

# Reactor

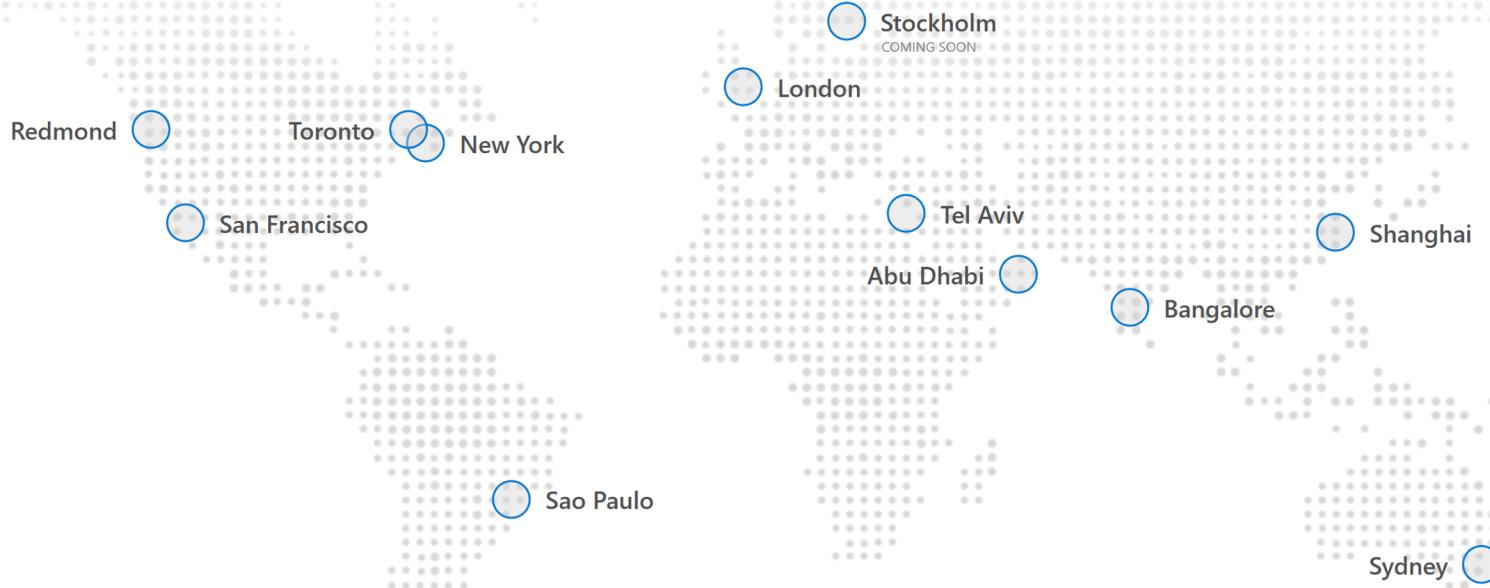
一起学人工智能系列  
- 卷积神经网络

---

2021-12-15



# Map



# 个人介绍



## Kinfey Lo - (卢建晖)

Microsoft Cloud Advocate

前微软MVP、Xamarin MVP和微软RD，拥有超过10年的云原生、人工智能和移动应用经验，为教育、金融和医疗提供应用解决方案。 Microsoft Ignite , TechEd 会议讲师，Microsoft AI 黑客马拉松教练，目前在微软，为技术人员和不同行业宣讲技术和相关应用场景。



爱编程(Python , C# , TypeScript , Swift , Rust , Go )

专注于人工智能，云原生，跨平台移动开发

Github : <https://github.com/kinfey>

Email : [kinfeylo@microsoft.com](mailto:kinfeylo@microsoft.com) Blog : <https://blog.csdn.net/kinfey>

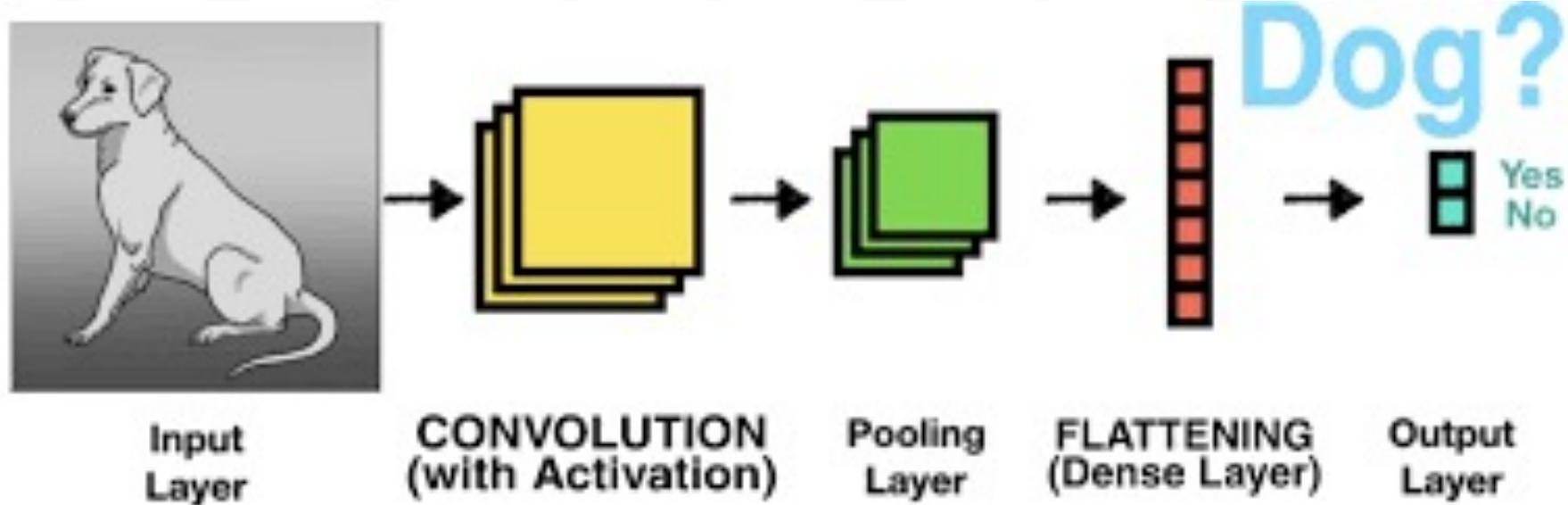
Twitter : @Ljh8304

# 卷积神经网络原理



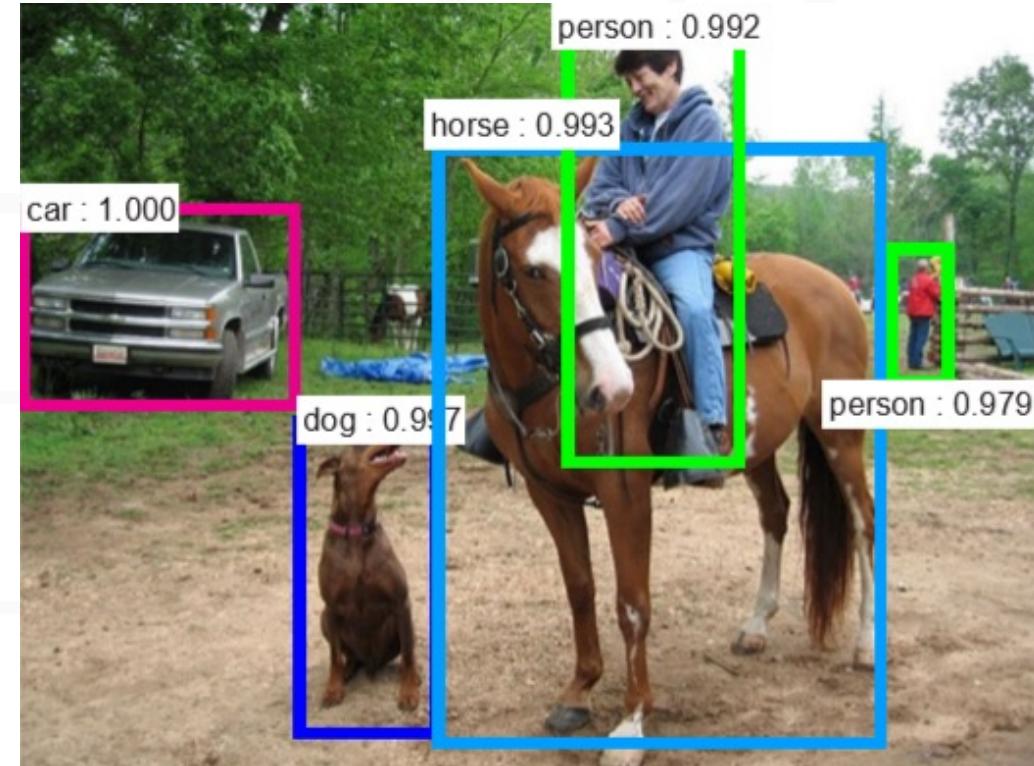
# 卷积神经网络

卷积神经网络是深度学习中的一个里程碑式的技术，有了这个技术，才会让计算机有能力理解图片和视频信息，才会有计算机视觉的众多应用。

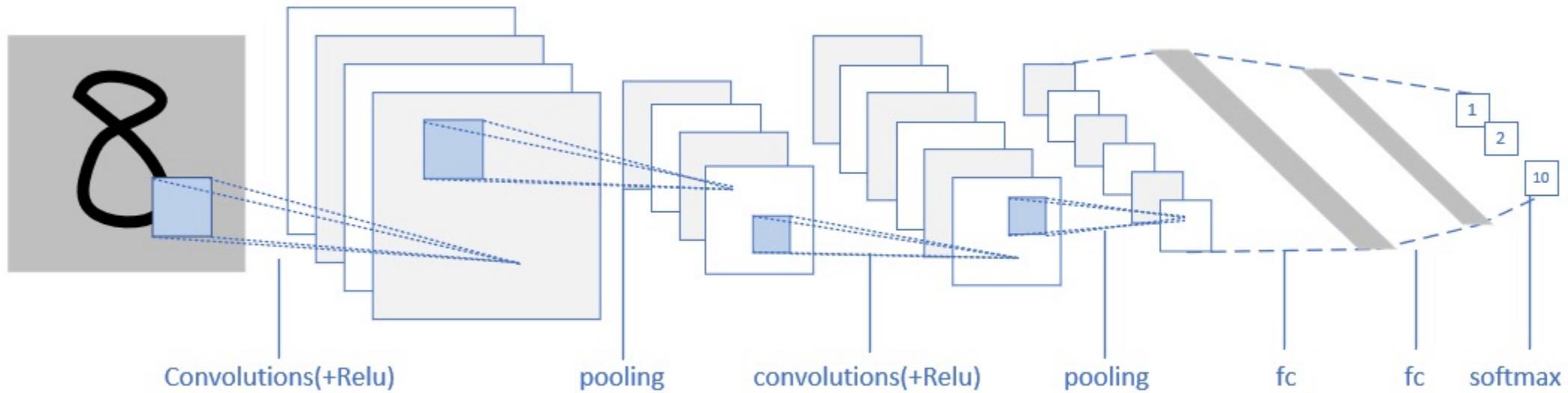


# 卷积神经网络原理

卷积神经网络 (CNN, Convolutional Neural Net)是神经网络的类型之一，在图像识别和分类领域中取得了非常好的效果，比如识别人脸、物体、交通标识等，这就为机器人、自动驾驶等应用提供了坚实的技术基础。

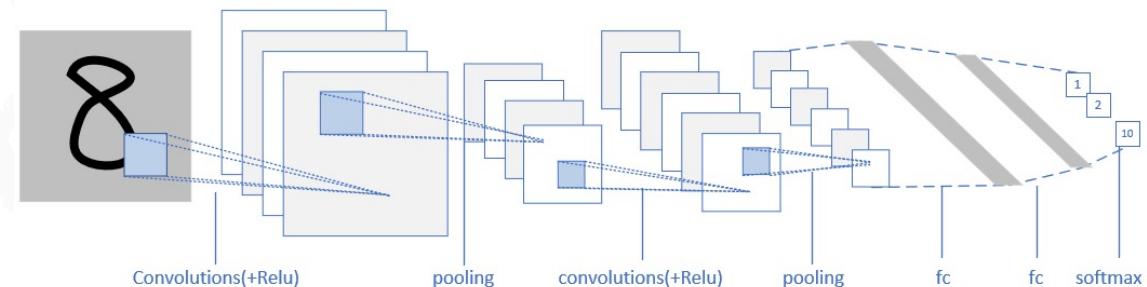


# 卷积神经网络的典型结构



# 卷积神经网络的典型结构

1. 原始的输入是一张图片，可以是彩色的，也可以是灰度的或黑白的。这里假设是只有一个通道的图片，目的是识别0~9的手写体数字；
2. 第一层卷积，我们使用了4个卷积核，得到了4张feature map； 激活函数层没有单独画出来，这里我们紧接着卷积操作使用了Relu激活函数；
3. 第二层是池化，使用了Max Pooling方式，把图片的高宽各缩小一倍，但仍然是4个feature map；
4. 第三层卷积，我们使用了 $4 \times 6$ 个卷积核，其中4对应着输入通道，6对应着输出通道，从而得到了6张feature map，当然也使用了Relu激活函数；
5. 第四层再次做一次池化，现在得到的图片尺寸只是原始尺寸的四分之一左右；
6. 第五层把第四层的6个图片展平成一维，成为一个fully connected层；
7. 第六层再接一个小一些的fully connected层；
8. 最后接一个softmax函数，判别10个分类。



# 一个典型的卷积神经网络

卷积层

激活函数层

池化层

全连接分类层

# 卷积核作用

卷积网络之所以能工作，完全是卷积核的功劳

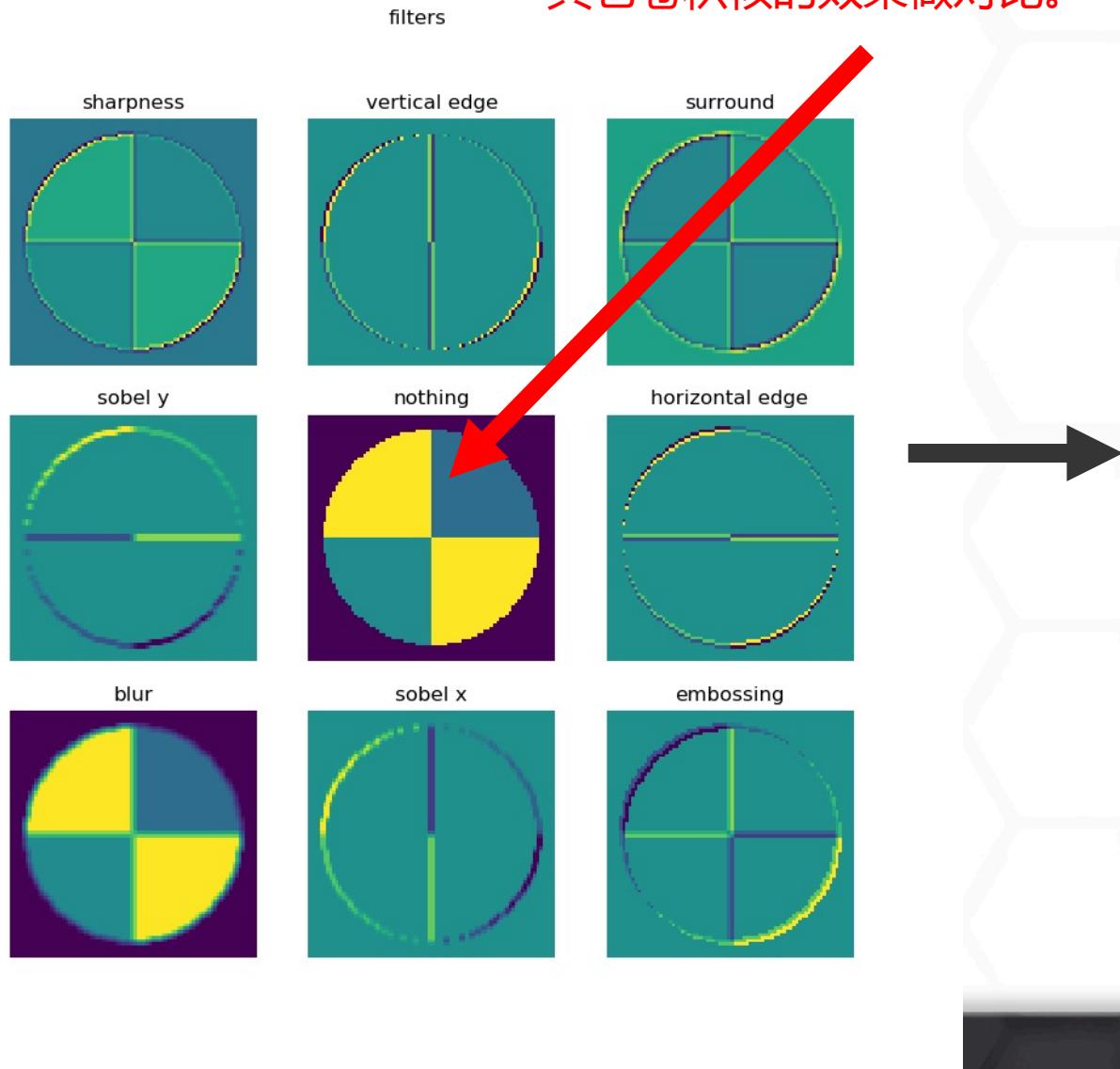
1	1.1	0.23	-0.45
2	0.1	-2.1	1.24
3	0.74	-1.32	0.01



这是一个 $3 \times 3$ 的卷积核，还会有 $1 \times 1$ 、 $5 \times 5$ 、 $7 \times 7$ 、 $9 \times 9$ 、 $11 \times 11$ 的卷积核。在卷积层中，我们会用输入数据与卷积核相乘，得到输出数据，就类似全连接层中的Weights一样，所以卷积核里的数值，也是通过反向传播的方法学习到的。

# 卷积核作用

"nothing"。为什么叫nothing呢？因为这个卷积核在与原始图片计算后得到的结果，和原始图片一模一样，所以我们看到的图5就是相当于原始图片，放在中间是为了方便和其它卷积核的效果做对比。



	1	2	3
1	0,-1, 0 -1, 5,-1 0,-1, 0	0, 0, 0 -1, 2,-1 0, 0, 0	1, 1, 1 1,-9, 1 1, 1, 1
	sharpness	vertical edge	surround
2	-1,-2, -1 0, 0, 0 1, 2, 1	0, 0, 0 0, 1, 0 0, 0, 0	0,-1, 0 0, 2, 0 0,-1, 0
	sobel y	nothing	horizontal edge
3	0.11,0.11,0.11 0.11,0.11,0.11 0.11,0.11,0.11	-1, 0, 1 -2, 0, 2 -1, 0, 1	2, 0, 0 0,-1, 0 0, 0,-1
	blur	sobel x	embossing

# 卷积核作用

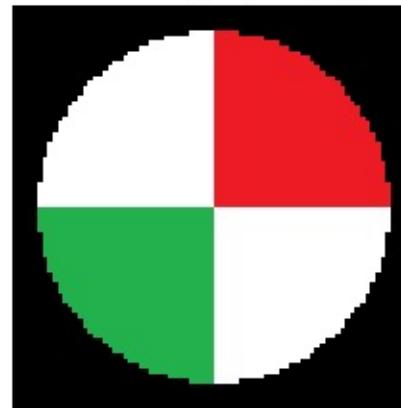
序号	名称	说明
1	锐化	如果一个像素点比周围像素点亮，则此算子会令其更亮
2	检测竖边	检测出了十字线中的竖线，由于是左侧和右侧分别检查一次，所以得到两条颜色不一样的竖线
3	周边	把周边增强，把同色的区域变弱，形成大色块
4	Sobel-Y	纵向亮度差分可以检测出横边，与横边检测不同的是，它可以使得两条横线具有相同的颜色，具有分割线的效果
5	Identity	中心为1四周为0的过滤器，卷积后与原图相同
6	横边检测	检测出了十字线中的横线，由于是上侧和下侧分别检查一次，所以得到两条颜色不一样的横线
7	模糊	通过把周围的点做平均值计算而“杀富济贫”造成模糊效果
8	Sobel-X	横向亮度差分可以检测出竖边，与竖边检测不同的是，它可以使得两条竖线具有相同的颜色，具有分割线的效果
9	浮雕	形成大理石浮雕般的效果

# 卷积后续的运算

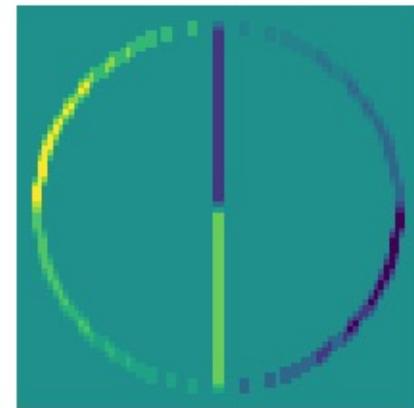
卷积神经网络通过反向传播而令卷积核自我学习，找到分布在图片中的不同的feature，最后形成的卷积核中的数据。但是如果想达到这种效果，只有卷积层的话是不够的，还需要激活函数、池化等操作的配合。

conv-relu-pool

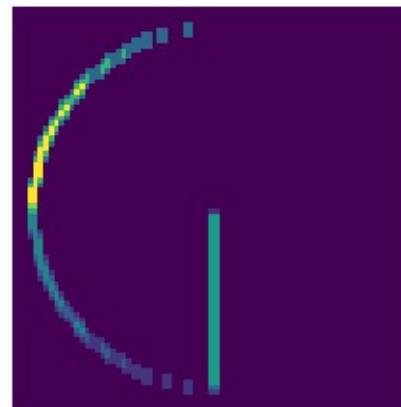
source:(80, 80, 3)



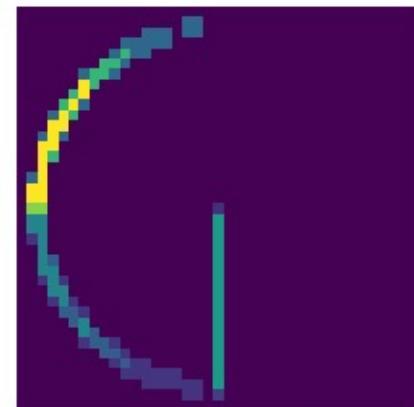
conv:(1, 1, 78, 78)



relu:(1, 1, 78, 78)



pooling:(1, 1, 39, 39)



# 卷积后续的运算

经过卷积-激活-池化操作后的效果

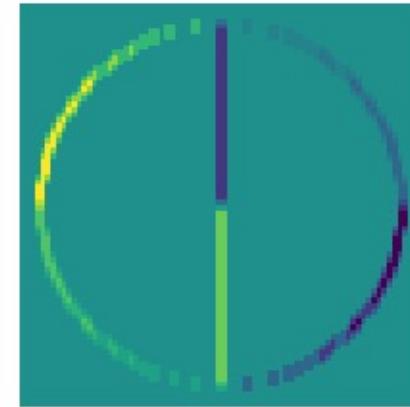
- 1.注意图一是原始图片，用cv2读取出来的图片，其顺序是反向的，即：
- 2.第一维是高度
- 3.第二维是宽度
- 4.第三维是彩色通道数，但是其顺序为BGR，而不是常用的RGB
- 5.我们对原始图片使用了一个 $3 \times 1 \times 3 \times 3$ 的卷积核，因为原始图片为彩色图片，所以第一个维度是3，对应RGB三个彩色通道；我们希望只输出一张feature map，以便于说明，所以第二维是1；我们使用了 $3 \times 3$ 的卷积核，用的是sobel x算子。所以图二是卷积后的结果。
- 6.图三做了一层Relu激活计算，把小于0的值都去掉了，只留下了一些边的特征。
- 7.图四是图三的四分之一大小，虽然图片缩小了，但是特征都没有丢失，反而因为图像尺寸变小而变得密集，亮点的密度要比图三大而粗。

conv-relu-pool

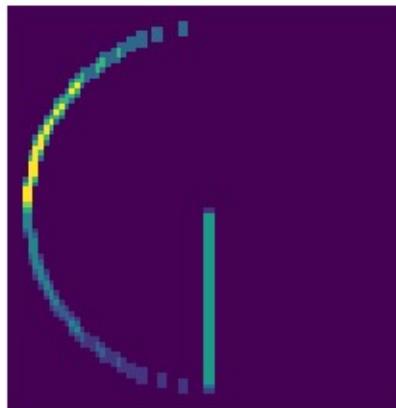
source:(80, 80, 3)



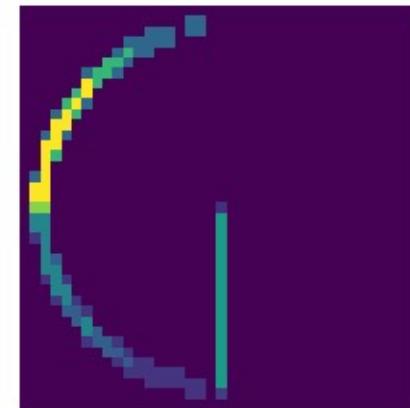
conv:(1, 1, 78, 78)



relu:(1, 1, 78, 78)



pooling:(1, 1, 39, 39)

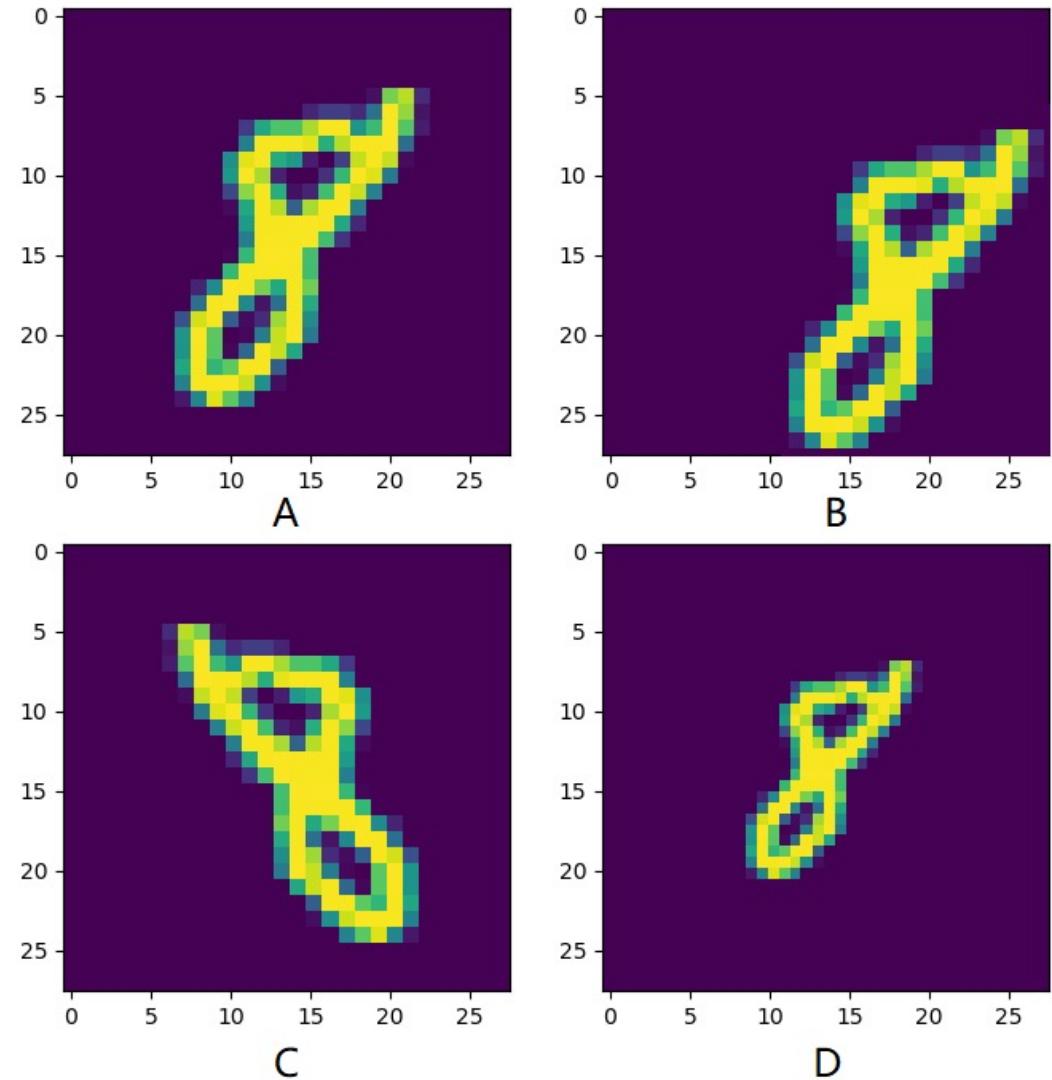


# 卷积神经网络的学习

[8]

在卷积-池化等一些列操作的后面，要接全连接层，这里的全连接层和我们在前面学习的深度网络的功能一模一样，都是做为分类层使用。

在最后一层的池化后面，把所有特征数据变成一个一维的全连接层，然后就和普通的深度全连接网络一样了，通过在最后一层的softmax分类函数，以及多分类交叉熵函数，对比图片的OneHot编码标签，回传误差值，从全连接层传回到池化层，通过激活函数层再回传给卷积层，对卷积核的数值进行梯度更新，实现卷积核数值的自我学习。



# 卷积神经网络的学习

[8]

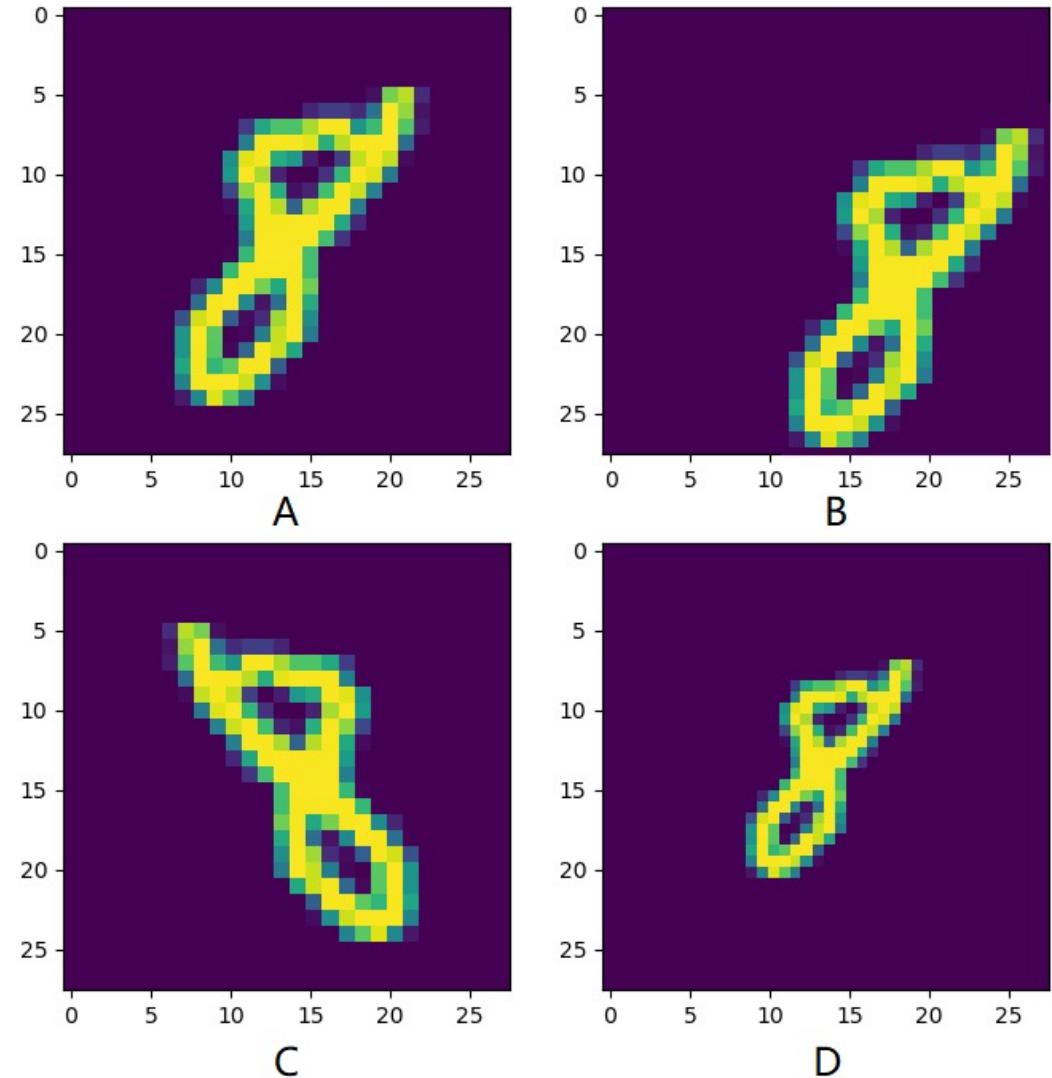
同一个背景下数字8的大小、位置、形状的不同  
我们的问题是：

如果这个“8”的位置很大地偏移到了右下角，使得左侧留出来一大片空白，即发生了平移，如上图右上角子图B

“8”做了一些旋转或者翻转，即发生了旋转视角，  
如上图左下角子图C

“8”缩小了很多或放大了很多，即发生了尺寸变化，  
如上图右下角子图D

尽管发生了变化，但是对于人类的视觉系统来说都可以轻松应对，即平移不变性、旋转视角不变性、尺度不变性。那么卷积神经网络如何处理呢？



# 卷积的前向计算



# 卷积的前向计算 – 数学定义

连续定义

$$h(x) = (f * g)(x) = \int_{-\infty}^{\infty} f(t)g(x - t)dt \quad (1)$$

卷积与傅里叶变换有着密切的关系。利用这点性质，即两函数的傅里叶变换的乘积等于它们卷积后的傅里叶变换，能使傅里叶分析中许多问题的处理得到简化。

离散定义  $\oplus$

$$h(x) = (f * g)(x) = \sum_{t=-\infty}^{\infty} f(t)g(x - t) \quad (2)$$

# 卷积的前向计算 – 一维卷积

有两枚骰子 $f, g$ , 掷出后二者相加为4的概率如何计算?

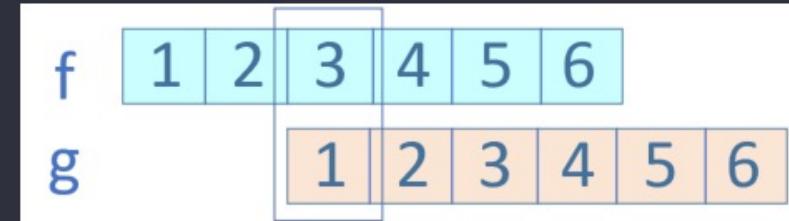
因此, 两枚骰子点数加起来为4的概率为:

$$\begin{aligned} h(4) &= f(1)g(3) + f(2)g(2) + f(3)g(1) \\ &= f(1)g(4-1) + f(2)g(4-2) + f(3)g(4-3) \end{aligned}$$

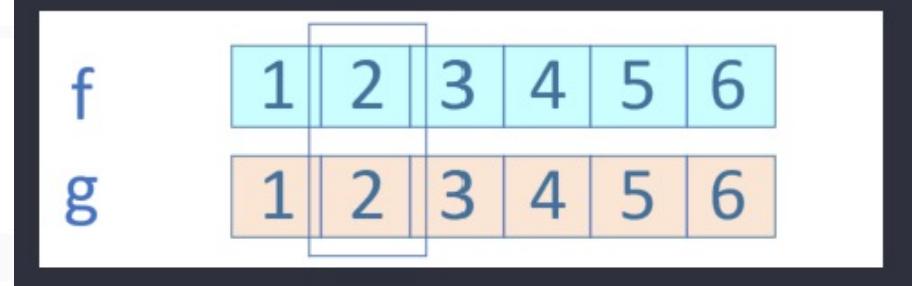
符合卷积的定义, 把它写成标准的形式就是公式2:

$$h(4) = (f * g)(4) = \sum_{t=1}^3 f(t)g(4-t)$$

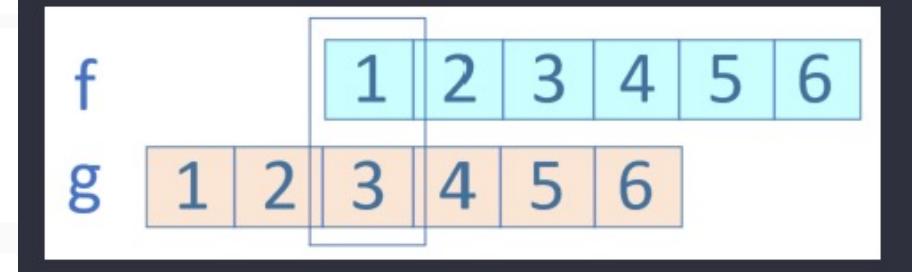
第一种情况:  $f(1)g(3), 3+1=4$ , 如图17-9所示。



第二种情况:  $f(2)g(2), 2+2=4$ , 如图17-10所示。



第三种情况:  $f(3)g(1), 1+3=4$ , 如图17-11所示。



# 卷积的前向计算 – 单入单出的二维卷积

二维卷积一般用于图像处理上。在二维图片上做卷积，如果把图像Image简写为 $I$ ，把卷积核Kernel简写为 $K$ ，则目标图片的第 $(i, j)$ 个像素的卷积值为：

$$h(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n) \quad (3)$$

可以看出，这和一维情况下的公式2是一致的。从卷积的可交换性，我们可以把公式3等价地写作：

$$h(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n) \quad (4)$$

公式4的成立，是因为我们将Kernel进行了翻转。在神经网络中，一般会实现一个互相关函数(corresponding function)，而卷积运算几乎一样，但不反转Kernel：

$$h(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n) \quad (5)$$

在图像处理中，自相关函数和互相关函数定义如下：

- 自相关：设原函数是 $f(t)$ ，则 $h = f(t) \star f(-t)$ ，其中 $\star$ 表示卷积
- 互相关：设两个函数分别是 $f(t)$ 和 $g(t)$ ，则 $h = f(t) \star g(-t)$

# 卷积的前向计算 – 单入单出的二维卷积

互相关函数的运算，是两个序列滑动相乘，两个序列都不翻转。卷积运算也是滑动相乘，但是其中一个序列需要先翻转，再相乘。所以，从数学意义上说，机器学习实现的是互相关函数，而不是原始意义上的卷积。但我们为了简化，把公式5也称作为卷积。这就是卷积的来源。

结论：

1. 我们实现的卷积操作不是原始数学含义的卷积，而是工程上的卷积，可以简称为卷积
2. 在实现卷积操作时，并不会反转卷积核

在传统的图像处理中，卷积操作多用来进行滤波，锐化或者边缘检测啥的。我们可以认为卷积是利用某些设计好的参数组合（卷积核）去提取图像空域上相邻的信息。

$$\begin{array}{|c|c|c|c|} \hline 2 & 5 & 0 & 1 \\ \hline 6 & 7 & 1 & 2 \\ \hline 3 & 0 & 4 & 0 \\ \hline 3 & 9 & 7 & 4 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 0 & 1 \\ \hline -1 & 1 & 0 \\ \hline 0 & 1 & -1 \\ \hline \end{array} = \begin{array}{|c|c|} \hline -1 & \\ \hline & \\ \hline \end{array}$$

$$\begin{aligned}
 & 2 \times 1 + 5 \times 0 + 0 \times 1 + \\
 & 6 \times (-1) + 7 \times 1 + 1 \times 0 + \\
 & 3 \times 0 + 0 \times 1 + 4 \times (-1) \\
 & = -1
 \end{aligned}$$

$$\begin{array}{|c|c|c|c|} \hline 2 & 5 & 0 & 1 \\ \hline 6 & 7 & 1 & 2 \\ \hline 3 & 0 & 4 & 0 \\ \hline 3 & 9 & 7 & 4 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 0 & 1 \\ \hline -1 & 1 & 0 \\ \hline 0 & 1 & -1 \\ \hline \end{array} = \begin{array}{|c|c|} \hline -1 & 4 \\ \hline & \\ \hline \end{array}$$

$$\begin{aligned}
 & 5 \times 1 + 0 \times 0 + 1 \times 1 + \\
 & 7 \times (-1) + 1 \times 1 + 2 \times 0 + \\
 & 0 \times 0 + 4 \times 1 + 0 \times (-1) \\
 & = 4
 \end{aligned}$$

$$\begin{array}{|c|c|c|c|} \hline 2 & 5 & 0 & 1 \\ \hline 6 & 7 & 1 & 2 \\ \hline 3 & 0 & 4 & 0 \\ \hline 3 & 9 & 7 & 4 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 0 & 1 \\ \hline -1 & 1 & 0 \\ \hline 0 & 1 & -1 \\ \hline \end{array} = \begin{array}{|c|c|} \hline -1 & 4 \\ \hline & 6 \\ \hline \end{array}$$

$$\begin{aligned}
 & 6 \times 1 + 7 \times 0 + 1 \times 1 + \\
 & 3 \times (-1) + 0 \times 1 + 4 \times 0 + \\
 & 3 \times 0 + 9 \times 1 + 7 \times (-1) \\
 & = 6
 \end{aligned}$$

$$\begin{array}{|c|c|c|c|} \hline 2 & 5 & 0 & 1 \\ \hline 6 & 7 & 1 & 2 \\ \hline 3 & 0 & 4 & 0 \\ \hline 3 & 9 & 7 & 4 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 0 & 1 \\ \hline -1 & 1 & 0 \\ \hline 0 & 1 & -1 \\ \hline \end{array} = \begin{array}{|c|c|} \hline -1 & 4 \\ \hline & 6 \\ \hline & 16 \\ \hline \end{array}$$

$$\begin{aligned}
 & 7 \times 1 + 1 \times 0 + 2 \times 1 + \\
 & 0 \times (-1) + 4 \times 1 + 0 \times 0 + \\
 & 9 \times 0 + 7 \times 1 + 4 \times (-1) \\
 & = 16
 \end{aligned}$$

# 卷积的前向计算 – 单入多出的升维卷积

原始输入是一维的图片，但是我们可以用多个卷积核分别对其进行计算，从而得到多个特征输出。如图17-13所示。

The diagram illustrates 2D convolution with one input channel and two output channels. The input is a 4x4 matrix:

$$\begin{bmatrix} 2 & 5 & 0 & 1 \\ 6 & 7 & 1 & 2 \\ 3 & 0 & 4 & 0 \\ 3 & 9 & 7 & 4 \end{bmatrix}$$

The first output channel is calculated using the following kernel:

$$\begin{bmatrix} 1 & 0 & 1 \\ -1 & 1 & 0 \\ 0 & 1 & -1 \end{bmatrix}$$

The second output channel is calculated using the following kernel:

$$\begin{bmatrix} -1 & 1 & 0 \\ 0 & 1 & -1 \\ 1 & 0 & 1 \end{bmatrix}$$

The resulting feature maps are:

$$\begin{bmatrix} -1 & 4 \\ 6 & 16 \end{bmatrix}$$
$$\begin{bmatrix} 16 & -6 \\ 7 & 11 \end{bmatrix}$$

图17-13 单入多出的升维卷积

一张 $4 \times 4$ 的图片，用两个卷积核并行地处理，输出为2个 $2 \times 2$ 的图片。在训练过程中，这两个卷积核会完成不同的特征学习。

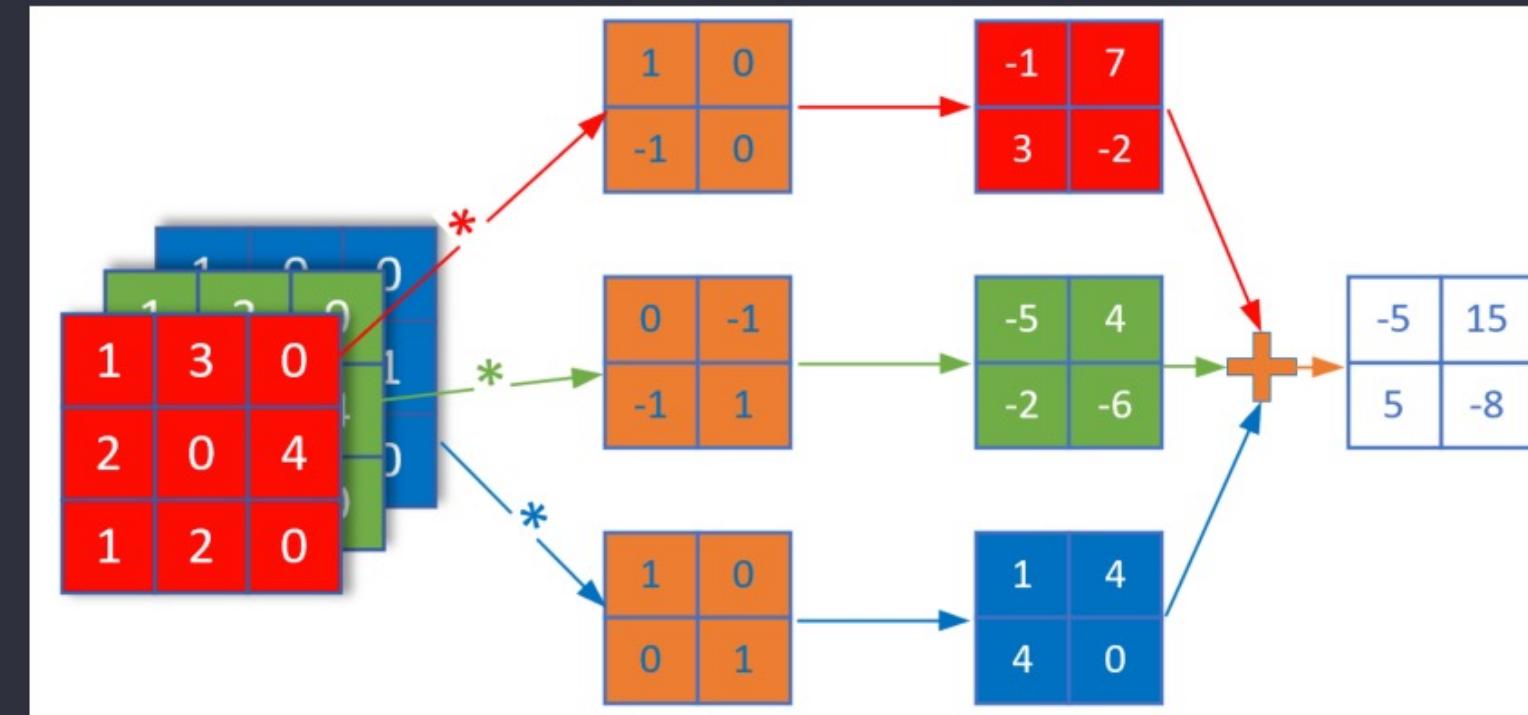
# 卷积的前向计算 – 多入单出的降维卷积



一张图片，通常是彩色的，具有红绿蓝三个通道。我们可以有两个选择来处理：

1. 变成灰度的，每个像素只剩下一个值，就可以用二维卷积
2. 对于三个通道，每个通道都使用一个卷积核，分别处理红绿蓝三种颜色的信息

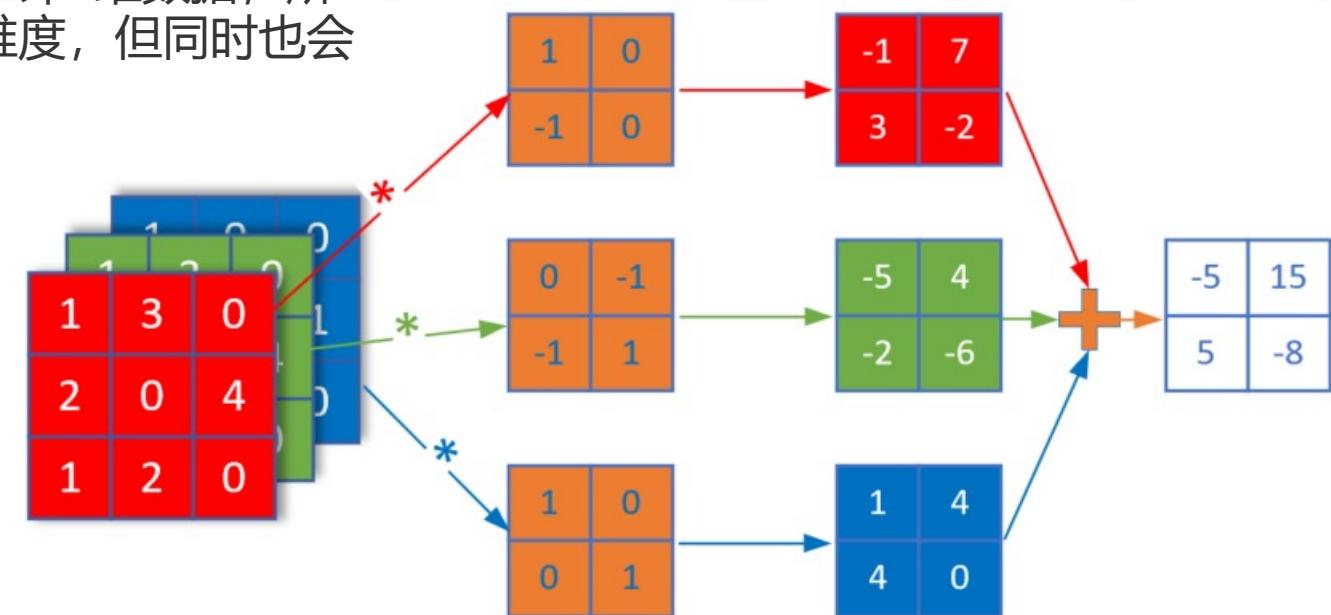
显然第2种方法可以从图中学习到更多的特征，于是出现了三维卷积，即有三个卷积核分别对应图的三个通道，三个子核的尺寸是一样的，比如都是 $2 \times 2$ ，这样的话，这三个卷积核就是一个 $3 \times 2 \times 2$ 的立体核，称为过滤器Filter，所以称为三维卷积。



# 卷积的前向计算 – 多入单出的降维卷积

每一个卷积核对应着左侧相同颜色的输入通道，三个过滤器的值并不一定相同。对三个通道各自做卷积后，得到右侧的三张特征图，然后再按照原始值不加权地相加在一起，得到最右侧的白色特征图，这张图里面已经把三种颜色的特征混在一起了，所以画成了白色，表示没有颜色特征了。

虽然输入图片是多个通道的，或者说是三维的，但是在相同数量的过滤器的计算后，相加在一起的结果是一个通道，即2维数据，所以称为降维。这当然简化了对多通道数据的计算难度，但同时也会损失多通道数据自带的颜色信息。



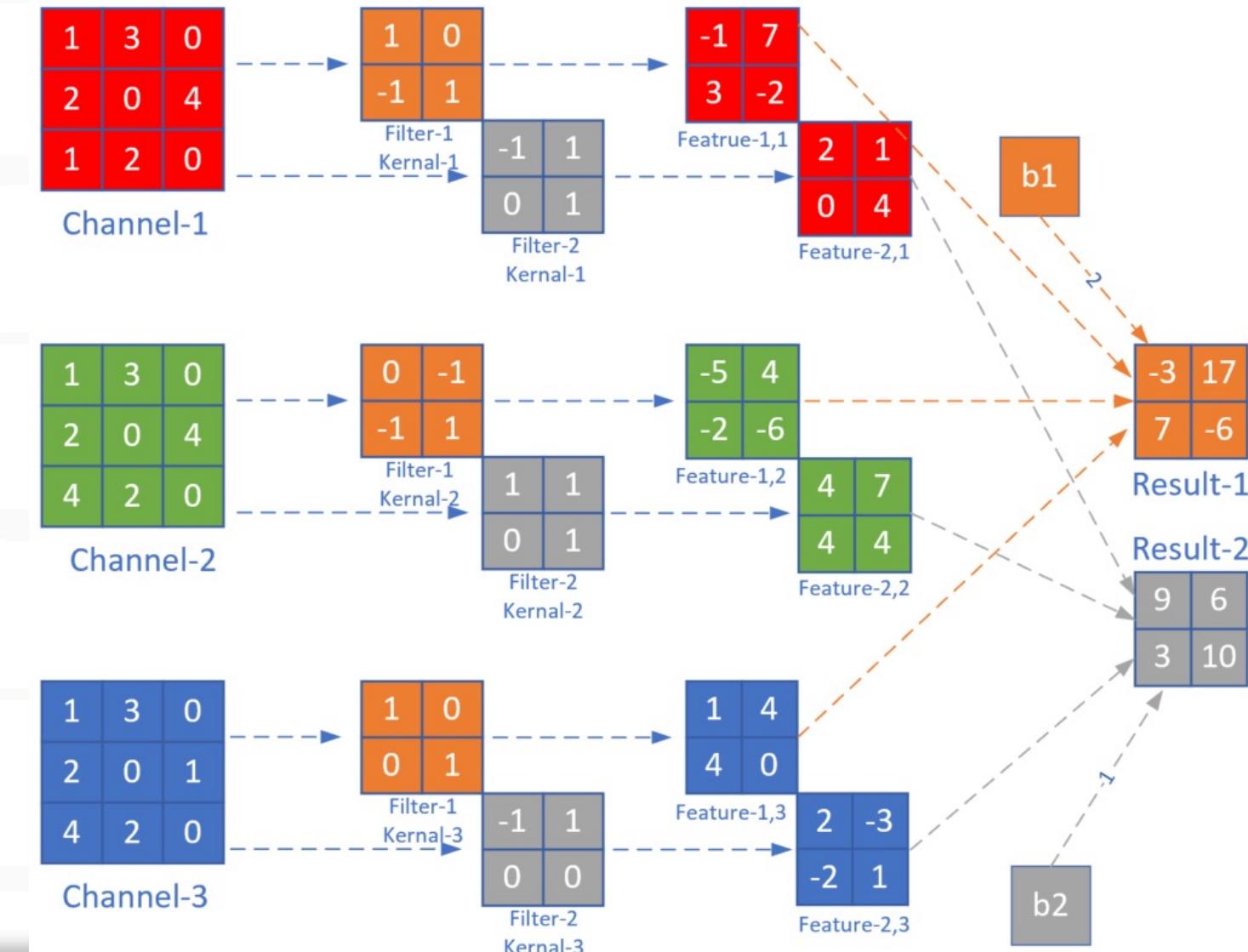
# 卷积的前向计算 – 多入多出的同维卷积

一个过滤器Filter内含三个卷积核Kernel。我们假设有一个彩色图片为 $3 \times 3$ 的，如果有两组 $3 \times 2 \times 2$ 的卷积核的话，会做什么样的卷积计算？

第一个过滤器Filter-1为棕色所示，它有三个卷积核(Kernal)，命名为Kernel-1, Kernel-2, Kernel-3，分别在红绿蓝三个输入通道上进行卷积操作，生成三个 $2 \times 2$ 的输出Feature-1,n。然后三个Feature-1,n相加，并再加上b1偏移值，形成最后的棕色输出Result-1。

对于灰色的过滤器Filter-2也是一样，先生成三个Feature-2,n，然后相加再加b2，最后得到Result-2。

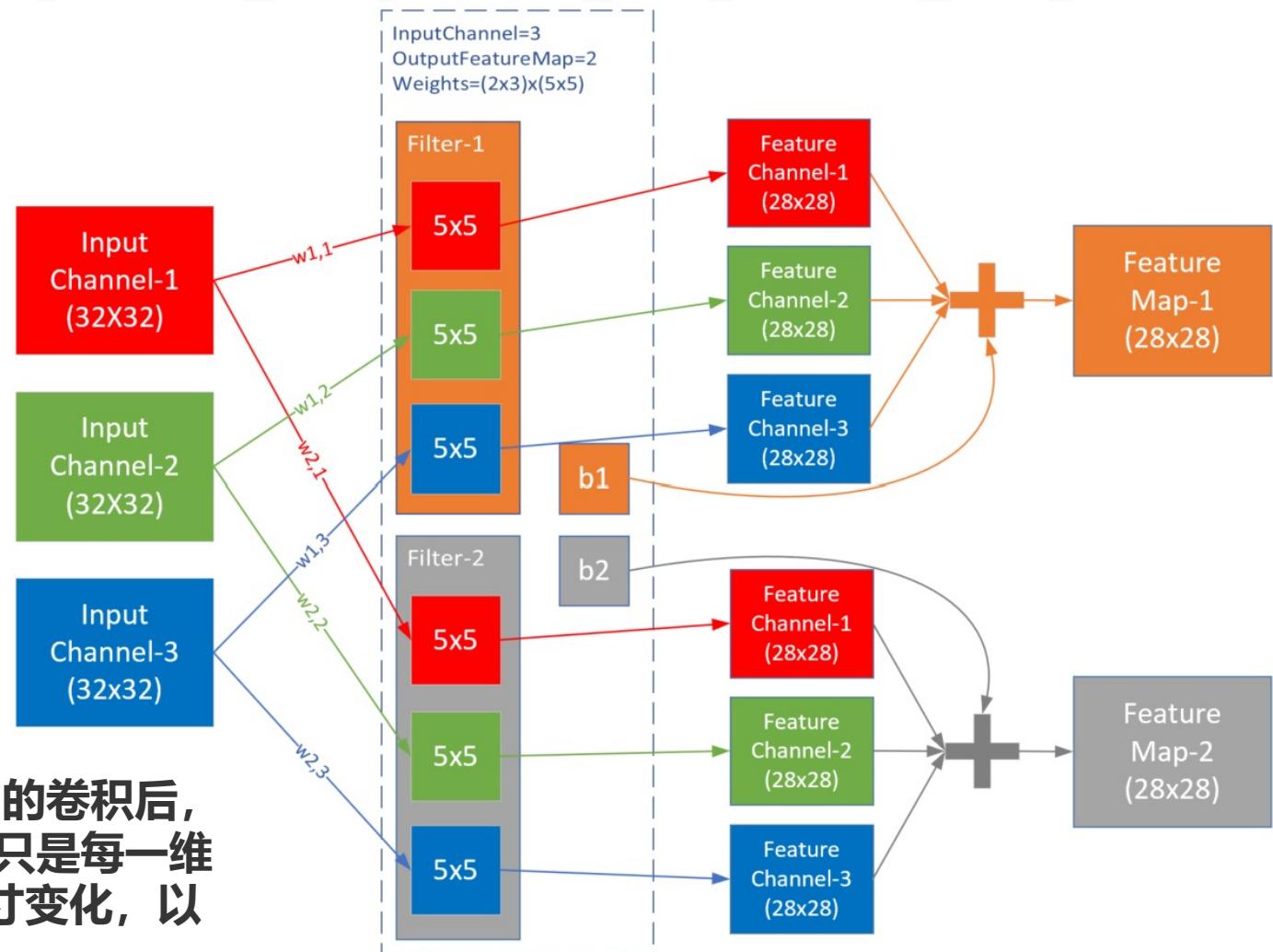
之所以Feature-m,n还用红绿蓝三色表示，是因为在此时，它们还保留着红绿蓝三种色彩的各自的信息，一旦相加后得到Result，这种信息就丢失了



# 卷积的前向计算 – 卷积编程模型

- 输入 Input Channel
- 卷积核组 WeightsBias
- 过滤器 Filter
- 卷积核 kernel
- 输出 Feature Map

输入是三维数据 (3x32x32) , 经过 $2 \times 3 \times 5 \times 5$ 的卷积后,输出为三维 (2x28x28) , 维数并没有变化, 只是每一维内部的尺寸有了变化, 一般都是要向更小的尺寸变化, 以便于简化计算。



# 卷积的前向计算 – 卷积编程模型

对于三维卷积，有以下特点：

预先定义输出的feature map的数量，而不是根据前向计算自动计算出来，此例中为2，这样就会有两组WeightsBias

对于每个输出，都有一个对应的过滤器Filter，此例中Feature Map-1对应Filter-1

每个Filter内都有一个或多个卷积核Kernel，对应每个输入通道(Input Channel)，此例为3，对应输入的红绿蓝三个通道

每个Filter只有一个Bias值，Filter-1对应b1，Filter-2对应b2

卷积核Kernel的大小一般是奇数如：1x1, 3x3, 5x5, 7x7等，此例为5x5

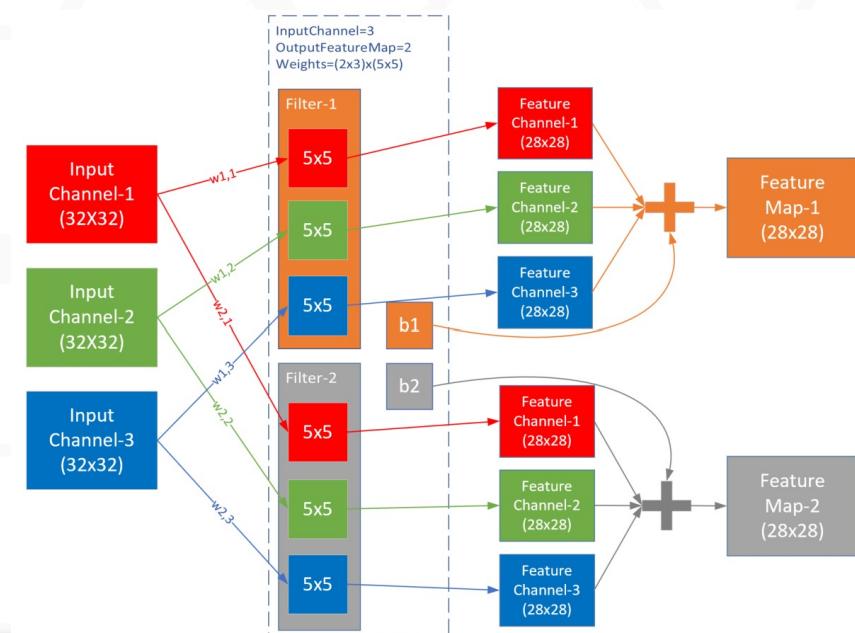
我们可以用在全连接神经网络中的学到的知识来理解：

每个Input Channel就是特征输入，在上图中是3个

卷积层的卷积核相当于隐层的神经元，上图中隐层有2个神经元

$W(m,n), m=[1,2], n=[1,3]$ 相当于隐层的权重矩阵  $w_{11}, w_{12}, \dots$

每个卷积核（神经元）有1个偏移值



# 卷积的前向计算 – 步长 stride

前面的例子中，每次计算后，卷积核会向右或者向下移动一个单元，即步长  $\text{stride} = 1$ 。而在图17-17这个卷积操作中，卷积核每次向右或向下移动两个单元，即  $\text{stride} = 2$ 。

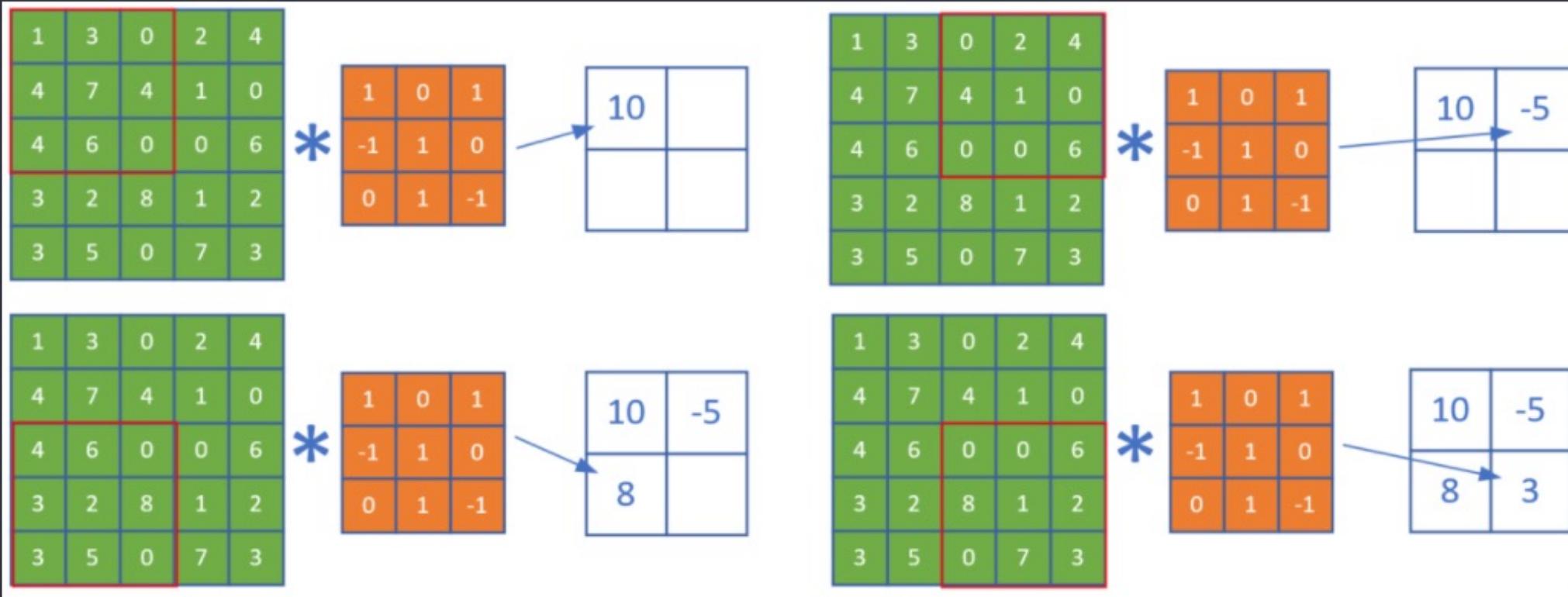


图17-17 步长为2的卷积

在后续的步骤中，由于每次移动两格，所以最终得到一个 $2 \times 2$ 的图片。

# 卷积的前向计算 – 填充 padding

如果原始图为 $4 \times 4$ , 用 $3 \times 3$ 的卷积核进行卷积后, 目标图片变成了 $2 \times 2$ 。如果我们想保持目标图片和原始图片为同样大小, 该怎么办呢? 一般我们会向原始图片周围填充一圈0, 然后再做卷积。如图17-18。

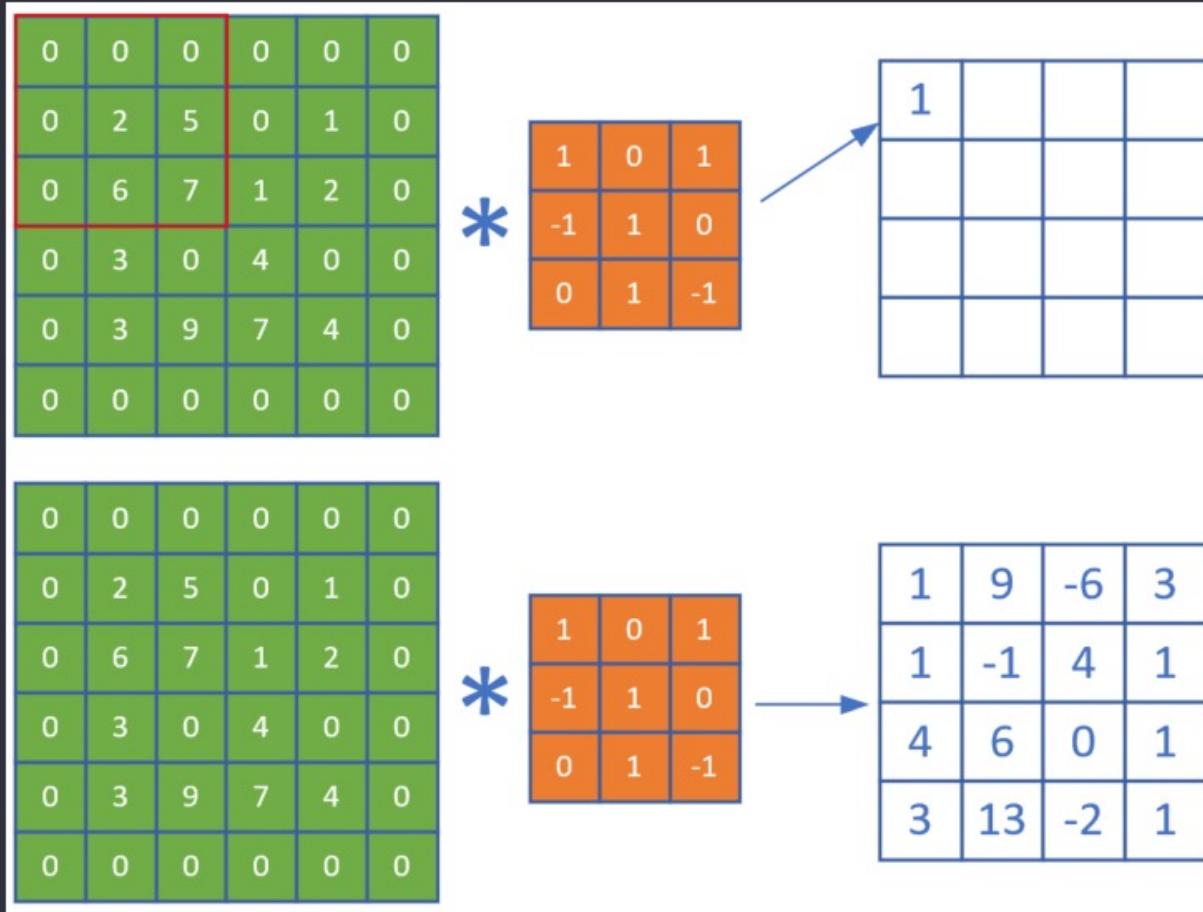


图17-18 带填充的卷积

# 卷积的前向计算 – 输出结果

综合以上所有情况，可以得到卷积后的输出图片的大小的公式：

$$H_{Output} = \frac{H_{Input} - H_{Kernel} + 2Padding}{Stride} + 1$$

$$W_{Output} = \frac{W_{Input} - W_{Kernel} + 2Padding}{Stride} + 1$$

以图17-17为例：

$$H_{Output} = \frac{5 - 3 + 2 \times 0}{2} + 1 = 2$$

以图17-18为例：

$$H_{Output} = \frac{4 - 3 + 2 \times 1}{1} + 1 = 4$$

两点注意：

1. 一般情况下，我们用正方形的卷积核，且为奇数
2. 如果计算出的输出图片尺寸为小数，则取整，不做四舍五入

# 卷积的反向传播原理



# 卷积层的训练

同全连接层一样，卷积层的训练也需要从上一层回传的误差矩阵，然后计算：

- 1.本层的权重矩阵的误差项
- 2.本层的需要回传到下一层的误差矩阵

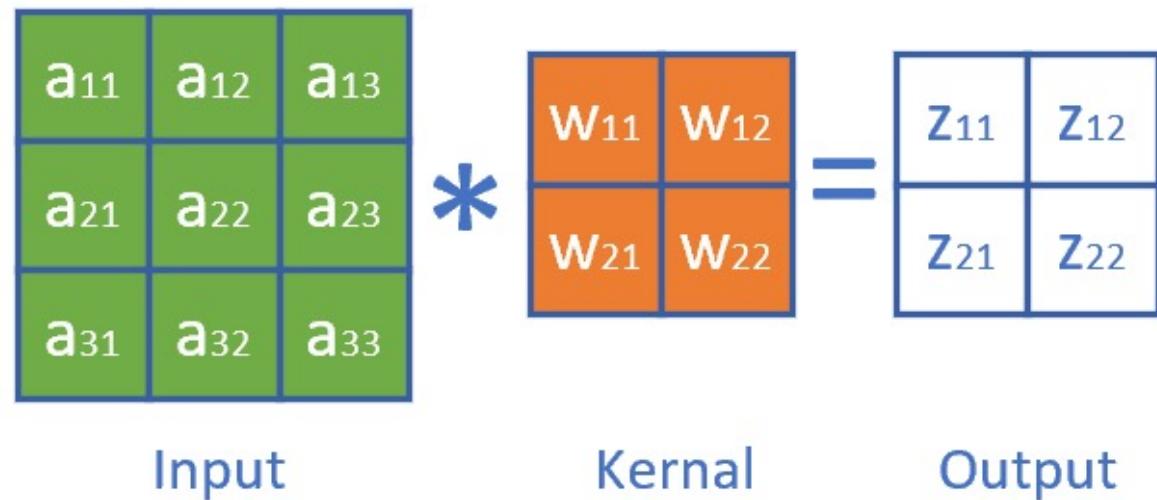
# 计算反向传播的梯度矩阵



正向公式：

$$Z = W * A + b \quad (0)$$

其中，W是卷积核，\*表示卷积（互相关）计算，A为当前层的输入项，b是偏移（未在图中画出），Z为当前层的输出项，但尚未经过激活函数处理。



# 计算反向传播的梯度矩阵

求损失函数 $J$ 对 $a_{11}$ 的梯度：

$$\frac{\partial J}{\partial a_{11}} = \frac{\partial J}{\partial z_{11}} \frac{\partial z_{11}}{\partial a_{11}} = \delta_{z11} \cdot w_{11} \quad (5)$$

上式中， $\delta_{z11}$ 是从网络后端回传到本层的 $z_{11}$ 单元的梯度。

求 $J$ 对 $a_{12}$ 的梯度时，先看正向公式，发现 $a_{12}$ 对 $z_{11}$ 和 $z_{12}$ 都有贡献，因此需要二者的偏导数相加：

$$\frac{\partial J}{\partial a_{12}} = \frac{\partial J}{\partial z_{11}} \frac{\partial z_{11}}{\partial a_{12}} + \frac{\partial J}{\partial z_{12}} \frac{\partial z_{12}}{\partial a_{12}} = \delta_{z11} \cdot w_{12} + \delta_{z12} \cdot w_{11} \quad (6)$$

最复杂的是求 $a_{22}$ 的梯度，因为从正向公式看，所有的输出都有 $a_{22}$ 的贡献，所以：

$$\frac{\partial J}{\partial a_{22}} = \frac{\partial J}{\partial z_{11}} \frac{\partial z_{11}}{\partial a_{22}} + \frac{\partial J}{\partial z_{12}} \frac{\partial z_{12}}{\partial a_{22}} + \frac{\partial J}{\partial z_{21}} \frac{\partial z_{21}}{\partial a_{22}} + \frac{\partial J}{\partial z_{22}} \frac{\partial z_{22}}{\partial a_{22}} \$\$ = \delta_{z11} \cdot w_{22} + \delta_{z12} \cdot w_{21} + \delta_{z21} \cdot w_{12} + \delta_{z22} \cdot w_{11}$$

同理可得所有 $a$ 的梯度。

# 计算反向传播的梯度矩阵

观察公式7中的 $w$ 的顺序，貌似是把原始的卷积核旋转了180度，再与传入误差项做卷积操作，即可得到所有元素的误差项。而公式5和公式6并不完备，是因为二者处于角落，这和卷积正向计算中的padding是相同的现象。因此，我们把传入的误差矩阵Delta-In做一个zero padding，再乘以旋转180度的卷积核，就是要传出的误差矩阵Delta-Out，如图17-22所示。

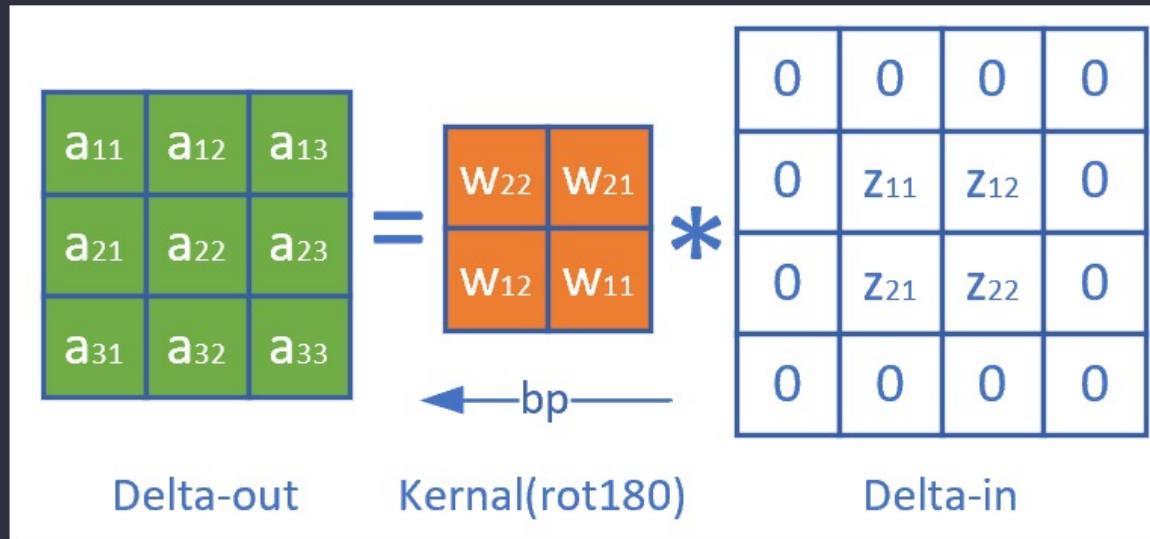


图17-22 卷积运算中的误差反向传播

最后可以统一成为一个简洁的公式：

$$\delta_{out} = \delta_{in} * W^{rot180} \quad (8)$$

这个误差矩阵可以继续回传到下一层。

- 当Weights是 $3 \times 3$ 时， $\delta_{in}$ 需要padding=2，即加2圈0，才能和Weights卷积后，得到正确尺寸的 $\delta_{out}$
- 当Weights是 $5 \times 5$ 时， $\delta_{in}$ 需要padding=4，即加4圈0，才能和Weights卷积后，得到正确尺寸的 $\delta_{out}$
- 以此类推：当Weights是 $N \times N$ 时， $\delta_{in}$ 需要padding=N-1，即加N-1圈0

# 步长不为1时的梯度矩阵还原

1	3	0	2	4
4	7	4	1	0
4	6	0	0	6
3	2	8	1	2
3	5	0	7	3

Stride=2



1	0	1
-1	1	0
0	1	-1



10	-5
8	3



10	0	-5
0	0	0
8	0	3



1	3	0	2	4
4	7	4	1	0
4	6	0	0	6
3	2	8	1	2
3	5	0	7	3

Stride=1



1	0	1
-1	1	0
0	1	-1



10	2	-5
4	9	3
8	5	3

二者的差别就是中间那个结果图的灰色部分。如果反向传播时，传入的误差矩阵是stride=2时的2x2的形状，那么我们只需要把它补上一个十字，变成3x3的误差矩阵，就可以用步长为1的算法了。

以此类推，如果步长为3时，需要补一个双线的十字。所以，当知道当前的卷积层步长为S (S>1) 时：

1. 得到从上层回传的误差矩阵形状，假设为 $M \times N$
2. 初始化一个 $(M \cdot S) \times (N \cdot S)$ 的零矩阵
3. 把传入的误差矩阵的第一行值放到零矩阵第0行的 $0, S, 2S, 3S, \dots$ 位置
4. 然后把误差矩阵的第二行的值放到零矩阵第 $S$ 行的 $0, S, 2S, 3S, \dots$ 位置
5. .....

步长为2时，用实例表示就是这样：

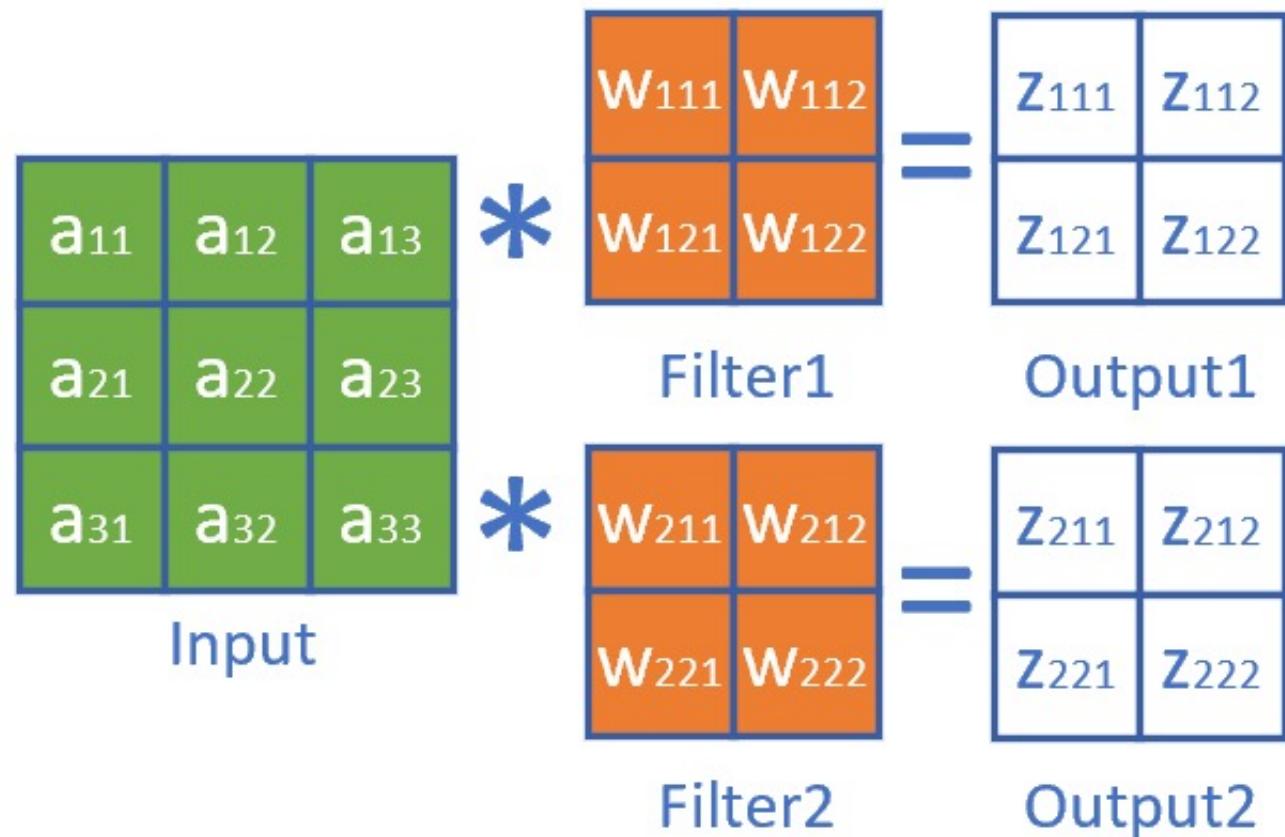
$$\begin{bmatrix} \delta_{11} & 0 & \delta_{12} & 0 & \delta_{13} \\ 0 & 0 & 0 & 0 & 0 \\ \delta_{21} & 0 & \delta_{22} & 0 & \delta_{23} \end{bmatrix}$$

步长为3时，用实例表示就是这样：

$$\begin{bmatrix} \delta_{11} & 0 & 0 & \delta_{12} & 0 & 0 & \delta_{13} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \delta_{21} & 0 & 0 & \delta_{22} & 0 & 0 & \delta_{23} \end{bmatrix}$$

# 有多个卷积核时的梯度计算

有多个卷积核也就意味着有多个输出通道。



# 有多个卷积核时的梯度计算

正向公式：

$$z_{111} = w_{111} \cdot a_{11} + w_{112} \cdot a_{12} + w_{121} \cdot a_{21} + w_{122} \cdot a_{22} \quad z_{112} = w_{111} \cdot a_{12} + w_{112} \cdot a_{13} + w_{121} \cdot a_{22} + w_{122} \cdot a_{23}$$

$$z_{211} = w_{211} \cdot a_{11} + w_{212} \cdot a_{12} + w_{221} \cdot a_{21} + w_{222} \cdot a_{22} \quad z_{212} = w_{211} \cdot a_{12} + w_{212} \cdot a_{13} + w_{221} \cdot a_{22} + w_{222} \cdot a_{23}$$

求 $J$ 对 $a_{22}$ 的梯度：

$$\begin{aligned}\frac{\partial J}{\partial a_{22}} &= \frac{\partial J}{\partial Z_1} \frac{\partial Z_1}{\partial a_{22}} + \frac{\partial J}{\partial Z_2} \frac{\partial Z_2}{\partial a_{22}} \\&= \frac{\partial J}{\partial z_{111}} \frac{\partial z_{111}}{\partial a_{22}} + \frac{\partial J}{\partial z_{112}} \frac{\partial z_{112}}{\partial a_{22}} + \frac{\partial J}{\partial z_{121}} \frac{\partial z_{121}}{\partial a_{22}} + \frac{\partial J}{\partial z_{122}} \frac{\partial z_{122}}{\partial a_{22}} \\&\quad + \frac{\partial J}{\partial z_{211}} \frac{\partial z_{211}}{\partial a_{22}} + \frac{\partial J}{\partial z_{212}} \frac{\partial z_{212}}{\partial a_{22}} + \frac{\partial J}{\partial z_{221}} \frac{\partial z_{221}}{\partial a_{22}} + \frac{\partial J}{\partial z_{222}} \frac{\partial z_{222}}{\partial a_{22}} \\&= (\delta_{z111} \cdot w_{122} + \delta_{z112} \cdot w_{121} + \delta_{z121} \cdot w_{112} + \delta_{z122} \cdot w_{111}) \\&\quad + (\delta_{z211} \cdot w_{222} + \delta_{z212} \cdot w_{221} + \delta_{z221} \cdot w_{212} + \delta_{z222} \cdot w_{211}) \\&= \delta_{z1} * W_1^{rot180} + \delta_{z2} * W_2^{rot180}\end{aligned}$$

因此和公式8相似，先在 $\delta_{in}$ 外面加padding，然后和对应的旋转后的卷积核相乘，再把几个结果相加，就得到了需要前传的梯度矩阵：

$$\delta_{out} = \sum_m \delta_{in\_m} * W_m^{rot180} \tag{9}$$

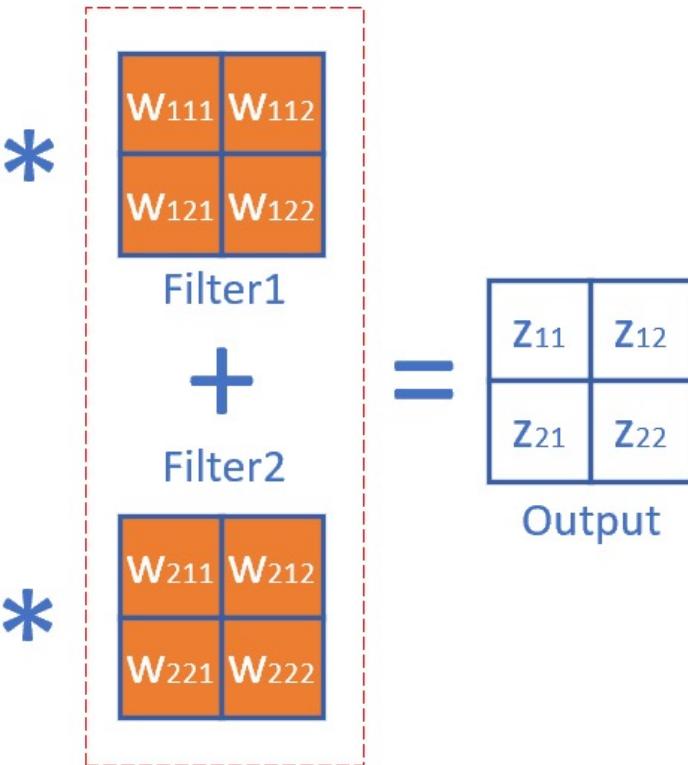
# 有多个输入时的梯度计算

$a_{111}$	$a_{112}$	$a_{113}$
$a_{121}$	$a_{122}$	$a_{123}$
$a_{131}$	$a_{132}$	$a_{133}$

InputChannel1

$a_{211}$	$a_{212}$	$a_{213}$
$a_{221}$	$a_{222}$	$a_{223}$
$a_{231}$	$a_{232}$	$a_{233}$

InputChannel2



所以有前向公式：

Multiple \tag{14}

最复杂的情况，求 $J$ 对 $a_{122}$ 的梯度：

$$\begin{aligned}\frac{\partial J}{\partial a_{111}} &= \frac{\partial J}{\partial z_{11}} \frac{\partial z_{11}}{\partial a_{122}} + \frac{\partial J}{\partial z_{12}} \frac{\partial z_{12}}{\partial a_{122}} + \frac{\partial J}{\partial z_{21}} \frac{\partial z_{21}}{\partial a_{122}} + \frac{\partial J}{\partial z_{22}} \frac{\partial z_{22}}{\partial a_{122}} \\ &= \delta_{z_{11}} \cdot w_{122} + \delta_{z_{12}} \cdot w_{121} + \delta_{z_{21}} \cdot w_{112} + \delta_{z_{22}} \cdot w_{111}\end{aligned}$$

泛化以后得到：

$$\delta_{out1} = \delta_{in} * W_1^{rot180} \quad (14)$$

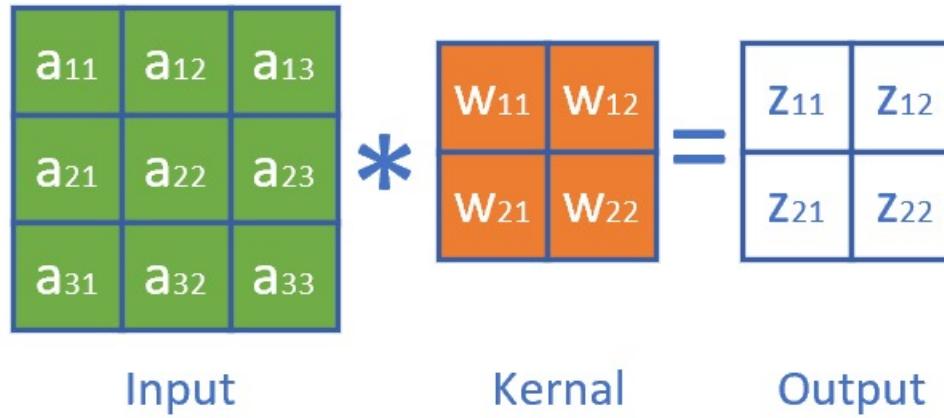
求 $J$ 对 $a_{222}$ 的梯度：

$$\begin{aligned}\frac{\partial J}{\partial a_{211}} &= \frac{\partial J}{\partial z_{11}} \frac{\partial z_{11}}{\partial a_{222}} + \frac{\partial J}{\partial z_{12}} \frac{\partial z_{12}}{\partial a_{222}} + \frac{\partial J}{\partial z_{21}} \frac{\partial z_{21}}{\partial a_{222}} + \frac{\partial J}{\partial z_{22}} \frac{\partial z_{22}}{\partial a_{222}} \\ &= \delta_{z_{11}} \cdot w_{222} + \delta_{z_{12}} \cdot w_{221} + \delta_{z_{21}} \cdot w_{212} + \delta_{z_{22}} \cdot w_{211}\end{aligned}$$

泛化以后得到：

$$\delta_{out2} = \delta_{in} * W_2^{rot180} \quad (15)$$

# 权重 (卷积核) 梯度计算



要求J对w11的梯度，从正向公式可以看到，w11对所有的z都有贡献，所以：

$$\begin{aligned}\frac{\partial J}{\partial w_{11}} &= \frac{\partial J}{\partial z_{11}} \frac{\partial z_{11}}{\partial w_{11}} + \frac{\partial J}{\partial z_{12}} \frac{\partial z_{12}}{\partial w_{11}} + \frac{\partial J}{\partial z_{21}} \frac{\partial z_{21}}{\partial w_{11}} + \frac{\partial J}{\partial z_{22}} \frac{\partial z_{22}}{\partial w_{11}} \\ &= \delta_{z11} \cdot a_{11} + \delta_{z12} \cdot a_{12} + \delta_{z21} \cdot a_{21} + \delta_{z22} \cdot a_{22}\end{aligned}\quad (9)$$

对W22也是一样的：

$$\begin{aligned}\frac{\partial J}{\partial w_{12}} &= \frac{\partial J}{\partial z_{11}} \frac{\partial z_{11}}{\partial w_{12}} + \frac{\partial J}{\partial z_{12}} \frac{\partial z_{12}}{\partial w_{12}} + \frac{\partial J}{\partial z_{21}} \frac{\partial z_{21}}{\partial w_{12}} + \frac{\partial J}{\partial z_{22}} \frac{\partial z_{22}}{\partial w_{12}} \\ &= \delta_{z11} \cdot a_{12} + \delta_{z12} \cdot a_{13} + \delta_{z21} \cdot a_{22} + \delta_{z22} \cdot a_{23}\end{aligned}\quad (10)$$

# 权重 (卷积核) 梯度计算

观察公式8和公式9，其实也是一个标准的卷积（互相关）操作过程，因此，可以把这个过程看成图17-27。

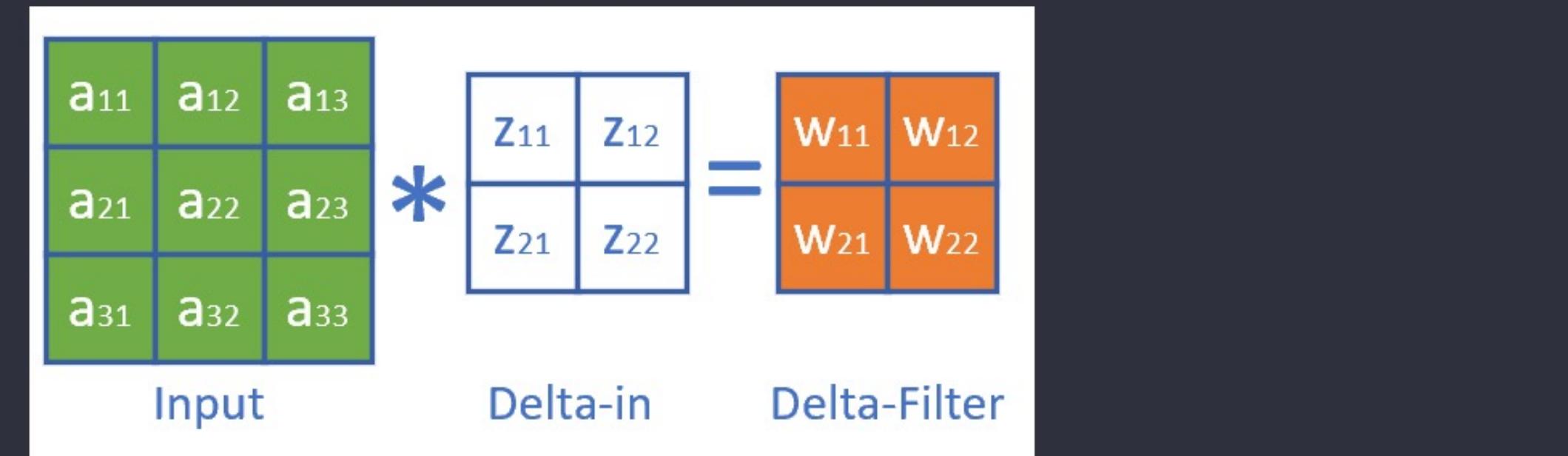


图17-27 卷积核的梯度计算

总结成一个公式：

$$\delta_w = A * \delta_{in} \quad (11)$$

# 偏移的梯度计算

根据前向计算公式1, 2, 3, 4, 可以得到:

$$\begin{aligned}\frac{\partial J}{\partial b} &= \frac{\partial J}{\partial z_{11}} \frac{\partial z_{11}}{\partial b} + \frac{\partial J}{\partial z_{12}} \frac{\partial z_{12}}{\partial b} + \frac{\partial J}{\partial z_{21}} \frac{\partial z_{21}}{\partial b} + \frac{\partial J}{\partial z_{22}} \frac{\partial z_{22}}{\partial b} \\ &= \delta_{z11} + \delta_{z12} + \delta_{z21} + \delta_{z22}\end{aligned}\tag{12}$$

所以:

$$\delta_b = \delta_{in}\tag{13}$$

每个卷积核W可能会有多个filter, 或者叫子核, 但是一个卷积核只有一个偏移, 无论有多少子核。

# 池化的前向计算与反向传播



# 池化层

## 常用池化方法

池化 pooling，又称为下采样，downstream sampling or sub-sampling。

池化方法分为两种，一种是最大值池化 Max Pooling，一种是平均值池化 Mean/Average Pooling。

1	3	0	3
2	2	4	1
4	2	0	0
3	3	3	1

Max Pooling

1	3	0	3
2	2	4	1
4	2	0	0
3	3	3	1

Average Pooling

- 最大值池化，是取当前池化视野中所有元素的最大值，输出到下一层特征图中。

- 平均值池化，是取当前池化视野中所有元素的平均值，输出到下一层特征图中。

# 池化层

目的是：

- 扩大视野：就如同先从近处看一张图片，然后离远一些再看同一张图片，有些细节就会被忽略
- 降维：在保留图片局部特征的前提下，使得图片更小，更易于计算
- 平移不变性，轻微扰动不会影响输出：比如上图中最大值池化的4，即使向右偏一个像素，其输出值仍为4
- 维持同尺寸图片，便于后端处理：假设输入的图片不是一样大小的，就需要用池化来转换成同尺寸图片

# 池化的其他方式

我们很少使用步长值与池化尺寸不同的配置，所以只是提一下，如图17-33。

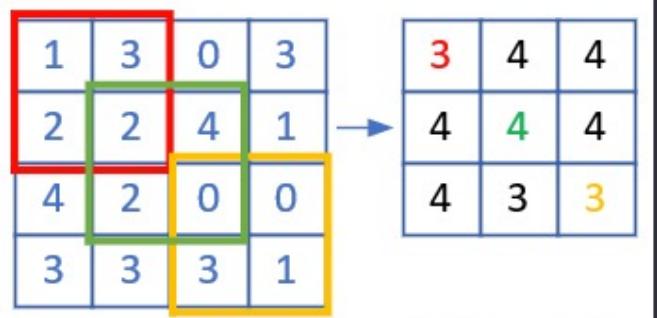


图17-33 步长为1的池化

上图是stride=1, size=2x2的情况，可以看到，右侧的结果中，有一大堆的3和4，基本分不开了，所以其池化效果并不好。

假设输入图片的形状是  $W_1 \times H_1 \times D_1$ ，其中W是图片宽度，H是图片高度，D是图片深度（多个图层），F是池化的视野（正方形），S是池化的步长，则输出图片的形状是：

$$\begin{cases} W_2 = (W_1 - F)/S + 1 \\ H_2 = (H_1 - F)/S + 1 \\ D_2 = D_1 \end{cases}$$

池化层不会改变图片的深度，即D值前后相同。

# 池化层的训练

我们假设图17-34中， $[[1, 2], [3, 4]]$ 是上一层网络回传的残差，那么：

- 对于最大值池化，残差值会回传到当初最大值的位置上，而其它三个位置的残差都是0。
- 对于平均值池化，残差值会平均到原始的4个位置上。

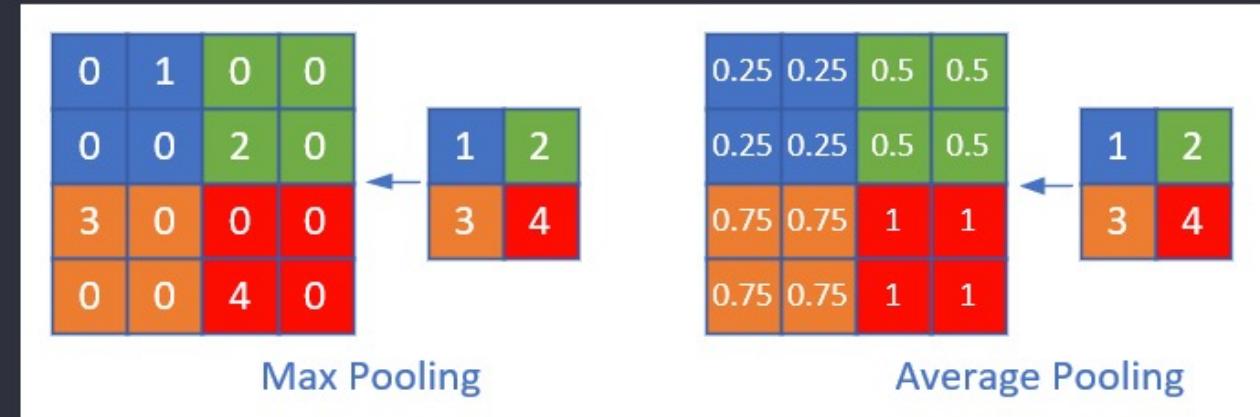


图17-34 平均池化与最大池化

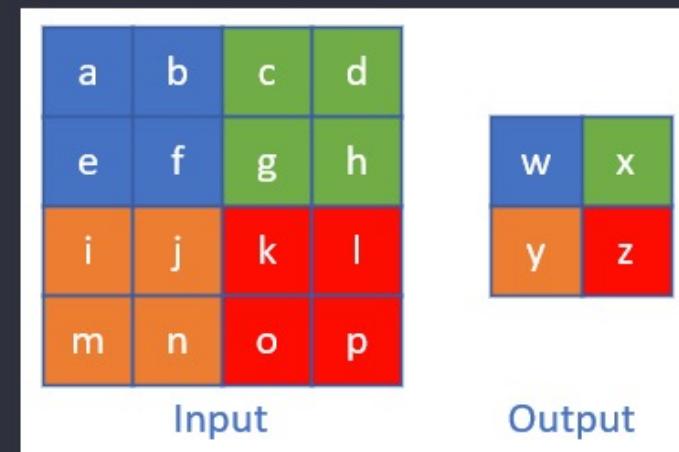


图17-35 池化层反向传播的示例

# 池化层的训练

## Max Pooling

严格的数学推导过程以图17-35为例进行。

正向公式：

$$w = \max(a, b, e, f)$$

反向公式（假设Input Layer中的最大值是b）：

$$\frac{\partial w}{\partial a} = 0, \quad \frac{\partial w}{\partial b} = 1$$

$$\frac{\partial w}{\partial e} = 0, \quad \frac{\partial w}{\partial f} = 0$$

因为a,e,f对w都没有贡献，所以偏导数为0，只有b有贡献，偏导数为1。

$$\delta_a = \frac{\partial J}{\partial a} = \frac{\partial J}{\partial w} \frac{\partial w}{\partial a} = 0$$

$$\delta_b = \frac{\partial J}{\partial b} = \frac{\partial J}{\partial w} \frac{\partial w}{\partial b} = \delta_w \cdot 1 = \delta_w$$

$$\delta_e = \frac{\partial J}{\partial e} = \frac{\partial J}{\partial w} \frac{\partial w}{\partial e} = 0$$

$$\delta_f = \frac{\partial J}{\partial f} = \frac{\partial J}{\partial w} \frac{\partial w}{\partial f} = 0$$

# 池化层的训练

## Mean Pooling

正向公式：

$$w = \frac{1}{4}(a + b + e + f)$$

反向公式（假设Layer-1中的最大值是b）：

$$\frac{\partial w}{\partial a} = \frac{1}{4}, \quad \frac{\partial w}{\partial b} = \frac{1}{4}$$

$$\frac{\partial w}{\partial e} = \frac{1}{4}, \quad \frac{\partial w}{\partial f} = \frac{1}{4}$$

因为a,b,e,f对w都有贡献，所以偏导数都为1：

$$\delta_a = \frac{\partial J}{\partial a} = \frac{\partial J}{\partial w} \frac{\partial w}{\partial a} = \frac{1}{4} \delta_w$$

$$\delta_b = \frac{\partial J}{\partial b} = \frac{\partial J}{\partial w} \frac{\partial w}{\partial b} = \frac{1}{4} \delta_w$$

$$\delta_e = \frac{\partial J}{\partial e} = \frac{\partial J}{\partial w} \frac{\partial w}{\partial e} = \frac{1}{4} \delta_w$$

$$\delta_f = \frac{\partial J}{\partial f} = \frac{\partial J}{\partial w} \frac{\partial w}{\partial f} = \frac{1}{4} \delta_w$$

无论是max pooling还是mean pooling，都没有要学习的参数，所以，在卷积网络的训练中，池化层需要做的只是把误差项向后传递，不需要计算任何梯度。

# 经典的神经网络模型



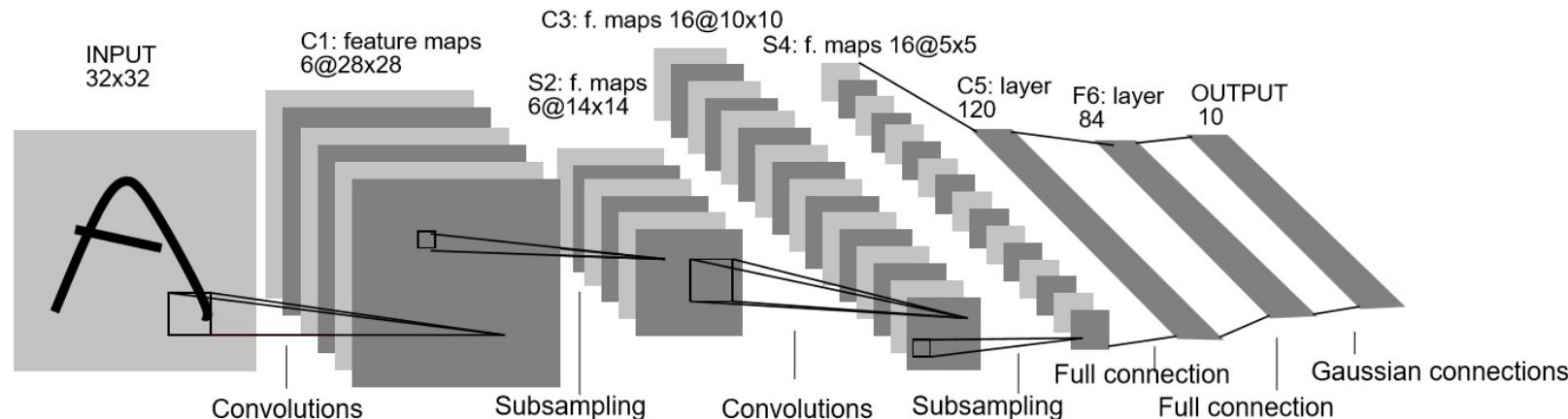
# 经典的卷积神经网络模型

卷积神经网络是现在深度学习领域中最有用的网络类型，尤其在计算机视觉领域更是一枝独秀。卷积神经网络从90年代的LeNet开始，沉寂了10年，也孵化了10年，直到2012年AlexNet开始再次崛起，后续的ZF Net、VGG、GoogLeNet、ResNet、DenseNet，网络越来越深，架构越来越复杂，解决反向传播时梯度消失的方法也越来越巧妙。下面让我们一起学习一下这些经典的网络模型。

# LeNet(1998)

LeNet是卷积神经网络的开创者LeCun在1998年提出，用于解决手写数字识别的视觉任务。自那时起，卷积神经网络的最基本的架构就定下来了：卷积层、池化层、全连接层。

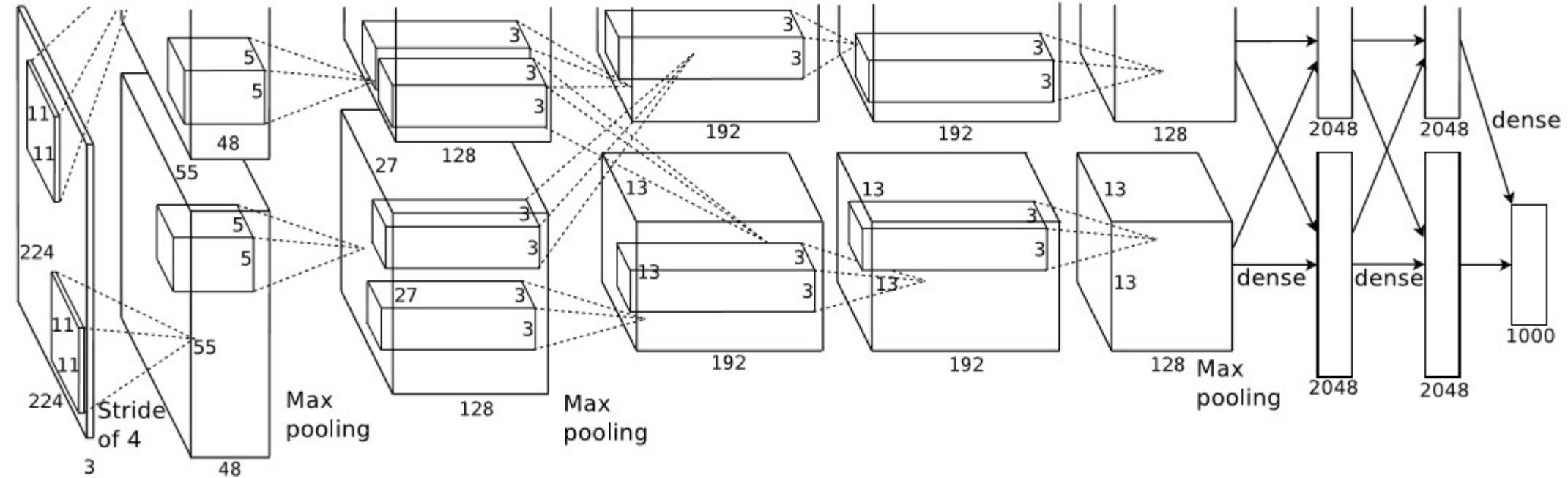
1. 输入为单通道32x32灰度图
2. 使用6组 $5 \times 5$ 的过滤器，每个过滤器里有一个卷积核， $\text{stride}=1$ ，得到6张 $28 \times 28$ 的特征图
3. 使用 $2 \times 2$ 的池化， $\text{stride}=2$ ，得到6张 $14 \times 14$ 的特征图
4. 使用16组 $5 \times 5$ 的过滤器，每个过滤器里有6个卷积核，对应上一层的6个特征图，得到16张 $10 \times 10$ 的特征图
5. 池化，得到16张 $5 \times 5$ 的特征图
6. 接全连接层，120个神经元
7. 接全连接层，84个神经元
8. 接全连接层，10个神经元，softmax输出



# AlexNet(1998)

AlexNet网络结构在整体上类似于LeNet，都是先卷积然后在全连接。但在细节上有很大不同。AlexNet更为复杂。AlexNet有60 million个参数和65000个神经元，五层卷积，三层全连接网络，最终的输出层是1000通道的Softmax。

AlexNet用两块GPU并行计算，大大提高了训练效率，并且在ILSVRC-2012竞赛中获得了top-5测试的15.3%的error rate，获得第二名的方法error rate是26.2%，差距非常大，足以说明这个网络在当时的影响力。



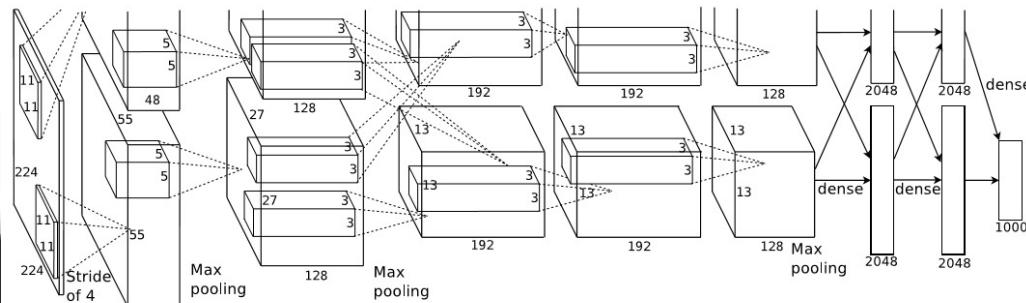
# AlexNet(1998)

1. 原始图片是 $3 \times 224 \times 224$ 三通道彩色图片，分开到上下两块GPU上进行训练
2. 卷积，96核， $11 \times 11$ ，stride=4，pad=0，输出 $96 \times 55 \times 55$
3. LRN+池化， $3 \times 3$ ，stride=2，pad=0，输出 $96 \times 27 \times 27$
4. 卷积，256核， $5 \times 5$ ，stride=1，pad=2，输出 $256 \times 27 \times 27$
5. LRN+池化， $3 \times 3$ ，stride=2，输出 $256 \times 13 \times 13$
6. 卷积，384核， $3 \times 3$ ，stride=1，pad=1，输出 $384 \times 13 \times 13$
7. 卷积，384核， $3 \times 3$ ，stride=1，pad=1，输出 $384 \times 13 \times 13$
8. 卷积，256核， $3 \times 3$ ，stride=1，pad=1，输出 $256 \times 13 \times 13$
9. 池化， $3 \times 3$ ，stride=2，输出 $256 \times 6 \times 6$
10. 全连接层，4096个神经元，接Dropout和Relu
11. 全连接层，4096个神经元，接Dropout和Relu
12. 全连接层，1000个神经元做分类

AlexNet的特点：

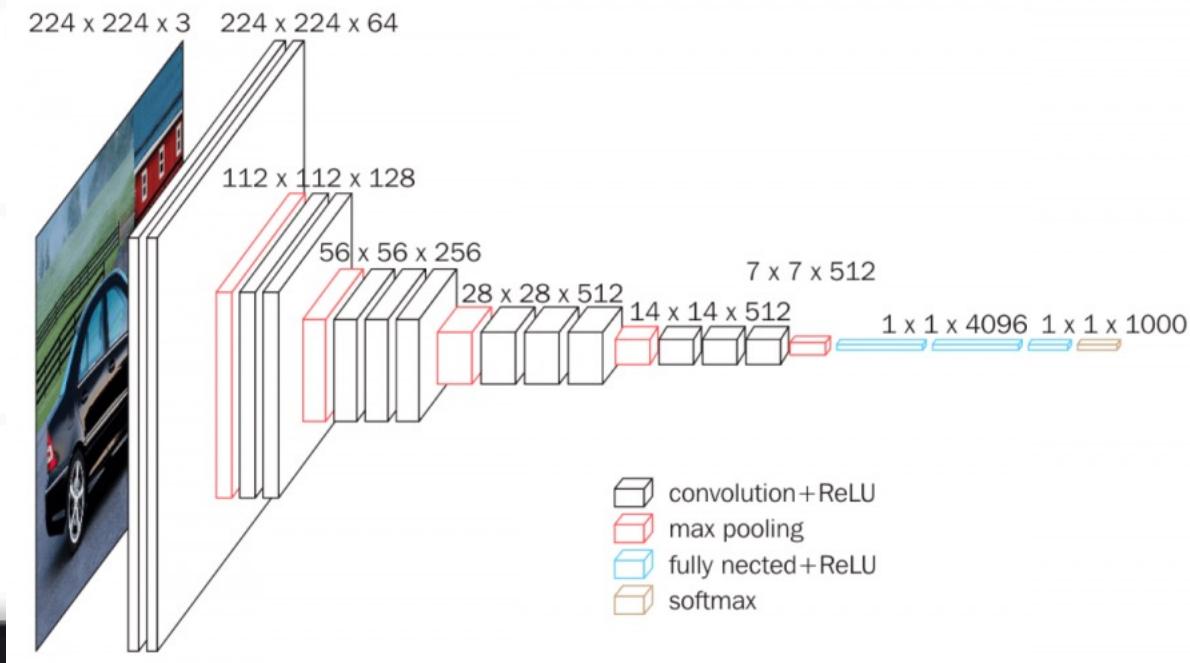
- 比LeNet深和宽的网络
- 使用了5层卷积和3层全连接，一共8层。特征数在最宽处达到384。
- 数据增强
- 针对原始图片 $256 \times 256$ 的数据，做了随机剪裁，得到 $224 \times 224$ 的图片若干张。
- 使用ReLU做激活函数
- 在全连接层使用DropOut
- 使用LRN

LRN的全称为Local Response Normalization，局部响应归一化，是想对线性输出做一个归一化，避免上下越界。发展至今，这个技术已经很少使用了。



# VGGNet(2015)

VGG Net由牛津大学的视觉几何组 (Visual Geometry Group) 和 Google DeepMind公司的研究员一起研发的深度卷积神经网络，在 ILSVRC 2014 上取得了第二名的成绩，将 Top-5错误率降到7.3%。它主要的贡献是展示出网络的深度 (depth) 是算法优良性能的关键部分。目前使用比较多的网络结构主要有ResNet (152-1000层) , GooleNet (22层) , VGGNet (19层) , 大多数模型都是基于这几个模型上改进，采用新的优化算法，多模型融合等。到目前为止，VGG Net 依然经常被用来提取图像特征。其论文名称为《VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION》, 为大规模图片识别而建立的非常深的卷积网络。短短3、4年时间，现在看起来这个也不能算是“非常深”了。



# VGGNet(2015)

VGG比较出名的是VGG-16和VGG-19，最常用的是VGG-16。

VGGNet的卷积层有一个特点：特征图的空间分辨率单调递减，特征图的通道数单调递增，使得输入图像在维度上流畅地转换到分类向量。用通俗的语言讲，就是特征图尺寸单调递减，特征图数量单调递增。从上面的模型图上来看，立体方块的宽和高逐渐减小，但是厚度逐渐增加。

AlexNet的通道数无此规律，VGGNet后续的GoogLeNet和Resnet均遵循此维度变化的规律。

一些其它的特点如下：

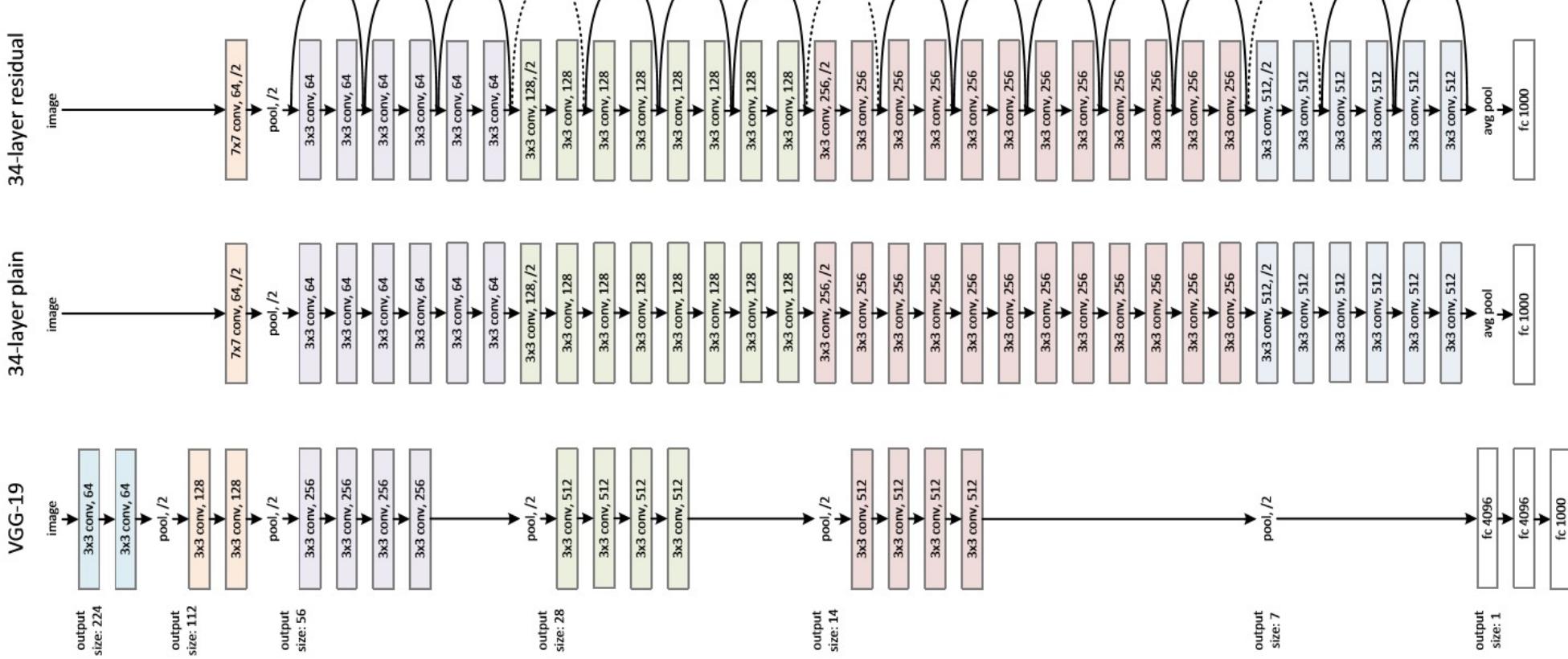
1. 选择采用 $3 \times 3$ 的卷积核是因为 $3 \times 3$ 是最小的能够捕捉像素8邻域信息的尺寸。
2. 使用 $1 \times 1$ 的卷积核目的是在不影响输入输出的维度情况下，对输入进行形变，再通过ReLU进行非线性处理，提高决策函数的非线性。
3. 2个 $3 \times 3$ 卷积堆叠等于1个 $5 \times 5$ 卷积，3个 $3 \times 3$ 堆叠等于1个 $7 \times 7$ 卷积，感受野大小不变，而采用更多层、更小的卷积核可以引入更多非线性（更多的隐藏层，从而带来更多非线性函数），提高决策函数判决力，并且带来更少参数。
4. 每个VGG网络都有3个FC层，5个池化层，1个softmax层。
5. 在FC层中间采用dropout层，防止过拟合。

虽然 VGGNet 减少了卷积层参数，但实际上其参数空间比 AlexNet 大，其中绝大多数的参数都是来自于第一个全连接层，耗费更多计算资源。在随后的 NIN 中发现将这些全连接层替换为全局平均池化，对于性能影响不大，同时显著降低了参数数量。

采用 Pre-trained 方法训练的 VGG model (主要是 D 和 E)，相对其他的方法参数空间很大，所以训练一个 VGG 模型通常要花费更长的时间，所幸有公开的 Pre-trained model 让我们很方便的使用。

# ResNets (2015)

2015年何恺明推出的ResNet在ISLVRC和COCO上横扫所有选手，获得冠军。ResNet在网络结构上做了大创新，而不再是简单的堆积层数，ResNet在卷积神经网络的新思路，绝对是深度学习发展历程上里程碑式的事情。

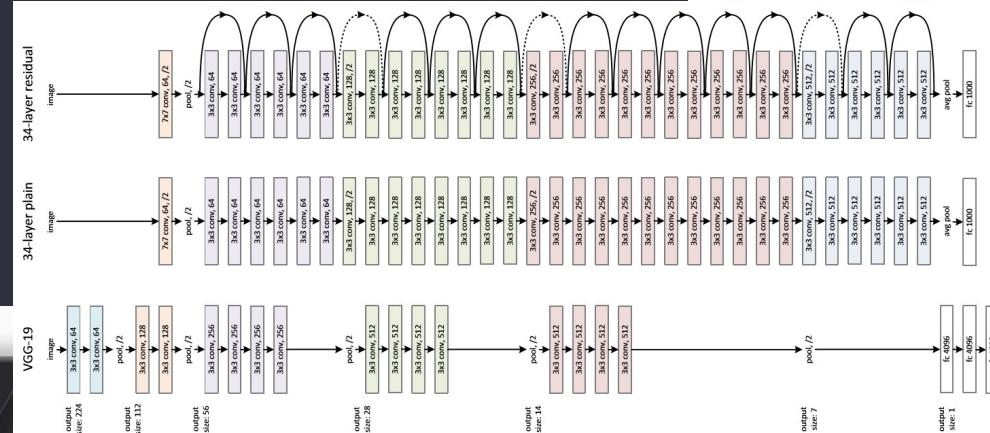
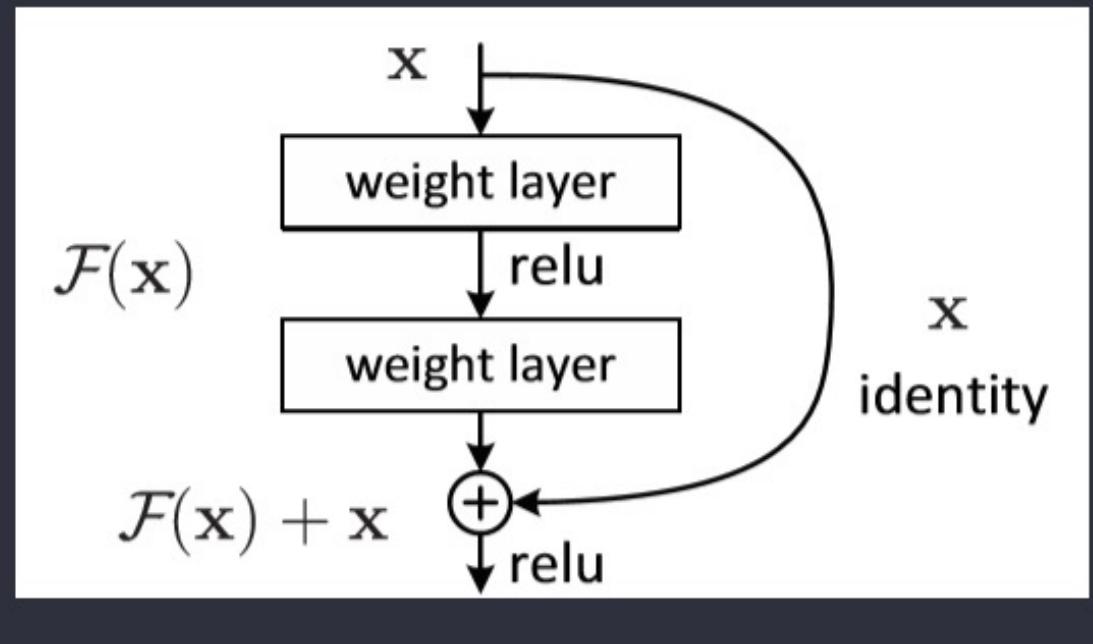


# ResNets (2015)

ResNet称为残差网络。什么是残差呢？

若将输入设为 $X$ , 将某一有参网络层设为 $H$ , 那么以 $X$ 为输入的此层的输出将为 $H(X)$ 。一般的卷积神经网络如Alexnet/VGG等会直接通过训练学习出参数函数 $H$ 的表达, 从而直接学习 $X \rightarrow H(X)$ 。而残差学习则是致力于使用多个有参网络层来学习输入、输出之间的参差即 $H(X) - X$ 即学习 $X \rightarrow (H(X) - X) + X$ 。其中 $X$ 这一部分为直接的identity mapping, 而 $H(X) - X$ 则为有参网络层要学习的输入输出间残差。

图18-11为残差学习这一思想的基本表示。



# ResNets (2015)

图18-12展示了两种形态的残差模块，左图是常规残差模块，有两个 $3 \times 3$ 卷积核组成，但是随着网络进一步加深，这种残差结构在实践中并不是十分有效。针对这问题，右图的“瓶颈残差模块”（bottleneck residual block）可以有更好的效果，它依次由 $1 \times 1$ 、 $3 \times 3$ 、 $1 \times 1$ 这三个卷积层堆积而成，这里的 $1 \times 1$ 的卷积能够起降维或升维的作用，从而令 $3 \times 3$ 的卷积可以在相对较低维度的输入上进行，以达到提高计算效率的目的。

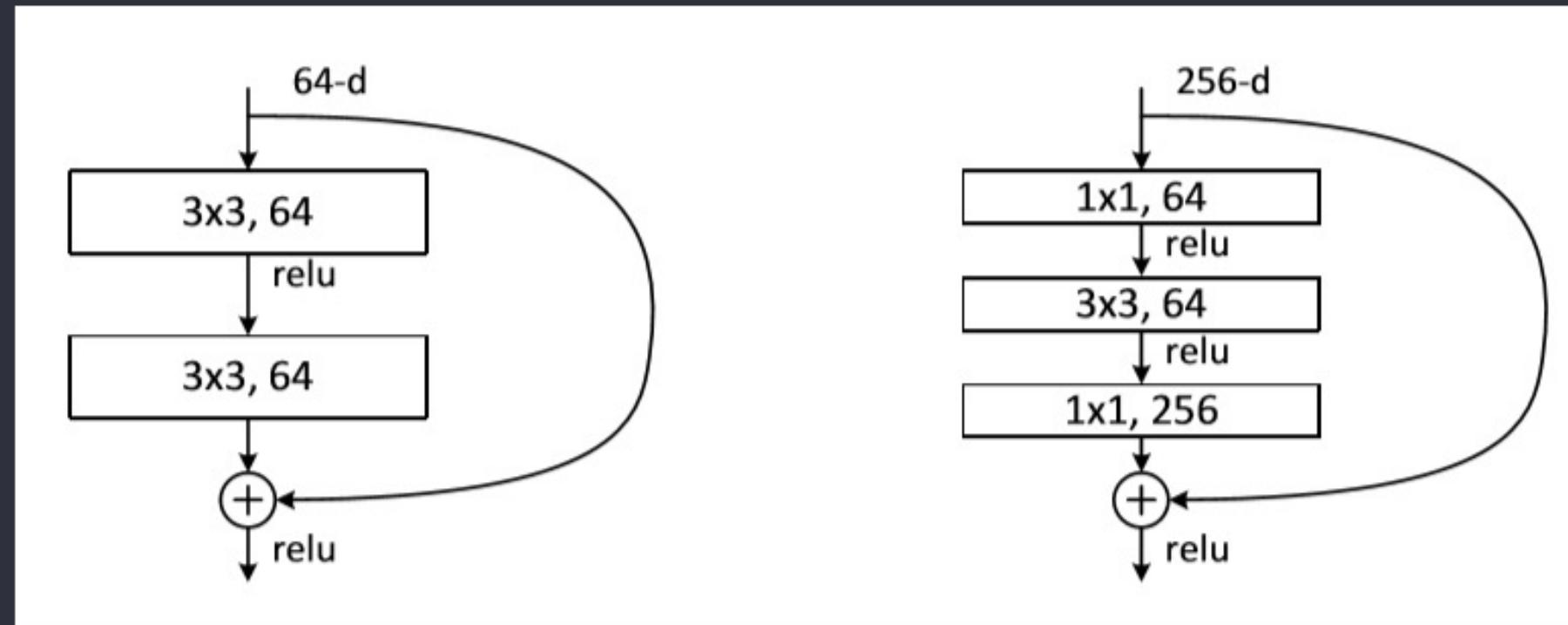
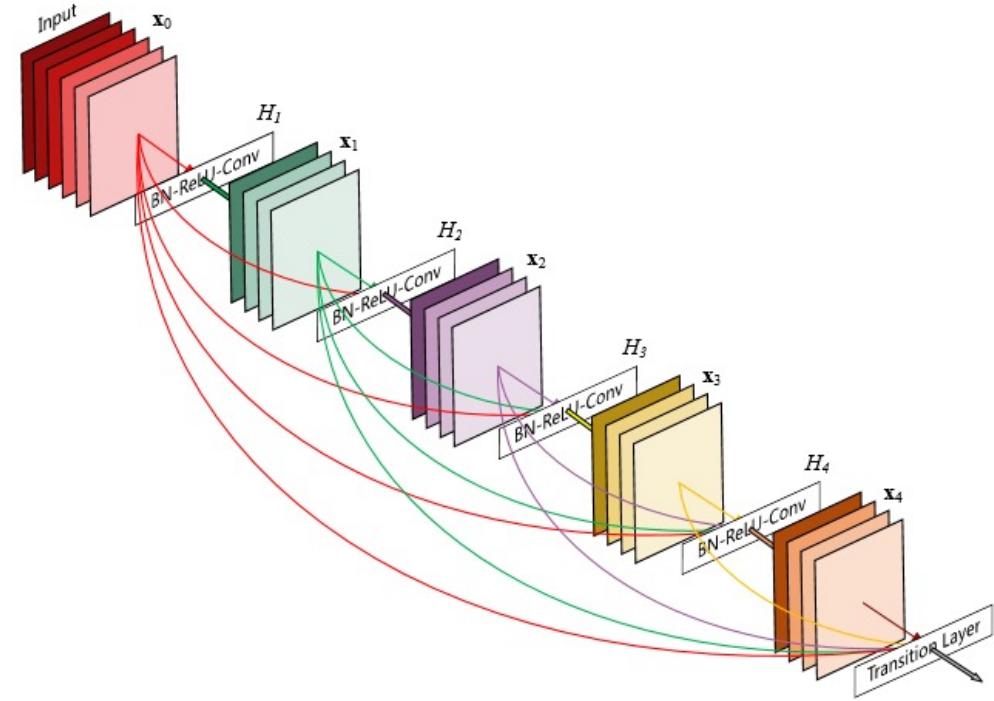


图18-12 两种形态的残差模块

# DenseNet (2017)

DenseNet 是一种具有密集连接的卷积神经网络。在该网络中，任何两层之间都有直接的连接，也就是说，网络每一层的输入都是前面所有层输出的并集，而该层所学习的特征图也会被直接传给其后面所有层作为输入。下图是 DenseNet 的一个dense block示意图，一个block里面的结构如下，与 ResNet中的BottleNeck基本一致：BN-ReLU-Conv( $1 \times 1$ )-BN-ReLU-Conv( $3 \times 3$ )，而一个 DenseNet则由多个这种block组成。每个 DenseBlock的之间层称为transition layers，由 BN- $\rightarrow$ Conv( $1 \times 1$ )- $\rightarrow$ averagePooling( $2 \times 2$ )组成。



**Figure 1:** A 5-layer dense block with a growth rate of  $k = 4$ . Each layer takes all preceding feature-maps as input.

# DenseNet (2017)

DenseNet作为另一种拥有较深层数的卷积神经网络，具有如下优点：

1. 相比ResNet拥有更少的参数数量
2. 旁路加强了特征的重用
3. 网络更易于训练，并具有一定的正则效果
4. 缓解了gradient vanishing和model degradation的问题

何恺明先生在提出ResNet时做出了这样的假设：若某一较深的网络多出另一较浅网络的若干层有能力学习到恒等映射，那么这一较深网络训练得到的模型性能一定不会弱于该浅层网络。通俗的说就是如果对某一网络中增添一些可以学到恒等映射的层组成新的网路，那么最差的结果也是新网络中的这些层在训练后成为恒等映射而不会影响原网络的性能。同样DenseNet在提出时也做过假设：与其多次学习冗余的特征，特征复用是一种更好的特征提取方式。

# 案例



# QnA





# Reactor

## Thank You!