Aprendizaje Bioestadístico

Convolutional Neural Networks

Santiago Alférez

Escola d'Enginyeria de Barcelona Est (EEBE)
Departament de Matemàtiques
Universitat Politècnica de Catalunya (UPC)

- Introduction
- Convolution
- Pooling
- Multilayer structure

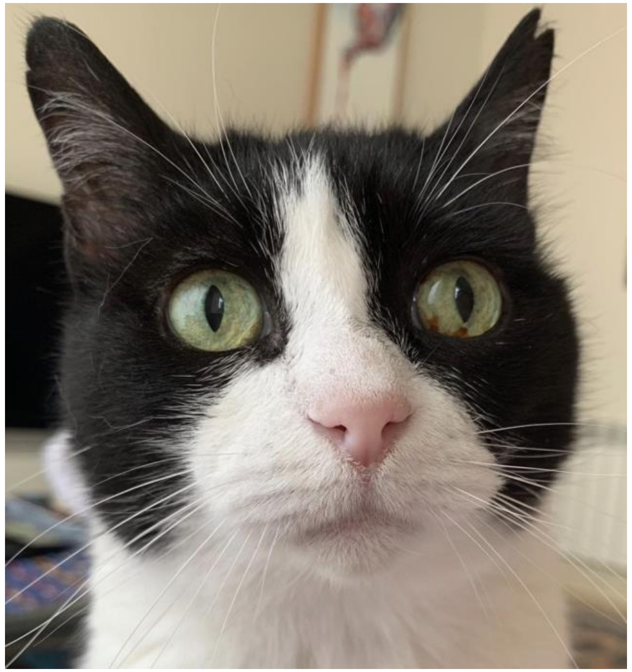
Inspiration and motivation

Inspired by how the **visual cortex** of the **human brain** functions to recognize objects.

The visual cortex is conformed by different **layers**:

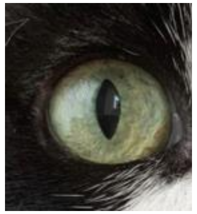
- The first layer mainly detects **simple shapes** like edges and straight lines.
- Higher-order layers focus more on extracting **complex shapes** and patterns.

How humans classify images



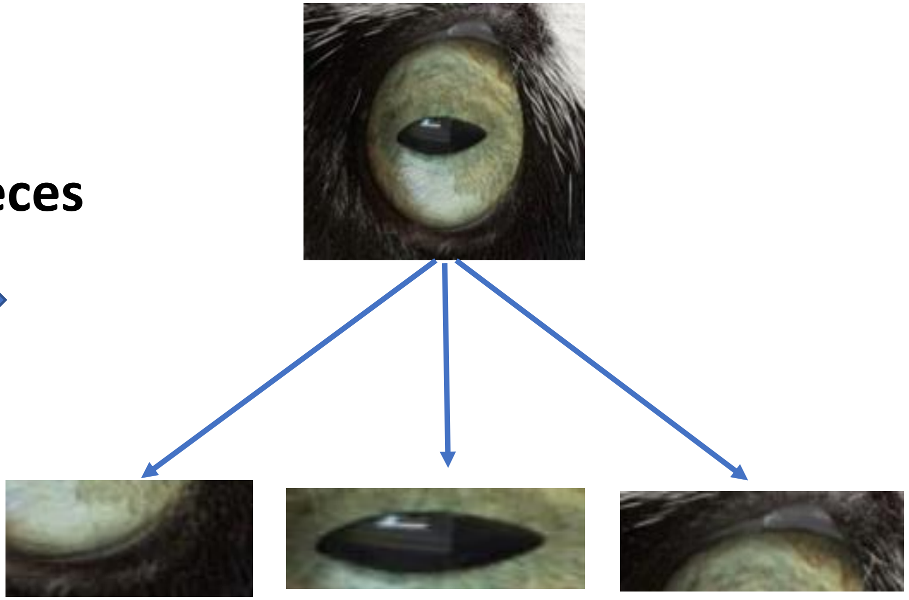
"Carrie"

We identify specific parts: ears, eyes, nose, whisker



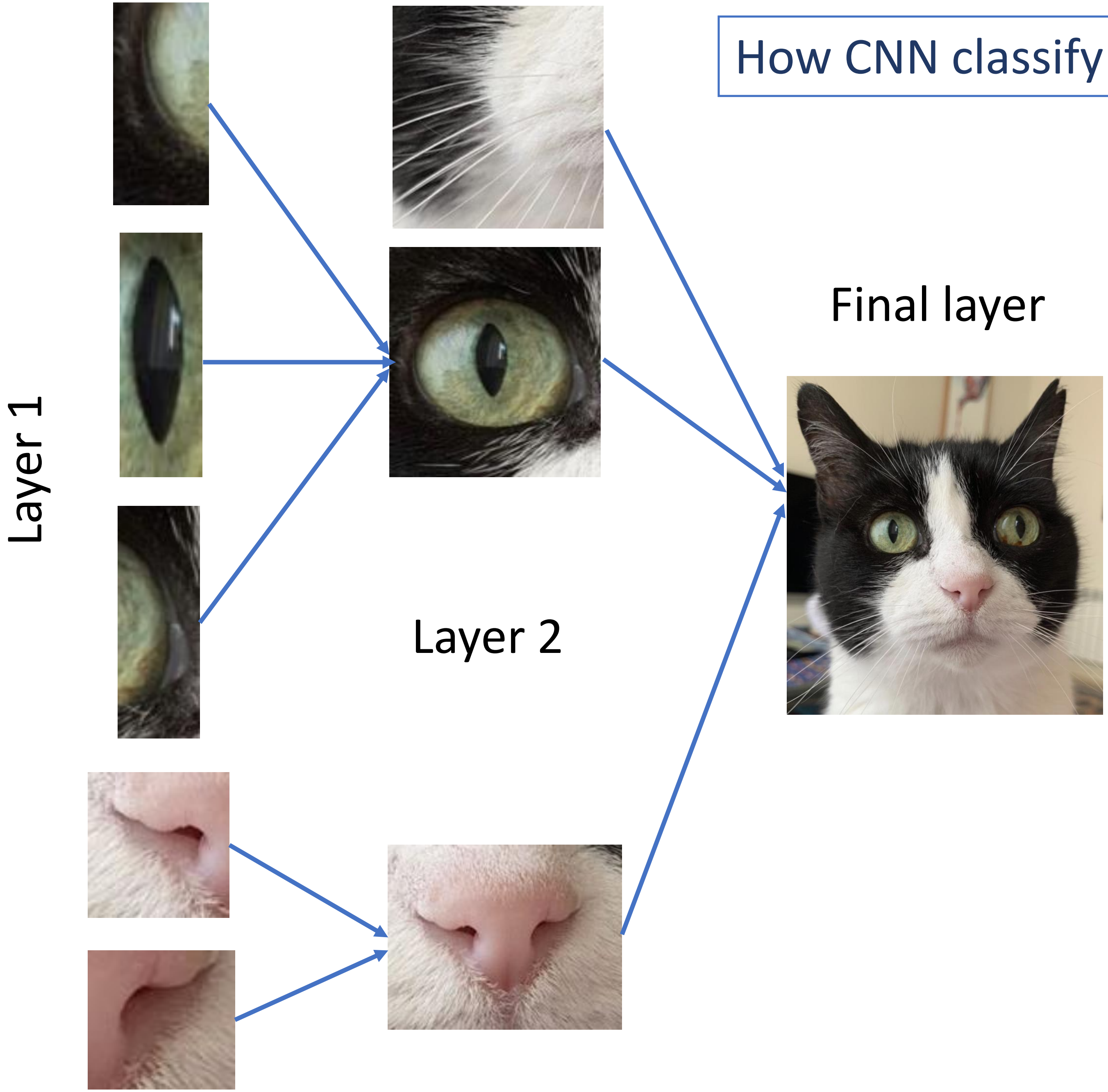
Combine parts to identify the overall object (cat)

We break parts
into smaller pieces



- An **eye** can be seen as a circle with an oval inside.
- A **nose** can be seen as two holes.

How CNN classify images

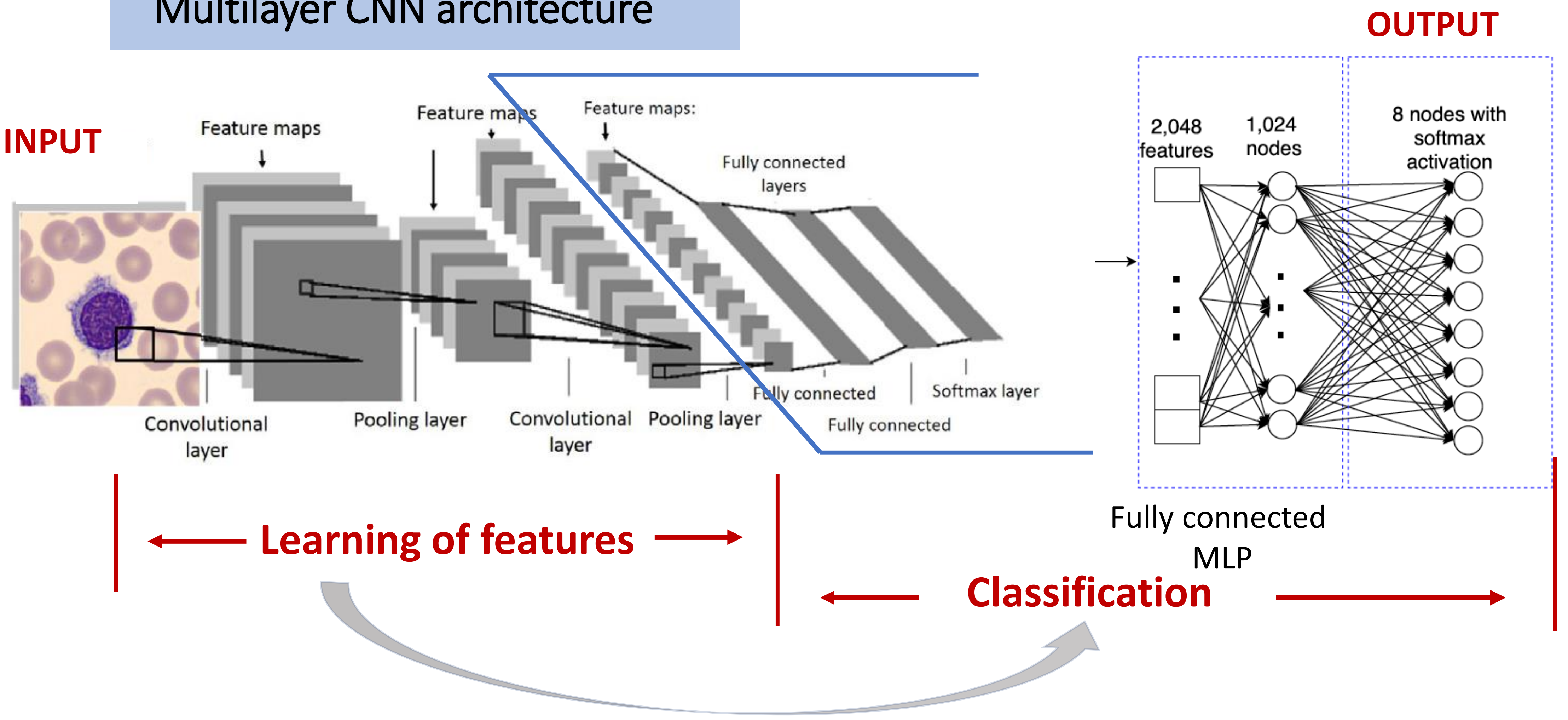


Hierarchy of layers

- **Recognize basic lines and curves**
- **Recognize simple shapes and spots**: edges, circles, ovals
- **Identify Increasingly complex objects**: eyes, nose, whiskers
- ……
- ……

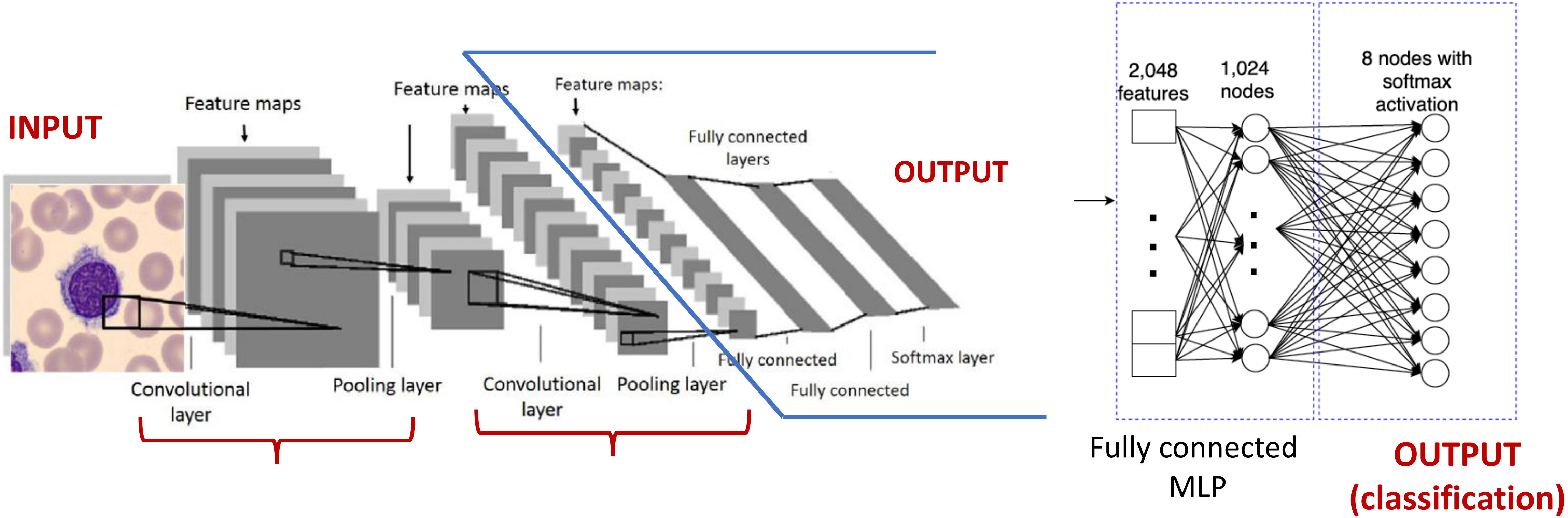
- Finally, the **CNN classifies** the object (cat) as whole through a subsequent combination of more complex objects

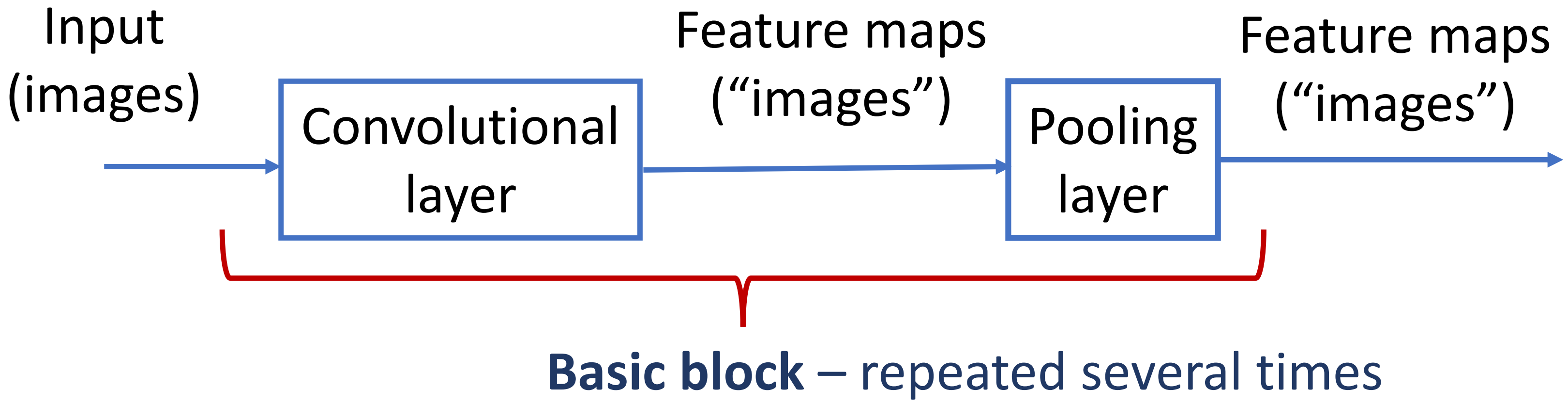
Multilayer CNN architecture



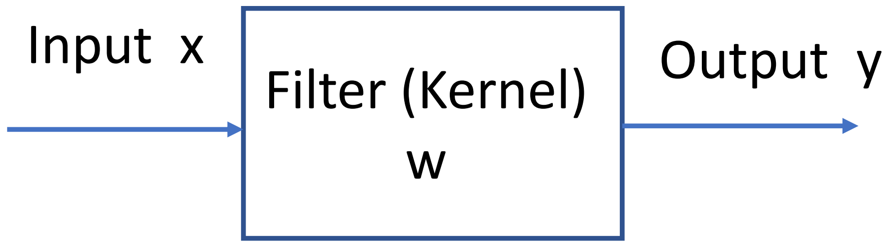
The CNN just learns from the **training set** and discovers which **characteristics** of the image are worth for the **classification**.

Multilayer CNN architecture





Discrete convolution – One dimension



$$x = (x_0, x_1, \cdots x_n)$$

$$w = (w_0, w_1, \cdots w_m)$$

$$m \leq n$$

$$y = (y_0, y_1, \ldots, m + n)$$

Mathematical definition

$$y = x * w$$

$$y(i) = \sum_{-\infty}^{\infty} x(i-k)w(k)$$

In practice, vectors have **finite dimension** and negative index values are not used. Therefore, we implement the formula

$$y(i) = \sum_{k=0}^{m} x(i-k)w(k)$$

assuming that the values of *x* for negative index *(i-k)* are equal to 0.

Example

$$x = (x_0, x_1, \dots, x_7)$$

$$w = (w_0, w_1, w_2, w_3)$$

We apply the formula

$$y(i) = \sum_{k=0}^{m=3} x(i-k)w(k)$$

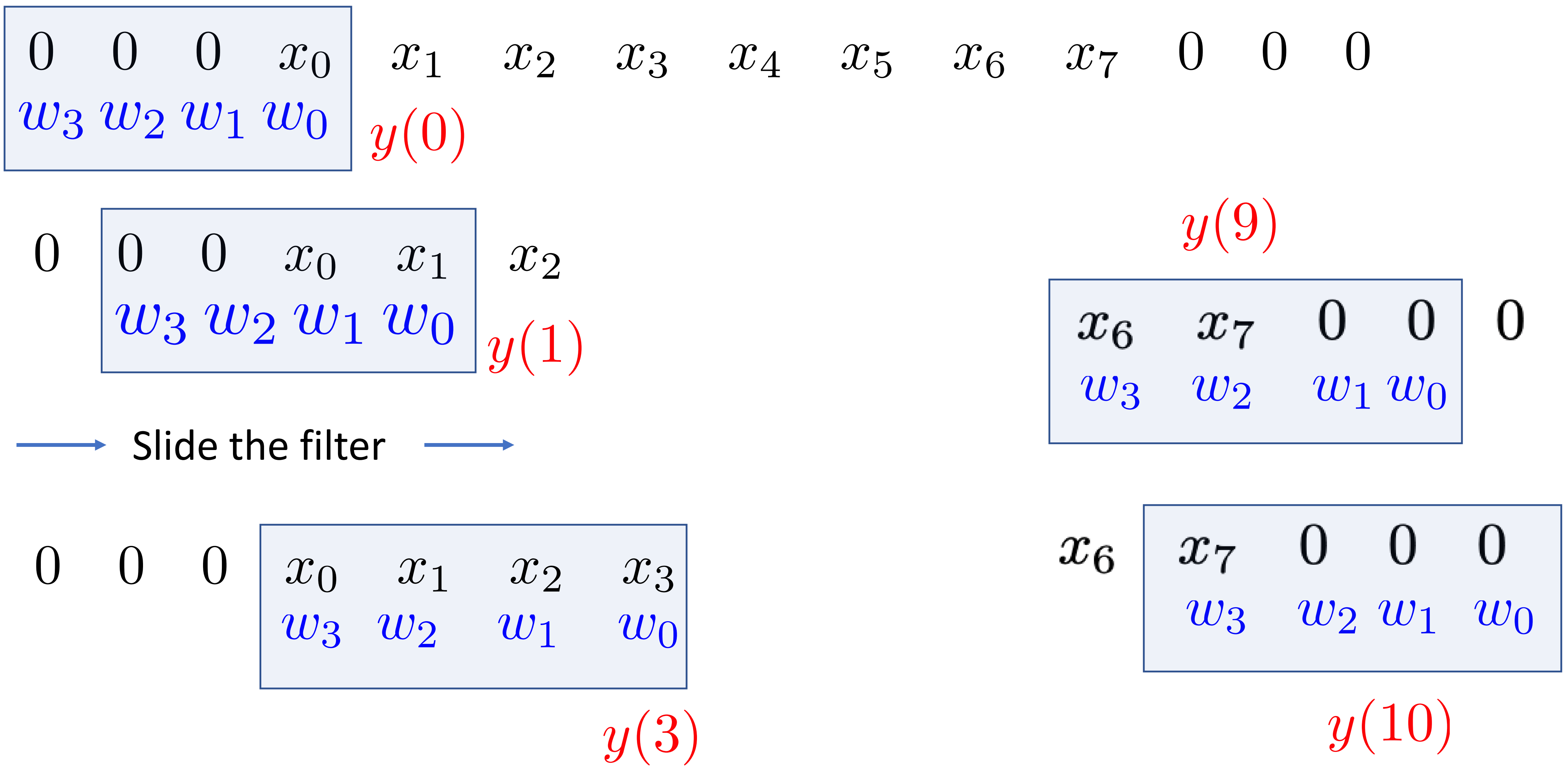
$$y(0) = x(0)w(0)$$
$$y(1) = x(1)w(0) + x(0)w(1)$$
$$y(2) = x(2)w(0) + x(1)w(1) + x(0)w(2)$$

$$y(3) = x(3)w(0) + x(2)w(1) + x(1)w(2) + x(0)w(3)$$
$$y(4) = x(4)w(0) + x(3)w(1) + x(2)w(2) + x(1)w(3)$$

$$\vdots \qquad \ldots\ldots\ldots\ldots\ldots\ldots$$

$$y(7) = x(7)w(0) + x(6)w(1) + x(5)w(2) + x(4)w(3)$$

$$y(8) = x(7)w(1) + x(6)w(2) + x(5)w(3)$$
$$y(9) = x(7)w(2) + x(6)w(3)$$
$$y(10) = x(7)w(3)$$


$0 \quad 0 \quad 0 \quad x_0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_7 \quad 0 \quad 0 \quad 0$
$w_3 \; w_2 \; w_1 \; w_0$
$y(0)$
$0 \quad 0 \quad 0 \quad x_0 \quad x_1 \quad x_2$
$w_3 \; w_2 \; w_1 \; w_0$
$y(1)$
Slide the filter
$0 \quad 0 \quad 0 \quad x_0 \quad x_1 \quad x_2 \quad x_3$
$w_3 \; w_2 \; w_1 \; w_0$
$y(3)$
$y(9)$
$x_6 \quad x_7 \quad 0 \quad 0 \quad 0$
$w_3 \; w_2 \; w_1 \; w_0$
$x_6 \quad x_7 \quad 0 \quad 0 \quad 0$
$w_3 \; w_2 \; w_1 \; w_0$
$y(10)$

Padding

Adding zeros to the input is called zero padding, or just ***padding.***

In the example,

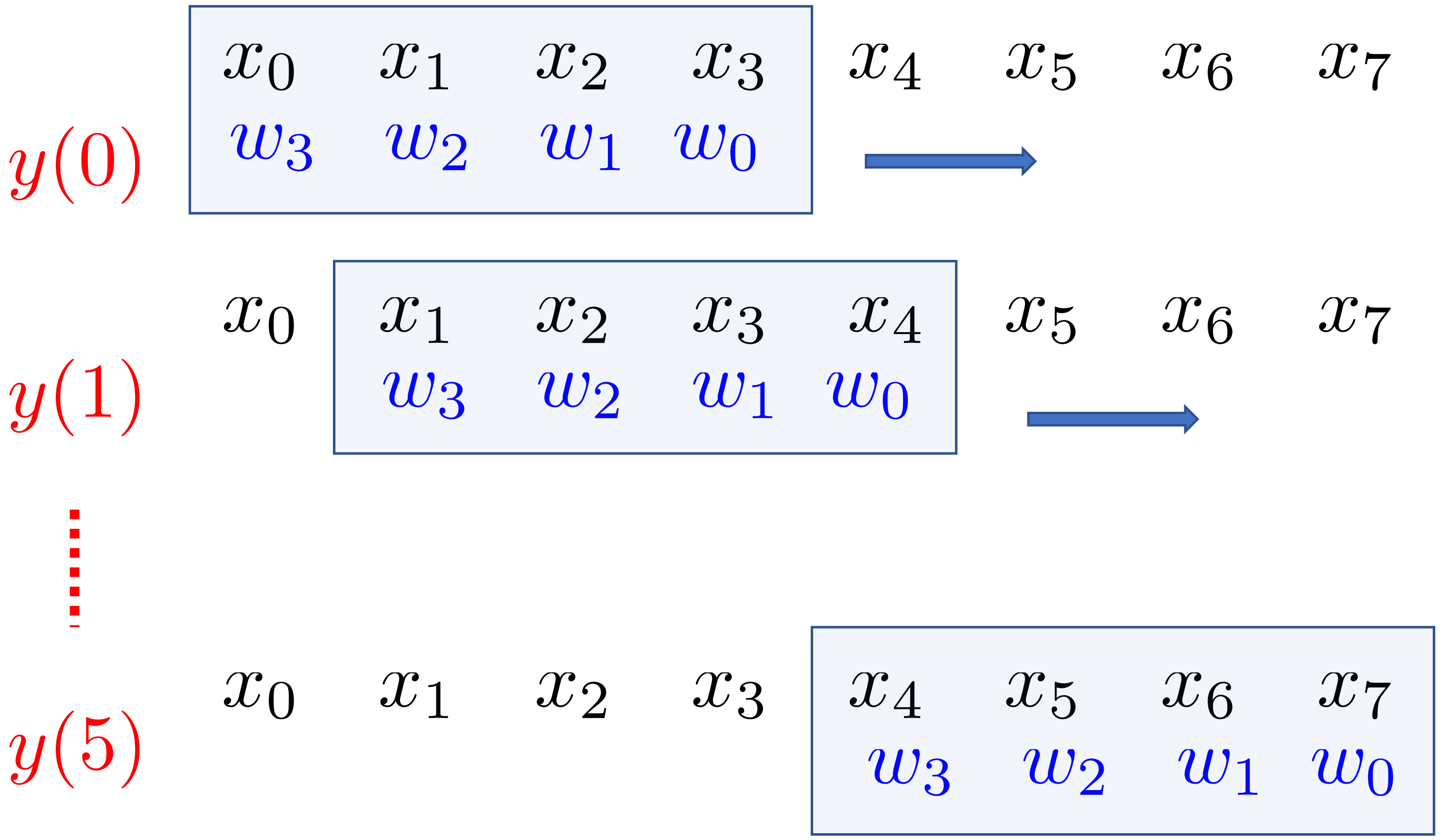
$$\underbrace{0 \quad 0 \quad 0}_{p=3} \quad x_0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_7 \quad \underbrace{0 \quad 0 \quad 0}_{p=3}$$

We have added 2p = 6 zeros

This is called **full padding**

- Note that, doing this padding, the **output has greater dimension** than the input.
- This is rarely used in CNN.

With no padding



- Output dimension = 6
- **Lower** than the dimension of the input

Stride

$$y(0) \quad \boxed{\begin{matrix} x_0 & x_1 & x_2 & x_3 \\ w_3 & w_2 & w_1 & w_0 \end{matrix}} \quad x_4 \quad x_5 \quad x_6 \quad x_7$$

$$y(1) \quad x_0 \quad \boxed{\begin{matrix} x_1 & x_2 & x_3 & x_4 \\ w_3 & w_2 & w_1 & w_0 \end{matrix}} \quad x_5 \quad x_6 \quad x_7$$

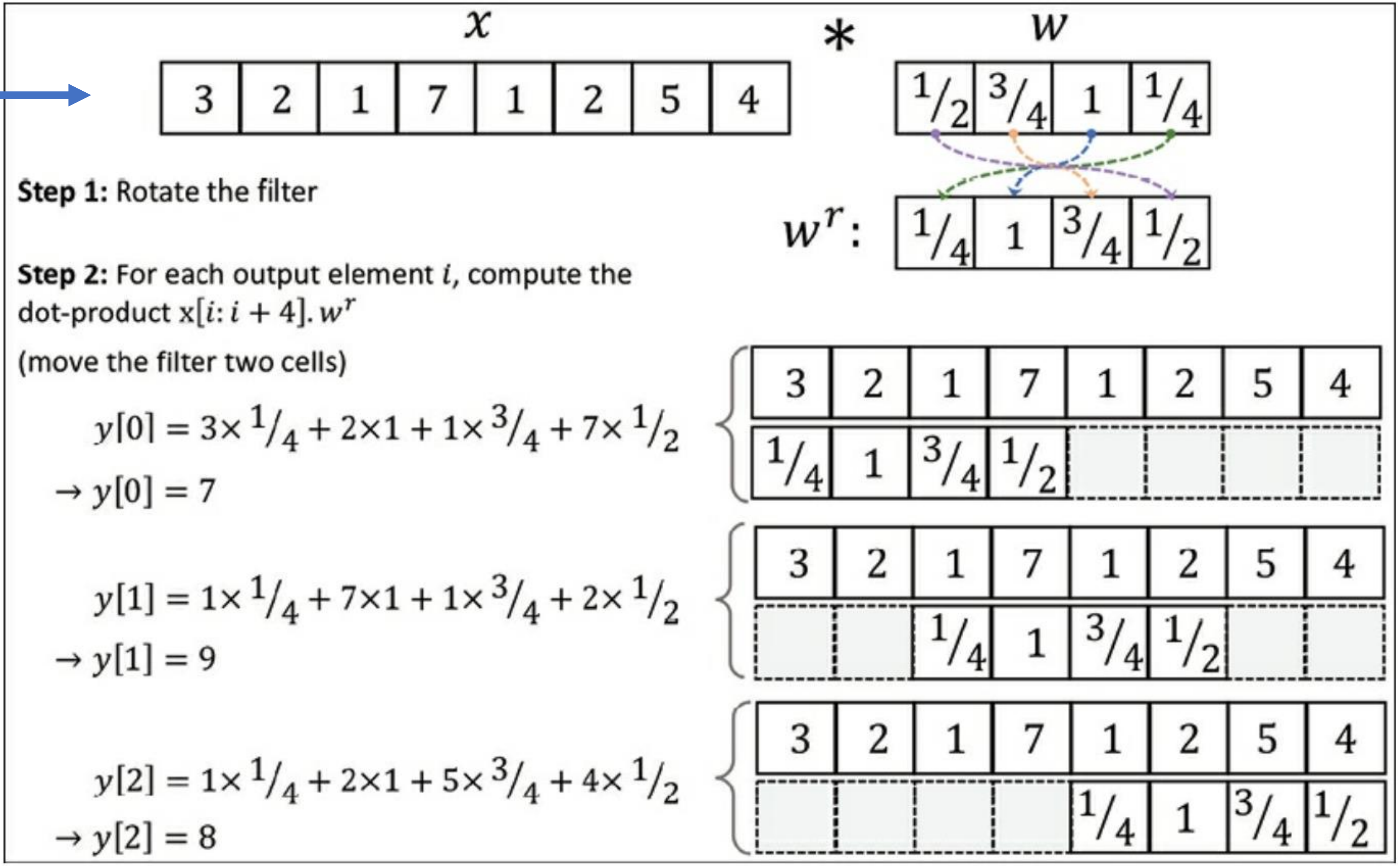
In this example the filter is **shifted** by one element each time.

In general, we call ***stride (s)*** the number of cells by which the filter is shifted.
It can be >= 1.

Example

No padding
p=0

Stride s=2



Determining the size of the convolution output

Hyperparameters: padding and stride

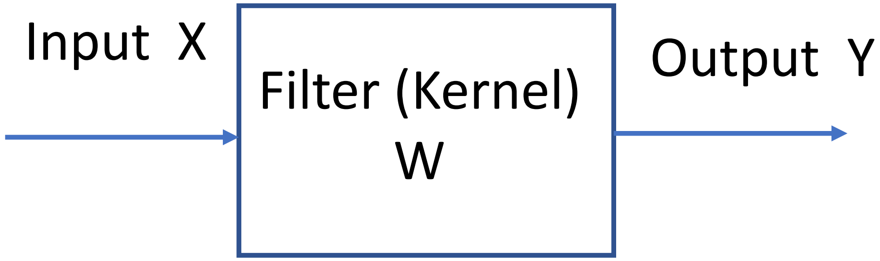
The output size of a convolution is determined by the number of times that we shift the filter along the input vector.

$$o = \frac{n + 2p - m}{s} + 1$$

n: input vector size
m: filter size
p: padding
s: stride
o: **output size**

In CNN architectures, the usual practice is to choose **padding** and **stride** to have the output with the same size as the input

Discrete convolution in 2 dimensions



X: Matrix of size $n_1 \times n_2$

W: Matrix of size $m_1 \times m_2$

Y: Matrix of size $o_1 \times o_2$

Mathematical definition

$$Y = X * W$$

$$Y(i,j) = \sum_{k_1} \sum_{k_2} X(i-k_1, j-k_2) W(k_1, k_2)$$

Discrete convolution in 2 dimensions

1D convolution

$$
\begin{array}{ccccccc}
\boxed{\begin{array}{cccc} 0 & 0 & 0 & x_0 \\ w_3 & w_2 & w_1 & w_0 \end{array}} & x_1 & x_2 & x_3 \\
 & y(0) & &
\end{array}
$$

Previous techniques seen for 1D are also applicable for 2D:

- Padding and stride
- Rotate and slide filter

In 2D the filter kernel matrix is rotated 180º

$$
\begin{pmatrix} W_{00} & W_{01} & W_{02} \\ W_{10} & W_{11} & W_{12} \\ W_{20} & W_{21} & W_{22} \end{pmatrix} \longrightarrow \begin{pmatrix} W_{22} & W_{21} & W_{20} \\ W_{12} & W_{11} & W_{10} \\ W_{02} & W_{01} & W_{00} \end{pmatrix}
$$

and slides over a 2D input matrix

Example

X: Original image 3x3

W: Filter 3x3

Padding p(1,1): one array of zeros are added on each side

Padded image has size 5x5

X

0	0	0	0	0
0	2	1	2	0
0	5	0	1	0
0	1	7	3	0
0	0	0	0	0

$*$

W

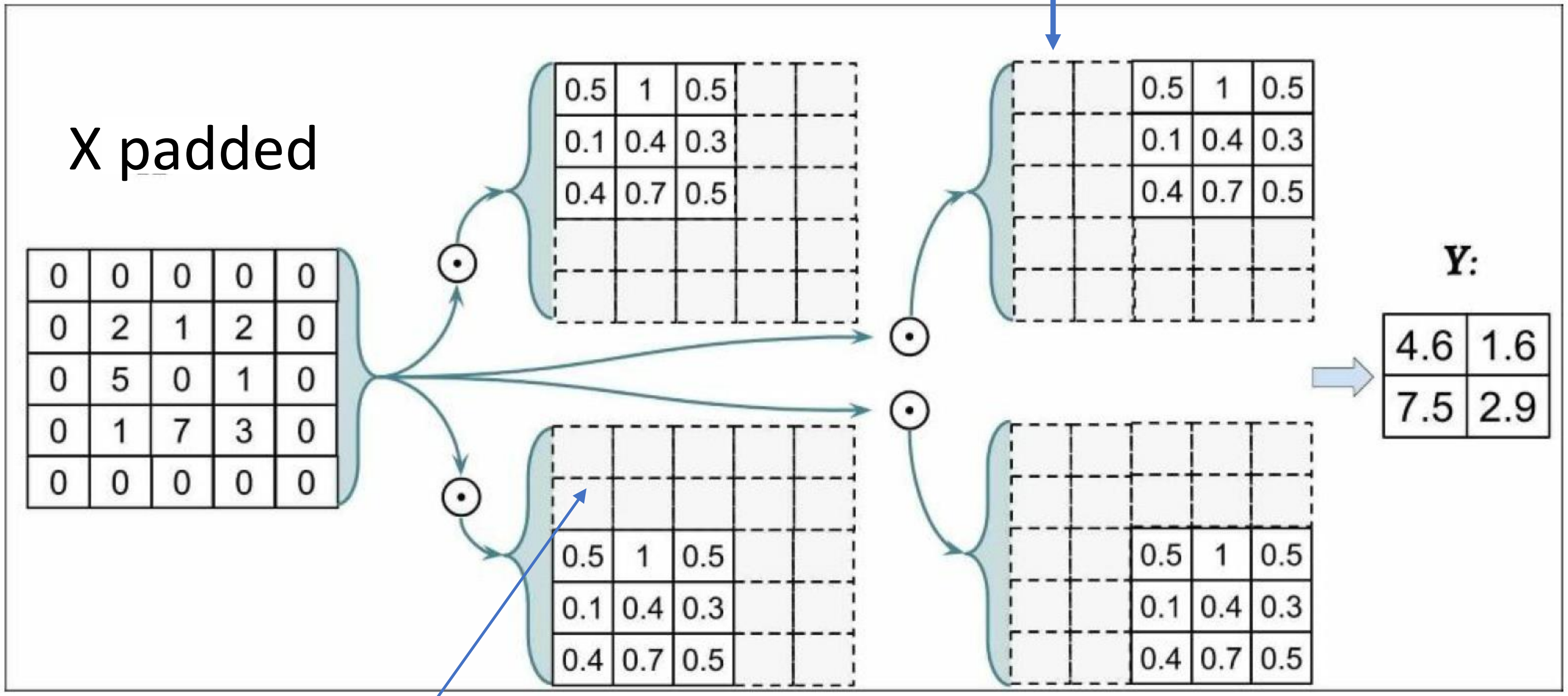
0.5	0.7	0.4
0.3	0.4	0.1
0.5	1	0.5

W rotated

$$\begin{bmatrix} 0.5 & 1 & 0.5 \\ 0.1 & 0.4 & 0.3 \\ 0.4 & 0.7 & 0.5 \end{bmatrix}$$

Example

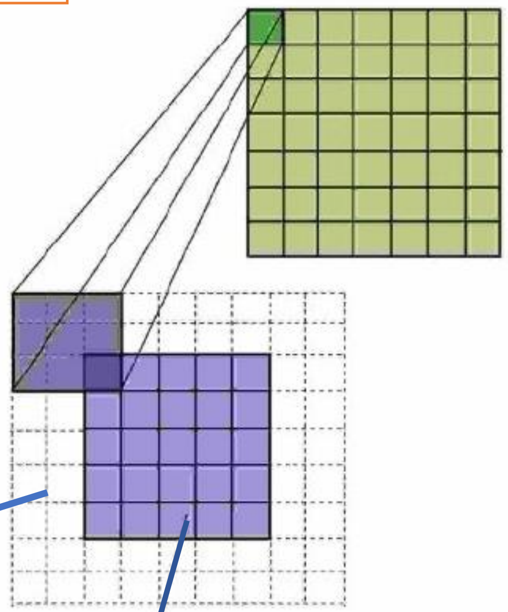
The rotated filter slides with stride (2,2): 2 pixels at a time in both directions.



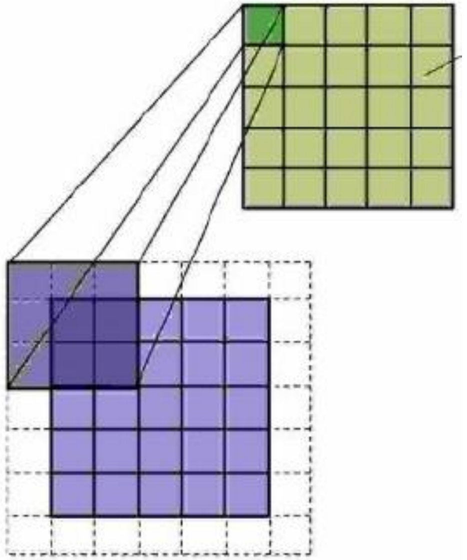
stride (2,2)

Three types of padding

“Full” padding

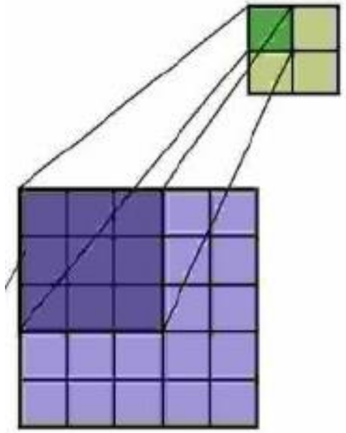


“Same” padding



Output keeps the size of the input

“Valid” padding



No padding

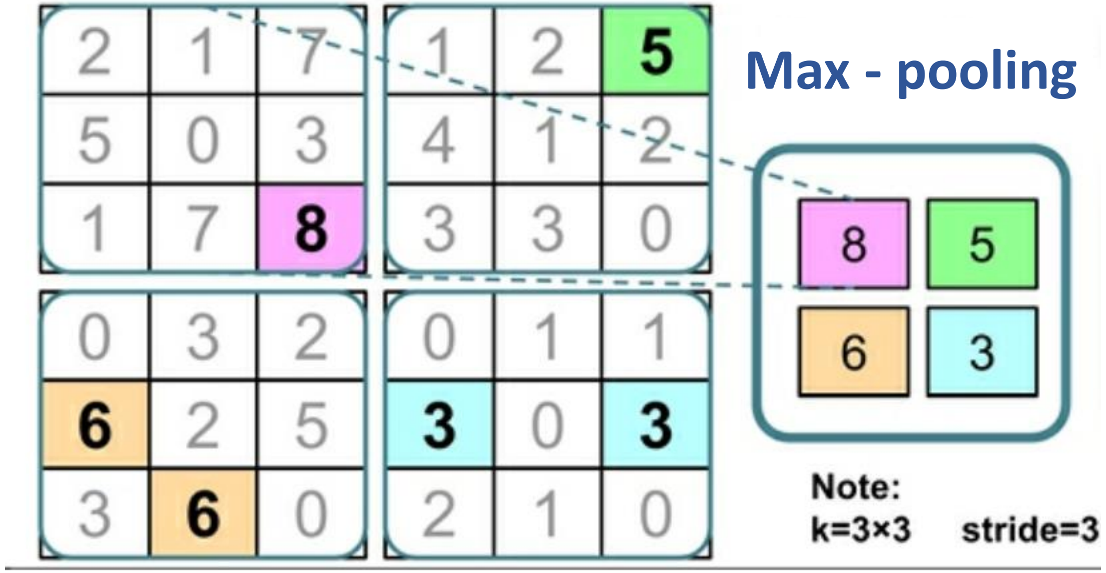
The size of the output image depends on filter size, padding and stride

Subsampling (pooling)

Once the image output is obtained, **subsampling** consists in replacing the value at specific locations by a summary statistic of neighbor output values.

$P_{d1\times d2}$ is a matrix that states the **pooling size**: number of **adjacent pixels** in each dimension where the pooling operation is performed.

$P_{3\times3}$



Max - pooling

Mean - pooling

(average value)

Subsampling (pooling)

<u>Advantages</u>

- **Reduce the size** of output features, which reduces computational cost in CNN networks and helps to reduce the degree of overfitting.

- Max-pooling introduces some local **invariance**.

This means that small changes in the input do not change most of the pooled outputs.

Therefore, it helps generate features that are more **robust** to noise in the input data.

See the example

Example

Max-pooling P (2x2)

$$X1=\begin{bmatrix} \boxed{\begin{matrix}10 & 255\\ 70 & 255\end{matrix}} & \begin{matrix}125 & 0\\ 105 & 25\end{matrix} & \begin{matrix}170 & 100\\ 25 & 70\end{matrix} \\ \begin{matrix}255 & 0\\ 0 & 255\end{matrix} & \begin{matrix}150 & 0\\ 10 & 10\end{matrix} & \begin{matrix}10 & 10\\ 150 & 20\end{matrix} \\ \begin{matrix}70 & 15\\ 35 & 25\end{matrix} & \begin{matrix}200 & 100\\ 100 & 20\end{matrix} & \boxed{\begin{matrix}95 & 0\\ 0 & 60\end{matrix}} \end{bmatrix}$$

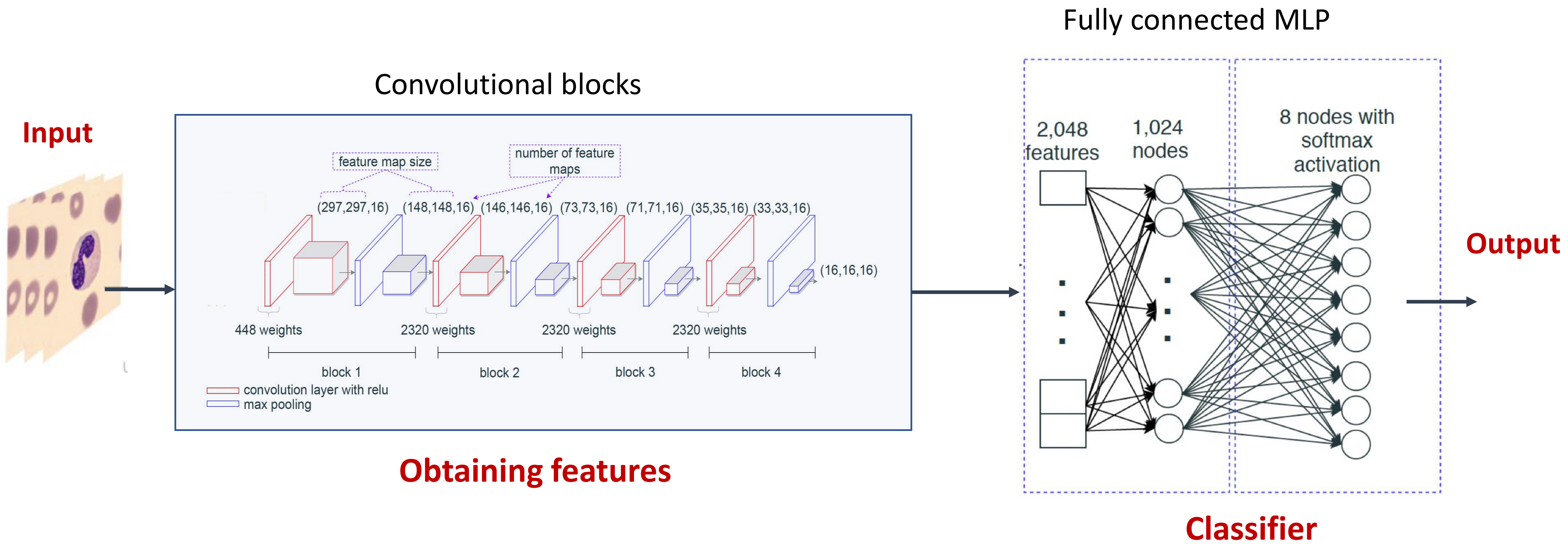
$$\begin{bmatrix} \boxed{255} & 125 & 170 \\ 255 & 150 & 150 \\ 70 & 200 & \boxed{95} \end{bmatrix}$$

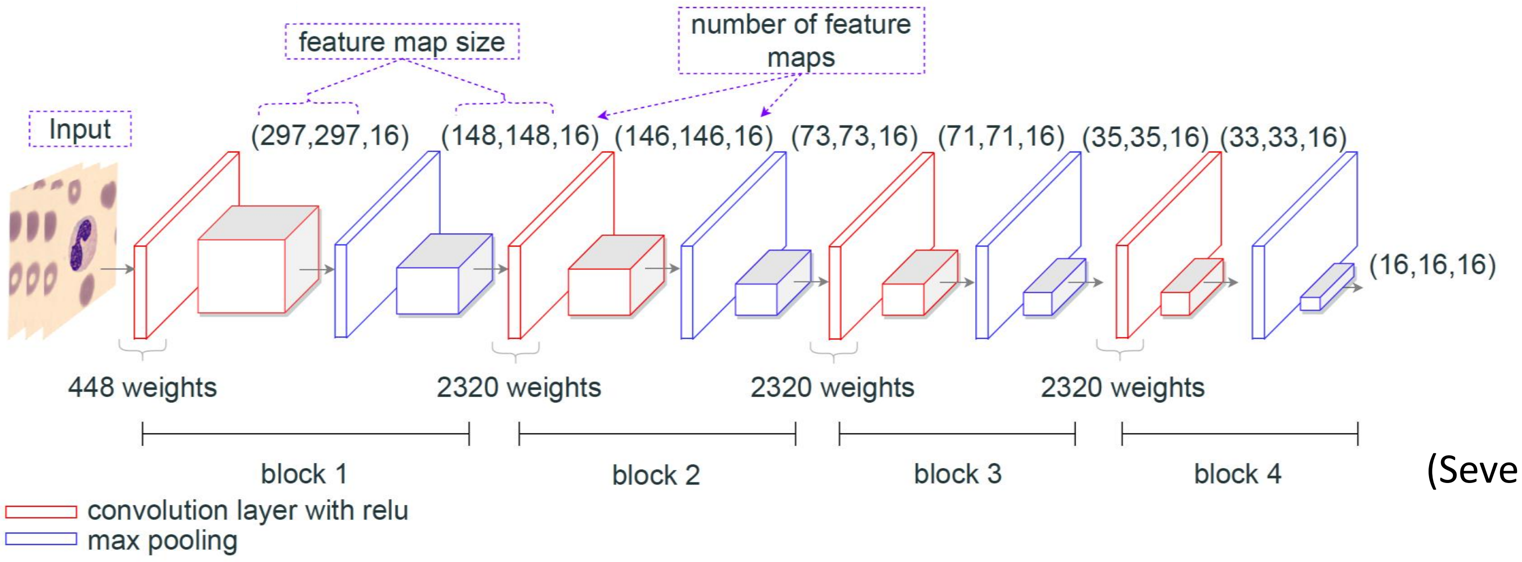
$$X2=\begin{bmatrix} \boxed{\begin{matrix}100 & 100\\ 95 & 255\end{matrix}} & \begin{matrix}100 & 50\\ 100 & 125\end{matrix} & \begin{matrix}100 & 50\\ 125 & 170\end{matrix} \\ \begin{matrix}80 & 40\\ 255 & 30\end{matrix} & \begin{matrix}10 & 10\\ 150 & 20\end{matrix} & \begin{matrix}125 & 150\\ 120 & 125\end{matrix} \\ \begin{matrix}30 & 30\\ 70 & 30\end{matrix} & \begin{matrix}150 & 100\\ 100 & 200\end{matrix} & \boxed{\begin{matrix}70 & 70\\ 70 & 95\end{matrix}} \end{bmatrix}$$

Two different input matrices (images) result into the same pooled output.

Multilayer CNN structure

Multilayer CNN architecture

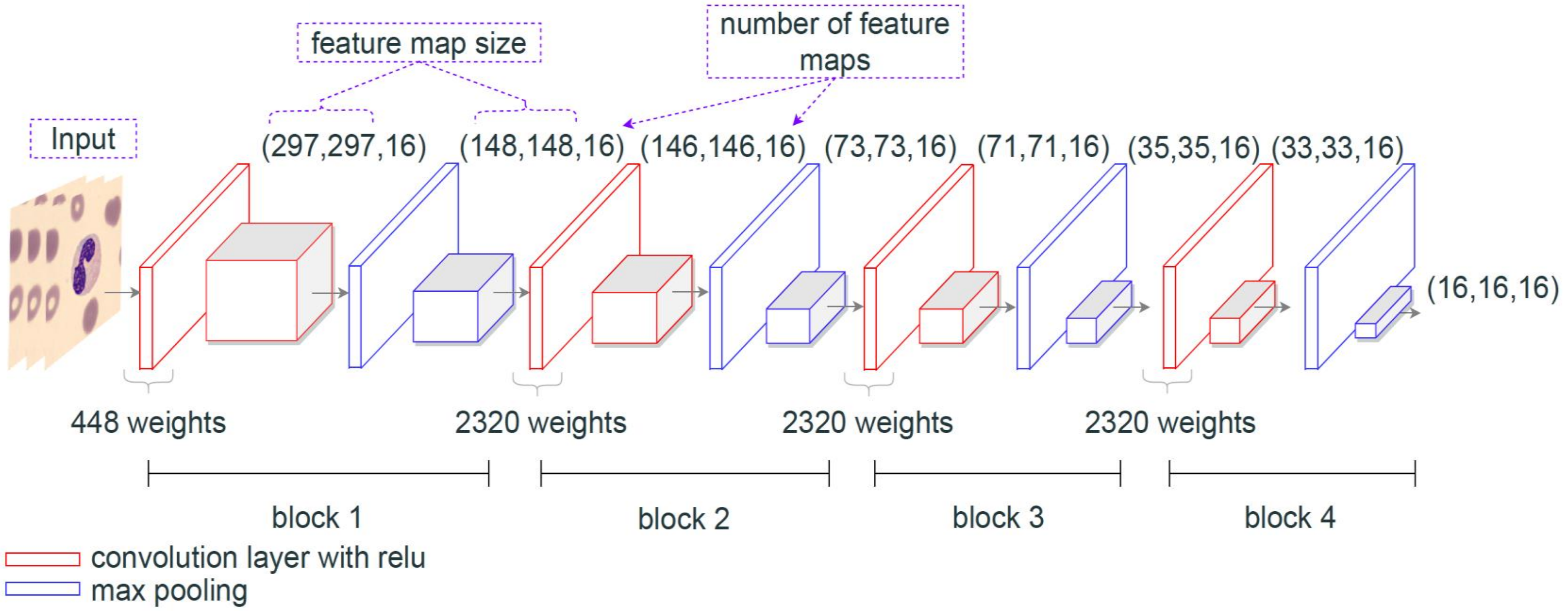


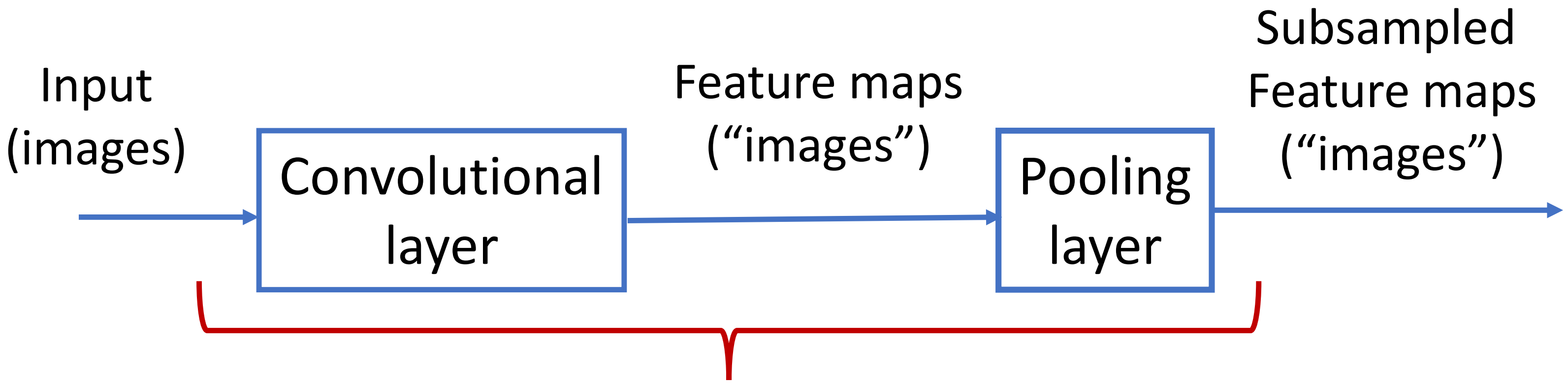


(Several convolution blocks)

- Convolutional blocks simply learn from the training set and find out which image **features** are relevant for classification.
- They are not based on handcrafted features obtained from image processing and segmentation, which saves most computational steps.
- However, more sophisticated hardware resources and larger data sets are required to train a model.

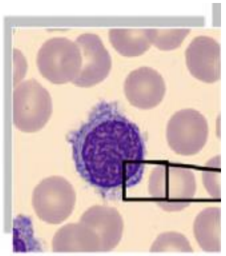
Multilayer CNN architecture





Basic block – repeated several times

Building the CNN structure - The convolutional layer



Input (images)

Convolutional layer

Feature maps ("images")

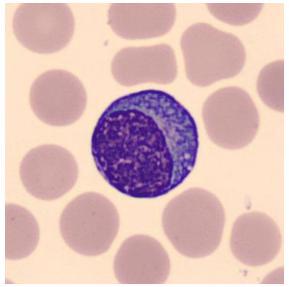
The input to the CNN is composed of planes (**channels**) with fixed width and height (n1 x n2). The number of planes defines the **depth** (*D*).

When the input is a **RGB image**, we have *D* = 3 and each plane contains the corresponding component pixel matrix in gray scale.

$$X_c : \text{Input image (matrix) of channel } c$$

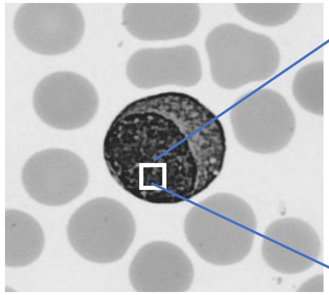
$$\text{Size } n_1 \times n_2 \qquad [c = 1, 2, \cdots, D]$$

The structure of a digital image

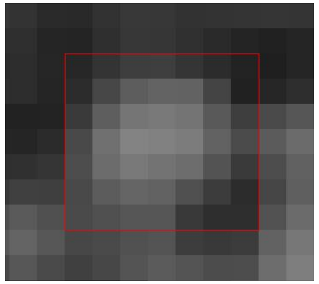


Original image

Decomposed into 3 **gray level images** (RGB)



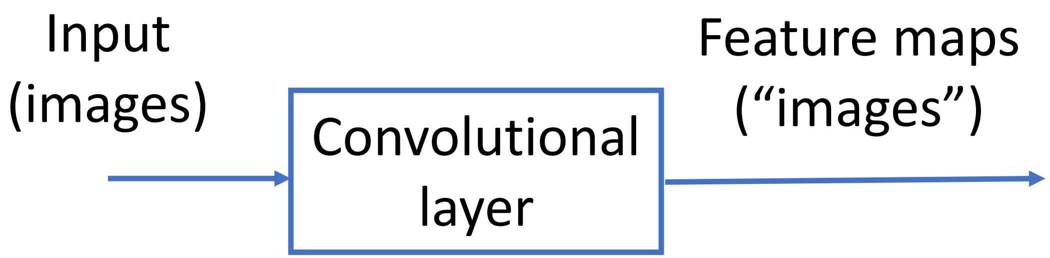
39	52	62	63	53	44	35
44	71	93	99	98	68	33
58	94	117	121	116	89	62
73	112	131	129	124	98	75
77	108	121	115	109	83	58
73	92	101	98	80	60	44
74	80	86	86	57	46	46



The gray image is a grid of **pixels** quantitatively described by the light intensity within a scale, for instance [0,255]

A single image of 360 x 360 pixels contains 129,600 values

The convolutional layer



- The convolutional layer has a number ***N*** of planes **(Filter depth).**

- Each plane is defined by a **filter (kernel)** of small size (*F*).

  For instance, *F* = 3 means that the filter is a 3 ×3 matrix with **weights**.

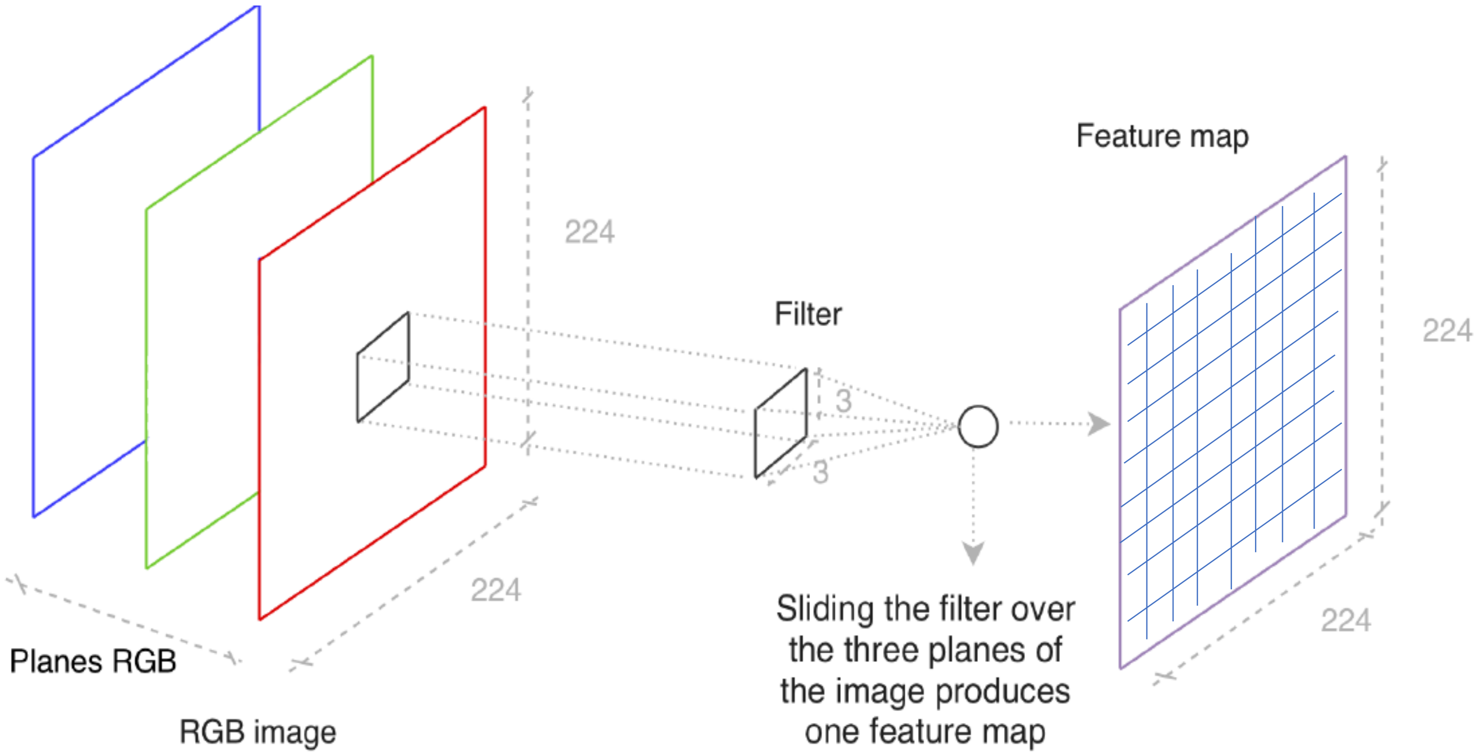
Consider a single filter acting on a single channel and the associated convolution:

$$W_{cj} \; : \; \text{Filter matrix } j \text{ applied to channel } c \qquad [c = 1, 2, \cdots, D]$$

$$\text{Convolution matrix } (X_c * W_{cj}) \qquad [j = 1, 2, \cdots, N]$$

Convolutional filters

They are the key elements to extract features in CNN models



The convolutional layer

Input (images) → Convolutional layer → Feature maps ("images")

These convolutions are added for all the input channels to have a single matrix for each filter *j* :

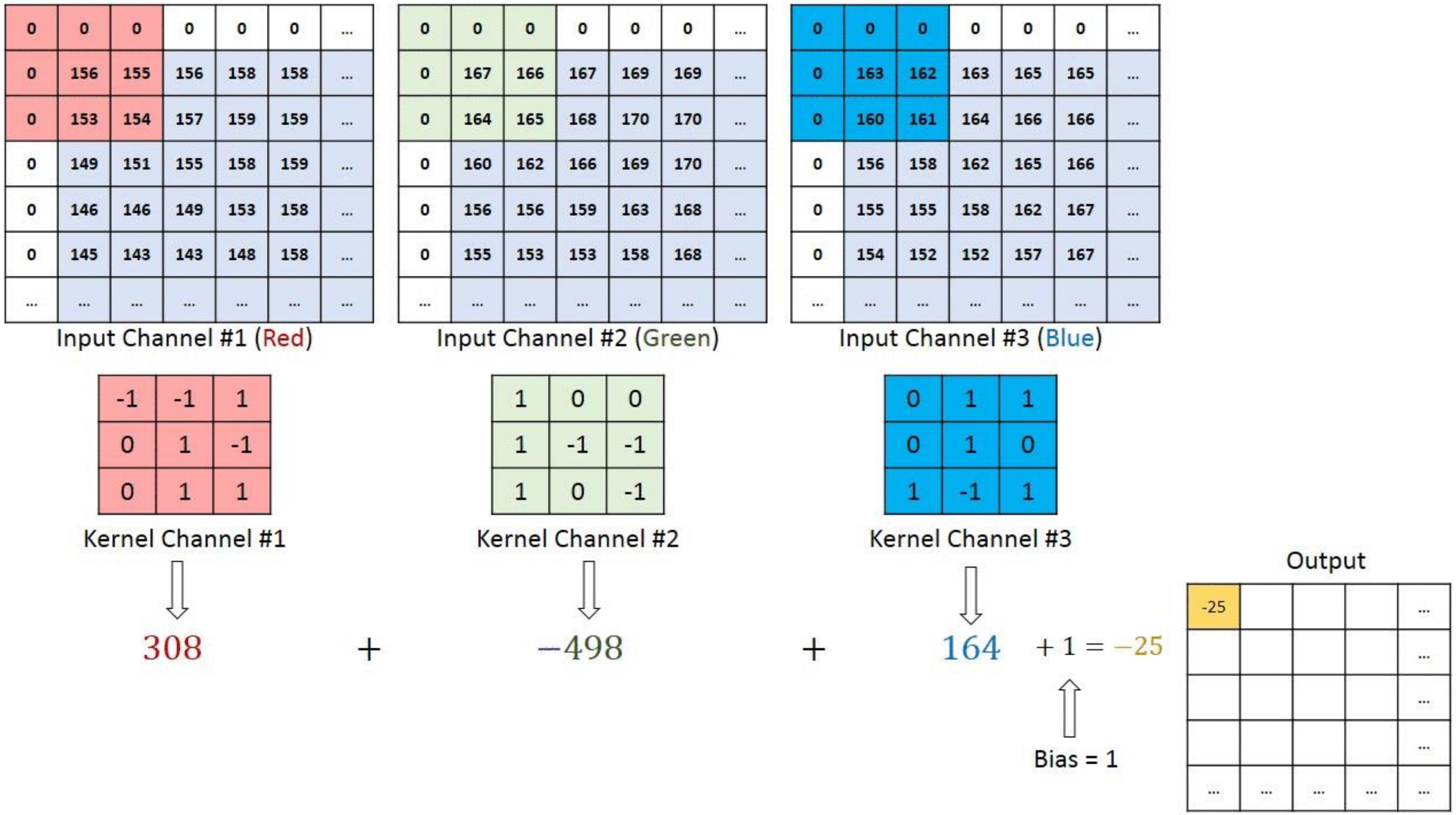
$$Z_j = \sum_{c=1}^{D} X_c * W_{cj}$$

In a similar way as for the multilayer perceptron, we add a **Bias term** and apply an **activation function:**

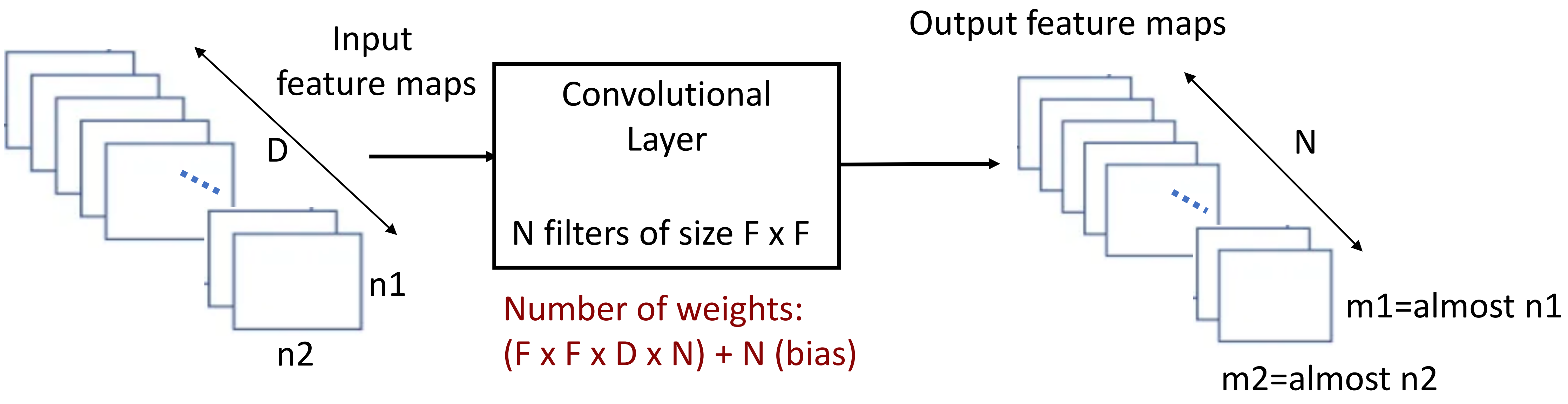
$$Y_j = \phi(B_j + Z_j) \qquad [j = 1, 2, \cdots, N]$$

They are called Feature Maps

Example of 3D convolution

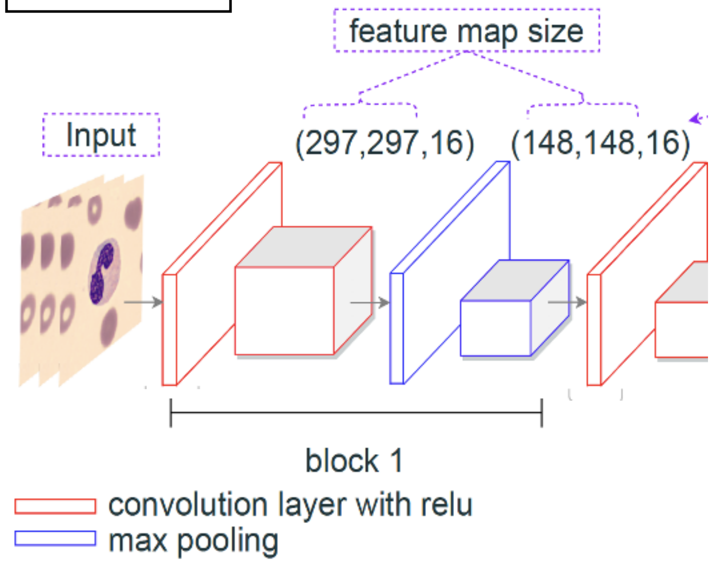


Summary



Number of weights:
(F x F x D x N) + N (bias)

Example



D=3 (RGB channels)
N= 16 filters
F = 3
Input size 299 x 299
Output size 297 x 297

448 weights

Filter depth

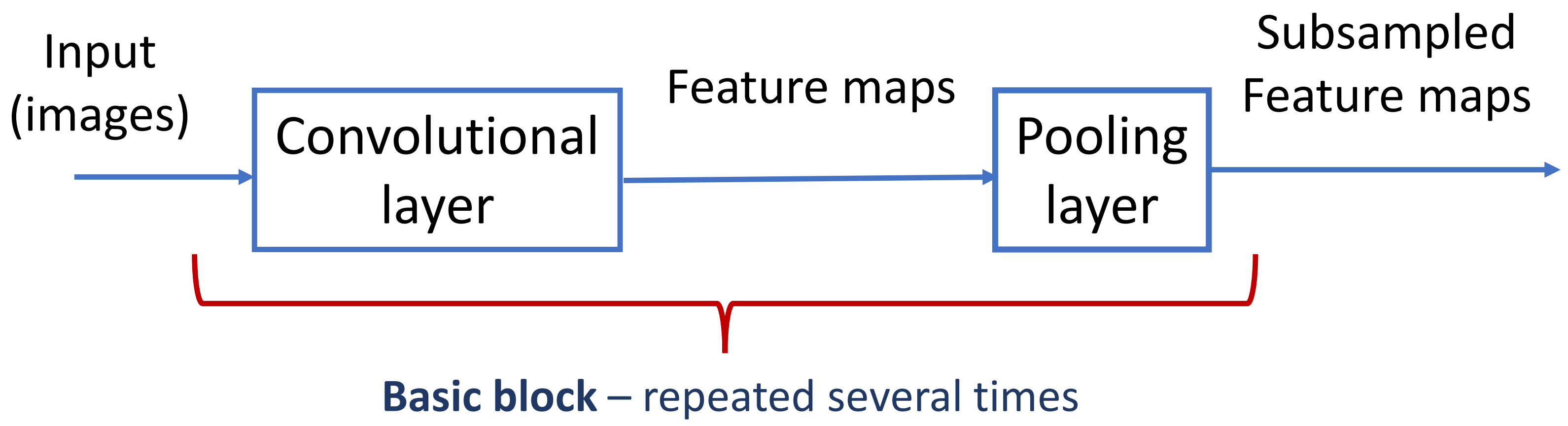
It is common to **use several filters**. Specific filters capture specific characteristics. For example, one filter might look for a particular color, while another might look for a kind of object of a specific shape.

This part of the dog has many **interesting features**: teeth, whiskers, and the pink color of the tongue.

Having **multiple neurons (filters)** for a given patch ensures that the CNN can learn to capture significant characteristics.

Remember that the CNN is not "programmed" to look for certain features. It **learns on its own** which features are relevant for classification.

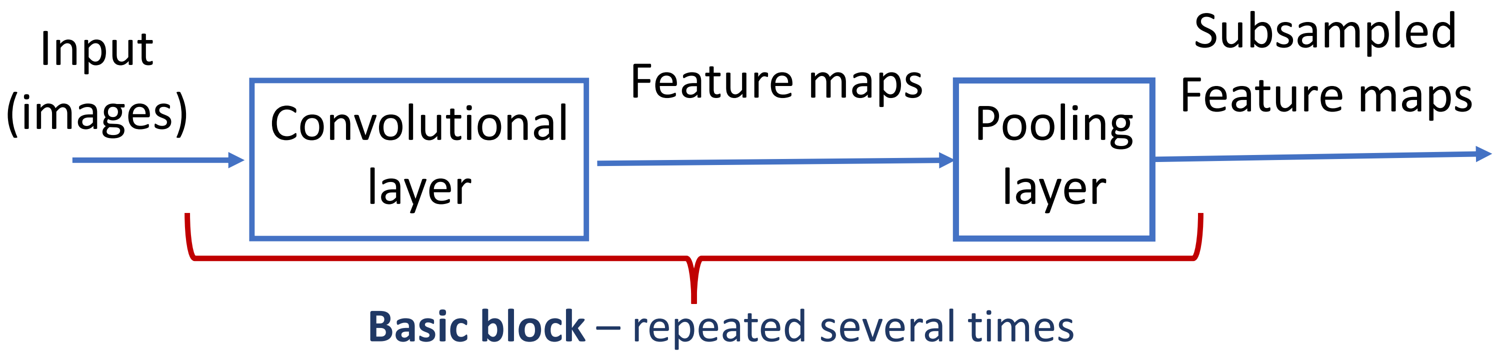
Pooling layer



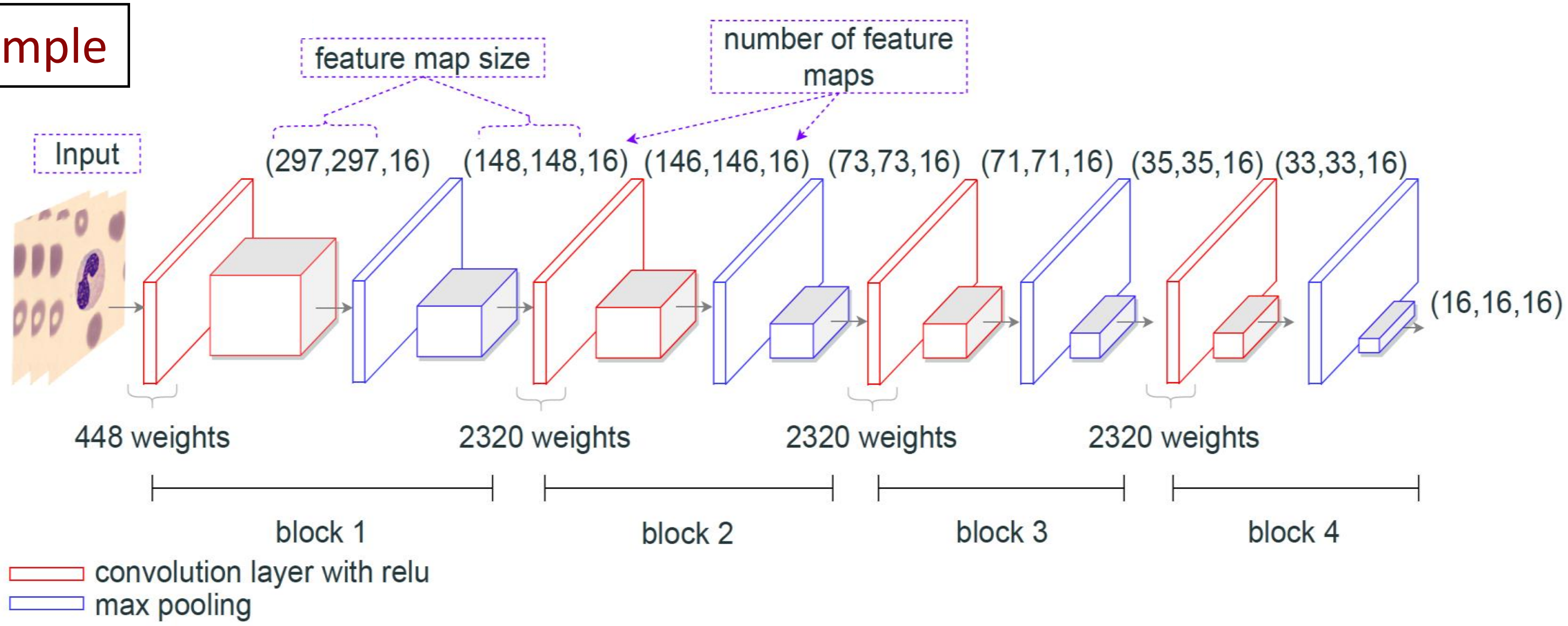
The size of feature maps produced by the convolutional layer depend on the padding and stride in the convolutions. Usually, they are adjusted to keep the same dimension as the input.

The feature maps are downsampled by the **pooling layer** giving a new map with reduced size.

Pooling layer

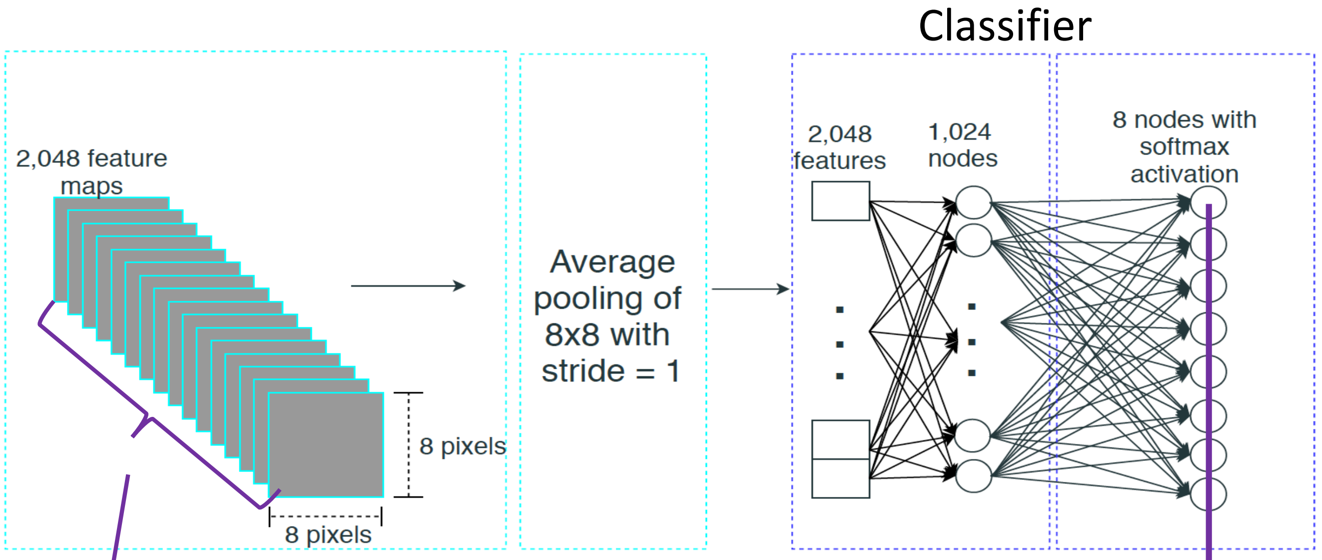


Example



After each convolution layer, we placed a max-pooling layer, which takes the maximum value from a 2 x 2 frame of pixels from each feature map.

Classification



The number of nodes is the number of classes

- ➢ The last feature maps are put in a single array, which is the input to the classifier.

The outputs are the **probabilities** predicted for each class

Training the CNN

The **training** is performed in a similar manner as for the multilayer perceptron using a training set of labelled images:

1) Images are propagated **forward**

2) Parameters are updated incrementally using the **backpropagation** approach using a loss function and gradient descent principle.

Trainable parameters

- Parameters associated with the kernel filters
- Bias for each output feature map of the convolutional layer

$$\left.\right\} (F \times F \times D \times N) + N$$

- Pooling layers do not have any (trainable) parameters;

Suppose that we used a **fully connected neural network** instead a CNN

$$\underbrace{(n_1 \times n_2 \times D)}_{\text{Input}} \times \underbrace{(m_1 \times m_2 \times N)}_{\text{Output}}$$

$$F \ll n_i, m_i \Rightarrow \text{a significant reduction in trainable parameters using CNN}$$

Thanks for your kind attention