# Convolutional Neural Networks

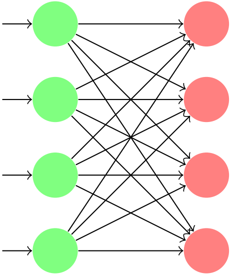
## Machine Learning

A. Carlier

2024

# Image classification on ImageNet

Images of shape $224 \times 224 \times 3$, 1000 classes :



$224 \times 224 \times 3$ pixels

$224 \times 224 \times 3$ neurons

1000 neurons

A single-layer perceptron would require $224 \times 224 \times 3 \times 1000 + 1000$ parameters, i.e. $\approx$150 million parameters !

# Outline

# Convolution

$$\begin{bmatrix}1&1&0&1\\0&0&0&1\\1&1&1&0\\1&0&0&1\end{bmatrix} * \begin{bmatrix}-1&-2&-1\\0&0&0\\1&2&1\end{bmatrix} = \begin{bmatrix} \ & \ \\ \ & \ \end{bmatrix}$$

Image

Filter $3 \times 3$

$f = 3$

Response

# Convolution

$$\begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & \\ & \end{bmatrix}$$

Image

Filter $3 \times 3$
$f = 3$

Response

$$1 * -1 + 1 * -2 + 0 * -1 + 0 * 0 + 0 * 0 + 0 * 0 + 1 * 1 + 1 * 2 + 1 * 1 = 1$$

# Convolution

$$
\begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}
*
\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}
=
\begin{bmatrix} 1 & 1 \\ & \end{bmatrix}
$$

Image

Filtee $3 \times 3$
$f = 3$

Response

# Convolution

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

Image

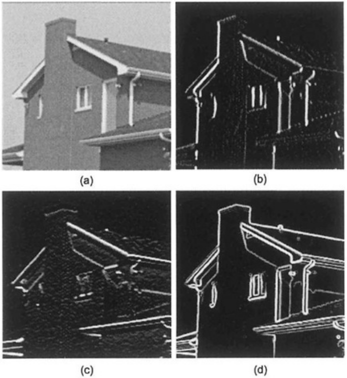
Filter $3 \times 3$
$f = 3$

Response

# Convolution

$$\underset{\text{Image}}{\begin{bmatrix}1&1&0&1\\0&0&0&1\\1&1&1&0\\1&0&0&1\end{bmatrix}} * \underset{\substack{\text{Filter } 3\times 3\\ f=3}}{\begin{bmatrix}-1&-2&-1\\0&0&0\\1&2&1\end{bmatrix}} = \underset{\text{Response}}{\begin{bmatrix}1&1\\1&0\end{bmatrix}}$$

# Convolution in Signal Processing




| | | |
|---|---|---|
| -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 response of the filter, i.e. a pixel located on the contour of an object, with a strong local gradient.

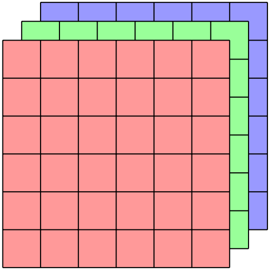
# Tensor dimension

Given :

- I a grayscale image (a single color channel) of shape $w \times h$,
- a filter K of dimension $f$,

Then the response $\mathrm{I} \circledast \mathrm{K}$ of image I to filter K is of shape $(w - f + 1) \times (h - f + 1)$

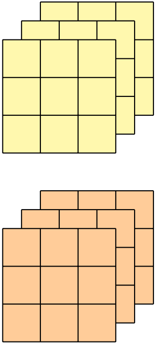
# Convolution (2D) de volumes



Image

$6 \times 6 \times \mathbf{3}$

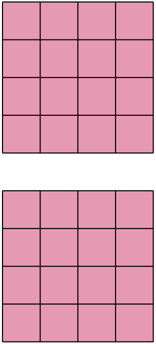
$*$



Filters

$3 \times 3 \times \mathbf{3} \times 2$

$=$



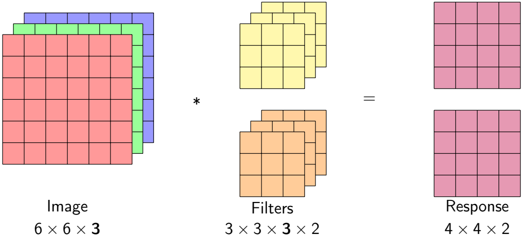
Response

$4 \times 4 \times 2$

$\rightarrow$ in Keras : Conv2D(*filters*,*kernel_size*)

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

# Number of parameters in convolutional layers

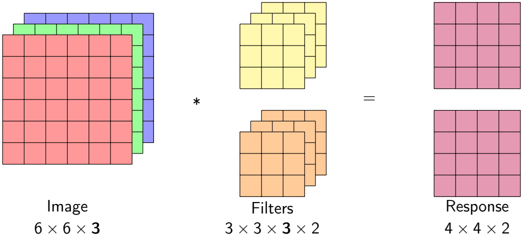



There are 2 types of parameters in a convolutional layer :

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

So in the example above, there are 3x3x3x2 + 2 = 56 parameters.

# Number of operations in convolutional layers




It is interesting to count the number of operations of a neural network to characterize its complexity, and the resources required for its execution. We are mainly interested in additions and multiplications (*FLOPs*).
Here, each element of the answer requires :

- ($f \times f \times$ #channels) multiplications
- ($f \times f \times$ #channels - 1) additions between the multiplied values
- 1 extra addition for the bias

Thus in the above example, there are (4x4x2) x (3x3x3x2) = 1728 operations.

## *Padding*

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

Image Filter Response

# *Padding*

| | | | | | |
|---|---|---|---|---|---|
| 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 |

Image
$p = 1$

$*$

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

Filter

$=$

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

Response

Adding zeros (*zero-padding*) to the image border allows to obtain a response of the same dimension as the input image (*same* parameter in Keras), which makes neural network architectures simpler.

# Tensor dimension

Given :

- I a grayscale image (a single color channel) of shape $w \times h$,
- a filter K of dimension $f$, and *padding* $p$,

Then the response $\mathrm{I} \circledast \mathrm{K}$ of image I to filter K is of shape
$(w + 2p - f + 1) \times (h + 2p - f + 1)$

# *Stride*

$$\begin{bmatrix} 3 & 1 & 3 & 5 & 3 & 3 \\ 2 & 2 & 8 & 8 & 3 & 9 \\ 3 & 4 & 7 & 7 & 2 & 7 \\ 5 & 3 & 6 & 8 & 4 & 7 \\ 3 & 8 & 8 & 5 & 7 & 4 \\ 7 & 9 & 6 & 4 & 6 & 9 \end{bmatrix} * \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 10 & 13 & 7 & 4 \\ 3 & -3 & -1 & 0 \\ 9 & 4 & 2 & 5 \\ 14 & 2 & -6 & 2 \end{bmatrix}$$

Image
$p = 0$

Filter
$f = 3$

$stride = 1$

Response

# *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
$p = 0$

$*$

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

Filter
$f = 3$

$stride = 2$

$=$

| | |
|---|---|
| 10 | 7 |
| 9 | 2 |

Response

Allows to **reduce the dimension** of tensors, limiting the loss of information due to the fact that the same coefficient influences several elements of the response to the convolution filter.

# Tensor dimension

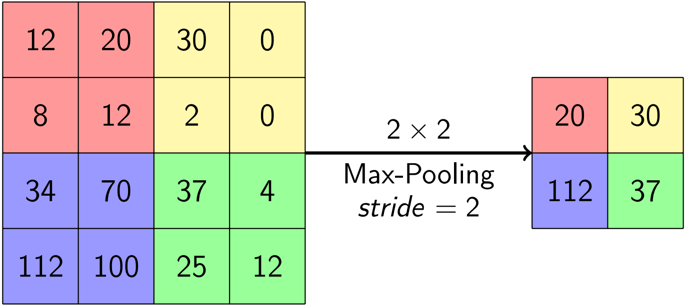
Given :

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

Then the response $\mathrm{I} \circledast \mathrm{K}$ of image I to filter K is of shape

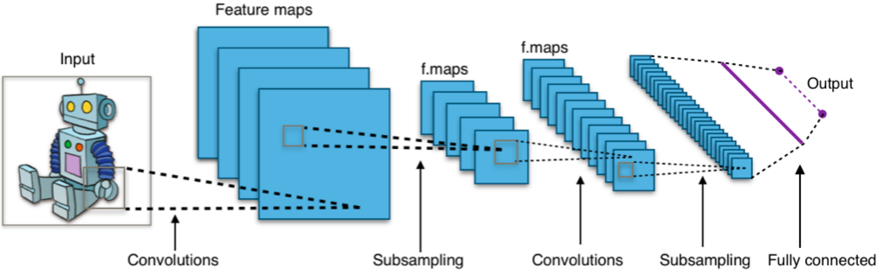
$$\lfloor \frac{w + 2p - f}{s} + 1 \rfloor \times \lfloor \frac{h + 2p - f}{s} + 1 \rfloor \tag{1}$$

# *Pooling* layers




- Allows to **reduce** tensor **dimension**
- **Preserve the high responses** of the convolution filters.
- **No parameters** to learn !

# Standard architecture of a convolutional neural network




There are 3 types of layers in a typical convolutional neural network :

- **Convolutional** layers, combined with **pooling** layers, in the first layers of the network.
- Fully connected (dense) layers in the last layers of the network.

# Parameter sharing

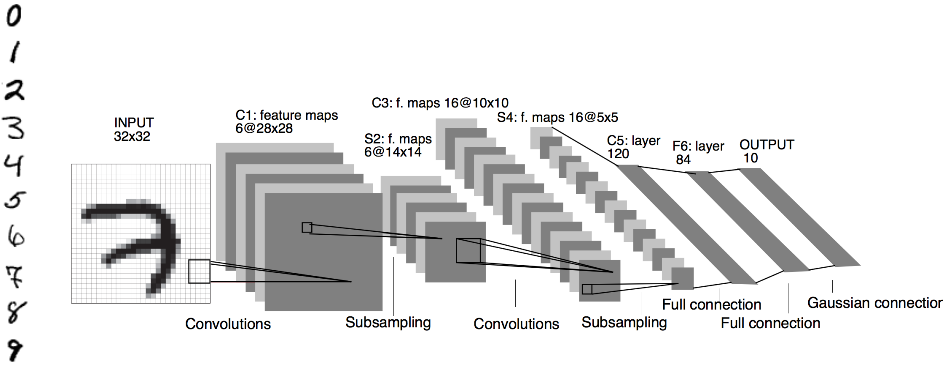
A convolutional layer is equivalent to a fully connected layer in which some synaptic weights are shared, and the majority of which are 0.

A single example from the training set allows these weights (the convolution coefficients) to be updated multiple times.

This results in a much smaller number of parameters for a convolutional network than for a fully connected network.

# Outline

# LeNet-5 : a pioneer (1998)

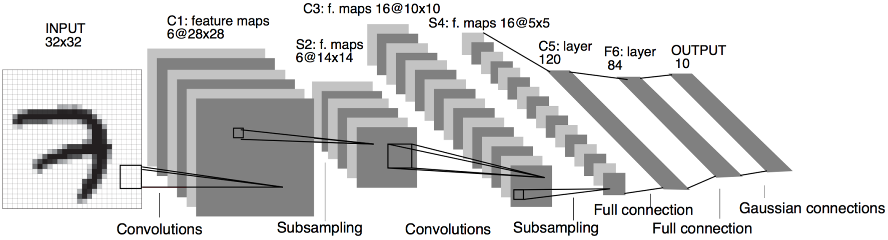



- $\simeq$ 60k parameters
- 2 to 3 days of training for 20 *epochs* on MNIST (1998 !).
- A majority of sigmoid activation functions

Visualization : `https://adamharley.com/nn_vis/cnn/2d.html`

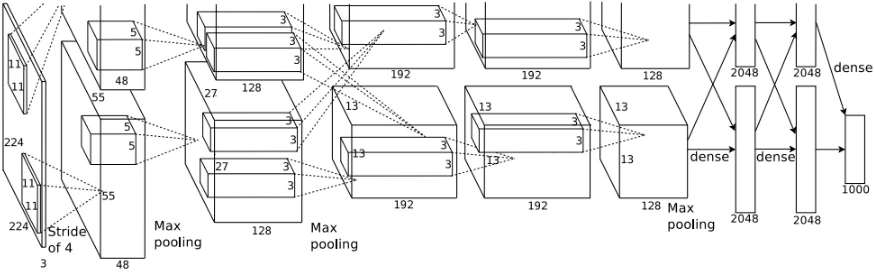
[LeCun et al.] Gradient-based learning applied to document recognition.

# Question




Number of parameters ? Number of operations ?

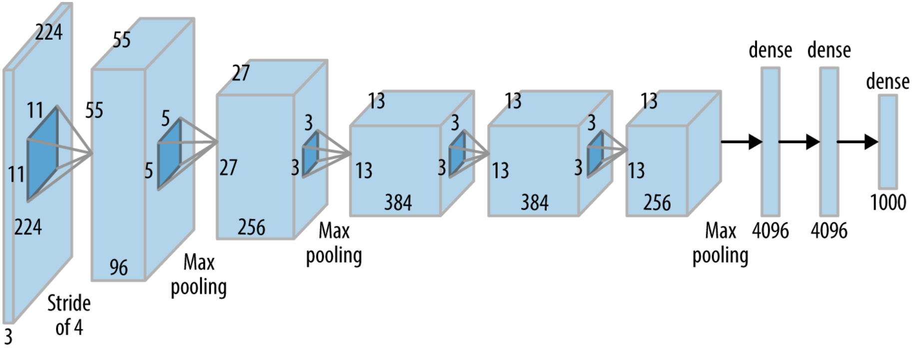
# AlexNet : a game changer (2012)




- $\simeq$ 60M parameters, 8 layers
- introduces the use of `reLU` function as a standard for neural network training.

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

# AlexNet : a game changer (2012)

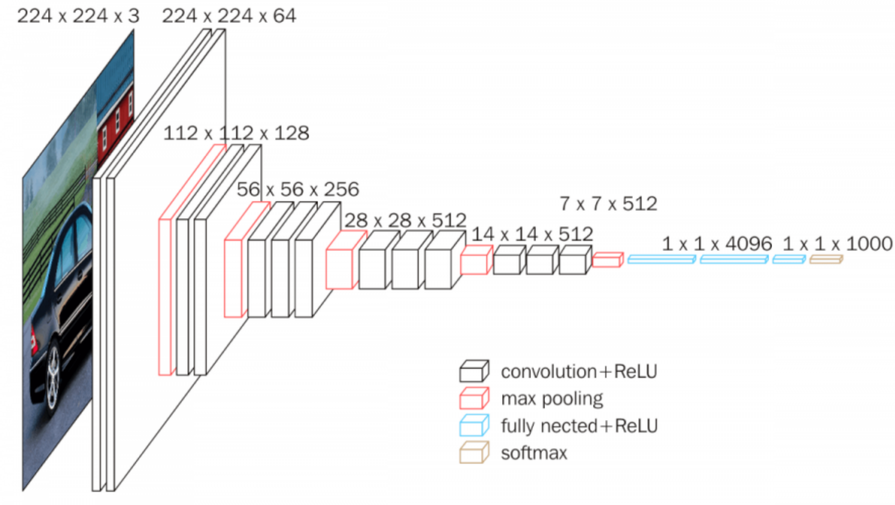



Observations :

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

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

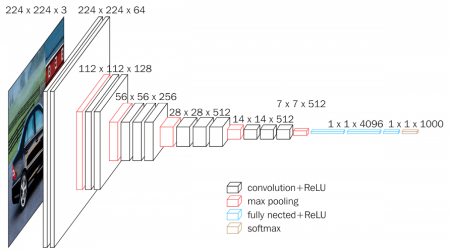
# VGG-16 : a new standard (2014)




$\simeq$ 138M parameters, 16 layers.

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

# VGG-16 : a new standard (2014)

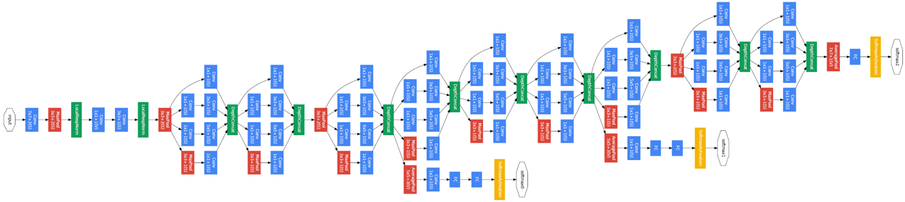



Objective : to study the impact of depth on network performance.
$\rightarrow$ For this purpose, the authors have made the network architecture very regular :

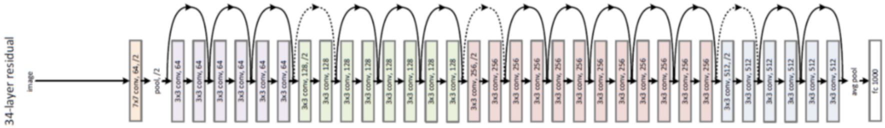
- Systematic use of $3 \times 3$ convolutions
- The main characteristics of `AlexNet` are taken up, but regularized :
  - Progressive decrease of the image size ($224 \rightarrow 112 \rightarrow 56$ ...)
  - Progressive increase in the number of filters ($64 \rightarrow 128 \rightarrow 256$...)

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

# More advanced architectures

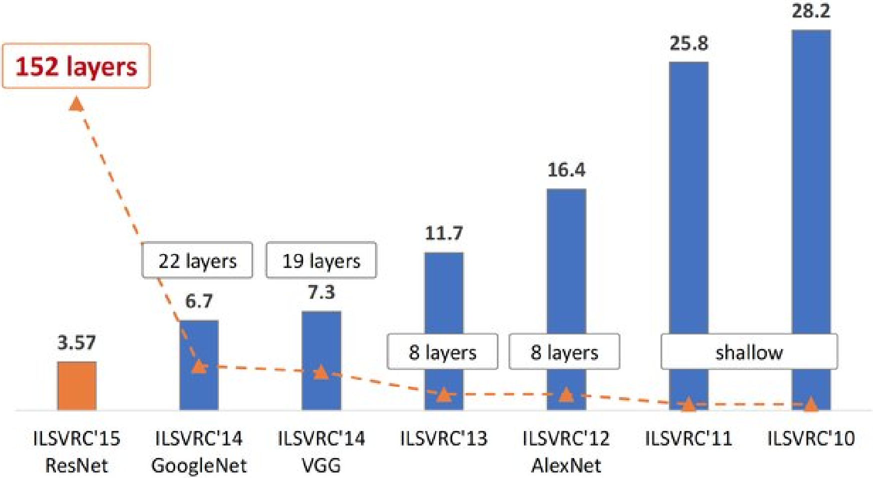


[Szegedy et al.] Going deeper with convolutions.

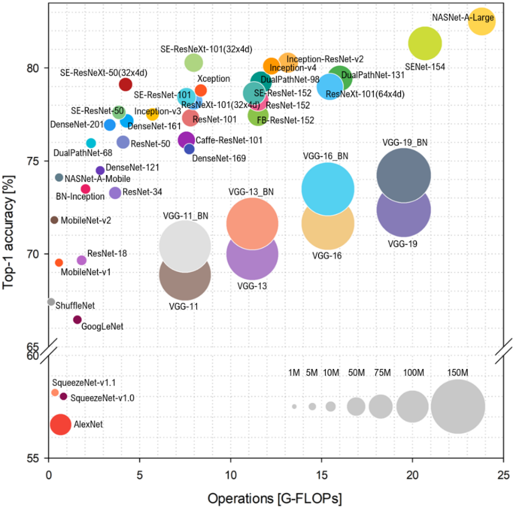



[He et al.] Deep Residual Learning for Image Recognition

# 2015 : end of ImageNet challenge on image classification



# After 2015




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

# Outline

# Representation learning

$X$

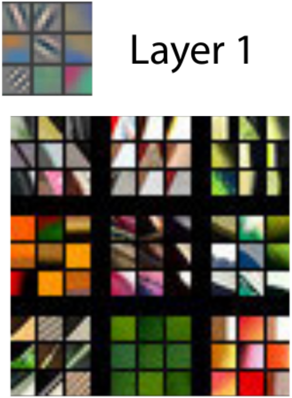
$Y$

$\xrightarrow{h}$ "person"

$\xrightarrow{f} \boxed{f(X)} \xrightarrow{h'}$

features

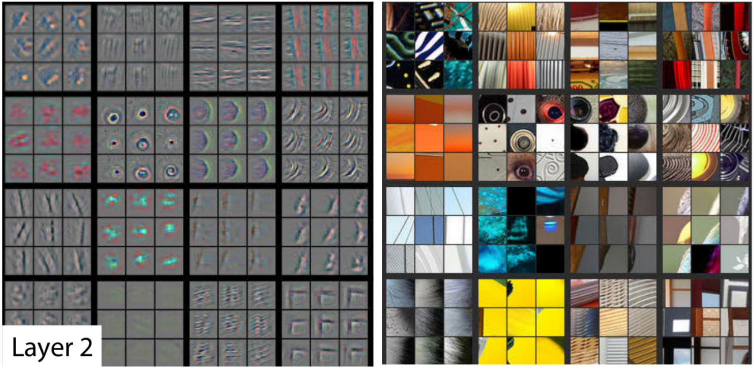
# What do CNN learn ?




Example of filters learned on the first layer of a convolutional neural network (similar to AlexNet), and for each filter, enumeration of the 9 image patches causing the highest activation of these filters.

[Zeiler et al.] Visualizing and understanding convolutional networks.

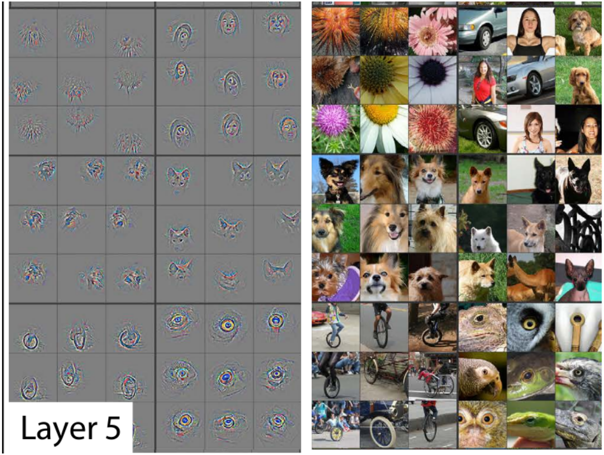
# What do CNN learn ?




Same visualization as before, but for the second layer filters. Note that the filters on the left side are not present as is in the network : this visual representation is reconstructed.

[Zeiler et al.] Visualizing and understanding convolutional networks.

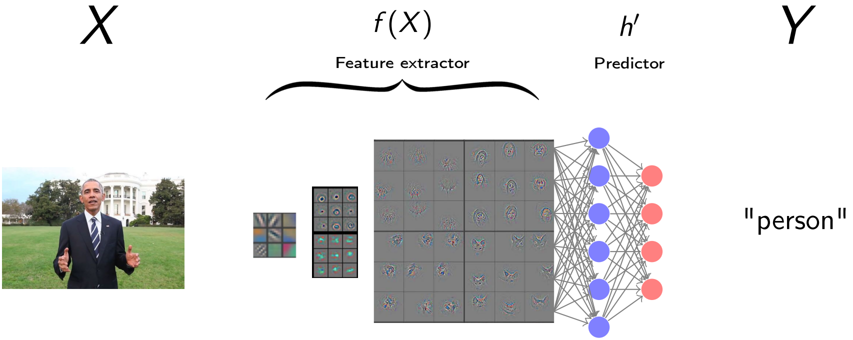
# What do CNN learn ?




The detected patterns are of a higher semantic level as we progress in the network layers.

[Zeiler et al.] Visualizing and understanding convolutional networks.

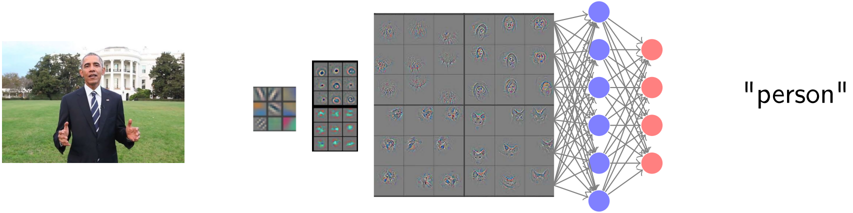
# CNN interpretation




We can see a CNN as a Representation learning algorithm : the convolutional part is a feature extractor $f(X)$, and the extra dense layers are the actual predictor $h'$.
Deep neural networks learn a feature extractor from the data. That is what makes them efficient, **in cases where a large annotated dataset is available**.

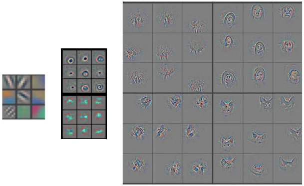
# Outline

# Transfer Learning



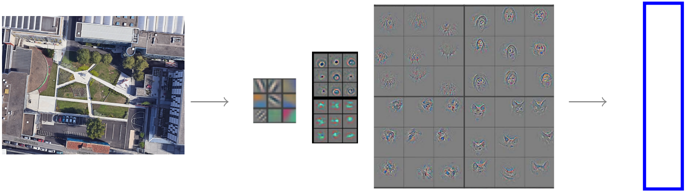

Let's assume that we have a CNN trained on a large database, such as ImageNet ($\approx$ 14 million images).

# Transfer Learning

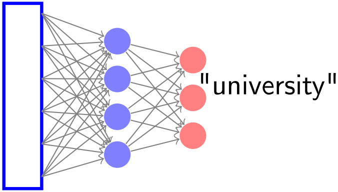


We can extract the convolutional base which acts as a feature extractor, and reuse it for another task. This is what is called **Transfer Learning**.

# Static Transfer Learning



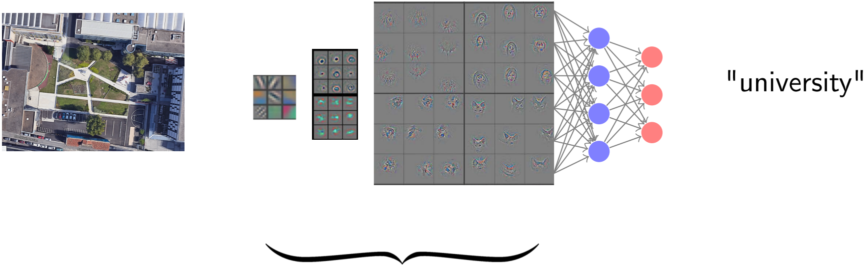
**1**- Feature Extraction




**2**- Feature classification training

One can use a pre-trained network to extract features from a new database, and then train a simple classifier of those features.

# Transfer Learning

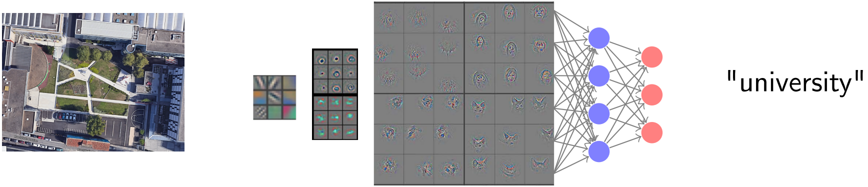



**1**- Feature extractor parameters are frozen

**2**- Dense layers from the classifier are trained

If feature extraction is included in the classifier, but its parameters are frozen, Transfer Learning supports data augmentation.

# Fine-Tuning




Feature extractor parameters are unfrozen
and the classifier is re-trained as a whole

Once the last layers of the classifier have been trained, we can then unfreeze the parameters of the convolutional base and re-train the whole network, in order to "specify" it to the new task : this is called **fine-tuning**. **Caution : the learning rate must be very small in order not to risk destroying the general filters which were obtained during pre-training**.