



성균관대학교  
SUNG KYUNKWAN UNIVERSITY

# Deep Learning

## - Convolutional Neural Networks 1 -

Eunbyung Park

Assistant Professor

School of Electronic and Electrical Engineering

[Eunbyung Park \(silverbottlep.github.io\)](https://silverbottlep.github.io)

# Convolution

# 1D Convolution

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

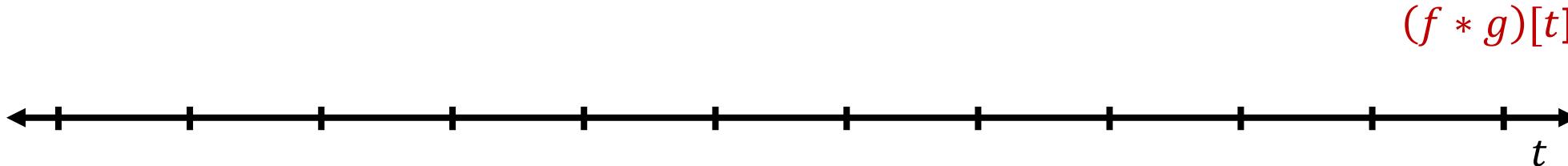
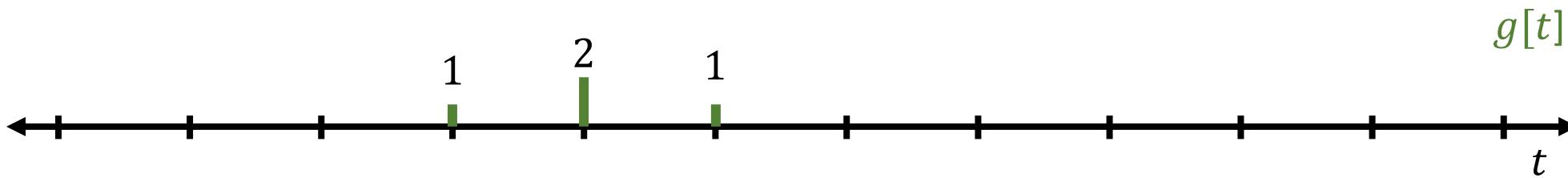
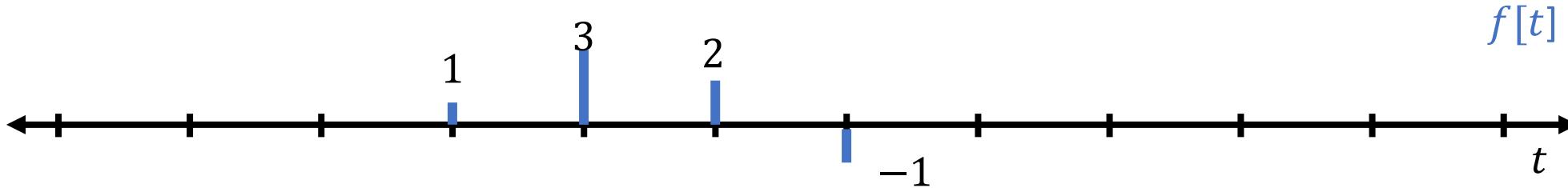
$$(f * g)(t) := \int_{-\infty}^{\infty} f(t - \tau)g(\tau)d\tau$$

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

# 1D Convolution

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

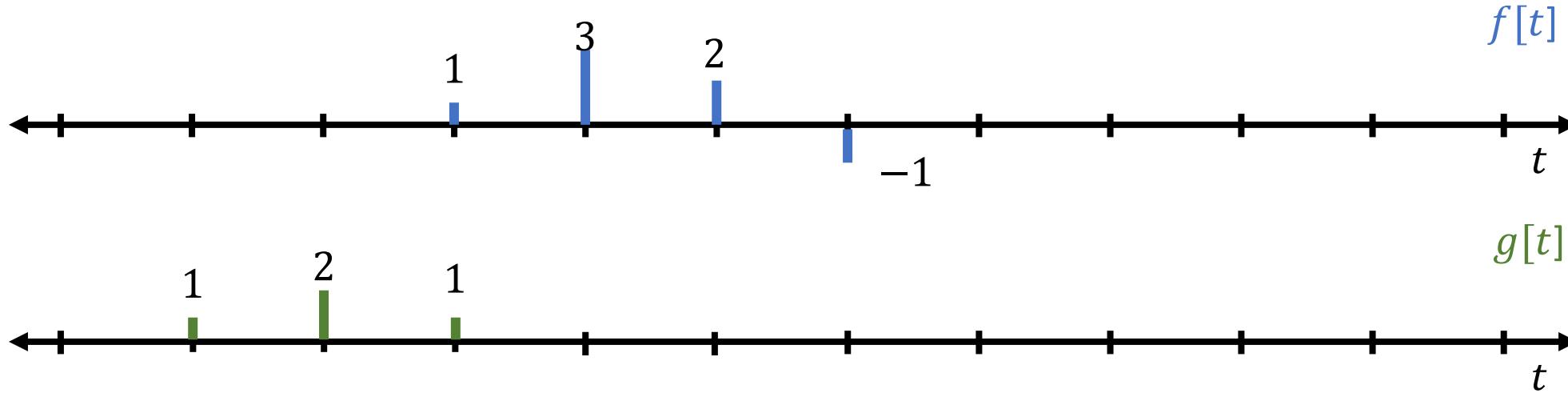
- Flip the filter and sliding



# 1D Convolution

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

- Flip the filter and sliding



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

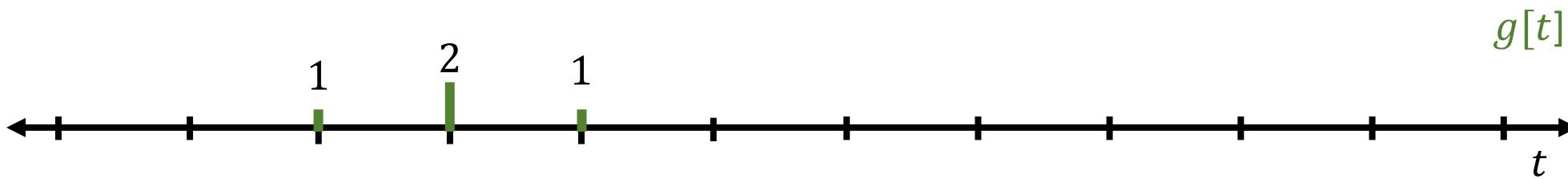
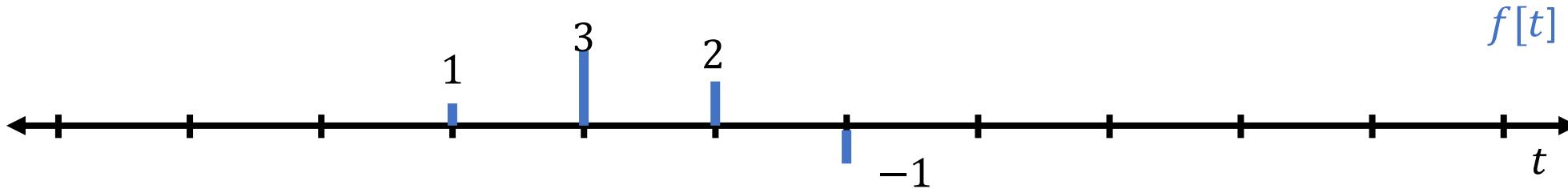
$$(f * g)[t]$$



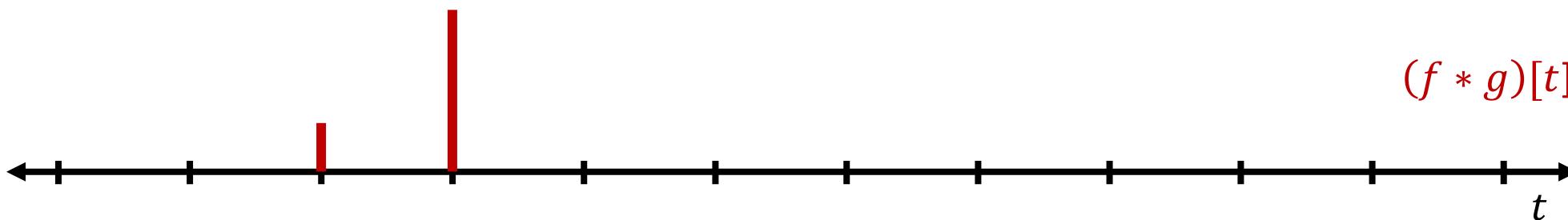
# 1D Convolution

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

- Flip the filter and sliding



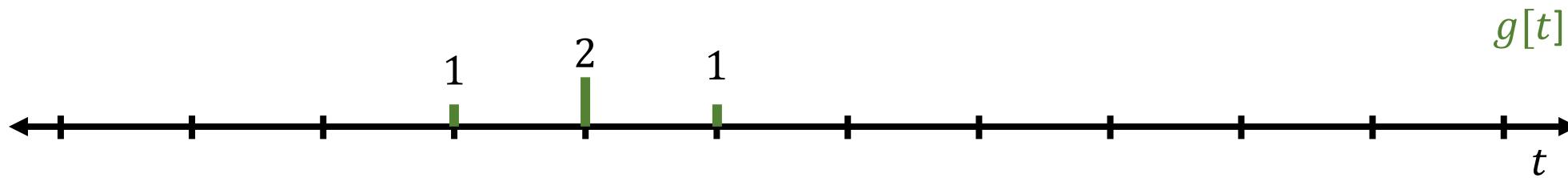
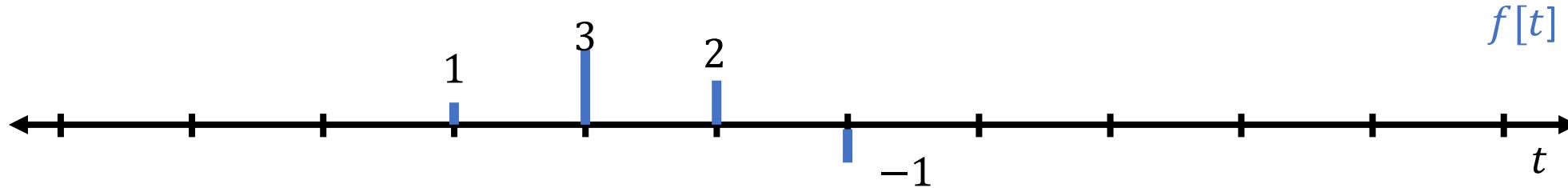
$$0 \cdot 1 + 1 \cdot 2 + 3 \cdot 1 = 5$$



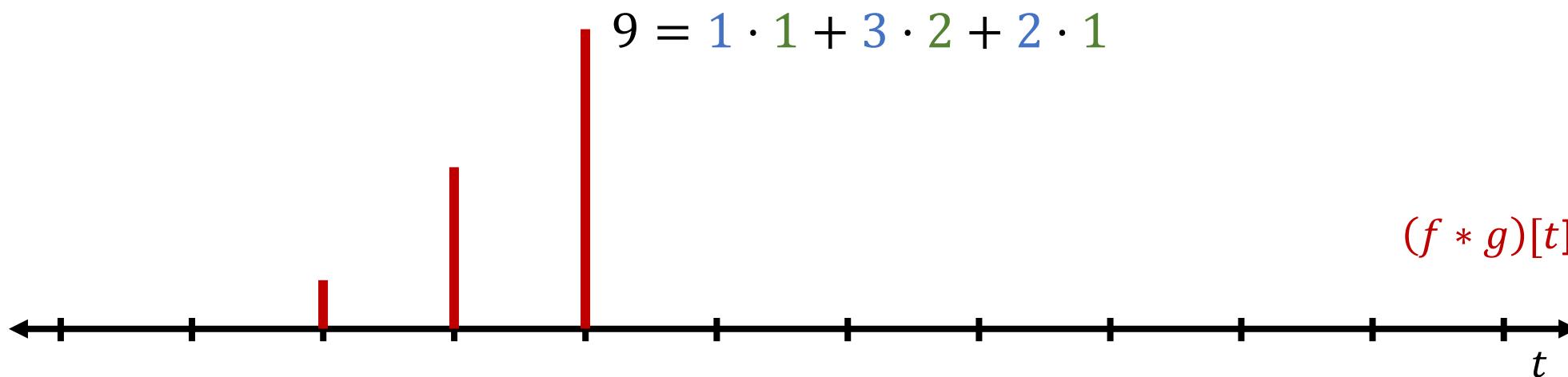
# 1D Convolution

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

- Flip the filter and sliding



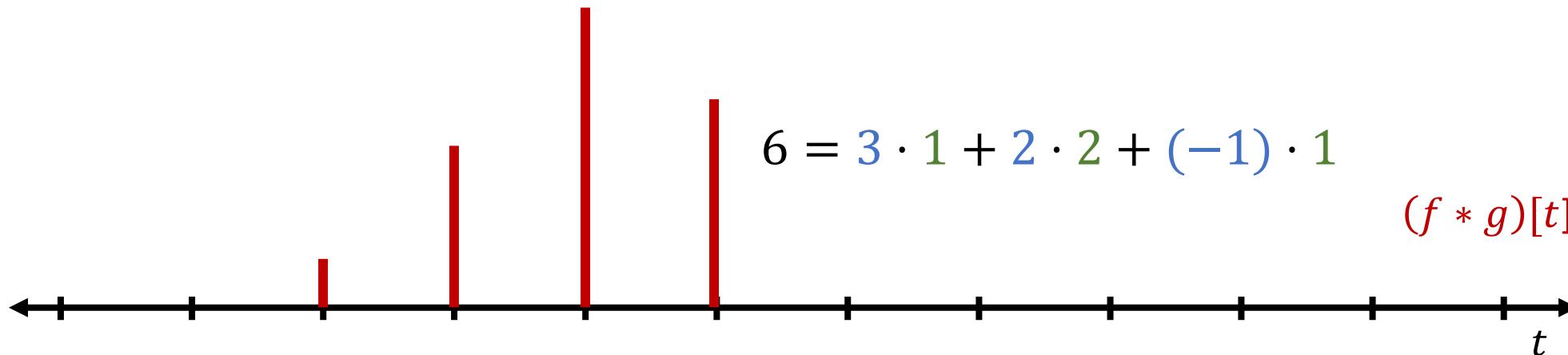
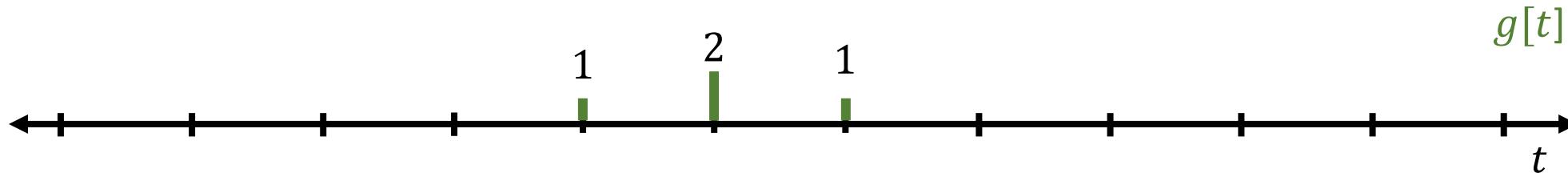
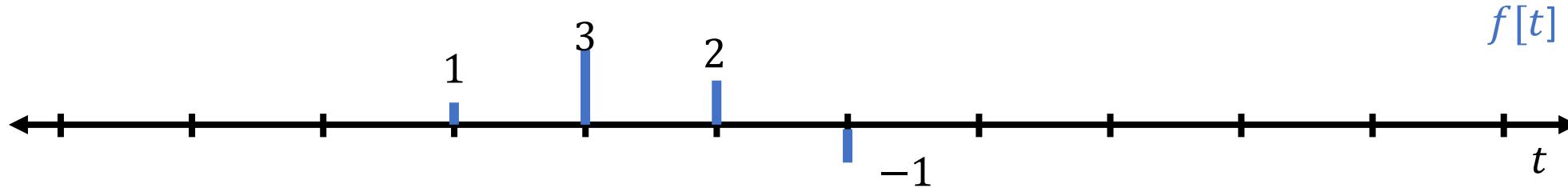
$$9 = 1 \cdot 1 + 3 \cdot 2 + 2 \cdot 1$$



# 1D Convolution

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

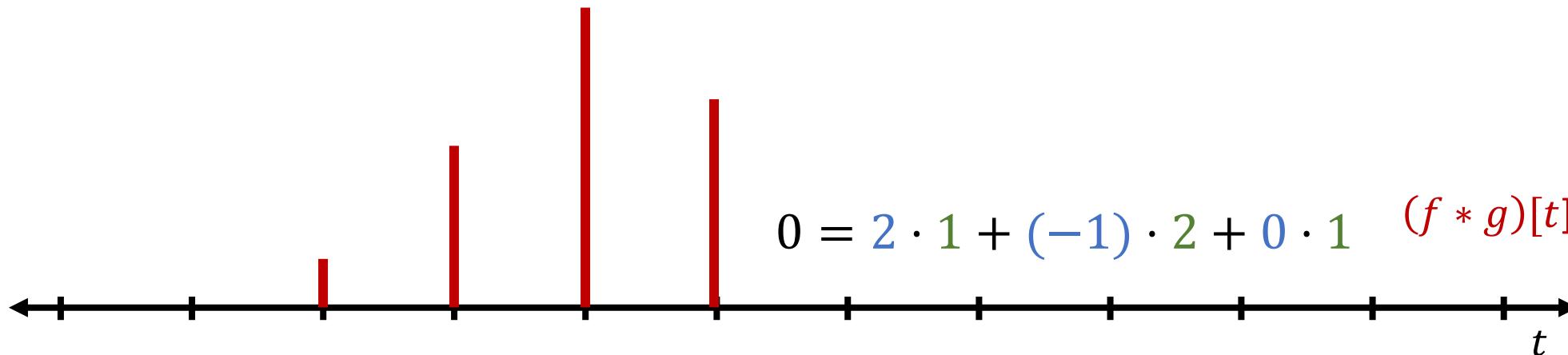
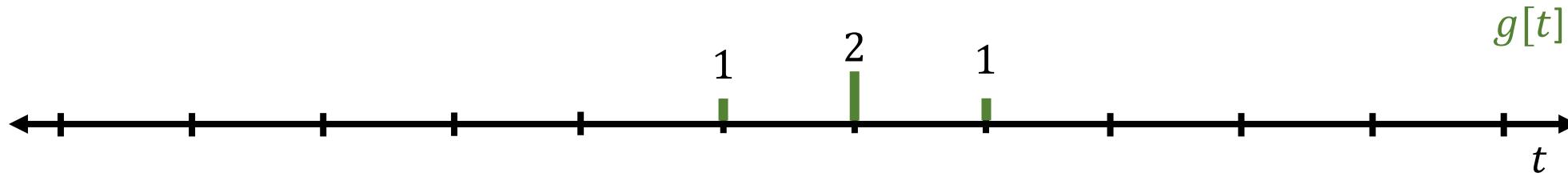
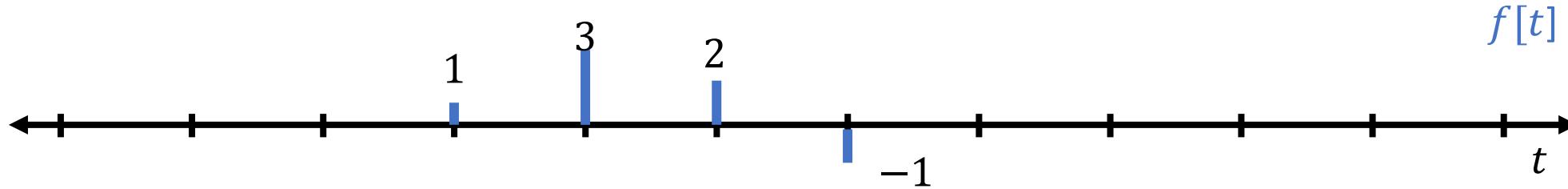
- Flip the filter and sliding



# 1D Convolution

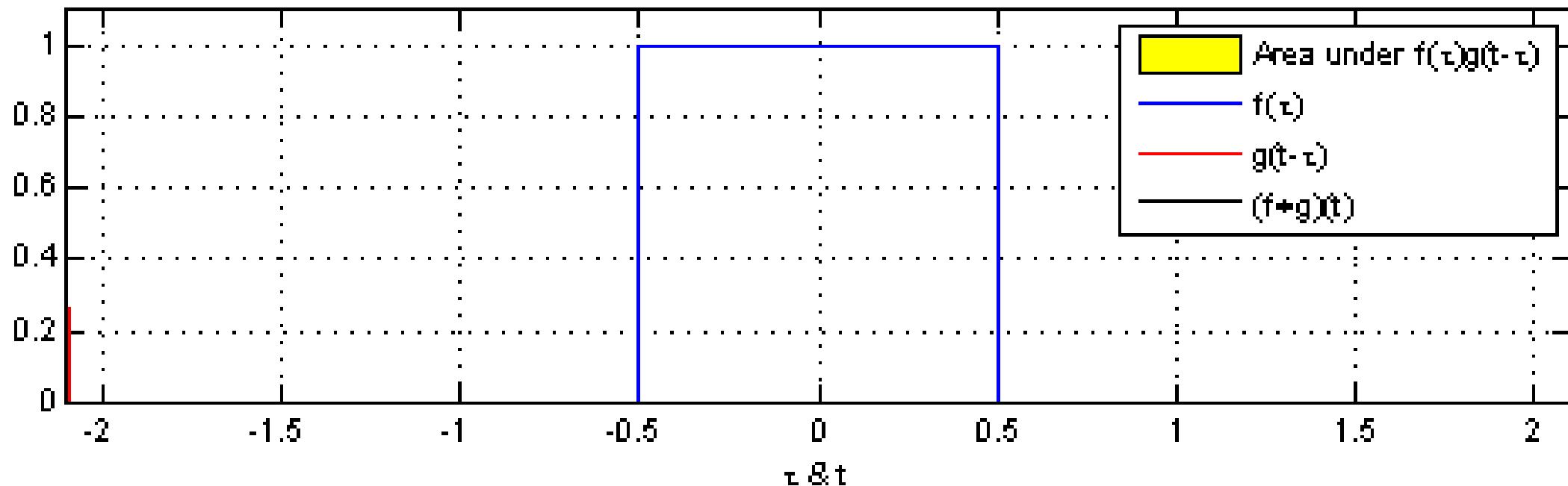
$$(f * g)[t] := \sum_{\tau} f[t - \tau]g[\tau]$$

- Flip the filter and sliding



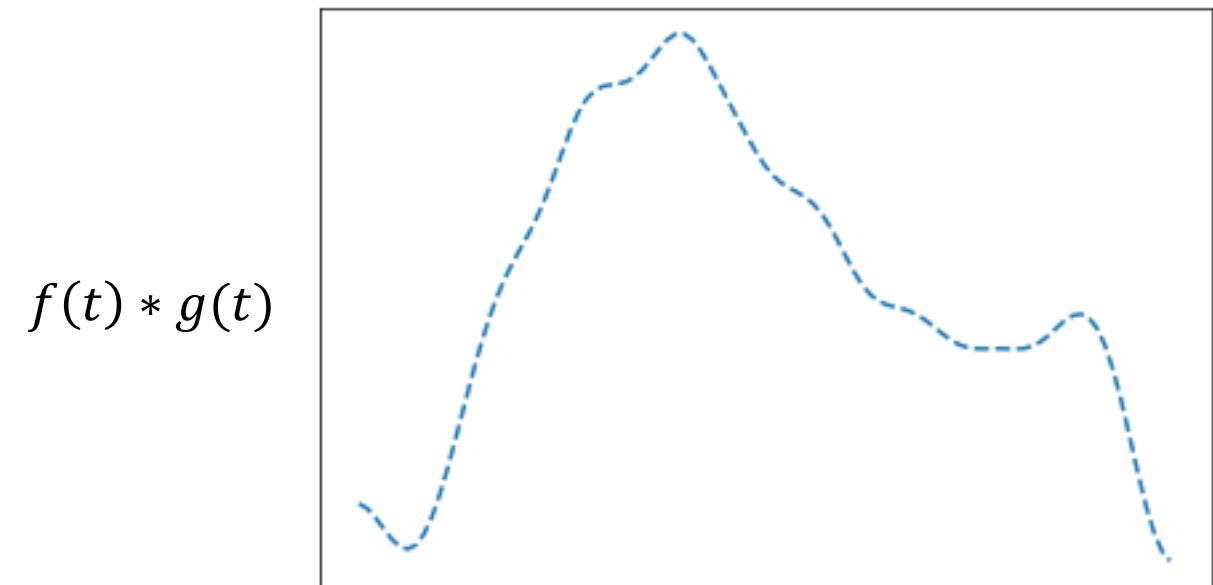
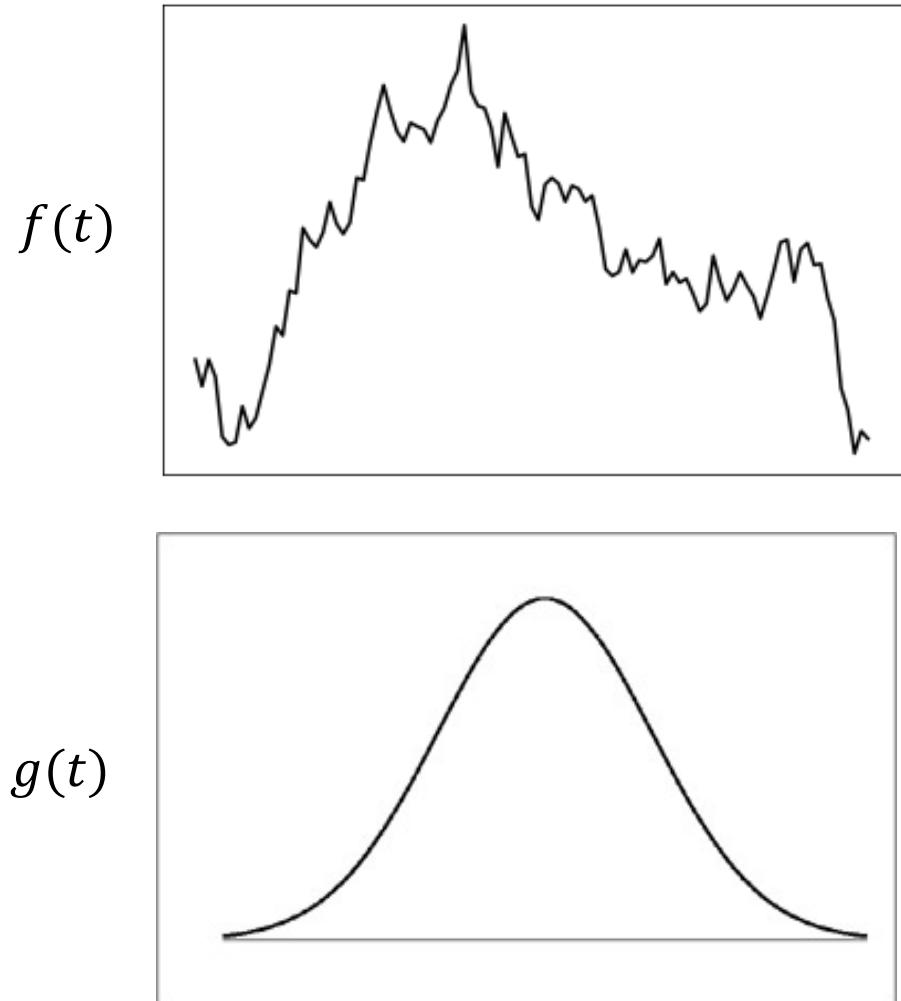
# 1D Convolution

- Example



# 1D Convolution

- Gaussian filter



## 2D Convolution

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

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

# 2D Convolution

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

1 <sub>0</sub>	3 <sub>0</sub>	2 <sub>0</sub>	0	0	0	0
1 <sub>0</sub>	3 <sub>1</sub>	3 <sub>3</sub>	2	3	3	0
3 <sub>0</sub>	1 <sub>3</sub>	1 <sub>1</sub>	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

16				

# 2D Convolution

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

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

16	28			

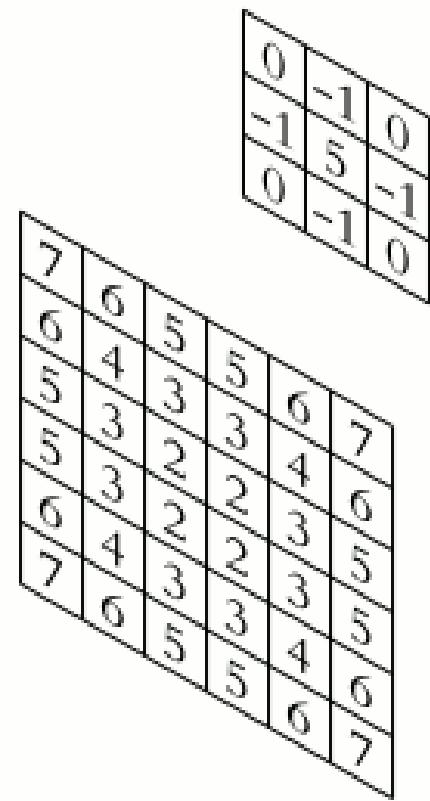
# 2D Convolution

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

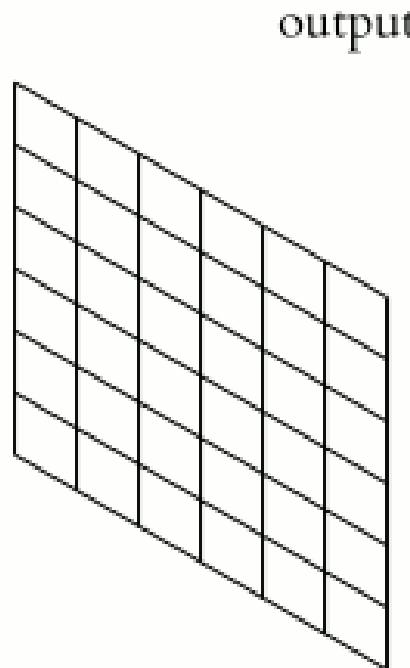
0	0	1	3	2	0	0
0	1	1	3	3	3	0
0	3	3	1	1	1	0
0	3	3	3	1	2	0
0	2	2	1	2	1	0
0	2	3	2	1	2	0
0	0	0	0	0	0	0

16	28	24		

# 2D Convolution



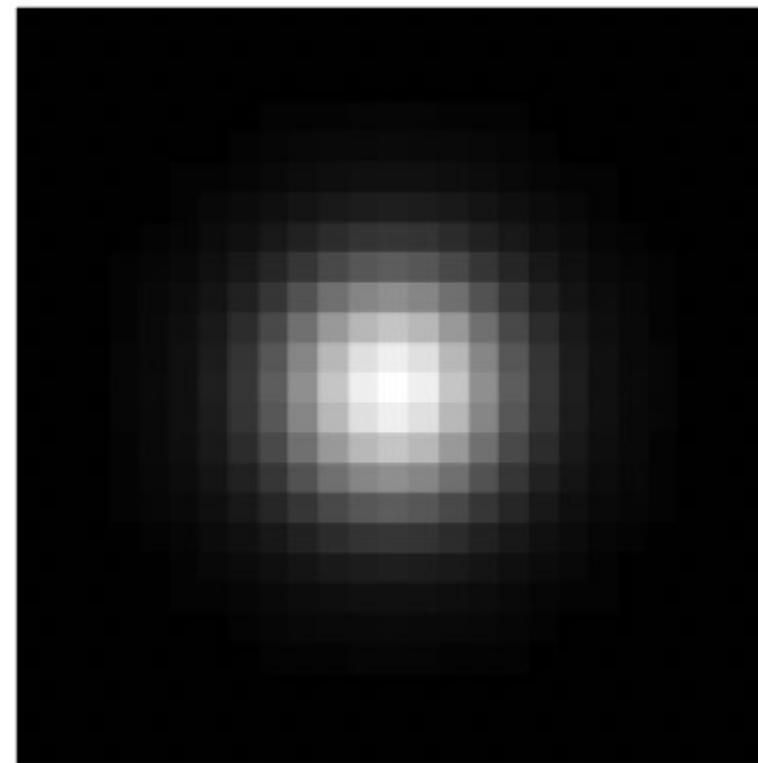
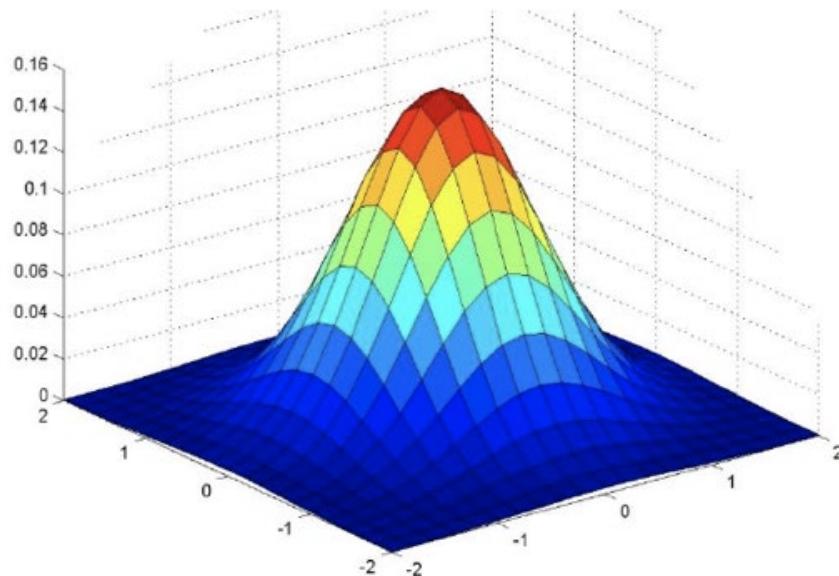
input



output

# 2D Convolution

- 2D gaussian filter



# 2D Convolution

- 2D gaussian filter

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



[2-D Gaussian filtering of images - MATLAB imgaussfilt \(mathworks.com\)](#)

# Properties of Convolution

$$f * g = g * f$$

Commutative

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

Associative

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

Linear

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

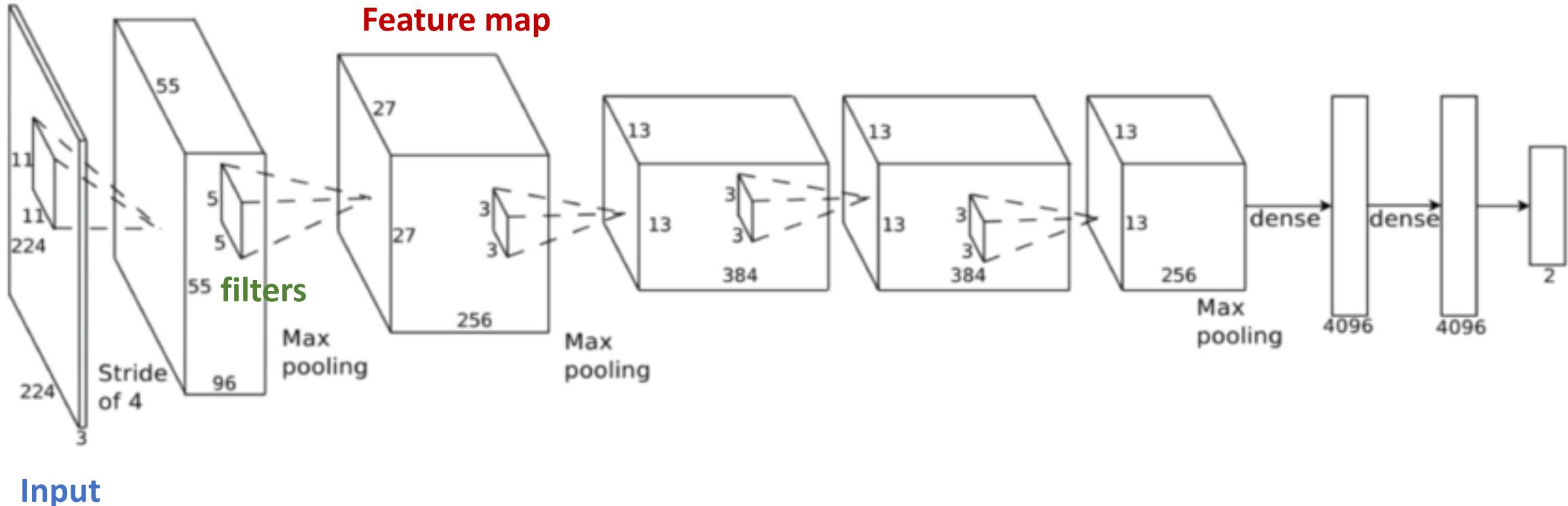
$$(L_t f) * h = L_t(f * h)$$

Translation equivariance

Translate  $f$  by  $t$

# Convolutional Neural Networks

# Convolutional Neural Network



# 2D Convolution

- Input\_channel=1, output\_channel=1, padding=1, stride=1

1 <sub>0</sub>	3 <sub>0</sub>	2 <sub>0</sub>	0	0	0	0
1 <sub>0</sub>	3 <sub>1</sub>	3 <sub>3</sub>	2	3	3	0
3 <sub>0</sub>	1 <sub>3</sub>	1 <sub>1</sub>	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

16				

# 2D Convolution

- Input\_channel=1, output\_channel=1, padding=1, stride=1

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

16	28			

# 2D Convolution

- Input\_channel=1, output\_channel=1, padding=1, stride=1

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

16	28	24		

# 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

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

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**

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

16		

# 2D Convolution

- Input\_channel=1, output\_channel=1, padding=1, stride=2

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

16	24	

# 2D Convolution

- Input\_channel=1, output\_channel=1, padding=1, **stride=2**

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

16	24	16

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

16	24	16
33		

# 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

$$\begin{matrix} & * & \begin{matrix} 1 & 3 & 2 \\ 1 & 3 & 3 \\ 3 & 1 & 1 \end{matrix} & = & \begin{matrix} 16 & 24 & 16 \\ 33 & 33 & 18 \\ 25 & 21 & 12 \end{matrix} \end{matrix}$$

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

\*

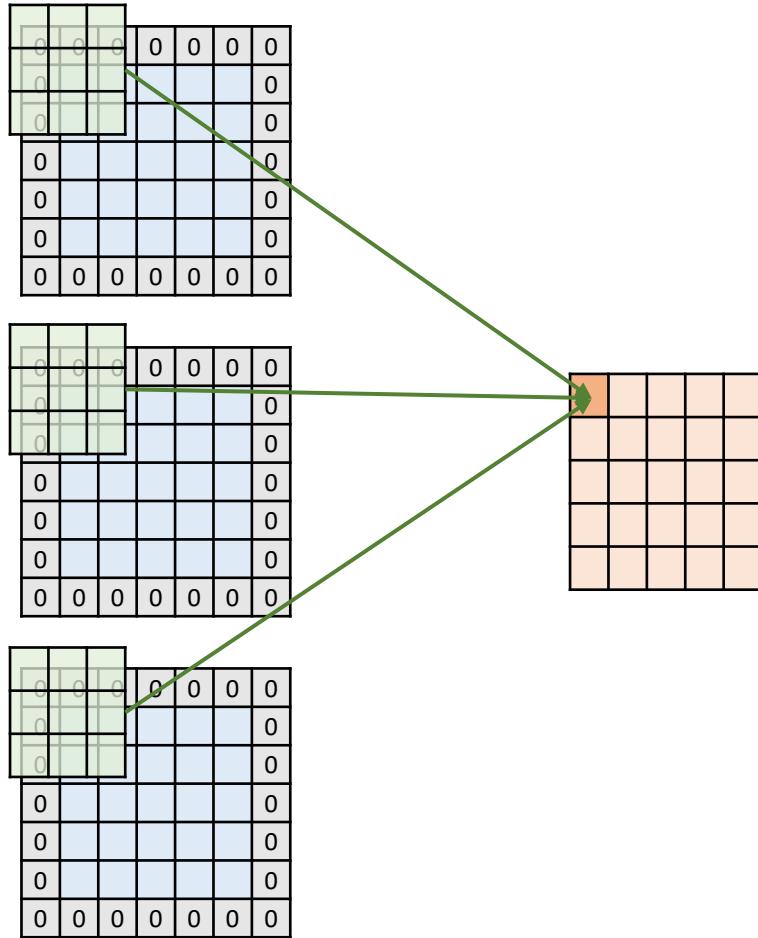
1	3	2
1	3	3
3	1	1

=

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

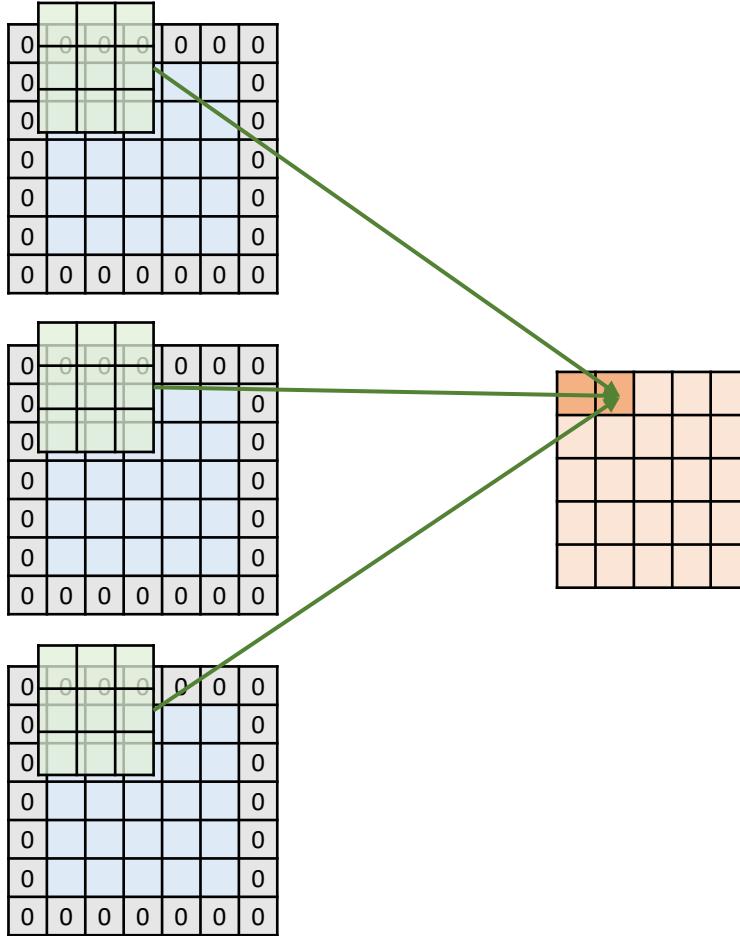
# 2D Convolution

- Input\_channel=3, output\_channel=1, padding=1, stride=1



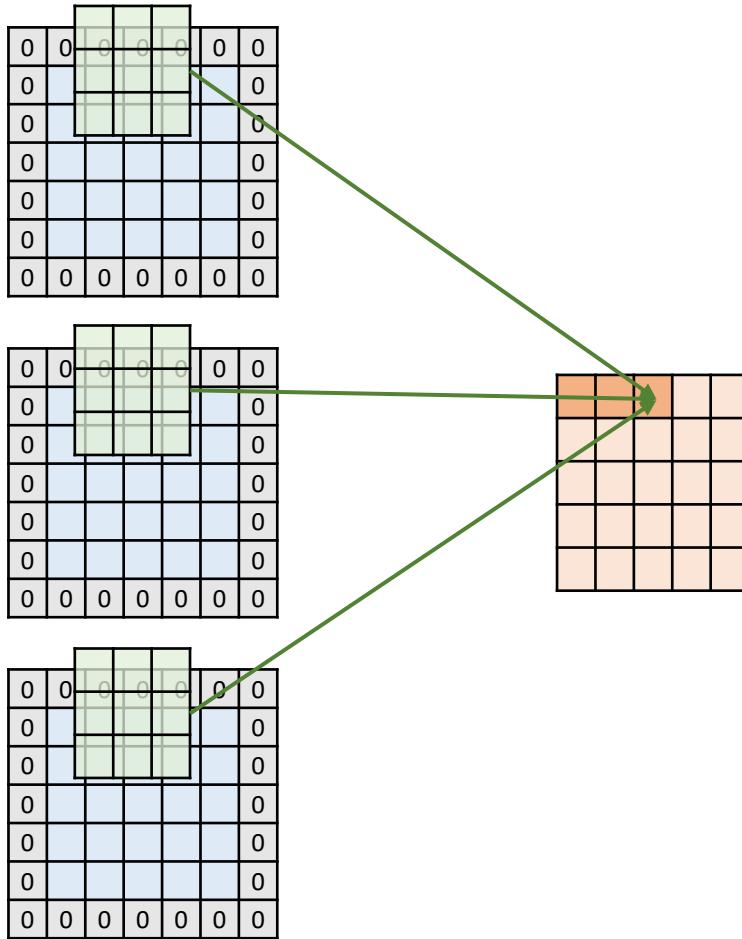
# 2D Convolution

- Input\_channel=3, output\_channel=1, padding=1, stride=1



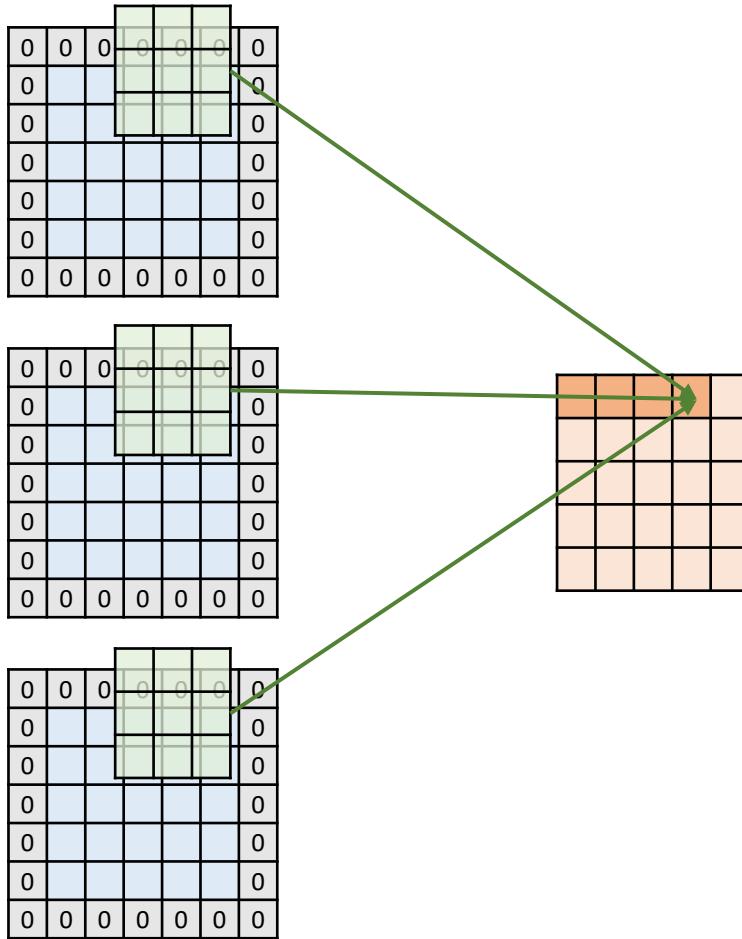
# 2D Convolution

- Input\_channel=3, output\_channel=1, padding=1, stride=1



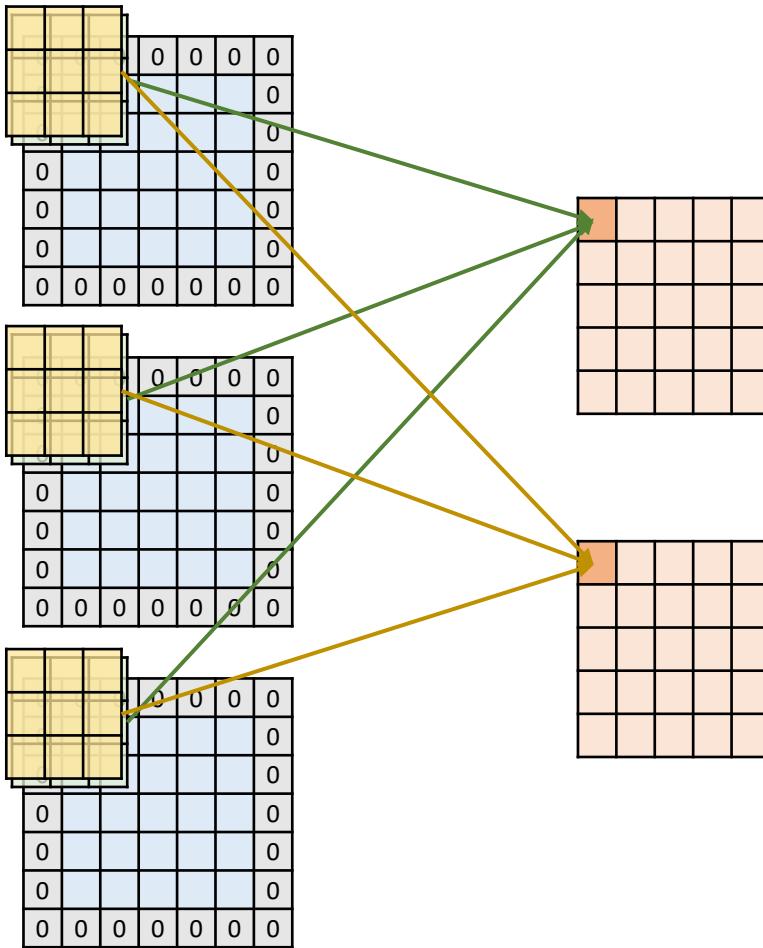
# 2D Convolution

- Input\_channel=3, output\_channel=1, padding=1, stride=1



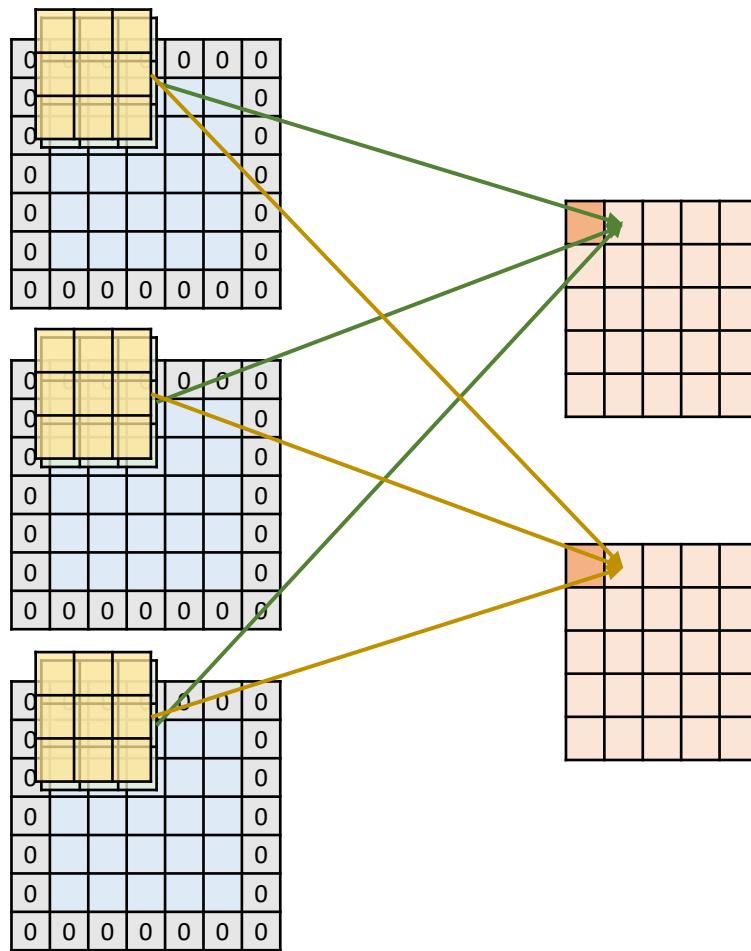
# 2D Convolution

- Input\_channel=3, output\_channel=2, padding=1, stride=1



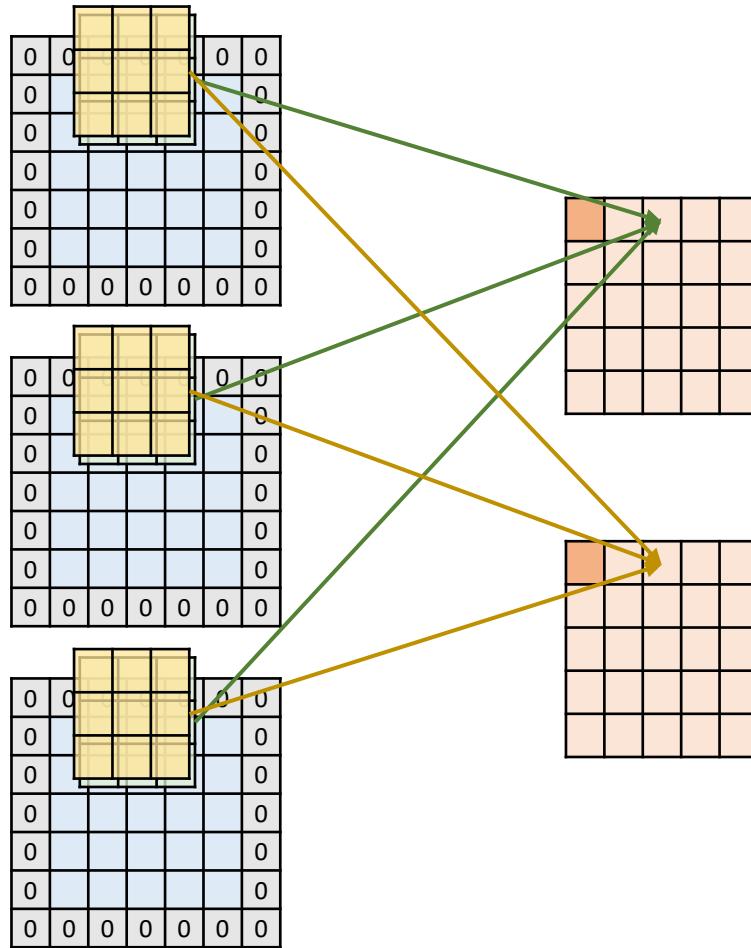
# 2D Convolution

- Input\_channel=3, output\_channel=2, padding=1, stride=1



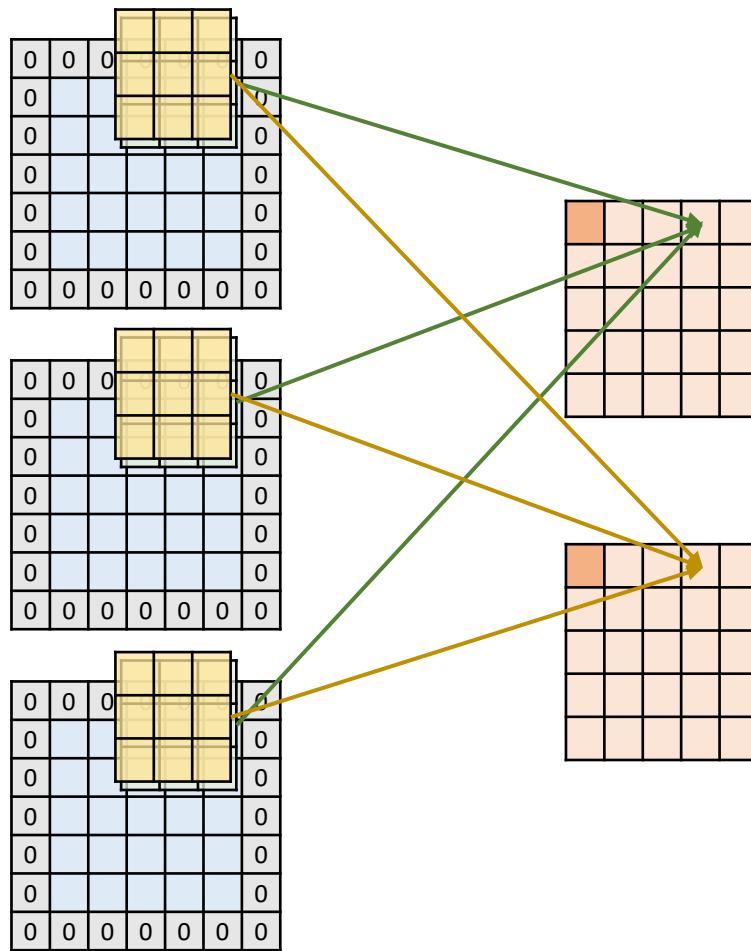
# 2D Convolution

- Input\_channel=3, output\_channel=2, padding=1, stride=1



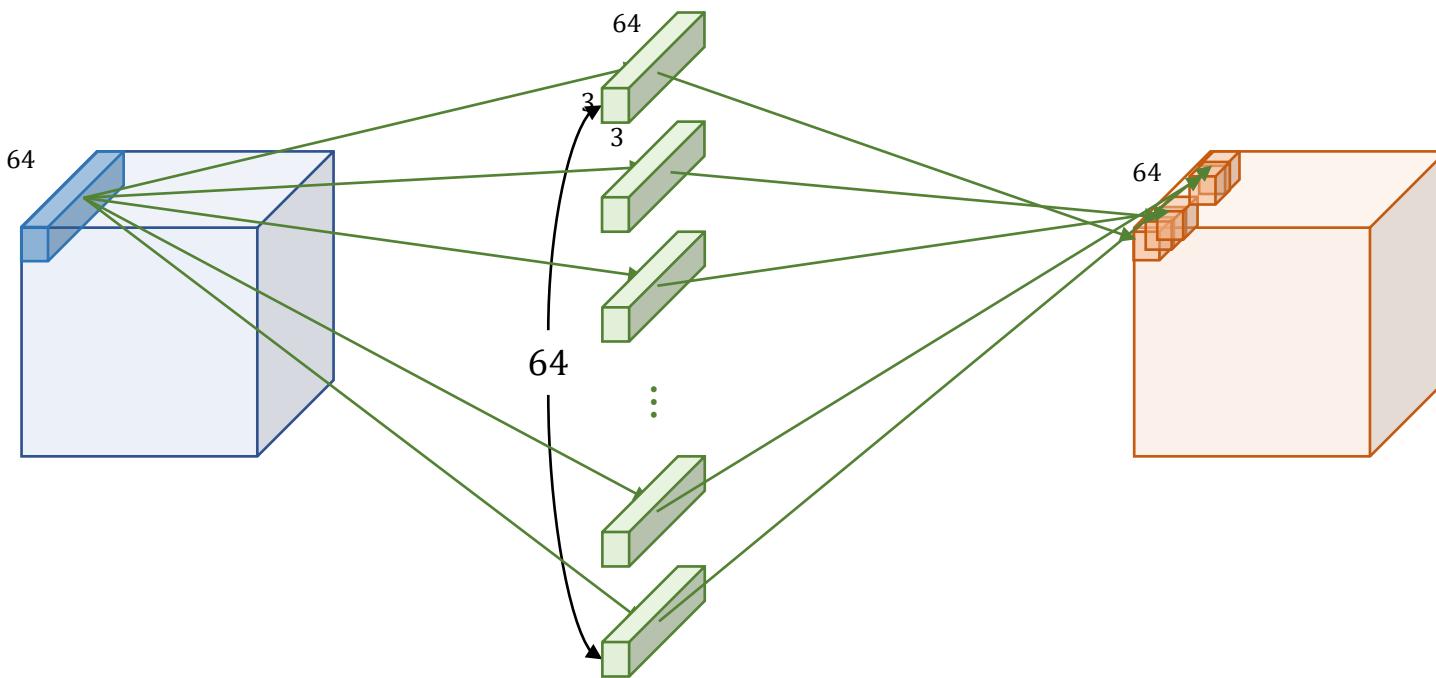
# 2D Convolution

- Input\_channel=3, output\_channel=2, padding=1, stride=1



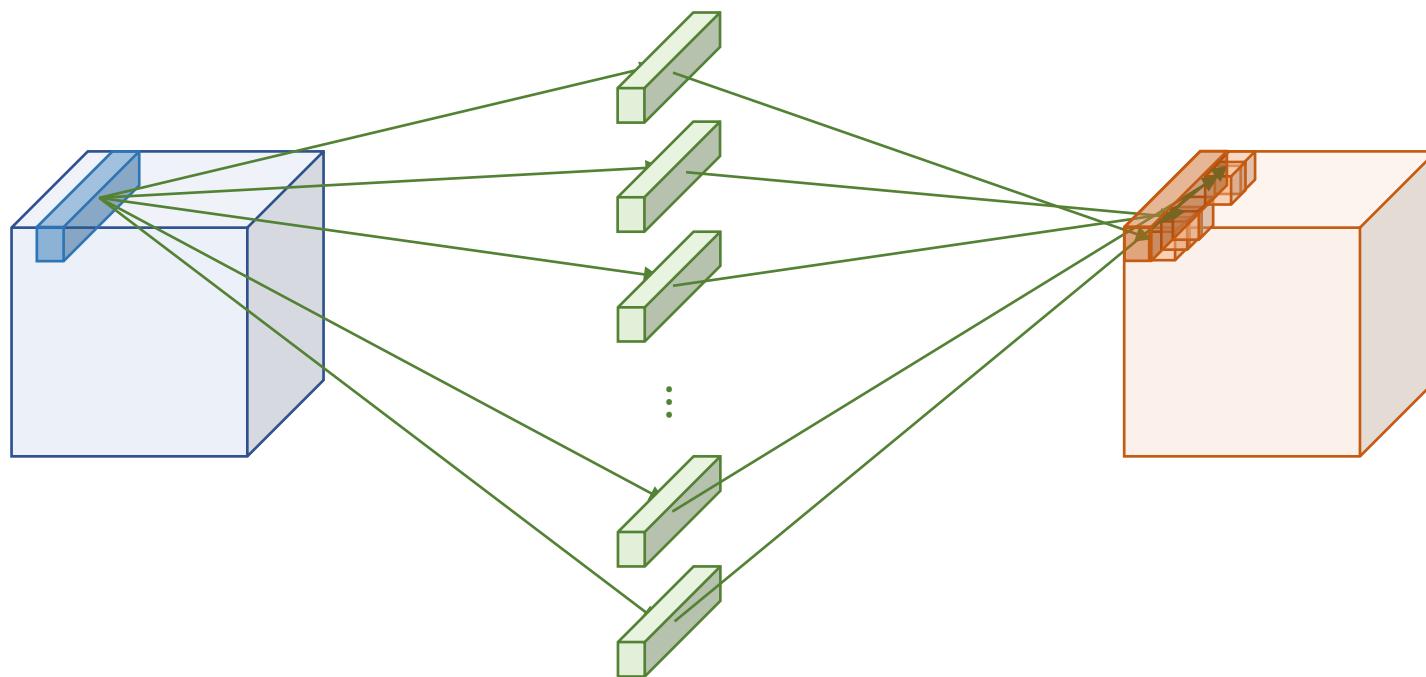
# 2D Convolution

- Input\_channel=64, output\_channel=64, kernel\_size=3, padding=1, stride=1



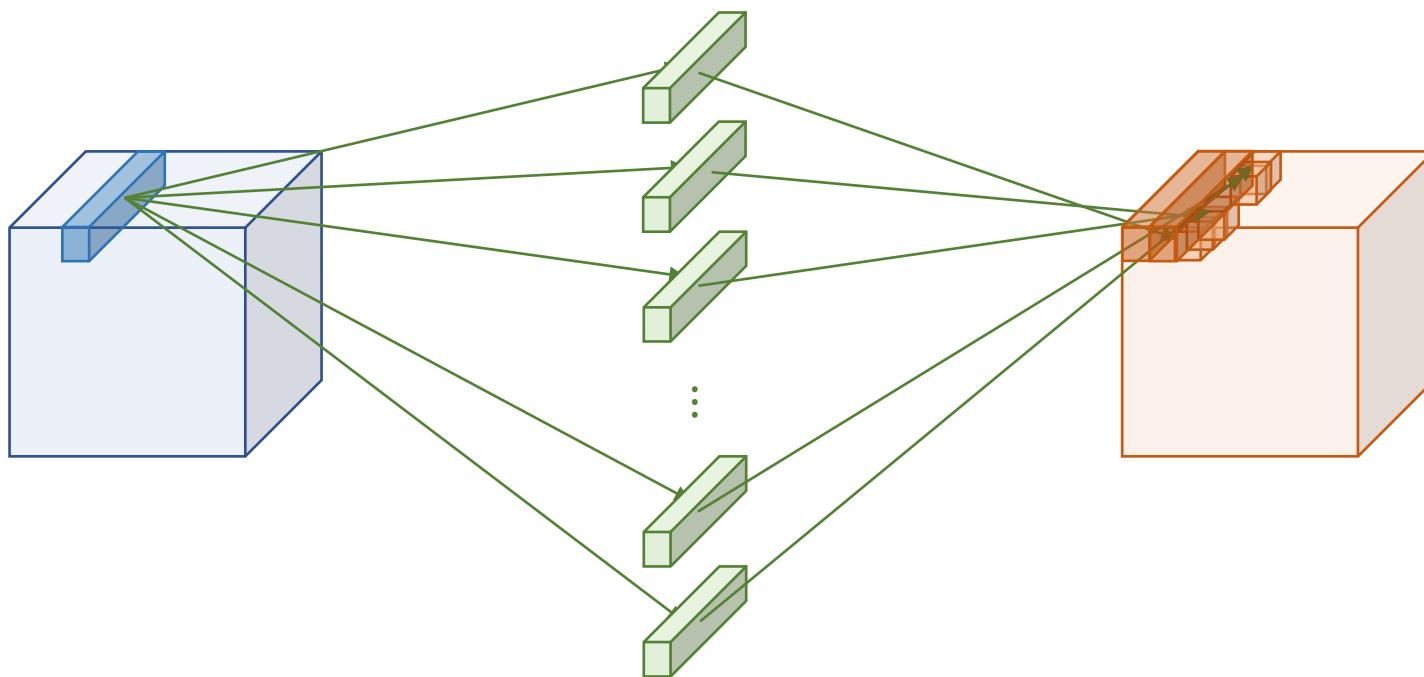
# 2D Convolution

- Input\_channel=64, output\_channel=64, kernel\_size=3, padding=1, stride=1



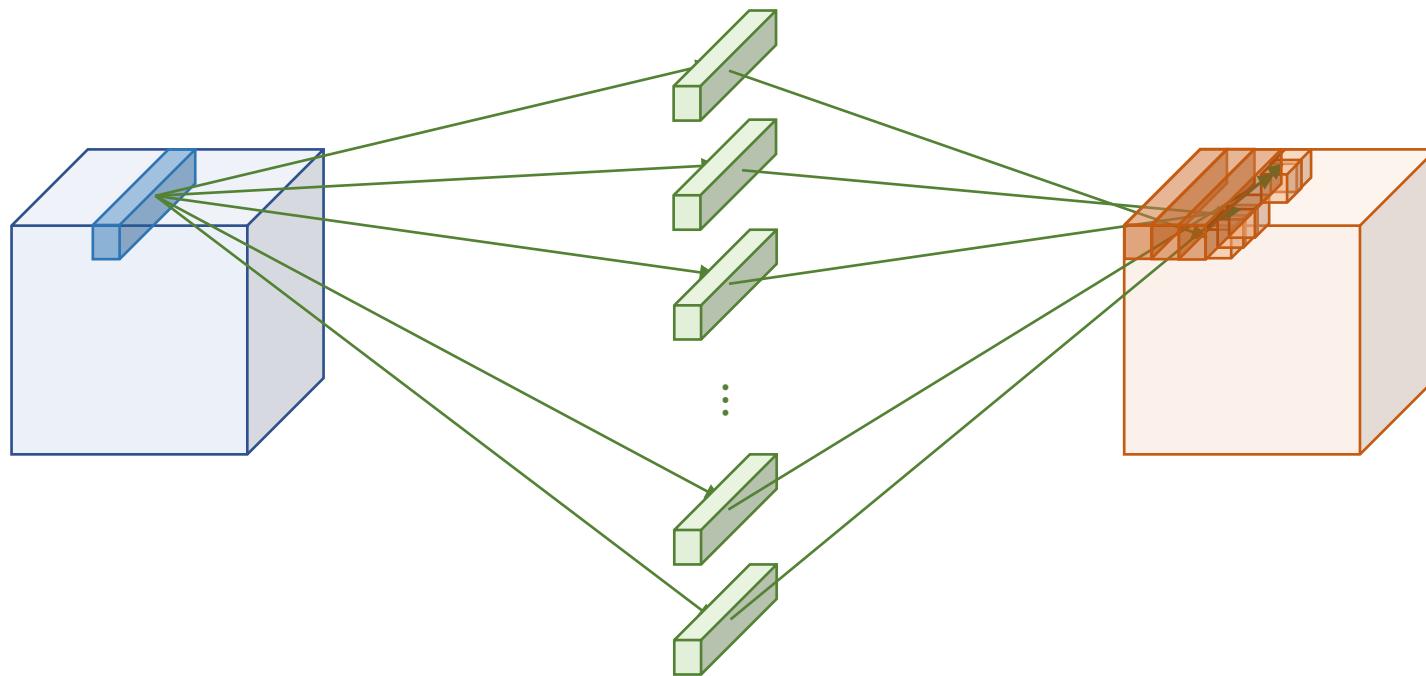
# 2D Convolution

- 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

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

3		

# Max Pooling

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

3	3	

# Max Pooling

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

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

3	3	3

# 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)`

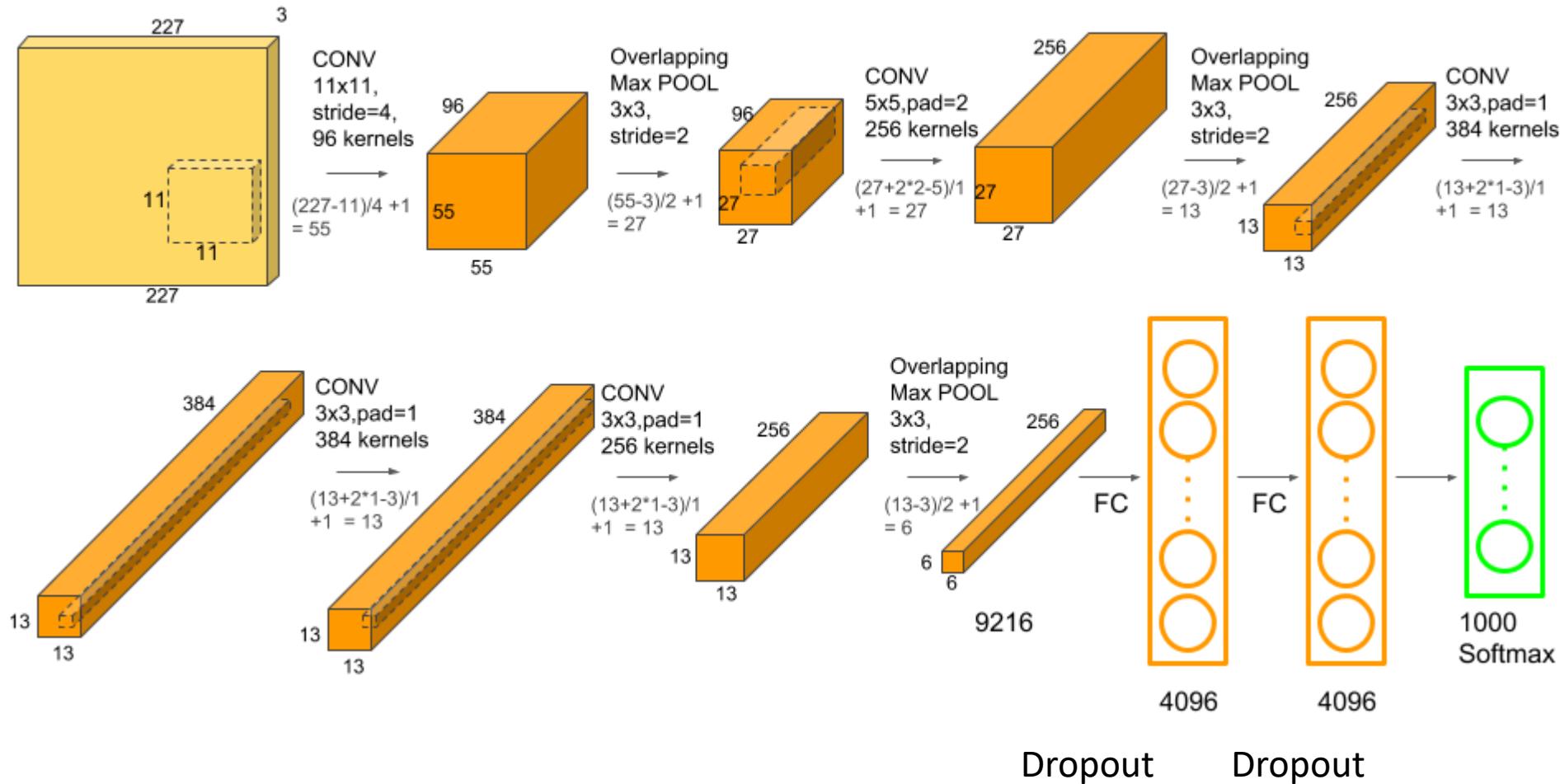
[SOURCE]

---

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

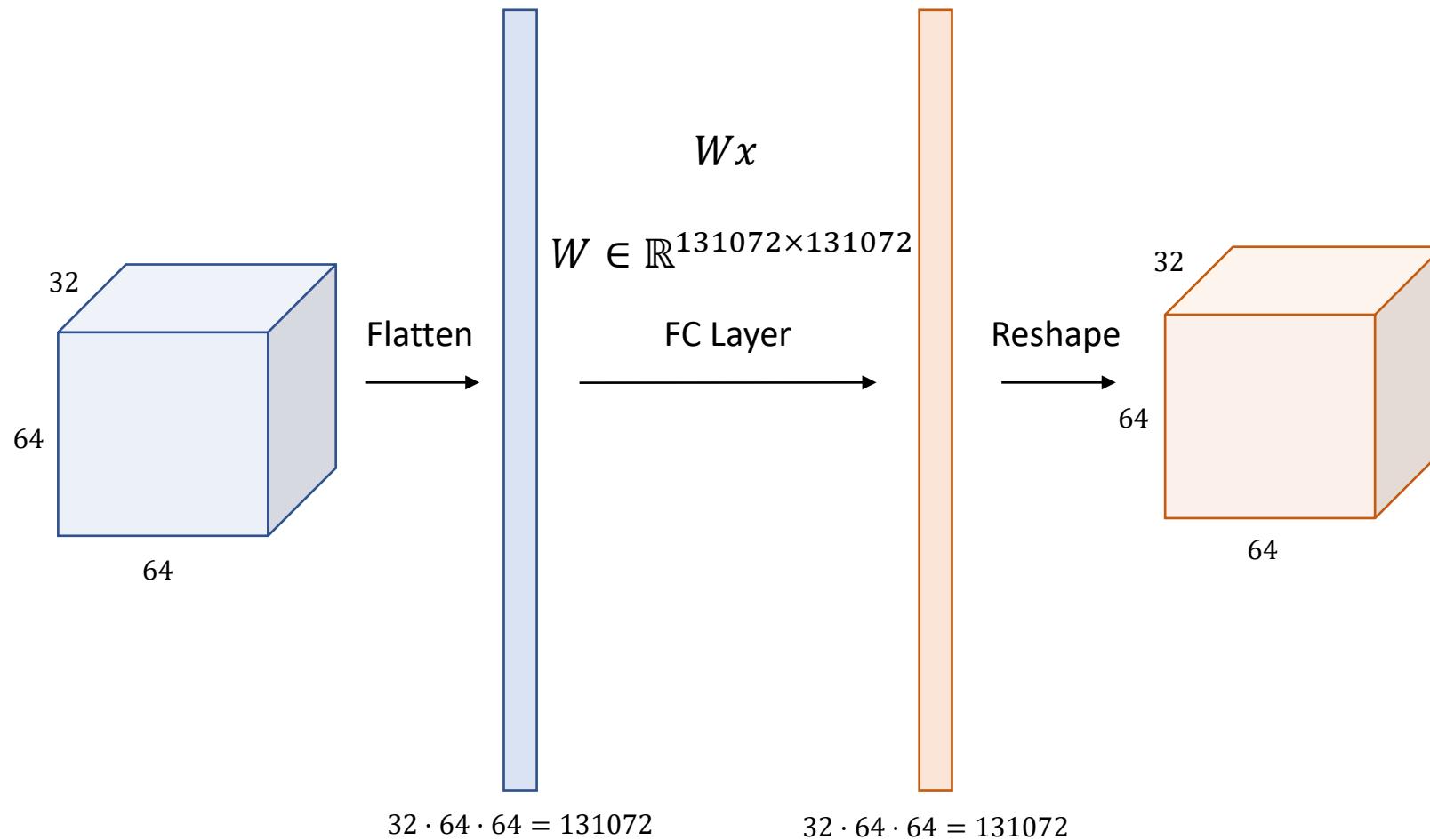
[SOURCE]

# AlexNet



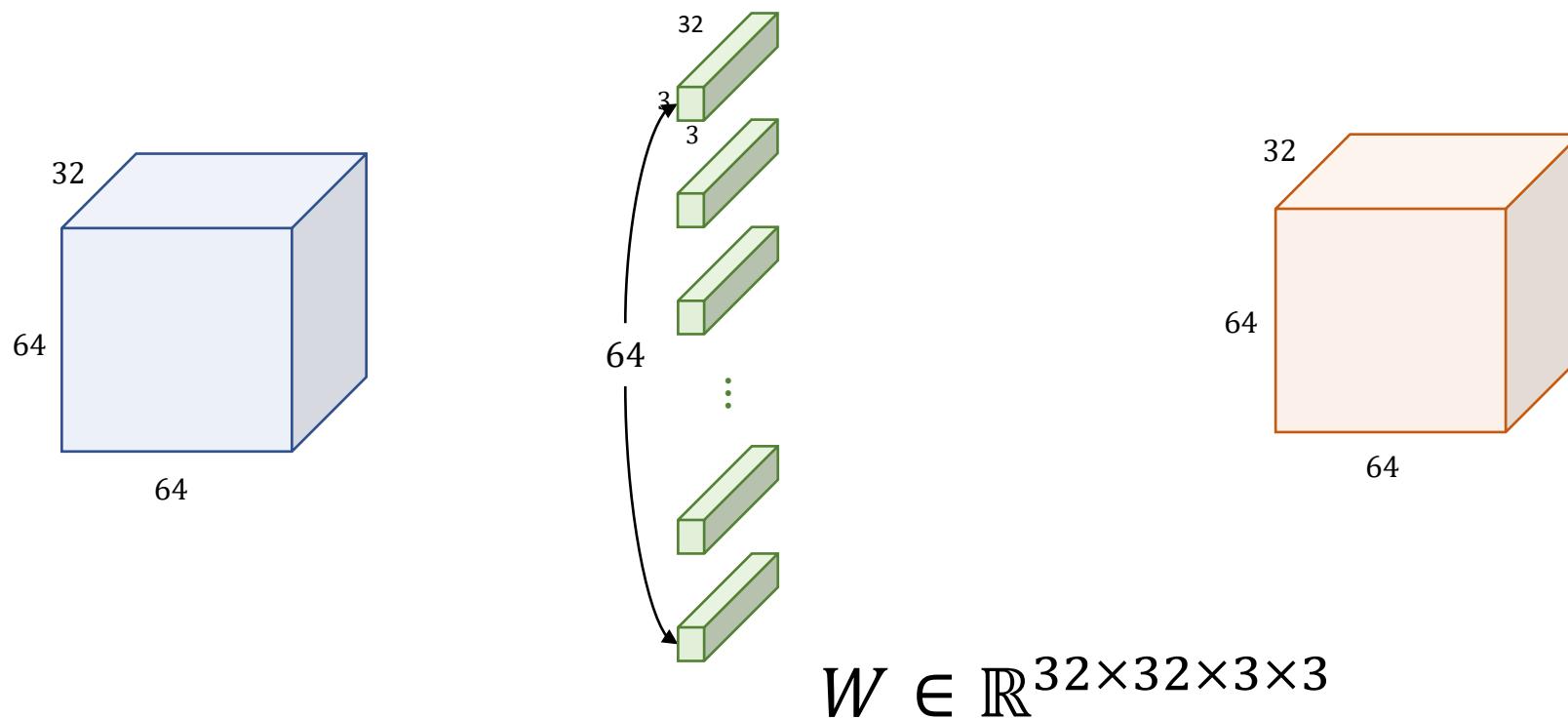
# Fully Connected Layer vs Convolutional Layer

- Translation equivariance and parameter sharing



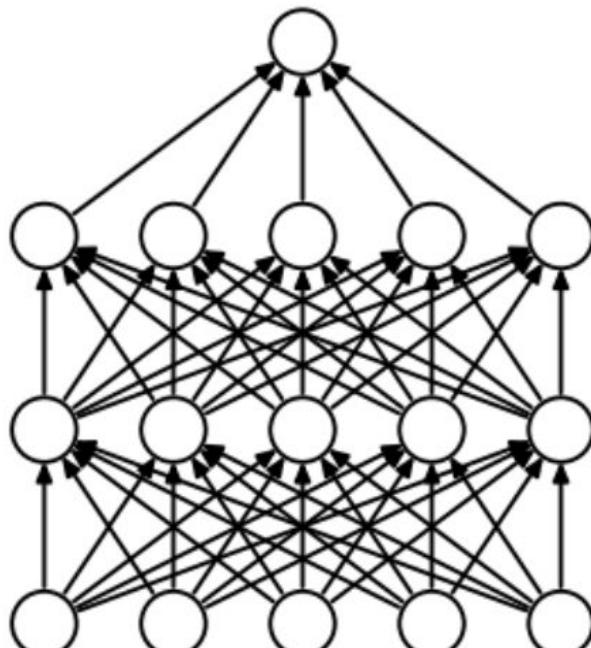
# Fully Connected Layer vs Convolutional Layer

- Translation equivariance and parameter sharing

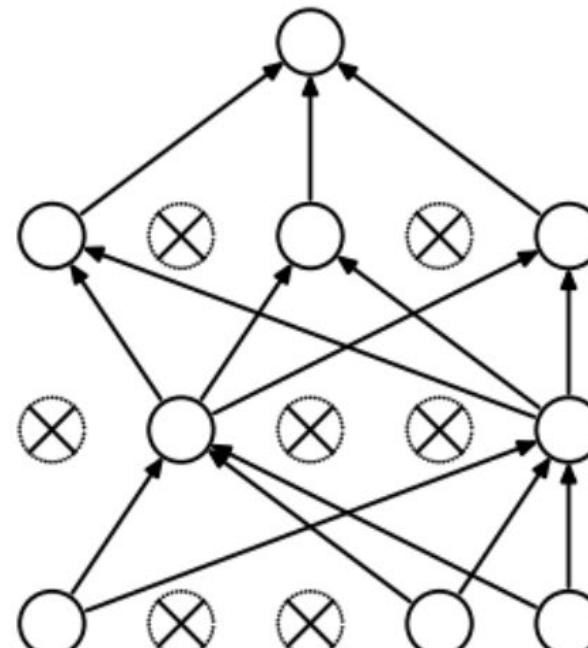


# Dropout

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



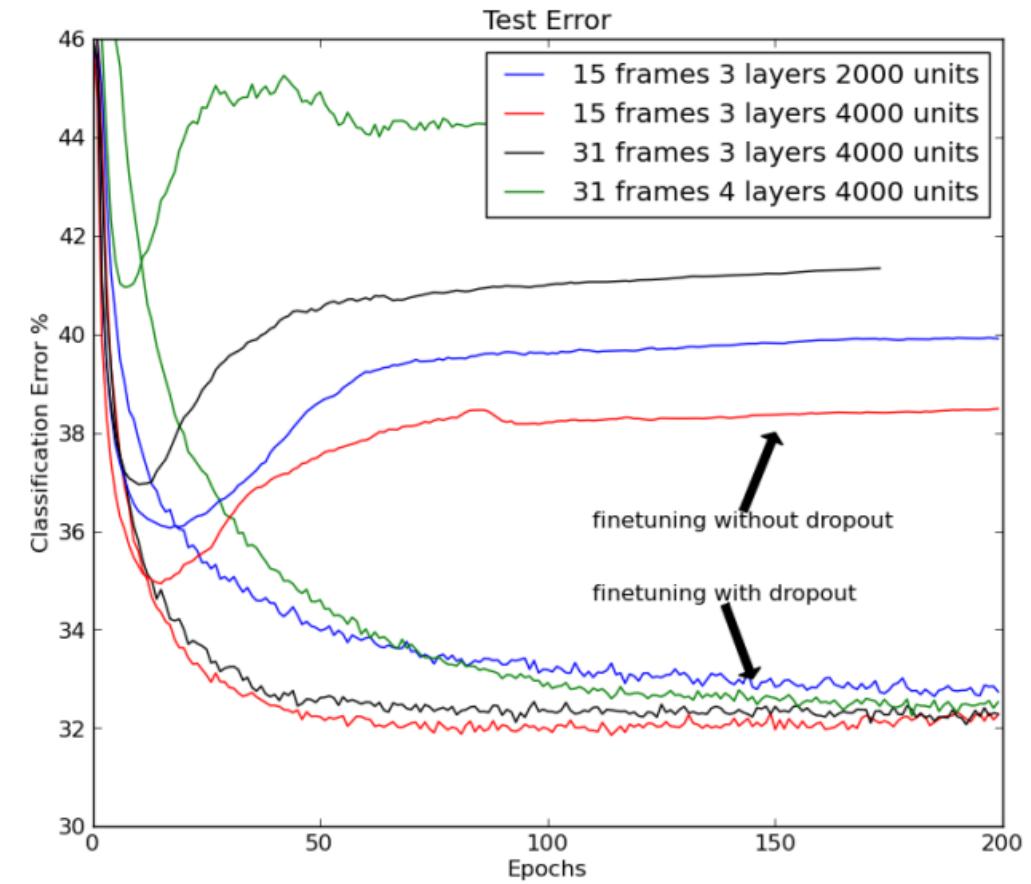
(a) Standard Neural Net



(b) After applying dropout.

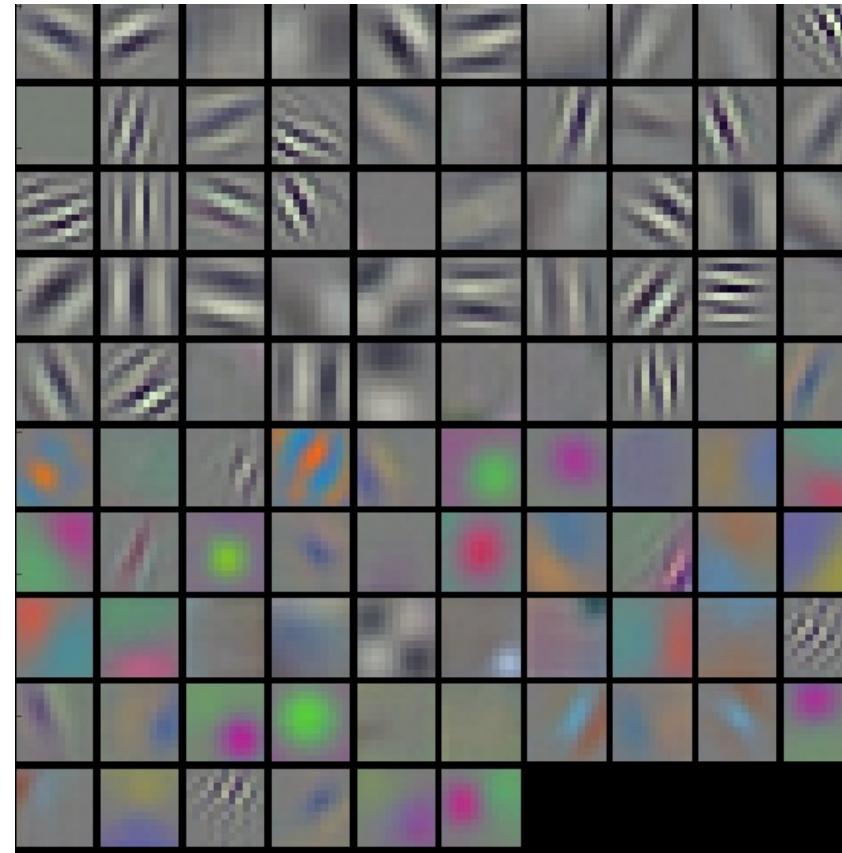
# Dropout

- 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

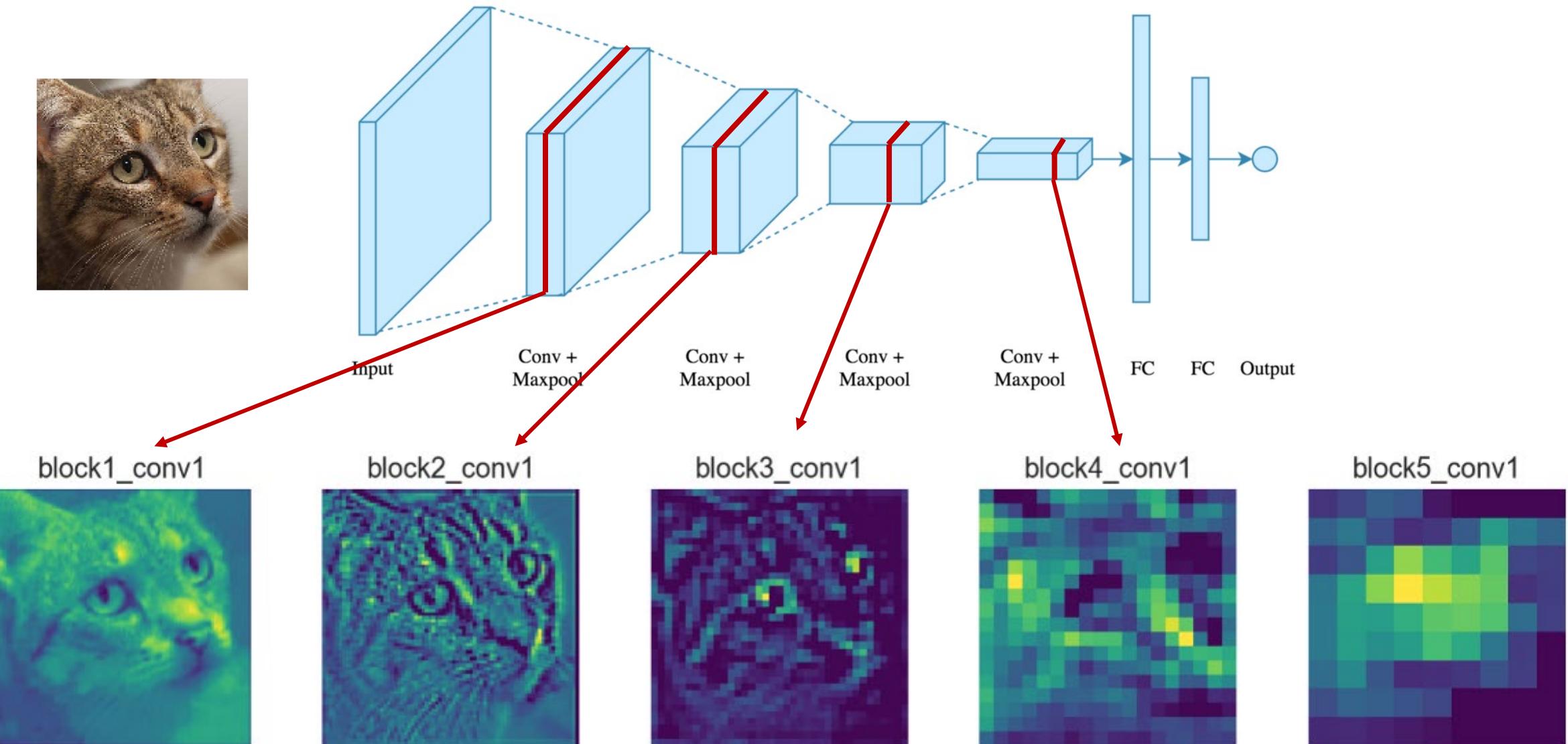


# Visualization of Learned Filter

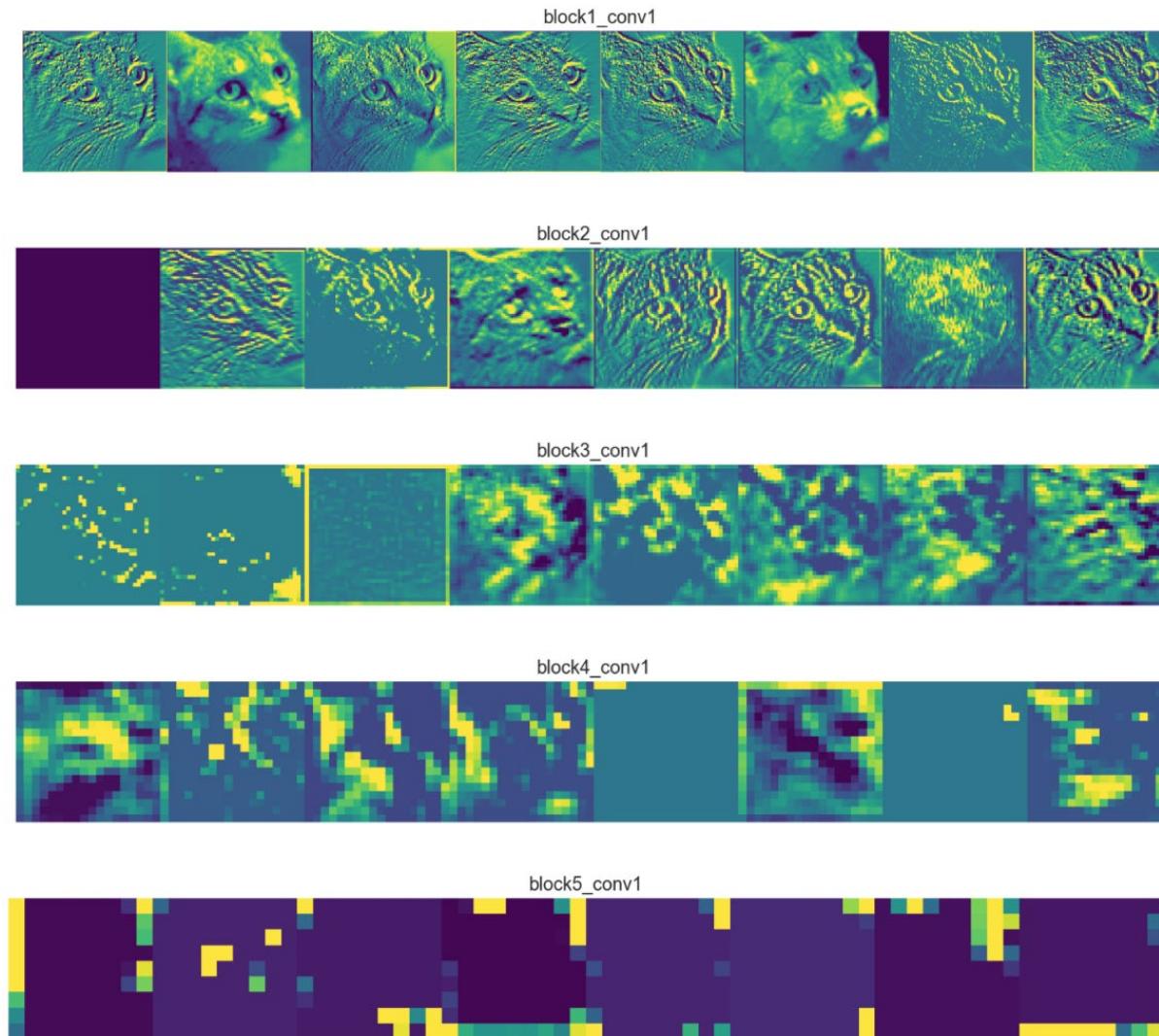
- First layer conv filters



# Visualization of Learned Feature Maps



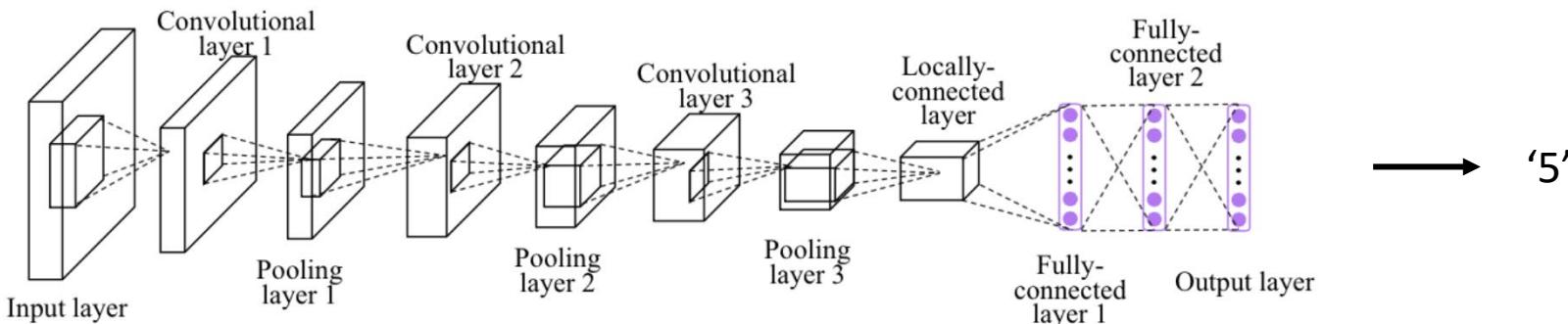
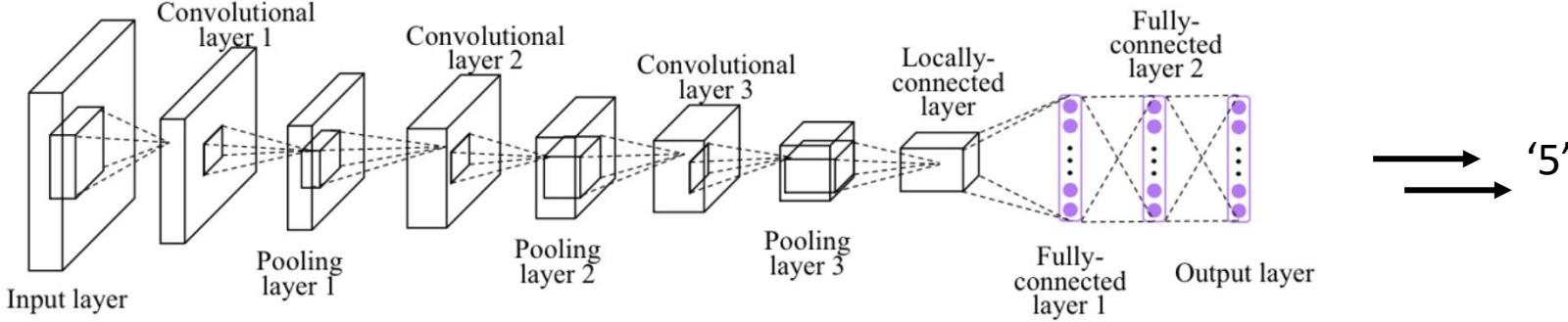
# Visualization of Learned Feature Maps



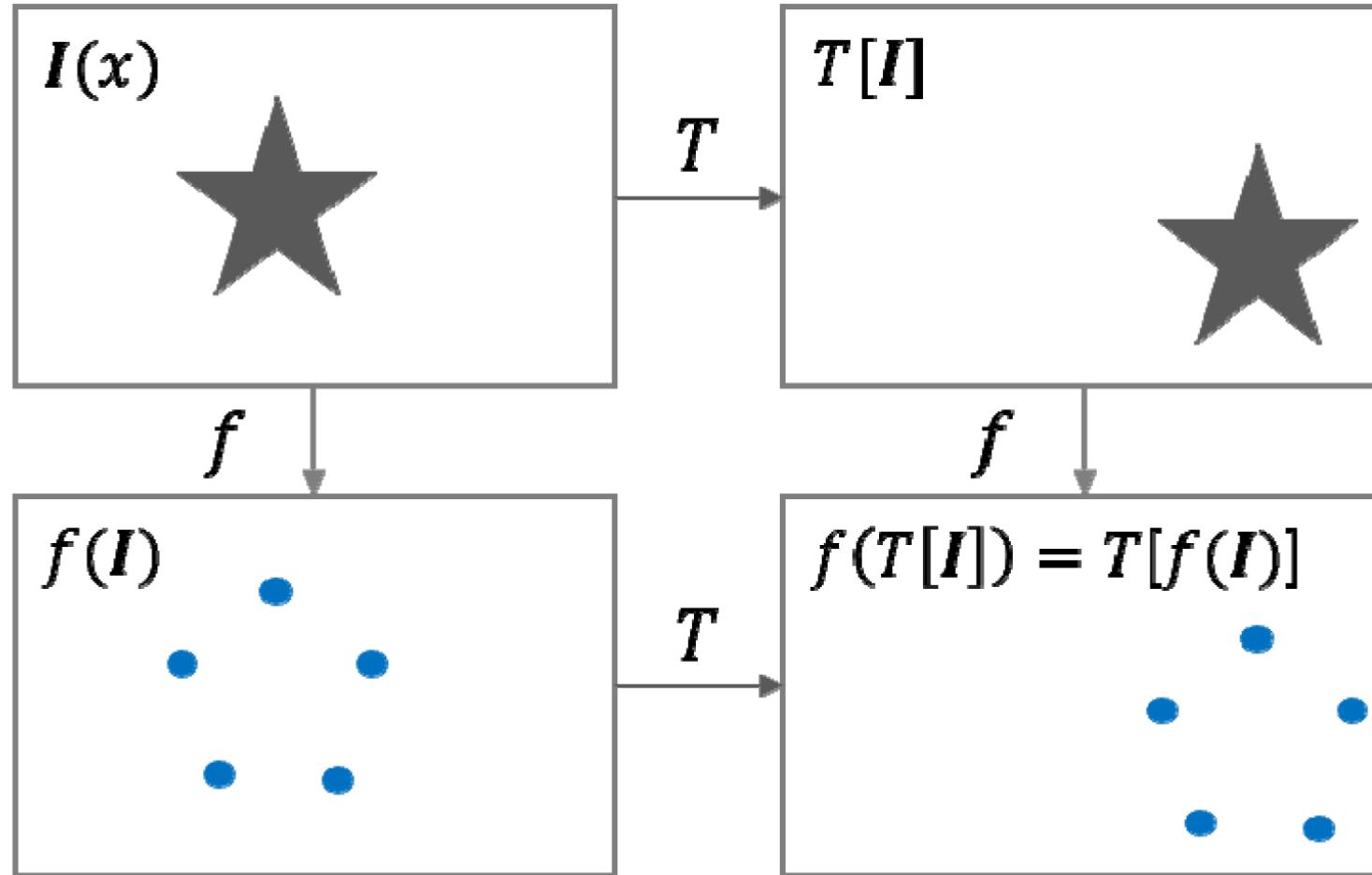
# Translation Equivariance vs Invariance

- 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 \(poloclub.github.io\)](https://poloclub.github.io/CNNExplainer)

# Convolutions Behind the Scene

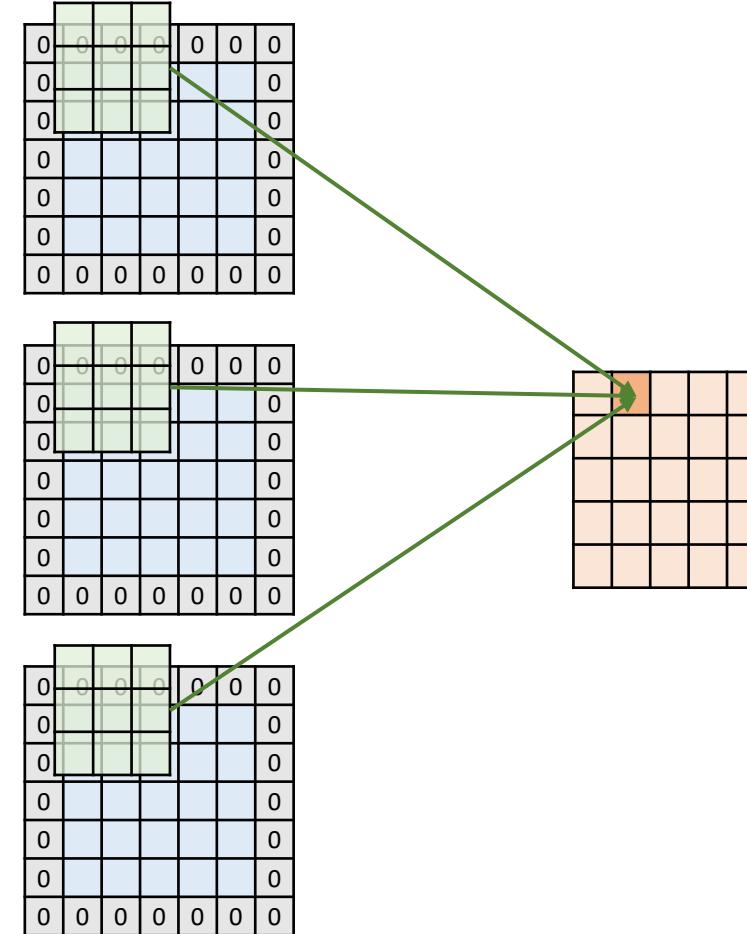
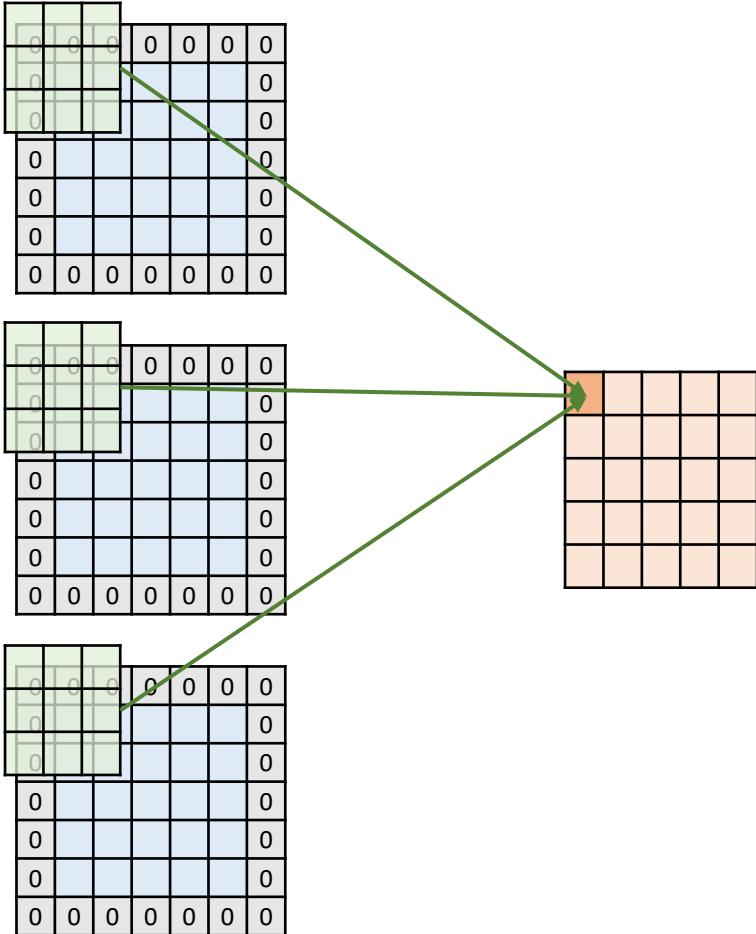
# Sequential Implementation (Conv2D)

- 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))
```

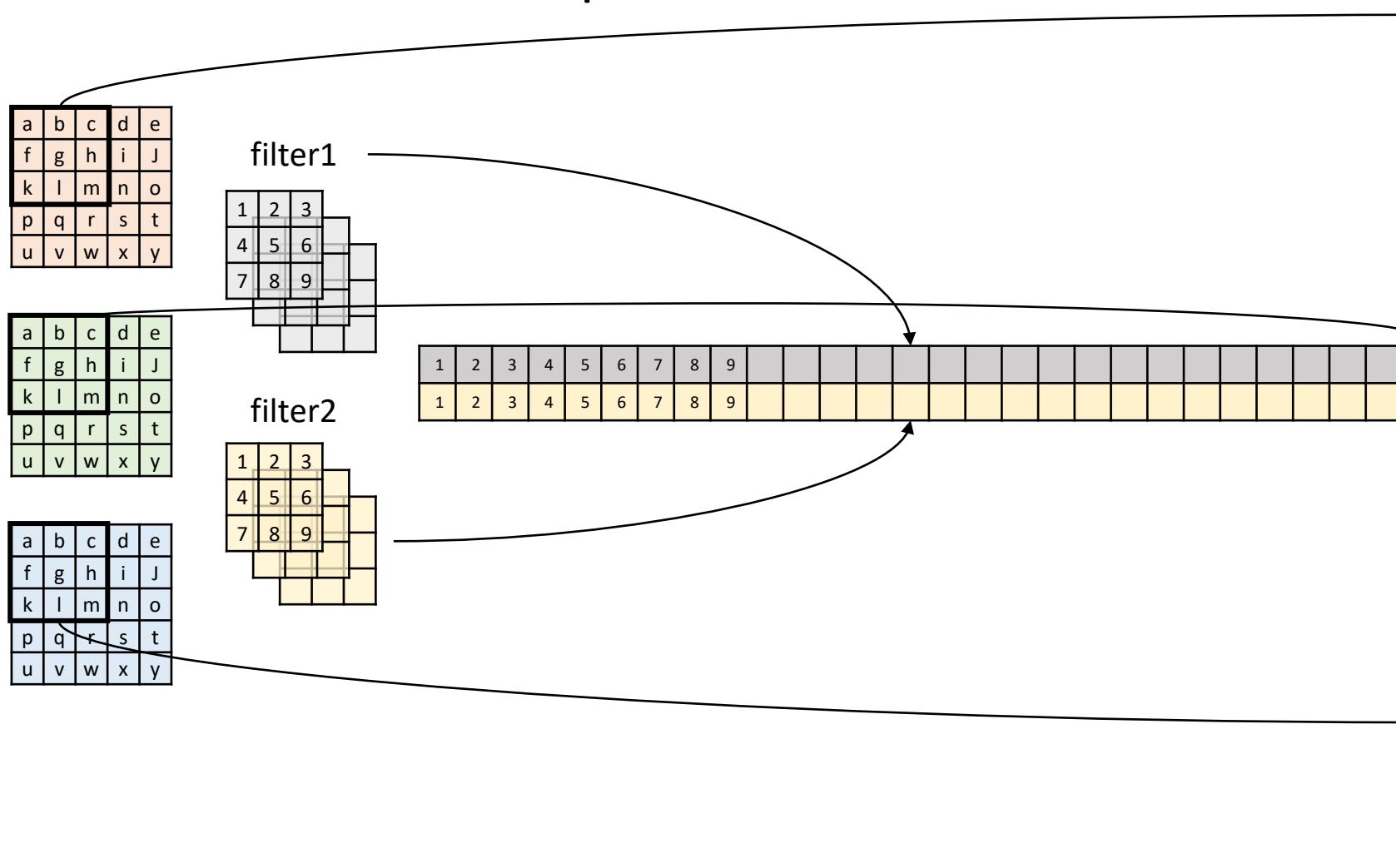
# Parallel Implementation

- Convolutions are parallelizable



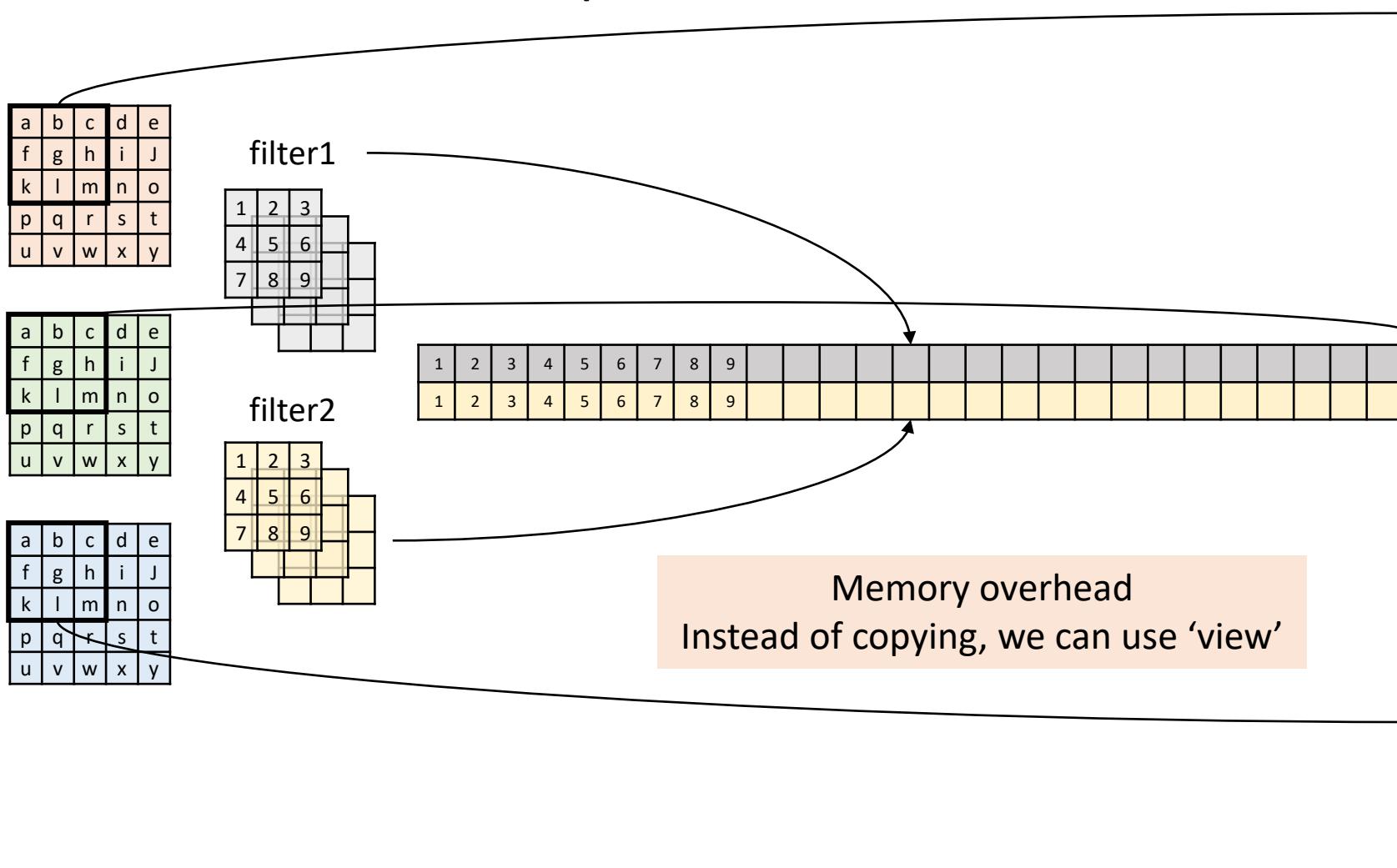
# Parallel Implementation

- Im2Col + Matrix Multiplication



# Parallel Implementation

- Im2Col + Matrix Multiplication



# Further Optimizations

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

# Backprop in Convolutional Layers

# Transposed Convolution

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

1	0	1
1	1	0
0	1	2
2	1	2

1	0	1		
1	1	0		
0	1	2		

# Transposed Convolution

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

1	0	1
1	1	0
0	1	2
2	1	2

1	0	1		
1	1	0		
0	1	2		

+

	2	0	2	
	2	2	0	
	0	2	4	

=

1	2	1	2	
1	3	2	0	
0	1	4	4	

# Transposed Convolution

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

The diagram illustrates a transposed convolution operation. It consists of three input matrices (green, orange, blue) and a kernel (orange), followed by an addition step and an equals sign.

**Input Matrices:**

- Green matrix:

1	0	1
1	1	1
1	0	1
2	1	2
- Orange matrix:

1	2	1	2	
1	3	2	0	
0	1	4	4	
- Blue matrix:

1		

**Kernel:** An orange 3x3 matrix:

1	0	1
1	1	0
0	1	2

**Addition:** The result of the multiplication is added to the second input matrix (orange).  
The result of the addition is then added to the third input matrix (blue).

**Output:** The final result is:

1	2	2	2	1
1	3	3	1	0
0	1	4	5	2

# Transposed Convolution

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

1	0	1	1
1	1	0	1
0	1	2	2

1	2	2	2	1
1	3	3	1	0
0	1	4	5	2

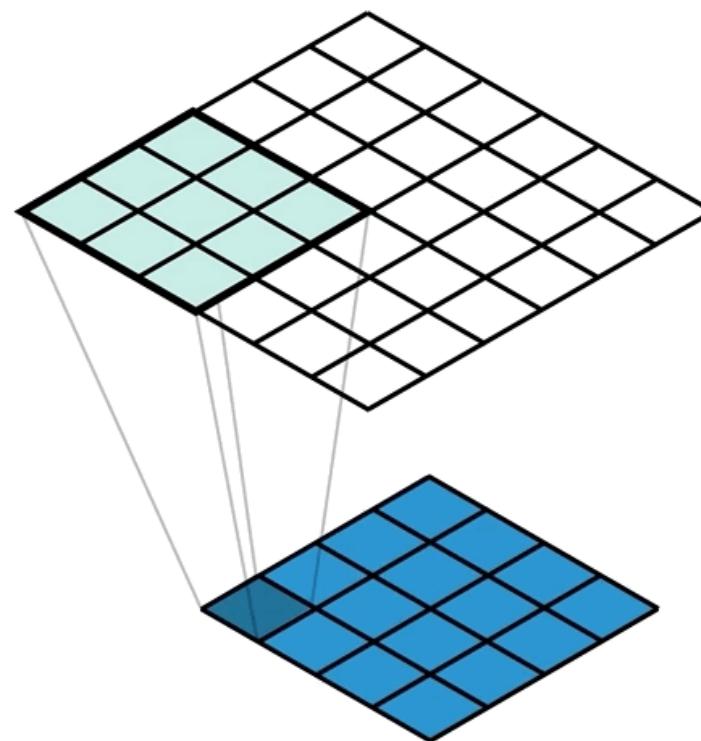
+

1	0	1		
1	1	0		
0	1	2		

=

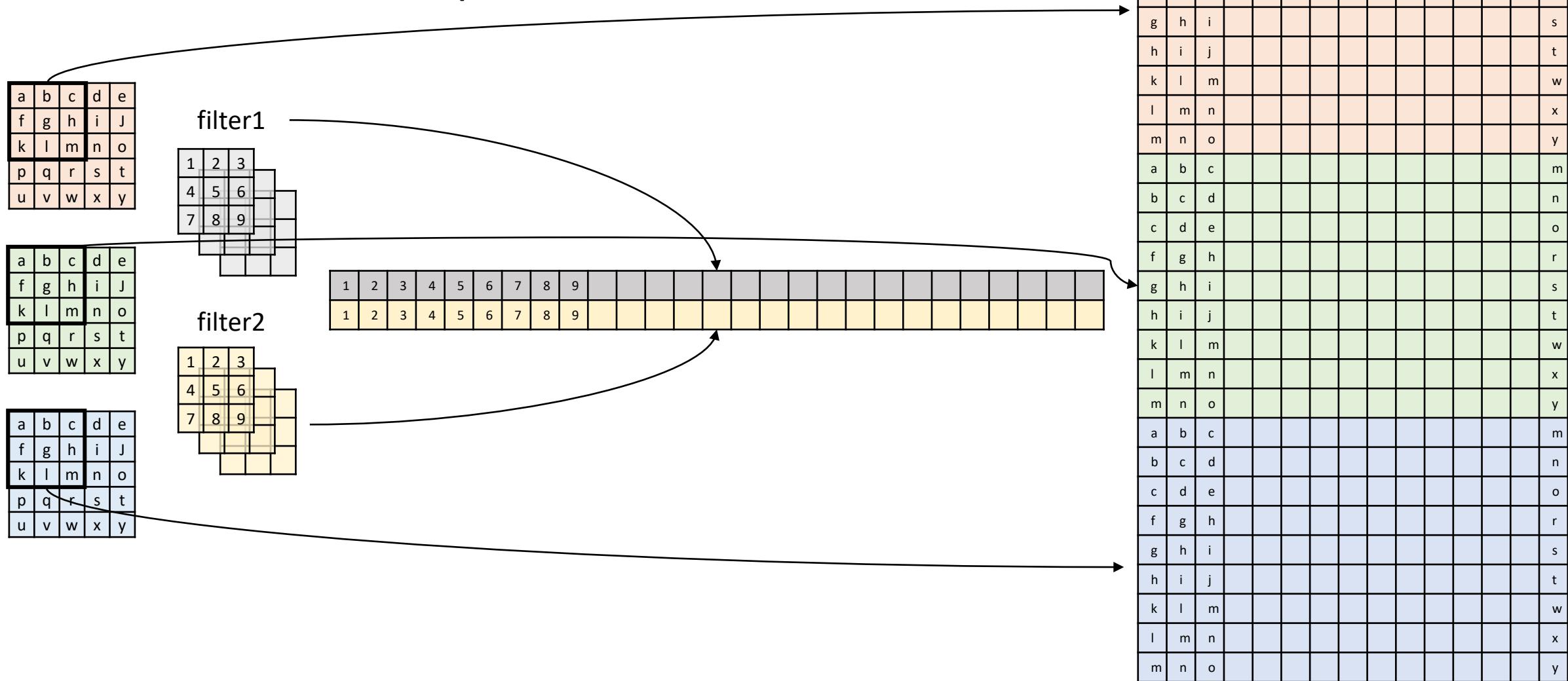
1	2	2	2	1
2	3	4	1	0
1	2	4	5	2
0	1	2		

# Transposed Convolution



# Convolution as Matrix Multiplication

- Im2Col + Matrix Multiplication



# Convolution as Matrix Multiplication

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

$$Y = WX$$

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

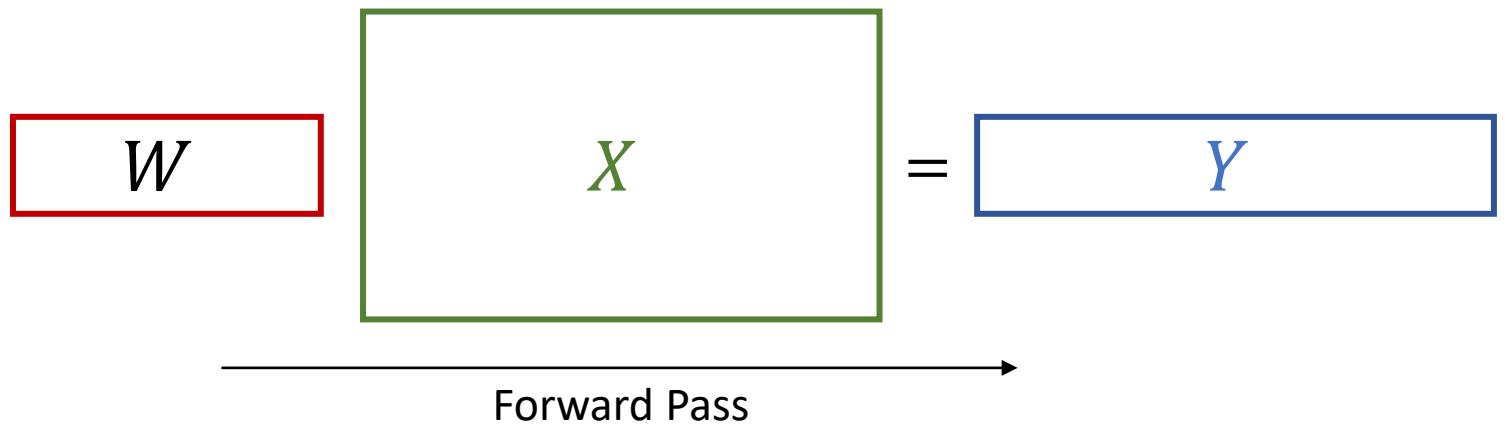
$$\frac{dz}{dX} = \frac{dz}{dY} \frac{dY}{dX} = W^\top \frac{dz}{dY}$$

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

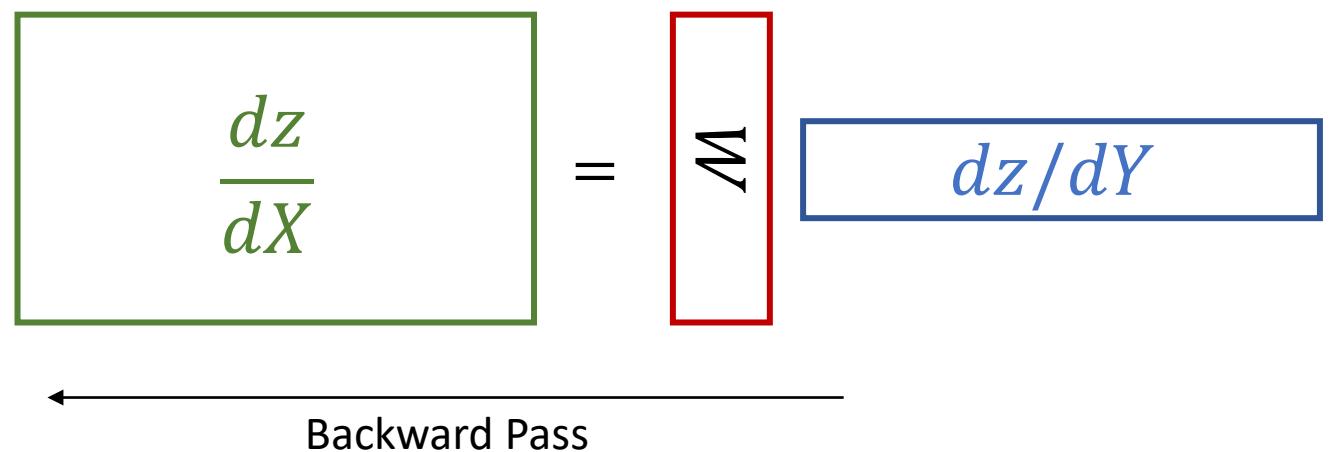
# Forward and Backward

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

$$Y = W X$$



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



# Backward Pass

- Matrix multiplication view

$$\frac{dz}{dX} \quad \leftarrow \quad \frac{dz}{d\hat{X}} = W^\top \quad \frac{dz}{dY}$$

The diagram illustrates the matrix multiplication view of the backward pass. It shows the computation of the gradient  $\frac{dz}{dX}$  as the product of three matrices:  $\frac{dz}{d\hat{X}}$ ,  $W^\top$ , and  $\frac{dz}{dY}$ .

The matrices are defined as follows:

- $\frac{dz}{d\hat{X}}$  is a 4x3 matrix where each column is labeled  $w_i z'_j$  for  $i \in \{1, 2, 3, 4\}$  and  $j \in \{1, 2, 3\}$ . The columns are:  $w_1 z'_1, w_1 z'_2, w_1 z'_3$ ;  $w_2 z'_1, w_2 z'_2, w_2 z'_3$ ;  $w_3 z'_1, w_3 z'_2, w_3 z'_3$ ; and  $w_4 z'_1, w_4 z'_2, w_4 z'_3$ .
- $W^\top$  is a 4x1 vector where each row is labeled  $w_i$  for  $i \in \{1, 2, 3, 4\}$ . The rows are:  $w_1$ ;  $w_2$ ;  $w_3$ ; and  $w_4$ .
- $\frac{dz}{dY}$  is a 4x1 vector where each element is labeled  $z'_i$  for  $i \in \{1, 2, 3, 4\}$ . The elements are:  $z'_1$ ;  $z'_2$ ;  $z'_3$ ; and  $z'_4$ .

A horizontal arrow points from  $\frac{dz}{d\hat{X}}$  to  $\frac{dz}{dX}$ , indicating the result of the multiplication. The equation  $= W^\top$  is placed between  $\frac{dz}{d\hat{X}}$  and  $\frac{dz}{dY}$ , and another arrow points from  $\frac{dz}{dY}$  to  $\frac{dz}{dX}$ .

# Backward Pass

- Matrix multiplication view

$$\frac{dz}{dX} \quad \leftarrow \quad \frac{dz}{d\hat{X}} = W^\top \quad \frac{dz}{dY}$$

Diagram illustrating the matrix multiplication view of the backward pass. The diagram shows the computation of gradients for a layer  $X$  given gradients for the output  $Y$ .

The input gradient  $\frac{dz}{dY}$  is represented by a vector of four green boxes:  $[z'_1 \ z'_2 \ z'_3 \ z'_4]$ . This vector is multiplied by the transpose of the weight matrix  $W^\top$ , which is represented by a vertical stack of four matrices:  $[w_1 \ w_2 \ w_3 \ w_4]$ . The result is the output gradient  $\frac{dz}{dX}$ , represented by a 3x3 grid of orange boxes:

$w_1 z'_1$	$w_2 z'_1$	
$w_3 z'_1$	$w_4 z'_1$	

The intermediate gradient  $\frac{dz}{d\hat{X}}$  is represented by a 4x4 grid of orange boxes:

$w_1 z'_1$	$w_1 z'_2$	$w_1 z'_3$	$w_1 z'_4$
$w_2 z'_1$	$w_2 z'_2$	$w_2 z'_3$	$w_2 z'_4$
$w_3 z'_1$	$w_3 z'_2$	$w_3 z'_3$	$w_3 z'_4$
$w_4 z'_1$	$w_4 z'_2$	$w_4 z'_3$	$w_4 z'_4$

A red double-line box highlights the first column of this matrix, corresponding to the first column of the  $\frac{dz}{dY}$  vector.

# Backward Pass

- Matrix multiplication view

$$\frac{dz}{dX} \quad \leftarrow \quad \frac{dz}{d\hat{X}} = W^\top \quad \frac{dz}{dY}$$

Diagram illustrating the matrix multiplication view of the backward pass. The diagram shows the computation of gradients  $\frac{dz}{dX}$  and  $\frac{dz}{dY}$  using a weight matrix  $W$  and its transpose  $W^\top$ .

The matrices involved are:

- $\frac{dz}{dX}$ : A 3x3 matrix where the first row contains  $w_1 z'_1$ ,  $w_2 z'_1 + w_1 z'_2$ , and  $w_2 z'_2$ . The second row contains  $w_3 z'_1$ ,  $w_4 z'_1 + w_3 z'_2$ , and  $w_4 z'_2$ . The third row is empty.
- $\frac{dz}{d\hat{X}}$ : A 4x4 matrix where each row  $i$  contains the elements  $w_i z'_1, w_i z'_2, w_i z'_3, w_i z'_4$ . The first three rows are highlighted with red boxes.
- $W^\top$ : A vertical vector containing the columns of  $W$  transposed, labeled  $w_1, w_2, w_3, w_4$ .
- $\frac{dz}{dY}$ : A horizontal vector containing the elements  $z'_1, z'_2, z'_3, z'_4$ , highlighted with green boxes.

A horizontal arrow points from  $\frac{dz}{d\hat{X}}$  to  $\frac{dz}{dX}$ , indicating the computation of the gradient  $\frac{dz}{dX}$  using the transpose of the weight matrix  $W^\top$  and the gradient  $\frac{dz}{dY}$ .

# Backward Pass

- Matrix multiplication view

$$\frac{dz}{dX} \quad \leftarrow \quad \frac{dz}{d\hat{X}} = W^\top = \frac{dz}{dY}$$

The diagram illustrates the backward pass through a neural network layer, showing the computation of gradients using matrix multiplication.

The left side shows the gradient matrix  $\frac{dz}{dX}$  (dimensions 3x3), which is the transpose of the weight matrix  $W$ . The entries are calculated as follows:

$w_1 z'_1$	$w_2 z'_1 + w_1 z'_2$	$w_2 z'_2$
$w_3 z'_1 + w_1 z'_3$	$w_4 z'_1 + w_3 z'_2 + w_2 z'_3$	$w_4 z'_2$
$w_3 z'_3$	$w_4 z'_3$	

The right side shows the gradient matrix  $\frac{dz}{d\hat{X}}$  (dimensions 4x4), where the last two columns are highlighted with a red border. This matrix is equal to the transpose of the weight matrix  $W^\top$ .

The weight matrix  $W$  (dimensions 4x3) is shown below:

$w_1$		
$w_2$		
$w_3$		
$w_4$		

The output gradient matrix  $\frac{dz}{dY}$  (dimensions 4x1) is shown on the far right, with its entries highlighted in green:

$z'_1$
$z'_2$
$z'_3$
$z'_4$

# Backward Pass

- Matrix multiplication view

$$\frac{dz}{dX} \quad \leftarrow \quad \frac{dz}{d\hat{X}} = W^\top = \begin{matrix} \frac{dz}{dY} \\ z'_1 \quad z'_2 \quad z'_3 \quad z'_4 \end{matrix}$$

The diagram illustrates the backward pass through a neural network layer. It shows the computation of gradients  $\frac{dz}{dX}$  from gradients  $\frac{dz}{d\hat{X}}$  using the transpose of the weight matrix  $W^\top$ .

The matrices involved are:

- $\frac{dz}{dX}$  (Input Gradients):

$w_1 z'_1$	$w_2 z'_1 + w_1 z'_2$	$w_2 z'_2$
$w_3 z'_1 + w_1 z'_3$	$w_4 z'_1 + w_3 z'_2 + w_2 z'_3 + w_1 z'_4$	$w_4 z'_2 + w_2 z'_4$
$w_3 z'_3$	$w_4 z'_3 + w_3 z'_4$	$w_4 z'_4$
- $\frac{dz}{d\hat{X}}$  (Output Gradients):

$w_1 z'_1$	$w_1 z'_2$	$w_1 z'_3$	$w_1 z'_4$
$w_2 z'_1$	$w_2 z'_2$	$w_2 z'_3$	$w_2 z'_4$
$w_3 z'_1$	$w_3 z'_2$	$w_3 z'_3$	$w_3 z'_4$
$w_4 z'_1$	$w_4 z'_2$	$w_4 z'_3$	$w_4 z'_4$

With the last column ( $w_1 z'_4, w_2 z'_4, w_3 z'_4, w_4 z'_4$ ) highlighted by a red box.
- $W^\top$  (Transpose of Weight Matrix):

$w_1$
$w_2$
$w_3$
$w_4$
- $\frac{dz}{dY}$  (Final Gradients):

$z'_1$	$z'_2$	$z'_3$	$z'_4$
--------	--------	--------	--------

# Backward Pass

- Matrix multiplication view

$$\frac{dz}{dX} = \begin{matrix} & w \\ \begin{matrix} w_1 & w_2 \\ w_3 & w_4 \end{matrix} & *^T \begin{matrix} z'_1 & z'_2 \\ z'_3 & z'_4 \end{matrix} \end{matrix}$$

Transposed convolution

# Backward Pass

- Matrix multiplication view

$$\frac{dz}{dX} = \begin{matrix} w_1z'_1 & w_1z'_2 & \\ w_1z'_3 & w_1z'_4 & \\ \end{matrix} \quad \begin{matrix} w \\ w_1 & w_2 \\ w_3 & w_4 \end{matrix} *^T \begin{matrix} z'_1 & z'_2 \\ z'_3 & z'_4 \end{matrix}$$

Transposed convolution

# Backward Pass

- Matrix multiplication view

$$\frac{dz}{dX} = \begin{matrix} & \begin{matrix} w_1z'_1 & w_1z'_2 + w_2z'_1 & w_2z'_2 \\ w_1z'_3 & w_1z'_4 + w_2z'_3 & w_2z'_4 \\ \hline \end{matrix} \\ \begin{matrix} w_1 & w_2 \\ w_3 & w_4 \end{matrix} & *^T \begin{matrix} z'_1 & z'_2 \\ z'_3 & z'_4 \end{matrix} \end{matrix}$$

Transposed convolution

# Backward Pass

- Matrix multiplication view

$$\frac{dz}{dX} = \begin{matrix} & \frac{dz}{dY} \\ \begin{matrix} w_1z'_1 & w_1z'_2 + w_2z'_1 & w_2z'_2 \\ w_1z'_3 + w_3z'_1 & w_1z'_4 + w_2z'_3 + w_3z'_2 & w_2z'_4 \\ w_3z'_3 & w_3z'_4 & \end{matrix} & \begin{matrix} w \\ w_1 & w_2 \\ w_3 & w_4 \end{matrix} *^T \begin{matrix} z'_1 & z'_2 \\ z'_3 & z'_4 \end{matrix} \end{matrix}$$

Transposed convolution

# Backward Pass

- Matrix multiplication view

$$\frac{dz}{dX} = \begin{array}{|c|c|c|}\hline w_1z'_1 & w_2z'_1 + w_1z'_2 & w_2z'_2 \\ \hline w_3z'_1 + w_1z'_3 & w_4z'_1 + w_3z'_2 + w_2z'_3 + w_1z'_4 & w_4z'_2 + w_2z'_4 \\ \hline w_3z'_3 & w_4z'_3 + w_3z'_4 & w_4z'_4 \\ \hline\end{array} = \begin{array}{|c|c|c|}\hline w_1z'_1 & w_1z'_2 + w_2z'_1 & w_2z'_2 \\ \hline w_1z'_3 + w_3z'_1 & w_1z'_4 + w_2z'_3 + w_3z'_2 + w_4z'_1 & w_2z'_4 + w_4z'_2 \\ \hline w_3z'_3 & w_3z'_4 + w_4z'_3 & w_4z'_4 \\ \hline\end{array} = \begin{matrix} W \\ \hline \begin{matrix} w_1 & w_2 \\ \hline w_3 & w_4 \end{matrix} \end{matrix} *^T \begin{array}{|c|c|}\hline z'_1 & z'_2 \\ \hline z'_3 & z'_4 \\ \hline\end{array} \frac{dz}{dY}$$

Transposed convolution