23/10/2024

Convolutional Neural Networks

Giacomo Boracchi,
DEIB, Politecnico di Milano

giacomo.boracchi@polimi.it
https://boracchi.faculty.polimi.it/

The Feature Extraction Perspective

The Feature Extraction Perspective

Images can not be directly fed to a classifier

We need some intermediate step to:

- Extract meaningful information (to our understanding)
- Reduce data-dimension

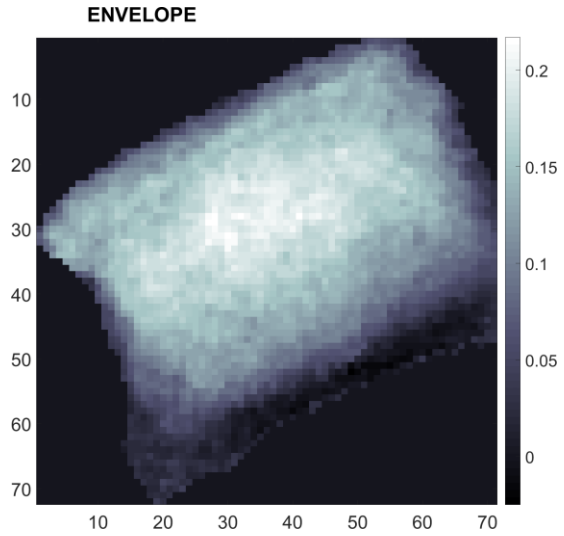
We need to extract features:

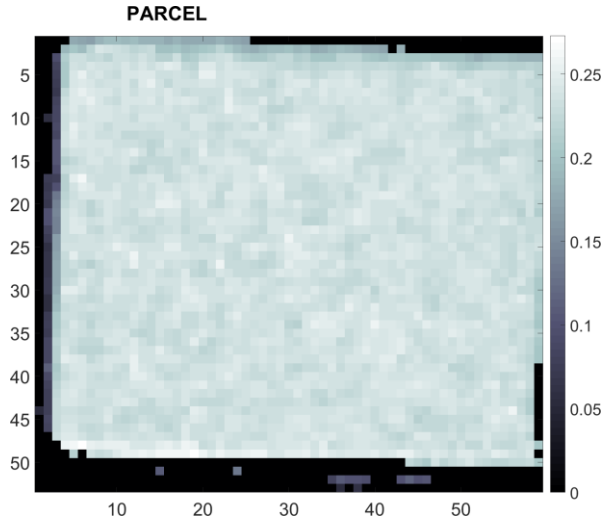
- The better our features, the better the classifier

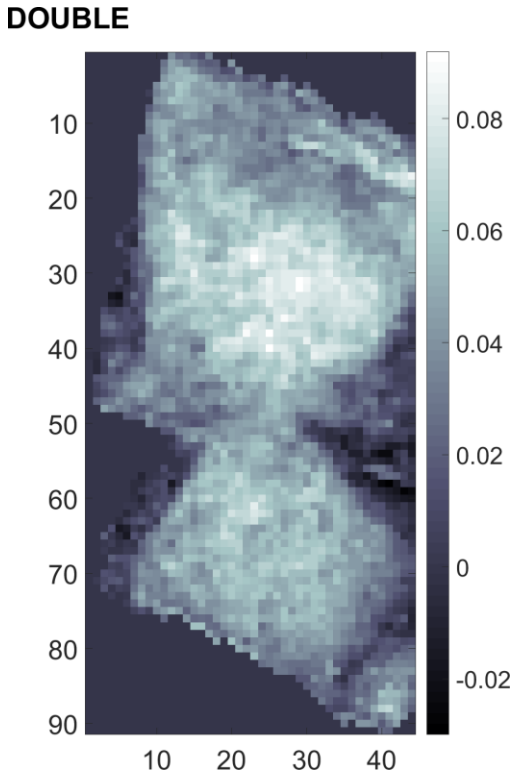
Example of Hand-Crafted Features

Example of features:

- Average height
- Area (coverage with nonzero measurements)
- Distribution of heights
- Perimeter
- Diagonals

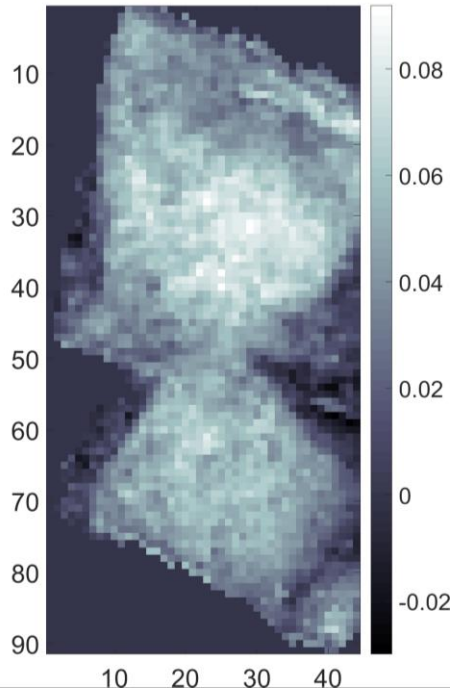






Neural Networks

Input image

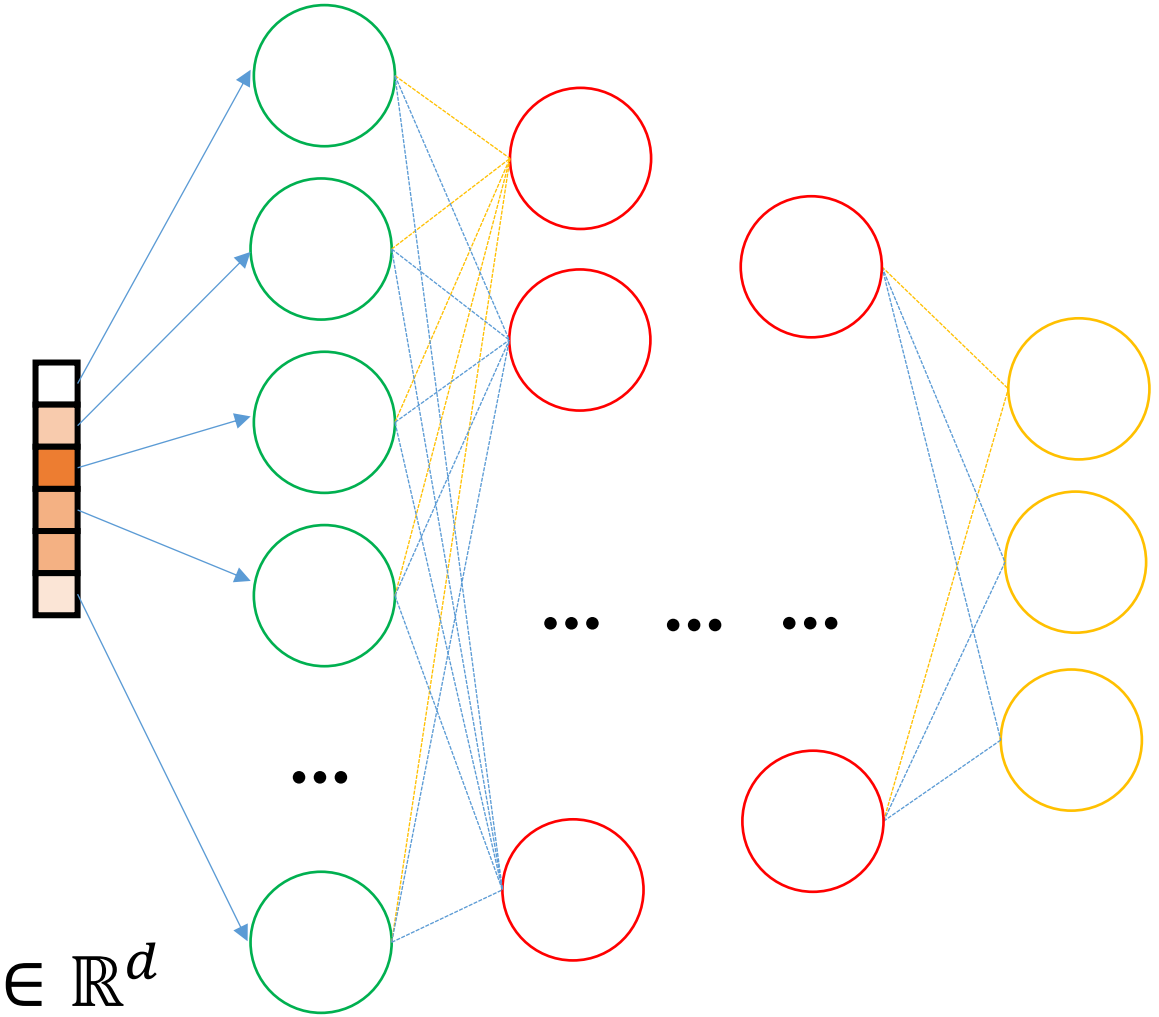


$I_1 \in \mathbb{R}^{r_1 \times c_1}$

Feature Extraction Algorithm

$(d \ll r_1 \times c_1)$

$\mathbf{x} \in \mathbb{R}^d$



input layer

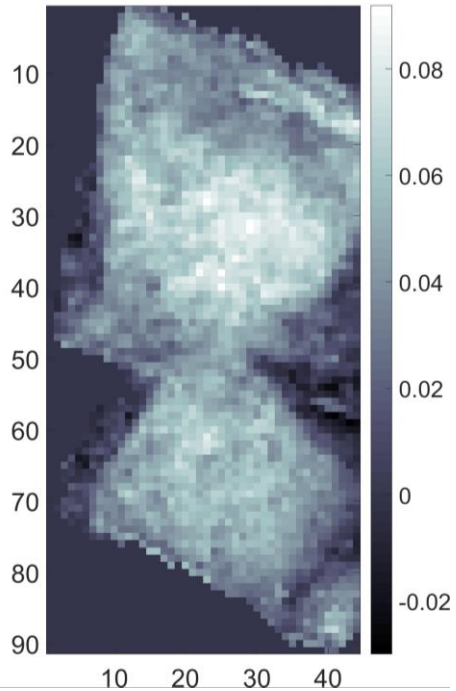
Hidden layer(s)

Output Layer

Giacomo Boracchi

Neural Network: After Training

Input image



$I_1 \in \mathbb{R}^{r_1 \times c_1}$

Feature Extraction Algorithm

$\mathbf{x} \in \mathbb{R}^d$

$P(l = \text{"doub."} | \boldsymbol{x})$

$P(l = \text{"env."} | \boldsymbol{x})$

$P(l = \text{"parc."} | \boldsymbol{x})$

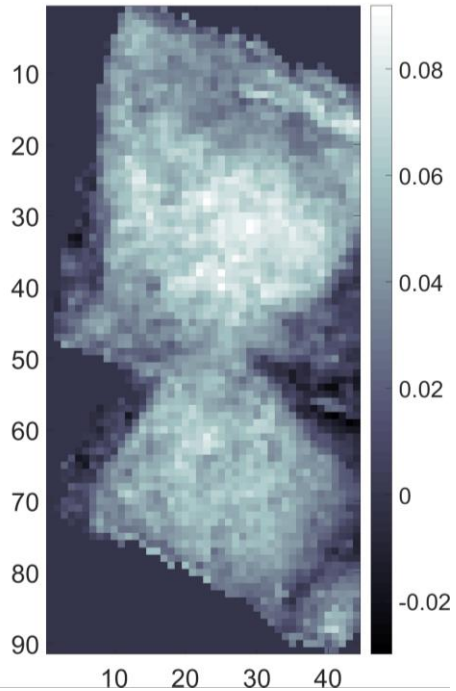
input layer

Hidden layer(s)

Output Layer

Image Classification by Hand Crafted Features

Input image



$I_1 \in \mathbb{R}^{r_1 \times c_1}$

Feature Extraction Algorithm

Hand Crafted

Data Driven

Giacomo Boracchi

Hand Crafted Featues, pros:

- **Exploit a priori** / expert **information**
- Features are **interpretable** (you might understand why they are not working)
- You can **adjust features** to improve your performance
- **Limited amount of training data** needed
- You can give more relevance to some features

Hand Crafted Featues, cons:

- Requires a lot of **design/programming efforts**
- **Not viable** in many **visual recognition** tasks (e.g. on natural images) which are easily performed by humans
- **Risk of overfitting** the training set used in the design
- **Not very general and "portable"**

Data-Driven Features

... the advent of deep learning

Data-Driven Features

Input image

$I_1 \in \mathbb{R}^{r_1 \times c_1}$

Feature Extraction

Data Driven

Data Driven

Convolutional Neural Networks

Setting up the stage

Giacomo Boracchi

Local Linear Filters RECAP

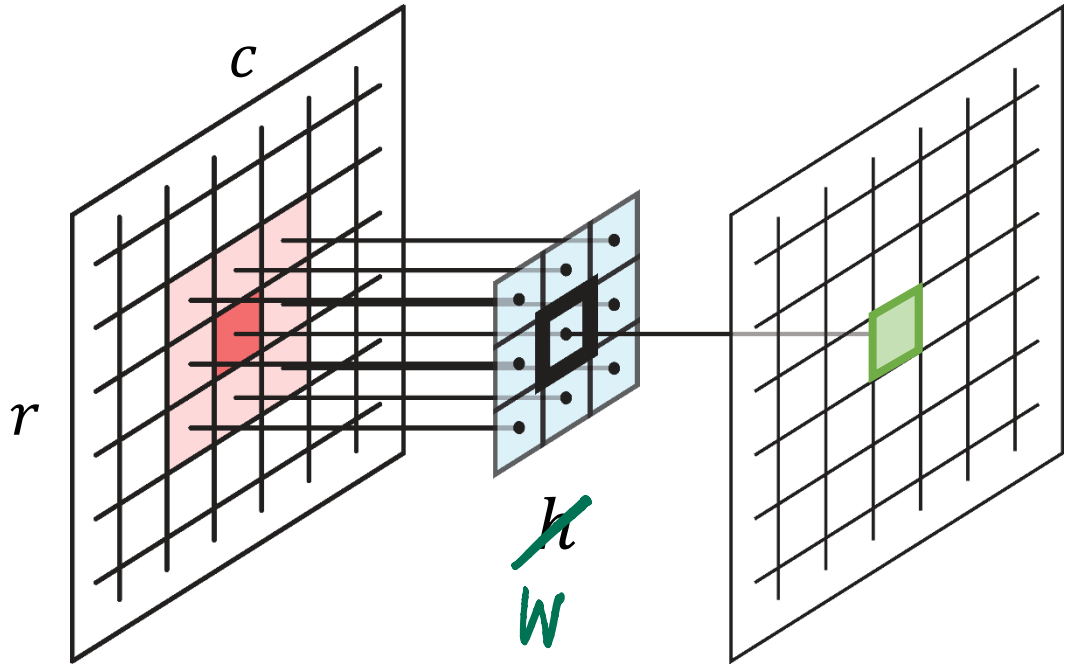
Linear Transformation: Linearity implies that the output $T[I](r,c)$ is a linear combination of the pixels in U:

$$T[I](r,c) = \sum_{(u,v)\in U} w_i(u,v) * I(r+u, c+v)$$

Considering *some weights* $\{w_i\}$

We can consider weights as an image, or **a filter** h

The filter h **entirely defines this operation**



Local Linear Filters RECAP

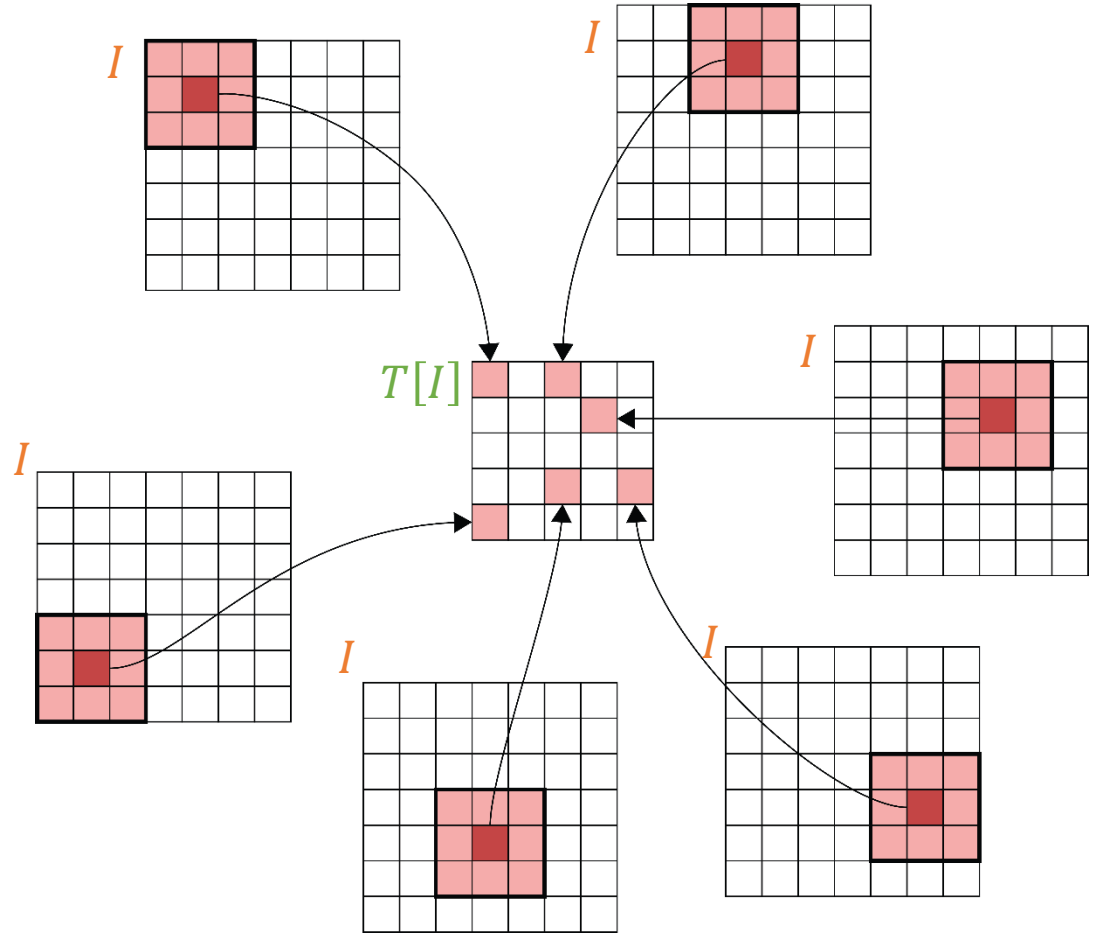
Linear Transformation: the filter weights can be assoicated to a matrix $\boldsymbol{w}$

$$T[I](r,c) = \sum_{(u,v)\in U} w_i(u,v) * I(r+u, c+v)$$

$\boldsymbol{w}$

$w(-1,-1)$	$w(-1,0)$	$w(-1,1)$
$w(0,-1)$	$w(0,0)$	$w(0,1)$
$w(1,-11)$	$w(1,0)$	$w(1,1)$

This operation is repeated for each pixel in the input image



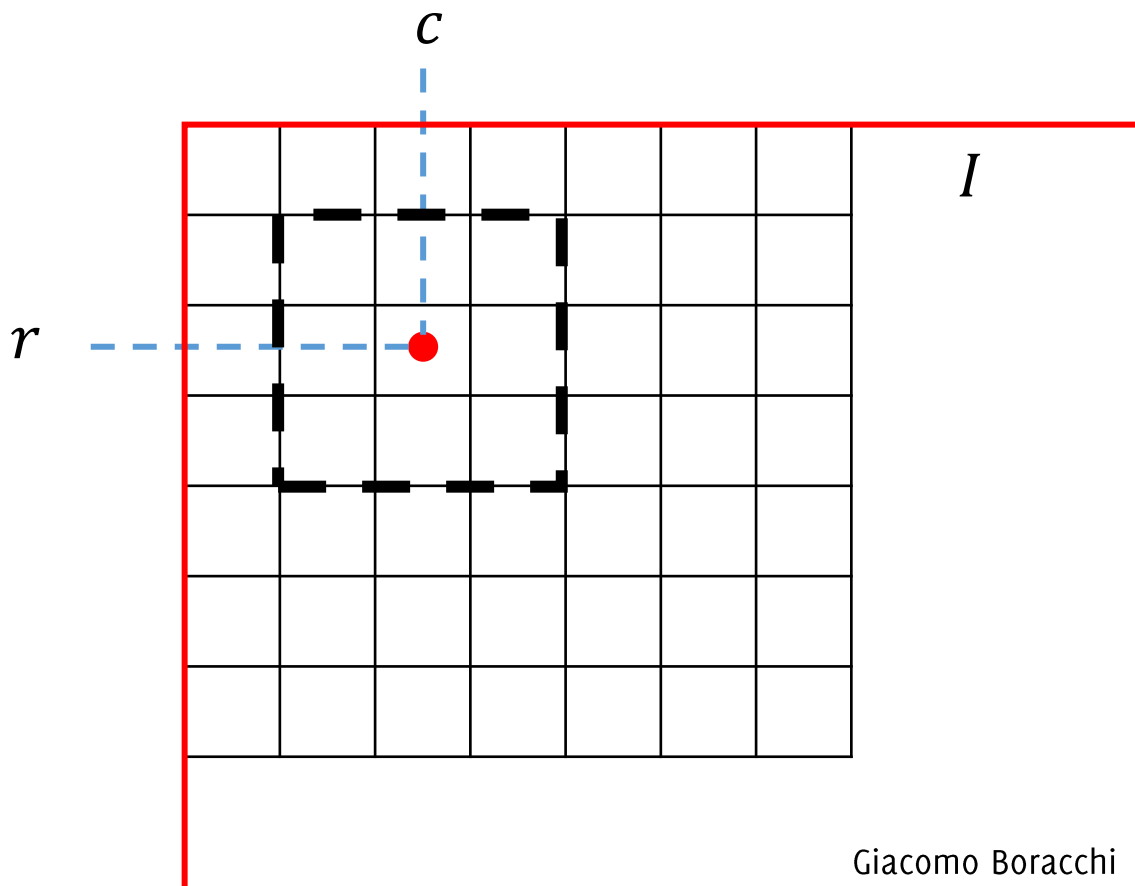
2D Correlation

Convolution is a linear transformation. Linearity implies that

$$(I \otimes w)(r,c) = \sum_{(u,v)\in U} w(u,v)\, I(r+u, c+v)$$

w

$w(-1,-1)$	$w(-1,0)$	$w(-1,1)$
$w(0,-1)$	$w(0,0)$	$w(0,1)$
$w(1,-1)$	$w(1,0)$	$w(1,1)$



We can consider weights as **a filter** h

The filter h **entirely defines convolution**

Convolution operates the same in each pixel

2D Convolution

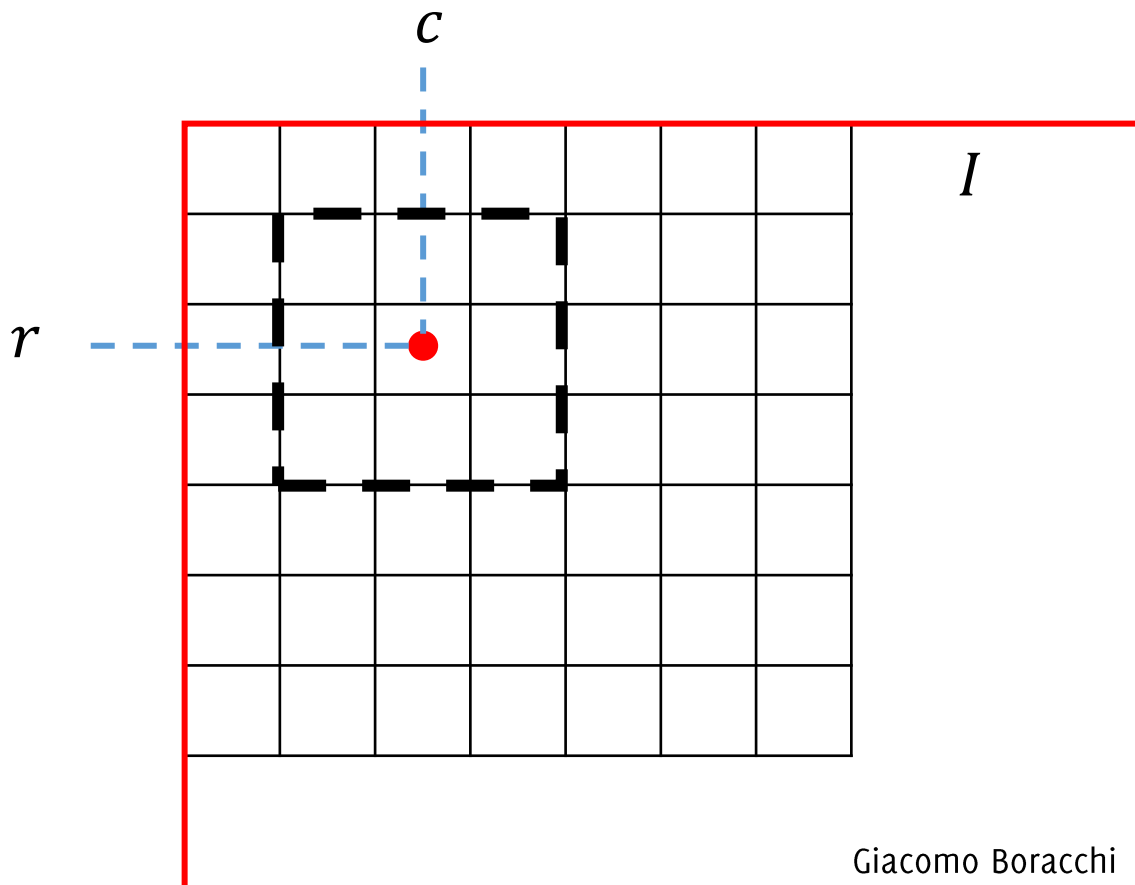
Convolution is a linear transformation. Linearity implies that

$$(I \circledast w)(r,c) = \sum_{(u,v)\in U} w(u,v)\, I(r-u, c-v)$$

w

$w(1,1)$	$w(1,0)$	$w(1,-1)$
$w(0,1)$	$w(0,0)$	$w(0,-1)$
$w(-1,1)$	$w(-1,0)$	$w(-1,-1)$

Rmk: indexes have been shifted in the filter w



We can consider weights as **a filter** h

The filter h entirely defines convolution

Convolution operates the same in each pixel

2D Convolution

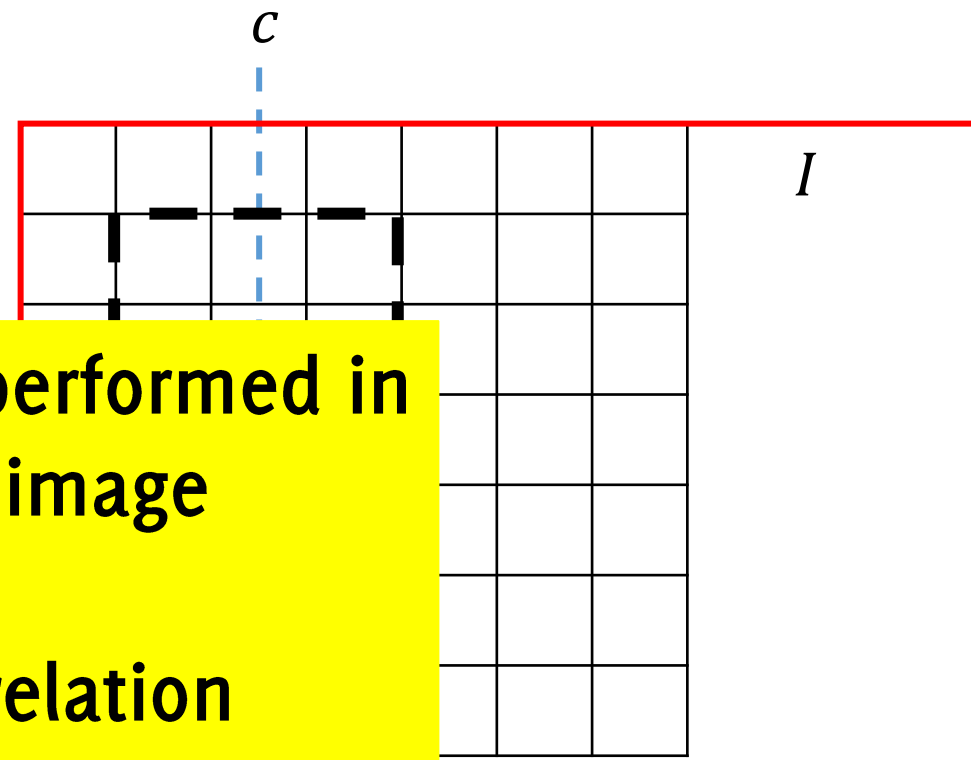
Convolution is a linear transformation. Linearity implies that

$$(I \circledast w)(r,c) = \sum_{(u,v)\in U} w(u,v)\, I(r-u, c-v)$$

w

$w(1,1)$	$w(1,0)$	$w(1,-1)$
$w(0,1)$	$w(0,0)$	$w(0,-1)$
$w(-1,1)$	$w(-1,0)$	

Rmk: indexes have been shifted in the filter w



We can con

The filter h e

Convolution op

The same operation is being performed in each pixel of the input image

It is equivalent to 2D Correlation up to a «flip» in the filter w

2D Convolution

Convolution is a linear transformation. Linearity implies that

$$(I \circledast w)(r,c) = \sum_{(u,v)\in U} w(u,v) * I(r-u, c-v)$$

Convolution is defined up to the "filter flip" for the Fourier Theorem to apply. Filter flip must be considered when computing convolution in Fourier domain and when designing filters.

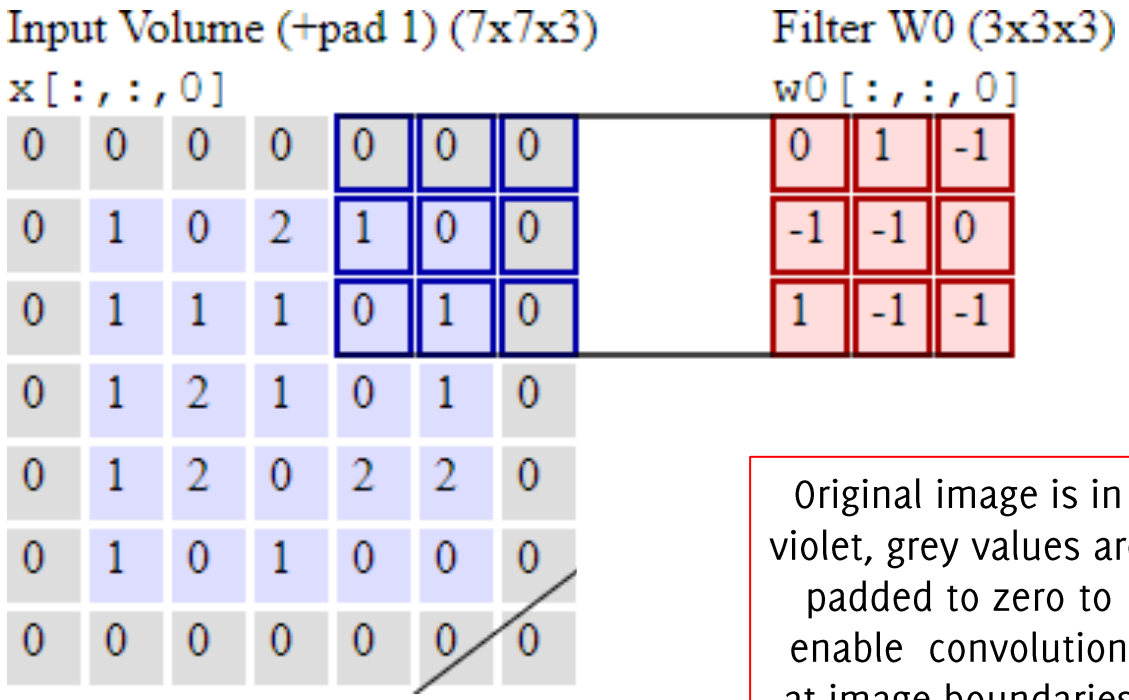
However, in CNN, convolutional filters are being learned from data, thus it is only important to use these in a consistent way.

In practice, **in CNN arithmetic there is no flip!**

Convolution: Padding

How to define convolution output close to image boundaries?

Padding with zero is the most frequent option, as this does not change the output size. However, no padding or symmetric padding are also viable options

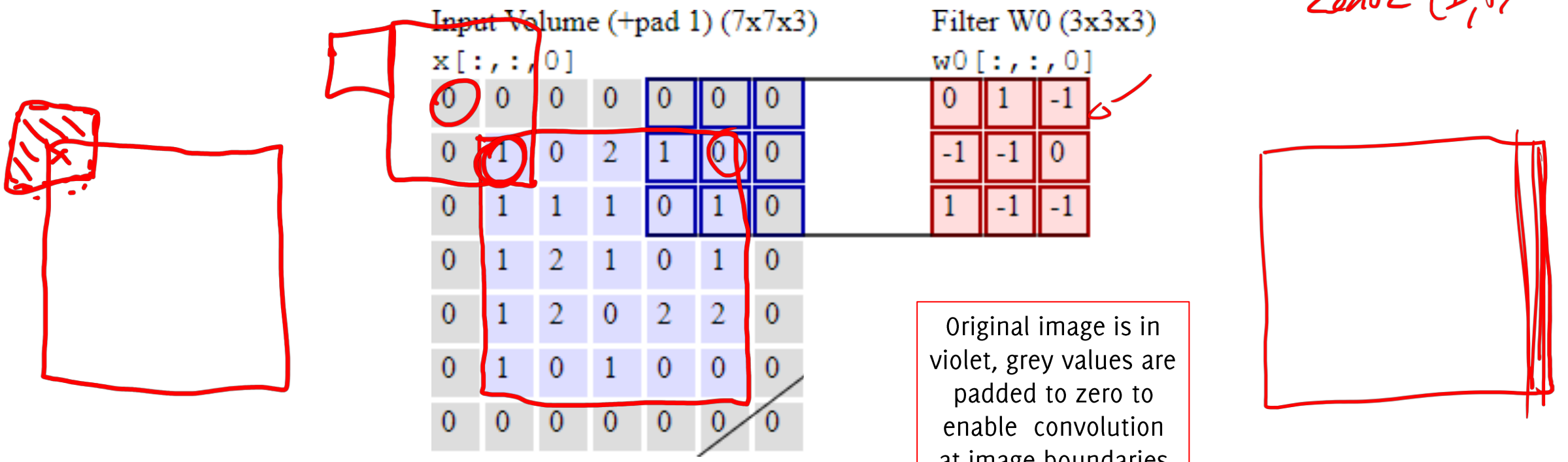


Original image is in violet, grey values are padded to zero to enable convolution at image boundaries

Convolution: Padding

How to define convolution output close to image boundaries?

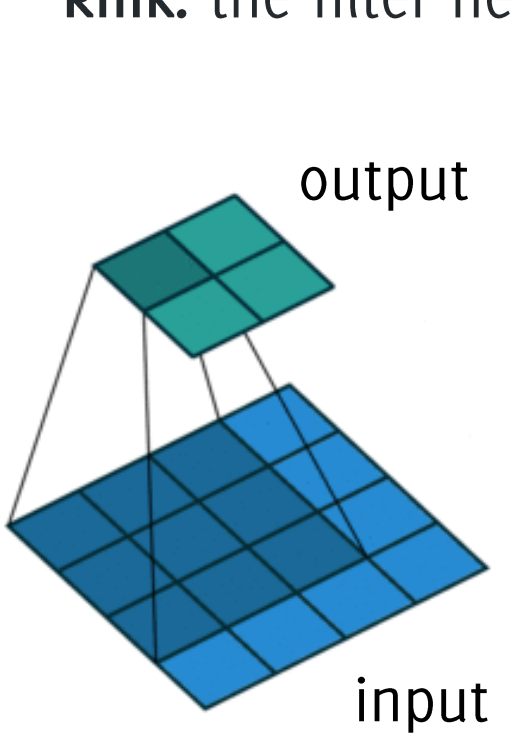
Padding with zero is the most frequent option, as this does not change the output size. However, no padding or symmetric padding are also viable options



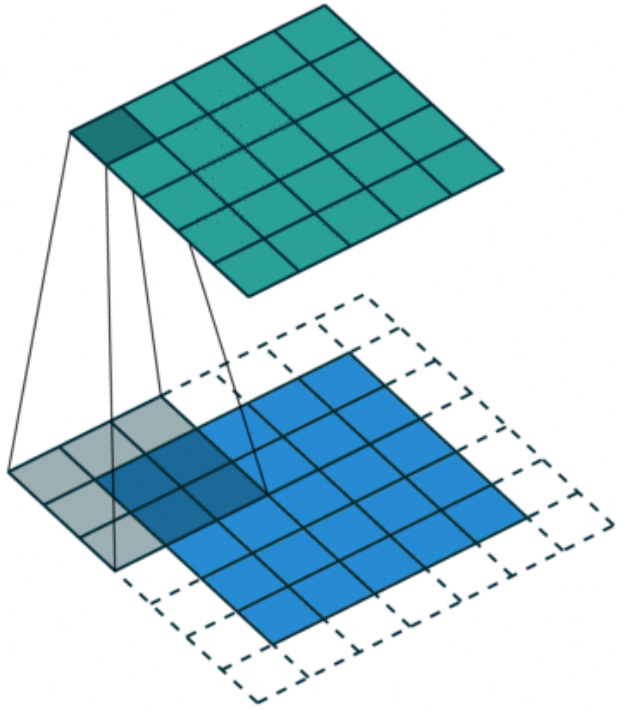
Original image is in violet, grey values are padded to zero to enable convolution at image boundaries

Padding Options in Convolution Animation

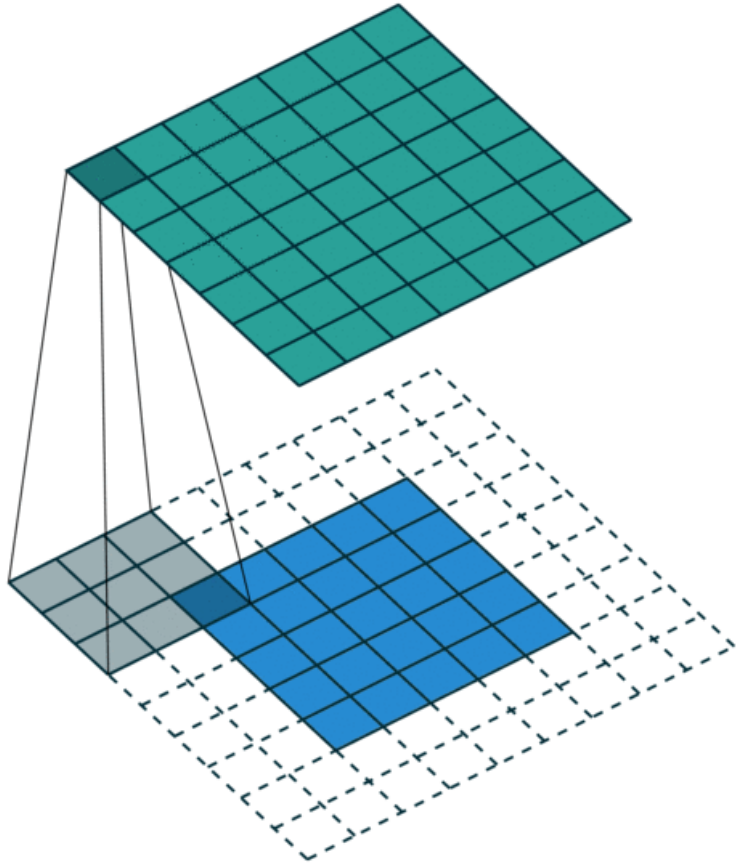
Rmk: Blue maps are inputs, and cyan maps the outputs.
Rmk: the filter here is 3×3



No padding
«valid»



Half padding
«same»



full padding
«full»

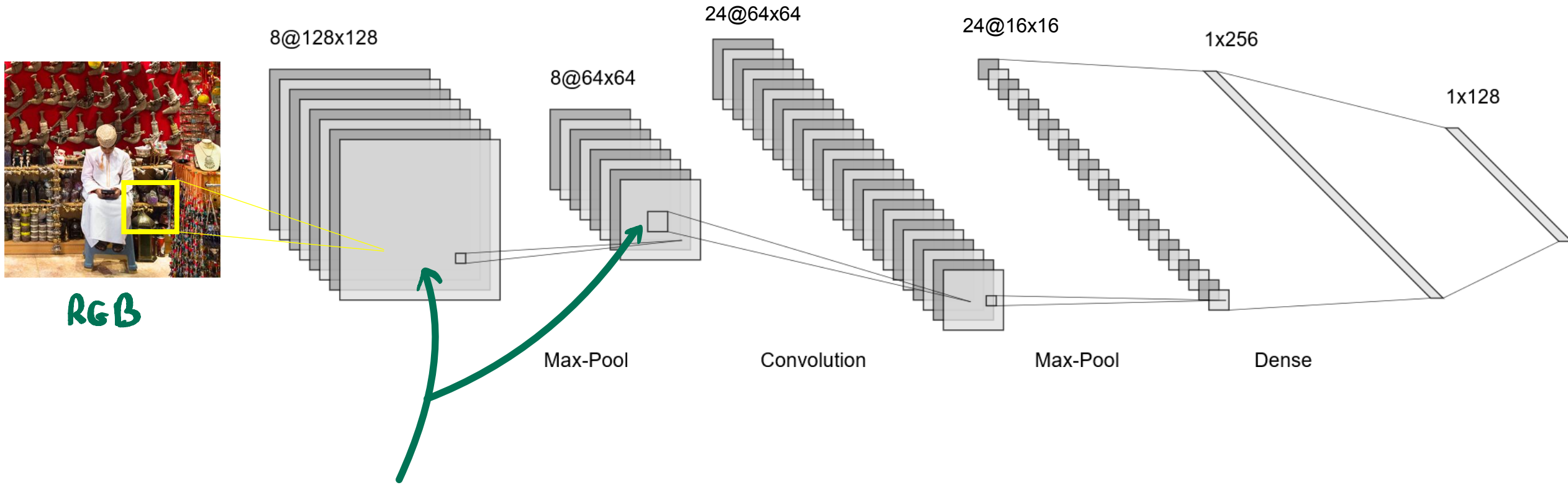
Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

Convolutional Neural Networks

CNNs

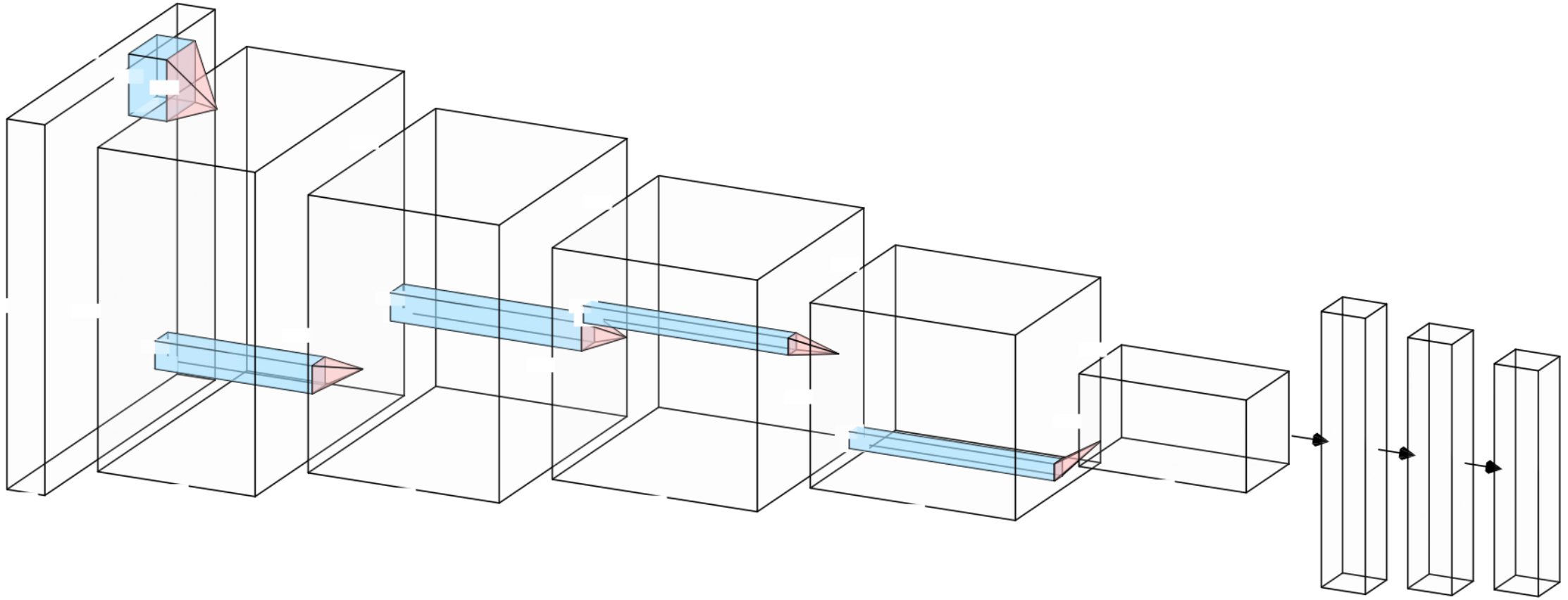
Giacomo Boracchi

The typical architecture of a CNN

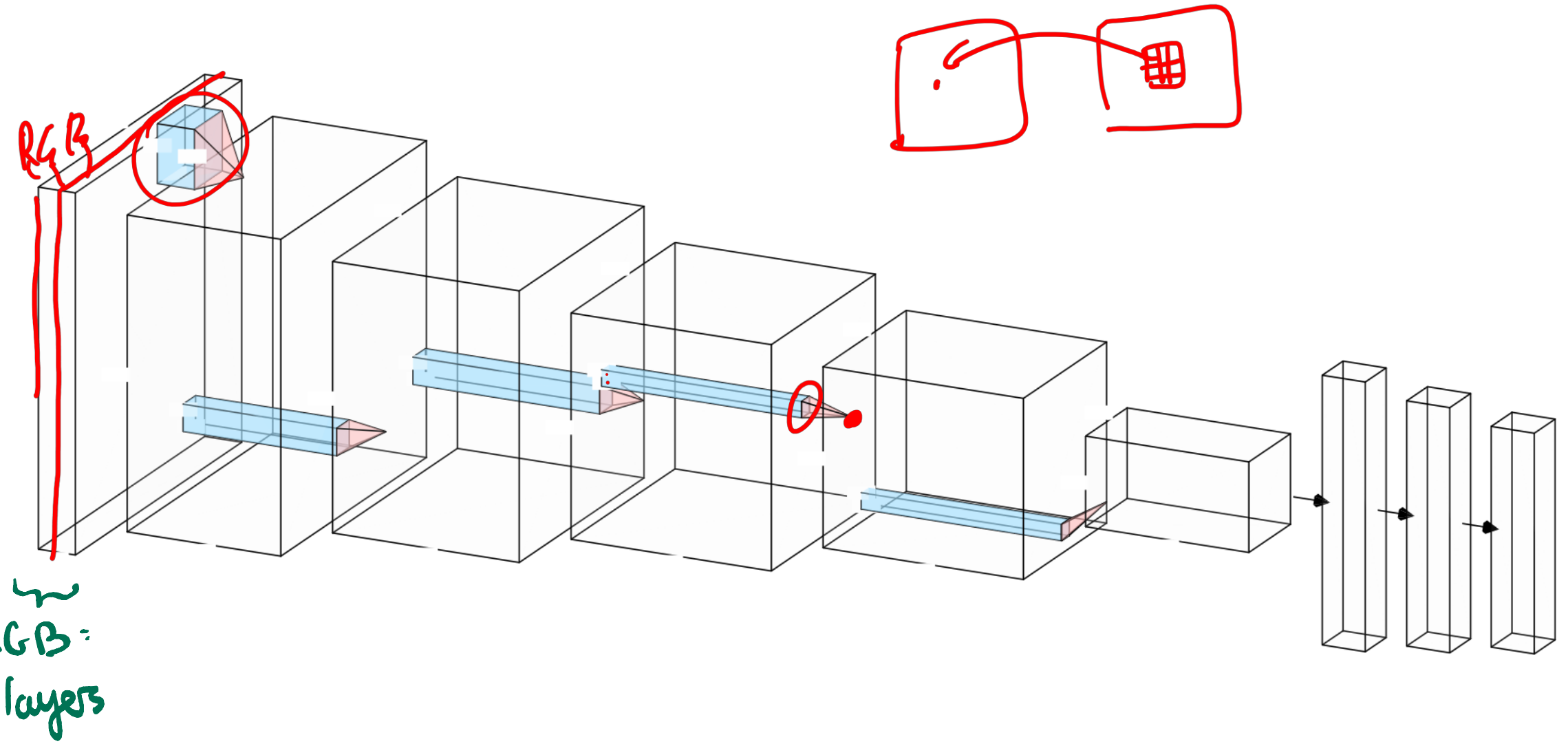


the "spatial size" (2D) ↘
but the nb of layers ↗ (3rd dim).

The typical architecture of a CNN



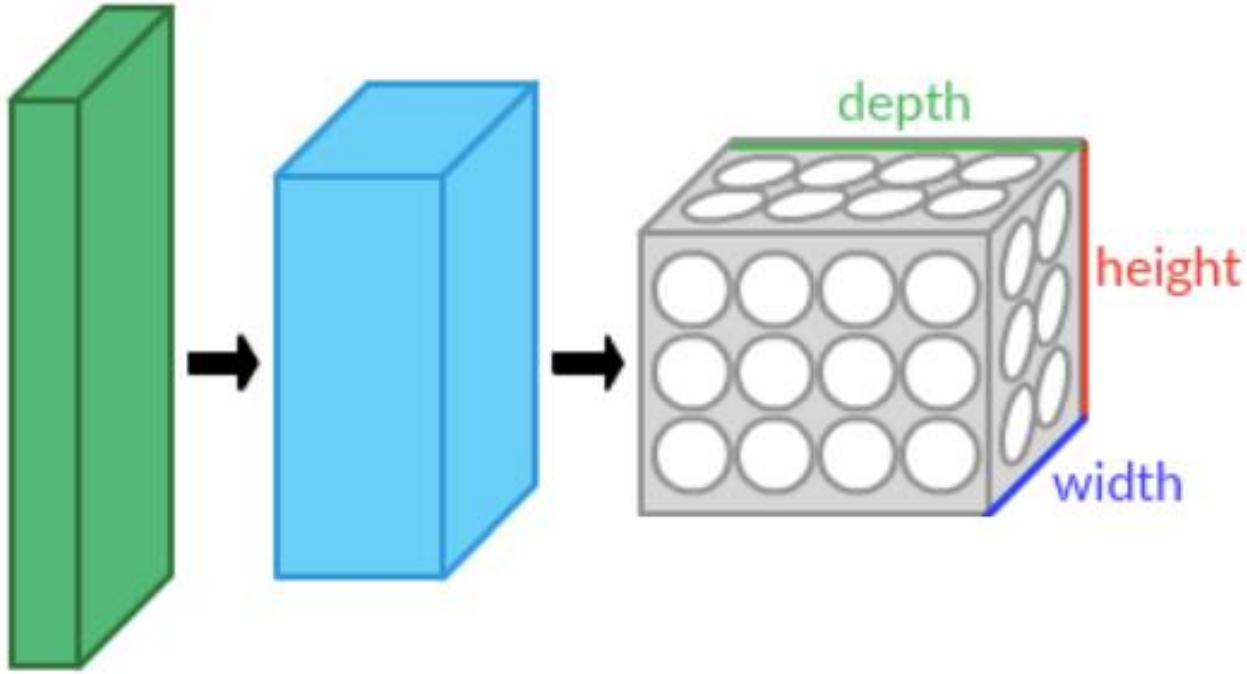
The typical architecture of a CNN



Convolutional Neural Networks (CNN)

CNN are typically made of blocks that include:

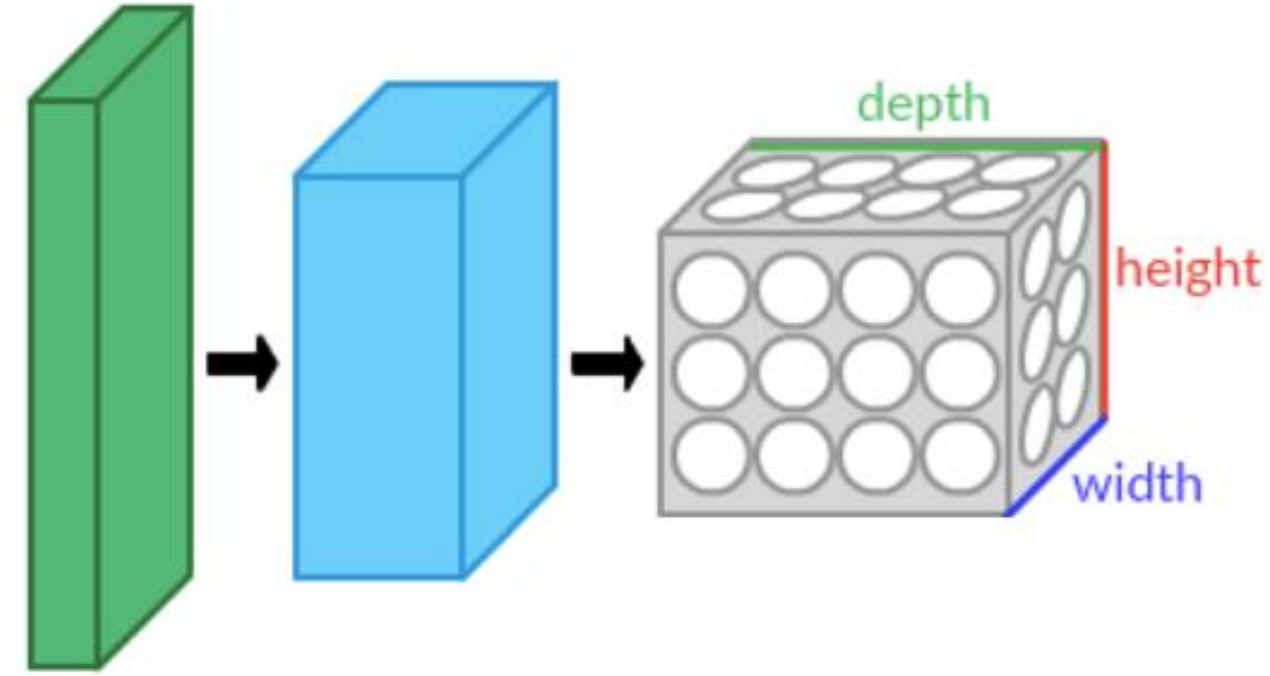
- Convolutional layers
- Nonlinearities (activation functions)
- Pooling Layers (Subsampling / maxpooling)



By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45661858

Convolutional Neural Networks (CNN)

- An image passing through a CNN is transformed in a sequence of volumes.
- As the depth increases, the height and width of the volume decreases
- Each layer takes as input and returns a volume



By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45661858

Convolutional Layers

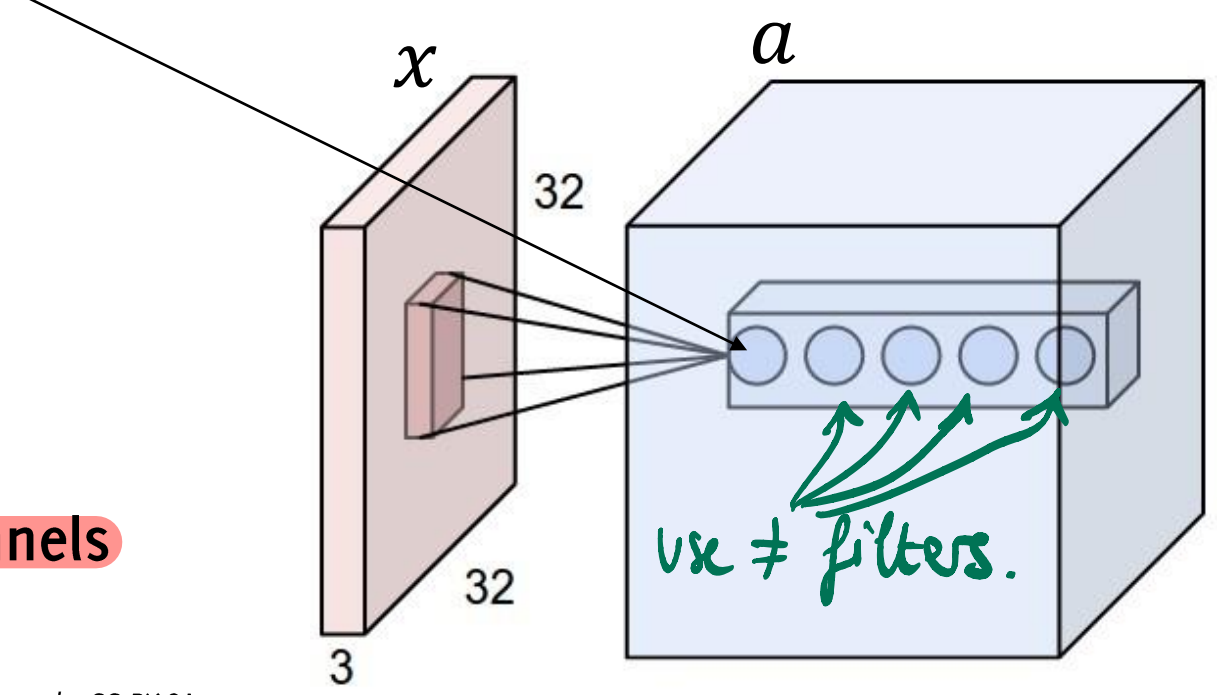
Giacomo Boracchi

Convolutional Layers

Convolutional layers "mix" all the input components

The output is a linear combination of **all the values in a region of the input, considering all the channels**

$$a(r, c, 1) = \sum_{(u,v) \in U, k} w^1(u, v, k)\, x(r + u, c + v, k) + b^1$$



Filters need to have **the same number of channels** as the input, to process all the values from the input layer

By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

Convolutional Layers

Convolutional layers "mix" all the input components

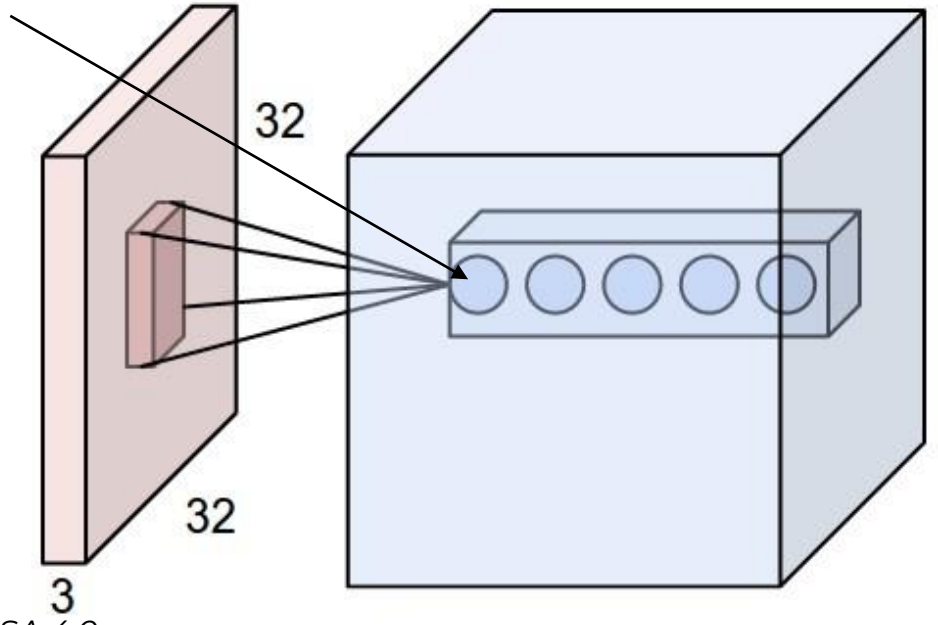
The output is a linear combination of **all the values in a region of the input, considering all the channels**

learned parameters.

$$a(r,c,1) = \sum_{(u,v)\in U,k} w^1(u,v,k)\,x(r+u,c+v,k) + b^1$$

The parameters of this layer are **called filters**.

The same filter is used through the whole spatial extent of the input



By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

Convolutional Layers

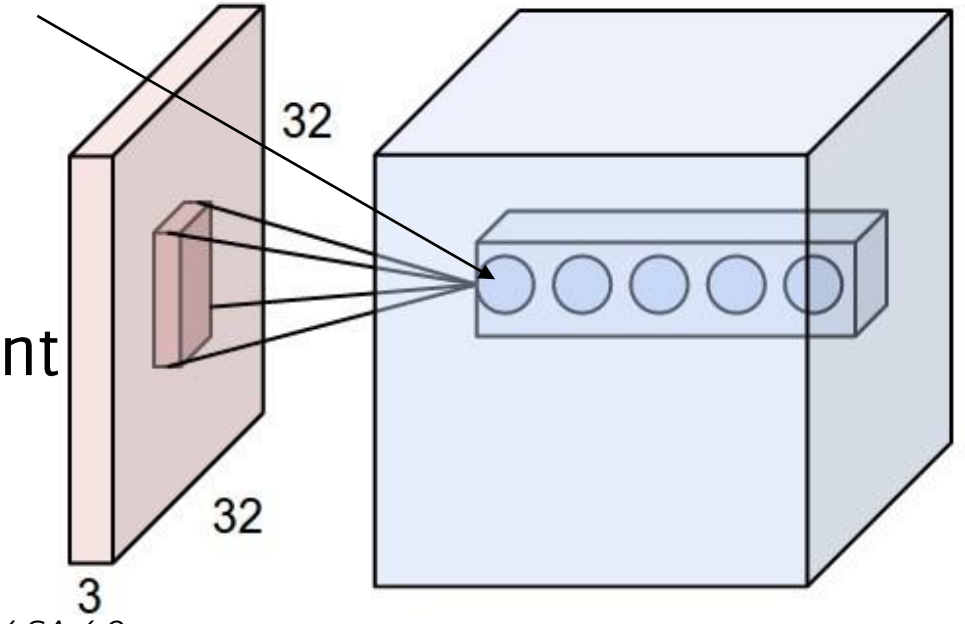
Convolutional layers "mix" all the input components

The output is a linear combination of **all the values in a region of the input, considering all the channels**

$$a(r, c, 1) = \sum_{(u,v) \in U, k} w^1(u, v, k)\, x(r + u, c + v, k) + b^1$$

The **spatial dimension**:

- spans a small neigborhood U (local processing, it's a convolution)
- U needs to be specified, it is a very important attribute of the filter



By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

Convolutional Layers

Convolutional layers "mix" all the input components

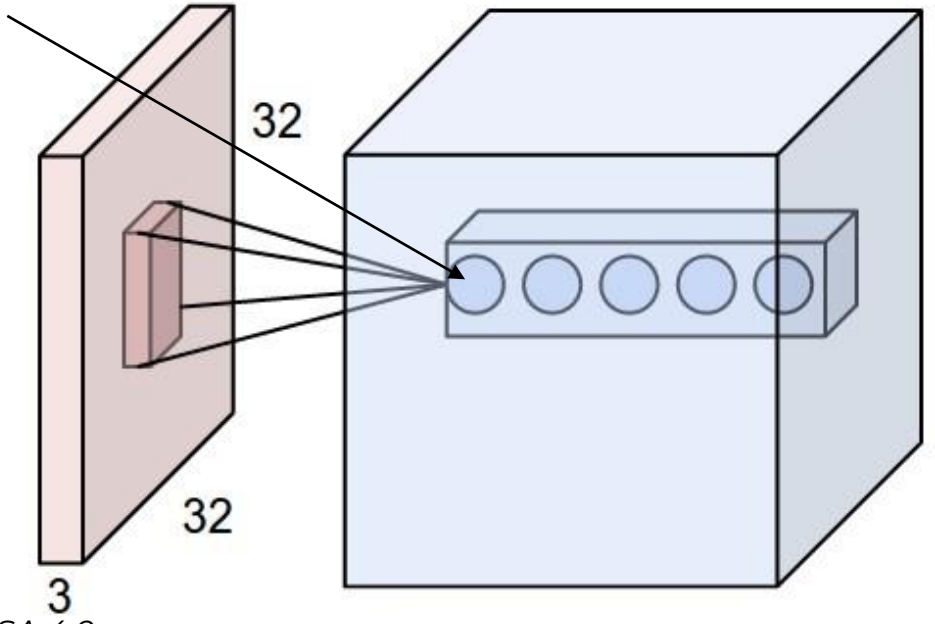
The output is a linear combination of **all the values in a region of the input, considering all the channels**

$$a(r, c, 1) = \sum_{(u,v) \in U, k} w^1(u, v, k)\, x(r + u, c + v, k) + b^1$$

The channel dimension:

- spans the entire input depth (no local processing, like spatial dimension)

- there is no need to specify that in the filter attributes

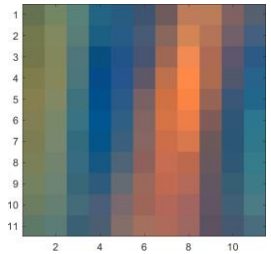


By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

Convolutional Layers

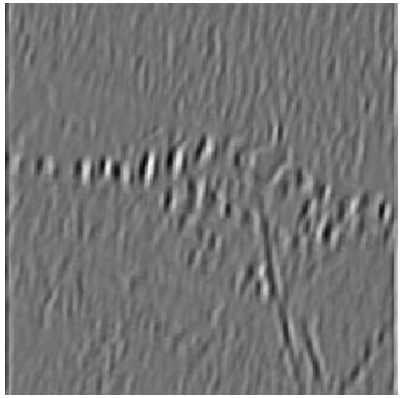
I

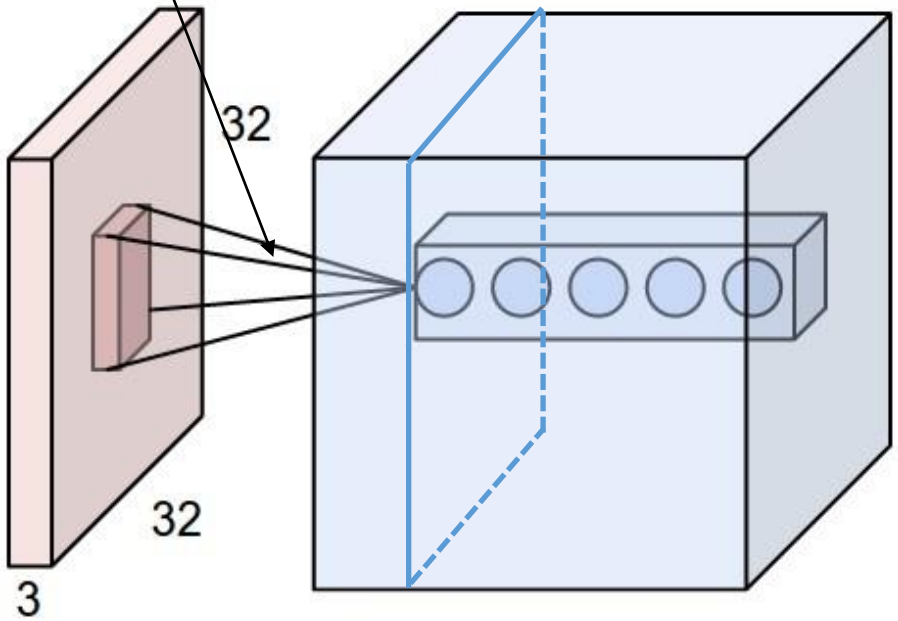
w^1



b^1

$a(:,:,1)$





By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

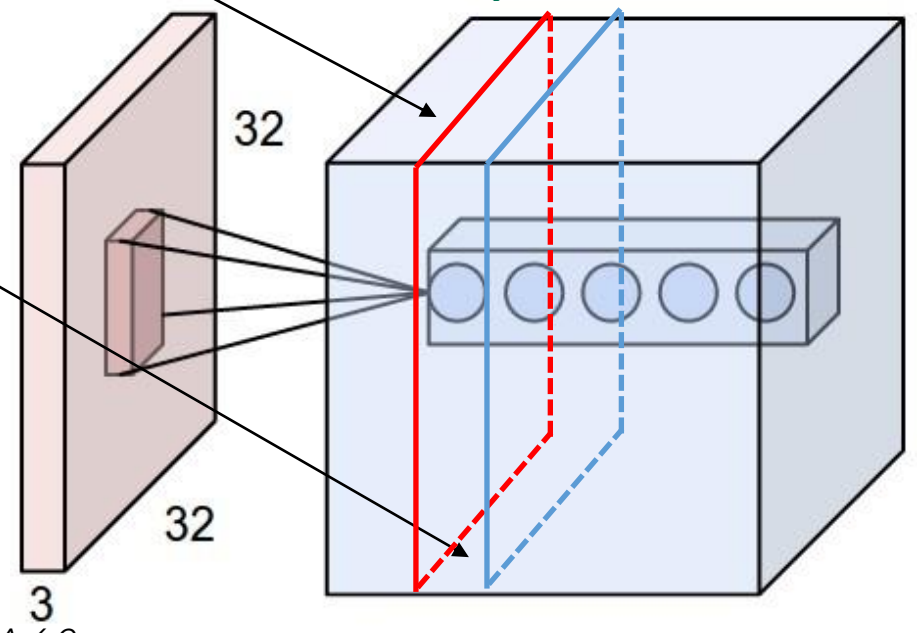
Convolutional Layers

Different filters yield different layers in the output

use ≠ filters to increase the depth!

$$a(r,c,1) = \sum_{(u,v)\in U,k} w^1(u,v,k)\, \boldsymbol{x}(r+u, c+v, k) + b^1$$

$$a(r,c,2) = \sum_{(u,v)\in U,k} w^2(u,v,k)\, \boldsymbol{x}(r+u, c+v, k) + b^2$$



Different filters of the same layer have the same spatial extent

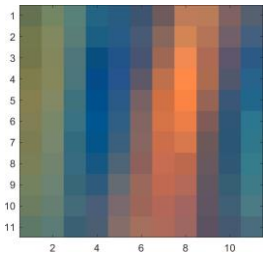
Convolutional Layers

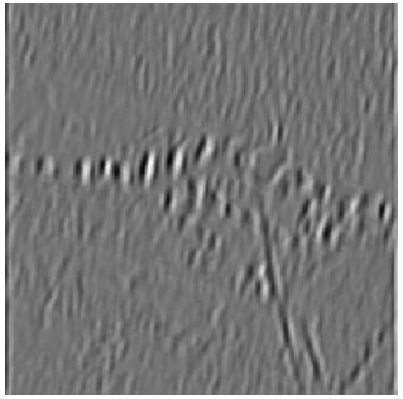
$a(:,:,1)$

$a(:,:,n)$

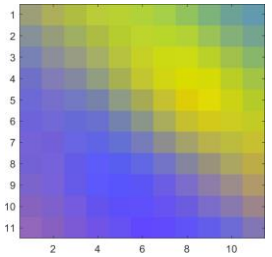
I

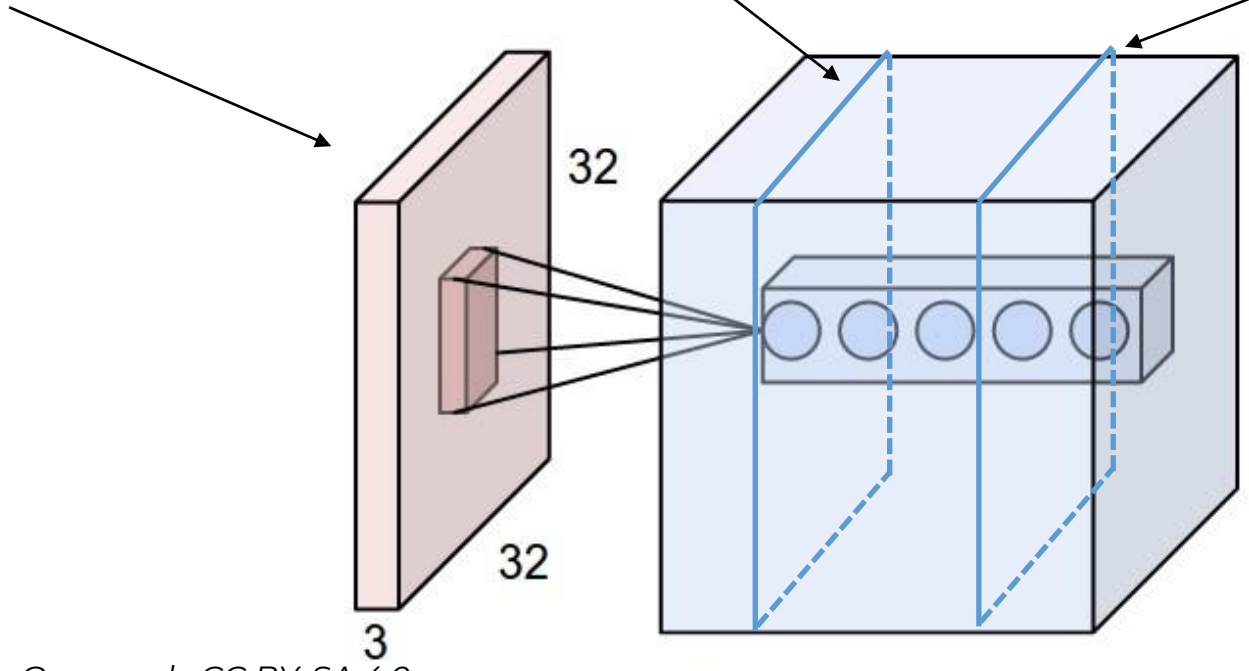
w^1, b^1





w^n, b^n





By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

Convolutional Layers, general formula:

$$a(r,c,l) = \sum_{\substack{(u,v)\in U,\\ k=1,..,C}} w^l(u,v,k)x(r+u,c+v,k) + b^l, \forall\,(r,c), l = 1,\dots,N_F$$

Where r, c denote a spatial location in the output activation a, while l is the channel.

The filter w is identified by

- a spatial neighborhood U having size $h_r \cdot h_c$
- the same number of channels C as the input activations

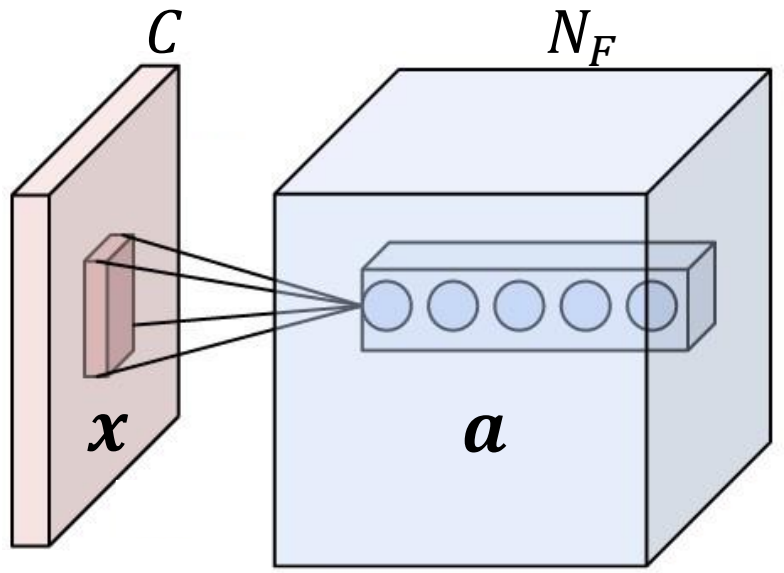
The parameters are the weights + one bias per filter

The overall number of parameters is

$$(h_r \cdot h_c \cdot C) \cdot N_F + N_F$$

bias.

$\forall l$: $h_r \times h_c \times C + 1$



Layers with the same hyperparameters can have different number of parameters depending on where these are located in the network

Convolutional Layers, remarks:

Given:

```
conv2 = tfkl.Conv2D(
        filters = n_f,
        kernel_size = (h_r,h_c),
        activation = 'relu',
        strides = (1,1),
        padding = 'same',
        name = 'conv2'
    )
```

→ see 3 slides later.

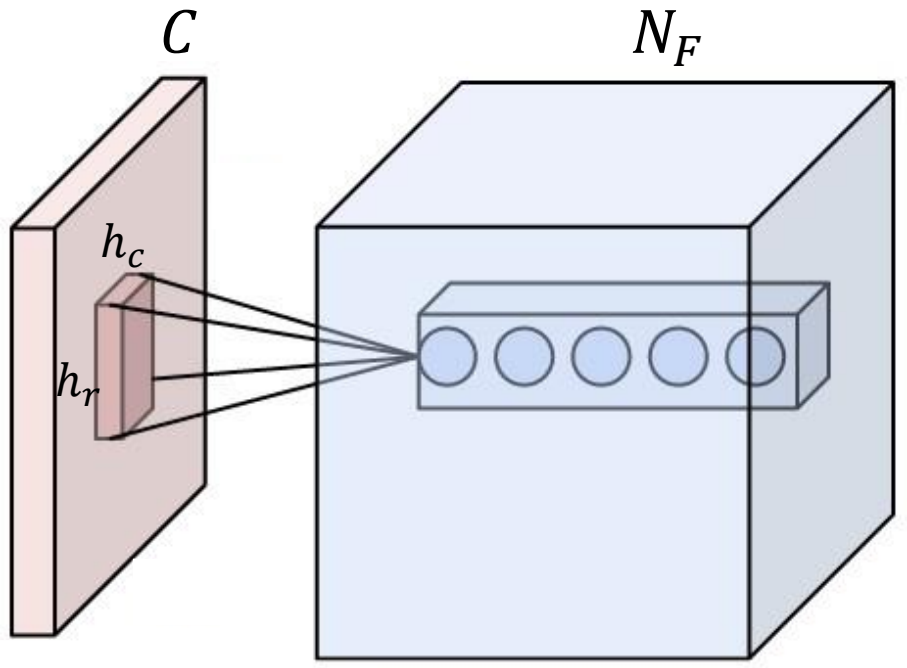
The parameters are the weights + one bias per filter

The overall number of parameters is

$$(h_r \cdot h_c \cdot C) \cdot N_F + N_F$$

Where d is the **depth of the input activation**

Layers with the same attribute can have different number of parameters depending on where these are located

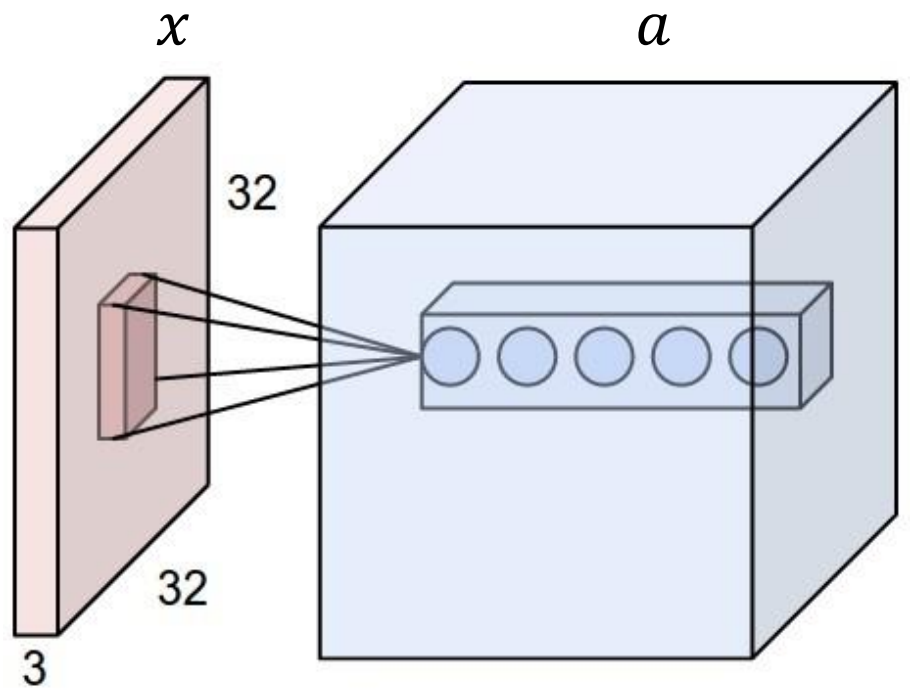


By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

Recap: Convolutional Layers

Convolutional layers "mix" all the input components

- The **output** is also called **volume** or **activation maps**
- Each **filter yields a different slice** of the output volume
- **Each filter has depth equal to the depth of the input volume**

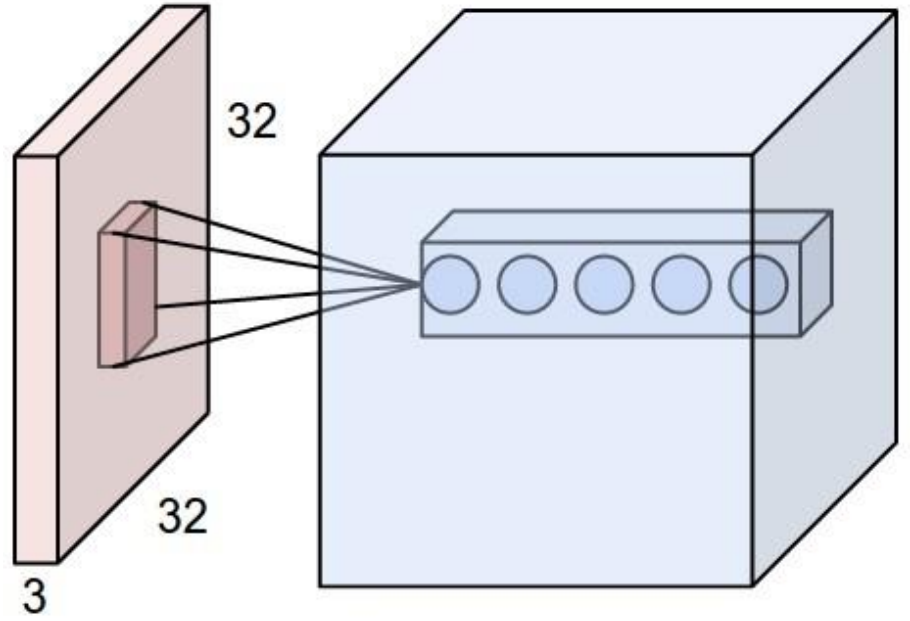


By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

Convolutional Layers, remarks:

- Convolutional Layers are described by a set of **filters**
- **Filters represent the weights of these linear combination.**
- Filters have **very small spatial extent and large depth extent**,
- The **filter depth** is typically not specified, as it corresponds to the **number of layers of the input volume**

The output of the convolution
against a filter becomes a slice
in the volume feed to the next layer



By Aphex34 - Own work, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=45659236

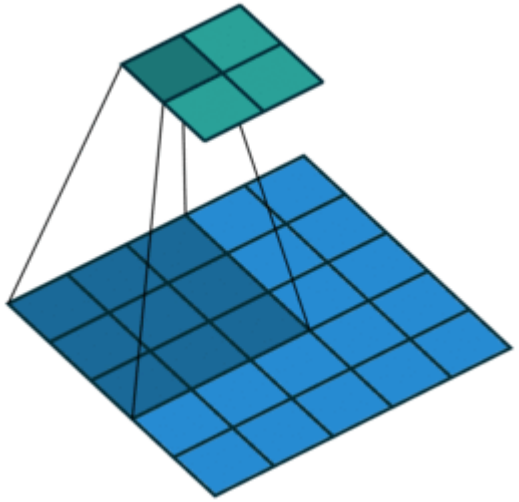
Stride Options in Convolutions

This is not very popular in image processing, but **used in CNNs**.

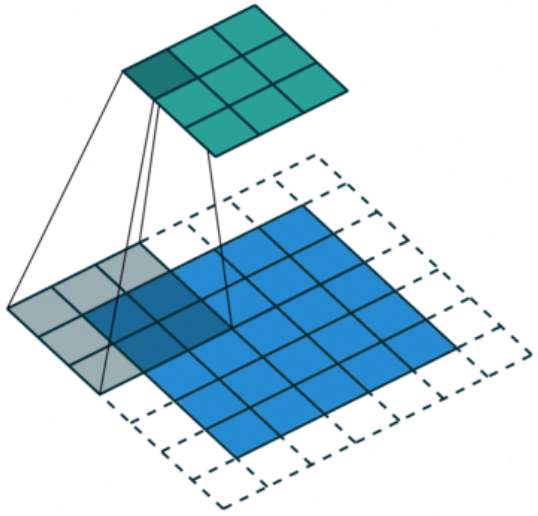
It is possible to **reduce the spatial extent by** «sliding» the filter over each channel of the input volume with a ***stride***. The larger the stride, the more the spatial dimensions get shrinked.

Stride = (1,1) corresponds to standard convolution, Stride = (2,2) means convolutional filter «jumps» every second row and every second column

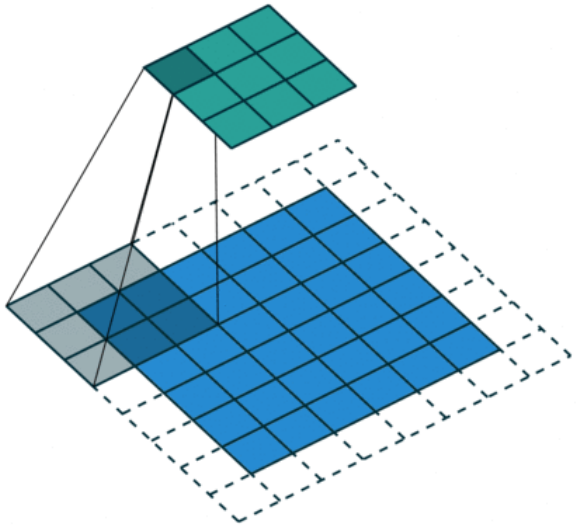
In principle this corresponds to convolution followed by downsampling (less efficient).



No Padding, stride 2



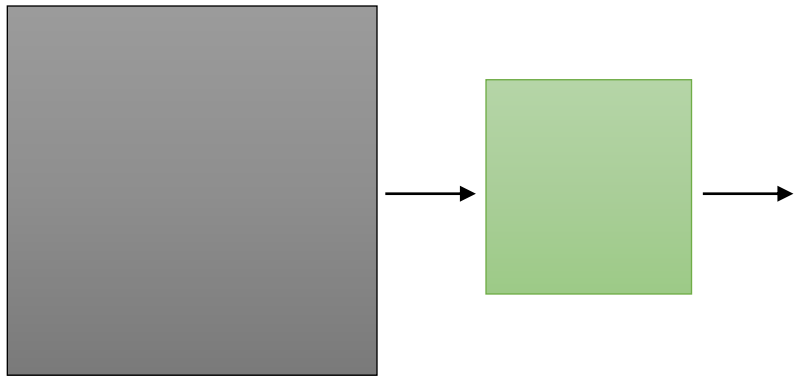
Padding, stride 2



Padding, stride 2 (odd)

Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

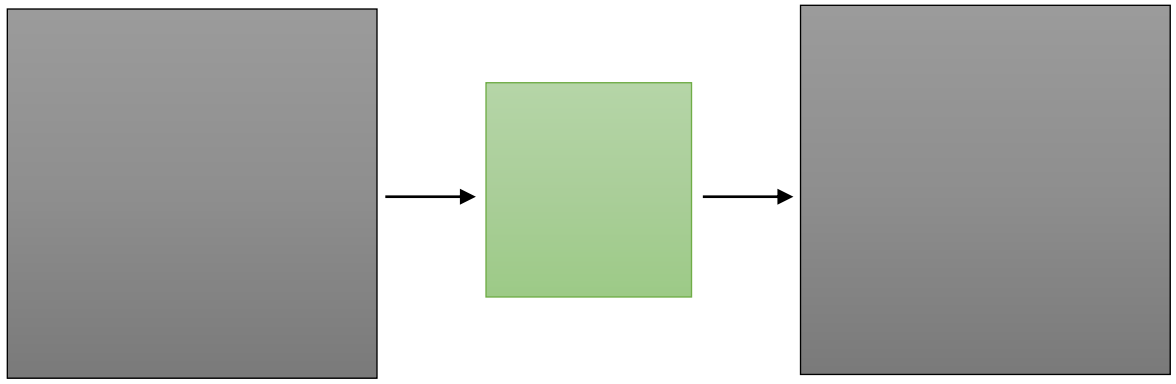
CNN Arithmetic



1 input map

1 filter 5x5

CNN Arithmetic

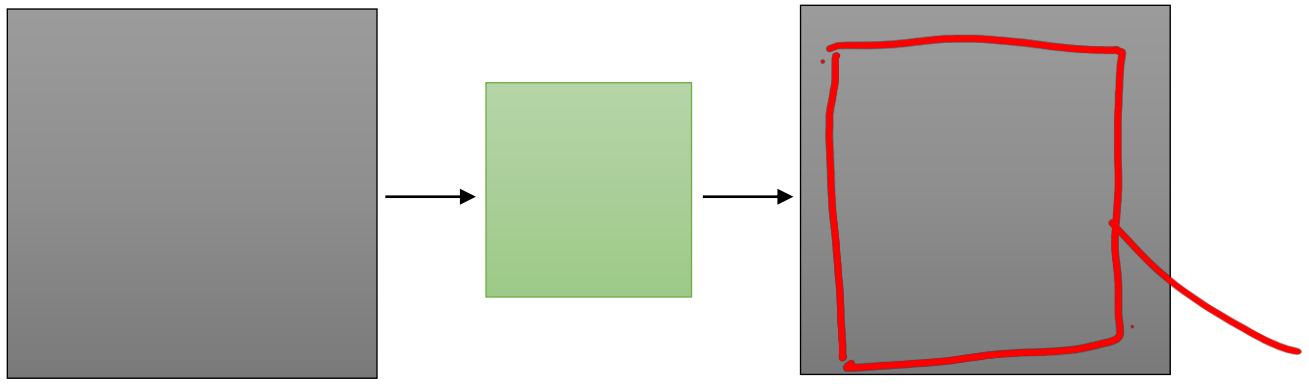


1 input map

1 output map

1 filters 5x5

CNN Arithmetic



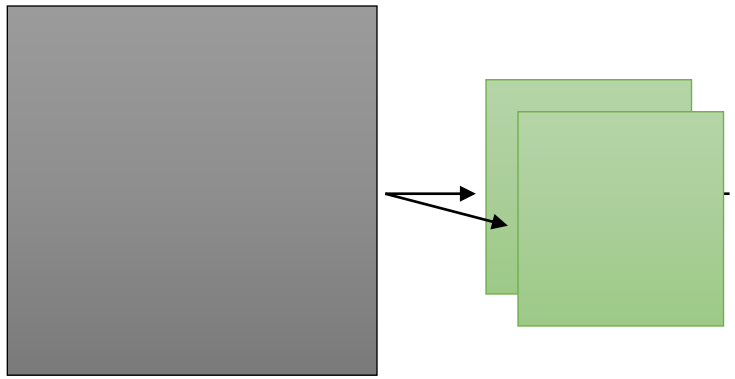
1 input map

1 filters 5x5

1 output map

in case of padding "valid".

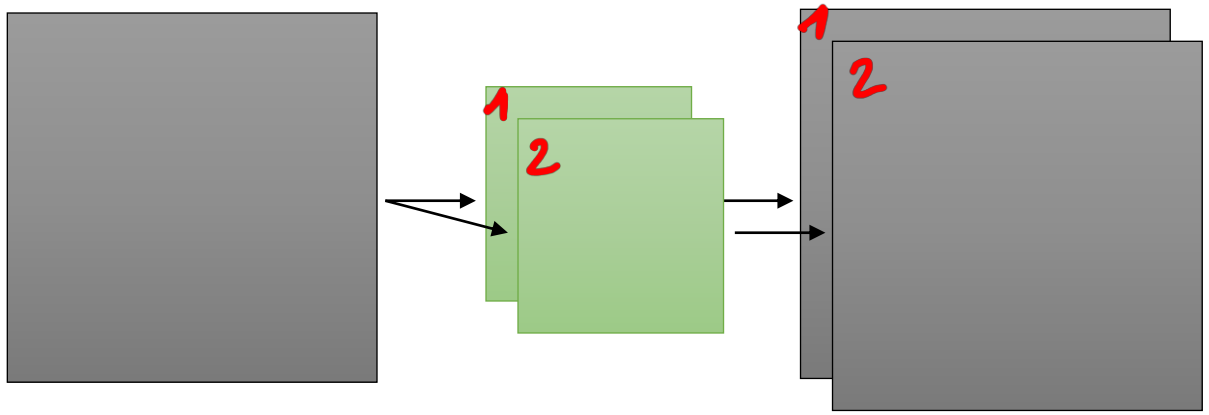
CNN Arithmetic



1 input map

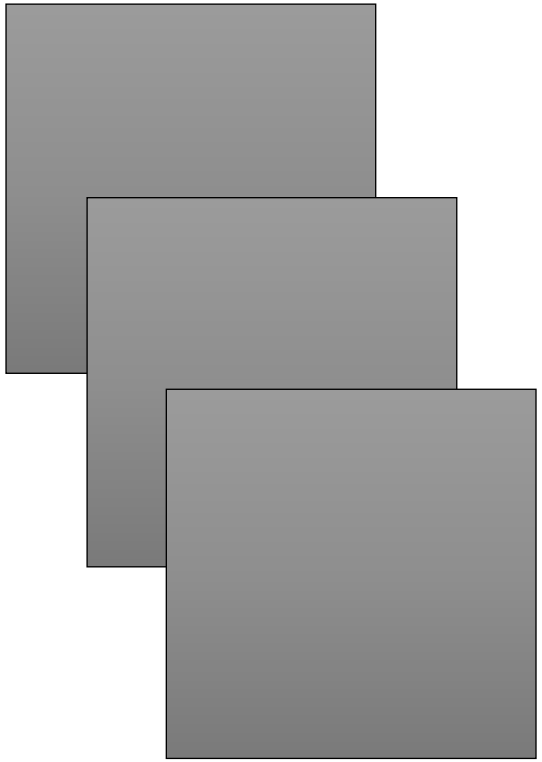
2 filters 5x5

CNN Arithmetic



1 input map 2 filters 5x5 2 output maps

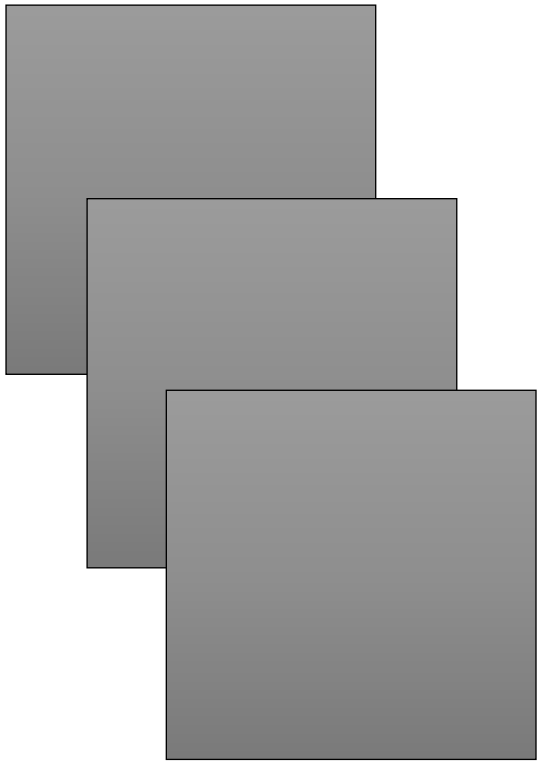
CNN Arithmetic

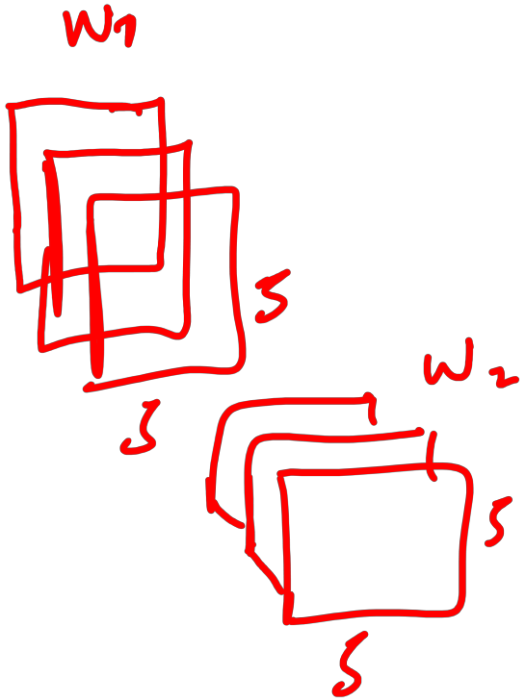


3 input maps

2 filters 5x5

CNN Arithmetic





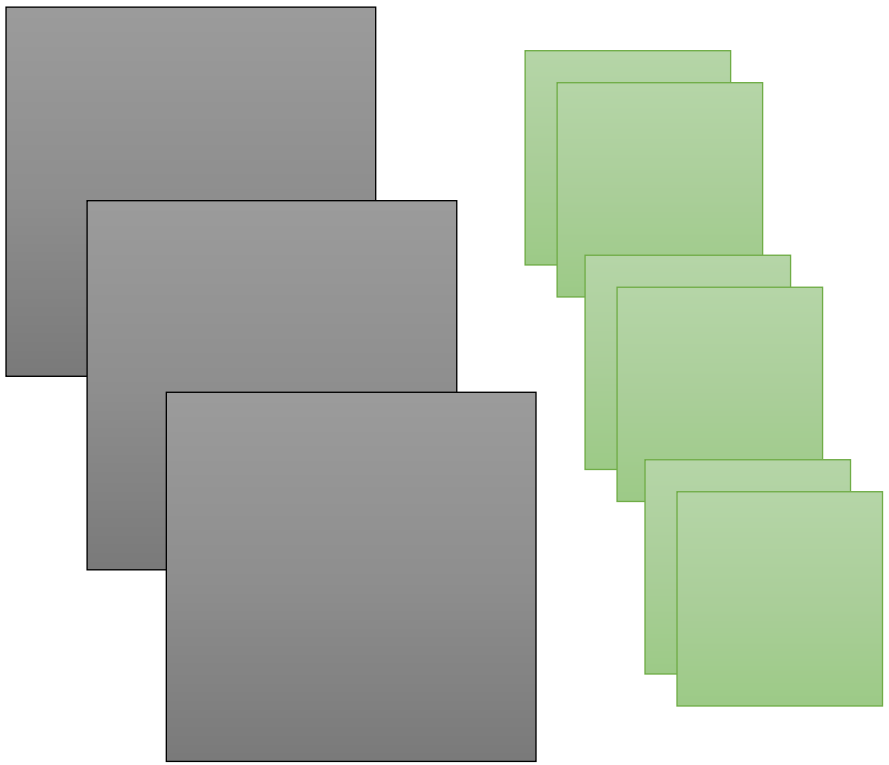
3 input maps

2 filters 5x5

↓

3 channels per filter.

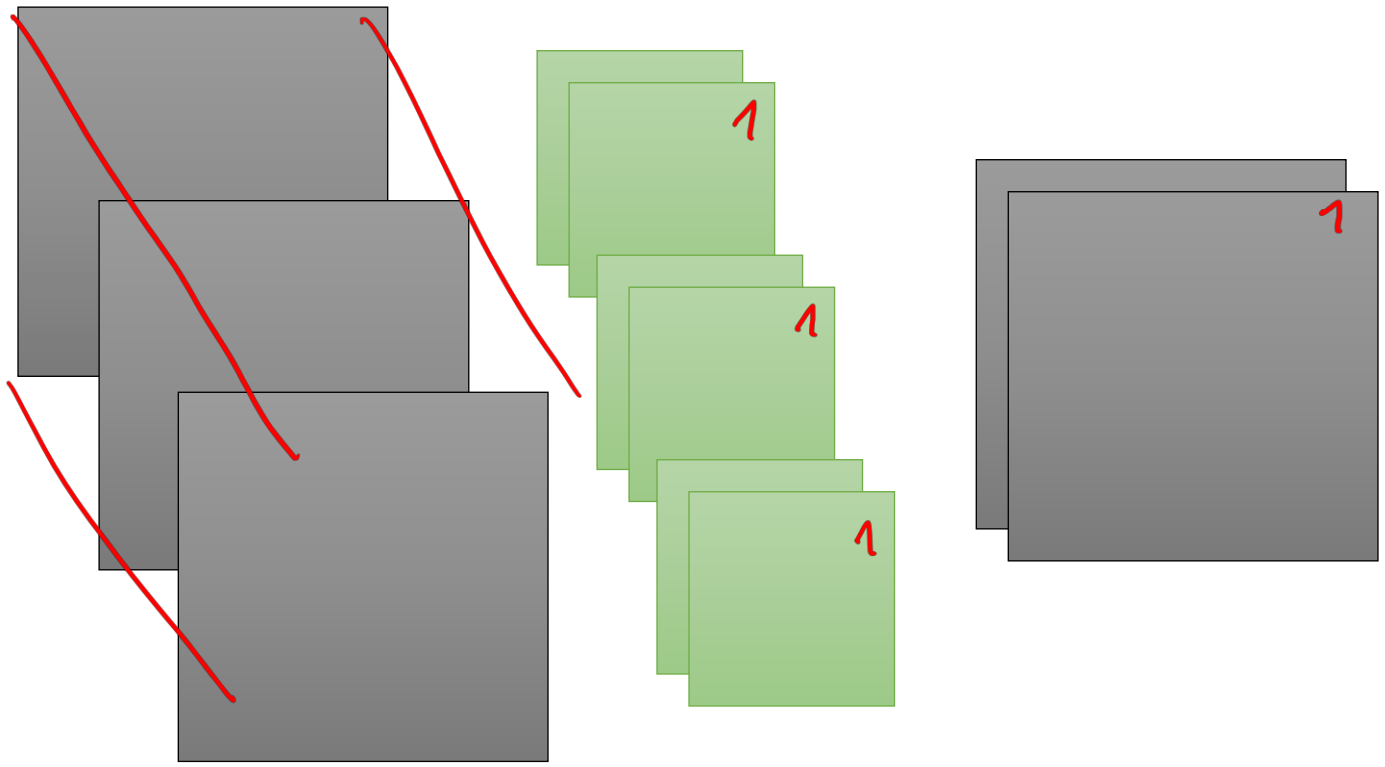
CNN Arithmetic



3 input maps

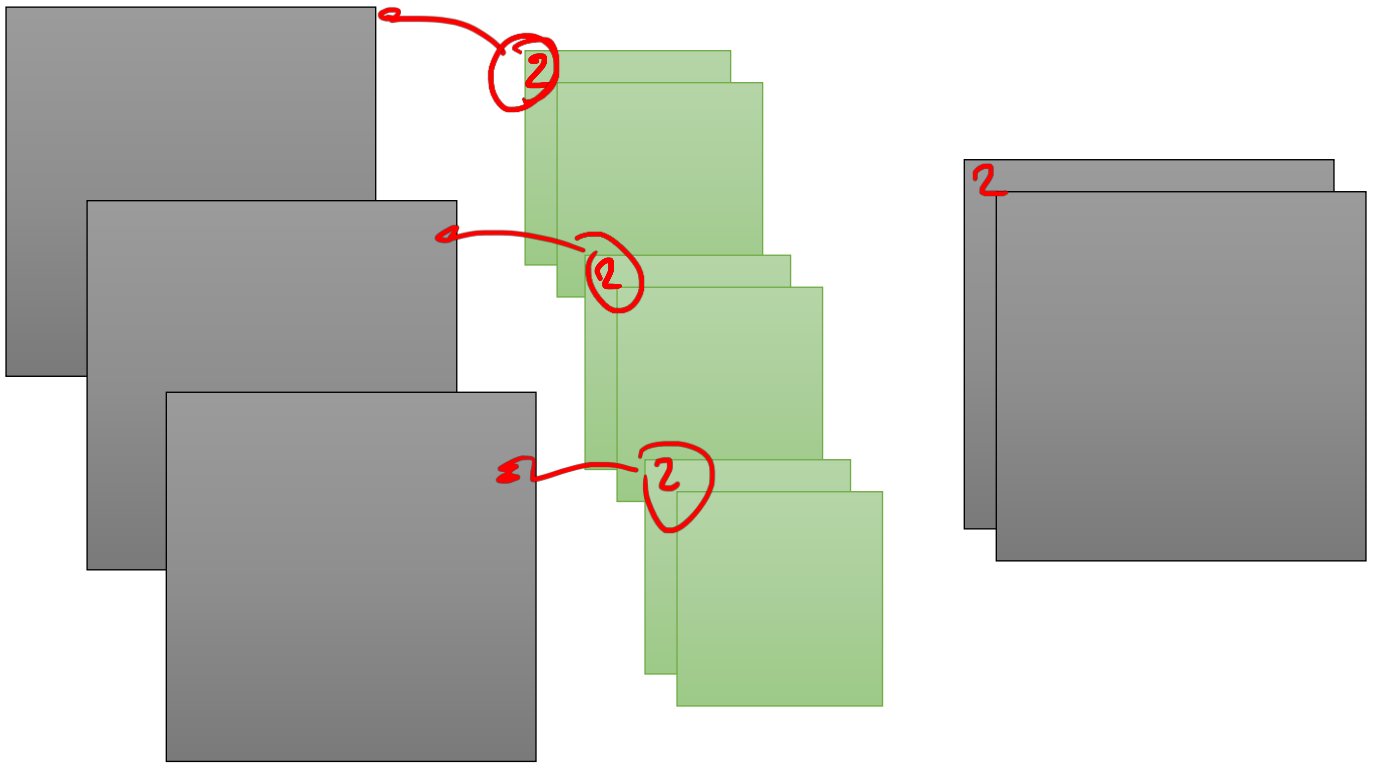
2 filters 5x5

CNN Arithmetic



3 input maps 2 filters 5x5 2 output maps

CNN Arithmetic



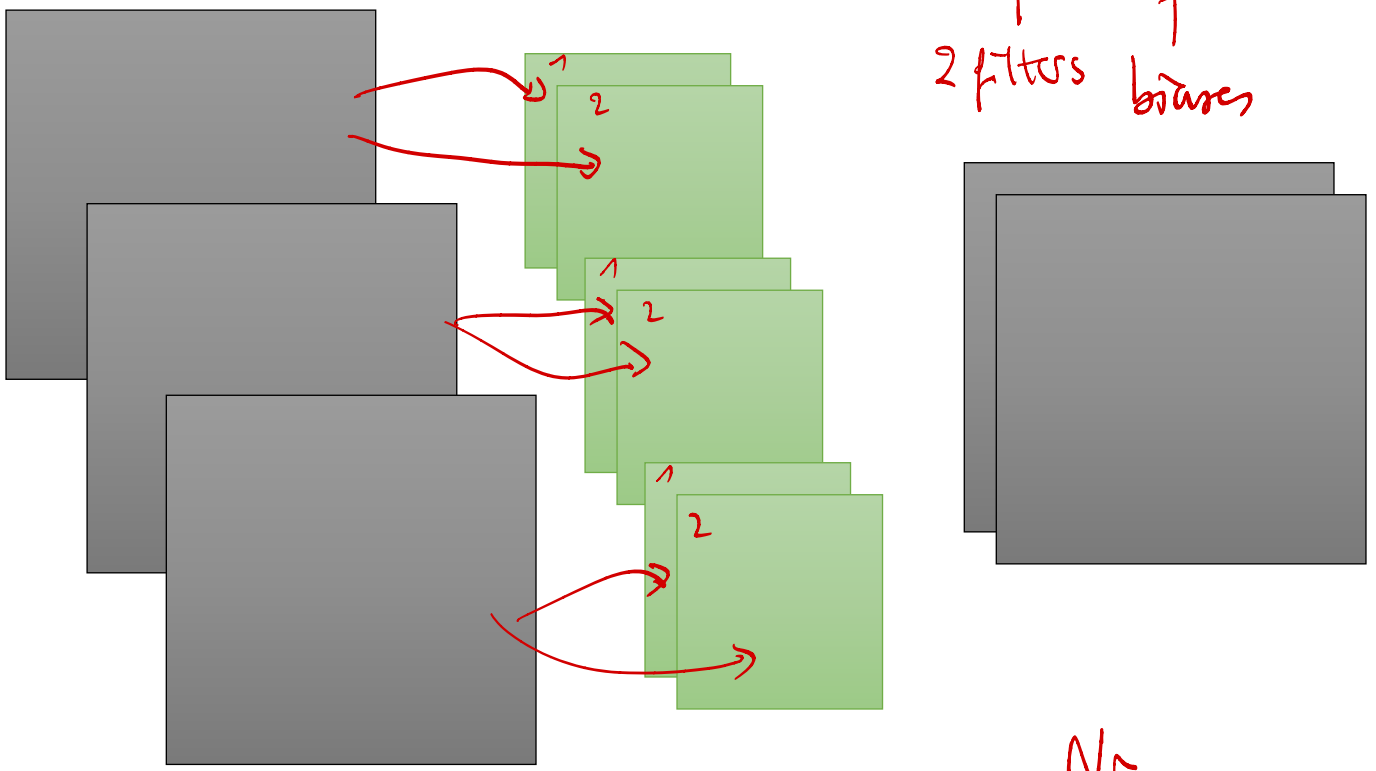
3 input maps 2 filters 5x5 2 output maps

CNN Arithmetic

3 channels <=> 3 input images.

$(5 \times 5 \times 3) \times 2 + 2 = 152$

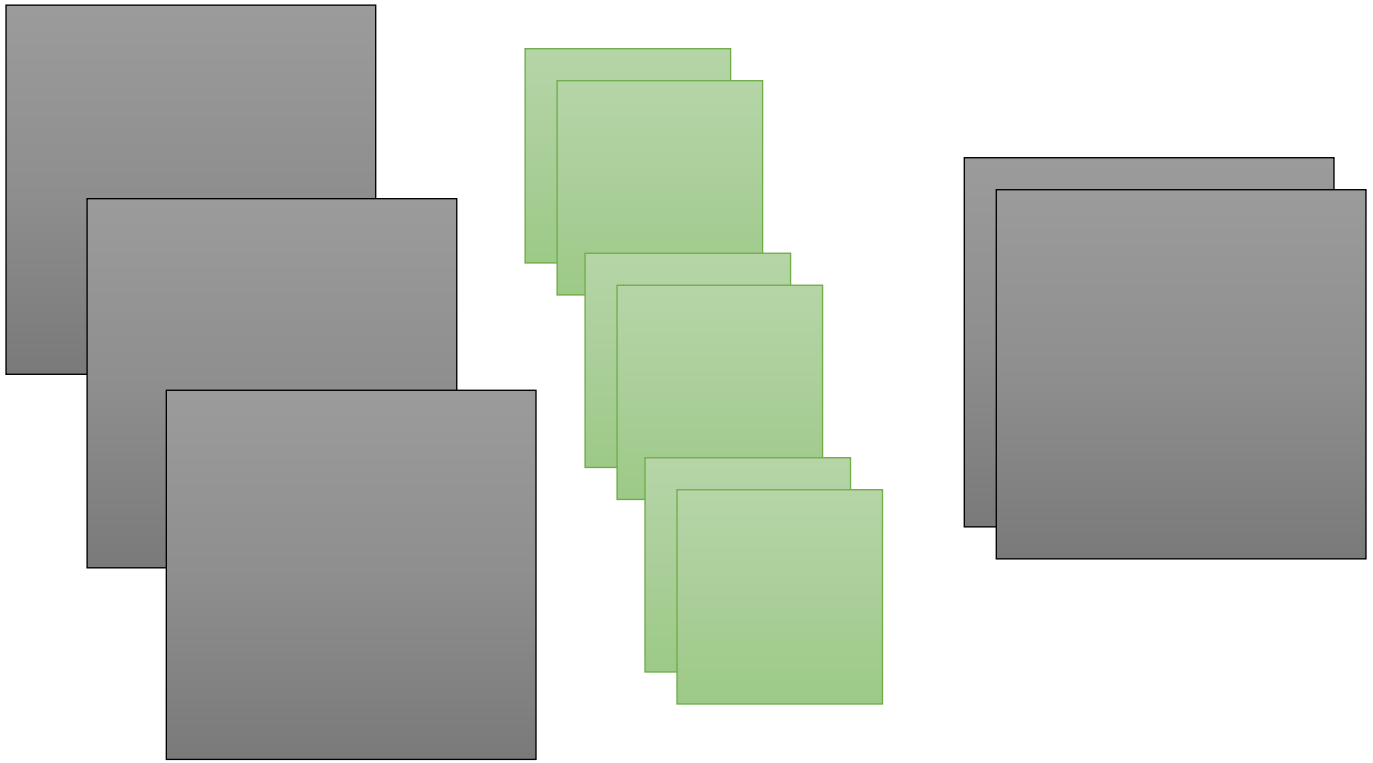
2 filters, biases



3 input maps 2 filters 5x5 N_f 2 output maps

Quiz: how many parameters does this layer have?

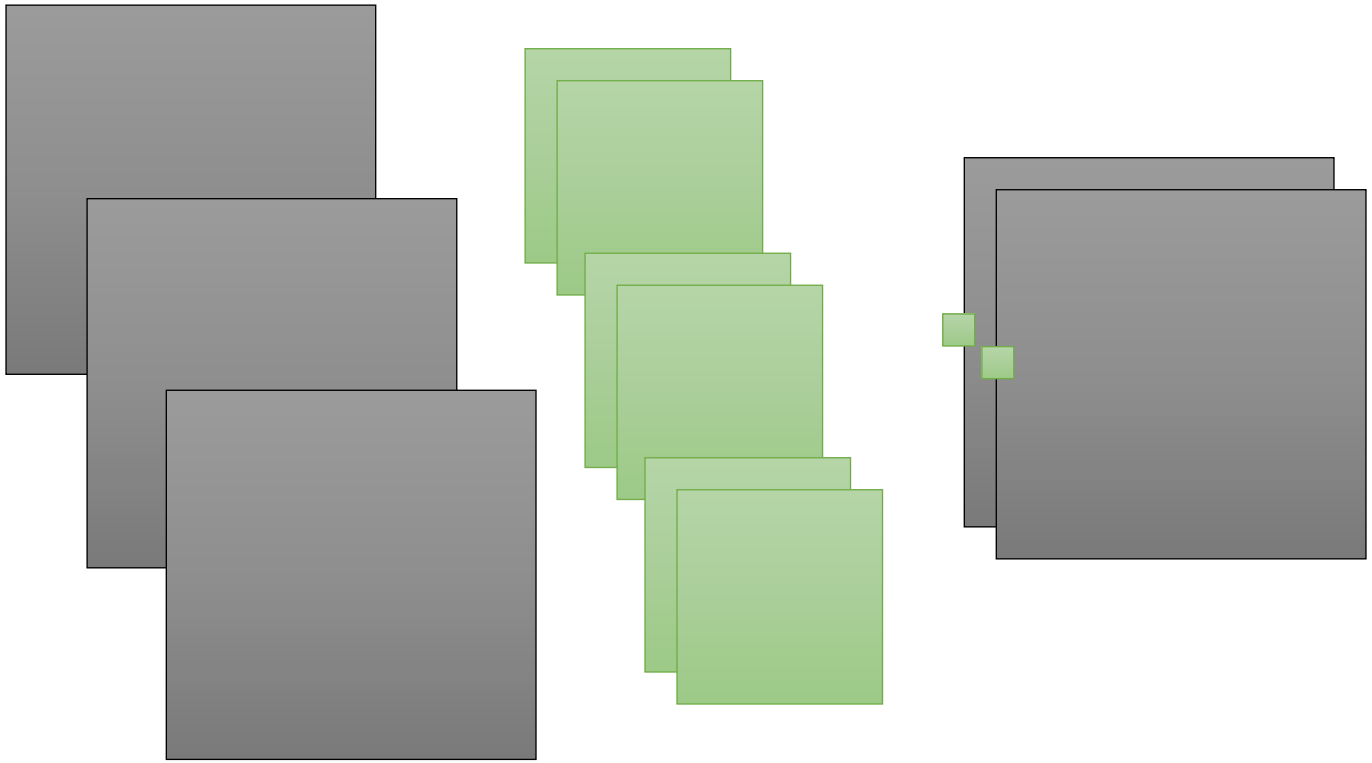
CNN Arithmetic



3 input maps 2 filters 5x5 2 output maps

= 150 parameters in the filters

CNN Arithmetic

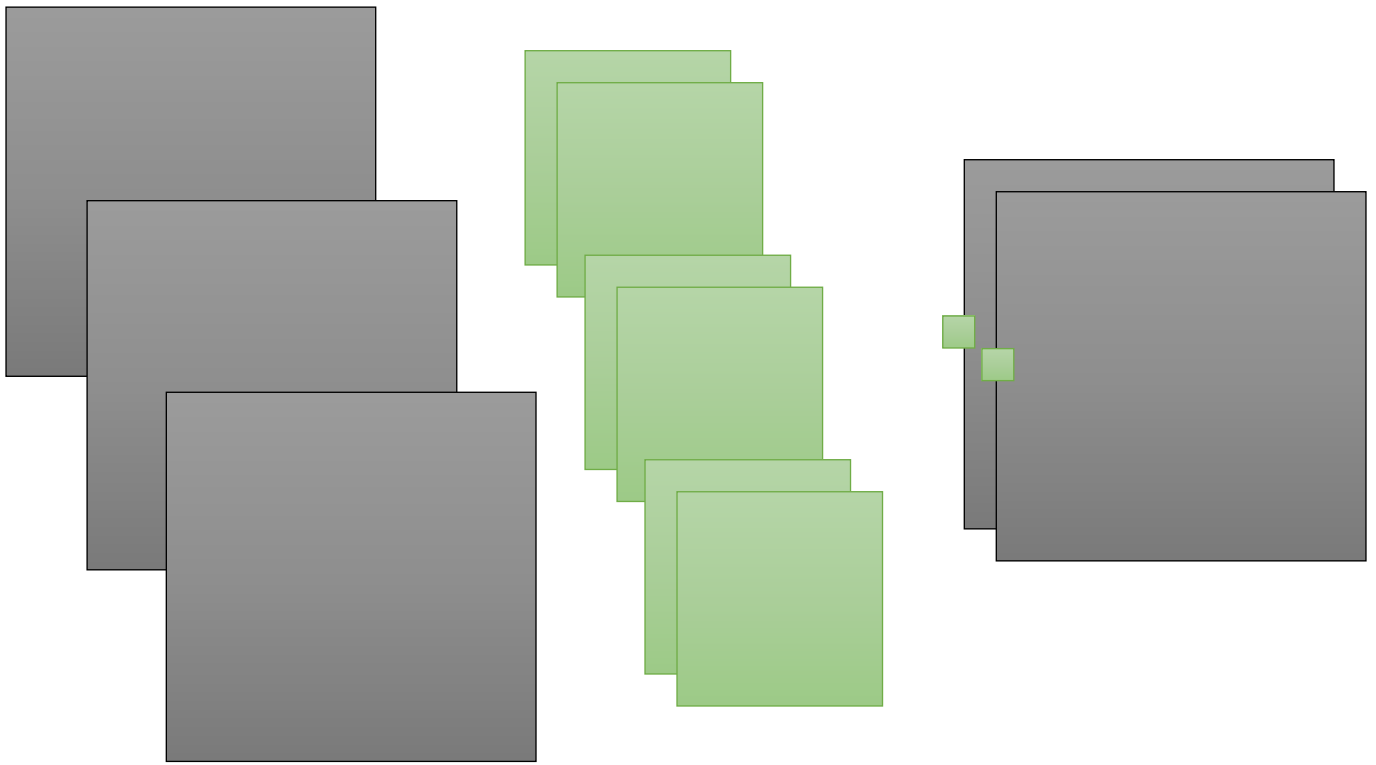


3 input maps

2 filters 5x5
= 150 ...

2 output maps
+ 2 biases

CNN Arithmetic



output maps = # of filters in the layer = N_F

3 input maps

2 filters 5x5

= 150 ...

2 output maps

+ 2 biases

= 152 trainable parameters (weights)

To Recap...

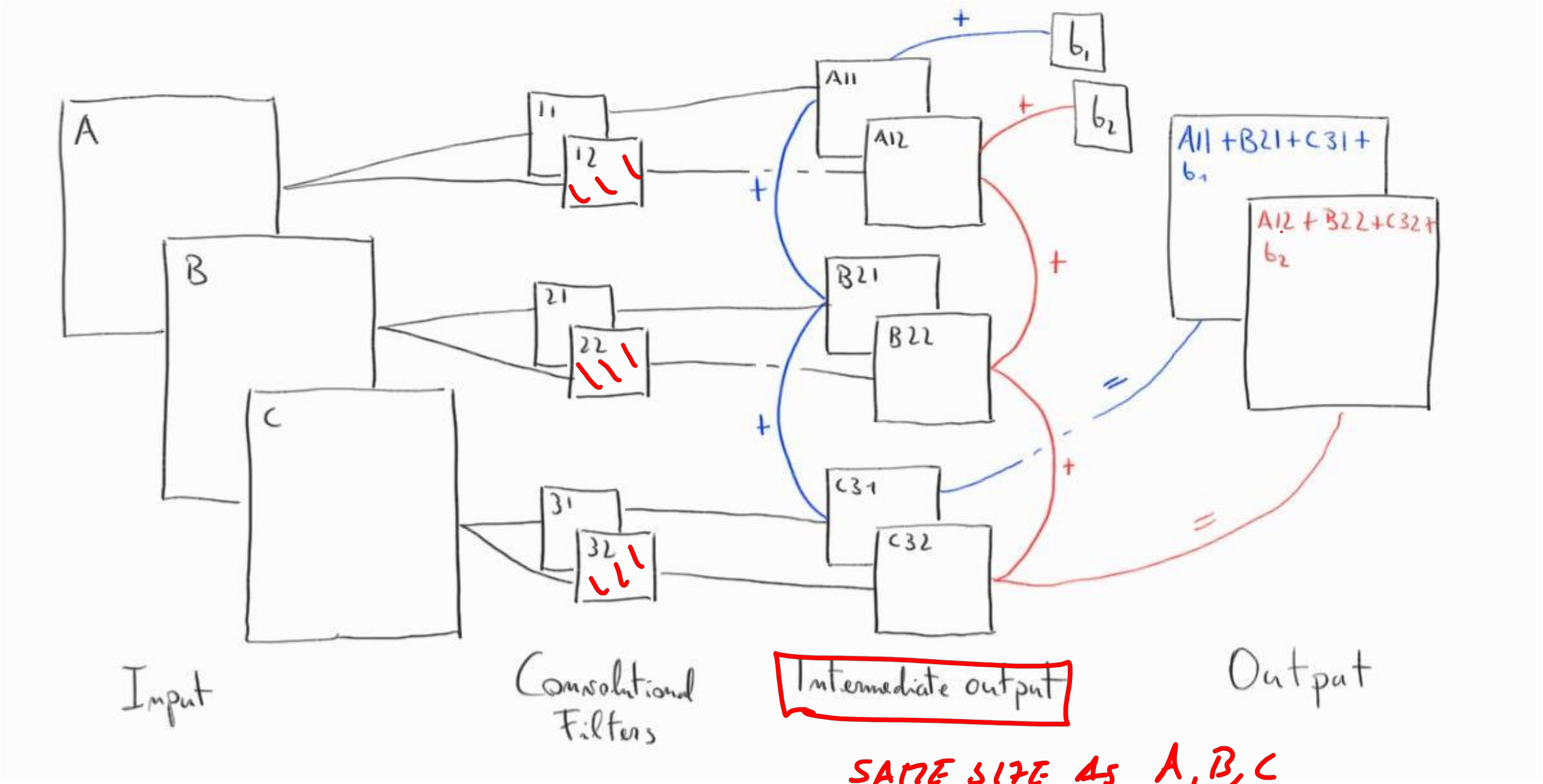


Image credits Nicolò Ghielmetti

(*) They are given by activation functions.

Other Layers

Activation and Pooling

Not only convolution, otherwise everything will be linear...

→ we need non-linearity (*)

Activation Layers

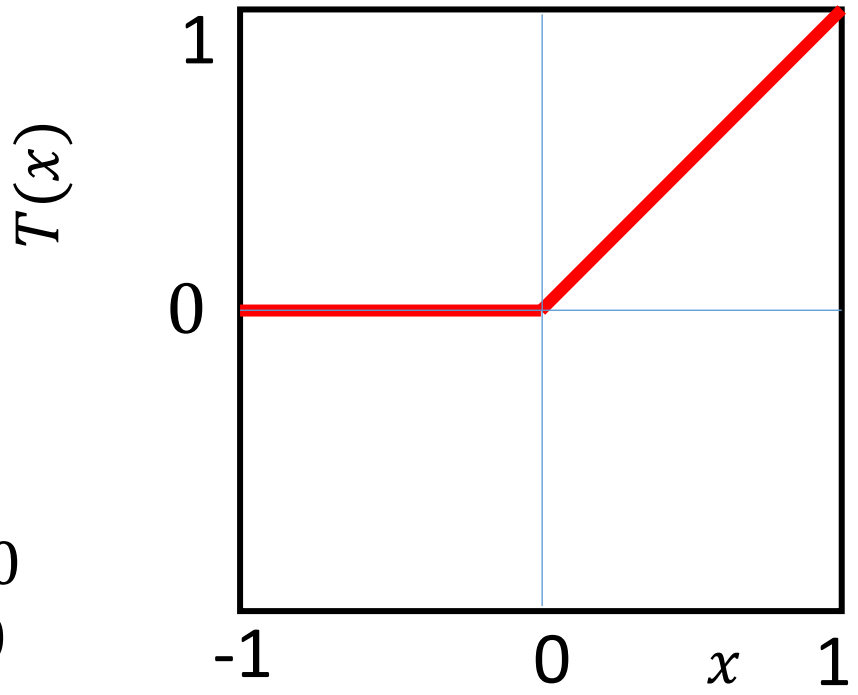
Introduce nonlinearities in the network, otherwise the CNN might be equivalent to a linear classifier..

Activation functions are scalar functions, namely they operate on each single value of the volume. **Activations don't change volume size**

RELU (Rectifier Linear Units): it's a thresholding on the feature maps, i.e., a $\max(0,\cdot)$ operator.

- By far the most popular activation function in deep NN (since when it has been used in AlexNet)
- Dying neuron problem: a few neurons become insensitive to the input (vanishing grandient problem)

$$T(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$$

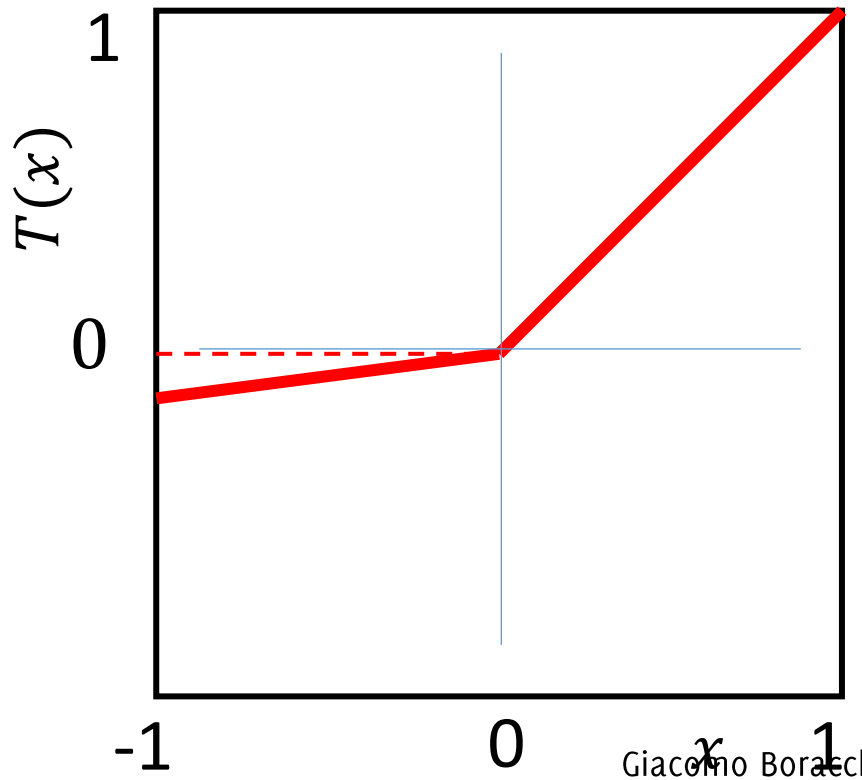


Activation Layers

Introduce nonlinearities in the network, otherwise the CNN might be equivalent to a linear classifier…

LEAKY RELU: like the relu but include a small slope for negative values

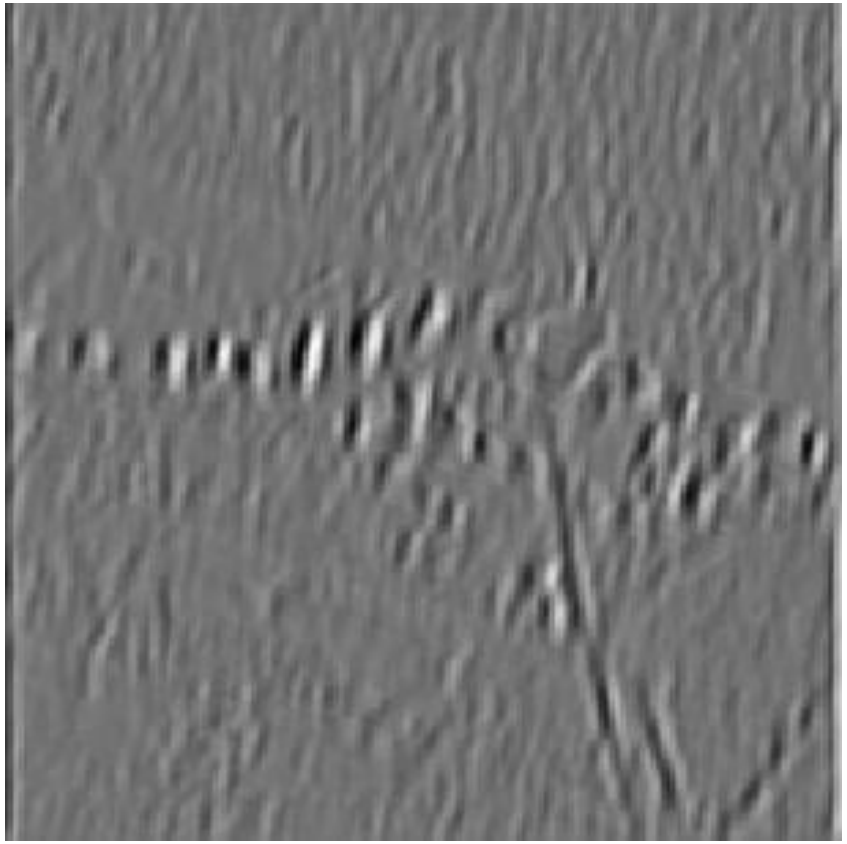
$$T(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0.01 * x & \text{if } x < 0 \end{cases}$$

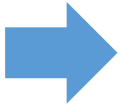


ReLu

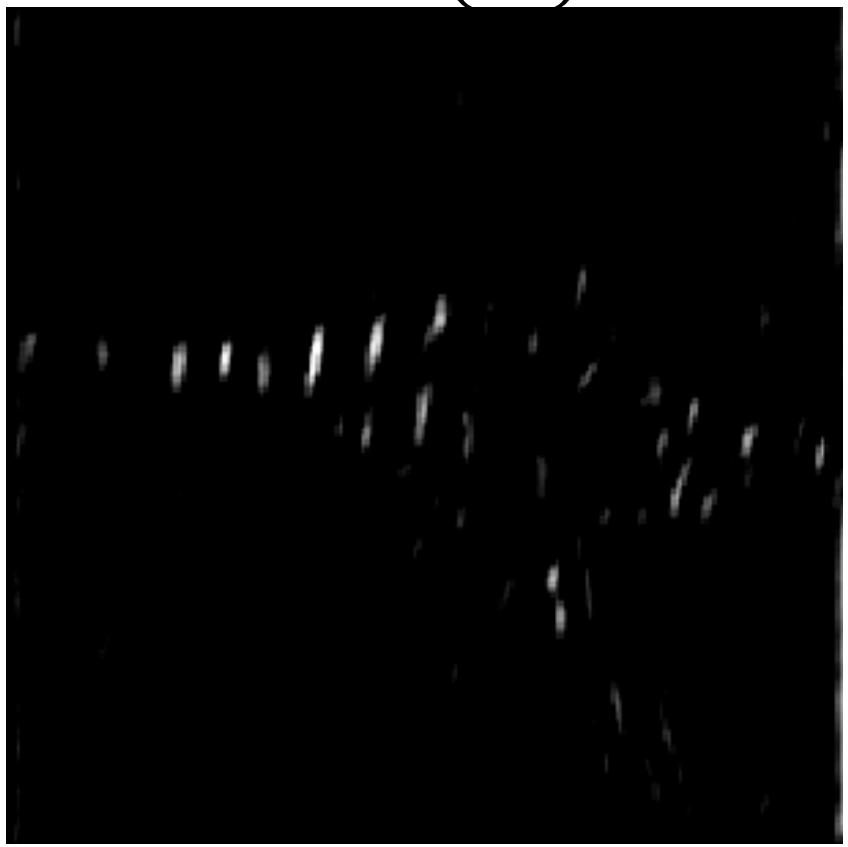
Acts separately on each layer

a^1





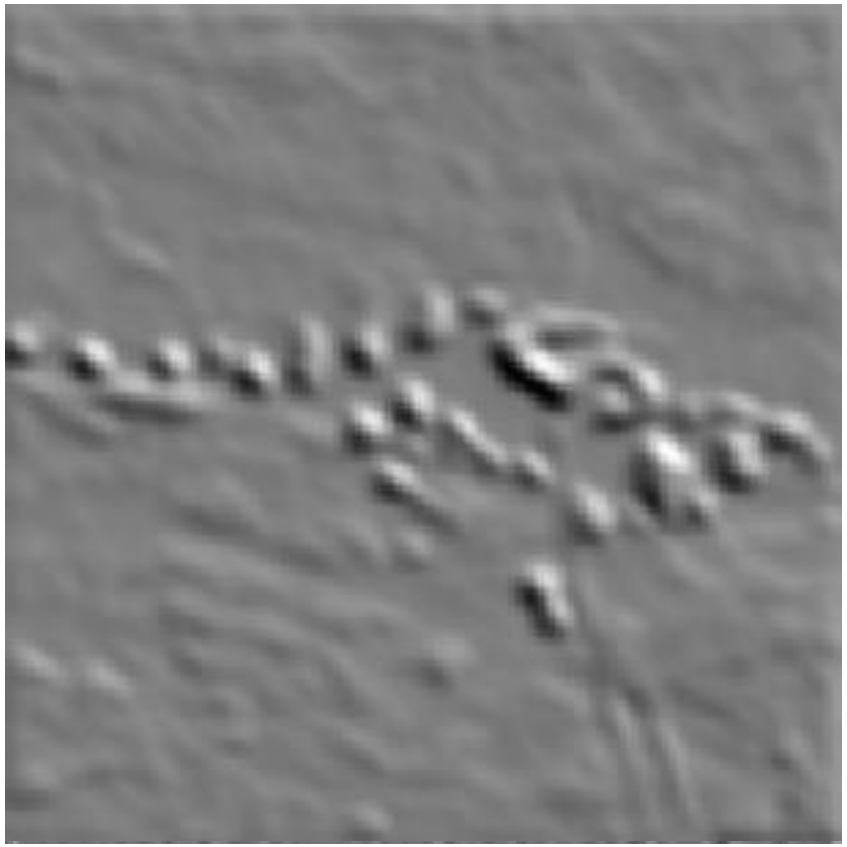
$ReLU(a^1)$



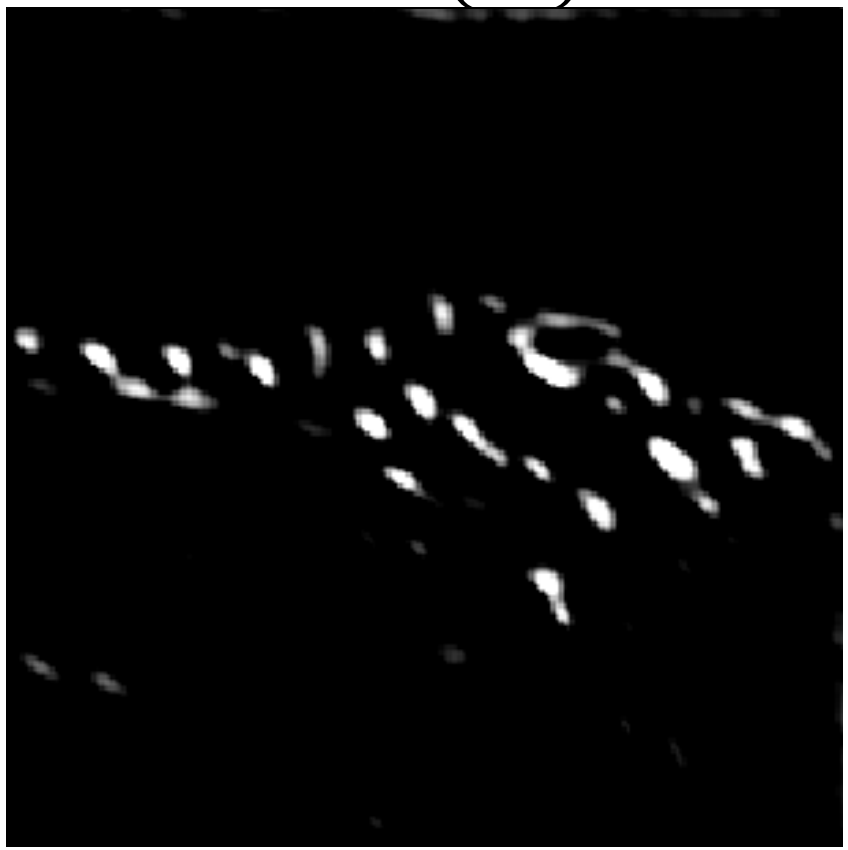
ReLu

Acts separately on each layer

a^2



$ReLU(a^2)$



Activation Layers

Introduce nonlinearities in the network, otherwise the CNN might be equivalent to a linear classifier…

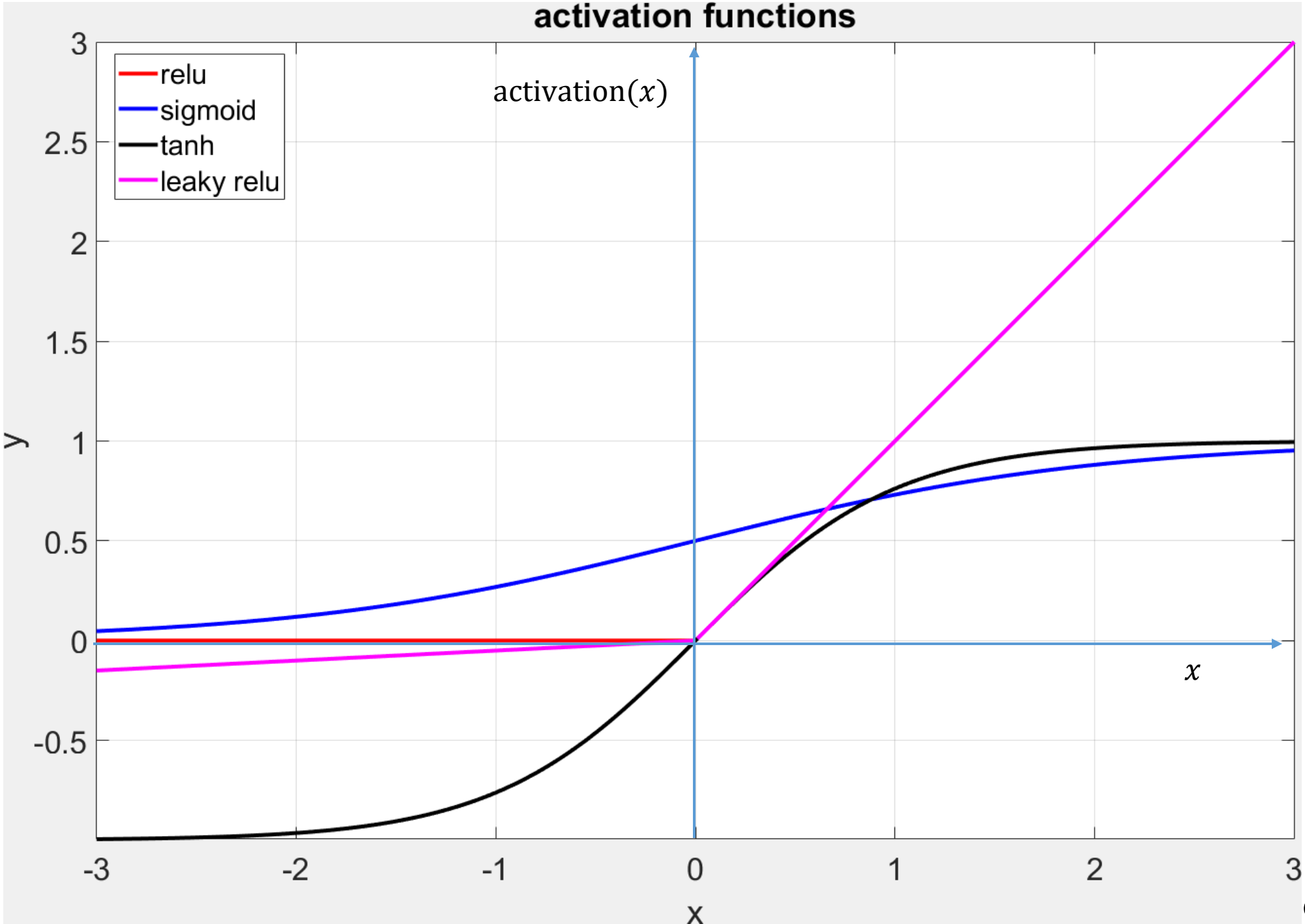
TANH (hyperbolic Tangent): has a range (-1,1), continuous and differentiable

$$T(x) = \frac{2}{1 + e^{-2x}} - 1$$

SIGMOID: has a range (0,1), continuous and differentiable

$$S(x) = \frac{1}{1 + e^{-2x}}$$

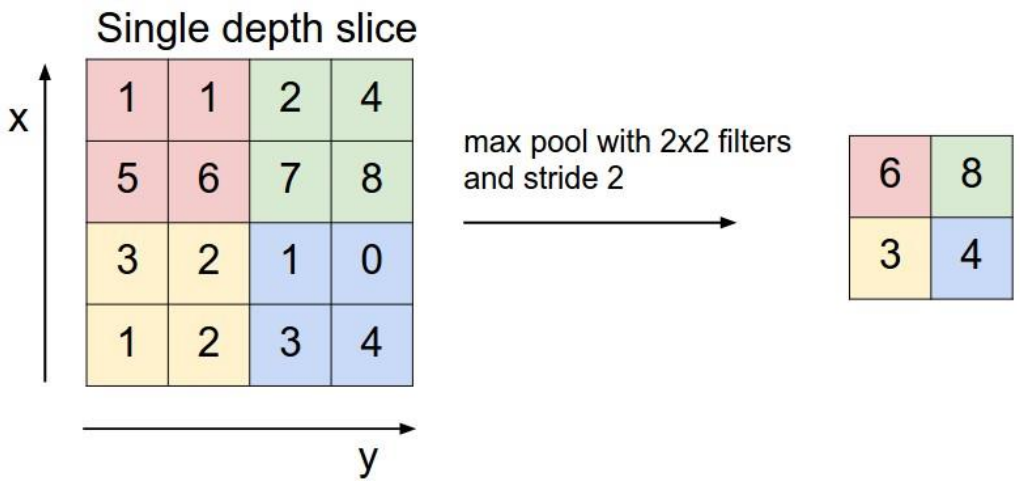
These activation functions are mostly popular **in MLP architectures**

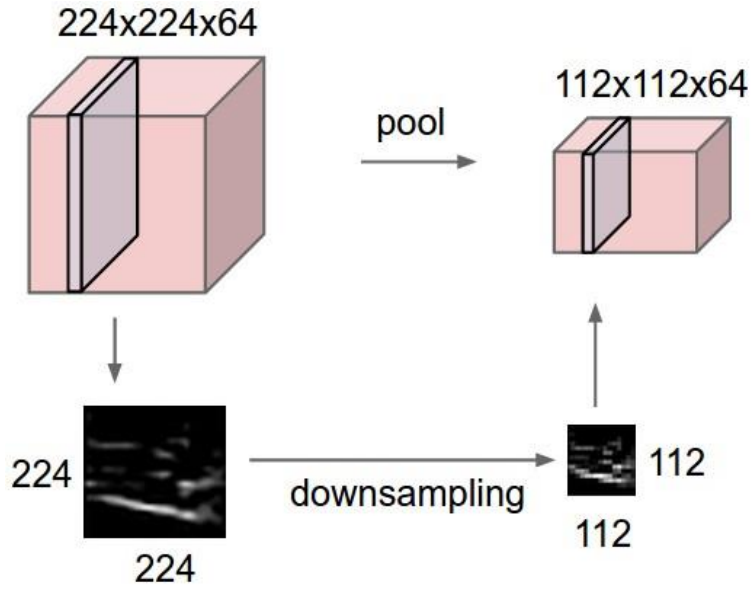

activation functions
relu
sigmoid
tanh
leaky relu
activation(x)
x
y
3
2.5
2
1.5
1
0.5
0
-0.5
-3
-2
-1
0
1
2
3
x

Pooling Layers

Pooling Layers reduce the **spatial** size of the volume.

The Pooling Layer operates **independently on every depth slice of the input and resizes it spatially, often using the MAX operation.**

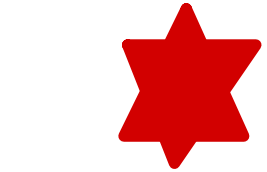




In a 2x2 support it discards 75% of samples in a volume

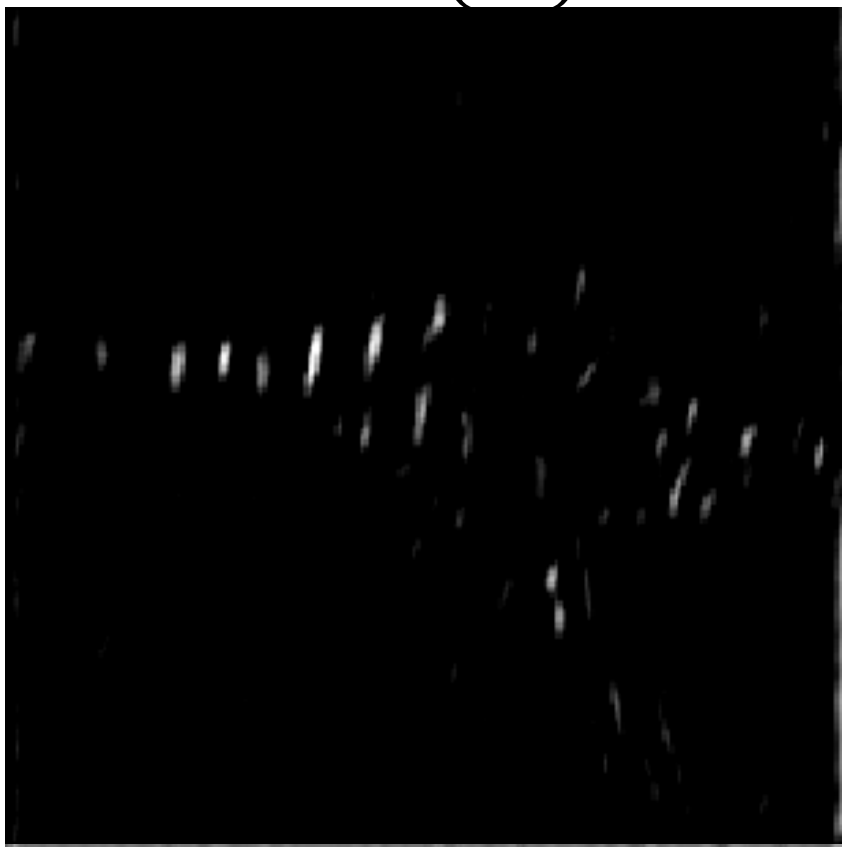
Max-Pooling (MP)

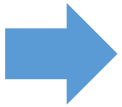
It's an additional non-linearity.



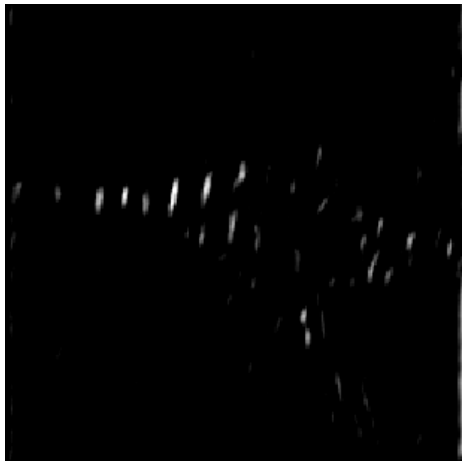
Acts separately on each layer

$ReLU(a^1)$





$\mathrm{MP}(ReLU(a^1))$



Strides in Pooling Layers

so it reduces the size ...

Typically, the **stride is assumed equal to the pooling size**

- Where note specified, maxpooling has stride 2×2 and reduces image size to 25%

It is also possible to use a different stride. In particular, it is possible to adopt stride = 1, which does not reduce the spatial size, but just perform pooling on each pixel

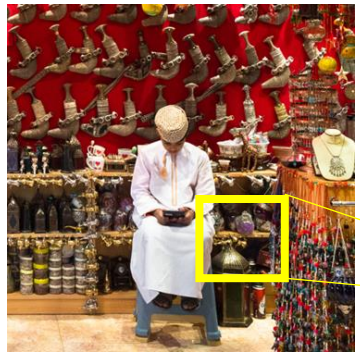
- this operation makes sense with nonlinear pooling (max-pooling)

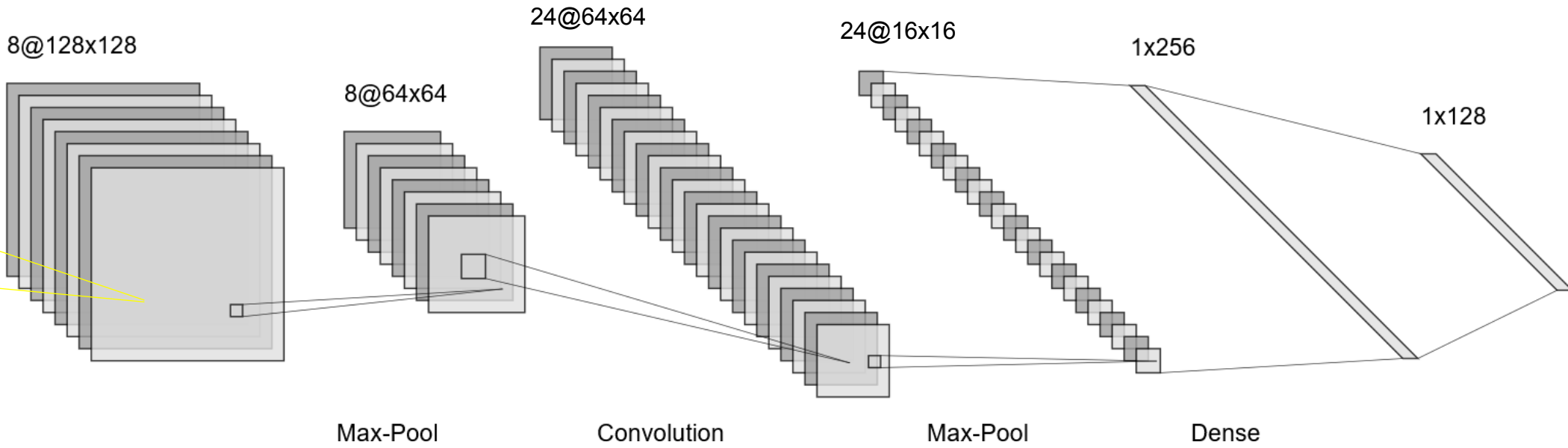
Dense Layers

As in feed-forward NN

The typical architecture of a CNN

The input of a CNN is an entire image

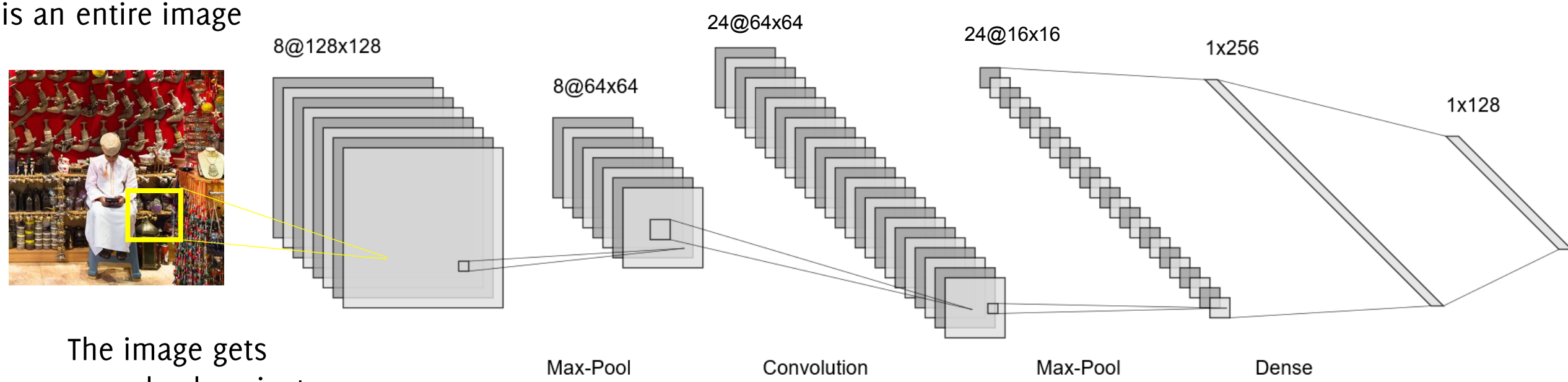




The image gets convolved against many filters

The typical architecture of a CNN

The input of a CNN is an entire image

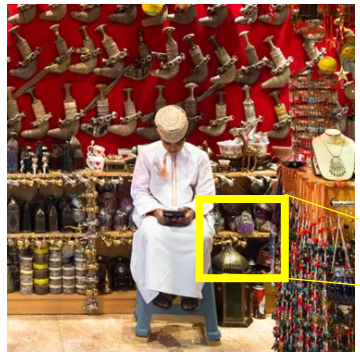


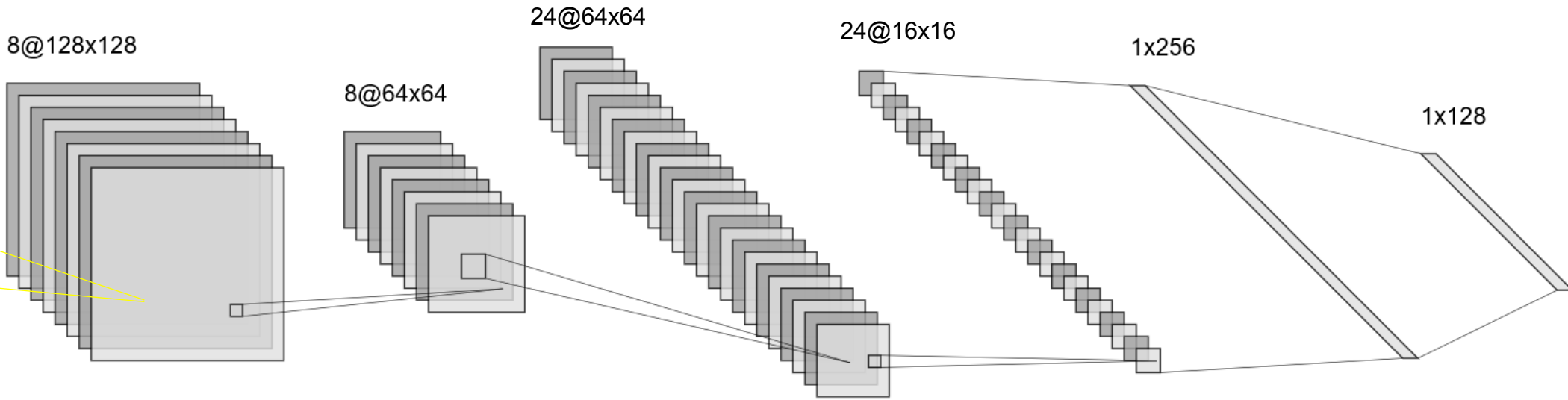
The image gets convolved against many filters

When progressing along the network, the «number of images» or the «number of channels in the images» increases, while the image size decreases

The typical architecture of a CNN

The input of a CNN is an entire image



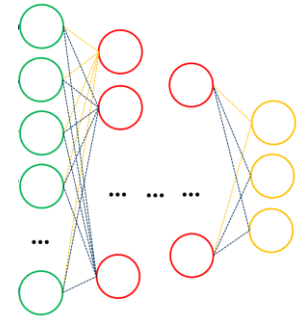


The image gets convolved against many filters

Max-Pool Convolution Max-Pool Dense

When progressing along the network, the «number of images» or the «number of channels in the images» increases, while the image size decreases

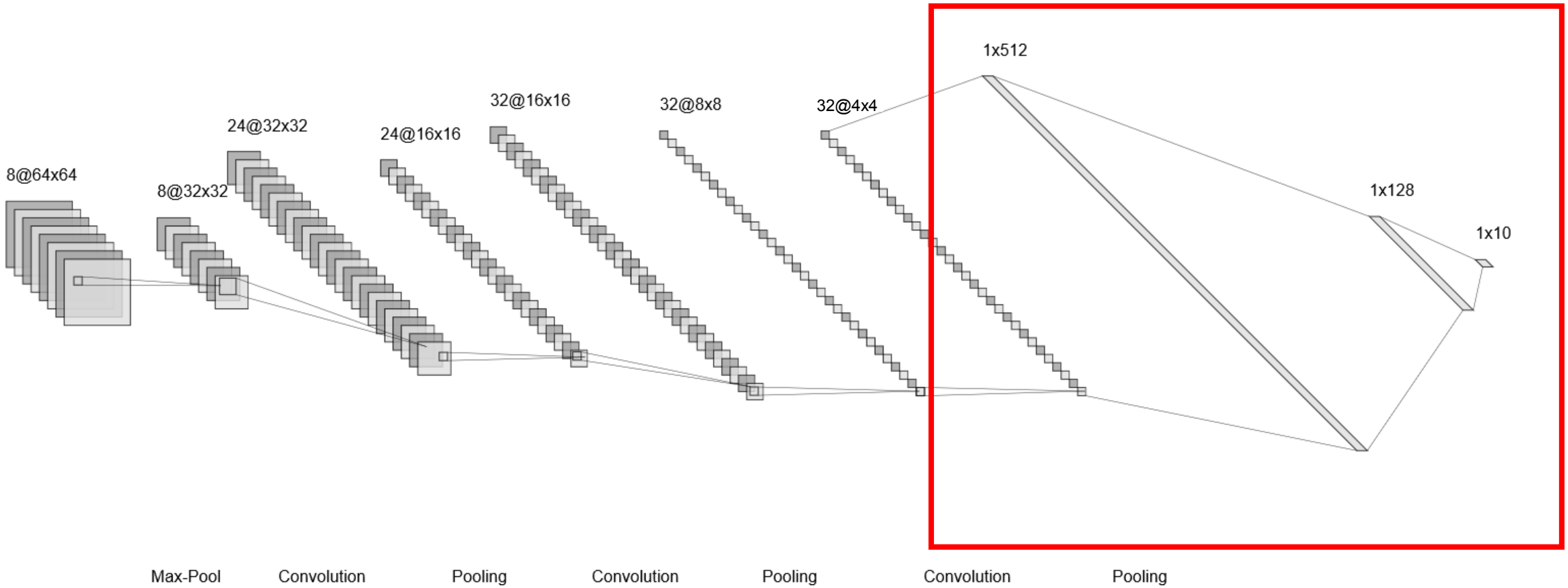
Once the image gets to a vector, this is fed to a traditional neural network



Giacomo Boracchi

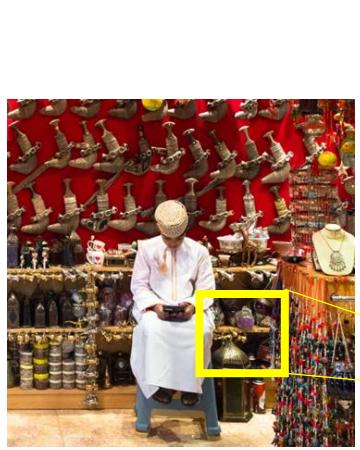
The Dense Layers

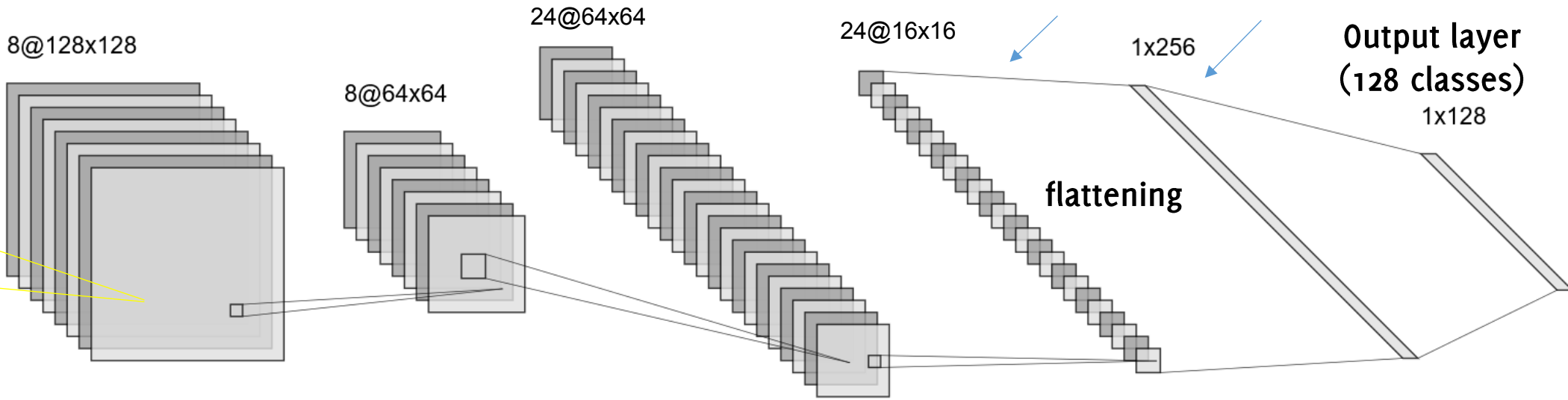
Here the spatial dimension is lost, the CNN stacks hidden layers from a MLP NN.
It is called Dense as each output neuron is connected to each input neuron



Convolutional Neural Networks (CNN)

Let's analyze our network in detail





Assume zero padding

Convolution: There are 8 filters

Max-pooling halves spatial dimensions

Convolution: There are 24 filters

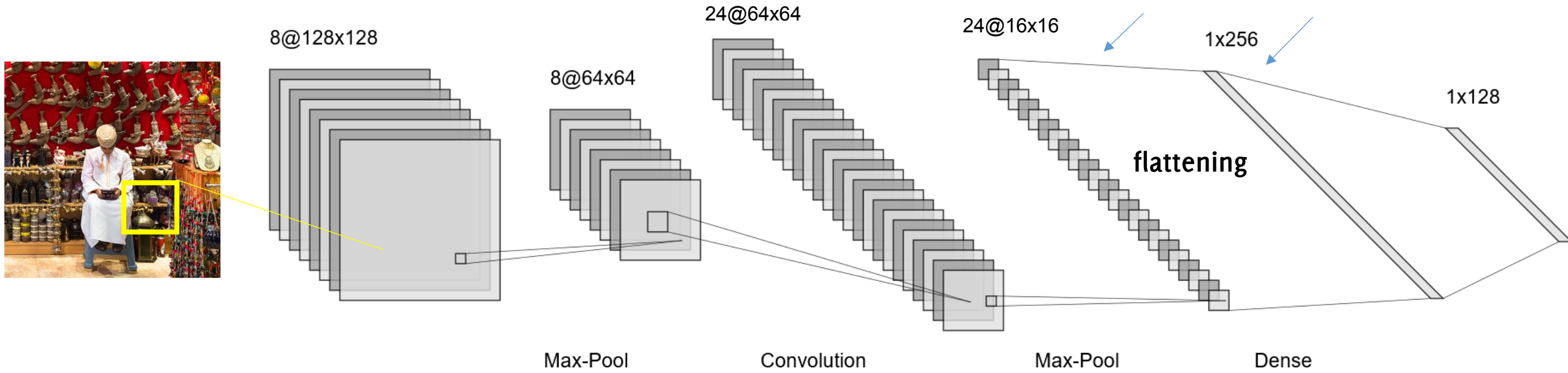
Max-pooling halves spatial dimensions

A flattening layer brings the volume into a 16x16x24 dimensional vector

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. *"Gradient-based learning applied to document recognition"* Proceedings of the IEEE, 1998 86(11), 2278-2324.

Convolutional Neural Networks (CNN)

Let's analyze our network in detail

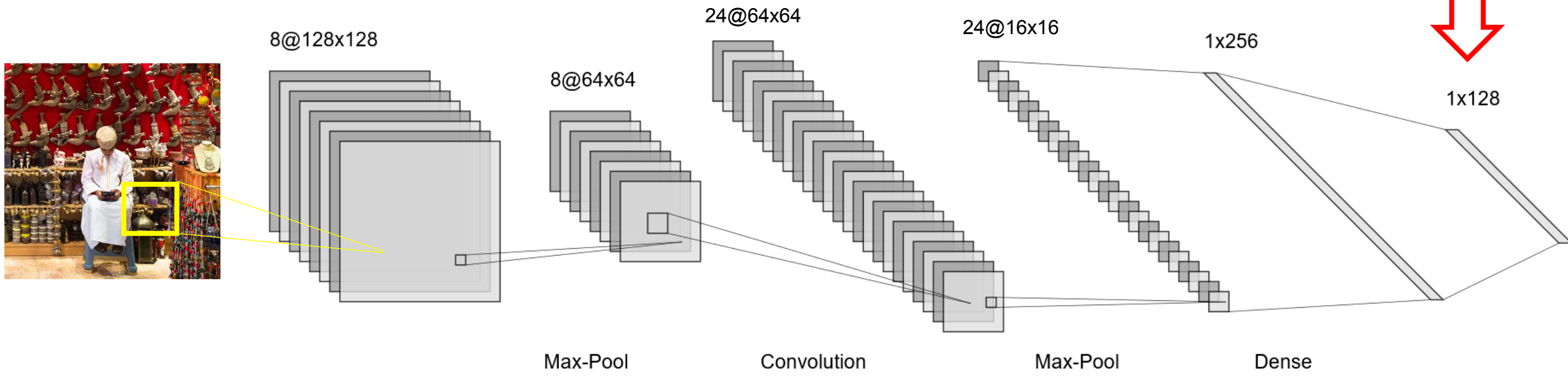


Flattening is needed to obtain a vector to be fed to the Dense layers in charge of classification.

Flattening is a good alternative to perform pooling till the spatial dimension gets reduced to 1

Convolutional Neural Networks (CNN)

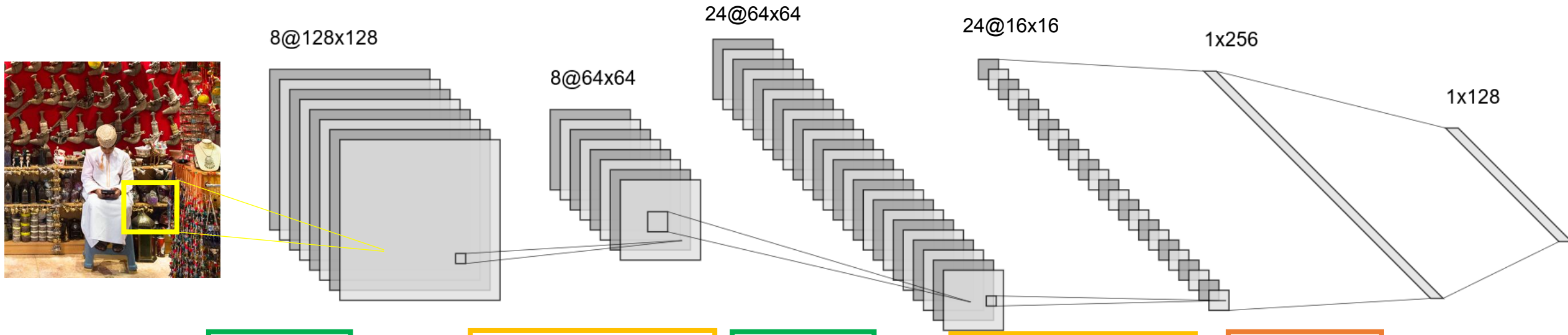
Let's analyze our network in detail



The output of the **fully connected (FC) layer** has the same size as the **number of classes**, and **provides a score** for the input image to belong to each class

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. *"Gradient-based learning applied to document recognition"* Proceedings of the IEEE, 1998 86(11), 2278-2324.

Convolutional Neural Networks (CNN)



Conv

Subsampling

Conv.

Subsampling

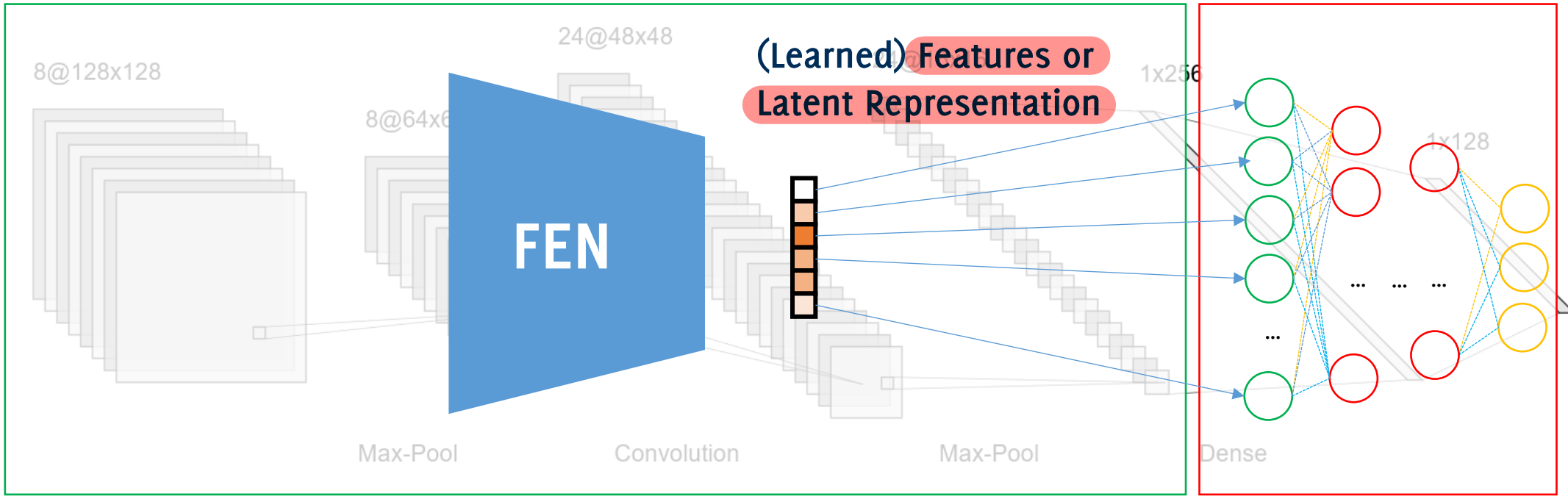
Dense

Convolution filters are learned for the classification task at hand

Thresholding + Downsampling (ReLu + Maxpooling)

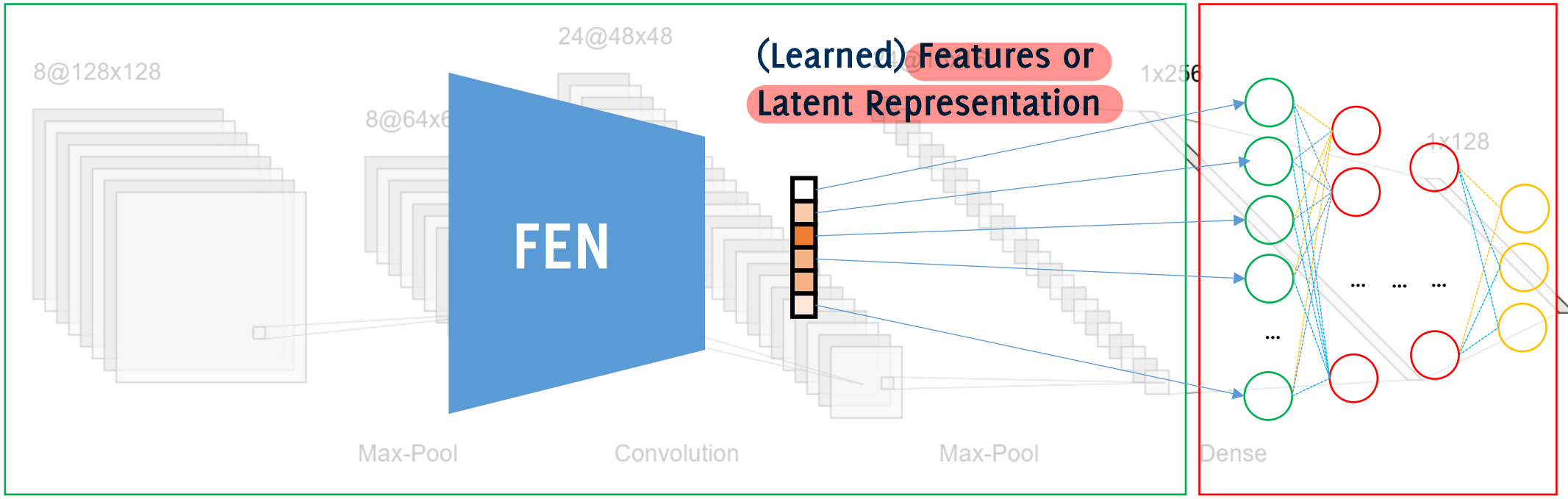
Fully Connected Neural Network providing as output class probabilities

The typical architecture of a CNN



FEN: FEATURE EXTRACTION NETWORK, the convolutional block of CNN

The typical architecture of a CNN

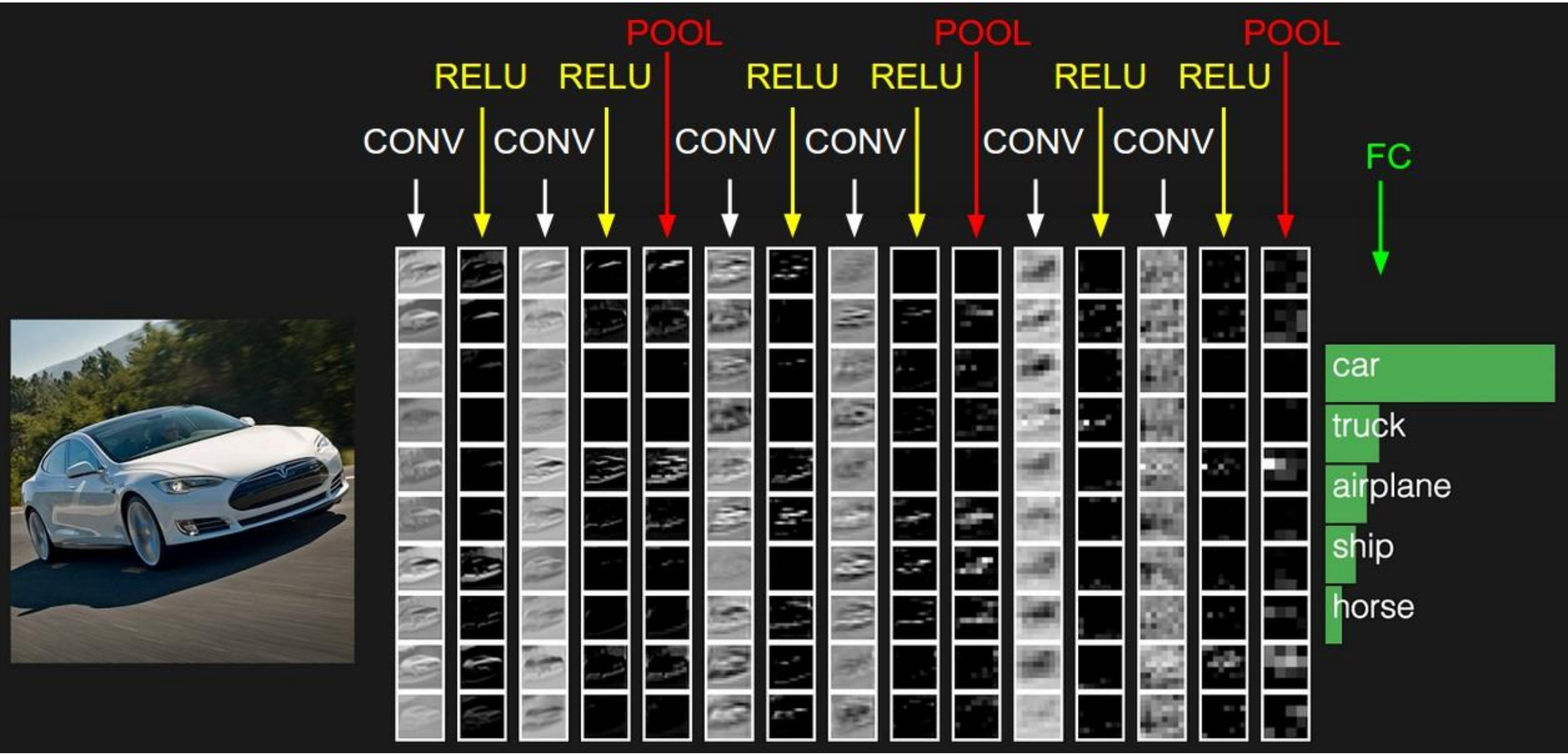


Typically, to learn meaningful representations, many layers are required

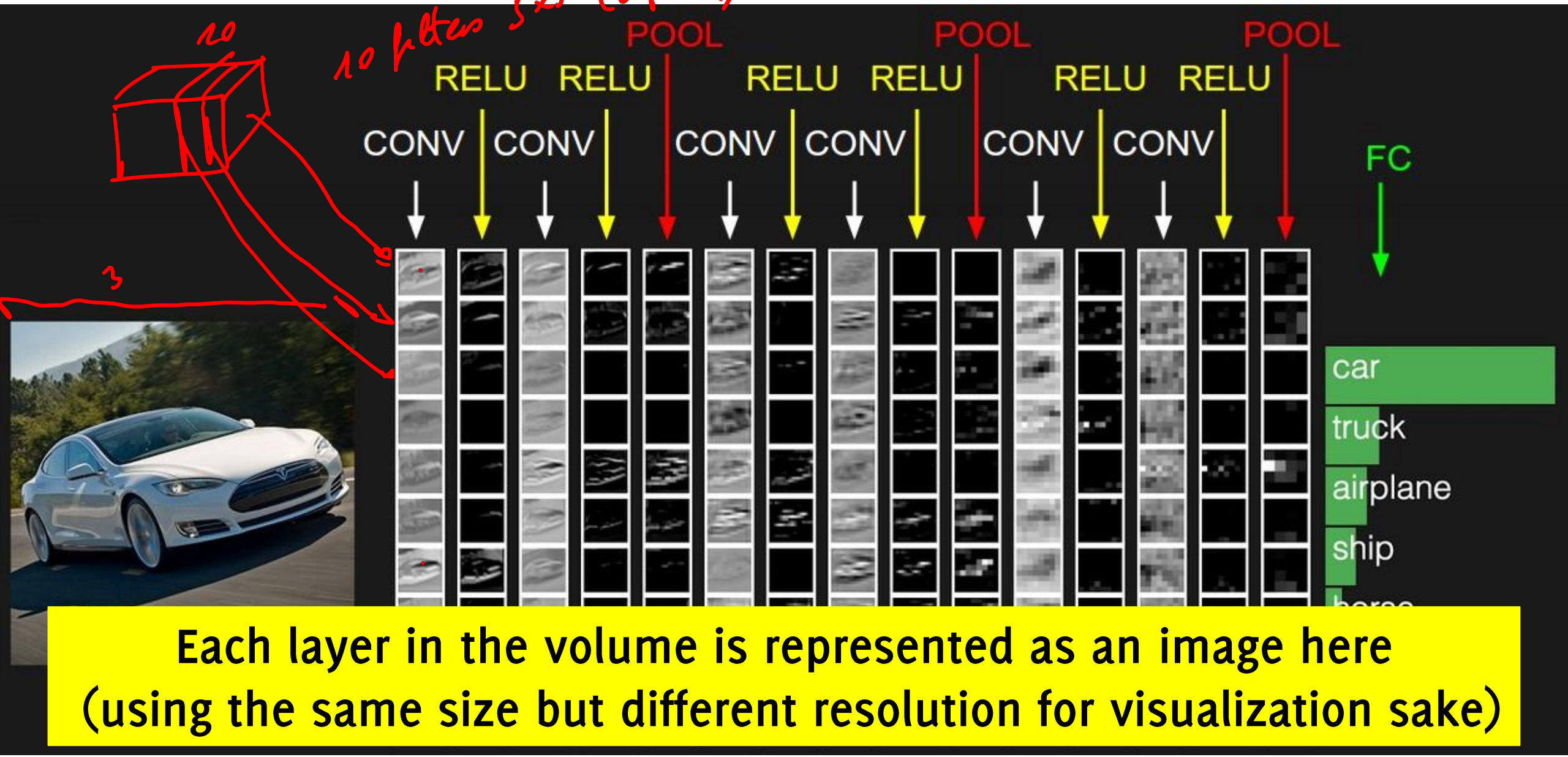
The network becomes deep

CNN «in action»

Activations in a convolutional network

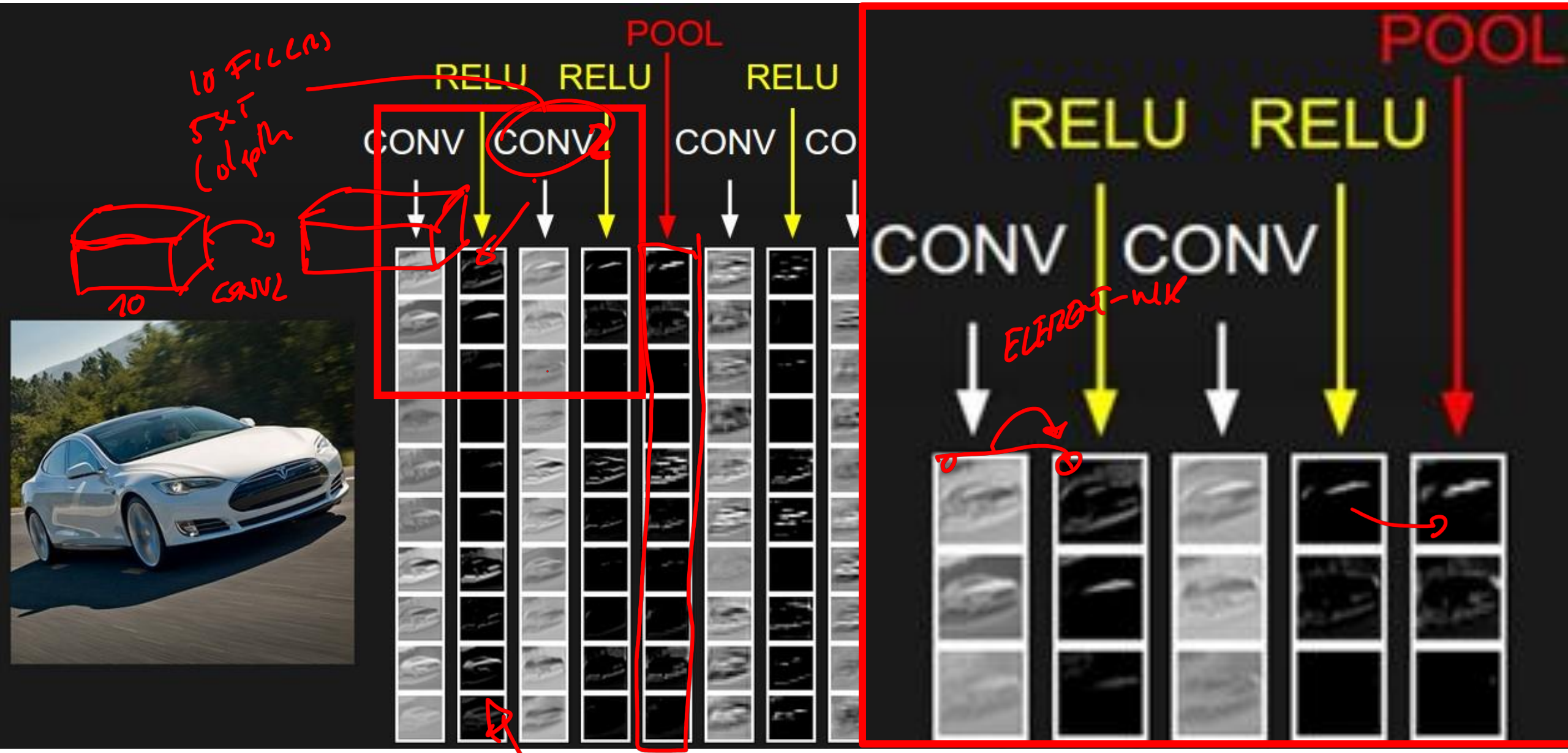


Activations in a convolutional network

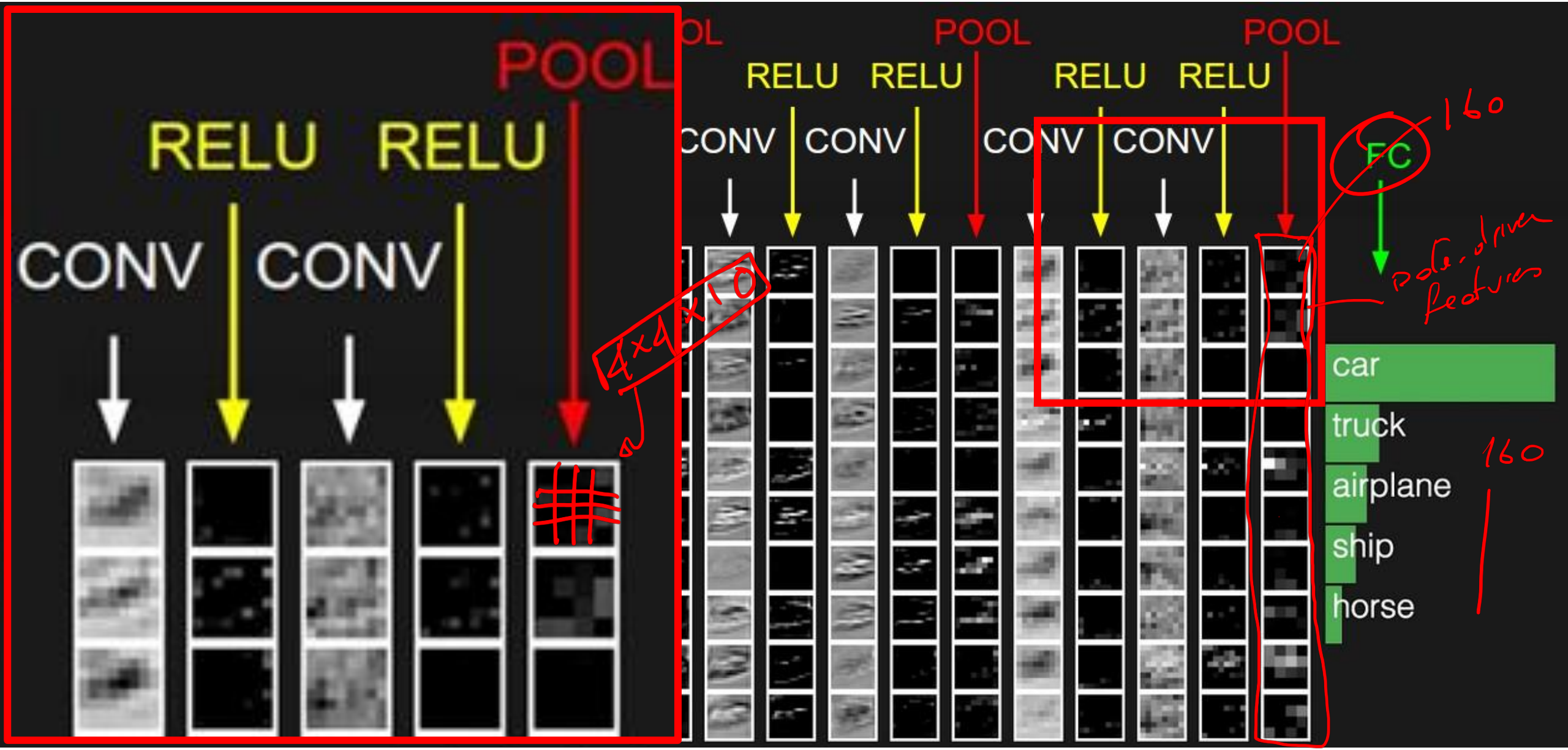


Each layer in the volume is represented as an image here (using the same size but different resolution for visualization sake)

Activations in a convolutional network

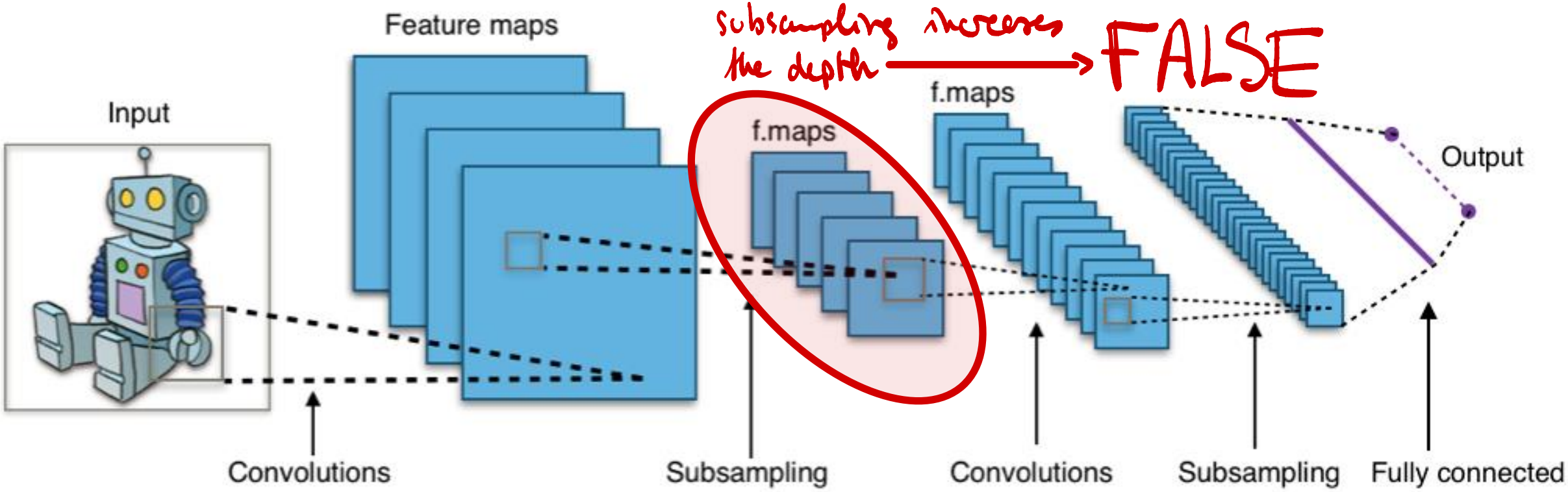


Activations in a convolutional network



Convolutional Neural Networks (CNN)

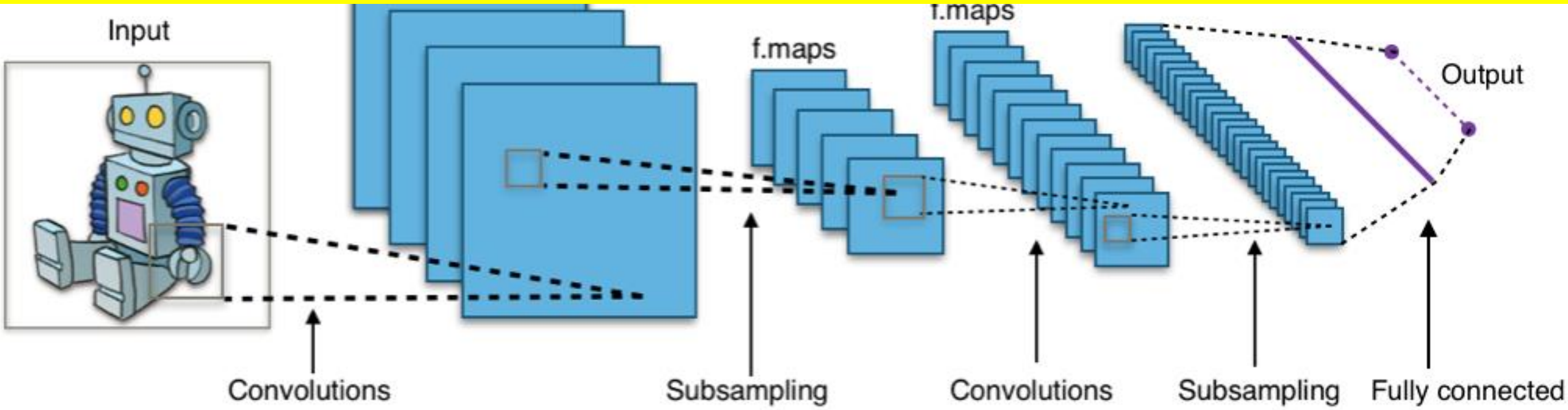
Btw, this figure contains an error.
If you are CNN-Pro, you should spot it!



By Aphex34 - Own work, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=45679374

Convolutional Neural Networks (CNN)

Btw, this figure contains an error.
If you are CNN-Pro, you should spot it!
However, that could be in principle addressable with strided convolution,…
but I don't think the designer was aware of that



By Aphex34 - Own work, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=45679374

The First CNN

Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

Abstract—
Multilayer Neural Networks trained with the backpropagation algorithm constitute the best example of a successful Gradient-Based Learning technique. Given an appropriate network architecture, Gradient-Based Learning algorithms can be used to synthesize a complex decision surface that can classify high-dimensional patterns such as handwritten characters, with minimal preprocessing. This paper reviews various methods applied to handwritten character recognition and compares them on a standard handwritten digit recognition task. Convolutional Neural Networks, that are specifically designed to deal with the variability of 2D shapes, are shown to outperform all other techniques.

I. Introduction

Over the last several years, machine learning techniques, particularly when applied to neural networks, have played an increasingly important role in the design of pattern recognition systems. In fact, it could be argued that the availability of learning techniques has been a crucial factor in the recent success of pattern recognition applications such as continuous speech recognition and handwriting recognition.

LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998)

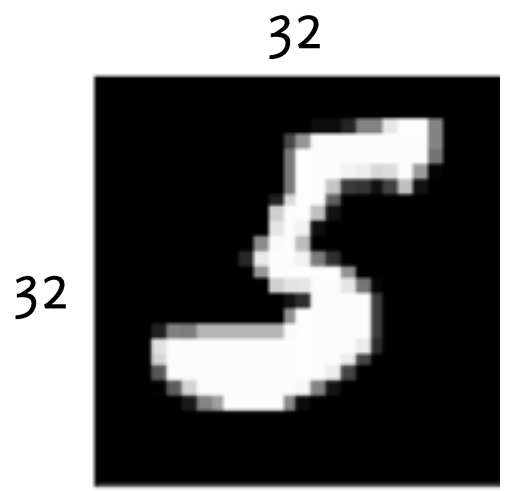
Fathers of the Deep Learning Revolution Receive ACM A.M. Turing Award

Bengio, Hinton and LeCun Ushered in Major Breakthroughs in Artificial Intelligence

https://awards.acm.org/about/2018-turing

Handwritten digit recognition

- Very small grayscale images.
- The task is to classify each image in the corresponding digit
- This can be used to classify numbers as sequences of digits



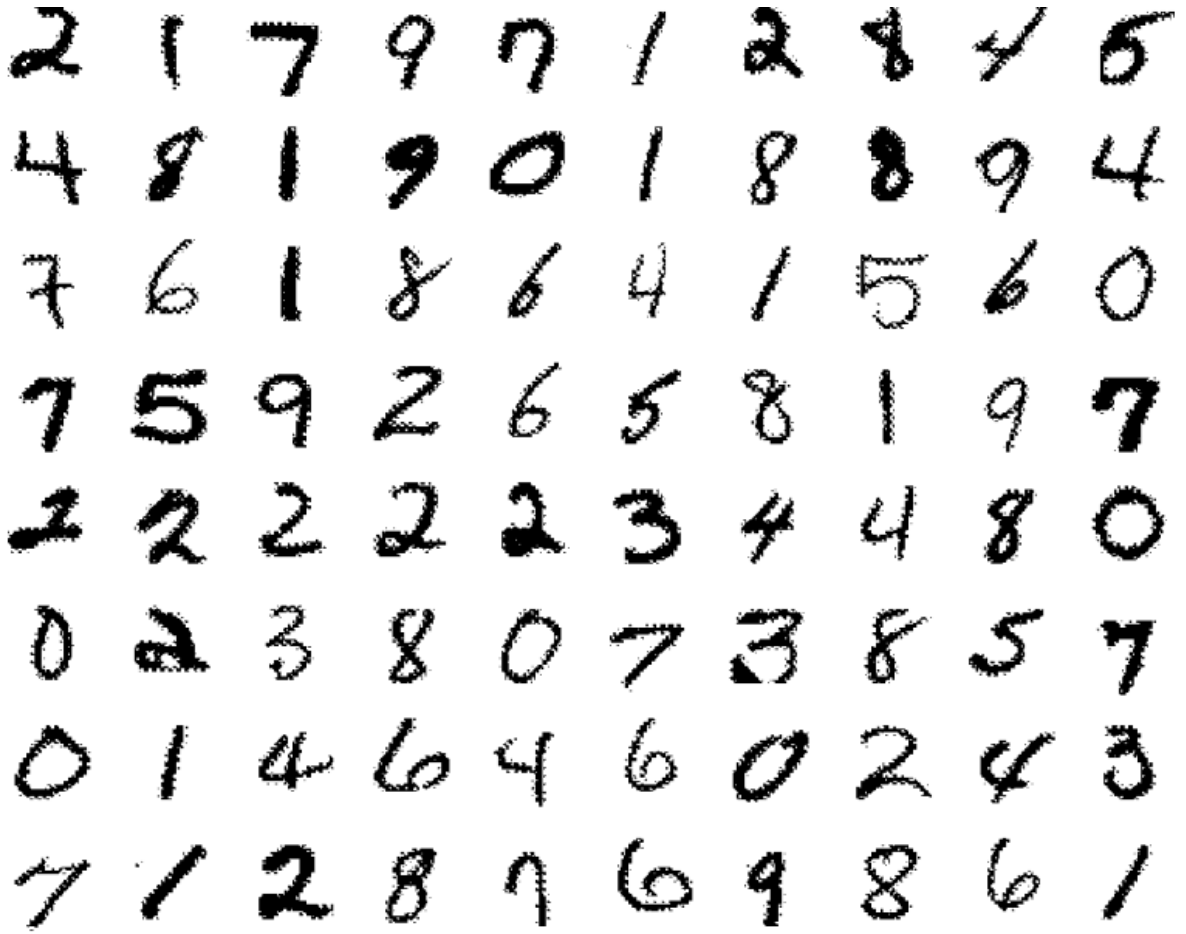
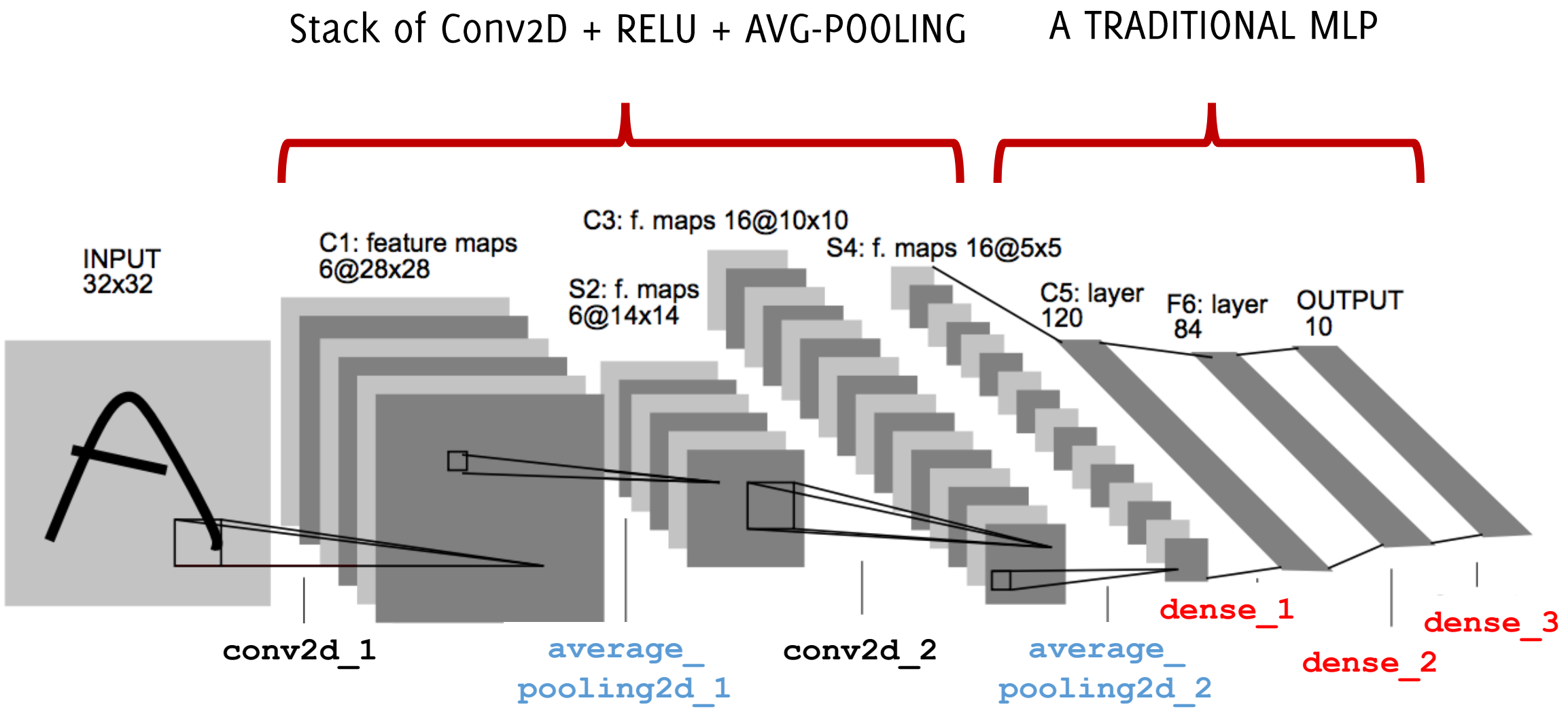


Fig. 4. Size-normalized examples from the MNIST database.

LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998)

LeNet-5 (1998)



LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998)

The First CNN

Do not use each pixel as a separate input of a large MLP, because:

- images are highly spatially correlated,
- using individual pixel of the image as separate input features would not take advantage of these correlations.

The first convolutional layer: 6 filters 5x5

The second convolutional layer: 16 filters 5x5

LeNet-5 in Keras

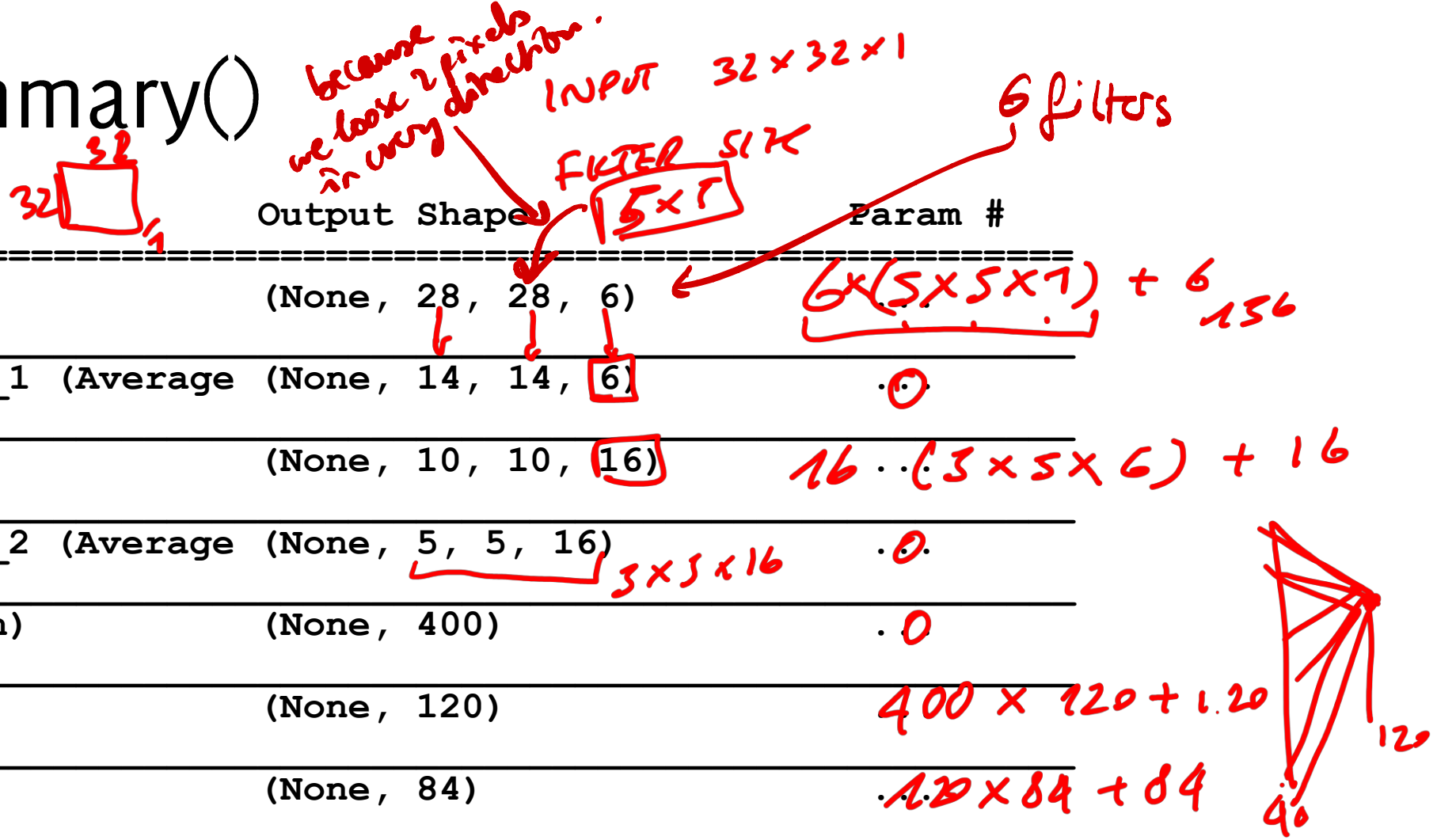
```
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, AveragePooling2D

num_classes = 10;
input_shape=(32, 32, 1);

model = Sequential()
model.add(Conv2D(filters = 6, kernel_size = (5, 5), activation='tanh', input_shape=inp
ut_shape, padding = 'valid'))
model.add(AveragePooling2D(pool_size=(2, 2)))
model.add(Conv2D(filters = 16, kernel_size = (5, 5), activation='tanh',
padding = 'valid'))
model.add(AveragePooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(120, activation='relu'))
model.add(Dense(84, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
```

model.summary()

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 6)         ...
_________________________________________________________________
average_pooling2d_1 (Average (None, 14, 14, 6)         ...
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 16)        ...
_________________________________________________________________
average_pooling2d_2 (Average (None, 5, 5, 16)          ...
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               ...
_________________________________________________________________
dense_1 (Dense)              (None, 120)               ...
_________________________________________________________________
dense_2 (Dense)              (None, 84)                ...
_________________________________________________________________
dense_3 (Dense)              (None, 10)                ...
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```

model.summary()

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d_1 (Average (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_2 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_1 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_2 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_3 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```



model.summary()

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 6)         156 (6 x 5 x 5 + 6)
_________________________________________________________________
average_pooling2d_1 (Average (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 16)        2416 (16 x 5 x 5 x 6 + 16)
_________________________________________________________________
average_pooling2d_2 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_1 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_2 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_3 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```

(400+1) x 120

Input is a grayscale image

The input is a volume having depth = 6

Most parameters are still in the MLP

model.summary()

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 6)         156 (6 x 5 x 5 + 6)
_________________________________________________________________
average_pooling2d_1 (Average (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 16)        2416 (16 x 5 x 5 x 6 + 16)
_________________________________________________________________
average_pooling2d_2 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_1 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_2 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_3 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
```

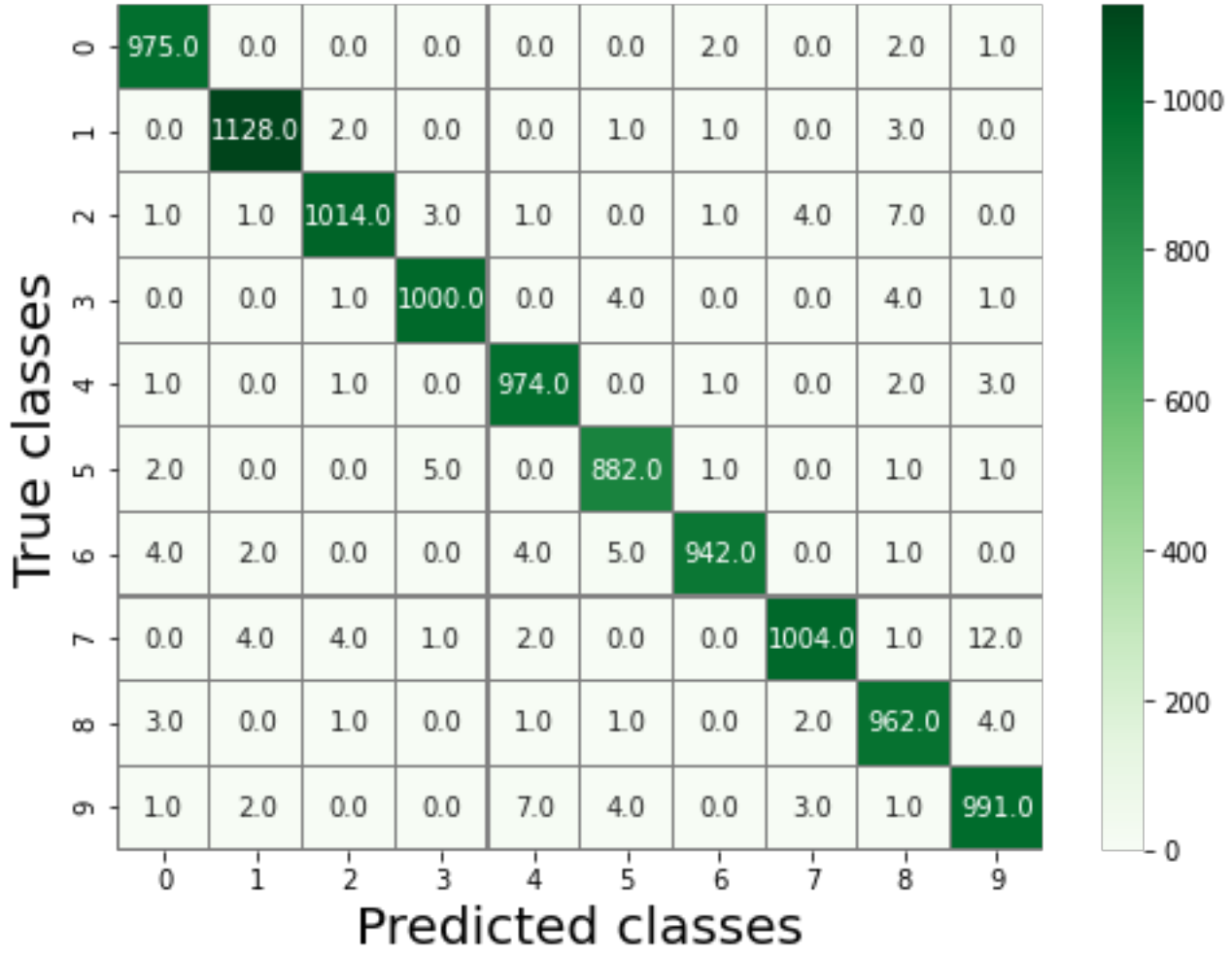
Input is a grayscale image

The input is a volume having depth = 6

Most parameters are still in the MLP

Here, no-padding at the first layer is necessary to reduce the size of the latent representation… and has no loss of information since images are black there!

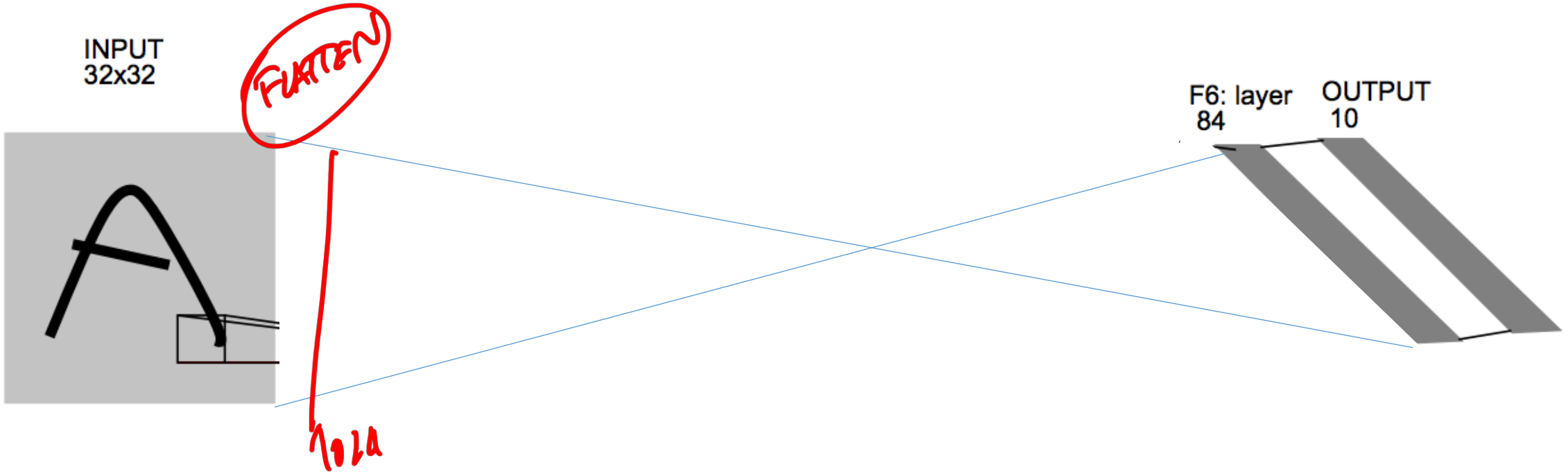
LeNet-5 performance



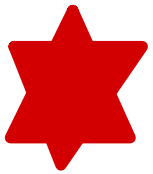
Most of parameters are in MLP

What about a MLP taking as input the whole image?

Input 32 x 32 = 1024 pixels, fed to a 84 neurons (the last FC layers of the network) -> 86950 parameters: 1024 * 84 + 84 + 84 * 10 + 10



Most of parameters are in MLP



What about a MLP taking as input the whole image?

Input 32 x 32 = 1024 pixels, fed to a 84 neurons (the last FC layers of the network) -> 86950 parameters

But.. If you take an RGB input: 32 x 32 x 3,

CNN: only the nr. of parameters in the filters at the first layer increases

$$156 + 61550 \rightarrow 456 + 61550$$
$$(6 \times 5 \times 5) \rightarrow (6 \times 5 \times 5 \times 3)$$

MLP: only the first layer increases the # of parameters by a factor 3

$$1024 \times 84 \rightarrow 1024 \times 84 \times 3$$

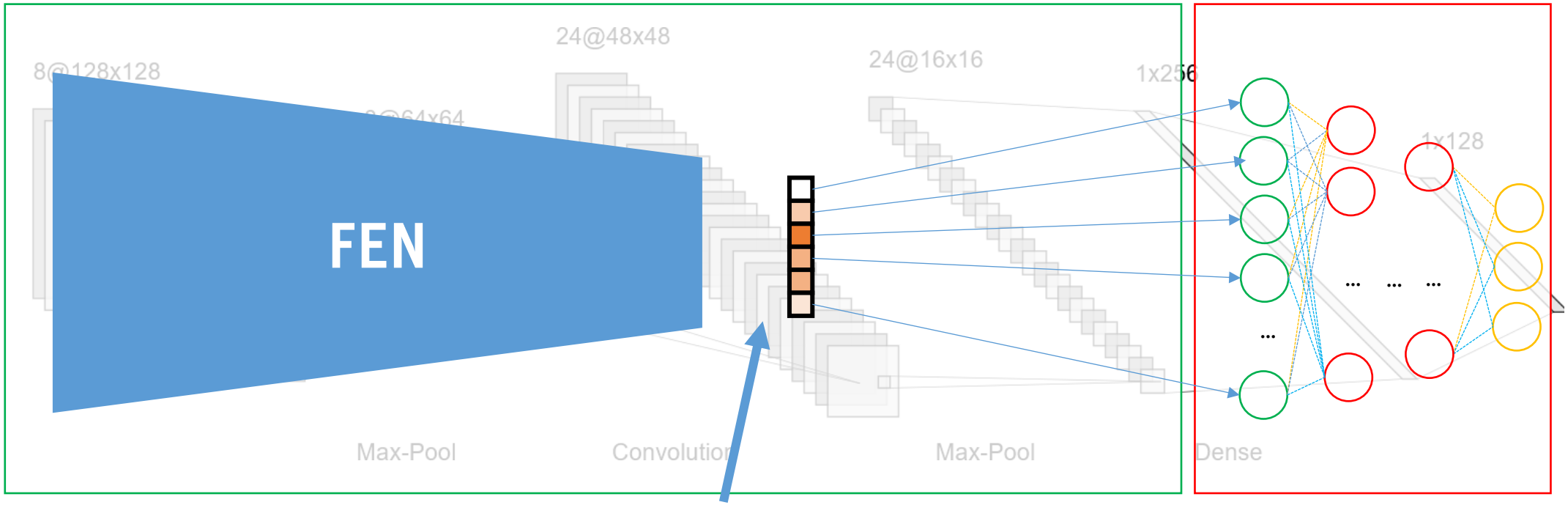
Latent representation in CNNs

Repeat the «t-SNE experiment» on the CIFAR dataset, using the last layer of the CNN as vectors

The typical architecture of a CNN

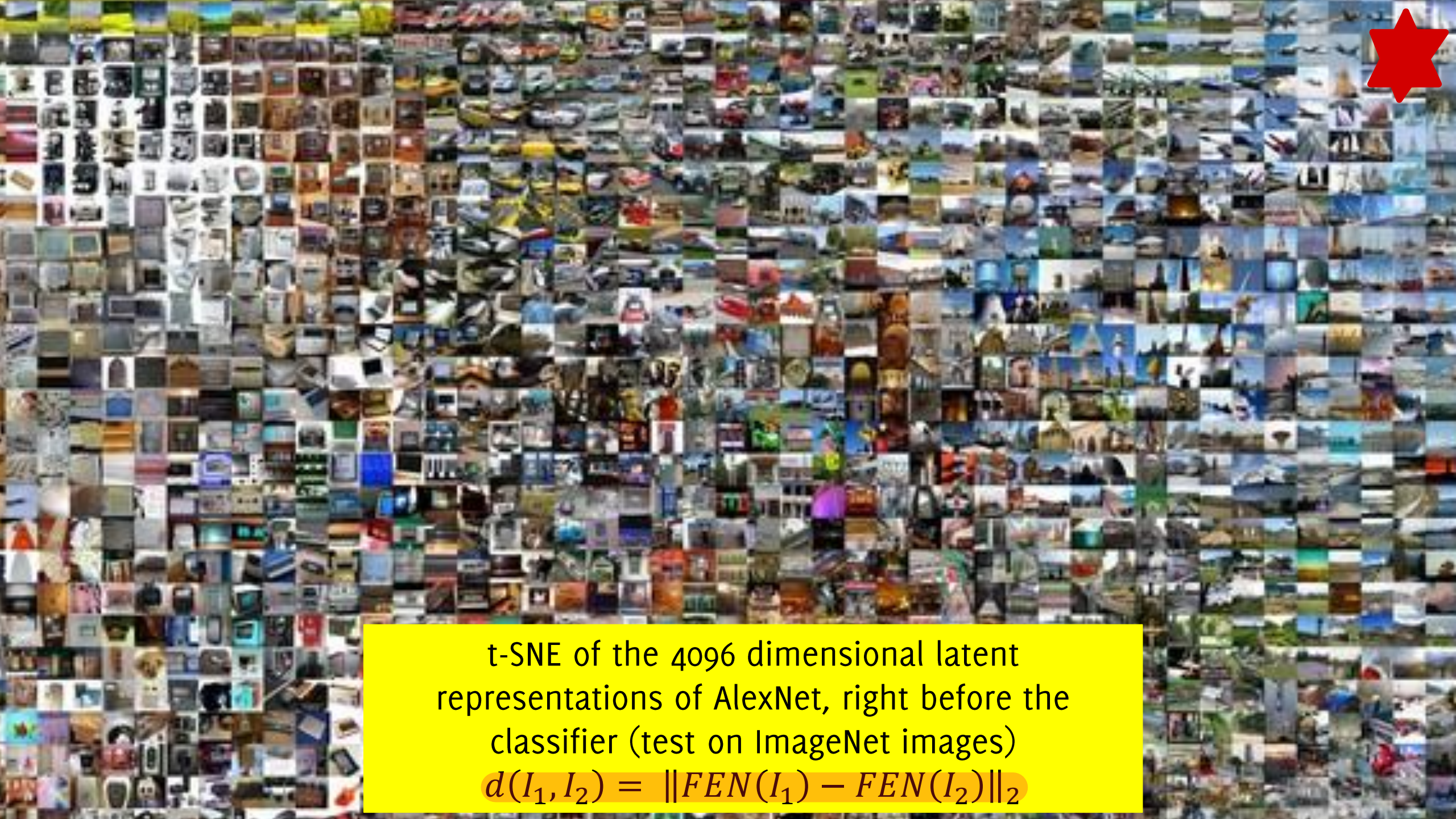
Convolutional Layers
Extract high-level features from pixels

Classify



Latent Representation:
Data-Driven Feature Vector

MLP for feature classification



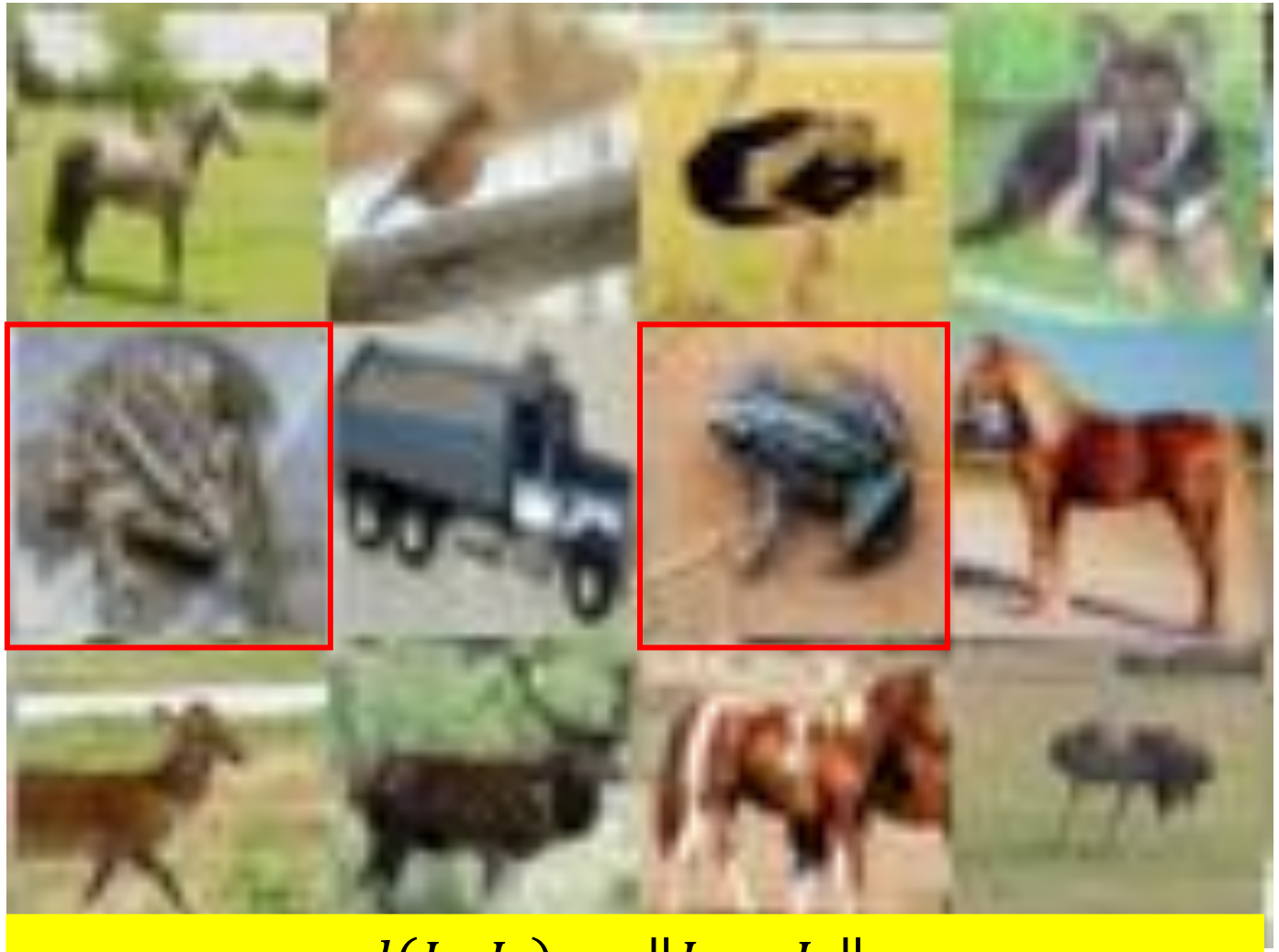
t-SNE of the 4096 dimensional latent representations of AlexNet, right before the classifier (test on ImageNet images)

$$d(I_1, I_2) = \|FEN(I_1) - FEN(I_2)\|_2$$

It seems really nice.

Distances in the latent representation of a CNN are much more meaningful than data itself

t-SNE representation using ℓ_2 distance



$$d(I_1, I_2) = \|I_1 - I_2\|_2$$

CNNs in Keras

Giacomo Boracchi

What is Keras?

An open-source library providing **high-level building blocks** for developing deep-learning models in Python

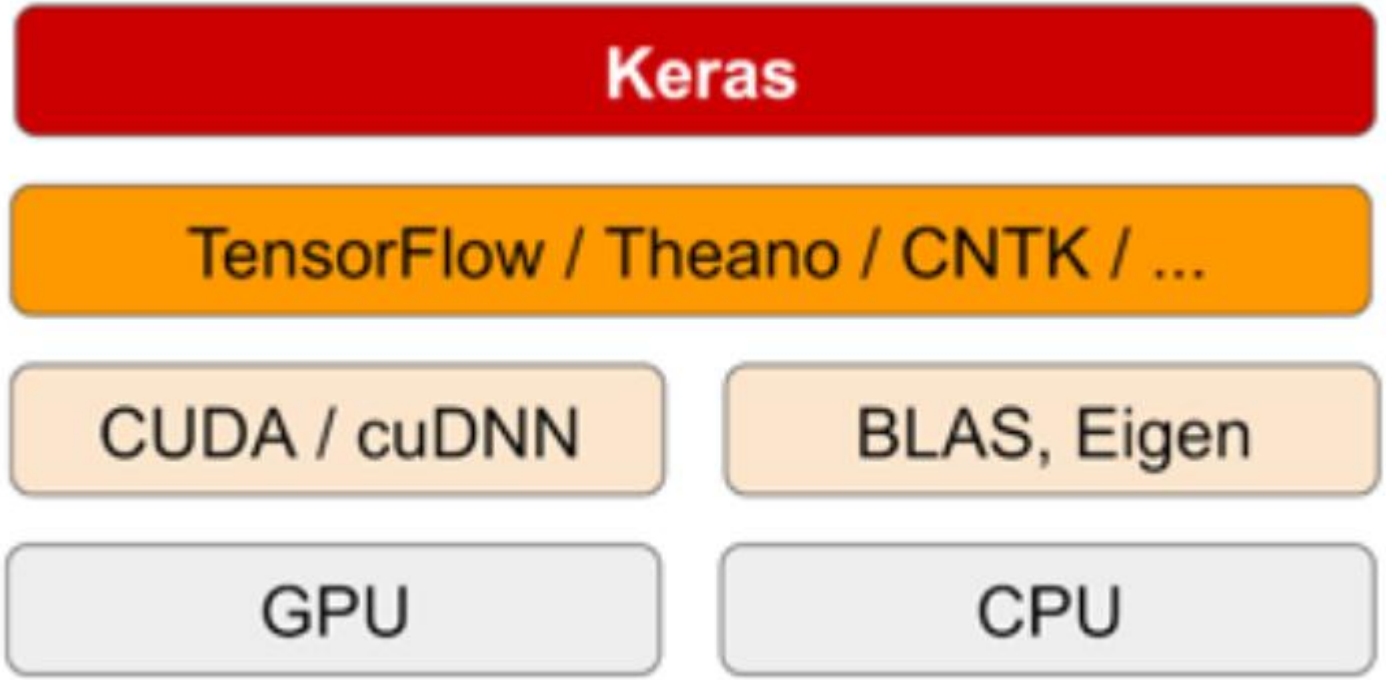
Designed to enable **fast experimentation with deep neural networks**, it focuses on being **user-friendly, modular, and extensible**

Doesn't handle low-level operations such as tensor manipulation and differentiation.

Relies on **backends** (TensorFlow, Microsoft Cognitive Toolkit, Theano, or PlaidML)

Enables **full access to the backend**

The software stack



Why Keras?

Pros:

Higher level → fewer lines of code

Modular backend → not tied to tensorflow

Way to go if you focus on applications

Cons:

Not as flexible

Need more flexibility? Access the backend directly!

We will manipulate 4D tensors

Images are represented in 4D tensors:

Tensorflow convention: (samples, height, width, channels)

A detailed presentation on CNN in Keras will be made in the lab session