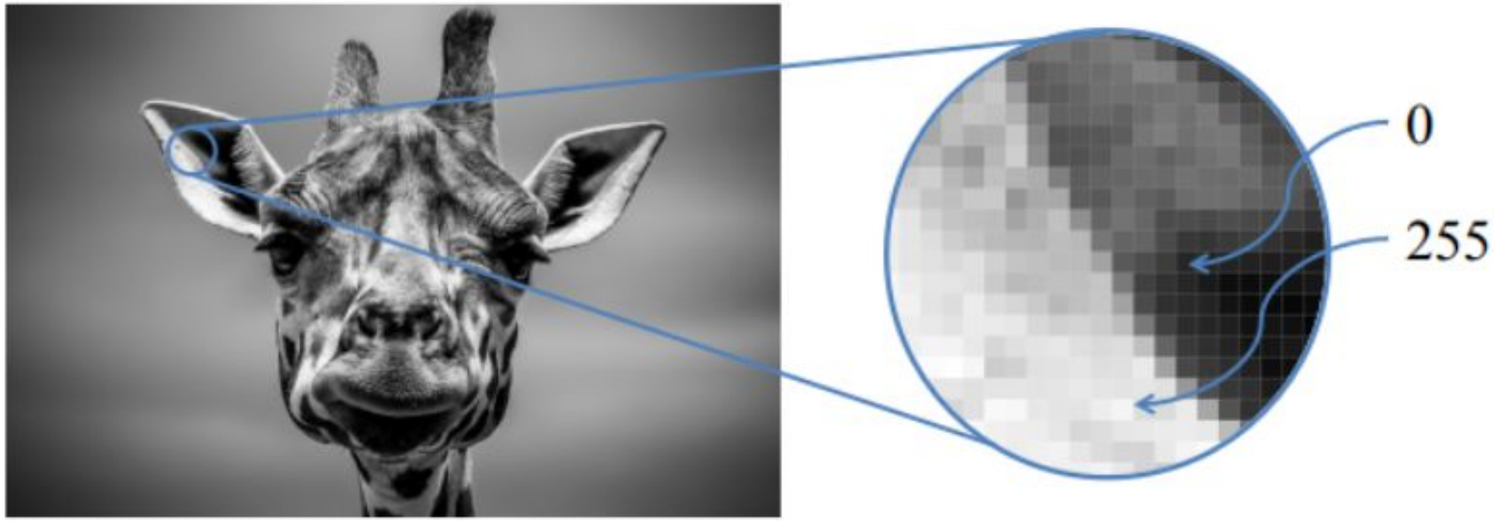
Digital representation of an image

Digital representation of an image

- Grayscale image is a matrix of pixels (**pic**ture **el**ements)
- Dimensions of this matrix are called image resolution (e.g. 300 x 300)
- Each pixel stores its brightness (or **intensity**) ranging from 0 to 255, 0 intensity corresponds to black color:



- Color images store pixel intensities for 3 channels: red, green and blue

Neural Network Issue for Computer Vision

- **Overfitting** due too many parameters(~millions), while working with medium-large sized images!
- Fail to handle variance in images - translation, rotation, illumination, size etc!

Neural Network Issue for Computer Vision

Translation Invariance

Rotation/Viewpoint Invariance

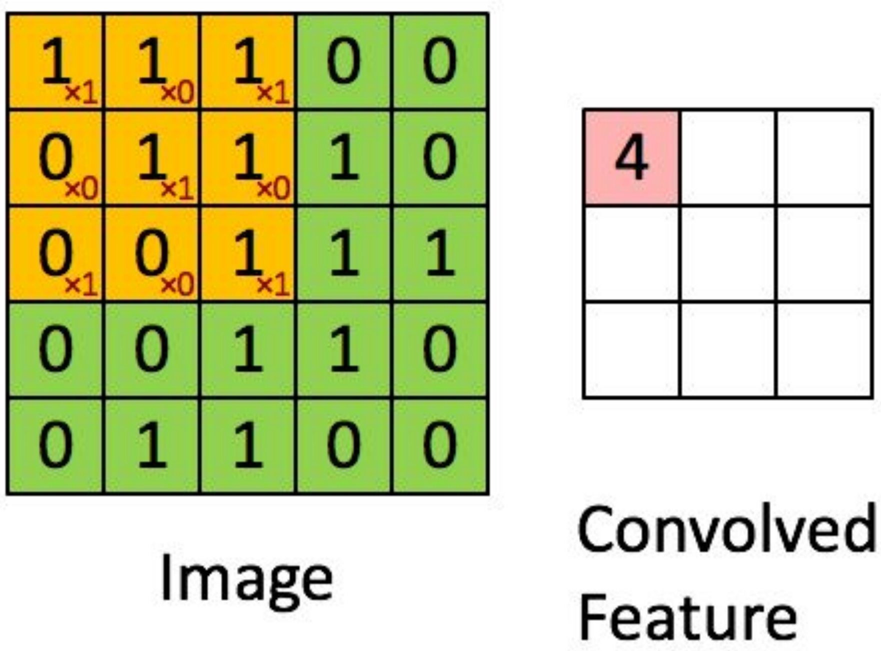
Size Invariance

Illumination Invariance

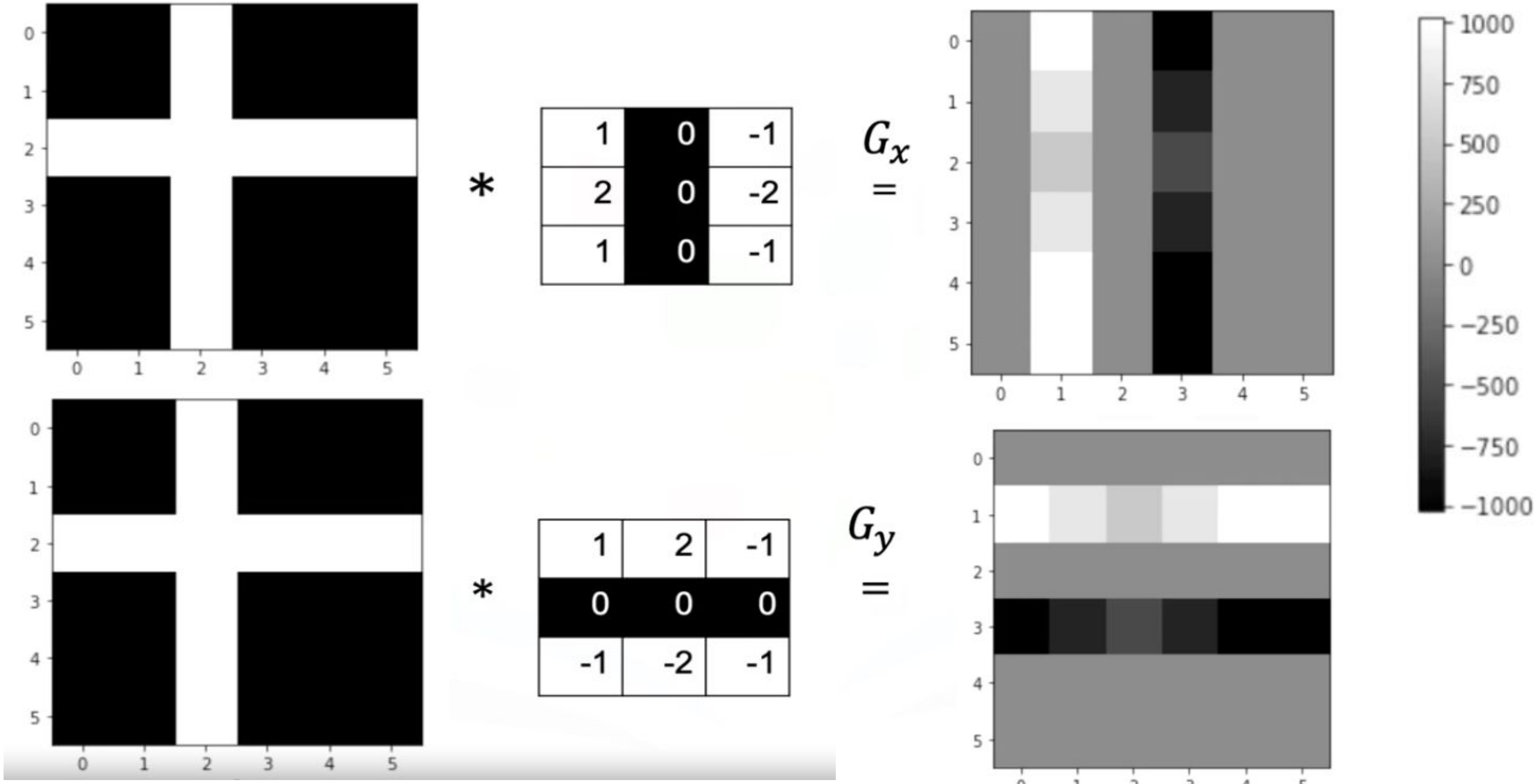
Convolutional Neural Networks

CNN can understand different position/size of the features

Convolutional Neural Networks



Convolutional Neural Networks



Convolutions have been used for a while

Kernel

$*$

-1	-1	-1
-1	8	-1
-1	-1	-1

$=$

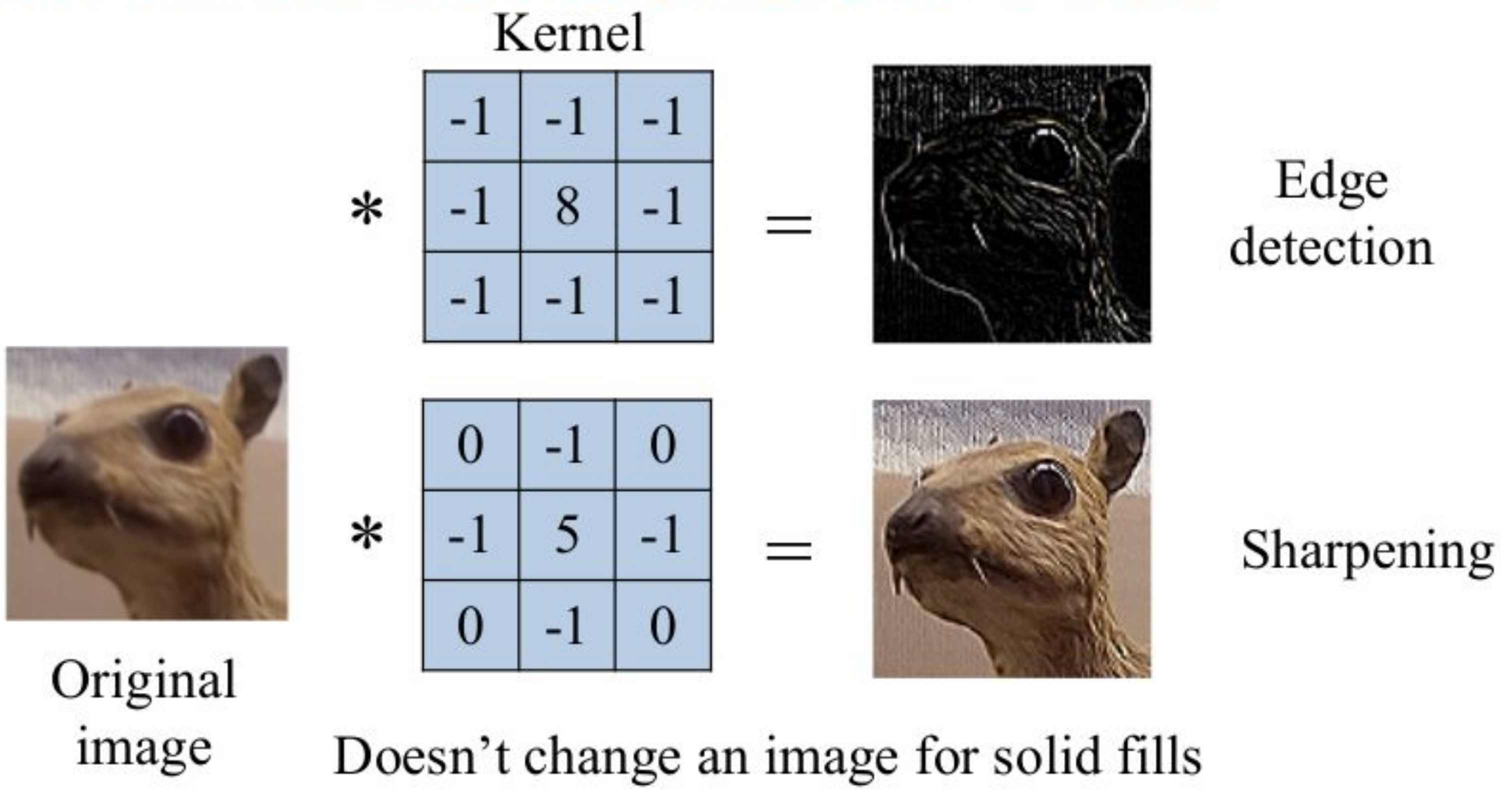
Edge detection

Original image

Sums up to 0 (black color) when the patch is a solid fill

Convolutional Neural Networks

Convolutions have been used for a while

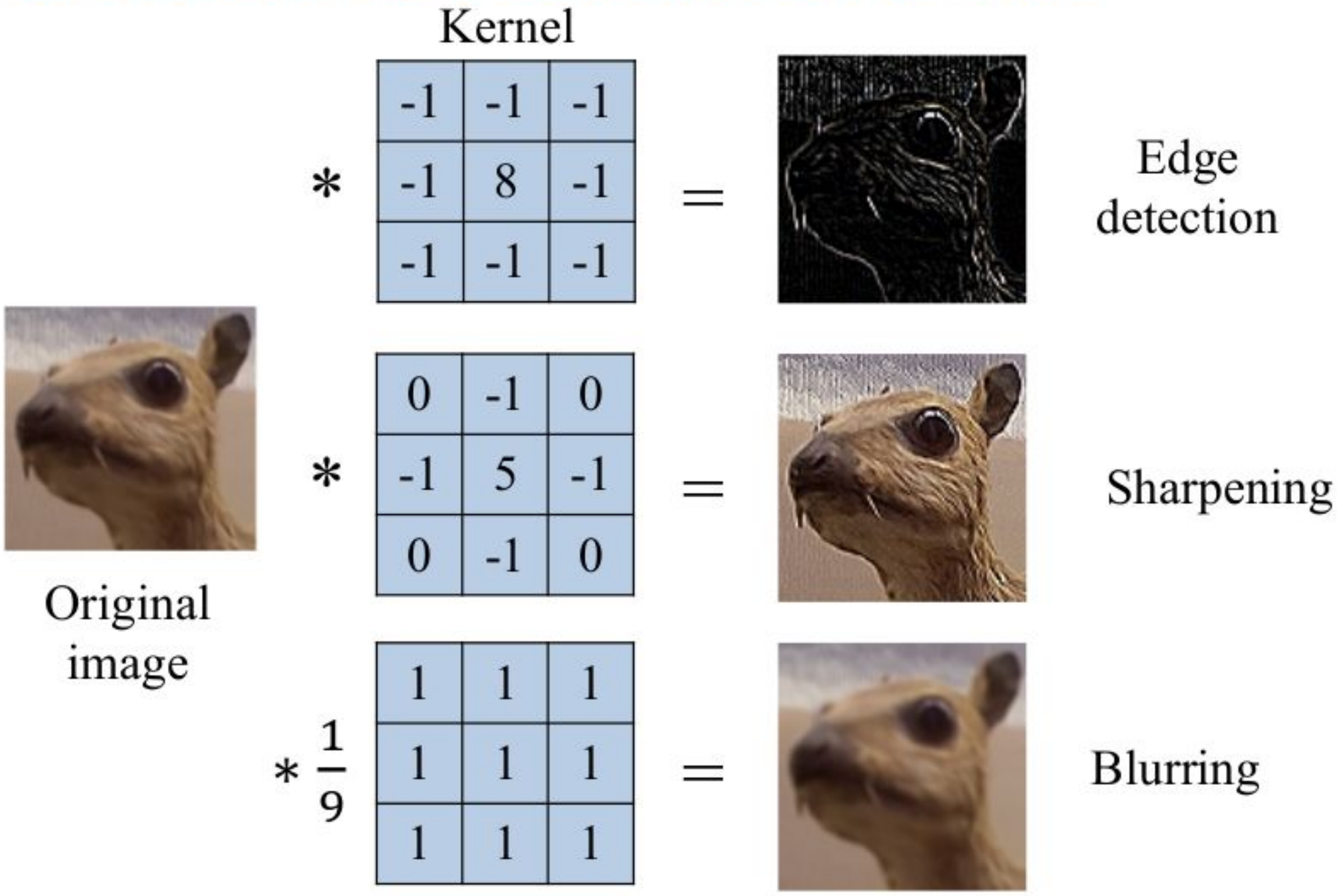


Doesn’t change an image for solid fills

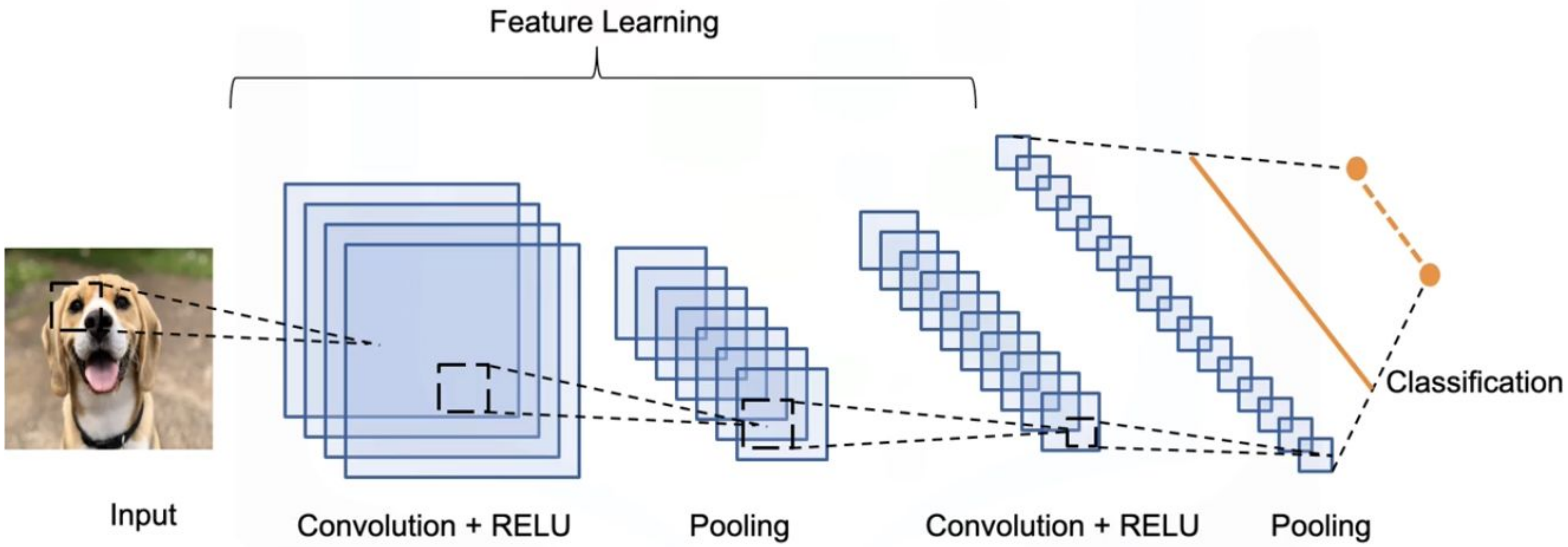
Adds a little intensity on the edges

Convolutional Neural Networks

Convolutions have been used for a while

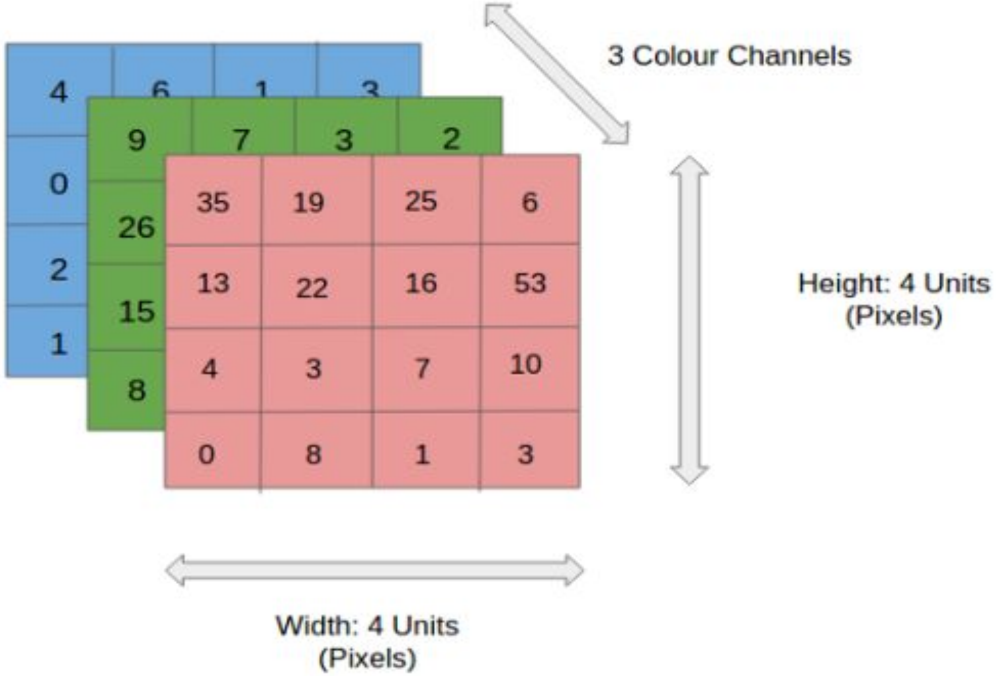


CNN for Image Classification



Convolutional Neural Networks - Input Layer

Fig: RGB Image



Convolutional Neural Network (CNN)

The output computation now only depends on a subset of inputs:

$$y_{11} = w_{11}x_{11} + w_{12}x_{12} + w_{13}x_{13} + w_{21}x_{21} + w_{22}x_{22} + w_{23}x_{23} + w_{31}x_{31} + w_{32}x_{32} + w_{33}x_{33}$$

$$O = \frac{M-F+2P}{S} + 1$$

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \\ x_{41} & x_{42} & x_{43} & x_{44} & x_{45} \\ x_{51} & x_{52} & x_{53} & x_{54} & x_{55} \end{bmatrix} * \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} = \begin{bmatrix} y_{11} & & \\ & & \\ & & \end{bmatrix}$$

Input Image (5*5)

Kernel or Filter (3*3)

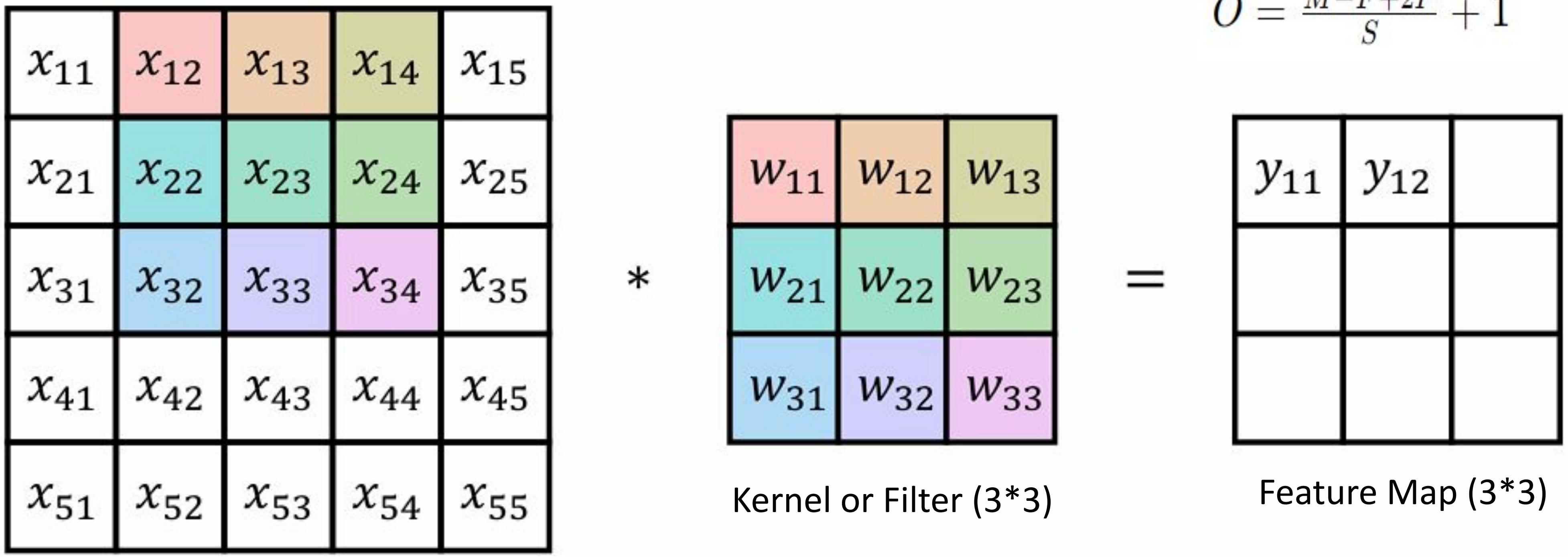
Feature Map (3*3)

Convolutional Neural Network (CNN)

The output computation now only depends on a subset of inputs:

$$y_{12} = w_{11}x_{12} + w_{12}x_{13} + w_{13}x_{14} + w_{21}x_{22} + w_{22}x_{23} + w_{23}x_{24} + w_{31}x_{32} + w_{32}x_{33} + w_{33}x_{34}$$

$$O = \frac{M-F+2P}{S} + 1$$



Kernel or Filter (3*3)

Feature Map (3*3)

Input Image (5*5)

The output computation now only depends on a subset of inputs:

$$y_{13} = w_{11}x_{13} + w_{12}x_{14} + w_{13}x_{15} + w_{21}x_{23} + w_{22}x_{24} + w_{23}x_{25} + w_{31}x_{33} + w_{32}x_{34} + w_{33}x_{35}$$

$$O = \frac{M-F+2P}{S} + 1$$

$$\begin{array}{|c|c|c|c|c|}\hline x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ \hline x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ \hline x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \\ \hline x_{41} & x_{42} & x_{43} & x_{44} & x_{45} \\ \hline x_{51} & x_{52} & x_{53} & x_{54} & x_{55} \\ \hline \end{array} * \begin{array}{|c|c|c|}\hline w_{11} & w_{12} & w_{13} \\ \hline w_{21} & w_{22} & w_{23} \\ \hline w_{31} & w_{32} & w_{33} \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline y_{11} & y_{12} & y_{13} \\ \hline & & \\ \hline & & \\ \hline \end{array}$$

Input Image (5*5)

Kernel or Filter (3*3)

Feature Map (3*3)

Convolutional Neural Network (CNN)

The output computation now only depends on a subset of inputs:

$$y_{23} = w_{11}x_{23} + w_{12}x_{24} + w_{13}x_{25} + w_{21}x_{33} + w_{22}x_{34} + w_{23}x_{35} + w_{31}x_{43} + w_{32}x_{44} + w_{33}x_{45}$$

$$O = \frac{M-F+2P}{S} + 1$$

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \\ x_{41} & x_{42} & x_{43} & x_{44} & x_{45} \\ x_{51} & x_{52} & x_{53} & x_{54} & x_{55} \end{bmatrix} * \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} = \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ & & y_{23} \\ & & \end{bmatrix}$$

Input Image (5*5)

Kernel or Filter (3*3)

Feature Map (3*3)

Convolutional Neural Network (CNN)

The output computation now only depends on a subset of inputs:

$$y_{22} = w_{11}x_{22} + w_{12}x_{23} + w_{13}x_{24} + w_{21}x_{32} + w_{22}x_{33} + w_{23}x_{34} + w_{31}x_{42} + w_{32}x_{43} + w_{33}x_{44}$$

$$O = \frac{M-F+2P}{S} + 1$$

$$\begin{array}{|c|c|c|c|c|} \hline x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ \hline x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ \hline x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \\ \hline x_{41} & x_{42} & x_{43} & x_{44} & x_{45} \\ \hline x_{51} & x_{52} & x_{53} & x_{54} & x_{55} \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline w_{11} & w_{12} & w_{13} \\ \hline w_{21} & w_{22} & w_{23} \\ \hline w_{31} & w_{32} & w_{33} \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline y_{11} & y_{12} & y_{13} \\ \hline & y_{22} & y_{23} \\ \hline & & \\ \hline \end{array}$$

Input Image (5*5)

Kernel or Filter (3*3)

Feature Map (3*3)

Convolutional Neural Network (CNN)

The output computation now only depends on a subset of inputs:

$$y_{21} = w_{11}x_{21} + w_{12}x_{22} + w_{13}x_{23} + w_{21}x_{31} + w_{22}x_{32} + w_{23}x_{33} + w_{31}x_{41} + w_{32}x_{42} + w_{33}x_{43}$$

$$O = \frac{M-F+2P}{S} + 1$$

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \\ x_{41} & x_{42} & x_{43} & x_{44} & x_{45} \\ x_{51} & x_{52} & x_{53} & x_{54} & x_{55} \end{bmatrix} * \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} = \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ & & \end{bmatrix}$$

Input Image (5*5)

Kernel or Filter (3*3)

Feature Map (3*3)

Convolutional Neural Network (CNN)

The output computation now only depends on a subset of inputs:

$$y_{31} = w_{11}x_{31} + w_{12}x_{32} + w_{13}x_{33} + w_{21}x_{41} + w_{22}x_{42} + w_{23}x_{43} + w_{31}x_{51} + w_{32}x_{52} + w_{33}x_{53}$$

$$O = \frac{M-F+2P}{S} + 1$$

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \\ x_{41} & x_{42} & x_{43} & x_{44} & x_{45} \\ x_{51} & x_{52} & x_{53} & x_{54} & x_{55} \end{bmatrix} * \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} = \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ y_{31} & & \end{bmatrix}$$

Input Image (5*5)

Kernel or Filter (3*3)

Feature Map (3*3)

Convolutional Neural Network (CNN)

The output computation now only depends on a subset of inputs:

$$y_{32} = w_{11}x_{32} + w_{12}x_{33} + w_{13}x_{34} + w_{21}x_{42} + w_{22}x_{43} + w_{23}x_{44} + w_{31}x_{52} + w_{32}x_{53} + w_{33}x_{54}$$

$$O = \frac{M - F + 2P}{S} + 1$$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

Input Image (5*5)

*

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

Kernel or Filter (3*3)

=

y_{11}	y_{12}	y_{13}
y_{21}	y_{22}	y_{23}
y_{31}	y_{32}	

Feature Map (3*3)

Convolutional Neural Network (CNN)

The output computation now only depends on a subset of inputs:

$$y_{33} = w_{11}x_{33} + w_{12}x_{34} + w_{13}x_{35} + w_{21}x_{43} + w_{22}x_{44} + w_{23}x_{45} + w_{31}x_{53} + w_{32}x_{54} + w_{33}x_{55}$$

$$O = \frac{M-F+2P}{S} + 1$$

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \\ x_{41} & x_{42} & x_{43} & x_{44} & x_{45} \\ x_{51} & x_{52} & x_{53} & x_{54} & x_{55} \end{bmatrix} * \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} = \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ y_{31} & y_{32} & y_{33} \end{bmatrix}$$

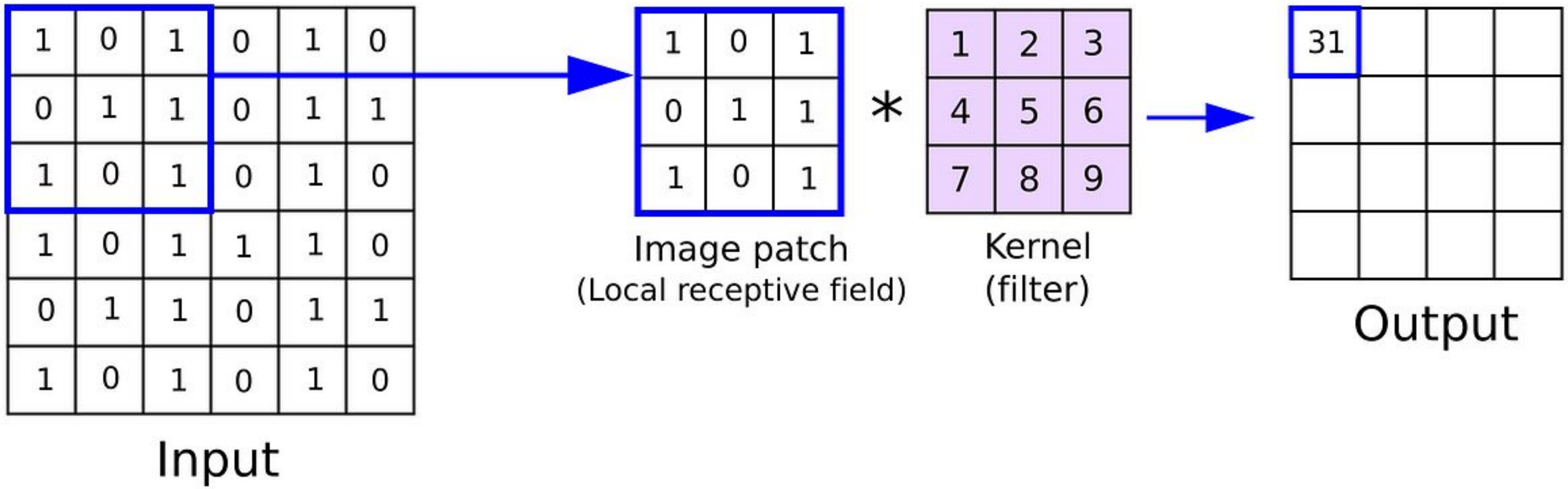
Input Image (5*5)

Kernel or Filter (3*3)

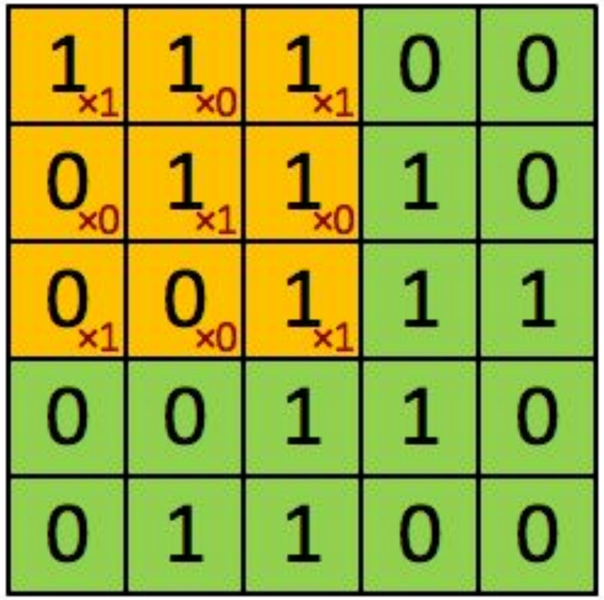
Feature Map (3*3)

Convolutional Neural Network (CNN)

$$O = \frac{M - F + 2P}{S} + 1$$

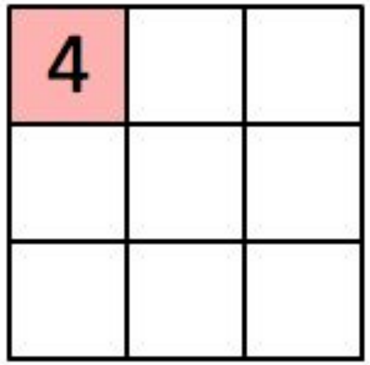


Convolutional Neural Network (CNN)



Image

$$O = \frac{M-F+2P}{S} + 1$$



Convolved Feature

1	0	1
0	1	0
1	0	1

Filter (3*3)

Convolutional Neural Network (CNN)

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

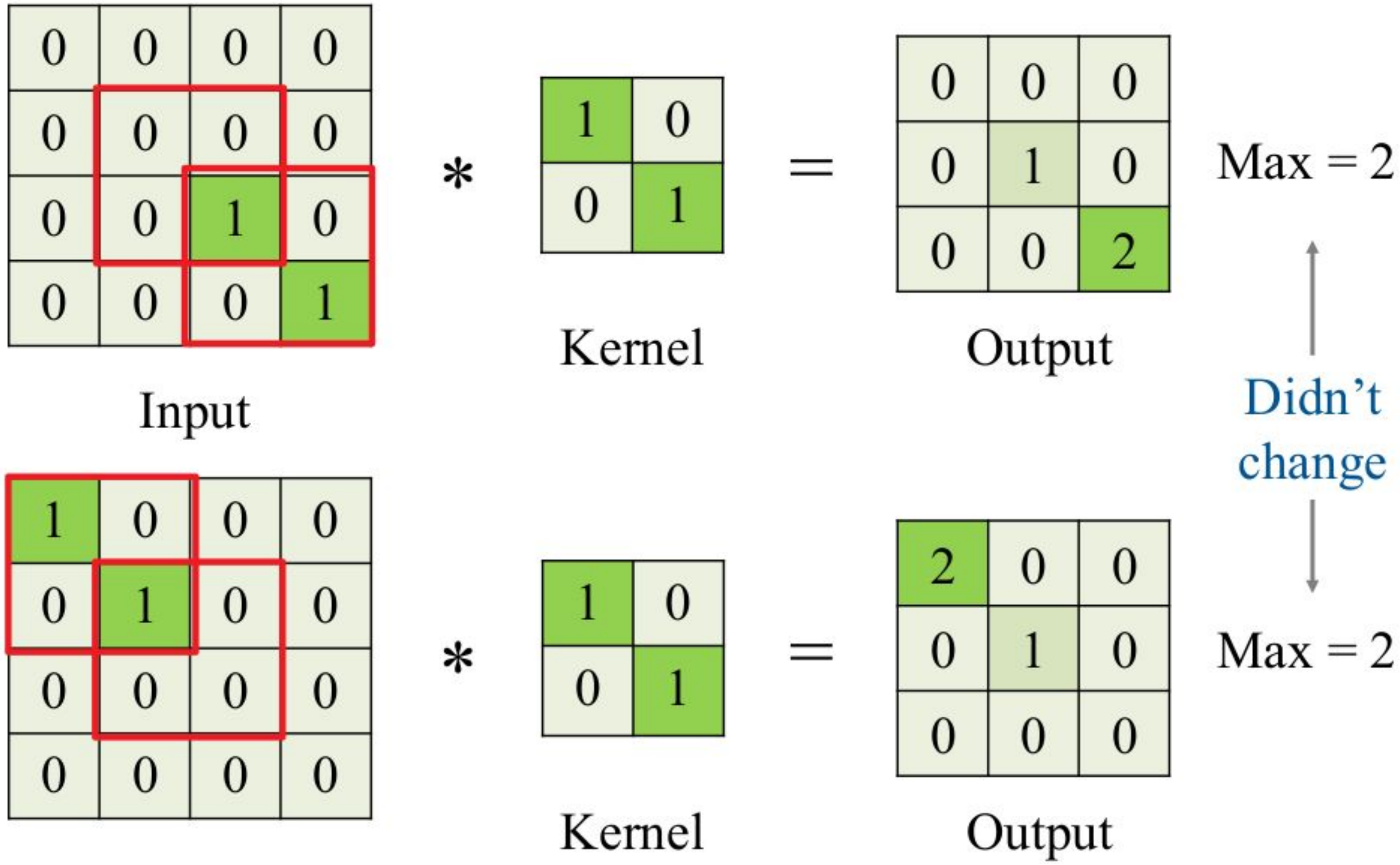
$$O = \frac{M - F + 2P}{S} + 1$$

114				

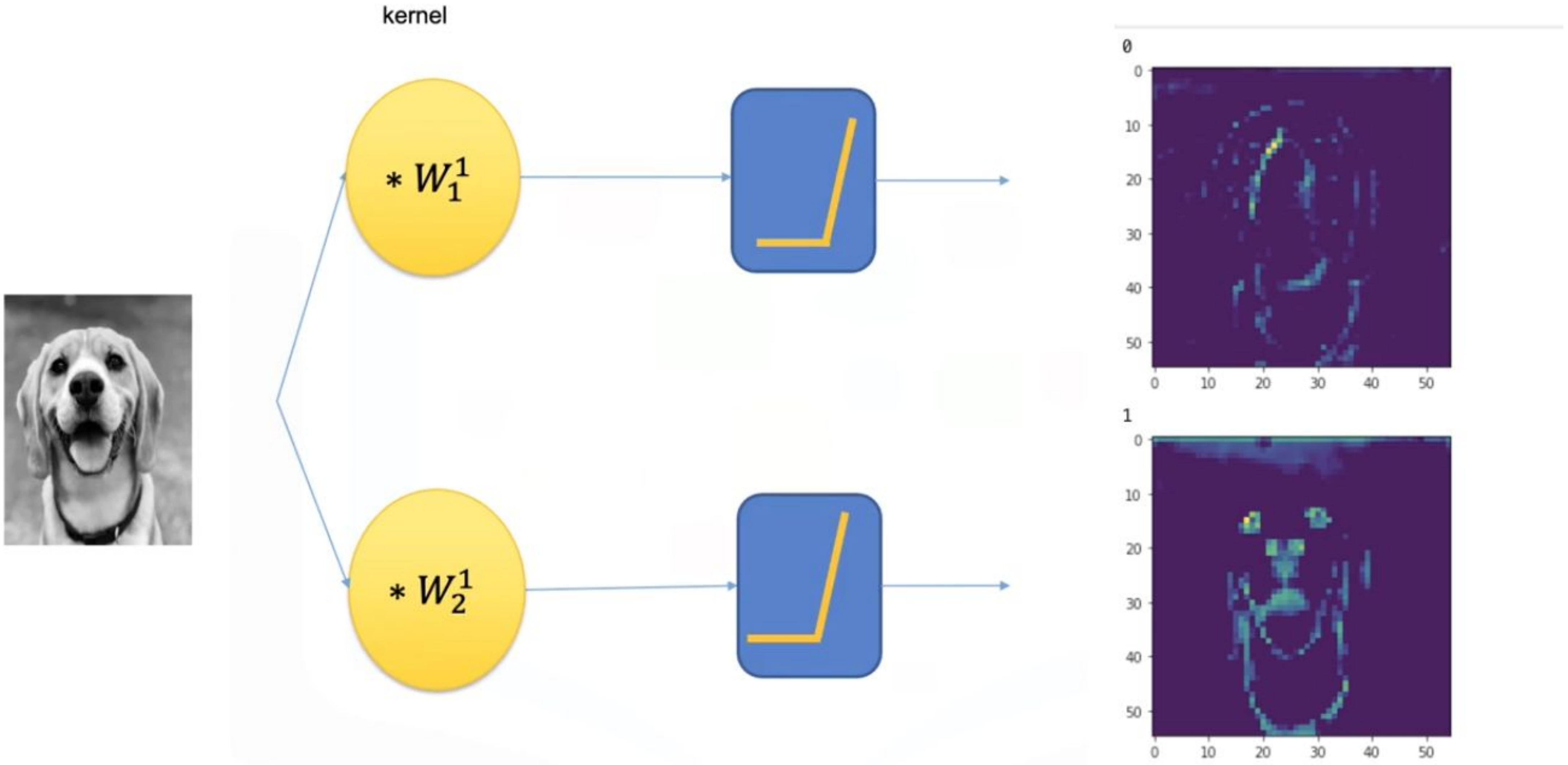
Fig: Zero-Padding in CNN

Convolutional Neural Networks - Solving translation invariance

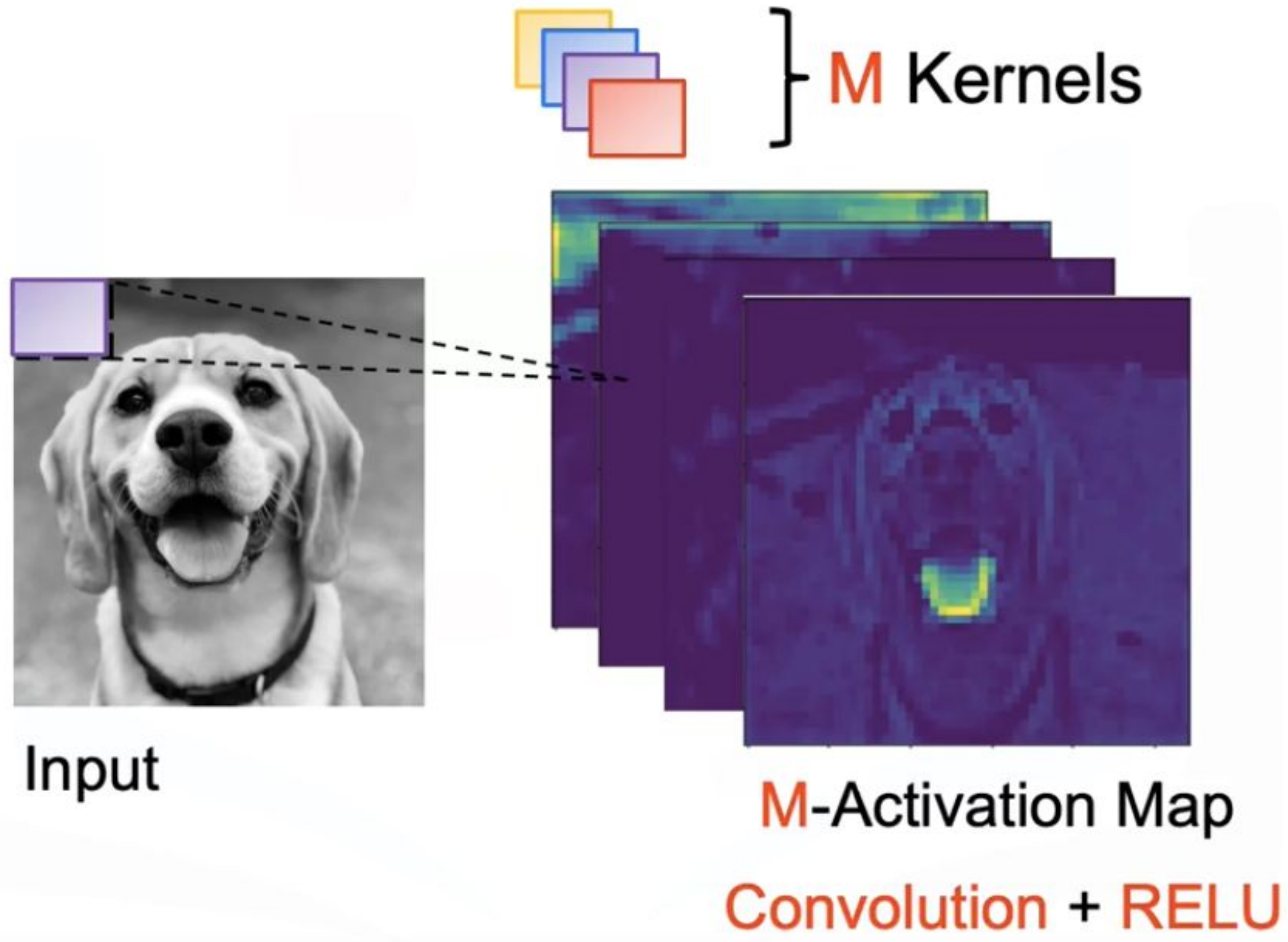
Convolution is translation equivariant



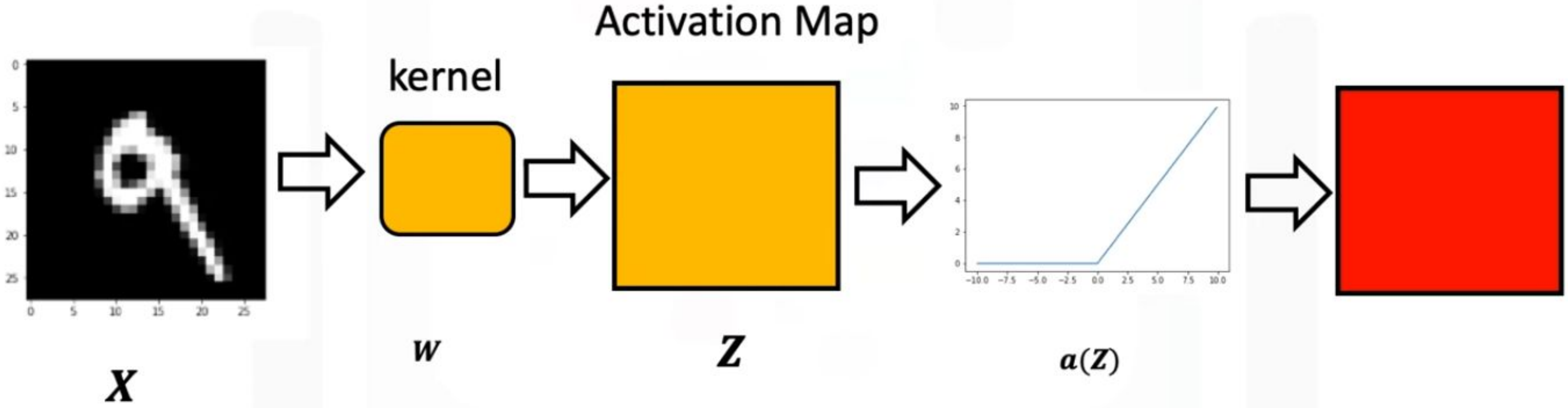
Convolutional Neural Networks - Filter/kernel is trainable



Convolutional Neural Networks - Filter/kernel is trainable



Convolutional Neural Networks - Forward Pass

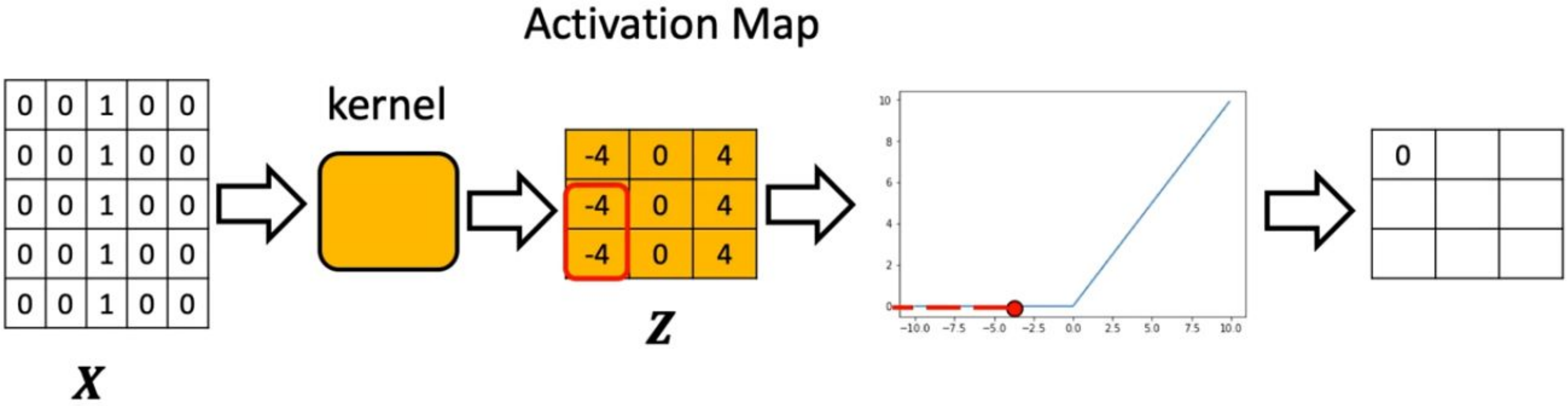


Activation Function

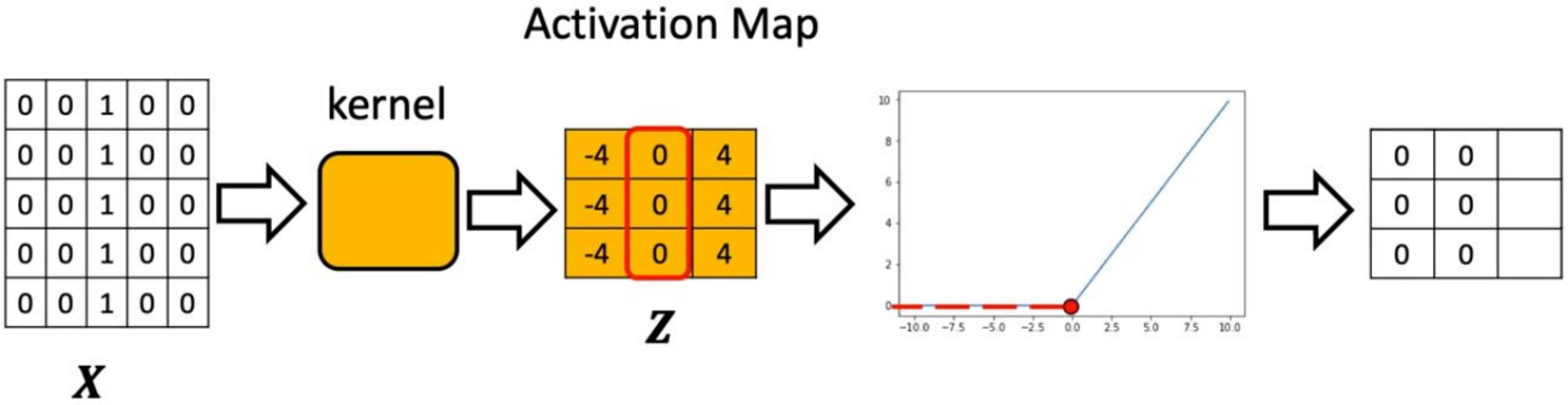
$$Z = W * X + b$$

$$A = a(Z)$$

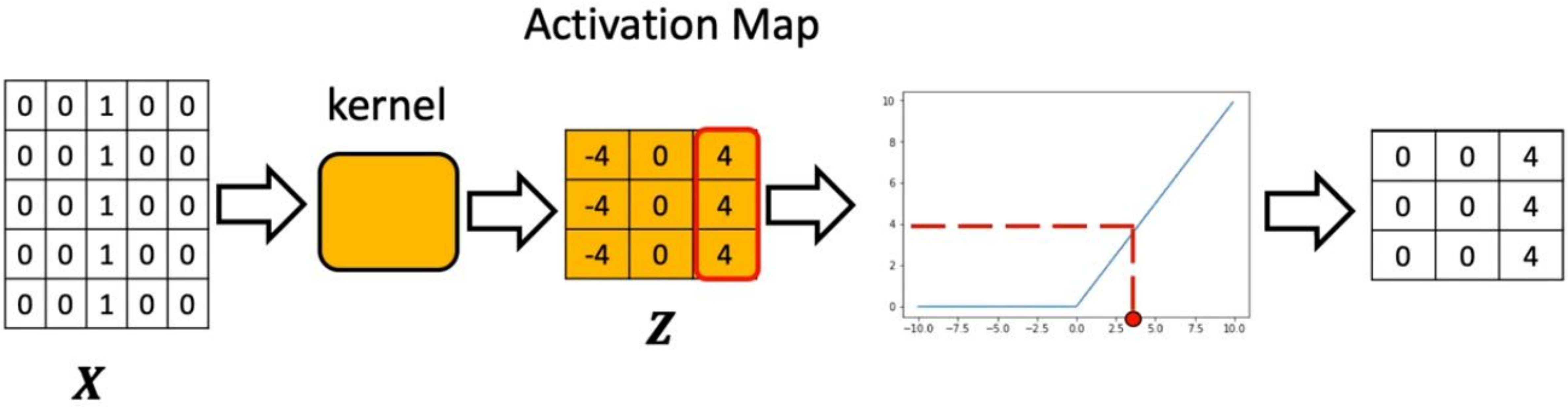
Convolutional Neural Networks - Forward Pass



Convolutional Neural Networks - Forward Pass



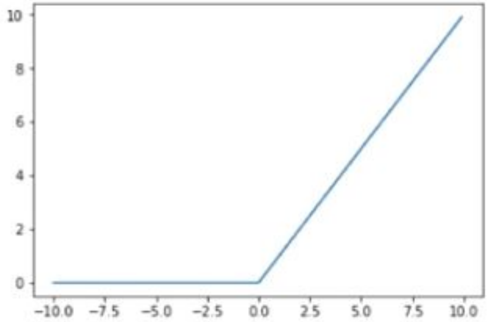
Convolutional Neural Networks - Forward Pass



Convolutional Neural Networks - Forward Pass

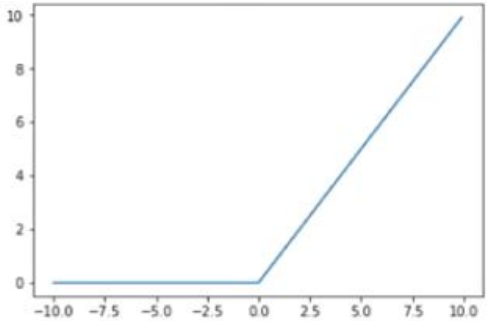
Activation Map 3 Channels

-1	-2	-1
0	0	0
1	2	1



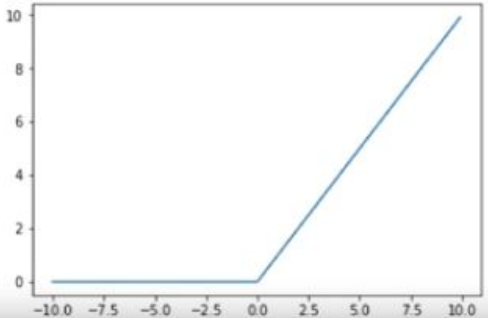
0	0	0
0	0	0
1	2	1

1	1	1
1	0	1
1	1	1



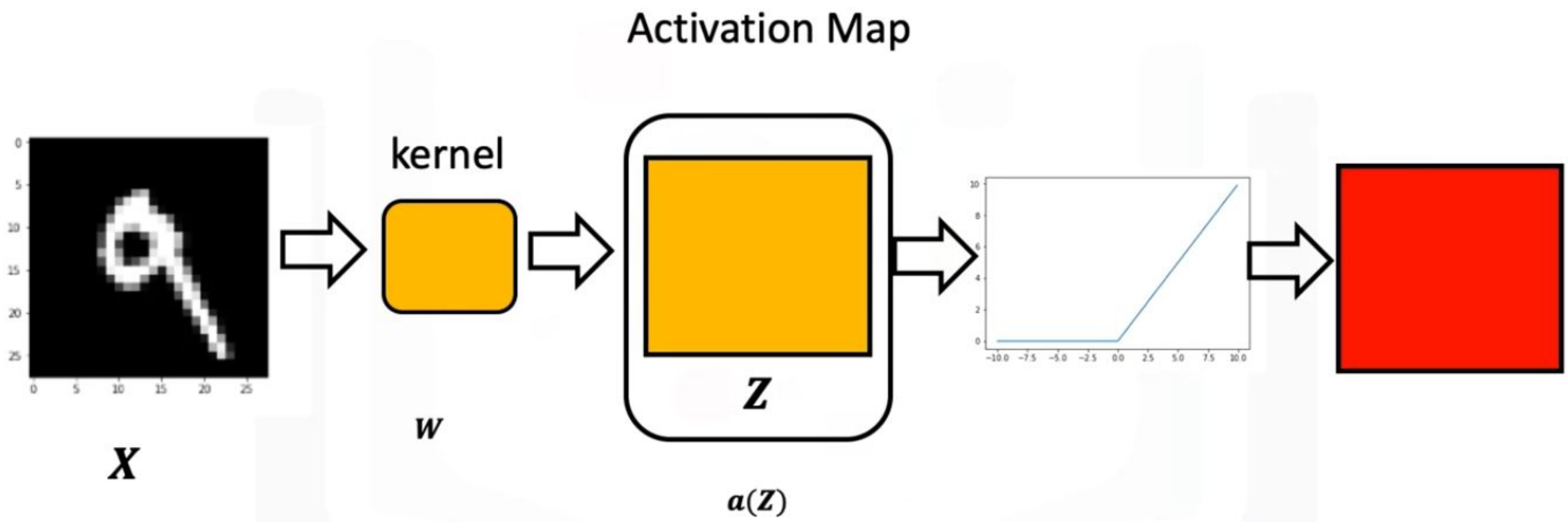
1	1	1
1	0	1
1	1	1

0.5	0.5	0.5
0.5	1	0.5
0.5	0.5	0.5



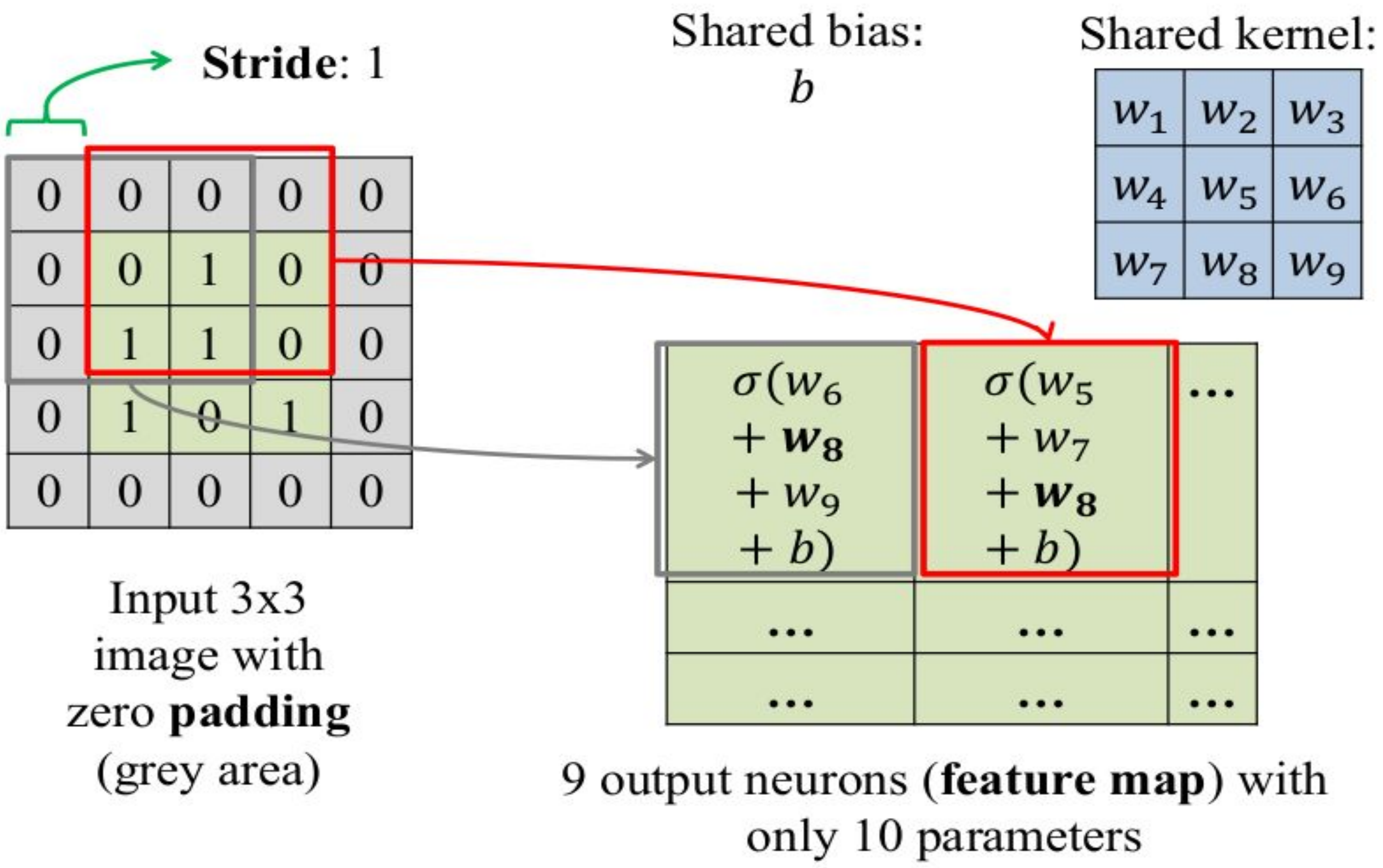
0.5	0.5	0.5
0.5	1	0.5
0.5	0.5	0.5

Convolutional Neural Networks - Forward Pass



Convolutional Neural Networks - in backpropagation weights are shared

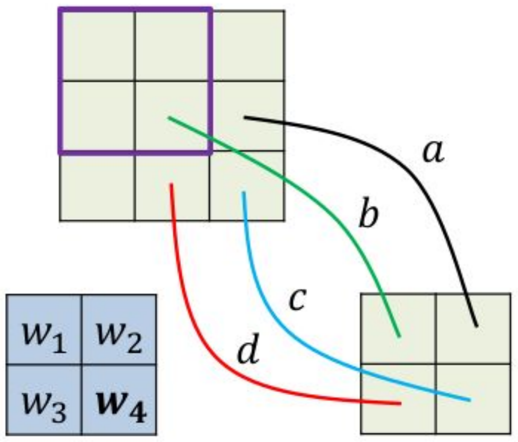
Convolutional layer in neural network



Convolutional Neural Networks - Backpropagation

Backpropagation for CNN

Gradients are first calculated as if the kernel weights were not shared:



$$a = a - \gamma \frac{\partial L}{\partial a} \qquad b = b - \gamma \frac{\partial L}{\partial b}$$

$$c = c - \gamma \frac{\partial L}{\partial c} \qquad d = d - \gamma \frac{\partial L}{\partial d}$$

$$w_4 = w_4 - \gamma \left(\frac{\partial L}{\partial a} + \frac{\partial L}{\partial b} + \frac{\partial L}{\partial c} + \frac{\partial L}{\partial d} \right)$$

Gradients of the same shared weight are summed up!

In a CNN, the same kernel weight (like w_4 in the example) is used multiple times as the kernel slides across the input, creating different outputs (a, b, c, d).
The image shows two important steps in backpropagation: First Step:

- Initially, gradients are calculated "as if the weights were not shared"
- Each position where w_4 is applied gets its own gradient: $\partial L/\partial a$, $\partial L/\partial b$, $\partial L/\partial c$, $\partial L/\partial d$

Second Step:

- Since w_4 is actually a shared weight, all these gradients must be combined
- The final gradient for w_4 is the sum of all individual gradients: $\partial L/\partial w_4 = \partial L/\partial a + \partial L/\partial b + \partial L/\partial c + \partial L/\partial d$

The update rule shown for w_4 reflects this summation: $w_4 = w_4 - \gamma (\partial L/\partial a + \partial L/\partial b + \partial L/\partial c + \partial L/\partial d)$ where γ is the learning rate

Pooling Layer

Convolutional Neural Network - Pooling Layer

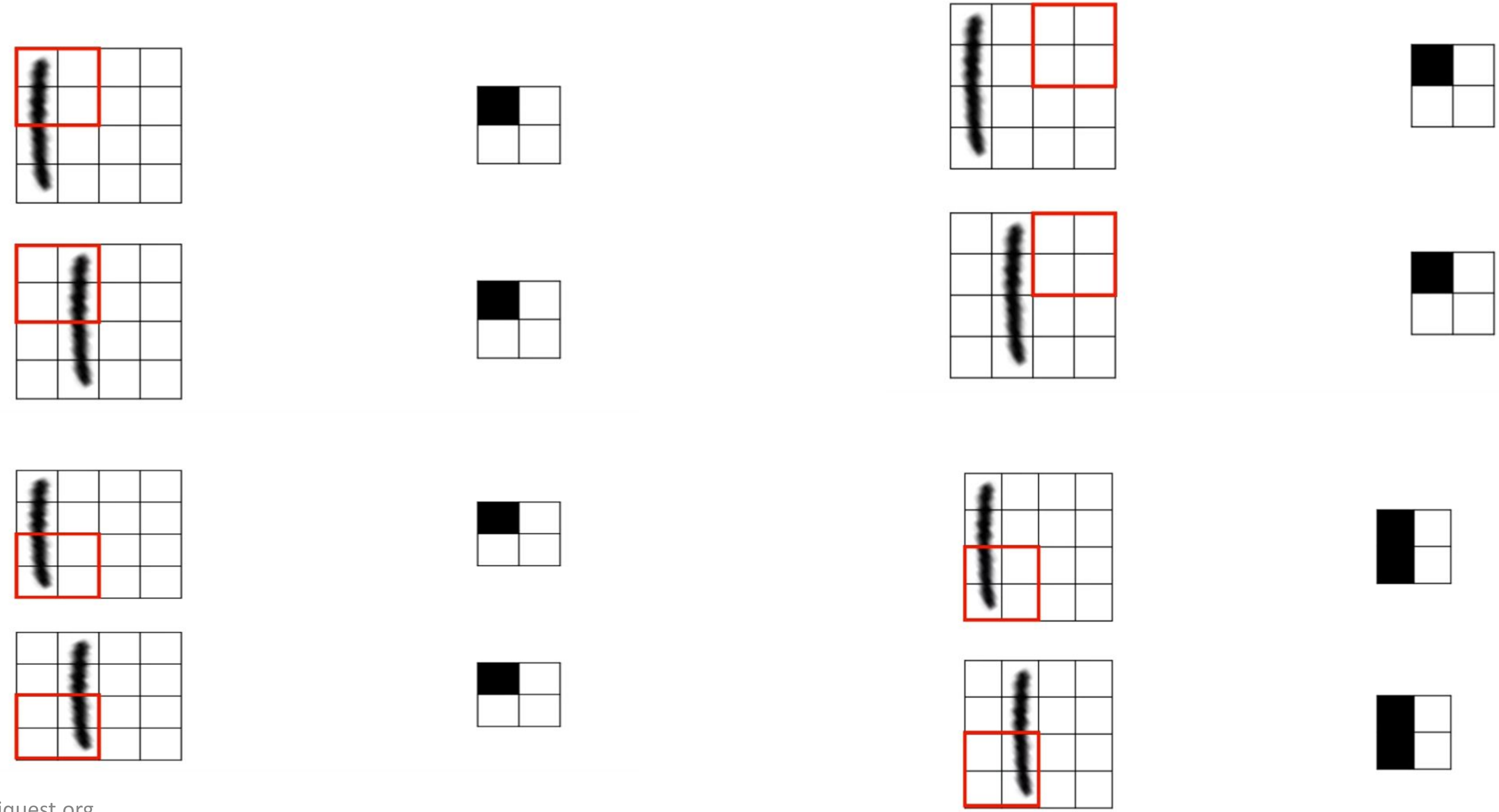
In a convolutional neural network (CNN), the pooling layer plays a crucial role in **reducing** the spatial dimensions (width and height) of the input volume for the subsequent layers.

Pooling layers in convolutional neural networks (CNNs) are important for three main reasons:

- **Efficiency:** They reduce the spatial dimensions of feature maps, lowering computational costs and the number of parameters.
- **Feature Invariance:** Pooling provides robustness to minor variations in the input, helping the network to recognize features regardless of their position.
- **Generalization:** By simplifying the features, pooling helps prevent overfitting, enhancing the model's ability to perform well on new, unseen data.

CNN - Pooling Layer - Solve Feature Invariance

Max Pooling work as Feature Invariance



Convolutional Neural Network - Pooling Layer Types

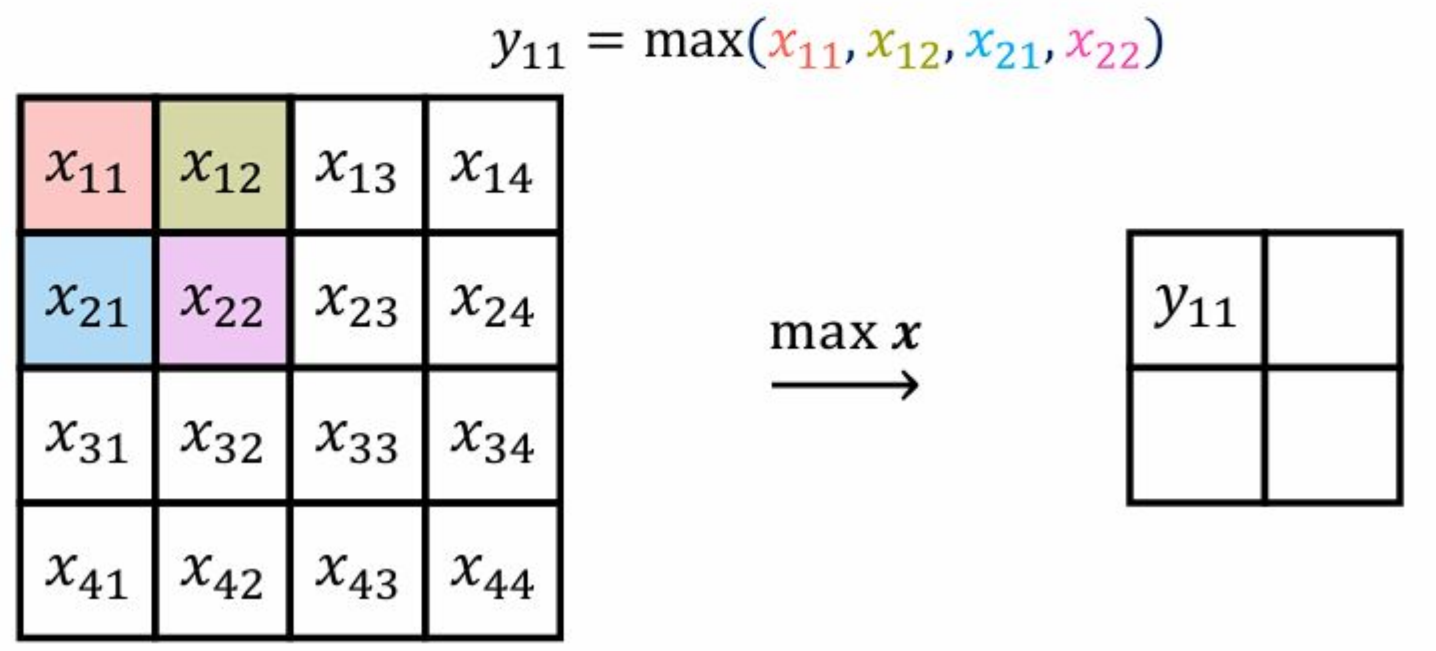
In a convolutional neural network (CNN), the pooling layer plays a crucial role in **reducing** the spatial dimensions (width and height) of the input volume for the subsequent layers.

The most common types of pooling are:

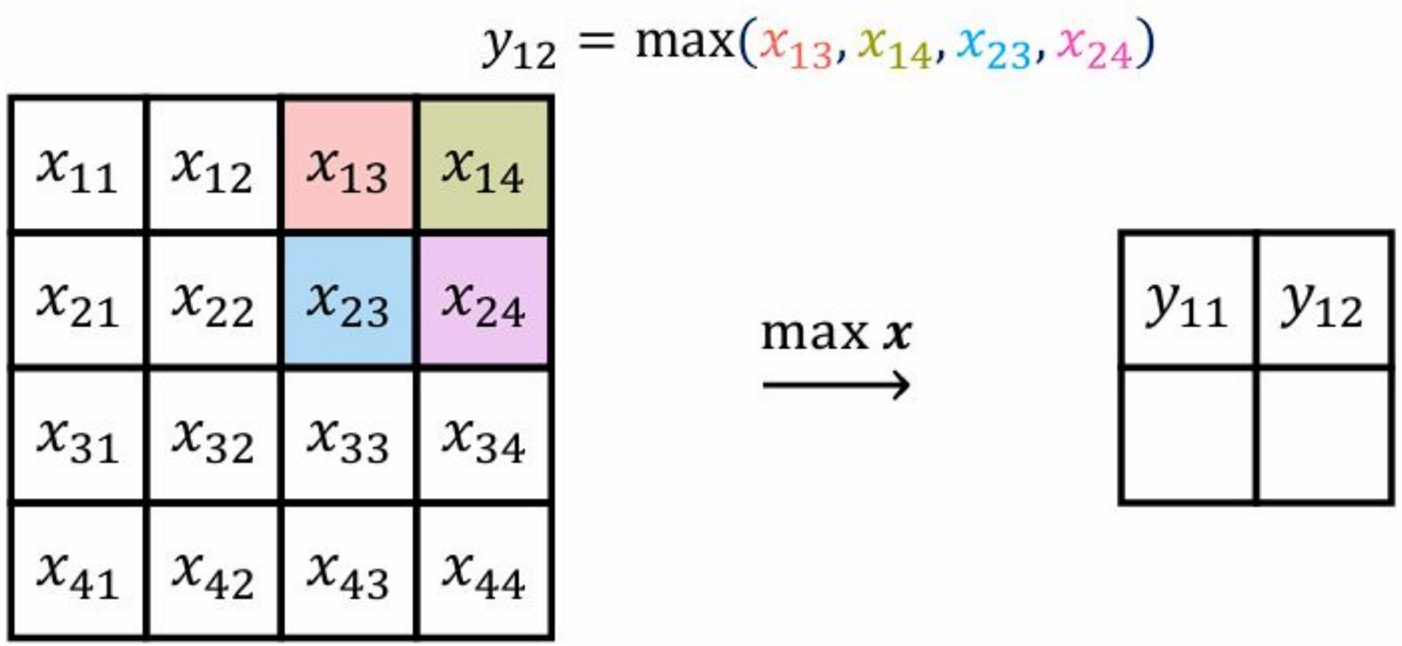
- **Max/Min Pooling:** Outputs the maximum/minimum value from each patch of the feature map.
- **Average Pooling:** Outputs the average value from each patch.

Convolutional Neural Network - Pooling Layer - Max Pooling

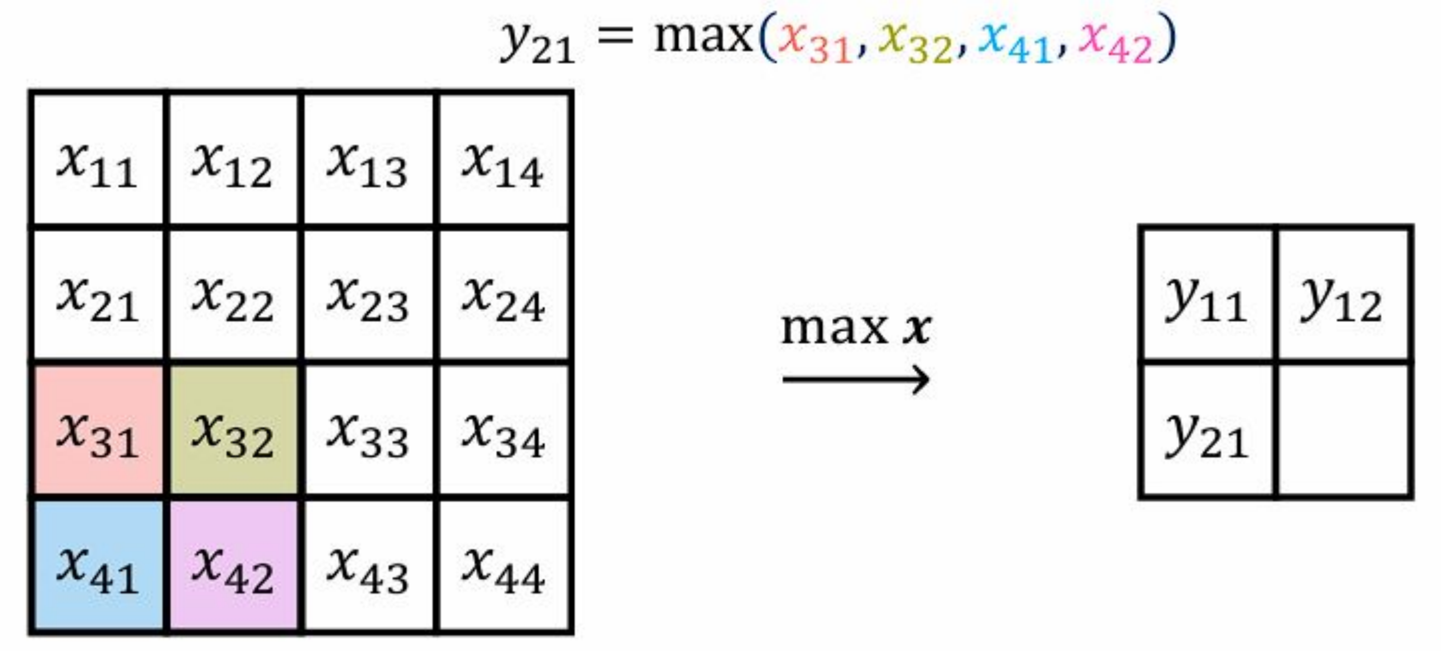
The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.



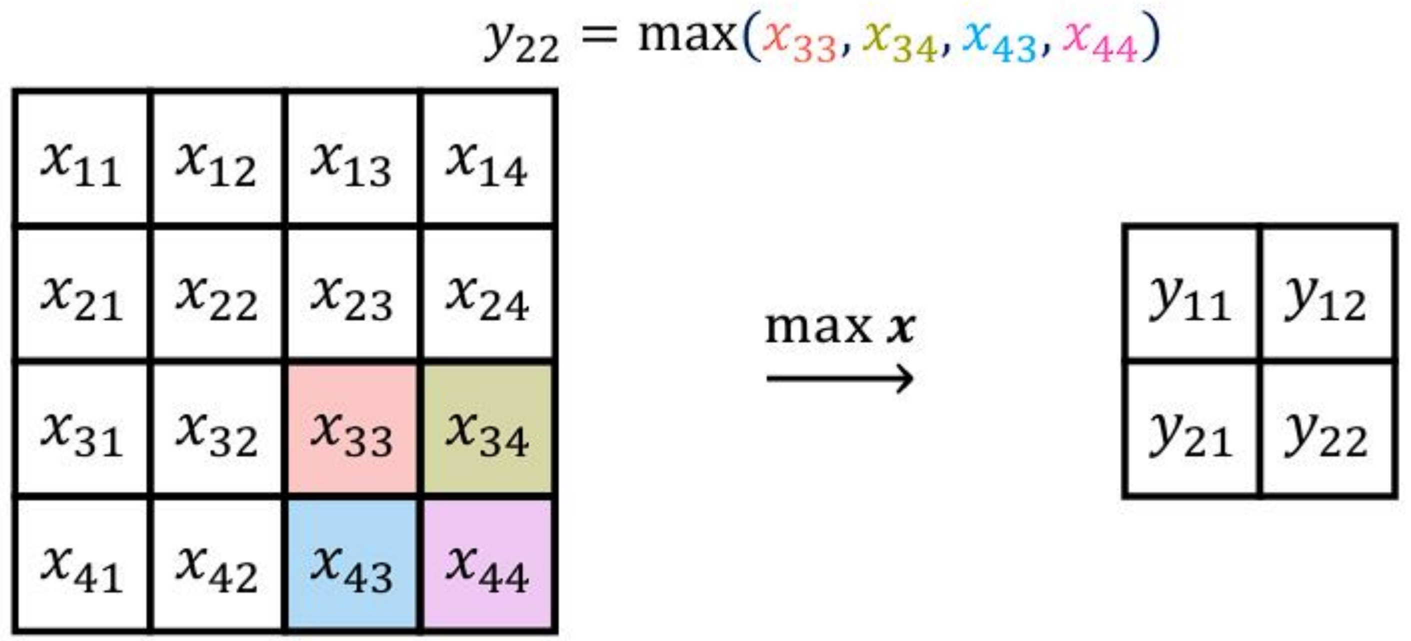
The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.



The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.



The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.



Convolutional Neural Network - Pooling Layer - Max Pooling

The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.

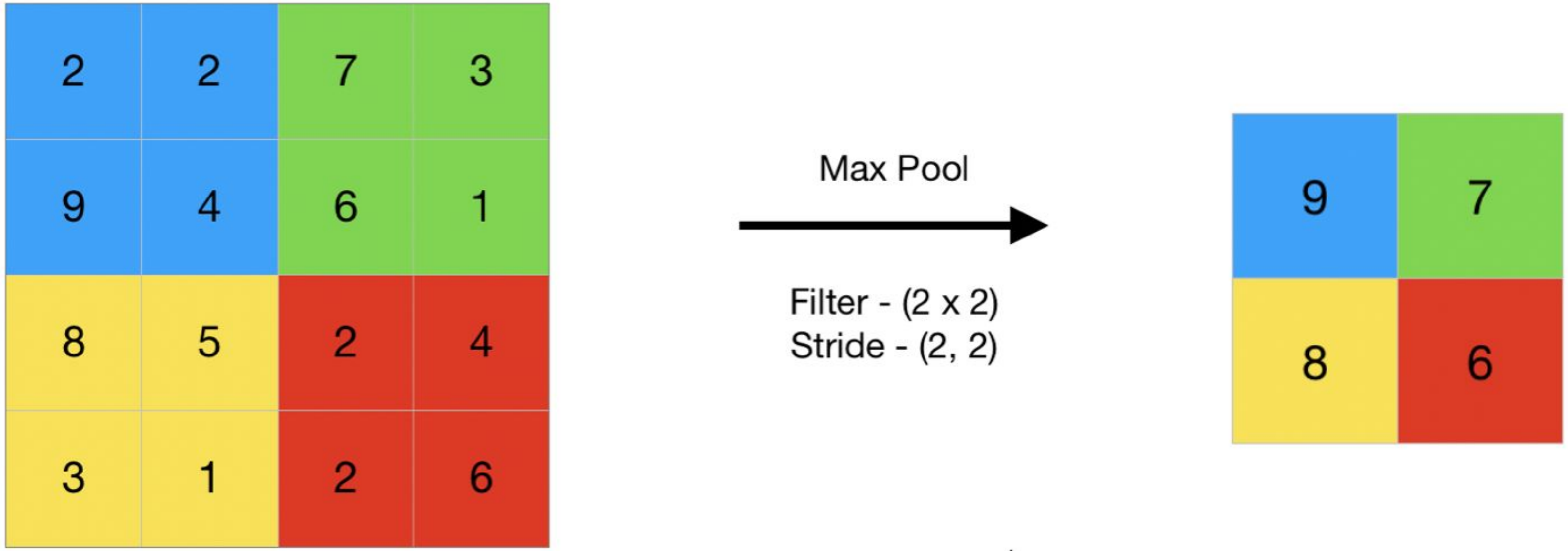


Fig: Max Pooling in CNN

The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.

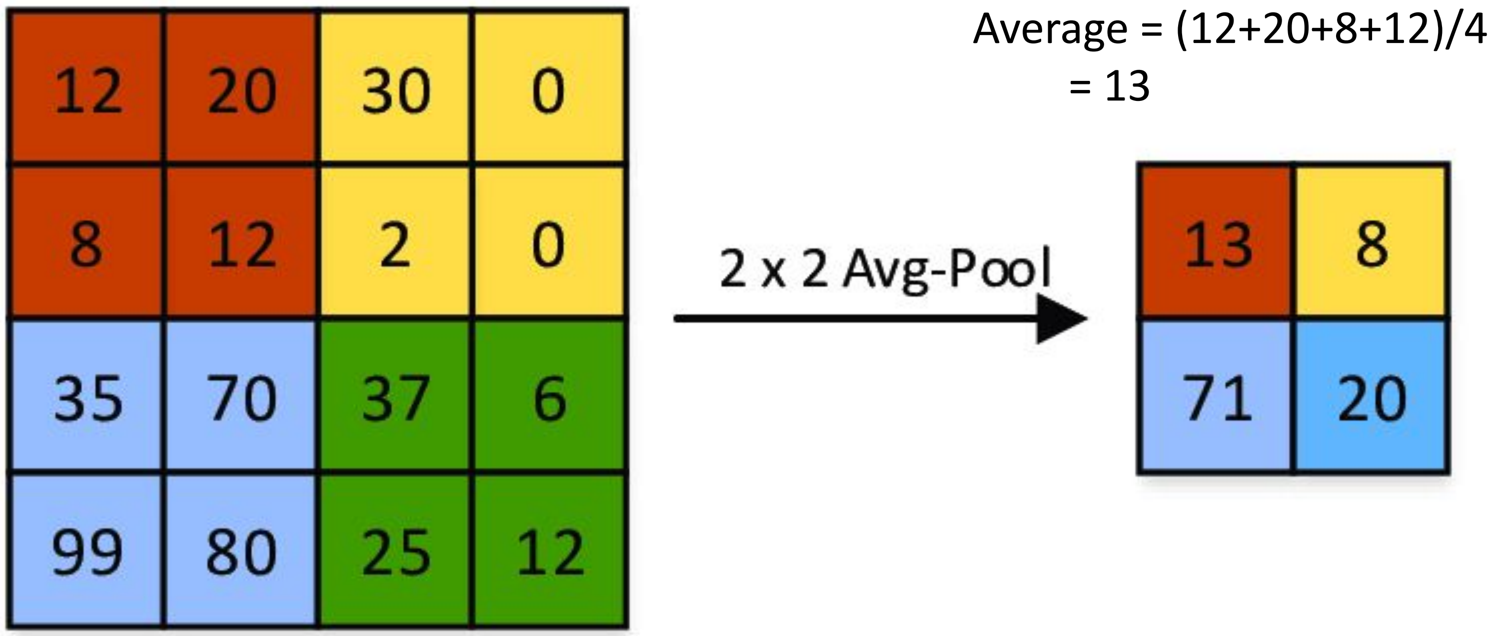
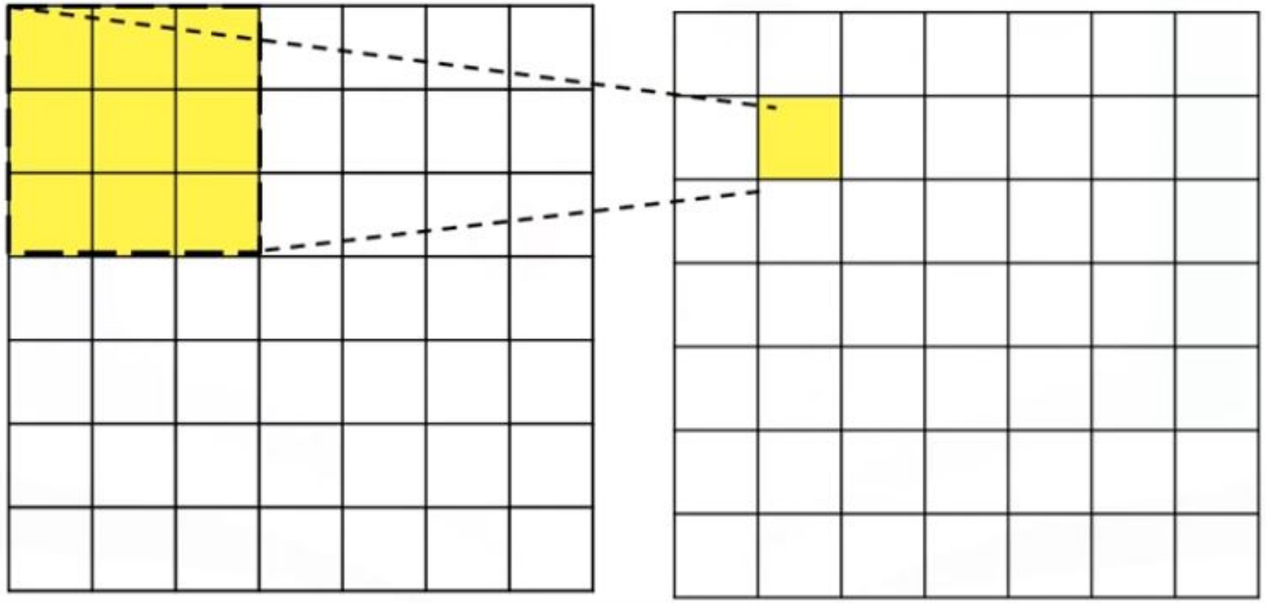


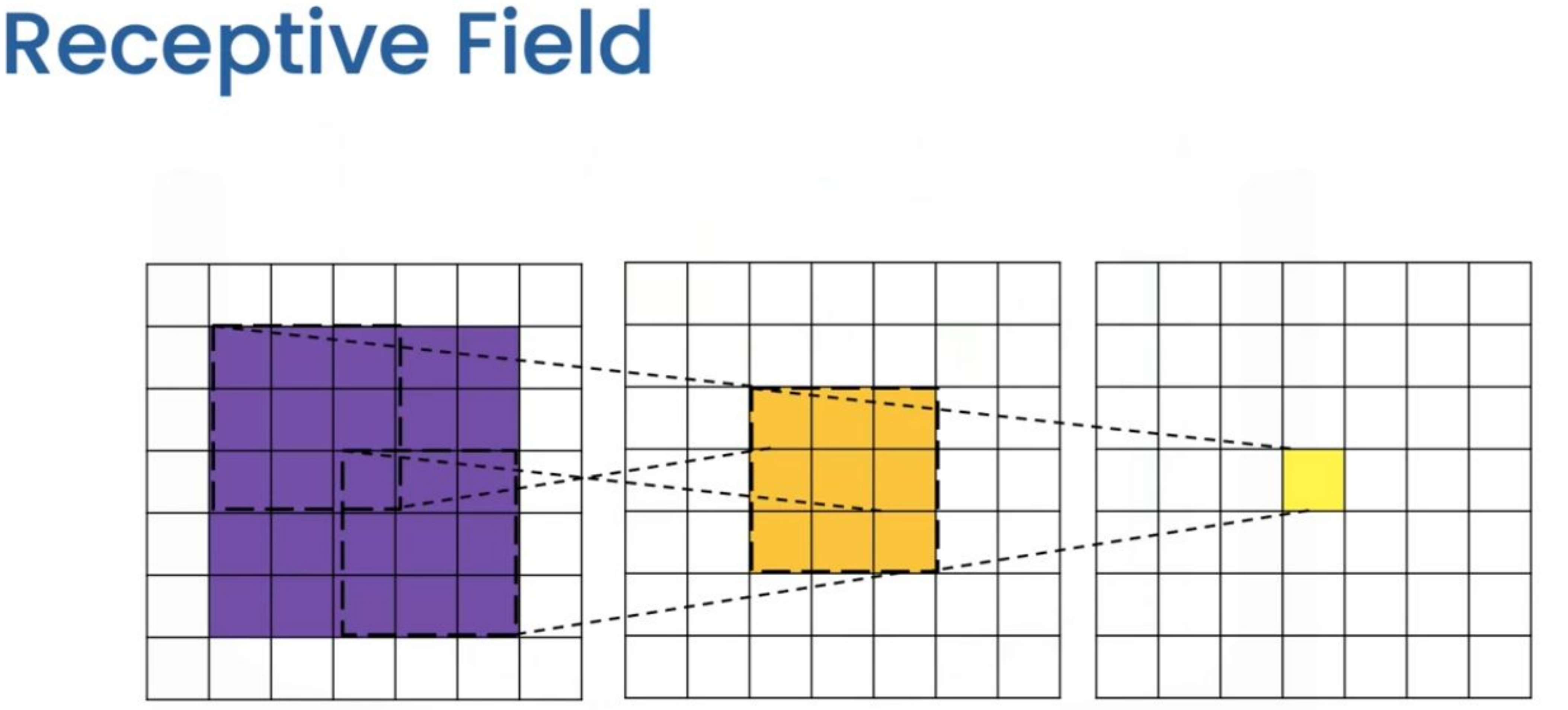
Fig: Average Pooling in CNN

Receptive Field

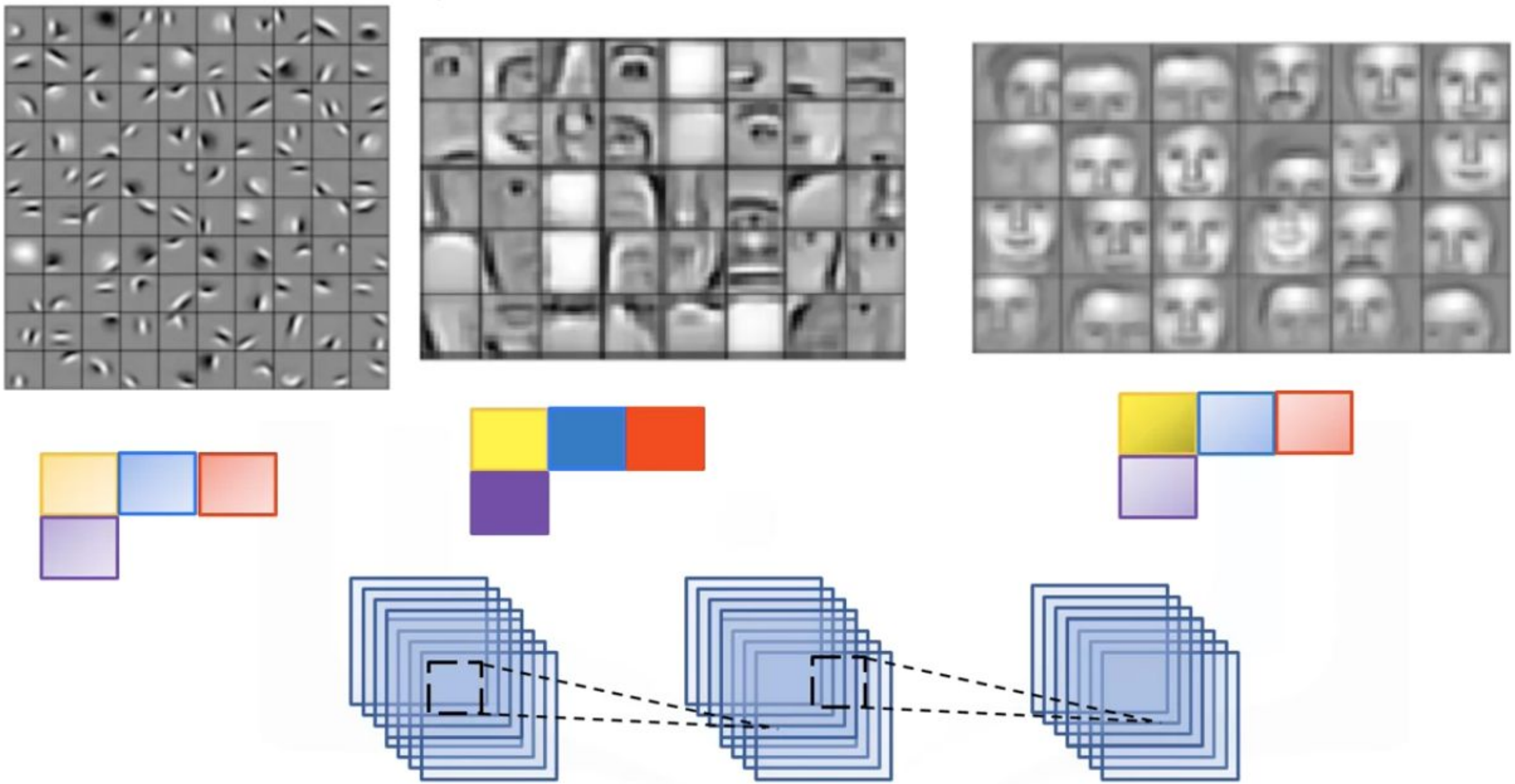
- **Receptive Field is the size of the region in the input that produces a pixel value in the activation Map**



Increasing Layers increase the Receptive field



Increasing Layers increase the Receptive field



Convolutional Neural Network - Default Architecture

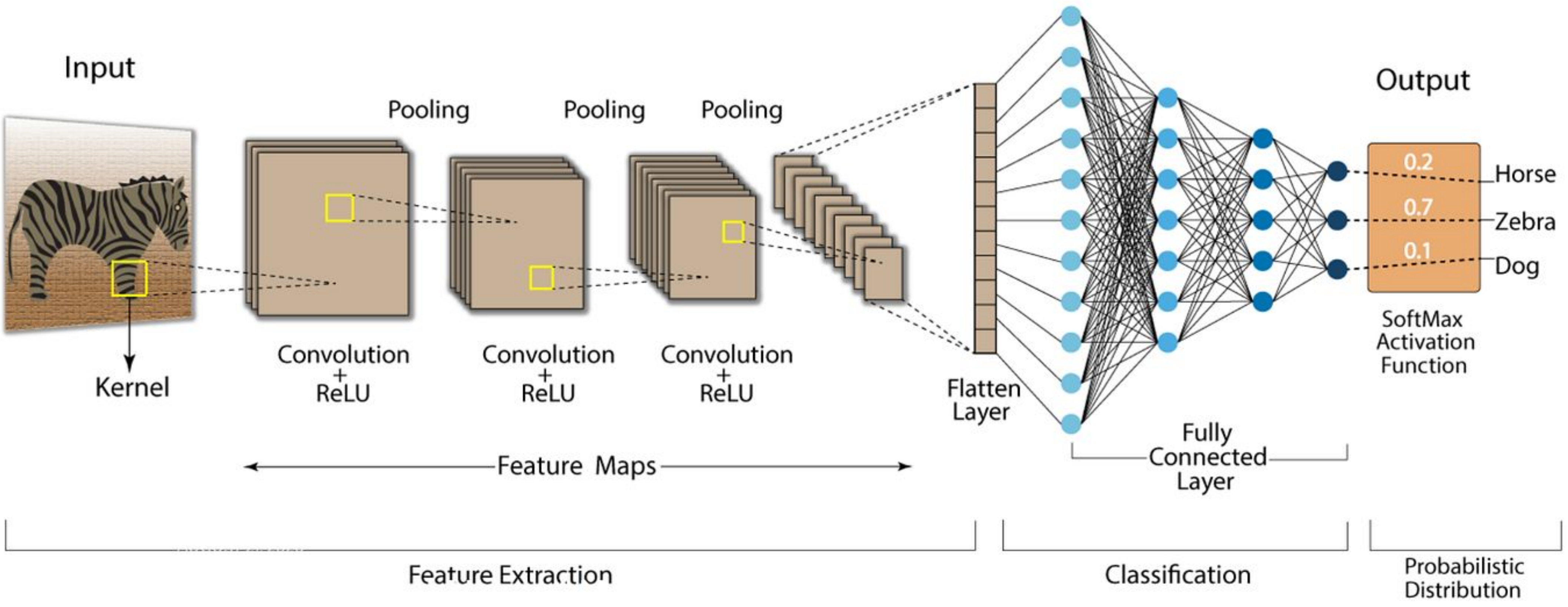


Fig: CNN Architecture

Convolutional Neural Network (CNN)

Output Dimensions | Parameters

CNN: Dimensions

Output dimensions describe the **size of the data tensor that results from a layer** within a CNN. This typically includes:

- **Width** and **Height**: These dimensions can change based on the type of layer (convolutional, pooling), the kernel size, the stride, and the padding used.

- **Depth** (or Channels): This dimension often changes in convolutional layers depending on the number of filters used. It stays the same through pooling layers unless pooling is done separately across channels.

CNN: Dimensions

Question: Consider the input tensor of dimensions 10×10×10 where the last dimension is the channel dimension. The following operations are applied:

1. 3×3 convolution (40 channels) with stride 1 and padding 1 for each dimension.
2. ReLU activation.
3. 3×3 max pooling with stride 1 and padding 1 for each dimension.
4. 3×3 convolution (20 channels) with stride 1 and padding 1 for each dimension.
5. ReLU activation.
6. 2×2 max pooling with stride 2 and padding 1 for each dimension.

What are the dimensions of the output tensor after each operation? Report the name of the operation and the output dimensions as width x height x channels. There are 6 operations in total.

CNN: Dimensions

$$\text{Output Dimension} = \left\lfloor \frac{\text{Input Dimension} - \text{Kernel Size} + 2 \times \text{Padding}}{\text{Stride}} \right\rfloor + 1$$

Dimensions After Each Operation:

1. **After 3×3 Convolution:** $10 \times 10 \times 40$
2. **After ReLU Activation:** $10 \times 10 \times 40$
3. **After 3×3 Max Pooling:** $10 \times 10 \times 40$
4. **After 3×3 Convolution:** $10 \times 10 \times 20$
5. **After ReLU Activation:** $10 \times 10 \times 20$
6. **After 2×2 Max Pooling:** $6 \times 6 \times 20$

CNN: Dimensions

Question: Consider the input tensor of dimensions **10×10×10** where the last dimension is the channel dimension. The following operations are applied:

1. **3×3 convolution (40 channels) with stride 1 and padding 1 for each dimension.**
2. ReLU activation.
3. 3×3 max pooling with stride 1 and padding 1 for each dimension.
4. 3×3 convolution (20 channels) with stride 1 and padding 1 for each dimension.
5. ReLU activation.
6. 2×2 max pooling with stride 2 and padding 1 for each dimension.

What are the dimensions of the output tensor after each operation? Report the name of the operation and the output dimensions as width x height x channels. There are 6 operations in total.

1. 3×3 Convolution (40 channels) with stride 1 and padding 1

- **Kernel Size:** 3×3
- **Stride:** 1
- **Padding:** 1
- **Input Dimensions:** $10 \times 10 \times 10$

$$\text{Output Width} = \left\lfloor \frac{10-3+2\times 1}{1} \right\rfloor + 1 = 10$$
$$\text{Output Height} = \left\lfloor \frac{10-3+2\times 1}{1} \right\rfloor + 1 = 10$$
$$\text{Output Channels} = 40$$

Dimension: 10 x 10 x 40

CNN: Dimensions

Question: Consider the input tensor of dimensions **10×10×10** where the last dimension is the channel dimension. The following operations are applied:

1. 3×3 convolution (40 channels) with stride 1 and padding 1 for each dimension.
2. **ReLU activation.**
3. **3×3 max pooling with stride 1 and padding 1 for each dimension.**
4. 3×3 convolution (20 channels) with stride 1 and padding 1 for each dimension.
5. ReLU activation.
6. 2×2 max pooling with stride 2 and padding 1 for each dimension.

What are the dimensions of the output tensor after each operation? Report the name of the operation and the output dimensions as width x height x channels. There are 6 operations in total.

CNN: Dimensions

2. ReLU Activation

- Does not change dimensions.
- **Output Dimensions:** $10 \times 10 \times 40$

3. 3×3 Max Pooling with stride 1 and padding 1

- **Kernel Size:** 3×3
- **Stride:** 1
- **Padding:** 1

$$\text{Output Width} = \left\lfloor \frac{10-3+2\times 1}{1} \right\rfloor + 1 = 10$$
$$\text{Output Height} = \left\lfloor \frac{10-3+2\times 1}{1} \right\rfloor + 1 = 10$$
$\text{Output Channels} = 40$ (Channel dimension remains unchanged in pooling)

Dimension: 10 x 10 x 40

CNN: Dimensions

Question: Consider the input tensor of dimensions **10×10×10** where the last dimension is the channel dimension. The following operations are applied:

1. 3×3 convolution (40 channels) with stride 1 and padding 1 for each dimension.
2. ReLU activation.
3. 3×3 max pooling with stride 1 and padding 1 for each dimension.
4. **3×3 convolution (20 channels) with stride 1 and padding 1 for each dimension.**
5. ReLU activation.
6. 2×2 max pooling with stride 2 and padding 1 for each dimension.

What are the dimensions of the output tensor after each operation? Report the name of the operation and the output dimensions as width x height x channels. There are 6 operations in total.

4. 3×3 Convolution (20 channels) with stride 1 and padding 1

- **Kernel Size:** 3×3
- **Stride:** 1
- **Padding:** 1
- **Input Dimensions:** $10 \times 10 \times 40$

$$\text{Output Width} = \left\lfloor \frac{10 - 3 + 2 \times 1}{1} \right\rfloor + 1 = 10$$
$$\text{Output Height} = \left\lfloor \frac{10 - 3 + 2 \times 1}{1} \right\rfloor + 1 = 10$$
$$\text{Output Channels} = 20$$

Dimension: 10 x 10 x 20

CNN: Dimensions

Question: Consider the input tensor of dimensions **10×10×10** where the last dimension is the channel dimension. The following operations are applied:

1. 3×3 convolution (40 channels) with stride 1 and padding 1 for each dimension.
2. ReLU activation.
3. 3×3 max pooling with stride 1 and padding 1 for each dimension.
4. 3×3 convolution (20 channels) with stride 1 and padding 1 for each dimension.
5. **ReLU activation.**
6. **2×2 max pooling with stride 2 and padding 1 for each dimension.**

What are the dimensions of the output tensor after each operation? Report the name of the operation and the output dimensions as width x height x channels. There are 6 operations in total.

5. ReLU Activation

- Does not change dimensions.
- **Output Dimensions:** $10 \times 10 \times 20$

6. 2×2 Max Pooling with stride 2 and padding 1

- **Kernel Size:** 2×2
- **Stride:** 2
- **Padding:** 1

$$\text{Output Width} = \left\lfloor \frac{10-2+2\times1}{2} \right\rfloor + 1 = 6$$
$$\text{Output Height} = \left\lfloor \frac{10-2+2\times1}{2} \right\rfloor + 1 = 6$$
$$\text{Output Channels} = 20 \text{ (Channel dimension remains unchanged in pooling)}$$

Dimension: 6 x 6 x 20

CNN: Counting the Parameters

Parameters reflect the model's learning capacity and complexity. More parameters can mean a more powerful model, but also one that is more prone to overfitting and is computationally more expensive to train and run.

Note: In typical Convolutional Neural Network (CNN) architectures, the **convolutional layers** are primarily responsible for adding parameters, which are the learnable weights and biases of the model.

CNN: Parameter Count

Question: Consider the input tensor of dimensions **10×10×10** where the last dimension is the channel dimension. The following operations are applied:

1. **3×3 convolution (40 channels) with stride 1 and padding 1 for each dimension.**
2. ReLU activation.
3. 3×3 max pooling with stride 1 and padding 1 for each dimension.
4. 3×3 convolution (20 channels) with stride 1 and padding 1 for each dimension.
5. ReLU activation.
6. 2×2 max pooling with stride 2 and padding 1 for each dimension.

What is the total parameter count?

CNN: Parameter Count

$$\text{Number of Parameters} = (\text{Kernel Height} \times \text{Kernel Width} \times \text{Input Channels} + 1) \times \text{Output Channels}$$

First 3x3 Convolution (40 channels):

- **Kernel Height**: 3
- **Kernel Width**: 3
- **Input Channels**: 10 (initial input channels)
- **Output Channels**: 40

$$\text{Parameters} = (3 \times 3 \times 10 + 1) \times 40 = (90 + 1) \times 40 = 3640$$

CNN: Parameter Count

Question: Consider the input tensor of dimensions **10×10×10** where the last dimension is the channel dimension. The following operations are applied:

1. 3×3 convolution (40 channels) with stride 1 and padding 1 for each dimension.
2. ReLU activation.
3. 3×3 max pooling with stride 1 and padding 1 for each dimension.
4. **3×3 convolution (20 channels) with stride 1 and padding 1 for each dimension.**
5. ReLU activation.
6. 2×2 max pooling with stride 2 and padding 1 for each dimension.

What is the total parameter count?

CNN: Parameter Count

$$\text{Number of Parameters} = (\text{Kernel Height} \times \text{Kernel Width} \times \text{Input Channels} + 1) \times \text{Output Channels}$$

Second 3x3 Convolution (20 channels):

- The input to this layer is the output of the first max pooling layer, which maintains 40 channels.
- **Output Channels**: 20

$$\text{Parameters} = (3 \times 3 \times 40 + 1) \times 20 = (360 + 1) \times 20 = 7220$$

Total Parameters

Now, to find the total number of parameters in the CNN:

$$\text{Total Parameters} = 3640(\text{First Conv}) + 7220(\text{Second Conv}) = 10860$$

Thus, the CNN has a total of 10,860 parameters, solely from the convolutional layers as pooling layers and ReLU activations do not add any learnable parameters.

CNN: Parameter Count

Question: Consider the input tensor of dimensions **10×10×10** where the last dimension is the channel dimension. The following operations are applied:

1. 3×3 convolution (40 channels) with stride 1 and padding 1 for each dimension.
2. ReLU activation.
3. 3×3 max pooling with stride 1 and padding 1 for each dimension.
4. 3×3 convolution (20 channels) with stride 1 and padding 1 for each dimension.
5. ReLU activation.
6. 2×2 max pooling with stride 2 and padding 1 for each dimension.

What is the total parameter count?

If you are adding another convolutional layer with 10 output channels following the previous convolutional layer that had 20 output channels, you would again use the formula to calculate the parameters. This layer also uses 3x3 kernels:

Parameters for the Third Convolutional Layer:

- **Kernel Size:** 3×3
- **Input Channels:** 20 (output of the second convolution layer)
- **Output Channels:** 10
- **Bias:** 1 bias per output channel (10 in total)

Formula and Calculation:

$$\text{Number of Parameters} = (\text{Kernel Height} \times \text{Kernel Width} \times \text{Input Channels} + 1) \times \text{Output Channels}$$

$$\text{Number of Parameters} = (3 \times 3 \times 20 + 1) \times 10$$

$$\text{Number of Parameters} = (180 + 1) \times 10$$

$$\text{Number of Parameters} = 181 \times 10$$

$$\text{Number of Parameters} = 1810$$