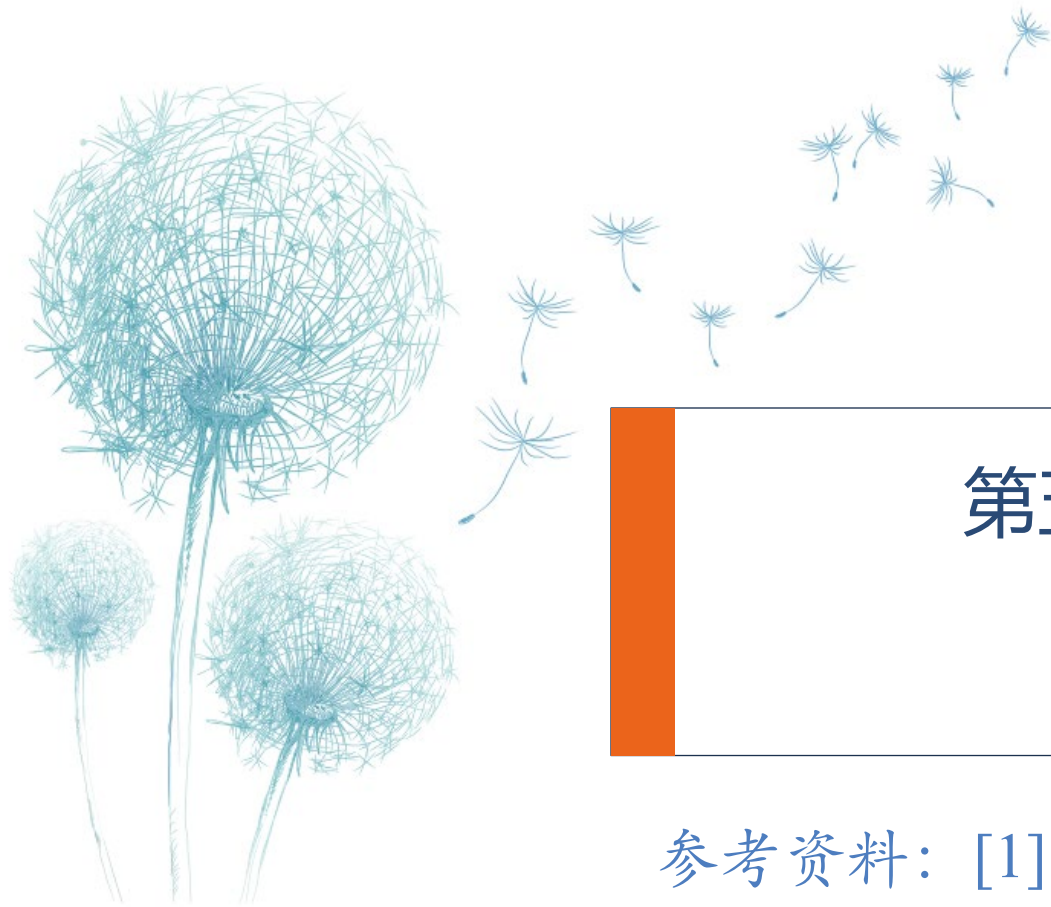


# 《计算机视觉》



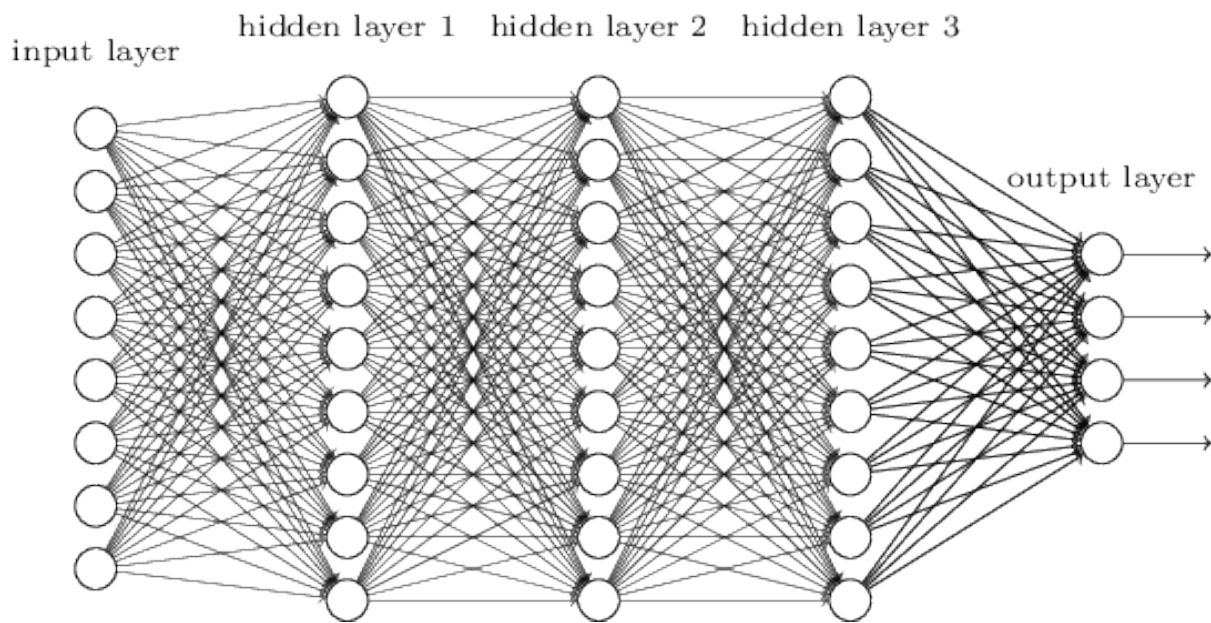
## 第五章 卷积神经网络

参考资料: [1] 邱锡鹏 《神经网络与深度学习》  
[2] 计算机视觉课程组资料

# 前馈神经网络

2

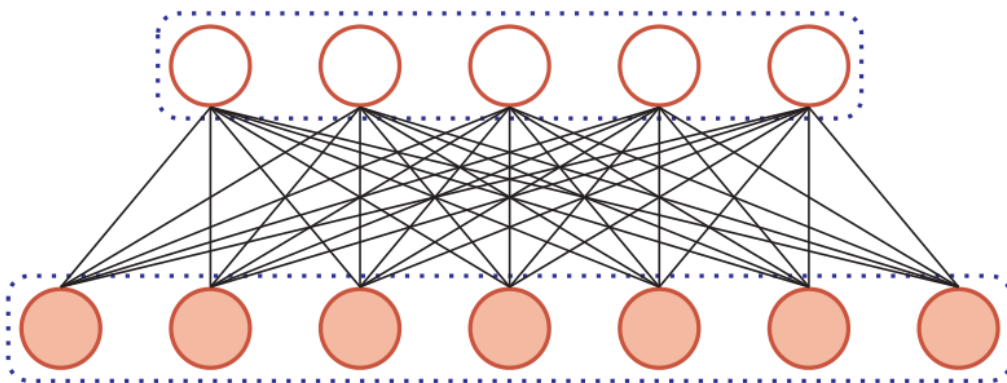
在没有CNN以及更先进的神经网络的时代，朴素的想法是用多层感知机（MLP）做图片分类的识别；但现实是，MLP做这事的效果并不理想。



# 全连接前馈神经网络

---

## ▶ 权重矩阵的参数非常多



## ▶ 局部不变性特征

- ▶ 自然图像中的物体都具有局部不变性特征
  - ▶ 尺度缩放、平移、旋转等操作不影响其语义信息。
- ▶ 全连接前馈网络很难提取这些局部不变特征

# 前馈神经网络

4

MLP的缺陷：

- 参数太大：神经元是全连接的方式构成的神经网络，全连接情况下，假设图片是 $1k \times 1k$ 像素大小，那么隐藏层个数和输入层尺寸一致时，不考虑RGB颜色通道，单通道下，权重 $w$ 参数个数会是： $(1k \times 1k) \times (1k \times 1k) = 10^{12}$
- 过深的全连接网络容易导致梯度消失，模型难以训练
- MLP无法做到图片的形变识别：例如每个人手写数字的习惯不一样，有的人写的歪了一点、或者大了一点，就会导致不同的识别结果。同一张图片进行旋转、平移之后，MLP就无法识别了

# 卷积神经网络

5

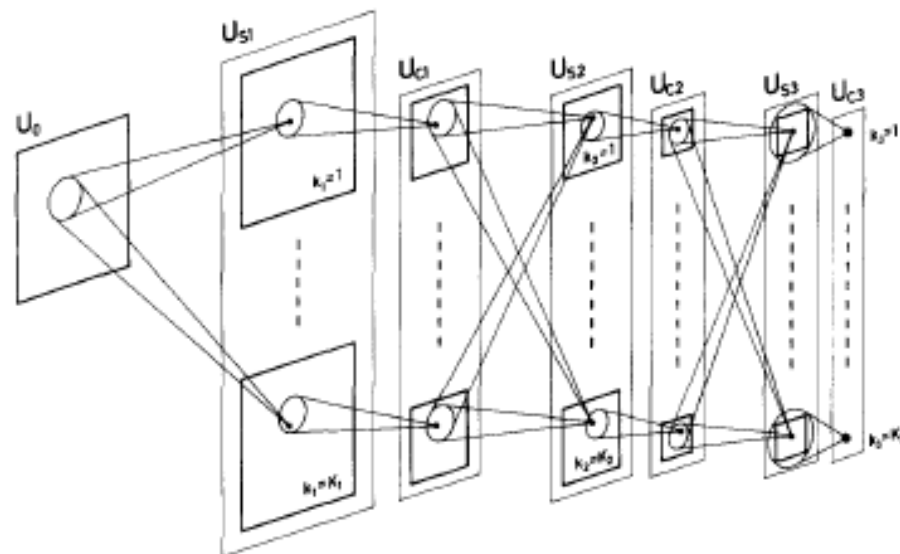
为了解决MLP的这些问题，卷积神经网络应运而生。



卷积神经网络发展历史中的第一件里程碑事件发生在上世纪60年代左右的神经科学（neuroscience）中，加拿大神经科学家David H. Hubel和Torsten Wiesel于1959年提出猫的初级视皮层中单个神经元的“感受野”（receptive field）概念，紧接着于1962年发现了猫的视觉中枢里存在感受野、双目视觉和其他功能结构，标志着神经网络结构首次在大脑视觉系统中被发现。

# 卷积神经网络

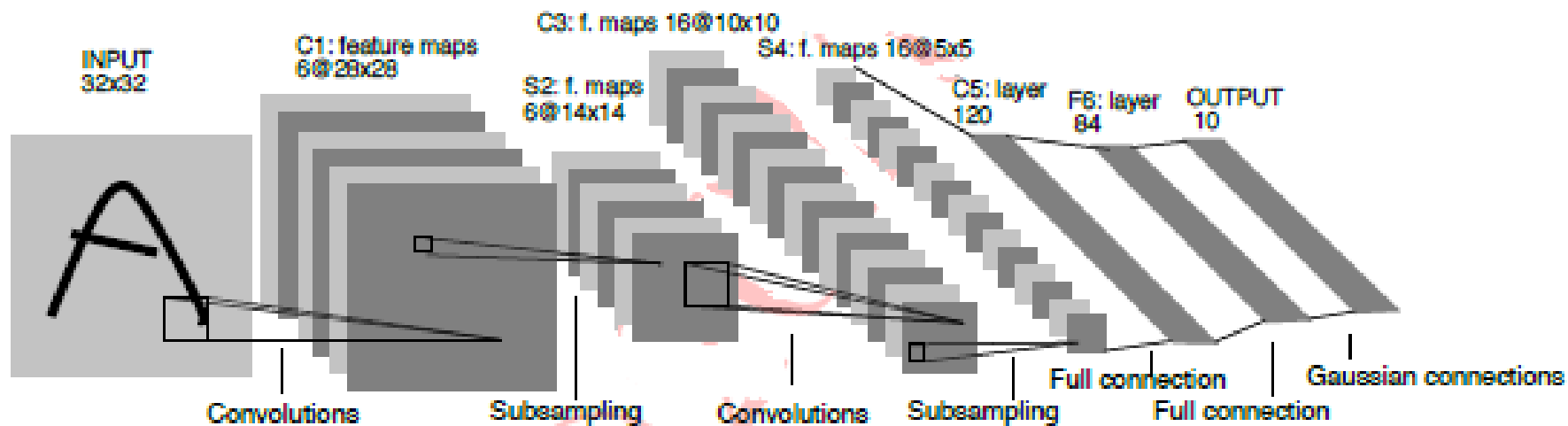
6



1980年前后，日本科学家福岛邦彦（Kunihiko Fukushima）在Hubel和Wiesel工作的基础上，模拟生物视觉系统并提出了一种层级化的多层人工神经网络，即“神经认知”（neurocognitron），以处理手写字符识别和其他模式识别任务。神经认知模型在后来也被认为是现今卷积神经网络的前身。在福岛邦彦的神经认知模型中，两种最重要的组成单元是“S型细胞”（S-cells）和“C型细胞”（C-cells），两类细胞交替堆叠在一起构成了神经认知网络。其中，S型细胞用于抽取局部特征（local features），C型细胞则用于抽象和容错，如图所示，不难发现这与现今卷积神经网络中的卷积层（convolution layer）和池化层（pooling layer）可一一对应。

# 卷积神经网络

7



在1998年，Yann LeCun等人提出基于梯度学习的卷积神经网络算法，并将其成功用于手写数字字符识别，在那时的技术条件下就能取得低于1%的错误率。因此，LeNet这一卷积神经网络便在当时效力于全美几乎所有的邮政系统，用来识别手写邮政编码进而分拣邮件和包裹。可以说，LeNet是第一个产生实际商业价值的卷积神经网络，同时也为卷积神经网络以后的发展奠定了坚实的基础。

# 卷积神经网络

---

## ▶ 卷积神经网络 (Convolutional Neural Networks, CNN)

- ▶ 一种前馈神经网络
- ▶ 受生物学上感受野 (Receptive Field) 的机制而提出的
  - ▶ 在视觉神经系统中，一个神经元的感受野是指视网膜上的特定区域，只有这个区域内的刺激才能够激活该神经元。
- ▶ 卷积神经网络有三个结构上的特性：
  - ▶ 局部连接
  - ▶ 权重共享
  - ▶ 空间或时间上的次采样



# 二维卷积

- 在图像处理中，图像是以二维矩阵的形式输入到神经网络中，因此我们需要二维卷积。

一个输入信息  $\mathbf{X}$  和滤波器  $\mathbf{W}$  的二维卷积定义为

$$\mathbf{Y} = \mathbf{W} * \mathbf{X},$$

$$y_{ij} = \sum_{u=1}^U \sum_{v=1}^V w_{uv} x_{i-u+1, j-v+1}.$$

1	1	1	1	1
-1	0	-3	0	1
2	1	1	-1	0
0	-1	1	2	1
1	2	1	1	1

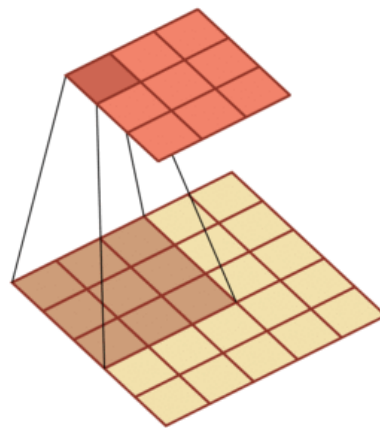
\*

1	0	0
0	0	0
0	0	-1

=

0	-2	-1
2	2	4
-1	0	0

Blue annotations on the first matrix: Row 1: (2,3) x-1, (3,4) x0, (4,5) x0; Row 2: (2,3) x0, (3,4) x0, (4,5) x0; Row 3: (2,3) x0, (3,4) x0, (4,5) x1.



# 卷积作为特征提取器

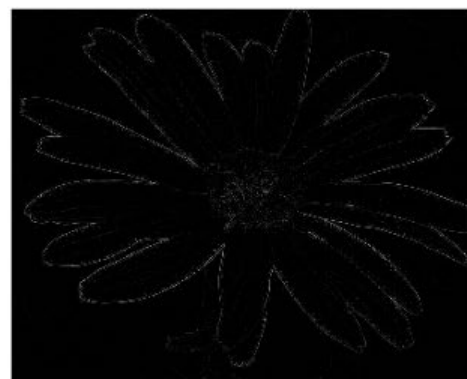


原始图像

$\otimes$

0	1	0
1	-4	1
0	1	0

=



0	1	1
-1	0	1
-1	-1	0

=



滤波器

输出特征映射

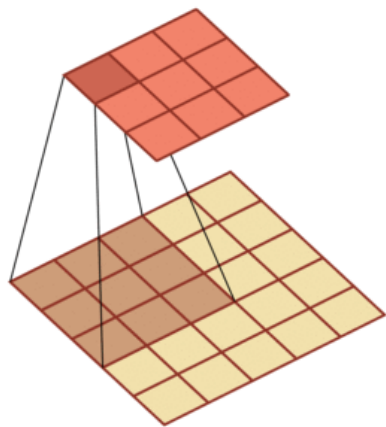
$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{16}$
$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{8}$
$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{16}$

=

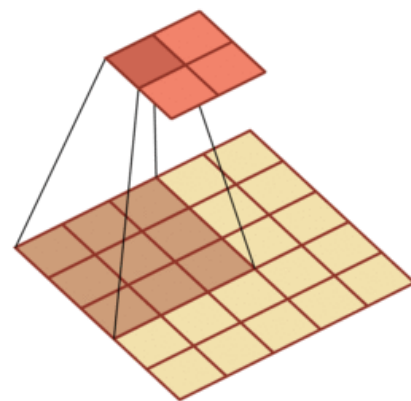


# 二维卷积

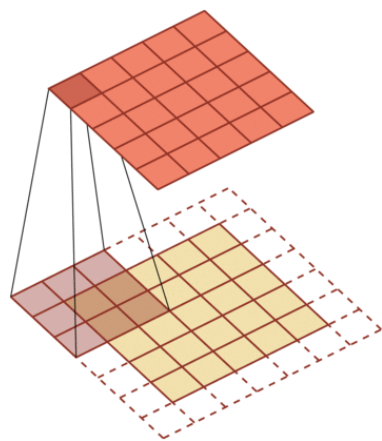
---



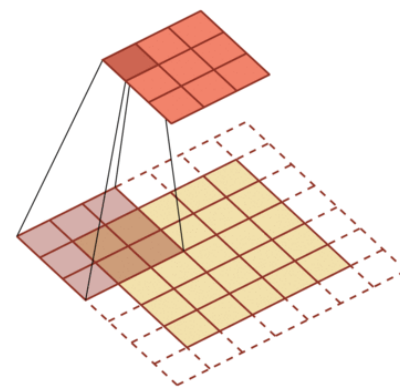
步长1，零填充0



步长2，零填充0



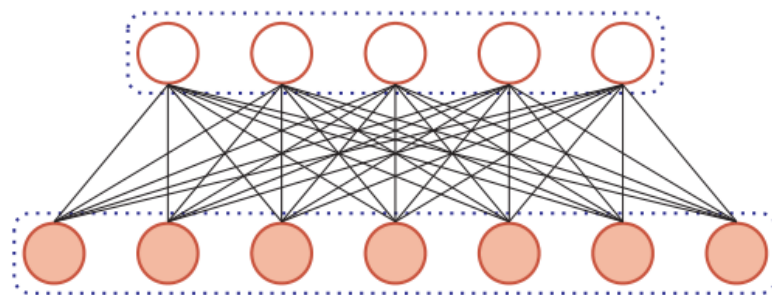
步长1，零填充1



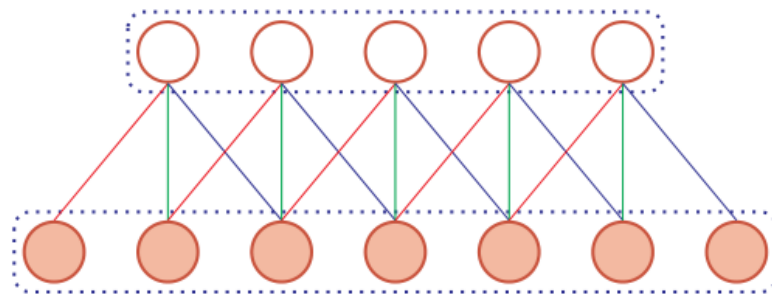
步长2，零填充1

# 卷积神经网络

## ► 用卷积层代替全连接层



(a) 全连接层



(b) 卷积层

# 多个卷积核

---

▶ 特征映射 (Feature Map) : 图像经过卷积后得到的特征。

▶ 卷积核看成一个特征提取器

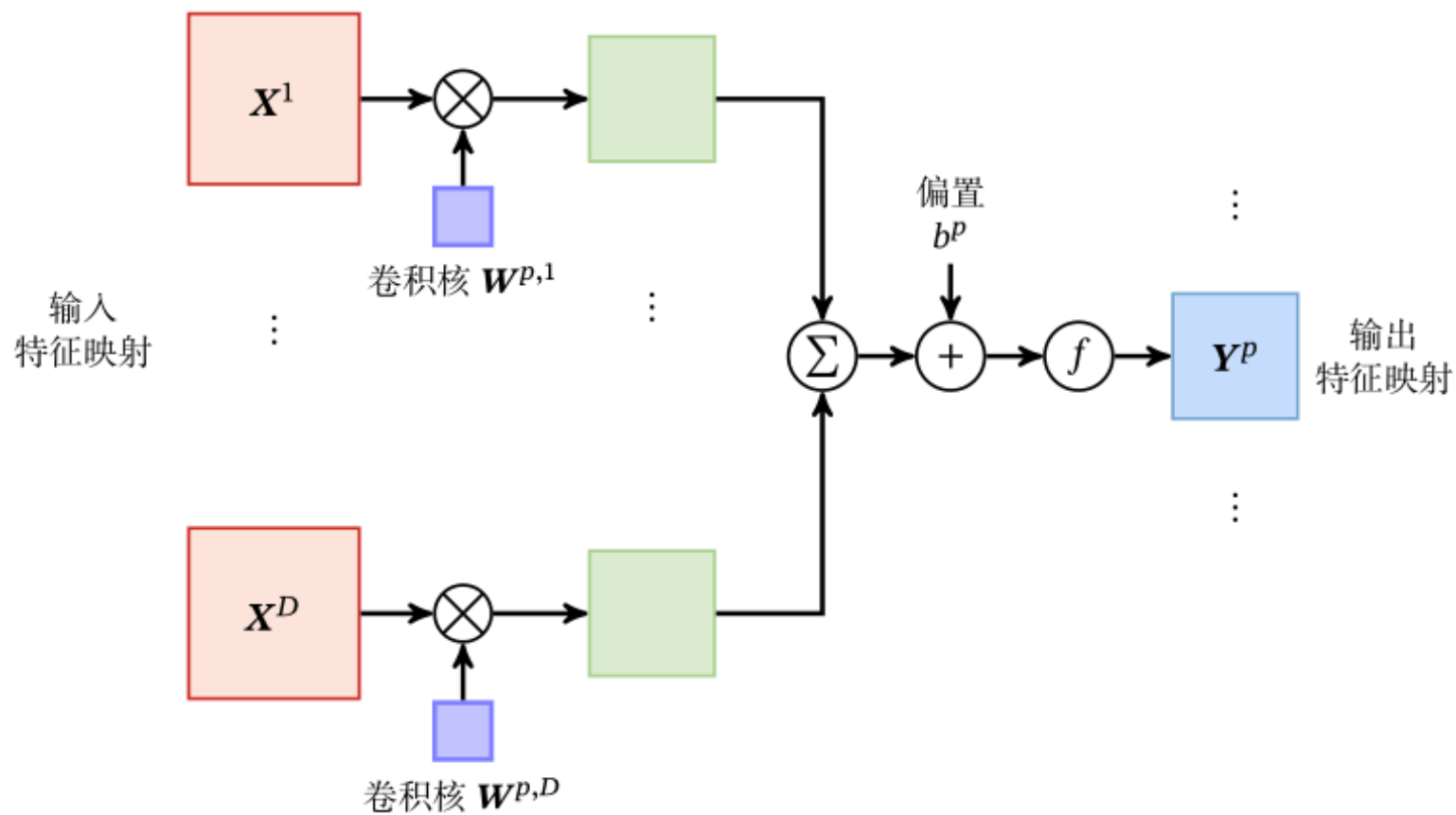
▶ 卷积层

▶ 输入:  $D$  个特征映射  $M \times N \times D$

▶ 输出:  $P$  个特征映射  $M' \times N' \times P$

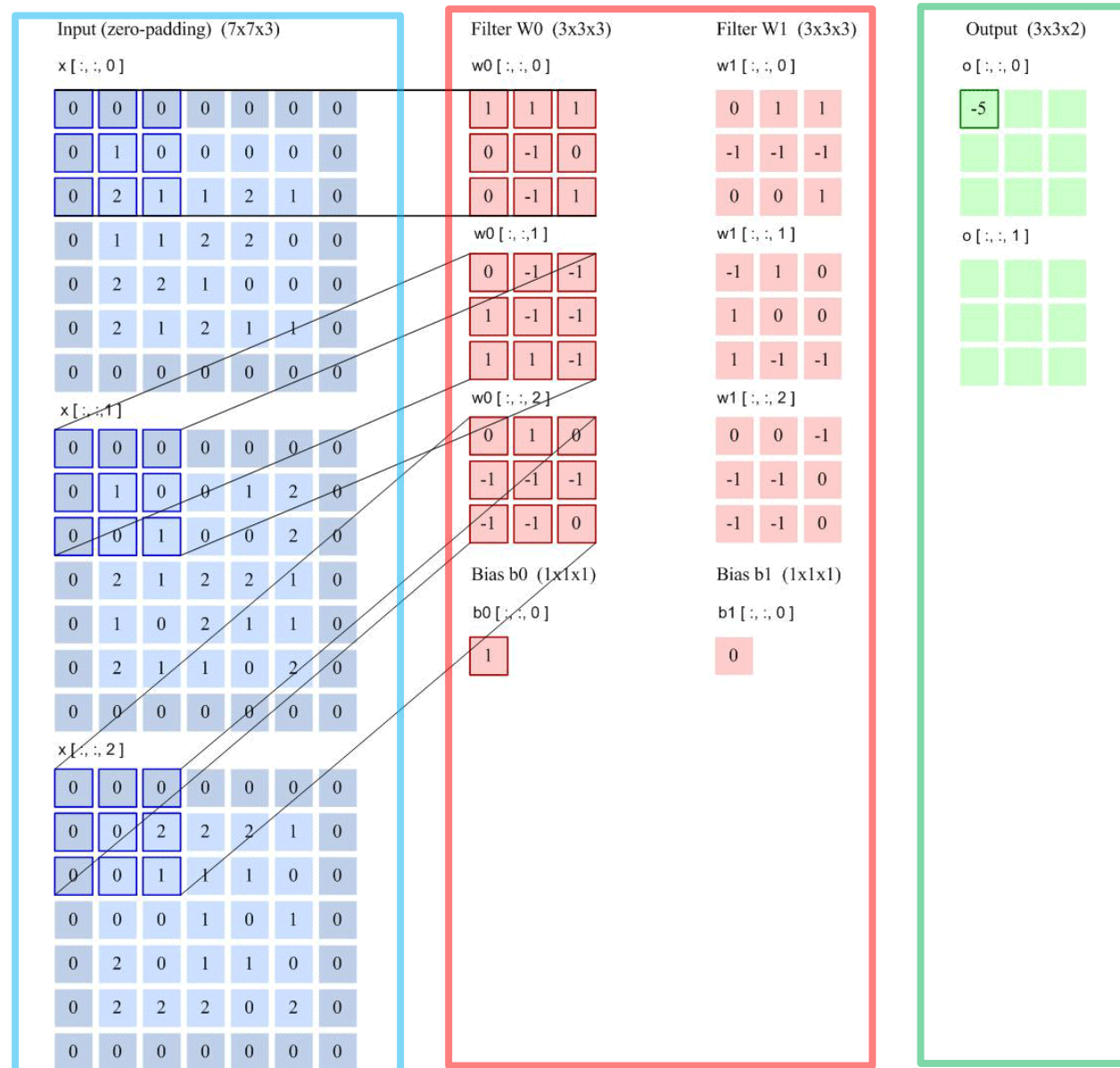
3维结构卷积层

# 卷积层的映射关系



$$Z^p = W^p \otimes X + b^p = \sum_{d=1}^D W^{p,d} \otimes X^d + b^p,$$

$$Y^p = f(Z^p).$$



步长2  
filter 3\*3  
filter个数6  
零填充 1



## 其它卷积种类



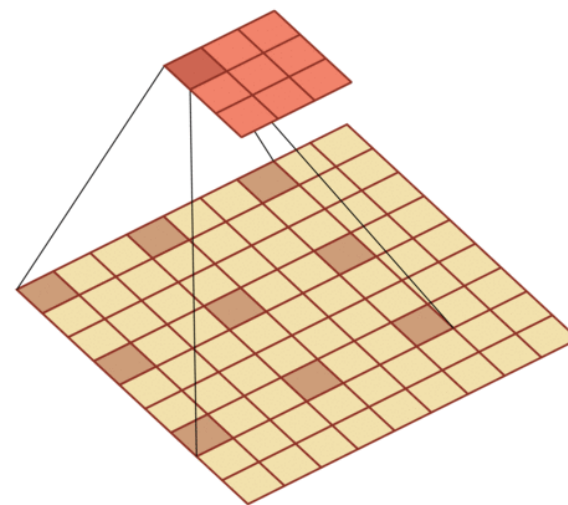
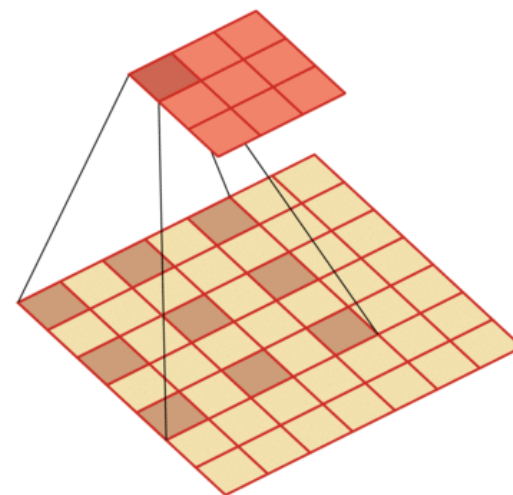
# 空洞卷积

## ► 如何增加输出单元的感受野

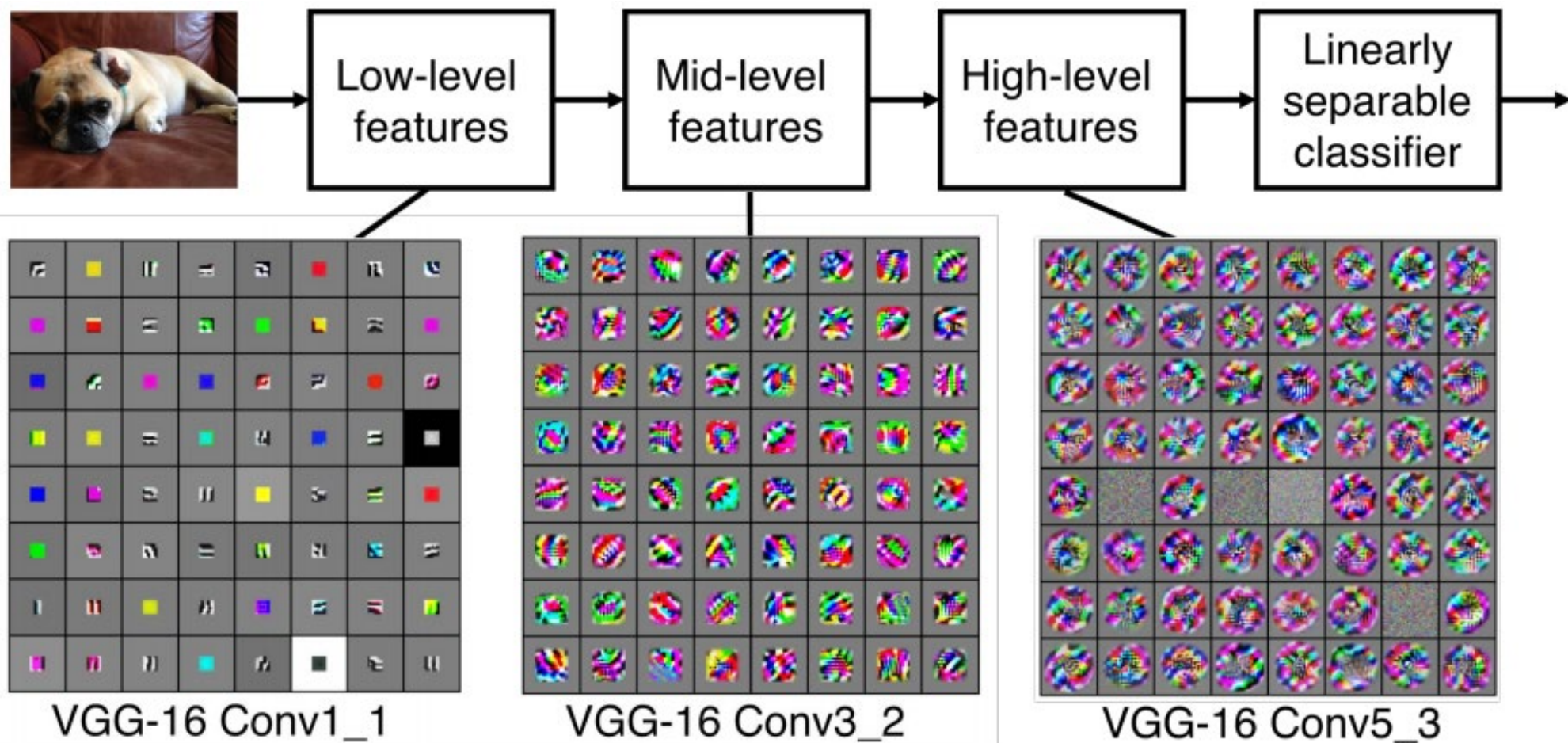
- 增加卷积核的大小
- 在卷积之前进行汇聚操作

## ► 空洞卷积

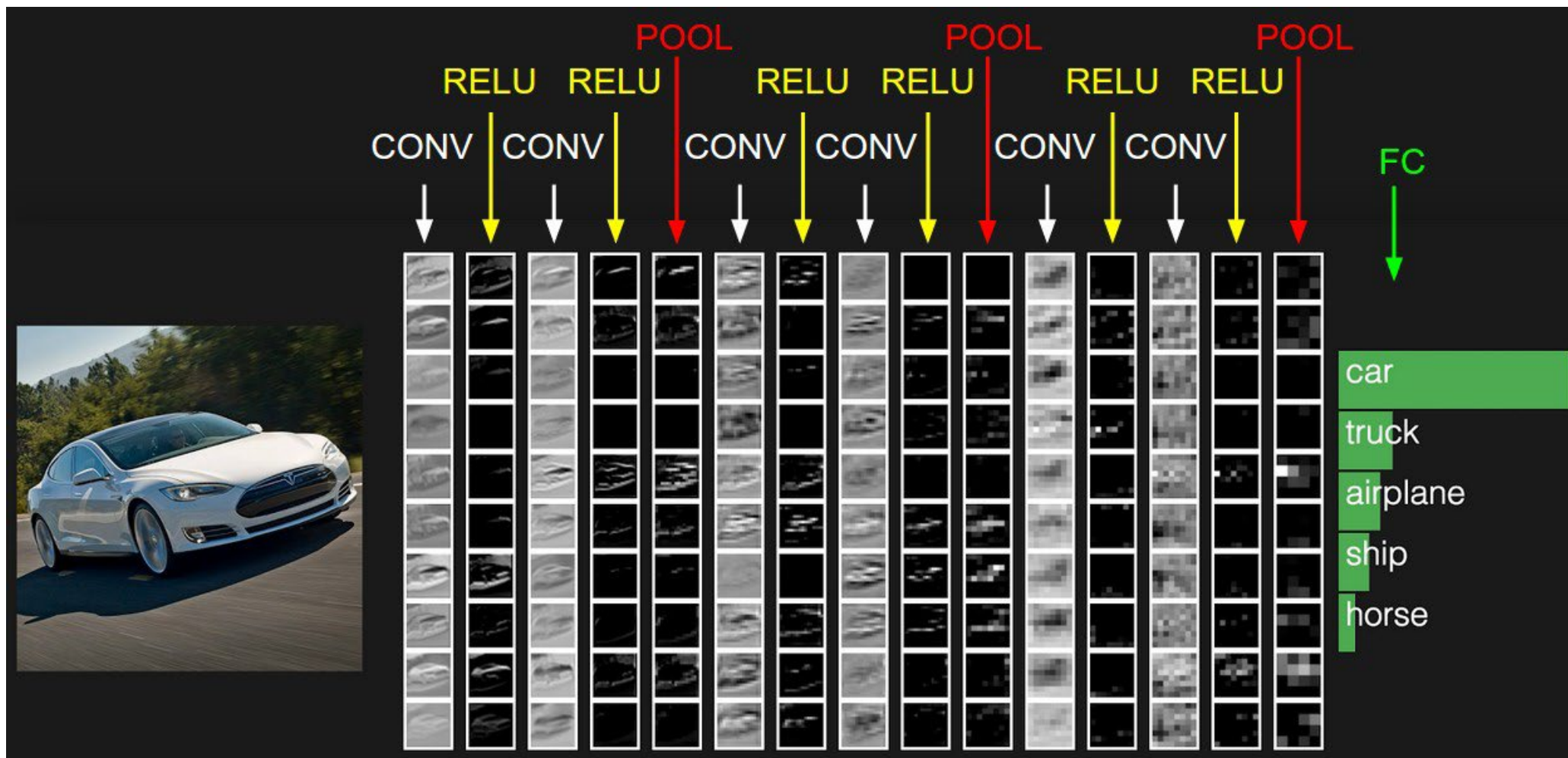
- 通过给卷积核插入“空洞”来变相地增加其大小。



# 表示学习



# 表示学习

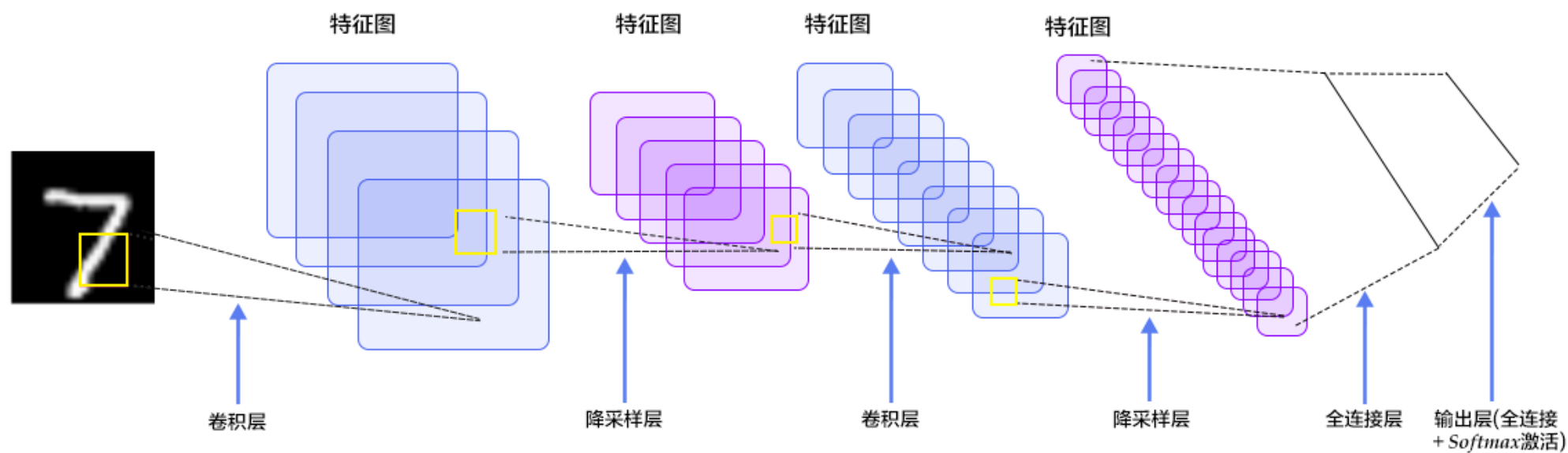




## 卷积网络的网络结构



# 卷积神经网络之网络结构

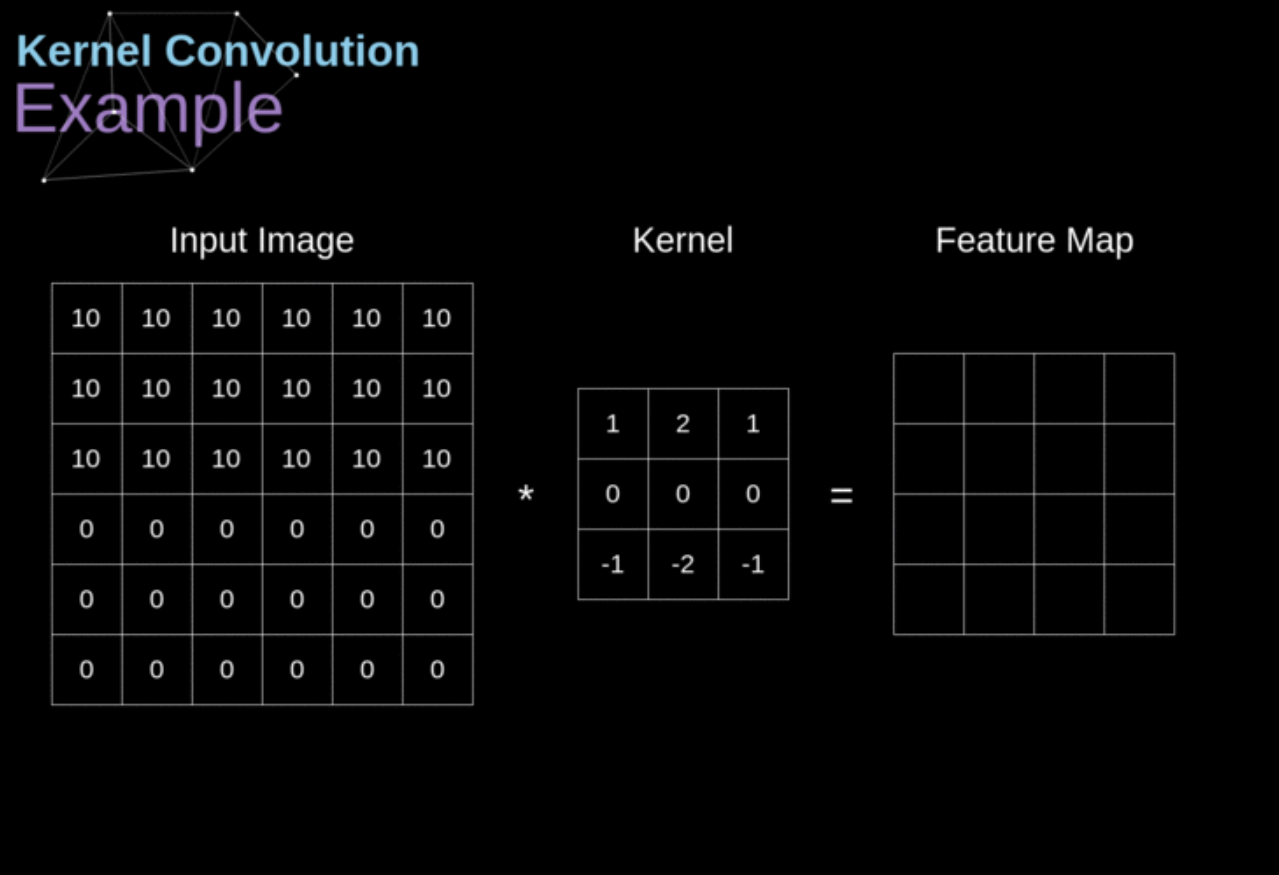


卷积神经网络是由很多不同的网络层组成，接下来将介绍卷积神经网络中最常见的三种网络层：卷积层、池化层（也可以叫做降采样层）以及全连接层。

# 卷积神经网络之网络结构

## (1) 卷积层

在网络结构中引入卷积层的目的是为了进行图像的特征提取。



卷积操作不是矩阵相乘，而是将卷积核放在输入矩阵上，对应的位置相乘然后再相加，之后就在输入矩阵上进行滑动，这样就得到了提取的特征图

由于卷积操作的特点，它由两个特性：

- **局部感知**
- **权值共享**（一个卷积核在滑动的过程中，卷积核的值是不变的。权值共享带来的好处是大大减少了参数量）

# 卷积神经网络之网络结构

## (1) 卷积层

卷积操作主要用于提取图像中的特征，那么，它到底是怎么提取的？

这里以提取图片中的垂直的边缘作为示例。

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

1	0	-1
1	0	-1
1	0	-1

卷积

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

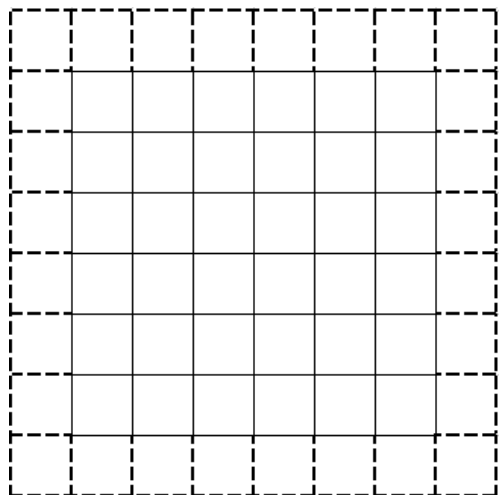


# 卷积神经网络之网络结构

## (1) 卷积层

- 卷积操作会使输入的图片变小，就像上一张中显示的那样 $6 \times 6$ 的输入矩阵经过 $3 \times 3$ 的卷积核之后变成了 $4 \times 4$ 的大小，如果在网络结构中多次使用卷积操作之后，图像可能就会变得非常小。
- 在卷积操作中有些像素的值只使用了一次，例如输入矩阵的左上角的像素，但是有些像素使用了很多次，比如在输入矩阵的中间的那些像素值

为了解决上面两个问题，我们可以在卷积操作之前，对输入的图像矩阵进行填充 (padding)



将输入图片 ( $6 \times 6$ ) 进行 $\text{pad}=1$ 的填充之后，就变成了 $8 \times 8$ 大小，这样在经过 $3 \times 3$ 的卷积操作后就会变成 $6 \times 6$ 大小，这样就能保证图片在卷积操作之后不会被缩小。

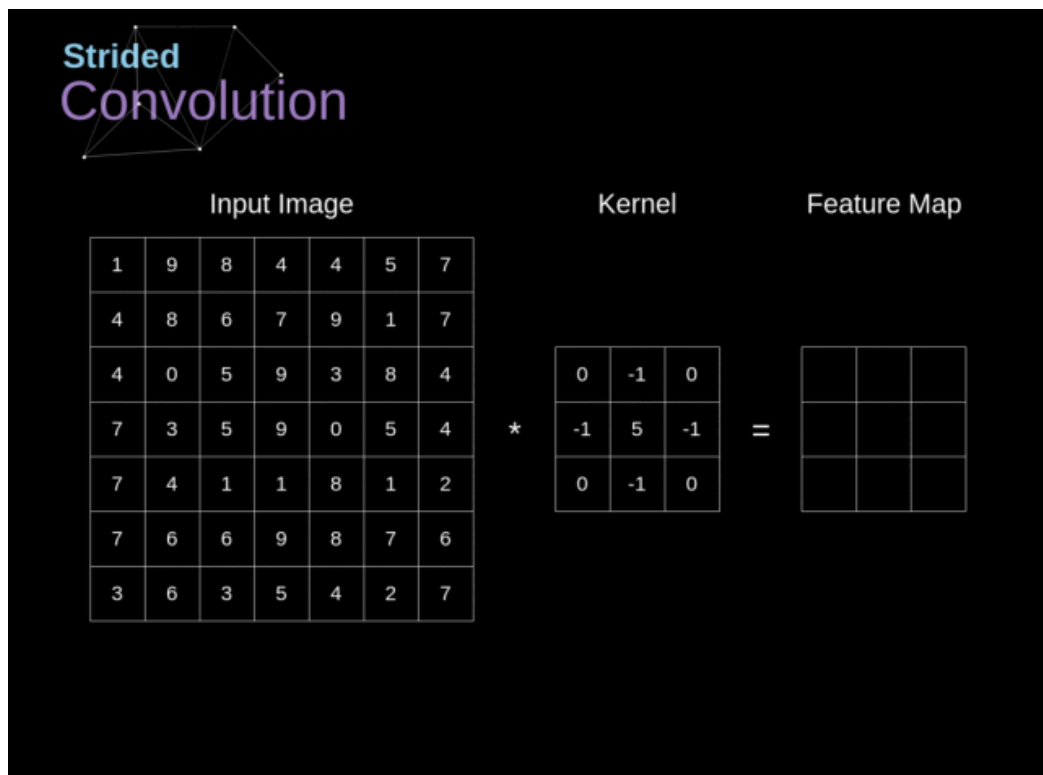
**通常**在填充的时候，我们会将填充的值设置为0，即在原图 $6 \times 6$ 的外面包围了一圈0。



# 卷积神经网络之网络结构

## (1) 卷积层

可以看到上面的例子中卷积核移动的步长都是1，实际上我们也可以设置步长为2或更多。步长的设置没有一个明确的标准，主要是看输入图像的大小和卷积核的大小。



- 步长小：提取的特征更全面，不会损失太多信息。但是会造成计算量大，甚至是过拟合的问题。
- 步长大：计算量会减少，但可能会丢失一些重要的特征。

# 卷积神经网络之网络结构

## (1) 卷积层

为了更好地理解卷积层是如何提取图像的特征，以及卷积时的步长和padding值如何影响卷积的效果，下面我们将会自己定义一个卷积层，并用它来提取特征。

```
class CNN(nn.Module):
    # 卷积层的初始化需要有卷积核、步长、padding这三个参数
    def __init__(self, kernel, stride, padding):
        super(CNN, self).__init__()
        self.stride = stride
        self.padding = padding
        self.weight = nn.Parameter(kernel)  # 自定义的权值

    # 前向传播，进行卷积后返回结果
    def forward(self, x):
        out = F.conv2d(x, self.weight, stride=self.stride, padding=self.padding)
        return out
```

# 卷积神经网络之网络结构

## (1) 卷积层

上面左图是定义的输入图像，它的可视化结果如右图所示，可以理解为在一个灰度图中白色与黑色的部分共同构成了一个垂直的边。

# 卷积神经网络之网络结构

## (1) 卷积层

```
# 定义卷积核参数
kernel = torch.FloatTensor(np.array([
    [1, 0, -1],
    [1, 0, -1],
    [1, 0, -1]
])).unsqueeze(0).unsqueeze(0)

# 将输入的灰度图进行处理, 并使用定义的卷积层进行卷积
a = torch.FloatTensor(a).unsqueeze(0).unsqueeze(0)
conv = CNN(kernel=kernel, stride=1, padding=0)
output = conv.forward(a)[0][0].detach().numpy()
```

给定一个卷积核的参数, 使其能够提取出垂直的边缘特征, 并且初始化一个卷积层, 这里的步长为1、padding为0。然后将输入的灰度图放入卷积层进行卷积。

# 卷积神经网络之网络结构

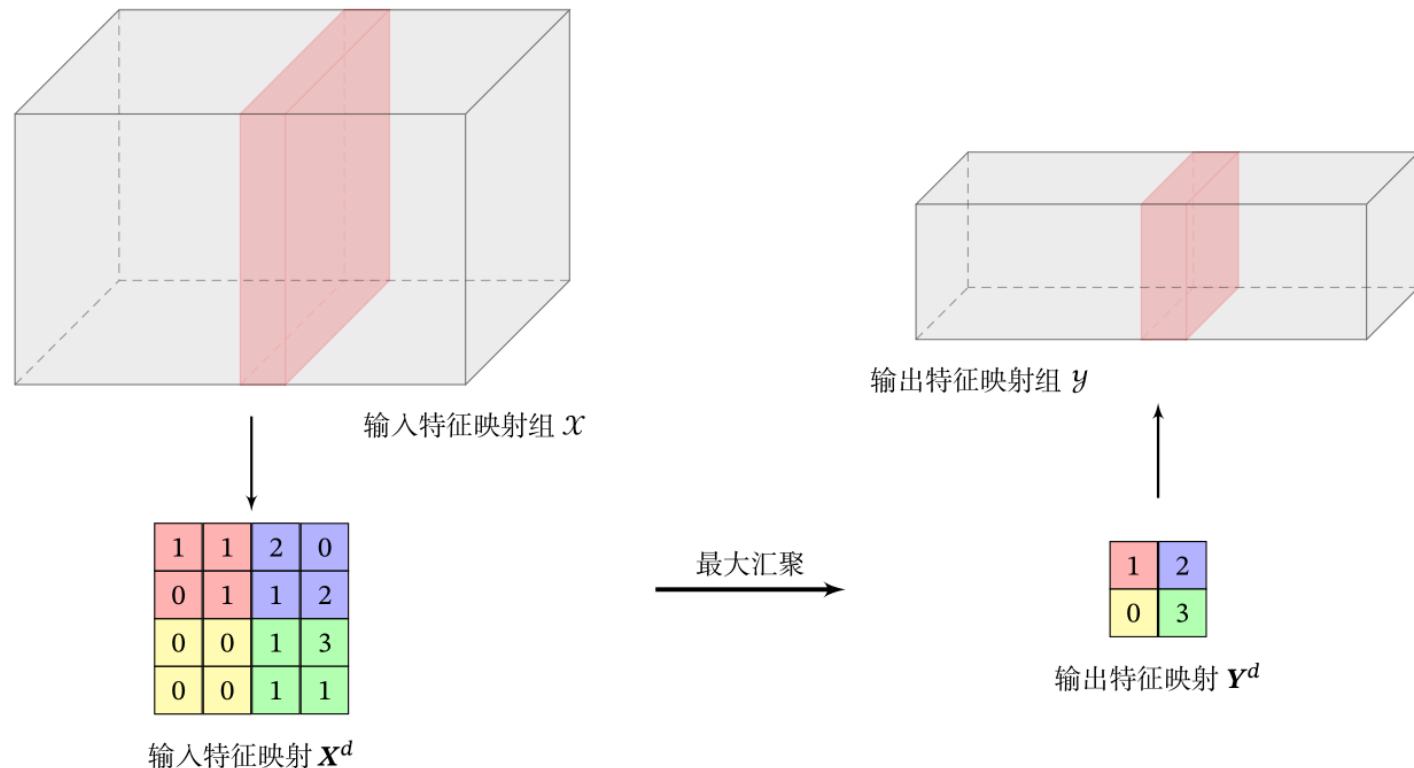
---

## (1) 卷积层

这是输入的灰度图经过卷积层处理后的可视化结果图，可以看到在输入中的边缘这在里已经被很好地检测出来了。

# 池化层

- 卷积层虽然可以显著减少连接的个数，但是每一个特征映射的神经元个数并没有显著减少。

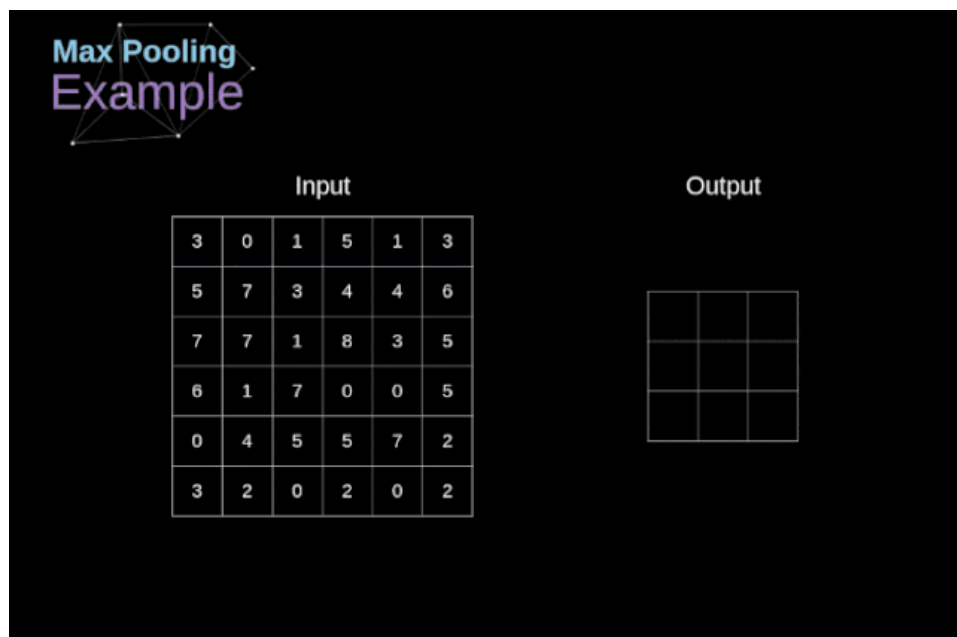


# 卷积神经网络之网络结构

## (2) 池化层

在网络结构中引入池化层，主要有以下几个原因：

- 减少运算量
- 实现不变性：平移不变性、旋转不变性、尺度不变性，提高模型的泛化能力

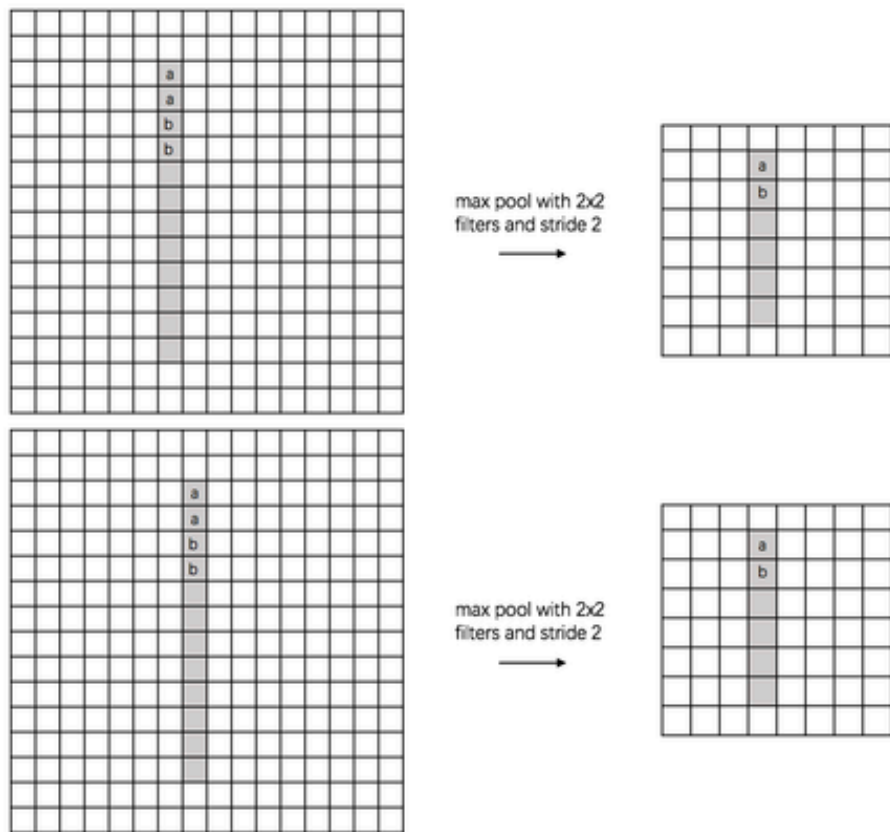


如图，将一个 $2 \times 2$ 的池化核放在图上进行最大池化（选取池化核中最大的元素），池化的步长是2。

# 卷积神经网络之网络结构

## (2) 池化层

池化层的平移不变性



如左图中，上下都是一个 $16 \times 16$ 的图片，里面有个数字1，区别是上面的1偏左一点，下面的偏右一点，但是在经过大小为 $2 \times 2$ ，步长为2的最大池化之后，得到的 $8 \times 8$ 特征矩阵是相同的。

这里将主要的特征捕获到了，同时还将问题的规模从 $16 \times 16$ 降到了 $8 \times 8$ ，而且还具有平移不变性的特点。



# 卷积神经网络之网络结构

## (2) 池化层

池化层的平移不变性

```
# a的灰度图
a = [
    [0, 0, 10, 0, 0, 0],
    [0, 0, 10, 0, 0, 0],
    [0, 0, 10, 0, 0, 0],
    [0, 0, 10, 0, 0, 0],
    [0, 0, 10, 0, 0, 0],
    [0, 0, 10, 0, 0, 0],
    [0, 0, 10, 0, 0, 0],
    [0, 0, 10, 0, 0, 0],
]

# b的灰度图
b = [
    [0, 0, 0, 10, 0, 0],
    [0, 0, 0, 10, 0, 0],
    [0, 0, 0, 10, 0, 0],
    [0, 0, 0, 10, 0, 0],
    [0, 0, 0, 10, 0, 0],
    [0, 0, 0, 10, 0, 0],
    [0, 0, 0, 10, 0, 0],
    [0, 0, 0, 10, 0, 0],
]
```

定义输入的图像a和b，这里的a和b的大小为8\*6，可以看到b是a的图像向右平移一格得到的图像

# 卷积神经网络之网络结构

## (2) 池化层

池化层的平移不变性

```
# 定义最大池化层
class MaxPooling(nn.Module):
    # 初始化需要给定池化核的大小，默认步长等于池化核的大小，默认的padding为0
    def __init__(self, kernelSize, stride=None, padding=0):
        super(MaxPooling, self).__init__()
        self.kernelSize = kernelSize
        self.stride = stride
        self.padding = padding

    def forward(self, x):
        return nn.MaxPool2d(kernel_size=self.kernelSize, stride=self.stride, padding=self.padding)(x)
```

定义一个最大池化层，之后将使用它来对之前的a、b两幅灰度图进行最大池化的处理

。

# 卷积神经网络之网络结构

## (2) 池化层

池化层的平移不变性

```
# 创建一个最大池化层，并将输入进行处理
maxPooling = MaxPooling(kernelSize=2)
a = torch.FloatTensor(a).unsqueeze(0)
b = torch.FloatTensor(b).unsqueeze(0)
outA = maxPooling(a)[0]
outB = maxPooling(b)[0]
print('outA shape:{}'.format(outA.shape))
print('outB shape:{}'.format(outB.shape))
```

```
outA shape:torch.Size([4, 3])
outB shape:torch.Size([4, 3])
```

创建一个最大池化层，这里定义的池化核的大小为2，然后将输入的a、b处理后送入池化层，得到outA和outB，并且outA和outB的大小是4\*3

# 卷积神经网络之网络结构

---

## (2) 池化层

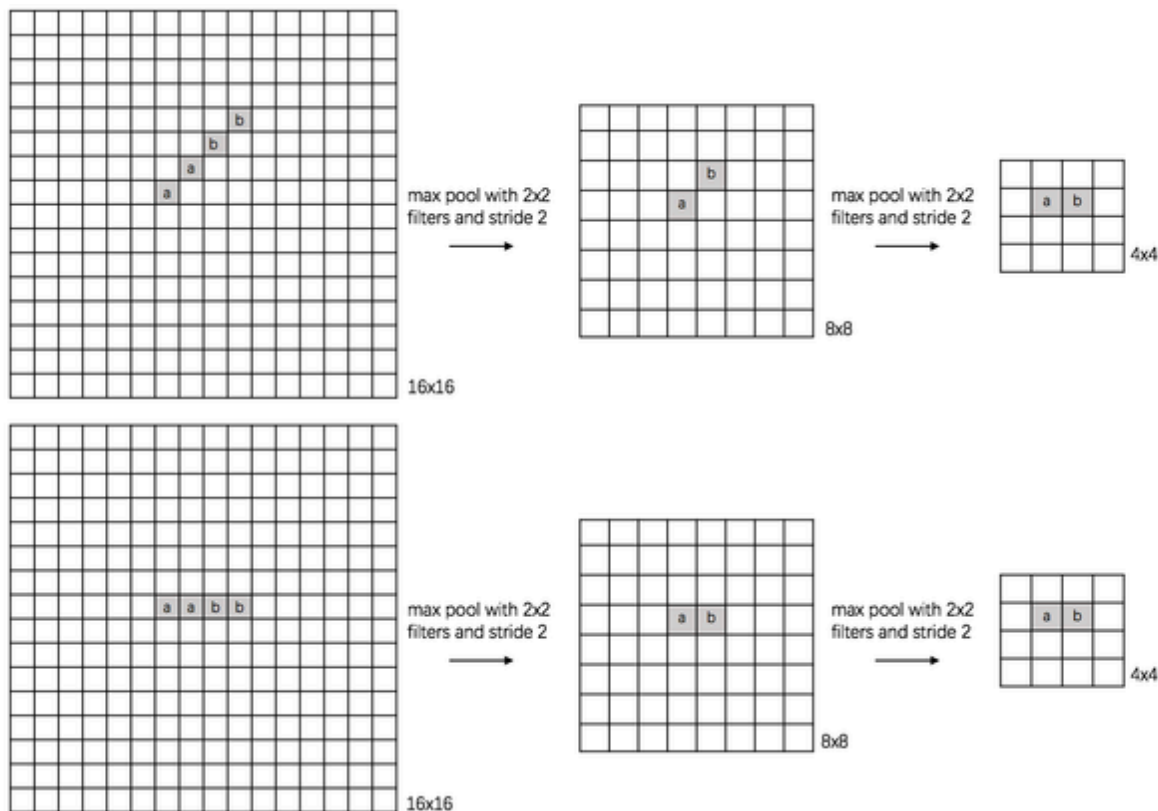
池化层的平移不变性

这是outA和outB的可视化结果图，可以看到a和b在经过最大池化后，它们的结果是一样的。

# 卷积神经网络之网络结构

## (2) 池化层

池化层的旋转不变性



左图中表示汉字“一”的识别，上面的图相对于下面的图具有一定的倾斜角，相当于是进行了旋转，但是经过两次大小为 $2 \times 2$ ，步长为2的池化操作后，得到的特征矩阵是相同的。

# 卷积神经网络之网络结构

## (2) 池化层

池化层的旋转不变性

```
# a的灰度图
a = [
    [0, 0, 10, 0],
    [0, 10, 0, 0],
    [0, 0, 0, 0],
    [0, 0, 0, 0],
]
# b的灰度图
b = [
    [0, 0, 0, 0],
    [0, 10, 10, 0],
    [0, 0, 0, 0],
    [0, 0, 0, 0],
]
```

定义输入的图像a和b，这里的a和b的大小为4\*4，可以看到b是a的图像顺时针旋转45度得到的

# 卷积神经网络之网络结构

---

## (2) 池化层

池化层的旋转不变性

```
outA shape:torch.Size([2, 2])  
outB shape:torch.Size([2, 2])
```

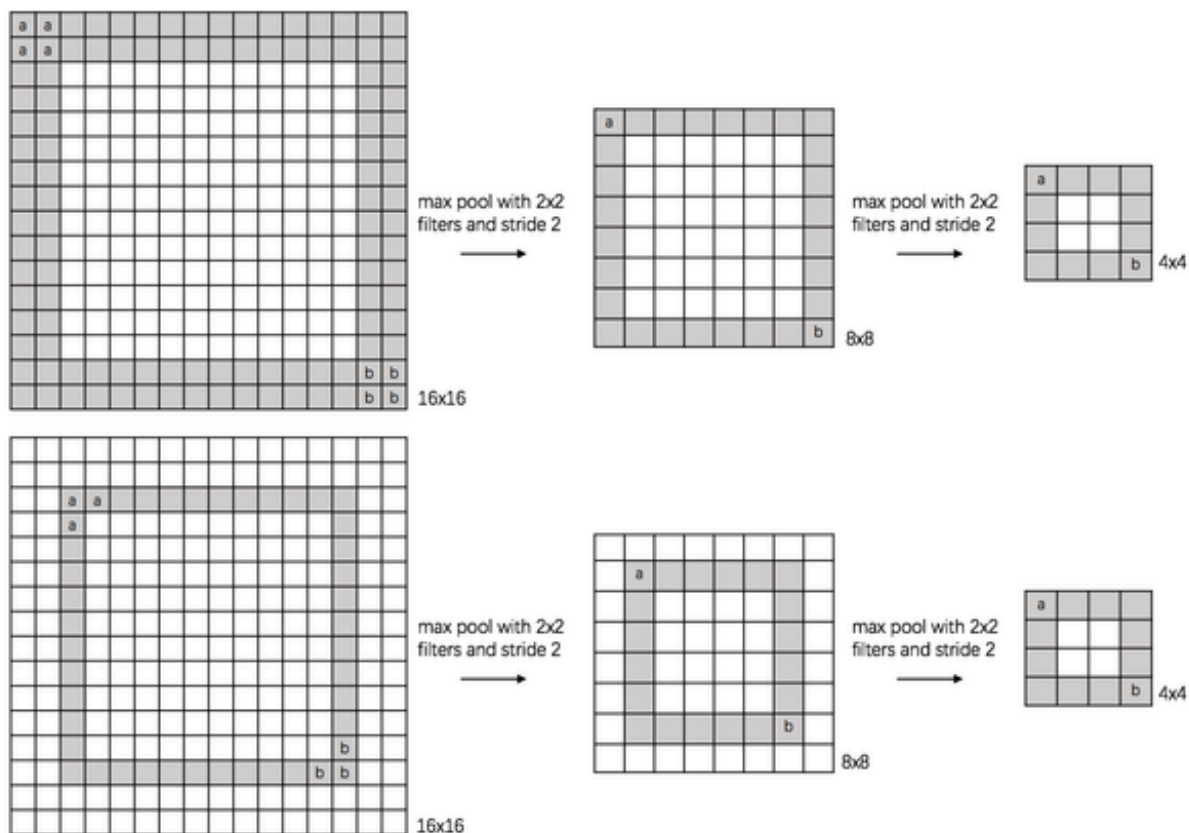
在经过最大池化层后，可以看到outA和outB的结果图是一样的，而且它们的大小都变成了2\*2

---

# 卷积神经网络之网络结构

## (2) 池化层

池化层的尺度不变性



左图是表示数字0的识别，上下两张图的区别在于0的尺度不一样。同样的经过两次最大池化之后，得到的特征矩阵是相同的。



# 卷积神经网络之网络结构

## (2) 池化层

```
# a的灰度图
a = [
    [10, 10, 10, 10, 10, 10, 10, 10],
    [10, 0, 0, 0, 0, 0, 0, 10],
    [10, 0, 0, 0, 0, 0, 0, 10],
    [10, 0, 0, 0, 0, 0, 0, 10],
    [10, 0, 0, 0, 0, 0, 0, 10],
    [10, 0, 0, 0, 0, 0, 0, 10],
    [10, 0, 0, 0, 0, 0, 0, 10],
    [10, 10, 10, 10, 10, 10, 10, 10],
]

# b的灰度图
b = [
    [0, 0, 0, 0, 0, 0, 0, 0],
    [0, 10, 10, 10, 10, 10, 10, 0],
    [0, 10, 0, 0, 0, 0, 10, 0],
    [0, 10, 0, 0, 0, 0, 10, 0],
    [0, 10, 0, 0, 0, 0, 10, 0],
    [0, 10, 0, 0, 0, 0, 10, 0],
    [0, 10, 10, 10, 10, 10, 10, 0],
    [0, 0, 0, 0, 0, 0, 0, 0],
]
```

池化层的尺度不变性

上面a、b两幅图可以看到b的白色圈是a的白色圈缩小后形成的，且大小都为8\*8

# 卷积神经网络之网络结构

---

## (2) 池化层

池化层的尺度不变性

```
outA shape:torch.Size([4, 4])  
outB shape:torch.Size([4, 4])
```

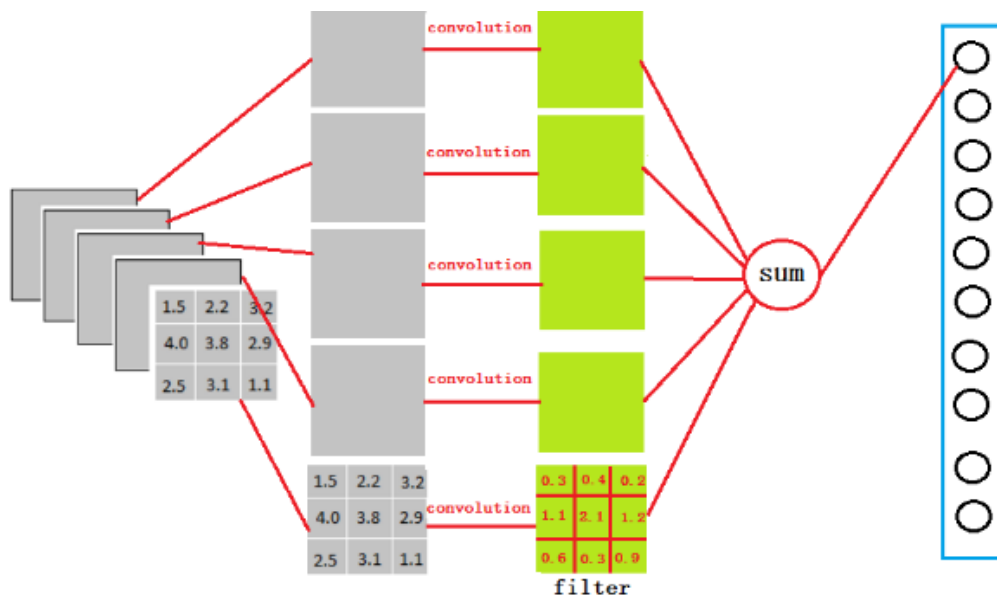
在经过最大池化后，outA和outB的特征图变成一样的了，同时它们的大小变成了4\*4

# 卷积神经网络之网络结构

## (3) 全连接层

全连接层的作用主要是将网络提取到的特征整合到一起

全连接层的每个节点都与上一层所有的节点相连，所以全连接的参数会特别多



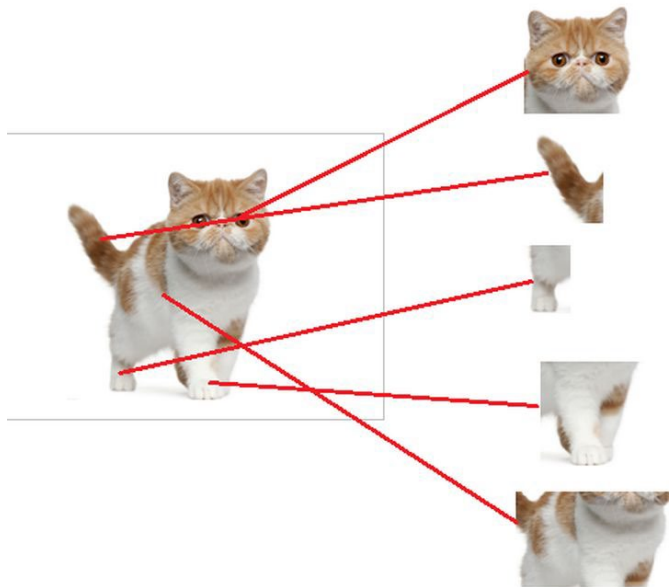
全连接实际上和卷积操作有点类似，只是全连接时，卷积核的大小和上一层输出的特征图大小相同，假如上一层的输出特征图是 $5 \times 5$ ，那么在全连接层中卷积核的大小就是 $5 \times 5$ ，这样就会将特征图卷积成一个值

# 卷积神经网络之网络结构

## (3) 全连接层

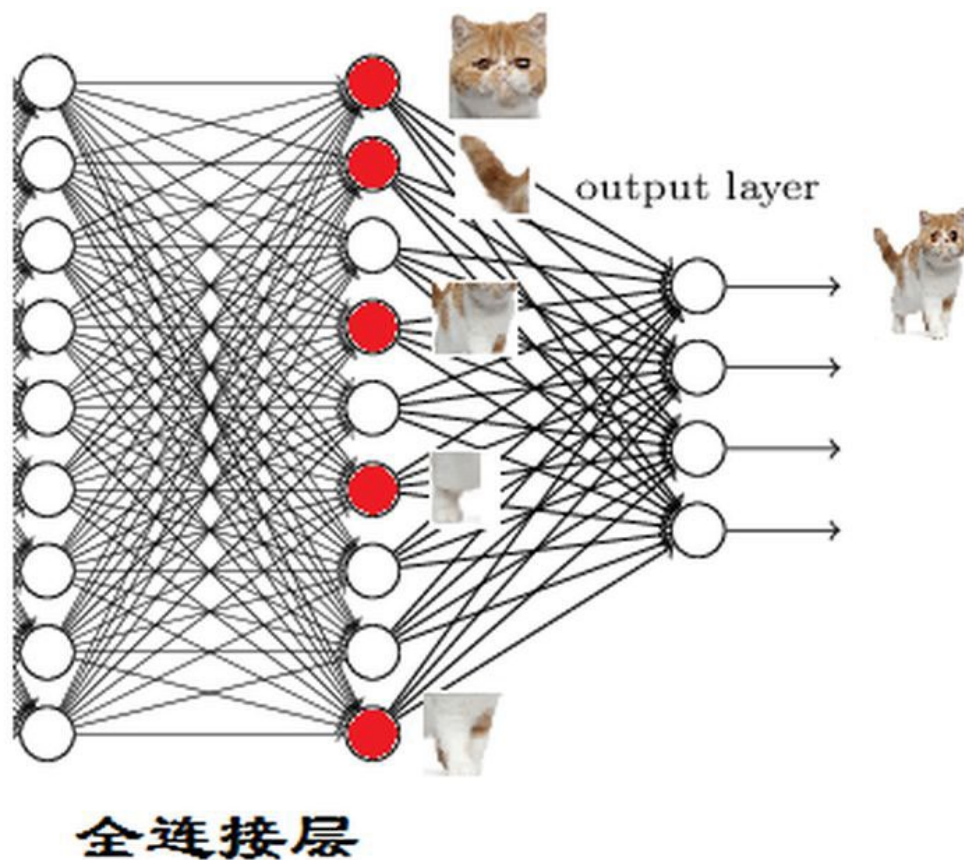
在卷积神经网络中，大部分网络的结构是有两层以上的全连接层，这是为了更好地拟合非线性数据

在全连接层之前的网络层主要是为了提取特征，全连接层的作用是为了分类，现在我们要去判断一幅图像是不是猫。假设在进行到全连接层时，得到了猫头、猫尾、猫身等特征，那么我们就判断图像是猫。



# 卷积神经网络之网络结构

## (3) 全连接层

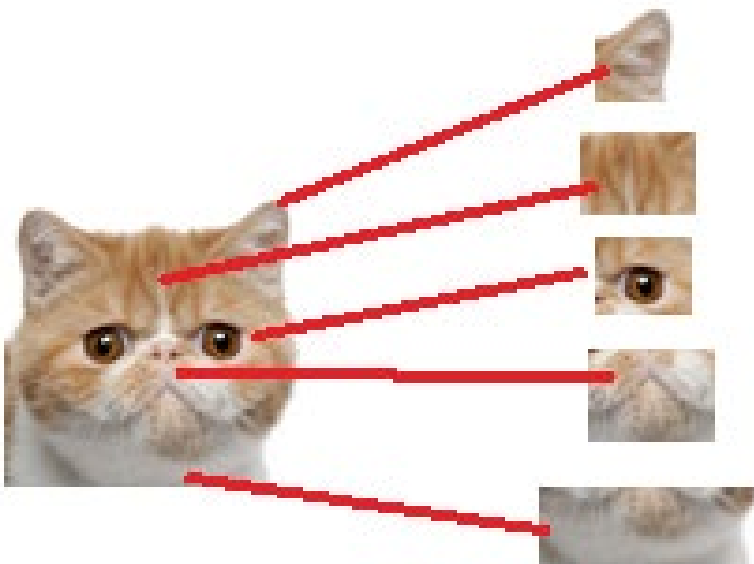


如左图所示，红色的神经元表示对应的特征被找到，经过全连接层将这些特征组合到一起，发现最符合这些特征的是猫，那么网络就会预测当前的输入图片是猫。

# 卷积神经网络之网络结构

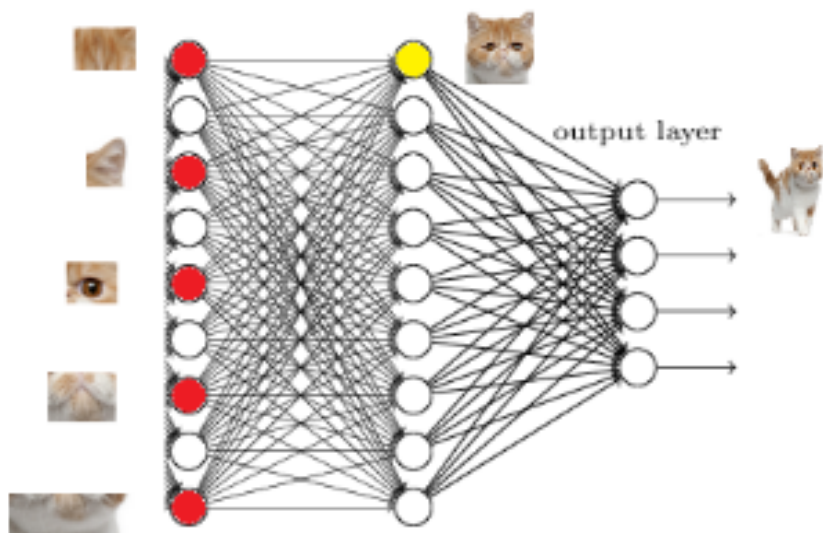
## (3) 全连接层

我们往前再走一层，要得到猫头的特征，就要先找到猫眼、耳朵、鼻子、嘴巴等特征



# 卷积神经网络之网络结构

## (3) 全连接层



于是在找到这些子特征之后，就能判断这些特征是否能组成猫头了。  
这些子特征例如猫眼，就是通过之前的卷积层、池化层得来的了。



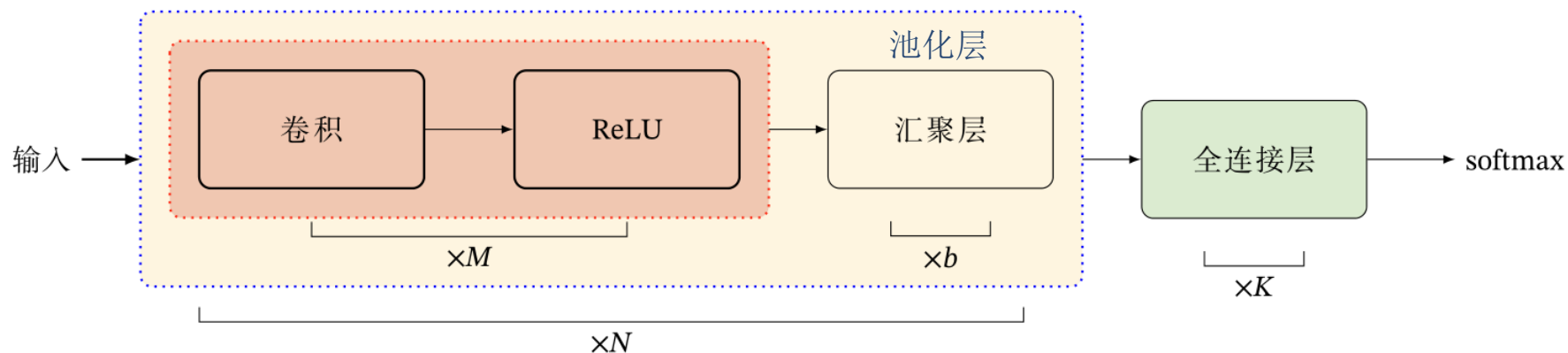
# 卷积网络结构

► 卷积网络是由卷积层、汇聚层、全连接层交叉堆叠而成。

► 趋向于小卷积、大深度

► 趋向于全卷积

► 典型结构



► 一个卷积块为连续 $M$ 个卷积层和 $b$ 个汇聚层 ( $M$ 通常设置为 $2 \sim 5$ ,  $b$ 为 $0$ 或 $1$ )。一个卷积网络中可以堆叠 $N$ 个连续的卷积块, 然后在接着 $K$ 个全连接层 ( $N$ 的取值区间比较大, 比如 $1 \sim 100$ 或者更大;  $K$ 一般为 $0 \sim 2$ )。



## 典型的卷积网络

# 经典的卷积神经网络

---

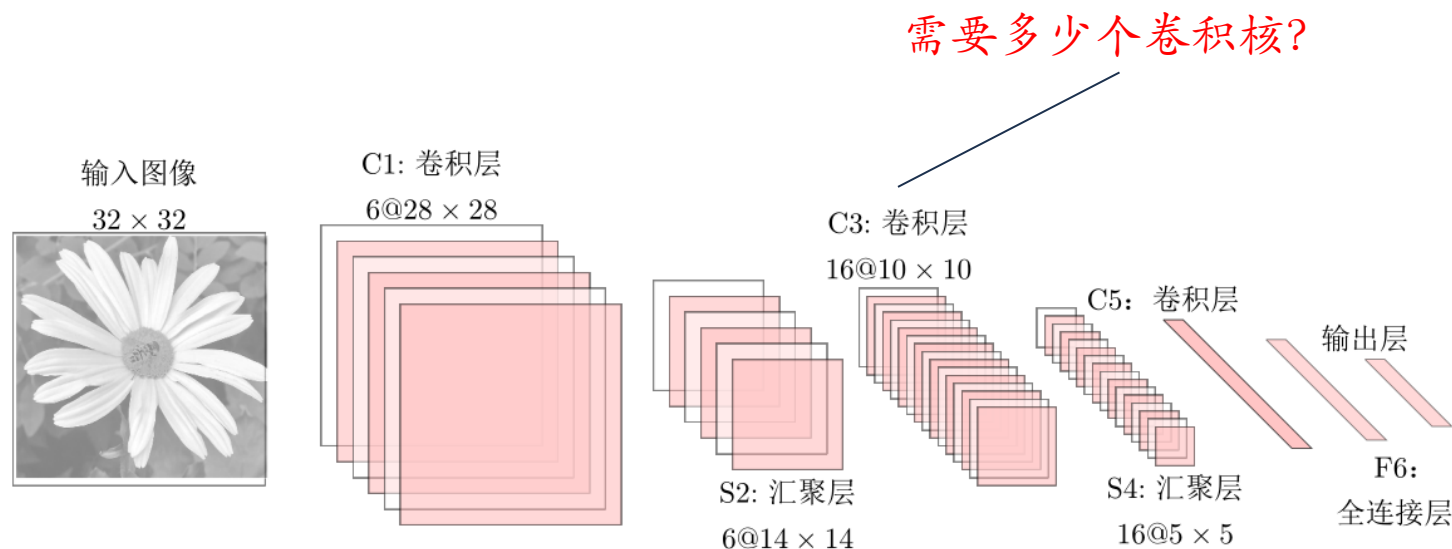
卷积神经网络的基本网络结构包括了卷积层、池化层和全连接层，许多研究在探索如何将这一些基本的网络模块整合起来以构建出一个高效的神经网络。

整合这些模块确实需要深入的理解和直觉上的感受，但更需要大量地**阅读**和**学习**别人的案例，所以这一节将会介绍一些经典的卷积神经网络：

- LeNet: LeCun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11): 2278-2324.
- AlexNet: Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[J]. Advances in neural information processing systems, 2012, 25: 1097-1105.
- ZFNet: Zeiler M D, Fergus R. Visualizing and understanding convolutional networks[C]//European conference on computer vision. Springer, Cham, 2014: 818-833.
- VGGNet: Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.

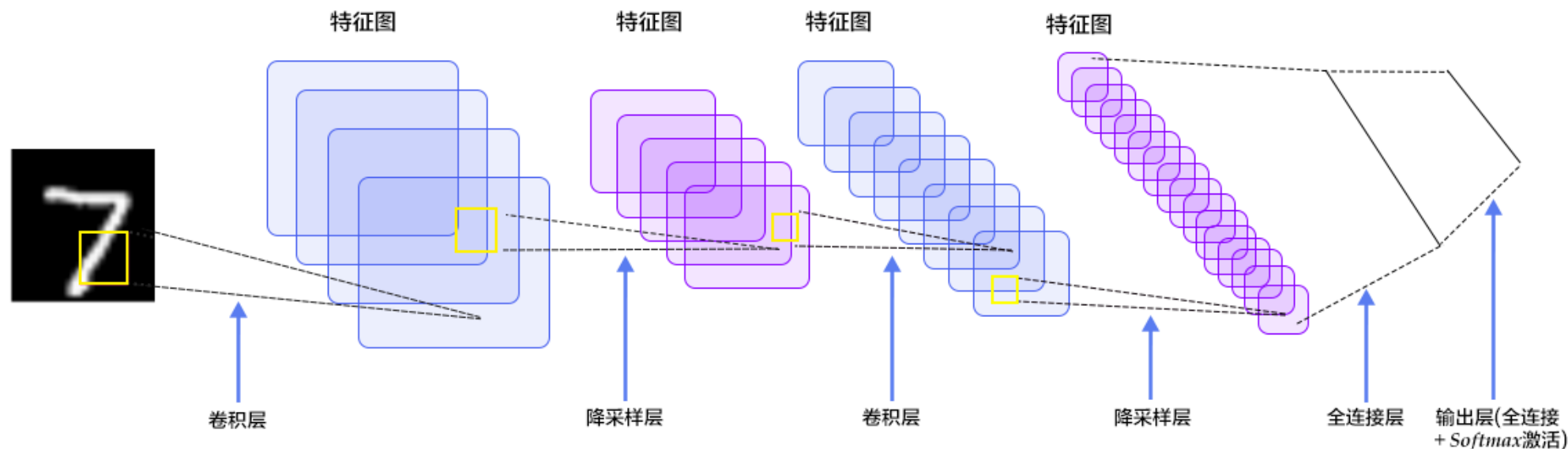
# LeNet-5

- ▶ **LeNet-5 是一个非常成功的神经网络模型。**
- ▶ 基于 LeNet-5 的手写数字识别系统在 90 年代被美国很多银行使用，用来识别支票上面的手写数字。
- ▶ LeNet-5 共有 7 层。



# 经典的卷积神经网络

## (1) LeNet



LeNet是卷积神经网络做识别任务的开山之作，虽然这篇论文的网络结构现在已经很少使用，但是它对后续卷积网络的发展起到了奠基的作用。而且由于它成功地应用于手写体识别等任务中，因此吸引了学术界和工业界的注意力。

# 经典的卷积神经网络

---

## (1) LeNet

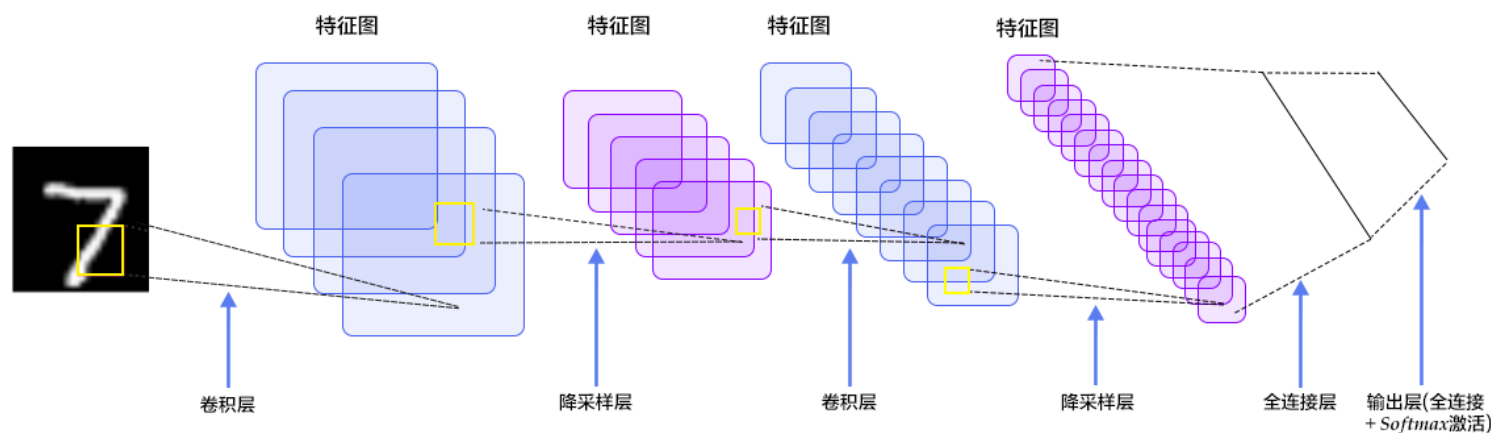
在正式开始介绍LeNet之前，先谈谈卷积神经网络的设计思想，一共有三个方面：

- 局部感受野：基于图像局部相关的原理，保留了图像的局部结构，同时减少了网络的权值
- 权值共享：也是基于图像局部相关的原理，同时减少了网络的权值参数
- 下采样：对平移和形变更加鲁棒，实现特征的不变性，同时起到了一定的降维的作用

从以上三方面我们可以看出，卷积网络的目的就是减小参数并适应不同尺度，它之所以能实现的依据就是图像的局部相关原则，图像的像素是有规则的顺序排列，这是所有方法能够起作用的前提。

# 经典的卷积神经网络

## (1) LeNet



网络结构:

卷积层1 (Conv1)

最大池化层1 (Max-Pooling1)

卷积层2 (Conv2)

最大池化层2 (Max-Pooling2)

全连接层1 (FC1)

全连接层2 (FC2)

全连接层3 (FC3)

# 经典的卷积神经网络

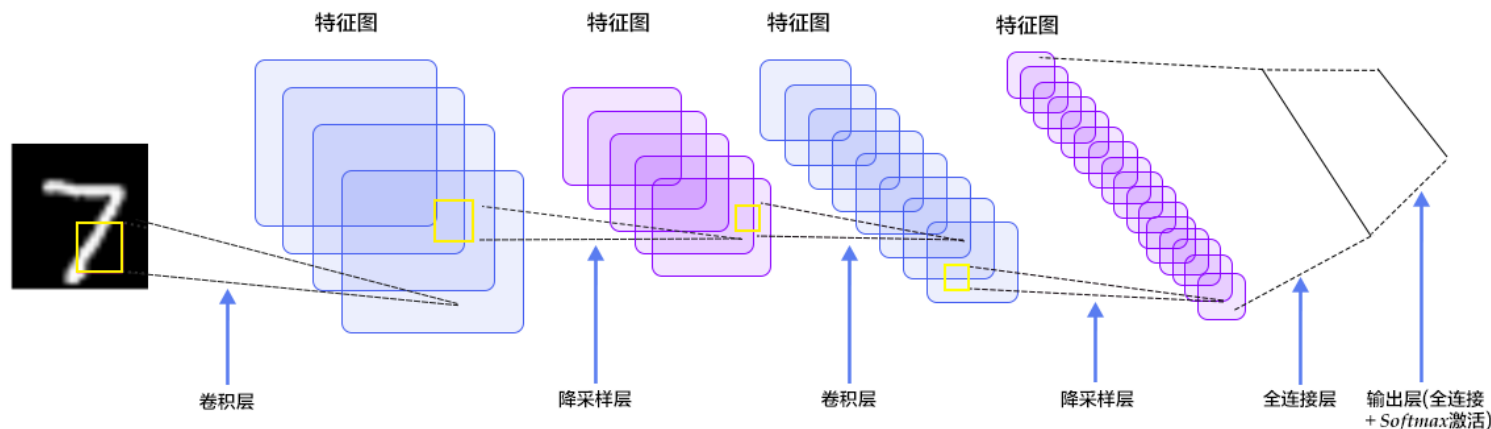
## (1) LeNet

输入图像是：32\*32\*1

卷积层1 (Conv1)：卷积核大小为5\*5，步长为1，padding为0，卷积核的个数是6

$$(W-F+2P) / S + 1 = (32-5+2*0) / 1 + 1 = 28$$

输出的特征图大小：28\*28\*6





# 经典的卷积神经网络

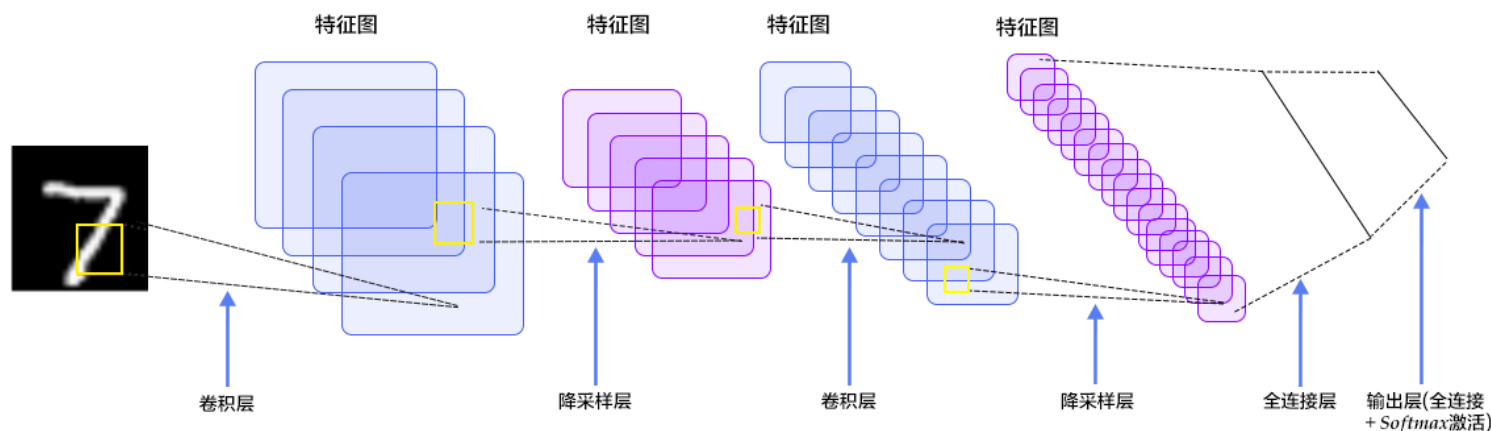
## (1) LeNet

输入特征是：28\*28\*6

最大池化层1 (Max-Pooling1) :  
卷积核大小为2\*2，步长为2，  
padding为0

$$(W-F+2P) / S + 1 = (28-2+2*0) / 2 + 1 = 14$$

输出的特征图大小：14\*14\*6



# 经典的卷积神经网络

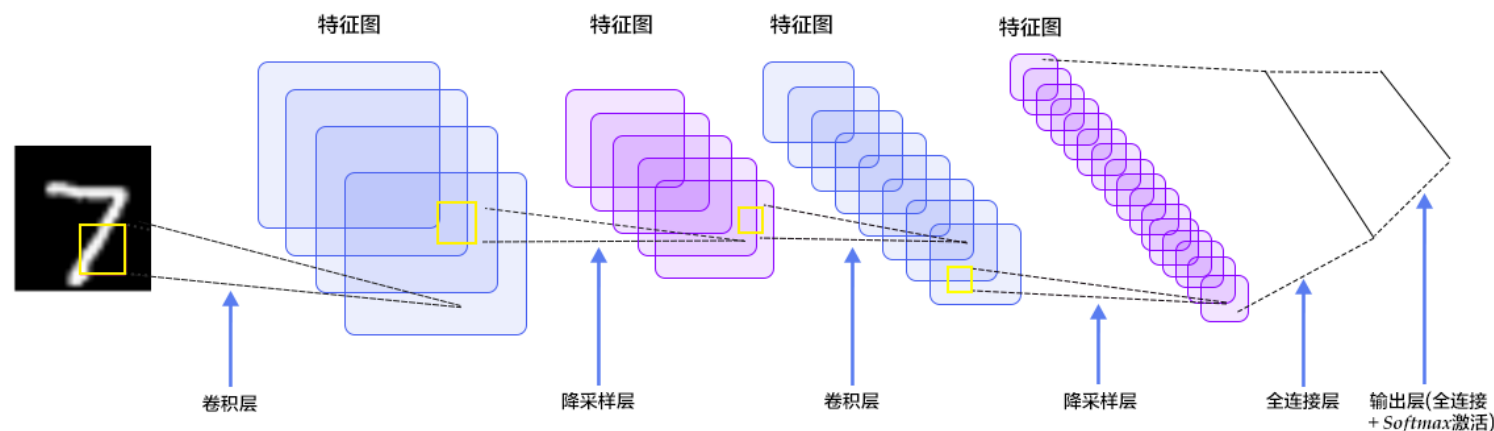
## (1) LeNet

输入特征是：14\*14\*6

卷积层2 (ConV2)：卷积核大小为5\*5，步长为1，padding为0，卷积核个数为16

$$(W-F+2P) / S + 1 = (14-5+2*0) / 1 + 1 = 10$$

输出的特征图大小：10\*10\*16



# 经典的卷积神经网络

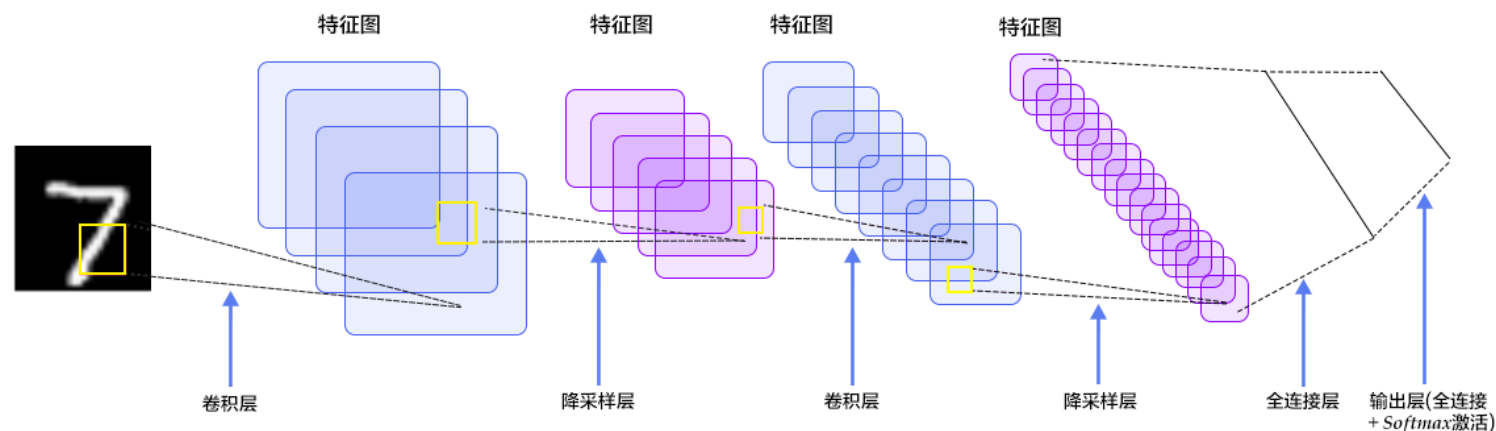
## (1) LeNet

输入特征是：10\*10\*16

最大池化层2 (Max-Pooling2) :  
卷积核大小为2\*2，步长为2，  
padding为0

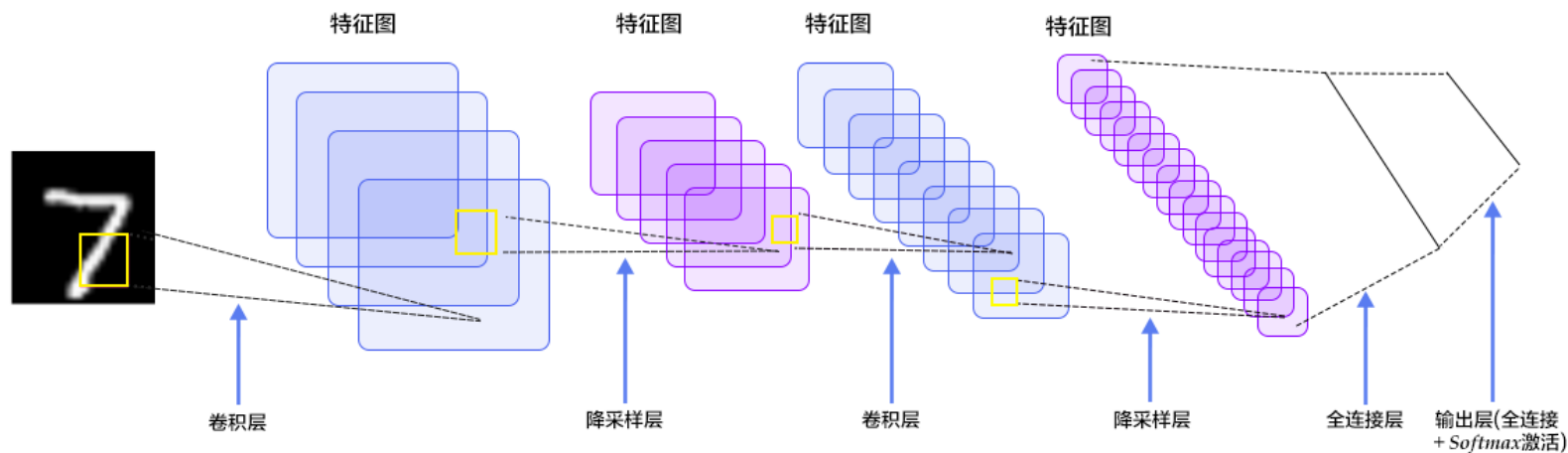
$$(W-F+2P) / S + 1 = (10-2+2*0) / 2 + 1 = 5$$

输出的特征图大小：5\*5\*16



# 经典的卷积神经网络

## (1) LeNet



最后再经过3层全连接层，利用Softmax得到10个类别的概率，这样就可以知道输入图像是属于10个类别的哪一个

# AlexNet

---

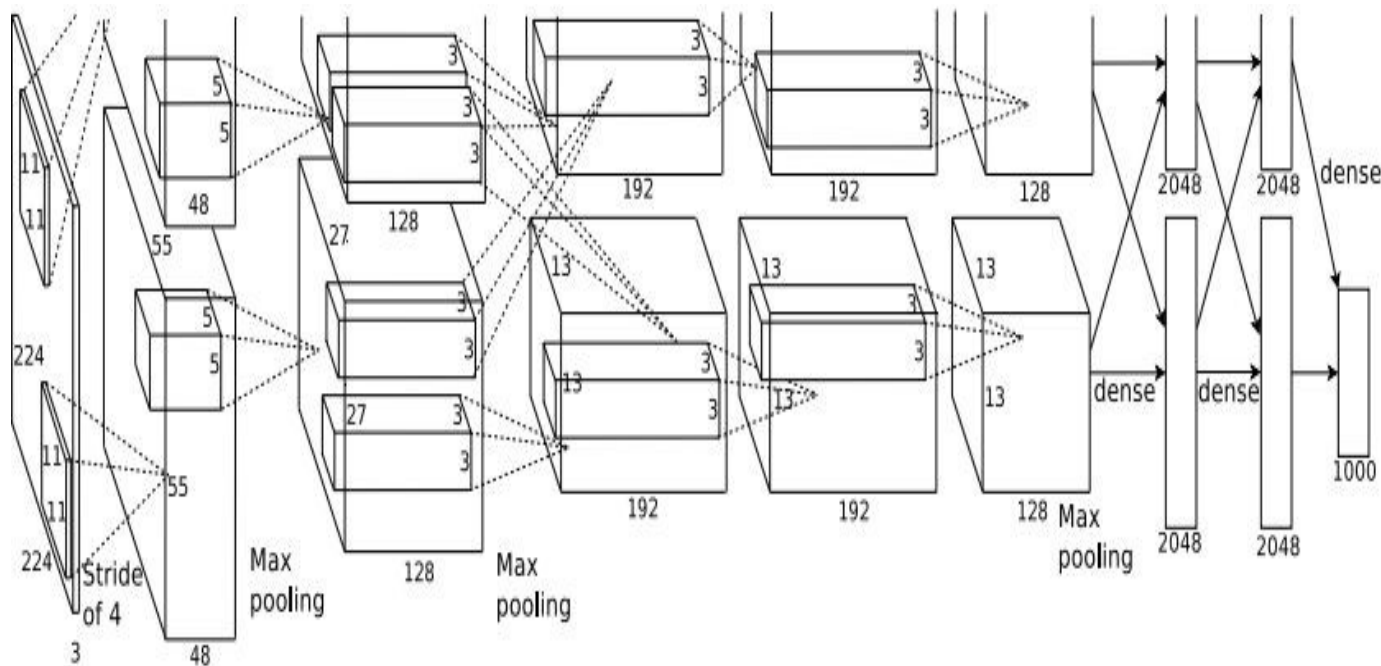
## ▶ 2012 ILSVRC winner

- ▶ (top 5 error of 16% compared to runner-up with 26% error)
- ▶ 第一个现代深度卷积网络模型
  - ▶ 首次使用了很多现代深度卷积网络的一些技术方法
    - 使用GPU进行并行训练，采用了ReLU作为非线性激活函数，使用Dropout防止过拟合，使用数据增强
- ▶ 5个卷积层、3个汇聚层和3个全连接层

AlexNet是首次在大规模图像数据集上实现的深层卷积神经网络，其在2012年ILSVRC（ImageNet Large Scale Visual Recognition Challenge）竞赛中取得了冠军，将分类的准确率由传统的70%+提升到80%+，碾压了其他传统的特征方法，使得计算机视觉从业者从繁重的特征工程中解脱出来，做到从数据中自动提取需要的特征。AlexNet是由Hinton和他的学生Alex Krizhevsky设计的，也是在这之后深度学习开始迅速发展。

# 经典的卷积神经网络

## (2) AlexNet



网络结构：

卷积层1 (Conv1)

最大池化层1 (Max-Pooling1)

卷积层2 (Conv2)

最大池化层2 (Max-Pooling2)

卷积层3 (Conv3)

卷积层4 (Conv4)

卷积层5 (Conv5)

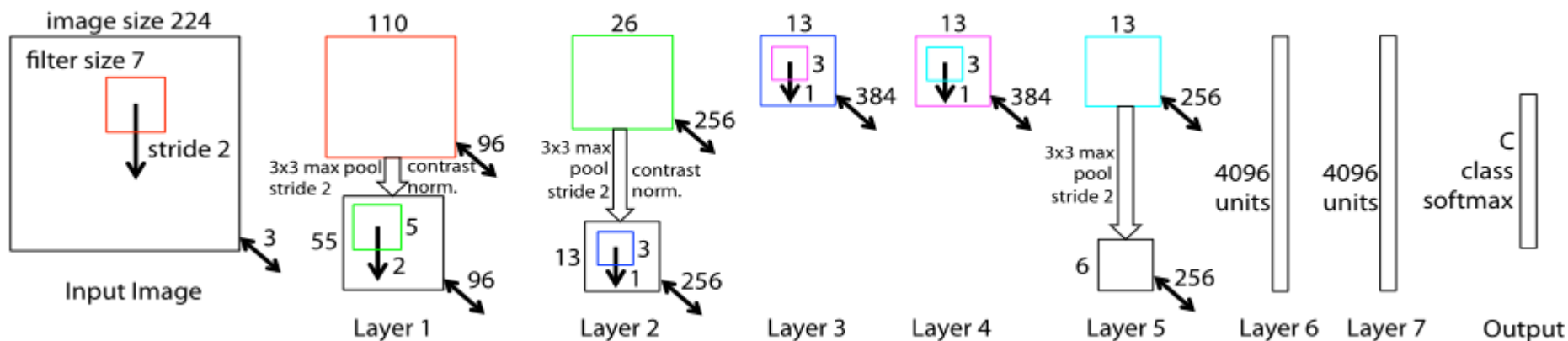
最大池化层3 (Max-Pooling3)

全连接层1 (FC1)

全连接层2 (FC2)

# 经典的卷积神经网络

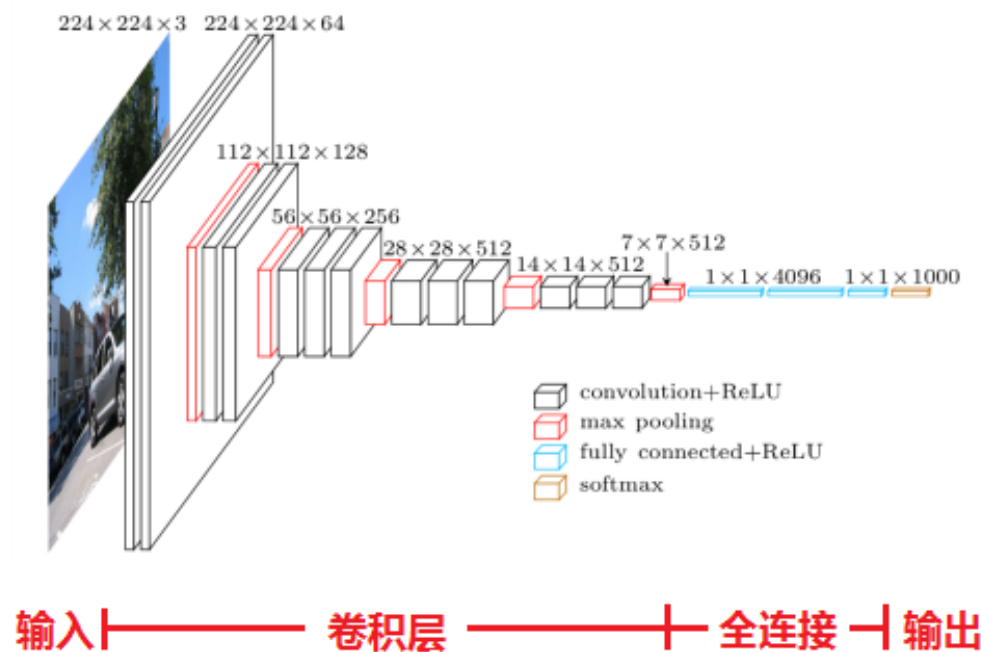
## (3) ZFNet



ZFNet是2013年的ILSVRC竞赛的冠军模型，但是ZFNet的网络结构在AlexNet上并没有做很大的改进，只是将第一层的卷积核大小从 $11 \times 11$ 改成了 $7 \times 7$ ，并将步长改为了2。ZFNet只使用了一块GPU来进行训练，并且它的意义在于可视化了网络的特征图，并且通过实验证明了网络的深度增加时，可以学习到更具区分度的特征。

# 经典的卷积神经网络

## (4) VGG

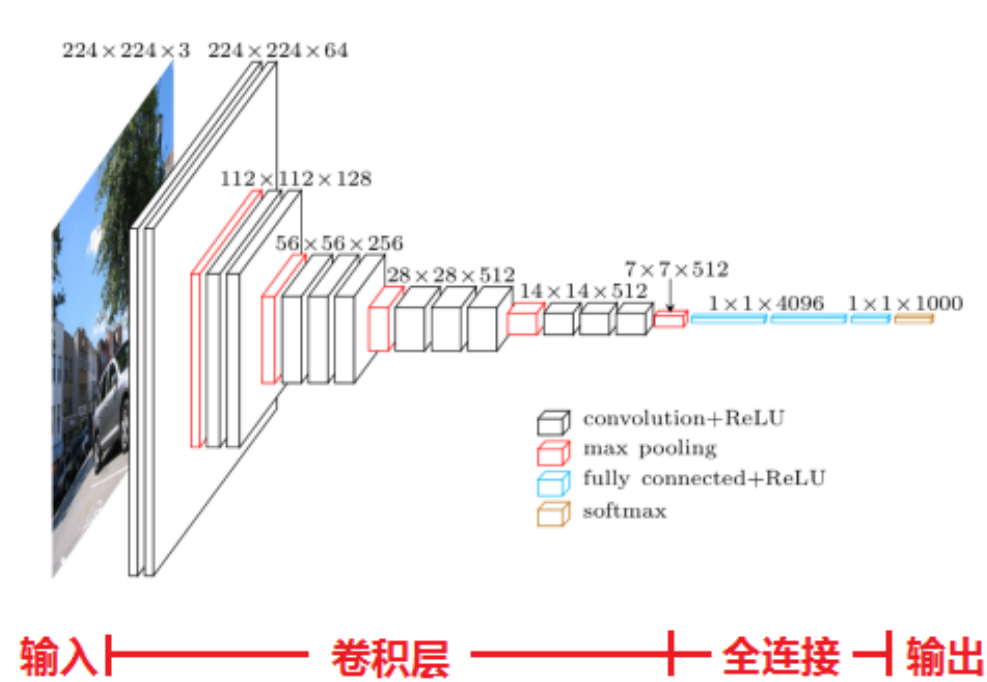


2014年，牛津大学计算机视觉组和Google DeepMind公司一起研发出了VGGNet，并取得了ILSVRC2014比赛分类项目的第二名和定位项目的第一名。VGGNet探索了卷积神经网络的深度和性能之间的关系，并且构造了16~19层深的卷积神经网络，证明了网络的深度能够在一定的程度上影响网络的最终性能，使错误率大幅下降，同时拓展性又很强，并在之后的卷积神经网络的应用中仍然常常被用来提取图像特征



# 经典的卷积神经网络

## (4) VGG



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv3-128	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

右图中的表格是VGG论文中截取的，作者尝试了6个不同的配置，但是在使用的过程中常常使用D配置，也被称为VGG16。

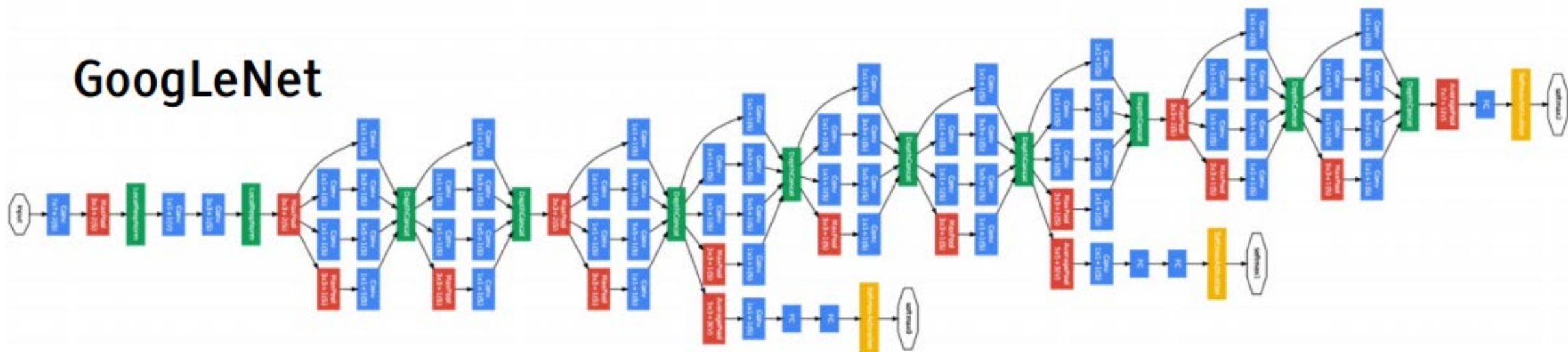
## (5) Inception网络

### ► 2014 ILSVRC winner (22层)

► 参数: GoogLeNet: 4M VS AlexNet: 60M

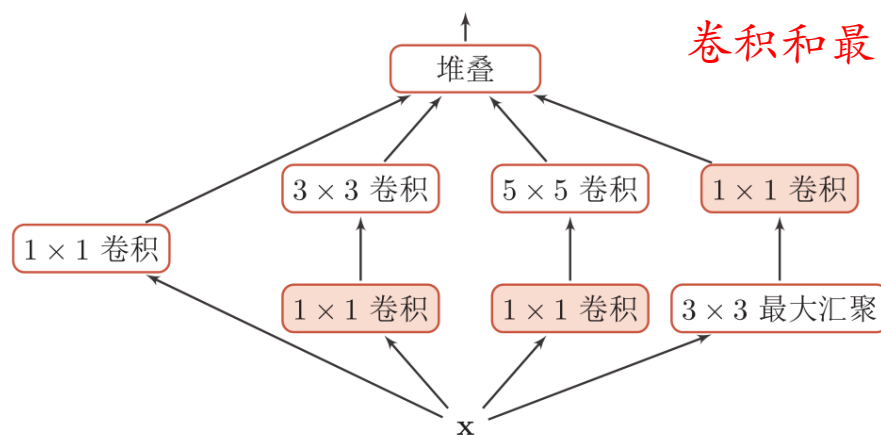
► 错误率: 6.7%

► Inception网络是由有多个inception模块和少量的汇聚层堆叠而成。



# Inception模块 v1

- ▶ 在卷积网络中，如何设置卷积层的卷积核大小是一个十分关键的问题。
- ▶ 在Inception网络中，一个卷积层包含多个不同大小的卷积操作，称为Inception模块。
- ▶ Inception模块同时使用 $1 \times 1$ 、 $3 \times 3$ 、 $5 \times 5$ 等不同大小的卷积核，并将得到的特征映射在深度上拼接（堆叠）起来作为输出特征映射。



# Inception模块 v3

- ▶ 用多层小卷积核替换大卷积核，以减少计算量和参数量。
- ▶ 使用两层 $3 \times 3$ 的卷积来替换v1中的 $5 \times 5$ 的卷积
- ▶ 使用连续的 $n \times 1$ 和 $1 \times n$ 来替换 $n \times n$ 的卷积。

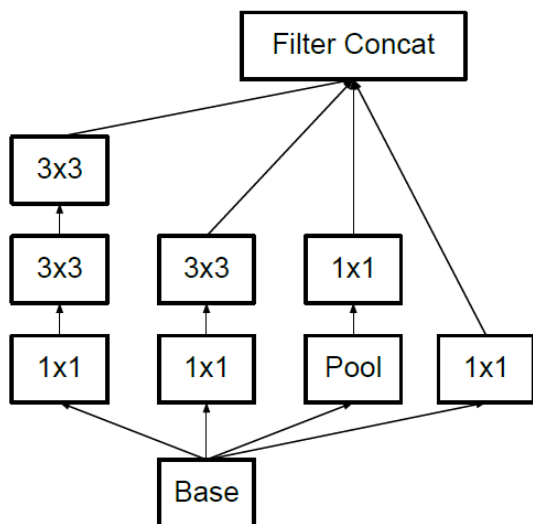


Figure 5. Inception modules where each  $5 \times 5$  convolution is replaced by two  $3 \times 3$  convolution, as suggested by principle 3 of Section 2.

<http://blog.csdn.net/xbinworld>

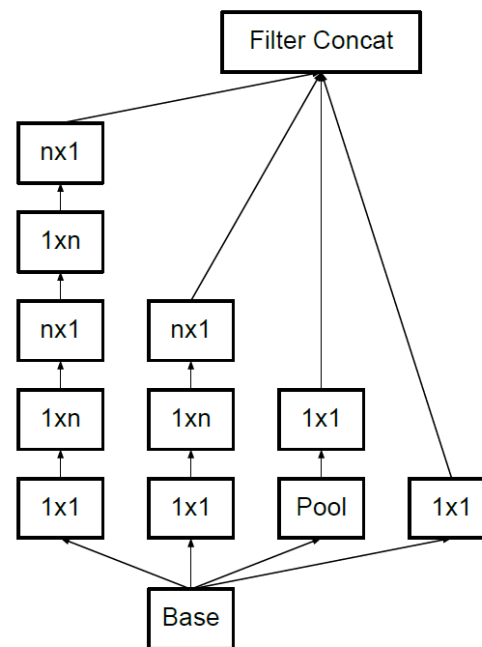


Figure 6. Inception modules after the factorization of the  $n \times n$  convolutions. In our proposed architecture, we chose  $n = 7$  for the  $17 \times 17$  grid. (The filter sizes are picked using principle 3)


<http://blog.csdn.net/xbinworld>

## (6) 残差网络 ResNet

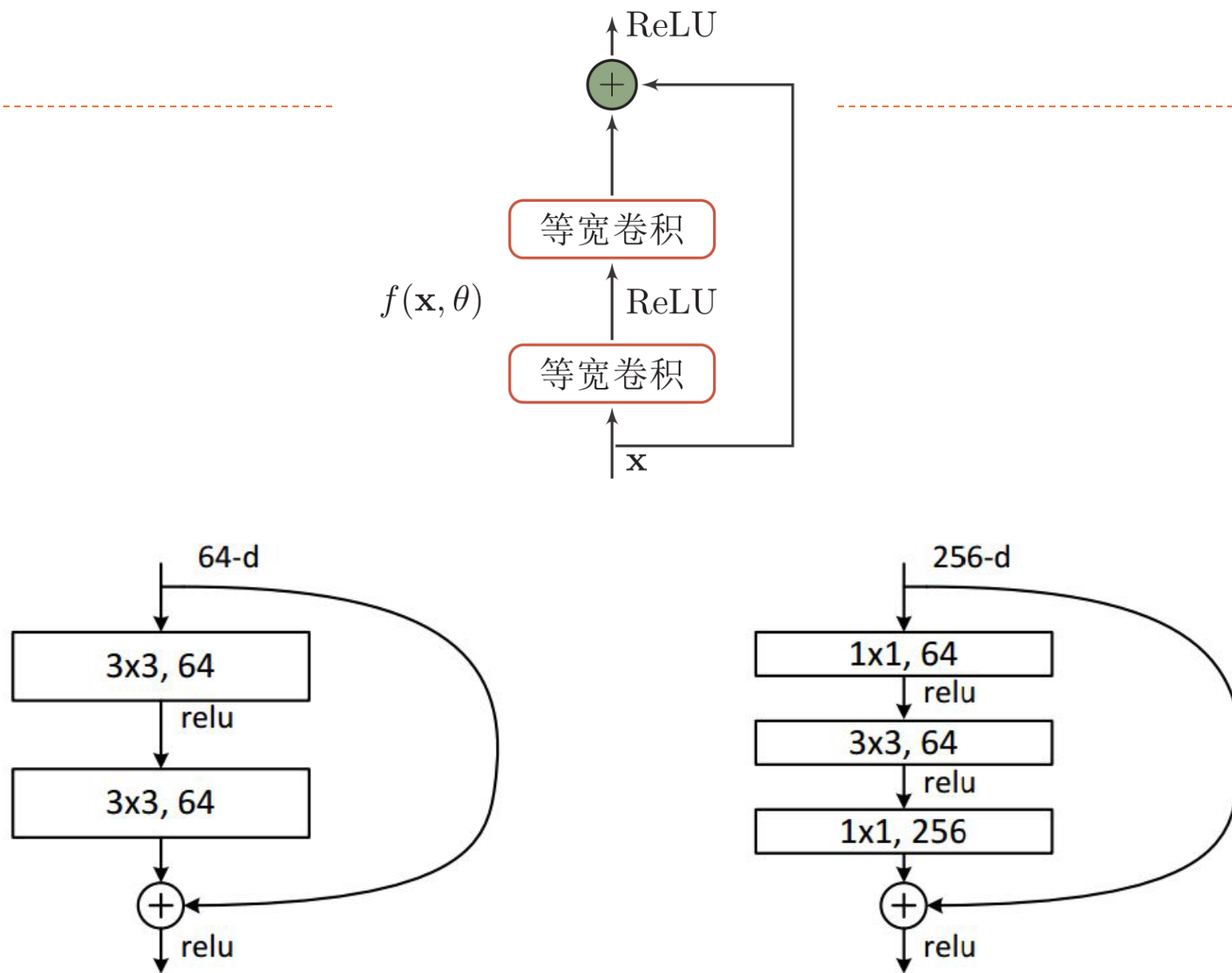
---

- ▶ 残差网络 (Residual Network, ResNet) 是通过给非线性的卷积层增加**直连边**的方式来提高信息的传播效率。
- ▶ 假设在一个深度网络中, 我们期望一个非线性单元 (可以为一层或多层的卷积层)  $f(\mathbf{x}, \theta)$  去逼近一个目标函数为  $h(\mathbf{x})$ 。
- ▶ 将目标函数拆分成两部分: **恒等函数** 和 **残差函数**

$$h(\mathbf{x}) = \underbrace{\mathbf{x}}_{\text{恒等函数}} + \underbrace{(h(\mathbf{x}) - \mathbf{x})}_{\text{残差函数}}$$

  $f(\mathbf{x}, \theta)$

# 残差单元



► 2015 ILSVRC winner (152层)

The diagram illustrates the architecture of three VGG models: VGG-19, VGG-16, and VGG-19 residual. Each model starts with an input image of size 224x224x3.

- VGG-19:** The architecture consists of seven layers of convolutional and pooling operations. The output sizes at each stage are: 112, 56, 28, 14, 7, and 1. The final output size is 1.
- VGG-16:** The architecture is similar to VGG-19 but with fewer layers. The output sizes at each stage are: 112, 56, 28, 14, and 7. The final output size is 1.
- VGG-19 residual:** This model is a modified version of VGG-19, incorporating residual connections. The output sizes at each stage are: 112, 56, 28, 14, 7, and 1. The final output size is 1.

The diagram shows the flow of data from the input image through various convolutional and pooling layers, with output sizes indicated at each stage. The layers are color-coded: blue for convolutional layers, orange for pooling layers, and green for fully connected layers. The residual connections are shown as dashed lines.