Image Analysis and Computer Vision

Giacomo Boracchi
giacomo.boracchi@polimi.it

February 16th 2024

UEM, Maputo

https://boracchi.faculty.polimi.it

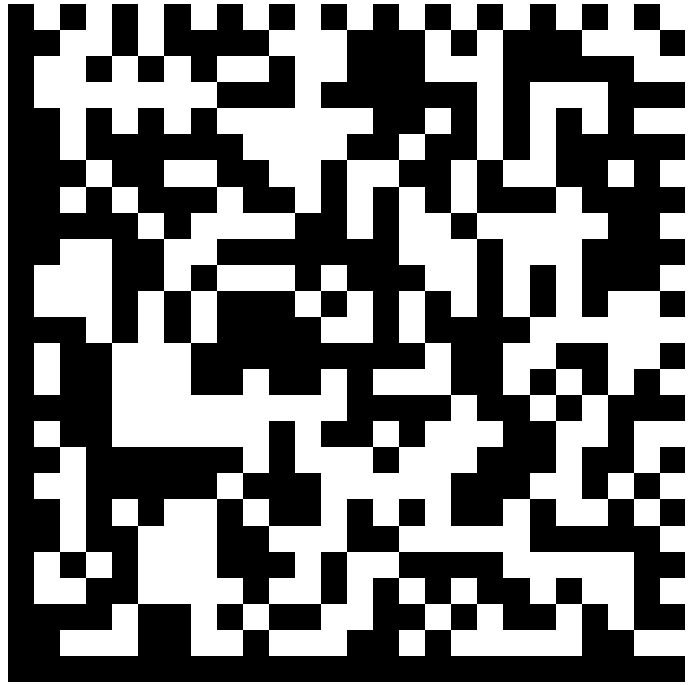
Course Slides

Slides can be found on my website

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

and follow Tutorials and Talks

https://boracchi.faculty.polimi.it/seminars.html

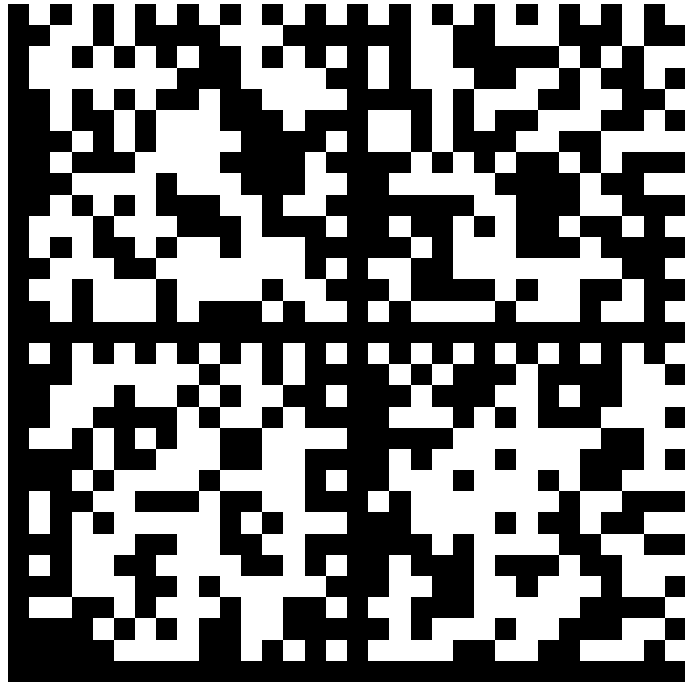


Colab Folder

In this folder you will find, regularly updated notebooks

https://drive.google.com/drive/folders/10j99rb2kK04KpLxca-uMe7uesy-8RZeD

Notebooks require you to “fill in” some codes or to extend codes we illustrate during lectures to new data/new challenges



Project Assignment

1. Implement an image classifier, based on hand crafted features for the parcel dataset (see the colab script 2023_Lez_03_handcrafted_feature_classifier_parcel.pynb in Lecture 3 folder)

2. Implement a deep neural network (CNN) alterative and train it using data augmentation / transfer learning until you get to good performance.

3. Train and test both solutions (make sure training and test set remain separated). Show inference results on selected images

4. Send me a 1 page report (where you discuss what is your contribution) and the Py notebook with the results already in.

Convolutional Neural Netwokrs

Giacomo Boracchi
giacomo.boracchi@polimi.it

February 14th 2024

UEM, Maputo

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

The Feature Extraction Perspective

Input image

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

Feature Extraction Algorithm

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

$(d \ll r_1 \times c_1)$

Classifier (NN)

"wheel"

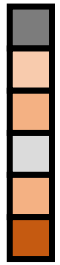
$y \in \Lambda$

The Feature Extraction Perspective

Input image

$I_2 \in \mathbb{R}^{r_2 \times c_2}$

Feature Extraction Algorithm



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

$(d \ll r_2 \times c_2)$

Classifier (NN)

"castle"

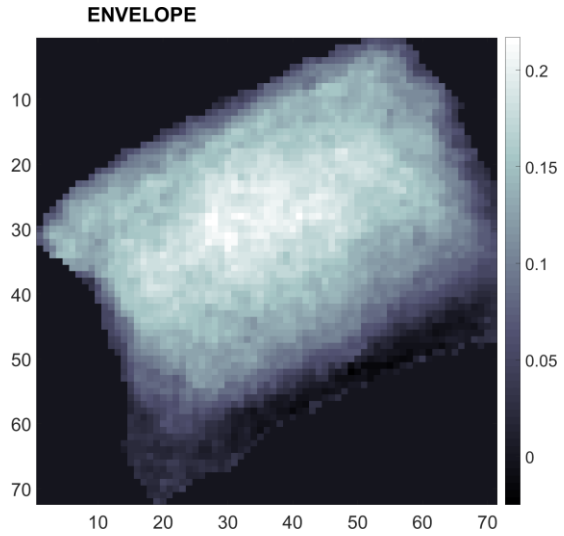
$y \in \Lambda$

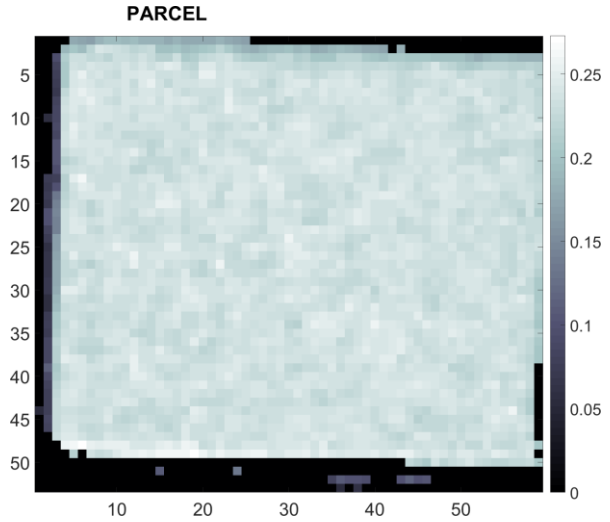
Hand-Crafted Features

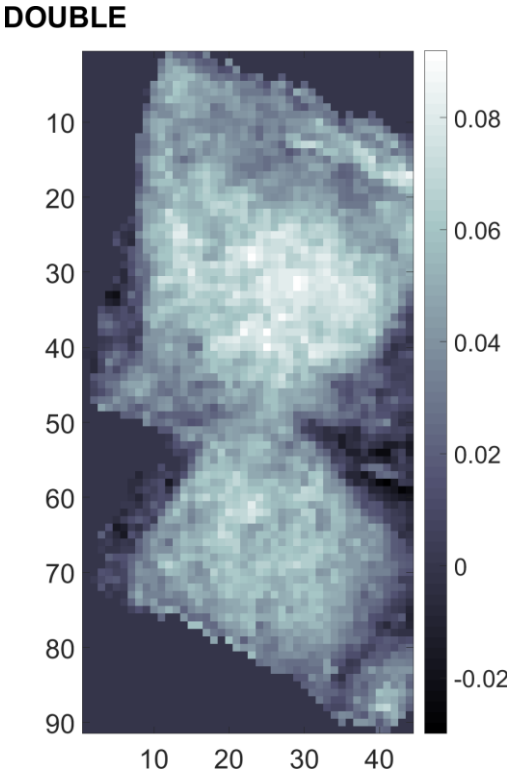
Example of Hand-Crafted Features

Example of features:

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

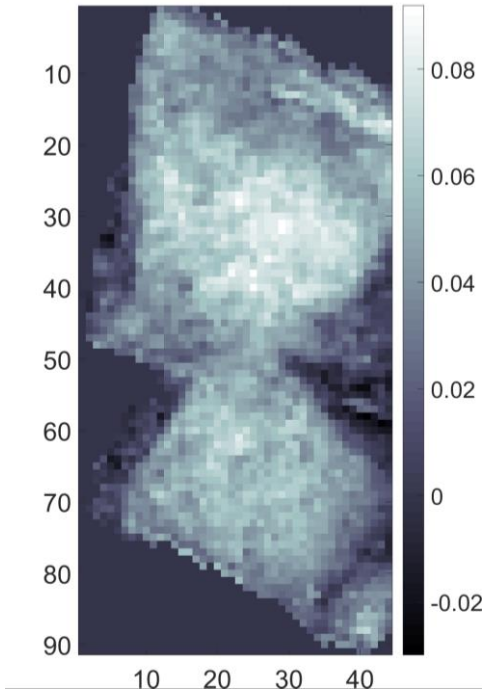






Neural Networks

Input image



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

Feature Extraction Algorithm

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

input layer

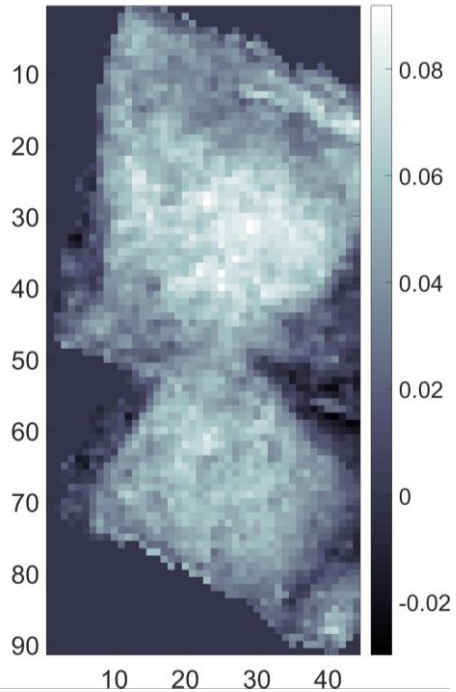
Hidden layer(s)

Output Layer

Neural Networks

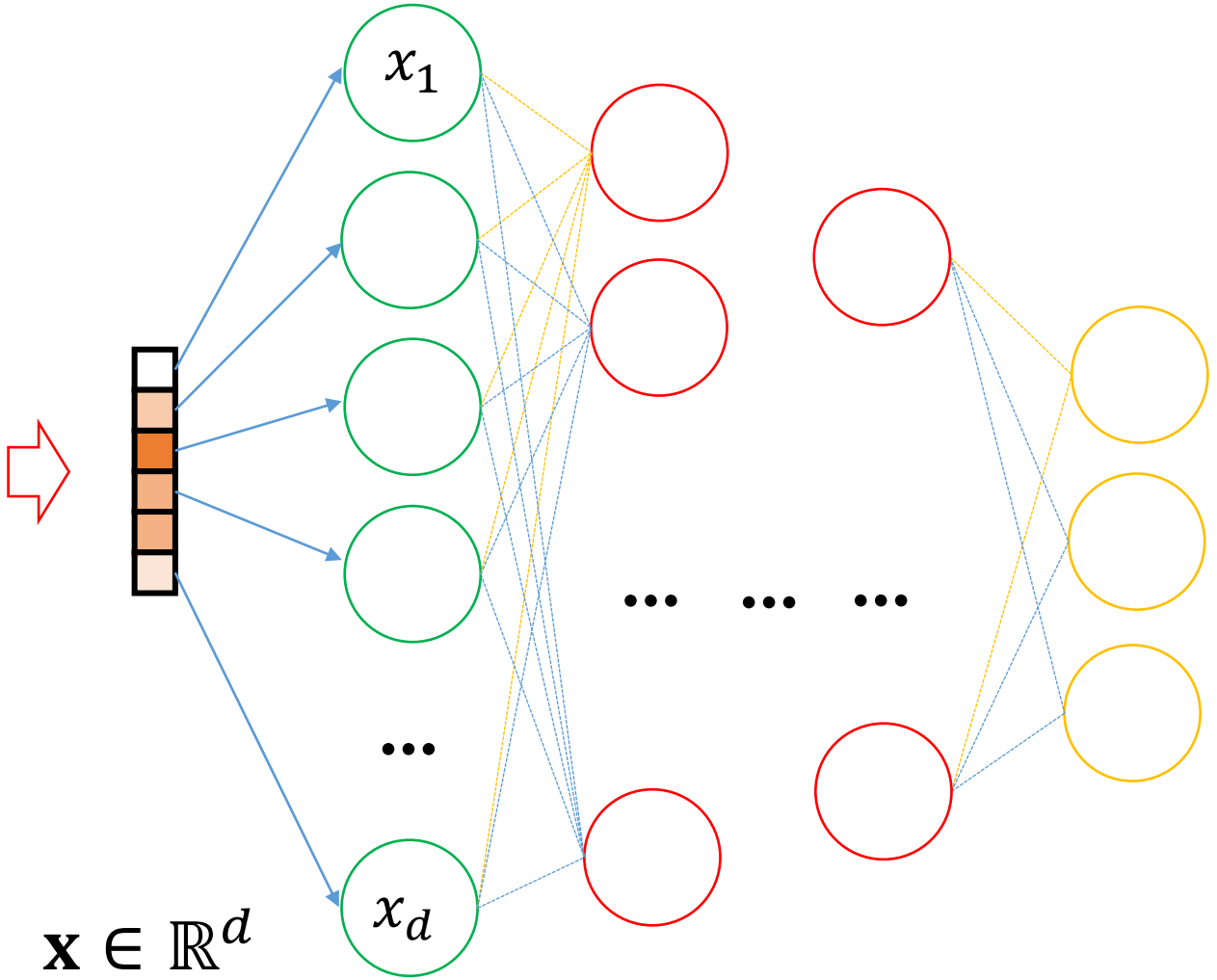
Input layer: Same size of the feature vector

Input image



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

Feature Extraction Algorithm



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

input layer

Hidden layer(s)

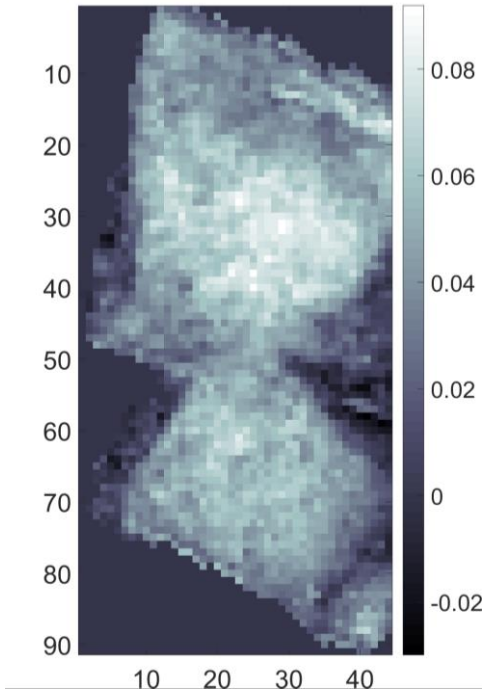
Output Layer

Giacomo Boracchi

Neural Networks

Output layer: Same size as the number of classes

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

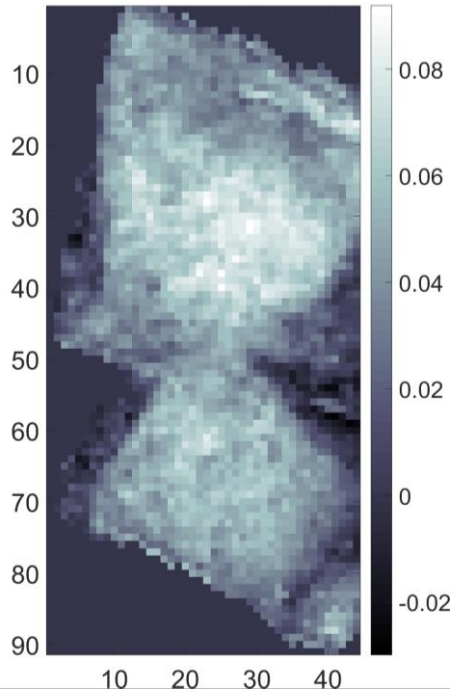
Hidden layer(s)

Output Layer

Neural Networks

Hidden layers: arbitrary size

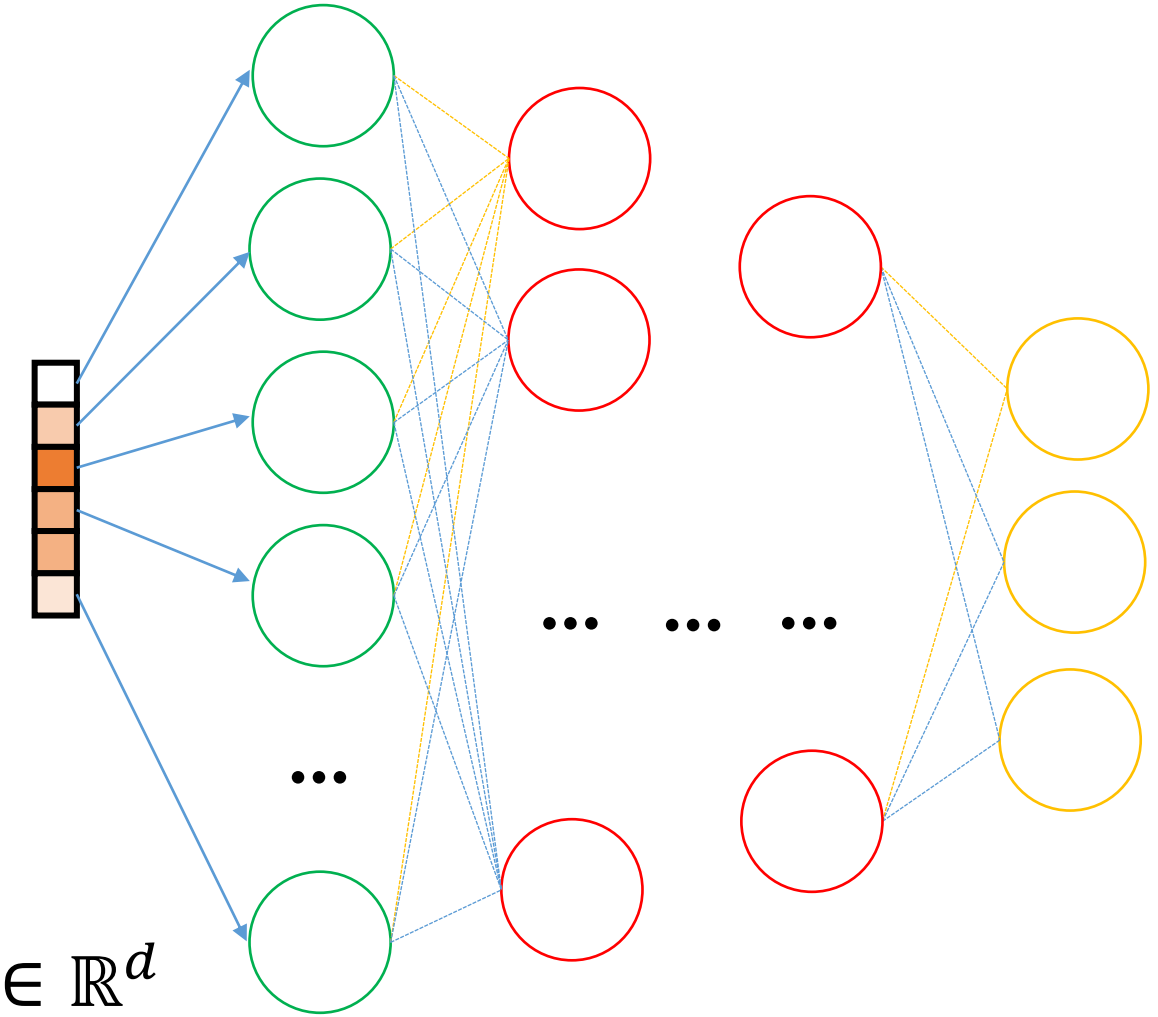
Input image



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

Feature Extraction Algorithm

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



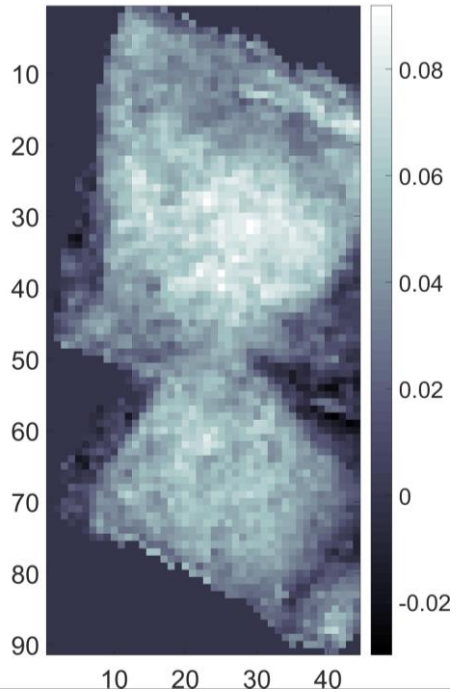
input layer

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

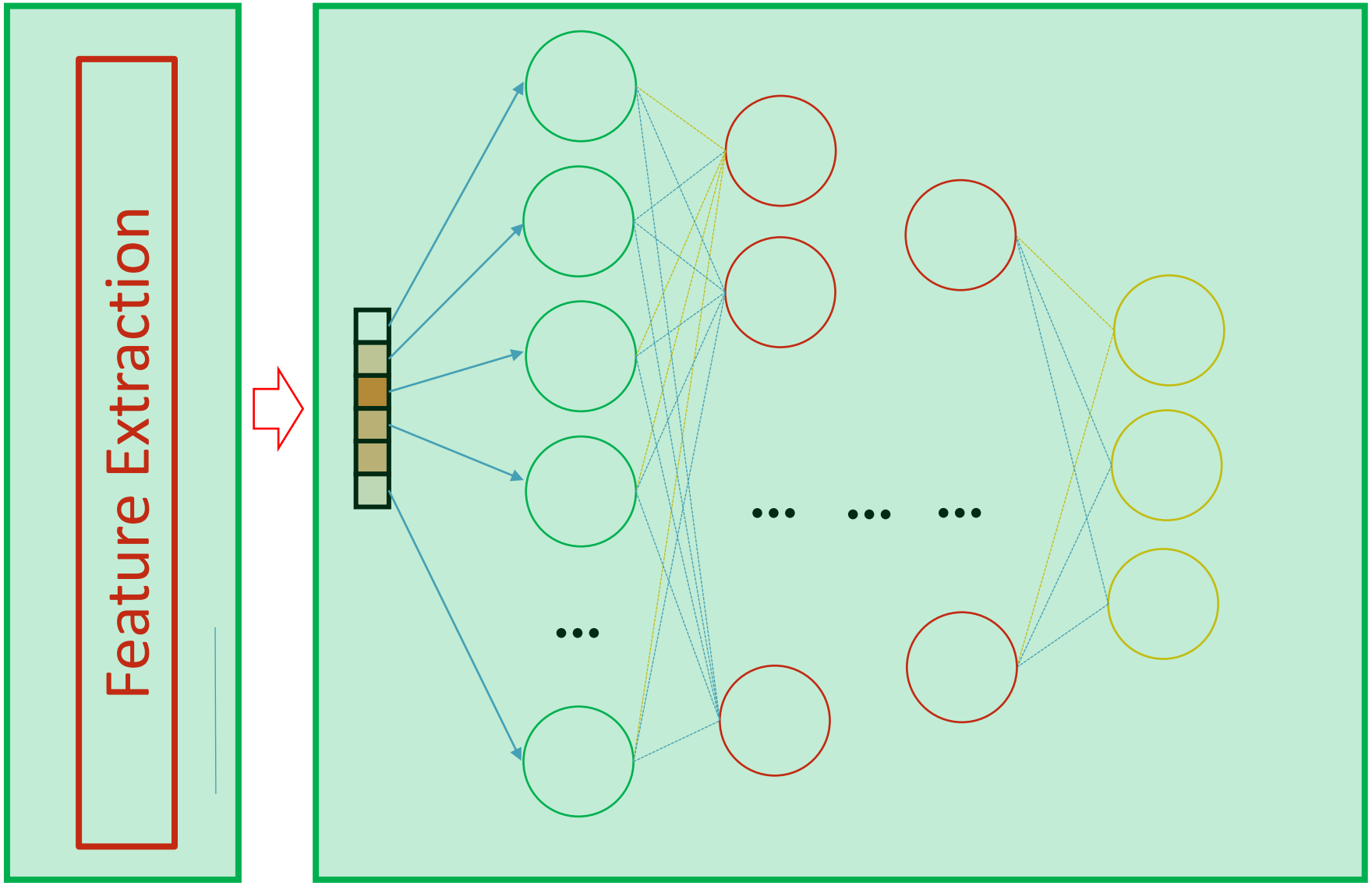
… the advent of deep learning

Giacomo Boracchi

Data-Driven Features

Input image

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



Convolutional Neural Networks

Setting up the stage

Giacomo Boracchi

Local Linear Filters

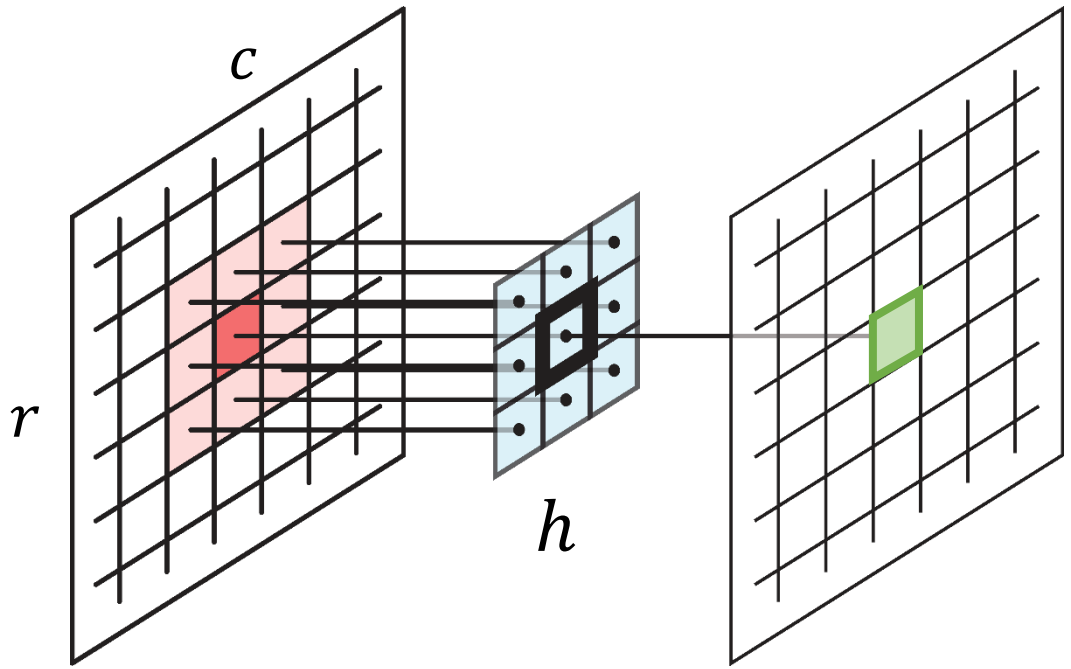
Linear Transformation: Linearity implies that the output $T[I](r,c)$ is a linear combination of the pixels in U:

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

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

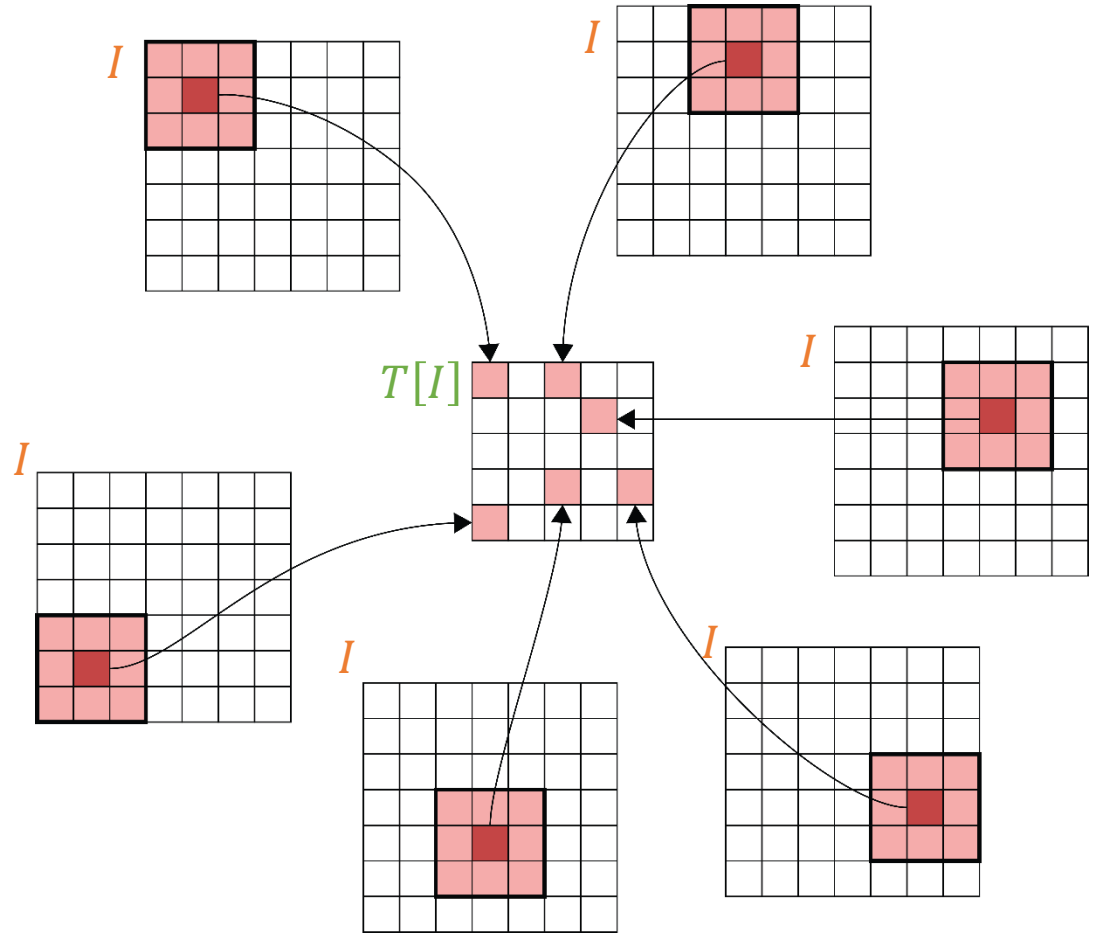
Linear Transformation: the filter weights can be assoicated to a matrix $\boldsymbol{w}$

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

$\boldsymbol{w}$

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

This operation is repeated for each pixel in the input image



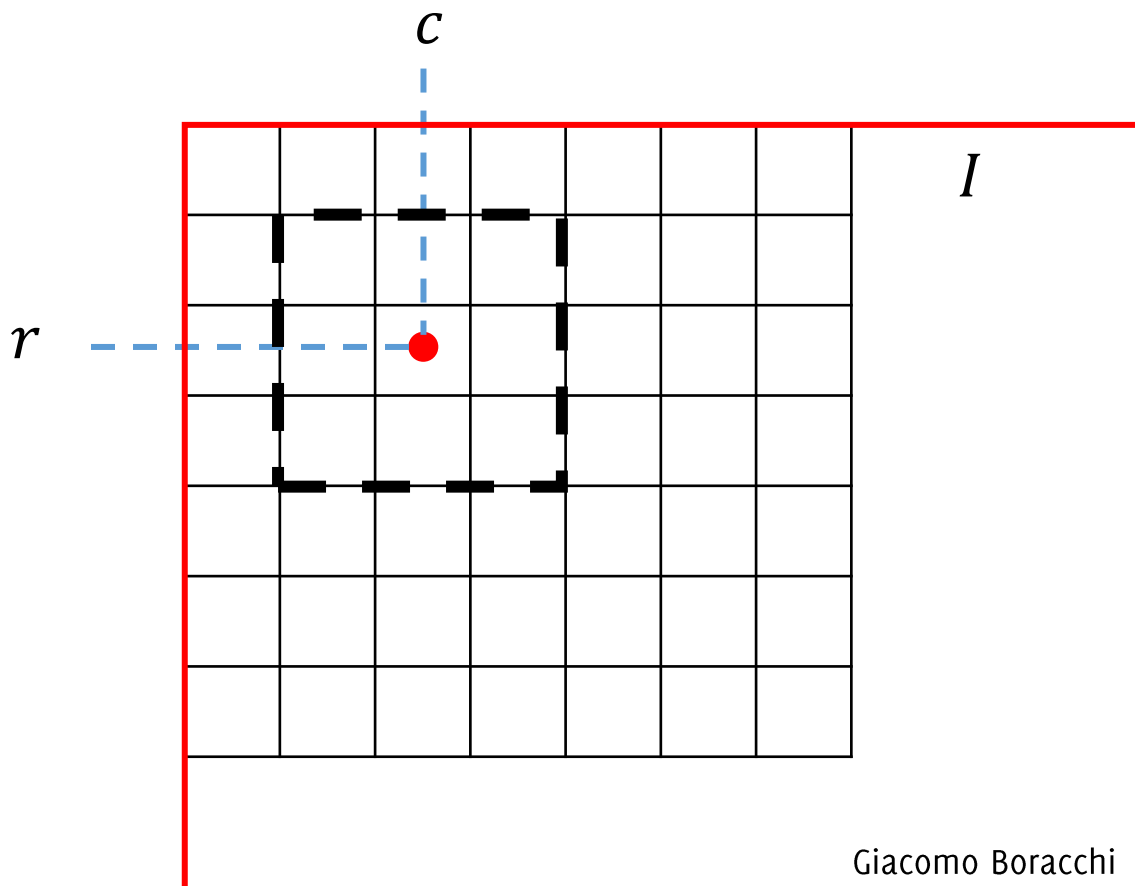
2D Correlation

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

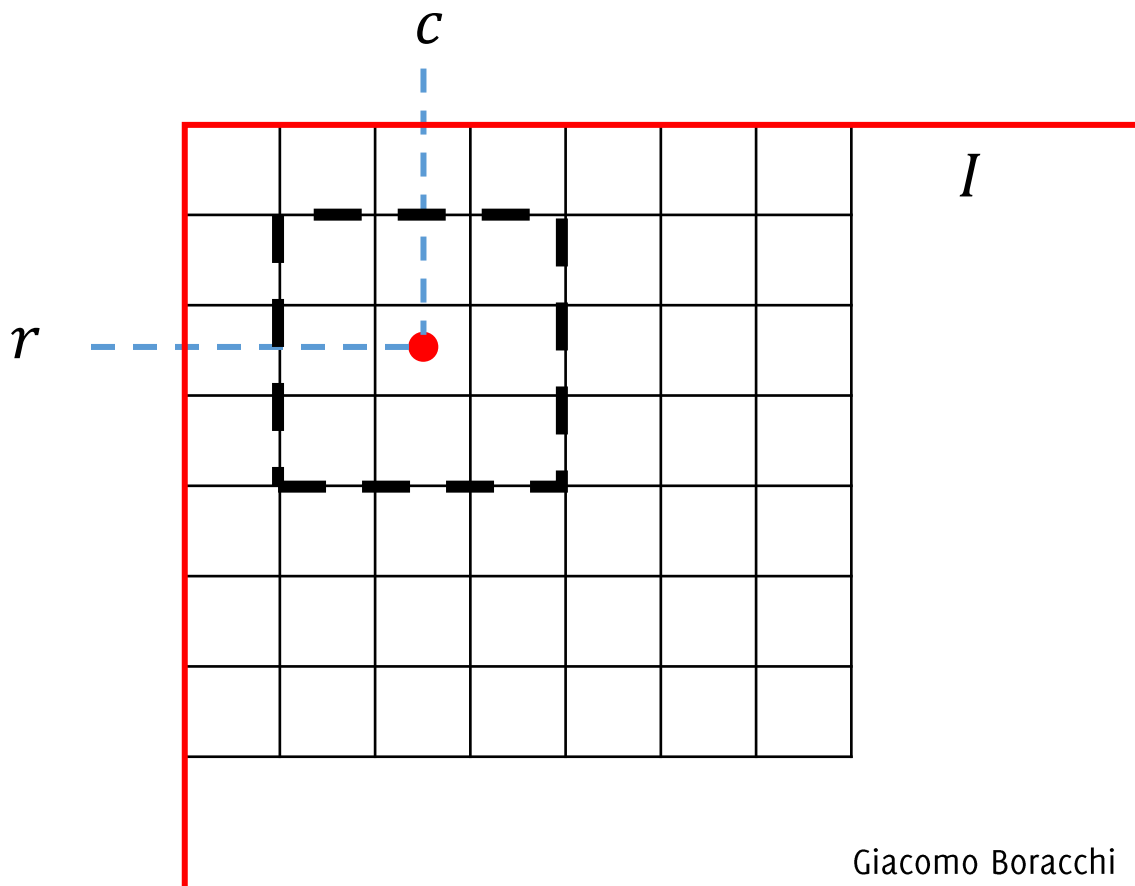
Convolution is a linear transformation. Linearity implies that

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

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

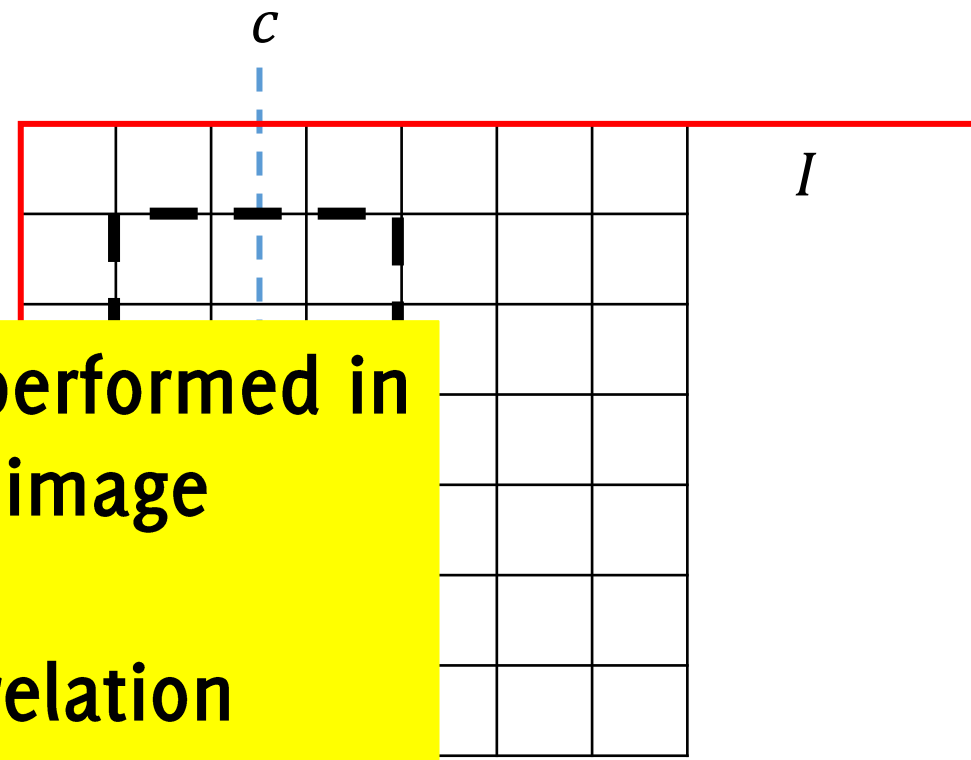
Convolution is a linear transformation. Linearity implies that

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

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



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

We can con

The filter h e

Convolution op

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.

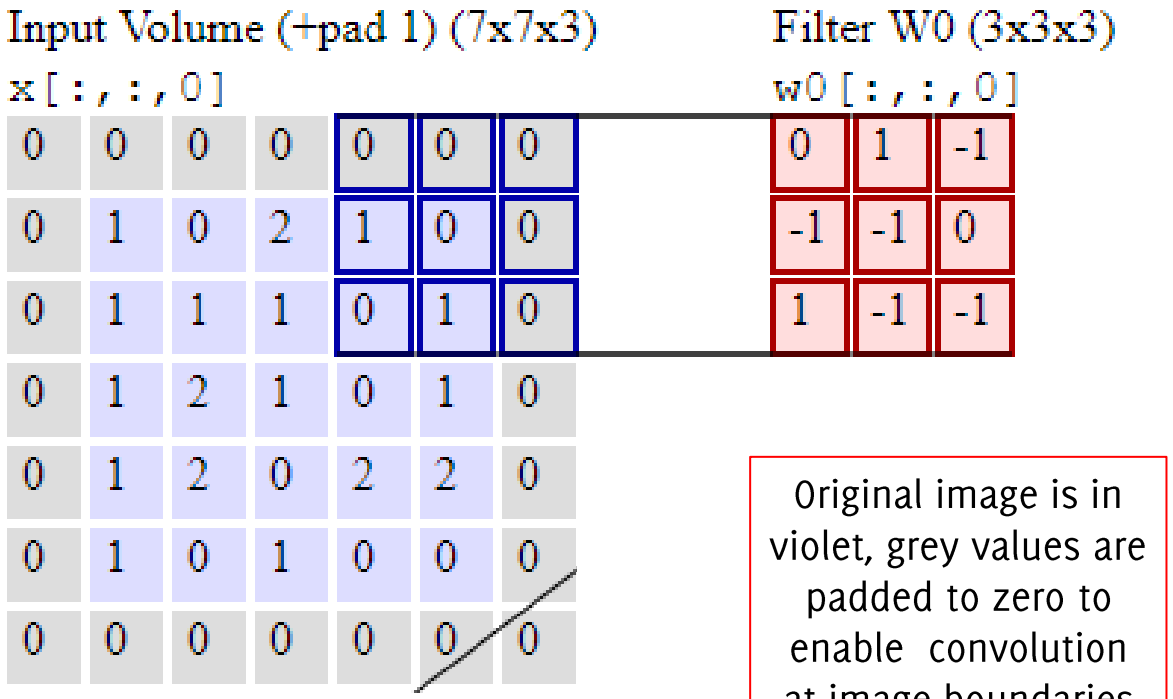
However, in CNN, convolutional filters are being learned from data, thus it is only important to use these in a consistent way.

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

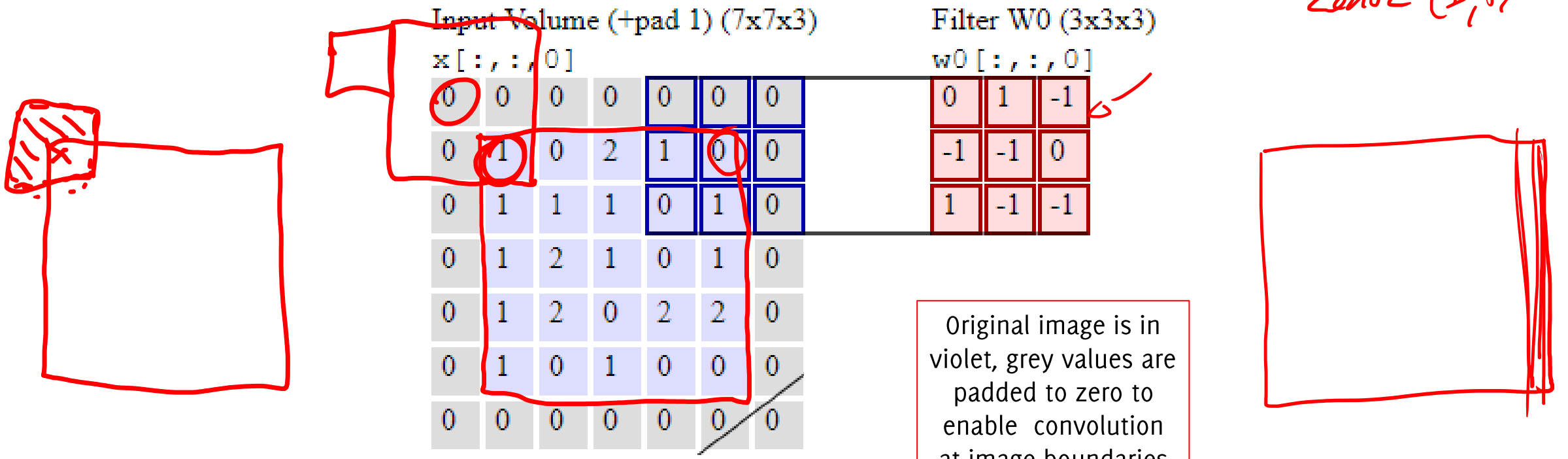


Original image is in violet, grey values are padded to zero to enable convolution at image boundaries

Convolution: Padding

How to define convolution output close to image boundaries?

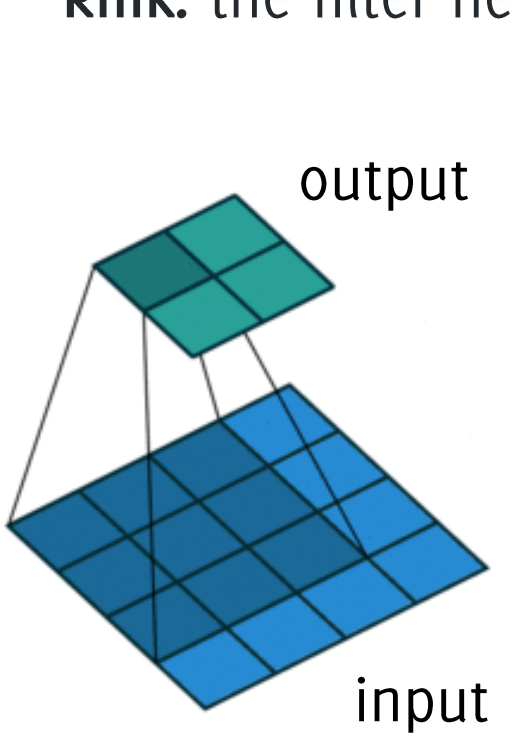
Padding with zero is the most frequent option, as this does not change the output size. However, no padding or symmetric padding are also viable options



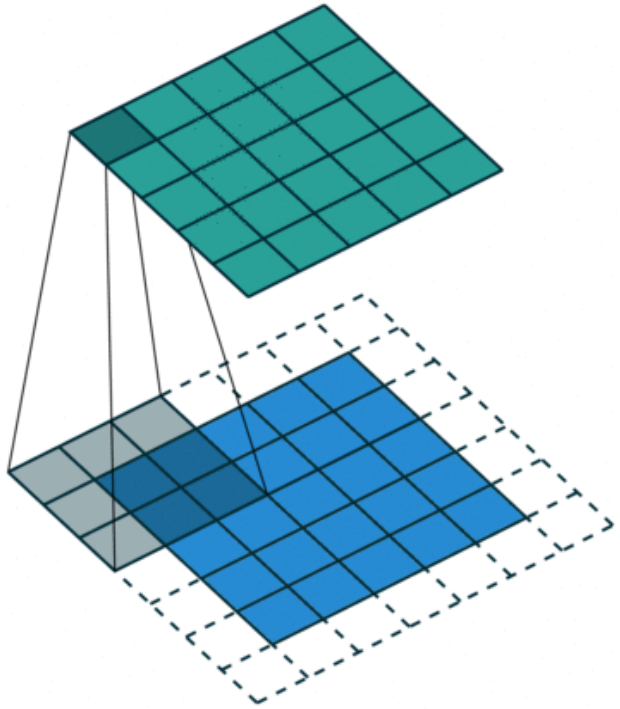
Original image is in violet, grey values are padded to zero to enable convolution at image boundaries

Padding Options in Convolution Animation

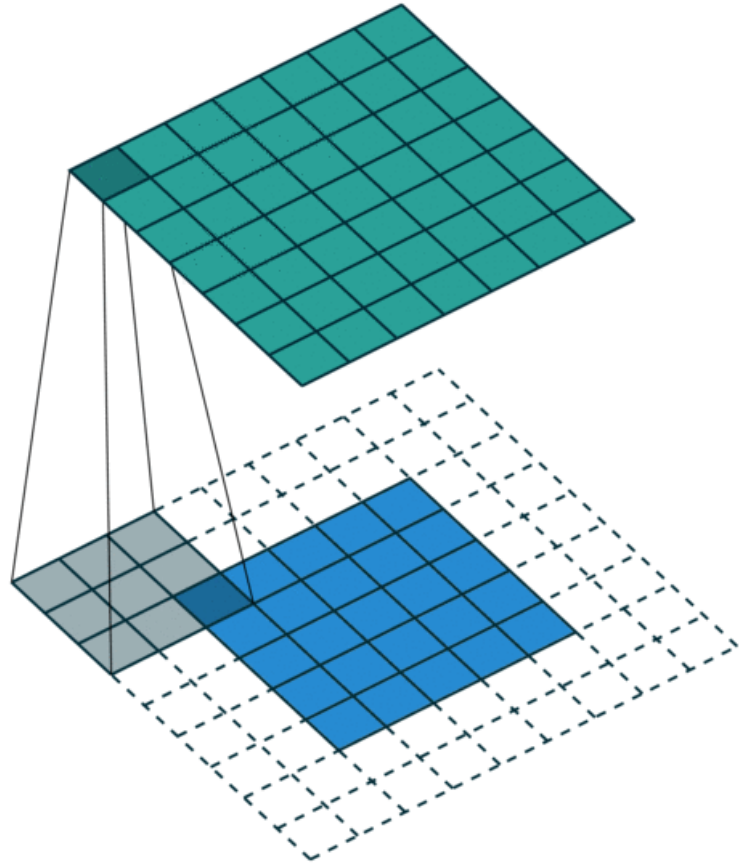
Rmk: Blue maps are inputs, and cyan maps the outputs.
Rmk: the filter here is 3×3



No padding
«valid»



Half padding
«same»



full padding
«full»

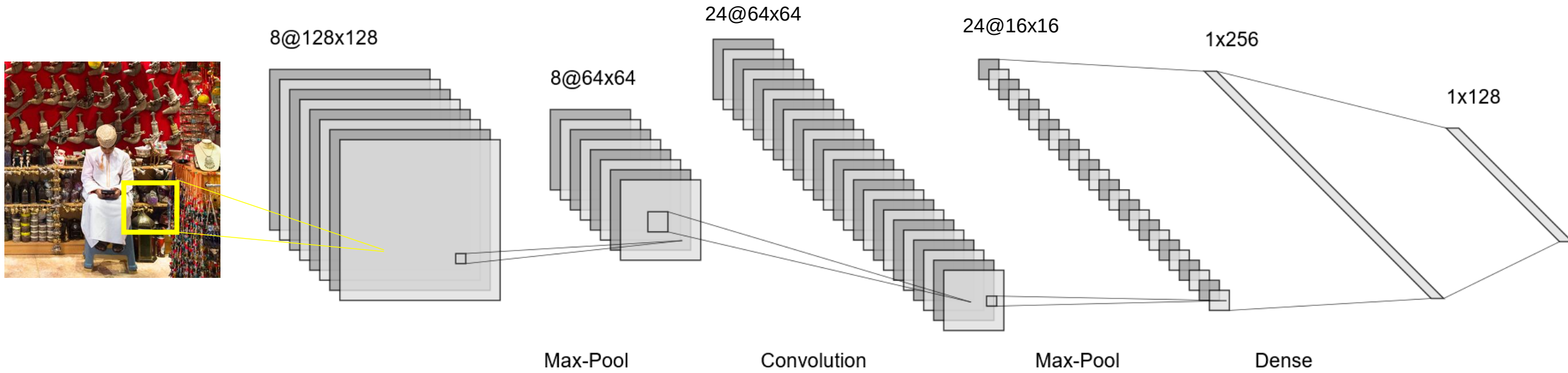
Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

Convolutional Neural Networks

CNNs

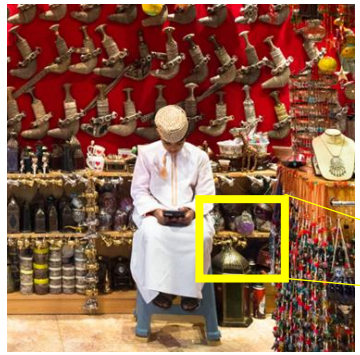
Giacomo Boracchi

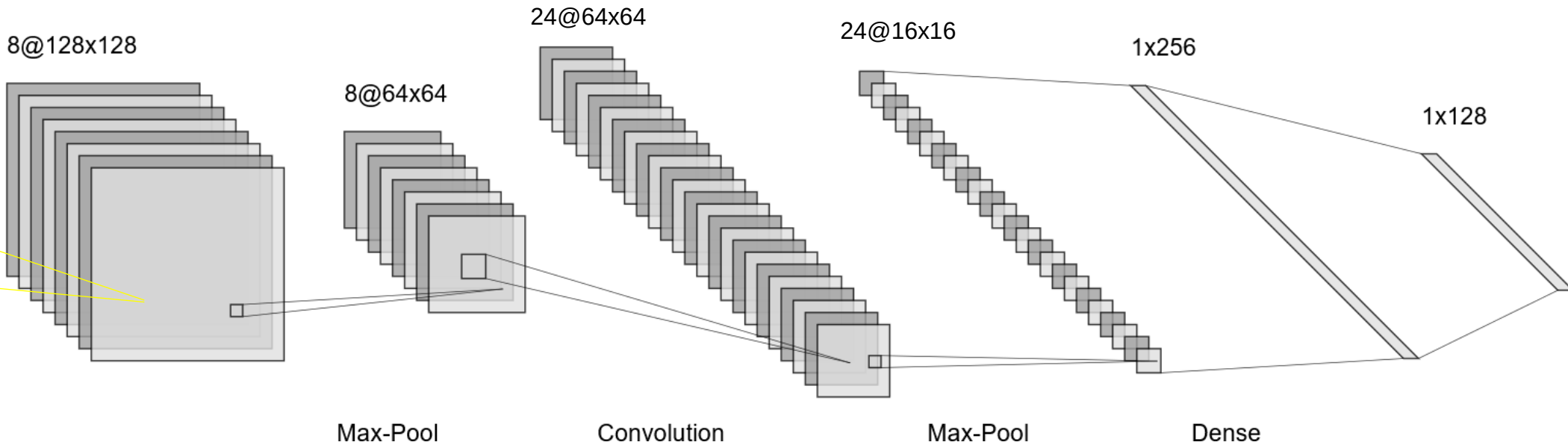
The typical architecture of a CNN



The typical architecture of a CNN

The input of a CNN is an entire image

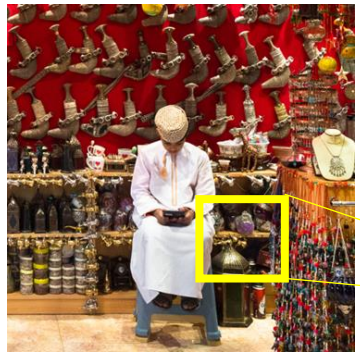


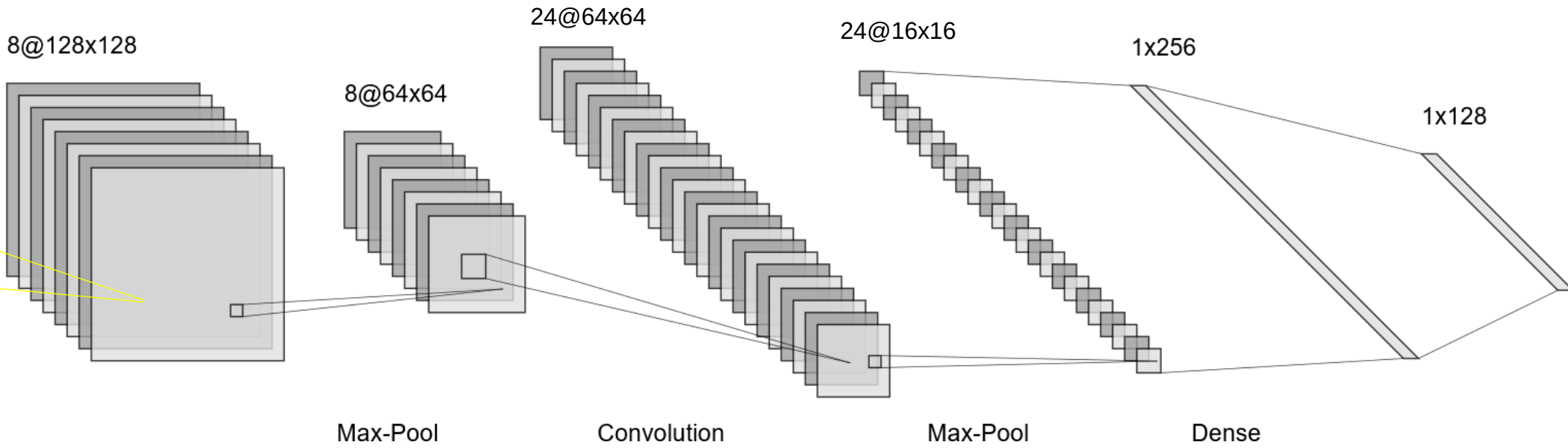


The image gets convolved against many filters

The typical architecture of a CNN

The input of a CNN is an entire image



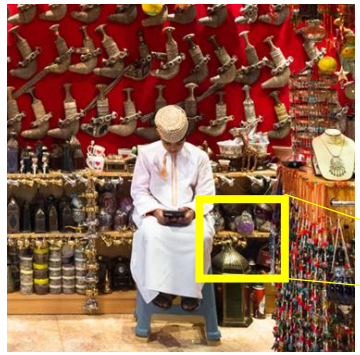


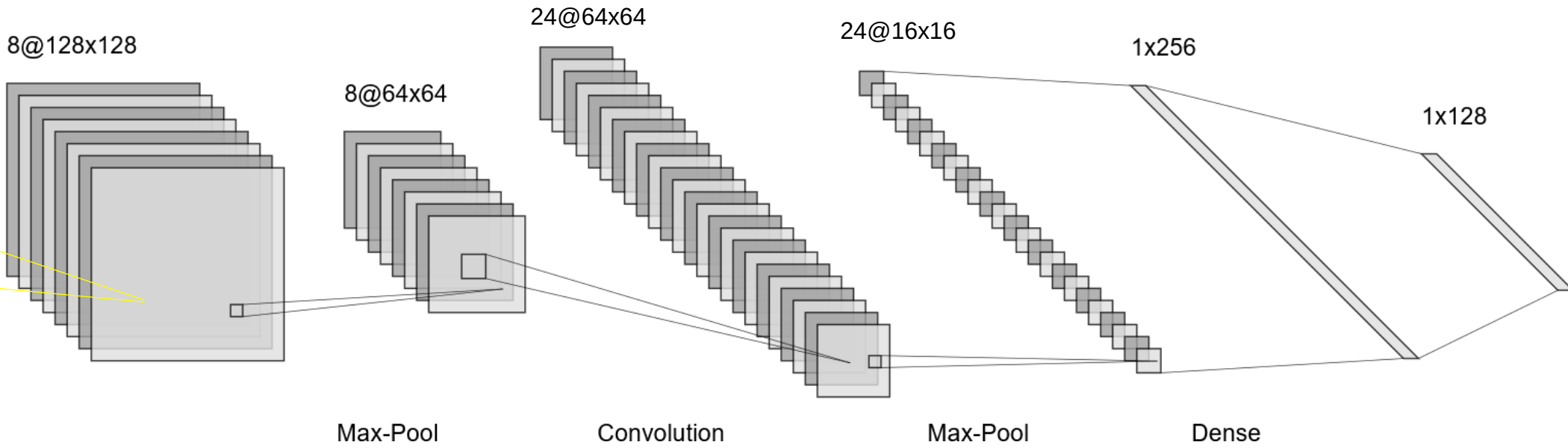
The image gets convolved against many filters

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

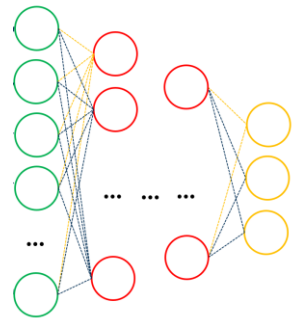




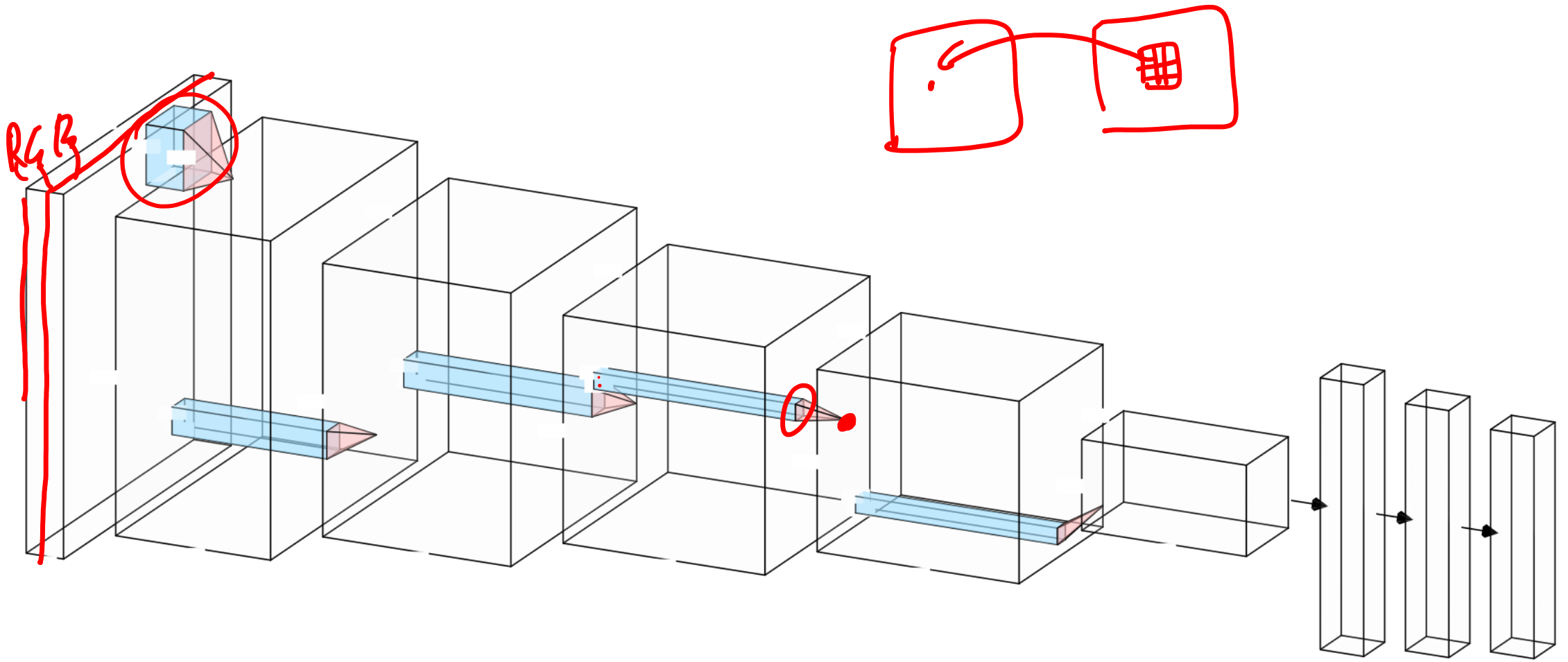
The image gets convolved against many filters

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



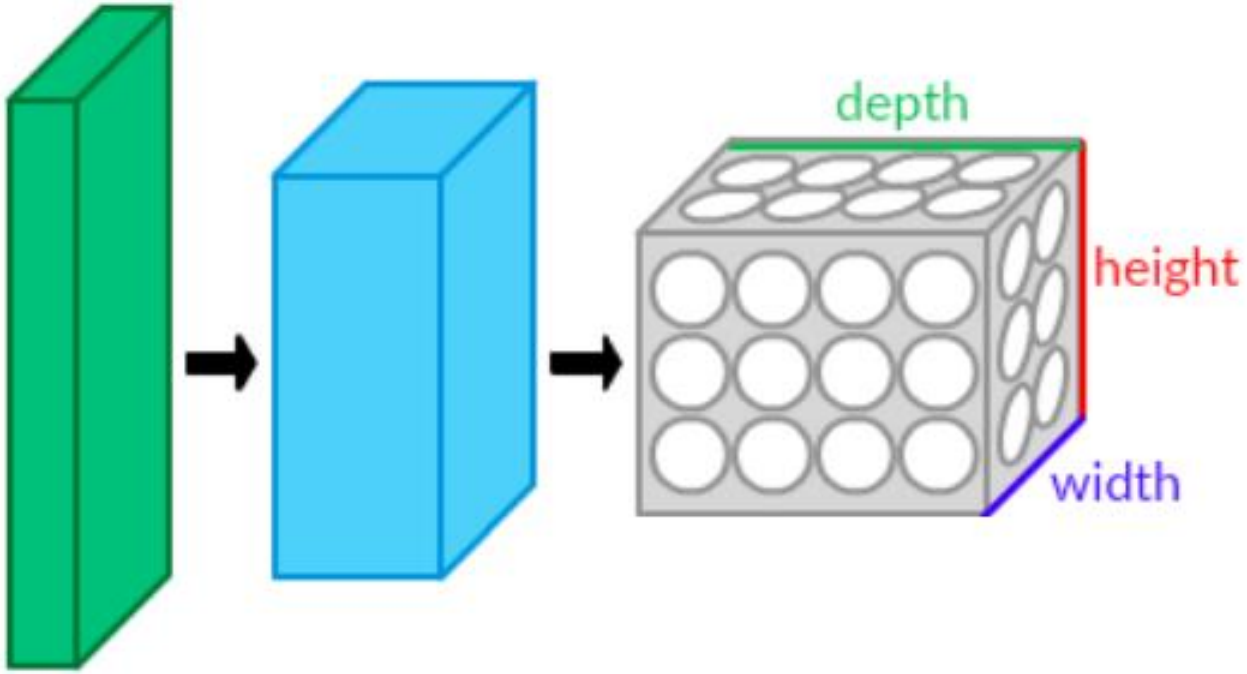
The typical architecture of a CNN



Convolutional Neural Networks (CNN)

CNN are typically made of blocks that include:

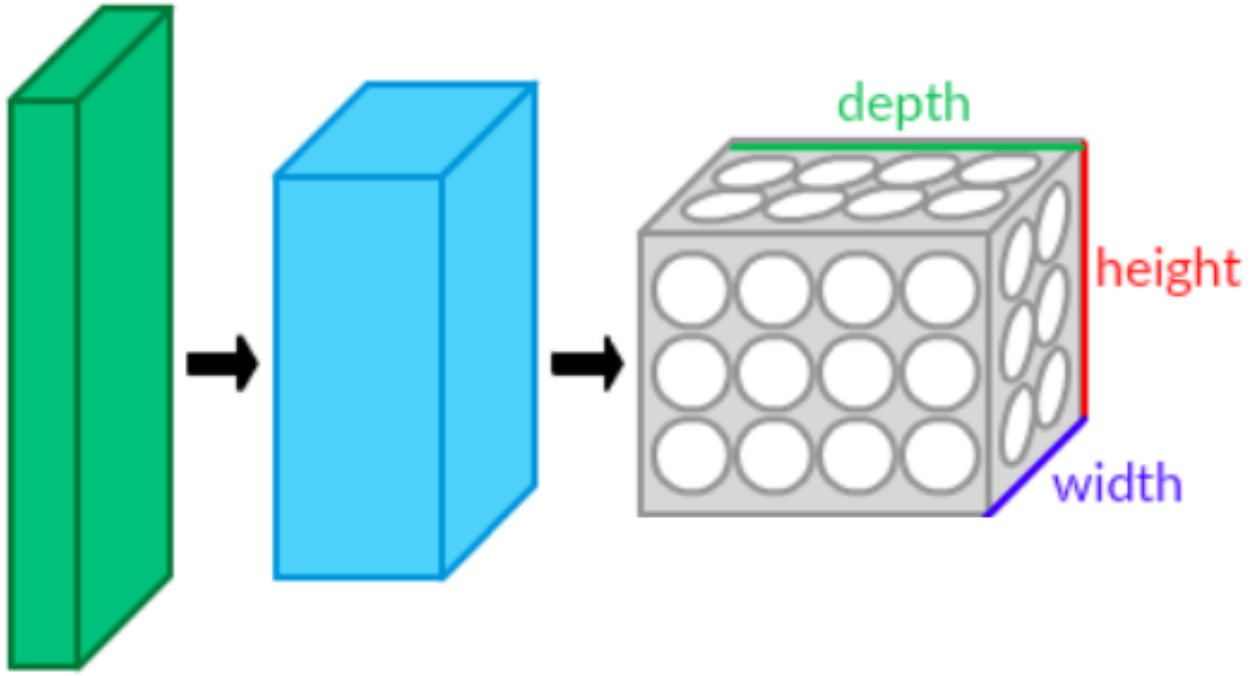
- Convolutional layers
- Nonlinearities (activation functions)
- Pooling Layers (Subsampling / maxpooling)



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

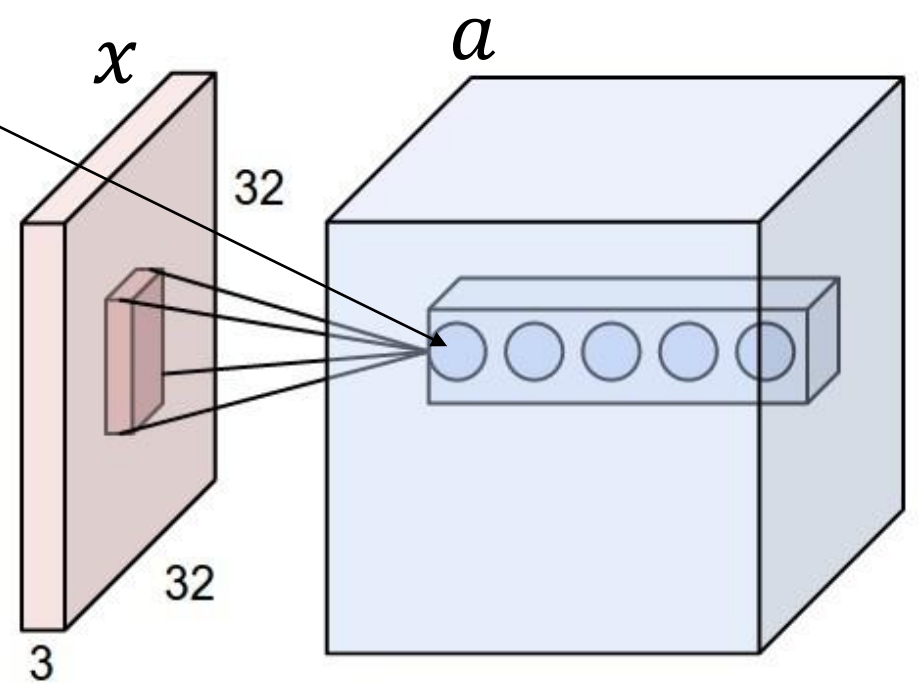
Giacomo Boracchi

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

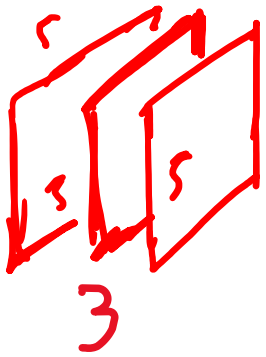
By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

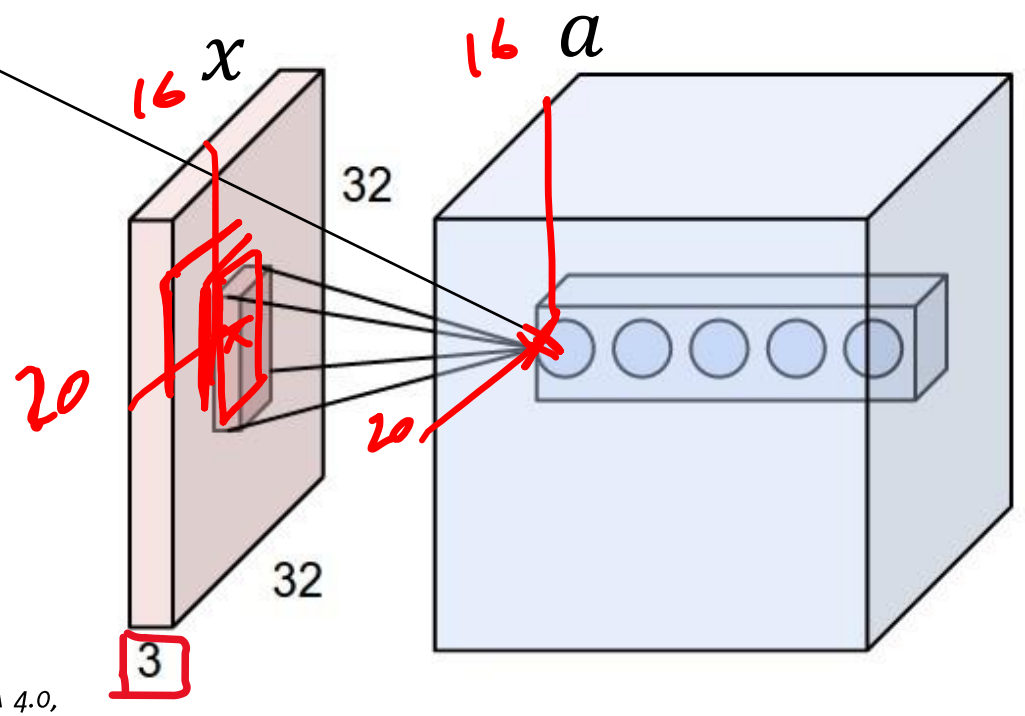
Convolutional Layers

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**

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





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

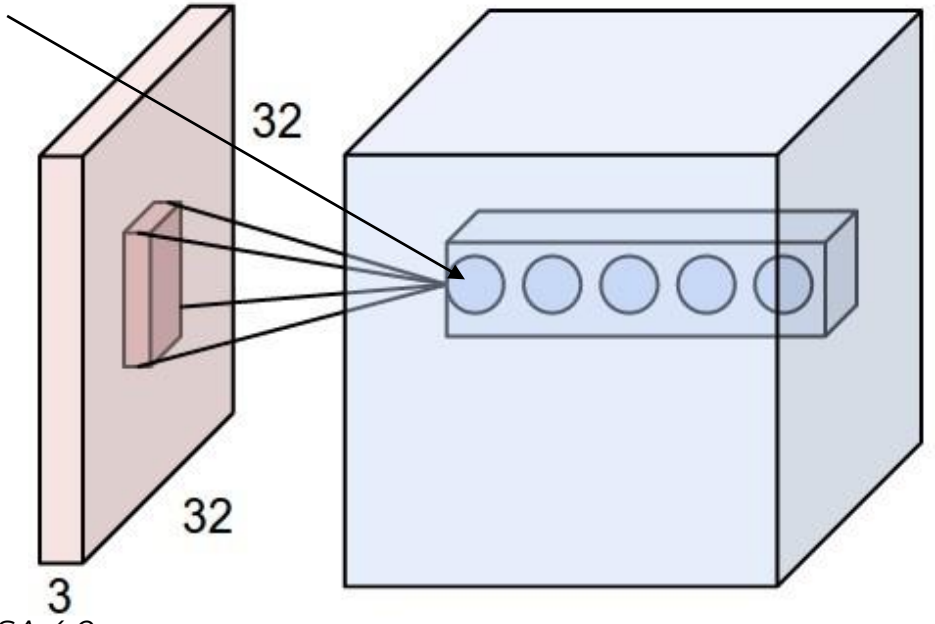
Convolutional layers "mix" all the input components

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

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

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

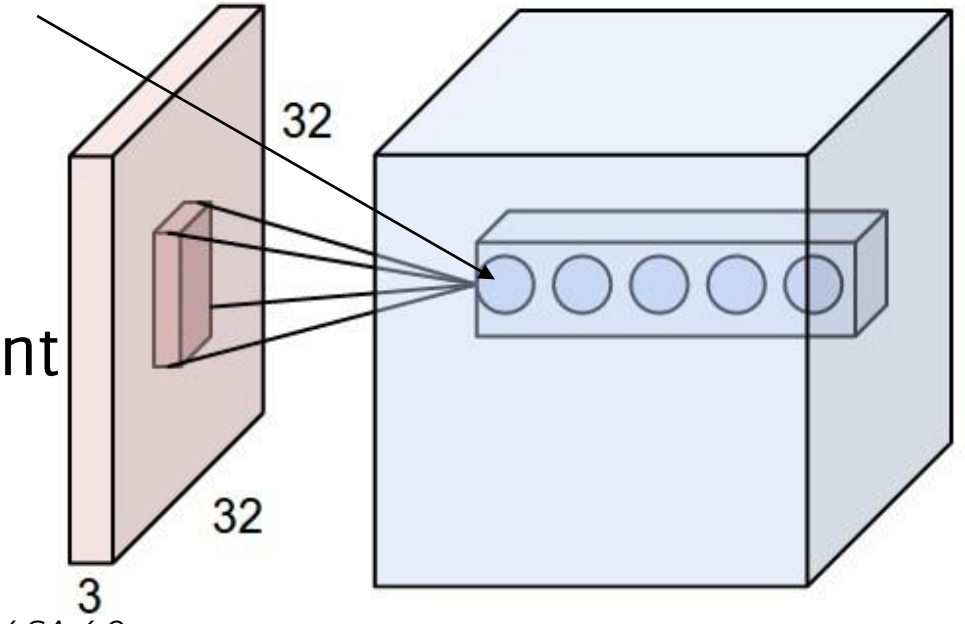
The output is a linear combination of **all the values in a region of the input, considering all the channels**

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

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

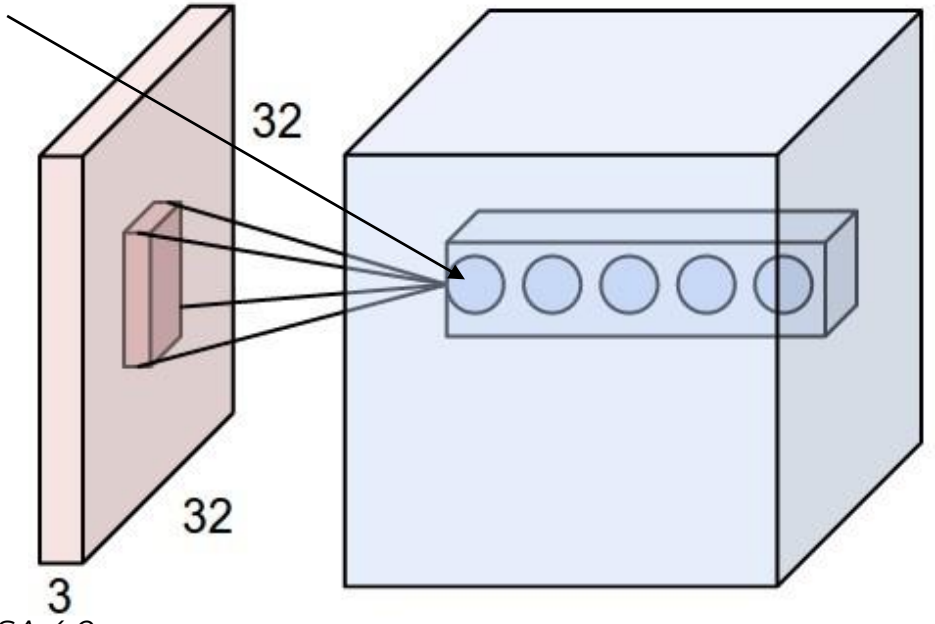
The output is a linear combination of **all the values in a region of the input, considering all the channels**

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

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

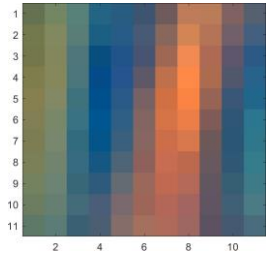


By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

Convolutional Layers

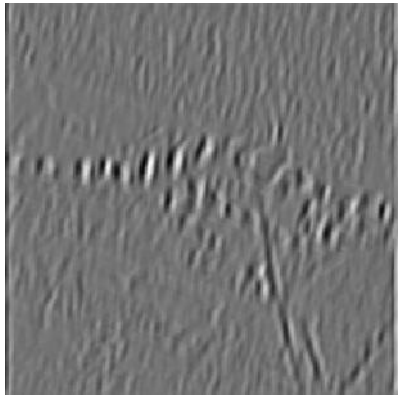
I

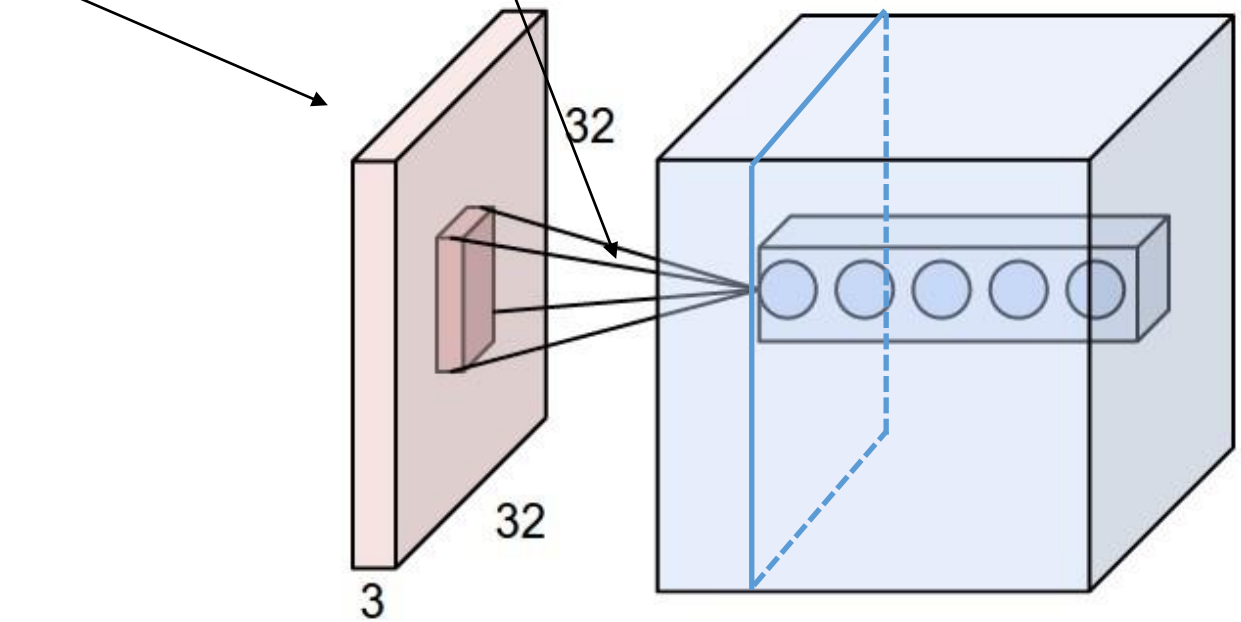
w^1



b^1

$a(:,:,1)$





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

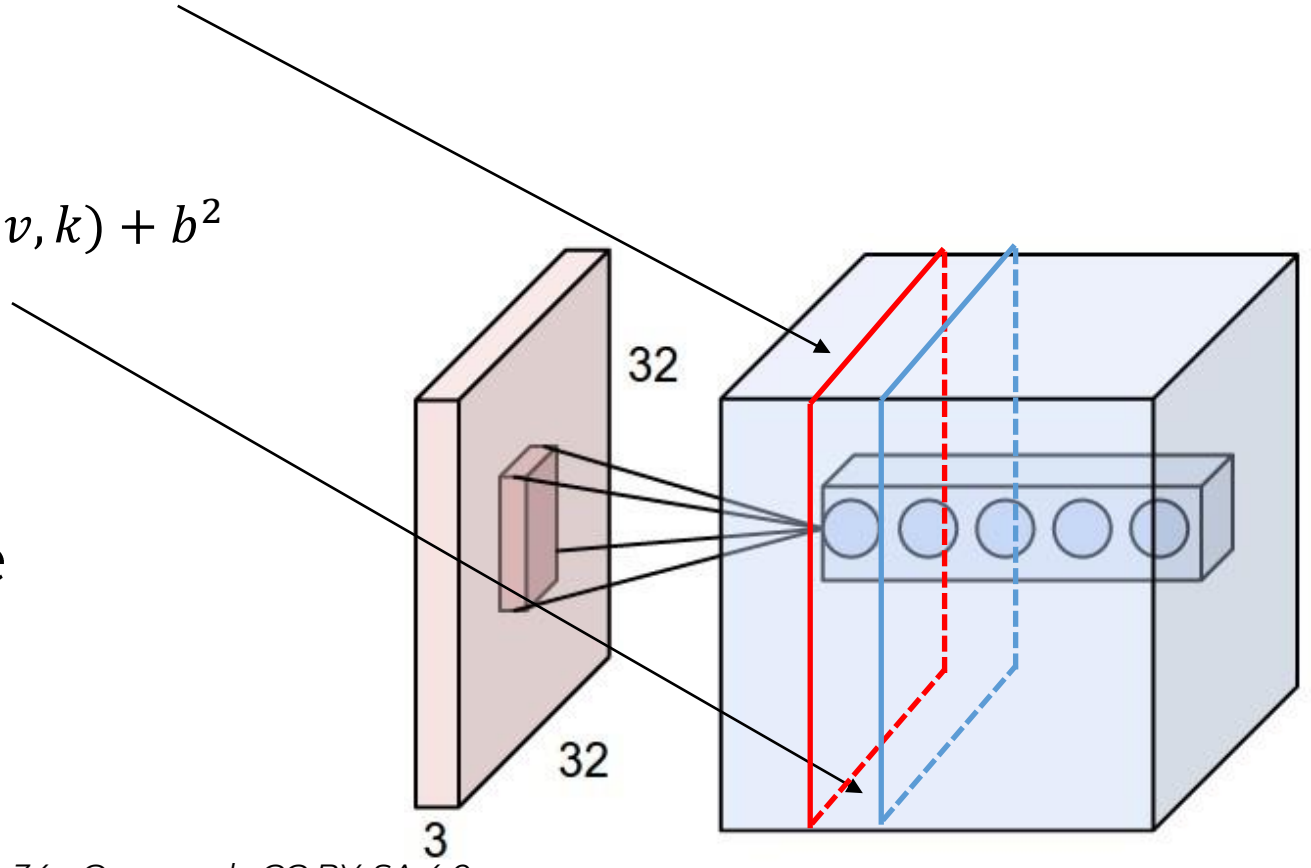
Convolutional Layers

Different filters yield different layers in the output

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

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

Different filters of the same layer have the same spatial extent



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

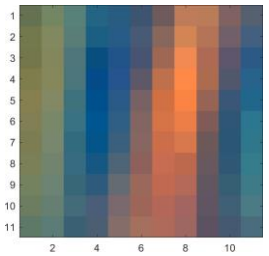
Convolutional Layers

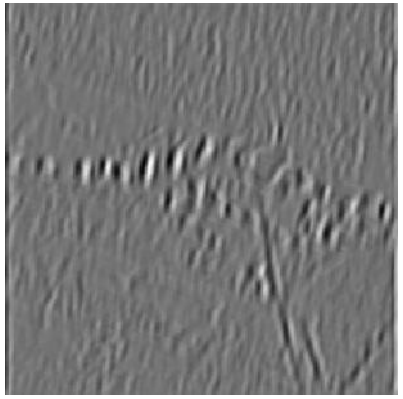
$a(:,:,1)$

$a(:,:,n)$

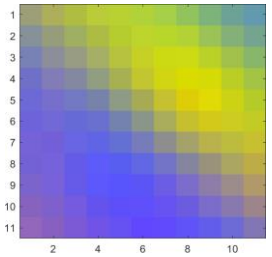
I

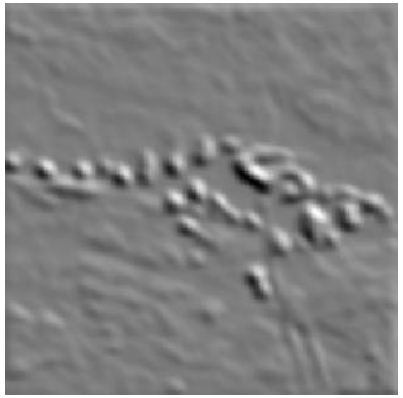
w^1, b^1

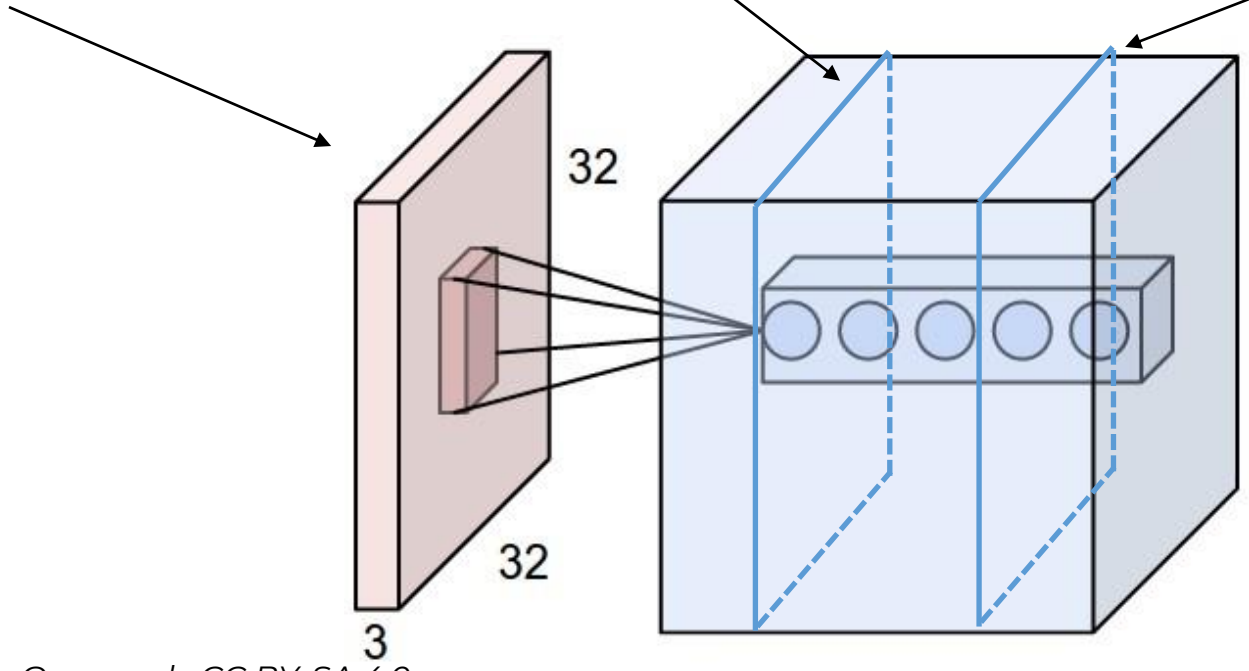




w^n, b^n







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

Convolutional Layers, remarks:

Given:

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

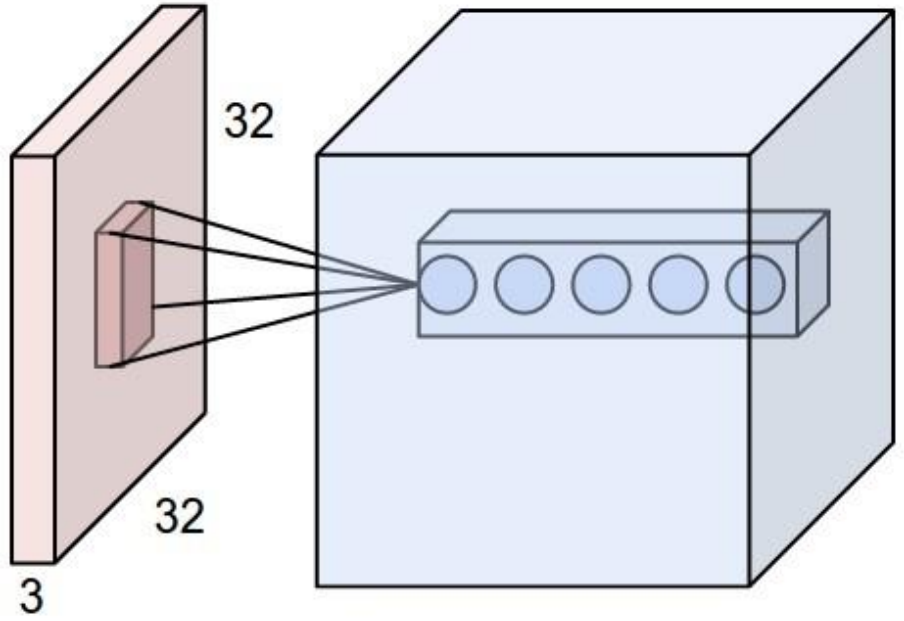
The parametres are the weights + one bias per filter

The overall number of parameters is

$$\left(h_x \cdot h_y \cdot d\right) \cdot n_f + n_f$$

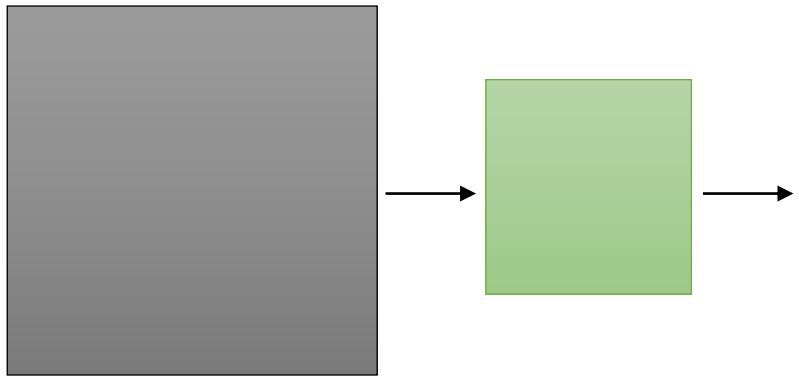
Where d is the **depth of the input activation**

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

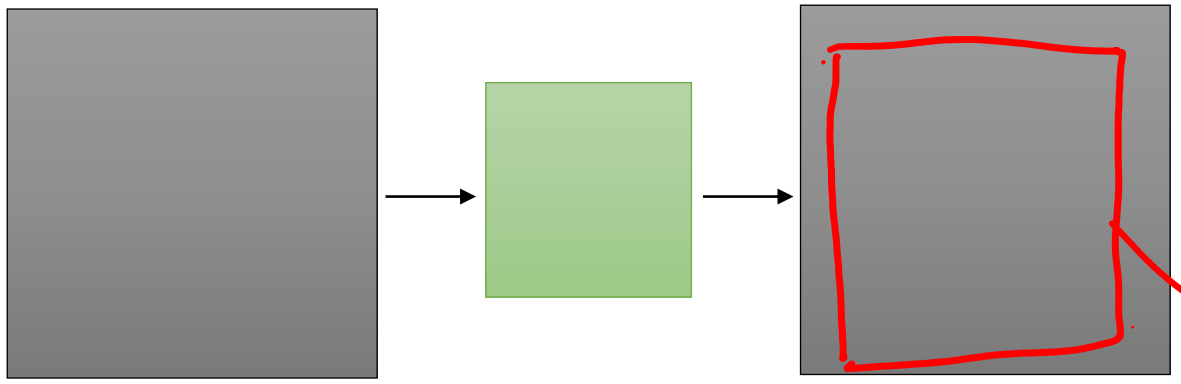
CNN Arithmetic



1 input map

1 filter 5x5

CNN Arithmetic



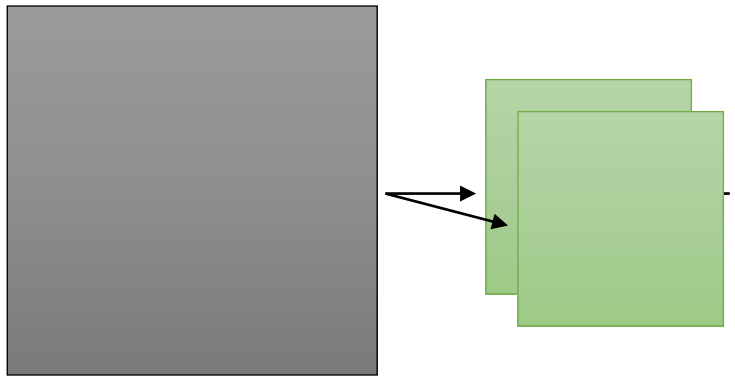
1 input map

1 filters 5x5

1 output map

In case of padding "valid"

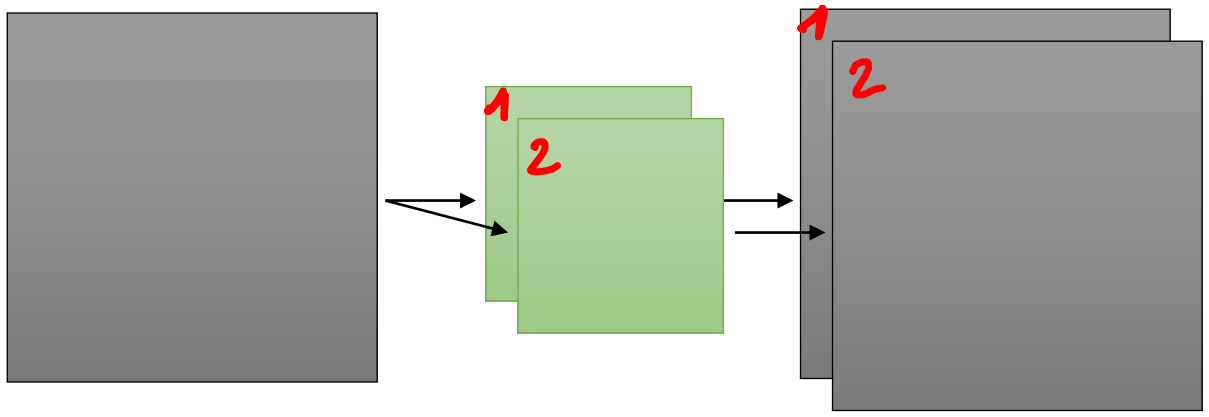
CNN Arithmetic



1 input map

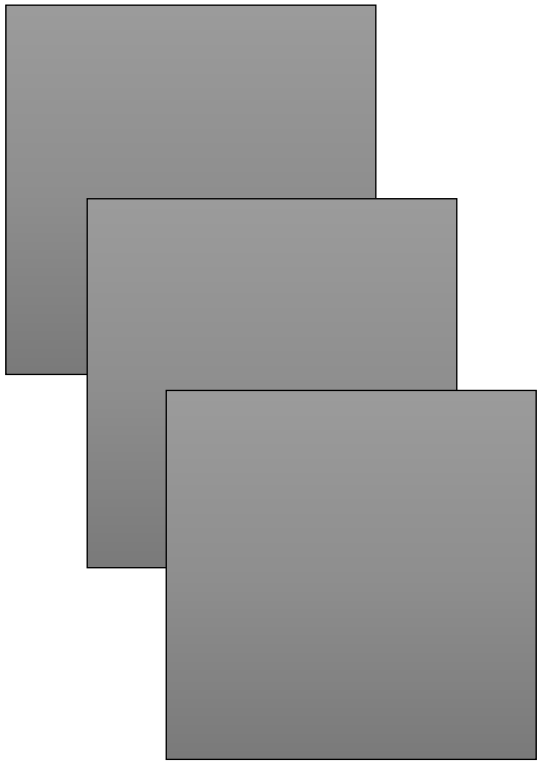
2 filters 5x5

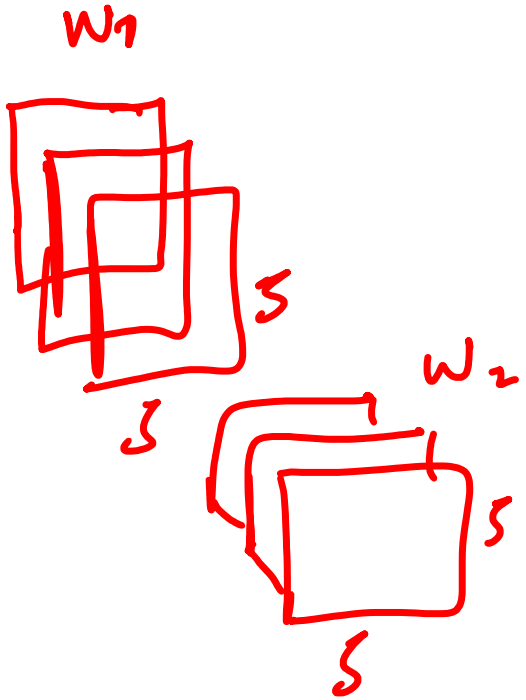
CNN Arithmetic



1 input map 2 filters 5x5 2 output maps

CNN Arithmetic

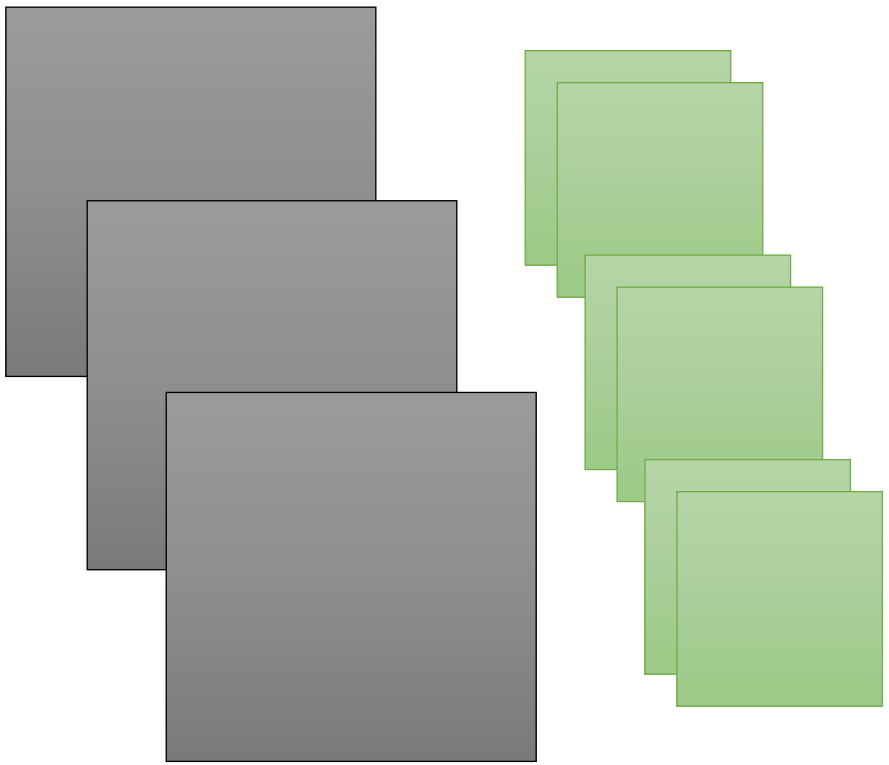




3 input maps

2 filters 5x5

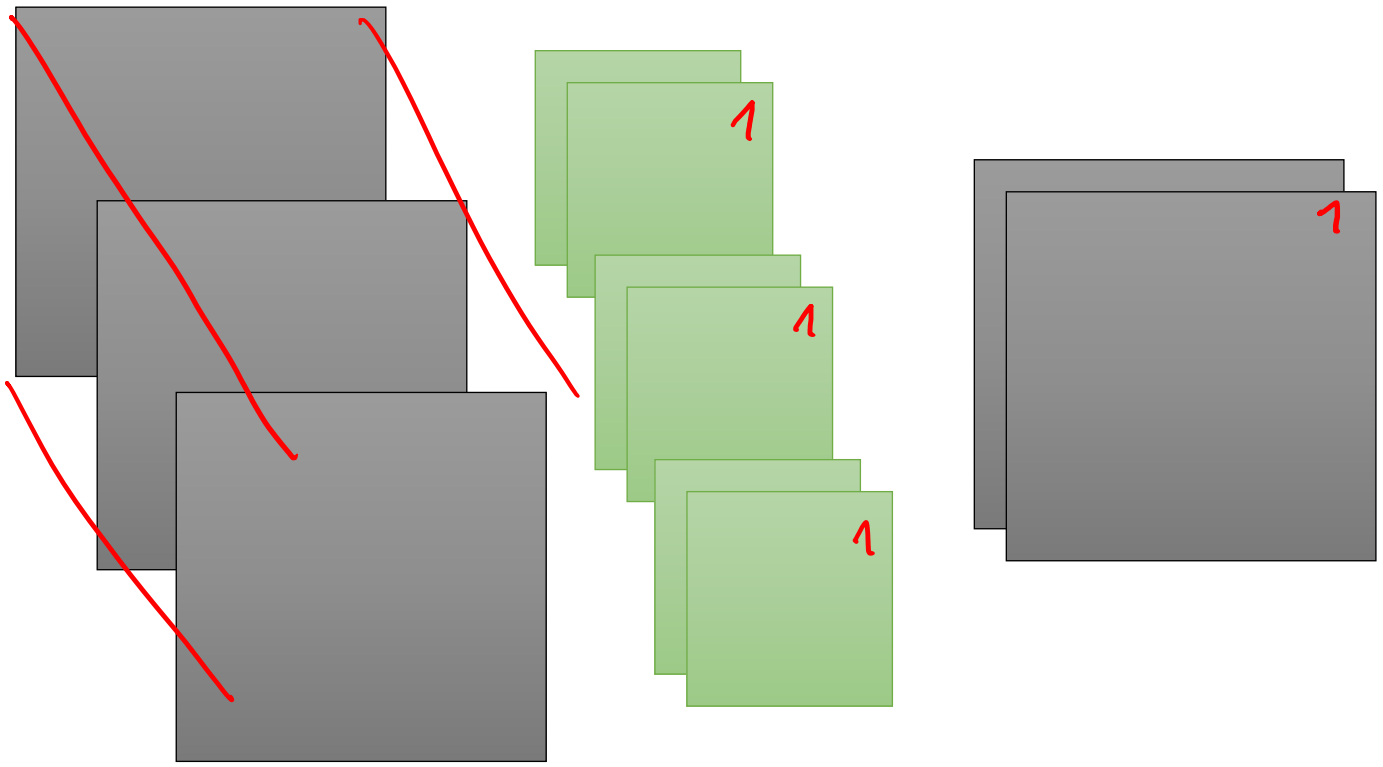
CNN Arithmetic



3 input maps

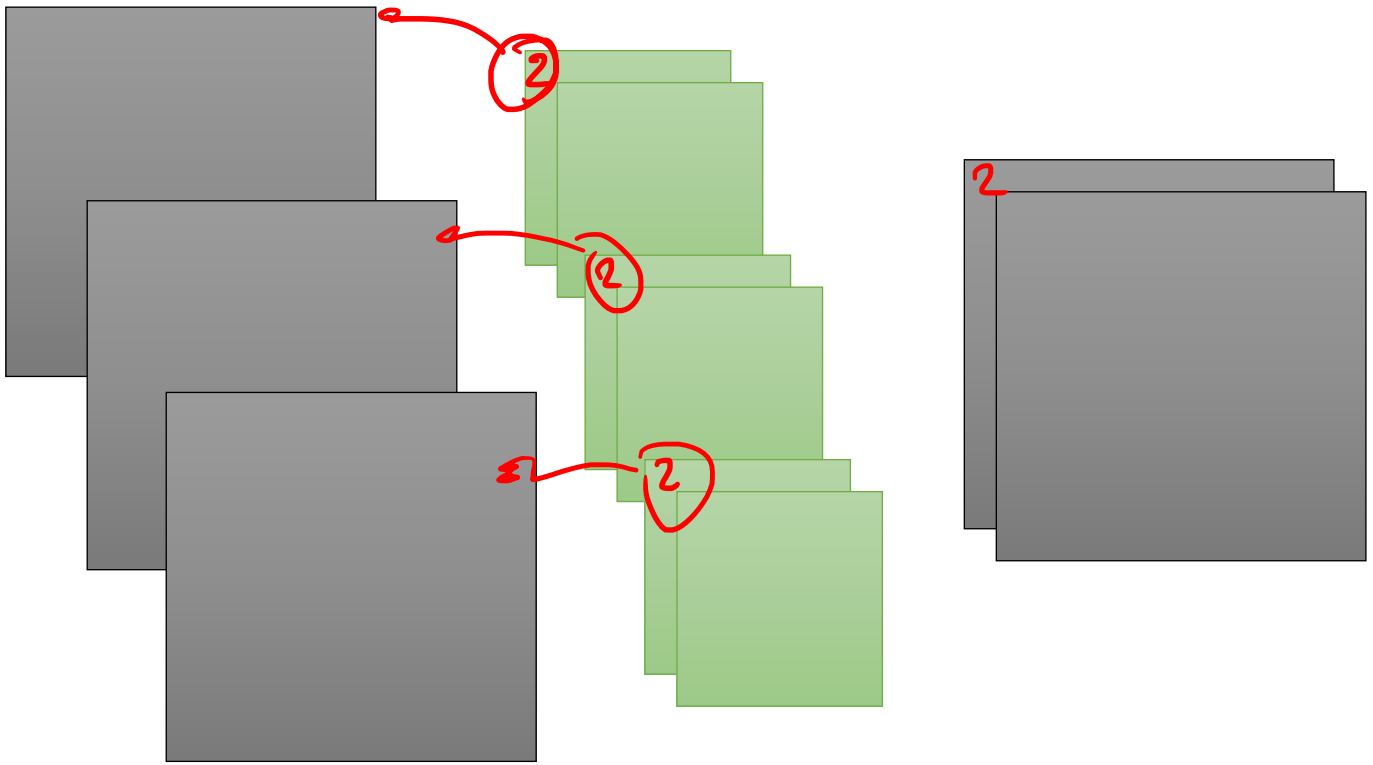
2 filters 5x5

CNN Arithmetic



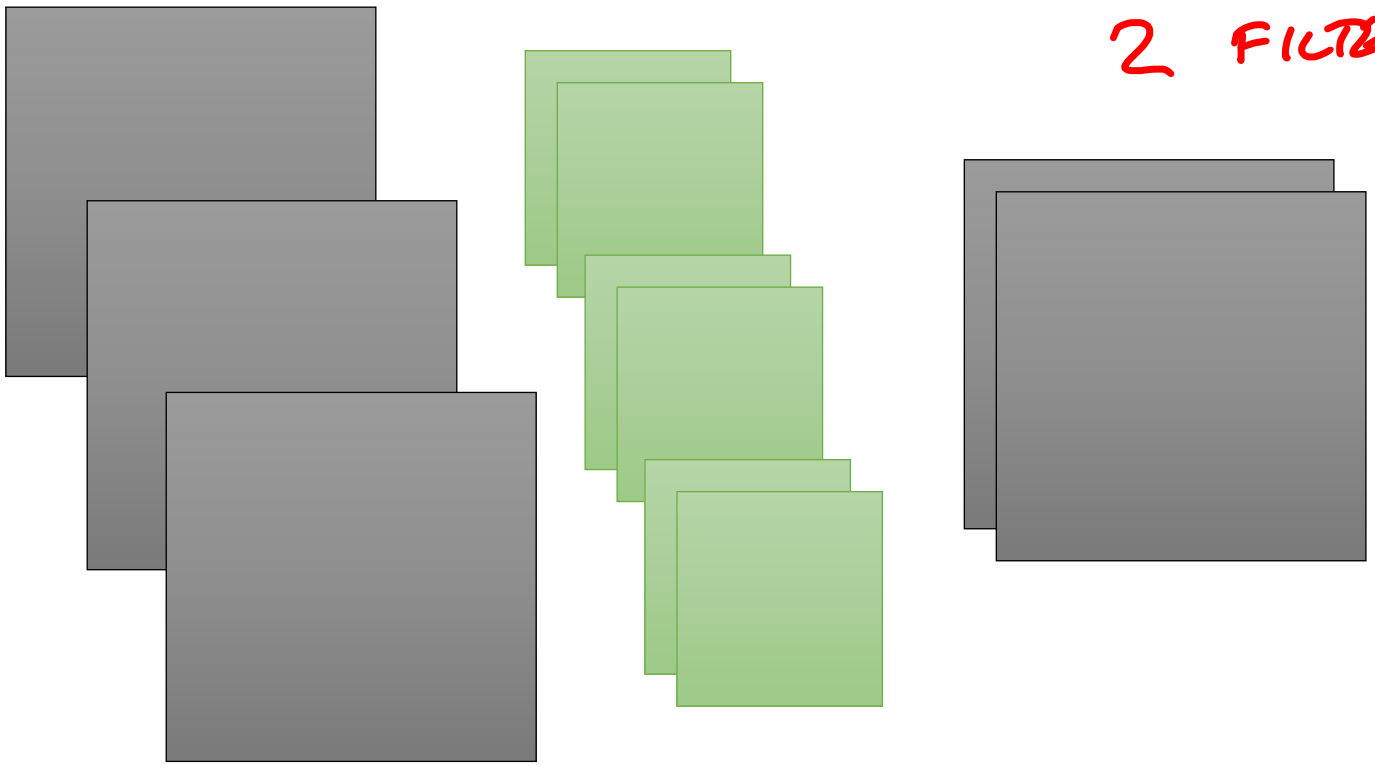
3 input maps 2 filters 5x5 2 output maps

CNN Arithmetic



3 input maps 2 filters 5x5 2 output maps

CNN Arithmetic



2 FILTERS 5x5x3

150 params

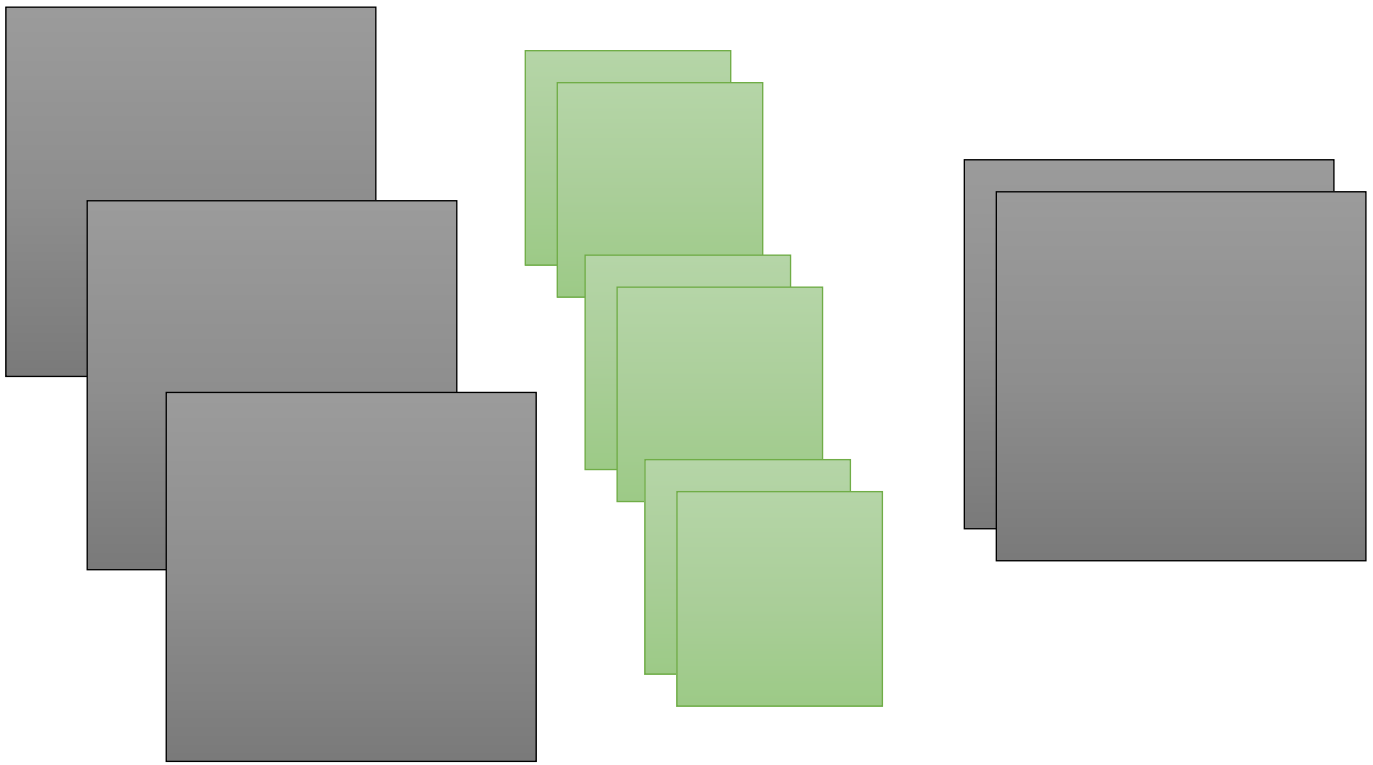
+ 2

152 params

3 input maps 2 filters 5x5 2 output maps

Quiz: how many parameters does this layer have?

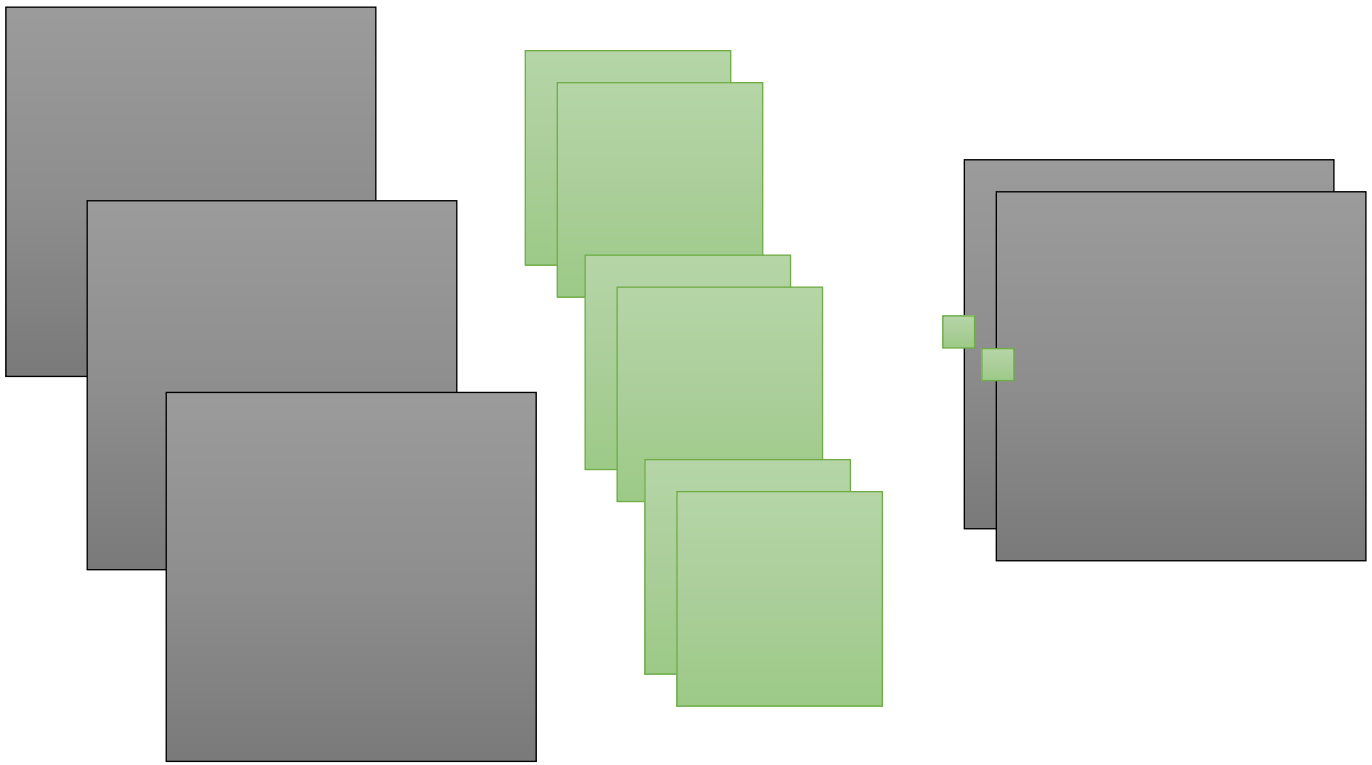
CNN Arithmetic



3 input maps 2 filters 5x5 2 output maps

= 150 parameters in the filters

CNN Arithmetic

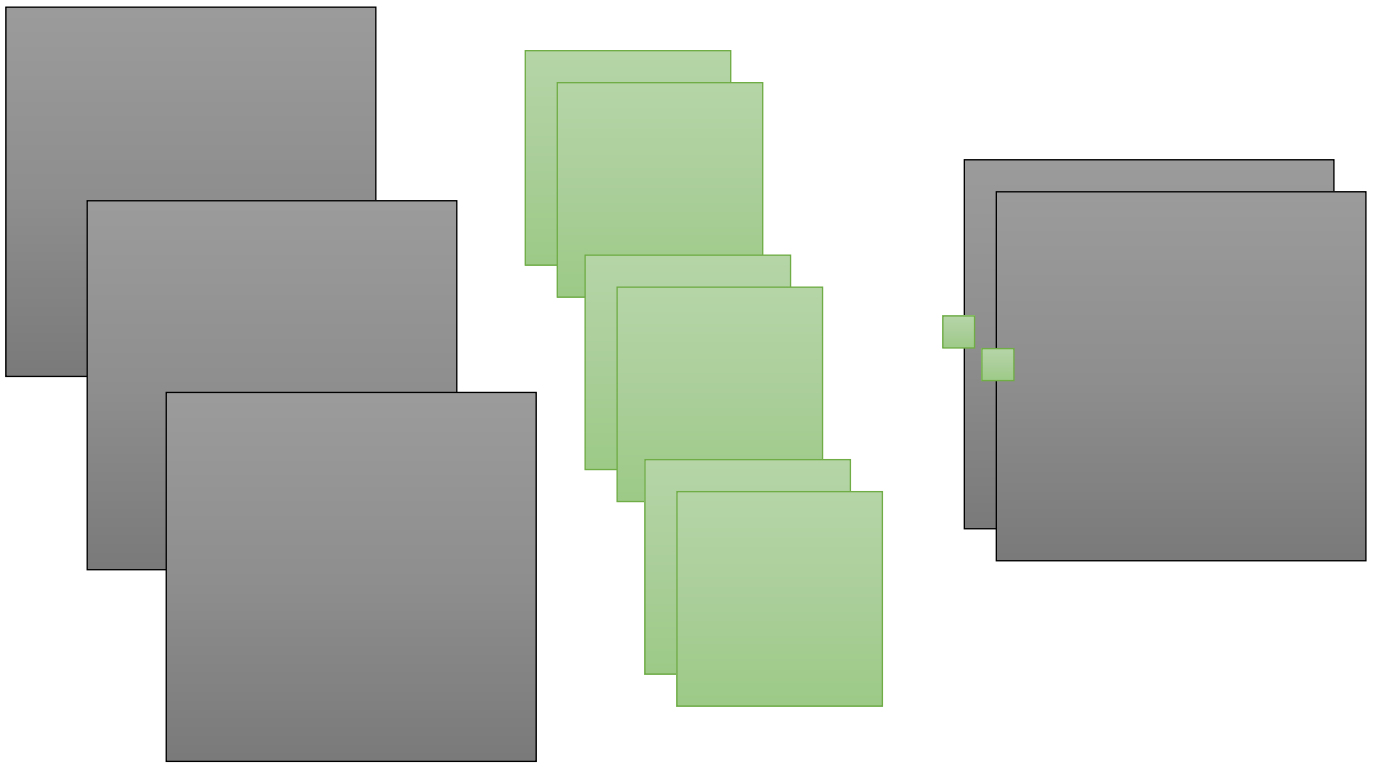


3 input maps

2 filters 5x5
= 150 ...

2 output maps
+ 2 biases

CNN Arithmetic



output maps = # of filters in the layer

3 input maps

2 filters 5x5

= 150 ...

2 output maps

+ 2 biases

= 152 trainable parameters (weights)

To Recap...

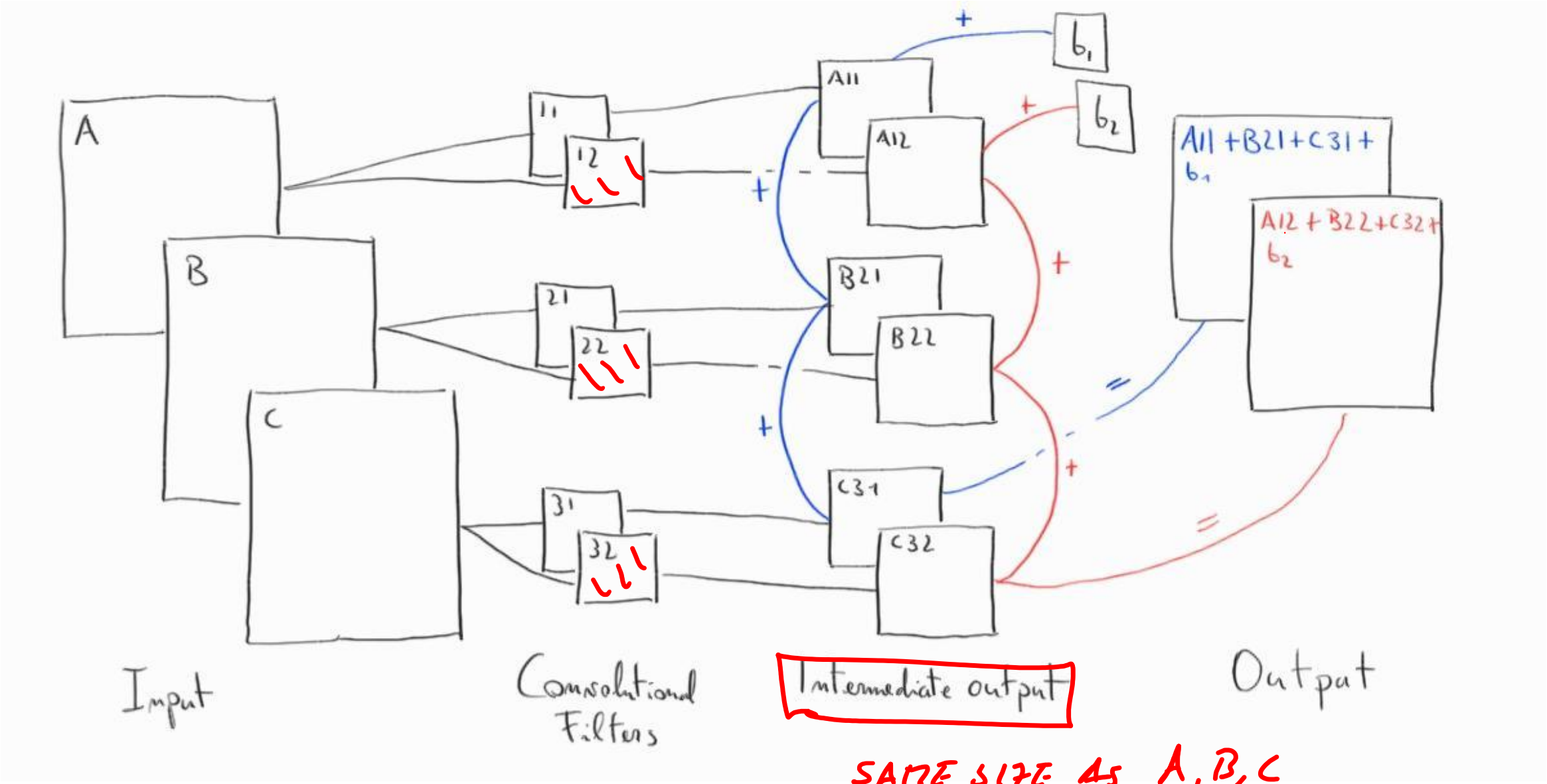


Image credits Nicolò Ghielmetti

Other Layers

Activation and Pooling

Giacomo Boracchi

Activation Layers

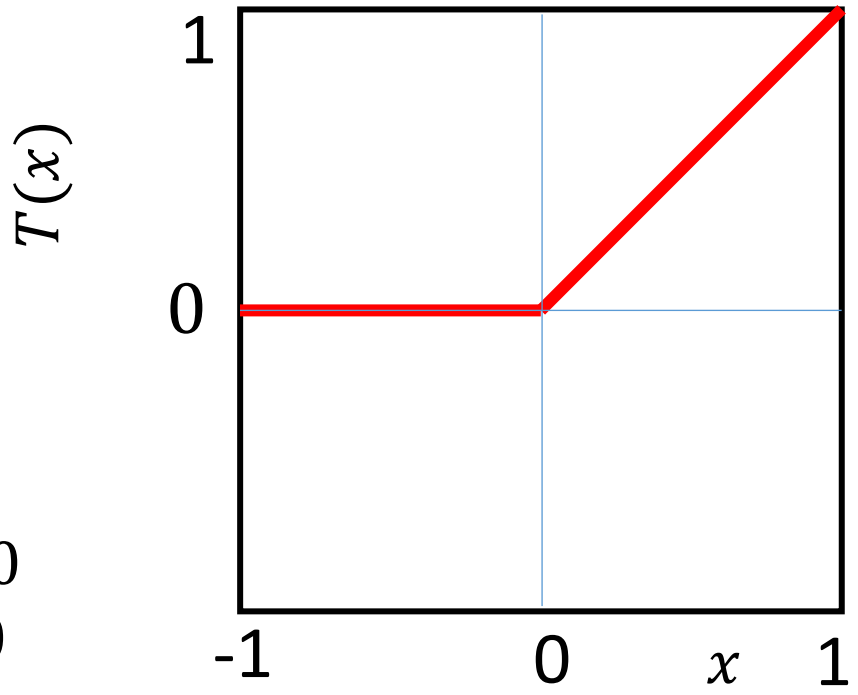
Introduce nonlinearities in the network, otherwise the CNN might be equivalent to a linear classifier…

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)

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

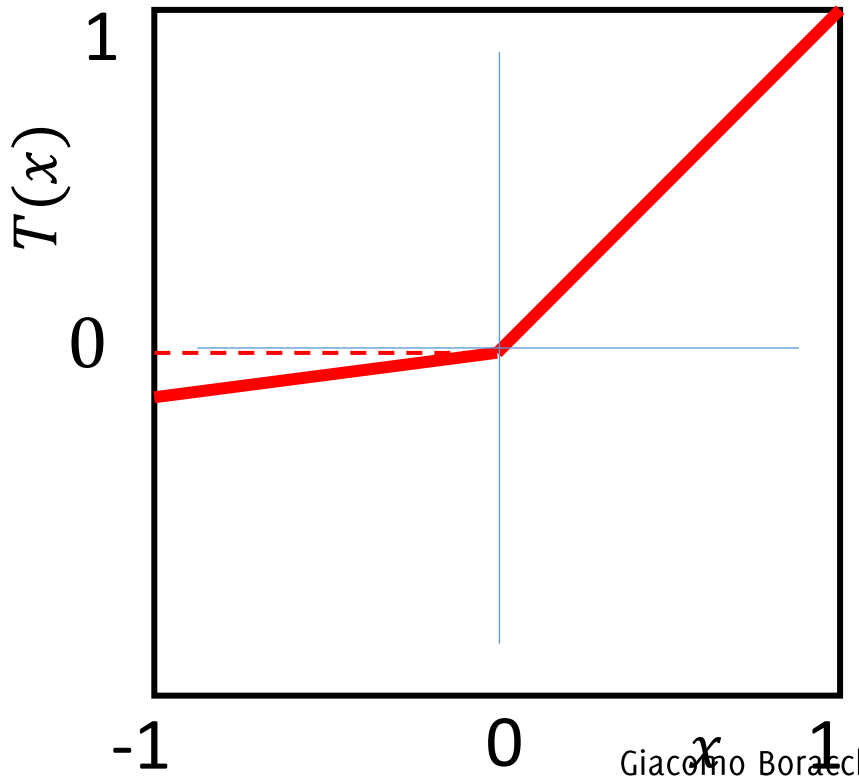


Activation Layers

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

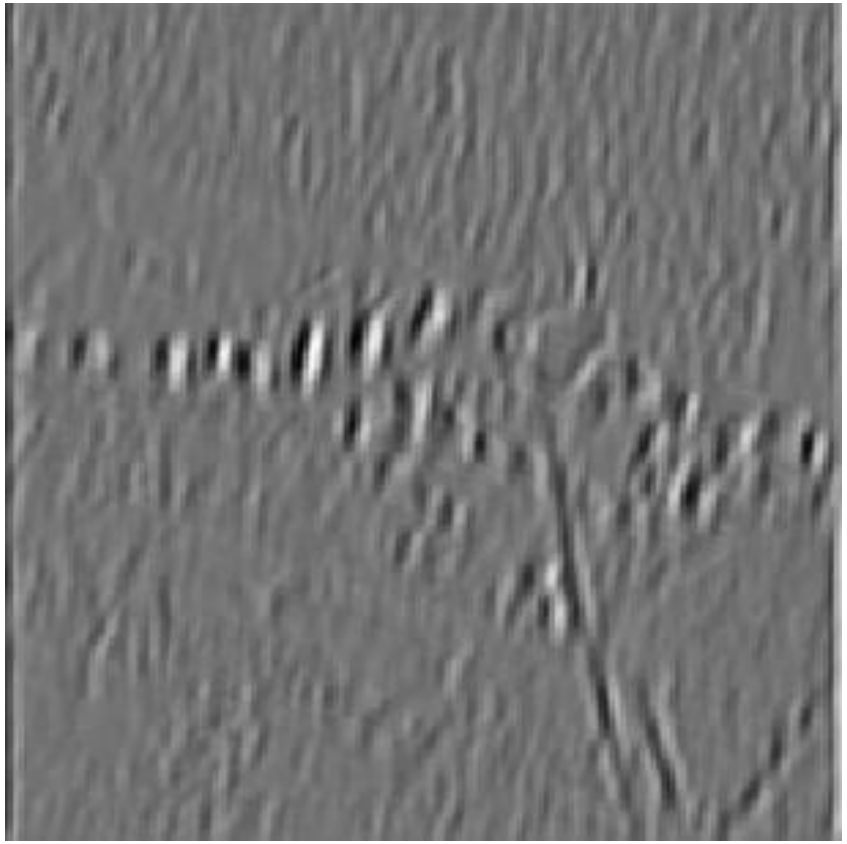
$$T(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0.01 * x & \text{if } x < 0 \end{cases}$$

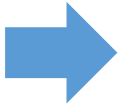


ReLu

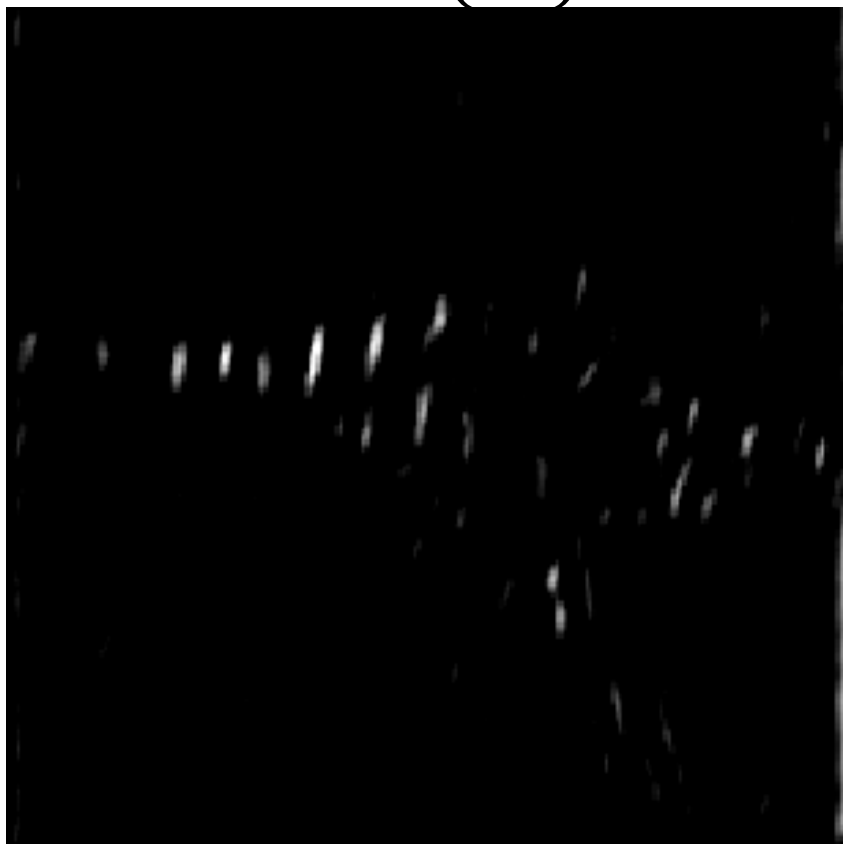
Acts separately on each layer

a^1





$ReLU(a^1)$

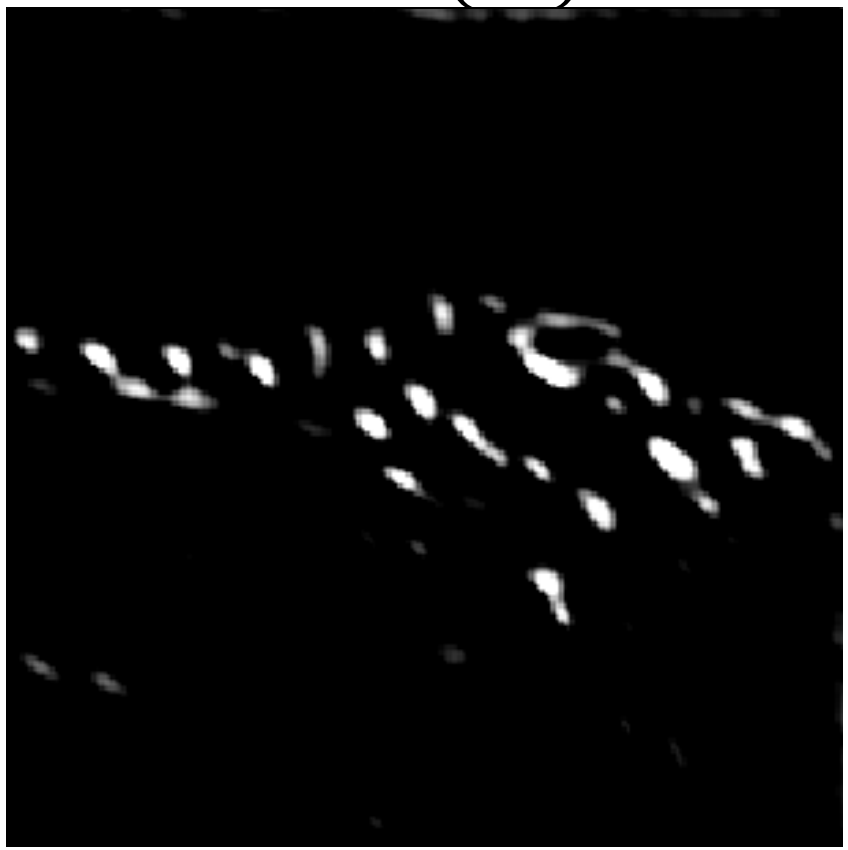


ReLu

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…

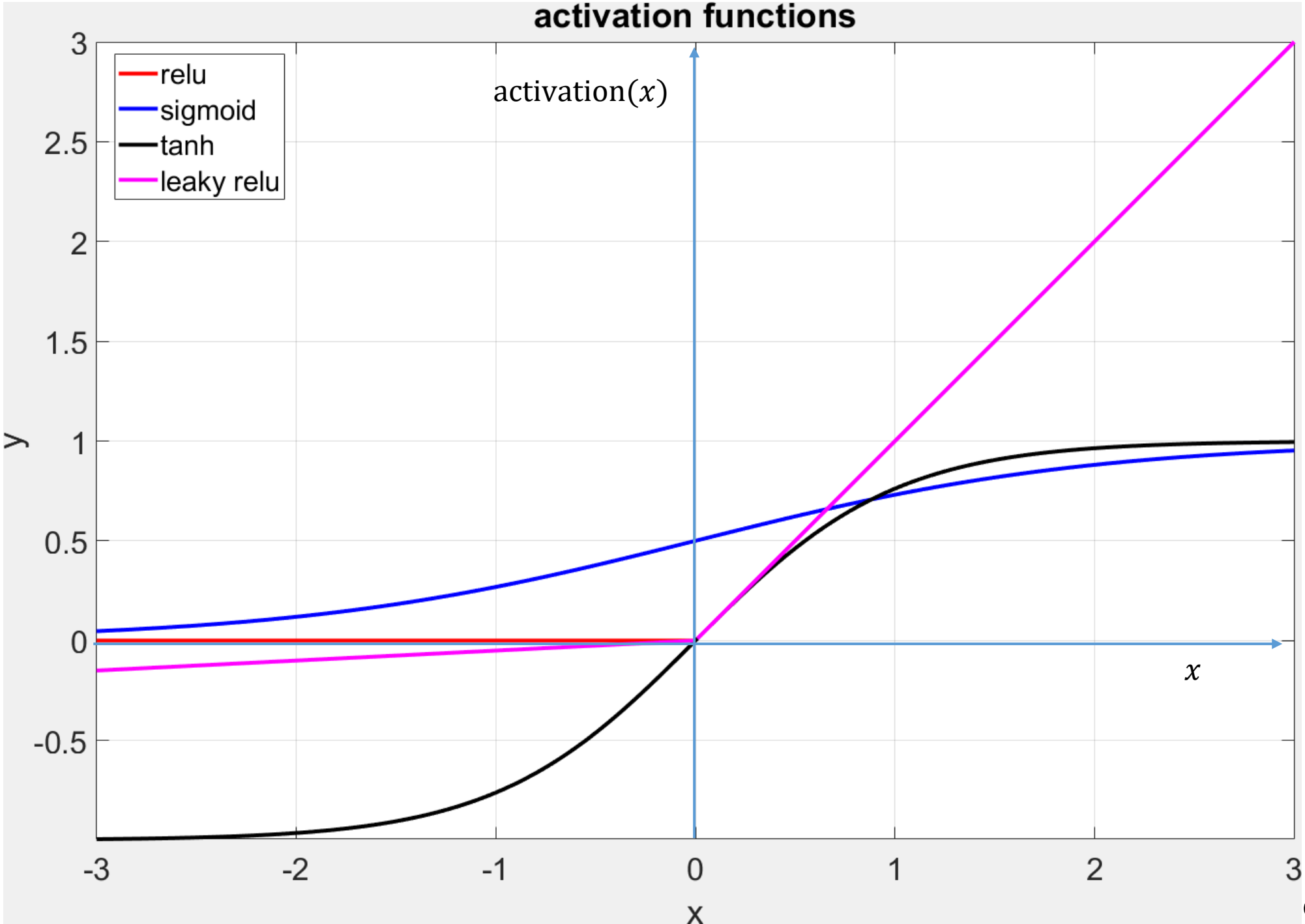
TANH (hyperbolic Tangent): has a range (-1,1), continuous and differentiable

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

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

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

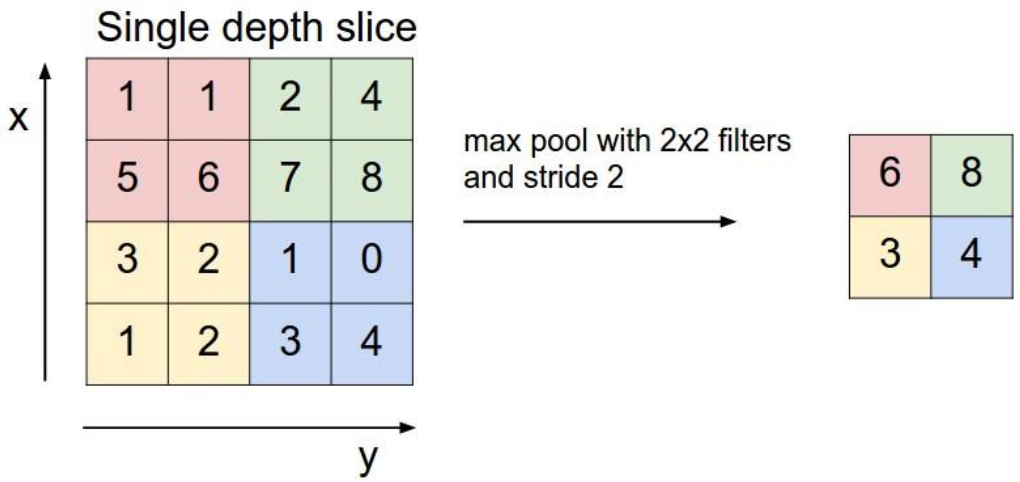
These activation functions are mostly popular **in MLP architectures**

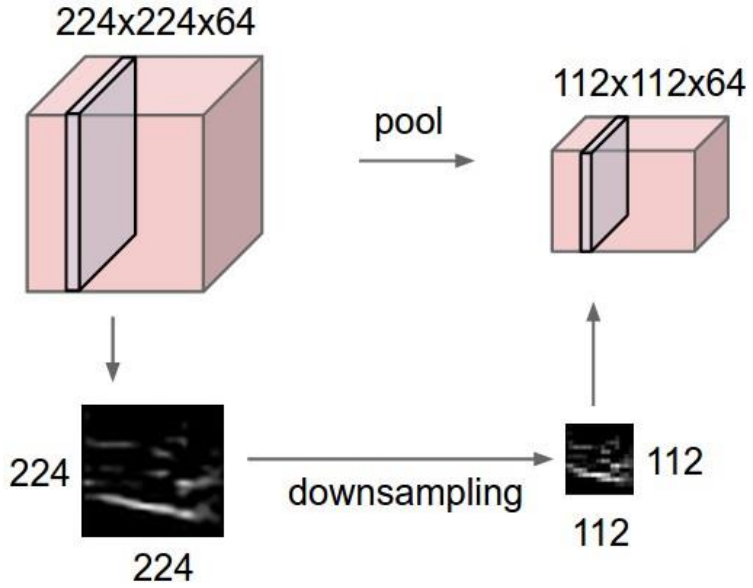

activation functions
relu
sigmoid
tanh
leaky relu
activation(x)
x
y
3
2.5
2
1.5
1
0.5
0
-0.5
-3
-2
-1
0
1
2
3
x

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.**



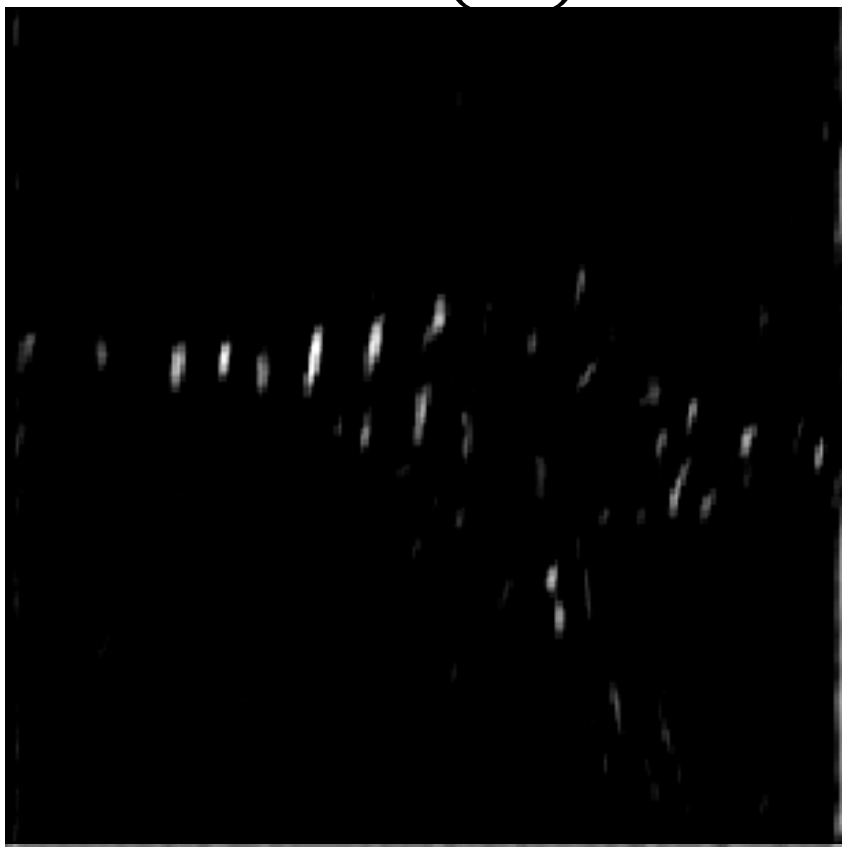


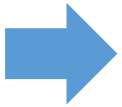
In a 2x2 support it discards 75% of samples in a volume

Max-Pooling (MP)

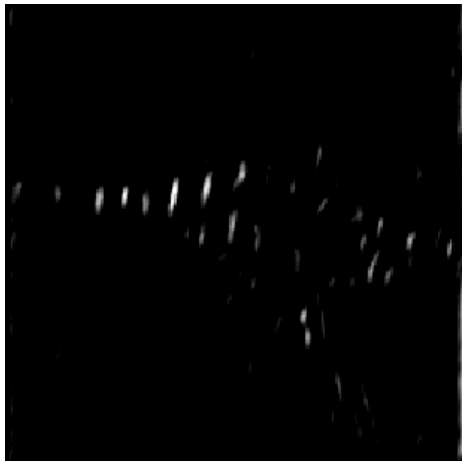
Acts separately on each layer

$ReLU(a^1)$





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



Strides in Pooling Layers

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

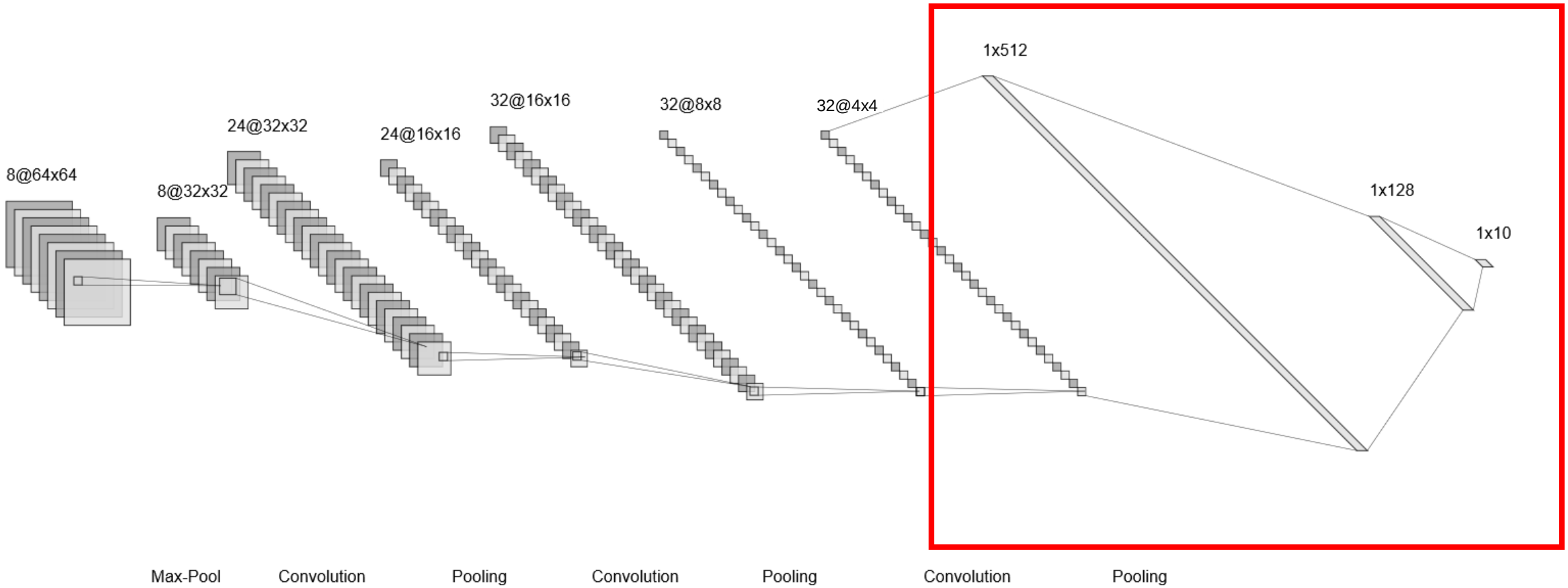
- this operation makes sense with nonlinear pooling (max-pooling)

Dense Layers

As in feed-forward NN

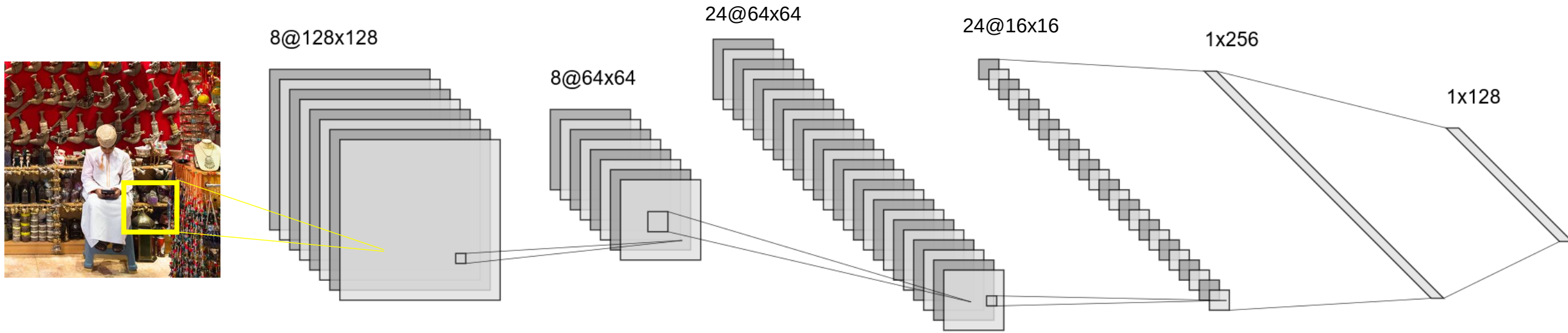
The Dense Layers

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)

The typical architecture of a convolutional neural network



Assume zero padding

Convolution: There are 8 filters

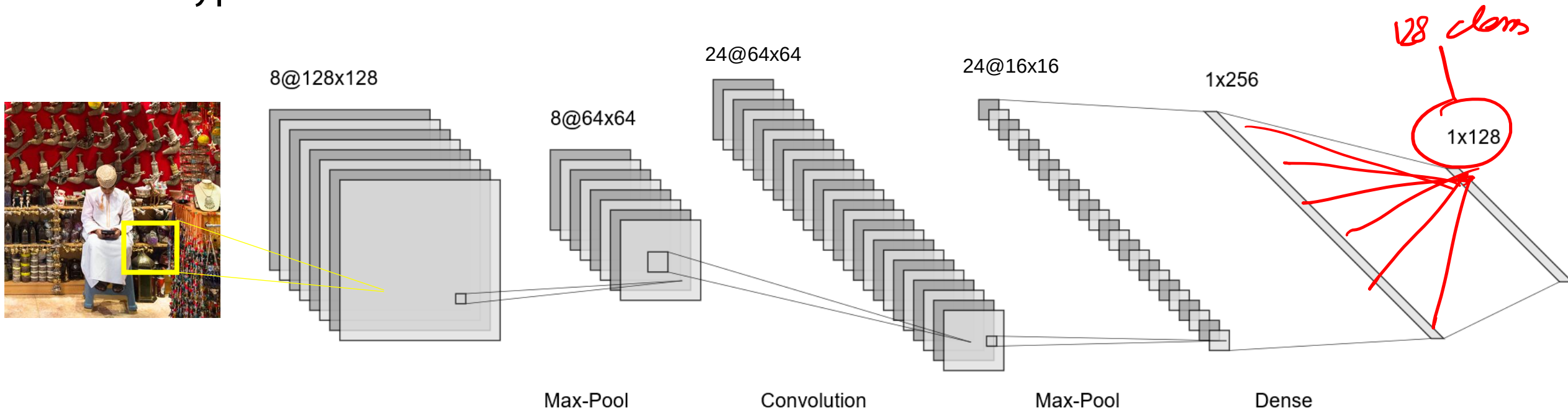
Max-pooling reduces spatial dimensions by 2

Convolution: There are 24 filters

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)

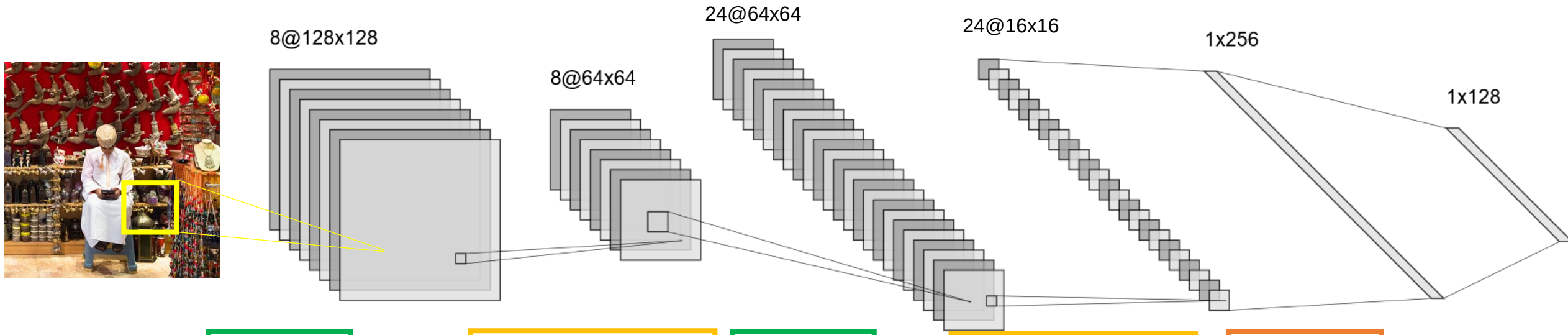
The typical architecture of a convolutional neural network



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

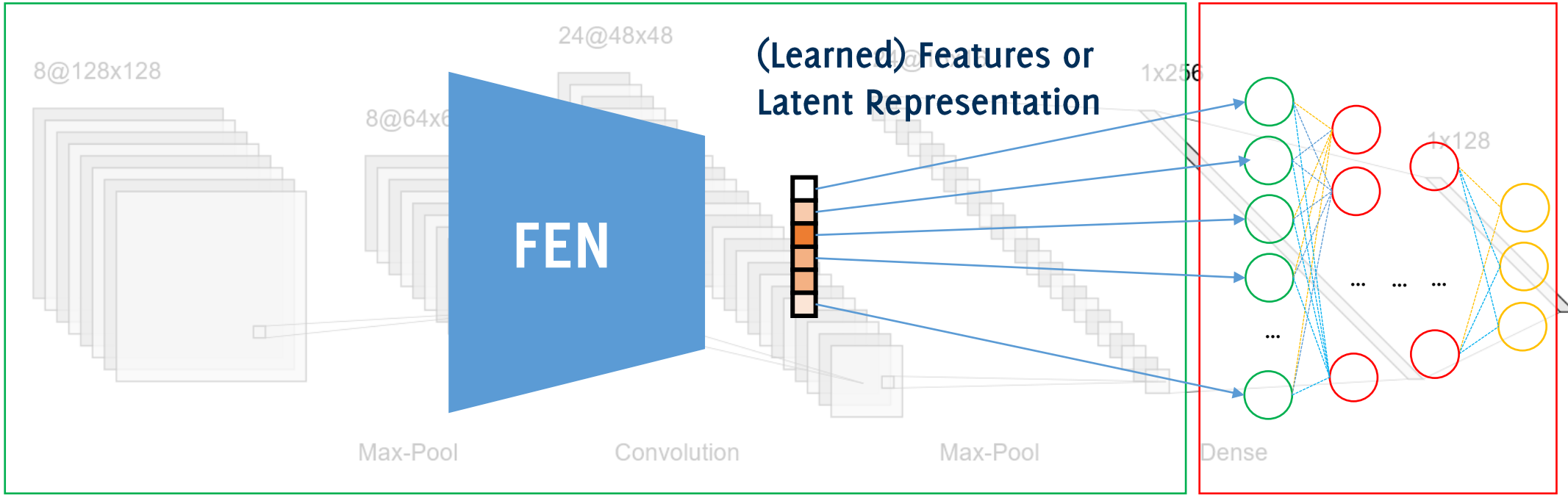
Dense

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

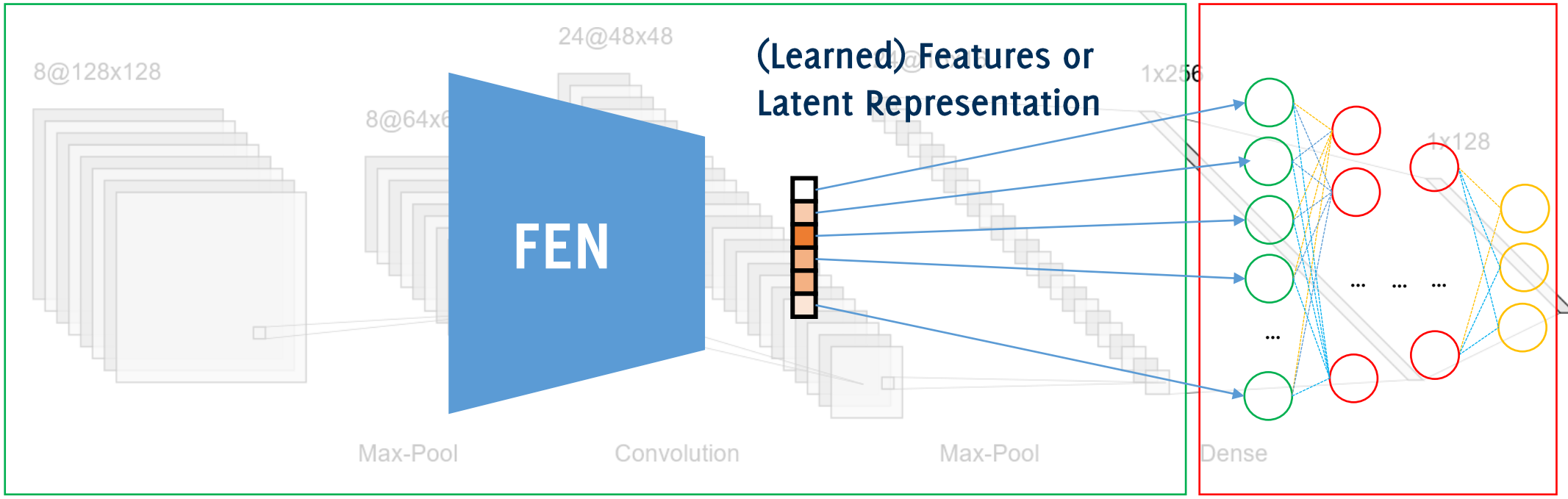


Data-Driven Feature extraction

Feature Classification

FEN: FEATURE EXTRACTION NETWORK, the convolutional block of CNN

The typical architecture of a CNN



Data-Driven Feature extraction

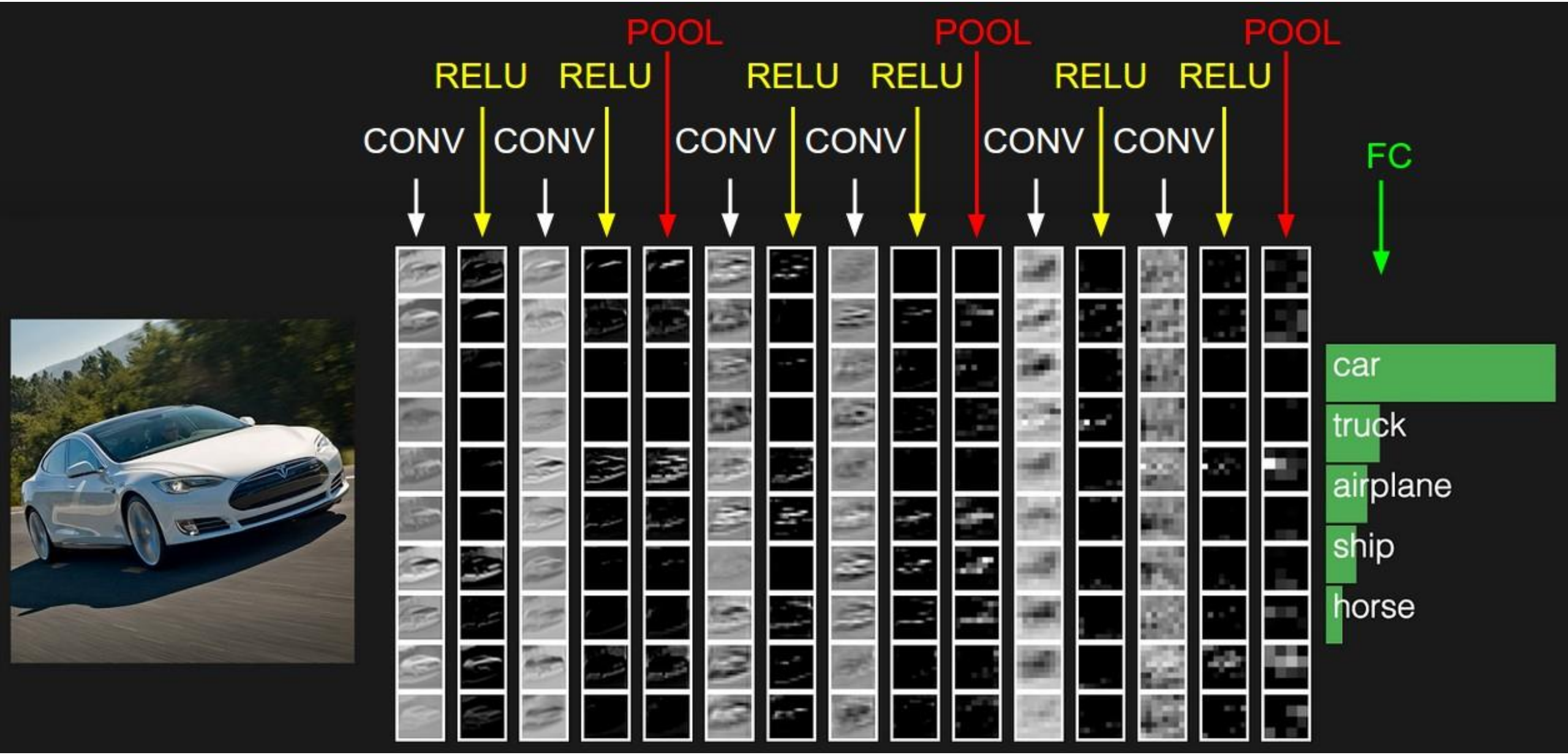
Feature Classification

Typically, to learn meaningful representations, many layers are required

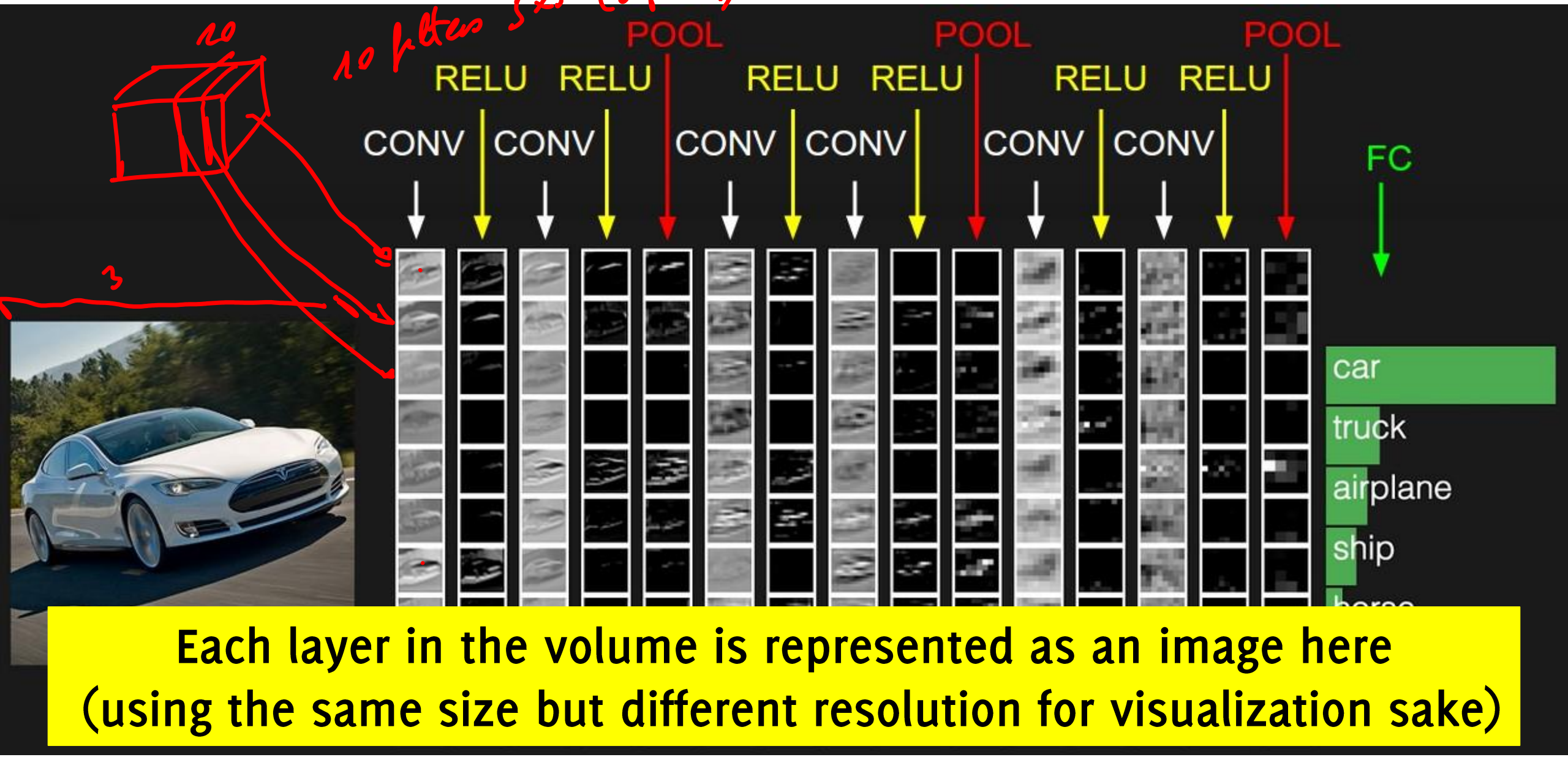
The network becomes deep

CNN «in action»

Activations in a convolutional network

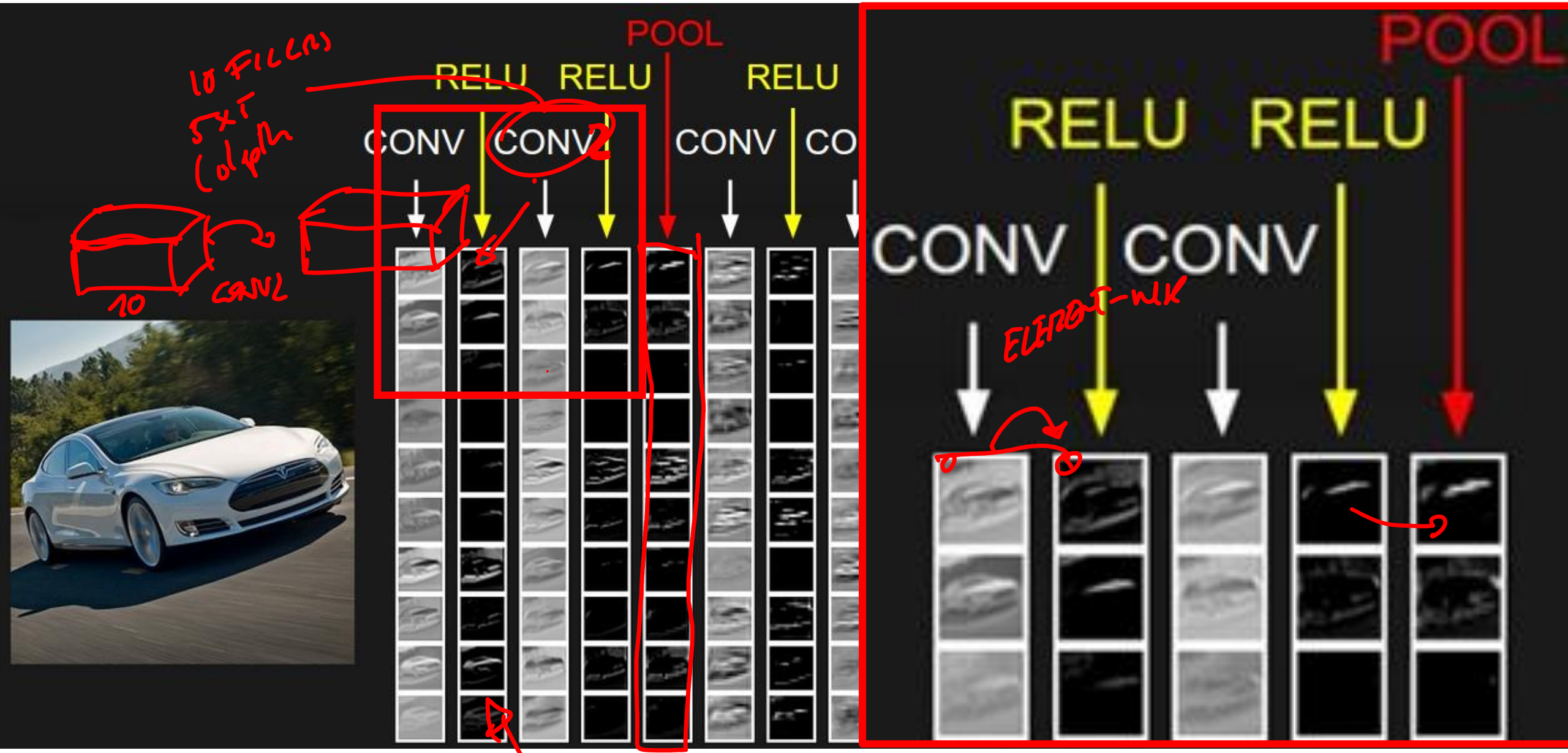


Activations in a convolutional network

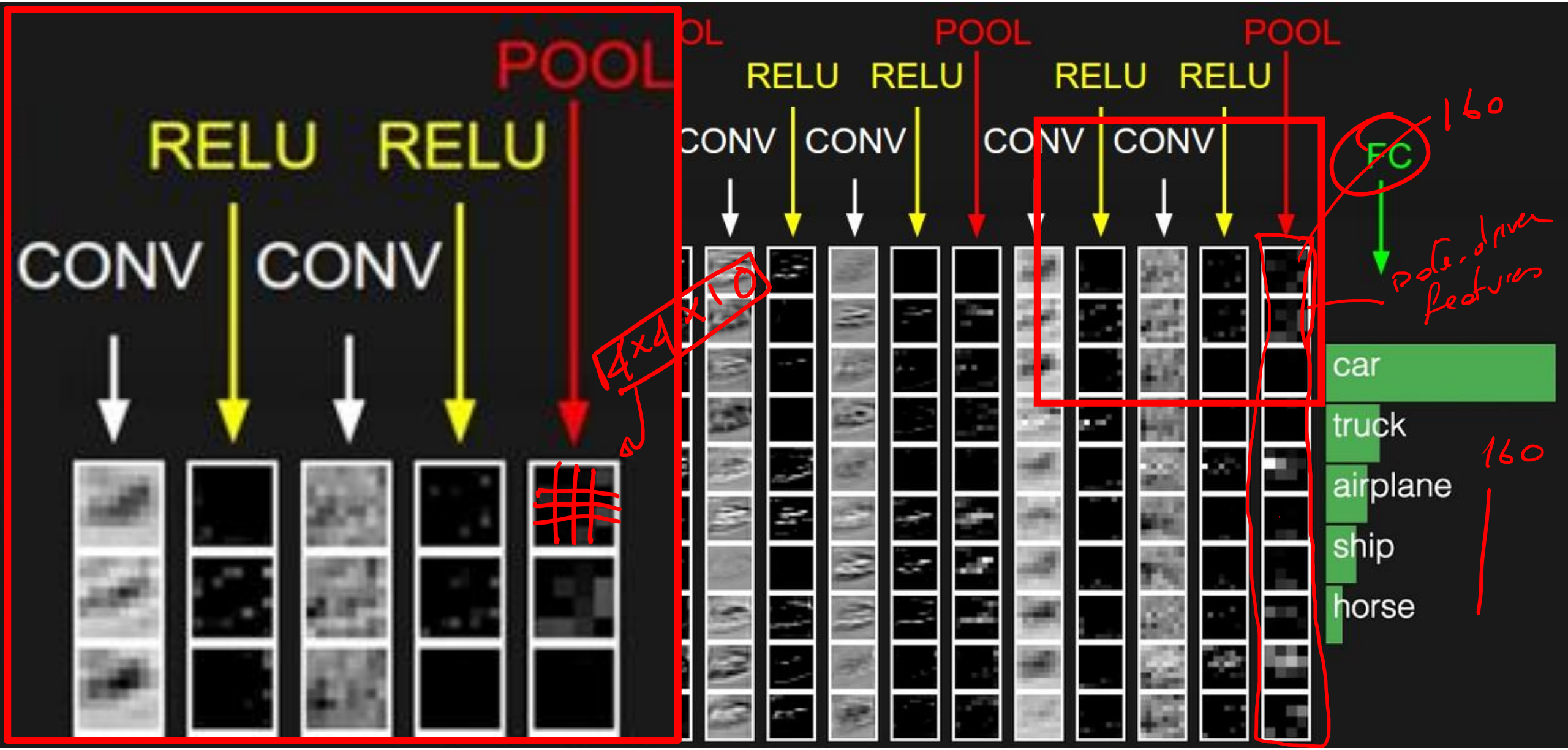


Each layer in the volume is represented as an image here
(using the same size but different resolution for visualization sake)

Activations in a convolutional network

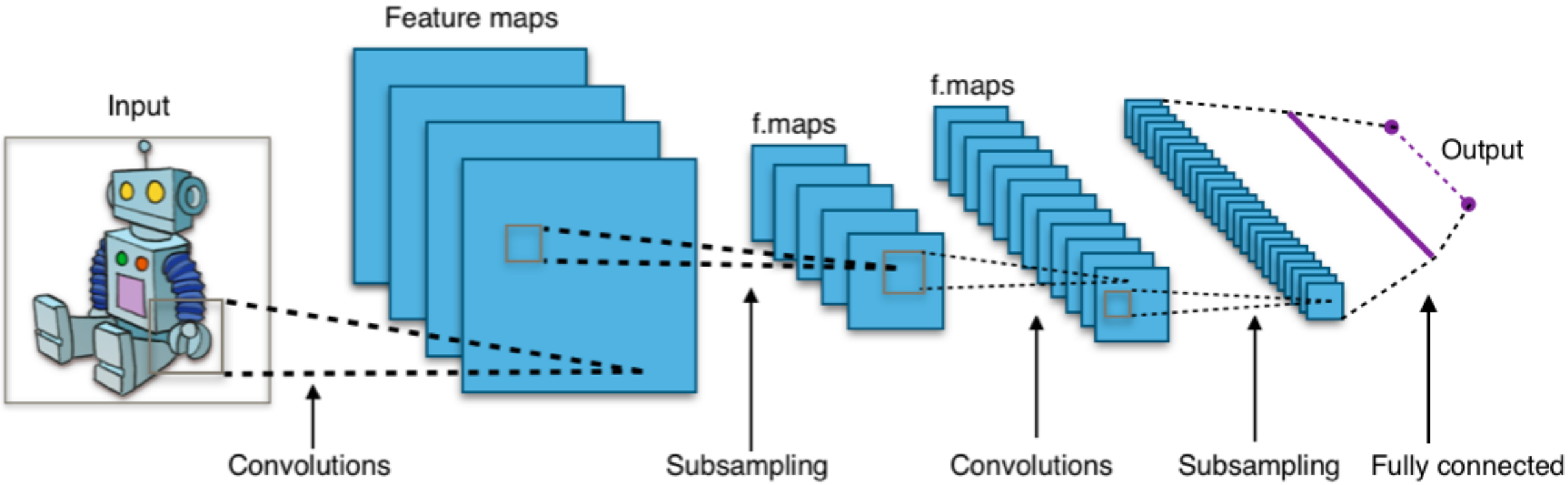


Activations in a convolutional network



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

The First CNN

Giacomo Boracchi

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.

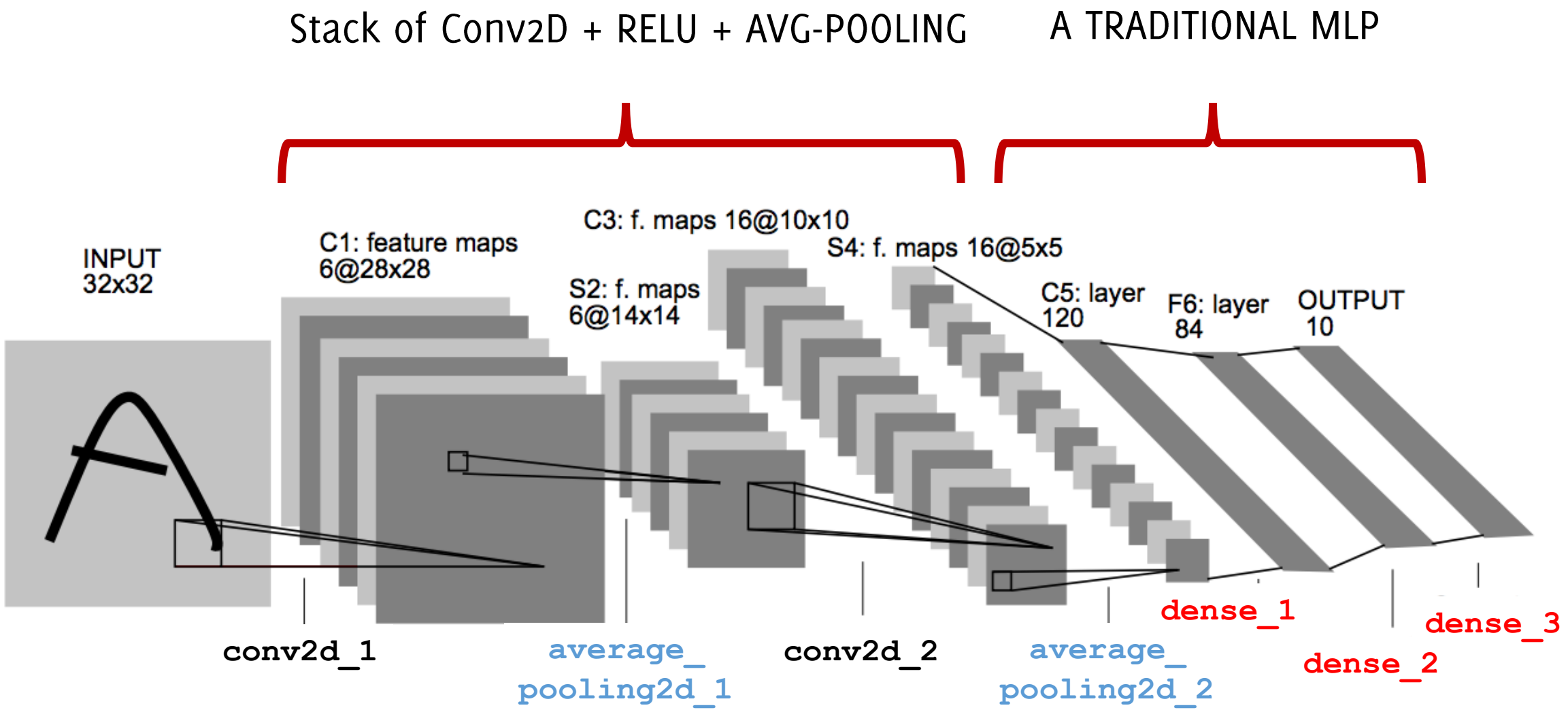
LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998)

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

LeNet-5 (1998)



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

chi

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

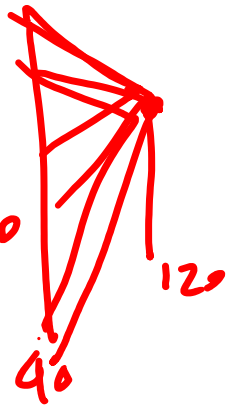
```
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, AveragePooling2D

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
_________________________________________________________________
```

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

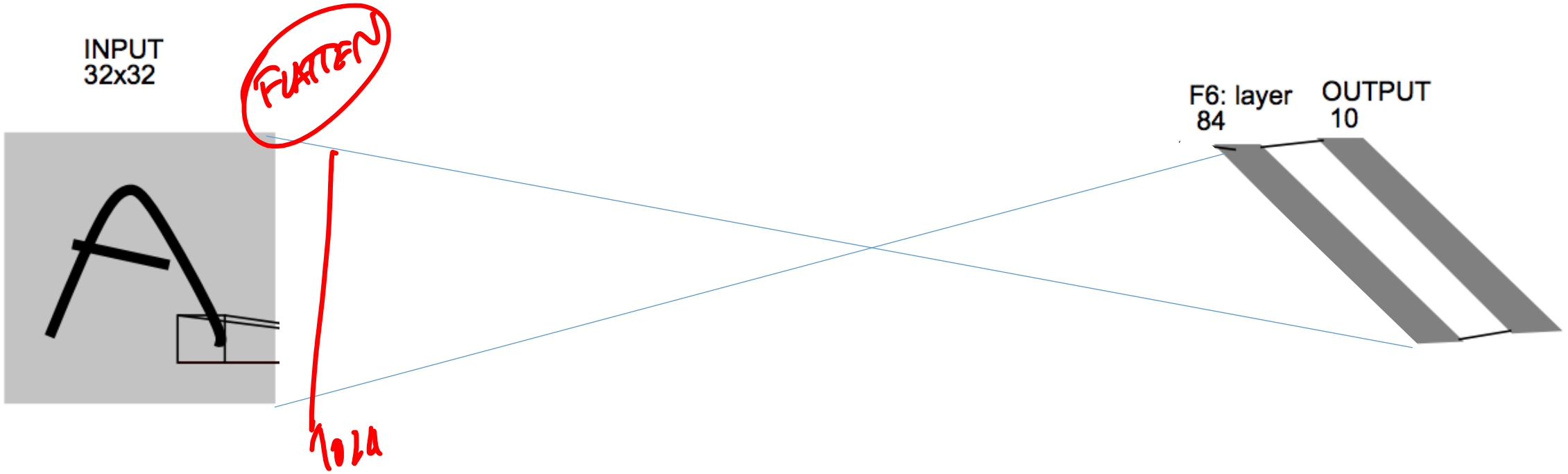
Most parameters are still in the MLP

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!

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,

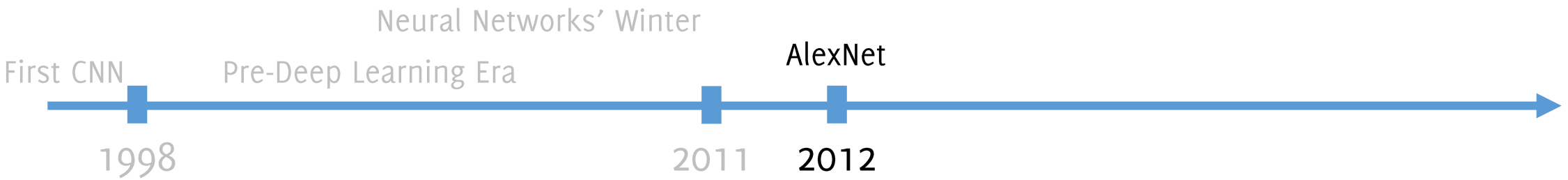
CNN: only the nr. of parameters in the filters at the first layer increases

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

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

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

Award Winning CNNs



ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012.

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012.

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

AlexNet (2012)

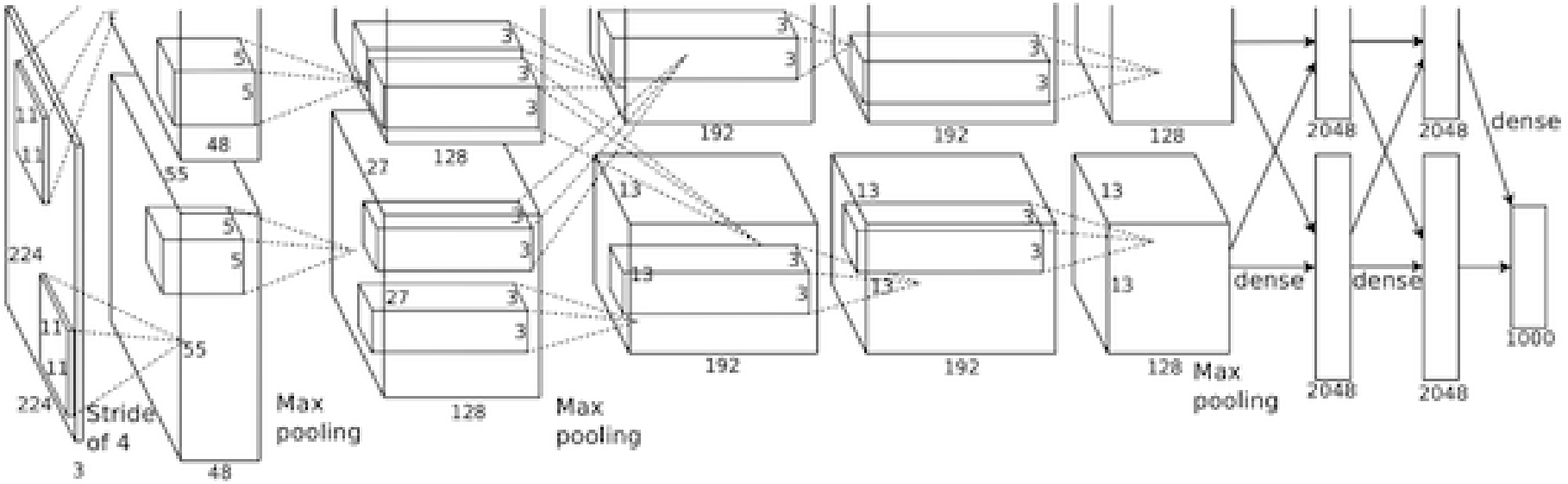
Developed by Alex Krizhevsky et al. in 2012 and won Imagenet competition

Architecture is quite similar to LeNet-5:

- 5 convolutional layers (rather large filters, 11x11, 5x5),
- 3 MLP

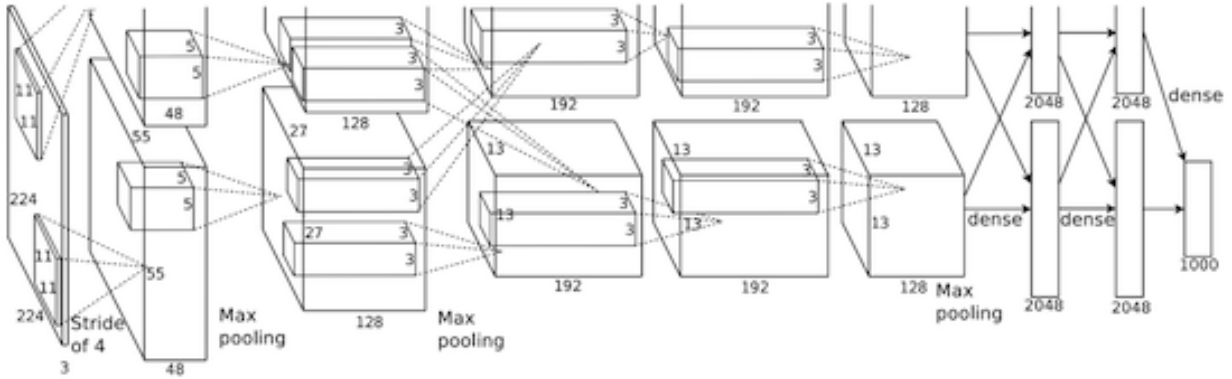
Input size 224 × 224 x 3 (the paper says 227 x 227 x 3)

Parameters: 60 million [**Conv**: 3.7million (6%), **FC**: 58.6 million (94%)]



Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012.

AlexNet (2012)



To counteract overfitting, they introduce:

- RELU (also faster than tanh)
- Dropout (0.5), weight decay and norm layers (not used anymore)
- Maxpooling

The first conv layer has 96 **11x 11 filters, stride 4.**

The output are **two volumes of 55 x 55 x 48 separated over two GTX 580 GPUs** (1.5GB each GPU, 90 epochs, 5/6 days to train).

Most **connections are among** feature maps **of the same GPU**, which will be mixed at the last layer.

Won the ImageNet challenge in 2012

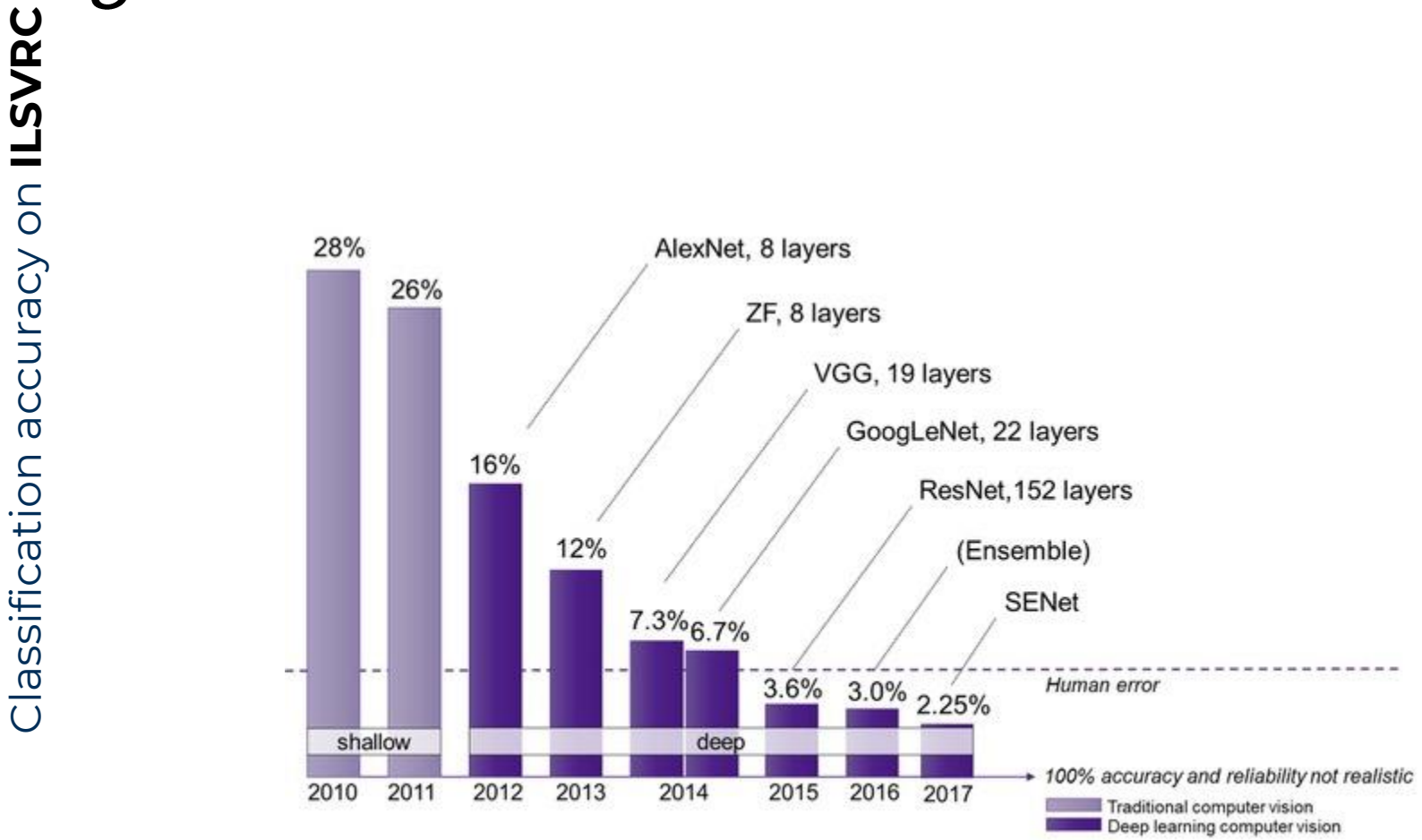
At the end they also trained an **ensemble of 7 models** to drop error: 18.2%->15.4%

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012.

A Breaktrough in Image Classification

Giacomo Boracchi

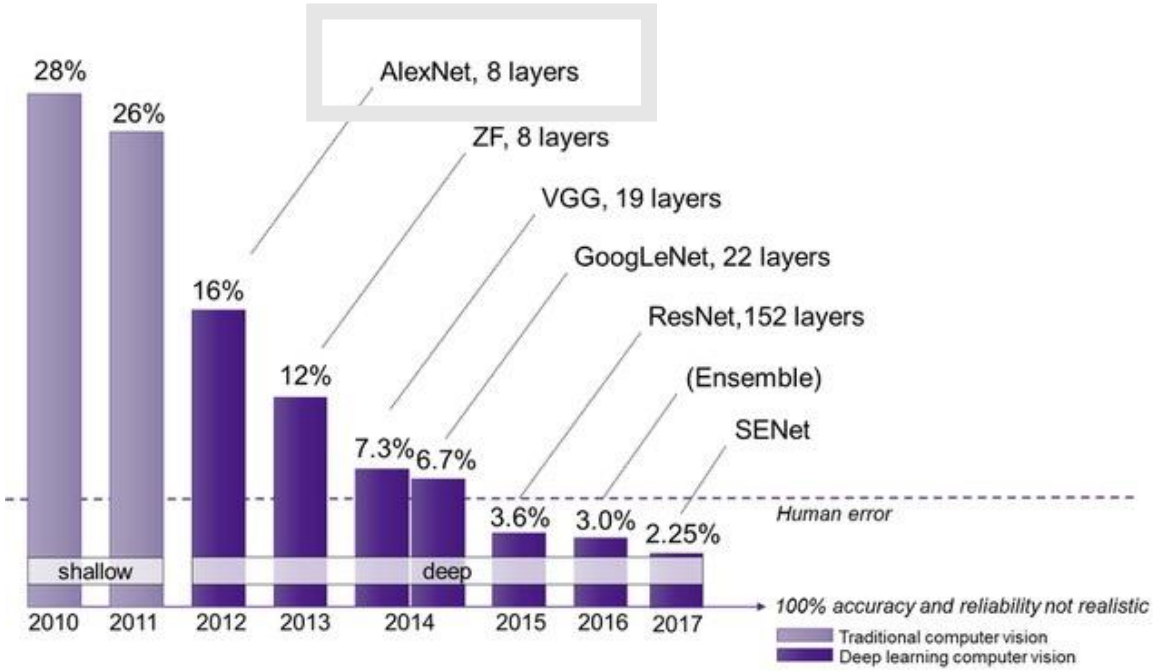
The Impact of Deep Learning in Visual Recognition

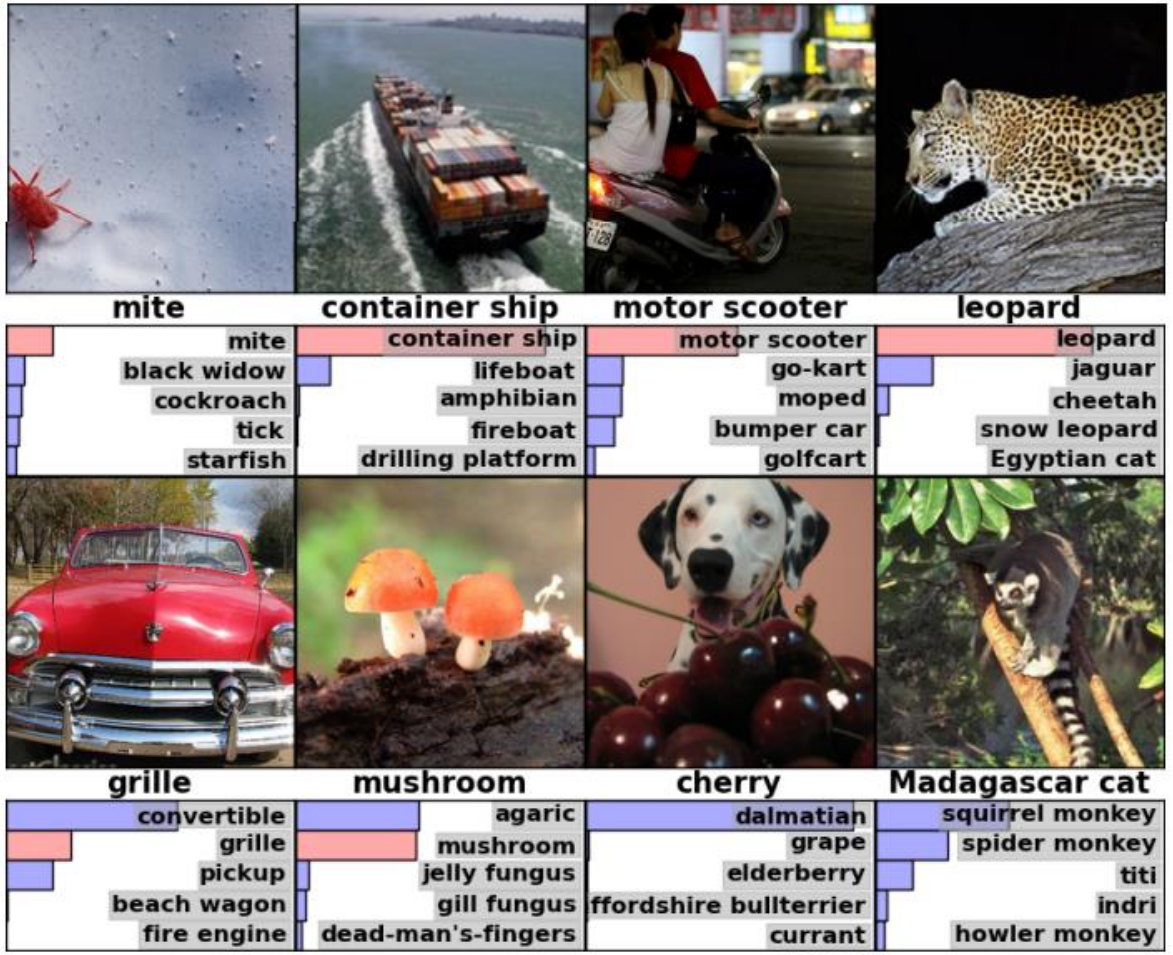


Many layers!

ILSVRC: ImageNet Large Scale Visual Recognition Challenge

AlexNet / Imagenet Images





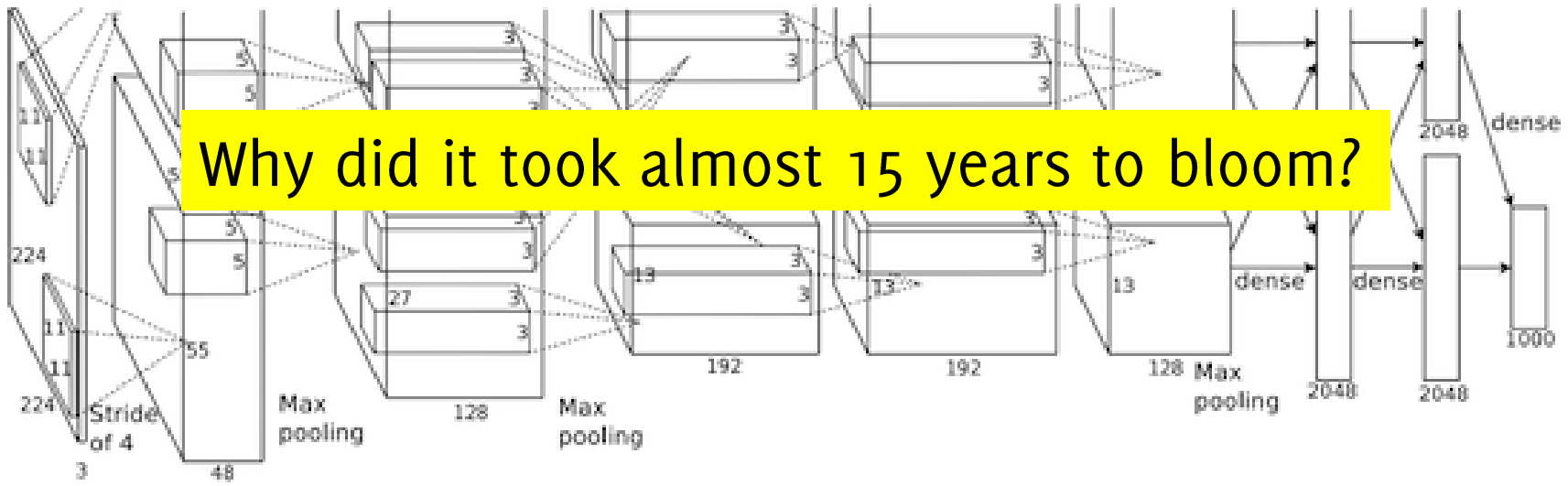
Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems* 25 (2012).

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

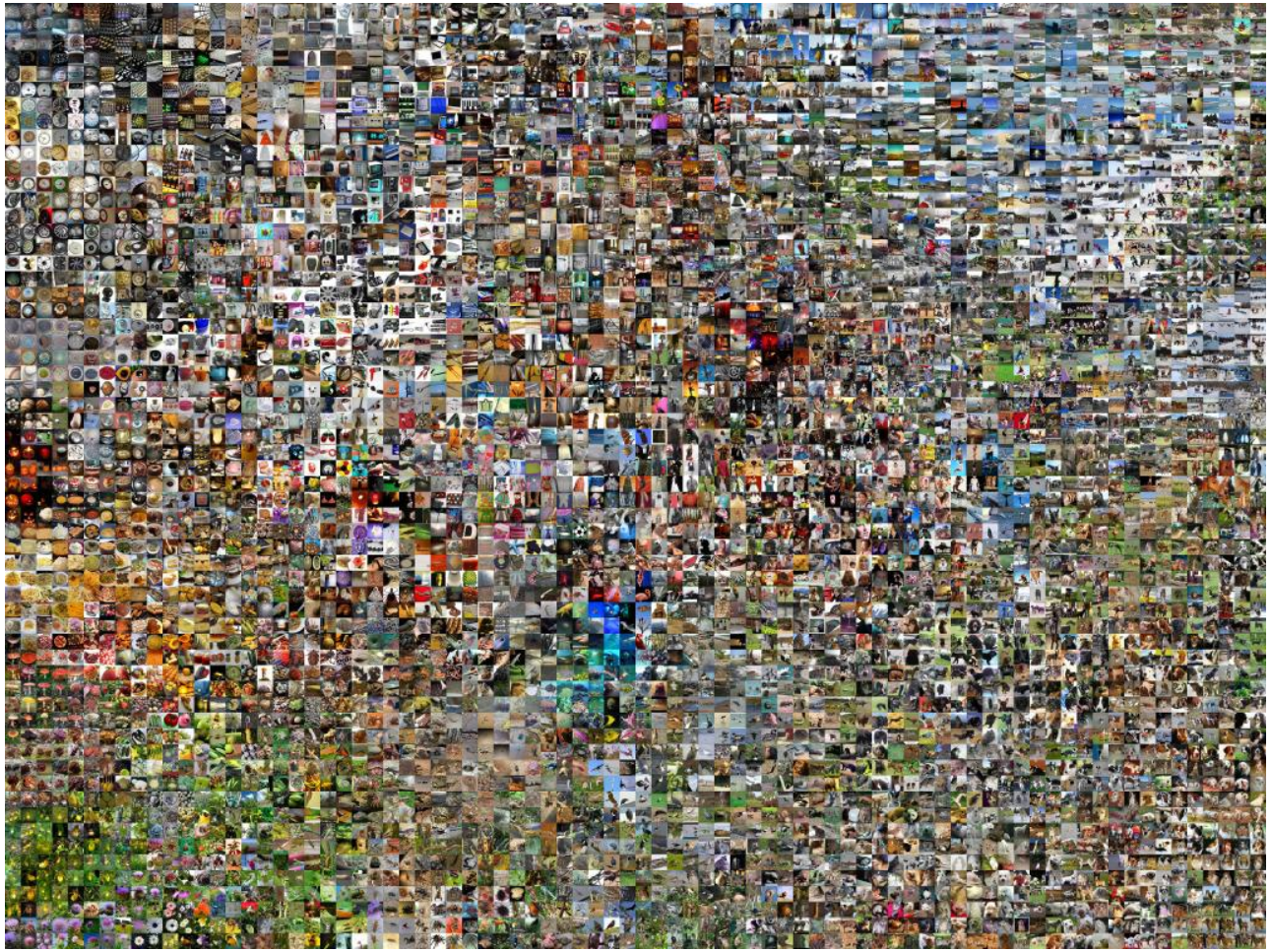


Why did it took almost 15 years to bloom?

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012.

How was this possible?

Large Collections of Annotated Data



The ImageNet project is a large visual database designed for use in visual object recognition software research. ***More than 14 million images*** *have been hand-annotated by the project to indicate what objects are pictured and in at least one million of the images, bounding boxes are also provided.[3]* ***ImageNet contains more than 20,000 categories***

From Wikipedia October 2021

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, ImageNet: A Large-Scale Hierarchical Image Database. *CVPR, 2009*.

Parallel Computing Architectures

And more recently…. Software libraries

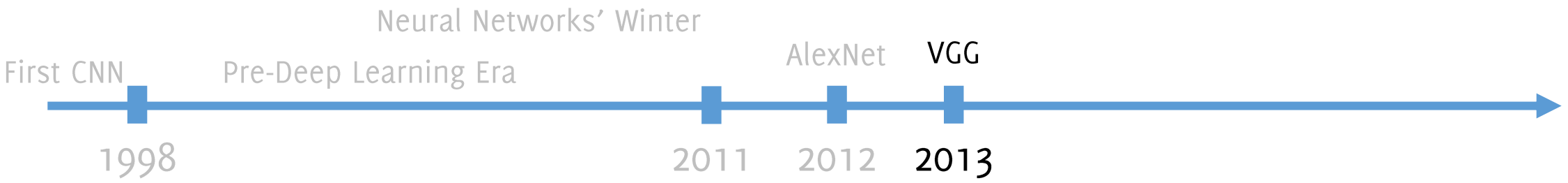
TensorFlow

PyTorch

Google LLC, Public domain, via Wikimedia Commons

PyTorch, BSD <http://opensource.org/licenses/bsd-license.php>, via Wikimedia Commons

VGG: going deeper!



VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

Karen Simonyan* **& Andrew Zisserman+**
Visual Geometry Group, Department of Engineering Science, University of Oxford
{karen,az}@robots.ox.ac.uk

ABSTRACT

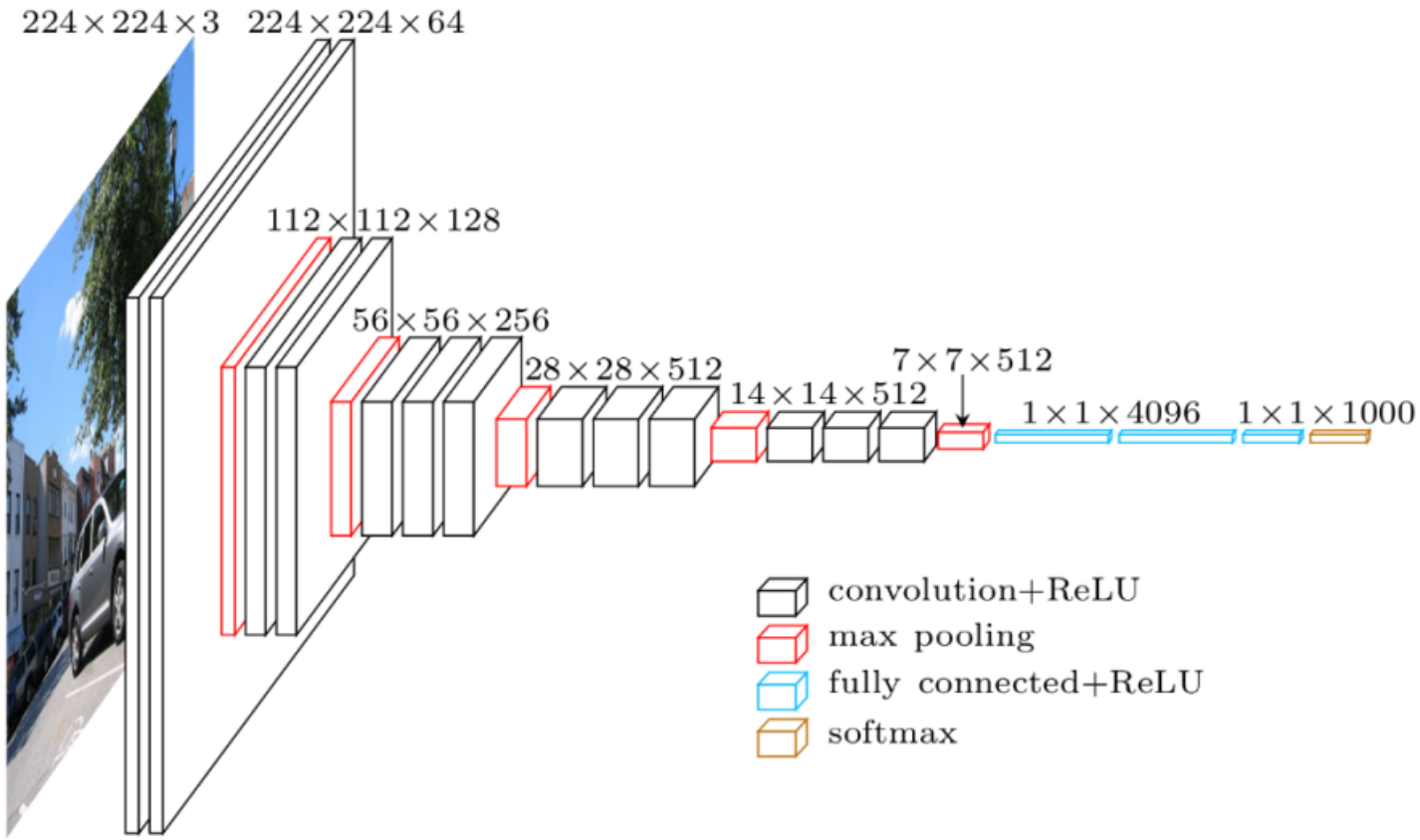
In this work we investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. Our main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small (3×3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19 weight layers. These findings were the basis of our ImageNet Challenge 2014 submission, where our team secured the first and the second places in the localisation and classification tracks respectively. We also show that our representations generalise well to other datasets, where they achieve state-of-the-art results. We have made our two best-performing ConvNet models publicly available to facilitate further research on the use of deep visual representations in computer vision.

Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." ICLR (2015)

VGG16 (2014)

The VGG16, introduced in 2014 is a deeper variant of the AlexNet convolutional structure. Smaller filters are used and the network is deeper

Parameters: 138 million [Conv: 11%, FC: 89%]



Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." ICLR (2015)

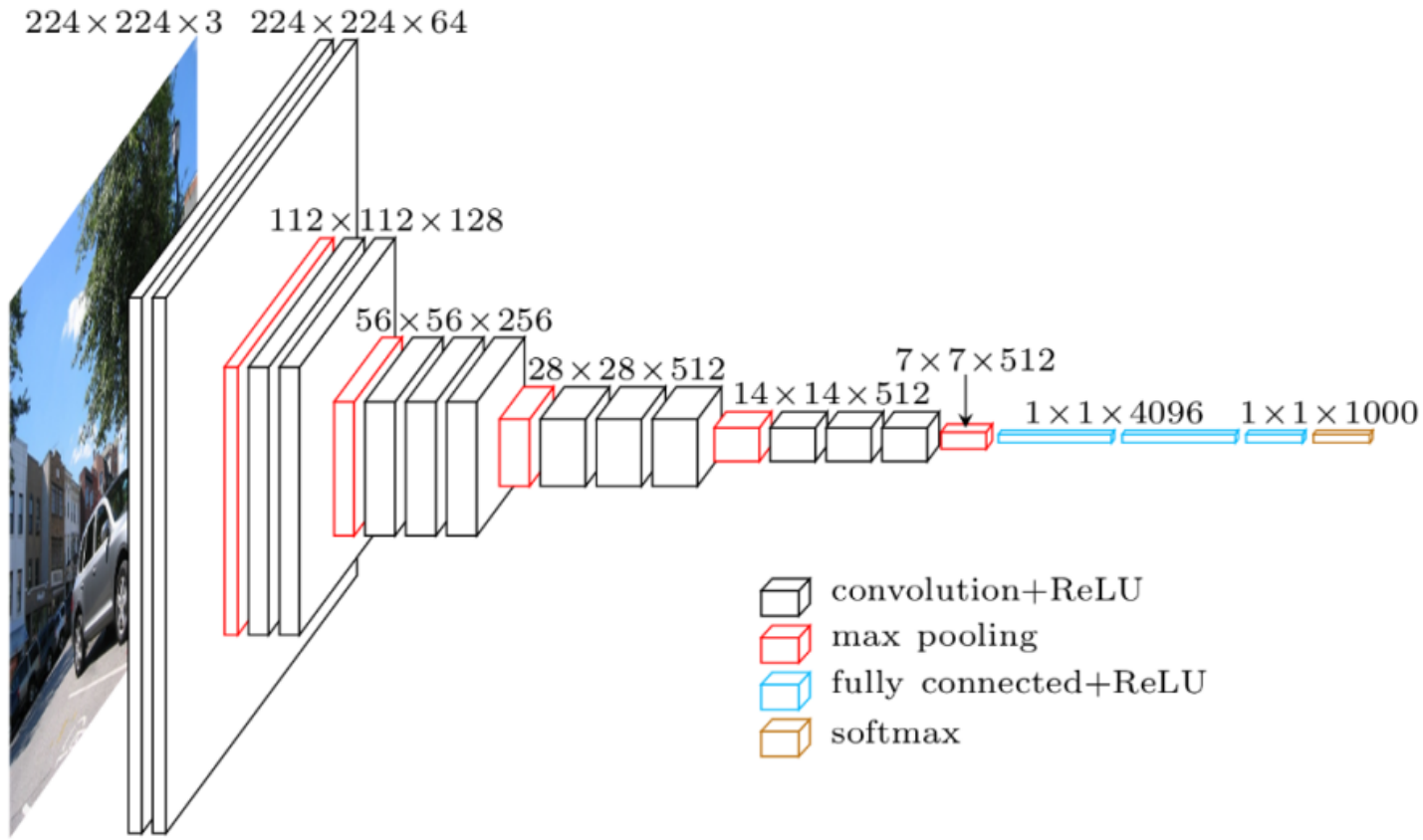
VGG16 (2014)

The VGG16, introduced in 2014 is a deeper variant of the AlexNet convolutional structure. Smaller filters are used and the network is deeper

Parameters: 138 million [Conv: 11%, FC: 89%]

These architecture **won the first place places (localization) and the second place (classification) tracks in ImageNet Challenge 2014**

Input size 224 × 224 x 3



Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." ICLR (2015)

VGG16 (2014): Smaller Filter, Deeper Network

The paper actually present a thorough **study on the role of network depth**.

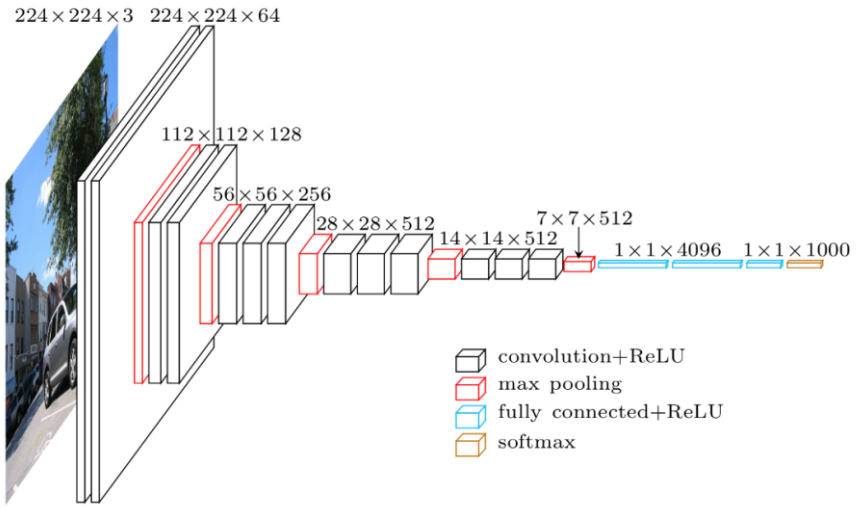
[…]Fix other parameters of the architecture, and steadily increase the depth of the network by adding more convolutional layers, which is feasible due ***to the use of very small (3. × 3) convolution filters in all layers.***

Idea: Multiple 3×3 convolution in a sequence achieve large receptive fields with:

- less parameters
- more nonlinearities

than larger filters in a single layer

	3 layers 3x3	**1 layer 7x7**
Receptive field	7x7	7x7
Nr of parameters	3 x 3 x 3 = 27	49
Nr of nonlinearities	3	1



Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." ICLR (2015)

VGG16

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
[…]		

Layer (type)	Output Shape	Param #
[…]		
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0

VGG16

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
[…]		

Layer (type)	Output Shape	Param #
[…]		
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0

Many convolutional blocks without maxpooling

VGG16

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
[…]		

Layer (type)	Output Shape	Param #
[…]		
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0

Most parameters in FC layers : **123,642,856**

Giacomo Boracchi

VGG16

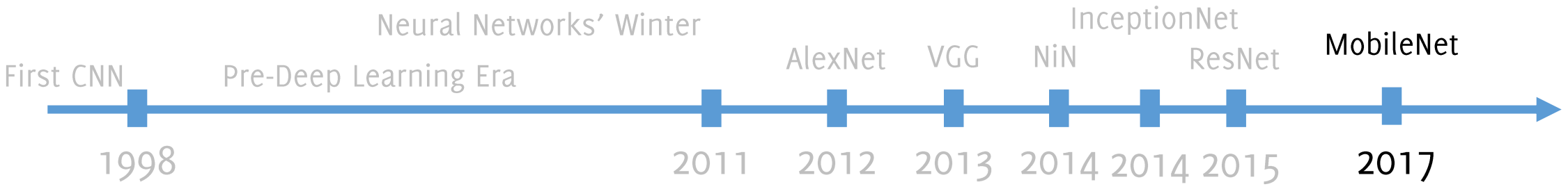
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)		
block2_conv2 (Conv2D)		
block2_pool (MaxPooling2D)		
block3_conv1 (Conv2D)		
block3_conv2 (Conv2D)		
block3_conv3 (Conv2D)		
block3_pool (MaxPooling2D)		
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
[…]		

Layer (type)	Output Shape	Param #
[…]		
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
	, 14, 512)	2359808
	7, 512)	0
	088)	0
	96)	102764544
	96)	16781312
	00)	4097000

Trainable params: 138,357,544
Non-trainable params: 0

High memory request, about 100MB per image ($224 \times 224 \times 3$) to be stored in all the activation maps, only for the forward pass. During training, with the backward pass it's about twice

MobileNet: Reducing Computational Costs



MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

Andrew G. Howard Menglong Zhu Bo Chen Dmitry Kalenichenko
Weijun Wang Tobias Weyand Marco Andreetto Hartwig Adam

Google Inc.

{howarda,menglong,bochen,dkalenichenko,weijunw,weyand,anm,hadam}@google.com

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

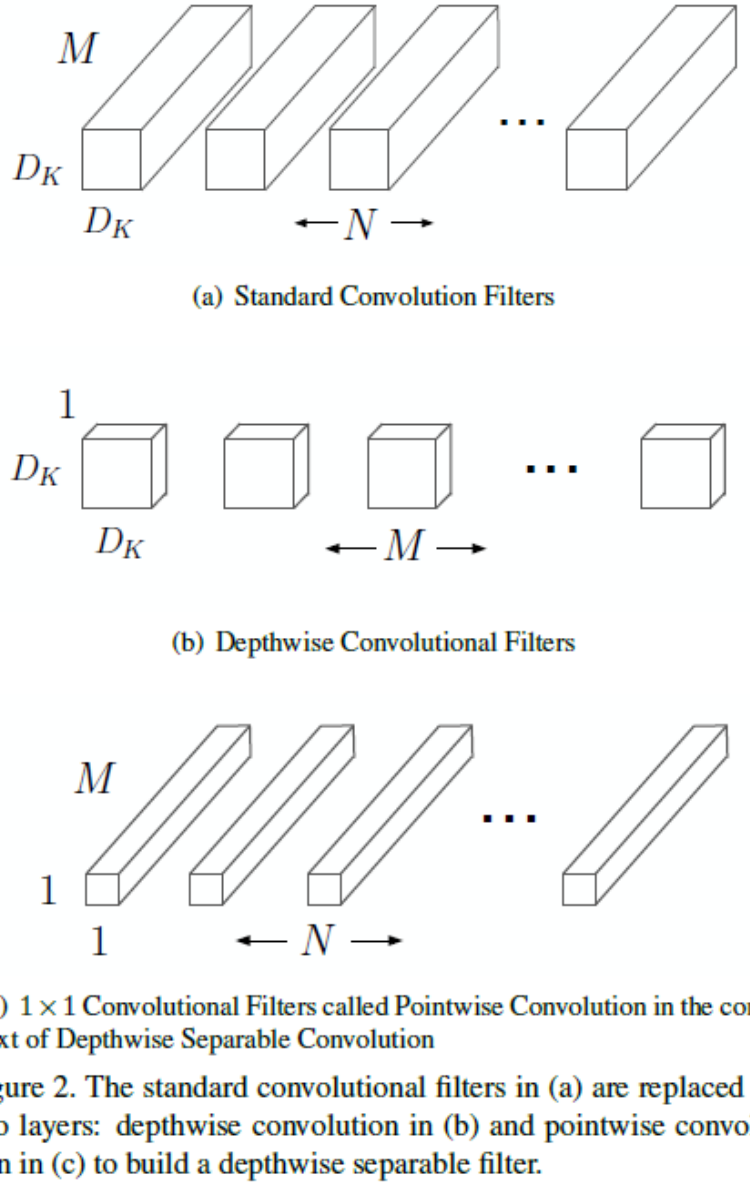
Depth-wise Separable Convolutions

Goal: reduce the number of parameters and of operations, to embed networks in mobile application

- conv2D layers have quite a few parameters
- conv2D layers mixes all the channels

In contrast, separable convolutoins

- Perform first a depth-wise convolution: a spatial only operation, without mixing the components
- Point-wise convolution that mixes the resulting channes without considering the spatial dimension.



(a) Standard Convolution Filters

(b) Depthwise Convolutional Filters

(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 2. The standard convolutional filters in (a) are replaced by two layers: depthwise convolution in (b) and pointwise convolution in (c) to build a depthwise separable filter.

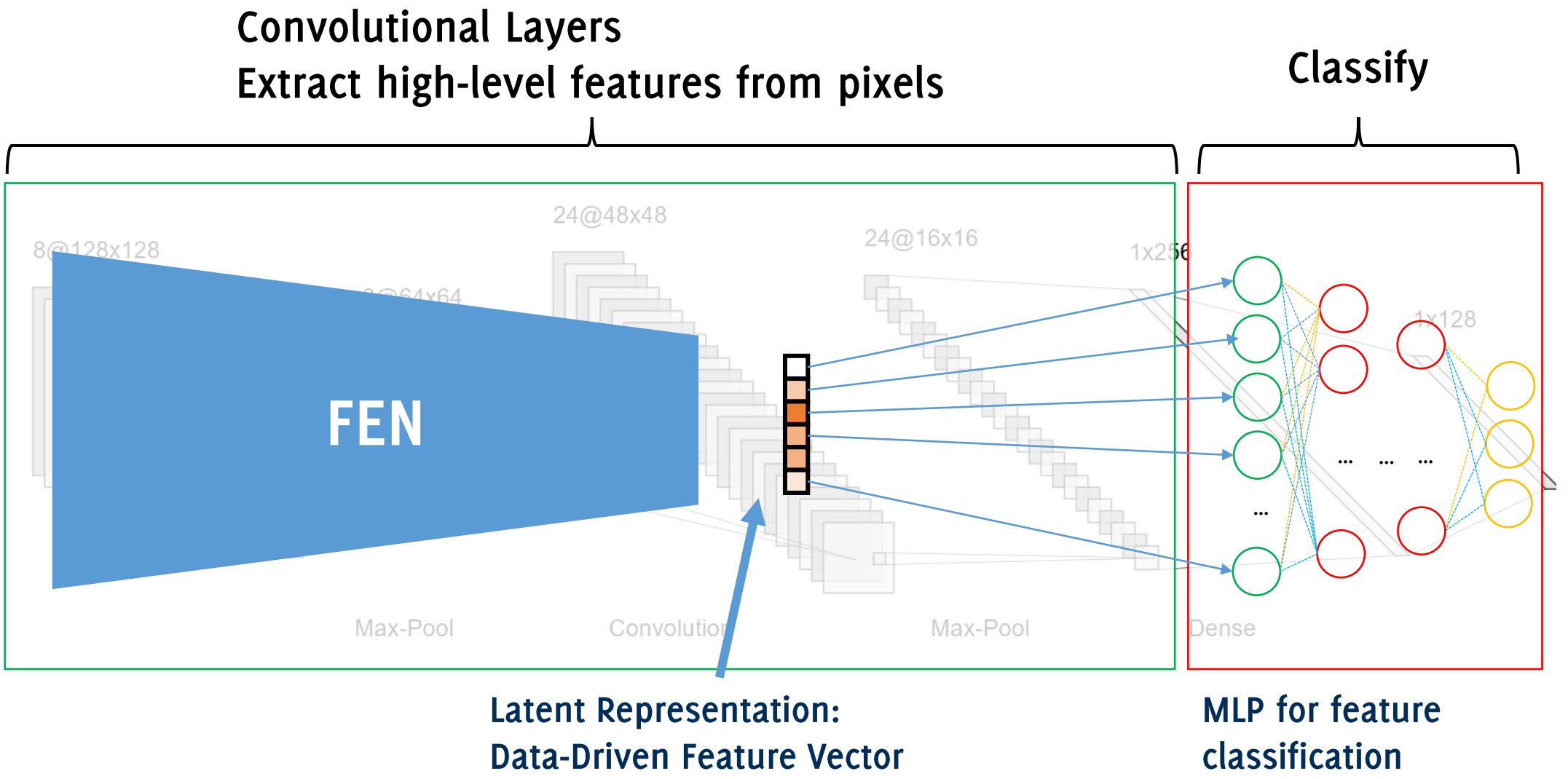
Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

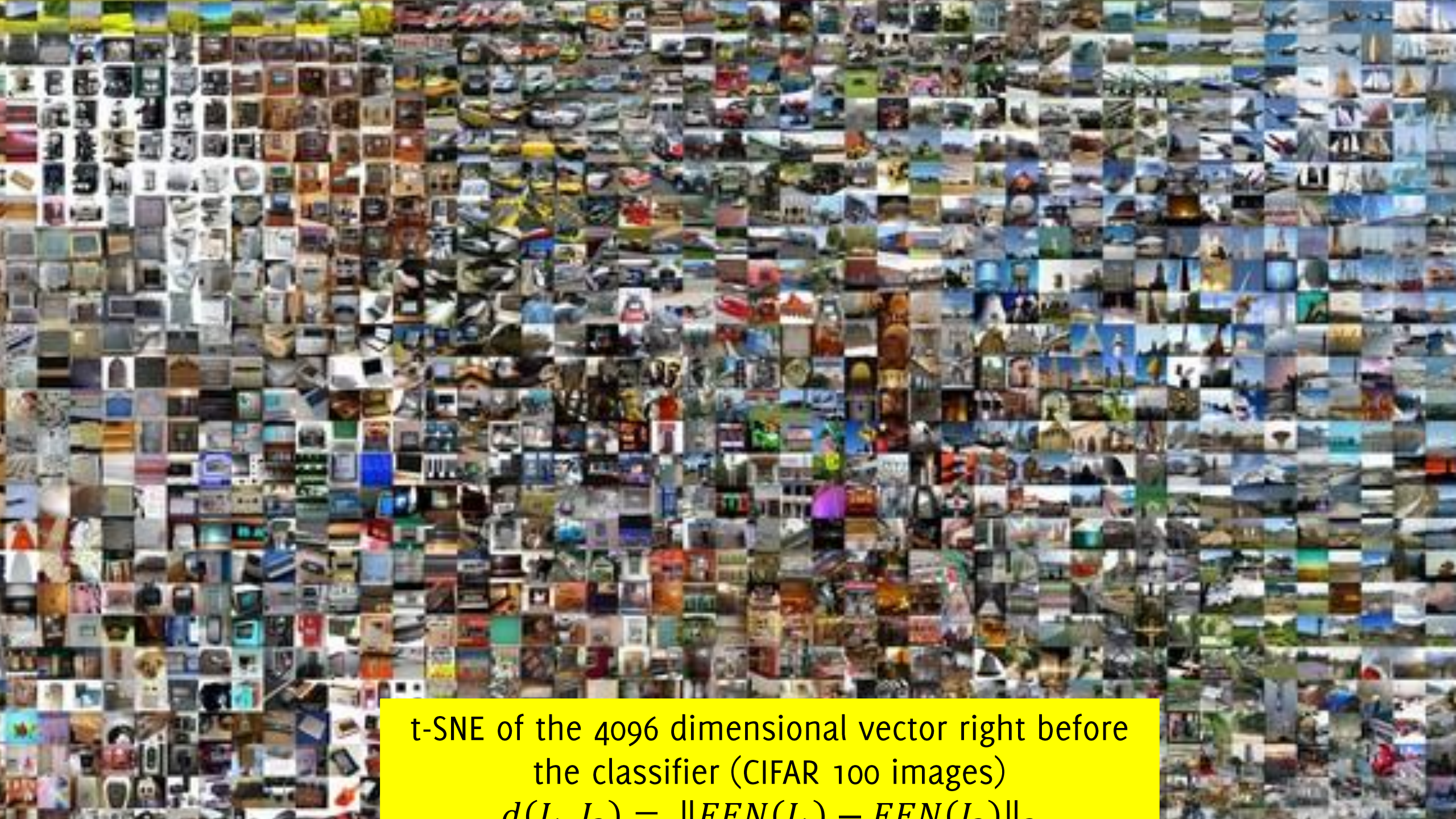
Latent representation in CNNs

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

Giacomo Boracchi

The typical architecture of a CNN





t-SNE of the 4096 dimensional vector right before the classifier (CIFAR 100 images)

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


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

t-SNE representation using ℓ_2 distance

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

Limited Amount of Data: Data Augmentation

Training a CNN with Limited Aumont of Data

Giacomo Boracchi

The need of data

Deep learning models are very data hungry.

… watch out: each image in the training set have to be annotated!

How to train a deep learning model with a few training images?

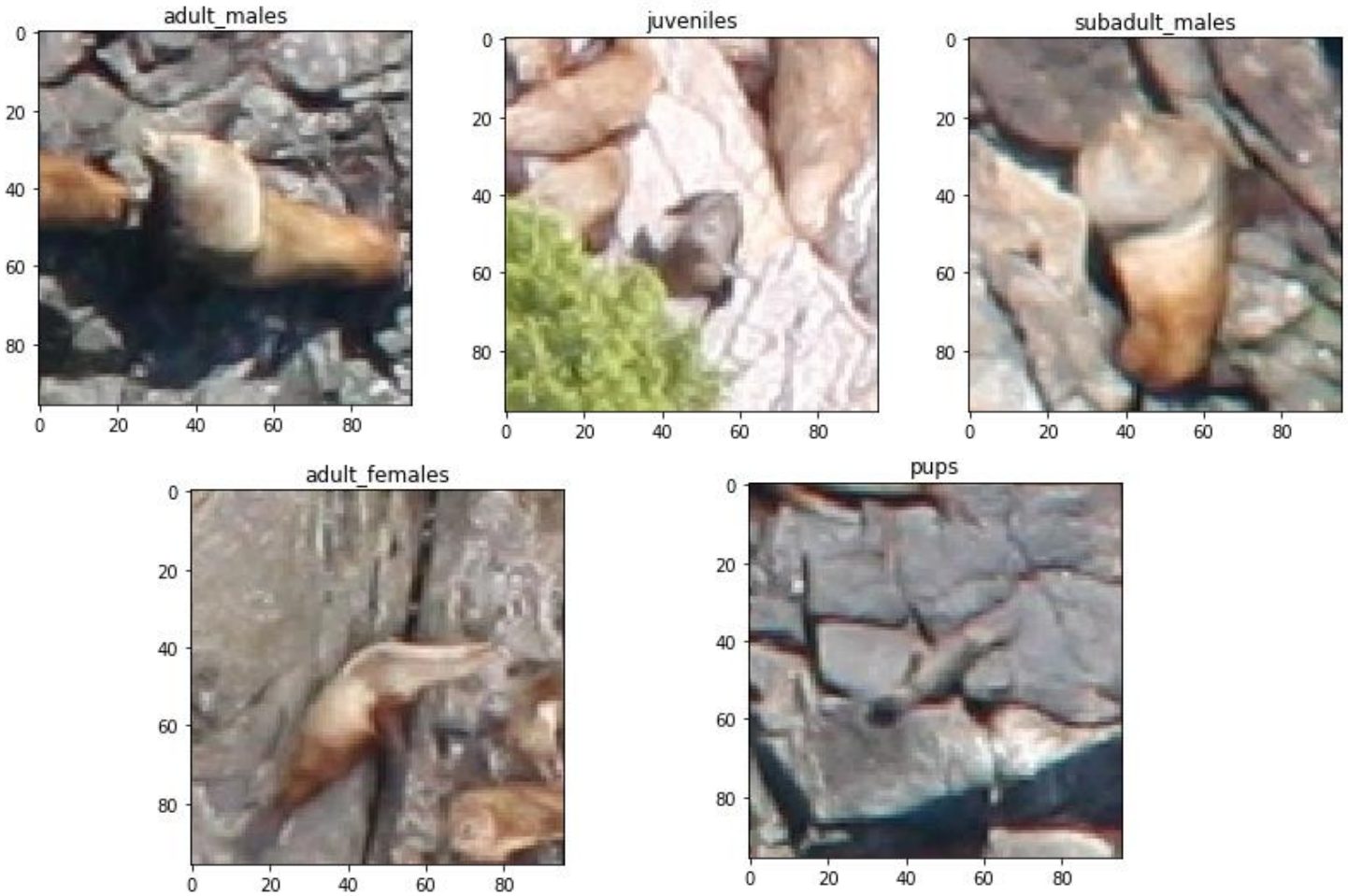
- Data augmentation
- Transfer Learning

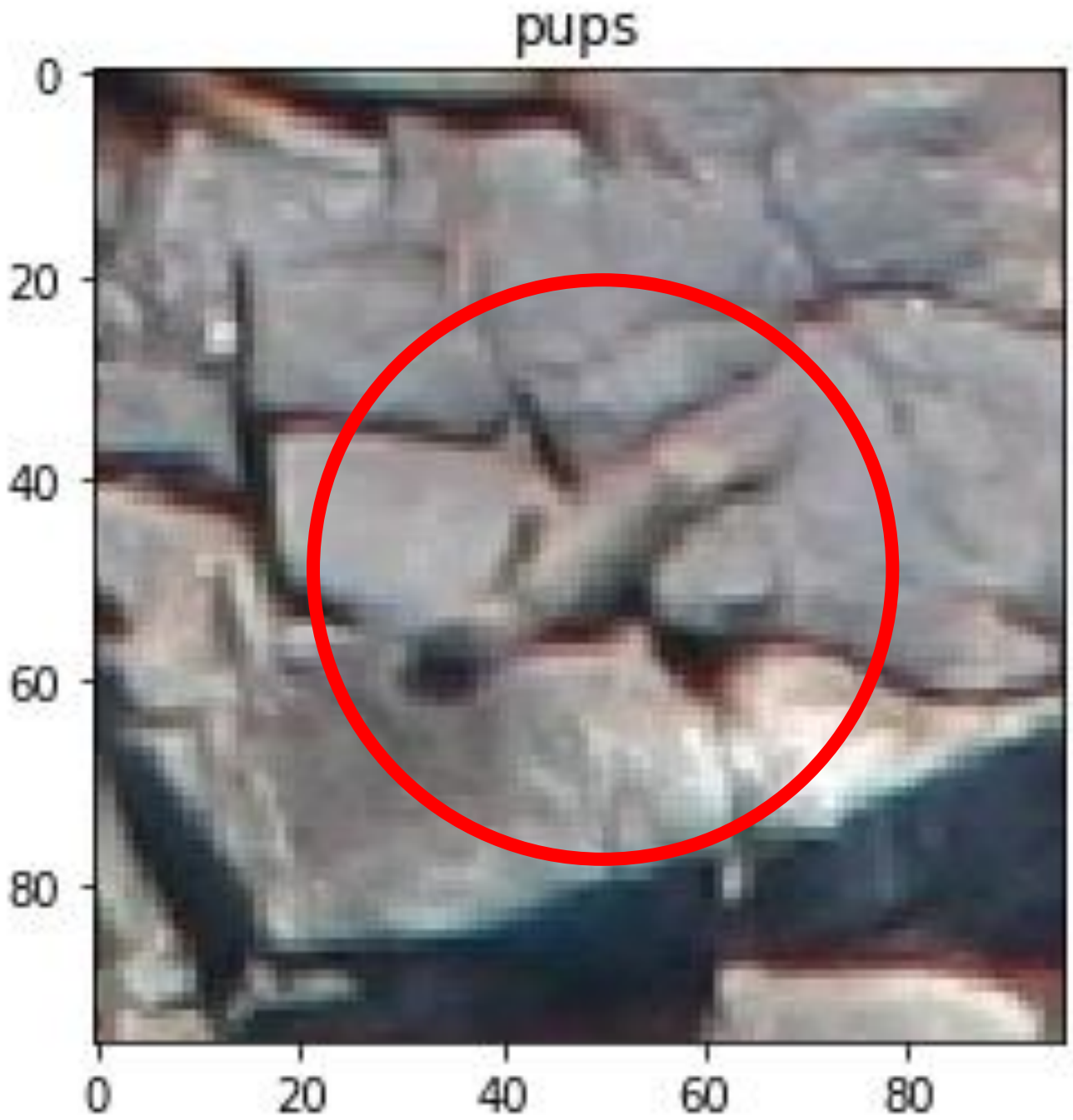

Canada
Aleutian Islands
Stati Uniti
Messico
Steller sea lions in the western Aleutian Islands have declined 94% in the last 30 years.
3D


Kaggle in 2017 have opened a competition to develop algorithms which accurately count the number of sea lions in aerial photographs
Credits Yinan Zhou
https://github.com/marioZYN/FC-CNN-Demo

The Challenge

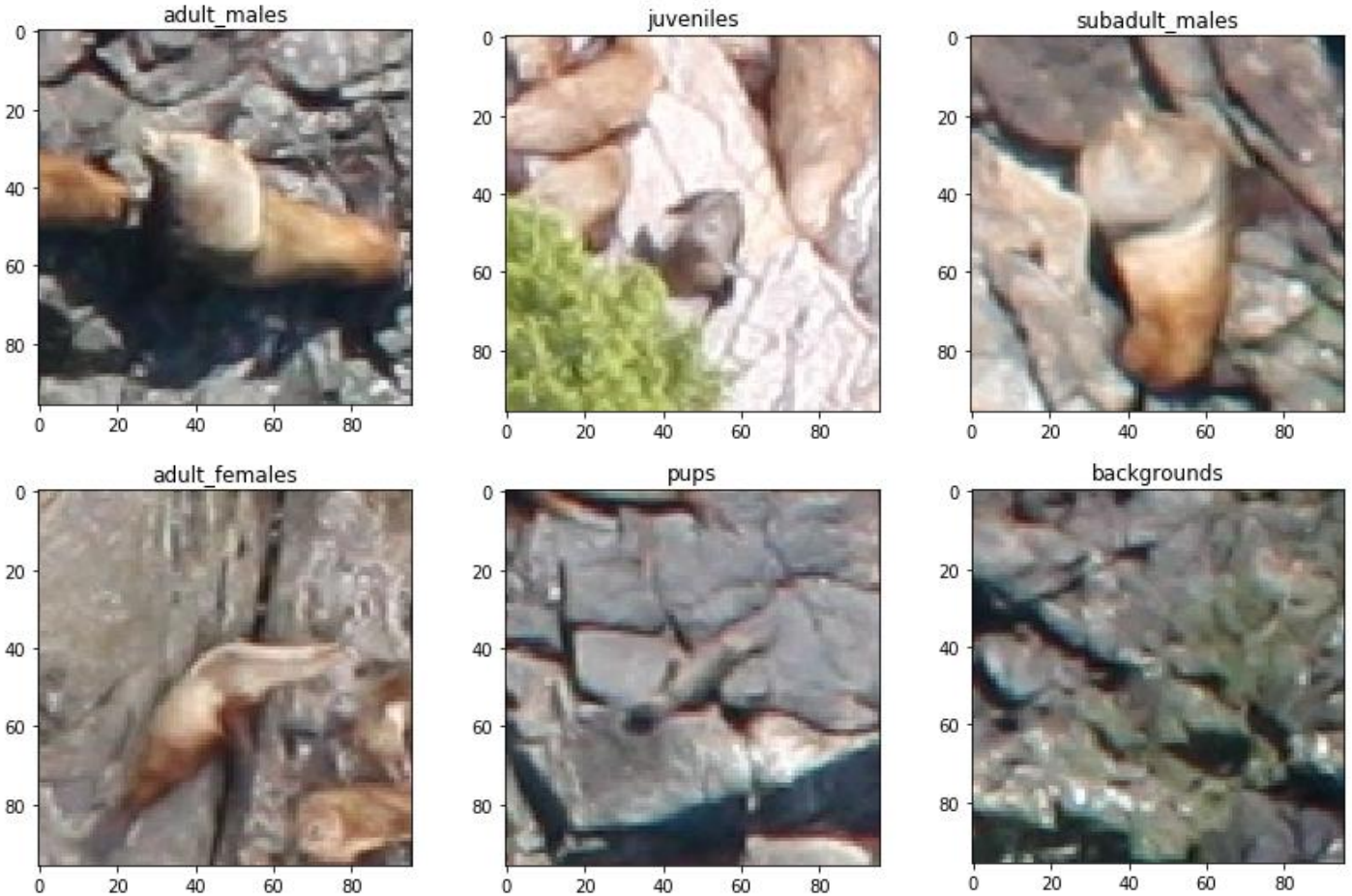
In very large aerial images (≈ 5K x 4K) shot by drones, automatically count the number of sealions per each category



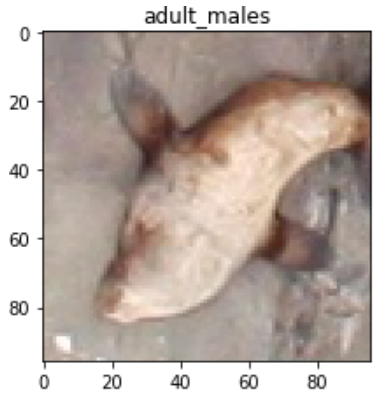

pups
0
20
40
60
80
0
20
40
60
80

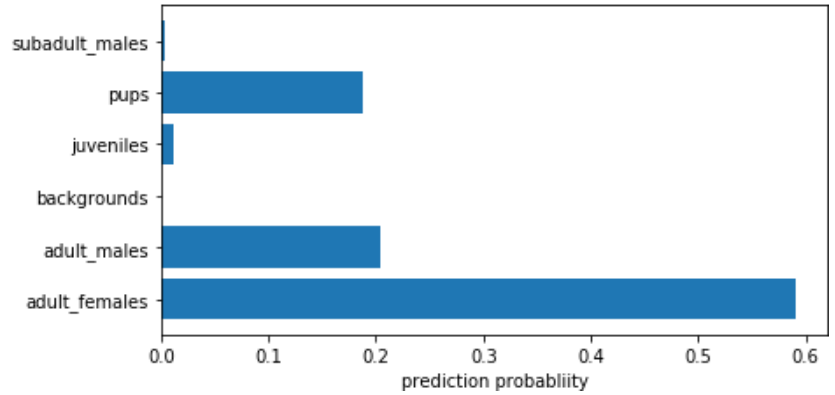
The Challenge

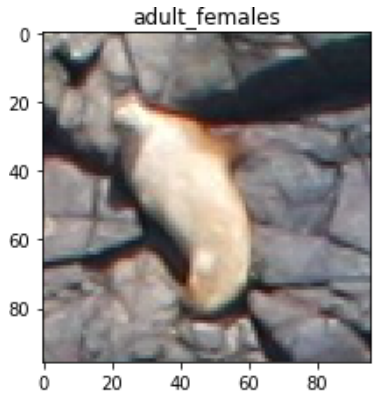
This problem can be naively casted in a patch-by-patch 6-class classification problem, where we include also background

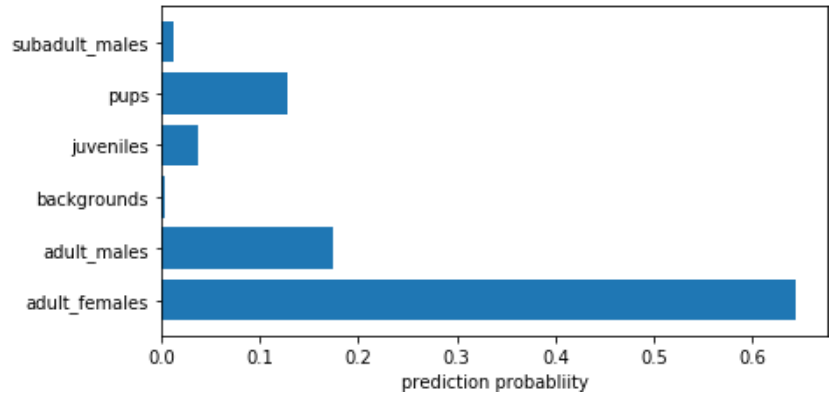


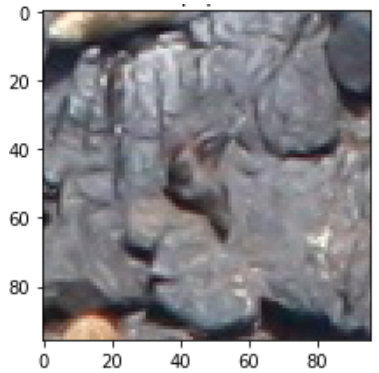
An Example of CNN predictions

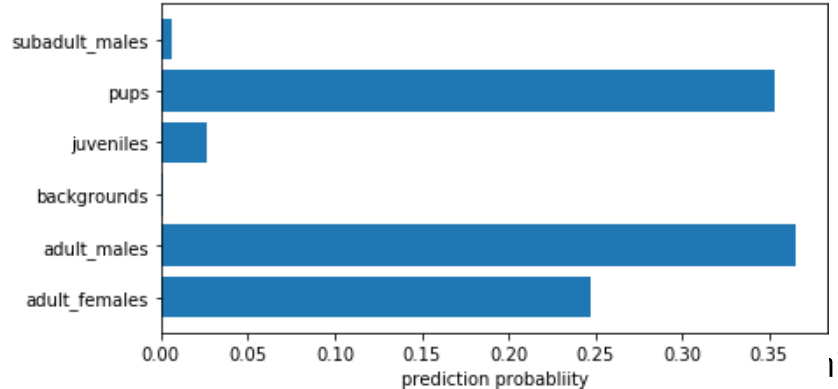










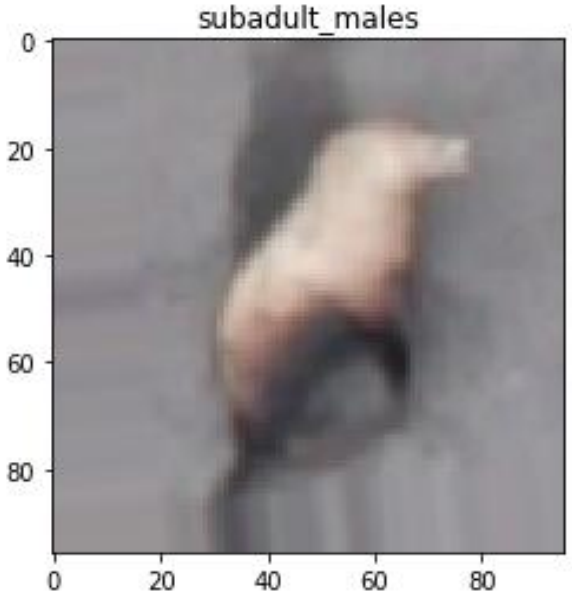


Credits Yinan Zhou
https://github.com/marioZYN/FC-CNN-Demo

Data Augmentation

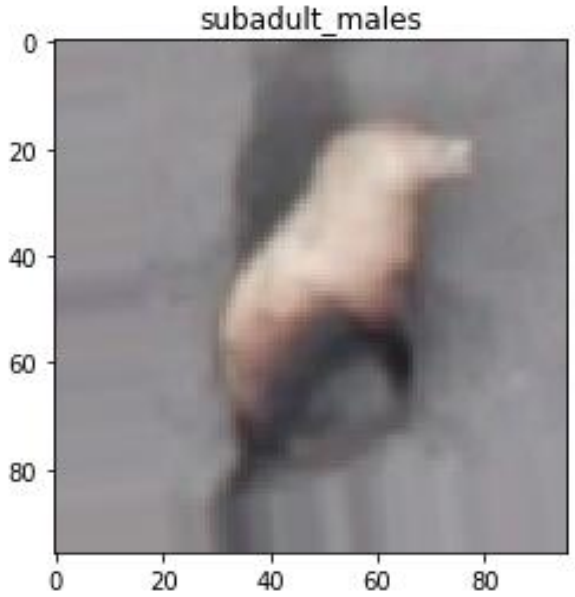
Often, each annotated image represents a class of images that are all likely to belong to the same class

In aereal photograps, for instance, it is normal to have rotated, shifted or scaled images without changing the label

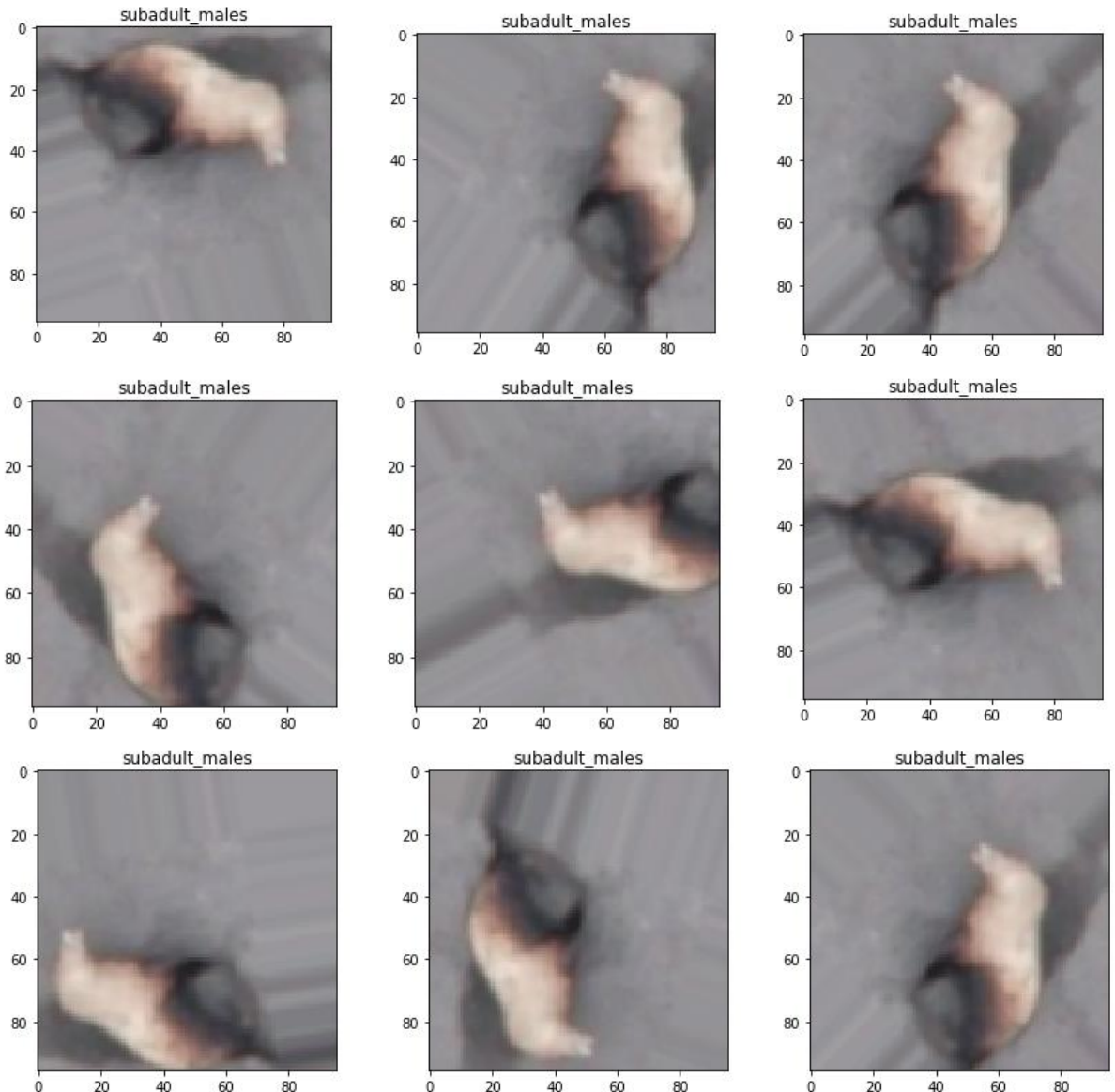


Data Augmentation

Original image



Augmented Images



Data Augmentation

Data augmentation is typically performed by means of

Geometric Transformations:

- Shifts /Rotation/Affine/perspective distortions
- Shear
- Scaling
- Flip

Photometric Transformations:

- Adding noise
- Modifying average intensity
- Superimposing other images
- Modifying image contrast

Data Augmentation: Criteria

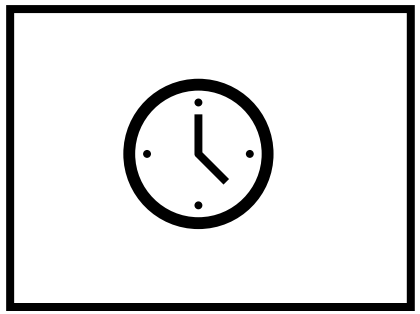
Augmented versions **should preserve the input label**

- e.g. if size/orientation is a key information to determine the output target (either the class or the value in case of regression), wisely consider scaling/rotation as transformation

Augmentation is meant to **promote network invariance** w.r.t. transformation used for augmentation

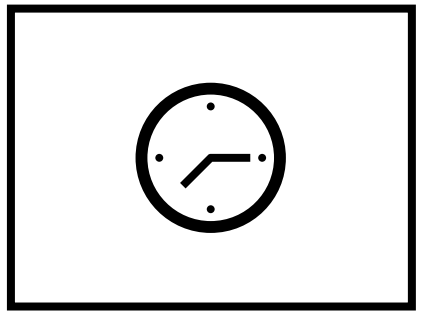
Data Augmentation

You don't want to introduce transformations that ruin distinctive information of a given class



16:00

(I, y)



16:00

$(R(I), y)$

A network predicting the time from an image of a clock without numbers is not invariant w.r.t rotations

Image Augmentation and CNN invariance

Given an annotated image (I, y) and a set of augmentation transformations $\{A_l\}_l$, we train the network using these pairs

$$\{(A_l(I), y)_l\}_l$$

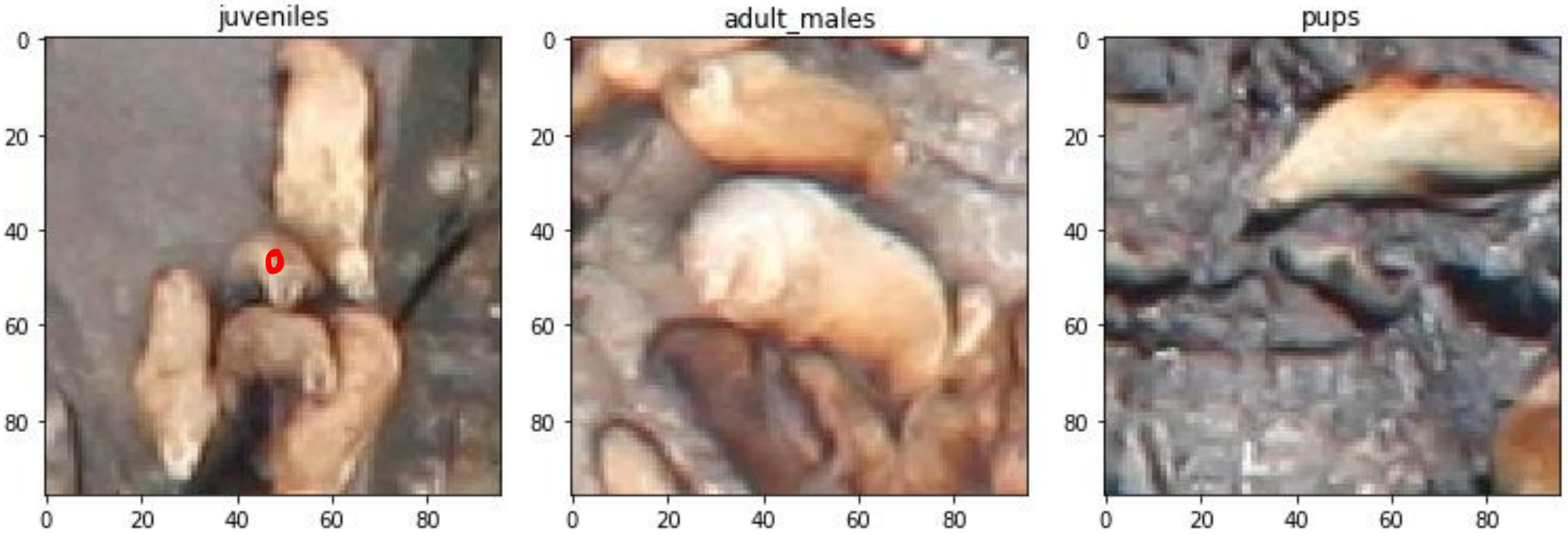
Through data augmentation we train the network to «become invariant» to selected transformations. Since the same label is associated to I and $A_l(I)$ $\forall l$

Unfortunately, invariance might not be always achieved in practice

However...

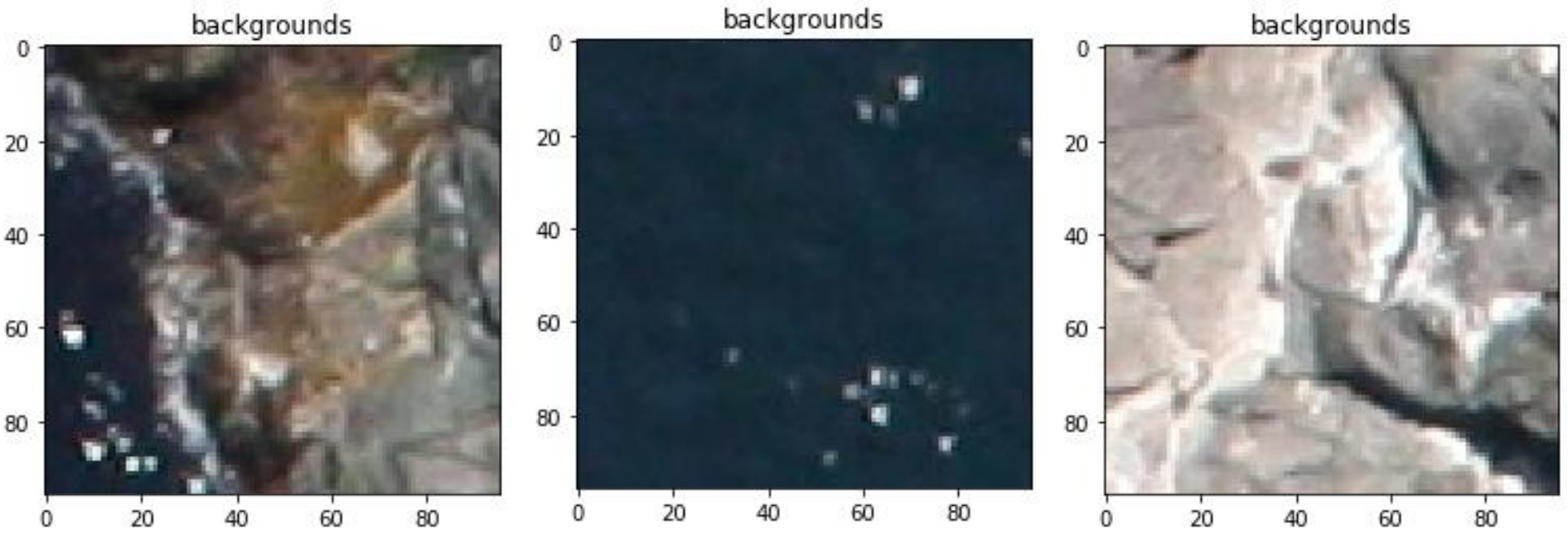
This sort of data augmentation might not be enough to capture the inter-class variability of images...

Superimposition of targets



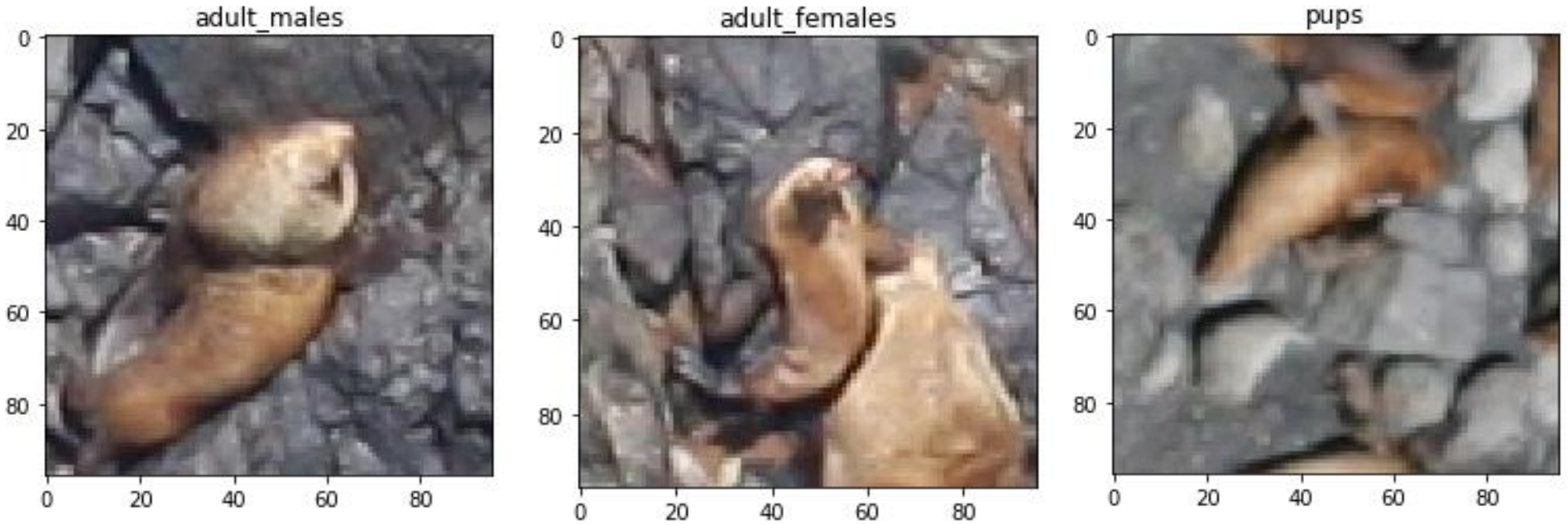
However…

Background variations



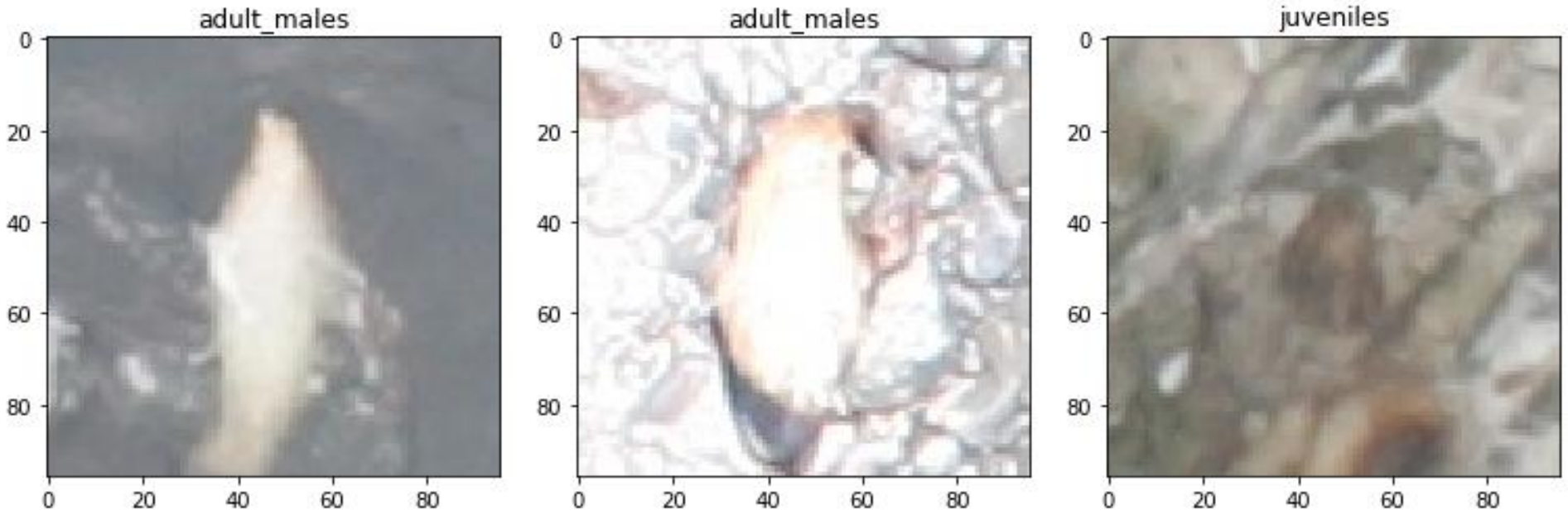
However...

Background variations



However...

Out of focus, bad exposure



Test Time Augmentation

Giacomo Boracchi

Test Time Augmentation (TTA) or Self-ensembling

Even if the CNN is trained using augmentation, **it won't achieve perfect invariance** w.r.t. considered transformations

Test time augmentation (TTA): augmentation can be also performed at test time to improve prediction accuracy.

- Perform a few random augmentation of each test image I

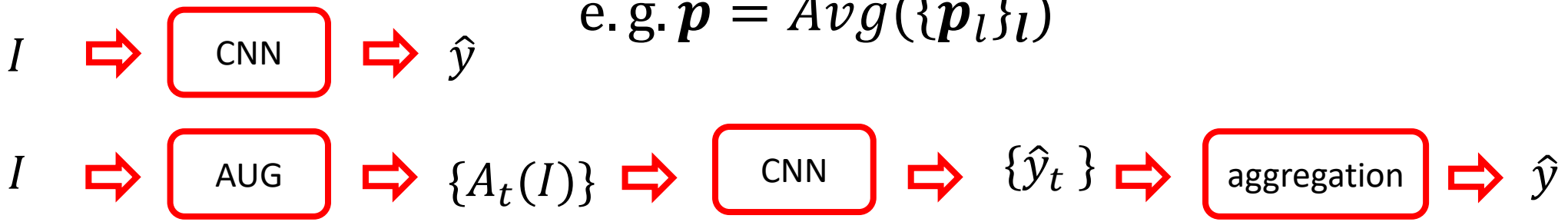
$$\{A_l(I)\}_l$$

- Classify all the augmented images and save the posterior vectors

$$\boldsymbol{p_l} = CNN(A_l(I))$$

- Define the CNN prediction by aggregating the posterior vectors $\{\boldsymbol{p}_l\}$

$$\text{e.g. } \boldsymbol{p} = Avg(\{\boldsymbol{p}_l\}_l)$$



Test Time Augmentation (TTA) or Self-ensembling

TTA:

- particularly useful for test images where the model is quite unsure.
- extremely computationally demanding

Need to wisely configure the number and type of transformations to be performed at test time

Test Time Augmentation

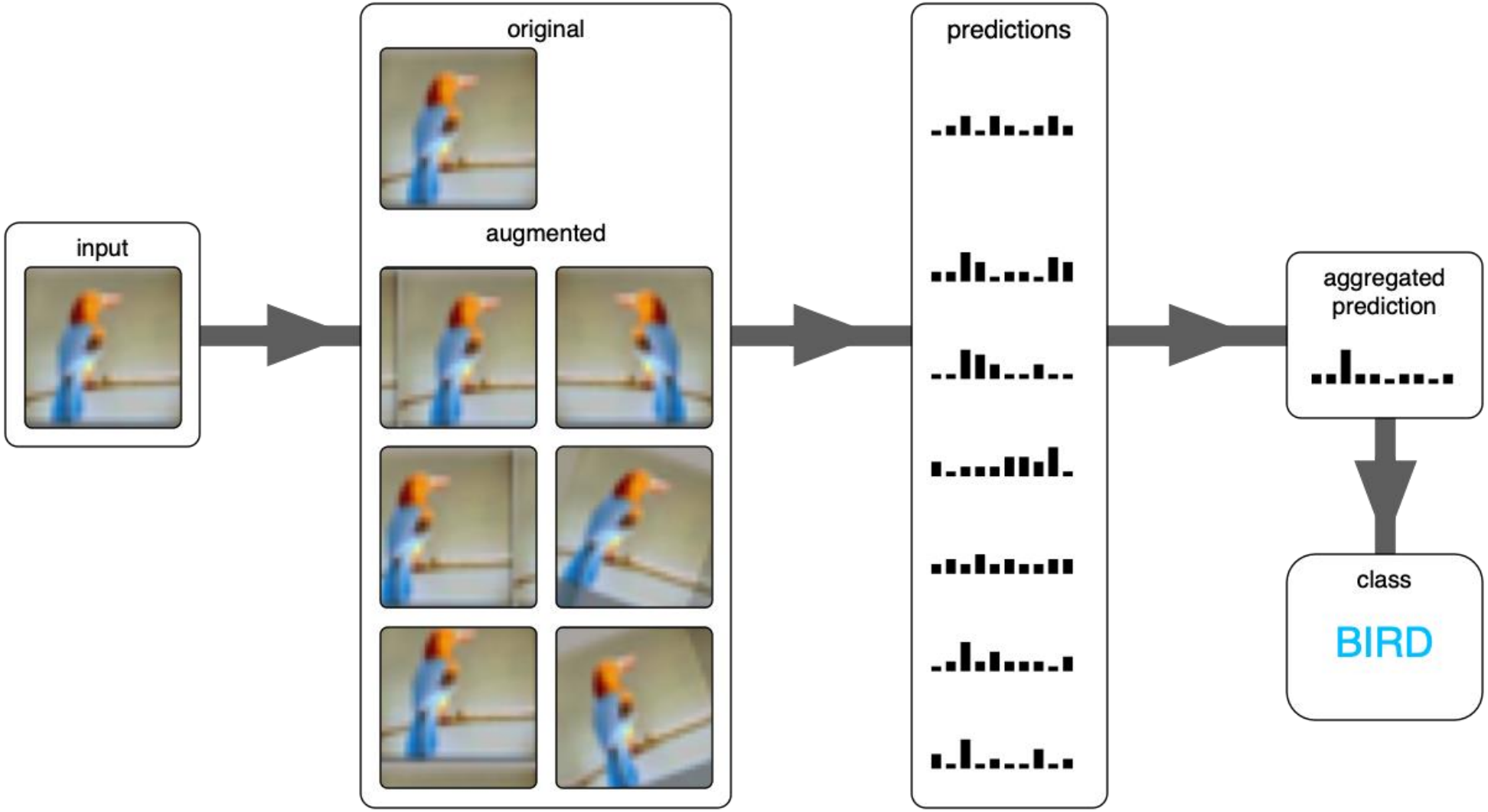


Figure source: https://stepup.ai/test_time_data_augmentation/

Benefits of Data Augmentation

Image Augmentation and Overfitting

Given an annotated image (I, y) and a set of augmentation transformations $\{A_l\}_l$, we train the network using these pairs

$$\{(A_l(I), y)_l\}_l$$

Training including augmentation **reduces the risk of overfitting**, as it significantly increase the training set size

Image Augmentation and Overfitting

Moreover, data augmentation can be used to compensate for class imbalance in the training set, by **creating more realistic examples from the minority class**

In general, **transformations used in data-augmentation** $\{A_l\}$ **can be also class-specific**, in order to preserve the image label

Mixup Augmentation

Augmented copies $\{A_l(I)\}_l$ of an image I live in a ***vicinity*** of I, and **have the same label** of I

Transformations (photometric or geometric) are *expert-driven*

Mixup is a **domain-agnostic** data augmentation technique

- No need to know which (label-preserving) transformations to use
- mixup trains a neural network on *virtual samples* that are **convex combinations of pairs of examples and their labels**

Zhang, H., Cisse, M., Dauphin, Y. N., & Lopez-Paz, D. (2018). Mixup: Beyond empirical risk minimization. ICLR 2018

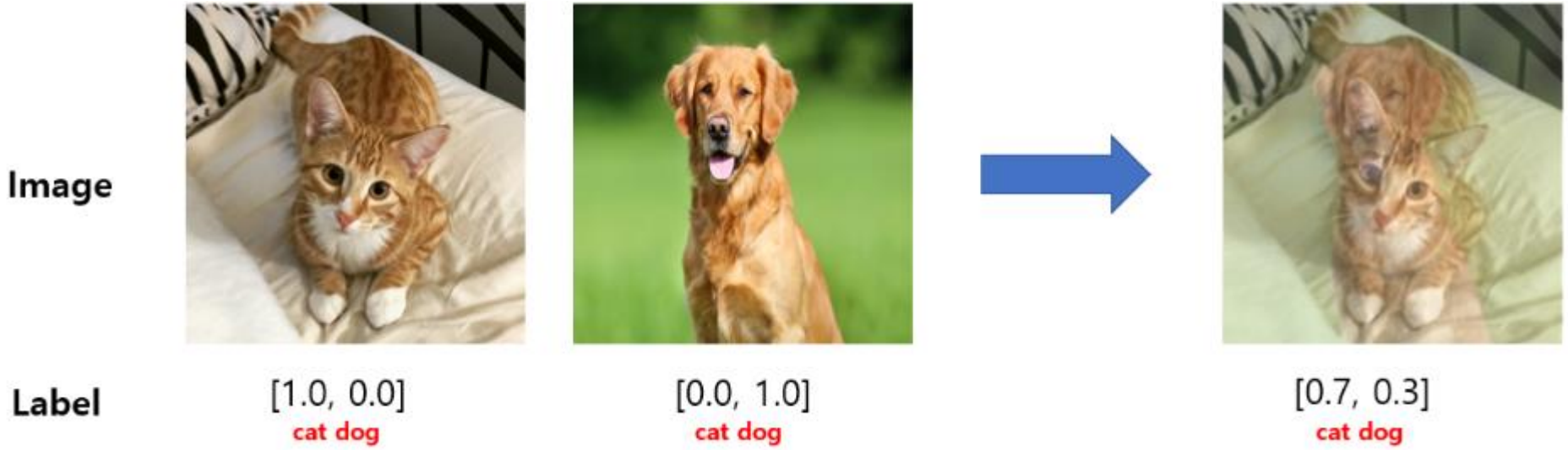
Mixup Augmentation

Given a pair of training samples (I_i, y_i) and (I_j, y_j) of drawn at random possibly belonging to different classes, we define

Virtual samples (and their label)

$$\tilde{I} = \lambda I_i + (1 - \lambda) I_j$$
$$\tilde{y} = \lambda y_i + (1 - \lambda) y_j$$

Where $\lambda \in [0,1]$ and y_i, and y_j are **one-hot encoded labels**



https://www.kaggle.com/code/kaushal2896/data-augmentation-tutorial-basic-cutout-mixup

Mixup Augmentation, Intuition

mixup extends the training distribution by incorporating the prior knowledge that linear interpolations of feature vectors should lead to linear interpolations of the associated targets.

mixup can be implemented in a few lines of code, and introduces minimal computation overhead.

Mixup in keras:
https://keras.io/guides/keras_cv/cut_mix_mix_up_and_rand_augment/

Zhang, H., Cisse, M., Dauphin, Y. N., & Lopez-Paz, D. (2018). Mixup: Beyond empirical risk minimization. ICLR 2018

Augmentation In Keras

Giacomo Boracchi

Augmentation in Keras

There are multiple **preprocessing layers** to be introduced after the input layer to perform:

- photometric transformations
- geometric transformations

to the image

https://keras.io/api/layers/preprocessing_layers/image_augmentation/

Augmentation Layers

These layers apply random augmentation transforms to a batch of images. **They are only active during training.**

tf.keras.layers.RandomCrop

tf.keras.layers.RandomFlip

tf.keras.layers.RandomTranslation

tf.keras.layers.RandomRotation

tf.keras.layers.RandomZoom

tf.keras.layers.RandomHeight

tf.keras.layers.RandomWidth

tf.keras.layers.RandomContrast

Preprocessing Layers

Image preprocessing layers, these are active at inference

- Resizing layer
- Rescaling layer
- CenterCrop layer

Augmenting Images

Define a simple network that performs a random flip of the input

```
flip = tf.keras.Sequential([
  tfkl.RandomFlip("horizontal_and_vertical"),
])
```

Invoke this network to apply augmentation to images

```
flipped_X_train = flip(X_train)
```

Augmenting Images

You can stuck multiple layers

```
# pack a few augmentation layers in a sequence
augmentationNet = tf.keras.Sequential([
    tfkl.RandomFlip("horizontal_and_vertical"),
    tfkl.RandomTranslation(0.1,0.1),
    tfkl.RandomRotation(0.1),
], name='augmentationNet’)
```

Invoke this network to apply augmentation to images

```
augmentated_X_train = augmentationNet(X_train)
```

Training with data augmentation

You can include augmentation / preprocessing layers directly in the network architecture

Note:

- Augmentation layers will be active only during training
- Preprocessing layers will be active also during inference

```
def build_model_with_augmentation(input_shape, output_shape):
    tf.random.set_seed(seed)

    # Build the neural network layer by layer
    input_layer = tfkl.Input(shape=input_shape, name='Input')

    # include augmentation layers
    a = tfkl.RandomFlip("horizontal_and_vertical")(input_layer)
    b = tfkl.RandomTranslation(0.1,0.1)(a)
    c = tfkl.RandomRotation(0.1)(b)

    conv1 = tfkl.Conv2D(…)(c)
```

Augmentation in Keras (other option)

```
from keras.preprocessing.image import
ImageDataGenerator

ImageDataGenerator(
rotation_range=0,
width_shift_range=0.0, height_shift_range=0.0,
brightness_range=None, shear_range=0.0,
zoom_range=0.0, channel_shift_range=0.0,
fill_mode='nearest',
horizontal_flip=False, vertical_flip=False,
rescale=None,
preprocessing_function=None)
```

Augmentation in Keras: flow from images

The Image generator has a method **`flow_from_directory`** that allows to load images in folder where different classes are arranged in subfolders.

```
ImageDataGenerator.flow_from_directory(
    directory=PATCH_PATH + 'train/',
    target_size=(img_width, img_width),
    batch_size=batch_size,
    shuffle=True)
```

Augmentation in Keras (other option)

```
from keras.preprocessing.image import
ImageDataGenerator

ImageDataGenerator(
rotation_range=0,
width_shift_range=0.0, height_shift_range=0.0,
brightness_range=None, shear_range=0.0,
zoom_range=0.0, channel_shift_range=0.0,
fill_mode='nearest',
horizontal_flip=False, vertical_flip=False,
rescale=None,
preprocessing_function=None)
```

… in case you need some extra flexibility

Data Driven Features

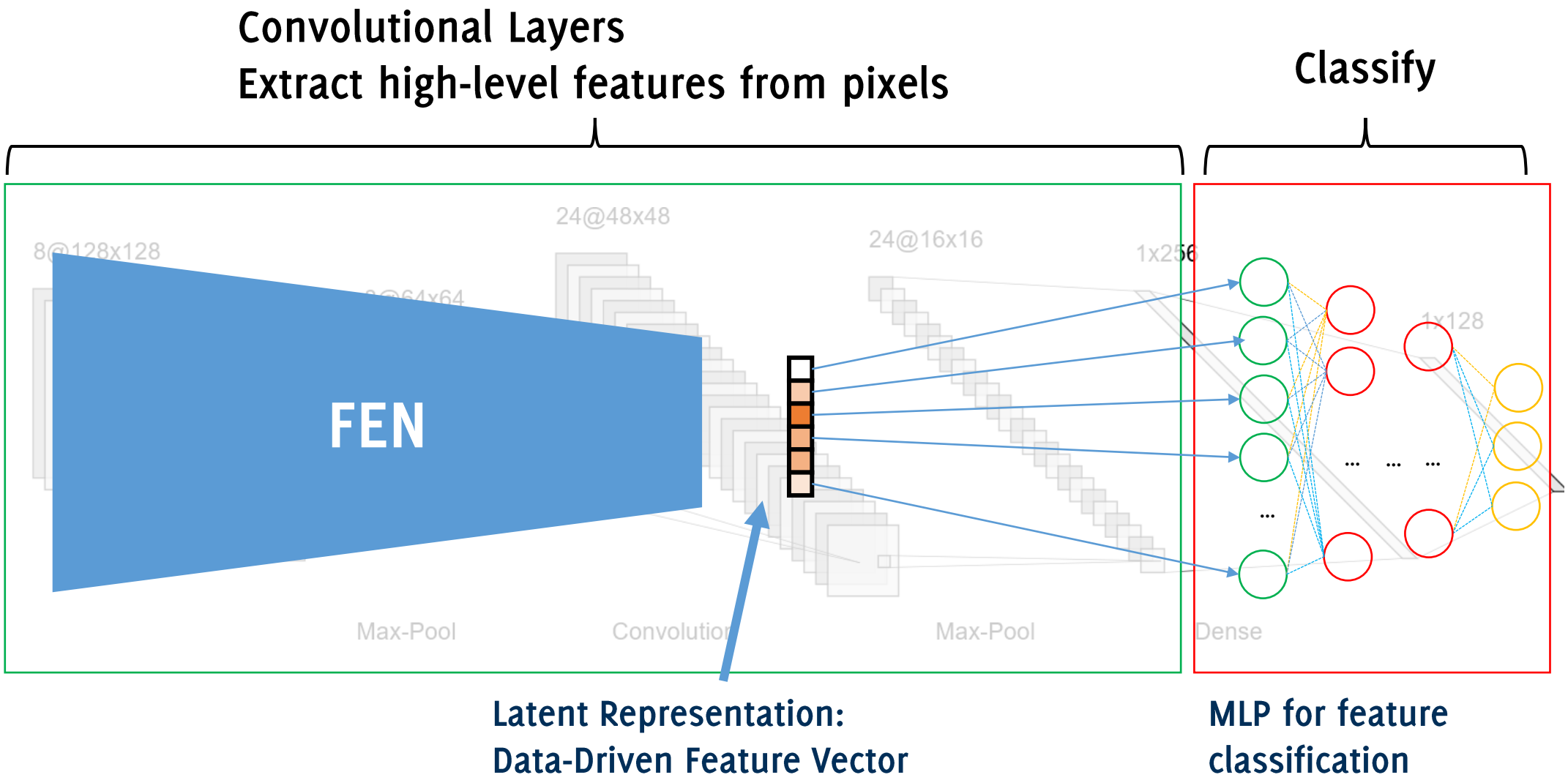
Giacomo Boracchi

Latent representation in CNNs

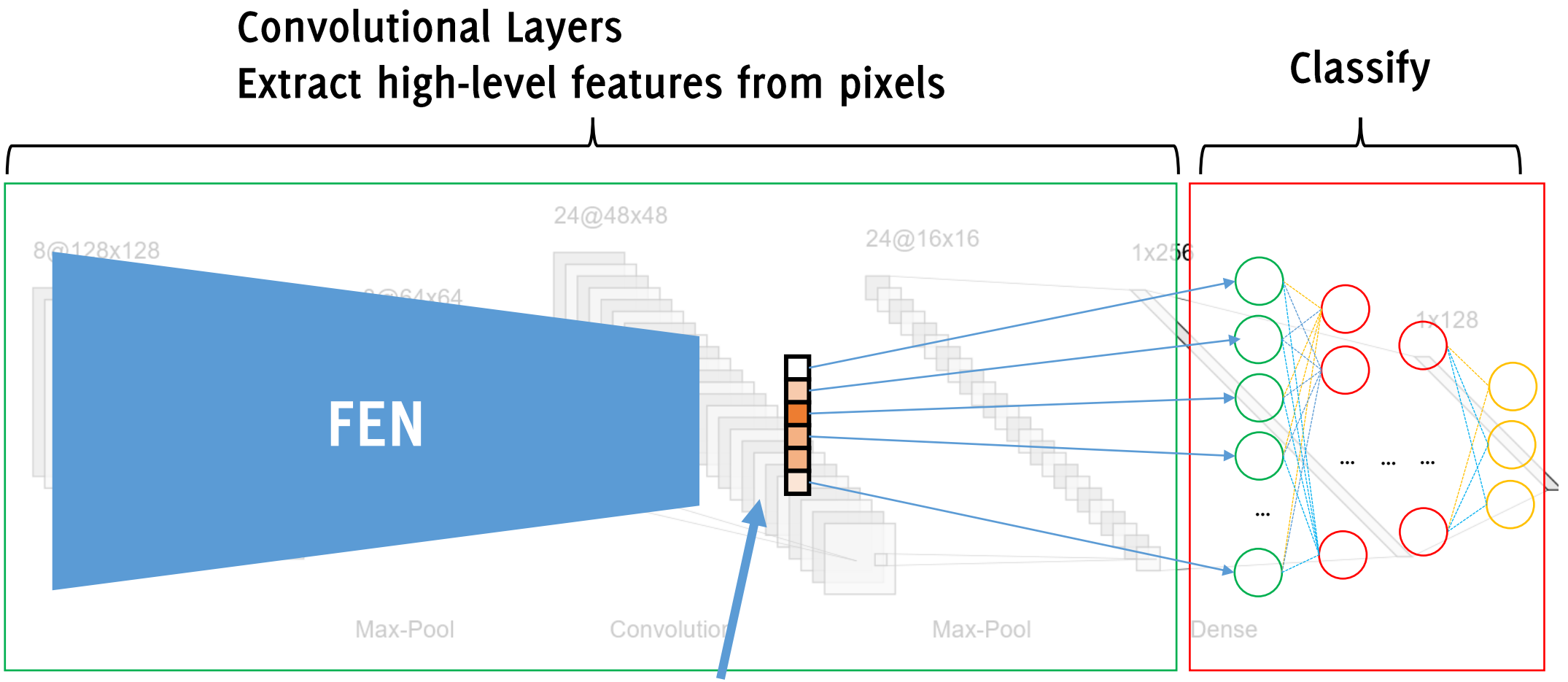
Repeat the «t-SNE experiment» on the CIFAR dataset, using the last layer of the CNN as vectors

Giacomo Boracchi

The typical architecture of a CNN



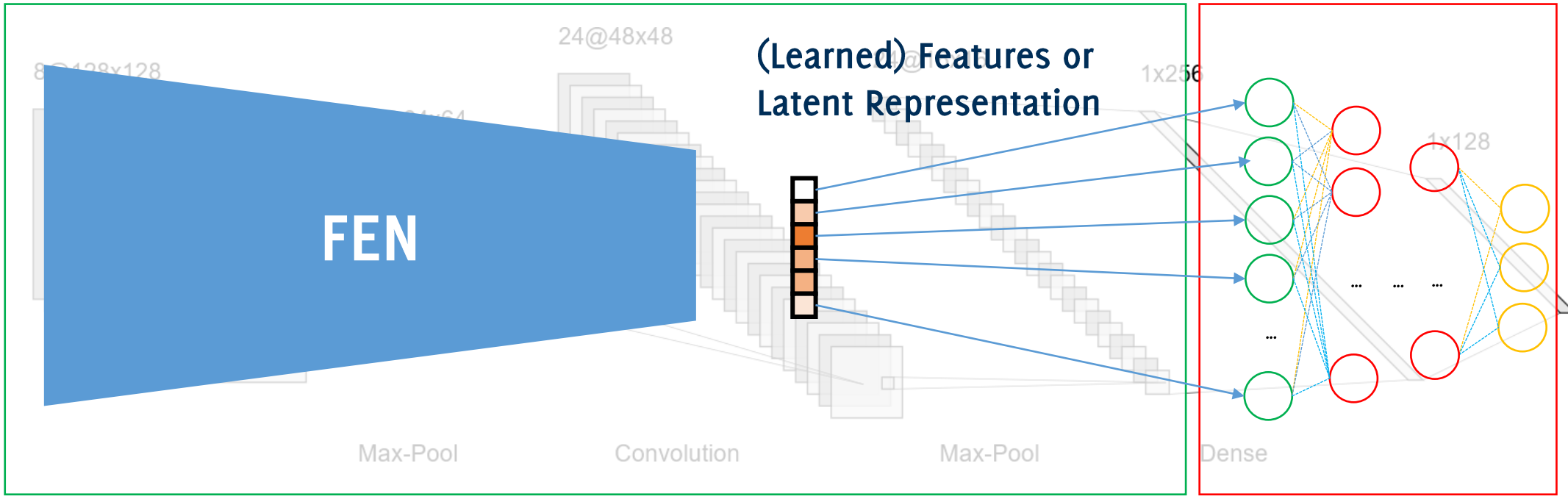
The typical architecture of a CNN



Typically, to learn meaningful representations, many layers are required

The network becomes deep

Feature Extraction Networks



Data-Driven Feature extraction

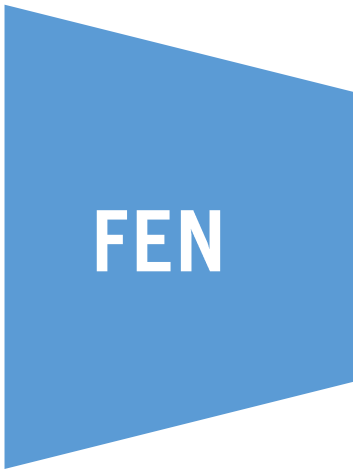
Feature Classification

FEN: FEATURE EXTRACTION NETWORK, the convolutional block of CNN

Feature Extraction Networks

The new paradigm to solve visual recognition problems

- Instead of engineering features
- Train a CNN in an end-to-end manner and you get better features that optimized for solving the problem at-hand



Key advantages:

- **Everything** (feature extraction and classification) **is optimized** for improving the task at hand.
- **End-to-end trainable** solutions require no experts, just annotated data.
- Plenty of high-level frameworks (Keras, Tensorflow, PyTorch, TensorFlow Lite) that allows solving complex visual recognition by simply **programming black-boxes.**
- Democratisation of Computer Vision!
- Very effective…

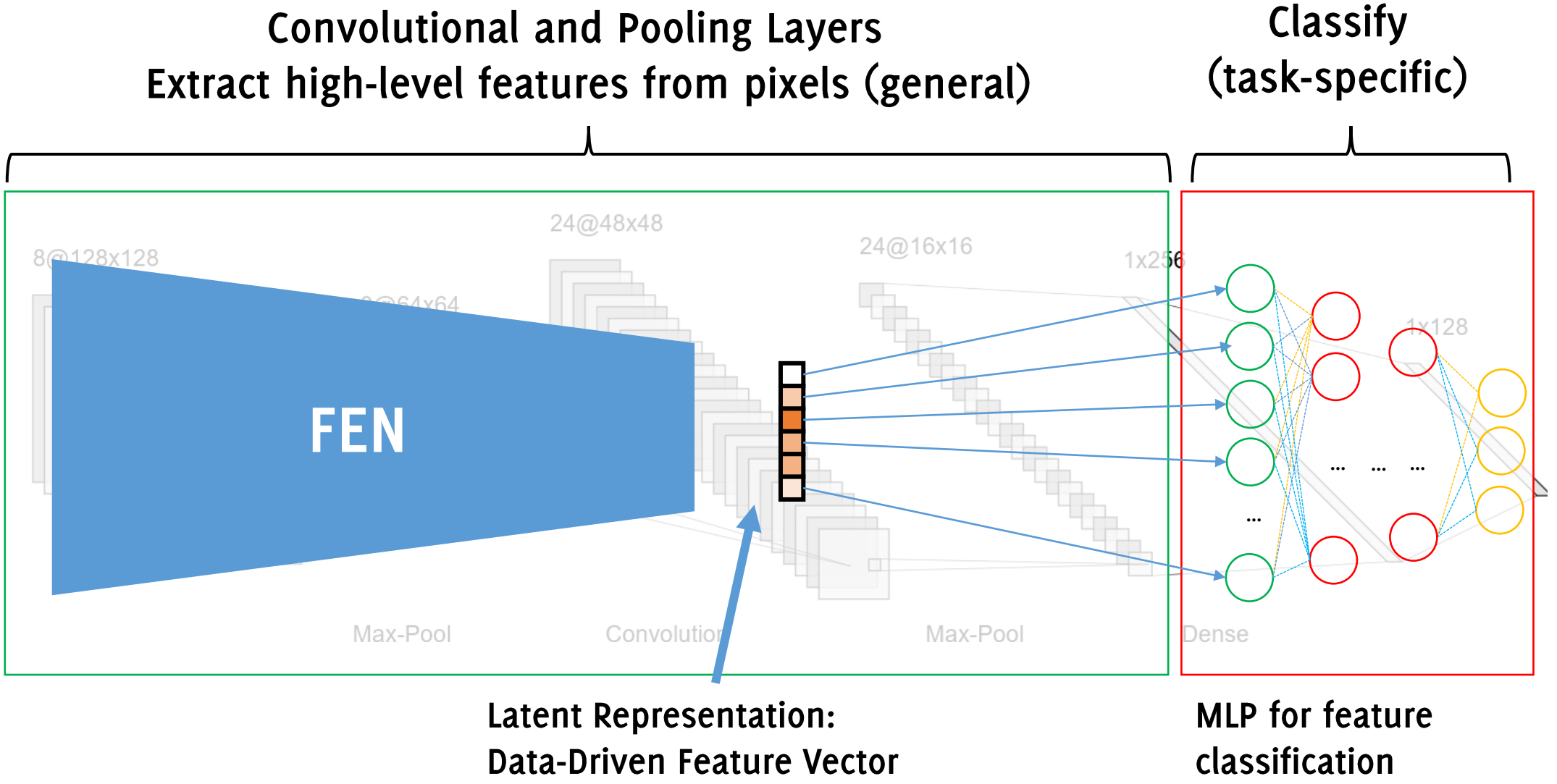
Limited Amount of Data: Transfer Learning

Training a CNN with Limited Aumont of Data

Giacomo Boracchi

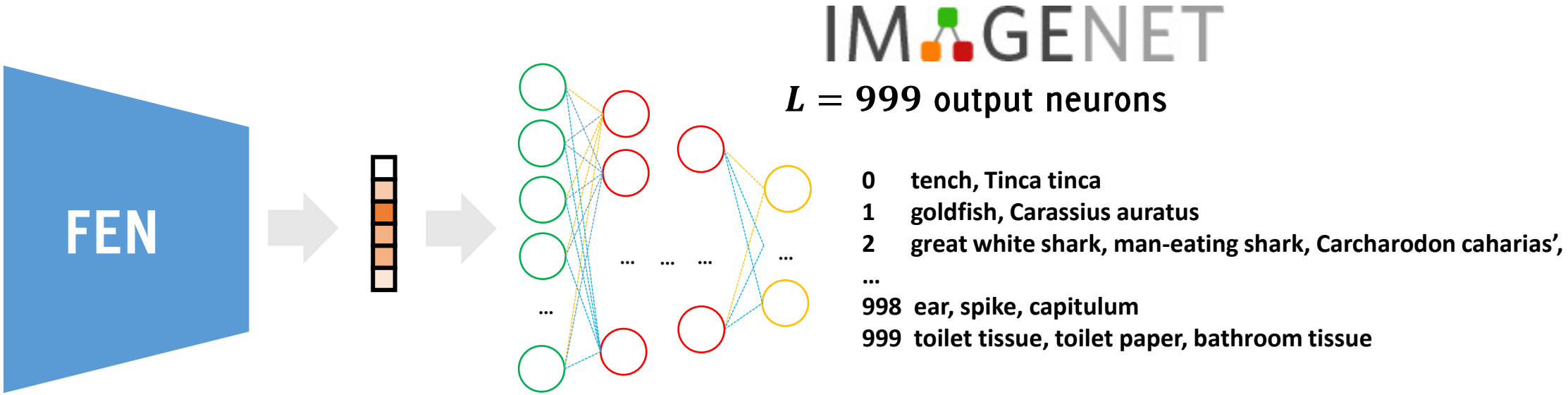
The Rationale Behind Transfer Learning

The typical architecture of a CNN

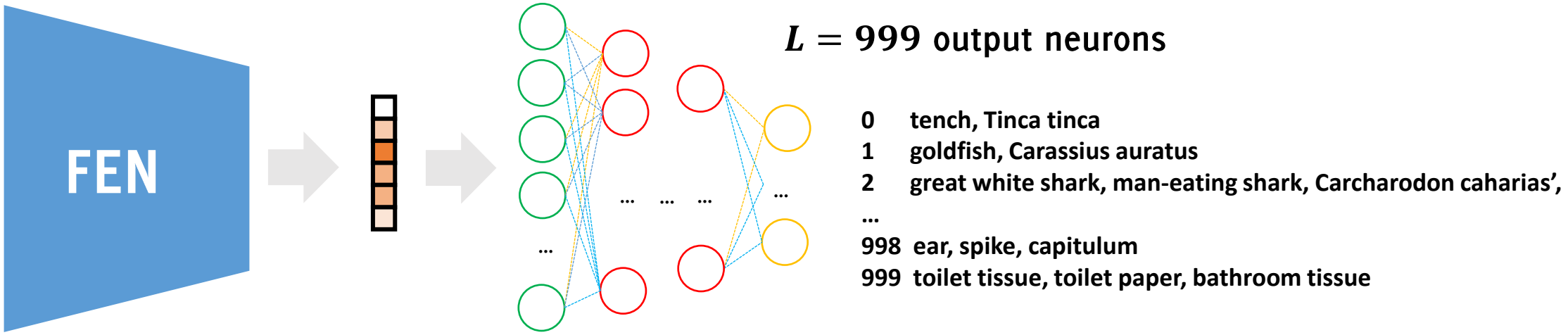


Very Good Features!

FEN is trained on large training sets (e.g. ImageNet) typically including hundres of classes.



Very Good Features!

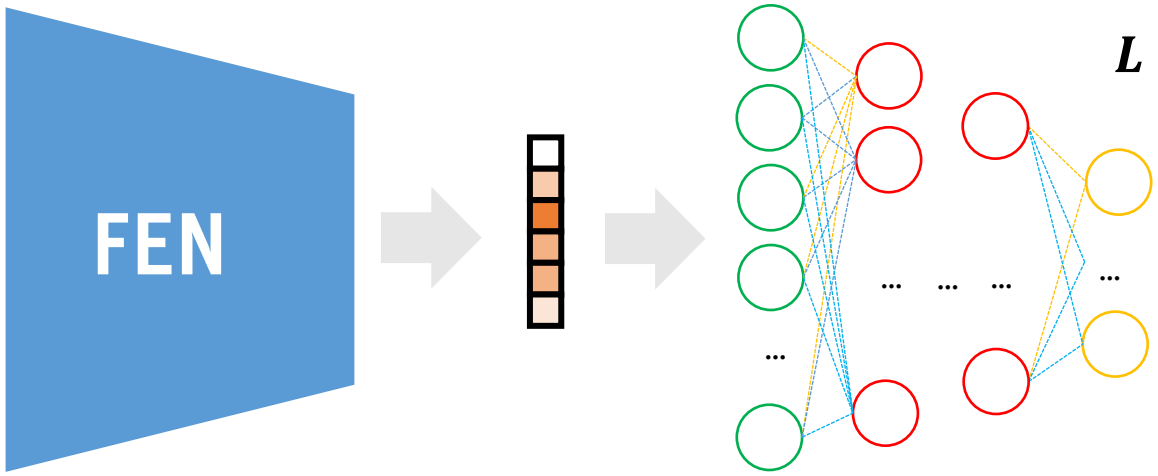


The **output of the fully connected layer** has the same size as the number of classes L, and each component provide a score for the input image to belong to a specific class.

This is **very task-specific:**

- What if I have a *small* TR of images of **cats and dogs** for training?
- What if I want to train a classifier for the six types of sealions?
- Can we use these feature for solving other classification problems?

Transfer Learning

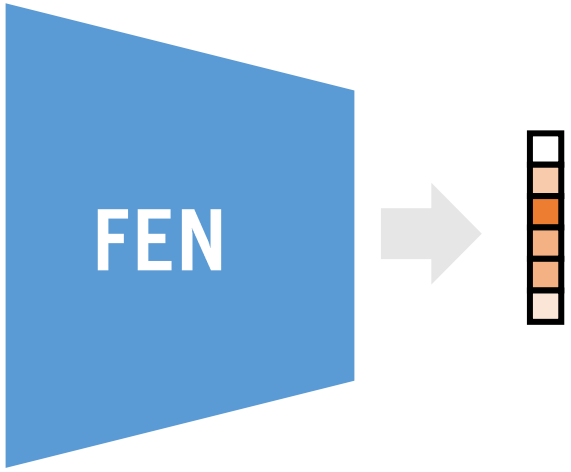


$L = 999$ output neurons

0 tench, Tinca tinca
1 goldfish, Carassius auratus
2 great white shark, man-eating shark, Carcharodon caharias',
...
998 ear, spike, capitulum
999 toilet tissue, toilet paper, bathroom tissue

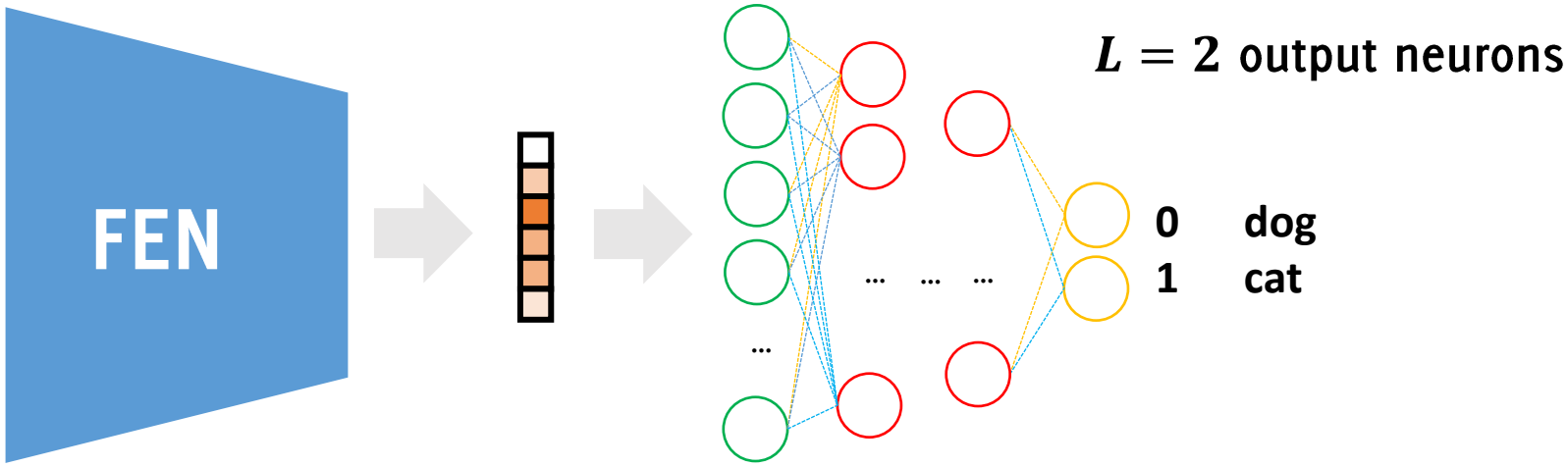
1. Take a powerful pre-trained NN (e.g., ResNet, EfficientNet, MobileNet)

Transfer Learning



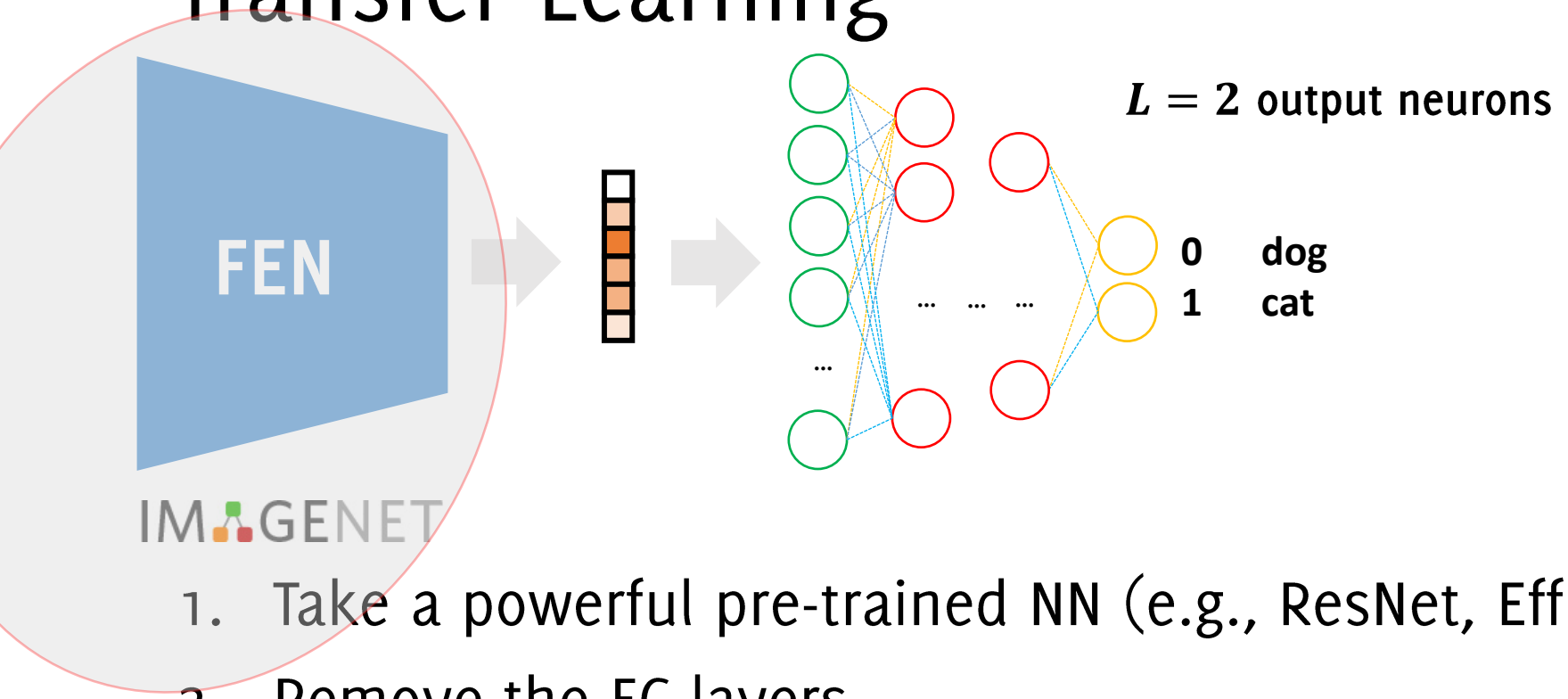
1. Take a powerful pre-trained NN (e.g., ResNet, EfficientNet, MobileNet)
2. Remove the FC layers.

Transfer Learning



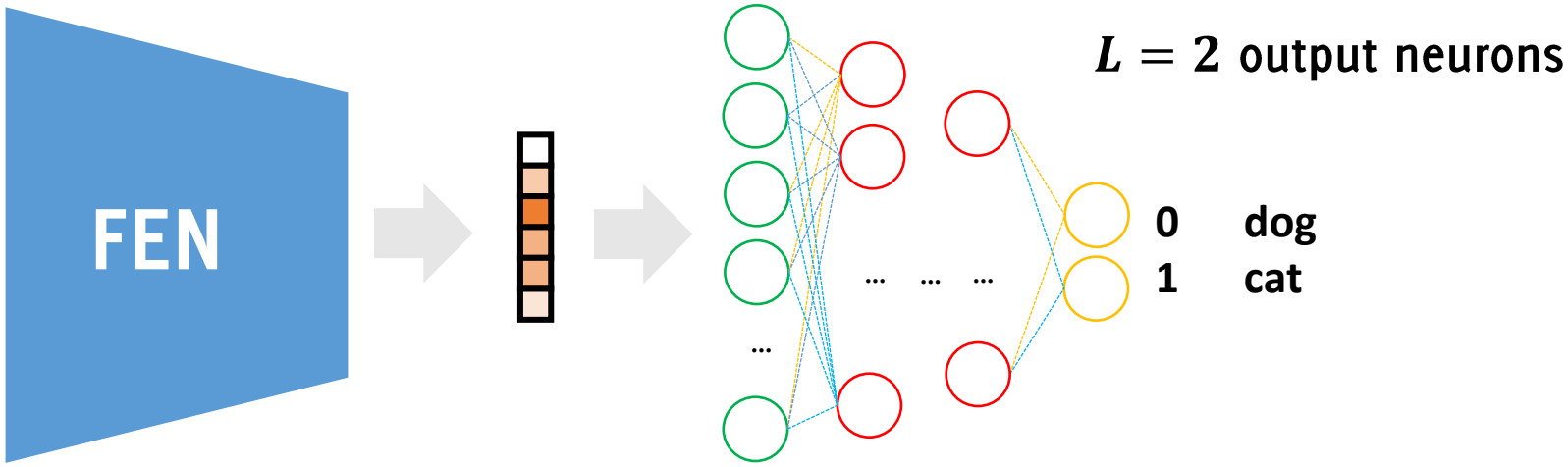
1. Take a powerful pre-trained NN (e.g., ResNet, EfficientNet, MobileNet)
2. Remove the FC layers.
3. Design new FC layers to match the new problem, plug after the FEN (initialized at random)

Transfer Learning



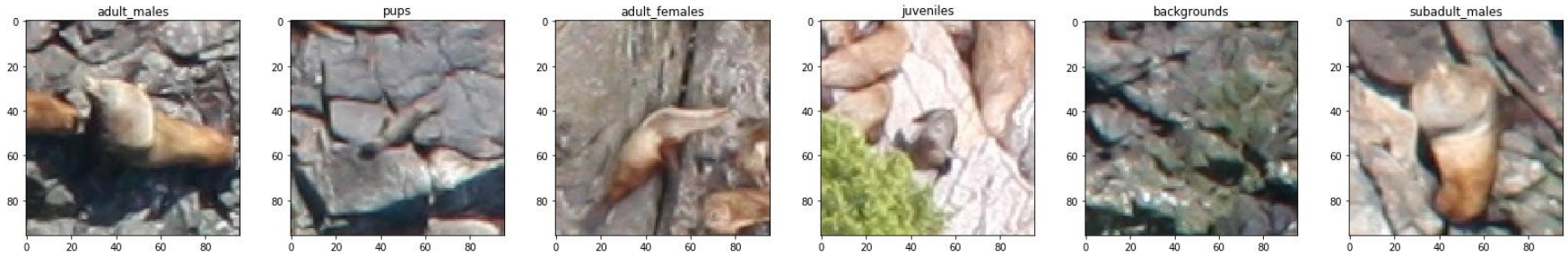
1. Take a powerful pre-trained NN (e.g., ResNet, EfficientNet, MobileNet)
2. Remove the FC layers.
3. Design new FC layers to match the new problem, plug after the FEN (initialized at random)
4. «Freeze» the weights of the FEN.

Transfer Learning



1. Take a powerful pre-trained NN (e.g., ResNet, EfficientNet, MobileNet)
2. Remove the FC layers.
3. Design new FC layers to match the new problem, plug after the FEN (initialized at random)
4. «Freeze» the weights of the FEN.
5. Train the whole network on the new training data TR

Transfer Learning in the Sealion Case



VGG16

«Froozen»

IMAGENET

FC Layers having
6 output neurons

Trainable

6×1

1024x1

4096x1

2x2

1024

Train the whole architecture on the training set of augmented patches from sealions

https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

Transfer Learning vs Fine Tuning

Different Options:

- **Transfer Learning**: **only the FC** layers are being trained. A good option when **little training data** are provided and the **pre-trained model** is expected to **match** the problem at hand
- **Fine tuning:** the whole **CNN is retrained**, but the **convolutional layers** are **initialized to the pre-trained model**. A good option when **enough training data** are provided or when the **pre-trained model is not expected to match** the problem at hand.

Typically, for the same optimizer, **lower learning rates** are used when performing fine tuning than when training from scratches

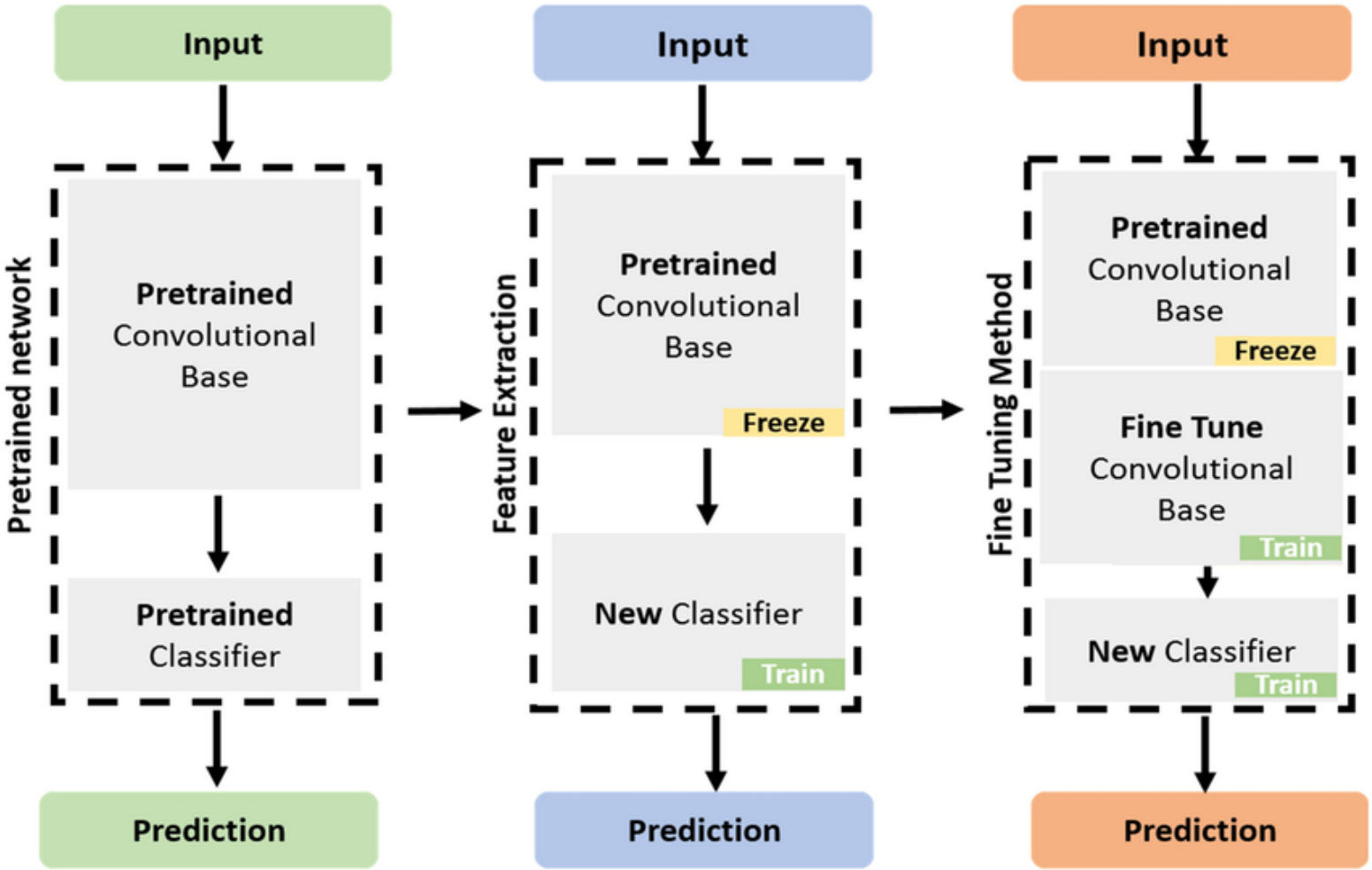
Best Practice

Typically, to take the most out of a pretrained model:

- Connect a new output layer (having few parameters)
- Transfer Learning: train the output layer only
- Make all the “last layers” trainable
- Fine tuning: train the entire network with a low learning rate

```
# Compile the model
ft_model.compile(loss=tfk.losses.BinaryCrossentropy(), optimizer=tfk.optimizers.Adam(1e-5), metrics='accuracy')
```

This strategy allows defining good predictions once the output layer has been trained



Giacomo Boracchi

Transfer Learning In Keras

Giacomo Boracchi

Where to find pretrained models?

https://keras.io/api/applications/

Available models

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7
NASNetLarge	343	82.5%	96.0%	88.9M	533	344.5	20.0
EfficientNetB0	29	77.1%	93.3%	5.3M	132	46.0	4.9

Importing Pretrained Models in keras…

Pre-trained models are available, typically in two ways:

- **`include_top = True`**: provides the entire network, including the fully convolutional layers. This network can be used to solve the classification problem it was trained for
- **`include_top = False`**: contains only the convolutional layers of the network, and it is specifically meant for transfer learning.

Have a look at the size of these models in the two options!

Importing Pretrained Models in keras…

```
from keras import applications

base_model = applications.VGG16(weights =
"imagenet", include_top=False, input_shape =
(img_width, img_width, 3), pooling = “avg")
```

Importing Pretrained Models in keras…

```
from keras import applications
base_model = applications.VGG16(weights =
"imagenet", include_top=False, input_shape =
(img_width, img_width, 3), pooling = "avg")
```

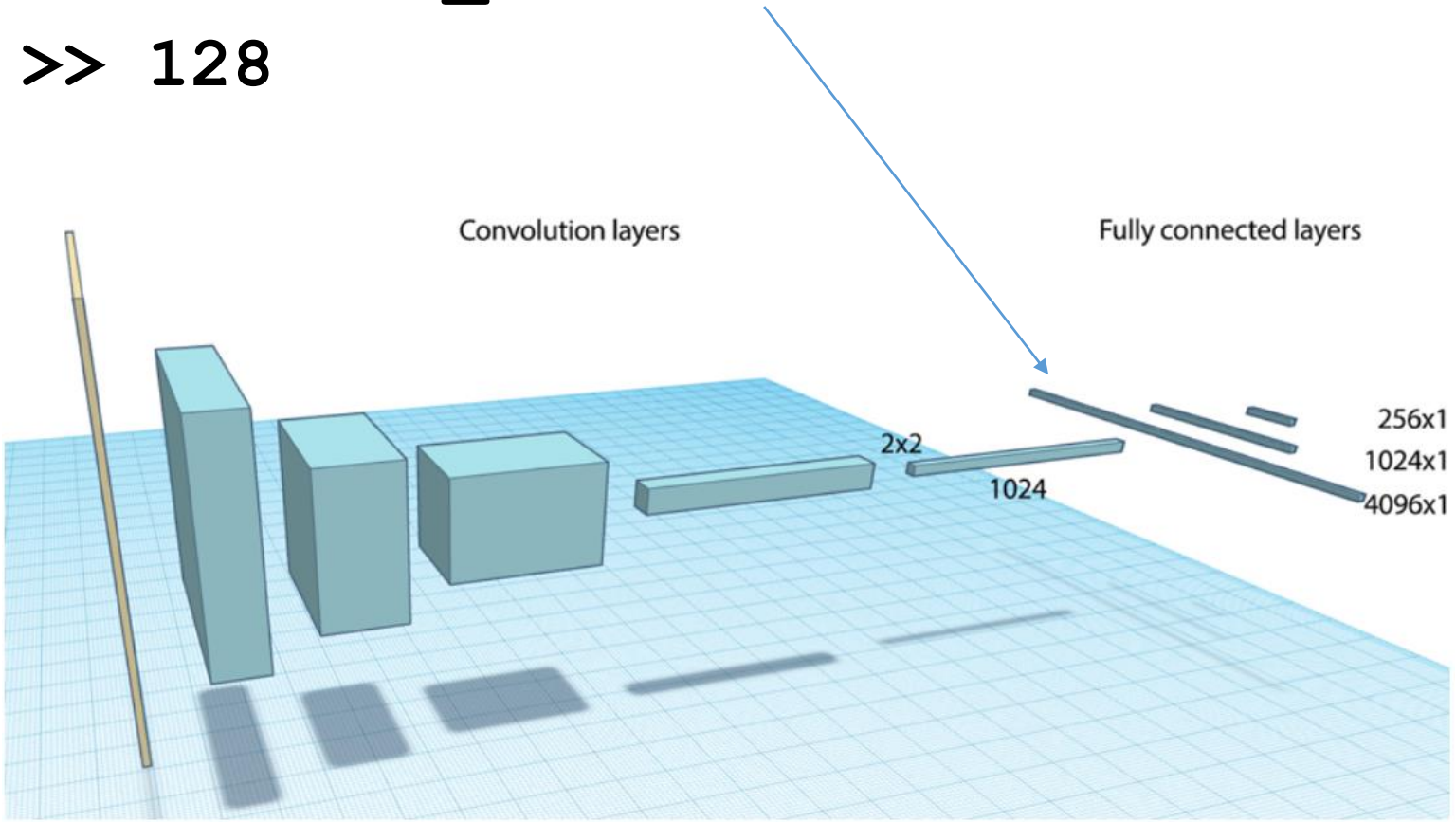
When **`include_top=False`**, the network returns the output of a global pooling layer, which can be:

- **`pooling = "avg"`** Global Averaging Pooling (GAP)
- **`pooling = "max"`** Global Max Pooling (GMP)
- **`pooling = "none"`** There is no pooling, it returns the activations

How to extract the feature extraction network?

Actually, for sequential models, you create feature extraction network

```
fen = tfk.Sequential(model.layers[:-2])
fen.output_shape
>> 128
```



How to extract the feature extraction network?

Actually, for sequential models, you create feature extraction network

```
fen = tfk.Sequential(model.layers[:-2])
```

Note: each Keras Application expects a specific kind of input **preprocessing**.

For MobileNetV2, call
`tf.keras.applications.mobilenet_v2.preprocess_input`

on your inputs before passing them to the model. mobilenet_v2.preprocess_input will scale input pixels between -1 and 1.

Transfer Learning in Keras…

Requires a bit of TensorFlow Backend to add the modified Fully connected layer at the top of a pretrained model

Then, before training it is necessary to loop through the network layers (they are in **`model.layers`**) and then modify the trainable property

```
for layer in model.layers[: lastFrozen]:
    layer.trainable=False
```

An example of model loading

```
# load a pre-
trained MobileNetV2 model without weights
mobile = tfk.applications.MobileNetV2(
    input_shape=(224, 224, 3),
    include_top=False,
    pooling='avg',
)
```

Transfer Learning: adding the new Network Top

Requires a bit of TensorFlow Backend to add the modified Fully connected layer at the top of a pretrained model

Then, before training it is necessary to loop through the network layers (they are in **`model.layers`**) and then modify the trainable property

```
# Add the classifier layer to the MobileNet
inputs = tfk.Input(shape=(224,224,3))
x = mobile(inputs) # concatenates inputs and the output of the pretrined network... the entire mobileNet is hand led as a layer
x = tfkl.Dropout(0.5)(x) # good to prevent overfitting
outputs = tfkl.Dense(1, activation='sigmoid')(x) # conne ct a new output layer
```

Transfer Learning: setting layers trainable property

Requires a bit of TensorFlow Backend to add the modified Fully connected layer at the top of a pretrained model

Then, before training it is necessary to loop through the network layers (they are in **`model.layers`**) and then modify the trainable property

```
for layer in model.layers[: lastFrozen]:
    layer.trainable=False
```

Image Retrieval From The Latent Space

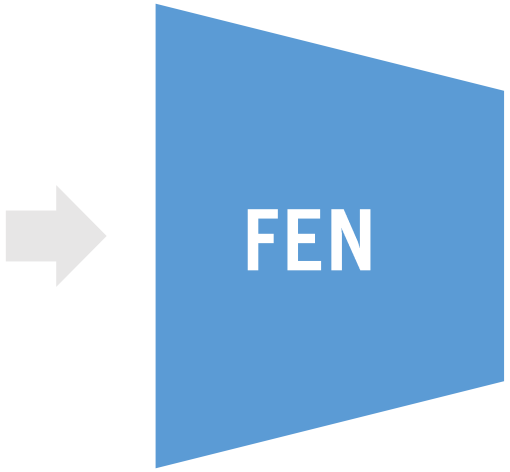
Giacomo Boracchi

Features are Good For Image Retrieval

Feed a test image and compute its latent representation

Test image

I



$\boldsymbol{x}$

Latent Representation:
Data-Driven Feature Vector

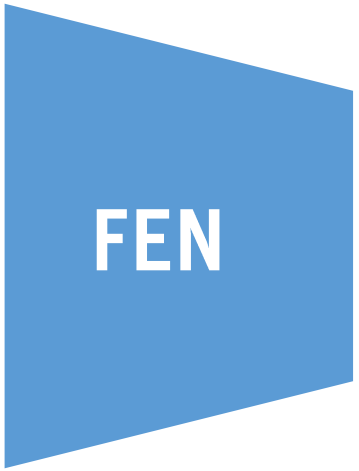
Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012

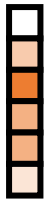
Features are Good For Image Retrieval

Feed a test image and compute its latent representation

Test image

I



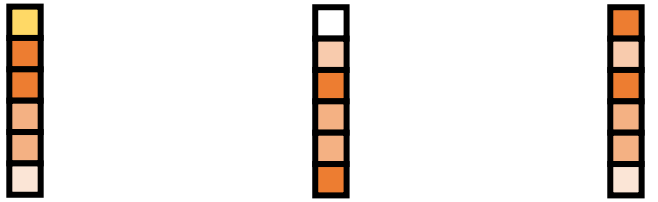


$\boldsymbol{x}$

Latent Representation:
Data-Driven Feature Vector

Retrieve the training images having the closest latent representations

The 3- nearest neighborhood of $\boldsymbol{x}$



$\boldsymbol{x_1}$ $\boldsymbol{x_2}$ $\boldsymbol{x_3}$

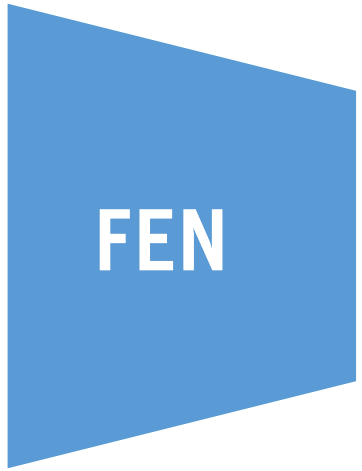
Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012

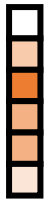
Features are Good For Image Retrieval

Feed a test image and compute its latent representation

Test image

I



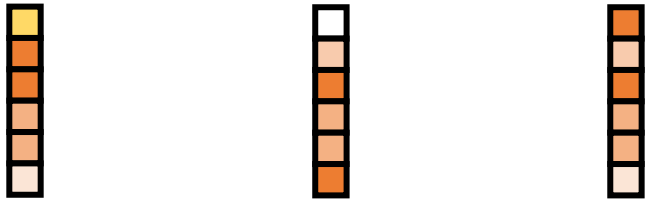


x

Latent Representation:
Data-Driven Feature Vector

Retrieve the training images having the closest latent representations

The 3- nearest neighborhood of x



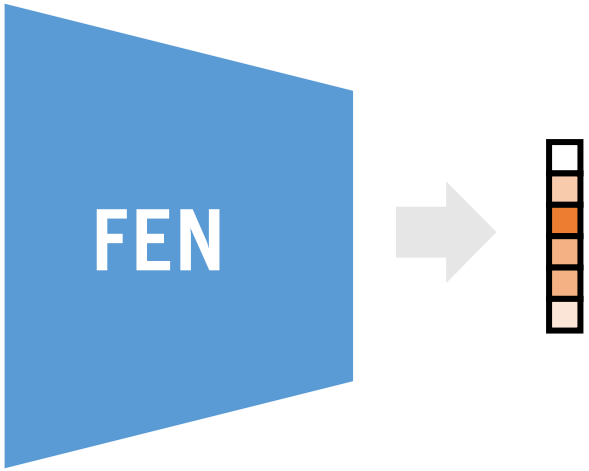
x_1 x_2 x_3

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012

Features are Good For Image Retrieval

Feed a test image and compute its latent representation

Test image



Training Images corresponding to the closest latent representations!

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems 25 (2012).

1-NN classification in the latent space

```
# feed the test imate to the fen
image_features = fen.predict(test_image)


# feed fen with the entire training set (use batches of 512)
features = fen.predict(X_train_val,batch_size=512,verbose=0)

# compute distances (e.g. ell1) between image_featres and features,
distances = np.mean(np.abs(features - image_features),axis=-1)
sortedDistances = distances.argsort()


# sort images (and labels) according to the distance computed above
ordered_images = X_train_val[sortedDistances]
ordered_labels = y_train_val[sortedDistances]
# associate to image_features the closest image ordered_images[0]
```

CNNs in Keras

Giacomo Boracchi

What is Keras?

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

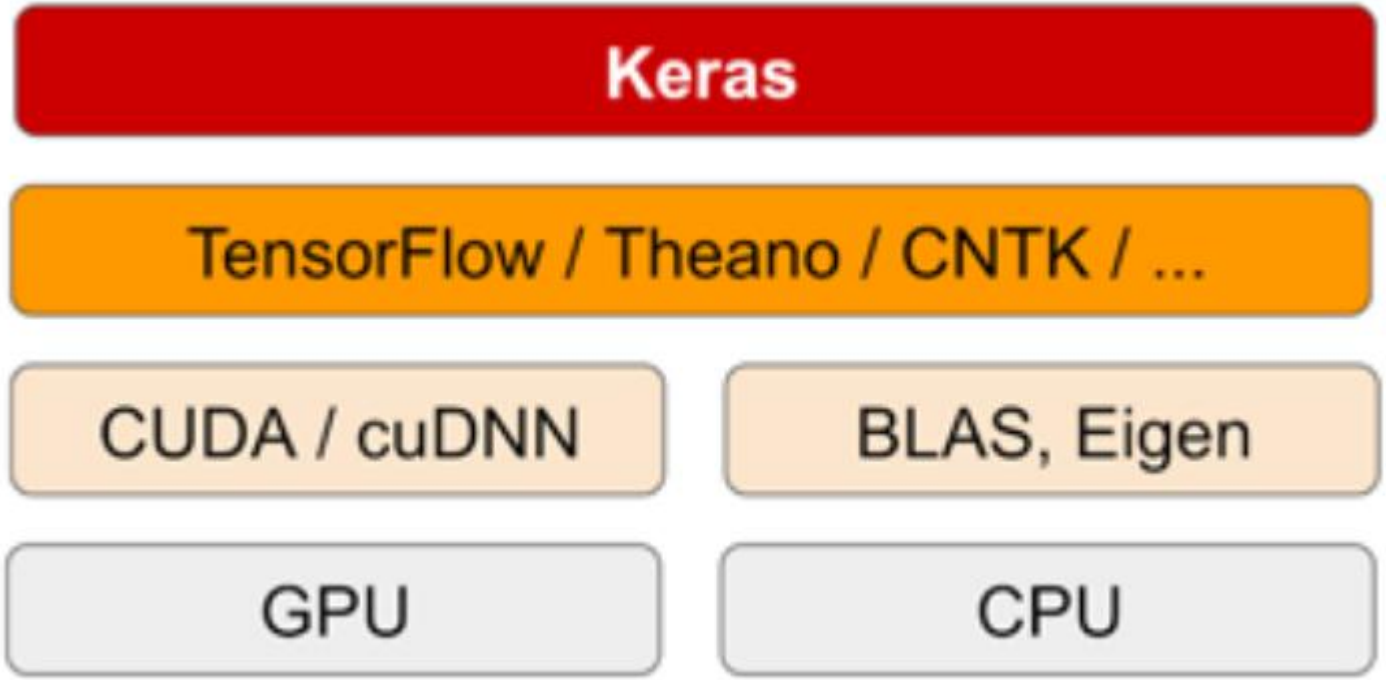
Designed to enable **fast experimentation with deep neural networks**, it focuses on being **user-friendly, modular, and extensible**

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:

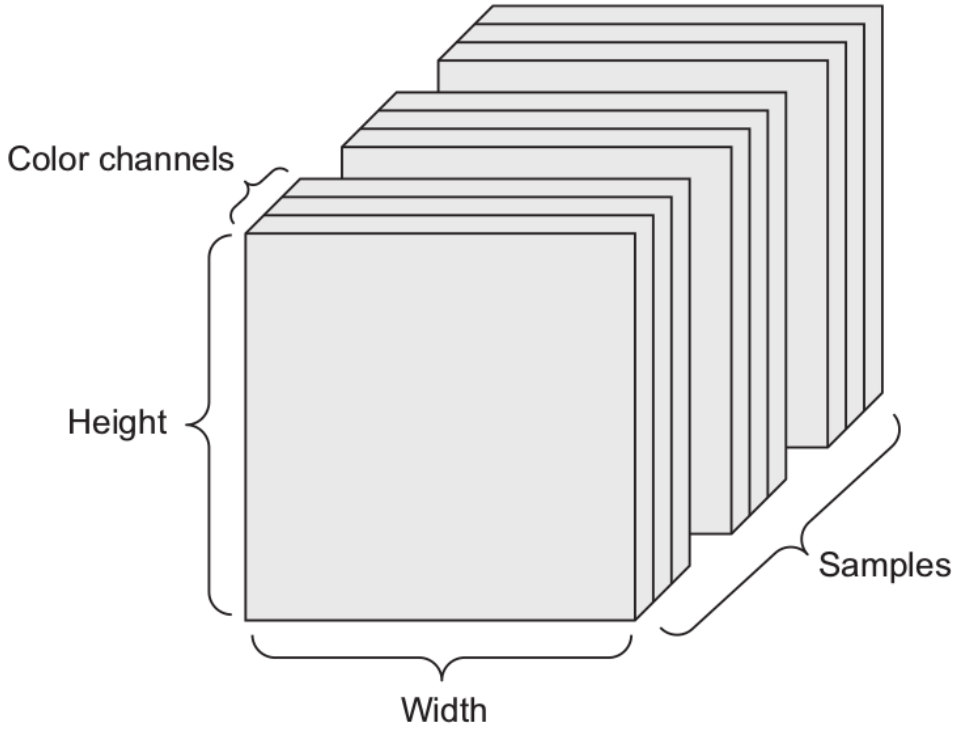
Not as flexible

Need more flexibility? Access the backend directly!

We will manipulate 4D tensors

Images are represented in 4D tensors:

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



Building the Network

Convolutional Networks in Keras

```
# it is necessary to import some package
from keras.models import Sequential
from keras.layers import Dense, Flatten
from keras.layers import Conv2D, MaxPooling2D

# and initialize an object from Sequential()
model = Sequential()
```

A very simple CNN

```
# Network Layers are stacked by means of the
.add() method
model.add(Conv2D(filters = 64, kernel_size=3,
activation='relu', input_shape=(28,28,1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
```

Convolutional Layers

```
# Convolutional Layer
model.add(Conv2D(filters = 64, kernel_size=3,
activation='relu', input_shape=(28,28,1)))
# the input are meant to define:
# - The number of filters,
# - The spatial size of the filter (assumed
squared), while the depth depends on the network
structure
# - the activation layer (always include a
nonlinearity after the convolution)
# - the input size: (rows, cols, n_channels)
```

Convolutional Layers

```
# Convolutional Layer
model.add(Conv2D(filters = 64, kernel_size=3,
activation='relu', input_shape=(28,28,1)))
# This layer creates a convolution kernel that
is convolved with the layer input to produce a
tensor of outputs.
# When using this layer as the first layer in a
model, provide the keyword argument input_shape
(tuple of integers, does not include the batch
axis), e.g. input_shape=(128, 128, 3) for
128x128 RGB pictures in
data_format="channels_last".
```

Conv2D help

Arguments

filters: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).

kernel_size: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.

strides: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1.

padding: one of "valid" or "same" (case-insensitive). Note that "same" is slightly inconsistent across backends with strides != 1, as described here

data_format: A string, one of "channels_last" or "channels_first". The ordering of the dimensions in the inputs. "channels_last" corresponds to inputs with shape (batch, height, width, channels) while "channels_first" corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

Conv2D help

Arguments

dilation_rate: an integer or tuple/list of 2 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any dilation_rate value != 1 is incompatible with specifying any stride value != 1.

activation: Activation function to use (see activations). If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x).

use_bias: Boolean, whether the layer uses a bias vector.

kernel_initializer: Initializer for the kernel weights matrix (see initializers).

bias_initializer: Initializer for the bias vector (see initializers).

kernel_regularizer: Regularizer function applied to the kernel weights matrix (see regularizer).

bias_regularizer: Regularizer function applied to the bias vector (see regularizer).

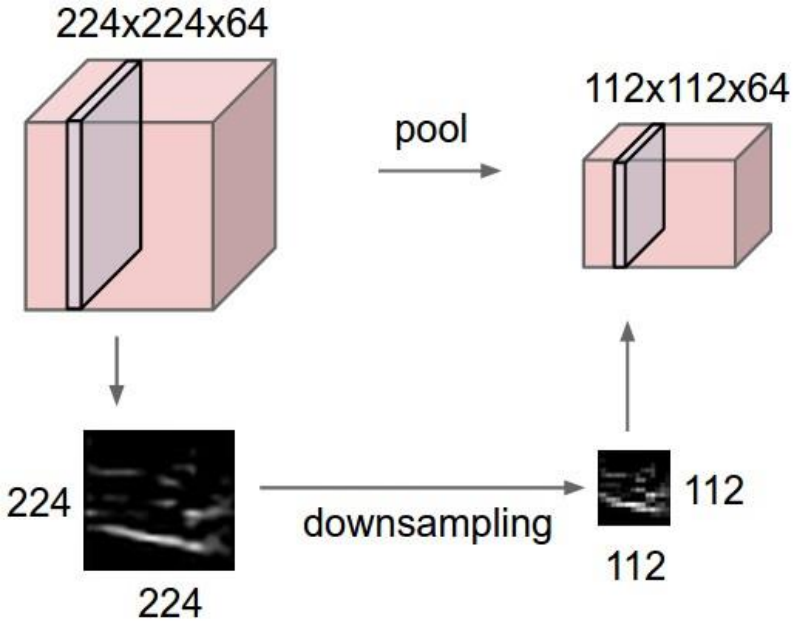
activity_regularizer: Regularizer function applied to the output of the layer (its "activation"). (see regularizer).

kernel_constraint: Constraint function applied to the kernel matrix (see constraints).

bias_constraint: Constraint function applied to the bias vector (see constraints).

MaxPooling Layers

```
# Maxpooling layer
model.add(MaxPooling2D(pool_size=(2, 2)))
# the only parameter here is the (spatial) size
to be reduced by the maximum operator
```



MaxPooling2D help

Arguments:

pool_size: integer or tuple of 2 integers, factors by which to downscale (vertical, horizontal). (2, 2) will halve the input in both spatial dimension. If only one integer is specified, the same window length will be used for both dimensions.

strides: Integer, tuple of 2 integers, or None. Strides values. If None, it will default to pool_size.

padding: One of "valid" or "same" (case-insensitive).

data_format: A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

MaxPooling2D help

Input shape:

If data_format='channels_last': 4D tensor with shape: (batch_size, rows, cols, channels)

If data_format='channels_first': 4D tensor with shape: (batch_size, channels, rows, cols)

Output shape:

If data_format='channels_last': 4D tensor with shape: (batch_size, pooled_rows, pooled_cols, channels)

If data_format='channels_first': 4D tensor with shape: (batch_size, channels, pooled_rows, pooled_cols)

Fully Connected Layers

```
# at the end the activation maps are "flattened" i.e.
they move from an image to a vector (just unrolling)
model.add(Flatten())
# Dense is a Fully Connected layer in a traditional
Neural Network.
model.add(Dense(units=10, activation='softmax'))
# Implements:
# output = activation(dot(input, kernel) + bias)
```

- activation is the element-wise activation function passed as the activation argument,
- kernel is a weights matrix created by the layer,
- bias is a bias vector created by the layer

```
# "Units" defines the number of neurons
```

Visualizing the model

```
# a nice output describing the model architecture
model.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 26, 26, 64)        640
_________________________________________________________________
flatten_3 (Flatten)          (None, 43264)             0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                432650
=================================================================
Total params: 433,290
Trainable params: 433,290
Non-trainable params: 0
```

Training the Model

Compiling the model

Then we need to compile the model using the compile method and specifying:

- **optimizer** which controls the learning rate. Adam is generally a good option as it adjusts the learning rate throughout training.
- **loss function** the most common choice for classification is ‘categorical_crossentropy’ for our loss function. The lower the better.
- **Metric** to assess model performance, ‘accuracy’ is more interpretable

```
model.compile(optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy'])
```

Training the model using

The **`fit()`** method of the model is used to train the model.

Specify at least the following inputs:

- training data (input images),
- target data (corresponding labels in categorical format),
- validation data (a pair of data, labels to be used only for computing validation performance)
- number of epochs (number of times the whole dataset is scanned during training)

```
model.fit(X_train, y_train,
validation_data=(X_test, y_test), epochs=3)
```

Training output

```
Epoch 22/100

18000/18000 [==============================] - 136s 8ms/step - loss: 0.7567
- acc: 0.6966 - val_loss: 1.9446 - val_acc: 0.4325

Epoch 23/100

18000/18000 [==============================] - 137s 8ms/step - loss: 0.7520
- acc: 0.6959 - val_loss: 1.9646 - val_acc: 0.4275

Epoch 24/100

18000/18000 [==============================] - 137s 8ms/step - loss: 0.7442
- acc: 0.7024 - val_loss: 1.9067 - val_acc: 0.4129
```

Advanced Training Options

Callbacks in Keras

A callback is a **set of functions** to be **applied at given stages** of the training procedure.

Callbacks give a **view on internal states** and statistics of the model **during training**.

You can pass **a list of callbacks** (as the keyword argument callbacks) to the .fit() method of the Sequential or Model classes.

The relevant methods of the callbacks will then be called at each stage of the training.

```
callback_list = [cb1,…,cbN]

model.fit(X_train, y_train,
validation_data=(X_test, y_test), epochs=3,
callbacks = callback_list)
```

Model Checkpoint

Training a network might take up to several hours

Checkpoints are snapshots of the state of the system to be saved in case of system failure.

When training a deep learning model, the checkpoint is the weights of the model. These weights can be used to make predictions as is, or used as the basis for ongoing training.

```
from keras.callbacks import ModelCheckpoint
```

[…]

```
cp = ModelCheckpoint(filepath,
monitor='val_loss', verbose=0,
save_best_only=False, save_weights_only=False,
mode='auto', period=1)
```

Early Stopping

The only stopping criteria when training a Deep Learning model is "reaching the required number of epochs."

However, it might be enough to train a model further, as sometimes the training error decreases but the validation error does not (overfitting)

Checkpoints are used to stop training when a monitored quantity has stopped improving.

```
from keras.callbacks import EarlyStopping
```

[…]

```
es = EarlyStopping(monitor='val_loss',
min_delta=0, patience=0, verbose=0, mode='auto',
baseline=None, restore_best_weights=False)
```

Testing the model

Predict() method

```
#returns the class probabilities for the input
image X_test
score = model.predict(X_test)
# select the class with the largest score
prediction_test = np.argmax(score, axis=1)
```

Tensorboard

When training a model it is important to monitor its progresses

Google has developed tensorboard a very useful tool for visualizing reports.

```
from keras.callbacks import TensorBoard
```

[…]

```
  tb = TensorBoard(log_dir="dirname")
```

… and add tb to the checkpoint list as well