Deep Learning
Part II

Yin Li

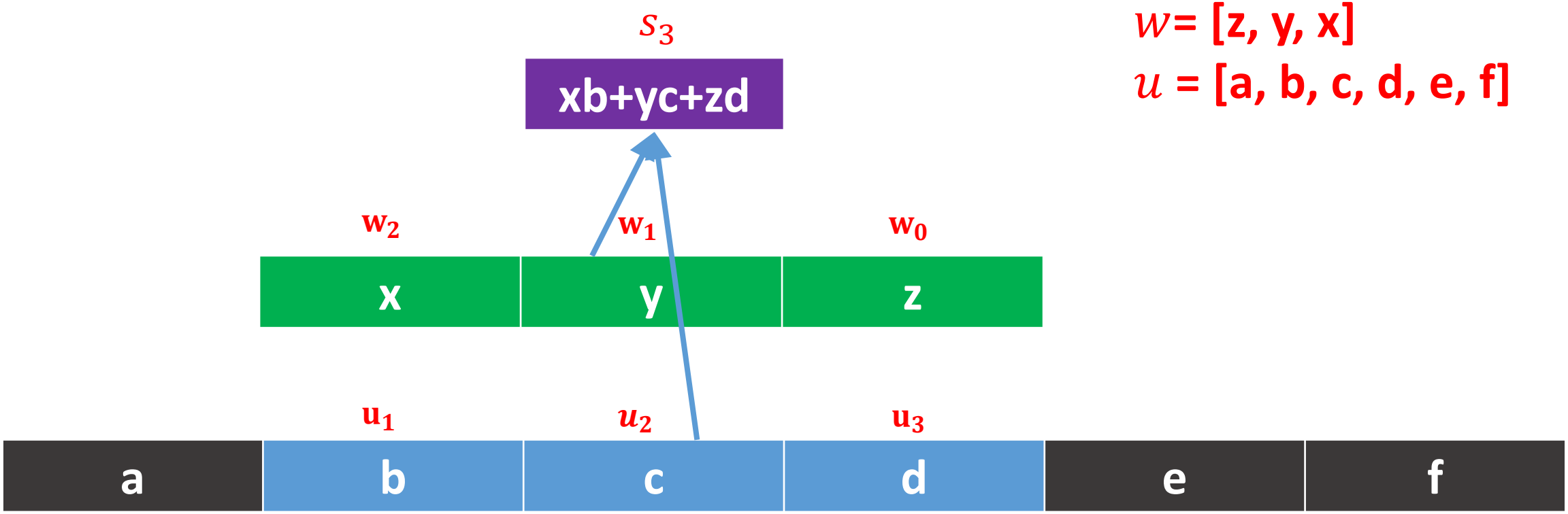
yin.li@wisc.edu

University of Wisconsin, Madison

Some of the slides from Yingyu Liang, Marc'Aurelio Ranzato and others

Convolution (continued)

Illustration 1



w= [z, y, x]

u = [a, b, c, d, e, f]

Illustration 1

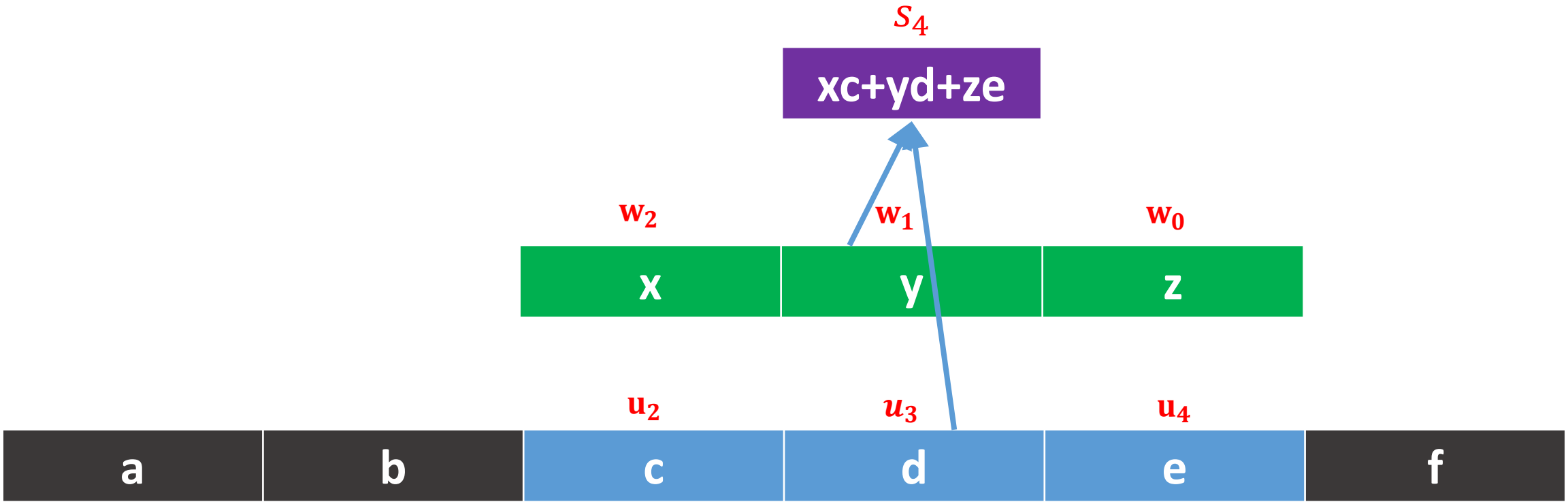


Illustration 1

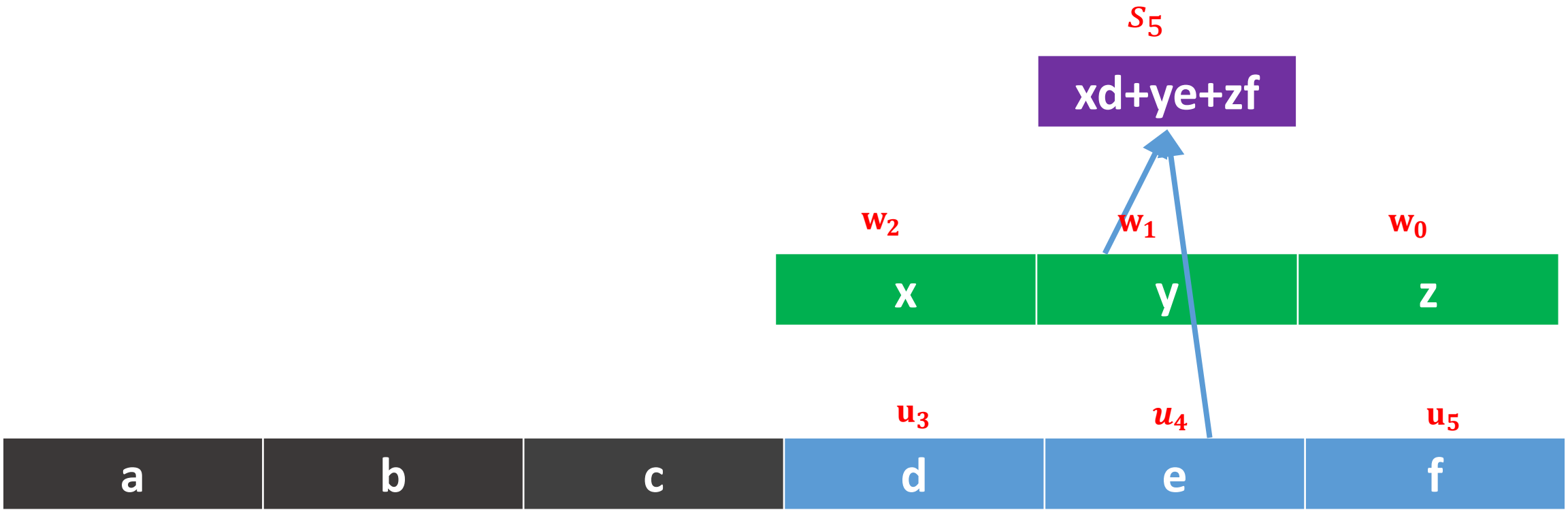
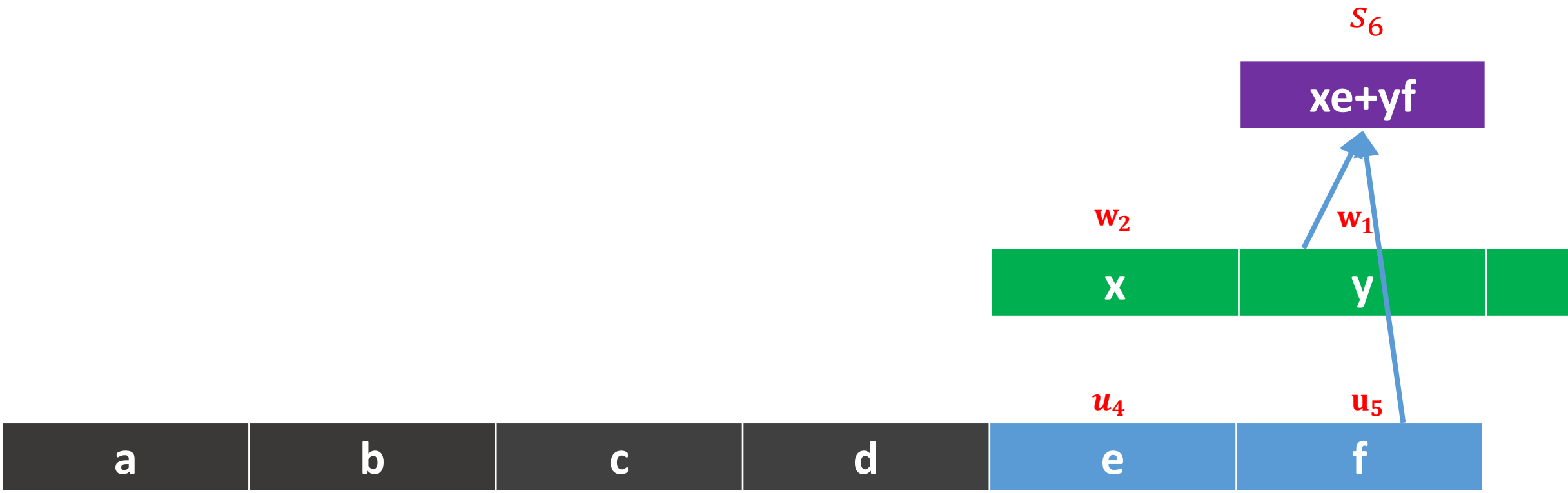


Illustration 1: boundary case



Gradient of convolution

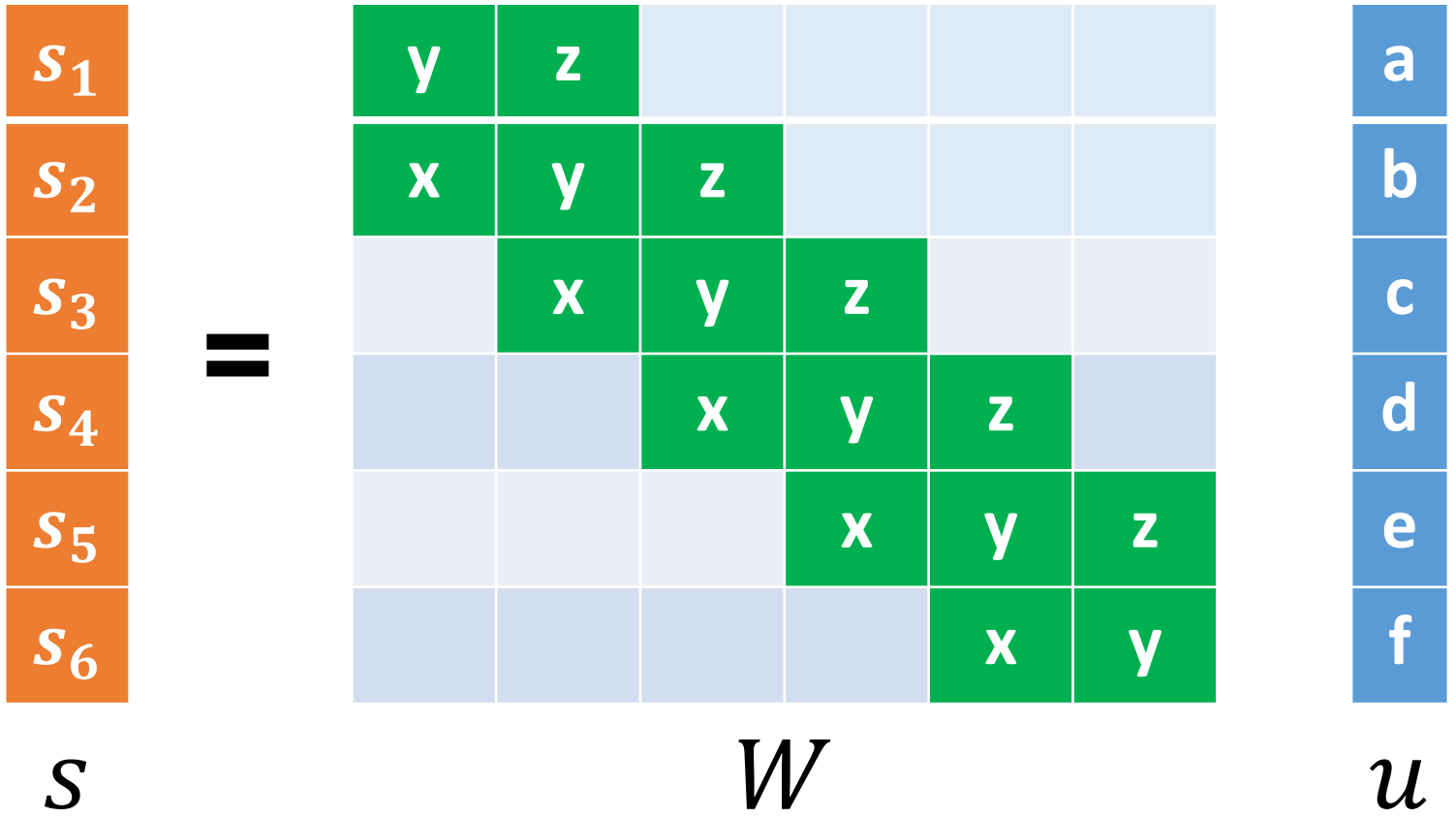
- Convolution $s = (u * w)$

- We will need to compute the gradient for back-propagation
  - Gradient with respect to the input u
  - Gradient with respect to the filter / kernel w

Gradient of convolution

w= **[z, y, x]** u = **[a, b, c, d, e, f]** $s = u * w$

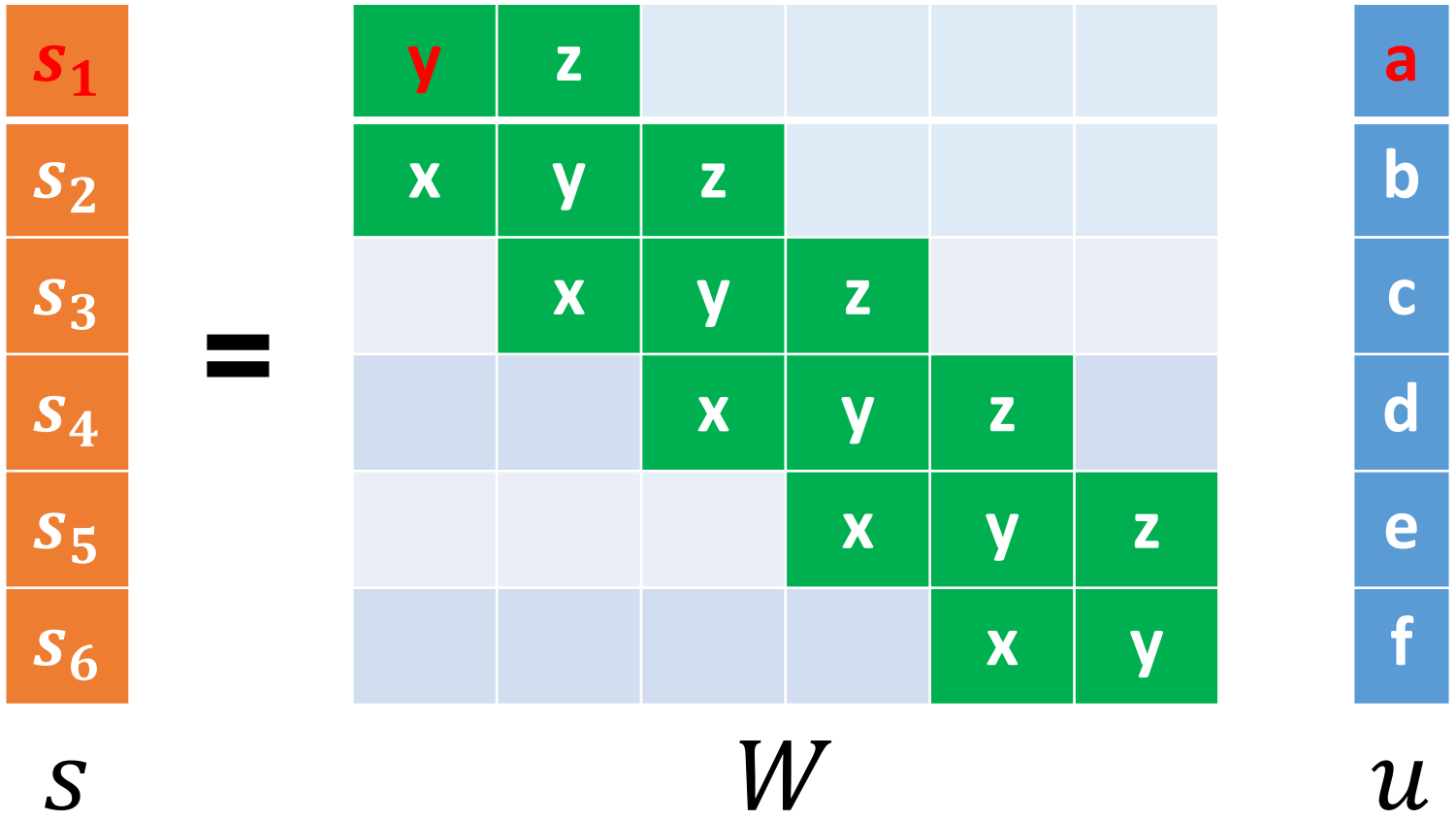
$$\frac{\partial s}{\partial u} = ?$$



Gradient of convolution

w= **[z, y, x]** u = **[a, b, c, d, e, f]** $s = u * w$

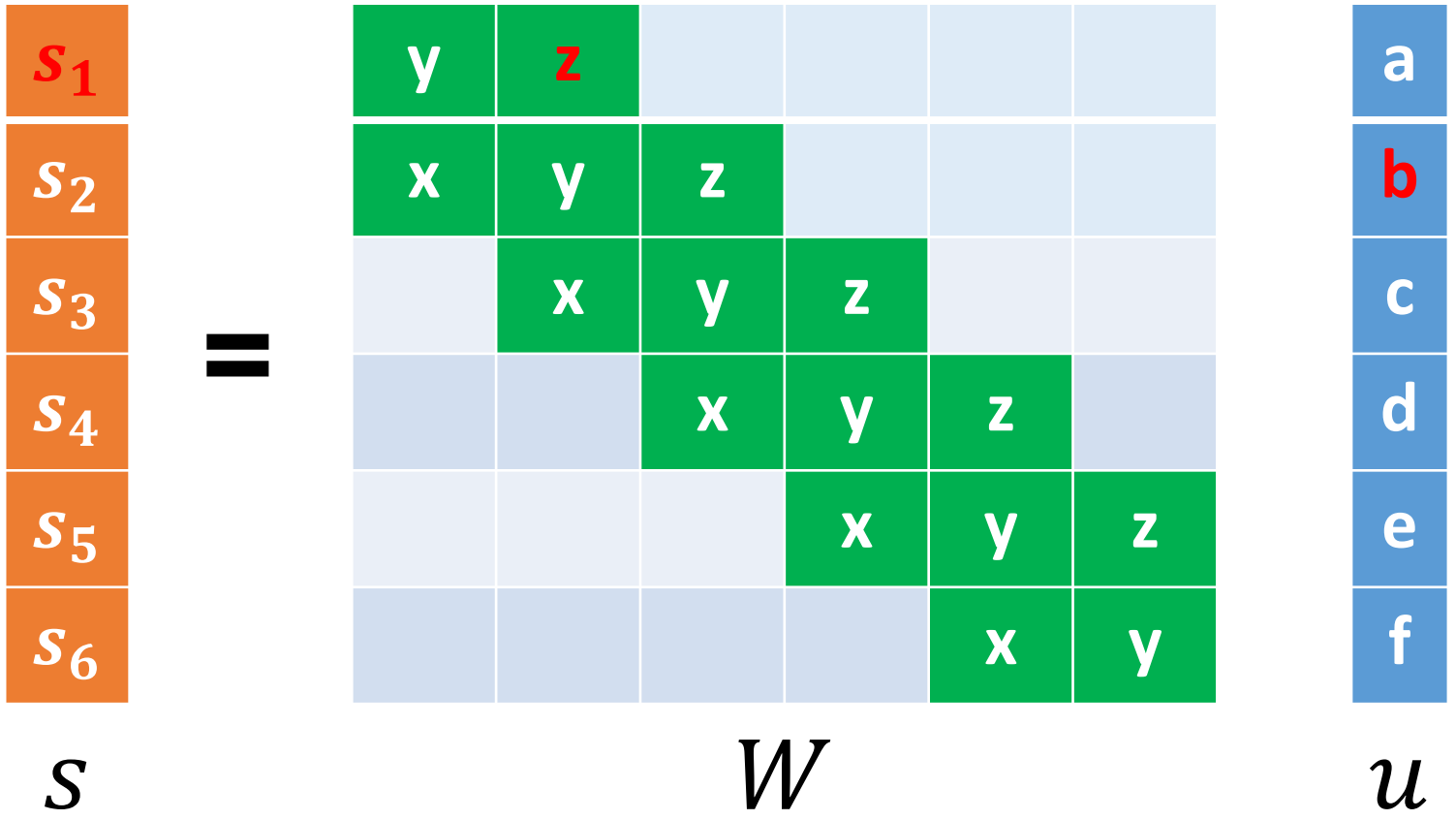
$$\frac{\partial s_1}{\partial u_1} = y$$



Gradient of convolution

w= **[z, y, x]** u = **[a, b, c, d, e, f]** $s = u * w$

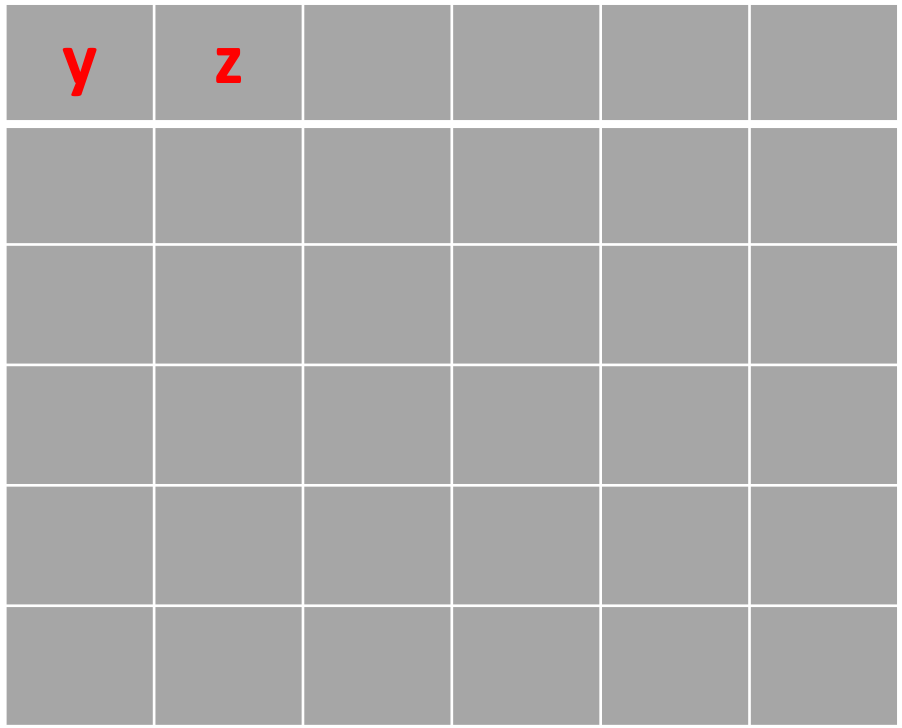
$$\frac{\partial s_1}{\partial u_2} = z$$



Gradient of convolution

w= **[z, y, x]** u = **[a, b, c, d, e, f]** $s = u * w$

$$\frac{\partial s_1}{\partial u} = \left[\frac{\partial s_1}{\partial u_1}, \dots \frac{\partial s_1}{\partial u_6}\right]$$



$$\frac{\partial s}{\partial u}$$

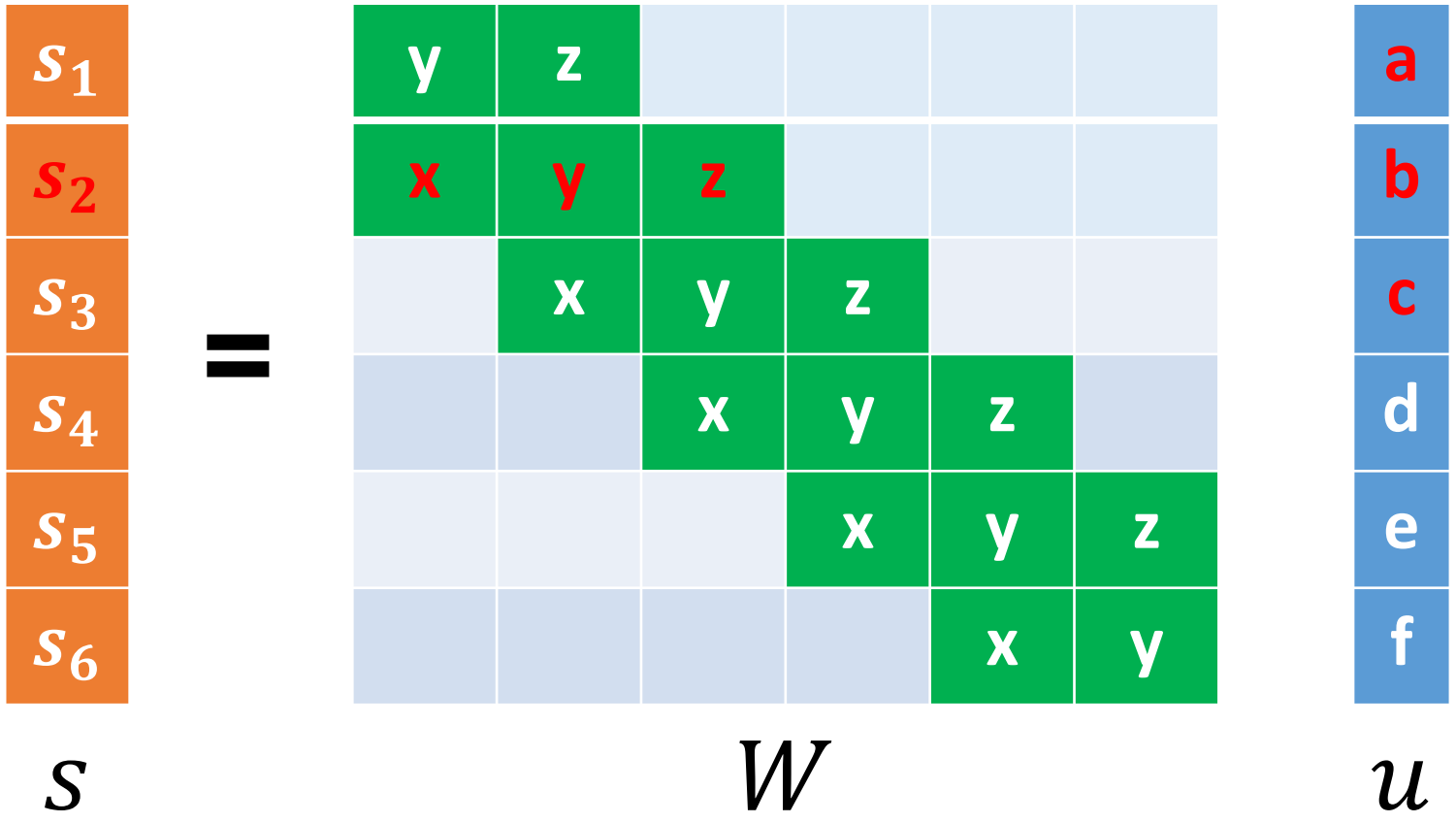
Gradient of convolution

w= **[z, y, x]** u = **[a, b, c, d, e, f]** $s = u * w$

$$\frac{\partial s_2}{\partial u_1} = x$$

$$\frac{\partial s_2}{\partial u_2} = y$$

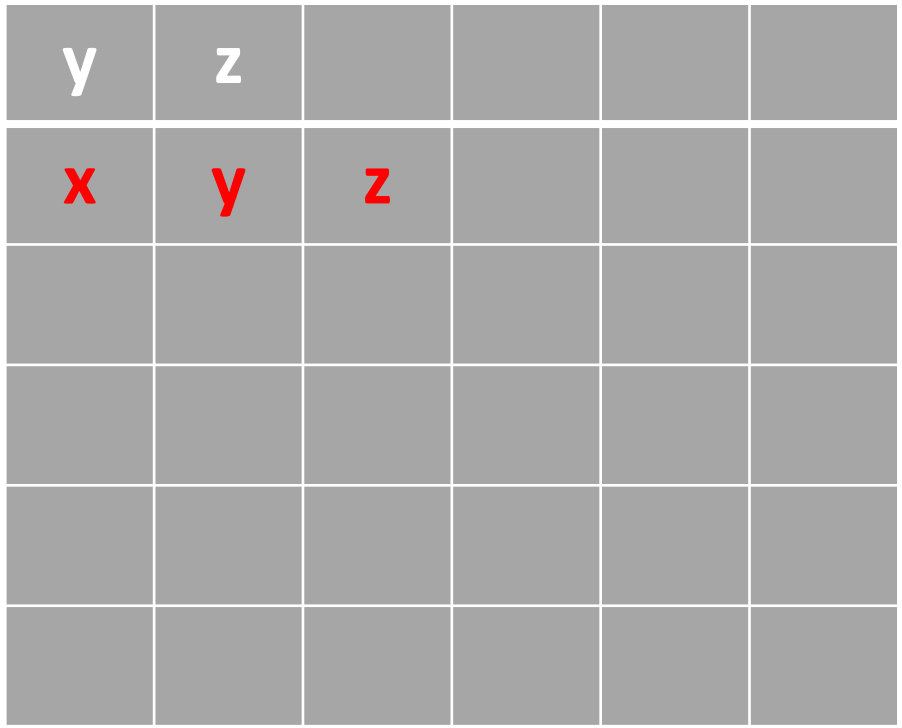
$$\frac{\partial s_2}{\partial u_3} = z$$



Gradient of convolution

w= **[z, y, x]** u = **[a, b, c, d, e, f]** $s = u * w$

$$\frac{\partial s_2}{\partial u} = \left[\frac{\partial s_2}{\partial u_1}, \dots \frac{\partial s_2}{\partial u_6}\right]$$



$$\frac{\partial s}{\partial u}$$

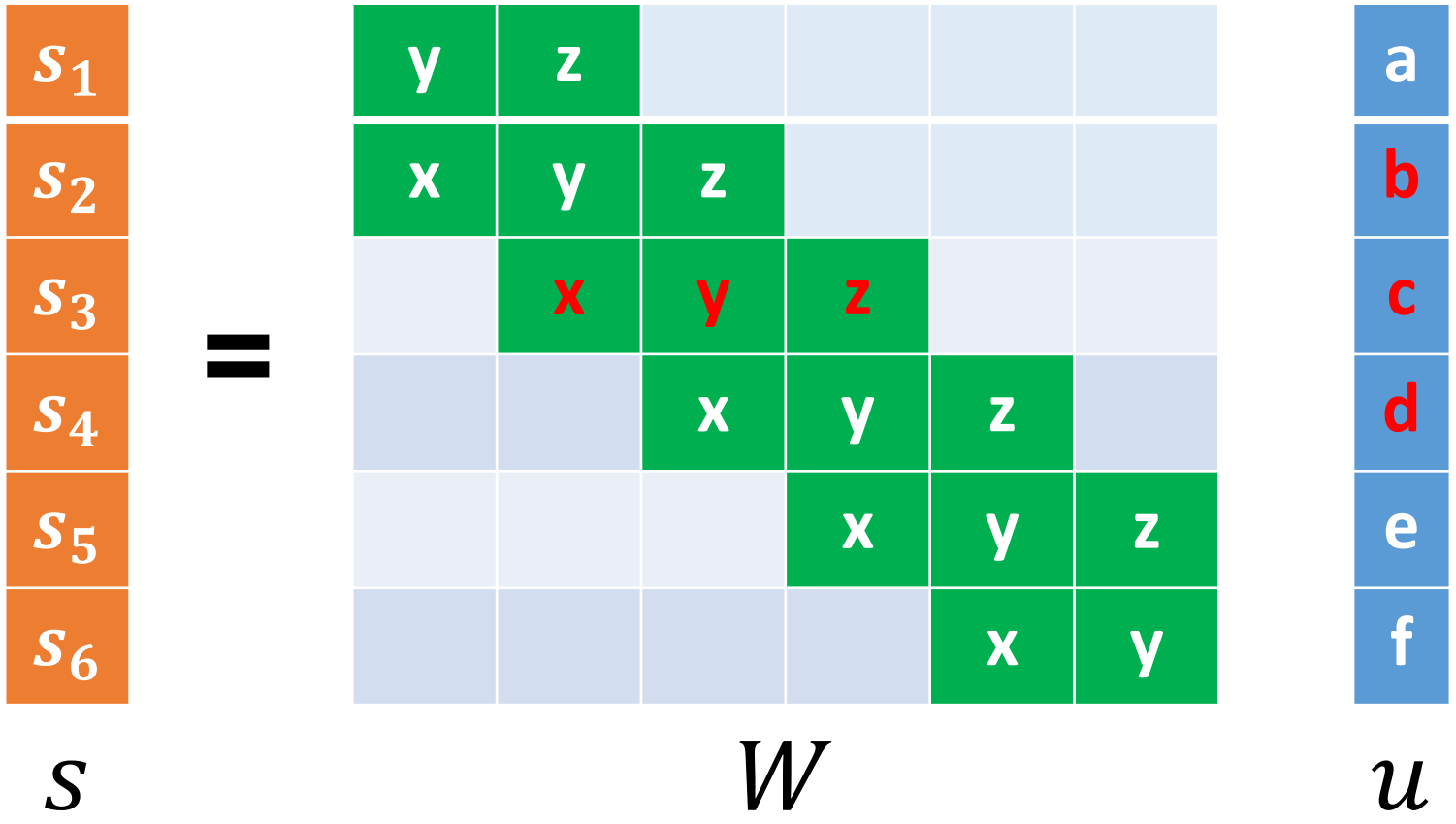
Gradient of convolution

w= **[z, y, x]** u = **[a, b, c, d, e, f]** $s = u * w$

$$\frac{\partial s_3}{\partial u_2} = x$$

$$\frac{\partial s_3}{\partial u_3} = y$$

$$\frac{\partial s_3}{\partial u_4} = z$$



Gradient of convolution

w= **[z, y, x]** u = **[a, b, c, d, e, f]** $s = u * w$

$$\frac{\partial s_3}{\partial u} = \left[\frac{\partial s_3}{\partial u_1}, \dots \frac{\partial s_3}{\partial u_6}\right]$$

y	z				
x	y	z			
	x	y	z		

$$\frac{\partial s}{\partial u}$$

Gradient of convolution

w= **[z, y, x]** u = **[a, b, c, d, e, f]** $s = u * w$

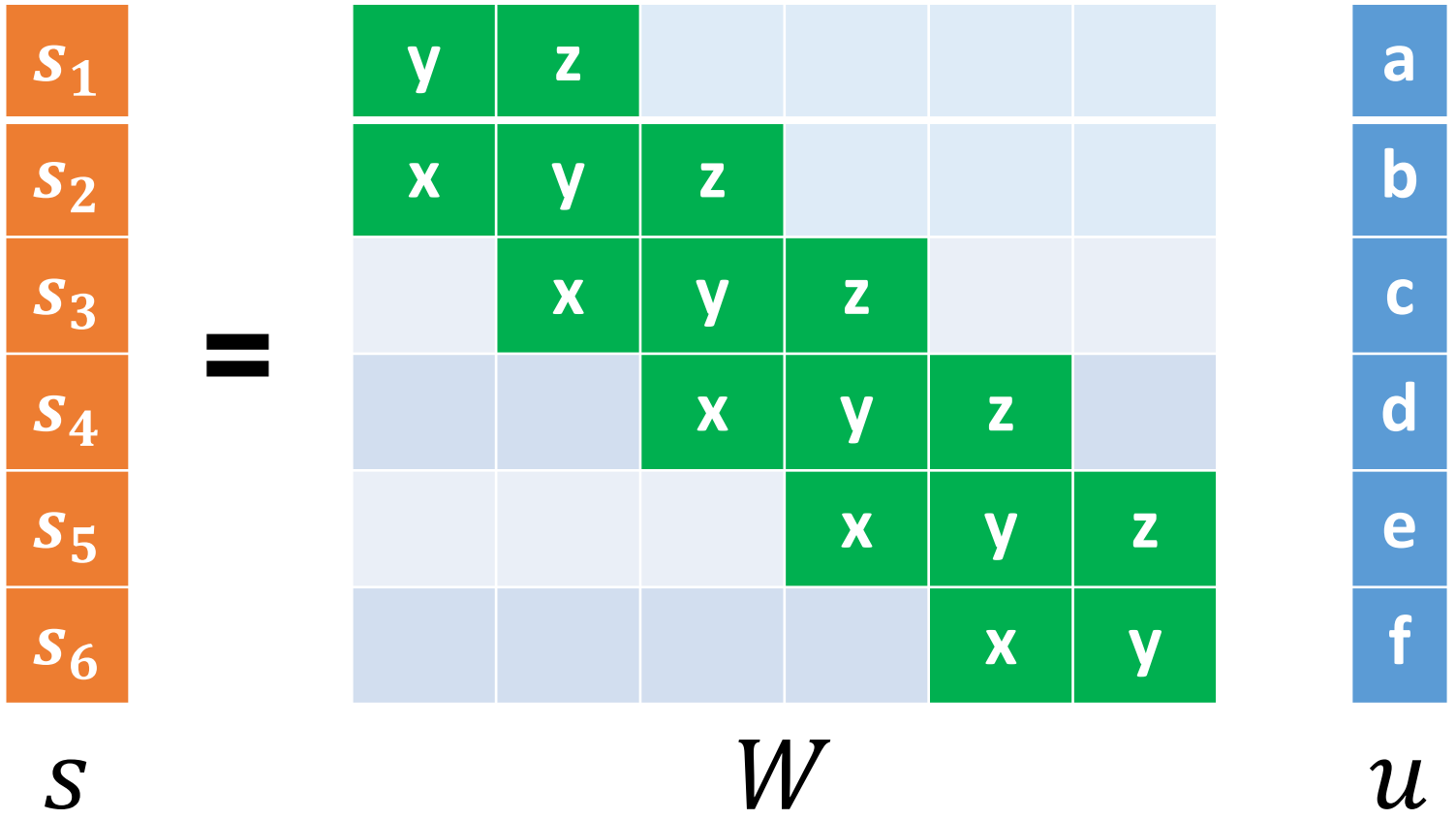
$$\frac{\partial s}{\partial u} = W$$

y	z				
x	y	z			
	x	y	z		
		x	y	z	
			x	y	z
				x	y

Gradient of convolution

w= **[z, y, x]** u = **[a, b, c, d, e, f]** $s = u * w$

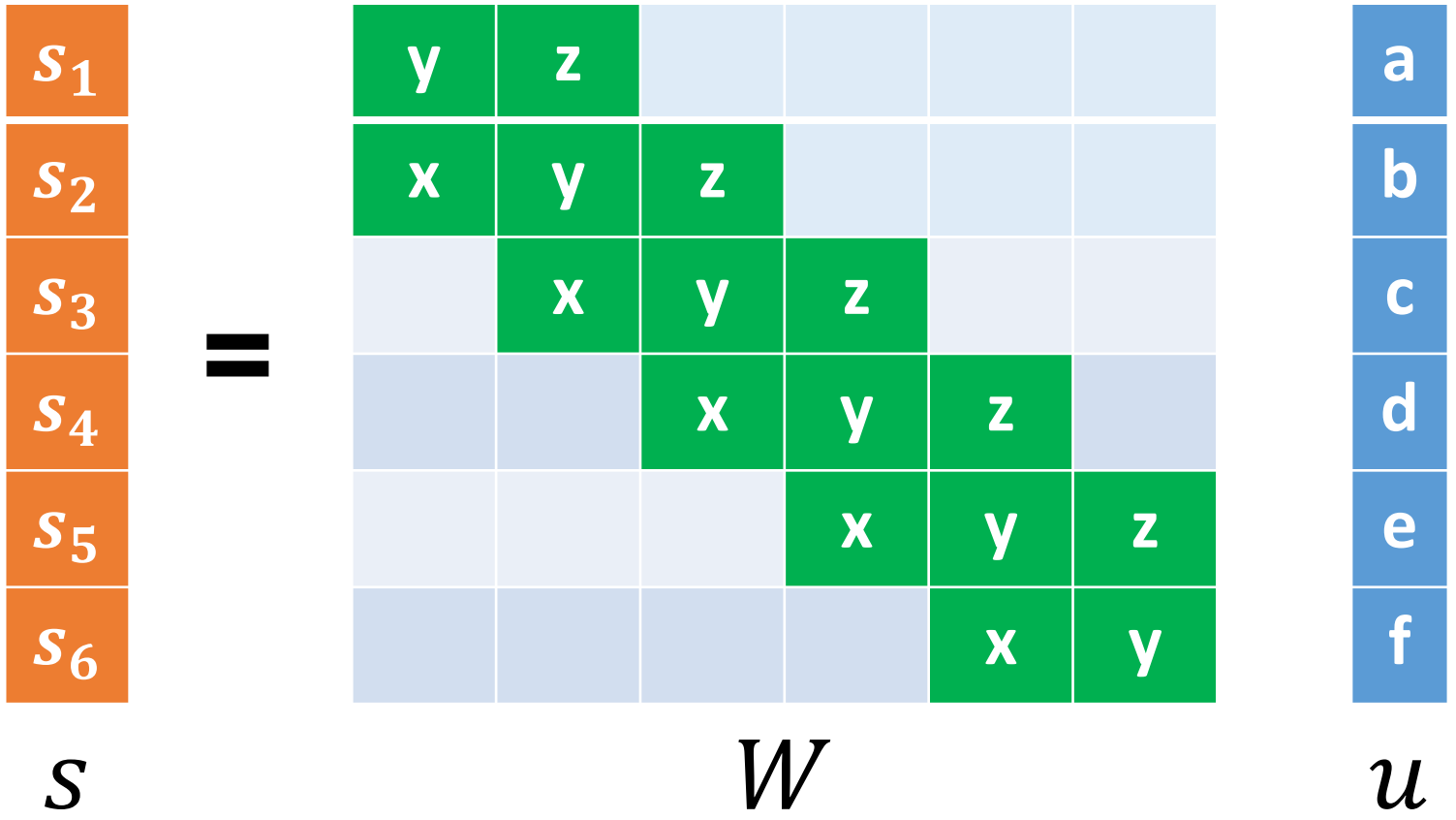
$$s = u * w$$
$$= Wu$$
$$\frac{\partial s}{\partial u} = W$$



Gradient of convolution

w= **[z, y, x]** u = **[a, b, c, d, e, f]** $s = u * w$

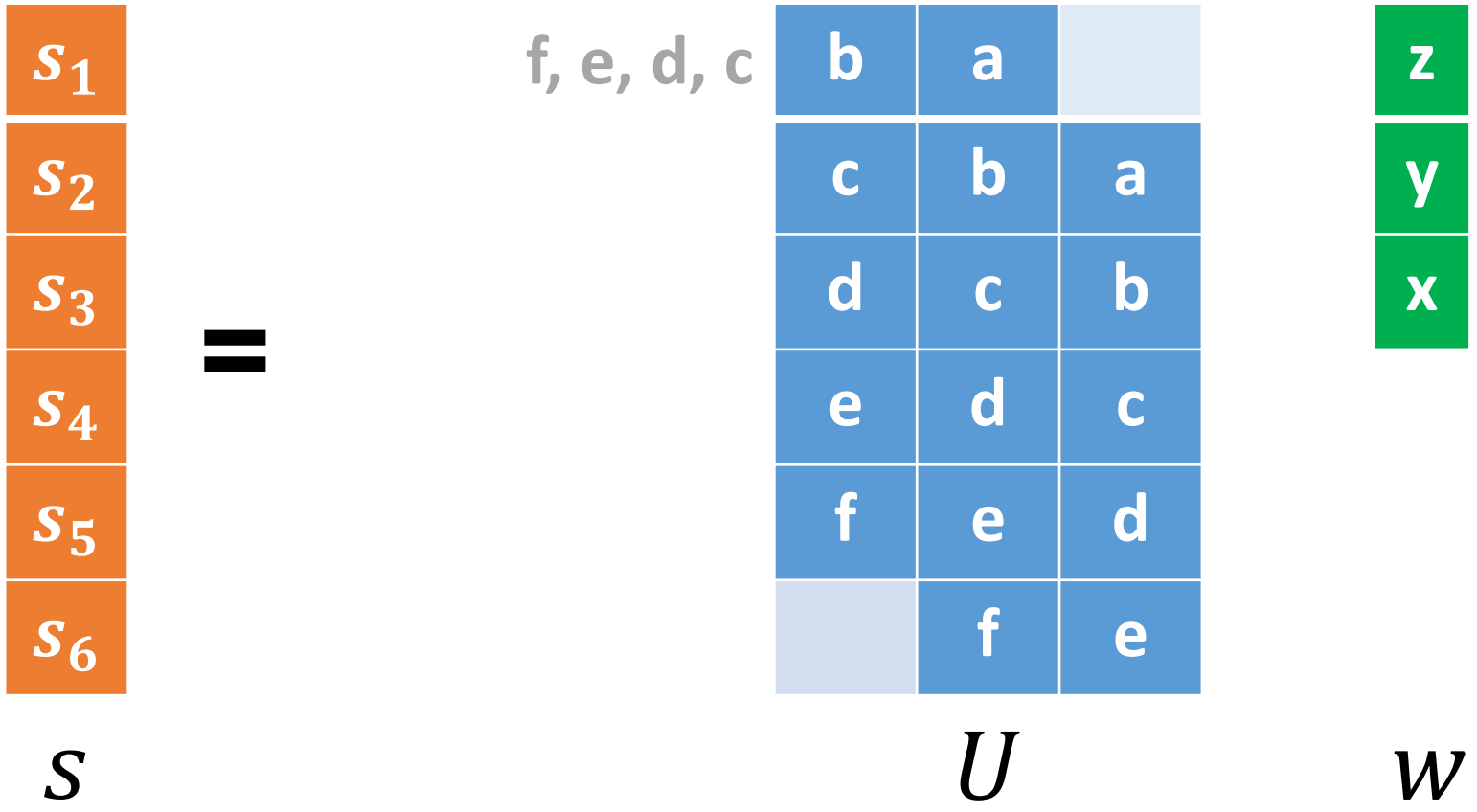
$$\frac{\partial s}{\partial w} = ?$$



Gradient of convolution

w= **[z, y, x]** u = **[a, b, c, d, e, f]** $s = w * u$

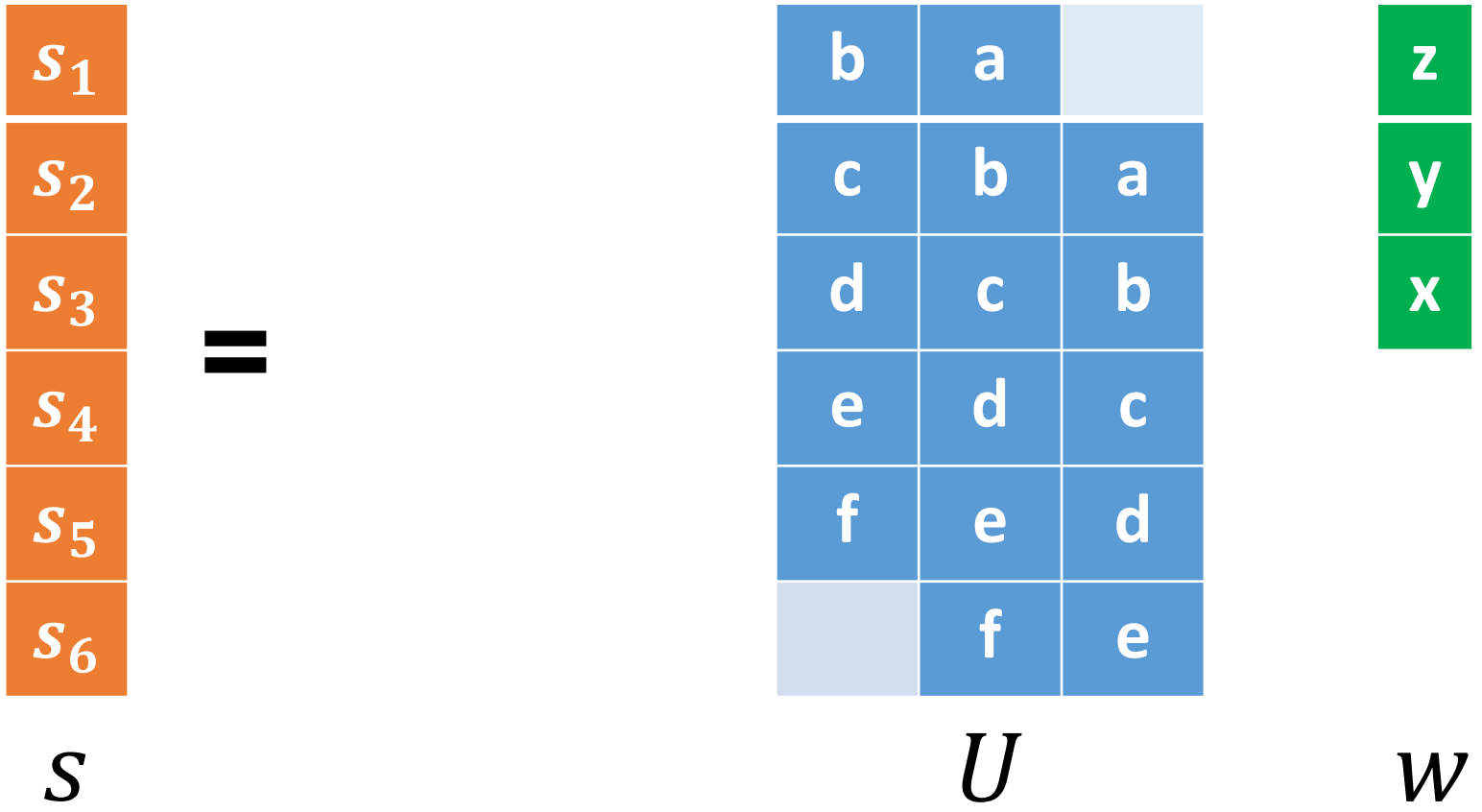
$$\frac{\partial s}{\partial w} = ?$$



Gradient of convolution

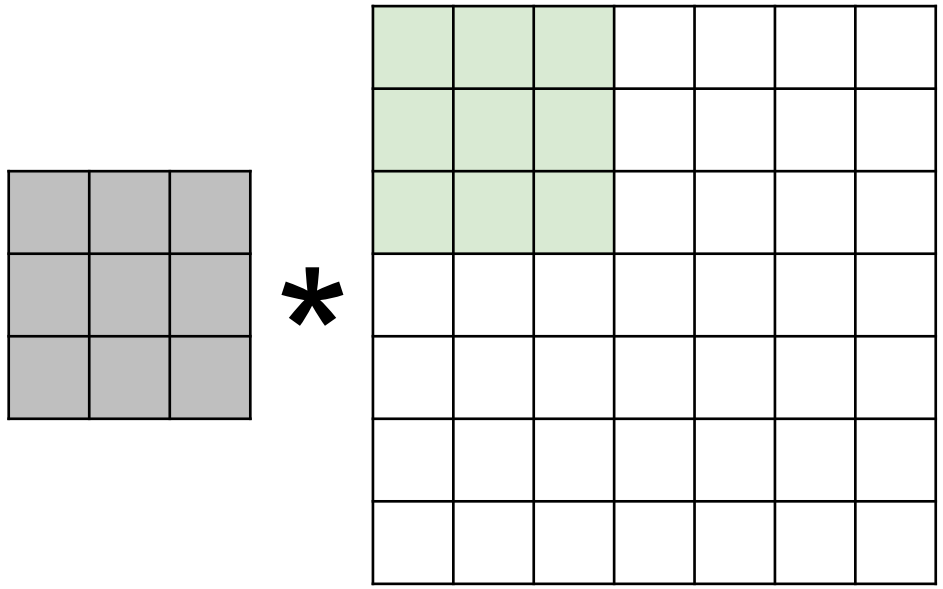
w= **[z, y, x]** u = **[a, b, c, d, e, f]** $s = u * w$

$$\frac{\partial s}{\partial w} = U$$



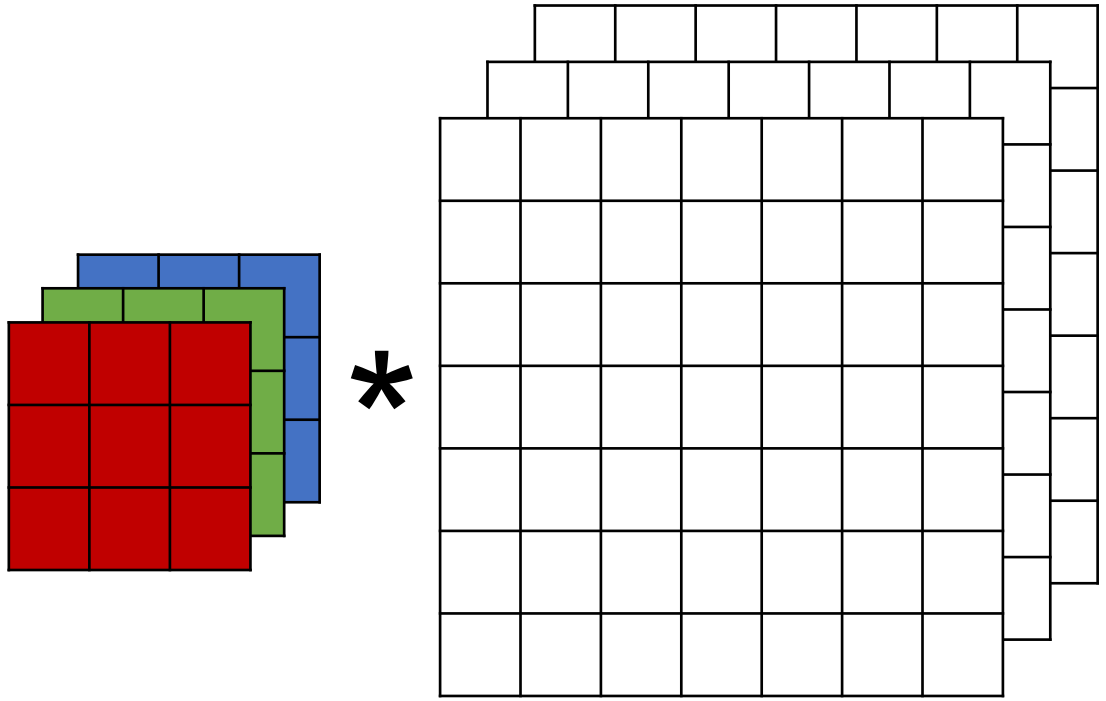
Convolution with high dimensional data

- Input and kernel can be 1D / 2D



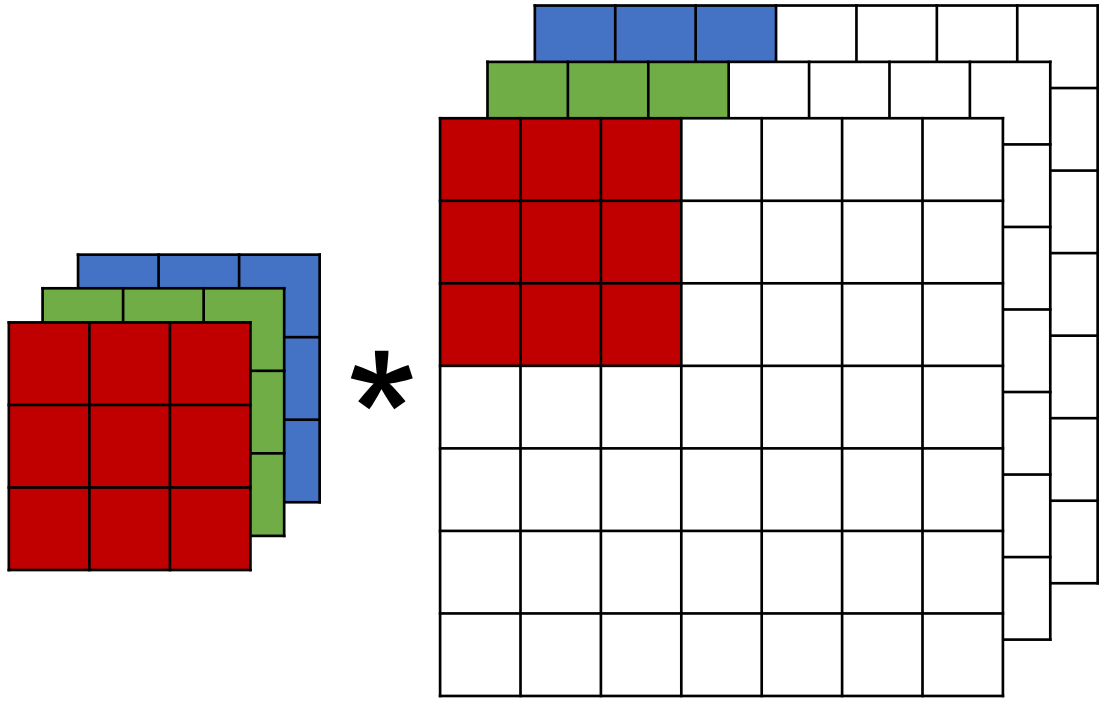
Convolution with high dimensional data

- Input and kernel can be 3D, e.g., an RGB image have 3 channels



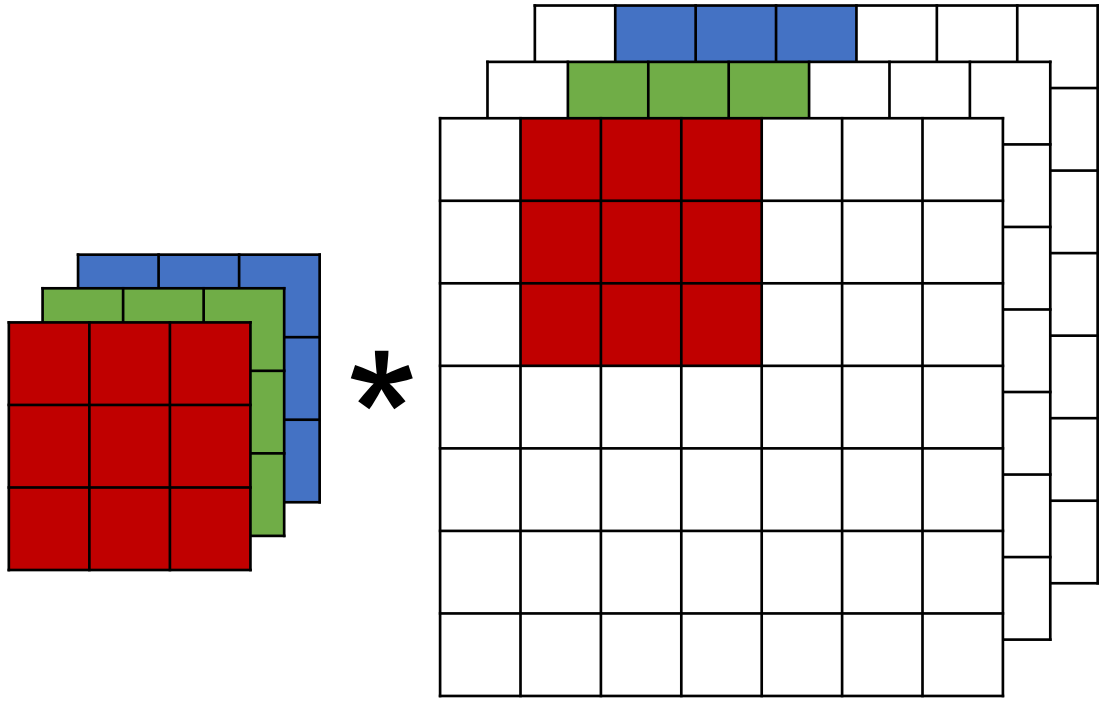
Convolution with high dimensional data

- Input and kernel can be 3D, e.g., an RGB image have 3 channels



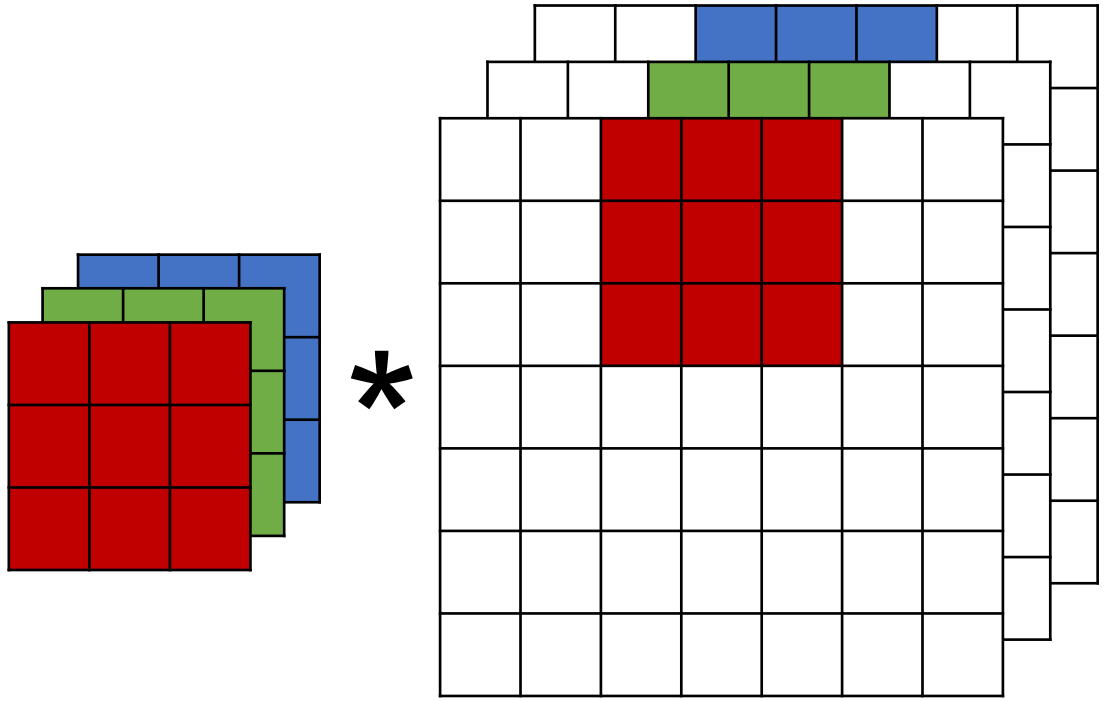
Convolution with high dimensional data

- Input and kernel can be 3D, e.g., an RGB image have 3 channels



Convolution with high dimensional data

- Input and kernel can be 3D, e.g., an RGB image have 3 channels



Pooling

Pooling

- Summarizing the input (i.e., output the max of the input)

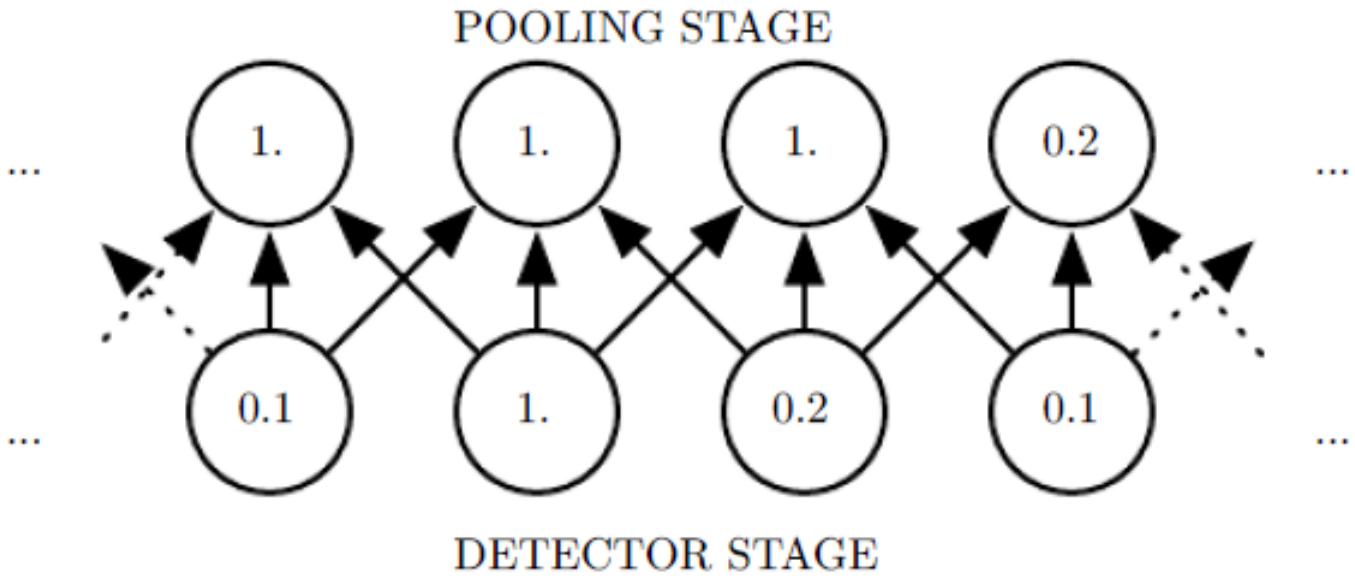
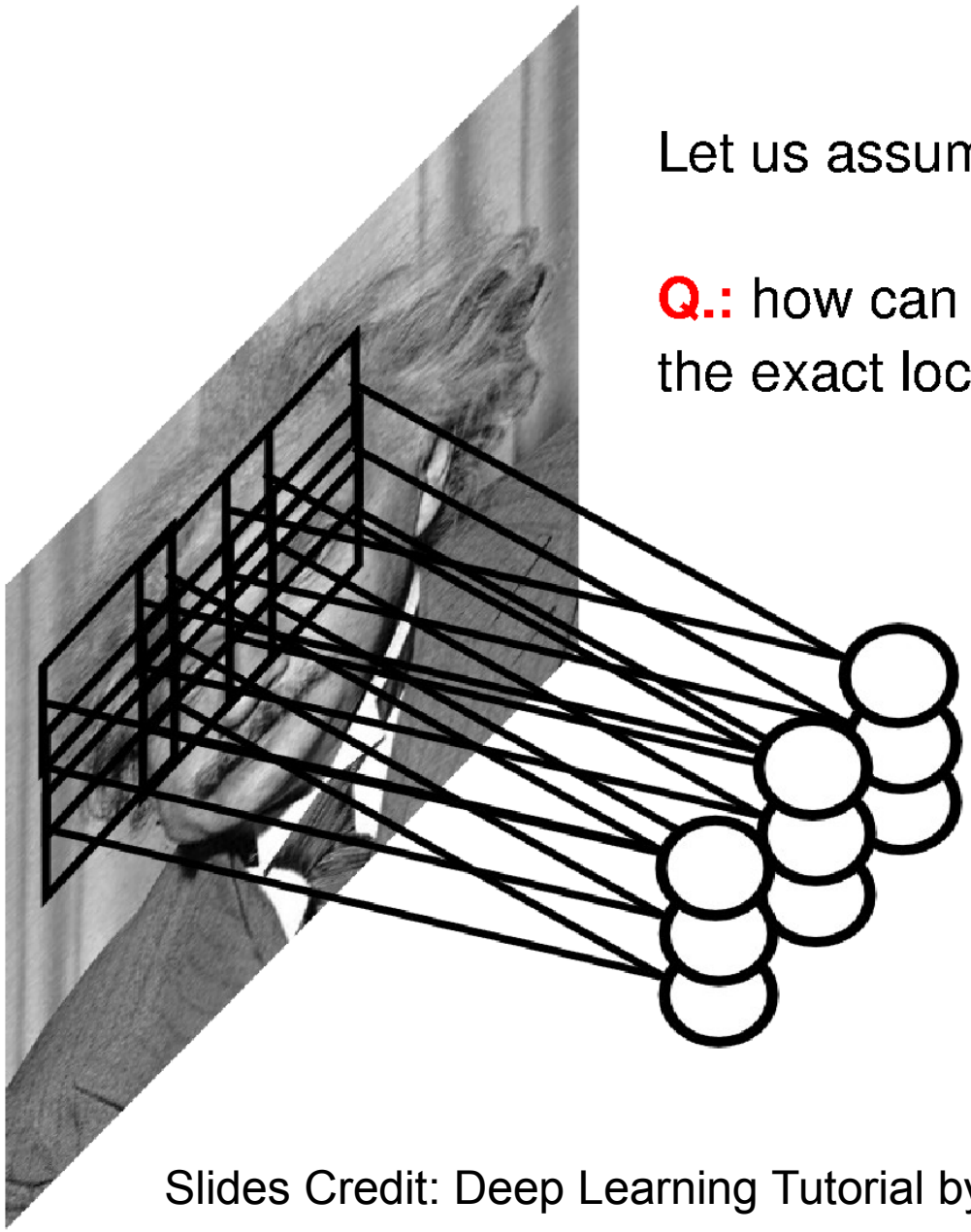


Figure from *Deep Learning,* by Goodfellow, Bengio, and Courville

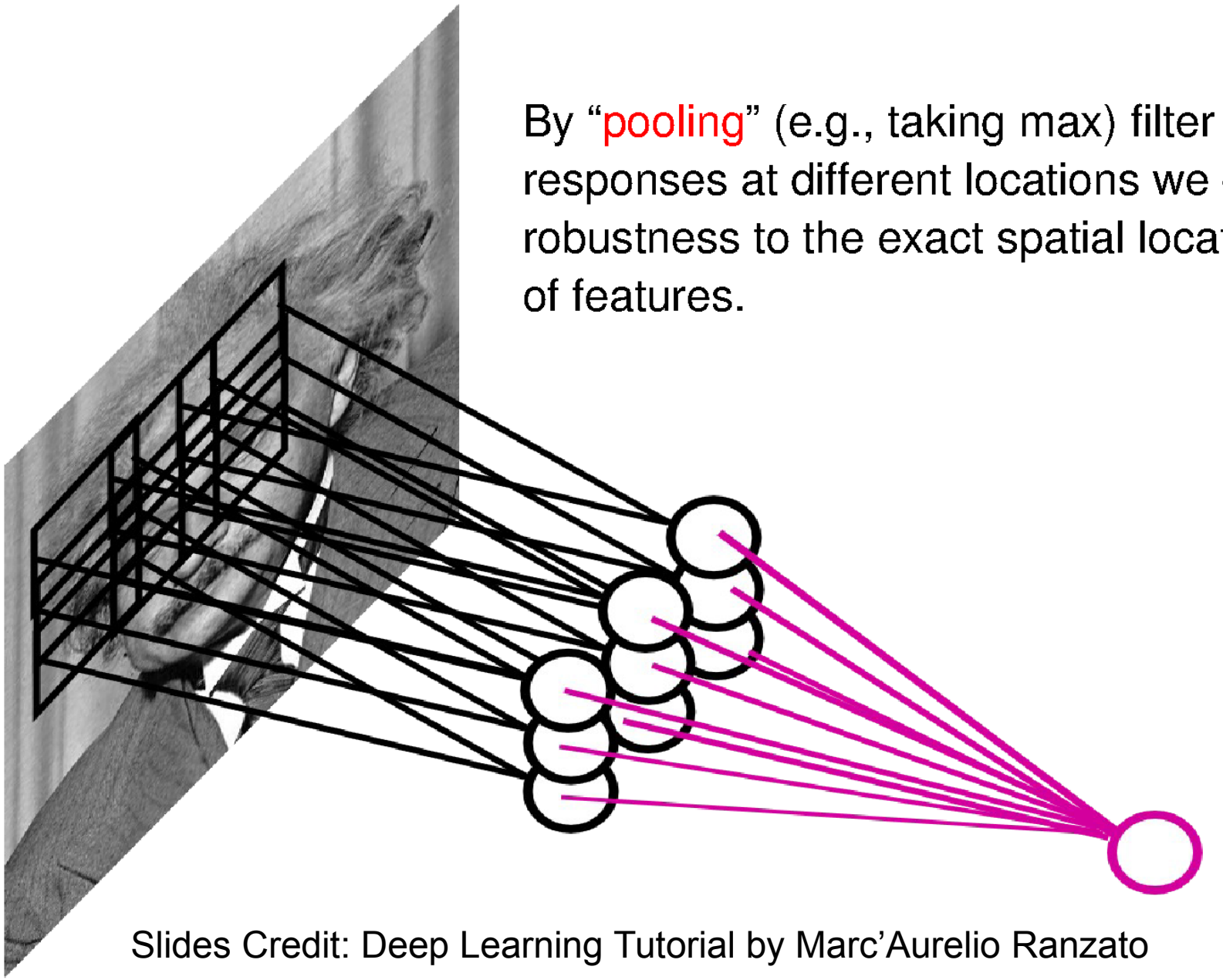


Let us assume filter is an “eye” detector.

Q.: how can we make the detection robust to the exact location of the eye?

Slides Credit: Deep Learning Tutorial by Marc’Aurelio Ranzato

Ranzato



By “pooling” (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

Slides Credit: Deep Learning Tutorial by Marc’Aurelio Ranzato

Ranzato

Motivation from neuroscience

- David Hubel and Torsten Wiesel studied early visual system in human brain (V1 or primary visual cortex), and won Nobel prize for this

- V1 properties
  - 2D spatial arrangement
  - Simple cells: inspire convolution layers
  - Complex cells: inspire pooling layers

David H. Hubel

Torsten N. Wiesel

Variants of pooling

- Stride

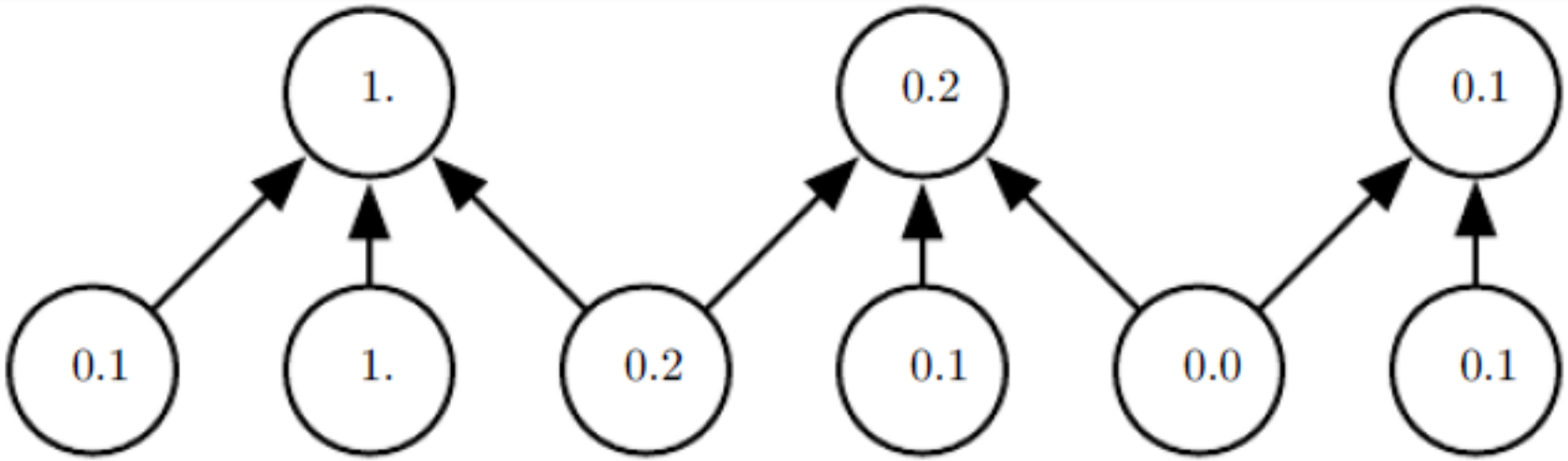
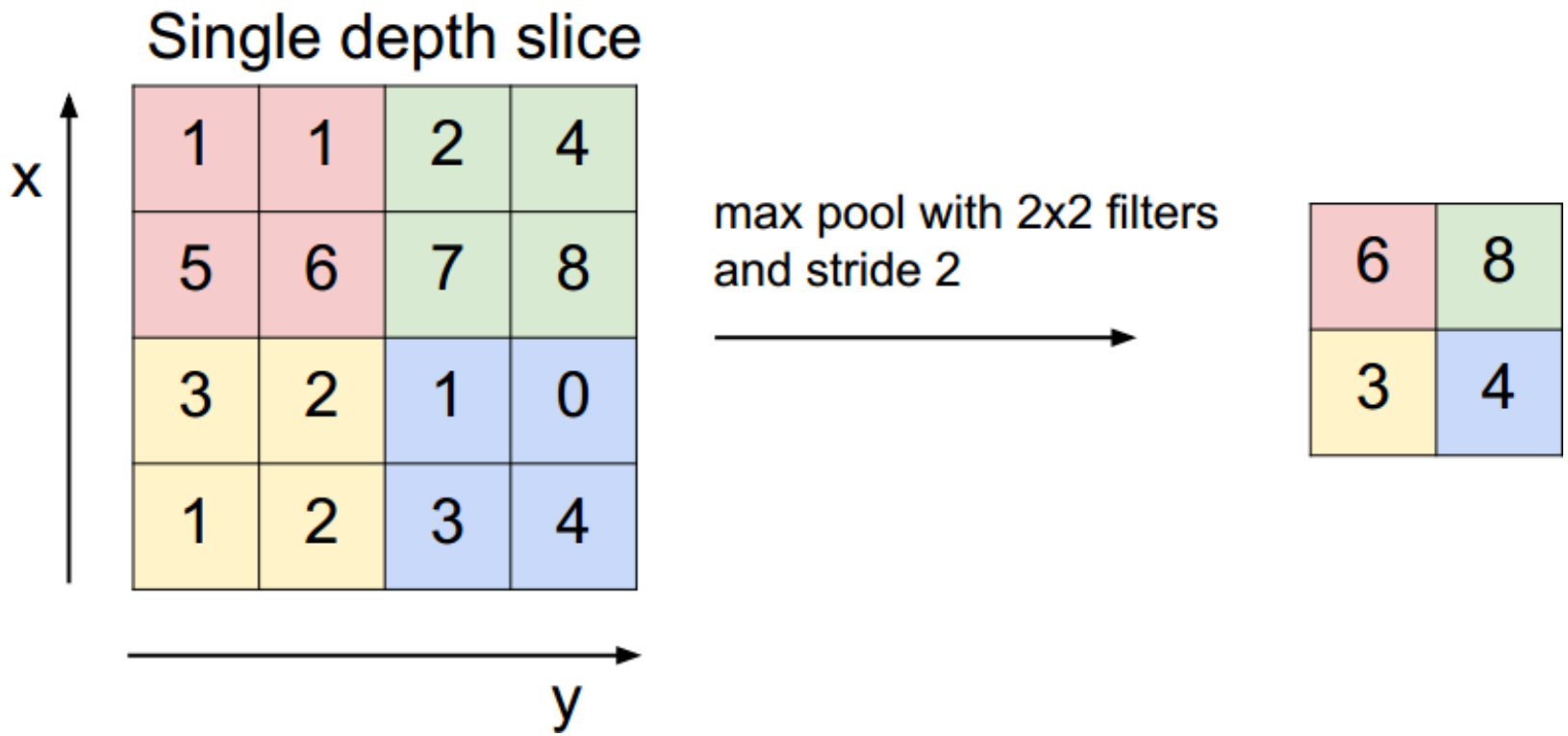


Figure from *Deep Learning,* by Goodfellow, Bengio, and Courville

MAX POOLING


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Gradient of Pooling

- 1 for max values and 0 elsewhere
- Bookkeeping: simply remember the index of the max values.

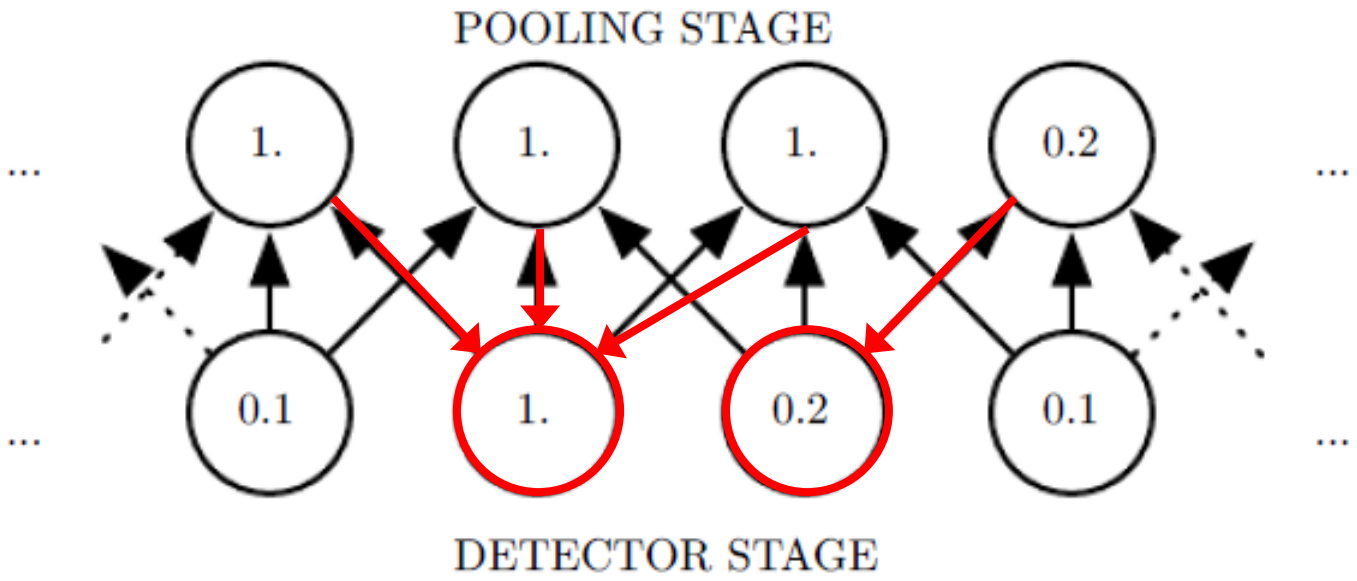


Figure from *Deep Learning,* by Goodfellow, Bengio, and Courville

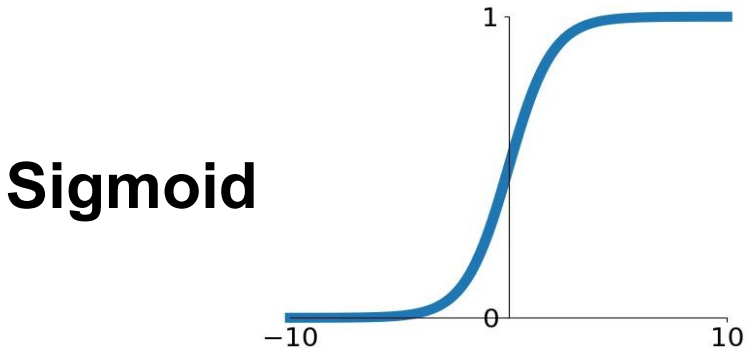
Output normalization

Output normalization

- Normalize the output of a neural network to keep things in range
- Often used for classification (normalize the output to a proper probability distribution)

Output normalization: Sigmoid

- Normalize the output into the range of (0,1)
- As a probability distribution for a *binary* variable
- No parameters and differentiable



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

Output normalization: Softmax

- Normalize a vector such that
  - Each element in the range of (0, 1)
  - All elements sum to 1

- As a probability distribution for a *categorical* variable (e.g., $x = \{1, \dots K\}$)

- No parameters and differentiable

Softmax

$$softmax(x_k) = \frac{\exp(x_k)}{\sum_j \exp(x_j)}$$

Loss functions

The training criterion: loss functions

- A loss function compares *the output* of a neural network to *the label*
- A loss function is similar to an error function
- Loss -> 0 if the output matches the label, otherwise a large value

The training criterion: loss functions

- Classification
  - Cross entropy loss
  - C-way classification problem
  - Often in combination with sigmoid (binary) or softmax (C-way)

$$H(y,p) = -\sum_j y_j \log(p_j)$$

- Regression
  - L2 loss

$$L_2(y,\hat{y}) = \sum_j \left(y_j - \hat{y}_j\right)^2$$

How to get the deep networks work?

Deep Learning: Composing a set of (nonlinear) functions g

$$f(\boldsymbol{x};\boldsymbol{\theta}) = g_1(\dots g_{n-1}(g_n(\boldsymbol{x};\boldsymbol{\theta_n}),\boldsymbol{\theta_{n-1}})\dots,\boldsymbol{\theta_1})$$

Each of the function g is represented using a layer of a neural network

- Key element: $\sigma(\boldsymbol{W^T x} + \boldsymbol{b})$
  - Which activation function to use?
  - What linear function to use?
  - Other operations?
  - The design of the network …

Case study: LeNet-5

LeNet-5

- Proposed in *"Gradient-based learning applied to document recognition"*, *by Yann LeCun, Leon Bottou, Yoshua Bengio and Patrick Haffner, in Proceedings of the IEEE, 1998*

LeNet-5

- Proposed in *"Gradient-based learning applied to document recognition", by Yann LeCun, Leon Bottou, Yoshua Bengio and Patrick Haffner, in Proceedings of the IEEE, 1998*

- Apply convolution on 2D images (MNIST) and use backpropagation

LeNet-5

- Proposed in *"Gradient-based learning applied to document recognition", by Yann LeCun, Leon Bottou, Yoshua Bengio and Patrick Haffner, in Proceedings of the IEEE, 1998*

- Apply convolution on 2D images (MNIST) and use backpropagation

- Structure: 2 convolutional layers (with pooling) + 3 fully connected layers
  - Input size: 32x32x1
  - Convolution kernel size: 5x5
  - Pooling: 2x2

LeNet-5

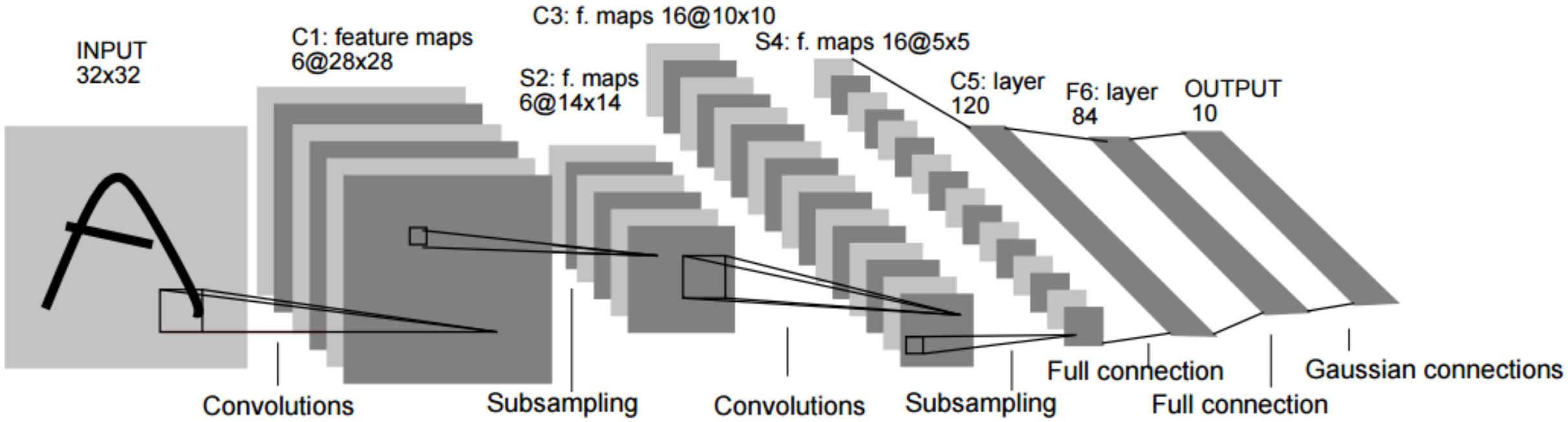


Figure from *Gradient-based learning applied to document recognition, by Y. LeCun, L. Bottou, Y. Bengio and P. Haffner*

LeNet-5

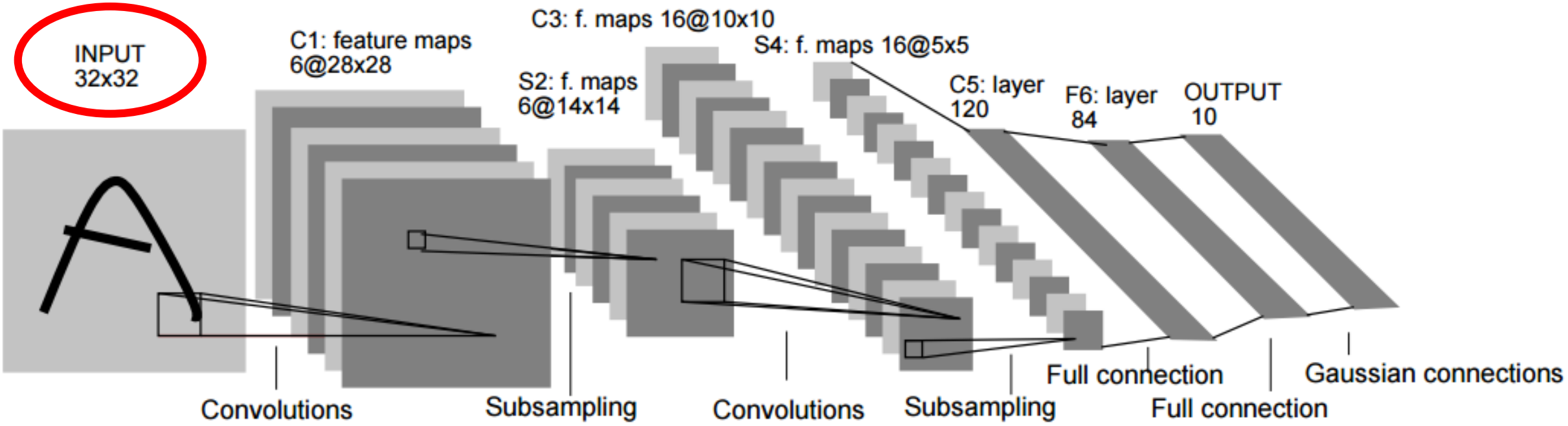


Figure from *Gradient-based learning applied to document recognition, by Y. LeCun, L. Bottou, Y. Bengio and P. Haffner*

LeNet-5

Filter: 5x5, stride: 1x1,
#filters: 6
Activation: sigmoid

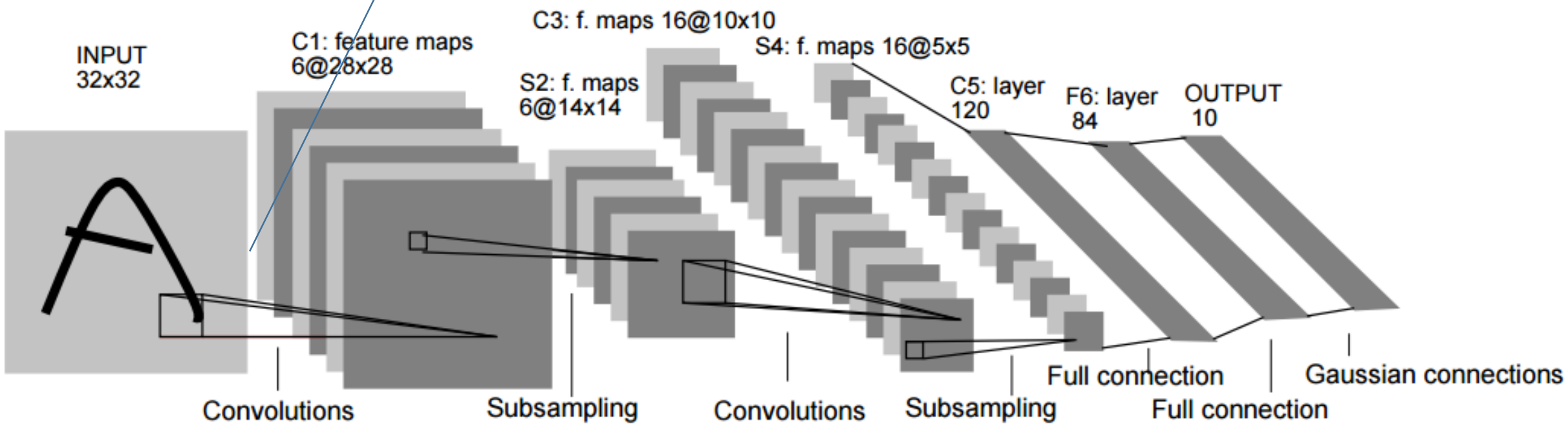


Figure from *Gradient-based learning applied to document recognition, by Y. LeCun, L. Bottou, Y. Bengio and P. Haffner*

LeNet-5

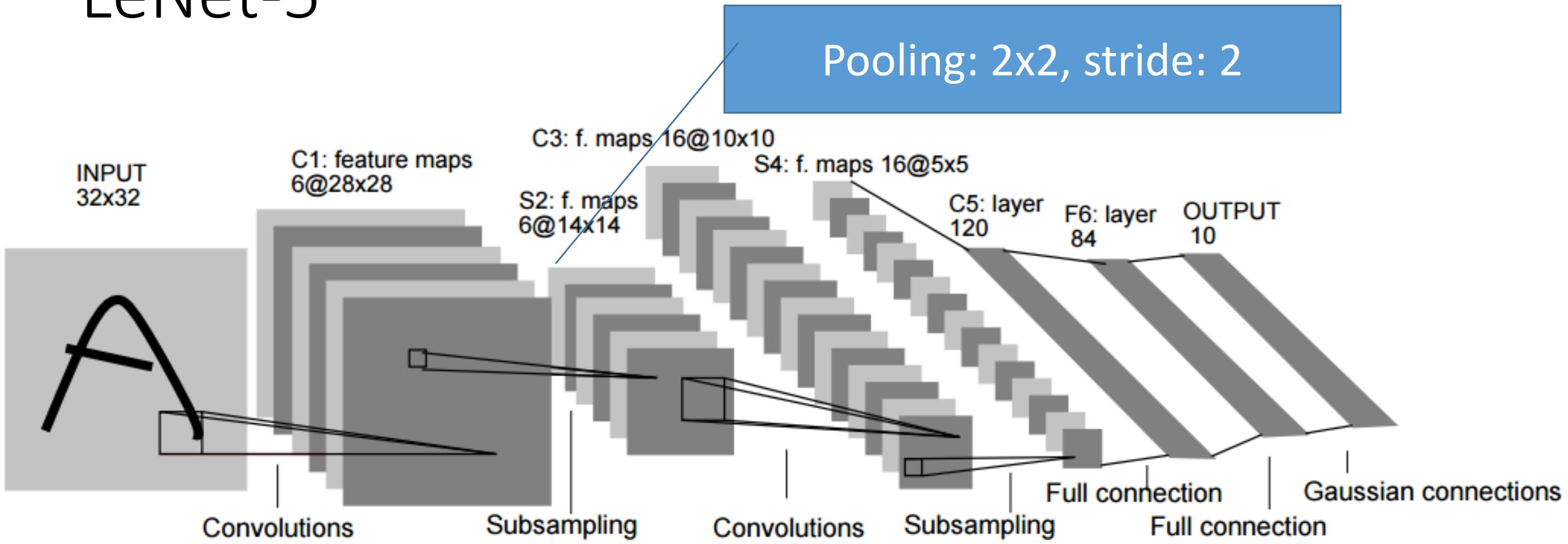


Figure from *Gradient-based learning applied to document recognition, by Y. LeCun, L. Bottou, Y. Bengio and P. Haffner*

LeNet-5

Filter: 5x5x6, stride: 1x1,
#filters: 16
Activation: sigmoid

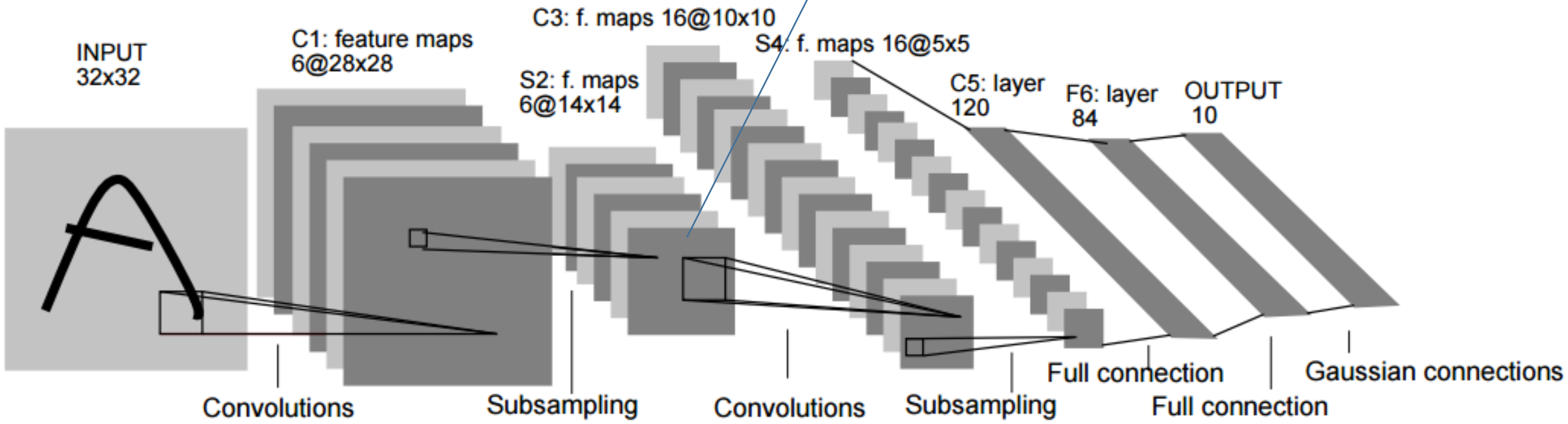


Figure from *Gradient-based learning applied to document recognition, by Y. LeCun, L. Bottou, Y. Bengio and P. Haffner*

LeNet-5

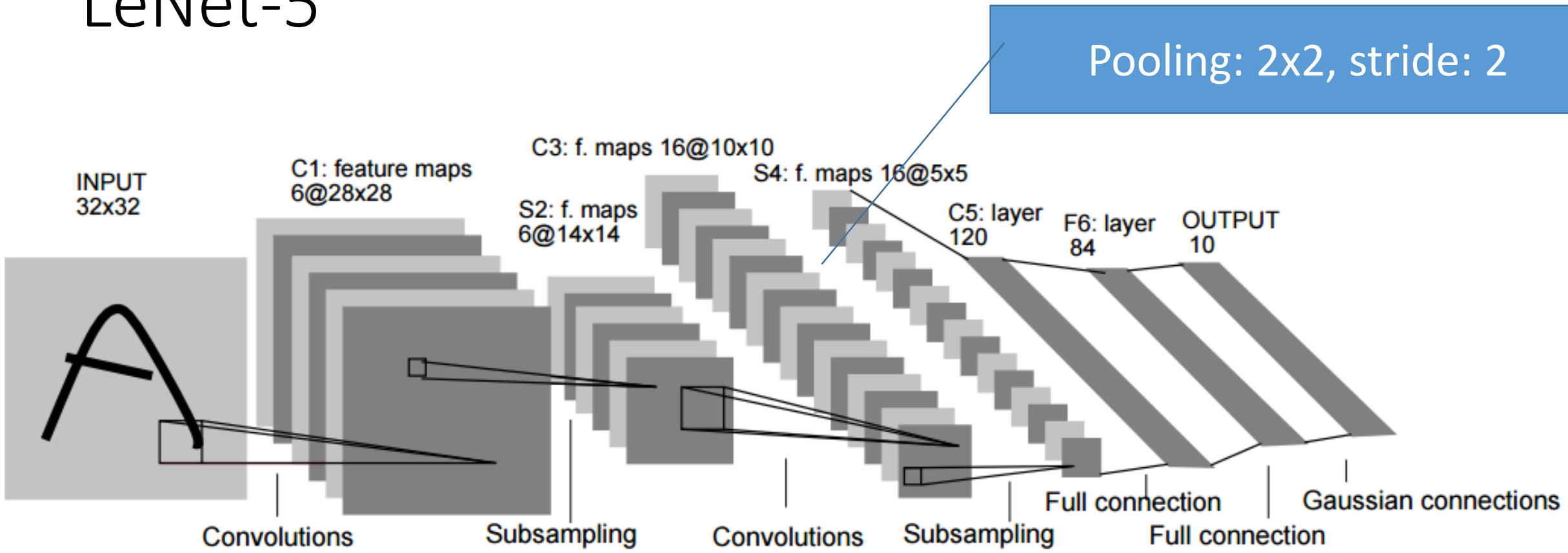


Figure from *Gradient-based learning applied to document recognition, by Y. LeCun, L. Bottou, Y. Bengio and P. Haffner*

LeNet-5

Weight matrix: 400x120
Activation: sigmoid

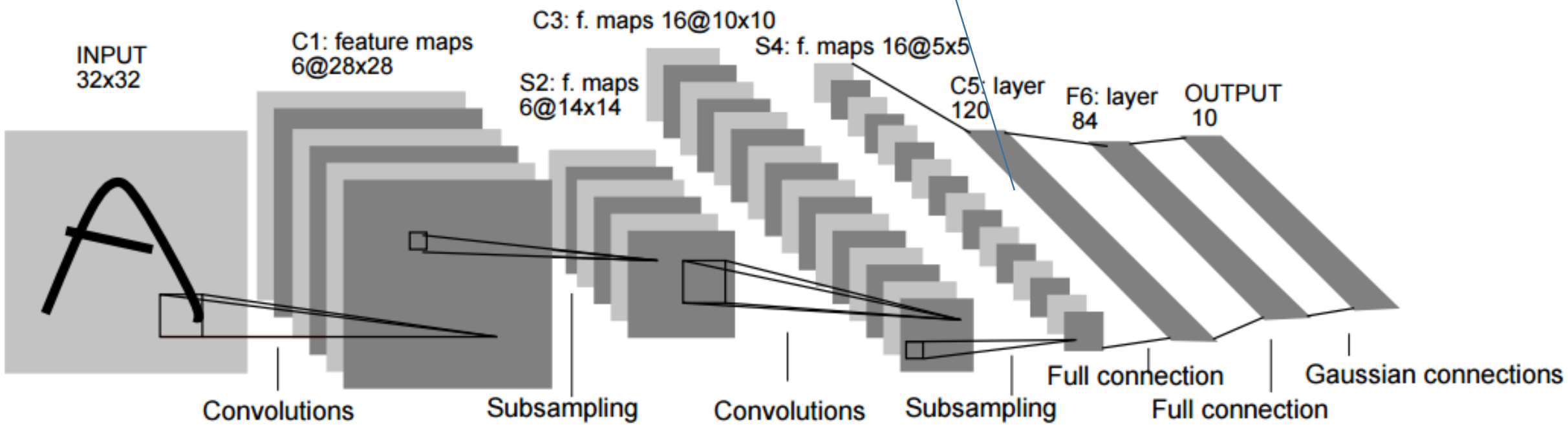


Figure from *Gradient-based learning applied to document recognition, by Y. LeCun, L. Bottou, Y. Bengio and P. Haffner*

LeNet-5

Weight matrix: 84x10

Weight matrix: 120x84
Activation: sigmoid

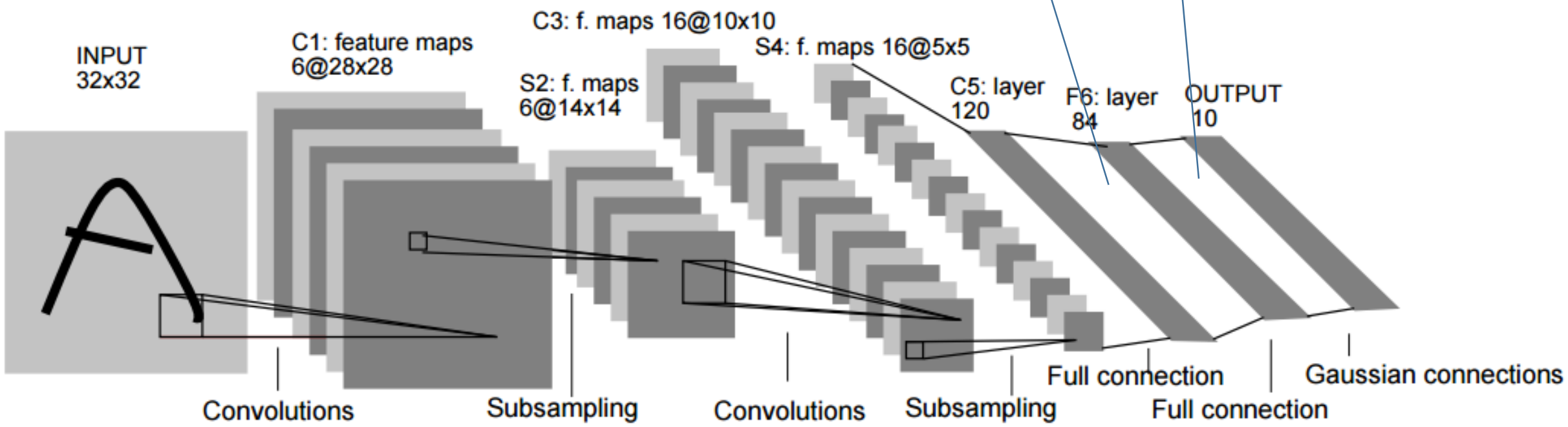


Figure from *Gradient-based learning applied to document recognition, by Y. LeCun, L. Bottou, Y. Bengio and P. Haffner*

LeNet-5

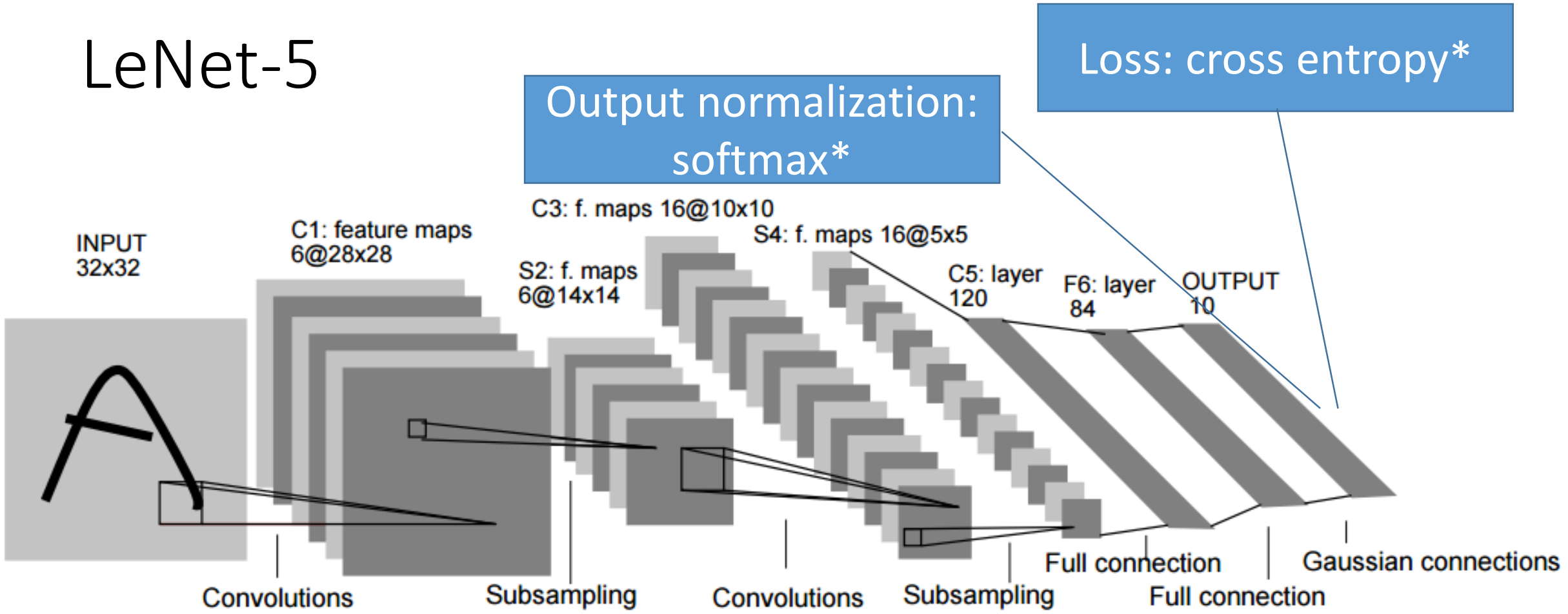


Figure from *Gradient-based learning applied to document recognition, by Y. LeCun, L. Bottou, Y. Bengio and P. Haffner*