Deep Learning

- Convolutional Neural Networks 1 -

Eunbyung Park

Assistant Professor

School of Electronic and Electrical Engineering

Eunbyung Park (silverbottlep.github.io)

Convolution

1D Convolution

- Convolution is a mathematical operation on two functions (f , g) that produces a third function $f * g$

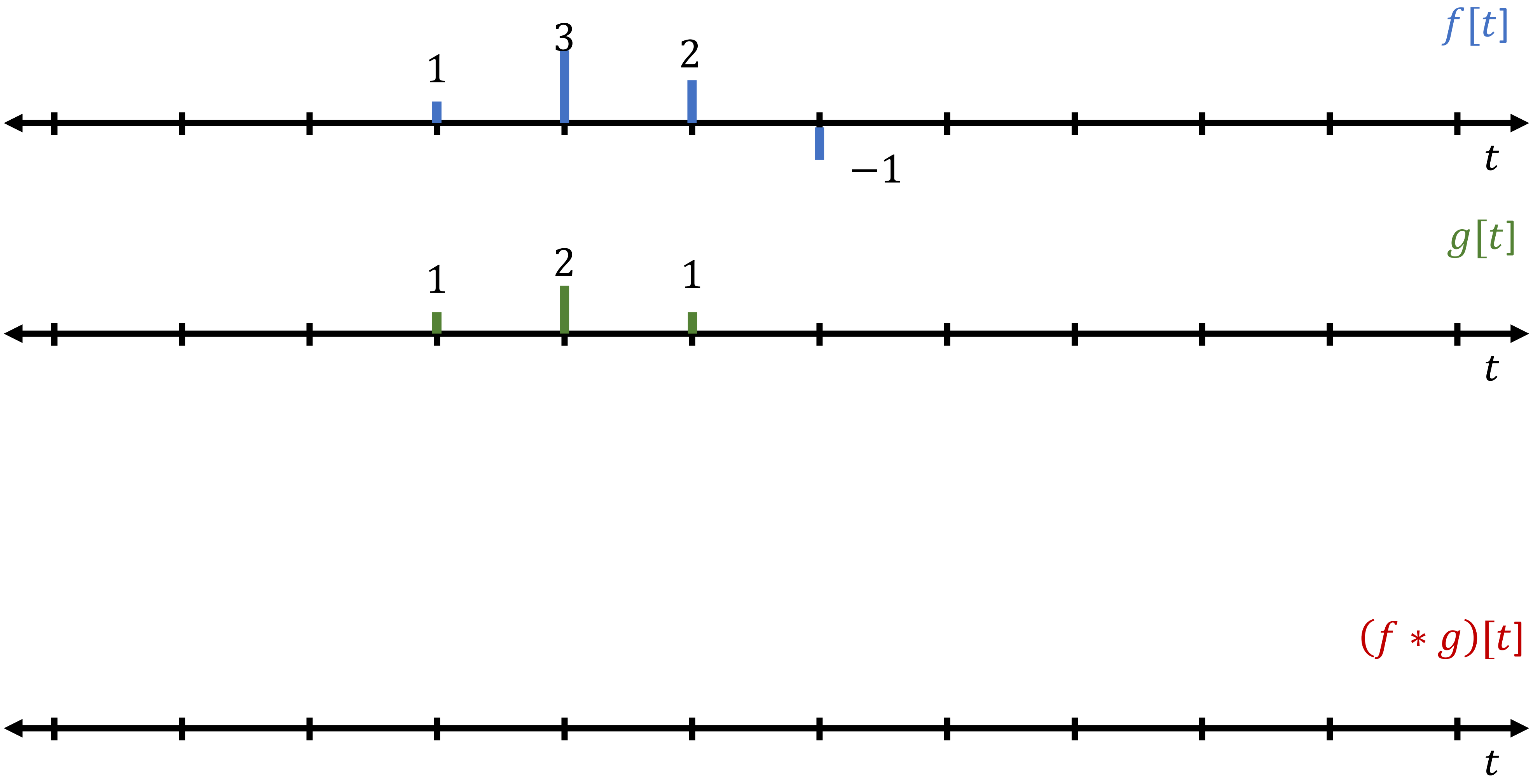
$$(f * g)(t) \coloneqq \int_{-\infty}^{\infty} f(t-\tau)g(\tau)d\tau$$

$$(f * g)[t] \coloneqq \sum_{\tau} f[t-\tau]g[\tau]$$

1D Convolution

$$(f * g)[t] := \sum_{\tau} f[t-\tau]g[\tau]$$

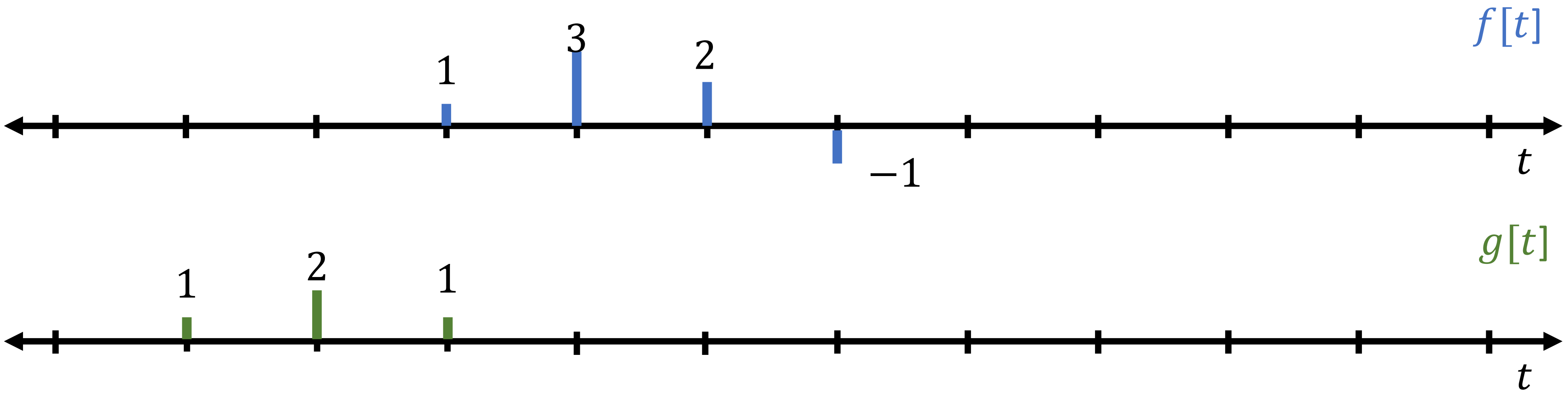
- Flip the filter and sliding



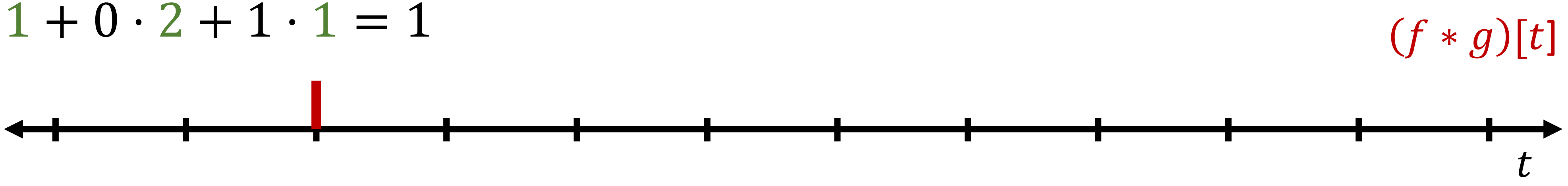
1D Convolution

$$(f * g)[t] \coloneqq \sum_{\tau} f[t-\tau]g[\tau]$$

- Flip the filter and sliding



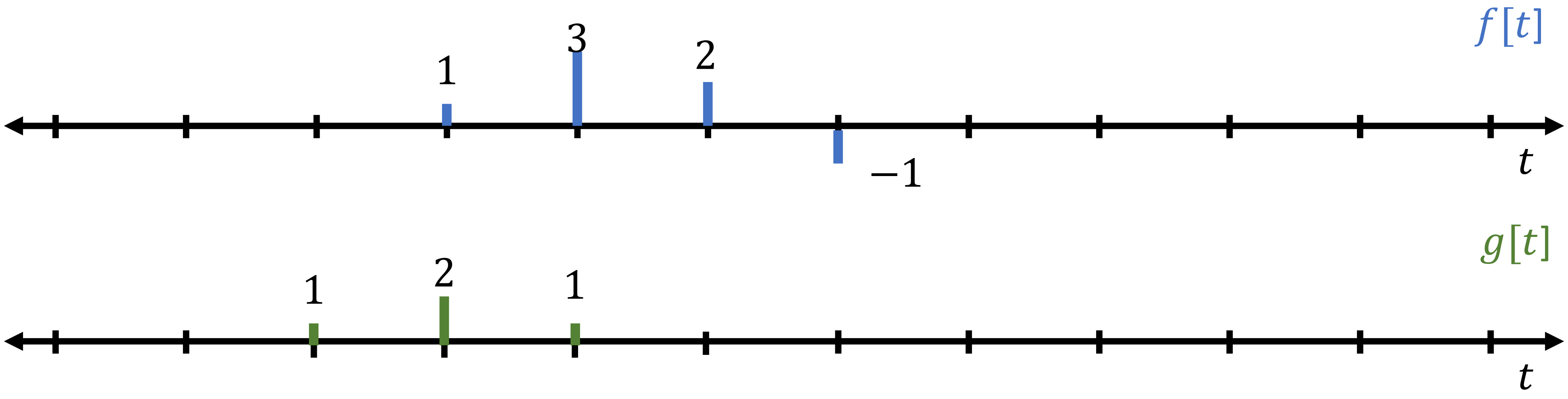
$0 \cdot 1 + 0 \cdot 2 + 1 \cdot 1 = 1$



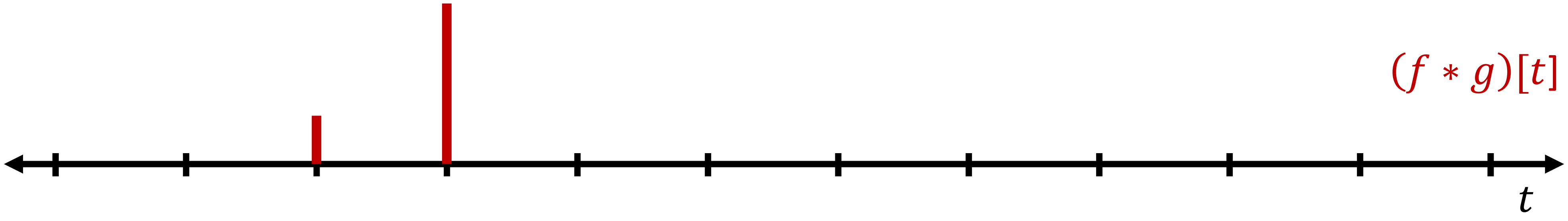
1D Convolution

$$(f * g)[t] \coloneqq \sum_{\tau} f[t-\tau]g[\tau]$$

- Flip the filter and sliding



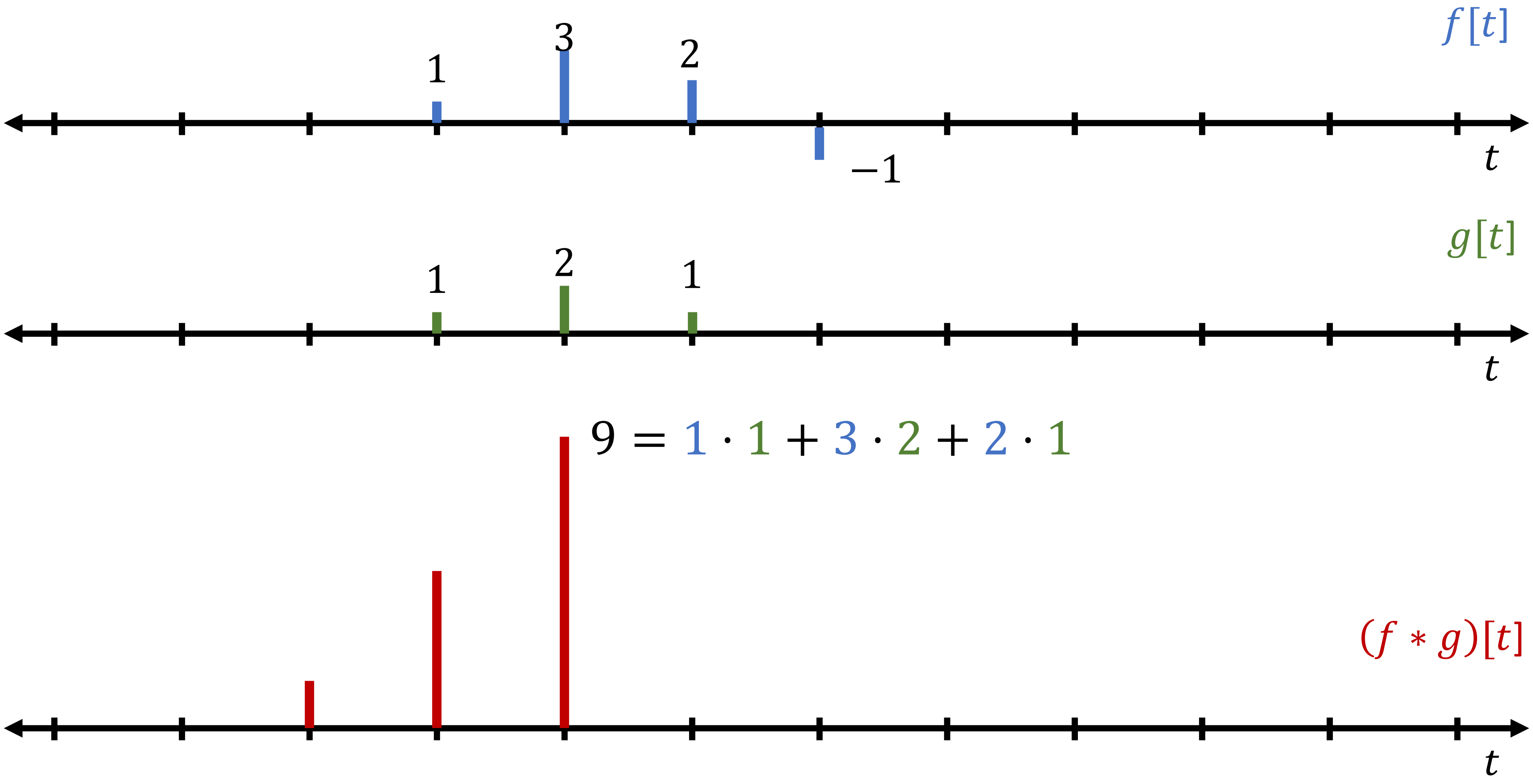
$0 \cdot 1 + 1 \cdot 2 + 3 \cdot 1 = 5$



1D Convolution

$$(f * g)[t] \coloneqq \sum_{\tau} f[t-\tau]g[\tau]$$

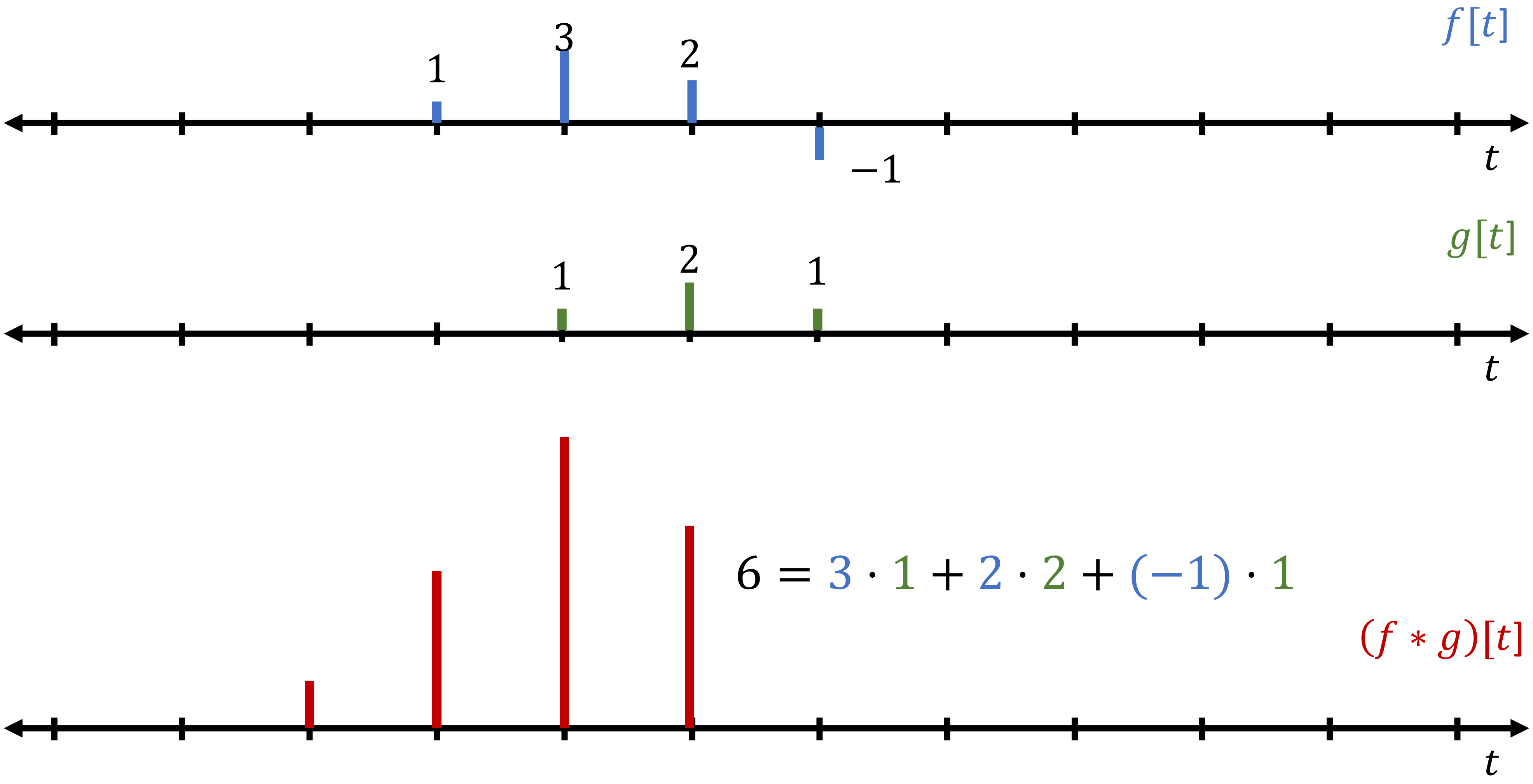
- Flip the filter and sliding



1D Convolution

$$(f * g)[t] \coloneqq \sum_{\tau} f[t-\tau]g[\tau]$$

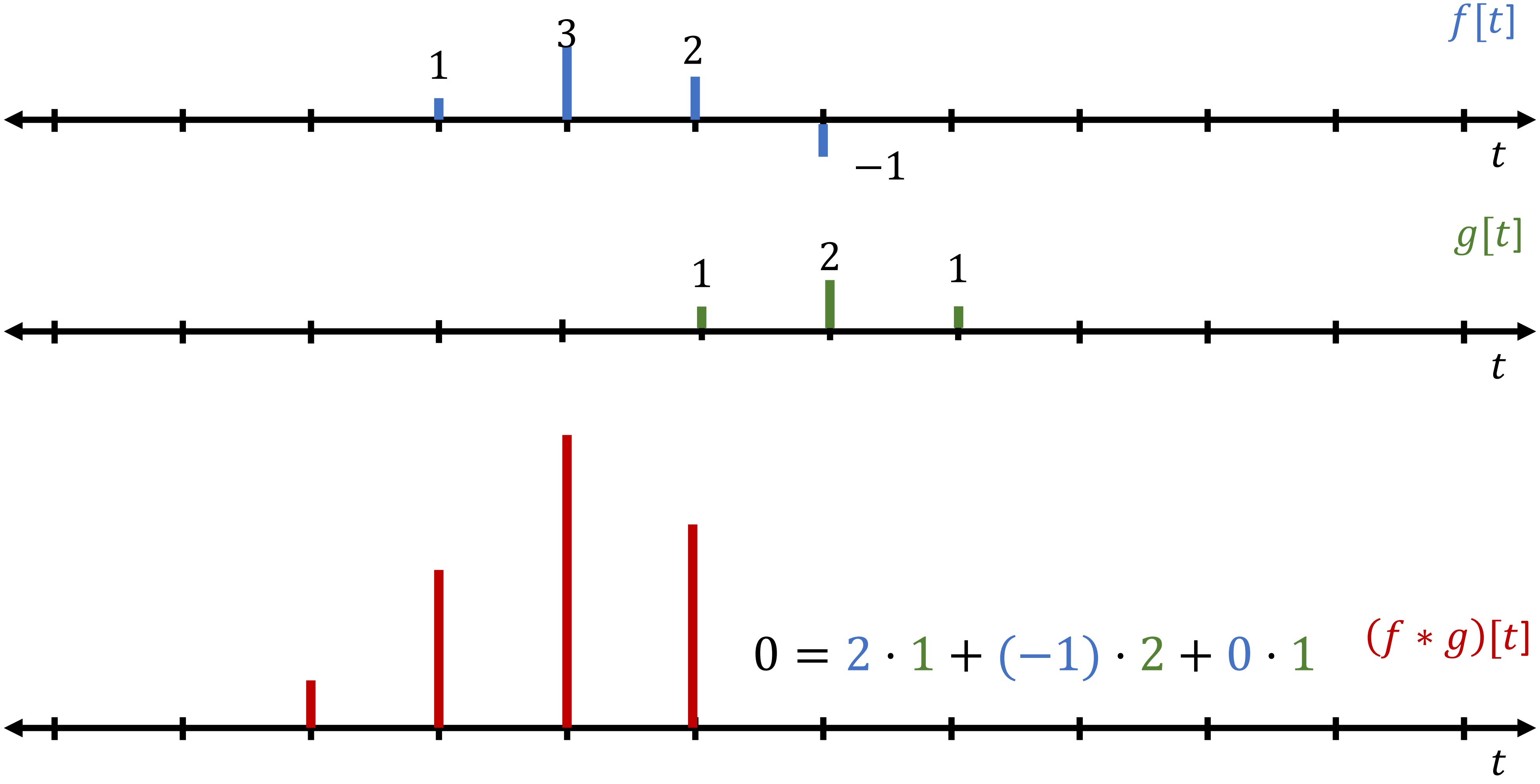
- Flip the filter and sliding



1D Convolution

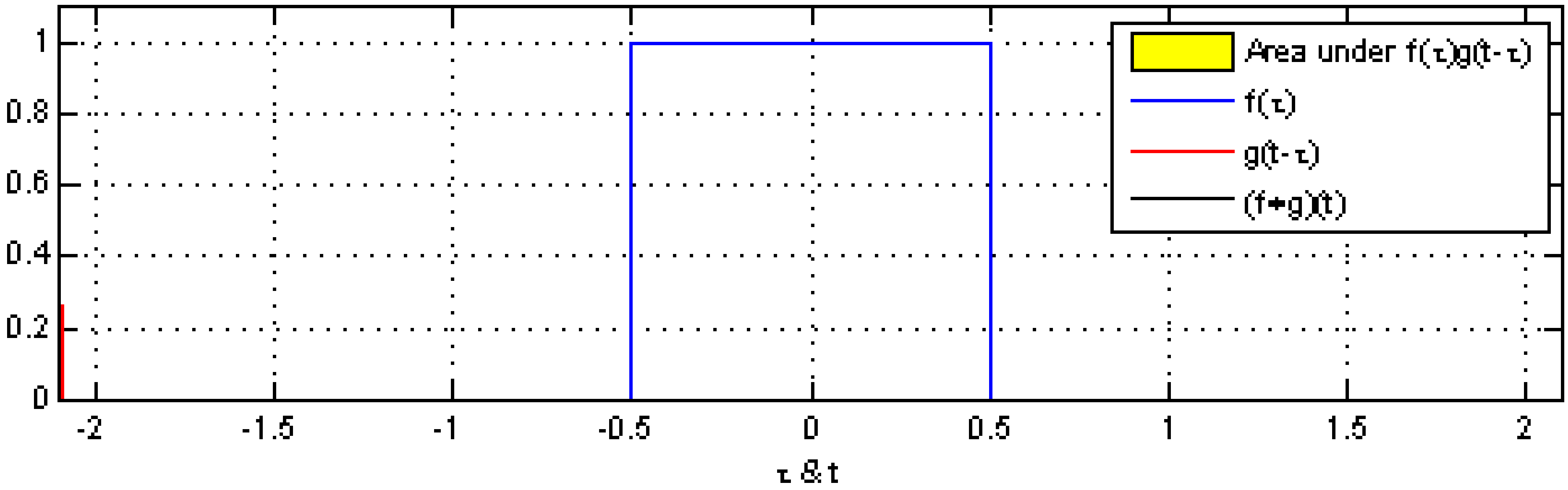
$$(f * g)[t] \coloneqq \sum_{\tau} f[t-\tau]g[\tau]$$

- Flip the filter and sliding



1D Convolution

- Example

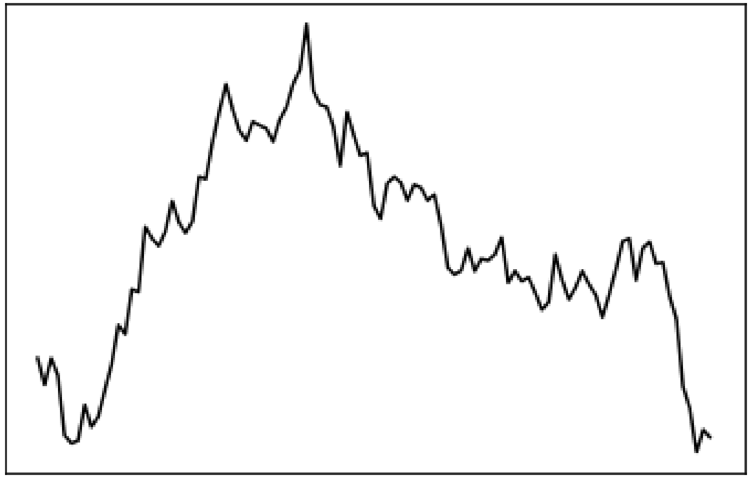


Convolution - Wikipedia

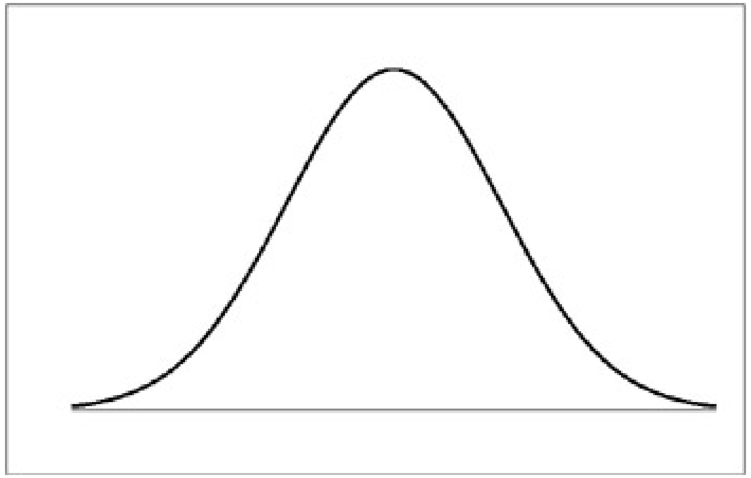
1D Convolution

- Gaussian filter

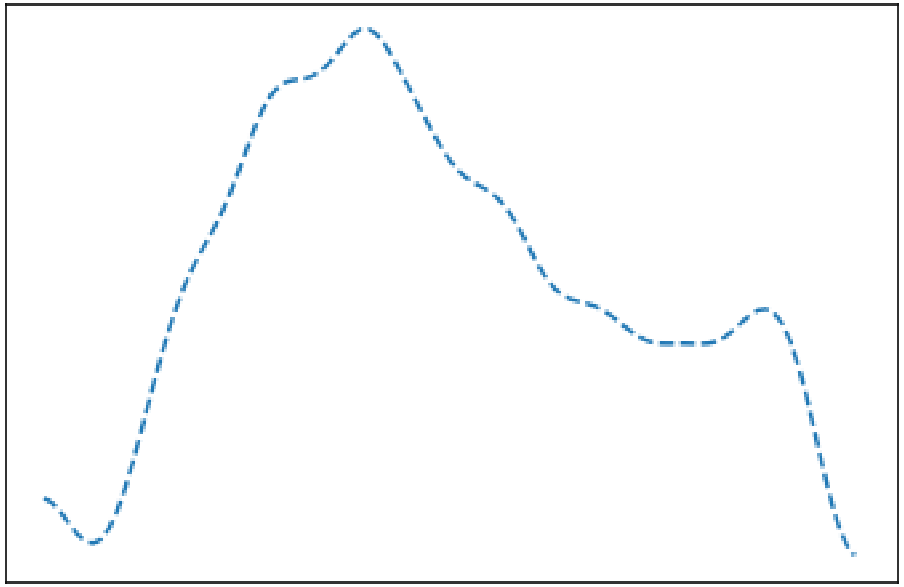
$f(t)$



$g(t)$



$f(t) * g(t)$



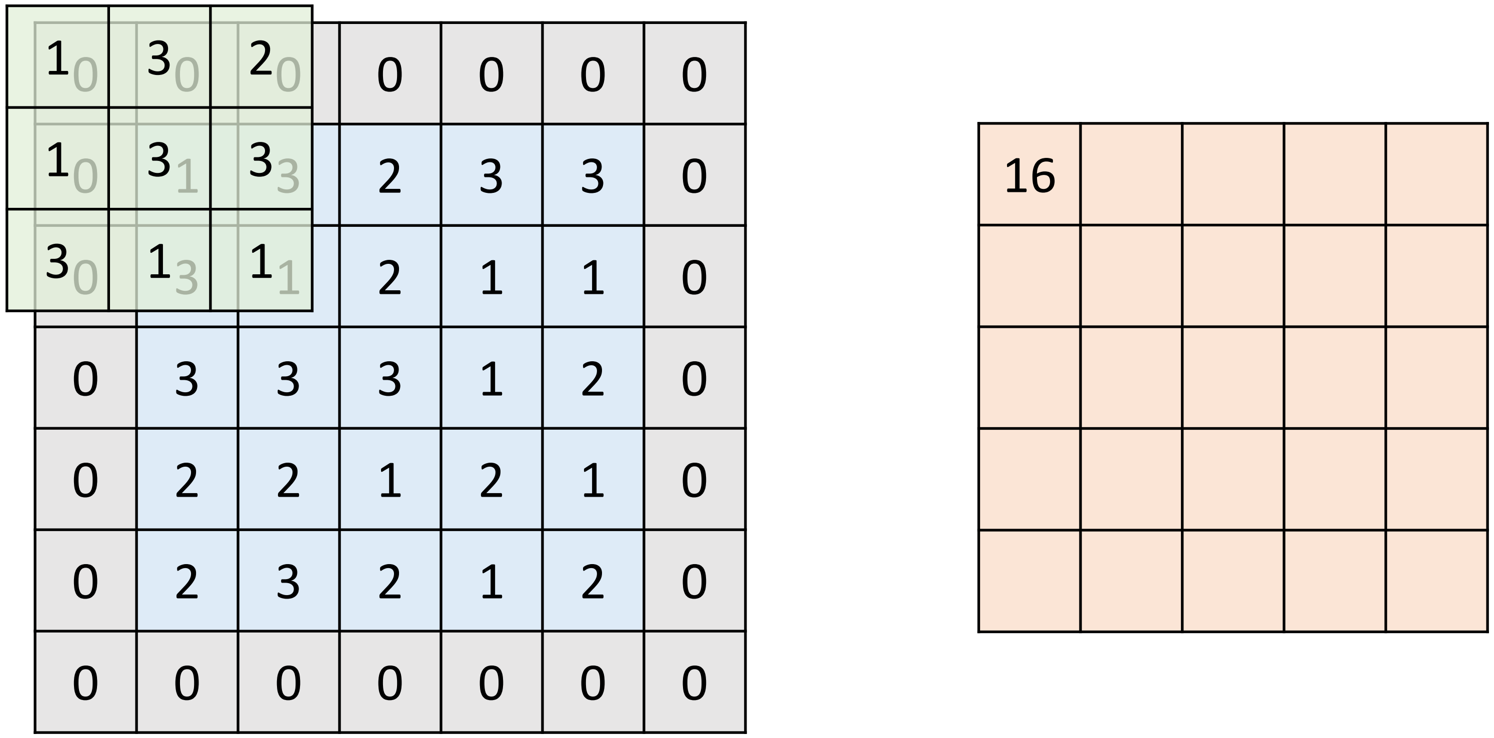
2D Convolution

$$(f * g)(s,t) \coloneqq \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(s-\tau_1, t-\tau_2)g(\tau_1,\tau_2)d\tau_1 d\tau_2$$

$$(f * g)[s,t] \coloneqq \sum_{\tau_1}\sum_{\tau_2} f[s-\tau_1, t-\tau_2]g[\tau_1,\tau_2]$$

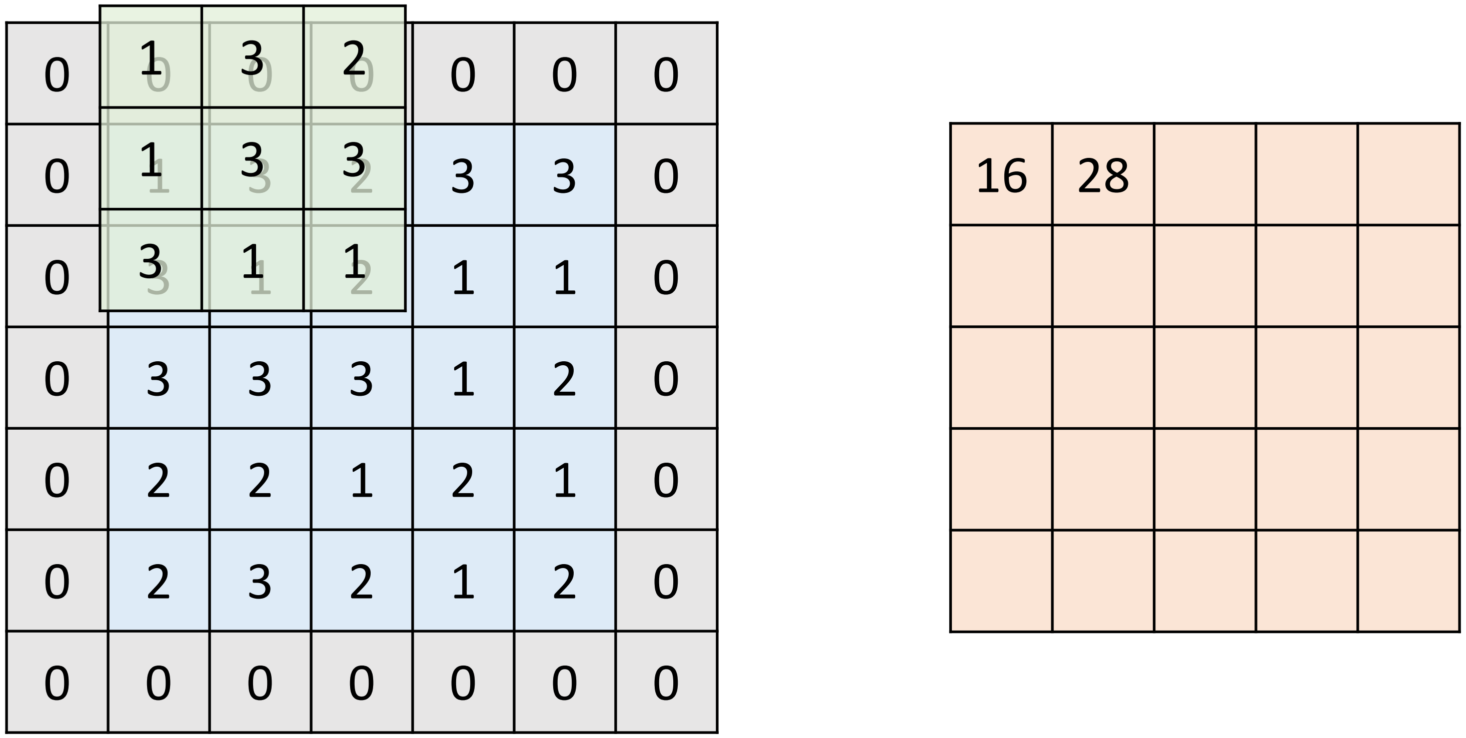
2D Convolution

- One input channel, e.g. gray color image
  - Padding=1, stride=1



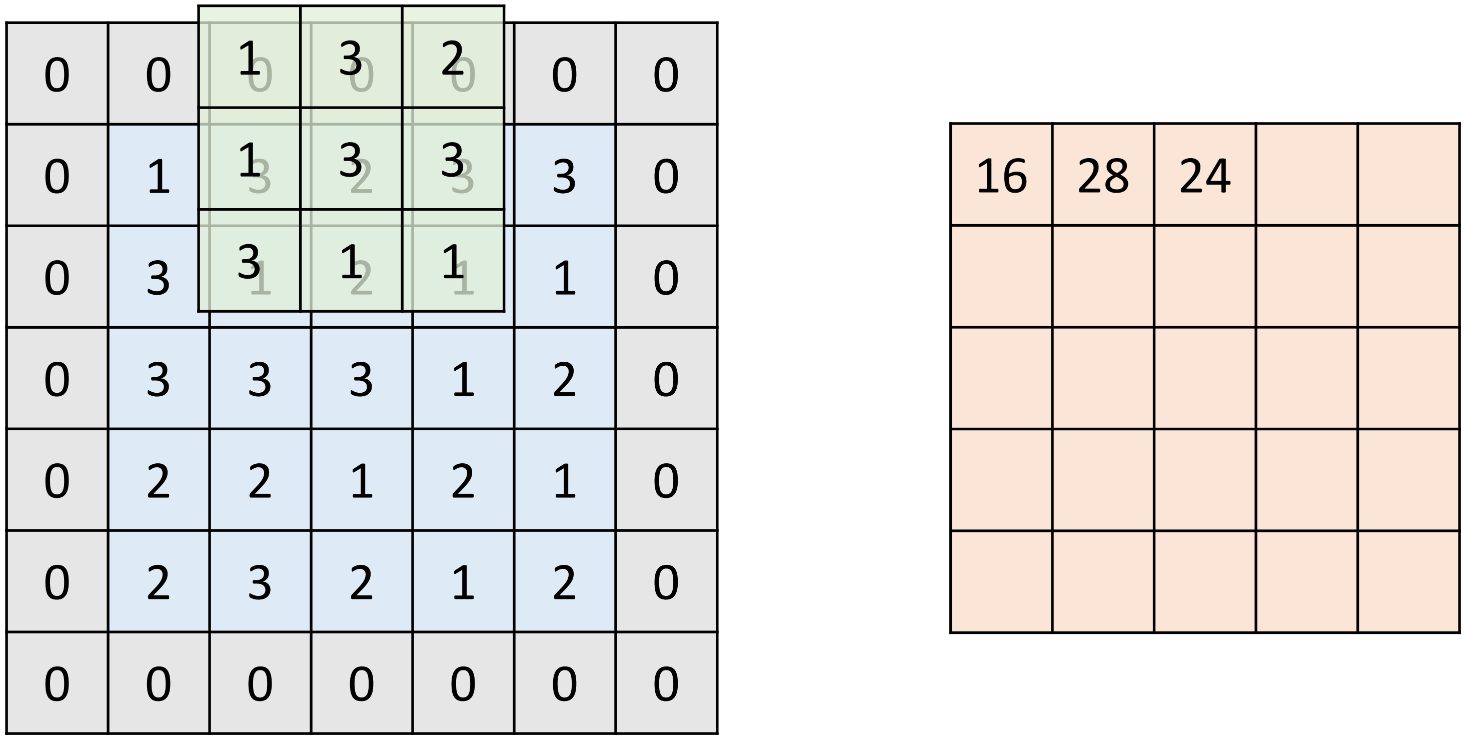
2D Convolution

- One input channel, e.g. gray color image
  - Padding=1, stride=1

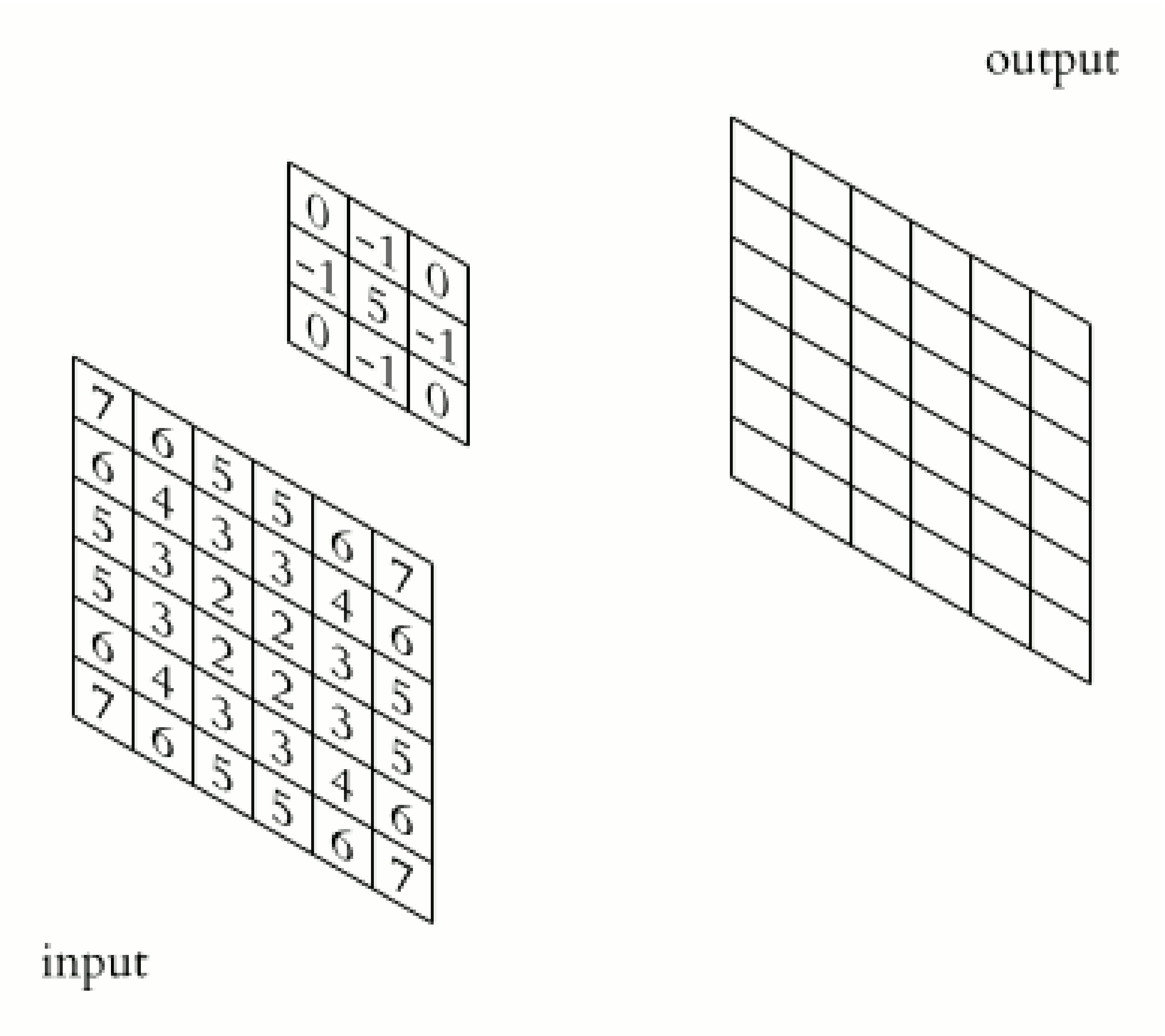


2D Convolution

- One input channel, e.g. gray color image
  - Padding=1, stride=1



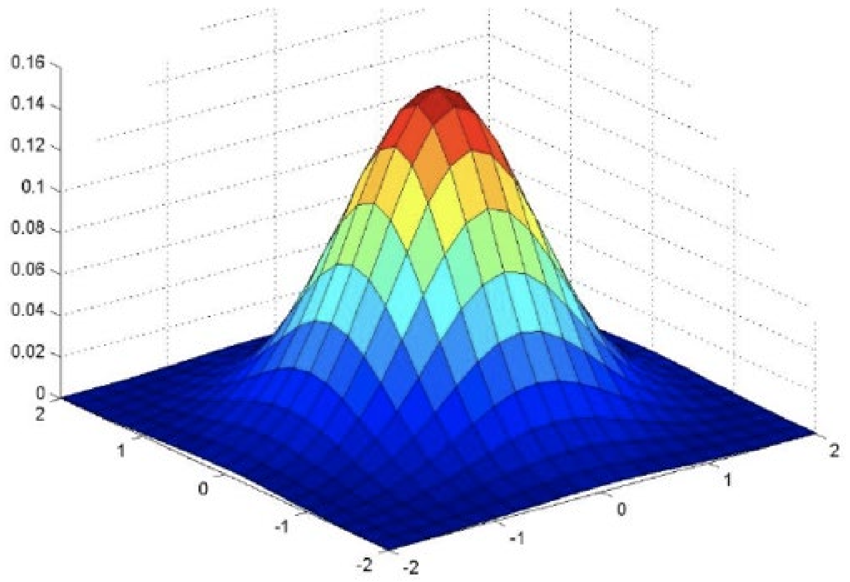
2D Convolution

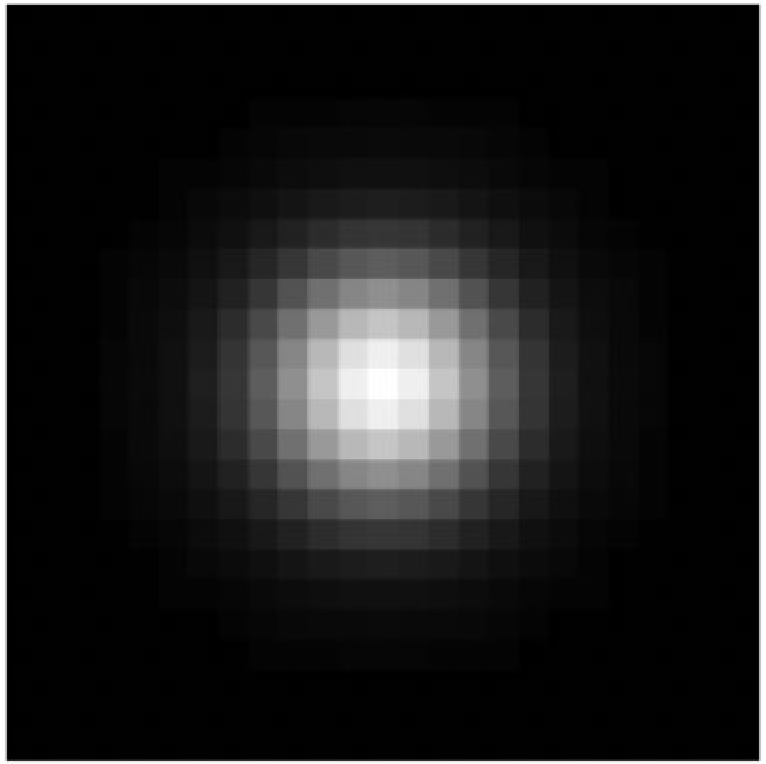


File:2D Convolution Animation.gif - Wikimedia Commons

2D Convolution

- 2D gaussian filter





File:2D Convolution Animation.gif - Wikimedia Commons

2D Convolution

- 2D gaussian filter

Original Image (Left) Vs. Gaussian Filtered Image (Right)

[2-D Gaussian filtering of images - MATLAB imgaussfilt (mathworks.com)](https://www.mathworks.com)

Properties of Convolution

$f * g = g * f$ — Commutative

$(f * g) * h = f * (g * h)$ — Associative

$(f + g) * h = (f * h) + (g * h)$

Linear

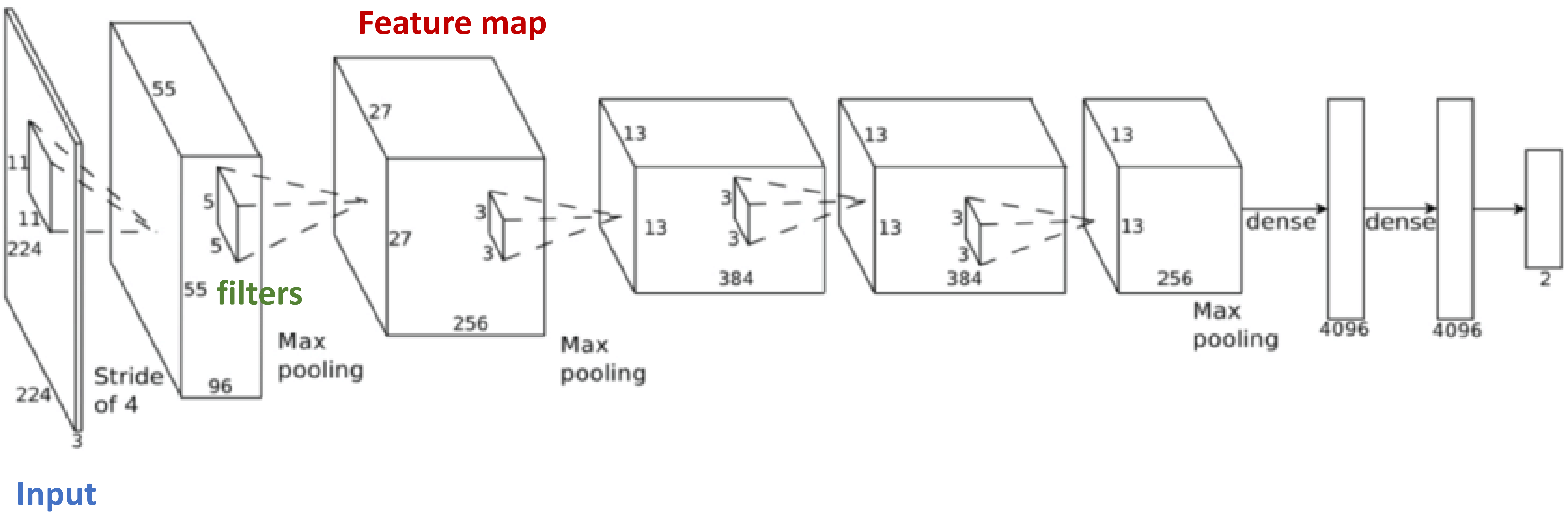
$cf * h = c(f * h)$

$(L_t f) * h = L_t(f * h)$ — Translation equivariance

Translate f by t

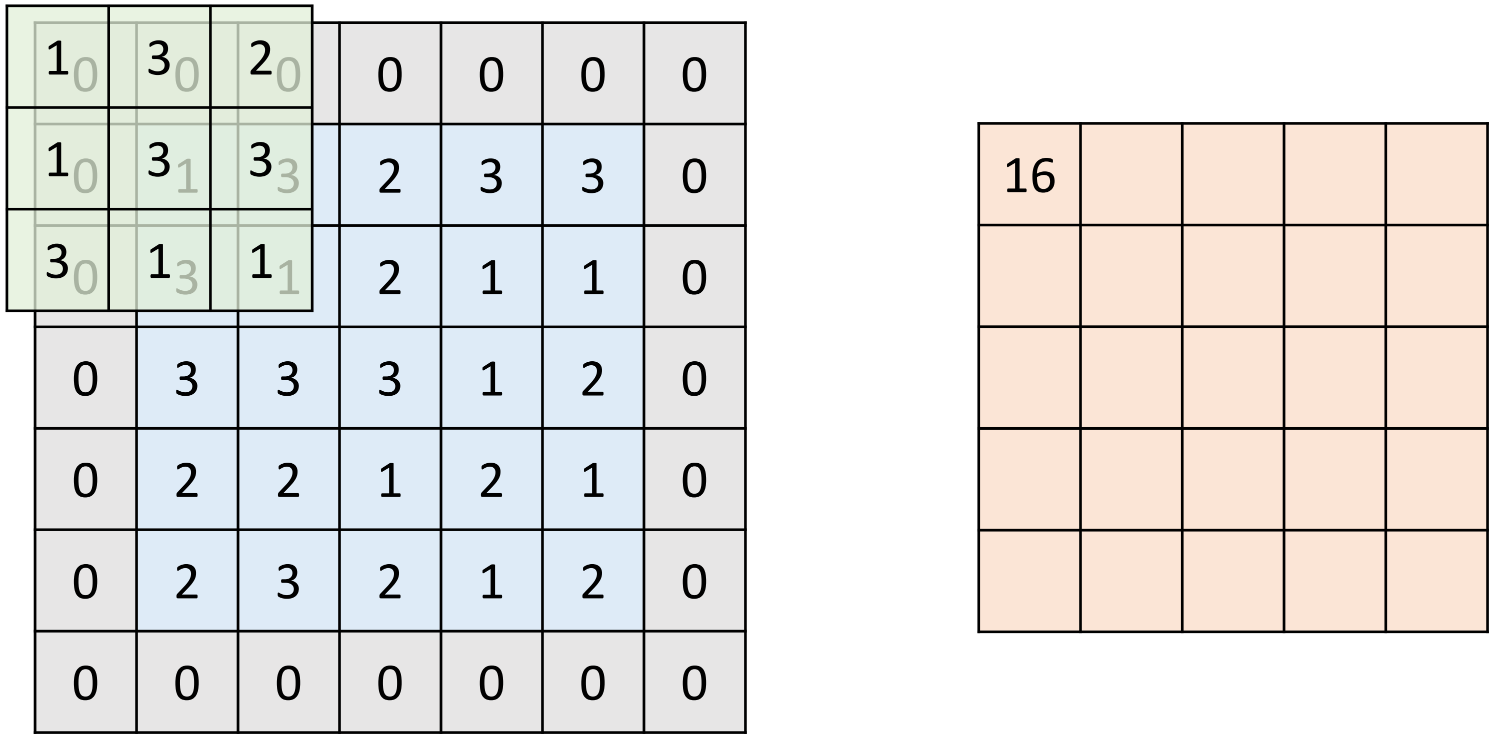
Convolutional Neural Networks

Convolutional Neural Network



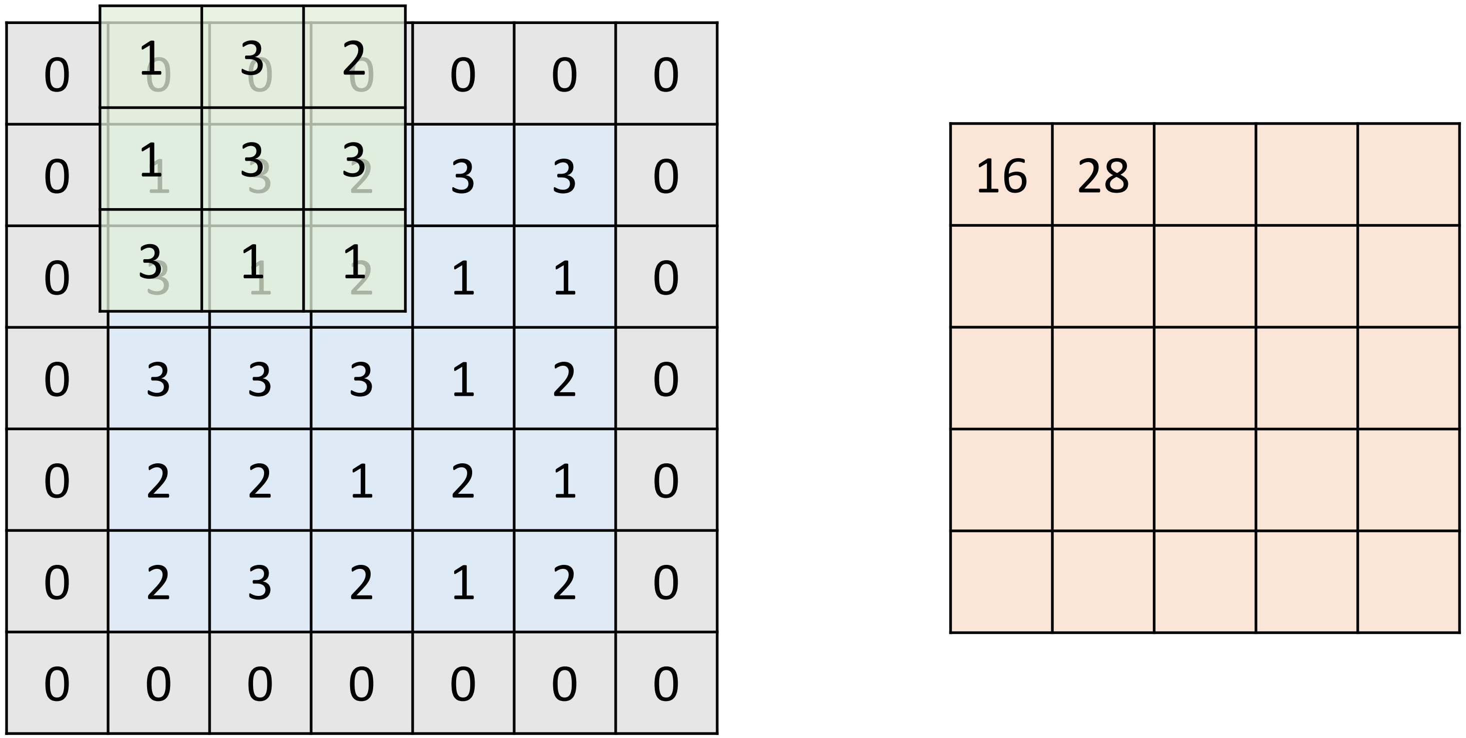
2D Convolution

- Input_channel=1, output_channel=1, padding=1, stride=1



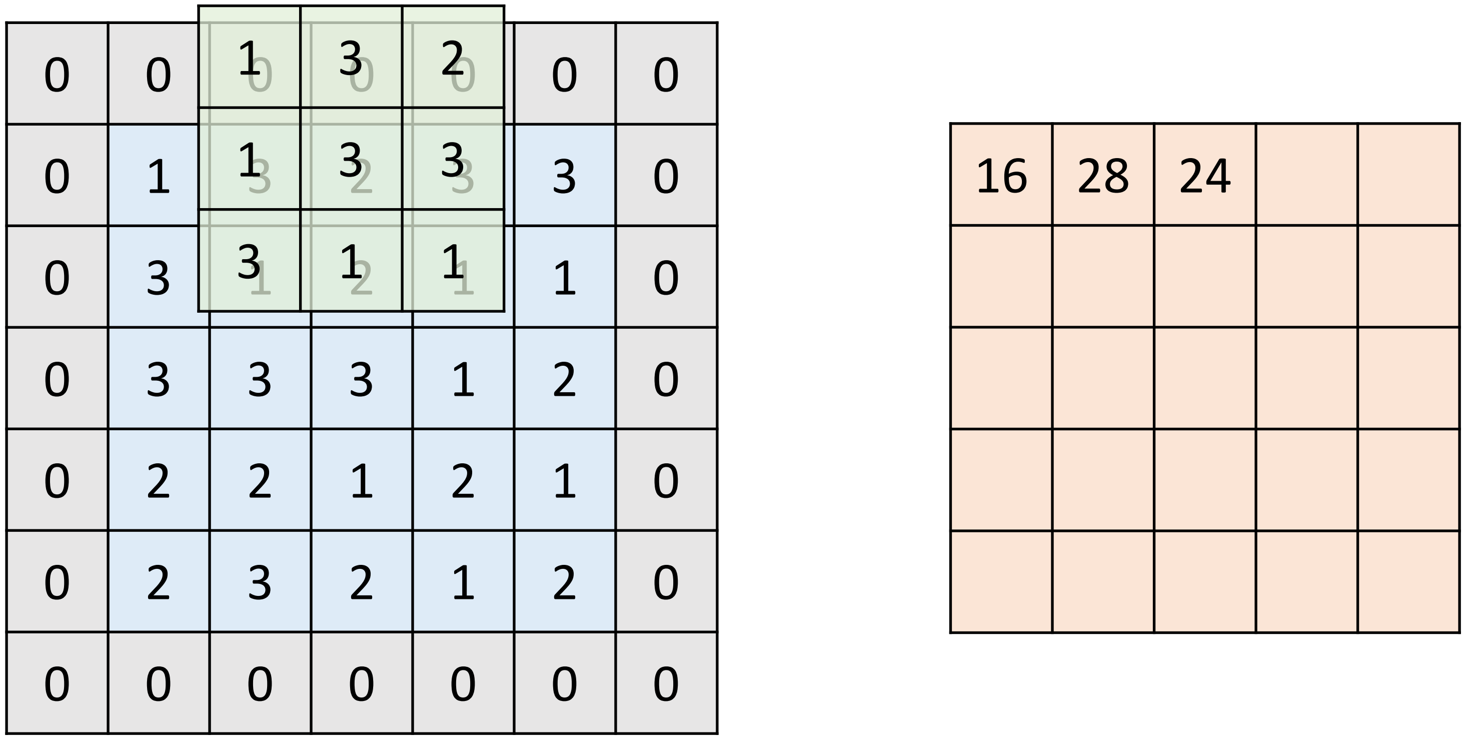
2D Convolution

- Input_channel=1, output_channel=1, padding=1, stride=1



2D Convolution

- Input_channel=1, output_channel=1, padding=1, stride=1



2D Convolution

- Input_channel=1, output_channel=1, padding=1, stride=1

0	0	0	0	0	0	0
0	1	3	2	3	3	0
0	3	1	2	1	1	0
0	3	3	3	1	2	0
0	2	2	1	2	1	0
0	2	3	2	1	2	0
0	0	0	0	0	0	0

Input

*

1	3	2
1	3	3
3	1	1

filter

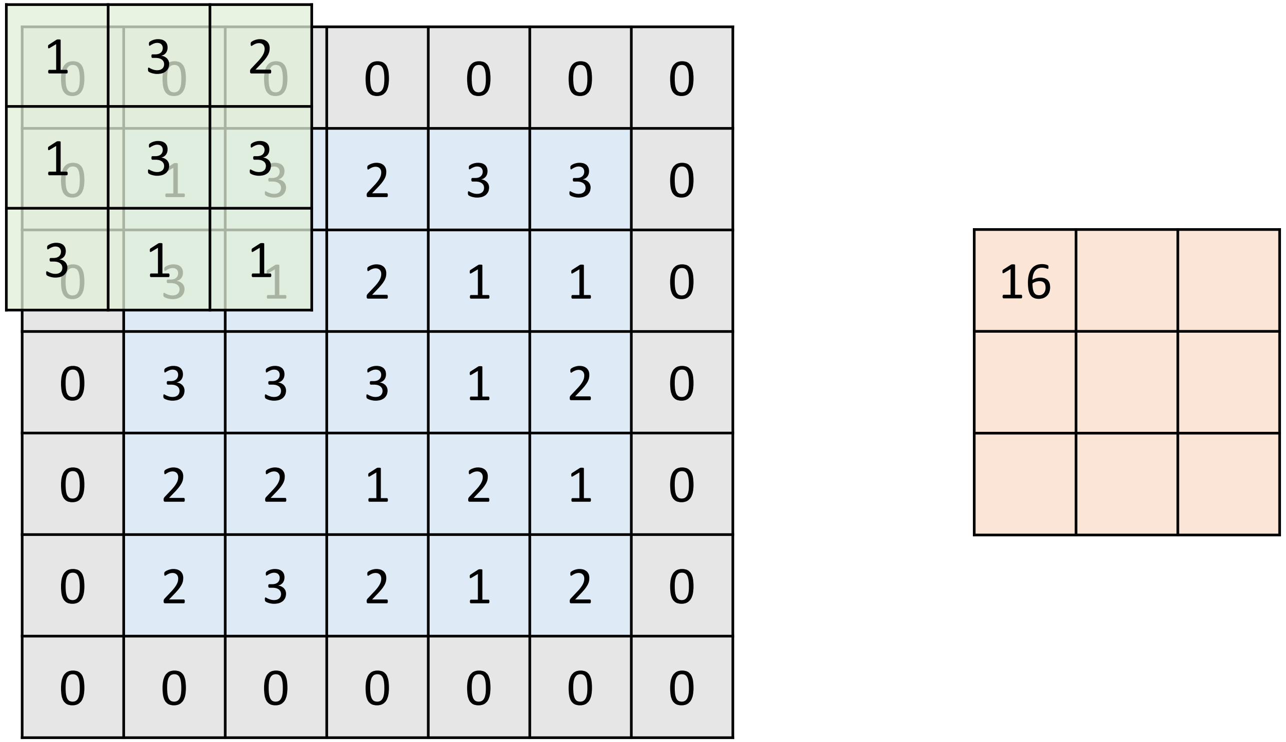
=

16	28	24	28	16
27	41	38	37	21
33	40	33	25	18
32	40	37	29	17
25	27	21	20	12

Output

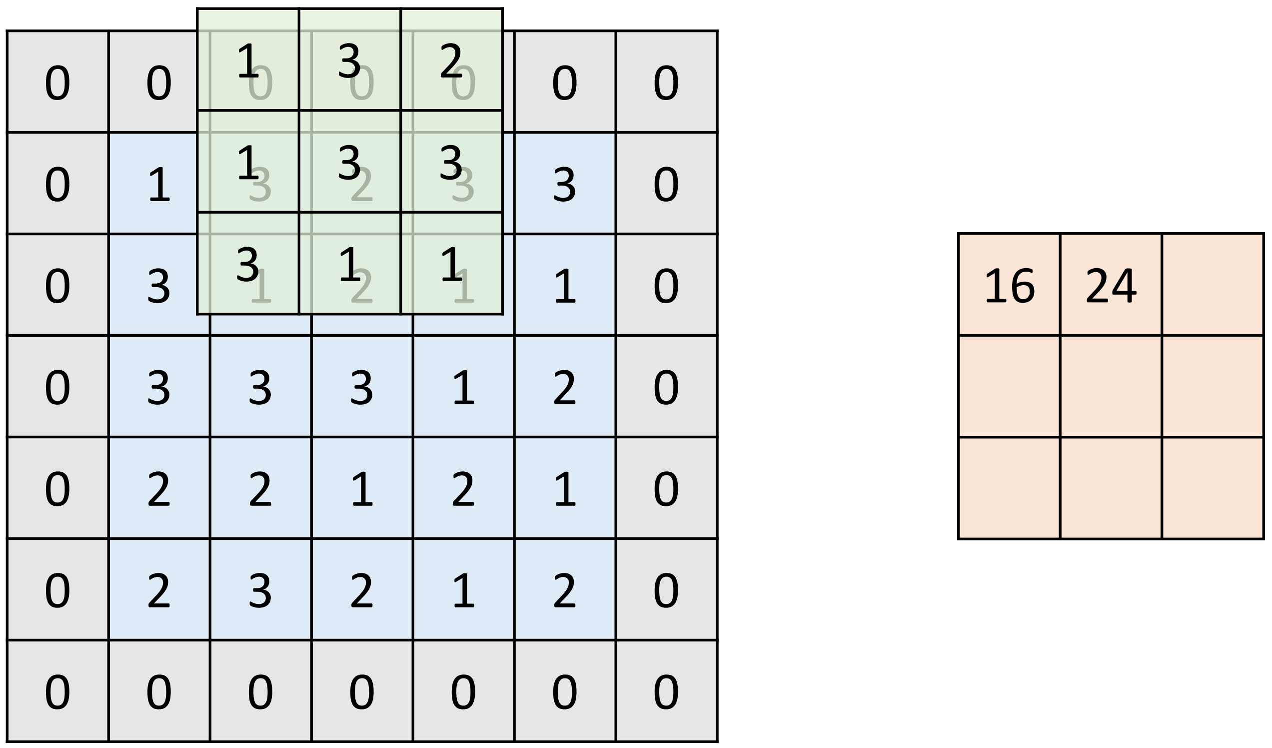
2D Convolution

- Input_channel=1, output_channel=1, padding=1, stride=2



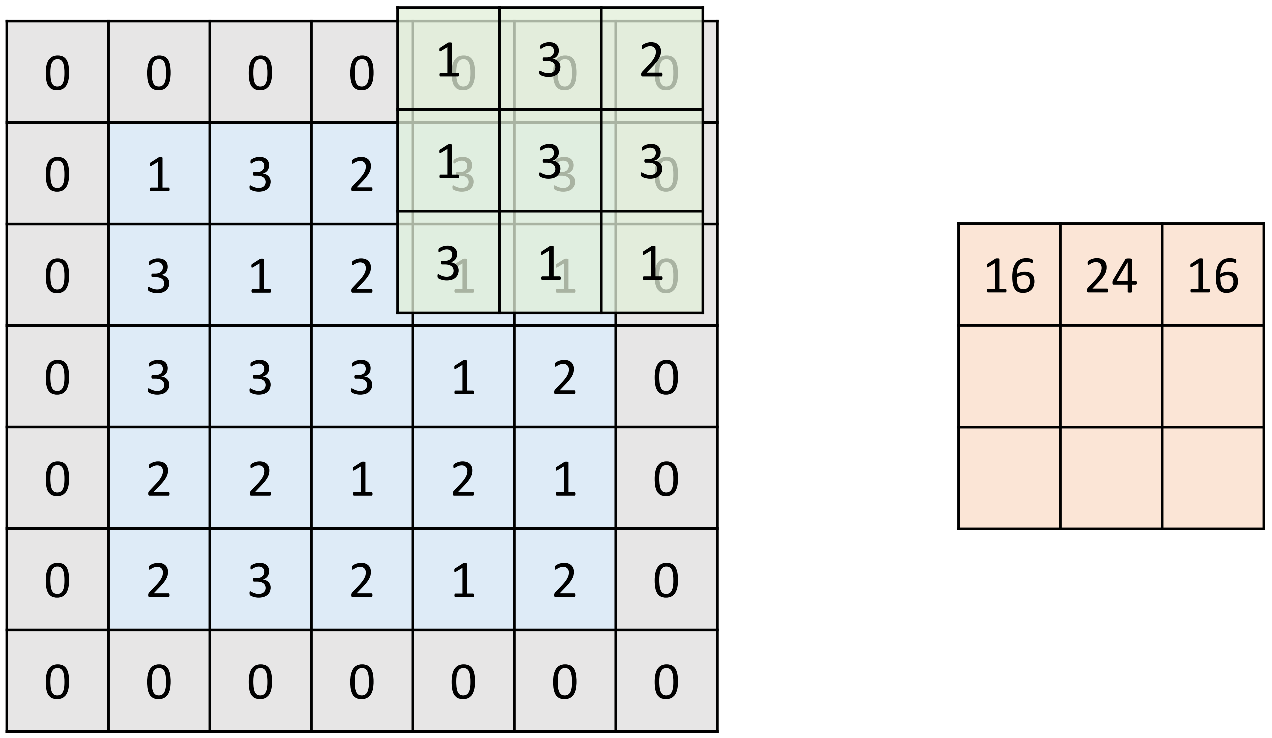
2D Convolution

- Input_channel=1, output_channel=1, padding=1, stride=2



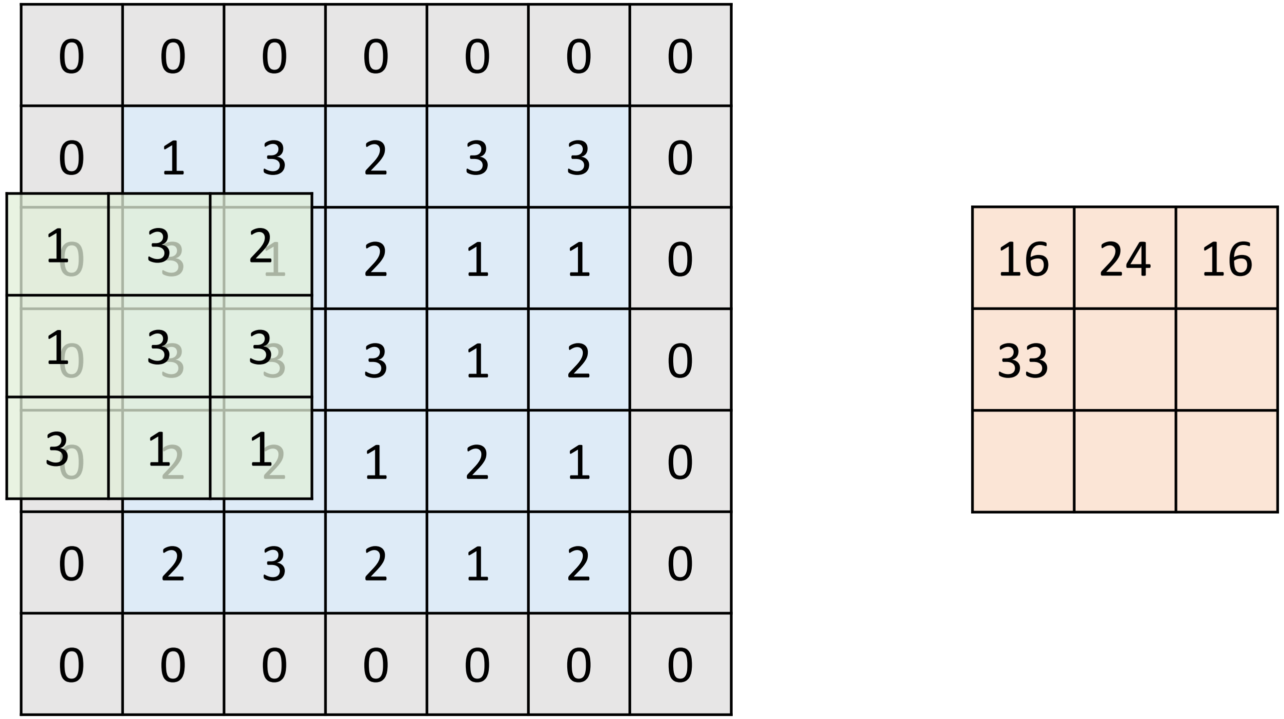
2D Convolution

- Input_channel=1, output_channel=1, padding=1, stride=2



2D Convolution

- Input_channel=1, output_channel=1, padding=1, stride=2



2D Convolution

- Input_channel=1, output_channel=1, padding=1, stride=2

0	0	0	0	0	0	0
0	1	3	2	3	3	0
0	3	1	2	1	1	0
0	3	3	3	1	2	0
0	2	2	1	2	1	0
0	2	3	2	1	2	0
0	0	0	0	0	0	0

Input

*

1	3	2
1	3	3
3	1	1

filter

=

16	24	16
33	33	18
25	21	12

Output

2D Convolution

- Input_channel=1, output_channel=1, padding=2, stride=1

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	1	3	2	3	3	0	0
0	0	3	1	2	1	1	0	0
0	0	3	3	3	1	2	0	0
0	0	2	2	1	2	1	0	0
0	0	2	3	2	1	2	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

*

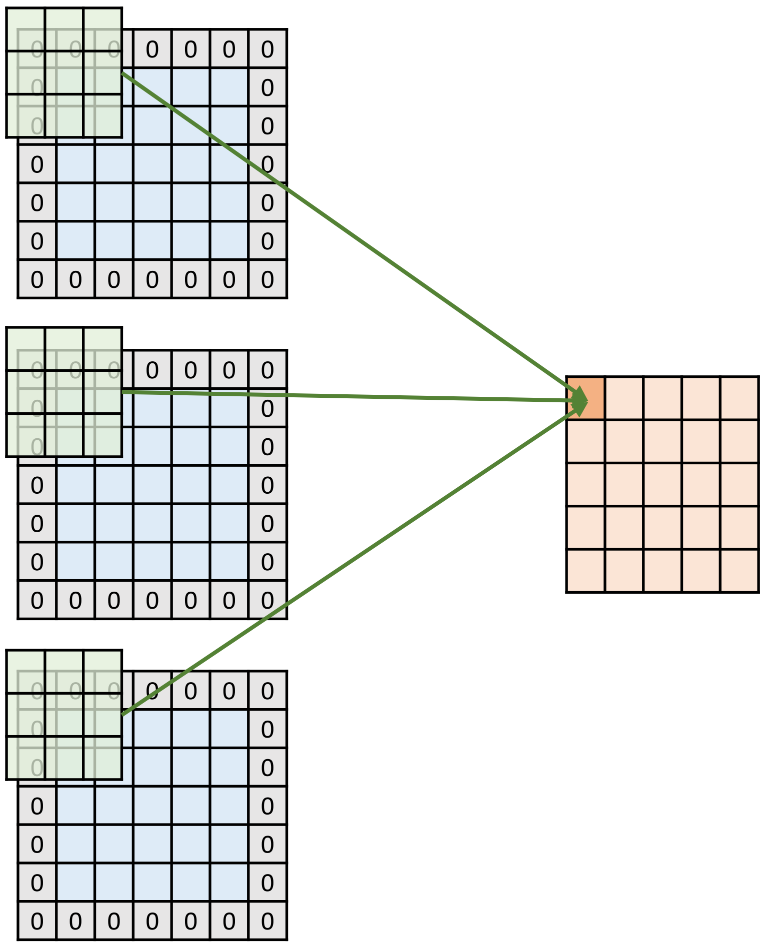
1	3	2
1	3	3
3	1	1

=

1	4	8	14	12	12	9
6	16	28	24	28	16	6
14	27	41	38	37	21	10
17	33	40	33	25	18	6
14	32	40	37	29	17	9
10	25	27	21	20	12	3
4	12	15	11	9	7	2

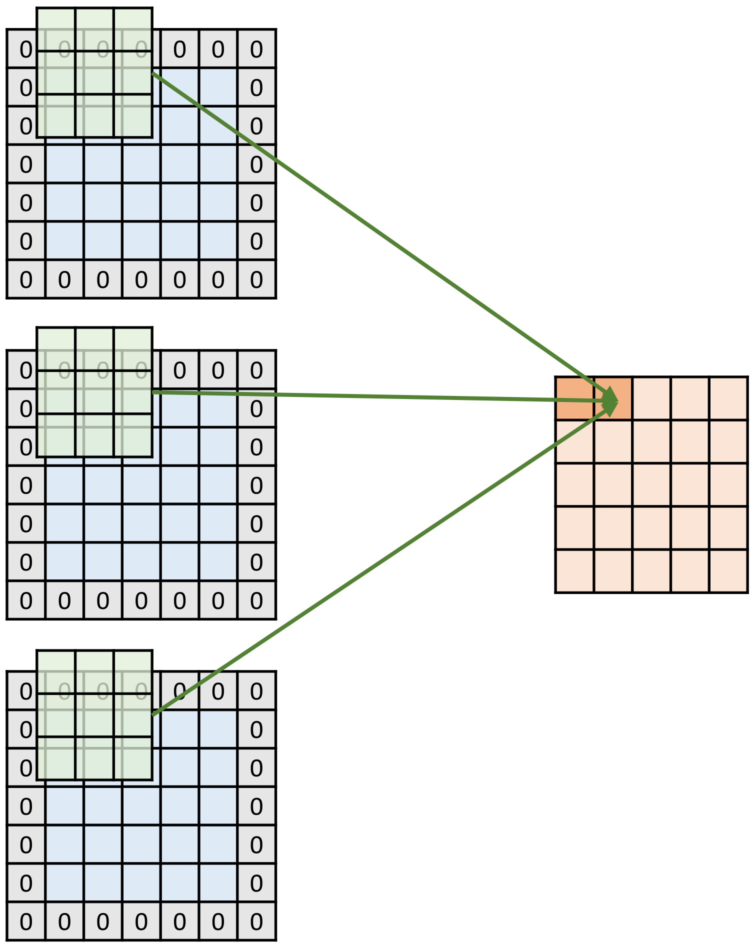
2D Convolution

- Input_channel=3, output_channel=1, padding=1, stride=1



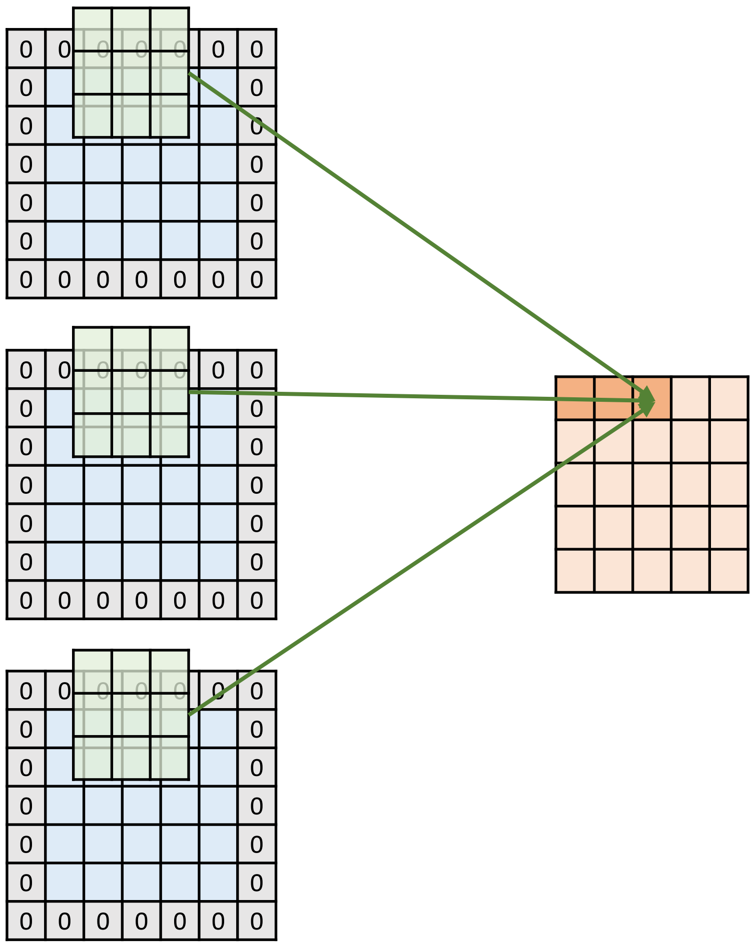
2D Convolution

- Input_channel=3, output_channel=1, padding=1, stride=1



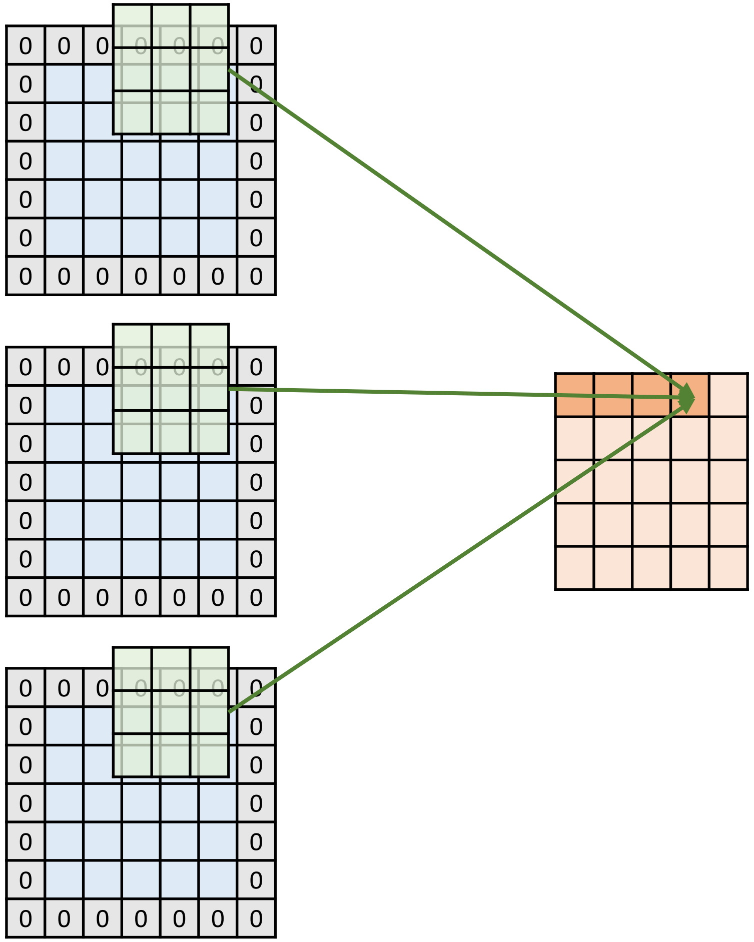
2D Convolution

- Input_channel=3, output_channel=1, padding=1, stride=1



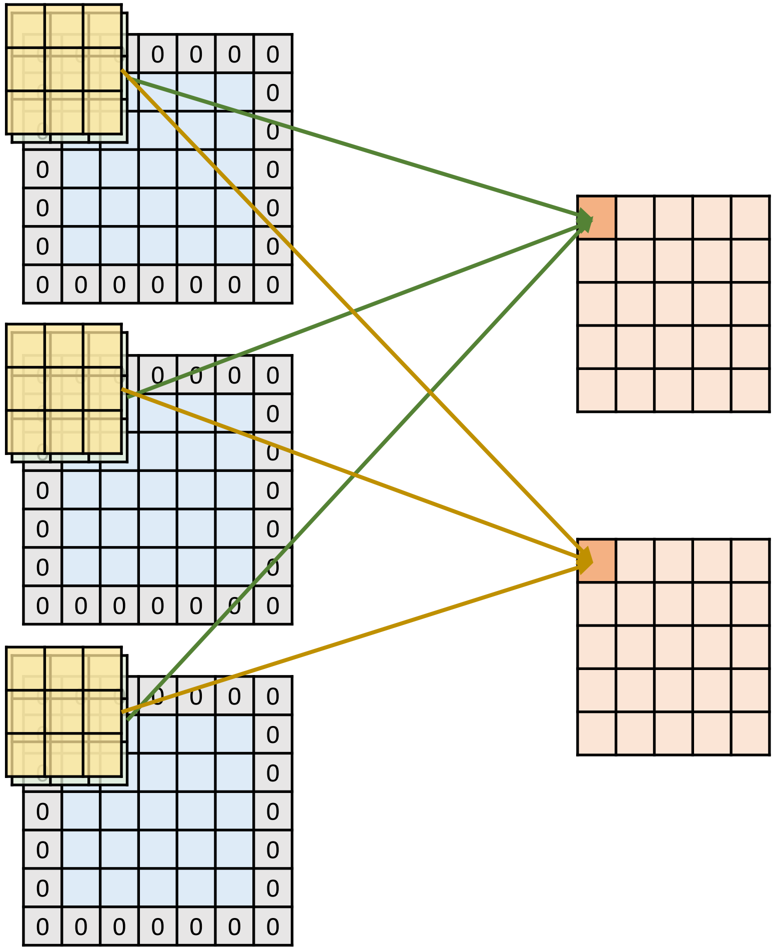
2D Convolution

- Input_channel=3, output_channel=1, padding=1, stride=1



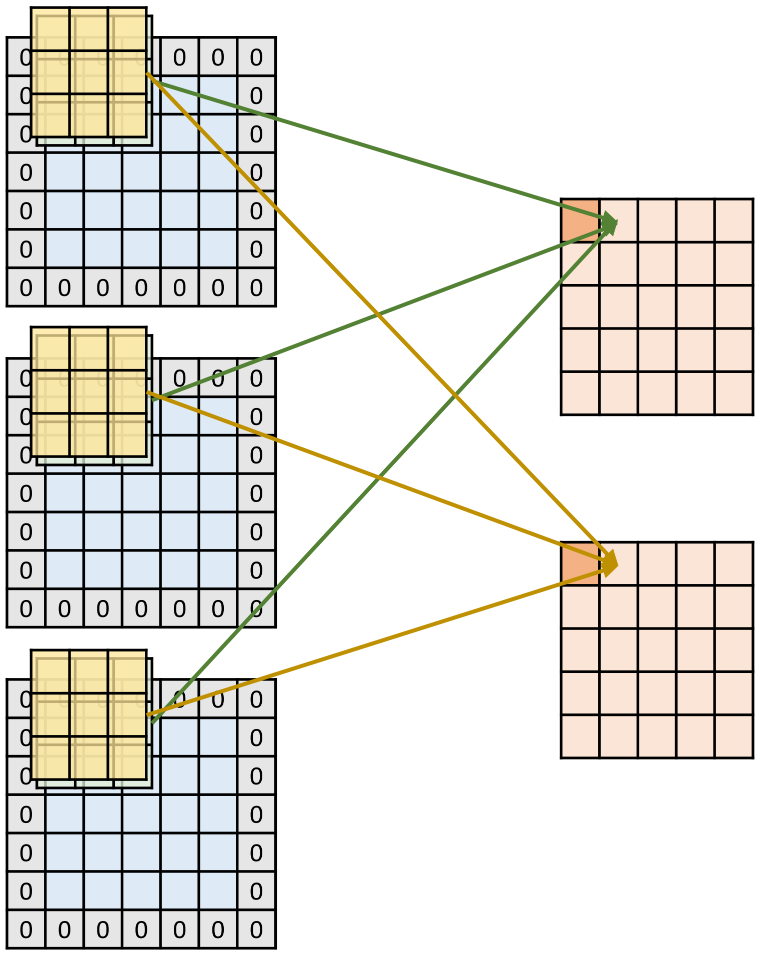
2D Convolution

- Input_channel=3, output_channel=2, padding=1, stride=1



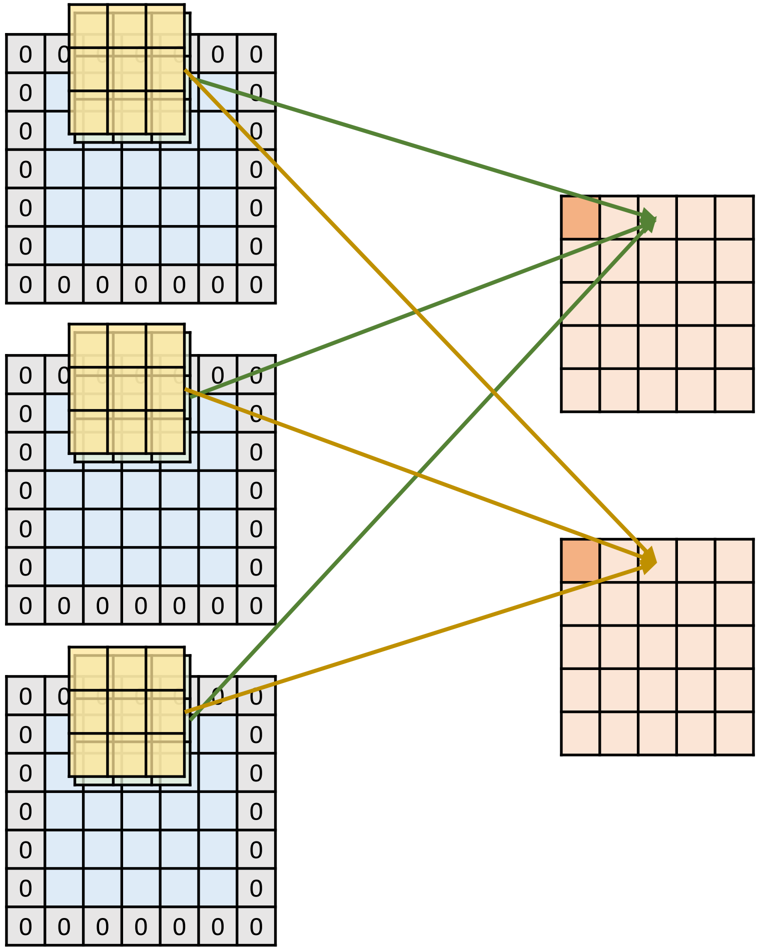
2D Convolution

- Input_channel=3, output_channel=2, padding=1, stride=1



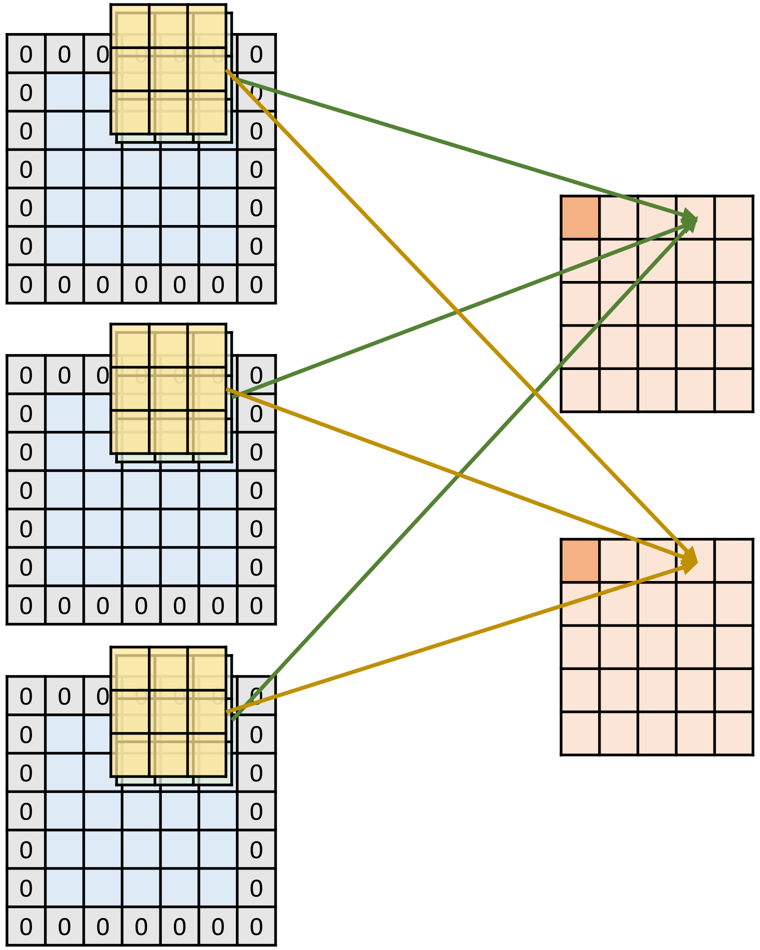
2D Convolution

- Input_channel=3, output_channel=2, padding=1, stride=1



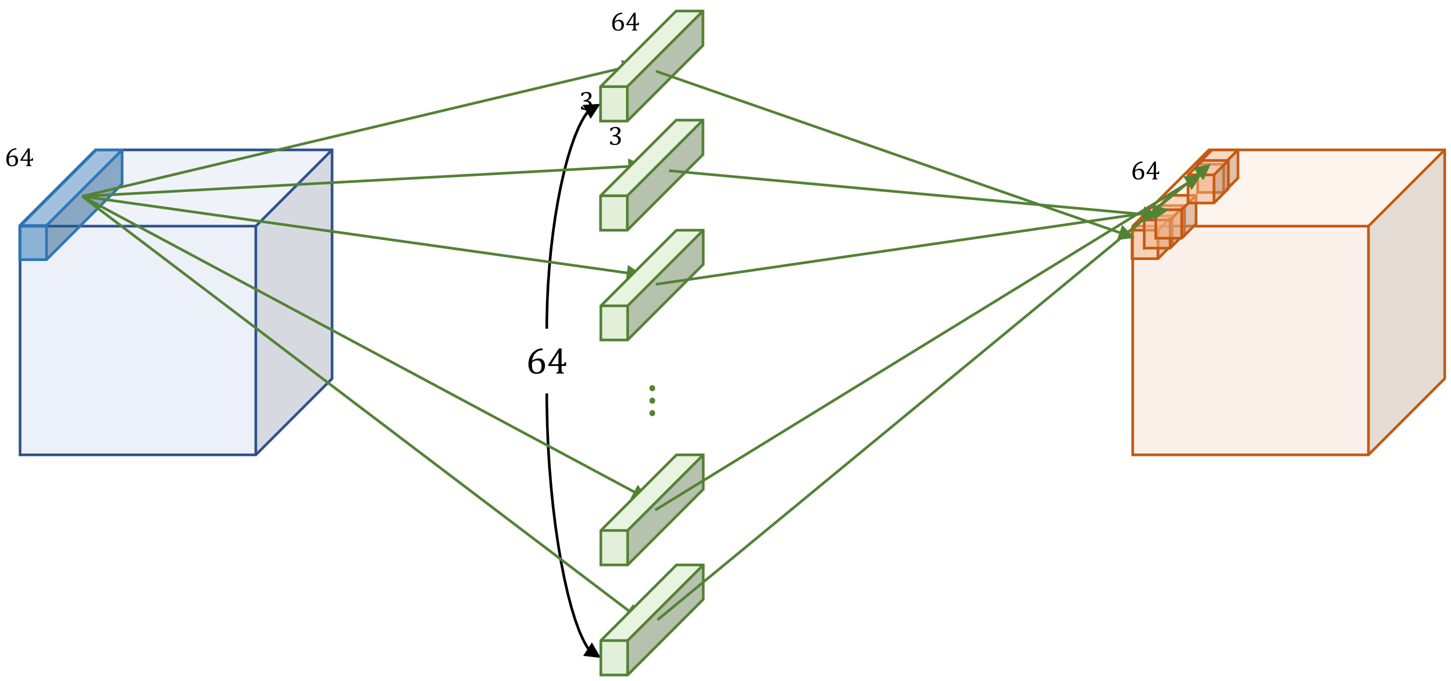
2D Convolution

- Input_channel=3, output_channel=2, padding=1, stride=1



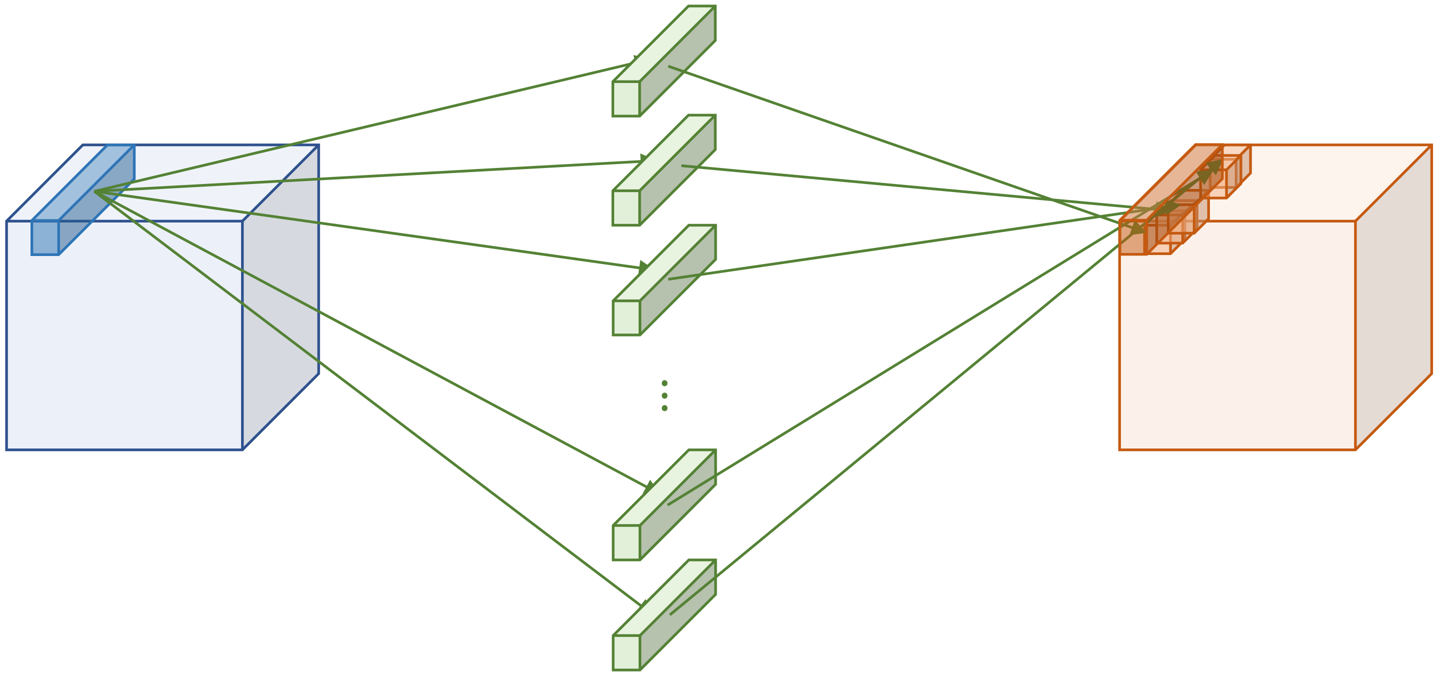
2D Convolution

- Input_channel=64, output_channel=64, kernel_size=3, padding=1, stride=1



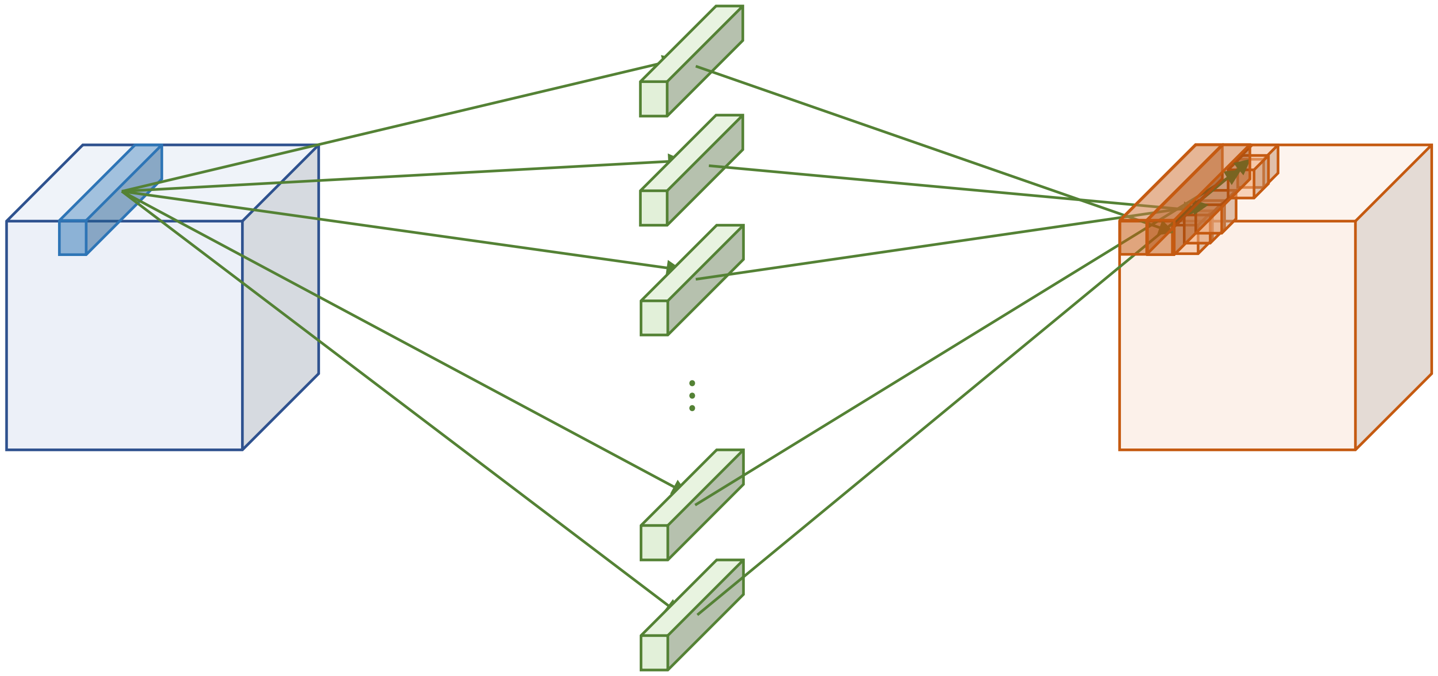
2D Convolution

- Input_channel=64, output_channel=64, kernel_size=3, padding=1, stride=1



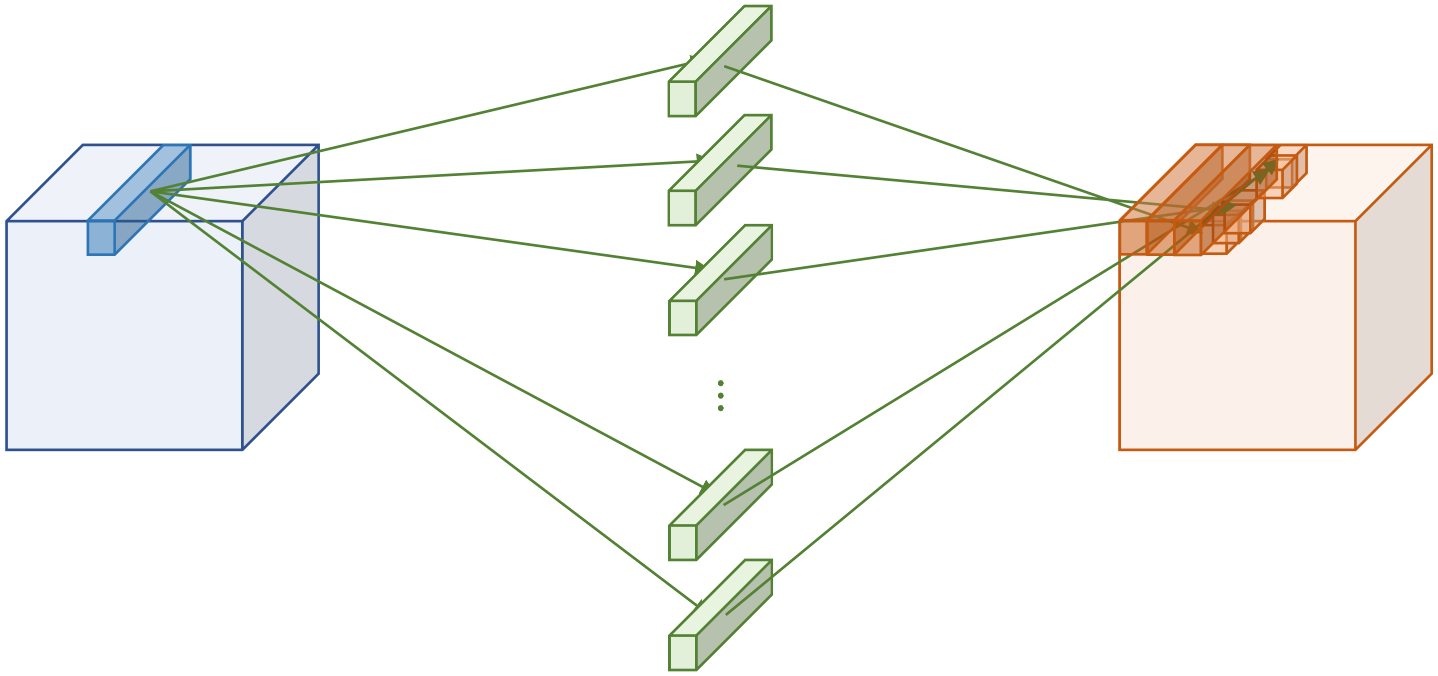
2D Convolution

- Input_channel=64, output_channel=64, kernel_size=3, padding=1, stride=1



2D Convolution

- Input_channel=64, output_channel=64, kernel_size=3, padding=1, stride=1



Convolutions in PyTorch

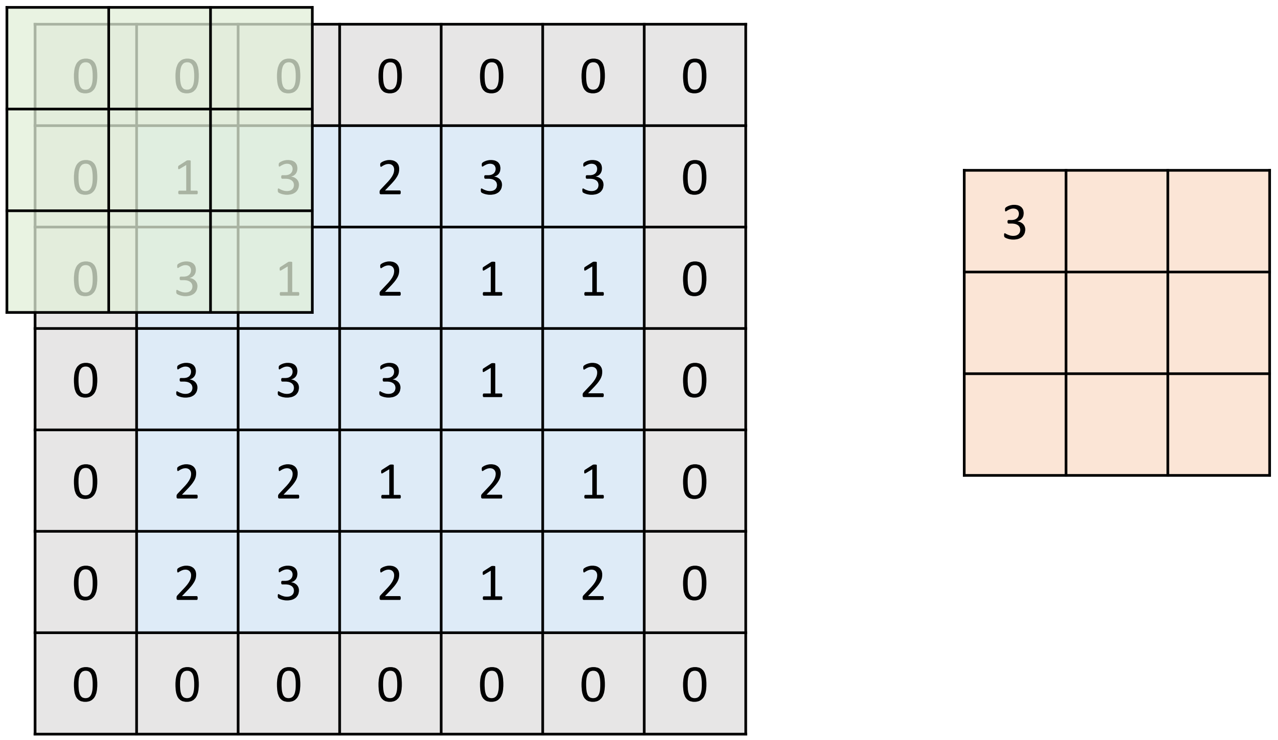
CLASS `torch.nn.Conv1d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)` [SOURCE]

CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)` [SOURCE]

CLASS `torch.nn.Conv3d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)` [SOURCE]

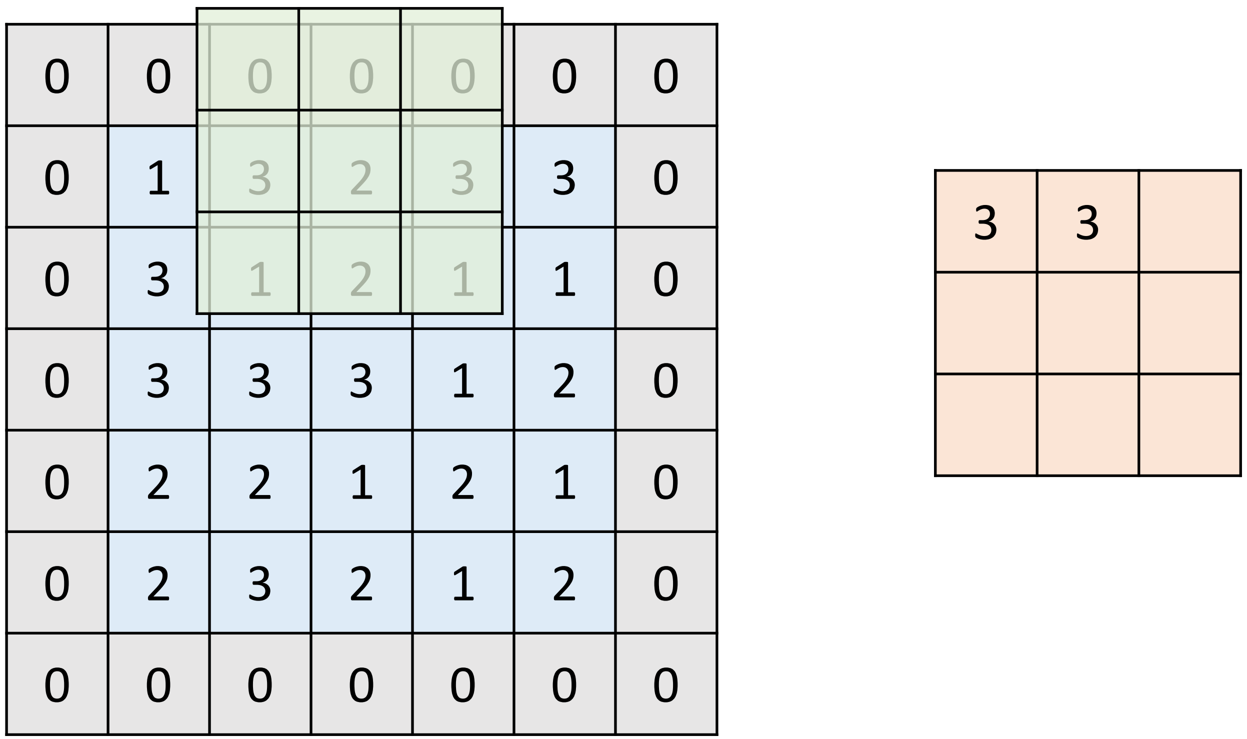
Max Pooling

- Pooling a maximum value given the window
- Used to reduce the size of feature maps
- Example) stride=2, padding=1



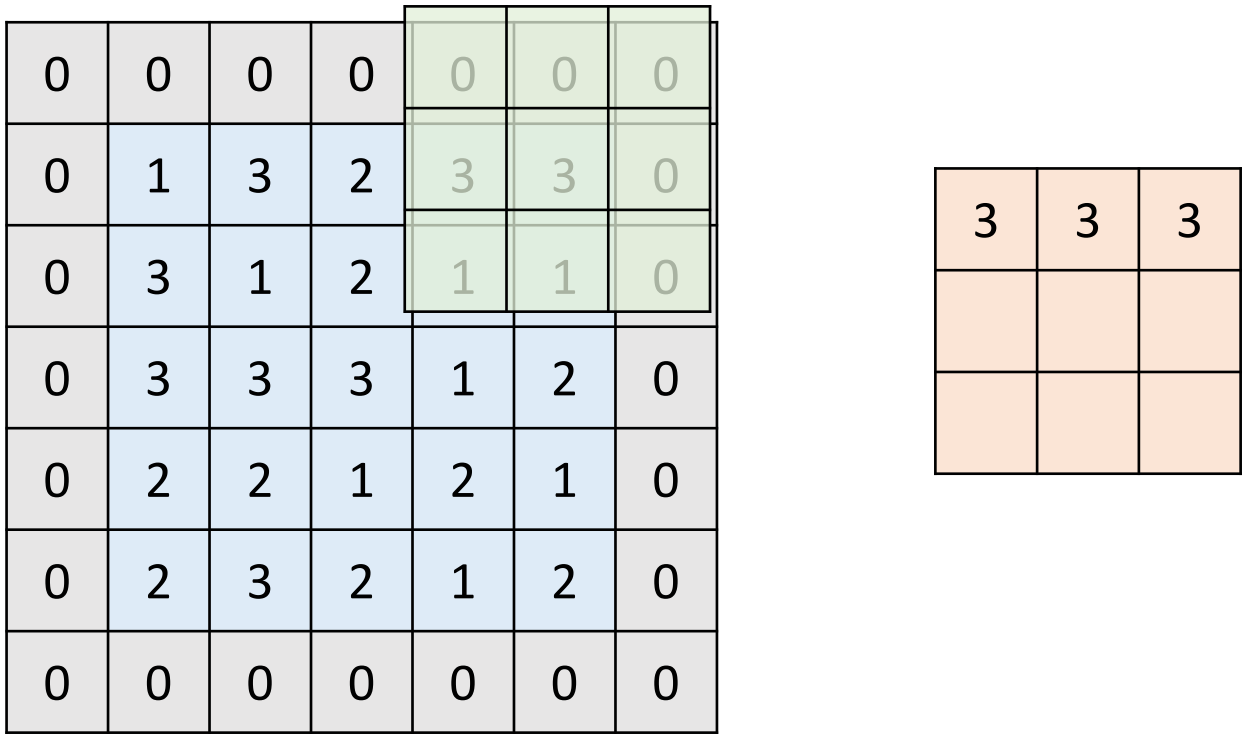
Max Pooling

- Pooling a maximum value given the window
- Used to reduce the size of feature maps
- Example) stride=2, padding=1



Max Pooling

- Pooling a maximum value given the window
- Used to reduce the size of feature maps
- Example) stride=2, padding=1



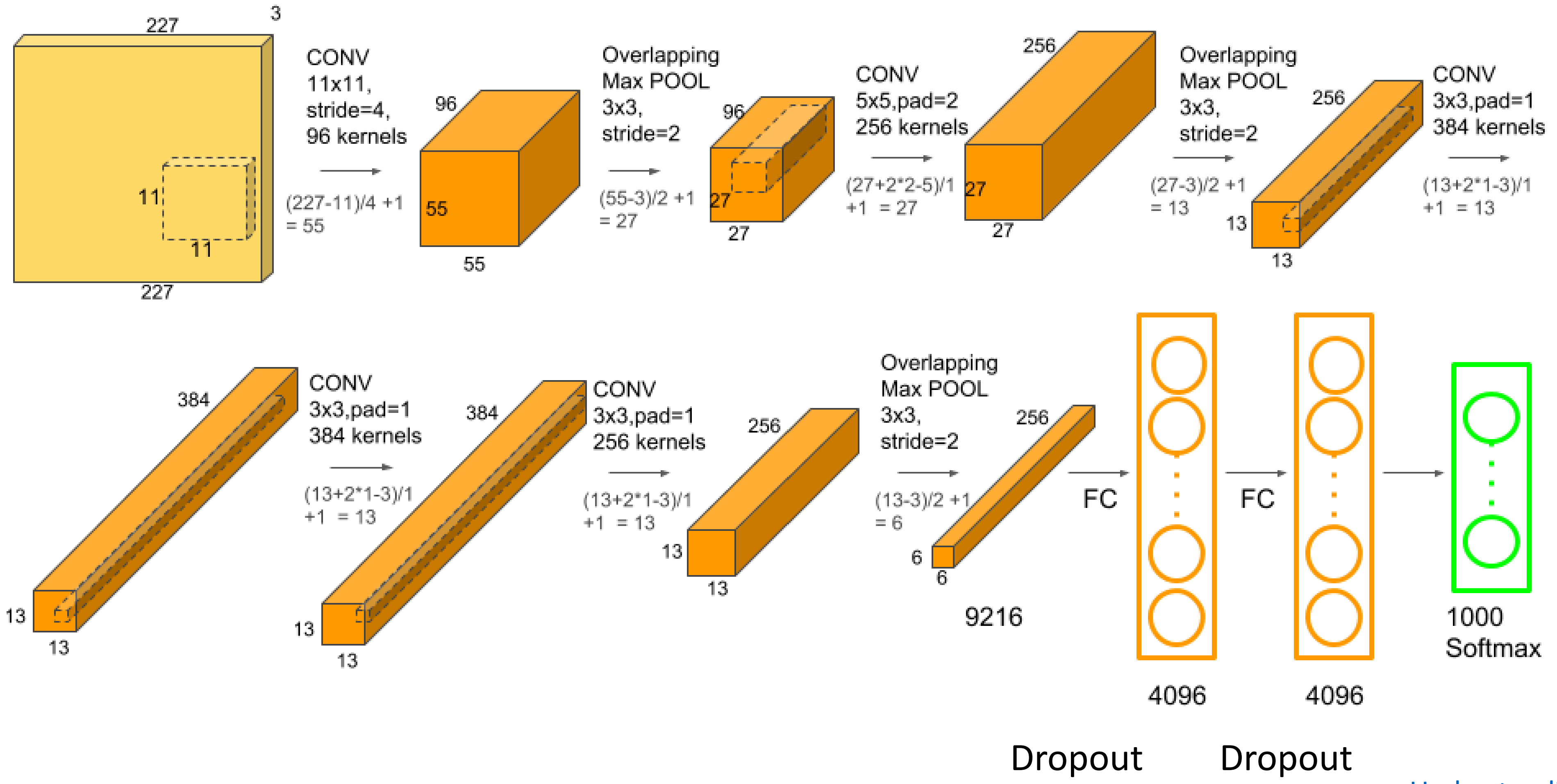
Max Pooling in PyTorch

CLASS `torch.nn.MaxPool1d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False)` [SOURCE]

CLASS `torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False)` [SOURCE]

CLASS `torch.nn.MaxPool3d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False)` [SOURCE]

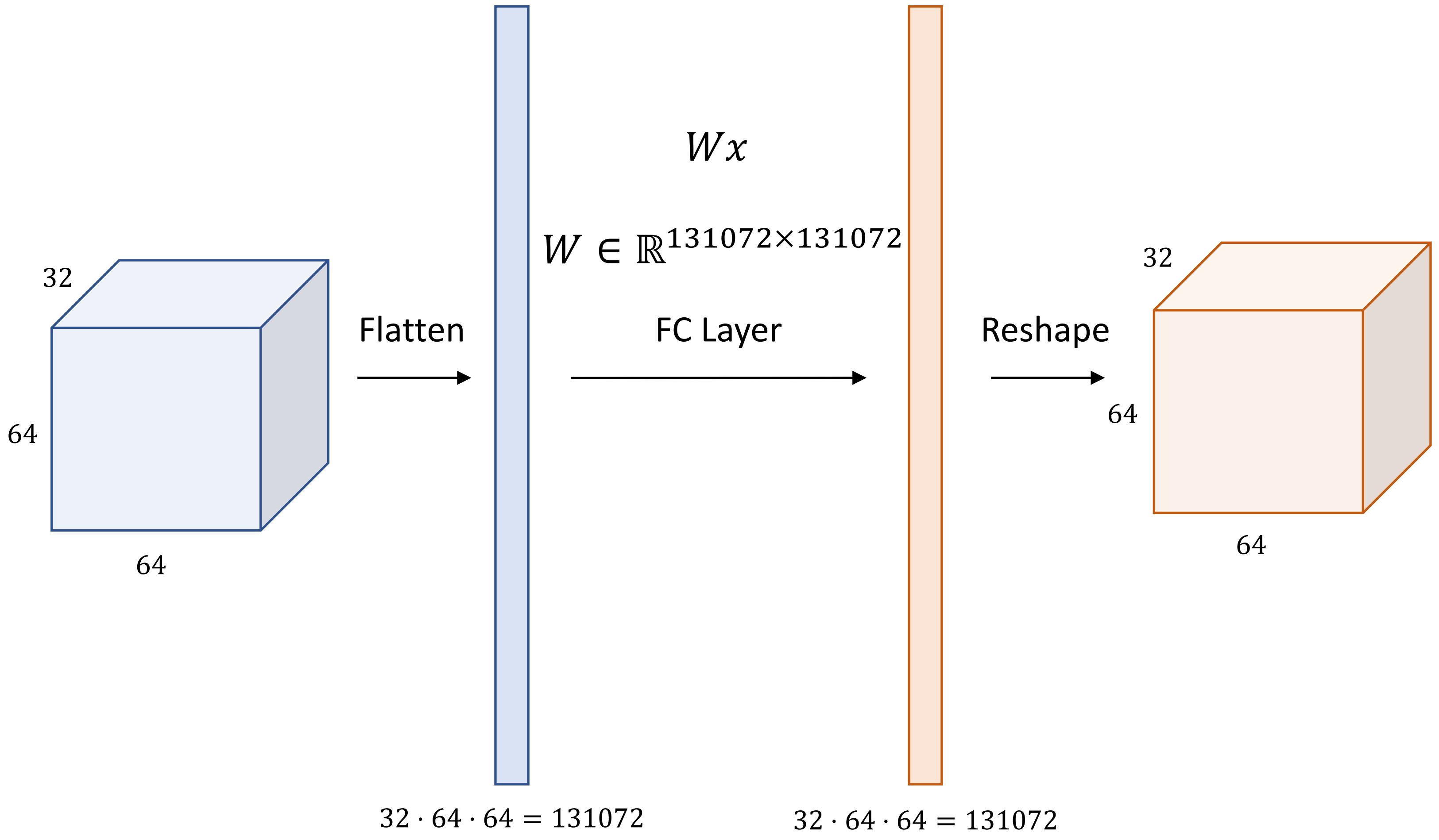
AlexNet



[Understanding AlexNet | LearnOpenCV #](#)

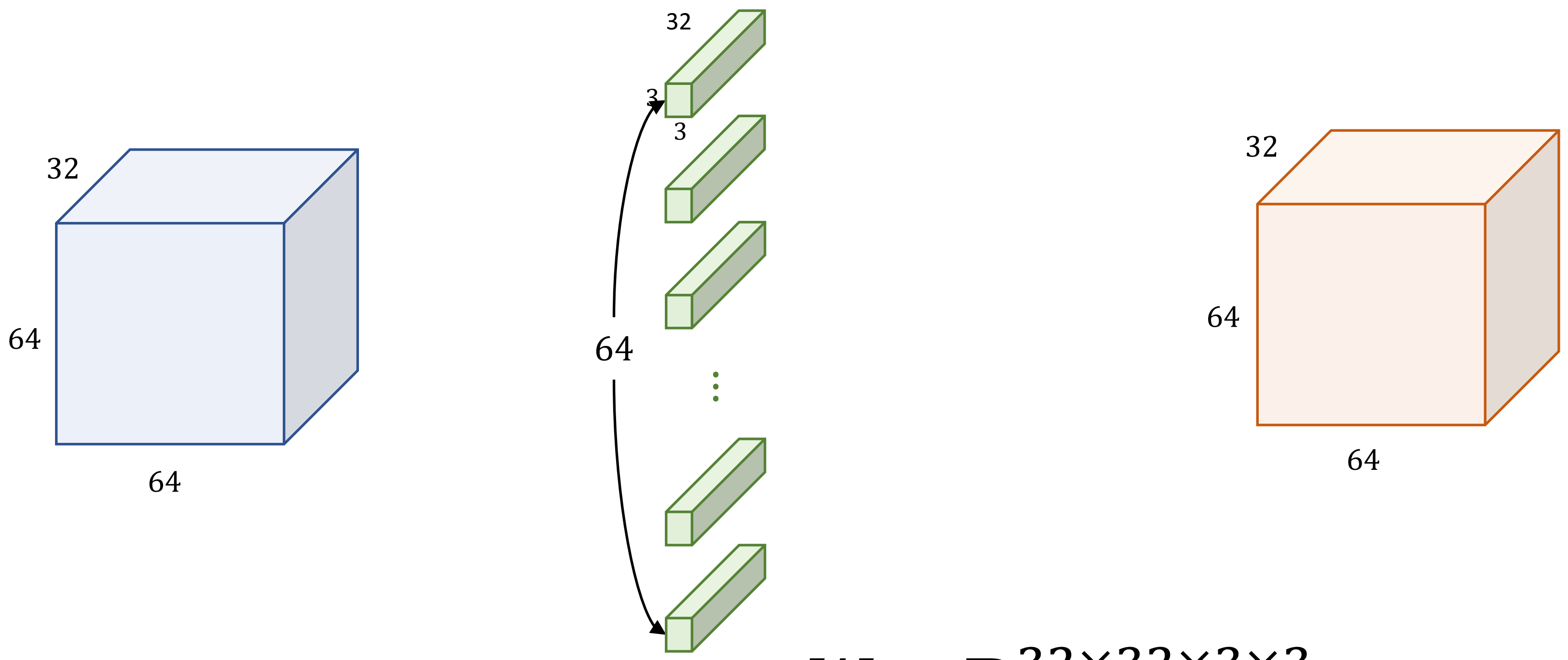
Fully Connected Layer vs Convolutional Layer

- Translation equivariance and parameter sharing



Fully Connected Layer vs Convolutional Layer

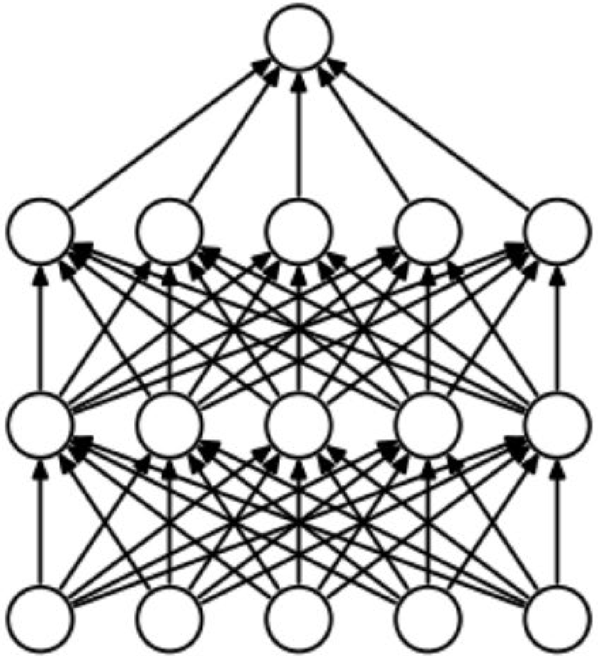
- Translation equivariance and parameter sharing



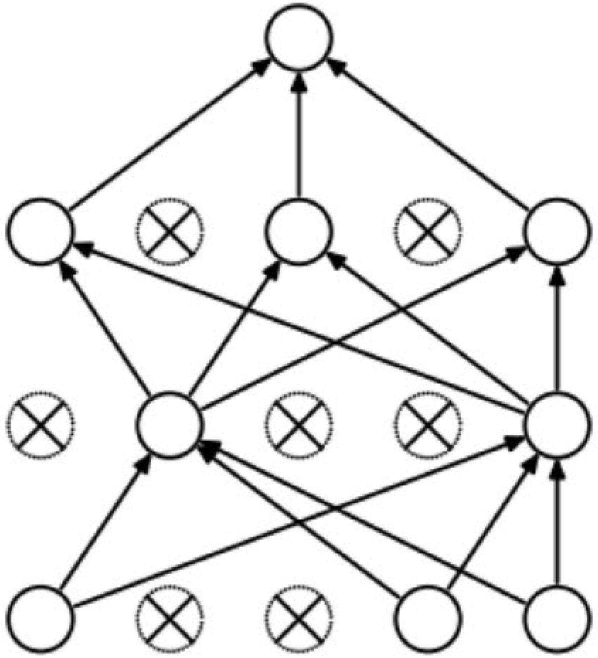
$$W \in \mathbb{R}^{32\times32\times3\times3}$$

Dropout

- Turning off neurons w/ given probability (e.g. 0.5)
- Every iterations, new network architectures emerge



(a) Standard Neural Net

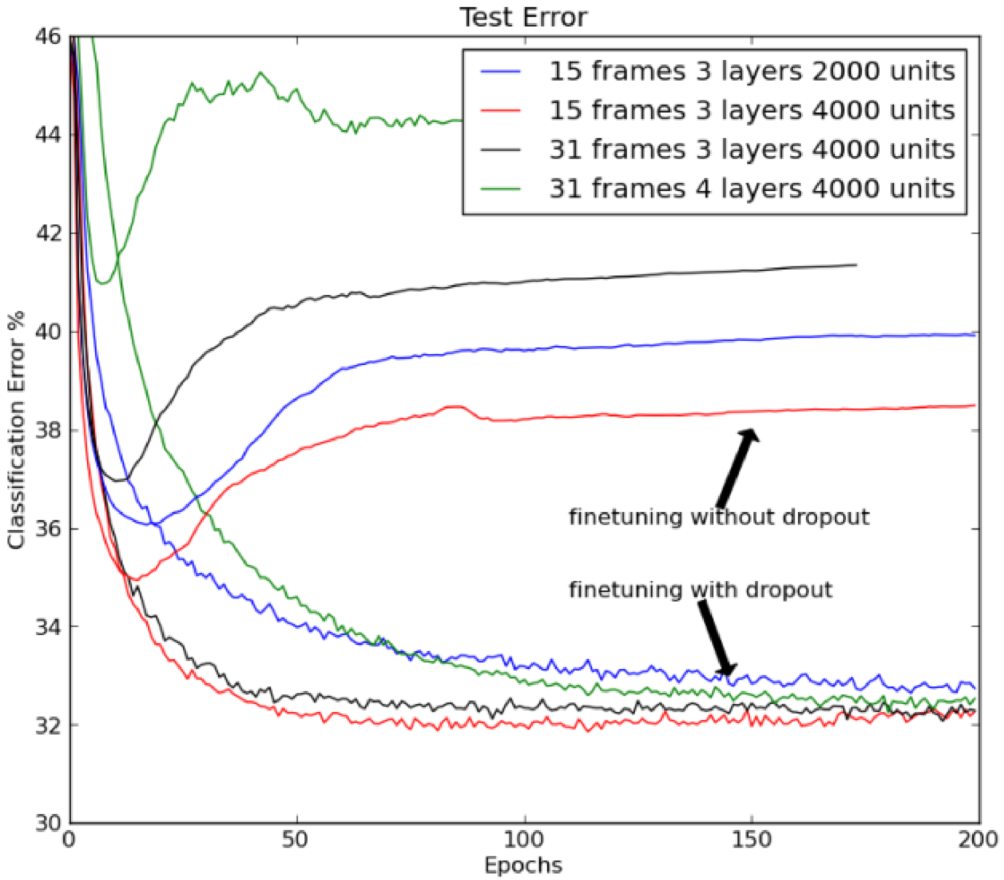


(b) After applying dropout.

Improving neural networks by preventing co-adaptation of feature detectors, hinton et al, arXiv 2012

Dropout

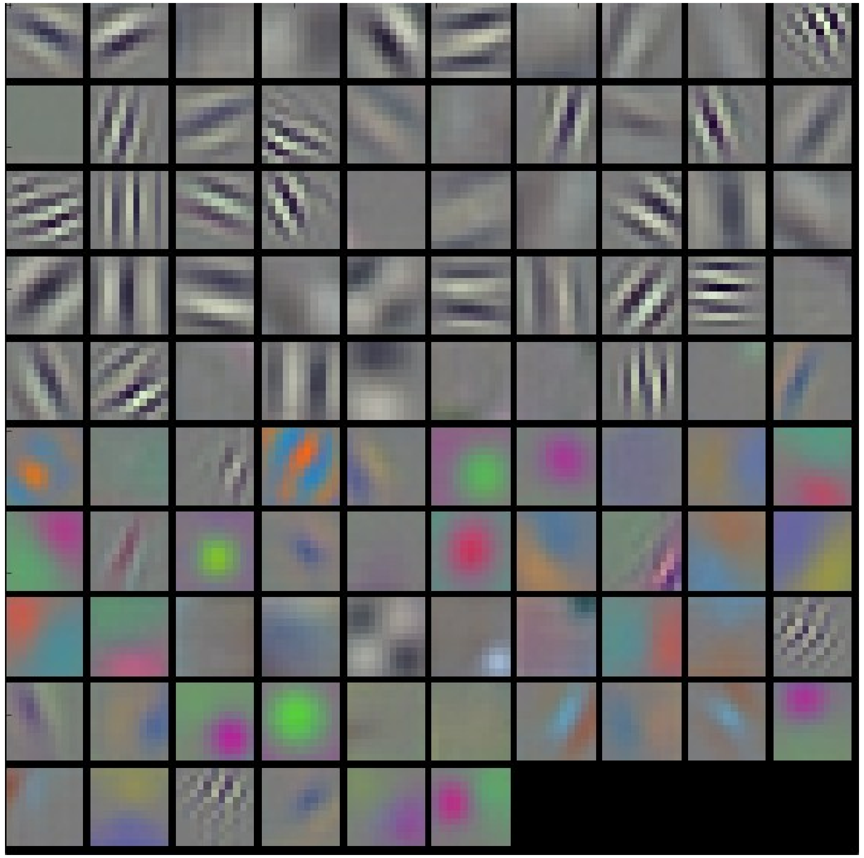
- A simple way to train deep neural networks for improving generalization performance
- Avoiding co-adaptations: a hidden unit cannot rely on other hidden units being present
- Model averaging



Improving neural networks by preventing co-adaptation of feature detectors, hinton et al, arXiv 2012

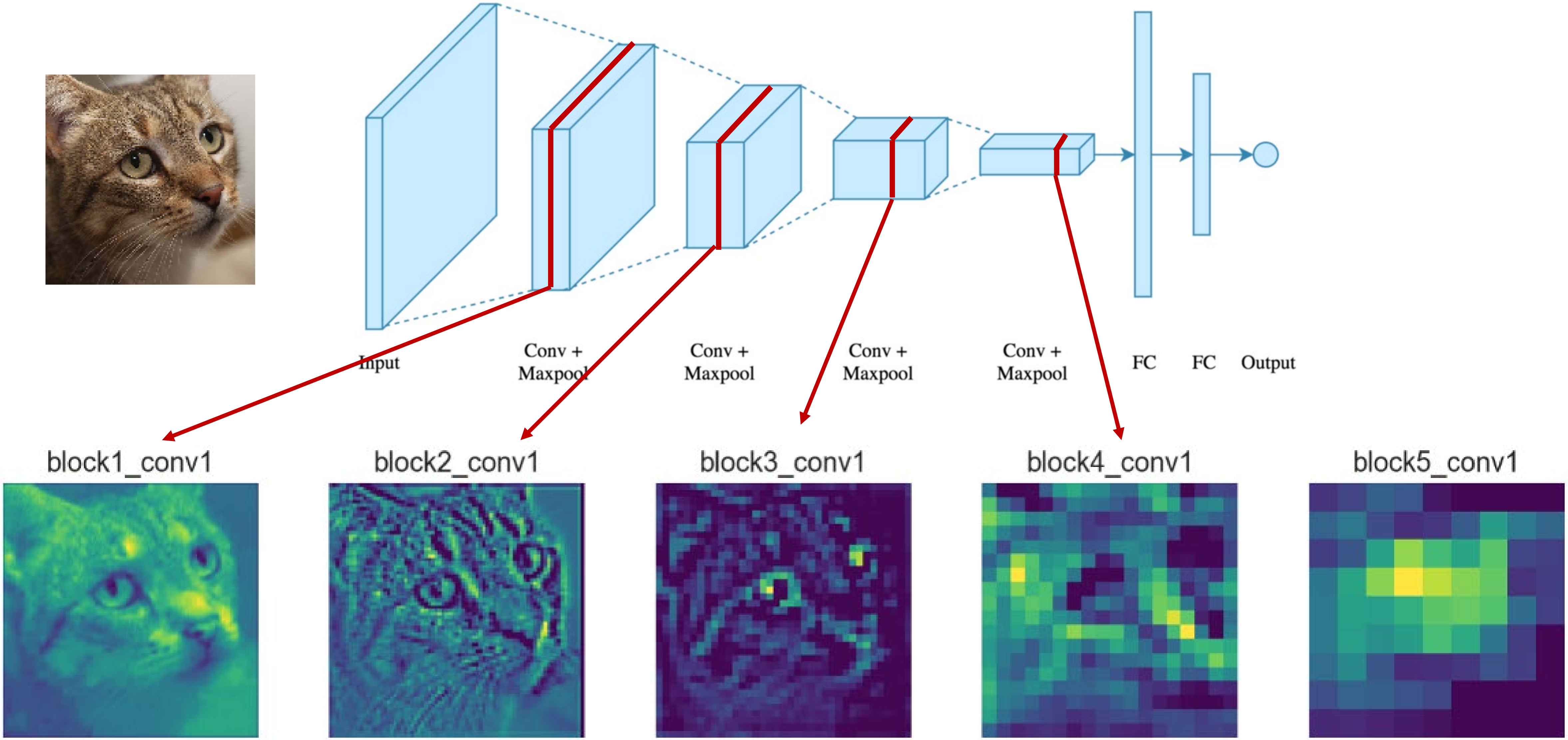
Visualization of Learned Filter

- First layer conv filters



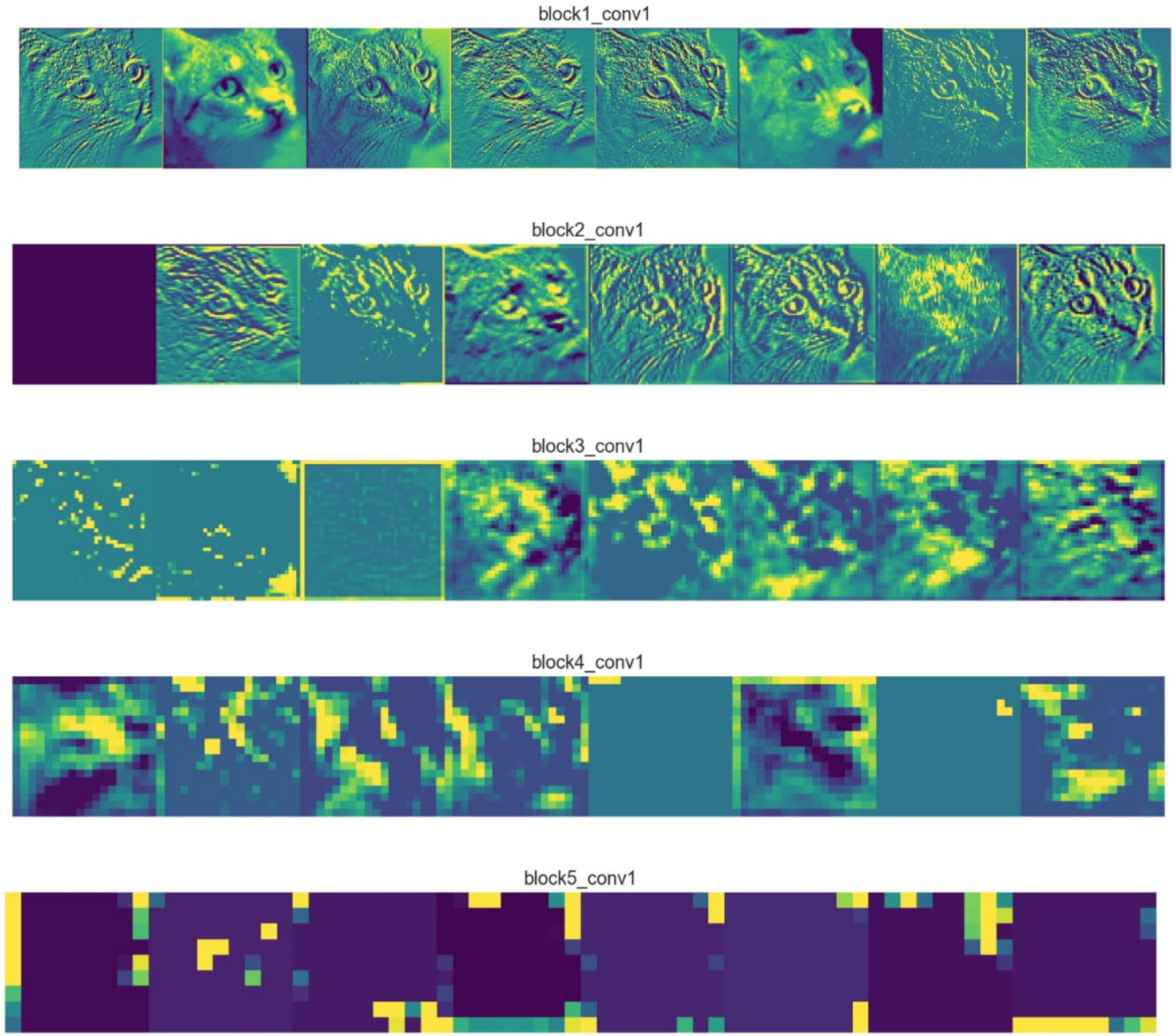
CS231n Convolutional Neural Networks for Visual Recognition

Visualization of Learned Feature Maps



[Applied Deep Learning - Part 4: Convolutional Neural Networks | by Arden Dertat | Towards Data Science]

Visualization of Learned Feature Maps



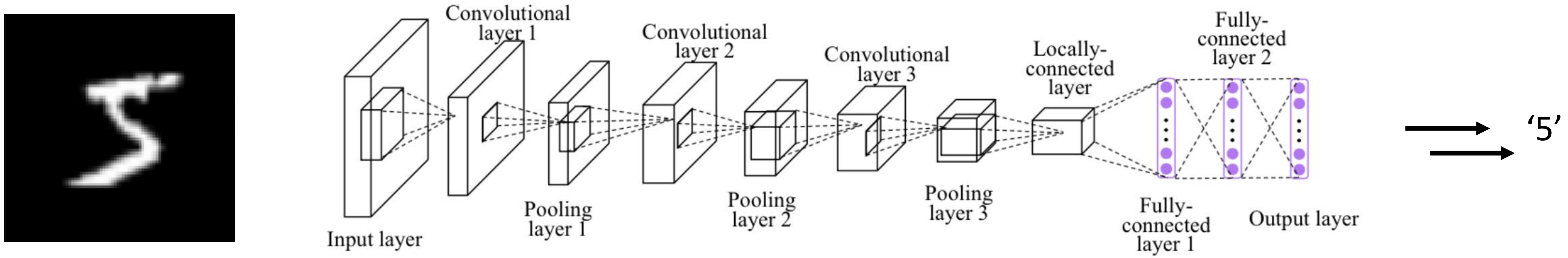
[Applied Deep Learning - Part 4: Convolutional Neural Networks | by Arden Dertat | Towards Data Science]

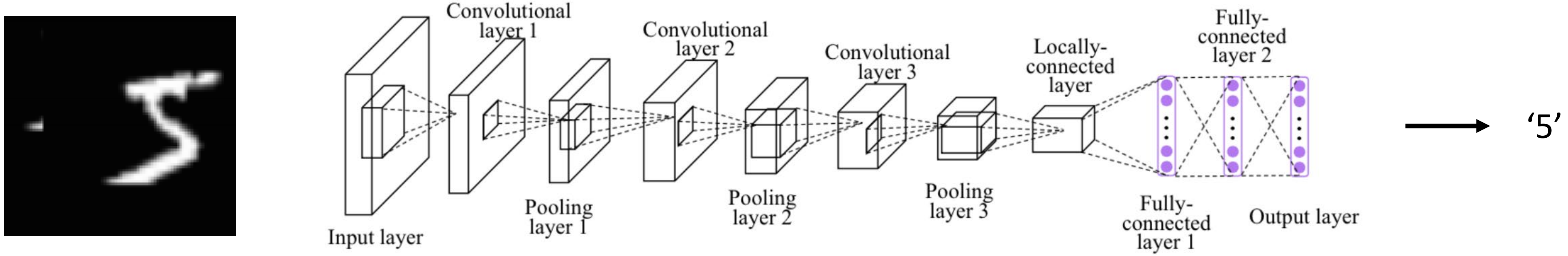
Translation Equivariance vs Invariance

- Convolutional layers are translation equivariant
  - If you shift 1 pixel of input image, then the outputs will be 1 pixel shifted
- Pooling layers are translation invariant upto small shifting
  - If you shift 1 pixel of input image, then the outputs will be same

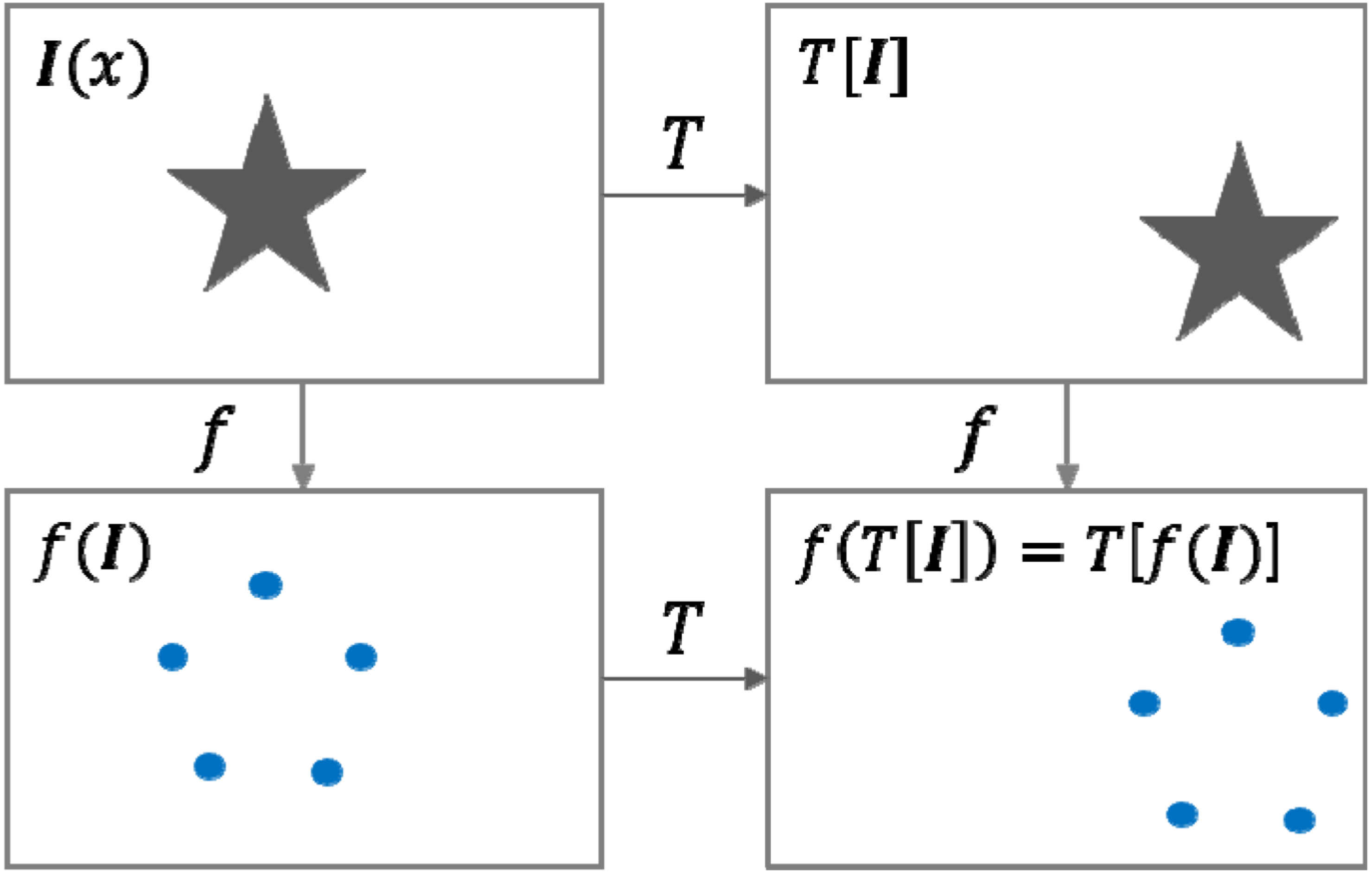
- We want network's prediction to be translation invariant
  - If you shift, you still want network to classify same as before

Translation Equivariance vs Invariance





Translation Equivariance vs Invariance



Cross correlation vs Convolution

- Cross-correlation: sliding a kernel across an image
- Convolution: sliding a flipped kernel across an image

- Most of deep learning libraries are actually doing 'cross-correlation'
  - But, it doesn't change the results since the same weight values would be learned in flipped manner

CNNExplainer

- CNN Explainer (poloclub.github.io)

Convolutions Behind the Scene

Sequential Implementation (Conv2D)

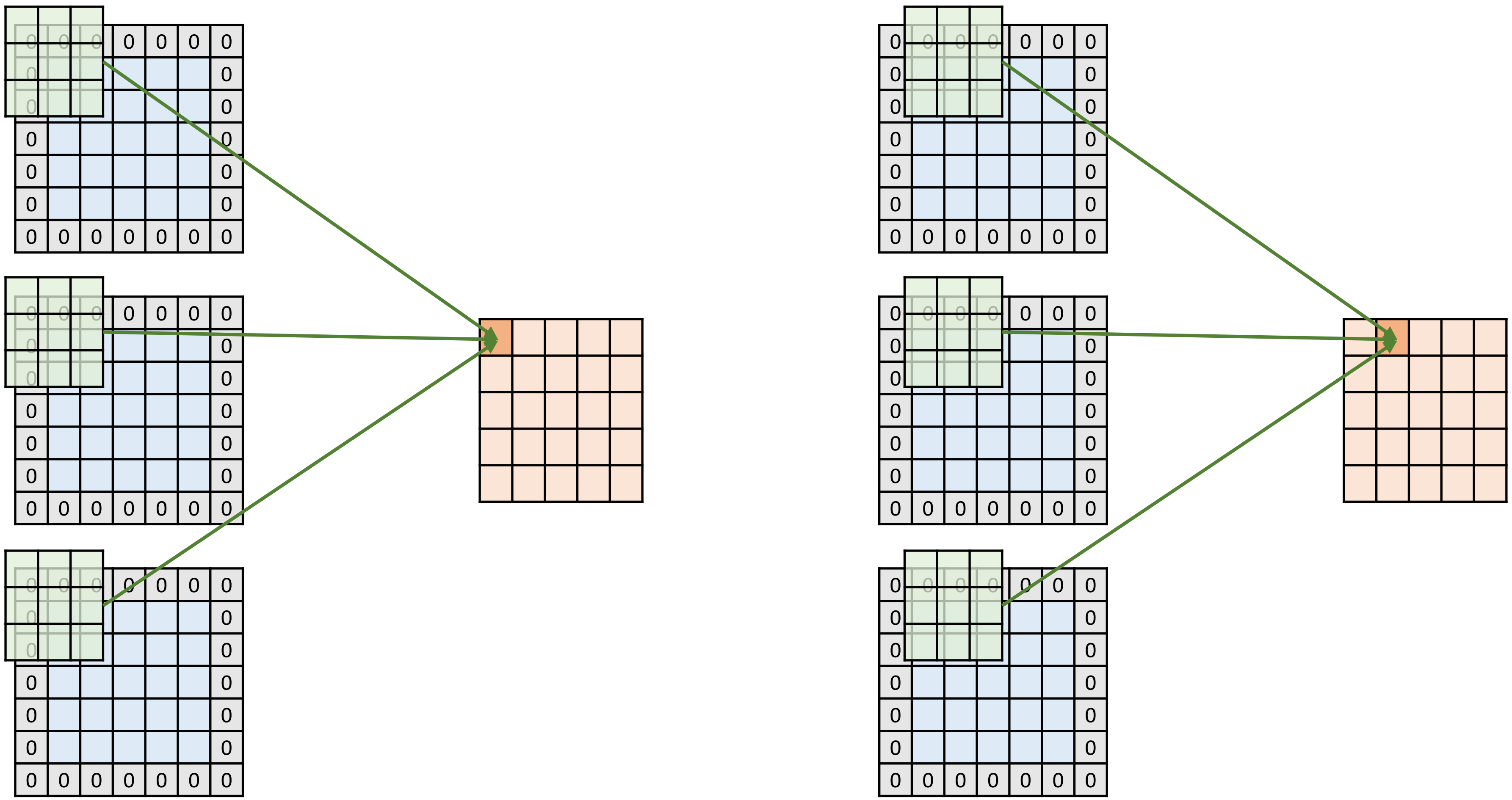
- Sliding filters via ‘for loops’
  - Slow, and we never do this

```
for row in range(x.shape[0] - 1):
    for col in range(x.shape[1] - 1):
        window = x[row: row + kernel_shape, col: col + kernel_shape]
        result[row, col] = np.sum(np.multiply(kernel,window))
```

[How Are Convolutions Actually Performed Under the Hood? | by Anirudh Shenoy | Towards Data Science]

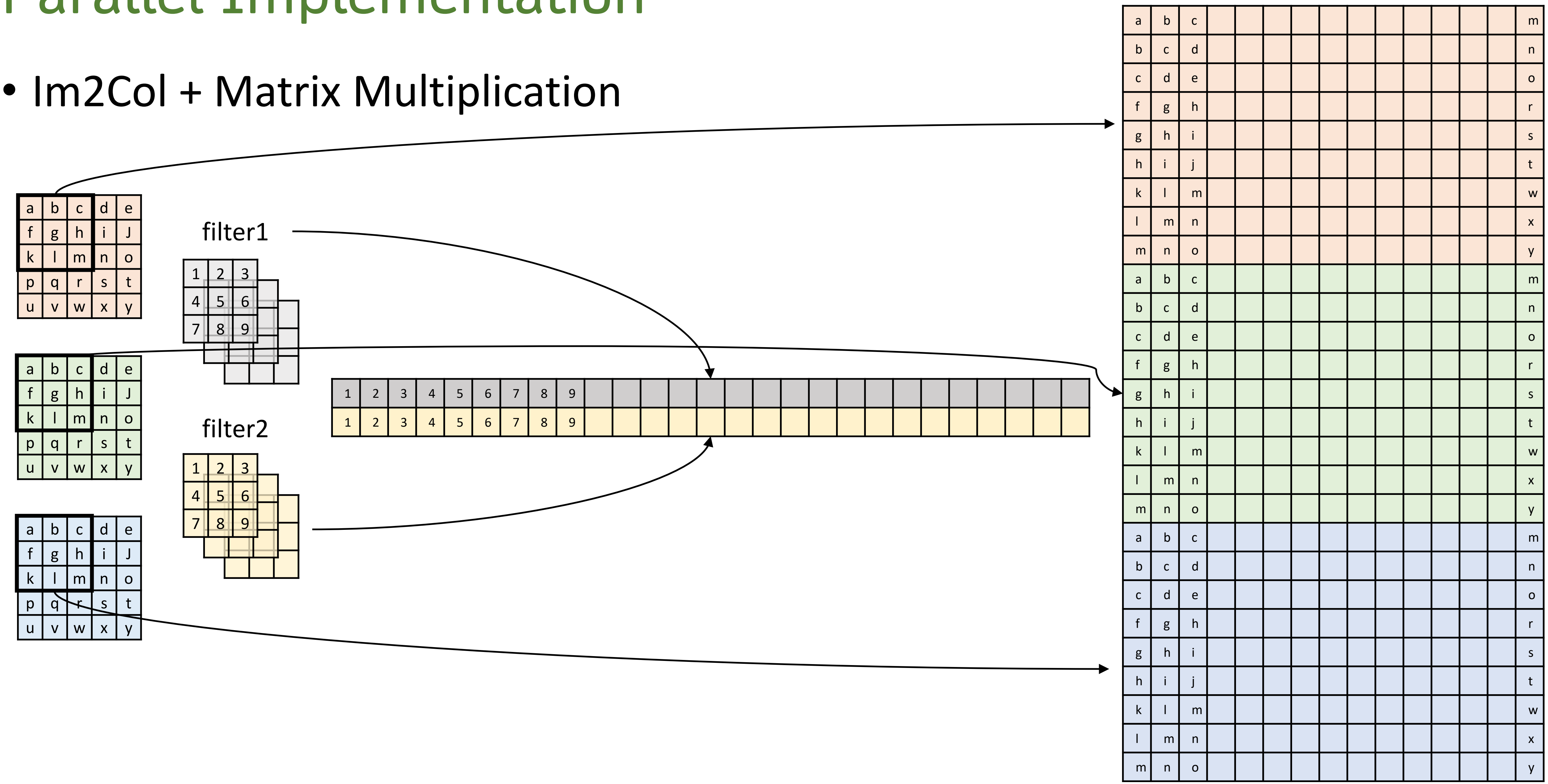
Parallel Implementation

- Convolutions are parallelizable



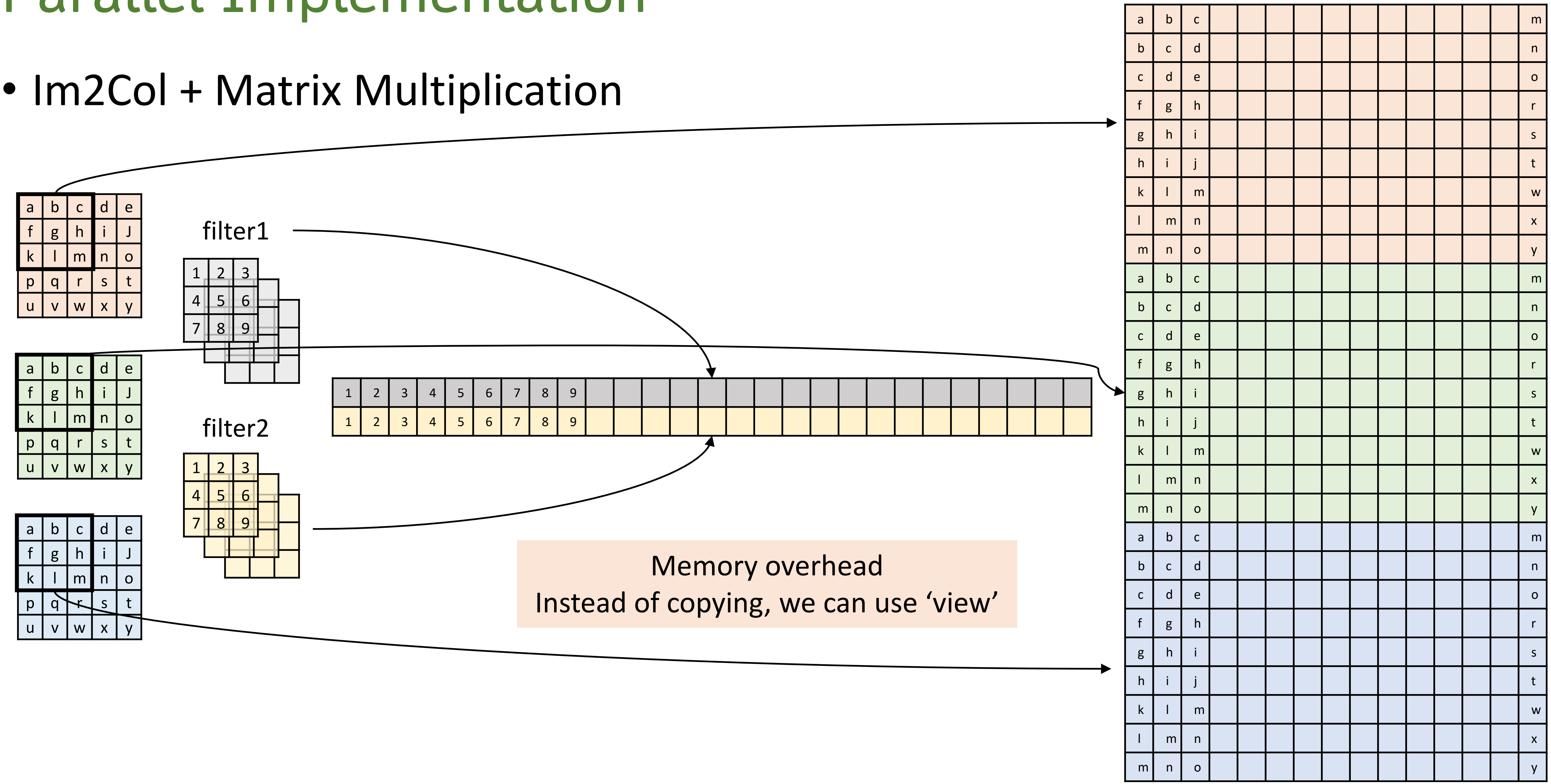
Parallel Implementation

- Im2Col + Matrix Multiplication



Parallel Implementation

- Im2Col + Matrix Multiplication



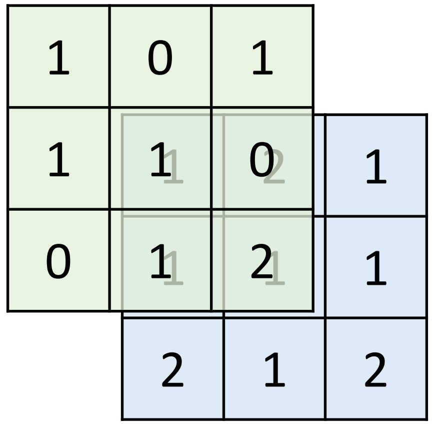
Further Optimizations

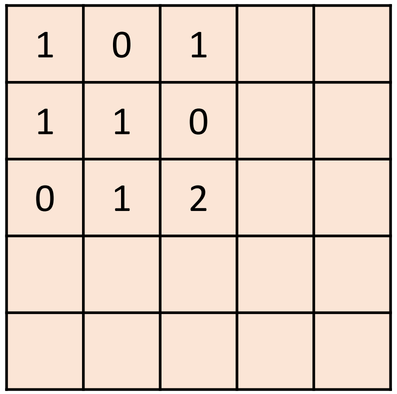
- GEMM optimization
- Winograd fast convolution
  - [1509.09308] Fast Algorithms for Convolutional Neural Networks (arxiv.org)
- FFT
  - For larger kernels

Backprop in Convolutional Layers

Transposed Convolution

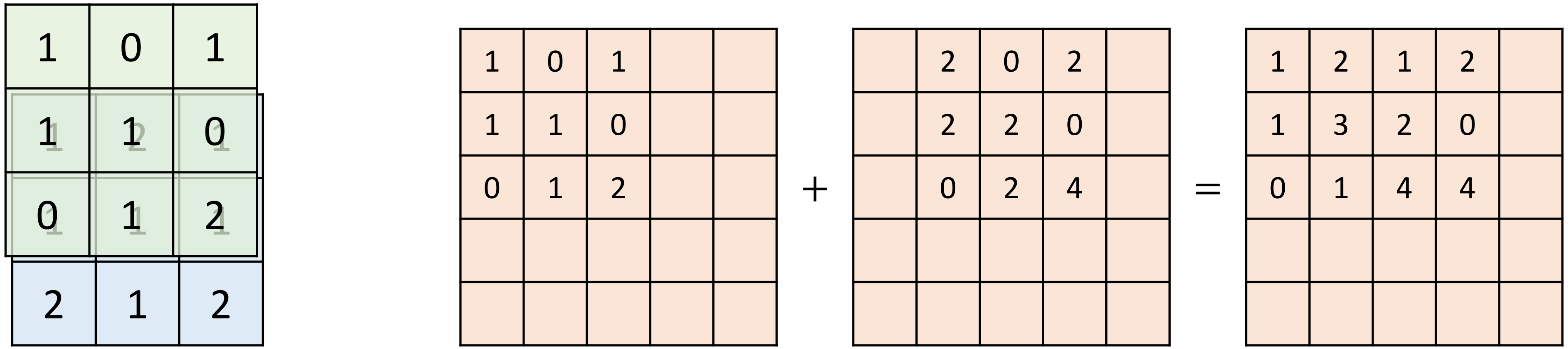
- Opposite of normal convolution
- Can be used to up-sampling





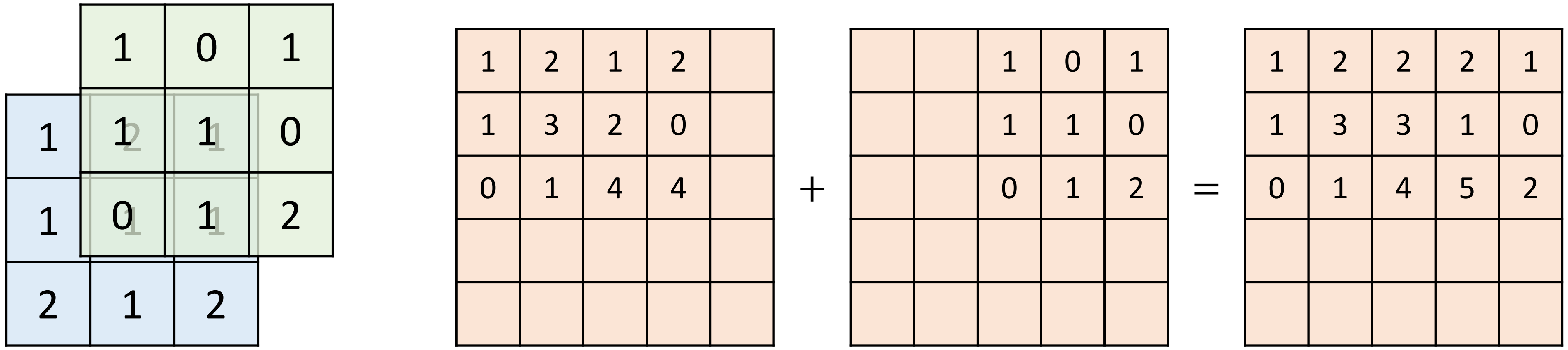
Transposed Convolution

- Opposite of normal convolution
- Can be used to up-sampling



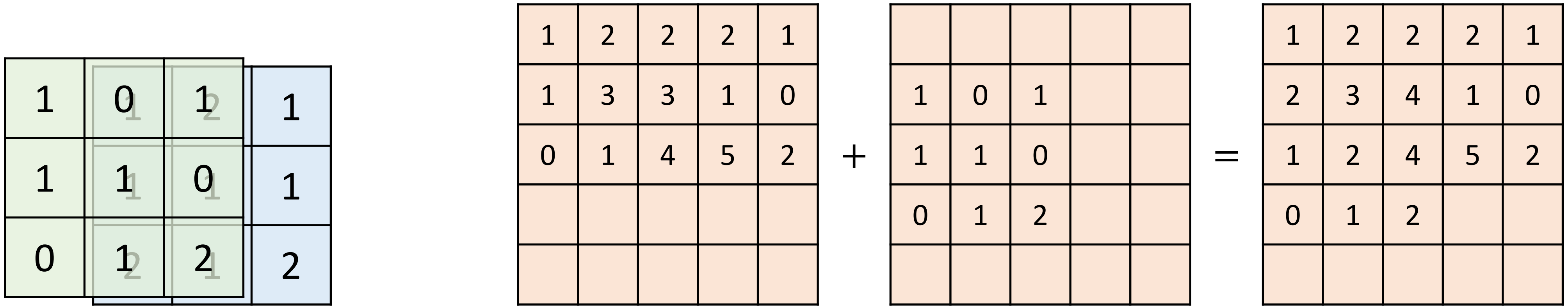
Transposed Convolution

- Opposite of normal convolution
- Can be used to up-sampling

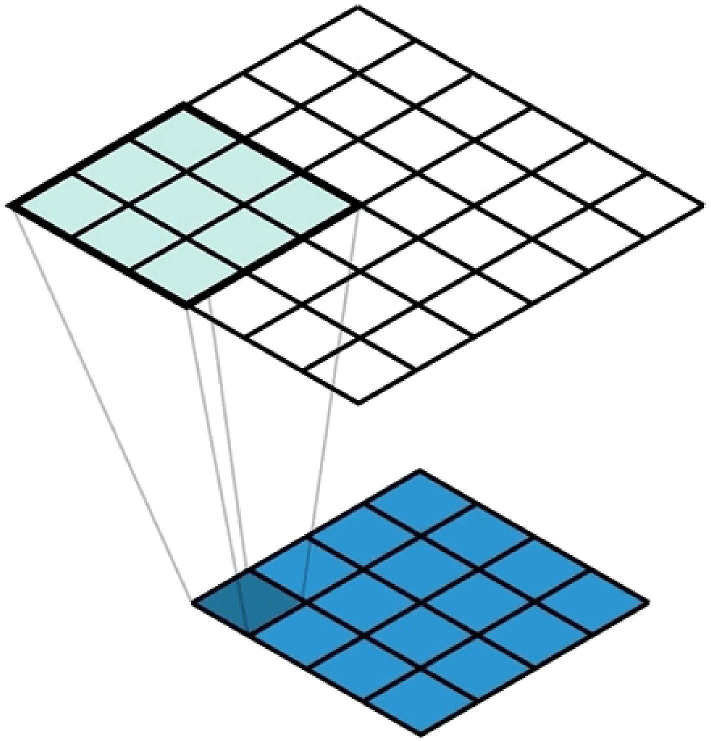


Transposed Convolution

- Opposite of normal convolution
- Can be used to up-sampling



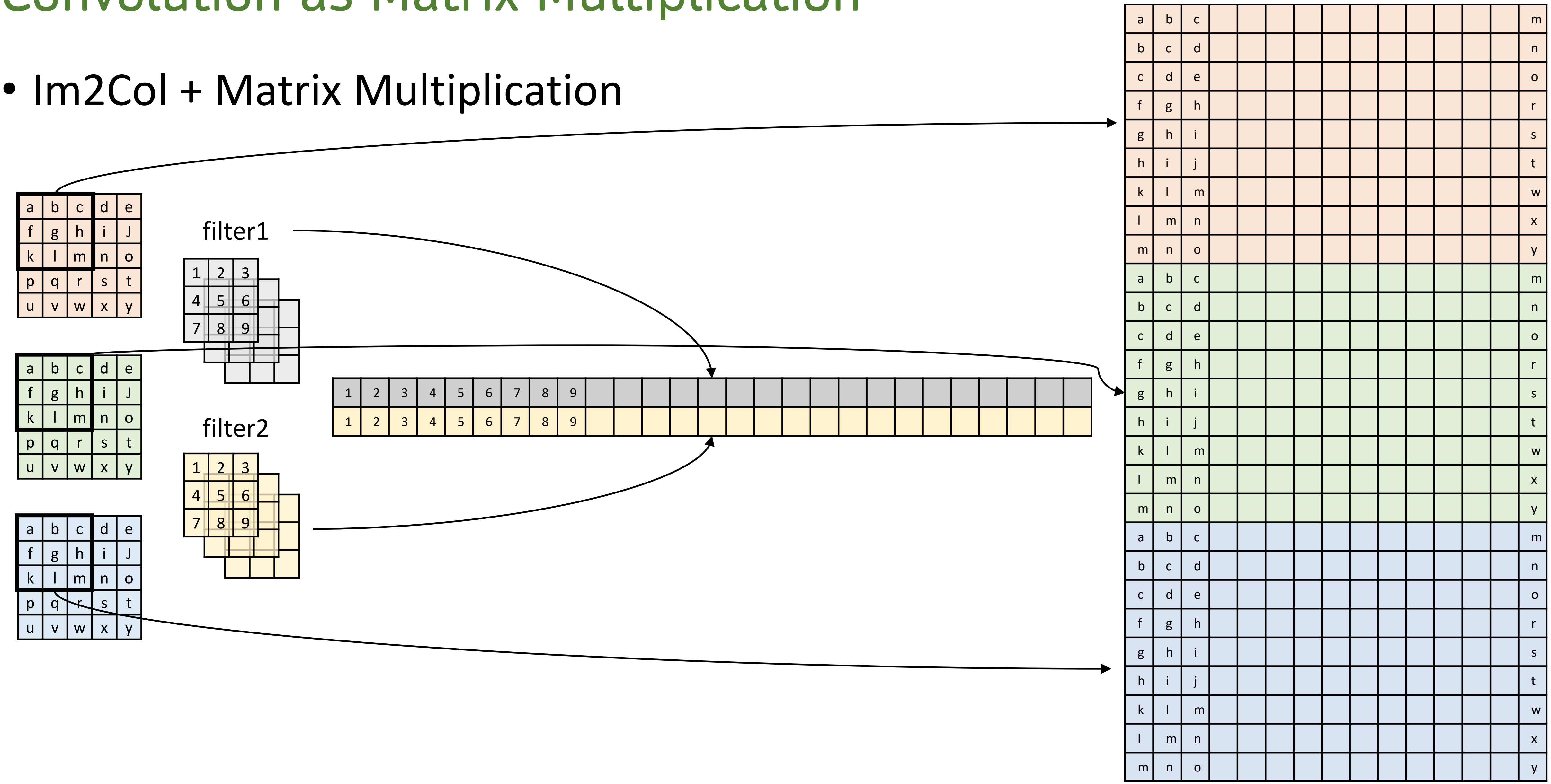
Transposed Convolution



[Transposed Convolutions explained with… MS Excel! | by Thom Lane | Apache MXNet | Medium]

Convolution as Matrix Multiplication

- Im2Col + Matrix Multiplication



Convolution as Matrix Multiplication

$X \in \mathbb{R}^{n\times d}, Y \in \mathbb{R}^{m\times d}, z \in \mathbb{R}, W \in \mathbb{R}^{m\times n}$

$Y = WX$

$\mathbb{R}^{n\times d}$ $\mathbb{R}^{n\times m}$ $\mathbb{R}^{m\times d}$

$$\frac{dz}{dX} = \frac{dz}{dY}\frac{dY}{dX} = W^\top \frac{dz}{dY}$$

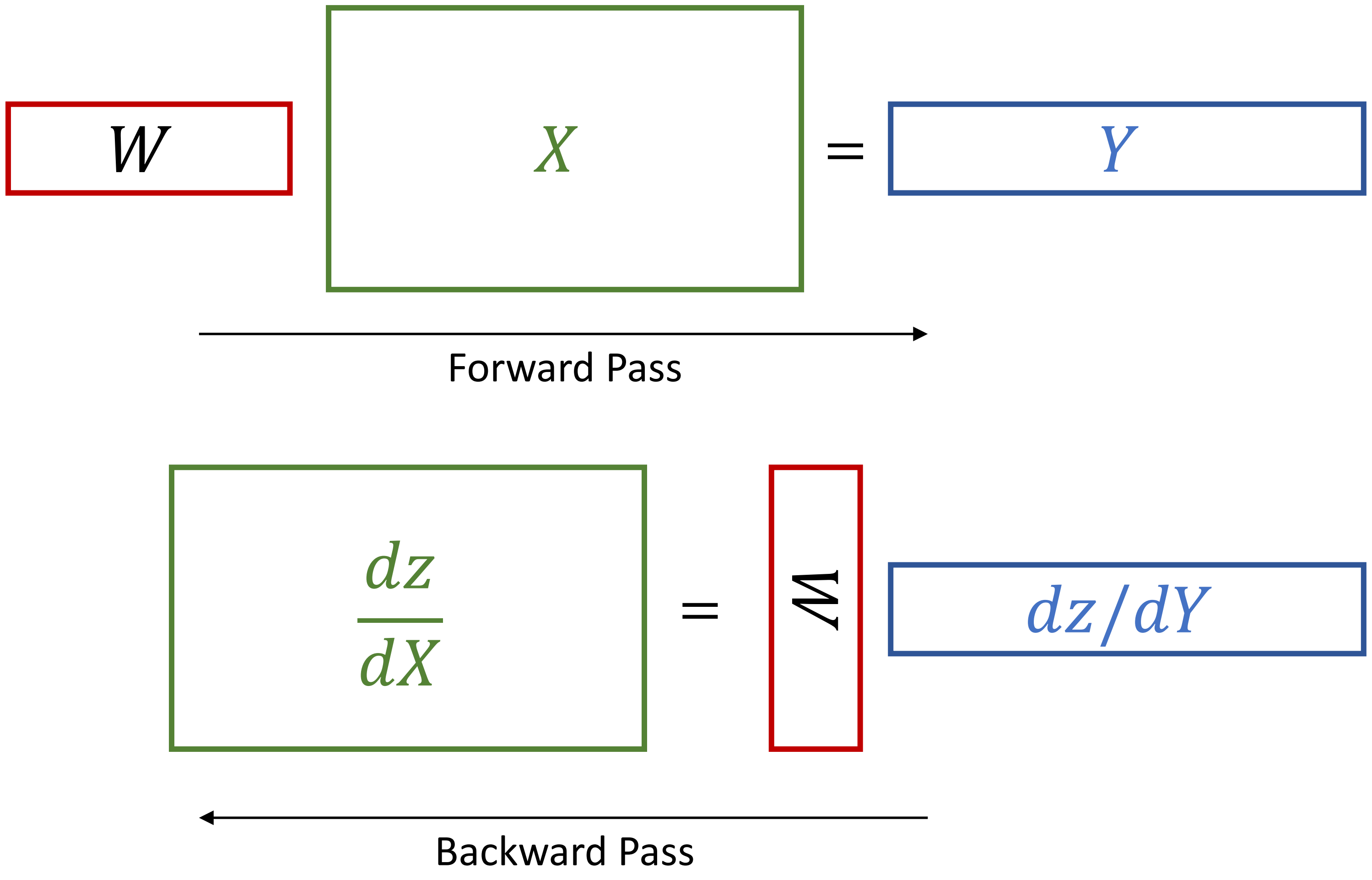
$\mathbb{R}^{m\times d}$ $\mathbb{R}^{m\times d\times n\times d}$

Forward and Backward

$X \in \mathbb{R}^{n \times d}, Y \in \mathbb{R}^{m \times d}, z \in \mathbb{R}, W \in \mathbb{R}^{m \times n}$

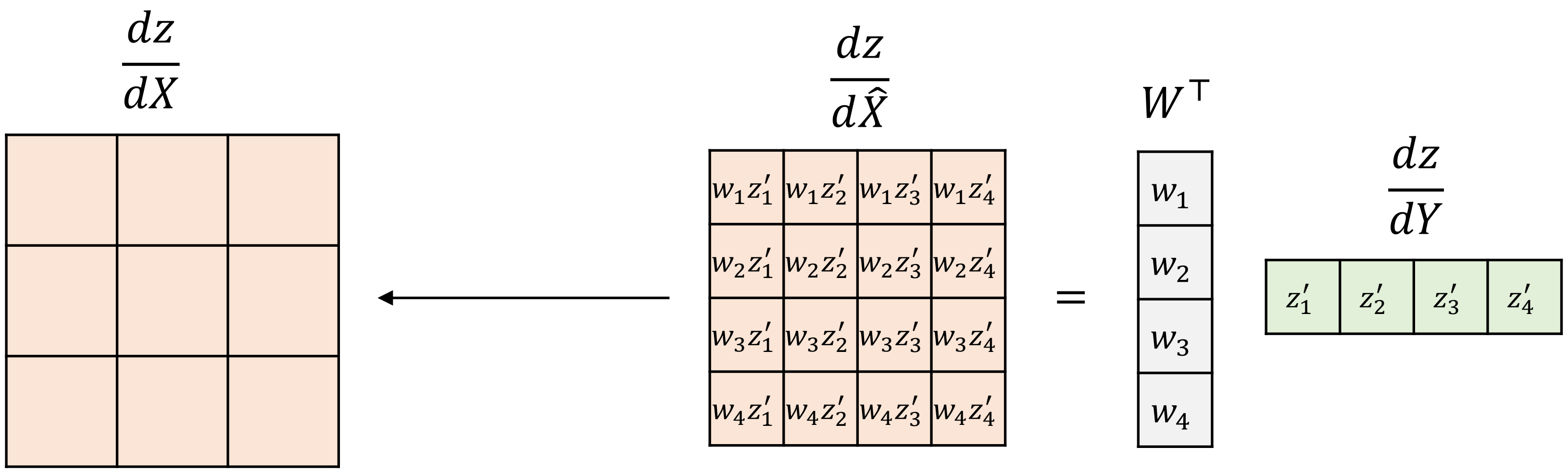
$Y = WX$

$\frac{dz}{dX} = W^{\top} \frac{dz}{dY}$



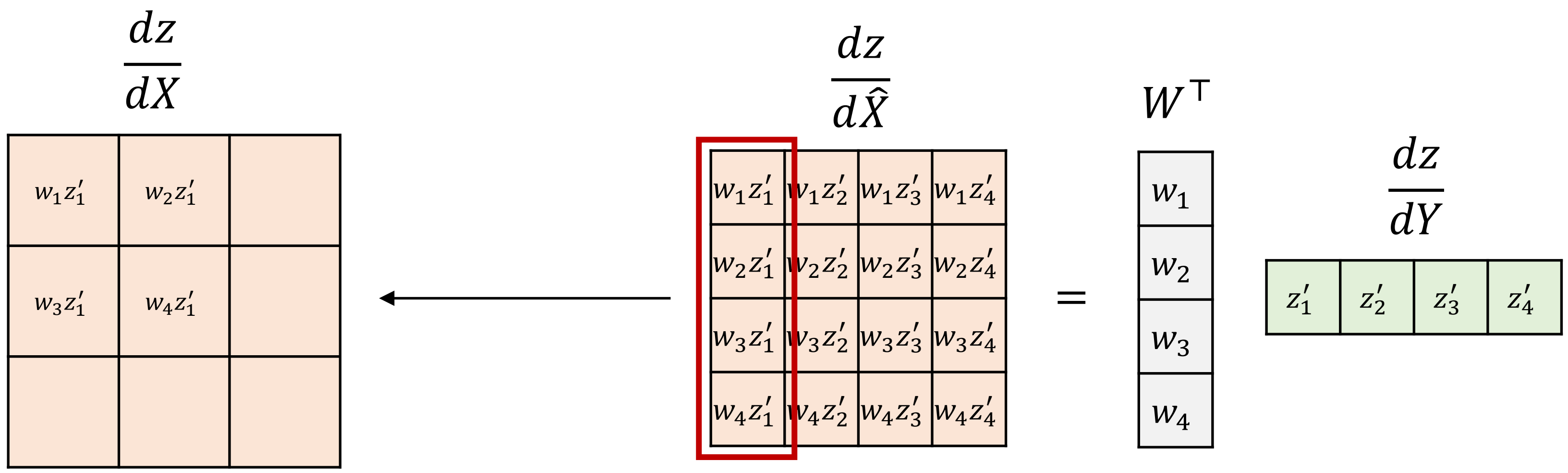
Backward Pass

- Matrix multiplication view



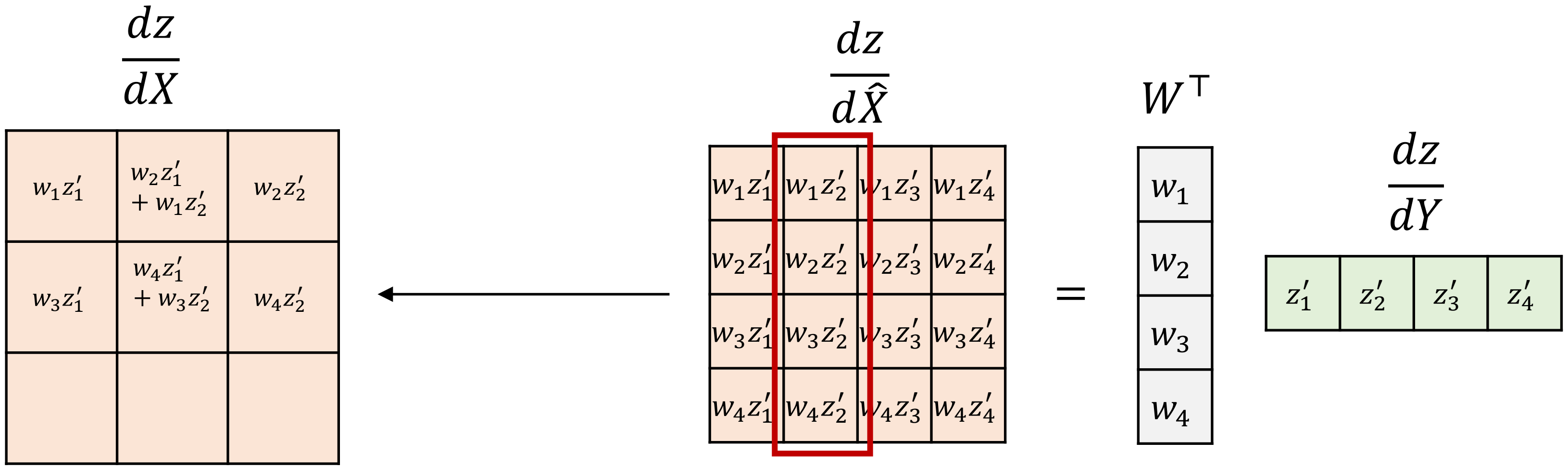
Backward Pass

- Matrix multiplication view



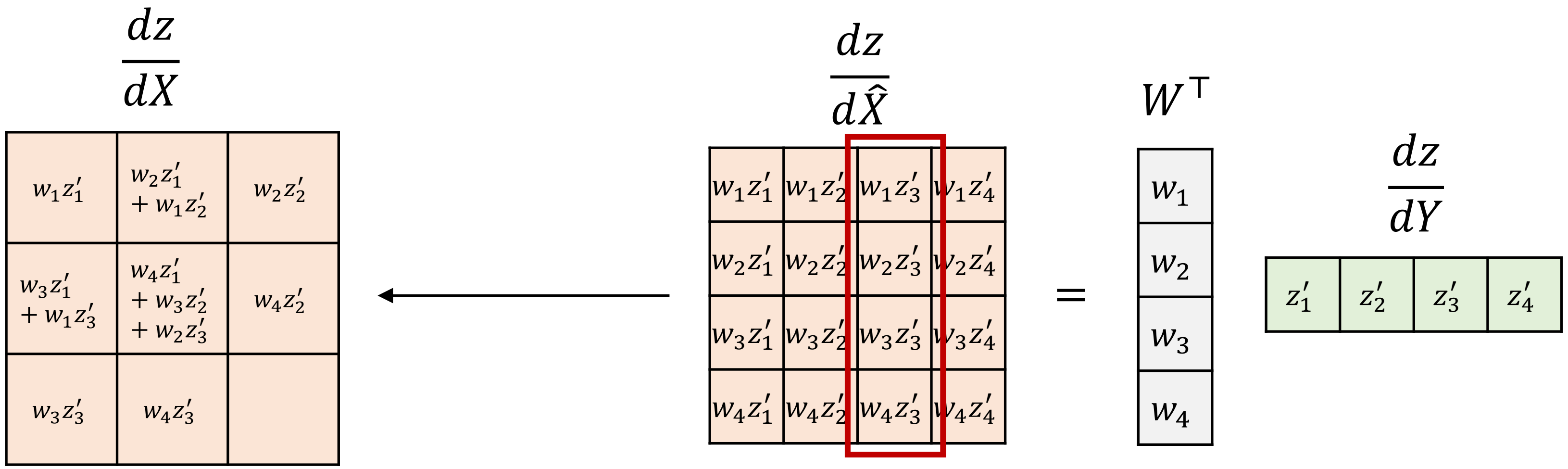
Backward Pass

- Matrix multiplication view



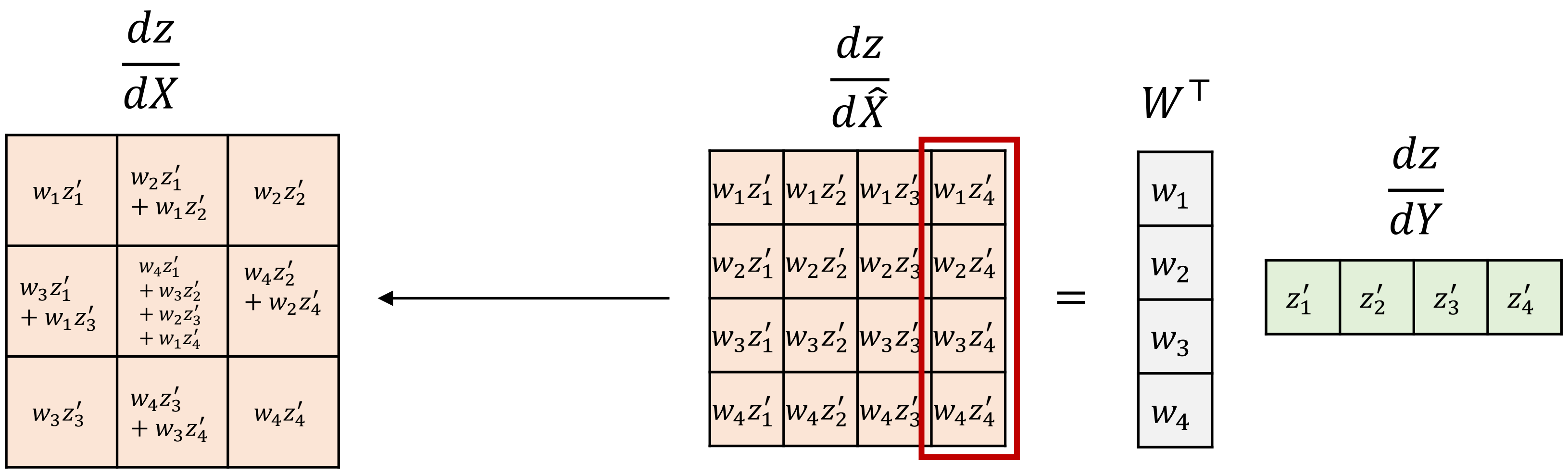
Backward Pass

- Matrix multiplication view



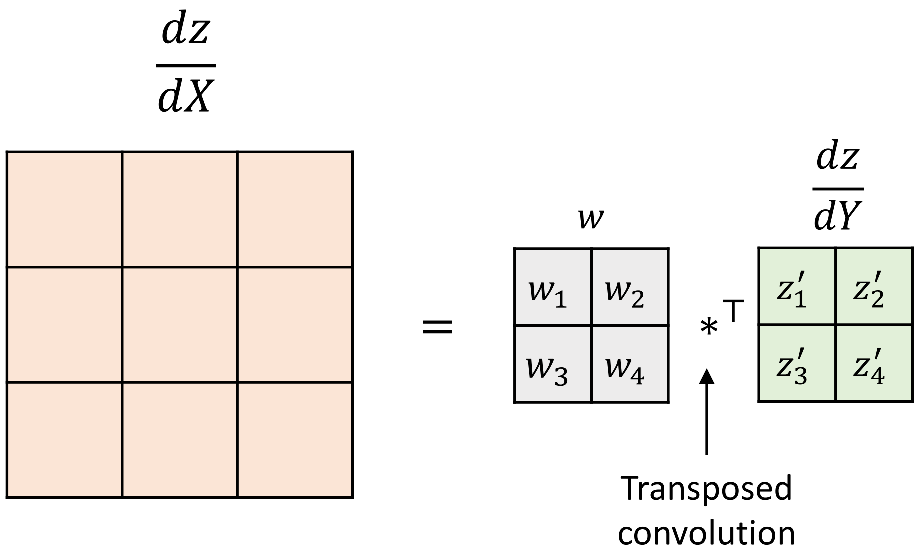
Backward Pass

- Matrix multiplication view



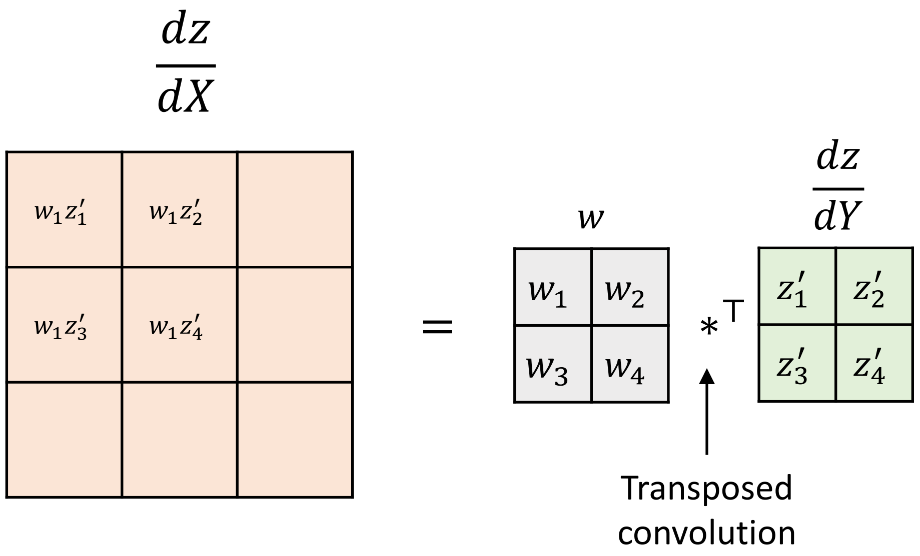
Backward Pass

- Matrix multiplication view



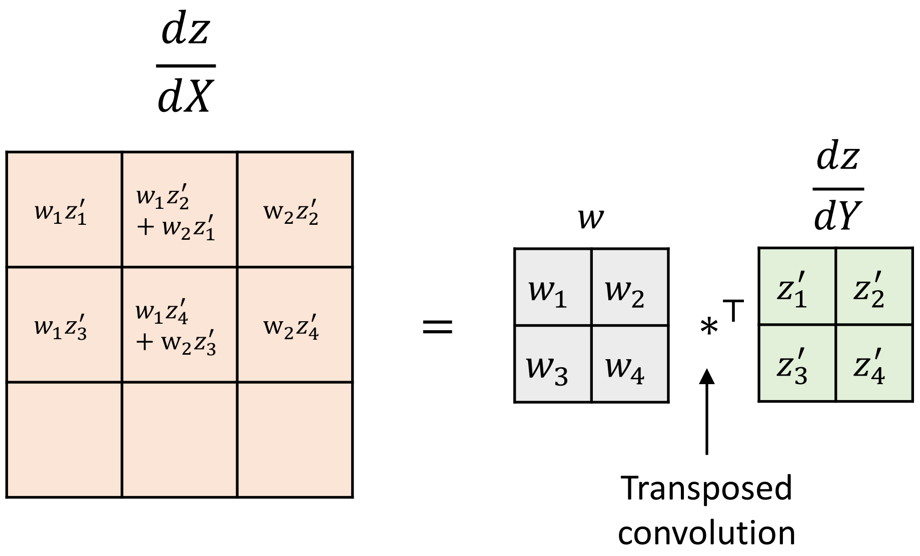
Backward Pass

- Matrix multiplication view



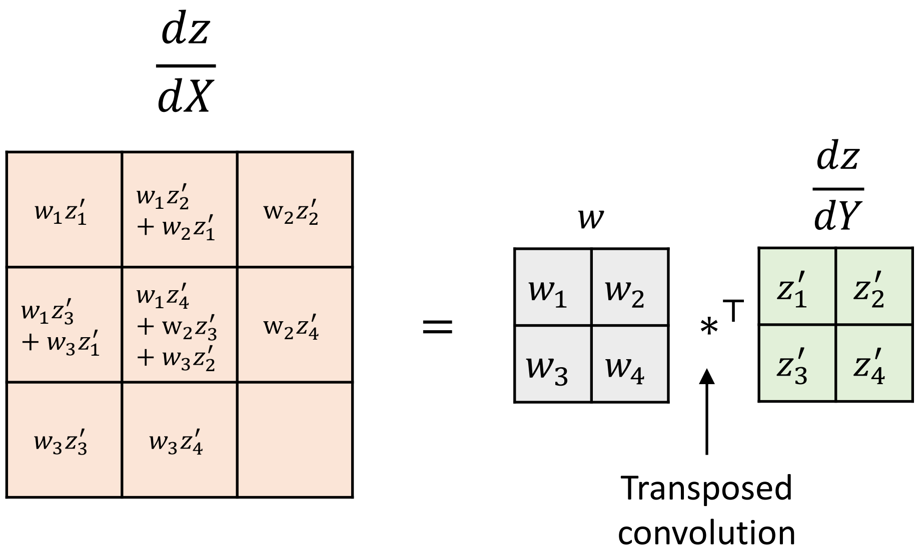
Backward Pass

- Matrix multiplication view



Backward Pass

- Matrix multiplication view



Backward Pass

- Matrix multiplication view