Empirical Study of Spatial and Channel Reconstruction Convolution

Zih-Ci Lin
Department of Computer Science
National Chengchi University
Taipei, Taiwan
jasonlin930309@gmail.com

Tsung-Yen Yu s
Department of Computer Science
National Chengchi University
Taipei, Taiwan
benny999.yu@gmail.com

Sheng-Yuan Yen
Department of Computer Science
National Chengchi University
Taipei, Taiwan
iscoopy321@gmail.com

Guan-Zhi, Wang
Department of Computer Science
National Chengchi University
Taipei, Taiwan
kidflash7011@gmail.com

***Abstract*—Convolutional Neural Networks (CNNs) often suffer from substantial computational redundancy in both spatial and channel dimensions, which limits their deployment in resource-constrained environments. This paper investigates the effectiveness of Spatial and Channel Reconstruction Convolution (SCConv) as a plug-and-play module for suppressing feature redundancy and compressing convolutional architectures. We integrate SCConv into ResNet50 and DenseNet121 backbones and evaluate their performance on the CIFAR-100 and Food-101 benchmarks.**

Experimental results demonstrate that SCConv effectively reduces theoretical model complexity. For example, the SCConv-R50 variant achieves significant reductions in both parameter count and FLOPs compared to the baseline ResNet50. However, our empirical analysis reveals a critical trade-off: despite lower theoretical computational cost, SCConv-based models incur substantially higher GPU memory consumption and require longer training time, indicating that SCConv is memory-bound in practice. Furthermore, while maintaining competitive classification accuracy, SCConv-enhanced models exhibit increased instability in validation behavior on the fine-grained Food-101 dataset. These findings highlight the gap between theoretical efficiency gains and practical training efficiency, providing important insights into the real-world deployment of reconstruction-based convolution modules.

Keywords—convolutional neural networks, SCConv, feature redundancy, model efficiency, lightweight architectures, training stability

I. INTRODUCTION

A. Research Background and Problem Statement

In recent years, Convolutional Neural Networks (CNNs) have achieved remarkable success in various computer vision tasks due to their powerful feature extraction capabilities. However, this performance improvement often comes with substantial computational resource and storage requirements. The primary reason for this is that convolutional layers generate a significant amount of "feature redundancy" during the feature extraction process. Current solutions generally fall into two categories: model compression (e.g., network pruning, quantization) and lightweight network design (e.g., MobileNet, ShuffleNet). However, model compression techniques are often regarded as post-processing steps and typically result in a significant drop in model accuracy at high compression rates. On the other hand, while existing lightweight network designs reduce parameters, most focus on reducing redundancy in only a single dimension—either "spatial" or "channel"—leaving the network with insufficient feature utilization.

B. Research Significance

The significance of this research lies in addressing the challenges of deploying deep neural networks in resource-constrained environments, such as mobile devices and embedded systems. Since spatial and channel redundancies in convolutional layers account for the majority of computational costs, effectively and jointly reducing redundancies in both dimensions can break through the bottlenecks of existing lightweight models. This approach not only significantly reduces the model's FLOPs and number of parameters but also further enhances the model's feature representation capabilities and inference accuracy by minimizing the interference of invalid features. Therefore, developing a method that can compress the model while maintaining or even improving performance is of high research value and practical utility.

C. Research Objectives

This project aims to propose an efficient convolution module named SCConv (Spatial and Channel Reconstruction Convolution) to replace standard convolutional layers. The specific objectives of this research are as follows:

1) Joint Reduction of Redundancy:
To design a Spatial Reconstruction Unit (SRU) to suppress redundancy in the spatial dimension, and a Channel Reconstruction Unit (CRU) to eliminate redundancy in the channel dimension. (The detailed mechanisms of these units will be further discussed in the Methodology section.)

2) Plug-and-Play Design:
To ensure that SCConv features a plug-and-play design, allowing it to be directly embedded into various existing CNN architectures (such as ResNet) without requiring additional modifications to the base architecture.

3) Performance Optimization:
To validate that the proposed module can enhance feature representation and achieve performance superior to existing state-of-the-art (SOTA) methods in tasks such as image classification and object detection, all while substantially reducing computational complexity and costs.

II. Related Work

A. Lightweight Convolutional Neural Networks

To deploy deep learning models on resource-constrained devices, various lightweight architectures have been proposed. MobileNets [Ref1] pioneered the concept of compression , which decomposes a standard convolution into a depthwise convolution (DWConv) and a pointwise convolution (PWConv) to reduce both spatial and channel redundancy:

1) Pointwise Convolution (PWConv)

Utilizing a 1x1 kernel in manipulating the channel dimension. Unlike standard convolutions that focus on spatial feature extraction, PWConv's kernel sweeps across the entire depth of the input feature map at every spatial location, effectively performing a linear combination of all input pixels and achieving channel compression.

2) Depthwise Convolution (DWConv)

DWConv applies a *single, unique 2D kernel* to *each* corresponding input channel. This mechanism ensures that features within one channel are processed without mixing information from other channels, reducing the total number of parameters and computation.

While the combination of DWC, PWC significantly reduces the computational cost (FLOPs) and parameters compared to conventional convolutions, they often treat spatial and channel dimensions separately or rely heavily on the assumption that features are fully informative, potentially overlooking the inherent redundancy within the feature maps themselves.

B. Reduction of Spatial Redundancy

Beyond simple factorization, some works focus specifically on the redundancy found in the spatial dimension of feature maps. A representative work is Octave Convolution[Ref3], which addresses the issue of spatial redundancy by mimicking the frequency decomposition of images.

As shown in the literature, OctConv decomposes feature maps into high-frequency and low-frequency groups. Since low-frequency information (representing global structure) changes slowly, it can be processed at a lower spatial resolution, thereby reducing spatial redundancy and computational cost.

1) Limitation:

As noted in the analysis, while OctConv effectively handles spatial resolution, it leaves the question of "Channel Redundancy?" largely unanswered. It focuses on reducing the spatial footprint but treats the channel dimension with standard processing, which may still carry duplicate or invalid feature information.

C. Reduction of Channel Redundancy

Other research aims to mitigate redundancy along the channel dimension. GhostNet (CVPR 2020) [Ref2] observes that successful CNNs often generate many similar feature maps, akin to "ghosts" of one another. Instead of generating all feature maps via expensive convolutions, GhostNet generates a few intrinsic feature maps and then applies cheap linear operations to produce more features, thus exploiting channel redundancy.

1) Limitation:

While GhostNet efficiently creates more features from cheap operations, it primarily targets the channel dimension. The question of "Spatial information?" remains. It does not explicitly design mechanisms to reduce redundancy in the spatial domain (e.g., pixel-level repetition or unnecessary high-resolution processing for simple features) simultaneously.

III. Methodology

In this Section, we introduce the proposed SCConv and how it's used as shown in Fig. 1. As previously mentioned, SCConv consists of 2 parts, Spatial Reconstruction Unit (SRU) and Channel Reconstruction Unit (CRU). We replace all the bottleneck convolutional blocks (usually a 3-by-3 conv2D) in the network with SCConv module to decrease redundancy among intermediate feature maps and boost the feature representation of CNNs.

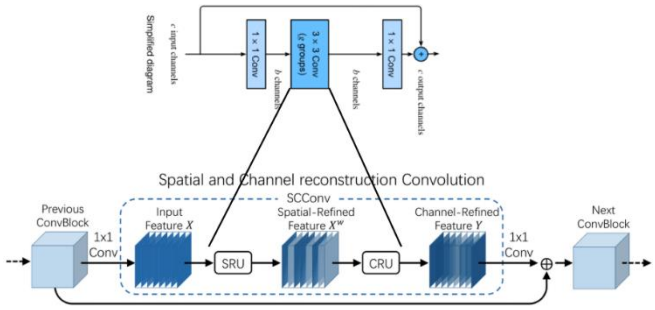

Fig. 1. SCConv overview, this figure shows the exact position of the SCConv module within a ResBlock.

A. SRU for Spatial Redundancy

To exploit the spatial redundancy of features, we introduce Spatial Reconstruction Unit (SRU) as shown in Fig. 2 The high-level idea is to extract features from space without compressing any dimensions. The following demonstrates the step-by-step process of how the SRU component achieves this goal.

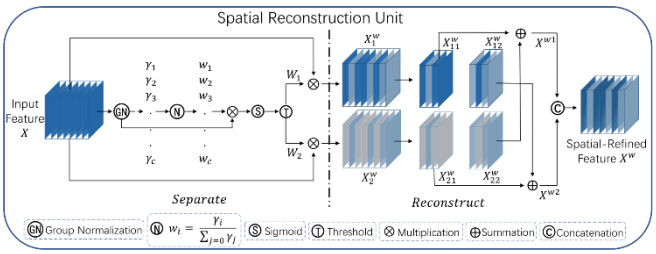

Fig. 2. The architecture of Spatial Reconstruction Unit.

First, we apply group normalization to the original feature maps to get normalized maps called X^N. In this step, we can get trainable parameters γ shown in equation 1, that indicates the level of variation in spatial pixels. The larger the value of γ is, the less we want the map to be normalized, and that implies there is rich spatial information within the feature map.

$$X_{out} = \gamma \frac{X - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \quad (1)$$

After we get C γ, where C is the input channel size, we can normalize the γ to get correlation weights w shown in equation 2.

$$w_i = \frac{\gamma_i}{\sum_{j=1}^{C} \gamma_j} \quad (2)$$

Then, we multiply the w with the normalized maps X^N and apply *Sigmoid* function to each of the pixels to squeeze them into probability distribution. After that, we apply a threshold of 0.5 to turn this $Sigmoid(X^N \times w)$ into 2 complementary binary masks, W^1 and W^2. Concretely, if the value of a pixel is greater than 0.5, we set the corresponding value in W^1 to be 1, value in W^2 to be 0.

After having both W^1 and W^2, we can filter out the original feature maps X to get X_1^w and X_2^w. Then we separate both in half to apply cross summation, which is $X_{11}^w + X_{22}^w$ and $X_{12}^w + X_{21}^w$ to get the last 2 components X^{w1} and X^{w2}. The core idea behind this is to see whether other channels contain complementary information regarding the filtered pixels. For example, if the left corner pixels in the first channel are filtered to 0 in the X_{11}^w, maybe the $(\frac{C}{2} + 1)^{th}$ channel map in X_{22}^w will have different thoughts on the left corner pixels so that we can add them back in X^{w1}.

Finally, we concatenate X^{w1} and X^{w2} together into X^w which has the same dimension as the input X.

B. CRU for Channel Redundancy

To exploit the channel redundancy of features, we introduce Channel Reconstruction Unit (CRU) as shown in Fig. 3 This is the part where most computation and training take place. We first split channels in half and apply simple channel-wise attention mechanism to extract and reconstruct channel features. Similar to the section in SRU, we provide step-by-step explanation to demonstrate how CRU actually works.

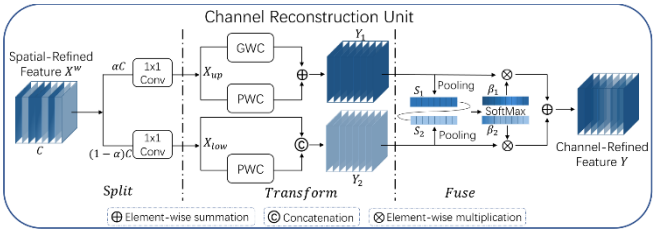

Fig. 3. The architecture of Channel Reconstruction Unit.

First, we split the X^w according to the ratio α (set to 0.5 in the paper) and then use 1-by-1 convolution (PWC) to further squeeze the channel size (squeeze ratio is 2 in the paper). At this point, we have both $\frac{C}{4}$ channels for X_{up} and X_{low}.

Secondly, we make a crucial assumption that the upper part, which is also the front part, of the channels contain richer features than the lower part of the channels contain. That's why we perform both Group-wise convolution and Point-wise convolution for the upper part but only concatenate the X_{low} and the Point-wise convolution result for the lower part.

Notice that the GWC and PWC in this step are designed to reconstruct the channels. Therefore, the number of the kernels should be exactly the same as the output feature map Y's channel size. For example, if the channel size of X_{up} and X_{low} are 16, and the channel size of Y is 48, we need 48 kernels for the GWC and PWC in the upper part. On the other hand, we need (48 - 16) kernels for the PWC in the lower part.

After having Y_1 and Y_2, we start performing channel-wise attention mechanism as shown in equation 3. First, we Global Average Pool these 2 feature maps to get 2 vectors, named S_1 and S_2 with size of C_{output}. Then, we do pairwise *Softmax* to turn them into β_1 and β_2 which are probability vectors $(\beta_1[i] + \beta_2[i] = 1)$.

$$\beta_1 = \frac{e^{S1}}{e^{S1}+e^{S2}}, \beta_2 = \frac{e^{S2}}{e^{S1}+e^{S2}} \quad (3)$$

Finally, multiply the β_1 and β_2 back to the Y_1 and Y_2 and then do element-wise summation to output the final channel-refined feature Y.

C. SCConv

As previously mentioned, the SCConv is designed as a plug-and-play module that can be easily embedded into various existing well designed neural architectures to reduce computation and storage costs. After a series of experiments, the paper found that the spatial-first order (SRU+CRU) achieves the best accuracy and decided to use this as the comparison model against the baseline models. In short, SRU block keeps the dimensions unchanged while CRU block accounts for the majority of training process and alters the output channel size.

IV. EXPERIMENTAL RESULTS

A. Experimental Setup

1) Datasets

*a) CIFAR-100:*The CIFAR-100 dataset consists of 32×32 RGB images distributed across 100 classes. In our experiments, all images were resized to 224×224 to accommodate the input resolution of the backbone models. We recomputed the channel-wise mean and standard deviation from the training set to ensure accurate normalization. Standard data augmentation techniques were applied, including resizing to 224×224, random horizontal flipping, and normalization based on the computed statistics

b) Food-101: The Food-101 dataset contains 512×512 RGB images covering 101 fine-grained food categories, and we utilized the official train/test split for evaluation. The mean and standard deviation were computed specifically from the training split. During training, we employed data augmentation consisting of random resized cropping to 224×224, random horizontal flipping, and normalization. For the testing phase, images were resized to 256×256 followed by a center crop to 224×224 before normalization.

2) Model Variants: We compare four convolutional neural network architectures across all experiments.

a) DenseNet121 (baseline): The standard DenseNet121 architecture serves as the baseline model.

b) SCConv-D121 (DenseNet121 with SCConv) : For the SCConv-D121 variant, each DenseLayer in the DenseNet121 architecture is modified by inserting an SCConv block between the bottleneck 1×1 convolution and the 3×3 convolution. As a result, the layer structure evolves from the original sequence of BN-ReLU-1×1 Conv-BN-ReLU-3×3 Conv to an enhanced configuration where the SCConv module is placed immediately before the 3×3 convolution. All transition layers in the network remain unchanged.

c) ResNet50 (baseline): The standard torchvision implementation is used as the baseline ResNet50 model.

d) SCConv-R50 (ResNet50 with SCConv): In the SCConv-R50 variant, the standard 3×3 convolution within every Bottleneck block is replaced by an SCConv module. This modification alters the block structure from the original sequence of 1×1 convolution, 3×3 convolution, and 1×1 convolution to a new configuration where the SCConv module is positioned between the two 1×1 layers. For bottlenecks requiring spatial downsampling (stride=2), the SCConv module internally utilizes AvgPool2d to align with the resolution change , while all other components, including the shortcut connections and the surrounding 1×1 convolutions, remain unchanged.

3) Training Protocol: Training hyperparameters for both datasets are summarized in Table 1.

TABLE I. TRAINING HYPERPARAMETERS

Setting	CIFAR-100	Food-101
Epochs	200	100
Optimizer	SGD (momentum=0.9, wd=5e-4)	SGD (momentum=0.9, wd=1e-4)
Initial LR	0.05	0.1
LR Schedule	×0.1 at epochs 100 / 150	×0.1 at epochs 30 / 60 / 90
Batch Size (DenseNet / SCConv-D)	64	64
Batch Size (ResNet / SCConv-R)	128	128
Training Augmentations	Resize224, RandomHorizontalFlip, Normalize	RandomResizedCrop224, RandomHorizontalFlip Normalize
Test Augmentation	–	Resize256 → CenterCrop224

4) Evaluation Metrics & Protocol: To comprehensively evaluate model performance and efficiency, we report the Test Accuracy (%) alongside computational metrics including the number of parameters (M) and FLOPs (G). Additionally, we monitor practical resource usage by recording the GPU VRAM consumption (MB) and the total training time (formatted in hh:mm:ss).

B. Overall Quantitative Results

1) Overall Quantitative Results:

TABLE II. CIFAR-100 SUMMARY

Model	Epochs	Best Epoch	Test Acc (%)
DenseNet121	200	174	79.35
SCConv-D121	200	104	76.31
ResNet50	200	141	73.13
SCConv-R50	200	133	73.80

TABLE III. FOOD-101 SUMMARY

Model	Epochs	Best Epoch	Test Acc (%)
DenseNet121	100	67	84.01
SCConv-D121	100	96	83.55
ResNet50	100	98	78.87
SCConv-R50	100	64	77.57

2) Learning Curve Results:

a) CIFAR-100 Learning Curves:

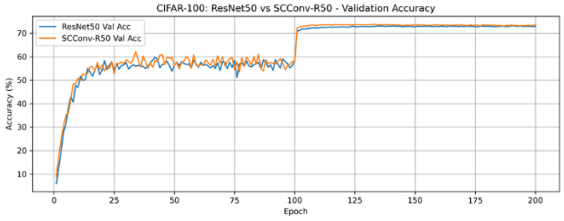

Fig. 4. CIFAR-100 — Validation Accuracy of ResNet50 and SCConv-R50.

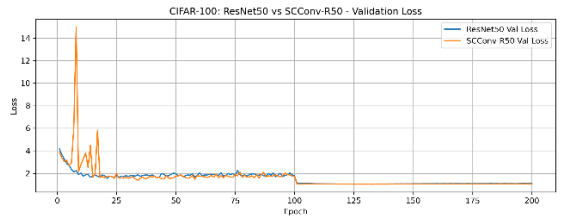

Fig. 5. CIFAR-100 — Validation Loss of ResNet50 and SCConv-R50.

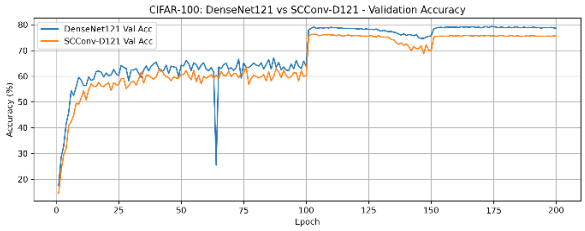

Fig. 6. CIFAR-100 — Validation Accuracy of DenseNet121 and SCConv-D121.

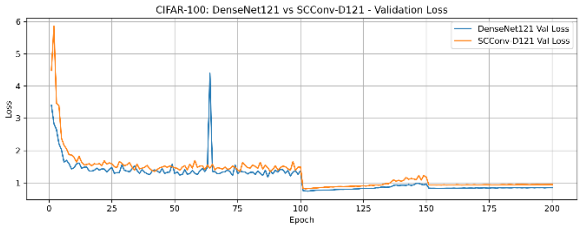

Fig. 7. CIFAR-100 — Validation Loss of DenseNet121 and SCConv-D121.

b) Food-101 Learning Curves:

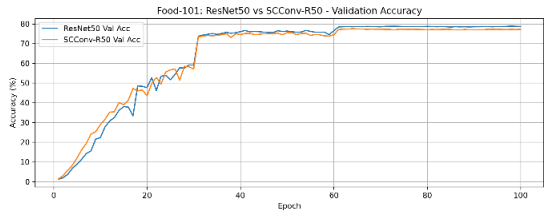

Fig. 8. Food-101 — Validation Accuracy of ResNet50 and SCConv-R50.

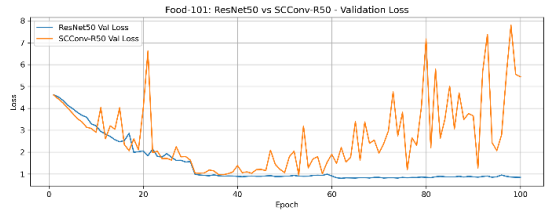

Fig. 9. Food-101 — Validation Loss of ResNet50 and SCConv-R50.

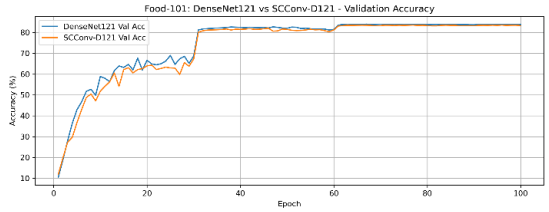

Fig. 10. Food-101 — Validation Loss of DenseNet121 and SCConv-D121.

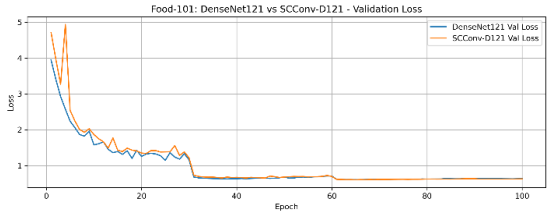

Fig. 11. Food-101 — Validation Accuracy of DenseNet121 and SCConv-D121.

3) Model Complexity and Efficiency: To complement the accuracy results, we also compare each model variant in terms of computational complexity (FLOPs), parameter count, GPU memory usage during training, and total training time. The summary is shown in Table IV. SCConv reduces FLOPs and parameter count for both backbones, consistent with the original paper's goal of reducing feature redundancy. However, both SCConv variants require substantially more VRAM (approximately 40–70% higher) and significantly longer training time (up to 1.5–2× slower), indicating that SCConv is memory-bound rather than compute-bound in our implementation.

TABLE IV. MODEL COMPLEXITY AND TRAINING EFFICIENCY

Model	Params (M)	FLOPs (G)	VRAM	Food-101 Time (100 epoch)	CIFAR-100 Time (200 epoch)
ResNet50	23.71	4.132	12,526 MiB	4:59:50 (h:m:s)	6:37:22 (h:m:s)
SCConv-R50	14.70	2.658	16,216 MiB	7:32:43 (h:m:s)	9:59:44 (h:m:s)
DenseNet121	7.06	2.896	9,576 MiB	6:45:46 (h:m:s)	7:41:45 (h:m:s)
SCConv-D121	5.51	1.983	13,650 MiB	13:47:32 (h:m:s)	16:09:21 (h:m:s)

C. Discussion

Our experiments provide a comprehensive view of how SCConv behaves when integrated into two widely used convolutional backbones. The findings extend the original SCConv paper in several ways.

1) SCConv achieves its intended goal of reducing FLOPs and parameter count.: SCConv consistently achieves its intended goal of reducing model complexity across both architectures. For the ResNet50 backbone, the integration of SCConv decreases the parameter count from 23.7M to 14.7M and FLOPs from 4.132G to 2.658G. Similarly, the SCConv-D121 variant reduces parameters from 7.06M to 5.50M and FLOPs from 2.896G to 1.983G compared to the DenseNet121 baseline. These reductions align with the motivation of SCConv to reconstruct spatial and channel information more efficiently to reduce feature redundancy, thereby supporting the core claims of the original work.

2) Despite lower FLOPs, SCConv increases VRAM usage and training time.: Contrary to the expectation that 'lighter' convolutions should be computationally cheaper in practice, we observe that SCConv variants consume substantially more VRAM and require much longer training time. This reveals that SCConv is memory-bound rather than compute-bound. Multiple feature branches, element-wise fusion operations, and normalization layers all contribute to higher memory usage and slower training. As a result, SCConv lowers theoretical compute cost but increases practical memory overhead and wall-clock training time.

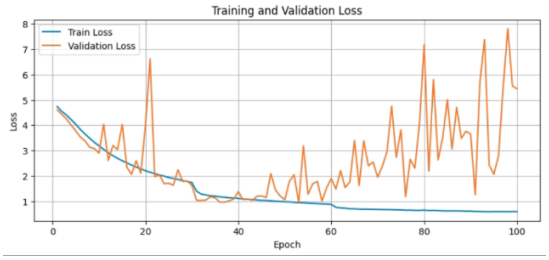

Fig. 12. Food-101 — Training and Validation Loss of SCConv-R50.

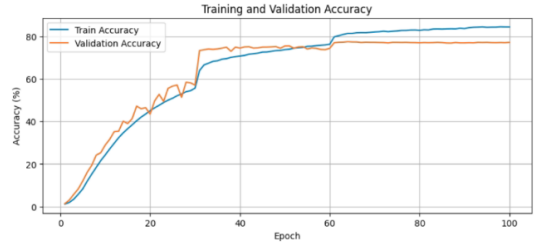

Fig. 13. Food-101 — Training and Validation Accuracy of SCConv-R50.

3) SCConv-R50 exhibits significantly higher training instability on Food-101.: As shown in Fig. 12 and Fig. 13, the SCConv-enhanced ResNet-50 demonstrates a clear discrepancy between training and validation behavior on Food-101. While the training loss decreases smoothly and the training accuracy improves steadily, the validation loss exhibits pronounced fluctuations throughout training, in

contrast to the more stable validation behavior observed in the baseline ResNet-50. Despite these fluctuations, SCConv-R50 ultimately achieves validation accuracy comparable to the baseline model, indicating that the instability is not caused by optimization difficulties during training. Moreover, this behavior persists even after learning-rate tuning, suggesting that it cannot be simply attributed to inappropriate optimization hyperparameters. A plausible explanation is that the cross-scale feature reconstruction mechanism introduced by SCConv may increase the sensitivity of learned representations to variations or perturbations in the validation data distribution. Although the exact cause has not yet been identified, this phenomenon highlights a potential limitation of directly replacing standard 3×3 convolutions with SCConv: while SCConv improves efficiency in terms of FLOPs and parameter count, it may introduce increased generalization sensitivity on fine-grained recognition datasets such as Food-101.

s

REFERENCES

[1] Qin, D., Leichner, C., Delakis, M., Fornoni, M., Luo, S., Yang, F., Wang, W., Banbury, C., Ye, C., Akin, B., Aggarwal, V., Zhu, T., Moro, D., & Howard, A. (2024). *MobileNetV4 - Universal Models for the Mobile Ecosystem*. arXiv preprint arXiv:2404.10518.

[2] Han, K., Wang, Y., Tian, Q., Guo, J., Xu, C., & Xu, C. (2020). GhostNet: More features from cheap operations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 1580-1589).

[3] Chen, Y., Fan, H., Xu, B., Yan, Z., Kalantidis, Y., Rohrbach, M., Yan, S., & Feng, J. (2019). Drop an octave: Reducing spatial redundancy in convolutional neural networks with octave convolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (pp. 3432–3441).