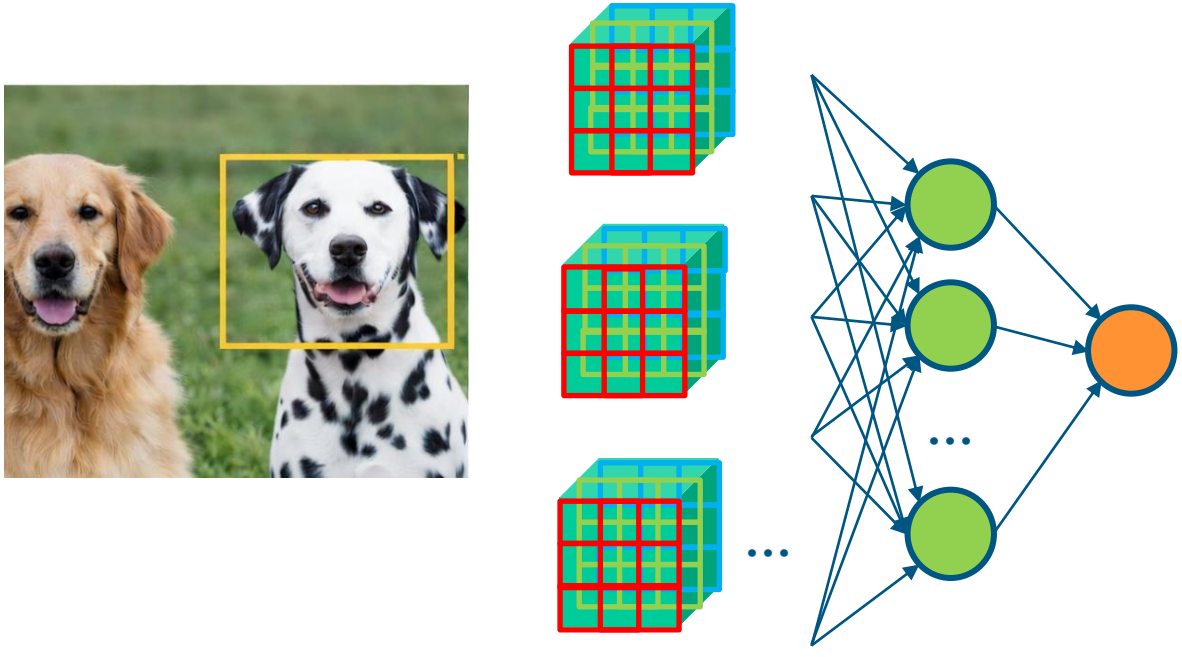
Redes Neuronales y Aprendizaje Profundo

Bloque 2: Convolutional Neural Networks

Fundamentals



Convolutional Neural Networks

- **Objectives**
  - **Explain the limitations** of fully connected neural networks when applied to image data **and justify the need for convolutional architectures**.
  - **Describe the convolution operation** in convolutional neural networks, including the role of kernels, feature maps, stride, and padding.
  - **Analyze the effect of basic architectural choices** (e.g., number of filters, kernel size, pooling) on the dimensionality of feature maps.
  - **Interpret how spatial structure and local connectivity are exploited by CNNs** to learn meaningful visual representations.

Convolutional Neural Networks

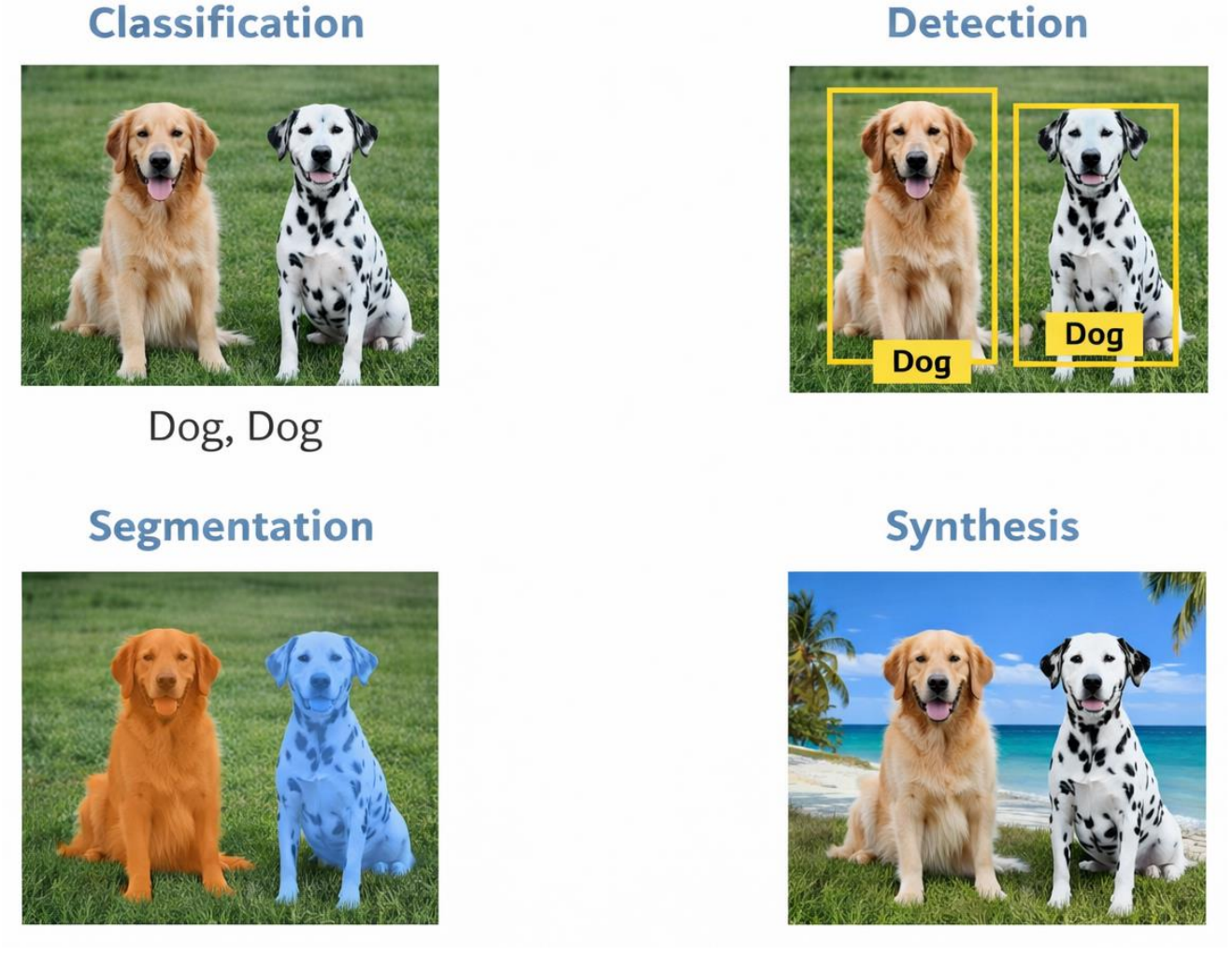
- **Objectives**
  - **Identify and explain the main building blocks of a CNN**, including convolutional layers, activation functions, pooling layers, and fully connected layers.
  - **Design a simple CNN architecture** for an image classification task at a conceptual level.

Contents

- Motivation
- The Convolution Operation
- Core Building Blocks of a CNN
- Hierarchical Feature Learning
- Designing a Basic CNN Architecture

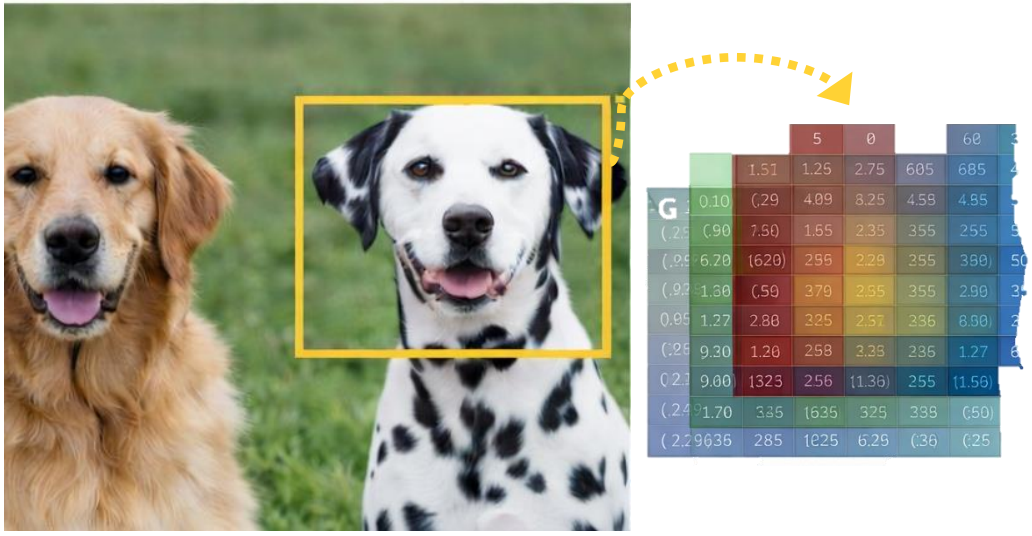
Motivation

- Convolutional Neural Networks (CNNs) are specifically tailored for computer vision tasks:



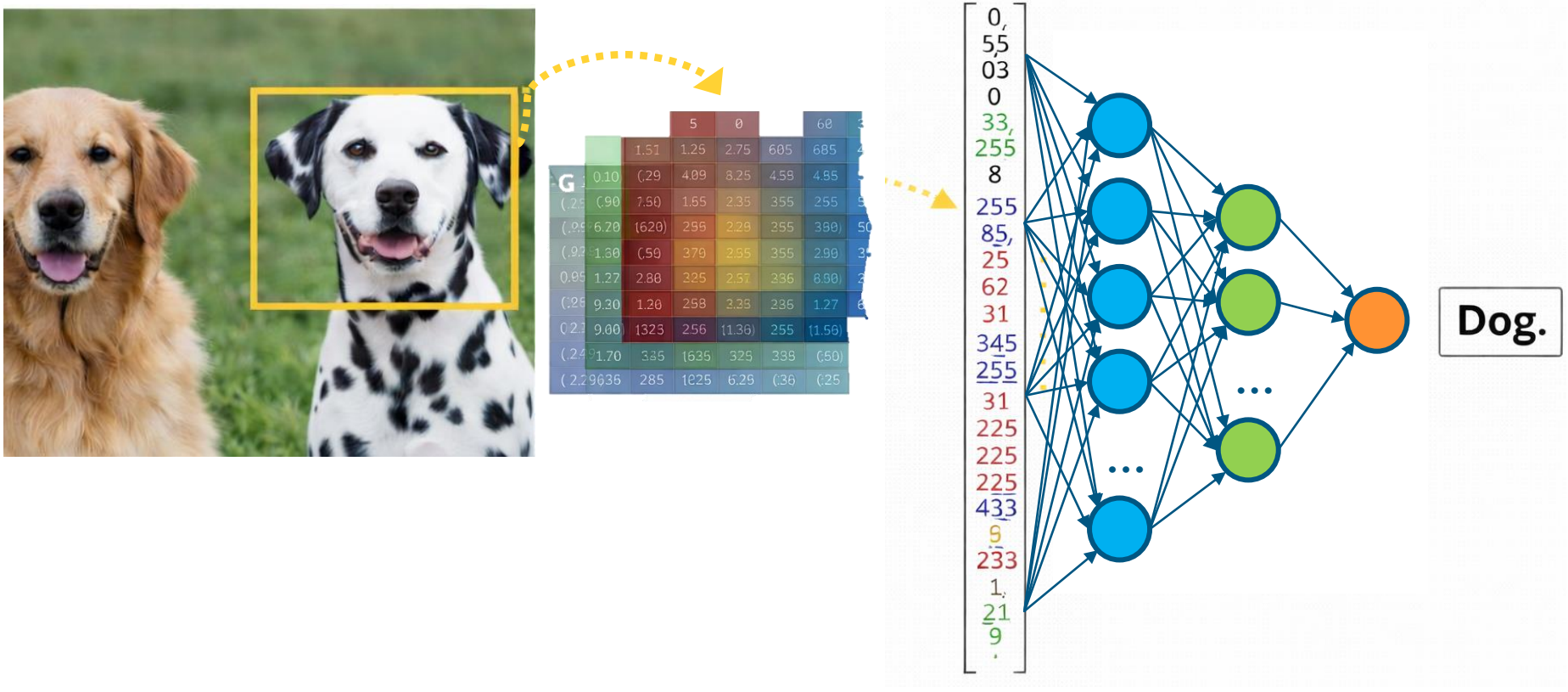
Motivation

- How we can use a FCN to classify images?



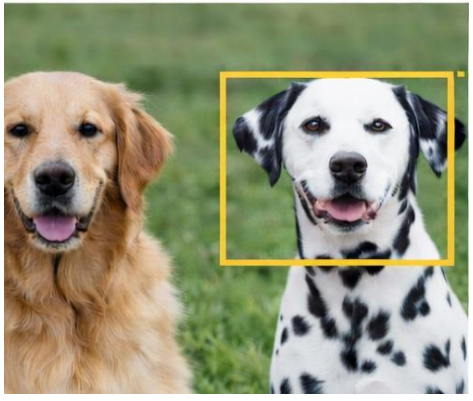
Motivation

- How we can use a FCN to classify images?

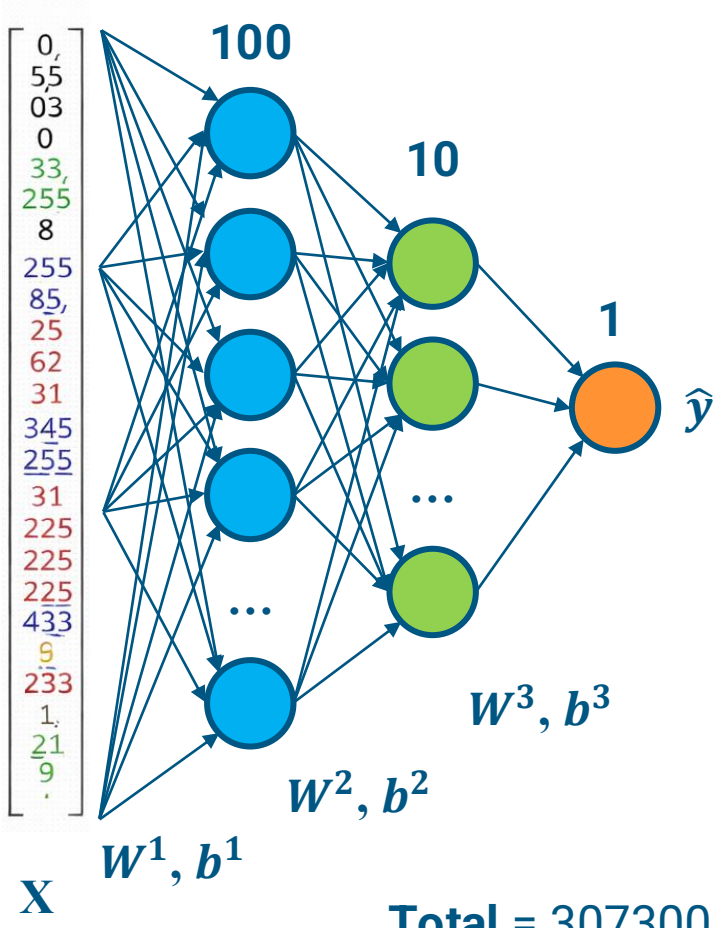


Motivation

- How many **parameters** do we have to learn?



32x32x3



X → 3072

First layer

$W^1 \rightarrow$ 3072x100

$b^1 \rightarrow$ 100

Second layer

$W^2 \rightarrow$ 100x10

$b^2 \rightarrow$ 10

Output layer

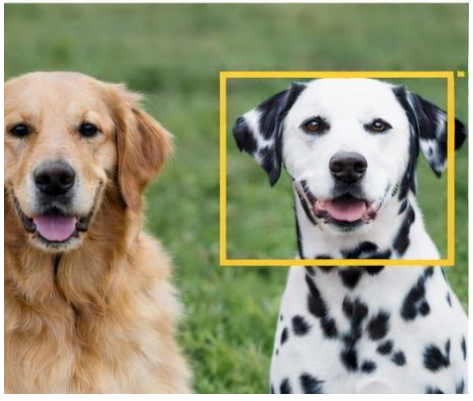
$W^3 \rightarrow$ 10x1

$b^3 \rightarrow$ 1

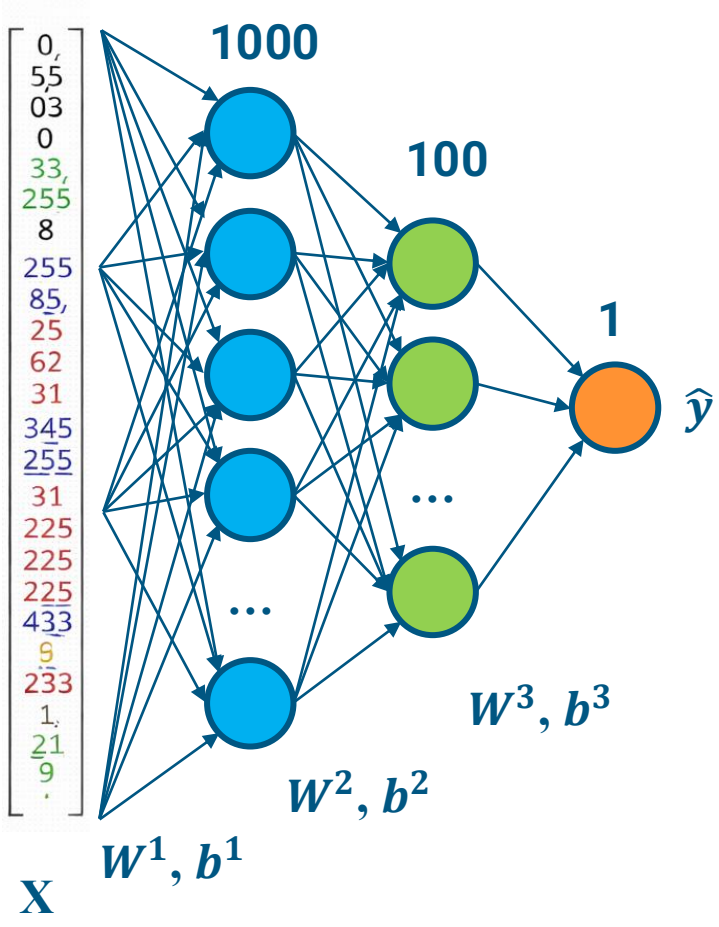
Total = 307300 + 1010 +11 = **308.321**

Motivation

- And now?



1024x1024x3



X → 3M

First layer

$W^1 \rightarrow$ 3M x 1000

$b^1 \rightarrow 1000$

Second layer

$W^2 \rightarrow$ 1000x100

$b^2 \rightarrow 100$

Output layer

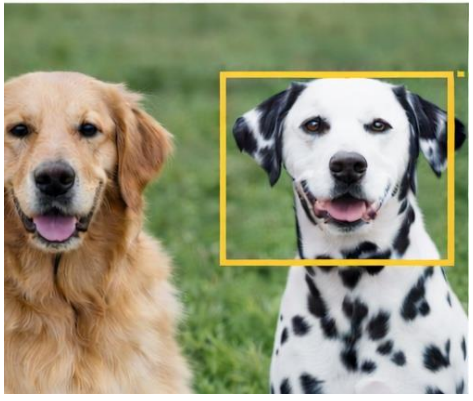
$W^3 \rightarrow$ 100x1

$b^3 \rightarrow 1$

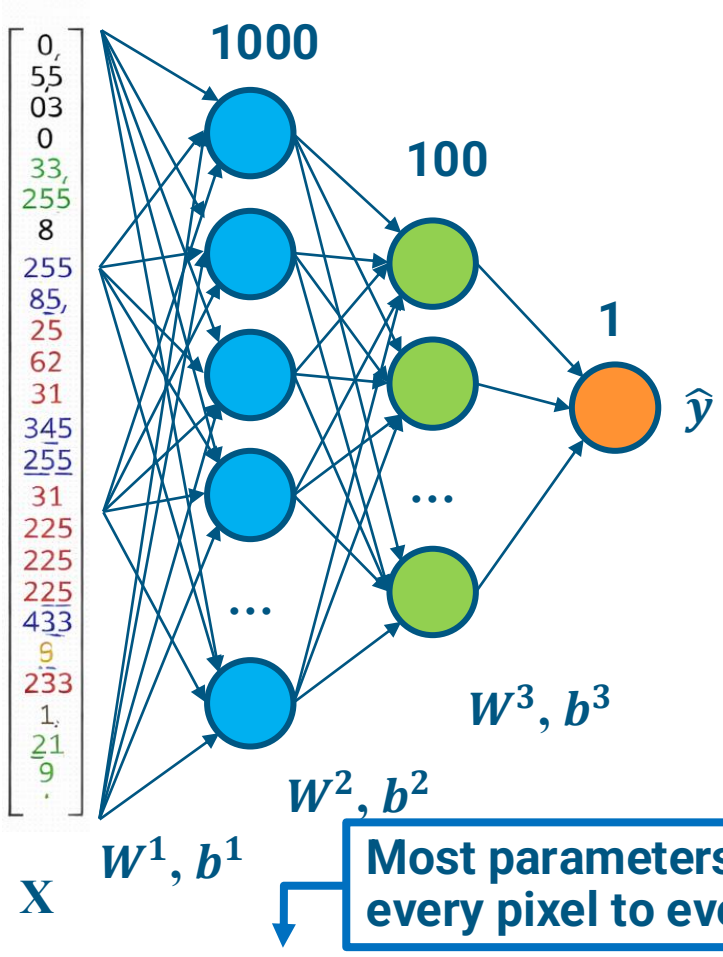
Total = 3.145.729.000 + 100.100 + 101 = **3.145.829.201**

Motivation

- And now?



1024x1024x3



X → 3M

First layer

$W^1 \rightarrow$ 3M x 1000

$b^1 \rightarrow$ 1000

Second layer

$W^2 \rightarrow$ 1000x100

$b^2 \rightarrow$ 100

Output layer

$W^3 \rightarrow$ 100x1

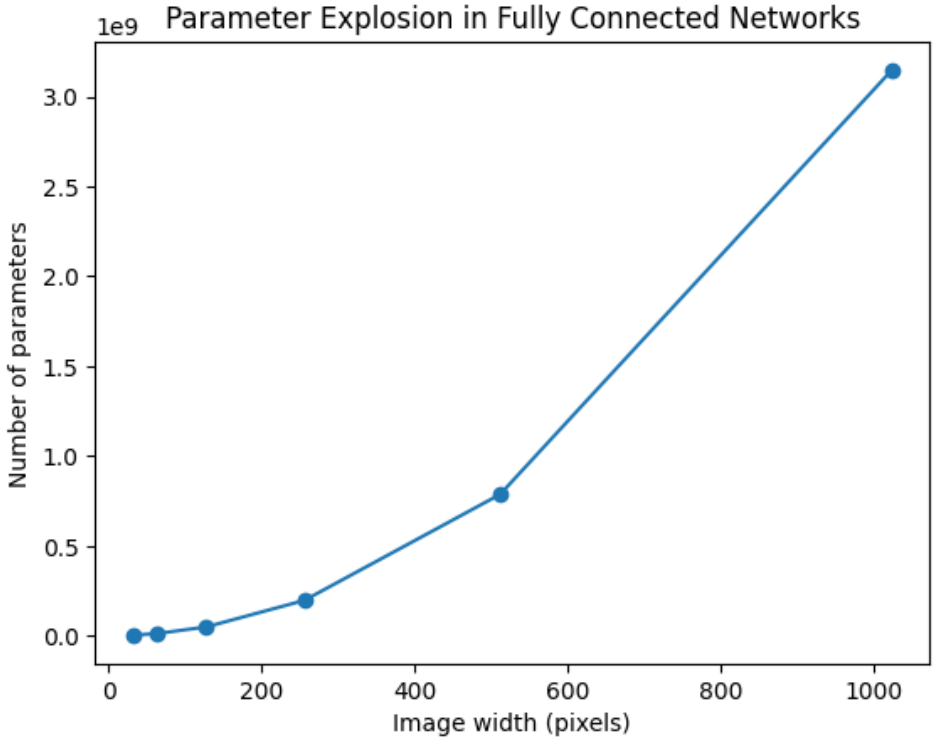
$b^3 \rightarrow$ 1

Most parameters are wasted connecting every pixel to every neuron.

Total = 3.145.729.000 + 100.100 + 101 = **3.145.829.201**

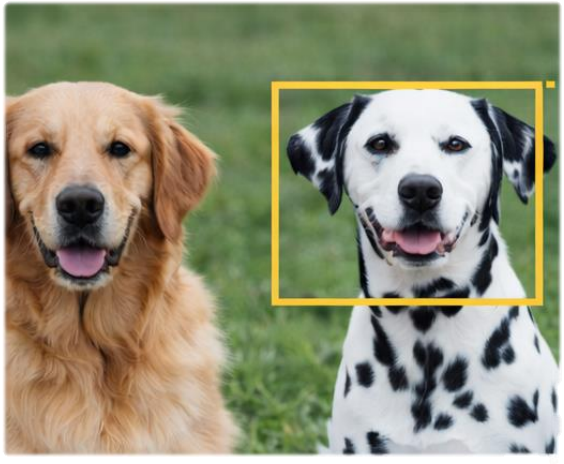
Motivation

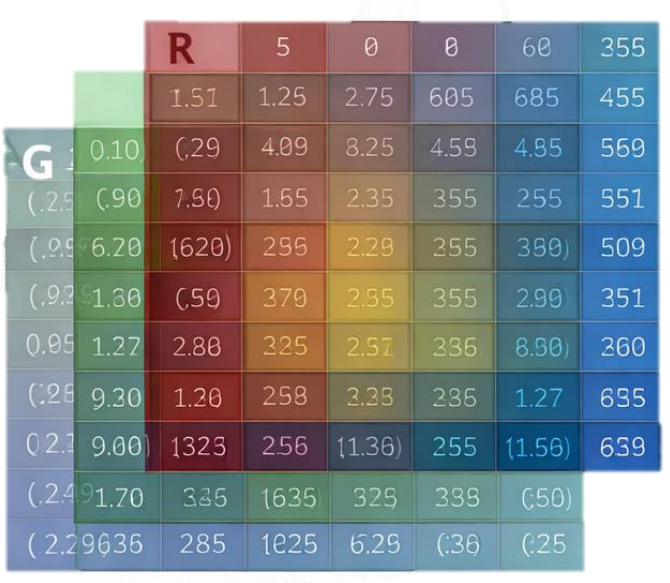
- **Parameter Explosion**
  - Parameters grow linearly with input size
  - High risk of overfitting
  - Computationally inefficient
- **Fully connected networks do not scale to realistic image sizes**



Motivation

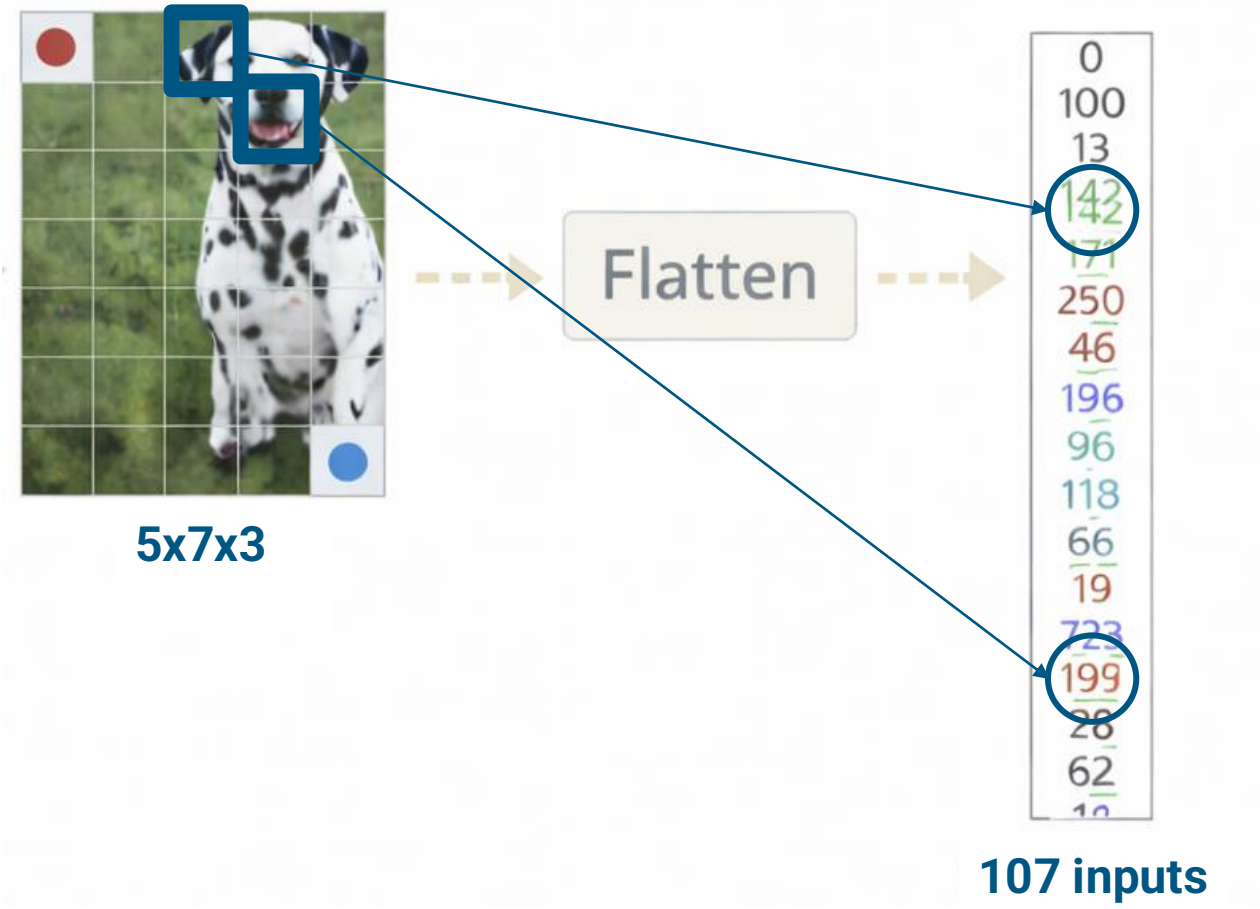
- **Images are not just vectors**
  - An image is a 3D tensor: H × W × C
  - Strong spatial correlations between neighboring pixels
  - Local patterns (edges, corners, textures) matter more than absolute pixel values





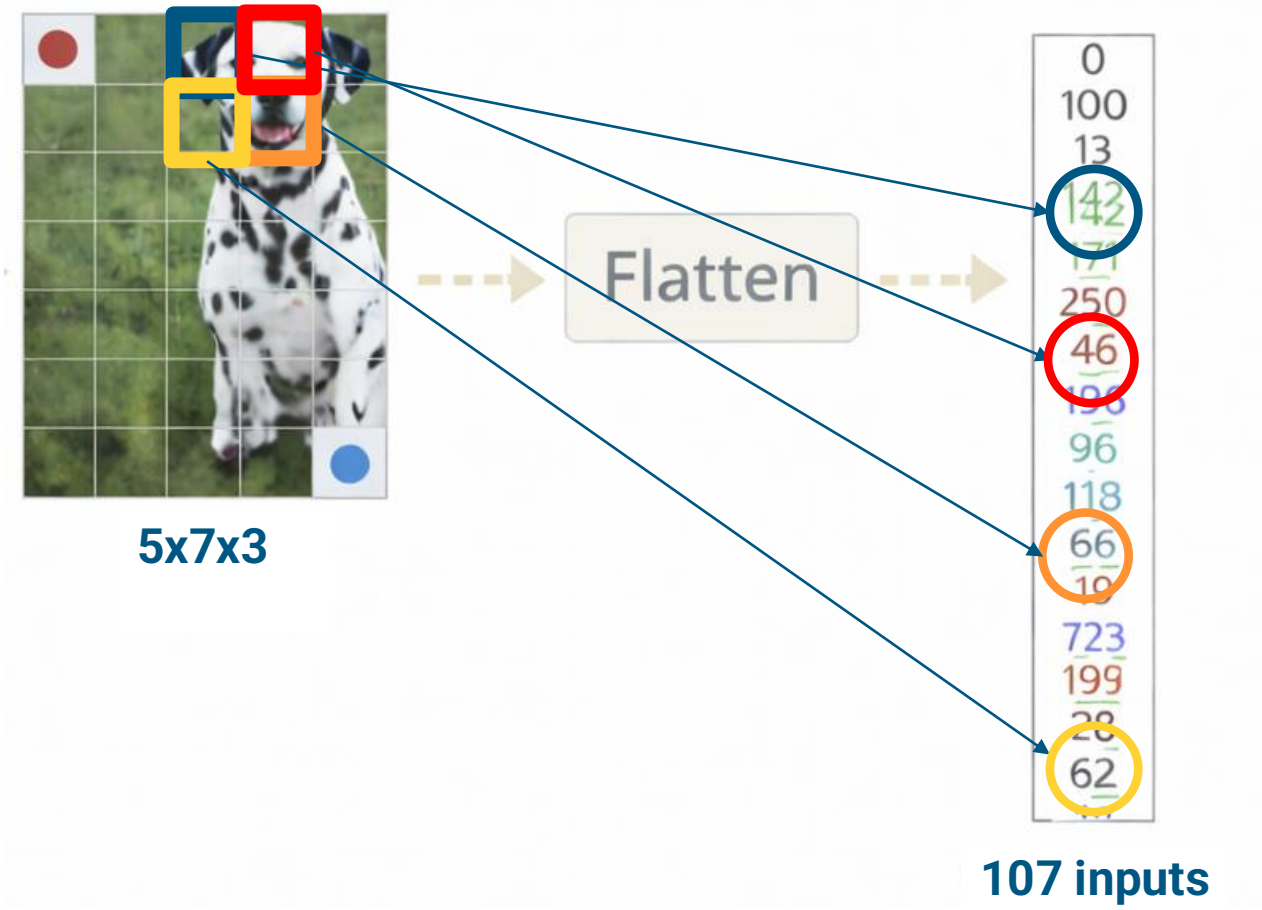
Motivation

- **Loss of spatial information.** Flattening removes:
  - Local neighborhoods



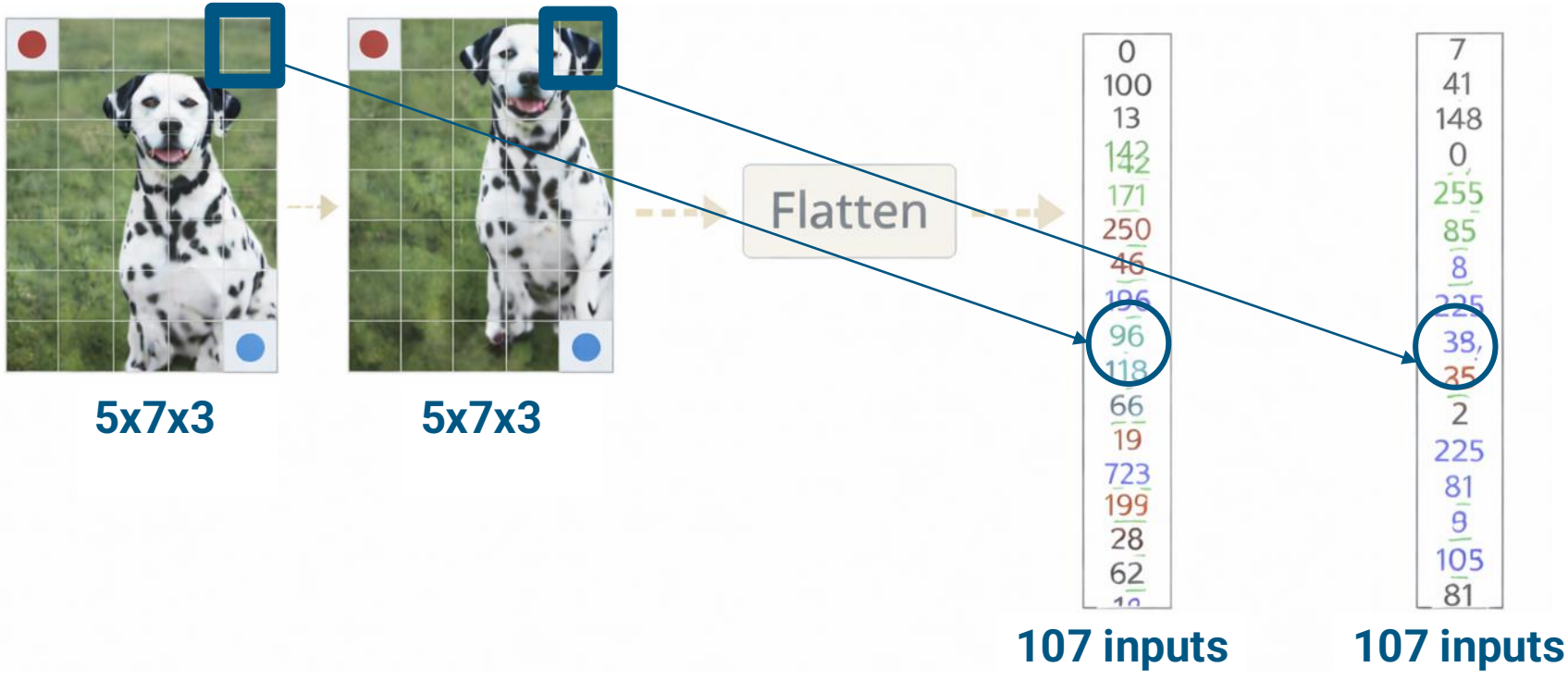
Motivation

- **Loss of spatial information.** Flattening removes:
  - Local neighborhoods
  - Relative positions: vision relies on spatial relationships: edges, corners, and object parts are defined by their relative positions.



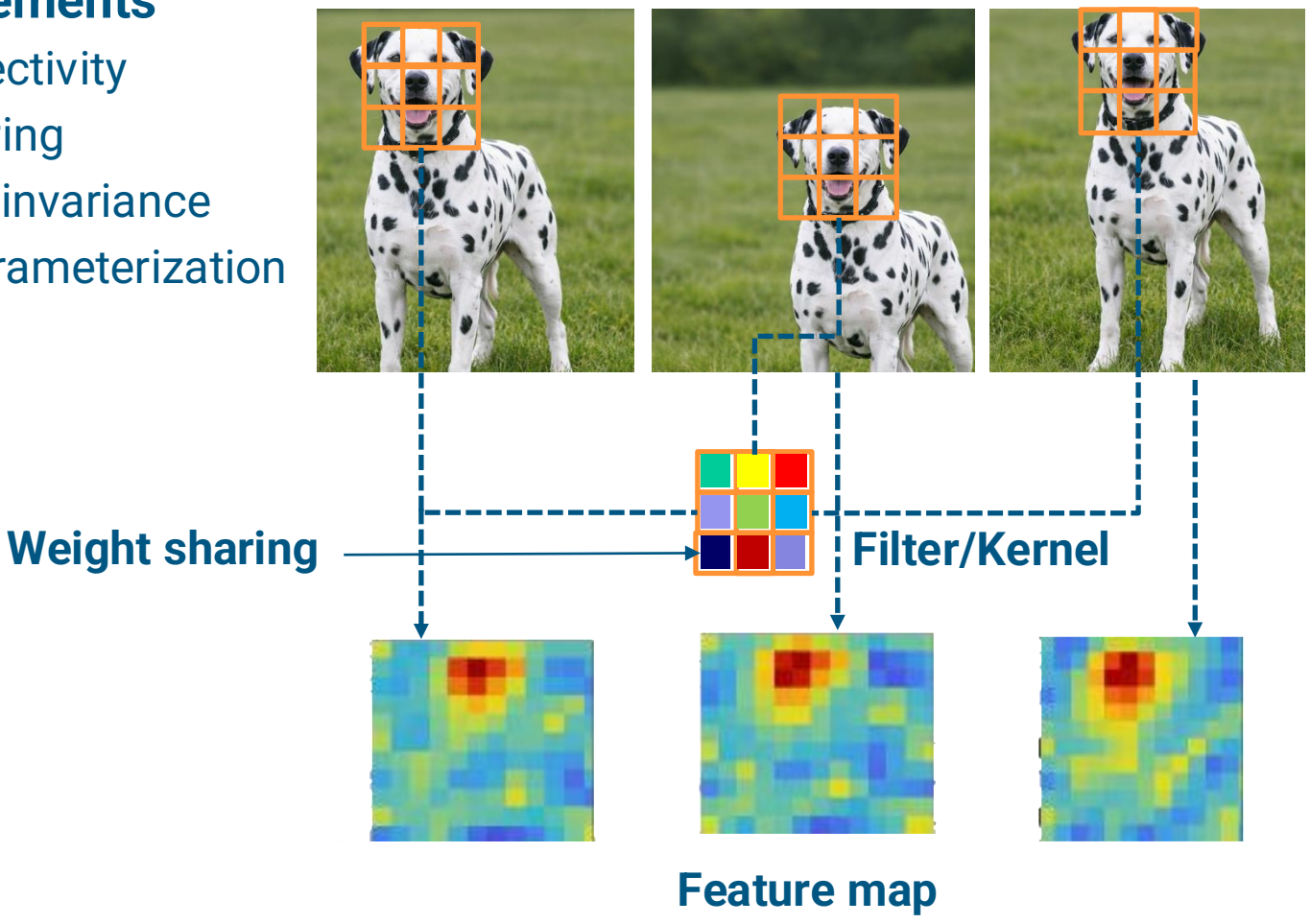
Motivation

- **Loss of spatial information.** Flattening removes:
  - Local neighborhoods
  - Relative positions
  - Translation awareness: network doesn't understand they are the same pattern but traslated



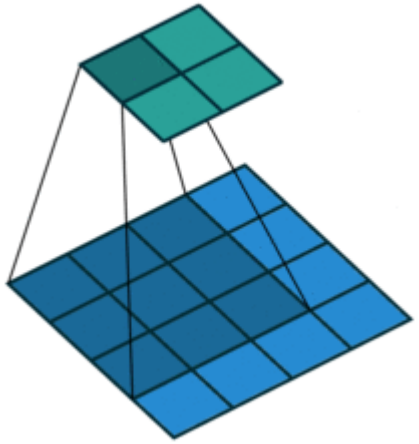
Motivation

- If patterns are local and can appear anywhere in the image → we should **learn them locally and reuse them everywhere**.
- **Design requirements**
  - Local connectivity
  - Weight sharing
  - Translation invariance
  - Efficient parameterization



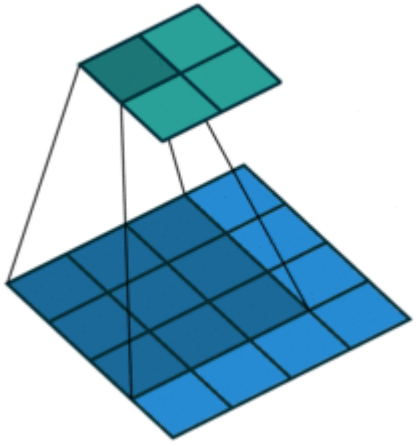
Convolutional operation

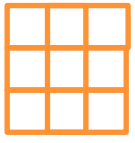
- **Parameters**
  - Filter size
  - Stride
  - Padding



Convolutional operation. **Parameters**

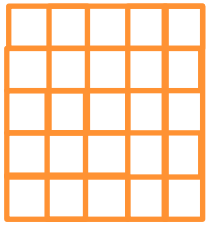
- **Filter size**
  - **Local Receptive Fields**
    - Each neuron sees only a **small region of the image**
      - Not the whole image
      - Only a local patch
      - Same size as the kernel





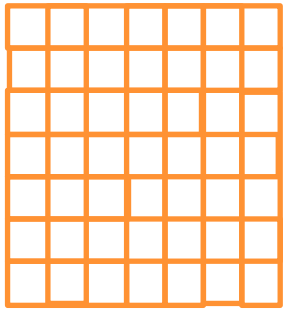
3x3

- Used in most modern CNNs
- VGG, ResNet…
- 9 params



5x5

- Early CNNs
- Inception
- 25 params.



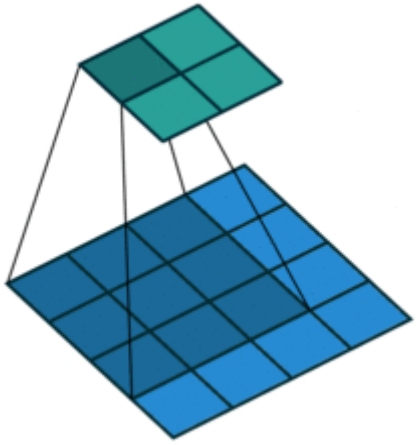
7x7

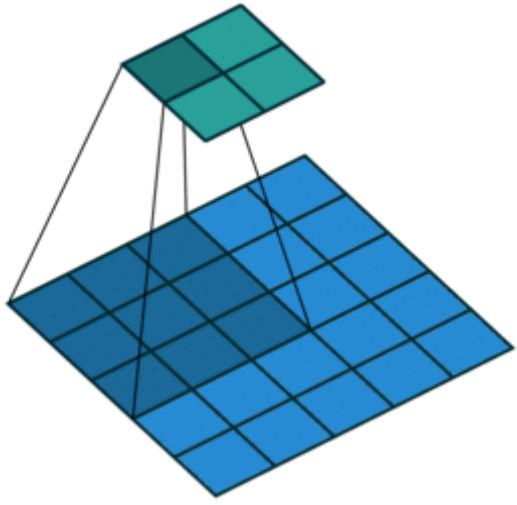
- Modern lightweight networks avoid large kernels
- First layer ResNet
- 49 params

Explicar bias

Convolutional operation. **Parameters**

- **Stride**
  - Defines the **step size** of the kernel as it moves across the image.
  - Stride = 1 → the filter moves one pixel at a time.
  - Stride > 1 → the filter skips positions.
  - Larger stride → smaller output feature map.
  - Increasing stride reduces spatial resolution and computational cost.

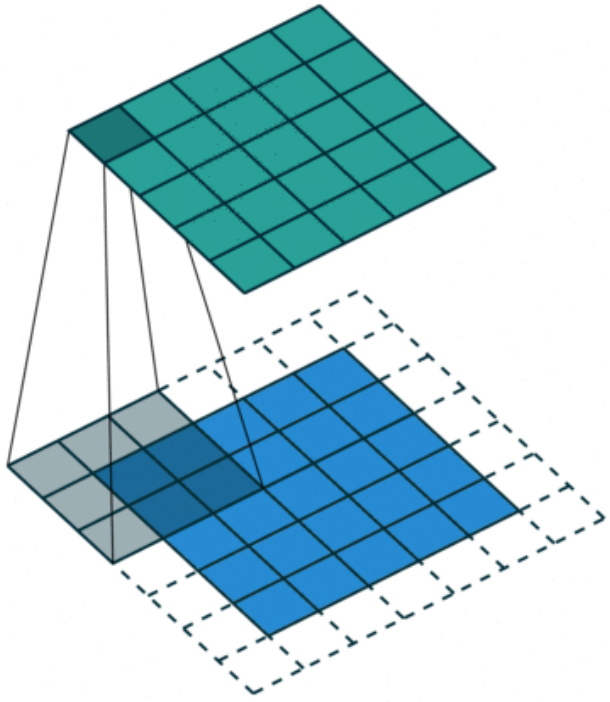




  - Most convolutions use stride = 1.
  - Stride = 2 is typically used for spatial downsampling.
  - Stride > 2 → rare

Convolutional operation. **Parameters**

- **Padding**
  - Adding borders to the image
  - Prevents shrinking
  - Preserves spatial dimensions
  - Helps detect patterns at borders
  - Types:
    - Valid (no padding)
    - Same (keeps size)



- Example
  - What is the output size using a filter of 3x3 and stride 1 for an image of 32x32?
    - Without padding
    - With padding = 1

Convolutional operation. **Parameters**

- **Exercise**: Calculate the feature map for the following parameters

  - Image 64x64
  - Filter 3x3
  - Stride 1
  - Padding 0

  - Image 64x64
  - Filter 3x3
  - Stride 2
  - Padding 0

  - Image 64x64
  - Filter 3x3
  - Stride 1
  - Padding 1

  - Image 64x64
  - Filter 5x5
  - Stride 2
  - Padding 2

Convolutional operation. **Parameters**

- **Exercise**: Calculate the feature map for the following parameters

  - Image 64x64
  - Filter 3x3
  - Stride 1
  - Padding 0

  → **62 x 62**

  - Image 64x64
  - Filter 3x3
  - Stride 1
  - Padding 1

  → **64 x 64**

  - Image 64x64
  - Filter 3x3
  - Stride 2
  - Padding 0

  → **31 x 31**

  - Image 64x64
  - Filter 5x5
  - Stride 2
  - Padding 2

  → **32 x 32**

Stride controls downsampling, while padding controls size preservation.

Convolutional operation. **Parameters**

- **Size of the feature map**
  - The result of a convolution filter with size (fxf) to an image of (h,w) size with a padding p is:

$$(h + 2p - f + 1,\ w + 2p - f + 1)$$

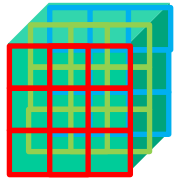
  - The result of a convolution filter with size (f·f) to an image of (h,w) size with a padding p and a stride of s is:

$$\left(\left\lfloor \frac{h + 2p - f}{s} \right\rfloor + 1,\ \left\lfloor \frac{w + 2p - f}{s} \right\rfloor + 1\right)$$

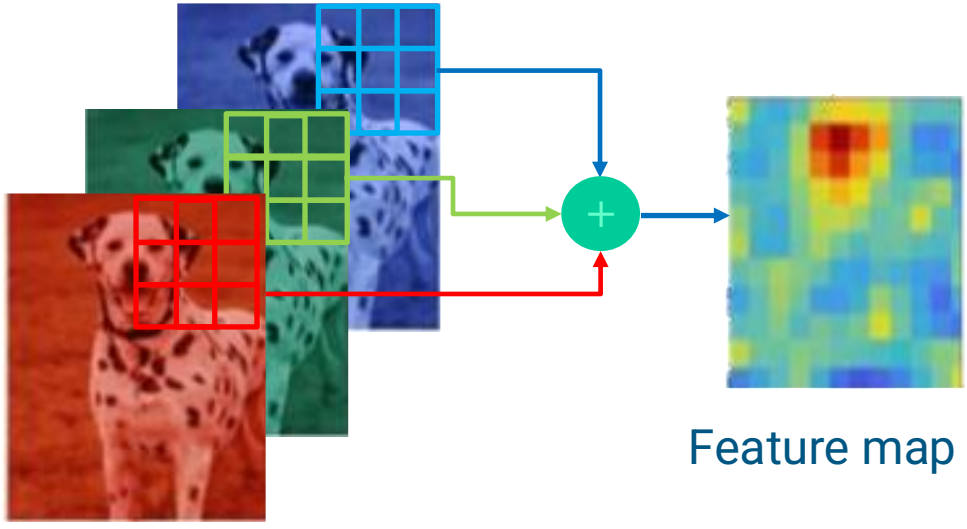
Convolutional operation. **Volumes**

- Convolution over volumes. Each filter:
  - Spans all input channels
  - Produces one feature map

Input (H×W×3)



Kernel (3×3×3)

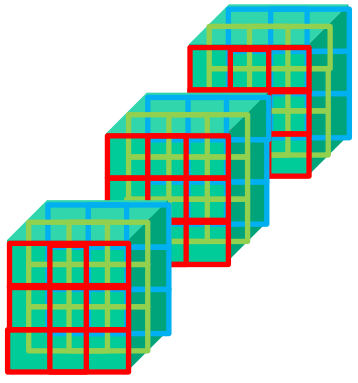


Feature map $\left(\left(\left\lfloor\frac{H+2p-f}{s}\right\rfloor+1, \left\lfloor\frac{W+2p-f}{s}\right\rfloor+1\right)\right)$

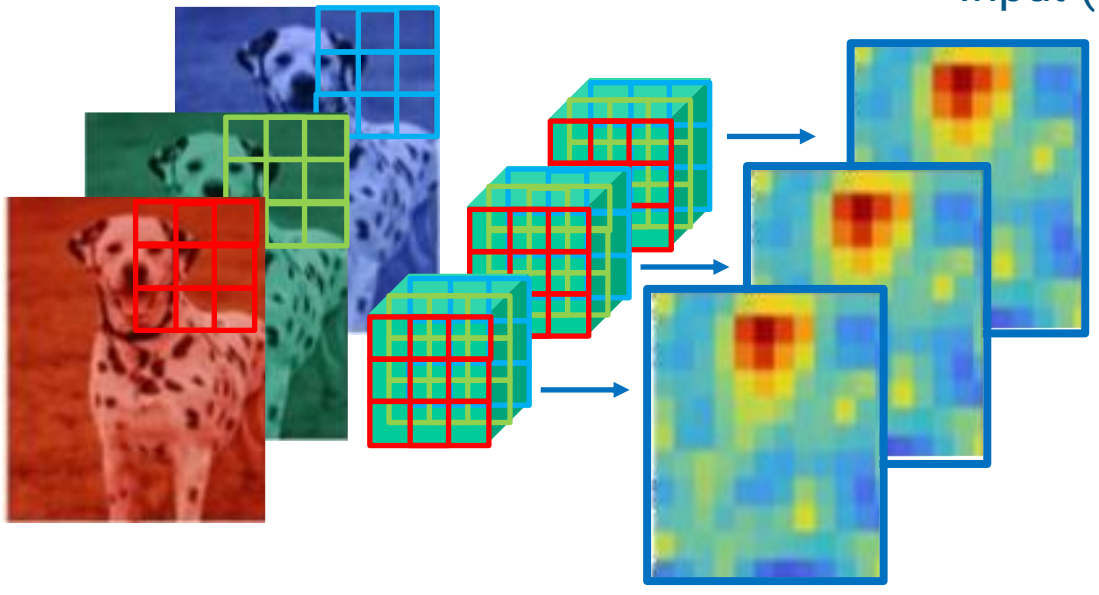
Convolutional operation

- **Multiple filters**
  - Each filter learns different things (vertical and horizontal borders, corners,….)
  - Filters compete to specialize

Input (H×W×3)



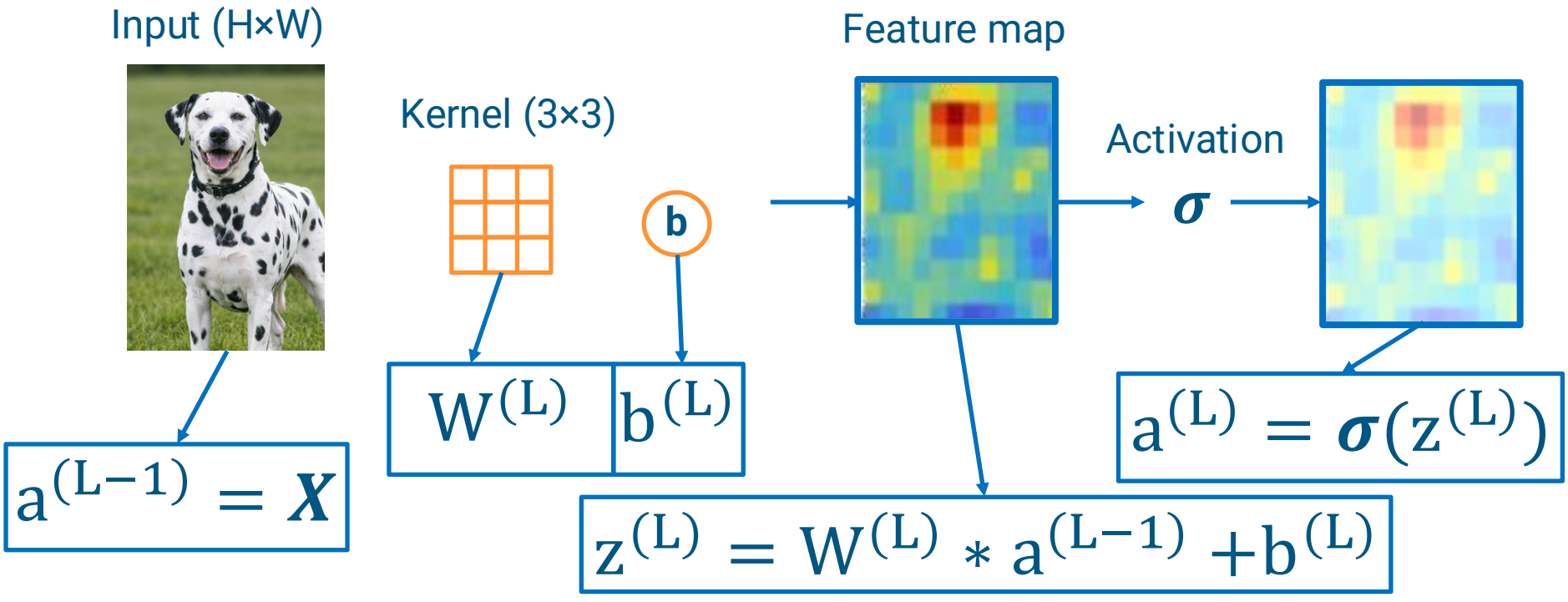
N Kernels (3×3×3)



N Feature maps $\left(\left(\left\lfloor\frac{H+2p-f}{s}\right\rfloor+1, \left\lfloor\frac{W+2p-f}{s}\right\rfloor+1\right)\right)$

Convolutional building blocks

- All together. Including bias



$*$ **Convolution**

$$z^{(L)}_{i,j,k} = \sum_{m,n,c} W^{(L)}_{m,n,c,k} \, a^{(L-1)}_{i+m,j+n,c} + b^{(L)}_{k}$$

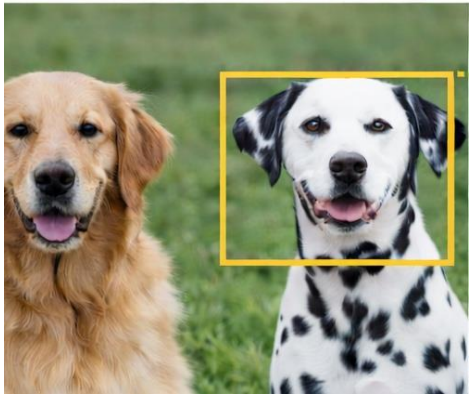
Convolutional building blocks

- Number of **parameters to learn** in a Convolutional layer
  - Let c_{L-1} the number of channels of the previous layer L of a convolutonal layer
  - f the filter height and widht and
  - c the number of filters in the layer.

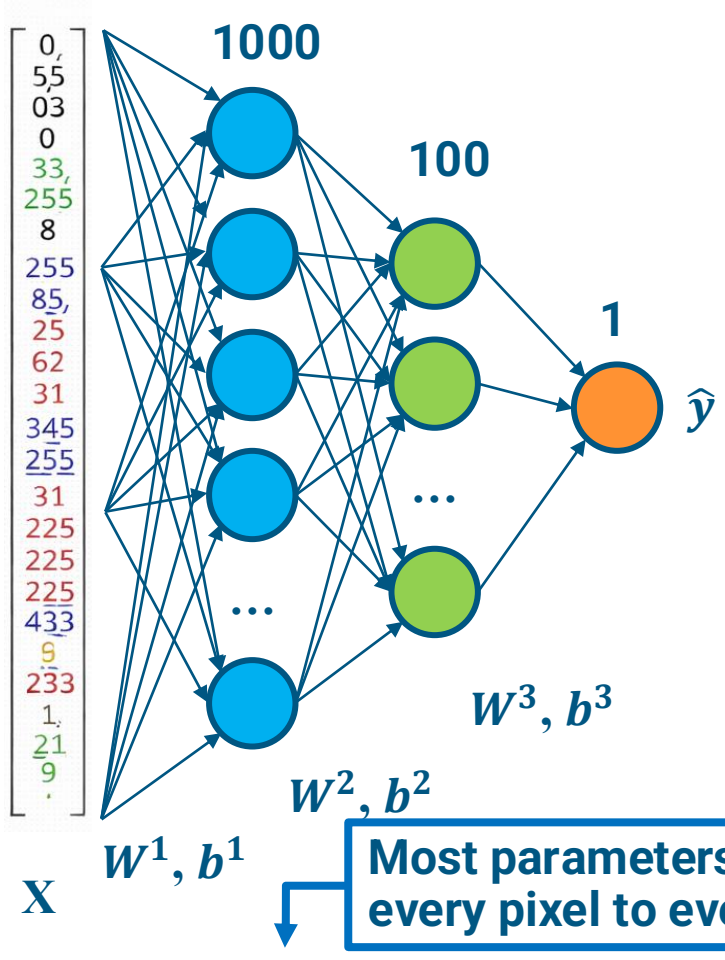
$$\#\ \text{params} = (f \cdot f \cdot c_{L-1} + 1) \cdot c_L$$

Convolutional building blocks

- And now?



1024x1024x3



X → 3M

First layer

$W^1 \rightarrow$ 3M x 1000

$b^1 \rightarrow 1000$

Second layer

$W^2 \rightarrow$ 1000x100

$b^2 \rightarrow 100$

Output layer

$W^3 \rightarrow$ 100x1

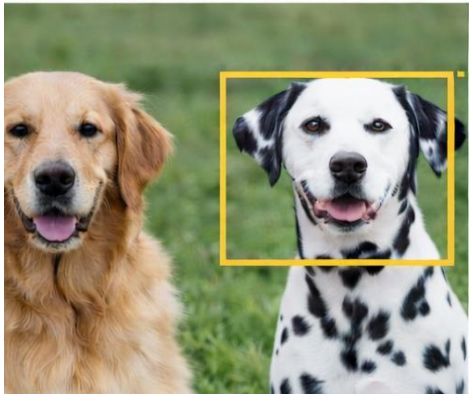
$b^3 \rightarrow 1$

Most parameters are wasted connecting every pixel to every neuron.

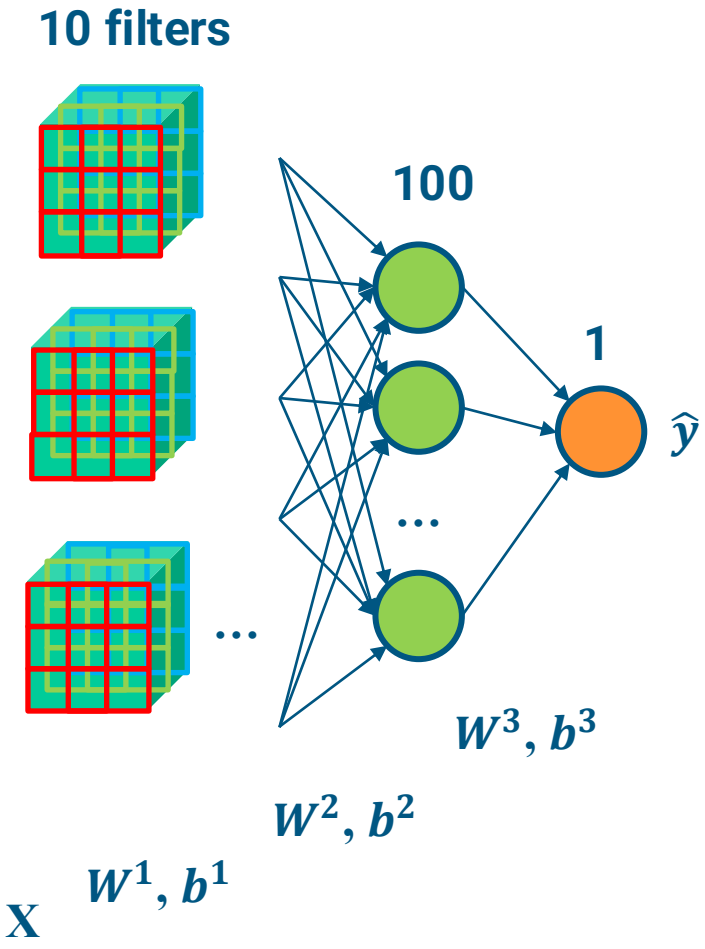
Total = 3.145.729.000 + 100.100 + 101 = **3.145.829.201**

Convolutional building blocks

- And now?



1024x1024x3



X → 3M

First layer

$W^1 \rightarrow 270$

$b^1 \rightarrow 10$

Second layer

$W^2 \rightarrow ....$

$b^2 \rightarrow ...$

Output layer

$W^3 \rightarrow ...$

$b^3 \rightarrow ....$

Difference = 3.145.729.000 vs **280 parameters**

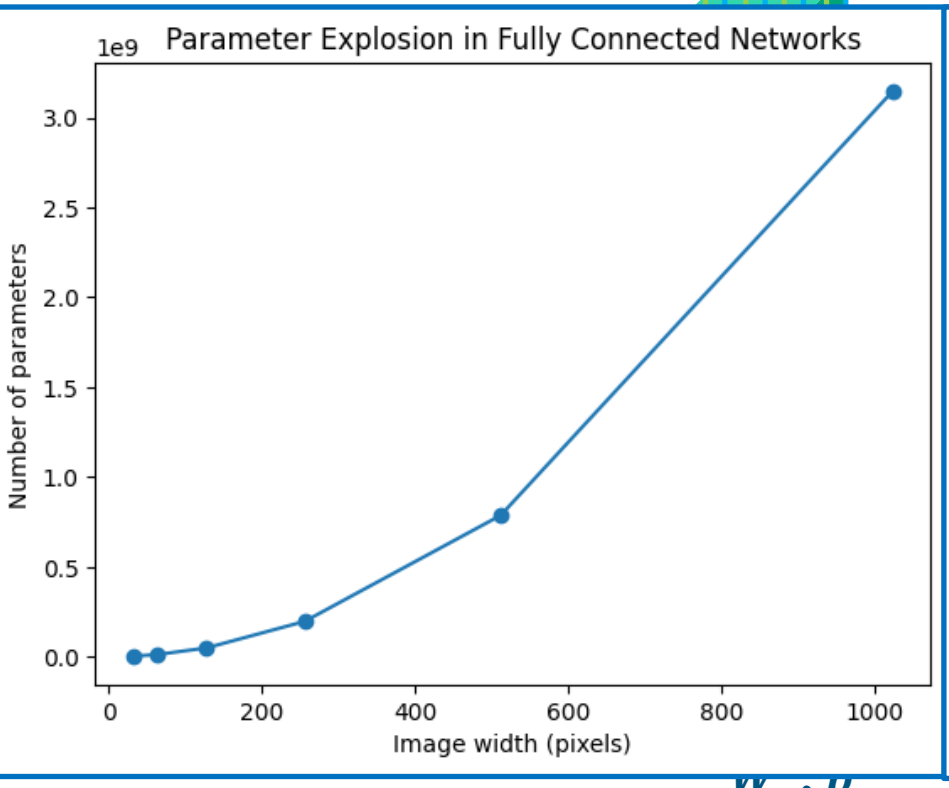
Convolutional building blocks

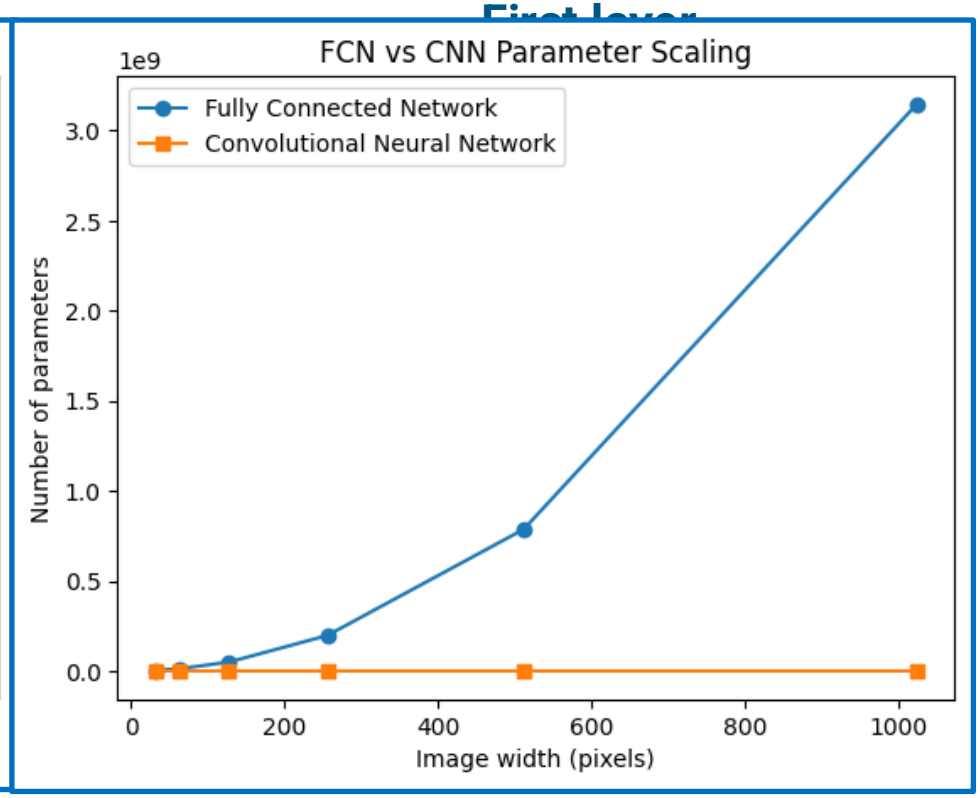
- And now?

10 filters

X → 3M

First layer



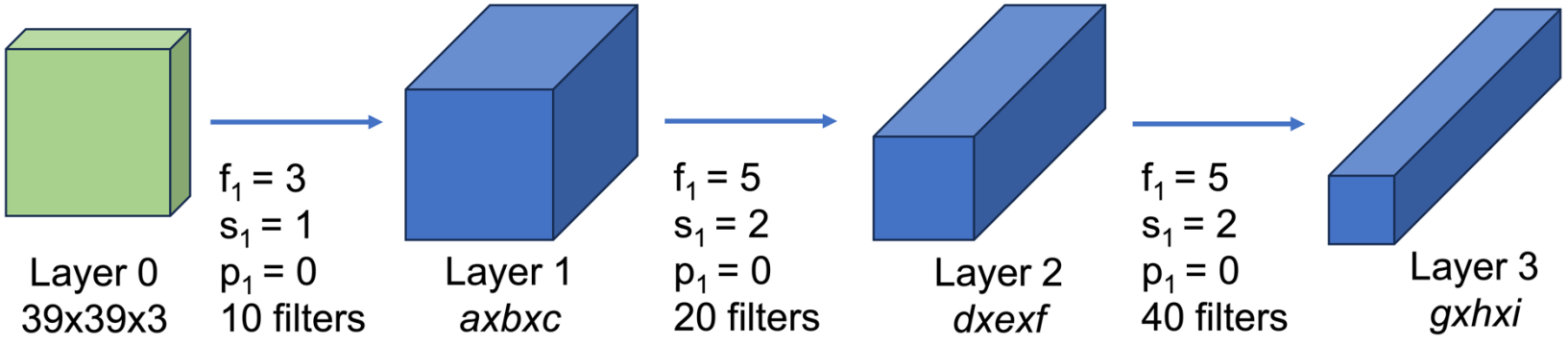


X W, b

Difference = 3.145.729.000 vs 280 parameters

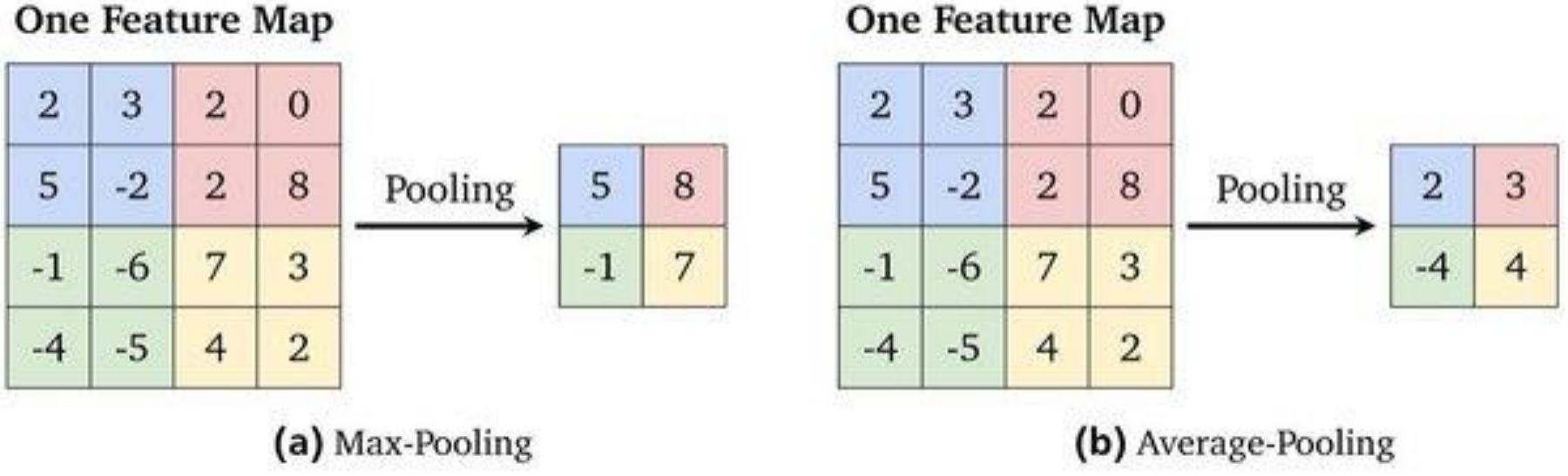
Convolutional building blocks

- **Exercise**
  - Calculate the number of parameters



Convolutional building blocks

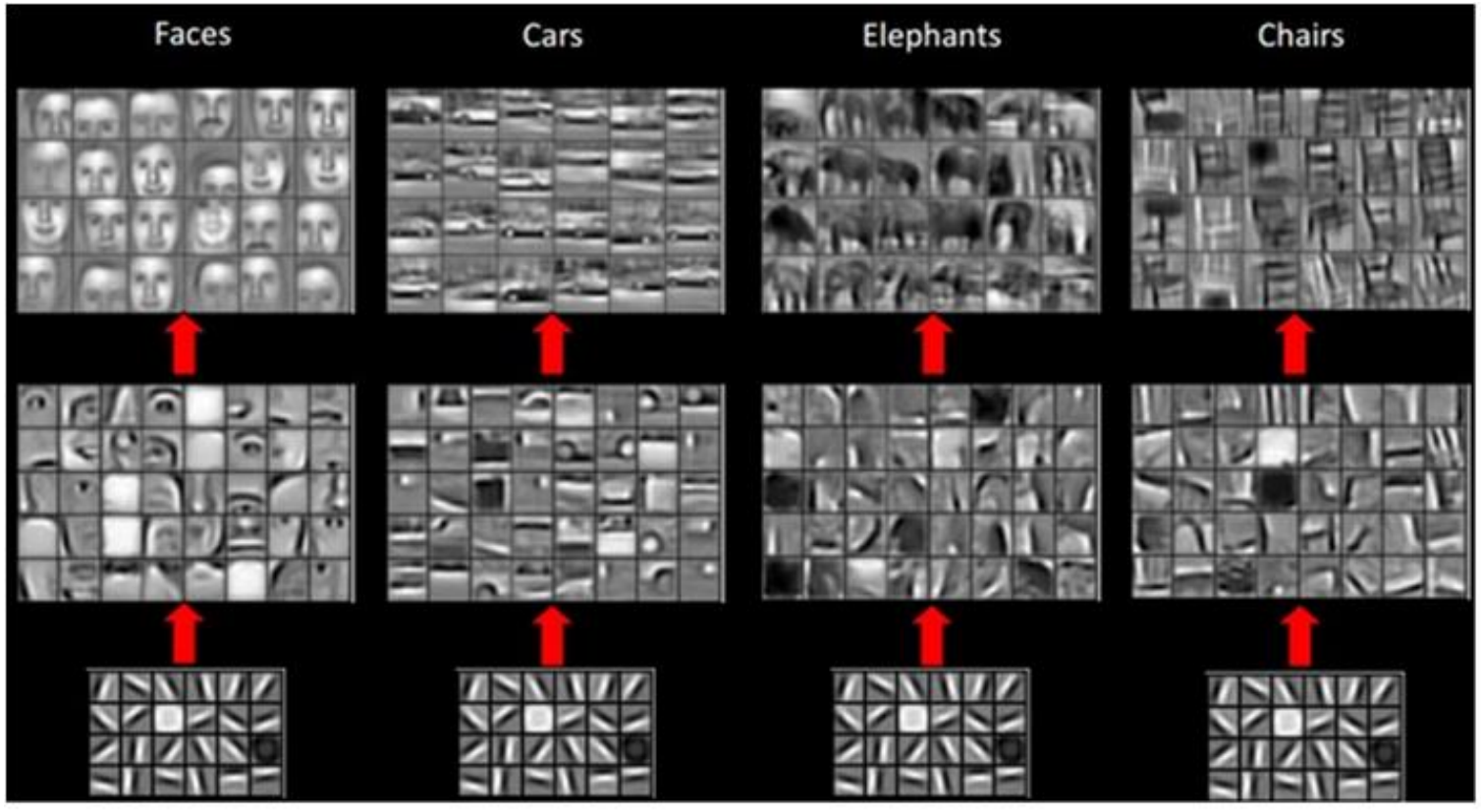
- **Pooling layers**: Reduces spatial dimensions
  - Reduce computation
  - Reduce overfitting
  - Increase receptive field
  - Introduce local translation invariance



No parameters to learn !!

Hierarchical Feature Learning

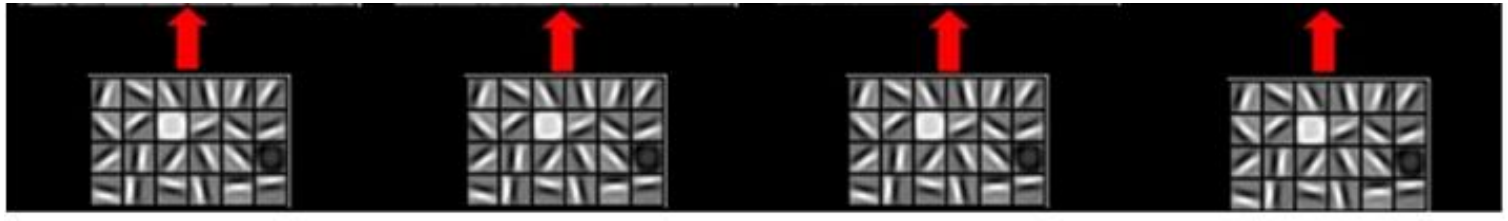
- CNNs learn representations in stages:
  - Early layers → low-level features
  - Middle layers → intermediate patterns
  - Deep layers → high-level concepts



Hierarchical Feature Learning

- CNNs learn representations in stages:
  - **Early layers → low-level features**
  - Middle layers → intermediate patterns
  - Deep layers → high-level concepts

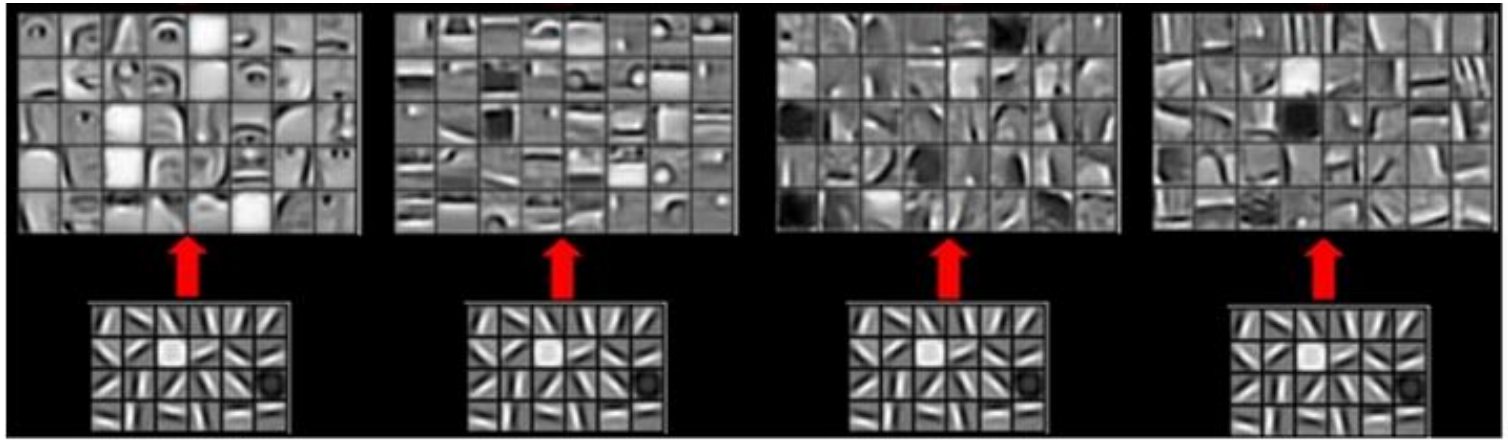
- Typically learn:
  - Edges (vertical, horizontal)
  - Corners
  - Color contrasts
  - Simple textures



Hierarchical Feature Learning

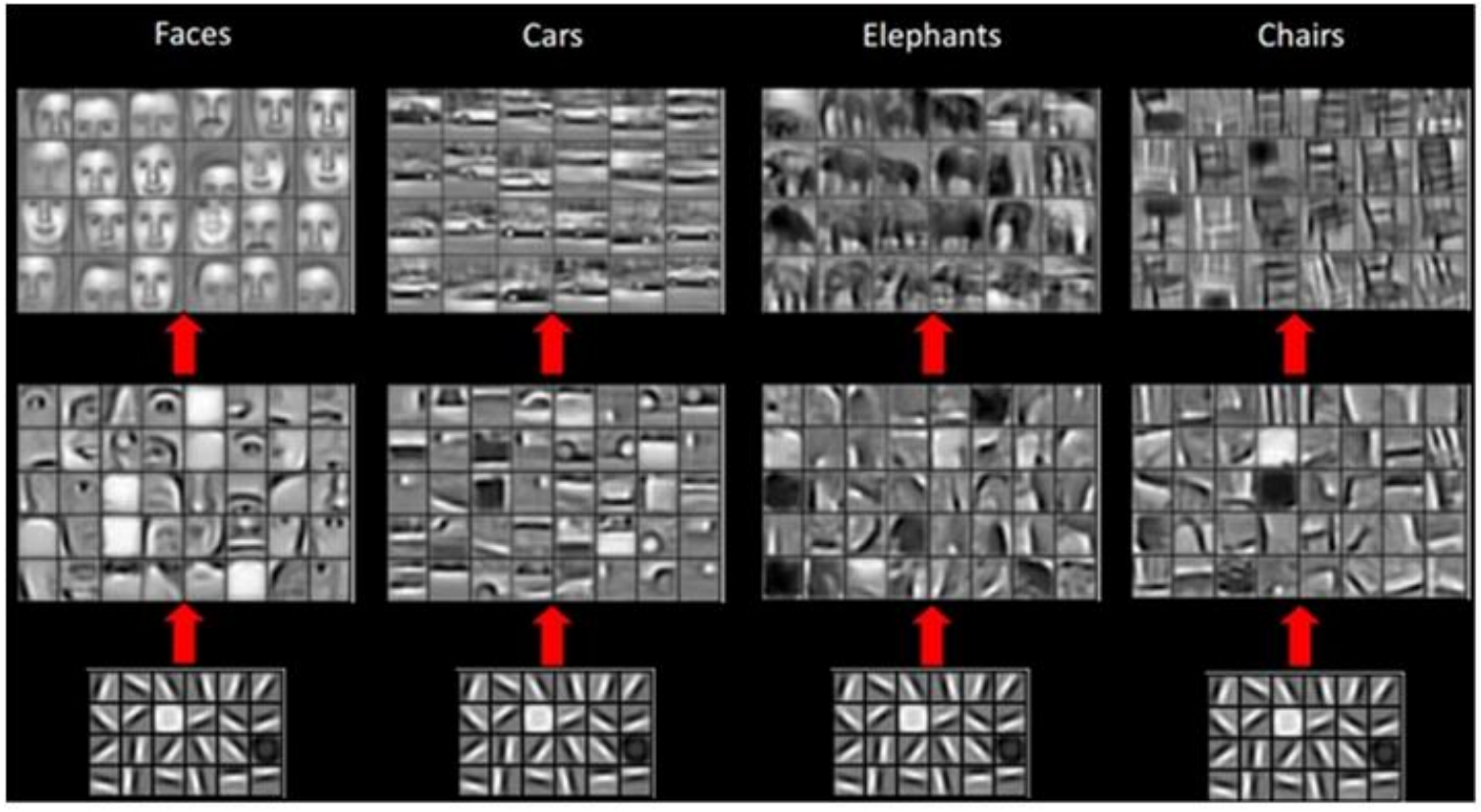
- CNNs learn representations in stages:
  - Early layers → low-level features
  - **Middle layers** → intermediate patterns
  - Deep layers → high-level concepts

- Combine low-level features into:
  - Textures
  - Repeated patterns
  - Parts of objects (eyes, wheels,…).



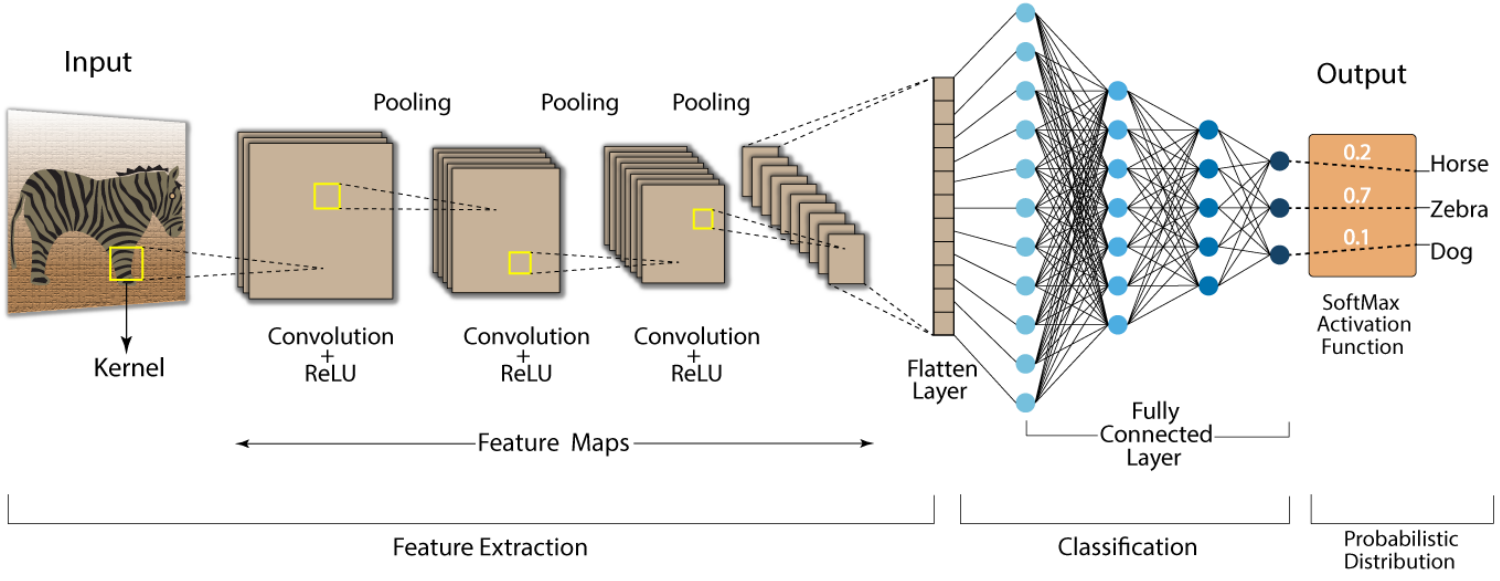
Hierarchical Feature Learning

- CNNs learn representations in stages:
  - Early layers → low-level features
  - Middle layers → intermediate patterns
  - **Deep layers** → high-level concepts



Designing a basic CNN architecture

- A typical CNN has:
  - Several **convolution plus pooling layers**
    - Each responsible for feature extraction at different levels of abstraction
  - A **final classifier**

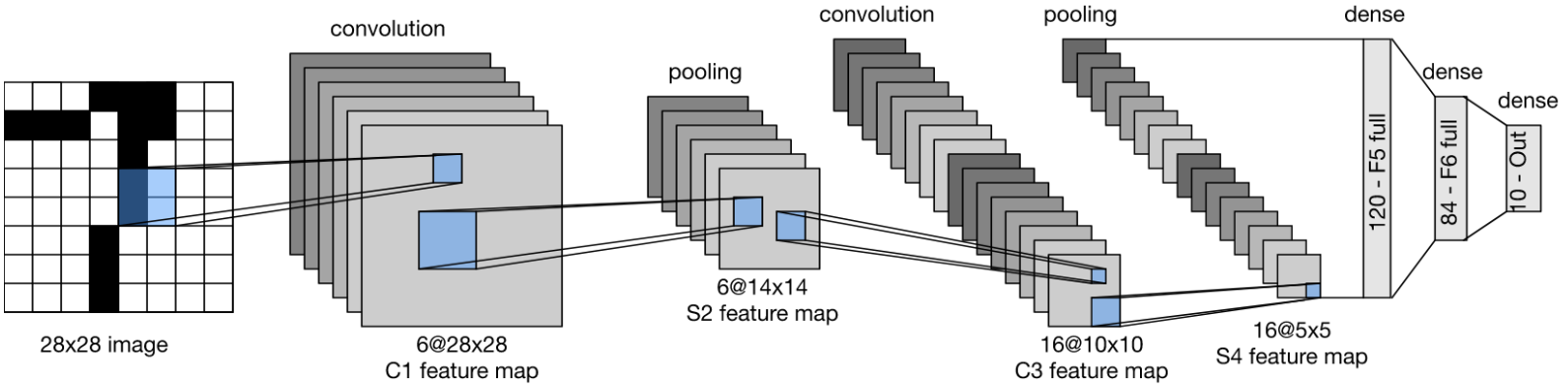


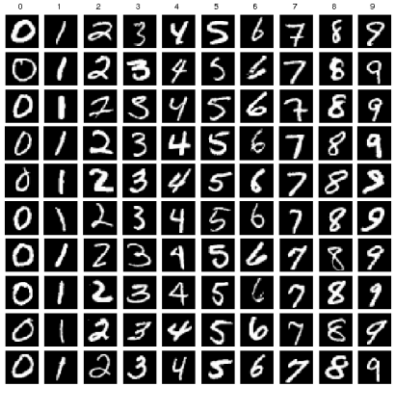
As we go deeper:

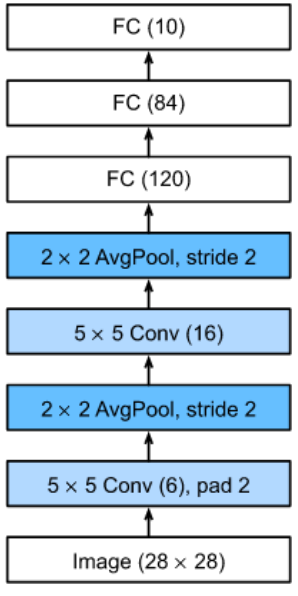
- Size ↓
- Channels ↑
- Spatial resolution ↓
- Semantic richness ↑

Exercise

- Calculate the activation size and the number of parameters of LeNet







Exercise (optional)

- Calculate the Forward pass and Backward pass for the first convolutional layer