AI300学院

从LeNet到SENet

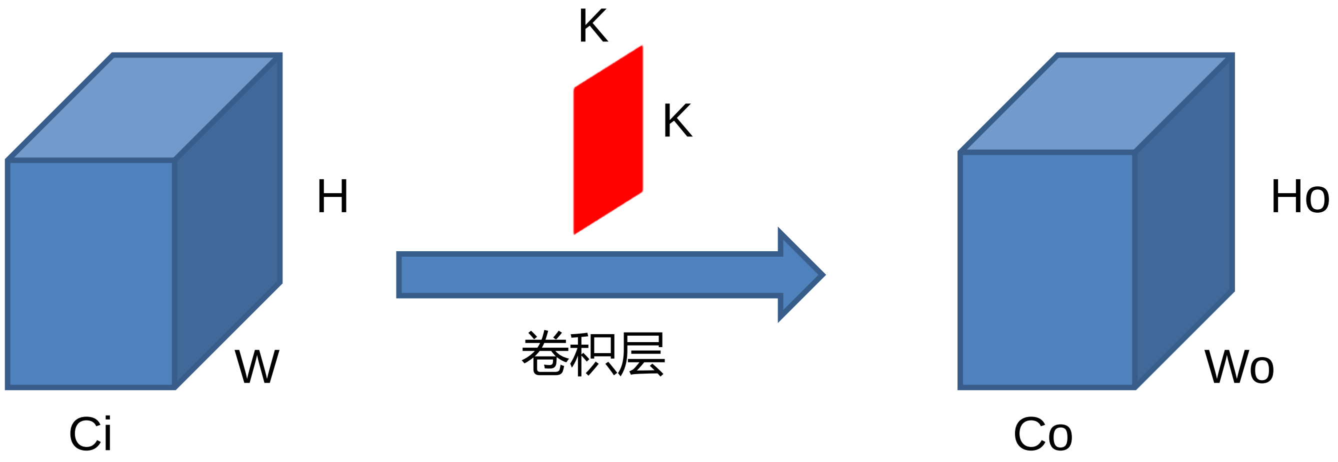
罗浩
浙江大学

大纲

- 卷积结构的类型
- 常用的卷积神经网络
- 常用的小型卷积网络

卷积结构的类型

正常卷积(Convolution)



Ho=(H+2×padding+1-K)/stride

Wo=(W+2×padding+1-K)/stride

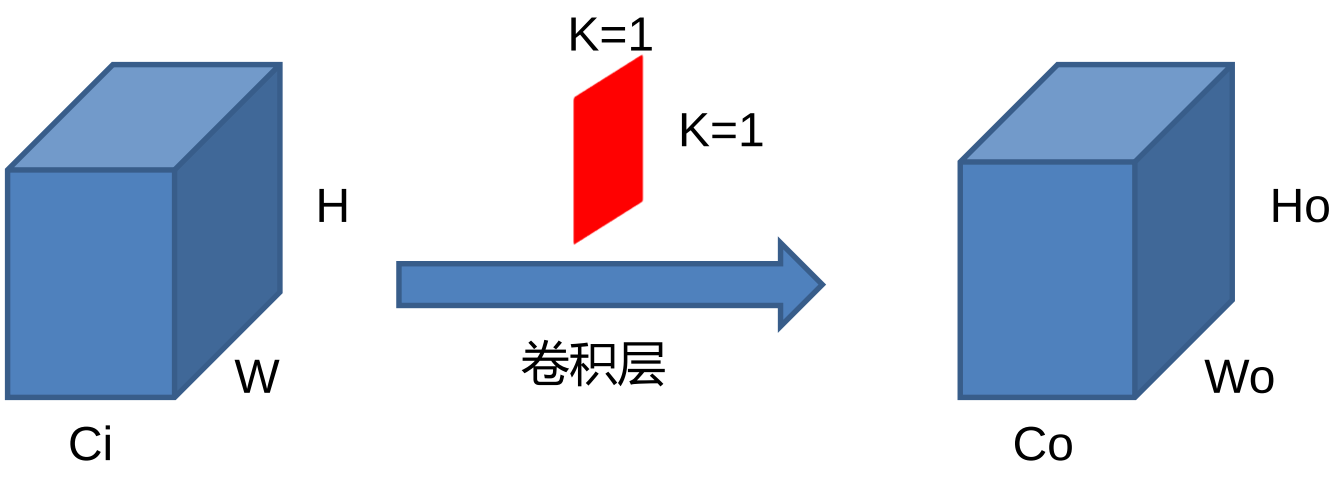
参数量：Ci×K×K×Co+bias

torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*)

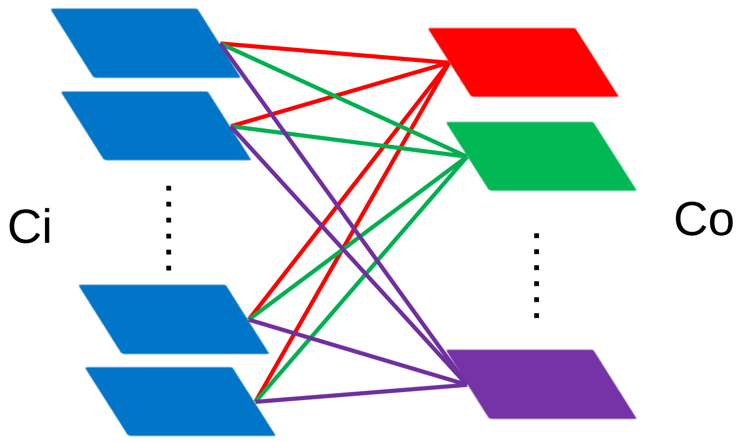
- **in_channels** (*int*) – Number of channels in the input image
- **out_channels** (*int*) – Number of channels produced by the convolution
- **kernel_size** (*int or tuple*) – Size of the convolving kernel
- **stride** (*int or tuple, optional*) – Stride of the convolution. Default: 1
- **padding** (*int or tuple, optional*) – Zero-padding added to both sides of the input. Default: 0
- **bias** (*bool, optional*) – If **True**, adds a learnable bias to the output. Default: True

卷积结构的类型

Pointwise Convolution



- kernel_size = 1
- 参数量：Ci×Co+bias
- 该卷积操作没有空间信息
- 通道维度上的全连接



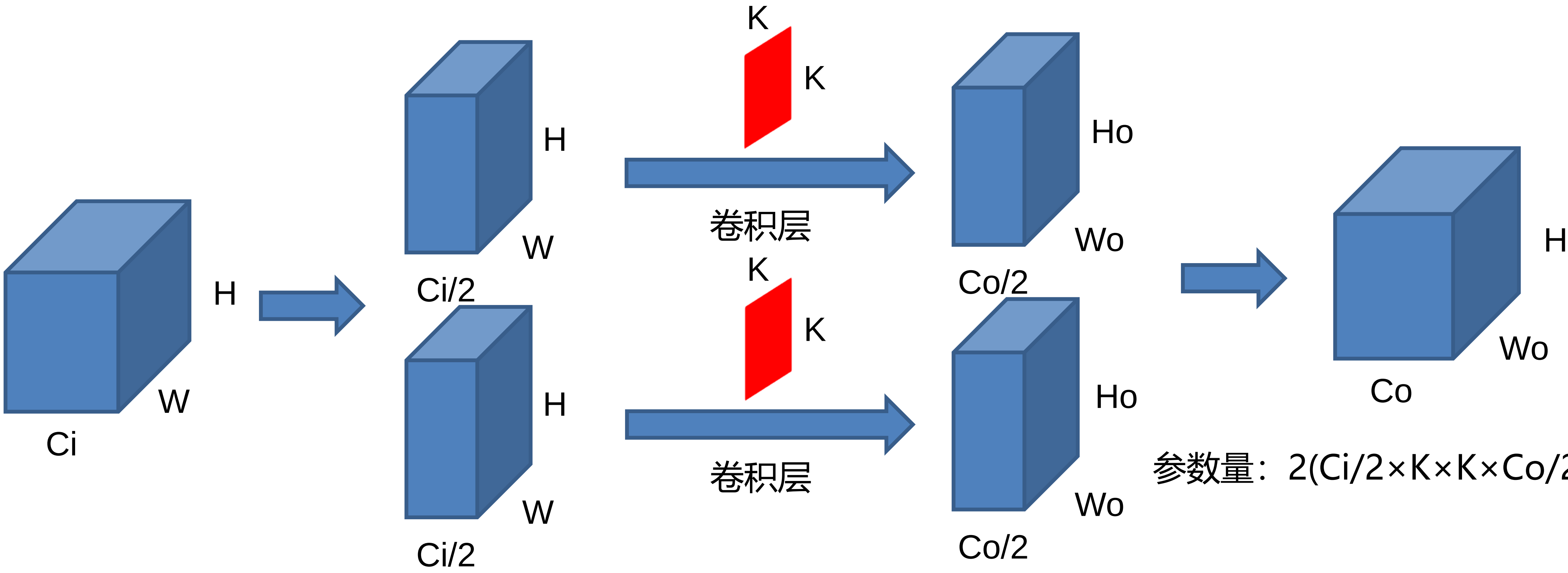
1×1卷积示例

torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*)

AI300学院

卷积结构的类型

分组卷积(Group Convolution)



参数量：2(Ci/2×K×K×Co/2)+bias

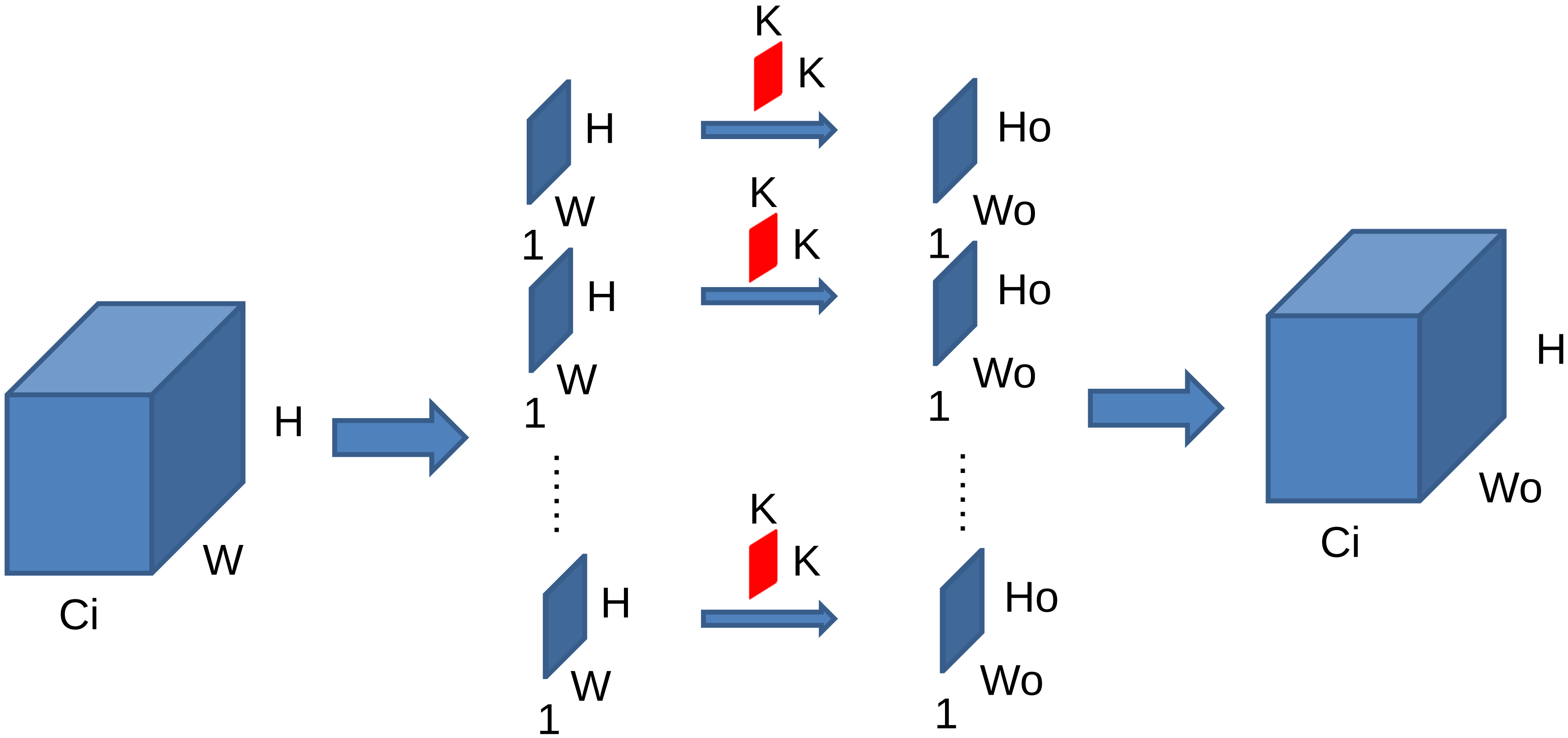
torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, ***groups=2***, *bias=True*)

- **groups** ([*int*], *optional*) – Number of blocked connections from input channels to output channels. Default: 1

AI300学院

卷积结构的类型

Channel-wise/Depthwise Convolution



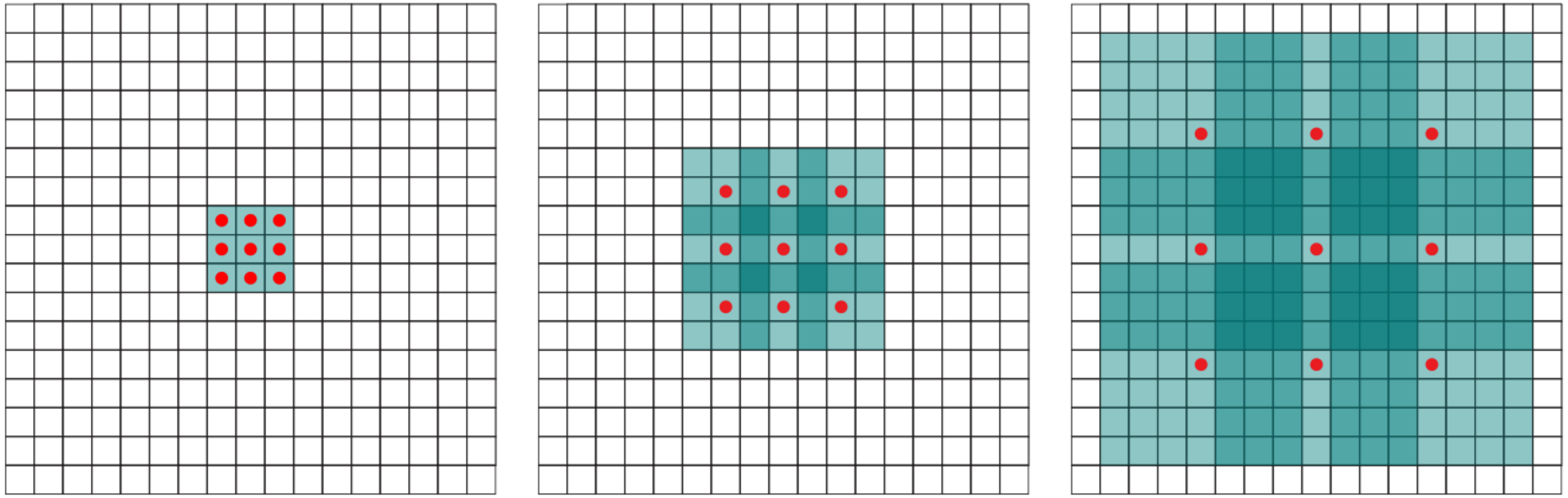
- 参数量：Ci×K×K+bias
- 分组卷积的极端形式
- 参数量少
- 通道之间的信息没有打通
- 通常后接1×1卷积打通通道信息

torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=Ci*, *bias=True*)

- **groups** (*int*, *optional*) – Number of blocked connections from input channels to output channels. Default: 1

卷积结构的类型

空洞卷积(Dilated Convolution)



(a) (b) (c)

- 参数量：Ci×K×K×Co+bias
- 参数量不变
- 扩大感受野，在分割任务中常见
- 提取多尺度的特征

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

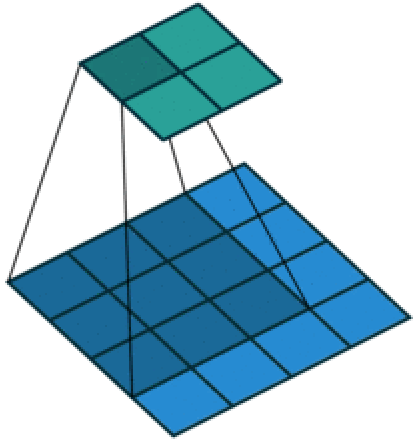
$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=2*, *groups=2*, *bias=True*)

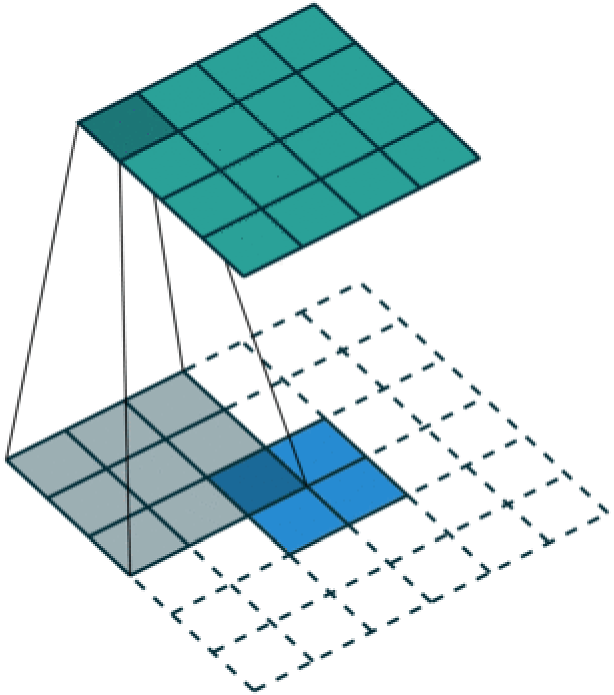
- **dilation** (*int* or *tuple*, *optional*) – Spacing between kernel elements. Default: 1

卷积结构的类型

转置卷积/反卷积(Dilated Convolution)



卷积



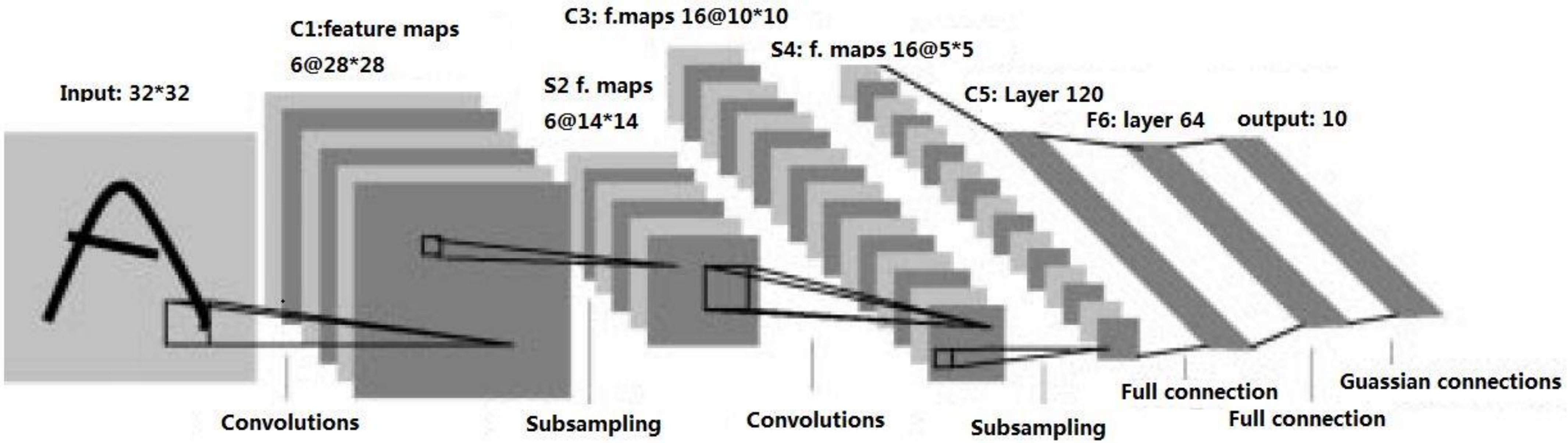
反卷积

- 参数量：Ci×K×K×Co+bias
- 卷积的逆操作
- 可学习的上采样层，在图像分割，图像生成中广泛应用

torch.nn.ConvTranspose2d*(in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *output_padding=0*, *groups=1*, *bias=True*, *dilation=1*)

常用的卷积神经网络

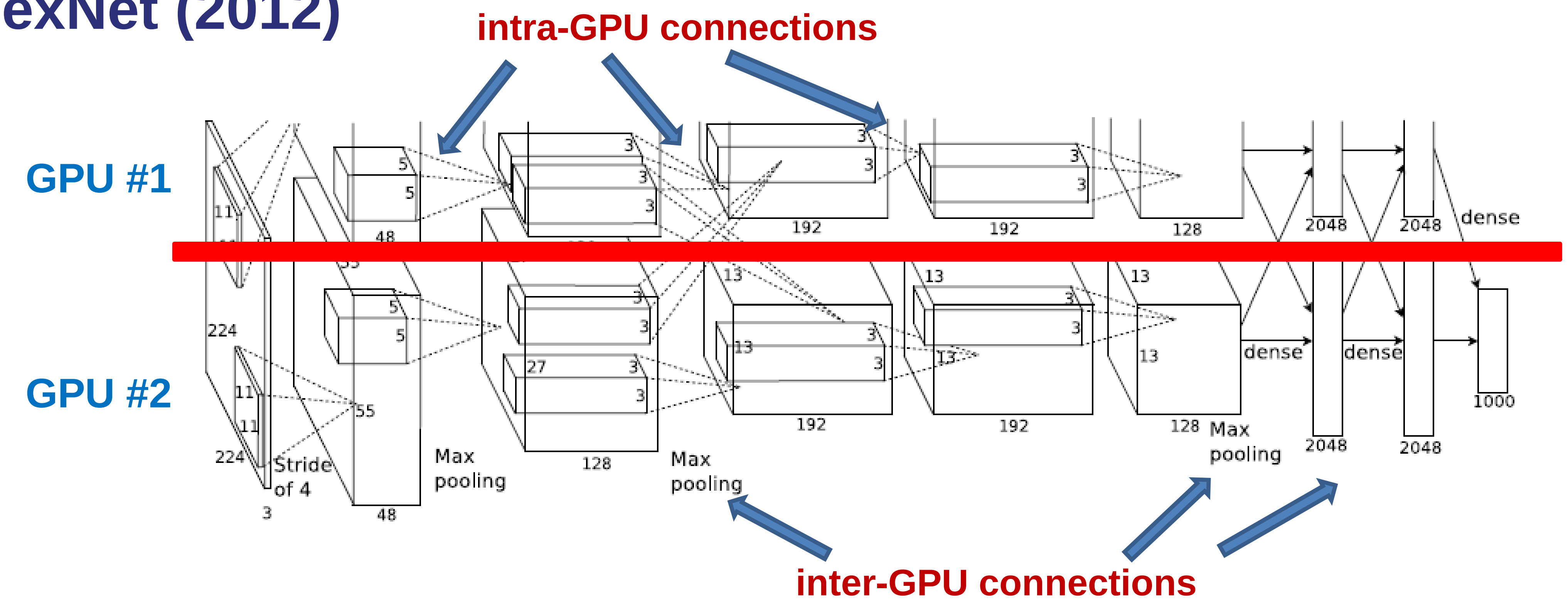
LeNet-5 (1990’s)




```
class LeNet5(nn.Module):
    def __init__(self):
        super(LeNet5).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5, padding=0)
        self.conv2 = nn.Conv2d(6, 16, 5)

        self.fc1 = nn.Linear(16*5*5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        x = F.max_pool2d(F.relu(self.conv2(x)), (2, 2))
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

常用的卷积神经网络

AlexNet (2012)

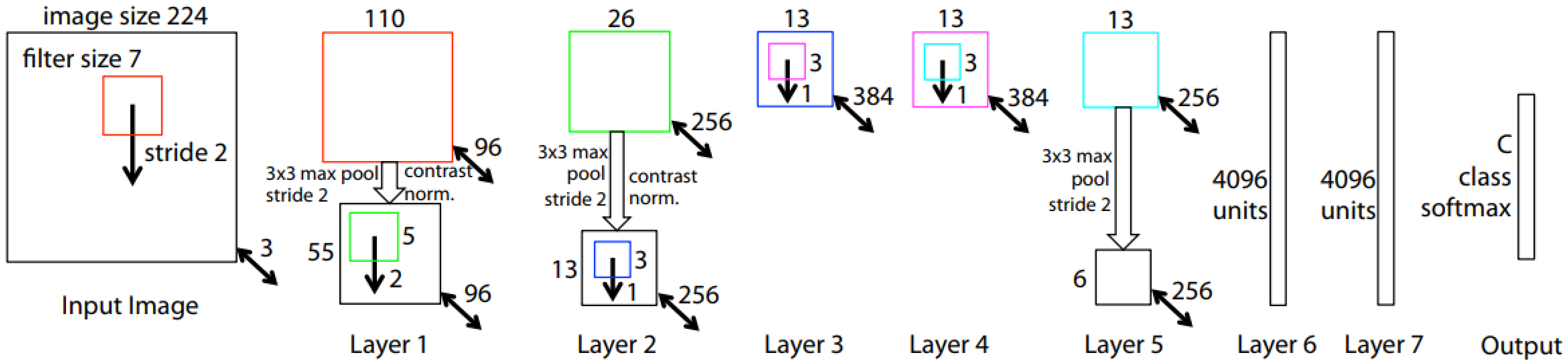


AlexNet有5个卷积层和3个全连接层，移除任意一层都会降低最终的效果

- Multiple GPU
- Group convolution
- ReLU
- Max pooling
- Dropout
- Local normalization

常用的卷积神经网络

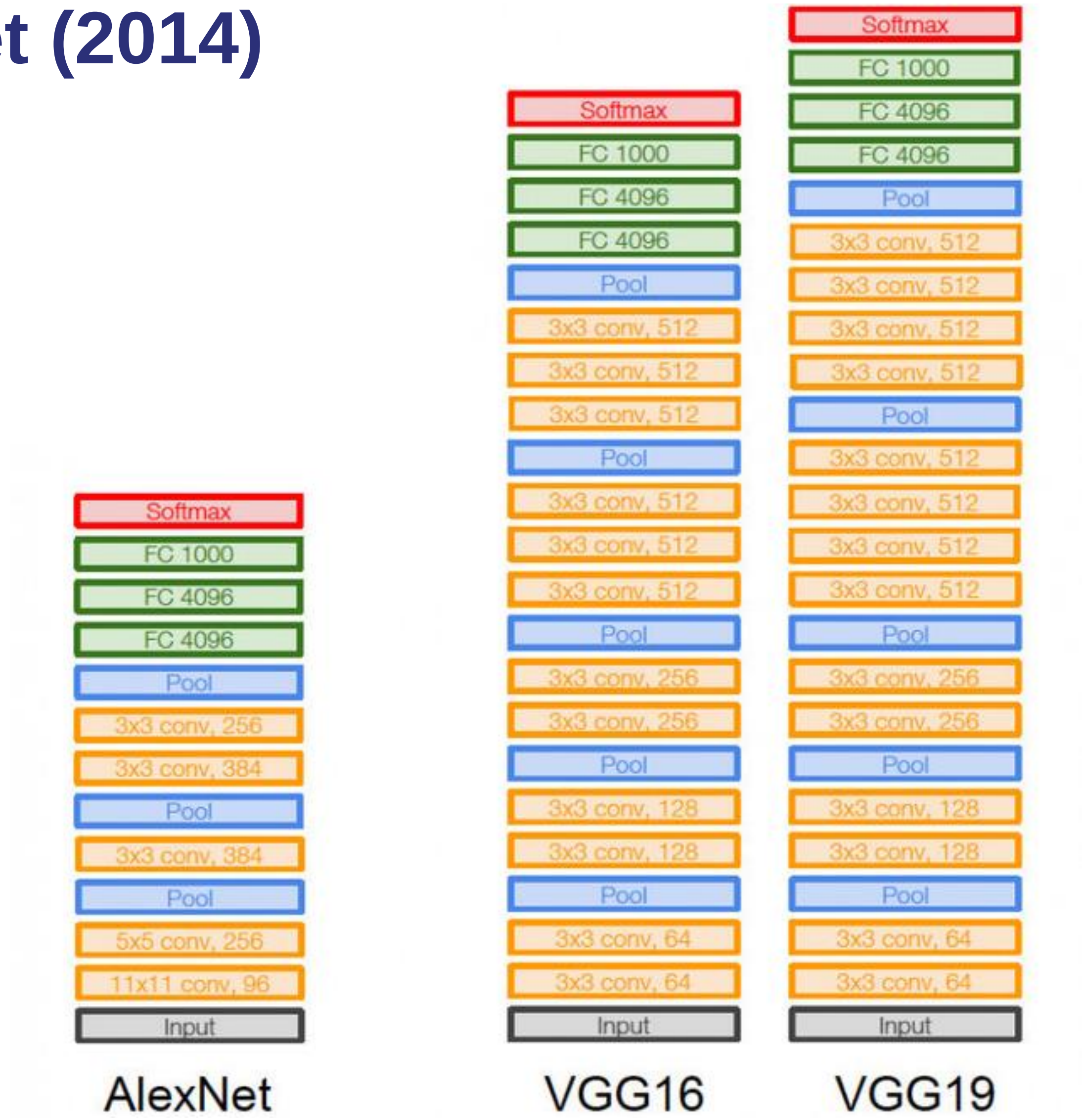
ZFNet (2013)



ZFNet在保留AlexNet的基本结构的同时利用反卷积网络可视化的技术对特定卷积层的卷积核尺寸进行了调整，第一层的卷积核从11*11减小到7*7，将stride从4减小到2，Top5的错误率比AlexNet比降低了1.7%。

常用的卷积神经网络

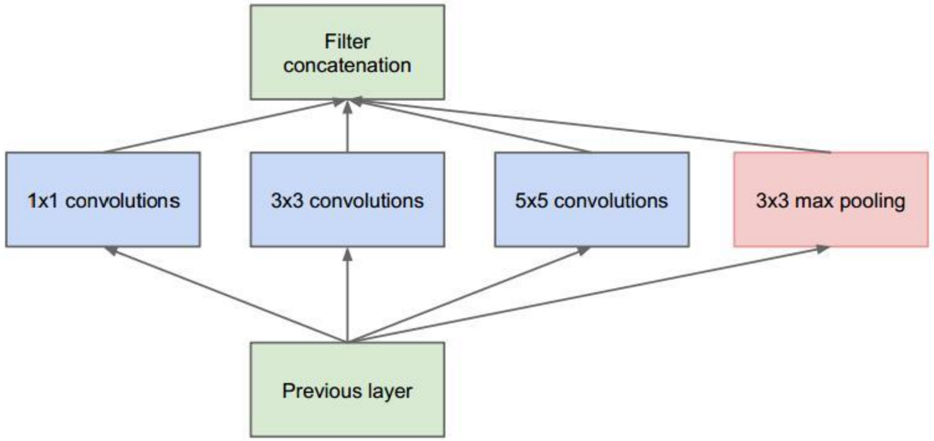
VGGNet (2014)



- 更深的网络
- 小卷积核的堆叠
- 卷积Block的重复
- 通道数呈现二进制的增加

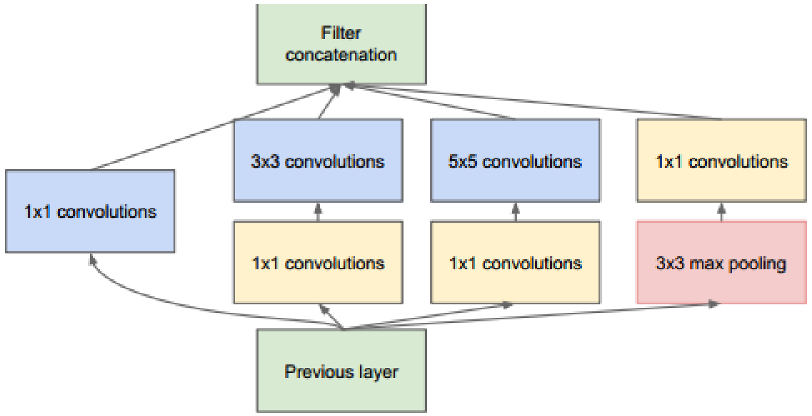
常用的卷积神经网络

GoogLeNet (2014)



(a) Inception module, naïve version

Inception-v1



(b) Inception module with dimension reductions

Inception-v2

- 不同尺度的feature map的融合
- Depthwise convolution减少参数

常用的卷积神经网络

GoogLeNet (2014)

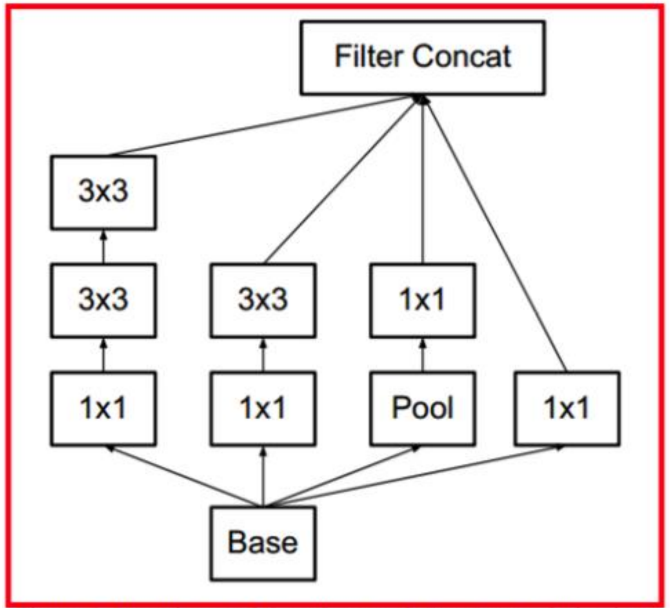


Figure 5. Inception modules where each 5 × 5 convolution is replaced by two 3 × 3 convolution, as suggested by principle 3 of Section 2.

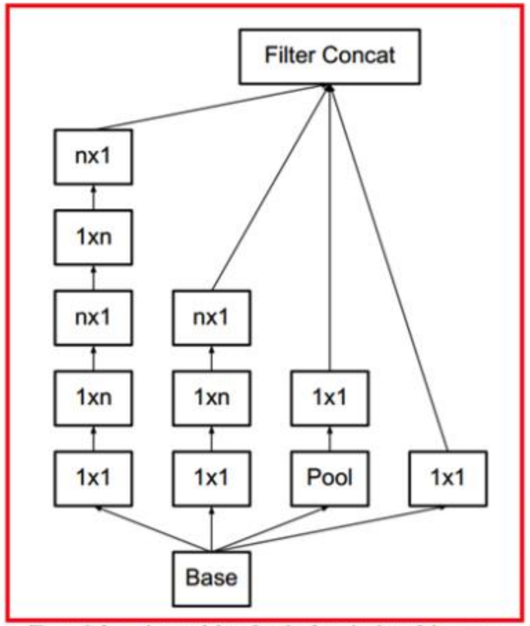


Figure 6. Inception modules after the factorization of the $n \times n$ convolutions. In our proposed architecture, we chose $n = 7$ for the 17×17 grid. (The filter sizes are picked using principle 3)

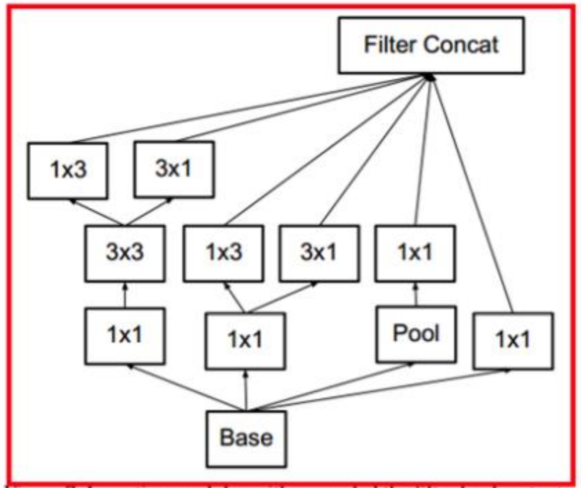


Figure 7. Inception modules with expanded the filter bank outputs. This architecture is used on the coarsest (8 × 8) grids to promote high dimensional representations, as suggested by principle 2 of Section 2. We are using this solution only on the coarsest grid, since that is the place where producing high dimensional sparse representation is the most critical as the ratio of local processing (by 1 × 1 convolutions) is increased compared to the spatial aggregation.

Inception-v2 **Inception-v3** **Inception-v4**

常用的卷积神经网络

GoogLeNet (2014)

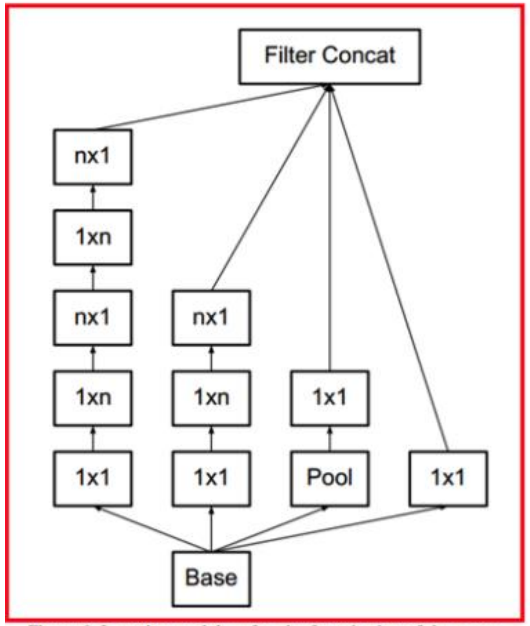


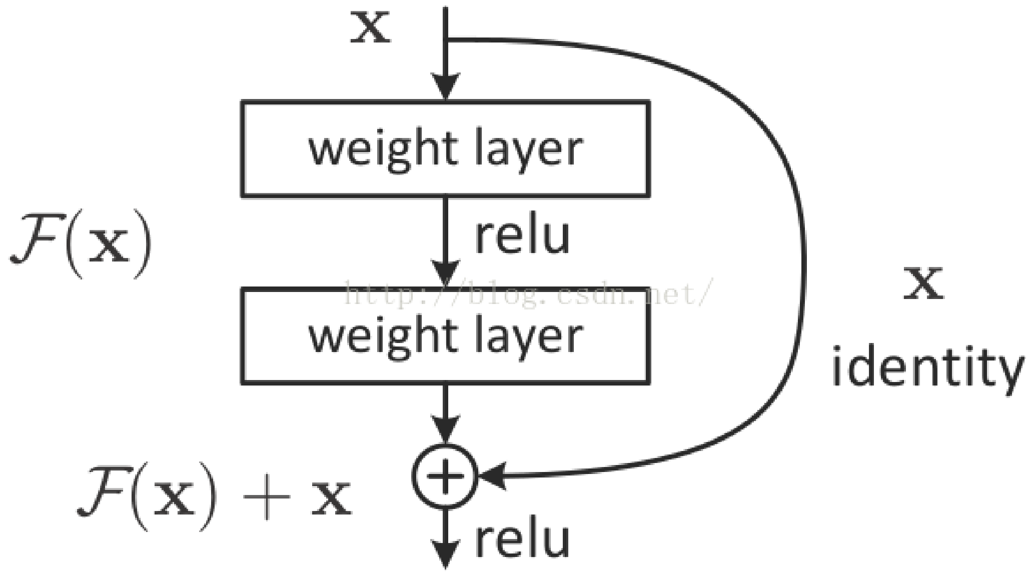
Figure 6. Inception modules after the factorization of the $n \times n$ convolutions. In our proposed architecture, we chose $n = 7$ for the 17×17 grid. (The filter sizes are picked using principle 3)

Inception-v3

```
class InceptionC(nn.Module):
    def __init__(self, in_channels, channels_7x7):
        super(InceptionC, self).__init__()
        self.branch1x1 = BasicConv2d(in_channels, 192, kernel_size=1)

        c7 = channels_7x7
        self.branch7x7_1 = BasicConv2d(in_channels, c7, kernel_size=1)
        self.branch7x7_2 = BasicConv2d(c7, c7, kernel_size=(1, 7), padding=(0, 3))
        self.branch7x7_3 = BasicConv2d(c7, 192, kernel_size=(7, 1), padding=(3, 0))

        self.branch7x7dbl_1 = BasicConv2d(in_channels, c7, kernel_size=1)
        self.branch7x7dbl_2 = BasicConv2d(c7, c7, kernel_size=(7, 1), padding=(3, 0))
        self.branch7x7dbl_3 = BasicConv2d(c7, c7, kernel_size=(1, 7), padding=(0, 3))
        self.branch7x7dbl_4 = BasicConv2d(c7, c7, kernel_size=(7, 1), padding=(3, 0))
        self.branch7x7dbl_5 = BasicConv2d(c7, 192, kernel_size=(1, 7), padding=(0, 3))

        self.branch_pool = BasicConv2d(in_channels, 192, kernel_size=1)
    def forward(self, x):
        branch1x1 = self.branch1x1(x)

        branch7x7 = self.branch7x7_1(x)
        branch7x7 = self.branch7x7_2(branch7x7)
        branch7x7 = self.branch7x7_3(branch7x7)

        branch7x7dbl = self.branch7x7dbl_1(x)
        branch7x7dbl = self.branch7x7dbl_2(branch7x7dbl)
        branch7x7dbl = self.branch7x7dbl_3(branch7x7dbl)
        branch7x7dbl = self.branch7x7dbl_4(branch7x7dbl)
        branch7x7dbl = self.branch7x7dbl_5(branch7x7dbl)

        branch_pool = F.avg_pool2d(x, kernel_size=3, stride=1, padding=1)
        branch_pool = self.branch_pool(branch_pool)

        outputs = [branch1x1, branch7x7, branch7x7dbl, branch_pool]
        return torch.cat(outputs, 1)
```

常用的卷积神经网络

ResNet (2016)

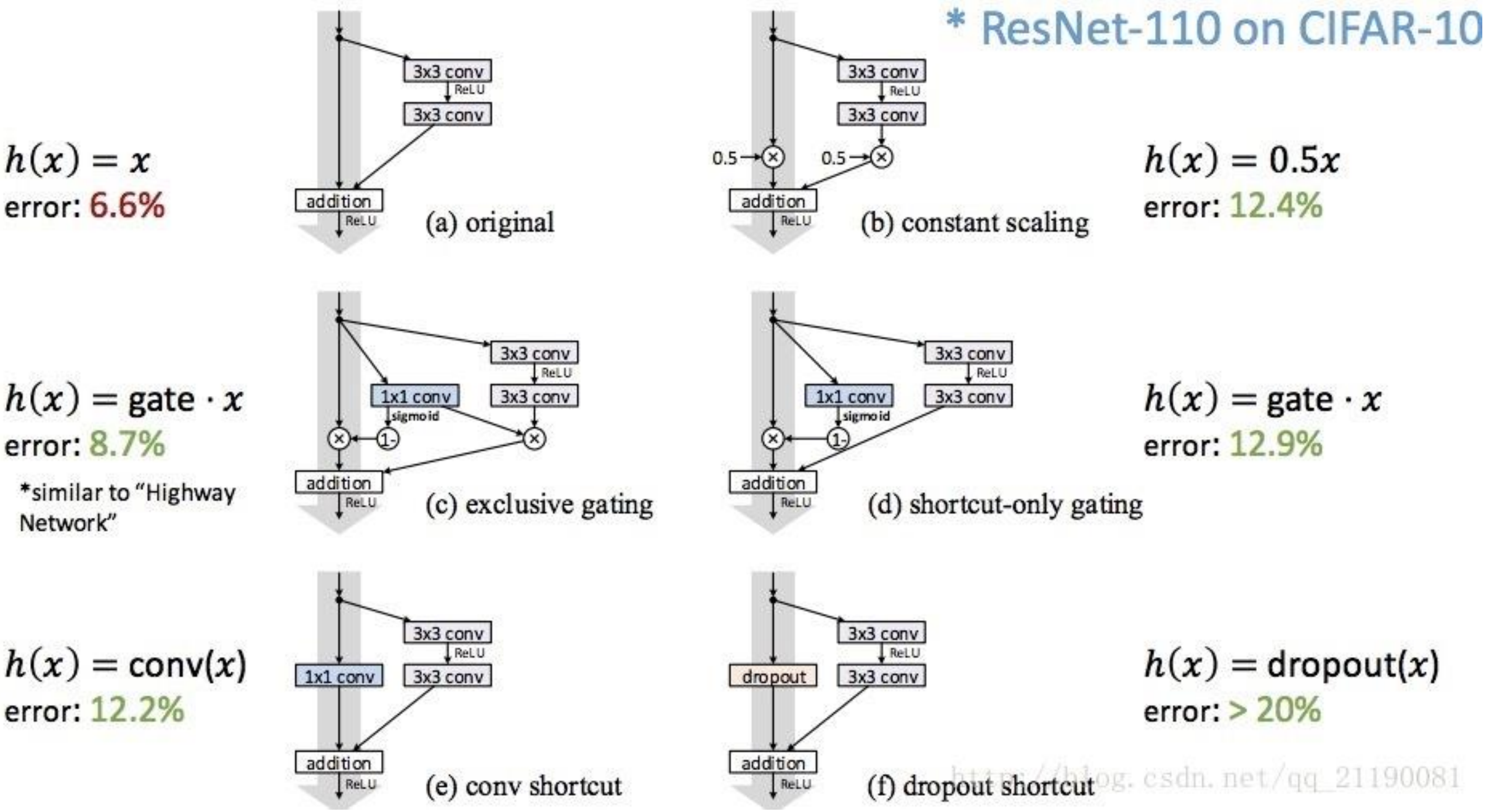


Residual block

- Resnet的出发点是认为深层网络不应该比浅层网络性能差，所以为了防止网络退化，引入了大量identity恒等映射，这样就可以把原始信息流入更深的层，抑制了信息的退化
- 残差块有用是因为identity这一支路的导数是1，所以可以把深层的loss很好的保留传递给浅层，因为神经网络一个很大的问题就是梯度链式法则带来的梯度弥散
- 残差块就是一个差分放大器
- …

常用的卷积神经网络

ResNet (2016)

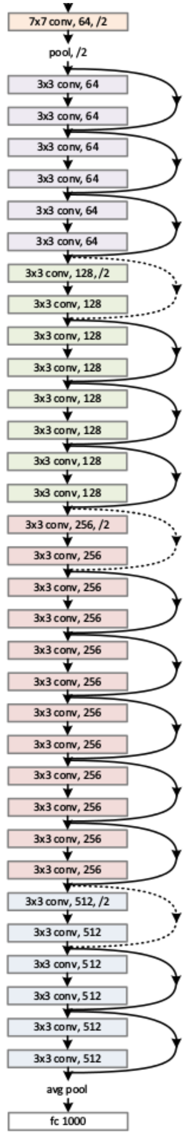


Shortcut支路就只能是identity恒等映射，用其他映射效果都不如identity

常用的卷积神经网络

ResNet (2016)

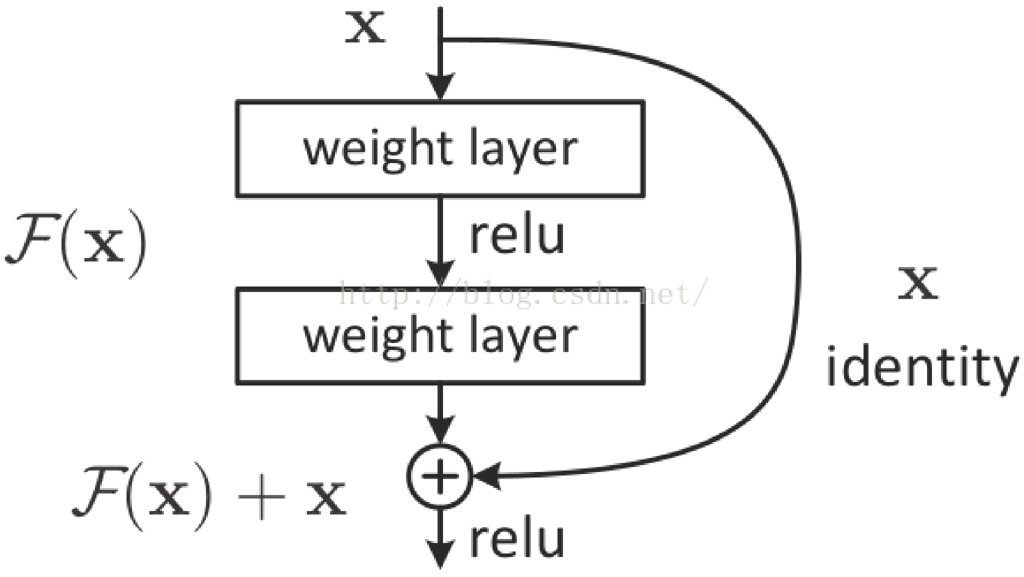
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9



ResNet最多可到1001层

常用的卷积神经网络

Resnet (2016)



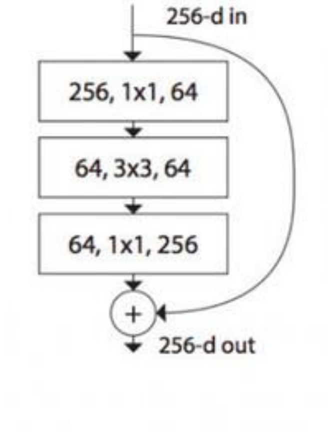
$$H(x) = F(x) + x$$

Residual block

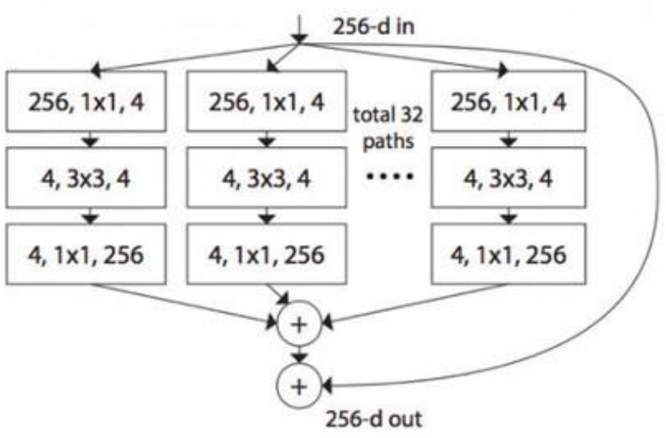
```python
def conv3x3(in_planes, out_planes, stride=1):
    """3x3 convolution with padding"""
    return nn.Conv2d(in_planes, out_planes, kernel_size=3, stride=stride,
                     padding=1, bias=False)

class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, inplanes, planes, stride=1, downsample=None):
        super(BasicBlock, self).__init__()
        self.conv1 = conv3x3(inplanes, planes, stride)
        self.bn1 = nn.BatchNorm2d(planes)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = conv3x3(planes, planes)
        self.bn2 = nn.BatchNorm2d(planes)
        self.downsample = downsample
        self.stride = stride

    def forward(self, x):
        residual = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)

        if self.downsample is not None:
            residual = self.downsample(x)

        out += residual
        out = self.relu(out)

        return out
```

常用的卷积神经网络

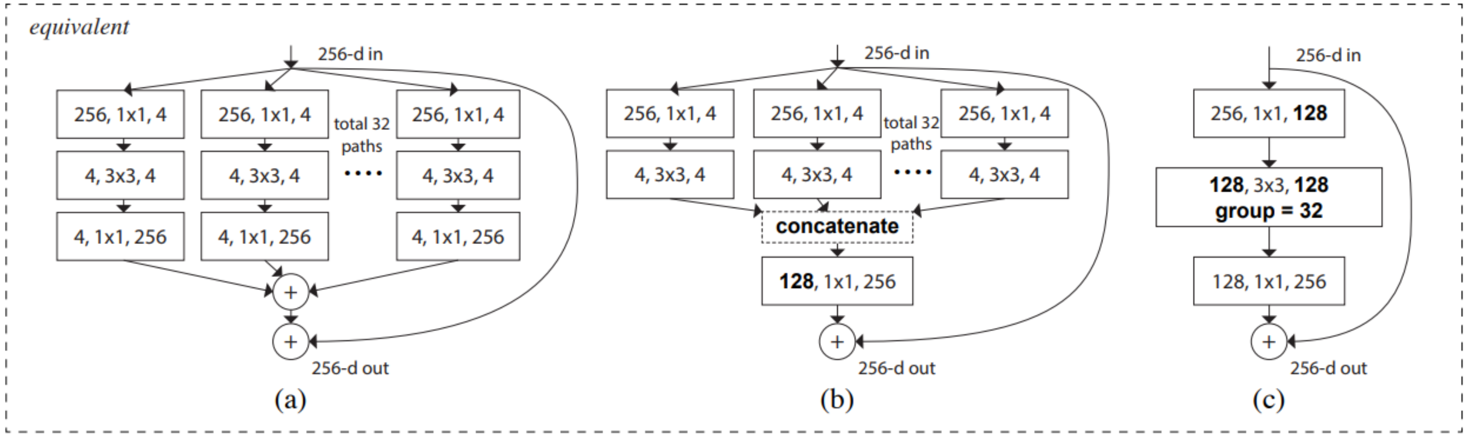
ResNeXt (2017)



ResNet



ResNeXt

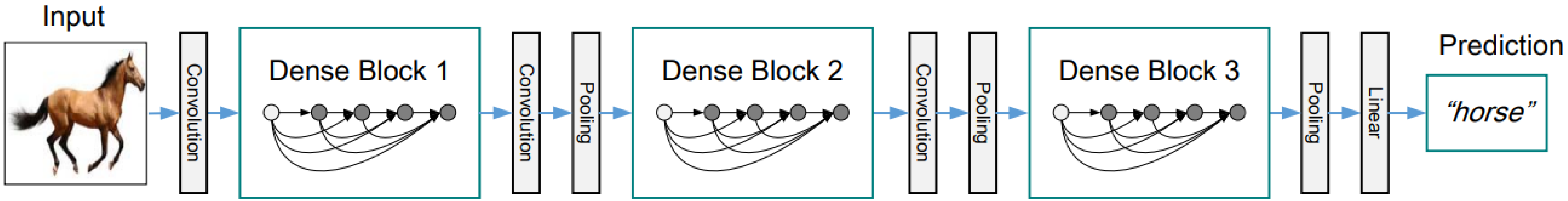


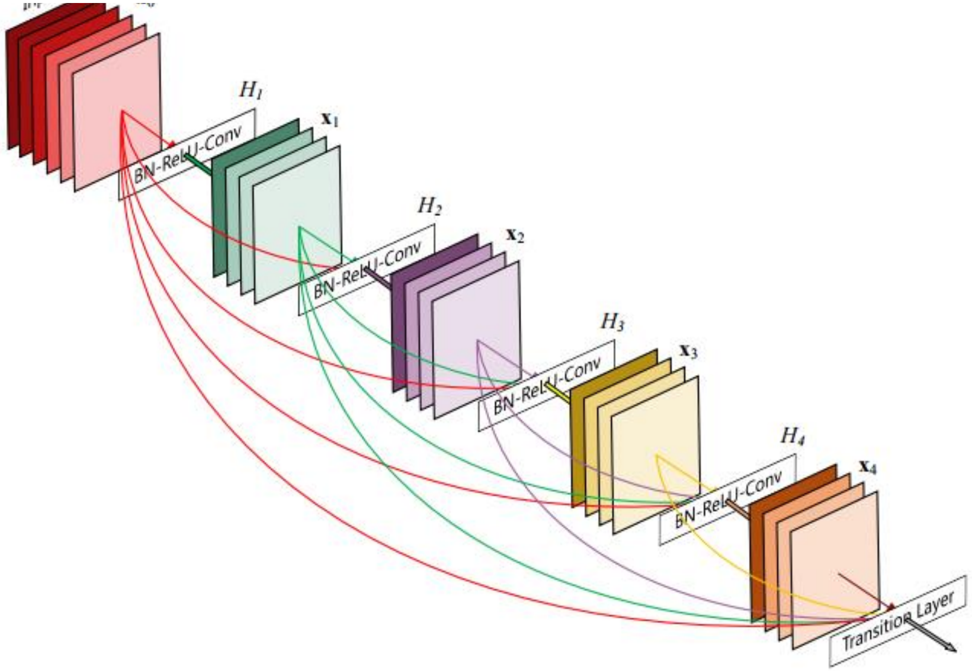
Equivalent

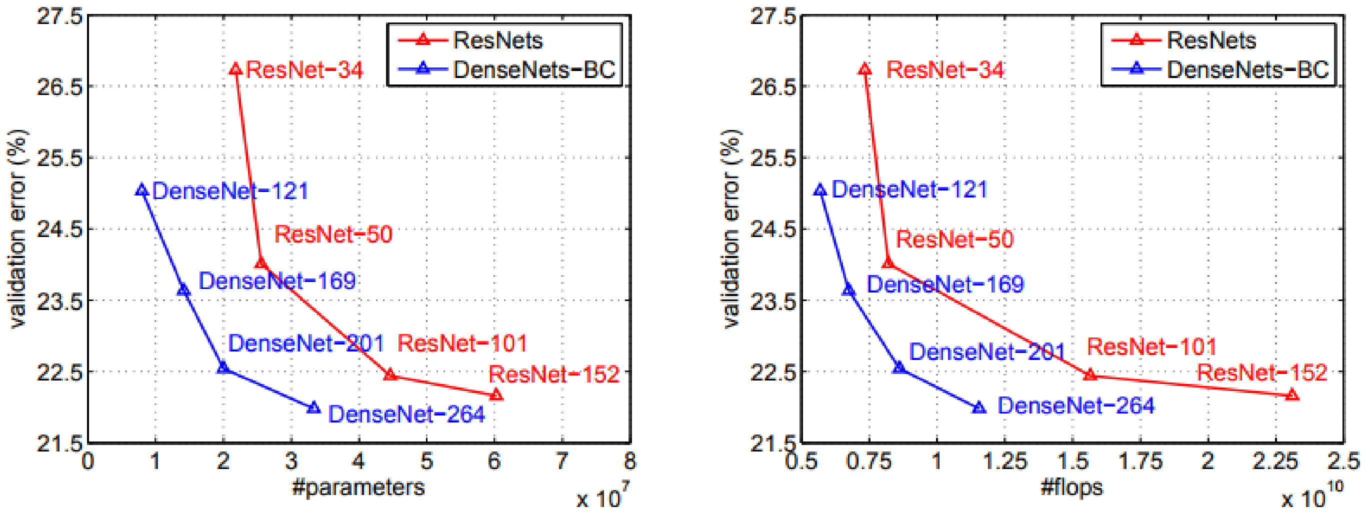
ResNeXt在ResNet里面引入分组卷积的思想

常用的卷积神经网络

DenseNet (2017)



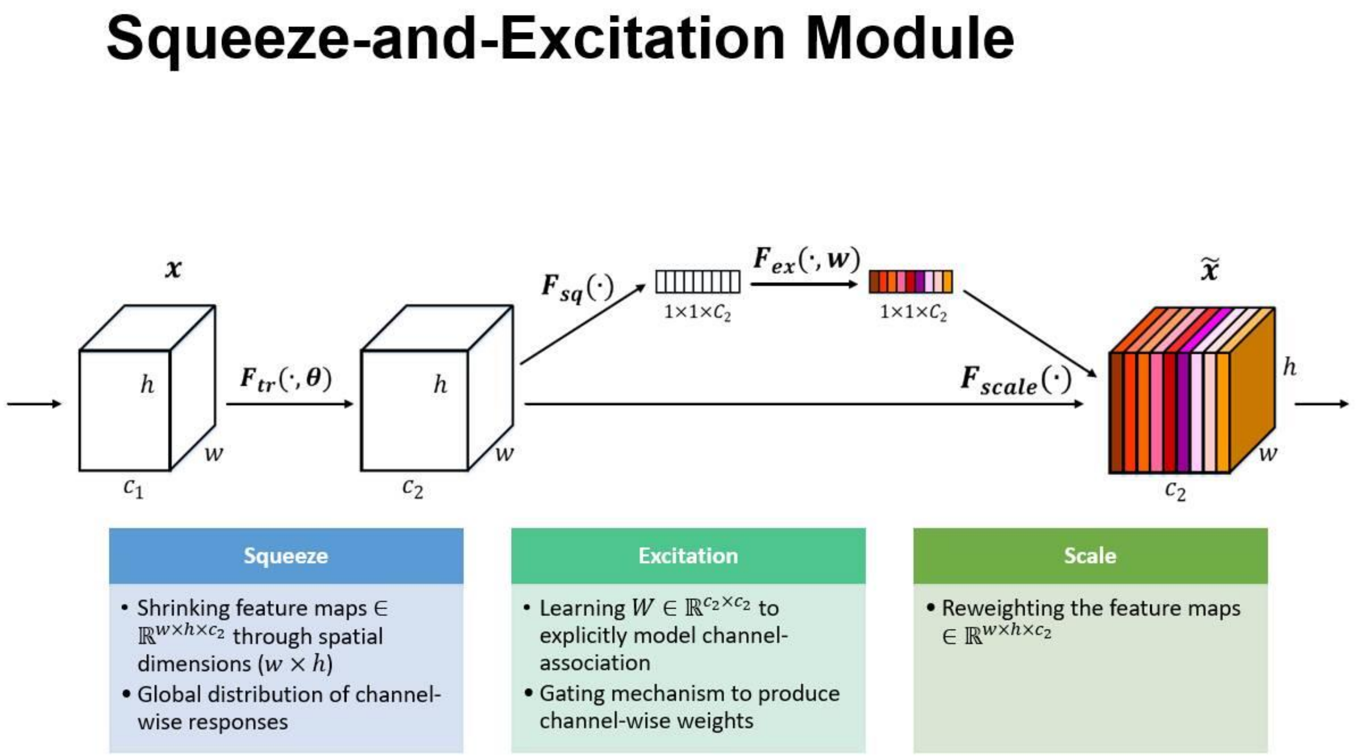


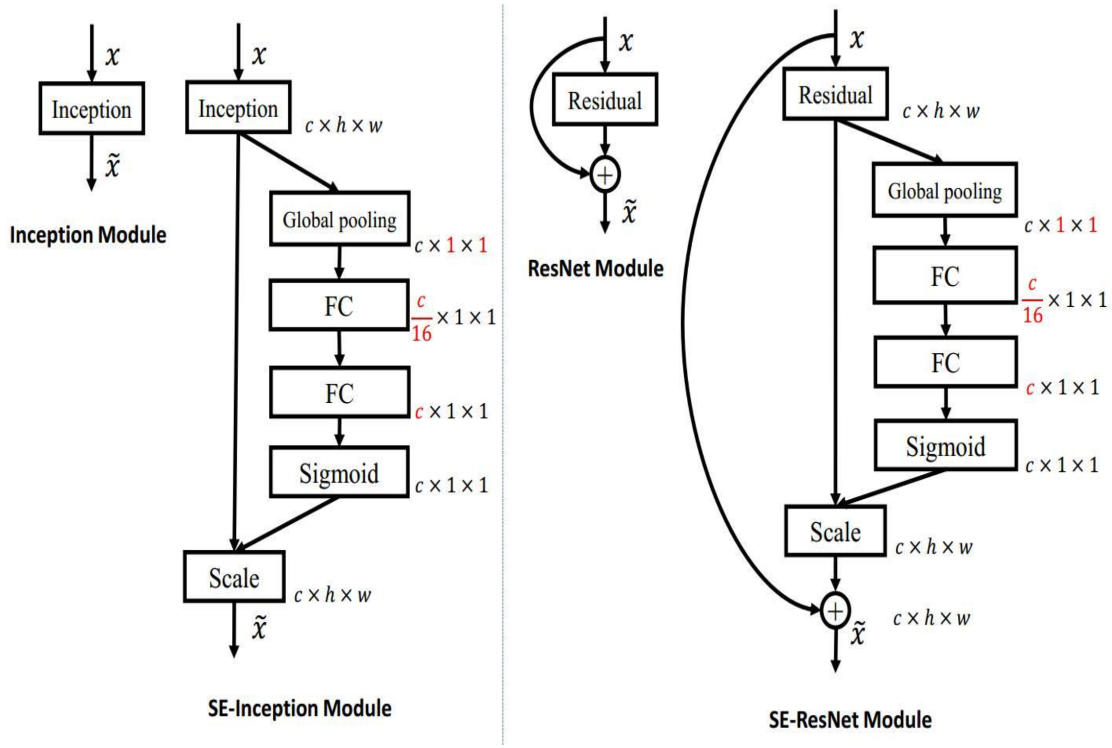


DenseNet把shortcut思想发挥到了极致

常用的卷积神经网络

SENet(2017)

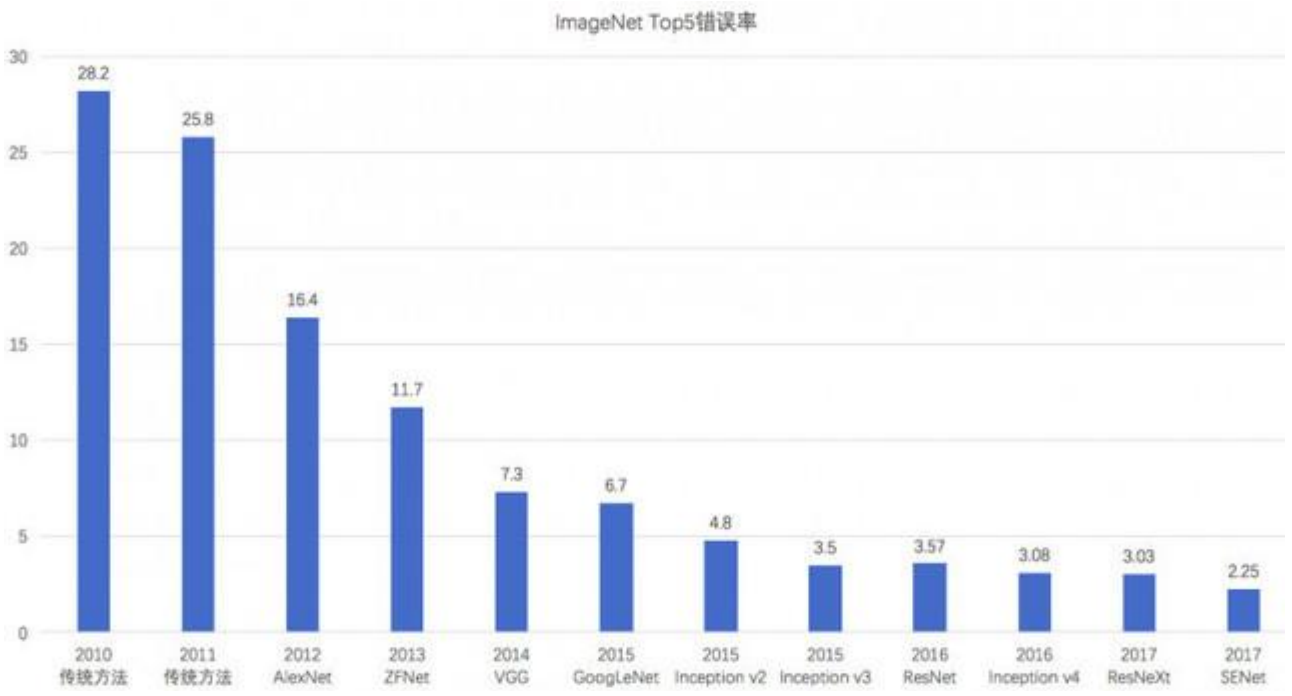


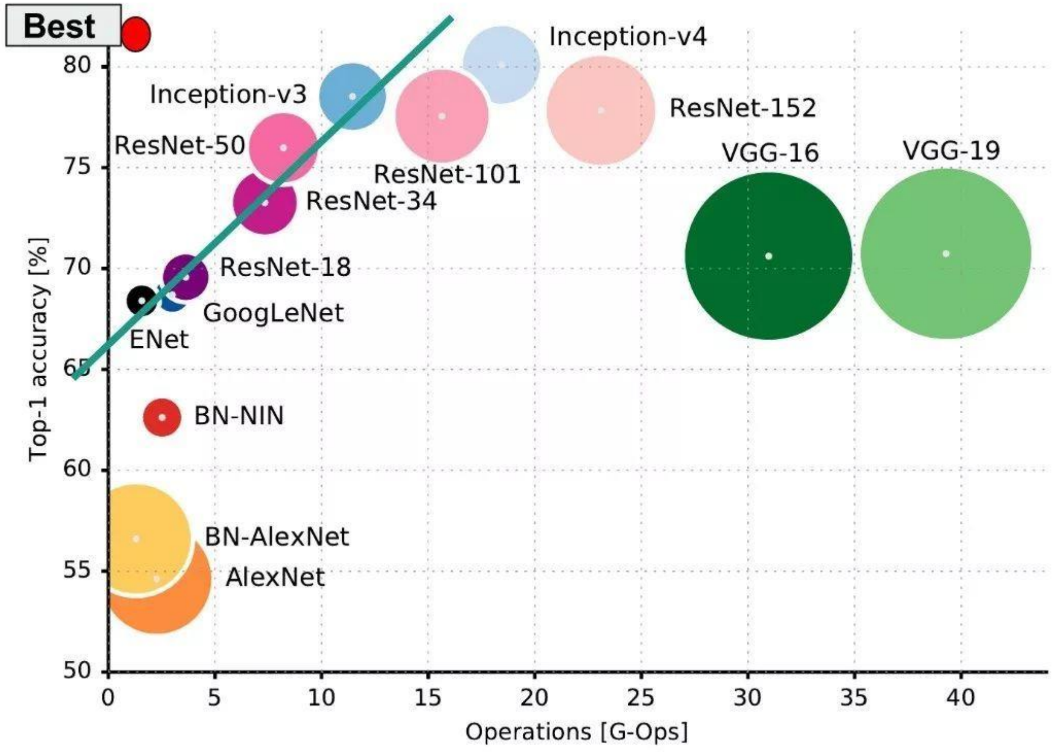


- 融合spatial信息和channel-wise信息
- SE module的核心就是一个channel attention

常用的卷积神经网络

Summary





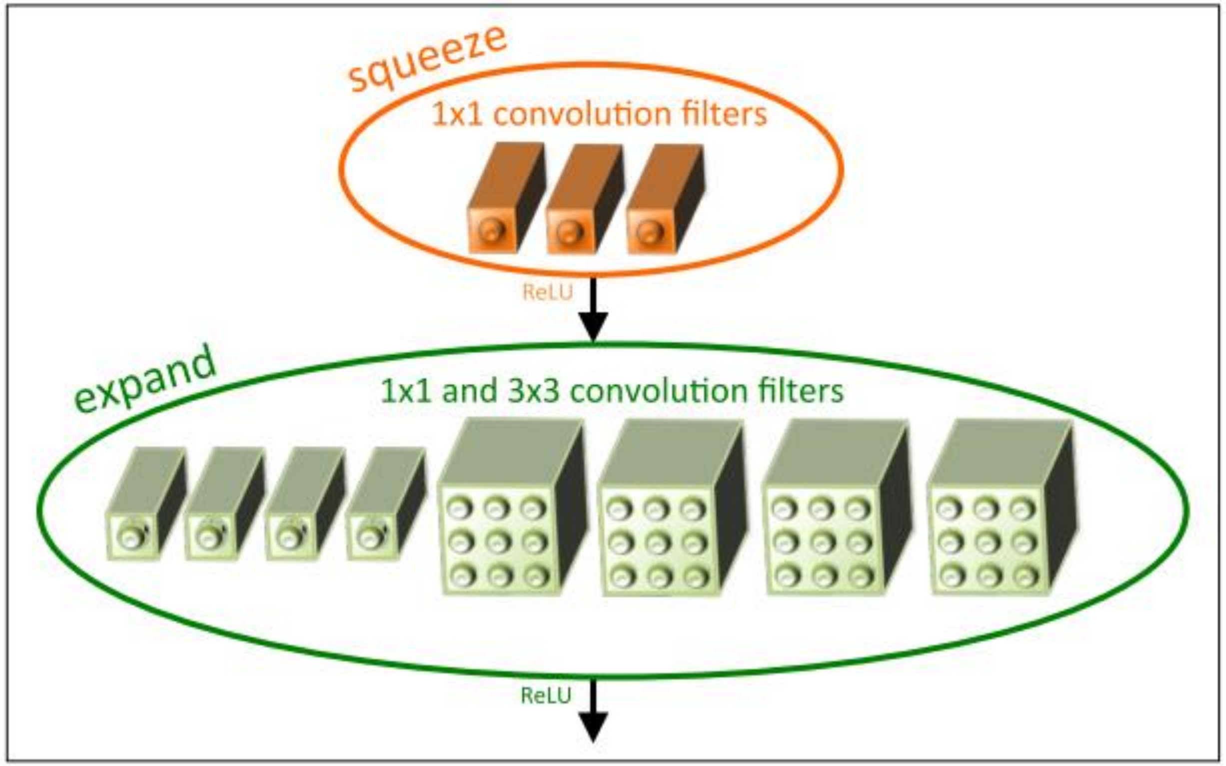
常用的小型网络结构

常用网络计算量

Year	Model	Layers	Parameter	FLOPs	ImageNet Top-5 error
2012	AlexNet	5+3	60M	725M	16.4%
2013	Clarifai	5+3	60M	—	11.7%
2014	MSRA	5+3	200M	—	8.06%
2014	VGG-19	16+3	143M	19.6B	7.32%
2014	GoogLeNet	22	6.8M	1.566B	6.67%
2015	ResNet	152	19.4M	11.3B	3.57%

常用的小型网络结构

SqueezeNet




```python
class Fire(nn.Module):

    def __init__(self, inplanes, squeeze_planes,
                 expand1x1_planes, expand3x3_planes):
        super(Fire, self).__init__()
        self.inplanes = inplanes
        self.squeeze = nn.Conv2d(inplanes, squeeze_planes, kernel_size=1)
        self.squeeze_activation = nn.ReLU(inplace=True)
        self.expand1x1 = nn.Conv2d(squeeze_planes, expand1x1_planes,
                                   kernel_size=1)
        self.expand1x1_activation = nn.ReLU(inplace=True)
        self.expand3x3 = nn.Conv2d(squeeze_planes, expand3x3_planes,
                                   kernel_size=3, padding=1)
        self.expand3x3_activation = nn.ReLU(inplace=True)

    def forward(self, x):
        x = self.squeeze_activation(self.squeeze(x))
        return torch.cat([
            self.expand1x1_activation(self.expand1x1(x)),
            self.expand3x3_activation(self.expand3x3(x))
        ], 1)
```

Fire Module

常用的小型网络结构

SqueezeNet

1.使用更小的1*1卷积核来替换3*3卷积核

2.减少输入3*3卷积的特征图的数量

3.减少pooling

4.比起AlexNet，SqueezeNet可以压缩510x的情况下在ImageNet上得到类似精度

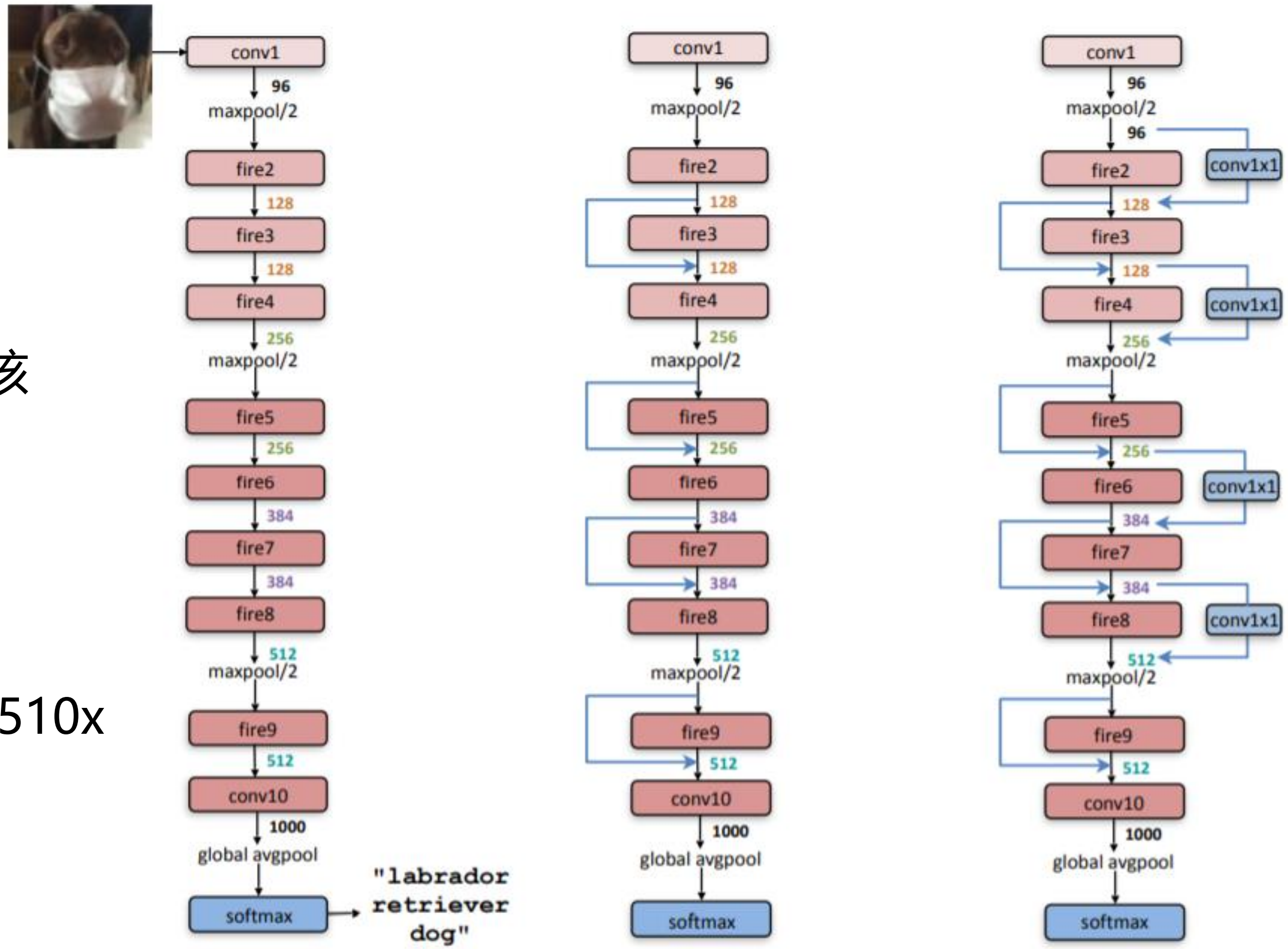
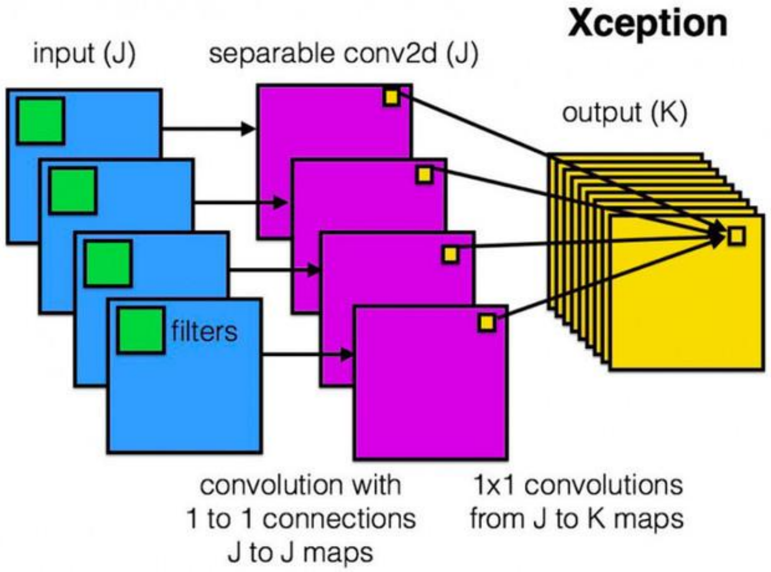


Figure 2: Macroarchitectural view of our SqueezeNet architecture. Left: SqueezeNet (Section 3.3); Middle: SqueezeNet with simple bypass (Section 6); Right: SqueezeNet with complex bypass (Section 6).

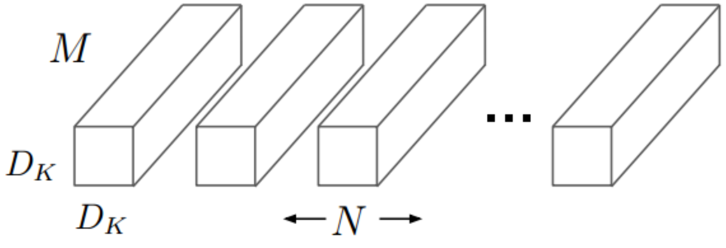
常用的小型网络结构

Xception结构

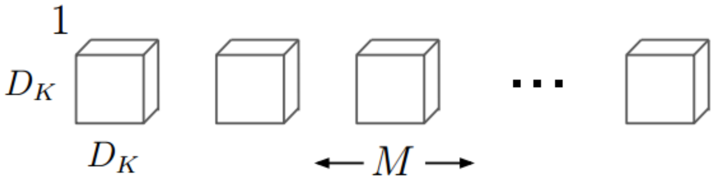


Xception cell

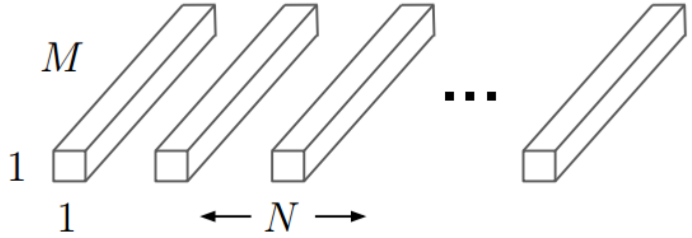
Xception



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

把普通卷积操作分成两部分

- Depthwise Convolution

  计算量 $D_K \cdot D_K \cdot M \cdot D_F \cdot D_F$

- Pointwise Convolution

  计算量 $M \cdot N \cdot D_F \cdot D_F$

上面两步合称Depthwise Separable Convolution

与原卷积计算量之比 $\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_K^2}$

- MobileNet里面利用了大量的Xception结构
- Xception使计算量减少，但是显存消耗增多

AI300学院

常用的小型网络结构

MobileNet

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5× Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
5× Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Table 2. Resource Per Layer Type

Type	Mult-Adds	Parameters
Conv 1×1	94.86%	74.59%
Conv DW 3×3	3.06%	1.06%
Conv 3×3	1.19%	0.02%
Fully Connected	0.18%	24.33%

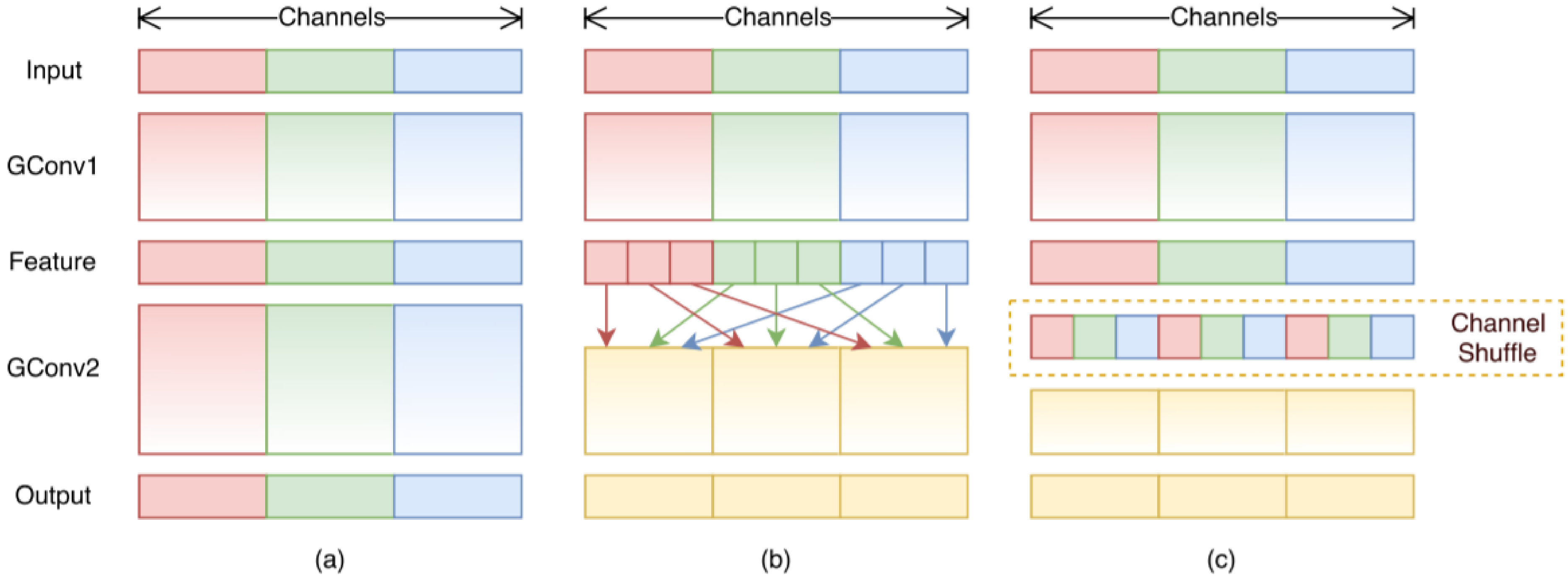
- MobileNet 是 Xception 结构的级联
- 大量网络消耗集中在1×1的Pointwise卷积上

Howard A G, Zhu M, Chen B, et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications[J]. arXiv preprint arXiv:1704.04861, 2017.

AI300学院

常用的小型网络结构

ShuffleNet



- ShuffleNet使用通道打乱操作来代替1×1卷积，实现通道信息的融合
- Shuffle操作通过Reshape操作实现，不包含参数

常用的小型网络结构

ShuffleNet

```python
class ChannelShuffle(nn.Module):
    def __init__(self, num_groups):
        super(ChannelShuffle, self).__init__()
        self.g = num_groups

    def forward(self, x):
        b, c, h, w = x.size()
        n = c / self.g
        # reshape
        x = x.view(b, self.g, n, h, w)
        # transpose
        x = x.permute(0, 2, 1, 3, 4).contiguous()
        # flatten
        x = x.view(b, c, h, w)
        return x
```

```
>>> x = torch.randn(2, 3, 5)
>>> x.size()
torch.Size([2, 3, 5])
>>> x.permute(2, 0, 1).size()
torch.Size([5, 2, 3])
```

```
In [1]: a = [[1,2,3],[4,5,6],[7,8,9]]

In [2]: import torch

In [3]: x = torch.tensor(a)

In [4]: x
Out[4]:
tensor([[ 1,  2,  3],
        [ 4,  5,  6],
        [ 7,  8,  9]])

In [5]: y = x.permute(1,0)

In [6]: y
Out[6]:
tensor([[ 1,  4,  7],
        [ 2,  5,  8],
        [ 3,  6,  9]])
```

Permute操作可以交换维度

AI300学院

常用的小型网络结构

ShuffleNet

Model	Cls err. (%)	Complexity (MFLOPs)
VGG-16 [30]	28.5	15300
ShuffleNet 2× ($g = 3$)	26.3	**524**
GoogleNet [33]*	31.3	1500
ShuffleNet 1× ($g = 8$)	32.4	**140**
AlexNet [21]	42.8	720
SqueezeNet [14]	42.5	833
ShuffleNet 0.5× ($g = 4$)	41.6	**38**

Model	Cls err. (%)	FLOPs	224 × 224	480 × 640	720 × 1280
ShuffleNet 0.5× ($g = 3$)	43.2	38M	15.2ms	87.4ms	260.1ms
ShuffleNet 1× ($g = 3$)	32.6	140M	37.8ms	222.2ms	684.5ms
ShuffleNet 2× ($g = 3$)	26.3	524M	108.8ms	617.0ms	1857.6ms
AlexNet [21]	42.8	720M	184.0ms	1156.7ms	3633.9ms
1.0 MobileNet-224 [12]	29.4	569M	110.0ms	612.0ms	1879.2ms

Xiangyu Z, Xinyu Z, Mengxiao L, et al. Shufflenet: an extremely efficient convolutional neural network for mobile devices[C]//Computer Vision and Pattern Recognition. 2017.

课后思考

1. 阅读DenseNet和SENet源码：
   https://github.com/pytorch/vision/tree/master/torchvision/models
2. 实现Xception结构
3. 阅读MobileNet_v2和ShuffleNet_v2论文

欢迎关注AI300学院