Computers, Materials & Continua

Tech Science Press

Doi:10.32604/cmc.2025.070990

ARTICLE

ResghostNet: Boosting GhostNet with Residual Connections and Adaptive-SE Blocks

Yuang Chen[1,2], **Yong Li**[1,*], **Fang Lin**[1,2], **Shuhan Lv**[1,2] **and Jiaze Jiang**[1,2]

[1]Key Laboratory of CTC & IE (Engineering University of PAP), Ministry of Education, Xi'an, 710086, China
[2]Graduate Student Brigade, Engineering University of PAP, Xi'an, 710086, China

*Corresponding Author: Yong Li. Email: liyong@nudt.edu.cn

Received: 29 July 2025; Accepted: 26 September 2025; Published: 09 December 2025

ABSTRACT: Aiming at the problem of potential information noise introduced during the generation of ghost feature maps in GhostNet, this paper proposes a novel lightweight neural network model called ResghostNet. This model constructs the Resghost Module by combining residual connections and Adaptive-SE Blocks, which enhances the quality of generated feature maps through direct propagation of original input information and selection of important channels before cheap operations. Specifically, ResghostNet introduces residual connections on the basis of the Ghost Module to optimize the information flow, and designs a weight self-attention mechanism combined with SE blocks to enhance feature expression capabilities in cheap operations. Experimental results on the ImageNet dataset show that, compared to GhostNet, ResghostNet achieves higher accuracy while reducing the number of parameters by 52%. Although the computational complexity increases, by optimizing the usage strategy of GPU cache memory, the model's inference speed becomes faster. The ResghostNet is optimized in terms of classification accuracy and the number of model parameters, and shows great potential in edge computing devices.

KEYWORDS: Residual connections; adaptive-SE blocks; lightweight neural network; GPU memory usage

1 Introduction

With the rapid development of artificial intelligence technologies, the demand for lightweight models in edge computing devices is growing urgently [1], especially in scenarios such as industrial monitoring [2], equipment status perception [3], and real-time safety early warning [4], where efficient and low-latency vision models have become a key support for realizing intelligent upgrades. Traditional convolutional neural networks improve performance by increasing network depth, expanding parameter scales, and optimizing growing network structures [5], whereas lightweight models need to achieve high performance computing in resource-constrained environments [6,7]. This fundamental difference in design philosophy makes the design of lightweight models more challenging.

Currently, the main methods for model lightweighting include pruning [8], architecture design [6], neural architecture search (NAS) [9], and knowledge distillation [10]. Based on these methods, researchers have designed many classic lightweight models, such as the MobileNet series [11–13], ShuffleNet series [14,15], SqueezeNet [16], and GhostNet [17], etc. These classic lightweight models typically combine pointwise convolutions and depthwise convolutions to reduce computational complexity and achieve weight reduction. Among them, GhostNet creatively introduced the Ghost module in its architecture, using a combination of primary convolutions and cheap operations instead of pointwise convolutions to achieve a breakthrough in feature generation mechanisms.

However, there are several key issues in this innovative design that deserve in-depth exploration: What is the basis for generating ghost feature maps using the Ghost module? Can all original feature maps generate ghost feature maps? Will some ghost feature maps become noise during feature extraction?

In response to these questions, this paper conducts a new exploration of the method for GhostNet to generate ghost feature maps and designs the ResghostNet model. Its design concept is as follows:

1. Transform the original Ghost Module into a residual connection [18], named Resghost Module, to retain original input information and reduce the impact of low-quality ghost feature maps on subsequent feature extraction. This improvement ensures that key feature information can be directly passed to deeper network layers.
2. The SE module is added to the shortcut connection branch of the Resghost Module to improve the model's expressive ability and overall performance.
3. Design an Adaptive SE block and introduce it before cheap operations to enhance the expressive capabilities of high-quality feature maps in cheap operations.

As shown in Fig. 1, the model architecture designed in this paper is based on the above design methods. Although the computational complexity has increased, the additional computations occur in the cached memory on the GPU during runtime. Meanwhile, the reduced number of parameters alleviates the pressure on the allocated memory on the GPU [19]. Therefore, compared with GhostNet, the inference speed has even become faster.

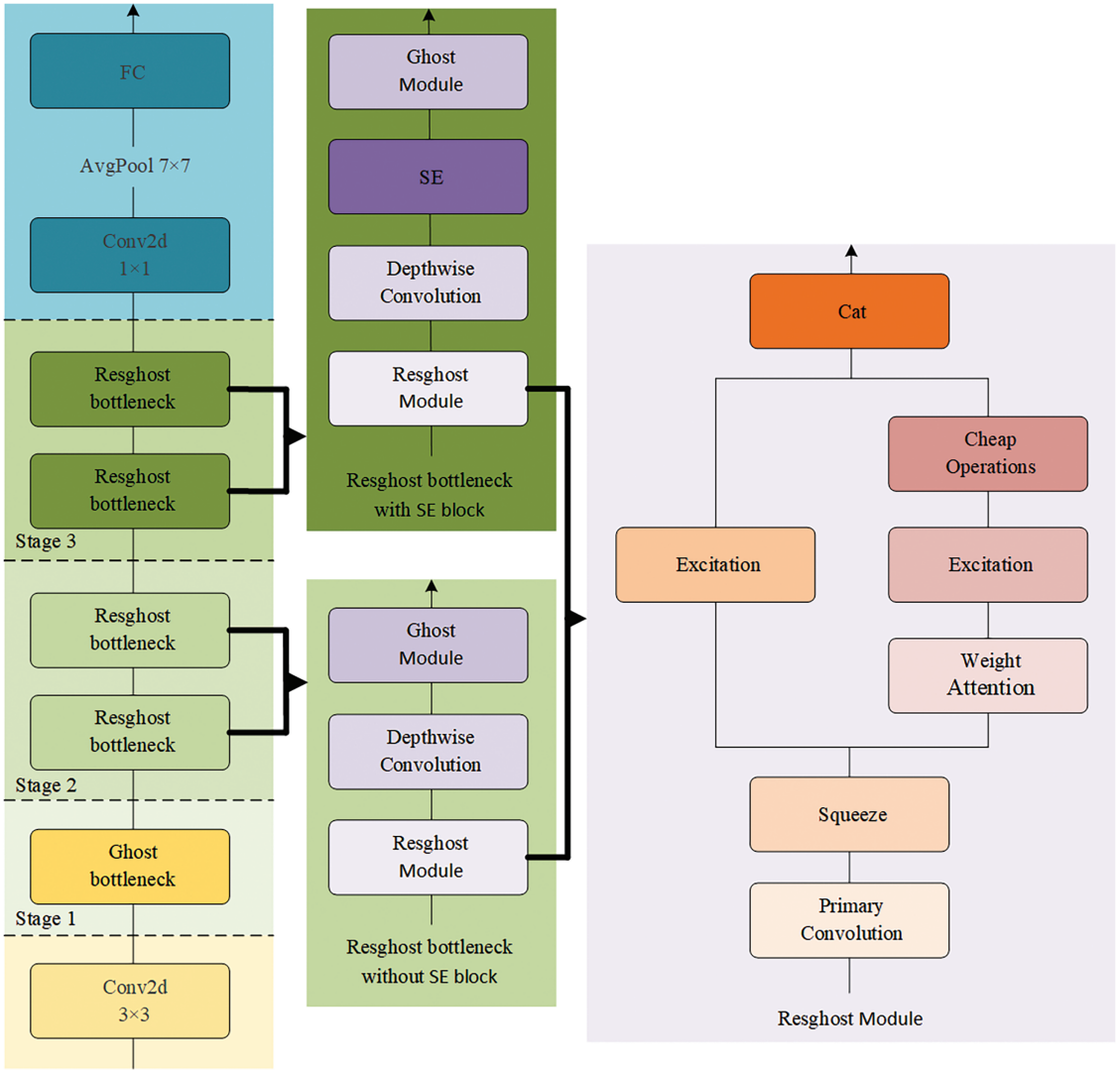

Figure 1: Architecture of ResghostNet. The leftmost part is the backbone of ResghostNet. In the middle section, the dark-green backgrounds denote Resghost bottlenecks with an SE block, while the light-green backgrounds indicate Resghost bottlenecks without an SE block. The rightmost part is the Resghost Module

2 Related Work

2.1 GhostNet

Han et al. [17] proposed GhostNet, an efficient lightweight convolutional neural network architecture in 2020, aiming to reduce computational costs through cheap operations while maintaining high performance. As shown in Fig. 2a, GhostNet consists of 16 Ghost bottlenecks, mainly used for feature extraction, with the Ghost Module as its core component. As shown in Fig. 2b, the operation steps of the Ghost Module are to first perform a small number of ordinary convolution operations on the input feature maps to generate some basic feature maps, and then apply a series of cheap operations to these basic feature maps to generate more ghost feature maps. The ingenuity of this design lies in achieving effects similar to pointwise convolutions with fewer computational resources.

However, while cheap operations can improve computational efficiency, its processing of basic feature maps is not detailed enough. In the process of generating ghost feature maps, all basic feature maps are not distinguished according to their importance or relevance. This means that some basic feature maps that may contain key information do not receive more detailed processing, while some less important ones may receive excessive attention. This non-selective processing method may affect the quality of feature extraction and potentially negatively impact the overall model performance. Based on these considerations, this paper improves the GhostNet design to optimize the generation process of ghost feature maps.

It should be noted that in recent years, some lightweight models based on Transformers, such as MobileViT [20], MobileFormer [21], and EfficientFormer [22], have achieved excellent performance in image classification tasks by combining the local perception ability of CNNs with the global modeling advantages of Transformers. These models typically adopt a "hybrid" architecture, using convolutions in the shallow layers to extract local features and introducing lightweight attention mechanisms in the deeper layers to capture long-range dependencies.

Although the above methods have certain advantages in terms of accuracy, they are limited by the higher computational overhead brought by the architecture they rely on. Despite lightweight improvements, they still limit the deployment efficiency on extremely low-power edge devices. In contrast, pure convolutional architectures such as GhostNet, although having a limited receptive field, have higher computational density and better hardware compatibility, and are especially suitable for delay-sensitive application scenarios.

Therefore, this paper chooses to improve the convolutional architecture rather than turning to the Transformer paradigm. We propose to optimize the generation process of "ghost feature maps" in GhostNet by introducing residual connections and adaptive channel attention mechanisms, thereby improving the quality and selectivity of feature expression without significantly increasing the number of parameters. This design concept aims to balance model accuracy, computational efficiency, and deployment feasibility, and is especially suitable for resource-constrained edge computing environments.

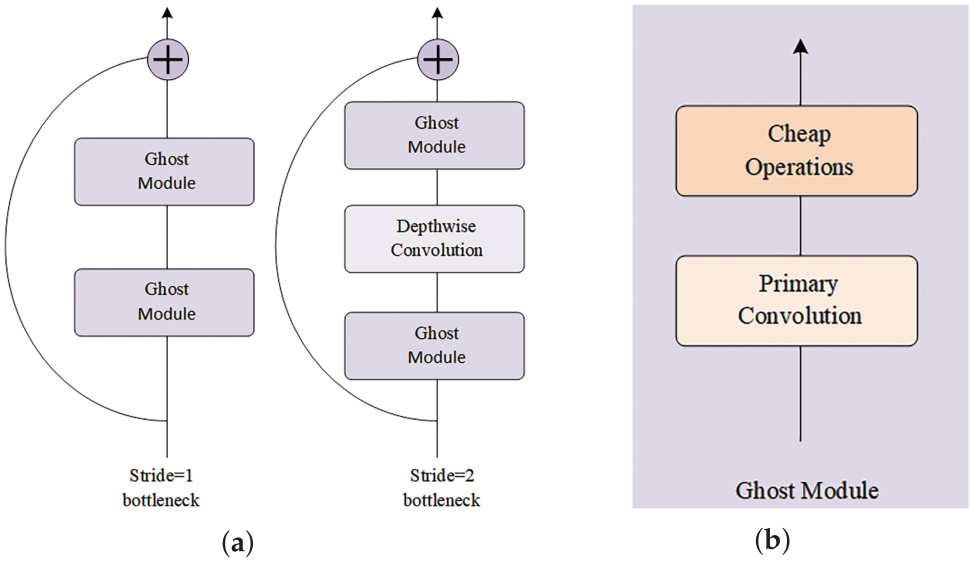

Figure 2: The two-layer structure within GhostNet: (**a**) GhostNet bottleneck; (**b**) Ghost Module

2.2 SE Block

Hu et al. [16] proposed Squeeze-and-Excitation Networks (SENet), an architecture that enhances the representation capabilities of neural networks by modeling the interdependencies between feature channels. The core of this architecture is the SE block, which can be embedded into existing CNNs to improve performance.

In the SE block, S represents the Squeeze operation and E represents the Excitation operation. As shown in Fig. 3, the Squeeze operation uses global average pooling (GAP) to compress the spatial dimension of each channel into a single value. Specifically, for an input feature map with a shape of $B \times C \times H \times W$, the Squeeze operation generates a tensor z with a shape of $B \times C \times 1 \times 1$, where each element represents the global average value of the corresponding channel. In the Excitation operation, first, a fully-connected layer is used to reduce the number of channels from C to C/r for dimensionality reduction, and then another fully-connected layer is used to increase the number of channels back to C, generating a weight tensor $\hat{z}$ with the weight values for each channel. After the Squeeze-Excitation processing, the weight tensor $\hat{z}$ is multiplied element-by-element with the original input feature map X to obtain the recalibrated feature map $\hat{X}$, thereby enhancing useful channels and suppressing less important ones.

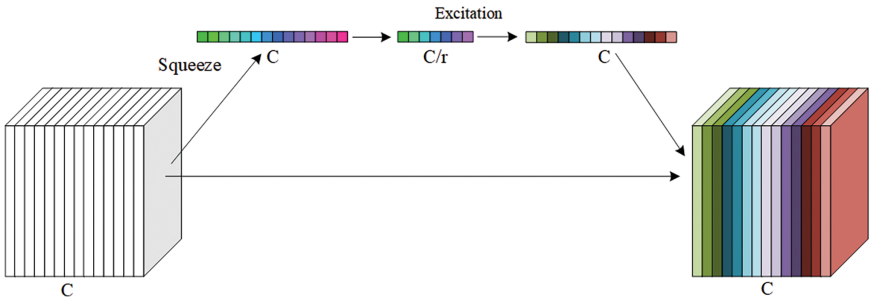

Figure 3: Principle of how the squeeze-and-excitation block processes the input tensor

The design of the SE block enables the model to effectively re-calibrate the channel feature responses, thus enhancing the model's representational ability. It can be well applied before the cheap operations of GhostNet to distinguish the importance of basic feature maps. However, the use of global average pooling in the Squeeze operation, while capable of compressing spatial information to capture global information of each channel, also has the limitation of losing local detail information. Therefore, it can also be improved for application.

2.3 GPU Memory Allocation

Literature [16] proposed that using direct metrics is more important than indirect metrics when designing efficient convolutional neural network architectures. For example, examining the model's inference speed is more direct than comparing FLOPs to reflect the model's computational speed. Therefore, the author Ma et al. proposed four principles for GPU use in the field of deep learning in the literature. On this basis, experiments have found that the performance of deep learning models is also closely related to GPU memory allocation. During the training and inference processes, the different proportions of allocated memory and cached memory on the GPU are crucial for whether the model can achieve better performance.

Allocated memory refers to the actual physical memory directly allocated for a model. When a model is loaded onto a GPU for training or inference, the model's parameters, activation values, gradients, etc. need to be stored in the GPU memory, and this part of the memory is reserved specifically for this model and

cannot be used by other processes [23]. Therefore, when designing lightweight models, reducing the number of parameters is an effective means reducing the occupied allocated memory.

Cached memory refers to the additional memory cached on the GPU in addition to the memory directly allocated for the current task. This memory usually contains temporary data generated during the model training or inference process. This data can be reused to speed up the calculation without having to read data again from the main memory or other slower storage media. Therefore, it is mainly used to predict and preload data that may be needed, and retain frequently accessed data in the cache to effectively reduce waiting time and improve the overall computational efficiency [19,23].

From the operating mechanisms of allocated memory and cached memory, it can be seen that model parameters mainly occupy allocated memory, while data generated during forward or backward propagation mainly occupie cached memory, and the cached memory can improve computational efficiency. Therefore, this paper hypothesizes that by reducing the overall number of model parameters, the allocated memory can be decreased, and the SE blocks, originally residing in the cache memory, can be relocated before the cheap operations. This aims to achieve higher accuracy without any slowdown—or even faster—inference speed.

3 Method

3.1 Adaptive-SE Block

Although the SE block can basically distinguish the importance of different channels, its method of learning channel importance weights through global average pooling and simple fully-connected layers may not be able to fully capture the dynamic relationships and relative importance of each channel within the entire input feature map. Therefore, the SE block still has certain limitations in capturing the complex interdependencies between channels. Thus, in order to further enhance the model's ability to understand the dependencies between channels, this paper designs a weight self-attention mechanism based on the principle of the attention mechanism [24], and its formula is:

$$Weight\_self\_Attention = softmax(z \cdot z^T / \sqrt{d_z}) \tag{1}$$

where z is the weight matrix generated after the Squeeze operation, z^T is the transpose of z, and $\sqrt{d_z}$ is the scaling factor. The role of this attention mechanism is to adjust the weight values according to the relationship between each element and the global, so as to more accurately reflect the relative importance of each channel within the entire input. Specifically, by calculating the similarity scores and normalizing these scores using the $softmax$ function, a probability distribution representing the interrelationships between channels can be obtained. This distribution is used to re-weight the weight matrix after the Squeeze operation to emphasize more important channel features.

The SE block combined with the weight self-attention mechanism is named the Adaptive-SE block in this paper. As shown in Fig. 4, this is the processing process of the Adaptive-SE block for the input feature map. The input feature maps first undergo the Squeeze operation to obtain their weight matrix Subsequently, through the weight self-attention mechanism, each element in the weight matrix adjusts its weight value according to its own relationship with the global, so that important channel features are strengthened. Finally, after the Excitation operation further adjusts the channel weights and uses the $sigmoid$ function to generate the final channel importance weights, these weights are used to re-calibrate the input feature map, thereby outputting the optimized feature map.

Figure 4: Principle of how the Adaptive Squeeze-and-Excitation block processes the input tensor. Compared with the standard SE block, this paper inserts a weight-based self-attention mechanism between the Squeeze and Excitation steps

By introducing the weight self-attention mechanism into the SE block, it can effectively alleviate the problem of single-dimensional feature information extraction caused by only using global average pooling. This enables the model to dynamically adjust the importance weights of each channel based on the current input feature map, rather than simply relying on the fixed results of global average pooling, so as to better capture the key information in the input data.

3.2 Resghost Module

In the design of lightweight neural networks, model parameters (which affect allocated memory) and the memory occupation of temporary feature maps during inference (i.e., cached memory) are two key but often confused resource dimensions. Traditional optimizations often focus on reducing FLOPs or the number of parameters, yet overlook the utilization efficiency of GPU cached memory during actual inference. The Resghost Module proposed in this paper was designed from a systemic perspective from the very beginning, aiming to efficiently utilize the feature maps in cached memory for dynamic computation without significantly increasing the number of parameters, thereby achieving a dual improvement in accuracy and speed.

Specifically, although the Adaptive-SE block introduced in the Resghost Module increases the amount of computation, it does not introduce additional learnable parameters and thus exerts minimal pressure on the model's memory allocation. On the contrary, these computations fully leverage the existing feature data in the GPU's high-speed cache, realizing an efficient strategy of "trading computation for quality"—that is, enhancing the feature expression capability through lightweight attention mechanisms without increasing the memory burden during deployment. This design philosophy enables ResghostNet to maintain high-throughput, low-latency inference performance on edge devices.

Although the Adaptive-SE block can effectively identify important feature maps through channel attention mechanisms, solving the problem of how to determine the importance of feature map channels, in the process of generating ghost feature maps, relying only on simple linear operations is likely to lead to information loss and weakening of feature expression ability. This is especially true in deep-network structures, where it may cause gradient vanishing or network degradation. Inspired by the role of residual connections in stabilizing the training process of deep neural networks, this paper fuses the original feature map and the ghost feature map across layers to construct the Resghost Module with an identity mapping path.

As shown in Fig. 5, the Resghost Module adopts a dual-branch structure to achieve feature reuse: the first branch uses a standard SE block to re-weight the feature maps, strengthening the semantic information of important channels; the second branch generates ghost feature maps through the Adaptive-SE block and cheap operations.

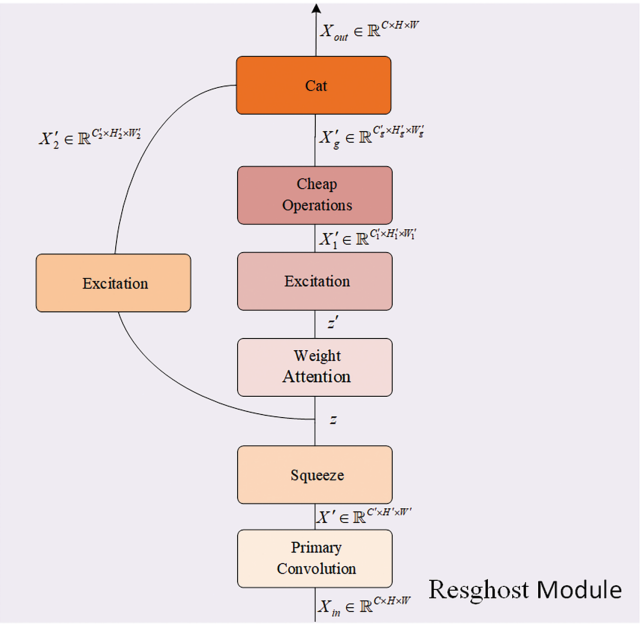

Figure 5: Internal structure of the Resghost Module, comprising a main branch and a residual connection

Specifically, in the design of the Resghost Module, the original input first passes through a primary convolution layer, which is mainly used to generate basic feature maps in preparation for subsequent operations. Let the input feature map be $X_{in} \in \mathbb{R}^{C\times H\times W}$, and after passing through the primary convolution layer, it is transformed into $X' \in \mathbb{R}^{C'\times H'\times W'}$. Since both branches process $X' \in \mathbb{R}^{C'\times H'\times W'}$, the Squeeze operation can be performed on z before the branches to obtain the weight matrix $X' \in \mathbb{R}^{C'\times H'\times W'}$, thereby reducing computational complexity.

$$z = F_s(x') = \frac{1}{H\times W}\sum_{i=1}^{H}\sum_{j=1}^{W} x'(i,j) \tag{2}$$

In the cheap operation branch, the weight matrix z first completes the processing steps of weight self-attention and the Excitation operation in the Adaptive-SE block to obtain the feature map $X_1' \in \mathbb{R}^{C_1'\times H_1'\times W_1'}$.

$$z' = Weight\_self\_Attention(z) = softmax\left(z\cdot z^T/\sqrt{d_z}\right) \tag{3}$$

$$X_1' = F(x', s) = s\cdot x' = F_E(z', W)\cdot x' \tag{4}$$

$$F_E(z', W)\cdot x' = \sigma(g(z', W))\cdot x' = \sigma(W_2\cdot\delta(W_1 z'))\cdot x' \tag{5}$$

where δ refers to the ReLU function, and $W_1 \in \mathbb{R}^{c/r\times c}$ and $W_2 \in \mathbb{R}^{c\times c/r}$ are the weight matrices for dimensionality reduction and dimensionality increase, respectively. After the important channels are weighted by the Adaptive-SE block, a linear operation is performed on the feature map $X_1' \in \mathbb{R}^{C_1'\times H_1'\times W_1'}$ to generate the

ghost feature map $X'_g \in \mathbb{R}^{C'_g \times H'_g \times W'_g}$.

$$X'_g = \Phi(X'_1) \tag{6}$$

Here, Φ represents the cheap operation, which is used to generate ghost feature maps.

In the residual connection path, the previously obtained weight matrix z is processed by the Excitation operation to obtain $X'_2 \in \mathbb{R}^{C'_2 \times H'_2 \times W'_2}$.

$$X'_2 = F(x', s) = s \cdot x' = F_E(z, W) \cdot x' \tag{7}$$

$$F_E(z, W) \cdot x' = \sigma(g(z, W)) \cdot x' = \sigma(W_2 \cdot \delta(W_1 z)) \cdot x' \tag{8}$$

Finally, the outputs $X'_g \in \mathbb{R}^{C'_g \times H'_g \times W'_g}$ and $X'_2 \in \mathbb{R}^{C'_2 \times H'_2 \times W'_2}$ of the two branches are concatenated in the channel dimension to obtain the output $X_{out} \in \mathbb{R}^{C \times H \times W}$.

The dual path design of the Resghost Module constructs an identity mapping path, ensuring the integrity of the original features and optimizing the stability of gradient propagation, effectively alleviating the issue of gradient diffusion in deep networks. The branch for generating ghost feature maps, combined with the Adaptive-SE block, enhances the semantic feature expression ability in cheap linear operations.

3.3 ResghostNet

Based on the innovative design of the Resghost Module, taking an input image of size 224 × 224 as an example, we propose the ResghostNet architecture as shown in Table 1. This model inherits the lightweight concept of GhostNet and uses the basic structure of MobileNetV3 as a reference framework. However, it reconstructs the core bottleneck block, the ghost bottleneck, into a Resghost bottleneck containing the Resghost Module. The initial layer of the network is a standard 3 × 3 convolution layer with 16 filters. Subsequently, it consists of three stages grouped according to the resolution of the feature maps, and each stage contains several Resghost bottlenecks. The stage division and layer configuration are as follows: The first stage contains one Ghost bottleneck layer, using a 3 × 3 depthwise convolution and maintaining input and output channels at 16; the second stage includes two Resghost bottleneck layers, where the first layer performs downsampling with a stride of 2 using a 3 × 3 depthwise convolution, reducing the channels from 48 to 24, and the second layer maintains the feature map size and expands the channels from 72 to 24; the third stage deploys two Resghost bottleneck layers, where the first layer uses a 5 × 5 depthwise convolution for downsampling (stride of 2), expanding the channels from 72 to 40 and introducing an Adaptive SE module with a reduction ratio of 0.25, and the second layer maintains the resolution and expands the channels from 120 to 40, retaining the SE module in the Resghost bottleneck backbone. Downsampling in all stages is achieved through the first layer's convolution with a stride of 2 in each stage. This design reduces computational complexity by adjusting the resolution upfront.

Compared with the traditional GhostNet, the innovations of ResghostNet are reflected in three aspects: First, it introduces the Adaptive-SE block to replace the standard SE block, which enhances the semantic expression of important channels through dynamic weight allocation and mitigates information loss caused by linear operations when combined with cheap operations. Second, it achieves feature reuse through dual-branch residual connections, balancing computational efficiency and representation capability. Third, it reduces the model's depth and consequently the number of parameters while significantly improving performance.

Table 1: Overall architecture of ResghostNet

Input size	Operator	Output size	SE ratio
$224^2 \times 3$	Conv2d 3×3	$112^2 \times 16$	–
$112^2 \times 16$	Ghost bottleneck	$112^2 \times 16$	–
$112^2 \times 16$	Resghost bottleneck	$56^2 \times 24$	–
$56^2 \times 24$	Resghost bottleneck	$56^2 \times 24$	–
$56^2 \times 24$	Resghost bottleneck	$28^2 \times 40$	0.25
$28^2 \times 40$	Resghost bottleneck	$28^2 \times 40$	0.25
$28^2 \times 40$	Conv2d 1×1	$28^2 \times 1280$	–
$28^2 \times 1280$	AvgPool 7×7	$1^2 \times 1280$	–
$1^2 \times 1280$	FC	1000	–

4 Experiments

4.1 Related Configurations

4.1.1 Parameter Settings

The experimental environment of this study is based on the Ubuntu 22.04.5 LTS operating system, using the PyTorch 2.4.1 deep learning framework and accelerated by GPU through CUDA 11.1. The experimental hardware configuration includes an Intel(R) Xeon(R) Gold 6133 CPU @ 2.50 GHz and four NVIDIA RTX 4090 D GPUs (each with 24 GB of GPU memory). RGB images are employed as the input modality, and the optimizer is stochastic gradient descent (SGD). Training is conducted for 200 epochs with a step-wise learning-rate schedule: the learning rate is reduced by 10% at the 92nd and 136th epochs. Table 2 summarizes the experimental performance metrics of the ResghostNet model on three benchmark datasets.

Table 2: Experimental results of ResghostNet on three datasets. Proportion refers to the proportion of cached memory on the GPU

Dataset	Top-1 Acc (%)	Parameters (M)	FLOPs (M)	Inference time (1×10^{-6})	Proportion (%)
CIFAR-10	90.85	0.79	154	148	95.37
CIFAR-100	83.2	0.93	285	169	95.21
ImageNet	74.7	2.49	567	202	95.76

4.1.2 Datasets

The CIFAR-10 and CIFAR-100 datasets used in this study are two classic image datasets widely applied in the field of computer vision. Both of these datasets were compiled by scholars Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton from the University of Toronto in Canada and are part of the CIFAR (Canadian Institute for Advanced Research) series of datasets.

The CIFAR-10 dataset contains 60,000 color images divided into 10 categories, with 6000 images in each category. Among them, 50,000 images are for training and 10,000 are for testing. Each image has a resolution of 32×32 pixels and uses the RGB color space. The images in CIFAR-10 are rich and diverse in content, covering a wide range of object types such as airplanes, cars, and birds. Moreover, the number of samples in each category is balanced, which helps to avoid the problem of class imbalance. Due to the low image resolution, CIFAR-10 is an ideal choice for training lightweight models and is also very suitable for rapid experiments and model development.

In contrast, the CIFAR-100 dataset contains 60,000 color images divided into 100 categories, with 600 images in each category. Among them, 50,000 images are for training and 10,000 are for testing. Each image has a resolution of 32 × 32 pixels and also uses the RGB color space. Different from CIFAR-10, the categories in CIFAR-100 are more fine-grained, covering a wider range of object types such as animals, vehicles, and electrical appliances. The images within each category have a high degree of similarity, which increases the difficulty of the classification task. Due to the low image resolution, CIFAR-100 is also an ideal choice for training lightweight models and is very suitable for tasks that require more precise classification.

The ImageNet dataset comprises more than 14 million manually annotated images spanning over 20,000 categories. Its unprecedented scale and diversity enable models to learn rich feature representations, thereby improving generalization across a wide range of visual recognition tasks. Through the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC), ImageNet has become an internationally recognized benchmark. Researchers and engineers participating in the challenge evaluate their algorithms on the same training and test splits, facilitating direct comparisons between different methods and driving rapid progress in deep learning. Owing to its broad acceptance and formidable difficulty, many pivotal technical and theoretical breakthroughs have emerged from efforts to tackle ImageNet-related problems. For instance, the success of convolutional neural networks is largely attributed to their outstanding performance in the ImageNet Challenge. Consequently, experimenting on ImageNet has become the "gold standard" for validating whether new ideas and technologies are truly effective.

4.2 Ablation Experiments

To investigate the impact of each component of the model on overall performance and to determine the optimal model structure, we conducted ablation experiments. The ablation experiments were carried out on a single computer, with each model trained and inferred on a separate GPU within the computer, using the CIFAR-10 dataset. In the ablation experiments, we explored the performance of four structural variants of the model shown in the Fig. 6. The four structures are as follows: Fig. 6a—the ResghostNet without the weight self-attention mechanism; Fig. 6b—the ResghostNet without the Adaptive-SE block; Fig. 6c—the ResghostNet without the SE block in the shortcut connection branch; Fig. 6d—the ResghostNet with the channel shuffle operation [14].

As shown in Table 3, after removing the Weighted Attention (WA) mechanism and the Adaptive SE block (ASE), the model's Top-1 accuracy dropped to 90.37% and 89.90%, respectively, which is a significant decrease compared to the original ResghostNet's accuracy. This indicates that both the Weighted Attention mechanism and the Adaptive SE block play a crucial role in enhancing the model's expressive power and feature extraction capabilities. Moreover, when the Shortcut connection branch did not integrate the SE block, the model's Top-1 accuracy further decreased to 87.90%. This suggests that the SE block not only improves the quality of feature maps but also effectively promotes the propagation of information through deeper layers of the network. In the model variant that introduced Channel Shuffle, although the operation was designed to optimize the interaction between feature maps and enhance the model's expressive power, the Top-1 accuracy of this variant was only 89.97% in the actual experiment, lower than other variants. This result indicates that Channel Shuffle may not be suitable for the current network architecture, or it needs to be combined with other optimization strategies to realize its potential advantages. From the perspective of inference efficiency, although ResghostNet's floating-point operations (FLOPs) increased compared to GhostNet, the model's inference speed was not significantly affected due to the substantial reduction in the number of parameters. This proves that our assumptions about computational resource allocation in the design phase are valid, that is, by optimizing the use of cached memory on the GPU, higher model accuracy can be achieved without compromising inference efficiency.

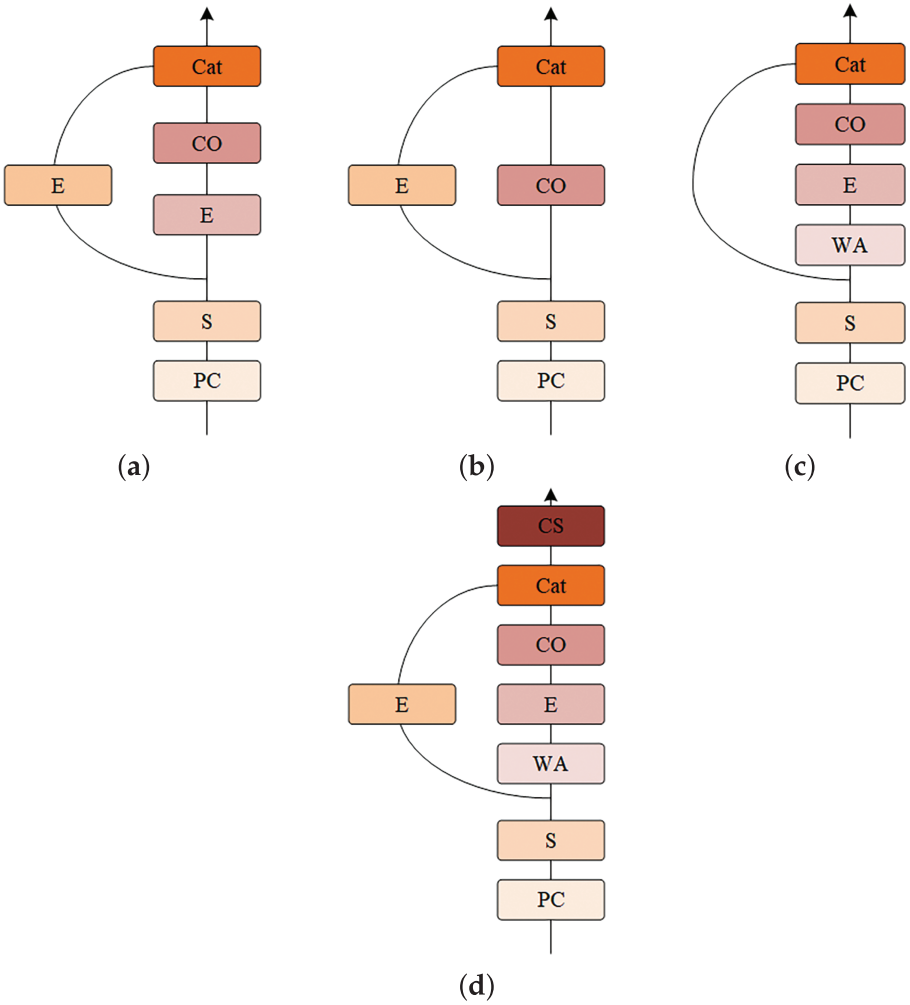

Figure 6: The four model structures used in the ablation experiments. (**a**) the ResghostNet without the weight self-attention mechanism, (**b**) the ResghostNet without the Adaptive-SE block, (**c**) the ResghostNet without the SE block in the shortcut connection branch, (**d**) the ResghostNet with the channel shuffle operation

Table 3: Results of ablation experiments. Inference time is measured as the per-image processing duration. Proportion refers to the proportion of cached memory on the GPU

Model	Top-1 Acc (%)	Parameters	FLOPs (M)	Inference time (1×10^{-6})	Proportion (%)
ResghostNet	90.85	789,518	154	148	95.37
ResghostNet-WA	90.37	789,518	148	172	94.23
ResghostNet-ASE	89.90	789,518	146	163	93.84
ResghostNet-SE	87.90	789,518	151	168	94.13
ResghostNet+CS	89.97	789,518	154	167	95.43

4.3 Experiments on the CIFAR-10 Dataset

We first conducted image classification experiments on the CIFAR-10 dataset. The experimental results in Table 4 fully demonstrate the effectiveness of the proposed ResghostNet and its variant ResghostNet Small in lightweight model design. Overall, by introducing residual connections, SE blocks, and Adaptive-SE

blocks, ResghostNet not only significantly improved the model's classification performance but also achieved optimizations in terms of model parameter count and memory usage.

Table 4: Experimental results of classic lightweight models on the CIFAR-10 dataset. Proportion refers to the proportion of cached memory on the GPU

Model	Top-1 Acc (%)	Parameters	Inference time (1×10^{-6})	Proportion
ResghostNet(ours)	90.85	789,518	148	95.37
ResghostNet Small(ours)	90.05	402,028	140	95.26
GhostNet [17]	83.44	3,912,650	193	85.98
SqueezeNet [16]	88.22	746,292	134	97.15
MobileNetV2×0.5 [12]	82.35	700,778	138	91.25
MobileNetV2 [12]	86.18	2,237,770	147	92.99
MobileNetV2×1.5 [12]	88.80	4,958,762	156	92.78
MobileNetV3 Small [13]	80.46	1,528,106	171	85.80
MobileNetV3 Large [13]	85.39	4,214,842	176	87.67
ShuffleNetV2×0.5 [15]	77.35	352,042	169	79.10
ShuffleNetV2 [15]	81.24	1,258,576	170	85.52
ShuffleNetV2×1.5 [15]	83.05	2,488,874	176	84.09

In terms of classification accuracy, ResghostNet achieved a Top-1 accuracy of 90.85% and a Top-5 accuracy of 99.61%, significantly outperforming other lightweight models. Although ResghostNet Small had a slightly lower Top-1 accuracy of 90.05% compared to ResghostNet, it still significantly surpassed other classic models. This result indicates that even with a substantial reduction in model parameters, ResghostNet Small can still maintain high classification performance.

In terms of model parameter count, ResghostNet Small demonstrates a significant advantage. It has only 402,028 parameters, much lower than ResghostNet and other classic lightweight models. This substantial reduction in parameters is attributed to the optimized design of the feature extraction layers in ResghostNet—by introducing residual connections and Adaptive-SE blocks, ResghostNet effectively reduced the number of feature extraction layers while improving the quality of the feature maps. This not only decreased the model's storage requirements but also provided greater flexibility for deploying on resource-constrained devices.

Regarding inference speed, ResghostNet and ResghostNet Small both demonstrated a good balance. Although ResghostNet has an inference speed of 148×10^{-6} seconds per image, and ResghostNet Small has an inference speed of 140×10^{-6} seconds per image, both models exhibit fast inference speeds. Compared to other models, the ResghostNet series shows how to achieve higher accuracy and more efficient resource utilization without sacrificing too much inference speed.

In terms of memory usage, ResghostNet and ResghostNet Small also demonstrated excellent performance. The proportion of GPU cached memory used by ResghostNet was 95.37%, and that of ResghostNet Small was 95.26%. This indicates that ResghostNet can effectively utilize the GPU memory mechanism by increasing the proportion of cached memory, thereby accelerating model inference speed.

Fig. 7 shows the performance of various classic models on the CIFAR-10 dataset. The x-axis denotes inference time per image, the y-axis denotes accuracy, and the bubble size reflects the model's memory footprint. Bubbles of the same color belong to the same model family; the scaling factor is 1.5×10^{-5}.

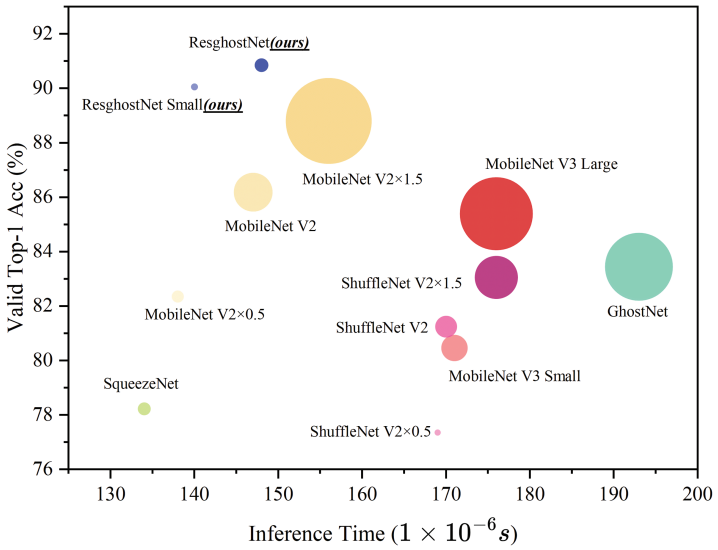

Figure 7: Performance of various classic models on the CIFAR-10 dataset. The x-axis denotes inference time per image, the y-axis denotes accuracy, and the bubble size reflects the model's memory footprint. Bubbles of the same color belong to the same model family; the scaling factor is 1.5×10^{-5}

4.4 Experiments on the CIFAR-100 Dataset

We validated the effectiveness of ResghostNet and ResghostNet Small on the CIFAR-100 dataset. To ensure the rigor of our conclusions, we conducted additional experiments on the CIFAR-100 dataset. The results in Table 5 further demonstrate the effectiveness of ResghostNet and ResghostNet Small in the design of lightweight models.

Table 5: Experimental results of classic lightweight models on the CIFAR-100 dataset. Proportion refers to the proportion of cached memory on the GPU

Model	Top-1 Acc (%)	Parameters	Inference time (1×10^{-6})	Proportion (%)
ResghostNet (ours)	83.2	927,802	169	95.21
ResghostNet Small (ours)	80.9	593,967	148	95.17
GhostNet [17]	76.2	4,027,940	206	85.90
SqueezeNet [16]	64.0	746,292	137	97.15
MobileNetV2 × 0.5 [12]	70.1	816,068	151	90.95
MobileNetV2 [12]	77.7	2,353,060	151	92.83
MobileNetV2 × 1.5 [12]	79.3	5,131,652	155	92.61
MobileNetV3 Small [13]	72.0	1,620,356	159	85.49
MobileNetV3 Large [13]	82.0	4,330,132	166	87.47
ShuffleNetV2 × 0.5 [15]	68.1	444,292	152	79.28
ShuffleNetV2 [15]	70.0	1,350,826	159	81.53
ShuffleNetV2 × 1.5 [15]	74.1	2,581,124	160	83.82

In terms of classification accuracy, ResghostNet achieved a Top-1 accuracy of 83.2%, while ResghostNet Small had a Top-1 accuracy of 80.9%. Although ResghostNet Small had a slightly lower Top-1 accuracy compared to ResghostNet, both models significantly outperformed other lightweight models.

For example, GhostNet achieved a Top-1 accuracy of 76.2%, much lower than the ResghostNet series; MobileNetV3 Small achieved a Top-1 accuracy of 72.0%, which is also significantly lower than the ResghostNet series; ShuffleNetV2 × 1.5 achieved a Top-1 accuracy of 74.1%, which is also lower than the ResghostNet series. These results indicate that the ResghostNet can better preserve the original input information and enhance feature expression capabilities, thereby significantly improving the model's classification performance.

In terms of the number of model parameters, ResghostNet Small demonstrates significant advantages. Its number of parameters is only 593,967, which is far lower than that of ResghostNet and other classic lightweight models. For instance, MobileNetV2 × 0.5 has 816,068 parameters, and ShuffleNetV2 × 1.5 has 2,581,124 parameters. This substantial reduction in the number of parameters is attributed to the optimized design of the feature extraction layers in ResghostNet. By introducing the residual structure and the Adaptive-SE block, ResghostNet effectively reduces the number of feature extraction layers while improving the quality of feature maps. This not only reduces the storage requirements of the model but also provides greater flexibility for deploying the model on resource-constrained devices.

In terms of inference speed, through adaptive adjust and optimization of the model structure, ResghostNet and ResghostNet Small have demonstrated significant performance advantages. Specifically, ResghostNet achieved an inference speed of 169×10^{-6} s per image, while ResghostNet Small further improved this to 148×10^{-6} s per image. This result indicates that the ResghostNet series outperforms most of the compared classic models in terms of inference speed. Notably, SqueezeNet, the fastest model, achieves an inference time of 134×10^{-6} s, yet its Top-1 accuracy is only 64.0%—substantially lower than the 83.2% of ResghostNet Small and the 80.9% of ResghostNet Small. This indicates that, while SqueezeNet enjoys a speed advantage, its classification performance cannot rival that of the ResghostNet family.

Fig. 8 shows the performance of various classic models on the CIFAR-100 dataset. The x-axis represents inference time per image, the y-axis denotes accuracy, and the bubble size indicates the model's memory footprint. Bubbles of the same color belong to the same model family; the scaling factor is 1×10^{-5}.

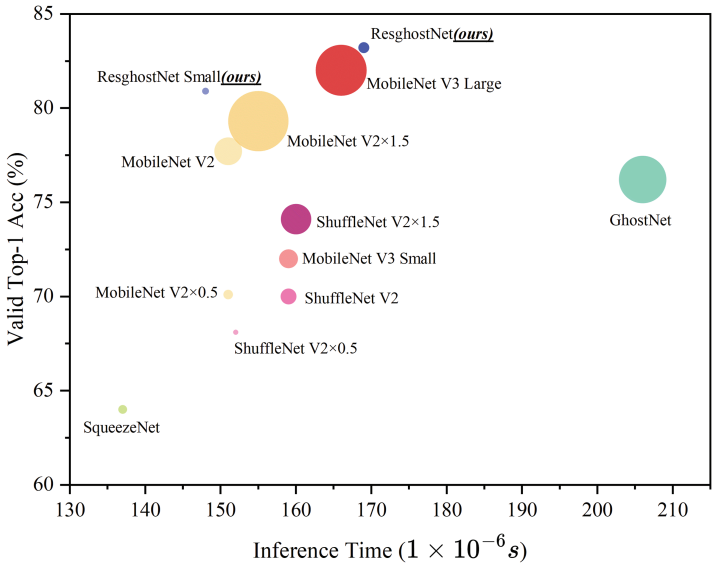

Figure 8: Performance of various classic models on the CIFAR-100 dataset. The x-axis represents inference time per image, the y-axis denotes accuracy, and the bubble size indicates the model's memory footprint. Bubbles of the same color belong to the same model family; the scaling factor is 1×10^{-5}

4.5 Experiments on ImageNet Dataset

Finally, we conduct experiments on ImageNet, comparing ResghostNet and ResghostNet Small against the GhostNet family, MobileNetV2 family, MobileNetV3_Large family, and MobileNetV3_Small family—representative lightweight models. The comparison results shown in Fig. 9 further demonstrate the effectiveness of ResghostNet and ResghostNet Small in lightweight model design.

Fig. 9 confirms the effectiveness of ResghostNet and ResghostNet Small in lightweight model design. For GhostNet, accuracy rises gradually with increasing parameters, but the gains are modest. MobileNetV2 performs well at low parameter counts yet plateaus as the model grows. MobileNetV3_Large delivers competitive results in the mid-parameter regime, but further increases yield diminishing returns. MobileNetV3_Small starts poorly at very low parameters, yet its accuracy improves markedly once capacity rises. Across all parameter levels, the ResghostNet family maintains superior accuracy, especially at the low-end, where it significantly outperforms every alternative. ResghostNet attains high accuracy with far fewer parameters, underscoring its advantage in lightweight design. As parameters continue to grow, accuracy keeps climbing, demonstrating excellent scalability and robustness. Both ResghostNet and ResghostNet Small thus excel in resource-constrained scenarios, offering strong practical potential and clear benefits for real-world deployment.

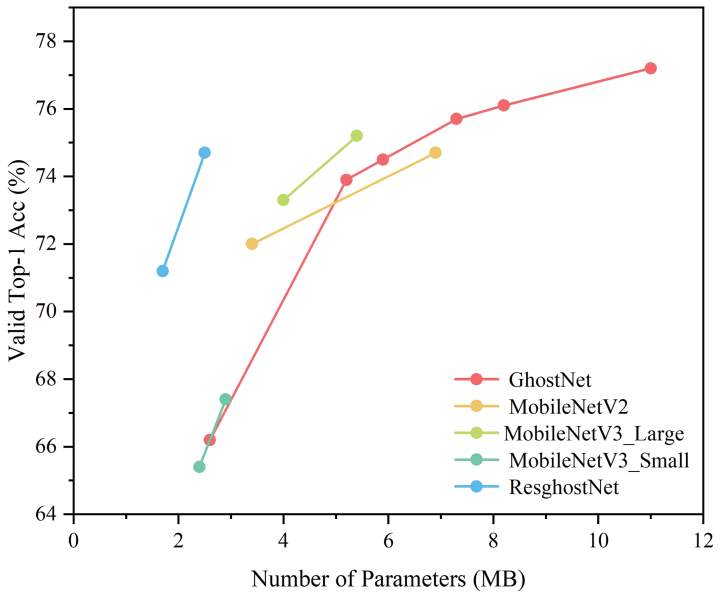

Figure 9: Performance comparison of representative models on ImageNet. The x-axis indicates the number of parameters, and the y-axis denotes Top-1 accuracy. Each line depicts how the accuracy of a lightweight model series evolves as its parameter count increases, with lines of the same color belonging to the same model family

5 Conclusion

By combining the Ghost module with the residual structure and introducing the Adaptive-SE block, this study has successfully improved the quality of feature maps and significantly reduced the model's depth and the number of parameters. Nevertheless, experimental results on ImageNet show that ResghostNet surpasses GhostNet in accuracy while cutting the parameter count by 52%. These results demonstrate that the design enhancements significantly boosted the classification performance of ResghostNet.

It is worth noting that, despite the increase in computational complexity, by optimizing the usage strategy of GPU cache memory, the inference speed of the model is not significantly affected. This indicates

that, on the premise of reasonable allocation of computing resources, ResghostNet can achieve higher model accuracy while maintaining an efficient inference speed.

These results indicate that the ResghostNet series of models provide an efficient and powerful solution for edge computing devices, with broad application prospects. Future research can further explore methods to optimize the balance between computational efficiency and model performance, and conduct in-depth studies on hardware limitations in different application scenarios to further enhance the resource utilization efficiency of the models.

Acknowledgement: Not applicable.

Funding Statement: This research was funded by Science and Technology Innovation Project grant No. ZZKY20222304.

Author Contributions: The authors confirm contribution to the paper as follows: Conceptualization, Yuang Chen and Yong Li; methodology, Yuang Chen and Yong Li; software, Yuang Chen; validation, Fang Lin and Yong Li; formal analysis, Yuang Chen; investigation, Fang Lin and Jiaze Jiang; resources, Yong Li; data curation, Shuhan Lv and Jiaze Jiang; writing—original draft preparation, Yuang Chen; writing—review and editing, Yong Li; visualization, Yuang Chen; supervision, Shuhan Lv; project administration, Yong Li; funding acquisition, Yong Li. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The datasets used in this study are publicly available as follows: 1. CIFAR-10 and CIFAR-100 Dataset: The CIFAR-10 dataset that supports the findings of this study is openly available at https://www.cs.toronto.edu/~kriz/cifar.html (accessed on 25 September 2025); 2. ImageNet Dataset: The ImageNet dataset that supports the findings of this study is openly available at https://image-net.org/. No restrictions apply to the availability of these datasets, and they can be accessed freely by following the provided URLs.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

1. Zhang X, Wang Y, Lu S, Liu L, Xu L, Shi W. OpenEI: an open framework for edge intelligence. Electr Eng Syst Sci. 2019;33:1840–51. doi:10.1109/icdcs.2019.00182.
2. Boumaraf S, Li P, Radi MA, Abdelhafez FO, Behouch A, Awadhi KYAl, et al. Optimized flare performance analysis through multi-modal machine learning and temporal standard deviation enhancements. IEEE Access. 2025;13:34362–77. doi:10.1109/access.2025.3540558.
3. Boumaraf S, Radi MA, Abdelhafez FO, Li P, Awadhi KYAl, Karki H, et al. Vision-based air-flow monitoring in industrial flares system design using deep convolutional neural networks. Expert Syst Appl. 2025;272:126733. doi:10.1016/j.eswa.2025.126733.
4. Chen Y, Li Y, Li S, Lv S, Lin F. DualCascadeTSF-MobileNetV2: a lightweight violence behavior recognition model. Appl Sci. 2025;15(7):3862. doi:10.3390/app15073862.
5. Khan A, Sohail A, Zahoora U, Qureshi AS. A survey of the recent architectures of deep convolutional neural networks. Artif Intell Rev. 2020;53(8):5455–516. doi:10.1007/s10462-020-09825-6.
6. Chen F, Li S, Han J, Ren F, Yang Z. Review of lightweight deep convolutional neural networks. Arch Comput Methods Eng. 2024;31(4):1915–37. doi:10.1007/s11831-023-09973-w.
7. Liu Y, Xue J, Li D, Zhang W, Chiew TK, Xu Z. Image recognition based on lightweight convolutional neural network: recent advances. Image Vis Comput. 2024;146:105037. doi:10.1016/j.imavis.2023.105037.
8. Lin K, Xu X, Gao L, Wang Z, Shen HT. Pruning from scratch. Proc AAAI Conf Artif Intell. 2020;34(7):12273–80. doi:10.1609/aaai.v34i07.6817.
9. Elsken THJ, Metzen JH, Hutter F. Neural architecture search: a survey. J Mach Learn Res. 2019;20(30):1–21.

10. Li T, Li J, Liu Z, Zhang C. Knowledge distillation from few samples. Statistics. arXiv:1811.05047. 2018.
11. Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, et al. MobileNets: efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861. 2017.
12. Sandler M, Howard A, Zhu M, Zhmoginov A, Chen LC. MobileNetV2: inverted residuals and linear bottlenecks. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR); 2018 Jun 18–22; Salt Lake City, UT, USA. p. 4510–20. doi:10.1109/CVPR.2018.00474.
13. Howard A, Sandler M, Chen B, Wang W, Chen LC, Tan M, et al. Searching for MobileNetV3. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV); 2019 Oct 27–31; Seoul, Republic of Korea. p. 1314–24. doi:10.1109/ICCV.2019.00140.
14. Zhang X, Zhou X, Lin M, Sun J. ShuffleNet: an extremely efficient convolutional neural network for mobile devices. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR); 2018 Jun 18–22; Salt Lake City, UT, USA. p. 6848–56. doi:10.1109/CVPR.2018.00716.
15. Ma N, Zhang X, Zheng HT, Sun J. ShuffleNet V2: practical guidelines for efficient CNN architecture design. In: Proceedings of the European Conference on Computer Vision (ECCV); 2018 Sep 8–14; Munich, Germany. p. 116–31. doi:10.1007/978-3-030-01228-1_7.
16. Hu J, Shen L, Albanie S, Sun G, Wu E. Squeeze-and-excitation networks. IEEE Trans Pattern Anal Mach Intell. 2020;42(8):2011–23. doi:10.1109/TPAMI.2019.2913372.
17. Han K, Wang Y, Tian Q, Guo J, Xu C, Xu C. GhostNet: more features from cheap operations. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR); 2020 Jun 13–19; Seattle, WA, USA. p. 1581–90. doi:10.1109/CVPR42600.2020.00166.
18. Shafiq M, Gu Z. Deep residual learning for image recognition: a survey. Appl Sci. 2022;12(8972):8972. doi:10.3390/app12178972.
19. Xiao W, Ren S, Li Y. AntMan: dynamic scaling on GPU clusters for deep learning. In: Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation (OSDI'20); 2020 Jul 13–15; Virtual. p. 357–72. doi:10.1109/msp.2010.134.
20. Mehta S, Rastegari M. MobileViT: Light-weight, general-purpose, and mobile-friendly vision transformer. arXiv:2201.00986. 2022.
21. Chen Y, Dai X, Chen D, Liu M, Dong X, Yuan L, et al. Mobile-former: bridging mobilenet and transformer. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR); 2022 Jun 18–24; New Orleans, LA, USA. p. 13396–405.
22. Li Y, Yuan G, Wen Y, Hu E, Evangelidis G, Tulyakov S, et al. EfficientFormer: vision transformers at MobileNet speed. arXiv:2206.01191. 2022.
23. Huang CC, Jin G, Li J. SwapAdvisor: pushing deep learning beyond the GPU memory limit via smart swapping. In: Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20); 2020 Mar 9–13; Lausanne, Switzerland. p. 675–88. doi:10.1145/3377555.3377901.
24. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is all you need. In: Advances in Neural Information Processing Systems (NeurIPS); 2017 Dec 4–9; Long Beach, CA, USA. p. 5998–6008.