

Introduction to Convolutional Neural Networks (CNN)

High-Dimensional-Deep-Learning – juliette.chevallier@insa-toulouse.fr

INSA Toulouse, ENSEEIHT, Modelling and Artificial Intelligence,

1. Introduction & Motivation

2. Convolution Layers

- 2.1 Convolution
- 2.2 Other Operations : Stride, Padding, Pooling, ...
- 2.3 Convolutional Neural Network

3. Convolutional Neural Networks and Image Classification

- 3.1 Main Architectures
- 3.2 ImageNet Challenge

4. Transfer Learning

- 4.1 Representational Learning
- 4.2 Transfer Learning

Some Classic Problems in Image Processing



Some Classic Problems in Image Processing



Classification

or: Annotation, *Labelling*

Main class: **Person**

Secondary classes:

Building, trees, grass, sky

Some Classic Problems in Image Processing



Classification

or: Annotation, *Labelling*

Main class: **Person**

Secondary classes:

Building, trees, grass, sky



Localization

Goal: Delineate the **position** of the object, using a *bounding box*.

Some Classic Problems in Image Processing



Classification

or: Annotation, *Labelling*

Main class: **Person**

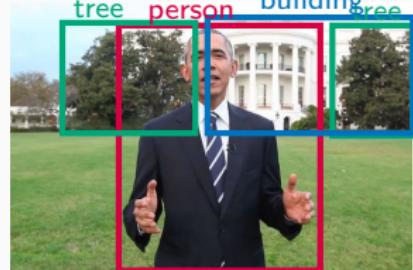
Secondary classes:

Building, trees, grass, sky



Localization

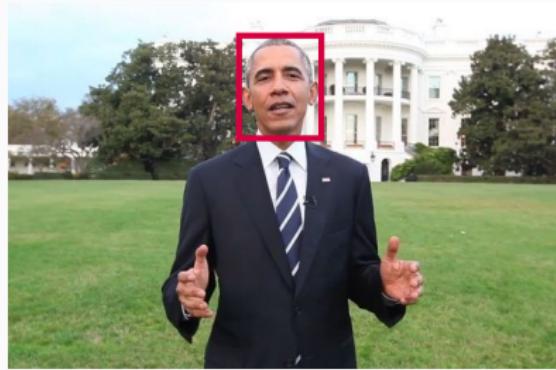
Goal: Delineate the **position** of the object, using a *bounding box*.



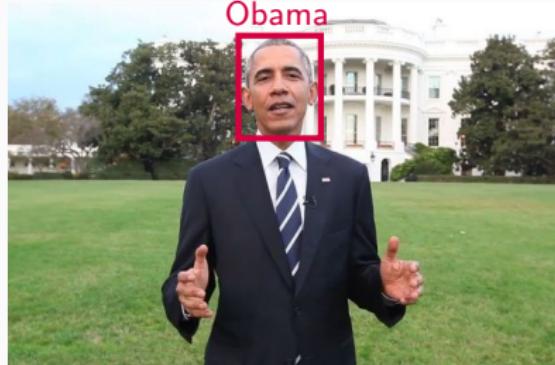
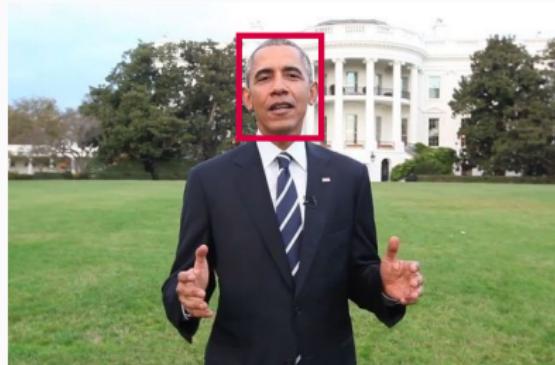
Object detection

Goal: Delineate the position of **multiple** objects, possibly with **multiple** instances, always using bounding boxes.

A Famous Sub-Problem of Object Detection: Face Detection



A Famous Sub-Problem of Object Detection: Face Detection ... and the recognition that goes with it!



Goal: Given a detected face and a face database, recognize the **person**.

Remark: This problem **cannot** be formulated as a classification problem: *Too many classes, and not enough examples.*

*This problem is a typical example of **few-shot learning**.*

Segmentation



Object Segmentation

Goal: Binary classification of image **pixels** as part of the object or background.

Segmentation



Object Segmentation

Goal: Binary classification of image **pixels** as part of the object or background.



Image Segmentation

ou *Image parsing*

Goal: Classification in n -area of the **pixels** of the image.

But Also

Pose estimation: Locating the “**joints**” or articulations of persons in images.

And many others, but rather using *generative models* like GAN:

Style transfer, B&W image colorization, Image reconstruction, Super-resolution image, Image synthesis, etc.

Dense Network Limit – MLP

Example : Image classification on *ImageNet*.

Images of size $224 \times 224 \times 3$, 1000 classes.

```
model = Sequential()
model.add(Input(shape=(224*224*3,)))
model.add(Dense(1000, activation='softmax'))
model.summary()
```

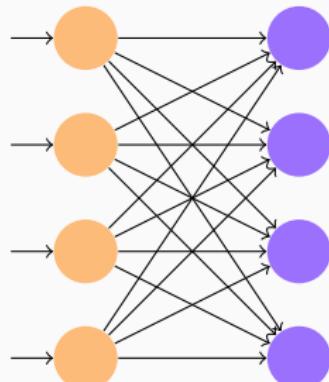
Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 1000)	150,529,000

Total params: 150,529,000 (574.22 MB)

Trainable params: 150,529,000 (574.22 MB)

Non-trainable params: 0 (0.00 B)



$224 \times 224 \times 3$ pixels

$224 \times 224 \times 3$

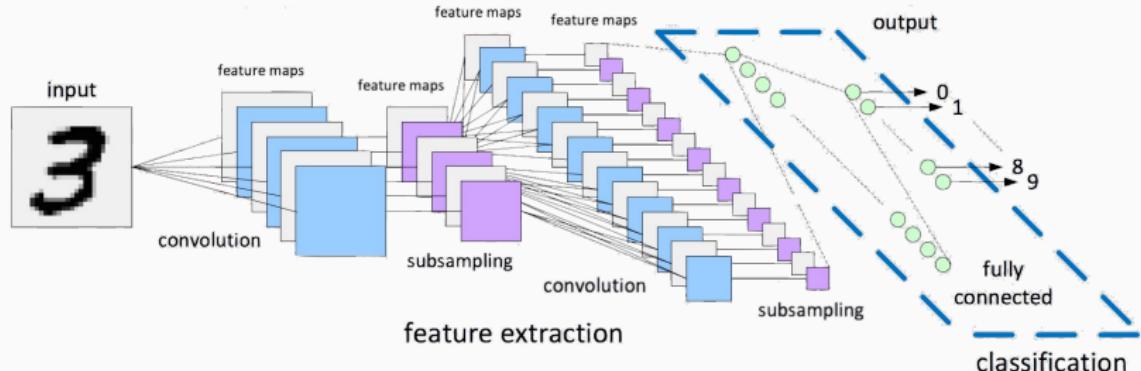
neurones

1000

neurones

- A single-layer perceptron requires $224 \times 224 \times 3 \times 1000 + 1000$ parameters, i.e more than 150 millions of parameters !
- Spatial information is lost with *flattening*.

Typical Architecture of a Convolutional Neural Network



3 types of Layers:

- *Firsts layers:* Convolution layers combined with pooling layers,
- *Last layers:* Fully-connected or dense layers.

Convolution Layers

2.1 Convolution

2.2 Other Operations : Stride, Padding, Pooling, ...

2.3 Convolutional Neural Network

Convolution

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

*

-1	-2	-1
0	0	0
1	2	1

=

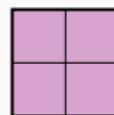


Image 4×4

Filter 3×3
 $f = 3$

Response 2×2

Convolution

1 ₋₁	1 ₋₂	0 ₋₁	1
0 ₀	0 ₀	0 ₀	1
1 ₁	1 ₂	1 ₁	0
1	0	0	1

⊗

-1	-2	-1
0	0	0
1	2	1

=

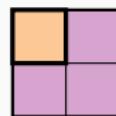


Image 4×4

Filter 3×3
 $f = 3$

Response 2×2

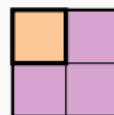
Convolution

1 ₋₁	1 ₋₂	0 ₋₁	1
0 ₀	0 ₀	0 ₀	1
1 ₁	1 ₂	1 ₁	0
1	0	0	1

⊗

-1	-2	-1
0	0	0
1	2	1

=

Image 4×4 Filter 3×3
 $f = 3$ Response 2×2

$$-1 \times 1 - 2 \times 1 - 1 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 1 \times 1 + 2 \times 1 + 1 \times 1 = 1$$

Convolution

1 ₋₁	1 ₋₂	0 ₋₁	1
0 ₀	0 ₀	0 ₀	1
1 ₁	1 ₂	1 ₁	0
1	0	0	1

⊗

-1	-2	-1
0	0	0
1	2	1

=

1	

Image 4×4 Filter 3×3
 $f = 3$ Response 2×2

$$-1 \times 1 - 2 \times 1 - 1 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 1 \times 1 + 2 \times 1 + 1 \times 1 = 1$$

Convolution

1	1 ₋₁	0 ₋₂	1 ₋₁
0	0 ₀	0 ₀	1 ₀
1	1 ₁	1 ₂	0 ₁
1	0	0	1

⊗

-1	-2	-1
0	0	0
1	2	1

=

1	1

Image 4×4 Filter 3×3
 $f = 3$ Response 2×2

$$-1 \times 1 - 2 \times 0 - 1 \times 1 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 1 \times 1 + 2 \times 1 + 1 \times 0 = 1$$

Convolution

$$\begin{array}{|c|c|c|c|} \hline 1 & 1 & 0 & 1 \\ \hline 0_{-1} & 0_{-2} & 0_{-1} & 1 \\ \hline 1_0 & 1_0 & 0_0 & 0 \\ \hline 1_1 & 0_2 & 0_1 & 1 \\ \hline \end{array} \quad * \quad \begin{array}{|c|c|c|} \hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 1 & 1 \\ \hline 1 & \\ \hline \end{array}$$

Image 4×4 Filter 3×3 Response 2×2

$$f = 3$$

$$-1 \times 0 - 2 \times 0 - 1 \times 1 + 0 \times 1 + 0 \times 1 + 0 \times 0 + 1 \times 1 + 2 \times 0 + 1 \times 0 = 1$$

Convolution

$$\begin{array}{|c|c|c|c|} \hline 1 & 1 & 0 & 1 \\ \hline 0 & 0_{-1} & 0_{-2} & 1_{-1} \\ \hline 1 & 0_0 & 0_0 & 1_0 \\ \hline 1 & 0_1 & 1_2 & 1_1 \\ \hline \end{array} \quad * \quad \begin{array}{|c|c|c|} \hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 1 & 1 \\ \hline 1 & 0 \\ \hline \end{array}$$

Image 4×4 Filter 3×3 Response 2×2

$$f = 3$$

$$-1 \times 0 - 2 \times 0 - 1 \times 1 + 0 \times 0 + 0 \times 0 + 0 \times 1 + 1 \times 0 + 2 \times 1 + 1 \times 1 = 0$$

Convolution

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

⊗

-1	-2	-1
0	0	0
1	2	1

=

1	1
1	0

Image 4×4

Filter 3×3
 $f = 3$

Response 2×2

2D Discrete Convolution

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

*

-1	-2	-1
0	0	0
1	2	1

=

1	1
1	0

Image 4×4

Filter 3×3

$$f = 3$$

Response 2×2

2D discrete convolution:

$$(F * I)(x, y) = \sum_{m,n} F(n, m) \cdot I(x-m, y-n)$$

2D Discrete Convolution

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

*

-1	-2	-1
0	0	0
1	2	1

=

1	1
1	0

Image 4×4

Filter 3×3

$$f = 3$$

Response 2×2

Given :

- A grayscale image I (a single *color channel*) of dimension $w \times h$,
- A filter F of size f ,

↝ Dimension of the response $F * I$:

$$(w - f + 1) \times (h - f + 1)$$

2D discrete convolution:

$$(F * I)(x, y) = \sum_{m,n} F(n, m) \cdot I(x-m, y-n)$$

2D Discrete Convolution

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

*

-1	-2	-1
0	0	0
1	2	1

=

1	1
1	0

Image 4×4

Filter 3×3

$$f = 3$$

Response 2×2

Given :

- A grayscale image I (a single *color channel*) of dimension $w \times h$,
- A filter F of size f ,

~ Dimension of the response $F * I$:

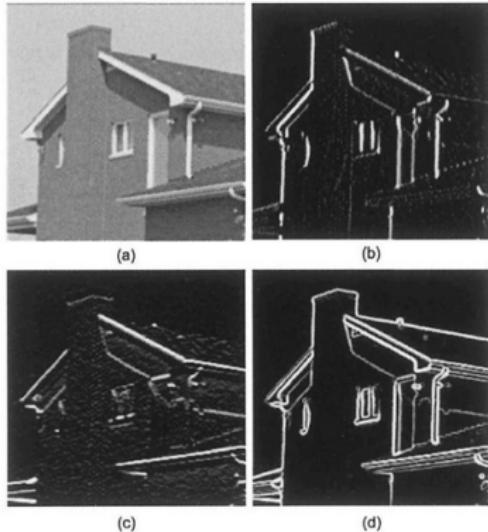
$$(w - f + 1) \times (h - f + 1)$$

2D discrete convolution:

$$(F * I)(x, y) = \sum_{m,n} F(n, m) \cdot I(x-m, y-n)$$

- takes into account the **spatial organization**,
- translation invariant,
- parameter sharing.

Convolution in Computer Vision



-1	0	+1
-2	0	+2
-1	0	+1

Gx

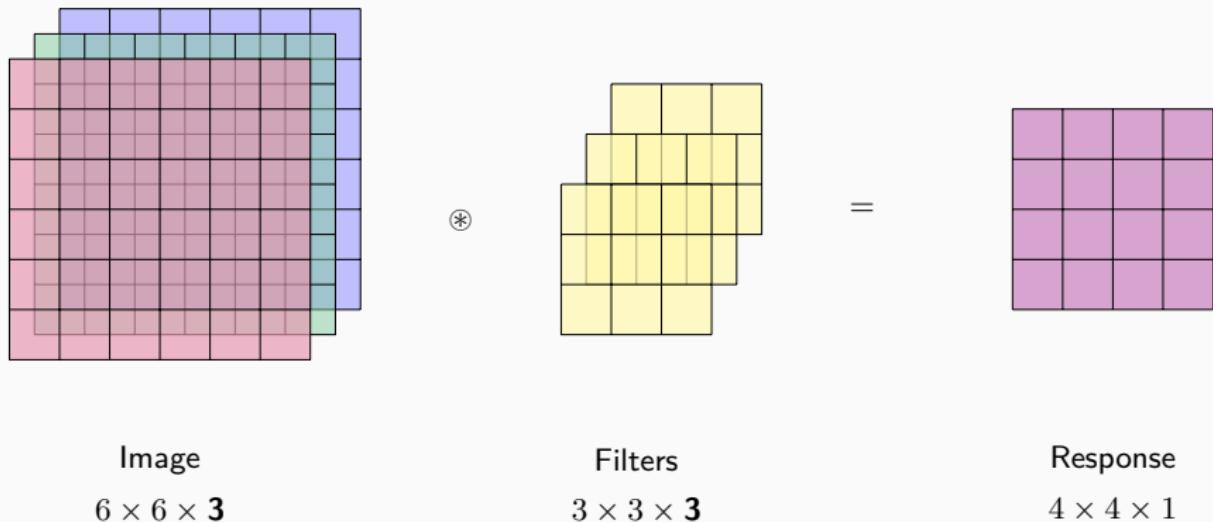
+1	+2	+1
0	0	0
-1	-2	-1

Gy

- Convolution filters (or kernels) have long been used to detect **patterns** in images, such as contours (here: *Sobel filters*);
- A white pixel indicates a high filter response. Here, this means a pixel located on the contour of an object, with a strong local gradient.

2D Volume Convolution: Multiple Color Channels

Consider a colored image, in 3 dimensions: height, width, channels (RGB).



*The number of **channels** in the input image and the **depth** of the convolution filters are necessarily identical.*

2D Volume Convolution: Multiple Color Channels

Consider a colored image, in 3 dimensions: height, width, channels (RGB).

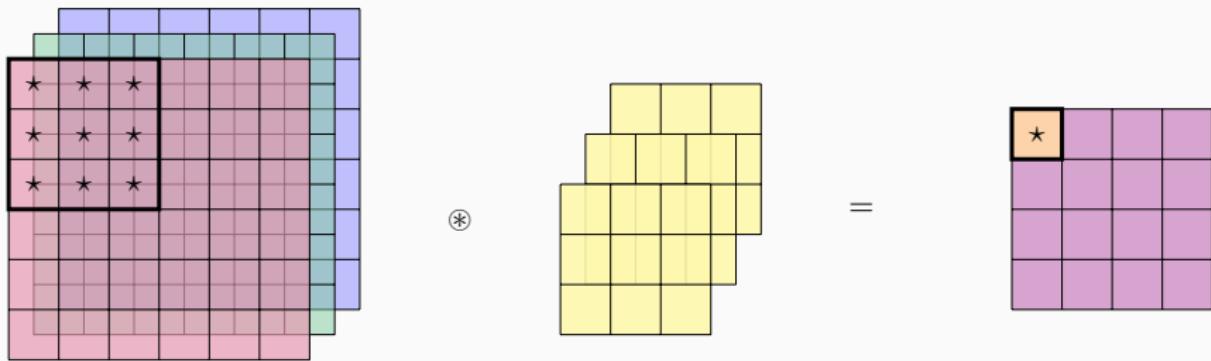


Image
 $6 \times 6 \times 3$

Filters
 $3 \times 3 \times 3$

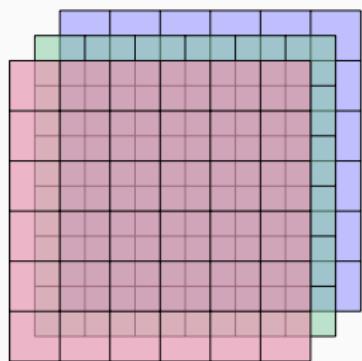
Response
 $4 \times 4 \times 1$

$$(F * I)(x, y) = \sum_{\textcolor{red}{c}} \sum_m \sum_n F^{\textcolor{red}{c}}(n, m) \cdot I^{\textcolor{red}{c}}(x - m, y - n)$$

The number of *channels* in the input image and the *depth* of the convolution filters are necessarily identical.

2D Volume Convolution Layer

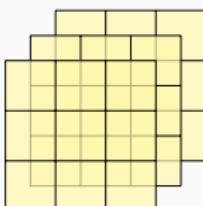
Consider a colored image, in 3 dimensions: height, width, channels (RGB).



Image

$$6 \times 6 \times 3$$

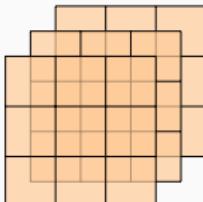
*



Filtres

$$3 \times 3 \times 3 \times 2$$

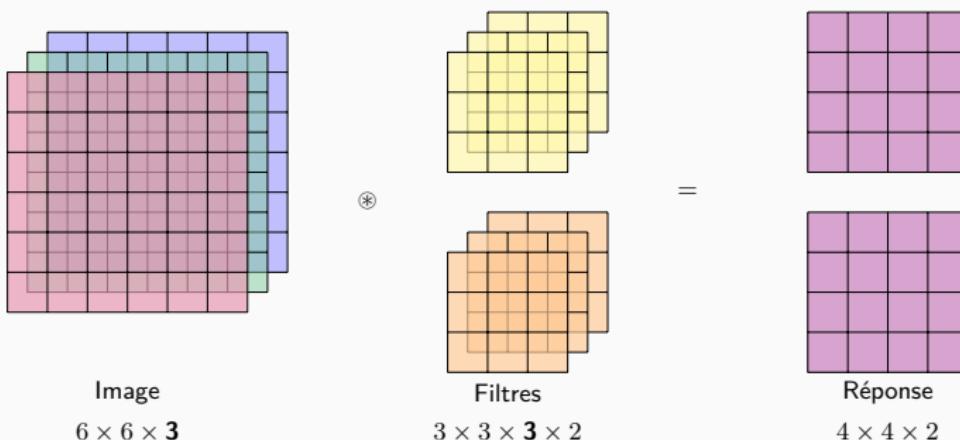
=



Réponse

$$4 \times 4 \times 2$$

Number of Parameters in a Convolution Layer

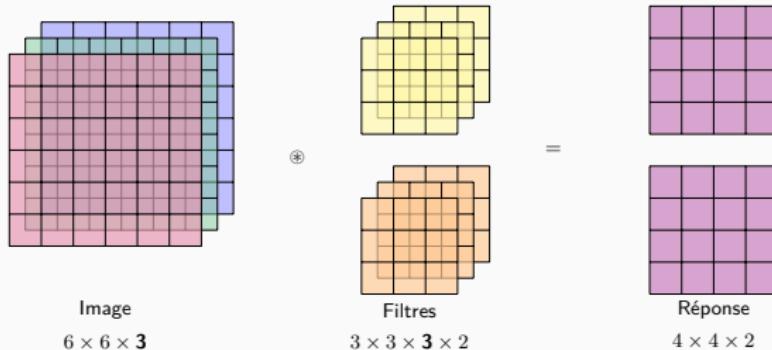


Two types of parameters in a convolution layer :

- The **coefficients of the convolution filters**: $f \times f \times \# \text{channels} \times \# \text{filters}$,
- The **biases** added to the *response* of the convolution filters, before application of the activation function. There is exactly one bias per filter, i.e. $\# \text{filters}$.

~ In the above example, there are $3 \times 3 \times 3 \times 2 + 2 = 56$ parameters.

Number of Operations in a Convolution Layer



It is interesting to count the number of operations of a neural network to characterize its **complexity**, and the required **resources** required for its execution. We focus mainly on **addition** and **multiplication**.

Here, each element of the answer requires :

- $(f \times f \times \# \text{channels})$ multiplications,
- $(f \times f \times \# \text{channels} - 1)$ additions between multiplied values,
- 1 additional addition for the bias.

~~~ In the above example, there are  $(4 \times 4 \times 2) \times (3 \times 3 \times 3 \times 2) = 1728$  operations.

**CPU:**

- Few cores ...
- ... but extremely complexe and fast.
- Shared memory system.

~> *Sequential tasks*

**GPU:**

- A lot of cores ...
- ... but simple and slow.
- Not shared memory system

~> *Parallel tasks*

**CPU:**

- Few cores ...
- ... but extremely complexe and fast.
- Shared memory system.

~~~ *Sequential tasks*

GPU:

- A lot of cores ...
- ... but simple and slow.
- Not shared memory system

~~~ *Parallel tasks*

- 
- GPU is not adapted to all ML algorithms
  - Data loading time to GPU can be really high
    - GPU is useful only if computation time exceed loading time
    - i.e. useful for *complex model*

## Convolution Layers

---

2.1 Convolution

2.2 Other Operations : Stride, Padding, Pooling, ...

2.3 Convolutional Neural Network

## Stride

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 3 | 1 | 3 | 5 | 3 | 3 |
| 2 | 2 | 8 | 8 | 3 | 9 |
| 3 | 4 | 7 | 7 | 2 | 7 |
| 5 | 3 | 6 | 8 | 4 | 7 |
| 3 | 8 | 8 | 5 | 7 | 4 |
| 7 | 9 | 6 | 4 | 6 | 9 |

Image

$$\begin{matrix} & \begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix} \end{matrix} = \begin{matrix} 10 & 13 & 7 & 4 \\ 3 & -3 & -1 & 0 \\ 9 & 4 & 2 & 5 \\ 14 & 2 & -6 & 2 \end{matrix}$$

\*

Filter  
 $f = 3$

Response

$stride = 1$

## Stride

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 3 | 1 | 3 | 5 | 3 | 3 |
| 2 | 2 | 8 | 8 | 3 | 9 |
| 3 | 4 | 7 | 7 | 2 | 7 |
| 5 | 3 | 6 | 8 | 4 | 7 |
| 3 | 8 | 8 | 5 | 7 | 4 |
| 7 | 9 | 6 | 4 | 6 | 9 |

Image

$$\begin{matrix} & \ast & & = & \\ \begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix} & & & & \begin{matrix} 10 & 7 \\ 9 & 2 \end{matrix} \end{matrix}$$

*stride = 2*

Filter  
 $f = 3$

Response

- Reduces the dimension of tensors,
- Limiting the loss of information:  
*Same coefficient influences several elements of the response to the convolution filter.*

## Stride

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 3 | 1 | 3 | 5 | 3 | 3 |
| 2 | 2 | 8 | 8 | 3 | 9 |
| 3 | 4 | 7 | 7 | 2 | 7 |
| 5 | 3 | 6 | 8 | 4 | 7 |
| 3 | 8 | 8 | 5 | 7 | 4 |
| 7 | 9 | 6 | 4 | 6 | 9 |

Image

$$\begin{matrix} & \ast & \\ \begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix} & = & \begin{matrix} 10 & 7 \\ 9 & 2 \end{matrix} \end{matrix}$$

stride = 2

Filter  
 $f = 3$

Response

- Reduces the dimension of tensors,
- Limiting the loss of information:  
*Same coefficient influences several elements of the response to the convolution filter.*

Given:

- A grayscale image  $I$  (a single *color channel*) of size  $w \times h$ ,
  - A filter  $F$  of size  $f$ , and *stride*  $s$ ,
- Dimension of the response  $F \circledast I$ :

$$\left\lfloor \frac{w-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{h-f}{s} + 1 \right\rfloor$$

## Padding

$$\begin{array}{|c|c|c|c|} \hline 1 & 1 & 0 & 1 \\ \hline 0 & 0 & 0 & 1 \\ \hline 1 & 1 & 1 & 0 \\ \hline 1 & 0 & 0 & 1 \\ \hline \end{array} \quad \circledast \quad \begin{array}{|c|c|c|} \hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array} \quad = \quad \begin{array}{|c|c|} \hline 1 & 1 \\ \hline 1 & 0 \\ \hline \end{array}$$

Image

Filter

Response

## Padding

$$\begin{array}{|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 1 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 1 & 1 & 1 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array} \quad \circledast \quad \begin{array}{|c|c|c|} \hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline 0 & 0 & 1 & 2 \\ \hline 0 & 1 & 1 & -1 \\ \hline 2 & 1 & 0 & 0 \\ \hline -3 & -4 & -3 & -1 \\ \hline \end{array}$$

Image                          Filter                          Response  
 $p = 1$

- **Zero-padding:** Add zeros to image edges,
- For example, allow to obtain a response of the same dimension as the input image (**same** convolution)
  - ~~ Facilitates the chaining of convolution layers in a neural network.

## Padding

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

⊗

|    |    |    |
|----|----|----|
| -1 | -2 | -1 |
| 0  | 0  | 0  |
| 1  | 2  | 1  |

=

|    |    |    |    |
|----|----|----|----|
| 0  | 0  | 1  | 2  |
| 0  | 1  | 1  | -1 |
| 2  | 1  | 0  | 0  |
| -3 | -4 | -3 | -1 |

Image                          Filter                          Response

$p = 1$

- **Zero-padding:** Add zeros to image edges,
- For example, allow to obtain a response of the same dimension as the input image (**same convolution**)
  - ~~ Facilitates the chaining of convolution layers in a neural network.

Given

- A grayscale image  $I$  (a single *color channel*) of size  $w \times h$ ,
  - A filter  $F$  of size  $f$ , and **padding**  $p$ ,
- ~~ Dimension of the response  $F \circledast I$ :

$$(w + 2p - f + 1) \times (h + 2p - f + 1)$$

## Impact on Tensor Dimension

The diagram illustrates a convolution operation. On the left, an **Image** of size  $6 \times 6$  is shown with padding  $p = 1$ . The image is divided into a  $3 \times 3$  grid of blocks, each of size  $2 \times 2$ . The top-left block is green, followed by purple, then yellow, then blue. The middle row has a green block at index (1,1), a yellow block at (2,1), a blue block at (3,1), and a yellow block at (4,1). The bottom row has a blue block at (1,1), a yellow block at (2,1), a blue block at (3,1), and a yellow block at (4,1). The rightmost column has a green block at (1,1), a yellow block at (2,1), a blue block at (3,1), and a yellow block at (4,1). The bottom-right block is yellow. In the center, there is a multiplication symbol ( $\circledast$ ) indicating the convolution operation. To the right of the multiplication symbol is a **Filter** of size  $3 \times 3$  with stride  $s = 2$ . The filter has values: -1, -2, -1 in the first row; 0, 0, 0 in the second row; and 1, 2, 1 in the third row. To the right of the filter is an equals sign (=). To the right of the equals sign is the resulting **Response**, which is a  $2 \times 2$  matrix with values 1, 0 in the top row and 0, 2 in the bottom row.

Image  
 $p = 1$

$\circledast$

Filter  
 $stride = 2$

=

Response

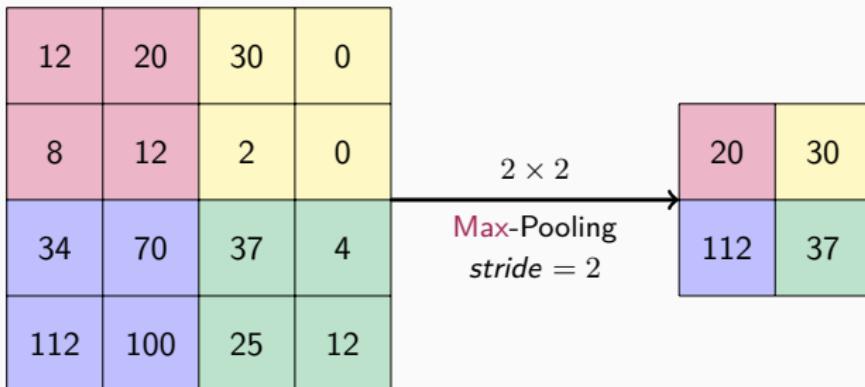
Given:

- A grayscale image  $I$  (a single *color channel*) of size  $w \times h$ ,
- A filter  $F$  of size  $f$ , **padding**  $p$  and **stride**  $s$

↝ Dimension of the response  $F \circledast I$  of image  $I$  to filter  $F$ :

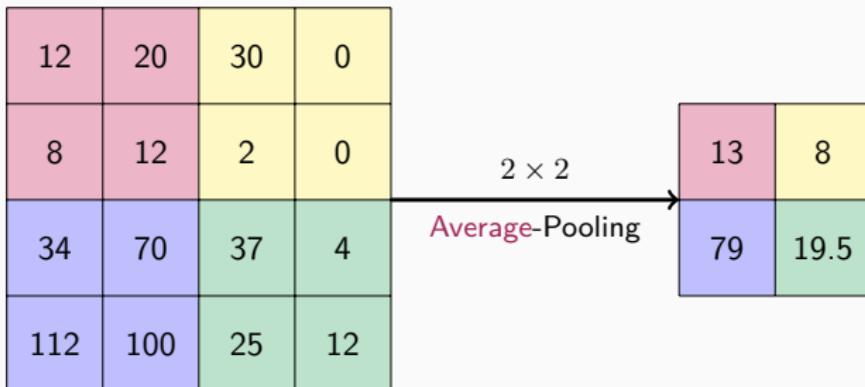
$$\left\lfloor \frac{w + 2p - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{h + 2p - f}{s} + 1 \right\rfloor$$

## Pooling Layer



- Reduces tensor dimensions, to counterbalance the multiplication of convolution filter responses,
- Idea: Preserve the high responses to convolution filters,
- Translation invariant,
- No parameters to learn!

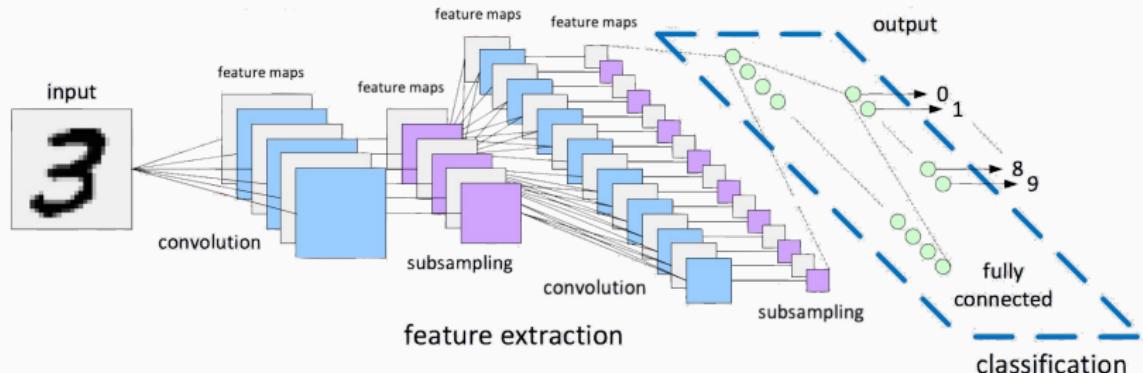
## Pooling Layer



- Reduces tensor dimensions, to counterbalance the multiplication of convolution filter responses,
- Idea: Preserve the **high responses** to convolution filters,
- Translation invariant,
- No parameters to learn!

- **Average-Pooling** : Alternatively, we can average the values rather than keeping the maximum,
- **But**: Standard architectures favor **Max-Pooling**.

# Typical Architecture of a Convolutional Neural Network (back again)



## 3 type of Layers:

- *Firsts layers:* Convolution layers combined with pooling layers,
- *Last layers:* Fully-connected or dense layers.

## **Convolution Layers**

---

2.1 Convolution

2.2 Other Operations : Stride, Padding, Pooling, ...

**2.3 Convolutional Neural Network**

## Advantages of Convolutional Neural Networks

### **Parameter sharing :**

- Convolution layer : Equivalent to a dense layer in which some synaptic weights are **shared**, and the majority is zero.
- These weights, i.e. the convolution coefficients, may be adjusted *multiple times* for the *same example* in the training database.  
~~> A **much smaller number of parameters** for a convolutional network than for a fully connected network.

## A Look Back at Image Classification on ImageNet

Images of size  $224 \times 224 \times 3$ , 1000 classes.

```
model = Sequential()
model.add(Input(shape=(224,224,3)))
model.add(Conv2D(1000, kernel_size=(3,3), activation='relu'))
model.summary()
```

Model: "sequential\_2"

| Layer (type)      | Output Shape           | Param # |
|-------------------|------------------------|---------|
| conv2d_2 (Conv2D) | (None, 222, 222, 1000) | 28,000  |

Total params: 28,000 (109.38 KB)

Trainable params: 28,000 (109.38 KB)

Non-trainable params: 0 (0.00 B)

- With a perceptron *mono*-layer, more than 150 **millions** of parameters,
- With a simple convolution layer, size 3 filter, a single filter:  
“only” 28 000 ( $= (3^3 + 1) \times 1000$ ) parameters.

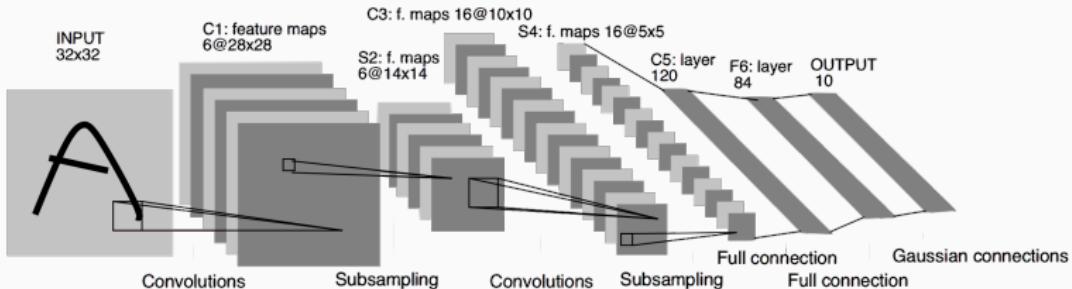
## CNNs and Image Classification

---

### 3.1 Main Architectures

### 3.2 ImageNet Challenge

# A Pioneer : LeNet (1998)



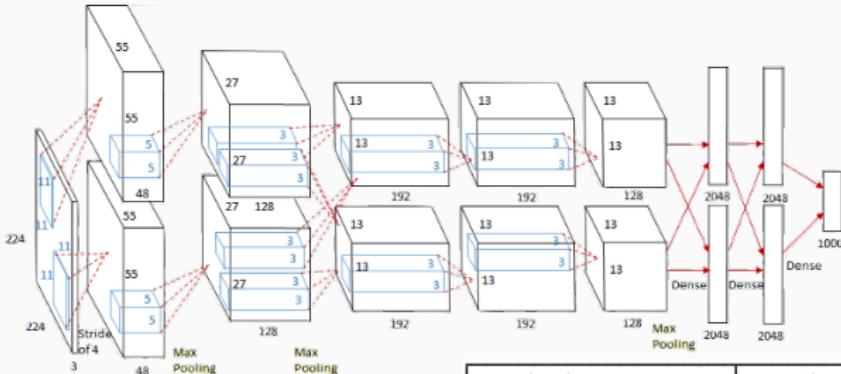
- $\approx 60k$  parameters,
- 2 to 3 days of training for 20 epochs on MNIST (in 1998!),
- Sigmoid or tanh activation functions.

[LeCun et al.] Gradient-based learning applied to document recognition.

| Layer (type)                           | Output Shape       | Param # |
|----------------------------------------|--------------------|---------|
| conv2d (Conv2D)                        | (None, 28, 28, 6)  | 156     |
| average_pooling2d (AveragePooling2D)   | (None, 14, 14, 6)  | 0       |
| activation (Activation)                | (None, 14, 14, 6)  | 0       |
| conv2d_1 (Conv2D)                      | (None, 10, 10, 16) | 2,416   |
| average_pooling2d_1 (AveragePooling2D) | (None, 5, 5, 16)   | 0       |
| activation_1 (Activation)              | (None, 5, 5, 16)   | 0       |
| conv2d_2 (Conv2D)                      | (None, 1, 1, 120)  | 48,120  |
| flatten (Flatten)                      | (None, 120)        | 0       |
| dense (Dense)                          | (None, 84)         | 10,164  |
| dense_1 (Dense)                        | (None, 10)         | 850     |

Total params: 61,706 (241.04 KB)  
Trainable params: 61,706 (241.04 KB)  
Non-trainable params: 0 (0.00 B)

# The Rocker : AlexNet (2012)



- $\approx$  60M parameters, 8 layers,
- Architecture designed to be deployed on 2 GPUs,
- Introduces the use of the **ReLU** function as a standard for deep network training.

[Krizhevsky et al.] ImageNet Classification with Deep Convolutional Neural Networks

| Layer (type)                   | Output Shape        | Param #    |
|--------------------------------|---------------------|------------|
| conv2d (Conv2D)                | (None, 54, 54, 96)  | 34,944     |
| max_pooling2d (MaxPooling2D)   | (None, 27, 27, 96)  | 0          |
| conv2d_1 (Conv2D)              | (None, 17, 17, 256) | 2,973,952  |
| max_pooling2d_1 (MaxPooling2D) | (None, 8, 8, 256)   | 0          |
| conv2d_2 (Conv2D)              | (None, 6, 6, 384)   | 885,120    |
| conv2d_3 (Conv2D)              | (None, 4, 4, 384)   | 1,327,488  |
| conv2d_4 (Conv2D)              | (None, 2, 2, 256)   | 884,992    |
| max_pooling2d_2 (MaxPooling2D) | (None, 1, 1, 256)   | 0          |
| flatten (Flatten)              | (None, 256)         | 0          |
| dense (Dense)                  | (None, 4096)        | 1,052,672  |
| dropout (Dropout)              | (None, 4096)        | 0          |
| dense_1 (Dense)                | (None, 4096)        | 16,781,312 |
| dropout_1 (Dropout)            | (None, 4096)        | 0          |
| dense_2 (Dense)                | (None, 1000)        | 4,097,000  |
| dropout_2 (Dropout)            | (None, 1000)        | 0          |
| dense_3 (Dense)                | (None, 17)          | 17,017     |

Total params: 28,854,497 (107.02 MB)

# The Rocker : AlexNet (2012)

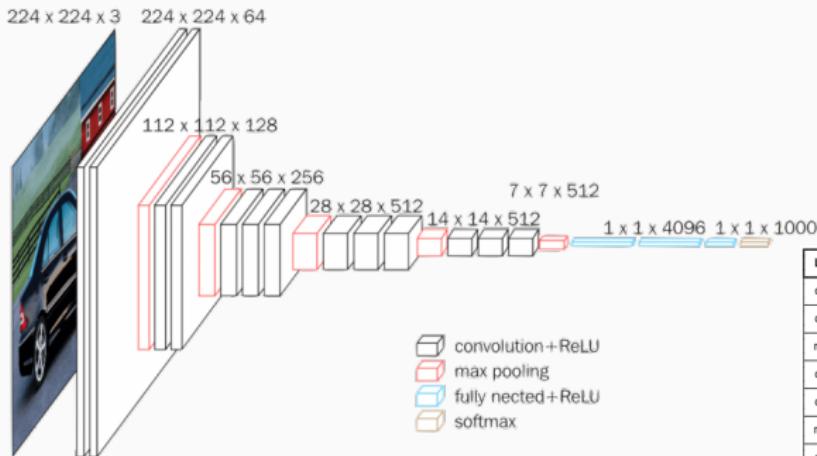


## Observations :

- Gradual reduction of *filter* size:  $11 \rightarrow 5 \rightarrow 3$ ;
- Gradual reduction of *image* size:  $224 \rightarrow 55 \rightarrow 27 \rightarrow 13$ ;
- Gradual increase in the number of filters:  $96 \rightarrow 256 \rightarrow 384$  ;
- *Stride then Max Pooling*

[Krizhevsky et al.] ImageNet Classification with Deep Convolutional Neural Networks.

# A “simplified” Version: VGG-16 (2014)



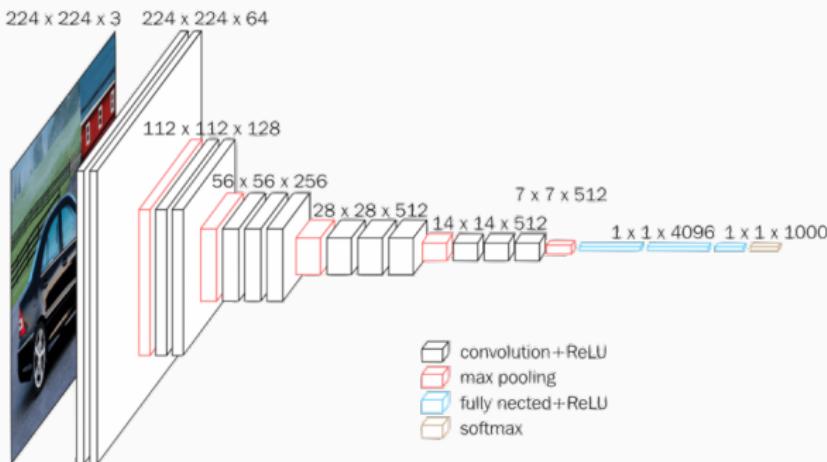
$\simeq 138M$  parameters, 16 layers in its standard version.

[Simonyan et Zisserman] Very Deep Convolutional Networks for Large-Scale Image Recognition

| Layer (type)                   | Output Shape          | Param #     |
|--------------------------------|-----------------------|-------------|
| conv2d (Conv2D)                | (None, 224, 224, 64)  | 1,792       |
| conv2d_1 (Conv2D)              | (None, 224, 224, 64)  | 36,928      |
| max_pooling2d (MaxPooling2D)   | (None, 112, 112, 64)  | 0           |
| conv2d_2 (Conv2D)              | (None, 112, 112, 128) | 73,856      |
| conv2d_3 (Conv2D)              | (None, 112, 112, 128) | 147,584     |
| max_pooling2d_1 (MaxPooling2D) | (None, 56, 56, 128)   | 0           |
| conv2d_4 (Conv2D)              | (None, 56, 56, 256)   | 295,168     |
| conv2d_5 (Conv2D)              | (None, 56, 56, 256)   | 590,080     |
| conv2d_6 (Conv2D)              | (None, 56, 56, 256)   | 590,080     |
| max_pooling2d_2 (MaxPooling2D) | (None, 28, 28, 256)   | 0           |
| conv2d_7 (Conv2D)              | (None, 28, 28, 512)   | 1,180,160   |
| conv2d_8 (Conv2D)              | (None, 28, 28, 512)   | 2,359,880   |
| conv2d_9 (Conv2D)              | (None, 28, 28, 512)   | 2,359,880   |
| max_pooling2d_3 (MaxPooling2D) | (None, 14, 14, 512)   | 0           |
| conv2d_10 (Conv2D)             | (None, 14, 14, 512)   | 2,359,880   |
| conv2d_11 (Conv2D)             | (None, 14, 14, 512)   | 2,359,880   |
| conv2d_12 (Conv2D)             | (None, 14, 14, 512)   | 2,359,880   |
| max_pooling2d_4 (MaxPooling2D) | (None, 7, 7, 512)     | 0           |
| flatten (Flatten)              | (None, 25088)         | 0           |
| dense (Dense)                  | (None, 4096)          | 182,764,544 |
| dense_1 (Dense)                | (None, 4096)          | 16,781,312  |
| dense_2 (Dense)                | (None, 17)            | 69,645      |

Total params: 134,330,193 (512.43 MB)

## A “simplified” Version: VGG-16 (2014)

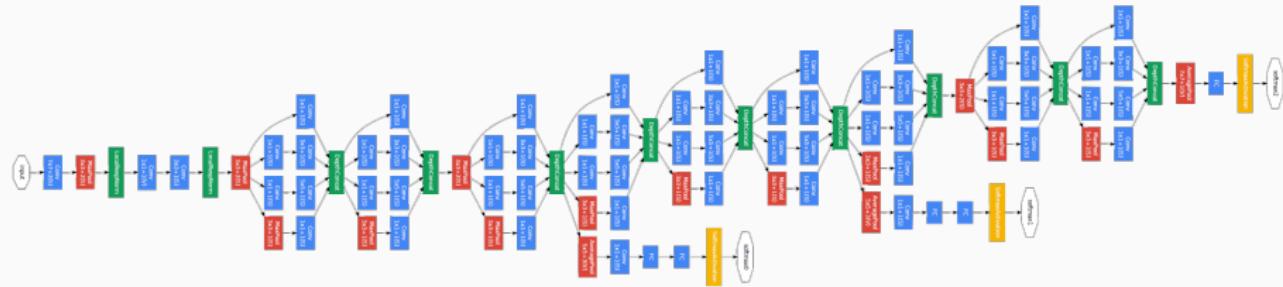


**Objective** : Study the impact of *depth* on network performance.

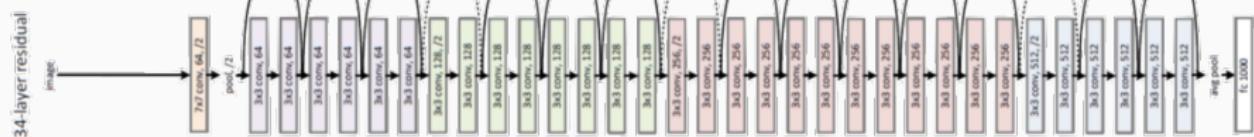
↝ To this end, the authors made the network **architecture very regular**:

- Systematic use of convolutions  $3 \times 3$ ,
- The main features of AlexNet are retained, but regularized:
  - *Gradual reduction in image size*:  $224 \rightarrow 112 \rightarrow 56 \dots$
  - *Gradual increase in the number of filters*:  $64 \rightarrow 128 \rightarrow 256$

## More Advanced Architectures: Inception & ResNet



[Szegedy et al.] Going deeper with convolutions.



[He et al.] Deep Residual Learning for Image Recognition

## **CNNs and Image Classification**

---

**3.1 Main Architectures**

**3.2 ImageNet Challenge**

# 2012-1017 : Large Scale Visual Recognition Challenge

IMAGENET

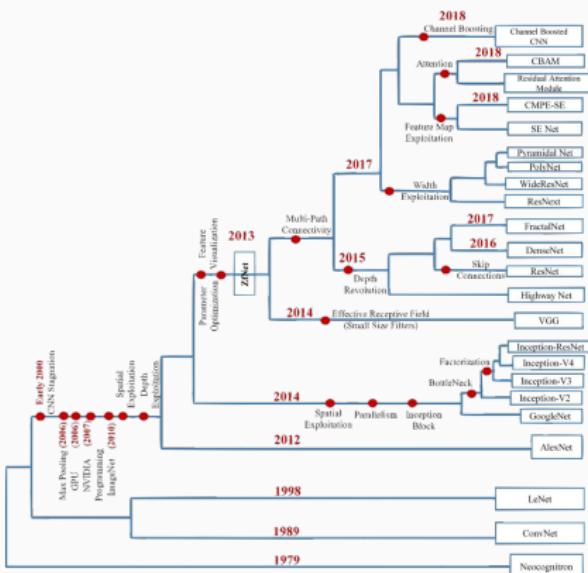
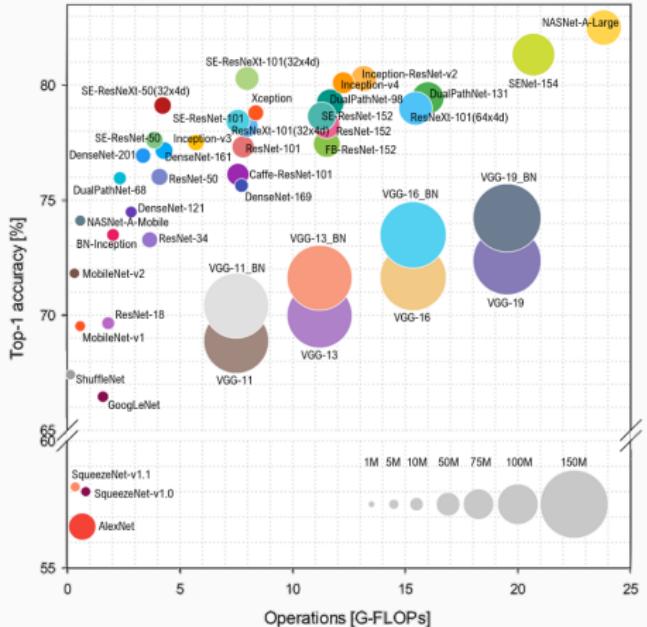
- 1000 object classes,
- 1 281 167 training images,
- 50 000 validation images,
- 100,000 test images.



ImageNet Large Scale Visual Recognition Challenge – ILSVRC



From 2015



[Bianco et al.] Benchmark Analysis of Representative Deep Neural Network Architectures (2018)

## **Transfer Learning**

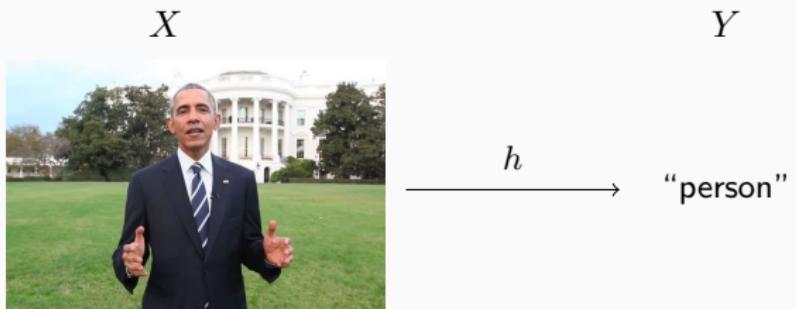
---

### **4.1 Representational Learning**

### **4.2 Transfer Learning**

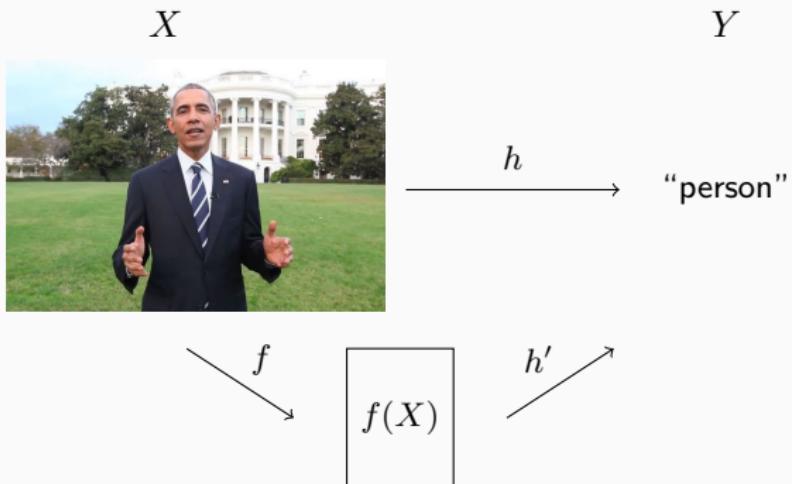
# Representation Learning

**Aim:** *Image classification*



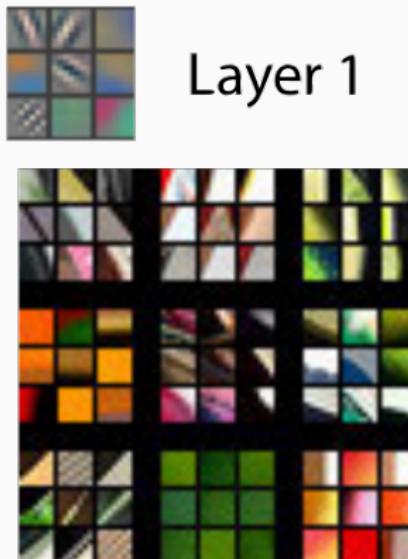
# Representation Learning

**Aim:** *Image classification*



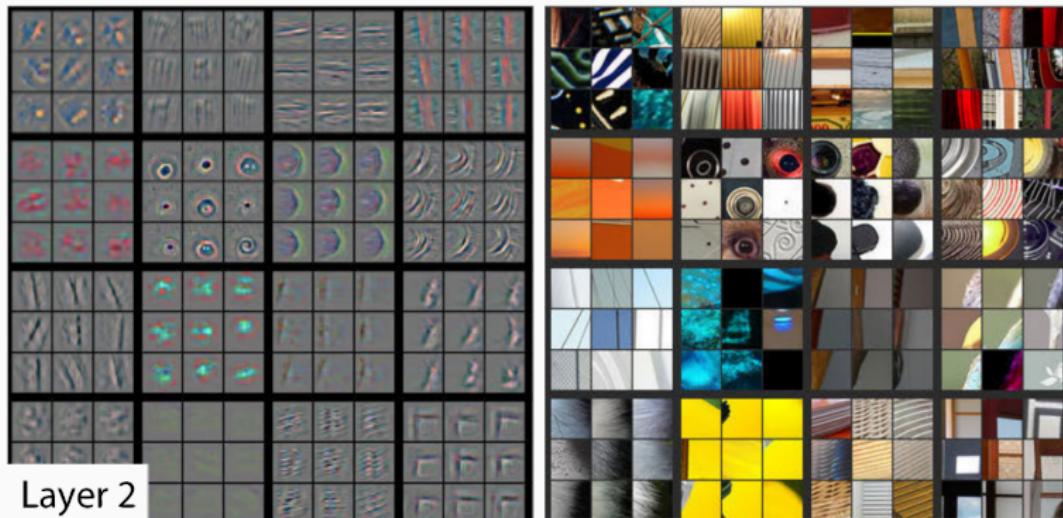
**Feature learning:**

## What do Convolutional Networks “Learn”?



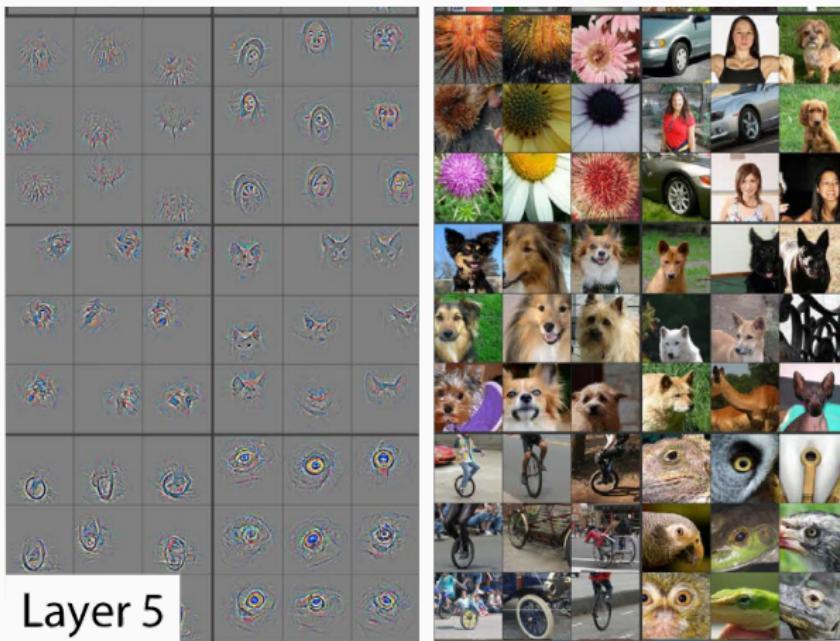
- Example of filters learned on the **first layer** of a network (similar to AlexNet),
- For each filter, listing of the 9 image patches with the **highest filter activation**.

# What do Convolutional Networks “Learn”?



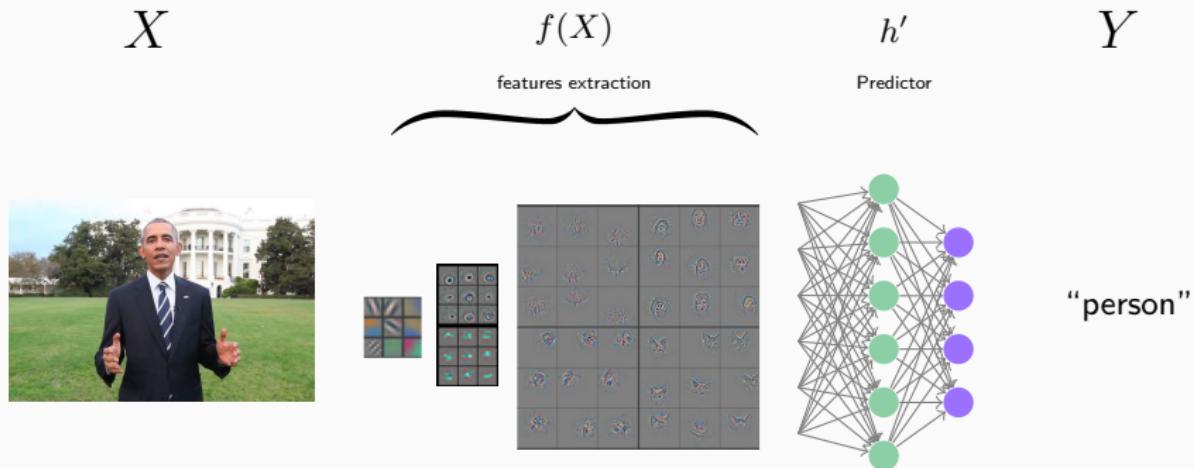
- Same visualization as above, but for **second-layer** filters,
- Note that the filters on the left-hand side are not present in the network as they are, but that *this visual representation is reconstructed*.

# What do Convolutional Networks “Learn”?



The patterns detected are of a higher *semantic level* as you progress through the network layers.

# Representation Learning: An Interpretation of CNNs



A CNN can be interpreted in terms of **representational learning**:

- The convolutional part is a **feature extractor**  $f(X)$  ...
- ... and dense top layers are a **predictor**  $h'$ .

*In this way, deep learning can learn features in addition to the predictor!*

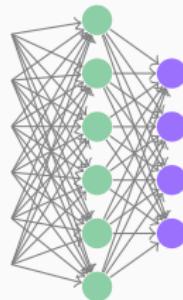
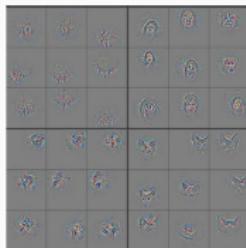
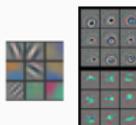
## **Transfer Learning**

---

4.1 Representational Learning

4.2 Transfer Learning

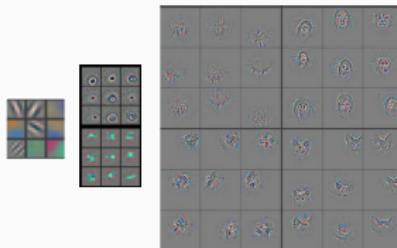
## Feature Extraction



“person”

- Assume we have a CNN trained on a *large database*, such as ImageNet ( $\approx 14$  million images),

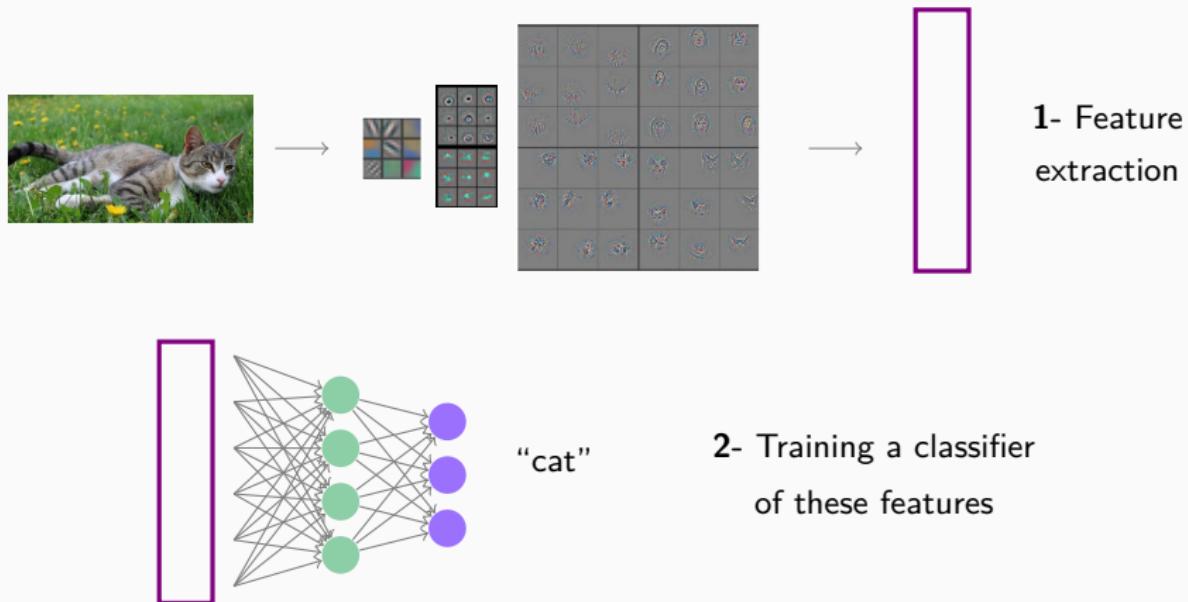
## Feature Extraction



- Assume we have a CNN trained on a *large database*, such as ImageNet ( $\approx 14$  million images),
- The convolutional base, which acts as a **feature extractor**, can be extracted and reused for another task.

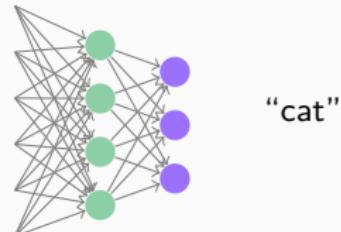
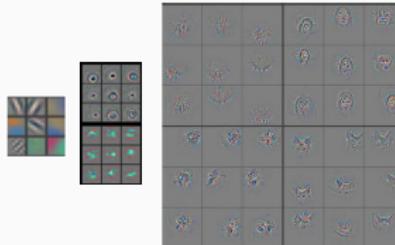
~~~ **Transfer learning**

“Static” Transfer Learning



We can use a **pre-trained** network to extract features from a new database, then train a simple classifier on these features.

Transfer Learning

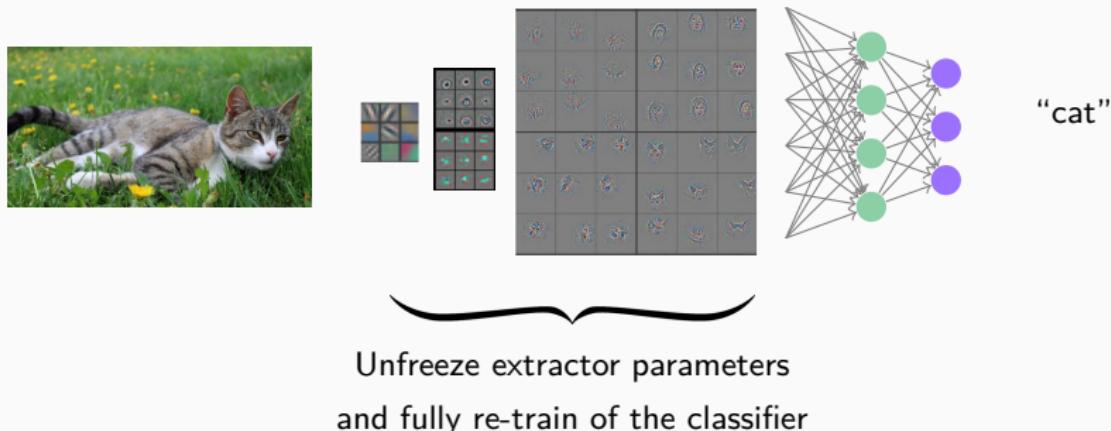


1- Freeze feature
extractor parameters

2- Training the last layers
of the classifier

If feature extraction is included in the classifier, but its parameters are locked, the learning transfer supports **data augmentation**.

Fine-Tuning



Fine tuning: Once the last layers of the classifier have been trained, the parameters of the convolutional base can be unlocked and the whole network re-trained, to “specify” it for the **new task**.

Attention! *The learning rate must be **very small** to avoid destroying the general filters obtained during pre-training.*