Deep Learning
Part I

Yin Li

yin.li@wisc.edu

University of Wisconsin, Madison

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

Neural Networks / Deep Learning

- What type of functions shall we consider for *f*?

→ Features + Decision $f(x;\ \theta)$ → Chair

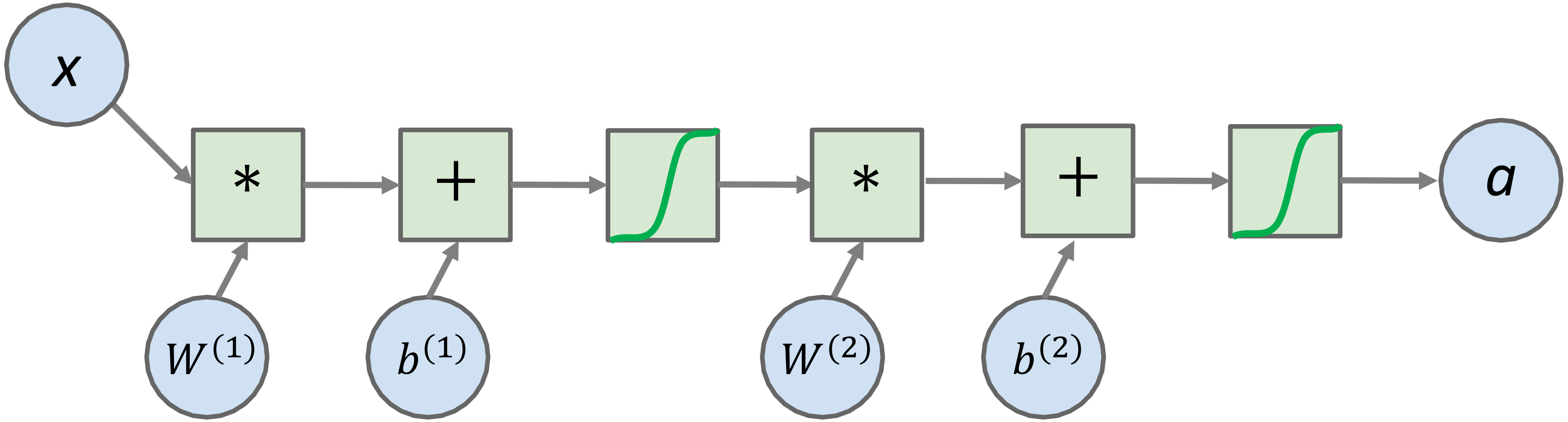
Proposal: Composing a set of (nonlinear) functions *g*

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

Example: $\mathbf{a} = sigmoid\left(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b}\right) = g(\boldsymbol{x};\boldsymbol{W},\boldsymbol{b})$

Neural network as a computational graph

- A two-layer neural network
- Forward propagation vs. backward propagation



What prevent us from learning a deep network?

- Say 100 layers …

- Way too many parameters
  - $\mathbf{a} = sigmoid\left(\boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{x} + \boldsymbol{b}\right) = g(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b})$
  - $\boldsymbol{x} \in R^{n},\ \boldsymbol{W} \in R^{n \times m},\ \boldsymbol{b} \in R^{m},\ \boldsymbol{a} \in R^{m}$
  - If you have a high dimensional input (e.g., an image)

- Gradient descent does not quite work any more …

Deep learning: a sketch

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

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

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

- General form for each layer $\mathbf{a} = \sigma\left(\boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{x} + \boldsymbol{b}\right) = g(\boldsymbol{x};\boldsymbol{W},\boldsymbol{b})$
- σ the activation function
- Key element: Linear operations + Nonlinear activations

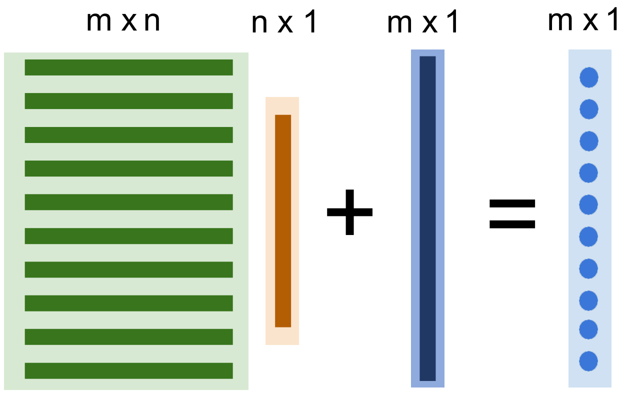
Deep learning: a sketch

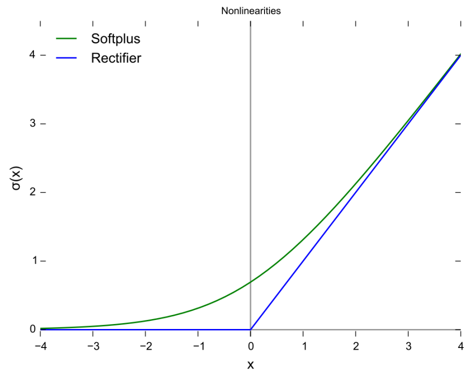
Deep Learning: Composing a set of (nonlinear) functions g

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

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

- Key element: Linear operations + Nonlinear activations $\sigma(\boldsymbol{W^T x} + \boldsymbol{b})$





How to get the deep networks work?

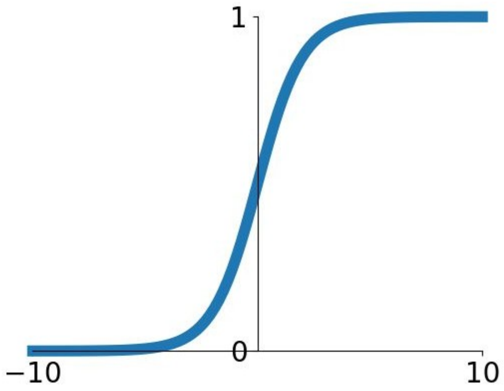
Deep Learning: Composing a set of (nonlinear) functions g

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

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

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

The choice of activation function

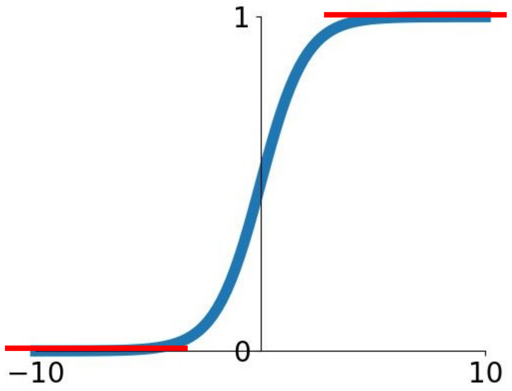


Sigmoid

$$g(x) = 1/(1 + \exp(-x))$$

$$g'(x) = g(x)(1 - g(x))$$

The choice of activation function



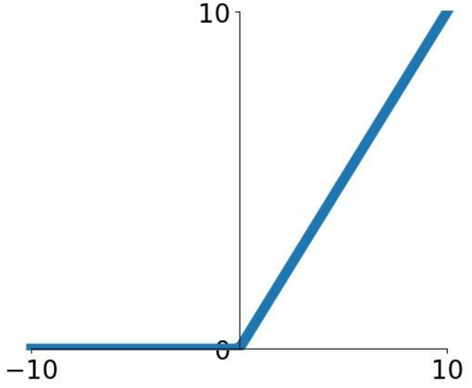
Sigmoid

$$g(x) = 1/(1 + \exp(-x))$$

$$g'(x) = g(x)(1 - g(x))$$

- Saturated neurons “kill” the gradients
- Exponential function is expensive

The choice of activation function

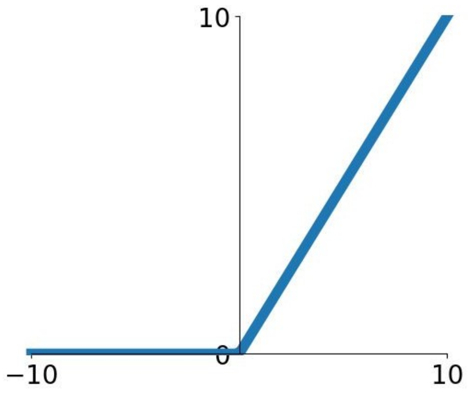


ReLU
(Rectified Linear Unit)

f(x) = max(0, x)

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid in practice
- Differentiable?

The choice of activation function

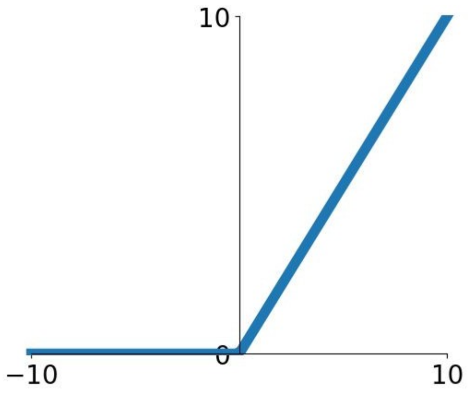


ReLU
(Rectified Linear Unit)

f(x) = max(0, x)

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid in practice
- Differentiable? Yes, if we fix $f'(0)$

The choice of activation function



ReLU

(Rectified Linear Unit)

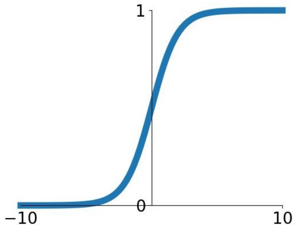
f(x) = max(0, x)

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid in practice
- Differentiable? Yes, if we fix $f'(0)$
- Zero gradient in -region

The choice of activation function

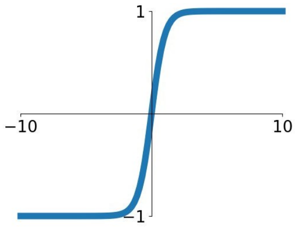
Sigmoid

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



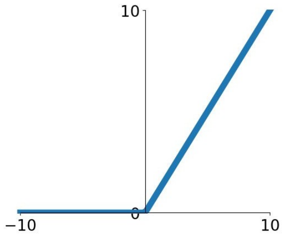
tanh

$\tanh(x)$



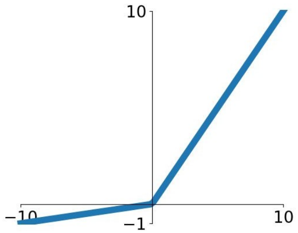
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

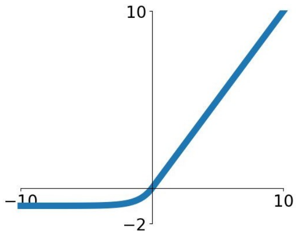


Maxout

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

ELU

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



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

Convolution layer

- Use convolution in place of general matrix multiplication

$$\mathrm{a} = \sigma\left(\boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{x} + \boldsymbol{b}\right)$$

for a specific kind of weight matrix $\boldsymbol{W}$

- Strong empirical application performance

Convolution

Convolution: discrete version

- Given array u_t and w_t, their convolution is a function s_t

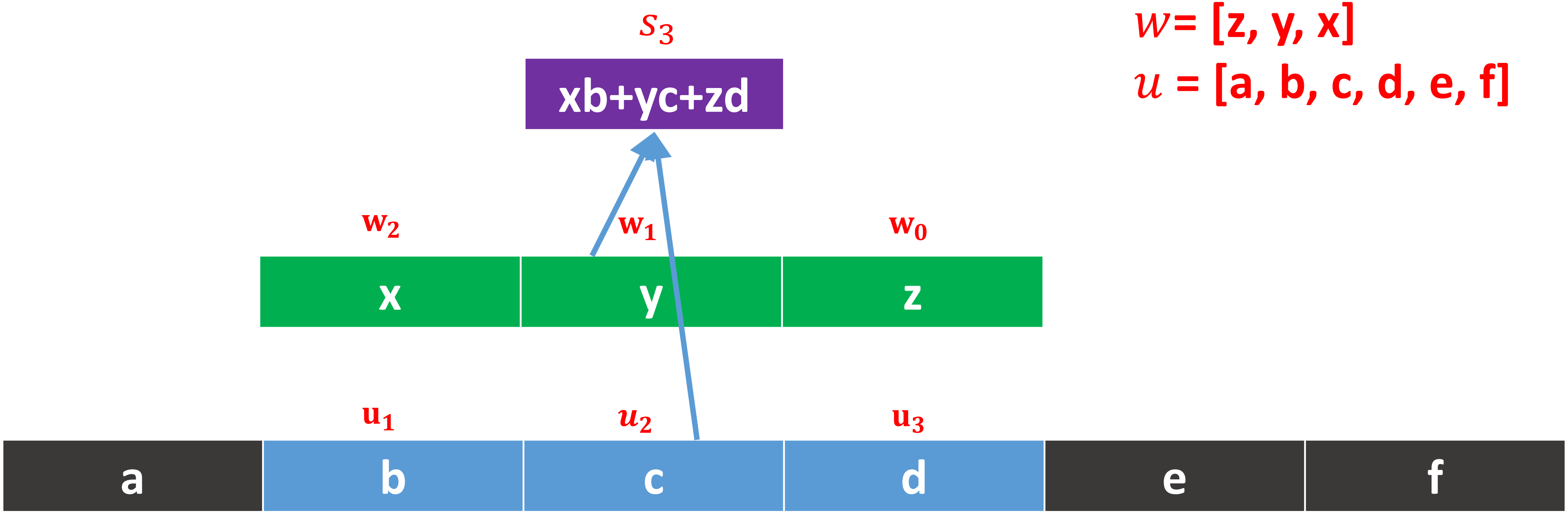
$$s_t = \sum_{a=-\infty}^{+\infty} u_a w_{t-a}$$

- Written as

$$s = (u * w) \quad \text{or} \quad s_t = (u * w)_t$$

- When u_t or w_t is not defined, assumed to be 0

Illustration 1



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

Illustration 1

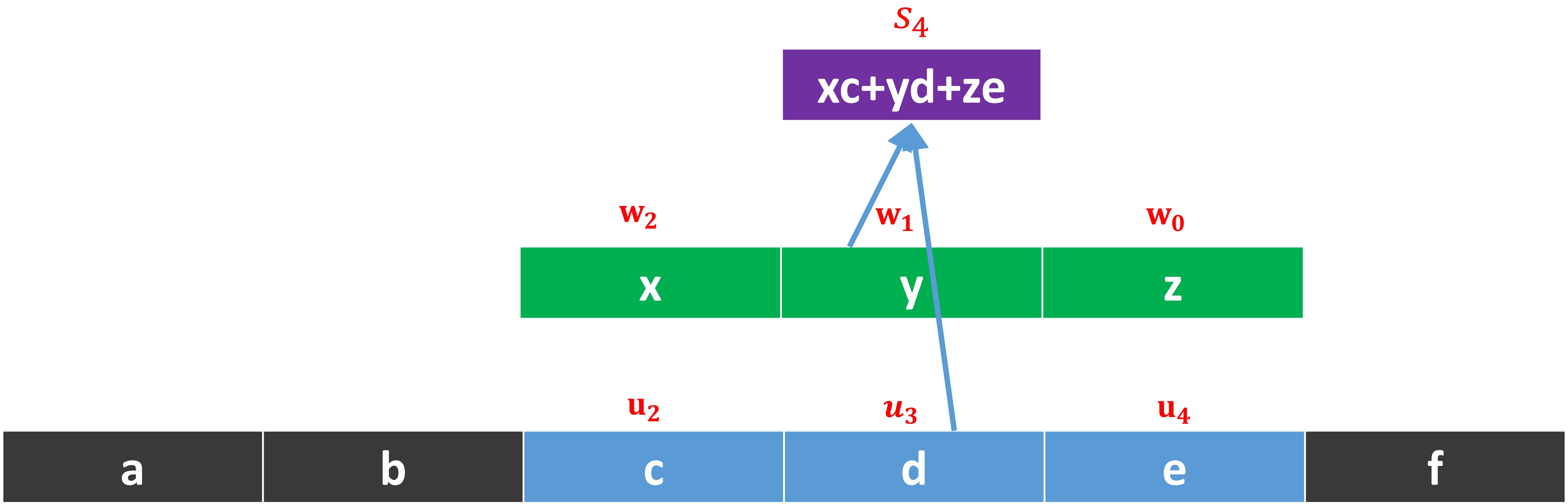


Illustration 1

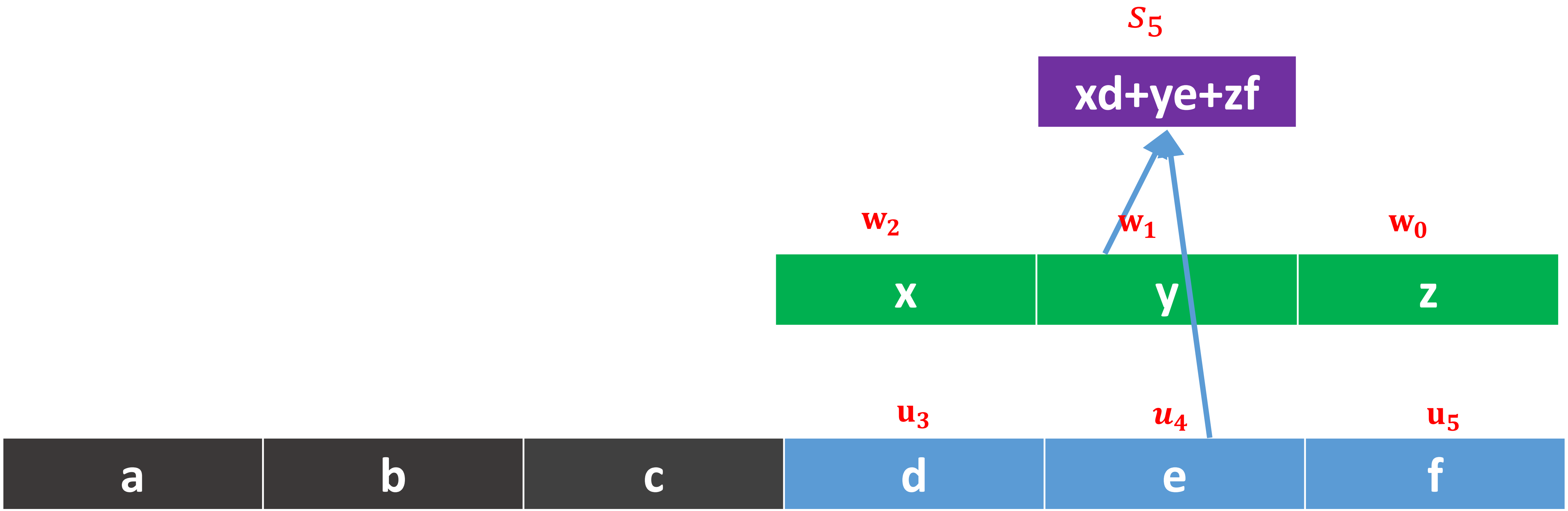


Illustration 1: boundary case

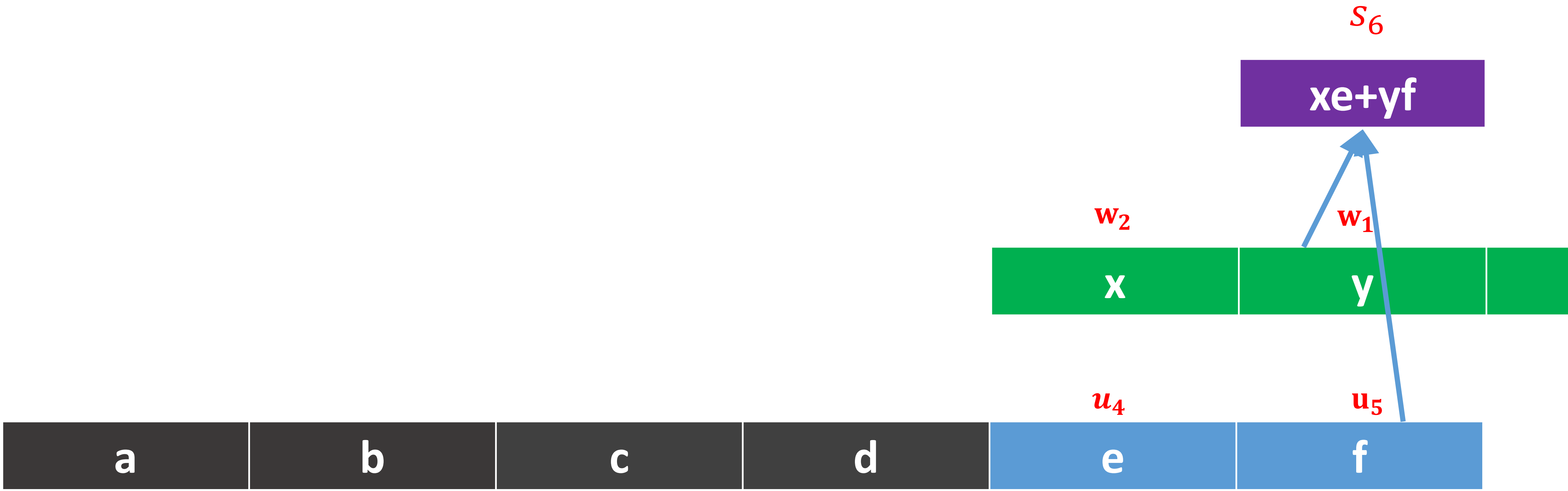


Illustration 1 as matrix multiplication

$$
\begin{bmatrix}
y & z & & & & \\
x & y & z & & & \\
 & x & y & z & & \\
 & & x & y & z & \\
 & & & x & y & z \\
 & & & & x & y
\end{bmatrix}
\quad
\begin{bmatrix}
a \\ b \\ c \\ d \\ e \\ f
\end{bmatrix}
$$

Illustration 2: two dimensional case

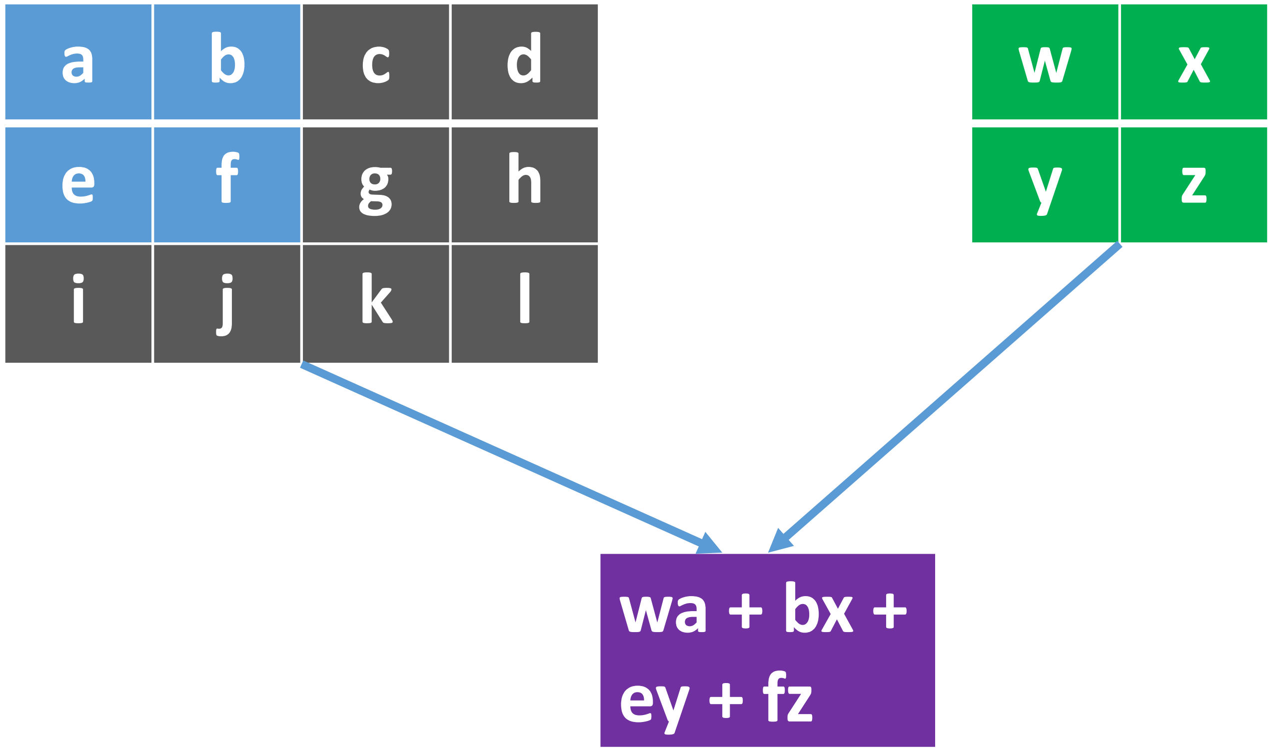


Illustration 2

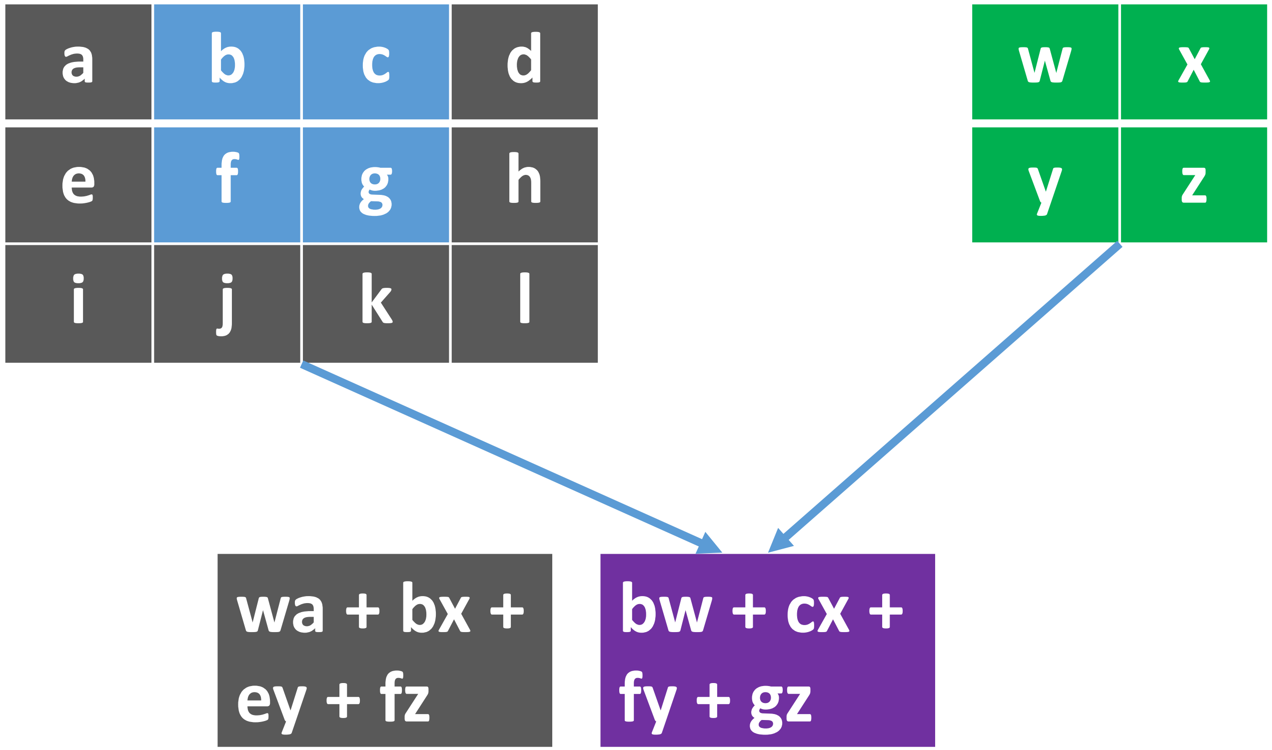
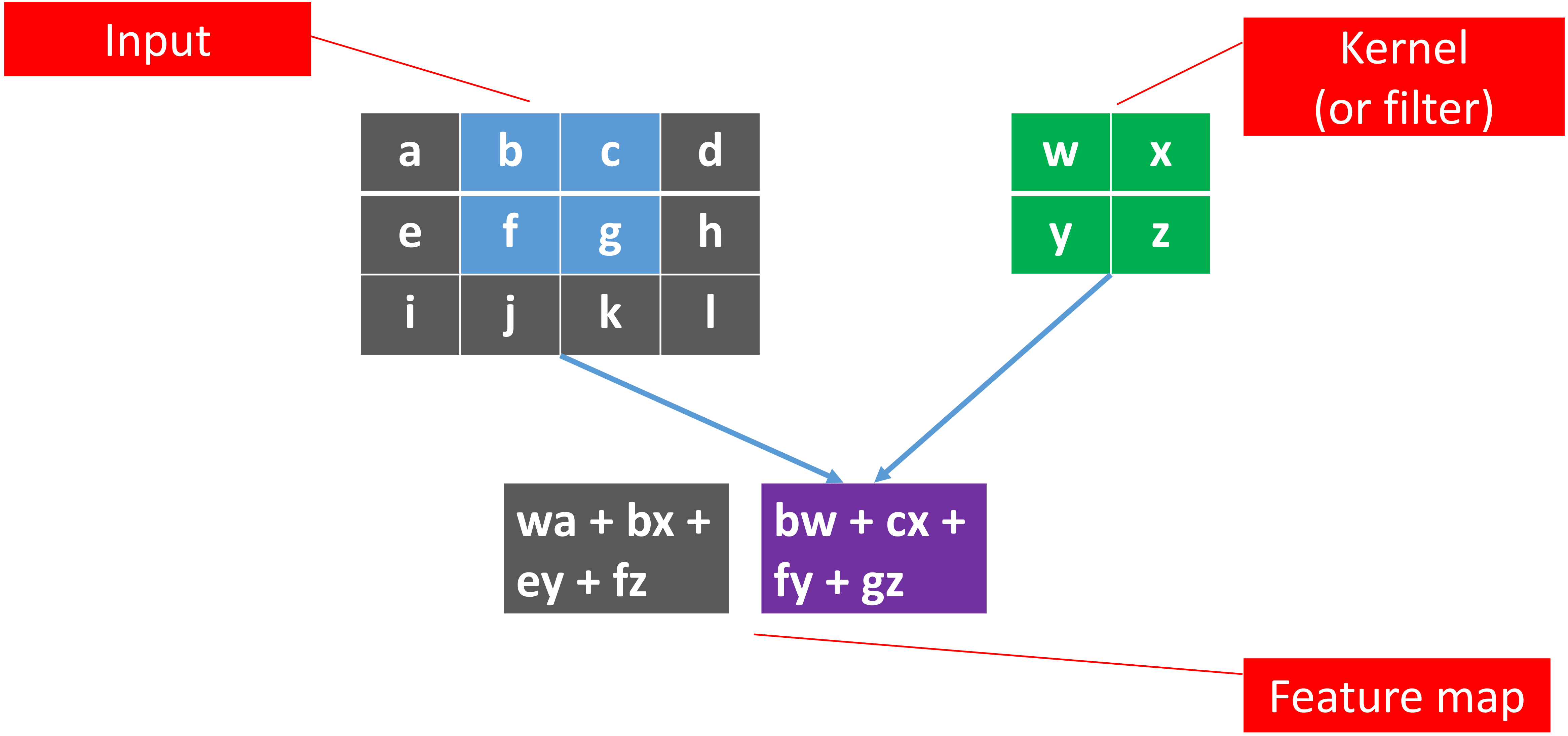
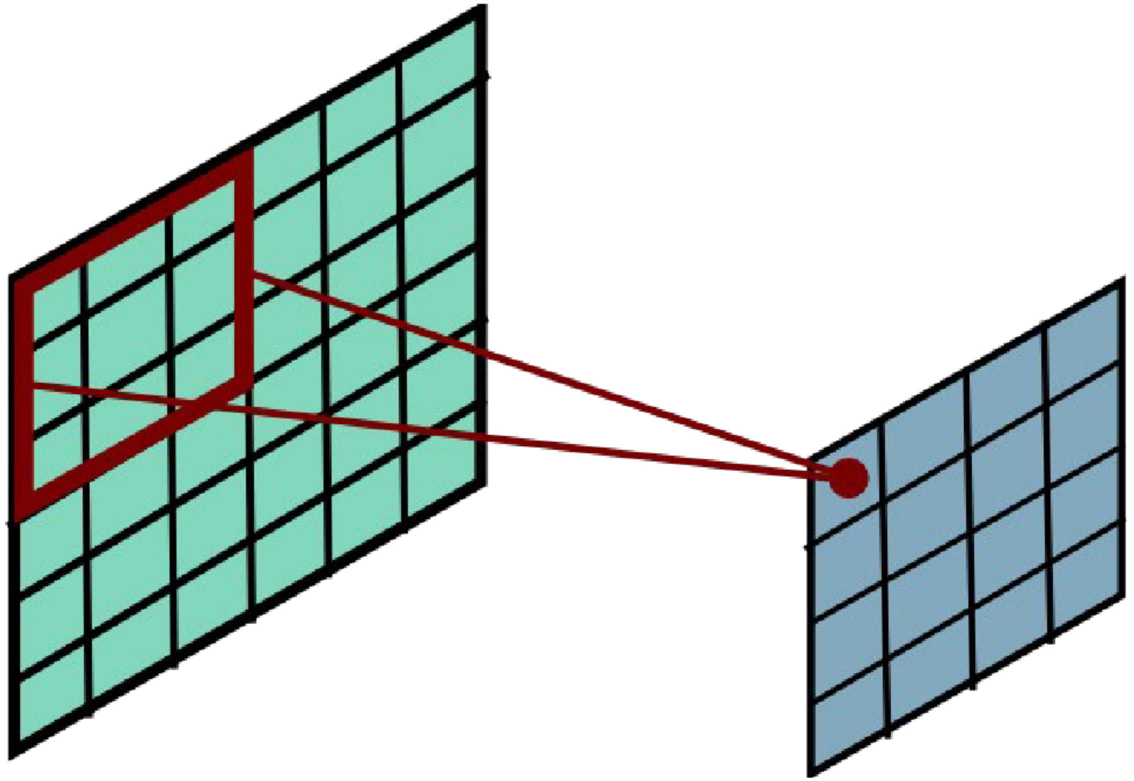
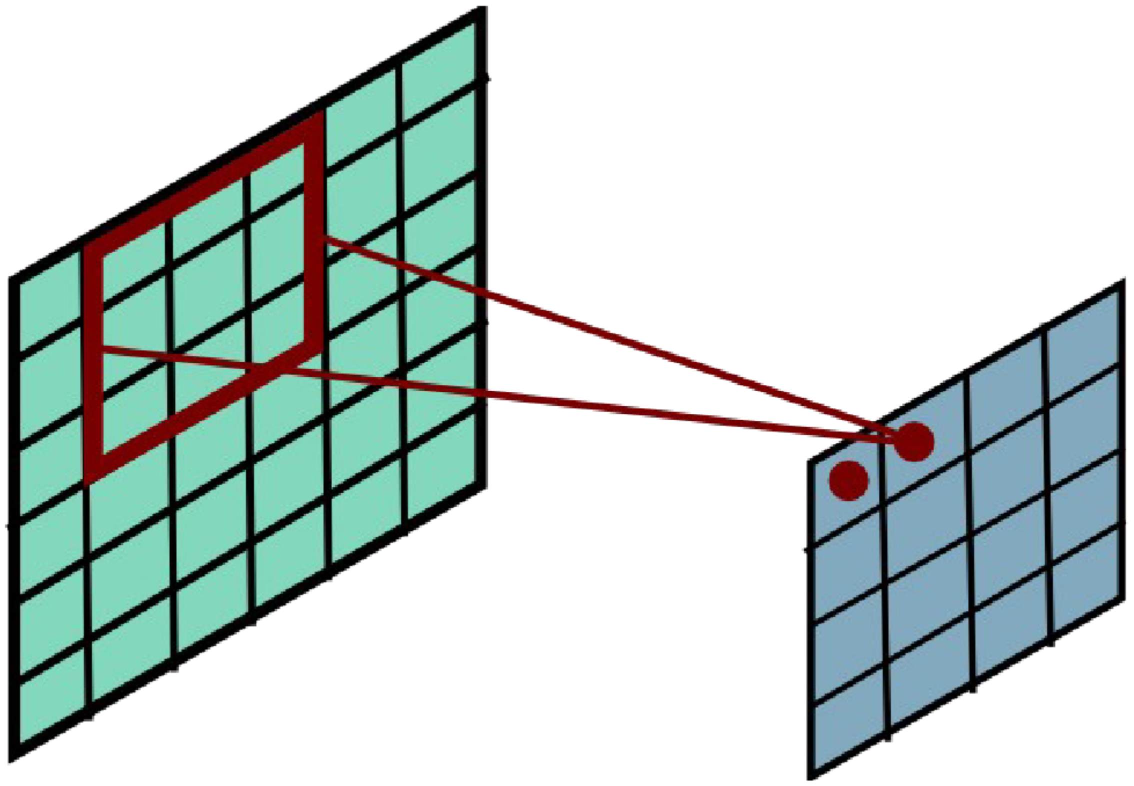


Illustration 2

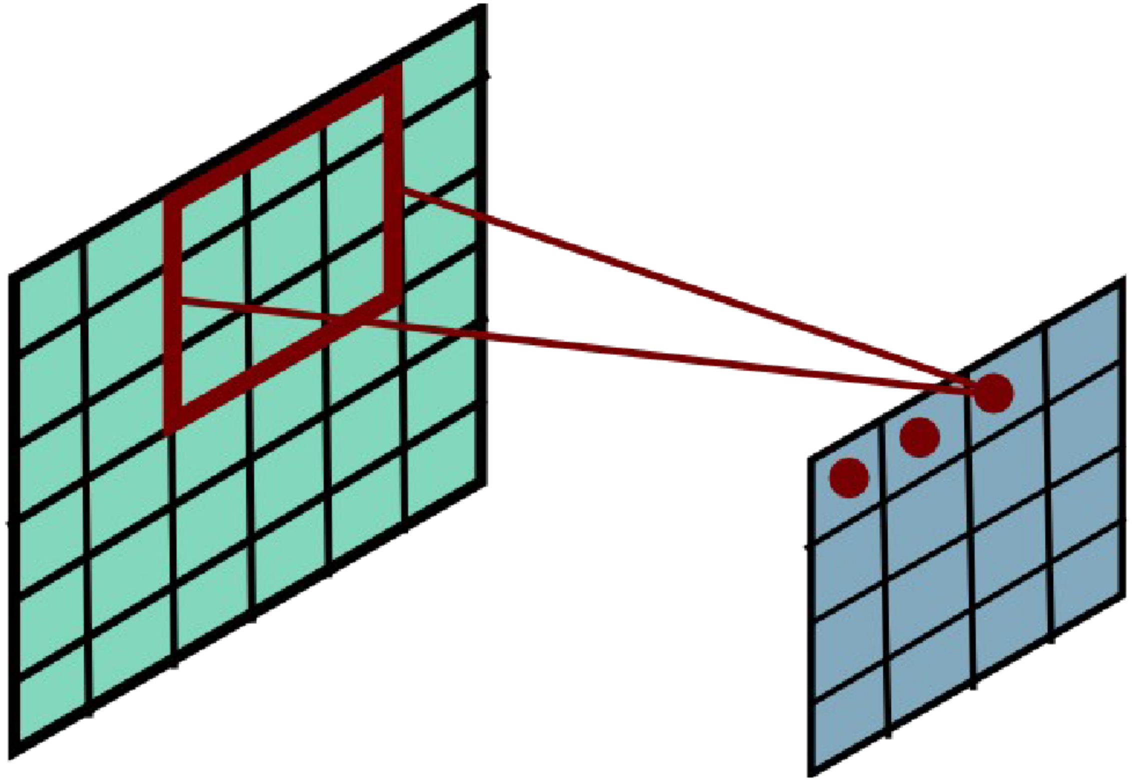




Slides Credit: Deep Learning Tutorial by Marc'Aurelio Ranzato

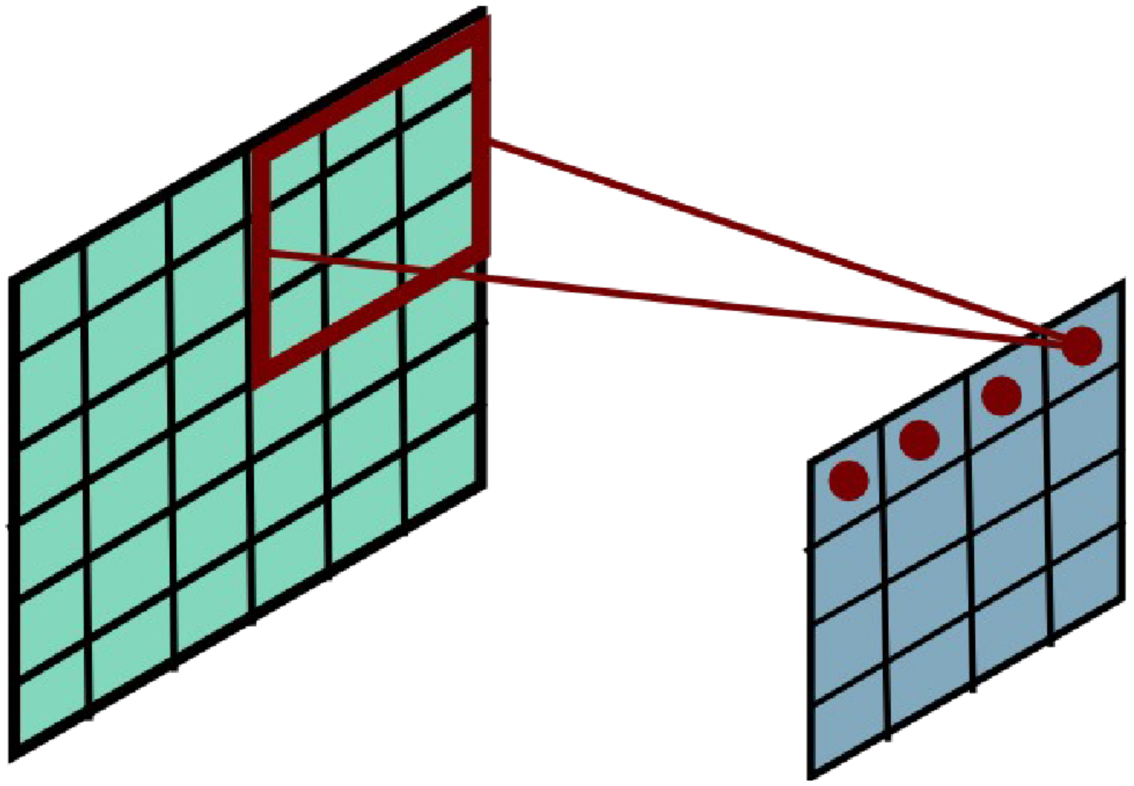


Slides Credit: Deep Learning Tutorial by Marc'Aurelio Ranzato

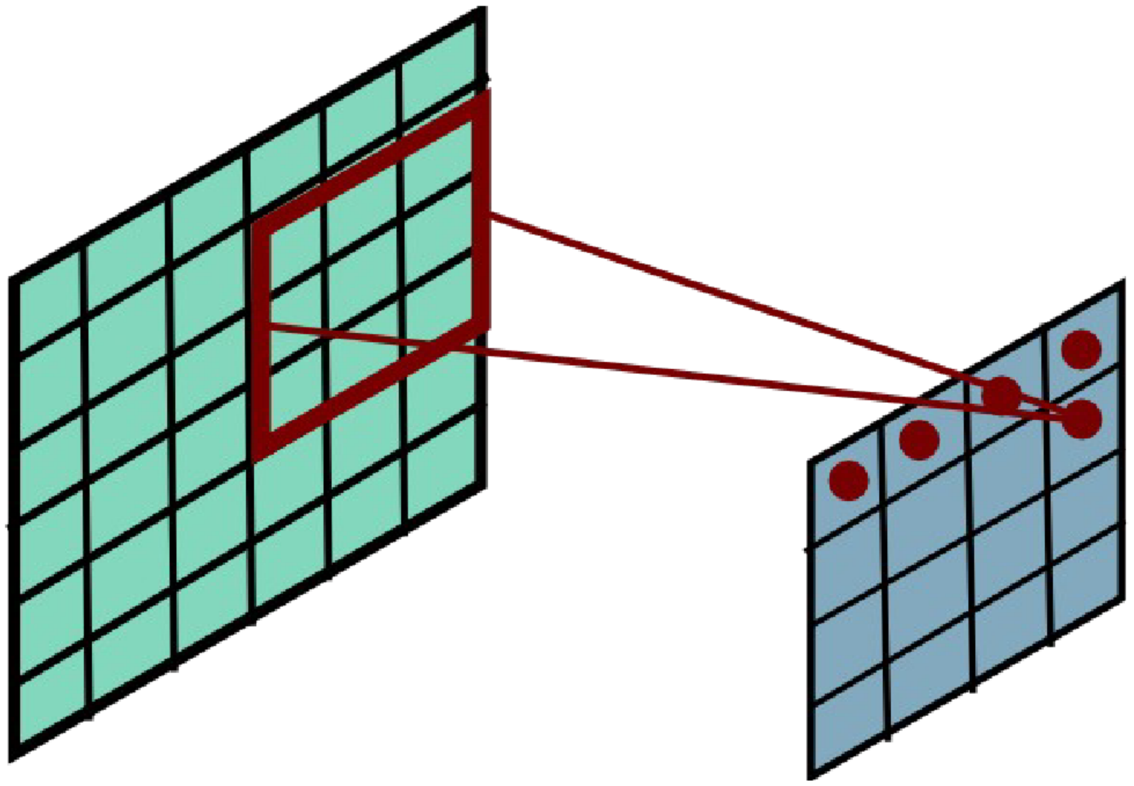


Slides Credit: Deep Learning Tutorial by Marc'Aurelio Ranzato

Ranzato

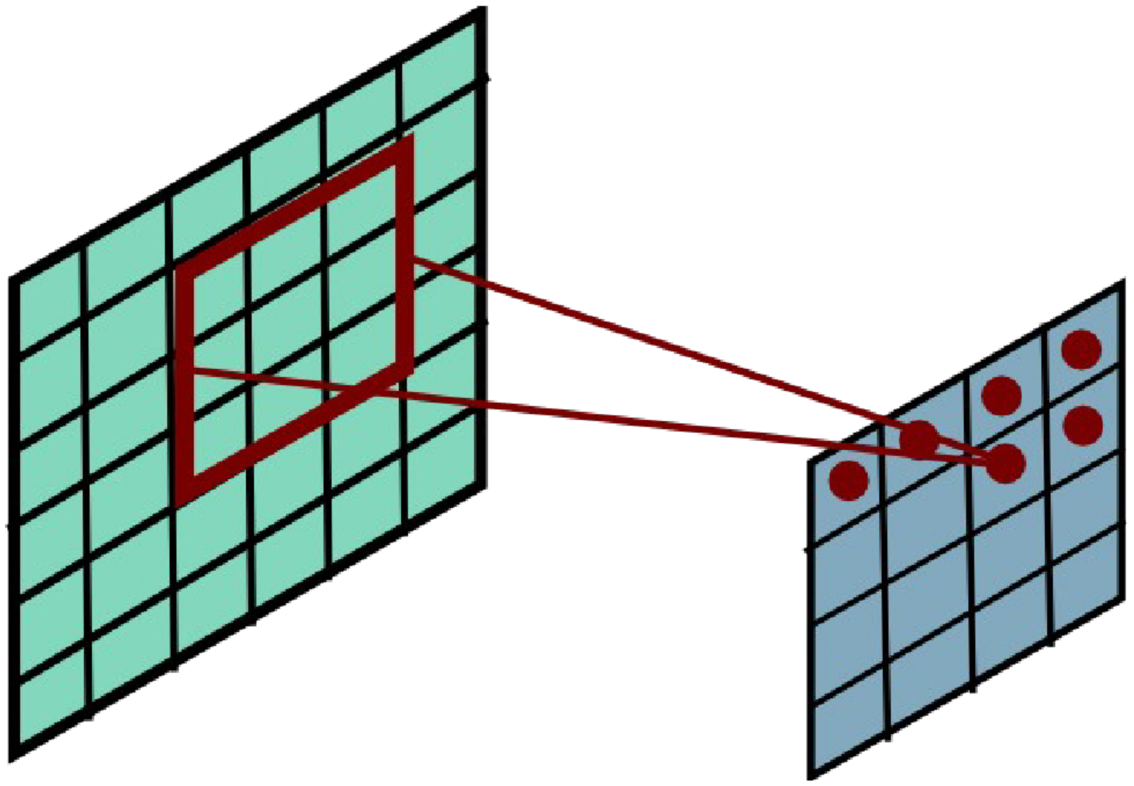


Slides Credit: Deep Learning Tutorial by Marc'Aurelio Ranzato

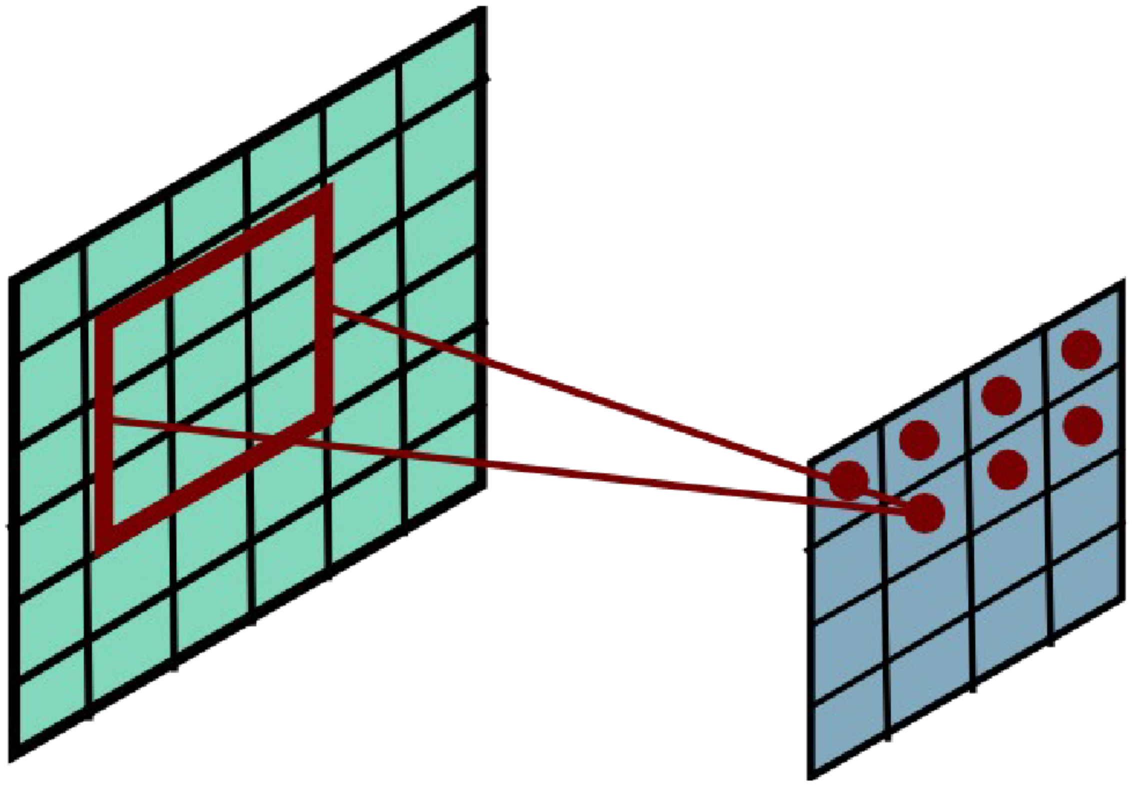


Slides Credit: Deep Learning Tutorial by Marc'Aurelio Ranzato

Ranzato

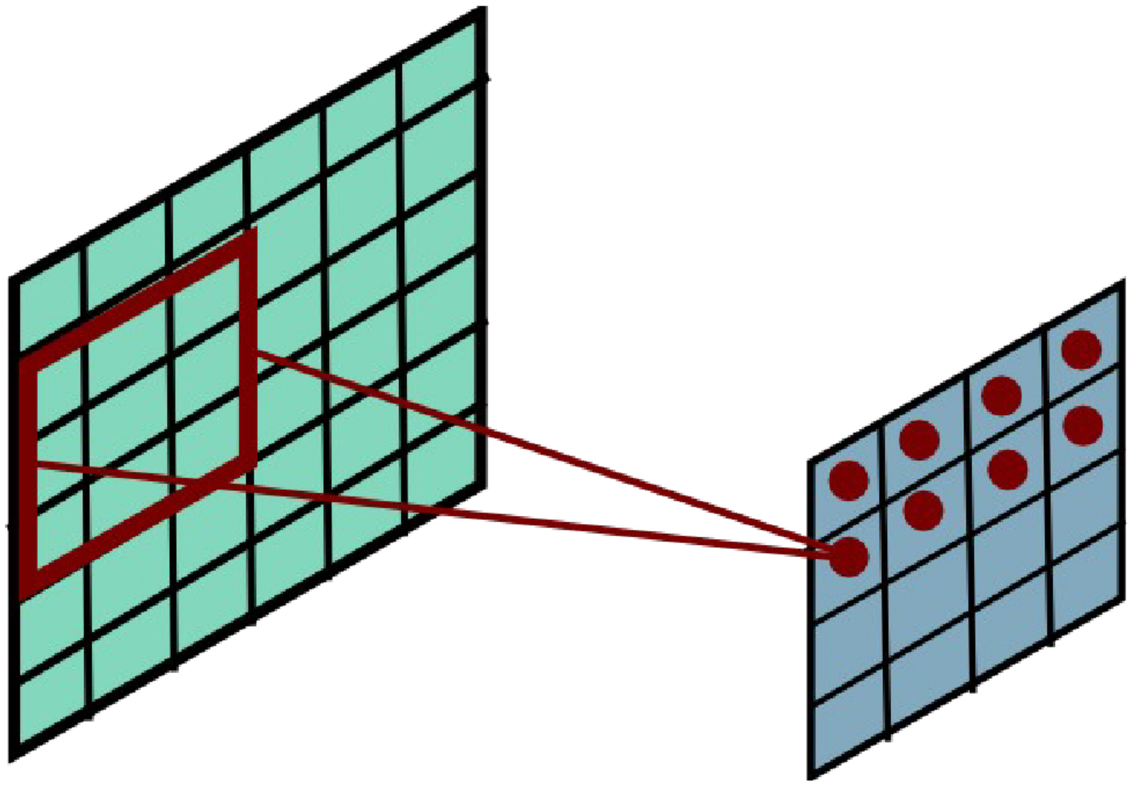


Slides Credit: Deep Learning Tutorial by Marc'Aurelio Ranzato



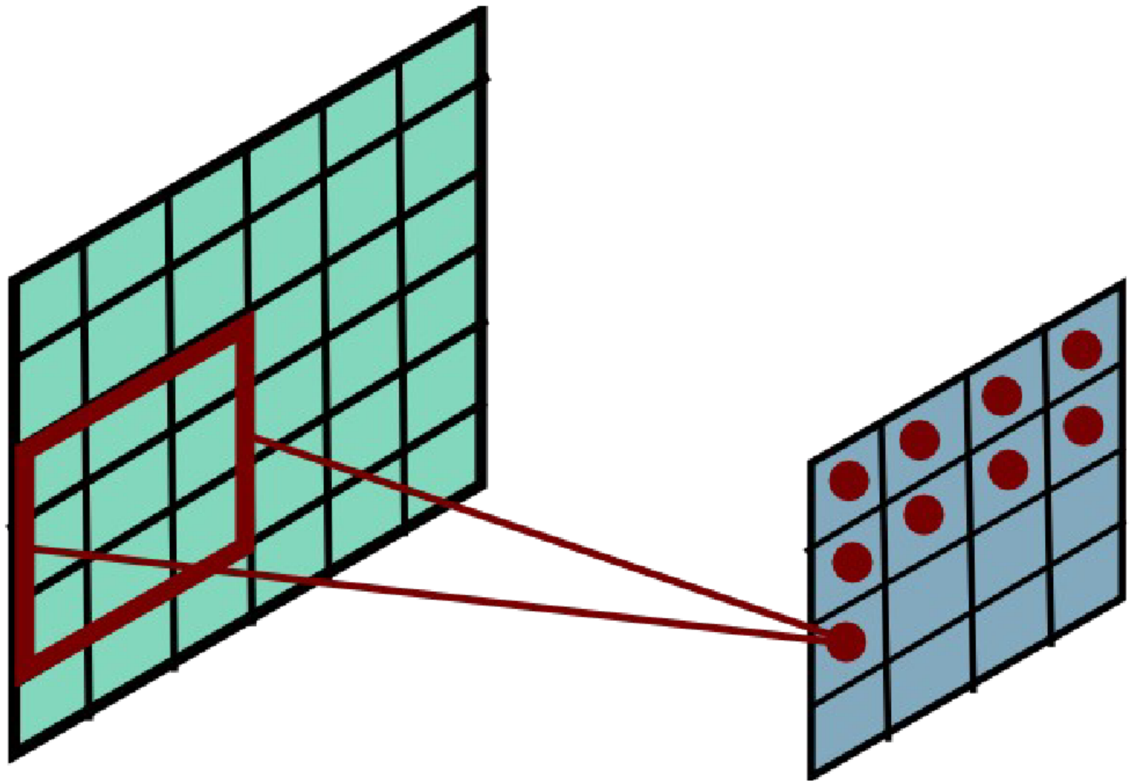
Slides Credit: Deep Learning Tutorial by Marc'Aurelio Ranzato

Ranzato



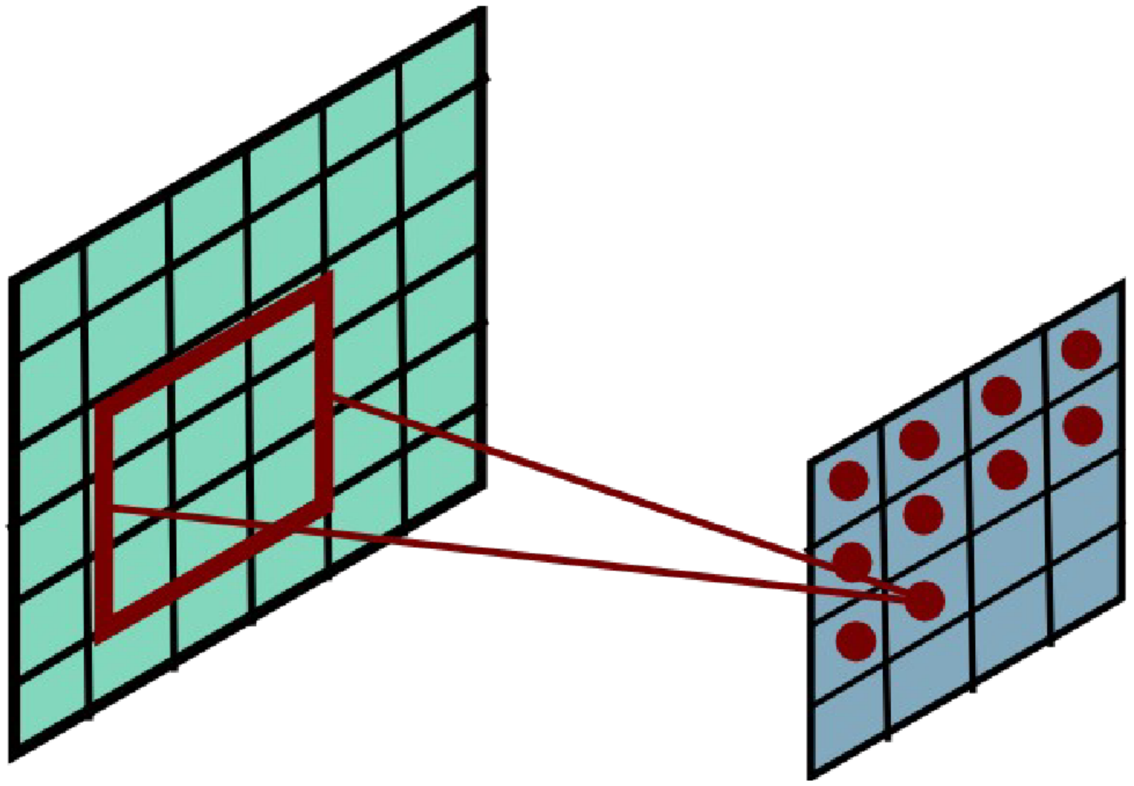
Slides Credit: Deep Learning Tutorial by Marc'Aurelio Ranzato

Ranzato

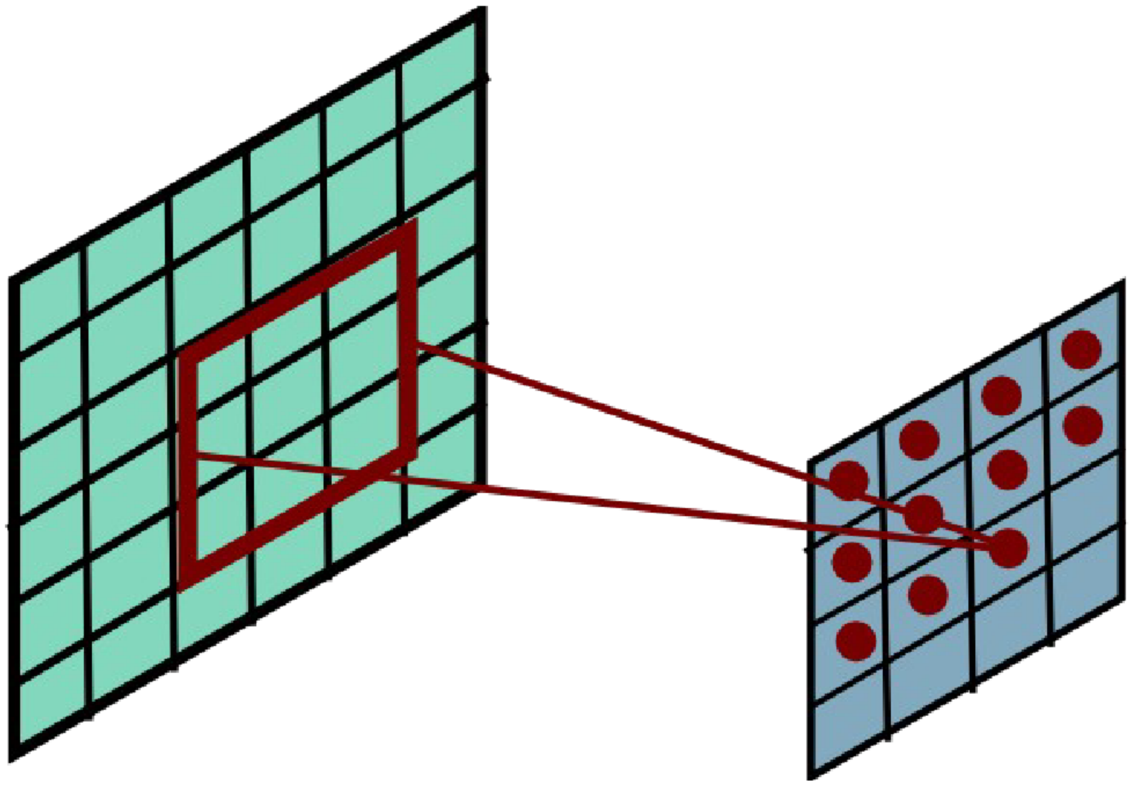


Slides Credit: Deep Learning Tutorial by Marc’Aurelio Ranzato

Ranzato

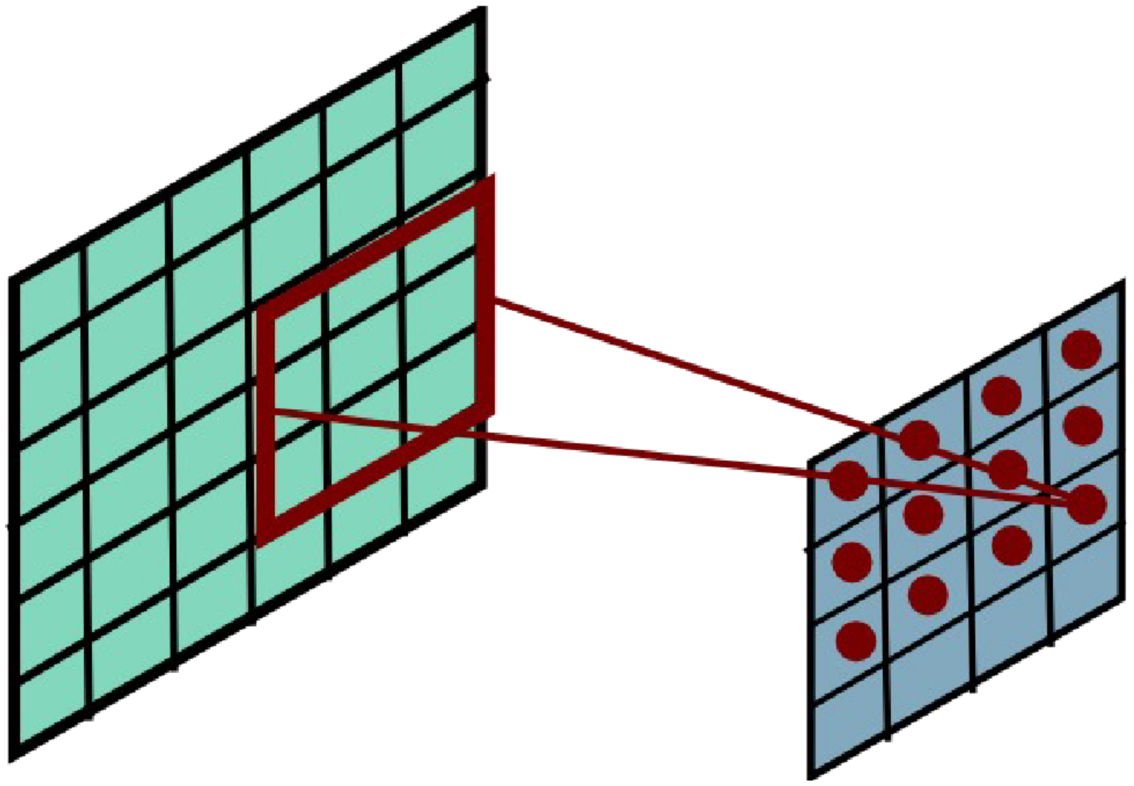


Slides Credit: Deep Learning Tutorial by Marc'Aurelio Ranzato



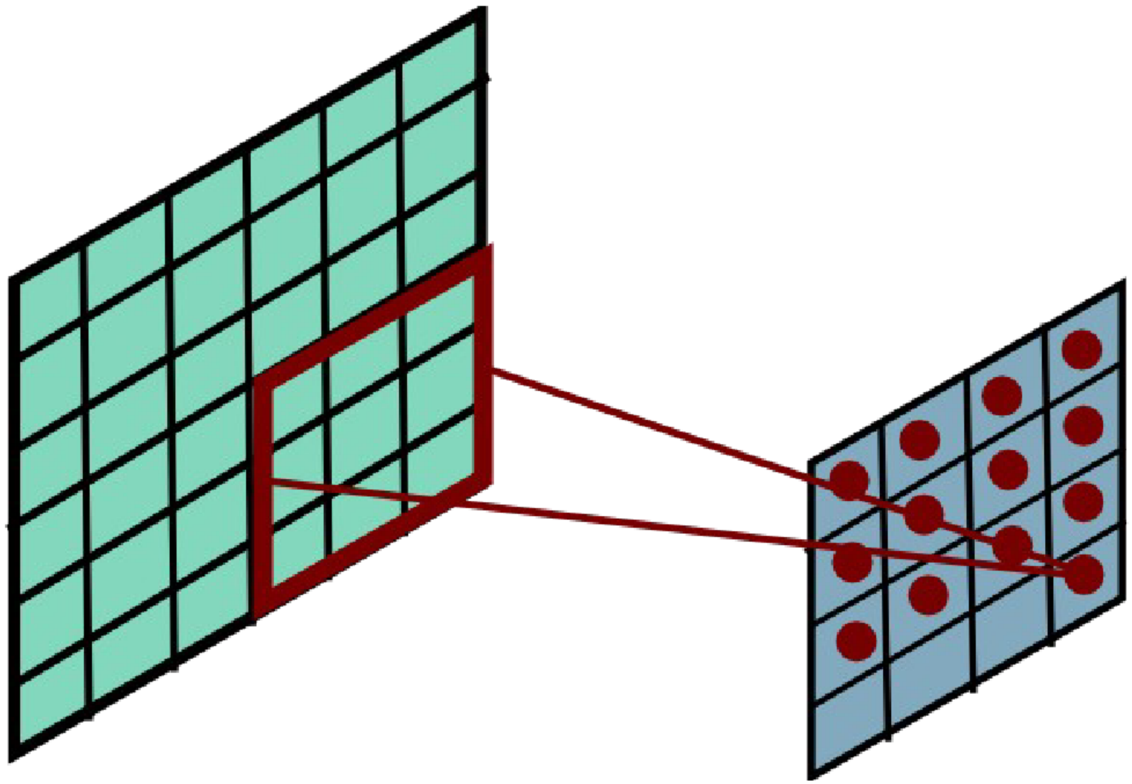
Slides Credit: Deep Learning Tutorial by Marc'Aurelio Ranzato

Ranzato

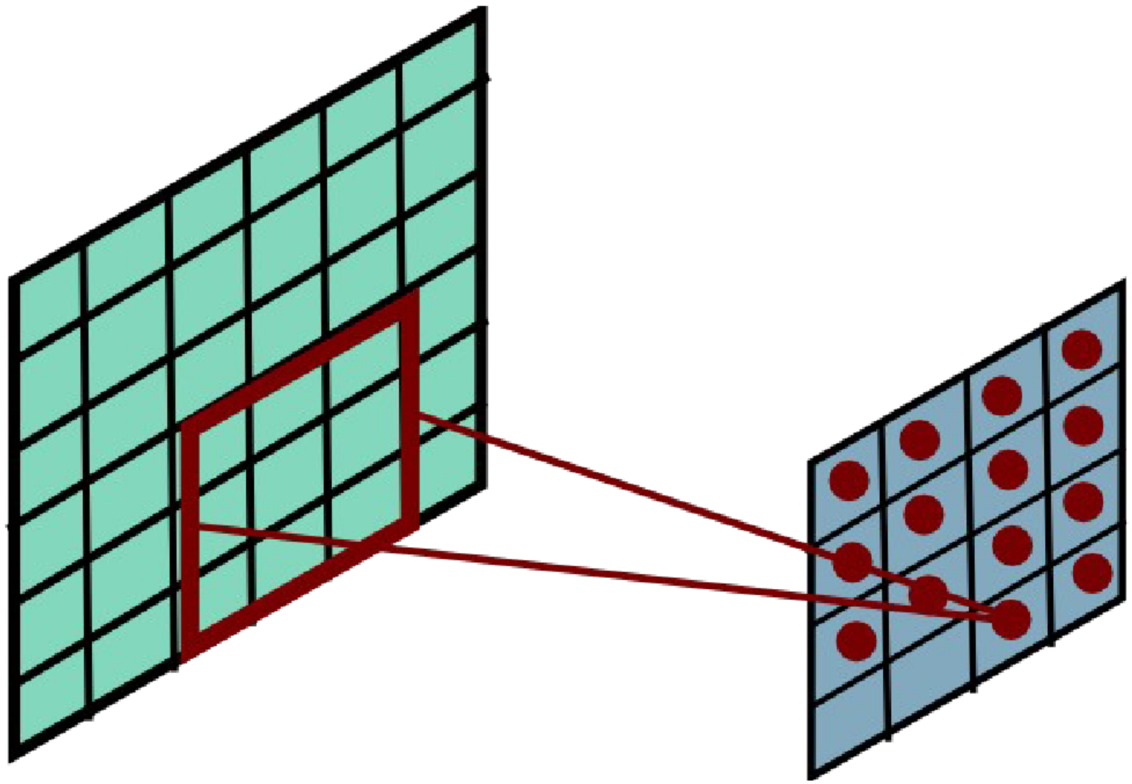


Slides Credit: Deep Learning Tutorial by Marc'Aurelio Ranzato

Ranzato

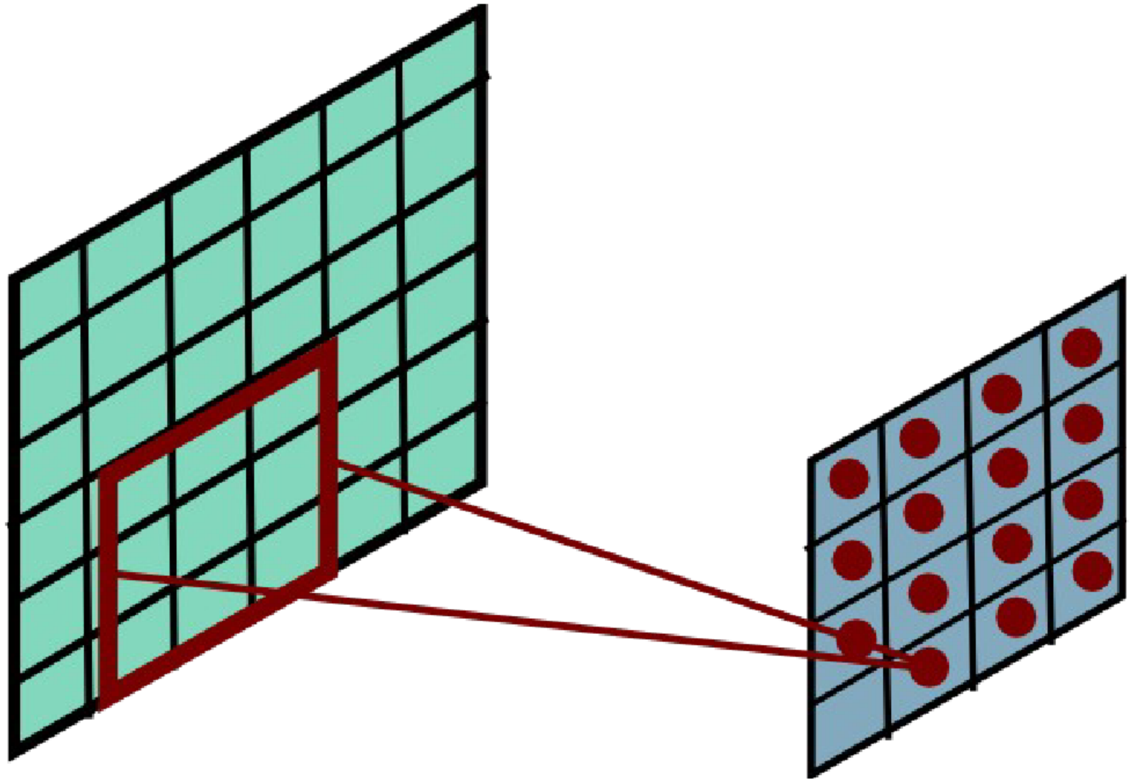


Slides Credit: Deep Learning Tutorial by Marc'Aurelio Ranzato



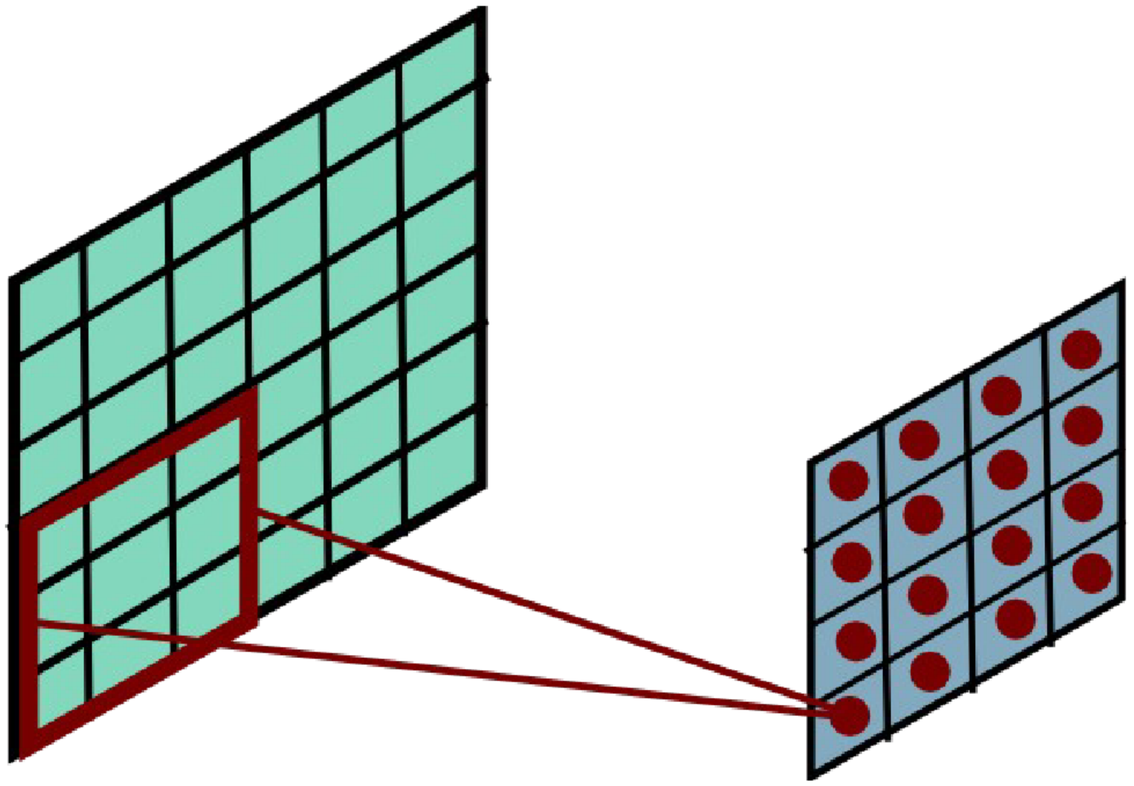
Slides Credit: Deep Learning Tutorial by Marc'Aurelio Ranzato

Ranzato



Slides Credit: Deep Learning Tutorial by Marc'Aurelio Ranzato

Ranzato

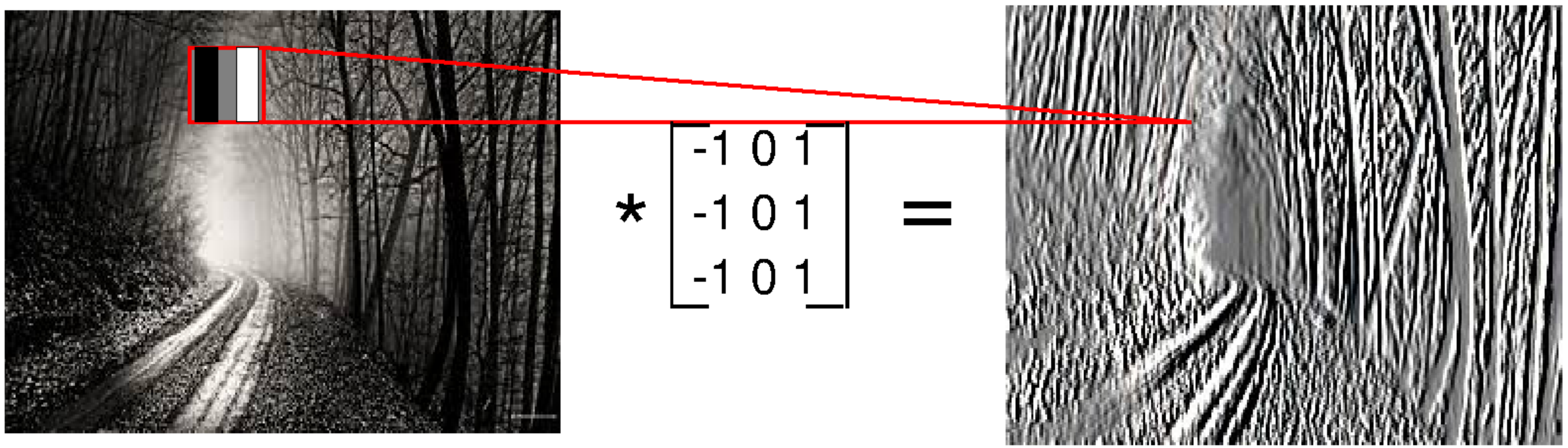


Slides Credit: Deep Learning Tutorial by Marc'Aurelio Ranzato

Ranzato

- Each convolution kernel is a local pattern detector

- Use many of them in a convolutional layer!



Slides Credit: Deep Learning Tutorial by Marc'Aurelio Ranzato

Ranzato

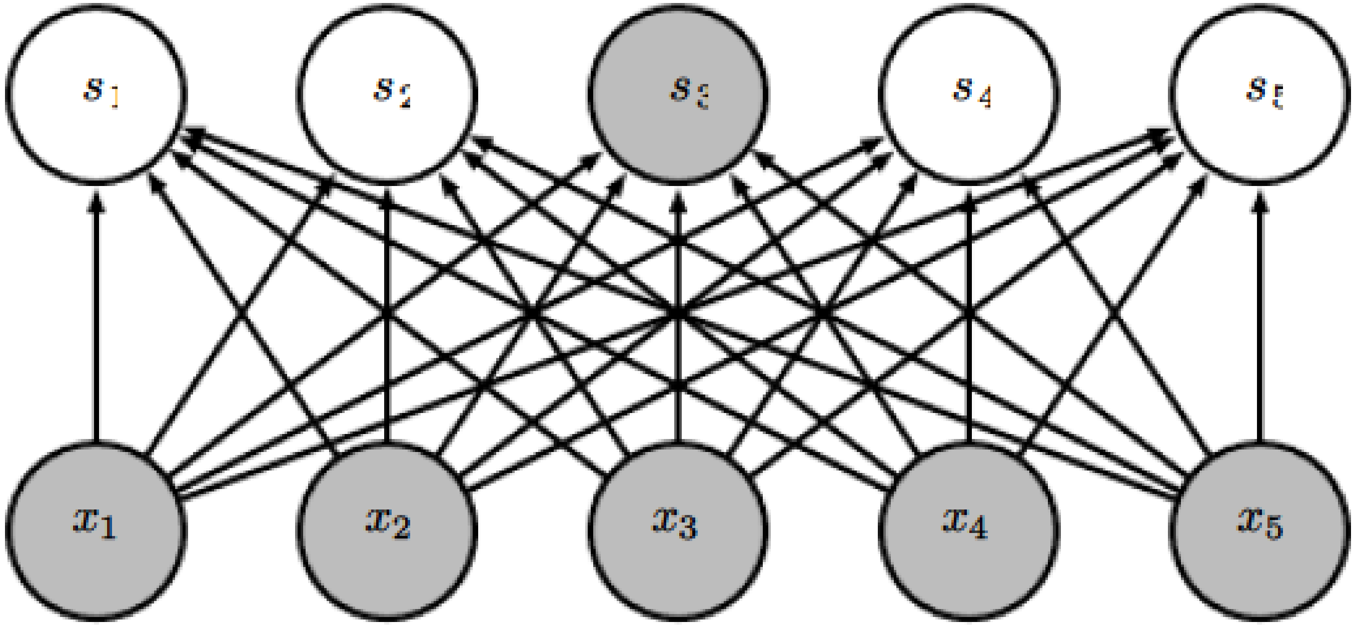
Convolutional neural networks

- Strong empirical application performance

- Convolutional networks: neural networks that use convolution in place of general matrix multiplication in at least one of their layers

Advantage: sparse interaction

Fully connected layer, $m \times n$ edges



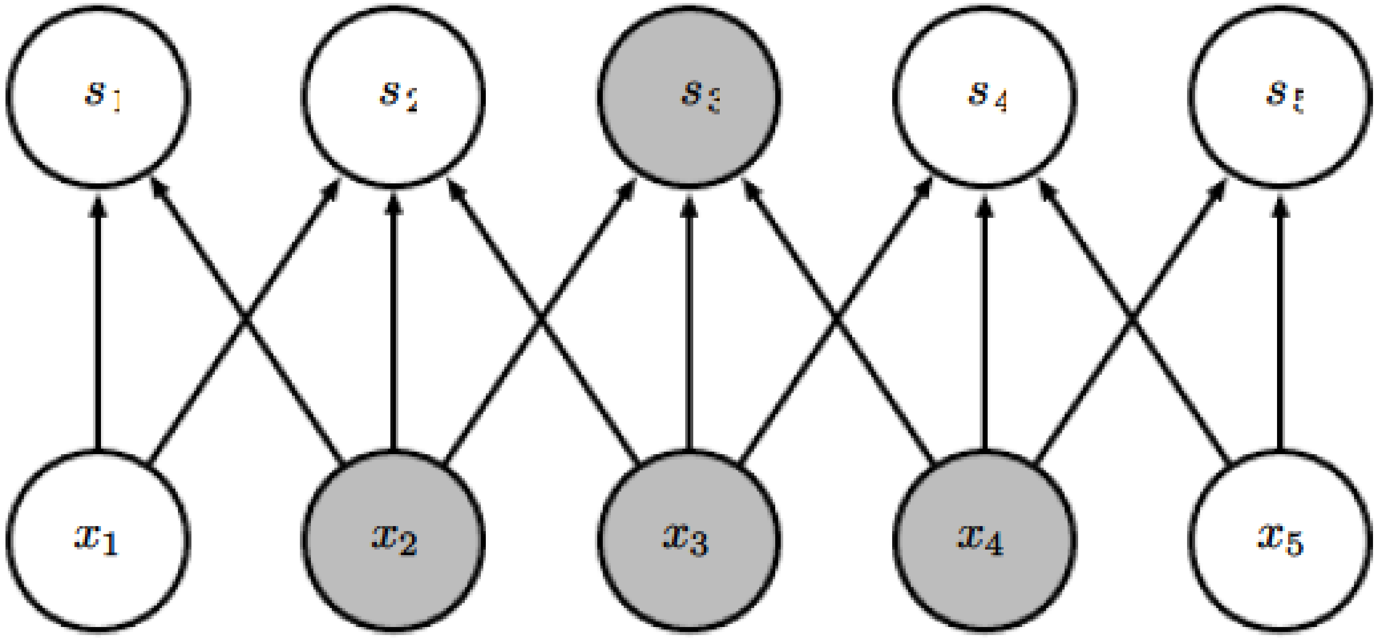
m output nodes

n input nodes

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

Advantage: sparse interaction

Convolutional layer, $\leq m \times k$ edges



m output nodes

k kernel size

n input nodes

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

Advantage: sparse interaction

Multiple convolutional layers: larger receptive field

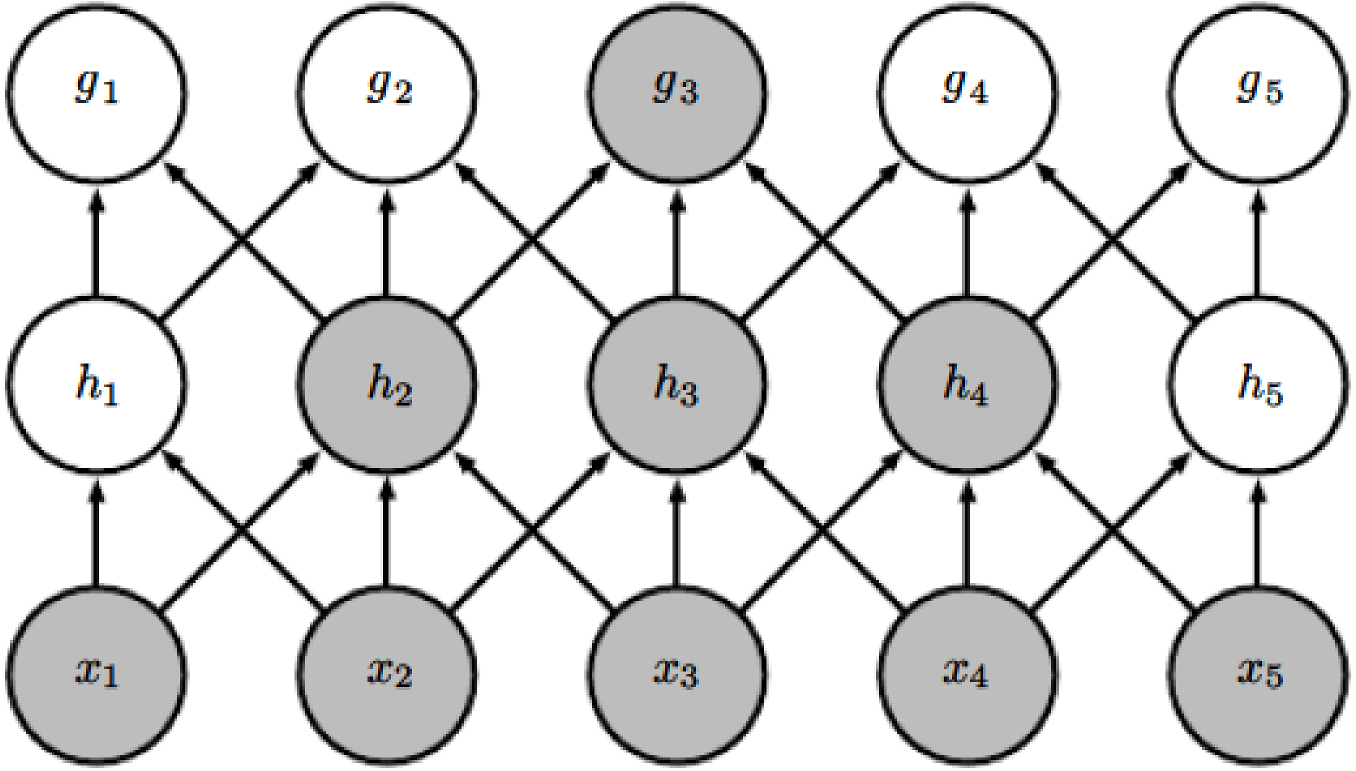
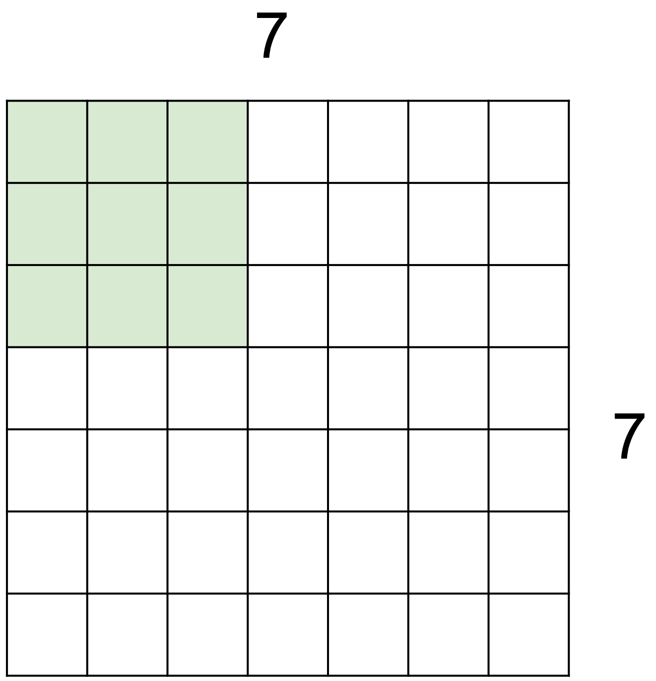


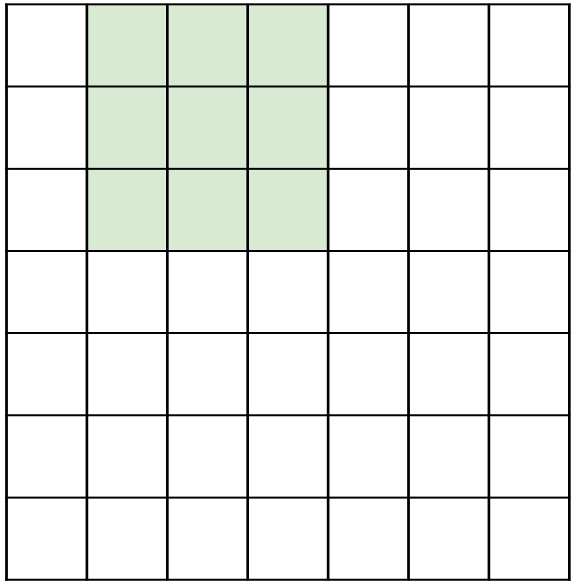
Figure from *Deep Learning,* by Goodfellow, Bengio, and Courville

2D convolution: spatial dimensions



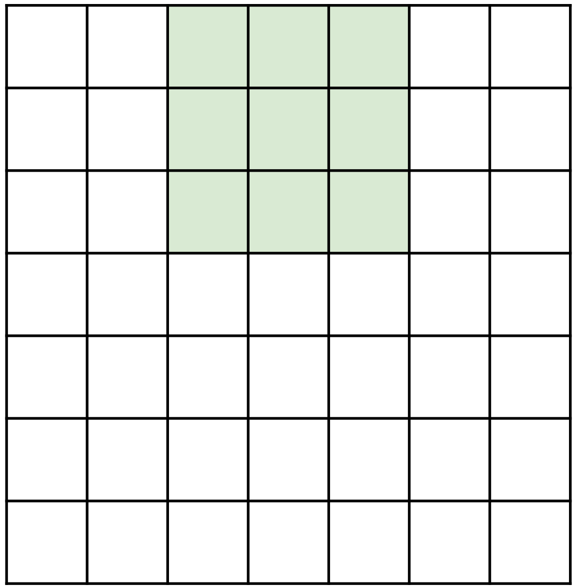
7x7 input (spatially)
assume 3x3 filter

2D convolution: spatial dimensions



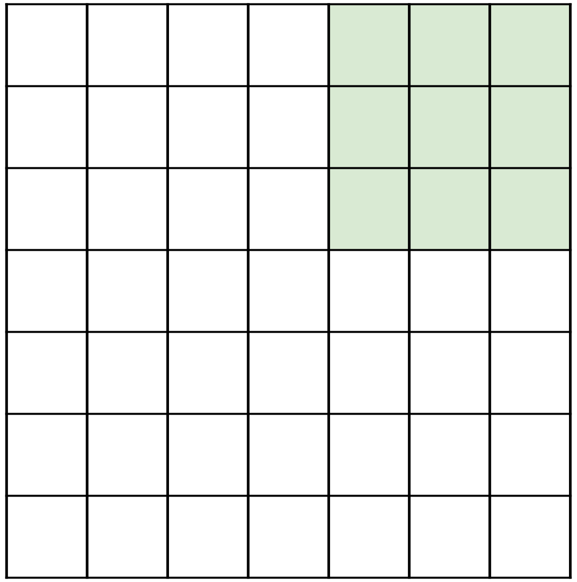
7x7 input (spatially)
assume 3x3 filter

2D convolution: spatial dimensions



7x7 input (spatially)
assume 3x3 filter

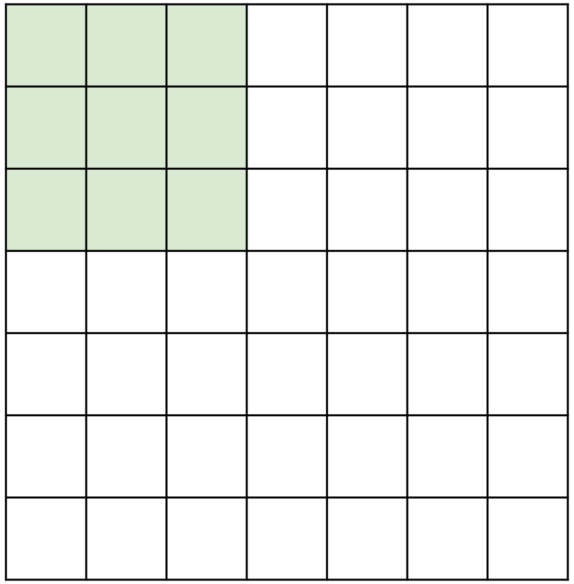
2D convolution: spatial dimensions



7x7 input (spatially)
assume 3x3 filter

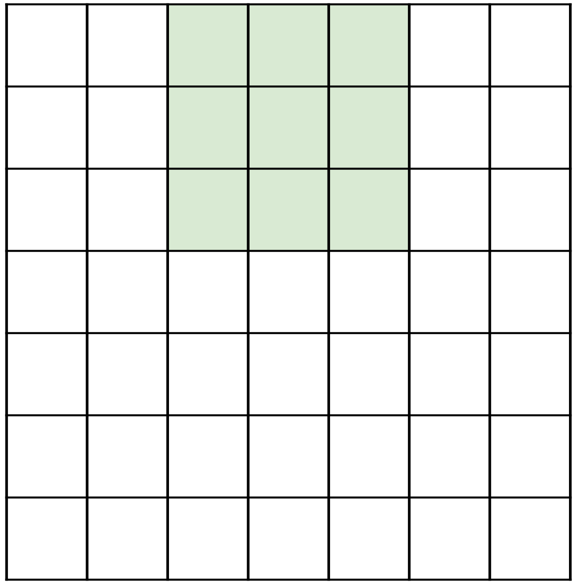
=> 5x5 output

2D convolution: spatial dimensions



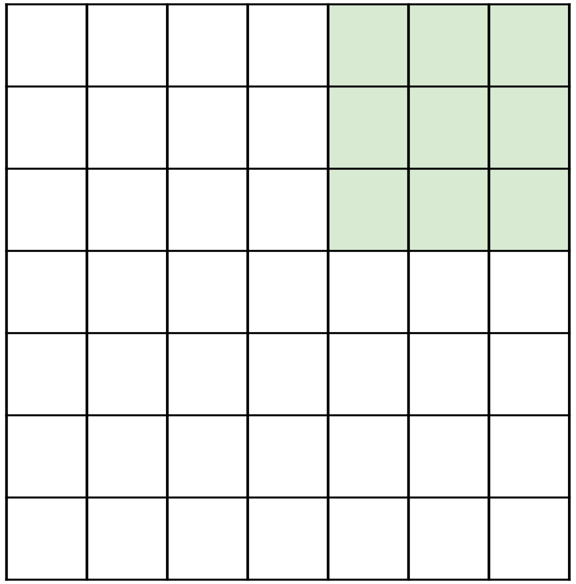
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

2D convolution: spatial dimensions



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

2D convolution: spatial dimensions



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output