

# **Apprentissage Automatique**

**« Deep Learning »**

**Stéphane Herbin**

`stephane.herbin@onera.fr`

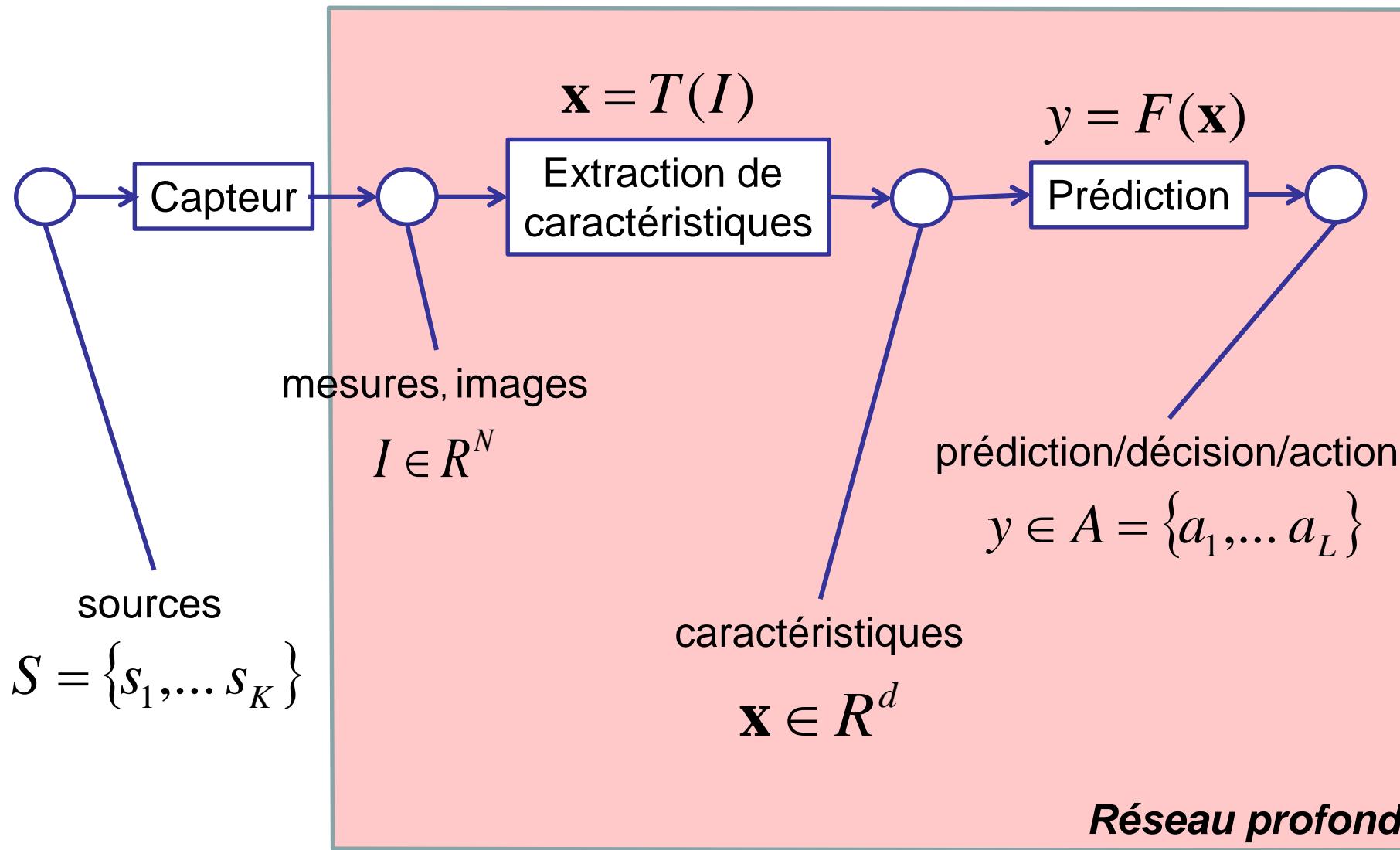
# Aujourd’hui

- Réseaux de neurones profonds (« Deep Neural Networks »)
  - Ce sont des réseaux de neurones multi couches
  - Contiennent des représentations intermédiaires utiles
  - Une architecture pour l'image: réseaux convolutifs
  - Una autre architecture: les « transformers »
  - Le principe « end to end » et la dérivation automatique
  - Compléments d'optimisation: ReLU et extinction du gradient, résidus, gestion du pas gradient
  - Des techniques pour que ça marche: augmentation de données, normalisation
  - Comprendre les architectures
  - D'autres architectures: transformers et réseaux « denses »
  - Pourquoi ça marche
- TD
  - Réseaux convolutifs pour les images

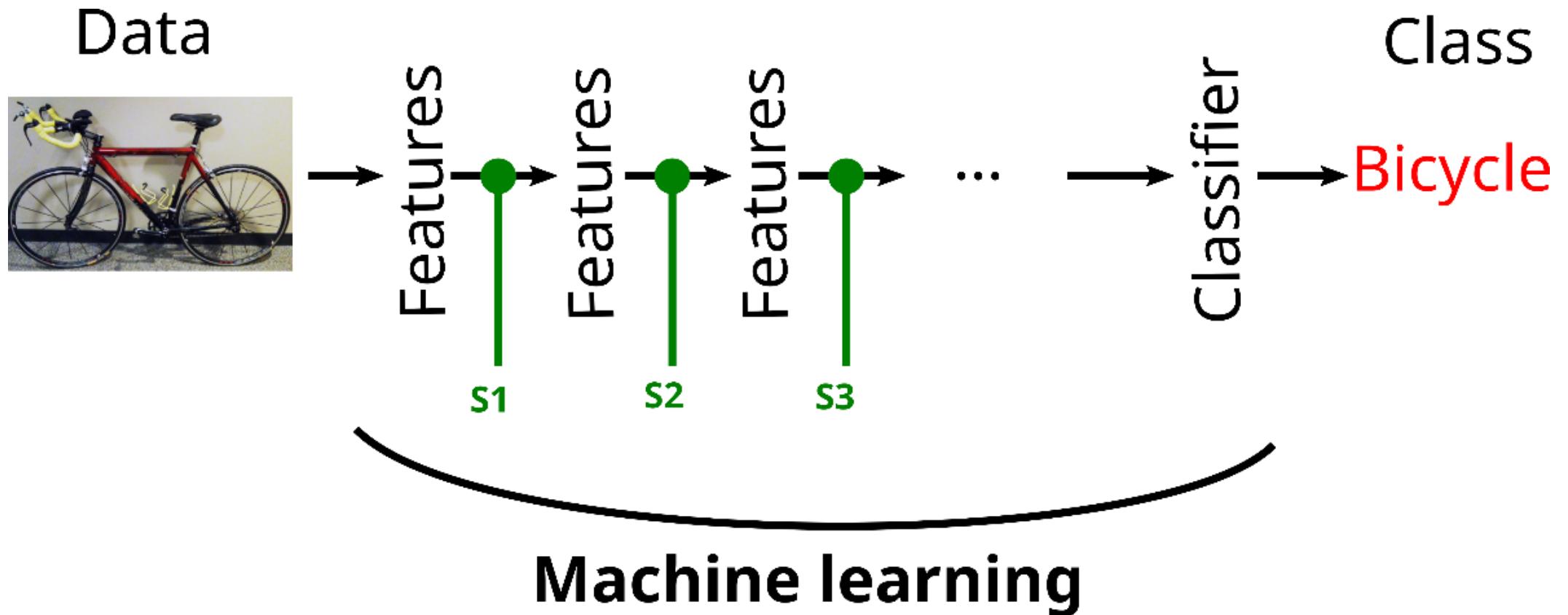
# Pourquoi profond?

- ~Cerveau
  - Zones corticales
  - Structure en couches
- Composition de fonctionnalités
  - Extraction de caractéristiques + classification
  - Multi-échelle
  - Représentation hiérarchique des données
- Approximation de fonction
  - Quelques résultats théoriques sur l'intérêt d'avoir un bon compromis entre nombre de couches (« depth ») et nombres de neurones (« width »)
- Les réseaux qui marchent sont profonds!

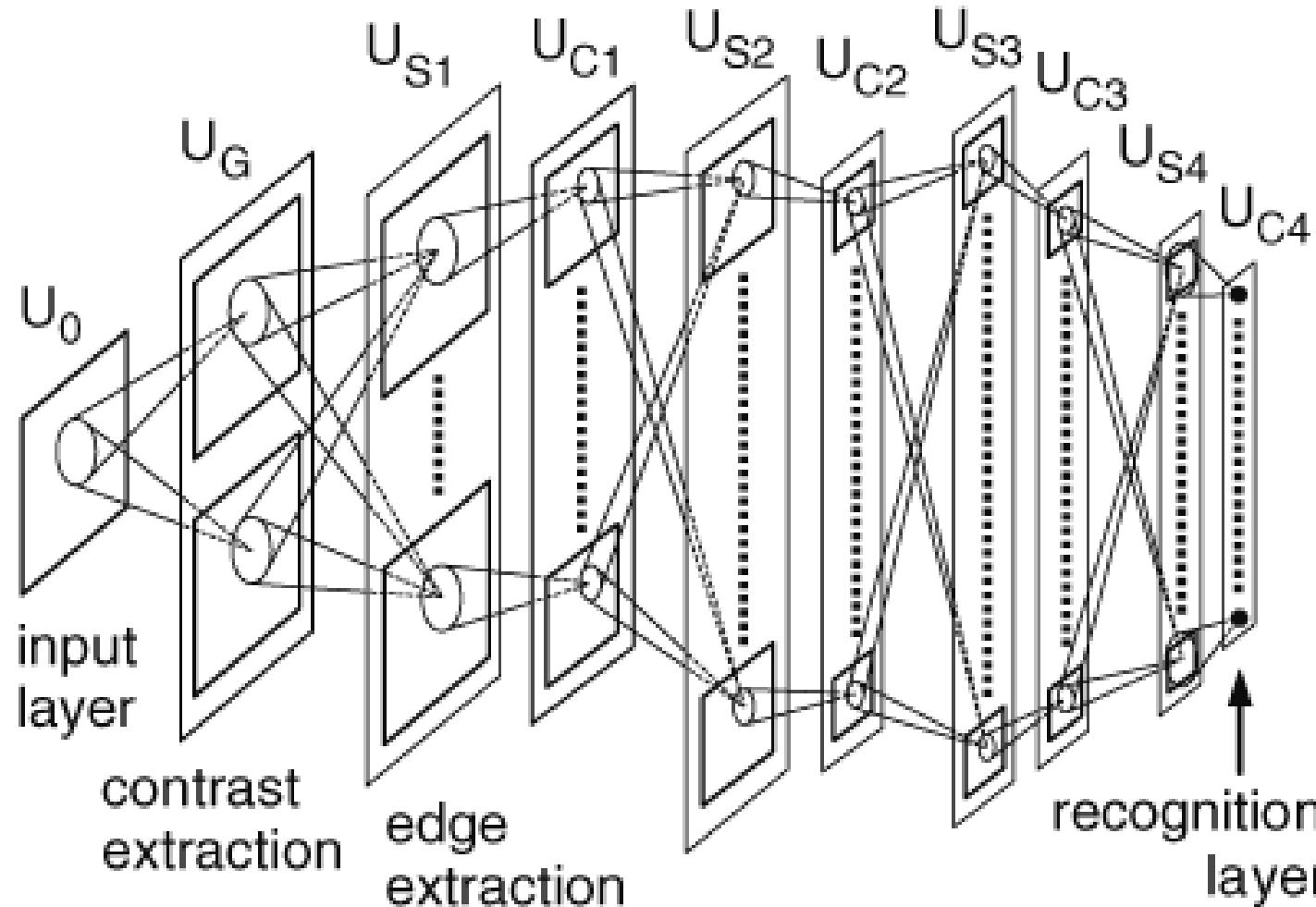
# Prédire & extraire les caractéristiques en même temps



Réseaux profonds = extraction d'information + prédiction

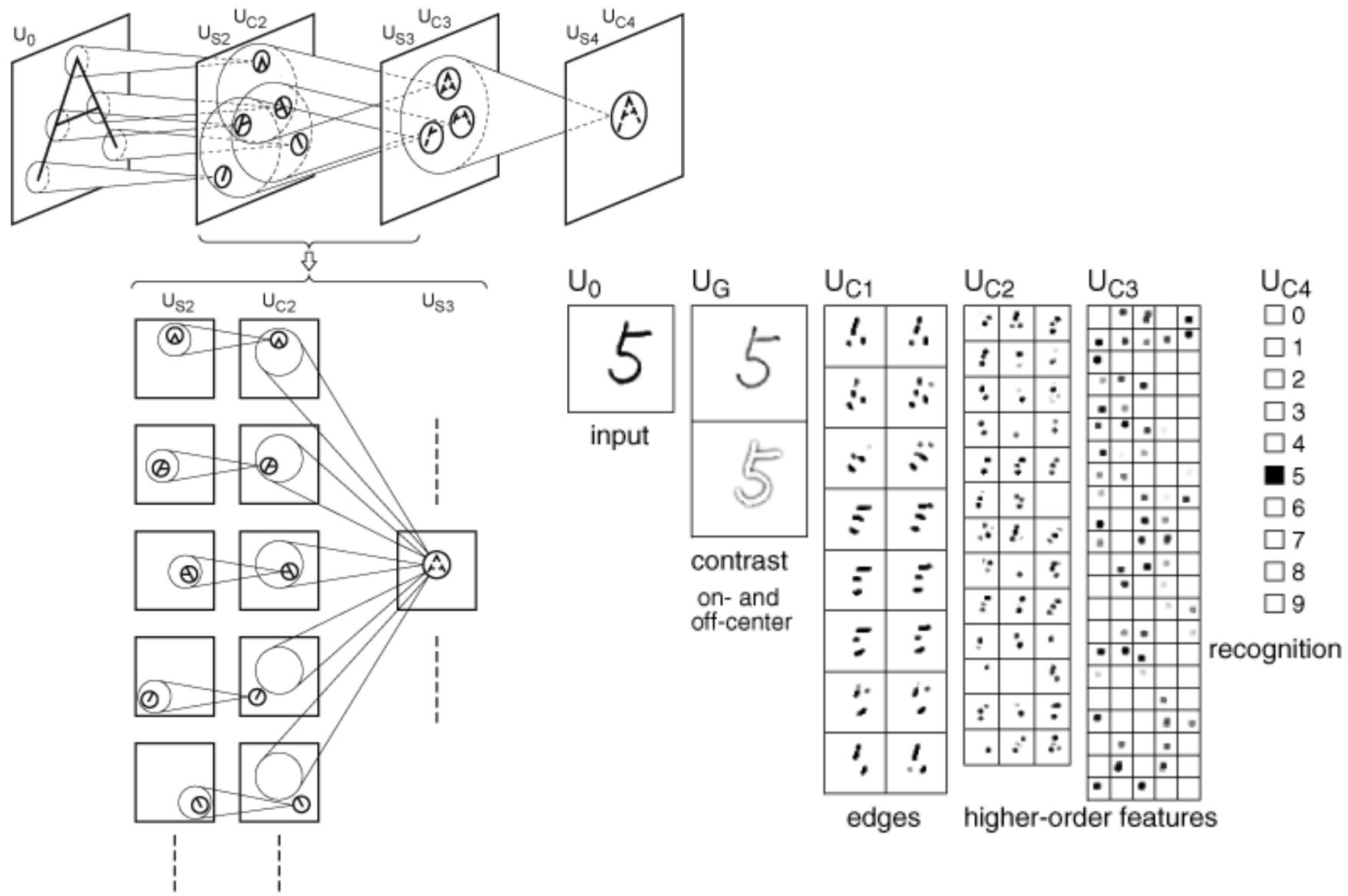


# Neocognitron (1980)

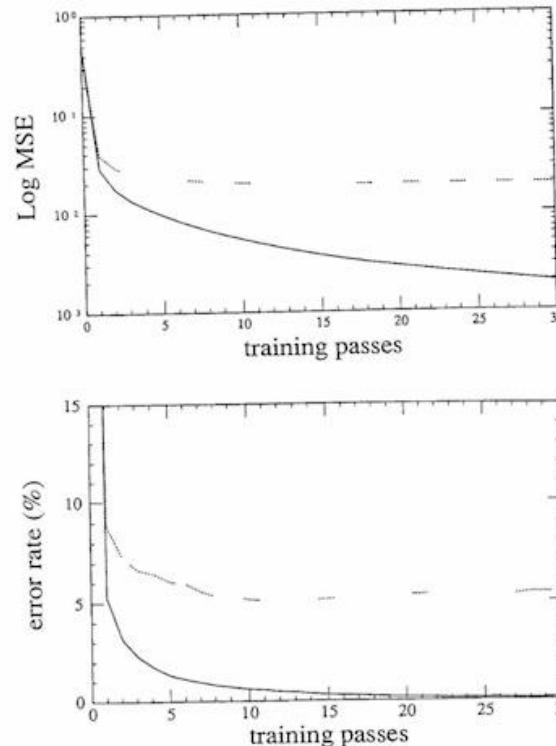
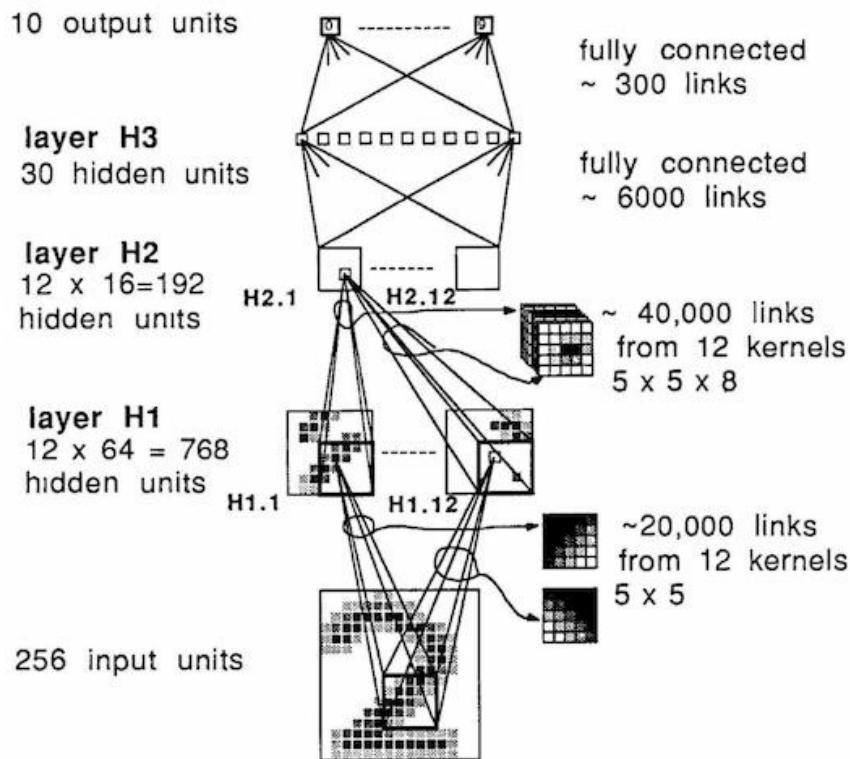


K. Fukushima: "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position", *Biological Cybernetics*, 36[4], pp. 193-202 (1980).

# Neocognitron (1980)



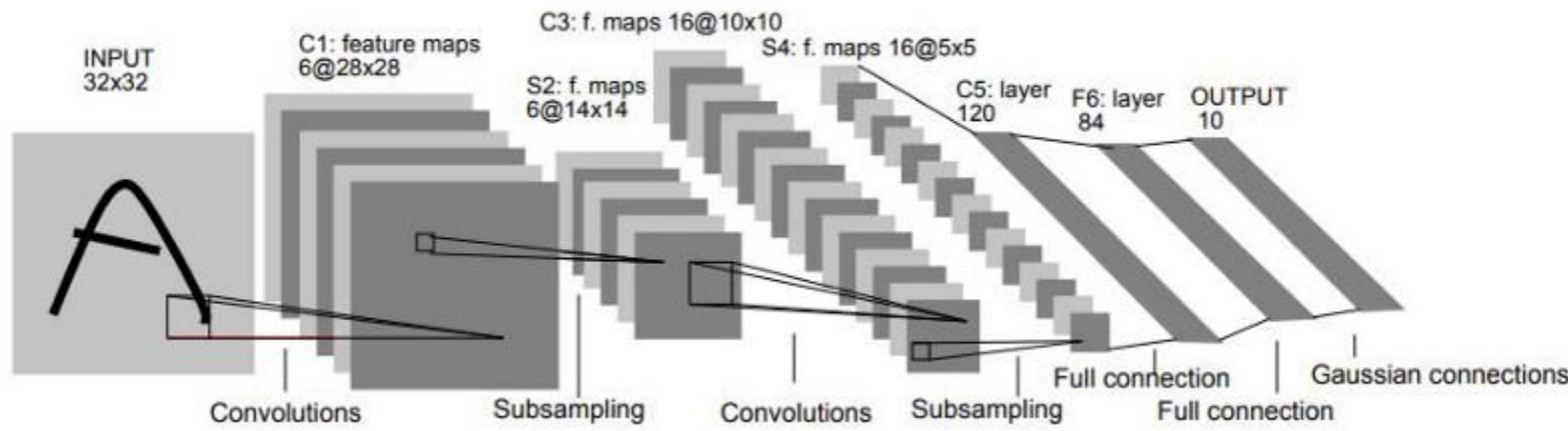
# LeNet (1989)



Introduit l'apprentissage par rétro-propagation dans les couches convolutives

LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W. & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541-551

# LeNet5 (1998)



Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	$32 \times 32$	-	-	-
1	Convolution	6	$28 \times 28$	$5 \times 5$	1	tanh
2	Average Pooling	6	$14 \times 14$	$2 \times 2$	2	tanh
3	Convolution	16	$10 \times 10$	$5 \times 5$	1	tanh
4	Average Pooling	16	$5 \times 5$	$2 \times 2$	2	tanh
5	Convolution	120	$1 \times 1$	$5 \times 5$	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

## Number of parameters

Convolutional layer C1:  $6 (5 * 5 + 1) = 156$  parameters

Pooling layer P1: No parameters

Convolutional layer C2:  $64 (5 * 5 + 1) = 1600$  parameters

pooling layer P2: No parameters

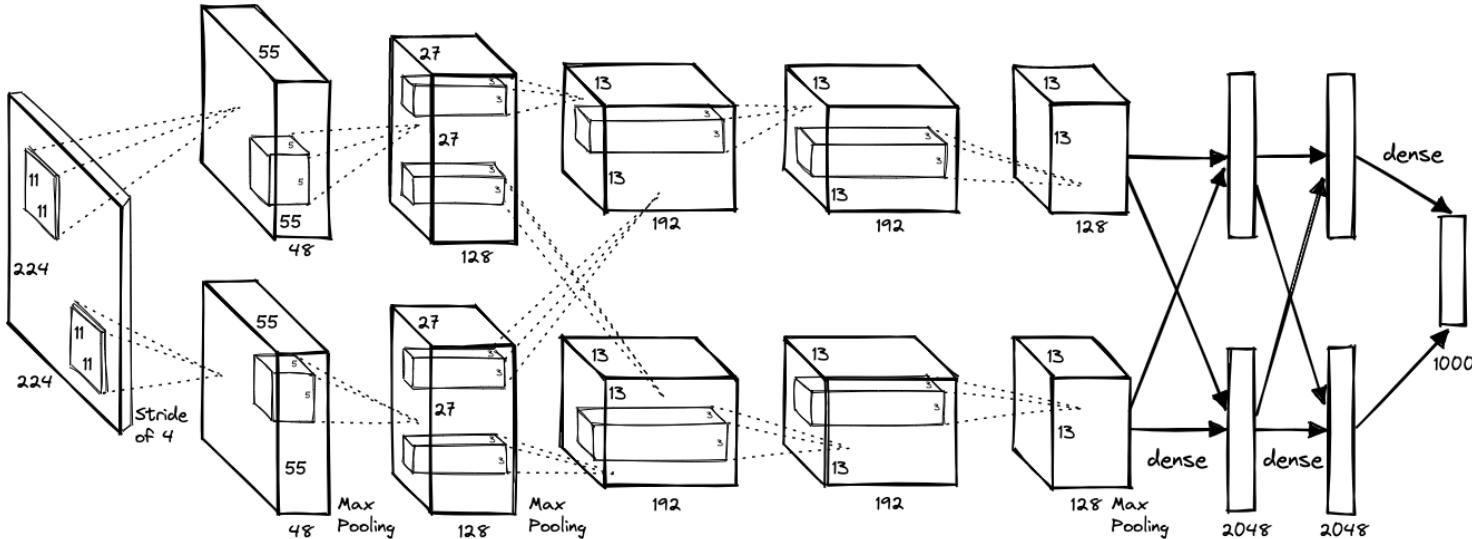
Fully connected layer F1:  $120 * 84 = 10,080$  parameters

Fully connected layer F2:  $84 * 10 = 840$  parameters

**Total = 60,206 parameters**

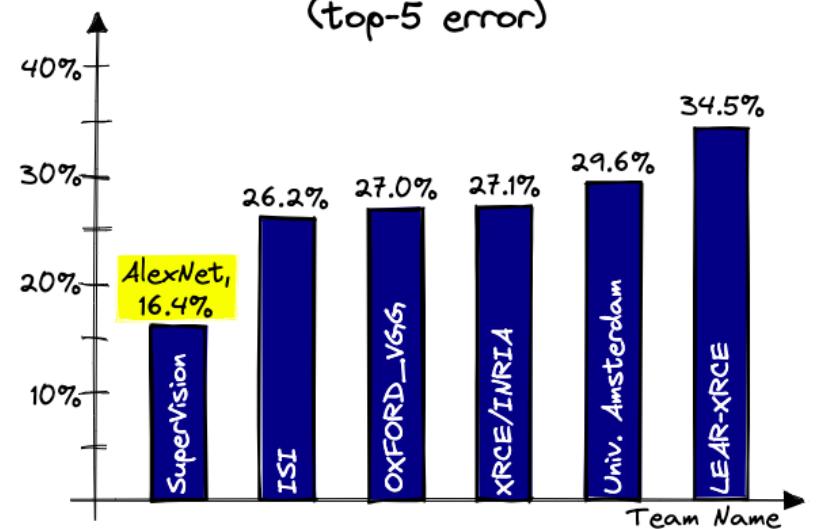
- Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. (1998). "Gradient-based learning applied to document recognition". Proceedings of the IEEE. 86 (11): 2278–2324

# AlexNet & Challenge ImageNet (2012)



AlexNet Network - Structural Details											
Input		Output			Layer	Stride	Pad	Kernel size	in	out	# of Param
227	227	3	55	55	96	conv1	4	0	11	11	34944
55	55	96	27	27	96	maxpool1	2	0	3	3	96
27	27	96	27	27	256	conv2	1	2	5	5	614656
27	27	256	13	13	256	maxpool2	2	0	3	3	256
13	13	256	13	13	384	conv3	1	1	3	3	256
13	13	384	13	13	384	conv4	1	1	3	3	384
13	13	384	13	13	256	conv5	1	1	3	3	384
13	13	256	6	6	256	maxpool5	2	0	3	3	256
					fc6			1	1	9216	4096
					fc7			1	1	4096	4096
					fc8			1	1	4096	1000
<b>Total</b>										<b>62,378,344</b>	

2012 ImageNet Challenge  
(top-5 error)



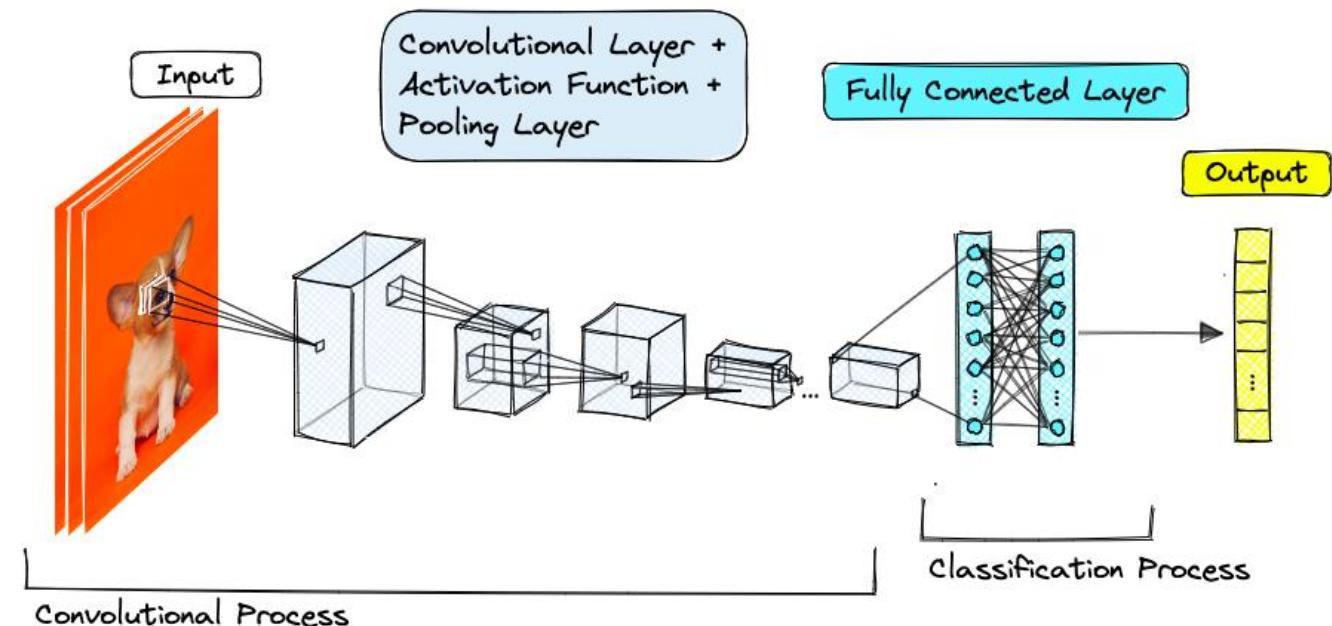
# Difference AlexNet vs. LeNet

## Architecture

- Larger input images (224x224)
- Larger number of classes (1000)
- More layers (and parameters)
- Overlapping Max pooling
- ReLU
- Normalization layers

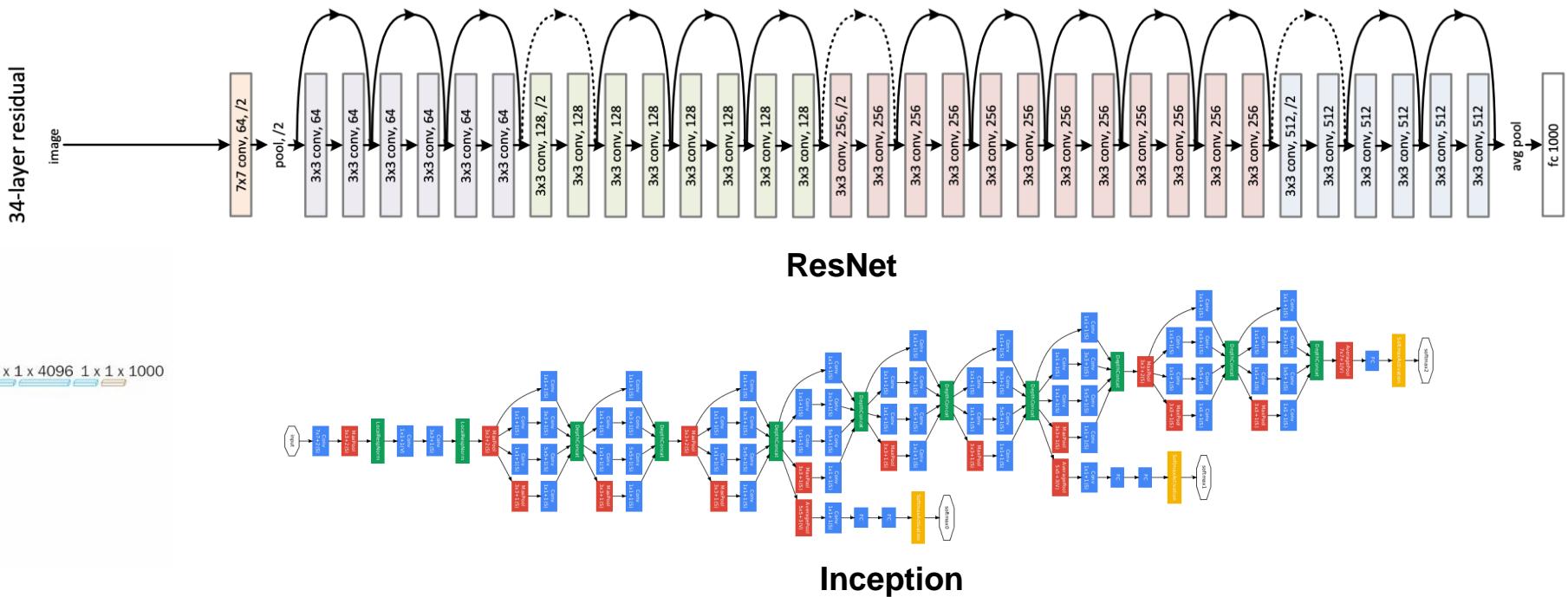
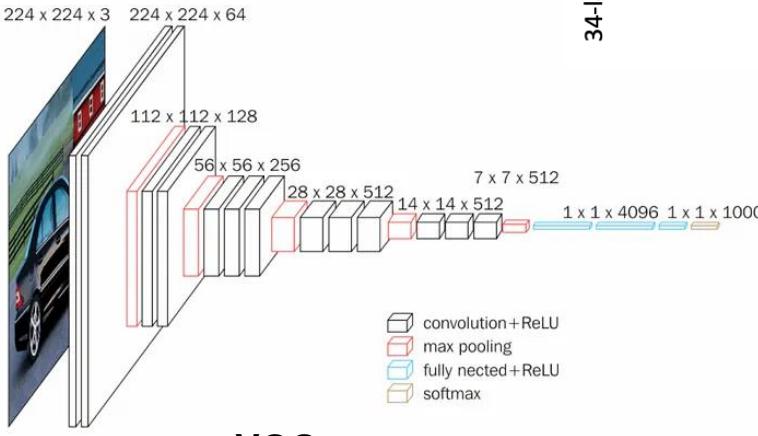
## Optimisation

- Drop Out
- SGD momentum
- Data augmentation
- Multi-GPU
- Batch of size 128



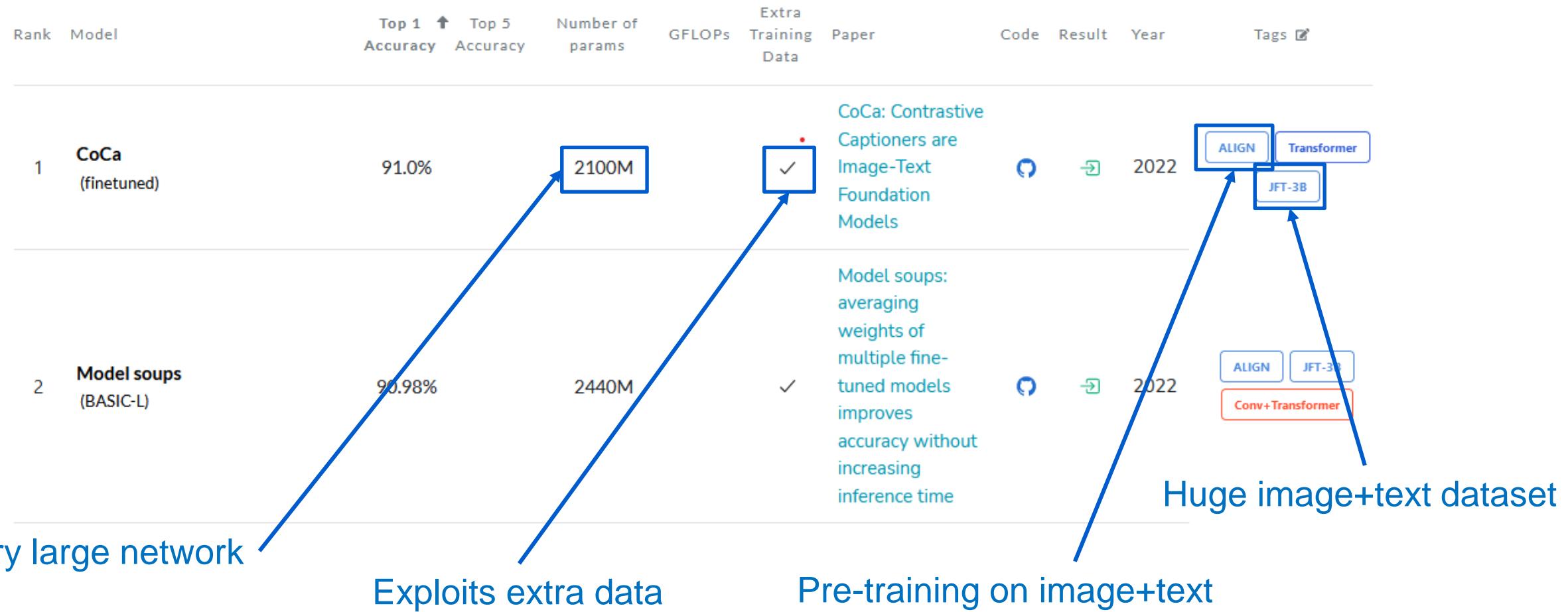
**Same architecture principle !**

# Autres architectures classiques



Comparison					
Network	Year	Salient Feature	top5 accuracy	Parameters	FLOP
AlexNet	2012	Deeper	84.70%	62M	1.5B
VGGNet	2014	Fixed-size kernels	92.30%	138M	19.6B
Inception	2014	Wider - Parallel kernels	93.30%	6.4M	2B
ResNet-152	2015	Shortcut connections	95.51%	60.3M	11B

# IMAGENET classification: tendances récentes



<https://paperswithcode.com/sota/image-classification-on-imagenet>

# Approche Deep Learning

1. Constituer des bases de données
2. Préparer les données: Analyser, visualiser, constituer les ensembles train/test
3. Concevoir le modèle
4. Définir un critère et Optimiser
5. Evaluer

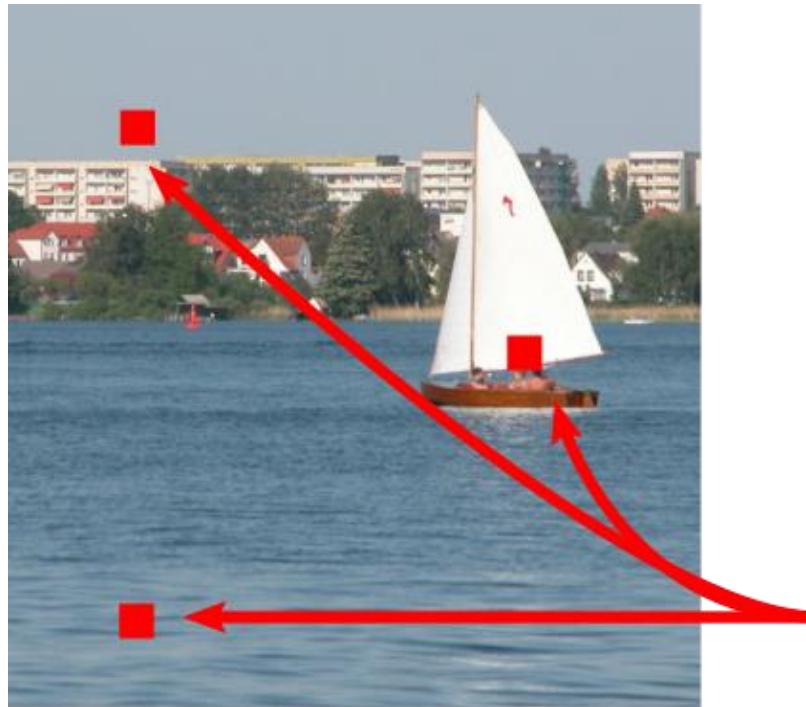
C'est la démarche classique d'apprentissage mais...

2. On n'a plus d'étape d'extraction de caractéristiques (c'est intégré dans le réseau)
3. Ca se réduit à trouver la bonne architecture neuronale (ce n'est pas toujours facile!)
4. Le critère est important mais l'optimisation encore plus!
5. L'évaluation permet de contrôler l'optimisation, pas seulement le résultat final.

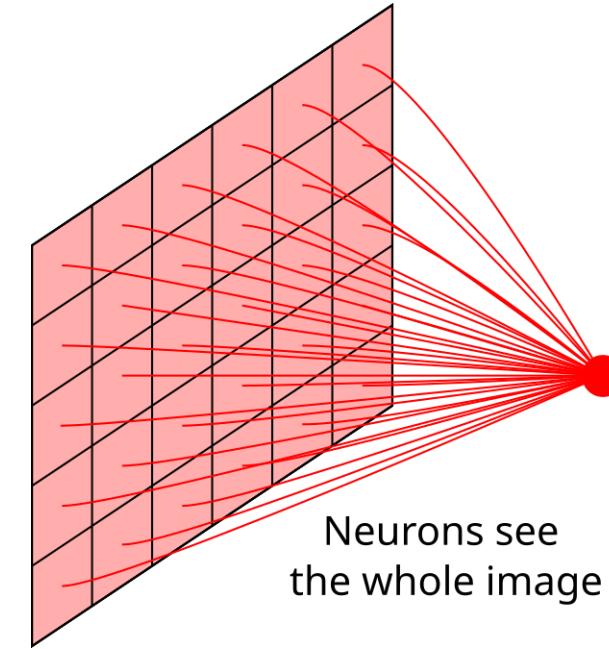
# Les architectures profondes

- Elles consistent à enchaîner des couches (~MLP)
- Mais plus grande variété de types de couches
- Démarche « End to end »: trouver la bonne architecture « complète » + fonction de coût qui modélise le problème puis optimiser par descente de gradient
- Un exemple important en traitement de données: les couches convolutives
  - Poids partagés
  - Opérations linéaires + activation non linéaire → descente de gradient possible
  - Plusieurs convolutions en parallèle: banques de filtres
  - Mélange séquentiel de couches: représentations de plus en plus complexes

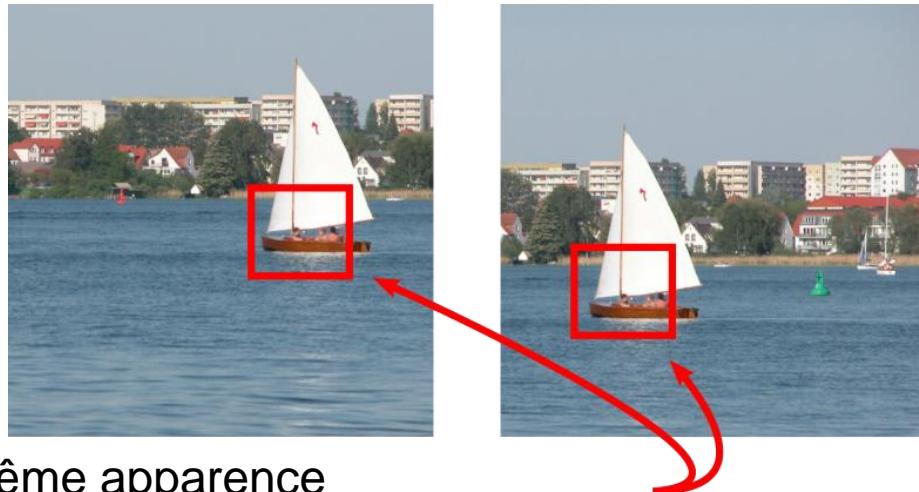
# Image et réseau entièrement connecté



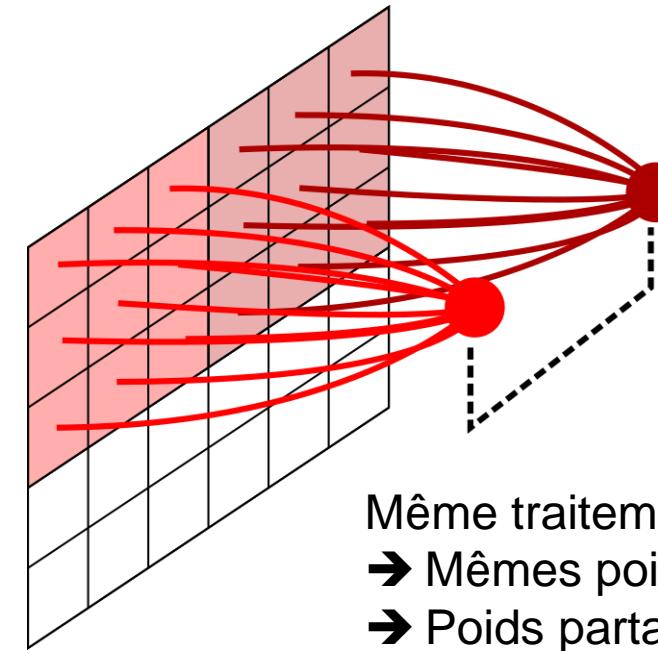
Un neurone  
pour traiter  
toute l'image



# Convolution = ANN « local » pour des images



Même apparence  
→ Même traitement quelle que soit la position du motif dans l'image



Même traitement  
→ Mêmes poids  
→ Poids partagés

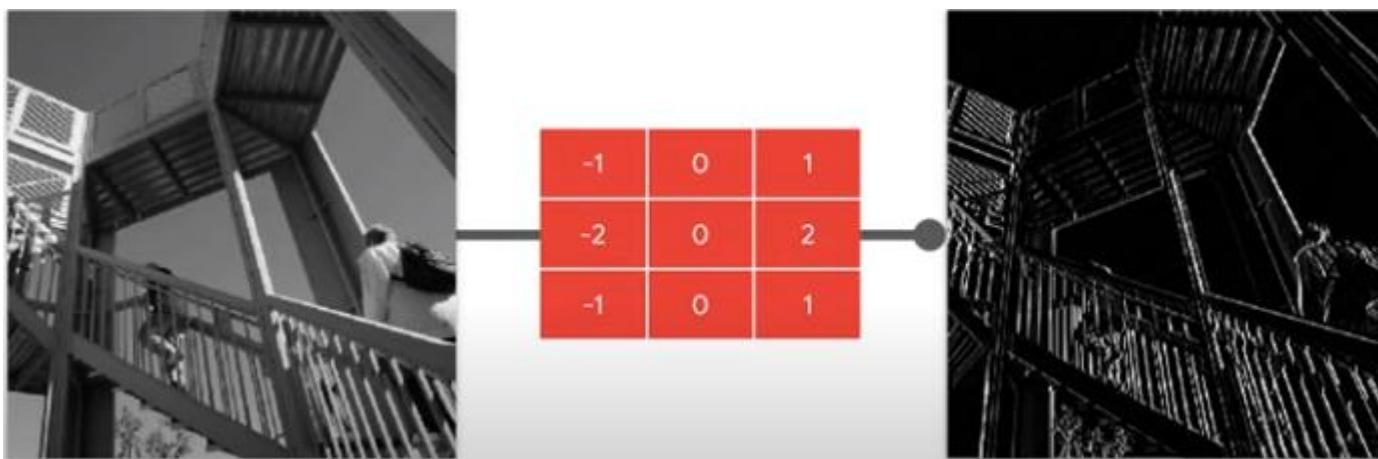
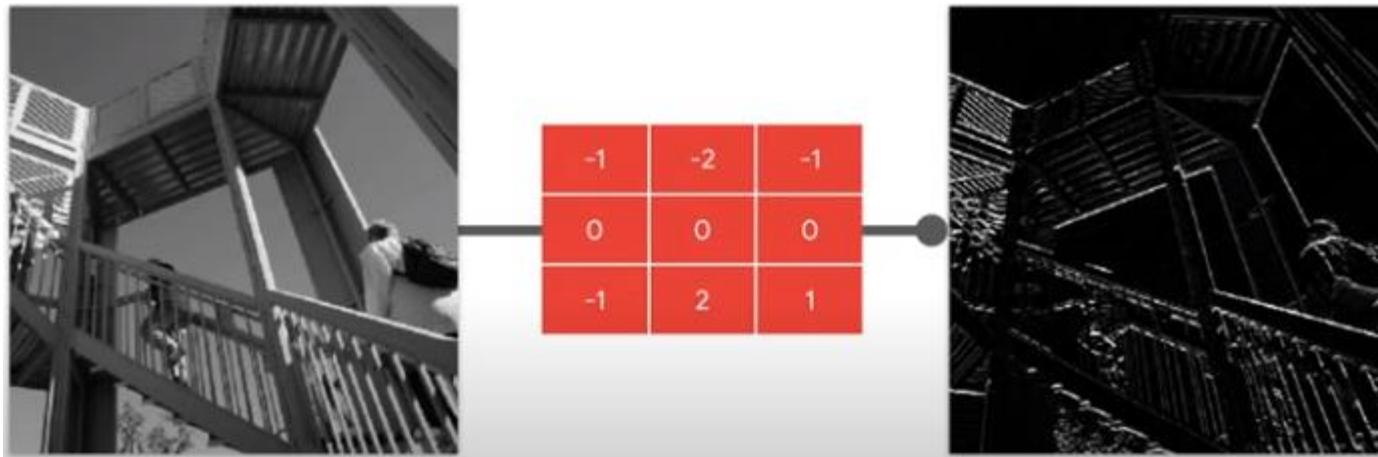
# Convolution 2D

Image d'entrée \* Masque de convolution = Neurones de sortie

$$\begin{array}{|c|c|c|c|c|} \hline 7 & 2 & 3 & 3 & 8 \\ \hline 4 & 5 & 3 & 8 & 4 \\ \hline 3 & 3 & 2 & 8 & 4 \\ \hline 2 & 8 & 7 & 2 & 7 \\ \hline 5 & 4 & 4 & 5 & 4 \\ \hline \end{array} \quad * \quad \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} \quad = \quad \begin{array}{|c|c|c|} \hline 6 & & \\ \hline & & \\ \hline & & \\ \hline \end{array}$$

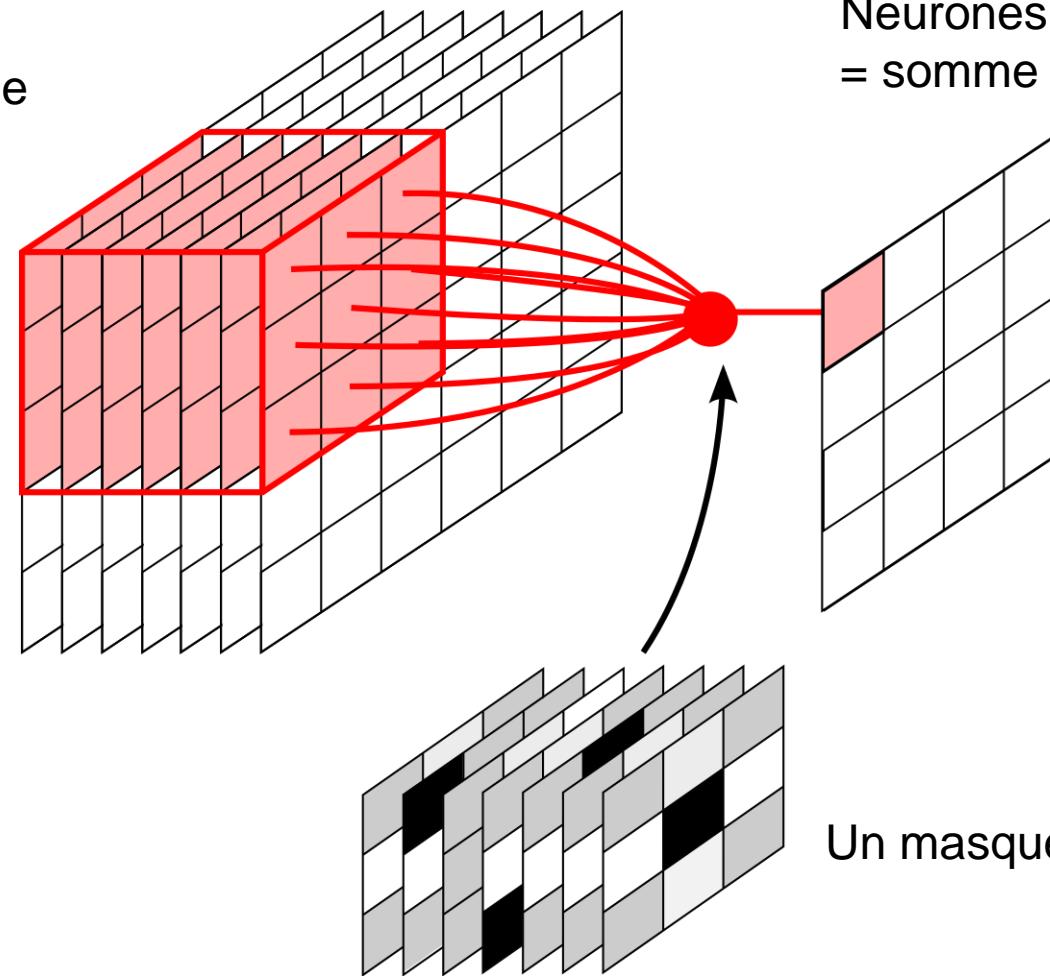
$$7 \times 1 + 4 \times 1 + 3 \times 1 + \\ 2 \times 0 + 5 \times 0 + 3 \times 0 + \\ 3 \times -1 + 3 \times -1 + 2 \times -1 \\ = 6$$

# Interprétation des convolutions comme filtres



# Convolution 3D

Plusieurs canaux en entrée



Neurones de sortie  
= somme des contributions « locales »

Un masque de convolution par canal

# Convolution 3D

Exemple de tenseur d'entrée (RGB)

0	0	0	0	0	0	0	...
0	156	155	156	158	158	158	...
0	153	154	157	159	159	159	...
0	149	151	155	158	159	159	...
0	146	146	149	153	158	158	...
0	145	143	143	148	158	158	...
...	...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	0	...
0	167	166	167	169	169	169	...
0	164	165	168	170	170	170	...
0	160	162	166	169	170	170	...
0	156	156	159	163	168	168	...
0	155	153	153	158	168	168	...
0	154	152	152	157	167	167	...
...	...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	0	...
0	163	162	163	165	165	165	...
0	160	161	164	166	166	166	...
0	156	158	162	165	166	166	...
0	155	155	158	162	167	167	...
0	154	152	152	157	167	167	...
...	...	...	...	...	...	...	...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1



308

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2



-498

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



164

$$+ 1 = -25$$

$$\text{Bias} = 1$$

-25			...
			...
			...
			...
...	...	...	...

Output

NERA

THE FRENCH AEROSPACE LAB

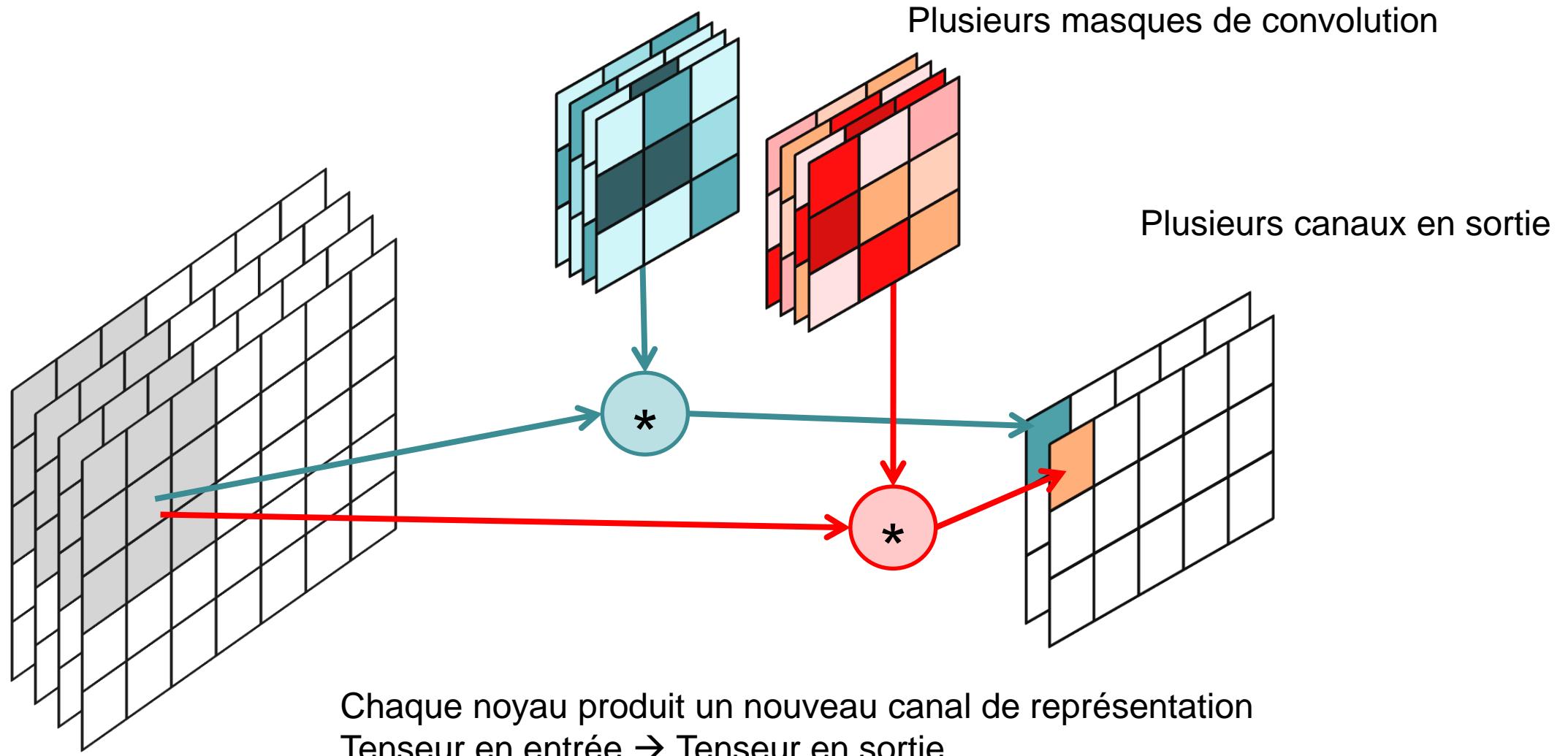
# Convolution et couche entièrement connectée

- Image  $x$  en entrée de  $C$  canaux et image en sortie scalaire, de tailles  $H \times L$
- Un noyau de convolution 3D de taille  $2\delta_H + 1 \times 2\delta_L + 1$
- Image en sortie

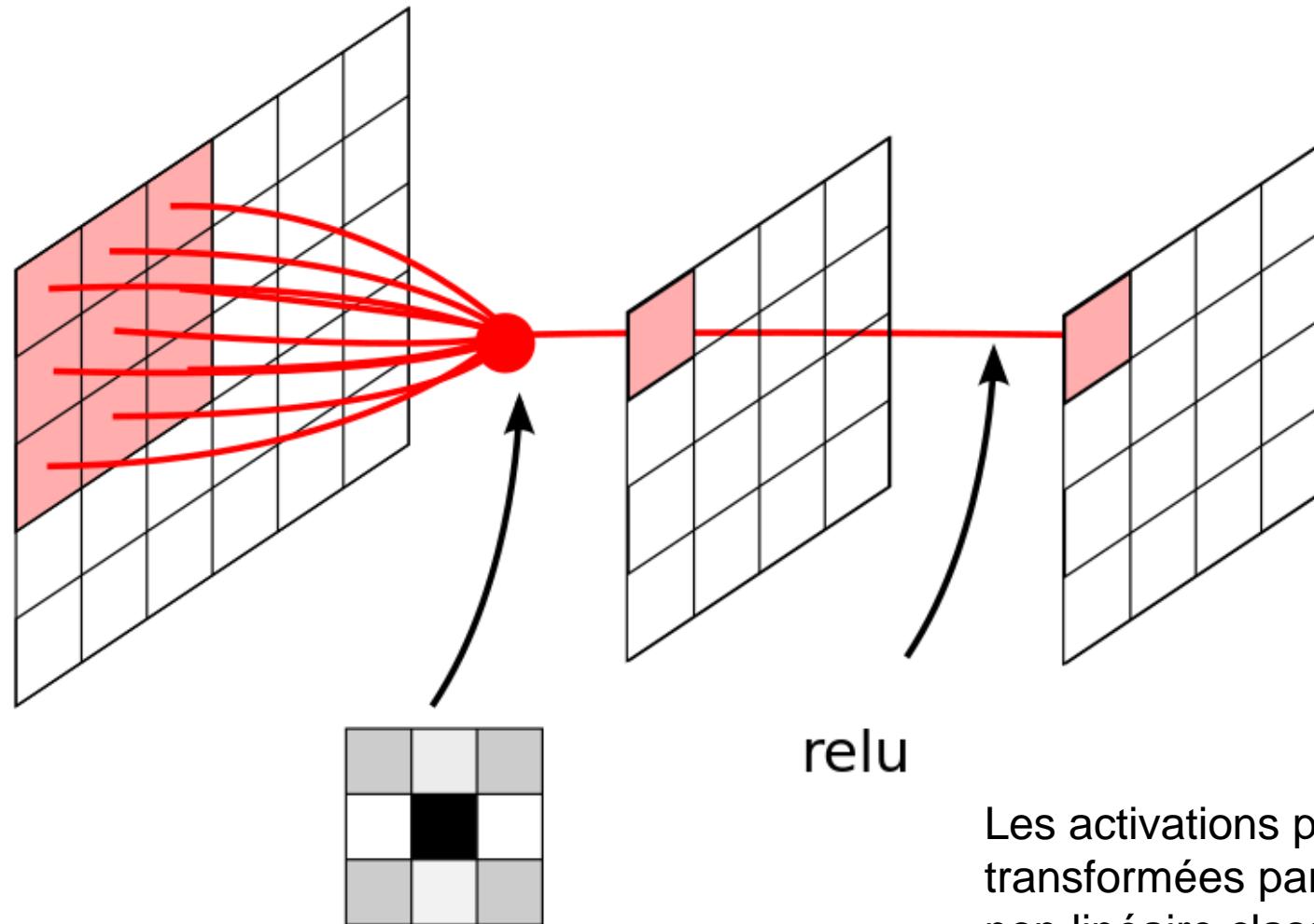
$$y(h, l) = \sum_{c=0}^{C-1} \sum_{i=0}^{2\delta_H} \sum_{j=0}^{2\delta_L} x(c, h - i + \delta_H, l - j + \delta_L). w(c, i, j)$$

- Les poids à apprendre sont les valeurs du noyau linéaire  $w(c, i, j)$
- On retrouve la formulation des réseaux de neurones
- Mais: la convolution n'a que  $O(\delta_H \times \delta_L)$  paramètres pour générer l'image, contre  $O(H^2 \times L^2)$  avec une couche classique entièrement connectée:
- Ex: 9 paramètres pour un noyau 3x3, contre >4Giga pour une image 256x256

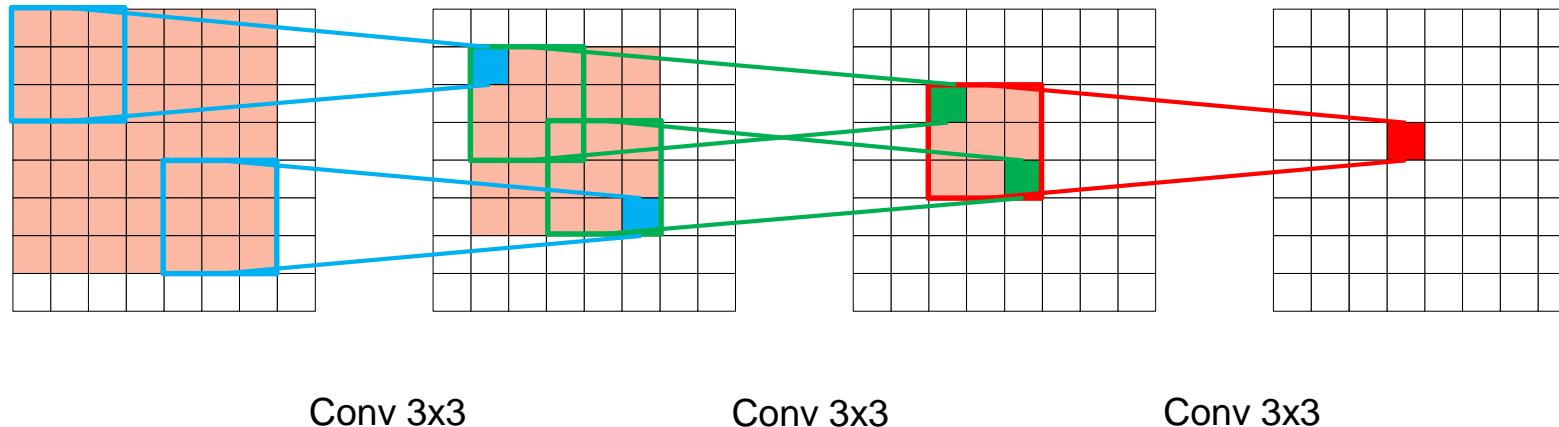
# Convolutions 3D



# Activation



# Champ récepteur



- Ce que « voit » un neurone
- Augmente linéairement selon la taille du noyau
- Pour représenter les interactions à long terme, il faut donc beaucoup de couches
- Mais:
  - On peut utiliser du sous-échantillonnage pour augmenter le champ récepteur
  - RF 3 convolutions 3x3 = RF 1 convolution 7x7, mais  $3^9 < 7^7$  paramètres (+ non linéarités)

# « Stride » et « Padding »

- On peut « déplacer » le noyau avec des sauts supérieurs à 1 (« stride »)

Input	Kernel	Output
$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 \\ 0 & 3 & 4 & 5 & 0 \\ 0 & 6 & 7 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$	$* \quad \begin{matrix} 0 & 1 \\ 2 & 3 \end{matrix}$	$= \quad \begin{matrix} 0 & 8 \\ 6 & 8 \end{matrix}$

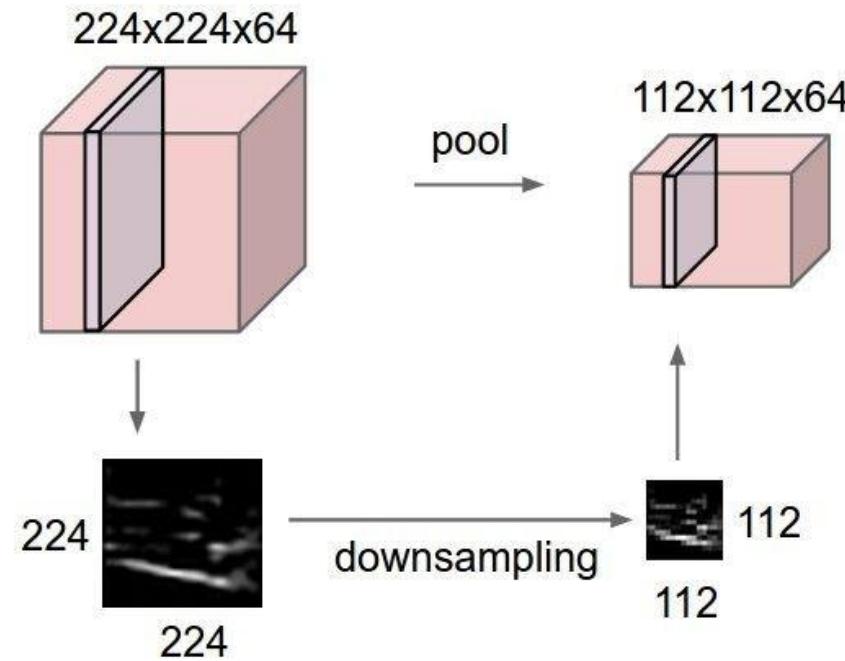
Stride = 2x3

- Pour garder une taille constante entre entrée et sortie, les données peuvent être complétées (« padding »)

Input	Kernel	Output
$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 \\ 0 & 3 & 4 & 5 & 0 \\ 0 & 6 & 7 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$	$* \quad \begin{matrix} 0 & 1 \\ 2 & 3 \end{matrix}$	$= \quad \begin{matrix} 0 & 3 & 8 & 4 \\ 9 & 19 & 25 & 10 \\ 21 & 37 & 43 & 16 \\ 6 & 7 & 8 & 0 \end{matrix}$

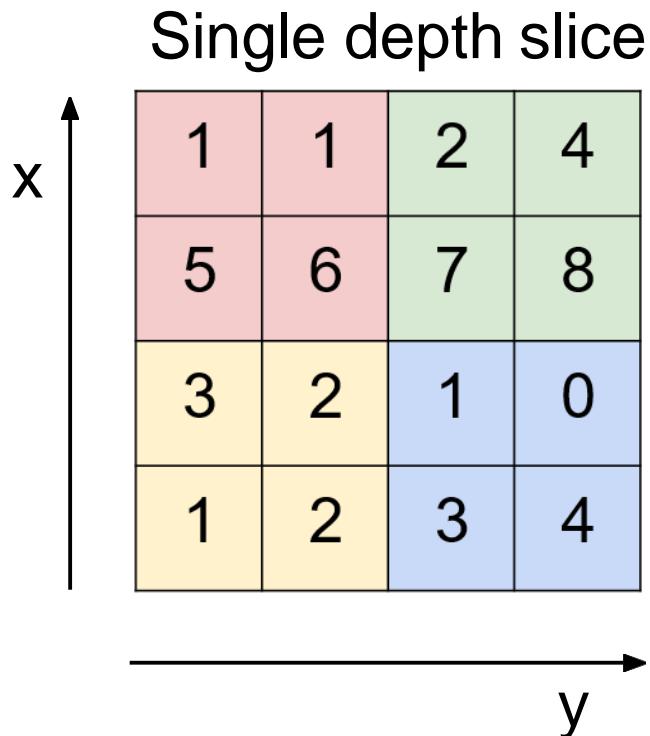
Padding pour noyau de taille 2x2

# Le « pooling »



- Rôle: réduire la taille des représentations (sous-échantillonnage, agrégation)
- Il faut que les opérations de réduction soient continues et différentiables pour calculer le gradient
- Plusieurs opérations possibles: max, moyenne...
- En fait, composition de fonction globale non paramétrique (max, mean, etc.) et « stride »

# Ex: Max pooling



max pool with 2x2 filters  
and stride 2

6	8
3	4

# Apprentissage des poids partagés

- Les poids sont les valeurs  $w(c, i, j)$  des noyaux de convolution

$$y(h, l) = \sum_{c=0}^{C-1} \sum_{i=0}^{2\delta_H} \sum_{j=0}^{2\delta_L} x(c, h - i + \delta_H, l - j + \delta_L) \cdot w(c, i, j)$$

- Calculer le gradient des poids **partagés**  $w(c, i, j)$  revient à sommer les gradients pour l'ensemble des variables contribuant à la fonction de coût:

$$\frac{\partial y(h, l)}{\partial w(c, i, j)} = x(c, h - i + \delta_H, l - j + \delta_L)$$

# Définir une couche de convolution

Couche de convolution dépend:

- Nombre de noyaux **K**
- Taille des noyaux **F**
- Pas d'échantillonnage (« stride ») **S**
- La complétion (« padding ») **P**
- Si tenseur d'entrée est de taille  $W_1 \times H_1 \times C$ , le tenseur de sortie sera de taille  $W_2 \times H_2 \times K$ , avec:  
$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$
  
$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

```
CLASS torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1,  
groups=1, bias=True, padding_mode='zeros', device=None, dtype=None) [SOURCE]
```

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size  $(N, C_{\text{in}}, H, W)$  and output  $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$  can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

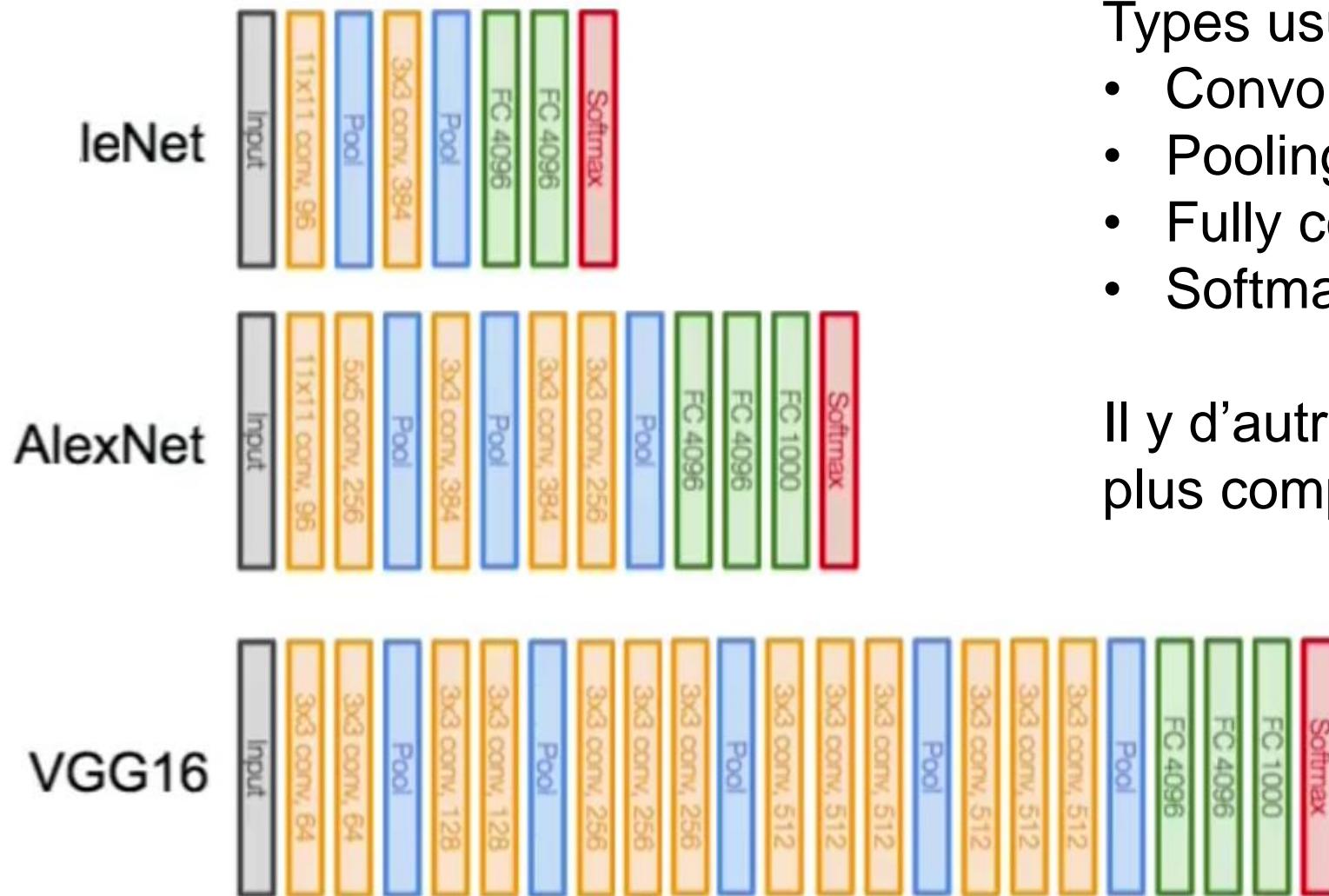
where  $\star$  is the valid 2D [cross-correlation](#) operator,  $N$  is a batch size,  $C$  denotes a number of channels,  $H$  is a height of input planes in pixels, and  $W$  is width in pixels.

This module supports [TensorFloat32](#).

On certain ROCm devices, when using float16 inputs this module will use [different precision](#) for backward.

- [stride](#) controls the stride for the cross-correlation, a single number or a tuple.
- [padding](#) controls the amount of padding applied to the input. It can be either a string {'valid', 'same'} or an int / a tuple of ints giving the amount of implicit padding applied on both sides.
- [dilation](#) controls the spacing between the kernel points; also known as the à trous algorithm. It is harder to describe, but this [link](#) has a nice visualization of what [dilation](#) does.
- [groups](#) controls the connections between inputs and outputs. [in\\_channels](#) and [out\\_channels](#) must both be divisible by [groups](#). For example,
  - At [groups=1](#), all inputs are convolved to all outputs.
  - At [groups=2](#), the operation becomes equivalent to having two conv layers side by side, each seeing half the input channels and producing half the output channels, and both subsequently concatenated.
  - At [groups= in\\_channels](#), each input channel is convolved with its own set of filters (of size  $\frac{\text{out\_channels}}{\text{in\_channels}}$ ).

# Deep Network = séquence de couches



Types usuels:

- Convolution
- Pooling
- Fully connected
- Softmax

Il y d'autres types de couches plus complexes

# Exemple: VGG16

INPUT: [224x224x3]      memory:  $224 \times 224 \times 3 = 150K$       params: 0      (not counting biases)

CONV3-64: [224x224x64]      memory:  $224 \times 224 \times 64 = 3.2M$       params:  $(3 \times 3 \times 3) \times 64 = 1,728$

CONV3-64: [224x224x64]      memory:  $224 \times 224 \times 64 = 3.2M$       params:  $(3 \times 3 \times 64) \times 64 = 36,864$

POOL2: [112x112x64]      memory:  $112 \times 112 \times 64 = 800K$       params: 0

CONV3-128: [112x112x128]      memory:  $112 \times 112 \times 128 = 1.6M$       params:  $(3 \times 3 \times 64) \times 128 = 73,728$

CONV3-128: [112x112x128]      memory:  $112 \times 112 \times 128 = 1.6M$       params:  $(3 \times 3 \times 128) \times 128 = 147,456$

POOL2: [56x56x128]      memory:  $56 \times 56 \times 128 = 400K$       params: 0

CONV3-256: [56x56x256]      memory:  $56 \times 56 \times 256 = 800K$       params:  $(3 \times 3 \times 128) \times 256 = 294,912$

CONV3-256: [56x56x256]      memory:  $56 \times 56 \times 256 = 800K$       params:  $(3 \times 3 \times 256) \times 256 = 589,824$

CONV3-256: [56x56x256]      memory:  $56 \times 56 \times 256 = 800K$       params:  $(3 \times 3 \times 256) \times 256 = 589,824$

POOL2: [28x28x256]      memory:  $28 \times 28 \times 256 = 200K$       params: 0

CONV3-512: [28x28x512]      memory:  $28 \times 28 \times 512 = 400K$       params:  $(3 \times 3 \times 256) \times 512 = 1,179,648$

CONV3-512: [28x28x512]      memory:  $28 \times 28 \times 512 = 400K$       params:  $(3 \times 3 \times 512) \times 512 = 2,359,296$

CONV3-512: [28x28x512]      memory:  $28 \times 28 \times 512 = 400K$       params:  $(3 \times 3 \times 512) \times 512 = 2,359,296$

POOL2: [14x14x512]      memory:  $14 \times 14 \times 512 = 100K$       params: 0

CONV3-512: [14x14x512]      memory:  $14 \times 14 \times 512 = 100K$       params:  $(3 \times 3 \times 512) \times 512 = 2,359,296$

CONV3-512: [14x14x512]      memory:  $14 \times 14 \times 512 = 100K$       params:  $(3 \times 3 \times 512) \times 512 = 2,359,296$

CONV3-512: [14x14x512]      memory:  $14 \times 14 \times 512 = 100K$       params:  $(3 \times 3 \times 512) \times 512 = 2,359,296$

POOL2: [7x7x512]      memory:  $7 \times 7 \times 512 = 25K$       params: 0

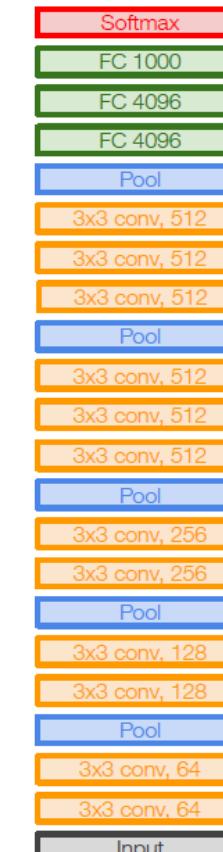
FC: [1x1x4096]      memory: 4096      params:  $7 \times 7 \times 512 \times 4096 = 102,760,448$

FC: [1x1x4096]      memory: 4096      params:  $4096 \times 4096 = 16,777,216$

FC: [1x1x1000]      memory: 1000      params:  $4096 \times 1000 = 4,096,000$

**TOTAL** memory:  $24M * 4 \text{ bytes} \approx 96\text{MB} / \text{image}$  (for a forward pass)

**TOTAL** params: 138M parameters



Nombre de paramètres importants dans couches hautes

Convolutions à noyaux petits (3x3)

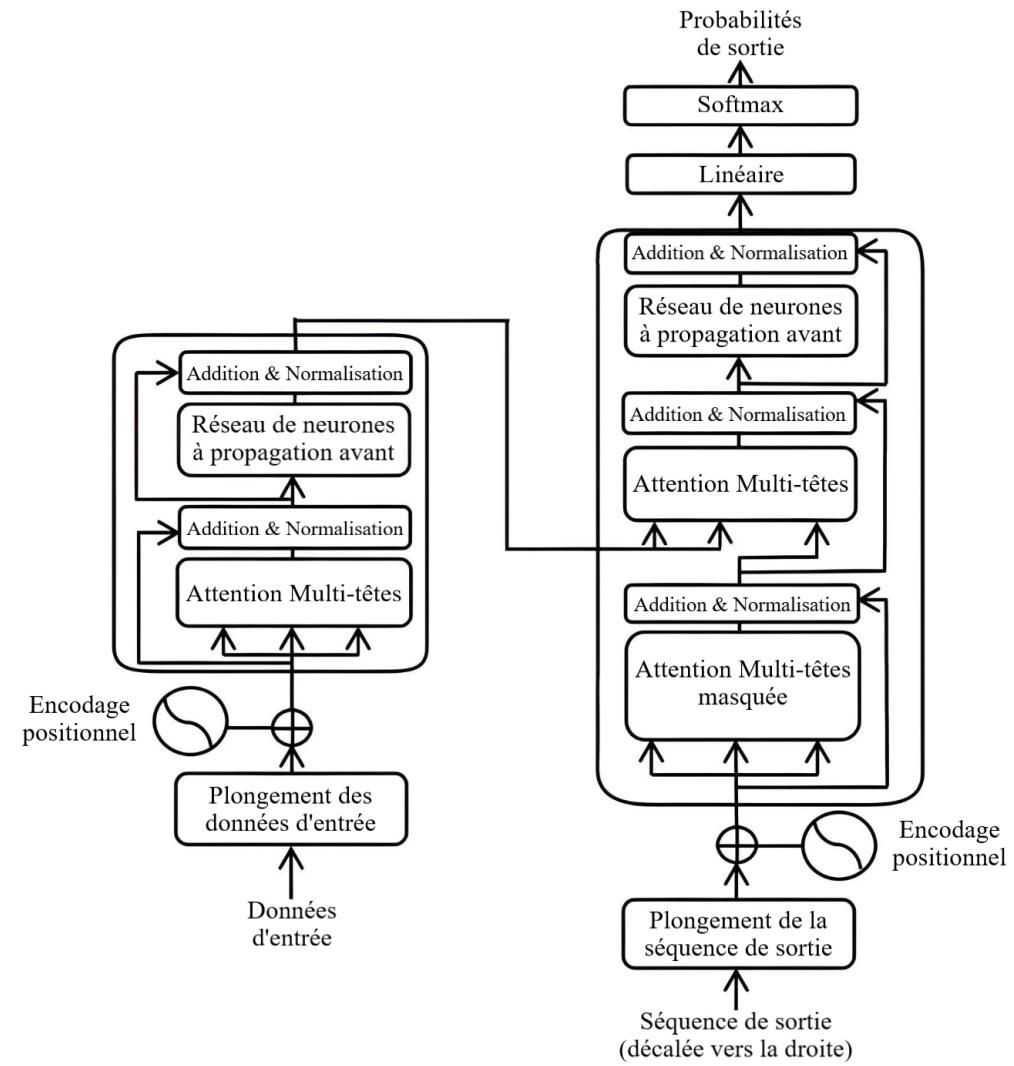
Mémoire importante dans couches basses

VGG16

- <https://poloclub.github.io/cnn-explainer/>

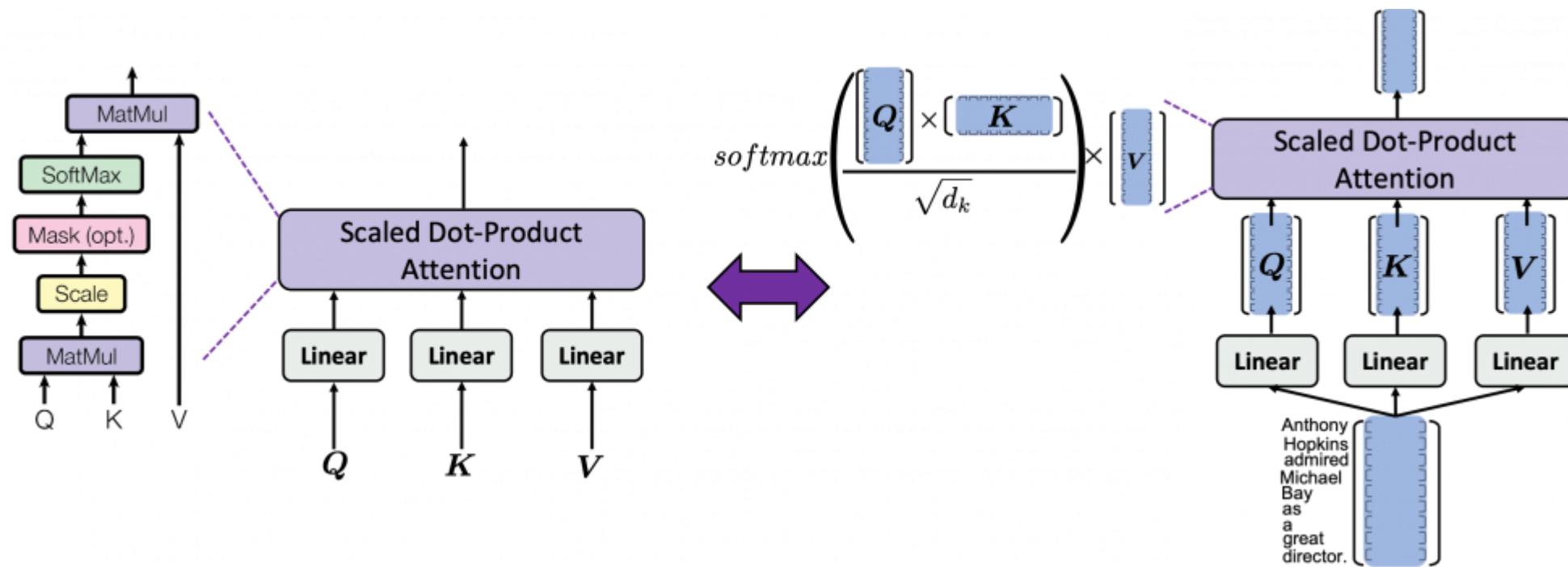
# L'autre architecture: « Transformers »

- « Attention is all you need »
- Architecture encodeur/décodeur
- Utilisé au départ pour de la traduction:  
séquence → séquence
- Justification: réussir à exploiter les interactions longue portée  
Interaction = auto-attention
- Transformer = garde la structure (la séquence) mais transforme les codes.
- Pas de réduction de dimension (= pas de *stride* ou *pooling*)
- Multi channel (convolution) → Multi head
- Position codée par ajout d'un signal (encore un code, mais connu!)



A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention Is All You Need,” in Advances in Neural Information Processing Systems 30 (NIPS 2017), 2017.

# Query, Key, Value



- Query = le signal à transformer = un ensemble de vecteurs
- Key = l'index ou le vecteur du code
- Value = la transformation associée à chaque code

# Query, Key, Value

- Interprétation: décomposition selon un dictionnaire « transformé » de l'entrée  $Q$

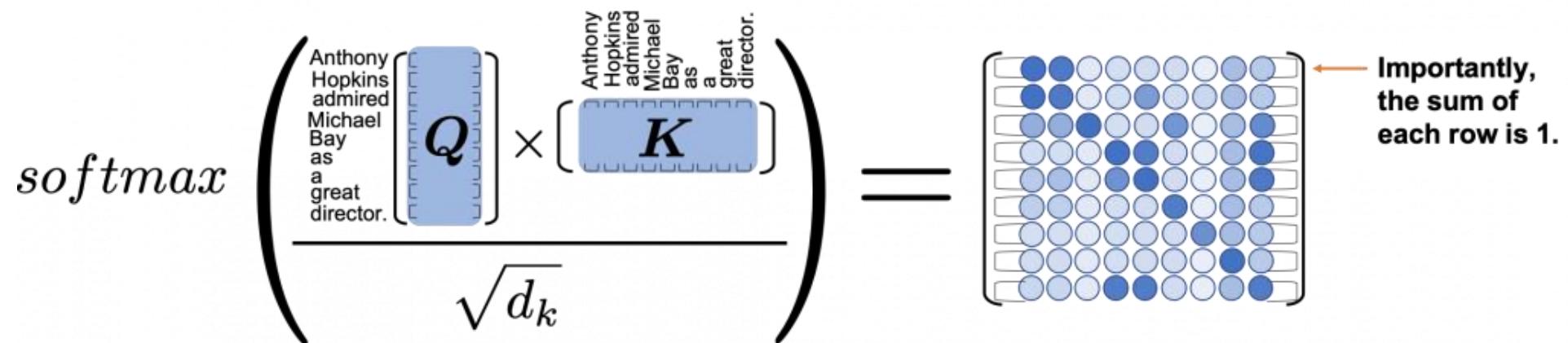
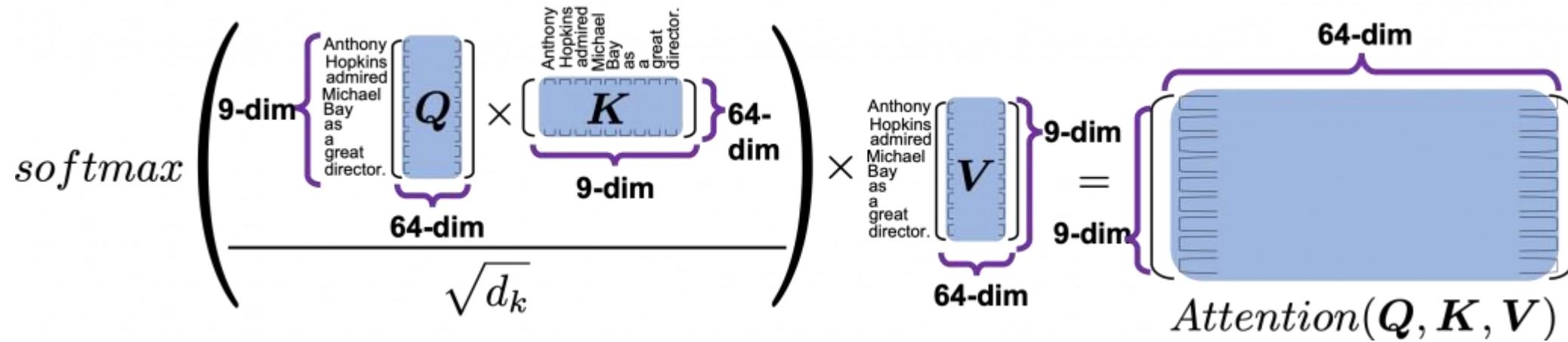
$$Y_i = \sum_j \langle Q_i, K_j \rangle V_j$$

$$Y_i = \sum_j \langle W_q Q_i, W_k K_j \rangle W_v V_j$$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{W_q Q [W_k K]^T}{\sqrt{d_k}}\right) W_v V$$

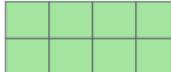
- Intérêt: Transforme un signal de taille quelconque en un signal de même taille.
- Adapté au texte si chaque  $Q_i$  est un vecteur
- Self-attention:  $Q = K = V$ , Cross attention:  $Q \neq V$

# Query, Key, Value

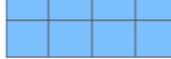


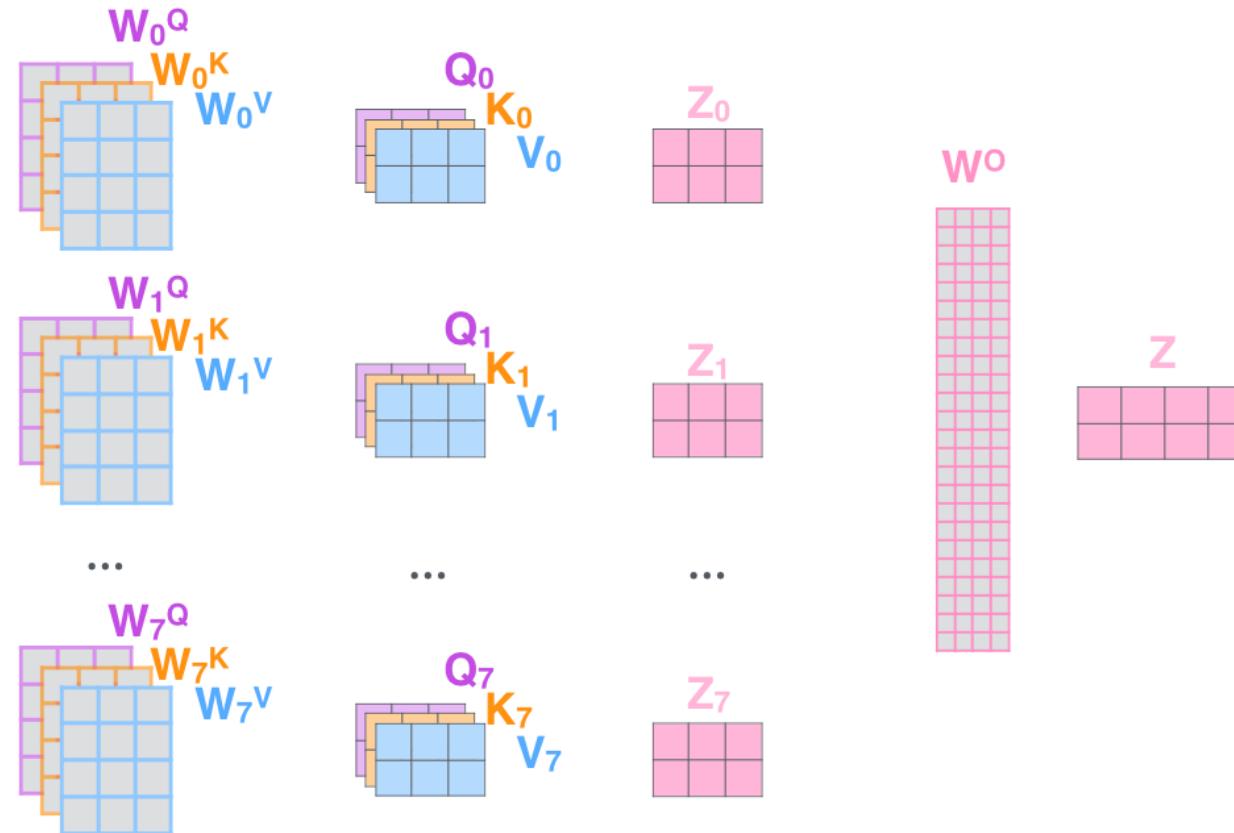
# Multi head = multi canaux

- 1) This is our input sentence\*  $X$
- 2) We embed each word\*
- 3) Split into 8 heads. We multiply  $X$  or  $R$  with weight matrices
- 4) Calculate attention using the resulting  $Q/K/V$  matrices
- 5) Concatenate the resulting  $Z$  matrices, then multiply with weight matrix  $W^o$  to produce the output of the layer

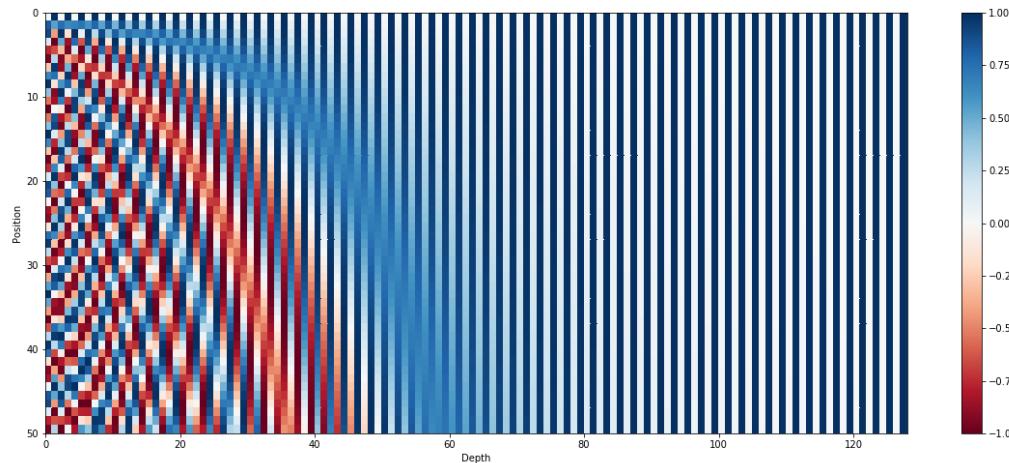
Thinking Machines  
 $X$   


\* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

$R$   




# Positional encoding



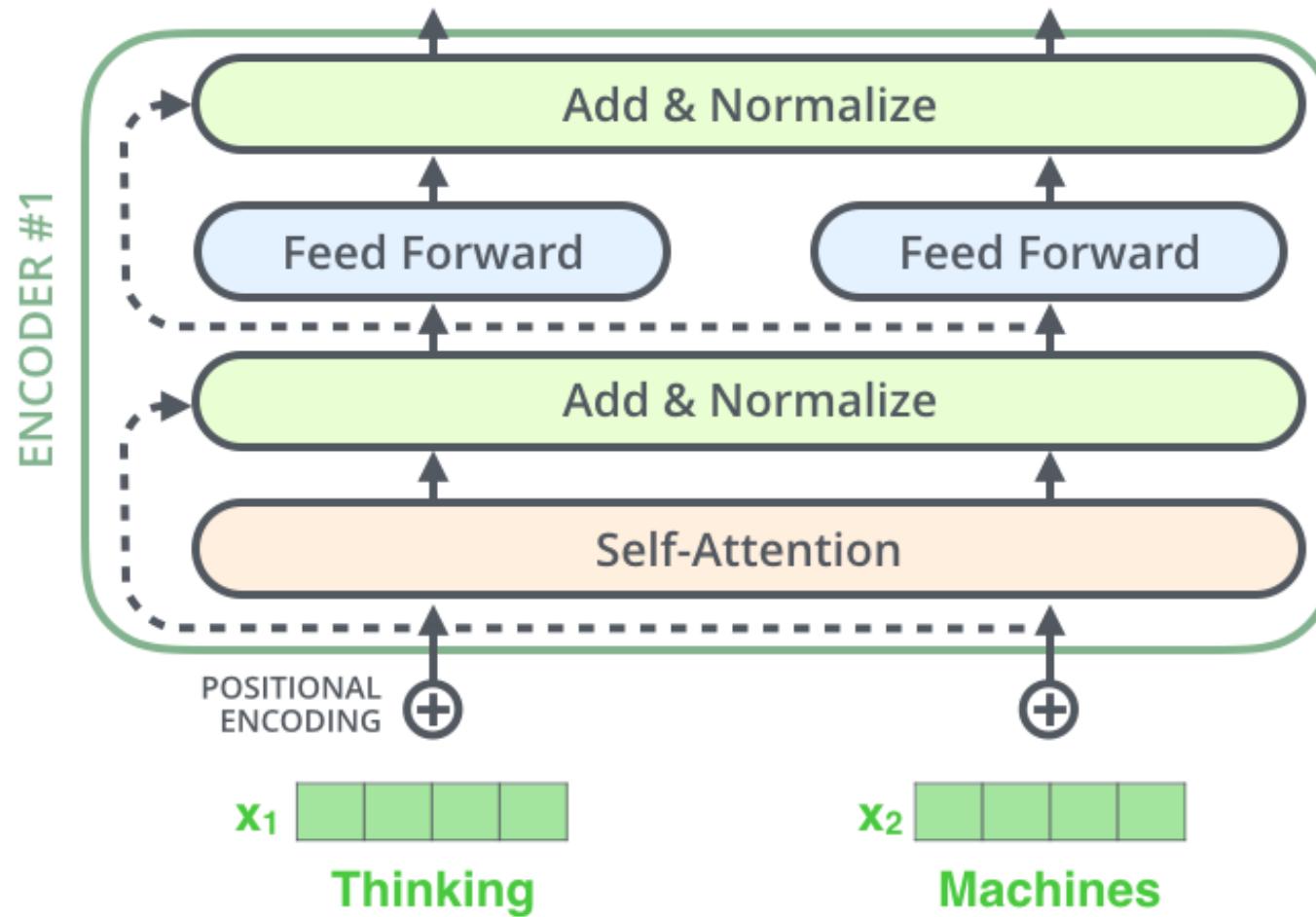
- Stratégie pour coder la localisation des éléments d'entrée, leur indice: utile pour représenter les « distances » (relatives ou absolues).
- Astuce: on ajoute un signal sinusoïdal faible qui dépend de la localisation  $t$   
$$p(t) = [\sin(w_1 \cdot t), \cos(w_1 \cdot t), \sin(w_2 \cdot t), \cos(w_2 \cdot t), \dots, \sin(w_{d/2} \cdot t), \cos(w_{d/2} \cdot t)]$$

Où

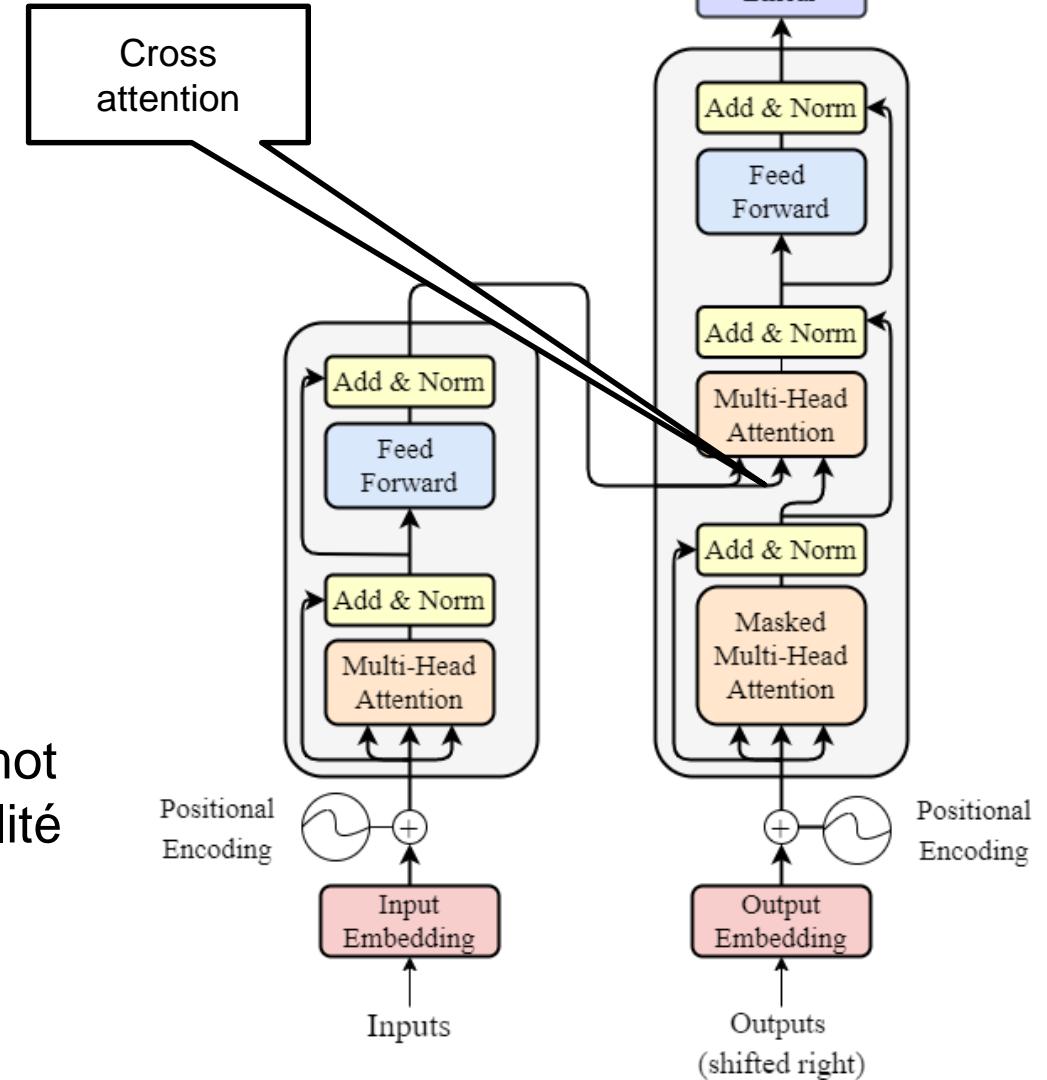
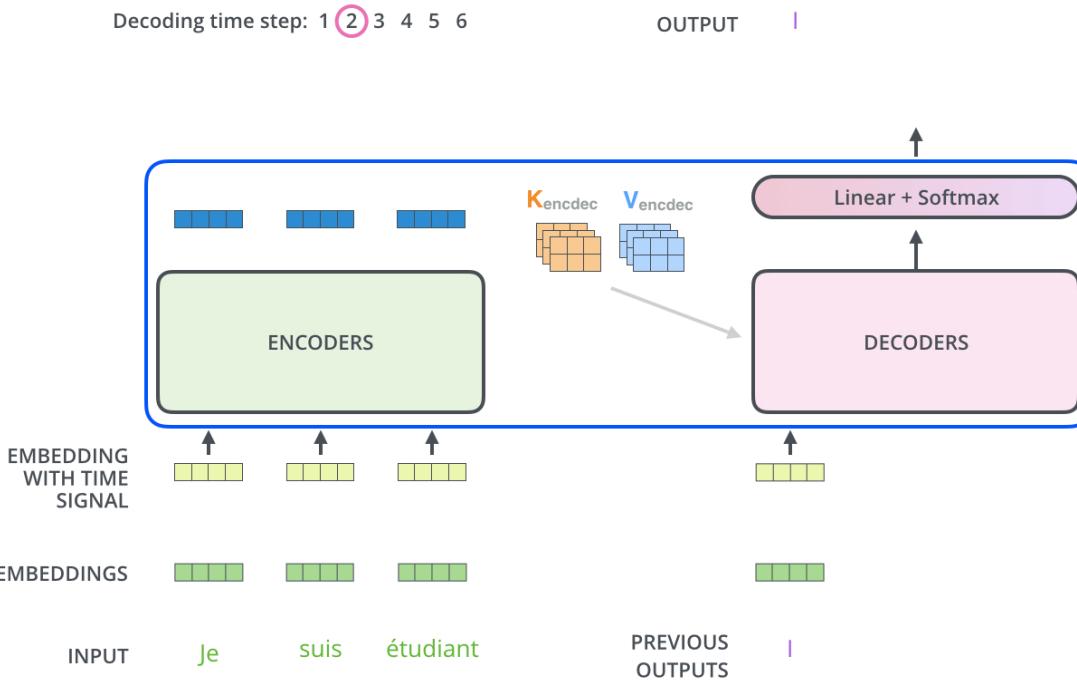
$$w_k = \frac{1}{10000^{2k/d}}$$

- La différence entre deux codes ne dépend pas de  $t$

# La couche « encoder » complète

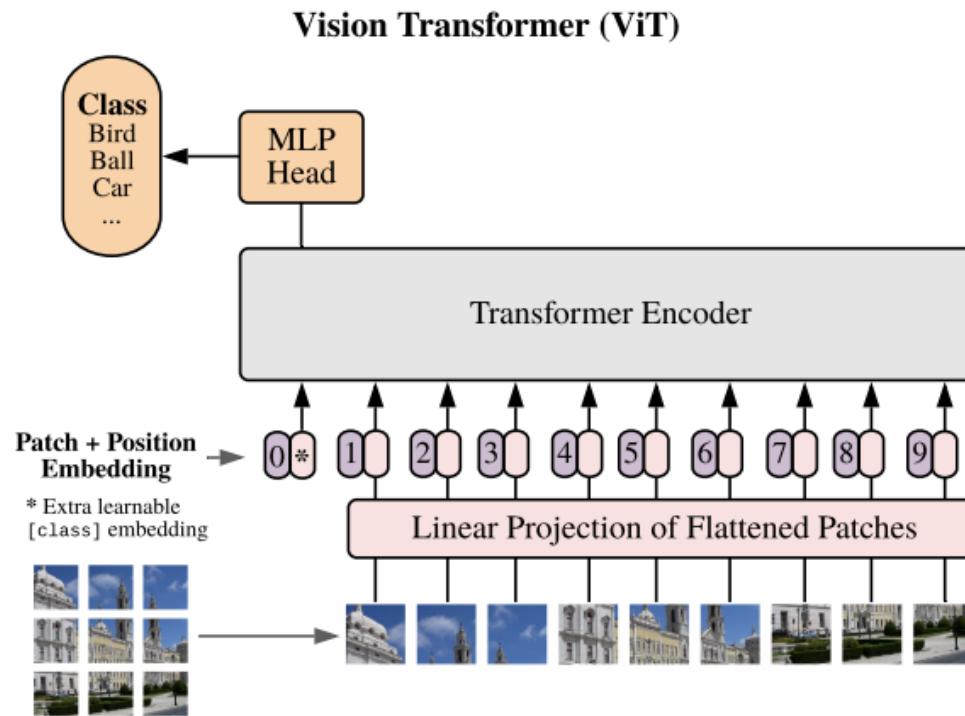


# Decodeur « transformer »

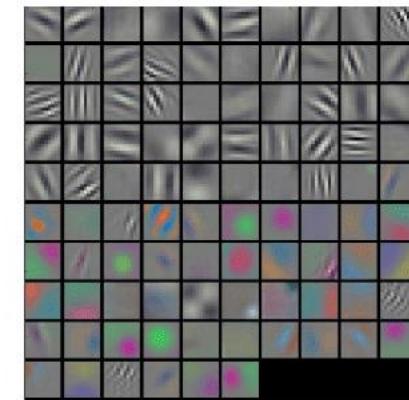


- Modèle auto-régressif: ajoute séquentiellement le mot décodé en entrée du décodeur pour générer la totalité de la séquence
- Étage de « cross attention » pour combiner code et processus de décodage

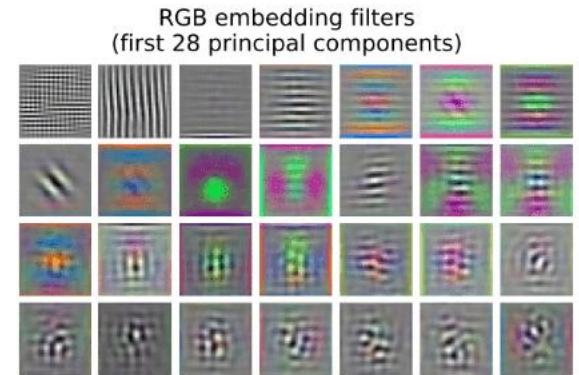
# ViT : transformer pour l'image



Alexnet 1st conv filters



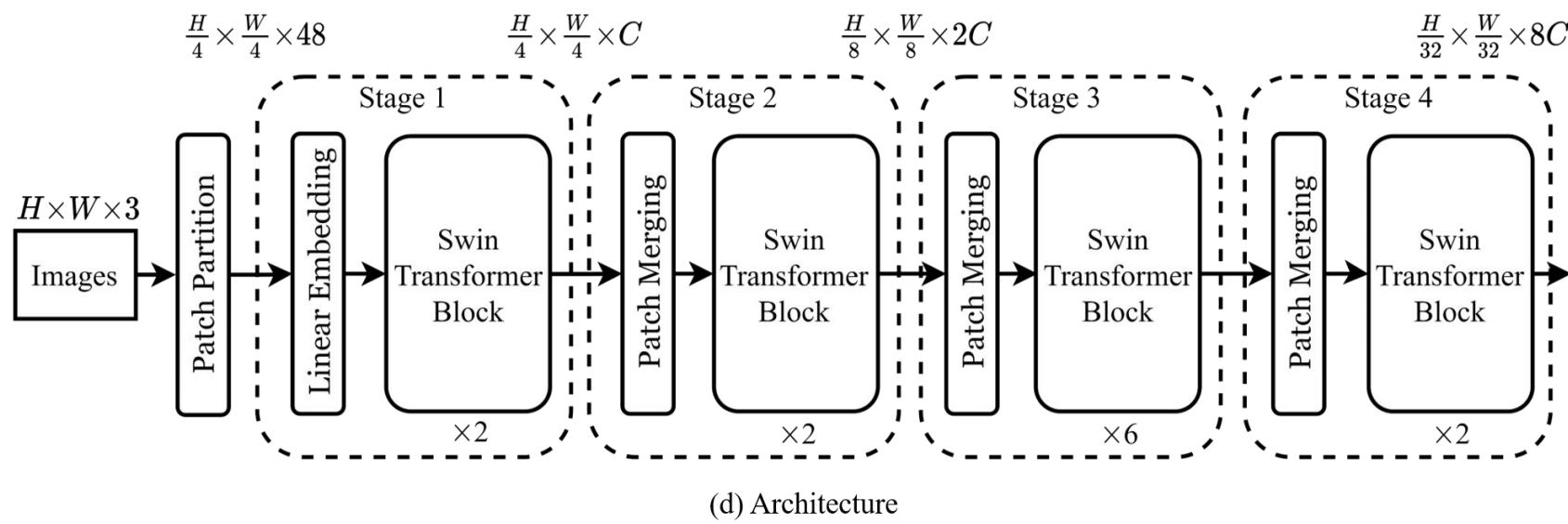
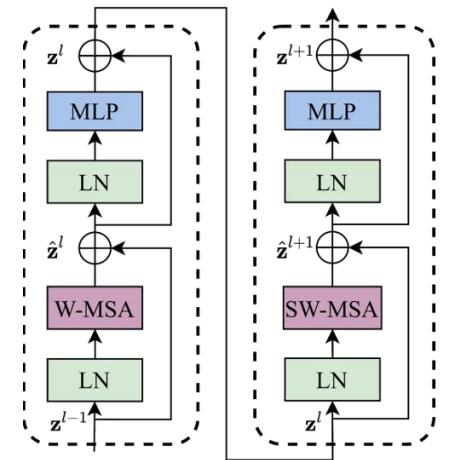
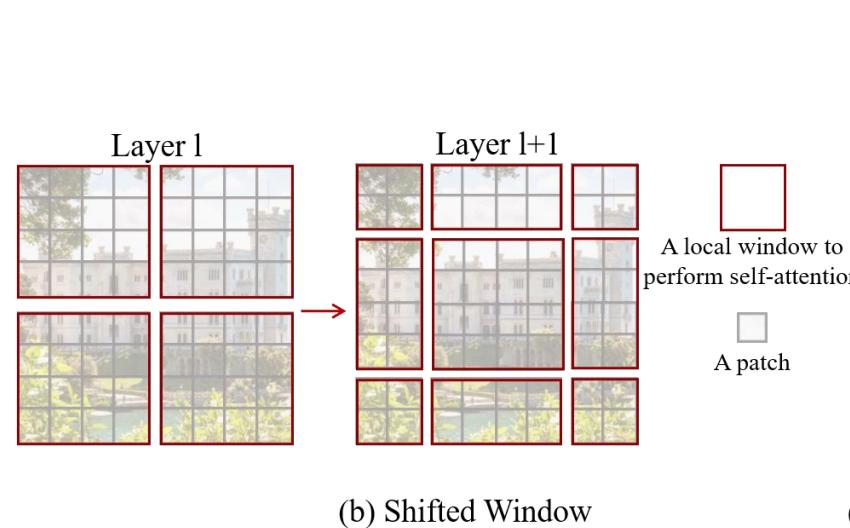
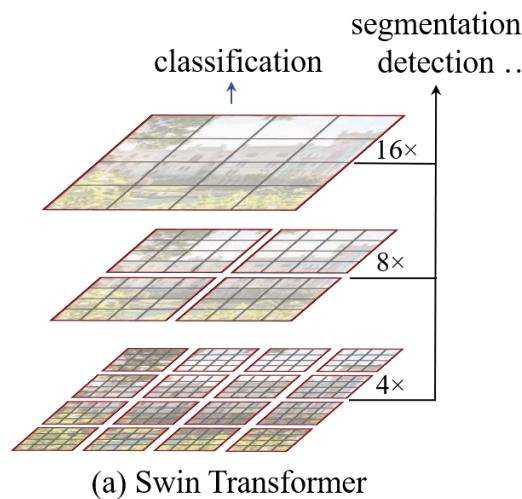
ViT 1st linear embedding filters



- Décompose une image en liste de patch et l'encode avec un transformer
- Un « token » fictif encode l'ensemble de l'information
- Utilisé pour classification, segmentation, détection...

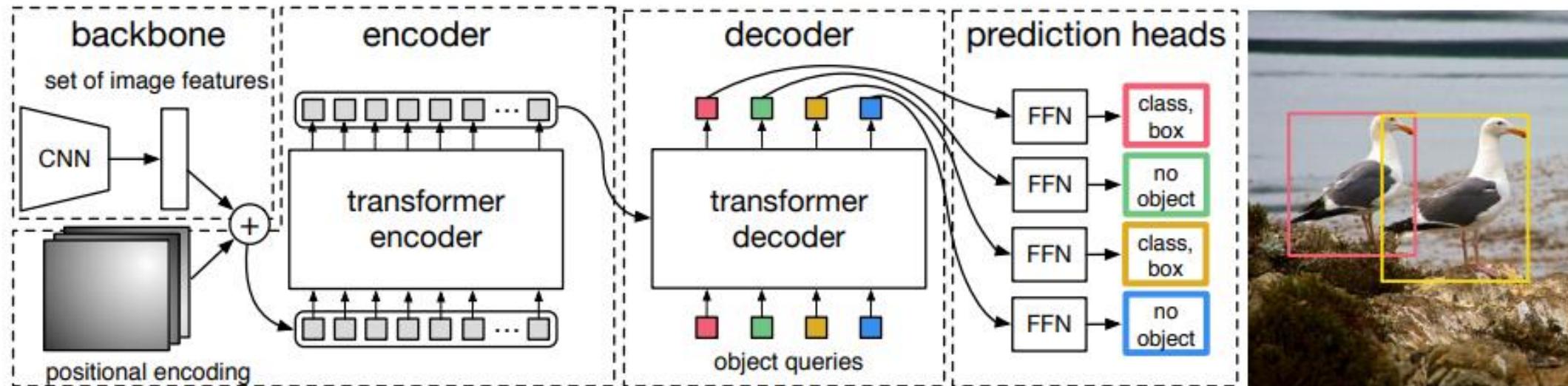
Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020, October). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In International Conference on Learning Representations.

# SWIN: un transformer image multi-échelle



Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., & Guo, B. (2021). Swin Transformer : Hierarchical Vision Transformer Using Shifted Windows. ICCV.

# DETR: détecter = transformer des représentations



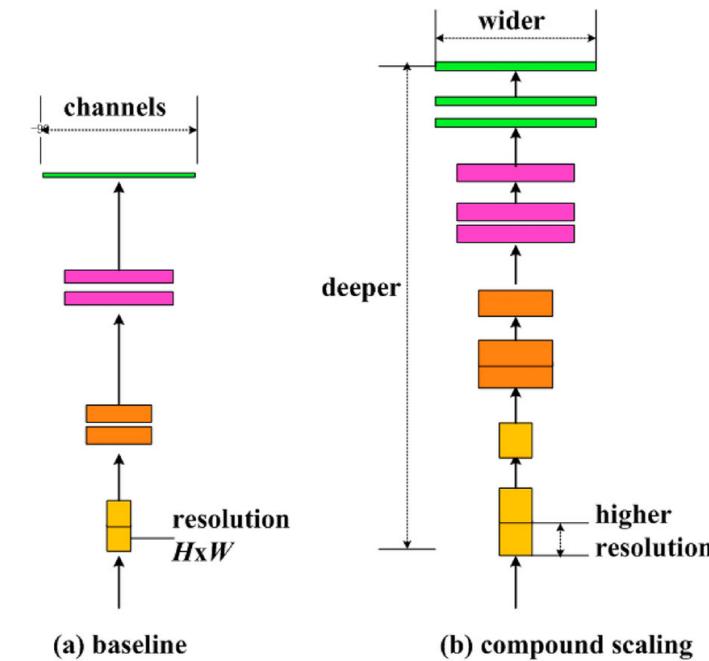
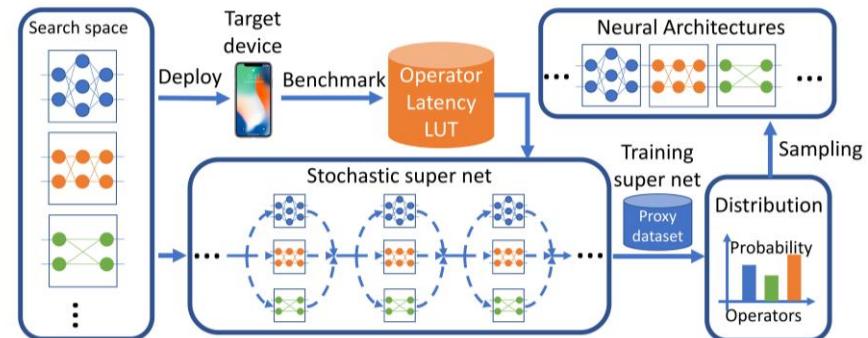
Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S. (2020). End-to-end object detection with transformers. In European conference on computer vision (pp. 213-229).

# Ce qui influe (potentiellement) sur les performances

1. L'architecture globale
2. Gestion du pas du gradient
3. Le type d'activation
4. Méthodes de régularisation: Drop out, weight decay
5. « Batch normalization »
6. Couches résiduelles
7. Taille du Batch
8. Initialisation des poids
9. Augmentation de données
10. Pré-apprentissage et « fine tuning »

# Une architecture optimale ?

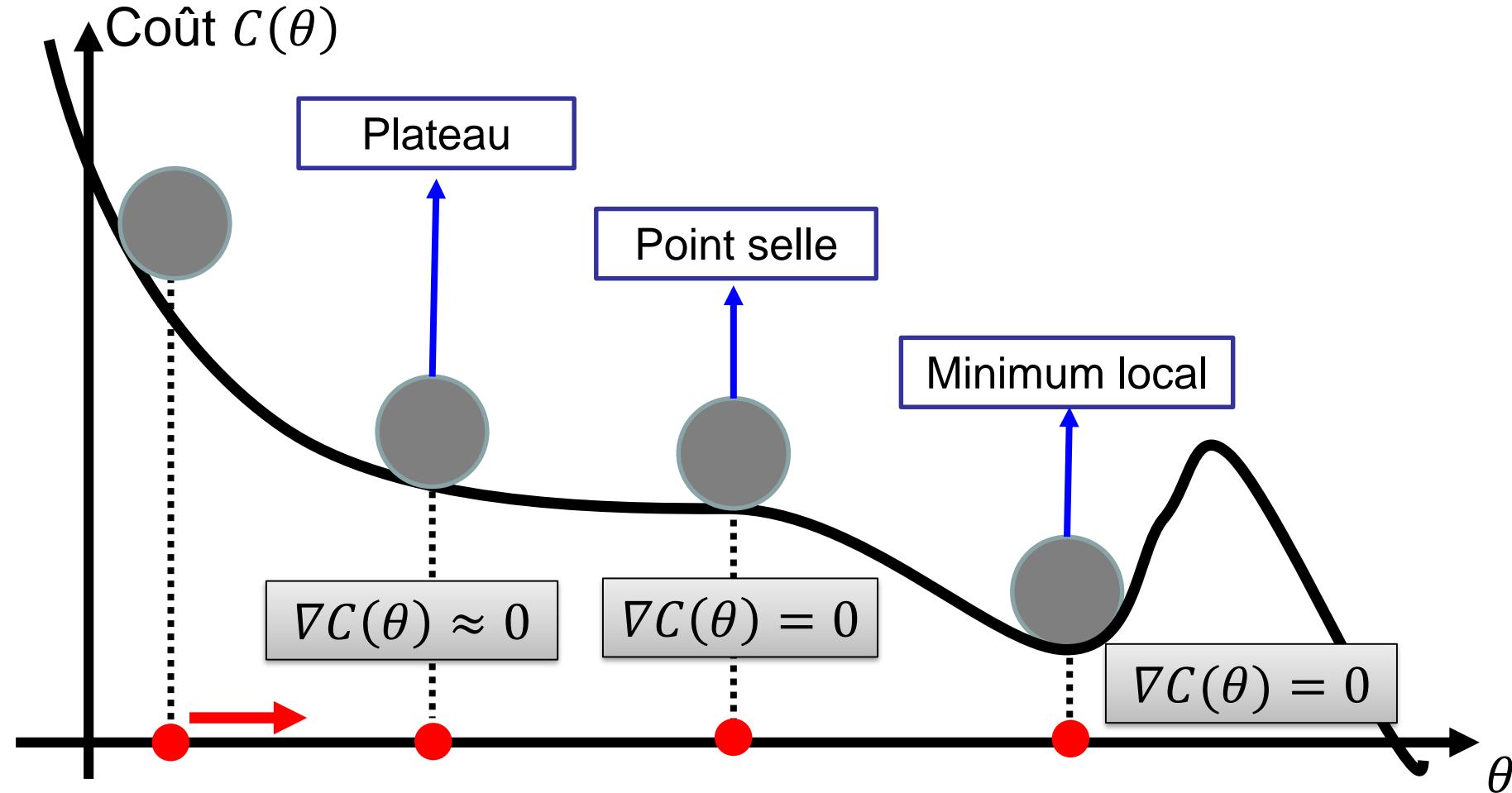
- Neural Architecture Search: un domaine de recherche à part entière.
- Compromis performances/temps d'exécution/mémoire...
- Ex: EfficientNet, une architecture « modulaire » et paramétrable
  - Compromis Largeur x Profondeur x Résolution x Canaux



Tan, M., & Le, Q. (2019, May). Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning* (pp. 6105-6114). PMLR.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., ... & Keutzer, K. (2019). Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 10734-10742).

# Les « mauvais » comportements de la descente de gradient



# Mieux gérer le pas du gradient

Mise à jour au temps  $t$  du paramètre  $\theta^t$  par incrément  $V^t$ :

$$\theta^t = \theta^{t-1} - V^t$$

- Momentum:  $V^t = \beta V^{t-1} + \alpha \nabla \mathcal{L}(\theta^{t-1})$

La direction de mise à jour est une moyenne pondérée des gradients précédents

- Nesterov:  $V^t = \beta V^{t-1} + \alpha \nabla \mathcal{L}(\theta^{t-1} + \beta V^{t-1})$

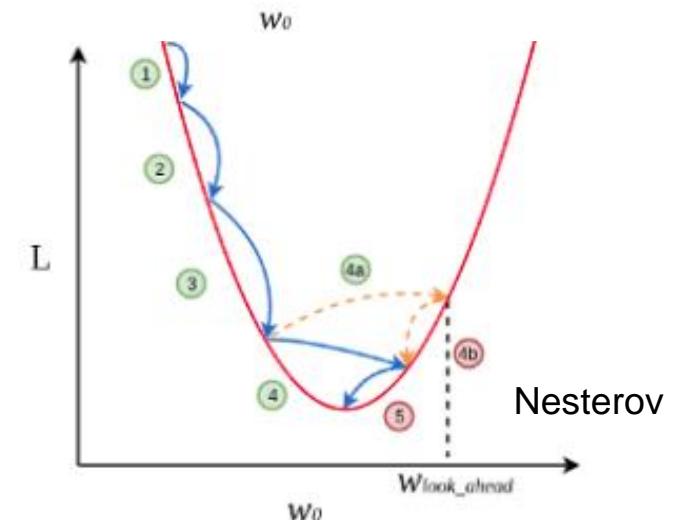
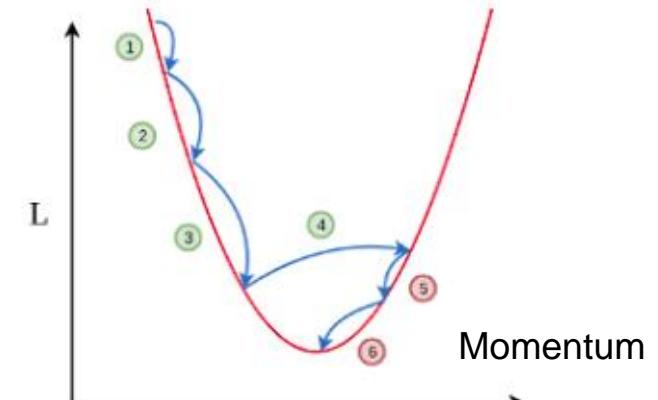
Pas du gradient pondéré par la prédiction en un pas

- Adaptation (RMS prop):  $V^t = \frac{\alpha \nabla \mathcal{L}(\theta^{t-1})}{\sqrt{U^t + \epsilon}}$

avec  $U^t = \alpha U^{t-1} + (1 - \alpha) \nabla \mathcal{L}(\theta^{t-1})^2$

Normalisation par la norme cumulée du gradient

- Adam: Momentum + RMS prop
- ...



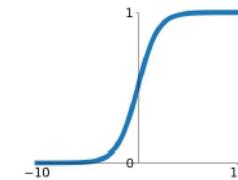
# L'intérêt du ReLU (et cousins)

Défauts des fonctions d'activation

- Ecrase les gradients: Sigmoïde, tanh (« Extinction »)
- Gradient nul pour certaines valeurs: Relu
- Non centrée: Sigmoïde, Relu et al.
- Espace mémoire: Maxout
- Calculatoire: Sigmoïde, ELU
- En pratique
  1. Utiliser ReLU pour produire des gradients même pour les grandes valeurs, mais contrôler avec soin le taux d'apprentissage.
  2. Si besoin, utiliser Leaky ReLU / Maxout / ELU
  3. Ne pas utiliser sigmoid or tanh

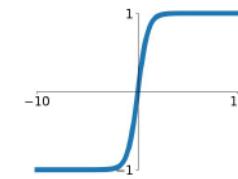
**Sigmoid**

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



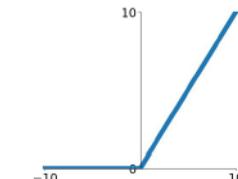
**tanh**

$$\tanh(x)$$



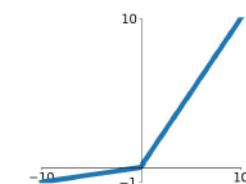
**ReLU**

$$\max(0, x)$$



**Leaky ReLU**

$$\max(0.1x, x)$$

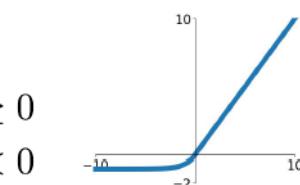


**Maxout**

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

**ELU**

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

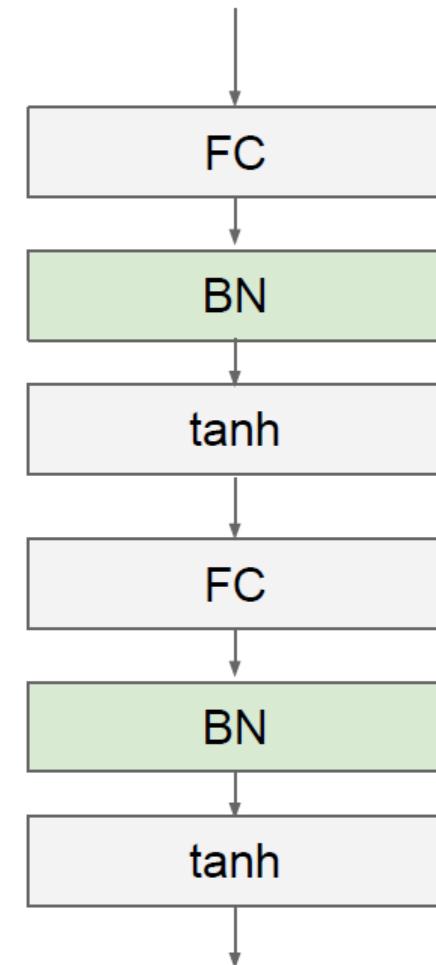


# « Batch normalization »

- Idée: normaliser les activations de chaque neurone en moyenne et intensité à partir des exemples de chaque minibatch

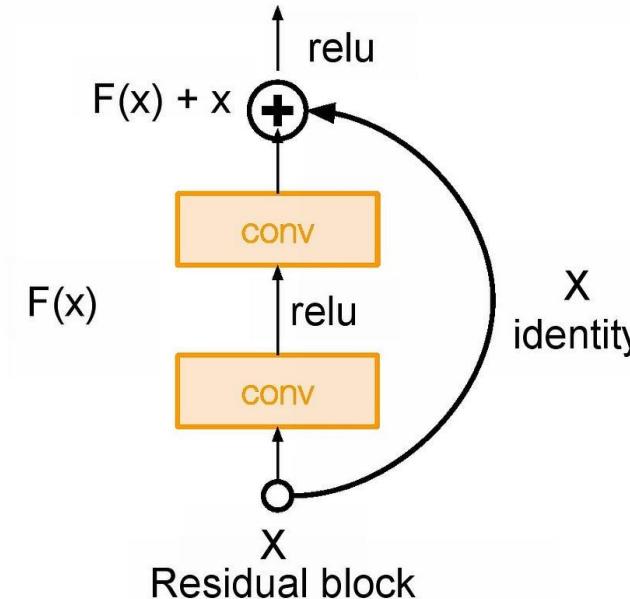
$$\hat{x}_{ij} = \frac{x_{ij} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

- Objectif: apprentissage plus stable et plus rapide, en général avec une meilleure généralisation
- S'implémente comme une couche
- La normalisation est faite pendant l'apprentissage et dépend des minibatch → elle peut s'apprendre comme une couche linéaire.
- D'autres types de normalisation: par couche, par instance, par groupe.

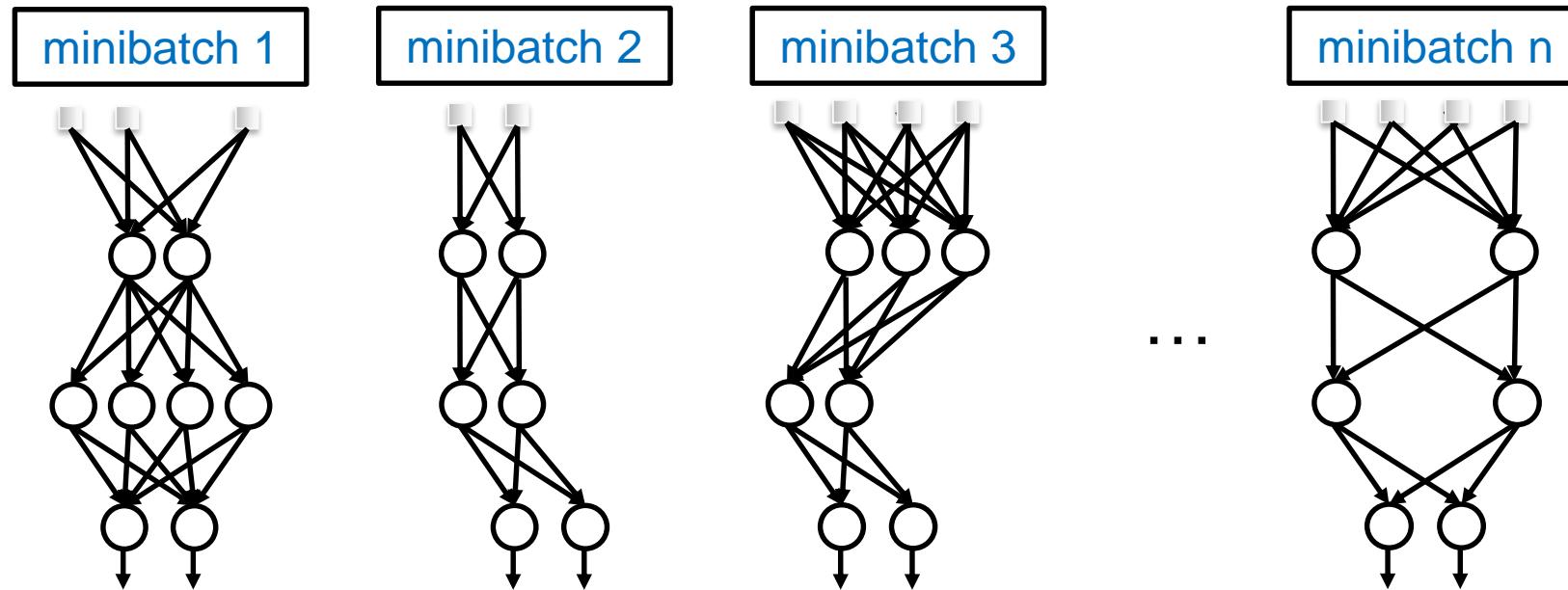


# Les couches résiduelles

- Couche résiduelle non strictement séquentielle (« skip connection »)
- Principe: on apprend un résidu d'approximation par rapport à l'identité
- Meilleure stabilité numérique
- Evite l'extinction du gradient des premières couches
- ➔ permet d'enchaîner un très grand nombre de couches ( $> 100$ )
- Utilisé dans un très grand nombre de réseaux récents (ResNet, Transformer, BERT...)



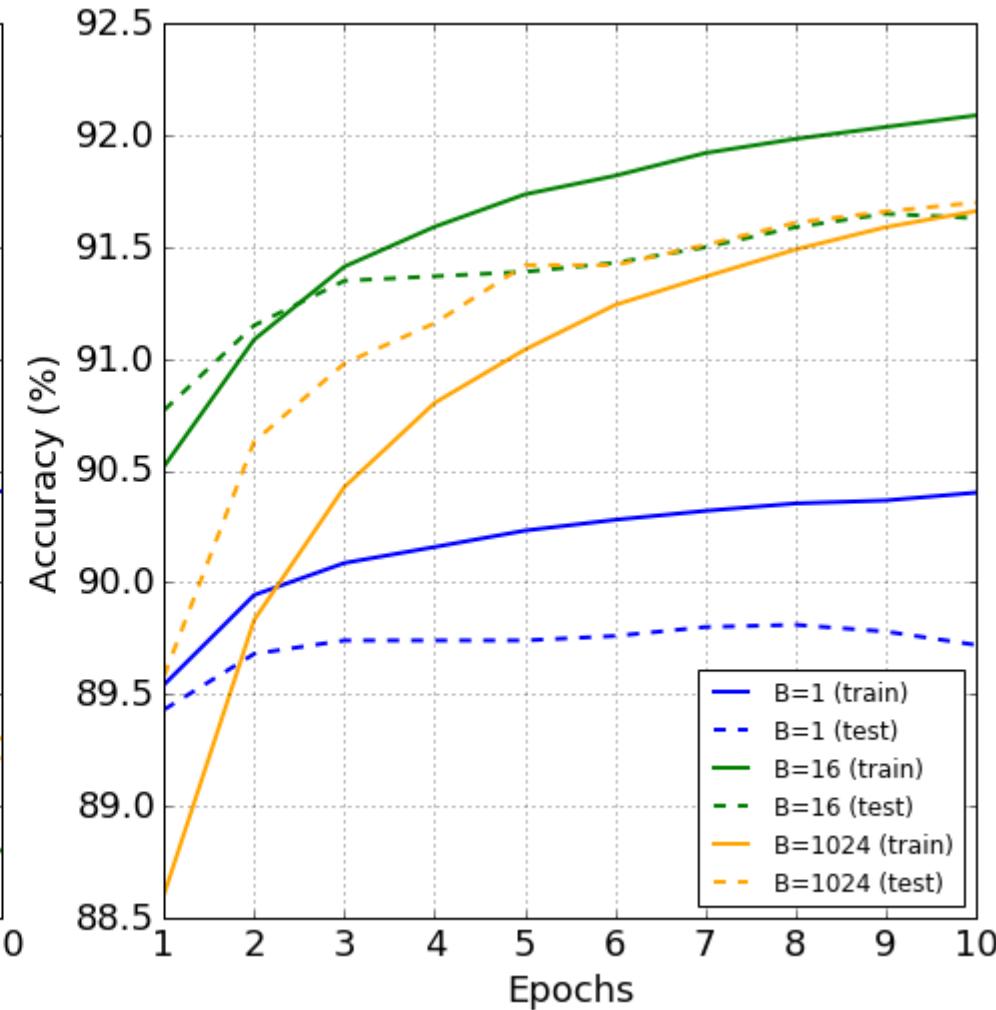
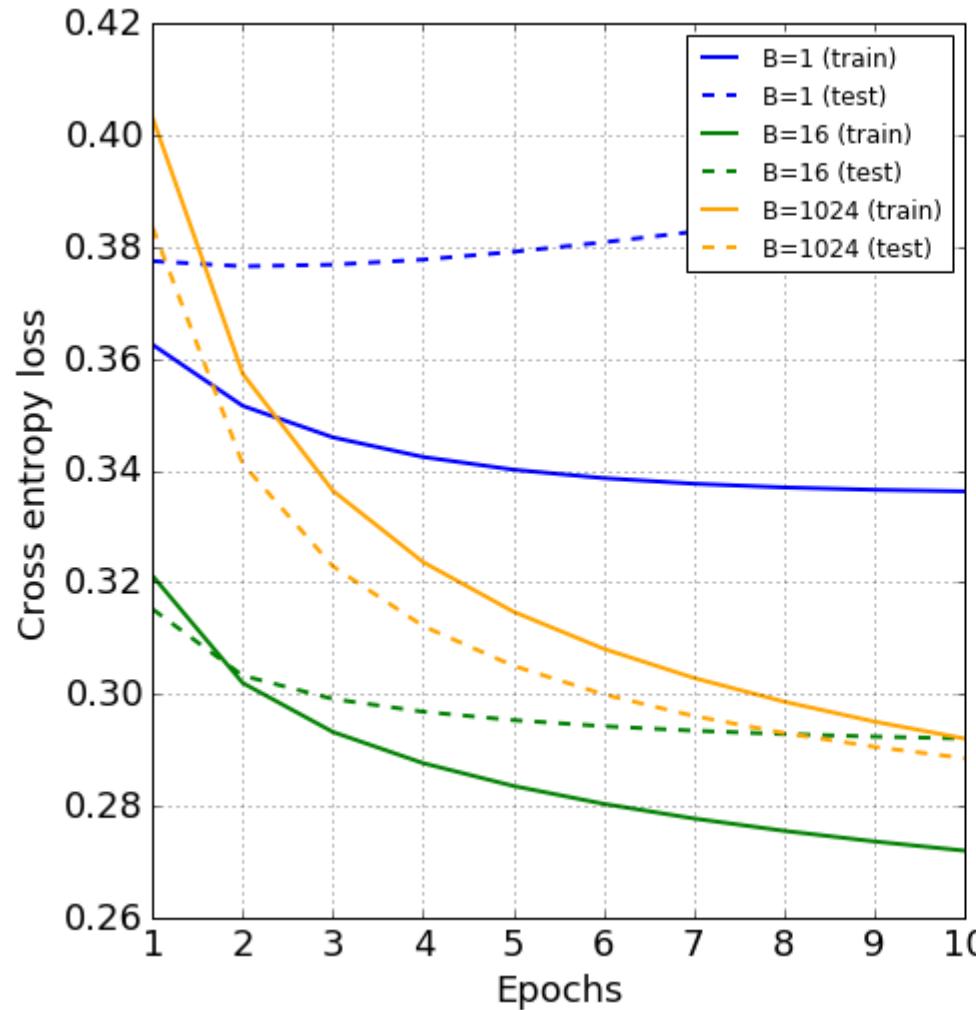
# « Drop out »



- Principe: ne mettre à jour qu'une proportion aléatoire  $p$  des paramètres pour chaque minibatch lors de l'apprentissage
- Lors du test, tous les paramètres sont actifs + normalisation pour que les activations moyennes Train et Test soient de même amplitude
- Peut être implémenté comme une couche
- Peut être interprété comme une méthode ensembliste (réseau final = ensemble de sous réseaux) → améliore la généralisation

# Impact de la taille du “minibatch”

Taux d'apprentissage  $\eta$  normalisé :  $\eta = 0.025 \cdot \sqrt{B}$

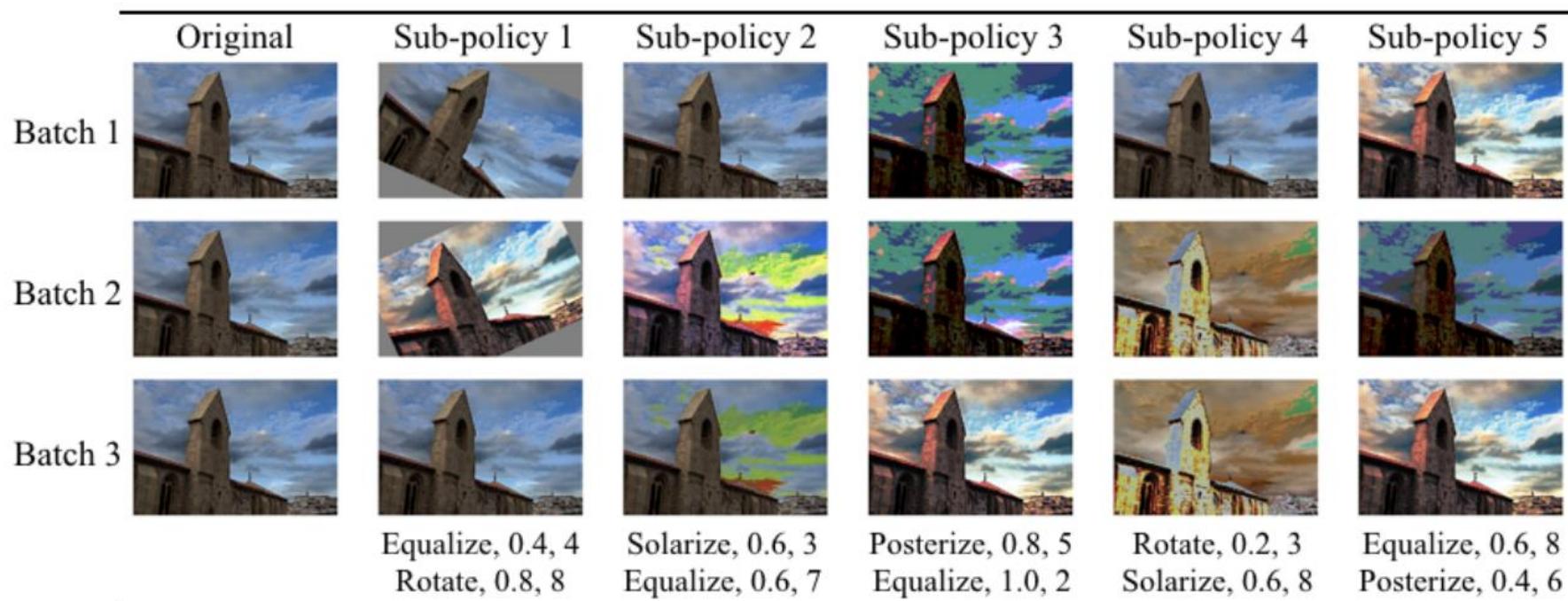


# Initialisation des réseaux

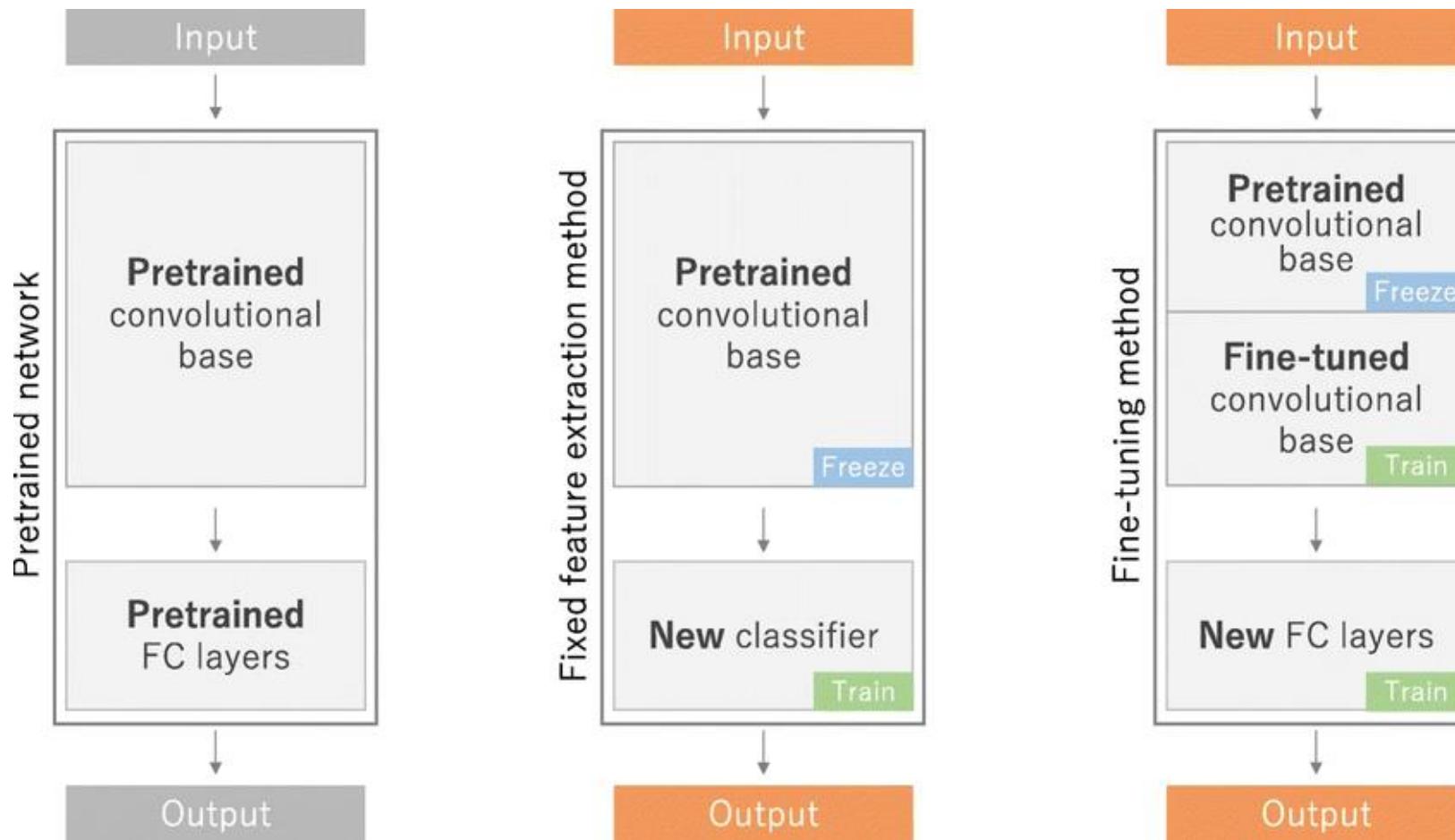
- L'algorithme d'optimisation est
  - stochastique
  - doit gérer plusieurs minima locaux
- Le résultat dépend de l'initialisation des paramètres: comment la choisir?
- Une stratégie simple (« Xavier »): les variances d'entrée et de sortie doivent être égales.
- Si la donnée d'entrée est de taille  $n$ :
$$Y = w_1 \cdot x_1 + w_2 \cdot x_2 + \dots + w_n \cdot x_n$$
- Sous hypothèse d'indépendance de  $w_i$  et  $x_i$ , et de centralité:
$$\text{var}(Y) = n \cdot \text{var}(X) \cdot \text{var}(W)$$
- Ce qui donne:
$$\text{var}(W) = \frac{1}{n}$$
- On initialise les poids selon une loi gaussienne de variance  $1/n$
- Il existe d'autres types d'initialisation.

# Augmentation de données

- En ML, la quantité de données disponible conditionne les performances
- Augmentation de données: générer de nouvelles données à partir de celles disponibles et cohérentes avec leur nature.
- Ex pour images: rotations, flip, crop, colorisation, changement de contraste...
- Implémenté dans torchvision



# « Transfer learning » & « Fine-tuning »



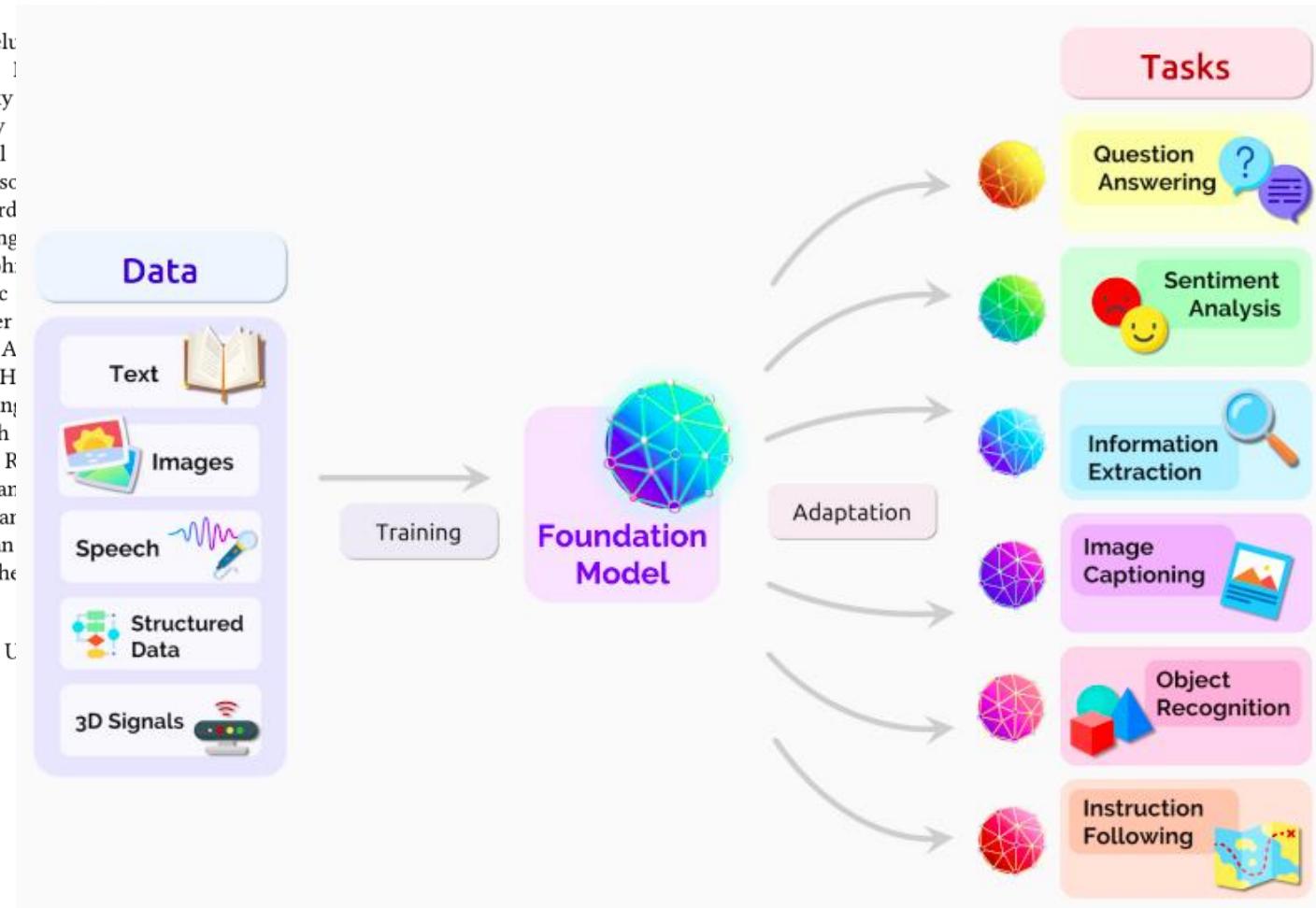
- Principe: Collecter un réseau appris sur une base de données de grande taille (ex. ImageNet) et l'adapter aux nouvelles données

# Modèles de fondation

## On the Opportunities and Risks of Foundation Models

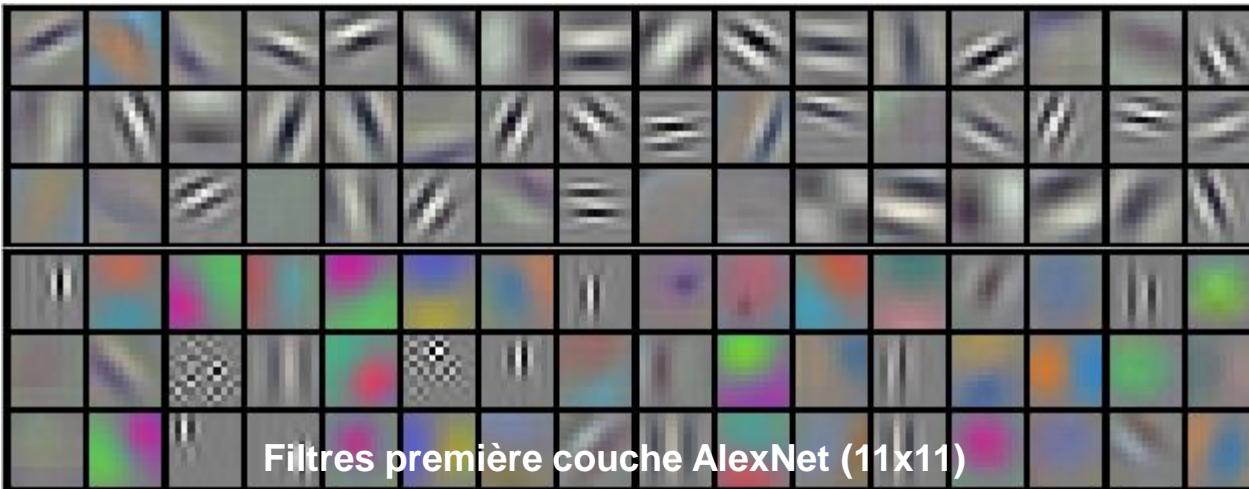
Rishi Bommasani\* Drew A. Hudson Ehsan Adeli Russ Altman  
Sydney von Arx Michael S. Bernstein Jeannette Bohg Antoine Bosselut  
Erik Brynjolfsson Shyamal Buch Dallas Card Rodrigo Castellon<sup>†</sup>  
Annie Chen Kathleen Creel Jared Quincy Davis Dorottya Demszky  
Moussa Doumbouya Esin Durmus Stefano Ermon John Etchemendy  
Li Fei-Fei Chelsea Finn Trevor Gale Lauren Gillespie Karan Goel  
Shelby Grossman Neel Guha Tatsunori Hashimoto Peter Hendersen  
Daniel E. Ho Jenny Hong Kyle Hsu Jing Huang Thomas Icard  
Dan Jurafsky Pratyusha Kalluri Siddharth Karamcheti Geoff Keeling  
Omar Khattab Pang Wei Koh Mark Krass Ranjay Krishna Roh  
Ananya Kumar Faisal Ladzhak Mina Lee Tony Lee Jure Leskovec  
Xiang Lisa Li Xuechen Li Tengyu Ma Ali Malik Christopher  
Suvir Mirchandani Eric Mitchell Zanele Munyikwa Suraj Nair A  
Deepak Narayanan Ben Newman Allen Nie Juan Carlos Niebles H  
Julian Nyarko Giray O gut Laurel Orr Isabel Papadimitriou Joon Sung  
Eva Portelance Christopher Potts Aditi Raghunathan Rob Reich  
Frieda Rong Yusuf Roohani Camilo Ruiz Jack Ryan Christopher R  
Shiori Sagawa Keshav Santhanam Andy Shih Krishnan Srinivasar  
Rohan Taori Armin W. Thomas Florian Tramèr Rose E. Wang Williar  
Jiajun Wu Yuhuai Wu Sang Michael Xie Michihiro Yasunaga Jiaxuan  
Michael Zhang Tianyi Zhang Xikun Zhang Yuhui Zhang Lucia Zhe  
Percy Liang<sup>\*1</sup>

Center for Research on Foundation Models (CRFM) – Stanford U



# Comprendre les architectures

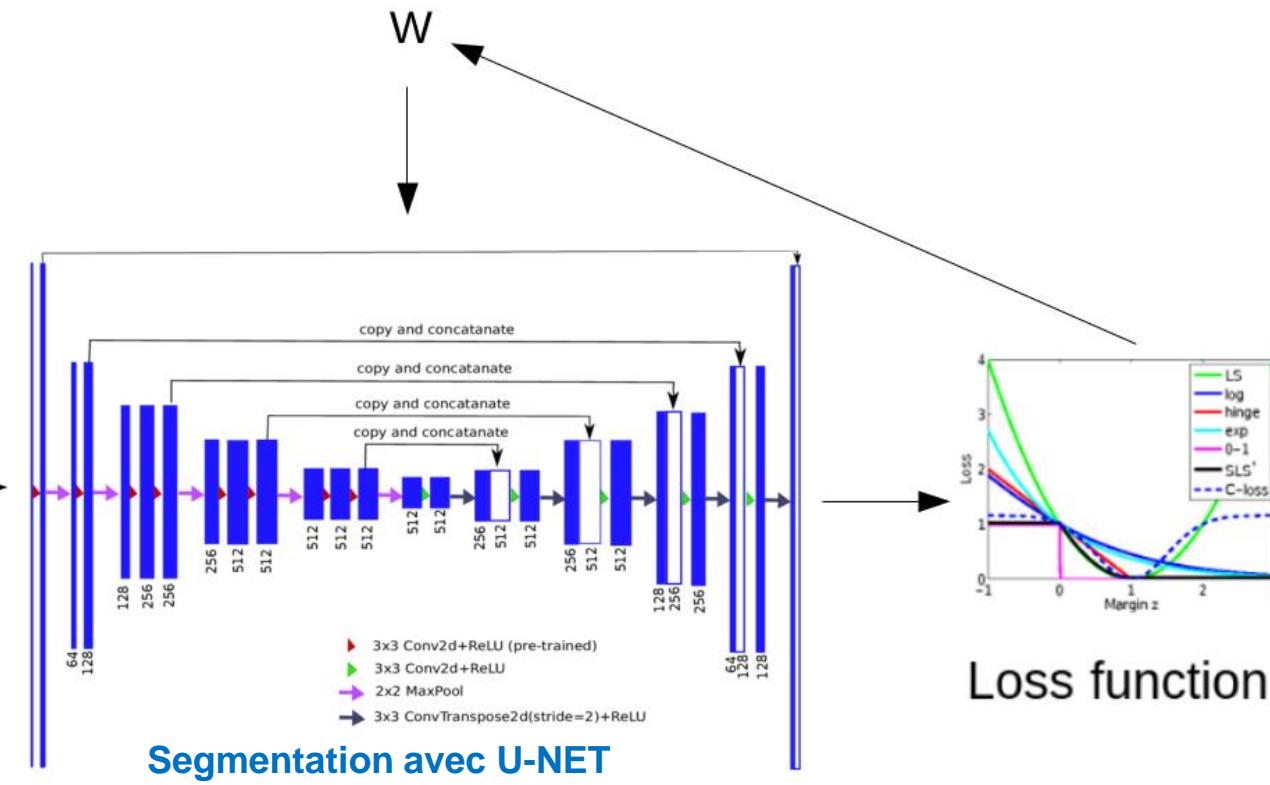
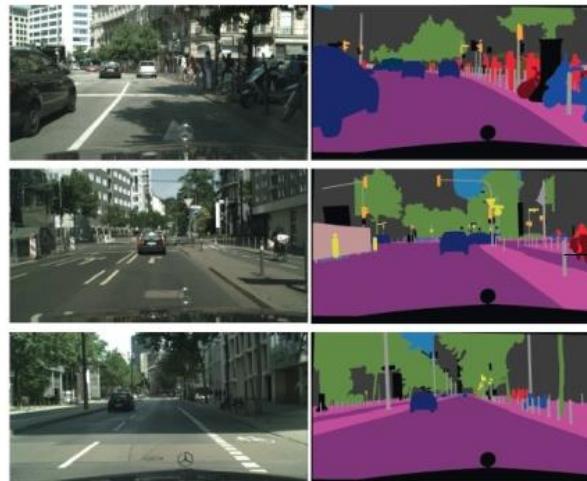
- Qu'apprennent les ANN? Peut-on interpréter le rôle des couches?



- Visualisation des filtres
- Calcul des images sensibles par optimisation
- <https://distill.pub/>
- Microscope: <https://microscope.openai.com/models>
- CNN explainer: <https://poloclub.github.io/cnn-explainer/>

# Les réseaux « denses »

- Image → Image, séquence → séquence
- Segmentation, Flot, Débruitage, traduction...
- A-t-on une version « deep learning » pour ces fonctions?... Oui!



# Le coût écologique du deep learning



Artificial Intelligence  
Index Report 2023

Chapter 2: Technical Performance  
2.8 Environment

## CO2 Equivalent Emissions (Tonnes) by Selected Machine Learning Models and Real Life Examples, 2022

Source: Luccioni et al., 2022; Strubell et al., 2019 | Chart: 2023 AI Index Report

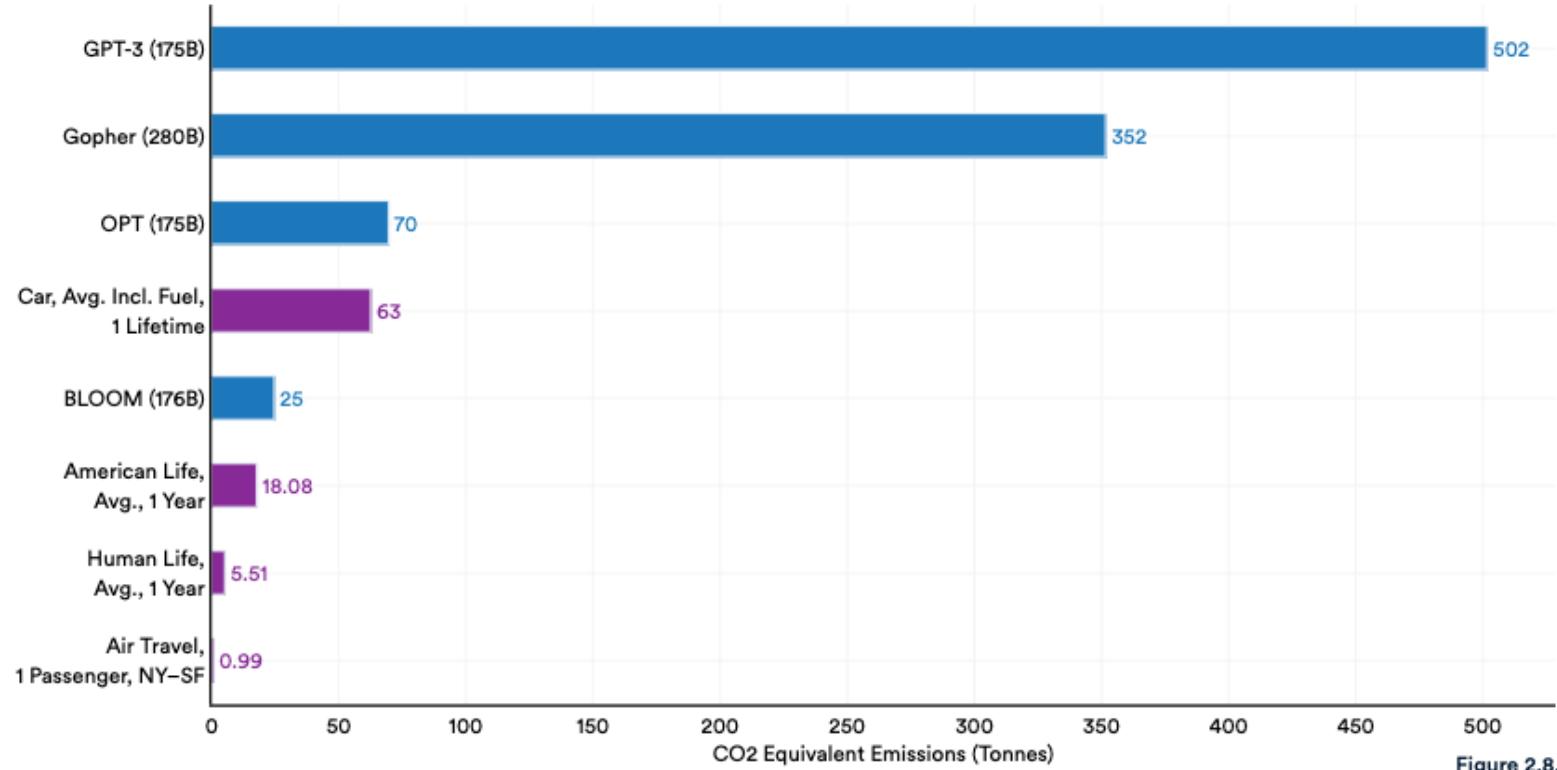


Figure 2.8.2

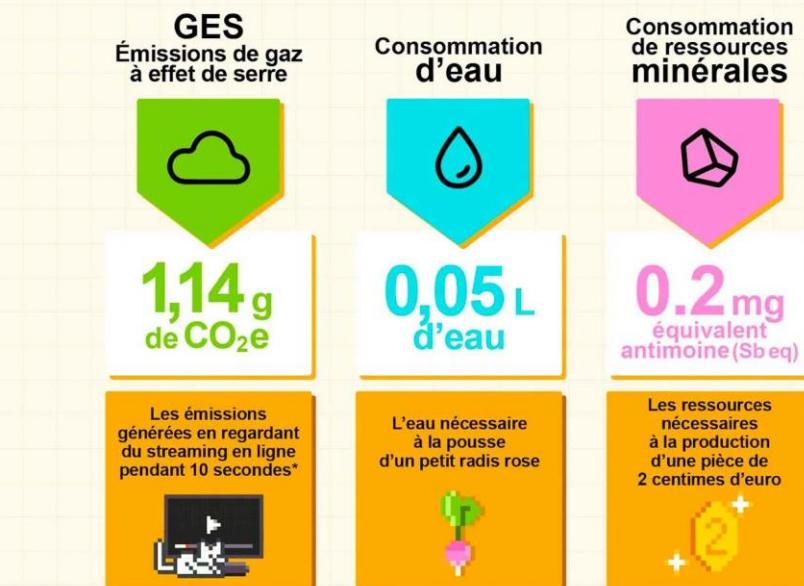
# Modèle Large 2 de Mistral AI

## Analyse du cycle de vie



## Modèle Large 2 de Mistral AI

Impact d'une page de texte générée (400 tokens)



<https://mistral.ai/fr/news/our-contribution-to-a-global-environmental-standard-for-ai>

# Deep Learning: Résumé

- Des réseaux de neurones organisés en couches (nombreuses) avec un apprentissage stochastique de descente de gradient.
- La phase d'optimisation peut être délicate.
- De nombreuses architectures avec des détails qui peuvent avoir un impact important sur les performances.
- Des environnements logiciels pour aider à la conception et à l'apprentissage (en TD).
- Pour certains types de données (image, texte, son) c'est l'approche contemporaine incontournable.