Deep Learning

Convolutional Neural Networks

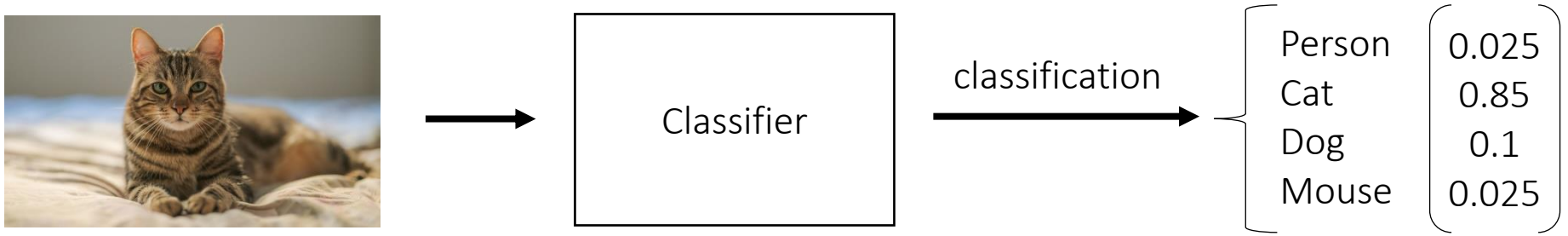
Slides by Carlos Grilo & Rolando Miragaia

Deep learning

- Deep learning transformed the area of computer vision (CV) because now the creators of AI systems do not need to tailor algorithms for specific tasks
- Instead, they can provide lots of data to the algorithm and later retrain the model to be able to execute another type of task
- For example, we can train a model to be able to execute face detection and later retrain the model to be able to detect diseases on medical images

Image classification

- Image classification is the task of taking an input image and outputting a class (a cat, dog, etc.), or a probability of classes that best describes the image



- For us, humans, this is a very natural task
- This is not the case with computers…

Inputs

What we see

```
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48
```

What computers see

Inputs

- When a computer takes an image as input, it will see an array of pixel values
- Let's say we have a color image in JPG form and its size is 480 x 480
- The representative array of numbers will have dimensions 480 x 480 x 3 (3 refers to RGB values)
- Each of these numbers is given a value from 0 to 255, describing the pixel intensity at that point
- These numbers are the only inputs available to the computer

Outputs

- The idea is that we give the computer this array of numbers and it will output numbers that describe the probability of the image belonging to a certain class (.80 for cat, .15 for dog, .05 for bird, etc.)

Feature detection

- In order to correctly classify an image, the system must be able to identify the features that are specific to each class

Eyes,
Nose,
Mouth,
…

Doors,
Windows
Roof,
…

Wheels,
Licence plate,
Headlights,
…

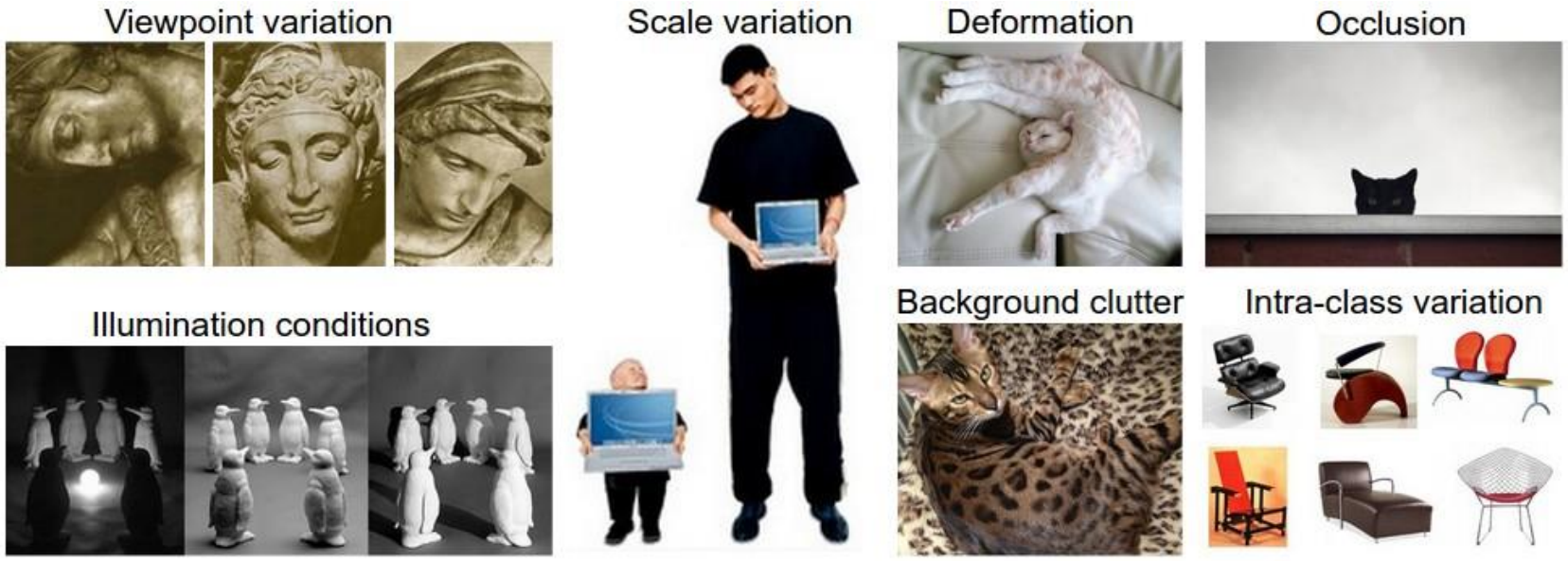
Manual feature detection

- In order to be able to classify images in some domain, we can use our knowledge about the domain to define the features that are needed and then build a system that is able to detect those features

Manual feature detection

- Challenge: How do we detect the features?

A good image classification model must be invariant to all these variations, while simultaneously retaining sensitivity to the inter-class variations

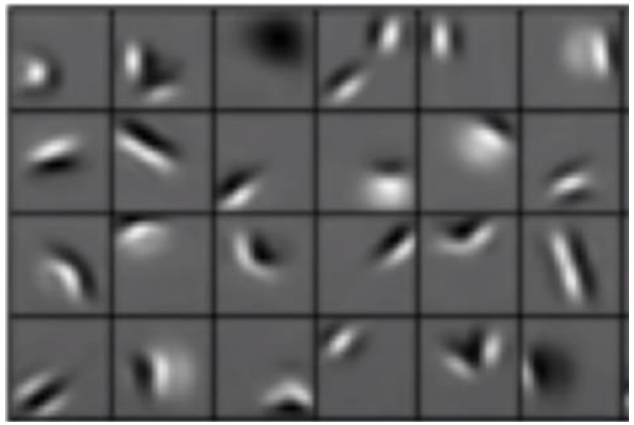


- Manual extraction of features (that is, developing algorithms able to extract features) is a very difficult task

Feature detection

- Can we automatically learn a hierarchy of features directly from the data instead of manual engineering them?

Low level features



Edges, dark spots,…

Mid level features

Eyes, ears, noses,…

High level features

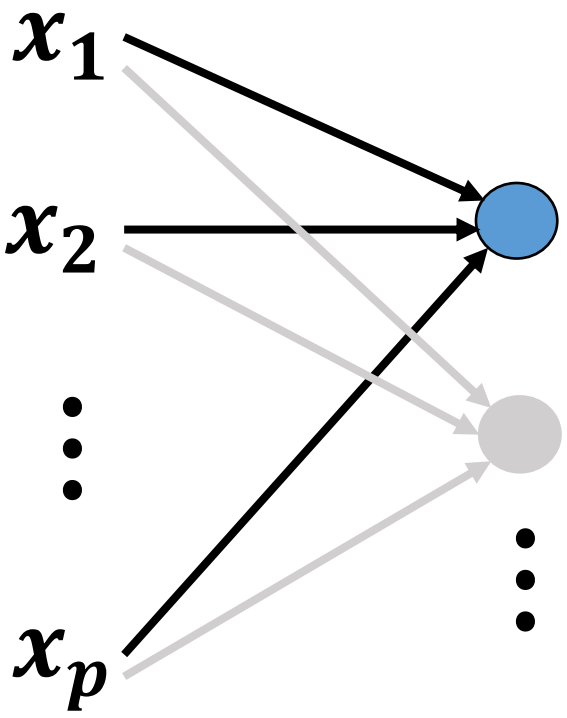
Faces

Convolutional neural networks

- This can be done with convolutional neural networks
- These networks are able to learn the visual features directly from data, as well as a hierarchy of these features and build a representation of what makes up our final classes labels
- The patterns they learn are translation invariant -> after learning a certain pattern, a convnet can recognize it anywhere
- Let us first see why don’t we do it with MLPs

Image classification using MLPs

Input: we would need to collapse our 2D image into a 1D vector of pixel values



Problems

- Spatial information lost
- Many parameters (weights)

Every pixel would feed each unit of the 1st hidden layer

Image classification with NNs

How can we use spatial structure in the input to design the architecture of the network?

POLITÉCNICO DE LEIRIA

Feature Extraction

Feature extraction

X

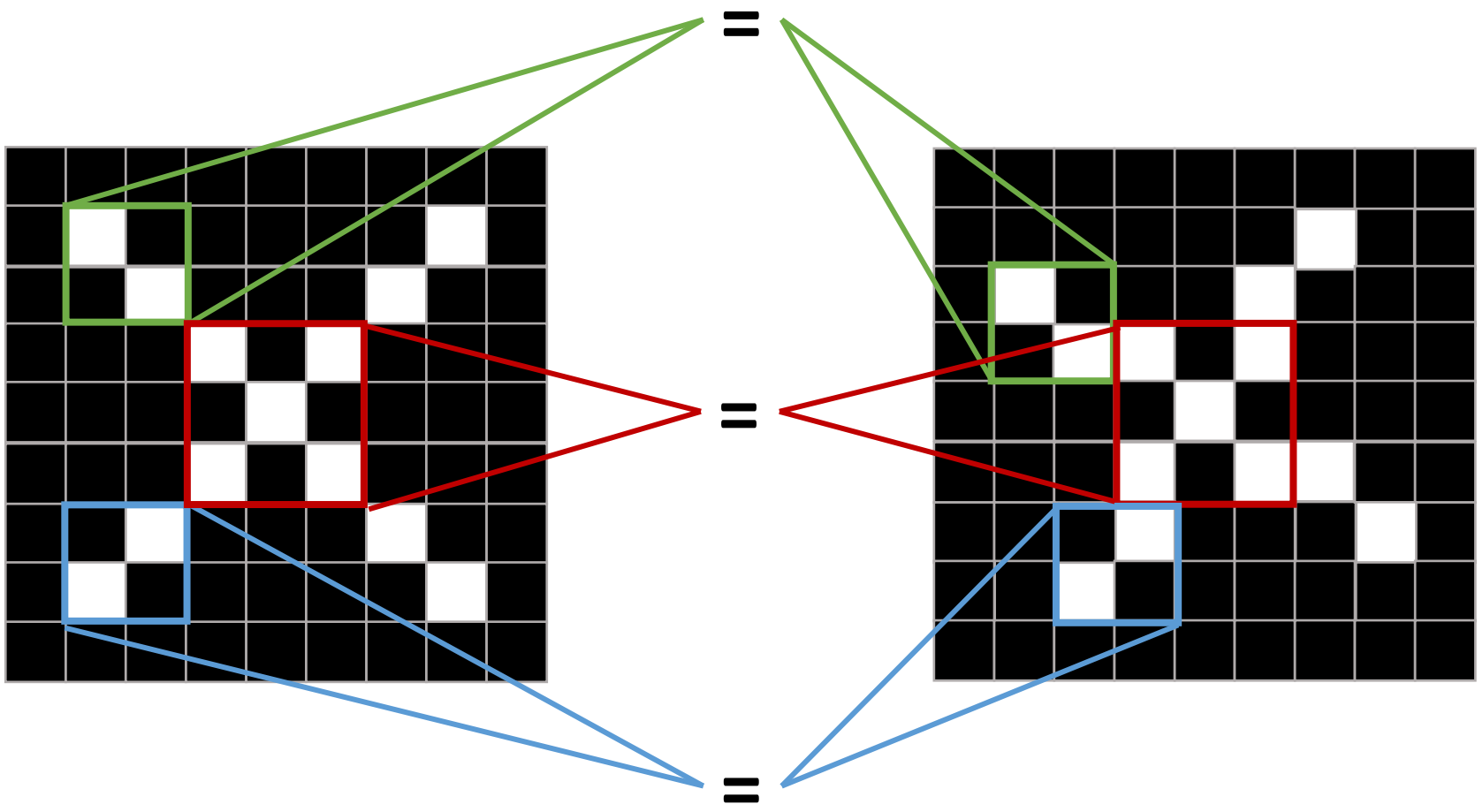
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

X?

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	-1
-1	1	1	-1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	1	-1	-1
-1	-1	-1	1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

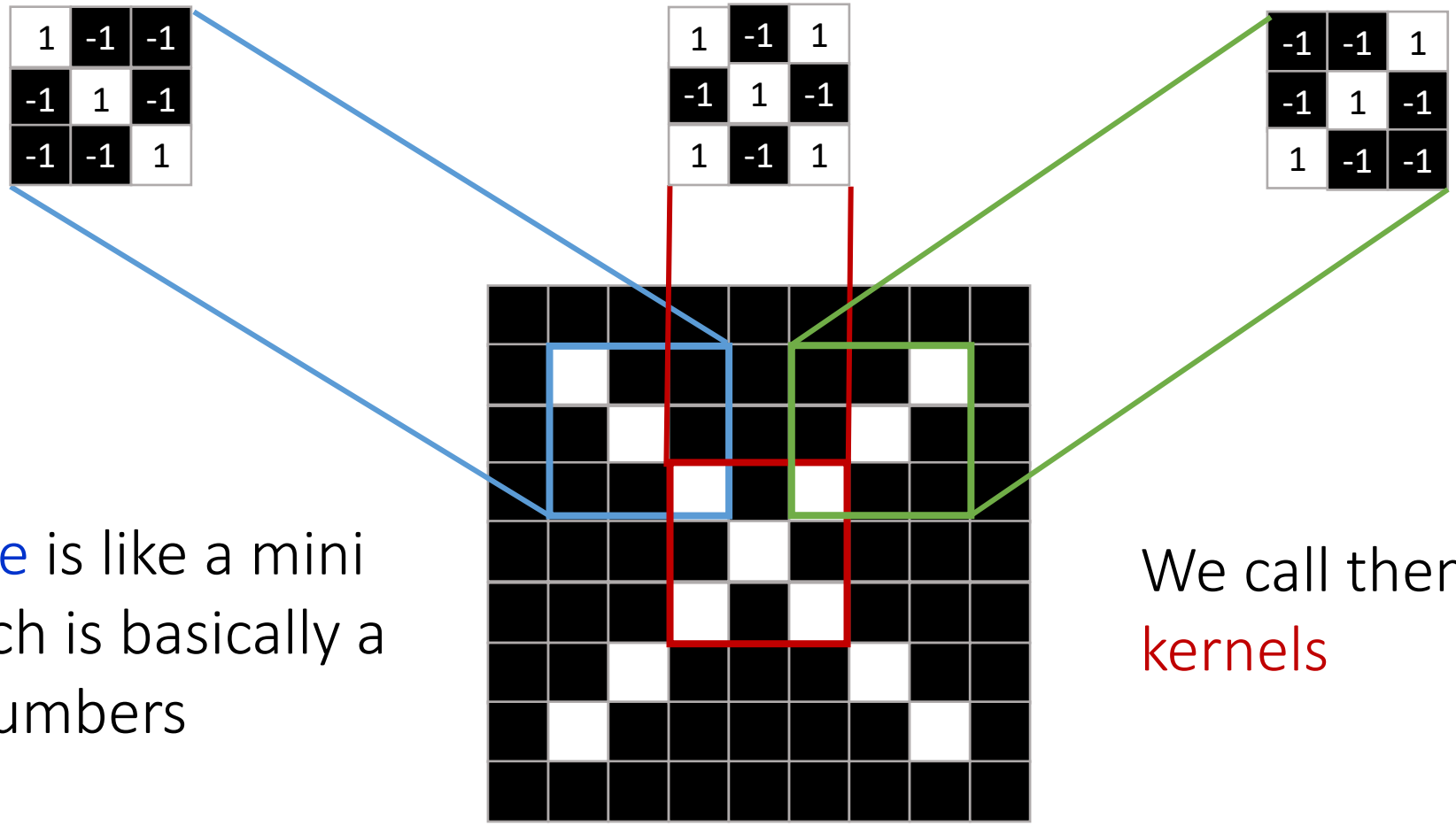
We want to be able to classify an X as an X even if it is shifted, shrunk, rotated or deformed

Feature extraction



Feature extraction

Filters



Each feature is like a mini image, which is basically a matrix of numbers

We call them filters or kernels

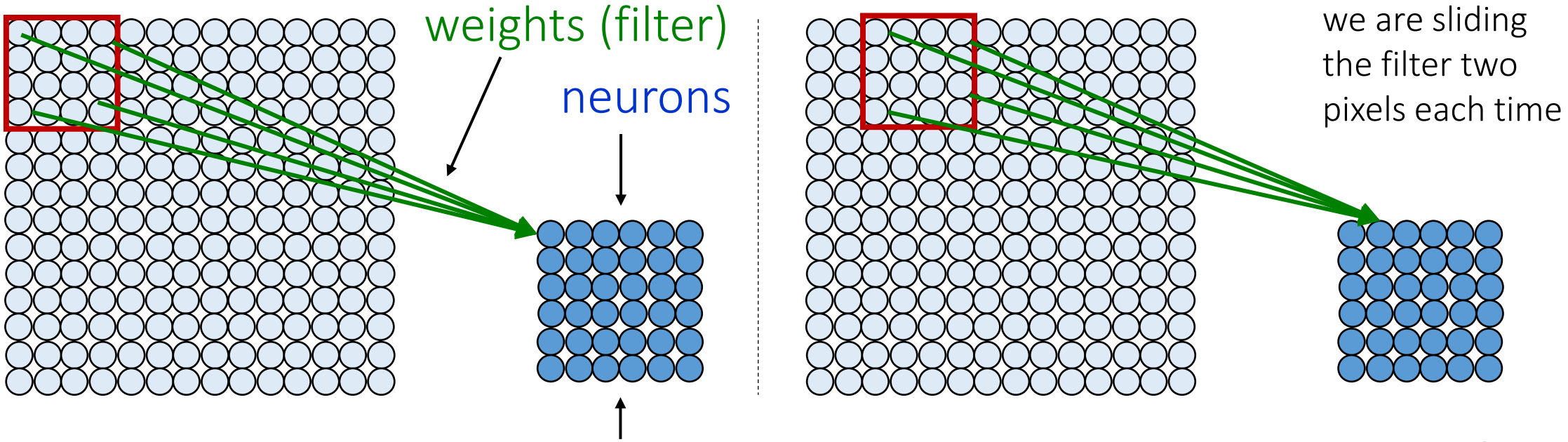
Feature extraction

- We want to have a way of computing if a feature occurs in the image and where does it occur (because it may occur more than once in different places)
- We may train different feature detectors (each one detects one feature) and each detetor moves around the image looking for that feature in different windows of the image
- We need also to somehow save information about the occurrence of each feature in each window

Feature extraction

1	-1	-1	1
-1	1	1	-1
-1	1	1	-1
1	-1	-1	1

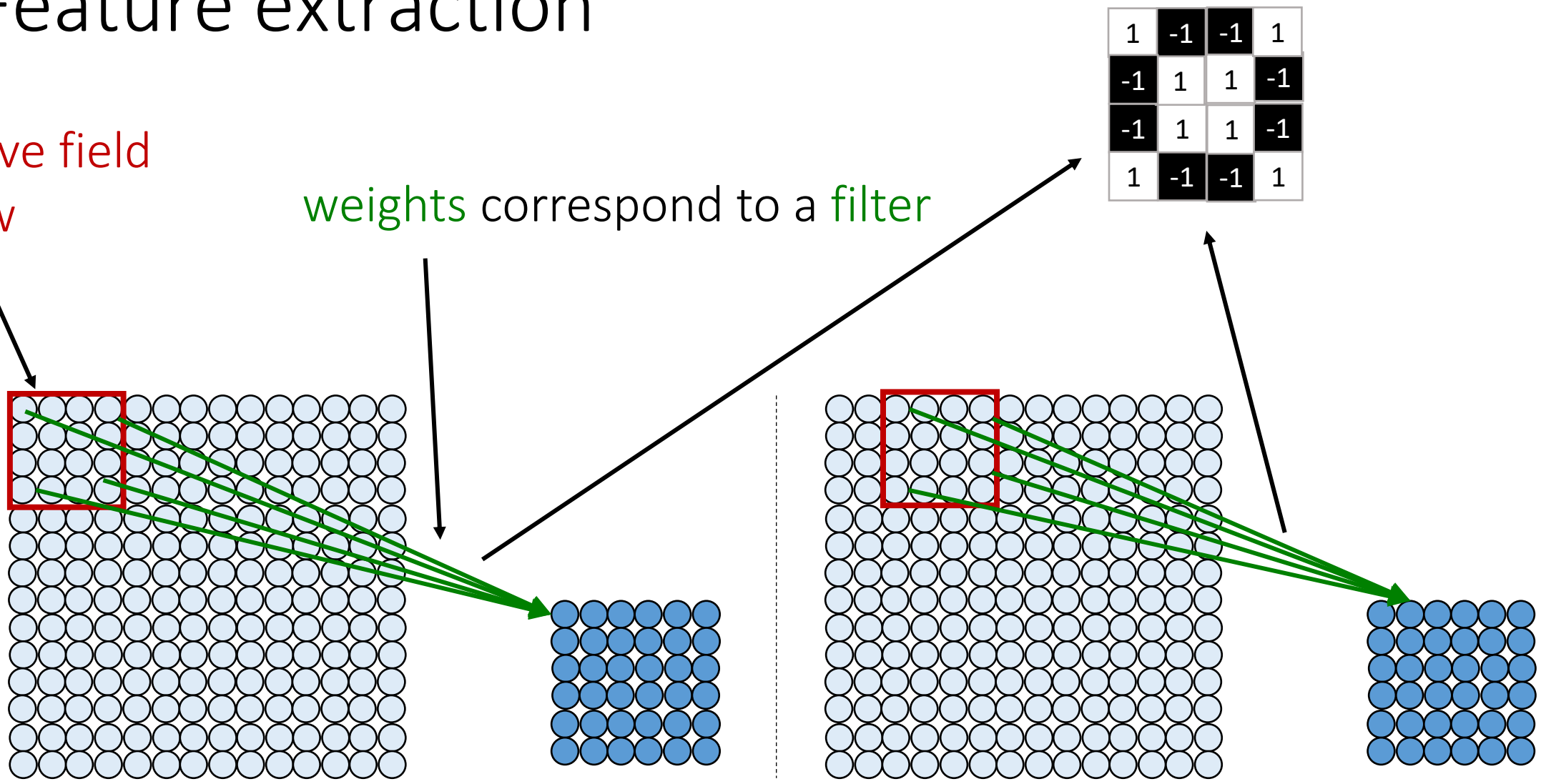
- We may slide each filter across the image. For each image window (called receptive field) *versus* filter, a value is computed and "saved" in a neuron in the next layer of the network



Feature extraction

receptive field
window
patch

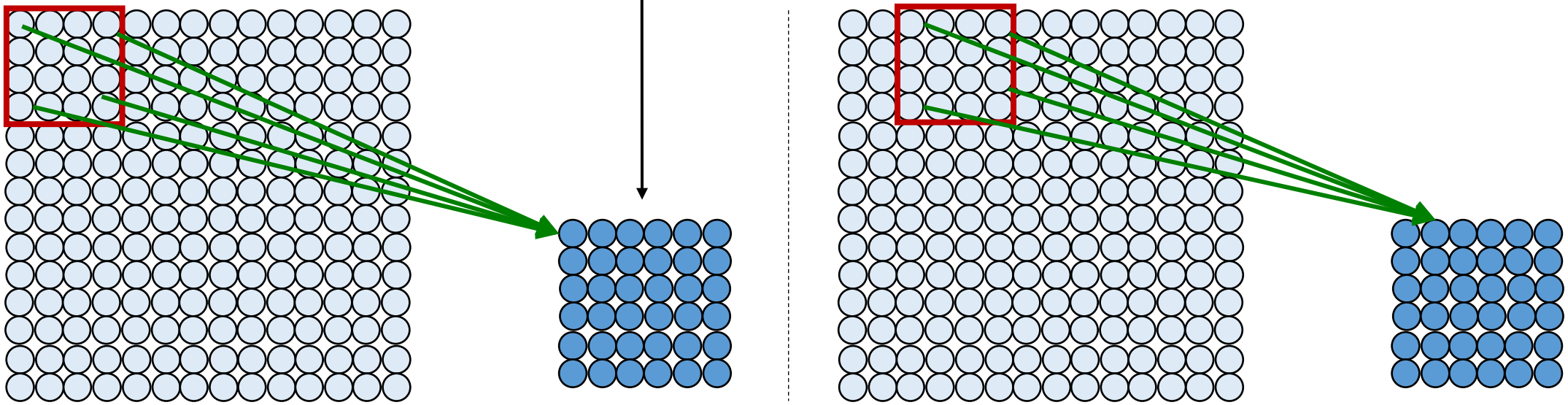
weights correspond to a filter



Feature extraction

1	-1	-1	1
-1	1	1	-1
-1	1	1	-1
1	-1	-1	1

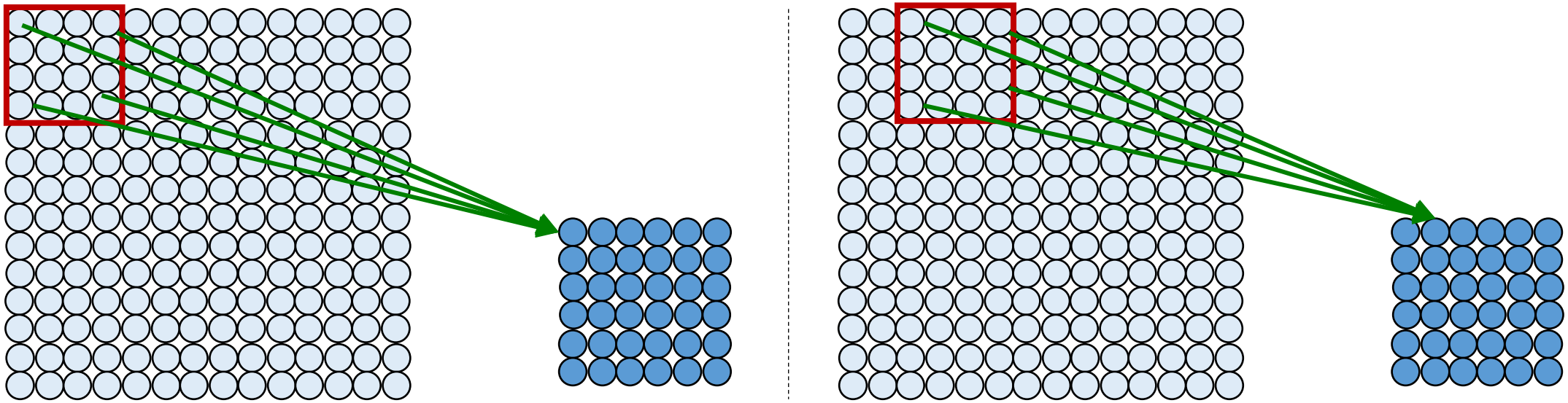
Each neuron is connected to just one window and it computes a value that somehow reflects “how much” is the feature present in that window



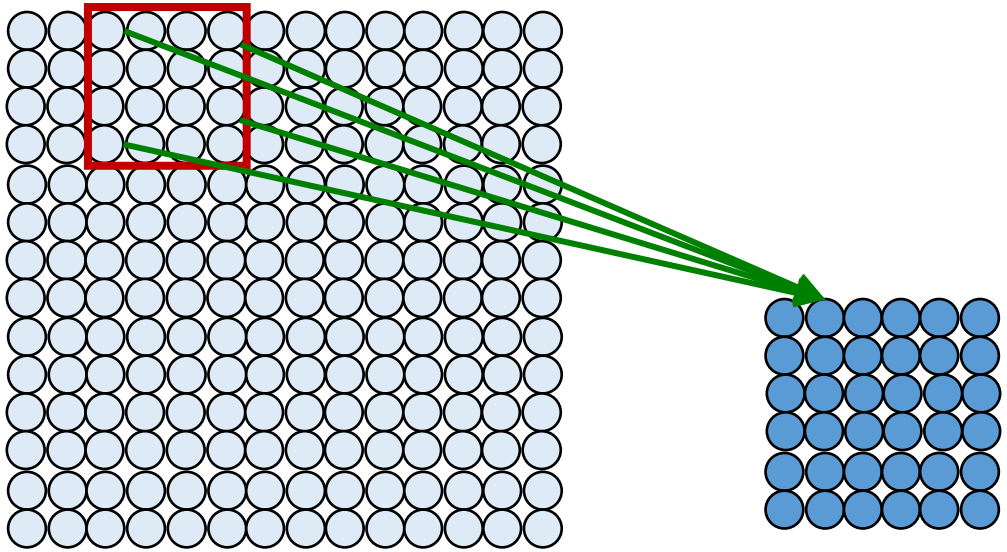
Feature extraction

1	-1	-1	1
-1	1	1	-1
-1	1	1	-1
1	-1	-1	1

The state/value of each neuron is computed as a weighted sum of the pixel values using the weights of the filter values and it is this operation that is used to extract features in the image



Feature extraction with convolution



- Filter of size 4x4: 16 weights (in this case)
- Apply this same filter to 4x4 windows (compute weighted sum)
- Shift the filter across all the image (by 2 pixels, in this example)

This operation is called Convolution

The convolution operation

The power of convolutional neural networks comes from the convolution operation

This operation is a way of extracting features from a signal

The convolution operation

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

Input

1	0	1
0	1	0
1	0	1

Filter / Kernel

The convolution operation

1	0	1
0	1	0
1	0	1

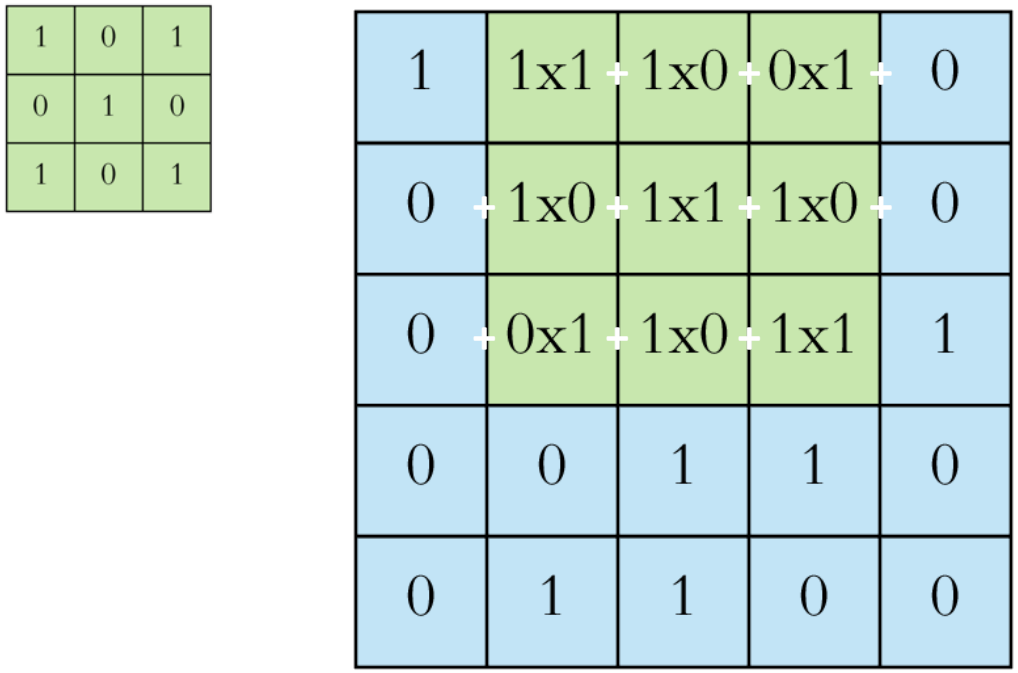
1x1 +	1x0 +	1x1 +	0	0
0x0 +	1x1 +	1x0 +	1	0
0x1 +	0x0 +	1x1	1	1
0	0	1	1	0
0	1	1	0	0

4		

The application of a filter to a window consists in performing an element-wise multiplication and then sum all the results

We can also think about this operation as a weigthed sum since the values of the “pixels” of each filter correspond to the weights of that filter

The convolution operation



The filter shifts accross the image and the weigthed sum is applied to different windows;

The output of each of each these operations becomes the state of a neuron

The convolution operation

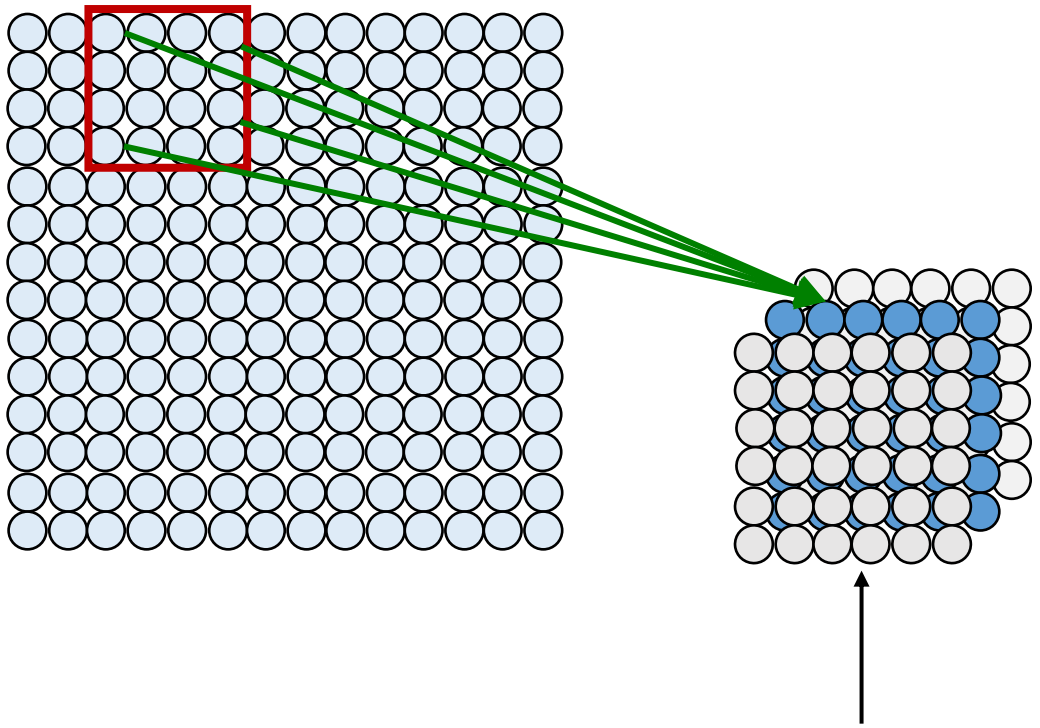
...

1	0	1
0	1	0
1	0	1

1	1	1	0	0
0	1	1	1	0
0	0	1x1 +	1x0 +	1x1
0	0	+ 1x0 +	1x1 +	0x0
0	1	+ 1x1 +	0x0 +	0x1

4	3	4
2	4	3
2	3	4

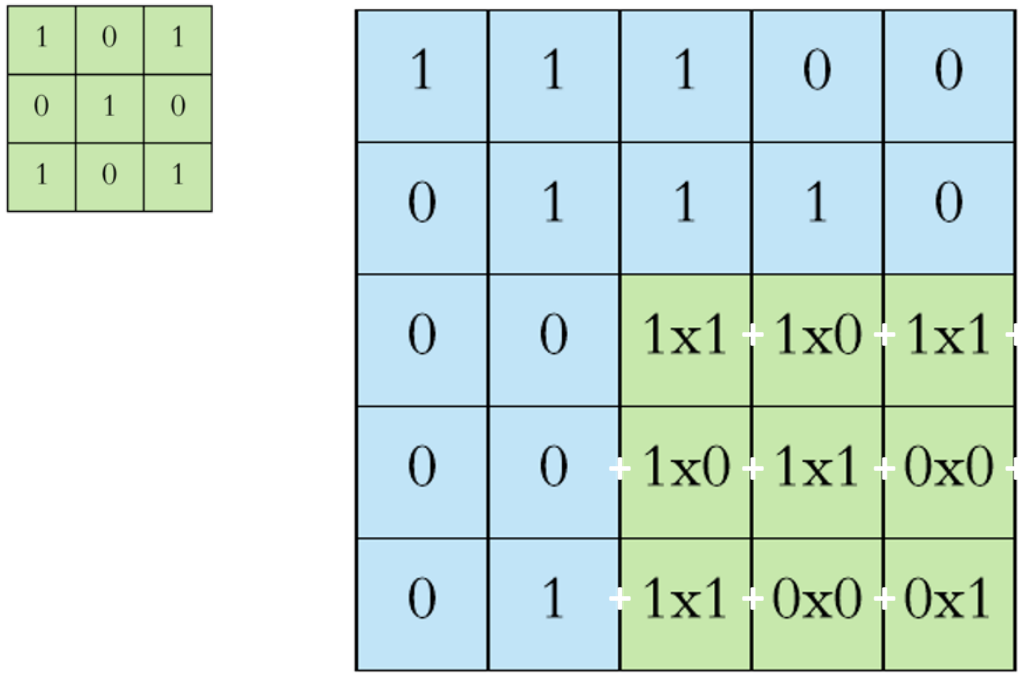
Feature maps



3 feature maps resulting from the application of 3 different filters

- So far, we have seen the application of just one filter
- However, each filter allows the extraction of just one feature across the image
- And we want our network to be able to extract many features
- So multiple filters are used to extract different features
- A set of neurons whose state is computed using the same filter is called a feature map

Convolution

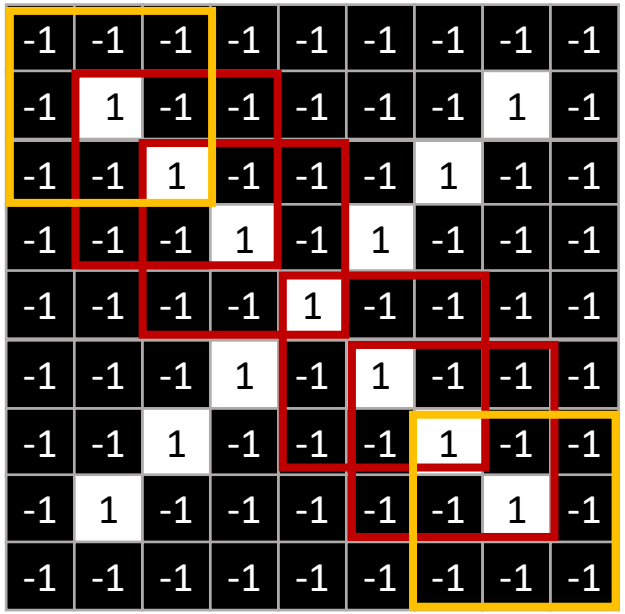


POLITÉCNICO DE LEIRIA

How does the convolution operation support the extraction of features?

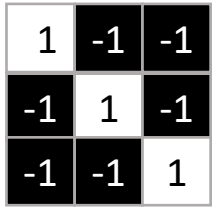
Feature extraction and convolution

input image



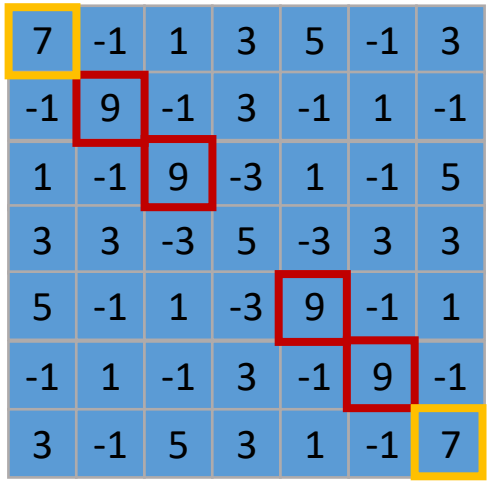
conv

filter



=

feature map



The larger values in the feature map correspond to windows that better resemble the filter/feature

The feature map reflects where in the image there is activation by this particular filter

Exercise

input image

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

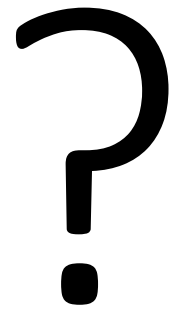
conv

filter

1	-1	1
-1	1	-1
1	-1	1

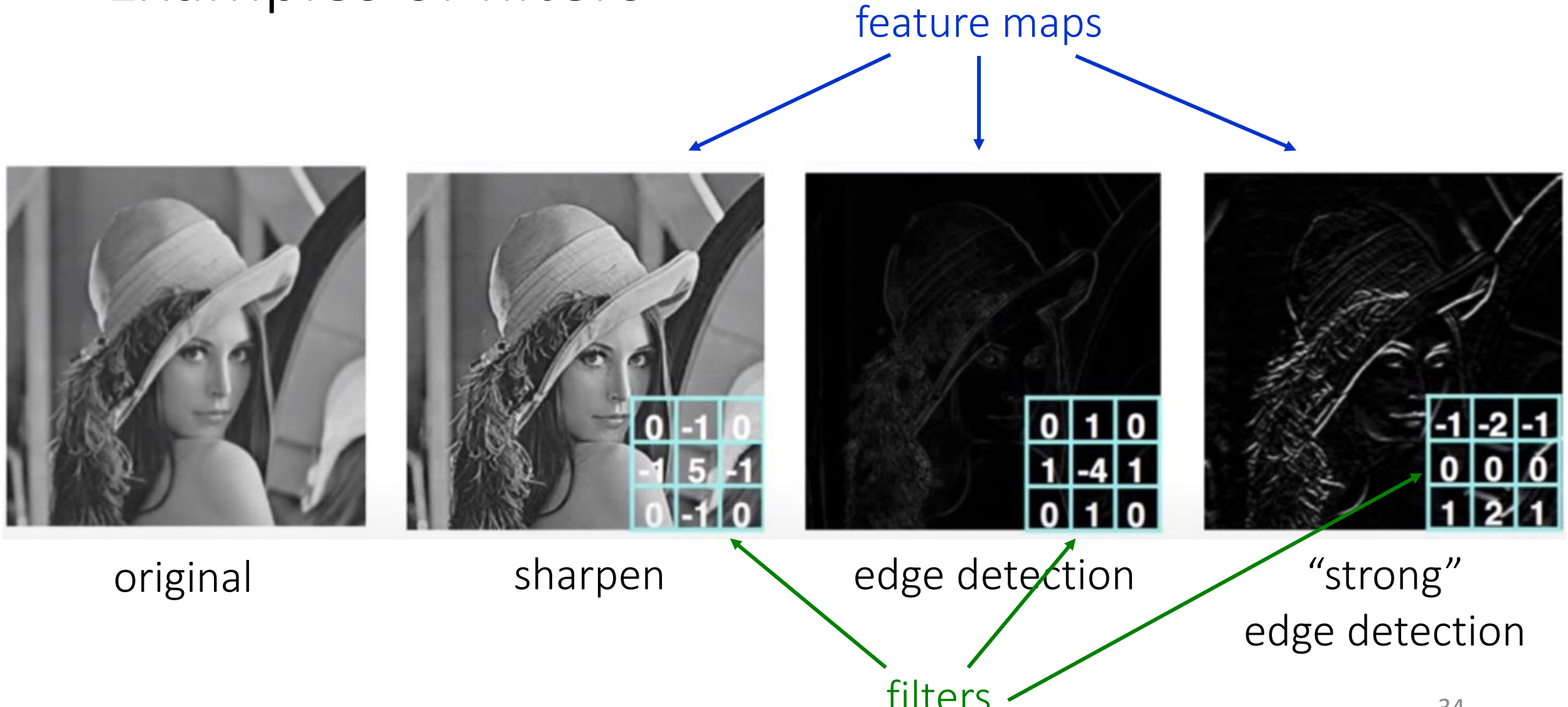
=

feature map



Note: Shift the filter one pixel to the right/down each time

Examples of filters

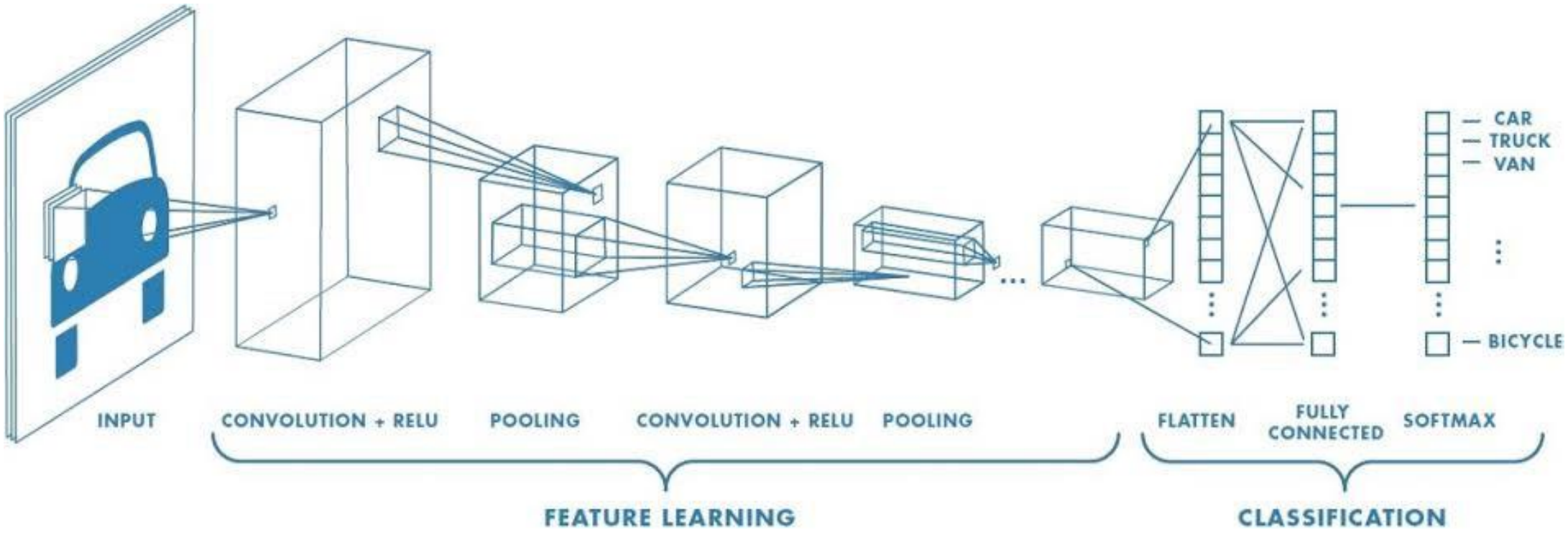


POLITÉCNICO DE LEIRIA

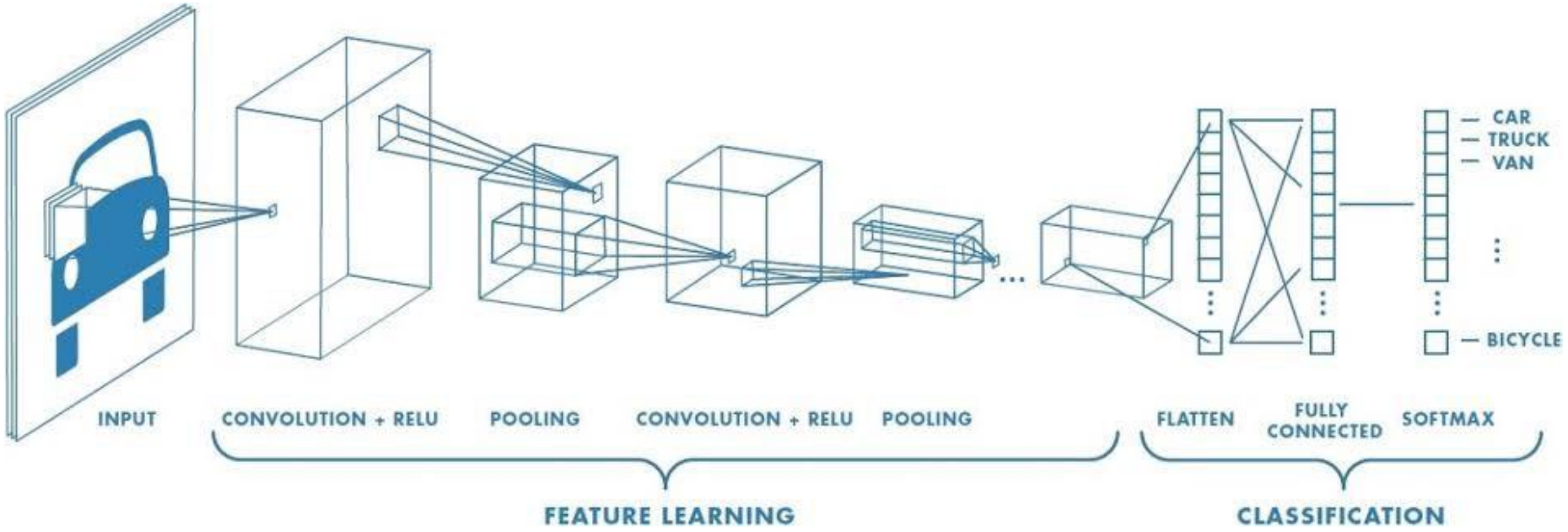
Convolutional Neural Networks

(CNNs)

CNNs general architecture

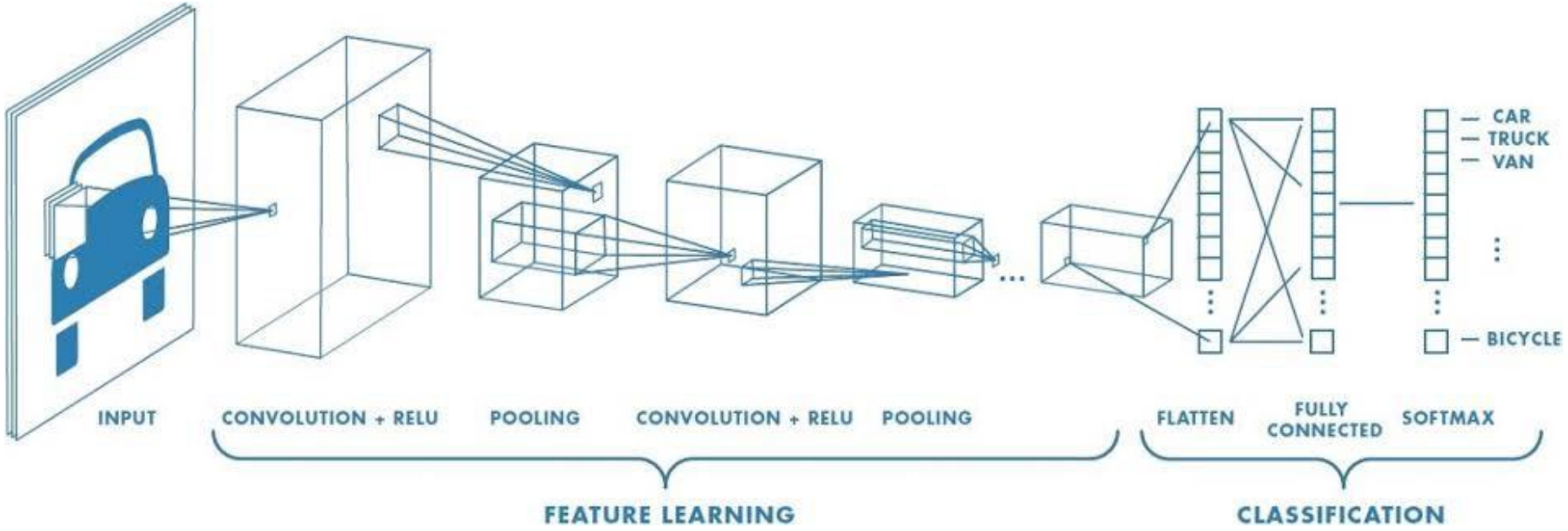


CNNs main components



1. Convolutional layers: apply filters to generate feature maps
2. Non-linearity (ReLU): non-linear function applied to the feature map values
3. Pooling layers: used to down sample feature maps
4. Fully connected layers: the layers responsible for the classification task

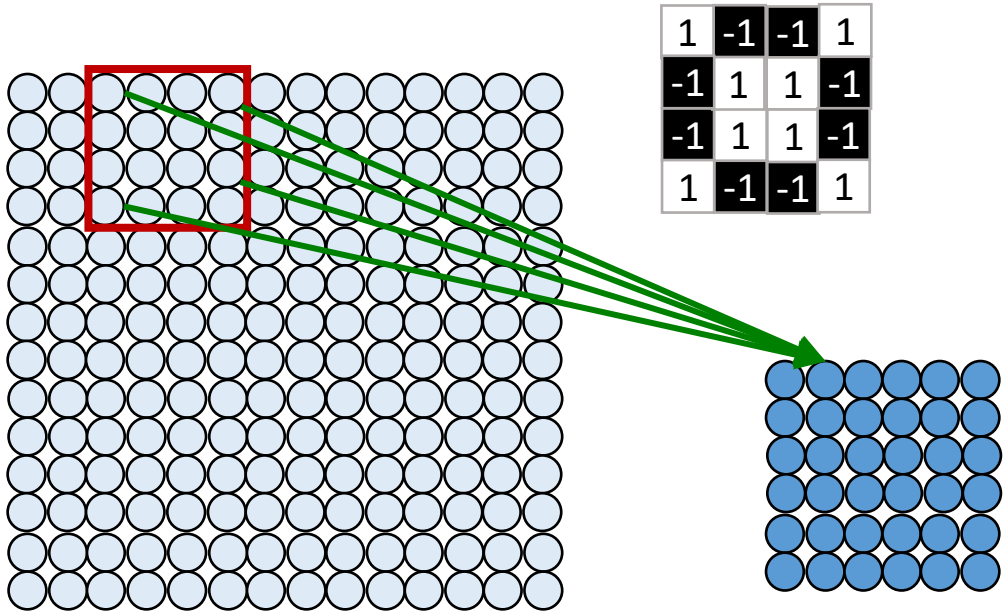
CNNs general architecture



There can be several Conv + Pool blocks

That's why these networks are called deep neural networks

Convolutional layers



In this example, each filter would be a 4x4 matrix of weights

For each neuron in the hidden layer:

1. Take inputs from window
2. Compute weighted sum
3. Apply bias

filter weight

pixel value

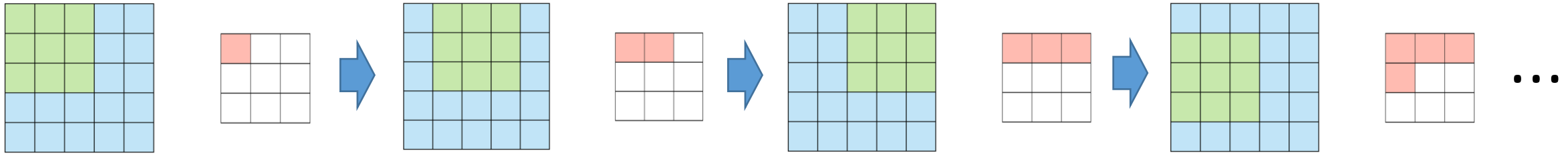
$$\sum_{i=1}^{4}\sum_{j=1}^{4} w_{ij}x_{p+i,q+j} + b$$

For neuron (p, q) in hidden layer

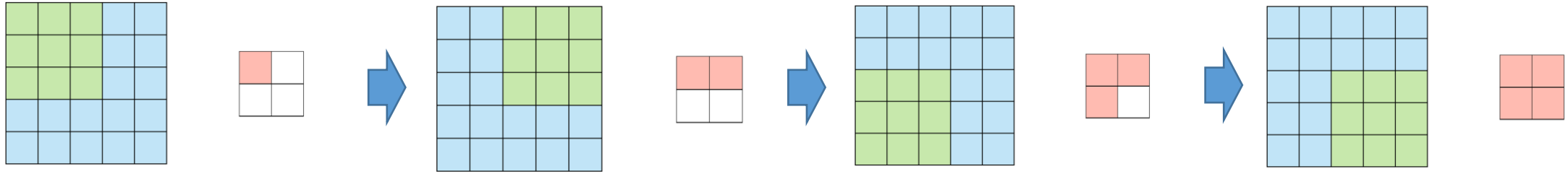
Stride

Specifies how much we move the convolution filter at each step

Stride 1



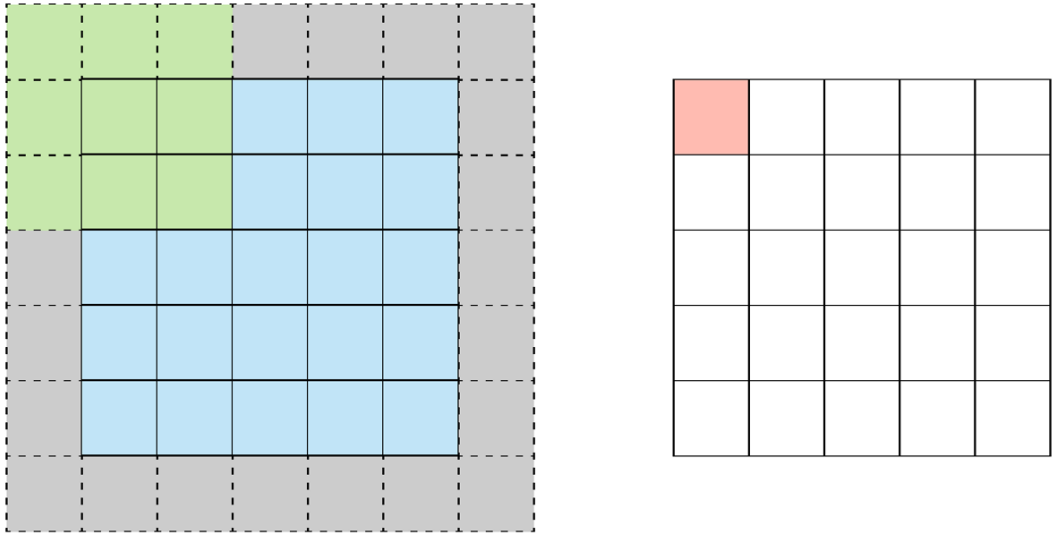
Stride 2



Usually, stride 1 is used

Padding

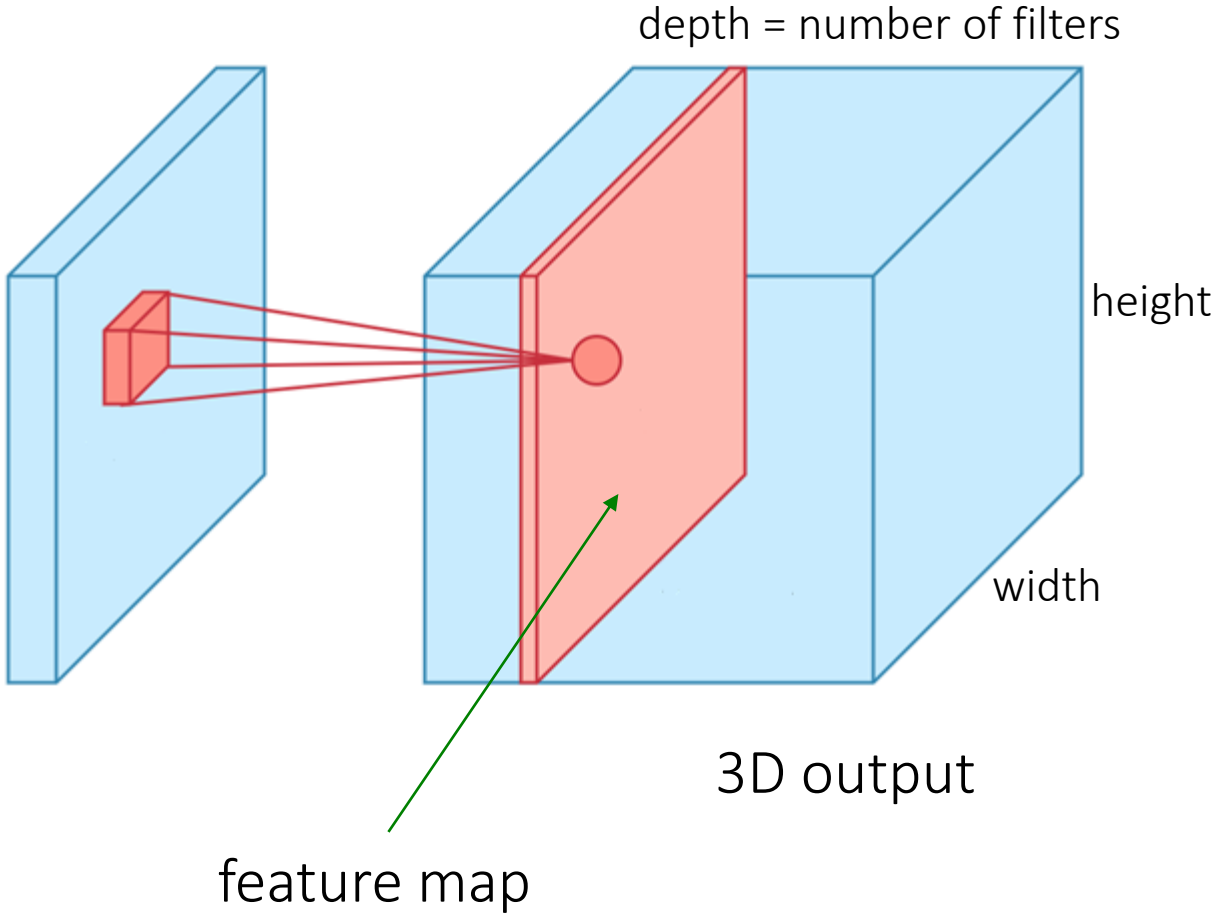
It is used to preserve the size of the feature map equal to the size of the input



Usually, 0s are used in the padding area (grey area) -> this is called zero-padding

Note: in this example, stride = 1

Convolutional layers – multiple filters

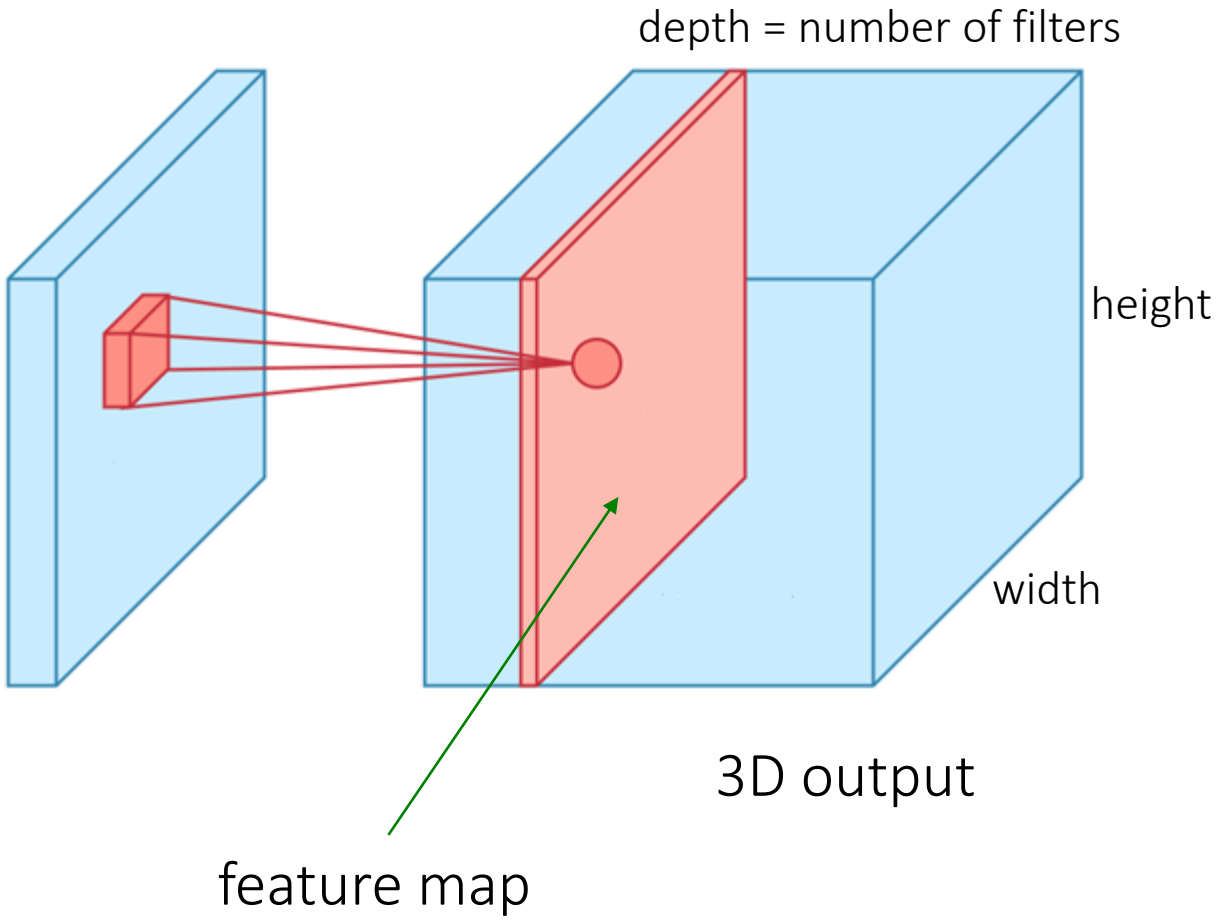


As we have seen before, we will have multiple filters, that allow us to extract multiple features

Therefore, the output of a convolutional layer is composed of a volume of feature maps (we may think on these feature maps as images)

The height and width of the feature maps depend on the filter size, on the stride value and if padding is used

Convolutional layers



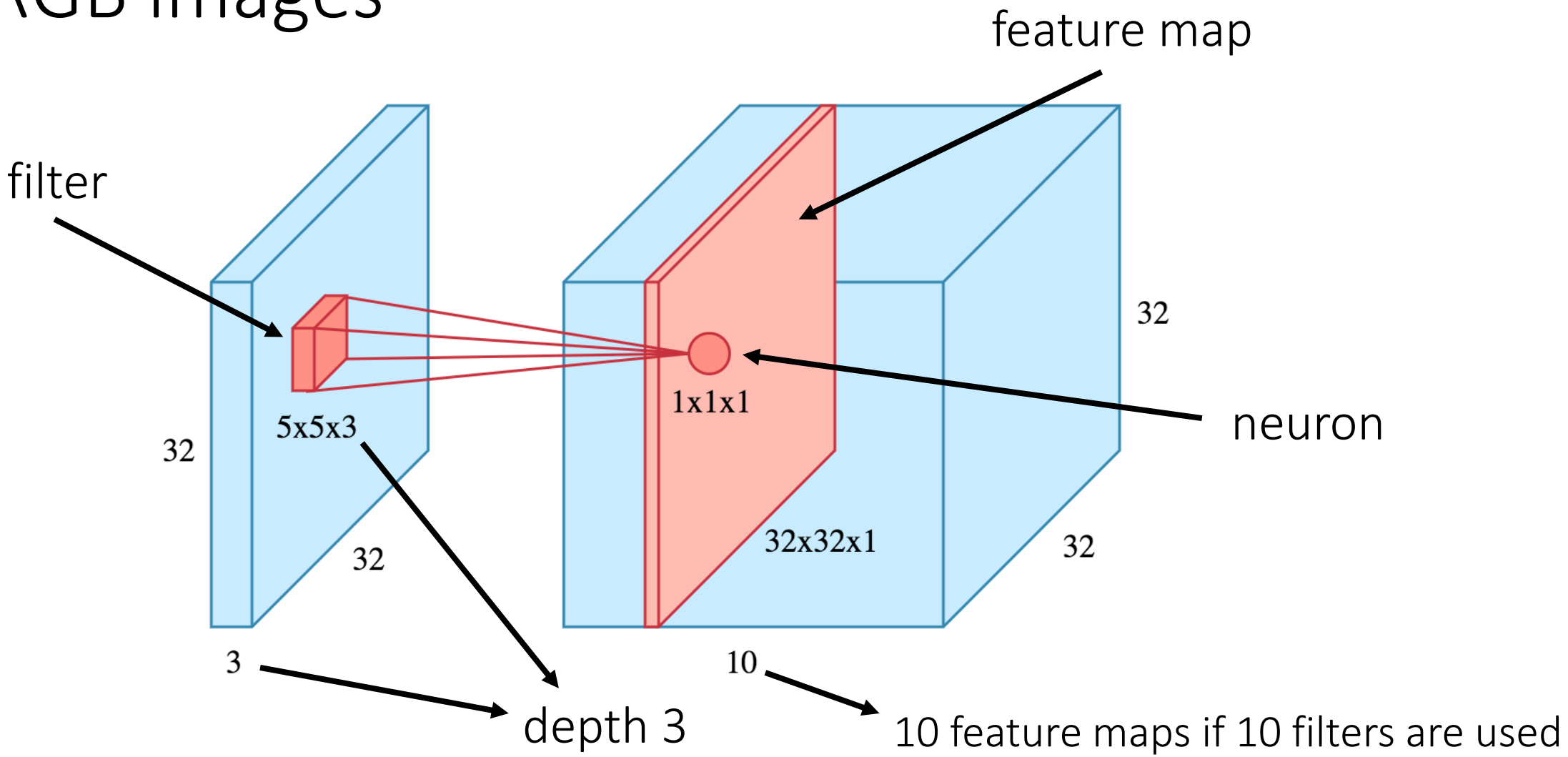
Common configurations:

- 3 x 3 filters
- Stride 1
- With padding

RGB images

- Until now, we have considered black and white or grey scale images
- What about RGB images (the majority)?
- RGB images are represented as a 3D matrix where the depth corresponds to color channels
- So, in these cases, the filters are also 3D, with depth 3
- But the result of the application of the filter to a receptive field is still a scalar

RGB images


feature map
filter
32
5x5x3
32
3
1x1x1
32x32x1
32
32
10
neuron
depth 3
10 feature maps if 10 filters are used

Feature map size

- Given
  - W: size of the input volume (for example, 28 for an image of 28 x 28 pixels)
  - F: size of the receptive field/filters (for example, 3 for 3 x 3 filters)
  - S: stride value
  - P: amount of zero padding

the size of the feature map is given by

$$\frac{W - F + 2P}{S} + 1$$

Exercise: confirm this with examples given in previous slides

Number of weights of a conv layer

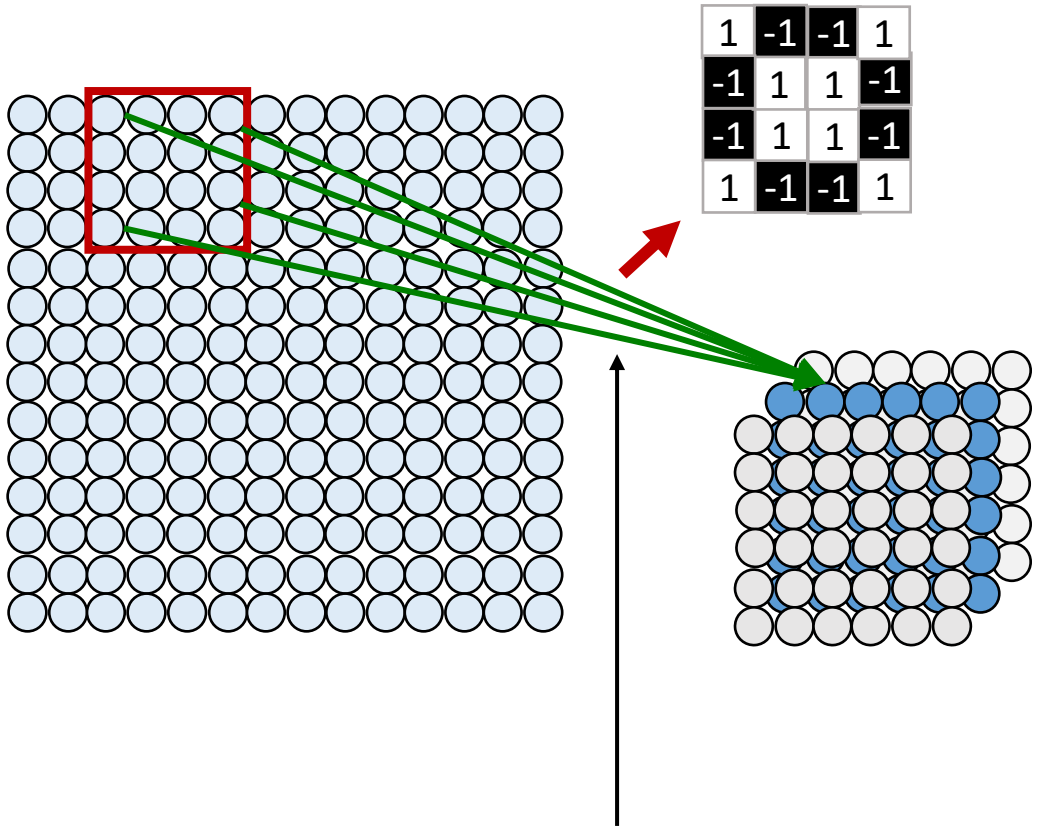
- Given
  - *FM*: number of feature maps of the layer
  - *F*: size of the receptive field/filters (for example, 3 for 3 x 3 filters)
  - *FMP*: number of feature maps of the previous layer

the number of weights of a conv layer is given by

$$FM \times (F \times F \times FMP + 1)$$

One bias per feature map

How are filters generated?



- The features are not engineered “manually”
- Instead, they are learnt during the training phase
- In fact, the training phase consists in changing progressively the weights of the network until it works as desired
- Remember that each filter/feature is defined by the weights associated with each feature map
- So, we can say that the features are discovered and tuned during the training phase

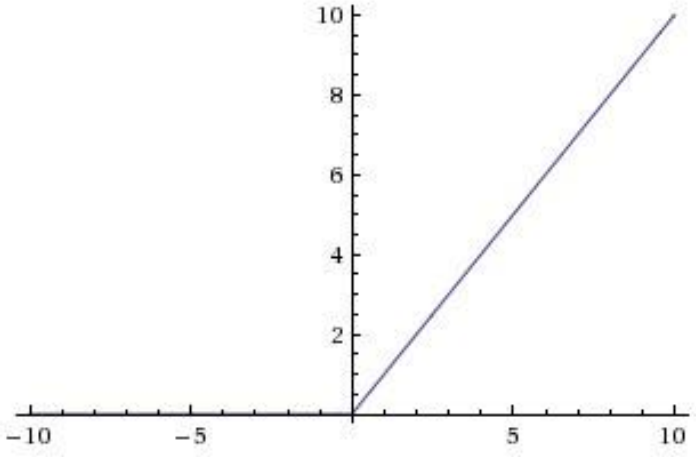
Applying non-linearity

Non-linearity is applied to all neurons/pixels of all feature maps after the convolution operation

This allows a model to respond in a non-linear way to the inputs

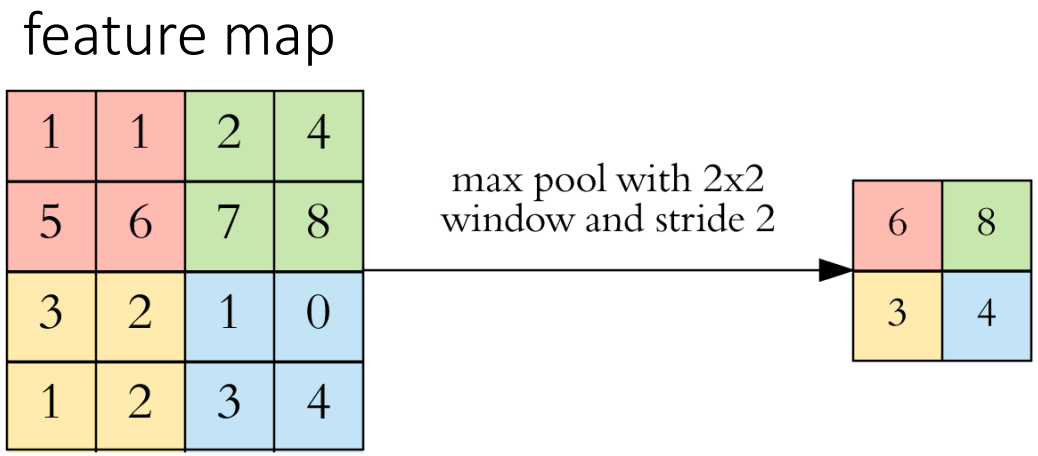
The most commonly used activation function after the convolutional layer is the ReLU function

Rectified linear unit (ReLU)



$$g(z) = \max(0, z)$$

Pooling layers



Goal of pooling layers: down sampling feature maps while keeping the important information

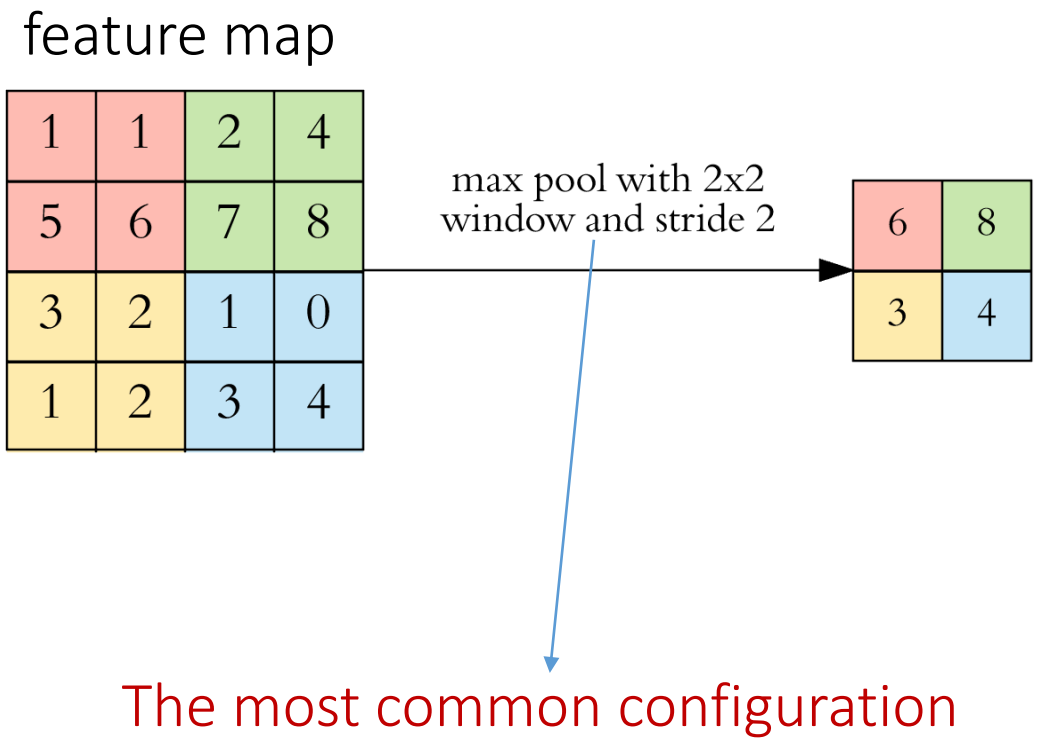
Most common technique: **max pooling**

Max pooling: slide a window over the feature map and simply take the max value in the (pooling) window

Contrary to the convolution operation, pooling has no parameters (weights)

As for convolution, we need to specify the window size and stride

Pooling layers



Pooling layers down sample each feature map independently

They reduce the height and width, keeping the depth intact

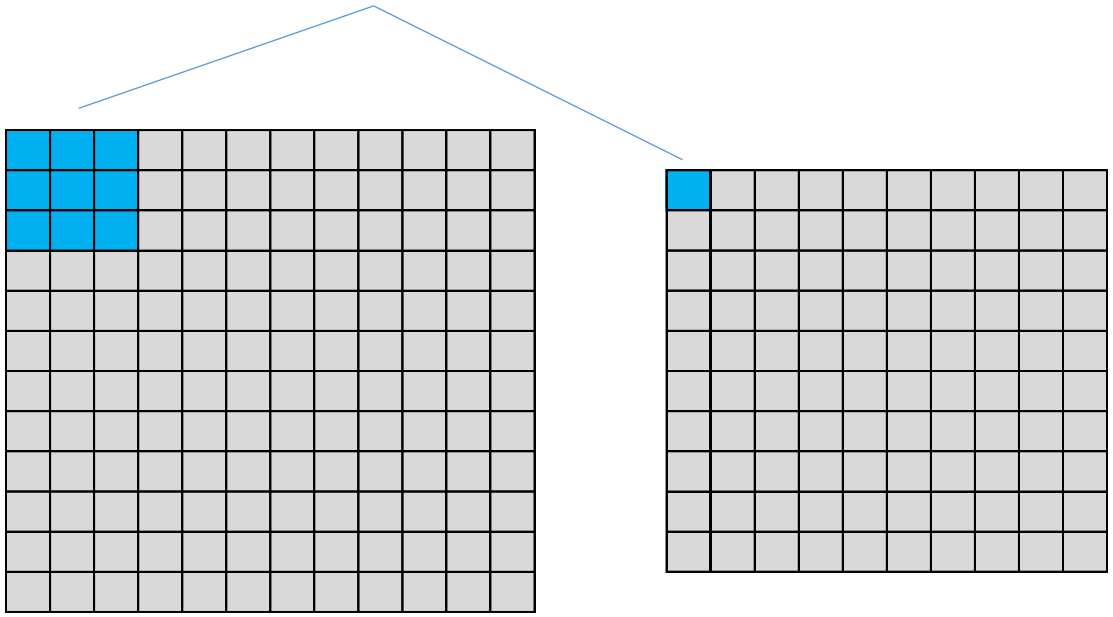
Thus, pooling layers reduce the number of parameters

This both shortens the training time and combats overfitting

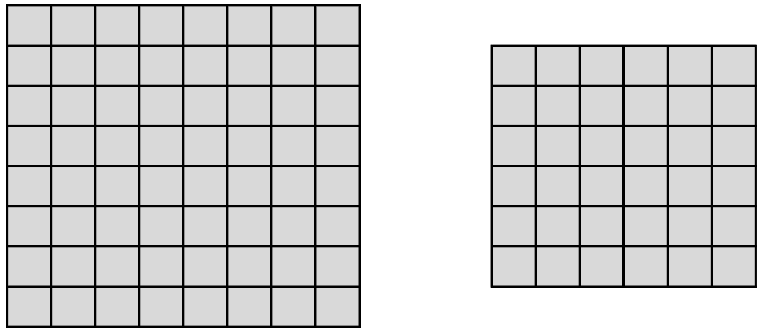
Why not average pooling?

- Well, we can use it…
- However, usually, max pooling achieves better results
- This is because the idea it to register/assess if some feature is present in some region of the image
- Computing the max value in that region allows us to do that
- Computing the average may cause us to miss or dilute feature-presence information

If we don’t use pooling



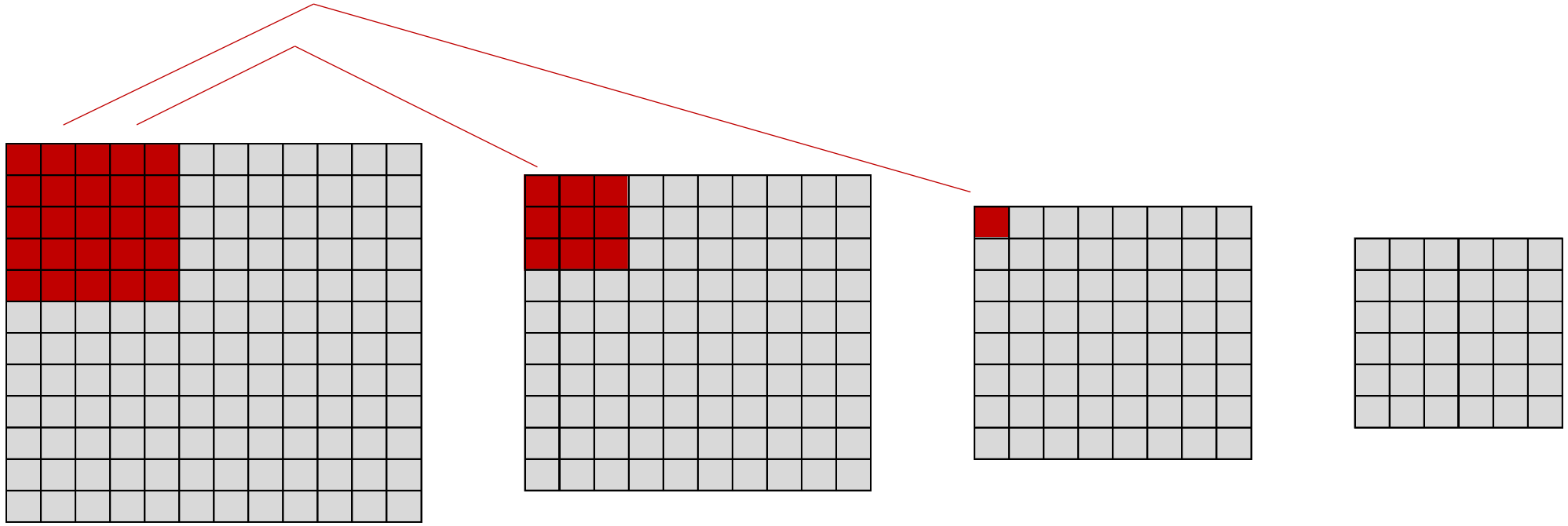
Let us consider that all filters are 3 x 3, stride = 1 and no padding is used



Input image | 1st conv layer | 2nd conv layer | 3rd conv layer

Each neuron in the 1st conv layer contains information coming from a 3x3 window in the input image

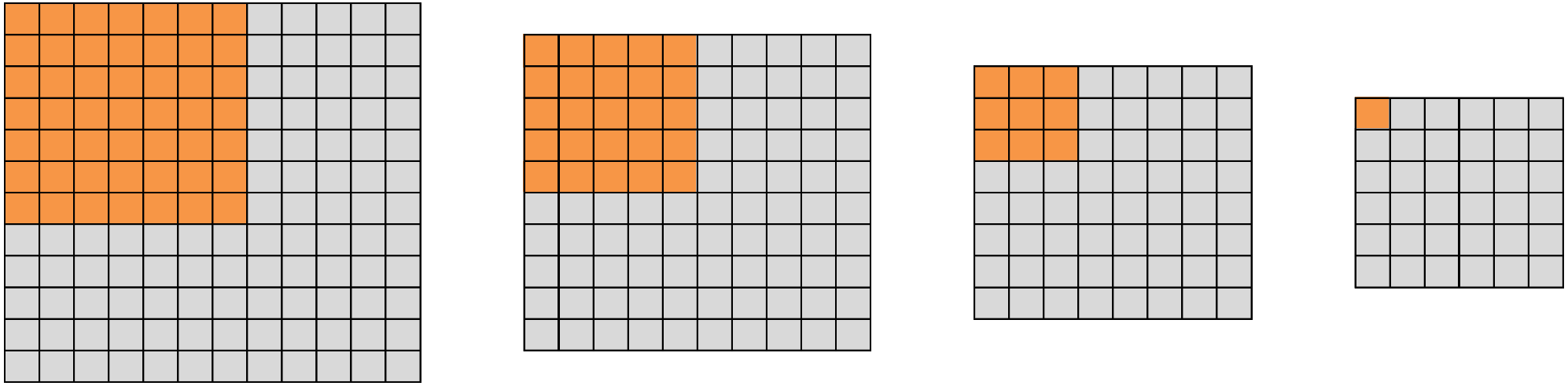
If we don’t use pooling



Input image | 1st conv layer | 2nd conv layer | 3rd conv layer

Each neuron in the 2nd conv layer contains information coming from a 5x5 window in the input image

If we don’t use pooling

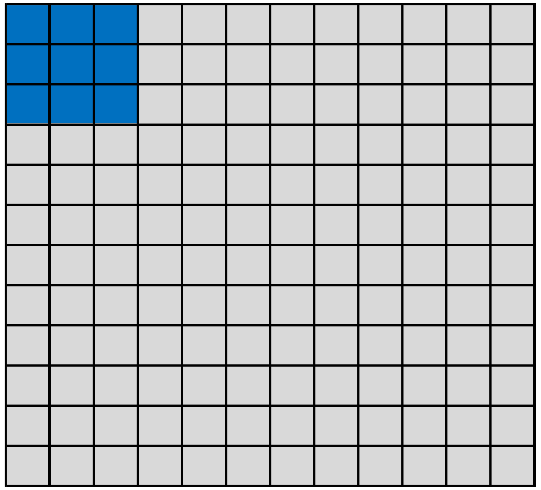


Input image | 1st conv layer | 2nd conv layer | 3rd conv layer

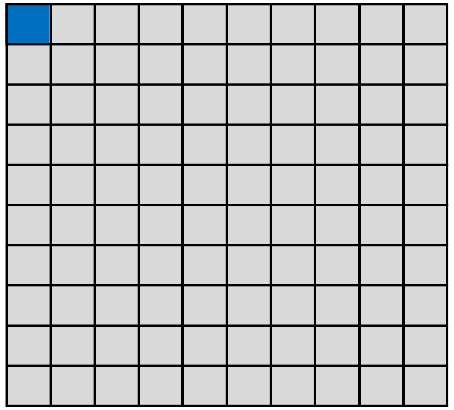
Each neuron in the 3rd conv layer contains information coming from a 7x7 window in the input image

If we use pooling

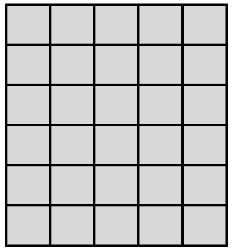
Let us consider that all filters are 3 x 3, stride = 1 and no padding is used



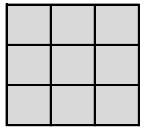
Input image



1st conv layer



Max pooling

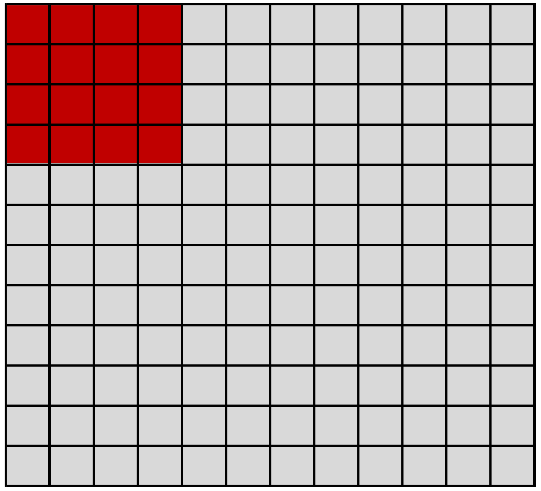


2nd conv layer

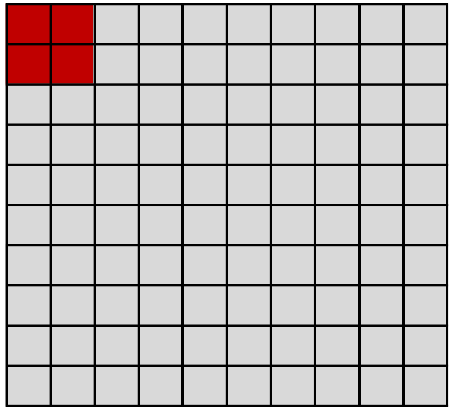
max pooling of 2 x 2 and stride 2

Each neuron in the 1st conv layer contains information coming from a 3x3 window in the input image

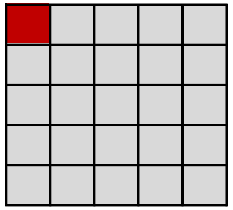
If we use pooling



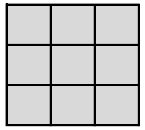
Input image



1st conv layer



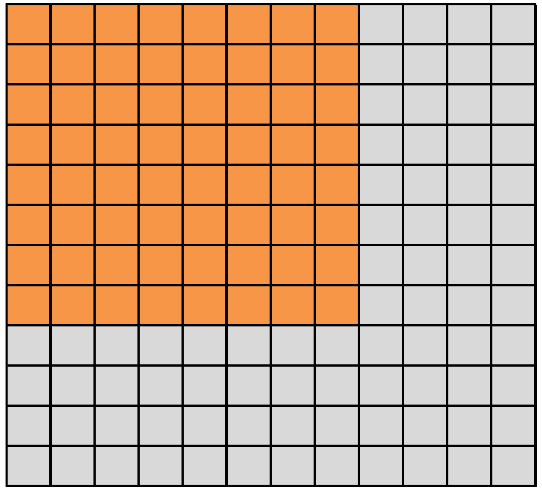
Max pooling



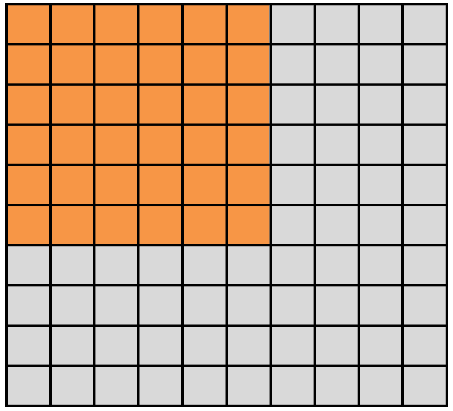
2nd conv layer

Each neuron in the max pooling layer contains information coming from a 4x4 window in the input image

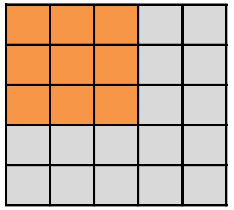
If we use pooling



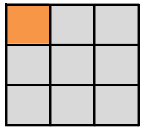
Input image



1st conv layer



Max pooling



2nd conv layer

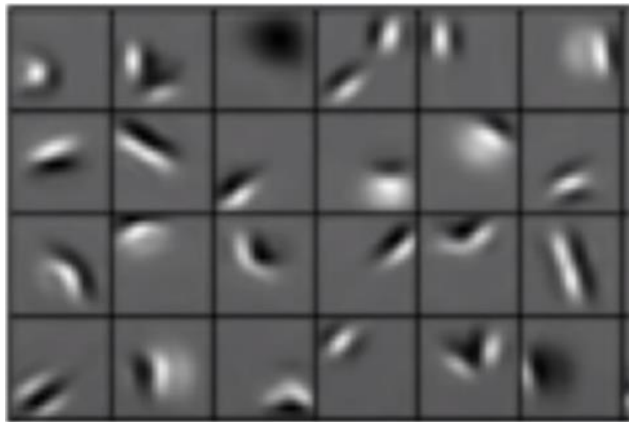
Each neuron in the 2nd conv layer contains information coming from an 8x8 window in the input image

Feature detection

Remember this slide?

- Can we automatically learn a hierarchy of features directly from the data instead of manual engineering them?

Low level features



Edges, dark spots,…

Mid level features

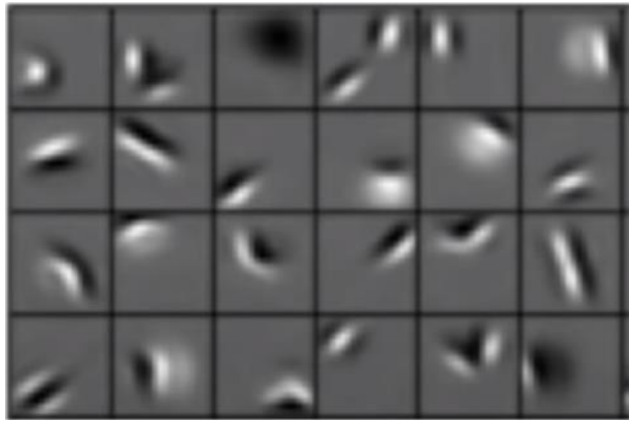
Eyes, ears, noses,…

High level features

Faces

Learning (a hierarchy of) features with CNNs

Low level features



Edges, dark spots,…

1st Conv layer

Mid level features

Eyes, ears, noses,…

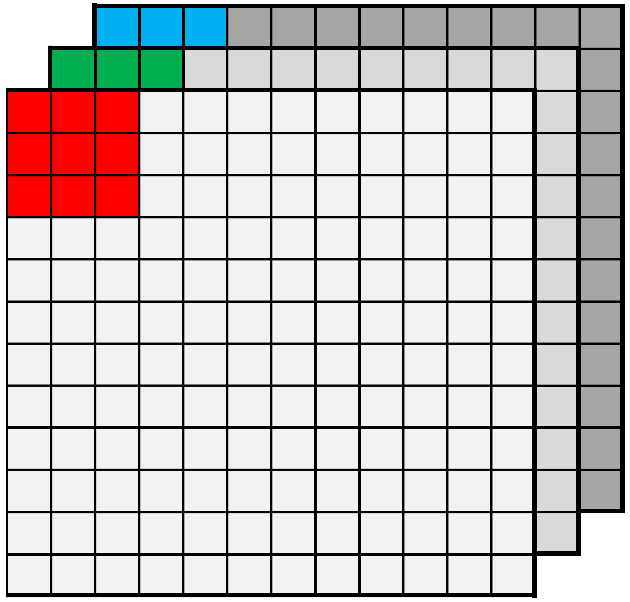
Some conv layers ahead

High level features

Faces

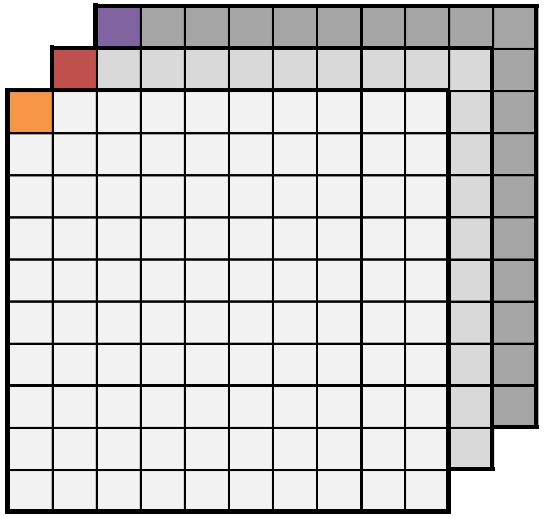
Last conv layer

How are features combined in higher-level ones?



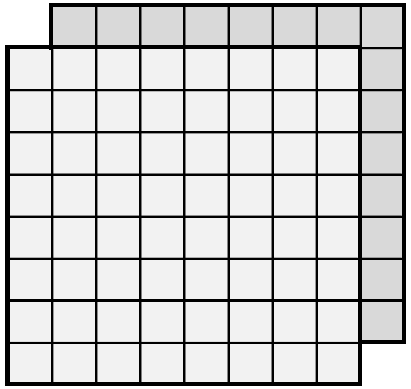
RGB image

3 feature maps

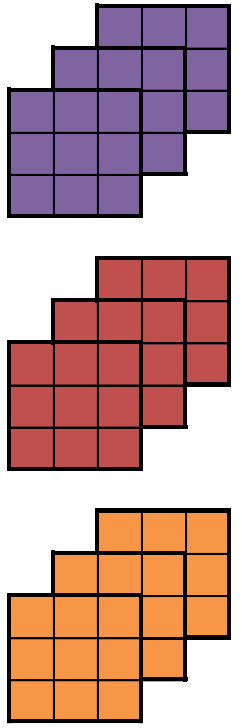


1st conv layer

2 feature maps



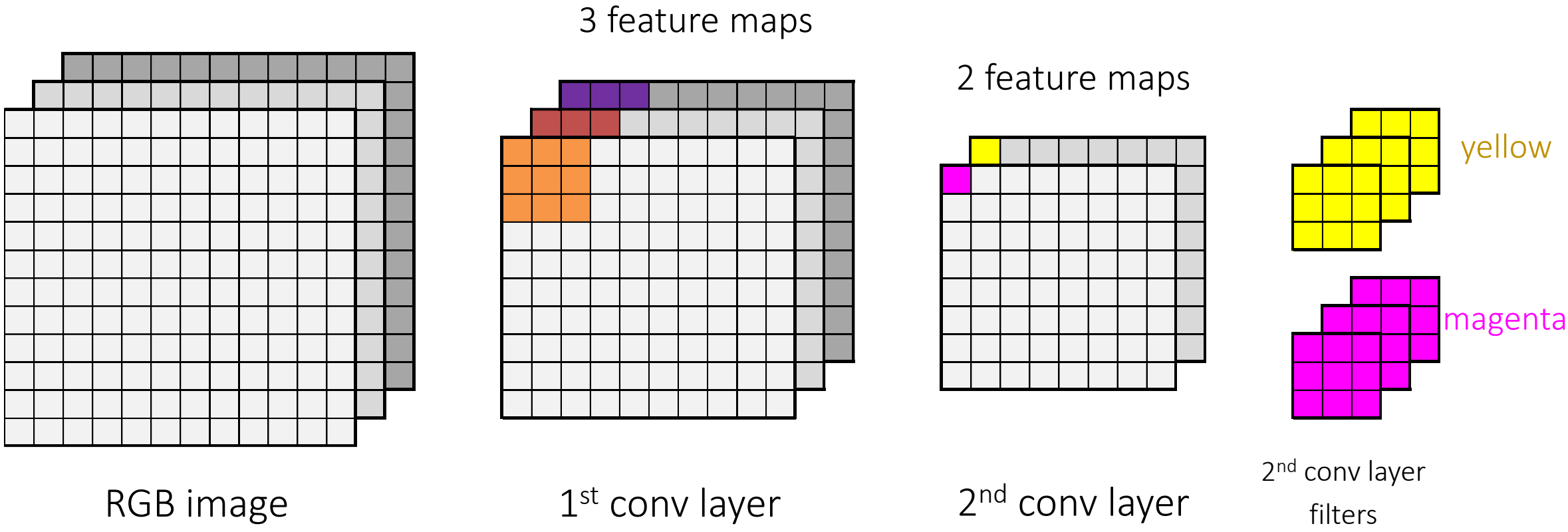
2nd conv layer



1st conv layer filters

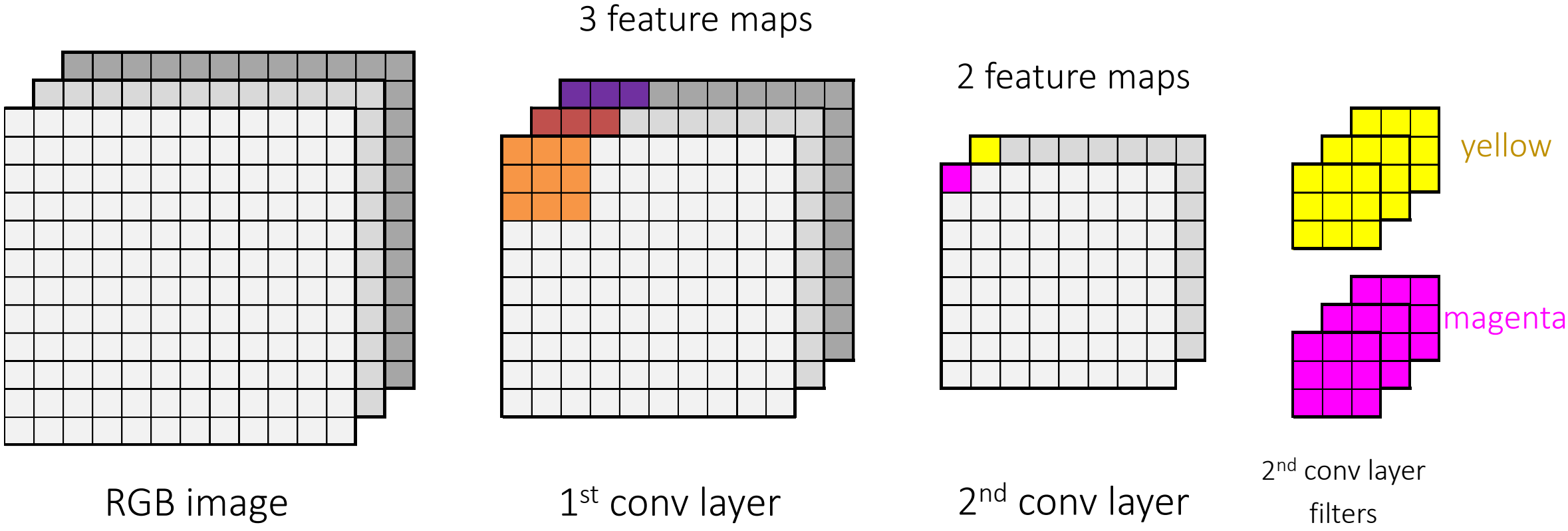
Remember: each filter/feature map corresponds to one feature

How are features combined in higher-level ones?



It may happen that the network learns a magenta filter that "fires" if the orange and purple features are present in the image

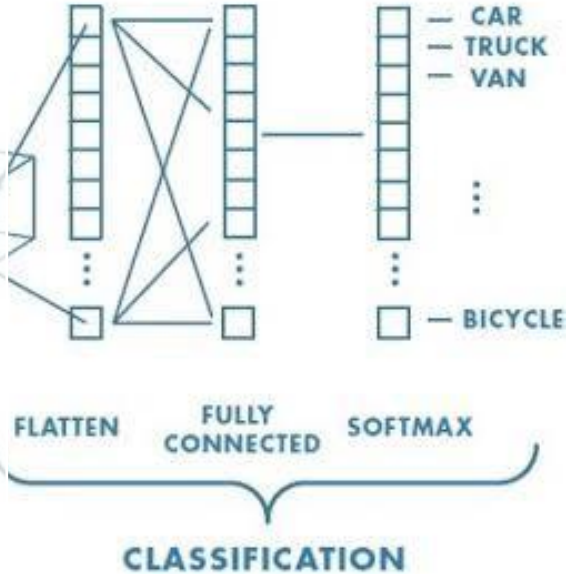
How are features combined in higher-level ones?



Or/and it may happen that the network learns a yellow filter that “fires” if the purple and brown features are present in the image

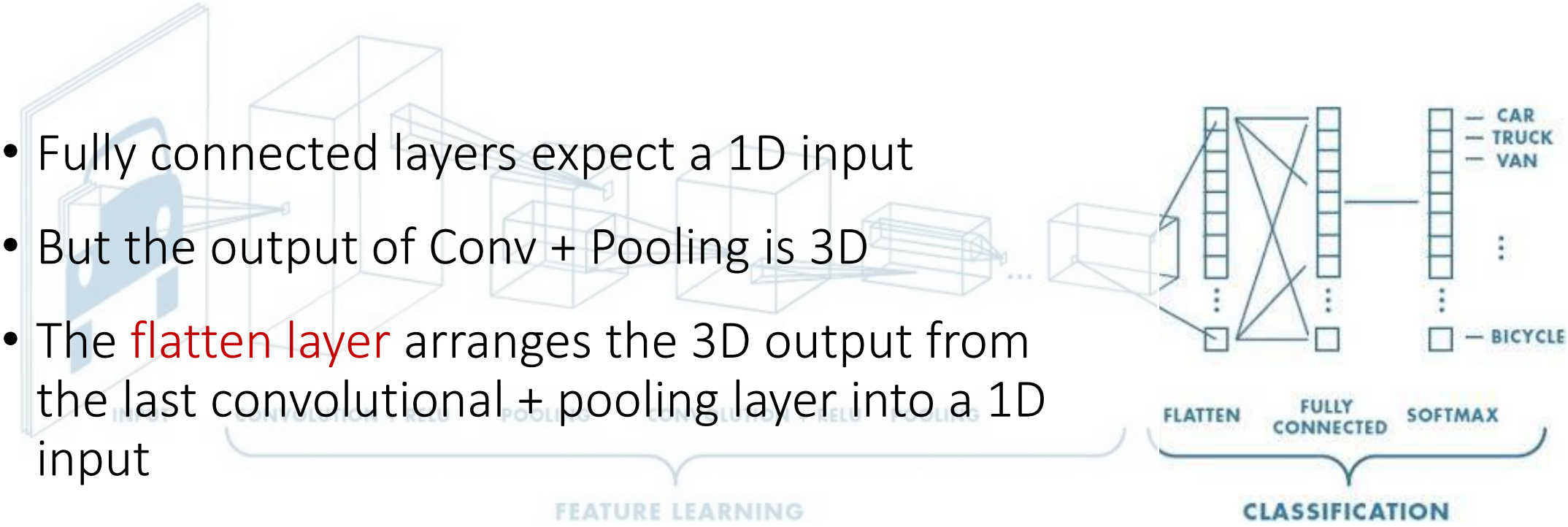
Classification in CNNs

- Convolutional + Pooling layers output high level features
- Fully connected layers use these features for classifying the input image
- They output the probability that the image belongs to each class
- This is done using the softmax function, if we have several classes, or the sigmoid function, if we have a binary classification problem



Classification in CNNs

- Fully connected layers expect a 1D input
- But the output of Conv + Pooling is 3D
- The flatten layer arranges the 3D output from the last convolutional + pooling layer into a 1D input



POLITÉCNICO DE LEIRIA

Bibliography

- Neural Networks and Deep Learning, Michael Nielsen
  - http://neuralnetworksanddeeplearning.com/
- Deep learning with Python, Francois Chollet, Manning Publications, 2018
- Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville, MIT Press, 2016
  - https://www.deeplearningbook.org/