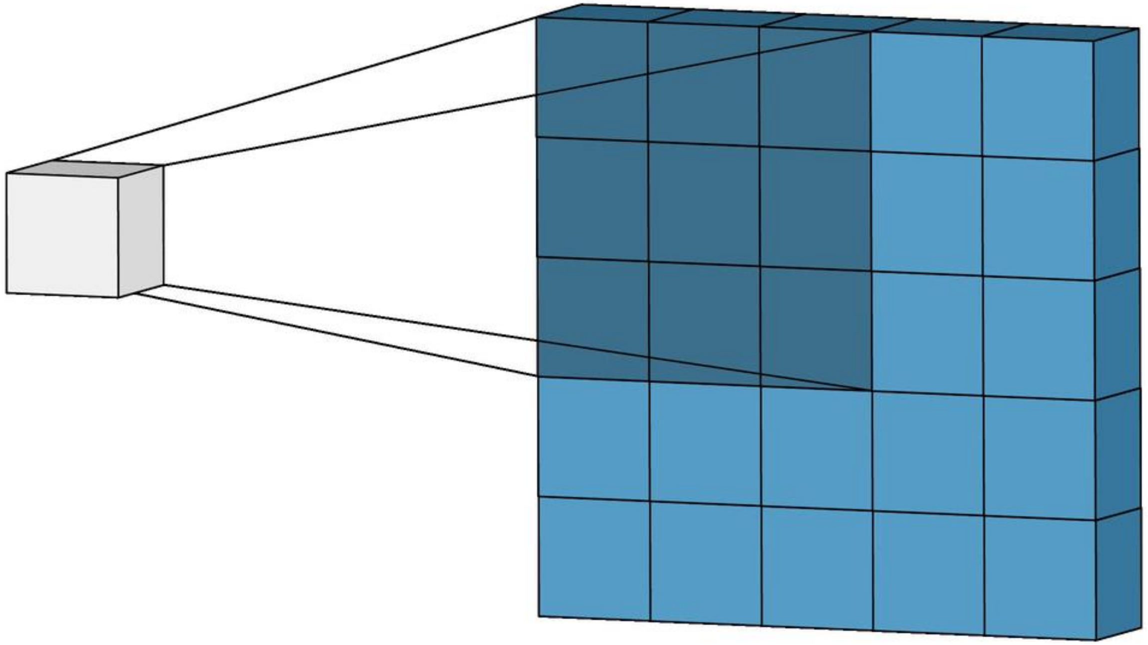
Convolutions & Pooling

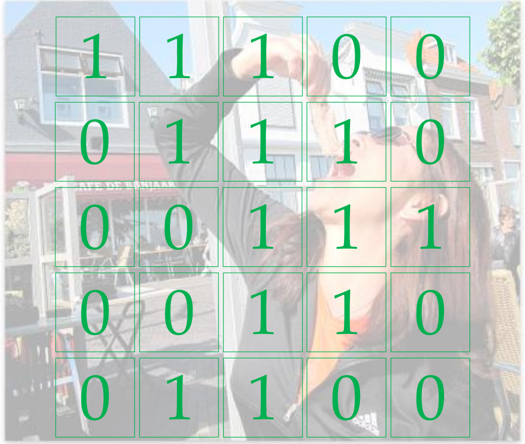


Sliding 2-D parameter matrices/filters → Convolutions

Input

Shared 2-D filters → Convolutions

Original image



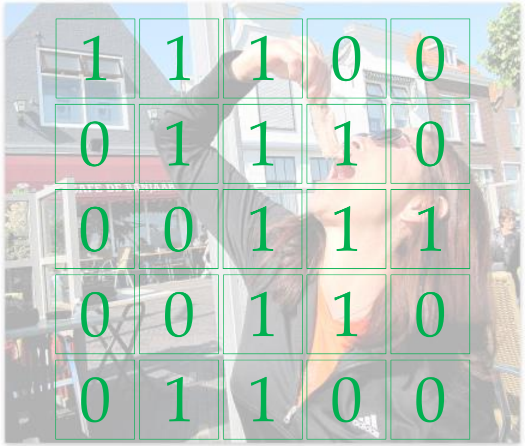
1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Convolutional filter 1

1	0	1
0	1	0
1	0	1

Shared 2-D filters → Convolutions

Original image



Convolutional filter 1

1	0	1
0	1	0
1	0	1

Convolving the image

1 x1	1 x0	1 x1	0	0
0 x0	1 x1	1 x0	1	0
0 x1	0 x0	1 x1	1	1
0	0	1	1	0
0	1	1	0	0

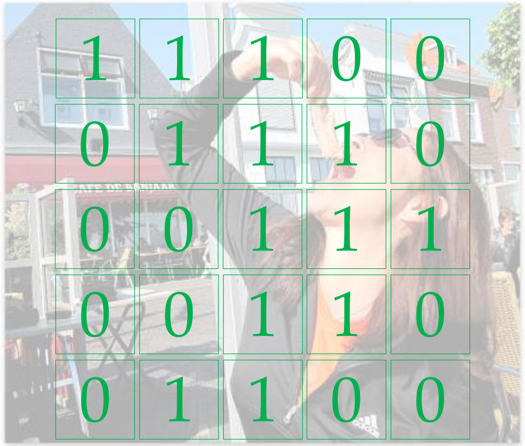
Result

4		

$$I(x,y) * h = \sum_{i=-a}^{a} \sum_{j=-b}^{b} \overbrace{I(x-i, y-j) \cdot h(i,j)}^{\text{Inner product}}$$

Shared 2-D filters → Convolutions

Original image



Convolutional filter 1

1	0	1
0	1	0
1	0	1

Convolving the image

1	1 x1	1 x0	0 x1	0
0	1 x0	1 x1	1 x0	0
0	0 x1	1 x0	1 x1	1
0	0	1	1	0
0	1	1	0	0

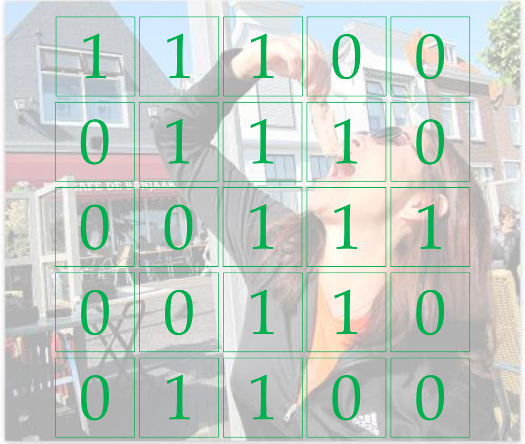
Result

4	3	

$$I(x,y) * h = \sum_{i=-a}^{a} \sum_{j=-b}^{b} \overbrace{I(x-i, y-j) \cdot h(i,j)}^{\text{Inner product}}$$

Shared 2-D filters → Convolutions

Original image



Convolutional filter 1

1	0	1
0	1	0
1	0	1

Convolving the image

1	1	1	0	0
0	1	1	1	0
0	0	1 x1	1 x0	1 x1
0	0	1 x0	1 x1	0 x0
0	1	1 x1	0 x0	0 x1

Result

4	3	4
2	4	3
2	3	4

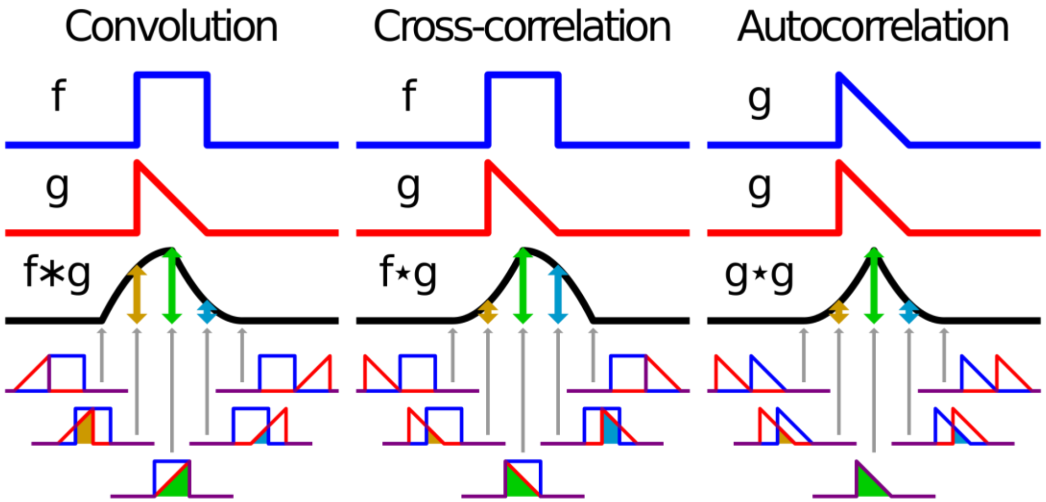
$$I(x,y) * h = \sum_{i=-a}^{a} \sum_{j=-b}^{b} \overbrace{I(x-i, y-j) \cdot h(i,j)}^{\text{Inner product}}$$

Convolution module

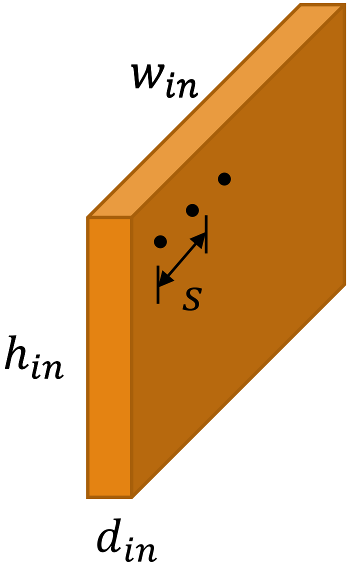
Definition *The convolution of two functions f and g is denoted by $*$ as the integral of the product of the two functions after one is reversed and shifted*

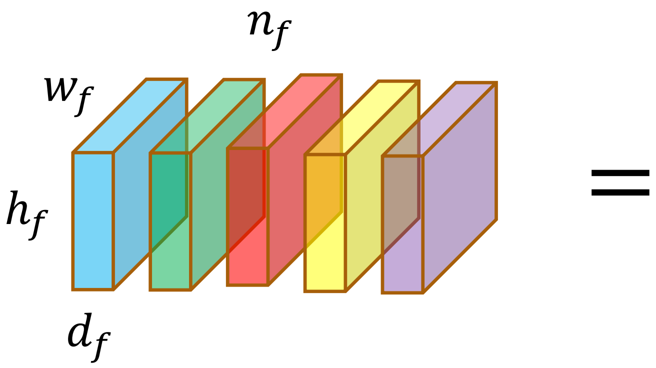
$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau = \int_{-\infty}^{\infty} f(t-\tau)g(\tau)\,d\tau$$

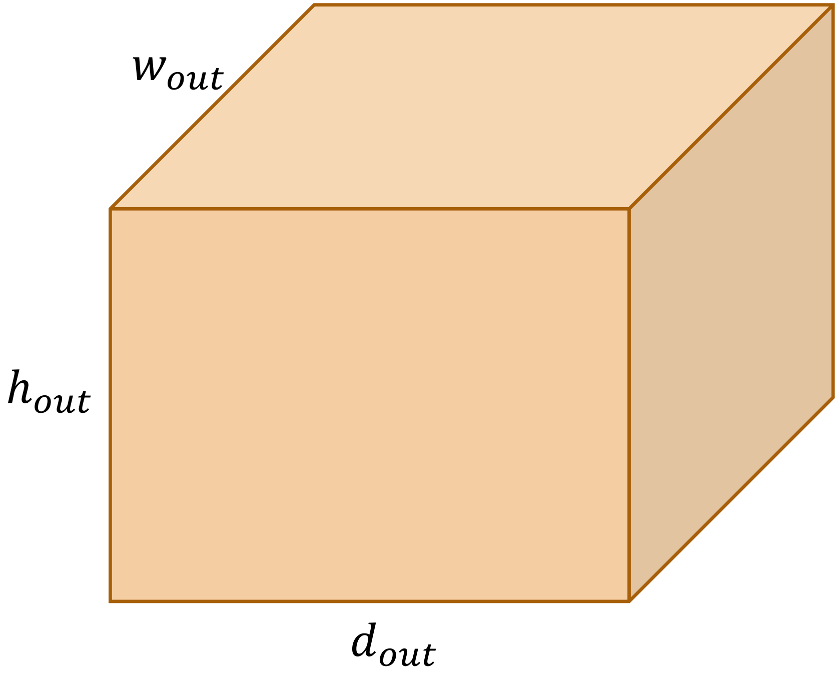
- For images $a_{rc} = \boldsymbol{x} * \boldsymbol{w} = \sum_{i=-a}^{a} \sum_{j=-b}^{b} x_{r-i,c-j} \cdot w_{ij}$



Output dimensions after convolution







$$h_{out} = \frac{h_{in} - h_f}{s} + 1$$

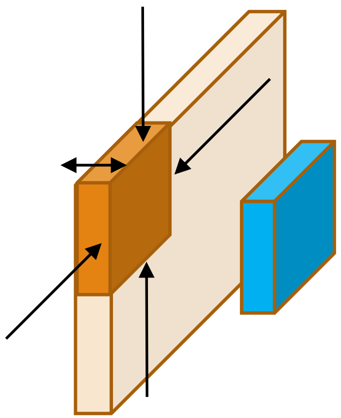
$$w_{out} = \frac{w_{in} - w_f}{s} + 1$$

$$d_{out} = n_f$$

$$d_f = d_{in}$$

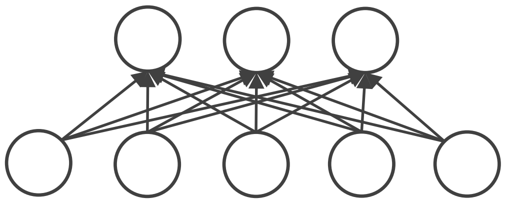
Local connectivity

- Local connectivity
  - Share weights spatially (after translation)
  - Surface-wise local
  - Depth-wise (across channels) global



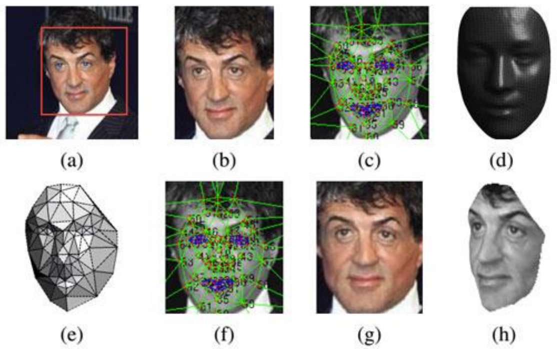
- In MLPs (fully connected), no local connectivity
  - Everything connected to everything
  - No notion of “space”, surface or depth
  - Shuffling the pixels → no difference

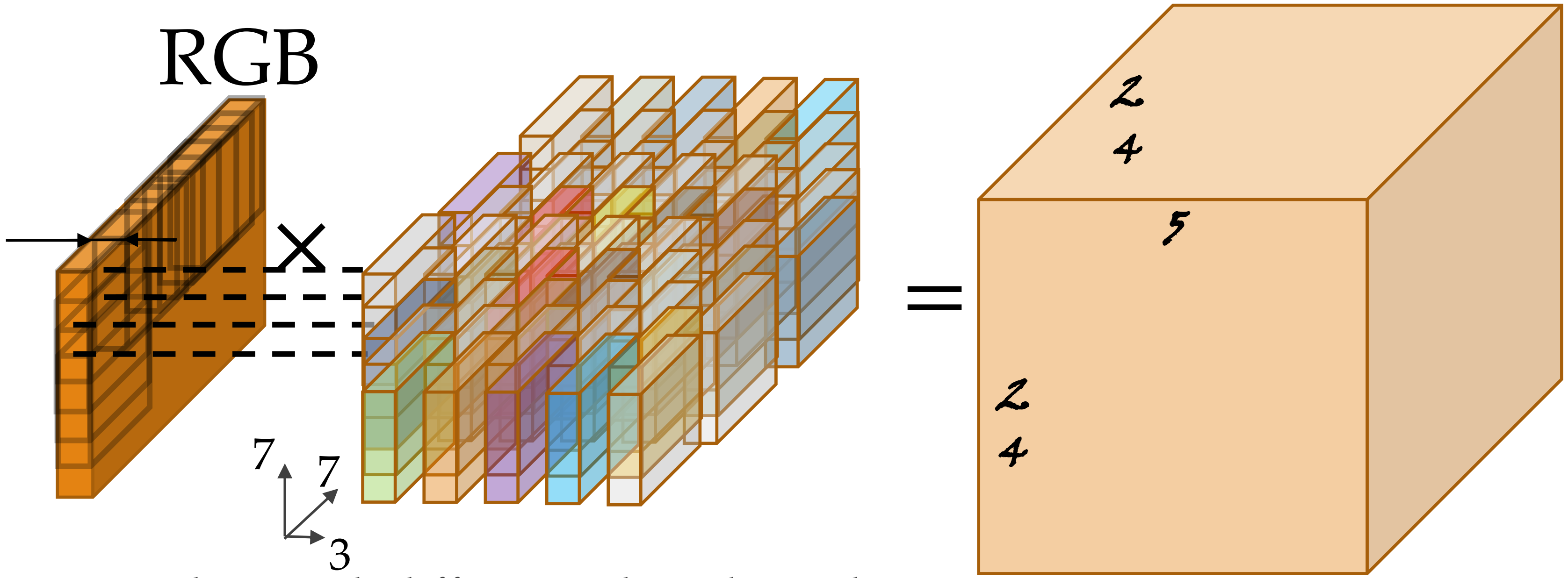
MLP



Local connectivity ≠ Convolutional filters

- Local but ***non-shareable*** filters are also possible
  - Still useful for some applications





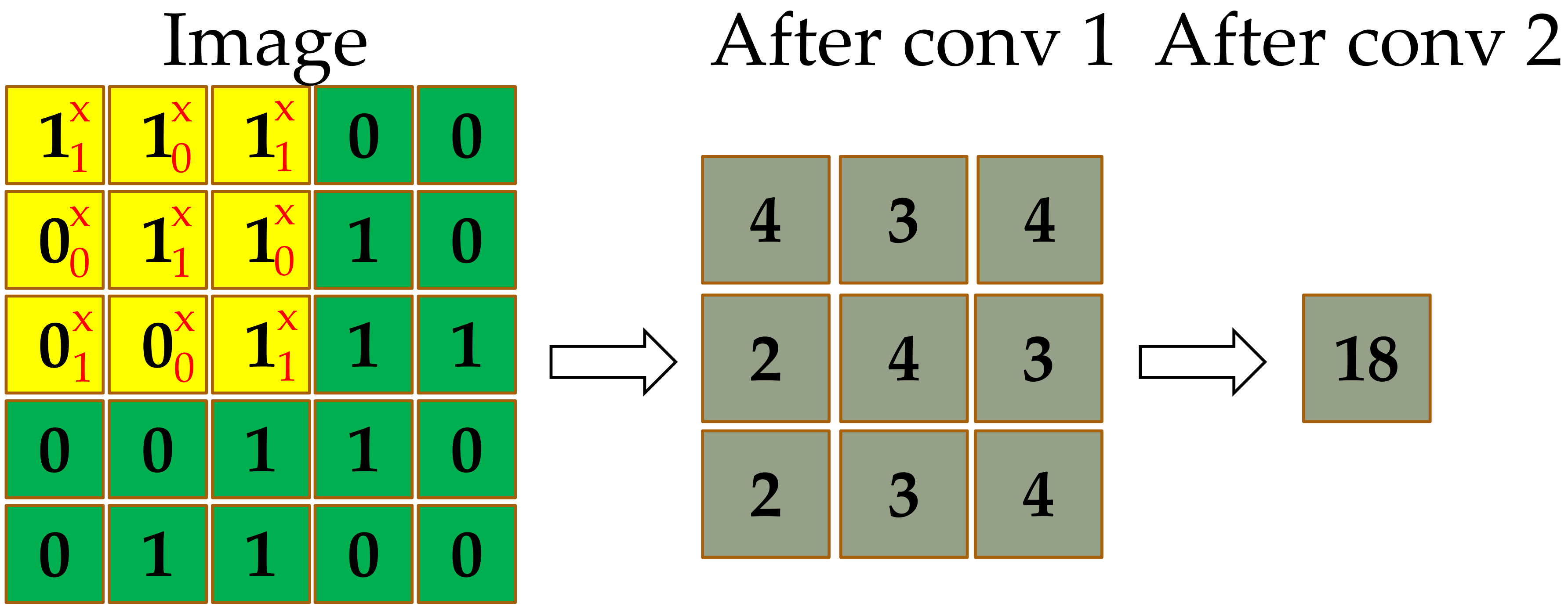
Filters with different values depending on image location and channel ID

Assume the image is 30x30x3.
1 filter every pixel (stride =1)
How many parameters in total?

$$
\begin{array}{rl}
 & \text{24 filters along the } x \text{ axis} \\
 & \text{24 filters along the } y \text{ axis} \\
 & \text{Depth of 5} \\
\times & 7 * 7 * 3 \text{ parameters per filter} \\
\hline
 & 423K \text{ parameters in total}
\end{array}
$$

Convolutions reduce dimensionality

- Our images get smaller and smaller
- We run out of “latent pixels” → not too deep architectures
- Details are lost → recognition accuracy drops



Zero-padding to maintain input dimensionality

- For s = 1, surround the image with $(h_f - 1)/2$ and $(w_f - 1)/2$ layers of 0

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
*
\begin{bmatrix}
0 & 0 & 1 \\
0 & 1 & 1 \\
1 & 1 & 1
\end{bmatrix}
=
\begin{bmatrix}
1 & 1 & 2 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 2 & 1 \\
1 & 0 & 2 & 1 & 0 \\
0 & 1 & 1 & 3 & 0
\end{bmatrix}
$$

Good practices

- Resize the image to have a size in the power of 2
- Stride $s = 1$
- A filter of $(h_f, w_f) = [3 \times 3]$ works quite alright with deep architectures
- Add 1 layer of zero padding

Good practices

- In general avoid combinations of hyper-parameters that do not click
  - E.g. $s = 2$
  - $[h_f \times w_f] = [3 \times 3]$ and
  - image size $[h_{in} \times w_{in}] = [6 \times 6]$
  - $[h_{out} \times w_{out}] = [2.5 \times 2.5]$
  - Programmatically worse, and worse accuracy because borders are ignored

1	1	1	0	0	1
0	1	1	1	0	0
0	0	1	1	1	0
0	0	1	1	0	0
0	1	1	0	0	0
0	1	1	0	0	0

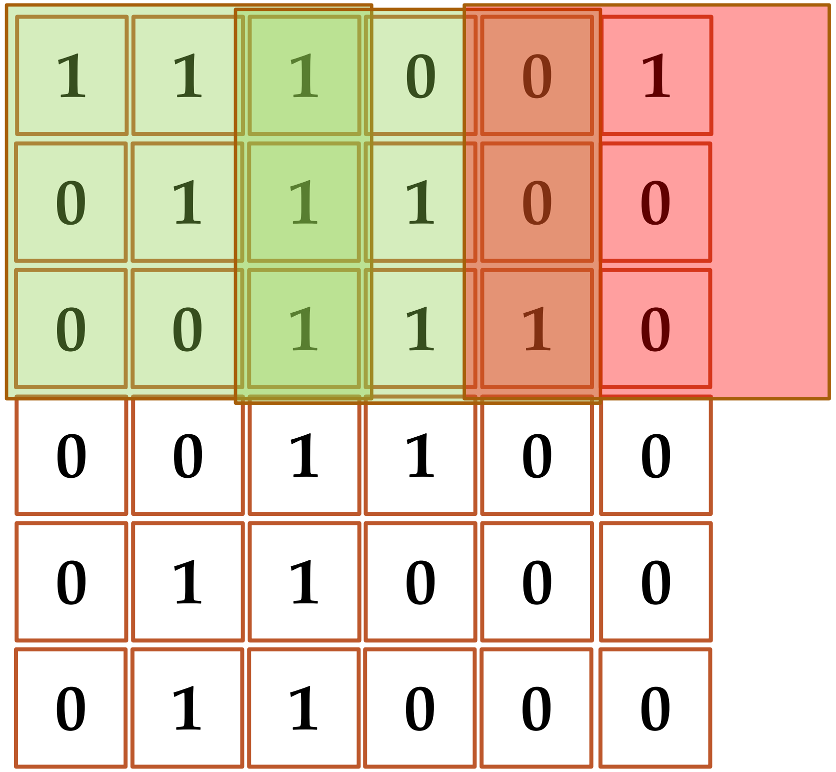
Good practices

- In general avoid combinations of hyper-parameters that do not click
  - E.g. $s = 2$
  - $[h_f \times w_f] = [3 \times 3]$ and
  - image size $[h_{in} \times w_{in}] = [6 \times 6]$
  - $[h_{out} \times w_{out}] = [2.5 \times 2.5]$
  - Programmatically worse, and worse accuracy because borders are ignored

1	1	1	0	0	1
0	1	1	1	0	0
0	0	1	1	1	0
0	0	1	1	0	0
0	1	1	0	0	0
0	1	1	0	0	0

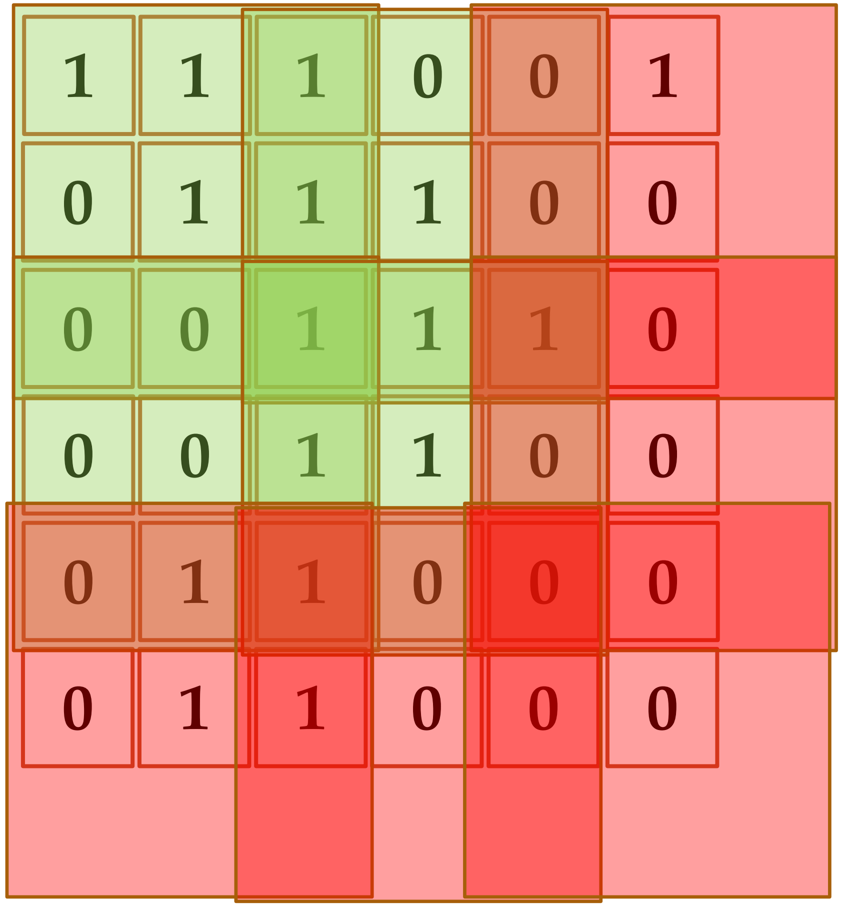
Good practices

- In general avoid combinations of hyper-parameters that do not click
  - E.g. $s = 2$
  - $[h_f \times w_f] = [3 \times 3]$ and
  - image size $[h_{in} \times w_{in}] = [6 \times 6]$
  - $[h_{out} \times w_{out}] = [2.5 \times 2.5]$
  - Programmatically worse, and worse accuracy because borders are ignored



Good practices

- In general avoid combinations of hyper-parameters that do not click
  - E.g. $s = 2$
  - $[h_f \times w_f] = [3 \times 3]$ and
  - image size $[h_{in} \times w_{in}] = [6 \times 6]$
  - $[h_{out} \times w_{out}] = [2.5 \times 2.5]$
  - Programmatically worse, and worse accuracy because borders are ignored



Pooling

- Aggregate multiple values into a single value
  - Invariance to small transformations
  - Reduces feature map size → Faster computations
  - Keeps most important information for the next layer
- Max pooling $\frac{\partial a_{rc}}{\partial x_{ij}} = \begin{cases} 1, & \text{if } i = \mathrm{i}_{\max}, \mathrm{j} = \mathrm{j}_{\max} \\ 0, & \text{otherwise} \end{cases}$
- Average pooling $\frac{\partial a_{rc}}{\partial x_{ij}} = \frac{1}{r \cdot c}$

Input

1	4	3	6
2	1	0	9
2	2	7	7
5	3	3	6

Max pooling

4	9
5	7

Average pooling

2.5	4
2	2.5

No dropout in convolutional layers

- Convolutional filters are much sparser weight arrays
  - No room for co-dependencies and overfitting
  - Dropping out 'single pixels' → too little influence
  - Likely dropout will not contribute
- Also, convolutions by definition capture local correlations
  - If use dropout within convolutions then local correlations get disturbed
  - Convolutions then become ineffective