



浙江大学城市学院
ZHEJIANG UNIVERSITY CITY COLLEGE

卷积神经网络

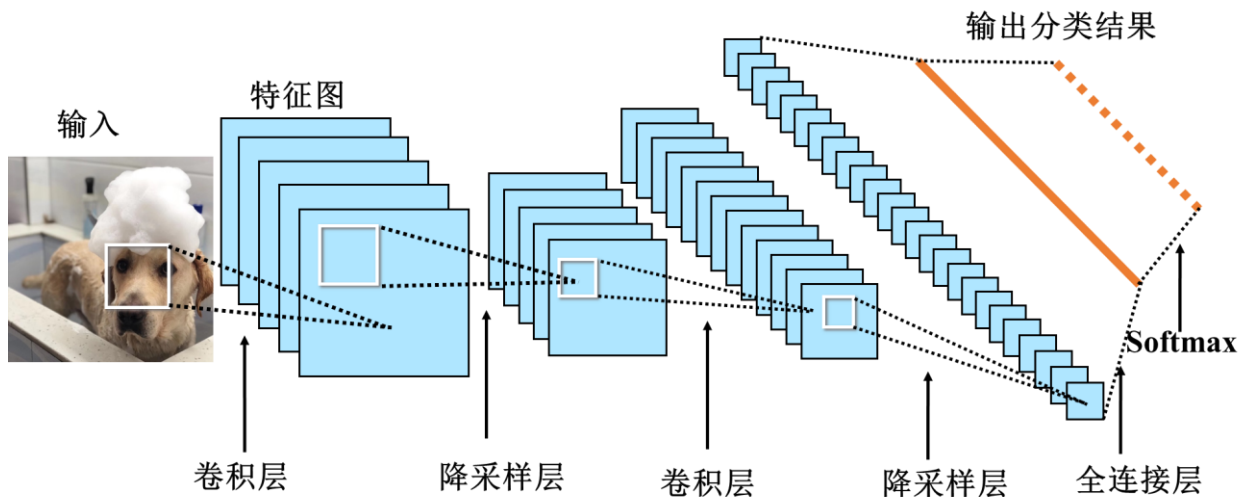


卷积神经网络的提出

- 1962年Hubel和Wiesel通过对猫视觉皮层细胞的研究，提出了感受野(receptive field)的概念。视觉皮层的神经元就是局部接受信息的，只受某些特定区域刺激响应，而不是对全局图像进行感知。
- 1984年日本学者Fukushima基于感受野概念提出神经认知机(neocognitron)。
- CNN可看作是神经认知机的推广形式。

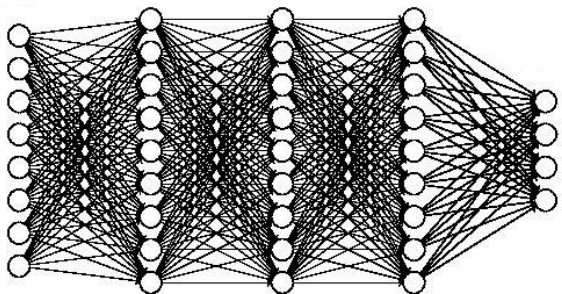


卷积神经网络结构

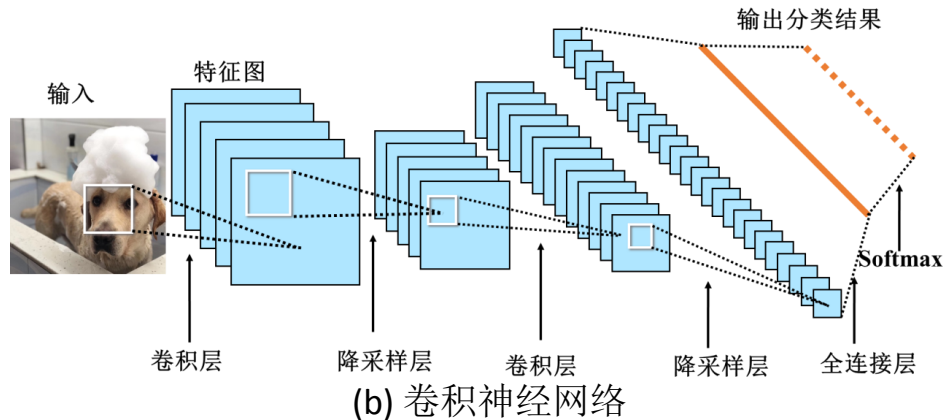


- CNN是一个多层的神经网络，每层由多个二维平面组成，其中每个平面由多个独立神经元组成。
- CNN结构与全连接神经网络的异同

卷积神经网络结构



(a) 全连接神经网络



(1) **输入层**：将每个像素代表一个特征节点输入到网络中。

(2) **卷积层**：卷积运算的主要目的是使原信号特征增强，并降低噪音。（在线演示）

(3) **降采样层**：降低网络训练参数及模型的过拟合程度。

(4) **全连接层**：对生成的特征进行加权。

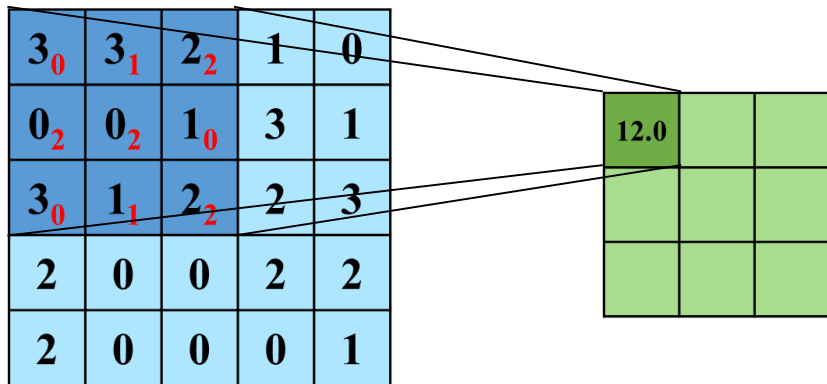


卷积



在介绍卷积神经网络的基本概念之前，我们先做一个矩阵运算：

(1) **求点积**：将 5×5 输入矩阵中 3×3 深蓝色区域中每个元素分别与其对应位置的权值（红色数字）相乘，然后再相加，所得到的值作为 3×3 输出矩阵（绿色）的第一个元素。



$$3 \times 0 + 3 \times 1 + 2 \times 2 + 0 \times 2 + 0 \times 2 + 1 \times 0 + 3 \times 0 + 1 \times 1 + 2 \times 2 = 12$$

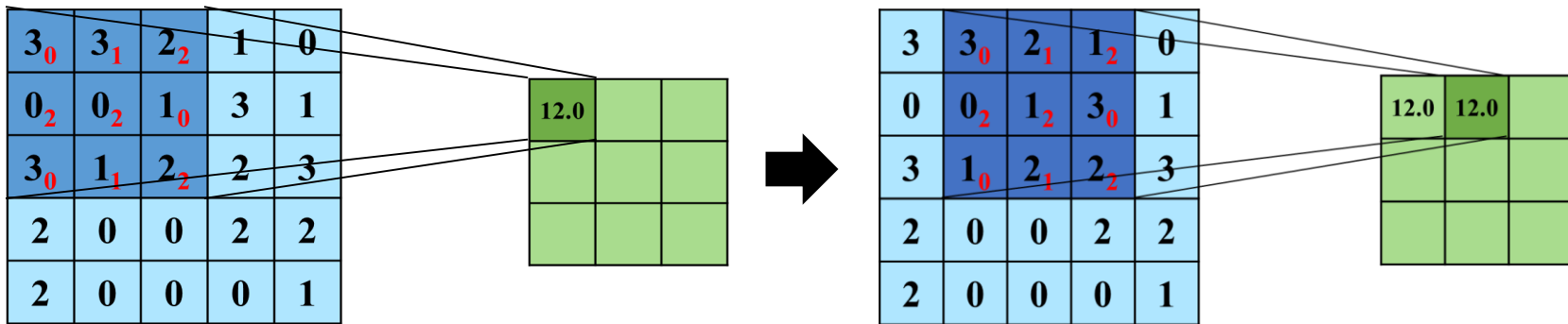


卷积



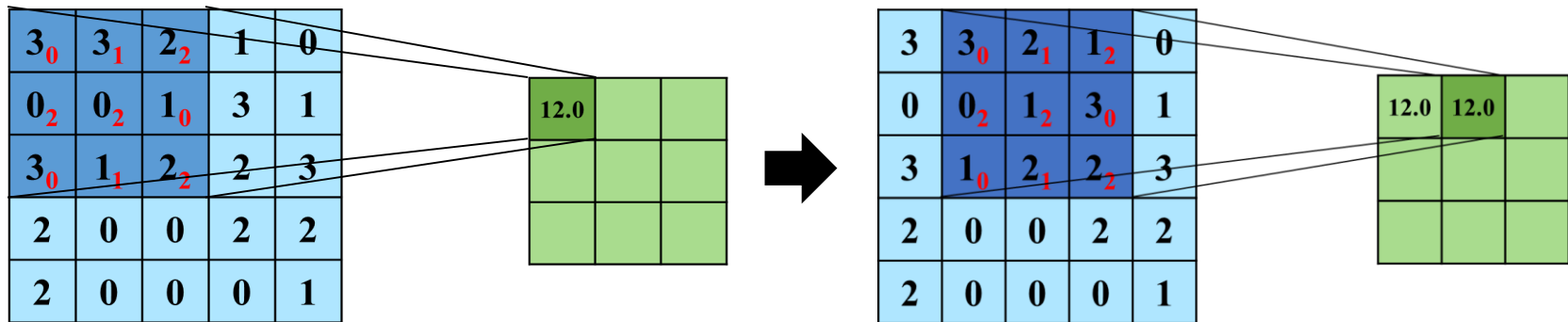
(2) **滑动窗口**：将 3×3 权值矩阵向右移动一个格（即，步长为1）

(3) **重复操作**：同样地，将此时深色区域内每个元素分别与对应的权值相乘然后再相加，所得到的值作为输出矩阵的第二个元素；重复上述“求点积-滑动窗口”操作，直至输出矩阵所有值被填满。





卷积



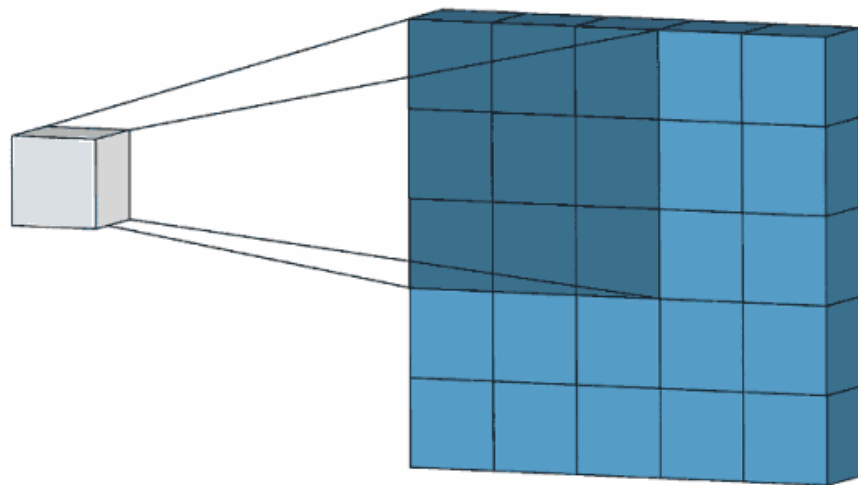
- 卷积核在 2 维输入数据上“滑动”，对当前输入部分的元素进行矩阵乘法，然后将结果汇为单个输出像素值，重复这个过程直到遍历整张图像，这个过程就叫做**卷积**
- 这个权值矩阵就是**卷积核**
- 卷积操作后的图像称为**特征图 (feature map)**



卷积



浙江大學城市學院
ZHEJIANG UNIVERSITY CITY COLLEGE

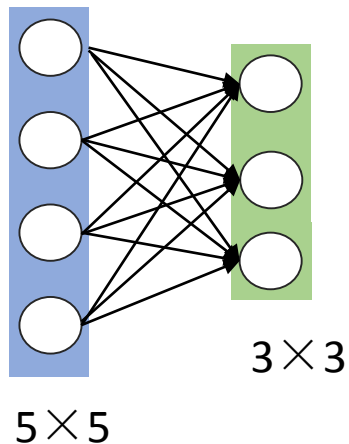




卷积



输入： $5 \times 5 = 25$ ，输出：为 $3 \times 3 = 9$



每个输出都是所有输入数据的加权求和。

➤ 全连接层所需参数： $25 \times 9 = 225$ 。

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

局部连接

每个输出特性不用查看每个输入特征，而只需查看部分输入特征。

权值共享

卷积核在图像上滑动过程中保持不变

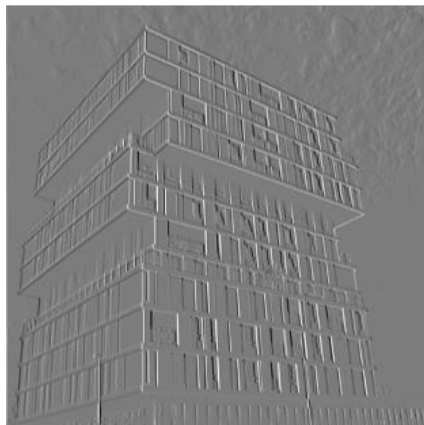
➤ 卷积层所需参数：9。



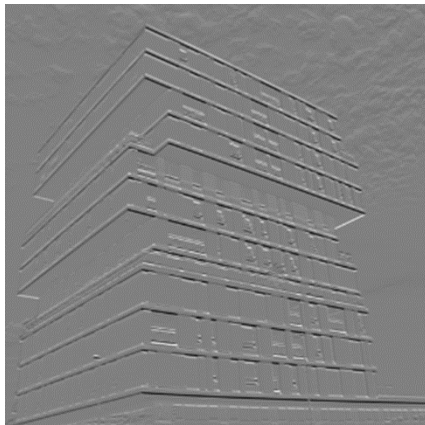
卷积



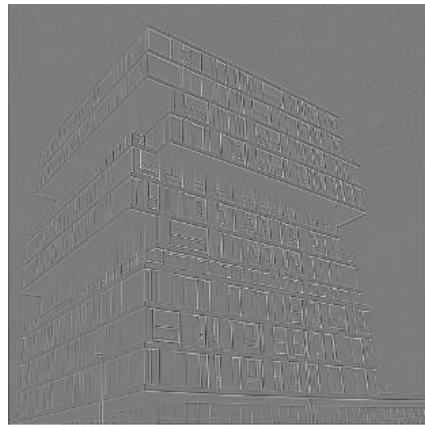
浙江大學城市學院
ZHEJIANG UNIVERSITY CITY COLLEGE



sobel_x



sobel_y



laplace

卷积层：卷积运算的主要目的是使原信号特征增强，并降低噪音。



卷积

```
import numpy as np
from PIL import Image
```

```
def ImgConvolve(image_array, kernel):
```

''' 参数说明 :

image_array : 原灰度图像矩阵

kernel : 卷积核

返回值: 原图像与算子进行卷积后的结果

'''

```
image_arr = image_array.copy()
```

```
img_dim1, img_dim2 = image_arr.shape
```

```
k_dim1, k_dim2 = kernel.shape
```

```
AddW = int((k_dim1-1)/2)
```

```
AddH = int((k_dim2-1)/2)
```

padding填充

```
temp = np.zeros([img_dim1 + AddW*2, img_dim2 + AddH*2])
```

#将原图拷贝到临时图片的中央

```
temp[AddW:AddW+img_dim1, AddH:AddH+img_dim2] = image_arr[:, :]
```

#初始化一张同样大小的图片作为输出图片

```
output = np.zeros_like(a=temp)
```

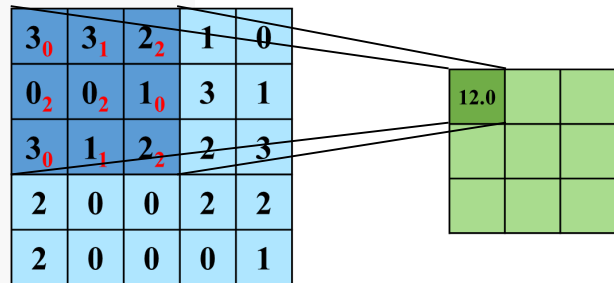
#将扩充后的图和卷积核进行卷积

```
for i in range(AddW, AddW+img_dim1):
```

```
    for j in range(AddH, AddH+img_dim2):
```

```
        output[i][j] = int(np.sum(temp[i-AddW:i+AddW+1, j-AddH:j+AddH+1]*kernel))
```

```
return output[AddW:AddW+img_dim1, AddH:AddH+img_dim2]
```



图像矩阵与权值矩阵点乘



卷积

定义卷积核

```
# 提取竖直方向特征
# sobel_x
kernel_1 = np.array([[-1, 0, 1],
                     [-2, 0, 2],
                     [-1, 0, 1]])

# 提取水平方向特征
# sobel_y
kernel_2 = np.array([[-1, -2, -1],
                     [0, 0, 0],
                     [1, 2, 1]])

# Laplace扩展算子
# 二阶微分算子
kernel_3 = np.array([[1, 1, 1],
                     [1, -8, 1],
                     [1, 1, 1]])
```



卷积操作



```
# 打开图像并转化成灰度图像
image = Image.open("img_087.png").convert("L")

# 将图像转化成数组
image_array = np.array(image)

# 卷积操作
sobel_x = ImgConvolve(image_array, kernel_1)
sobel_y = ImgConvolve(image_array, kernel_2)
laplace = ImgConvolve(image_array, kernel_3)
```

显示

```
# 显示图像

plt.imshow(image_array, cmap=cm.gray)
plt.axis("off")
plt.show()

plt.imshow(sobel_x, cmap=cm.gray)
plt.axis("off")
plt.show()

plt.imshow(sobel_y, cmap=cm.gray)
plt.axis("off")
plt.show()

plt.imshow(laplace, cmap=cm.gray)
plt.axis("off")
plt.show()
```

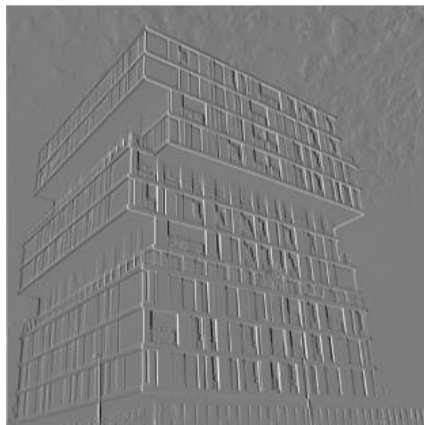




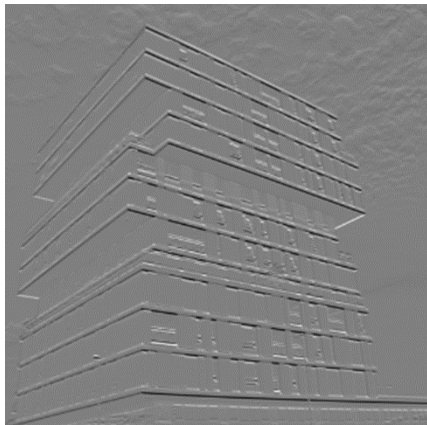
卷积



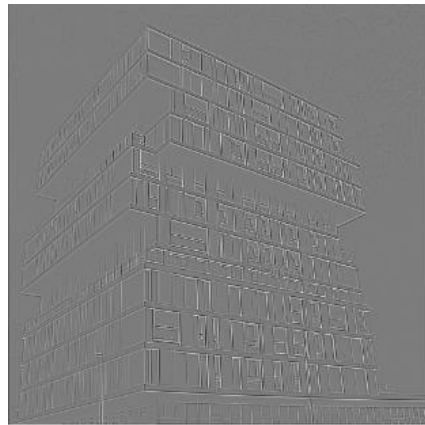
$$\begin{bmatrix} [-1, 0, 1] \\ [-2, 0, 2] \\ [-1, 0, 1] \end{bmatrix}$$



$$\begin{bmatrix} [-1, -2, -1] \\ [0, 0, 0] \\ [1, 2, 1] \end{bmatrix}$$



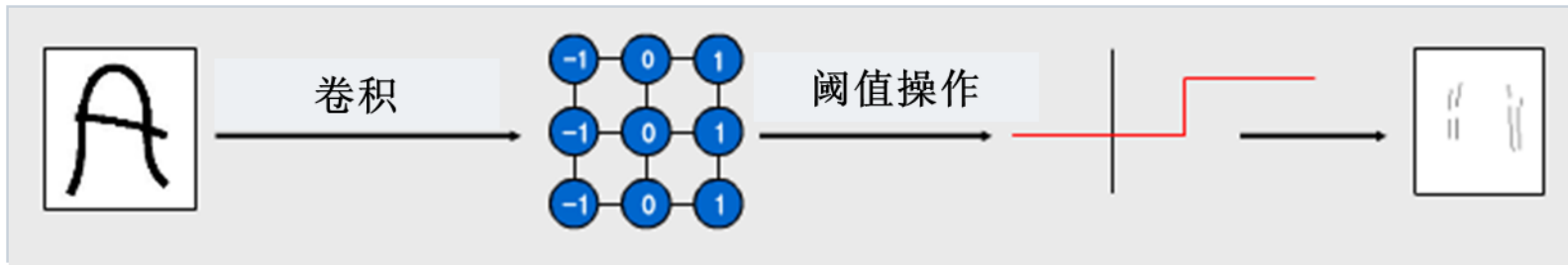
$$\begin{bmatrix} [1, 1, 1] \\ [1, -8, 1] \\ [1, 1, 1] \end{bmatrix}$$



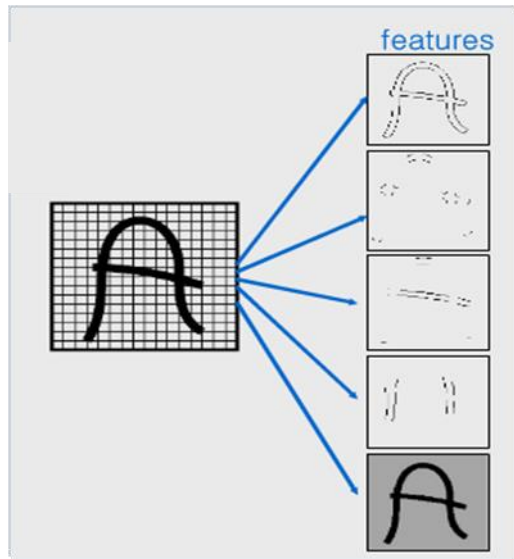
卷积层：卷积运算的主要目的是使原信号特征增强，并降低噪音。



卷积



- 卷积运算的主要目的是**使原信号特征增强，并降低噪音。**
- 对图像用一个卷积核进行卷积运算，实际上是一个滤波的过程。每个卷积核都是一种特征提取方式，就像是一个**筛子**，将图像中符合条件的部分筛选出来。





0填充 (Padding)



观察卷积示例，我们会发现一个**现象**：在卷积核滑动的过程中图像的边缘会被裁剪掉，将 5×5 特征矩阵转换为 3×3 的特征矩阵。

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

12.0		



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

12.0	12.0	



12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0



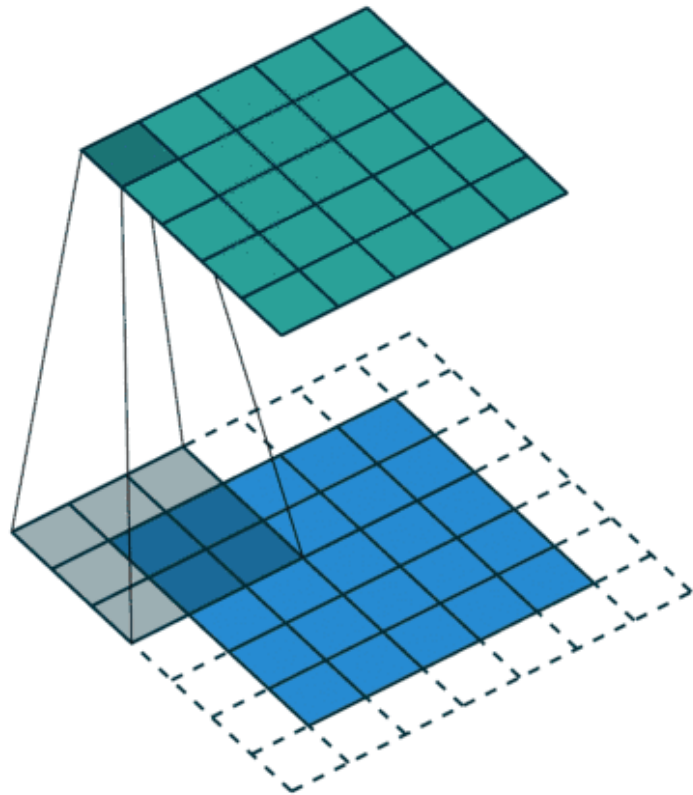
0填充 (Padding)



如何使得输出尺寸与输入保持一致呢？

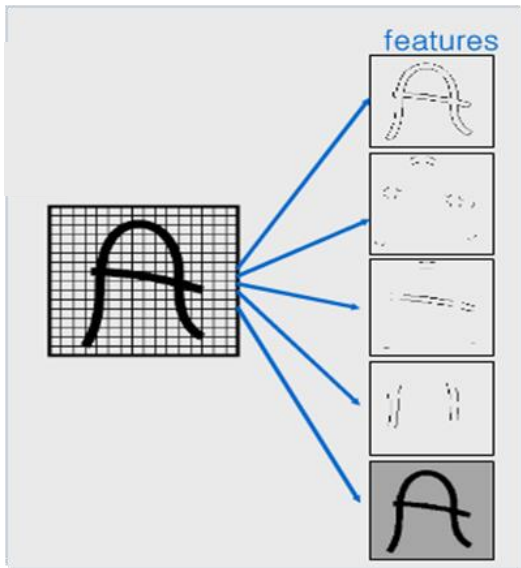
0填充：

用额外的“假”像素（通常值为0）填充边缘。这样，在滑动时的卷积核可以允许原始边缘像素位于卷积核的中心，同时延伸到边缘之外的假像素，从而产生与输入（ 5×5 蓝色）相同大小的输出（ 5×5 绿色）。





多通道卷积

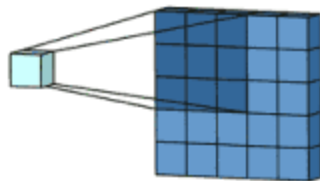
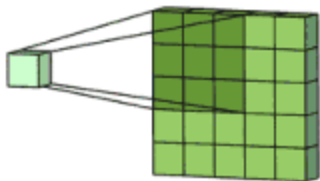
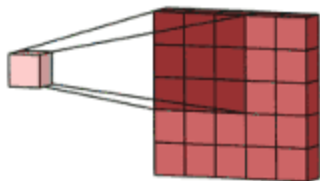


- 每个卷积核都会将图像生成
为另一幅特征映射图，即：
一个卷积核提取一种特征。
- 为了使特征提取更充分，可
以添加多个卷积核以提取不
同的特征，也就是，多通道
卷积。



多通道卷积

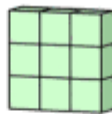
- 每个通道使用一个卷积核进行卷积操作，
- 然后将这些特征图相同位置上的值相加，生成一张特征图。





多通道卷积

- 每个通道使用一个卷积核进行卷积操作，
- 然后将这些特征图相同位置上的值相加，生成一张特征图。



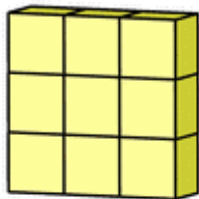


多通道卷积



浙江大學城市學院
ZHEJIANG UNIVERSITY CITY COLLEGE

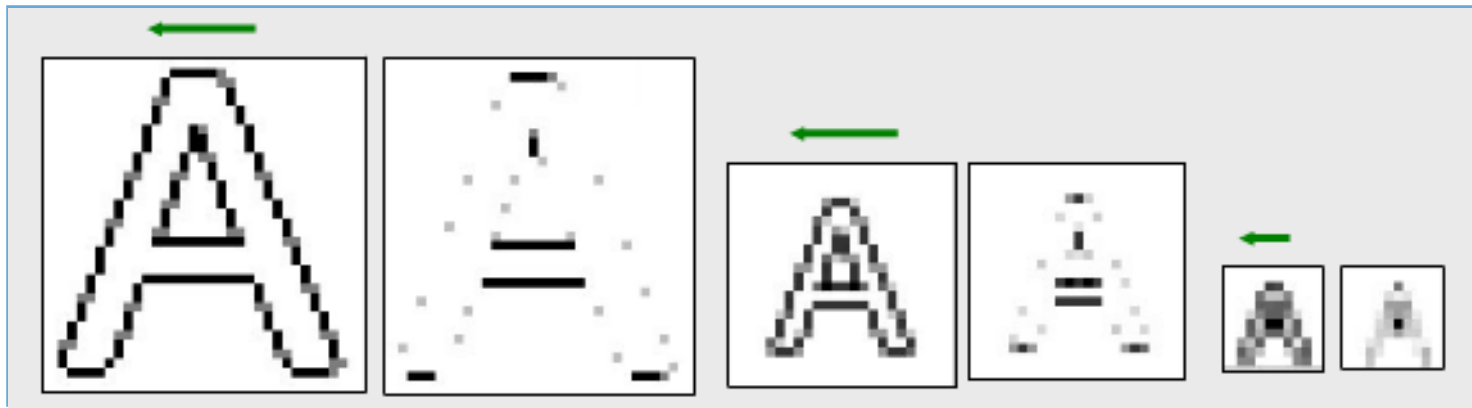
- **加偏置**。偏置的作用是对每个feature map加一个偏置项以便产生最终的输出特征图。





池化 (pooling)

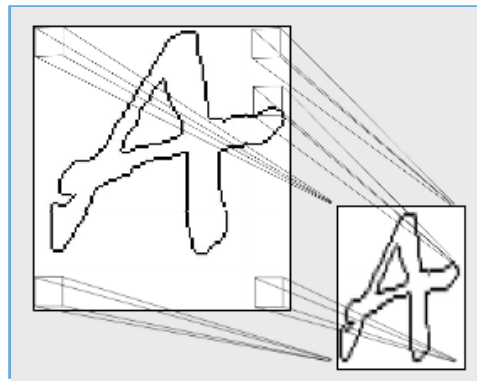
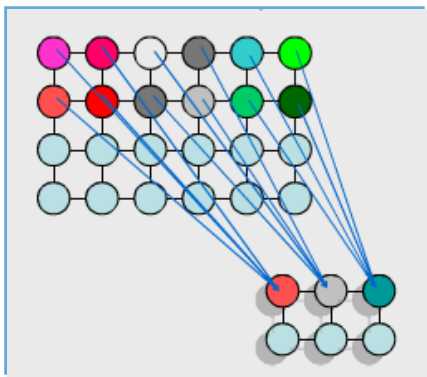
- 在卷积层之后常常紧接着一个降采样层，通过减小矩
阵的长和宽，从而达到减少参数的目的。
- 降采样是降低特定信号的采样率的过程。





池化 (pooling)

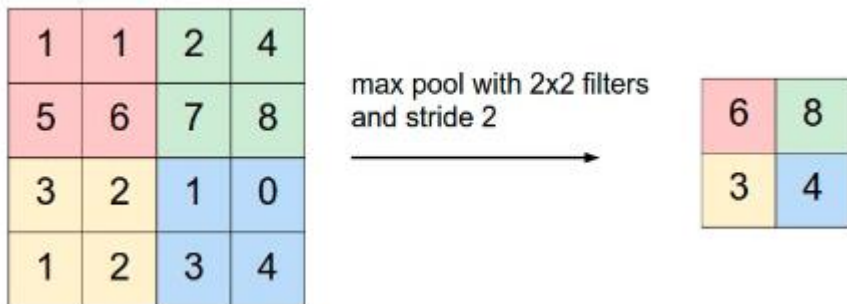
- 计算图像一个区域上的某个特定特征的平均值或最大值，这种聚合操作就叫做**池化 (pooling)**。
- 卷积层的作用是探测上一层特征的局部连接，而池化的作用是在**语义上把相似的特征合并起来**，从而达到降维目的。





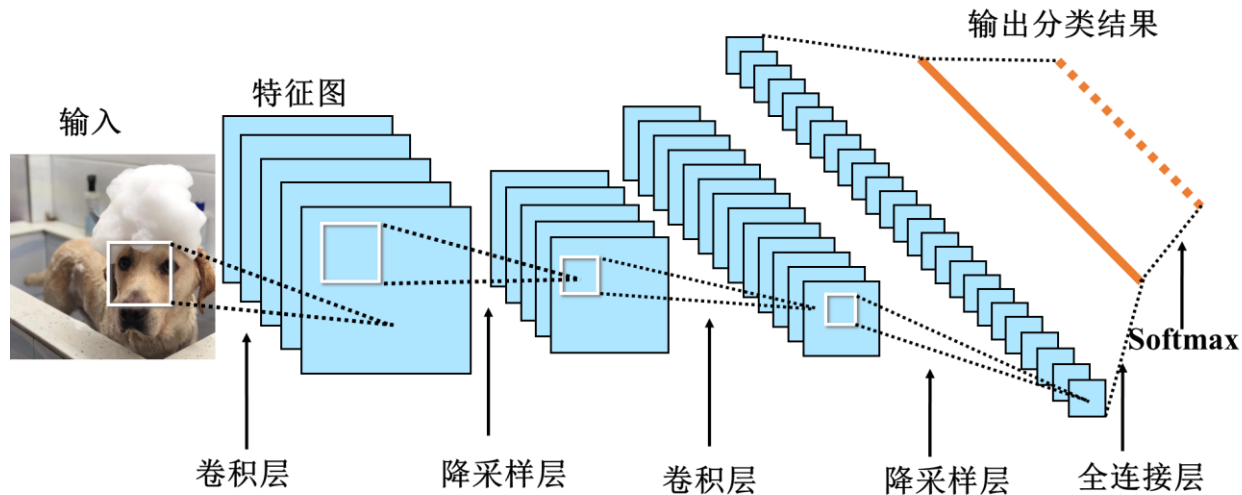
池化 (pooling)

- 这些概要统计特征不仅具有低得多的维度 (相比使用所有提取得到的特征)，同时还会改善结果(不容易过拟合)。
- 常用的池化方法：
 - (1) **均值池化**：对池化区域内的像素点取均值，这种方法得到的特征数据对背景信息更敏感。
 - (2) **最大池化**：对池化区域内所有像素点取最大值，这种方法得到的特征对纹理特征信息更加敏感。





池化 (pooling)



隐层与隐层之间空间分辨率递减，因此，为了检测更多的特征信息、形成更多不同通道特征的组合，从而形成更复杂的特征，需要逐渐增加每层所含的平面数（也就是特征图的数量）

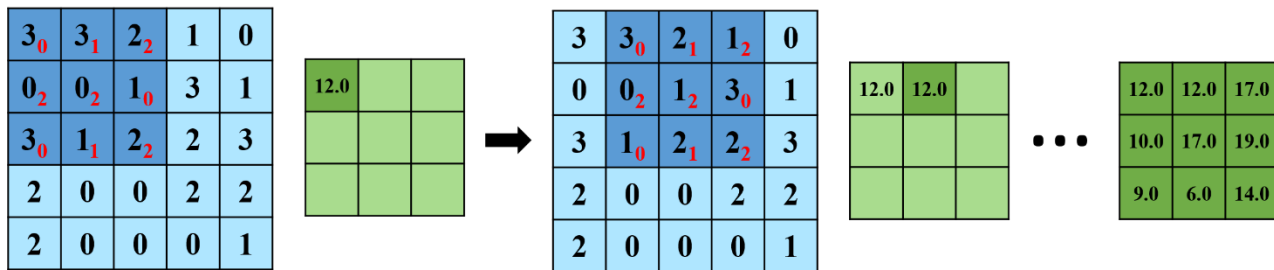


步长 (stride)

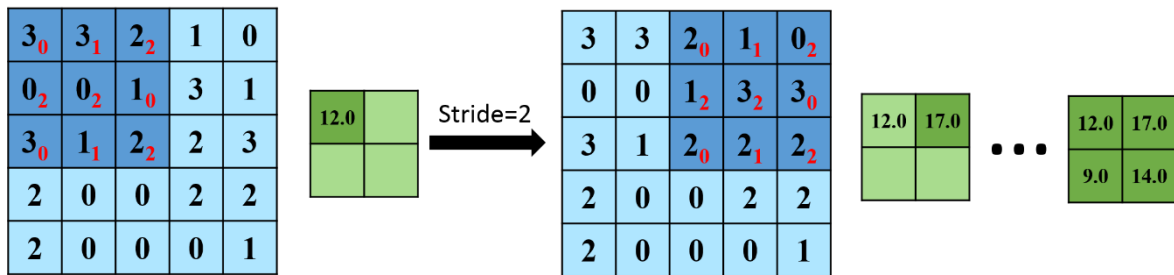


- 步长是卷积操作的重要概念，表示卷积核在图片上移动的格数。
- 通过步长的变换，可以得到不同尺寸的卷积输出结果。

● 当stride=1时

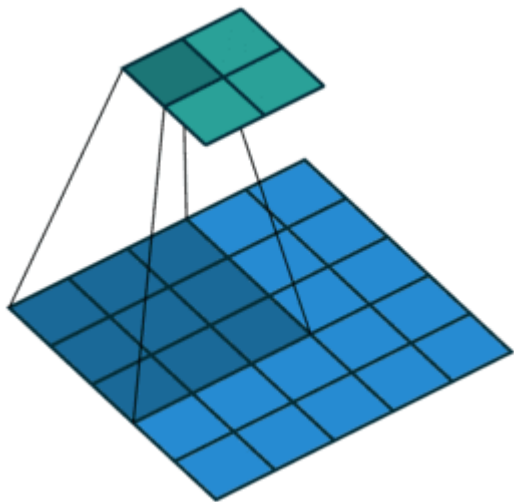


● 当stride=2时





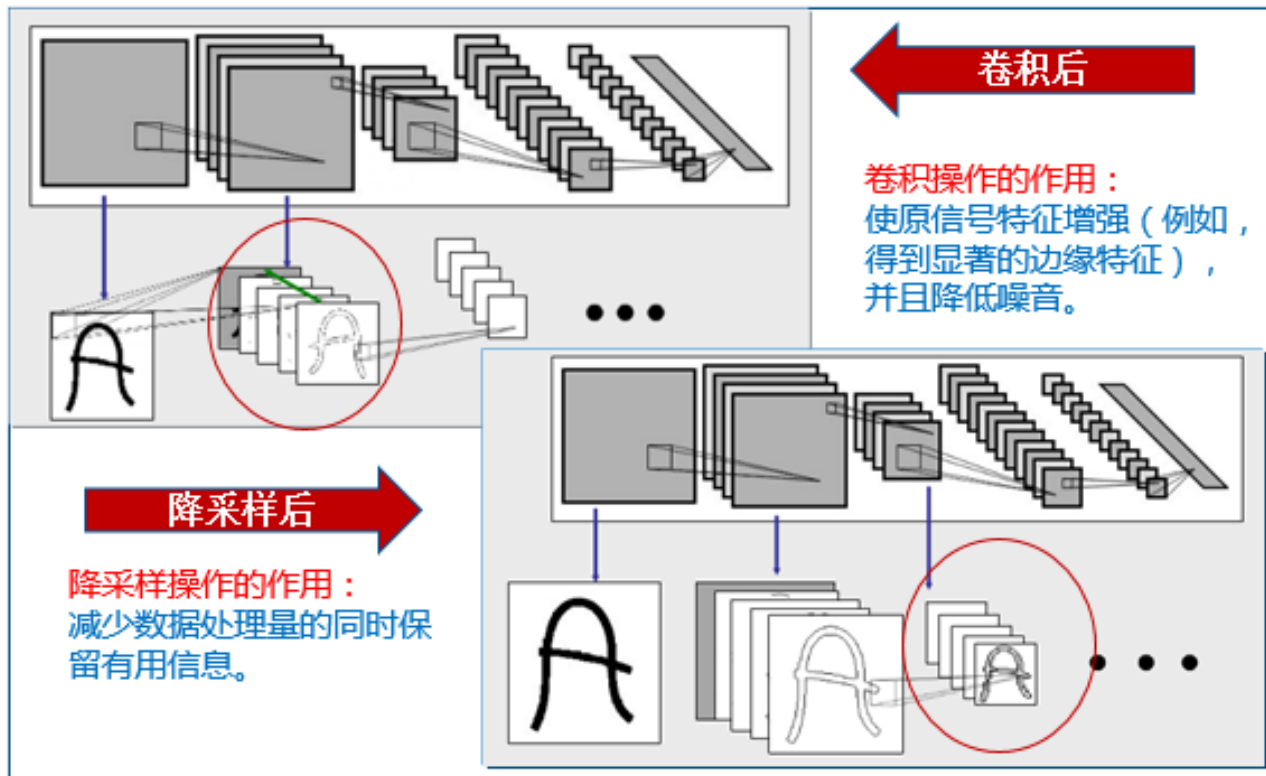
步长 (stride)



- 步长大于1的卷积操作也是**降维的一种方式**
- 卷积后图片尺寸：假如步长为 S ，原始图片尺寸为 $[N1, N1]$ ，卷积核大小为 $[N2, N2]$ ，那么**卷积之后图像大小**： $[(N1-N2)/S+1, (N1-N2)/S+1]$

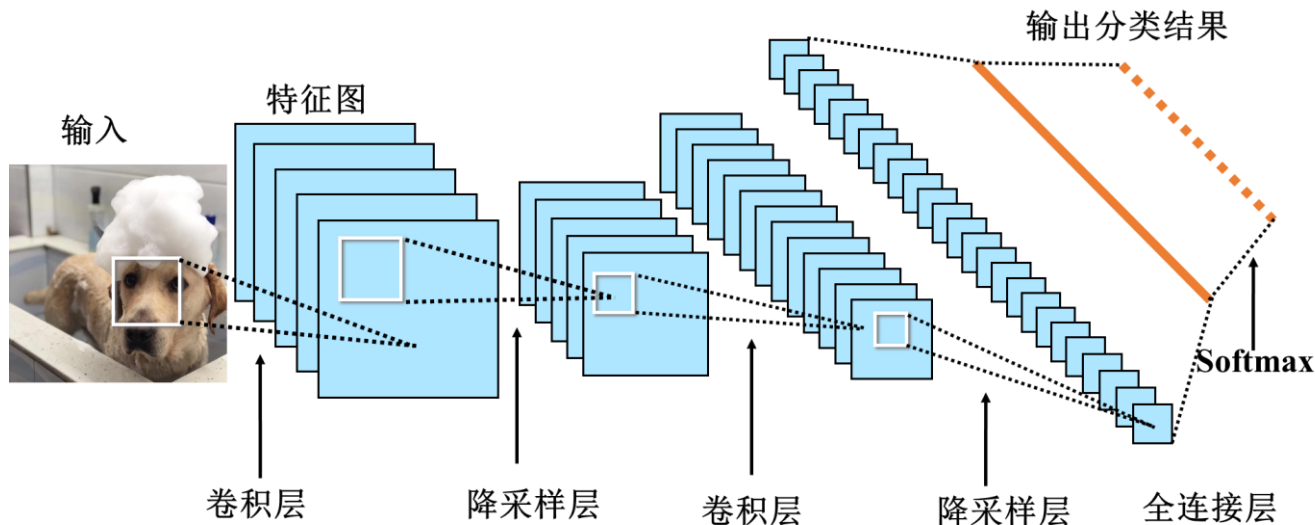


卷积神经网络





卷积神经网络



过程概括：输入图像通过若干个“卷积→降采样”后，连接成一个向量输入到传统的分类器层中，最终得到输出。

正则表达式：

输入层→（卷积层+→池化层？）+→全连接层+