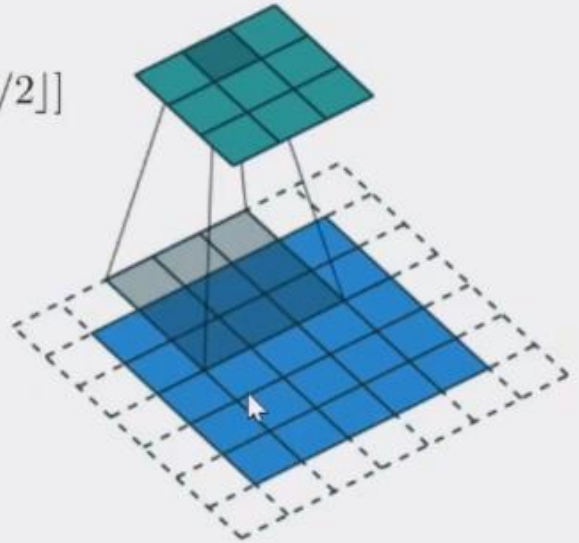
Involution: Inverting the Inherence of Convolution for Visual Recognition

Involution: Inverting the Inherence of Convolution for Visual Recognition

Convolution for Visual Recognition

- Convolution operation: $\mathbf{Y}_{i,j,k} = \sum_{c=1}^{C_i} \sum_{(u,v)\in\mathbf{\Delta}_K} \mathcal{F}_{k,c,u+\lfloor K/2\rfloor,v+\lfloor K/2\rfloor} \mathbf{X}_{i+u,j+v,c}$
  - input: $\mathbf{X} \in \mathbb{R}^{H\times W\times C_i}$
  - output: $\mathbf{Y} \in \mathbb{R}^{H\times W\times C_o}$
  - convolution kernel: $\mathcal{F} \in \mathbb{R}^{C_o\times C_i\times K\times K}$
  - receptive Field: $\mathbf{\Delta}_K = [-\lfloor K/2\rfloor, \cdots, \lfloor K/2\rfloor] \times [-\lfloor K/2\rfloor, \cdots, \lfloor K/2\rfloor]$

- Two inherent properties:
  - spatial-agnostic: same kernel for different positions
  - channel-specific: different kernels for different channels



Involution: Inverting the Inherence of Convolution for Visual Recognition

Principles of convolution

- spatial-agnostic:
  - pros: parameter efficiency, translation equivalence
  - cons: inflexible kernel weight, limited spatial span
- channel-specific:
  - pros: information encoding
  - cons: inter-channel redundancy

Long-range and self-adaptive relationship modeling is desired, provided that computational efficiency is preserved.

Involution: Inverting the Inherence of Convolution for Visual Recognition

Involution for Visual Recognition

- Two inverted properties from convolution:
  - spatial-specific: kernel privatized for different positions
  - channel-agnostic: kernel shared across different channels
- involution operation: $\mathbf{Y}_{i,j,k} = \sum_{(u,v)\in\Delta_K} \mathcal{H}_{i,j,u+\lfloor K/2\rfloor,v+\lfloor K/2\rfloor,\lceil kG/C\rceil}\mathbf{X}_{i+u,j+v,k}$
  - involution kernel: $\mathcal{H} \in \mathbb{R}^{H\times W\times K\times K\times G}$
  - #groups: G
- kernel generated based on input feature map $\mathcal{H}_{i,j} = \phi(\mathbf{X}_{\Psi_{i,j}})$

  -> kernel size aligned with the input tensor size

Involution: Inverting the Inherence of Convolution for Visual Recognition

A Simple Yet Effective Formulation

- kernel generated from **a single pixel** with the **bottleneck structure**

$$\Psi_{i,j} = \{(i,j)\} \qquad \mathcal{H}_{i,j} = \phi(\mathbf{X}_{i,j}) = \mathbf{W}_1\sigma(\mathbf{W}_0\mathbf{X}_{i,j})$$

  - linear transform

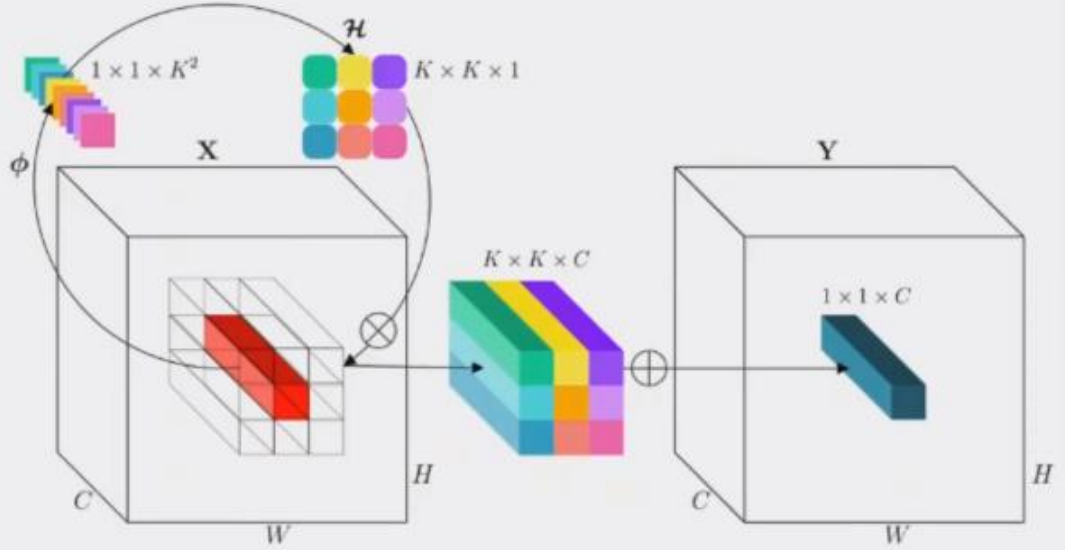
$$\mathbf{W}_0 \in \mathbb{R}^{\frac{C}{r}\times C} \quad \mathbf{W}_1 \in \mathbb{R}^{(K\times K\times G)\times\frac{C}{r}}$$

  - non-linearity

$$\sigma = \mathrm{ReLU}(\mathrm{BN}(\cdot))$$

  - channel-to-space reshape

$$1\times 1\times K^2G \rightarrow K\times K\times G$$



Involution: Inverting the Inherence of Convolution for Visual Recognition

$$\mathbf{X}_{i,j}: 1\times 1\times C \xrightarrow{FC} 1\times 1\times \frac{C}{r} \xrightarrow{FC} 1\times 1\times (K^2G) \xrightarrow{reshape} K\times K\times G \xrightarrow{MAdd} \mathbf{Y}_{i,j}: 1\times 1\times C$$

$$\mathbf{X}_{\Omega_{i,j}}: K\times K\times C \quad (MAdd \nearrow)$$

```python
import torch
import torch.nn as nn


class Involution(nn.Module):
    def __init__(self, channels, kernel_size=7, stride=1, group_channels=16, reduction_ratio=4):
        super().__init__()
        assert not (channels % group_channels or channels % reduction_ratio)

        # in_c=out_c
        self.channels = channels
        self.kernel_size = kernel_size
        self.stride = stride

        # 每组多少个通道
        self.group_channels = group_channels
        self.groups = channels // group_channels

        # reduce channels
        self.reduce = nn.Sequential(
            nn.Conv2d(channels, channels // reduction_ratio, 1),
            nn.BatchNorm2d(channels // reduction_ratio),
            nn.ReLU()
        )
```

Involution: Inverting the Inherence of Convolution for Visual Recognition

$$\mathbf{X}_{i,j}: 1\times 1\times C \xrightarrow{FC} 1\times 1\times \frac{C}{r} \xrightarrow{FC} 1\times 1\times (K^2G) \xrightarrow{reshape} K\times K\times G \xrightarrow{MAdd} \mathbf{Y}_{i,j}: 1\times 1\times C$$

$$\mathbf{X}_{\Omega_{i,j}}: K\times K\times C \nearrow_{MAdd}$$

```
,
# span channels
self.span = nn.Conv2d(
    channels // reduction_ratio,
    self.groups * kernel_size ** 2,
    1
)

self.down_sample = nn.AvgPool2d(stride) if stride != 1 else nn.Identity()
self.unfold = nn.Unfold(kernel_size, padding=(kernel_size - 1) // 2, stride=stride)
```

Involution: Inverting the Inherence of Convolution for Visual Recognition

$$\mathbf{X}_{i,j}: 1\times 1\times C \xrightarrow{FC} 1\times 1\times \frac{C}{r} \xrightarrow{FC} 1\times 1\times (K^2G) \xrightarrow{reshape} K\times K\times G \xrightarrow{MAdd} \mathbf{Y}_{i,j}: 1\times 1\times C$$

$$\mathbf{X}_{\Omega_{i,j}}: K\times K\times C \nearrow_{MAdd}$$

```
def forward(self, x):
    # Note that 'h', 'w' are height & width of the output feature.

    # generate involution kernel: (b, G*K*K, h, w)
    weight_matrix = self.span(self.reduce(self.down_sample(x)))
    b, _, h, w = weight_matrix.shape

    # unfold input: (b, C*K*K, h, w)
    x_unfolded = self.unfold(x)
    # (b, C*K*K, h, w)->(b, G, C//G, K*K, h, w)
    x_unfolded = x_unfolded.view(b, self.groups, self.group_channels, self.kernel_size ** 2, h, w)

    # (b, G*K*K, h, w) -> (b, G, 1, K*K, h, w)
    weight_matrix = weight_matrix.view(b, self.groups, 1, self.kernel_size ** 2, h, w)
    # (b, G, C//G, h, w)
    mul_add = (weight_matrix * x_unfolded).sum(dim=3)
    # (b, C, h, w)
    out = mul_add.view(b, self.channels, h, w)

    return out
```

Involution: Inverting the Inherence of Convolution for Visual Recognition

Relation to Self-Attention

- multi-head self-attention operation: $\mathbf{Y}_{i,j,k} = \sum_{(p,q)\in\mathbf{\Omega}} (\mathbf{Q}\mathbf{K}^\top)_{i,j,p,q,\lceil kH/C \rceil} \mathbf{V}_{p,q,k}$
  - query position: (i, j), key position: (p, q)
  - query, key, value: $\mathbf{Q} = \mathbf{X}\mathbf{W}^Q \quad \mathbf{K} = \mathbf{X}\mathbf{W}^K \quad \mathbf{V} = \mathbf{X}\mathbf{W}^V$
  - #heads: H

- yet another instantiation of involution:
  - kernel generated from **a patch of pixels** with **dense correspondence**

$$\mathbf{\Psi}_{i,j} = \mathbf{\Omega} \qquad \mathcal{H}_{i,j} = \phi(\mathbf{X}_{\mathbf{\Omega}}) = (\mathbf{X}\mathbf{W}^{\mathbf{Q}})(\mathbf{X}\mathbf{W}^{\mathbf{K}})^\top$$

Involution: Inverting the Inherence of Convolution for Visual Recognition

$$\mathbf{Y}_{i,j,k} = \sum_{(p,q)\in\Omega} ((\mathbf{X}\mathbf{W}^{\mathbf{Q}})(\mathbf{X}\mathbf{W}^{\mathbf{K}})^{\top})_{i,j,p,q,\lceil kH/C\rceil} (\mathbf{X}\mathbf{W}^{\mathbf{V}})_{p,q,k}$$

$$\mathbf{Y}_{i,j,k} = \sum_{(u,v)\in\Delta_K} \mathcal{H}_{i,j,u+\lfloor K/2\rfloor,v+\lfloor K/2\rfloor,\lceil kG/C\rceil} \mathbf{X}_{i+u,j+v,k}.$$

1.self-attention中不同的head对应到involution中不同的group(在通道上划分的组)；
2.self-attention中每个位置的关系矩阵 $QK^T_{i,j,,}$，对应到involution中每个位置的kernel $H_{i,j}$

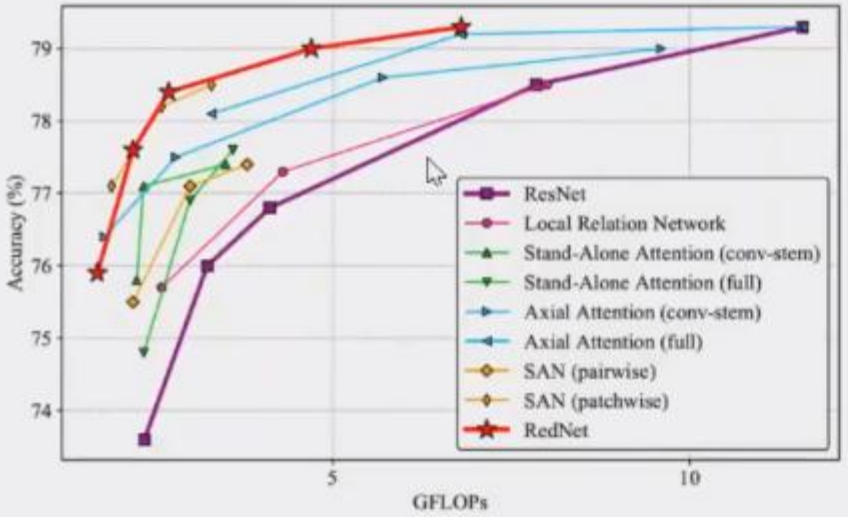
Involution: Inverting the Inherence of Convolution for Visual Recognition

$$\boldsymbol{\Psi}_{i,j} = \boldsymbol{\Omega}, \quad \boldsymbol{\mathcal{H}}_{i,j} = \boldsymbol{\phi}(\mathbf{X}_{\boldsymbol{\Omega}}) = (\mathbf{X}\mathbf{W}^{\mathbf{Q}})(\mathbf{X}\mathbf{W}^{\mathbf{K}})^{\top},$$

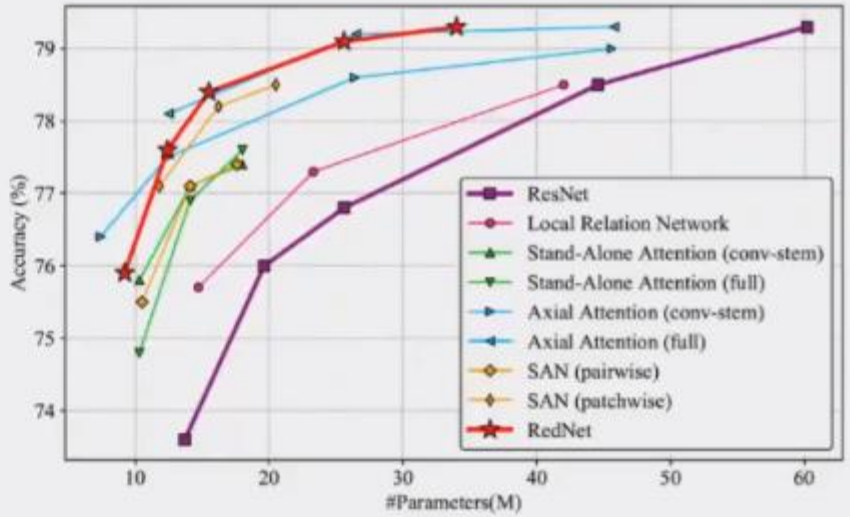
另外 $\mathbf{W}^{V}$对应于attention matrix multiplication前对 $\mathbf{X}$ 做的线性变换，self-attention操作后一般也会接另一个线性变换和残差连接，这个结构正好就对应于我们用involution替换resnet bottleneck结构中的 3×3convolution，前后也有两个 1×1convolution做线性变换

Involution: Inverting the Inherence of Convolution for Visual Recognition

Image Classification



(a) The accuracy-complexity envelope on ImageNet.



(b) The accuracy-parameter envelope on ImageNet.

RedNet achieves the optimal pareto frontier regarding the accuracy-efficiency tradeoffs, compared to both baseline ResNet and self-attention based models

Involution: Inverting the Inherence of Convolution for Visual Recognition

Ablation Analysis

$$\mathcal{H}_{i,j} = \phi(\mathbf{X}_{i,j}) = \mathbf{W}_1\sigma(\mathbf{W}_0\mathbf{X}_{i,j})$$

$$\mathbf{W}_0 \in \mathbb{R}^{\frac{C}{r}\times C} \quad \mathbf{W}_1 \in \mathbb{R}^{(K\times K\times G)\times\frac{C}{r}}$$

#parameters $\quad \frac{C^2 + K^2GC}{r}$

FLOPs $\quad HW \times \frac{C^2 + K^2GC}{r} + HW \times K^2C$

Kernel Size	#Params (M)	FLOPs (G)	Top-1 Acc. (%)
3×3	14.7	2.4	76.9
5×5	15.1	2.5	77.4
7×7	15.5	2.6	77.7
9×9	16.2	2.7	77.8

(a) Accuracy saturates with **kernel size** increasing.

#Group Channel	#Params (M)	FLOPs (G)	Top-1 Acc. (%)
1	30.2	5.0	77.9
4	18.5	3.0	77.7
16	15.5	2.6	77.7
C	14.6	2.4	76.5

(b) Appropriate **grouping channels** improves efficiency.

Function Form	#params (M)	FLOPs (G)	Top-1 Acc. (%)
$\mathbf{W}$	18.1	3.0	77.8
$\mathbf{W}_1\sigma\mathbf{W}_0, r = 1$	19.4	3.2	77.8
$\mathbf{W}_1\sigma\mathbf{W}_0, r = 4$	15.5	2.6	77.7
$\mathbf{W}_1\sigma\mathbf{W}_0, r = 16$	14.6	2.4	77.4

(c) Introducing the **bottleneck structure** reduces complexity.

谢谢大家