

Apprentissage automatique 1 (EMINF2C1)

Cours 2 : Réseaux convolutifs, CNN
M1 IAFA, IMA-RO

Contributeurs :
Farah Benamara
Ismail Khalfaoui Hassani
Thomas Pellegrini (resp.)
Contacts : prenom.nom@irit.fr



Qu'est-ce que les ordinateurs voient ?

Les images sont des nombres



Les images sont des nombres



187	183	174	168	160	152	129	161	172	163	155	156
155	182	163	74	75	62	99	17	110	210	180	354
180	180	50	14	34	6	10	33	48	106	159	181
206	109	6	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	106	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	218	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	163	158	227	178	143	182	106	56	190
205	174	186	282	236	231	149	176	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	234	147	106	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	238	75	1	81	47	0	6	217	256	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	129	207	177	121	123	200	179	13	96	218

Les images sont des nombres



167	163	174	168	150	162	129	161	172	161	156	156
166	182	163	74	75	62	39	17	110	210	180	154
180	180	50	14	34	6	10	93	46	106	159	181
206	109	6	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	237	87	71	201
172	106	207	233	233	214	220	239	228	98	74	206
188	68	179	209	185	215	211	158	139	75	20	169
189	97	166	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	179	143	182	106	36	190
205	174	156	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	137	103	36	101	258	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	236	76	1	81	47	0	6	217	258	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	19	96	218

197	163	174	168	150	162	129	151	172	141	195	166
195	182	163	74	75	62	39	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	6	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	237	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	68	179	209	185	215	211	158	139	75	20	169
189	97	166	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	179	143	182	106	36	190
205	174	156	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	137	103	36	101	255	234
190	214	173	66	103	143	96	50	2	109	249	215
187	196	236	76	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Ici, par exemple, l'image est une matrice de taille 16×12 avec des valeurs entre 0 et 255

Exemple de tâche en *Computer Vision*



187	183	174	168	160	152	138	131	123	161	180	194
185	182	163	74	76	62	38	37	31	210	180	154
180	180	50	14	34	6	10	33	40	126	188	181
206	199	6	124	131	111	120	204	196	76	94	180
194	66	137	203	237	236	236	236	237	87	71	201
172	105	207	230	230	214	221	230	230	98	74	206
188	89	179	204	185	216	271	188	188	76	20	169
189	97	165	84	10	168	134	11	31	62	22	148
168	168	191	191	198	227	178	143	182	126	36	191
205	174	195	202	206	231	140	179	228	43	94	234
182	216	116	140	236	187	86	182	79	38	210	241
180	204	147	108	227	210	127	122	36	101	206	224
180	214	173	66	109	149	96	99	2	108	249	216
187	186	236	76	3	41	47	0	4	217	206	211
182	202	237	145	0	0	12	108	200	136	243	236
196	206	129	87	177	121	129	205	175	12	94	218

classification

Lincoln
Washington
Jefferson
Obama

$\begin{bmatrix} 0.8 \\ 0.1 \\ 0.05 \\ 0.05 \end{bmatrix}$

Computer Vision : détecter des objets et concepts

Identifions des concepts clés à détecter dans les images suivantes



nez,
yeux,
bouche



roues,
phares,
plaque



portes,
fenêtres,
escalier

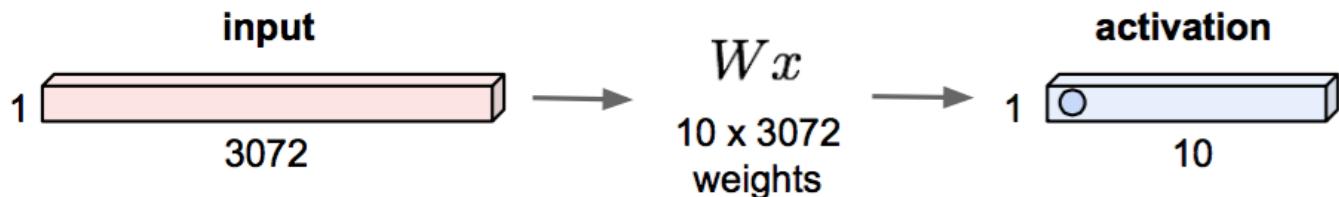
Comment faire automatiquement ?

Premières approches possibles

- Approche 1 : extraire des features pertinents et entraîner un classifieur (MLP, SVM, régression logistique, etc.)
- Approche 2 : utiliser un MLP sur les pixels vectorisés
- Approche 3 : utiliser un CNN...

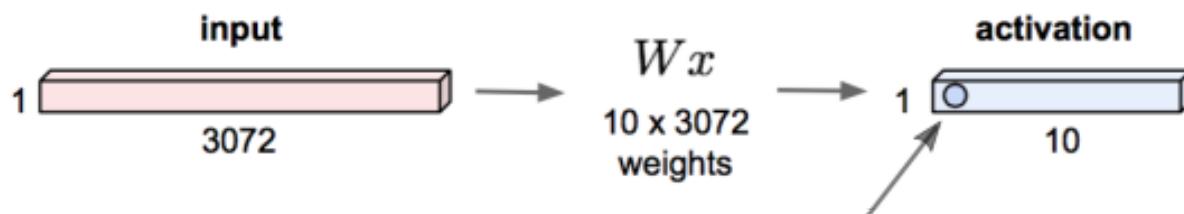
Une couche connectée

32x32x3 image RGB → vecteur 3072x1



Une couche connectée

32x32x3 image RGB → vecteur 3072x1



1 nombre :
le résultat du produit
scalaire d'une ligne de W
et du vecteur d'entrée
(prod. scalaire en dimension 3072)

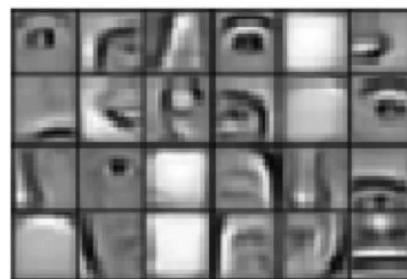
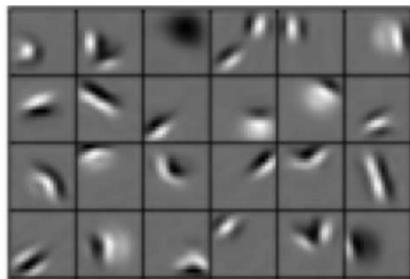
Une couche connectée : problèmes ?



- Changement d'échelle
- Translation
- Autres : rotation, déformations, variation de vue, de lumière, variation intra-classe, etc.

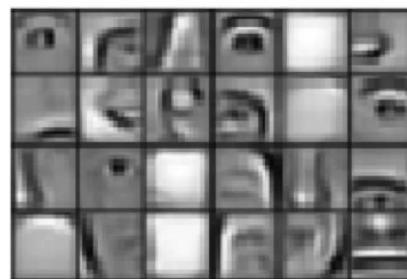
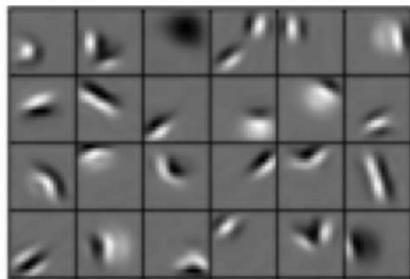
Nous voulons un modèle qui apprend :

- Ses propres représentations à partir des données brutes, *Representation Learning*,
- Des représentations "hiérarchiques" robustes aux problèmes cités précédemment



Nous voulons un modèle qui apprend :

- Ses propres représentations à partir des données brutes, *Representation Learning*,
- Des représentations "hiérarchiques" robustes aux problèmes cités précédemment



Une solution :

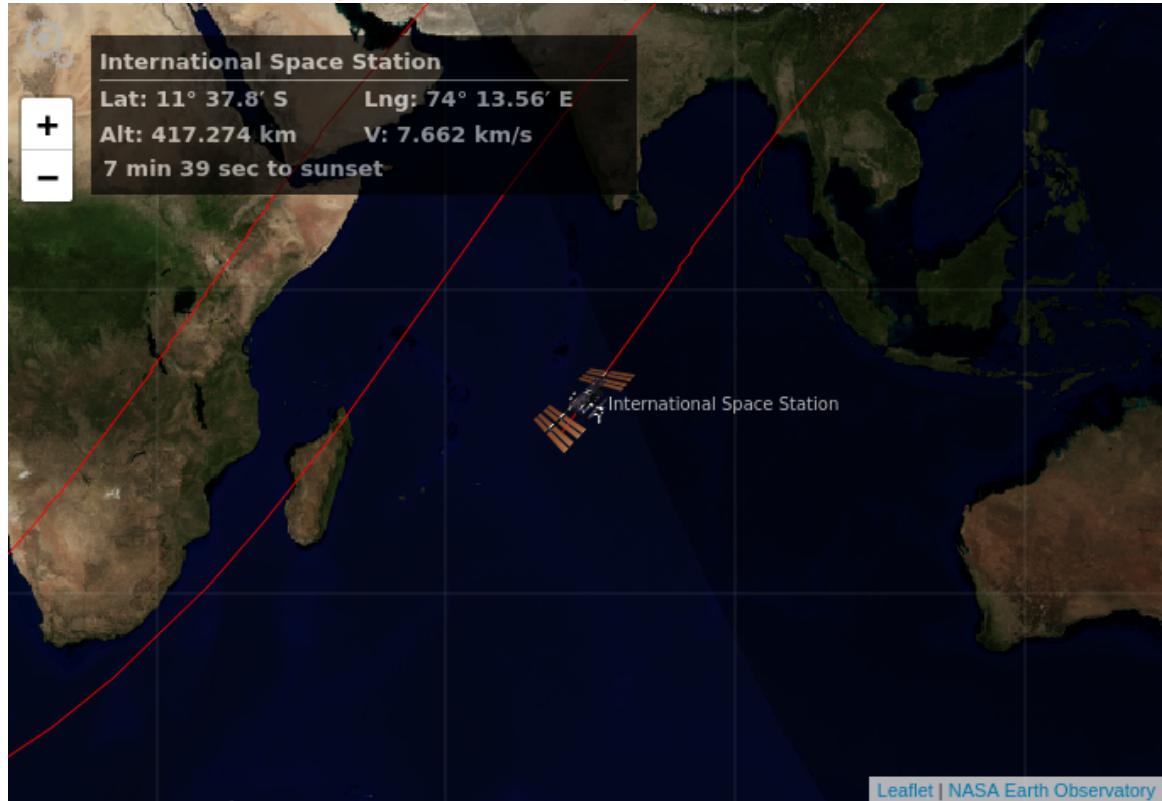
la couche de convolution !
(Plus deux ou trois autres trucs...)

Sommaire cours 2

- Motivation, intro opération de convolution
- Description d'une couche de convolution
 - Stride et padding
 - Dimensions en sortie, nombre de poids
 - Autres types de convolution : séparable, point-wise (1×1)
- Aperçu général d'un CNN : AlexNet, VGG16
- Couche de pooling
- Convolutions à trous

Intro convolution

Exemple : suivre la position de l'ISS



<http://www.lizard-tail.com/isana/tracking/>

Intro convolution

- Suivre la position de l'ISS : $x(t)$ avec x et t des variables réelles
- Le capteur donne une mesure bruitée → on veut faire une moyenne pondérée de plusieurs mesures proches dans le temps
- → Une fonction de pondération $w(a)$, où a est l'âge d'une mesure
- On obtient une estimation lissée s :

$$s(t) = \int x(a)w(t - a)da$$

Cette opération s'appelle une **convolution** et est notée * :

$$s(t) = (x * w)(t)$$

Terminologie des CNN

$$s(t) = (x * w)(t)$$

- x : l'**entrée** (série temporelle, image, texte, audio, etc)
- w : le **noyau** ou kernel, adapté (estimé) par l'algorithme d'apprentissage
- s : le résultat est appelé **carte de caractéristiques** ou feature map

Intro convolution

- Sur un ordinateur, les entrées sont échantillonnées (c'est du numérique, pas de l'analogique...)
- Dans notre exemple, le temps est donc discrétisé et la convolution s'écrit :

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

- Les valeurs de x et w sont stockées sur la machine sous forme de tableaux appelés des **tenseurs**
- On suppose que leurs valeurs sont nulles sauf dans l'ensemble de points stockés → en pratique, la somme infinie est mise en oeuvre sur un nombre fini d'éléments

Intro convolution

- la convolution est généralisable à la 2-d (images), 3-d (vidéos), etc.

$$S(i, j) = (I * K)(i, j) = \sum_{m,n} I(m, n)K(i - m, j - n)$$

- elle est commutative

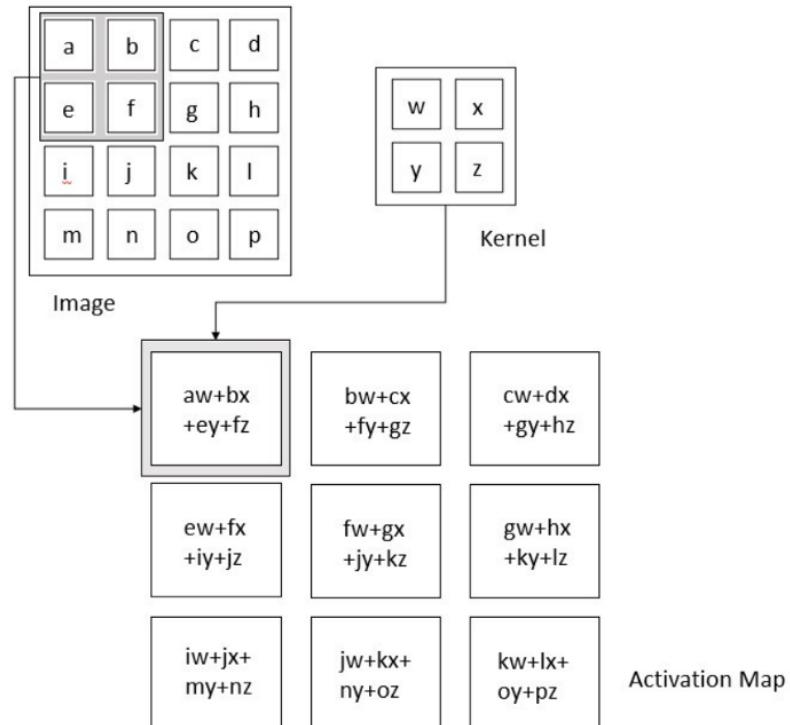
$$S(i, j) = (K * I)(i, j) = \sum_{m,n} I(i - m, j - n)K(m, n)$$

- cette propriété n'est en général pas pertinente dans les NN
- la convolution est plutôt implémentée comme une **corrélation croisée**

$$S(i, j) = (K * I)(i, j) = \sum_{m,n} I(i + m, j + n)K(m, n)$$

Intro convolution

$$S(i, j) = (K * I)(i, j) = \sum_{m,n} I(i + m, j + n)K(m, n)$$



I. Goodfellow, Y. Bengio, A. Courville. L'apprentissage profond, Quantmetry

20/78

Intro convolution

1	0	1
0	1	0
1	0	1

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

source : <http://ufldl.stanford.edu/>

21/78

Intro convolution

1 <small>$\times 1$</small>	1 <small>$\times 0$</small>	1 <small>$\times 1$</small>	0	0
0 <small>$\times 0$</small>	1 <small>$\times 1$</small>	1 <small>$\times 0$</small>	1	0
0 <small>$\times 1$</small>	0 <small>$\times 0$</small>	1 <small>$\times 1$</small>	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved Feature

source : <http://ufldl.stanford.edu/>

Exemple : détection de bords verticaux

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

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

=

Slide de Andrew Ng

Exemple : détection de bords verticaux

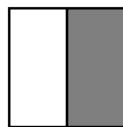
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

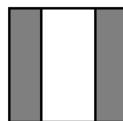
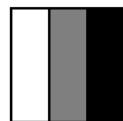
1	0	-1
1	0	-1
1	0	-1

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0



*



Slide de Andrew Ng

Exemple : détection de bords verticaux

10	10	10	0	0	0	0
10	10	10	0	0	0	0
10	10	10	0	0	0	0
10	10	10	0	0	0	0
10	10	10	0	0	0	0
10	10	10	0	0	0	0
10	10	10	0	0	0	0

*

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

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

0	0	0	10	10	10	10
0	0	0	10	10	10	10
0	0	0	10	10	10	10
0	0	0	10	10	10	10
0	0	0	10	10	10	10
0	0	0	10	10	10	10
0	0	0	10	10	10	10

*

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

=

0	-30	-30	0
0	-30	-30	0
0	-30	-30	0
0	-30	-30	0

Slide de Andrew Ng

Exemple : détection de bords

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

Vertical

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

*

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

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

Horizontal

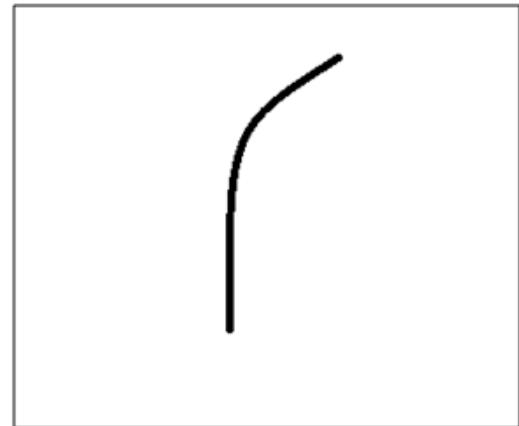
0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

Slide de Andrew Ng

Exemple : détection de bords

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

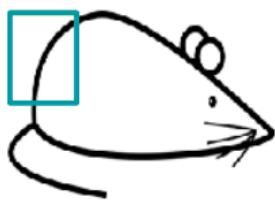
Pixel representation of filter



Visualization of a curve detector filter

source : <https://adeshpande3.github.io/>

Exemple : détection de bords



Visualization of the receptive field

0	0	0	0	0	0	30	
0	0	0	0	50	50	50	
0	0	0	20	50	0	0	
0	0	0	50	50	0	0	
0	0	0	50	50	0	0	
0	0	0	50	50	0	0	
0	0	0	50	50	0	0	
0	0	0	50	50	0	0	

Pixel representation of the receptive field

*

0	0	0	0	0	30	0	
0	0	0	0	30	0	0	
0	0	0	30	0	0	0	
0	0	0	30	0	0	0	
0	0	0	30	0	0	0	
0	0	0	30	0	0	0	
0	0	0	30	0	0	0	
0	0	0	0	0	0	0	

Pixel representation of filter

Multiplication and Summation = $(50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600$ (A large number!)



Visualization of the filter on the image

0	0	0	0	0	0	0	
0	40	0	0	0	0	0	
40	0	40	0	0	0	0	
40	20	0	0	0	0	0	
0	50	0	0	0	0	0	
0	0	50	0	0	0	0	
25	25	0	50	0	0	0	
0	0	0	0	0	0	0	

Pixel representation of receptive field

*

0	0	0	0	0	30	0	
0	0	0	0	30	0	0	
0	0	0	30	0	0	0	
0	0	0	30	0	0	0	
0	0	0	30	0	0	0	
0	0	0	30	0	0	0	
0	0	0	0	0	0	0	
0	0	0	0	0	0	0	

Pixel representation of filter

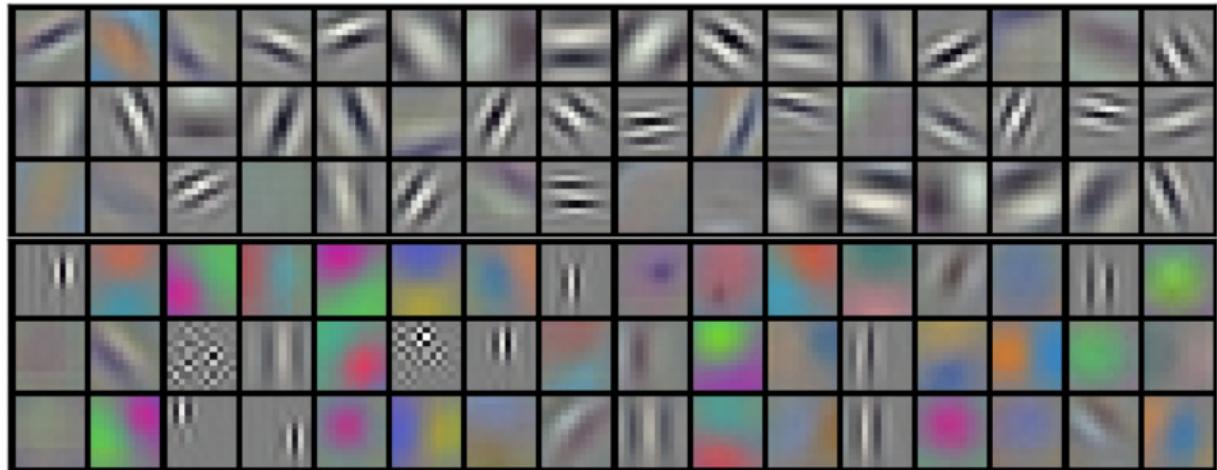
Multiplication and Summation = 0

Et si les poids des filtres étaient "appris" plutôt que choisis manuellement ?

w_{00}	w_{01}	w_{02}
w_{10}	w_{11}	w_{12}
w_{20}	w_{21}	w_{22}

⇒ Réseaux de neurones convolutifs ou CNN

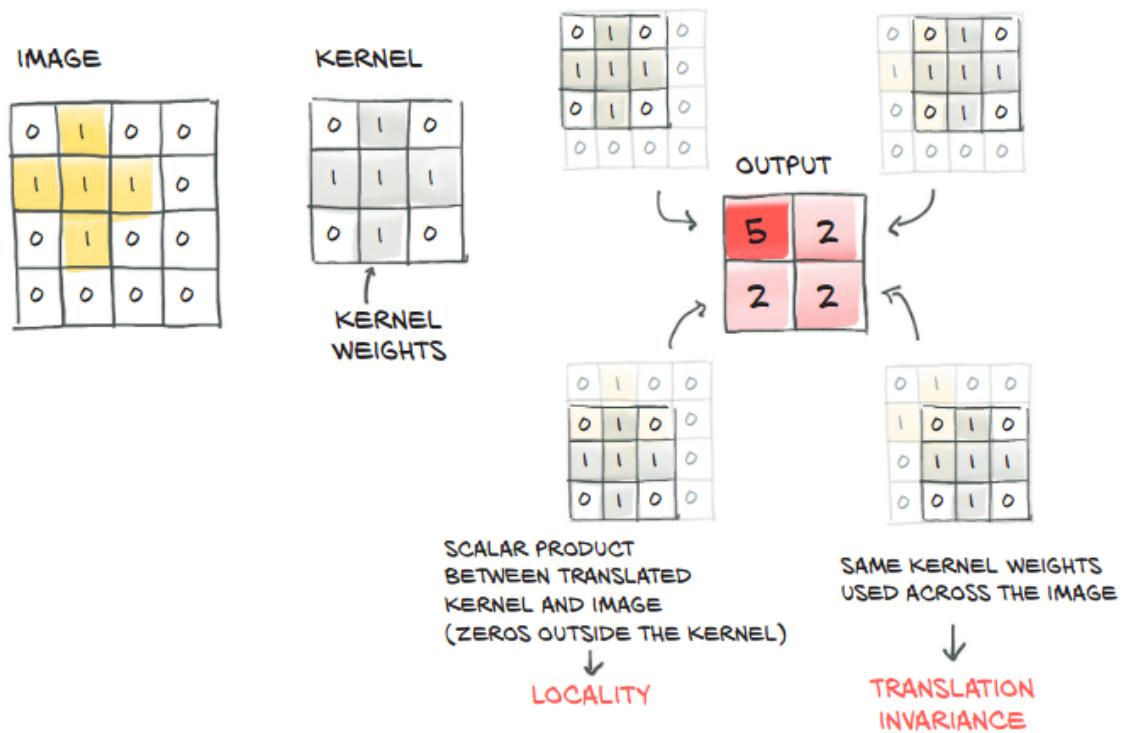
Exemple de filtres après apprentissage



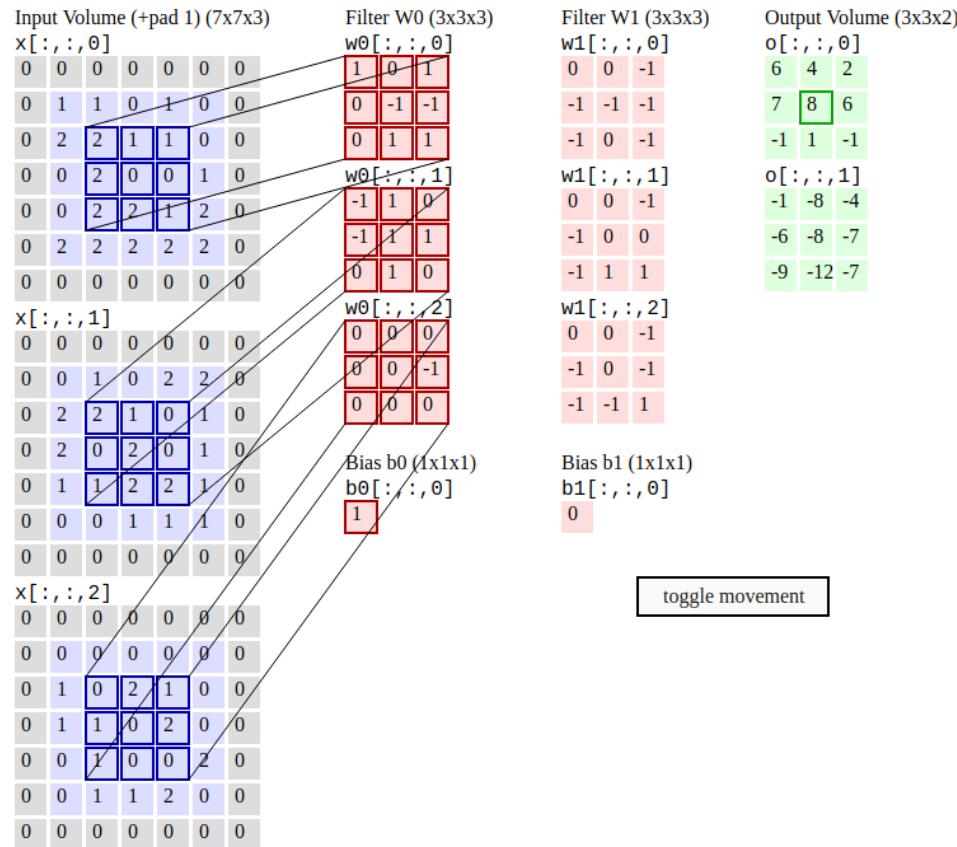
96 filtres de taille $11 \times 11 \times 3$ de la première couche d'un
CNN entraîné sur ImageNet

source : <http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf>

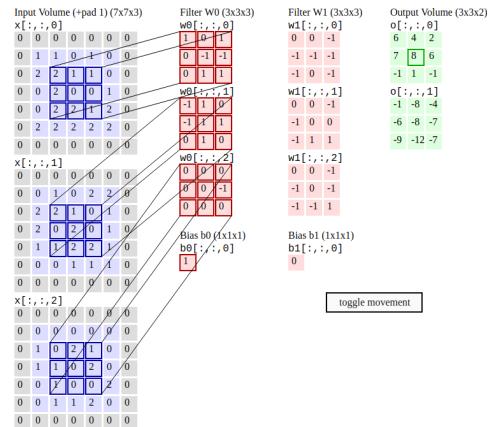
couche de convolution : connectivité locale, partage des poids, invariance par translation



- La couche de convolution standard est **multicanal** : chaque filtre a autant de matrices que l'entrée a de "canaux"



- La convolution standard dans un CNN est **multicanal** : chaque filtre a autant de matrices que l'entrée a de "canaux"
- Une feature map résultat est la somme des convolutions avec ces matrices
- C'est plus du tout commutatif !



source : <http://cs231n.github.io/convolutional-networks/>

Conv2D en Pytorch

```
▶ import torch
    import torch.nn as nn

    m = nn.Conv2d(8, 16, 3)

    x = torch.randn(2, 8, 50, 100)

    out = m(x)

    out.size()

⇨ torch.Size([2, 16, 48, 98])
```

<https://pytorch.org/docs/1.9.1/generated/torch.nn.Conv2d.html>

Question : quelle est la size du kernel m ?

Conv2D en Pytorch

```
import torch
import torch.nn as nn

m = nn.Conv2d(8, 16, 3)

x = torch.randn(2, 8, 50, 100)

out = m(x)

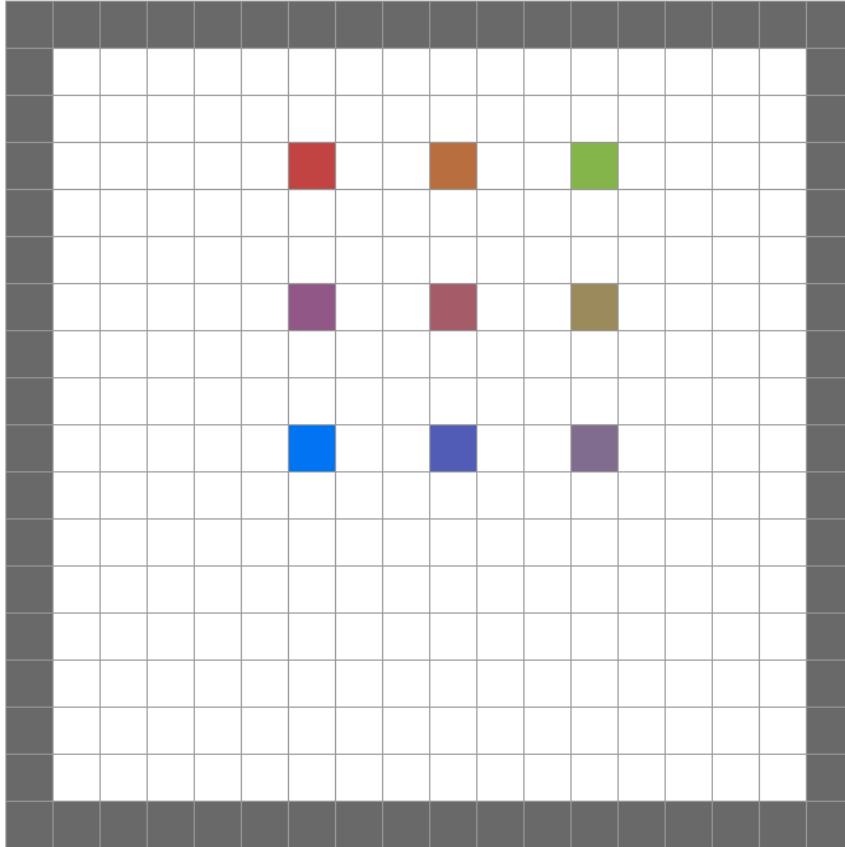
print("Taille de la sortie : ", out.size())
print("Taille des tenseurs de m :")
print(" Poids : ", m.weight.shape)
print(" Biais : ", m.bias.shape)

print("Nb total de paramètres : ", sum(p.numel() for p in m.parameters()))

Taille de la sortie : torch.Size([2, 16, 48, 98])
Taille des tenseurs de m :
 Poids : torch.Size([16, 8, 3, 3])
 Biais : torch.Size([16])
Nb total de paramètres :  1168
```

Visualisation de la Conv2D

Input (16 × 16):



Implémentation sous-jacente de PyTorch

Multiplication matrice-matrice denses (GEMM)

- GEMM (General Matrix Multiplication) effectue l'opération matricielle suivante :

$$C := \alpha AB + \beta C$$

avec A , B et C , des matrices **denses** de tailles respectives $M.K$, $K.N$ et $M.N$, α et β des scalaires.

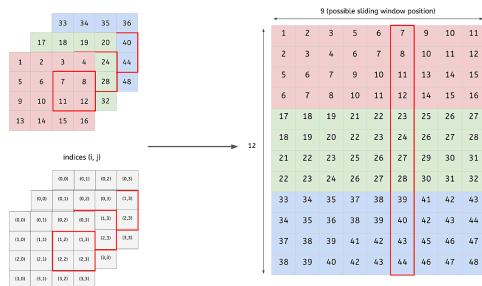
GEMM Intensité arithmétique

- I. ar. = Intensité arithmétique = $\frac{\text{nombre de FLOPS}}{\text{nombre de bytes accédés}} = \frac{2 \cdot (M \cdot N \cdot K)}{2 \cdot (M \cdot K + N \cdot K + M \cdot N)} = \frac{M \cdot N \cdot K}{M \cdot K + N \cdot K + M \cdot N}$
- GEMM a souvent une I. ar. élevée ($>> 1$) : *math-bound*.
- GEMV (produit matrice-vecteur) par exemple est toujours *memory-bound* car dans ce cas $M = N = 1$ d'où I. ar. $< \frac{1}{2}$.

Implémentation sous-jacente de PyTorch

L'algorithme Im2Col

- Transforme un batch d'images 2D (avec leurs channels) en une matrice selon une disposition qui correspond à la convolution souhaitée.
- Certains éléments peuvent être dupliqués selon le stride utilisé.
- Le module `torch.nn.Unfold` effectue l'opération Im2Col.



<https://raw.githubusercontent.com/valoxe/image-storage-1/master/blog-deep-learning/cnnumpy-fast/1.gif>

Implémentation sous-jacente de PyTorch

Forward

- ➊ On applique Im2Col sur le tenseur d'entrée.
 $(B, Ch_{in}, H_{in}, W_{in}) \rightarrow (Ch_{in} * K_h * K_w, B * H_{out} * W_{out})$.
- ➋ H_{out} et W_{out} sont déterminées par :

$$H_{\text{out}} = \left\lfloor \frac{H_{\text{in}} + 2 \times \text{pad}_h - \text{dil}_h \times (K_h - 1) - 1}{\text{stride}_h} + 1 \right\rfloor$$

et

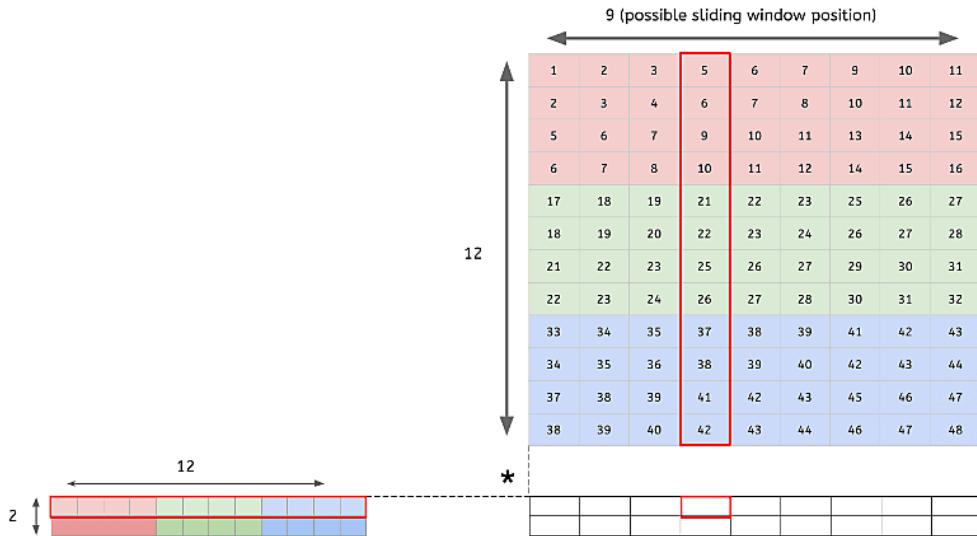
$$W_{\text{out}} = \left\lfloor \frac{W_{\text{in}} + 2 \times \text{pad}_w - \text{dil}_w \times (K_w - 1) - 1}{\text{stride}_w} + 1 \right\rfloor$$

- ➋ On reshape ensuite les **poids** pour obtenir une matrice compatible avec la sortie d'Im2Col (flatten operation).
 $(Ch_{out}, Ch_{in}, K_h, K_w) \rightarrow (Ch_{out}, Ch_{in} * K_h * K_w)$.
- ➌ On multiplie ces deux dernières matrices en utilisant cuBlas GEMM.

Backward

- ① On applique Im2Col sur le tenseur d'entrée.
 $(B, Ch_{in}, H_{in}, W_{in}) \rightarrow (Ch_{in} * K_h * K_w, B * H_{out} * W_{out})$.
- ② On reshape ensuite le **gradient de la sortie** pour obtenir une matrice compatible avec la sortie d'Im2Col.
 $(B, Ch_{out}, H_{out}, W_{out}) \rightarrow (B * H_{out} * W_{out}, Ch_{out})$.
- ③ On multiplie ces deux dernières matrices en utilisant cuBlas GEMM.
- ④ Le résultat est une matrice de taille $(Ch_{in} * K_h * K_w, Ch_{out})$ qu'on peut dès lors ressaper en un tenseur de dimension $(Ch_{out}, Ch_{in}, K_h, K_w)$.

Implémentation sous-jacente de PyTorch



<https://raw.githubusercontent.com/valoixe/image-storage-1/master/blog-deep-learning/cnnumpy-fast/11.gif>

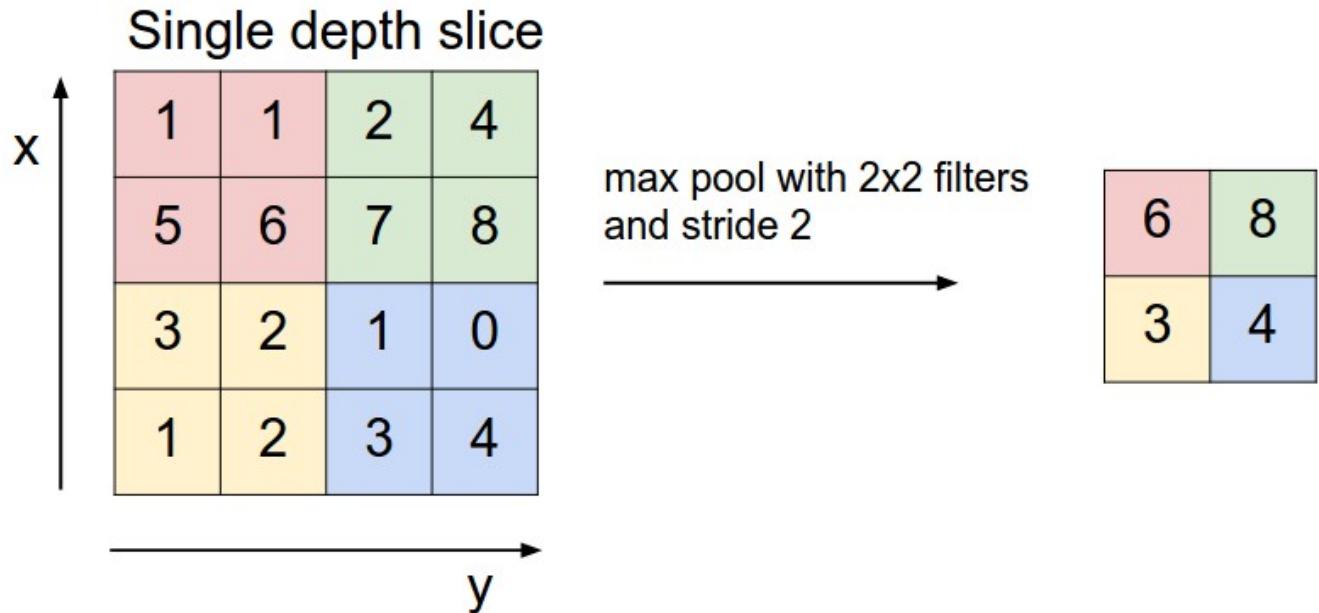
Optimisations éventuelles de l'algorithme de convolution

- L'algorithme de Winograd permet de réduire le nombre de FLOPs de la convolution 2D - Im2Col par un facteur de $\frac{9}{4} = 2.25$ et ce en remplaçant le produit matriciel GEMM par un produit plus adapté qui tient compte des éléments dupliqués par Im2Col.
- Lorsque les filtres de poids sont de très grande dimension (typiquement $K_h = K_w > 15$) il devient plus rentable (en temps de calcul) d'effectuer le produit de convolution dans le domaine des fréquences (FFT).

Résumé couche de convolution

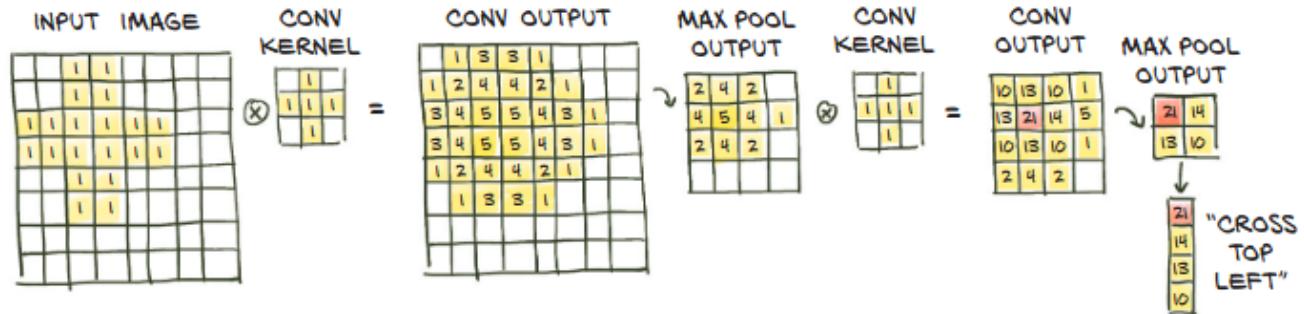
- Un filtre de convolution permet d'identifier la présence ou l'absence d'un type de caractéristique
- Chaque filtre répond à un type de caractéristique : il faut plusieurs filtres
- Le partage des poids (*weight sharing*) rend le filtre invariant à la position et réduit drastiquement le nombre de poids/paramètres du modèle, comparé à une couche connectée.

Couche de pooling (max-pooling)



- Réduire le nombre de paramètres
- Robustesse au bruit local

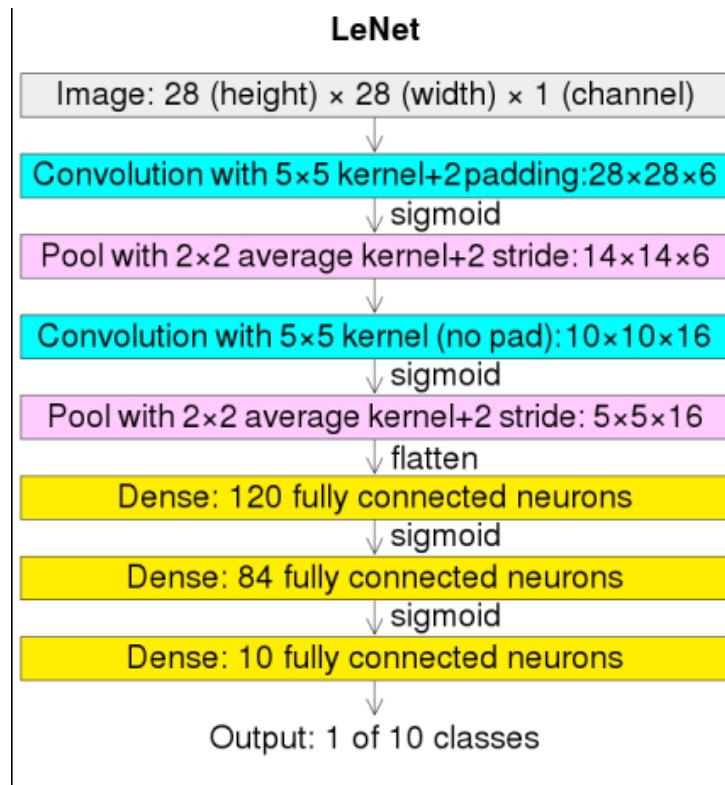
La notion de champ réceptif (*receptive field*)



Le nombre 21 issu de la deuxième convolution est influencé par une zone de pixels 8×8 dans l'image en entrée

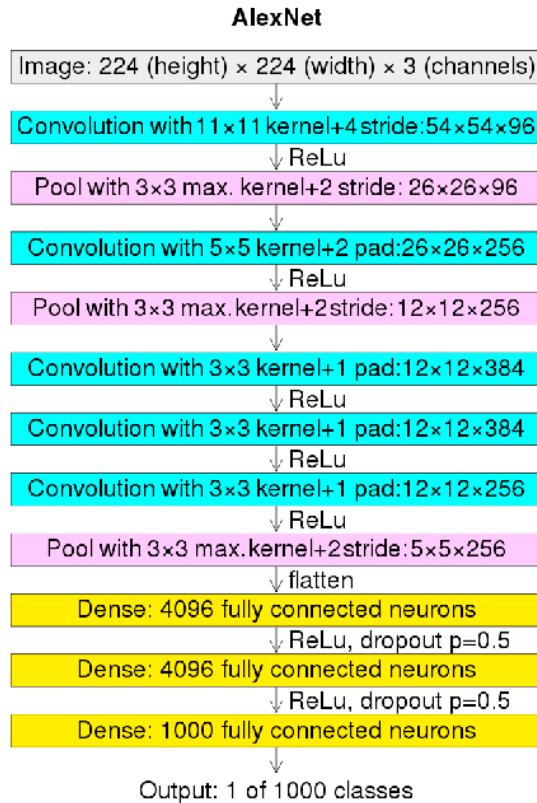
DLWPT

Réseau LeNet (LeNet-5) : 1989 !



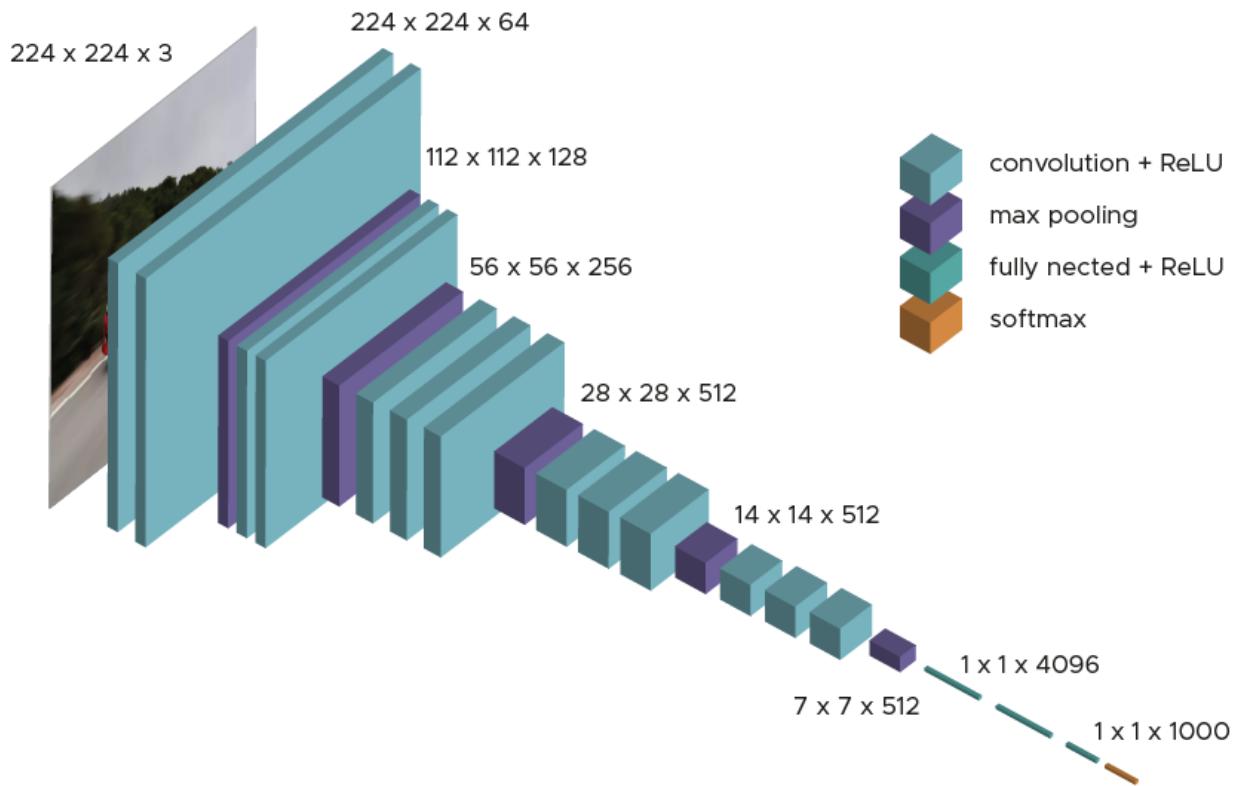
<https://en.wikipedia.org/wiki/LeNet>

Réseau AlexNet, 2012 (60 M params)



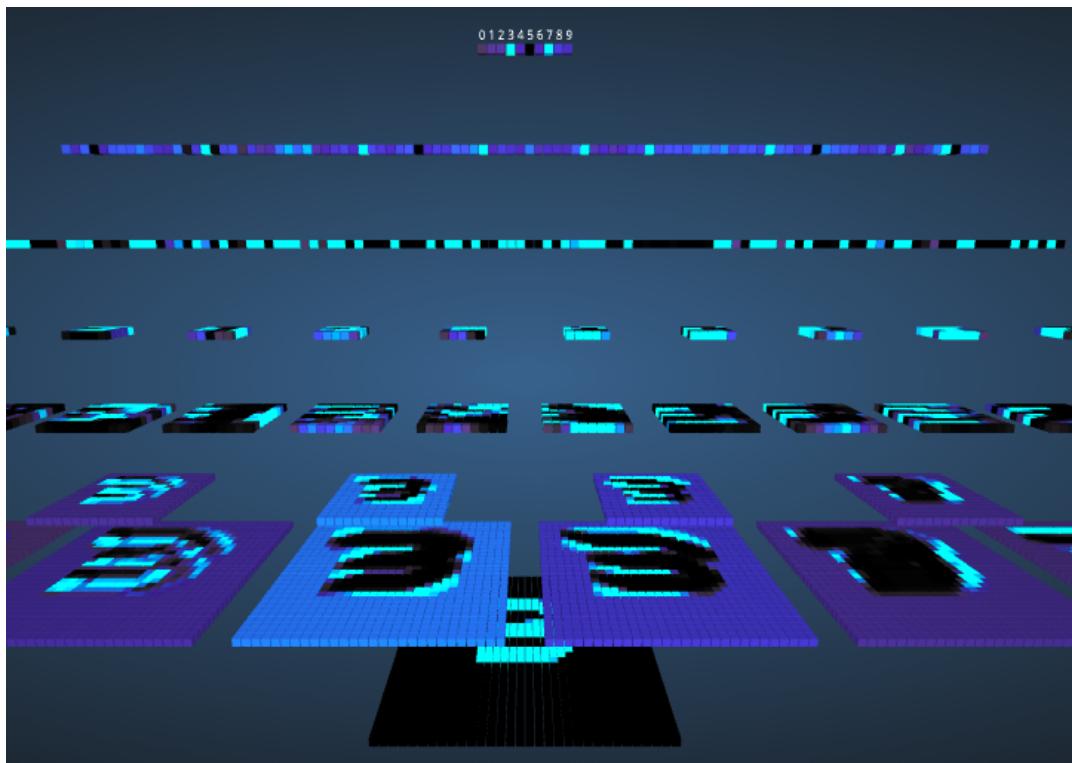
<https://en.wikipedia.org/wiki/AlexNet>

Réseau VGGNet (VGG16 et VGG19), 2014



<https://datascientest.com/quest-ce-que-le-modele-vgg>

Démo en ligne

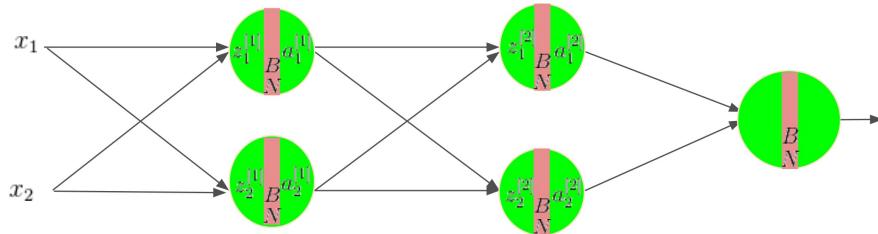


<https://cs.ryerson.ca/~aharley/vis/conv/flat.html>



<http://cs.stanford.edu/people/karpathy/convnetjs/>

Couche de batch-normalisation (2015)



$$\begin{aligned} z^{[l]} &= W^{[l]} a^{[l-1]} \longrightarrow & \mu^{[l]} &= \frac{1}{m} \sum_i z^{[l](i)} \\ && \sigma^{[l]2} &= \frac{1}{m} \sum_i (z^{[l](i)} - \mu^{[l]})^2 \\ && z_{norm}^{[l](i)} &= \frac{z^{[l](i)} - \mu^{[l]}}{\sqrt{\sigma^{[l]2} + \epsilon}} \\ && \tilde{z}^{[l](i)} &= \gamma^{[l]} z_{norm}^{[l](i)} + \beta^{[l]} \longrightarrow & a^{[l]} &= g^{[l]}(\tilde{z}^{[l]}) \end{aligned}$$

- But : normaliser les entrées à une couche supérieure
- Accélère l'apprentissage
- Possibilité d'utiliser des LR plus grands
- Tester si appliquer BN avant ou après la fonction d'activation

Couche de batch-normalisation

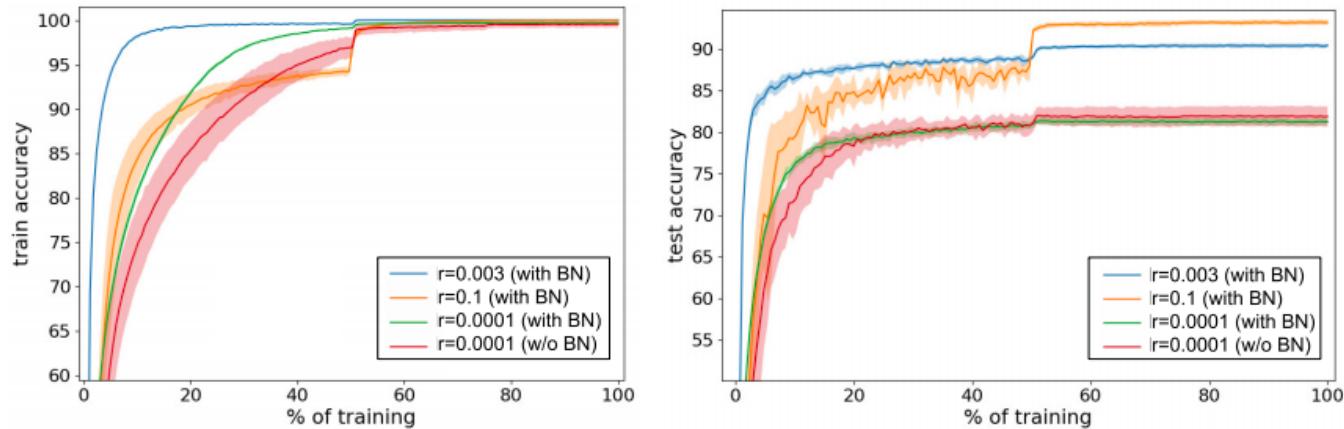
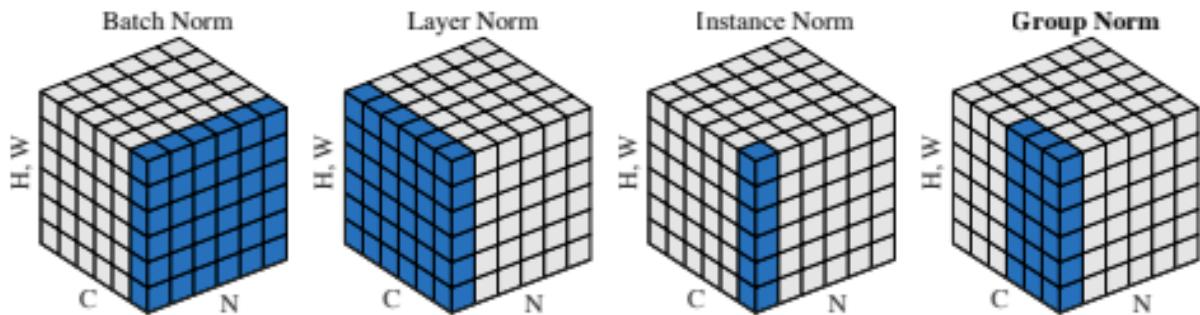


Figure 1: The training (*left*) and testing (*right*) accuracies as a function of progress through the training cycle. We used a 110-layer Resnet with three distinct learning rates 0.0001, 0.003, 0.1. The smallest, 0.0001 was picked such that the network without BN converges. The figure shows that with matching learning rates, both networks, with BN and without, result in comparable testing accuracies (red and green lines in right plot). In contrast, larger learning rates yield higher test accuracy for BN networks, and diverge for unnormalized networks (not shown). All results are averaged over five runs with std shown as shaded region around mean.

Understanding Batch Normalization, Johan Bjorck et al, NeurIPS 2018

Autres couches de normalisation



- BN : pas adapté aux petits batchs ni aux RNN
- Layer-norm : adapté aux RNN
- Et aussi : "Weight-normalization" layer souvent utilisé avec mean-only BN (meilleurs résultats sur CIFAR-10)

https:

//mlexplained.com/2018/11/30/an-overview-of-normalization-methods-in-deep-learning/

Réseaux purement convolutifs (fully convolutional, FCN)

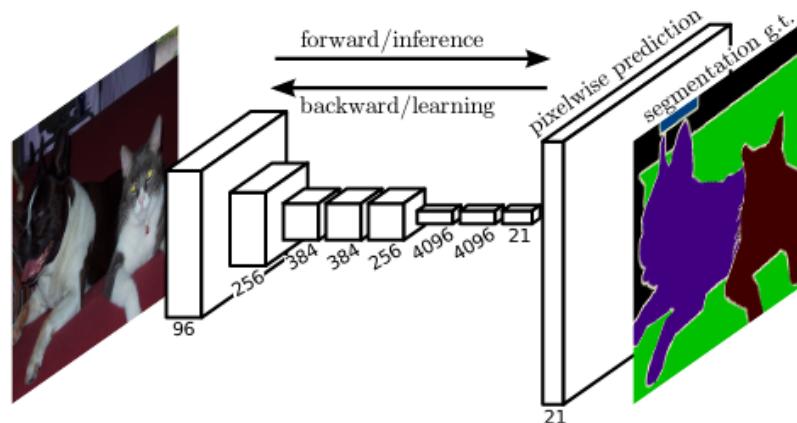


Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Long_Fully_Convolutional_Networks_2015_CVPR_paper.pdf

Réseaux purement convolutifs (fully convolutional, FCN)

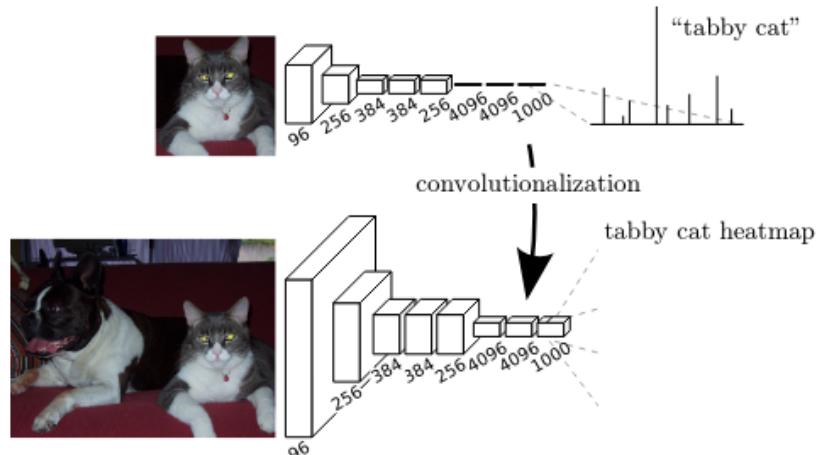


Figure 2. Transforming fully connected layers into convolution layers enables a classification net to output a heatmap. Adding layers and a spatial loss (as in Figure 1) produces an efficient machine for end-to-end dense learning.

https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Long_Fully_Convolutional_Networks_2015_CVPR_paper.pdf

Residual Networks (ResNets), 2015

- Comment rendre possible l'entraînement de réseaux de plusieurs centaines de couches ?
- Problème de l'évaporation des gradients ou *vanishing gradient*

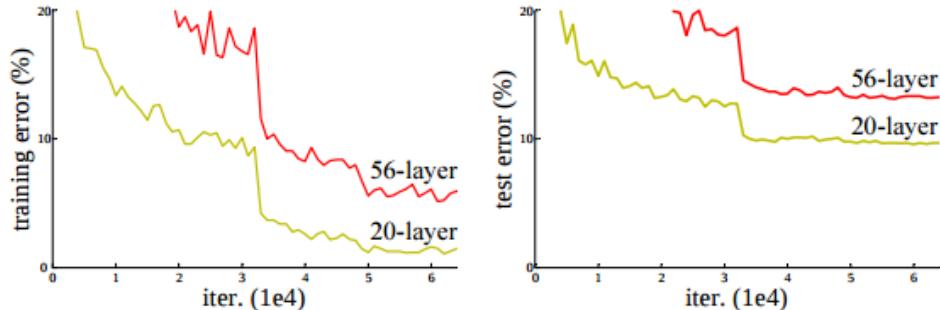
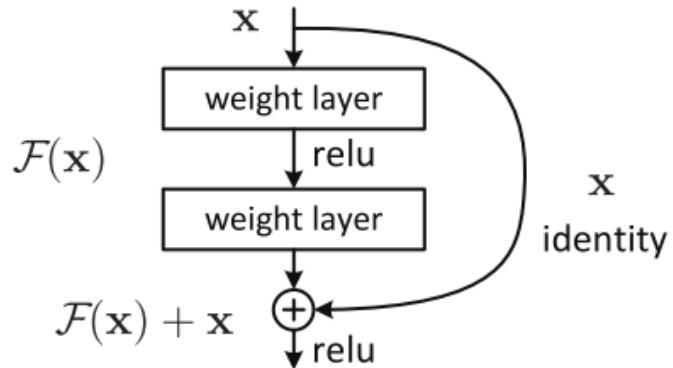


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

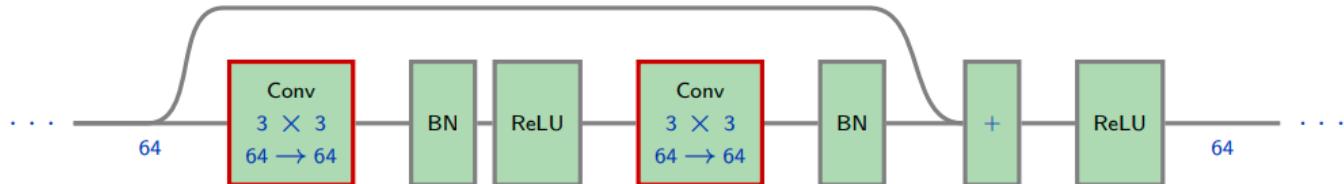
Residual Networks (ResNets)

- Comment rendre possible l'entraînement de réseaux de plusieurs centaines de couches ?
- Problème de l'évaporation des gradients ou *vanishing gradient*
- ResNet : ajouter des connexions résiduelles ou *skip connections*

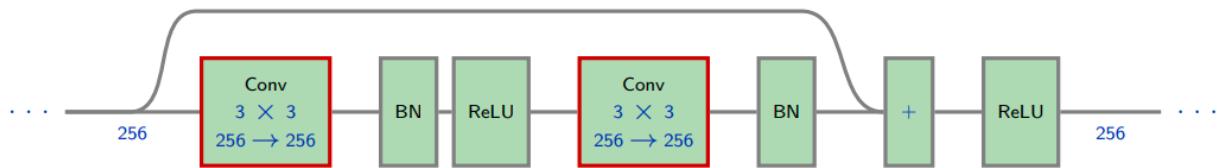


Residual Networks (ResNets)

- Block résiduel par défaut

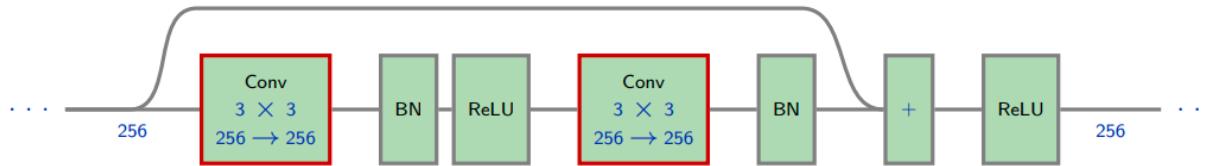


- Block résiduel par défaut pour ImageNet nécessite plus de channels

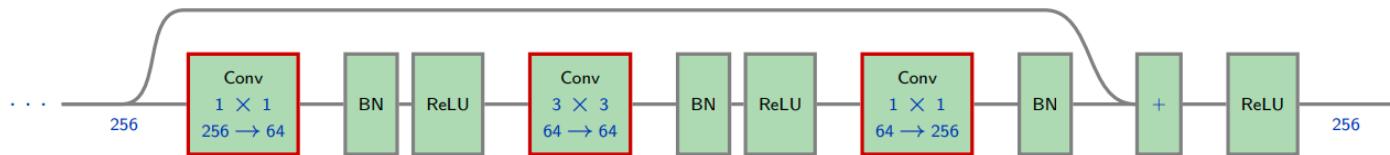


Residual Networks (ResNets)

- Block résiduel par défaut pour ImageNet nécessite plus de channels



- $2 \times (3 \times 3 \times 256 + 1) \times 256 = 1.2\text{m}$ paramètres
- C'est beaucoup → bloc résiduel dit *bottleneck*



- $256 \times 64 + (3 \times 3 \times 64 + 1) \times 64 + 64 \times 256 = 70\text{k}$ paramètres

Residual Networks (ResNets)

- Résultats sur ImageNet (2015)

method	top-5 err. (test)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	3.57

Table 5. Error rates (%) of **ensembles**. The top-5 error is on the test set of ImageNet and reported by the test server.

Autres types de convolution

- La convolution séparable ou *depthwise convolution*
- La convolution point-à-point ou *pointwise convolution*
- Une séquence de convolution séparable puis pointwise s'appelle une convolution séparable en profondeur ou *depthwise-separable*
- La convolution dilatée ou à-trous

Autres types de convolution

- La convolution séparable ou *depthwise convolution*

```
m = nn.Conv2d(8, 16, 3, groups=8)

x = torch.randn(2, 8, 50, 100)

out = m(x)

print("Taille de la sortie : ", out.size())
print("Taille des tenseurs de m :")
print(" Poids : ", m.weight.shape)
print(" Biais : ", m.bias.shape)

print("Nb total de paramètres : ", sum(p.numel() for p in m.parameters()))
```

```
Taille de la sortie : torch.Size([2, 16, 48, 98])
Taille des tenseurs de m :
 Poids : torch.Size([16, 1, 3, 3])
 Biais : torch.Size([16])
Nb total de paramètres : 160
```

Autres types de convolution

- La convolution pointwise
- Noyau 1×1 : un noyau qui itère sur chaque point
- Profondeur égale au nombre de canaux de l'image d'entrée
- Souvent utilisé conjointement avec les convolutions depthwise pour produire une classe efficace de convolutions connue sous le nom de convolutions séparables en profondeur ou *depthwise-separable*

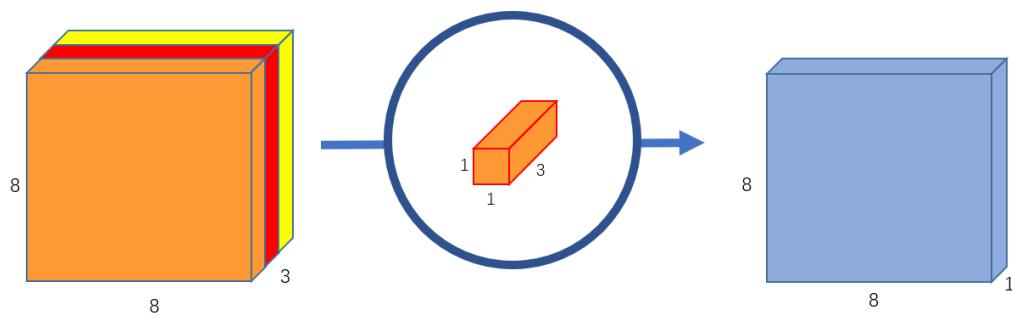
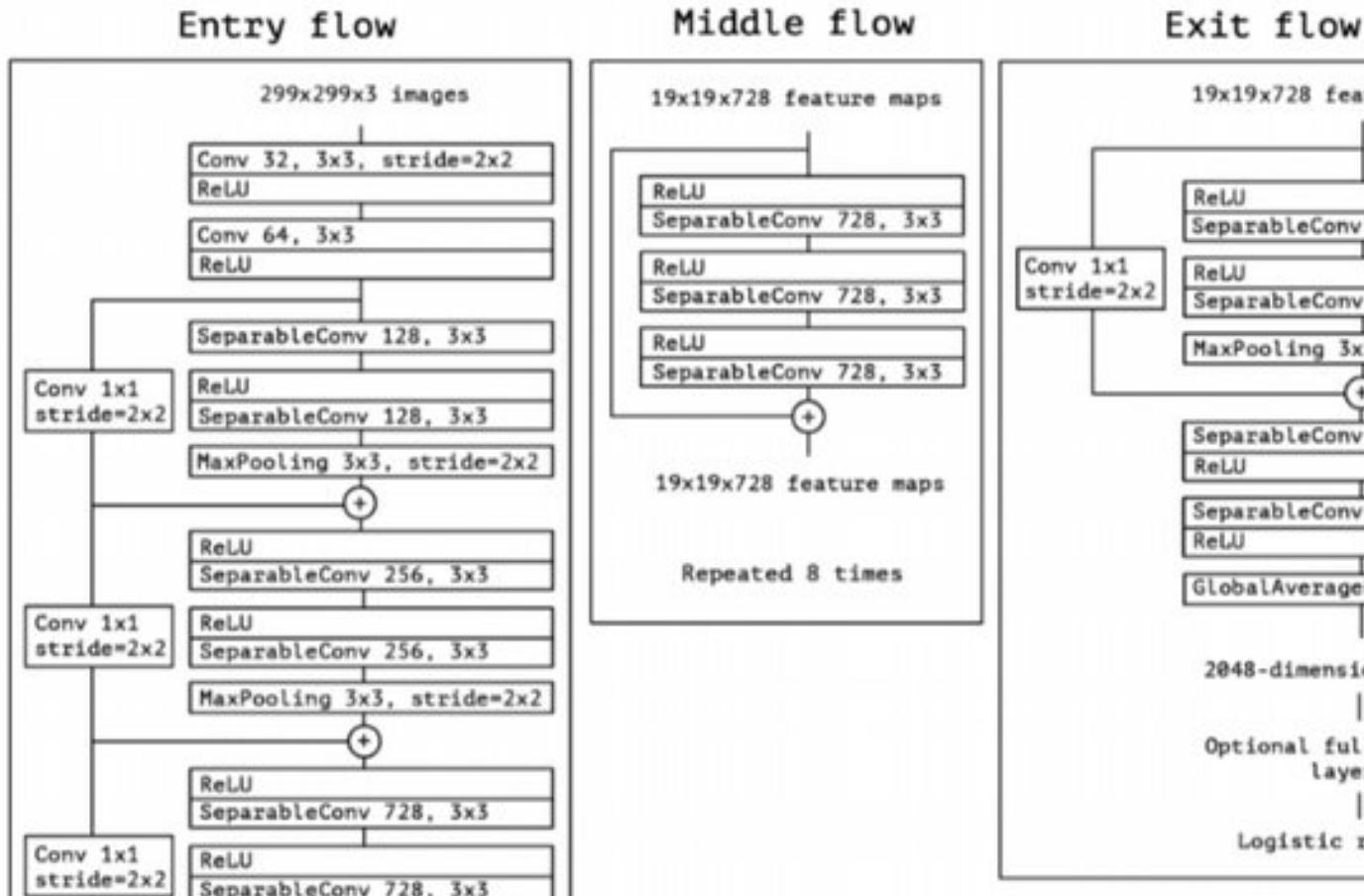


Image Credit : Chi-Feng Wang

63/78

L'architecture Xception



L'architecture Convmixer

Autres types de convolution

- La convolution dilatée

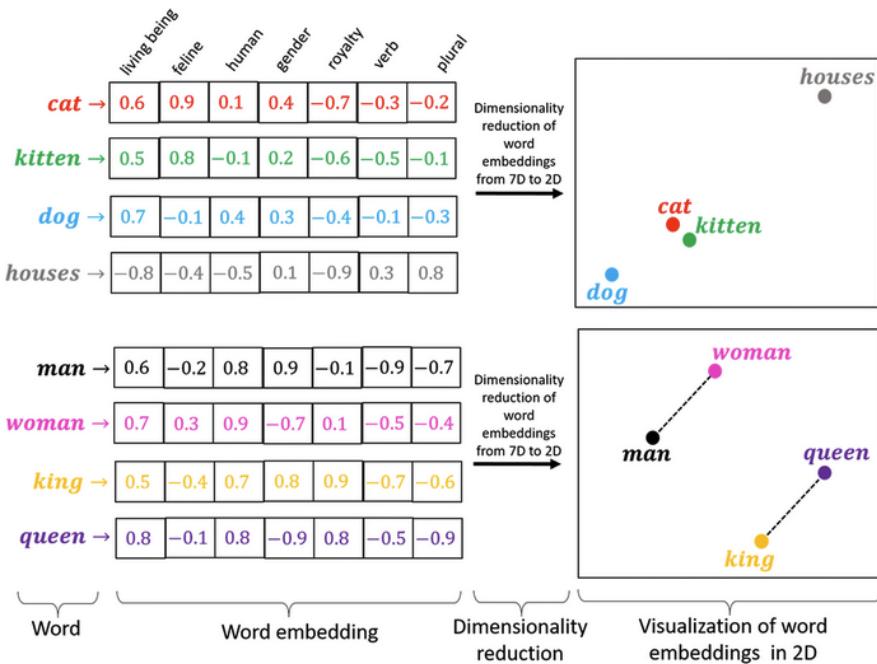
La convolution sur un signal sonore

- Convolution 1-d sur le signal brut
- Convolution 2-d sur des représentations temps-fréquence

La convolution sur des données textuelles

- Entrée d'un réseau :
 - une séquence de mots/sous-mots/caractères
 - chaque symbole est associé à une représentation numérique sous forme d'un vecteur à valeur réelle (par ex. dim 100)
 - Ces vecteurs sont appelés des "**plongements**" ou ***word embeddings***
- Sortie d'un réseau : classification de documents/phrases
 - Analyse de sentiments : positif/négatif/neutre
 - Filtrage de mails : spam/non-spam
 - Classification de topic : sport vs. politique vs. culture
- Rôle du CNN sur du texte : capturer l'information **des séquences de mots** (pas nécessairement grammaticalement corrects), puis la "résumer" pour effectuer une tâche donnée

Plongements de mots / word embeddings



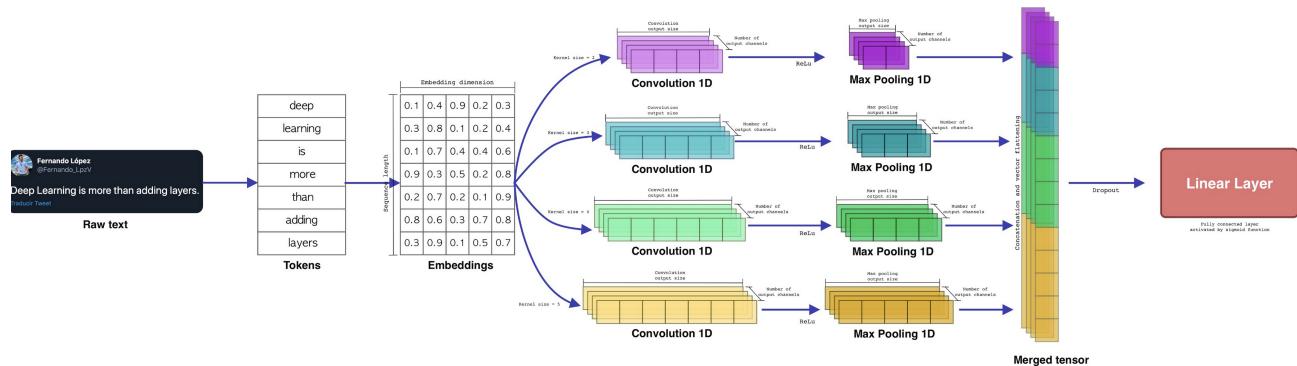
- Les valeurs de ces vecteurs peuvent être statiques ou mises à jour lors de l'apprentissage (fine-tuning)

Source : <https://medium.com/@hari4om/word-embedding-d816f643140>

Plongements de mots / word embeddings

- Plongements de mots ou *word embeddings*
- Générer à partir de modèles de langage pré-entraînés sur de gros corpus de données : Word2Vec, FastText, Glove, Bert, et bien d'autres
 - Construits par apprentissage auto-supervisé (self-supervision) : la supervision vient des données elles-mêmes, pas besoin d'annotation manuelle
 - Fondés sur la sémantique distributionnelle : deux mots qui apparaissent dans un même contexte sont susceptibles d'avoir des sens proches et donc des représentations vectorielles proches

La convolution sur des données textuelles



Source : Fernando López

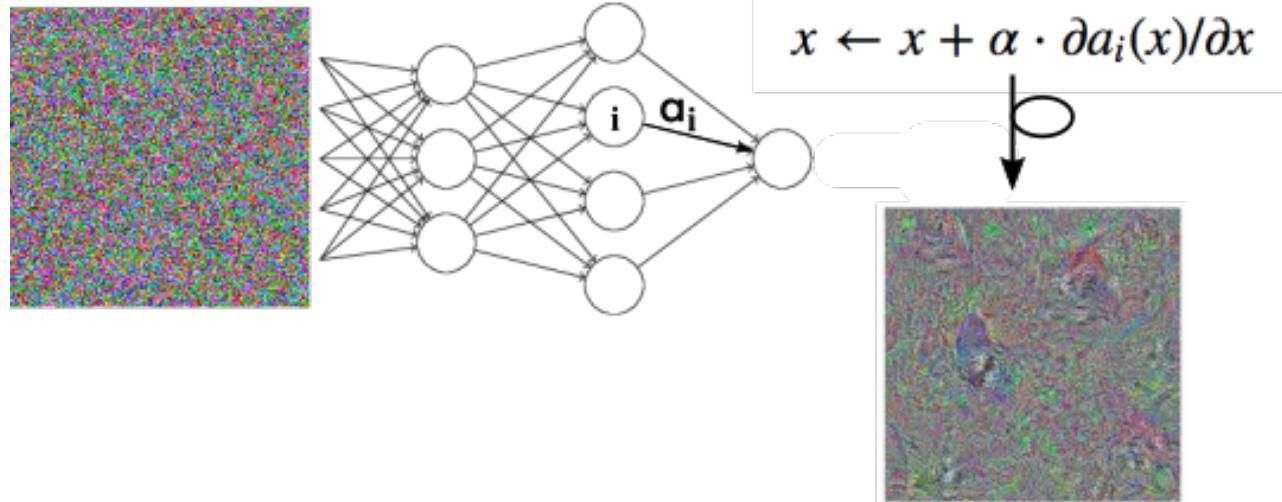
La convolution sur des données textuelles

- Des opérations de convolution 1-d s'appliquent sur des vecteurs provenant d'une couche "*embedding*"
 - Cette couche s'occupe de transformer une séquence de L tokens en entrée en une matrice $\mathbb{R}^{L \times d}$, d étant la dimension des embeddings
 - *embedding layer == lookup table* : à partir de l'indice entier d'un token, la couche renvoie le vecteur qui lui correspond
- Chaque couche de convolution est définie par des filtres (kernels/noyaux) de tailles différentes
- La dimension de chaque couche est réduite en sortie par un max-pooling
- Les vecteurs de l'ensemble des couches sont concaténés en sortie avant de les transmettre à la dernière couche (linéaire) responsable de la classification

Visualisation de cartes par maximisation des activations d'un neurone

Deepviz : Deep Visualization Toolbox (Jason Yosinski et al, ICML 2015)

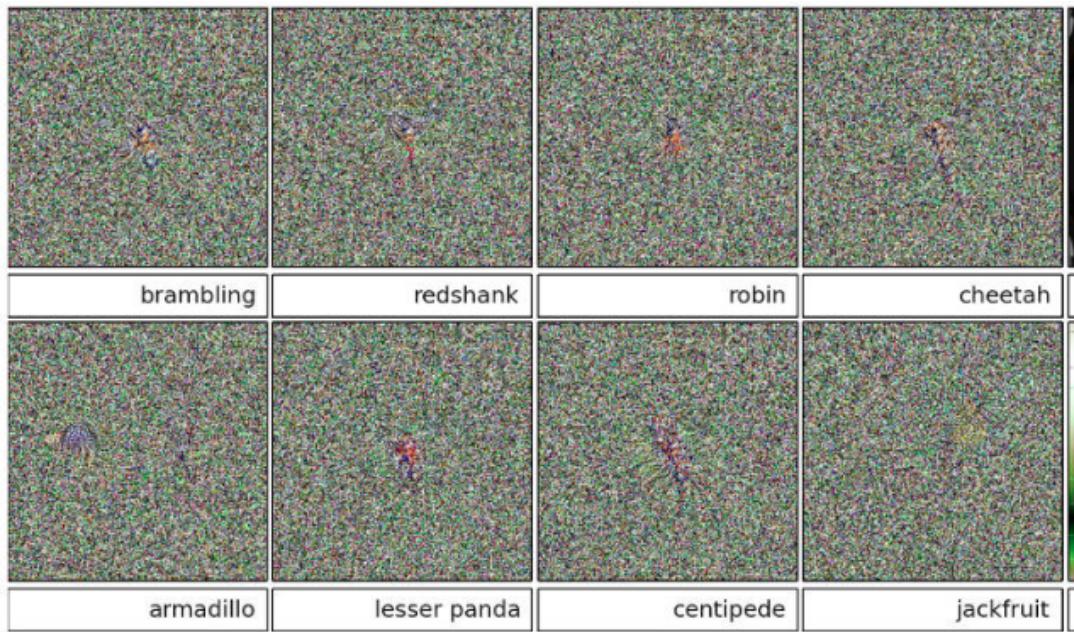
Principe de base pour un neurone i :



Vers l'interprétabilité des CNN

Problèmes de cette méthode basique :

- produit des images non-reconnaissables,
- produit des images "adversaires"

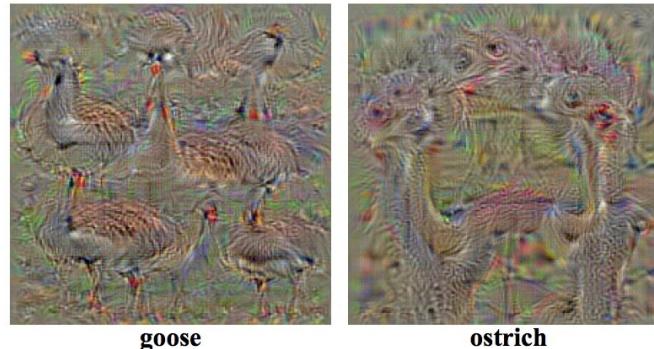


Vers l'interprétabilité des CNN

Optimisation du score d'une classe donnée par rapport à l'image d'entrée I :

$$I^* = \underset{I}{\operatorname{argmax}} S_c(I) - \lambda \|I\|_2^2$$

- Ajout d'un terme de régularisation : $-\lambda \|I\|_2^2$
- $S_c(I)$: score avant la fonction Softmax car si après Softmax, peut mener à la minimisation des scores des autres classes et non à la maximisation du score S_c



Karen Simonyan et al, 2014, <https://arxiv.org/pdf/1312.6034.pdf>

Vers l'interprétabilité des CNN

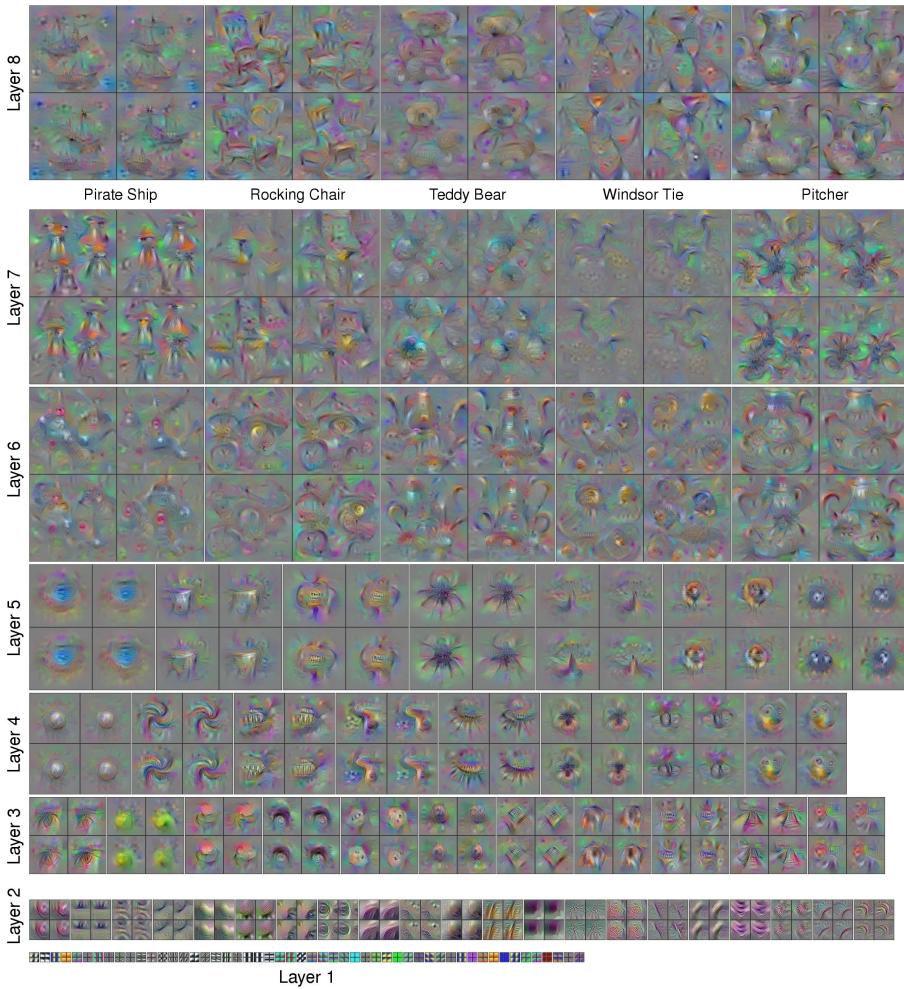
Deepvis : 4 termes de régul. différents

- ① Limitation des valeurs extrêmes de pixel : $-\lambda ||I||_2^2$
- ② Floutage gaussien : pour enlever les détails haute-fréquence
- ③ Mise à zéro des pixels de petite norme
- ④ Mise à zéro des pixels qui contribuent peu



Flamingo

Karen Simonyan et al, 2014, <https://arxiv.org/pdf/1312.6034.pdf>



<http://yosinski.com/deepvis>

Un réseau convolutionnel est un type de réseau créé au départ pour traiter les images.

Il est composé de différents éléments :

- Couche de convolution : apprentissage de caractéristiques des images. Connectivité locale. Invariance par translation. Partage de paramètres.
- Couche de normalisation : accélérer et éventuellement améliorer l'apprentissage
- Pooling : sous-échantillonnage. Max-Pooling et Average-Pooling
- Couche d'activation (ReLU)
- Couche complètement connectée : pour faire la classification
- Visualisation possible par création d'images "adversaires" et variantes