深度学习讨论班

第三节 Convolutional Neural Networks (卷积神经网络)

黄雷

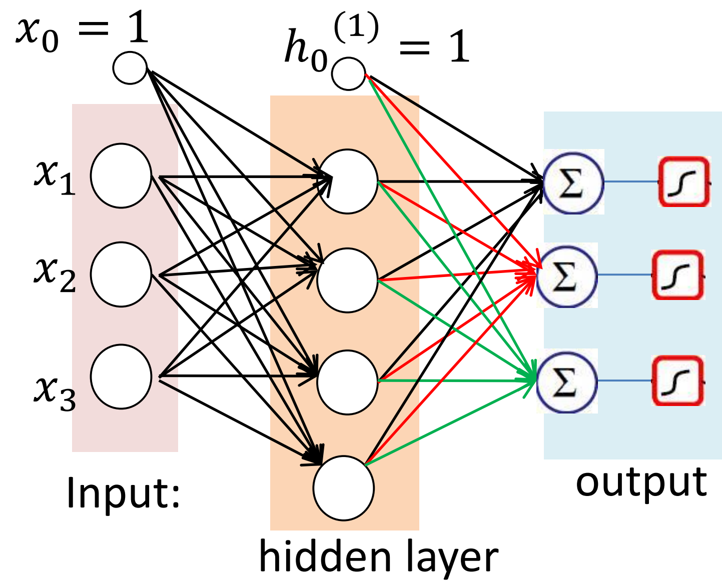
2016-12-13

上一讲主要内容

- Linear classifier (简单线性分类器)
  - One neuron (一个神经元)
  - Multiple neurons (多个神经元)
- Multi-layer perceptron (多层感知机)
  - Model representation (模型表示)
  - Loss function: the goal for learning
  - Training
    - Gradient based optimization
    - backpropagation

Multi-layer perceptron

- Training Algorithm



$(1, 0, 0)^T$

- 0.初始化权重 $\boldsymbol{W}^{(0)}$
- 1. 前向过程：
  - 1.1根据输入$\boldsymbol{x}$，计算输出值 $\boldsymbol{y}$
  - 1.2.计算损失函数值L($\mathbf{y}, \hat{\boldsymbol{y}}$)。
- 2.后向传播
  - 计算$\frac{\mathrm{d\ L}}{\mathbf{y}}$
  - 后向传播直到计算$\frac{\mathrm{d\ L}}{\mathbf{x}}$
- 3.计算梯度$\frac{d\,L}{\boldsymbol{d\,W}}$
- 4.更新梯度

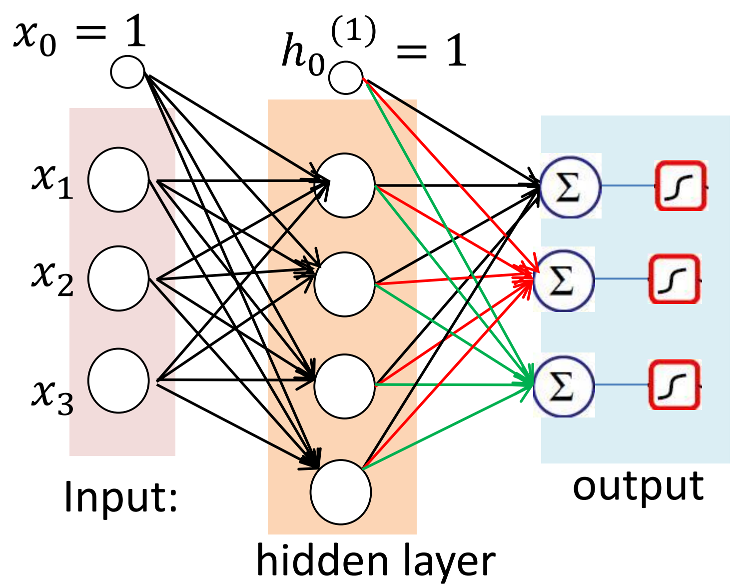
$$\boldsymbol{W}^{(t+1)} = \boldsymbol{W}^{(t)} - \eta \frac{d\,\mathrm{L}}{\boldsymbol{d\,W}^{(t)}}$$

outline

- Modeling of CNN
  – Module-wise architecture模块化结构
- Convolutional layer (module)
  – Convolution in general
  – Filters
  – Convolution module
- Pooling layer (module)

Feature extraction

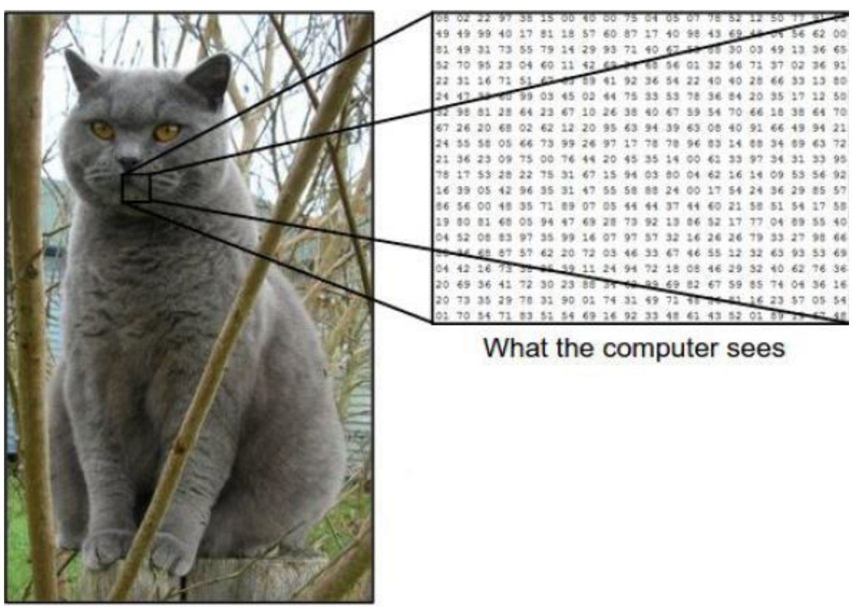
- Feature extraction
  - Pixel-wise input
  - Correlation between features



$(1, 0, 0)^T$

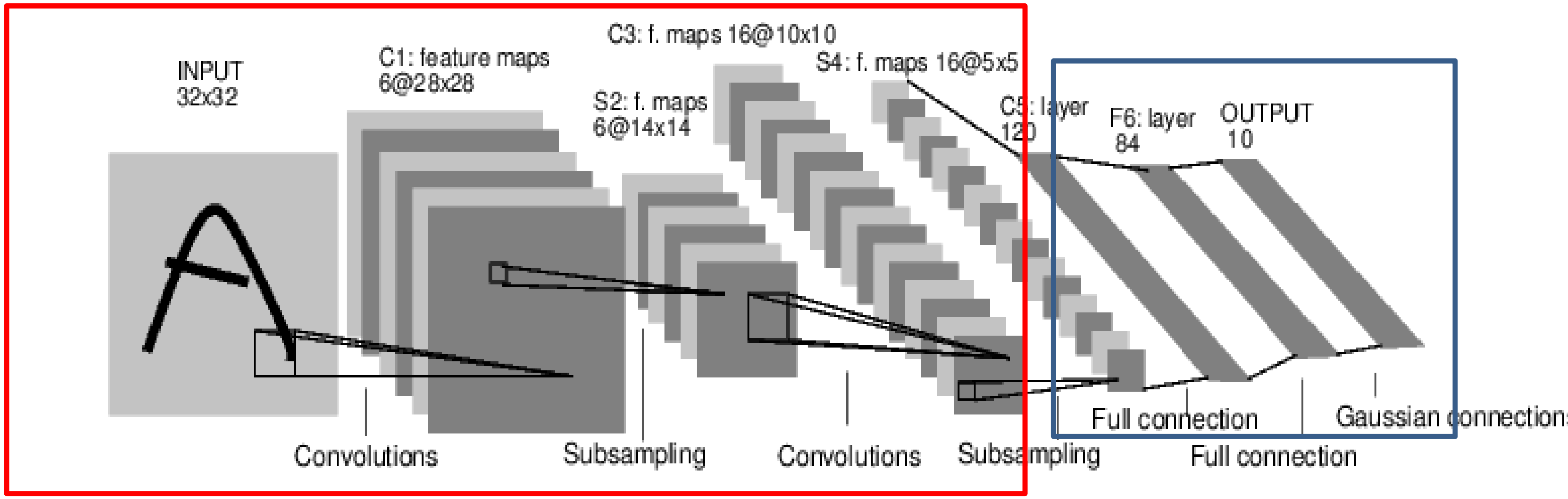
(x_1, x_2, x_3)

Convolutional Neural Network(CNN),卷积神经网络



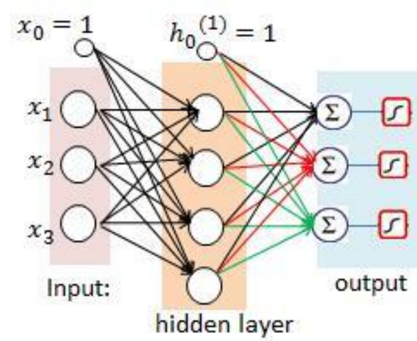
Convolution Neural Network

- Lenet-5



全连接层

Convolution related layers



outline

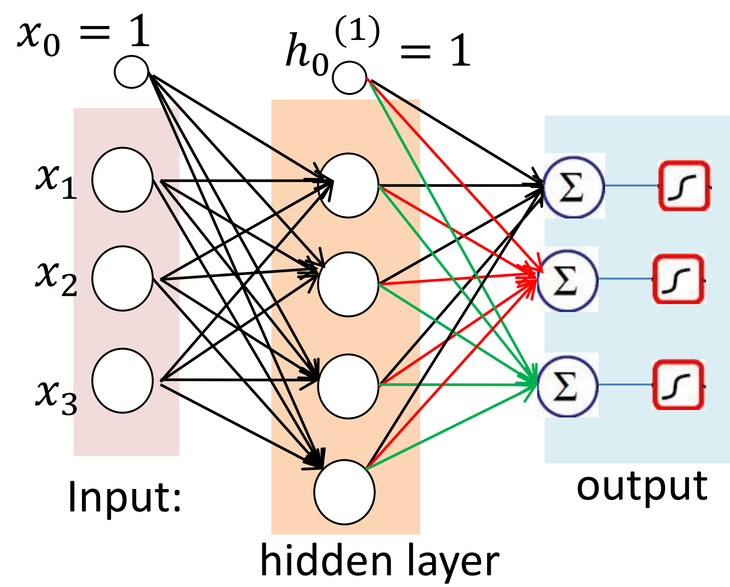
- Modeling of CNN
  - Module-wise architecture模块化结构
- Convolutional layer (module)
  - Convolution in general
  - Filters
  - Convolution module
- Pooling layer (module)

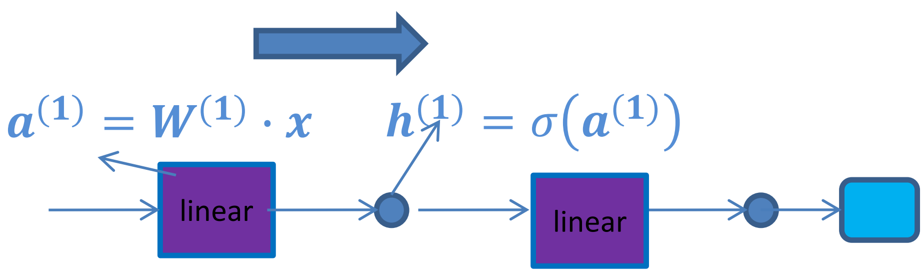
Module-wise architecture

- Torch 平台

  - **Model construction**

```
function create_model()
  model = nn.Sequential()
  model:add(nn.Linear(3, 4))
  model:add(nn.Sigmoid())
  model:add(nn.Linear(4, 3))
  model:add(nn.Sigmoid())
  criterion = nn.MSECriterion()
  return model, criterion
end
```



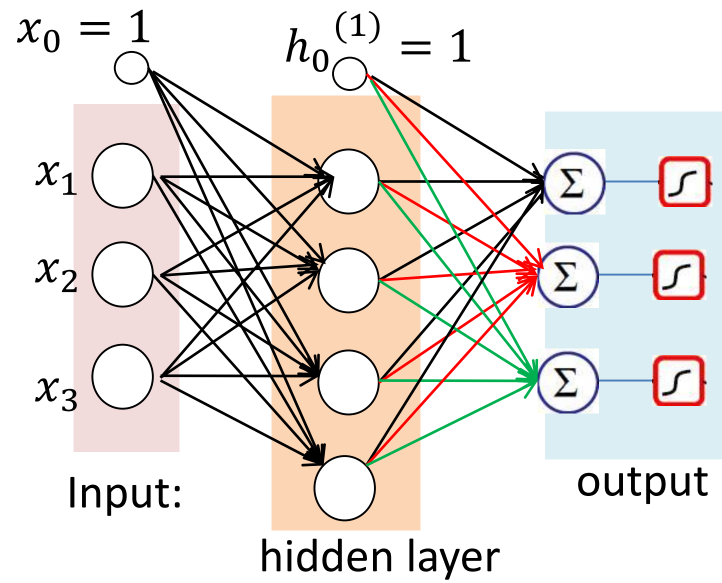


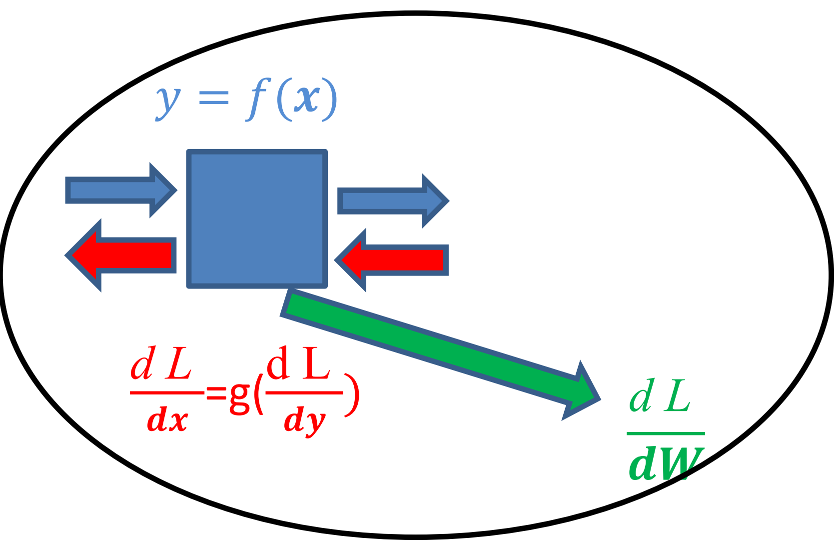
Module-wise architecture

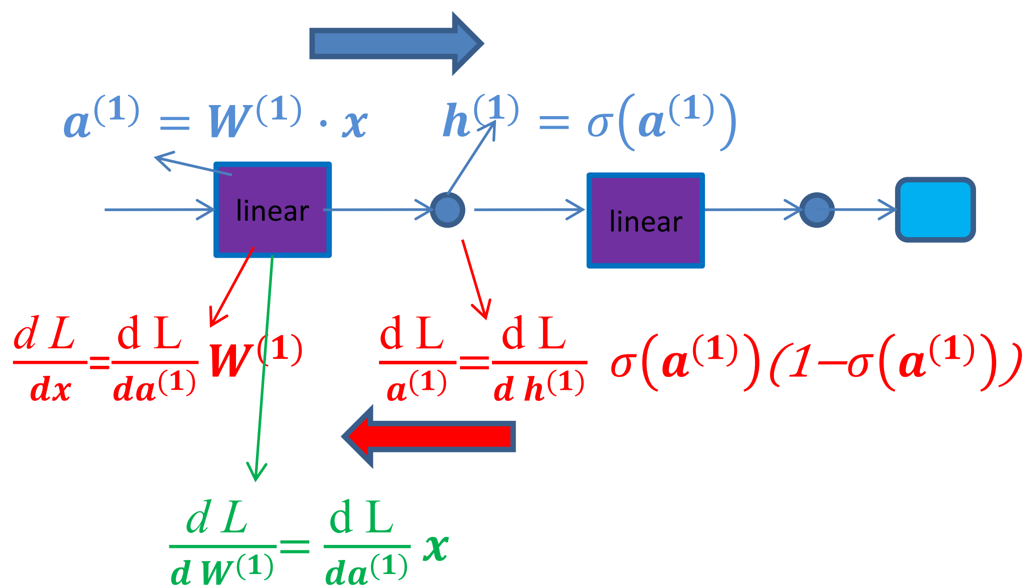
➢ Torch 平台

- **Training per iteration:**

```
-- forward
 outputs = model:forward(X)
 loss = criterion:forward(outputs, Y)
-- backward
 dloss_doutput = criterion:backward(outputs, Y)
 model:backward(X, dloss_doutput)
```

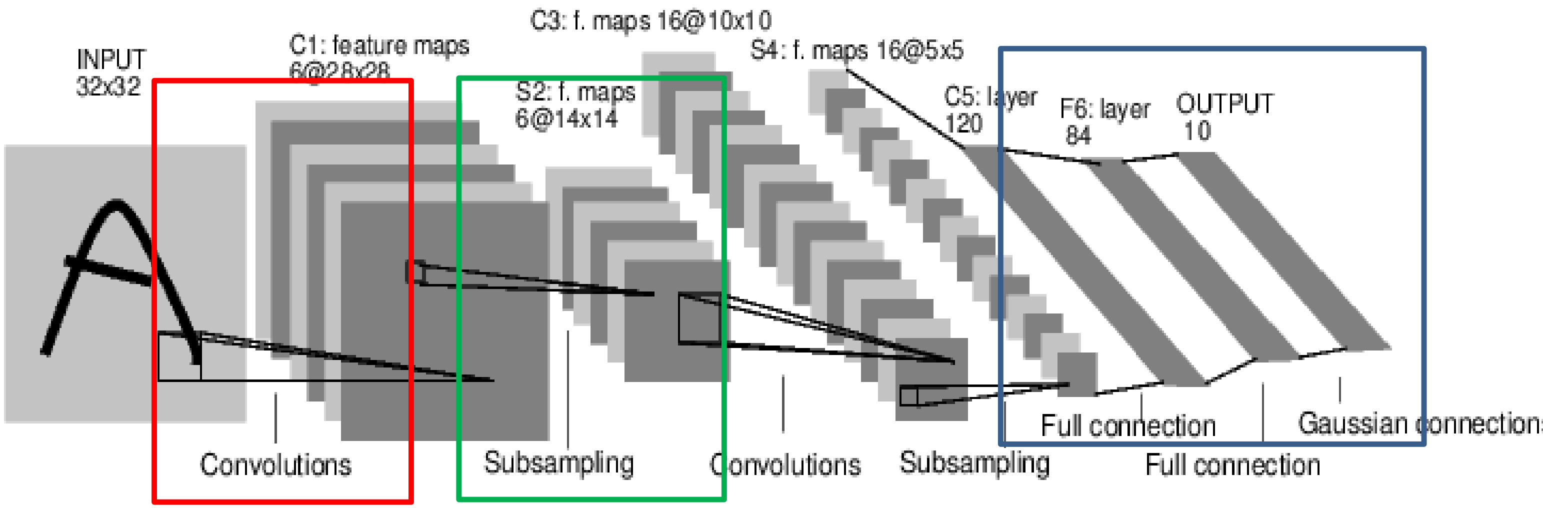






Convolution Neural Network

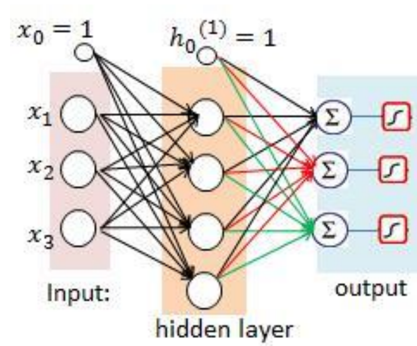
- Lenet-5



Convolution (卷积)

Pooling (池化)

全连接层

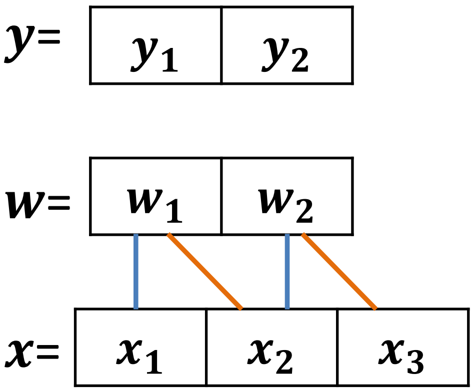


outline

- Modeling of CNN
  - Module-wise architecture模块化结构
- Convolutional layer (module)
  - Convolution in general
  - Filters
  - Convolution module
- Pooling layer (module)

Convolution

- 一维离散卷积例子



$$y_1 = w_1 x_1 + w_2 x_2$$

$$y_2 = w_1 x_2 + w_2 x_3$$

$$y_{i'} = \sum_{i=1}^{M_f=2} w_i x_{i'+i-1}$$

Correlation operator (similarity)

- Flip

$\bar{w}$= | w_2 | w_1 |

$$y_{i'} = \sum_{i=1}^{M_f=2} w_{M_f+1-i} x_{i'+i-1}$$

Convolution

- 离散空间卷积：

$$y(n) = x(n) * w(n) = \sum_{i=-\infty}^{i=+\infty} x(i) w(n-i)$$

Convolution

- 离散空间卷积：

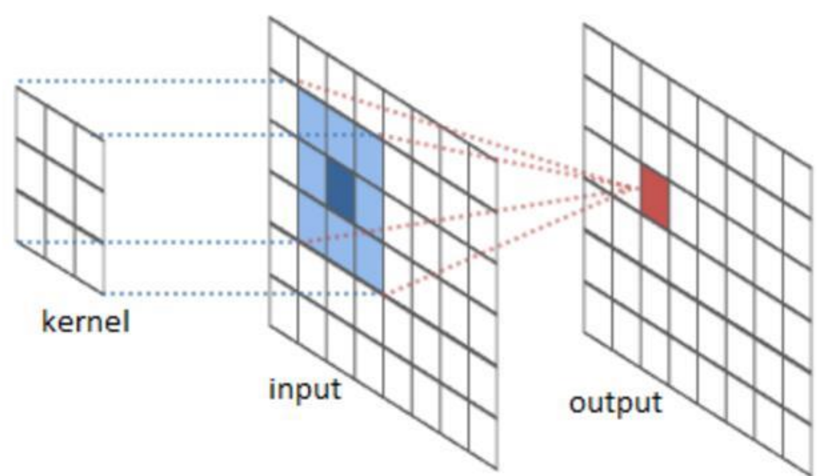
$$y(n) = x(n) * w(n) = \sum_{i=-\infty}^{i=+\infty} x(i)w(n-i)$$

- 连续空间的卷积:

$$y(t) = x(t) * h(t) = \int_{-\infty}^{+\infty} x(s)h(t-s)\, ds$$

- 图像卷积是二维离散卷积

$$g(i,j) = \sum_{k,l} f(k,l)\, w(i-k, j-l)$$



Convolution

- 图像卷积， 二维, 离散
  - Correlation Operator(相关算子)

定义：$g = f \otimes w$

$$g(i,j) = \sum_{k,l} f(i+k, j+l)\, w(k,l)$$

image

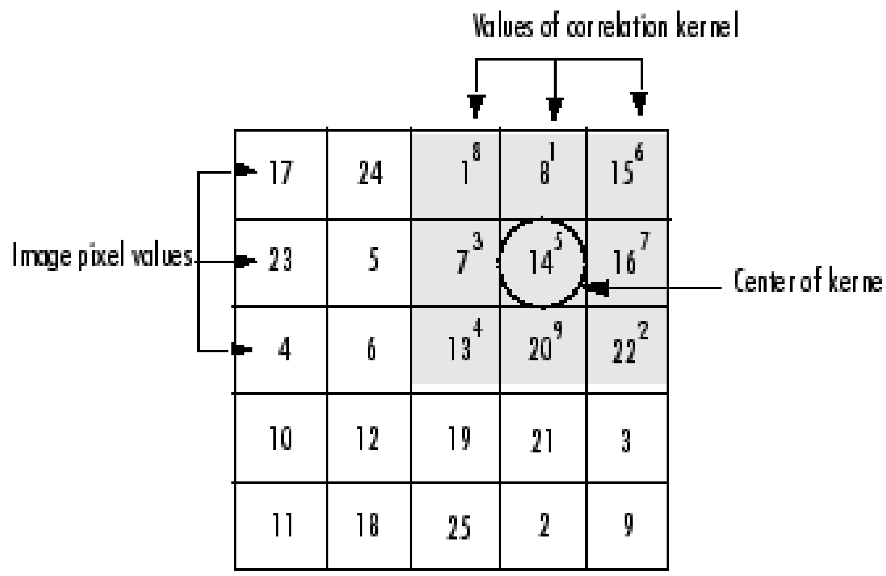
Kernel(filters)

f:

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

w:

8	1	6
3	5	7
4	9	2



Convolution

- 图像卷积，二维，离散
  - Convolution operator (卷积算子)

定义：$g = f * w$

image

Kernel(filters)

$$g(i,j) = \sum_{k,l} f(k,l)\, w(i-k, j-l)$$

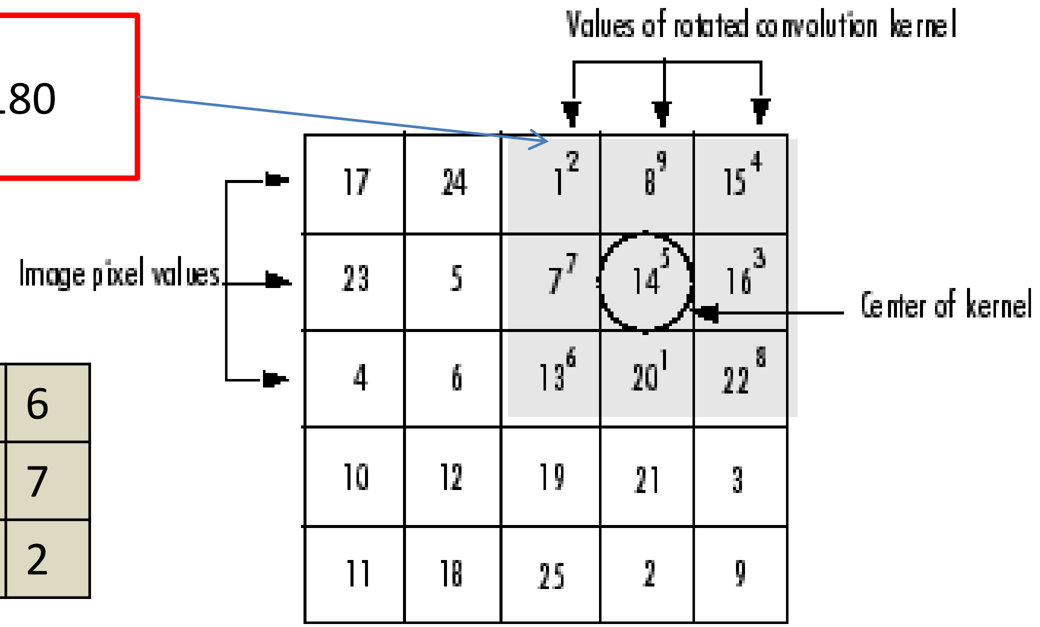
旋转180

f:

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

w:

8	1	6
3	5	7
4	9	2



outline

- Modeling of CNN
  - Module-wise architecture模块化结构
- Convolutional layer (module)
  - Convolution in general
  - Filters
  - Convolution module
- Pooling layer (module)

Practice with linear filters(线性滤波器)

Original

0	0	0
0	1	0
0	0	0

Filter

Filtered
(no change)

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	0	1
0	0	0

Filter

Shifted *left*
By 1 pixel

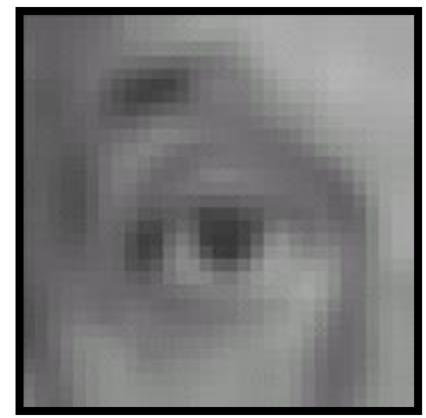
Source: D. Lowe

Practice with linear filters

Original

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Filter



Blur (with a box filter)

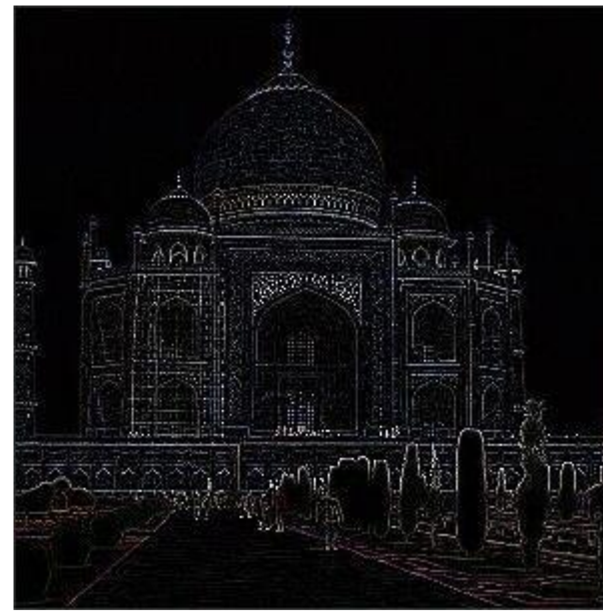
Source: D. Lowe

Practice with linear filters

Original

0	1	0
1	-4	1
0	1	0

Filter



Output Image

Edge detect （边缘检测）

Filters in practise

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

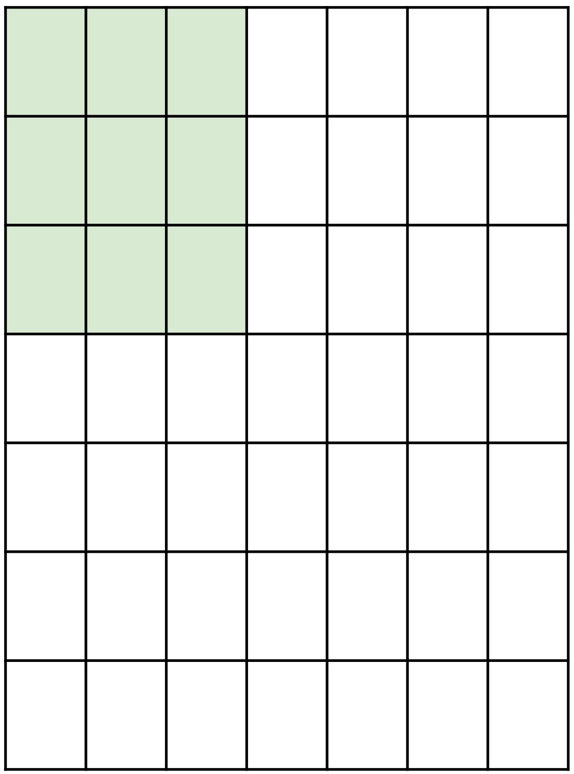
Input image

filter

output image

- Size of output image
  - How to move? stride
  - How about the border? padding

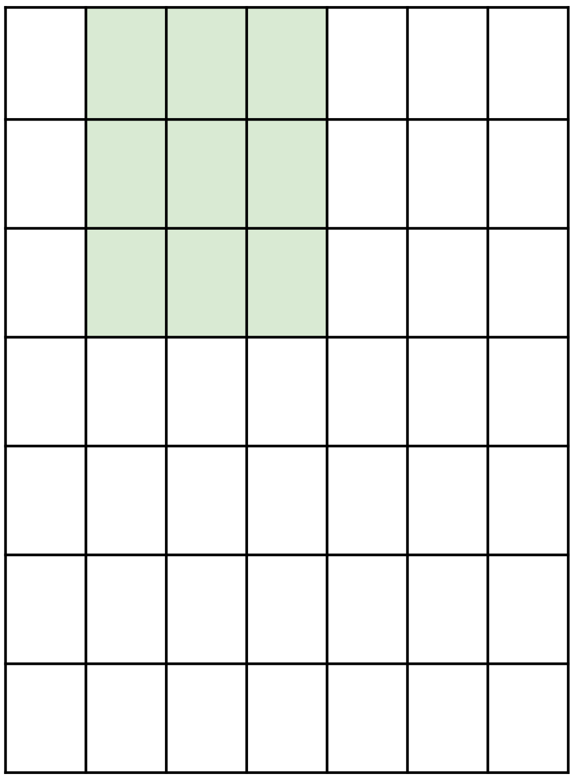
filters: stride（步幅）



7x7 input
assume 3x3 connectivity, stride 1

Source: Stanford CS231n,
Andrej Karpathy & Fei-Fei Li

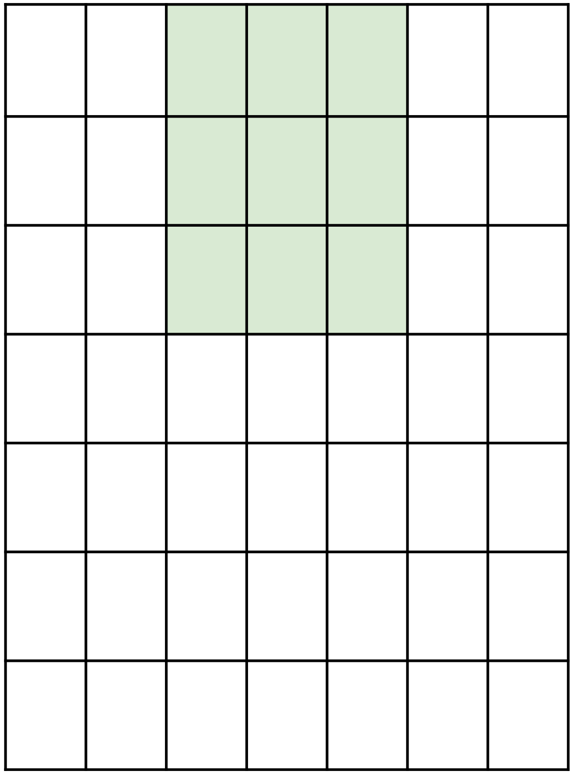
filters: stride



7x7 input
assume 3x3 connectivity, stride 1

Source: Stanford CS231n,
Andrej Karpathy & Fei-Fei Li

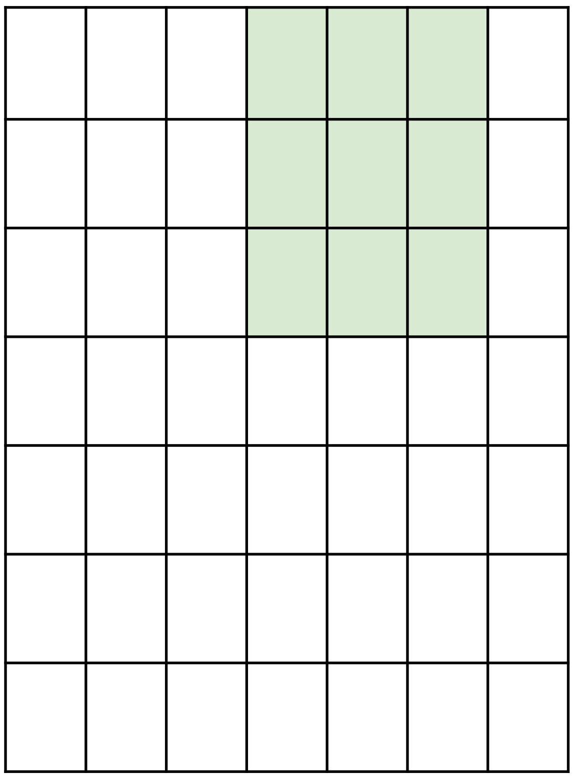
filters: stride



7x7 input
assume 3x3 connectivity, stride 1

Source: Stanford CS231n,
Andrej Karpathy & Fei-Fei Li

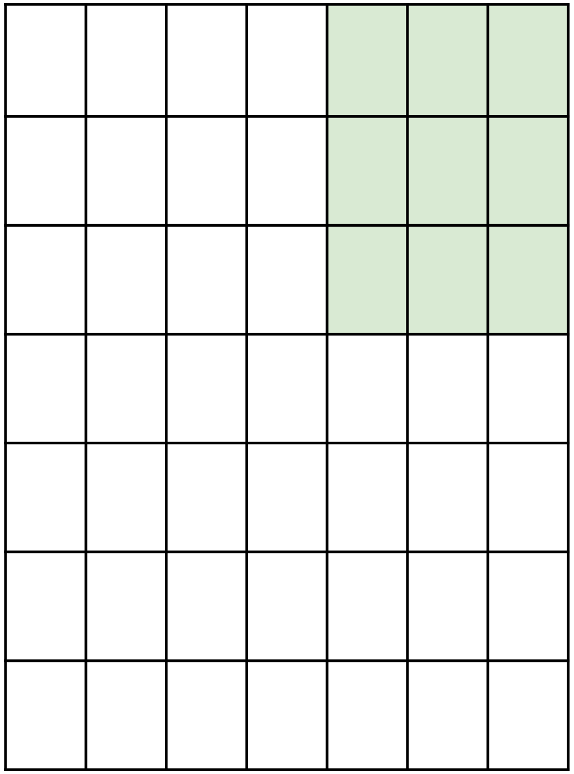
filters: stride



7x7 input
assume 3x3 connectivity, stride 1

Source: Stanford CS231n,
Andrej Karpathy & Fei-Fei Li

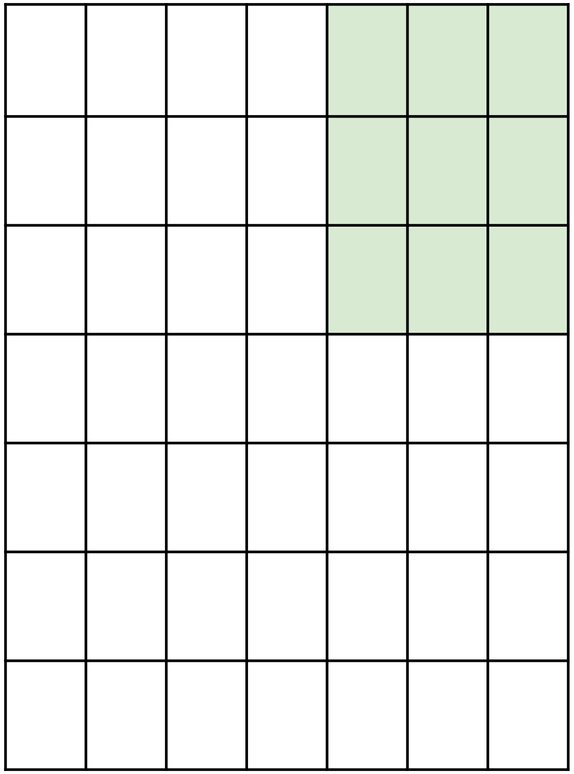
filters: stride



7x7 input
assume 3x3 connectivity, stride 1

Source: Stanford CS231n,
Andrej Karpathy & Fei-Fei Li

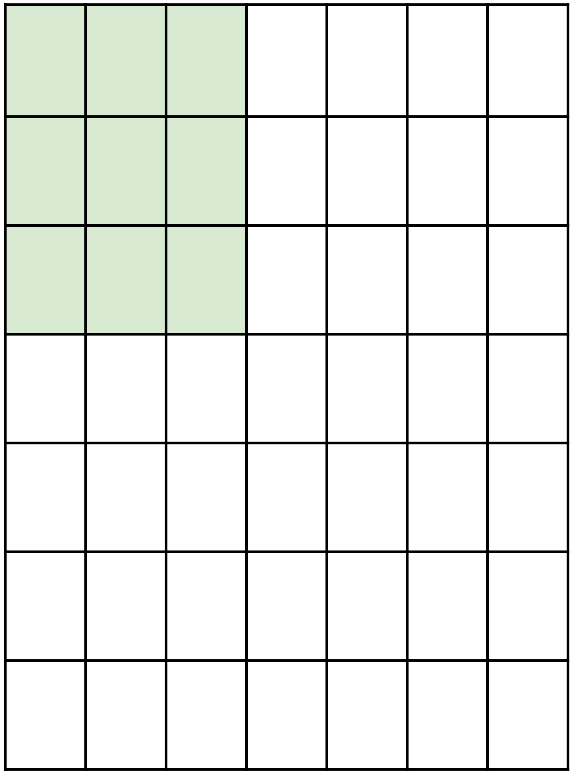
filters: stride



7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

Source: Stanford CS231n,
Andrej Karpathy & Fei-Fei Li

filters: stride

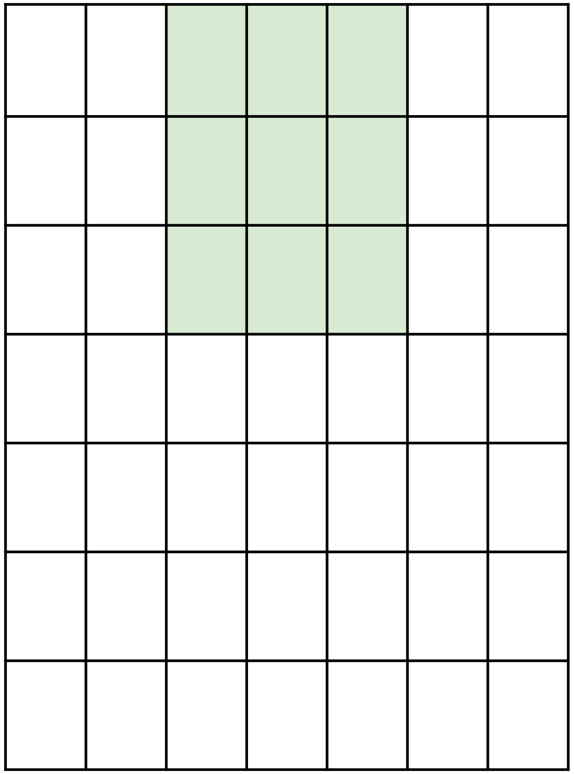


7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?

Source: Stanford CS231n,
Andrej Karpathy & Fei-Fei Li

filters: stride

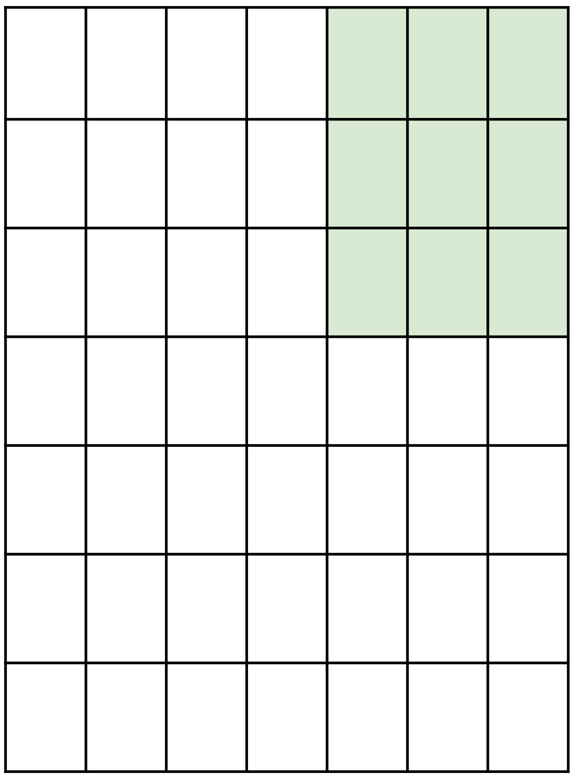


7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?

Source: Stanford CS231n,
Andrej Karpathy & Fei-Fei Li

filters: stride

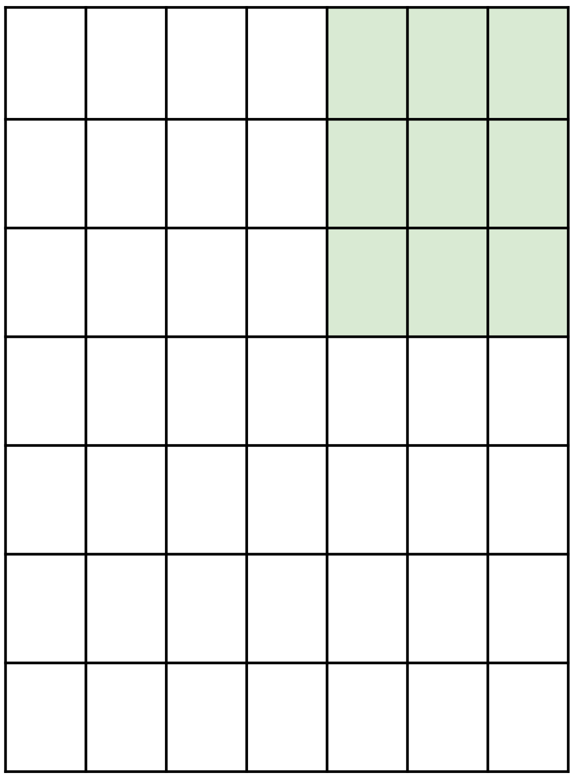


7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?

Source: Stanford CS231n,
Andrej Karpathy & Fei-Fei Li

filters: stride

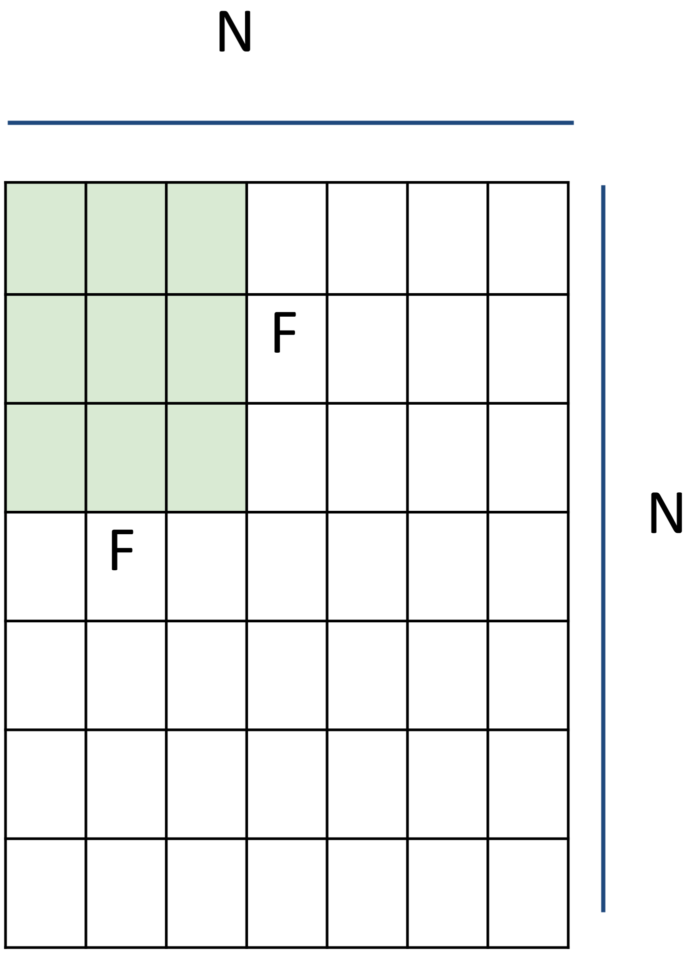


7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?
=> **3x3 output**

Source: Stanford CS231n,
Andrej Karpathy & Fei-Fei Li

filters: stride



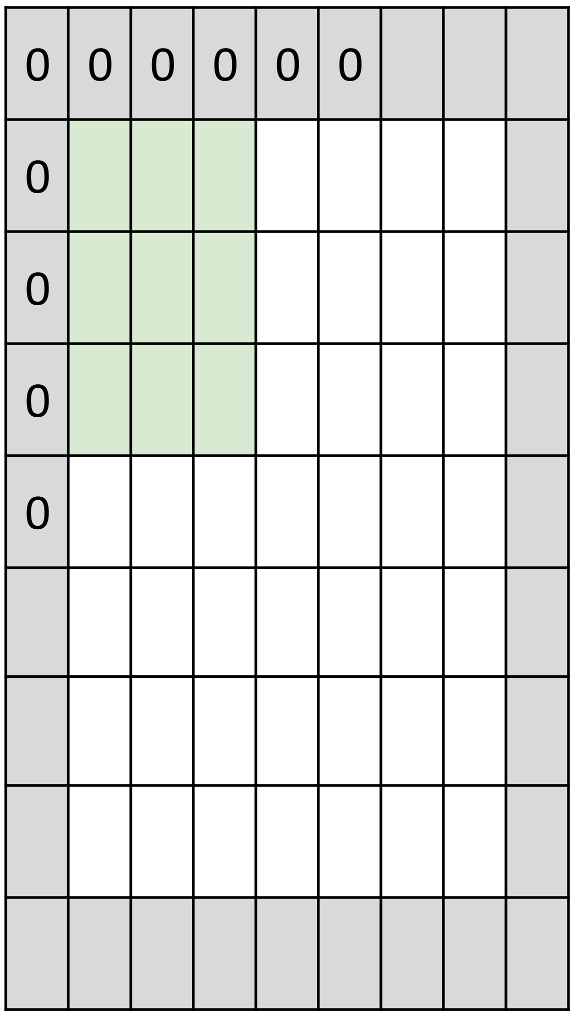
Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3

Source: Stanford CS231n,
Andrej Karpathy & Fei-Fei Li

filters: padding

- In practice: Common to zero pad the border



e.g. input 7x7
neuron with receptive field 3x3, stride 1
pad with 1 pixel border => what is the output?

7x7 => preserved size!

Source: Stanford CS231n,
Andrej Karpathy & Fei-Fei Li

Filters in practise

- **“Same convolution” (preserves size)**

Input [9x9]

3x3 neurons, stride 1, pad **1** => [9x9]

- No headaches when sizing architectures
- Works well

- **“Valid convolution” (shrinks size)**

Input [9x9]

3x3 neurons, stride 1, pad **0** => [7x7]

- **Headaches** with sizing the full architecture
- Works Worse!

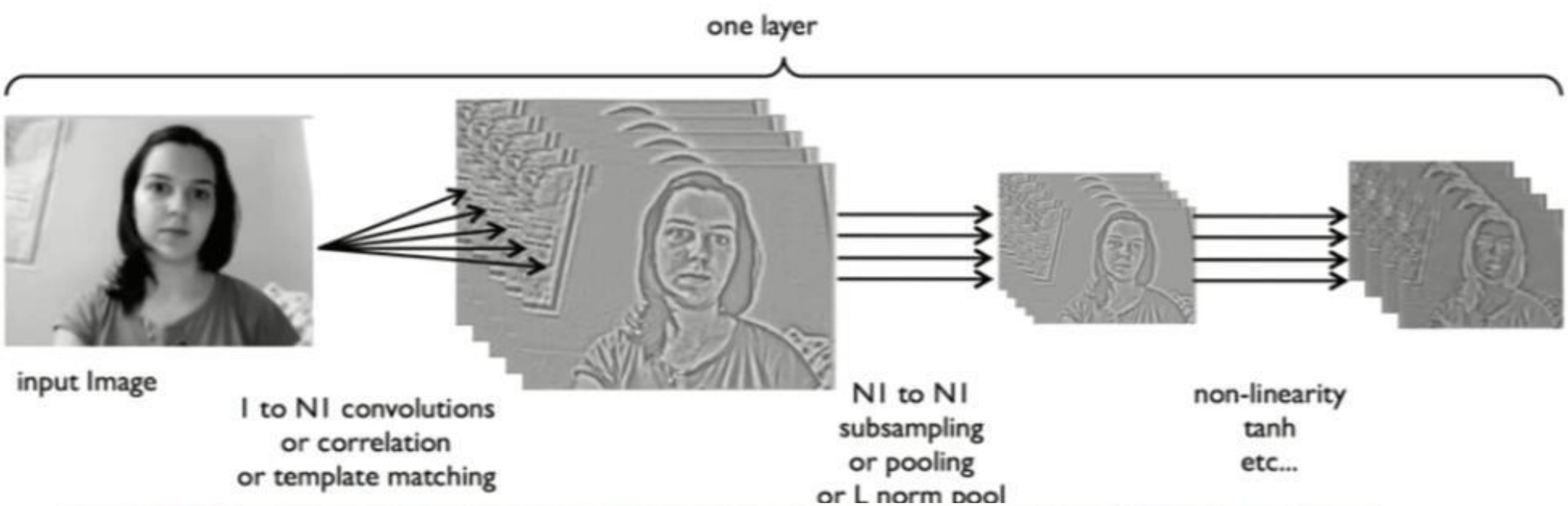
Source: Stanford CS231n, Andrej Karpathy & Fei-Fei Li

outline

- Modeling of CNN
  - Module-wise architecture模块化结构
- Convolutional layer (module)
  - Convolution in general
  - Filters
  - Convolution module
- Pooling layer (module)

Feature detection (特征检测)

- Learning filters (weights)

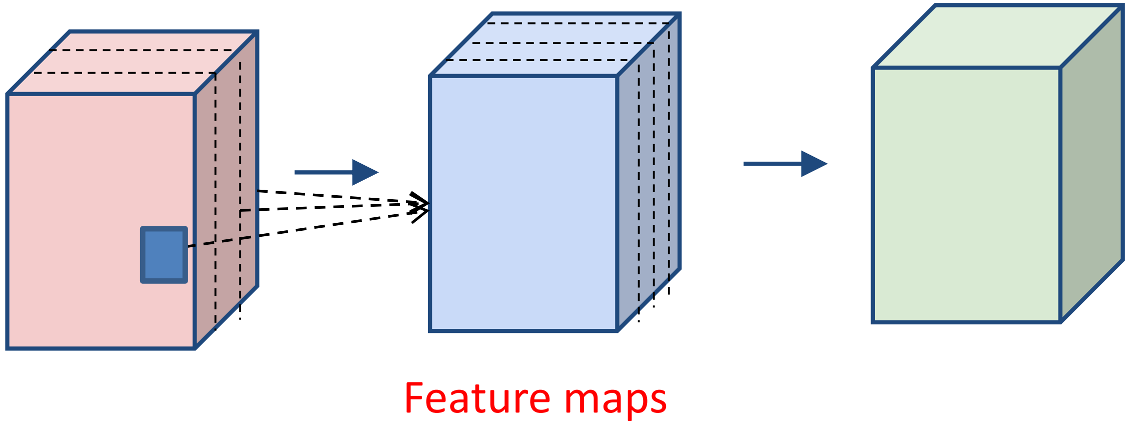


Convolution Layer (卷积层)

Input: X∈ $R^{d_{in} \times h \times w}$

weight: W∈ $R^{d_{out} \times d_{in} \times F_h \times F_w}$

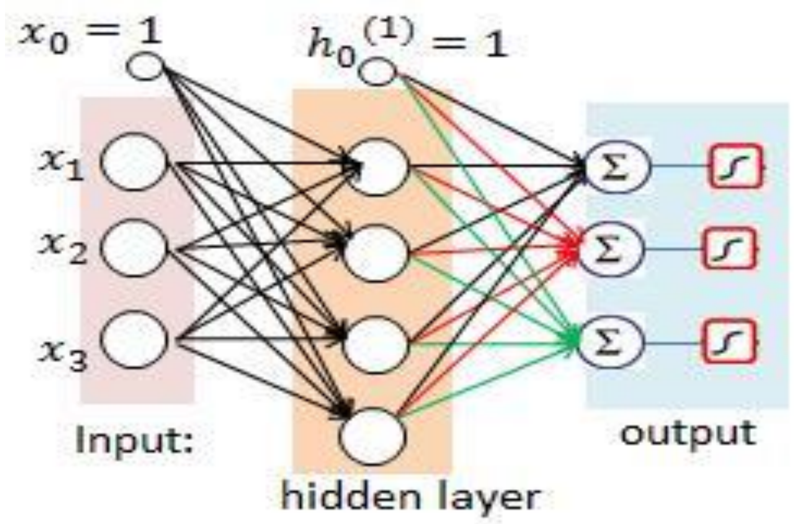
output: Y∈ $R^{d_{out} \times h \times w}$



Input: x∈ $R^{d_{in}}$

weight: W∈ $R^{d_{out} \times d_{in}}$

output: y∈ $R^{d_{out}}$

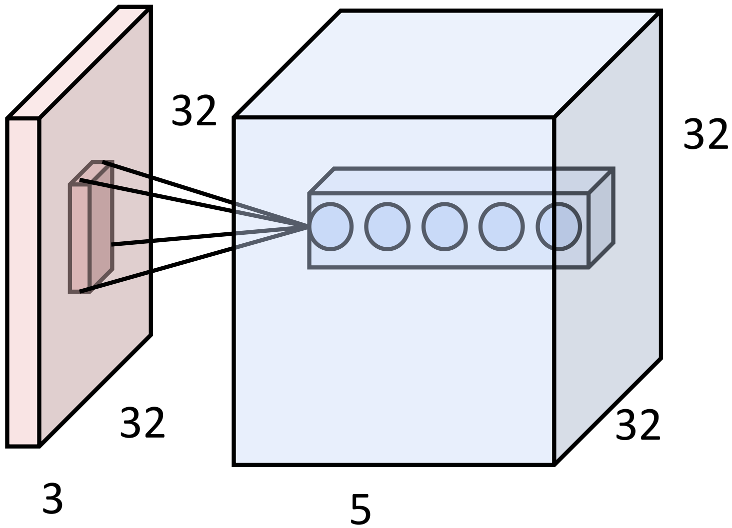


Forward (前向过程)

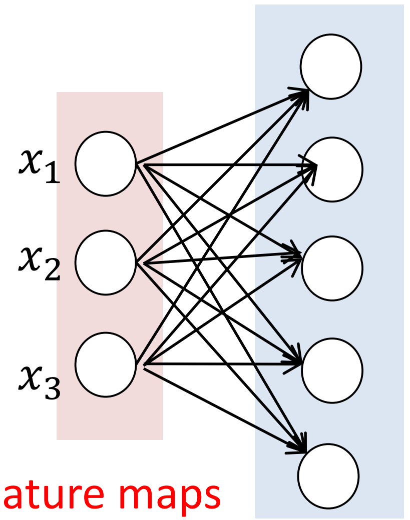
Input: $\mathbf{X} \in \boldsymbol{R}^{d_{in} \times h \times w}$

weight: $\mathbf{W} \in \boldsymbol{R}^{d_{out} \times d_{in} \times F_h \times F_w}$

output: $\mathbf{Y} \in \boldsymbol{R}^{d_{out} \times h \times w}$

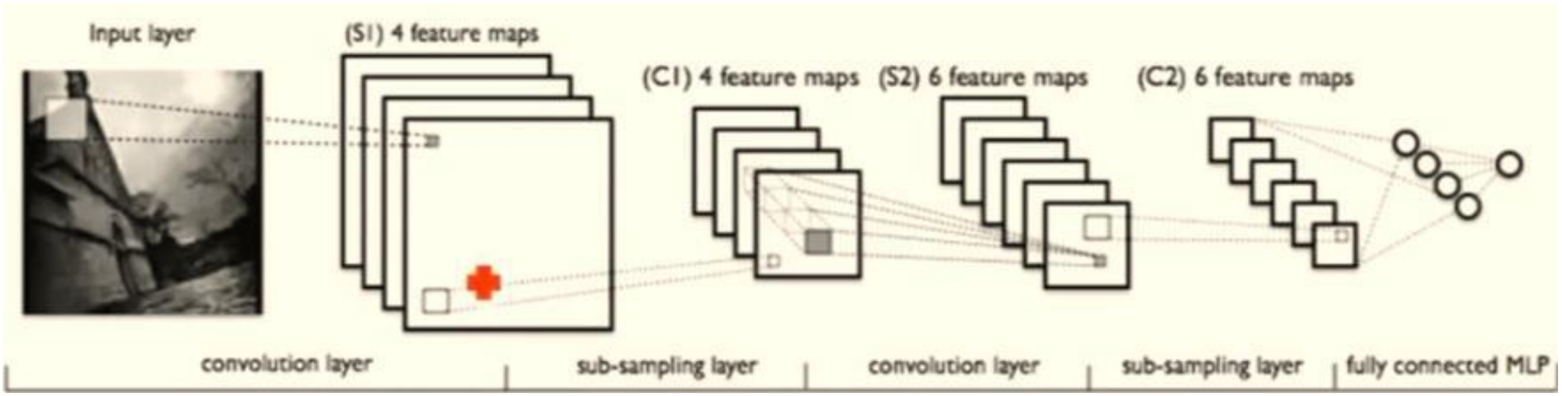


$$\boldsymbol{y}_{f',i',j'} = \sum_{f=1}^{d_{in}} \sum_{i=-\frac{F_h}{2}}^{\frac{F_h}{2}} \sum_{j=-\frac{F_w}{2}}^{\frac{F_w}{2}} \boldsymbol{x}_{f,i'+i-1,j'+j-1} \boldsymbol{w}_{f',f,i,j} + \boldsymbol{b}_{f'}$$



example

$$y_{4,10,10} = \sum_{i=-\frac{F_h}{2}}^{\frac{F_h}{2}} \sum_{j=-\frac{F_w}{2}}^{\frac{F_w}{2}} x_{1,10+i-1,10+j-1}\, w_{4,1,i,j} + b_4$$

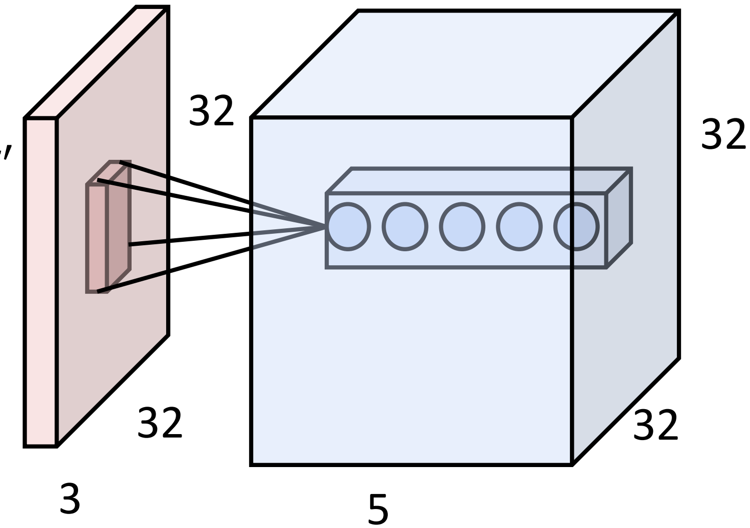


$$y_{4,10,100} = \sum_{i=-\frac{F_h}{2}}^{\frac{F_h}{2}} \sum_{j=-\frac{F_w}{2}}^{\frac{F_w}{2}} x_{1,10+i-1,100+j-1}\, w_{4,1,i,j} + \sum_{i=-\frac{F_h}{2}}^{\frac{F_h}{2}} \sum_{j=-\frac{F_w}{2}}^{\frac{F_w}{2}} x_{2,10+i-1,100+j-1}\, w_{4,2,i,j} +$$

$$\sum_{i=-\frac{F_h}{2}}^{\frac{F_h}{2}} \sum_{j=-\frac{F_w}{2}}^{\frac{F_w}{2}} x_{3,10+i-1,100+j-1}\, w_{4,3,i,j} + \sum_{i=-\frac{F_h}{2}}^{\frac{F_h}{2}} \sum_{j=-\frac{F_w}{2}}^{\frac{F_w}{2}} x_{4,10+i-1,100+j-1}\, w_{4,4,i,j} + b_4$$

Back-propagation

$$y_{f',i',j'} = \sum_{f=1}^{d_{in}} \sum_{i=-\frac{F_h}{2}}^{\frac{F_h}{2}} \sum_{j=-\frac{F_w}{2}}^{\frac{F_w}{2}} x_{f,i'+i-1,j'+j-1}\, w_{f',f,i,j} + b_{f'}$$



$$\frac{dL}{dx_{f,i,j}} = \sum_{f'=1}^{d_{out}} \sum_{i'=1}^{h} \sum_{j'=1}^{w} \frac{dL}{dy_{f',i',j'}} \frac{dy_{f',i',j'}}{dx_{f,i,j}}$$

$$\sum_{f'=1}^{d_{out}} \sum_{i'=1}^{h} \sum_{j'=1}^{w} \frac{dL}{dy_{f',i',j'}} w_{f',f,i-i'+1,j-j'+1}$$

Input: $\mathbf{X} \in \boldsymbol{R}^{d_{in} \times h \times w}$

$$\frac{dL}{w_{f',f,i,j}} = \sum_{f'=1}^{d_{out}} \sum_{i'=1}^{h} \sum_{j'=1}^{w} \frac{dL}{dy_{f',i',j'}} \frac{dy_{f',i',j'}}{w_{f',f,i,j}}$$

$$\sum_{f'=1}^{d_{out}} \sum_{i'=1}^{h} \sum_{j'=1}^{w} \frac{dL}{dy_{f',i',j'}} x_{f,i'+i-1,j'+j-1}$$

weight: $\mathbf{W} \in \boldsymbol{R}^{d_{out} \times d_{in} \times F_h \times F_w}$

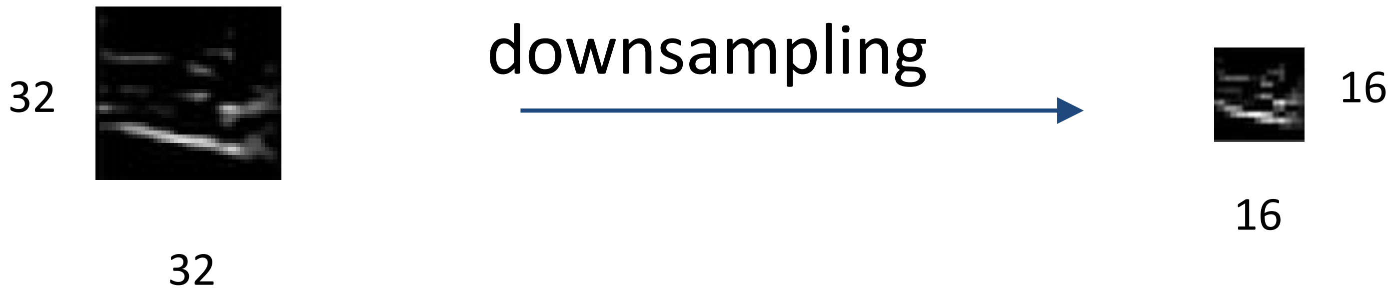
output: $\mathbf{Y} \in \boldsymbol{R}^{d_{out} \times h \times w}$

outline

- Modeling of CNN
  - Module-wise architecture模块化结构
- Convolutional layer (module)
  - Convolution in general
  - Filters
  - Convolution module
- Pooling layer (module)

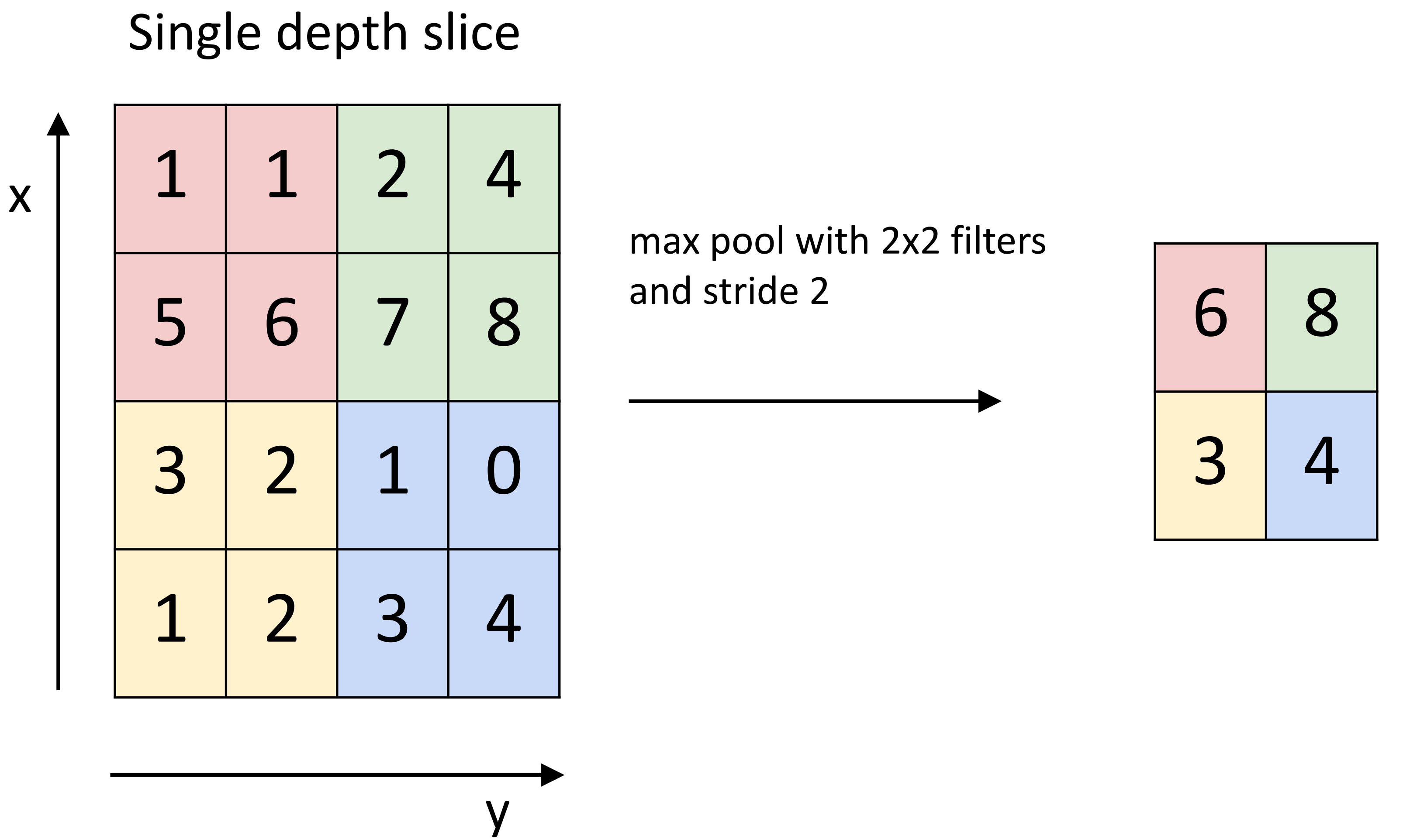
POOLING Layer

- In ConvNet architectures, **Conv** layers are often followed by **Pooling** layers
  - makes the representations smaller and more manageable without losing too much information.
  - Invariant in region.



Source: Stanford CS231n,
Andrej Karpathy & Fei-Fei Li

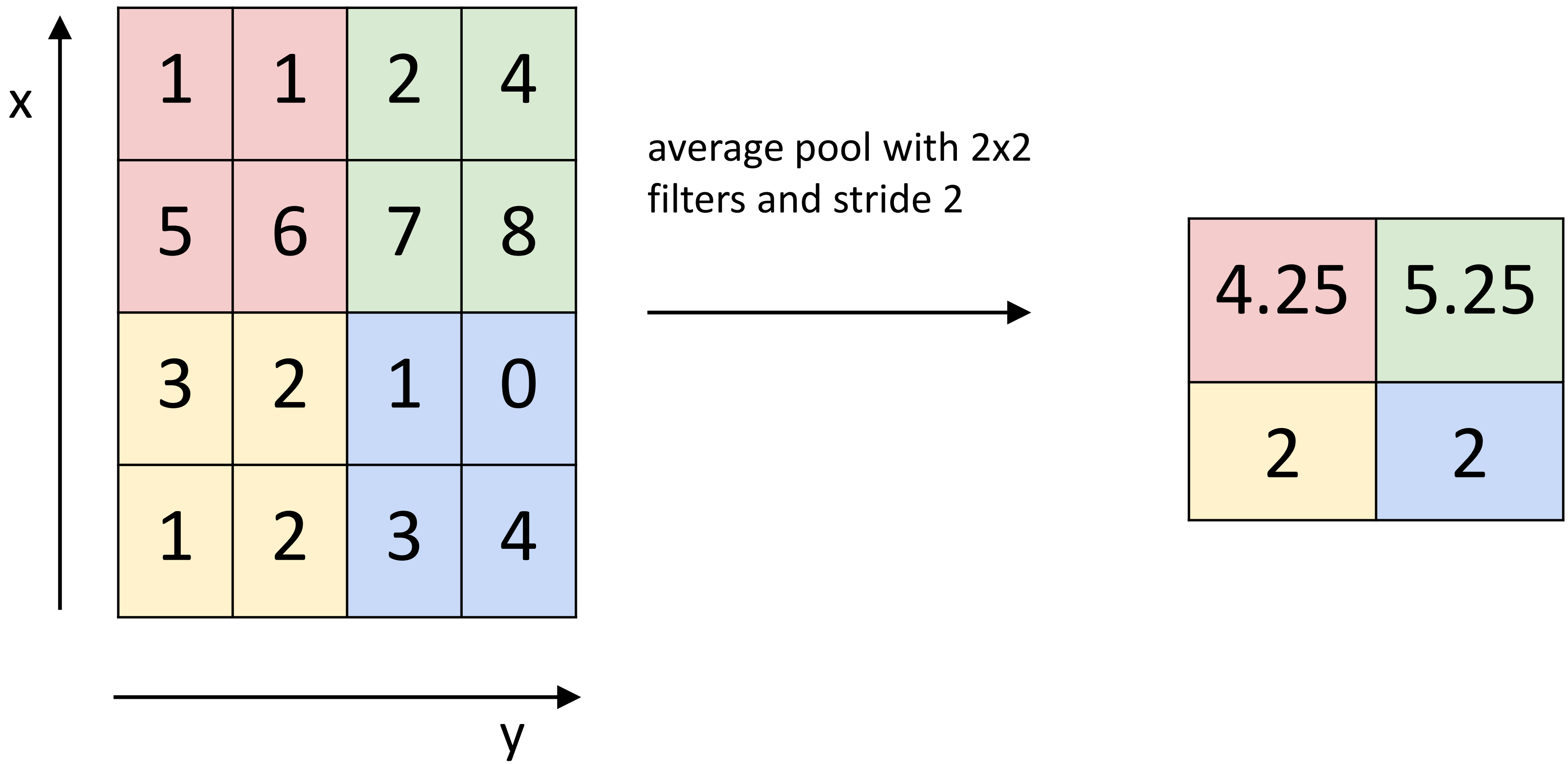
MAX POOLING



Source: Stanford CS231n,
Andrej Karpathy & Fei-Fei Li

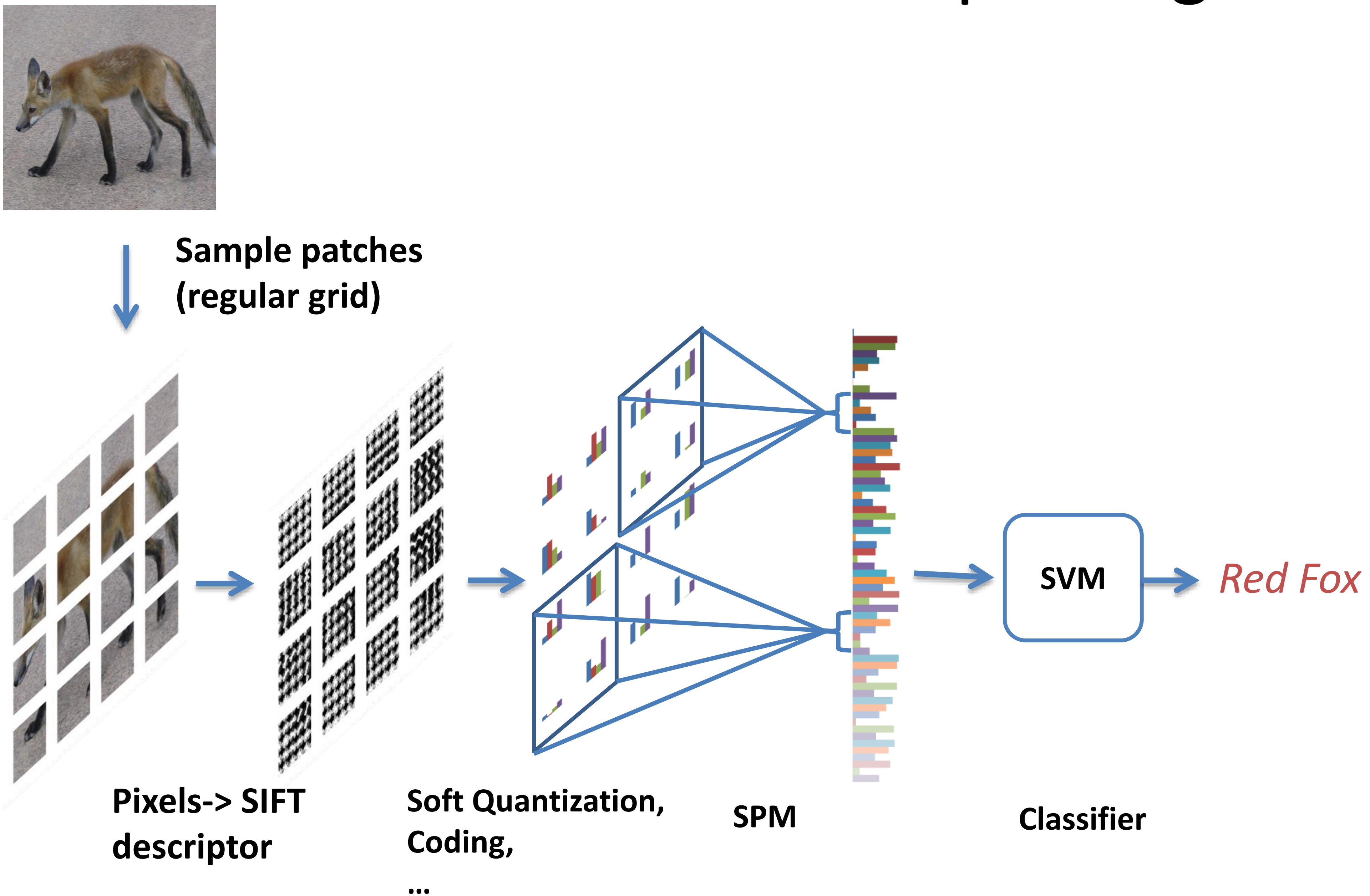
Average POOLING

Single depth slice

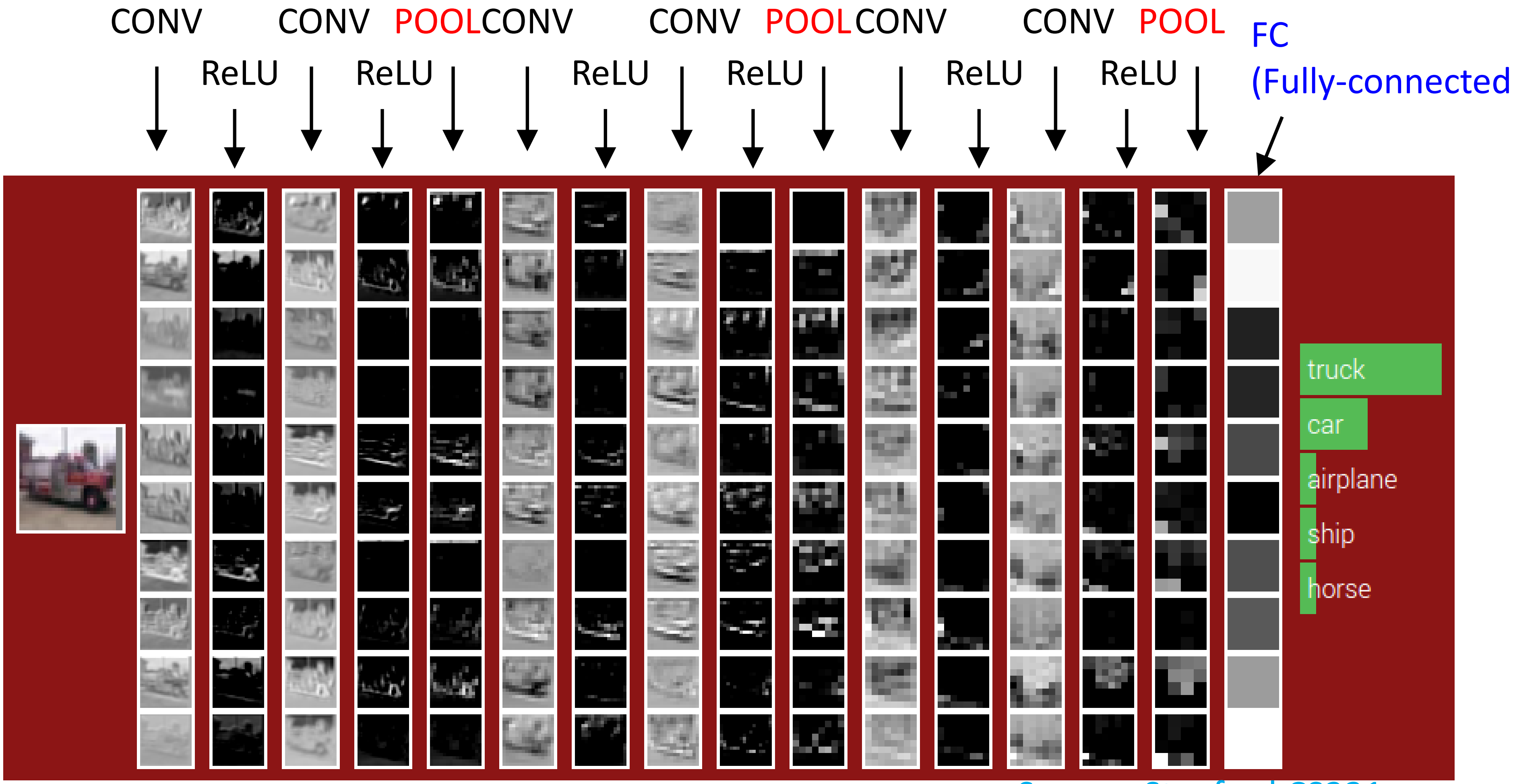


Source: Stanford CS231n,
Andrej Karpathy & Fei-Fei Li

Another Motivation of pooling



Intuitive example

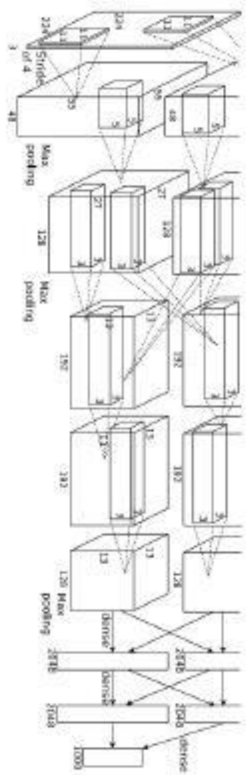


Source: Stanford CS231n,
Andrej Karpathy & Fei-Fei Li

Famous Net Architecture

Year 2012

SuperVision



[Krizhevsky NIPS 2012]

Year 2014

GoogLeNet

[Szegedy arxiv 2014]

VGG

[Simonyan arxiv 2014]

Year 2015

MSRA