Neural Networks in Computer Vision and Natural Language Processing

Stefano Melacci

Department of Information Enginnering

University of Siena

Disclaimer

The contents (and the style) of these slides are taken from Stefano Melacci's slides of the *Machine Learning & Deep Learning* course - Datum Academy (France)

They are used with the sole purpose of supporting Stefano Melacci's teaching activity.

They are not intended to be of public domain in any way, and they must not be published or shared out of the context in which they are presented by Stefano Melacci.

Computer Vision & Natural Language

- This module focuses on applications of deep networks to Computer Vision and Language Modeling, exploiting Convolutional Neural Networks and Recurrent Neural Networks, respectively

- **Convolutional Neural Networks**
  - Neural Networks that implement multi-layer convolutions
  - *Applications to several problems*
  - We will study the case of **Computer Vision (Image Classification)**

- **Recurrent Neural Networks**
  - Neural Networks that can process sequences
  - *Applications to several problems*
  - We will study the case of **Natural Language (Language Modeling)**

Computer Vision & Natural Language (2)

- Overview
  1. **Computer Vision, Image Classification & Neural Networks**
     - Classifying Images with Neural Networks, MLPs
  2. **Convolution & Images**
     - Convolution, Blurring, Receptive Fields, Feature Maps
  3. **Convolutional Neural Networks**
     - From MLPs to Convolutional Networks, Pooling, Stride
  4. **Image Classification with Convolutional Neural Networks**
     - Learning Convolutional Networks in Image Classification Tasks, Properties
  5. **Natural Language Modeling & Neural Networks**
     - Natural Language, Sequences & Neural Networks
  6. **Recurrent Neural Networks**
     - Recurrent Nets, Processing Sequences, Training, Long-term Dependencies
  7. **Language Modeling with Recurrent Neural Networks**
     - Exploiting RNNs in Language Modeling, Representing Words
  8. **Long-Short Term Memories**
     - LSTM Networks

Computer Vision, Image Classification & Neural Networks

Stefano Melacci

Computer Vision

- *Computer Vision is a very large and interdisciplinary field*
  - It studies how machines can be used to interpret/understand visual signals
  - Visual signals: images, videos (sequences of images)
  - Use machine to approach vision-based tasks that we (as humans) are able to approach
- **Machine Learning** and **Computer Vision**
  - Machine Learning tools are strongly exploited in the context of Computer Vision
  - Learn to extract information from images/videos
  - Develop machines to solve tasks that require to take decisions on images/video
- Popular tasks
  - Image Classification (main object, scene, …)
  - Object Detection
  - Event Classification
  - Semantic Segmentation
  - …

Image Classification

- Let us focus on **Image Classification**
- Given an input image I and a set of target classes, what is the class to which I belongs?

I

Classes:
dog
→ **cat**
rabbit
lion
…

Image Classification (2)

- **Rule-based** Image Classification
  - Define a list of rules that can be used to take a decision on the class to which the input image belongs
    - Rules operate on pixel intensities (the lowest-level information stored in the input image)
    - Rules could operate on higher-level concepts, *if available*

- Toy examples of pixel-level rules
  - *If I is mostly blue, then it belongs to class "sky"*
  - *If I is mostly green, then it belongs to class "grass"*
- Toy example of concept-level rules
  - *If I has a furry face, ears, nose, eyes and whiskers, then it belongs to class "cat"*

- Issues
  - **Rules do not scale well**
  - Concepts might be not available

Image Classification (3)

- **Machine Learning-based** Image Classification
  - Learn from data the function *f(I)* that can be used to determine the class of image *I*
  - *f(I)* is a **Neural Network** in the context of this course
  - It could be another classifier as well

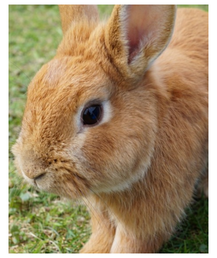
I

Classes:

f(I) = dog
→ **cat**
rabbit
lion
…

Supervised Image Classification

- We consider the context of **supervised learning**, where we are given a collection of images paired with class labels that are used to train the Neural Net
- Class labels (cat, dog, rabbit, lion, …) are converted into unique *numerical identifiers*, ranging from *1* to c (for example), where *c* is the number of classes


rabbit → *class 3*

cat → *class 1*

dog → *class 2*

…

Supervised Image Classification (2)

- Recognizing digits from images of handwritten elements can be seen as an Image Classification task
- It is straightforward to convert class labels to numerical class identifiers (digit number = class identifier), but a different mapping could be used as well

digit 3 → *class 3*

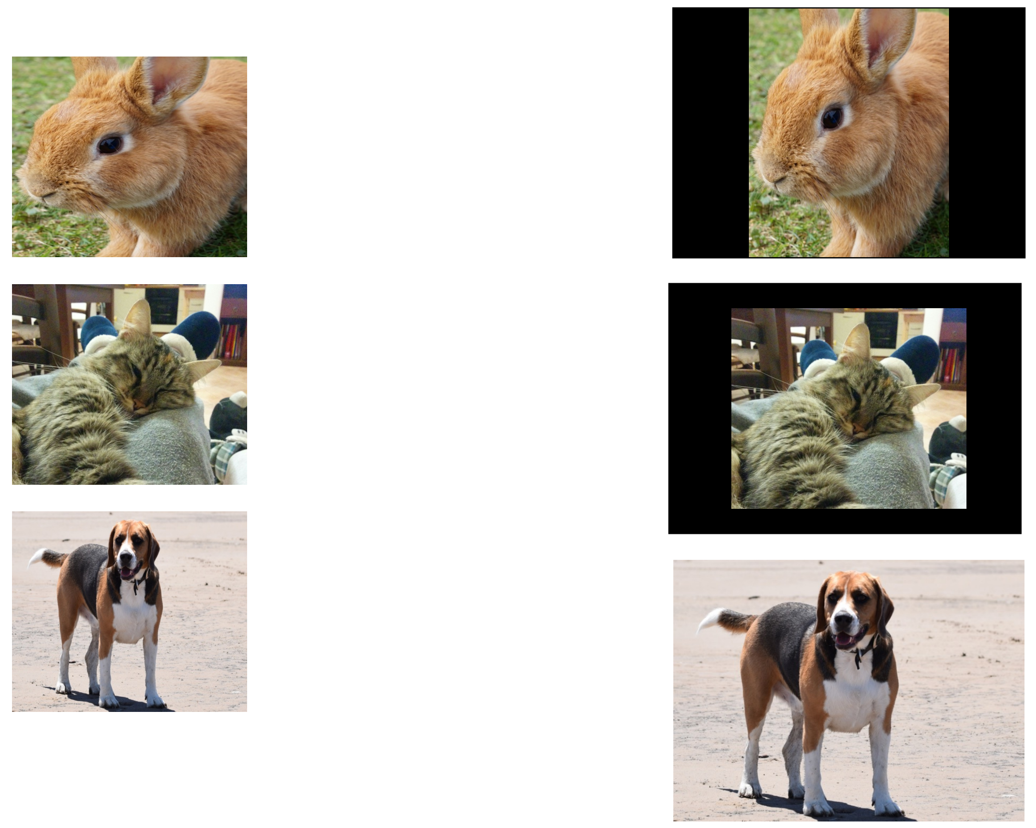
digit 6 → *class 6*

digit 8 → *class 8*

…

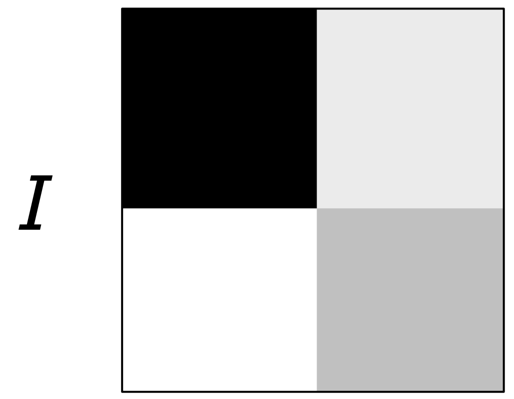
Supervised Image Classification (3)

- Images might have different **resolutions** and/or number of **channels**, or being represented in different color spaces
  - Landscape vs. portrait images, color vs. grayscale images
- Basic pre-processing step: standardize images to the same resolution, color space (example: color to grayscale)



Warning!

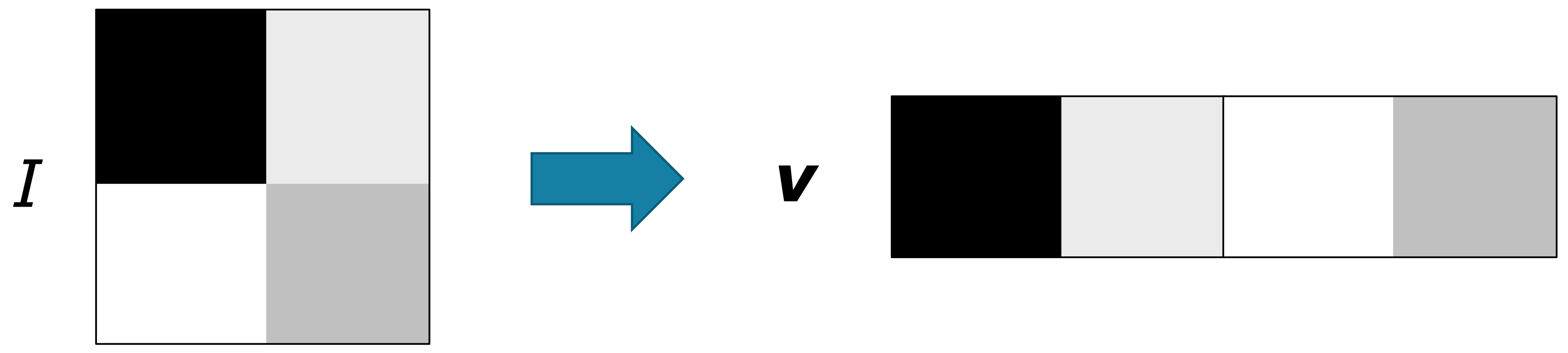
- In the next slides, the **generic input image** will be frequently represented with the following picture



- It is composed of **4 pixels**, so it is *very simple,* very good to make diagrams easy and readable
- Please consider that this picture is only used as a symbolic representation, it is just a placeholder, *think about it as a higher resolution picture*
  - It might also be a **rectangular** picture (either portrait or landscape)

Image Classification by MLPs

- If we consider **Multi Layer Perceptrons** (MLPs) as Neural Network type, we need to convert the image matrix I (tensor) into a *vector* $\boldsymbol{v}$
  - Example: stack the rows of the image into a vector



- Then, the MLP computes the function $\boldsymbol{o} = f(\boldsymbol{v})$
  - $\boldsymbol{o}$ is a vector with c components with the activation score of each class
  - The i-th component of $\boldsymbol{o}$ is the activation of the i-th class
  - $argmax(\boldsymbol{o})$ returns the identifier of the predicted class

Image Classification by MLPs (2)

- Each neuron of a MLP computes a projection that involve all the outputs of the neurons belonging to the previous layer
  - See the light-blue neuron in the first hidden layer of the following pictures, or the violet neuron in the output layer

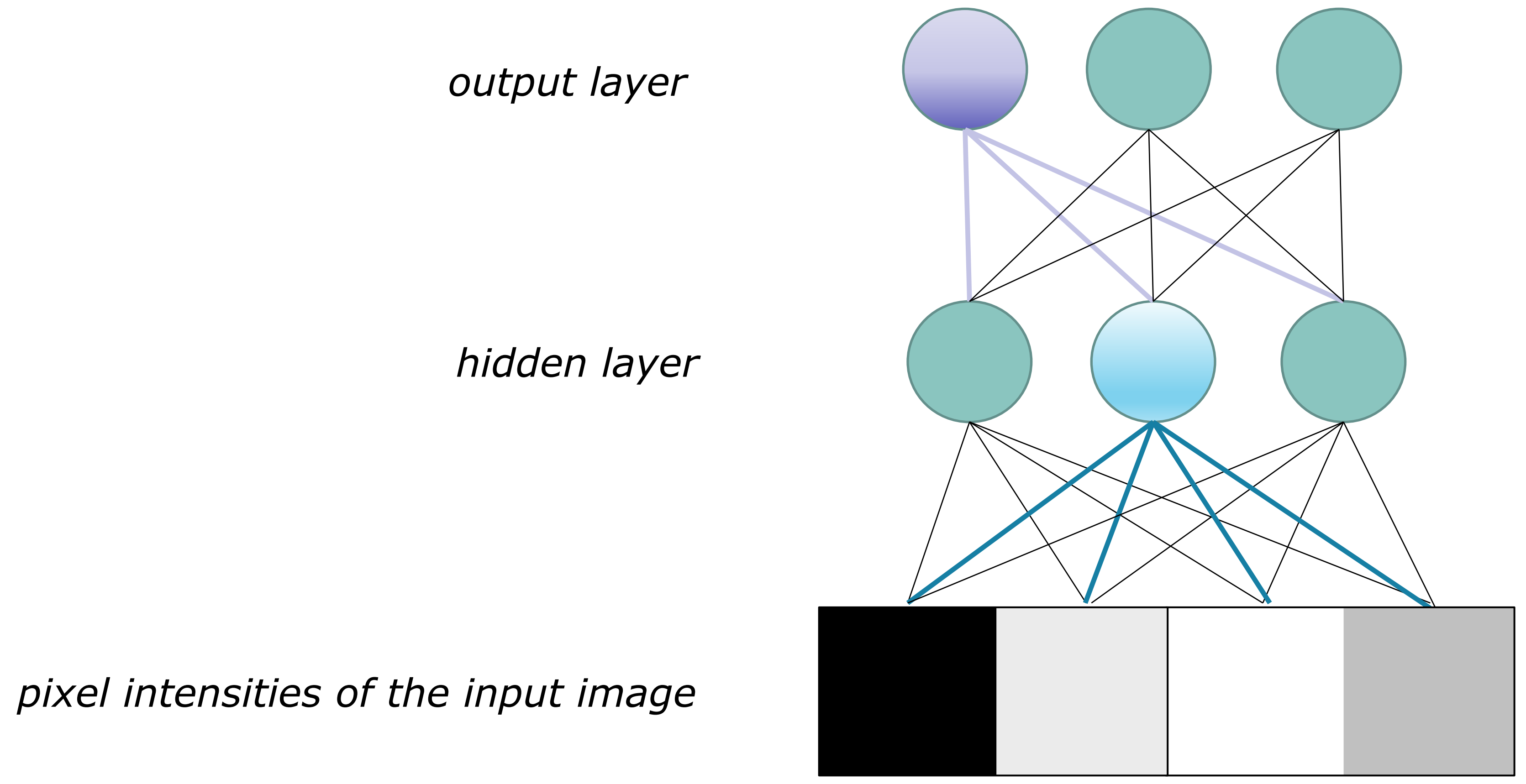
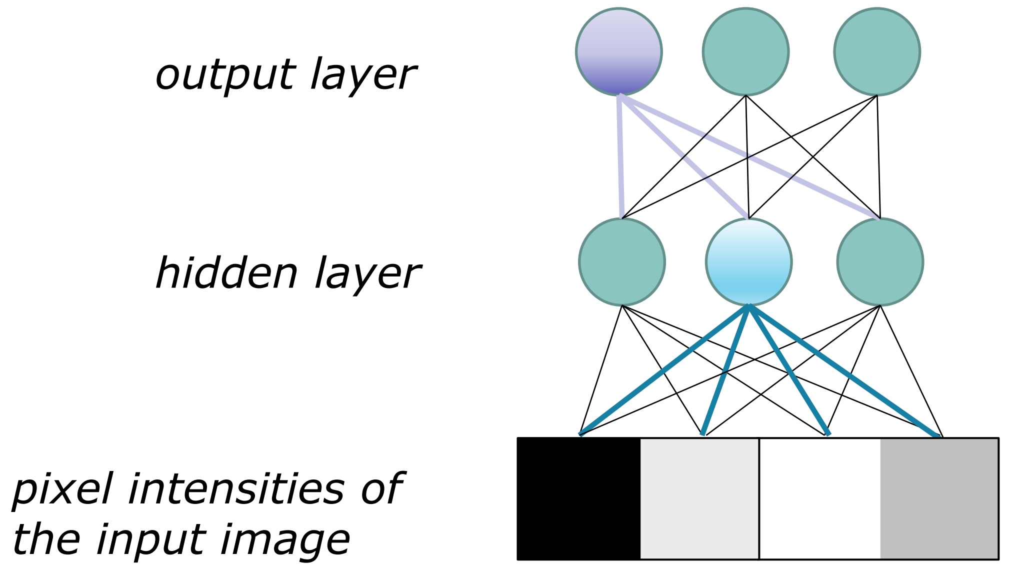


Image Classification by MLPs (3)

- Connections are **global**
  - Each neuron in the first hidden layer considers the *whole* image as input
  - Each neuron of the following layers involves *all* the neurons of the layer below
- **Spatial information** is not explicitly modeled by the MLP architecture
  - The input image is flattened into a vector
  - There are no explicit (a-priori) associations between the hidden/output neurons and specific spatial locations of the image



From MLPs to Convolutional Networks

- MLPs can indeed be used in Image Classification…
  - With the limits we highlighted so far

- …but other classes of Neural Networks are better suited for this task: **Convolutional Neural Networks (CNNs)**
  - They are based on the concept of convolution, that is a well known operation
  - State-of-the art results in Image Classification
  - Extremely popular nowadays

Convolution & Images

Stefano Melacci

Convolution

- Convolution is a well-known mathematical operation
- Given two functions *f* and *g*, convolution computes a new function *(f*g)*, where * is the convolution operator
- Convolution is defined as

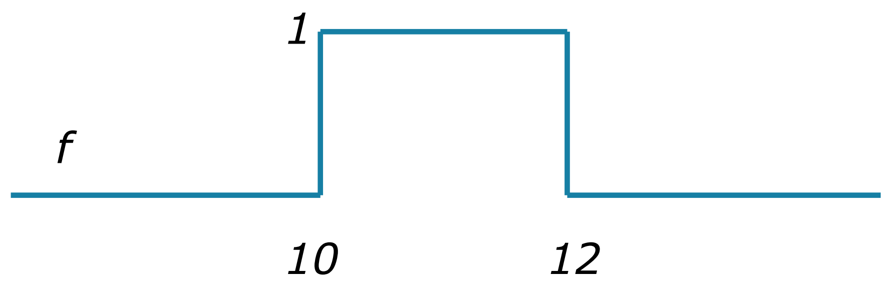
$$(f * g)(x) = \int_{-\infty}^{+\infty} f(t) g(x-t) dt$$

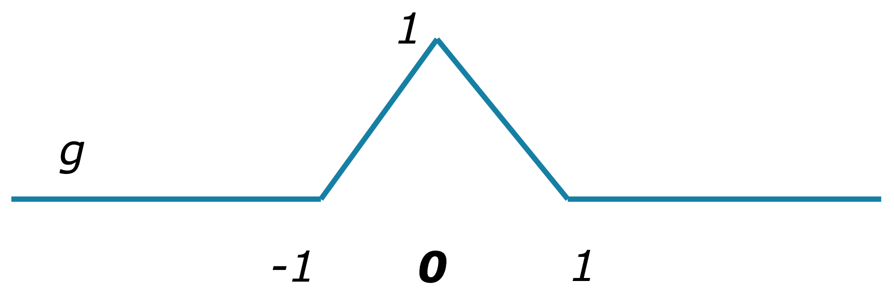
- If we assume to work in a discrete domain (as in the case of images) the **discrete** convolution is defined as

$$(f * g)[x] = \sum_{t=-\infty}^{+\infty} f[t] g[x-t]$$

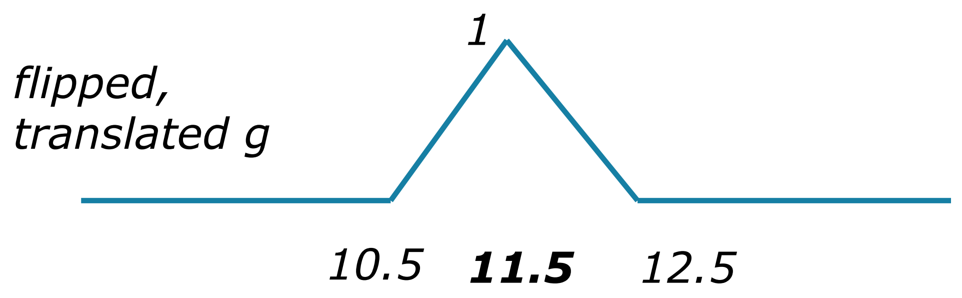
Convolution: 1d Example

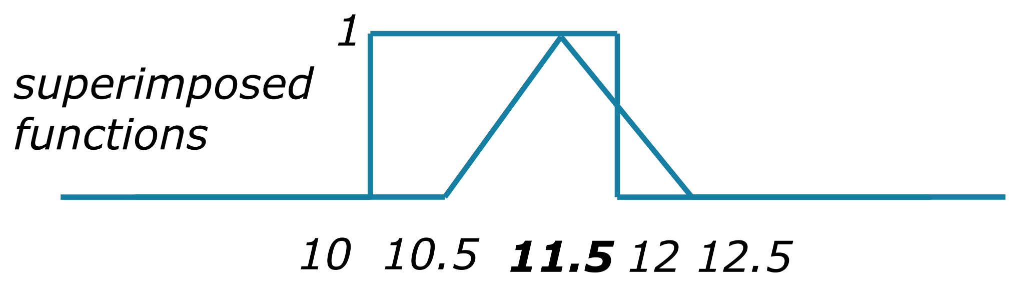
- What is the convolution of the following 1-dimensional *f* and *g* in point *x = 11.5*?
  - We are looking for the value of *(f*g)(11.5)*





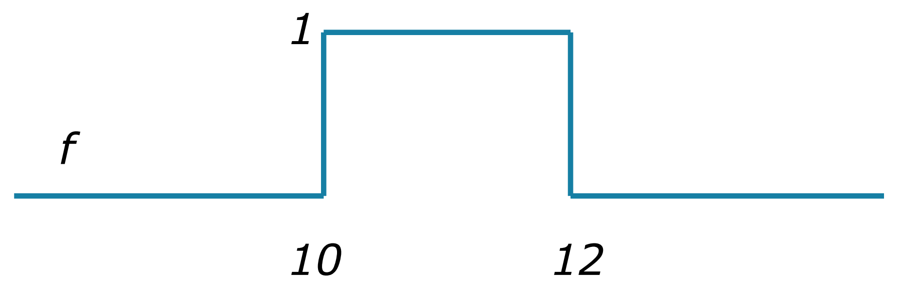
- **Flip** *g* around *x = 0* (in this example nothing changes, due to the symmetry of the function g), **translate** it by 11.5, and **superimpose** the two functions
- Perform the product of the two functions for each *x*, sum (integrate) the results
  - *(f*g)(11.5) = 0.875*

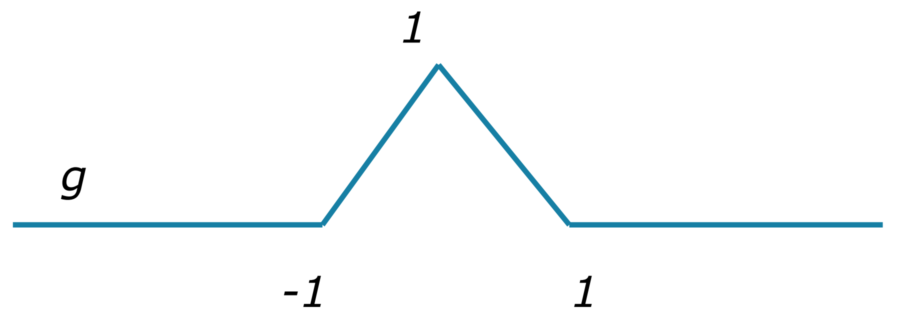


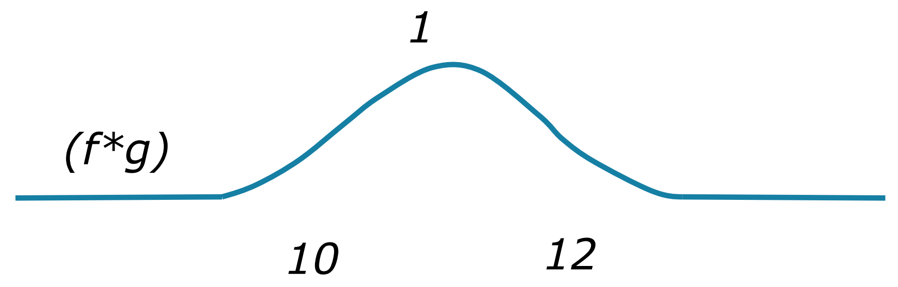


Convolution: 1d Example (2)

- **If we repeat this computation for all *possible values of x in*** *(f*g)(x)*, we can draw the function *(f*g)(x)* that is the outcome of the convolution







(hand drawn sketch)

Convolution and Correlation

- As it will become clearer at a later stage, with the aim of studying Convolutional Neural Networks, it is more straightforward to think about convolution *without the aforementioned "flip" operation on function g*
- This is due to the fact that the flipped-function *g* is **learned from data**
- It is simpler to avoid considering that the system is learning a function that is the flipped instance of another function

- Formally speaking, we should talk about **Correlation** instead of Convolution
- However, the Machine Learning literature/software packages does not always make this distinction explicit, so we will keep using the term Convolution

Convolution and Images

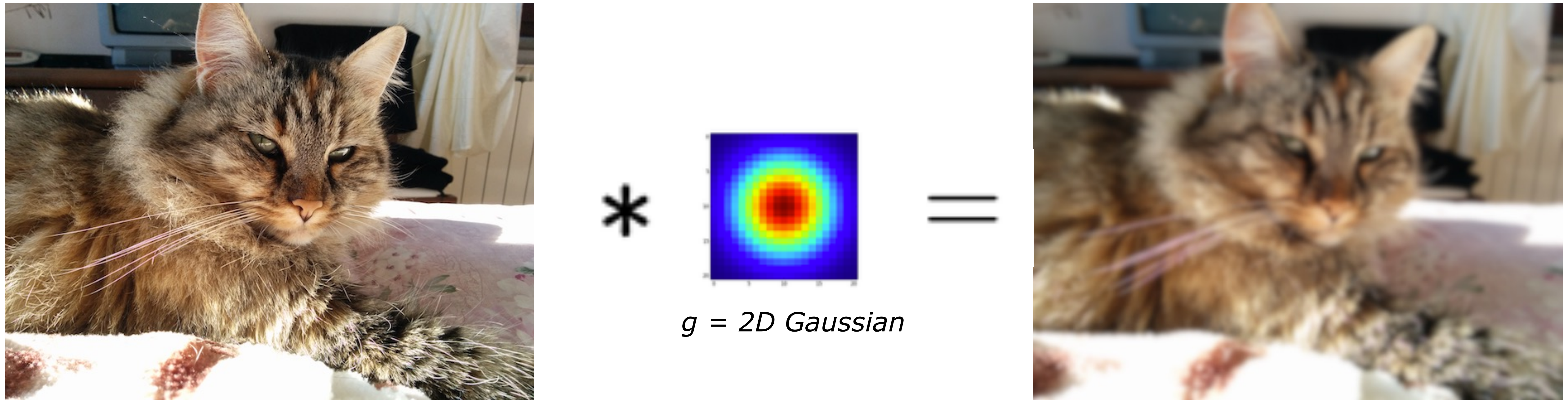
- Let’s consider the case in which we have an **image** I and we want to convolve it with a given **filter/kernel**

- The **image** is a 2-dimensional function, discretized on the pixel coordinates
  - What we called *f* in the previous examples

- The **filter/kernel** is, again, a 2-dimensional function discretized on a number of locations, defining the so called filter/kernel mask
  - The filter/kernel is we called *g* in the previous examples
  - Discrete convolution: we sum over the considered discrete locations
  - We have to perform a **2-dimensional convolution**

Convolution and Images: Gaussian Blur

➢In the case of a **Gaussian filter** *g*, the outcome of the convolution is blurred instance of the input image



g = 2D Gaussian

f = I

*(f*g) = blurred I*

Support & Receptive Fields

- In general, the function *g* has infinite support, but when it is discretized it also usually "cut" to a small region around the origin
  - The size of such region is a custom parameter
  - The shape of such region is also customizable, but it is usually assumed to be **squared**
  - *The filter is assumed to be **zero** outside of the considered region*

- The fact that convolutional filters are assumed to be limited to small areas (let's say, *k-times-k*) and to be zero outside of it, allows us to introduce the notion of **receptive field**
  - Performing the convolution at coordinates *(x,y)* corresponds with considering a **receptive field** of size *k-times-k* centered in *(x,y)*
  - The image pixels outside the receptive fields are **not** considered at all in the convolution

Discrete Convolution in 2D

➢Let us assume that we have a 5x5 grayscale image (pixel intensities in [0,1]) and a convolutional filter, what is the outcome of the convolution (correlation)?

0	1	1	1	0
0	0	1	0	0
0	1	0	1	0
0	1	1	1	0
0	1	1	1	0

f = *I*

$*$

0.2	*0.5*	*1*
-1.5	*2*	*0*
0	*-1*	*-0.2*

g

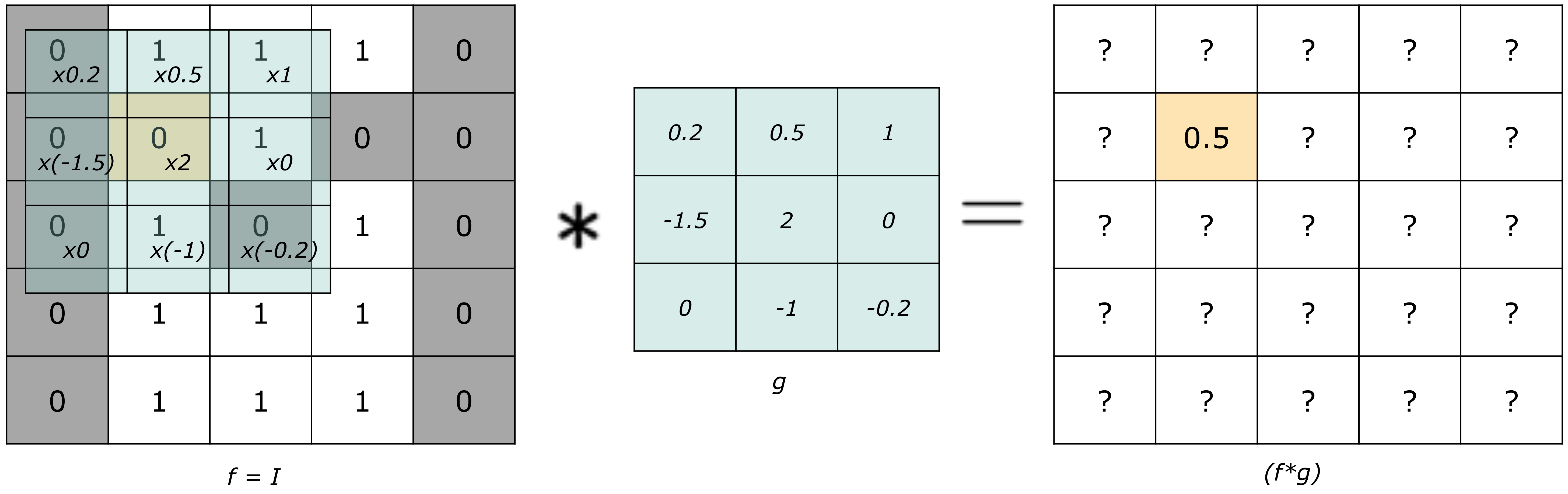
$=$

?	?	?	?	?
?	?	?	?	?
?	?	?	?	?
?	?	?	?	?
?	?	?	?	?

*(f*g)*

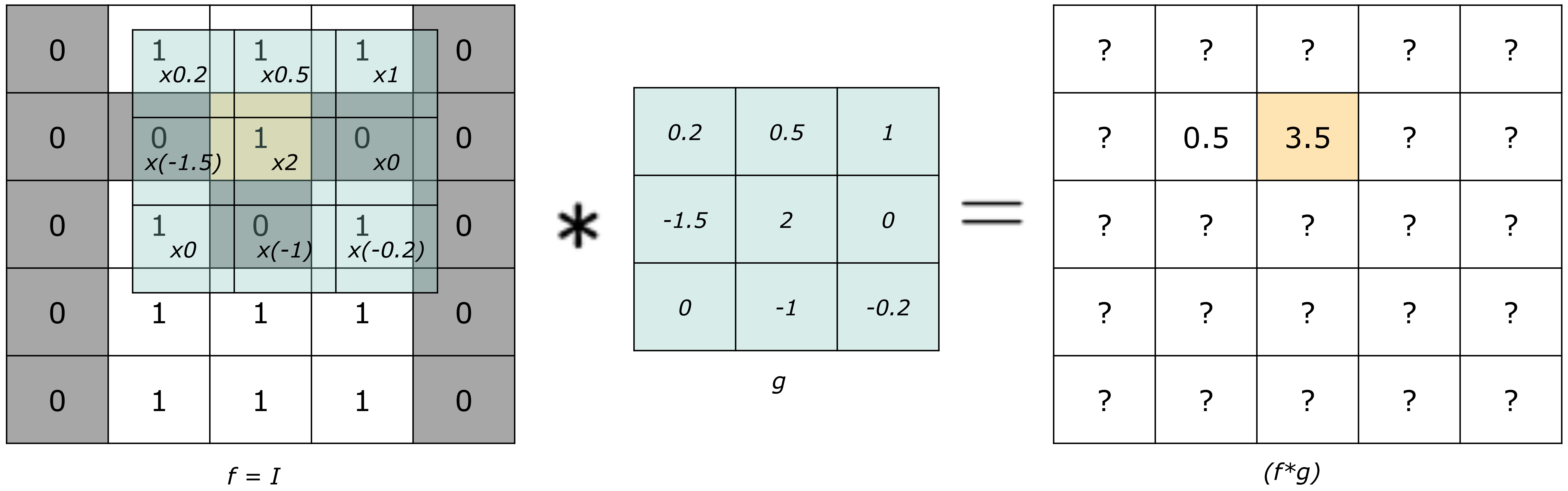
Discrete Convolution in 2D

- Let us assume that we have a 5x5 grayscale image (pixel intensities in [0,1]) and a convolutional filter, what is the outcome of the convolution (correlation)?
  - Let us compute the value of the convolution for some pixel coordinates (orange)



Discrete Convolution in 2D

- Let us assume that we have a 5x5 grayscale image (pixel intensities in [0,1]) and a convolutional filter, what is the outcome of the convolution (correlation)?
  - Let us compute the value of the convolution for some pixel coordinates (orange)



0	1 x0.2	1 x0.5	1 x1	0
0	0 x(-1.5)	1 x2	0 x0	0
0	1 x0	0 x(-1)	1 x(-0.2)	0
0	1	1	1	0
0	1	1	1	0

f = I

0.2	*0.5*	*1*
-1.5	*2*	*0*
0	*-1*	*-0.2*

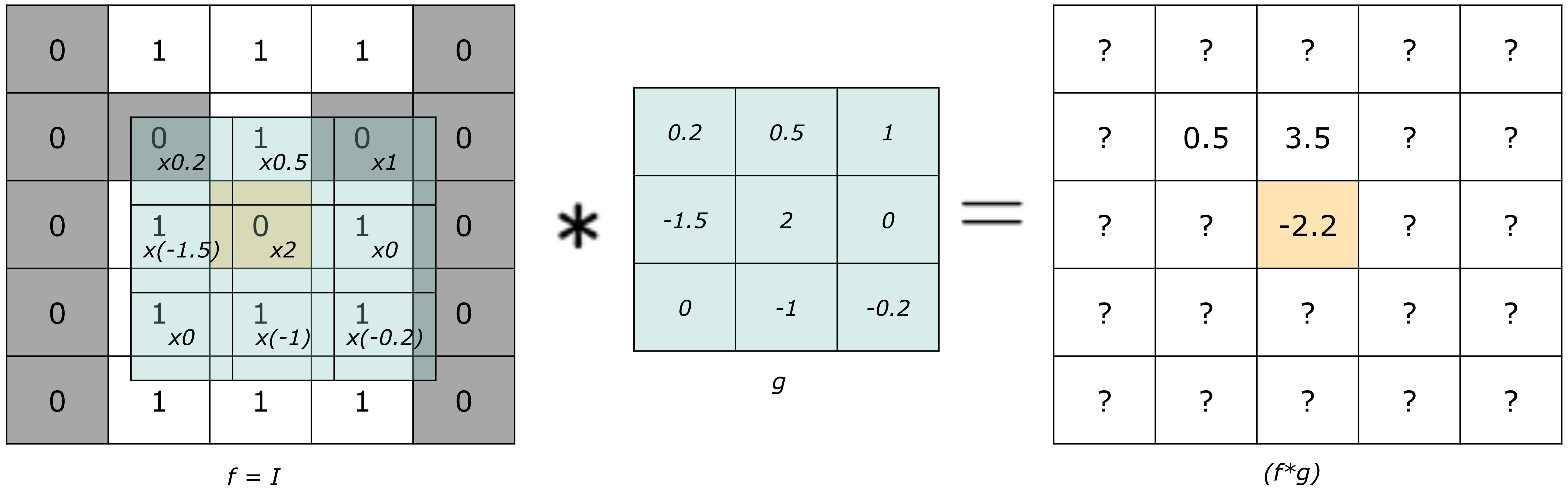
g

?	?	?	?	?
?	0.5	3.5	?	?
?	?	?	?	?
?	?	?	?	?
?	?	?	?	?

*(f*g)*

Discrete Convolution in 2D

- Let us assume that we have a 5x5 grayscale image (pixel intensities in [0,1]) and a convolutional filter, what is the outcome of the convolution (correlation)?
  - Let us compute the value of the convolution for some pixel coordinates (orange)



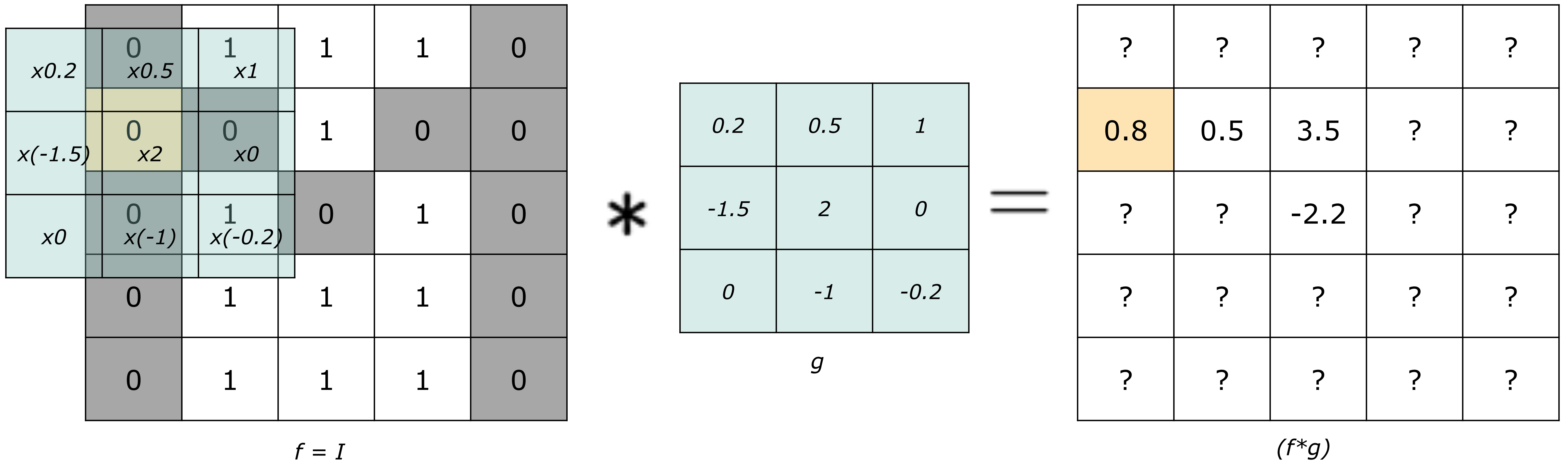
Discrete Convolution in 2D

- Computing the convolution at those coordinates that are at the borders of the image would require to access pixel values that are out-of-the-image
  - Border effects

- **Avoid computing convolution at such coordinates**
  - In this case *(f*g)* will be smaller than *I*
- **Zero padding**: out-of-bound coordinate corresponds to zero-valued pixels
  - The most common (and simpler) case
- **Mirroring**: the image is *mirrored* at the borders
- **Repetition**: the image is *repeated* at the borders

Discrete Convolution in 2D

- The case of zero padding



0	1	1	1	0
0	0	1	0	0
0	1	0	1	0
0	1	1	1	0
0	1	1	1	0

0.2	0.5	1
-1.5	2	0
0	-1	-0.2

?	?	?	?	?
0.8	0.5	3.5	?	?
?	?	-2.2	?	?
?	?	?	?	?
?	?	?	?	?

Convolution and Feature Maps

- The outcome of the convolution, for all the pixel coordinates, yields a 2D matrix
  - Each score in the matrix is associated to a certain coordinate pair *(x,y)*
  - Informally, the score at *(x,y)* represents "the strength" of the response to filter *g* at image coordinates *(x,y)*
- We will also refer to the *(f*g)* matrix as **Convolved Feature Map**, or more simply, **Feature Map**, that in the case of the previous example is:

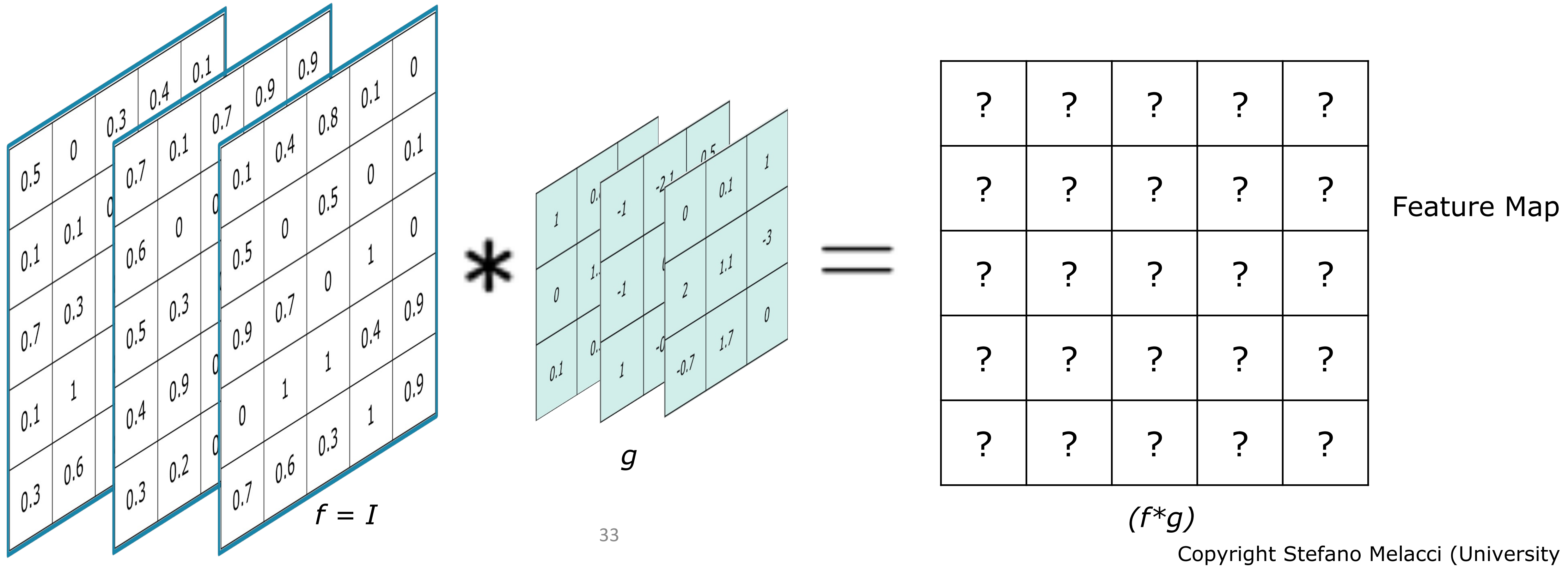
Feature Map

*(f*g)*

0	1.8	-0.5	0.5	-1.5
0.8	0.5	3.5	-1.8	0.2
-0.2	1.8	-2.2	1.2	-1.5
0.8	1.3	0.5	0	-1.3
1	3.5	2.2	1.2	-1.3

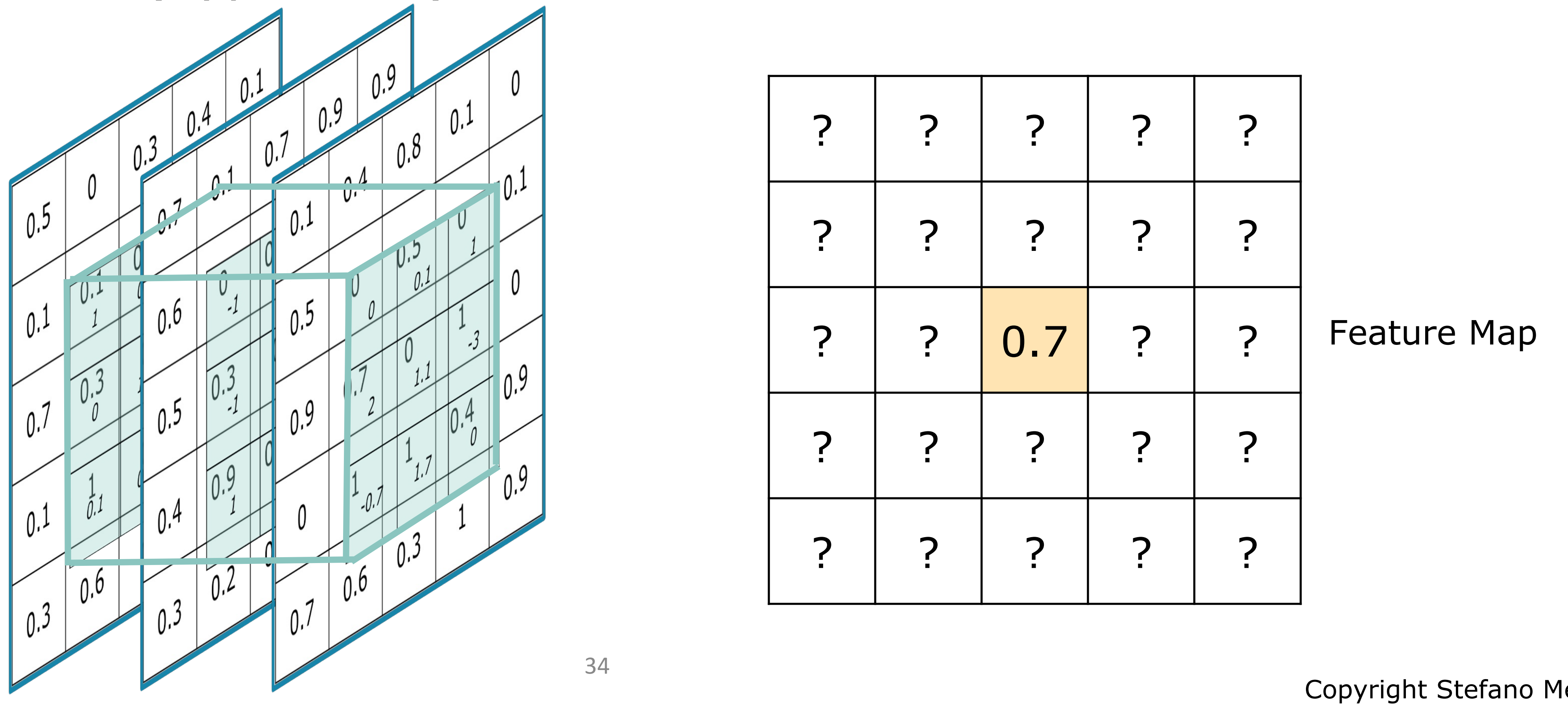
Convolution and Feature Maps (2)

- Up to this point, we mostly considered single-channel images
- Now let us focus on an RGB image (3 channels)
  - The previously considered image matrix is now a 3D tensor
- We can design 3D convolutional filters (3D tensors), and apply convolution
  - If we do not add paddings along the new dimension, we still end up with a 2D feature map



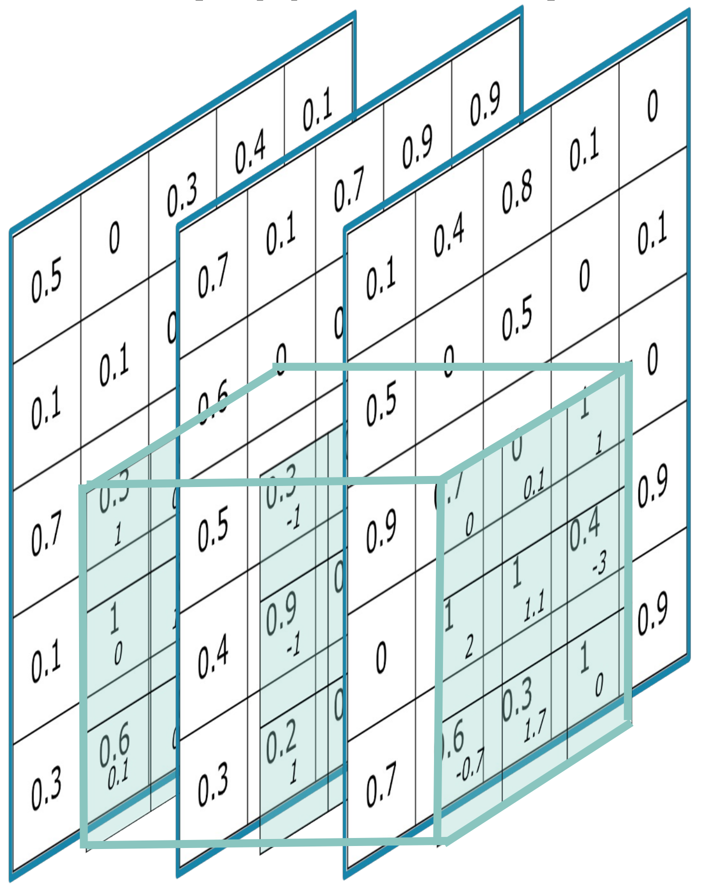
Convolution and Feature Maps (3)

- Performing the 2D convolution on 3D data corresponds with swiping the filter hypercube over the input image, performing element-wise products (between the input image and the filter coefficients) and summing up the result
  - The following picture shows the portion of image (receptive field) involved in computing the feature map element that is highlighted in orange
  - The convolutional filter (hypercube) is centered on the coordinates of the considered element



Convolution and Feature Maps (4)

- Performing the 2D convolution on 3D data corresponds with swiping the filter hypercube over the input image, performing element-wise products (between the input image and the filter coefficients) and summing up the result
  - The following picture shows the portion of image (receptive field) involved in computing the feature map element that is highlighted in orange
  - The convolutional filter (hypercube) is centered on the coordinates of the considered element



?	?	?	?	?
?	?	?	?	?
?	?	0.7	?	?
?	?	0.5	?	?
?	?	?	?	?

Feature Map

Convolutional Neural Networks

Stefano Melacci

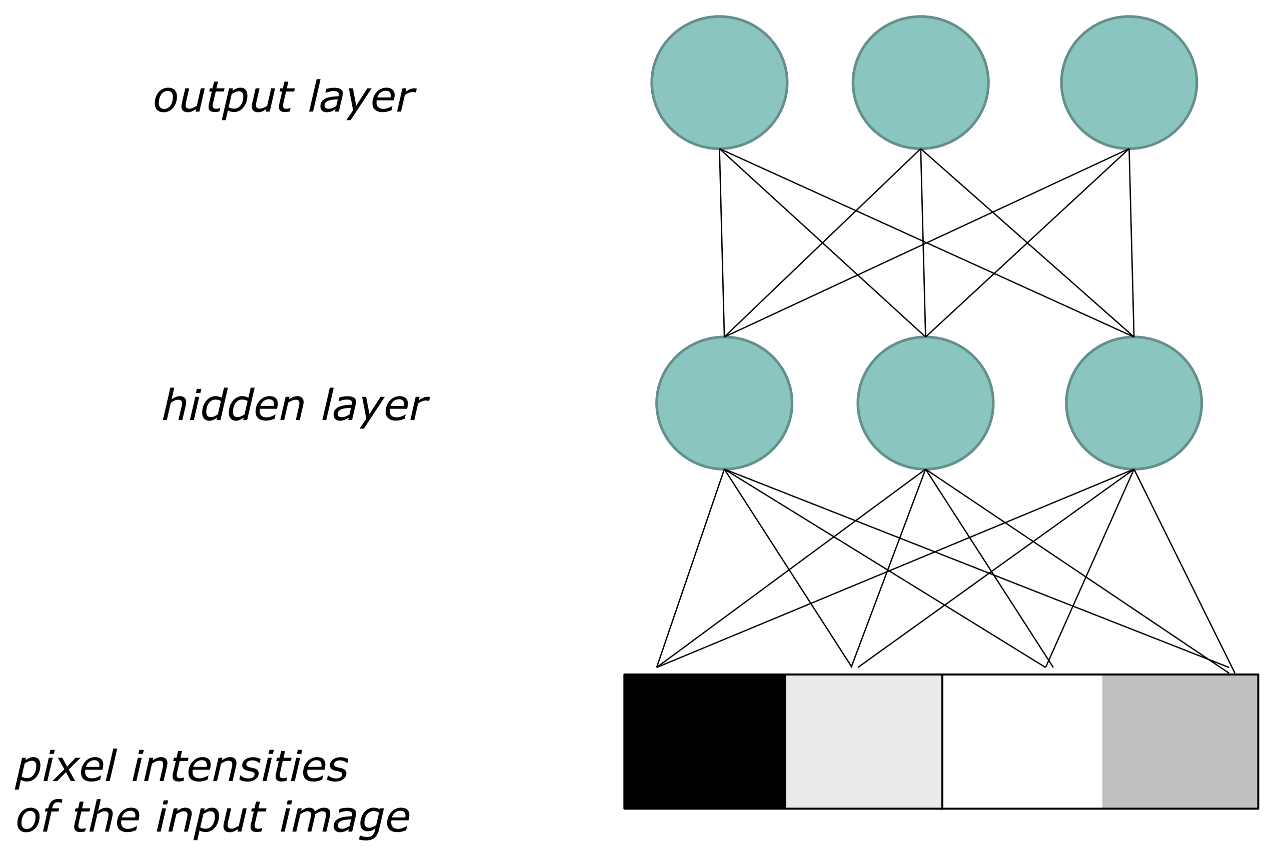
Intro to Convolutional Neural Networks

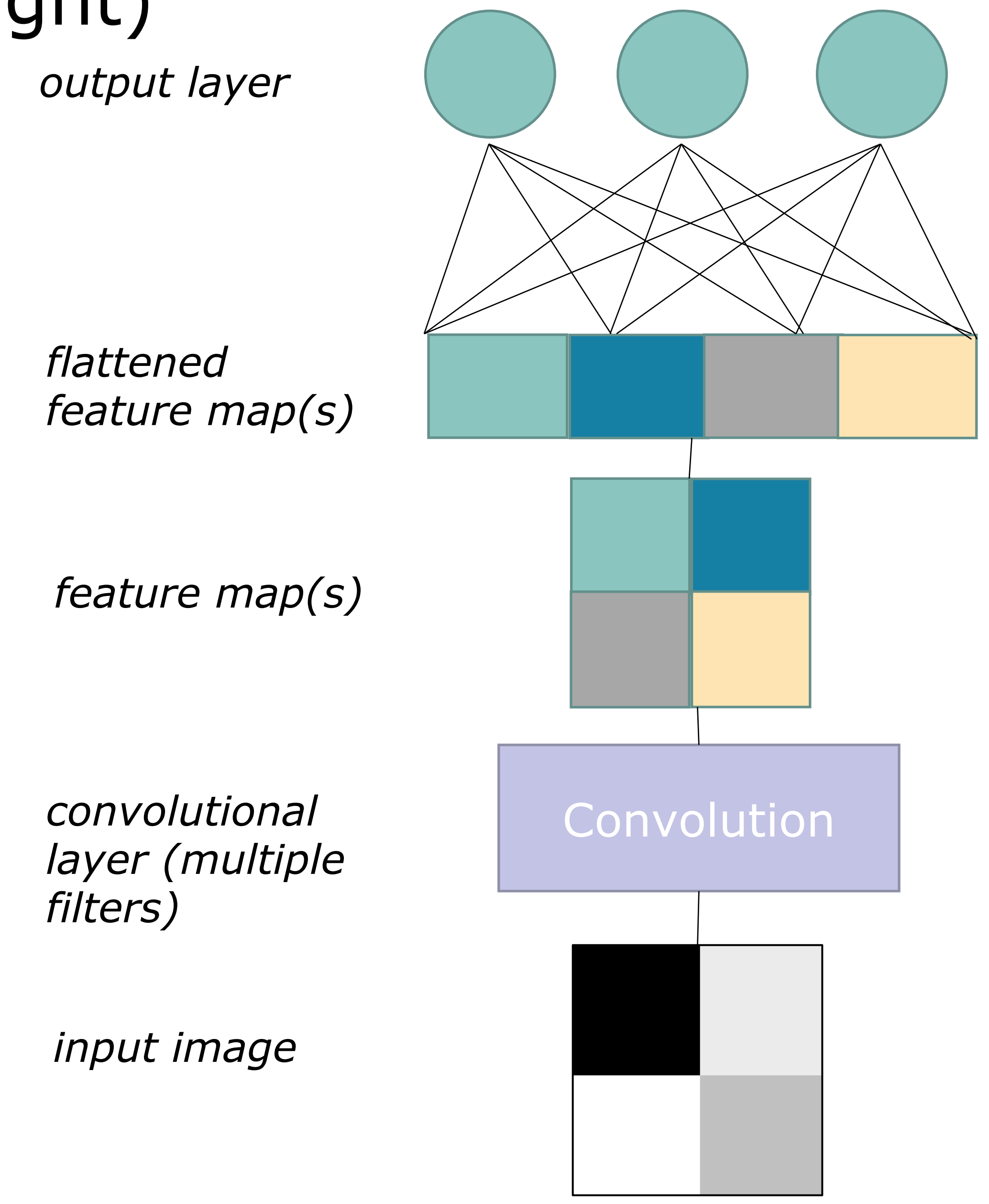
- Convolutional Neural Networks (CNNs) are a special class of Neural Networks that are based on the idea of stacking multiple **convolutional layers**
- A convolutional layer is composed by multiple convolutional filters
- Each layer outputs a certain number of feature maps (the outcome of convolving the input with the filters)
  - Then, they become the input of the next convolutional layer

- Differently from MLPs, each layer performs **local operations**, preserving the **spatial organization** of the data

From MLPs to Convolutional Neural Networks

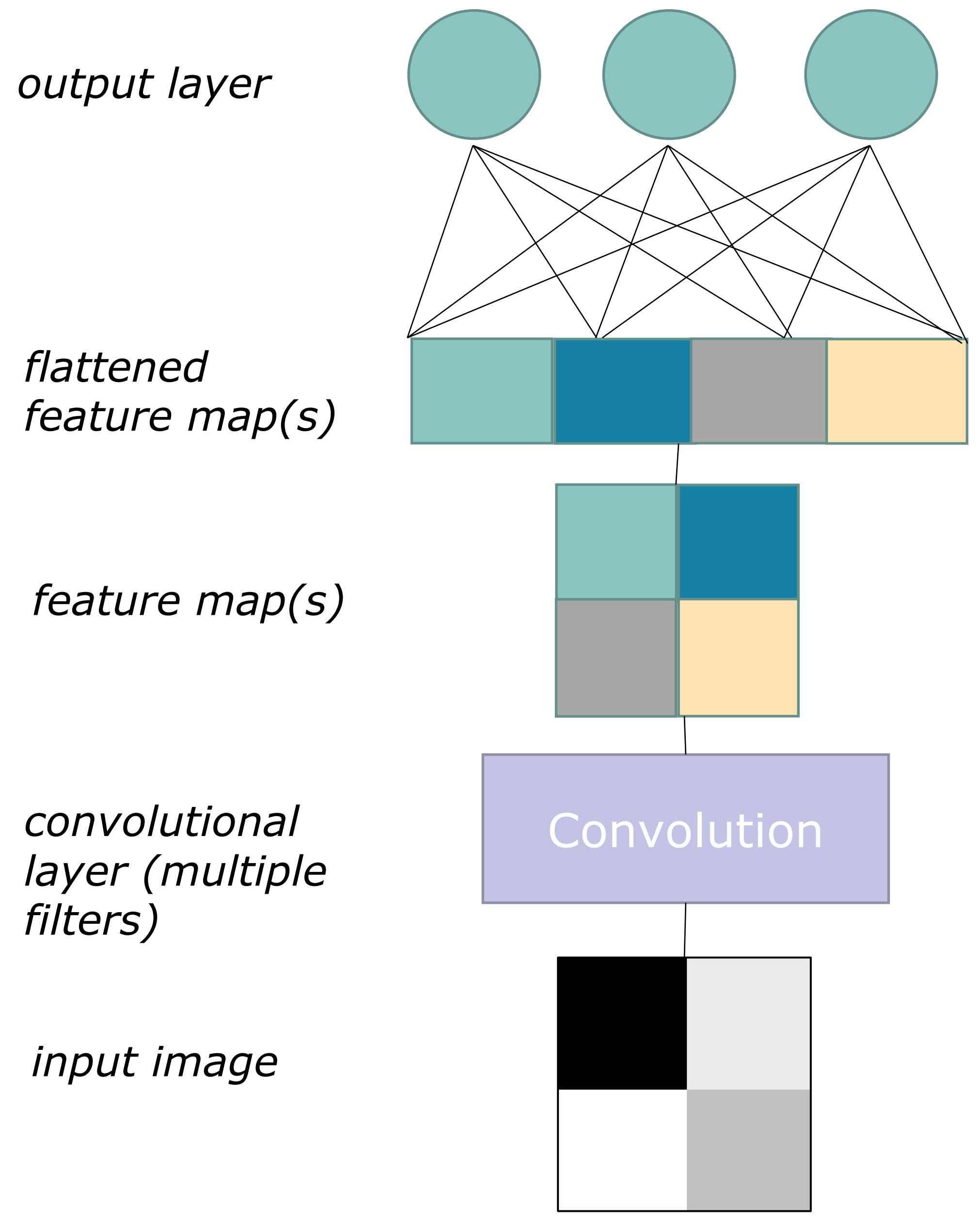
- Basic example of MLP (left) vs CNN (right)




output layer
flattened feature map(s)
feature map(s)
convolutional layer (multiple filters)
Convolution
input image

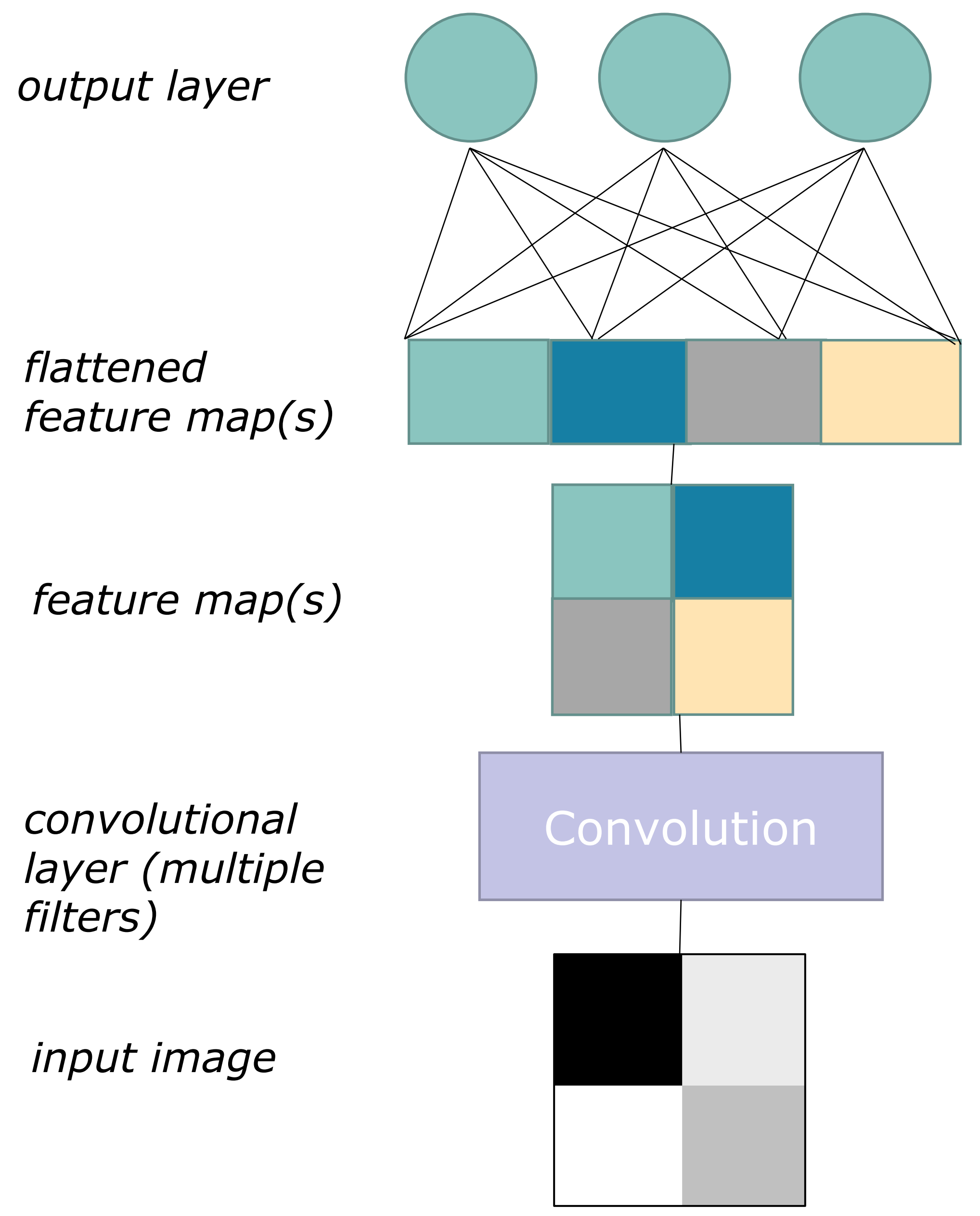
From MLPs to Convolutional Neural Networks

- The input image is **not** flattened into a vector
- The convolutional layer involves a ***k* filters of size n-by-n**, and it outputs *k* feature maps
  - *The size and the number of filters are user defined*
  - Multiple convolutional layers can be stacked (only one is shown in the picture, for simplicity)
  - The input of the next convolutional layer is composed by the feature maps generated by the layer below
- The **coefficients** of the filters are weights of the network, and they **are learned from data**
  - Backpropagation



From MLPs to Convolutional Neural Networks

- The feature maps of the last convolutional layer are flattened into a vector
- A neural network (MLP) processes the flattened representation
  - An single-layer network (fully connected layer) is shown in the picture, for simplicity

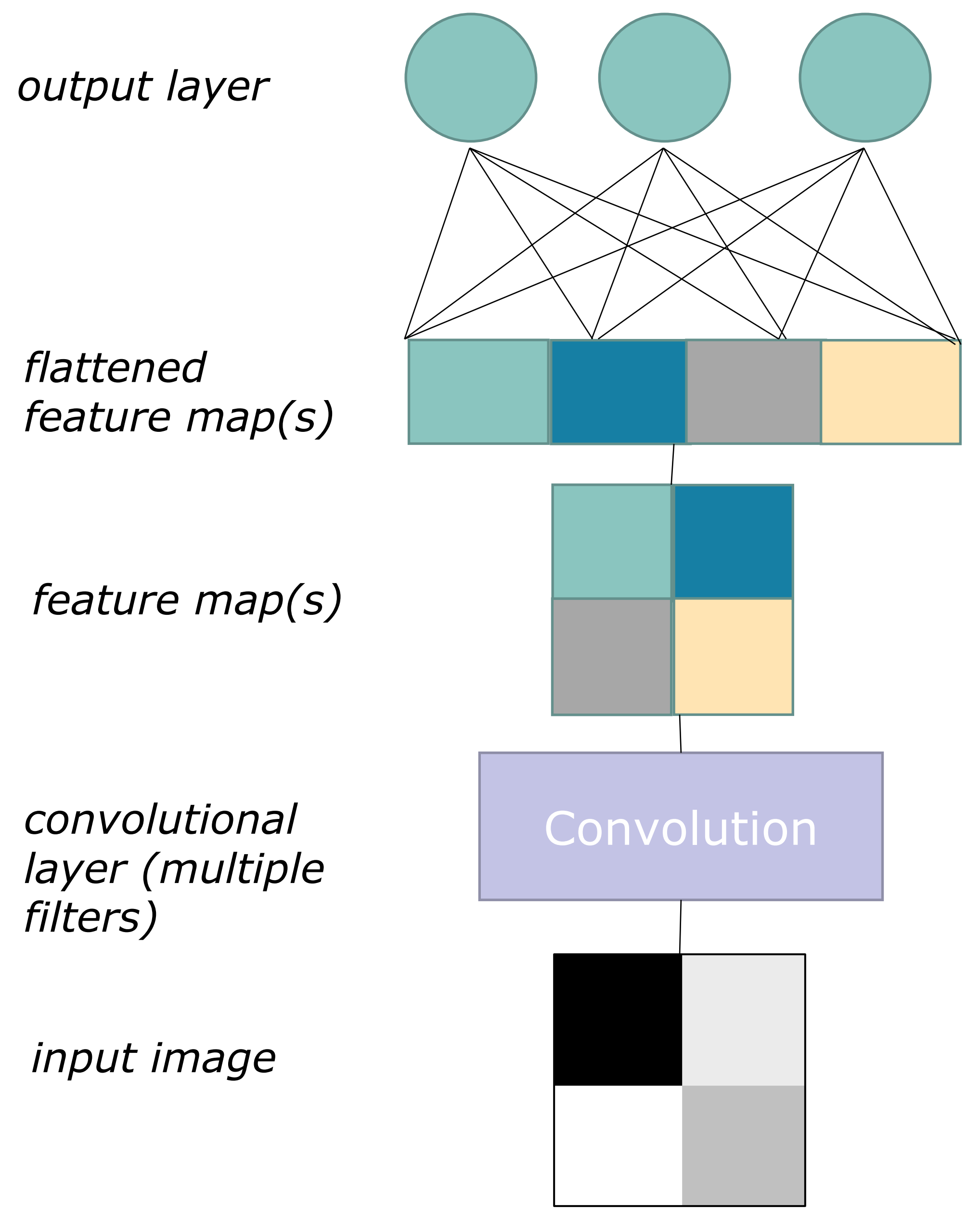


Convolutional Layer

- In order to “visualize” convolutional filters in the context of a neural layer composed of a number of neurons, just think that each element of a feature map is basically a neuron of the convolutional layer

- As already mentioned, the **coefficients** of the filter are weights of the network, and they are **the same for all the elements** of the feature map (by definition of convolution)

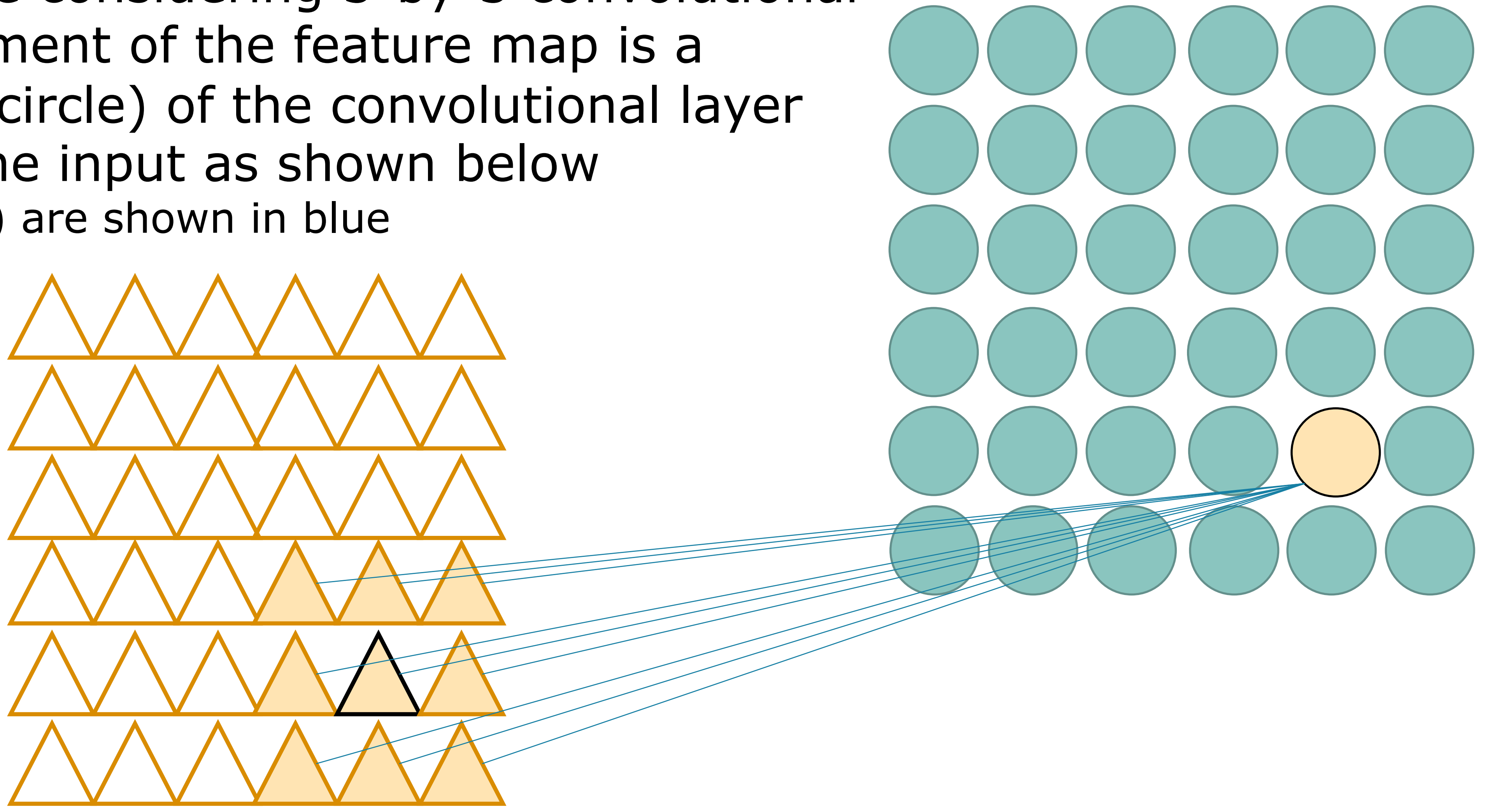
- In other words, for each filter, weights are **shared** among all the neurons of the convolutional layer



output layer
flattened feature map(s)
feature map(s)
convolutional layer (multiple filters)
Convolution
input image

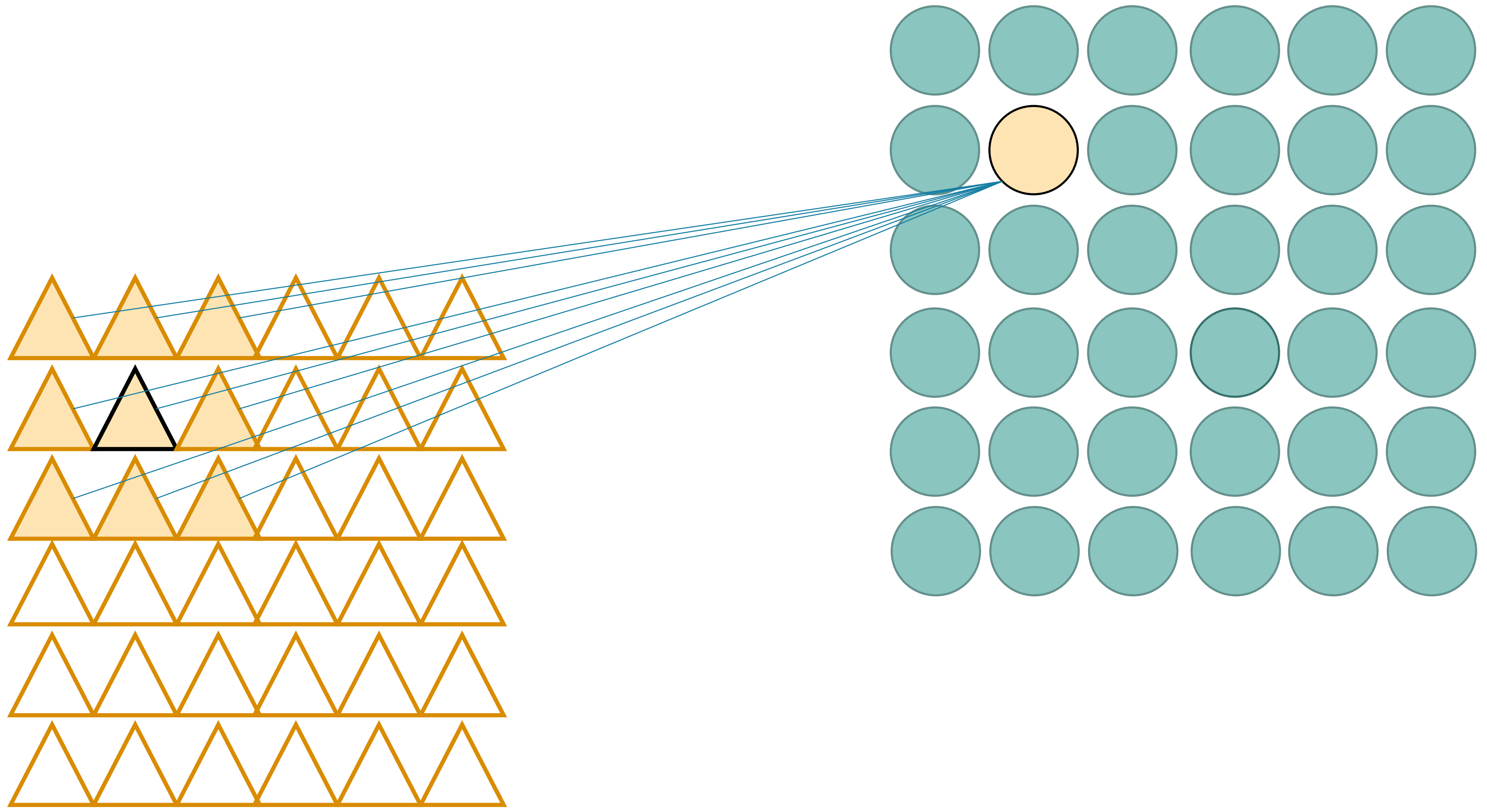
Convolutional Layer (2)

- Example: a single channel (grayscale) input image (orange triangles) is shown
  - Suppose we are considering 3-by-3 convolutional filter: each element of the feature map is a neuron (green circle) of the convolutional layer connected to the input as shown below
    - Weights (filter) are shown in blue



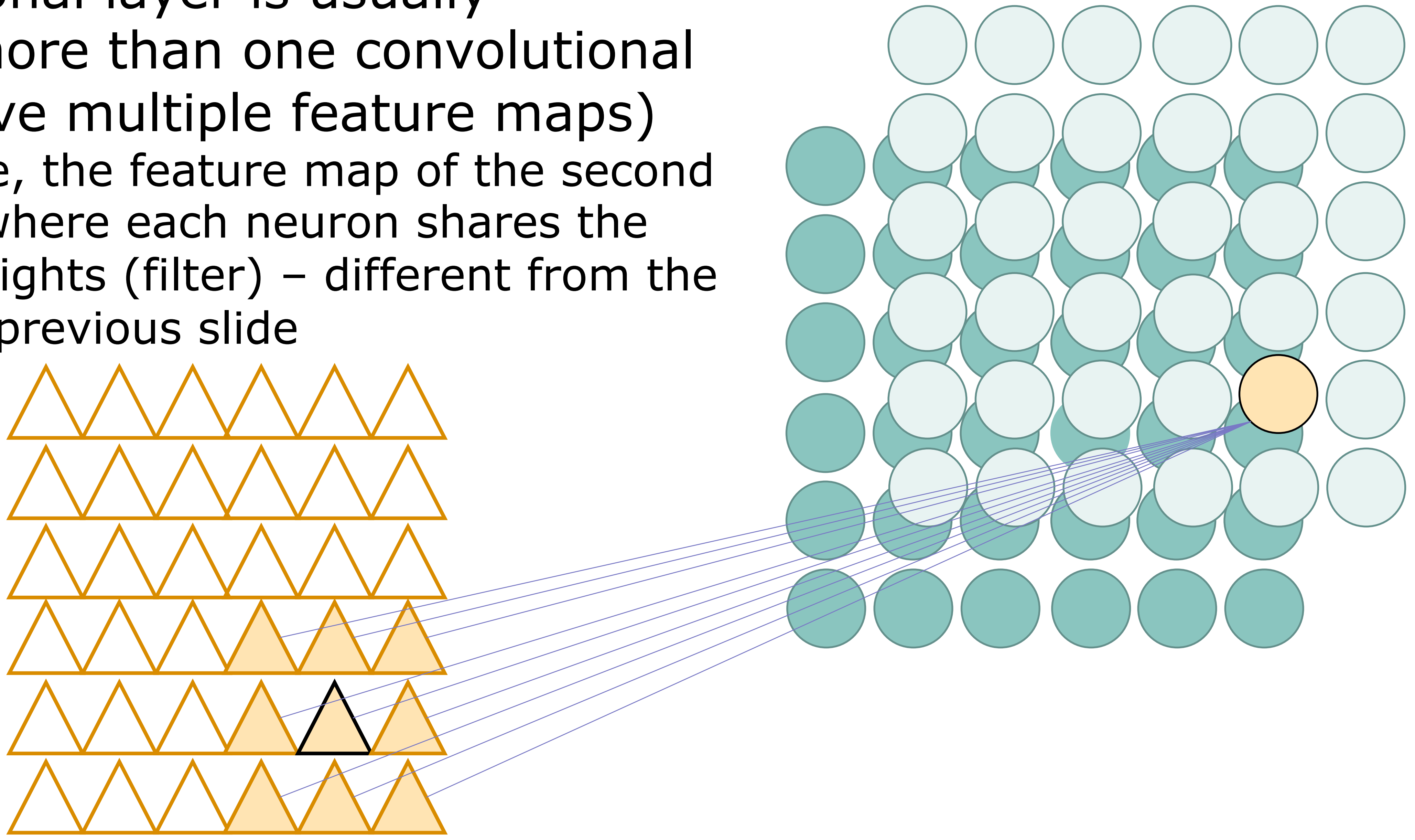
Convolutional Layer (3)

- All the neurons share the same set of weights (the same filter)



Convolutional Layer (4)

- Each convolutional layer is usually composed by more than one convolutional filter (so we have multiple feature maps)
  - In this example, the feature map of the second filter is show, where each neuron shares the same set of weights (filter) – different from the weights of the previous slide



Convolutional Neural Networks – History

- The ideas behind Convolutional Neural Networks (CNNs) can be traced back to the late 80s, and they are due to Yann LeCun (Facebook AI Research)

- A strongly cited paper (20k citations) that collects experiment on Convolutional Nets applied to handwritten recognition is the following one
  - *LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.*

- CNNs contributed to make Deep Learning popular, pushing the idea of multiple computational layers that contribute to create informative representation of the input data

Convolutional Neural Networks – History (2)

- CNNs are strongly based on the concept of **convolutional filter**
- This implies that, due to the limited support of the filters, CNNs also follow the idea of dealing with **receptive fields**

- The idea of connecting units to local receptive fields must be traced back to Hubel and Wiesel (1962)
  - *Hubel and Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. The Journal of physiology 1962*
- Such idea was implemented in an even older computational model: Fukushima's *Neocognitron* (70s-80s)
  - *Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. Neural networks 1988*

A Closer Look at CNNs - Nonlinearity

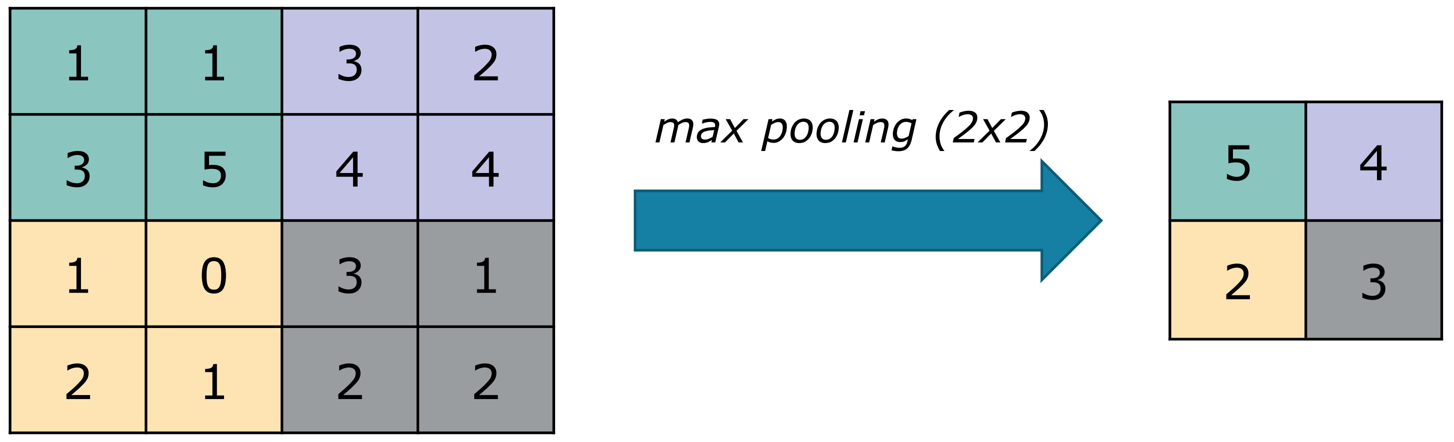
- After each convolutional layer, most common CNNs involve a **non-linear activation function**
- The activation function is applied to each element of the feature maps
  - In the case of rectifiers (ReLu), we have

$$o^{\ell}_{ijk} = \max(a^{\ell}_{ijk}, 0)$$

  - where a_{ijk} is a the *ij*-th element of the *k*-th feature map, and the superscript *l* is the layer index

A Closer Look at CNNs - Pooling

- **Pooling layers** are usually introduced after every convolutional layer (+ nonlinearity)
- Pooling is a way to aggregate feature map elements that are spatially close
  - *Max pooling*: Given a group of nearby coordinates, take the max value
  - *Average pooling*: Given a group of nearby coordinates, compute the average value
- It is usually applied to non-overlapping squared blocks of elements (the size of the blocks is a design choice)
  - It leads to a reduction of the size of the feature map (subsampling)



feature map

A Closer Look at CNNs – Strided Convolutions

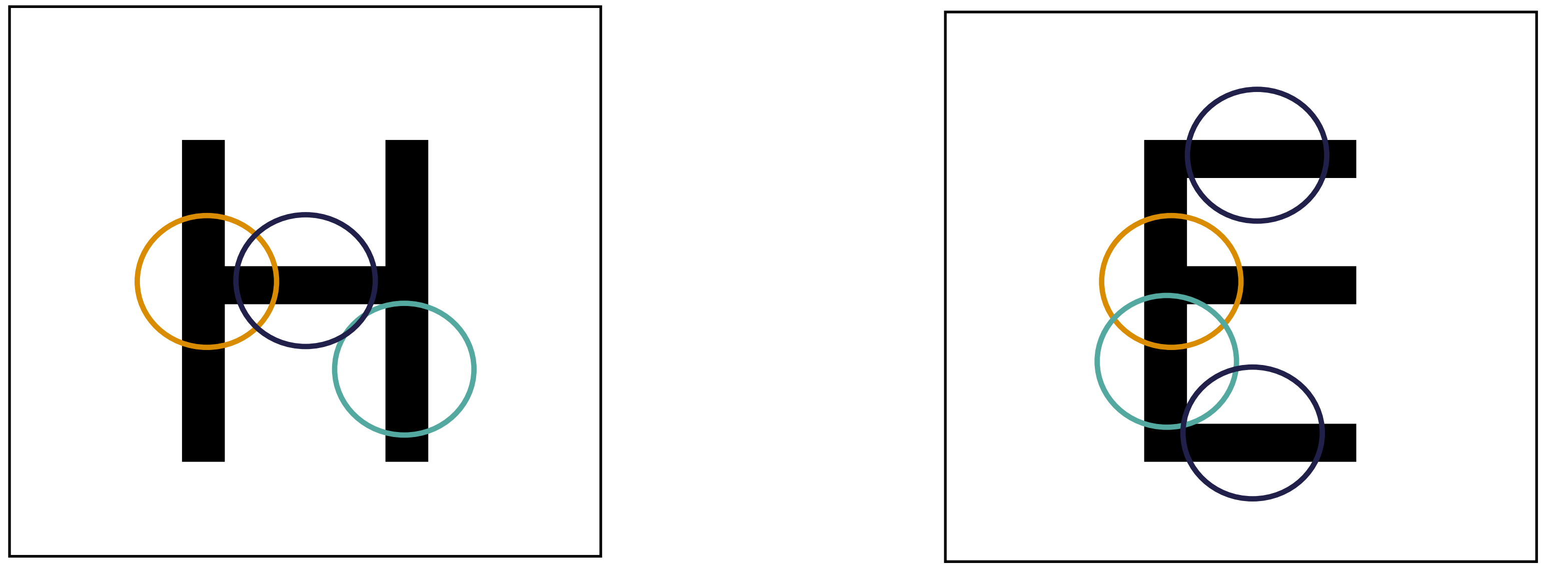
- Another customizable parameter that is commonly involved in the design of CNNs is the **stride** of the convolutions
- Instead of computing convolution on every coordinate of the input, the **stride** parameter defines the spacing (on each axis) between consecutive coordinates on which convolution is performed
  - Stride equal to 1 implies performing convolution on every coordinate
  - When the stride is greater than 1, the operations on the convolutional layer are **less costly**, since some convolutions are not performed
  - The size of the output feature map is also smaller, thus **saving memory**
  - *However, some information might be lost due to the skipped coordinates*

Image Classification with Convolutional Neural Networks

Stefano Melacci

Going Back To Image Classification

- CNNs progressively compute a representation of the input image that can be easily classified
- Images have local properties that might be repeated at different locations and in different images
  - Think about straight lines and corners, or also object parts and objects
  - The **same** convolutional filter can produce high responses in **different** locations



Circles with the same color = same convolutional filter

Going Back To Image Classification (2)

- In the case of real-world images, you can think of "similar" objects in the same of different pictures

Circles with the same color = same convolutional filter

Going Back To Image Classification (3)

- However, we do not want to pre-design filters that are able to respond in image regions that we think are important
  - It is not trivial to say what is important or not in a generic sense!
  - For example, what are the important objects? What are the important regions?

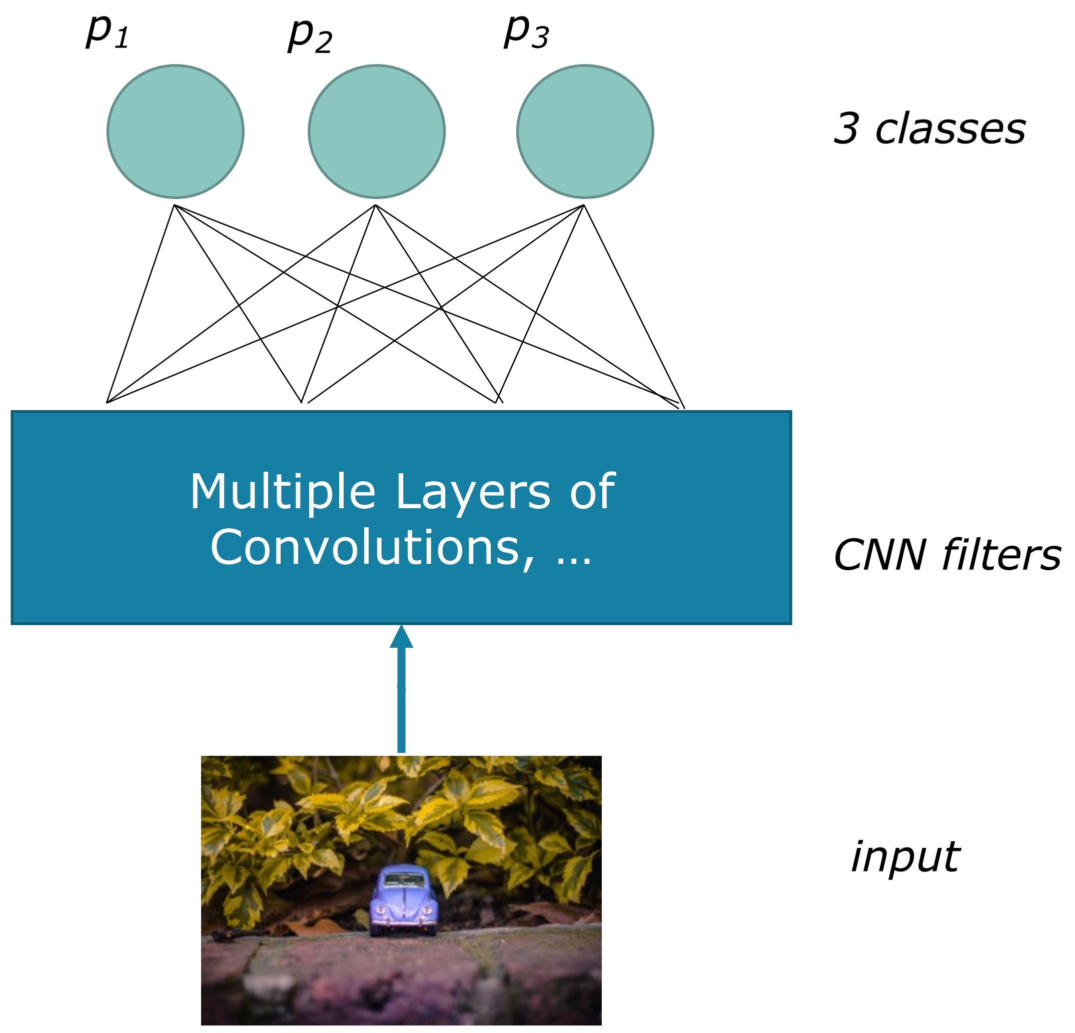
- *We aim at learning the coefficients of a bank of filters in function of the task that we want the network to solve*
  - In this case, we focus on **Image Classification**

Going Back To Image Classification (3)

- In order to approach Image Classification over *c* classes, the output layer of the CNN must be composed of *c* neurons
  - Softmax activation is usually applied to the neurons of the output layer
  - Each output neuron models the probability of belonging to one of the *c* classes

- And, of course, we need to have the use of a large collection of labeled (supervised) data
  - We focus on the case of supervised learning

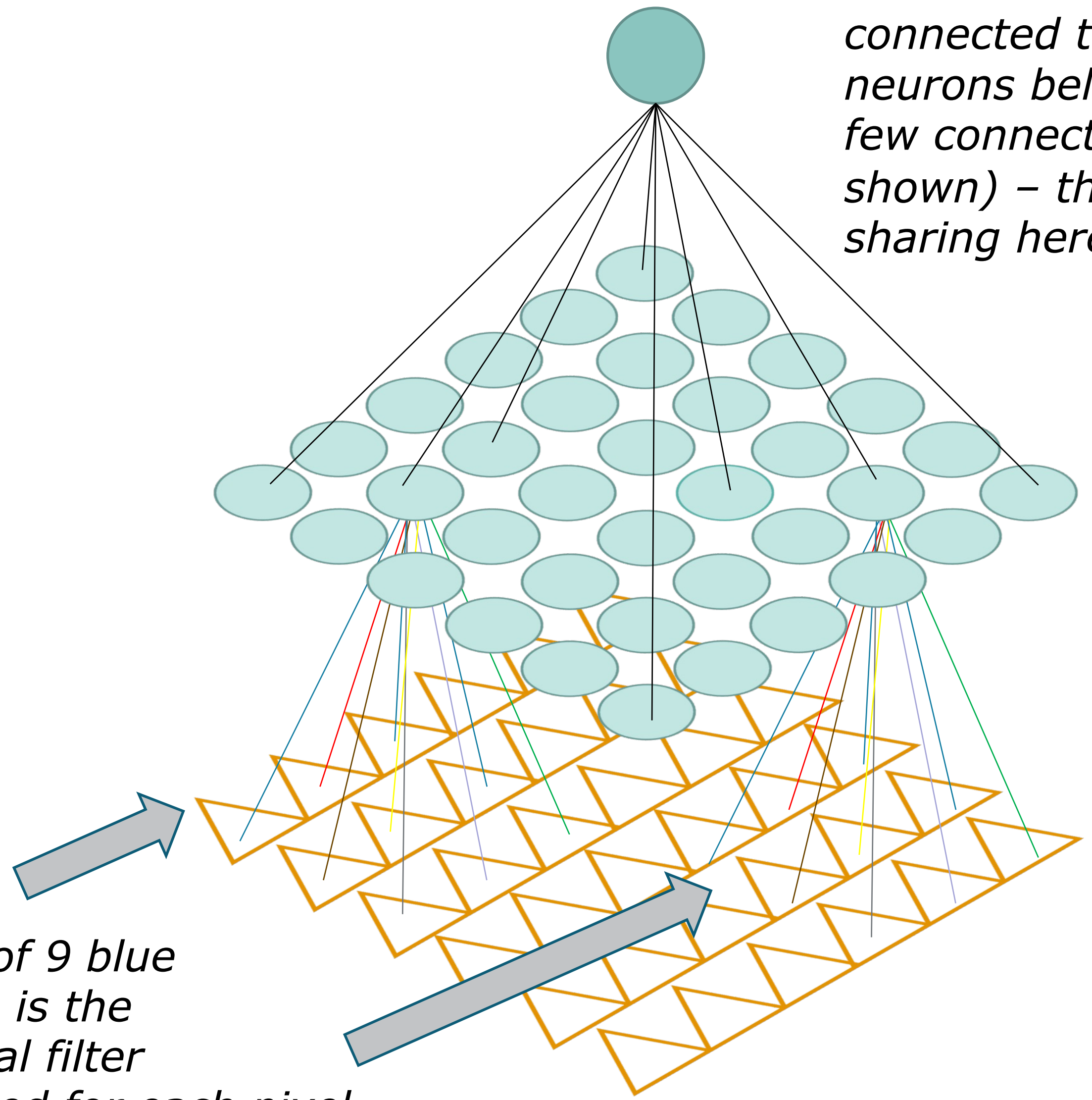
- *Training the network ➔ Learning the coefficients of the convolutional filters*



CNN: Image Classification & Backprop

- Once a loss function is introduced to measure the mismatch between the output of the network and the class labels of the training set images, the whole CNN is trained by **Backpropagation**
  - The convolutional filters and the weights of the fully connected layers are the parameters that are optimized by Backpropagation

- On the right, a convolutional network is depicted in an extended way
  - Toy example: 1 convolutional layer with a single filter and a fully connected layer with 1 output neuron (binary classification)
  - *It is nothing more than a feed-forward network with some **shared** connections (same color = shared weight)*



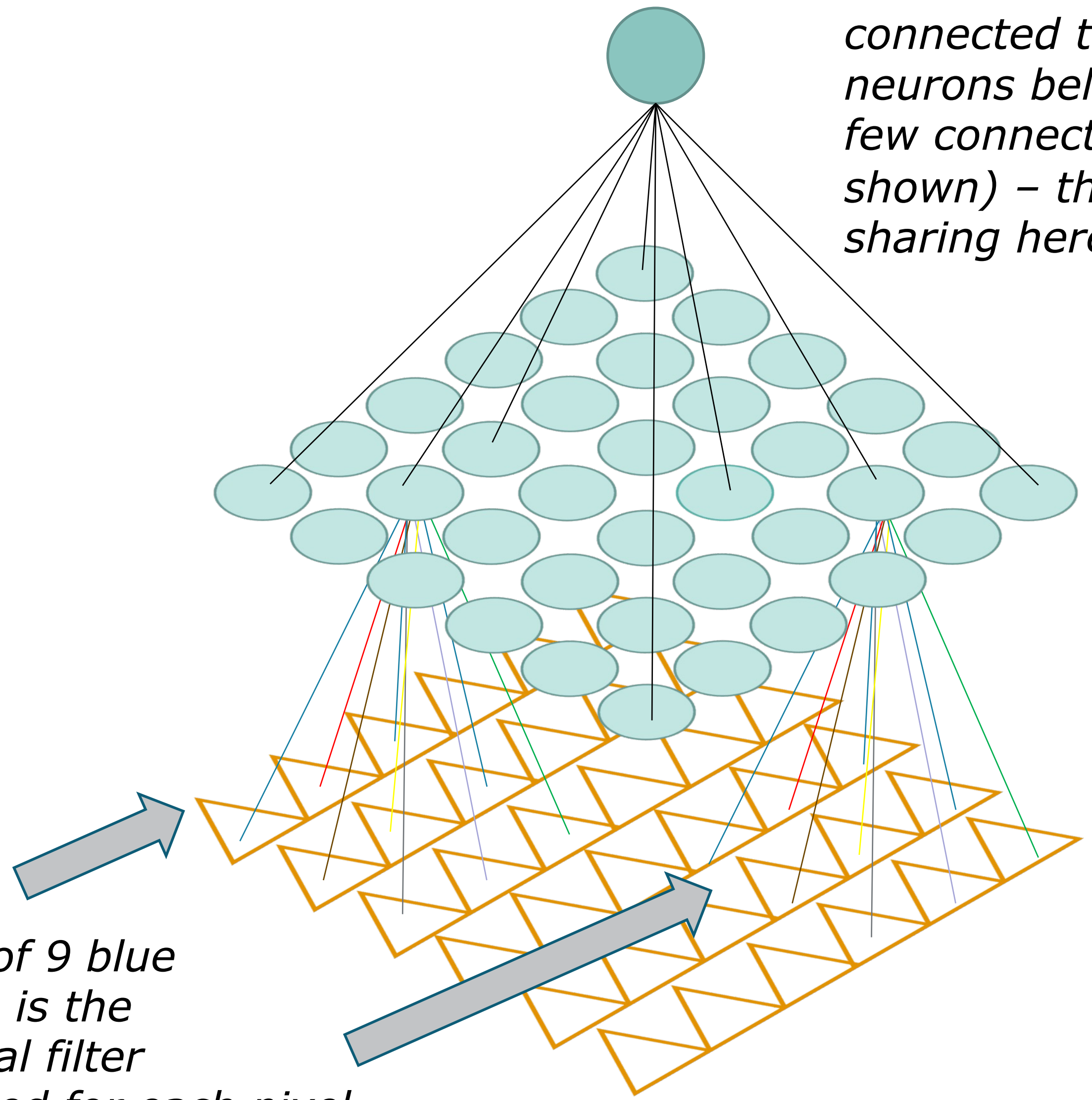
This neuron is connected to all the neurons below (only a few connections are shown) – there is no sharing here!

This group of 9 blue connections is the convolutional filter (it is repeated for each pixel of the input image, even if only two repetitions are shown) - same color = shared weight

CNN: Image Classification & Backprop (2)

- The derivatives with respect to the **shared** connections are computed as in the classic Backpropagation algorithm, and then they are accumulated (summed up)

- This means that the final gradient with respect to the filter components is the outcome of accumulating the gradients coming from filtering each pixel of the input of the convolutional layer

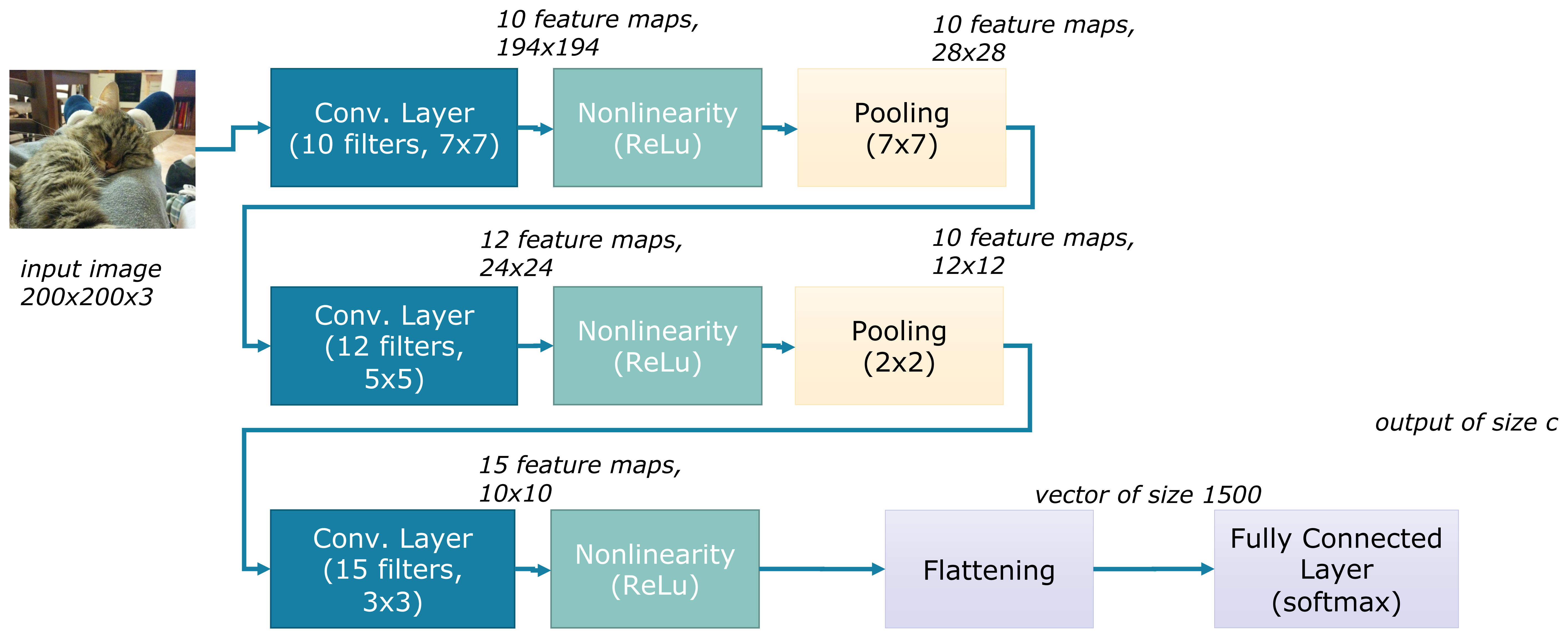


This neuron is connected to all the neurons below (only a few connections are shown) – there is no sharing here!

This group of 9 blue connections is the convolutional filter (it is repeated for each pixel of the input image, even if only two repetitions are shown) - same color = shared weight

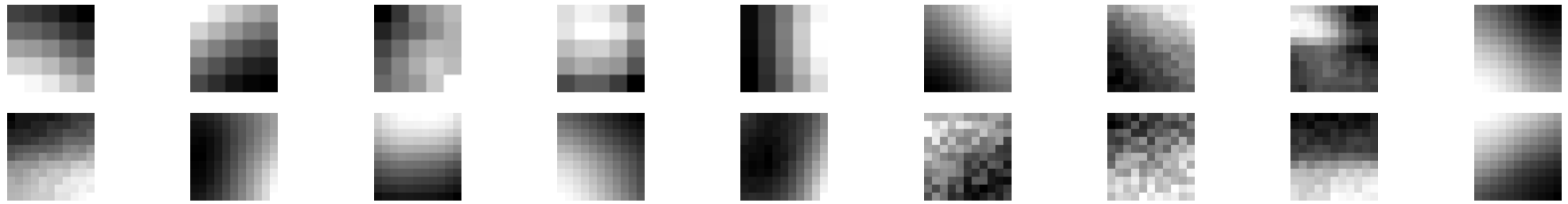
CNN: Image Classification

- In the following picture, a sketch of a CNN in *c*-class Image Classification is reported



CNN: Image Classification (2)

- Each convolutional layer learns a set of features that are **local** with respect to the **current input**



Examples of learned filters (dark = small; bright = large)

- However, due to the layer-wise organization, each input is, in turn, the output of another convolutional layer

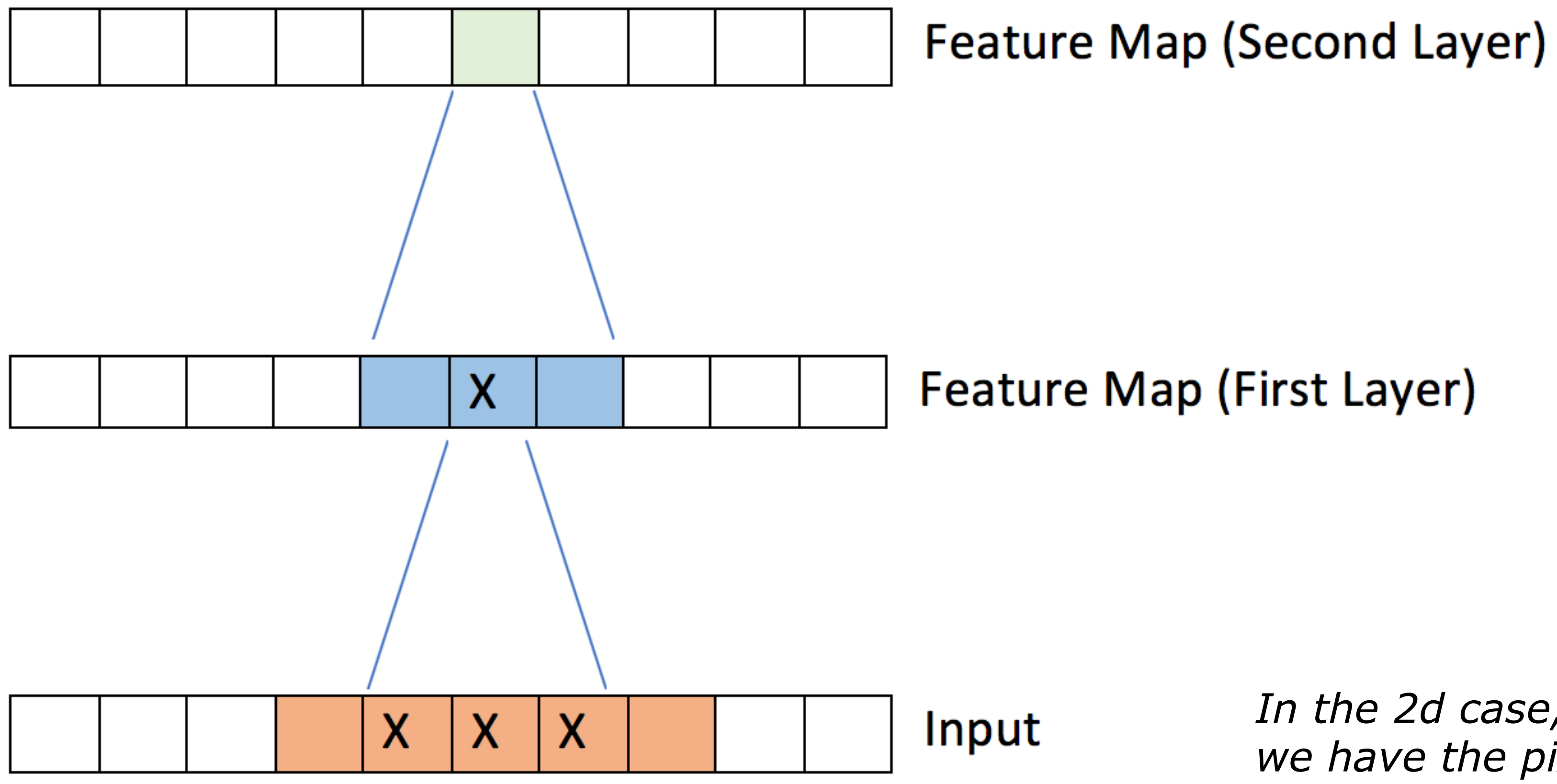
- The CNN learns to develop **local features** that, due to the compositionality of the network structure, corresponds to **higher-level information** that *progressively covers larger portion of the given image*

CNN: Image Classification (3)

- In order to better visualize this concept, let's consider the case of 1d convolutional networks (the 2d case is trivially an extension what happens in 1d)

Filters of size 3:
*Each element of this feature map has been computed "observing" 3 elements of each of the **feature maps below***
(recall that there are multiple feature maps in the layer below)

Filters of size 3:
Each element of this feature map has been computed "observing" 3 elements of the input
(this is only one feature map, but, of course, there are usually filters in each layer and then multiple feature maps)



In the 2d case, here we have the pixels of the input image

CNN: Image Classification (4)

- We could imagine that the feature map**s** of the first layer have high values in the location corresponding to certain object parts
  - Each map could be about an object part (e.g., *nose*, *eye*)
- A feature map of the second layer could have high responses in location corresponding to a full object (e.g., *face*)
  - **It virtually observes a larger portion of the input**

Notice that this is not necessarily what happens in all the CNNs trained in image classification task, but it helps in explaining the concept

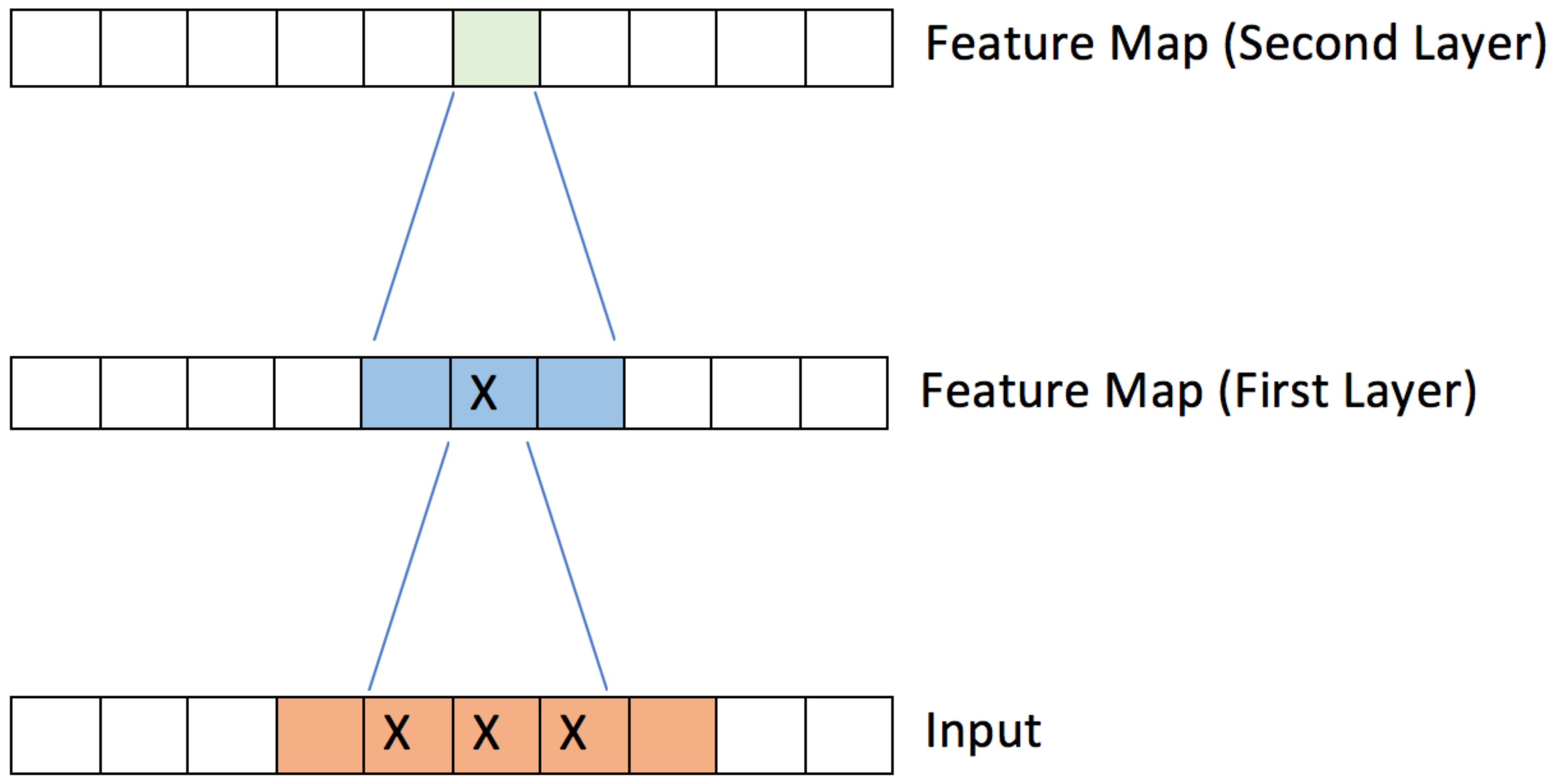


Image Classification & CNN Break-Through

- In 2012, a CNN-based model (*AlexNet*) won the Large Scale Visual Recognition Challenge
  - The challenge consisted in classifying real-world images over **1000 classes**
  - For more details see: *Krizhevsky, Alex and Sutskever, Ilya and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. NIPS 2012*

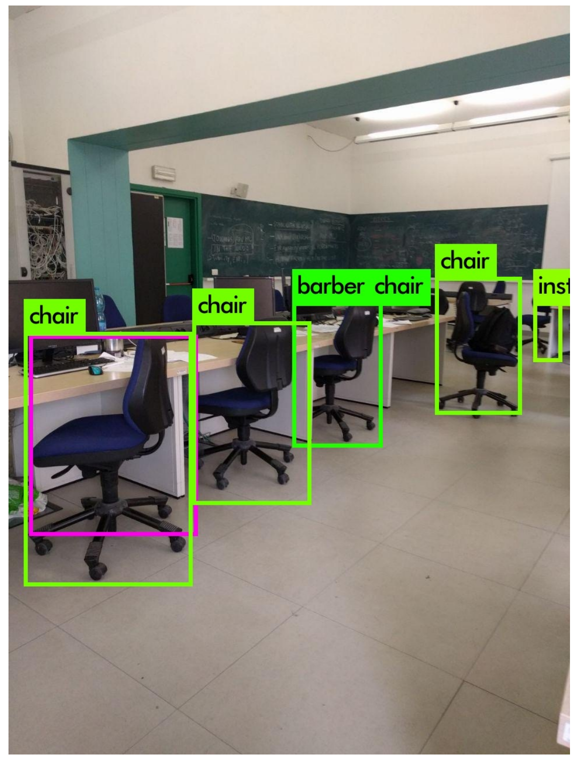
- *AlexNet* is composed by 5 convolutional layers, ReLu nonlinearities, max pooling, and 3 fully connected layers
  - Millions of parameters

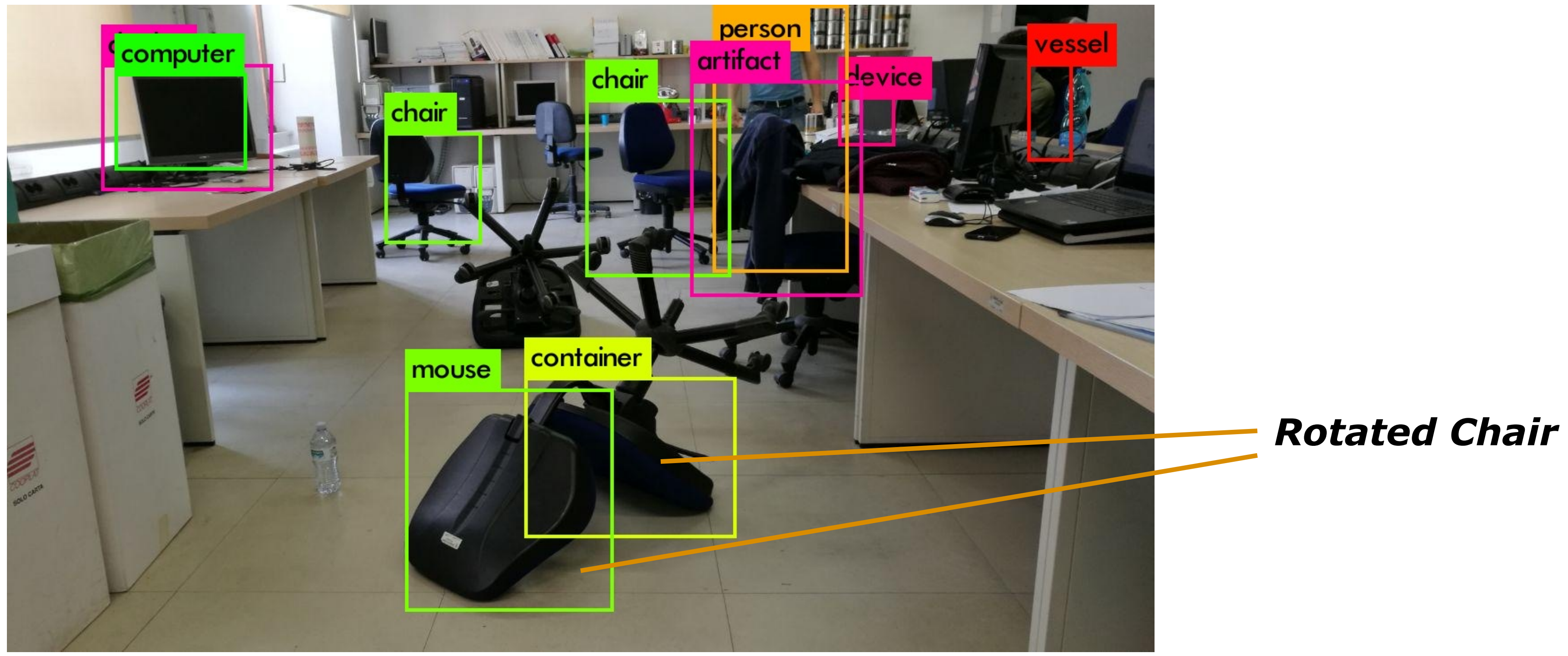
- CNNs can be efficiently trained on modern Graphical Processing Units (GPUs)
  - Pixel-wise filtering can be efficiently parallelized

CNN: Limits

- In principle, CNN-based systems (<u>not only image classifiers</u>) are not invariant to basic transformation of the input (rotations, scaling, …)
  - They are a bit more robust to small translations due to the pooling layers
  - **Data augmentation**: artificially generate rotated/scaled/cropped instances of the training images to make the network more robust

Example of CNN-based system for object localization and recognition





CNN: Limits (2)

- Under some perturbations of the input image, the network could take completely different decisions
  - If some offsets are added to pixels of the input image, the network might change decision from class A to class B, where class B is completely unrelated to A
    - Even if the image still look the same to a human observer
  - A "person" can become a "car", or a "bus" could become a "ostrich", and so on…

CNN output: Person!

CNN output: Car!