SCConv

Spatial and Channel Reconstruction Convolution for Feature Redundancy

CVPR 2023

111703004 資訊四 林子齊
111703040 資訊四 游宗諺
111703009 資訊四 嚴聲遠
110703066 資訊四 王冠智

AGENDA OVERVIEW

01 Introduction

02 Related Work

03 Methodology

04 Experimental Results

05 Conclusion

Introduction

What issue does this paper address?

The Challenge

- CNNs require intensive computation and storage resources.
- Difficult to deploy on resource-constrained devices.

The Root Cause: Feature Redundancy

- Significant redundancy exists in intermediate feature maps.
- Occurs in both Spatial and Channel dimensions.

SCConv: Spatial and Channel Reconstruction Convolution

Core Concept

- A novel CNN compression method to jointly reduce Spatial and Channel redundancy.

Key Components

- SRU (Spatial Reconstruction Unit): Separates and reconstructs redundant features.
- CRU (Channel Reconstruction Unit): Uses "Split-Transform-and-Fuse" strategy.

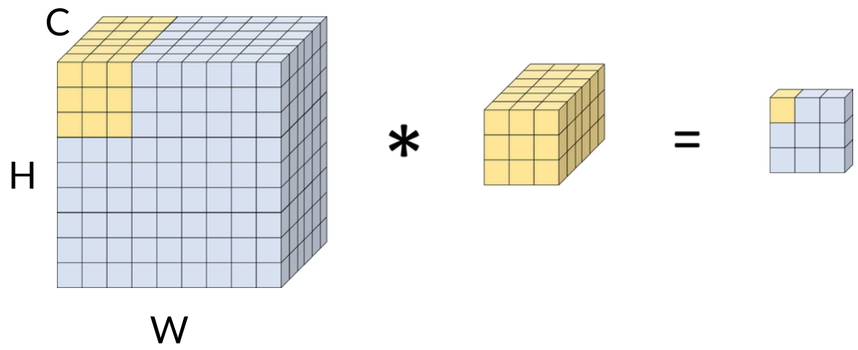
Key Contributions

- Plug-and-Play: Can directly replace standard convolutions (e.g., in ResNet).
- Better Trade-off: Reduces FLOPs/Params while maintaining or improving performance.

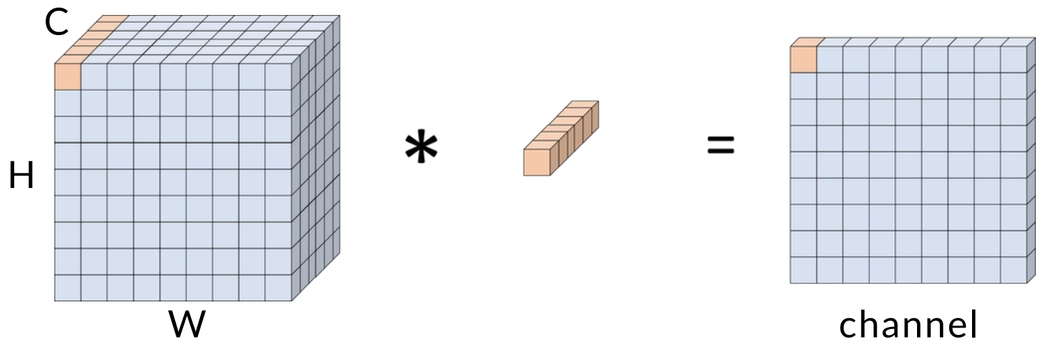
Related Work

The Elemental convolution operations

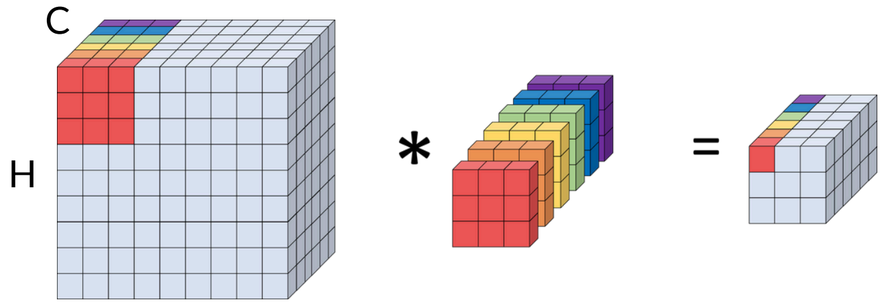
Conventional Convolution



PointWise Convolution (1*1 conv)



DepthWise Convolution

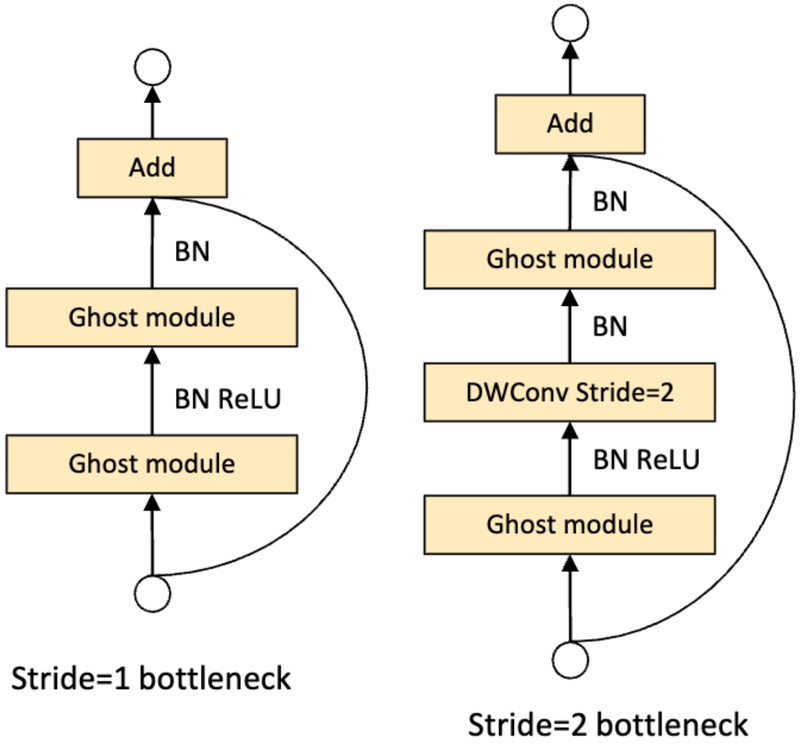


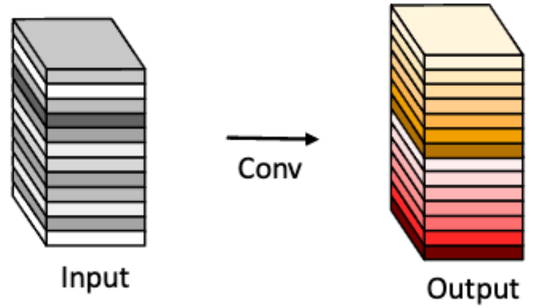
- *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications(Google 2017)*
- PWC → channel compression
- DWC → spatial compression

The current works of CNN

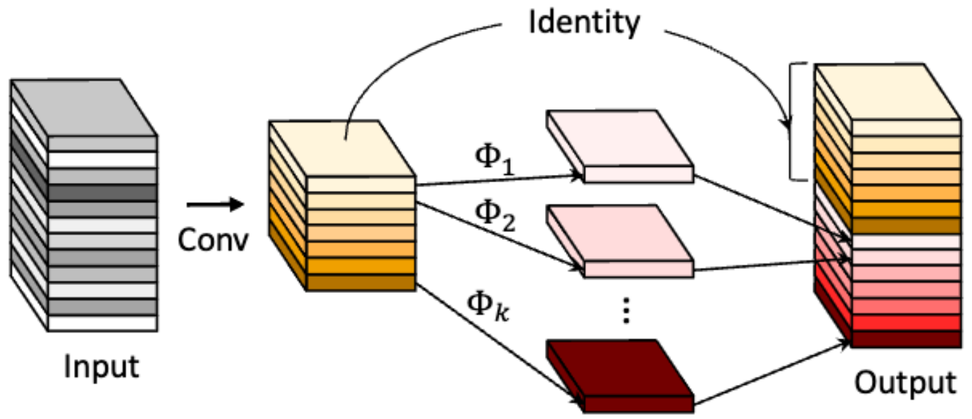
Reference work

- *GhostNet: More features from cheap operations (CVPR 2020)*
- The reduction of channel redundency
- Spatial information?





(a) The convolutional layer.



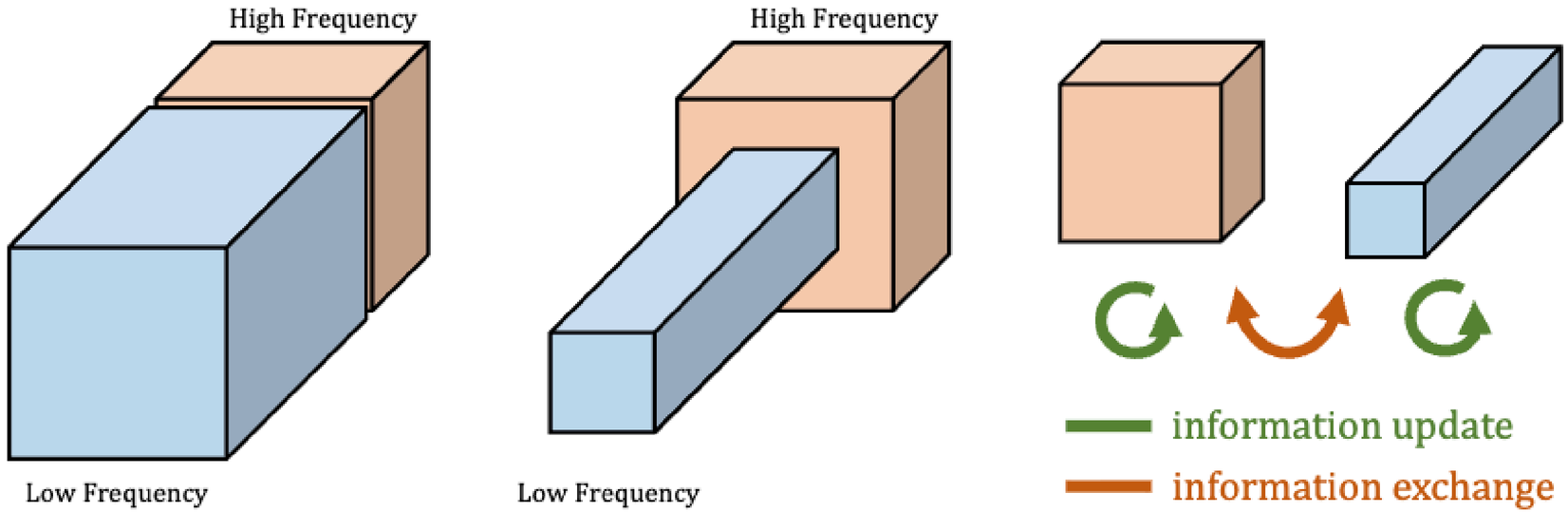
(b) The Ghost module.

The current works of CNN

Reference work

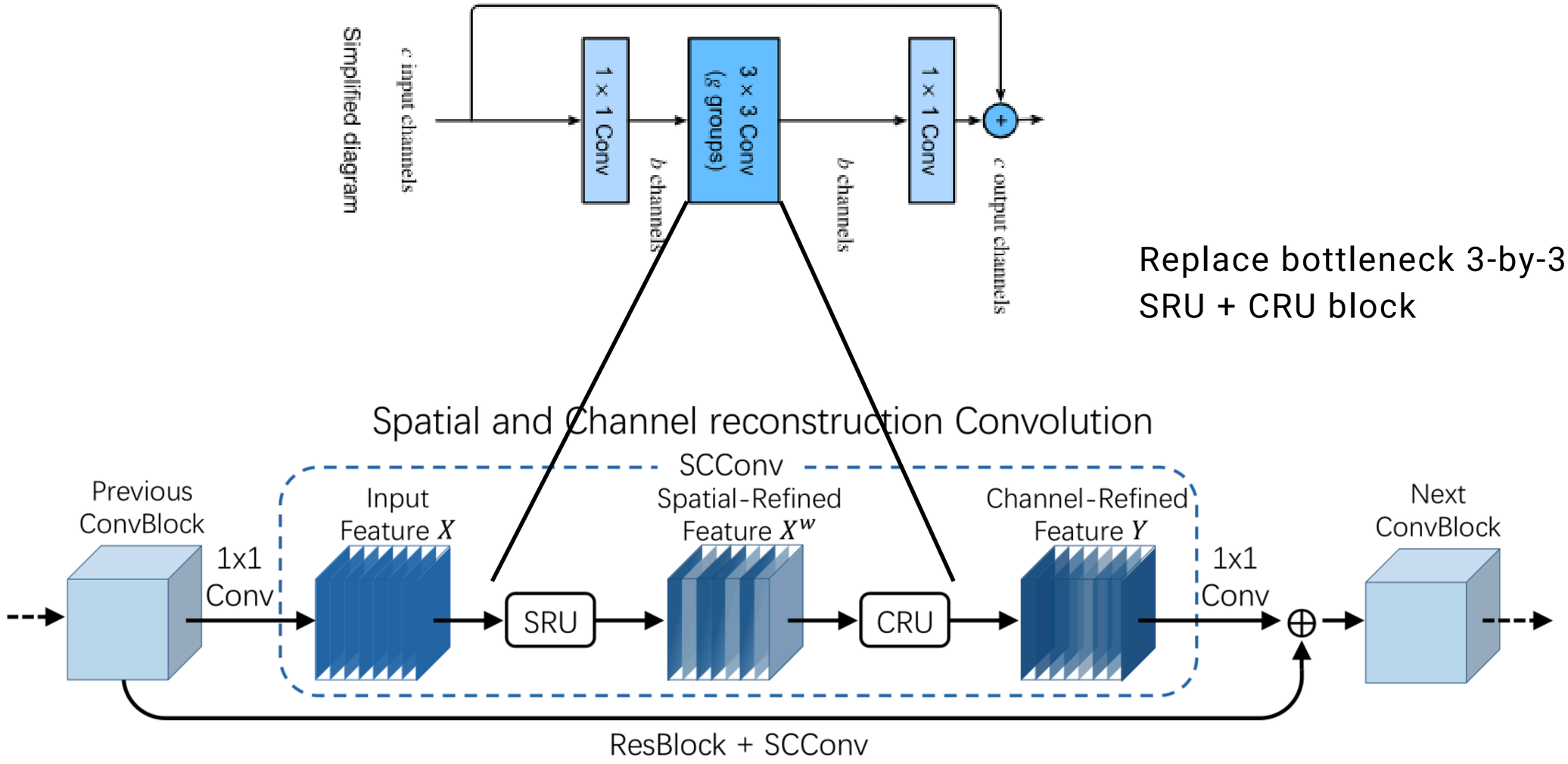
- *Drop an octave: Reducing spatial redundancy in convolutional neural networks (ICCV 2019)*
- *The reduction of spatial redundency: sacrifice a little bit resolution*
- *Channel Redundancy ?*
- *How about the Reconstruction?*

(a) Separating the low and high spatial frequency signal [1, 10].

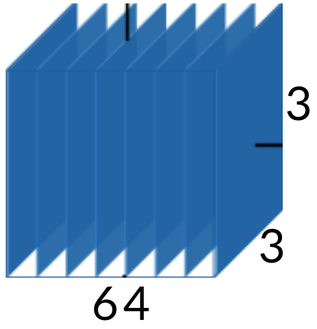


Methodology

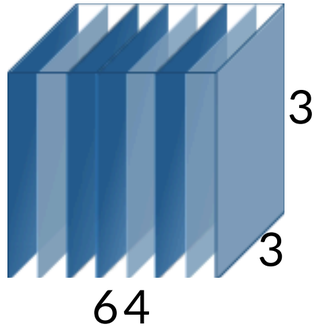
Methodology - Overview



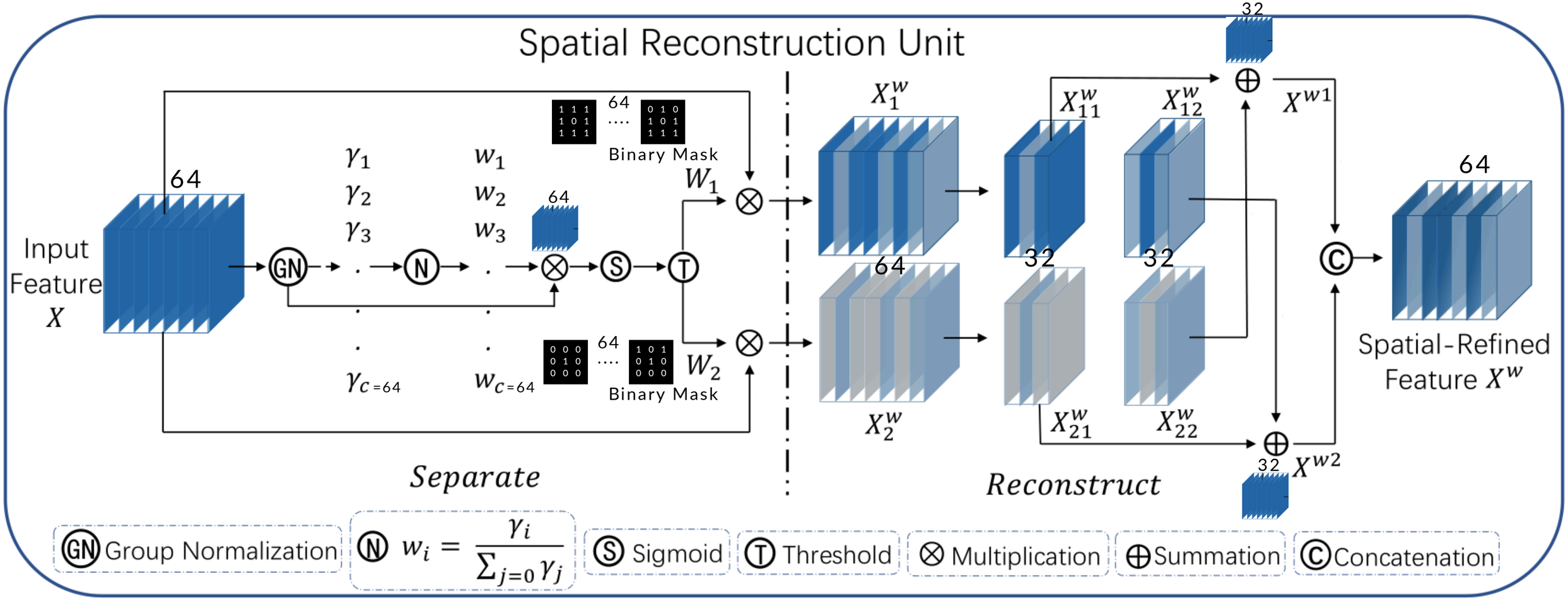
Replace bottleneck 3-by-3 block to SRU + CRU block

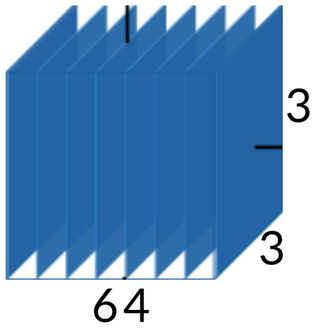


Methodology - SRU



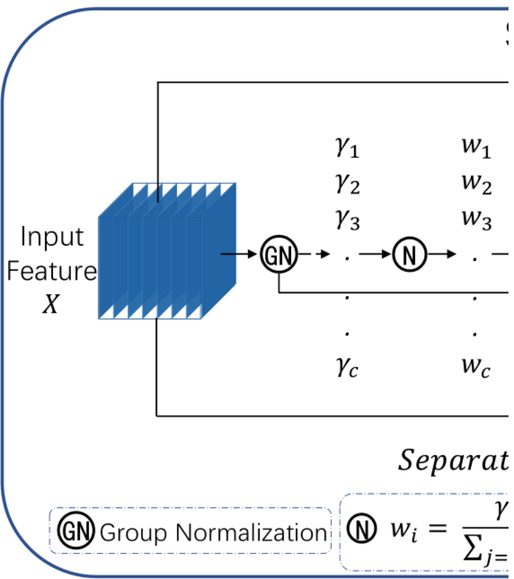
Key: Keep the same dimension for input and output





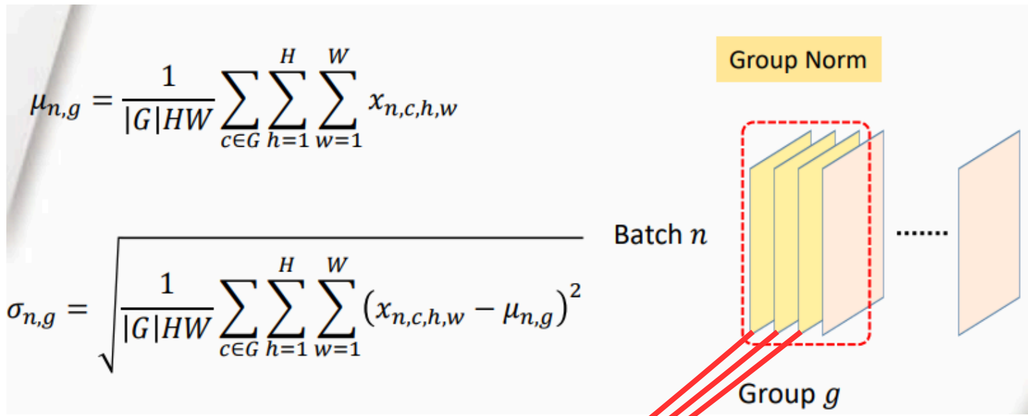
Methodology - SRU

1. After GN, we will have G γs, where G is the number of groups
2. We expand the G γs to C γs, where C is the channel size, by sharing the γ inside those channels that are grouped



After that, we sort of take average on these C γs to get C ws

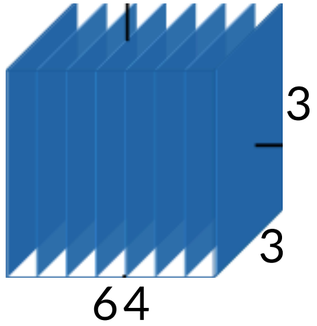
$$\{w_i\} = \frac{\gamma_i}{\sum_{j=1}^{C} \gamma_j}, \; i, j = 1, 2, \cdots, C$$



$$\hat{x} = \frac{x - \mu_{n,g}}{\sqrt{\sigma_{n,g} + \epsilon}}$$

$$y = \gamma \hat{x} + \beta$$

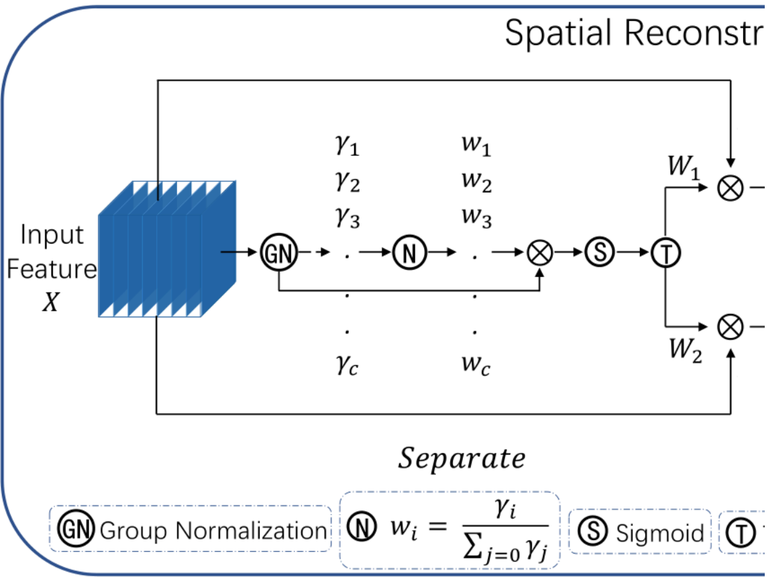
The bigger the γ the more feature is in the **space** (Don't want to normalize).



Methodology - SRU

1. We multiply each w to each channel of the normalized feature map
2. Now we get a new 64 channel feature map, we sigmoid each pixel
3. Gate the values to form 2 binary masks

For Channel 1 as example:



X_1

21	30	10
40	10	21
30	12	00

$GN(X_1)$

0.21	0.30	0.12
0.40	-0.1	0.21
0.30	0.12	0.00

$GN(X_1) \times w_1$

2.1	3.0	1.2
4.0	-1	2.1
3.0	1.2	0.0

$Sigmoid(GN(X_1) \times w_1)$

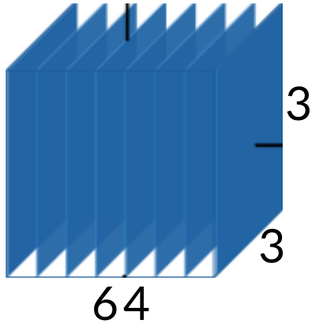
0.7	0.8	0.6
0.9	0.3	0.7
0.8	0.6	0.5

1st channel in W_2

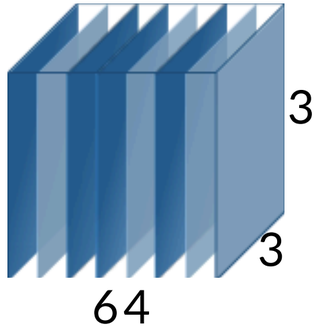
0	0	0
0	1	0
0	0	0

1	1	1
1	0	1
1	1	1

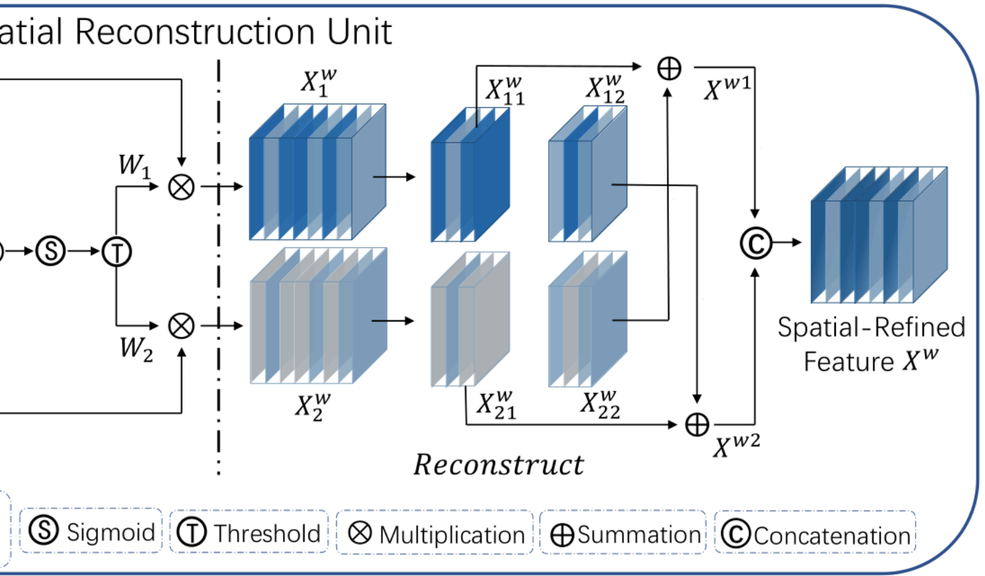
1st channel in W_1



Methodology - SRU



1. We multiply each mask to each channel of the original feature map
2. Now we get another new 128 (64+64) channel feature map, we separate both the upper and lower part in half
3. Cross element-wise summation (maybe other channels have some different thought on the **space** which is masked out)
4. Concatenation



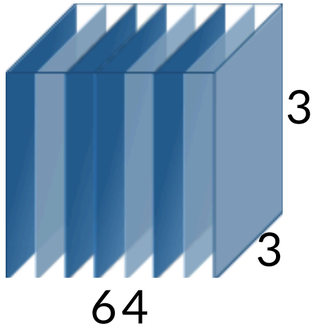
X_1

21	30	10	×	1	1	1	=	21	30	10
40	10	21		1	0	1		40	00	21
30	12	00		1	1	1		30	12	00

add this to lower part of channel 65

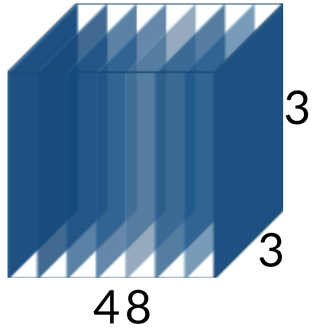
21	30	10	×	0	0	0	=	00	00	00
40	10	21		0	1	0		00	10	00
30	12	00		0	0	0		00	00	00

add this to upper part of channel 65

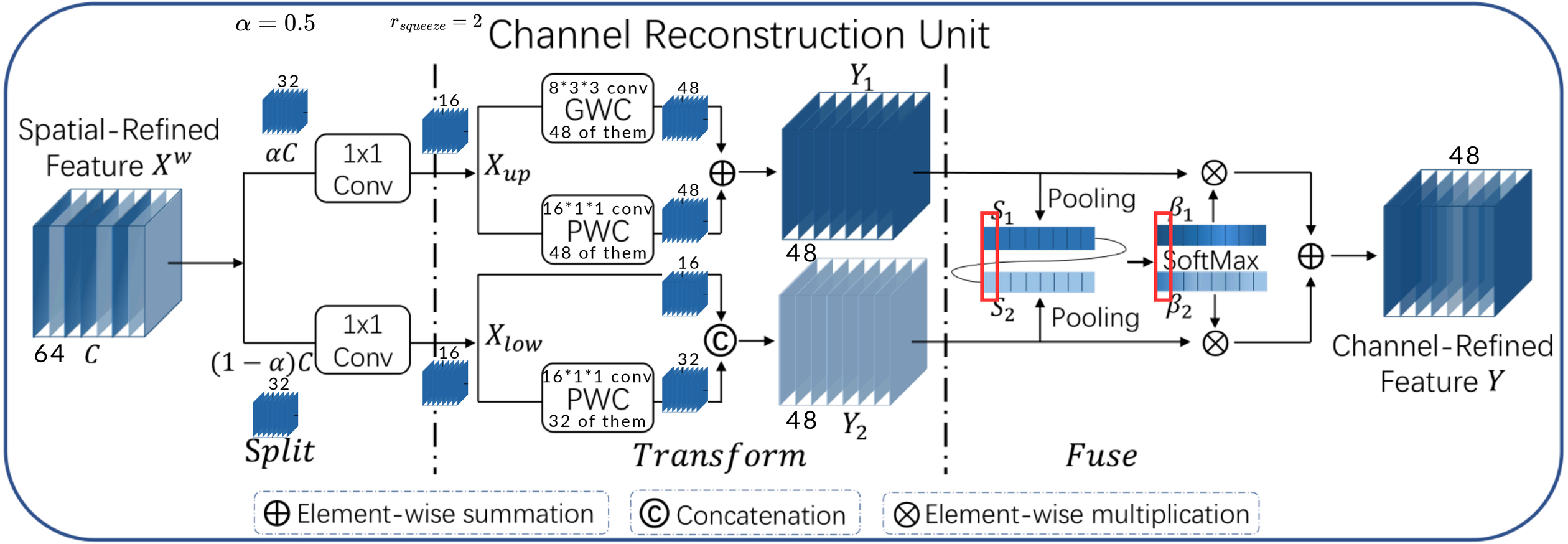


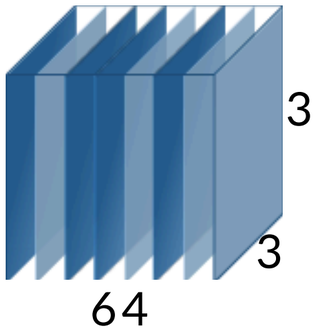
Methodology - CRU

Key: change the channel size here ($\geq \frac{C_{input}}{4}$)



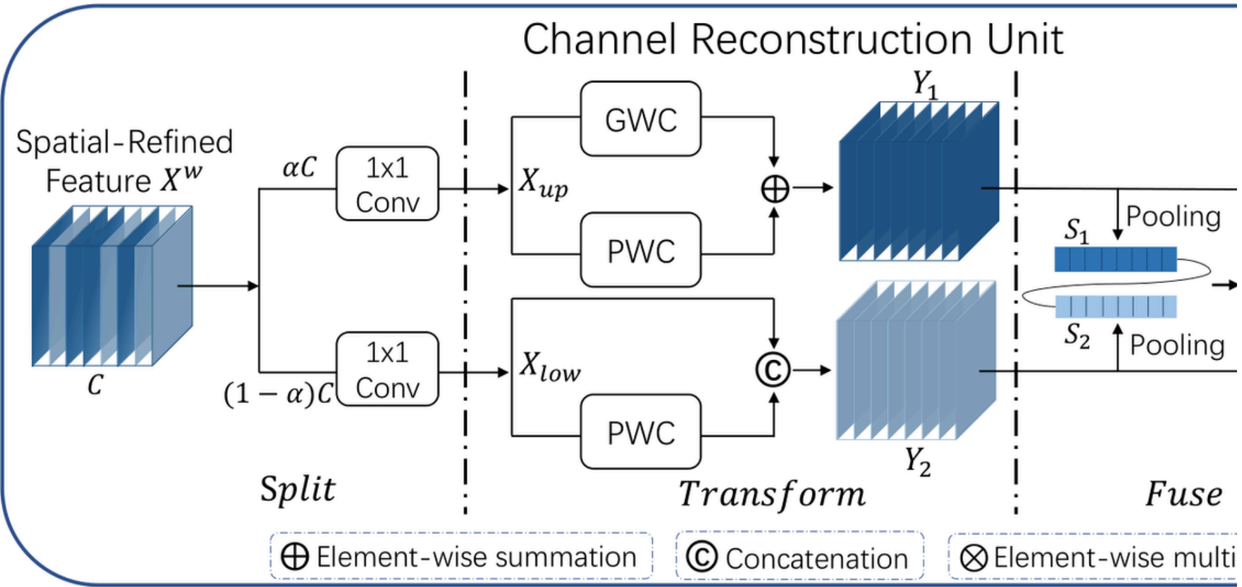
(48 > 16)

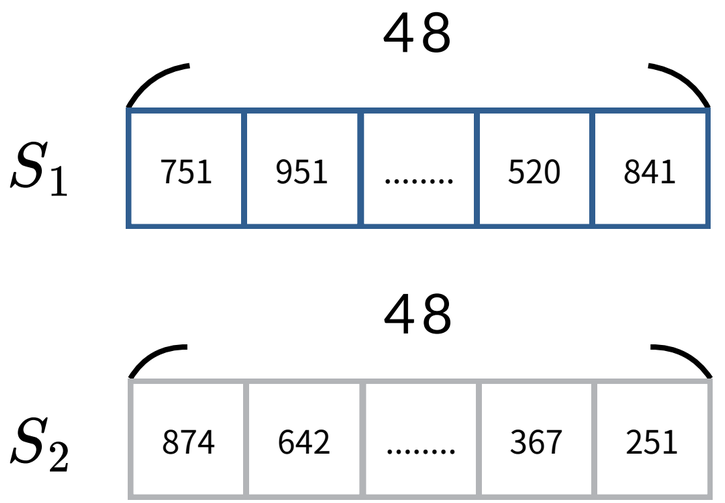


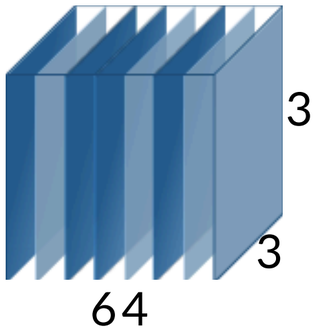


Methodology - CRU

1. α is set to 0.5, squeeze ration of 1x1 Conv is set to 2
2. We have (C_{input} /4 = 16) channels each. Now, we **assume** the first (upper) part has more featured channels and run GWC and PWC on them
3. **Assume** the second (lower) part has less featured channels and only run PWC
4. After that, we have Y_1, Y_2 both have the same channel size (48 in the example)
5. Global pooling to create two 48*1*1 tensors



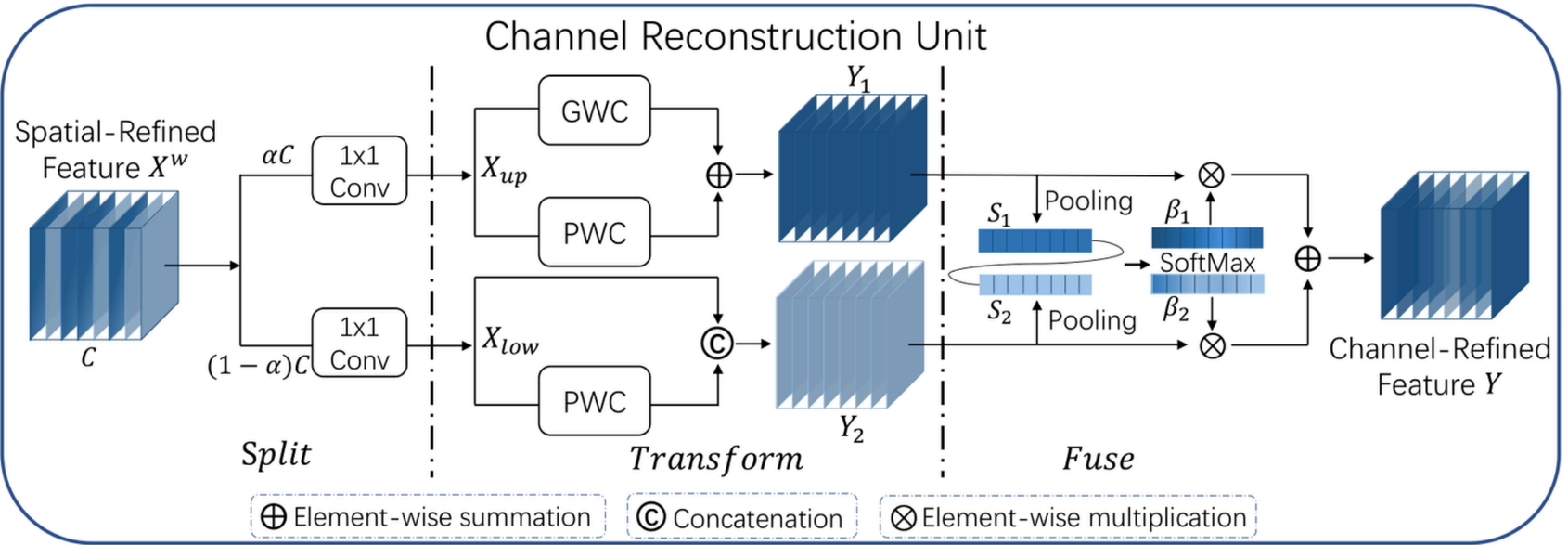




Methodology - CRU

1. Perform **channel-wise** Attention Mechanism
   a. Pair-wise Softmax
   b. Multiply each probability to each channel
   c. Summation

$$\beta_1 = \frac{e^{s_1}}{e^{s_1} + e^{s_2}}, \beta_2 = \frac{e^{s_2}}{e^{s_1} + e^{s_2}}, \beta_1 + \beta_2 = 1$$



S_1

751	951	.....	520	841

S_2

874	642	.....	367	251

multiply with channel 1 from Y_1

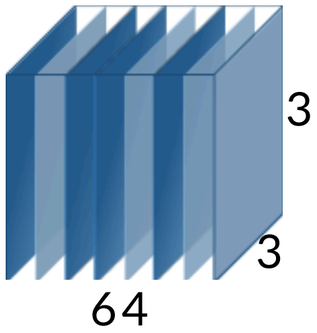
β_1

0.4		.....		

multiply with channel 1 from Y_2

β_2

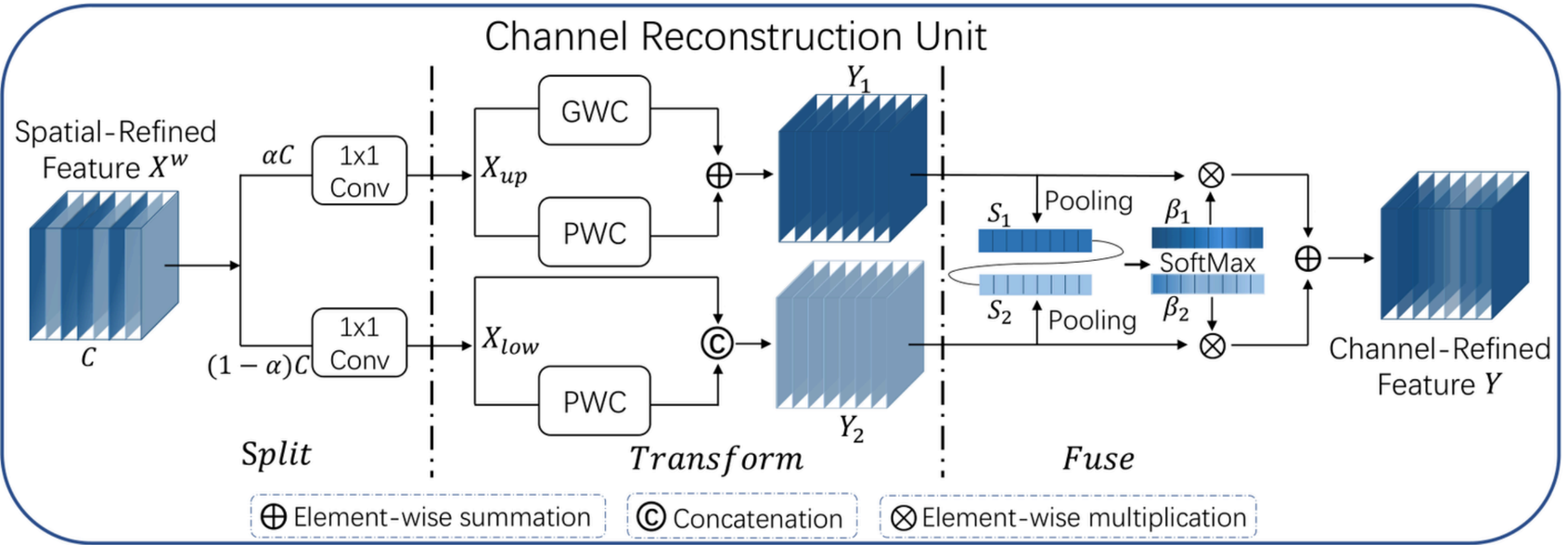
0.6		.....		



Methodology - CRU

1. Perform **channel-wise** Attention Mechanism
   a. Pair-wise Softmax
   b. Multiply each probability to each channel
   c. Summation

$$\beta_1 = \frac{e^{s_1}}{e^{s_1} + e^{s_2}}, \beta_2 = \frac{e^{s_2}}{e^{s_1} + e^{s_2}}, \beta_1 + \beta_2 = 1$$



S_1

751	951	.....	520	841

S_2

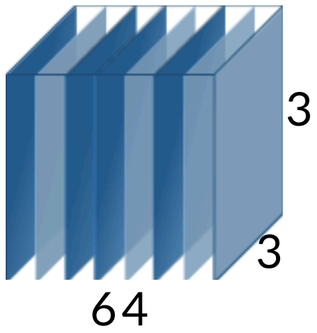
874	642	.....	367	251

β_1

0.4	0.8	.....		

β_2

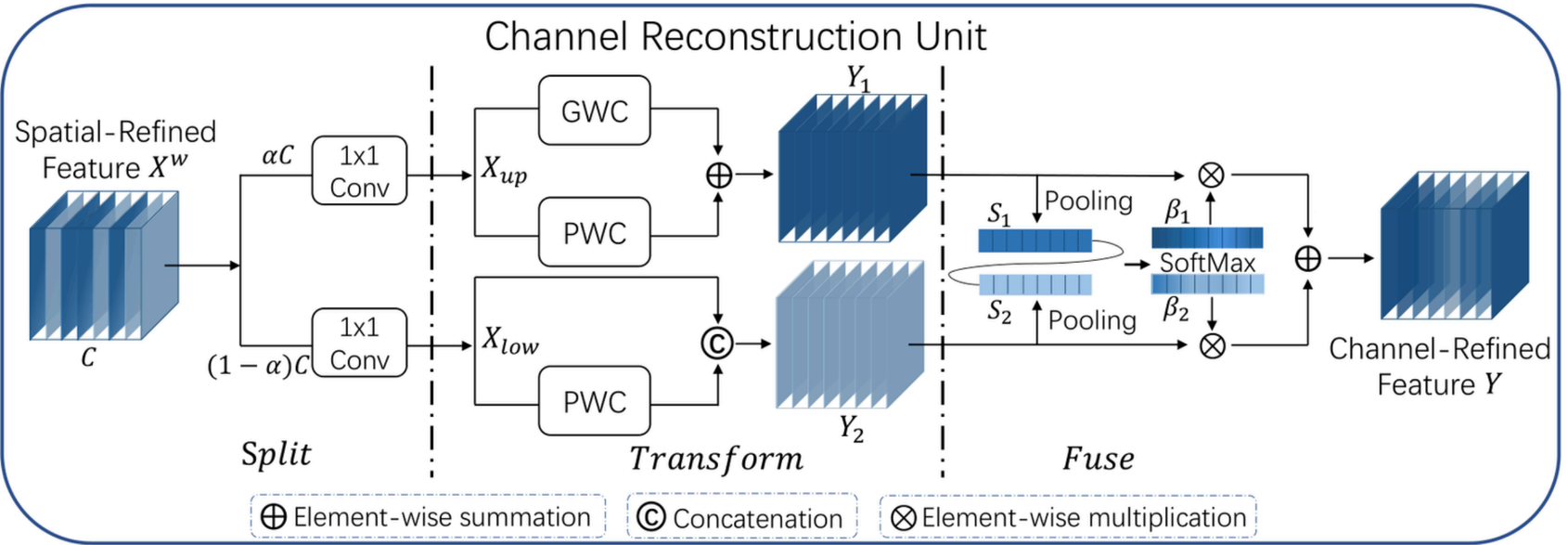
0.6	0.2	.....		



Methodology - CRU

1. Perform **channel-wise** Attention Mechanism
   a. Pair-wise Softmax
   b. Multiply each probability to each channel
   c. Summation

$$\beta_1 = \frac{e^{s_1}}{e^{s_1} + e^{s_2}}, \beta_2 = \frac{e^{s_2}}{e^{s_1} + e^{s_2}}, \beta_1 + \beta_2 = 1$$



S_1

751	951	.....	520	841

S_2

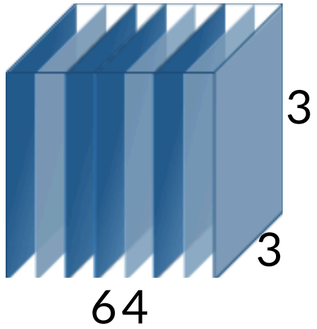
874	642	.....	367	251

β_1

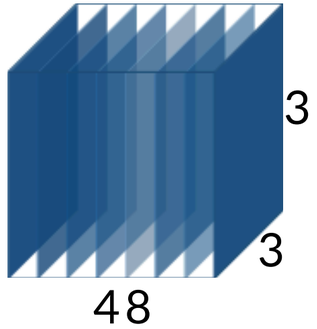
0.4	0.8	.....	0.7	

β_2

0.6	0.2	.....	0.3	

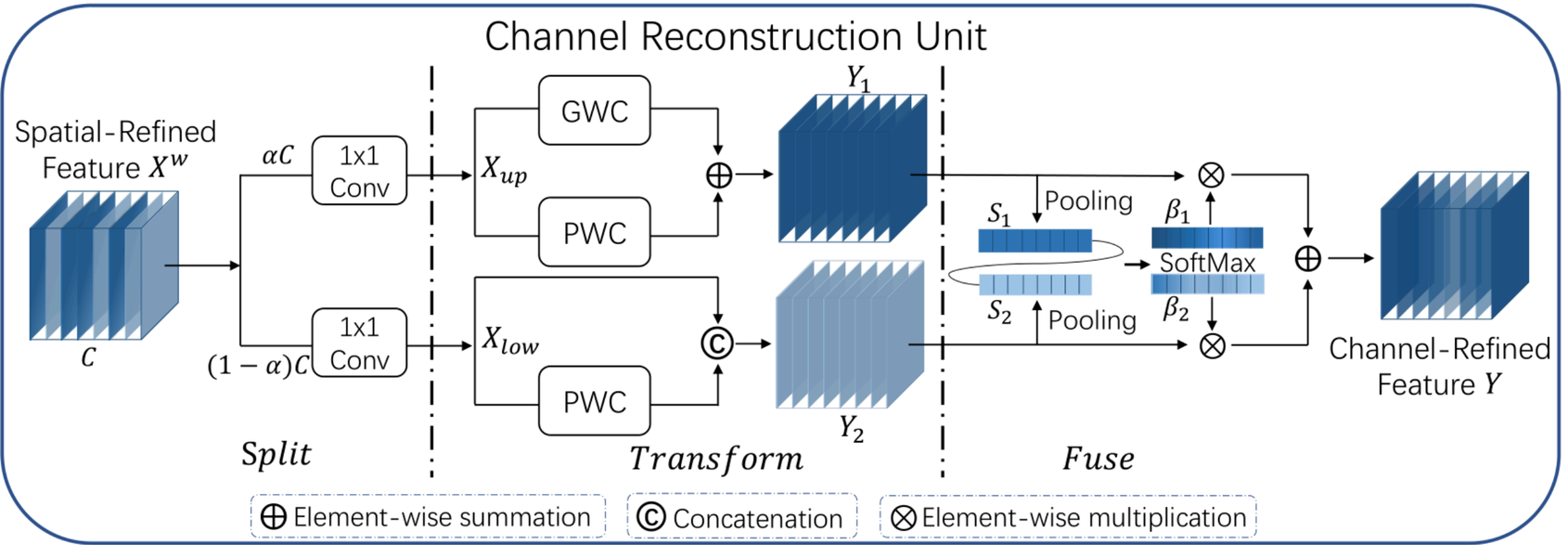


Methodology - CRU



1. Perform **channel-wise** Attention Mechanism
   a. Pair-wise Softmax
   b. Multiply each probability to each channel
   c. Summation

$$\beta_1 = \frac{e^{s_1}}{e^{s_1} + e^{s_2}}, \beta_2 = \frac{e^{s_2}}{e^{s_1} + e^{s_2}}, \beta_1 + \beta_2 = 1$$



S_1

751	951	.....	520	841

S_2

874	642	.....	367	251

β_1

0.4	0.8	.....	0.7	0.9

β_2

0.6	0.2	.....	0.3	0.1

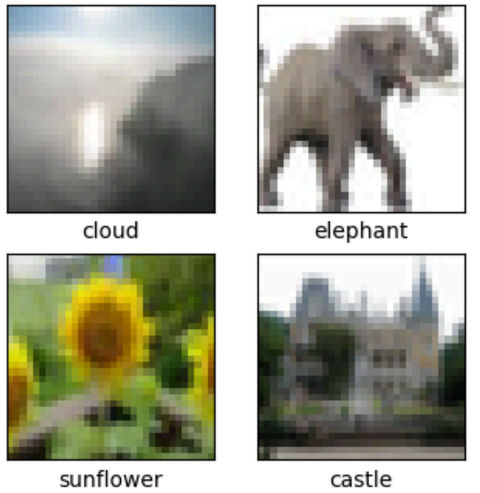
Experiment

Experiment Setup

Dataset

1. CIFAR-100

- The dataset contains 100 classes of 32×32 color images.
- All experiments rescale the images to 224×224 and recompute the channel-wise mean and standard deviation from the CIFAR-100 training set to serve as normalization parameters.



2. Food-101

- This dataset consists of 101 categories of food images, each with a resolution of 512×512, using the official train/test split.
- We first compute the mean and standard deviation on the training split, and apply ImageNet-style augmentation (RandomResizedCrop, RandomHorizontalFlip, Normalize).

Experiment Setup

We compare four convolutional neural network architectures in this study:

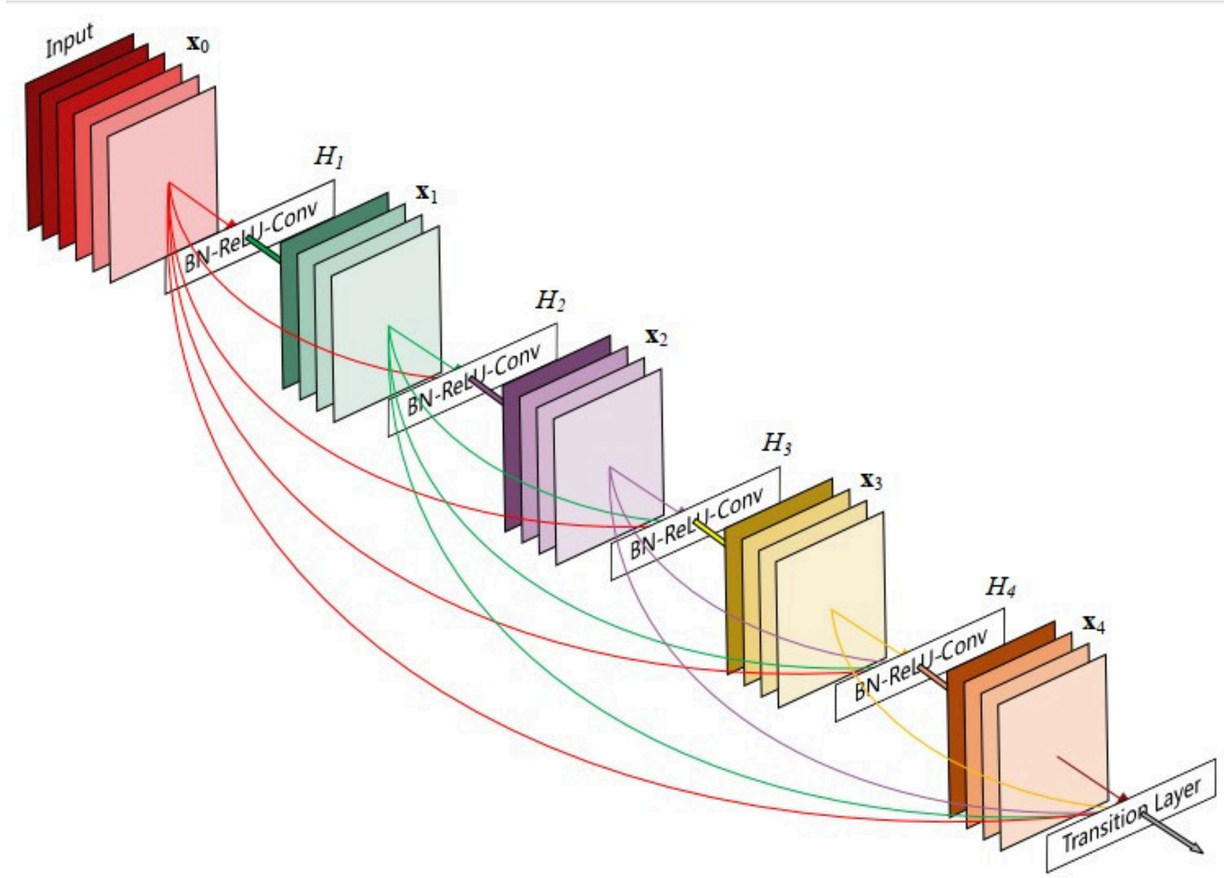
(1) DenseNet121 (baseline)

The original DenseLayer transformation is:
BN → ReLU → 1×1 Conv → BN → ReLU → 3×3 Conv

(2) SCConv-D121 (DenseNet121 with SCConv)

In this variant, an SCConv module is inserted into every DenseLayer of DenseNet121.

The SCConv-enhanced version becomes:
BN → ReLU → 1×1 Conv → BN → ReLU → **SCConv**



Experiment Setup

We compare four convolutional neural network architectures in this study:

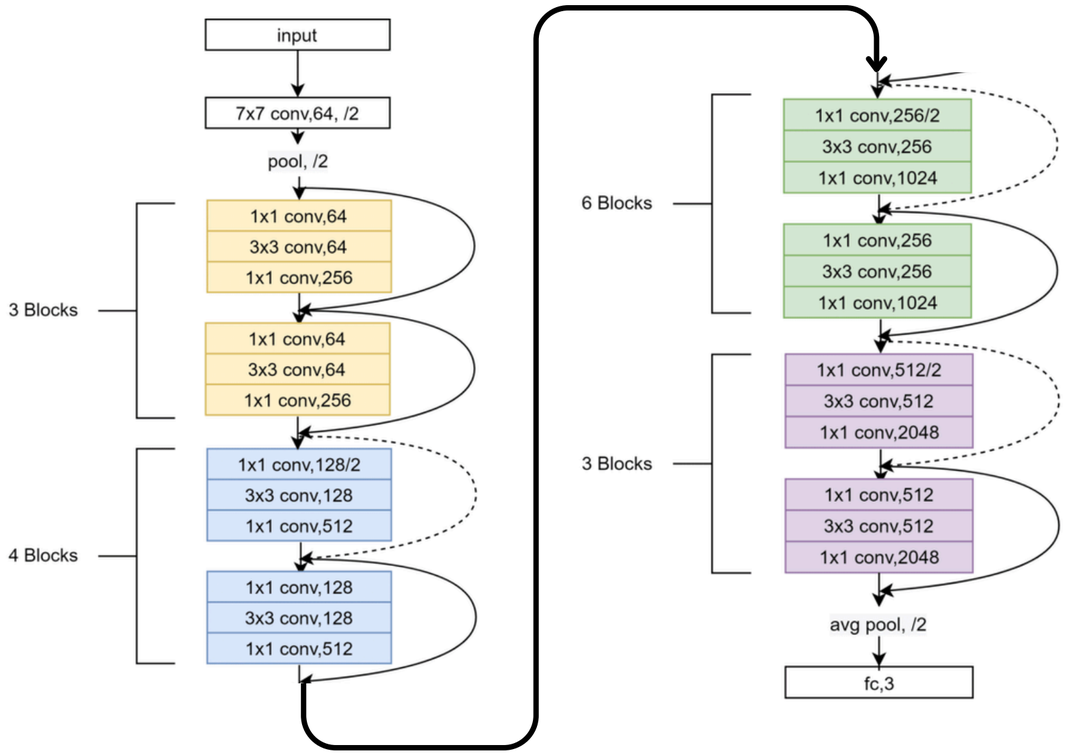
(3) ResNet50 (baseline)

Conv1 (1×1) → Conv2 (3×3) → Conv3 (1×1)

(4) SCConv-R50 (ResNet50 with SCConv)

For this model, the 3×3 convolution inside every Bottleneck block is replaced by an SCConv module:

Modified: Conv1 (1×1) → **SCConv** → Conv3 (1×1)



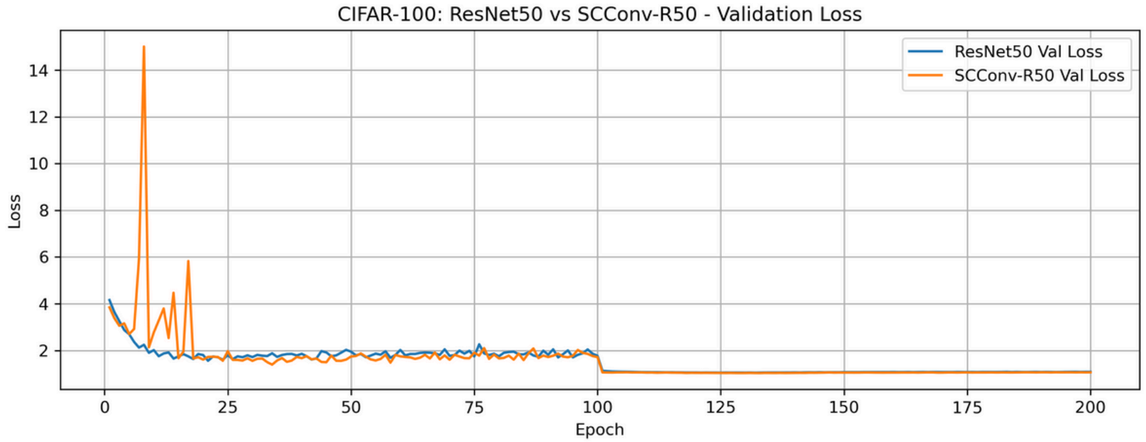
Experiment Setup

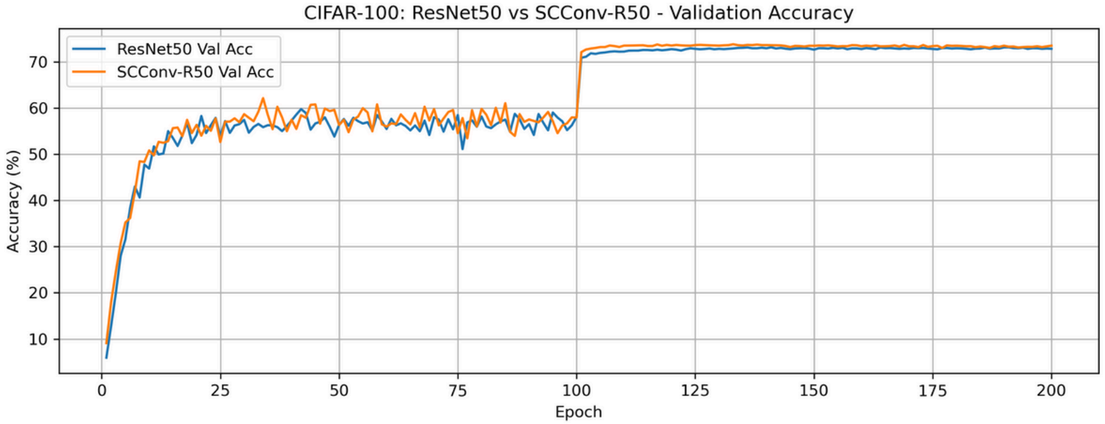
Setting	CIFAR-100	Food-101
Epochs	200	100
Optimizer	SGD (momentum=0.9, wd=5e-4)	SGD (momentum=0.9, wd=1e-4)
Initial LR	0.05	0.1
LR Schedule	×0.1 at epochs 100 / 150	×0.1 at epochs 30 / 60 / 90
Batch Size (DenseNet / SCConv-D)	64	64
Batch Size (ResNet / SCConv-R)	128	128
Training Augmentations	Resize224,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

Experiment Result

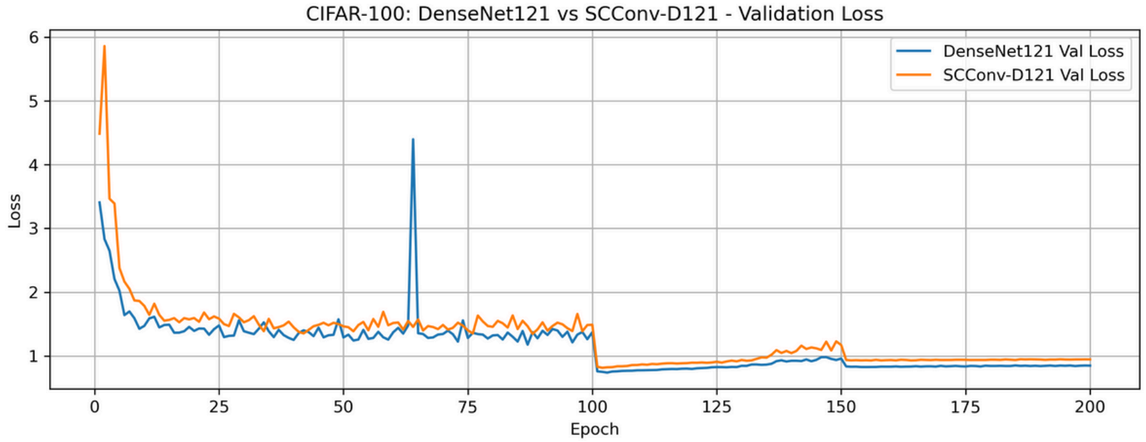
1. CIFAR-100

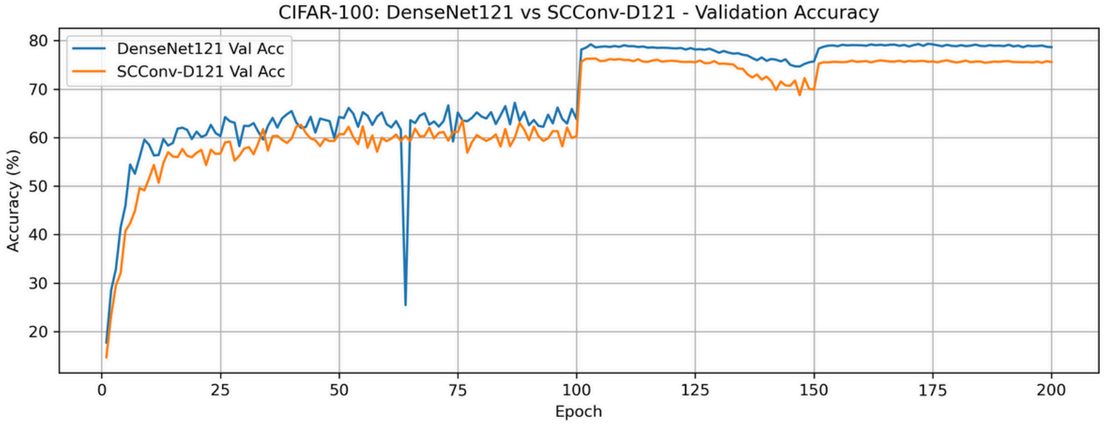
(1) DenseNet121 vs DenseNet121+SCConv





(2) ResNet50 v.s. ResNet50 + SCConv

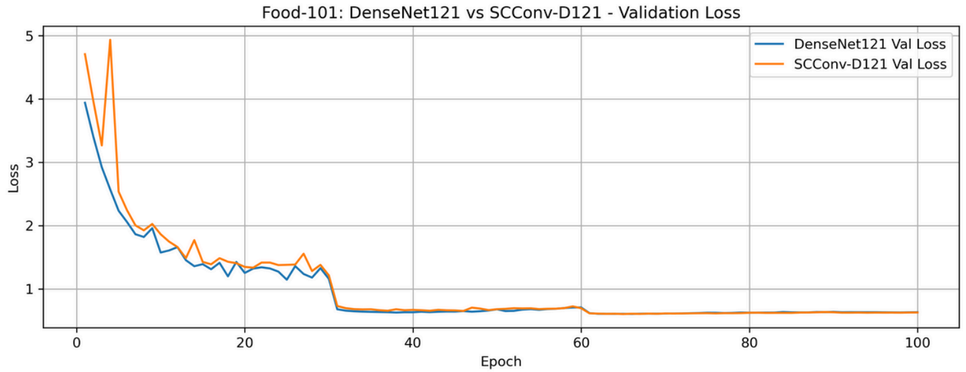


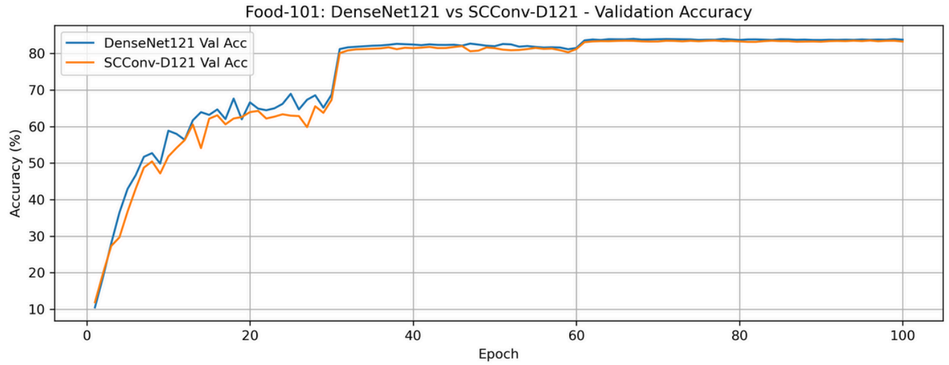


Experiment Result

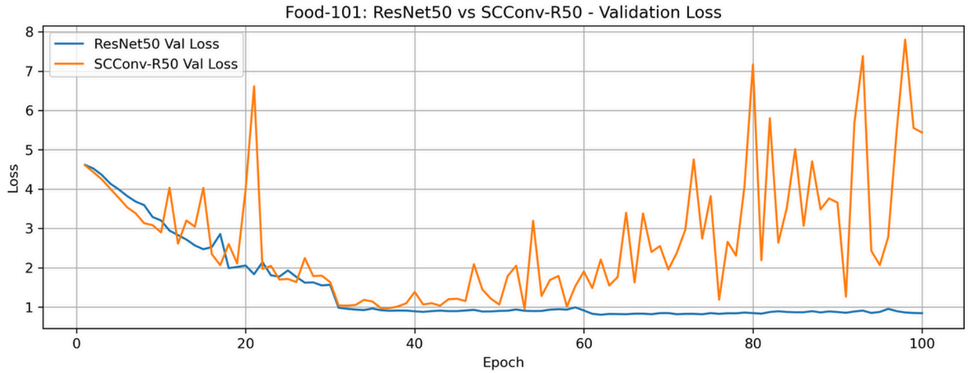
2. Food-101

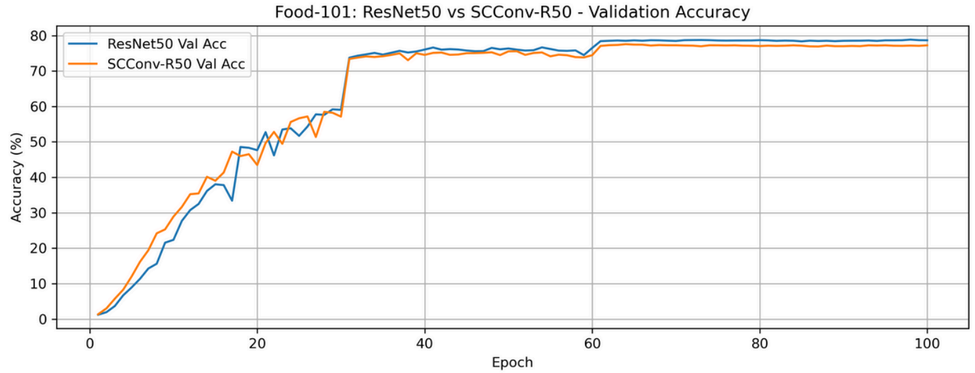
(1) DenseNet121 vs DenseNet121+SCConv





(2) ResNet50 v.s. ResNet50 + SCConv





CIFAR-100

Model	Total 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	73.8

Food-101

Model	Total 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

Model	Params (M)	FLOPs (G)	Food-101 Time (100 epoch)	VRAM	CIFAR-100 Time (200 epoch)
ResNet50	23.71	4.132	4:59:50 (h:m:s)	12,526 MiB	6:37:22 (h:m:s)
SCConv-R50	**14.7**	**2.658**	7:32:43 (h:m:s)	**16,216 MiB**	**9:59:44 (h:m:s)**
DenseNet121	7.06	2.896	6:45:46 (h:m:s)	9,576 MiB	7:41:45 (h:m:s)
SCConv-D121	**5.51**	**1.983**	13:47:32 (h:m:s)	**13,650 MiB**	**16:09:21 (h:m:s)**

Discussion

(1) SCConv achieves its intended goal of reducing FLOPs and parameter count.

Across both architectures, SCConv variants consistently reduce model complexity:
These reductions align with the SCConv paper's motivation to reconstruct spatial and channel information more efficiently and to reduce feature redundancy. In this sense, our results support the core claims of the original work.

Model	Params (M)	FLOPs (G)
ResNet50	**23.71**	**4.132**
SCConv-R50	**14.7**	**2.658**
DenseNet121	**7.06**	**2.896**
SCConv-D121	**5.51**	**1.983**

Food-101

Model	Total 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**

CIFAR-100

Model	Total 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	**73.8**

Discussion

(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.

Model	Params (M)	FLOPs (G)	Food-101 Time (100 epoch)	VRAM	CIFAR-100 Time (200 epoch)
ResNet50	**23.71**	**4.132**	4:59:50 (h:m:s)	12,526 MiB	6:37:22 (h:m:s)
SCConv-R50	**14.7**	**2.658**	7:32:43 (h:m:s)	**16,216 MiB**	**9:59:44 (h:m:s)**
DenseNet121	**7.06**	**2.896**	6:45:46 (h:m:s)	9,576 MiB	7:41:45 (h:m:s)
SCConv-D121	**5.51**	**1.983**	13:47:32 (h:m:s)	**13,650 MiB**	**16:09:21 (h:m:s)**

Conclusion

Although SCConv is theoretically attractive as a plug-and-play replacement for 3×3 convolutions, our empirical study shows that **it does not consistently improve accuracy**, reduces FLOPs and parameters at the cost of **higher memory consumption and slower training**, and behaves in a highly architecture- and dataset-dependent manner.

To fully benefit from SCConv, additional architectural tuning or selective placement may be required. Possible directions for future work include using SCConv only in selected layers, exploring kernel fusion strategies to mitigate memory-bound behavior, combining SCConv with lightweight architectures, or designing SCConv variants explicitly optimized for training efficiency on modern GPUs.

In summary, SCConv successfully reduces theoretical computational cost but introduces substantial memory overhead, which can slow training and degrade final accuracy. Its effectiveness is strongly dependent on the backbone architecture and dataset, and naïve plug-and-play usage may be insufficient for achieving practical efficiency gains.

參考論文

- SCConv: Spatial and Channel Reconstruction Convolution for Feature Redundancy(CVPR 2023)
- Drop an octave: Reducing spatial redundancy in convolutional neural networks (ICCV 2019)
- GhostNet: More features from cheap operations (CVPR 2020)
- MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications(Google 2017)