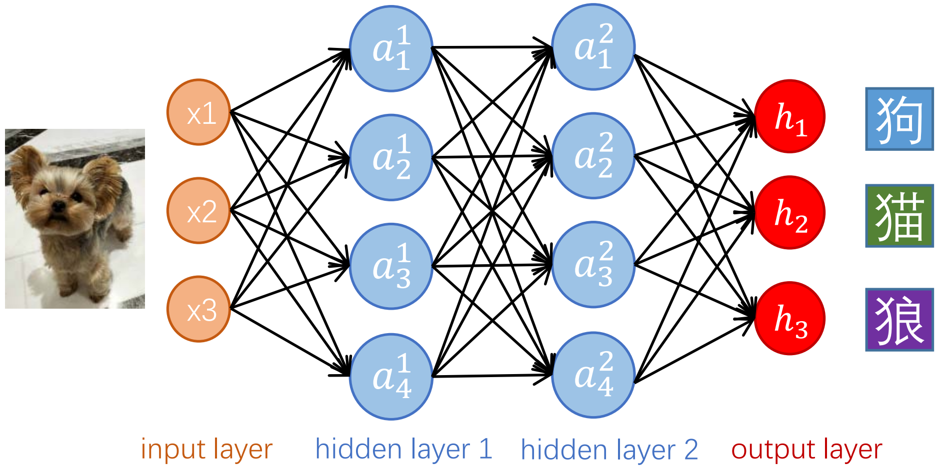
计算机视觉
Computer Vision

Lecture 5: 卷积神经网络

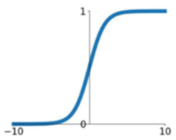
L04：神经网络和反向传播



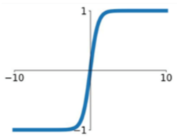
全连接神经网络

激活函数

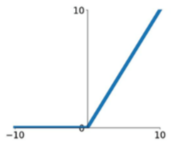
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



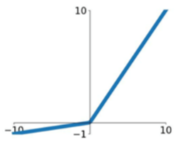
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

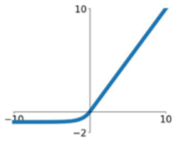


Leaky ReLU
$\max(0.1x, x)$



Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



反向传播

$$\frac{\partial L}{\partial \boldsymbol{W}^l} = \frac{\partial L}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{W}^l}$$

$$\frac{\partial L}{\partial \boldsymbol{W}^{l-1}} = \frac{\partial L}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{a}^l} \frac{\partial \boldsymbol{a}^l}{\partial \boldsymbol{W}^{l-1}}$$

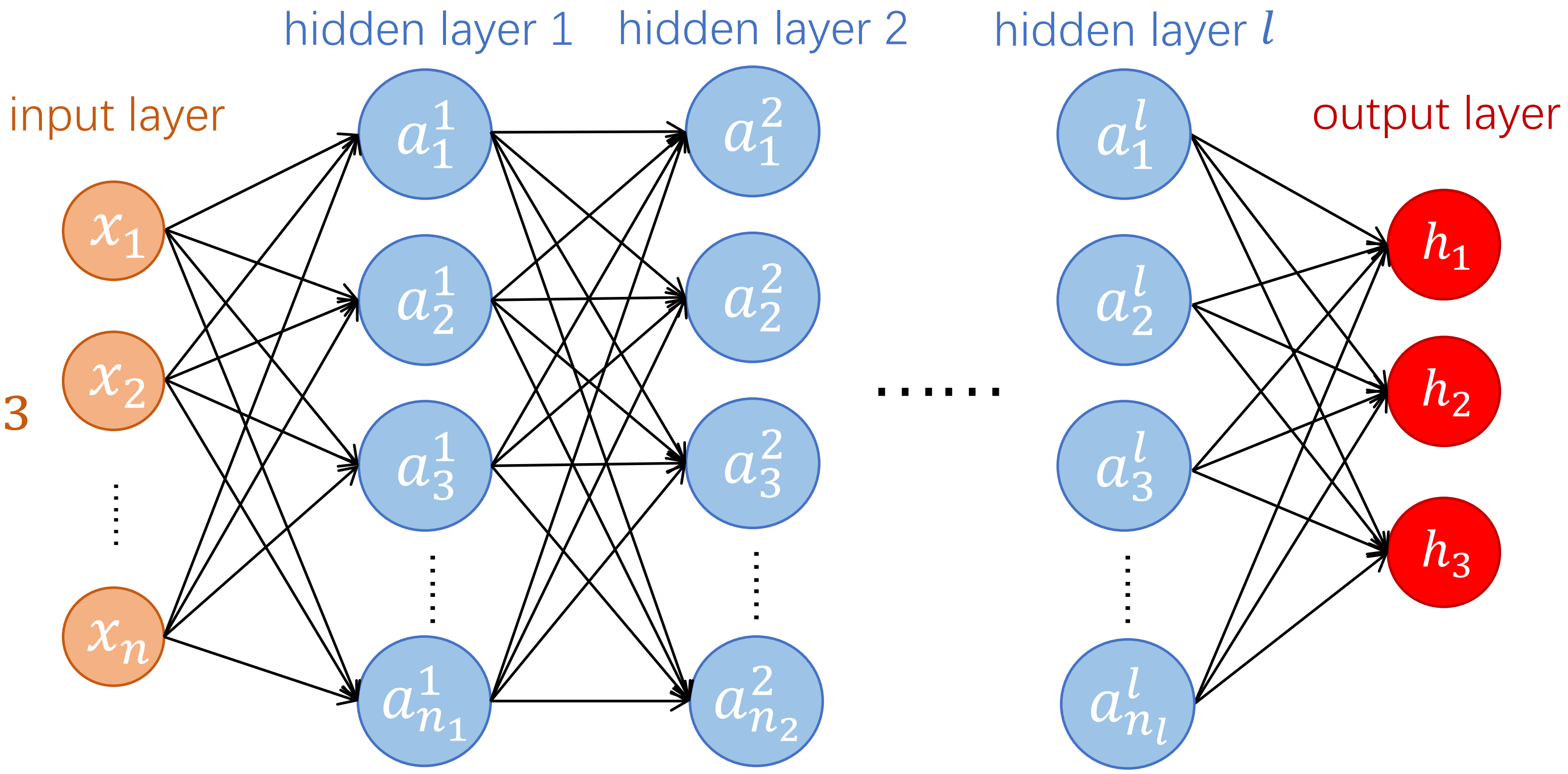
$$\frac{\partial L}{\partial \boldsymbol{W}^{l-2}} = \frac{\partial L}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{a}^l} \frac{\partial \boldsymbol{a}^l}{\partial \boldsymbol{a}^{l-1}} \frac{\partial \boldsymbol{a}^{l-1}}{\partial \boldsymbol{W}^{l-2}}$$

$$\vdots$$

Chain rule

$$\frac{\partial L}{\partial \boldsymbol{W}^1} = \frac{\partial L}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{a}^l} \frac{\partial \boldsymbol{a}^l}{\partial \boldsymbol{a}^{l-1}} \frac{\partial \boldsymbol{a}^{l-1}}{\partial \boldsymbol{a}^{l-2}} \cdots\cdots \frac{\partial \boldsymbol{a}^2}{\partial \boldsymbol{W}^1}$$

全连接神经网络（FCNN）的问题

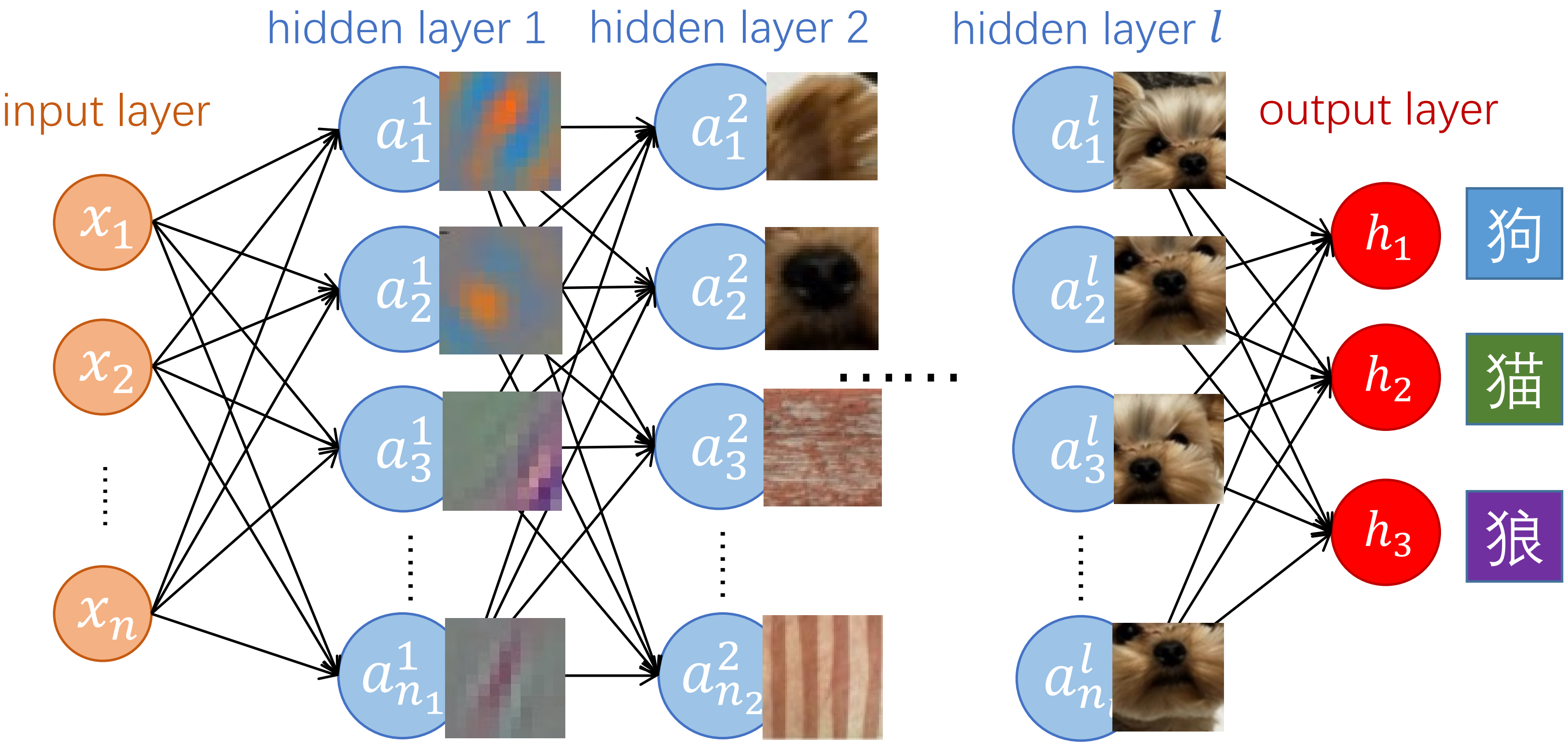


$x \in \mathbb{R}^{100\times100\times3}$

$\boldsymbol{a}^{1} \in \mathbb{R}^{4096}$

第一层参数个数： $30000 \times 4096 \approx 1.2$ 亿！！

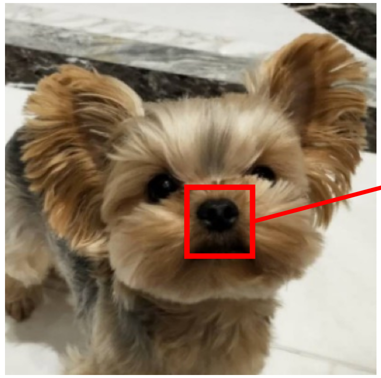
观察FCNN



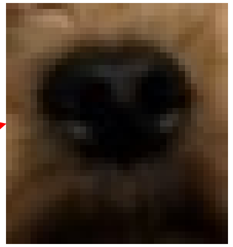
- ✓ 每个neural只关注某类比原图小很多的局部pattern
- ✓ 相似的局部pattern会出现在图片的不同部分
- ✓ 对像素做采样之后，图像的内容不会发生改变

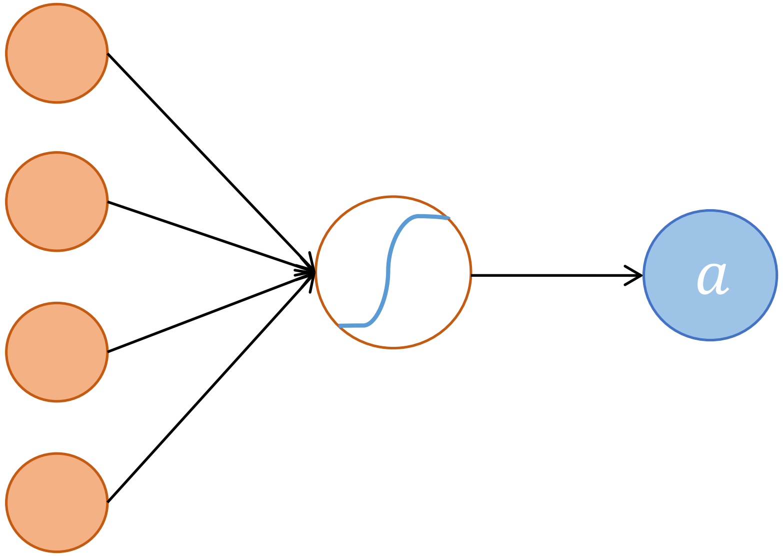
局部pattern

$100 \times 100 \times 3$



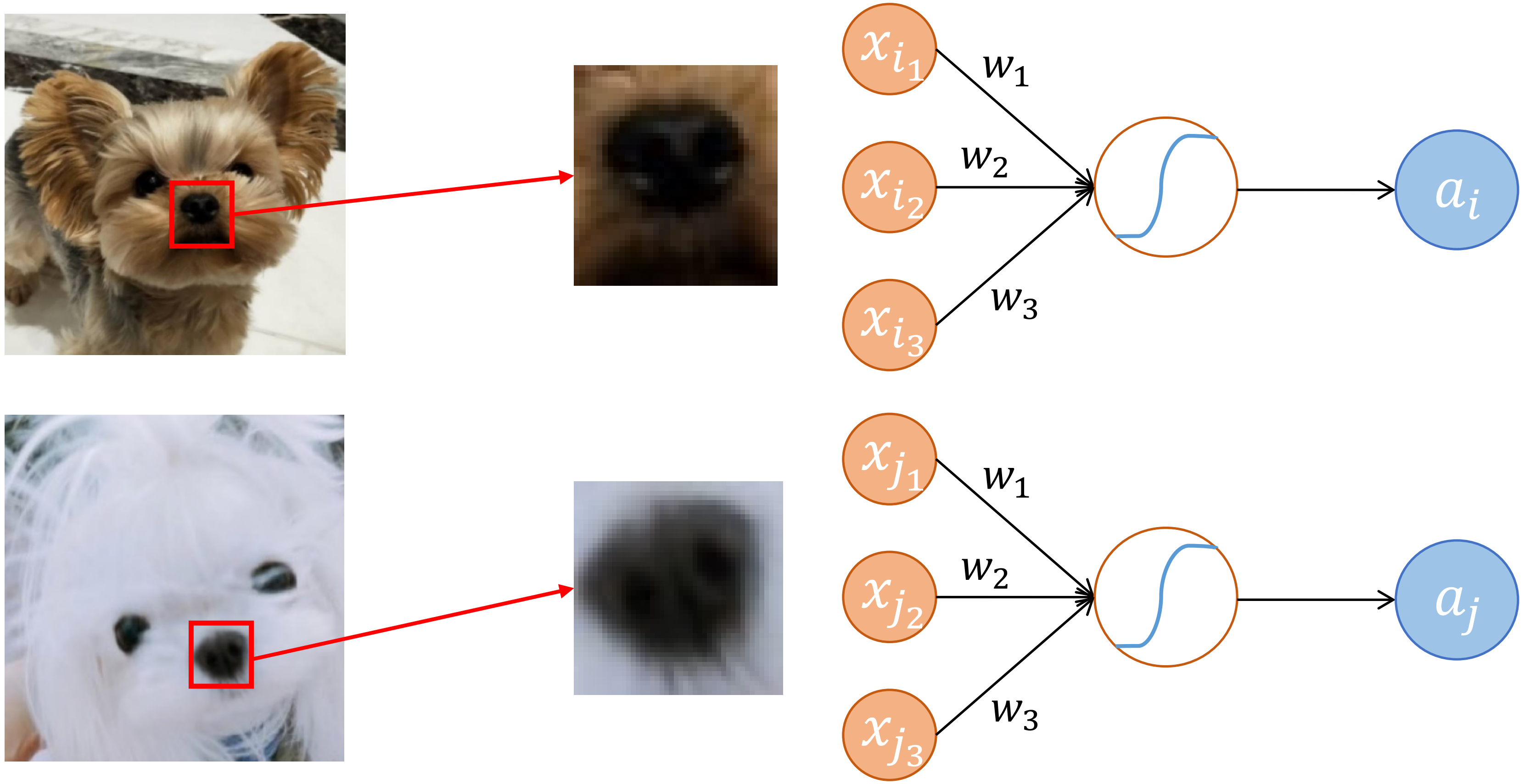
$5 \times 5 \times 3$





每个神经元捕获局部pattern，减少权重参数

相似pattern，不同区域



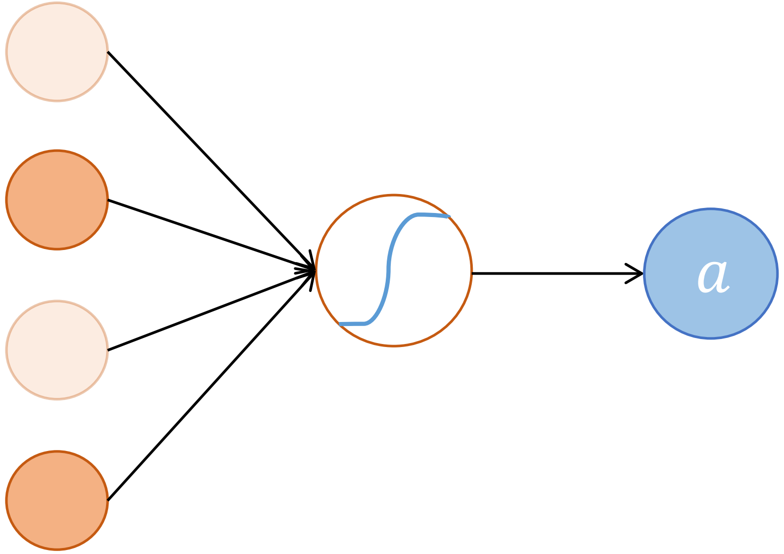
功能相似的神经元可以共享权重参数

像素采样

$100 \times 100 \times 3$

downsampling

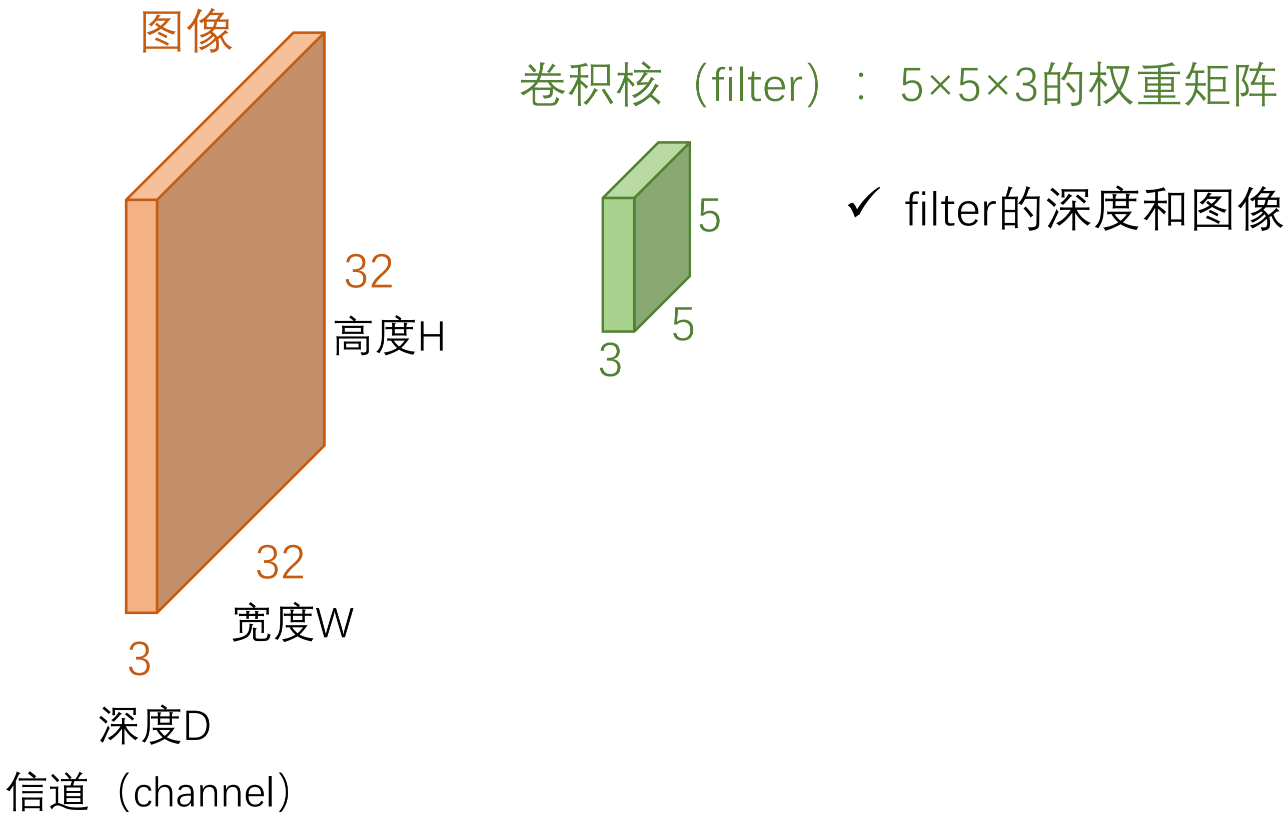
$50 \times 50 \times 3$



进一步减少参数数量

卷积神经网络

- Convolutional Neural Networks



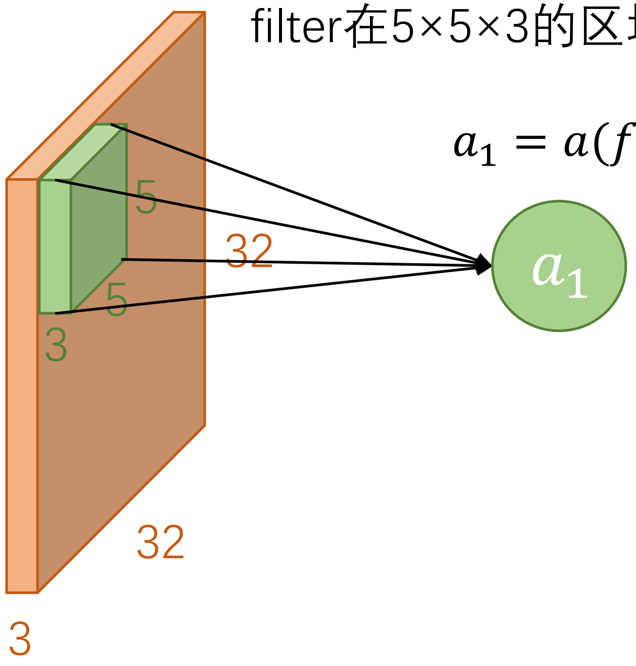
✓ filter的深度和图像的深度一样

卷积神经网络

卷积

filter在5×5×3的区域内做点积，然后激活（一般用ReLU）

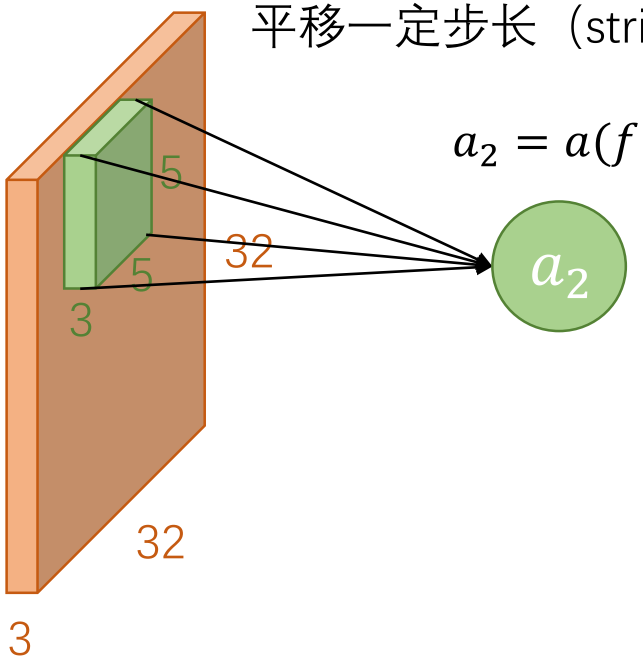
$$a_1 = a(f(\boldsymbol{w_1}^T \boldsymbol{x_1} + b))$$



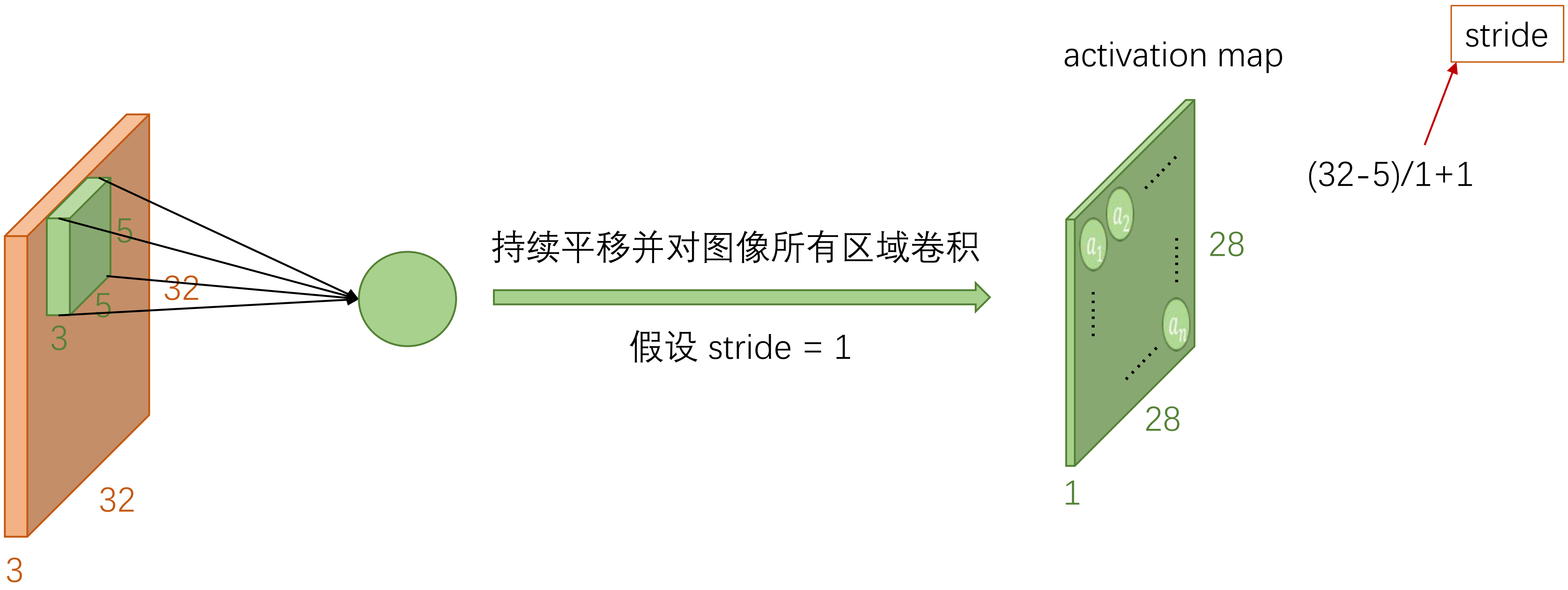
卷积神经网络

平移一定步长（stride），在相邻区域内做卷积

$$a_2 = a(f(\boldsymbol{w_1}^T \boldsymbol{x_2} + b))$$

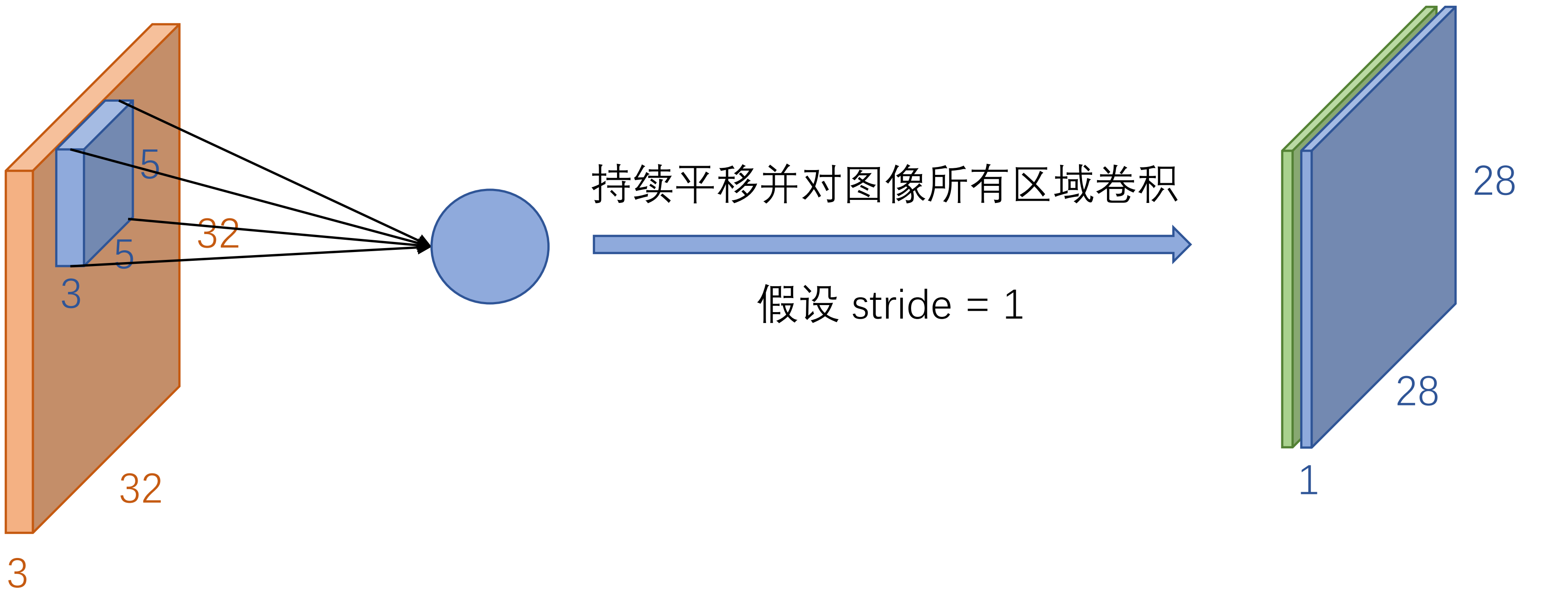


卷积神经网络

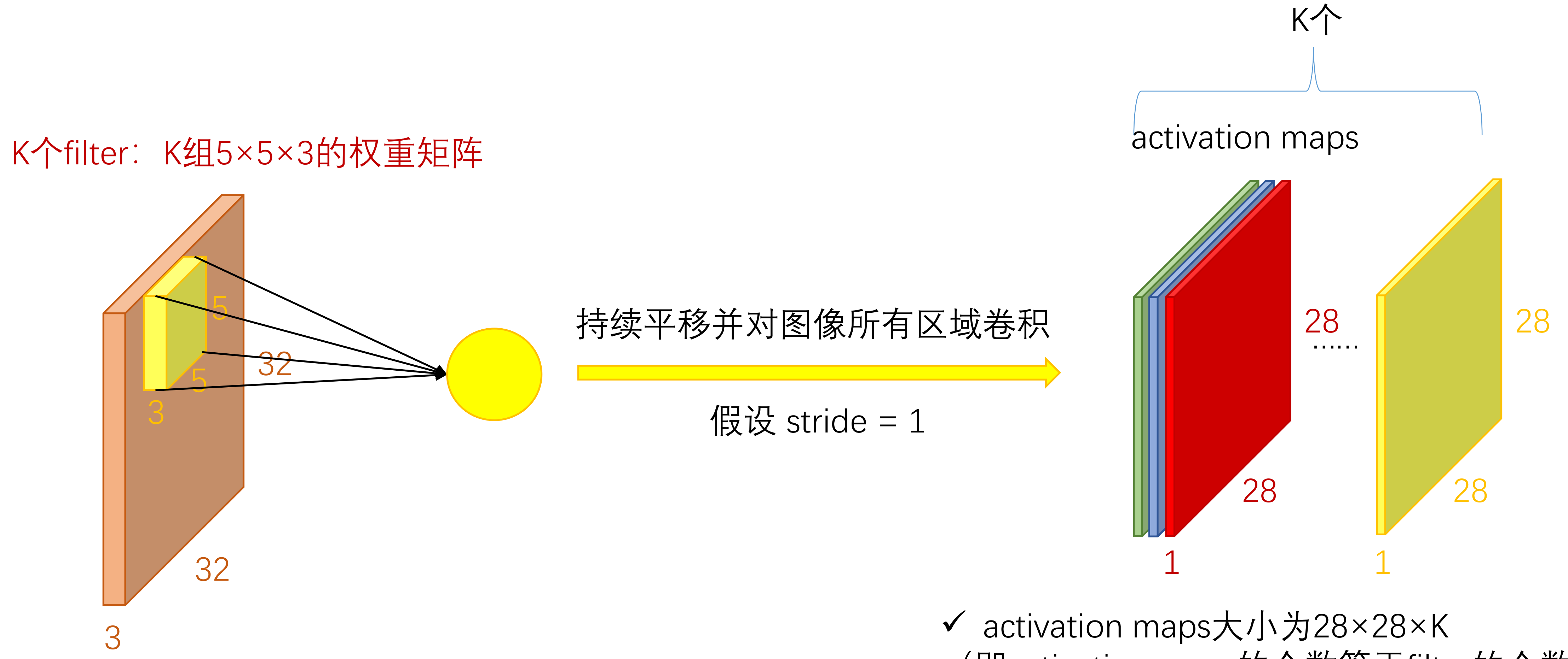

activation map
stride
(32-5)/1+1
5
32
5
3
32
3
持续平移并对图像所有区域卷积
假设 stride = 1
28
28
1

卷积神经网络

第二个filter：第二组5×5×3的权重矩阵



卷积神经网络



✓ activation maps大小为28×28×K
（即activation maps的个数等于filter的个数）

卷积层

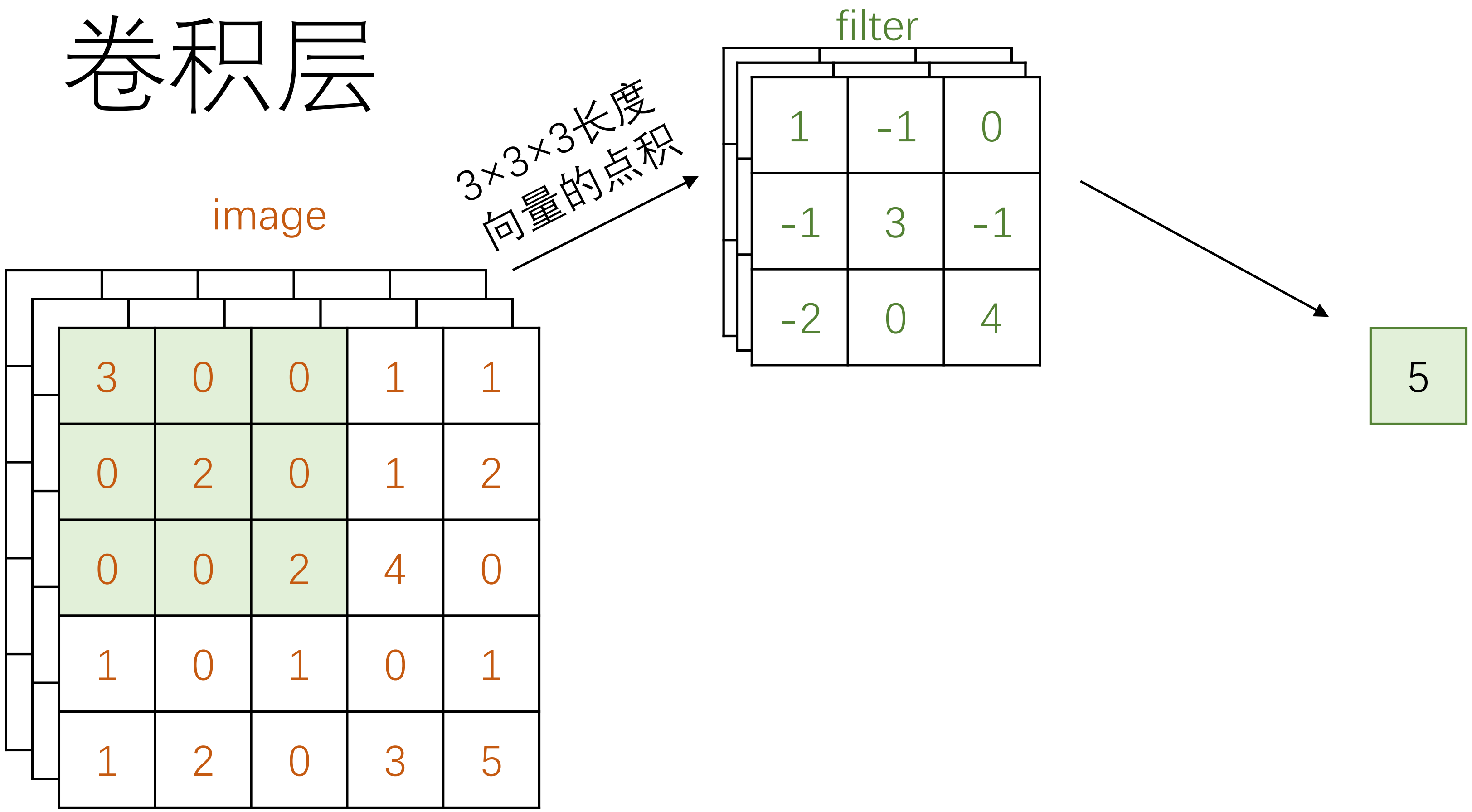
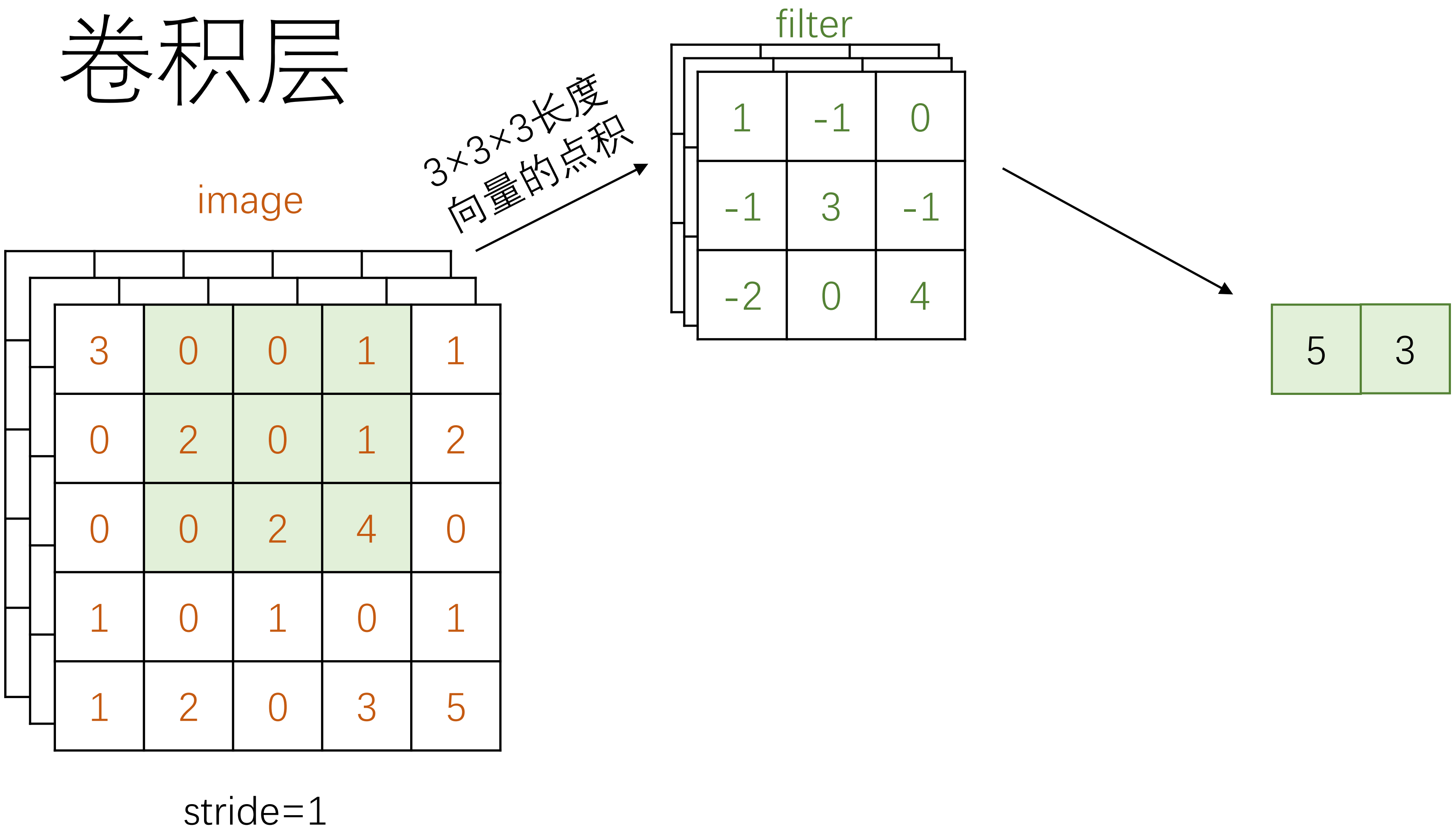


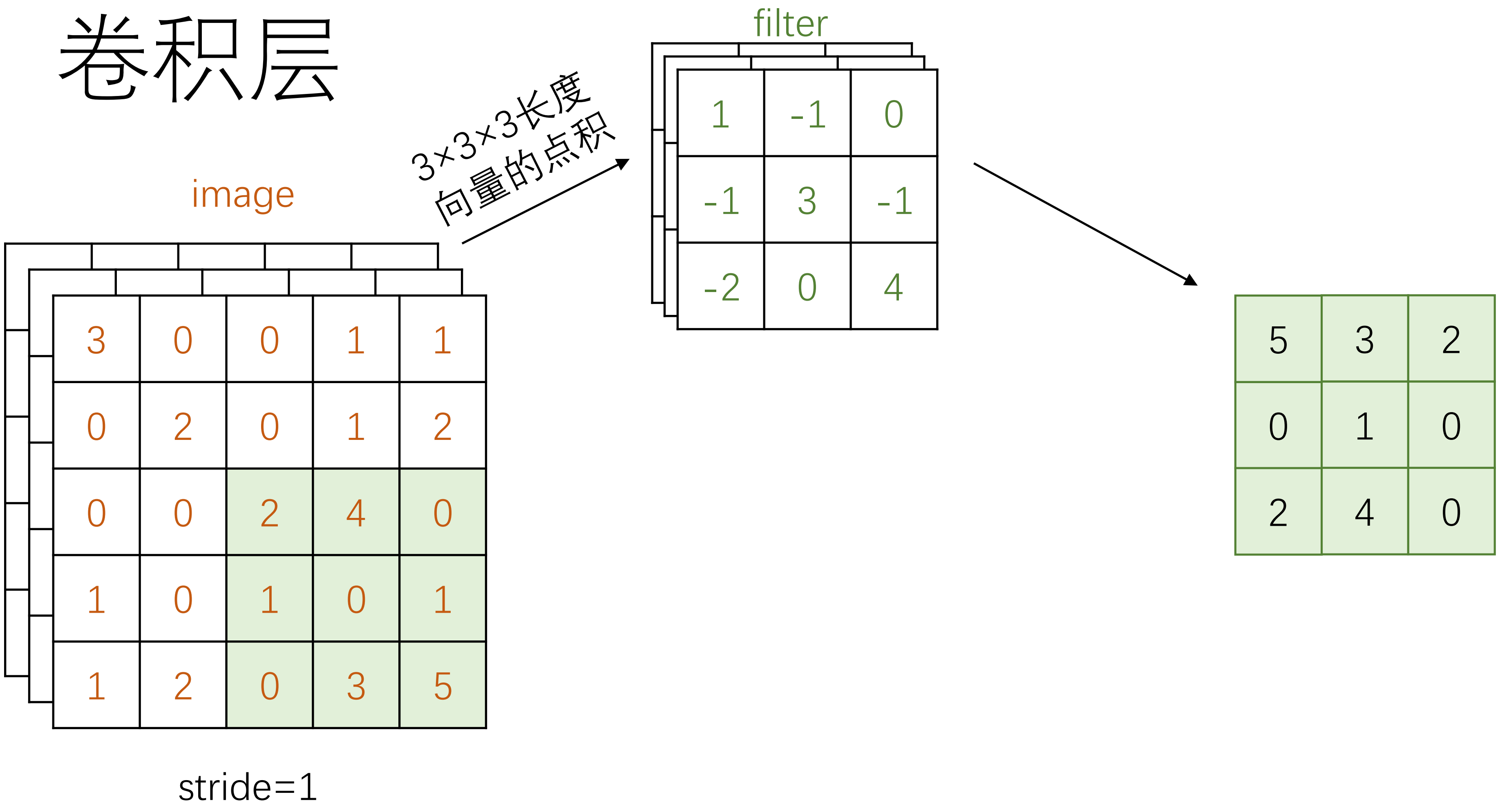
image
3 0 0 1 1
0 2 0 1 2
0 0 2 4 0
1 0 1 0 1
1 2 0 3 5
3×3×3长度
向量的点积
filter
1 -1 0
-1 3 -1
-2 0 4
5

卷积层

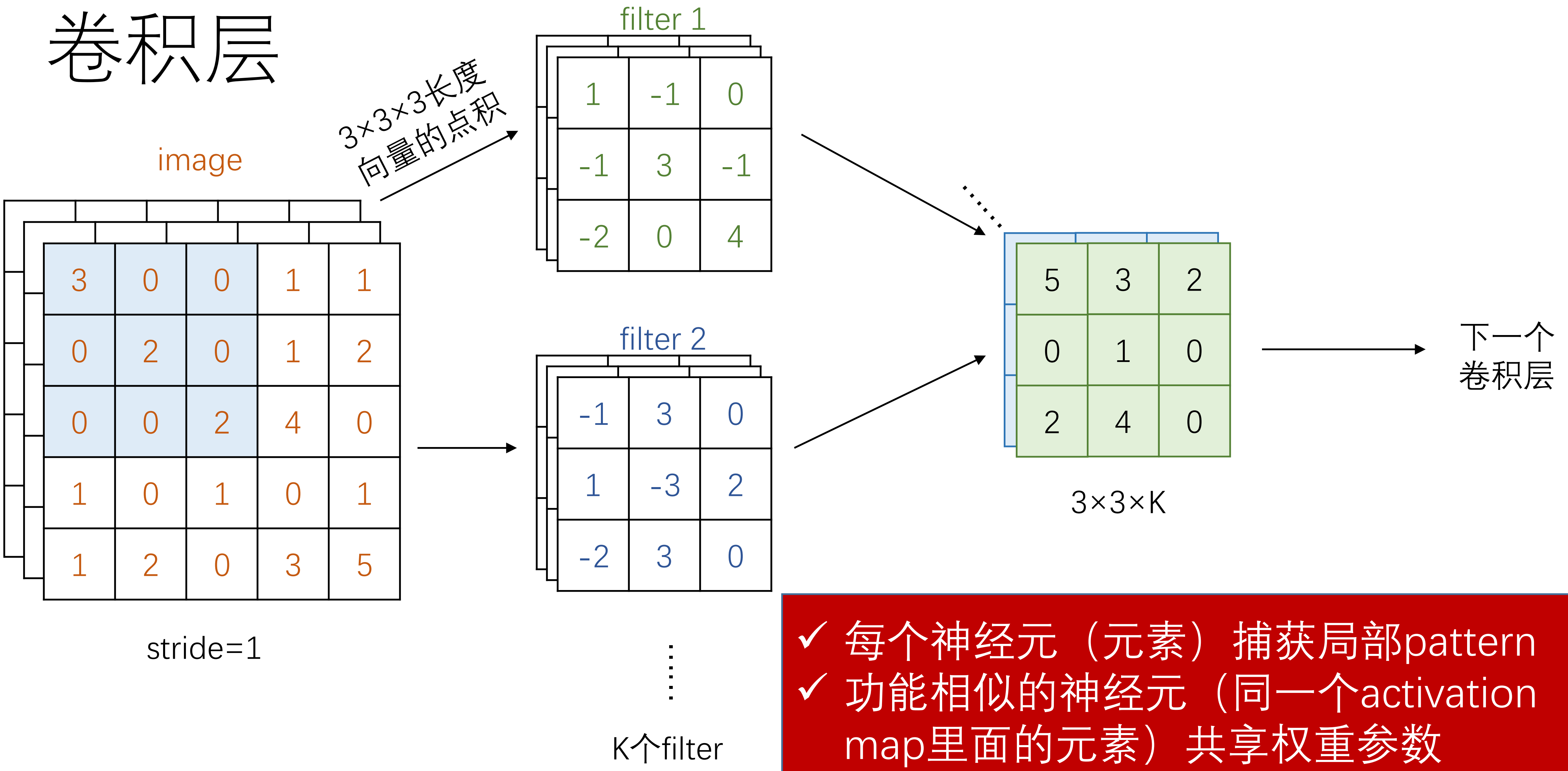


filter
3×3×3长度
向量的点积
image
3 0 0 1 1
0 2 0 1 2
0 0 2 4 0
1 0 1 0 1
1 2 0 3 5
1 -1 0
-1 3 -1
-2 0 4
5 3
stride=1

卷积层


filter
1 -1 0
-1 3 -1
-2 0 4
3×3×3长度
向量的点积
image
3 0 0 1 1
0 2 0 1 2
0 0 2 4 0
1 0 1 0 1
1 2 0 3 5
5 3 2
0 1 0
2 4 0
stride=1

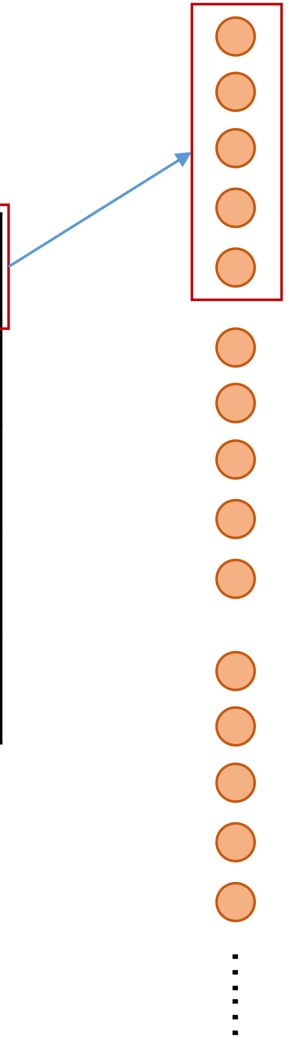
卷积层



- ✓ 每个神经元（元素）捕获局部pattern
- ✓ 功能相似的神经元（同一个activation map里面的元素）共享权重参数

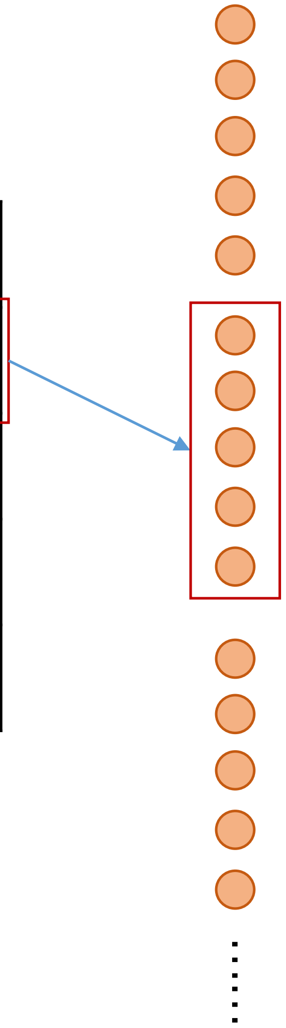
FC层 Vs 卷积层

3	0	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5



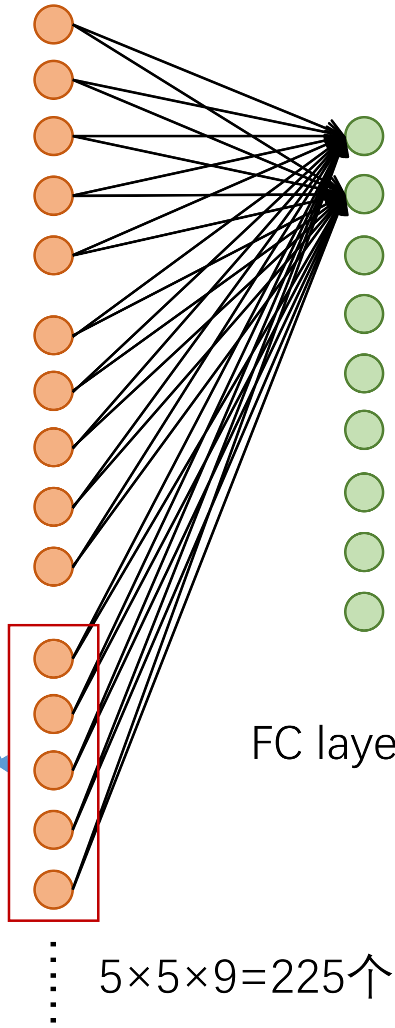
FC层 Vs 卷积层

3	0	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5



FC层 Vs 卷积层

3	0	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5

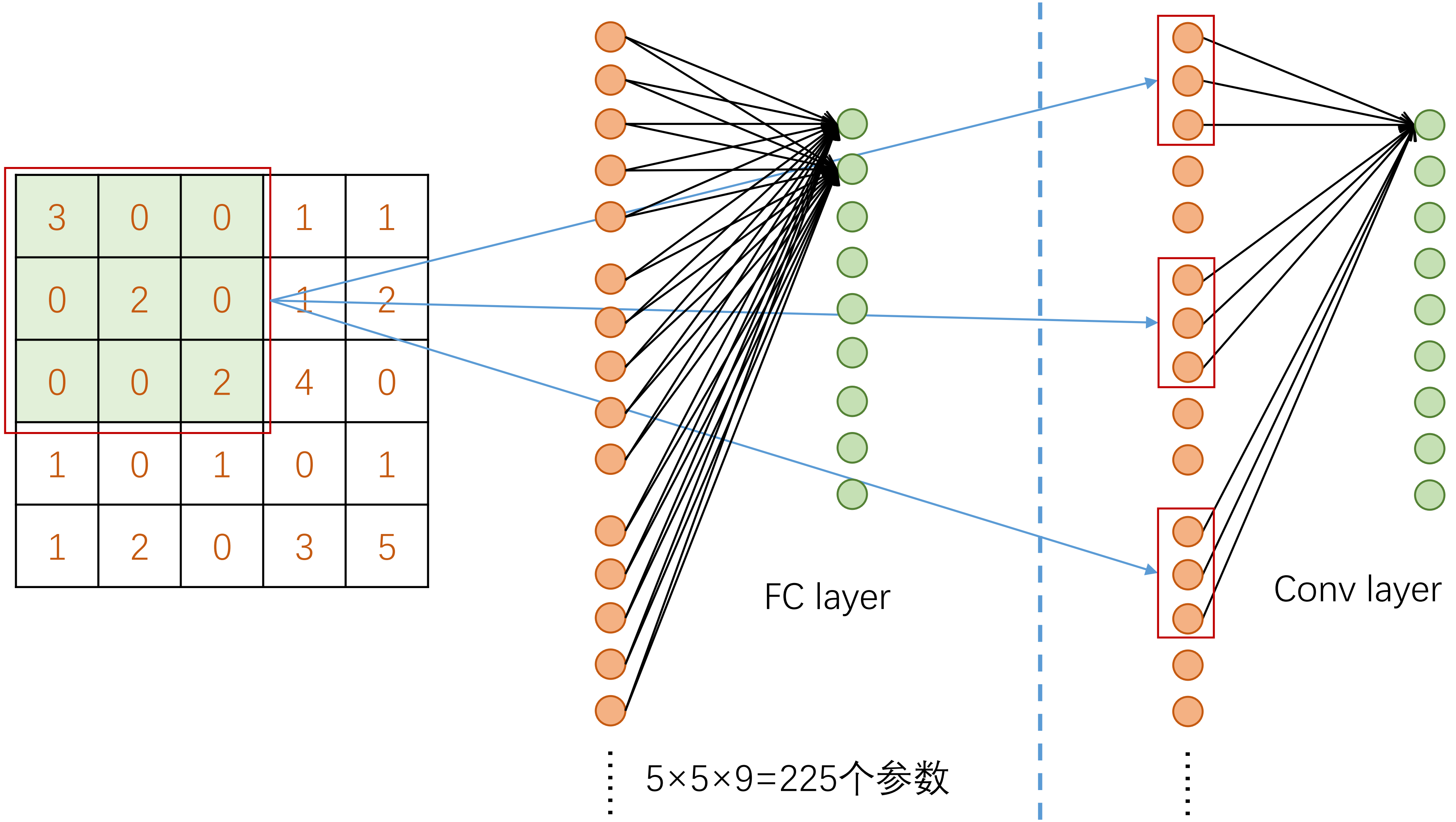


FC layer

$5\times5\times9=225$个参数

FC层 Vs 卷积层

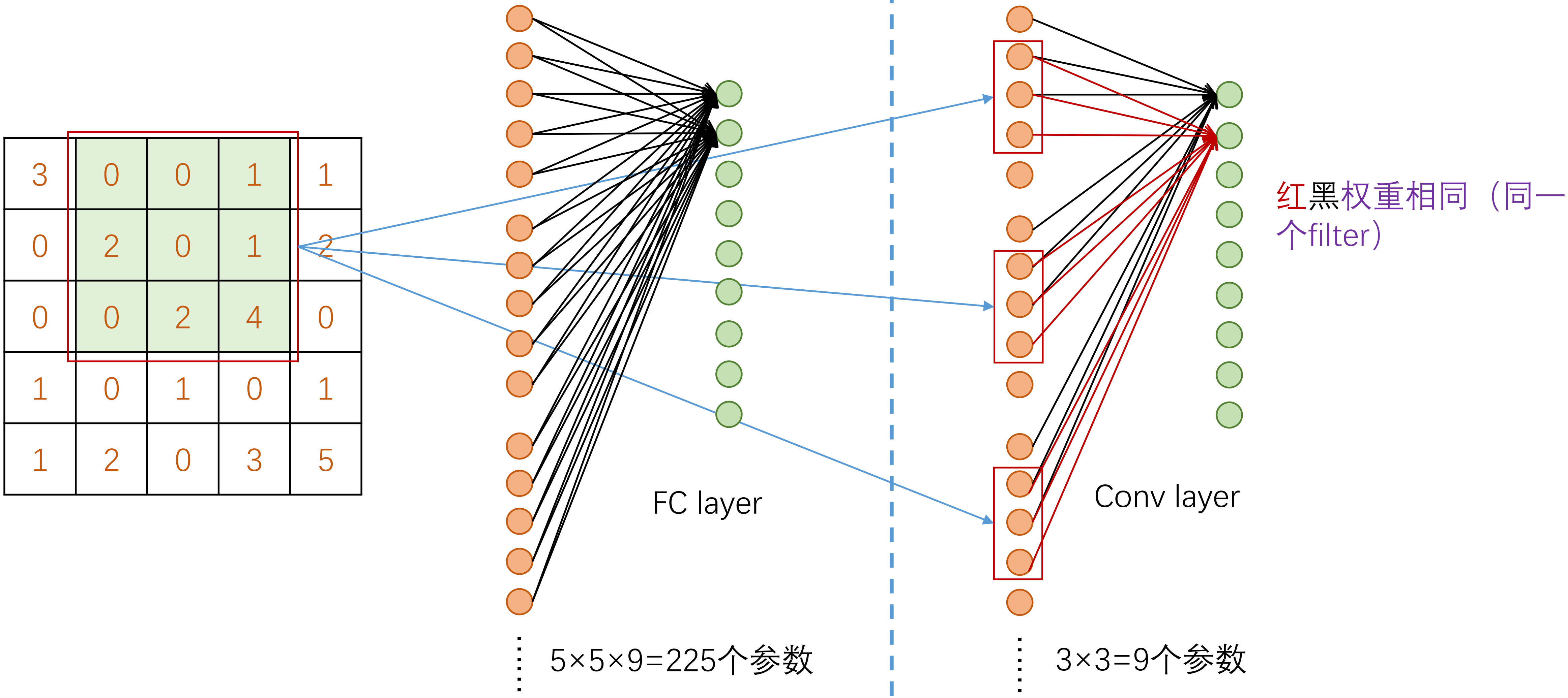
连接局部input→减少参数数量



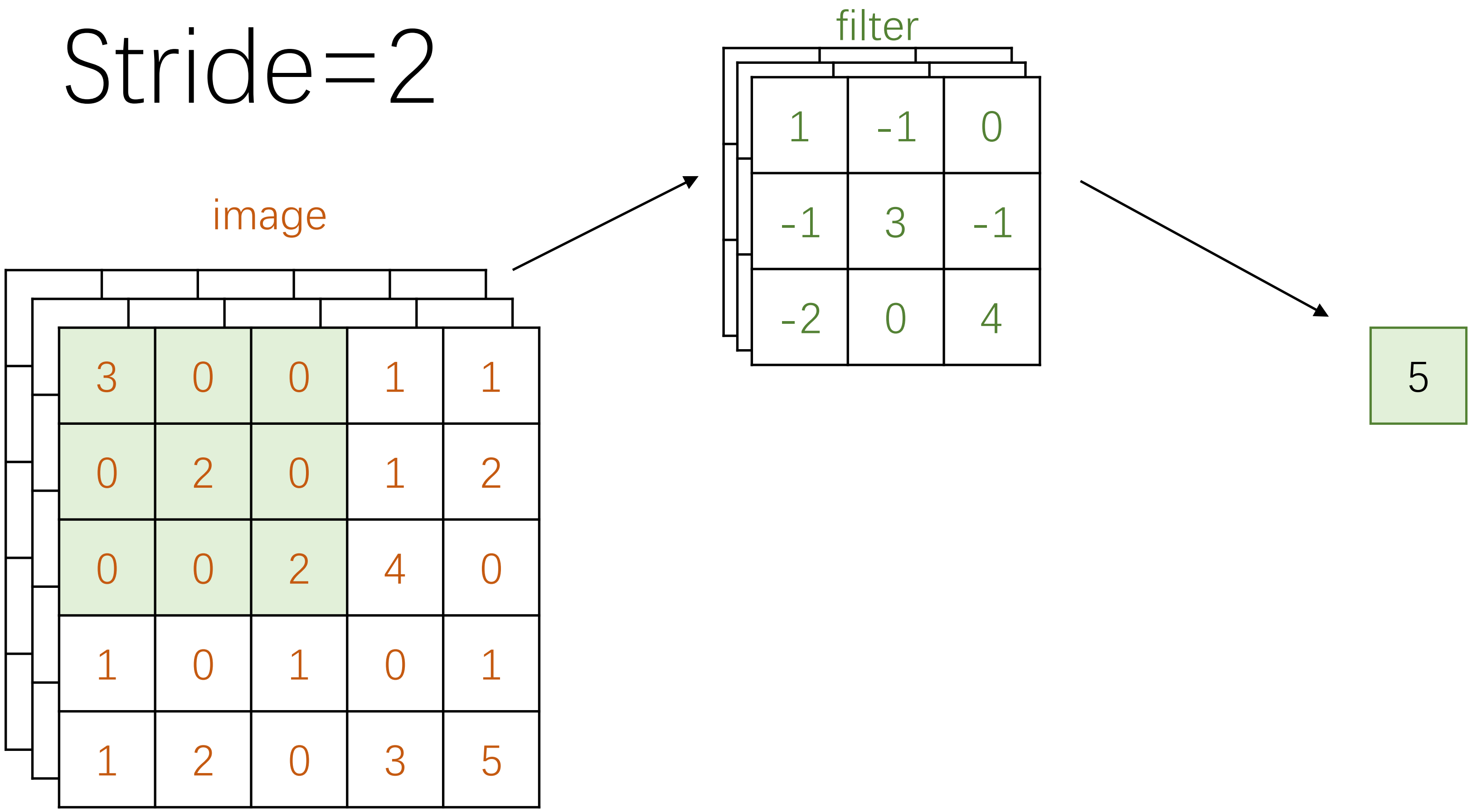
FC层 Vs 卷积层

连接局部input→减少参数数量

权重共享，进一步减少参数数量

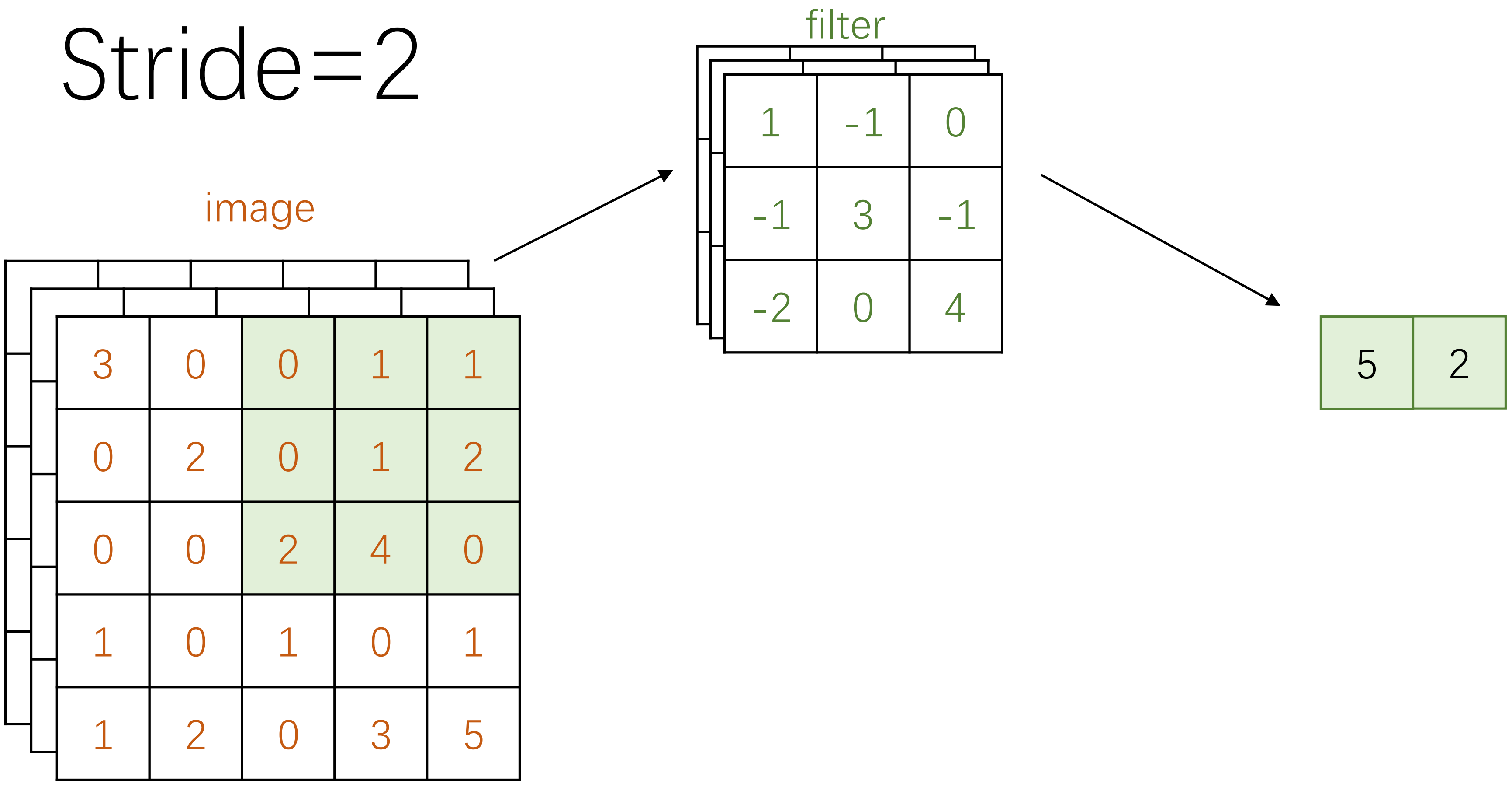


Stride=2



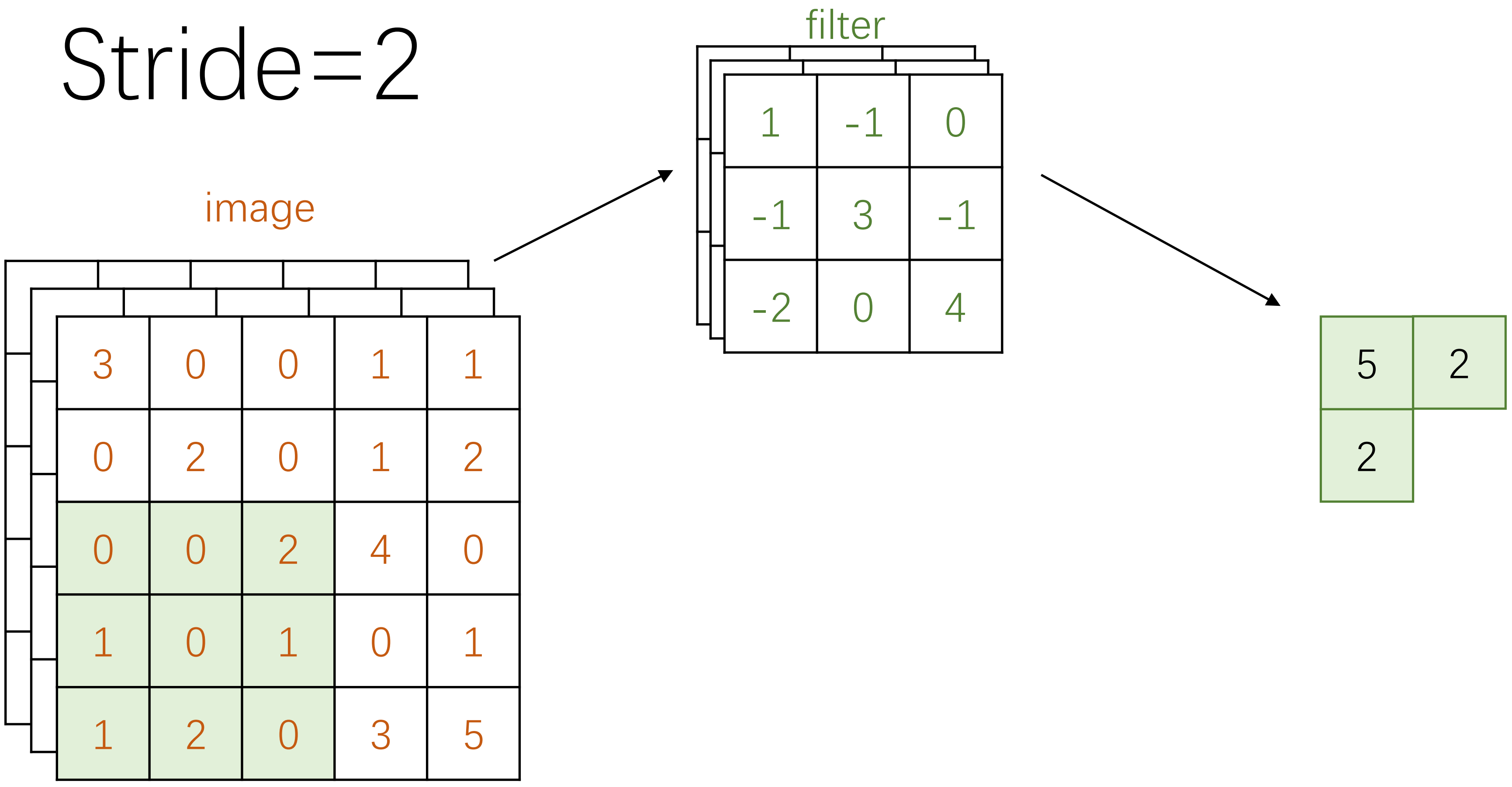
stride=2

Stride=2



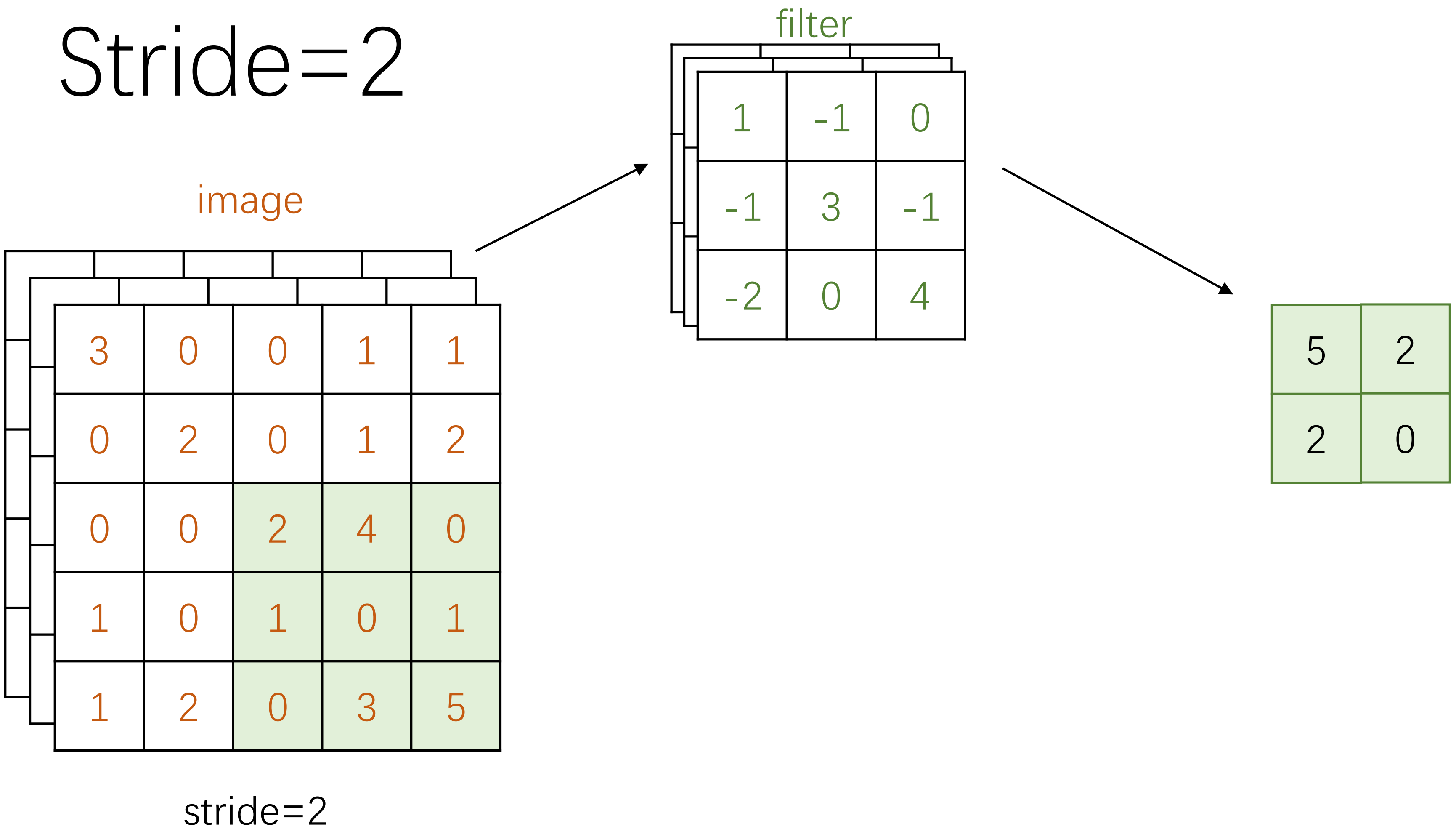
stride=2

Stride=2



stride=2

Stride=2


filter
1 -1 0
-1 3 -1
-2 0 4
image
3 0 0 1 1
0 2 0 1 2
0 0 2 4 0
1 0 1 0 1
1 2 0 3 5
stride=2
5 2
2 0

Stride=2

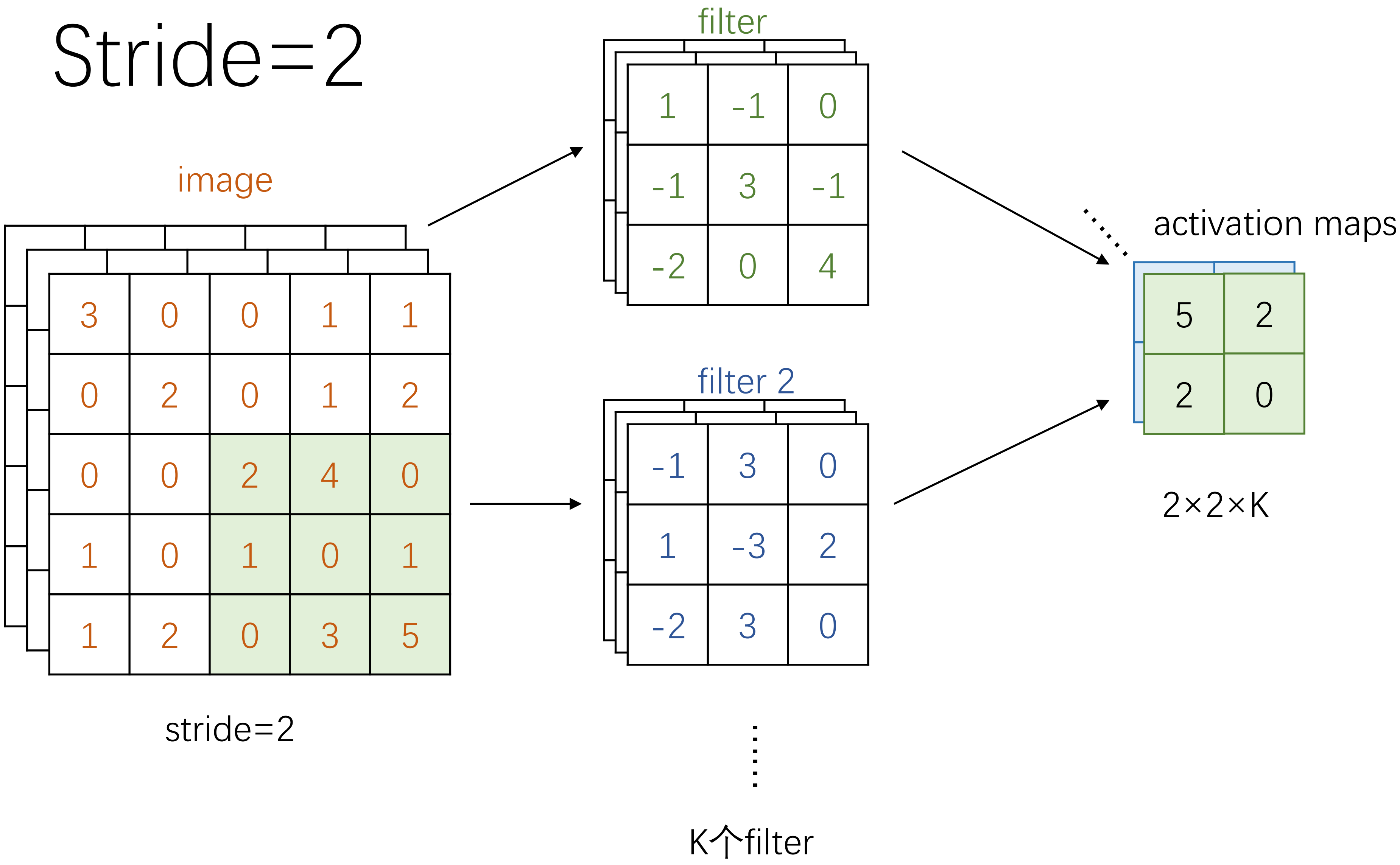


image
3 0 0 1 1
0 2 0 1 2
0 0 2 4 0
1 0 1 0 1
1 2 0 3 5
stride=2
filter
1 -1 0
-1 3 -1
-2 0 4
filter 2
-1 3 0
1 -3 2
-2 3 0
K个filter
activation maps
5 2
2 0
2×2×K

Zero Padding

filter无法捕获图像边缘的pattern

- ✓ 用0对图像边缘进行填充

0	0	0	0	0	0	0
0	3	0	0	1	1	0
0	0	2	0	1	2	0
0	0	0	2	4	0	0
0	1	0	1	0	1	0
0	1	2	0	3	5	0
0	0	0	0	0	0	0

Zero Padding

filter无法捕获图像边缘的pattern

- ✓ 用0对图像边缘进行填充

0	0	0	0	0	0	0
0	3	0	0	1	1	0
0	0	2	0	1	2	0
0	0	0	2	4	0	0
0	1	0	1	0	1	0
0	1	2	0	3	5	0
0	0	0	0	0	0	0

Zero Padding

filter无法捕获图像边缘的pattern

✓ 用0对图像边缘进行填充

0	0	0	0	0	0	0
0	3	0	0	1	1	0
0	0	2	0	1	2	0
0	0	0	2	4	0	0
0	1	0	1	0	1	0
0	1	2	0	3	5	0
0	0	0	0	0	0	0

Zero Padding

filter无法捕获图像边缘的pattern

✓ 用0对图像边缘进行填充

0	0	0	0	0	0	0
0	3	0	0	1	1	0
0	0	2	0	1	2	0
0	0	0	2	4	0	0
0	1	0	1	0	1	0
0	1	2	0	3	5	0
0	0	0	0	0	0	0

Zero Padding

0	0	0	0	0	0	0
0	3	0	0	1	1	0
0	0	2	0	1	2	0
0	0	0	2	4	0	0
0	1	0	1	0	1	0
0	1	2	0	3	5	0
0	0	0	0	0	0	0

filter无法捕获图像边缘的pattern

- ✓ 用0对图像边缘进行填充

image：N×N　　filter：F×F

stride：1　　padding：1

activation map：　(N+1×2-F)/1+1

image：5×5　　filter：3×3

stride：1　　padding：1

activation map：　(5+1×2-3)/1+1，5×5

Zero Padding

0	0	0	0	0	0	0
0	3	0	0	1	1	0
0	0	2	0	1	2	0
0	0	0	2	4	0	0
0	1	0	1	0	1	0
0	1	2	0	3	5	0
0	0	0	0	0	0	0

filter无法捕获图像边缘的pattern

- ✓ 用0对图像边缘进行填充

image： N×N　　filter： F×F

stride： 1　　padding： 1

activation map：　$(N+1\times2-F)/1+1$

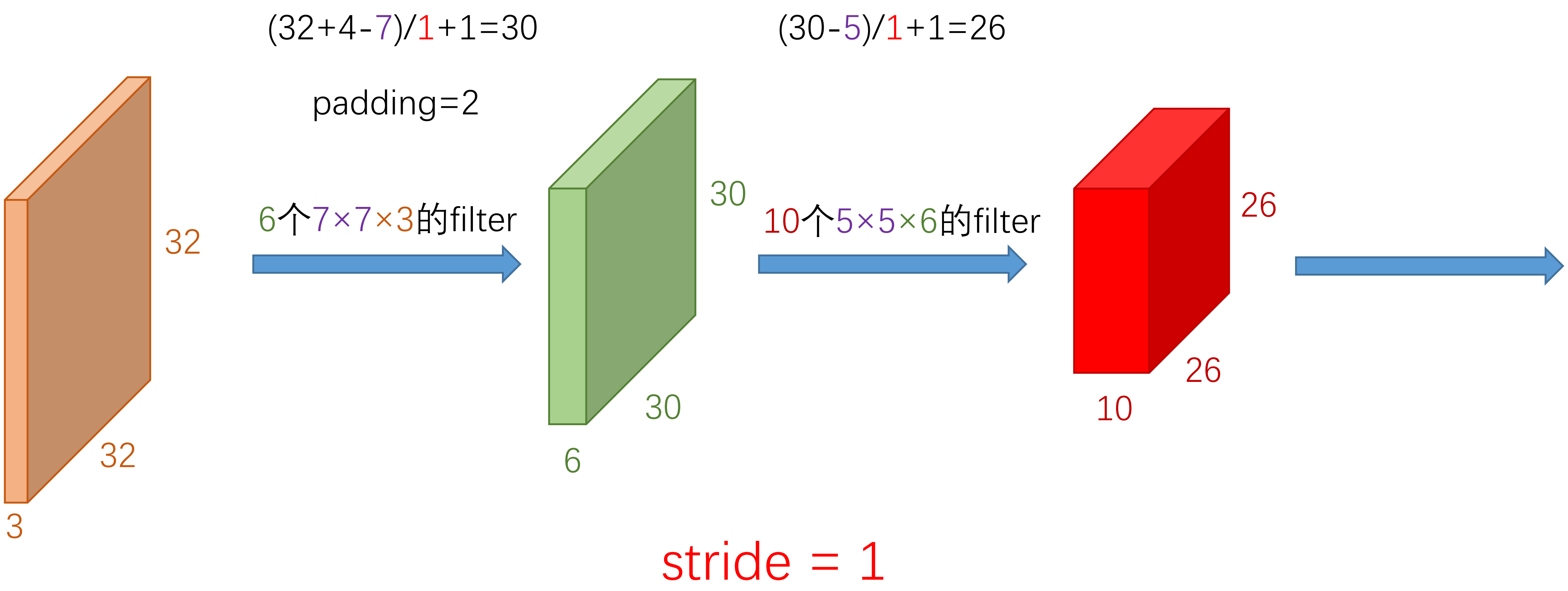
现实模型中常常会在某些卷积层使用stride=1，filter= F×F，zero-padding=(F-1)/2，来保持卷积后activation map大小不变（在pooling层减小map）

F=3 → padding大小为1

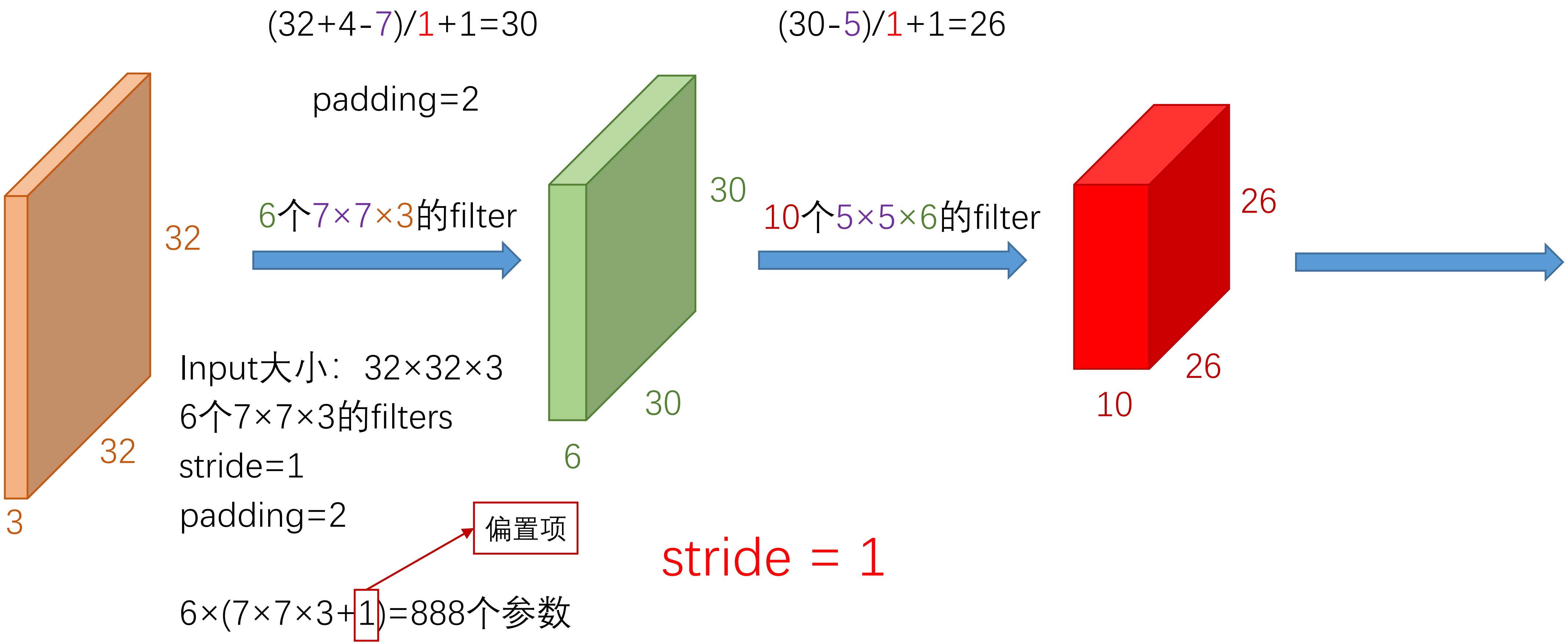
F=5 → padding大小为2

F=7 → padding大小为3

Activation Maps



Activation Maps



卷积层小结

- 当前层输入：$W_1 \times H_1 \times D_1$
- 设置4个超参数：
  - ✓filter个数K，大小F
  - ✓stride大小S
  - ✓zero padding数量P
- 引入K个$F \times F \times D_1$filter，共计$F \times F \times D_1 \times K$个权重参数和$K$个偏置项
- 生成大小为$W_2 \times H_2 \times D_2$的activation maps：
  - ✓$W_2 = \frac{W_1 - F + 2P}{S} + 1$
  - ✓$H_2 = \frac{H_1 - F + 2P}{S} + 1$
  - ✓$D_2 = K$
- 第d个activation map是第d个filter与输入层做卷积的结果（每个卷积都加上第d个偏置项）

卷积层小结

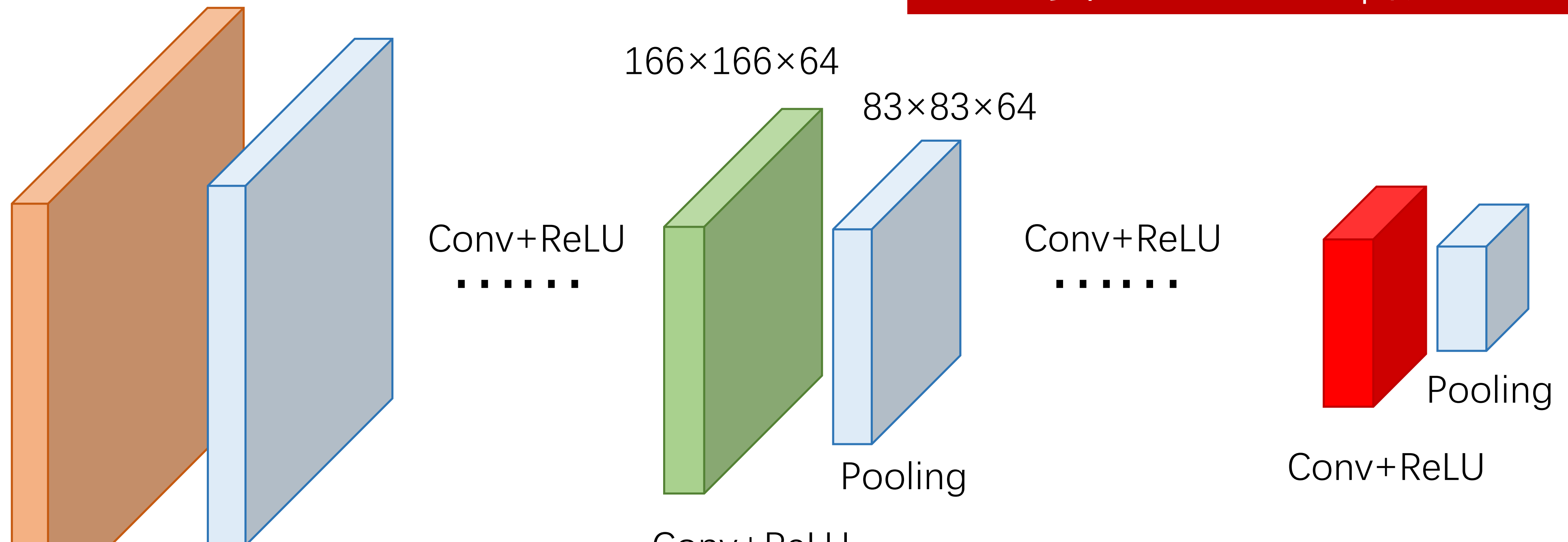
- 当前层输入：$W_1 \times H_1 \times D_1$
- 设置4个超参数：
  - ✓filter个数K，大小F
  - ✓stride大小S
  - ✓zero padding数量P
- 引入K个$F \times F \times D_1$filter，共计$F \times F \times D_1 \times K$个权重参数和$K$个偏置项
- 生成大小为$W_2 \times H_2 \times D_2$的activation maps：
  - ✓$W_2 = \frac{W_1 - F + 2P}{S} + 1$
  - ✓$H_2 = \frac{H_1 - F + 2P}{S} + 1$
  - ✓$D_2 = K$
- 第d个activation map是第d个filter与输入层做卷积的结果（每个卷积都加上第d个偏置项）

常用设置：
K=32，64，128，256，……

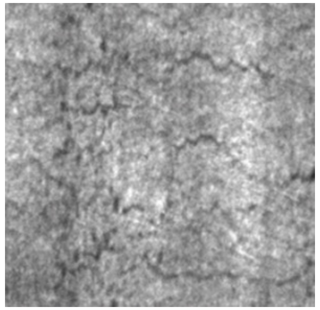
F=3，S=1，P=1
F=5，S=1，P=1或2
F=5，S=2，P=满足整除
F=1，S=1，P=0

池化层

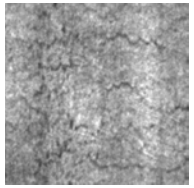
- ✓ 每隔几个Conv+ReLU层增加一层Pooling
- ✓ 对每个activation map做downsample



166×166



downsampling



83×83

减少了75%参数!

最大池化（Max Pooling）

单个activation map

3	0	0	4
0	2	5	1
3	8	2	4
1	0	1	0

最大池化（Max Pooling）

单个activation map

3	0	0	4
0	2	5	1
3	8	2	4
1	0	1	0

2×2的filter

stride=2

3	5
8	4

- ✓ 也可以做sum或者mean等
- ✓ 池化区域可以不重叠，也可以重叠

进一步减少参数数量

池化层小结

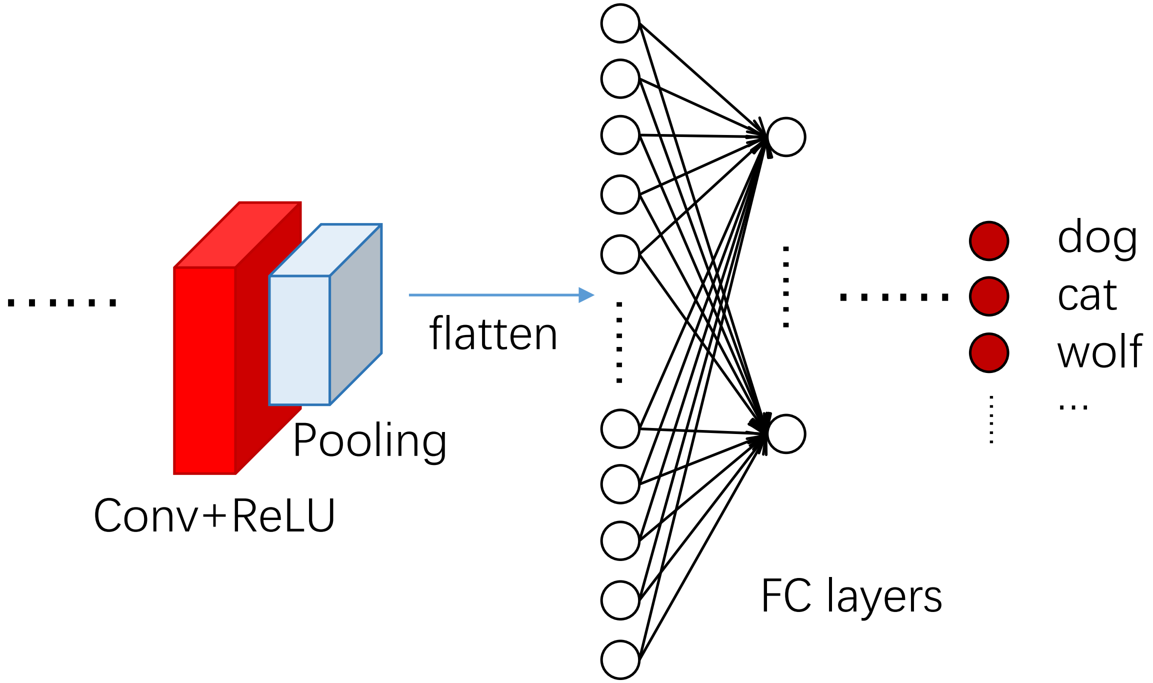
- Conv+ReLU层输出： $W_1 \times H_1 \times D_1$
- 设置2个超参数:
  - ✓filter大小F
  - ✓stride大小S

✓ 只需要一个没有参数的filter
✓ 一般不做padding

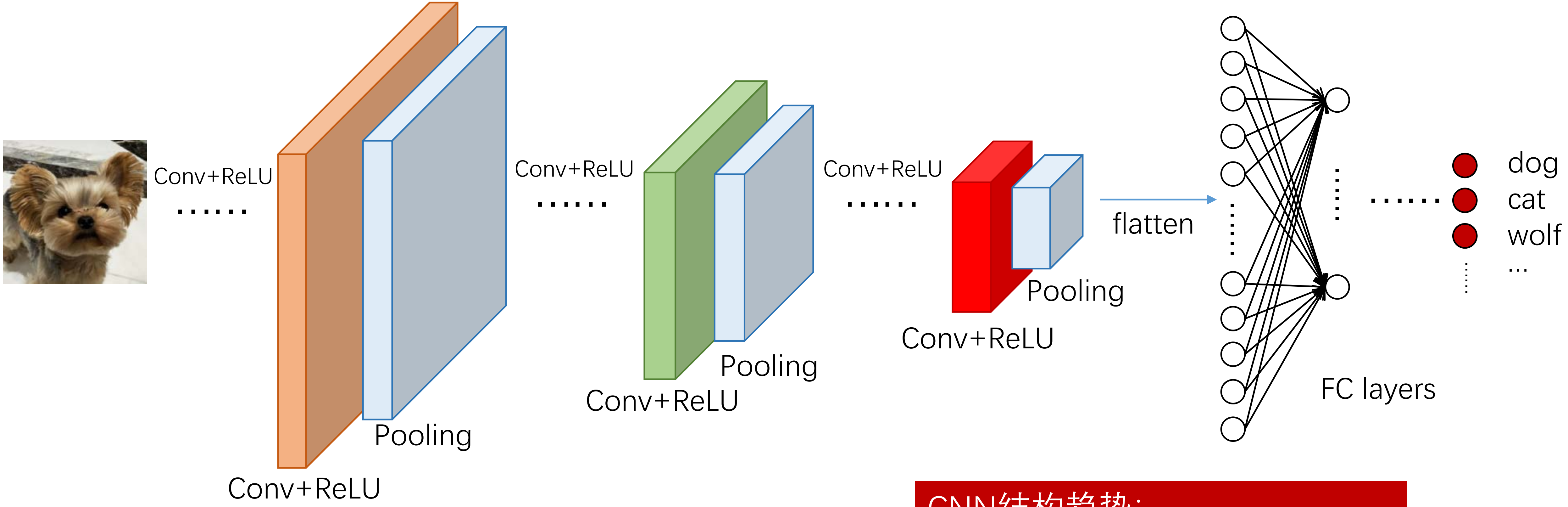
常用设置:
F=2， S=2
F=3， S=2

- 生成大小为$W_2 \times H_2 \times D_2$的feature maps:
  - ✓$W_2 = \frac{W_1 - F}{S} + 1$
  - ✓$H_2 = \frac{H_1 - F}{S} + 1$
  - ✓$D_2 = D_1$

FC层

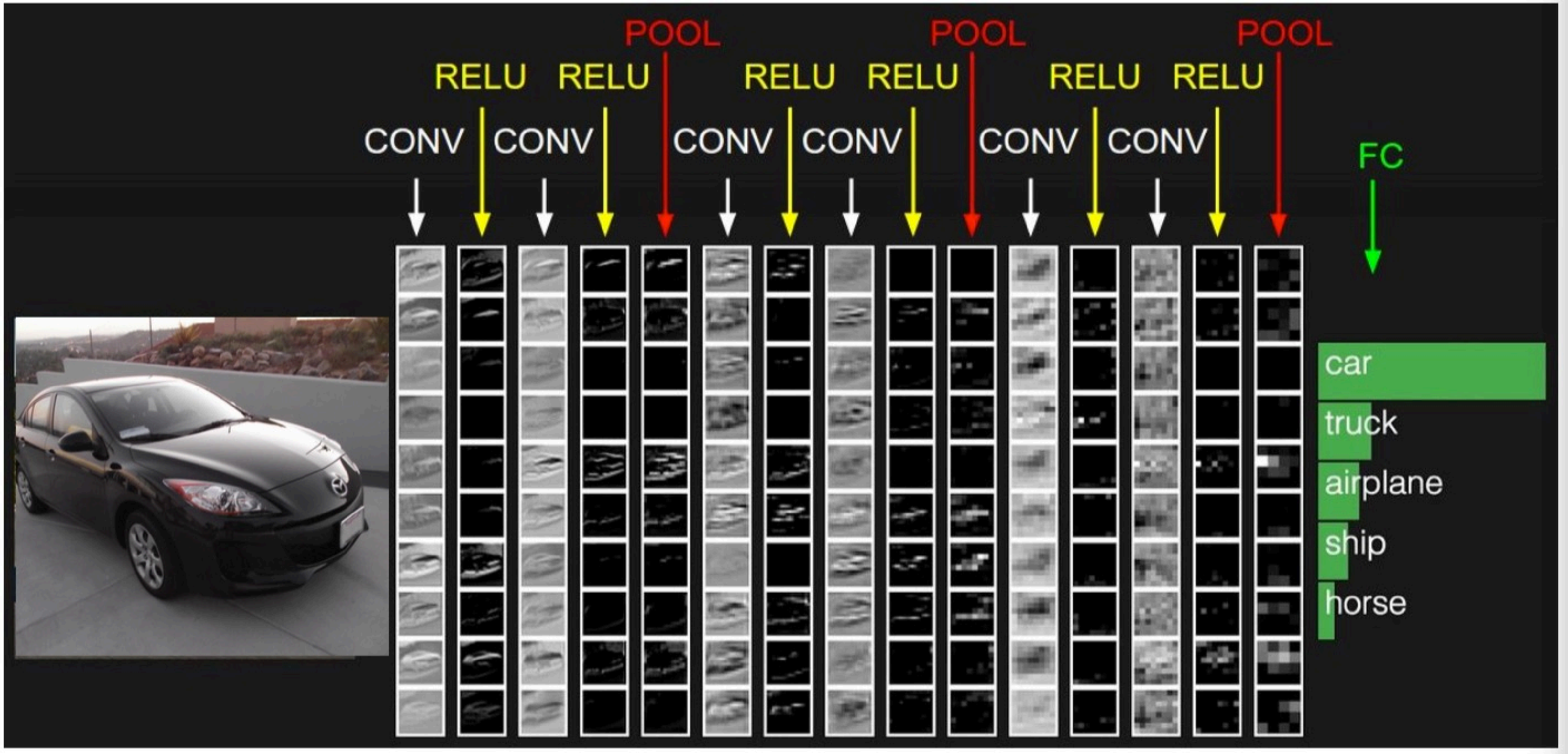


完整的CNN



CNN结构趋势：
- ✓ 使用较小的filter，加深网络
- ✓ 移除Pooling层和FC层

完整的CNN



POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

反向传播：池化层（Max pooling）

forward pass阶段记
住input最大的位置

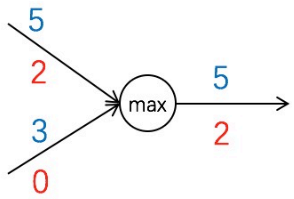
$\boxed{3}$	0	0	4
0	2	$\boxed{5}$	1
3	$\boxed{8}$	2	$\boxed{4}$
1	0	1	0

将上游梯度回传到
input最大的位置

$$\frac{\partial o}{\partial p}$$

-1	4
3	2

max gate：上游梯度路由给较大变量



反向传播：池化层（Max pooling）

$$\frac{\partial o}{\partial a}$$

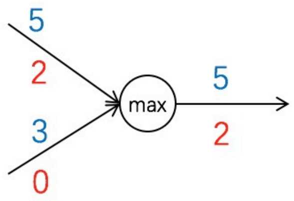
-1	0	0	0
0	0	4	0
0	3	0	2
0	0	0	0

将上游梯度回传到
input最大的位置

$$\frac{\partial o}{\partial p}$$

-1	4
3	2

max gate：上游梯度路由给较大变量



反向传播：激活层（ReLU）

$y = W^T X$

3	-2	-2	4
-3	2	5	1
3	8	2	4
1	-4	1	-2

- ✓ input>0，回传梯度
- ✓ 否则，回传0

$\frac{\partial o}{\partial a}$

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

反向传播：激活层（ReLU）

$\frac{\partial o}{\partial y}$

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

- ✓ input>0，回传梯度
- ✓ 否则，回传0

$\frac{\partial o}{\partial a}$

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

1. 回传梯度矩阵$\frac{\partial o}{\partial a}$
2. Input<=0处置0

反向传播：卷积层（不含激活）

Input X

3	0	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5

stride=1

×

$\frac{\partial o}{\partial w}$

Filter W

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

→

$\frac{\partial o}{\partial y}$

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

反向传播：卷积层（不含激活）

Input X

3	0	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5

stride=1

×

$\frac{\partial o}{\partial w}$

Filter W

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

→

$\frac{\partial o}{\partial y}$

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

反向传播：卷积层（不含激活）

Input X

3	0	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5

stride=1

×

$\frac{\partial o}{\partial w}$

Filter W

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

影响

$\frac{\partial o}{\partial y}$

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

反向传播：卷积层（不含激活）

Input X

3	0	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5

stride=1

×

$\frac{\partial o}{\partial w}$

Filter W

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

影响

$\frac{\partial o}{\partial y}$

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

$$\frac{\partial o}{\partial w_i} = \sum_j \frac{\partial o}{\partial y_j} \frac{\partial y_j}{\partial w_i}$$

反向传播：卷积层（不含激活）

Input X

3	0	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5

stride=1

×

$\frac{\partial o}{\partial w}$

Filter W

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

影响

$\frac{\partial o}{\partial y}$

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

$$\frac{\partial o}{\partial w_i} = \sum_j \frac{\partial o}{\partial y_j} \boxed{\frac{\partial y_j}{\partial w_i}}$$

反向传播：卷积层（不含激活）

Input X

3	0	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5

stride=1

×

$$\frac{\partial o}{\partial w}$$

Filter W

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

影响

$$\frac{\partial o}{\partial y}$$

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

$$\frac{\partial o}{\partial w_i} = \sum_j \frac{\partial o}{\partial y_j} \boxed{\frac{\partial y_j}{\partial w_i}}$$

反向传播：卷积层（不含激活）

Input X

3	0	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5

stride=1

×

$\frac{\partial o}{\partial w}$

Filter W

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

影响

$\frac{\partial o}{\partial y}$

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

$$\frac{\partial o}{\partial w_i} = \sum_j \frac{\partial o}{\partial y_j} \boxed{\frac{\partial y_j}{\partial w_i}}$$

$\frac{\partial o}{\partial y}$ 对X的每一个map做卷积!

反向传播：卷积层（不含激活）

$\frac{\partial o}{\partial x}$

Input X

3	0	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5

stride=1

×

Filter W

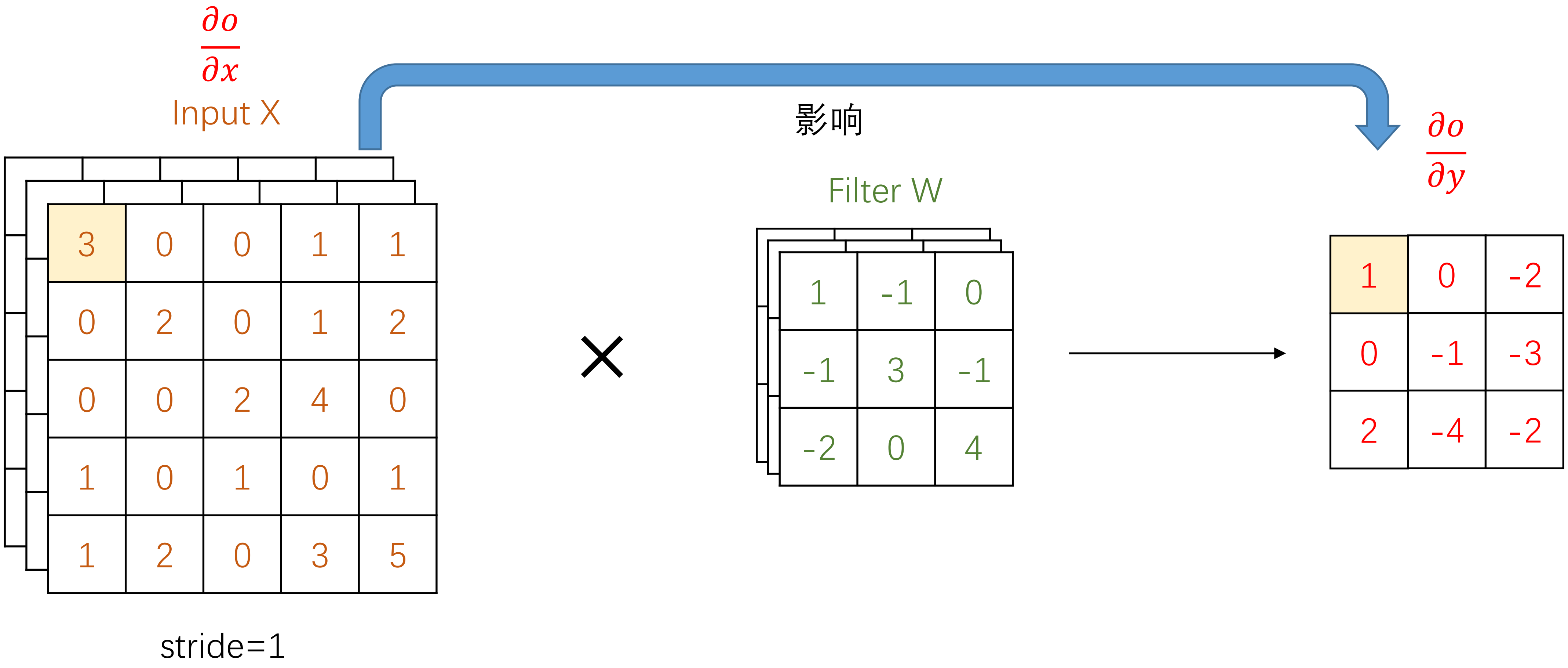
1	-1	0
-1	3	-1
-2	0	4

→

$\frac{\partial o}{\partial y}$

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

反向传播：卷积层（不含激活）



反向传播：卷积层（不含激活）

$\frac{\partial o}{\partial x}$

Input X

3	0	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5

stride=1

影响

$\times$

Local gradient　Filter W

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

$\frac{\partial o}{\partial y}$

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

反向传播：卷积层（不含激活）

$\frac{\partial o}{\partial x}$

Input X

3	0	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5

stride=1

影响

$\times$

Filter W

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

$\longrightarrow$

$\frac{\partial o}{\partial y}$

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

反向传播：卷积层（不含激活）

$\frac{\partial o}{\partial x}$

Input X

3	0	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5

stride=1

影响

Local gradient Filter W

×

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

$\frac{\partial o}{\partial y}$

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

反向传播：卷积层（不含激活）

$\frac{\partial o}{\partial x}$

Input X

3	0	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5

stride=1

影响

$\times$

Local gradient Filter W

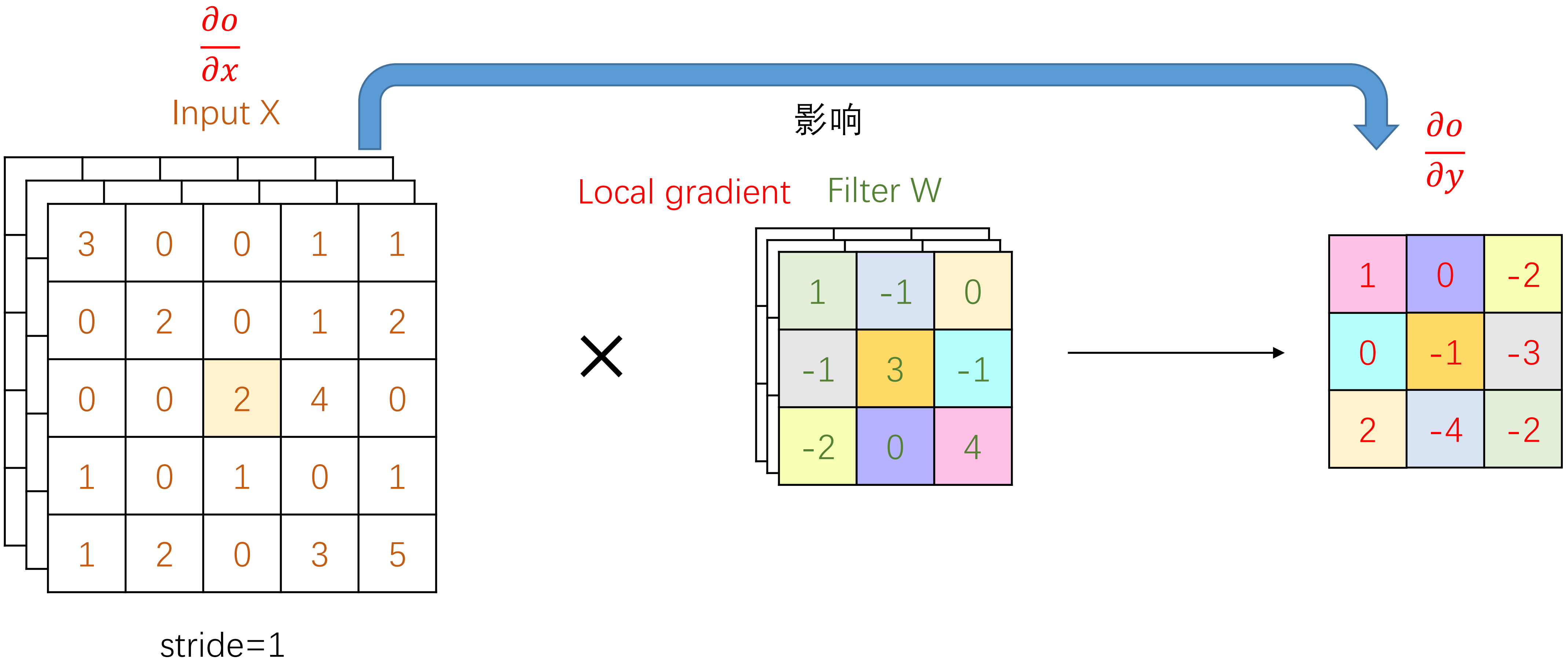
1	-1	0
-1	3	-1
-2	0	4

$\rightarrow$

$\frac{\partial o}{\partial y}$

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

反向传播：卷积层（不含激活）



反向传播：卷积层（不含激活）

原始$\frac{\partial o}{\partial y}$

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

$\frac{\partial o}{\partial x}$

Input X

3	0	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5

stride=1

影响

Local gradient × Filter W

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

翻转后的$\frac{\partial o}{\partial y}$

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

反向传播：卷积层（不含激活）

$\frac{\partial y}{\partial x}$

Input X

3	0	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5

stride=1

Loc

×

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

原始$\frac{\partial o}{\partial y}$

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

翻转后的$\frac{\partial o}{\partial y}$

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

反向传播：卷积层（不含激活）

$$\frac{\partial o}{\partial x}=\frac{\partial o}{\partial y}\frac{\partial y}{\partial x}$$

原始$\frac{\partial o}{\partial y}$

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

1	0	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5

stride=1

Loc

×

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

翻转后的$\frac{\partial o}{\partial y}$

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

反向传播：卷积层（不含激活）

$$\frac{\partial o}{\partial x} = \frac{\partial o}{\partial y}\frac{\partial y}{\partial x}$$

1	-1	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5

stride=1

Loc

×

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

原始$\frac{\partial o}{\partial y}$

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

翻转后的$\frac{\partial o}{\partial y}$

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

反向传播：卷积层（不含激活）

原始$\frac{\partial o}{\partial y}$

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

$$\frac{\partial o}{\partial x} = \frac{\partial o}{\partial y}\frac{\partial y}{\partial x}$$

1	-1	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5

stride=1

Loc

×

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

翻转后的$\frac{\partial o}{\partial y}$

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

将$\frac{\partial o}{\partial y}$翻转，然后对padding后的filter的每一层做卷积

Assignment 2

```python
def conv_forward_naive(x, w, b, conv_param):
    """
    A naive implementation of the forward pass for a convolutional layer.

    The input consists of N data points, each with C channels, height H and
    width W. We convolve each input with F different filters, where each filter
    spans all C channels and has height HH and width WW.

    Input:
    - x: Input data of shape (N, C, H, W)
    - w: Filter weights of shape (F, C, HH, WW)
    - b: Biases, of shape (F,)
    - conv_param: A dictionary with the following keys:
      - 'stride': The number of pixels between adjacent receptive fields in the
        horizontal and vertical directions.
      - 'pad': The number of pixels that will be used to zero-pad the input.


    During padding, 'pad' zeros should be placed symmetrically (i.e equally on both sides)
    along the height and width axes of the input. Be careful not to modfiy the original
    input x directly.

    Returns a tuple of:
    - out: Output data, of shape (N, F, H', W') where H' and W' are given by
      H' = 1 + (H + 2 * pad - HH) / stride
      W' = 1 + (W + 2 * pad - WW) / stride
    - cache: (x, w, b, conv_param)
    """
    out = None
    ###########################################################################
    # TODO: Implement the convolutional forward pass.                         #
    # Hint: you can use the function np.pad for padding.                      #
    ###########################################################################
    # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

    pass

    # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    ###########################################################################
    #                             END OF YOUR CODE                            #
    ###########################################################################
    cache = (x, w, b, conv_param)
    return out, cache
```

```python
def conv_backward_naive(dout, cache):
    """
    A naive implementation of the backward pass for a convolutional layer.

    Inputs:
    - dout: Upstream derivatives.
    - cache: A tuple of (x, w, b, conv_param) as in conv_forward_naive

    Returns a tuple of:
    - dx: Gradient with respect to x
    - dw: Gradient with respect to w
    - db: Gradient with respect to b
    """
    dx, dw, db = None, None, None
    ###########################################################################
    # TODO: Implement the convolutional backward pass.                        #
    ###########################################################################
    # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

    pass

    # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    ###########################################################################
    #                             END OF YOUR CODE                            #
    ###########################################################################
    return dx, dw, db
```

CNN using Keras

```
from keras.models import Sequential
from keras.layers import Dense, Convolution2D, Flatten, MaxPooling2D


#create model
model = Sequential()


#add model layers
model.add(Convolution2D(64, kernel_size=3, activation='relu', input_shape=(3,32,32)))

model.add(MaxPooling2D((2,2)))

model.add(Convolution2D(32, kernel_size=3, activation='relu'))

model.add(Flatten())

model.add(Dense(10, activation='softmax'))
```

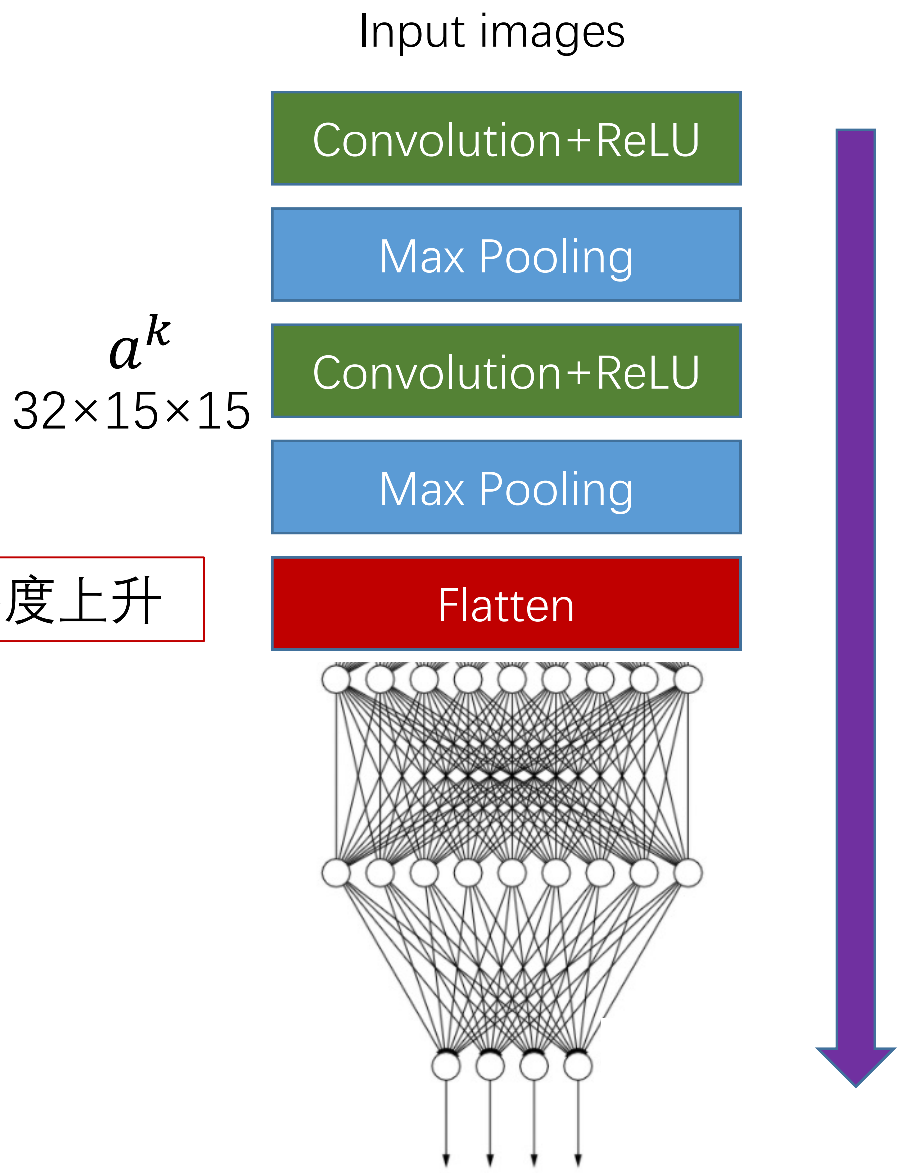
CNN学了什么?

假设每个卷积层activation map $a^k \in \mathbb{R}^{15\times15}$

定义activation degree: $\mathrm{ad}(a^k) \in \sum a_{ij}$

$\forall a^k$，寻找： $x^k = arg\max_x \mathrm{ad}(a^k)$

梯度上升



CNN学了什么?

假设每个卷积层activation map $a^k \in \mathbb{R}^{15\times15}$

定义activation degree: $\mathrm{ad}(a^k) \in \sum a_{ij}$

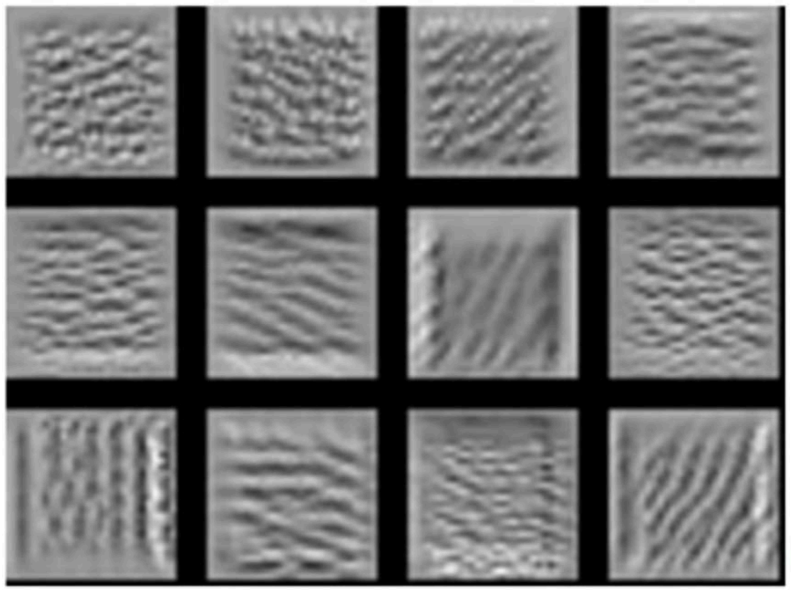
$\forall a^k$，寻找： $x^k = arg \max_{x} \mathrm{ad}(a^k)$

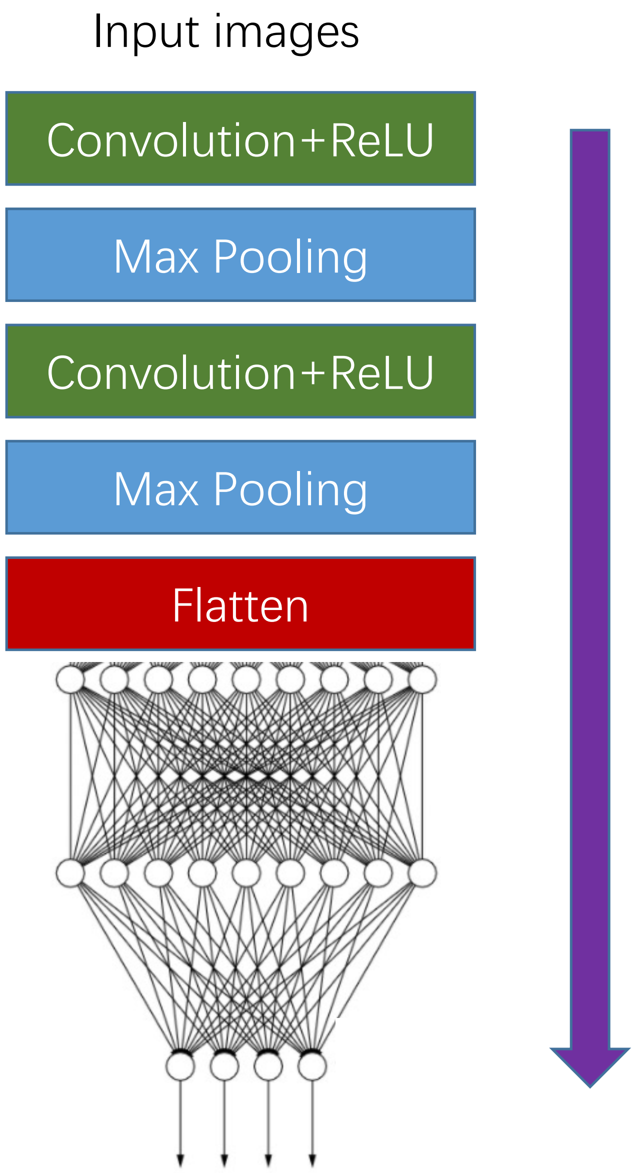
梯度上升

a^k

$32\times15\times15$

MNIST
其中9个x^k



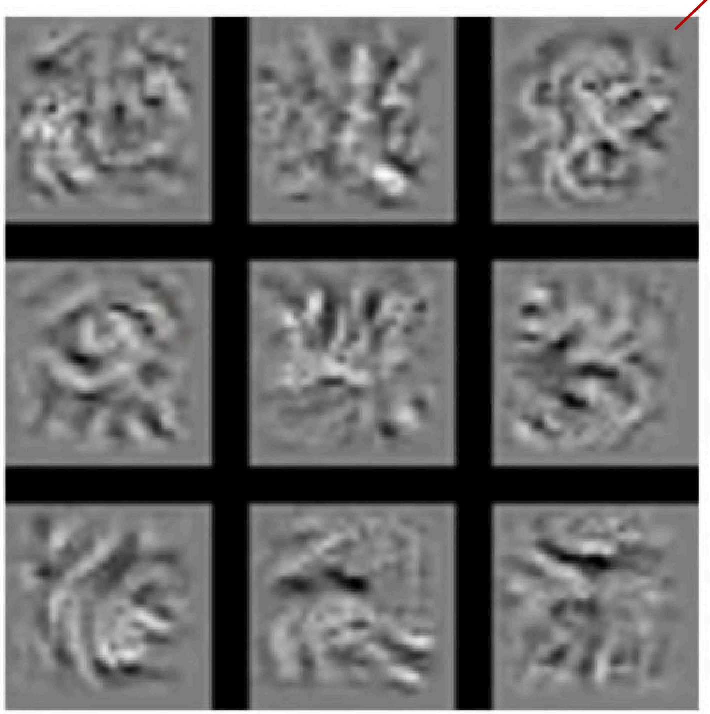


CNN学了什么？

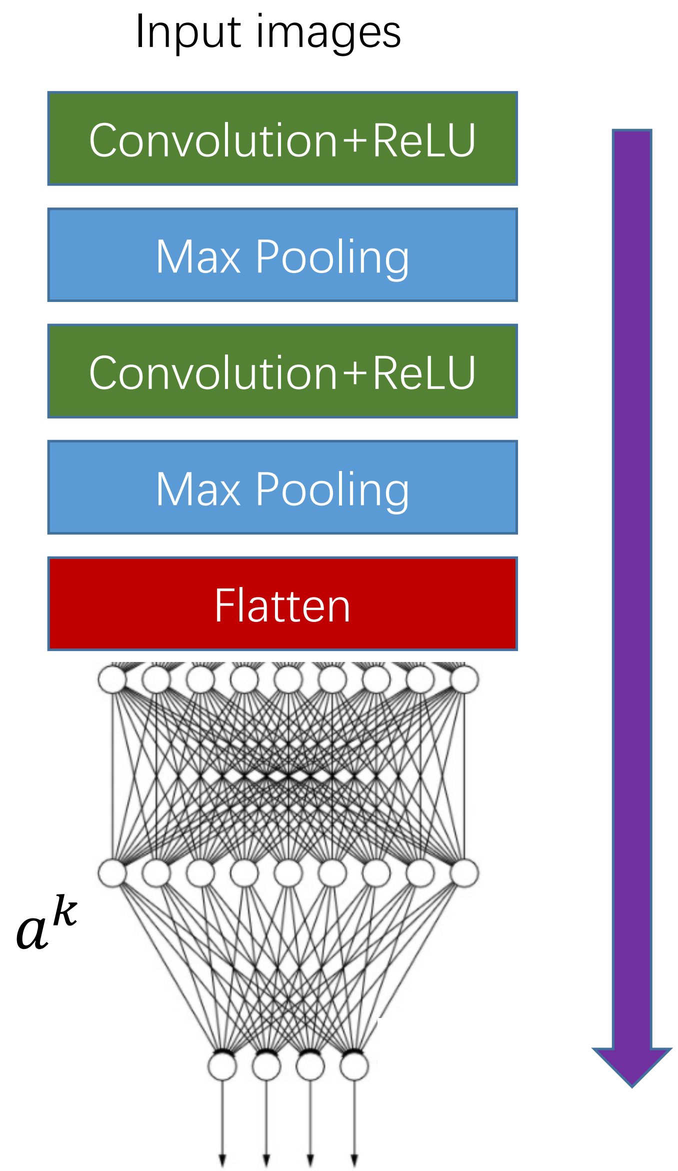
同理，当a^k为FC layer的一个神经元

$\forall a^k$，寻找： $x^k = arg\max_x a^k$

整个图片的信息



MNIST
其中9个x^k



小结

- 卷积神经网络
  - ✓Conv+ReLU
  - ✓Max pooling
- 各层的反向传播
- 理解CNN

L06

- CNN的应用

- 神经网络训练细节
  - ✓激活函数
  - ✓数据预处理
  - ✓参数初始化
  - ✓Batch Normalization