UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

Dipartimento
di Ingegneria Gestionale,
dell'Informazione e della Produzione

Lesson 11.

Computer vision - part II

Deep learning approaches

DATA SCIENCE AND AUTOMATION COURSE

MASTER DEGREE SMART TECHNOLOGY ENGINEERING

TEACHER
Mirko Mazzoleni

Michele Ermidoro

PLACE
University of Bergamo

Outline

1. Convolutional neural networks
2. Object detection
3. Transfer learning
4. Hardware
5. Application to pneumonia detection using X-ray images

Outline

1. **Convolutional neural networks**

2. Object detection

3. Transfer learning

4. Hardware

5. Application to pneumonia detection using X-ray images

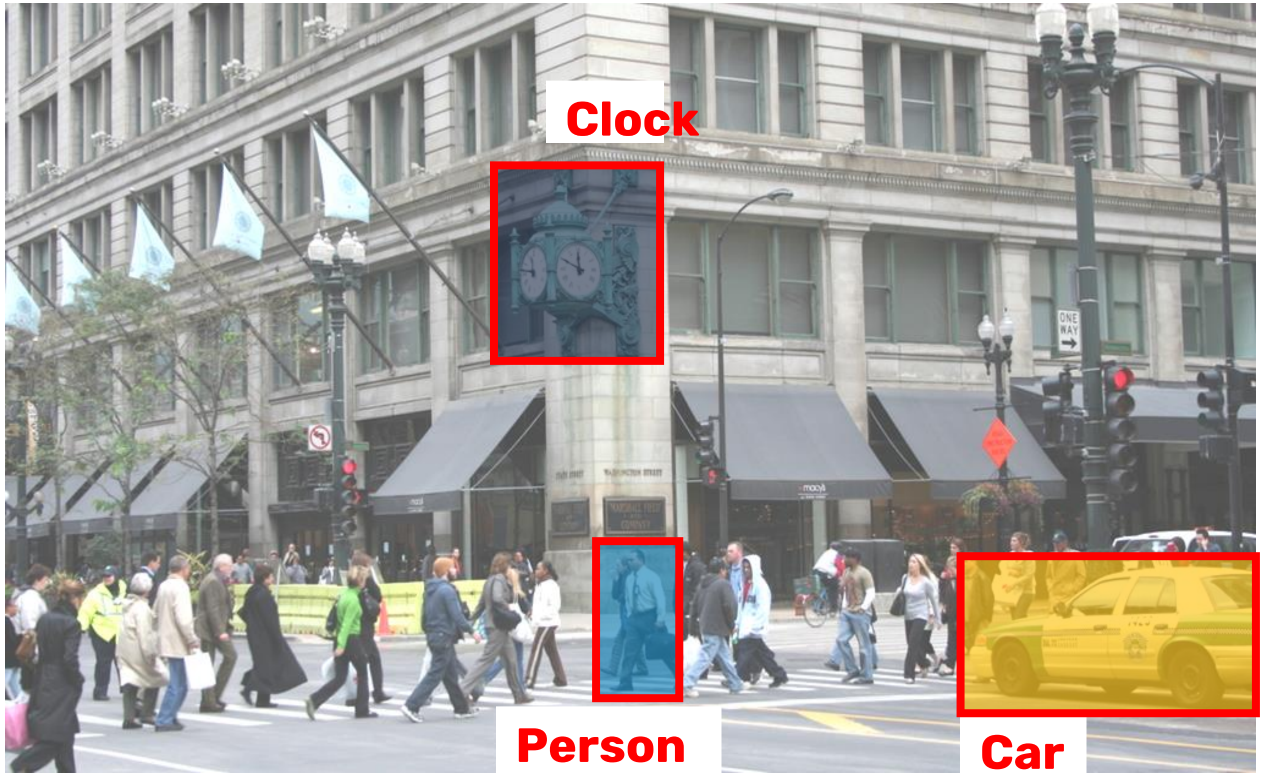
Computer vision tasks: reminder

Classification

What's in the image?

Detection

What's in the image? **Where** it is?



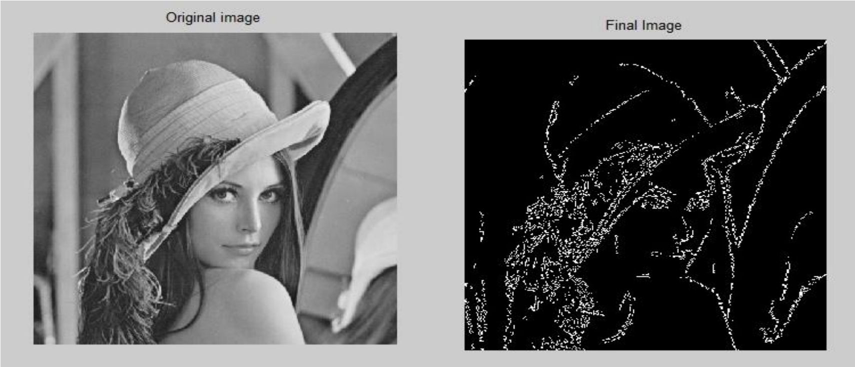
Convolutional neural networks

A **convolutional neural network** (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing images

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge detection filter

We have seen how **convolving** the image with **filters** (or kernels) is an effective method to extract useful information about the image (edges, corners, …) than can be used as **features** for training a classifier

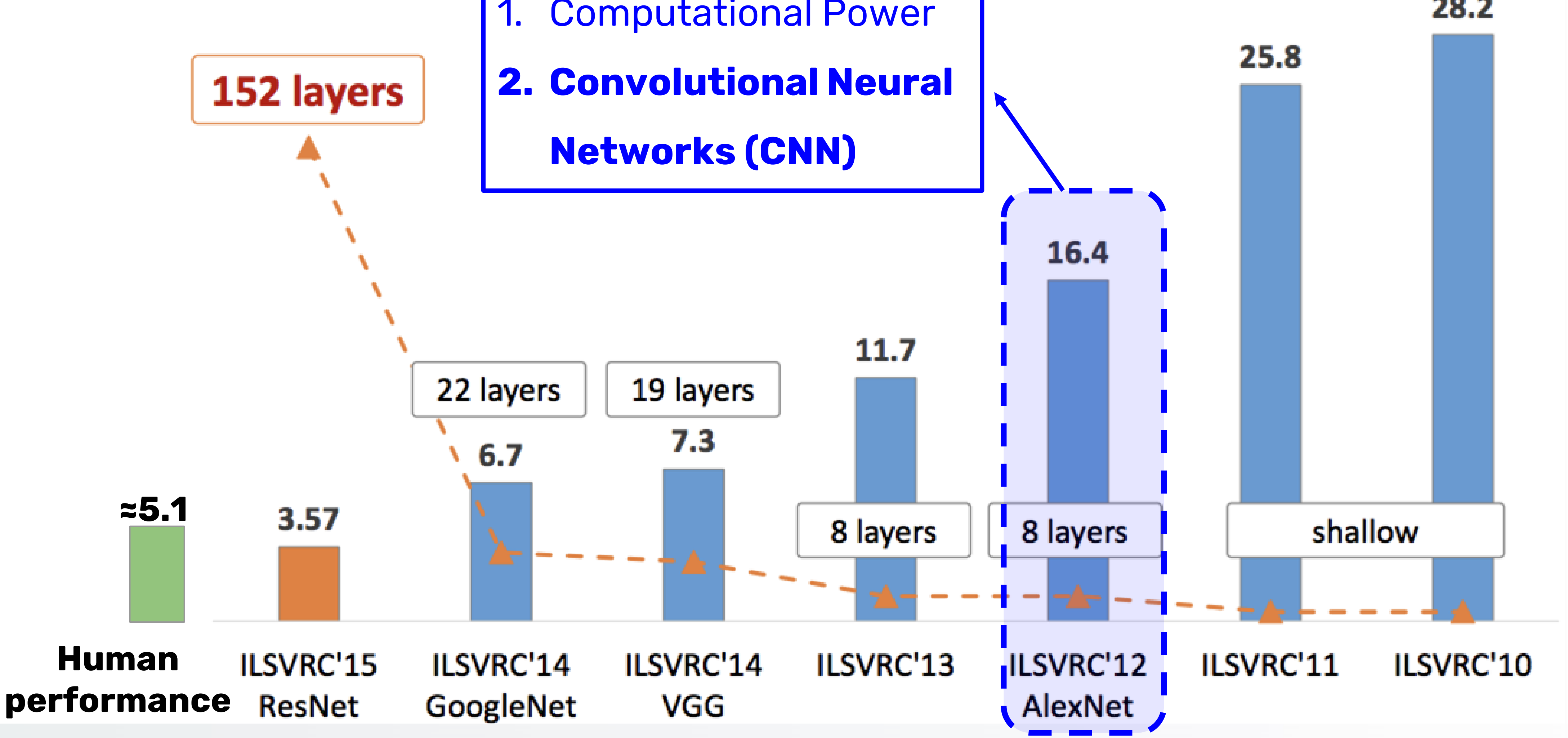


The main idea behind CNN is to **learn the filters**, instead of manually specifying them

- Each element in a filter is a number, and can be treated as a **parameter** to be learnt

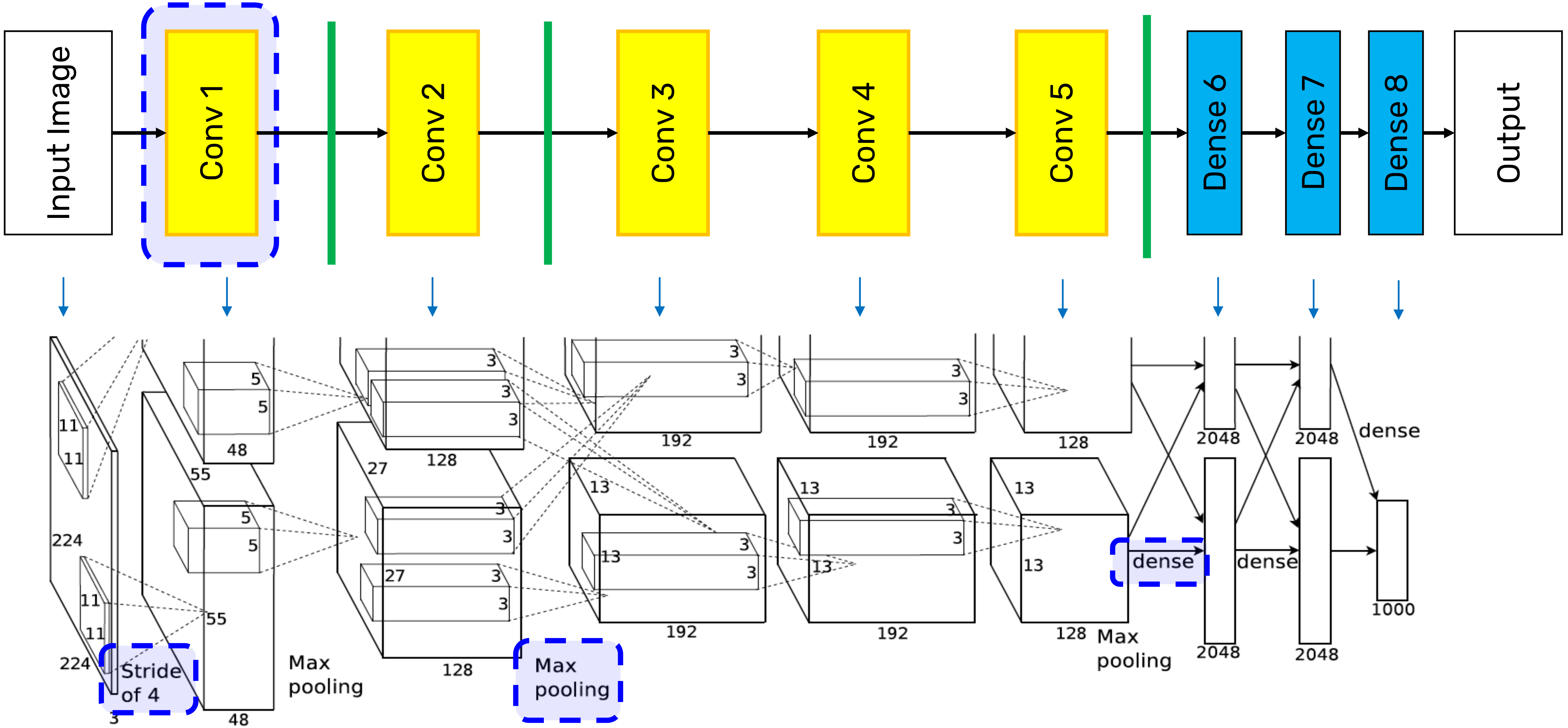
Convolutional neural networks

1. Computational Power
2. **Convolutional Neural Networks (CNN)**



ImageNet Classification top-5 error (%)

Alex-net structure



Convolution (recap)

Convolution is the process of adding each element of the image to its local neighbors, weighted by the kernel.

$$y[2,2] = \sum_{j} \sum_{i} x[i,j] \cdot k[2-i, 2-j]$$

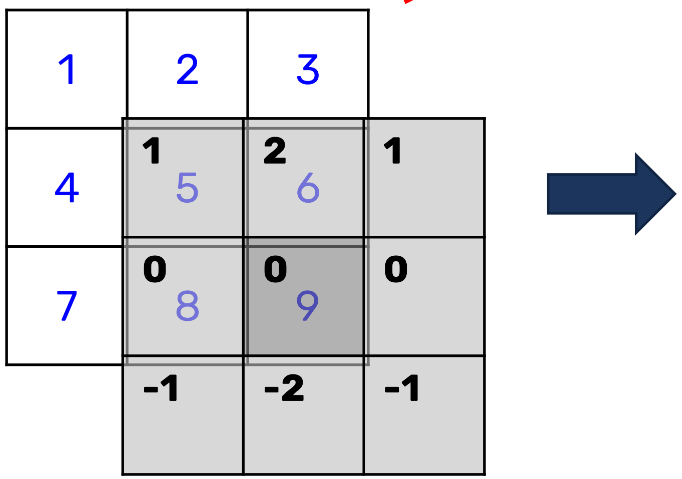
Input image $x[\cdot,\cdot]$

1	2	3
4	5	6
7	8	9

Kernel $k[\cdot,\cdot]$

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

STEP 9



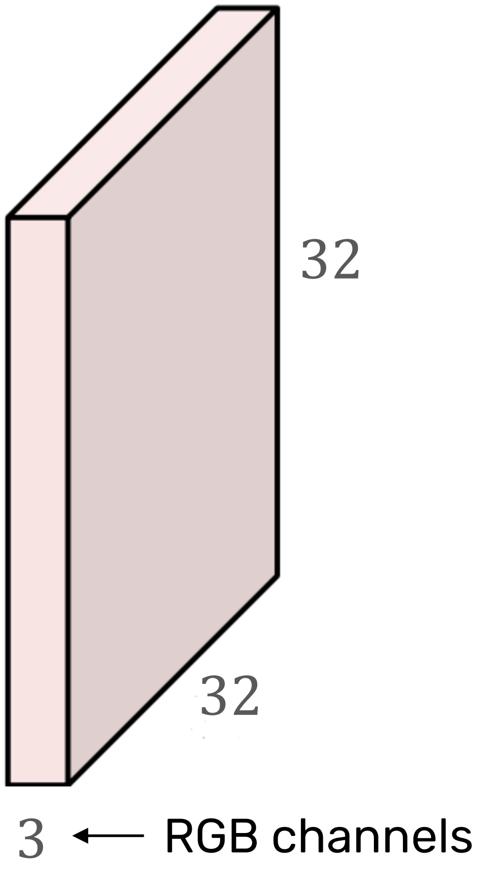
Output $y[\cdot,\cdot]$

-13	-20	-17
-18	-24	-18
13	20	**17**

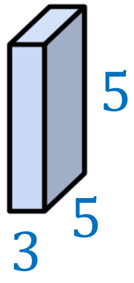
$$5\cdot\mathbf{1} + 6\cdot\mathbf{2} + 0\cdot\mathbf{1} + 8\cdot\mathbf{0} + 9\cdot\mathbf{0} + 0\cdot\mathbf{0} + 0\cdot(\mathbf{-1}) + 0\cdot(\mathbf{-2}) + 0\cdot(\mathbf{-1})$$

Convolutional layer

Input image: $32 \times 32 \times 3$
(height, width, depth)

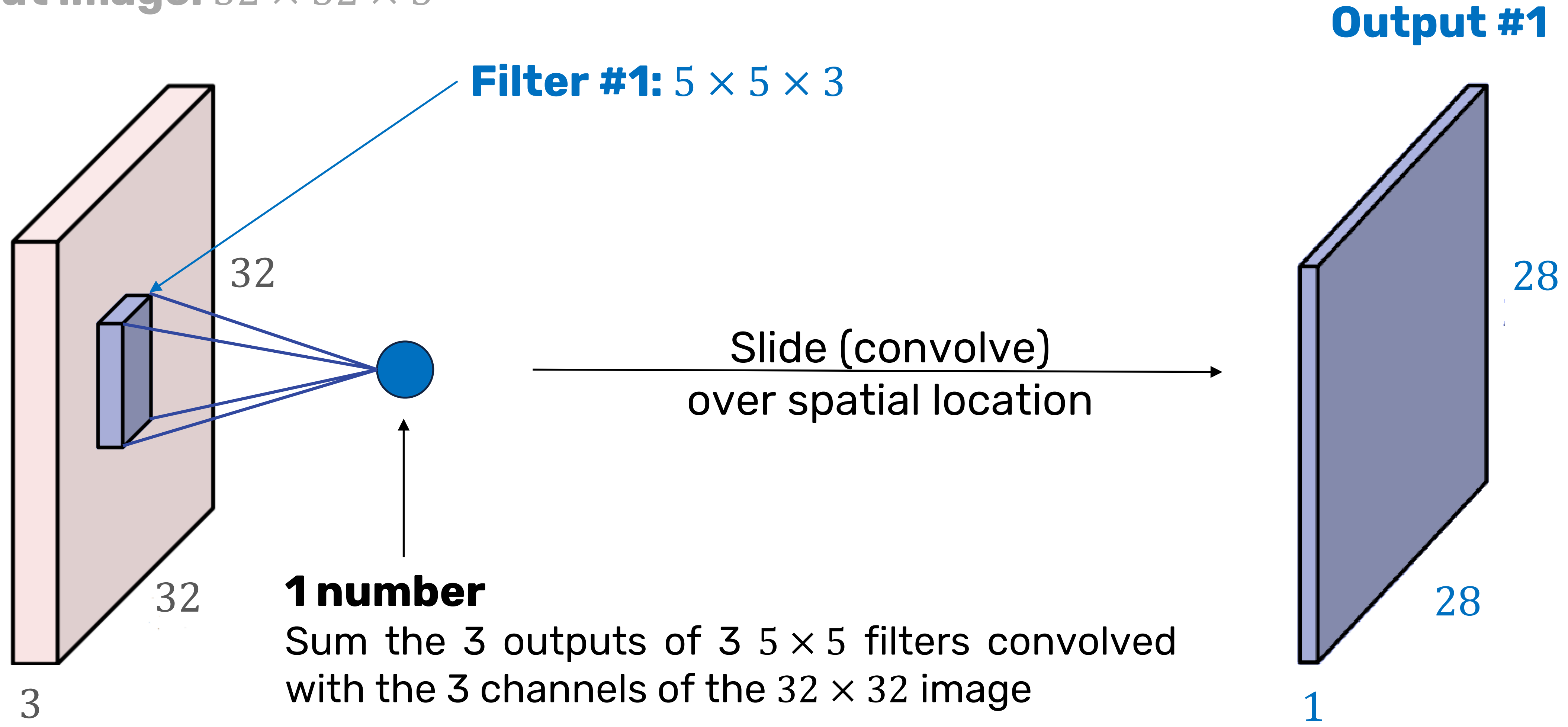


Filter: $5 \times 5 \times 3$



- **Convolve** the filter with the image (slide over the image spatially)
- The filter should have the **same depth** of the previous layer (in this case 3)
- Convolution preserves **spatial structure** (apply the filters on spatially-neighboring «pixels» across all axes)

Convolutional layer

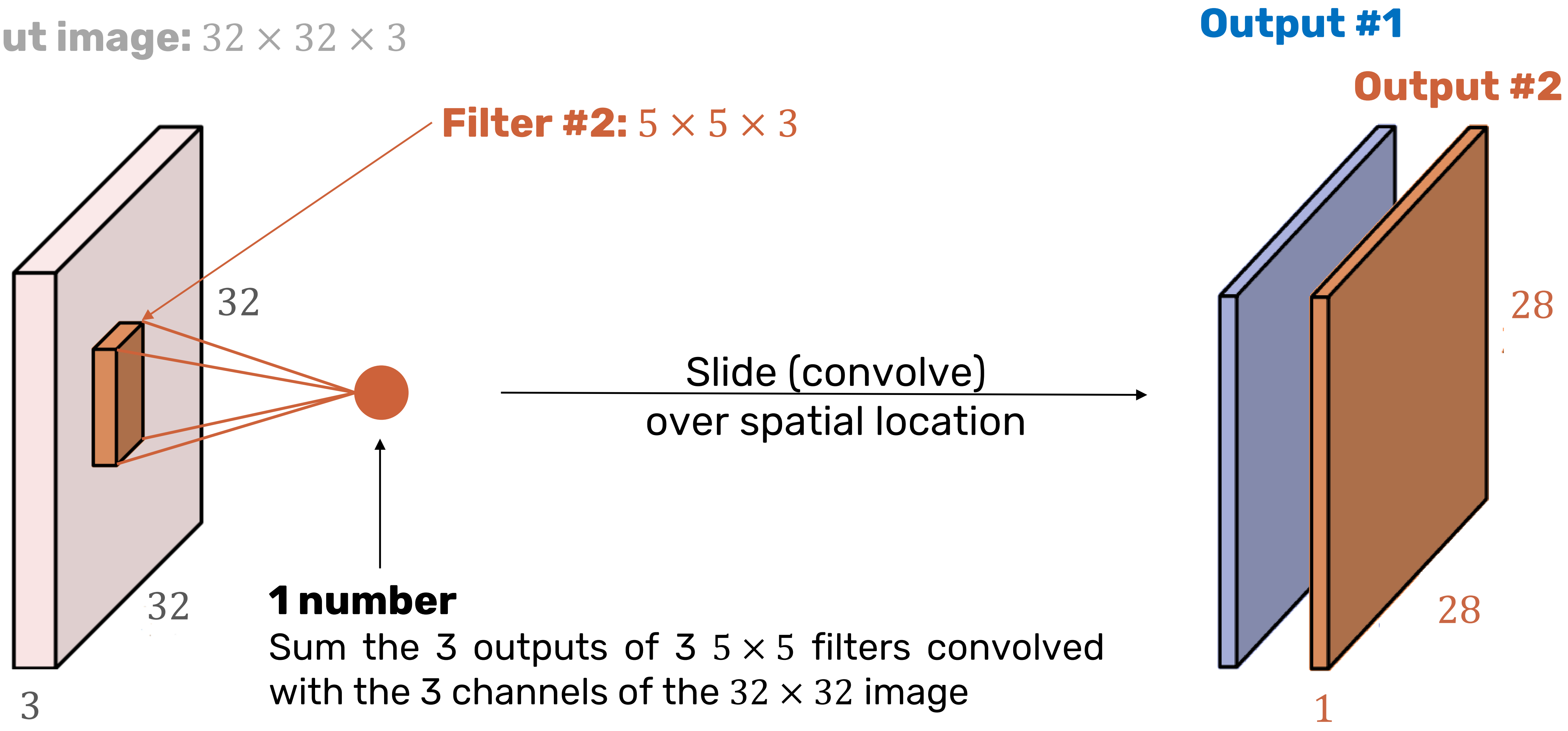

Input image: $32 \times 32 \times 3$
Filter #1: $5 \times 5 \times 3$
Output #1
32
32
3
Slide (convolve)
over spatial location
28
28
1

1 number

Sum the 3 outputs of 3 5×5 filters convolved with the 3 channels of the 32×32 image

Convolutional layer

Input image: $32 \times 32 \times 3$



Filter #2: $5 \times 5 \times 3$

Output #1

Output #2

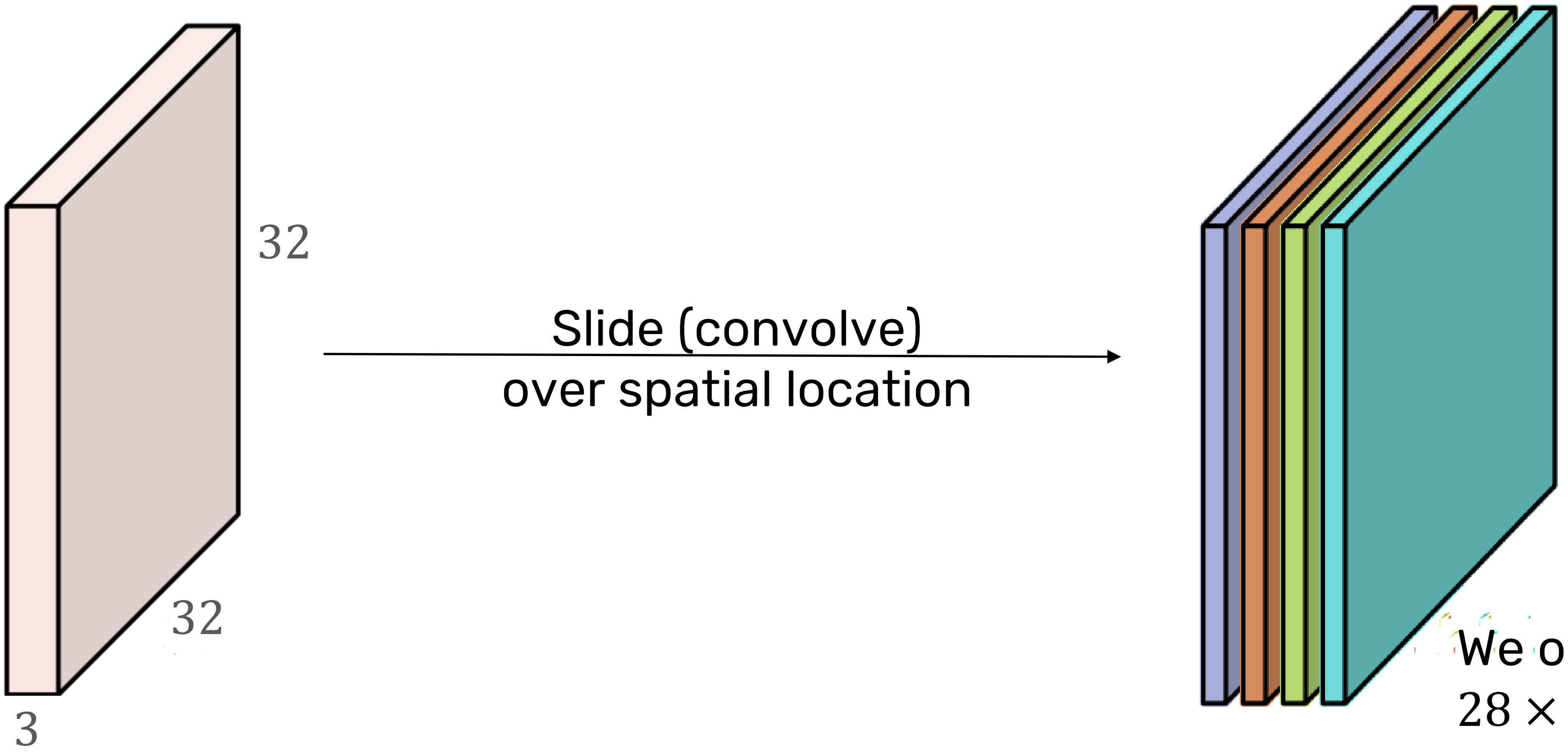
Slide (convolve)
over spatial location

1 number

Sum the 3 outputs of 3 5×5 filters convolved with the 3 channels of the 32×32 image

Convolutional layer

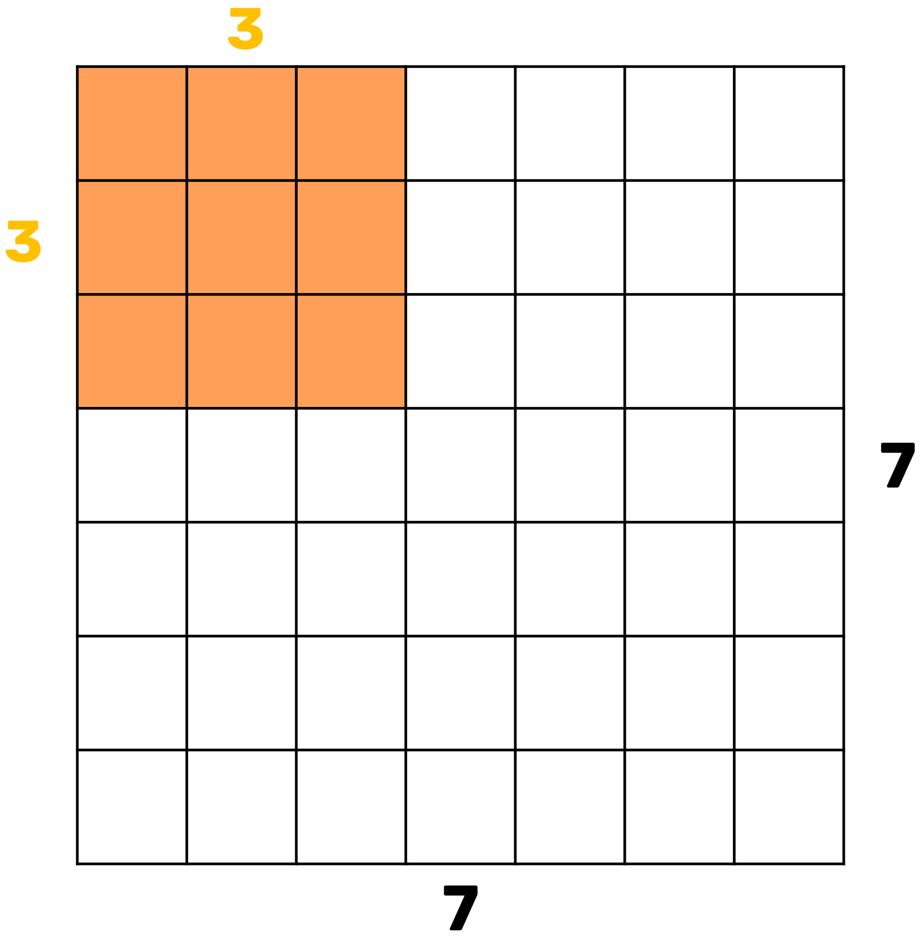
If we have a **4 filters**, we will have **4 outputs**



We obtain a new
$28 \times 28 \times 4$ image

Convolutional layer

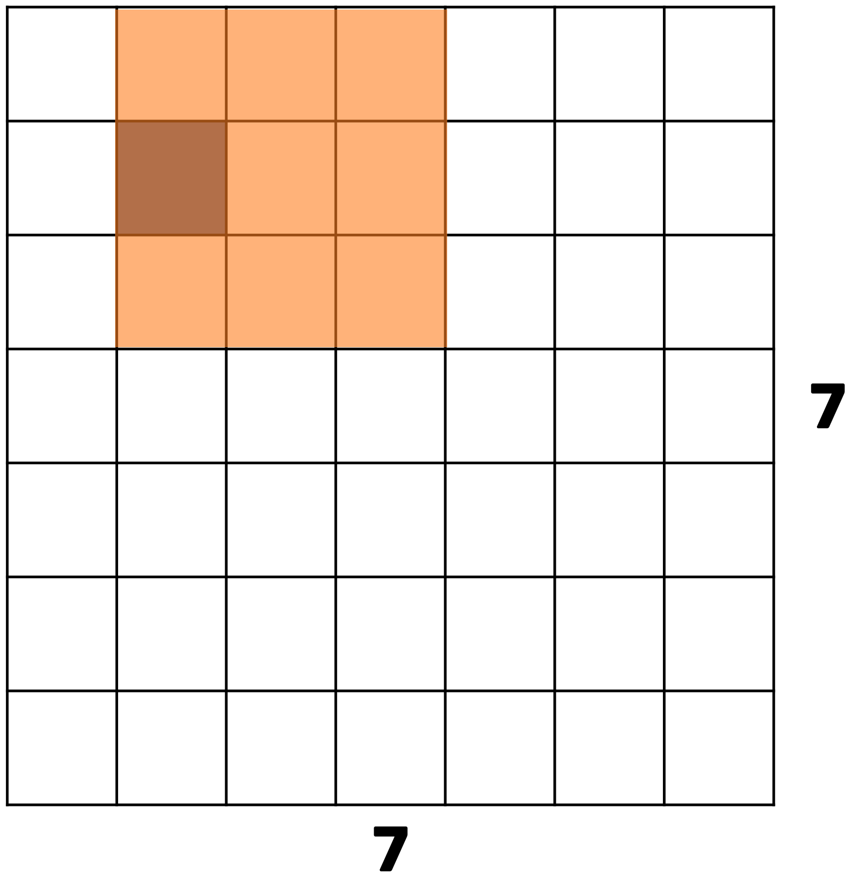
In the previous step convolution step, we reduced the dimension from 32 to 28



- 7×7 **image**
- 3×3 **filter**

Convolutional layer

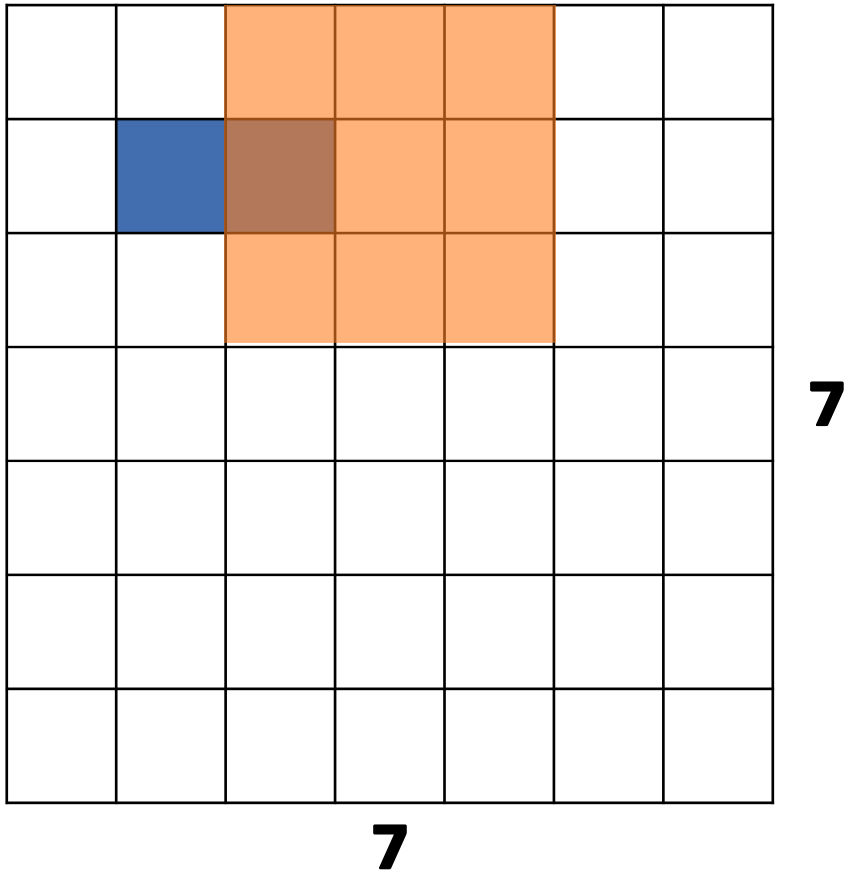
In the previous step convolution step, we reduced the dimension from 32 to 28



- 7×7 **image**
- 3×3 **filter**

Convolutional layer

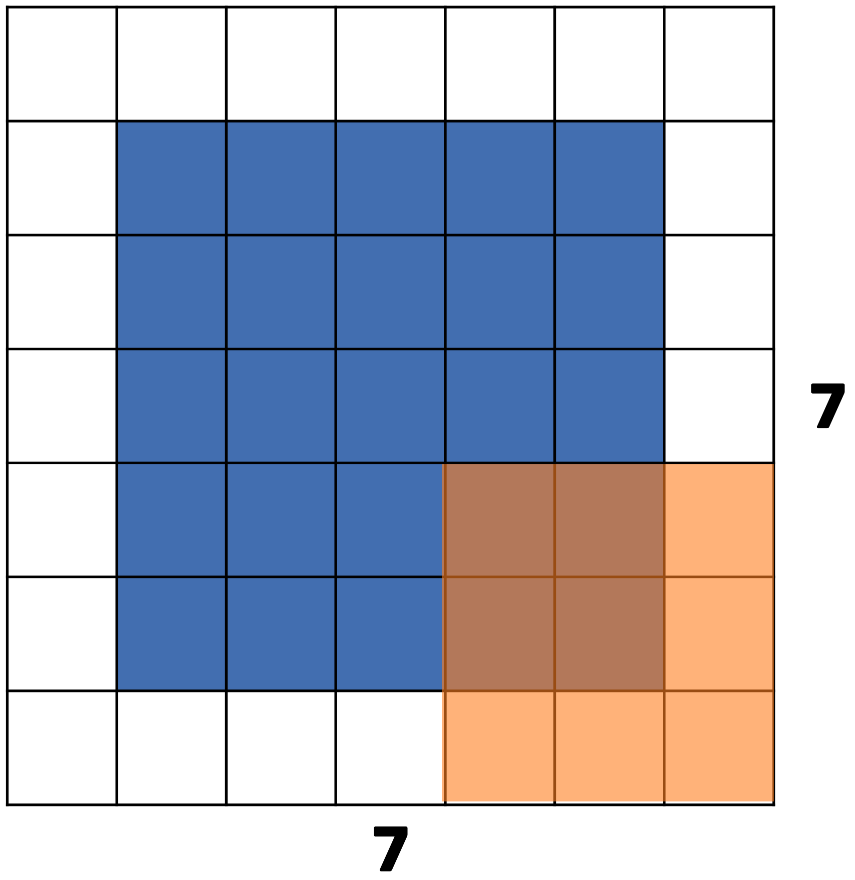
In the previous step convolution step, we reduced the dimension from 32 to 28



- **7×7 image**
- **3×3 filter**

Convolutional layer

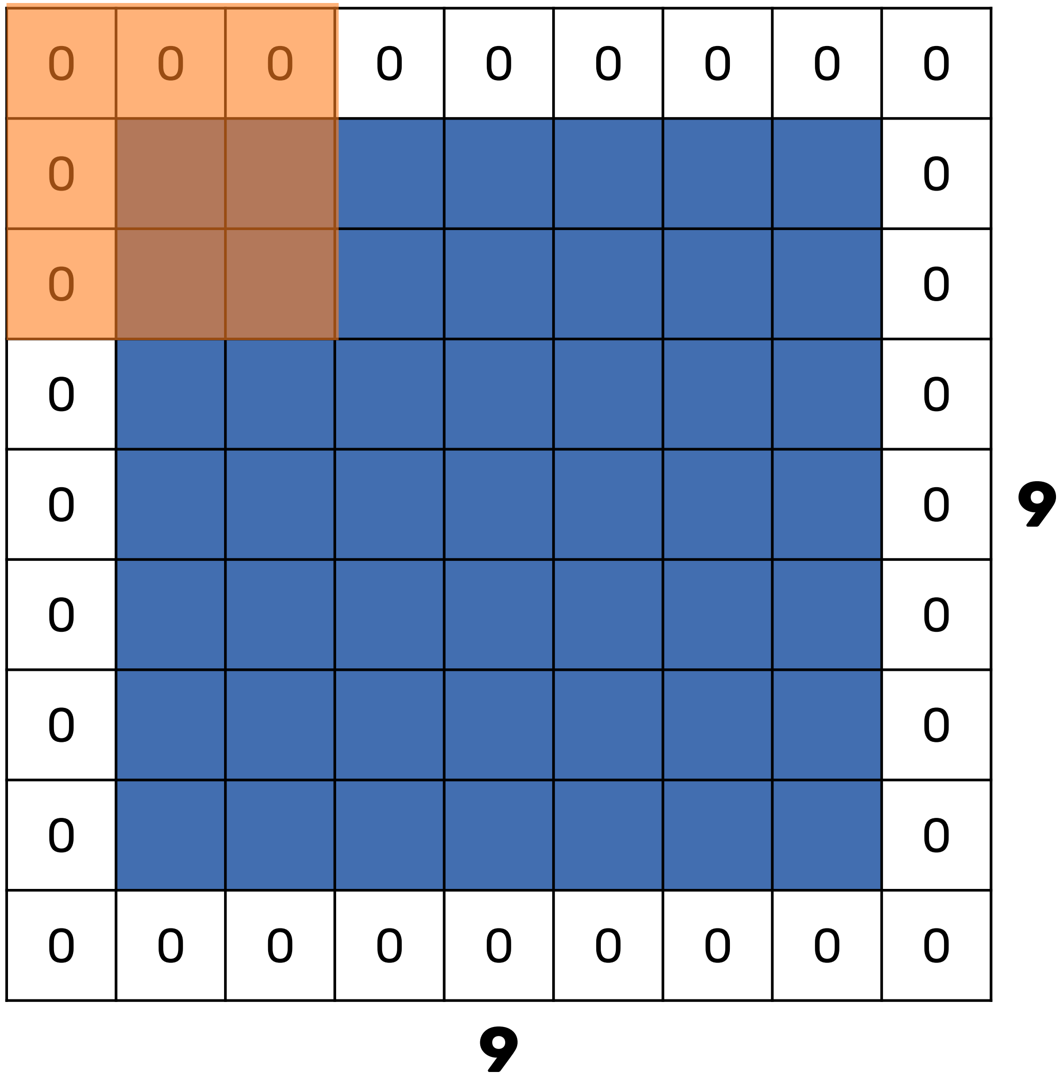
In the previous step convolution step, we reduced the dimension from 32 to 28



- 7×7 **image**
- 3×3 **filter**

We obtained a 5×5 **output**

Convolutional layer



- 9×9 **image** (7×7 image + 1 pixel padding)
- 3×3 **filter**

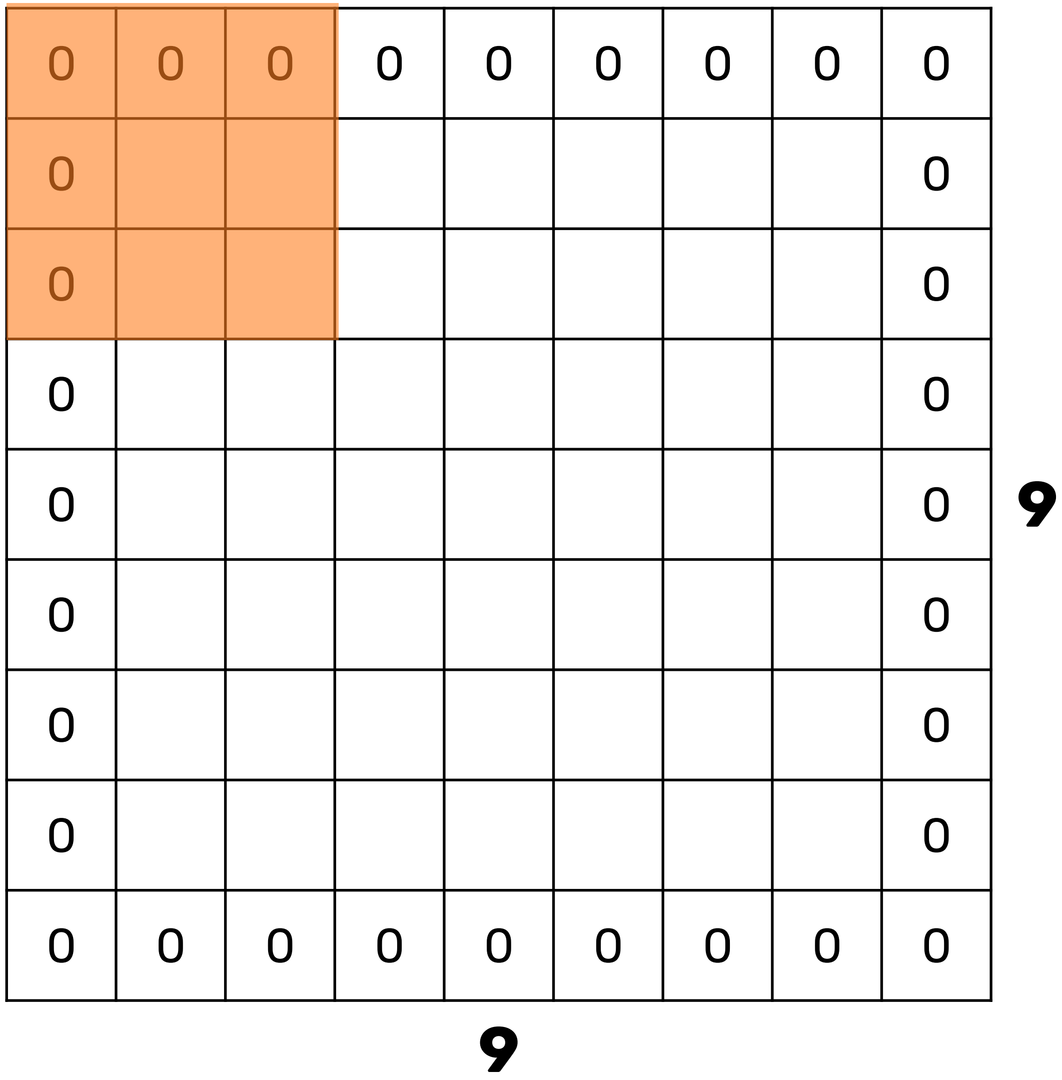
→ We obtained a 7×7 **output**

- In order to keep the output to the **same dimension** of the input, we can add a **frame** around the input image (the size depends on the filter)

↓

Padding

Convolutional layer



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

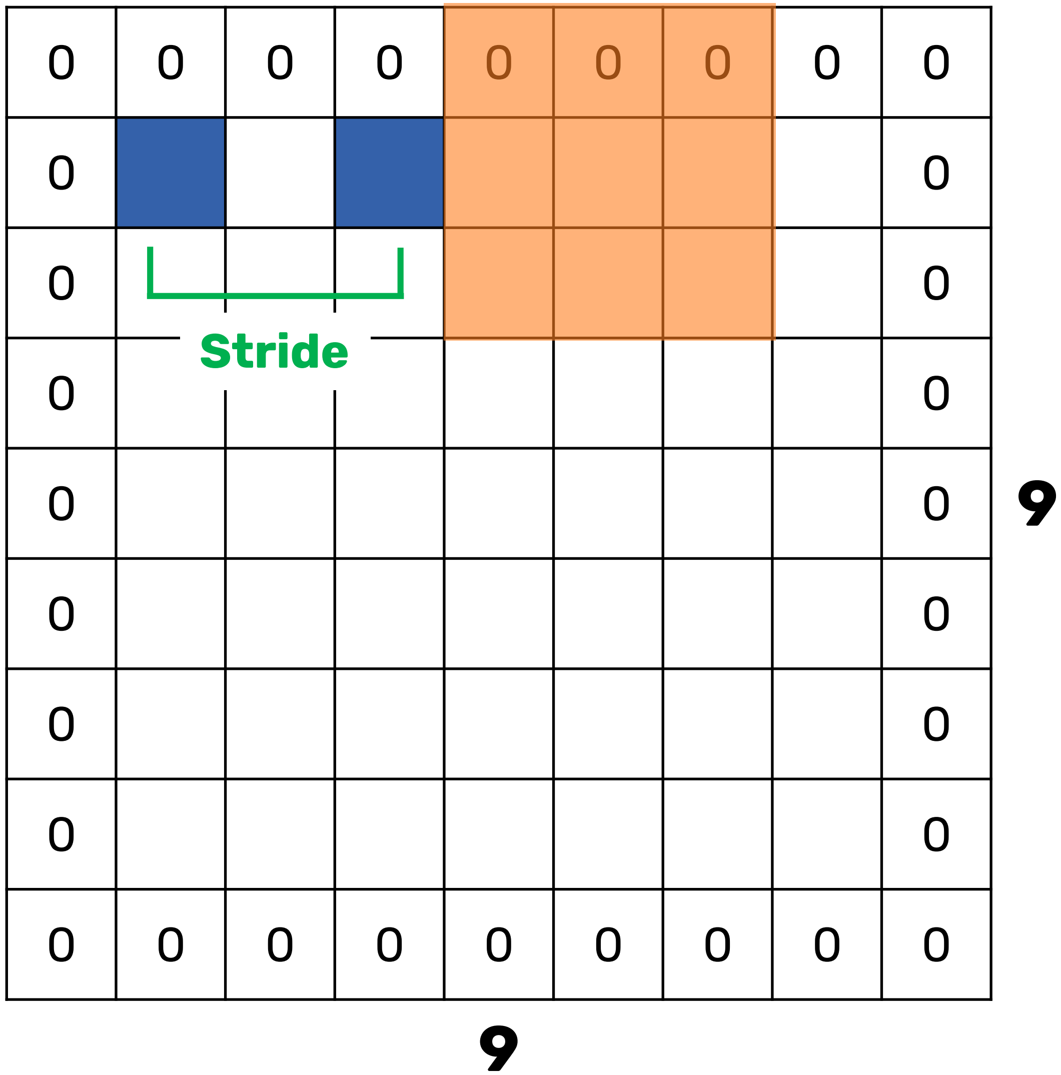
- 9×9 **image** (7×7 image + 1 pixel padding)
- 3×3 **filter**
- **Stride: 2**

- If we want to **reduce** the size of the output, we can use the **stride** parameter

⬇

Stride

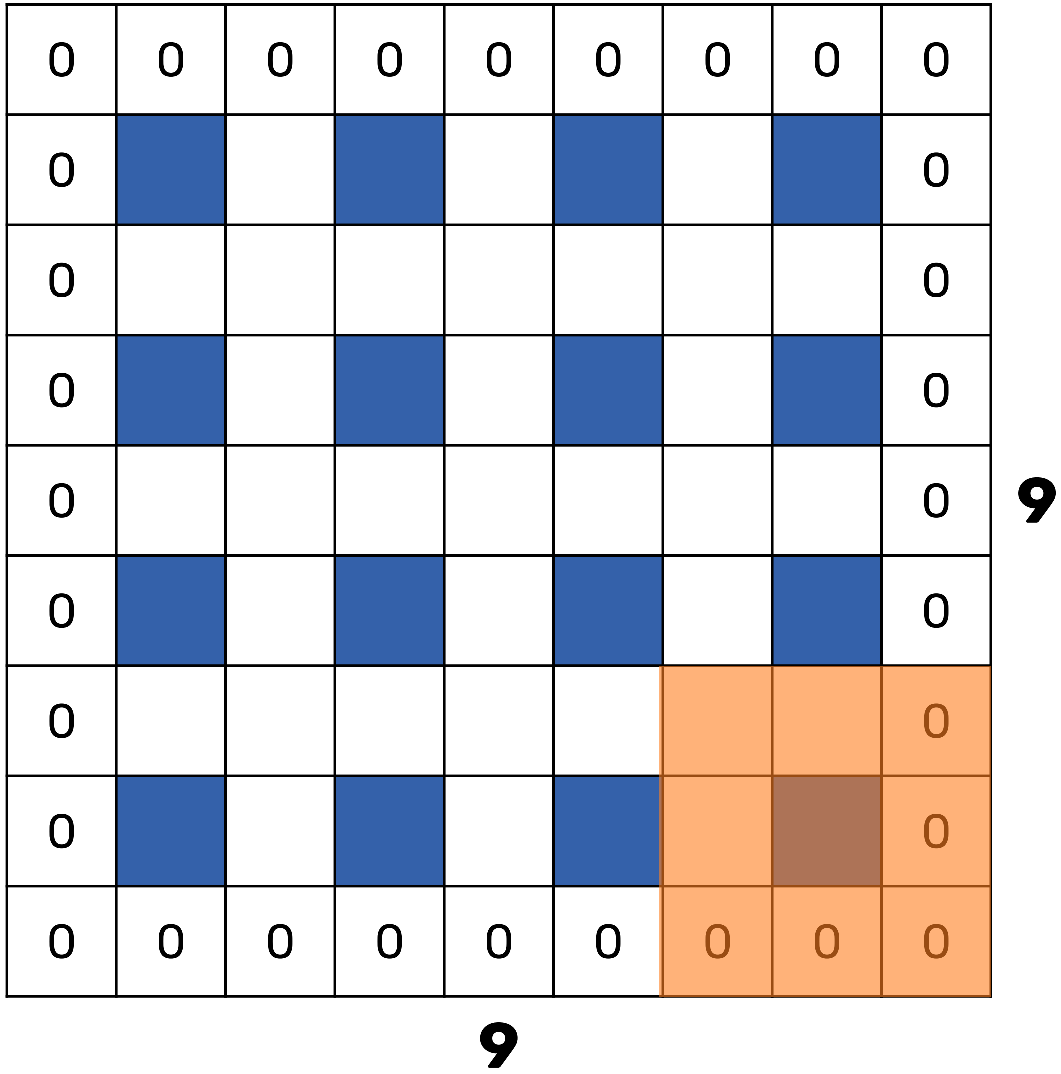
Convolutional layer



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

- 9×9 **image** (7×7 image + 1 pixel padding)
- 3×3 **filter**
- **Stride: 2**

Convolutional layer



0	0	0	0	0	0	0	0	0
0	■		■		■		■	0
0								0
0	■		■		■		■	0
0								0
0	■		■		■		■	0
0								0
0	■		■		■		■	0
0	0	0	0	0	0	0	0	0

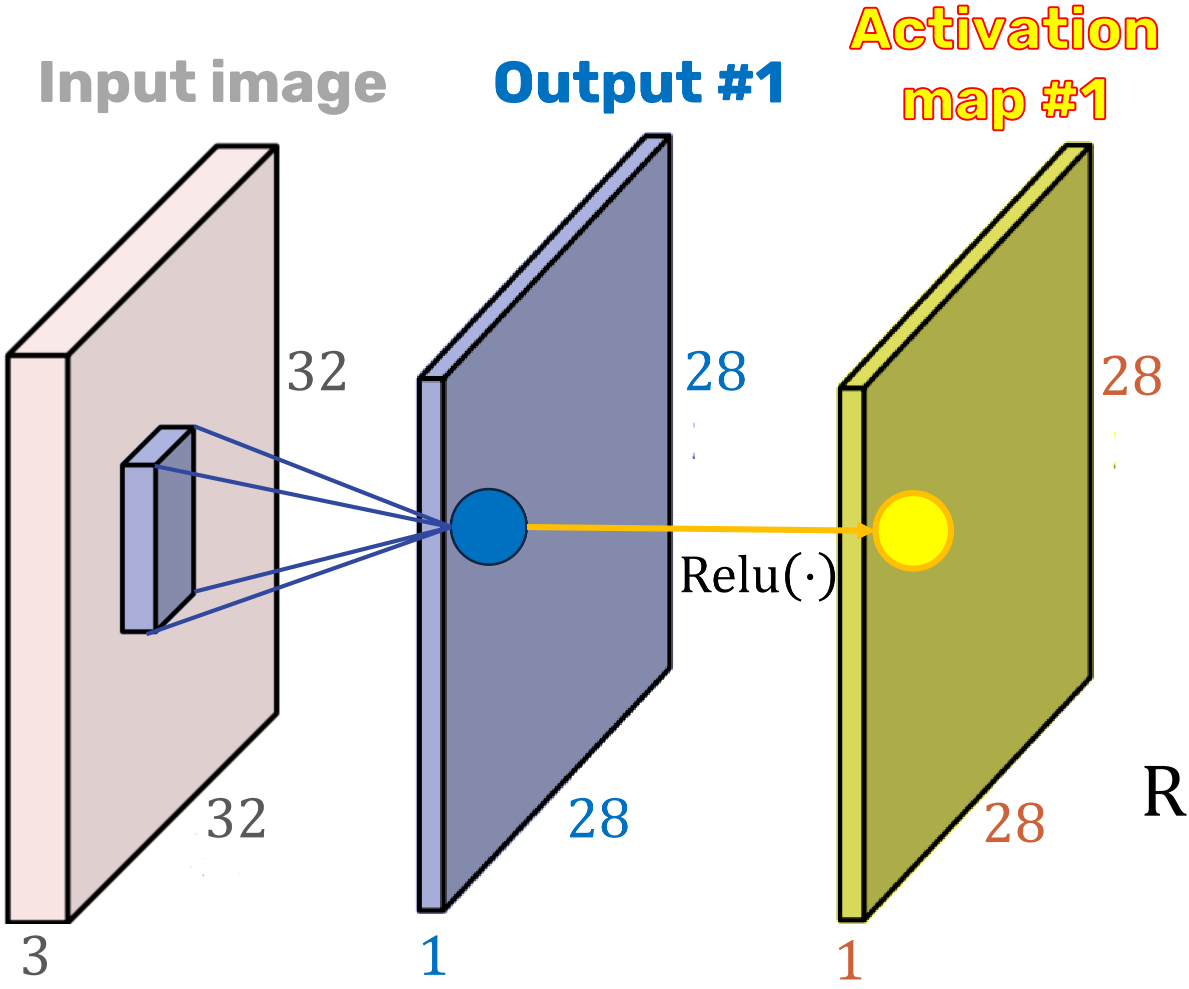
- 9×9 **image** (7×7 image + 1 pixel padding)
- 3×3 **filter**
- **Stride: 2**

→ We obtained a 4×4 **output**

$$\text{Output} = \frac{N - F}{S} + 1$$

- N: dimension of image [px]
- F: dimension of filter [px]
- S: stride length [px]

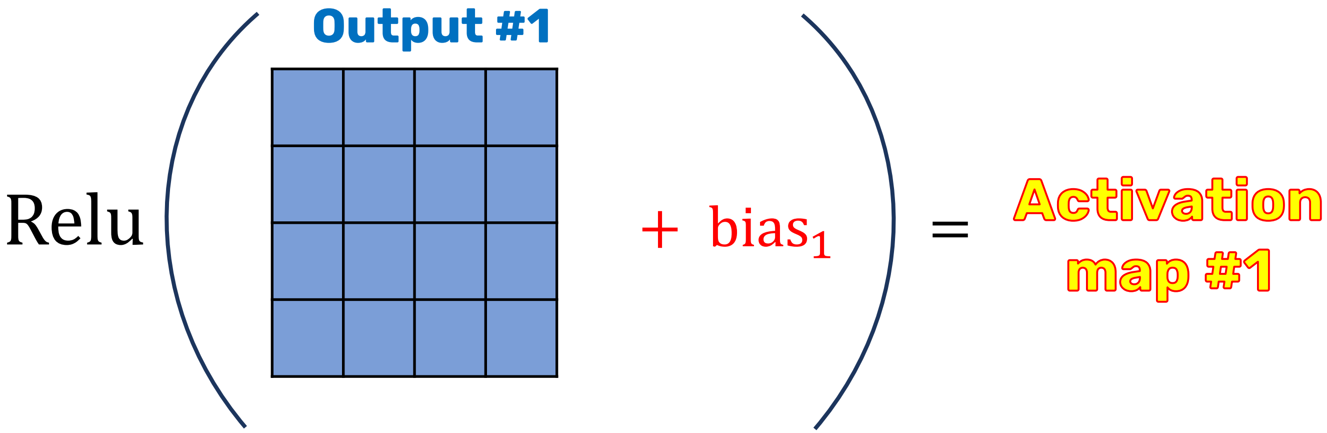
Convolutional layer



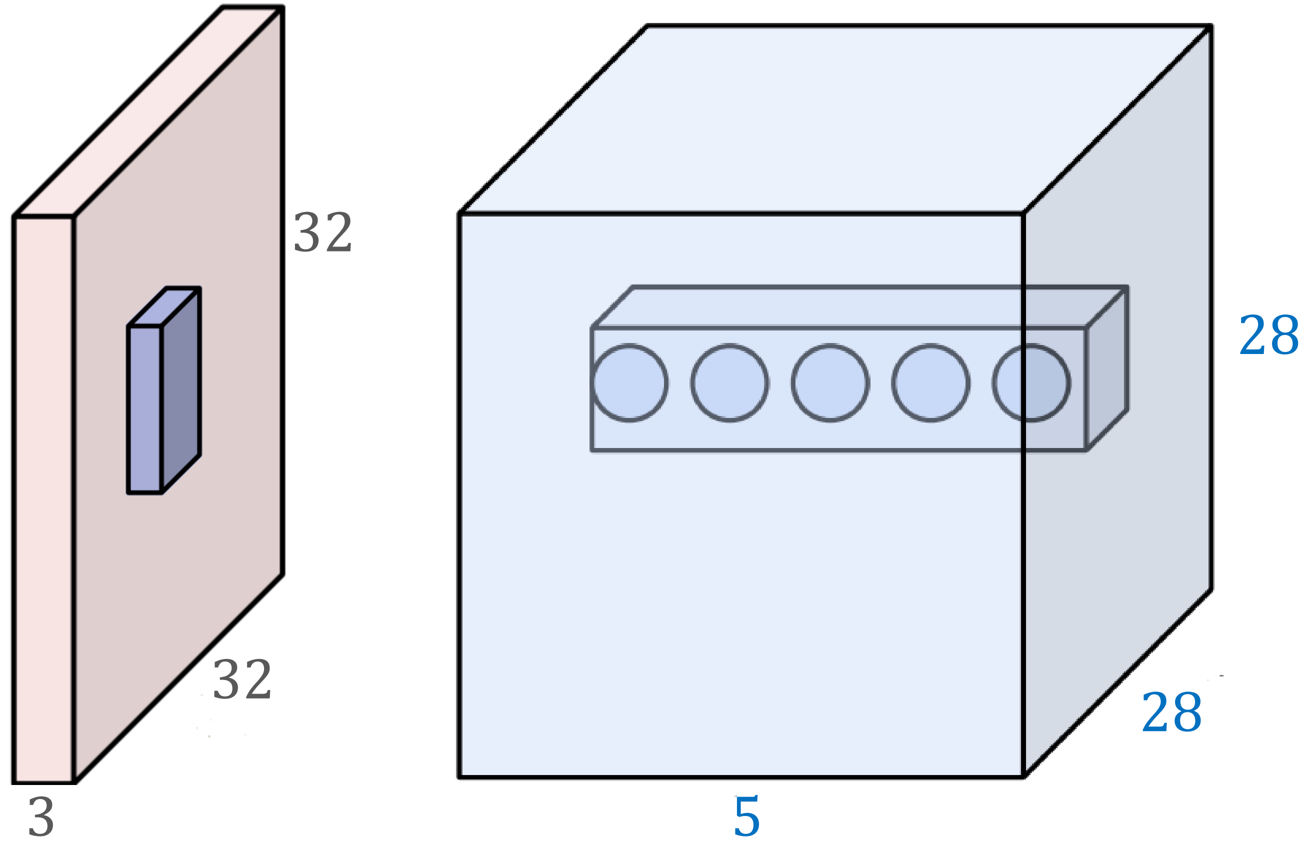
An **activation map** is a 28x28 sheet of neuron outputs:

1. Each is connected to a small input region
2. All of them share the same parameters

(the parameters are the filters values)



Convolutional layer



Using 5 filters, we are stacking neurons in a **matrix** $28 \times 28 \times 5$

This means that, somehow, **5 different neurons** are looking at the **same piece of image** to produce an output (each neuron will specialize to recognize a certain property of the image)

Convolutional layer: summary

- **Input:** a volume of size $W_1 \times H_1 \times D_1$
- **Output:** a volume of size $W_2 \times H_2 \times D_2$

✓ $W_2 = \frac{W_1 - F + 2P}{S} + 1$ ✓ $D_2 = K$

✓ $H_2 = \frac{H_1 - F + 2P}{S} + 1$

- It introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights (10 filters with a dimension of 5×5 on an RGB image will have $(5 \cdot 5 \cdot 3) \cdot 10 = \mathbf{750}$ parameters)

- **Hyperparameters**
  1. Number of filters K
  2. Their spatial extent F
  3. The amount of zero padding P
  4. The stride S

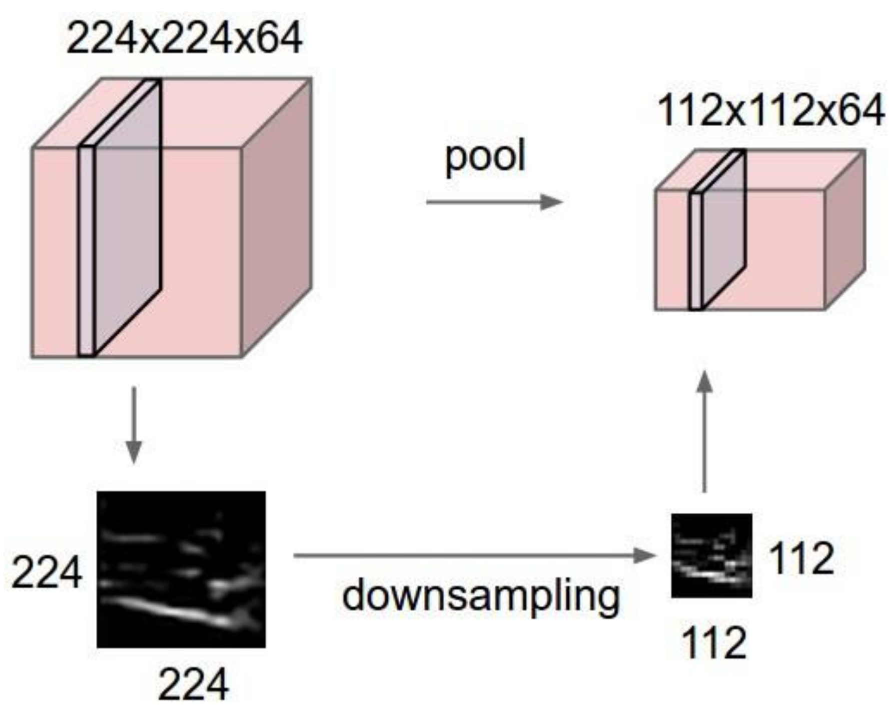
Common settings

- $K =$ powers of 2, e.g. 32, 64, 128, 512)
- $F = 3, \quad S = 1, \quad P = 1$
- $F = 5, \quad S = 1, \quad P = 2$
- $F = 5, \; S = 2, \; P = ?$ (whatever fits)
- $F = 1, \; S = 1, \; P = 0$

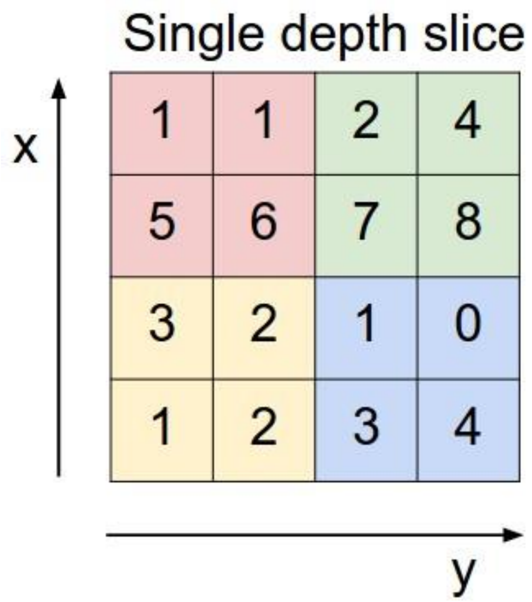
Pooling layer

This layer **reduces the spatial size** of the image representation

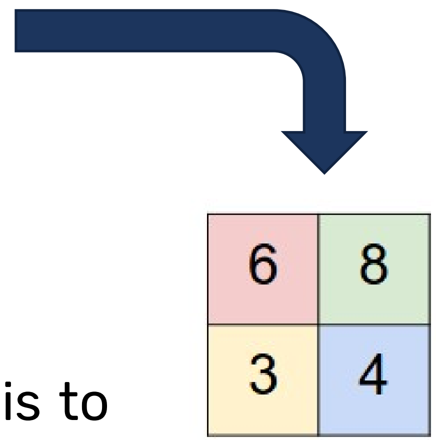
- It aims to reduce the amount of **parameters** and computation in the network, and hence to also control overfitting



MAX POOLING
2 × 2 filter
Stride of 2



1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4



6	8
3	4

The idea of **max pooling** is to reduce the size keeping the **«mostly activated»** neurons

Pooling layer

- **Input:** volume of size $W_1 \times H_1 \times D_1$
- **Output:** a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = \frac{W_1 - F}{S} + 1$
  - $H_2 = \frac{H_1 - F}{S} + 1$
  - $D_1 = D_2$

- **Hyperparameters**:
  - Their spatial extent F
  - The stride S

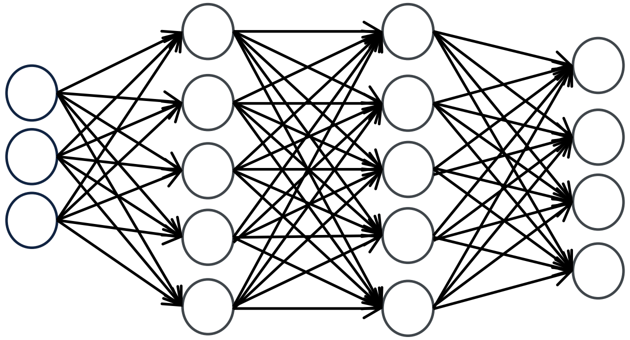
- Introduces **zero parameters** since it computes a fixed function of the input

- Different versions of pooling exist. The most used is **max pooling**. Other types are **average** pooling and **global** pooling

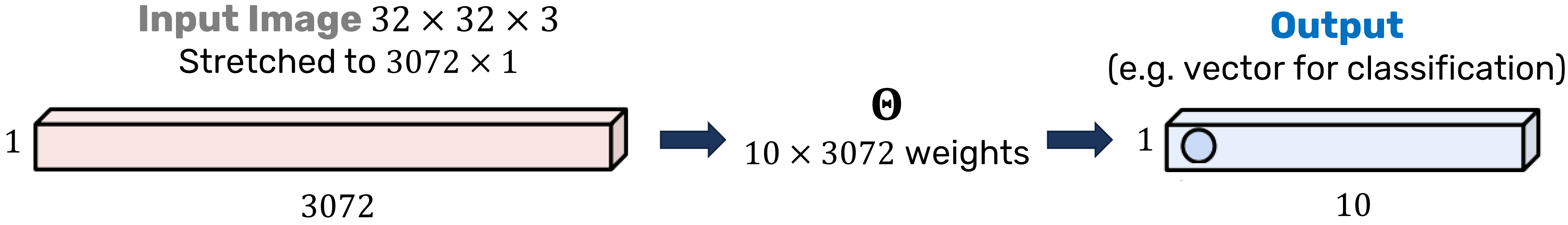
Fully connected layer

These layers are just like the classic neural network layers, i.e. **multi-layer perceptrons** (MLP) networks

- All the inputs are connected to all the outputs



They have to **«map»** the **high level features,** extracted by previous layers, to the output

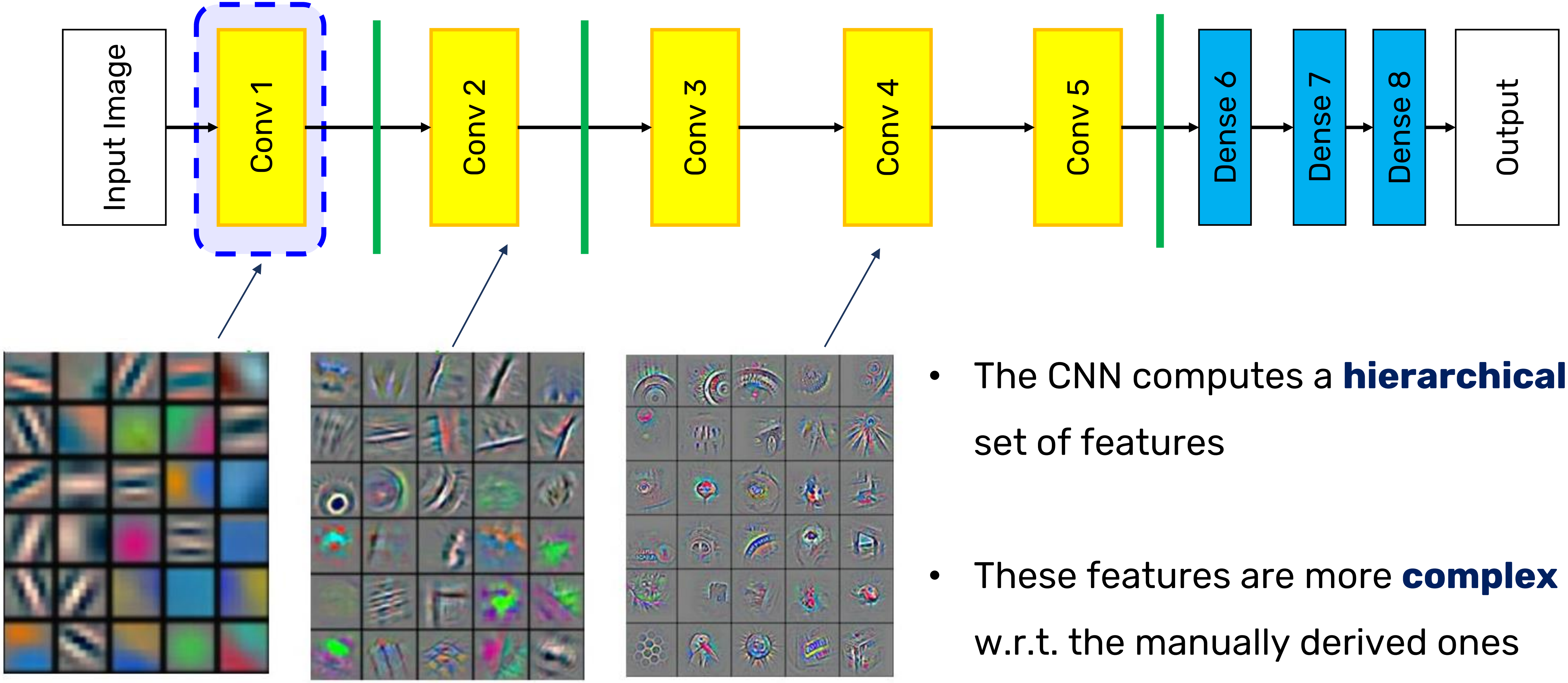


Input Image $32 \times 32 \times 3$
Stretched to 3072×1

$\mathbf{\Theta}$
10×3072 weights

Output
(e.g. vector for classification)

Alex-net structure



- The CNN computes a **hierarchical** set of features

- These features are more **complex** w.r.t. the manually derived ones

Figure credit: Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014

Other networks

- **GoogLeNet**. Its main contribution was the development of an *Inception Module* that dramatically reduced the number of parameters in the network (4M, AlexNet: 60M).

- **VGGNet**. Its main contribution was in showing that the depth of the network is a critical component for good performance. Their final best network contains 16 CONV/FC layers and an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end.

- **ResNet**. Residual Network was the winner of ILSVRC 2015. It features special *skip connections* and a heavy use of batch normalization. The architecture is also missing fully connected layers at the end of the network (as of May 10, 2016).

Architecture	# Params	Size	Accuracy	Year	# operations	FW time [GPU]	FW time [CPU]
AlexNet	61 Milions	238 MB	80.2 %	2012	724 M	3.1 ms	0.29 s
Inception V1	7 Milions	70 MB	88.3 %	2014	1.43 B	-	-
VGGNet	138 Milions	528 MB	91.2 %	2014	15.5 B	9.4 ms	4.36 s
ResNet-50	25.5 Milions	99 MB	93 %	2015	3.9 B	11 ms	1.13 s

Outline

1. Convolutional neural networks

2. **Object detection**

3. Transfer learning

4. Hardware

5. Application to pneumonia detection using X-ray images

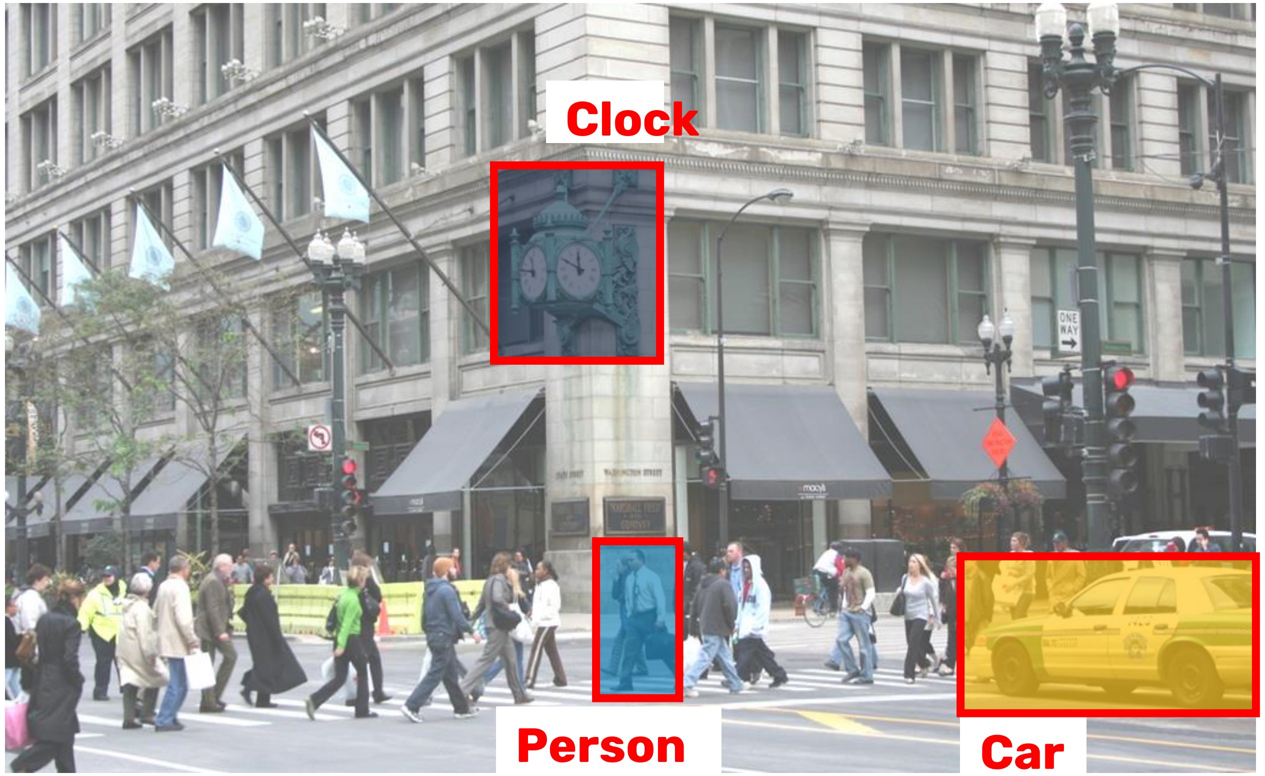
Computer vision tasks: reminder

Classification

What's in the image?

Detection

What's in the image? **Where** it is?



How to use a CNN on your own data

GENERAL SCHEME

1. **Gather** your own data
   - ✓ Data specific of the applicative domain (medical images, manufacturing production pieces,…)
   - ✓ Label the data (you can use open source tool for labelling images)

2. **Reuse** a CNN
   - ✓ You do not have to train a CNN from zero. You can download a pre-trained CNN and modify (train) only the last fully connected layers (transfer learning)

3. **Perform** task (object classification or object detection)

Data gathering

The dataset must be:

1. As **large** as possible (200 items per class at least)
2. With the same object in **different «conditions»** (background, lights)
3. With **random objects** along with the desired object
4. It should respect the **application condition**. Decide if you need partial objects, overlapping and so on
5. With **no label errors**
6. Not too large (less training time)

You can create the dataset taking pictures (e.g. smartphone) or from Google Images

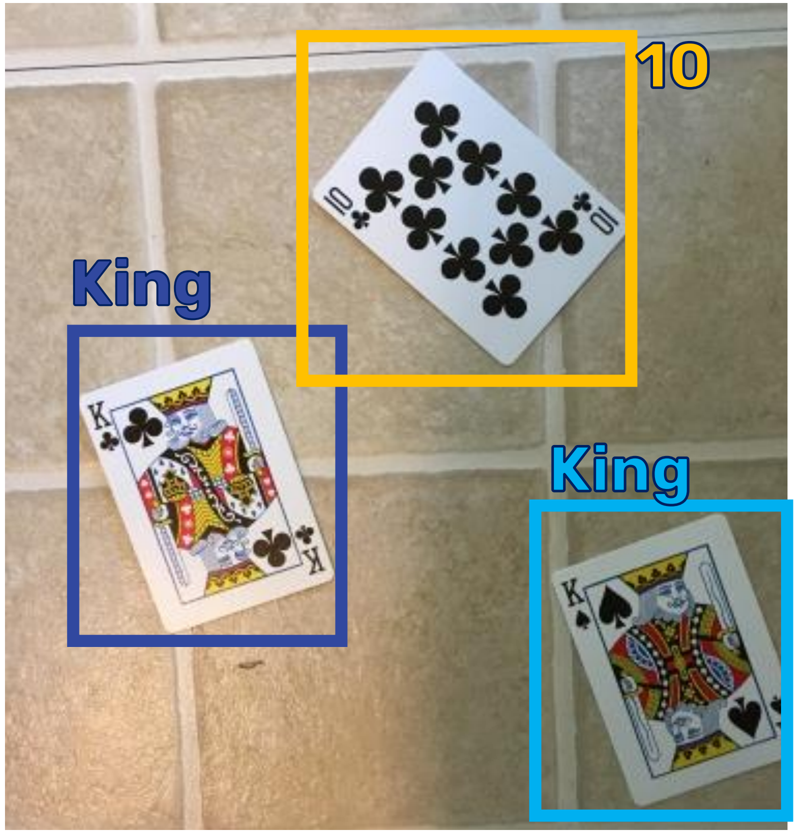
Labelling

We are building an object detection, so the label process will involve the creation of **bounding boxes** (i.e. the label is not only the class but also the box coordinates)

We can use a lot of open-source software (**LabelImg** https://github.com/tzutalin/labelImg)

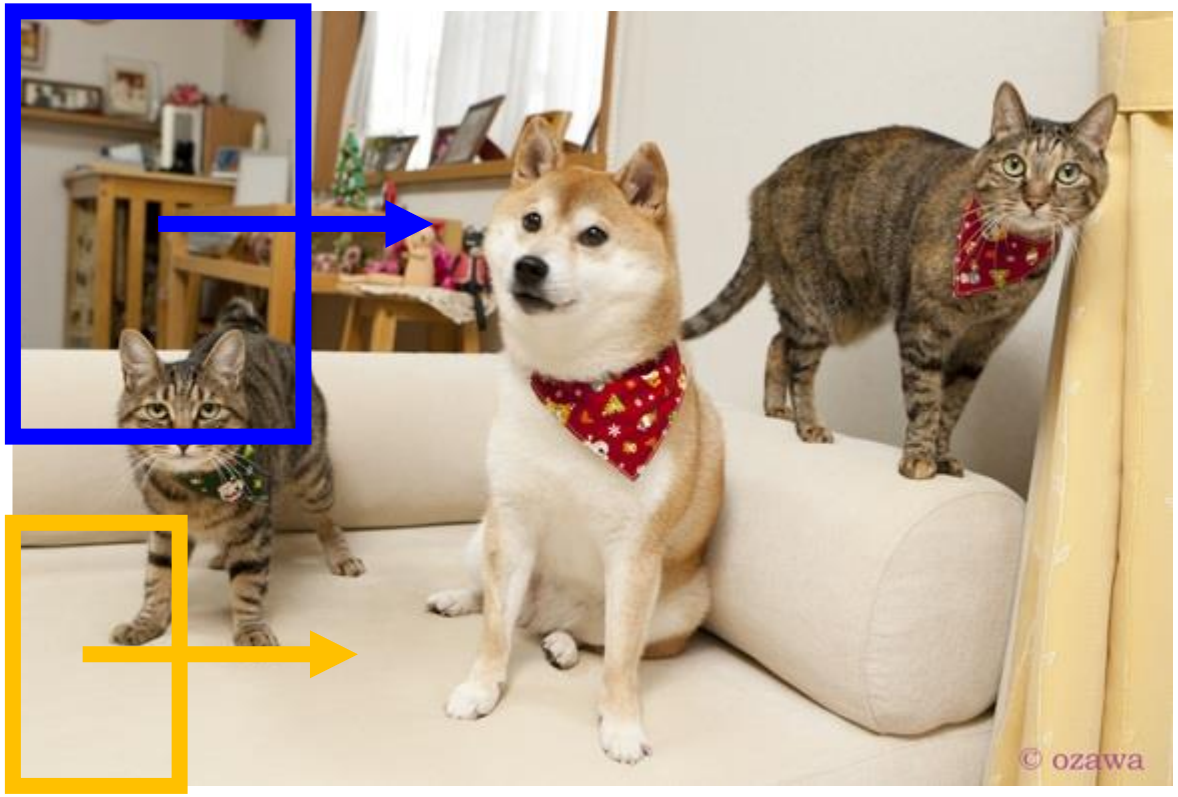
It will create an XML file associated to the image

```
<object>
        <name>ten</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
                <xmin>145</xmin>
                <ymin>68</ymin>
                <xmax>303</xmax>
                <ymax>225</ymax>
        </bndbox>
</object>
```



Object detection

So far we learned how to do Image Classification. We can use the **same models** and **slide a window** over the image



For each window, perform a classification

PROS: Effective

CONS: Not efficient. Different window shapes in different positions. **Huge amount of time**

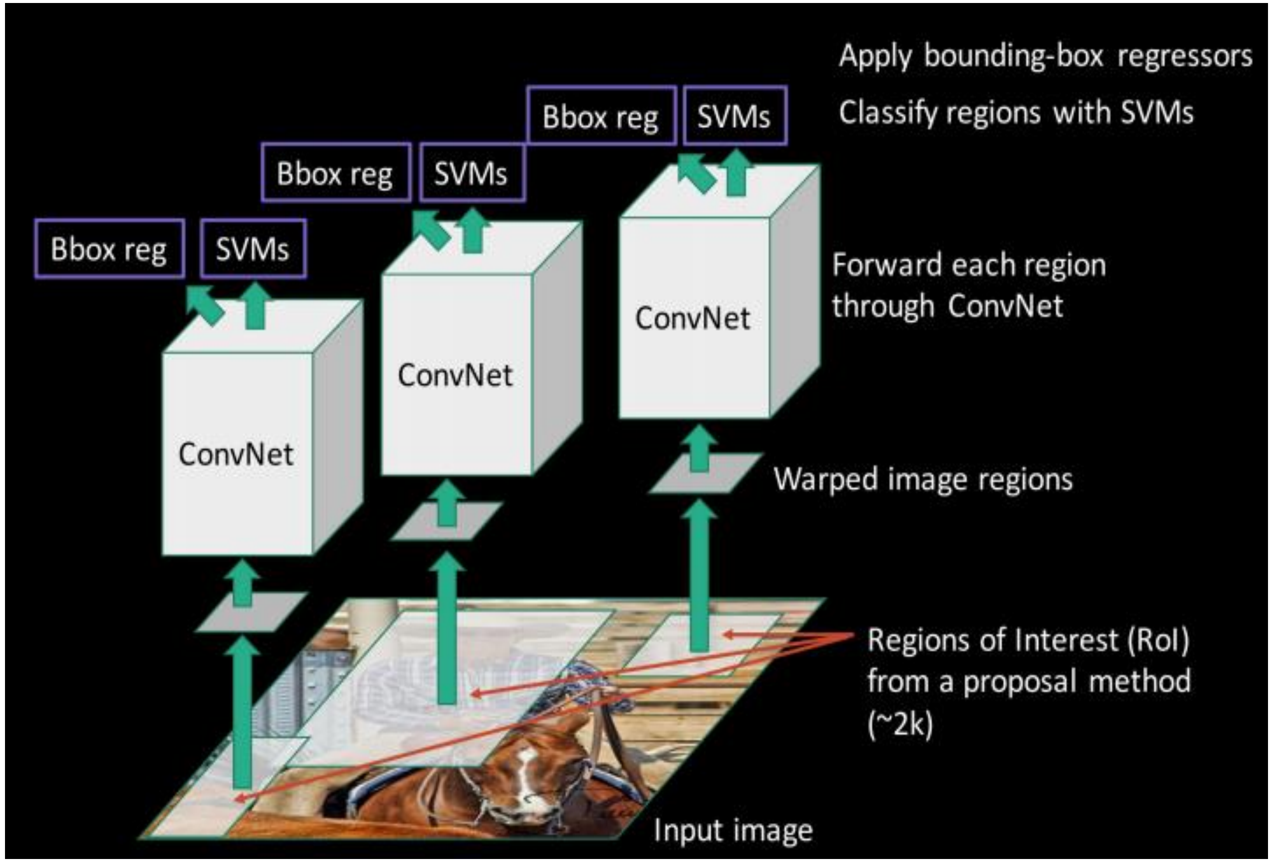
Object detection: classification-based methods

Region proposal: run the CNN only on image parts which can contain an object

Two-stage algorithms

1. A proposal algorithm is run to select proposal regions
2. Perform classification with CNN

- **R-CNN**
- **Fast-RCNN**
- **Faster-RCNN**

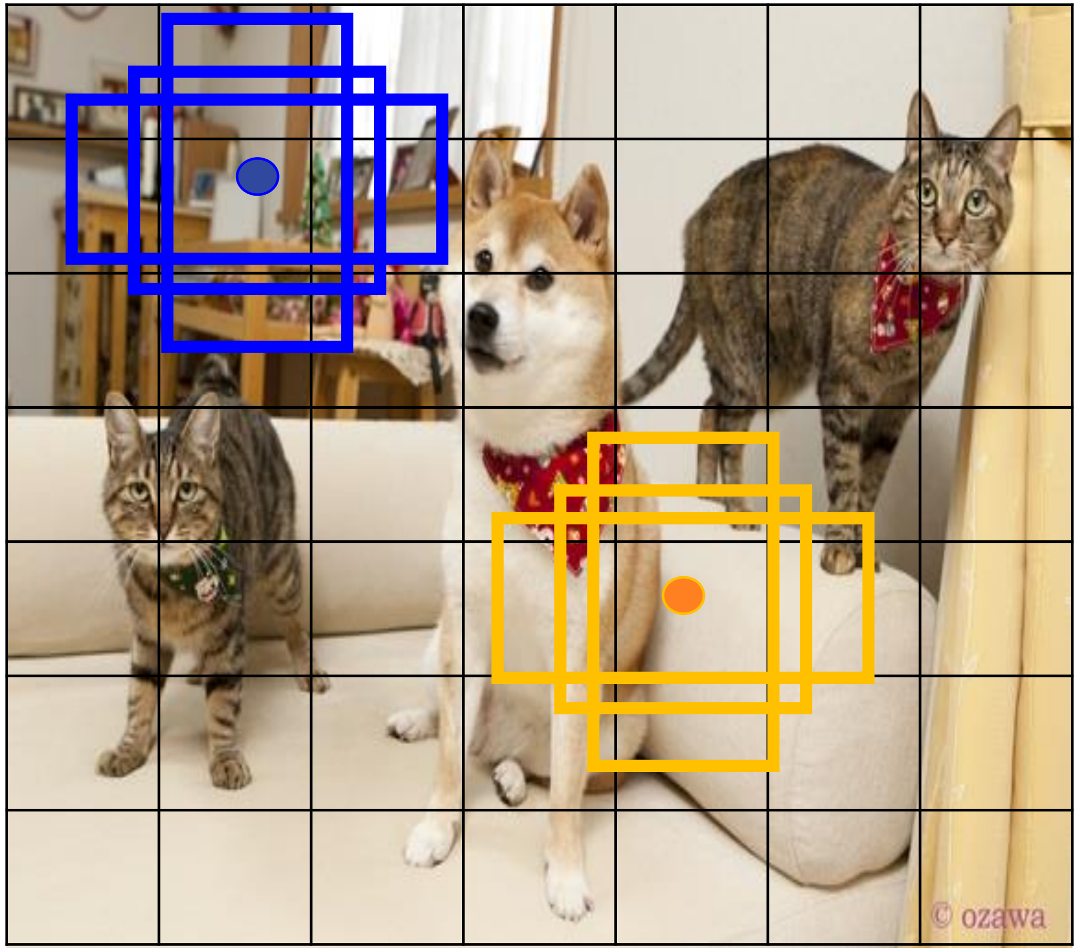

Apply bounding-box regressors
Classify regions with SVMs
Bbox reg
SVMs
Bbox reg
SVMs
Bbox reg
SVMs
ConvNet
ConvNet
ConvNet
Forward each region through ConvNet
Warped image regions
Regions of Interest (RoI) from a proposal method (~2k)
Input image

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014

Object detection: regression-based methods

NO region proposal: run the CNN in «pre-defined» areas

- The input to the CNN is the **whole image**
- Divide the image into a **grid** (7×7 in this case)
- Each grid cell predicts B **boxes** (centered at that grid cell) and their confidence for containing and object of each class
- The most confident boxes are retained



- **YOLO** (You only look once)
- **SSD** (Single shot detection)

Outline

1. Convolutional neural networks

2. Object detection

3. **Transfer learning**

4. Hardware

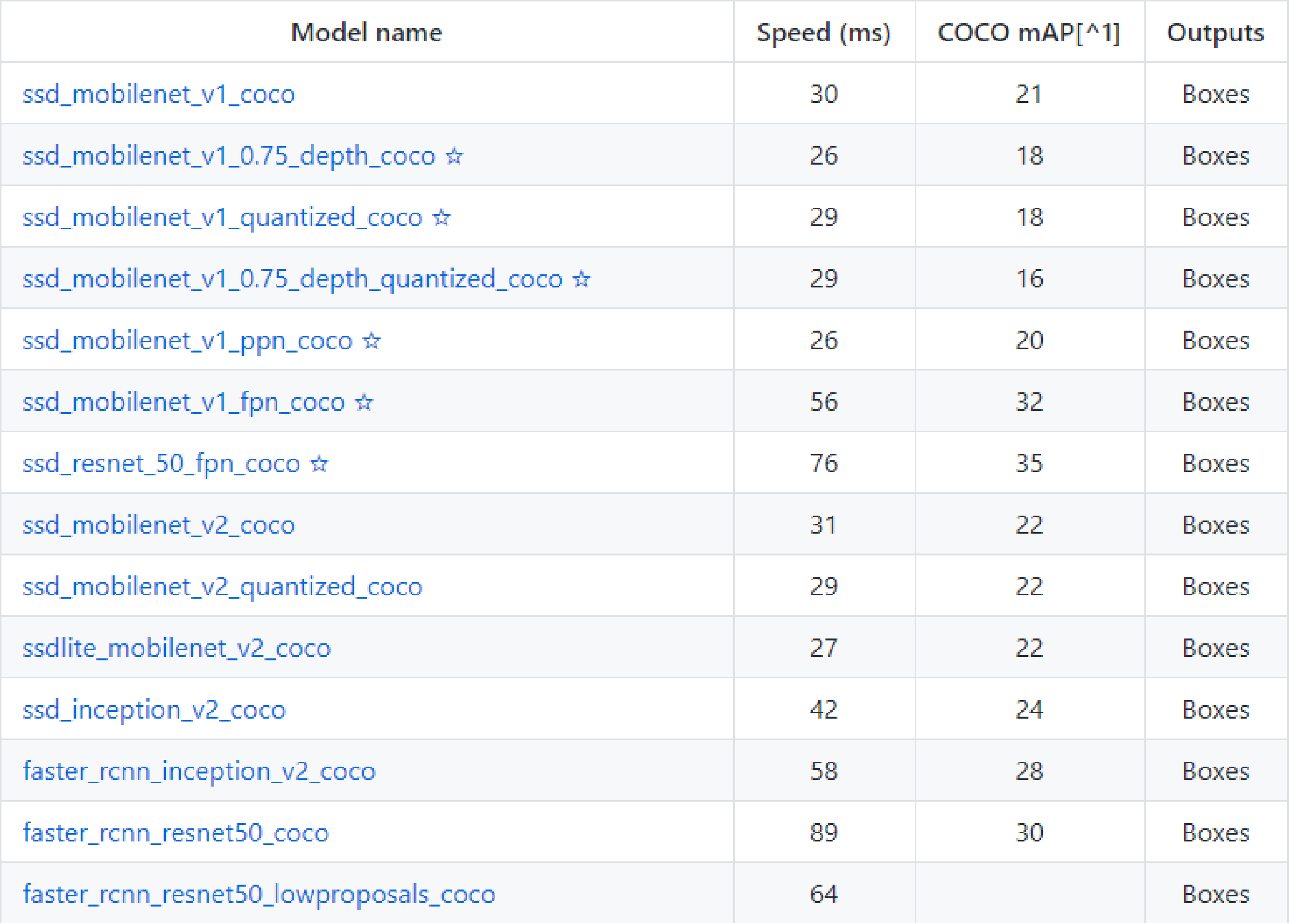
5. Application to pneumonia detection using X-ray images

Transfer learning

Pre-trained models are available

The weights we download, however, are trained on some **other dataset** (COCO, Pascal, Kitti, …)

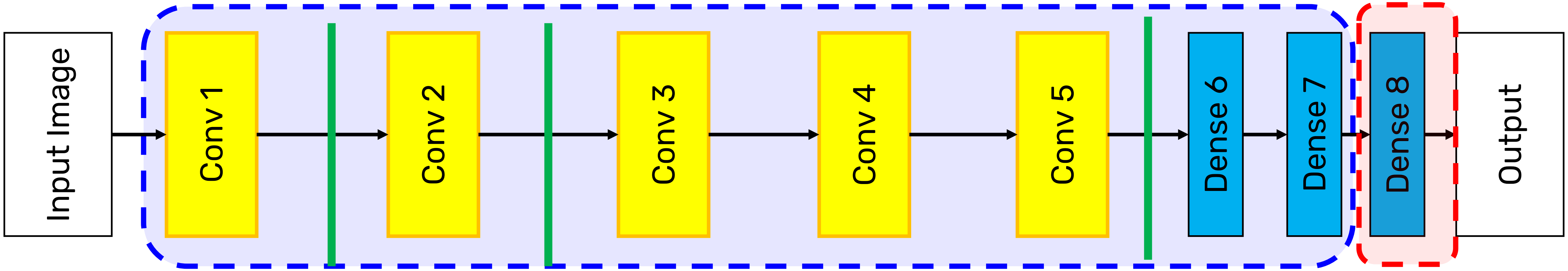
Transfer learning



Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v1_0.75_depth_coco ☆	26	18	Boxes
ssd_mobilenet_v1_quantized_coco ☆	29	18	Boxes
ssd_mobilenet_v1_0.75_depth_quantized_coco ☆	29	16	Boxes
ssd_mobilenet_v1_ppn_coco ☆	26	20	Boxes
ssd_mobilenet_v1_fpn_coco ☆	56	32	Boxes
ssd_resnet_50_fpn_coco ☆	76	35	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes
ssd_mobilenet_v2_quantized_coco	29	22	Boxes
ssdlite_mobilenet_v2_coco	27	22	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes

https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

Transfer learning



Transfer learning **re-uses the ability learnt** in another task

- **«Freeze»** the convolutional layers, that do the feature extraction part
- **Train** the last full connected layer, that specializes to classify your data. You can also **train all** the dense layers if you have enough data

Transfer learning

Example: object detection of golf cars. A network that learned how to detect cars can probably generate features that are important also for detecting golf cars

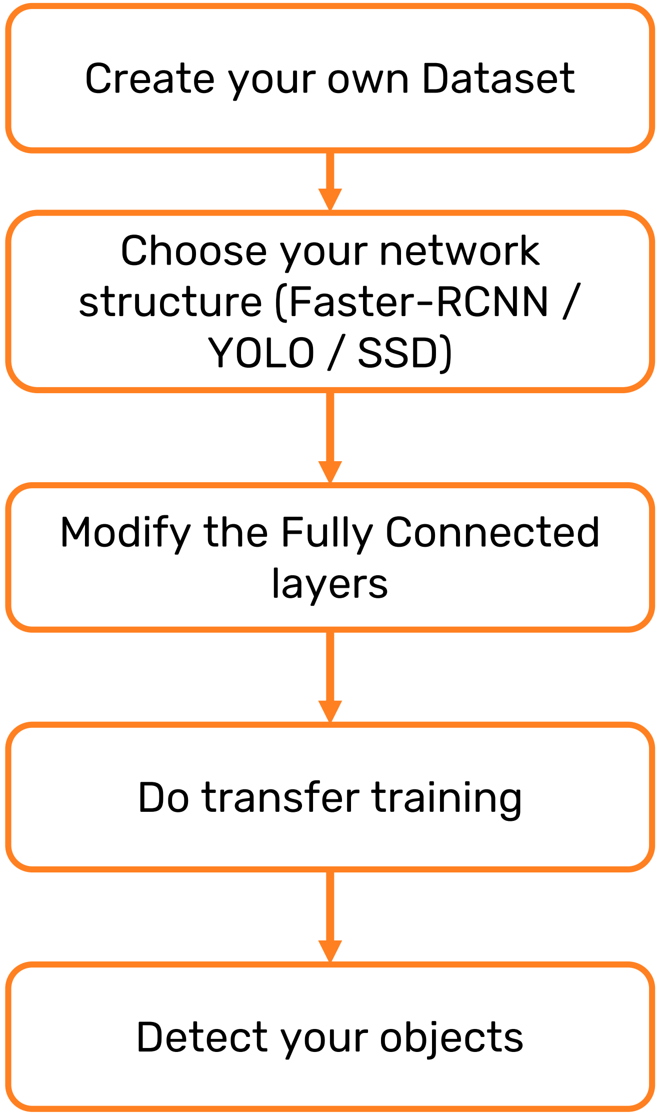
Car object detection

10000 images

Golf car object detection

100 images

Object detection pipeline



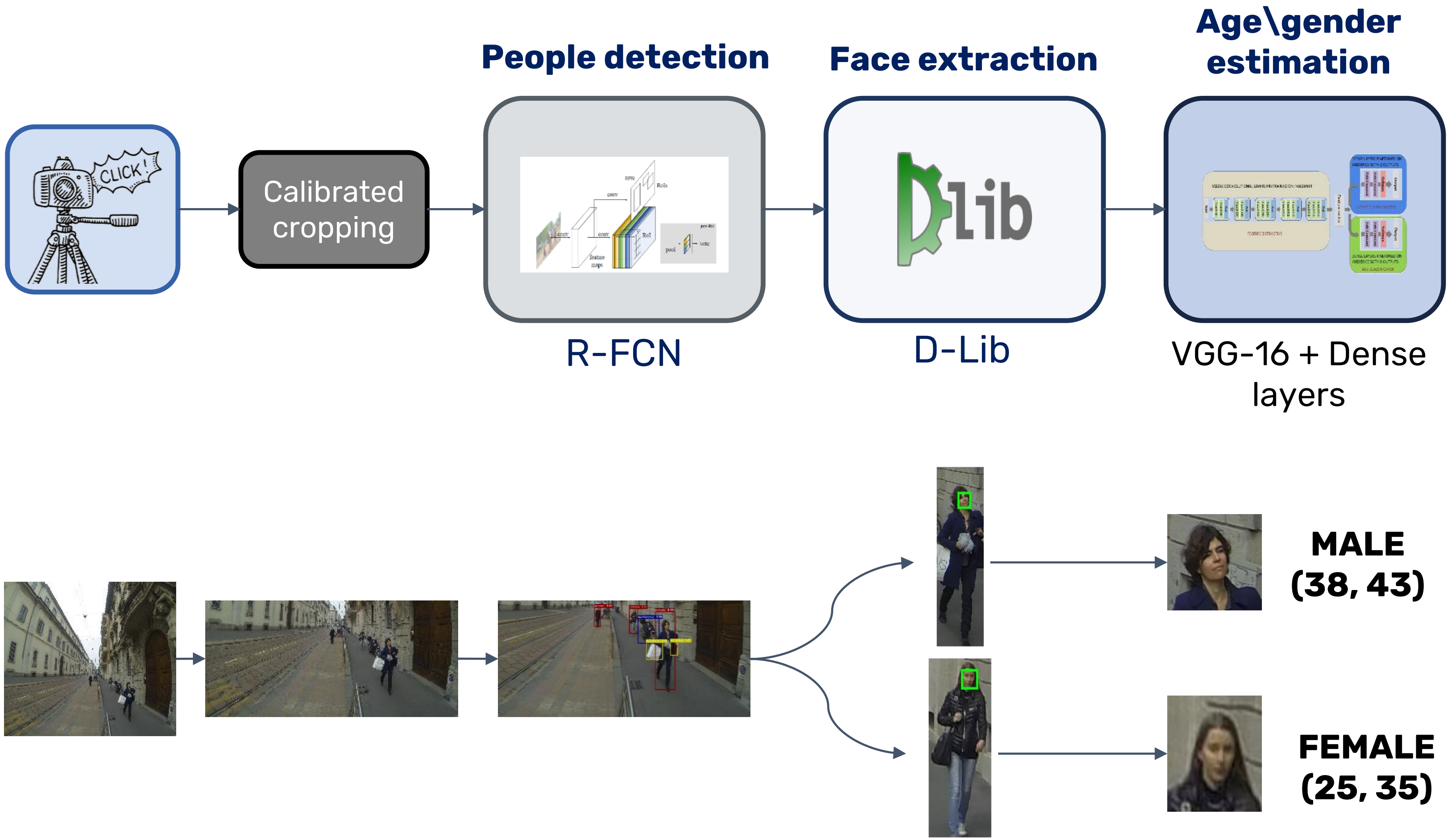
Create your own Dataset

Choose your network structure (Faster-RCNN / YOLO / SSD)

Modify the Fully Connected layers

Do transfer training

Detect your objects



People detection

Face extraction

Age\gender estimation

Calibrated cropping

R-FCN

D-Lib

VGG-16 + Dense layers

MALE (38, 43)

FEMALE (25, 35)

Outline

1. Convolutional neural networks

2. Object detection

3. Transfer learning

4. **Hardware**

5. Application to pneumonia detection using X-ray images

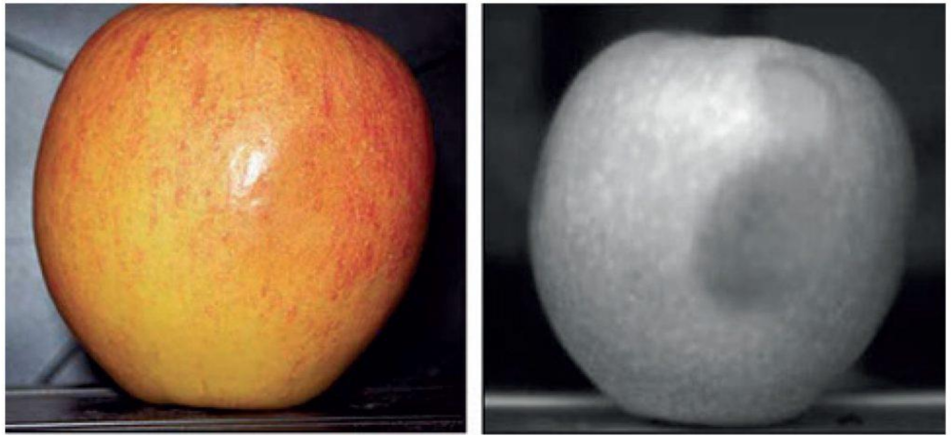
Hardware

The hardware plays an important role, since most the times images have to be **acquired**

Garbage IN, garbage OUT: if we feed the networks with low quality images, we should not expect good results

You need to consider, among other things:

- **Light conditions** (try to control the environment light with **illuminators**)
- **Camera type** (RGB, Near-infrared, …)
- **Optics** (360° field of view, 3D cameras,…)
- **Communication interface** (USB, PoE, VGA,…)
- **Megapixels**



Source: imagesspa.it

Outline

1. Convolutional neural networks
2. Object detection
3. Transfer learning
4. Hardware
5. **Application to pneumonia detection using X-ray images**

Disclaimer

This example is **ONLY** for **educational purposes**, in order to see how to train and use a convolutional neural network in practice with real data.

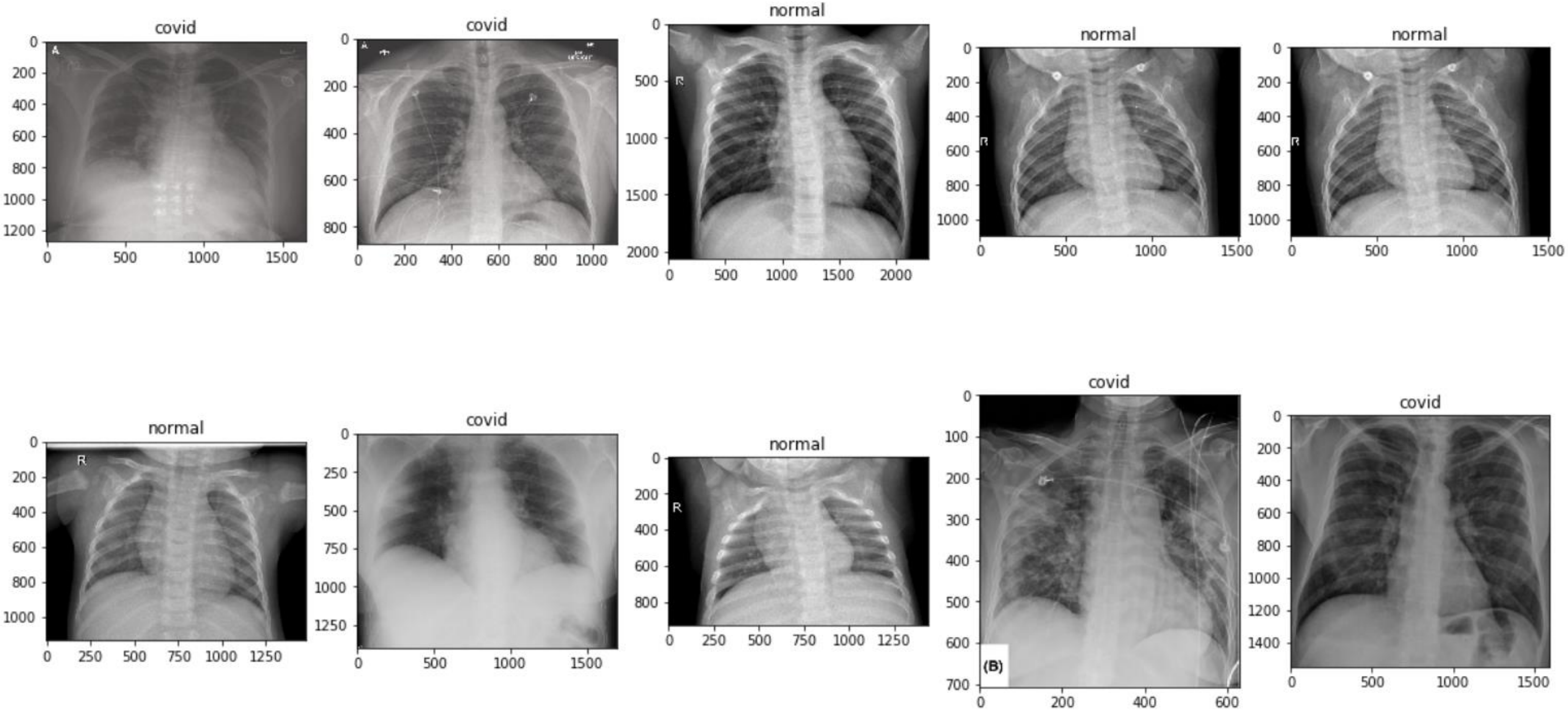
I am **NOT**, by any means, trying to say that this should be an accurate or valid system from a medical point of view.

Artificial intelligence tools show **ALWAYS be supported** by domain knowledge from humans.

Again, this example does not claim to solve COVID-19 detection.

Pneumonia detection

Suppose to have at disposal X-ray images of lungs: **Healthy** people - **Covid-19 disease** patients



This example is **ONLY** for **educational purposes**

Acknowledgments

- The COVID-19 X-ray image is curated by Dr. Joseph Cohen, a postdoctoral fellow at the University of Montreal, see https://josephpcohen.com/w/public-covid19-dataset/

- The previous data contain only X-ray images of people with a disease. To collect images of healthy people, we downloaded another X-ray dataset on the platform Kaggle https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

- The analysis is inspired from a tutorial by Adrian Rosebrock: https://www.pyimagesearch.com/2020/03/16/detecting-covid-19-in-x-ray-images-with-keras-tensorflow-and-deep-learning/

This example is **ONLY** for **educational purposes**

Pneumonia detection

We want to use a CNN to perform classification:

- **Healthy** patients: class 0
- Patients with a **disease**: class 1

Followed procedure:

1. **Split** the dataset in train and test data
2. **Data augmentation:** generate new images by transforming the original training ones
3. **Download** VGG-16 net already trained on ImageNet dataset
4. **Train** only the final dense layers of the network (transfer learning)

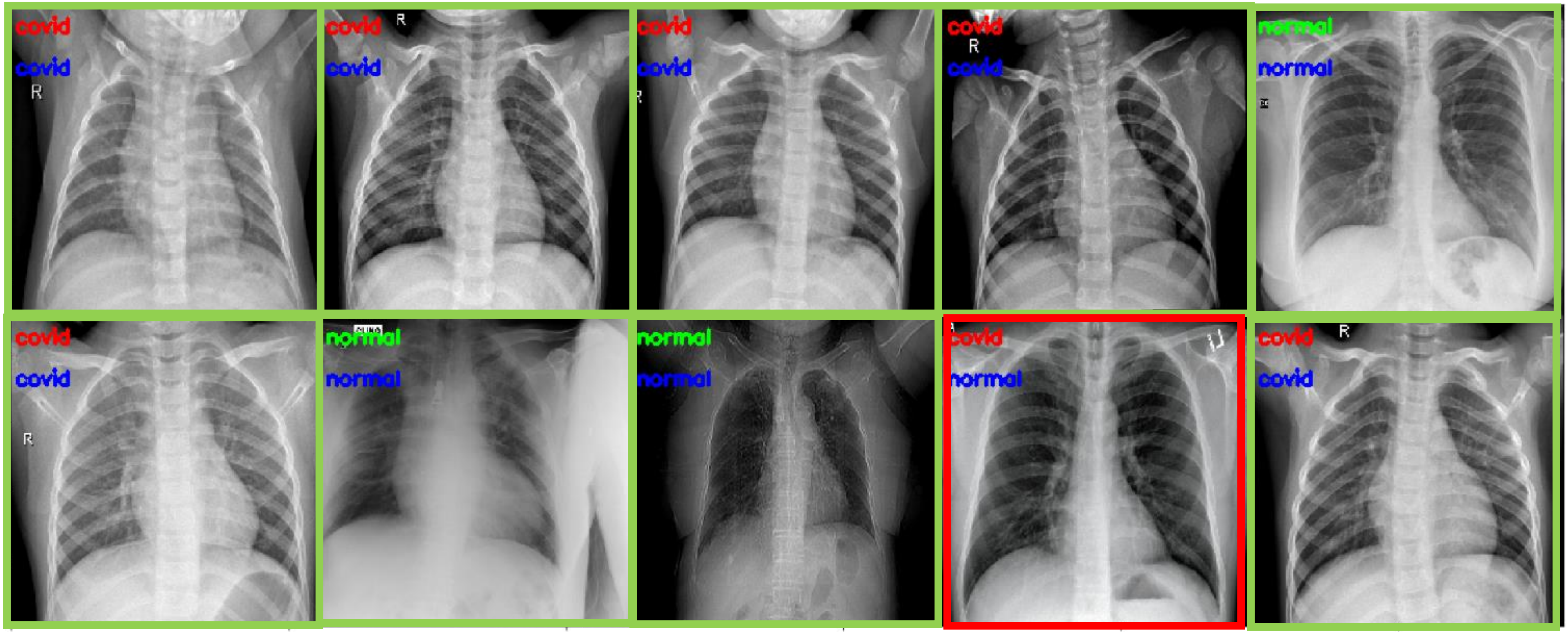
This example is **ONLY** for **educational purposes**

Pneumonia detection

True label

Predicted covid label

Predicted healthy label



This example is **ONLY** for **educational purposes**

Pneumonia detection

Classification results

Sensitivity (recall, true positive rate)

$$\frac{\text{True Positive}}{\text{\# Actual Positive}} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} = 0.92$$

Specificity (true negative rate)

$$\frac{\text{True Negative}}{\text{\# Actual Negative}} = \frac{\text{True Negative}}{\text{False Positive} + \text{True Negative}} = 1$$

		Actual class	
		1 (p)	0 (n)
Predicted class	1 (Y)	True positive 11	False positive 0
	0 (N)	False negative 1	True negative 11

- **Accuracy**: $\approx 96\%$

- Being able to accurately detect healthy patients with 100% accuracy is great
- We don't want to classify someone as «negative» when they are «COVID-19 positive»

This example is **ONLY** for **educational purposes**