# Convolutional Networks

## Motivation, working with images, trainable kernels

Machine Learning and Data Mining, 2024

Majid Sohrabi
National Research University Higher School of Economics

November 20, 2024

# How to work with image-like data?

# Working with images

Extreemly high-dimensional input

- E.g. even a small 640x480 color image would make up almost 1M input features (pixel brightness levels in R, G and B)
- So a fully-connected hidden representation with just 100 units would require 100M parameters

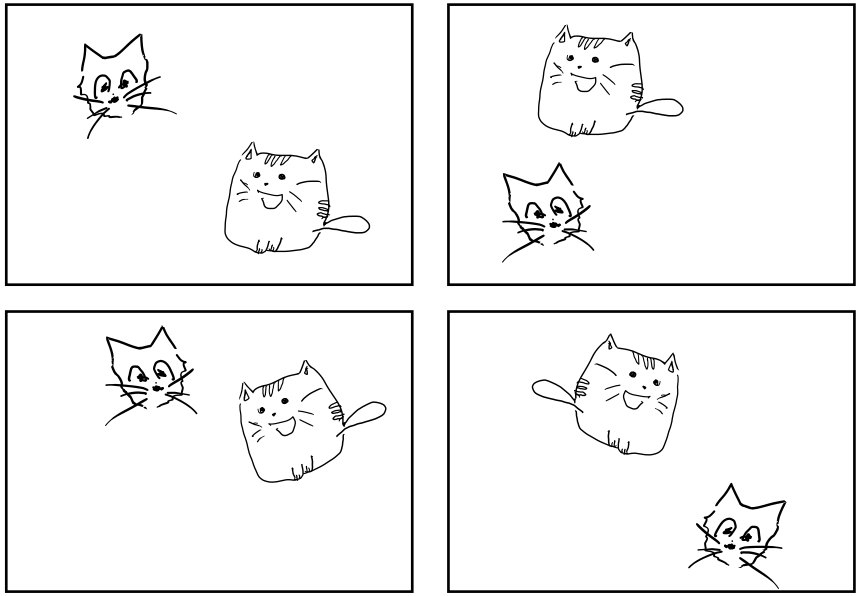
# Working with images

Extreemly high-dimensional input

- E.g. even a small 640x480 color image would make up almost 1M input features (pixel brightness levels in R, G and B)
- So a fully-connected hidden representation with just 100 units would require 100M parameters

Is quite data-hungry to train when using fully-connected layers:

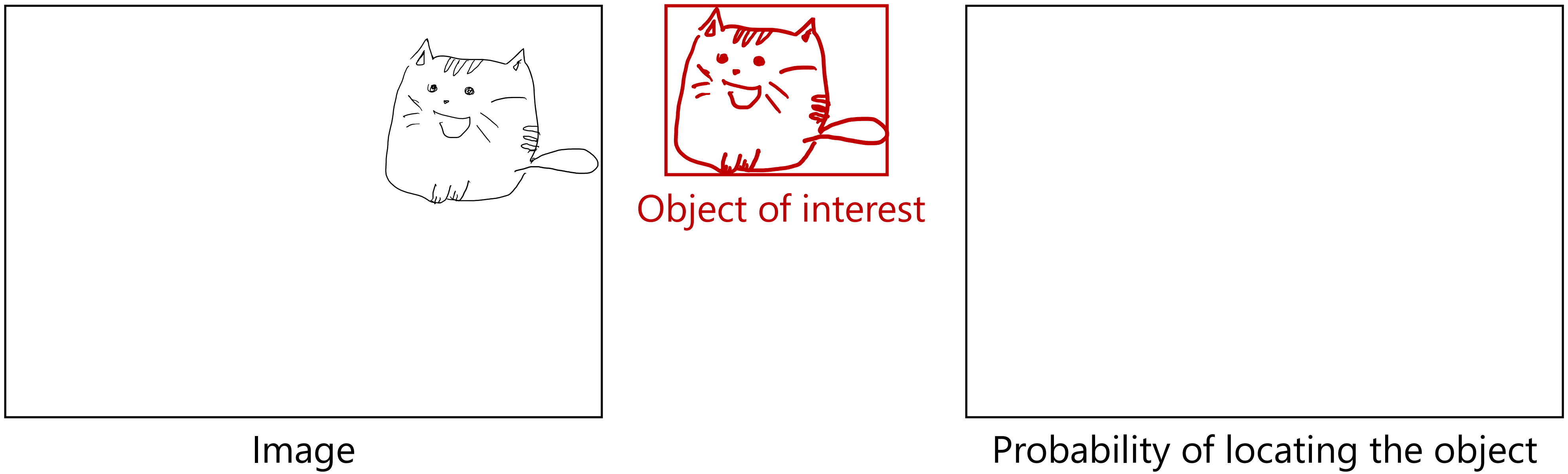
- Identifying an object on a picture would require examples with all possible locations of that object on the picture



# Translational symmetry

A cat moved from one part of an image to another is still a cat

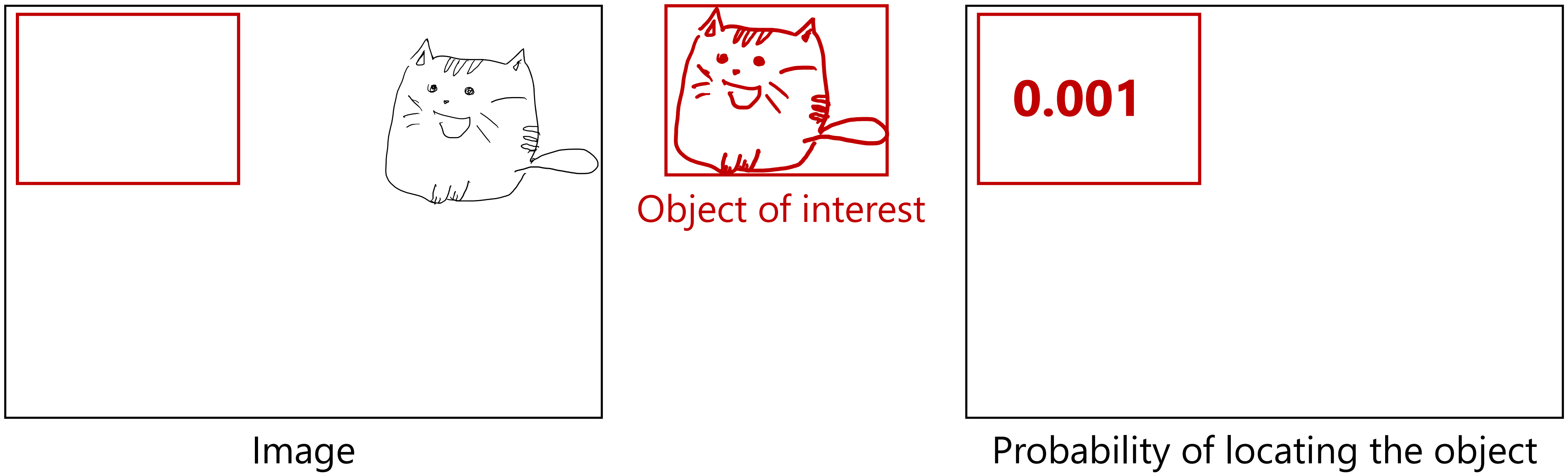
Why don't we use the same model to look at different patches of an image trying to identify the object of interest:



# Translational symmetry

A cat moved from one part of an image to another is still a cat

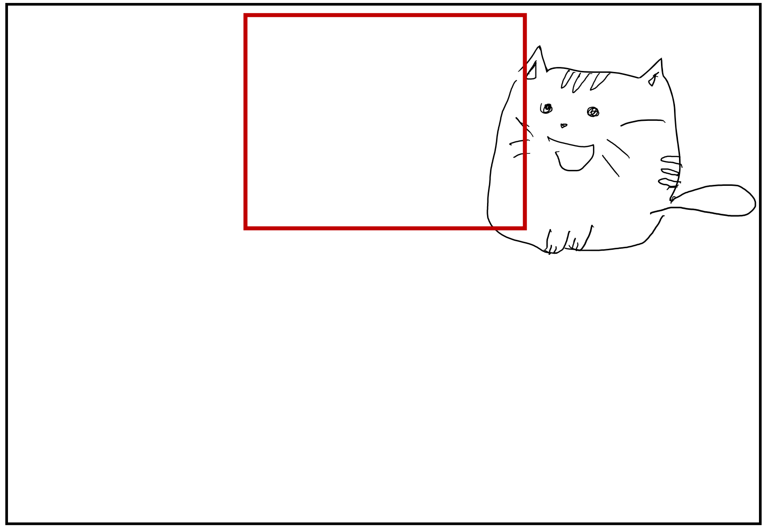
Why don't we use the same model to look at different patches of an image trying to identify the object of interest:



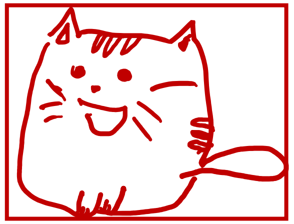
# Translational symmetry

A cat moved from one part of an image to another is still a cat

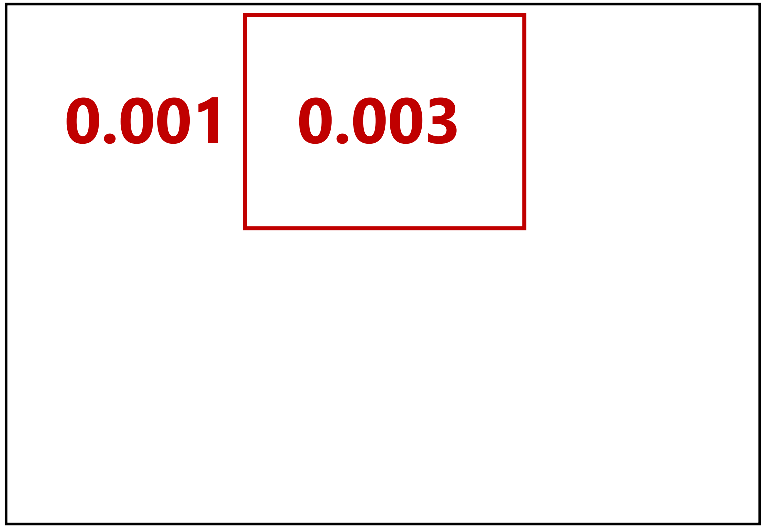
Why don’t we use the same model to look at different patches of an image trying to identify the object of interest:



Image



Object of interest

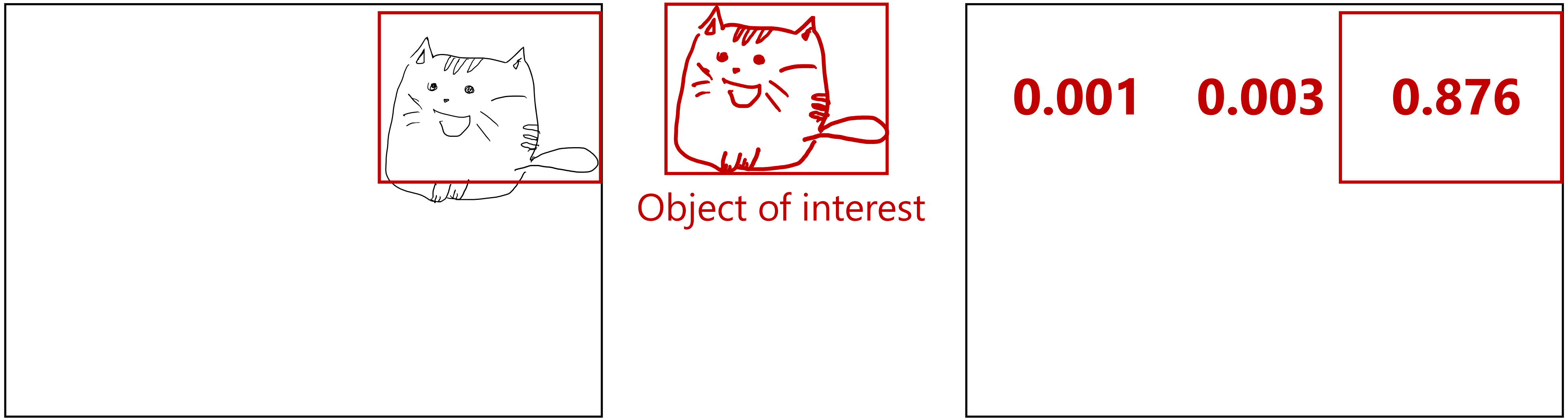



Probability of locating the object

# Translational symmetry

A cat moved from one part of an image to another is still a cat

Why don’t we use the same model to look at different patches of an image trying to identify the object of interest:



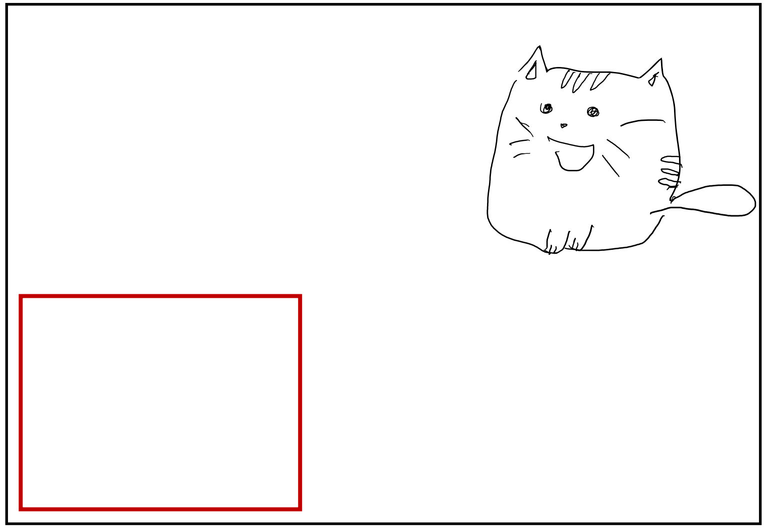

Image

Probability of locating the object

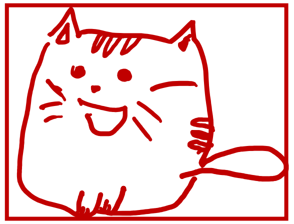
# Translational symmetry

A cat moved from one part of an image to another is still a cat

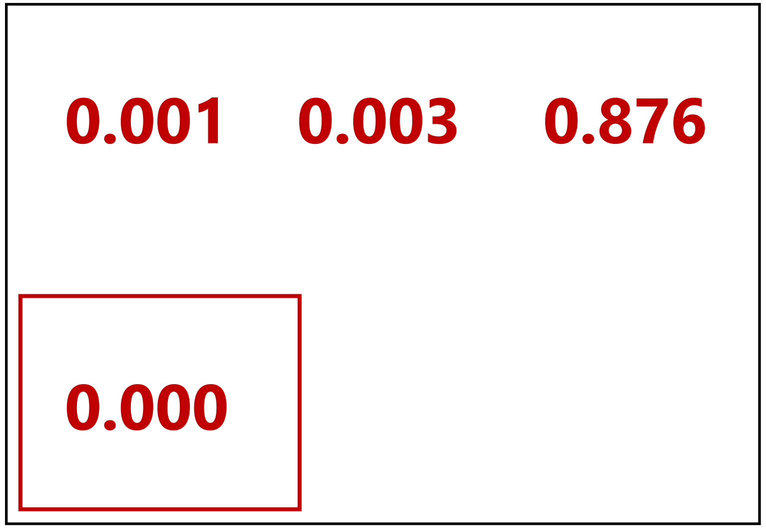
Why don't we use the same model to look at different patches of an image trying to identify the object of interest:



Image



Object of interest

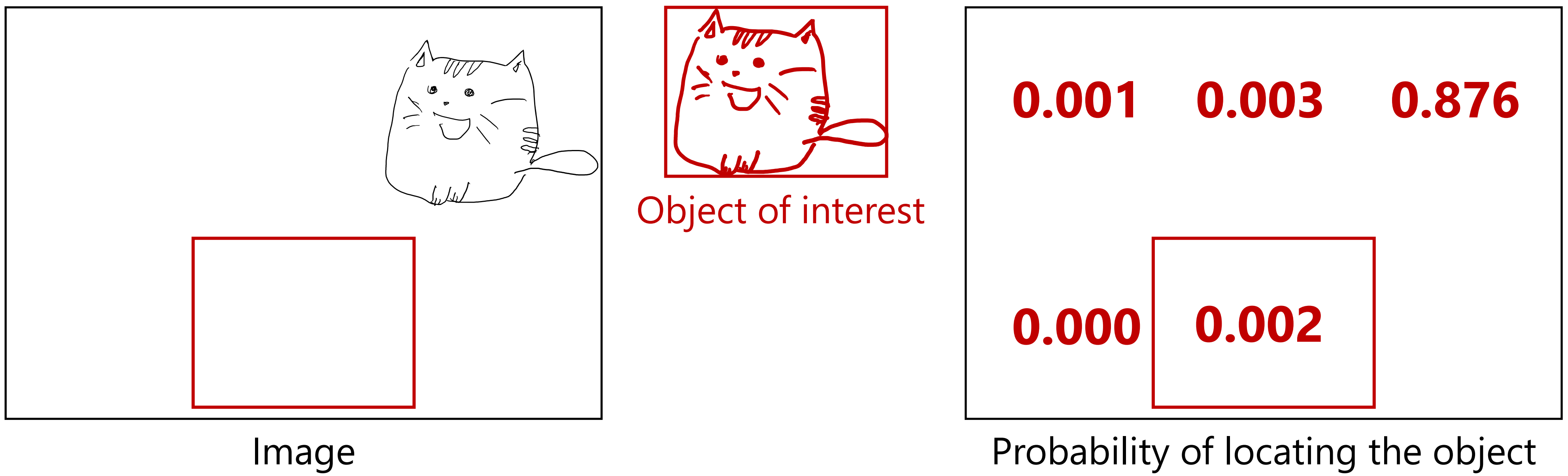



Probability of locating the object

# Translational symmetry

A cat moved from one part of an image to another is still a cat

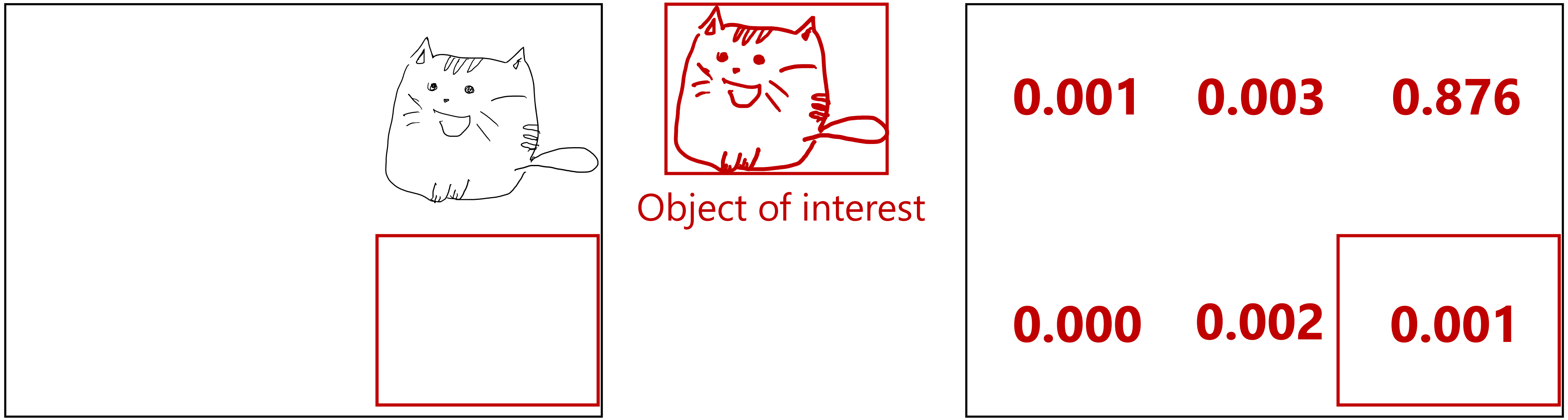
Why don't we use the same model to look at different patches of an image trying to identify the object of interest:



# Translational symmetry

A cat moved from one part of an image to another is still a cat

Why don’t we use the same model to look at different patches of an image trying to identify the object of interest:




Image

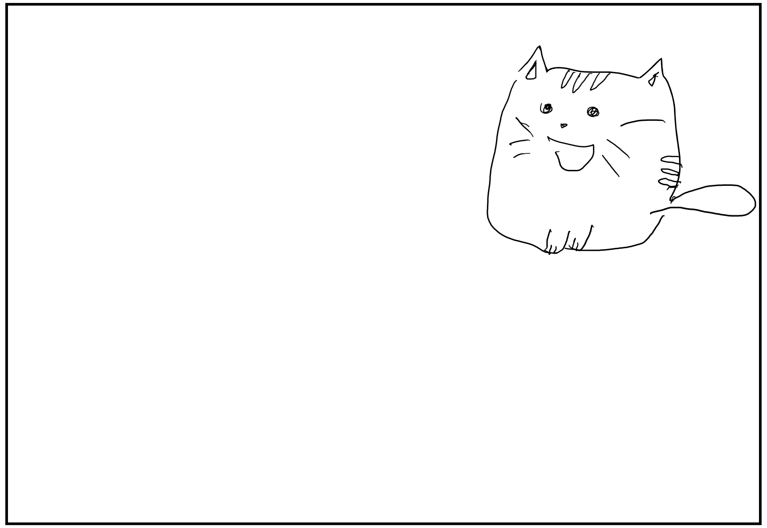
Probability of locating the object

# Translational symmetry

A cat moved from one part of an image to another is still a cat

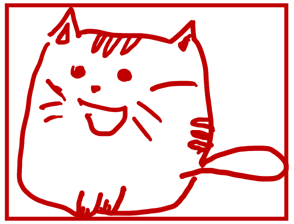
Why don't we use the same model to look at different patches of an image trying to identify the object of interest:

**This may be implemented with a 2D convolution!**



Image

**Input**



Object of interest

**Convolutional kernel**

| | | |
|---|---|---|
| **0.001** | **0.003** | **0.876** |
| **0.000** | **0.002** | **0.001** |

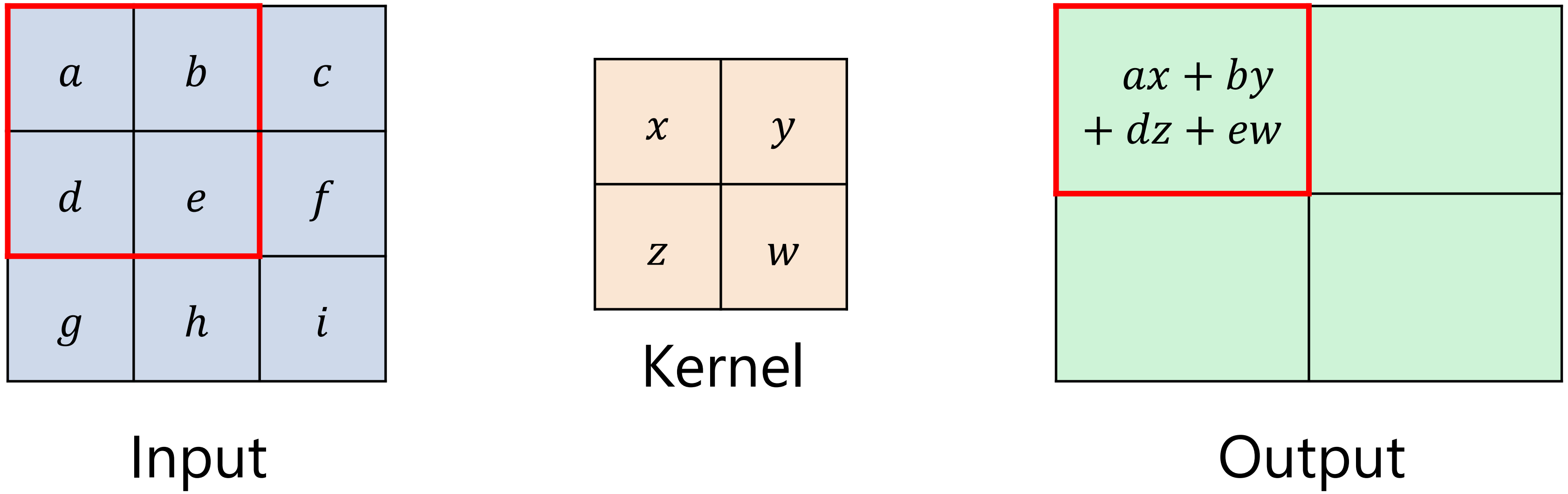
Probability of locating the object

**Output**

# 2D convolution

# 2D convolution

$$\mathrm{Output}(i,j) = \sum_{i',j'} \mathrm{Input}(i',j') \cdot \mathrm{Kernel}(i'-i, j'-j)$$



# 2D convolution

$$\text{Output}(i,j) = \sum_{i',j'} \text{Input}(i',j') \cdot \text{Kernel}(i'-i, j'-j)$$

| | | |
|---|---|---|
| $a$ | $b$ | $c$ |
| $d$ | $e$ | $f$ |
| $g$ | $h$ | $i$ |

Input

| | |
|---|---|
| $x$ | $y$ |
| $z$ | $w$ |

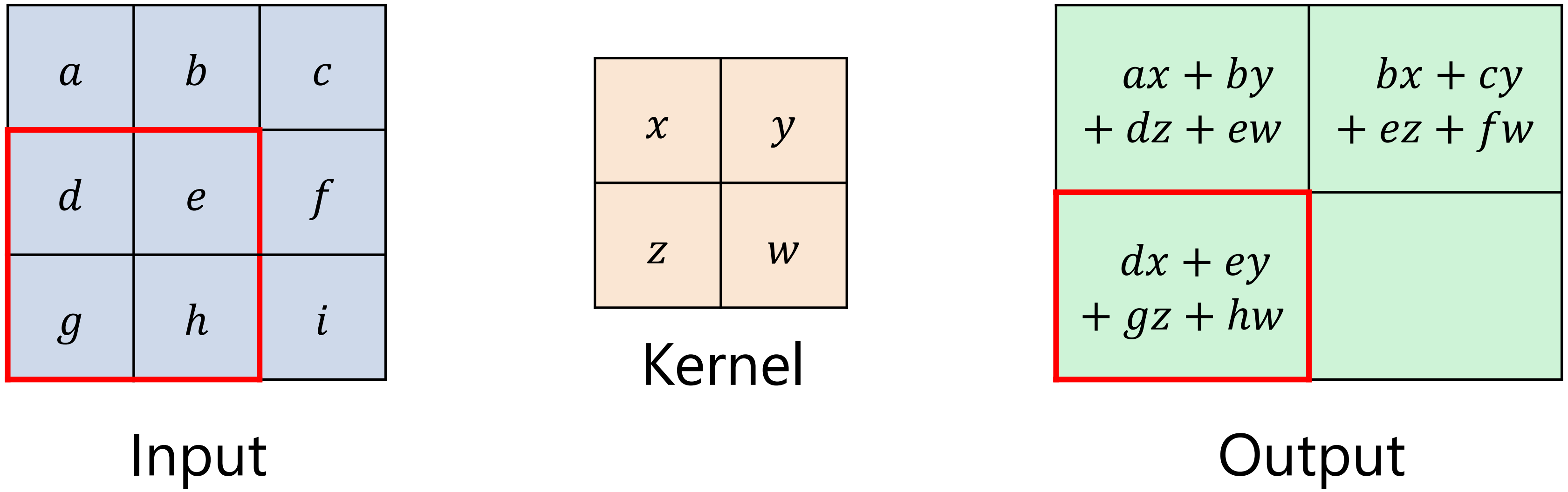
Kernel

| | |
|---|---|
| $ax + by + dz + ew$ | $bx + cy + ez + fw$ |
| | |

Output

# 2D convolution

$$\text{Output}(i,j) = \sum_{i',j'} \text{Input}(i',j') \cdot \text{Kernel}(i'-i, j'-j)$$



# 2D convolution

$$\text{Output}(i,j) = \sum_{i',j'} \text{Input}(i',j') \cdot \text{Kernel}(i'-i, j'-j)$$

| | | |
|---|---|---|
| $a$ | $b$ | $c$ |
| $d$ | $e$ | $f$ |
| $g$ | $h$ | $i$ |

Input

| | |
|---|---|
| $x$ | $y$ |
| $z$ | $w$ |

Kernel

| | |
|---|---|
| $ax + by + dz + ew$ | $bx + cy + ez + fw$ |
| $dx + ey + gz + hw$ | $ex + fy + hz + iw$ |

Output

# 2D convolution

$$\mathrm{Output}(i,j) = \sum_{i',j'} \mathrm{Input}(i',j') \cdot \mathrm{Kernel}(i'-i, j'-j)$$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \qquad \begin{bmatrix} x & y \\ z & w \end{bmatrix} \qquad \begin{bmatrix} ax+by+dz+ew & bx+cy+ez+fw \\ dx+ey+gz+hw & ex+fy+hz+iw \end{bmatrix}$$

Input

Kernel

Output

Different kernels may extract different features

# Examples

Input

Ouput

Blur:

$$\text{kernel} = \begin{pmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{pmatrix} \cdot \frac{1}{273}$$

# Examples

Input

Sharpen:

$$\text{kernel} = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

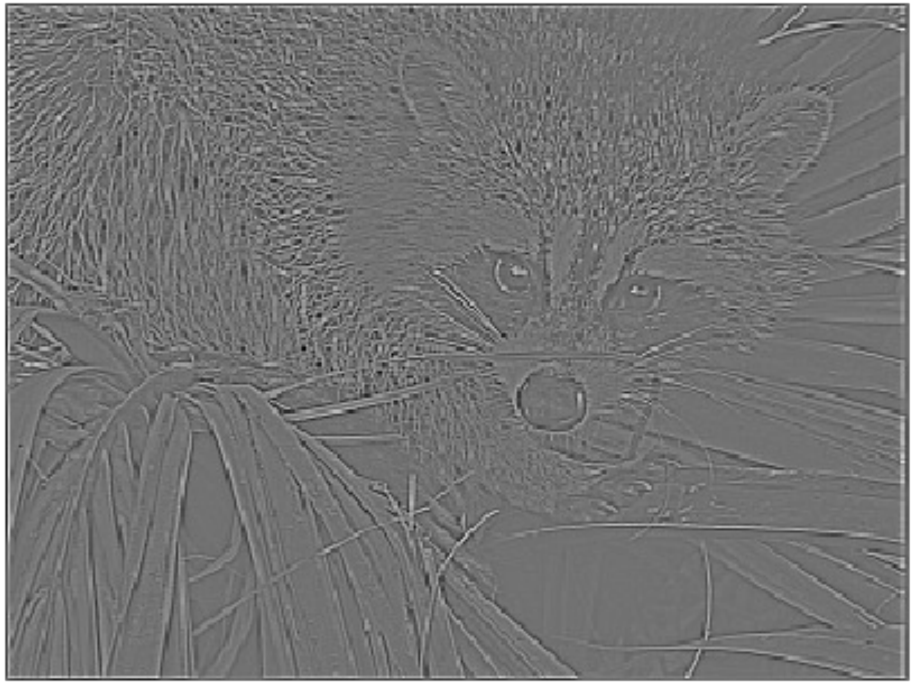
Ouput

# Examples

Input

Edge detection:

$$\text{kernel} = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$



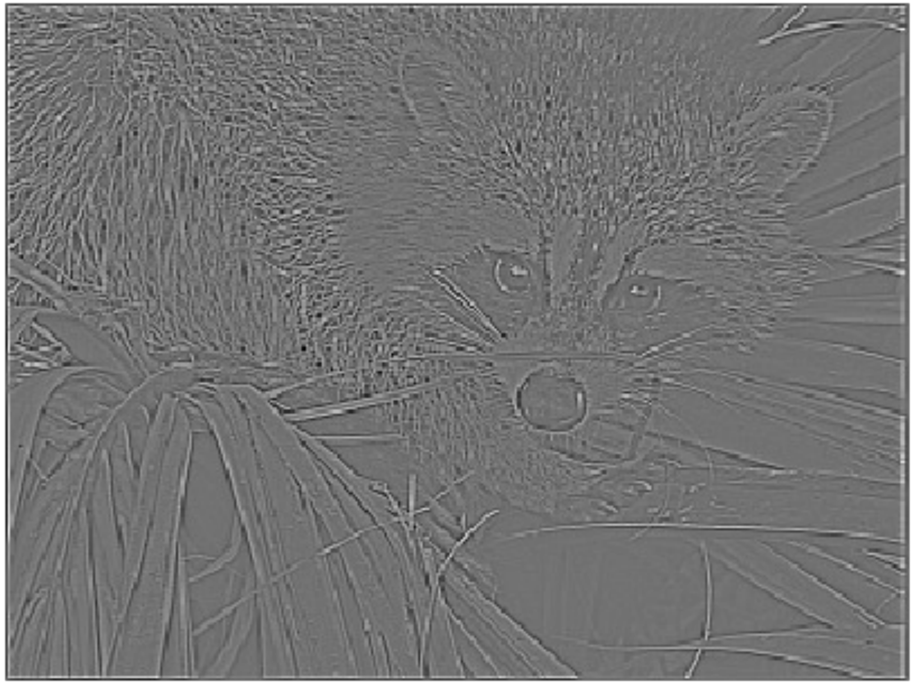
Ouput

# Examples

Input

Edge detection:

$$\text{kernel} = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$



Ouput

In the context of deep learning, the kernel parameters are **trainable**

I.e. the network **learns the kernel** to **extract useful features**

# 2D convolution as a matrix multiplication

Unwrap the 2D images into 1D vectors

Re-write the convolution as a regular matrix-vector multiplication

I.e. fully-connected layers comprise convolutions

- Yet they are much more complex

$$\underset{\text{Input}}{\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}} \quad \underset{\text{Kernel}}{\begin{bmatrix} x & y \\ z & w \end{bmatrix}} \quad \underset{\text{Output}}{\begin{bmatrix} ax+by+dz+ew & bx+cy+ez+fw \\ dx+ey+gz+hw & ex+fy+hz+iw \end{bmatrix}}$$

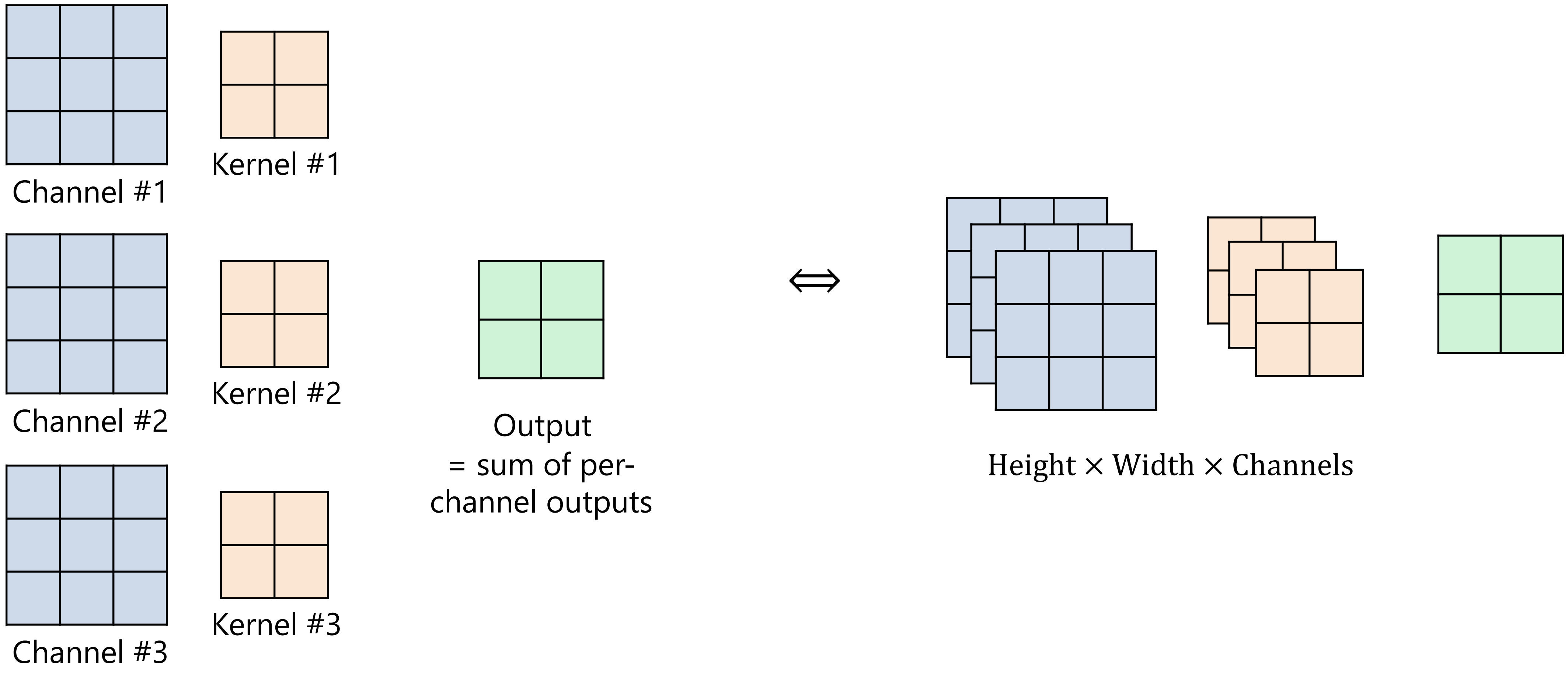
$$\begin{bmatrix} ax+by+dz+ew \\ bx+cy+ez+fw \\ dx+ey+gz+hw \\ ex+fy+hz+iw \end{bmatrix} = \begin{bmatrix} x & y & 0 & z & w & 0 & 0 & 0 & 0 \\ 0 & x & y & 0 & z & w & 0 & 0 & 0 \\ 0 & 0 & 0 & x & y & 0 & z & w & 0 \\ 0 & 0 & 0 & 0 & x & y & 0 & z & w \end{bmatrix} \times \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{bmatrix}$$

# 2D convolutional layers

# Input channels

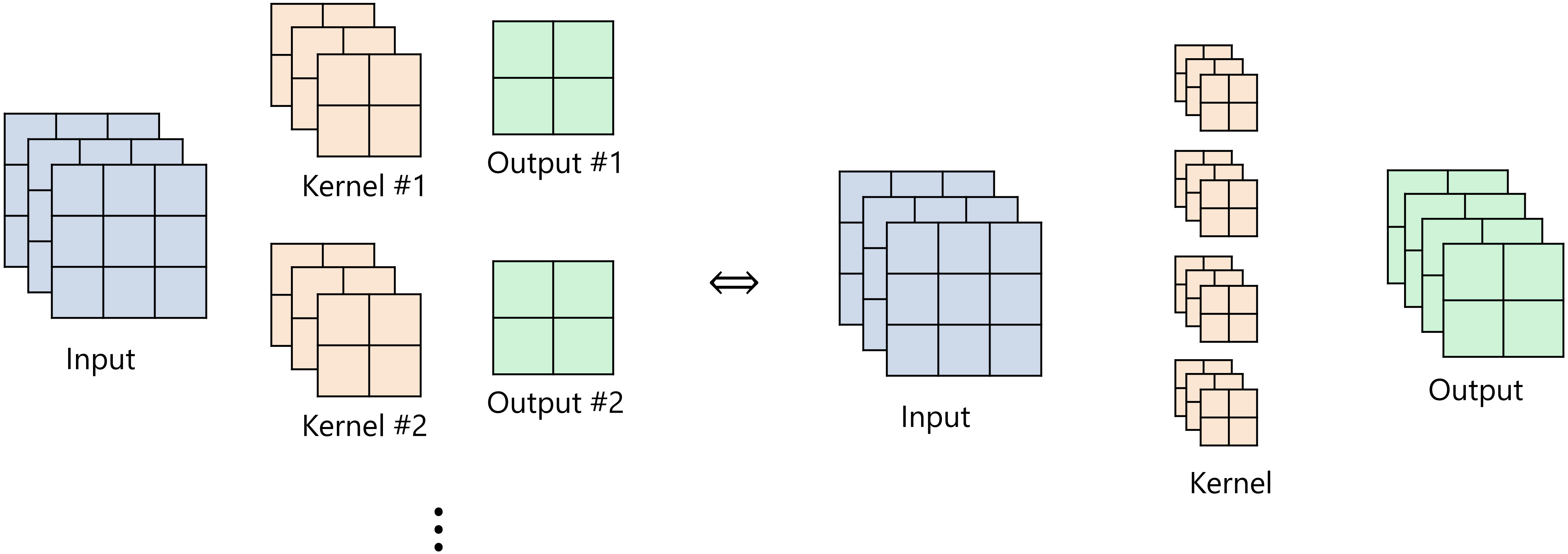
In practice images have multiple channels

- E.g. 3 color channels of a color image



# Output channels

In practice we want to extract multiple features




Kernel becomes 4D: $H_K \times W_K \times C_{in} \times C_{out}$

Output becomes 3D: $H \times W \times C_{out}$

# Putting it all together

bias term

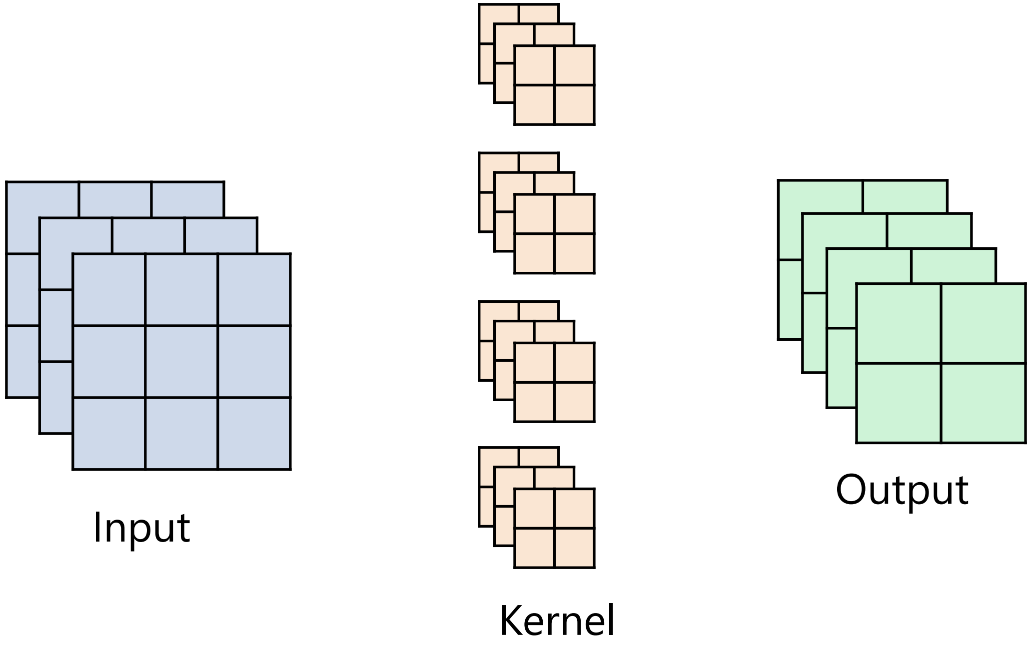
$$\text{Output}(i, j, c_{out}) = \sum_{i', j', c_{in}} \text{Input}(i', j', c_{in}) \cdot \text{Kernel}(i' - i, j' - j, c_{in}, c_{out}) + b_{out}$$

Note: with this approach the output width/height is smaller than the input width/height

- By how much (for a given kernel width and hight)?

Sometimes the border of the input image is **padded** with some values (e.g. s.t. the output has the same size)

- Controlled by the “padding” parameter



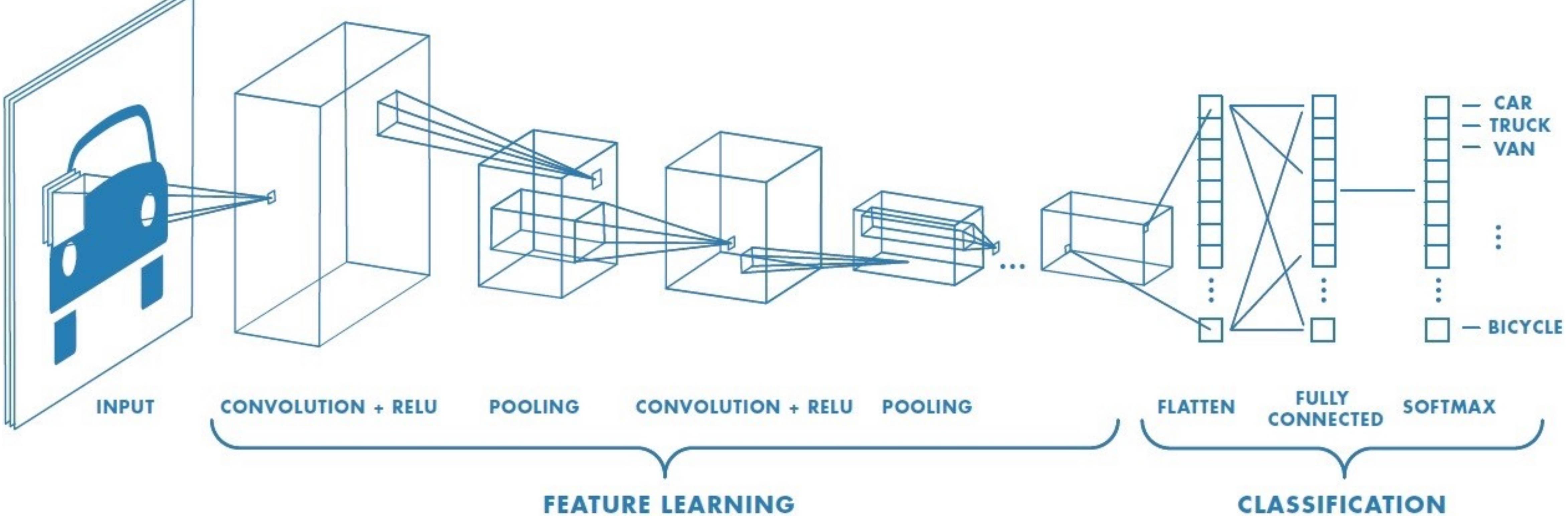

Some other parameters:

- “stride” – by how many pixels the kernel window steps (equals 1 in the examples here)
- “dilation” – kernel “spread” (e.g. see this animation)

# Typical network architecture

# Deep convolutional network

Image from: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f14

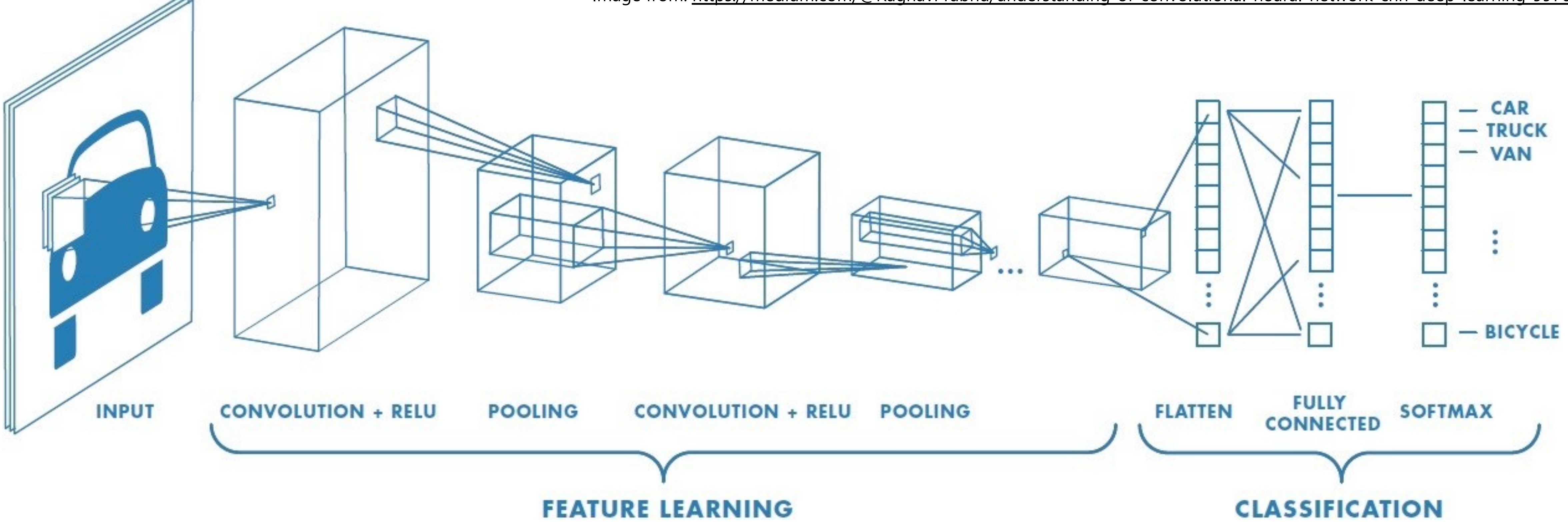



Convolution is typically followed by a pooling operation (aggregating values of nearby pixels), e.g. maxpool:

$$\begin{array}{|c|c|c|c|}\hline 1 & -2 & 5 & 7 \\ \hline 2 & 1 & 4 & -3 \\ \hline 4 & -9 & 2 & 0 \\ \hline -4 & 15 & 2 & 1 \\ \hline \end{array} \xrightarrow{\text{maxpool}(2\times 2)} \begin{array}{|c|c|}\hline 2 & 7 \\ \hline 15 & 2 \\ \hline \end{array}$$

# Deep convolutional network

Image from: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f14




Convolution is typically followed by a pooling operation (aggregating values nearby pixels), e.g. maxpool:

**Motivation:**

Deeper layers extract “higher level” (i.e. more complex) features, and we’re less interested in their spacial location.
Hence, fewer pixels and more channels.

| | | | |
|---|---|---|---|
| $-4$ | 15 | 2 | 1 |

| | |
|---|---|
| | 7 |
| 13 | 2 |

# Thank you!


Majid Sohrabi

msohrabi@hse.ru