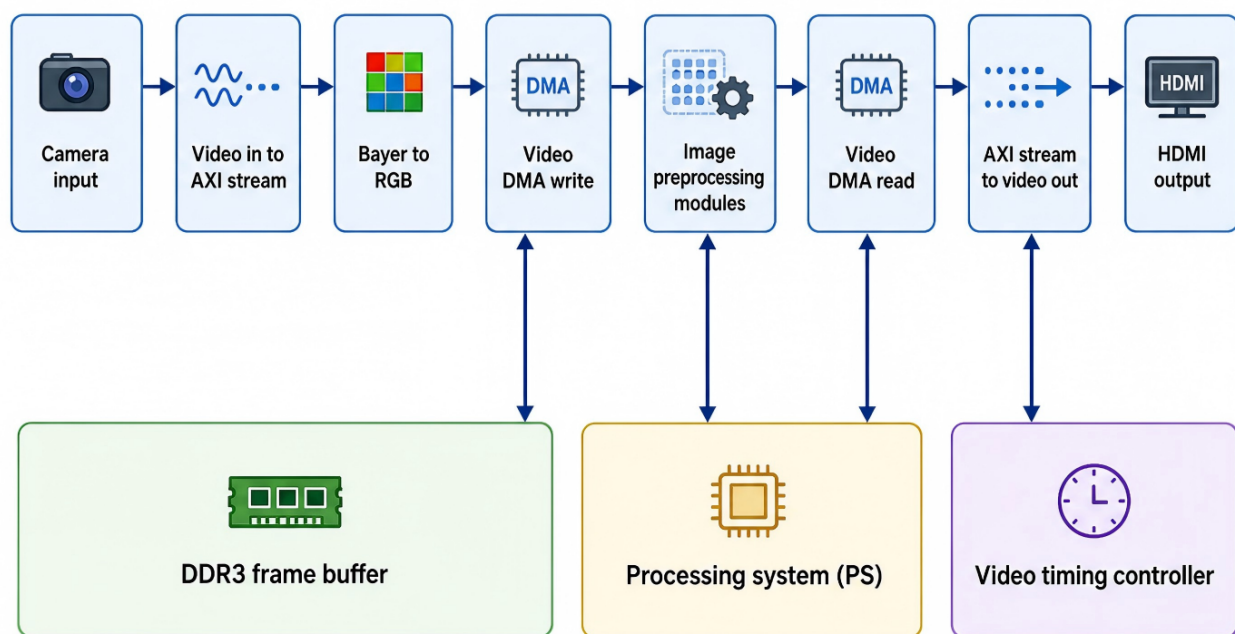


Figure 4. Video stream acquisition and processing data path on the PYNQ-Z2 platform.

Table 2 lists the experimental and deployment environments used in this study. The training and evaluation experiments are performed on an NVIDIA RTX 4090 workstation using PyTorch 2.7.1 [17]. The edge platform consists of a PYNQ-Z2 board and an OV5640 camera module. The PYNQ-Z2 board is based on the Zynq-7020 system-on-chip, which integrates a dual-core ARM Cortex-A9 processing system and programmable logic resources. The board provides 512 MB DDR3 memory, Gigabit Ethernet, PMOD expansion interfaces, a Python/Jupyter development environment, and a default Ethernet address of 192.168.2.99 in the used PYNQ image. The current deployment uses ONNX Runtime 1.22.1 on the central processing unit (CPU) of the processing system and does not include a custom field-programmable gate array (FPGA) acceleration core; this choice isolates the effect of model lightweighting and keeps the deployment pipeline simple.

Table 2. Hardware and software environment used in this study.

Stage	Item	Reported Configuration
Training/evaluation	Graphics processing unit (GPU) workstation	NVIDIA RTX 4090
Training/evaluation	Framework	PyTorch 2.7.1, Torchvision 0.22.1
Training/evaluation	Dataset	CIFAR-100 official split with 50,000 training and 10,000 test images; no validation subset
Edge deployment	Development board	PYNQ-Z2 based on Zynq-7020
Edge deployment	Processing system	Dual-core ARM Cortex-A9
Edge deployment	Memory and interfaces	512 MB DDR3, Gigabit Ethernet, PMOD interfaces
Edge deployment	Camera	OV5640 CMOS camera module
Edge deployment	Runtime environment	PYNQ Linux, Python/Jupyter Notebook, ONNX 1.20.1, ONNX Runtime 1.22.1
Edge deployment	Inference mode	Central processing unit (CPU) inference on the processing system; no custom FPGA accelerator
Edge deployment	Network access	Default Ethernet address 192.168.2.99

2.7. Board-Side Inference Procedure

The board-side procedure follows four steps. First, the trained model is converted to the folded deployment form and exported as an ONNX file with a 32-by-32 RGB input shape. Second, the ONNX file is uploaded through the PYNQ Jupyter Notebook environment to a model directory on the board-side Linux file system. Third, ONNX Runtime loads the model, parses the computation graph, and initializes input and output tensors. Fourth, camera frames are read from the DDR frame buffer, resized to 32-by-32 pixels, normalized with the same preprocessing used during training, rearranged into channel-first tensor order, and passed to the inference session. The output confidence vector is mapped to the CIFAR-100 label list, and the predicted class and confidence score are displayed in the Jupyter interface or optionally overlaid on the camera frame for HDMI display.

During board-side verification, the online inference program reads a camera frame, constructs the 32-by-32 model input, performs ONNX Runtime inference, and reports the predicted class ID, class name, and confidence score. This verifies that the proposed model is not only evaluated offline but also integrated into a complete board-side workflow covering image acquisition, preprocessing, inference, and visualization.

3. Results

3.1. Experimental Settings

The CIFAR-100 dataset [20] is used as the benchmark. It contains 60,000 RGB images from 100 classes. In this study, the experiments use the official split with 50,000 training images and 10,000 test images. The image resolution is 32-by-32 pixels. Training uses random crop with padding, random horizontal flipping, small-angle random rotation, normalization, and mini-batch loading. Test evaluation uses deterministic normalization only. In addition to the main result tables, the revision package includes loss and accuracy diagnostic curves to illustrate convergence behavior qualitatively. These traces are presented only as descriptive training/evaluation diagnostics and are not used for validation-based checkpoint selection or statistical claims.

The models are implemented in PyTorch 2.7.1 and trained on a workstation equipped with an NVIDIA RTX 4090 GPU. The optimizer is stochastic gradient descent with momentum. The momentum is 0.9, weight decay is 0.0005, the default initial learning rate is 0.1, the batch size is 128, and the training schedule lasts 200 epochs. The main experiments use a fixed random seed of 42. To evaluate statistical stability and ablation breadth, supplementary experiments are additionally conducted under the same official-split pipeline. Five-seed summaries are reported for the baseline ResNet18 and Ours, whereas the width-plus-distillation and branch-level checks are targeted supplementary ablations and, unless otherwise stated, are interpreted descriptively. GPU batch inference latency is measured

under 32-bit floating point (FP32) inference with a batch size of 128, disabled gradient computation, and no mixed precision or additional graph optimization. The evaluation metrics include Top-1 accuracy, Top-5 accuracy, parameter count, batch inference latency, parameter compression ratio, and inference speedup.

The experimental section is organized to evaluate the method, including model training, model complexity, comparative accuracy, component contribution, structural redundancy, and embedded deployment. Top-1 accuracy measures whether the highest-confidence class matches the ground truth, whereas Top-5 accuracy measures whether the ground truth class appears among the five highest-confidence predictions. Parameter count is used to represent storage complexity. GPU batch inference latency is used to measure the average batch forward time under the unified workstation setting. The PYNQ-Z2 board-side profiling uses a batch size of 1 and reports end-to-end latency, throughput, CPU utilization, recorded peak memory, and board-reported temperature over 100 CIFAR-100 test samples after 10 warm-up runs. Supplementary Materials provides the implementation scripts, model definitions, and ONNX/PYNQ-Z2 deployment utilities used in this study.

3.2. Training Behavior

Figure 5 shows training and evaluation loss diagnostics of the lightweight ResNet18. The left panel uses the logged training cross-entropy record, converted from iteration steps to epochs and smoothed with a one-epoch moving average. The right panel uses the evaluation loss record and rescales it back to the batch-mean cross-entropy scale. The training loss drops rapidly at the beginning of optimization, indicating that the model quickly learns the basic discriminative patterns of CIFAR-100. When scheduled learning-rate drops occur, both curves show clear step-like decreases and then converge to stable low levels. These curves are included as qualitative diagnostics of optimization behavior and are not used as validation-based model-selection evidence.

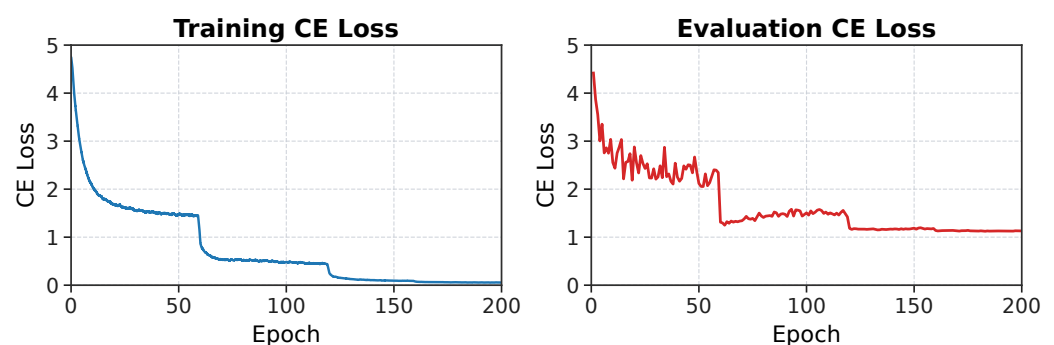


Figure 5. Training and evaluation loss diagnostics of the lightweight ResNet18.

Figure 6 reports the accuracy diagnostic curve. The accuracy increases quickly during the early epochs, then enters a slower improvement stage. Around learning-rate decay points, the curve exhibits step-like improvements, suggesting that the lower learning rate helps refine the parameters after the model has learned coarse discriminative features. In the middle and late training stages, the accuracy remains in a relatively stable high range without sharp collapse. This curve is included to visualize optimization behavior and is not used as a validation-based model-selection rule.

We further visualize the evolution of the last fully connected layer. As shown in Figure 7, the weight distribution gradually becomes more concentrated and symmetric during training, while the bias distribution remains in a narrow numerical range around zero. This parameter-level observation is consistent with the loss and accuracy curves: the model does not exhibit abnormal divergence, and the final classifier parameters remain numerically stable.

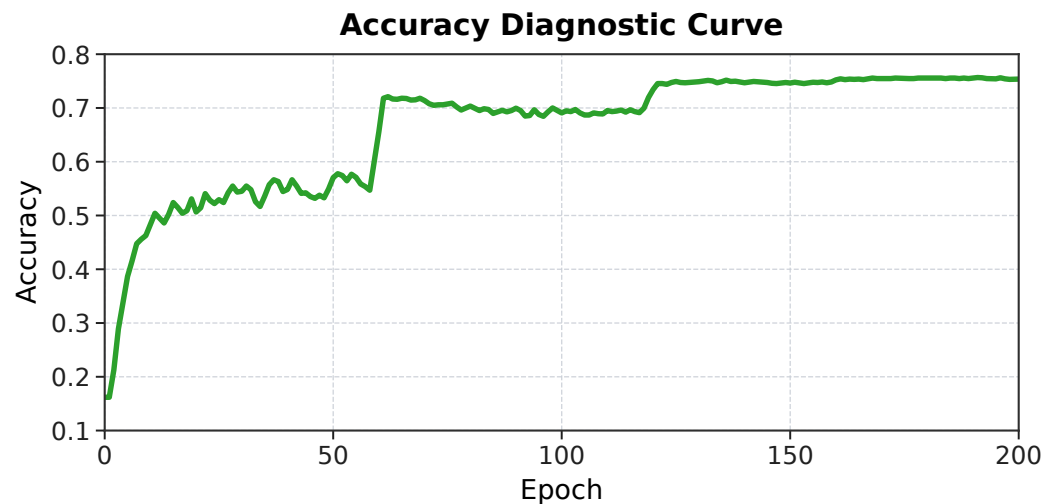


Figure 6. Accuracy diagnostic curve of the lightweight ResNet18.

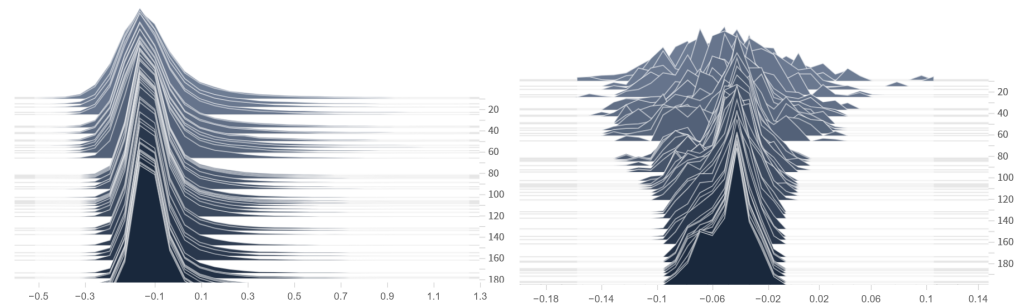


Figure 7. Evolution of the fully connected classifier parameters during training.

3.3. Effect of Channel Scaling

Table 3 reports the effect of width scaling. The results confirm the expected compression trend: smaller width multipliers reduce parameters and latency. However, they also reduce Top-1 accuracy. The half-width model reduces parameters from 11.220 M to 2.821 M and reduces latency from 3.279 ms/batch to 1.754 ms/batch, but its Top-1 accuracy drops from 0.7606 to 0.7299. The 0.3-width model is smaller but suffers a larger accuracy loss and does not further reduce the measured latency because very small models may suffer from non-computational overhead on the GPU runtime. Therefore, the 0.5-width model is selected as the student because it provides a favorable starting point for subsequent accuracy recovery. Figure 8 further visualizes the trade-off among parameter count, latency, and Top-1 accuracy.

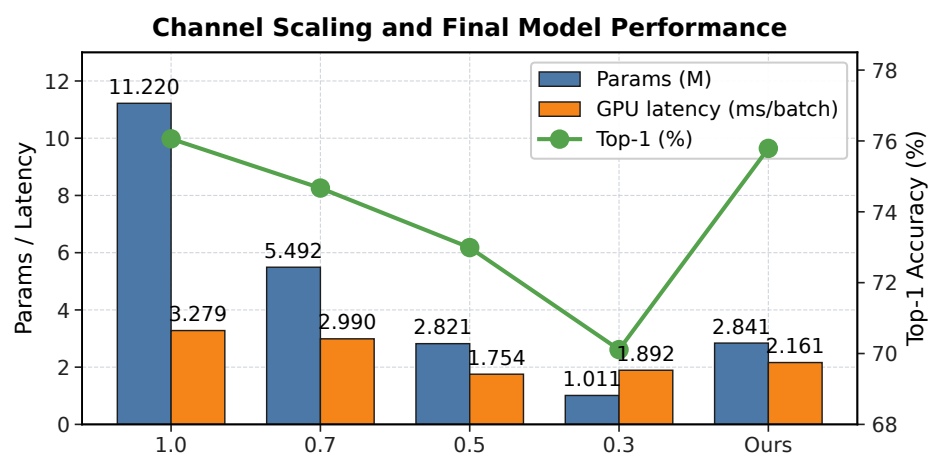


Figure 8. Channel-scaling results and the final model performance.

Table 3. Effect of channel scaling on ResNet18 using CIFAR-100. Latency is measured in ms/batch on an RTX 4090 graphics processing unit (GPU) with batch size 128.

Width	Top-1	Top-5	Params (M)	Lat. (ms)	Comp.	Speedup
1.0	0.7606	0.9253	11.220	3.279	1.000	1.000
0.7	0.7467	0.9201	5.492	2.990	0.489	1.097
0.5	0.7299	0.9161	2.821	1.754	0.251	1.869
0.3	0.7011	0.9176	1.011	1.892	0.090	1.733

Table 4 reports supplementary official-split experiments that combine channel scaling with distillation. Across all three tested compressed students, distillation provides a consistent Top-1 accuracy recovery trend. The Top-1 gain is 0.0145 for width 0.7, 0.0171 for width 0.5, and 0.0196 for width 0.3. These results support the interpretation that distillation remains beneficial across a range of compression levels.

Table 4. Supplementary experiments for width scaling combined with knowledge distillation under the official CIFAR-100 split.

Width	No KD Top-1	KD Top-1	Gain	No KD Top-5	KD Top-5	Params (M)
0.7	0.7467	0.7612	+0.0145	0.9201	0.9357	5.492
0.5	0.7299	0.7470	+0.0171	0.9161	0.9335	2.821
0.3	0.7011	0.7207	+0.0196	0.9176	0.9171	1.011

3.4. Overall Comparison with the Baseline

Table 5 compares the final model with the original ResNet18. The proposed model contains 2.841 M parameters, which is 25.3% of the baseline, i.e., a parameter reduction of approximately 74.7%. Its Top-1 accuracy remains close to the baseline (0.7579 vs. 0.7606), while Top-5 accuracy increases from 0.9253 to 0.9340. The GPU batch inference latency decreases from 3.279 ms to 2.161 ms, giving a 1.518-fold measured speedup. Figure 9 visualizes this accuracy–efficiency trade-off. These results indicate that the proposed framework does not merely shrink the network; it recovers most of the lost discriminative ability through re-parameterized attention and distillation.

Table 5. Comparison between the original ResNet18 and the proposed lightweight model.

Model	Top-1	Top-5	Params (M)	Param. Ratio	Lat. (ms)	Speedup
ResNet18	0.7606	0.9253	11.220	1.000	3.279	1.000
Ours	0.7579	0.9340	2.841	0.253	2.161	1.518

Table 6 adds a five-seed summary under the official CIFAR-100 split. The baseline ResNet18 reaches a 0.7598 ± 0.0028 Top-1 and 0.9280 ± 0.0012 Top-5 accuracy. The corresponding Ours summary reaches 0.7528 ± 0.0047 Top-1 and 0.9335 ± 0.0020 Top-5 with 2.841 M parameters. The Top-1 standard-deviation ranges overlap, so these supplementary results are interpreted as comparable Top-1 performance rather than a Top-1 advantage. The main advantage of the proposed model is the substantial parameter reduction together with the higher mean Top-5 accuracy and the reduced inference cost.

Table 6. Supplementary five-seed run-summary table under the official CIFAR-100 split.

Model	Runs	Top-1	Top-5	Params (M)	Protocol Note
ResNet18	5	0.7598 ± 0.0028	0.9280 ± 0.0012	11.220	Five runs
Ours	5	0.7528 ± 0.0047	0.9335 ± 0.0020	2.841	Five runs

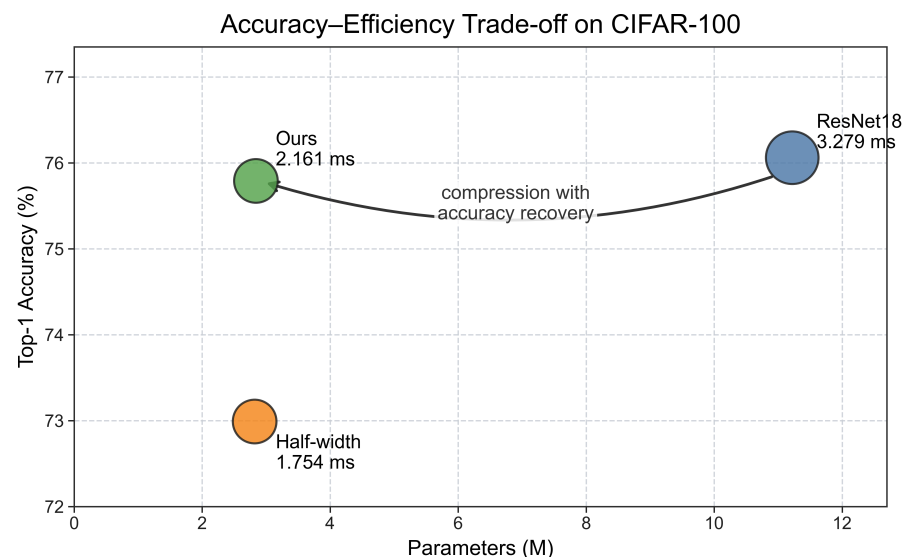


Figure 9. Accuracy–efficiency trade-off among the original ResNet18, the direct half-width model, and the final proposed model.

Table 7 summarizes the complexity and deployability indicators used in the system-level analysis. Parameter count, parameter compression ratio, GPU batch inference latency, inference speedup, operation count, model size, and ONNX size are organized together to evaluate both algorithmic efficiency and deployment cost, and Figure 10 provides the corresponding visual comparison. Multiply–accumulate operations (MACs) are obtained by layer-wise counting for the CIFAR-100 ResNet18 configuration, whose parameter count matches the reported experimental values. For the final model, the deployment-time graph is counted after structural folding; therefore, training-time re-parameterization branches are not included, while the lightweight SE overhead is included. The FP32 parameter payload is computed as

$$S_{\text{FP32}} = N_{\text{param}} \times 4 \text{ bytes}, \quad (10)$$

where N_{param} is the number of parameters. The ONNX size is reported according to the serialized FP32 parameter payload used by the deployment model.

Table 7. Model complexity and deployability summary. The table combines accuracy-related efficiency metrics with deployment-oriented storage and operation-count indicators; MACs denotes multiply–accumulate operations, and FP32 denotes 32-bit floating point.

Model	Params (M)	Param. Ratio	MACs (M)	FP32 Size (MB)	ONNX Size (MB)	GPU Lat. (ms/batch)	Speedup
ResNet18	11.220	1.000	555.47	44.88	44.9	3.279	1.000
Half-width	2.821	0.251	139.32	11.28	11.3	1.754	1.869
Ours	2.841	0.253	139.59	11.36	11.4	2.161	1.518

Figure 11 visualizes the supplementary summary statistics with standard-deviation error bars. The plot highlights the intended interpretation of the added evidence: Ours remains comparable to the full ResNet18 baseline in Top-1 accuracy, with overlapping Top-1 error bars, while showing a higher mean Top-5 accuracy across the five-seed reruns. These values are used as supplementary statistical context for the accuracy-recovery behavior of the proposed lightweight framework.

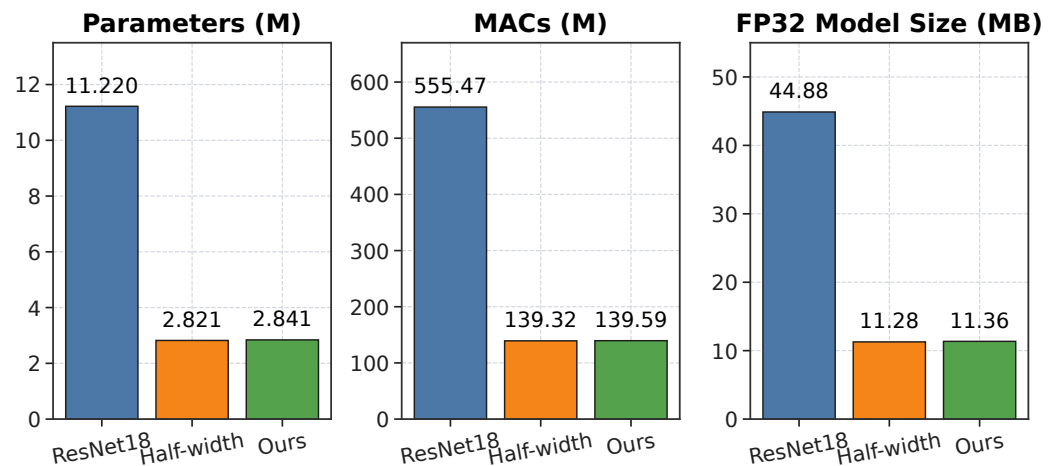


Figure 10. Model complexity indicators of the original ResNet18, direct half-width ResNet18, and the proposed model.

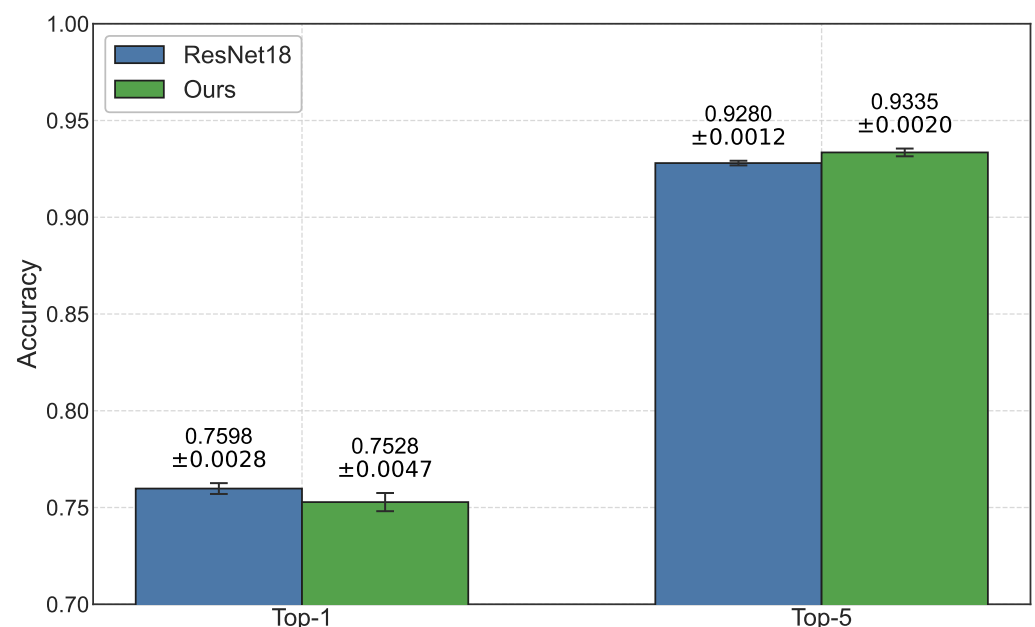


Figure 11. Supplementary five-seed run-summary under the official CIFAR-100 split. Error bars denote standard deviation.

3.5. Comparison with Representative Lightweight CNNs

Table 8 provides a descriptive single-seed comparison with representative lightweight CNNs under the same CIFAR-100 evaluation setting. SqueezeNet [21] and ShuffleNet [3] use fewer parameters, but their accuracy is lower in this reference run. MobileNet [2], EfficientNet-B0 [5], and GhostNet [4] have comparable or larger parameter counts, and they also obtain lower accuracy in this setting. These comparison models are trained with the same data preprocessing, 200-epoch schedule, batch size, and single random seed used for the ResNet18 experiments; no architecture-specific hyperparameter search or multi-seed averaging is included. Therefore, this table is used as supporting context rather than as statistical evidence across random seeds. Within this reference setting, the proposed model attains a favorable accuracy–efficiency balance among the listed architectures. This observation is consistent with the hypothesis that, for this task, preserving the residual topology and enhancing it internally can be useful compared with directly replacing it with another lightweight operator family.

Table 8. Descriptive single-seed comparison with representative lightweight convolutional neural networks (CNNs) on CIFAR-100.

Model	Top-1	Top-5	Params (M)	Latency (ms/batch)
SqueezeNet	0.6944	0.9119	0.781	2.847
MobileNet	0.6755	0.8913	3.315	3.213
ShuffleNet	0.7071	0.9232	1.012	6.334
EfficientNet-B0	0.5896	0.8609	4.136	6.376
GhostNet	0.5802	0.8514	4.030	7.907
Ours	0.7579	0.9340	2.841	2.161

3.6. Ablation Analysis

Table 9 analyzes the contribution of each component. Channel scaling alone causes the main accuracy loss. Adding the multi-branch re-parameterized block improves accuracy, and the folded version remains close in accuracy while restoring a compact parameter count. In this ablation setting, the SE variant performs better than efficient channel attention (ECA) [22]. Finally, distillation raises the half-width SE model from a 0.7381 to 0.7579 Top-1 accuracy and from a 0.9222 to 0.9340 Top-5 accuracy. The ablation sequence therefore supports the proposed causal chain: compression creates efficiency, re-parameterized attention restores representation, and distillation transfers additional class-relationship information.

Table 9. Ablation study of the proposed components. “Multi” denotes the training-time multi-branch form, and “Folded” denotes the deployment form after re-parameterization.

Width	Attention	Distill.	Mode	Top-1	Params (M)
1.0	None	No	Standard	0.7606	11.220
1.0	None	No	Multi	0.7734	25.819
1.0	None	No	Folded	0.7730	11.216
0.5	None	No	Standard	0.7299	2.821
0.5	None	No	Folded	0.7349	2.819
0.5	ECA	No	Folded	0.7290	2.819
0.5	SE	No	Folded	0.7381	2.841
0.5	SE	Yes	Folded	0.7579	2.841

Table 10 reports a focused supplementary comparison under the half-width setting. The direct half-width baseline and the pure 3-by-3 branch behave almost identically, whereas adding the 1-by-1 branch yields a modest Top-1 improvement with nearly unchanged parameter count. These results make the branch contribution more explicit.

Table 10. Supplementary branch-level experiments under the half-width setting.

Variant	Top-1	Top-5	Params (M)	Interpretation
Half-width baseline	0.7237	0.9146	2.821	Direct width scaling
3-by-3 only	0.7235	0.9145	2.819	Nearly unchanged from baseline
3-by-3 + 1-by-1	0.7293	0.9184	2.819	Modest positive branch gain

3.7. Redundancy and Feature Diversity Analysis

Beyond accuracy and latency, this study analyzes parameter redundancy using the effective rank of convolution kernels and the correlation between kernels. The effective-rank statistic is used to describe how many dominant dimensions are needed to represent most of the kernel information. For a convolutional kernel tensor, we reshape the weights

into a matrix $W \in \mathbb{R}^{C_{out} \times (C_{in}k^2)}$ and compute its singular values $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_R$, where $R = \min(C_{out}, C_{in}k^2)$. The 95% effective rank is defined as

$$r_{95} = \min \left\{ r : \frac{\sum_{i=1}^r \sigma_i^2}{\sum_{i=1}^R \sigma_i^2} \geq 0.95 \right\}. \quad (11)$$

The normalized ratio r_{95}/C_{out} is used to compare layers with different numbers of output channels. Table 11 reports the r_{95} values of the final convolutional layers in the four residual stages. For the original ResNet18, the fourth stage has a low r_{95}/C_{out} ratio of 0.1816, suggesting that a large number of output channels exist but only a smaller effective subspace is dominant. Direct channel compression increases the relative effective-rank ratio in several stages because fewer channels remain. The proposed lightweight model further changes this distribution: its fourth-stage ratio rises to 0.5312, indicating that the compressed high-level channels are used more effectively after re-parameterized attention and distillation.

Table 11. Effective-rank analysis of convolution kernels in different residual stages.

Model	Stage	C_{out}	r_{95}	r_{95}/C_{out}
ResNet18	1	64	35	0.5469
ResNet18	2	128	86	0.6719
ResNet18	3	256	176	0.6875
ResNet18	4	512	93	0.1816
Channel-scaled ResNet18	1	32	21	0.6562
Channel-scaled ResNet18	2	64	49	0.7656
Channel-scaled ResNet18	3	128	97	0.7578
Channel-scaled ResNet18	4	256	86	0.3359
Ours	1	32	13	0.4062
Ours	2	64	43	0.6719
Ours	3	128	94	0.7344
Ours	4	256	136	0.5312

Figure 12 shows the kernel-correlation heatmaps of the original ResNet18, the channel-scaled ResNet18, and the final lightweight model. The diagonal entries are high because they represent self-correlation. The off-diagonal regions are mostly close to zero and do not form large high-correlation blocks, indicating that the kernels do not collapse into strongly redundant clusters. Compared with direct channel compression, the proposed lightweight model maintains a relatively dispersed off-diagonal texture. This observation supports the interpretation that the re-parameterized attention block and distillation do not simply recover accuracy by increasing parameters; rather, they help the compressed model maintain useful feature diversity.

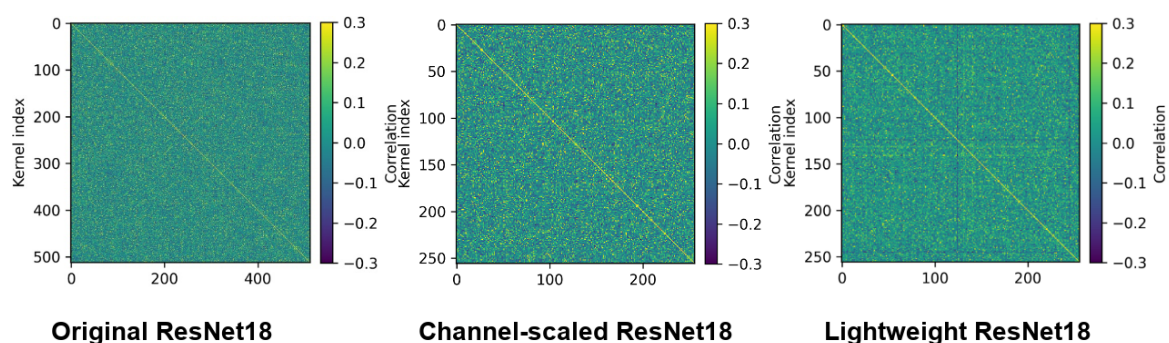


Figure 12. Kernel-correlation heatmaps of the original ResNet18, channel-scaled ResNet18, and the final lightweight model.

3.8. PYNQ-Z2 Deployment

The folded final student model and the comparison ONNX models are deployed on PYNQ-Z2 to verify practical feasibility under the same CPU-side runtime path. Official CIFAR-100 test images are used as fixed input samples during board-side measurement. The input size is fixed at 32-by-32 pixels, and the same normalization as training is applied. Before formal timing, 10 warm-up runs are executed, and runtime statistics are recorded with batch size 1. Table 12 reports the board-side results. The final folded model reaches 421.467 ms/frame mean latency, 420.112 ms/frame P50 latency, 423.844 ms/frame P95 latency, and approximately 2.37 FPS, confirming the functional deployment path. This rate is suitable for low-rate edge scenarios such as periodic inspection, event-triggered monitoring, and low-power visual sensing, but it is not intended for real-time continuous video analysis. Relative to ResNet18, this corresponds to a 55.2% reduction in end-to-end latency and about a 2.23-fold throughput improvement under the same CPU-side ONNX Runtime path. The width-scaled KD models further show the embedded scaling trend: 0.7 KD reaches 455.615 ms/frame, 0.5 KD reaches 245.170 ms/frame, and 0.3 KD reaches 93.725 ms/frame, whereas the ResNet18 baseline requires 940.234 ms/frame. The final model also reports 100.000% mean CPU utilization, 80.047 MB recorded peak memory, and 56.983/57.829 °C mean/maximum board-reported temperature. CPU utilization is close to saturation for all profiled rows, confirming that CPU-side ONNX Runtime inference is the main engineering bottleneck and that future acceleration should focus on inference operators. Power consumption is not directly reported because no calibrated board-level power meter or reliable board-side power monitor was available. Figure 13 visualizes the latency-resource comparison.

Table 12. PYNQ-Z2 board-side CPU-side ONNX Runtime results. Latency values are in ms/frame; P50 and P95 denote the 50th and 95th percentiles. CPU utilization, recorded peak memory, and board-reported temperature are included to characterize the embedded runtime cost.

Model	Top-1	Top-5	Mean Lat.	P50	P95	FPS	CPU Mean (%)	Peak Mem. (MB)	Temp. Mean/Max (°C)
ResNet18	0.7606	0.9253	940.234	925.283	964.933	1.064	99.993	110.977	55.372/57.460
0.7 KD	0.7612	0.9357	455.615	454.446	456.101	2.195	99.989	90.141	57.077/58.198
0.5 KD	0.7470	0.9335	245.170	244.559	245.286	4.079	100.000	79.066	56.890/57.706
0.3 KD	0.7207	0.9171	93.725	93.162	93.514	10.670	100.000	71.480	55.421/55.984
Ours	0.7579	0.9340	421.467	420.112	423.844	2.370	100.000	80.047	56.983/57.829

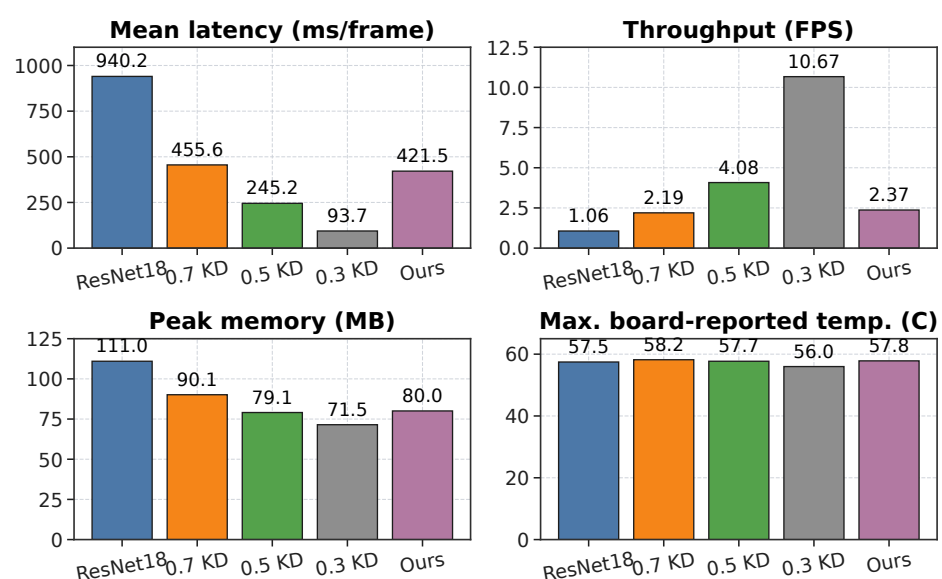


Figure 13. PYNQ-Z2 board-side profiling of latency, throughput, recorded peak memory, and board-reported temperature under CPU-side ONNX Runtime.

4. Discussion

The results reveal a clear division of labor among the proposed components. Channel scaling provides the main compression effect, but it also causes the primary accuracy degradation. The re-parameterized attention block addresses this degradation by increasing training-time feature diversity and selecting useful channels through SE attention. Structural folding then prevents the training-time complexity from becoming inference-time complexity. Distillation further improves the compact model by exposing class-similarity information that hard labels cannot provide. Therefore, the method works not because of a single module, but because the modules are arranged according to the compression–recovery–deployment chain.

Compared with lightweight architectures that replace standard residual blocks with specialized operators, the proposed framework emphasizes residual topology preservation. This has two implications. First, it offers a direct upgrade path for applications that already use ResNet18 as a baseline. Second, it keeps the deployment graph regular after re-parameterization, which is important for edge platforms where irregular sparsity or complex data movement may not translate into a real speedup. The descriptive comparison with representative lightweight CNNs is consistent with this interpretation under the current CIFAR-100 setting.

From a system perspective, the work is more than an offline model-compression experiment. The model is trained and folded on the workstation, exported to ONNX, transferred to the PYNQ-Z2 file system, loaded through ONNX Runtime, and connected to an OV5640 camera pipeline. The video stream is buffered through DDR3 memory, and the processing system performs preprocessing, tensor construction, inference, label mapping, and result visualization. This end-to-end route verifies that the proposed lightweight model can be integrated into a real embedded workflow rather than only evaluated on a desktop GPU.

The deployment experiment also clarifies the next engineering target. The PYNQ-Z2 implementation demonstrates that the folded model can run in a complete camera-driven workflow, and the board-side measurements quantify the CPU-side cost of the same ONNX Runtime deployment path. The final model reduces end-to-end latency from 940.234 ms/frame to 421.467 ms/frame relative to ResNet18 and increases throughput from 1.064 FPS to approximately 2.37 FPS, with P50 and P95 latencies of 420.112 and 423.844 ms/frame. The width-scaled KD comparison rows further show the embedded scaling trend down to 245.170 ms/frame at width 0.5 KD and 93.725 ms/frame at width 0.3 KD. The final model also reports 100.000% mean CPU utilization, 80.047 MB recorded peak memory, and 56.983/57.829 °C mean/maximum board-reported temperature. Therefore, the current system should be interpreted as a portable CPU-side embedded implementation that verifies deployment feasibility and model-lightweighting benefit, rather than a fully optimized real-time FPGA accelerator. In practical terms, the measured rate is appropriate for low-frequency recognition tasks, but applications requiring continuous real-time video analysis would need further acceleration. The measured CPU utilization also shows that inference on the processing system remains the dominant bottleneck, so future hardware-aware work should focus on quantization, operator fusion, and programmable-logic acceleration. Power consumption is not directly claimed because no calibrated board-level power measurement was available. In addition, the added official-split reruns provide supplementary statistical context for the baseline and Ours together with targeted width and branch checks. Accordingly, the supplementary summary tables support interpretation, while the width and branch reruns are used as targeted component evidence.

5. Conclusions

This paper studied the problem of deploying ResNet18 on resource-constrained edge devices and proposed a lightweight solution that combines channel scaling, re-parameterized attention blocks, and knowledge distillation. By using global channel scaling, the method reduces the structural complexity of the residual backbone; by using multi-branch training and SE attention, it compensates for the representation loss caused by compression; and by folding the linear branches into a single 3-by-3 convolution, it keeps the inference graph deployment-friendly.

The results show that the proposed model reduces the parameter count from 11.220 M to 2.841 M while maintaining comparable Top-1 accuracy and improving Top-5 accuracy on CIFAR-100. The FP32 parameter payload is reduced from 44.88 MB to 11.36 MB, and the CIFAR-100 computation after structural folding is 139.59 M MACs. The GPU batch inference latency is reduced from 3.279 ms to 2.161 ms, corresponding to a 1.518-fold speedup. Supplementary five-seed results further indicate that the Top-1 performance should be interpreted as comparable rather than superior, while the main practical gains are parameter reduction, higher mean Top-5 accuracy, and reduced inference cost. Additional supplementary experiments show that knowledge distillation consistently improves width-compressed students and that a simple 3-by-3 plus 1-by-1 branch combination can provide a modest positive branch contribution in the half-width setting. In addition, the folded student model is exported to ONNX and deployed on a PYNQ-Z2 platform, where it validates a camera-driven workflow including frame acquisition, preprocessing, CPU inference through ONNX Runtime, label mapping, and visualization. The final model reports a 421.467 ms/frame mean latency, 420.112 ms/frame P50 latency, 423.844 ms/frame P95 latency, approximately 2.37 FPS, 100.000% mean CPU utilization, 80.047 MB recorded peak memory, and 56.983/57.829 °C mean/maximum board-reported temperature. This deployment rate is suitable for low-rate edge use cases but not for real-time continuous video analysis. The comparison profiling further shows that narrower KD variants can reduce CPU-side latency along the same embedded runtime path. These findings provide a practical method for lightweight residual-network design and offer a reproducible CPU-side embedded verification path for future hardware-aware acceleration.

The present study also points to several extensions. The channel scaling strategy is uniform across layers and can be further improved by layer-wise sensitivity analysis. The board-side deployment uses ONNX Runtime on the processing system, so the measured latency does not represent the maximum acceleration potential of the FPGA fabric. Moreover, the supplementary multi-seed evidence is limited to the baseline and proposed model, and power consumption is not directly reported because no calibrated board-level power measurement was available. Future work will focus on non-uniform channel allocation, redundancy-aware structure search, broader multi-seed statistical evaluation, quantized inference, programmable-logic acceleration, and more complete edge-device resource monitoring. The method will also be evaluated in more realistic application scenarios, such as simple industrial inspection, environmental monitoring, and low-power visual sensing.

Supplementary Materials: The following supporting information can be downloaded at: <https://www.mdpi.com/article/10.3390/computers15060395/s1>, uploaded with this submission is intended to be published alongside the article. It contains the implementation scripts, model definitions, and ONNX/PYNQ-Z2 deployment utilities used in this study.

Author Contributions: Y.G. and J.Z. contributed equally to this work. Y.G. organized the manuscript materials and drafted the paper. J.Z. carried out model construction and experimental testing. X.L. provided experimental conditions and supported hardware testing. W.W. designed the overall model

structure and lightweighting strategy. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Project of Underwater Electronics and Material Testing (Grant No. 0000-6602-I5202501), the National Natural Science Foundation of China (Grant Nos. 62566020 and 62561057), and the Foundation of Hainan Institute of Zhejiang University (Grant No. 6602-A12202).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The CIFAR-100 dataset used in this study is publicly available at <https://www.cs.toronto.edu/~kriz/cifar.html> (accessed on 19 May 2026). No additional dataset was generated for this study.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AXI	Advanced eXtensible Interface
CNN	Convolutional neural network
CPU	Central processing unit
DDR3	Double data rate 3
ECA	Efficient channel attention
FP32	32-bit floating point
FPGA	Field-programmable gate array
GPU	Graphics processing unit
KL	Kullback–Leibler
MACs	Multiply–accumulate operations
ONNX	Open Neural Network Exchange
RGB	Red–green–blue
SE	Squeeze and excitation
VDMA	Video direct memory access

References

1. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778. [\[CrossRef\]](#)
2. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861.
3. Zhang, X.; Zhou, X.; Lin, M.; Sun, J. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 6848–6856. [\[CrossRef\]](#)
4. Han, K.; Wang, Y.; Tian, Q.; Guo, J.; Xu, C.; Xu, C. GhostNet: More features from cheap operations. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 1580–1589. [\[CrossRef\]](#)
5. Tan, M.; Le, Q. EfficientNet: Rethinking model scaling for convolutional neural networks. In Proceedings of the International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019; pp. 6105–6114.
6. Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; Graf, H.P. Pruning filters for efficient ConvNets. In Proceedings of the International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
7. Xue, Y.; Yao, W.; Peng, S.; Yao, S. Automatic filter pruning algorithm for image classification. *Appl. Intell.* **2024**, *54*, 216–230. [\[CrossRef\]](#)
8. Yang, L.; Gu, S.; Shen, C.; Zhao, X.; Hu, Q. Soft independence guided filter pruning. *Pattern Recognit.* **2024**, *153*, 110488. [\[CrossRef\]](#)
9. Denton, E.L.; Zaremba, W.; Bruna, J.; LeCun, Y.; Fergus, R. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 2014; Volume 27, pp. 1269–1277.

10. Jaderberg, M.; Vedaldi, A.; Zisserman, A. Speeding up convolutional neural networks with low rank expansions. In Proceedings of the British Machine Vision Conference, Nottingham, UK, 1–5 September 2014; pp. 1–13. [\[CrossRef\]](#)
11. Han, S.; Mao, H.; Dally, W.J. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In Proceedings of the International Conference on Learning Representations, San Juan, Puerto Rico, 2–4 May 2016.
12. Hinton, G.; Vinyals, O.; Dean, J. Distilling the knowledge in a neural network. *arXiv* **2015**, arXiv:1503.02531.
13. Tian, Y.; Krishnan, D.; Isola, P. Contrastive representation distillation. In Proceedings of the International Conference on Learning Representations, Addis Ababa, Ethiopia, 26–30 April 2020.
14. Ding, X.; Guo, Y.; Ding, G.; Han, J. ACNet: Strengthening the kernel skeletons for powerful CNN via asymmetric convolution blocks. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, Republic of Korea, 27 October–2 November 2019; pp. 1911–1920. [\[CrossRef\]](#)
15. Ding, X.; Zhang, X.; Ma, N.; Han, J.; Ding, G.; Sun, J. RepVGG: Making VGG-style ConvNets great again. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 20–25 June 2021; pp. 13733–13742. [\[CrossRef\]](#)
16. Hu, J.; Shen, L.; Sun, G. Squeeze-and-excitation networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 7132–7141. [\[CrossRef\]](#)
17. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*; Curran Associates Inc.: Red Hook, NY, USA, 2019; Volume 32.
18. ONNX. Open Neural Network Exchange. Available online: <https://onnx.ai/> (accessed on 19 May 2026).
19. ONNX Runtime Developers. ONNX Runtime. Available online: <https://onnxruntime.ai/> (accessed on 19 May 2026).
20. Krizhevsky, A. *Learning Multiple Layers of Features from Tiny Images*; University of Toronto: Toronto, ON, Canada, 2009.
21. Iandola, F.N.; Han, S.; Moskewicz, M.W.; Ashraf, K.; Dally, W.J.; Keutzer, K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and less than 0.5 MB model size. *arXiv* **2016**, arXiv:1602.07360.
22. Wang, Q.; Wu, B.; Zhu, P.; Li, P.; Zuo, W.; Hu, Q. ECA-Net: Efficient channel attention for deep convolutional neural networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 11534–11542. [\[CrossRef\]](#)

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.