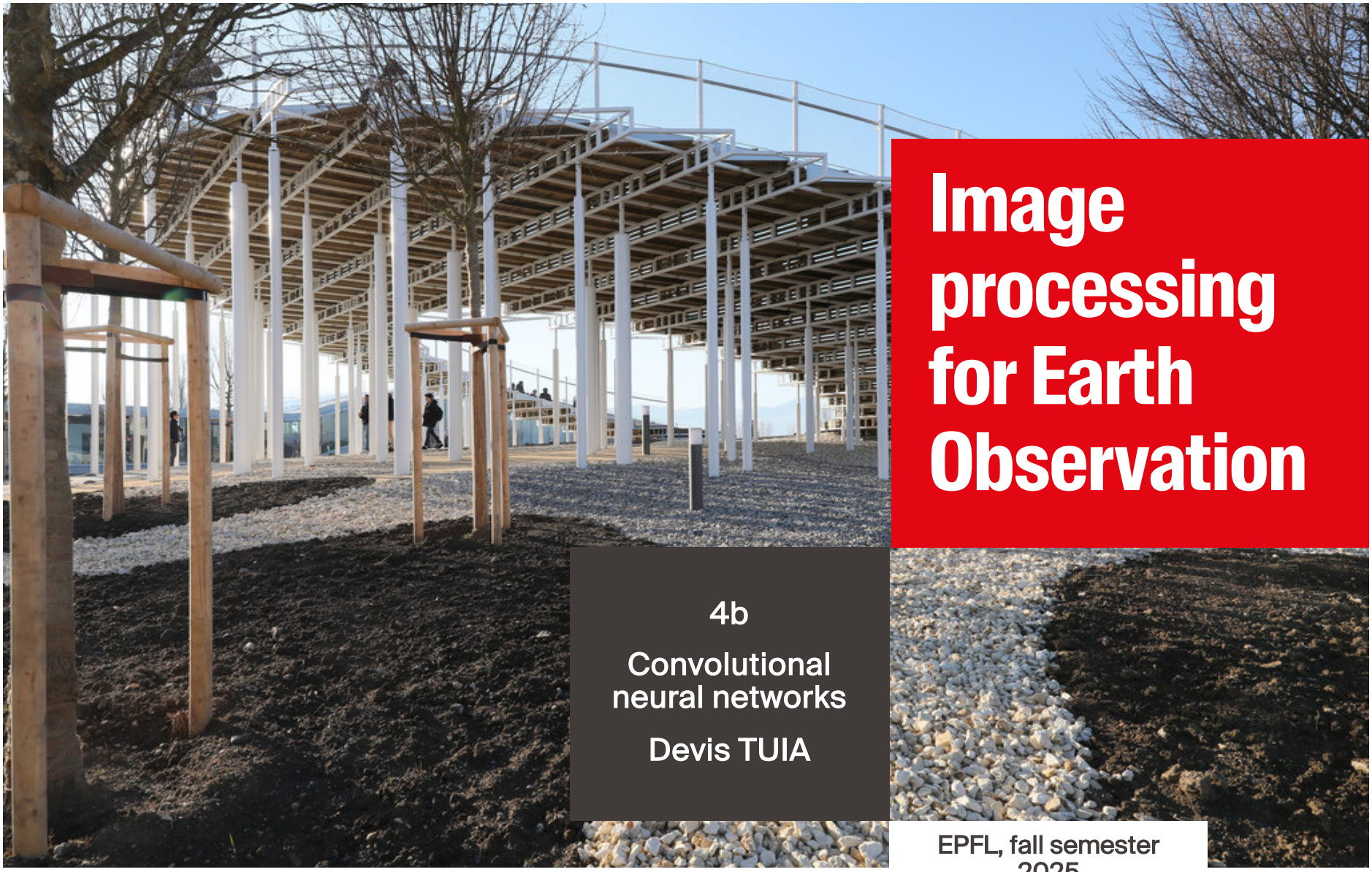


Image processing for Earth Observation

4b

Convolutional neural networks

Devis TUIA

EPFL, fall semester 2025

École polytechnique fédérale de Lausanne

Content (6 weeks)

- W1 General concepts of image classification / segmentation
  Traditional supervised classification methods (RF)
- W2 Traditional supervised classification methods (SVM)
  Best practices

- W3 Elements of neural networks
- W4 Convolutional neural networks
- W5 Convolutional neural networks for semantic segmentation
- W6 Sequence modeling, change detection

EPFL

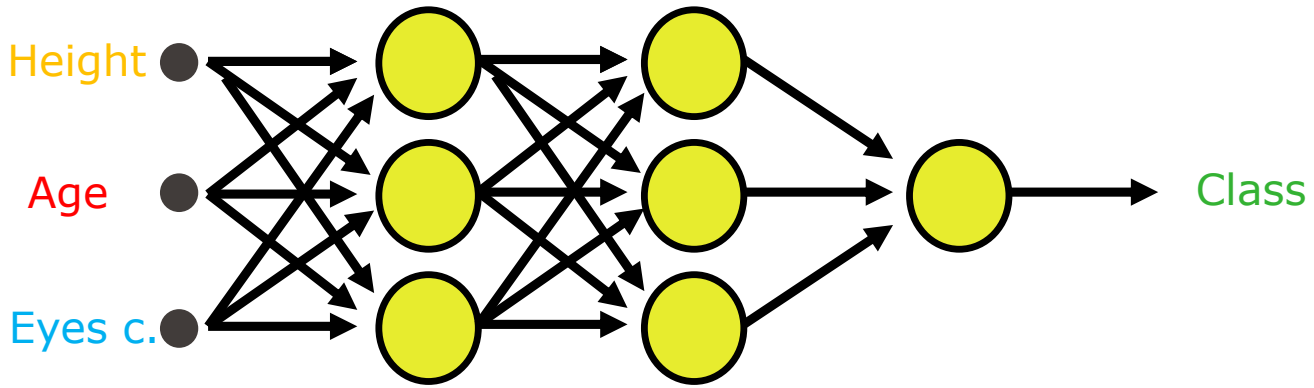
Convolutional neural networks

Larger inputs need weight sharing

From dense to convolutional layers

Padding, stride and pooling

MLPs consider feature interactions only

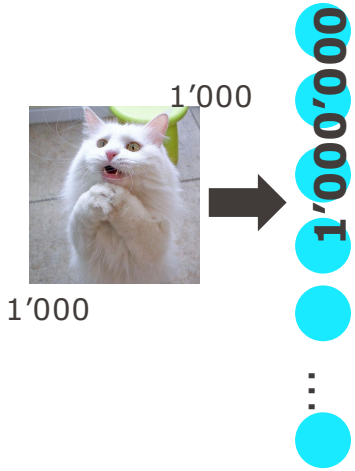


- MLPs consider each datapoint as a point in feature space
- It is often the best we can do if we have *tabular* data

ID	Height	Age	Eye color	class
1	65	7	blue	dog
2	34	3	brown	cat
3	55	12	brown	cat

When dealing with images...

- One could consider every pixel as a feature
- Ex: 1MPixel = 1000 x 1000 pixels = 1 million pixels*
- We could represent them as a table with a million features (1 per pixel*)

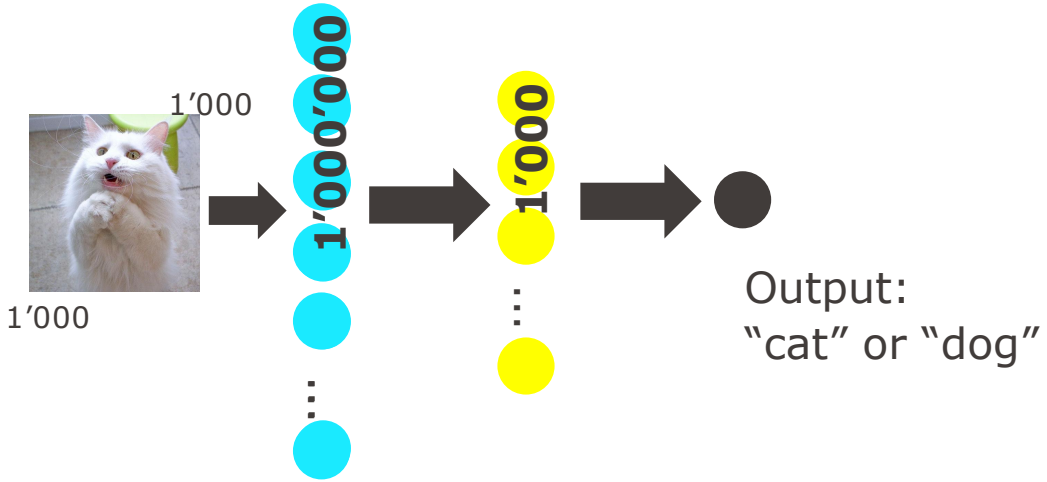


*This is not even true, a RGB image would have 3 millions pixels values, since there are 3 channels

But for the time being, let’s do as if it was grayscale.

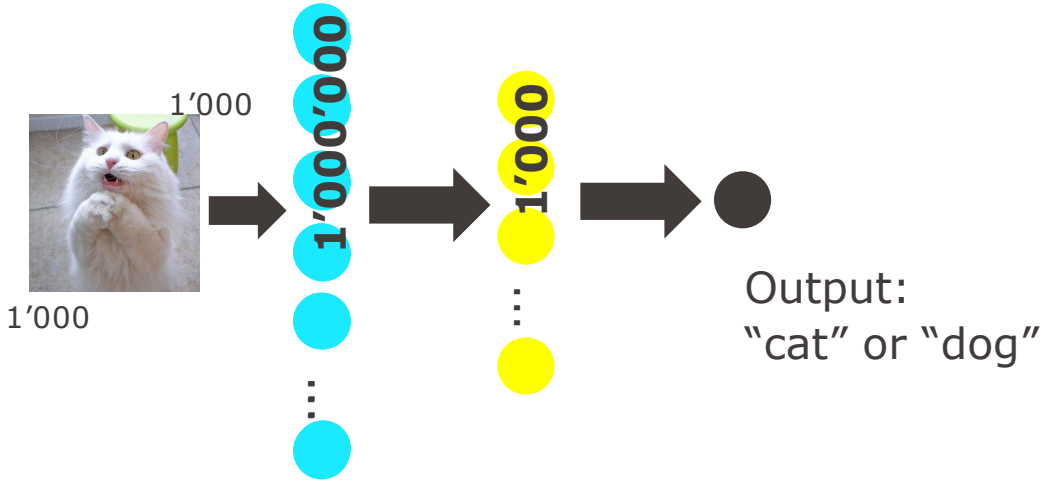
When dealing with images...

- One could consider every pixel as a feature
- Ex: 1MPixel = 1000 x 1000 pixels = 1 million pixels*
- We could represent them as a table with a million features (1 per pixel*)



When dealing with images...

- One could consider every pixel as a feature
- Ex: 1MPixel = 1000 x 1000 pixels = 1 million pixels*
- We could represent them as a table with a million features (1 per pixel*)



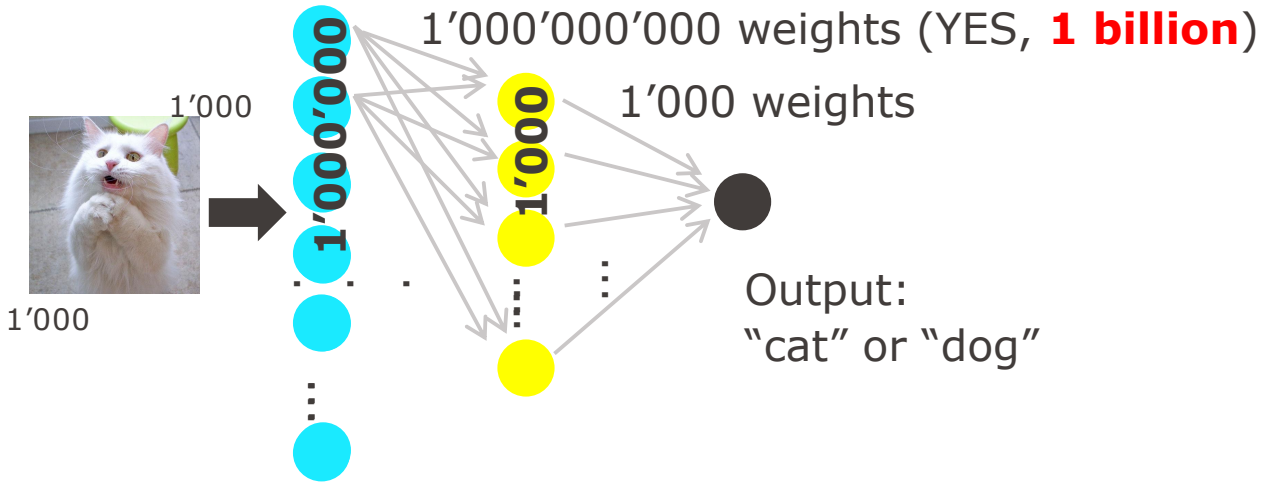
Looks feasible...

... wait a second...

... how many weights are we talking about?

When dealing with images...

- One could consider every pixel as a feature
- Ex: 1MPixel = 1000 x 1000 pixels = 1 million pixels*
- We could represent them as a table with a million features (1 per pixel*)



... it becomes a bigger problem.

- So we are talking 1 billion learnable weights.
- The value of a hidden neuron can be expressed as

$$h[i,j] = u[i,j] + \sum W[i,j,k,l] \cdot x[k,l]$$

- Where *i,j* and *k,l* are pixels locations in the image

... it becomes a bigger problem.

- So we are talking 1 billion learnable weights.
- The value of a hidden neuron can be expressed as

$$h[i,j] = u[i,j] + \sum_{k,l} W[i,j,k,l] \cdot x[k,l]$$

Meaning: each feature in the hidden layer *h* depends on **ALL** locations of the input image (hint: look at the sum on k,l).

- Where *i,j* and *k,l* are pixels locations in the image
  - we want to express a pixel (*i,j*) as a combination of all other pixels (*k,l*)
  - each neuron *h* is specific to one pixel location *i,j*

... it becomes a bigger problem.

- So we are talking 1 billion learnable weights.
- The value of a hidden neuron can be expressed as

$$h[i,j] = u[i,j] + \sum_{k,l} W[i,j,k,l] \cdot x[k,l]$$

$$= u[i,j] + \sum_{a,b} V[i,j,a,b] \cdot x[i+a, j+b]$$

- Where *i,j* and *a,b* are pixels locations in the image
  - *k = i+a* (we just re-express by relative positions)
  - *l= j+b*

... it becomes a bigger problem.

- So we are talking 1 billion learnable weights.
- The value of a hidden neuron can be expressed as

$$h[i,j] = u[i,j] + \sum_{k,l} W[i,j,k,l] \cdot x[k,l]$$

$$= u[i,j] + \sum_{a,b} V[i,j,a,b] \cdot x[i+a, j+b]$$

Final value of the feature (specific to a location *i,j*)

Bias (specific to a location *i,j*)

Filter weights (arrows in slide 8) specific to pairs of locations

Original pixel values at a location (*i+a,j+b*)

... it becomes a bigger problem.

- So we are talking 1 billion-ish learnable weights.
- To learn this, you would need m(b)illions of examples and a lot of computational power
- The population of cats and dogs on the planet is around 1.5 billions...
- We need to be more clever than that.

- What can we do?

Evidence #1
Location is NOT so important

- Our visual system recognizes objects independently of their specific location in the image.
- You will re-use the same filters at different locations

Source: petguide, boredpanda, countryliving

Evidence #1
Location is NOT so important

- This can simply be implemented in the neuron equation by removing dependence of the weights on location:

$$h[i,j] = u[i,j] + \sum_{a,b} V[i,j,a,b] \cdot x[i+a, j+b]$$

$$h[i,j] = u + \sum_{a,b} V[a,b] \cdot x[i+a, j+b]$$

- This implements **translation invariance** and is the main reason why CNNs are nowadays trainable.

Evidence #1
Location is NOT so important

- In other words (example involving a 4 x 5 pixels image):

FROM 400

Learnable weights

WEIGHTS MULTIPLYING THE GREEN PIXEL

w1	w2	w3	w4	w5
w6	w7	w8	w9	w10
w11	w12	w13	w14	w15
w16	w17	w18	w19	w20

w21	w22	w23	w24	w25
w26	w27	w28	w29	w30
w31	w32	w33	w34	w35
w36	w37	w38	w39	w40

...

w381	w382	w383	w384	w385
w386	w387	w388	w389	w390
w391	w392	w393	w394	w395
w396	w397	w398	w939	w400

TO 20!

WEIGHTS MULTIPLYING THE GREEN PIXEL

w1	w2	w3	w4	w5
w6	w7	w8	w9	w10
w11	w12	w13	w14	w15
w16	w17	w18	w19	w20

w1	w2	w3	w4	w5
w6	w7	w8	w9	w10
w11	w12	w13	w14	w15
w16	w17	w18	w19	w20

...

w1	w2	w3	w4	w5
w6	w7	w8	w9	w10
w11	w12	w13	w14	w15
w16	w17	w18	w19	w20

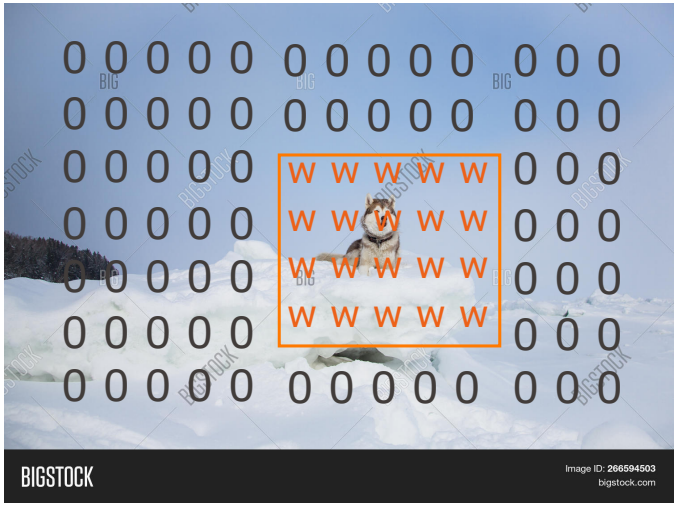
Evidence #2
Global context is NOT so important

- Our visual system recognizes objects (mostly) by looking at local features.
- You don't need to encode neuron dependencies between far away parts of the image

Source: bigstock, petguide, boredpanda, countryliving

Evidence #2
Global context is NOT so important

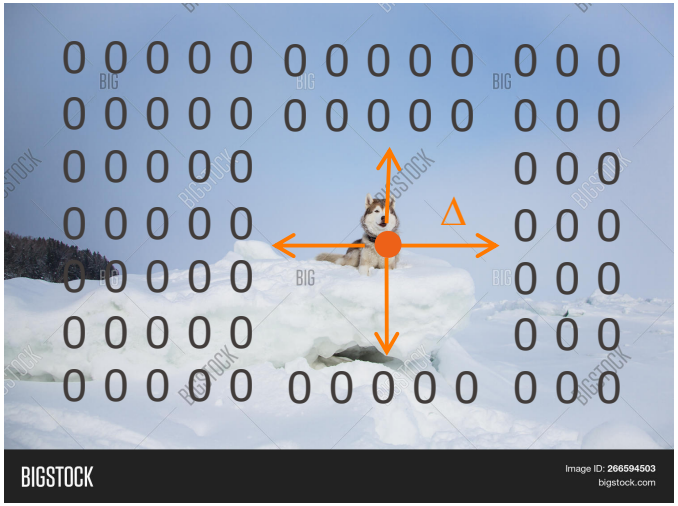
- In other words, the relevant information must be close to location *i,j*
- The weights should be nonzero only in the vicinity of the pixel you are considering



Source: bigstock, petguide, boredpanda, countryliving

Evidence #2
Global context is NOT so important

- In other words, the relevant information must be close to location *i,j*
- The weights should be nonzero only in the vicinity of the pixel you are considering
- If we consider a neighborhood of size 2Δ x 2Δ



Source: bigstock, petguide, boredpanda, countryliving

Evidence #2
Global context is NOT so important

- In other words, the relevant information must be close to location *i,j*
- The weights should be nonzero only in the vicinity of the pixel you are considering

$$h[i,j] = u + \sum_{a,b} V[a,b] \cdot x[i+a, j+b]$$

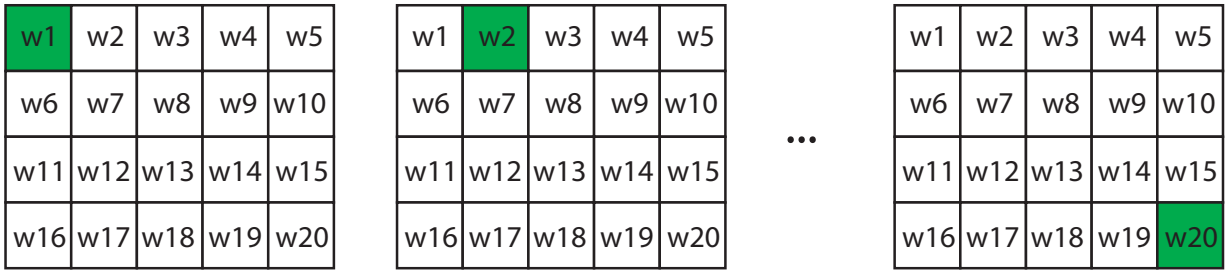
$$h[i,j] = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} V[a,b] \cdot x[i+a, j+b]$$

Evidence #2
Global context is NOT so important

- In other words (example involving a 4 x 5 pixels image and Δ=1):

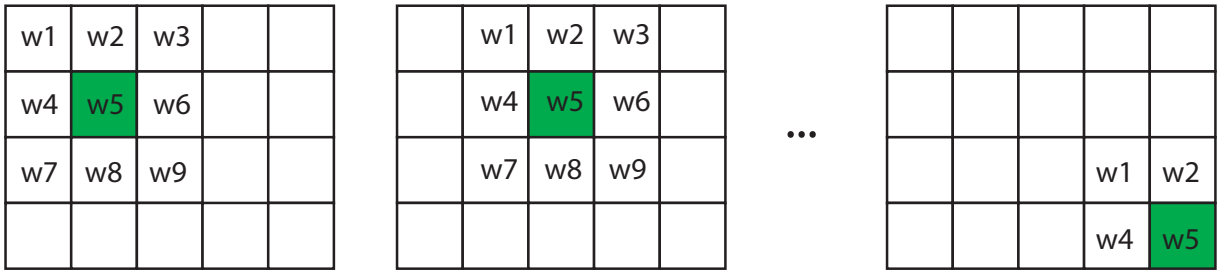
FROM 20
Learnable weights

WEIGHTS MULTIPLYING THE GREEN PIXEL



TO 9!

WEIGHTS MULTIPLYING THE GREEN PIXEL



This is called a (2D) convolution

- The weights are organized as a filter, slid over the image

- In the original example of cats and dogs:

  - Location-specific : (i x j x a x b) weights: 10^9
  - Translation invariant : (a x b) weights: 10^6
  - Local windows : ((2Δ+1)^2) weights: **9 * 1000**

Exercise

- Calculate the result of the following convolutional filters (*W*) when applied to an image (*I*):

I

3	1	4	3
3	2	3	1
4	3	6	4
3	3	1	7

W_1

1	-1	2
3	1	4
5	-2	9

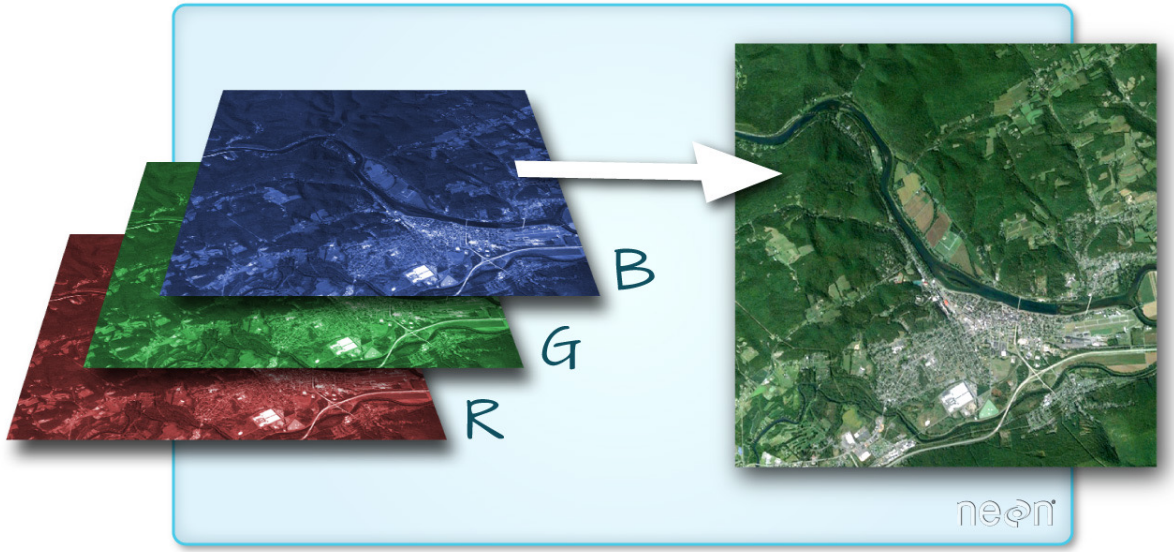
W_2

1	1	1
1	1	1
1	1	1

- What is the dimension of the output? Any idea why?

Images are 3D tensors though

- Images have (at least) 3 bands: Red, Green, Blue



Images are 3D tensors though

- Images have (at least) 3 bands: Red, Green, Blue
- The result of a 2D convolution is the sum of 2D filters over the single input bands:

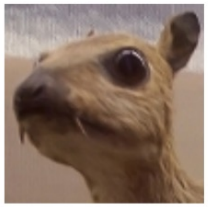
$$h[i,j] = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} V[a,b] \cdot x[i+a, j+b]$$

$$h[i,j] = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} \sum_{c} V[a,b] \cdot x[i+a, j+b, c]$$

But it's just sliding windows...

- Yes, but
- Different filters have different effects
- This time, the filters are **learned**
  - We don't know them in advance
  - We learn the weights by gradient descent

Examples

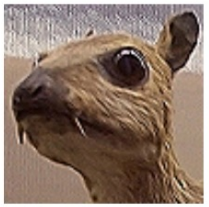


(wikipedia)

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

Edge Detection

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



Sharpen

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Gaussian Blur

courses.d2l.ai/berkeley-stat-157

Learning more filters

Attention! *k* here is a filter index, not the position index as in previous slides...

- Convolutional filters have different effects
- So we want to learn more than 1 filter
- For *k filters:*

$$h[i,j] = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} \sum_{c} V[a,b] \cdot x[i+a, j+b, c]$$

$$h[i,j,k] = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} \sum_{c} V[a,b,k] \cdot x[i+a, j+b, c]$$

EPFL

Convolutional neural networks

Larger inputs need weight sharing

From dense to convolutional layers

Padding, stride and pooling

Convnets are a bit like lego playing

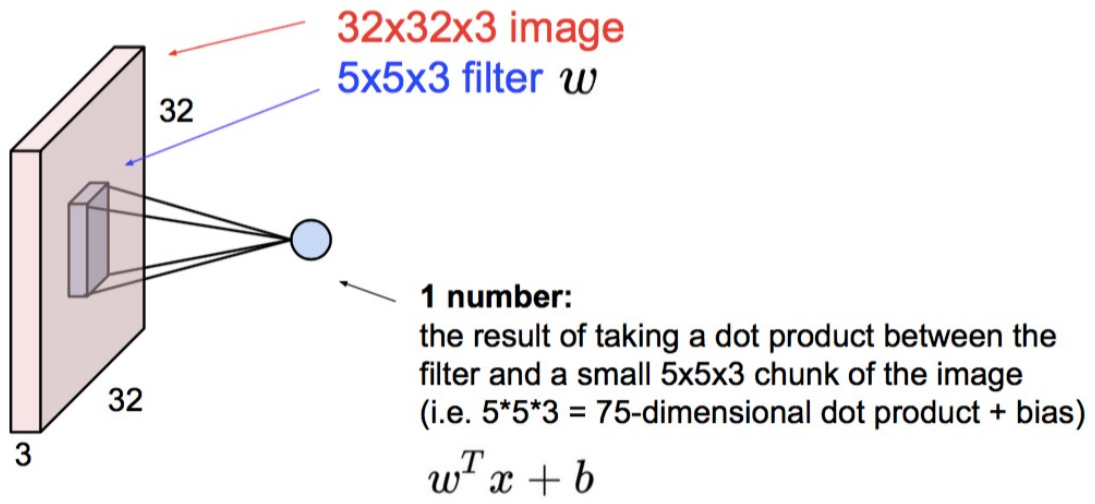
- Now you know how to compute a convolutional filter
- A convolutional network is a stack of such convolutions

http://www.inference.vc/deep-learning-is-easy/

Convolutional neural networks

In convolutional neural networks, the filters are spatial (on 2D grids).

- local : they convolve the values of the image in a local window



[From Wegner, ETH]

Convolutional neural networks

In convolutional neural networks, the filters are spatial (on 2D grids).

- local : they convolve the values of the image in a local window
- the convolution is followed by a nonlinearity to make the result nonlinear.

- your nonlinearity can be a sigmoid or (easier to compute) a Rectified Linear Unit (RELU):

$$f(x) = \max(f(x), 0)$$

Convolutional neural networks

In convolutional neural networks, the filters are spatial (on 2D grids).

- local : they convolve the values of the image in a local window
- shared: the same filter is applied everywhere in the image, leading to an **activation map**

[From Wegner, ETH]

Convolutional neural networks

In convolutional neural networks, the filters are spatial (on 2D grids).

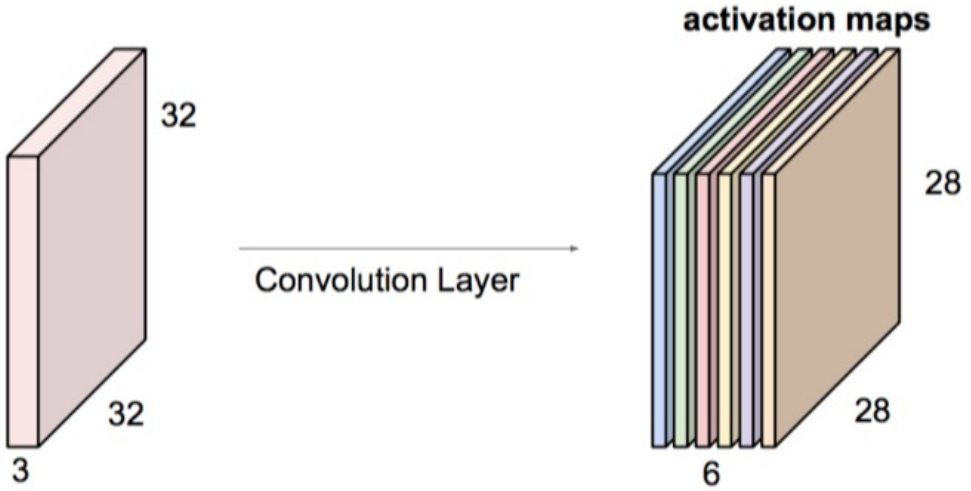
- local : they convolve the values of the image in a local window
- shared: the same filter is applied everywhere in the image, leading to an activation map
- **many filters are learned in parallel**

[From Wegner, ETH]

Convolutional neural networks

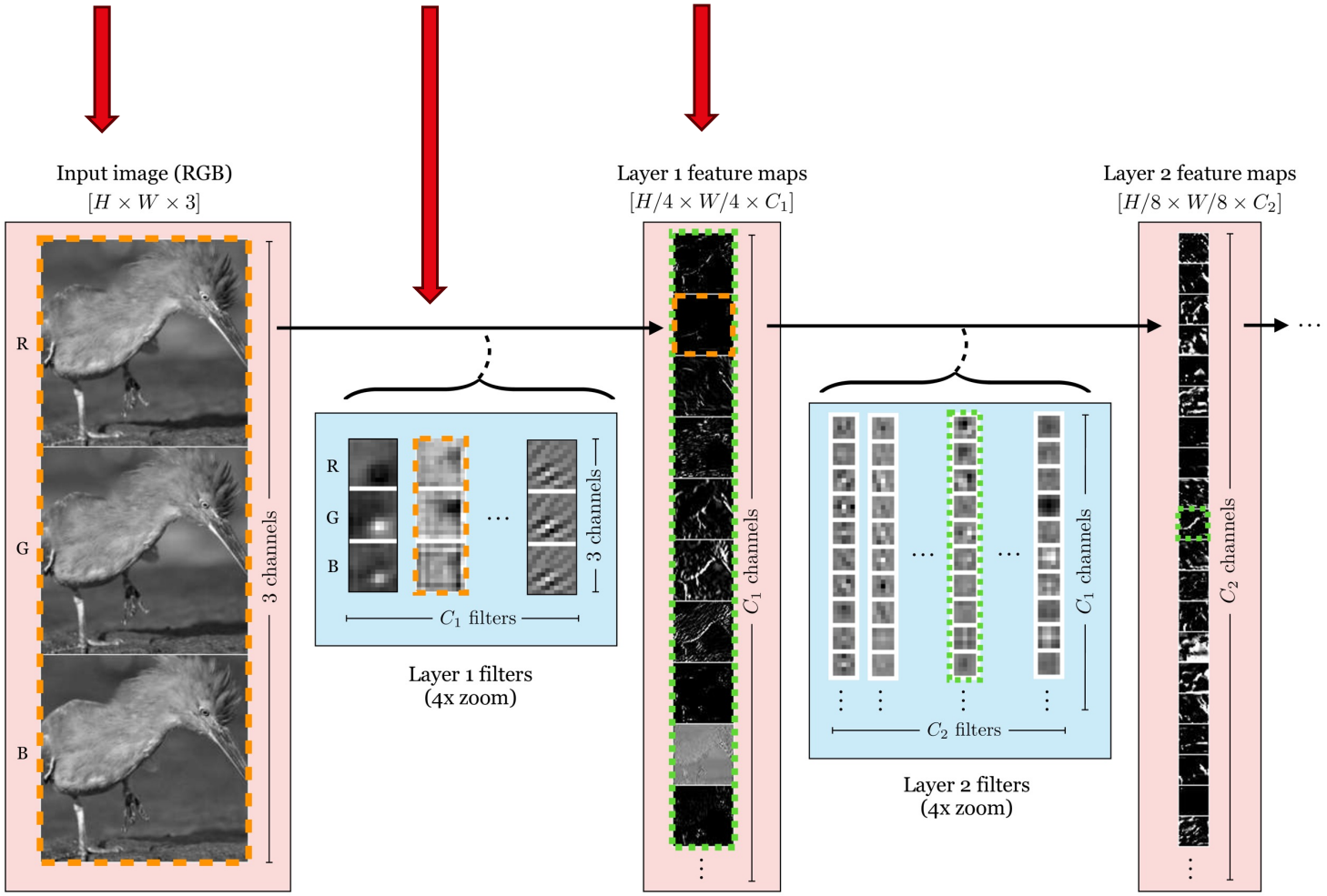
In convolutional neural networks, the filters are spatial (on 2D grids).

- local : they convolve the values of the image in a local window
- shared: the same filter is applied everywhere in the image, leading to an activation map
- **many filters are learned in parallel (in this case 6)**



[From Wegner, ETH]

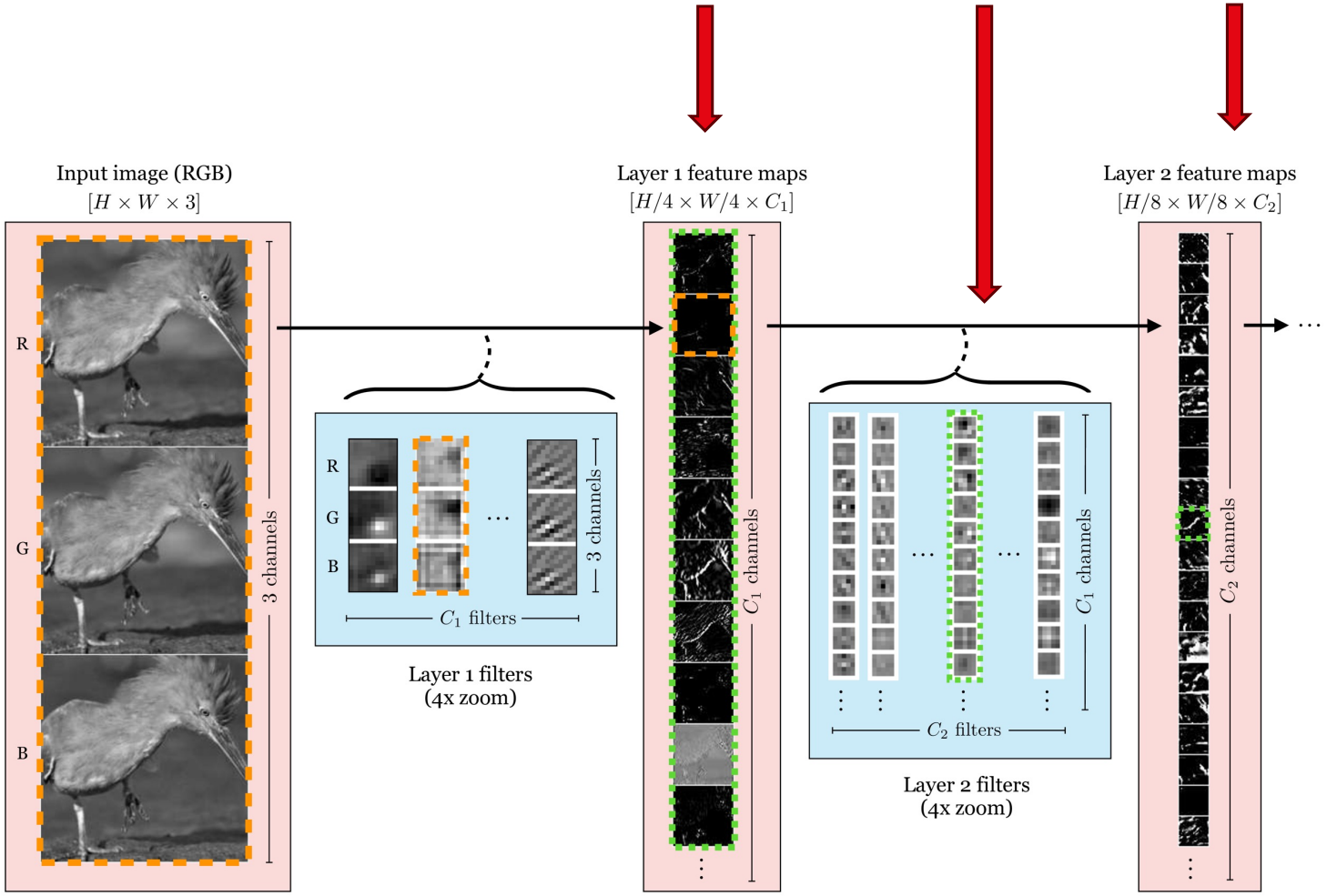
Image → filters → activation maps (1st "layer")



https://visionbook.mit.edu/convolutional_neural_nets.html

(2nd "layer")

Image → filters → activation map



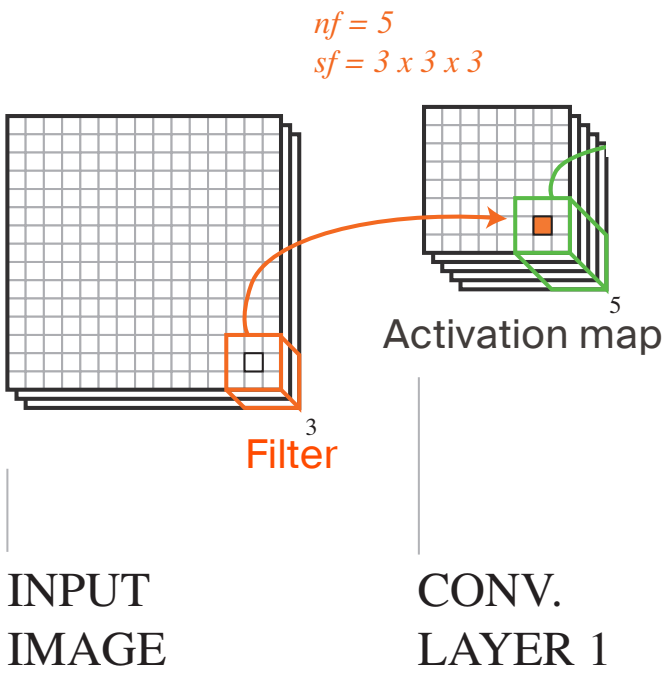
https://visionbook.mit.edu/convolutional_neural_nets.html

Convnets are a bit like lego playing

- The thing is: you need to be sure of the sizes of your activation maps
- Like Legos, they need to fit to eachother

http://www.inference.vc/deep-learning-is-easy/

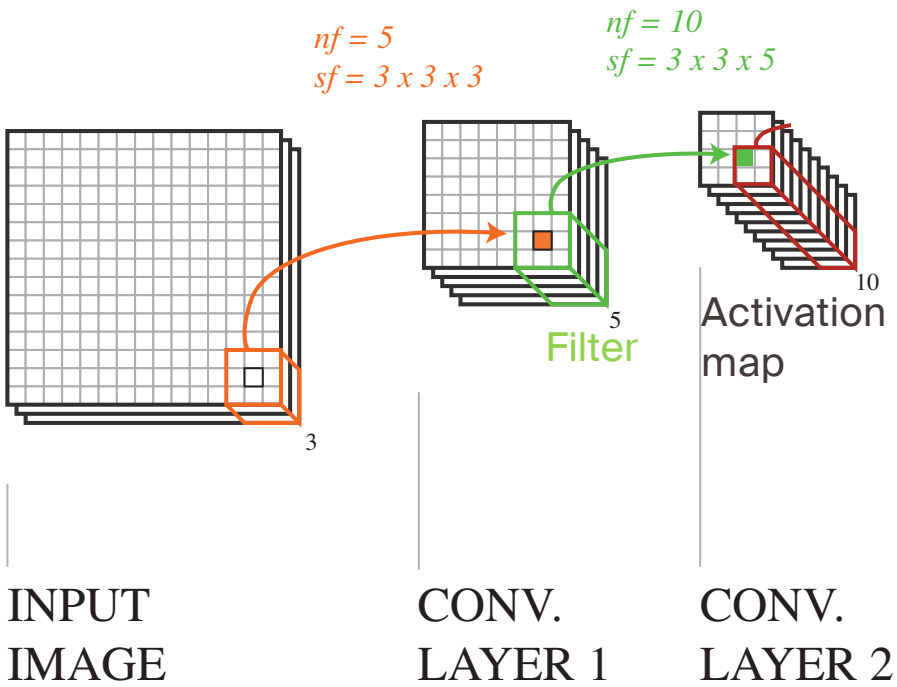
Convnets are a bit like lego playing



In this exemple, for the first layer:

- You use a sliding window conv filter, in this case 3 x 3 x 3 (3 rows, 3 columns, 3 bands, the little orange cube)
- Each central pixel location gets summarized into a single value in the activation map (the orange pixel in the result);
- In this exemple, we learned nf=5 different filters, which gives you a results with 5 “bands” (the green little cube);
- This activation map will become the input for the next layer.

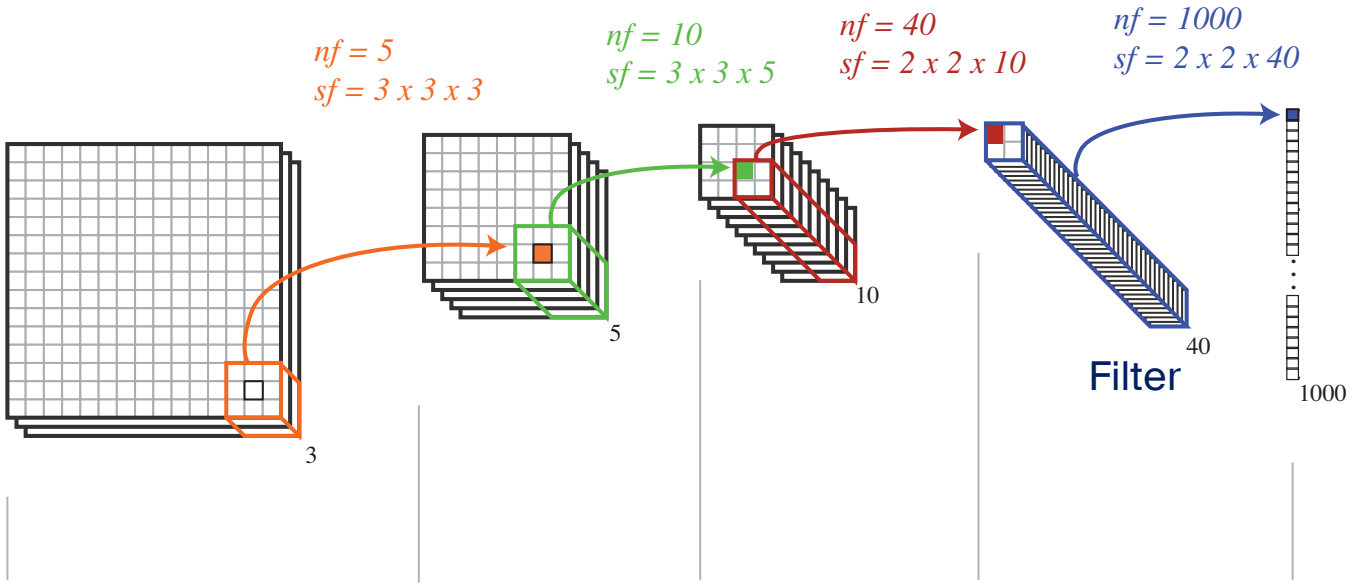
Convnets are a bit like lego playing



In this exemple, for the second layer:

- You use a sliding window conv filter, in this case 3 x 3 x 5, because your input activation map has 5 bands (the little green cube)
- Each central pixel location gets summarized into a single value in the activation map (the green pixel in the result);
- In this exemple, we learned nf=10 different filters, which gives you a results with 10 “bands” (the green little cube);
- This activation map will become the input for the next layer.

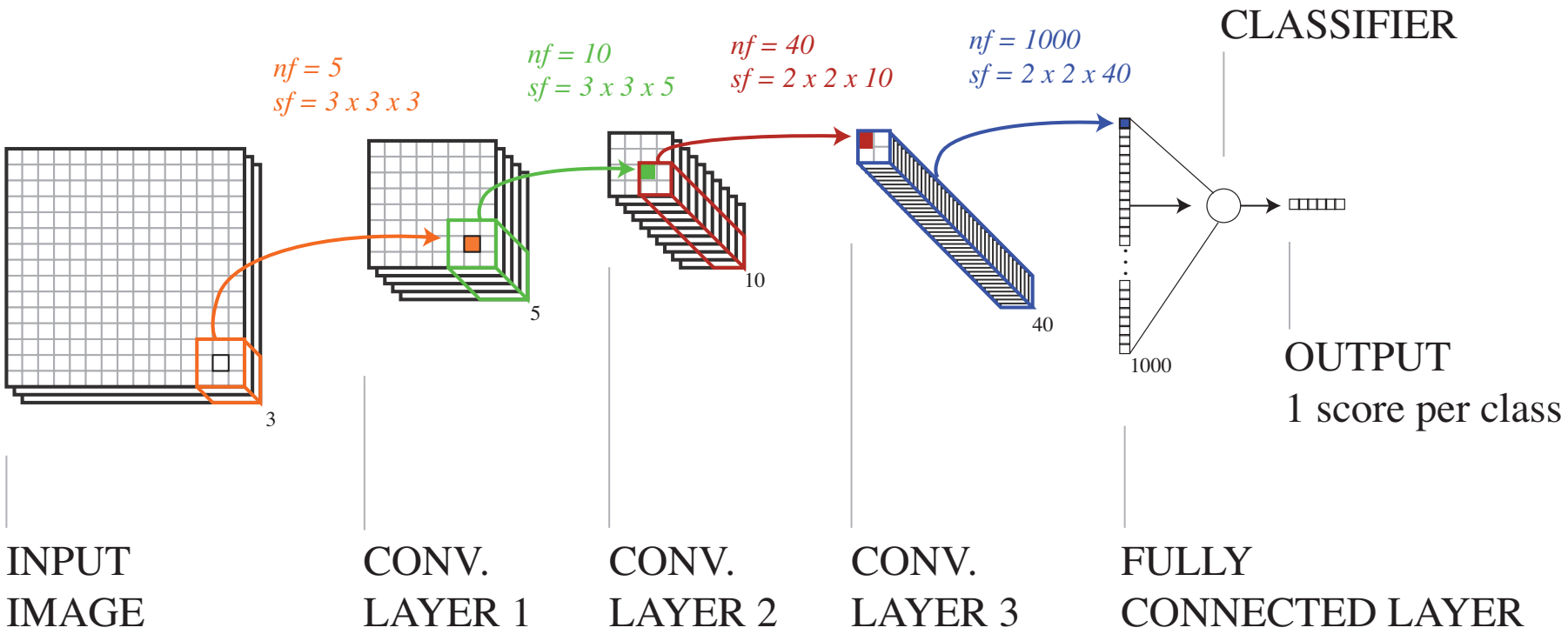
Convnets are a bit like lego playing



In this exemple, for the last layer, the “fully connected layer”:

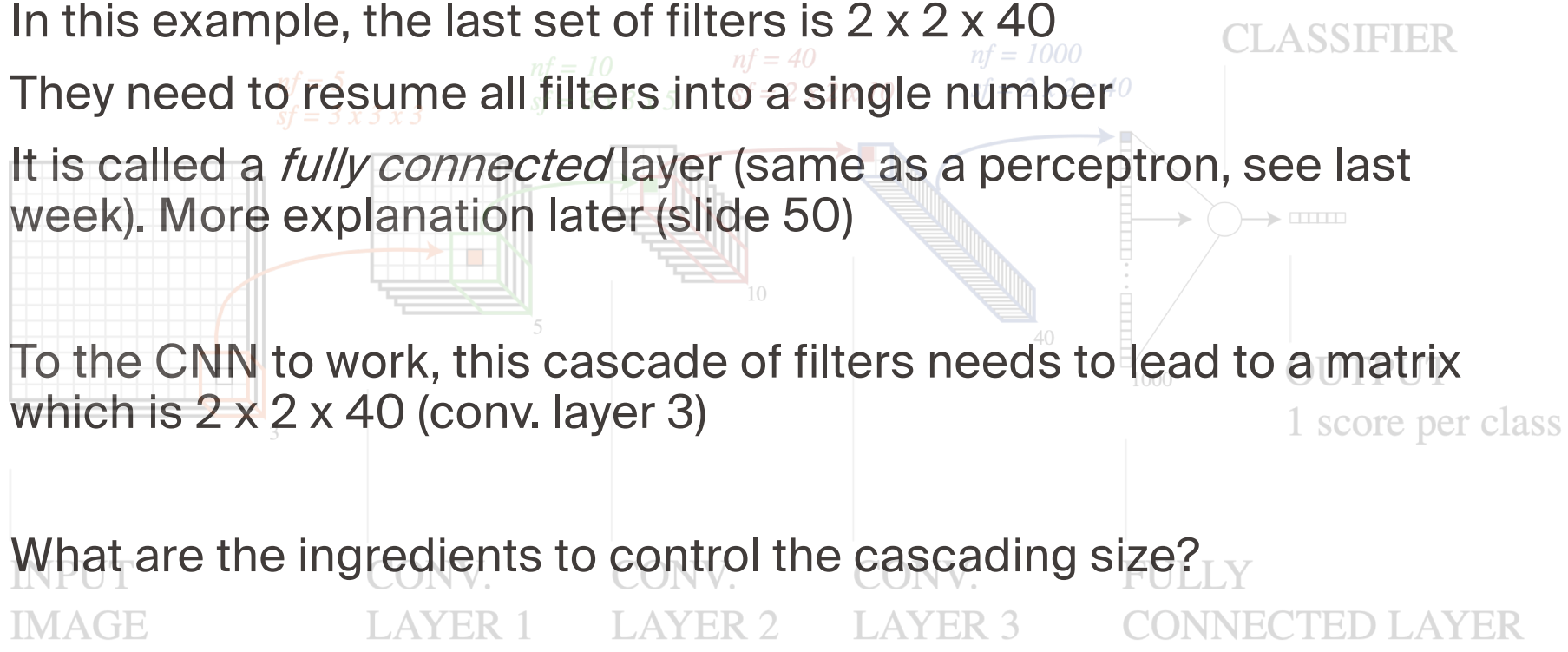
- You use a conv filter that summarizes the whole activation map 2 x 2 x 40 (the little blue cube) into a single value (the blue square);
- In this exemple, we learned nf=1000 different filters, which gives you a results with 1000 values (the final vector on the right);
- This vector will become the input for the MLP classifier.

Convnets are a bit like lego playing



Convnets are a bit like lego playing

- In this example, the last set of filters is 2 x 2 x 40
- They need to resume all filters into a single number
- It is called a *fully connected* layer (same as a perceptron, see last week). More explanation later (slide 50)



- To the CNN to work, this cascade of filters needs to lead to a matrix which is 2 x 2 x 40 (conv. layer 3)

- What are the ingredients to control the cascading size?

Ingredient #1 padding

I

3	1	4	3
3	2	3	1
4	3	6	4
3	3	1	7

W_1

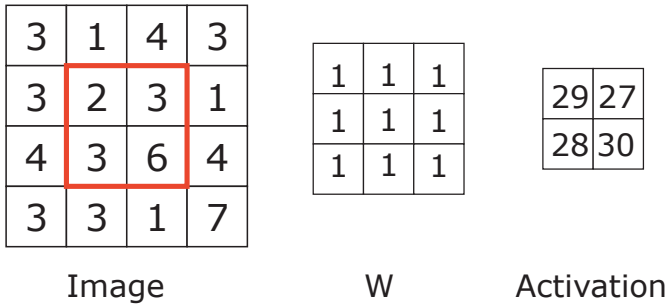
1	-1	2
3	1	4
5	-2	9

- In the previous exercise (slide 23), the resulting activation map was of size 2 x 2
- The convolution by W_1 could only consider the four pixels centered in the red square
- With an input image of size (n_h x n_w) and a filter (k_h x k_w), the output size is:

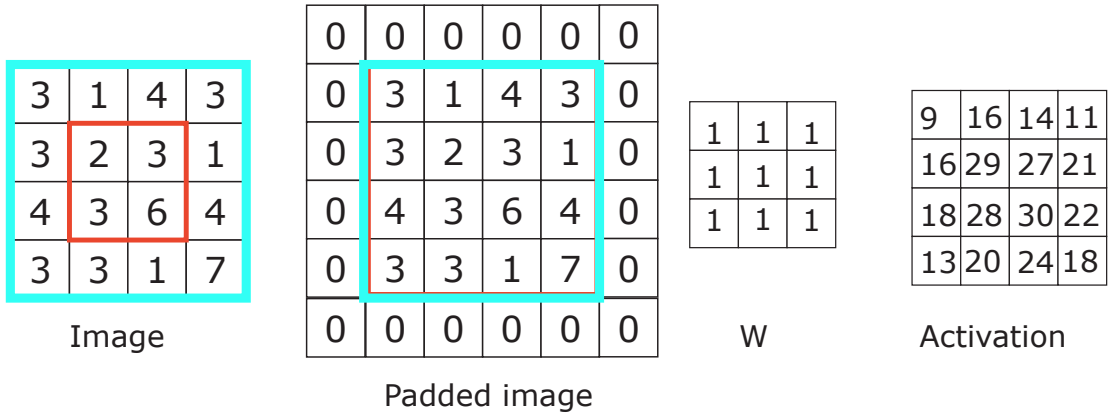
$$(n_h - k_h + 1) \times (n_w - k_w + 1)$$

Ingredient #1 padding

- Without padding



- With padding

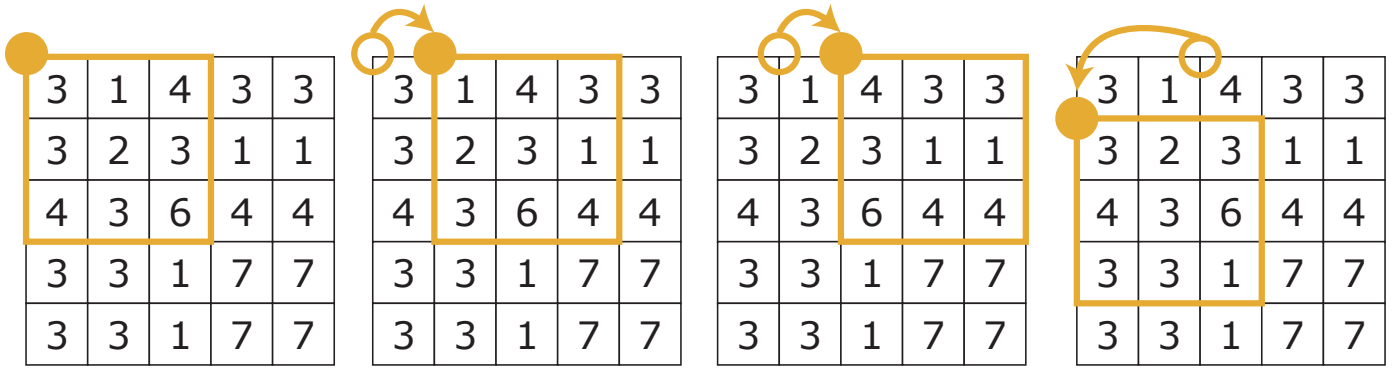


Ingredient #2 stride

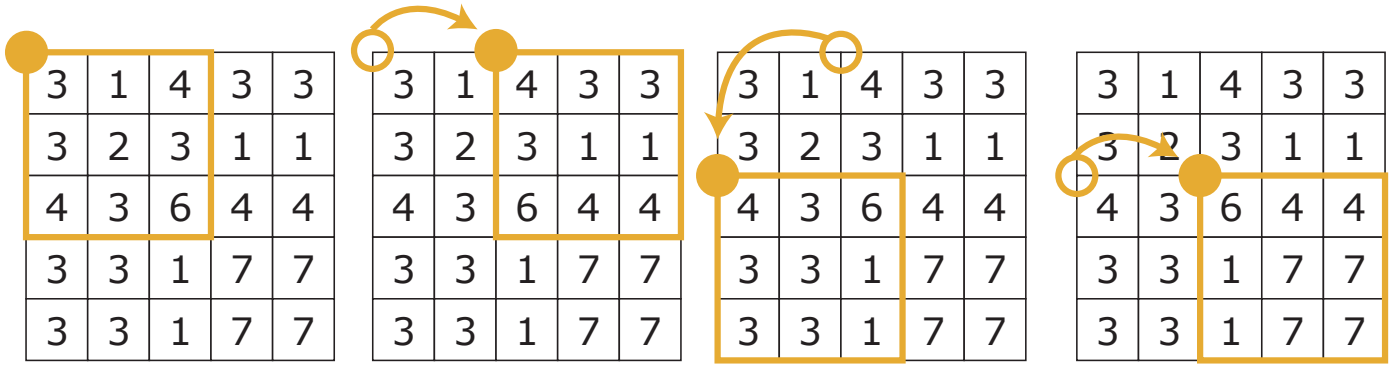
- By default, convolutions scan the image from top left to bottom right, one pixel at a time
- We call this *stride of 1*
- Sometimes, to reduce resolution or need less filters, we can compute only one every *k* pixels, which leads to *stride k*
- *If you use stride = 2*, the activation map size is half of the original image
- Different strides in row and column are rarely used.

Ingredient #2 stride

Filter 3 x 3, Padding = 0, Stride 1. Activation map size: 3 x 3

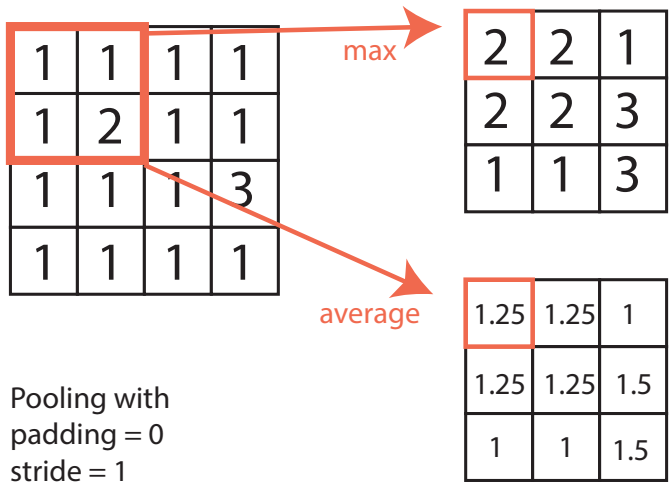


Filter 3 x 3, Padding = 0, Stride 2. Activation map size: 2 x 2



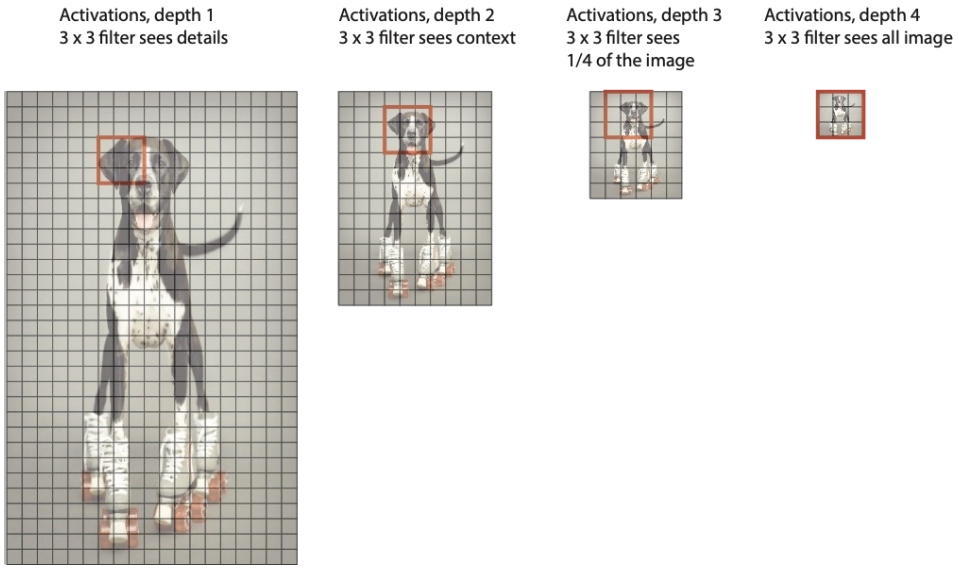
Ingredient #3 pooling

- Pooling is an operation that decreases the size of the activation map
- It seems similar to stride (it reduces size), but *selects the information to carry over*, rather than skipping calculations
- Pooling works by applying a function locally in an activation map:



Ingredient #3 pooling

- Pooling has desirable effects:
  - It increases the receptive field, i.e. allows to see more of the image with a fixed size filter
    - Example: if we constantly apply 3 x 3 filters, this is what they “see”



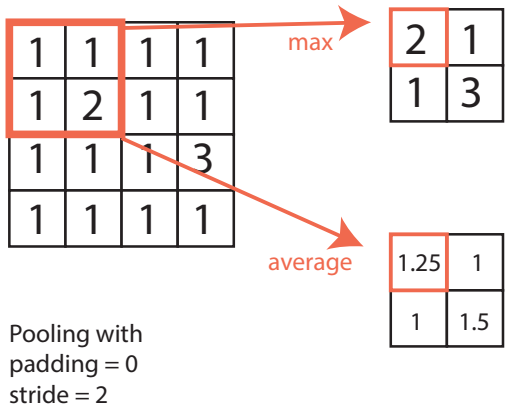
Ingredient #3 pooling

- Pooling has desirable effects:
  - It increases the receptive field, i.e. allows to see more of the image with a fixed size filter
  - Max pooling helps with translation invariance (the exact pixel position of the max is not so important, by shifting the image by 1 pixel the result is the same)

Ingredient #3 Pooling

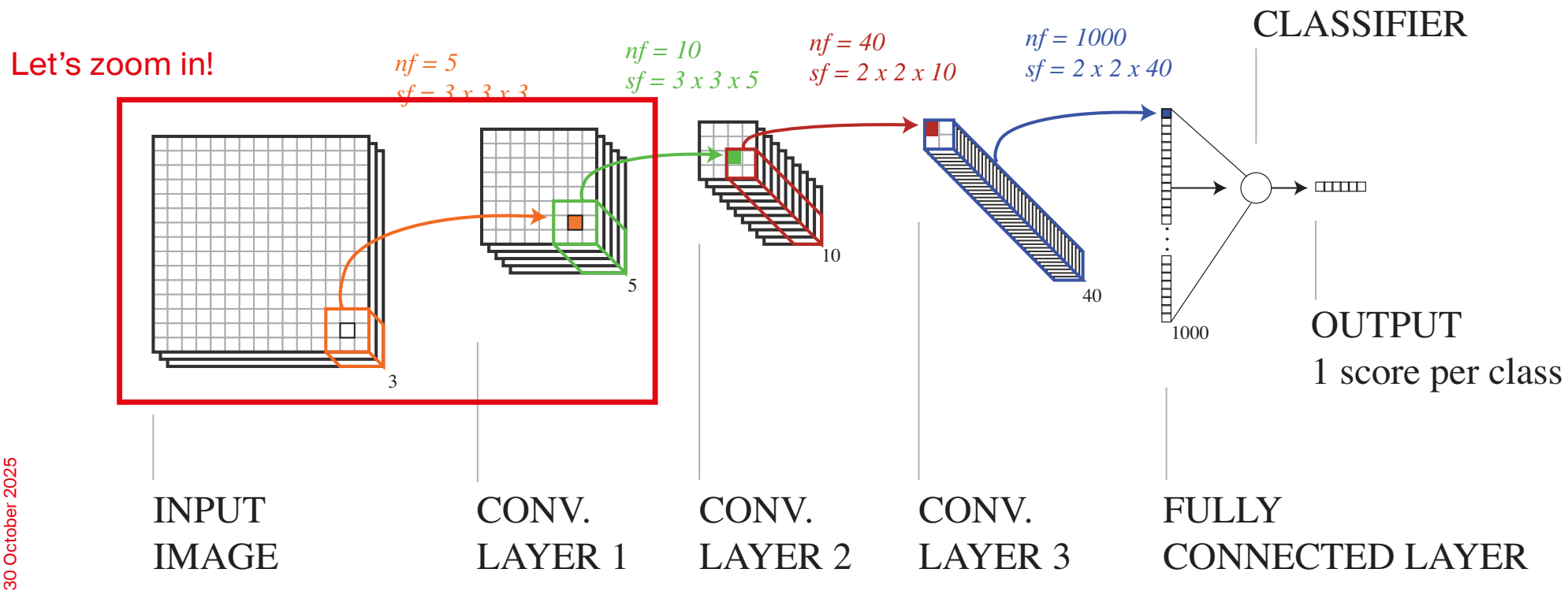
- Pooling has desirable effects:
  - It increases the receptive field, i.e. allows to see more of the image with a fixed size filter
  - Max helps with translation invariance (the exact pixel position of the max is not so important, by shifting the image by 1 pixel the result is the same)

- Stride is often set equal to the pooling size

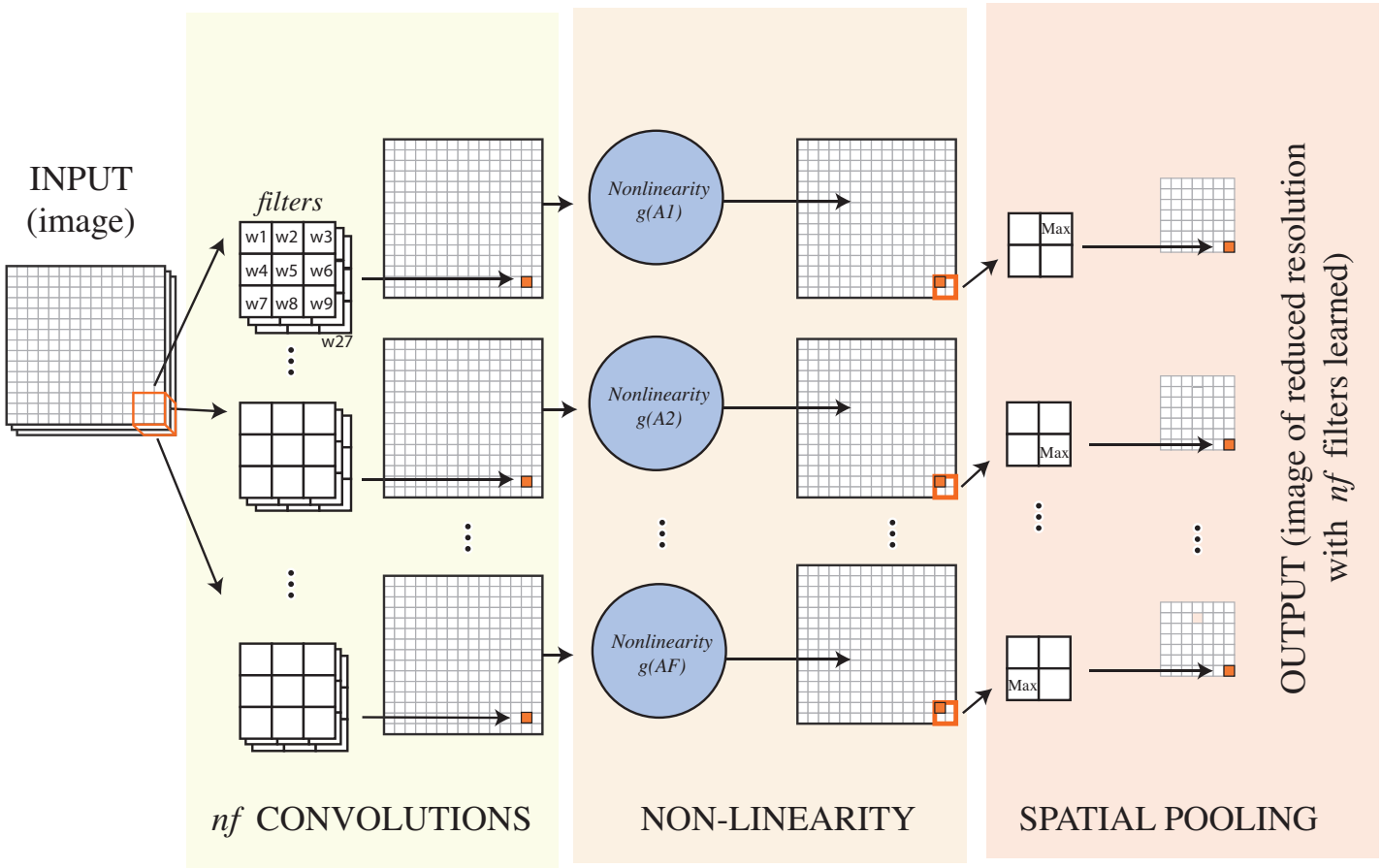


Pooling with
padding = 0
stride = 2

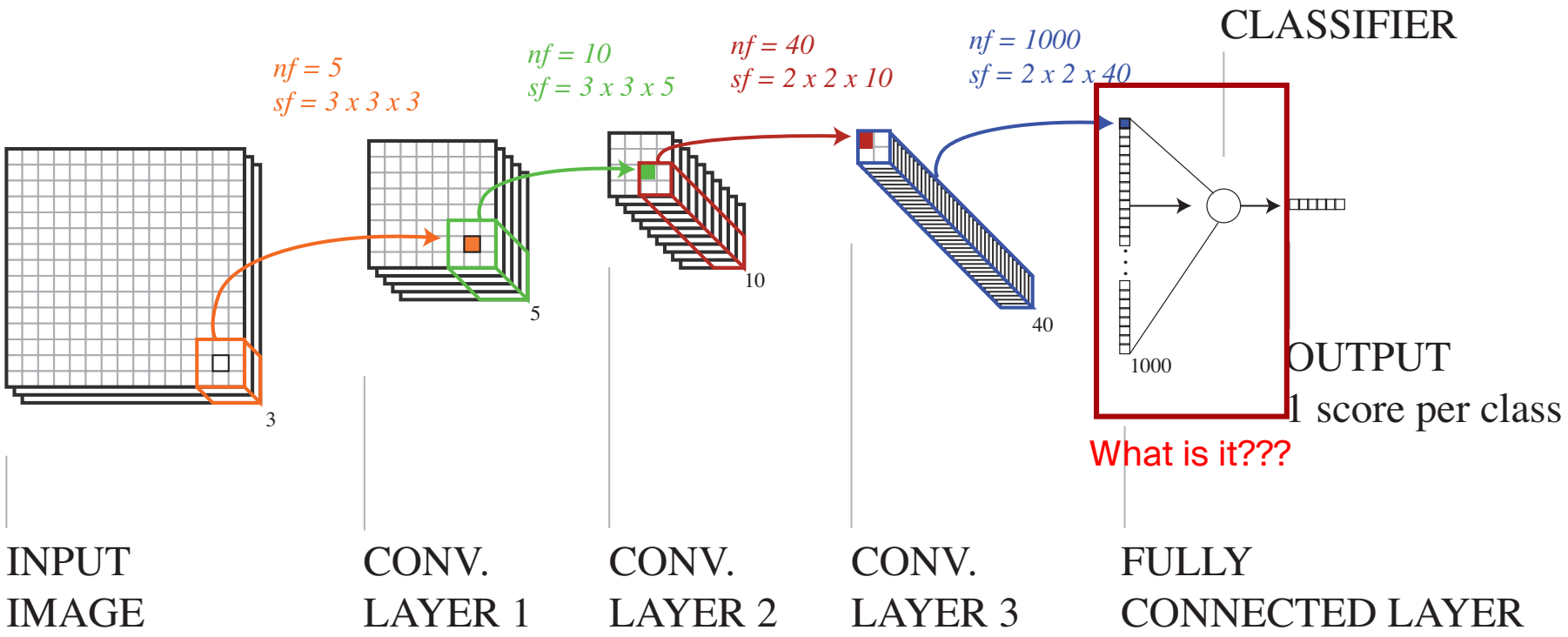
From the simplied convnet view...



... to the real one (for 1 layer, then repeat)

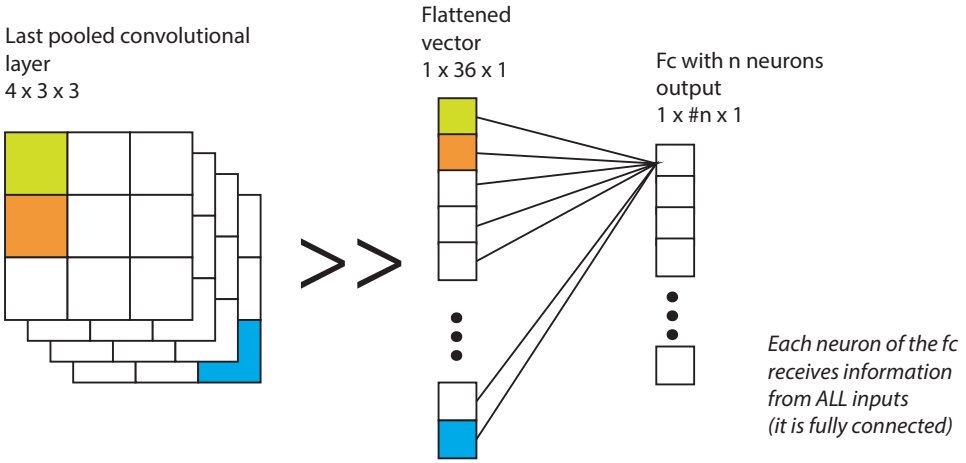


From the simplied convnet view...



The fully connected layer

- More or less what we saw in the MLP last week.
- First we flatten the tensor we have (either by learning a layer combining all activations (as in previous slide) or by simple reordering (as below)
- Then we use a fully connected layer (all inputs are mixed in each neuron of the output) and learn many neurons (here n)
- This final Fc layer is used for classification



EPFL

Convolutional neural networks

Larger inputs need weight sharing

From dense to convolutional layers

Padding, stride and pooling

LeNet as a base architecture

LeNet?

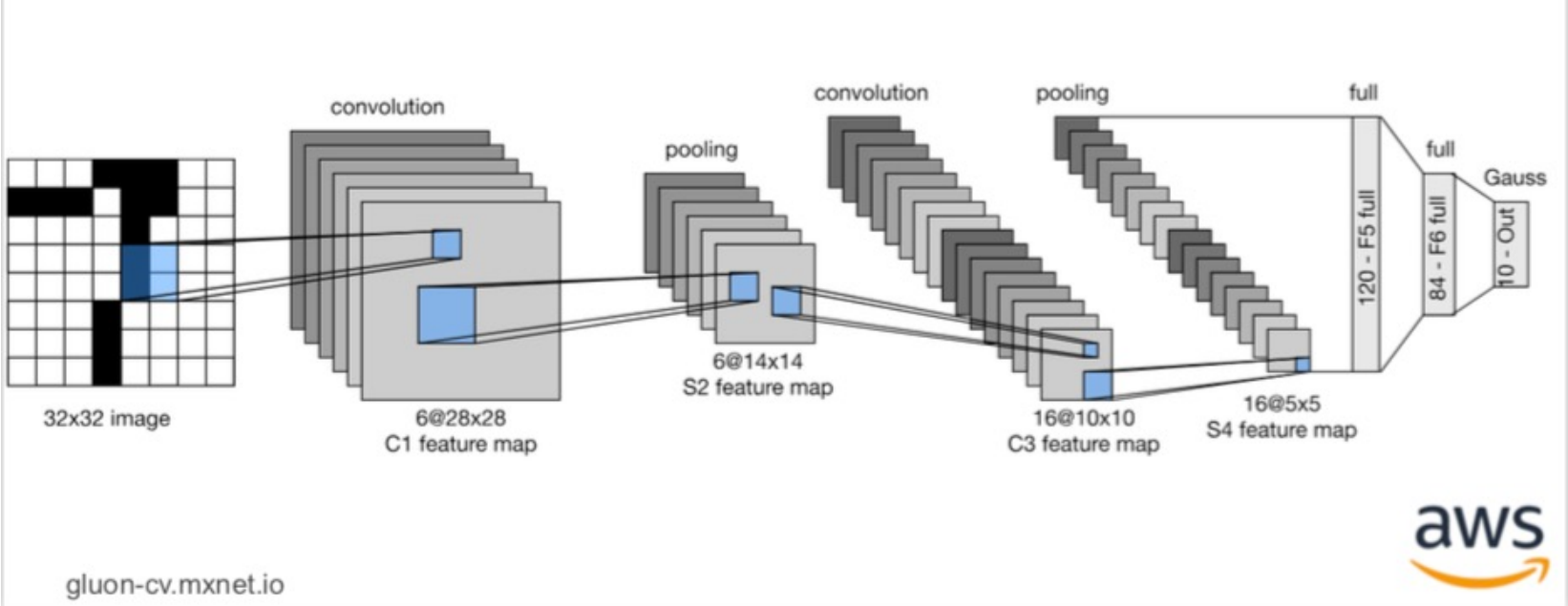
- The ‘90s .. A period many would like to forget
- But it has its good moments: in 1992, Y. LeCun and L. Bottou proposed LeNet, the first Convolutional Neural Network

- They used it to recognize digits for the U.S. post
- By lack of computational power and big datasets, the model didn’t beat state of art.
- But still, these concepts are at the core of basically all modern deep learning.

- **It is simple but powerful.**

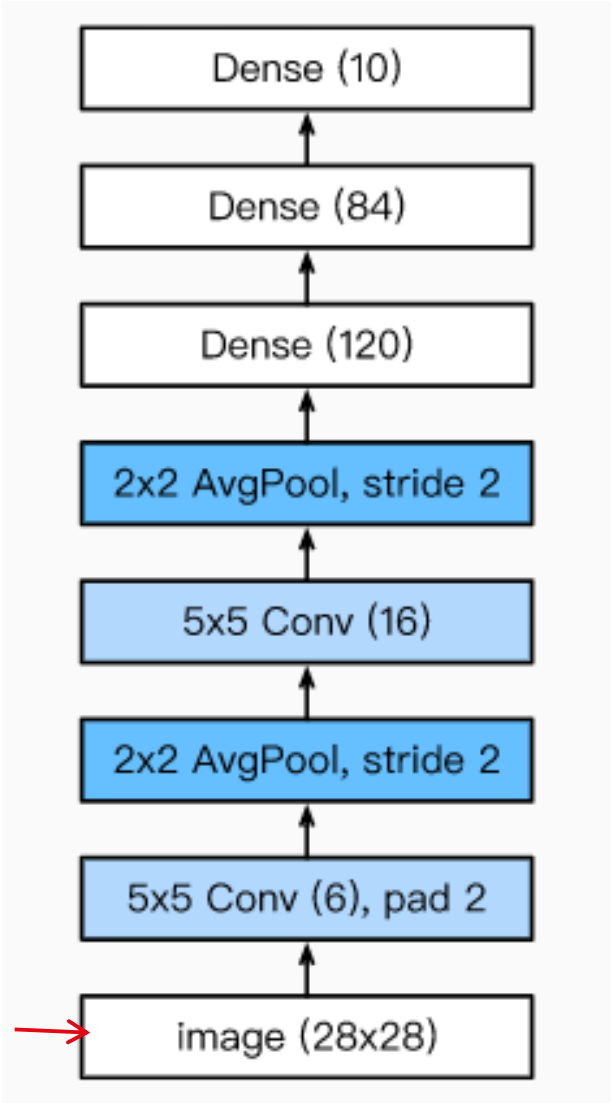
LeNet

- 2 convolutional layers, 5 x 5 filters >> neurons give image-like activation maps
- 2 average pooling layers >> reduce resolution of the maps
- Sigmoid nonlinearities >> no ReLUs back in the day
- 2 fully connected layers >> all the image gets summarized in a neuron, 120 times
- 10 output classes (10 digits) >> final output



LetNet

- 2 convolutional layers, 5 x 5 filters
- 2 average pooling layers
- Sigmoid nonlinearities
- 2 dense layers (same as “fully connected”)
- 10 output classes (10 digits)



You are now a convnet padawan

- CNN models design you can
- CNN parameters you understand
- Still much know you must (e.g. BatchNorm and Dropout)

- Before looking into semantic segmentation (next week), still some tricks you will learn...