INTRODUCTION AUX RÉSEAUX DE NEURONES

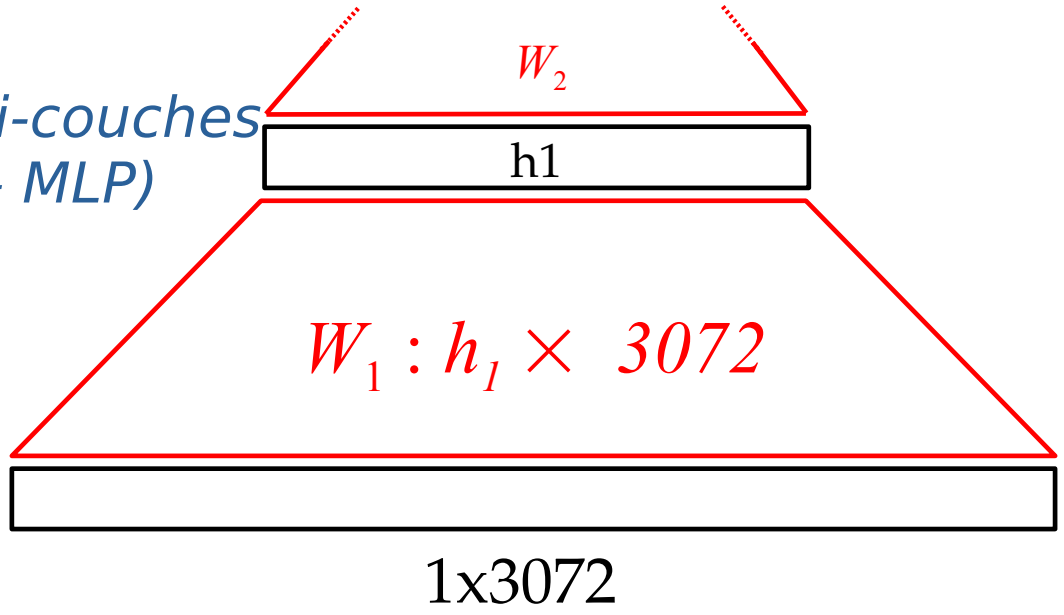
Réseaux de neurones convolutifs («CNN» : *Convolutional neural networks*)

Pascal Germain*, 2019

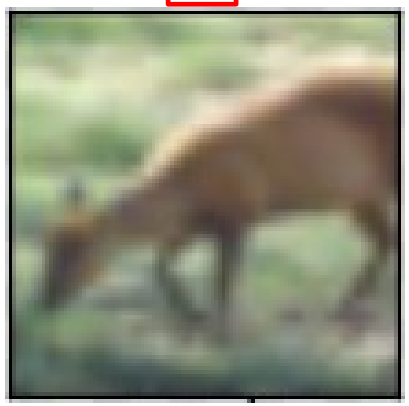
* Merci spécial à Philippe Giguère pour m'avoir permis de réutiliser une partie de ces transparents.

Réseau pleinement connecté (*fully connected)*

Aussi appelé perception multi-couches (multi-layers perceptron – MLP)



Vectorisation (*flatten*) de l'image



- Vectorisation détruit :
  - relations spatiales
  - canaux de couleurs

Image 32x32x3

La connaissance du problème influence la conception de l'architecture

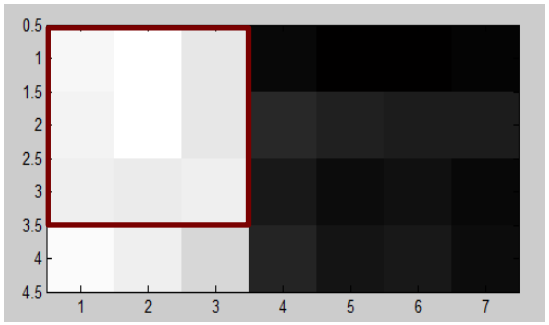
- Les images possède une structure 2D
- Forte corrélation locale dans les valeurs des pixels

Exemple « convolution »

Appellation de Filtre (*Filter*), ou Noyau (*Kernel*)

Soit une image $I \in \mathbb{R}^{m \times n}$ et un filtre de convolution $F \in \mathbb{R}^{k \times k}$ de taille impaire $k = 2d + 1$ (avec $d \in \mathbb{N}^+$):

$$(I * F)[x, y] = \sum_{i=-d}^{+d} \sum_{j=-d}^{+d} I[x+i, y+j] \times F[i+d+1, j+d+1]$$



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

207	210	195	63	57	56	59
204	212	197	82	76	74	75
202	198	202	72	65	67	63
209	201	187	78	69	71	64

$I * F$

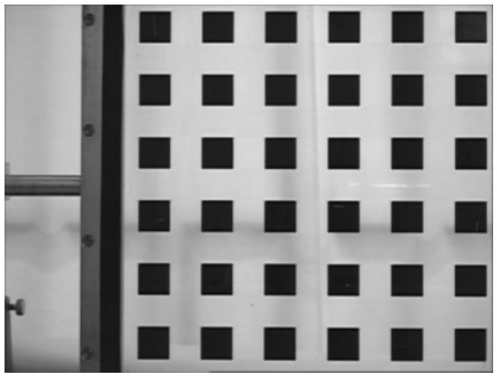
	-26	-533	-517	-28	
	-29	-505	-513	-25	

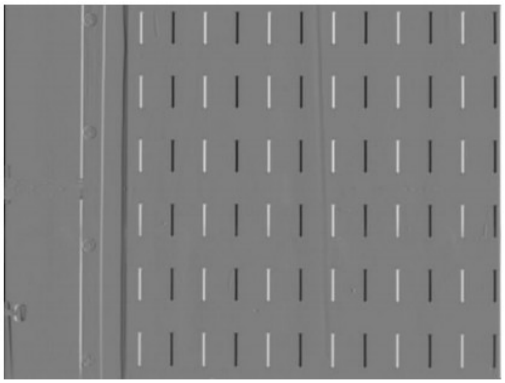
Exemples de filtres calibrés manuellement

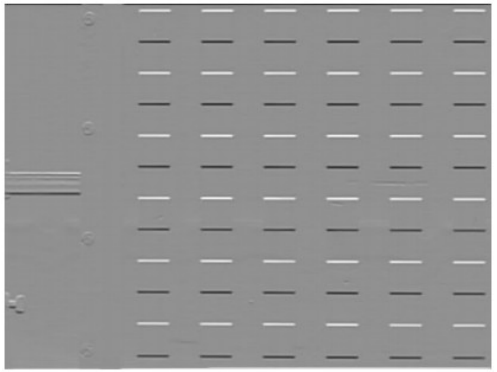
Détection bordure

Bordure verticale $\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$

Bordure horizontale $\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$







Flou

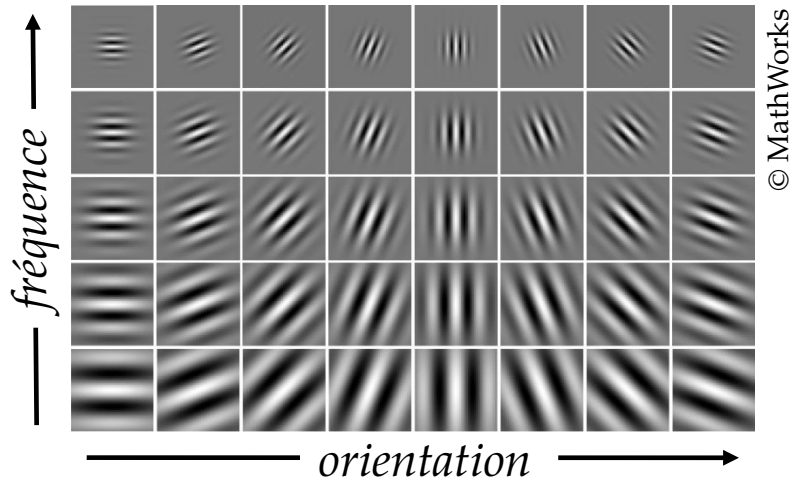
$\begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & 1 \\ 1 & 1 & 1 \end{bmatrix}$

5x5

15x15

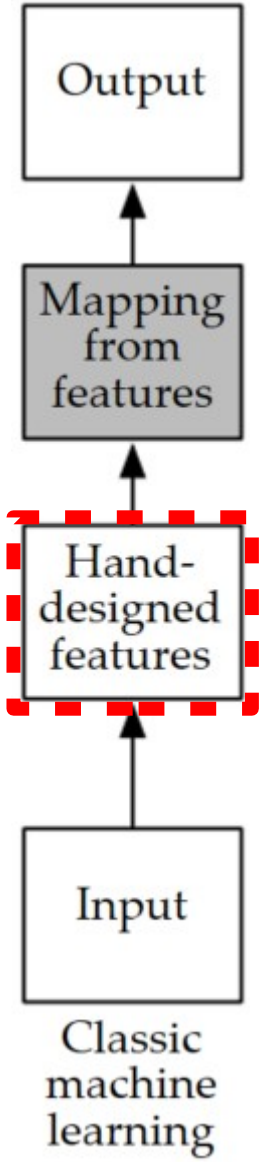
Filtres

- Vont extraire des *features* de bas niveau
  - Filtres *Gabor* :



  - Filtres de bordure, ondelettes
- Longtemps été un domaine de recherche

- Comme les filtres CNN sont différentiables, le réseau pourra les modifier à sa convenance
  - les ajuster pour maximiser les performances sur les données d'apprentissage

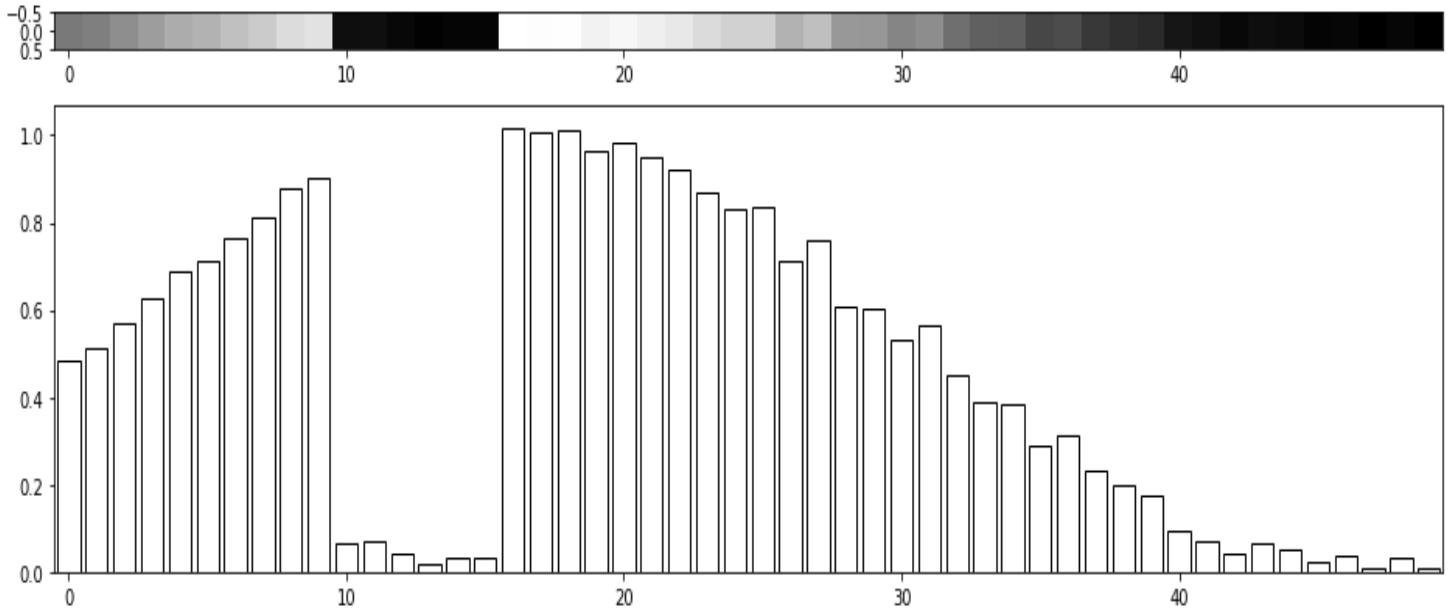


Exemple de convolution

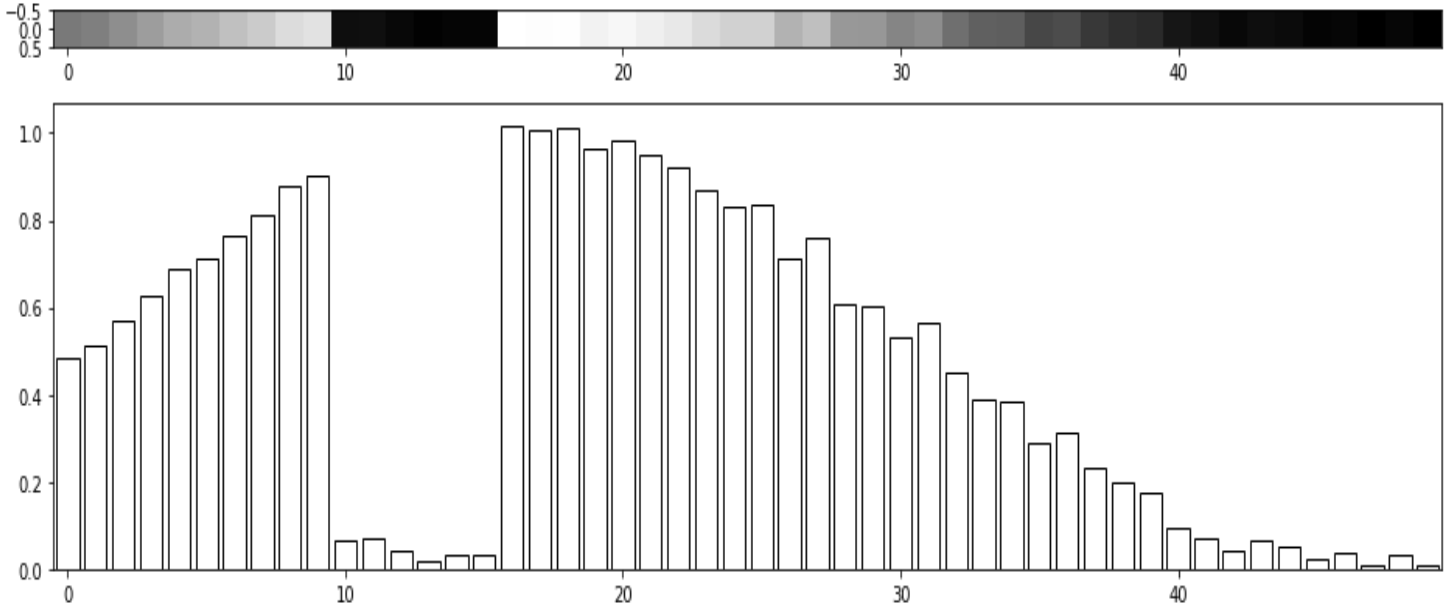
http://setosa.io/ev/image-kernels/

Concepts
en une dimension

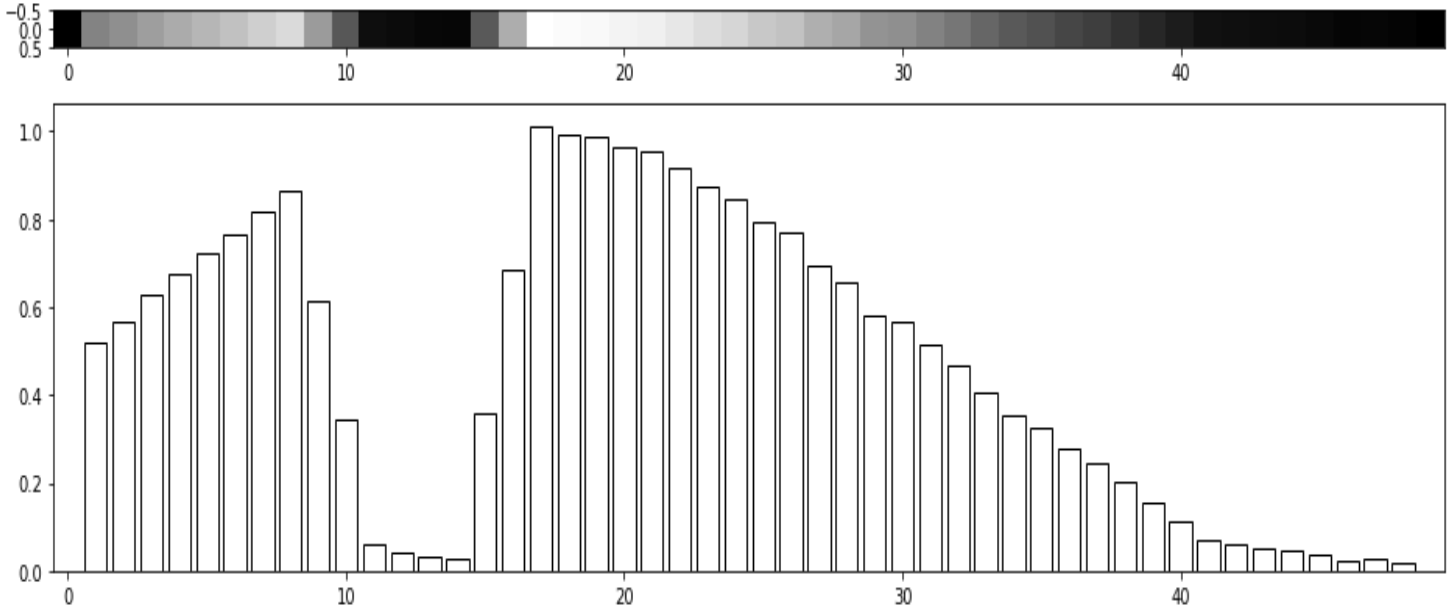
x



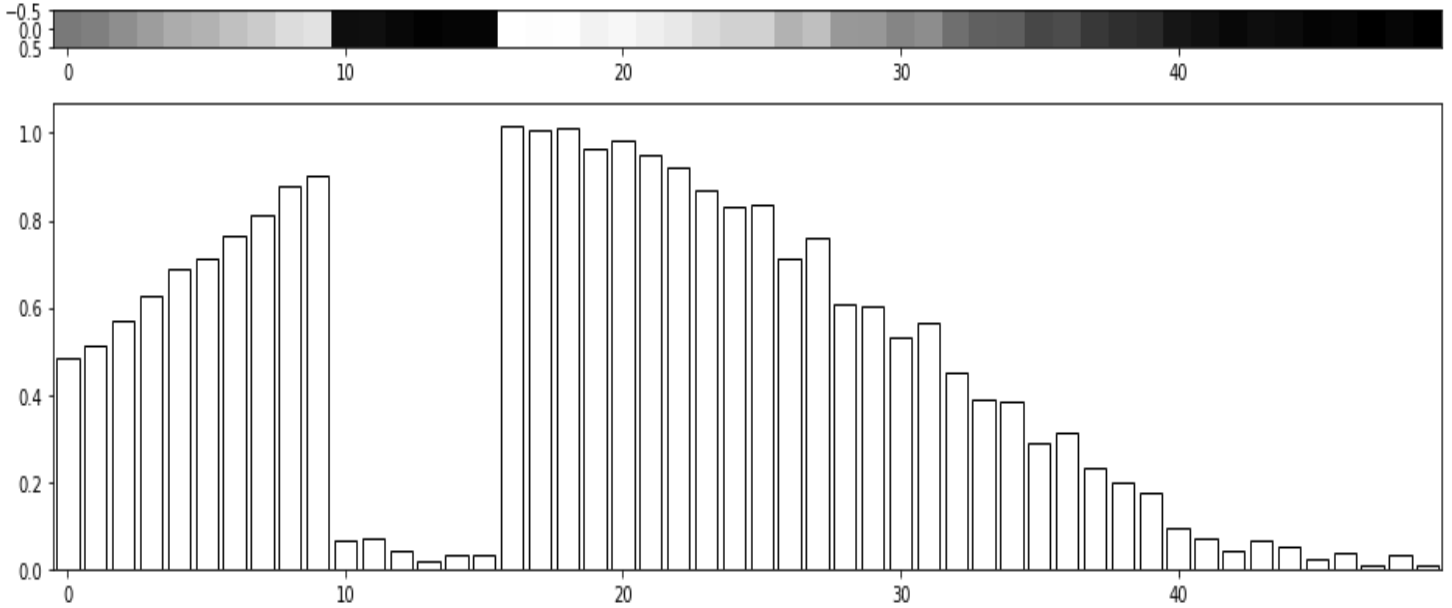
x



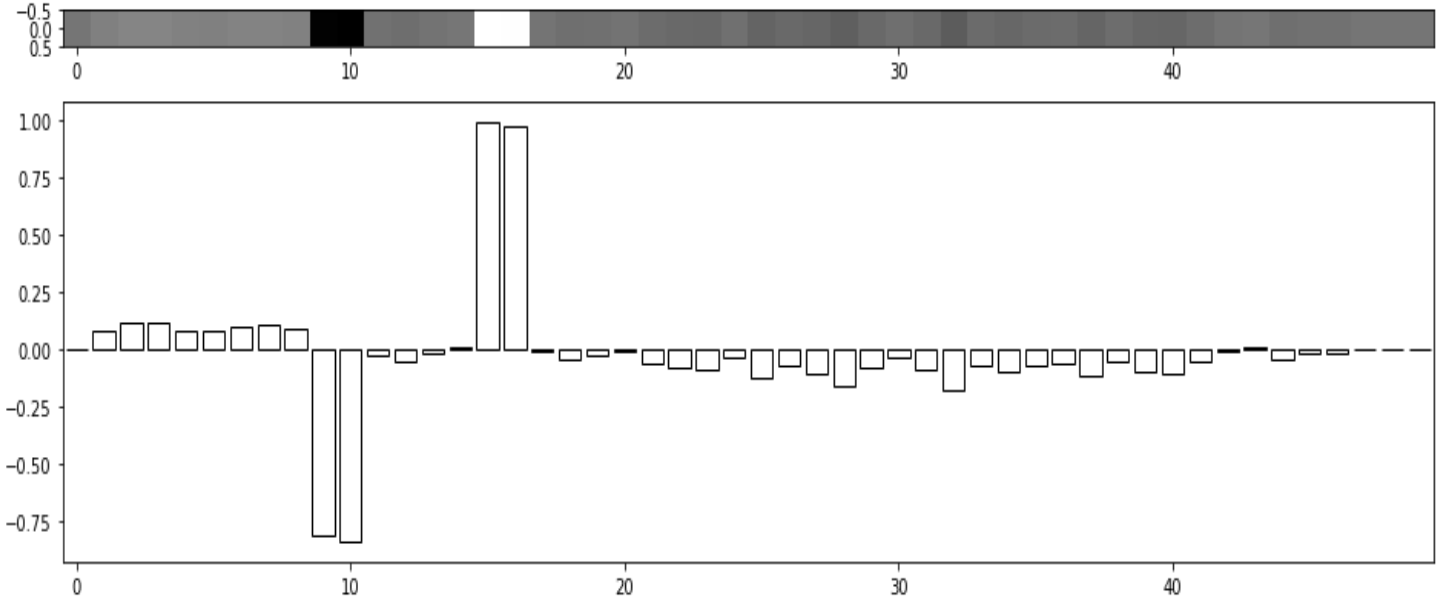
Filtre :
[1/3, 1/3, 1/3]



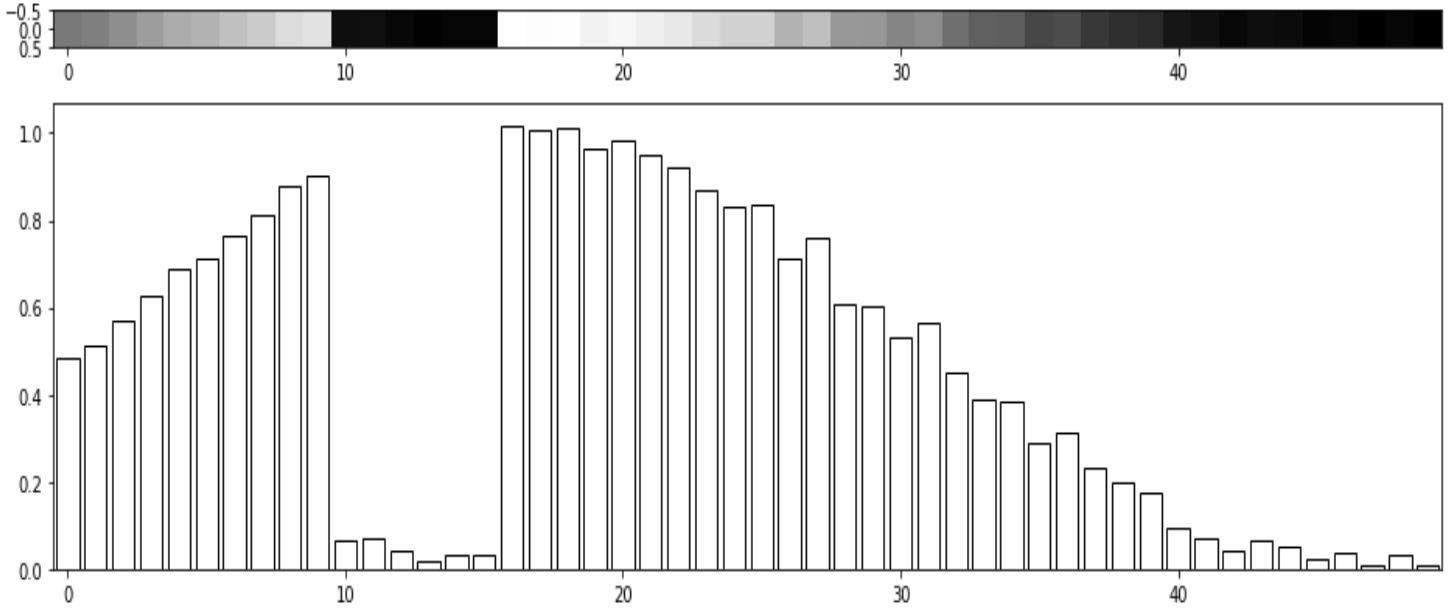
x


−0.5
0.0
0.5
0
10
20
30
40
1.0
0.8
0.6
0.4
0.2
0.0

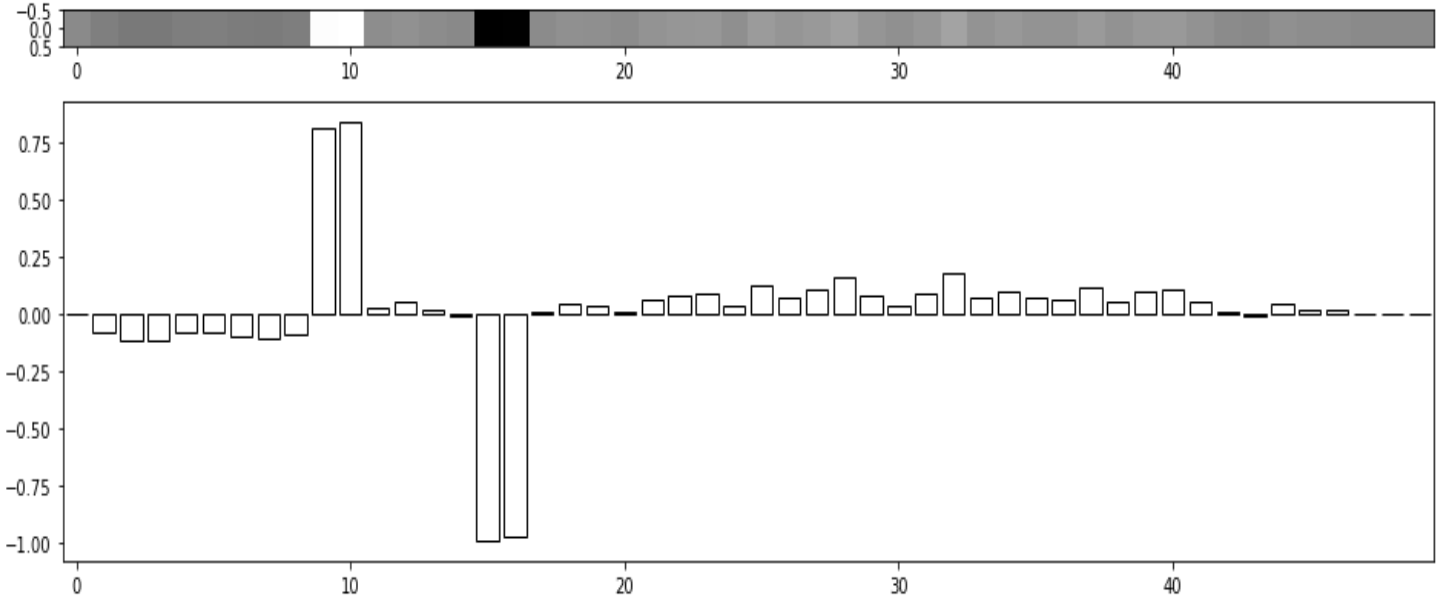
Filtre :
[-1, 0, 1]


−0.5
0.0
0.5
0
10
20
30
40
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

x

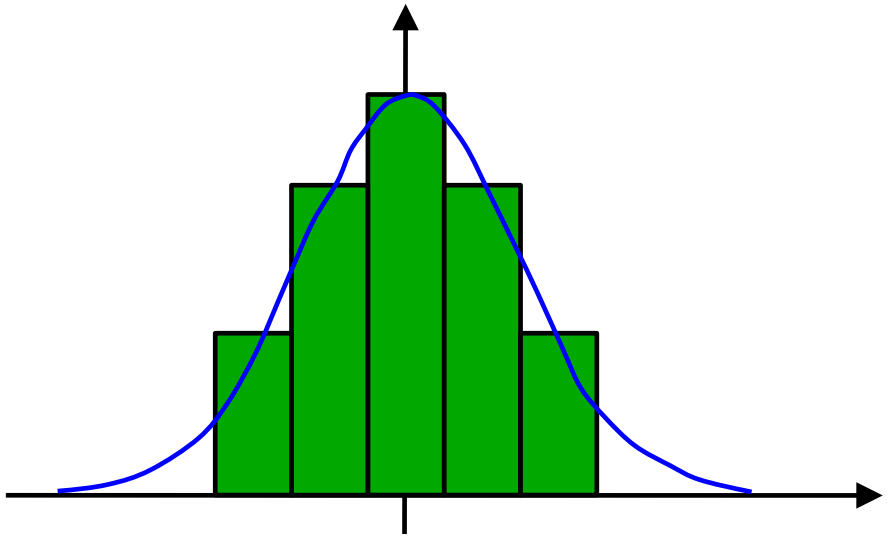

−0.5
0.0
0.5
0
10
20
30
40
1.0
0.8
0.6
0.4
0.2
0.0

Filtre :
[1, 0, -1]


−0.5
0.0
0.5
0
10
20
30
40
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

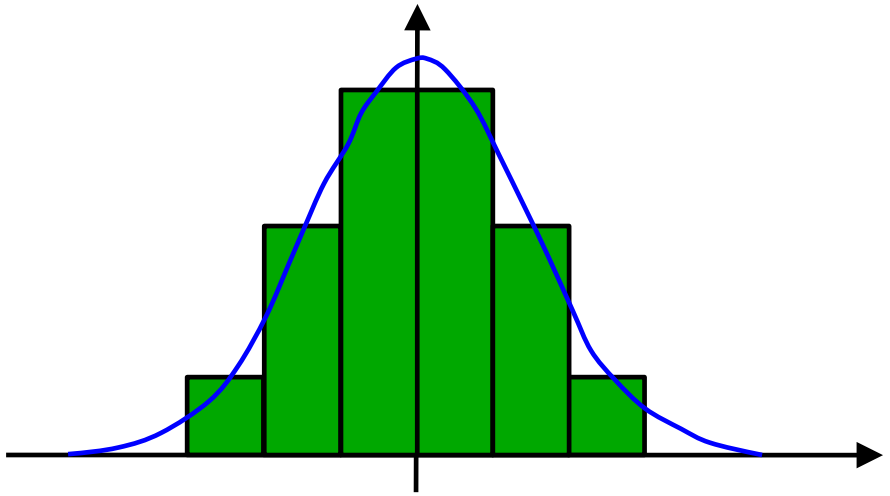
Pourquoi un filtre de taille impaire ?

- Pas de pixel « milieu » pour filtre taille paire
- Exemple :



filtre taille 5

La résultante est imputée à un seul pixel de l'image en sortie



filtre taille 6

La résultante tombe à cheval entre des pixels de l'image en sortie

(à titre informatif)

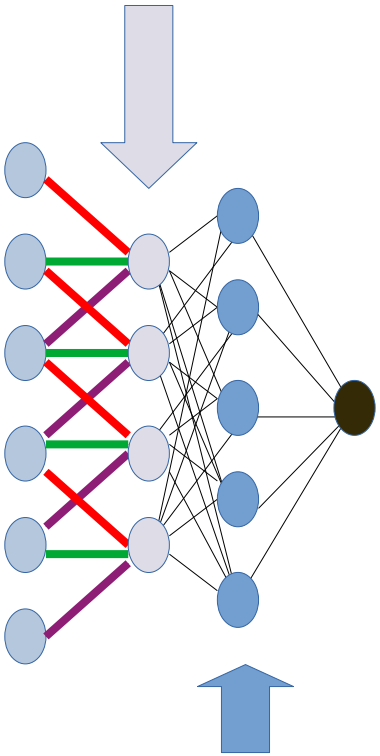
Succession de couches convolution → pleinement connectée

Couche
de convolution

Filtre : [w_1, w_2, w_3]

.....Équivalent à.....

$$\begin{bmatrix} w_1 & w_2 & w_3 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & 0 & w_1 & w_2 & w_3 \end{bmatrix}$$



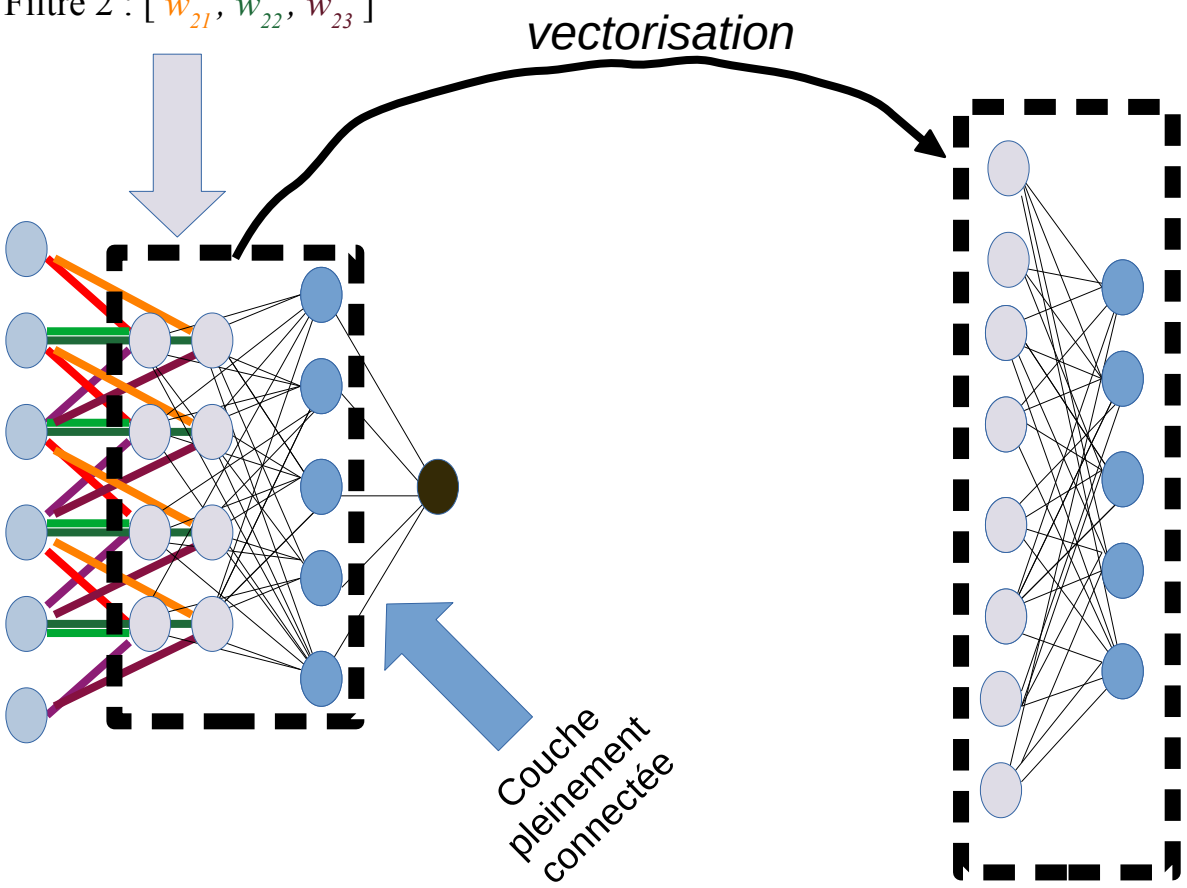
Couche
pleinement
connectée

$$\mathbf{W}^{(k)} = \begin{bmatrix} w_{1,1}^{(k)} & \cdots & w_{1,d_{k-1}}^{(k)} \\ \vdots & \ddots & \vdots \\ w_{d_k,1}^{(k)} & \cdots & w_{d_k,d_{k-1}}^{(k)} \end{bmatrix} \in \mathbb{R}^{d_k \times d_{k-1}}.$$

Plusieurs filtres pour une couche de convolution

Couche de convolution
de deux filtres

Filtre 1 : [w_{11}, w_{12}, w_{13}]
Filtre 2 : [w_{21}, w_{22}, w_{23}]



Succession de couches convolution → convolution

1ère couche de convolution

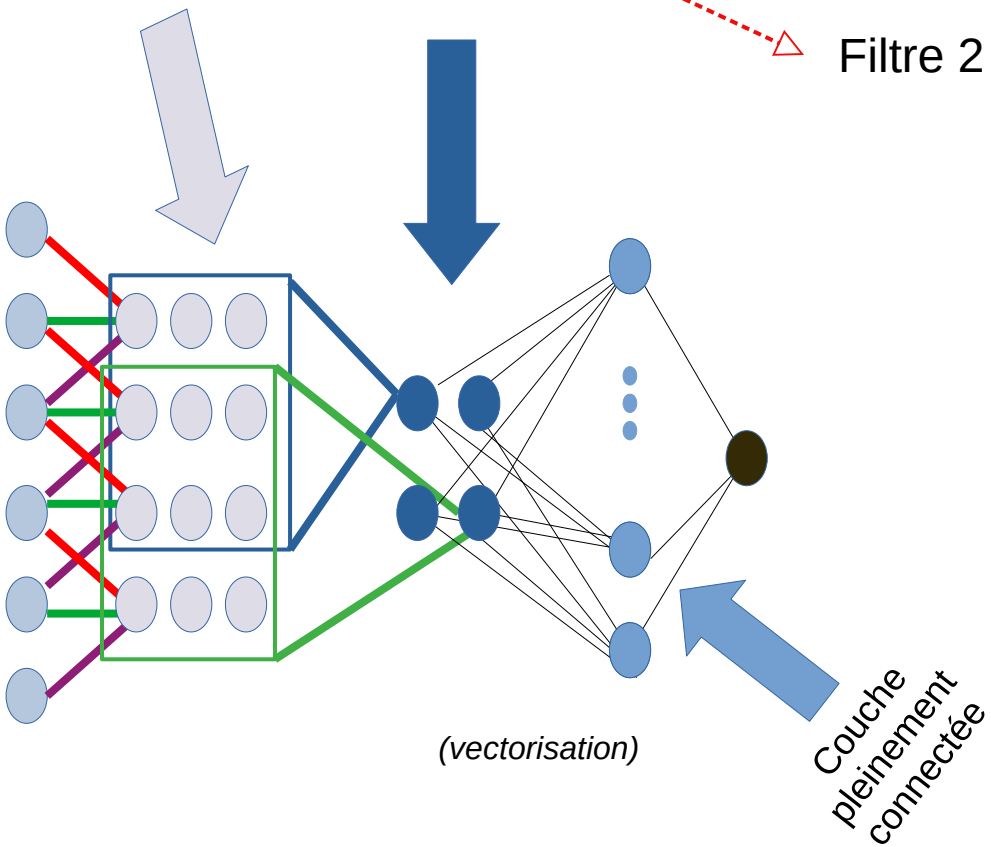
(quatre filtres de taille 1x3)

2e couche de convolution

(deux filtres de taille 3x3)

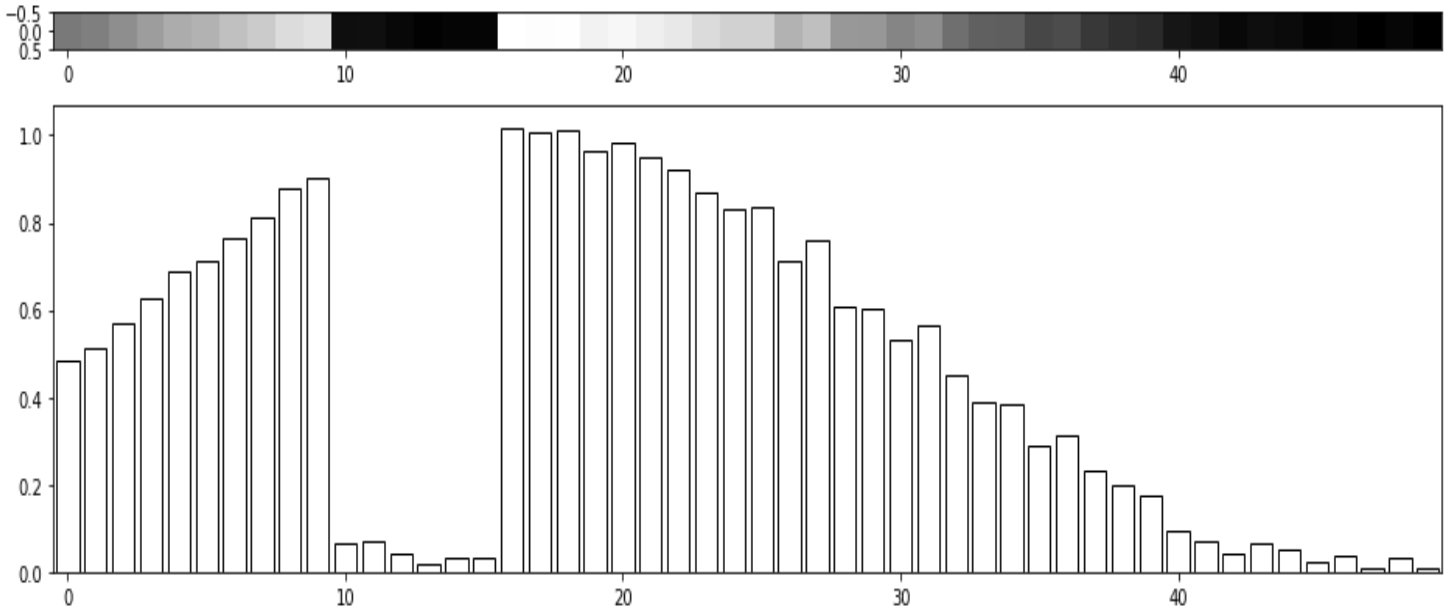
Filtre 1 : $\begin{bmatrix} w_{111}, w_{112}, w_{113} \\ w_{121}, w_{122}, w_{123} \\ w_{132}, w_{132}, w_{133} \end{bmatrix}$

Filtre 2 : $\begin{bmatrix} w_{211}, w_{212}, w_{213} \\ w_{221}, w_{222}, w_{223} \\ w_{232}, w_{232}, w_{233} \end{bmatrix}$



Fonction d’activation

x



Filtre 1: [1/3, 1/3, 1/3]
Filtre 2: [-1, 0, 1]
Filtre 3: [1, 0 , -1]

Activation

Valeurs de la couche cachée

Couches de convolutions sur des images en couleurs

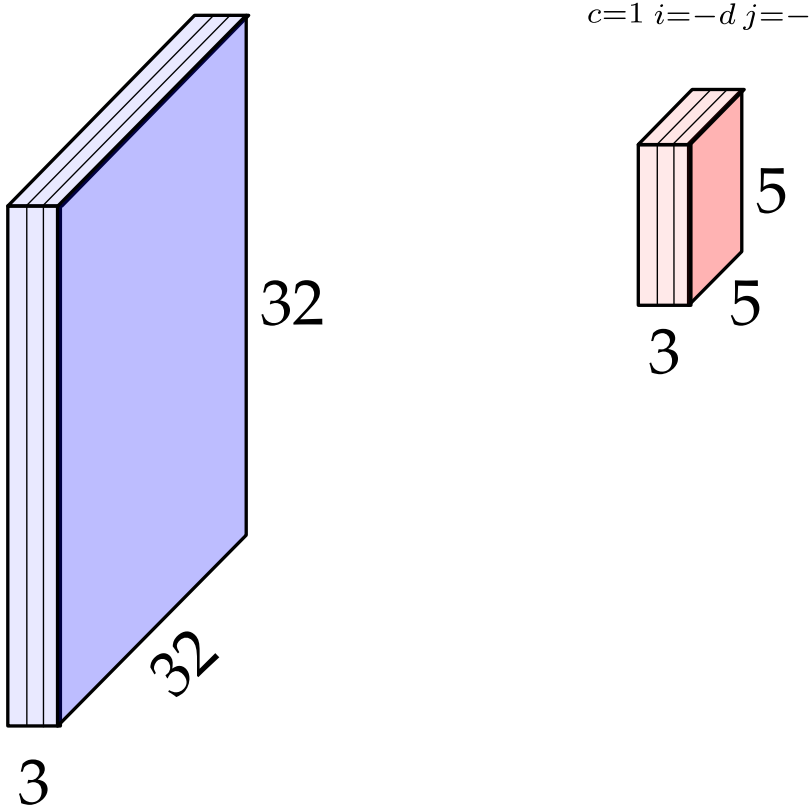
Photo de Sergueï Prokoudine-Gorski
voir https://fr.wikipedia.org/wiki/Sergue%C3%AF_Prokoudine-Gorski

Filtres convolutifs

Une image couleur possède typiquement *C* = *3* canaux (RGB) :

Soit une image $I \in \mathbb{R}^{C \times m \times n}$ et un filtre de convolution $F \in \mathbb{R}^{C \times k \times k}$:

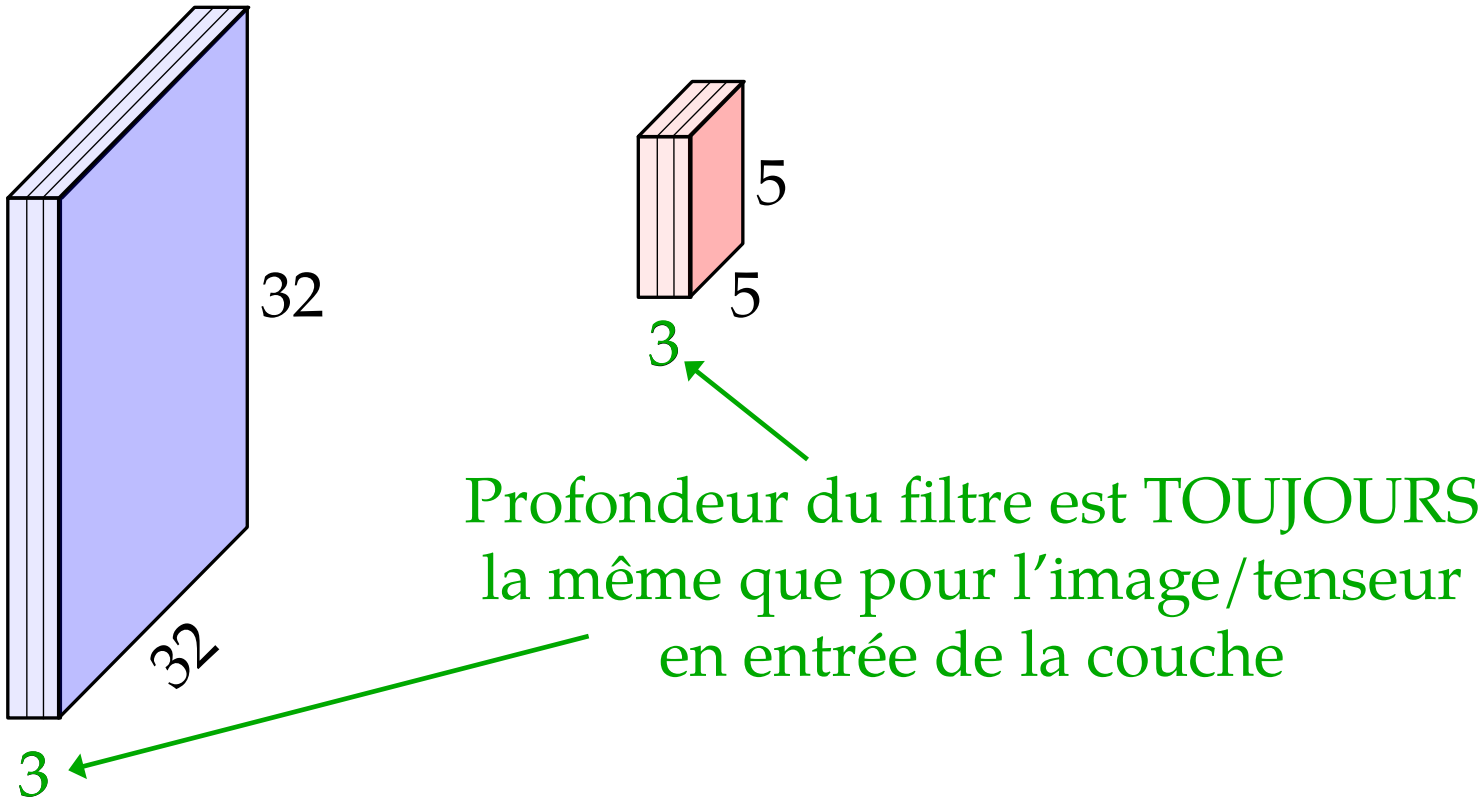
$$(I * F)[x, y] = \sum_{c=1}^{C} \sum_{i=-d}^{+d} \sum_{j=-d}^{+d} I[c, x+i, y+j] \times F[c, i+d+1, j+d+1]$$



adapté de : cs231n, Université Stanford

Filtres convolutifs

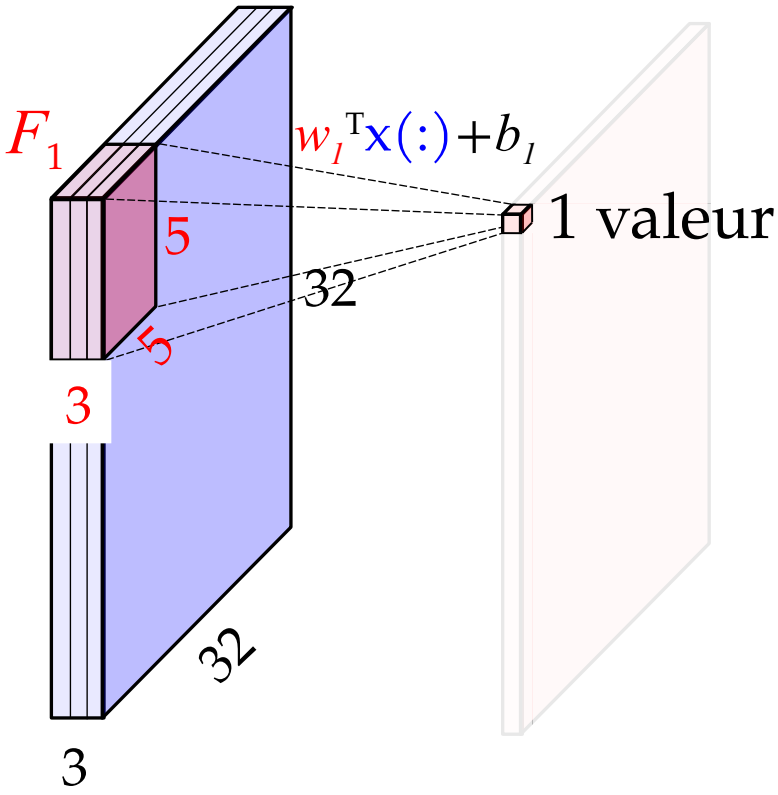
- Conserve la structure spatiale/couleur de l'entrée



adapté de : cs231n, Université Stanford

Filtres convolutifs

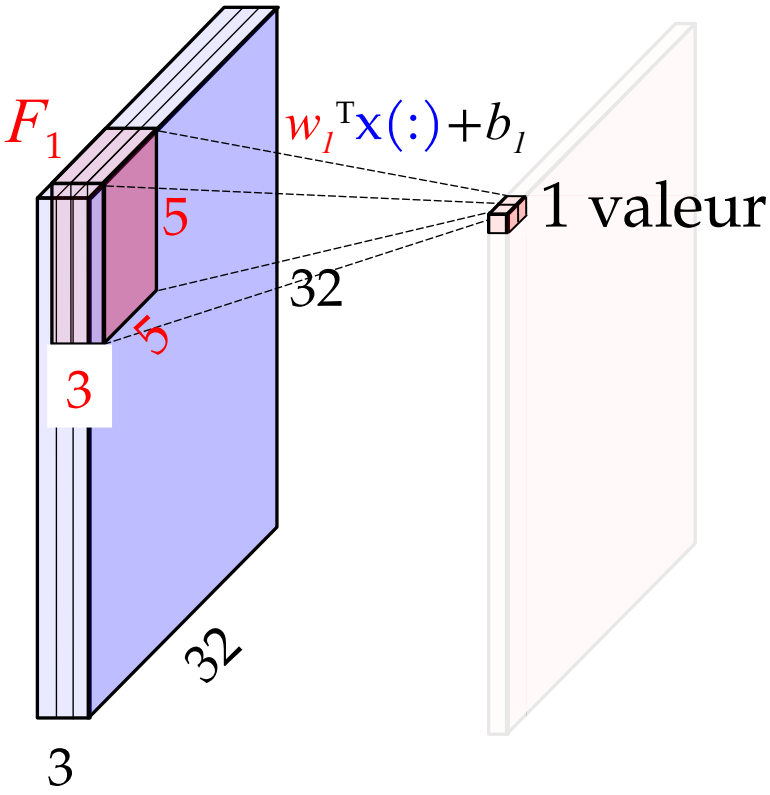
- « Glisser » spatialement le filtre F sur l'image, en calculant produit scalaire à chaque endroit de x



adapté de : cs231n, Université Stanford

Filtres convolutifs

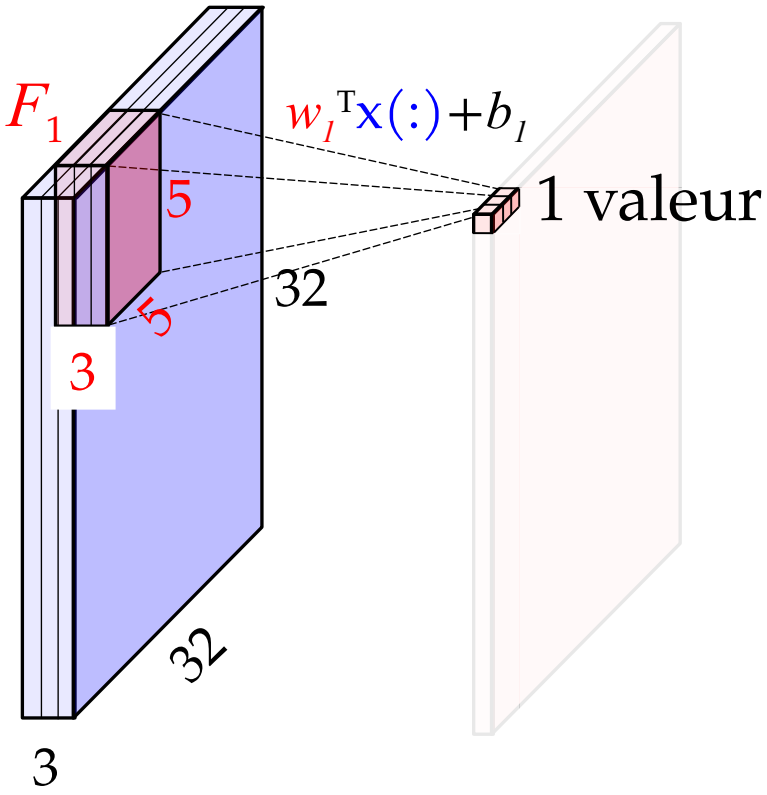
- « Glisser » spatialement le filtre F sur l'image, en calculant produit scalaire à chaque endroit de x



adapté de : cs231n, Université Stanford

Filtres convolutifs

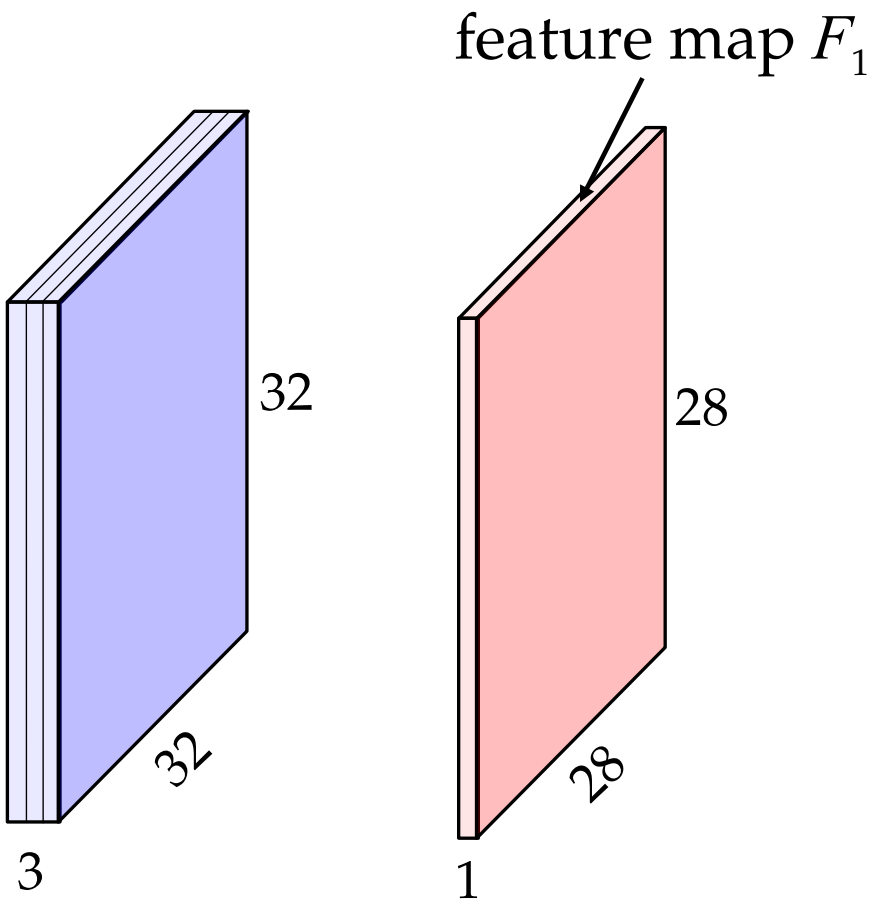
- « Glisser » spatialement le filtre F sur l'image, en calculant produit scalaire à chaque endroit de x



adapté de : cs231n, Université Stanford

Filtres convolutifs

- Sortie : *feature map*

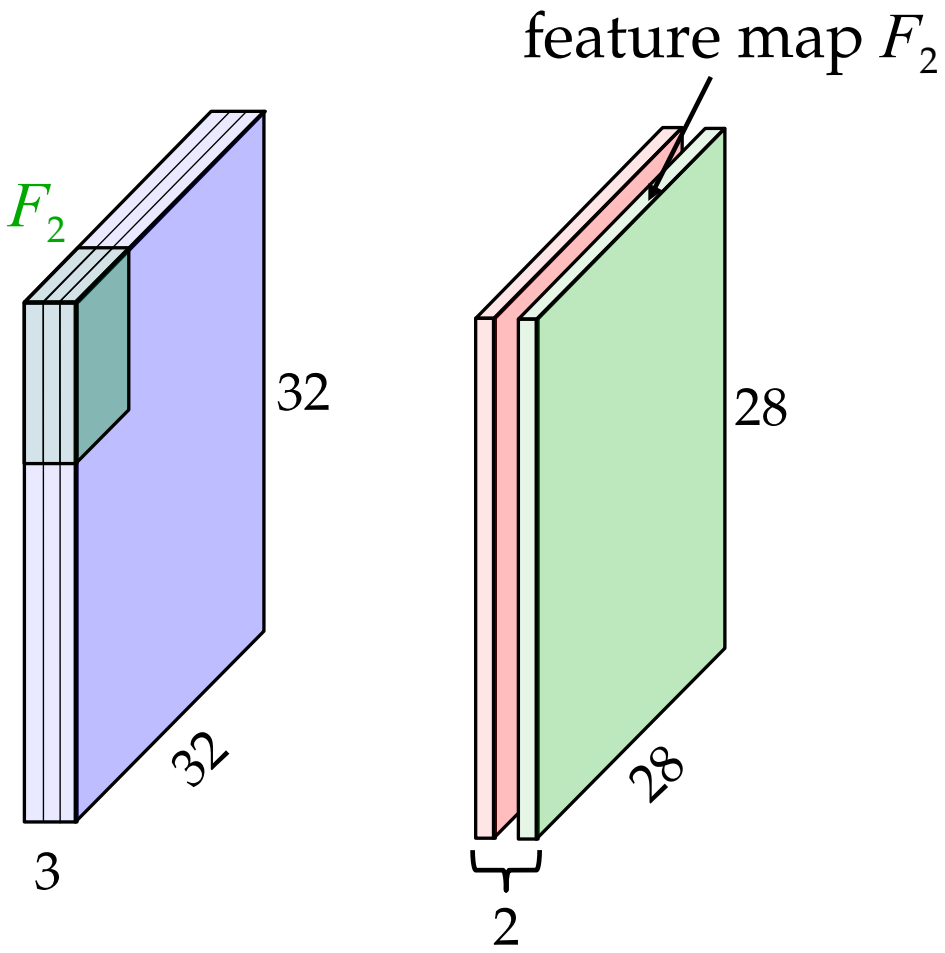


« mesure » la présence du feature F_1 à cet endroit de l'image.

adapté de : cs231n, Université Stanford

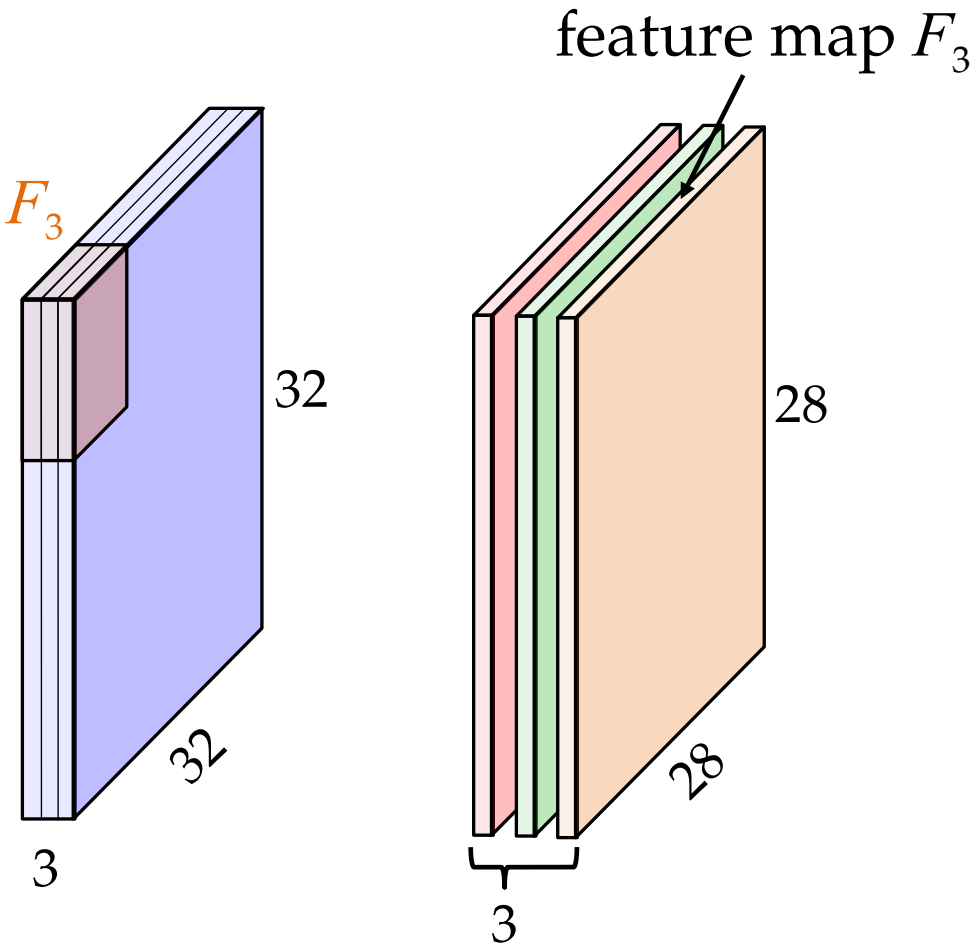
(*Note : la taille de sortie est différente ici…*)

Banque de filtres convolutifs



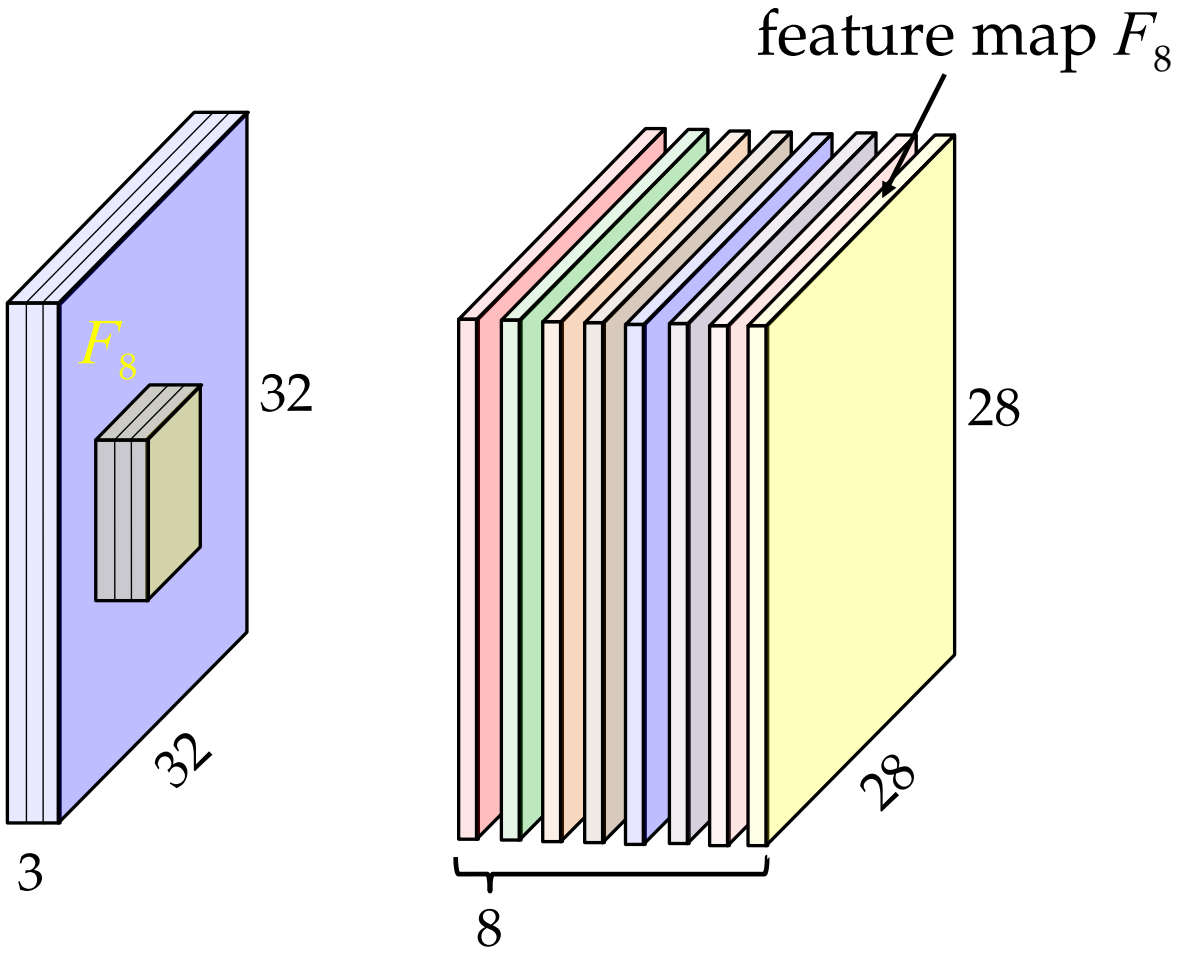
adapté de : cs231n, Université Stanford

Banque de filtres convolutifs



adapté de : cs231n, Université Stanford

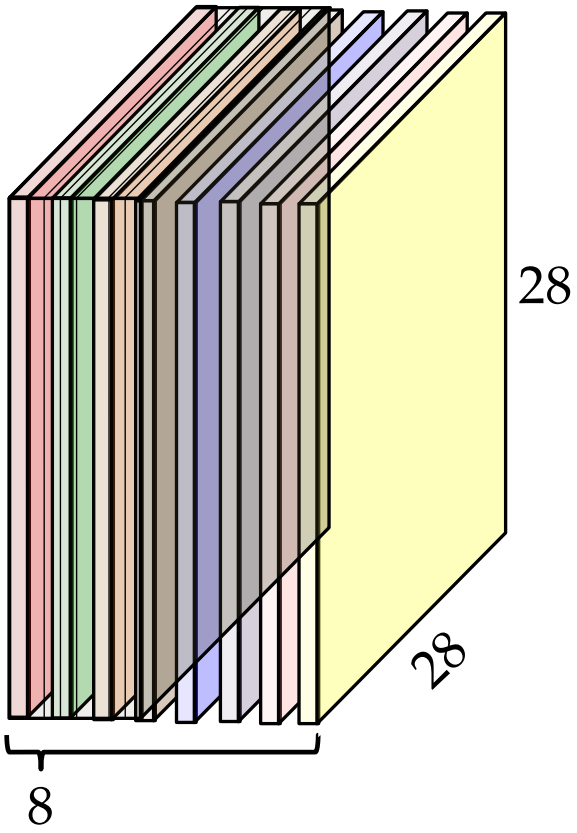
Banque de filtres convolutifs



adapté de : cs231n, Université Stanford

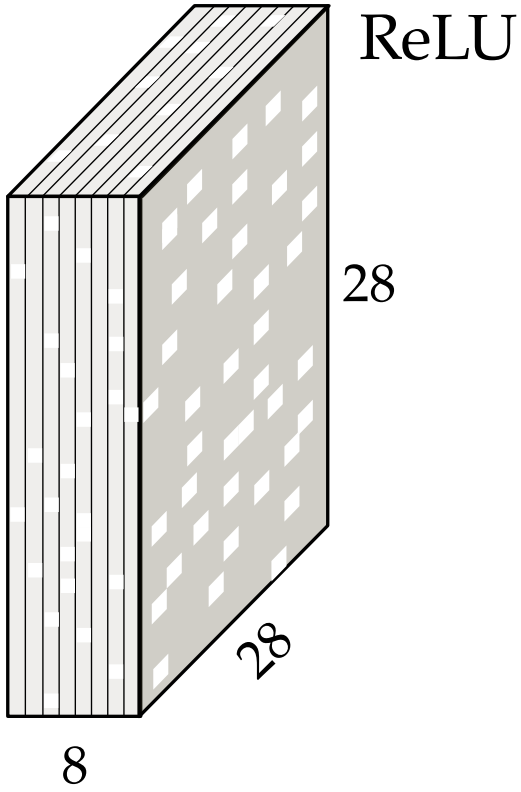
Résultante

Tenseur ordre 3



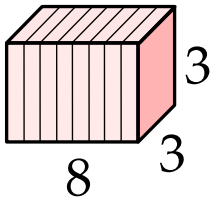
Résultante

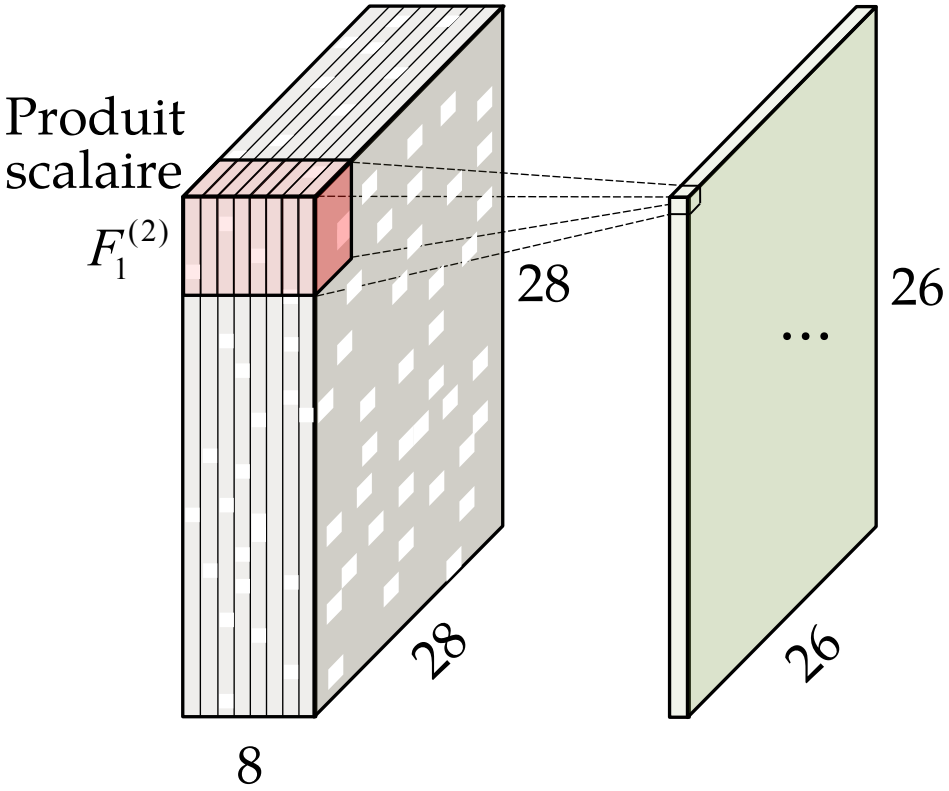
- Applique la fonction d'activation sur chaque entrée, individuellement

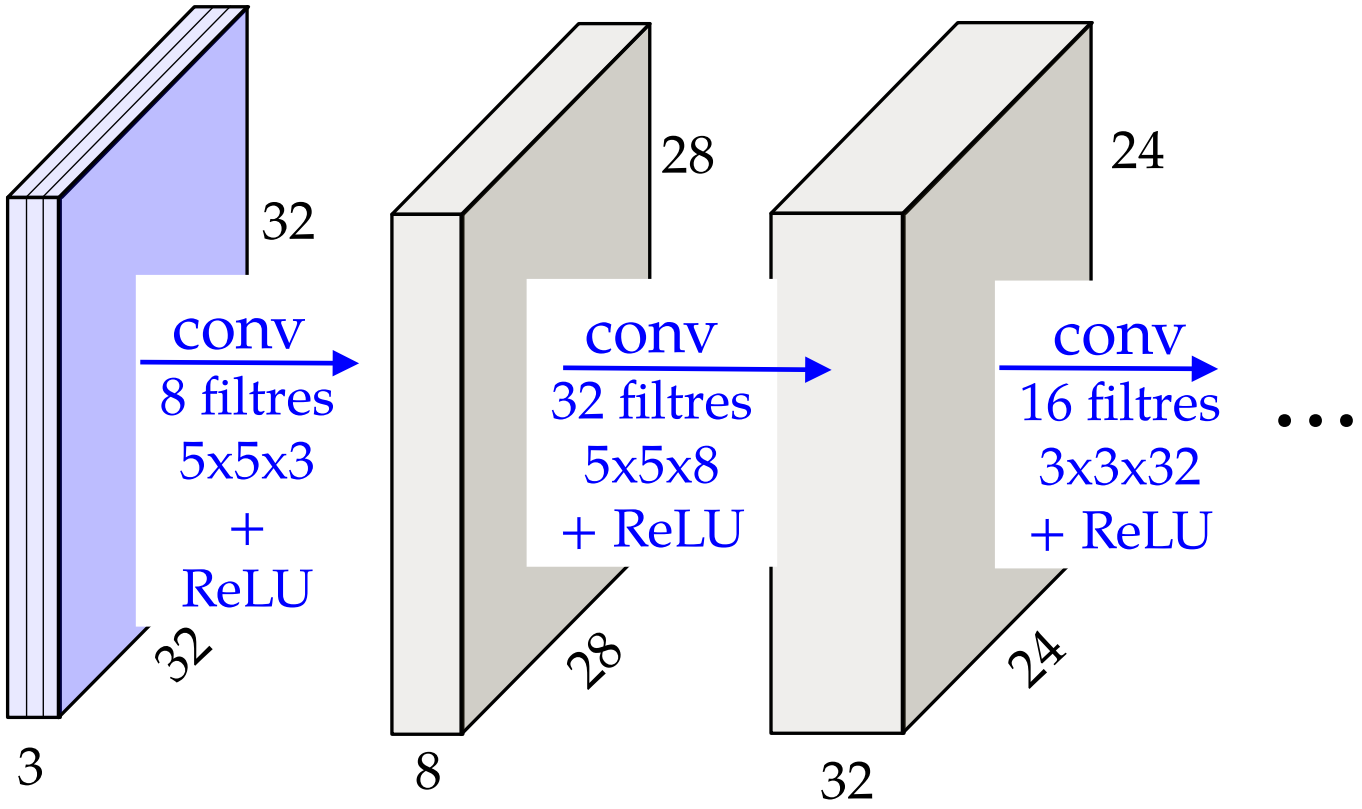


Succession de couches de convolutions

On recommence, avec une banque de filtres différents

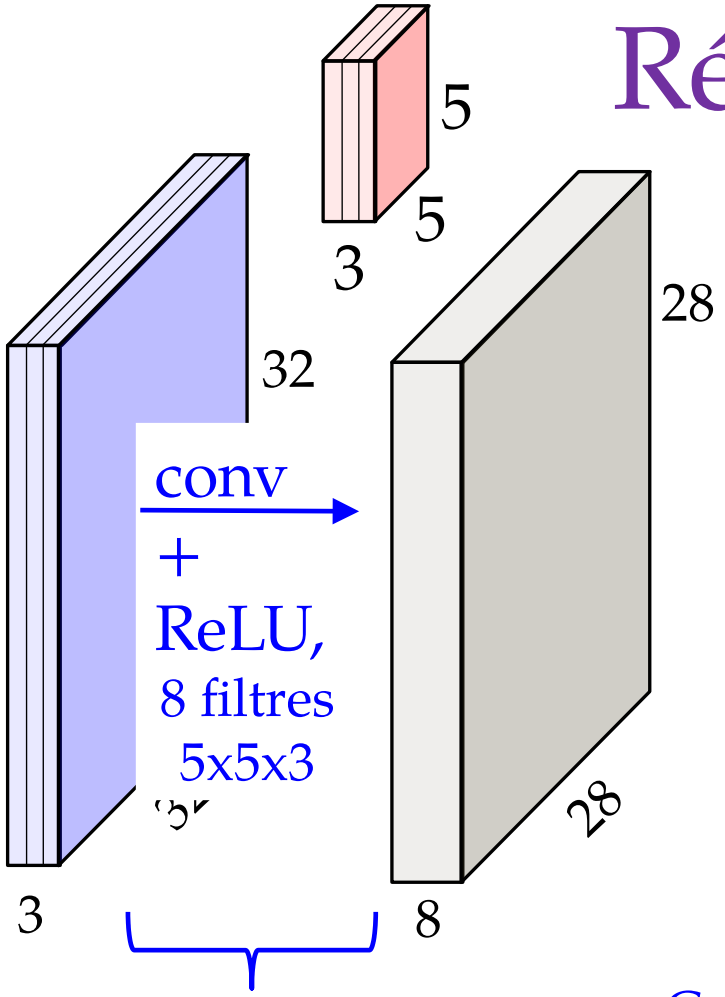






adapté de : cs231n, Université Stanford

Réduction # paramètres



- Perte de capacité d'expression vs. couche pleinement connectée
  - on ne peut pas établir de lien entre les pixels très éloignés
- Réduction importante du nombre de paramètre

Combien de paramètres pour cette couche?

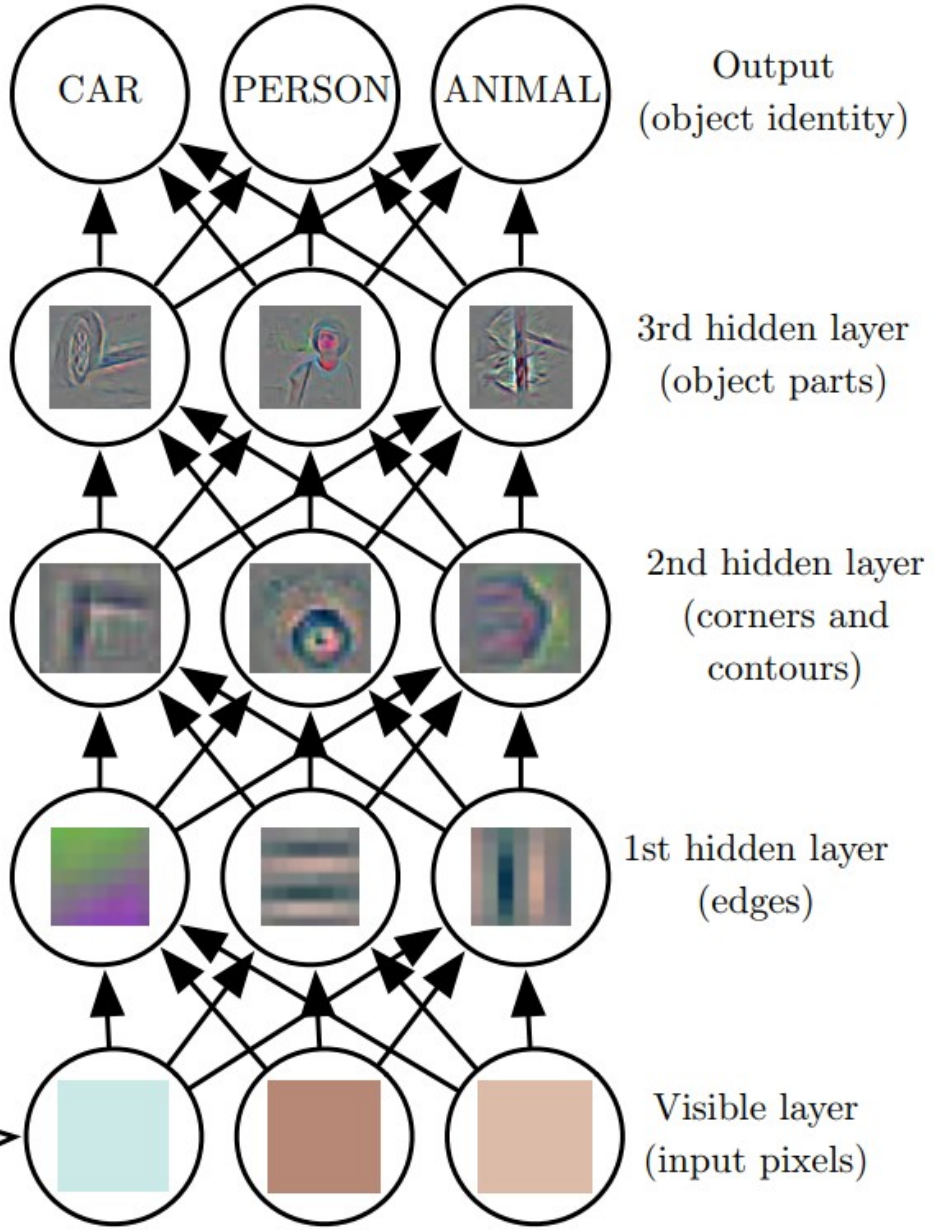
$$(5x5x3) \times 8^{\text{filtres}} = 600$$

Combien de paramètres pour une couche pleinement connectée ?
Couche d'entrée : 32x32x3 = 3072 neurones
Couche cachée: 28x28x8 = 6272 neurones
Total : 3072x6272 =19,267,584 paramètres

adapté de : cs231n, Université Stanford

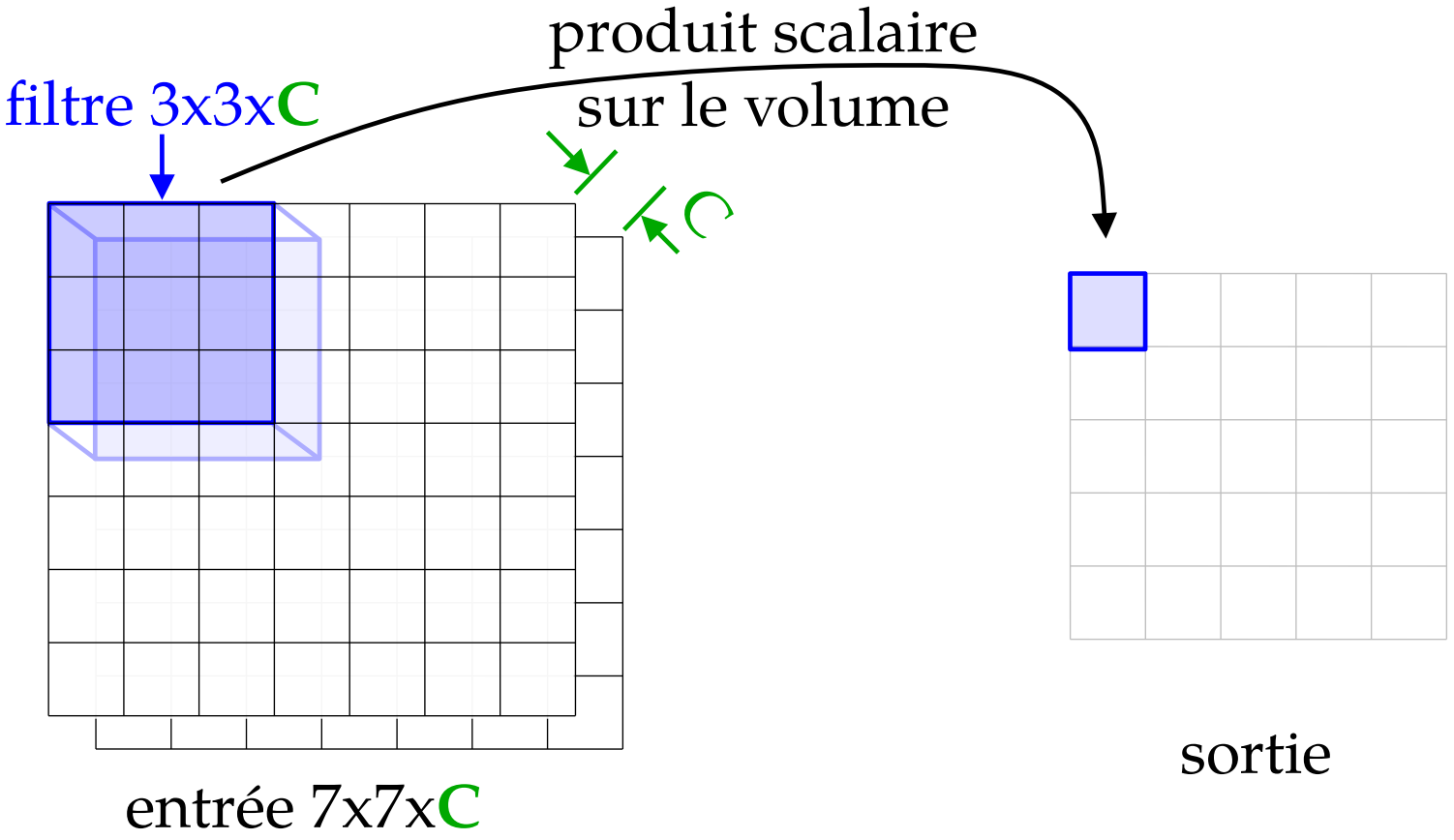
Hiérarchie de filtres

Perme d'établir des liens entre des pixels de plus en plus éloignés

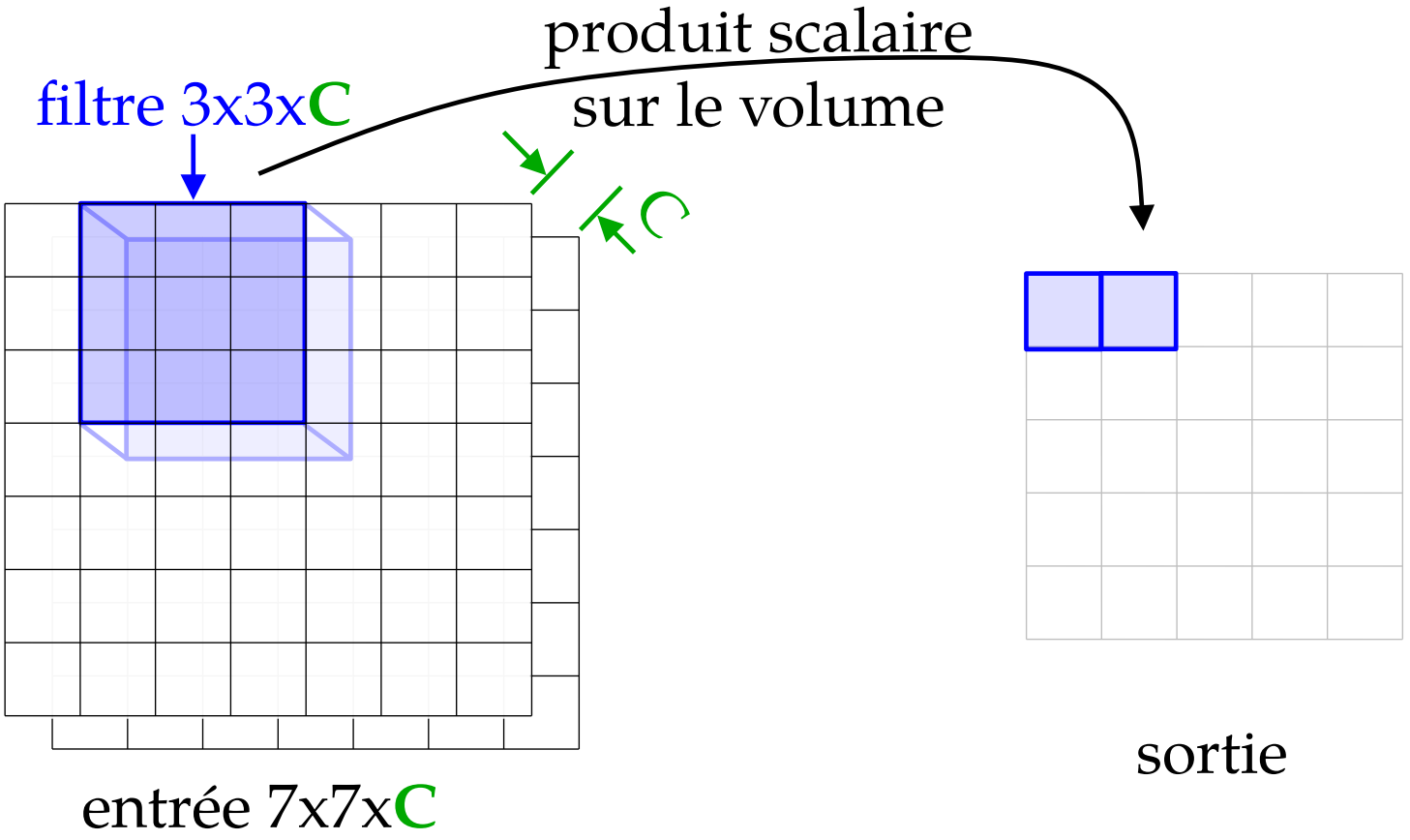


(I. Goodfellow) deeplearningbook.org

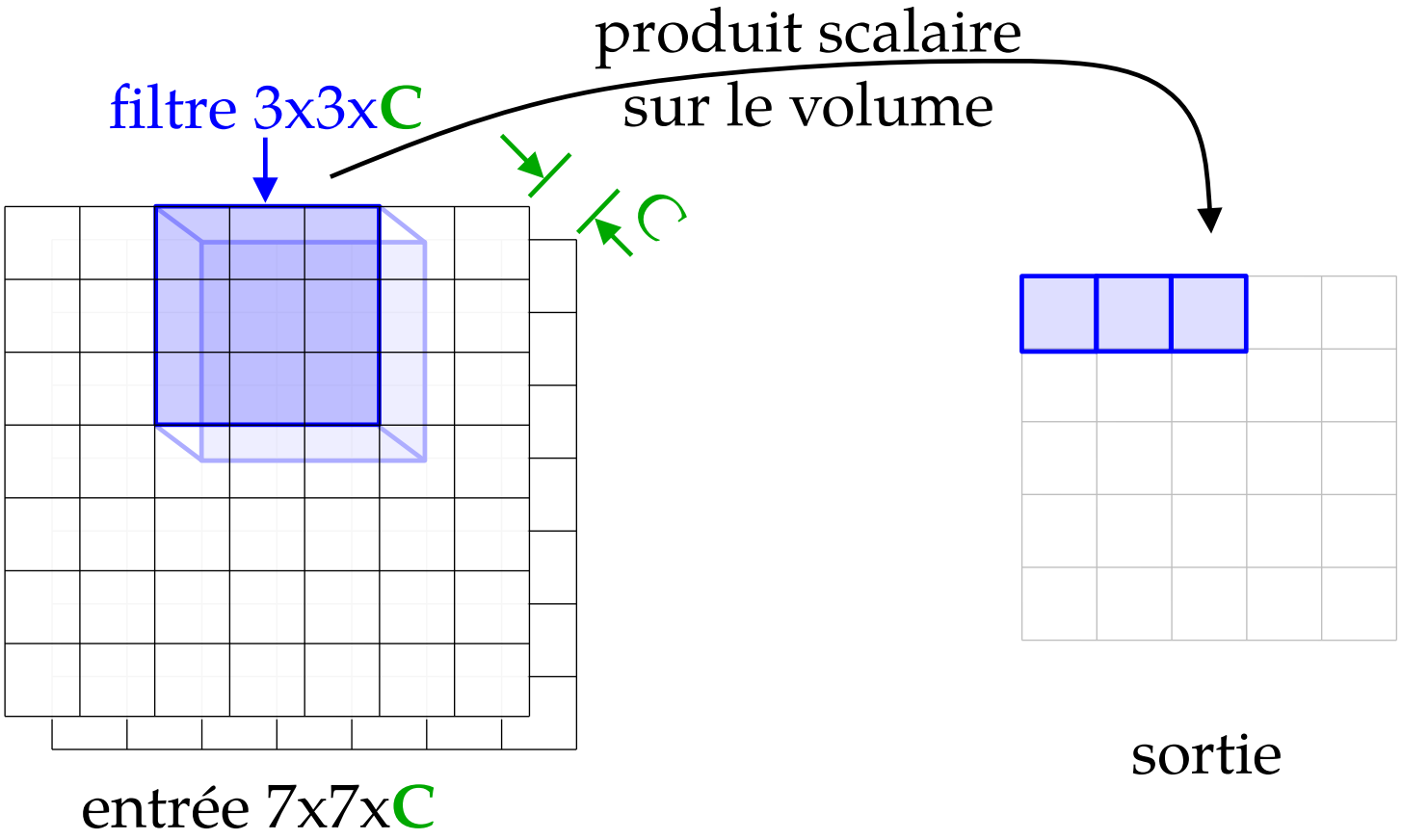
Effet de bord: changement de dimensions



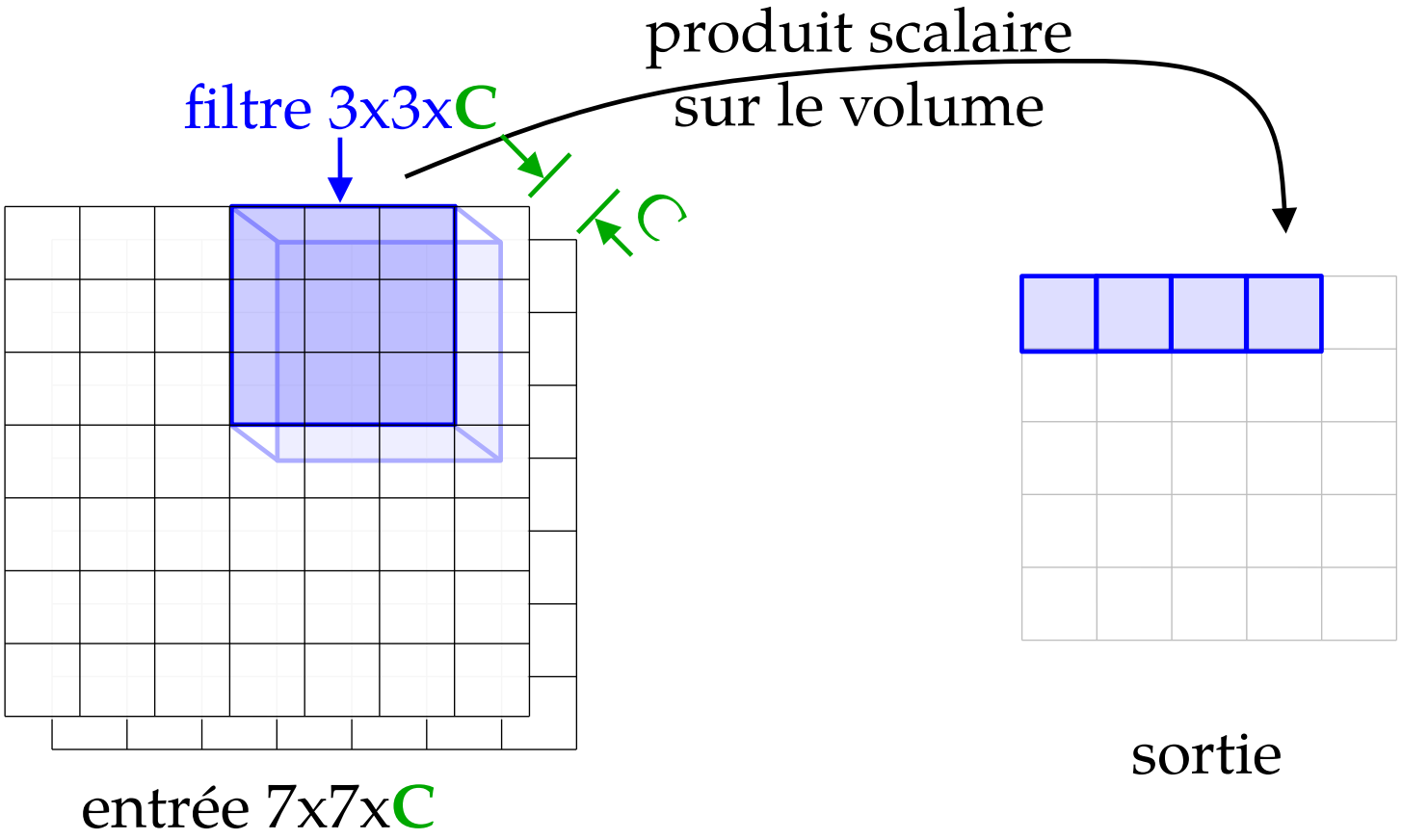
Effet de bord: changement de dimensions



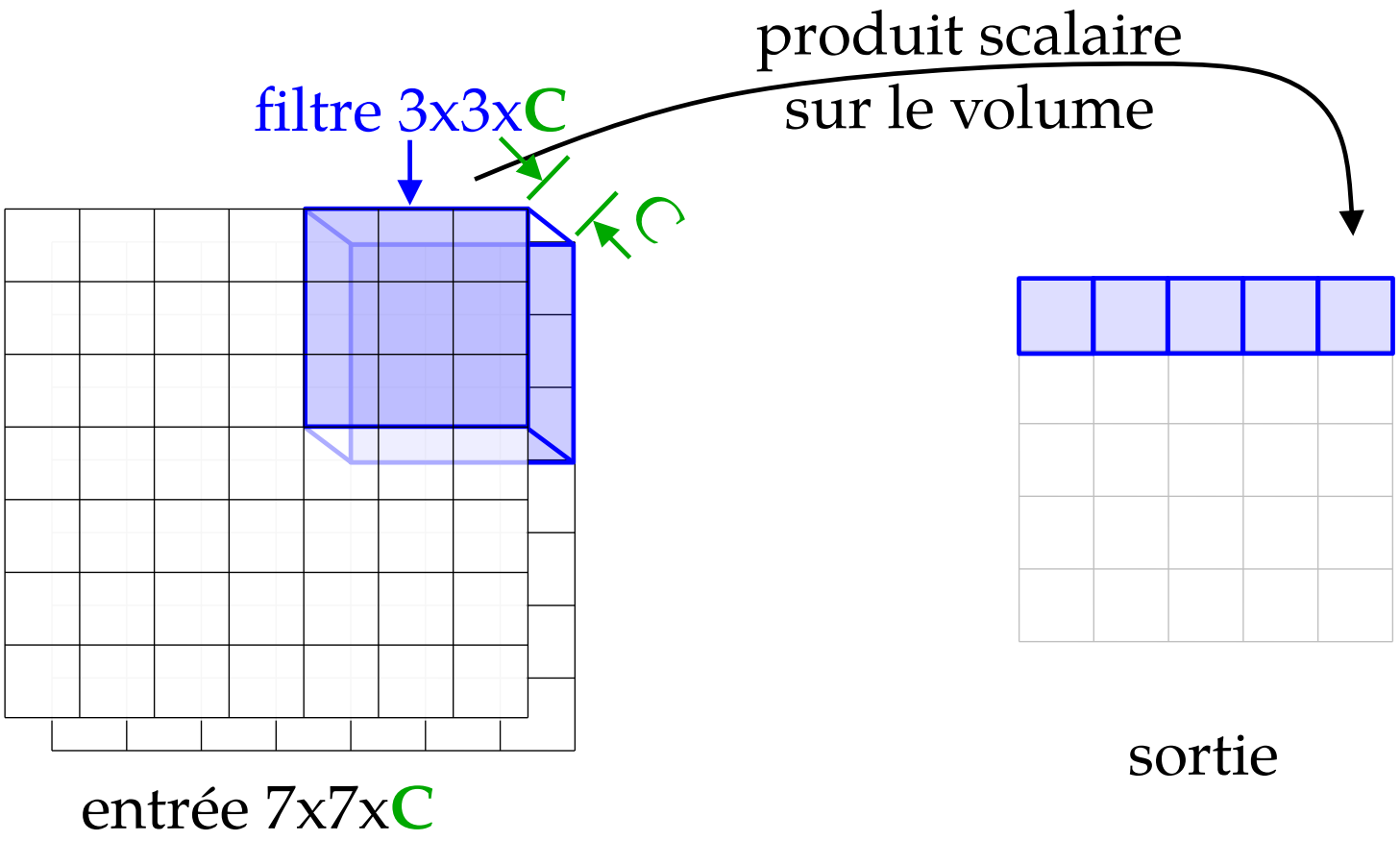
Effet de bord: changement de dimensions



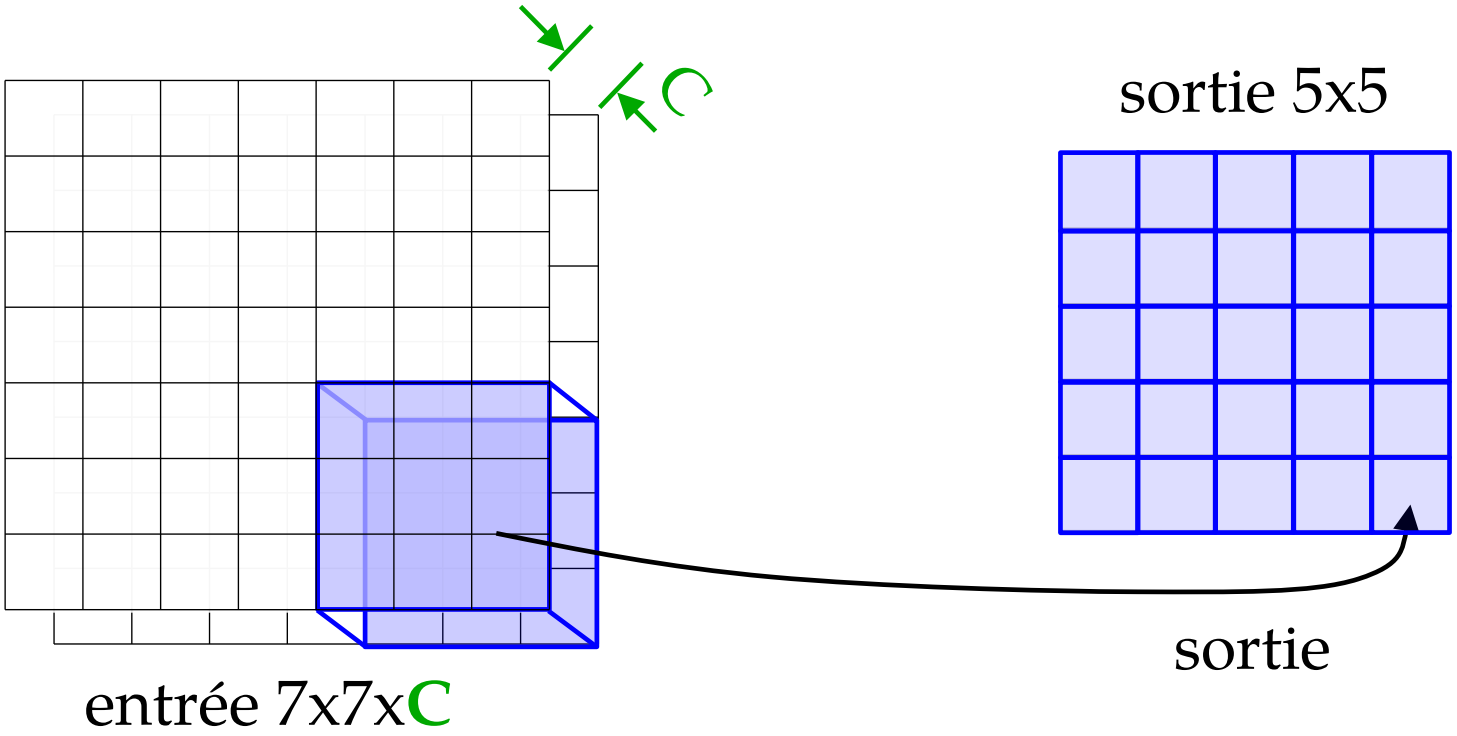
Effet de bord: changement de dimensions



Effet de bord: changement de dimensions



Effet de bord: changement de dimensions



Quelques questions!

- Si j'ai une image de 224x224x3 en entrée, complétez la taille du filtre : 5x5x**?**

  Réponse : 5x5x**3**

- Si j'ai 32 filtres 5x5 appliqués sur cette image 224x224x3, quelle sont les dimensions du tenseur de sortie?

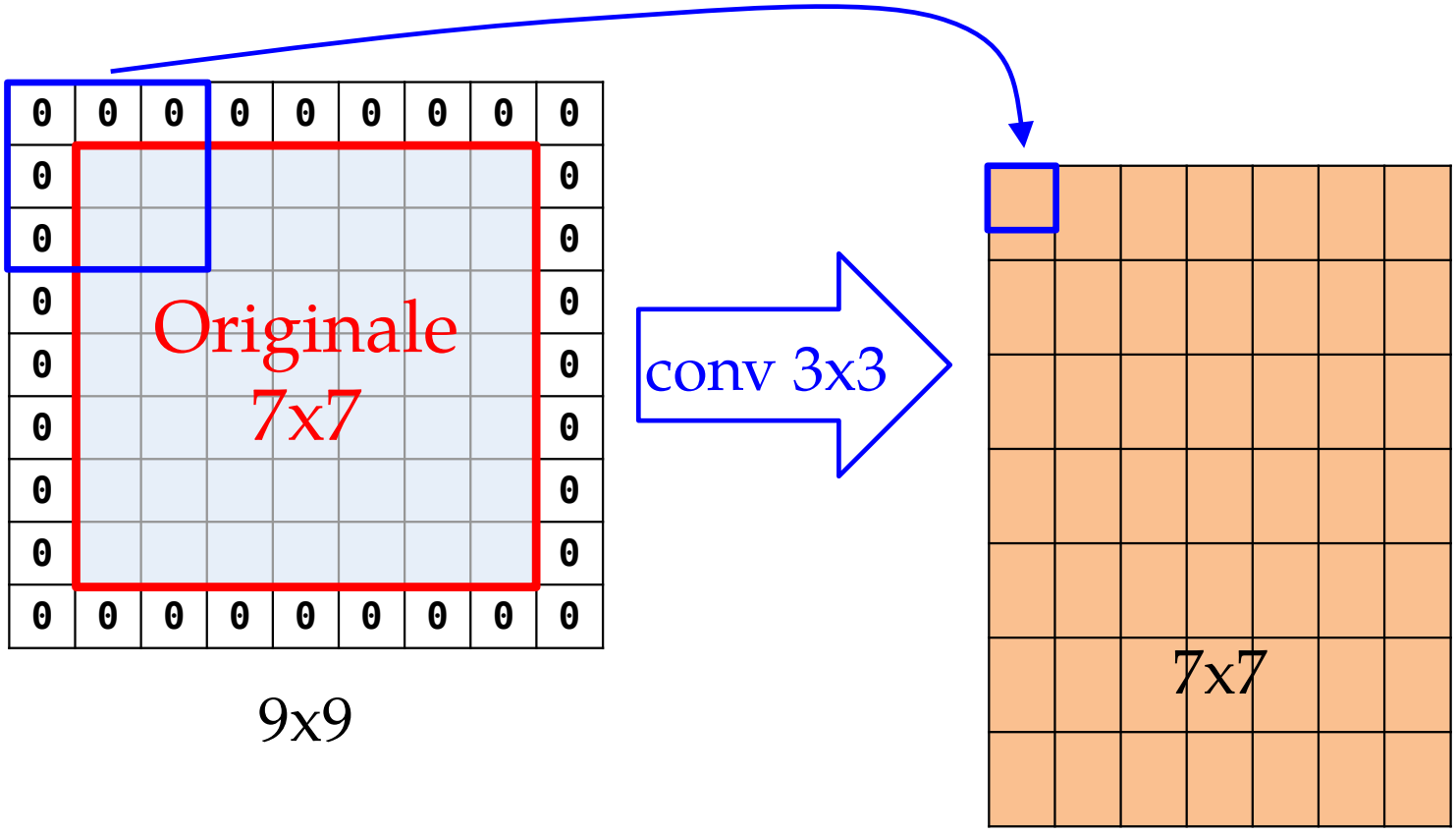
  Réponse : 220x220x32

- Combien de paramètres à optimiser comporte cette couche cachée ?

  Réponse : 5x5x3x32

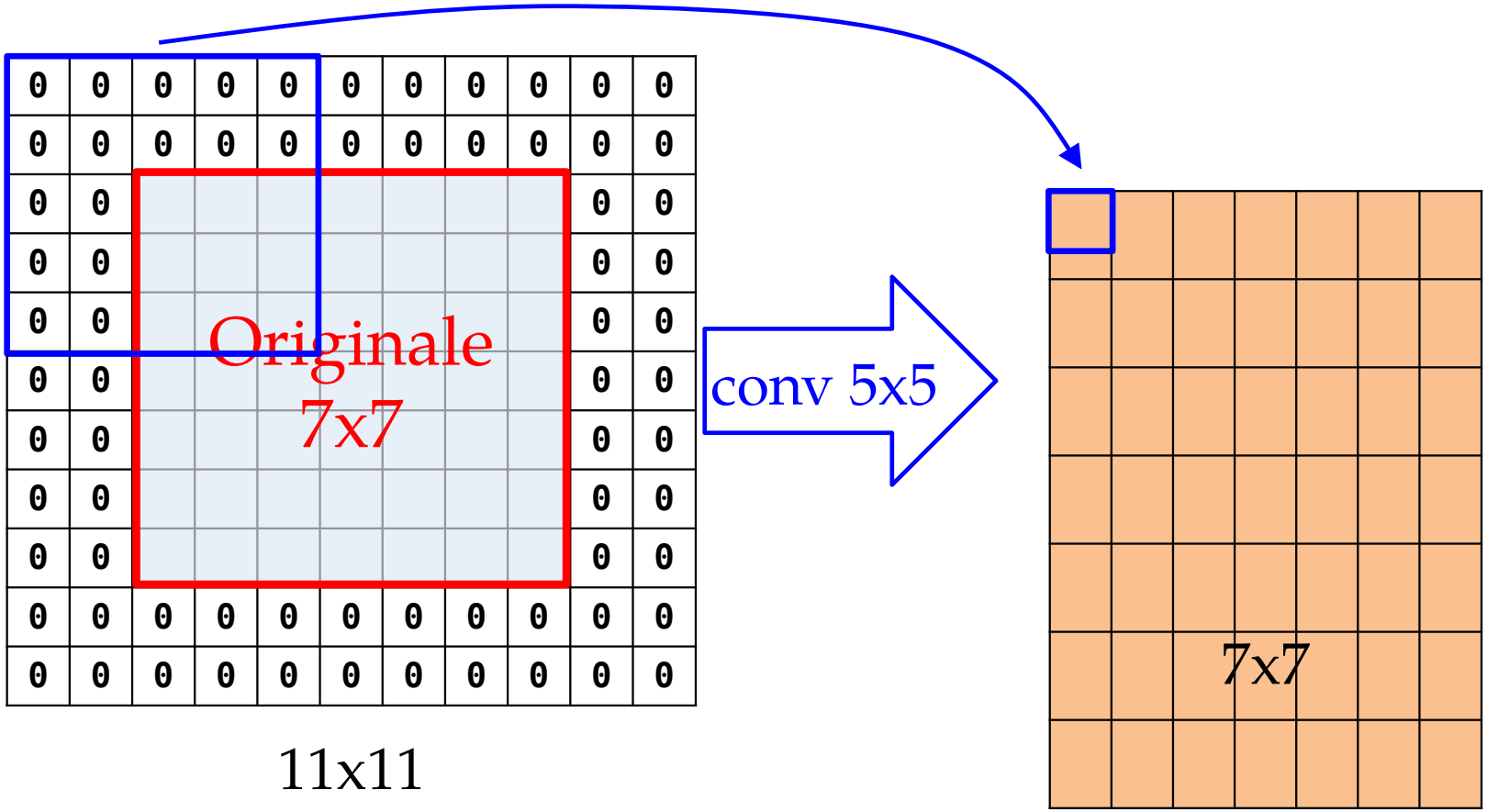
Ajout de zéros *(« zero padding »)*

- Ajoute des entrées à 0 en bordure



(par simplicité, j'ai retiré la 3ème dimension)

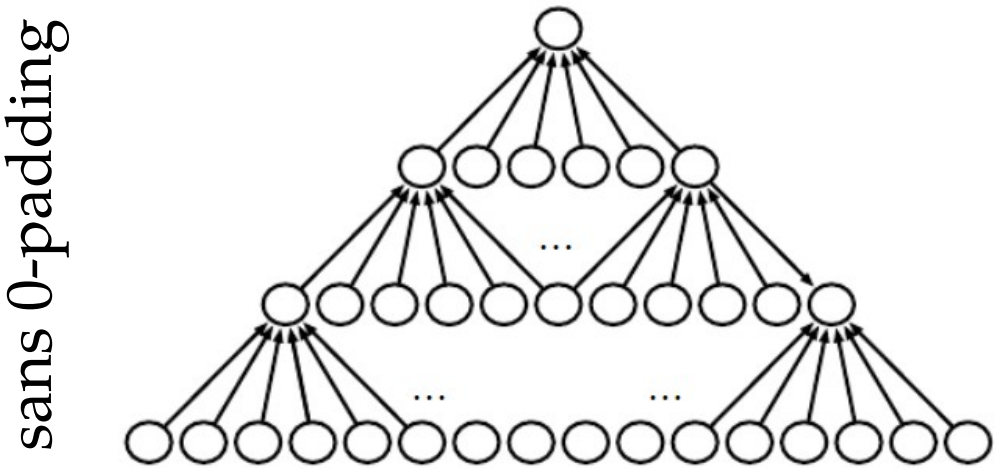
Ajout de zéros *(« zero padding »)*



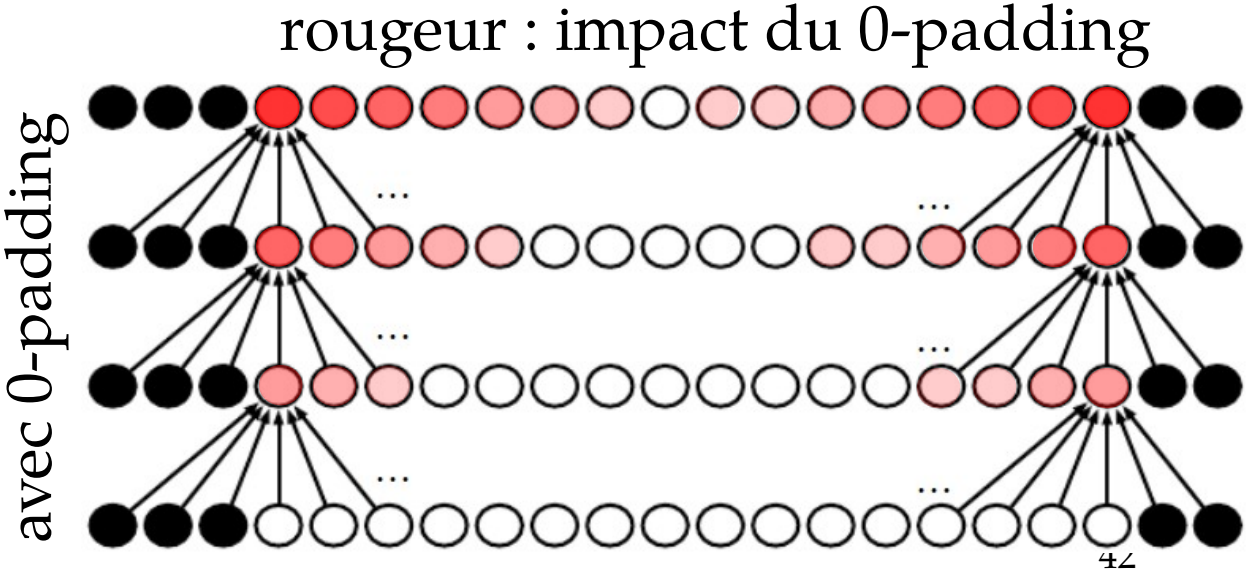
Pour filtre 3x3, padding de largeur 1
Pour filtre 5x5, padding de largeur 2
Pour filtre 7x7, padding de largeur 3

Ajout de zéros (« *zero padding* »)

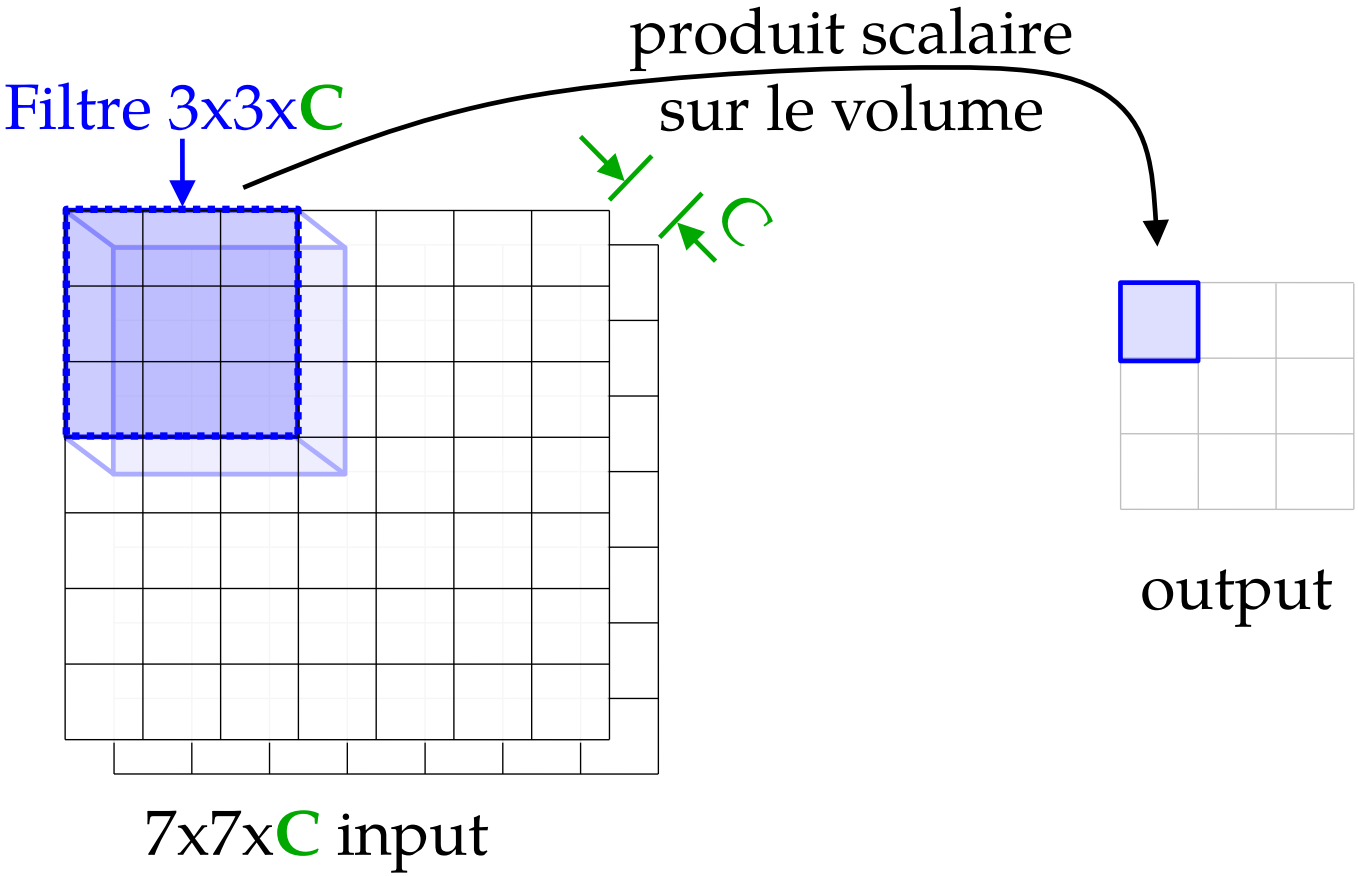
- Permet de conserver la largeur au fil des couches



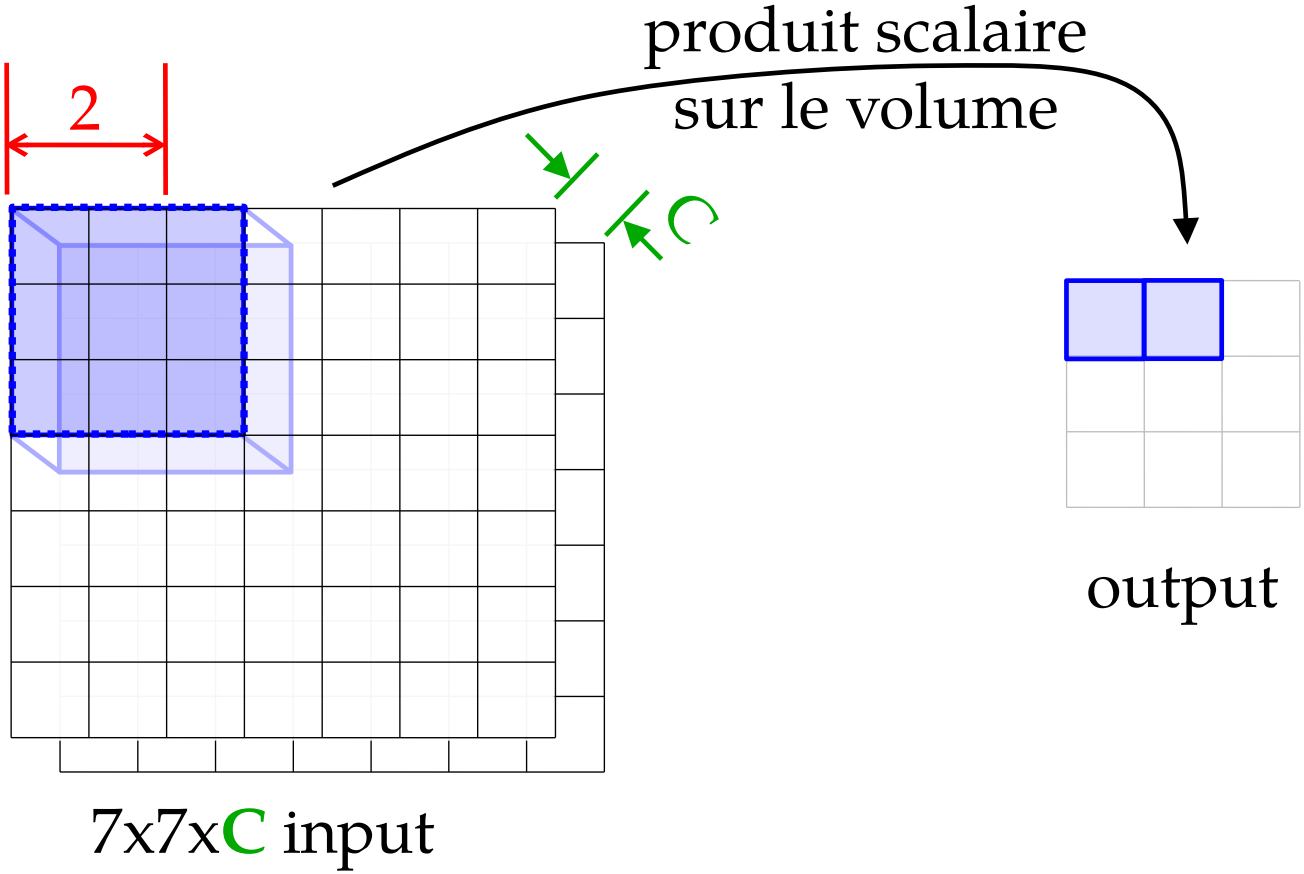
Impact incertain



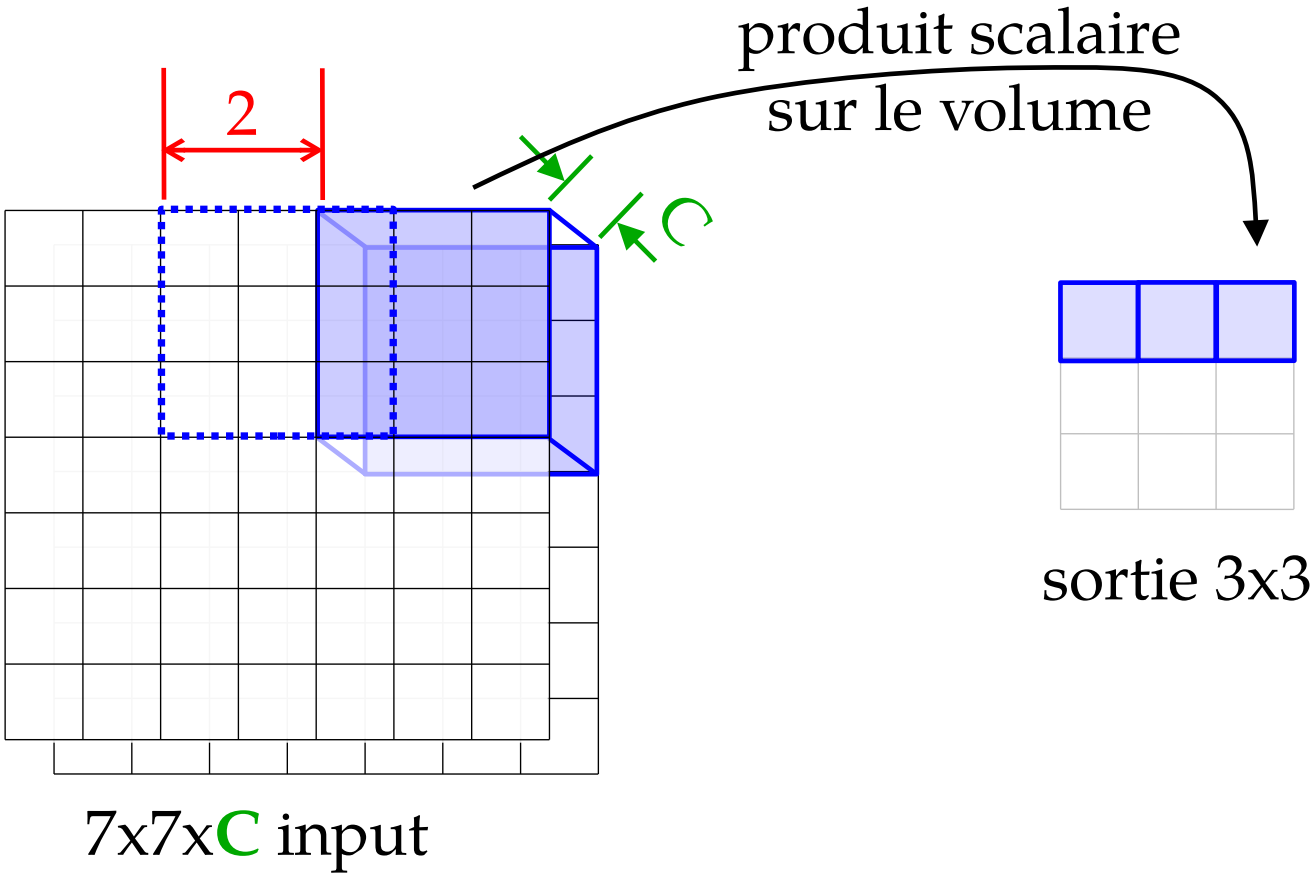
Pas du filtre (« *stride* »)



Pas du filtre (« *stride* »)



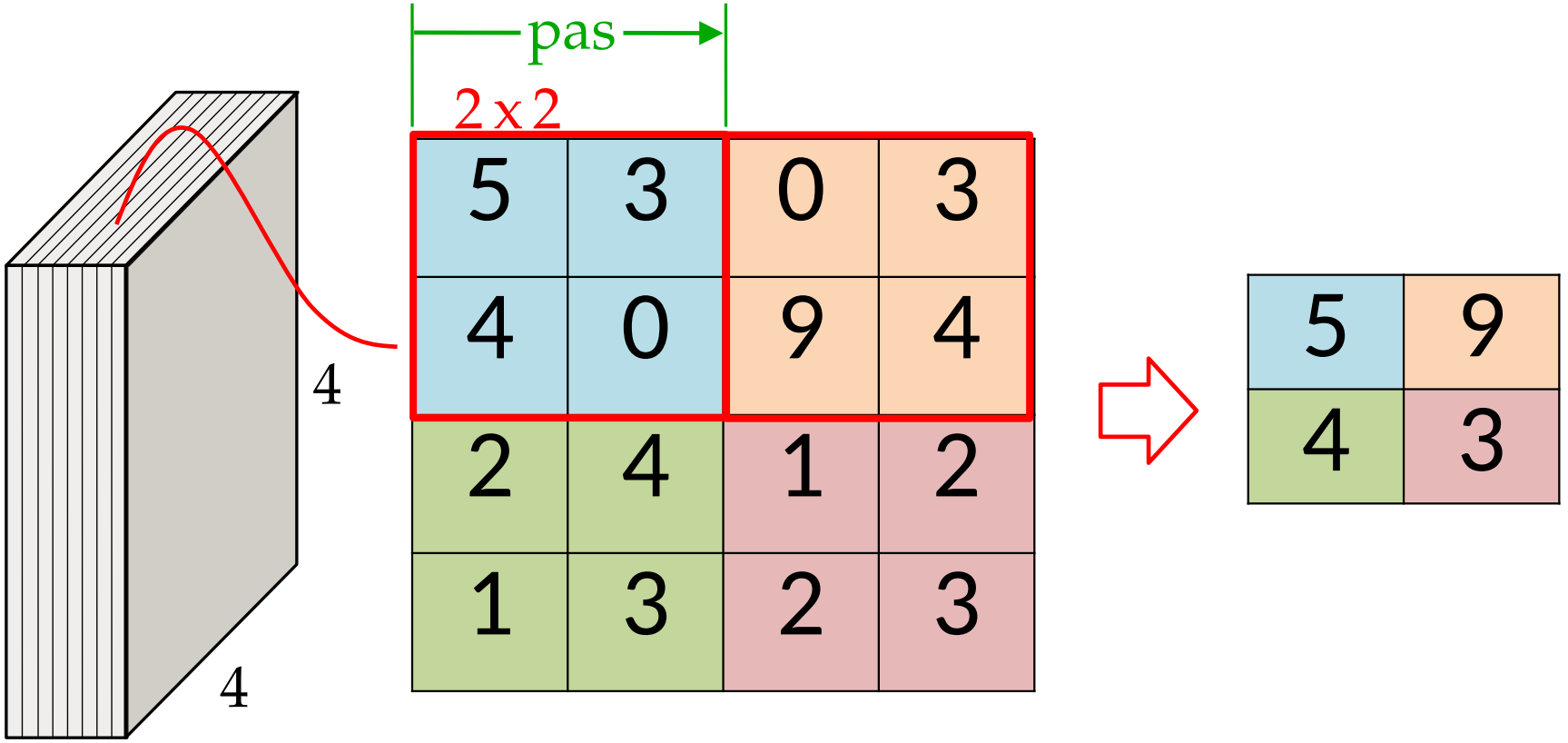
Pas du filtre (« *stride* »)



Couches de type « *Pooling* »

Max Pooling

- Appliqué pour chaque tranche, indépendamment



On doit spécifier le «pas»

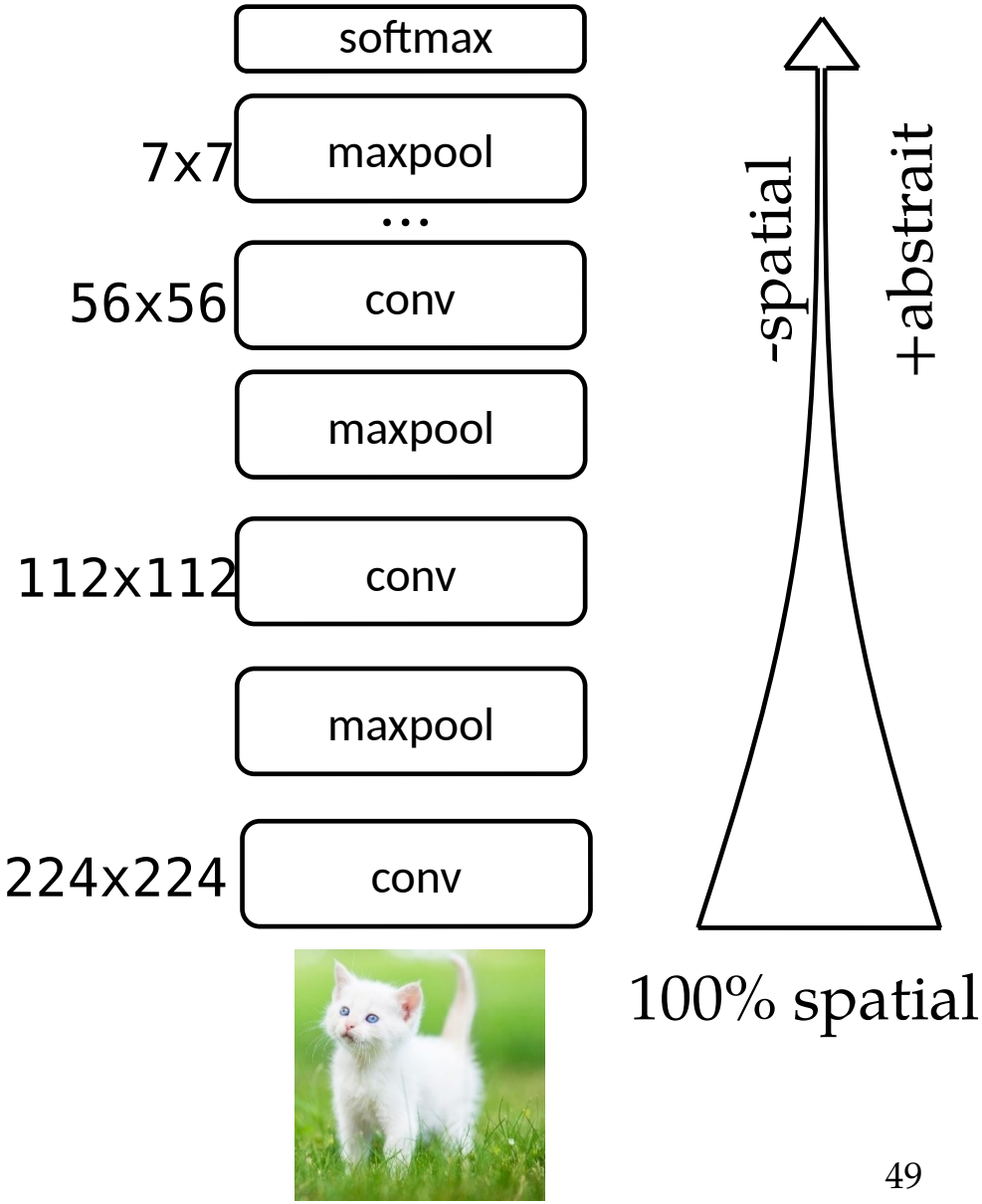
Max Pooling

- Réduit la dimension du *feature map*
- Souvent, on en profite pour augmenter le nombre de filtre
- Pourquoi ne pas faire plutôt une moyenne?
  - Le « Max Pooling » permet de détecter la présence d'un feature dans une région

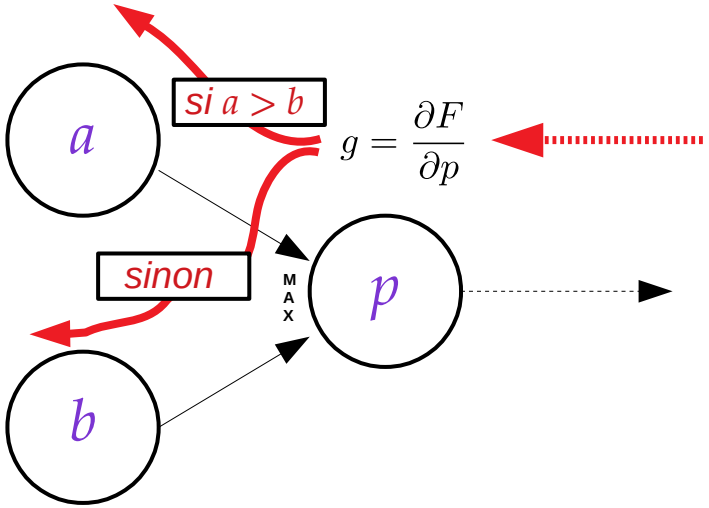
Pooling

- Augmente champ réceptif rapidement
- Réduit le nombre de paramètre
- Confère une certaine invariance aux transformations géométriques (rotation, translation, échelle)

Classe: aucune spatialité



Rétropropagation du gradient



$$p = \max(a, b) = \begin{cases} a & \text{si } a \geq b \\ b & \text{sinon.} \end{cases}$$

$$\frac{\partial p}{\partial a} = \begin{cases} 1 & \text{si } a \geq b \\ 0 & \text{sinon.} \end{cases}$$

$$\frac{\partial p}{\partial b} = \begin{cases} 0 & \text{si } a \geq b \\ 1 & \text{sinon} \end{cases}$$

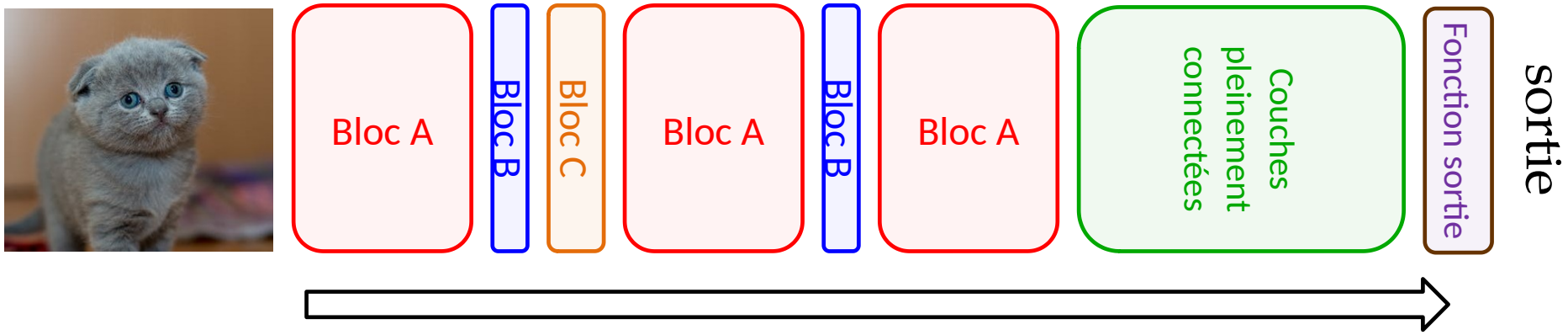
$$g(\mathbf{w}) = \max(\sigma(\mathbf{w})) = \max([\sigma(w_1), \sigma(w_2), \ldots, \sigma(w_d)])$$

$$\frac{\partial g(\mathbf{w})}{\partial w_k} = \begin{cases} \sigma'(w_k) & \text{si } k = \text{argmax}(\sigma(\mathbf{w})) \\ 0 & \text{si } k \neq \text{argmax}(\sigma(\mathbf{w})) \end{cases}$$

Architectures

Approche par blocs

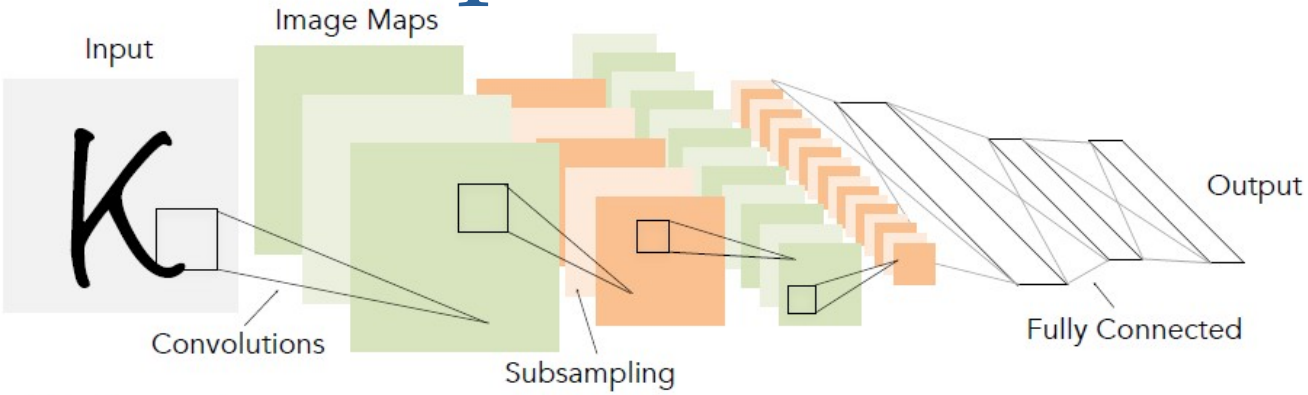
- La plupart des architectures sont organisées par alternance de blocs



- Facile de combiner des blocs de différents types, jouer sur la profondeur, réutiliser des idées

Ne date pas d'hier

1998
LeCun et al.



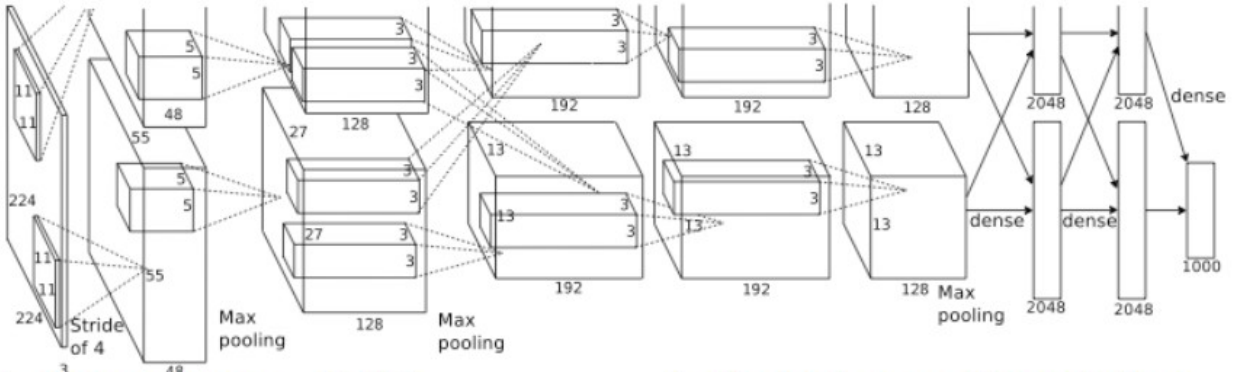
# of transistors

10^6

# of pixels used in training

10^7 NIST

2012
Krizhevsky et al.



# of transistors

10^9

GPUs

# of pixels used in training

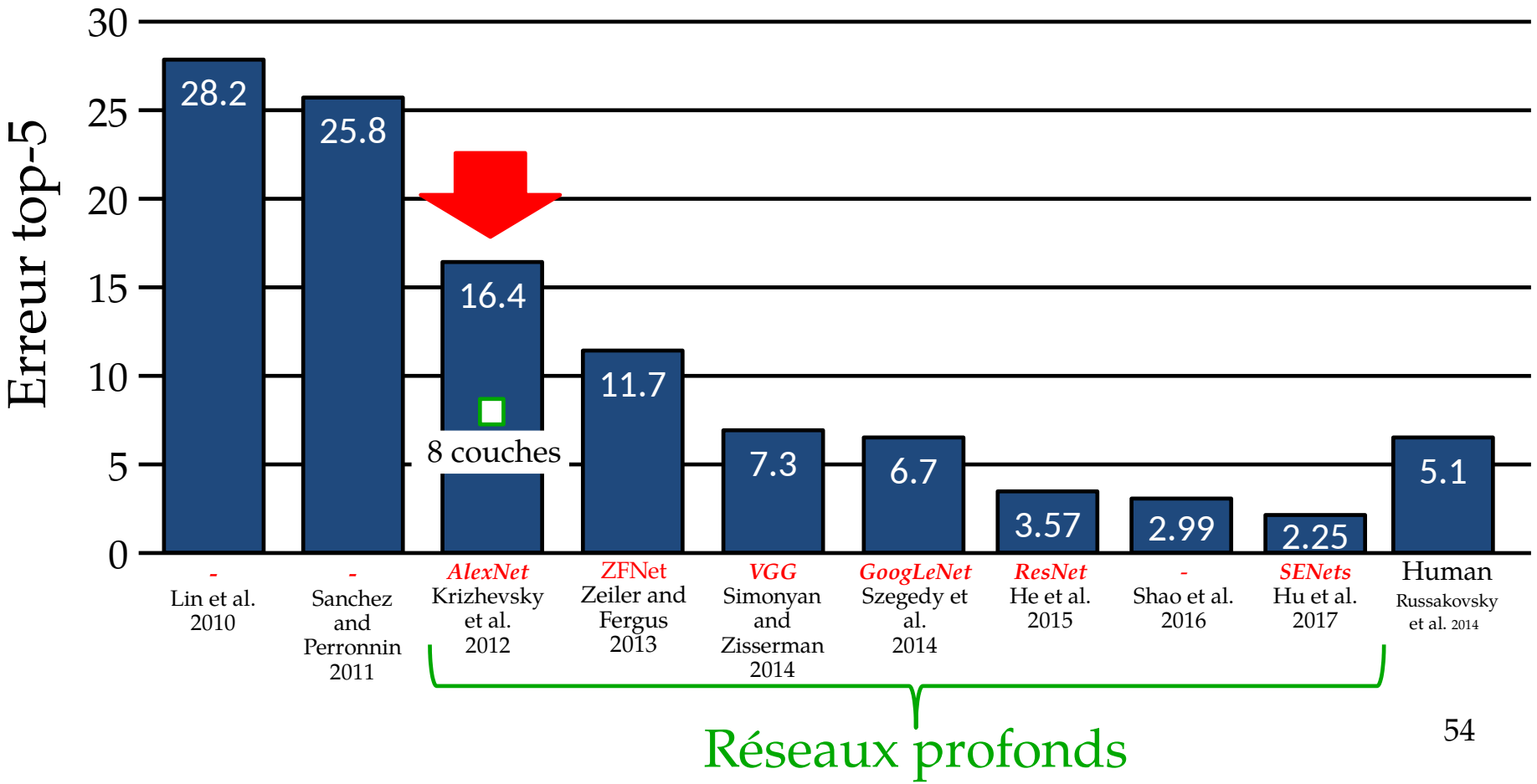
10^{14} IMAGENET

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

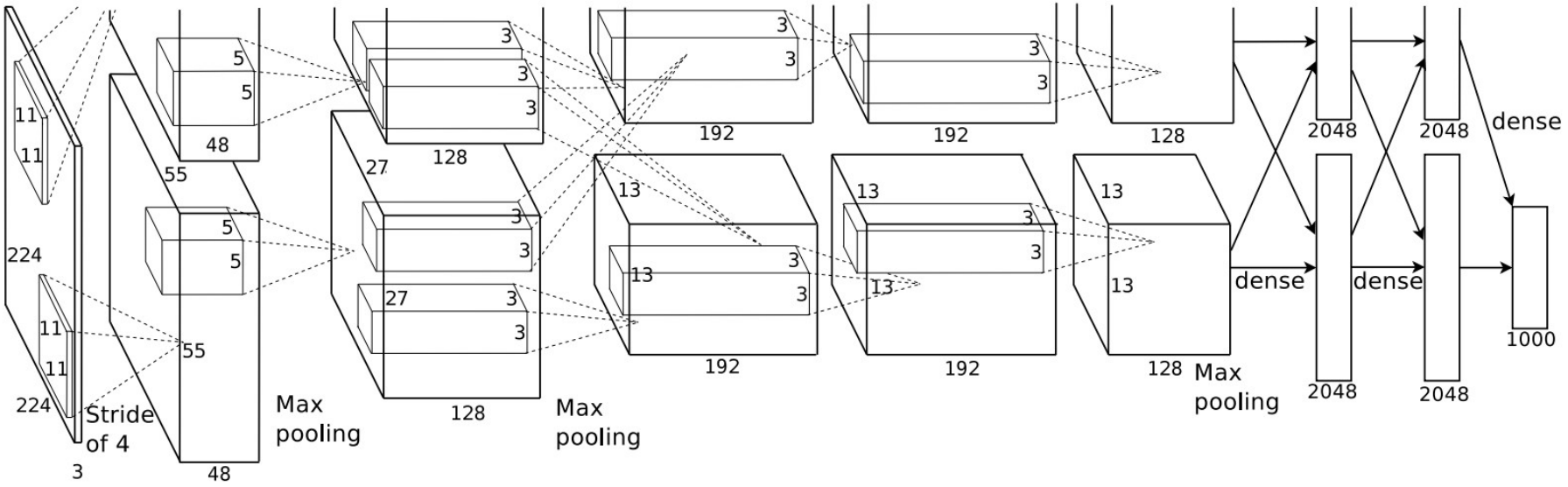
tiré de : cs231n, Université Stanford

Large Scale Visual Recognition Challenge

- *Image Classification Challenge* :
  - 1,000 classes d'objets
  - 1,431,167 images



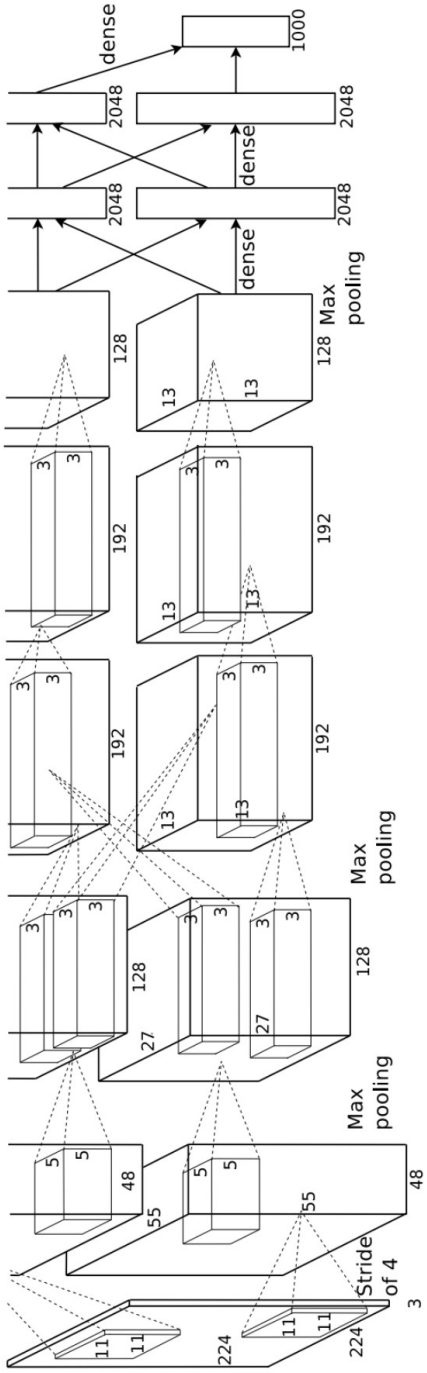
AlexNet



- 8 couches
- 60M paramètres
- Apparition des ReLU
- Dropout de 0.5
- Entraîné sur deux cartes GTX 580 (3 Go) en parallèle

adapté de : cs231n

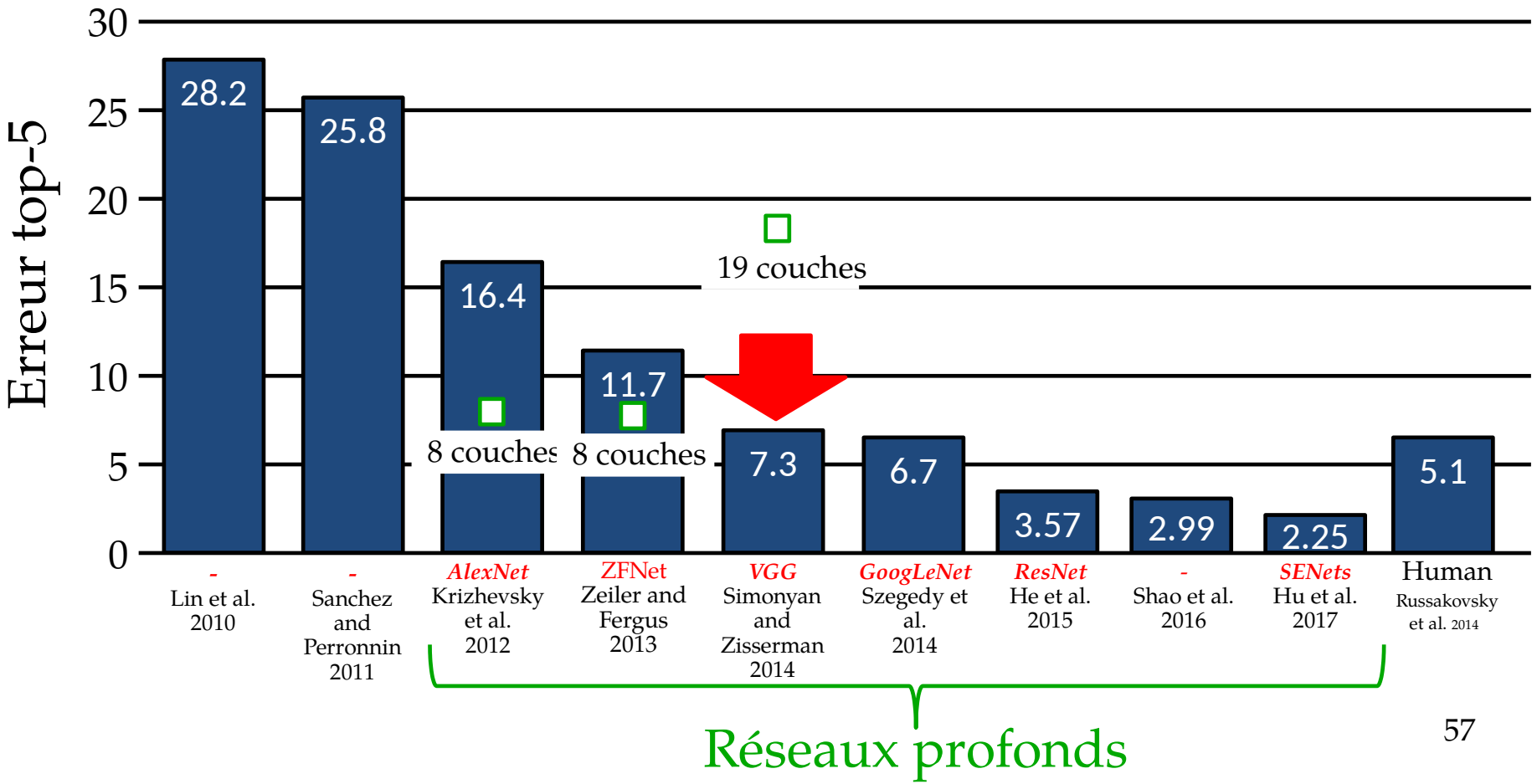
AlexNet



[1000] FC8: 1000 neurons (class scores)
[4096] FC7: 4096 neurons
[4096] FC6: 4096 neurons
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[13x13x256] CONV5: 256 **3x3** filters at stride 1, pad 1
[13x13x384] CONV4: 384 **3x3** filters at stride 1, pad 1
[13x13x384] CONV3: 384 **3x3** filters at stride 1, pad 1
[13x13x256] NORM2: Normalization layer
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[27x27x256] CONV2: 256 **5x5** filters at stride 1, pad 2
[27x27x96] NORM1: Normalization layer
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[55x55x96] CONV1: 96 **11x11** filters at stride 4, pad 0
[227x227x3] INPUT

adapté de : cs231n

Large Scale Visual Recognition Challenge



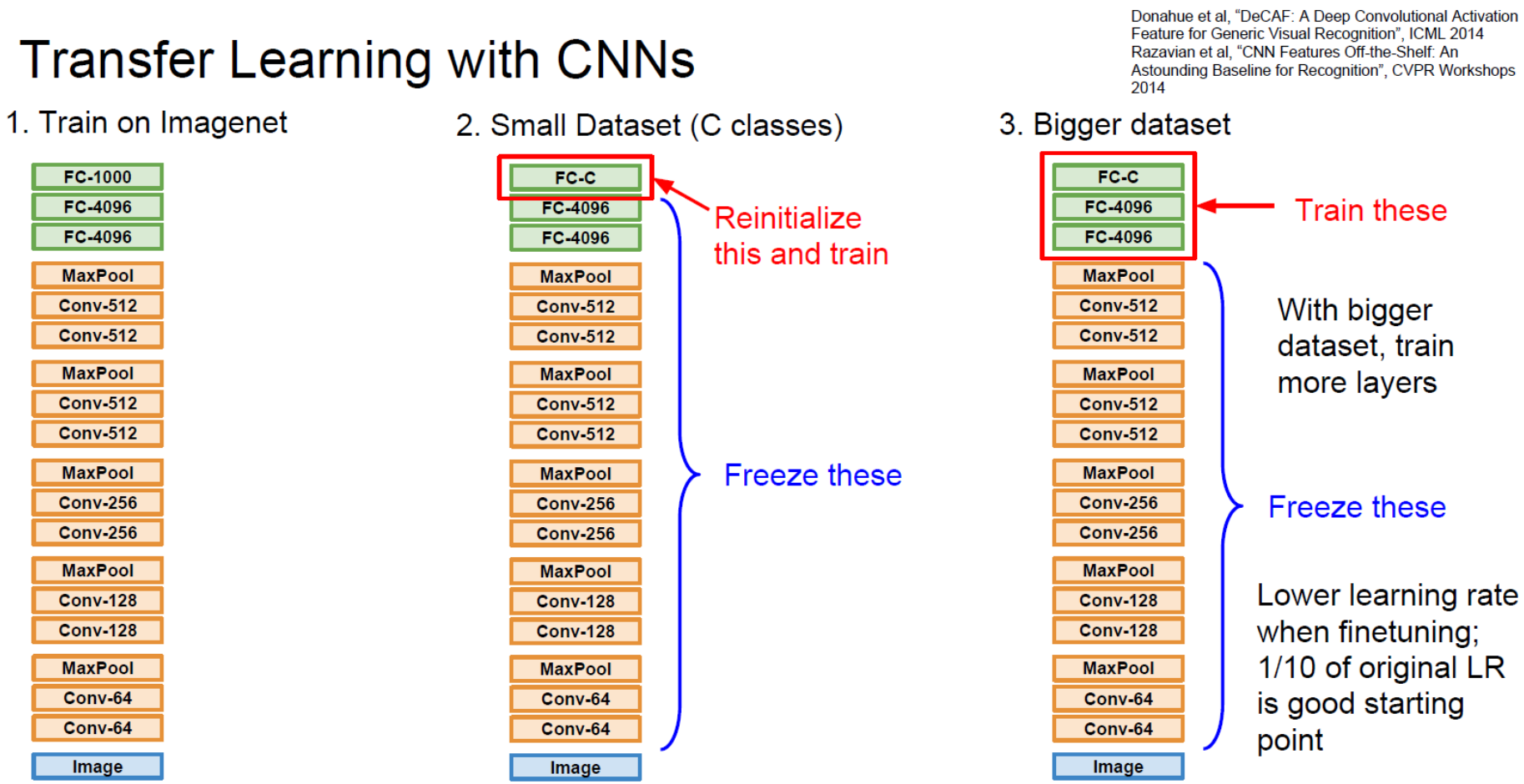
VGGNet

- Toujours 3 couches pleinement connectées
- 16-19 couches
- ~130 millions de paramètres
- Que des convolutions 3x3

adapté de : cs231n

« Finetuning »

Permet le transfert de l'apprentissage vers une tâche similaire



tiré de : cs231n