

# CONVOLUTIONAL NEURAL NETWORKS: DISCRETE CONVOLUTIONS



Convolution  
operations first  
published by  
D'Alembert in 1754

Discrete convolutions  
are matrix operations  
that can be used to  
apply **filters** to a  
matrix or array

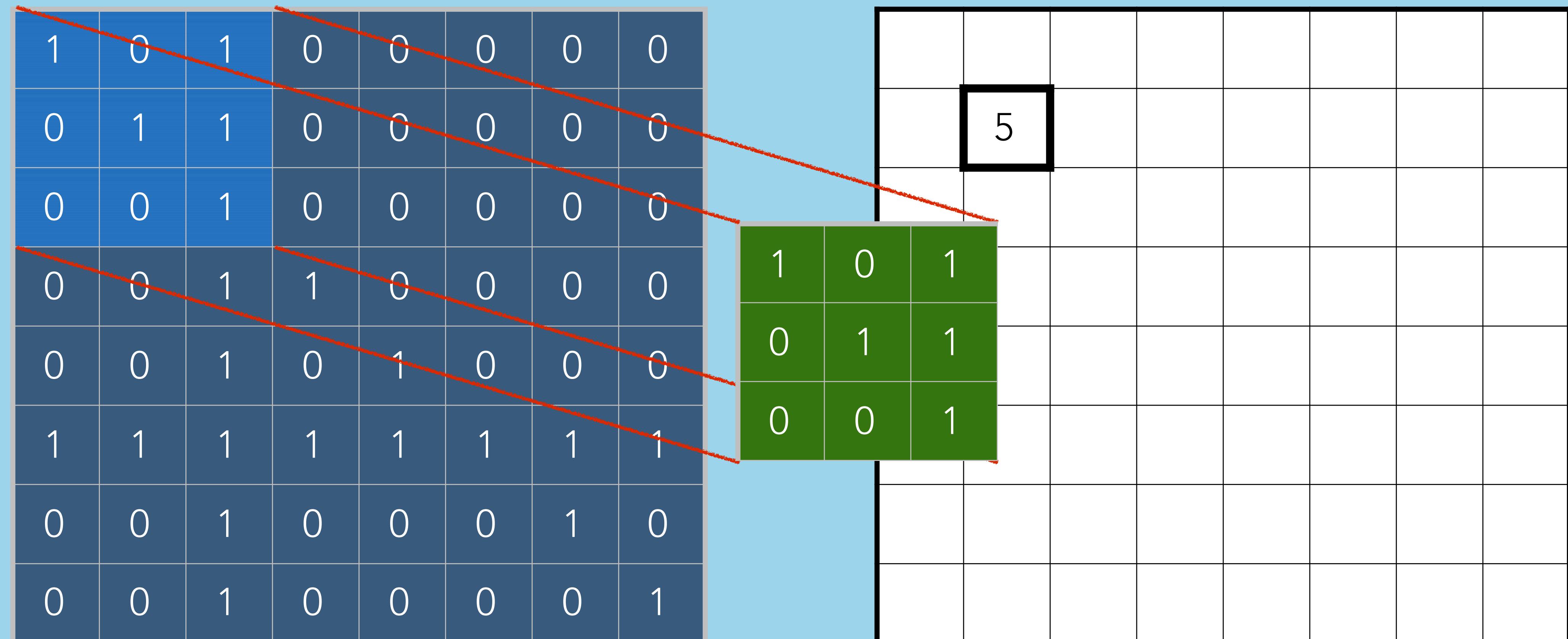
The convolutional  
neural network  
architecture was first  
described by  
Kunihiko Fukushima  
in 1980

# pre-defined filters

Discrete convolutions  
are matrix operations  
that can be used to  
apply **filters** to a  
matrix or array

# Discrete Convolutions $C = A \circledast h$

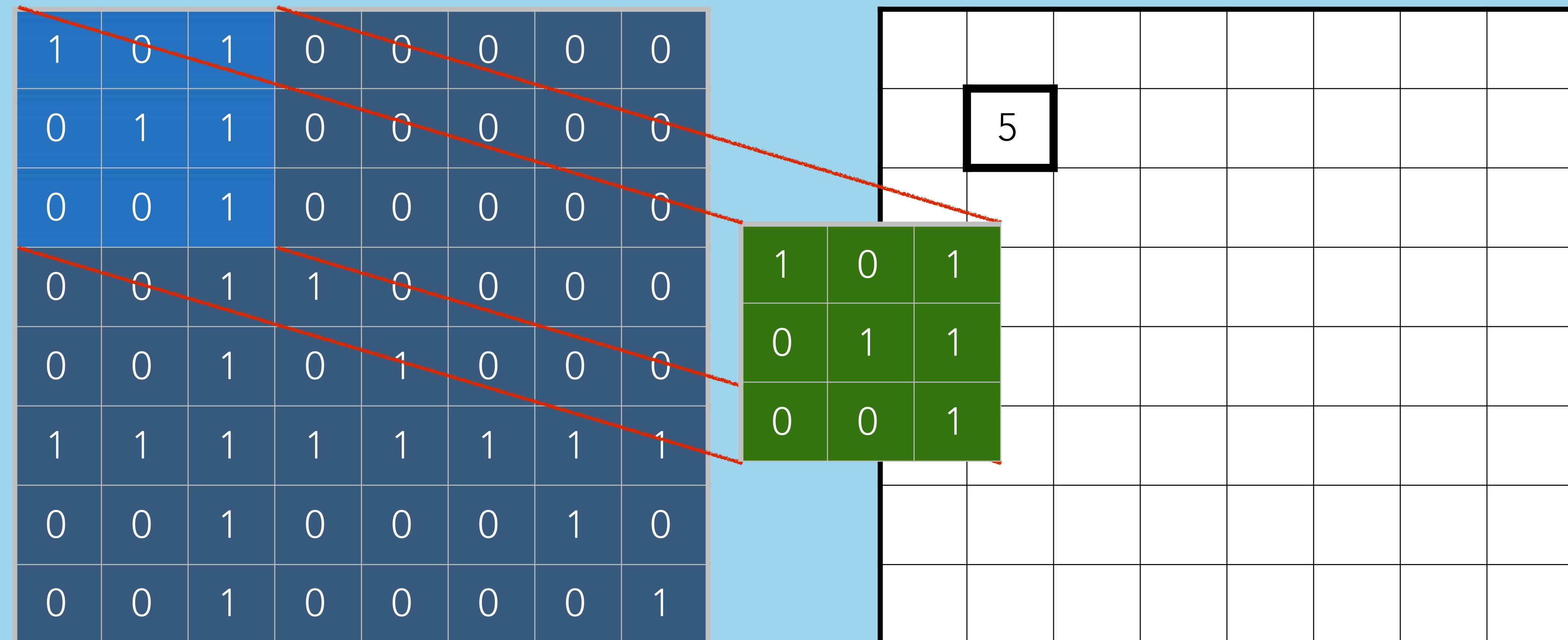
$$C[m, n] = \sum_{j=-\omega}^{\omega} \sum_{i=-\omega}^{\omega} h[i + \omega, j + \omega] * A[m + i, n + j]$$



# Discrete Convolutions $C = A \circledast h$

# Stride

# Edges



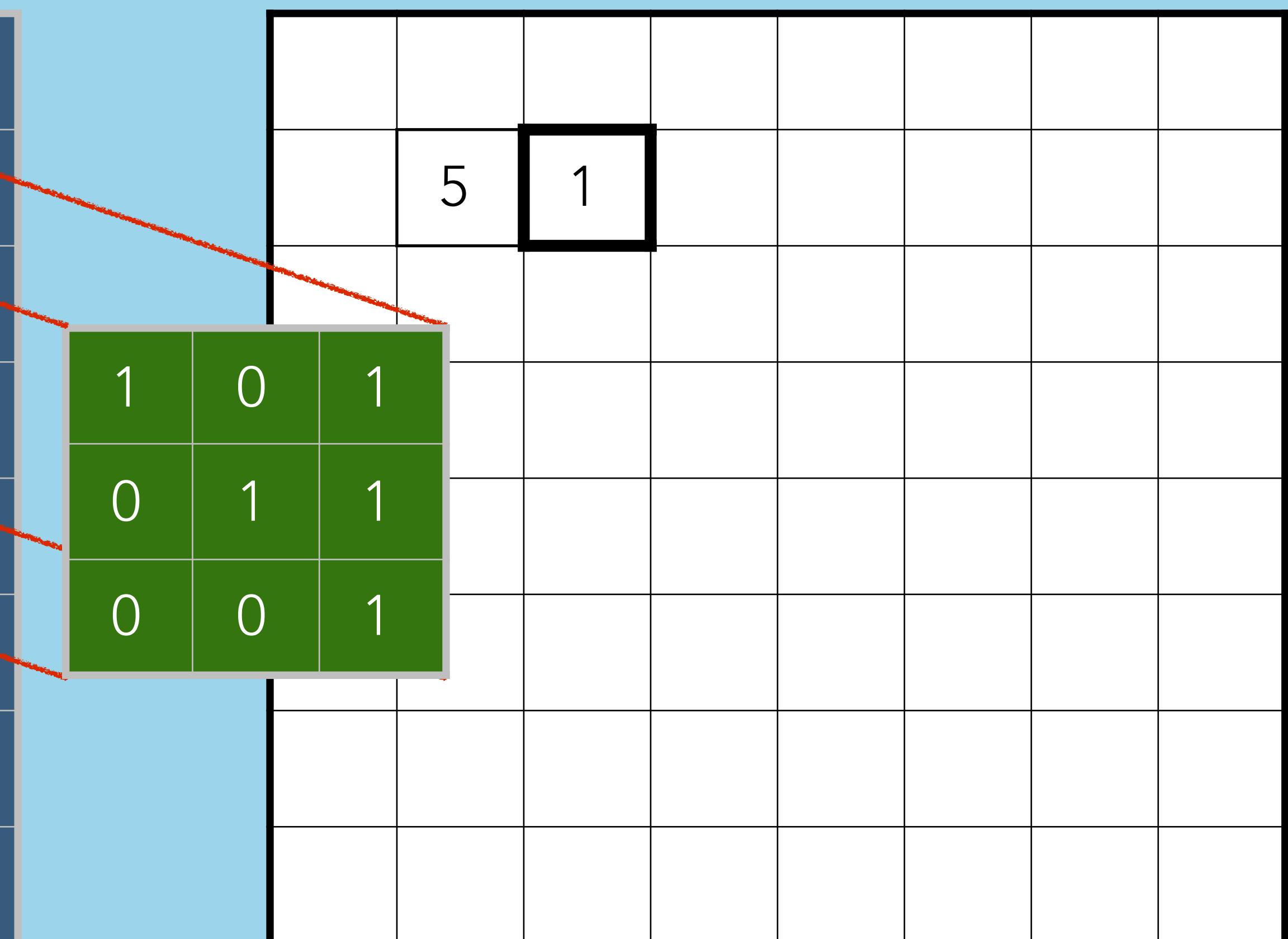
# Discrete Convolutions $C = A \circledast h$

Filter

Stride

Edges

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



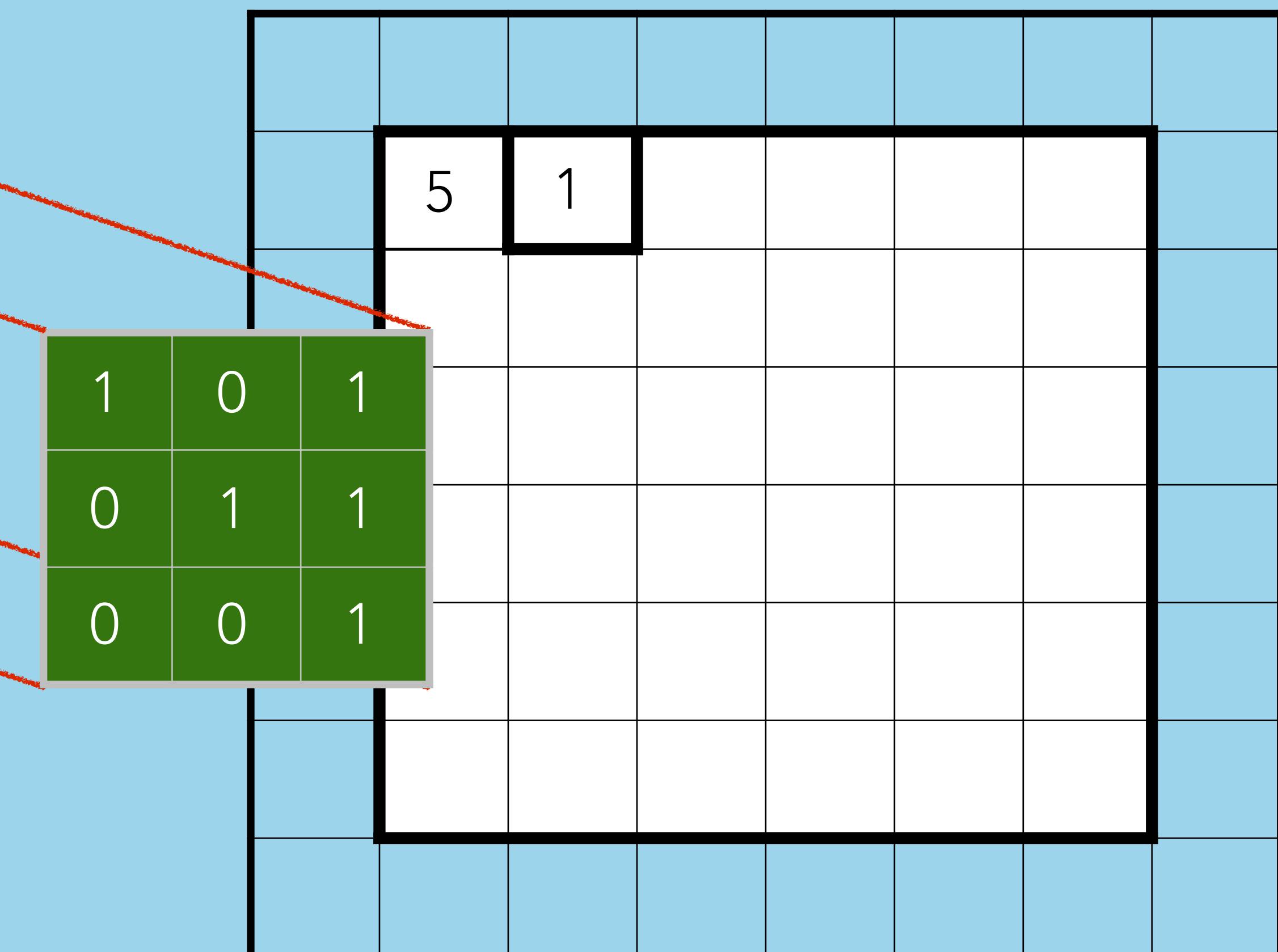
# Discrete Convolutions $C = A \circledast h$

Filter

Stride

Edges

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



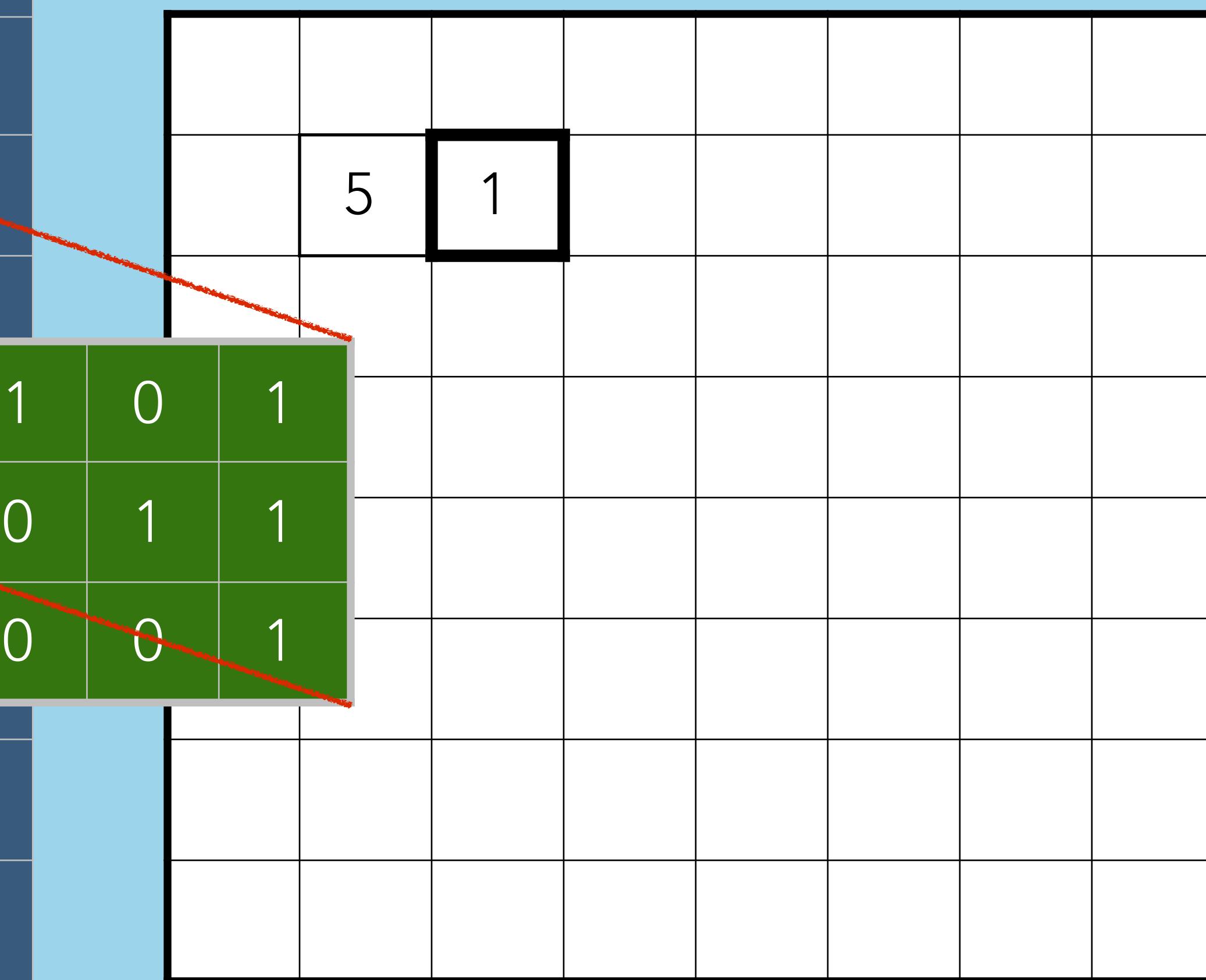
# Discrete Convolutions $C = A \circledast h$

Filter

Stride

Edges

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

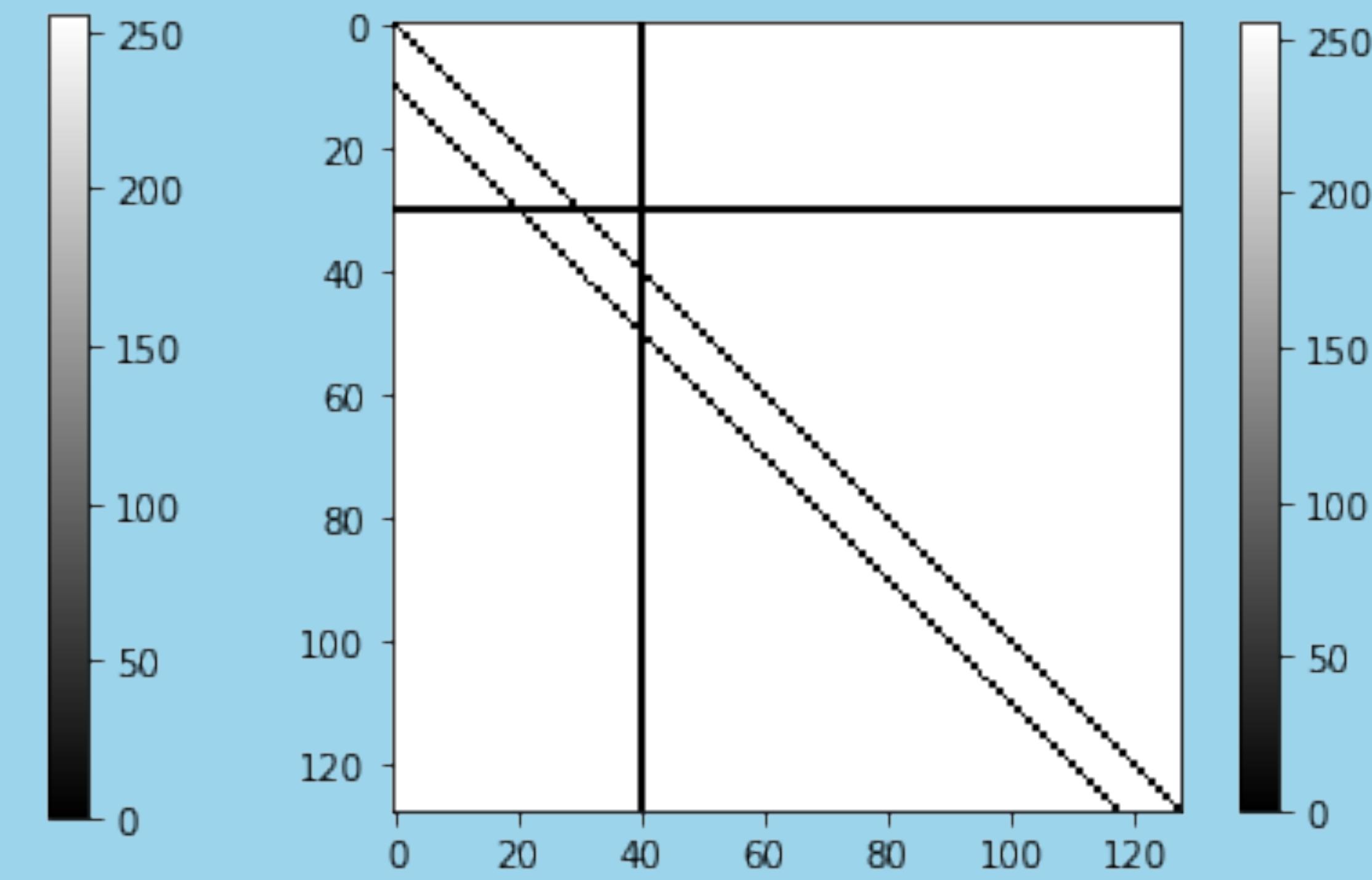
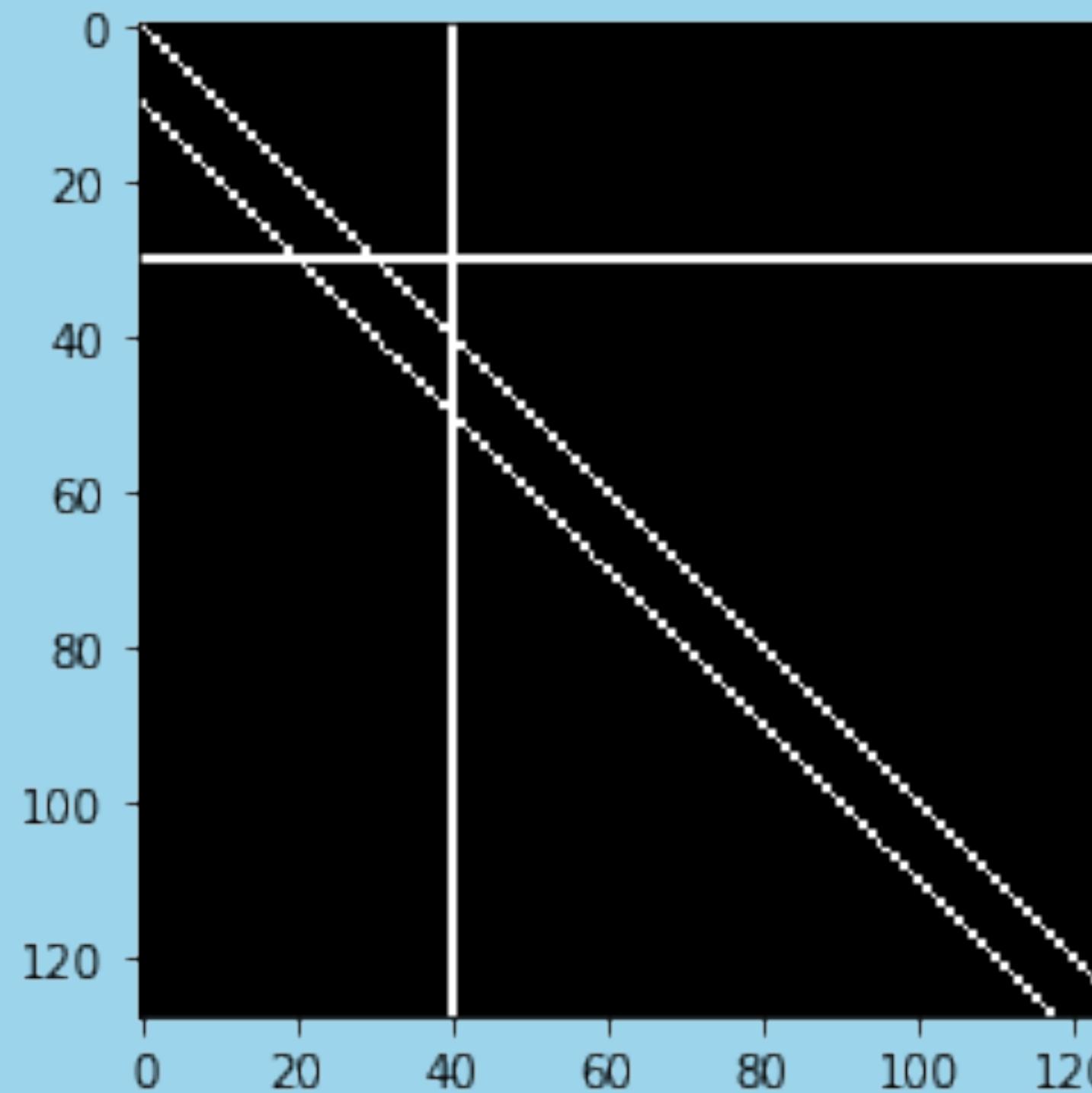


# Discrete Convolutions $C = A \circledast h$

Filter  
(11x11)

Stride  
1

Edges  
None

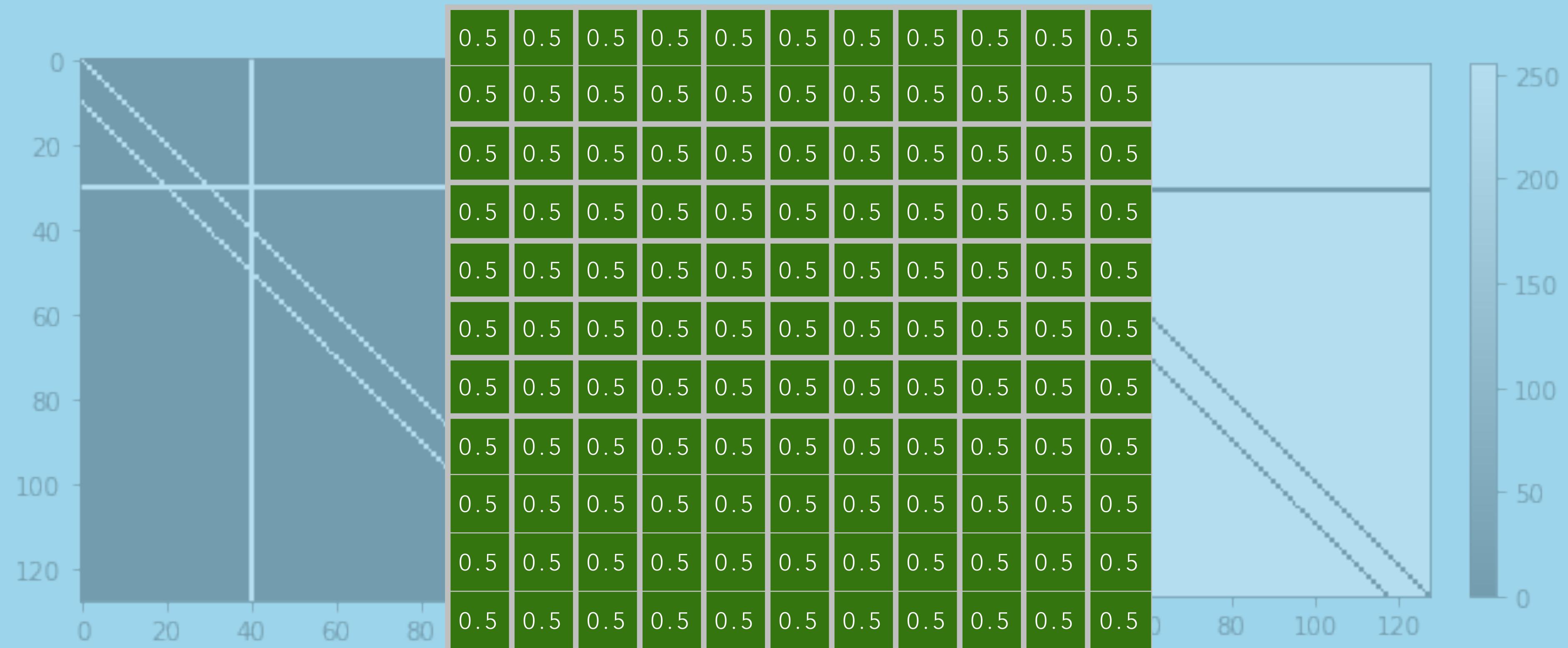


# Discrete Convolutions $C = A \circledast h$

# Stride

# 1

Edges  
None

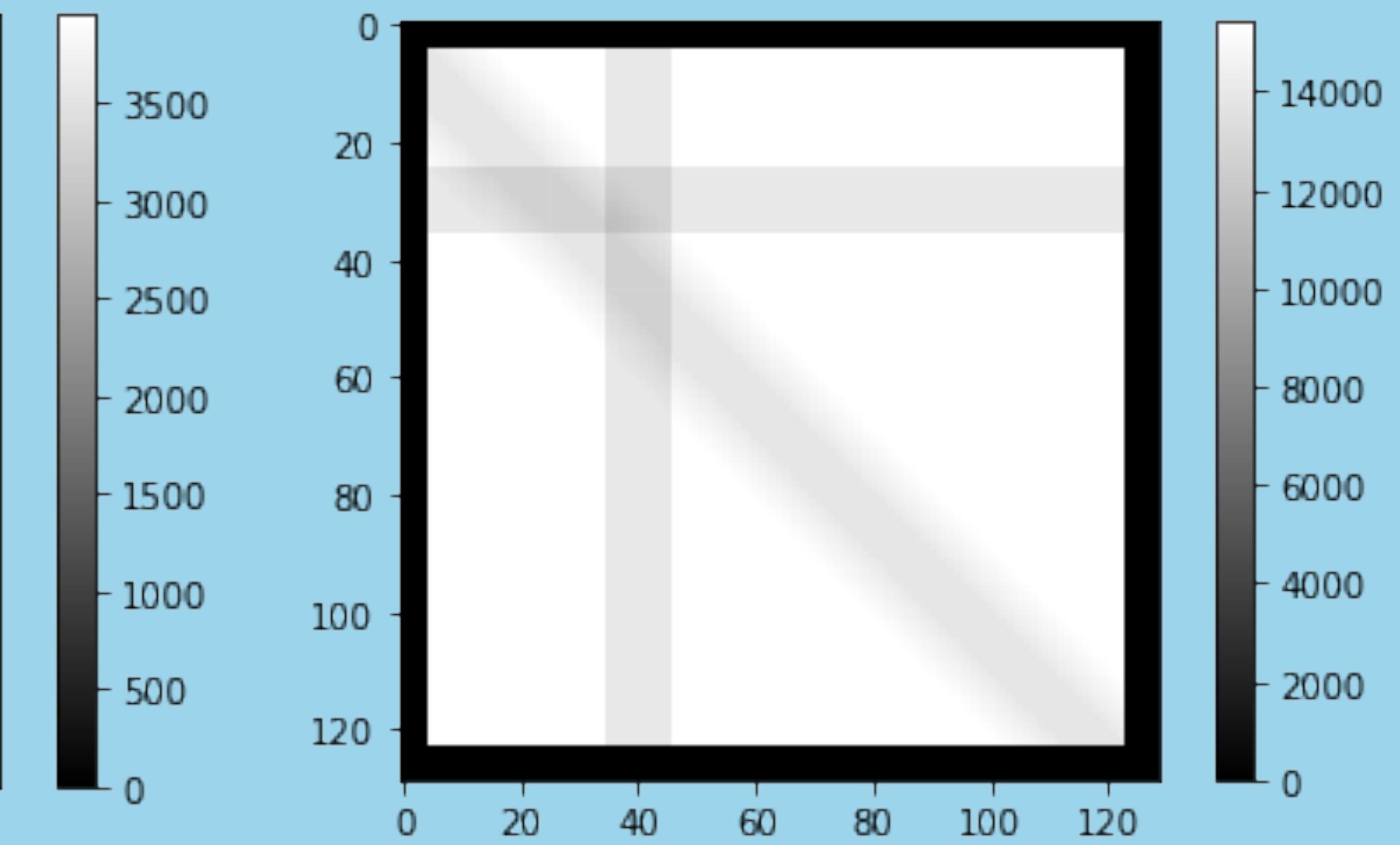
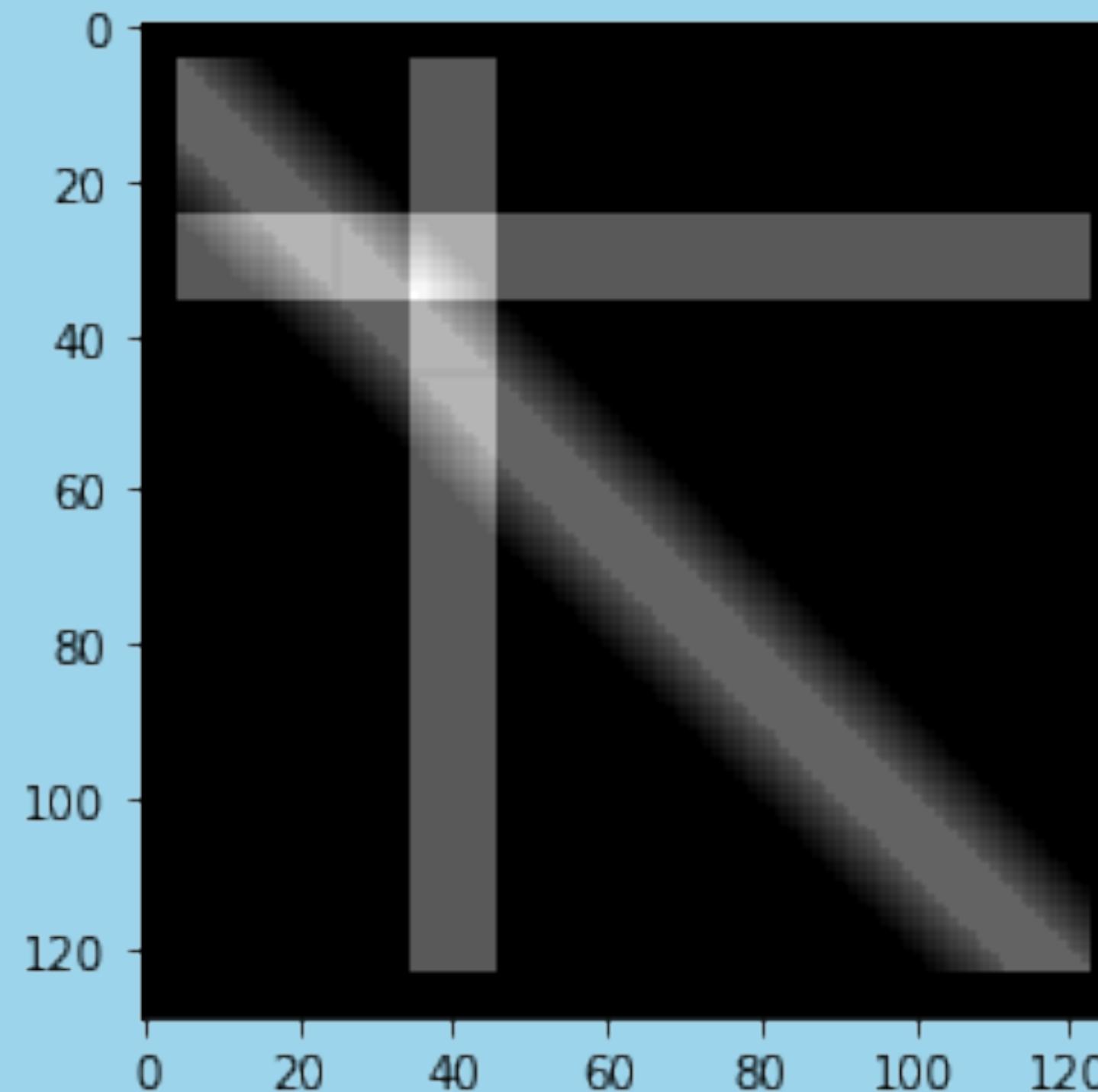


# Discrete Convolutions $C = A \circledast h$

Filter  
(11x11)

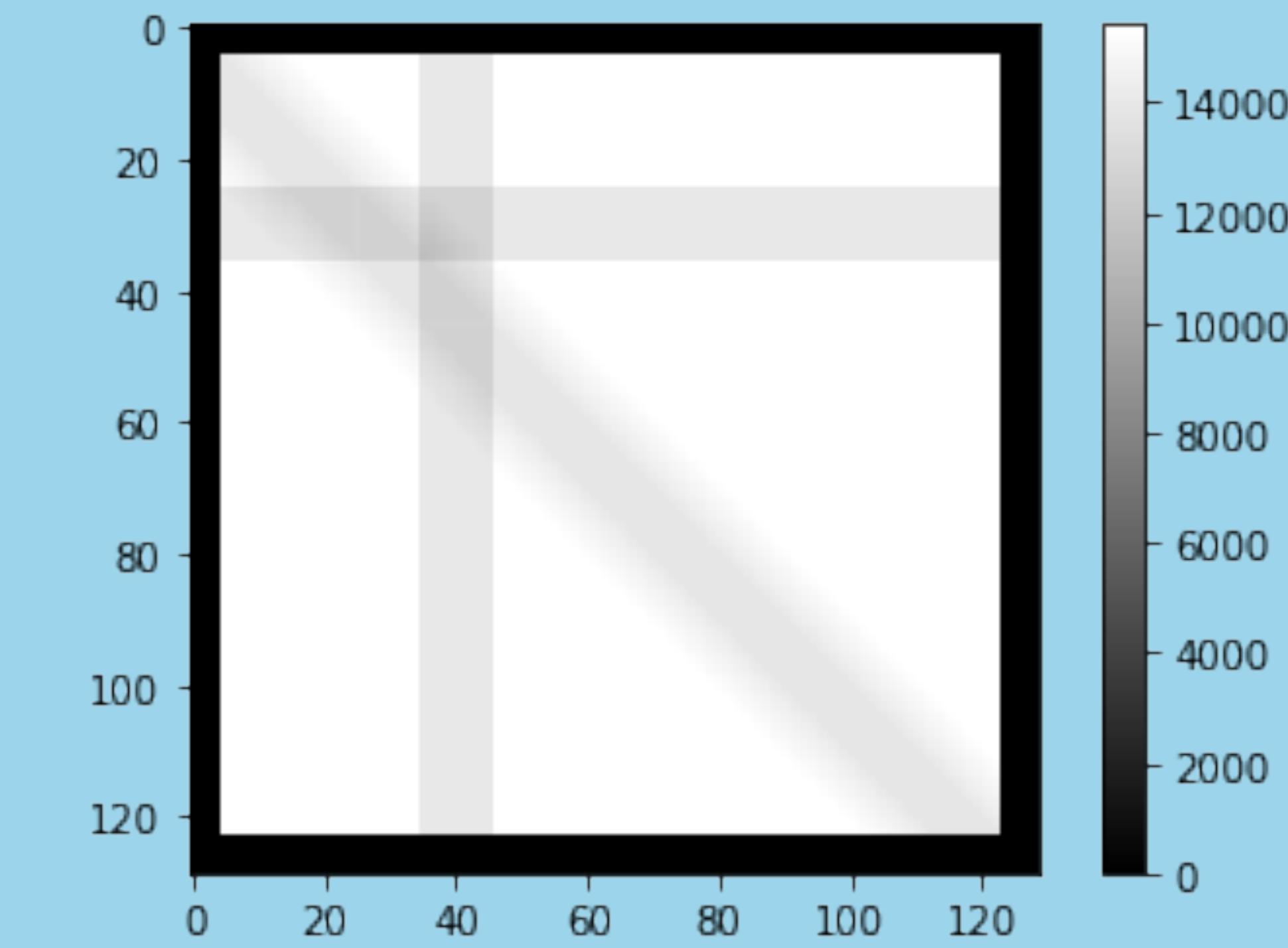
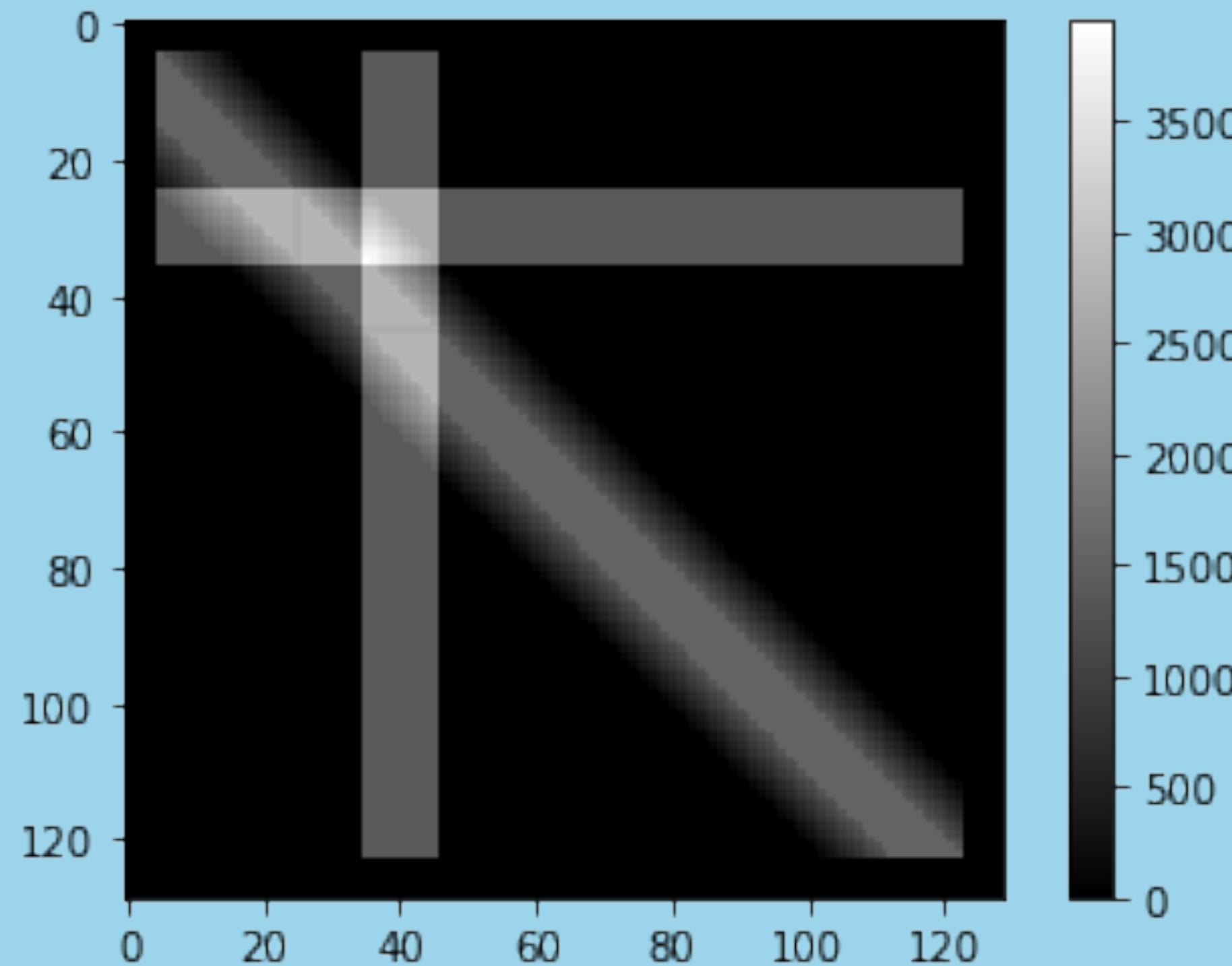
Stride  
1

Edges  
None



$$\text{Discrete Convolutions } C = A \circledast h$$

# Blurring Filter

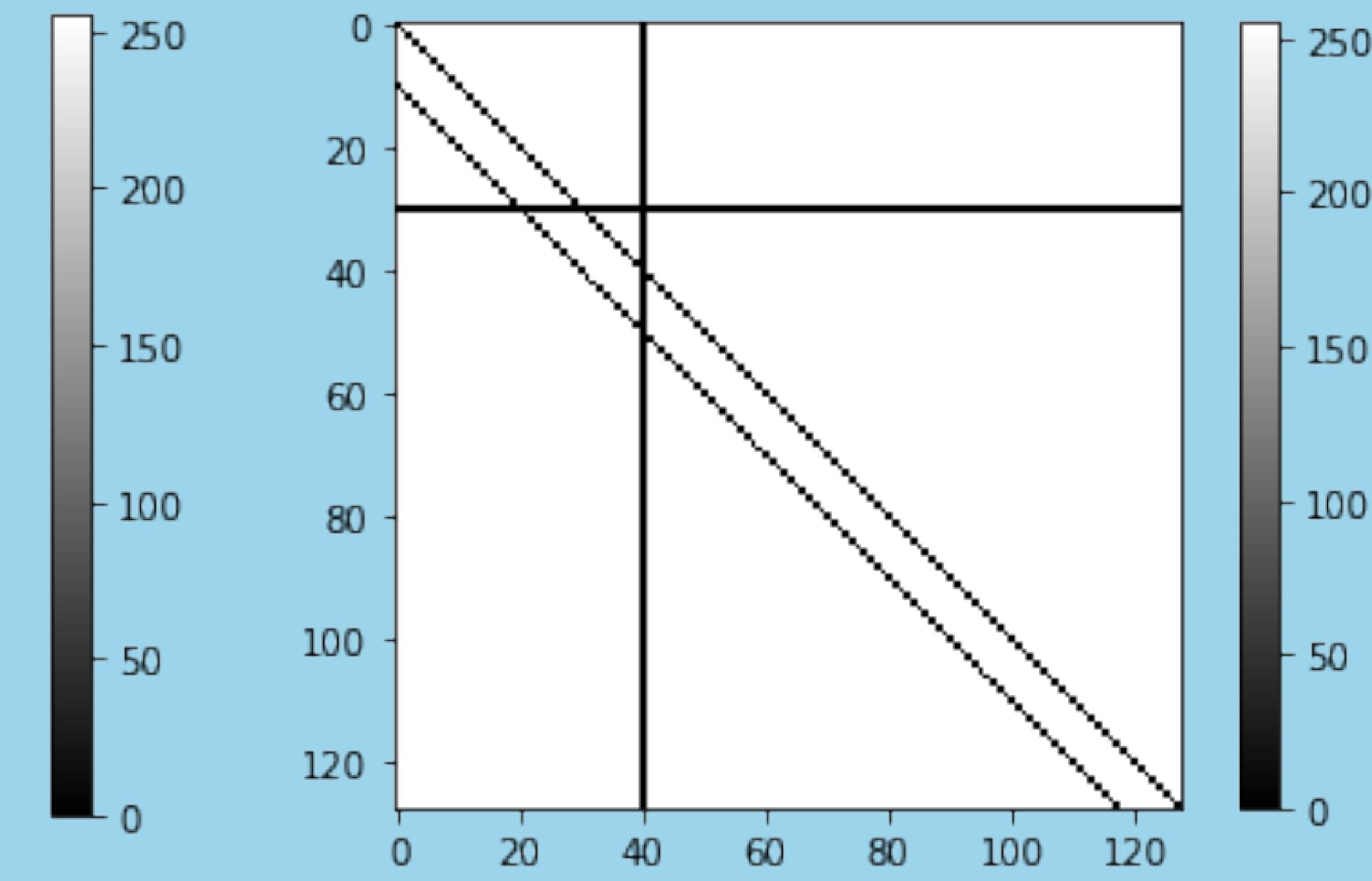
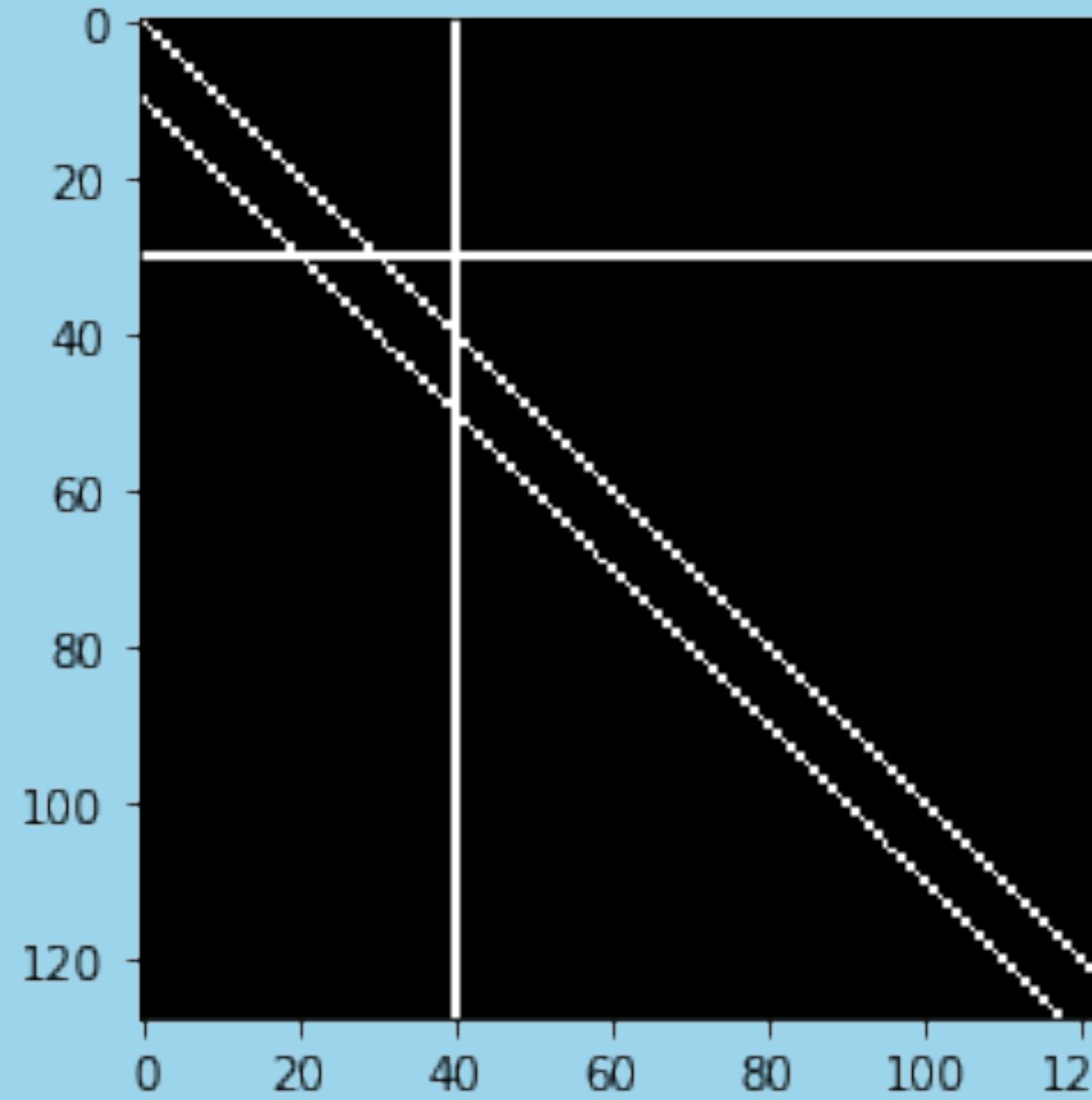


# Discrete Convolutions $C = A \circledast h$

Filter  
(3x3)

Stride  
1

Edges  
None

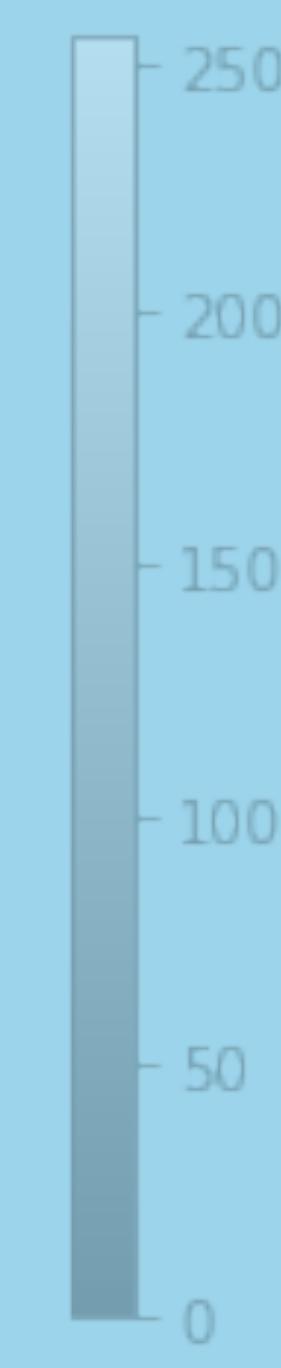
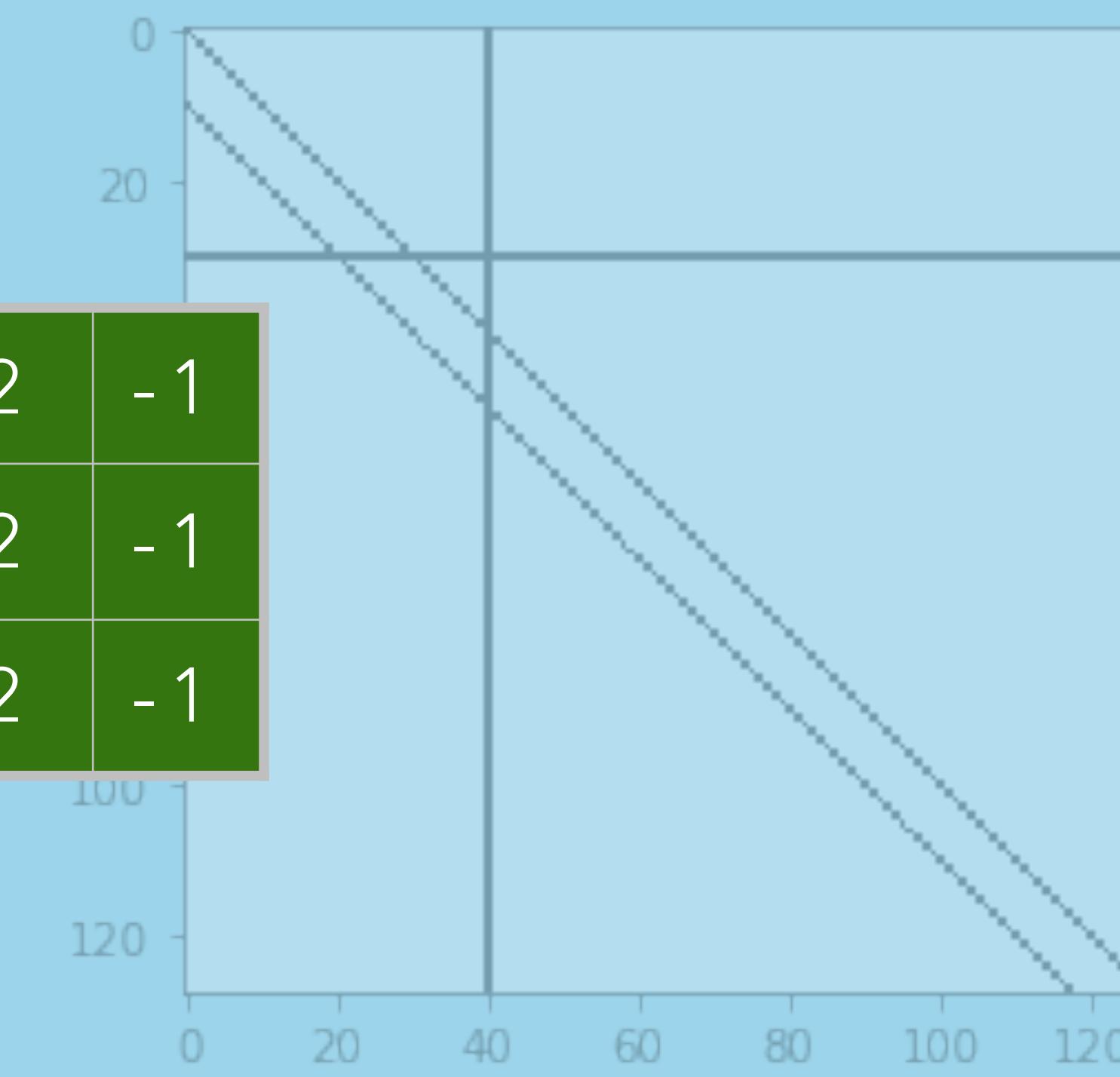
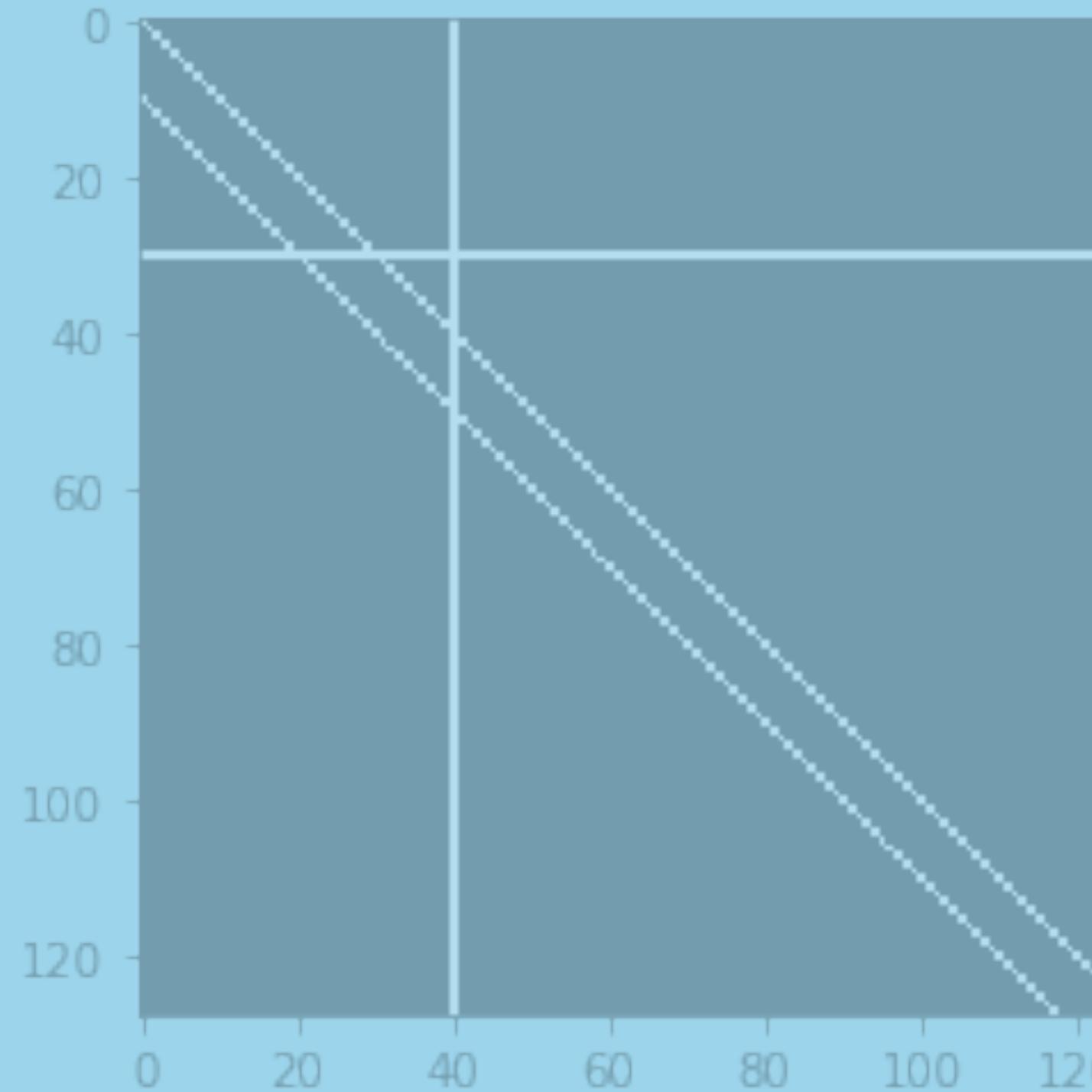


# Discrete Convolutions $C = A \circledast h$

Filter  
(3x3)

Stride  
1

Edges  
None

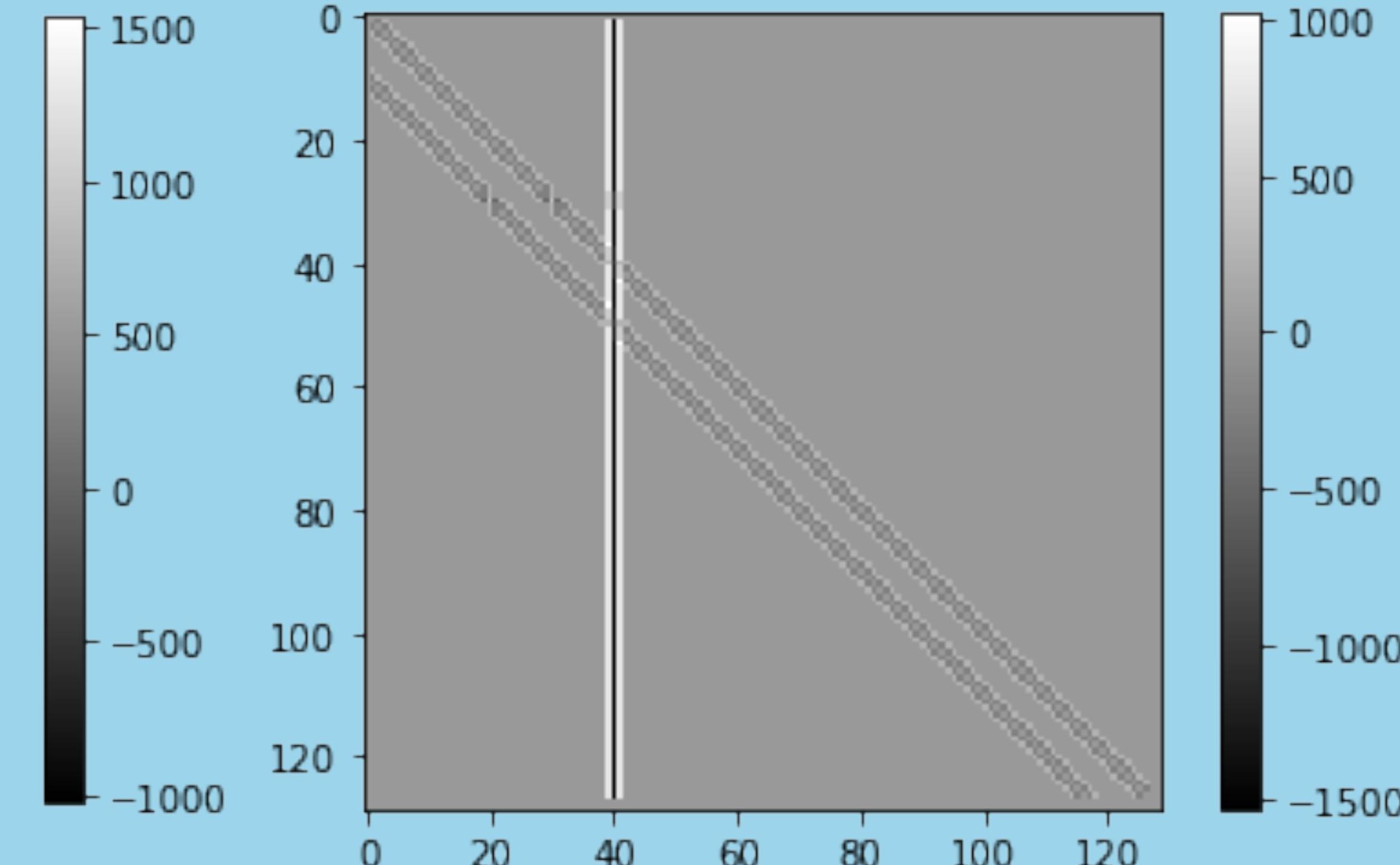
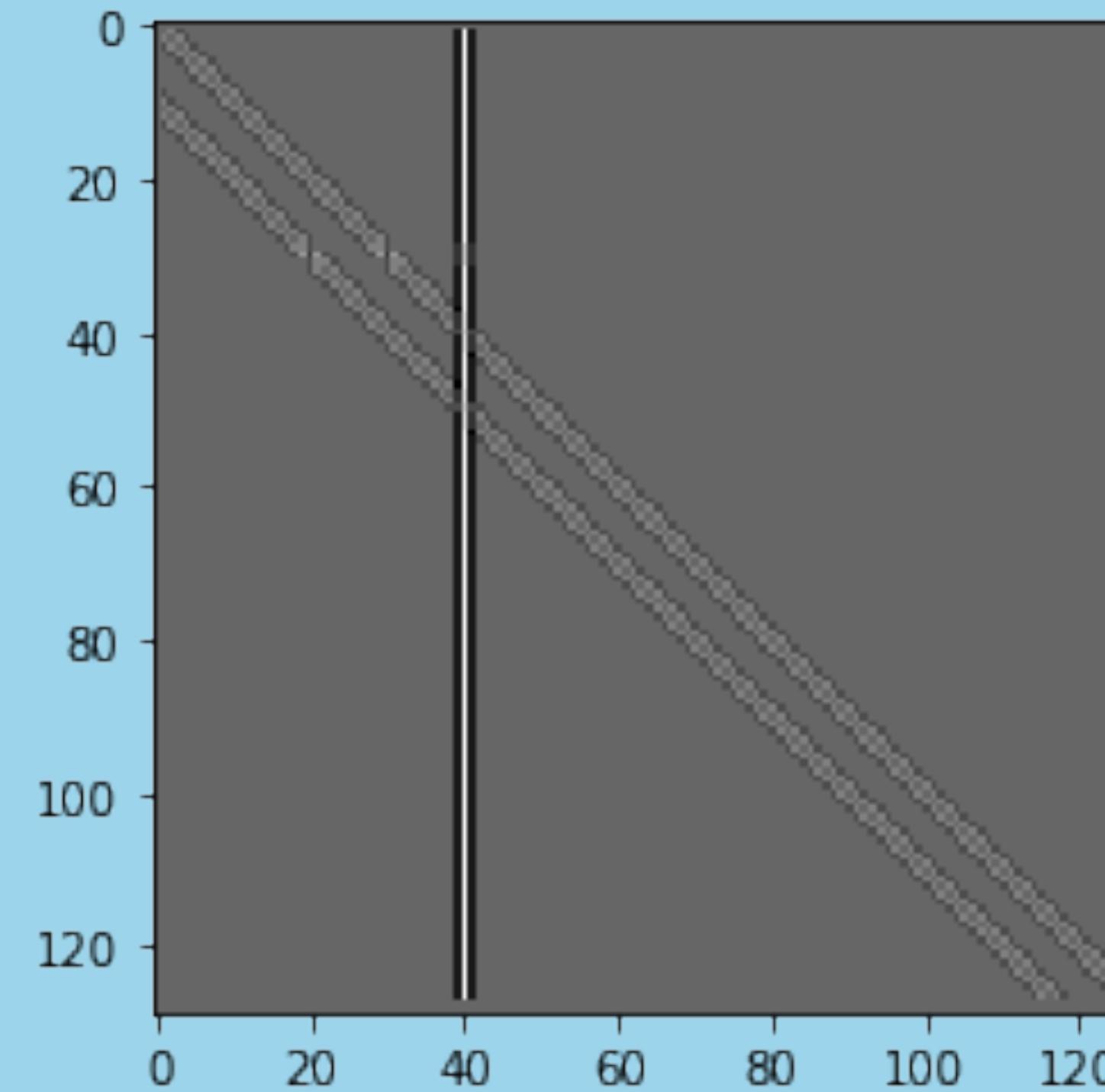


# Discrete Convolutions $C = A \circledast h$

Filter  
(3x3)

Stride  
1

Edges  
None

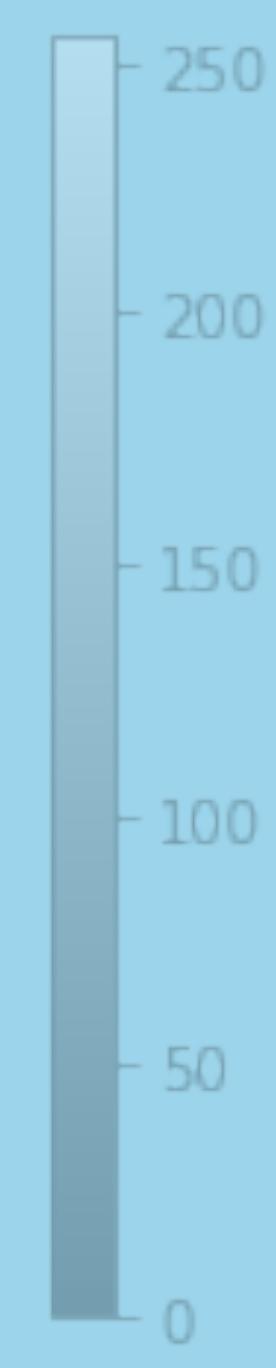
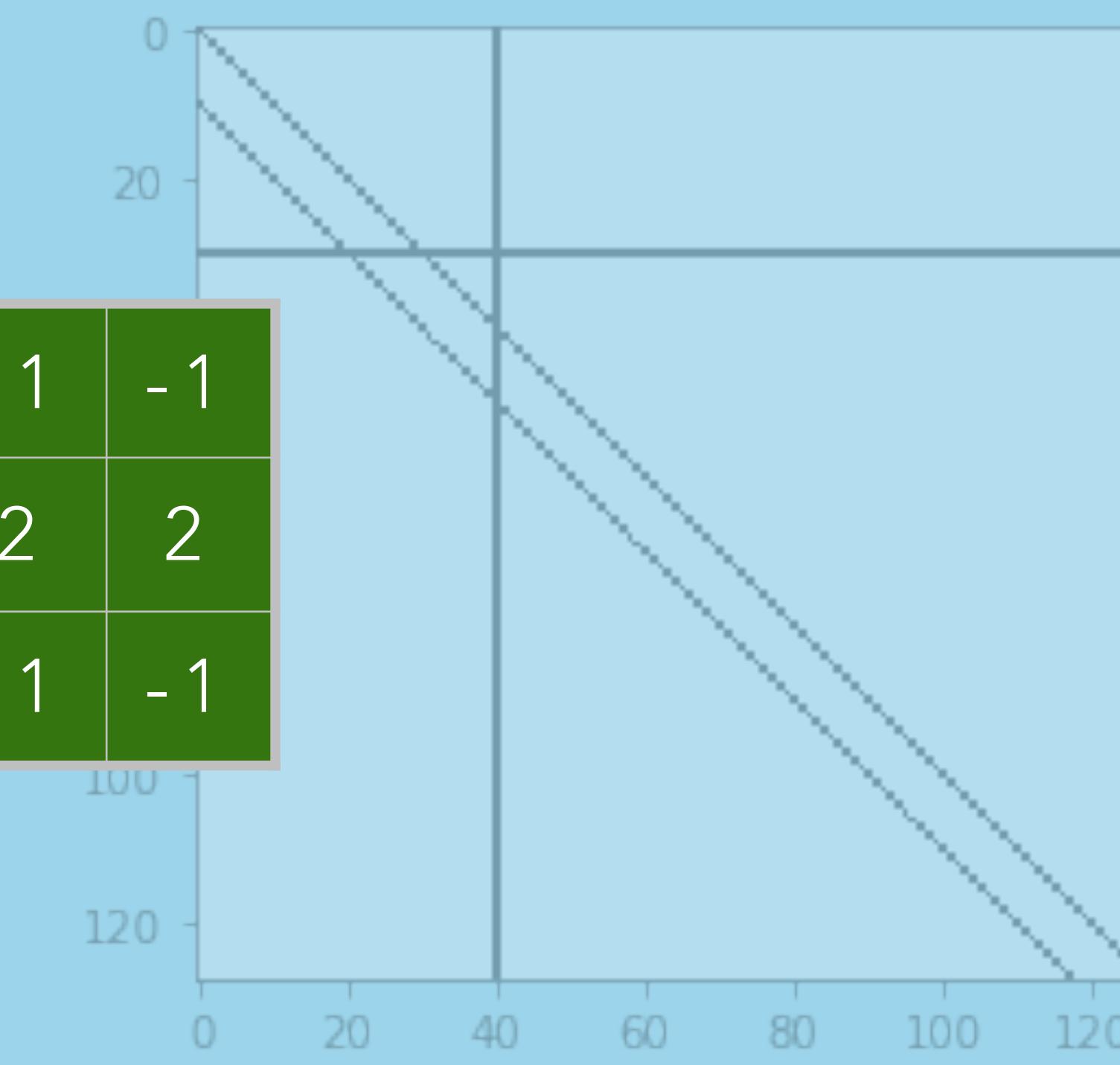
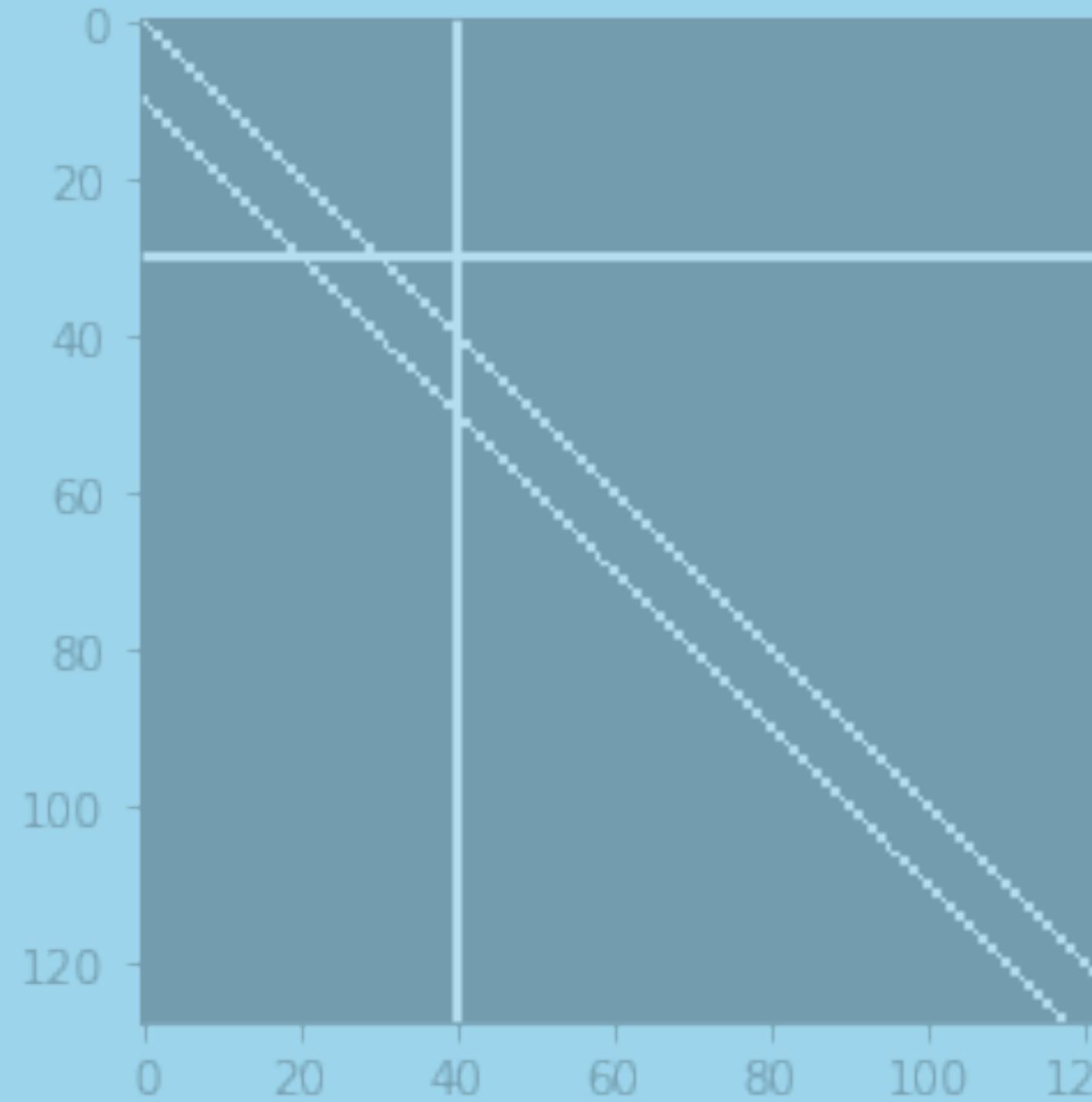


# Discrete Convolutions $C = A \circledast h$

Filter  
(3x3)

Stride  
1

Edges  
None

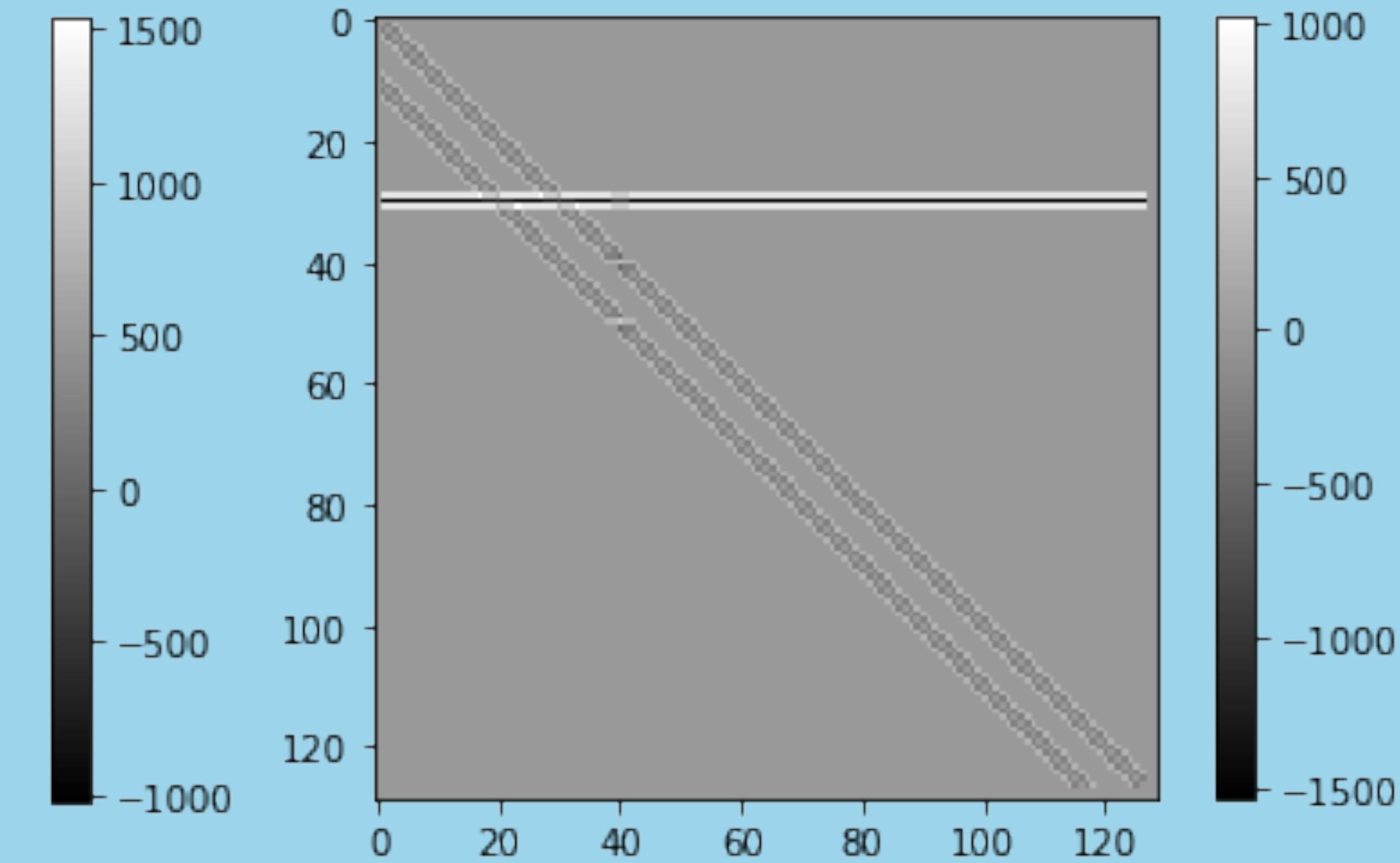
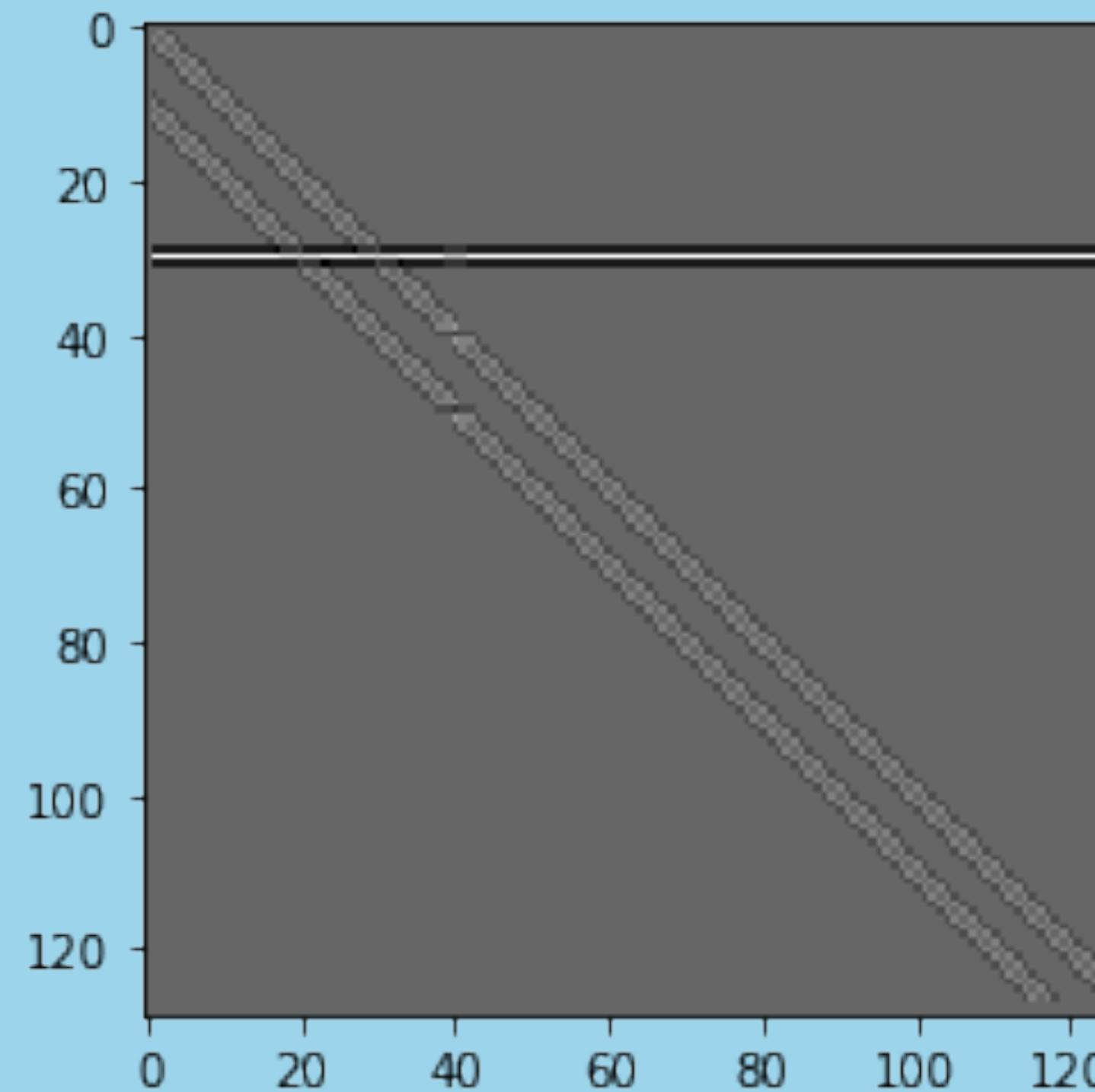


# Discrete Convolutions $C = A \circledast h$

Filter  
(3x3)

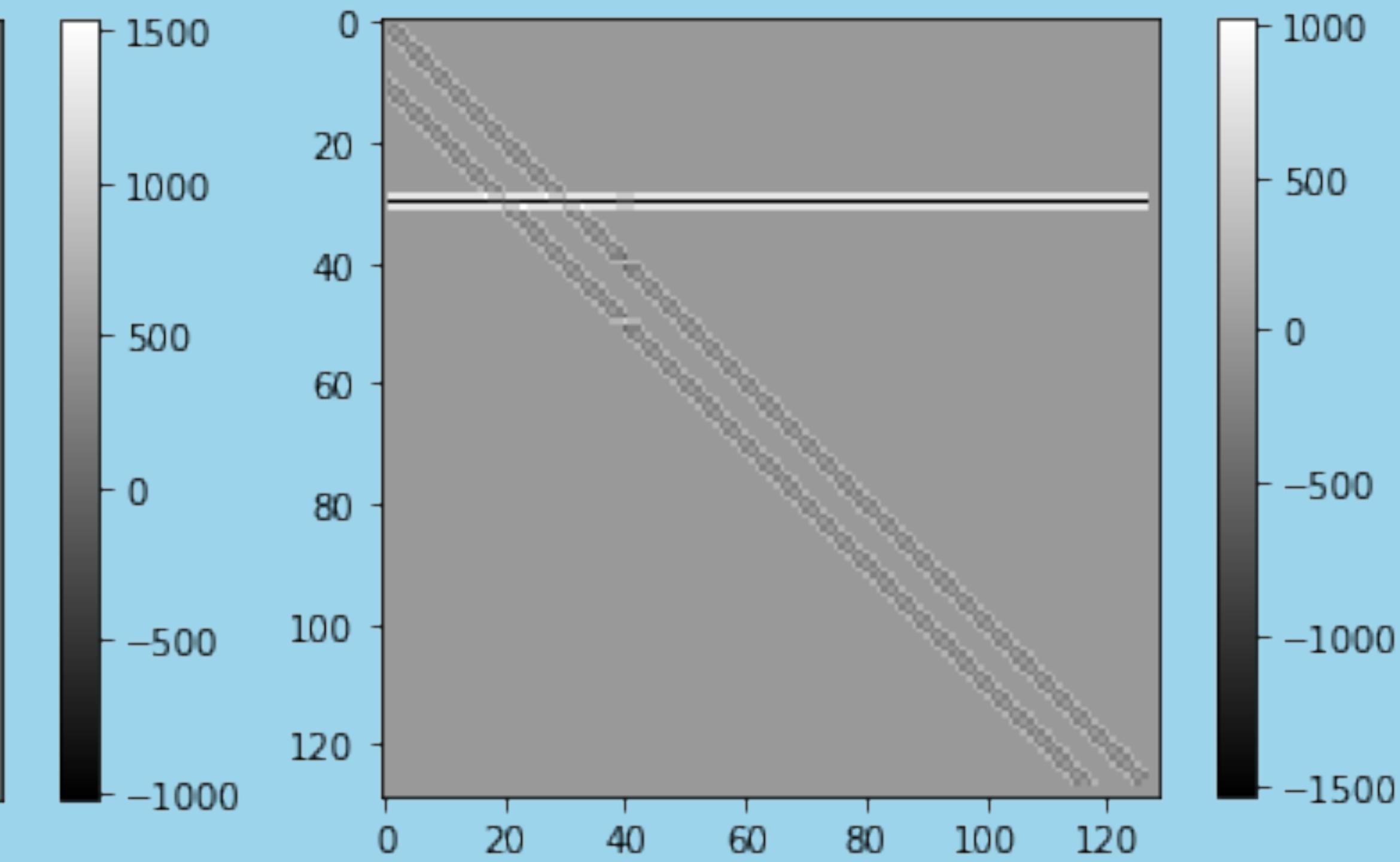
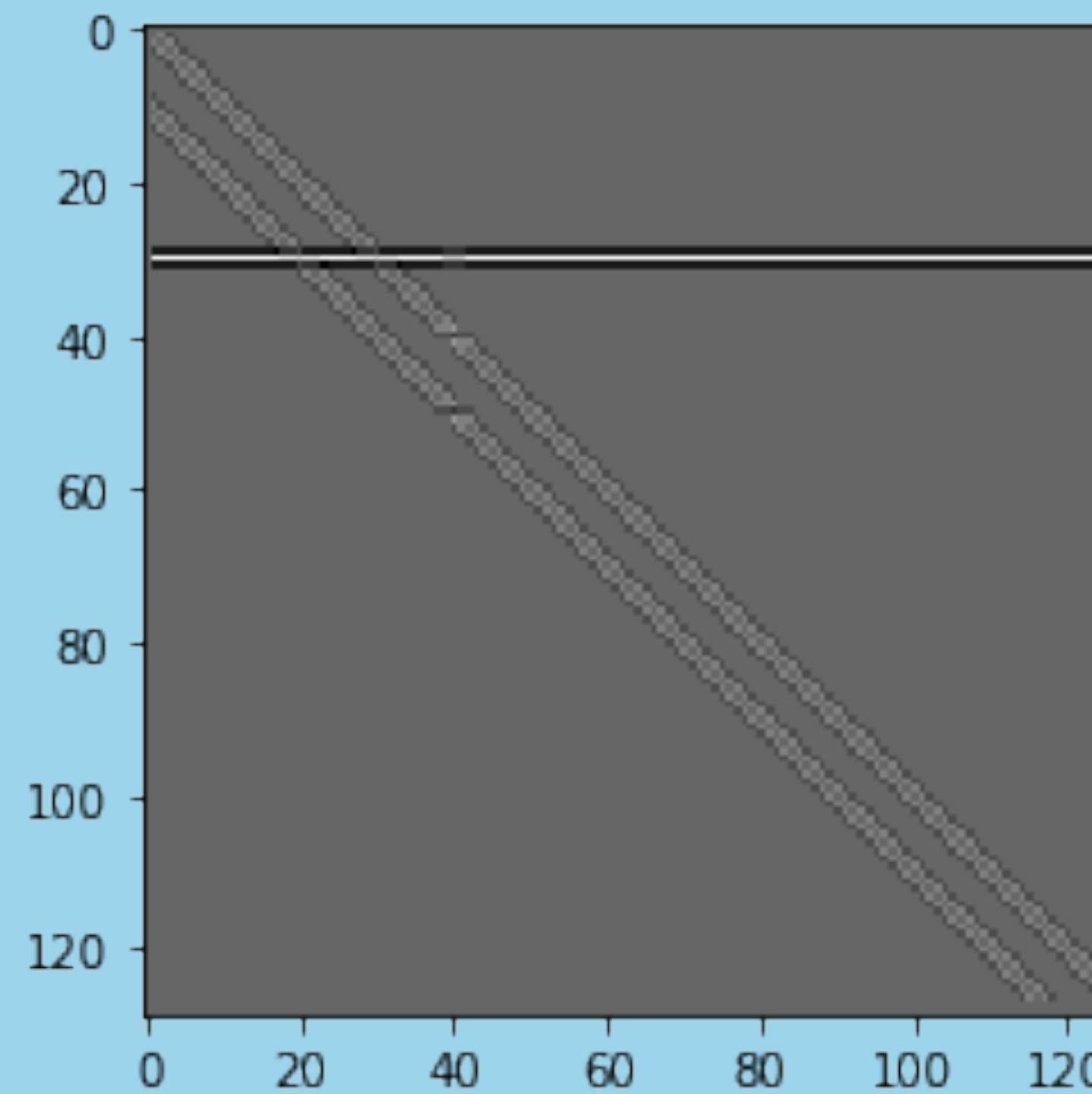
Stride  
1

Edges  
None



$$\text{Discrete Convolutions } C = A \circledast h$$

## Line Detection / Extraction



$$\text{Discrete Convolutions } C = A \circledast h$$

# Photograph



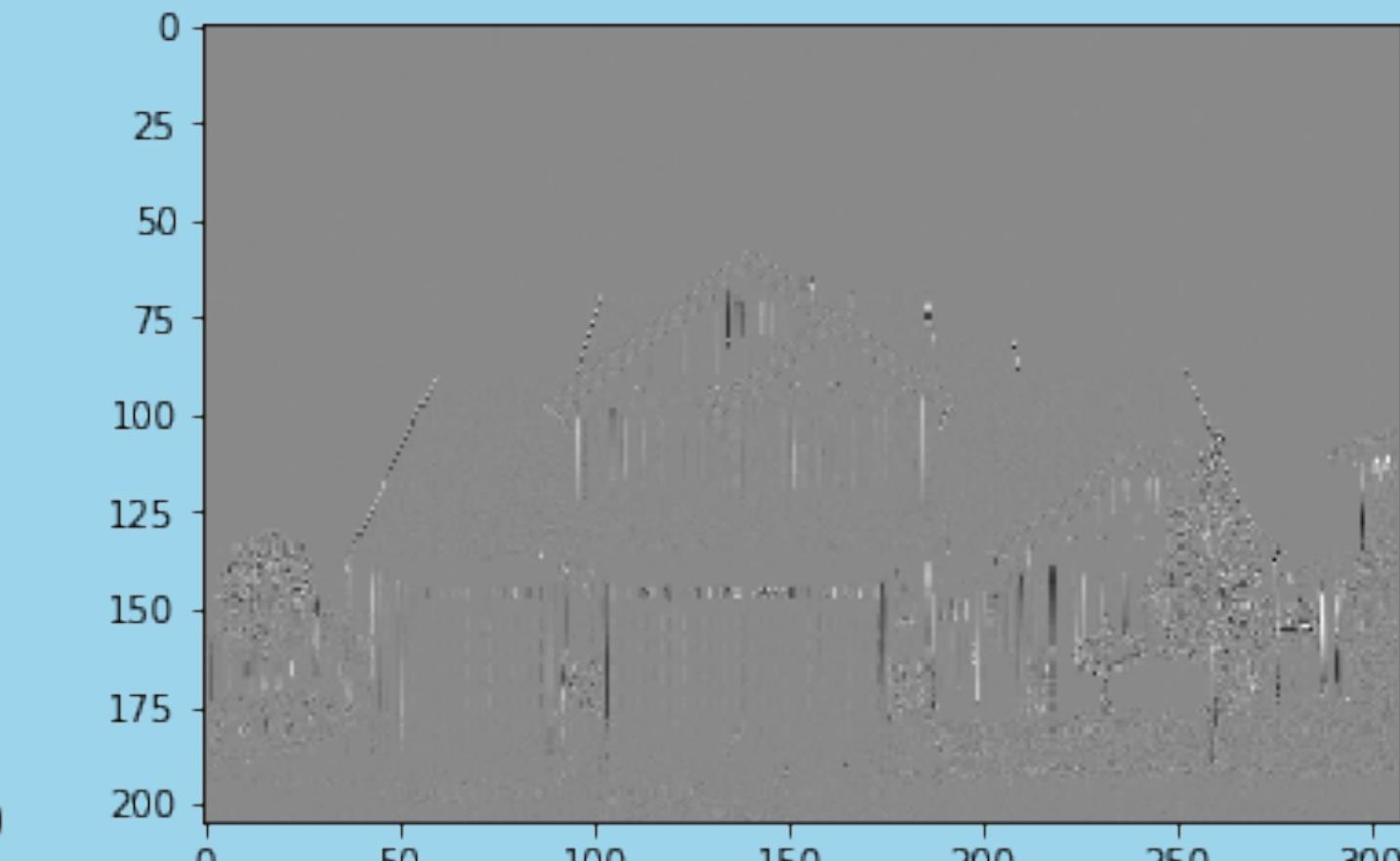
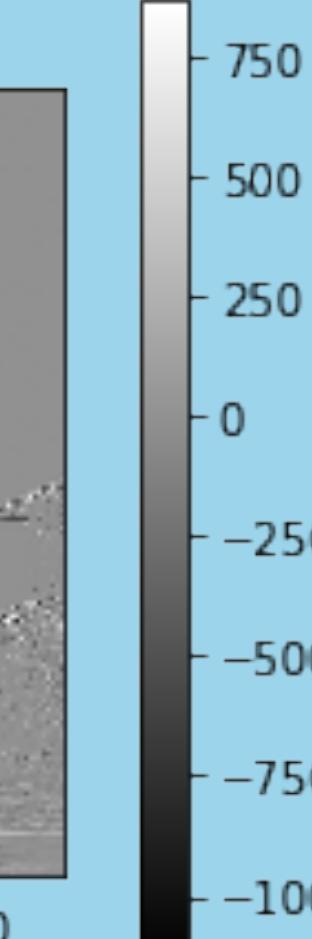
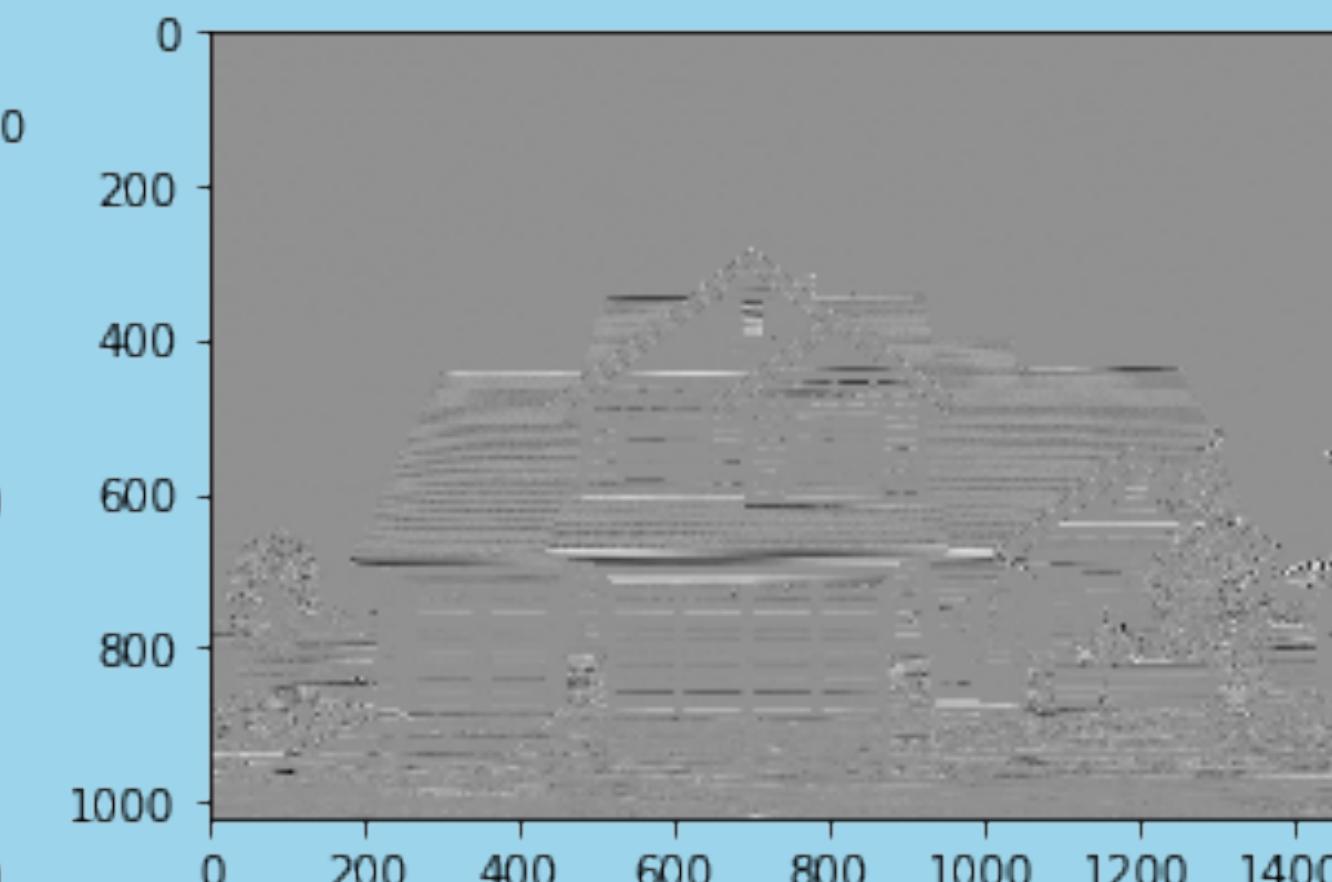
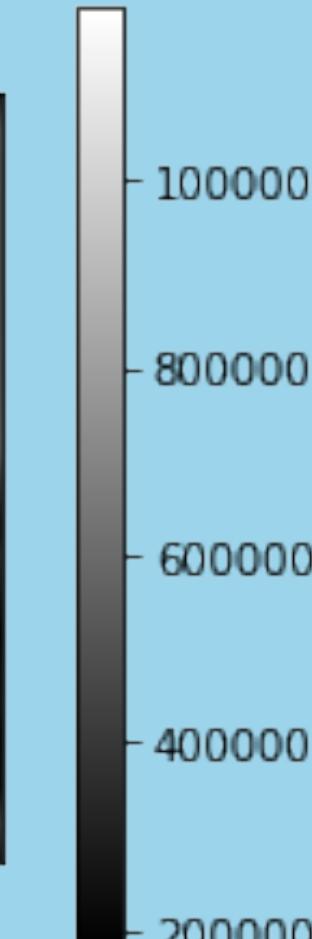
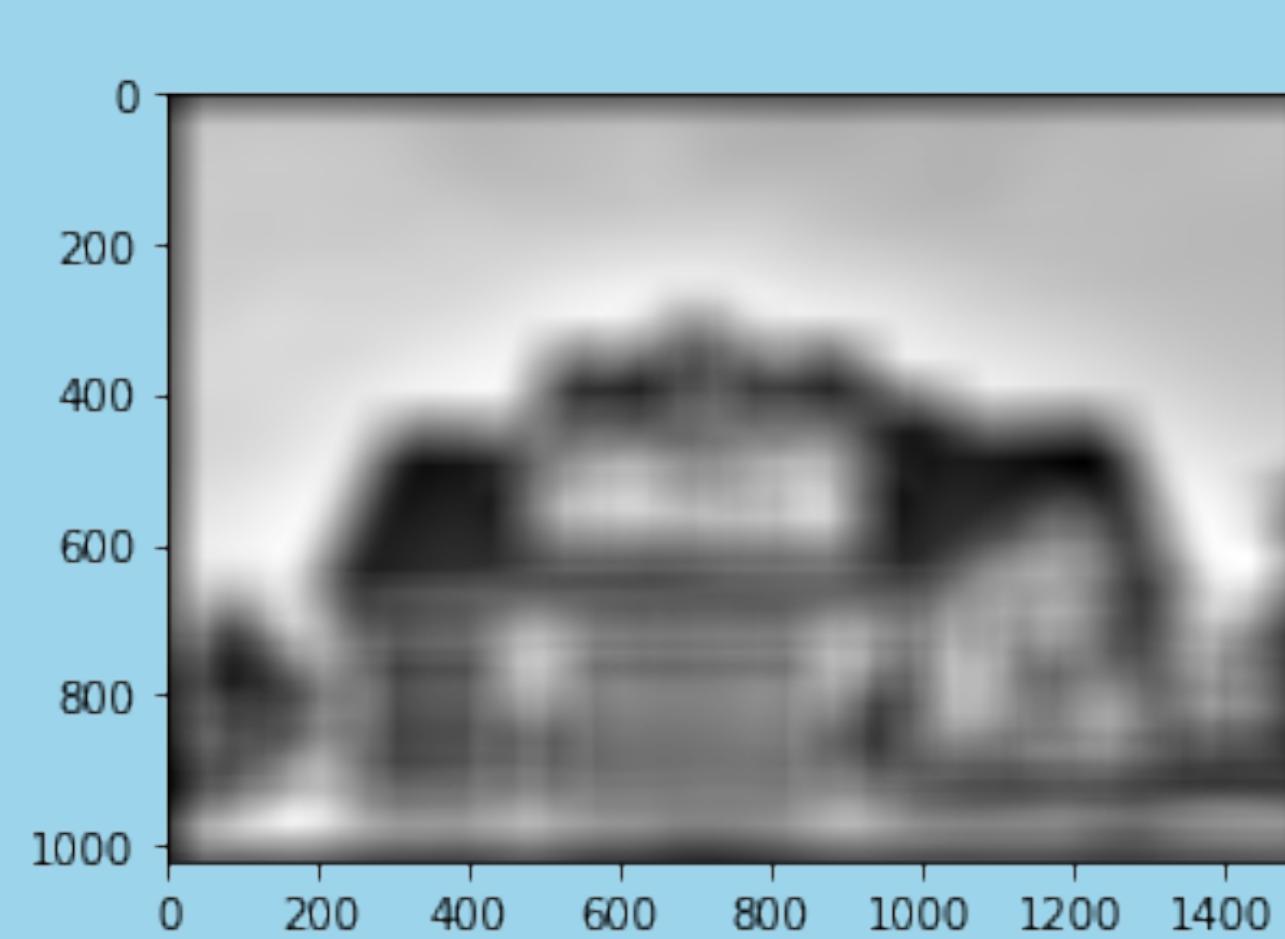
$$\text{Discrete Convolutions } C = A \circledast h$$

Photograph



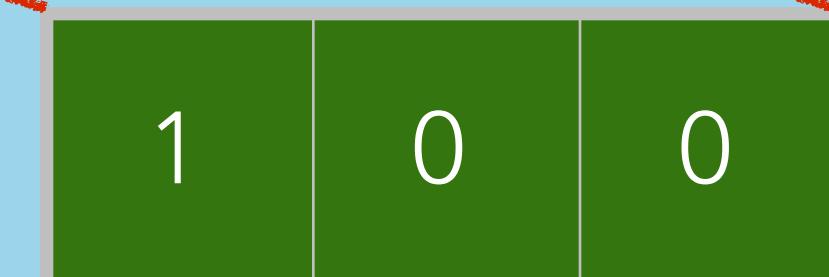
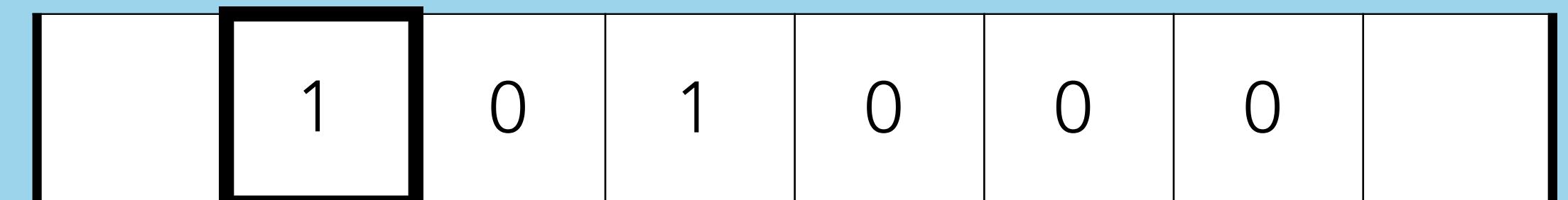
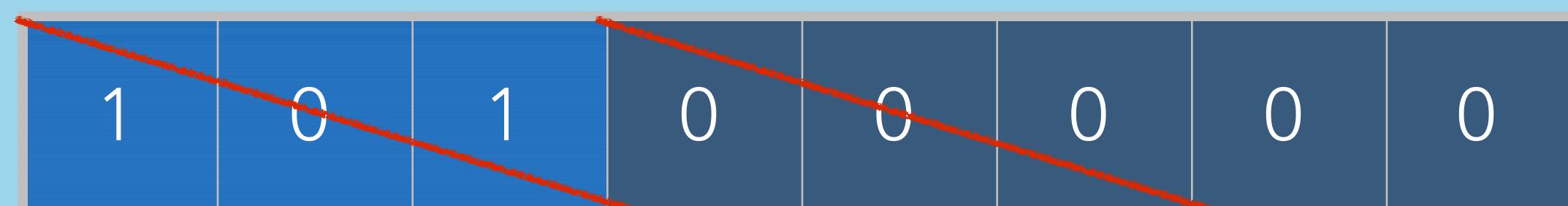
# Discrete Convolutions $C = A \circledast h$

Filter  
(101x101)



# N-D Discrete Convolutions

$$C[m, n] = \sum_{i=-\omega}^{\omega} h[i + \omega] * A[m + i]$$

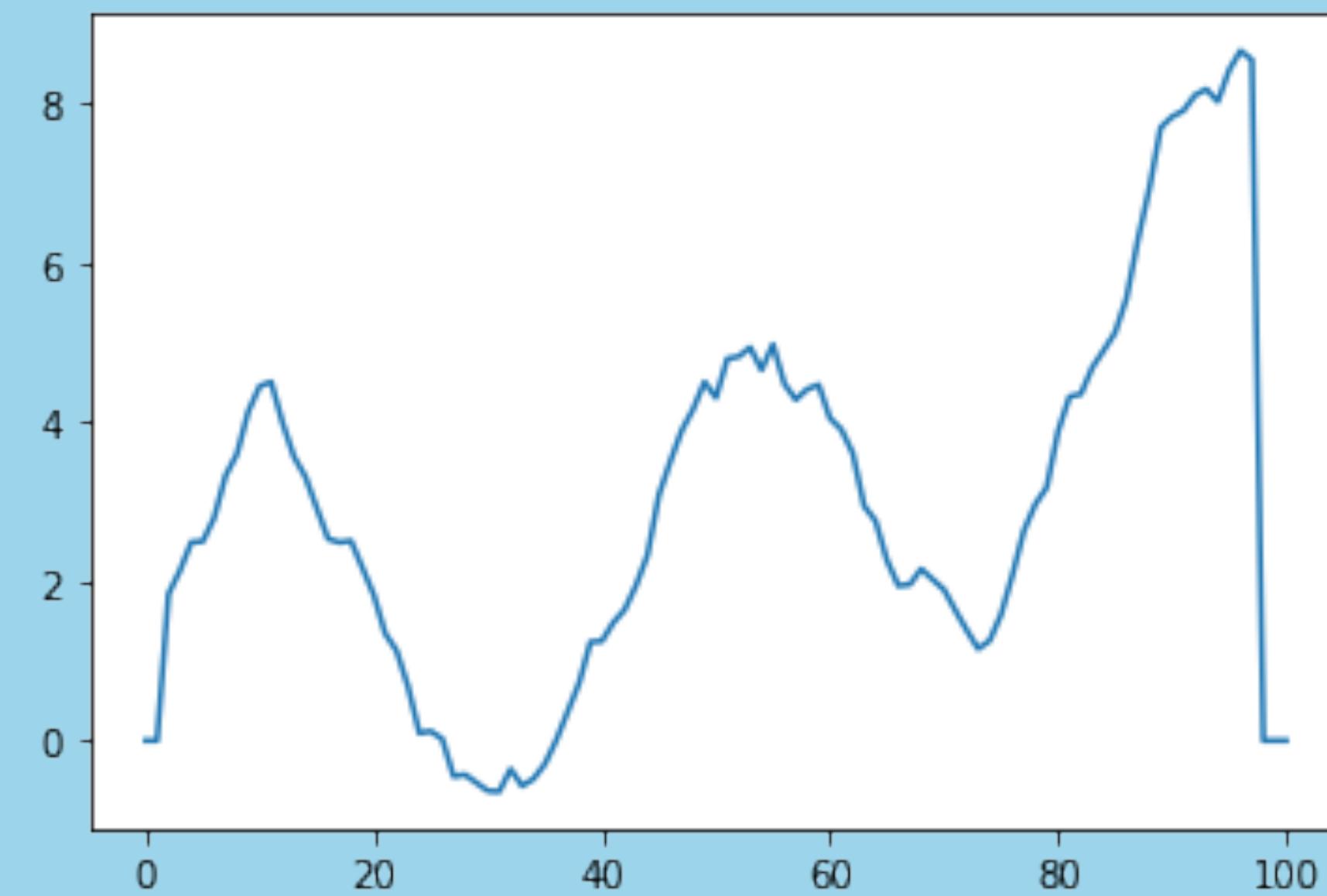
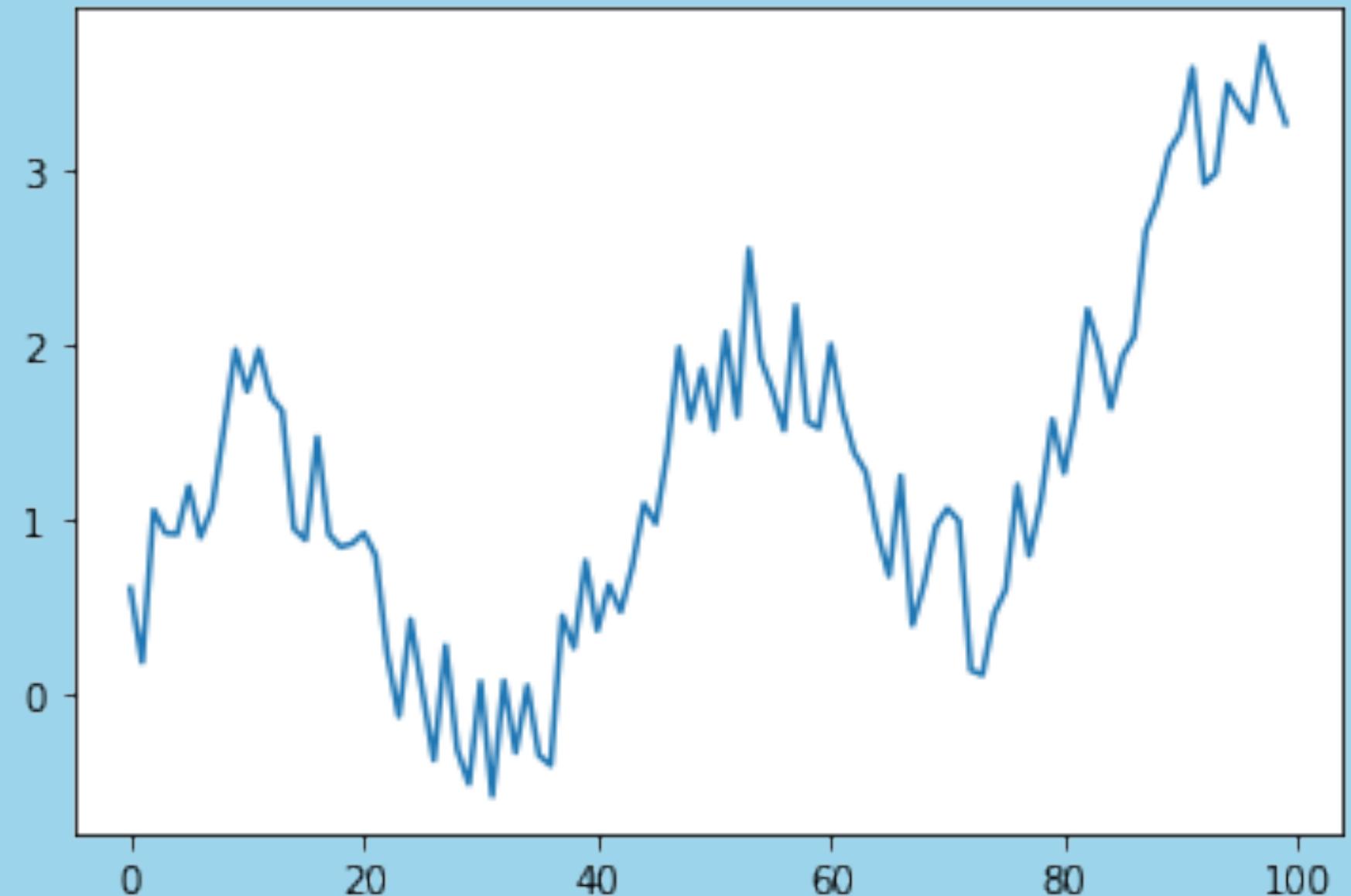


# N-D Discrete Convolutions

Filter  
(1x5)

Stride  
1

Edges  
None



0.5	0.5	0.5	0.5	0.5
-----	-----	-----	-----	-----

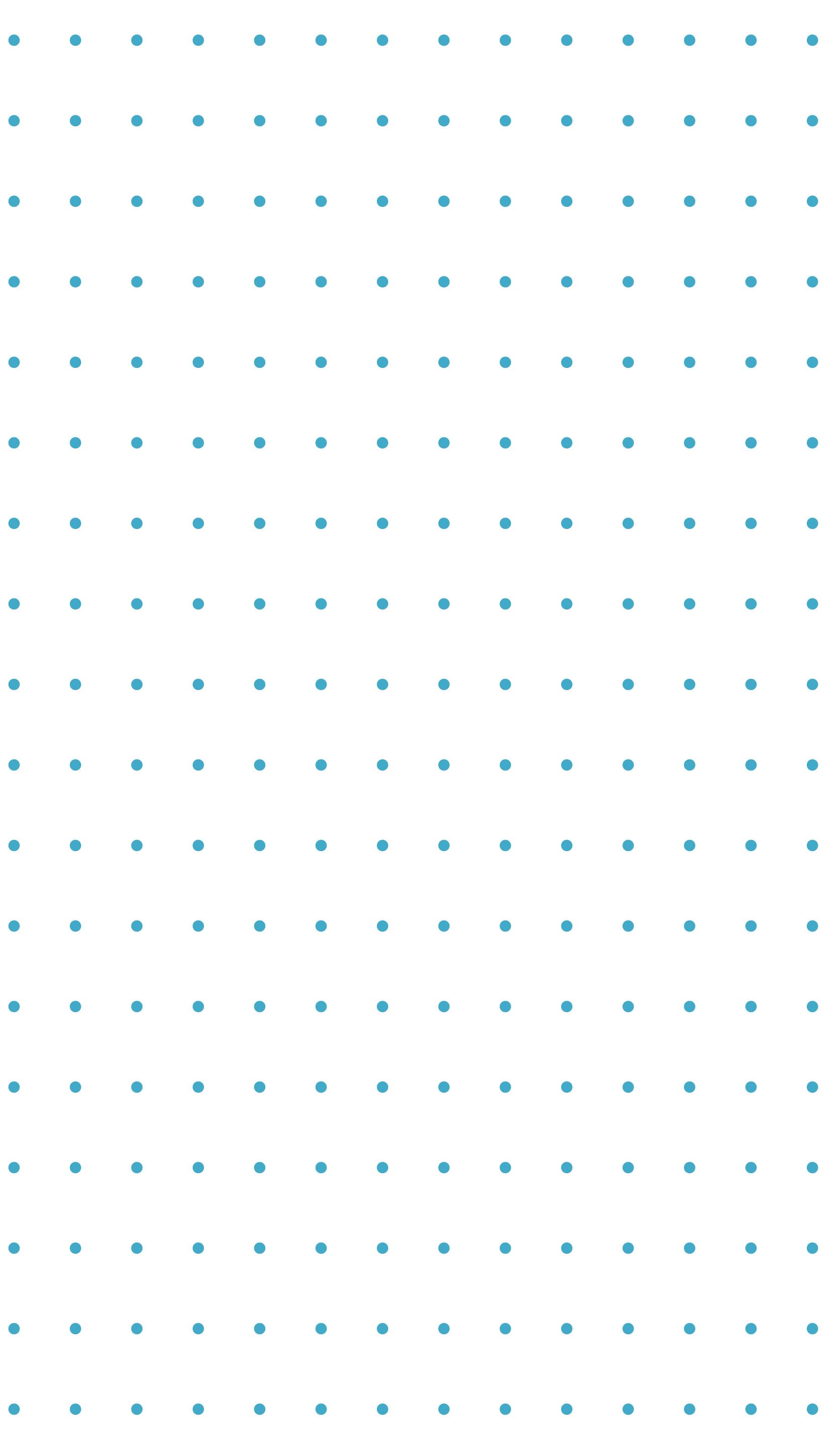
# Discrete Convolutions

Discrete convolutions apply to **array** or **matrix**-like data

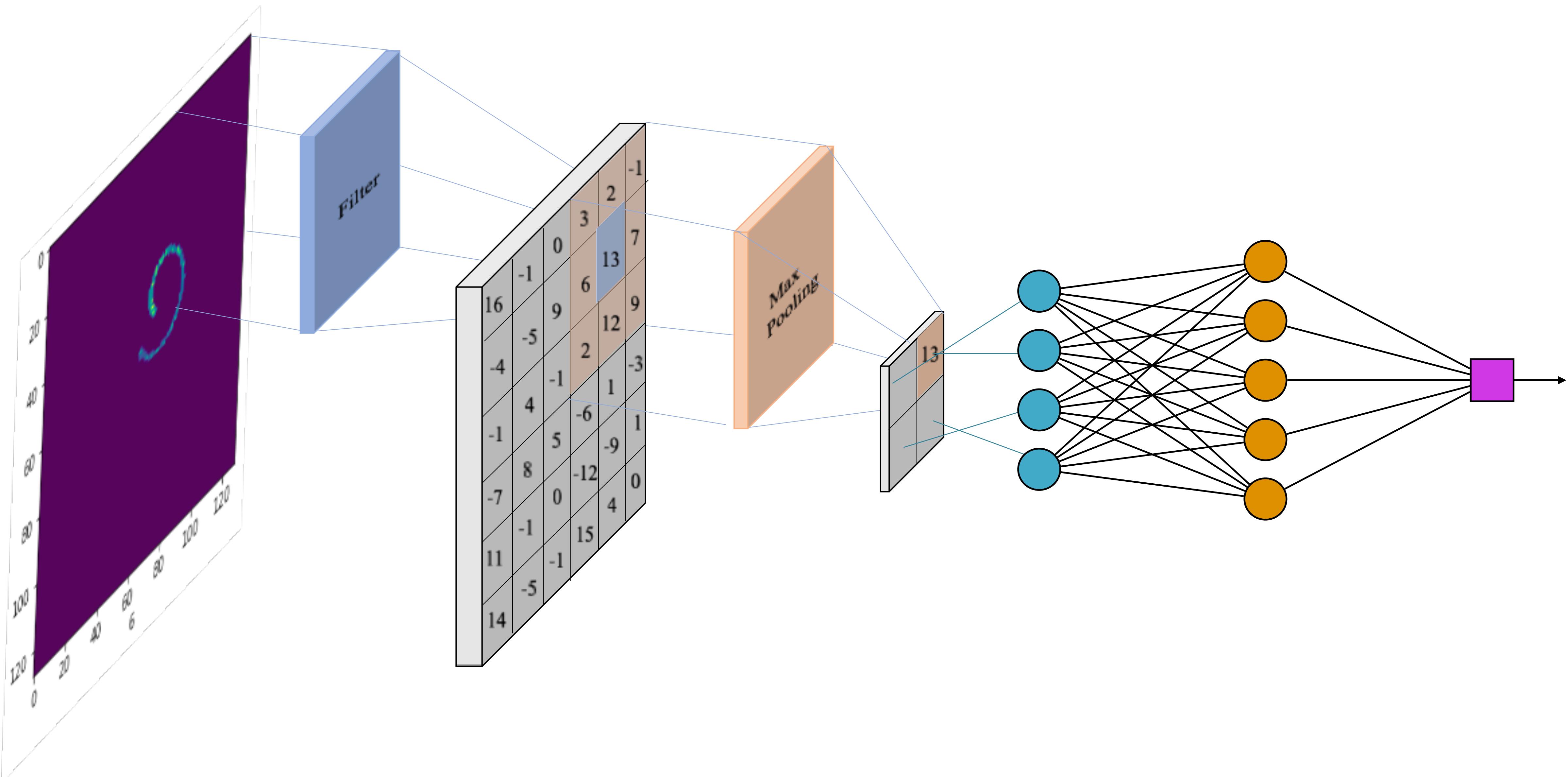
Discrete convolutions are matrix operations that can be used to apply **filters** to a matrix or array

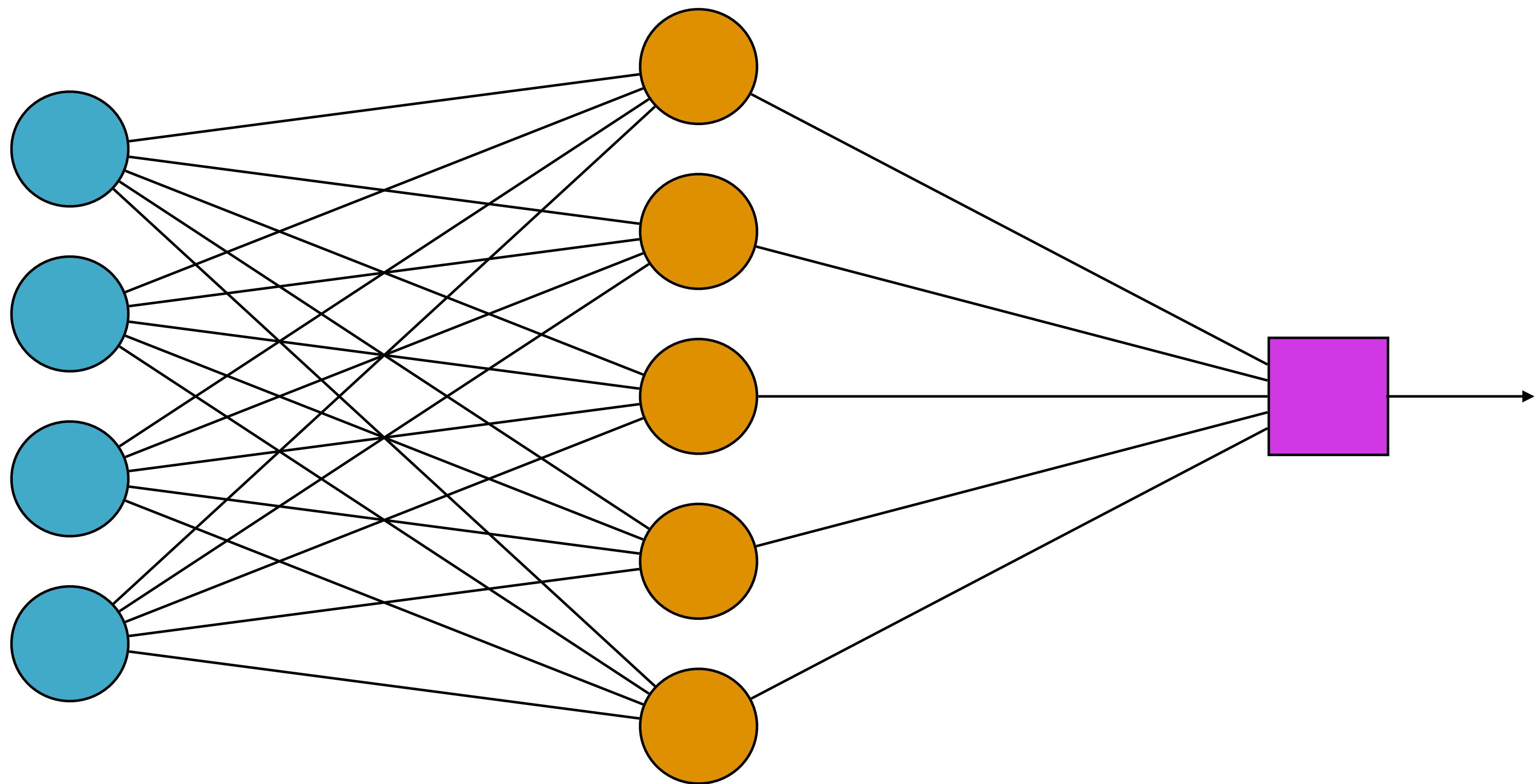
These filters can **extract features** or transform the input

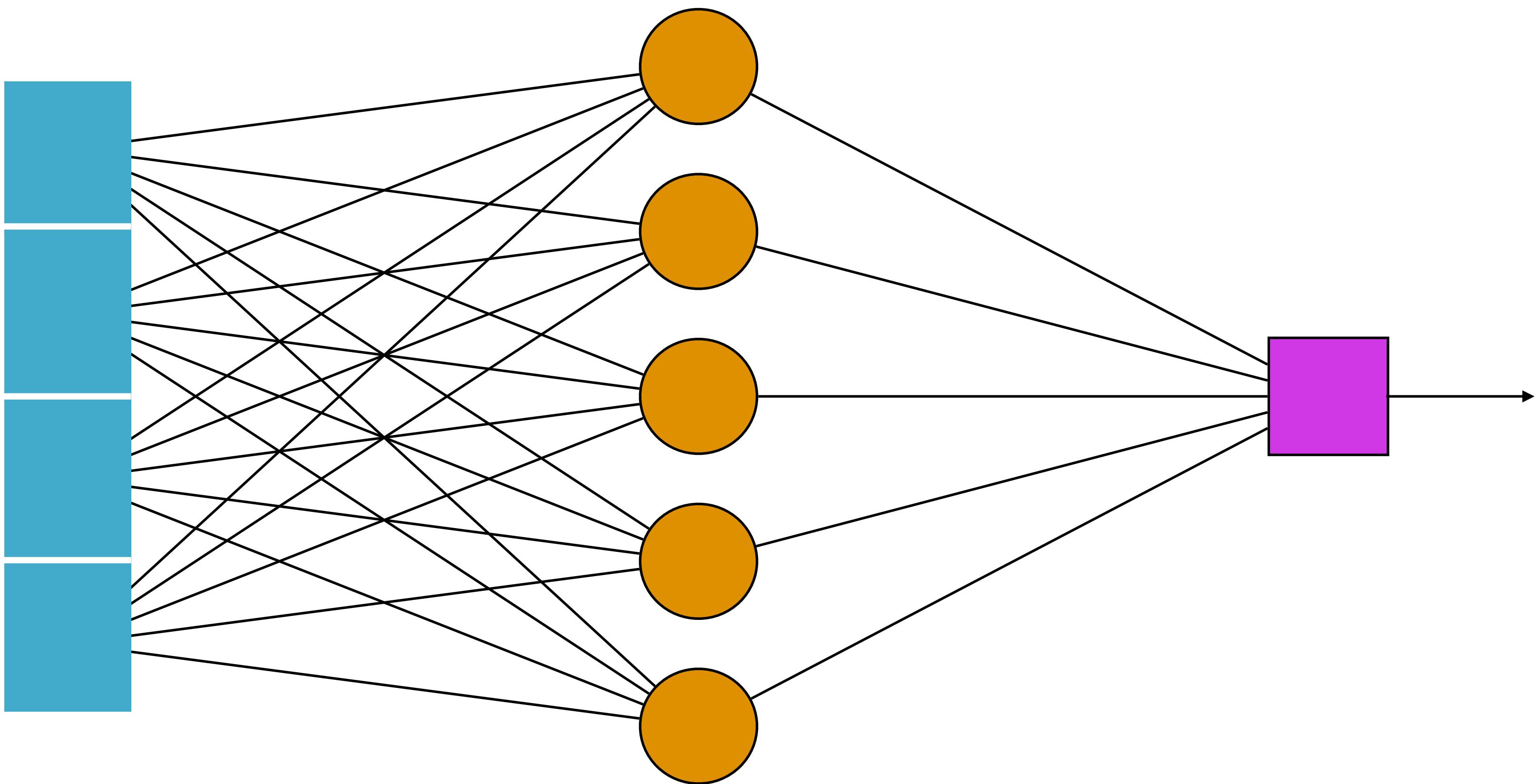
# CONVOLUTIONAL NEURAL NETWORKS: ARCHITECTURE AND TRAINING



# CONVOLUTIONAL NEURAL NETWORKS

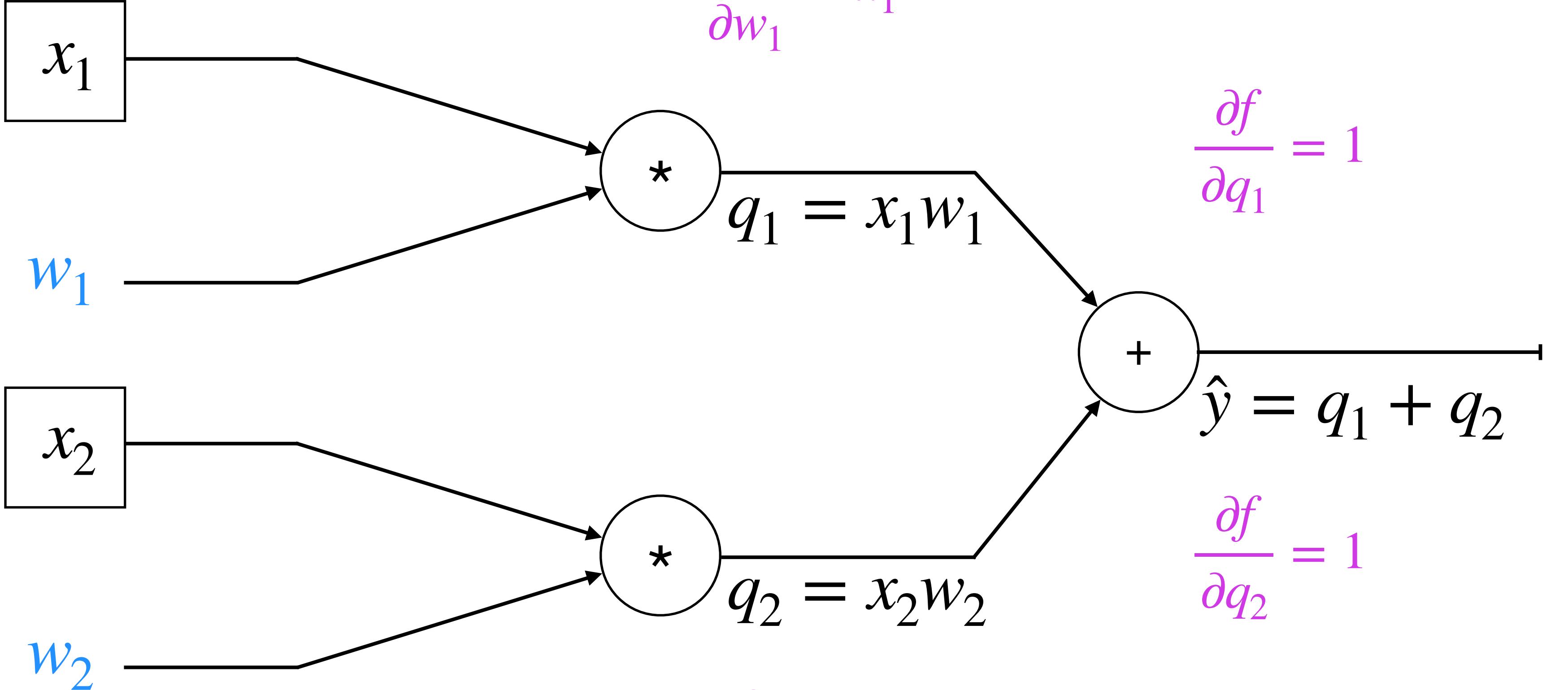






# BACKPROPAGATION

$$w_1 = w_1 + \eta * \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial q_1} \frac{\partial q_1}{\partial w_1}$$

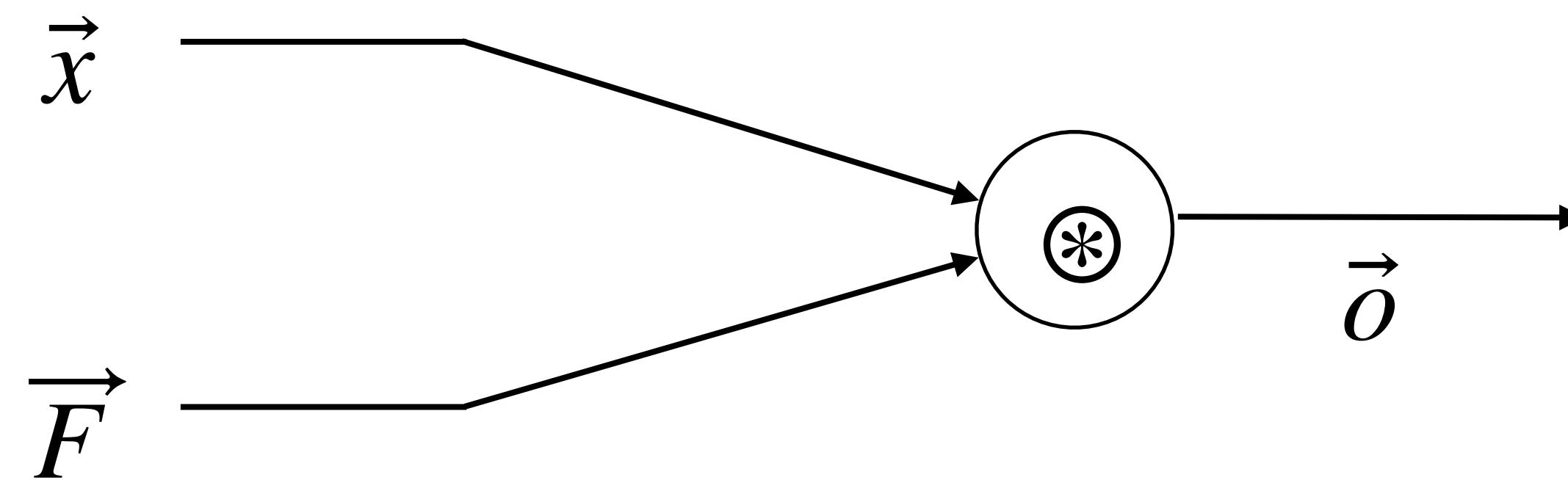


$$w_2 = w_2 + \eta * \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial q_1} \frac{\partial q_1}{\partial w_2}$$

Loss function  
 $J(w) = \hat{y} - y$

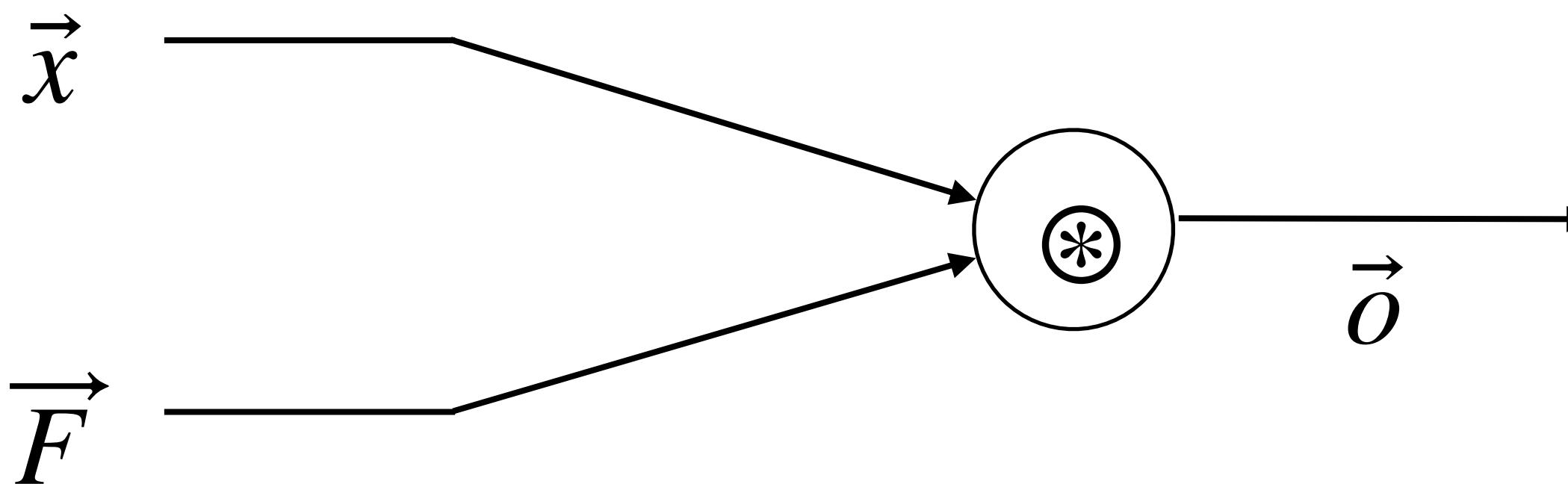
# BACKPROPAGATION

$$\vec{o} = \vec{x} \circledast \vec{F}$$



# BACKPROPAGATION

$$\vec{o} = \vec{x} \circledast \vec{F}$$



$$o[n] = (x \circledast F)[n] = \sum_{i=-\omega}^{\omega} x[i + n + \omega] * F[i + \omega]$$

$x_0$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$
-------	-------	-------	-------	-------	-------	-------	-------

$O_0$	$O_1$	$O_2$	$O_3$	$O_4$	$O_5$	$O_6$	$O_7$
-------	-------	-------	-------	-------	-------	-------	-------

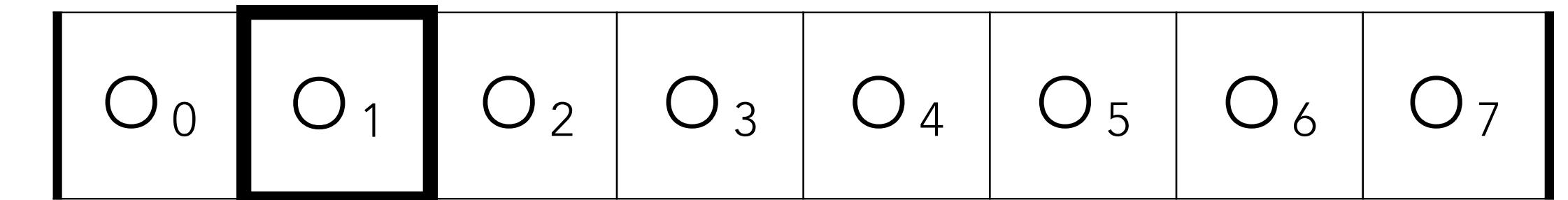
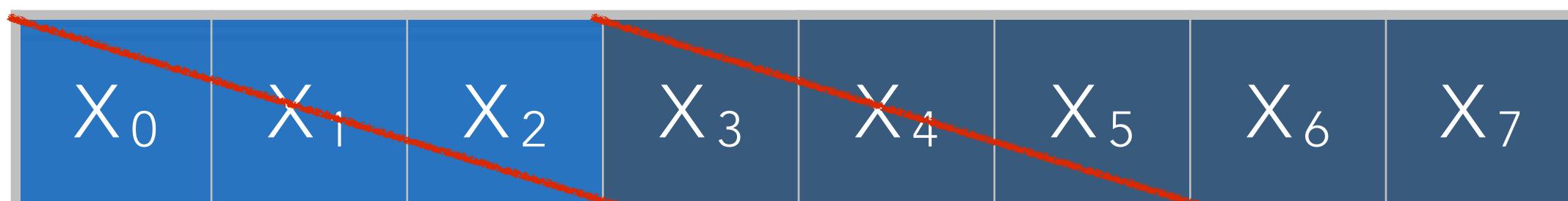
$F_1$	$F_2$	$F_3$
-------	-------	-------

# BACKPROPAGATION

$$F_i \leftarrow F_i - \eta \frac{\partial J}{\partial F_i}$$

$$\frac{\partial J}{\partial F_i} = \sum_{k=1}^M \frac{\partial J}{\partial o_k} \frac{\partial o_k}{\partial F_i}$$

$$o[n] = (x \circledast F)[n] = \sum_{i=-\omega}^{\omega} x[i + n + \omega] * F[i + \omega]$$

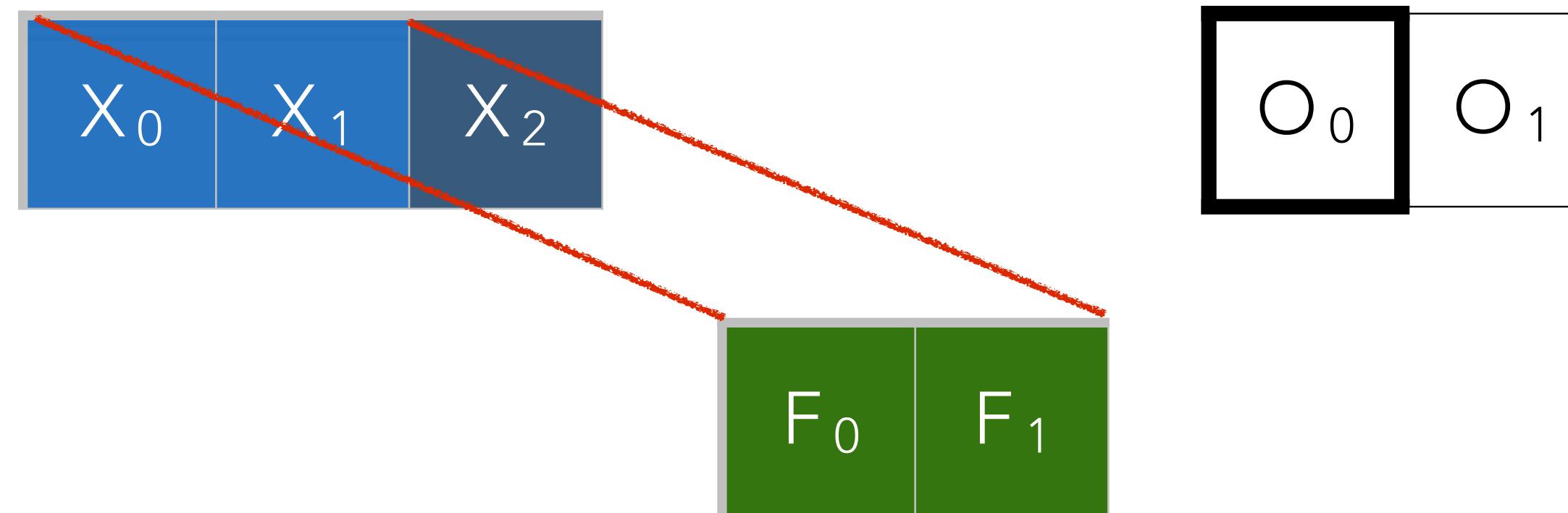


# BACKPROPAGATION

$$F_i \leftarrow F_i - \eta \frac{\partial J}{\partial F_i}$$

$$\frac{\partial J}{\partial F_i} = \sum_{k=1}^M \frac{\partial J}{\partial o_k} \frac{\partial o_k}{\partial F_i}$$

$$o[n] = (x \circledast F)[n] = \sum_{i=-\omega}^{\omega} x[i + n + \omega] * F[i + \omega]$$



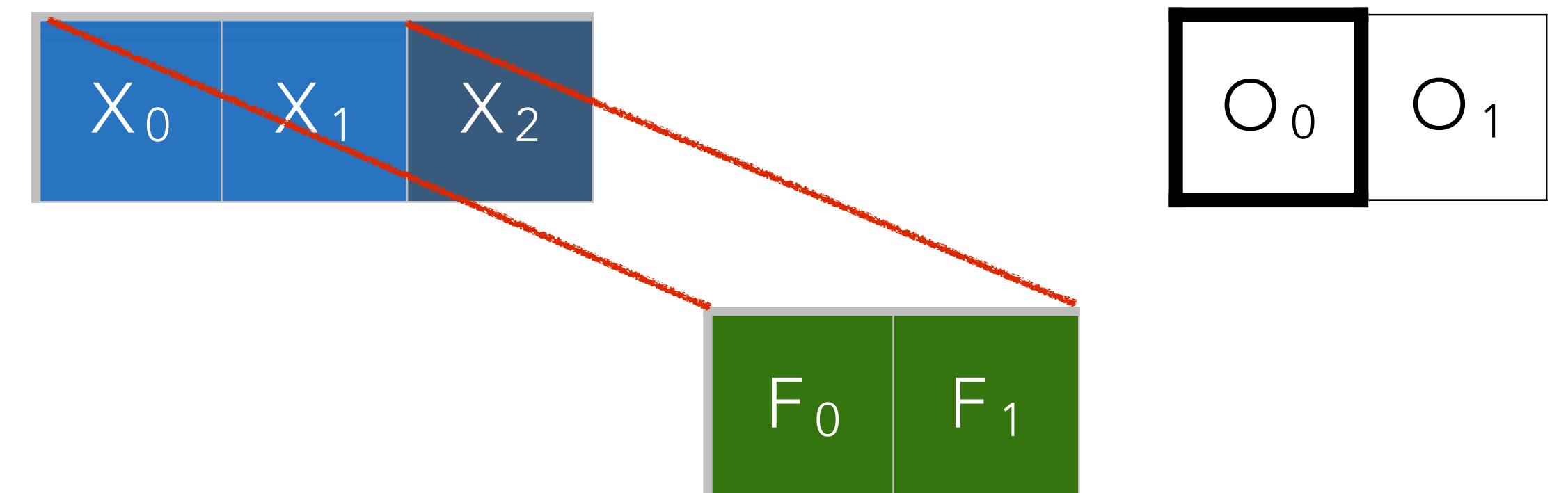
# BACKPROPAGATION

$$\frac{\partial J}{\partial F_1} = \frac{\partial J}{\partial o_0} \frac{\partial o_0}{\partial F_1} + \frac{\partial J}{\partial o_1} \frac{\partial o_1}{\partial F_1}$$

$$F_i \leftarrow F_i - \eta \frac{\partial J}{\partial F_i}$$

$$\frac{\partial J}{\partial F_i} = \sum_{k=1}^M \frac{\partial J}{\partial o_k} \frac{\partial o_k}{\partial F_i}$$

$$o[n] = (x \circledast F)[n] = \sum_{i=-\omega}^{\omega} x[i + n + \omega] * F[i + \omega]$$

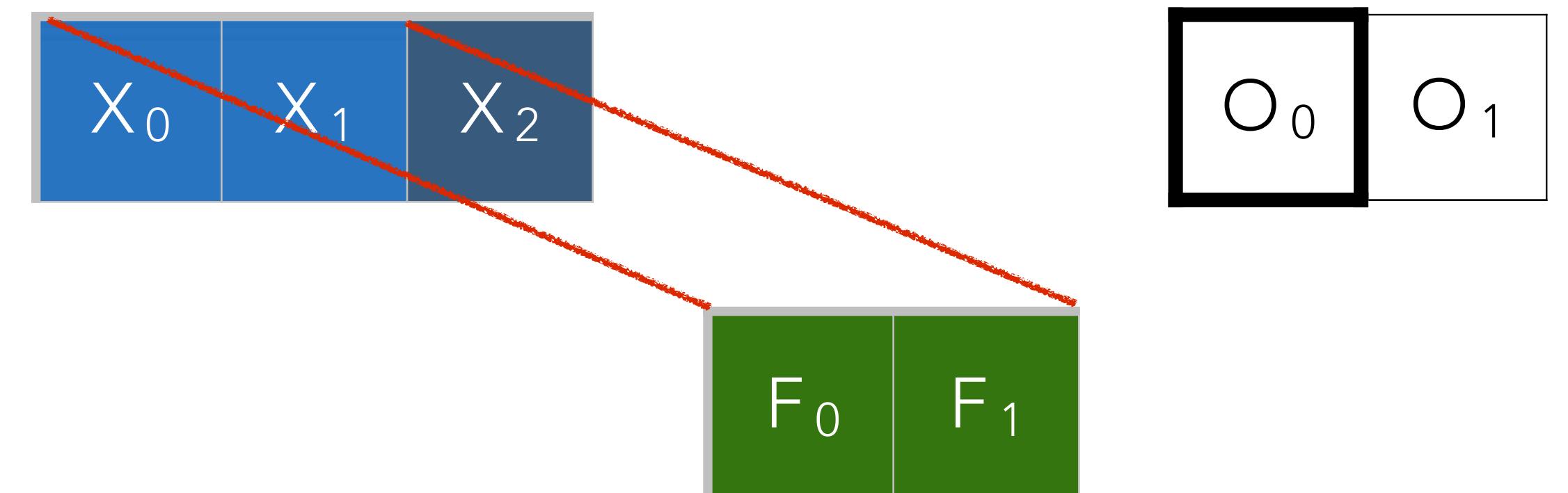


# BACKPROPAGATION

$$\frac{\partial J}{\partial F_1} = \frac{\partial J}{\partial o_0} \frac{\partial o_0}{\partial F_1} + \frac{\partial J}{\partial o_1} \frac{\partial o_1}{\partial F_1}$$

$$F_i \leftarrow F_i - \eta \frac{\partial J}{\partial F_i}$$
$$\frac{\partial J}{\partial F_i} = \sum_{k=1}^M \frac{\partial J}{\partial o_k} \frac{\partial o_k}{\partial F_i}$$

$$o[n] = (x \circledast F)[n] = \sum_{i=-\omega}^{\omega} x[i + n + \omega] * F[i + \omega]$$



# BACKPROPAGATION

$$\frac{\partial J}{\partial F_1} = \frac{\partial J}{\partial o_0} \frac{\partial o_0}{\partial F_1} + \frac{\partial J}{\partial o_1} \frac{\partial o_1}{\partial F_1}$$

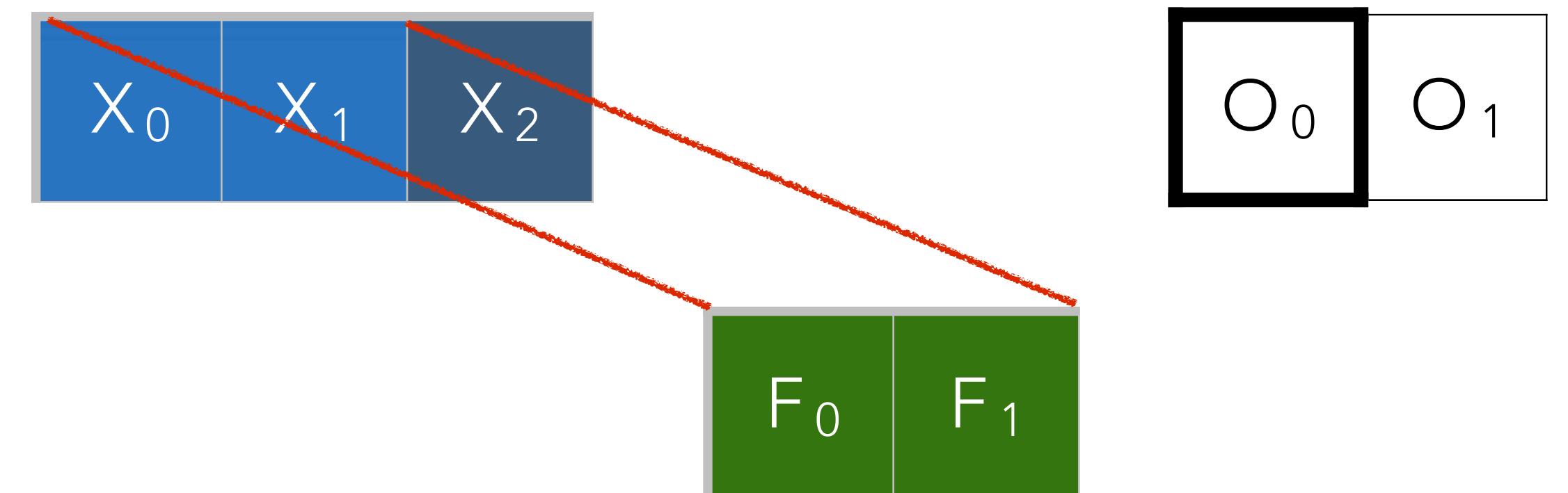
$$o_0 = F_0 x_0 + F_1 x_1$$

$$o_1 = F_0 x_1 + F_1 x_2$$

$$F_i \leftarrow F_i - \eta \frac{\partial J}{\partial F_i}$$

$$\frac{\partial J}{\partial F_i} = \sum_{k=1}^M \frac{\partial J}{\partial o_k} \frac{\partial o_k}{\partial F_i}$$

$$o[n] = (x \circledast F)[n] = \sum_{i=-\omega}^{\omega} x[i+n+\omega] * F[i+\omega]$$



# BACKPROPAGATION

$$\frac{\partial J}{\partial F_1} = \frac{\partial J}{\partial o_0} \frac{\partial o_0}{\partial F_1} + \frac{\partial J}{\partial o_1} \frac{\partial o_1}{\partial F_1}$$

$$o_0 = F_0 x_0 + F_1 x_1$$

$$o_1 = F_0 x_1 + F_1 x_2$$

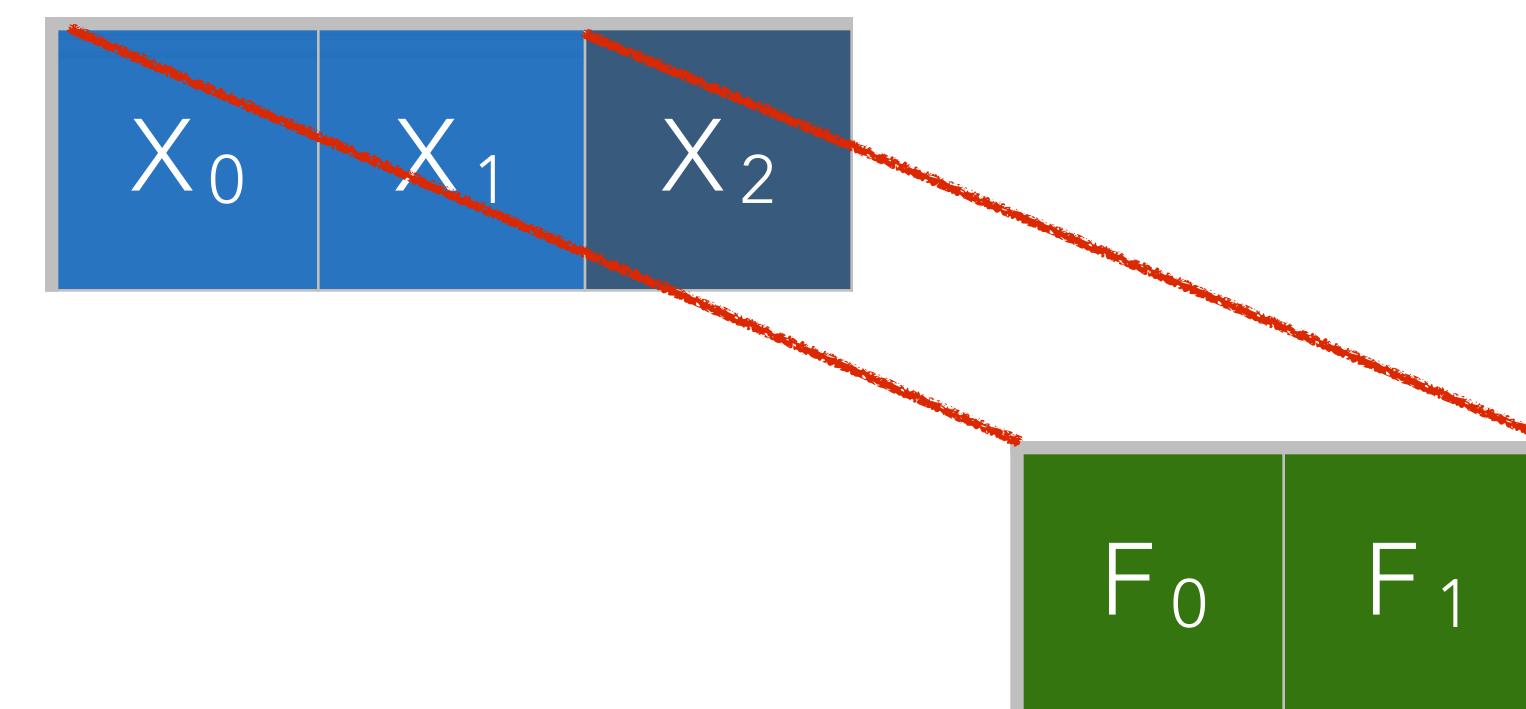
$$\frac{\partial o_0}{\partial F_1} = x_1$$

$$\frac{\partial o_1}{\partial F_1} = x_2$$

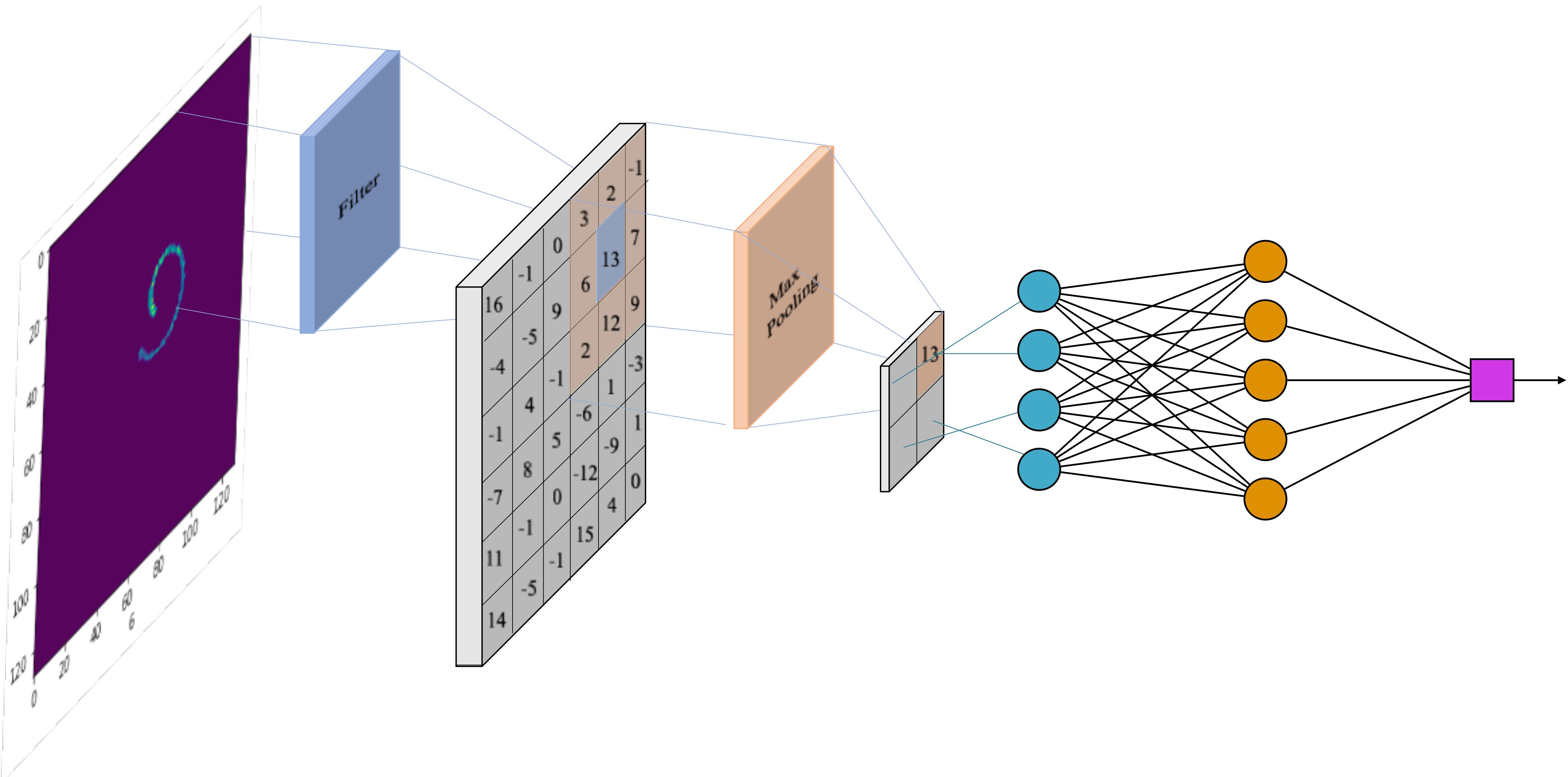
$$F_i \leftarrow F_i - \eta \frac{\partial J}{\partial F_i}$$

$$\frac{\partial J}{\partial F_i} = \sum_{k=1}^M \frac{\partial J}{\partial o_k} \frac{\partial o_k}{\partial F_i}$$

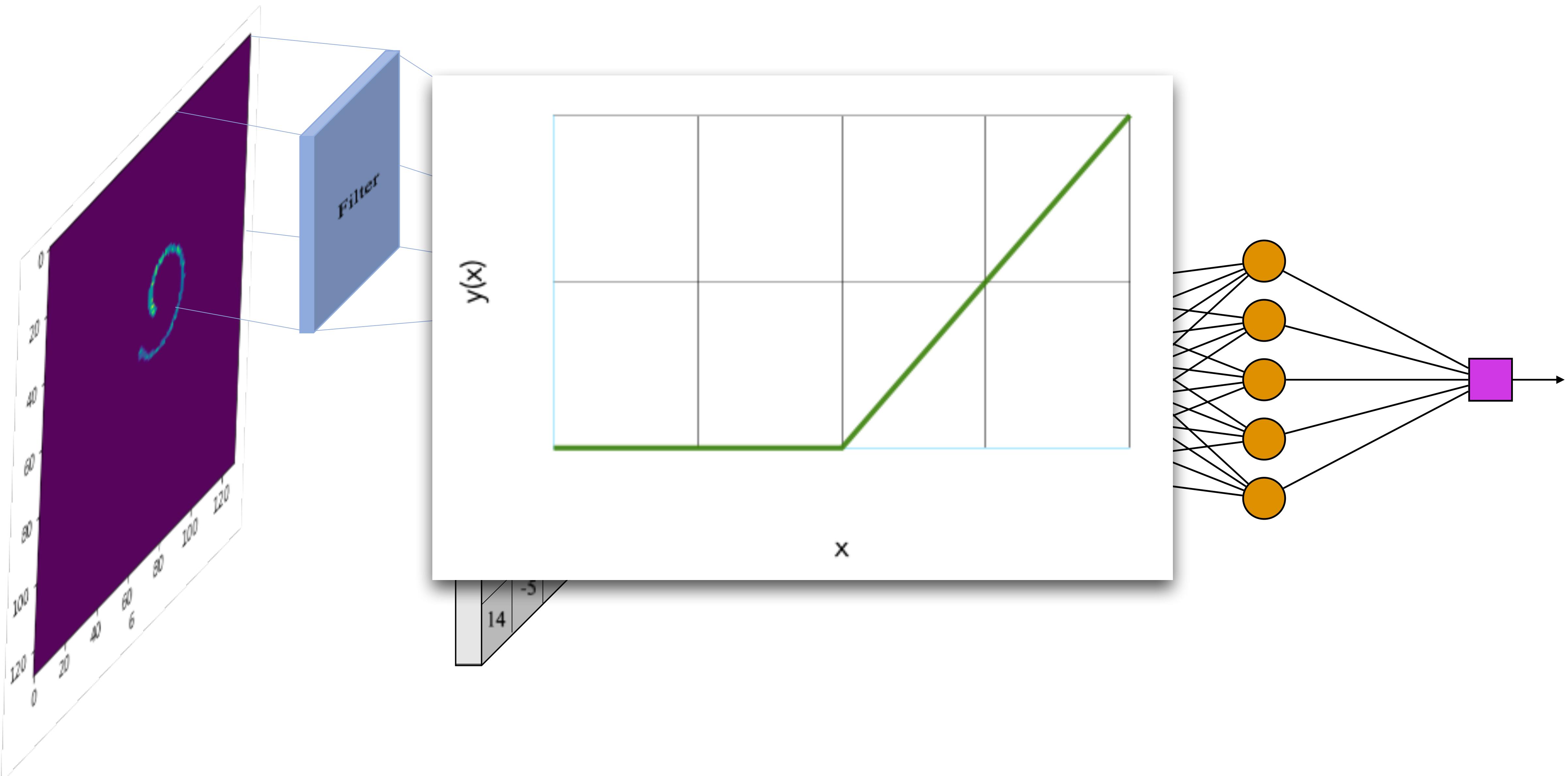
$$o[n] = (x \circledast F)[n] = \sum_{i=-\omega}^{\omega} x[i+n+\omega] * F[i+\omega]$$



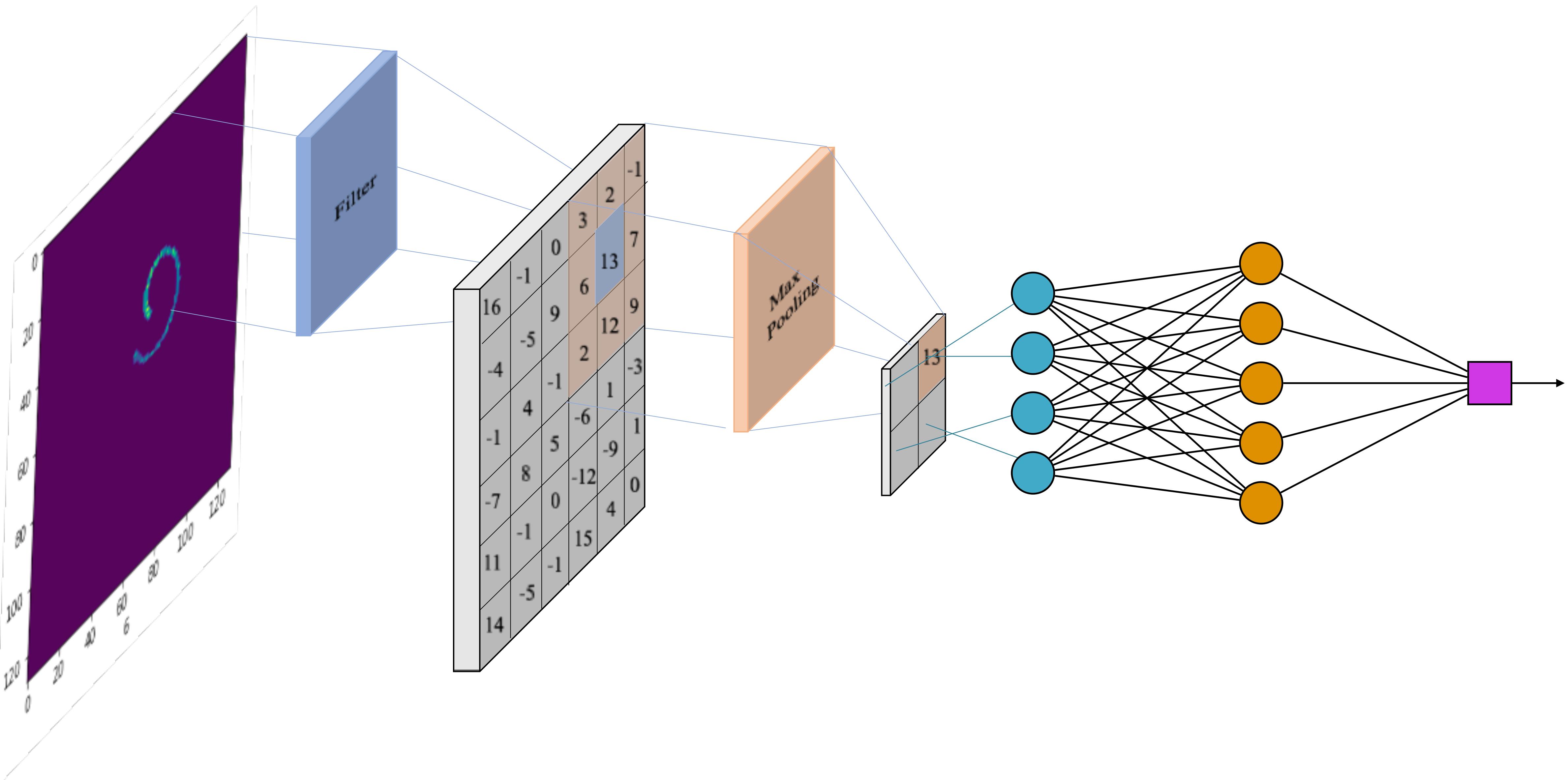
# CONVOLUTIONAL NEURAL NETWORKS



# CONVOLUTIONAL NEURAL NETWORKS



# CONVOLUTIONAL NEURAL NETWORKS



# POOLING

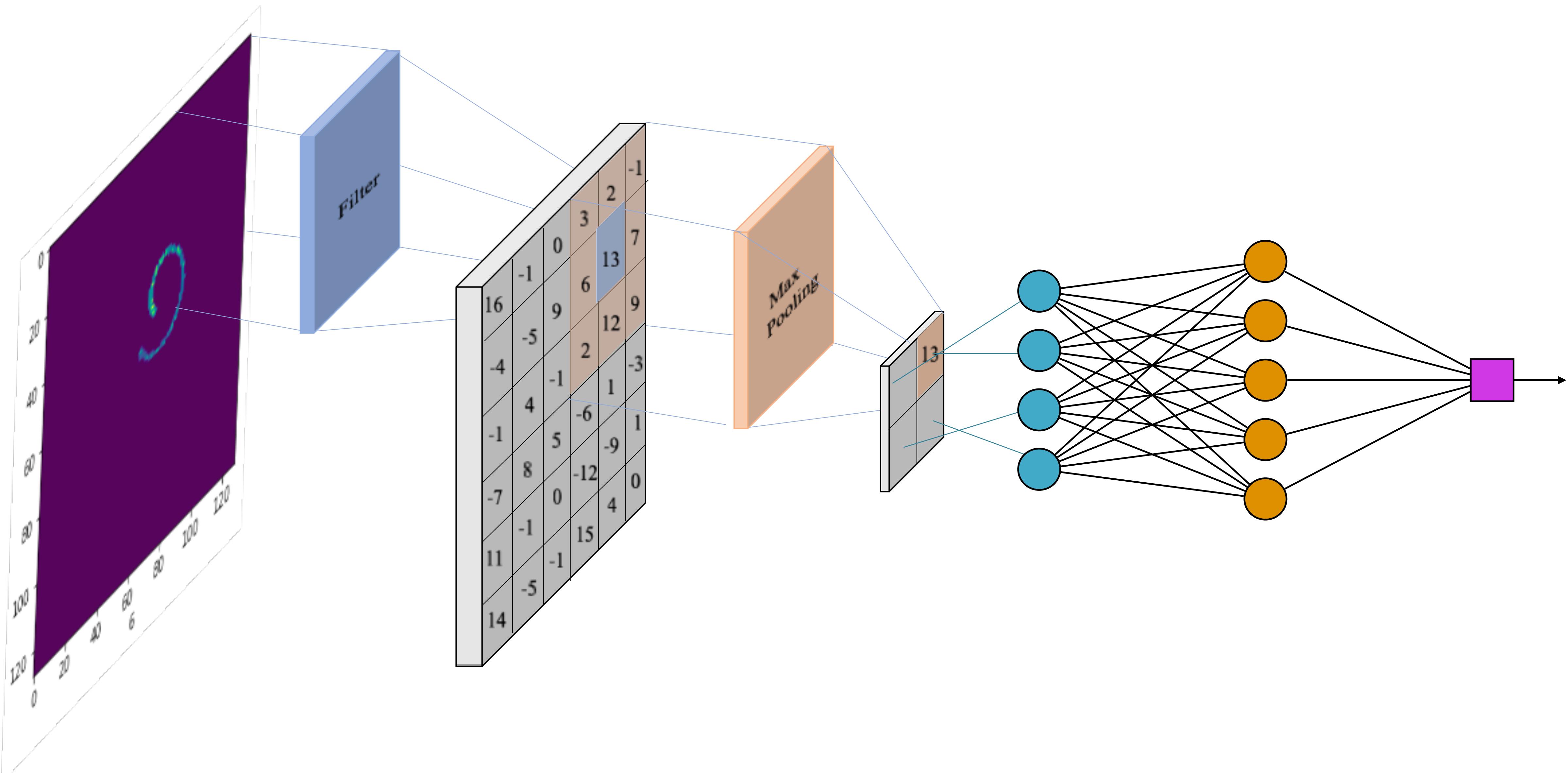
1	1	2	4
5	6	9	3
3	2	4	4
1	2	0	7

max pool with 2x2 filters  
and stride 2

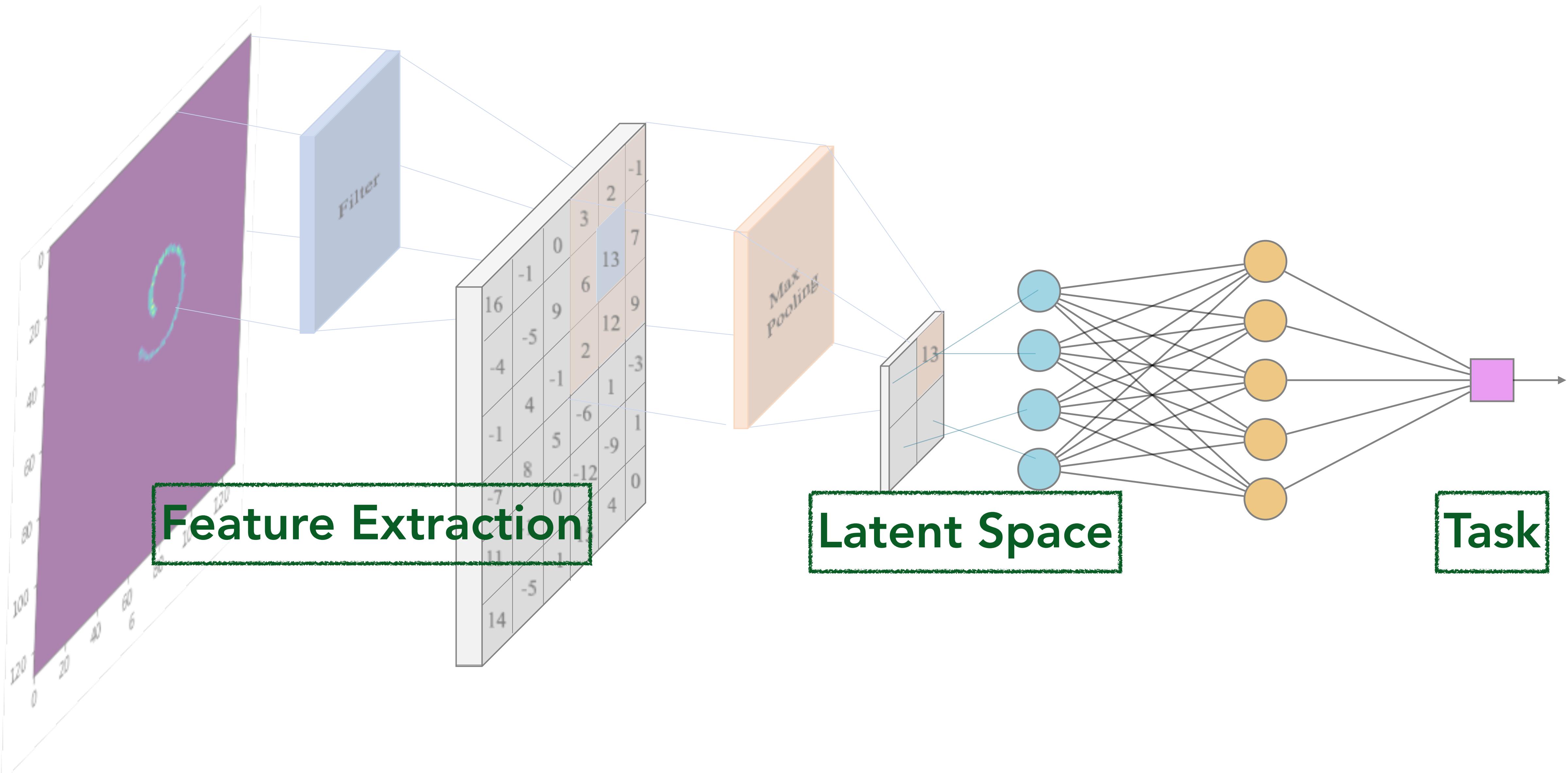


6	9
3	7

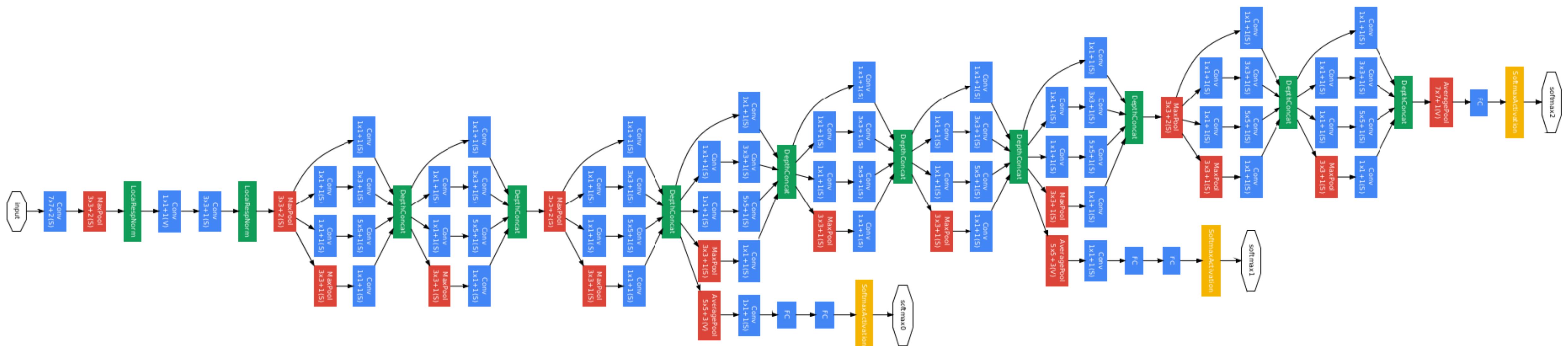
# BACKPROPOGATION



# CONVOLUTIONAL NEURAL NETWORKS

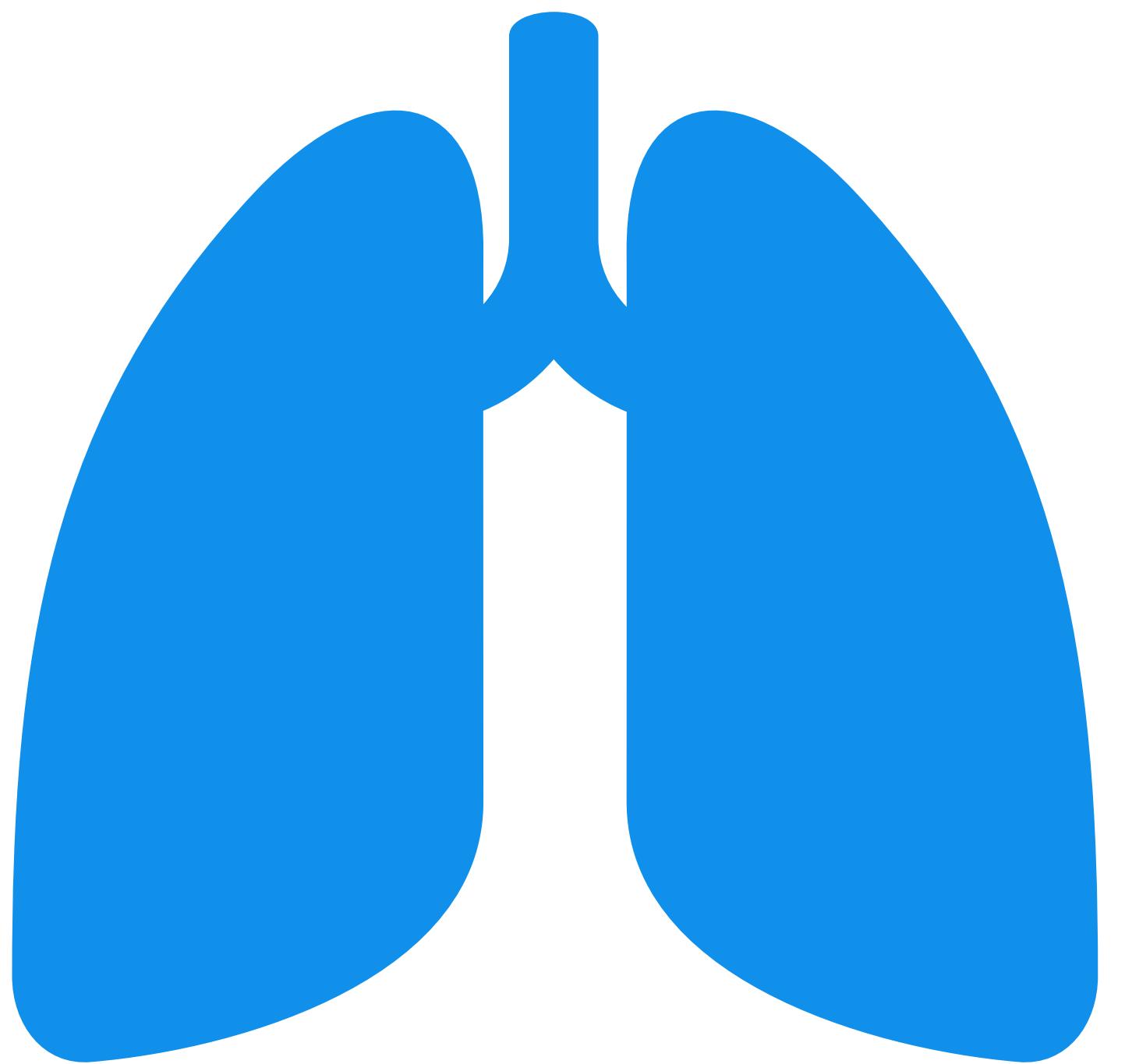


# CONVOLUTIONAL NEURAL NETWORKS



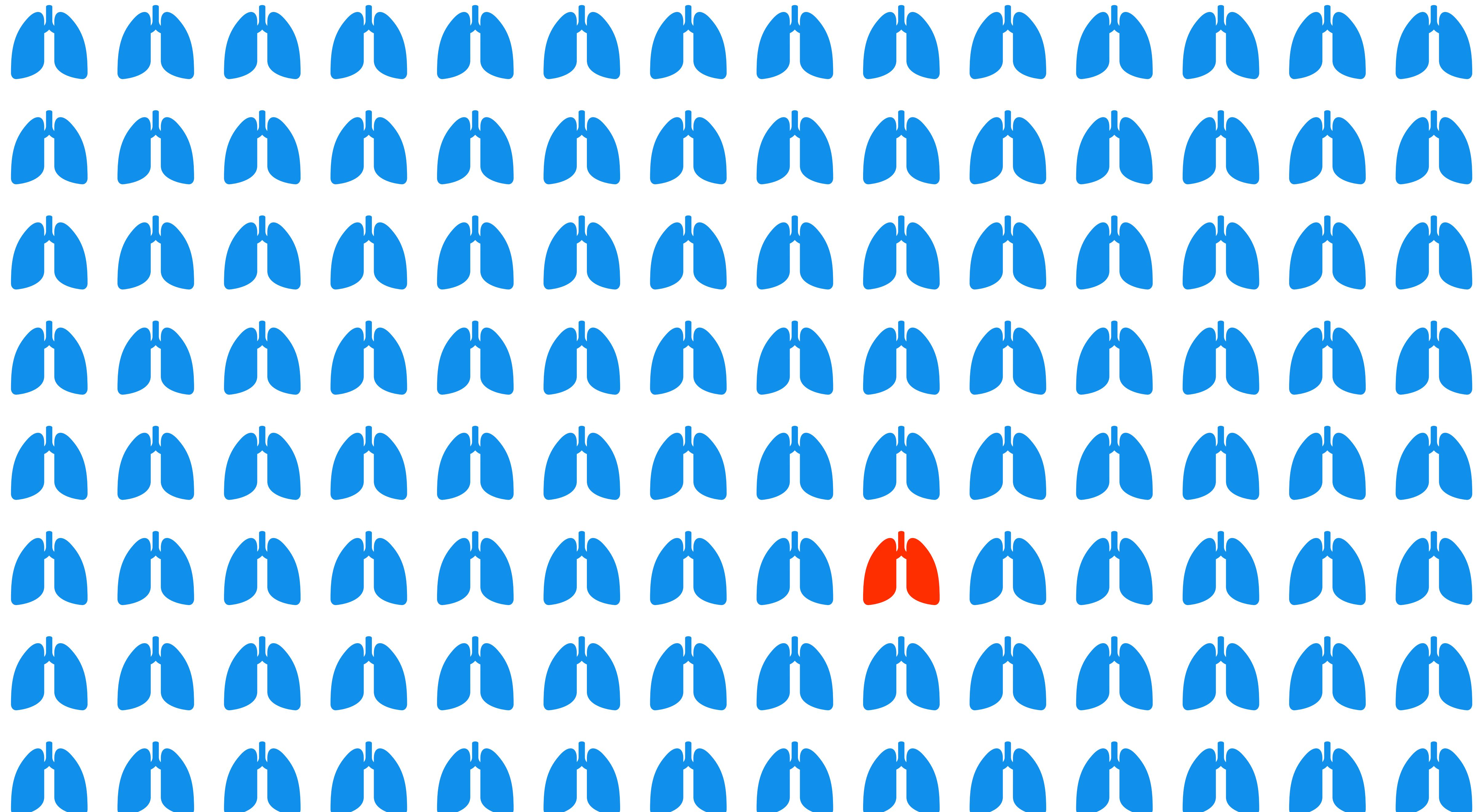
**Application 2:** Can we use machine learning to **accurately** classify events in detectors?

**Metrics**



Detect Lung Cancer

99% Accuracy



# PREDICTED

		Proton	Not Proton
		Proton	Not Proton
TRUE	Proton	TRUE POSITIVE (TP)	FALSE NEGATIVE (FN)
	Not Proton	FALSE POSITIVE (FP)	TRUE NEGATIVE (TN)

PREDICTED		
	Proton	Not Proton
TRUE	TRUE POSITIVE (TP)	FALSE NEGATIVE (FN)
Not Proton	FALSE POSITIVE (FP)	TRUE NEGATIVE (TN)

$$\text{accuracy} = \frac{TP + TN}{TP + FN + FP + TN}$$

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

$$F_1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

# PREDICTED

Proton

Not Proton

TRUE  
Proton

TRUE  
POSITIVE  
(TP)

TRUE  
Not Proton

TRUE  
NEGATIVE  
(TN)

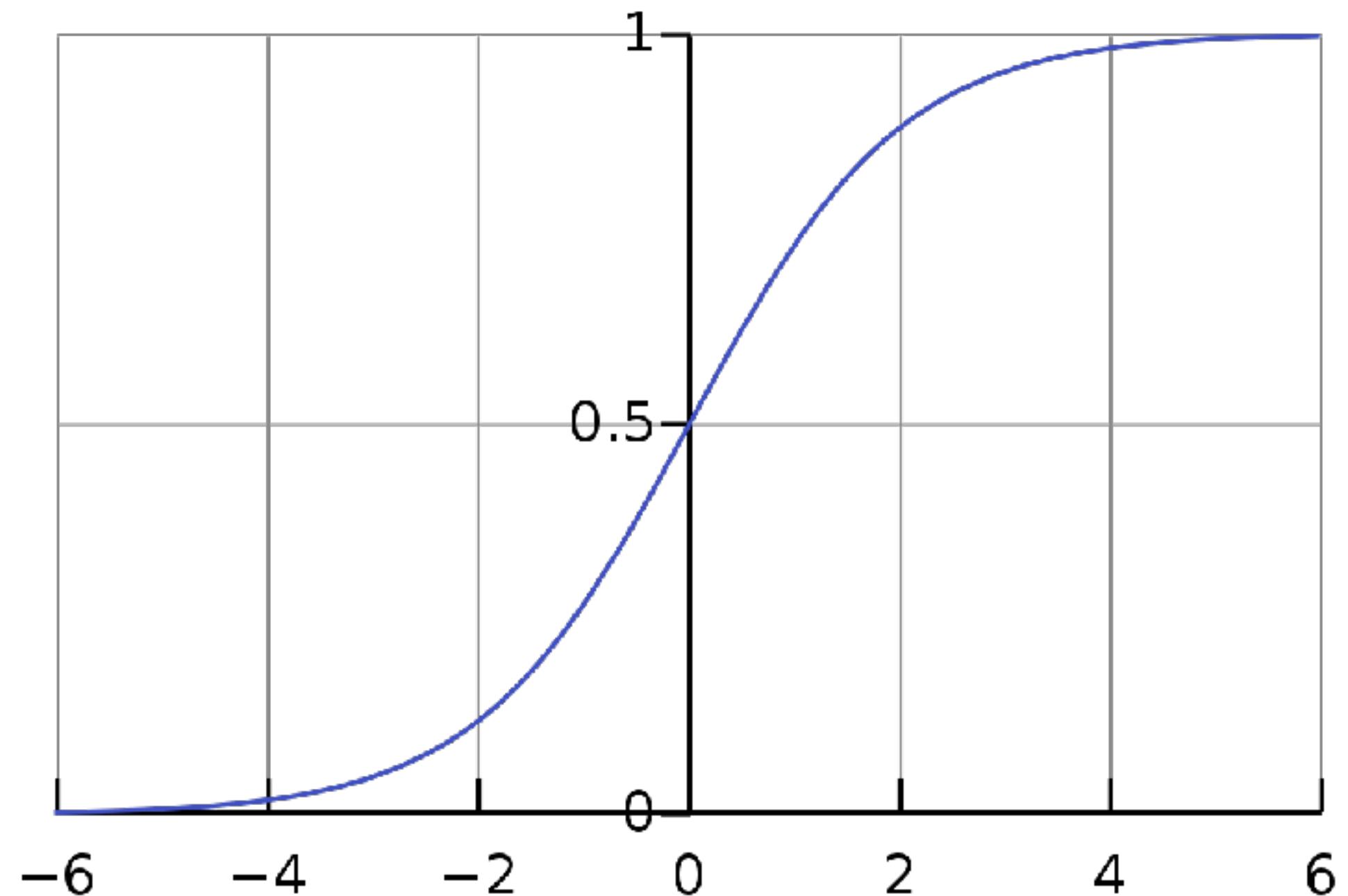
# PERFECT MODEL

# PREDICTED

		Proton	Not Proton
		Proton	Not Proton
TRUE	Proton	TRUE POSITIVE (TP)	FALSE NEGATIVE (FN)
	Not Proton	FALSE POSITIVE (FP)	TRUE NEGATIVE (TN)

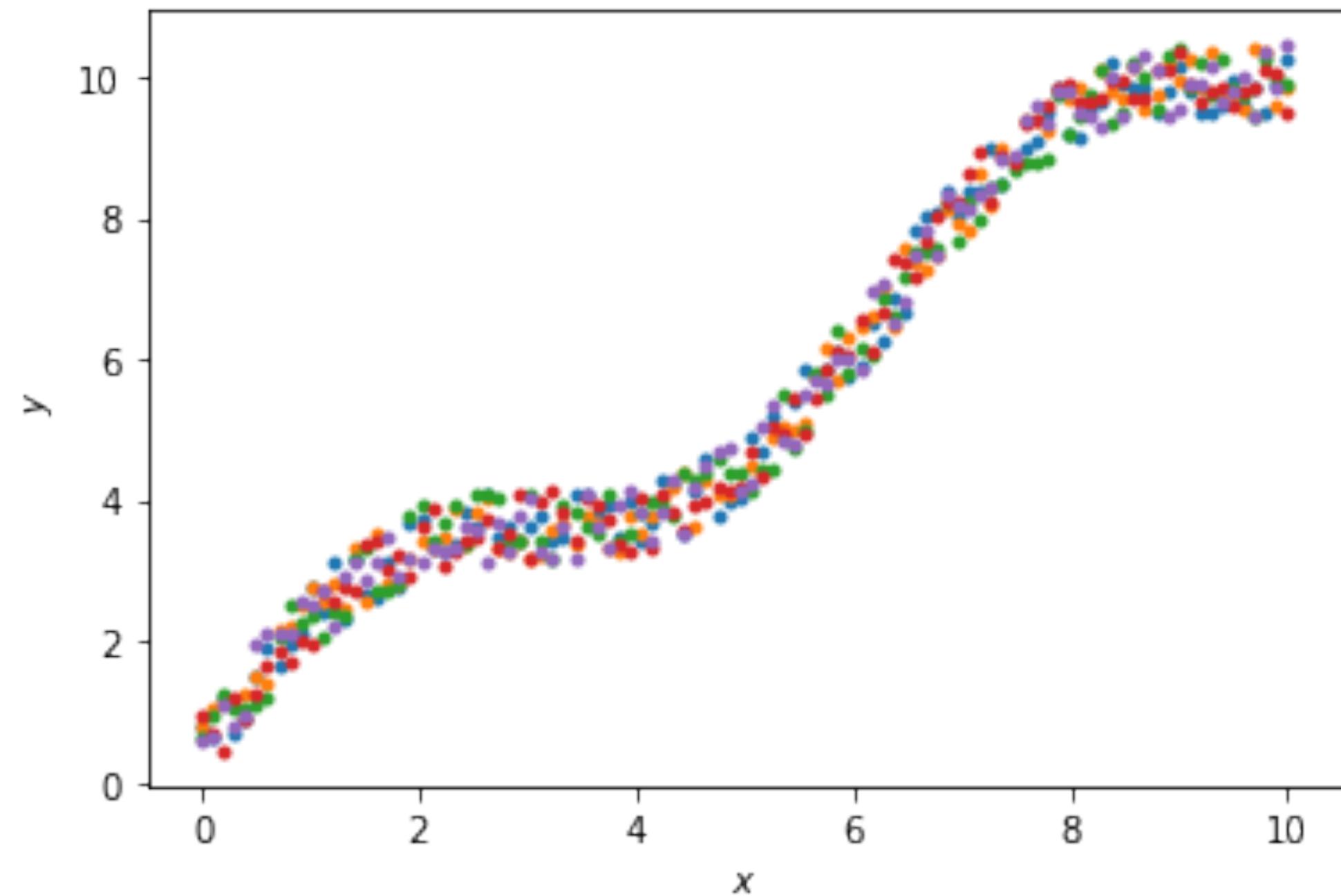
What does this matrix look like for multi-class classification?

# MULTI-CLASS CLASSIFICATION



$$\text{Softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^C e^{x_j}}$$

# LOSS FUNCTIONS



## Loss function

Mean squared error  
(MSE)

Mean absolute error  
(MAE)

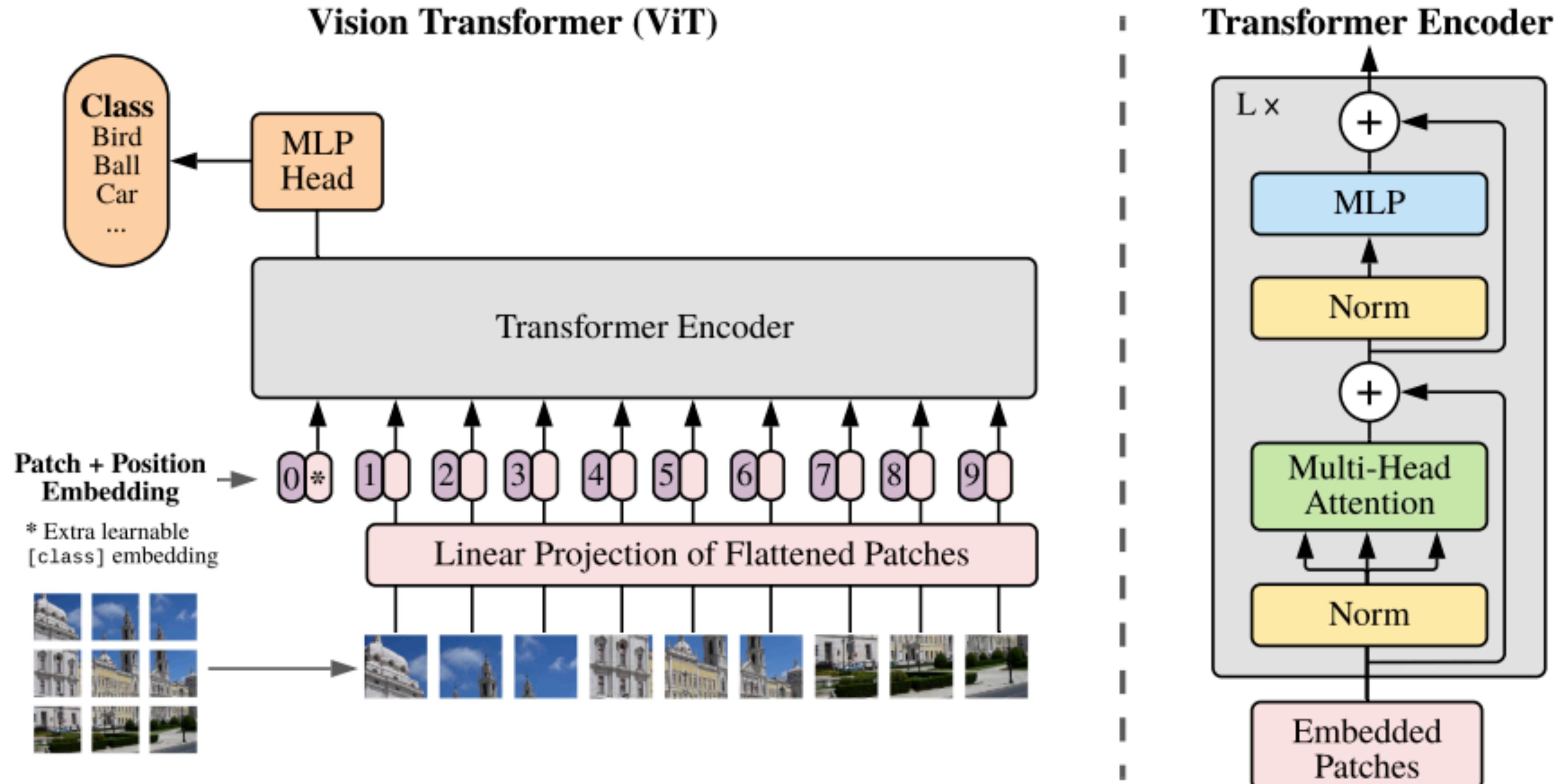
Cross entropy  
(CE)

$$J(w) = \frac{1}{N} \sum_{i=0}^N (\hat{y}_i - y_i)^2$$

$$J(w) = \frac{1}{N} \sum_{i=0}^N |\hat{y}_i - y_i|$$

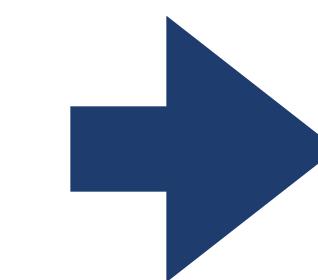
$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_{i=1}^N \sum_{j=1}^C y_{ij} \cdot \log(\hat{y}_{ij})$$

# Another approach:



# Visual Transformer

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



Patches

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

Linear embedding with position encoding

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

# Visual Transformer

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

$X =$

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

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

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

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

Patches

Linear embedding with position

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

# Visual Transformer

$$X = \begin{array}{c} \text{[matrix visualization]} \\ \begin{matrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 3 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{matrix} \end{array} N \times E$$

A 10x10 grid of black squares with white outlines. The first seven columns contain white symbols representing the Chinese character '人' (person), arranged in a pattern that suggests a group or team. The eighth column is empty, and the ninth and tenth columns are also empty.

# Visual Transformer

**X =**

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

$$Q = XW_Q$$

$$K = XW_K$$

$$V = XW_V$$

$$W_Q =$$

$$W_K =$$

A 10x10 grid of white shapes on a dark background. The shapes are arranged in a pattern where the first four columns contain a white semi-circle at the top and a white circle at the bottom, while the last six columns are empty. This pattern repeats five times vertically across the grid.

$$W_V =$$

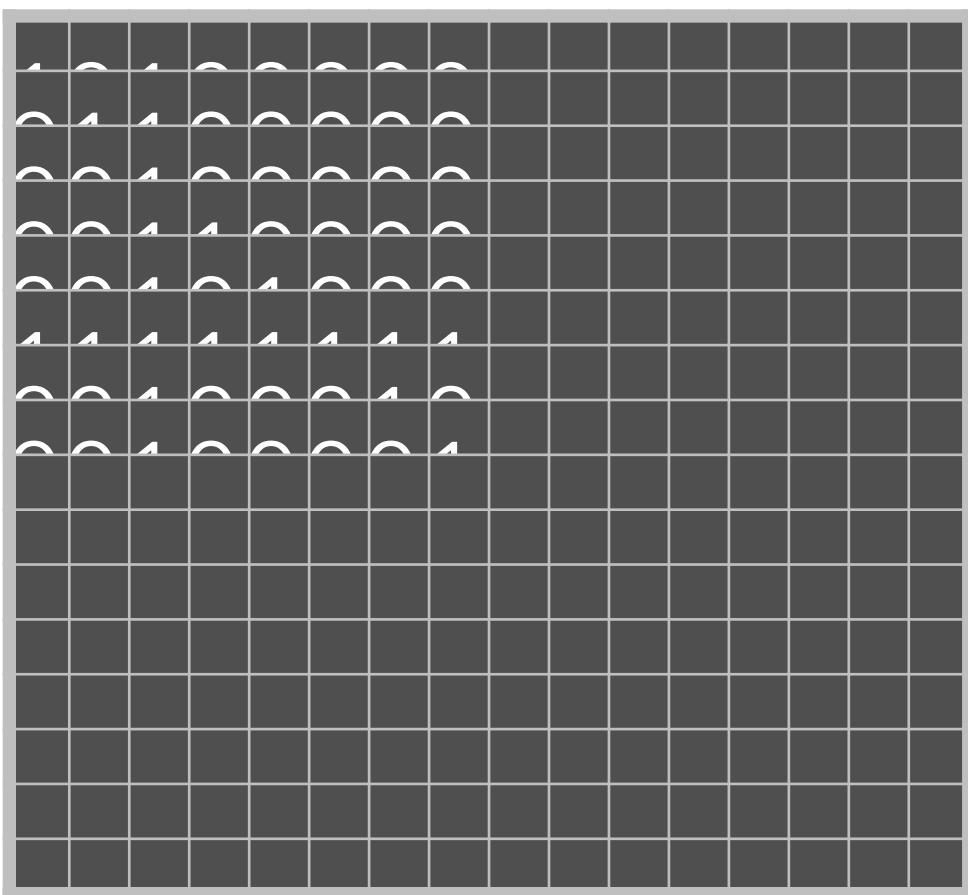
A 10x10 grid of white shapes on a black background. The first seven columns contain a repeating pattern of white shapes: a semi-circle at the top, followed by a circle, a semi-circle at the bottom, and a circle. The last three columns are empty.

# Visual Transformer

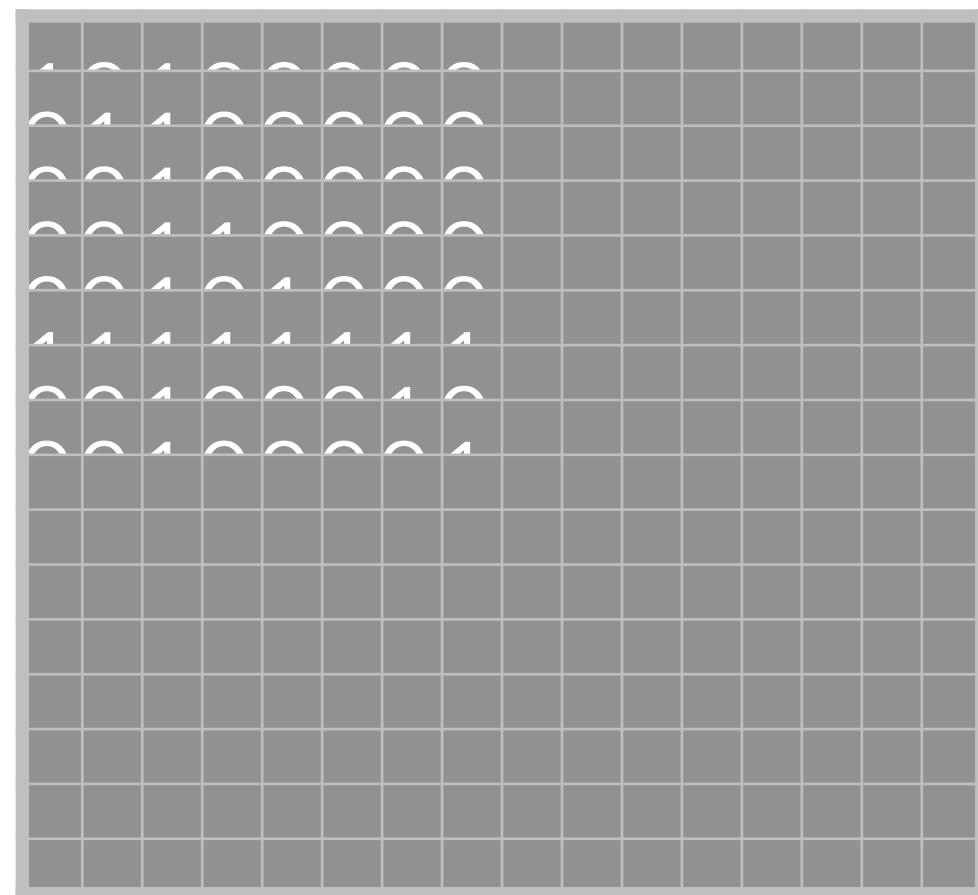
$X =$

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

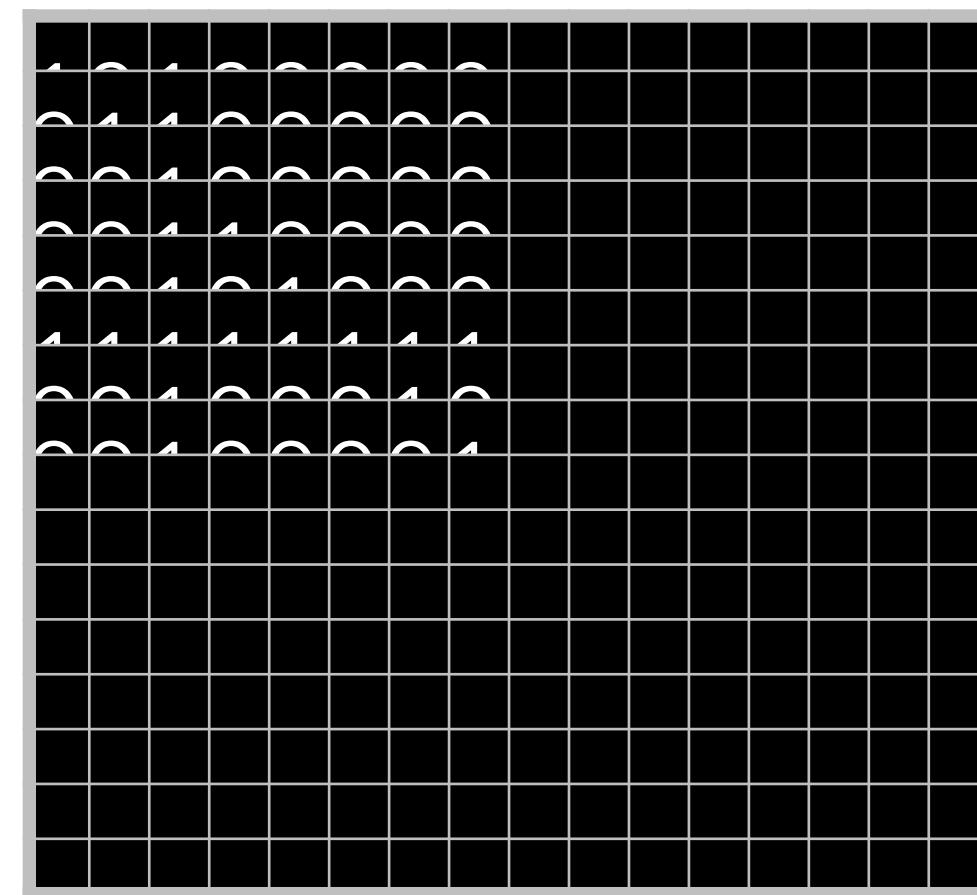
$W_Q =$



$W_K =$



$W_V =$



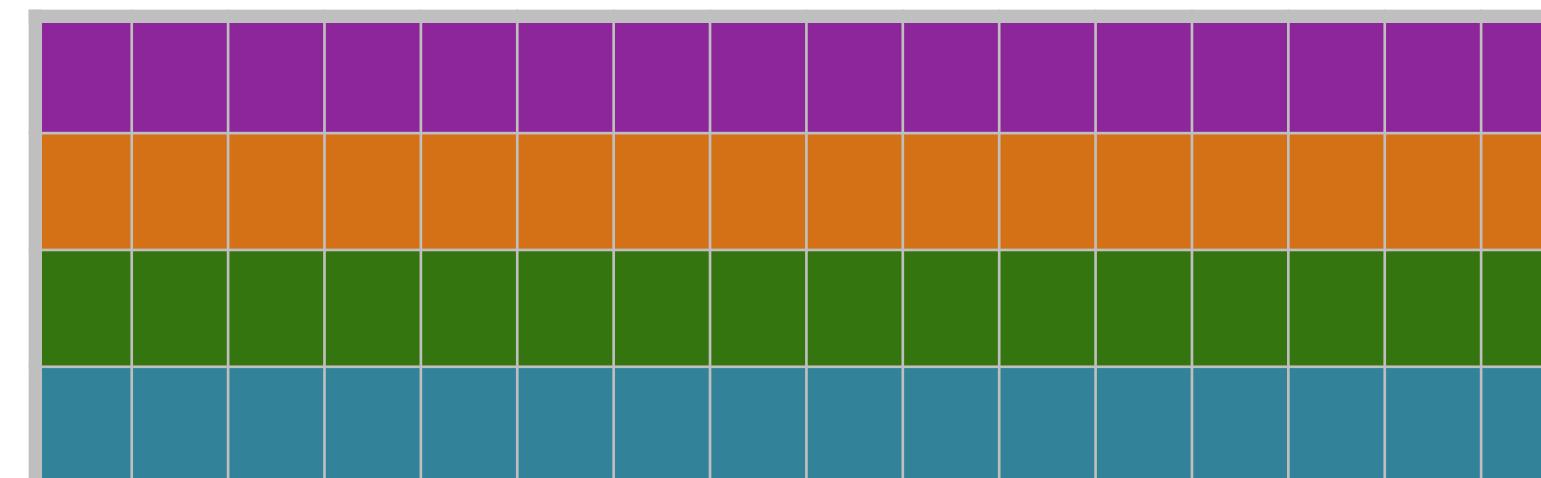
$$Q = XW_Q \quad (N \times E)$$

$$K = XW_K \quad (N \times E)$$

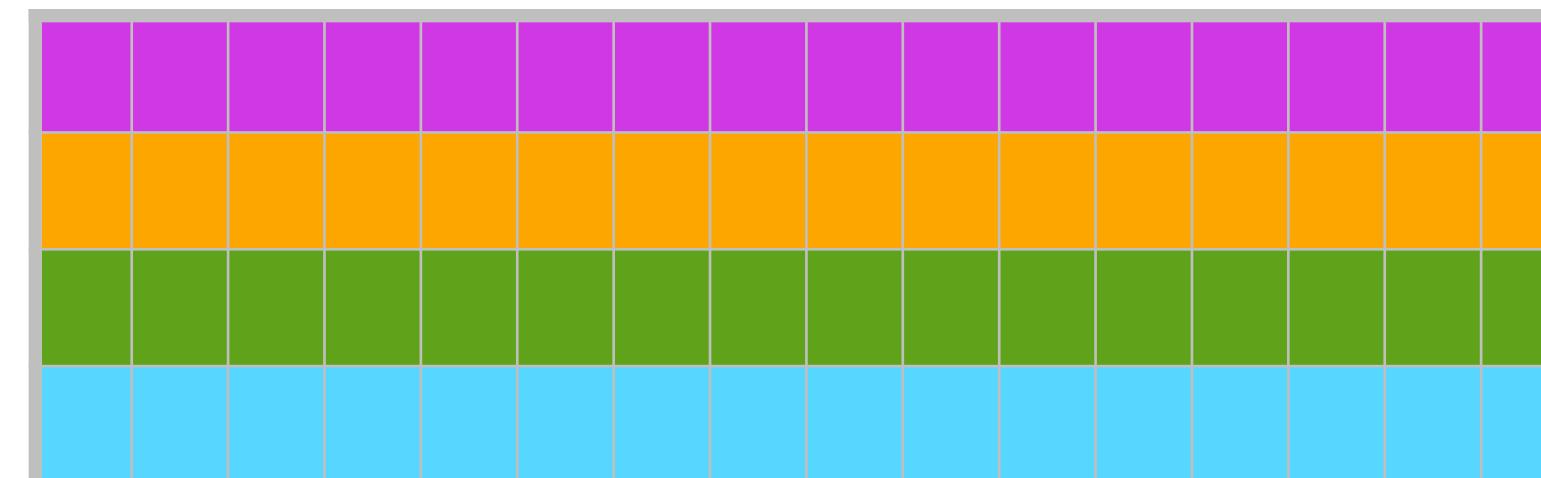
$$V = XW_V \quad (N \times E)$$

# Visual Transformer

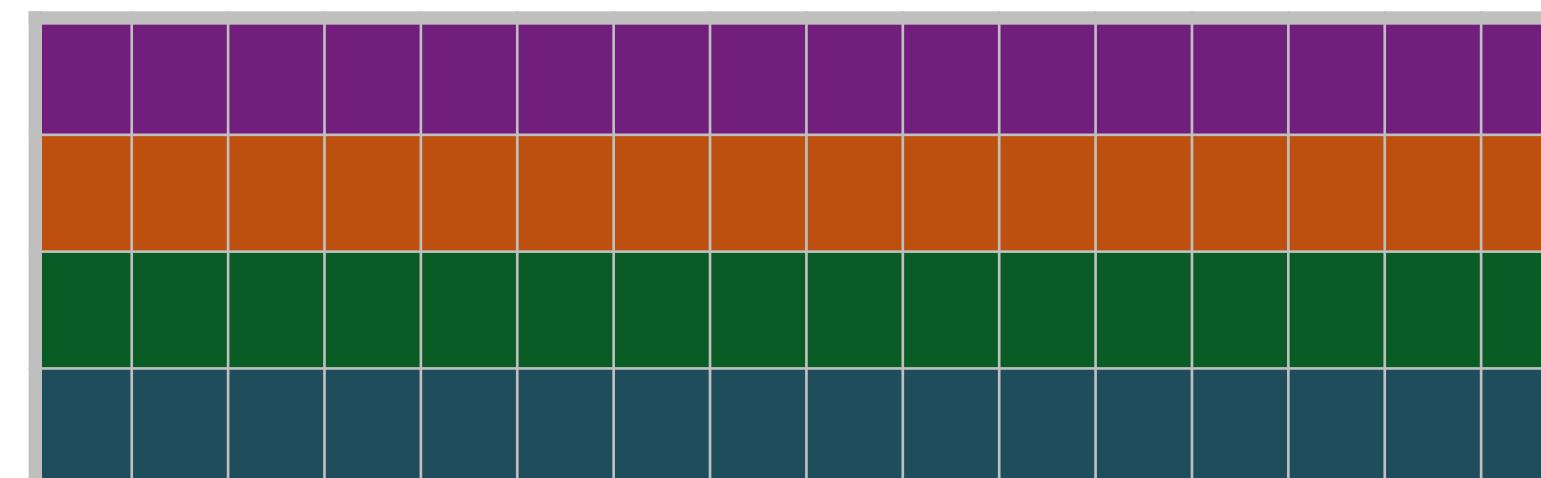
$$Q = XW_Q =$$



$$K = XW_K =$$



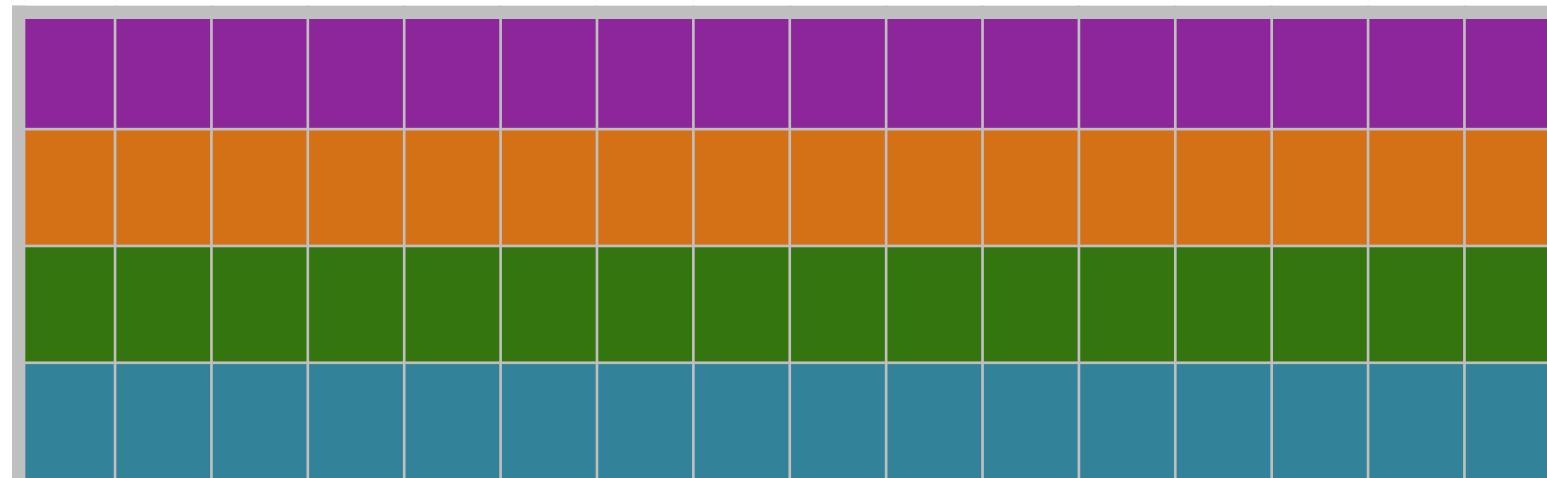
$$V = XW_V =$$



$$\text{scores} = QK^T$$

# Visual Transformer

scores =  $QK^T$



$Q$



$K^T$

$$= \begin{matrix} & & & \\ & & & \\ & & & \\ & & & \\ & & & \\ & & & \\ & & & \\ & & & \end{matrix}$$

**Attention:**

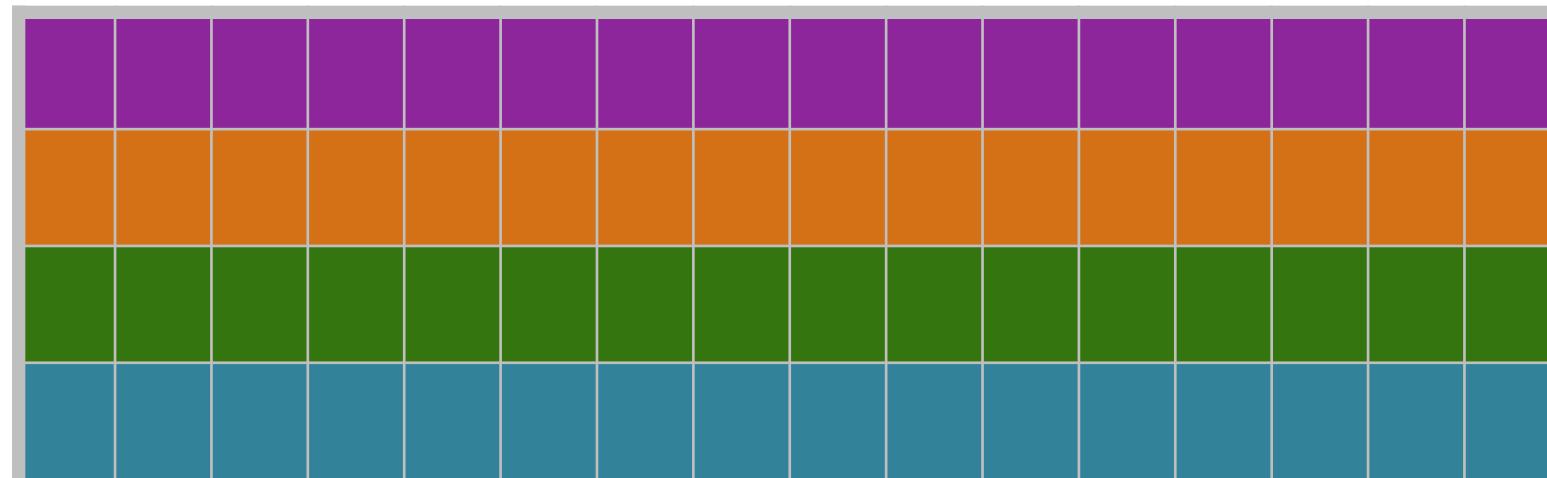
$$A = \text{softmax}(QK^T)$$

**Output**

$$Z = AV$$

# Visual Transformer

$$\text{scores} = QK^T$$

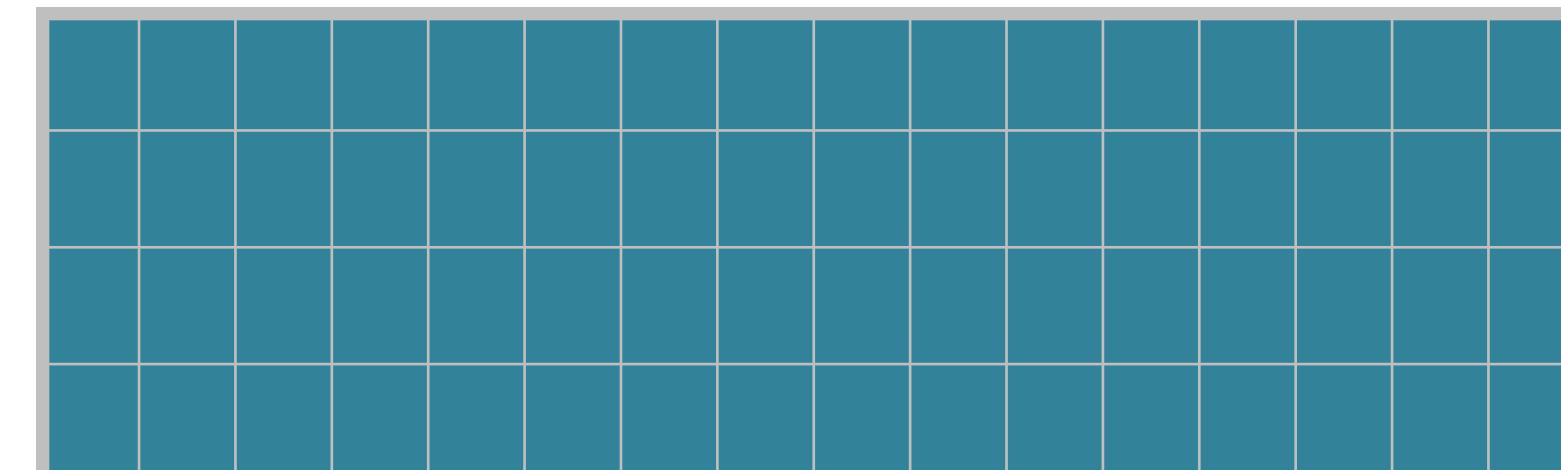
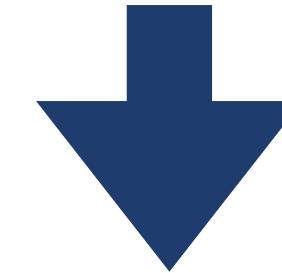
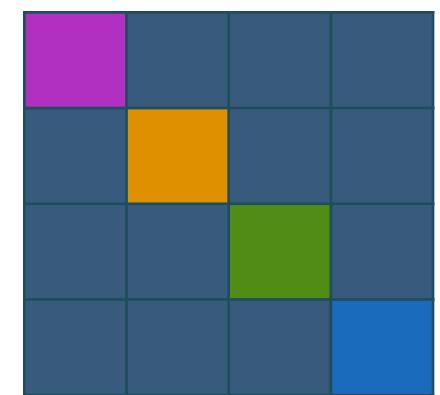


$Q$



$K^T$

=



**Attention:**

$$A = \text{softmax}(QK^T)$$

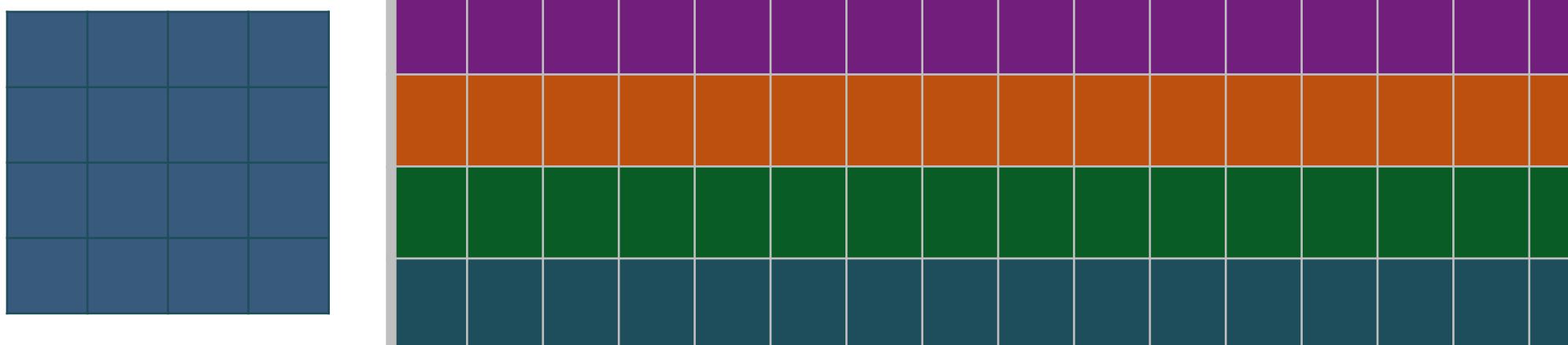
**Output**

$$Z = AV$$

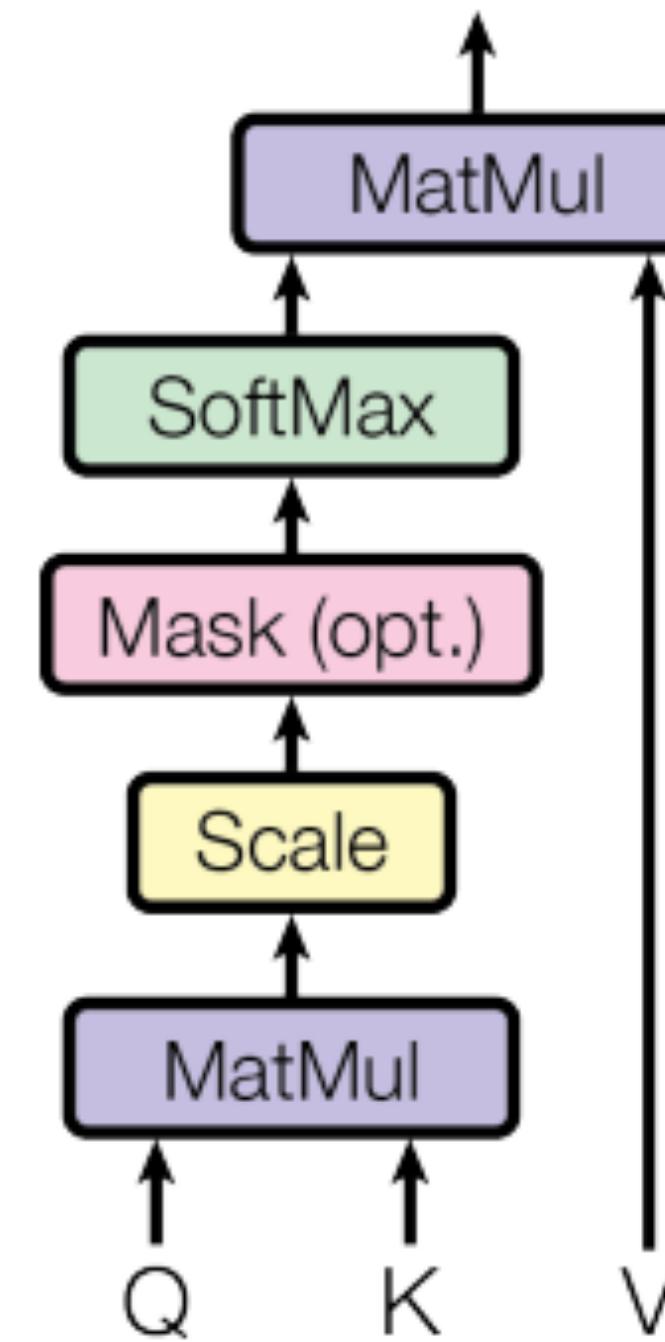
# Visual Transformer

## Output

$$Z = AV$$

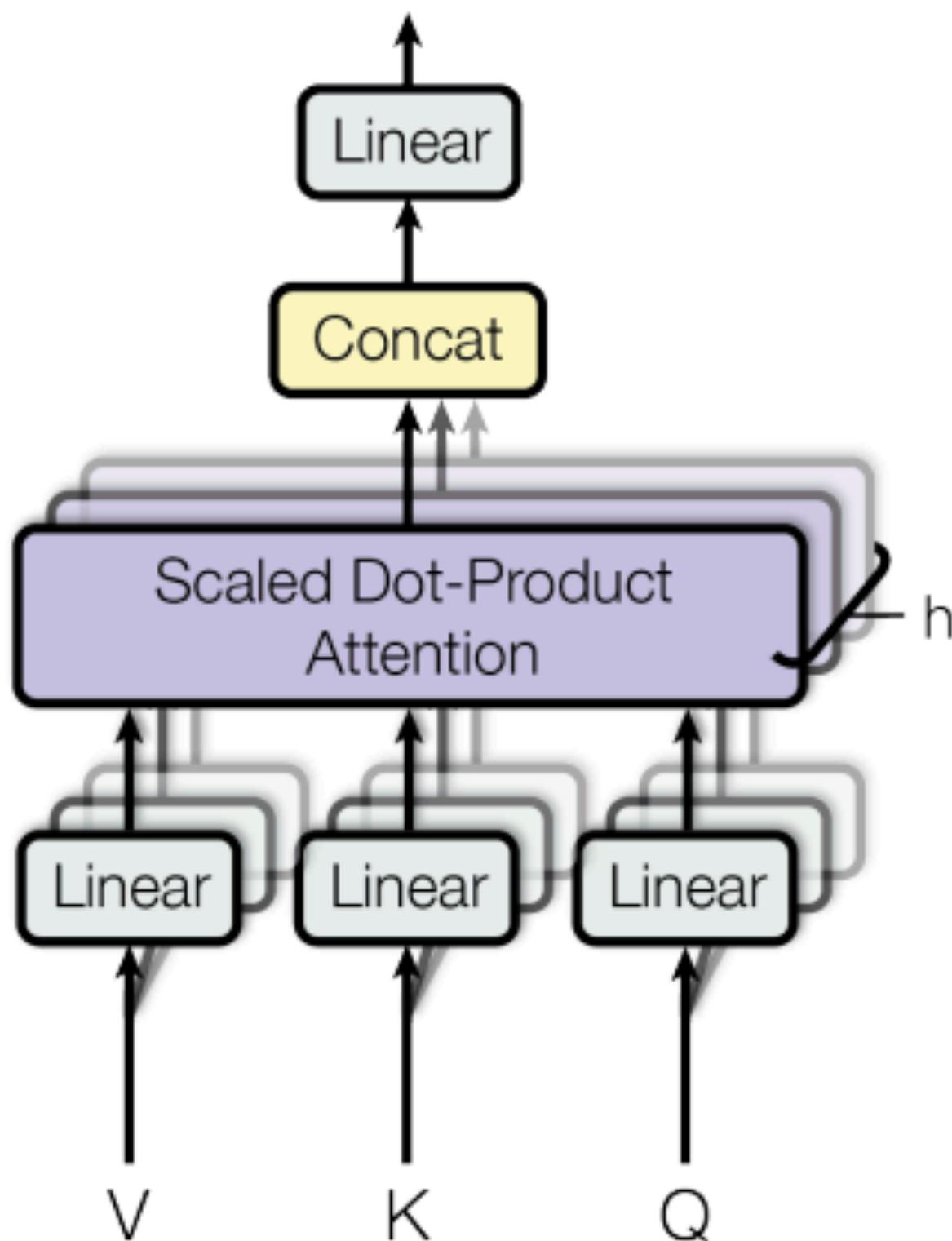


Scaled Dot-Product Attention

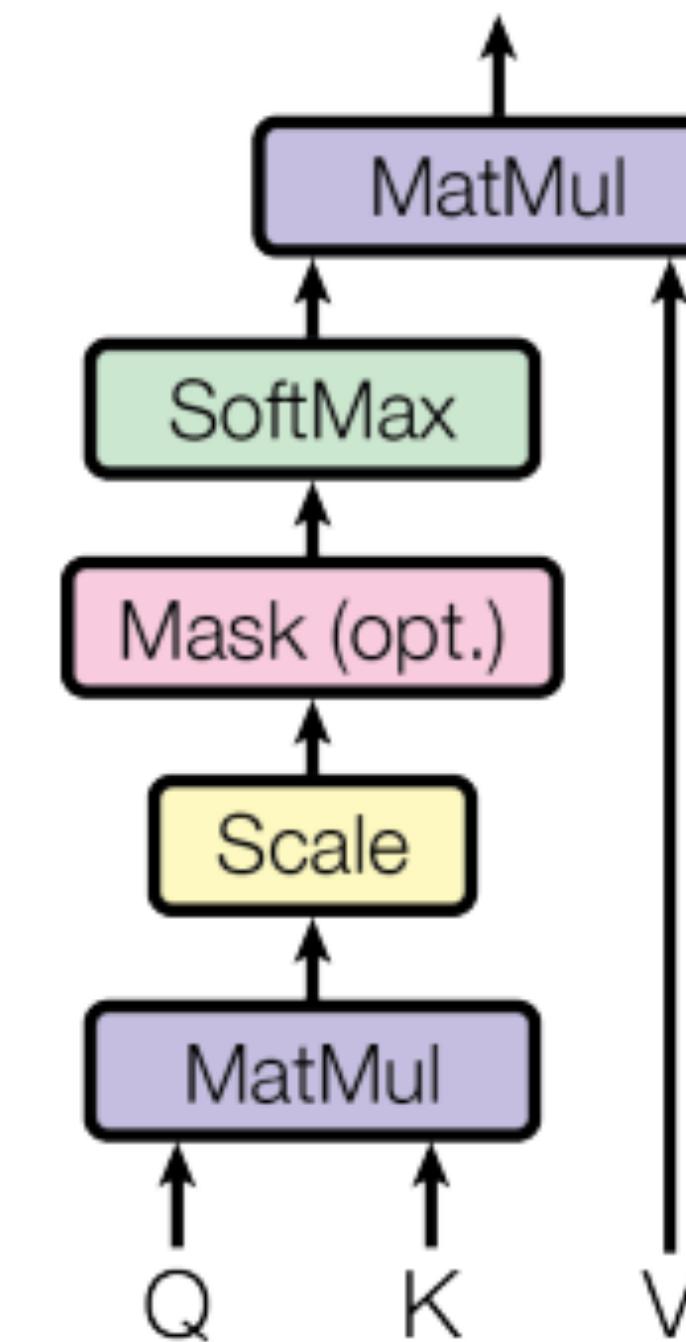


# Visual Transformer

## Multi-Head Attention



## Scaled Dot-Product Attention



---

**Attention Is All You Need**

---

Ashish Vaswani\*  
Google Brain  
avaswani@google.com

Noam Shazeer\*  
Google Brain  
noam@google.com

Niki Parmar\*  
Google Research  
nikip@google.com

Jakob Uszkoreit\*  
Google Research  
usz@google.com

Llion Jones\*  
Google Research  
llion@google.com

Aidan N. Gomez\* †  
University of Toronto  
aidan@cs.toronto.edu

Lukasz Kaiser\*  
Google Brain  
lukasz.kaiser@google.com

Illia Polosukhin\* ‡  
illia.polosukhin@gmail.com

# CHOOSING AN ARCHITECTURE

HOW MANY LAYERS?

HOW MANY NODES PER LAYER?

LEARNING RATE

DROPOUT?

WHAT ACTIVATION FUNCTION(S)?

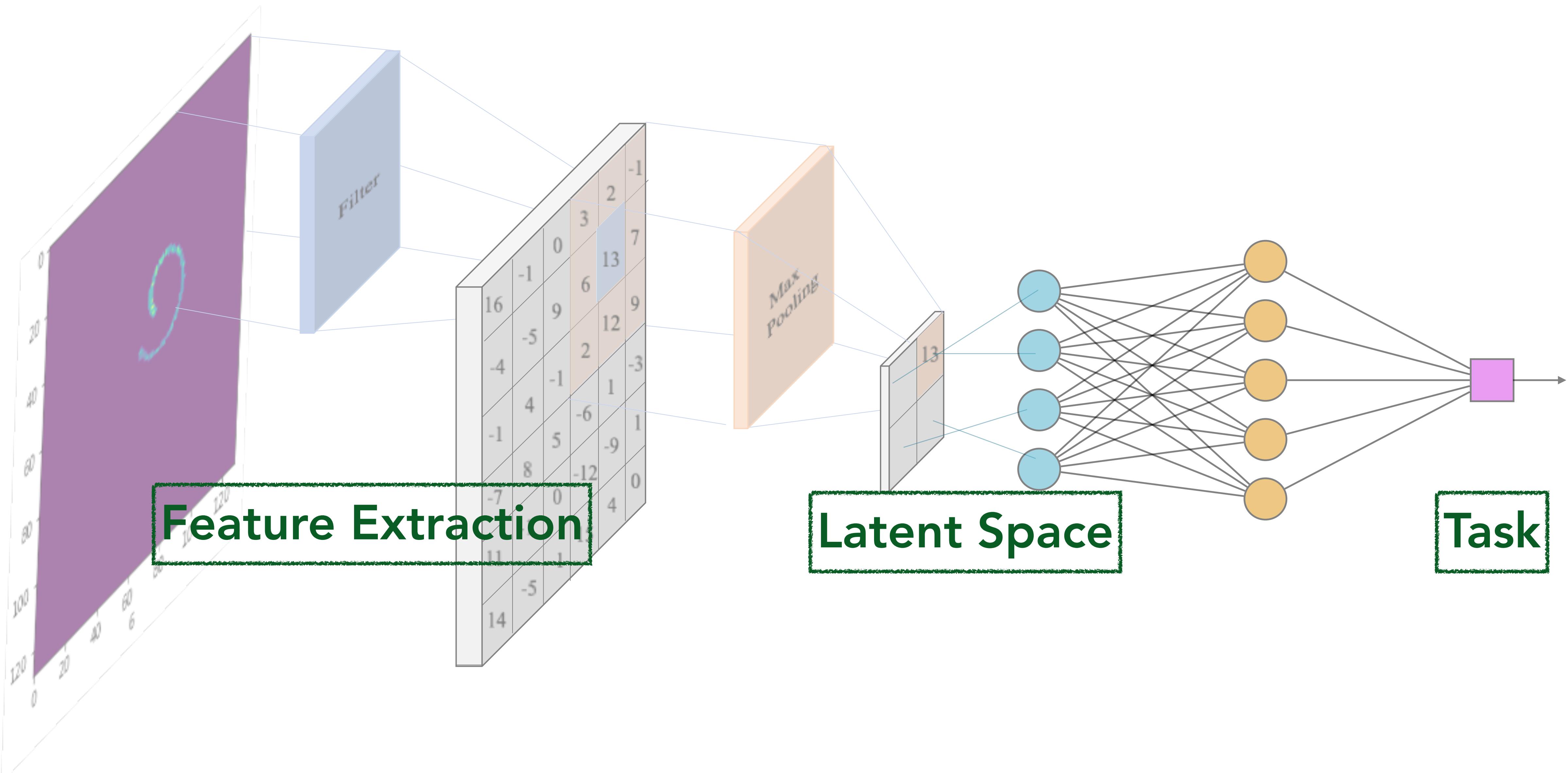
HOW MANY CONVOLUTION LAYERS?

FILTER SIZE?

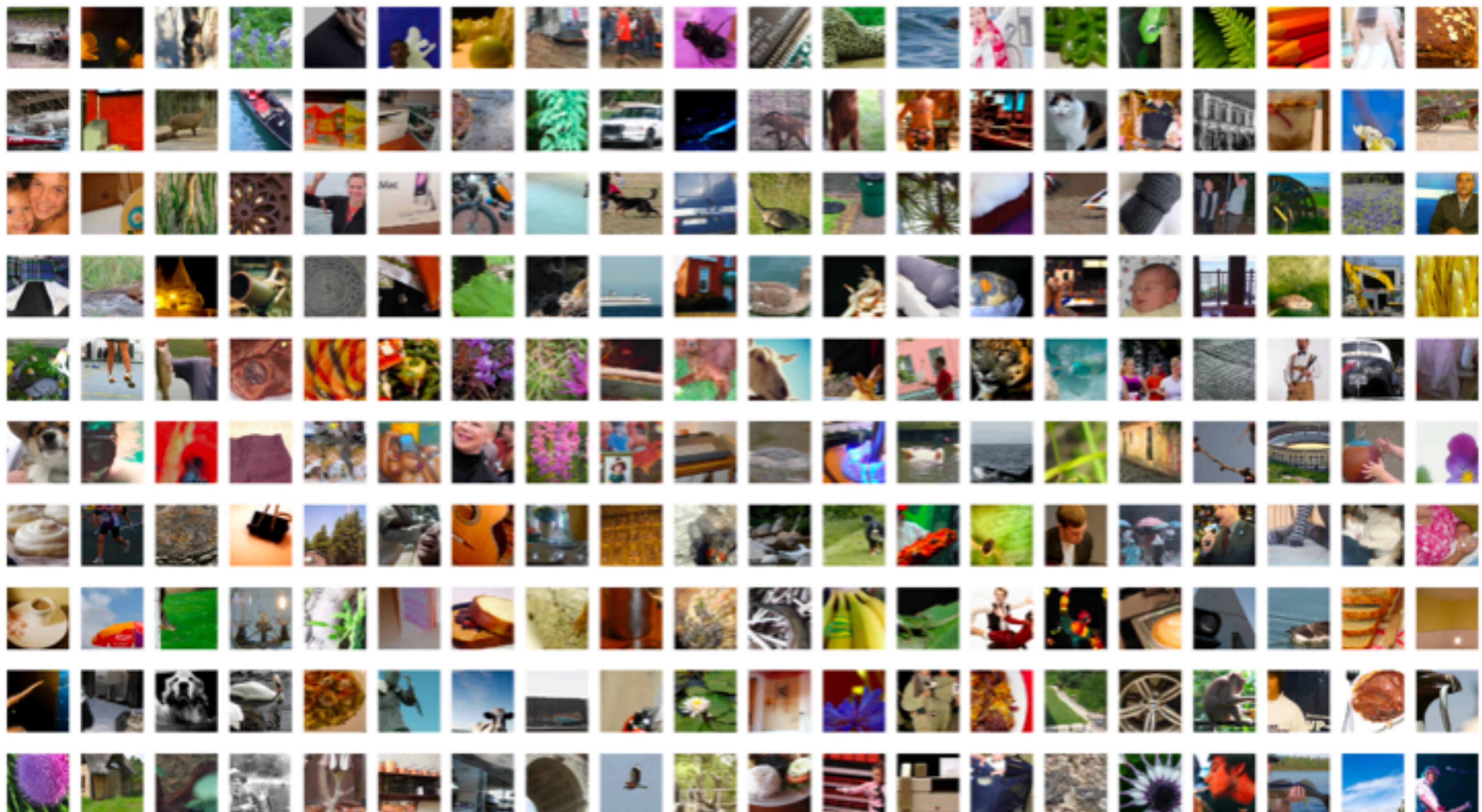
STRIDE?

POOLING?

# PRE-TRAINED MODELS



# PRE-TRAINED MODELS



# PRETRAINED MODELS

ALEXNET

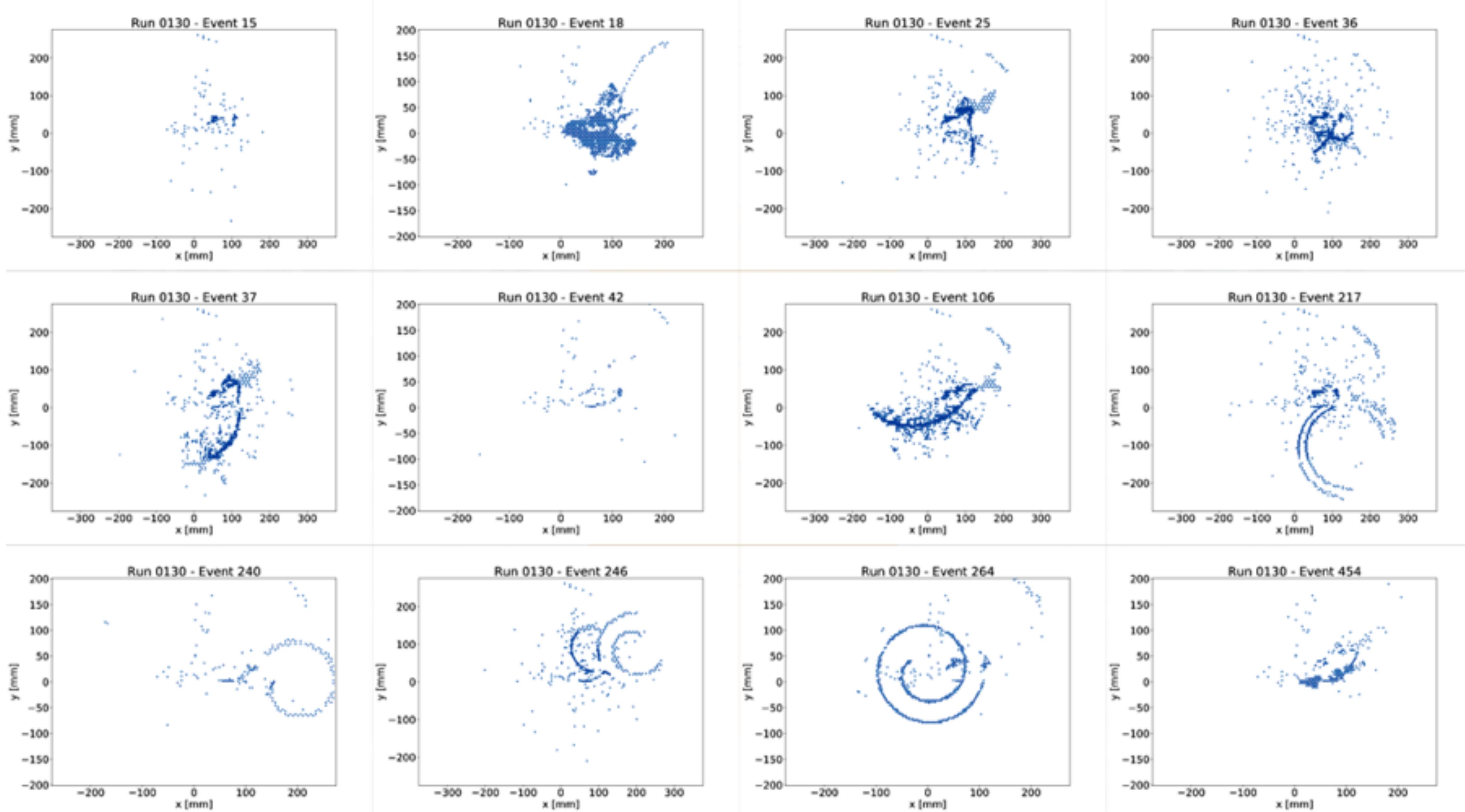
VGG

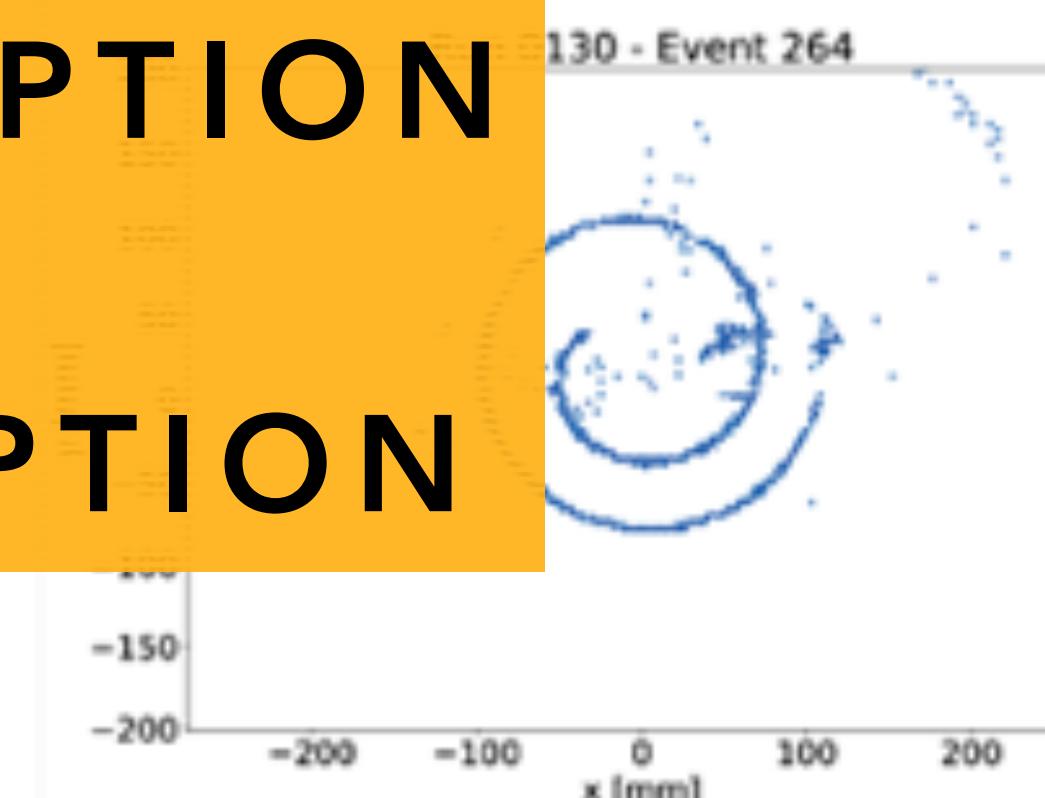
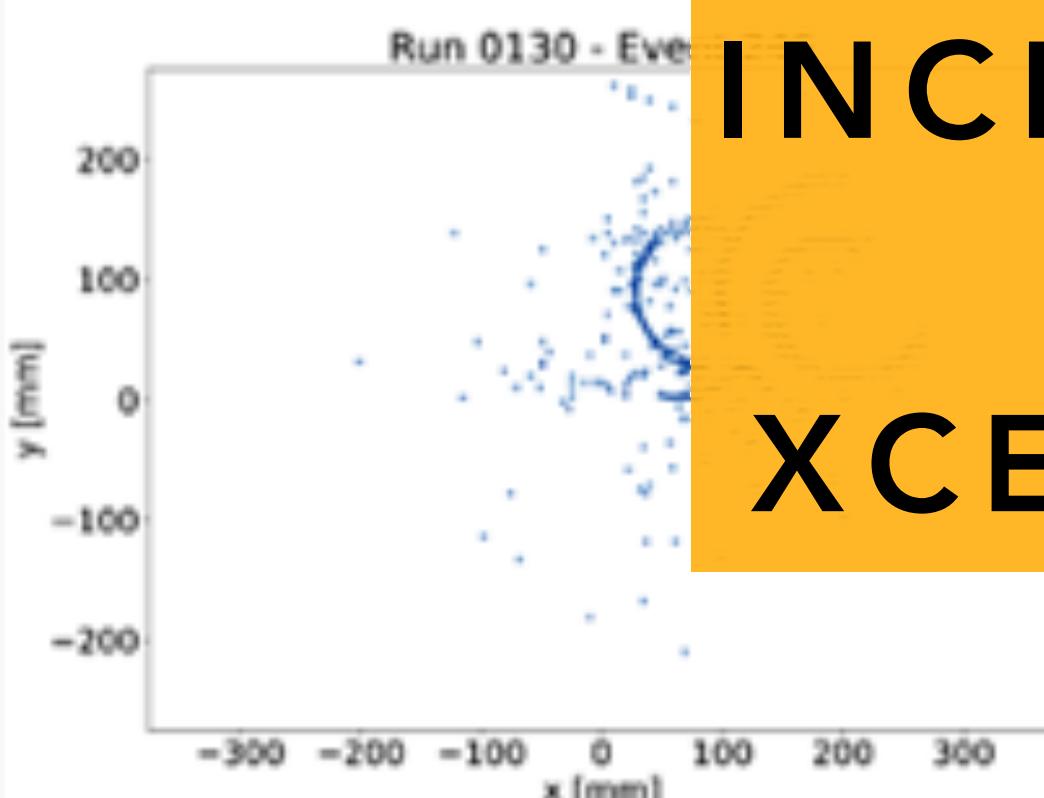
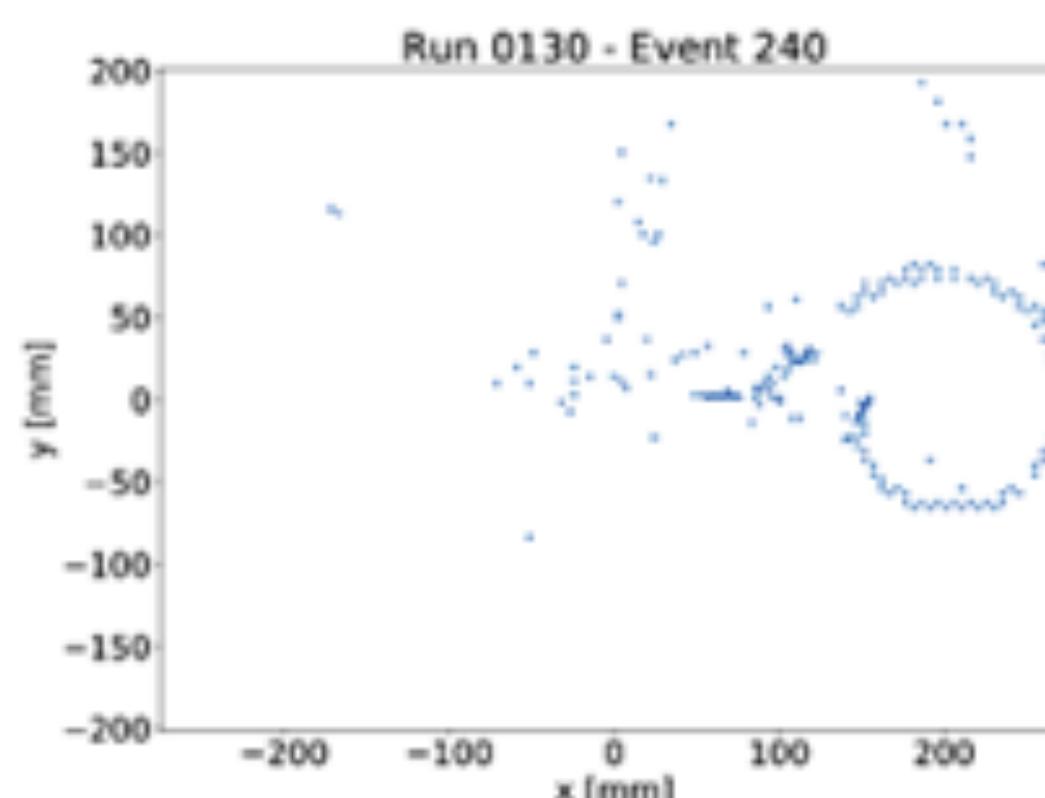
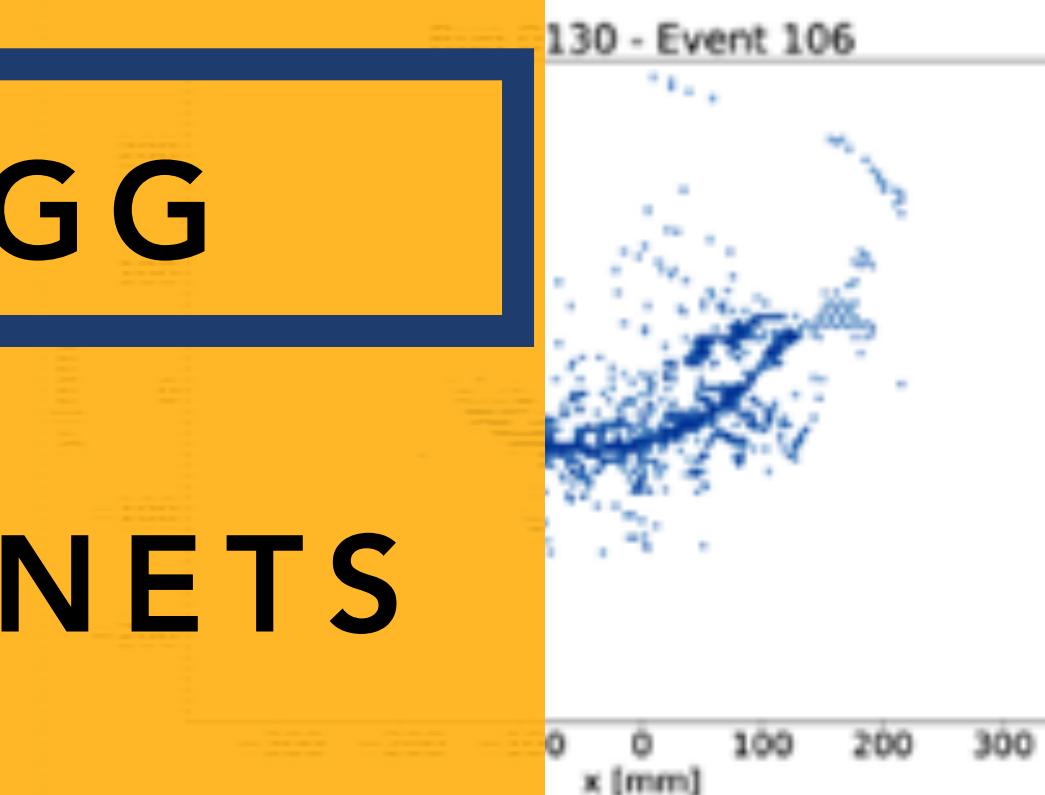
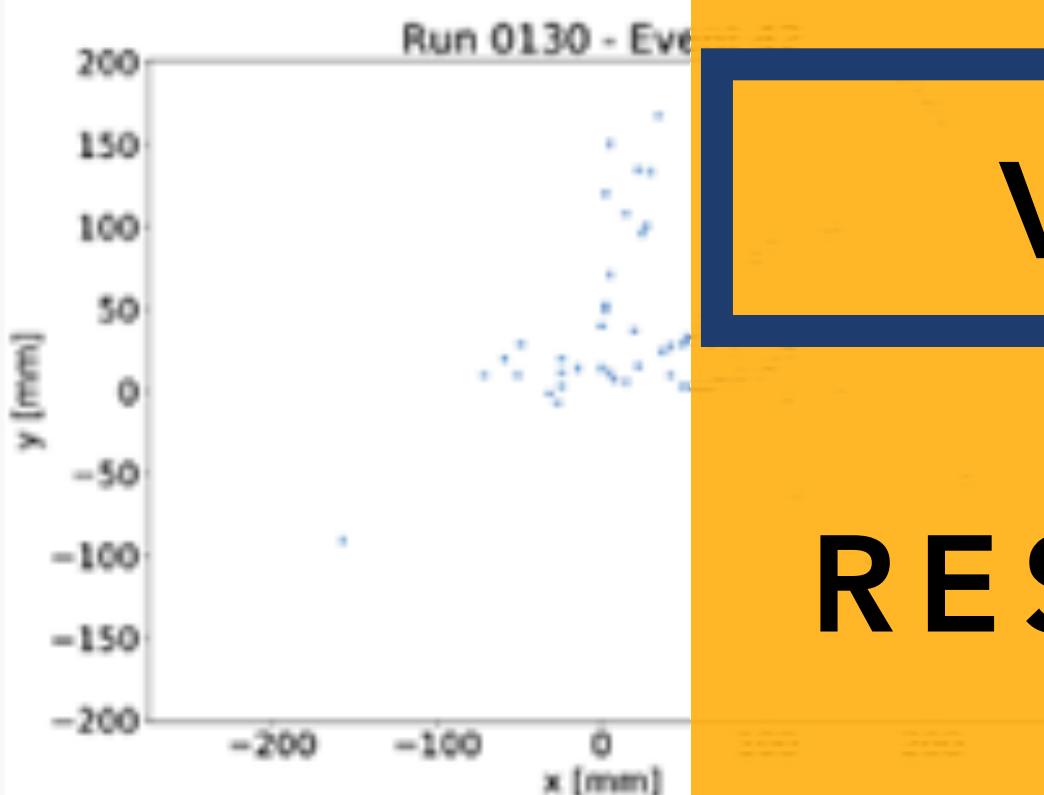
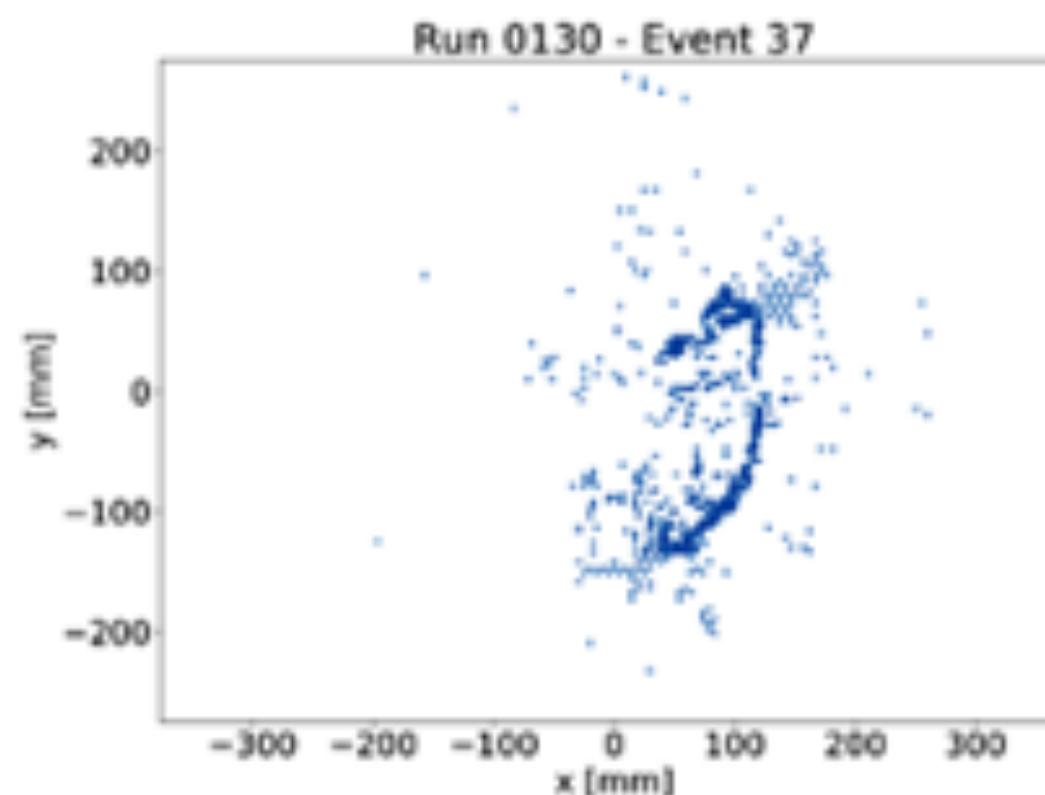
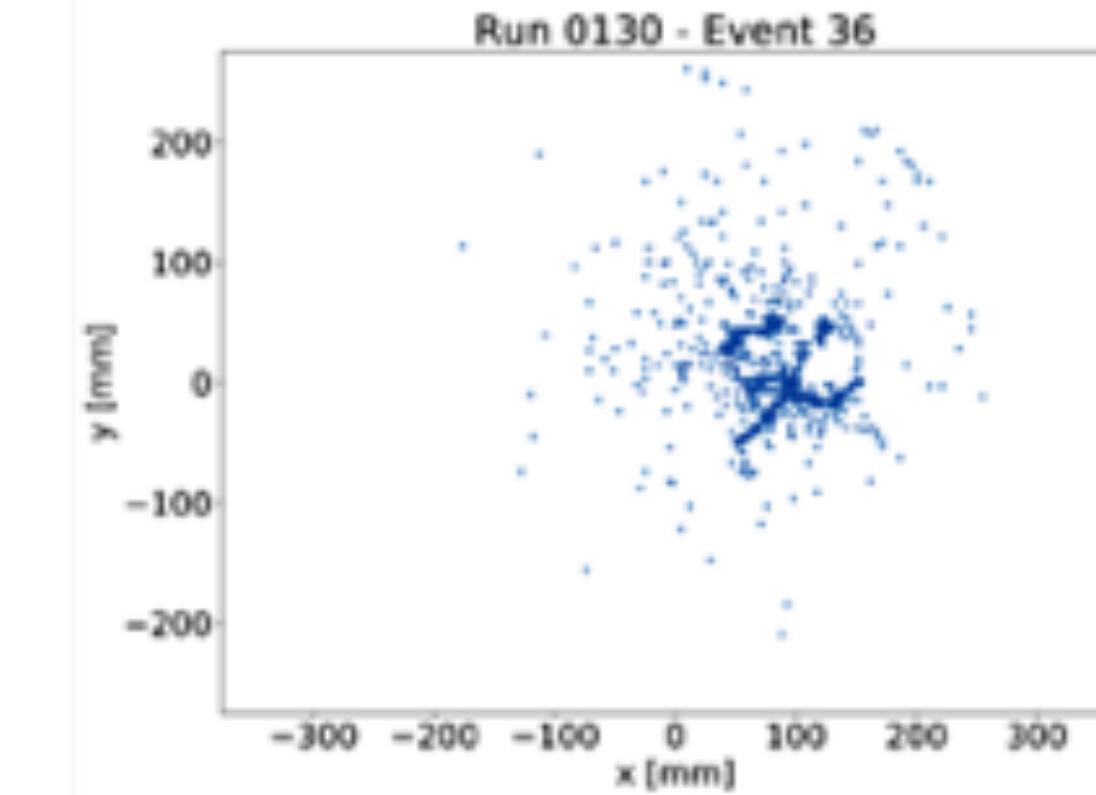
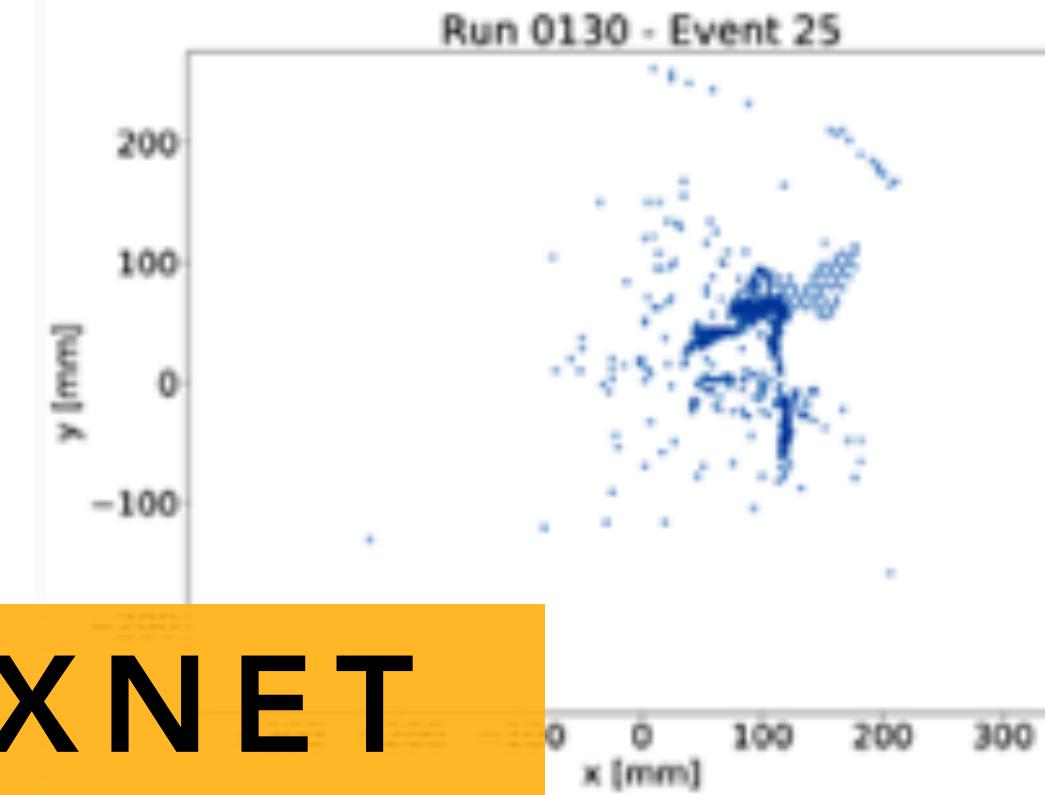
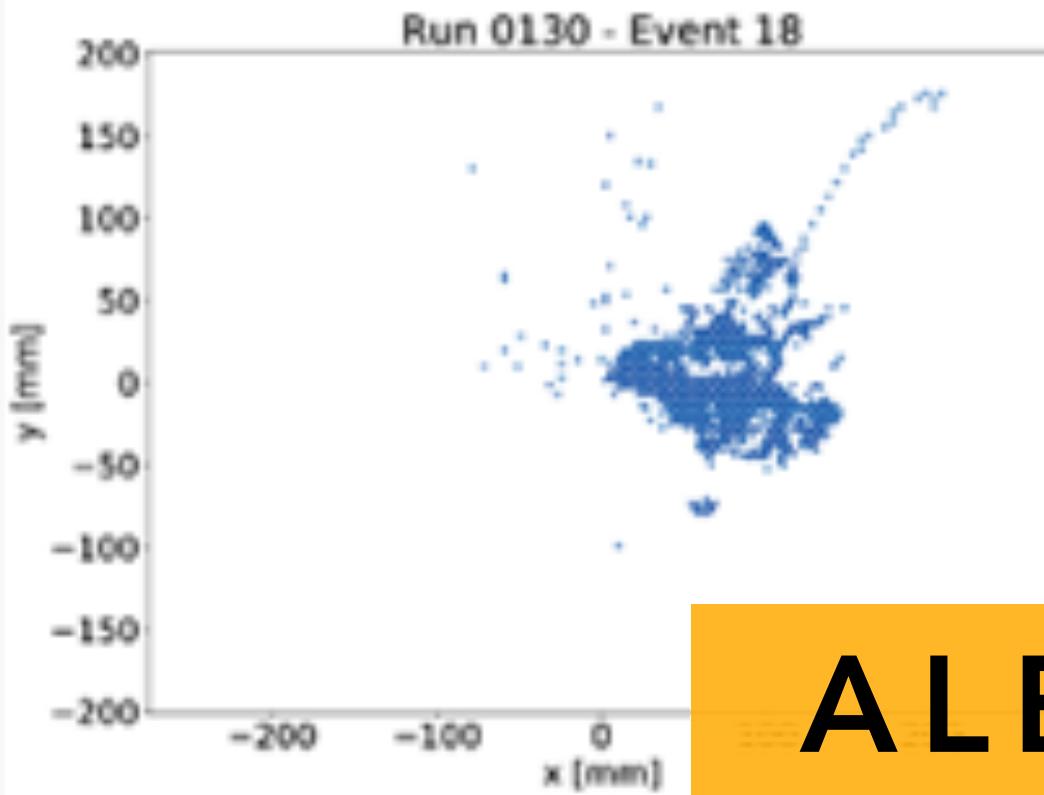
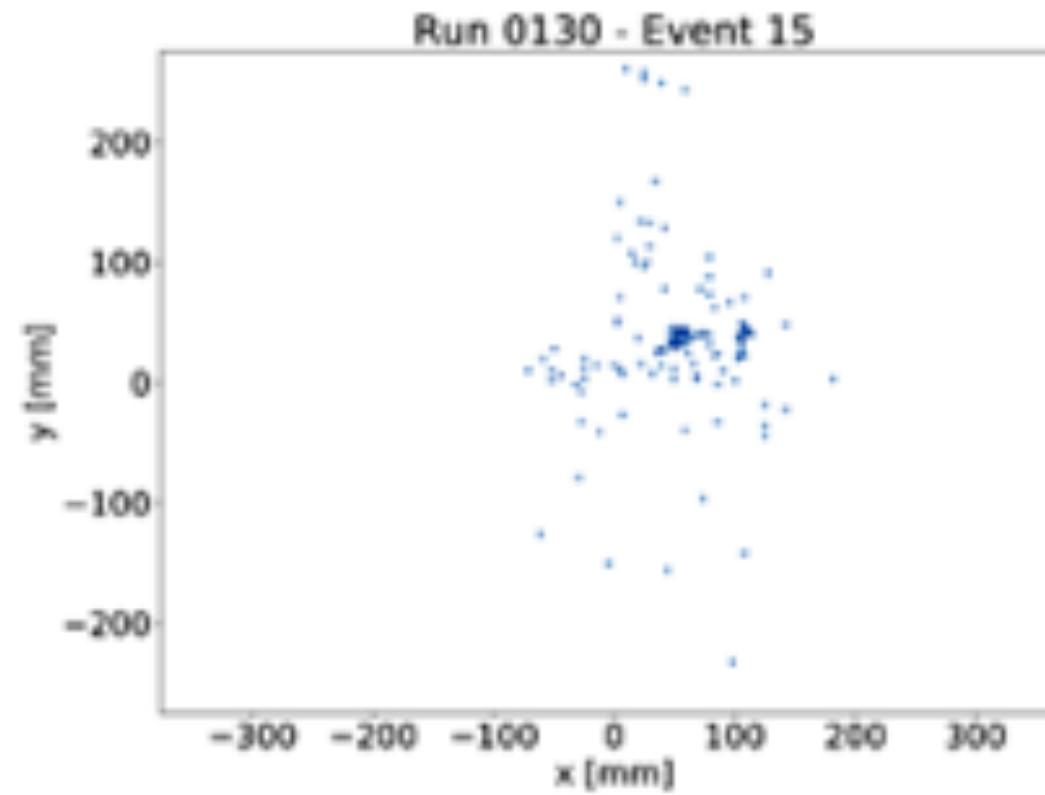
RESNETS

INCEPTION

XCEPTION







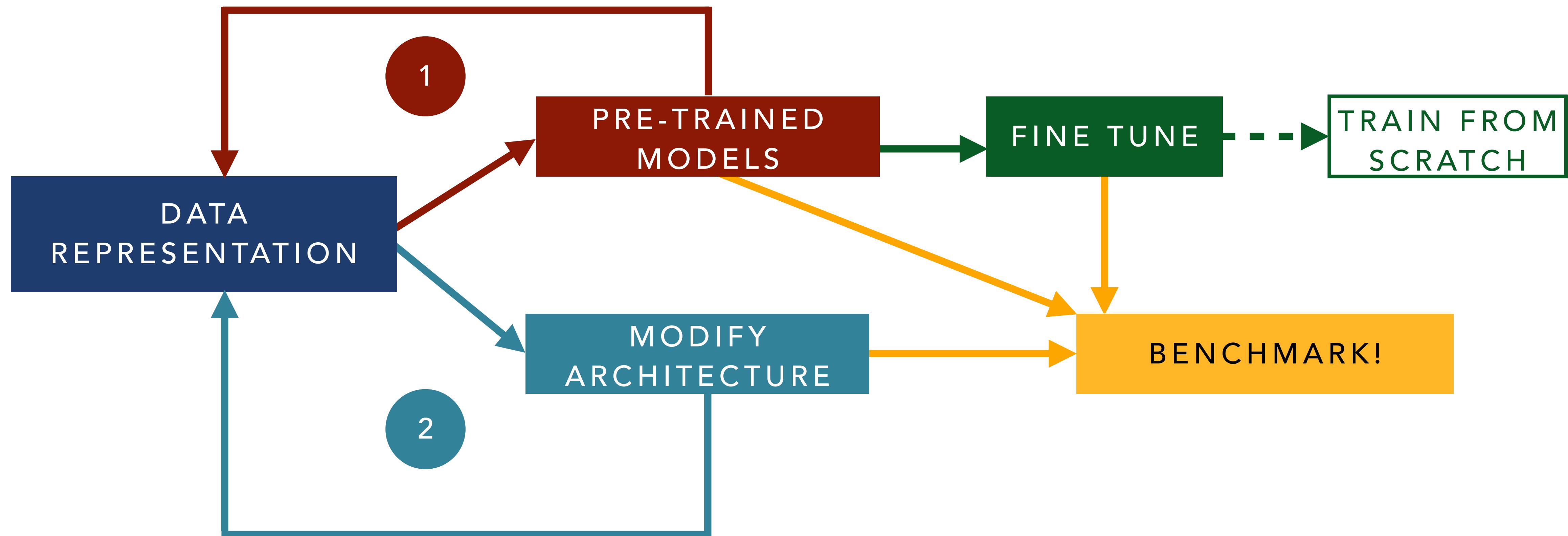
ALEXNET

VGG

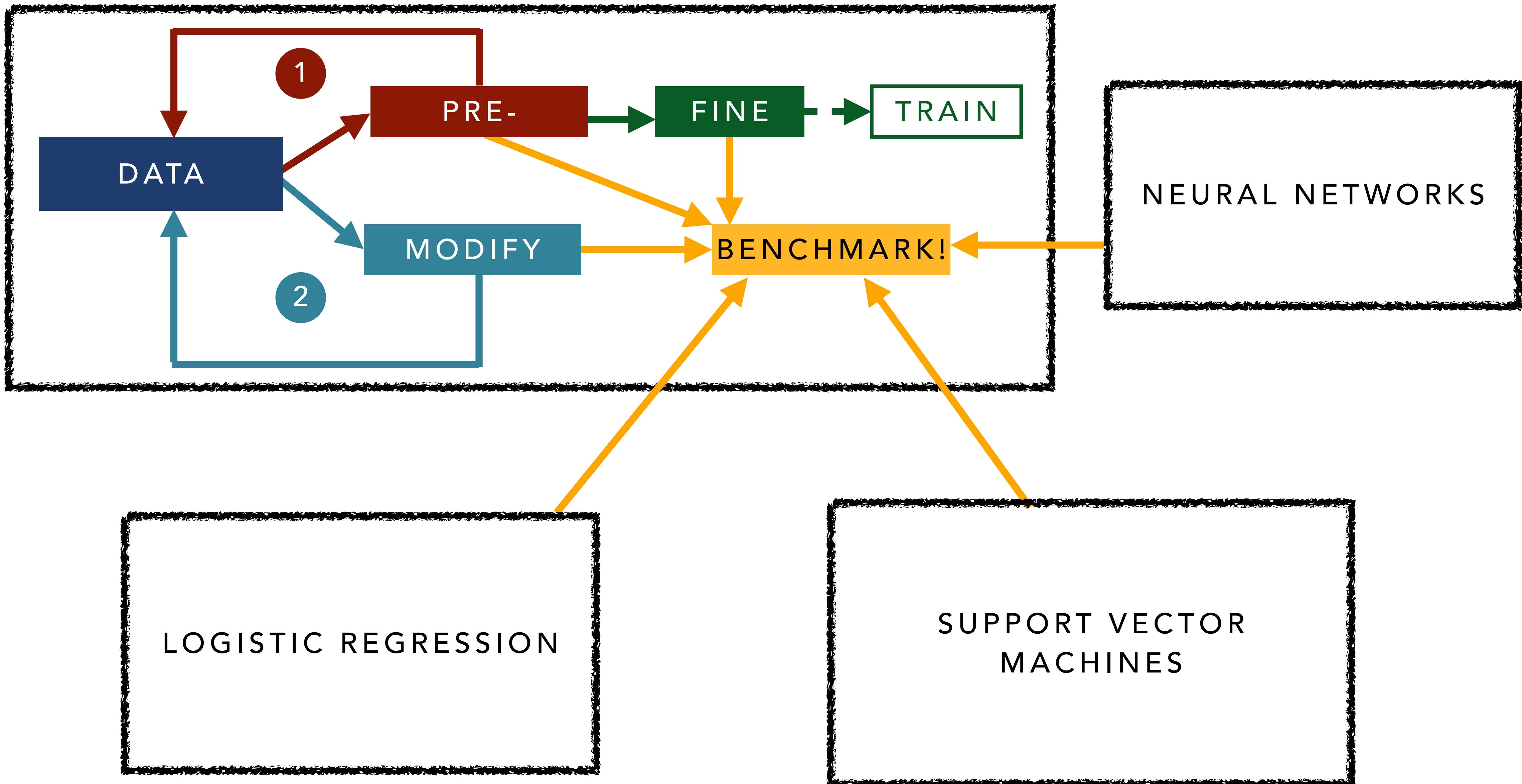
RESNETS

INCEPTION  
XCEPTION

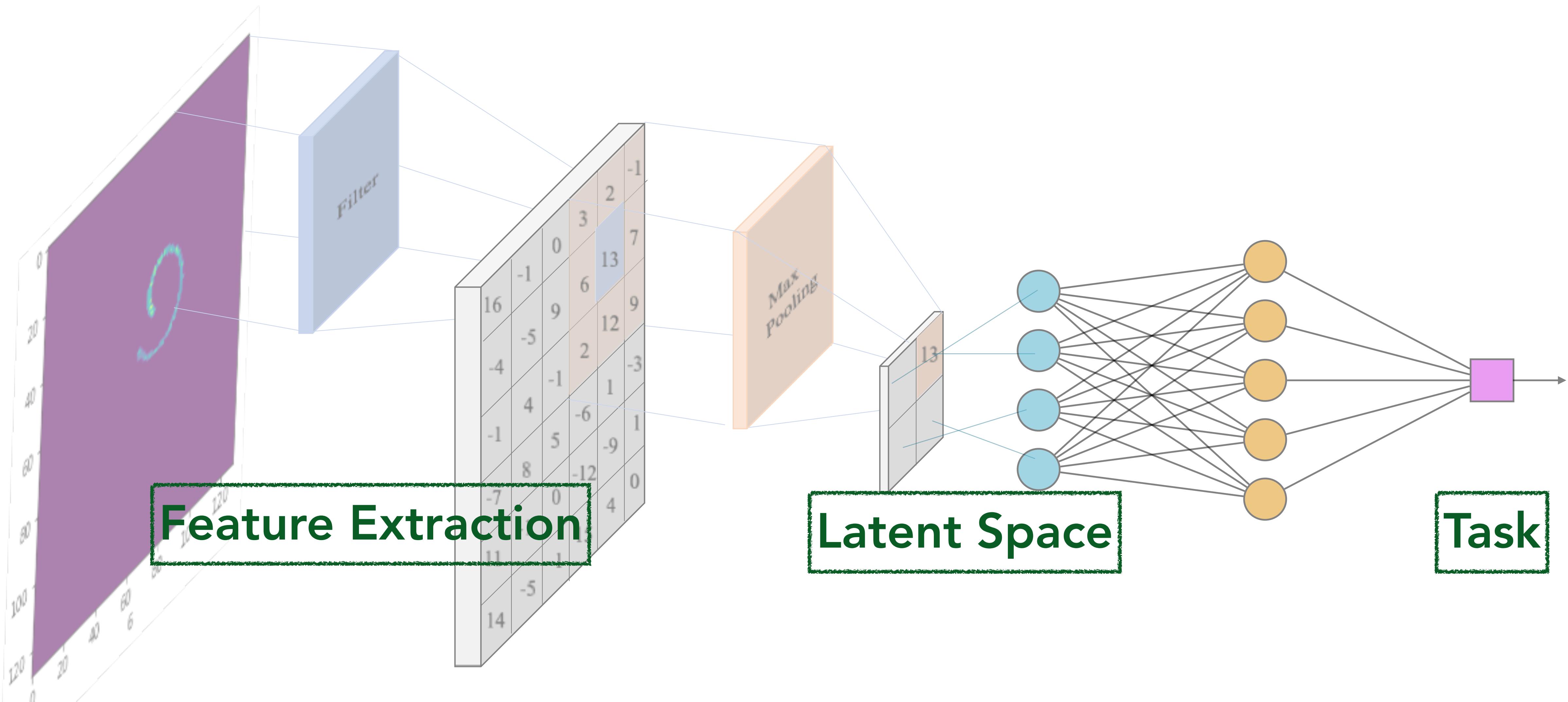
# EXAMPLE WORKFLOW



# EXAMPLE WORKFLOW



# CONVOLUTIONAL NEURAL NETWORKS



OTHER NETWORK  
MECHANISMS:  
-POINT CONVOLUTIONS  
-ATTENTION

