UNIVERSITÉ LAVAL

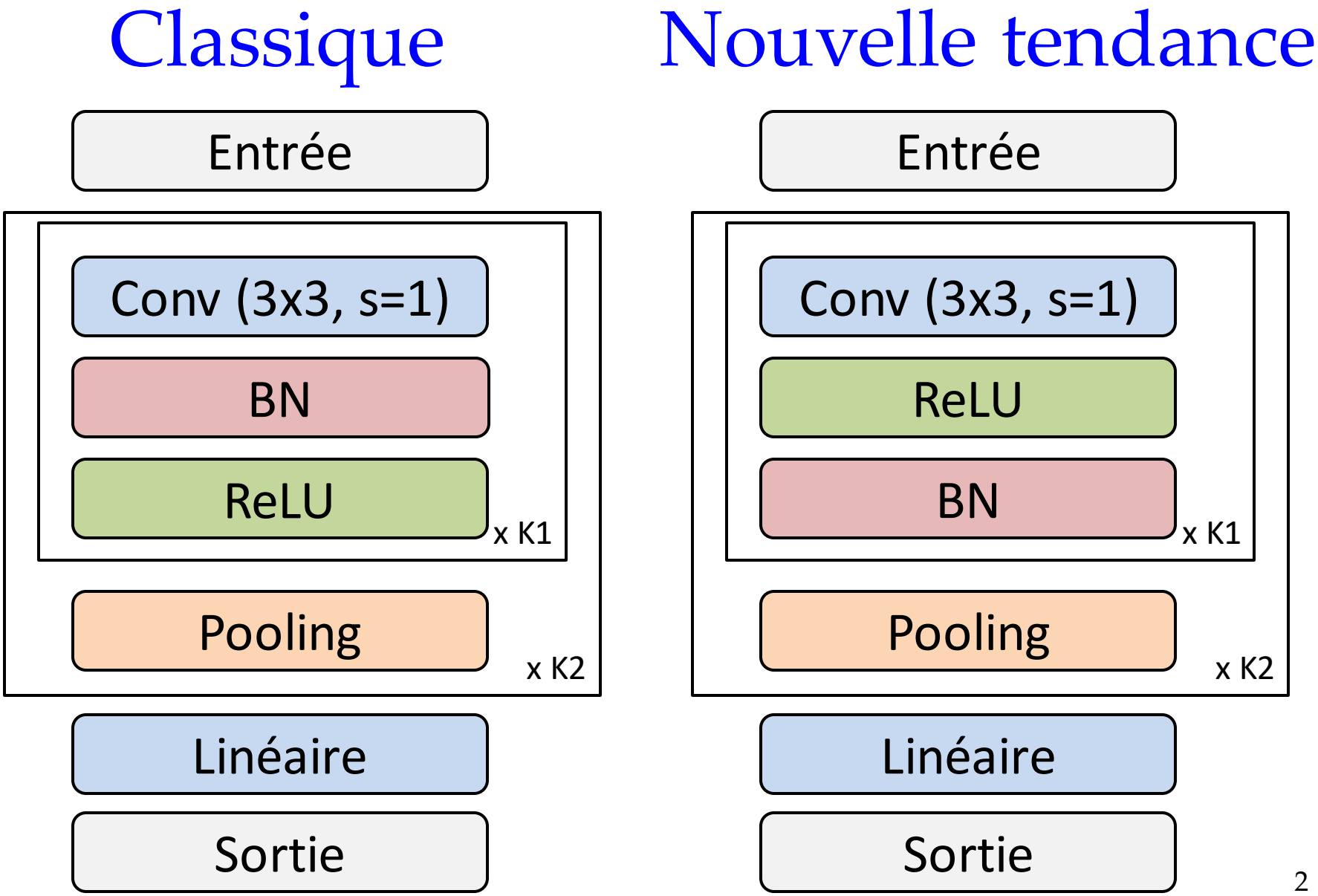
GLO-4030/7030
APPRENTISSAGE PAR RÉSEAUX DE NEURONES PROFONDS

Convolutional Neural Networks CNN

Partie I

Philippe Giguère

Position de la batch norm

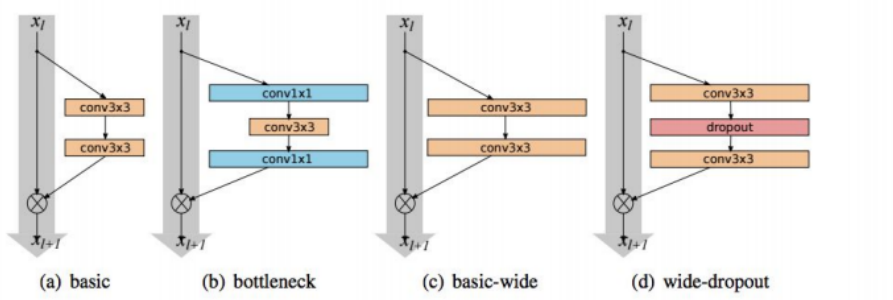


Introduction

- L'un des plus grand success-story du *Deep*
- Responsable (en partie) de la renaissance
- Bel exemple d'injection de *prior* via l'architecture
- Voir de nombreuses architectures, transposables à d'autres applications

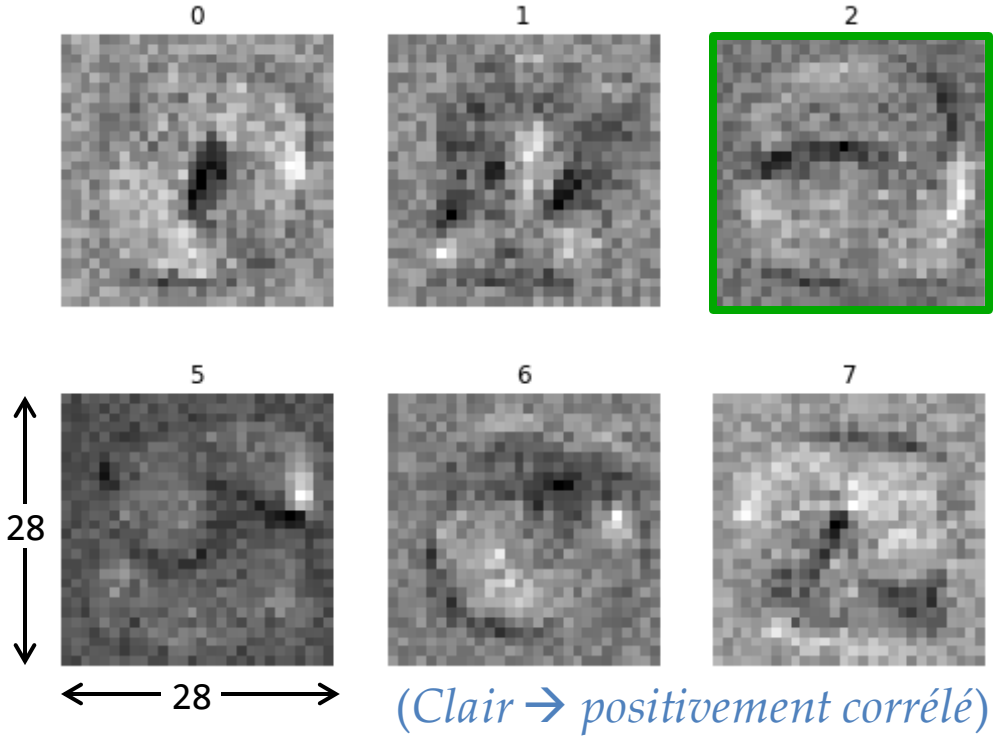
ResNet tweaks: Wide ResNets

- **Use pre-activation ResNet's basic block with more feature maps**
- Used parameter "k" to encode width
- Investigated relationship between width and depth to find a good tradeoff



Sergey Zagoruyko, Nikos Komodakis
Wide Residual Networks

Rappel : réseau 1 couche MNIST

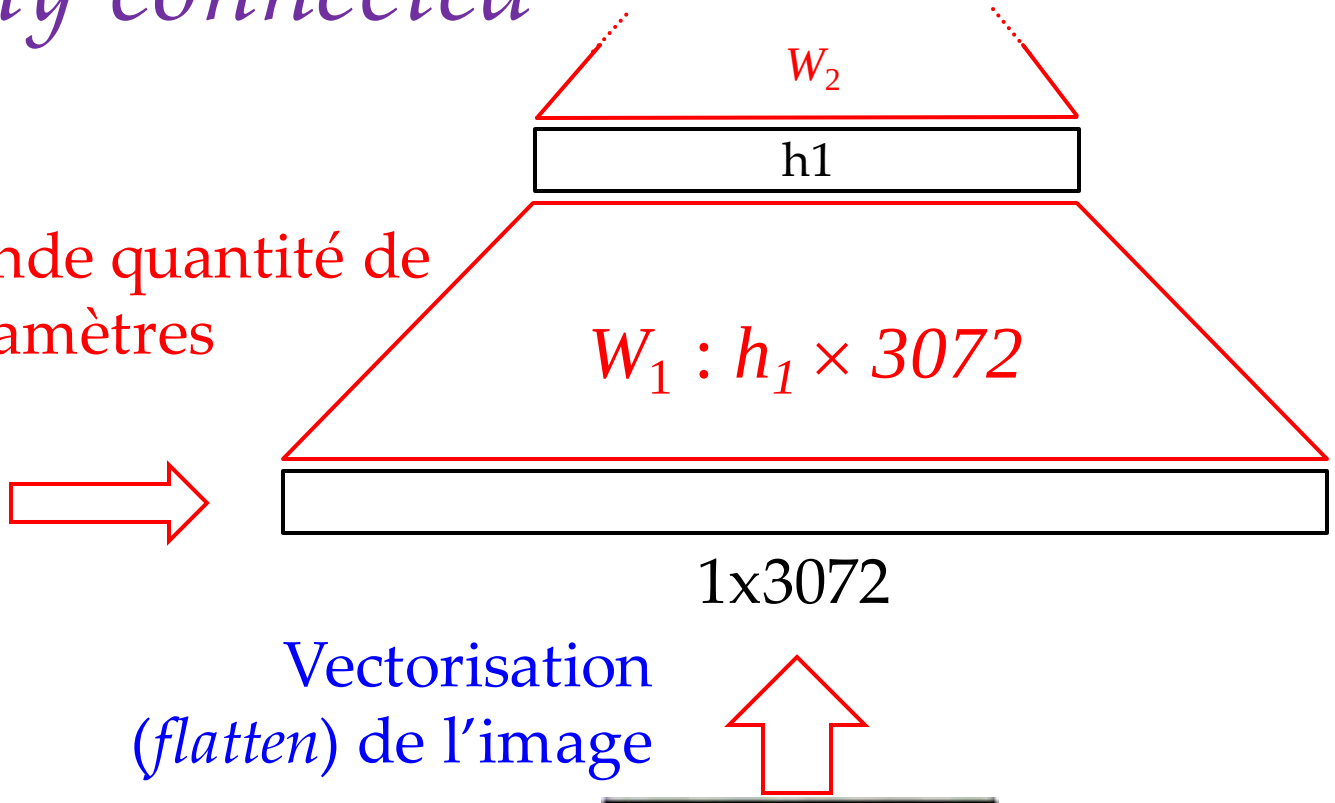


Patron == filtre

(*Clair → positivement corrélé*)

- On voit corrélation spatiale sur pixels voisins
- Réseau doit la découvrir : doit utiliser + d'exemples d'entraînement (*statistical efficiency*)

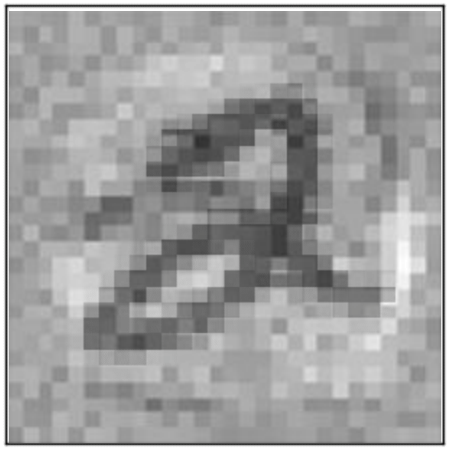
Réseau *fully connected*



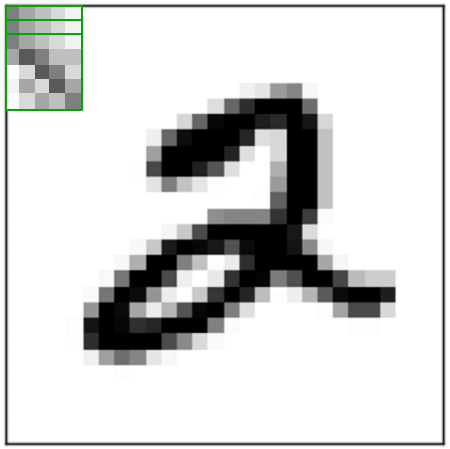
- Vectorisation détruit :
  - relations spatiales
  - canaux de couleurs

Filtre

Multiplier entrée-par-entrée, 1 fois



Multiplier entrée-par-entrée, plusieurs fois



Convolution $*$

- Opération mathématique très utilisée :
  - Traitement de signal
  - Probabilités (somme de 2 variables aléatoires)
  - Modélisation de systèmes via réponse impulsionnelle (GMC, GEL)

$$(I * F)(i, j) = (F * I)(i, j) = \sum_m \sum_n F(m,n) I(i-m, j-n)$$

commutative

- Au sens strict, les réseaux utilisent plutôt la corrélation croisée $\star$

$$(F \star I)(i, j) = \sum_m \sum_n F(m,n) I(i+m, j+n)$$

+ au lieu de -

Exemple « convolution »

(qui est plus une corrélation croisée, mais bon…)

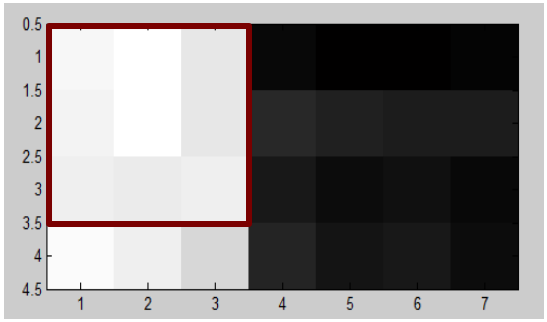
(Appellation de Filtre, *Filter*, ou *Kernel*)

$$F_1 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

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

$(I*F_1)(x,y)$

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



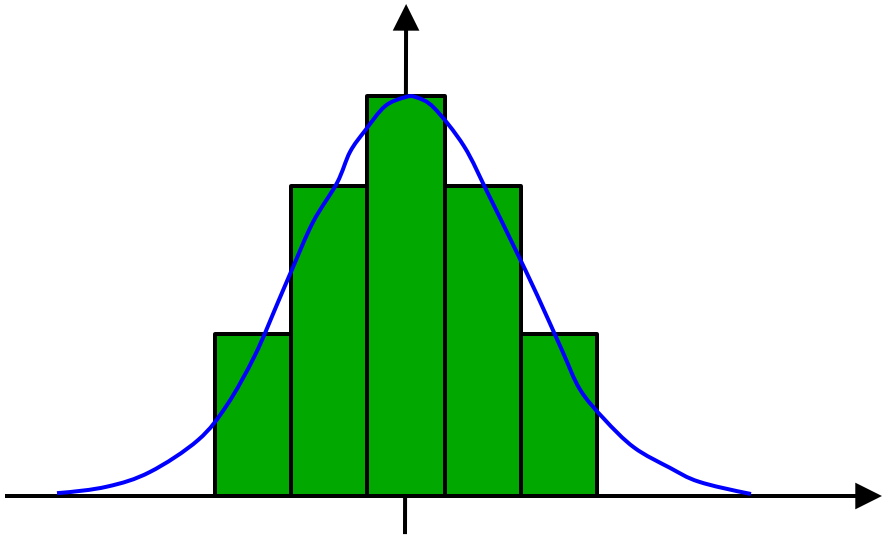
Exemple convolution

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

Merci à FAST.ai pour le lien

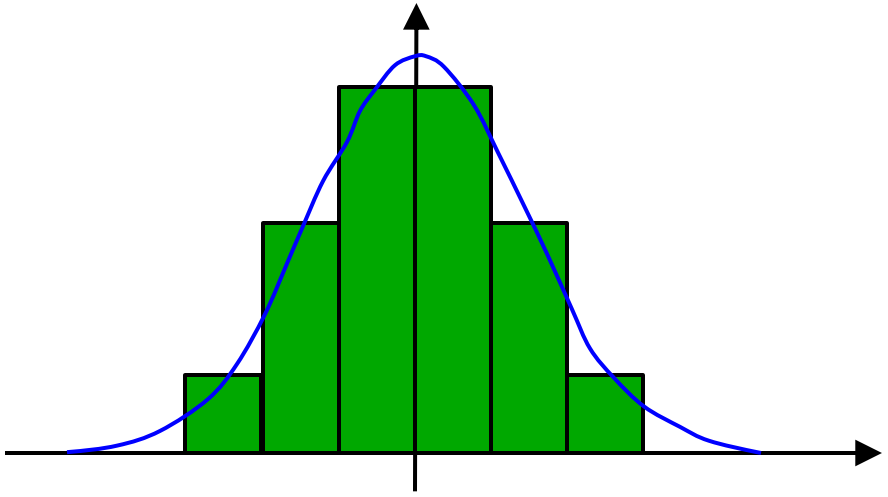
Pourquoi filtre taille impaire

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



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 (aliasing)

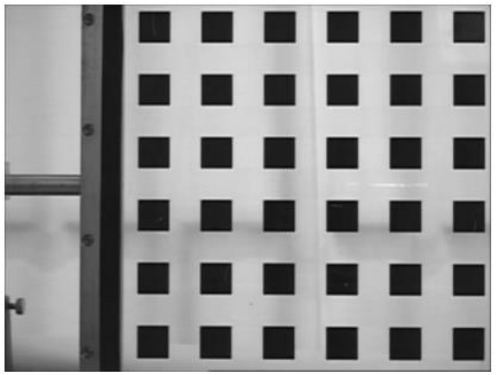
(à titre informatif)

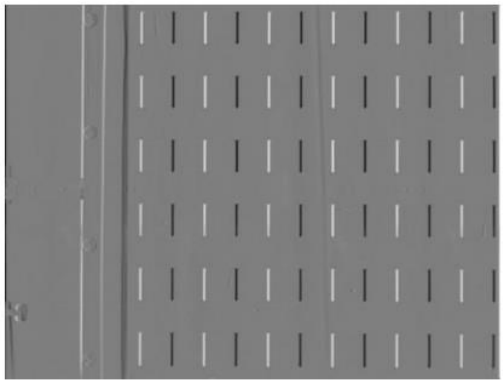
Exemples filtres *hand-tuned*

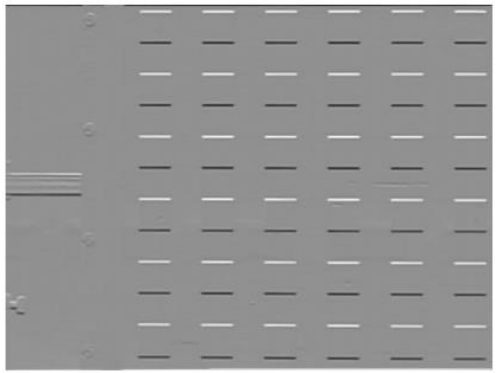
Détection bordure

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

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

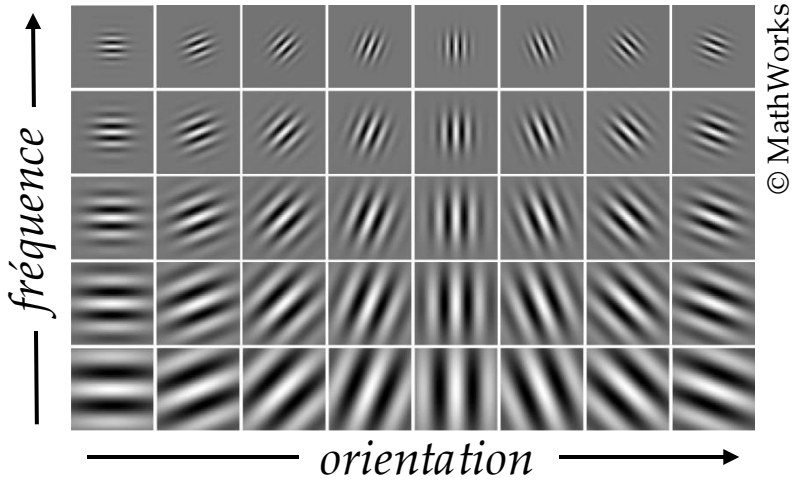
5x5

15x15

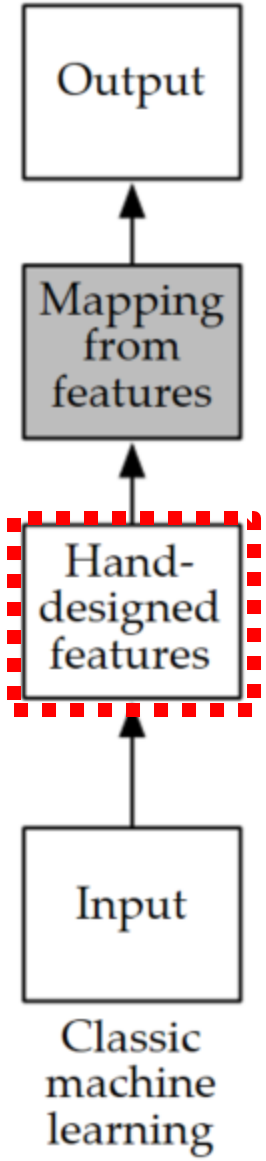
Adapté du cours de Design III (P. Hébert, D. Laurendeau)

Filtres

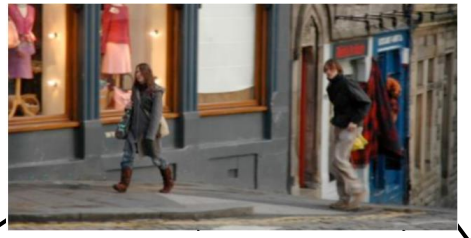
- Vont extraire des *features* de bas niveau
  - Filtres *Gabor* :



  - Filtres de bordure, ondelettes
- Longtemps été un domaine de recherche
  - que concevriez-vous comme filtre pour MNIST ?
- Comme les filtres CNN sont différentiables, le réseau pourra les modifier à sa convenance
  - les ajuster pour maximiser les performances sur les données d'entraînement



DPM



feature map

feature map 2x résolution

root filter

part filter 1

part filter *n*

réponse du filtre

réponse du filtre

réponse du filtre

tenant compte du coût de déformation

Algorithme **explicite**

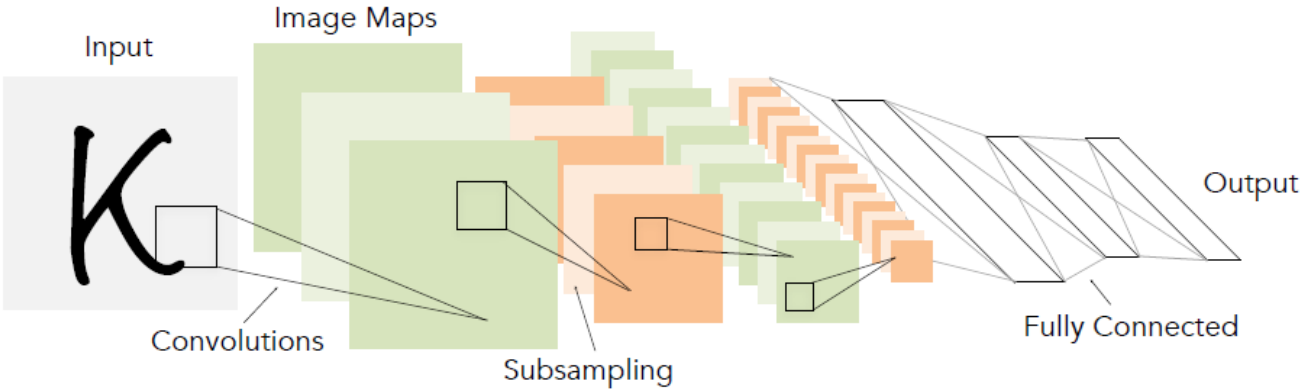
Adapté de P. Felzenszwalb

Injection de *prior* dans les CNN

- Forte corrélation locale dans les valeurs des pixels (champs réceptif)
  - structure 2D
- Régularisation par *weight sharing*
- Former des *features* de manière hiérarchique, de plus en plus abstraits

Ne date pas d'hier

1998
LeCun et al.



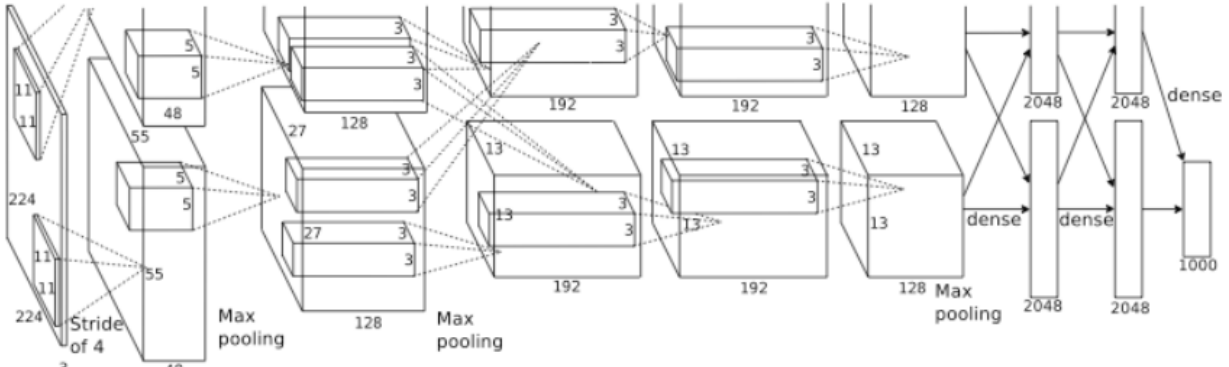
# of transistors

10^6

# of pixels used in training

10^7 NIST

2012
Krizhevsky et al.



# of transistors

10^9

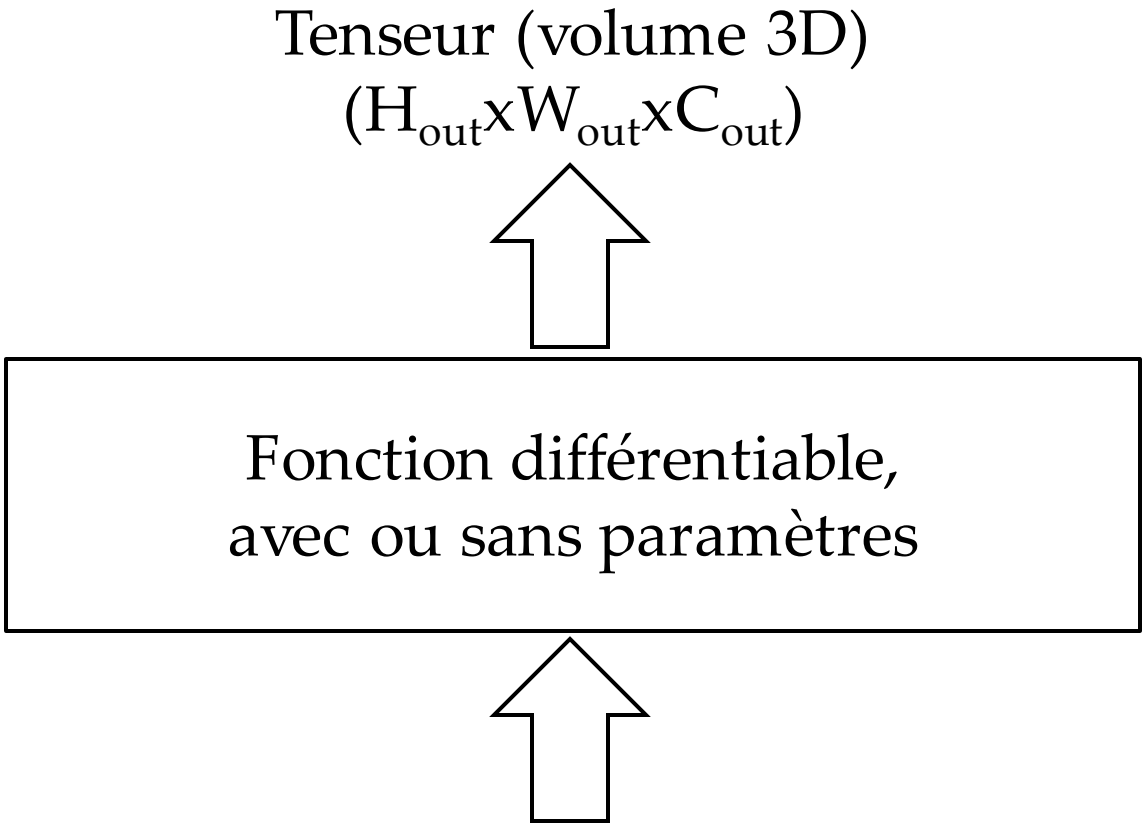
GPUs

# of pixels used in training

10^{14} IMAGENET

tiré de : cs231n, Université Stanford

CNN : couche typique

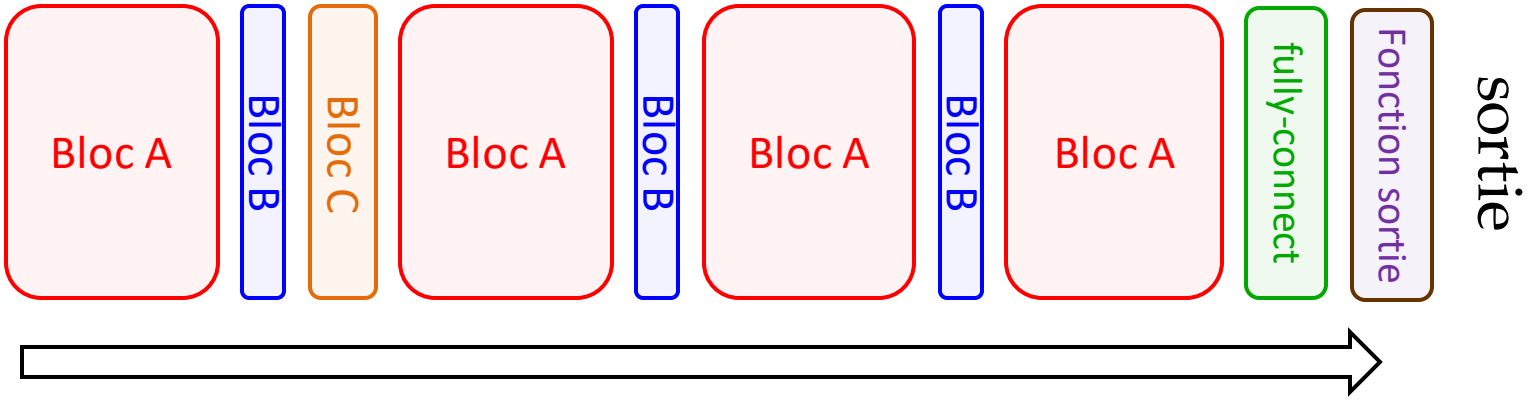


Principaux types de couche

- *Fully-Connected* (vous connaissez déjà)
- Convolutive
- Pooling
  - Max
  - Average et global average
  - Stochastic
  - Fractional

Approche par bloc

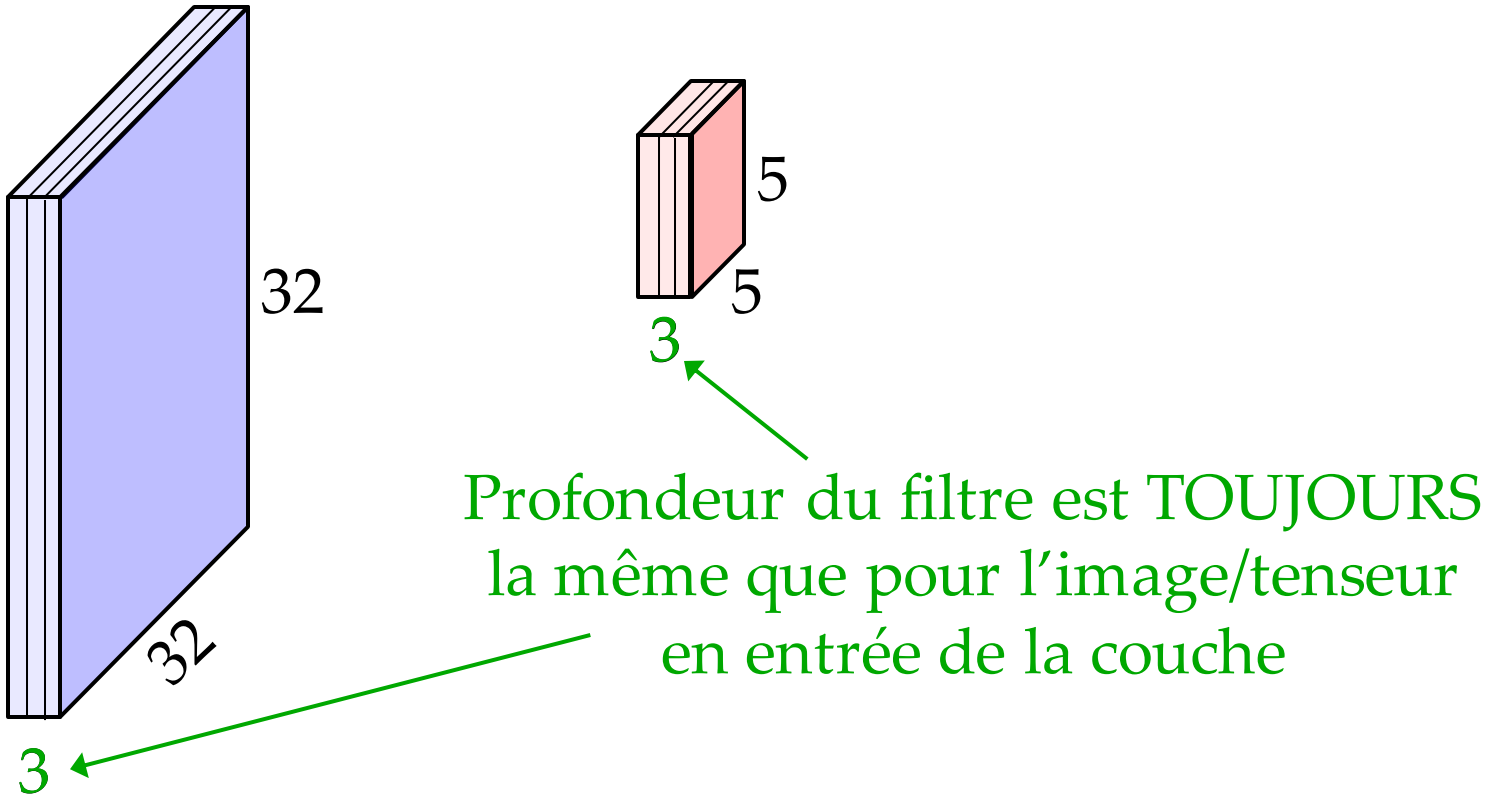
- La plupart des architectures sont organisées par alternance de blocs (pas de spaghetti)



- Facile de combiner des blocs de différents types, jouer sur la profondeur, réutiliser des idées
- Choix des blocs est en quelque sorte des *hyperparamètres* : trop difficile de parfaitement optimiser, donc on se restreint à un agencement limité

Filtres convolutifs

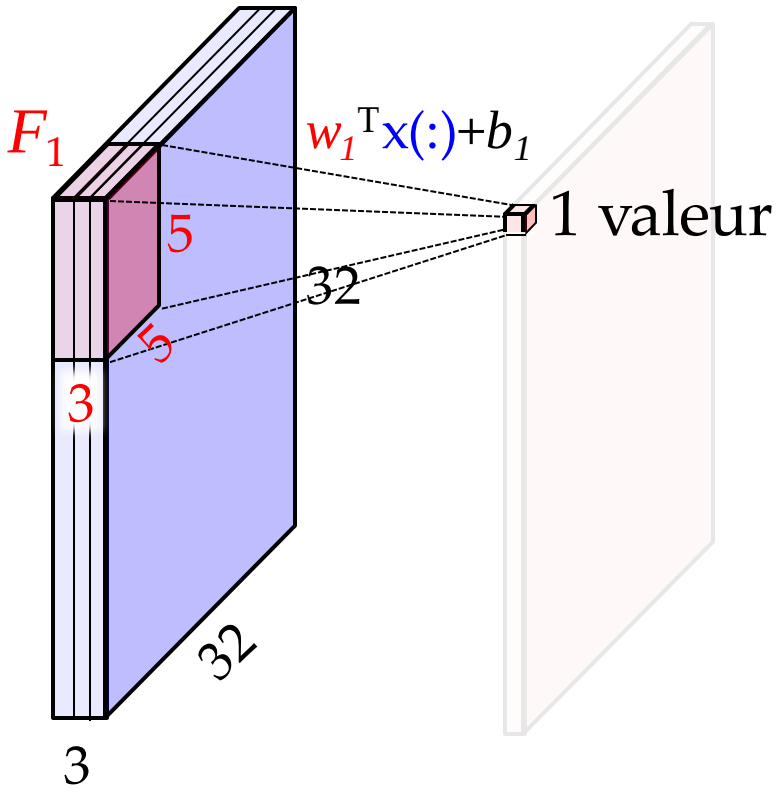
- Conserver la structure spatiale/couleur/*feature* de l'entrée



adapté de : cs231n, Université Stanford

Filtres convolutifs

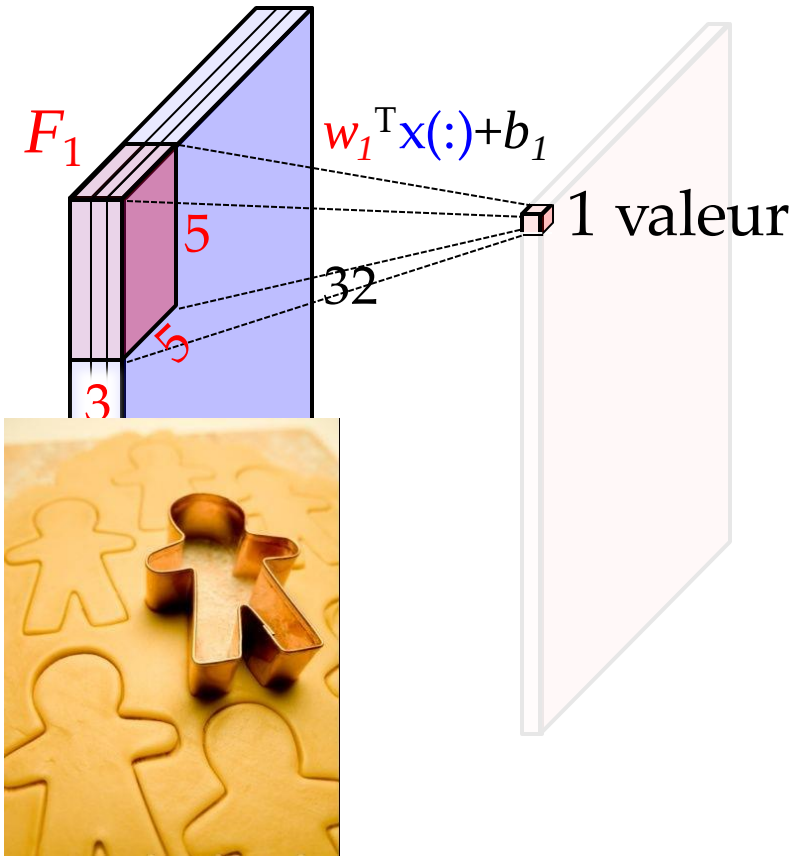
- Glisser spatialement le filtre F sur l'image, en calculant produit scalaire à chaque endroit de x



adapté de : cs231n, Université Stanford

Filtres convolutifs

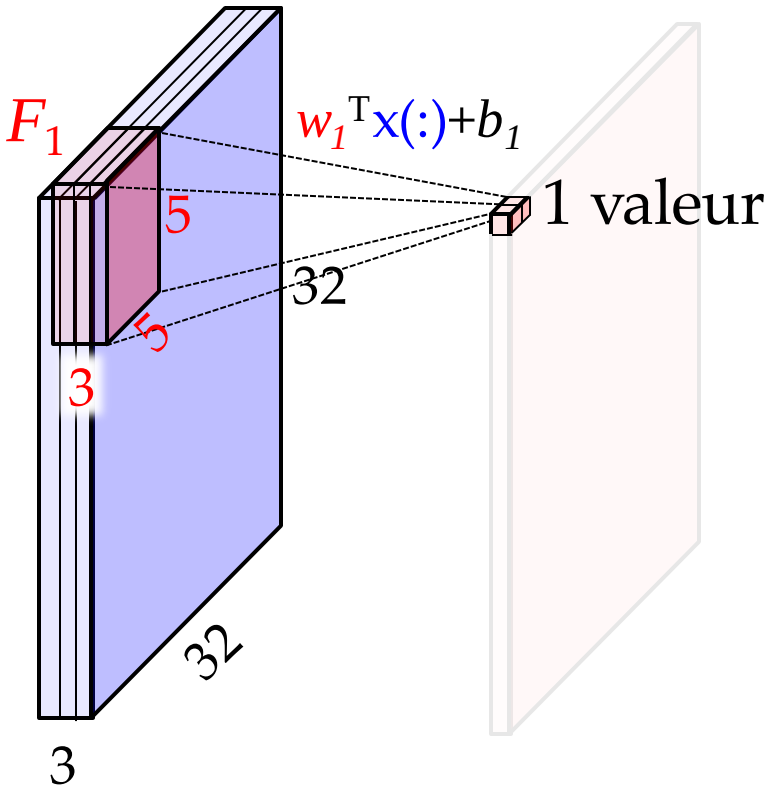
- Glisser spatialement le filtre F sur l'image, en calculant produit scalaire à chaque endroit de x



adapté de : cs231n, Université Stanford

Filtres convolutifs

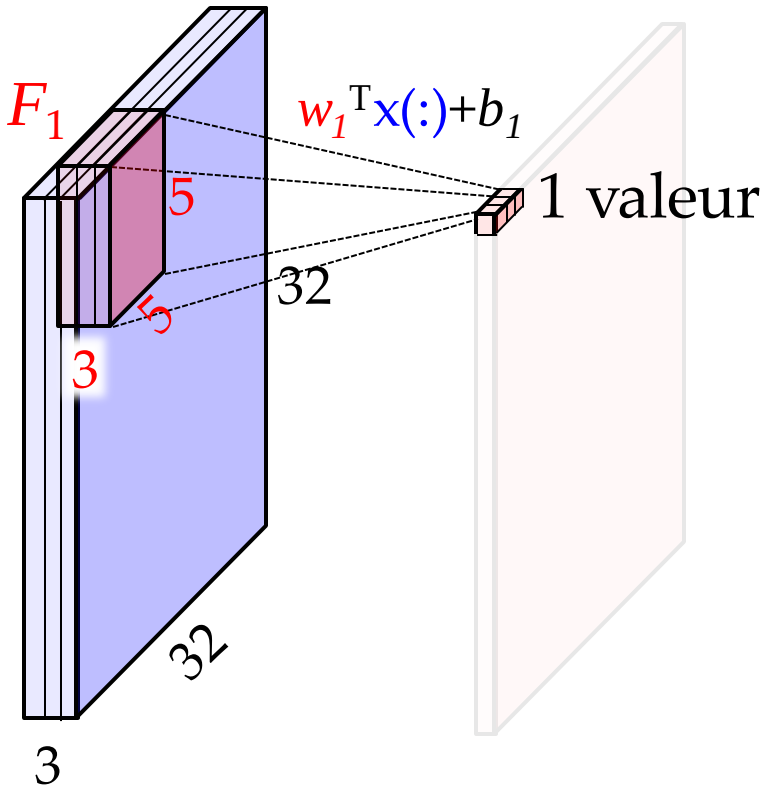
- Glisser spatialement le filtre F sur l'image, en calculant produit scalaire à chaque endroit de x



adapté de : cs231n, Université Stanford

Filtres convolutifs

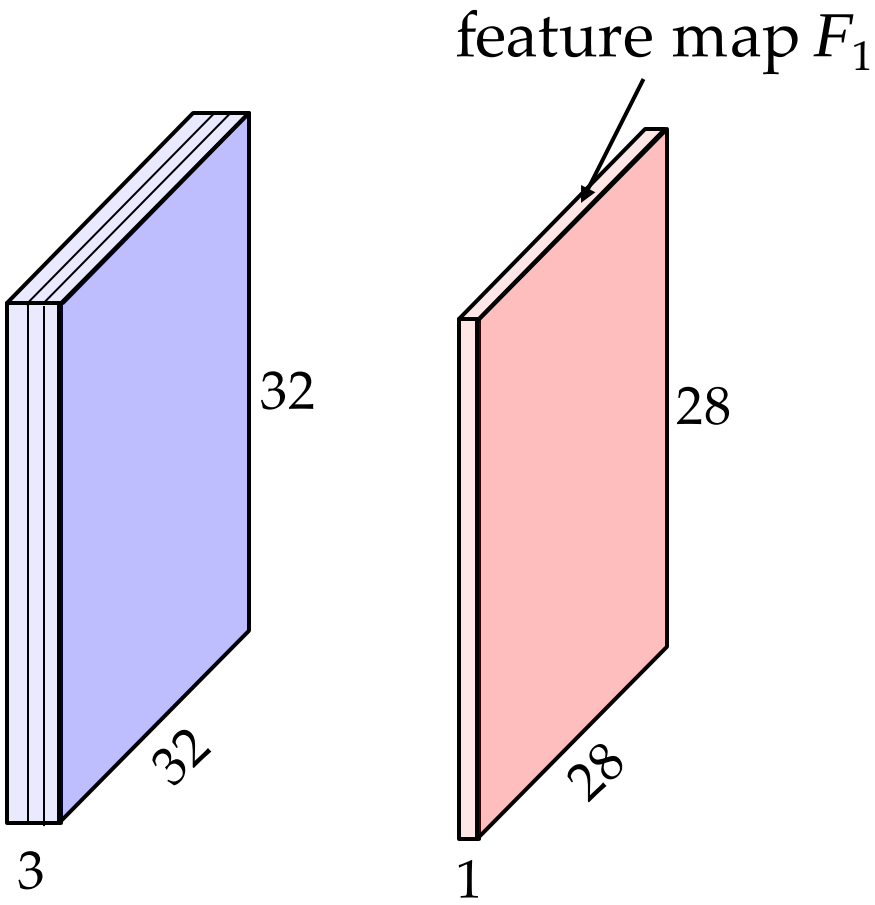
- Glisser spatialement le filtre F sur l'image, en calculant produit scalaire à chaque endroit de x



adapté de : cs231n, Université Stanford

Filtres convolutifs

- Sortie : *feature map*

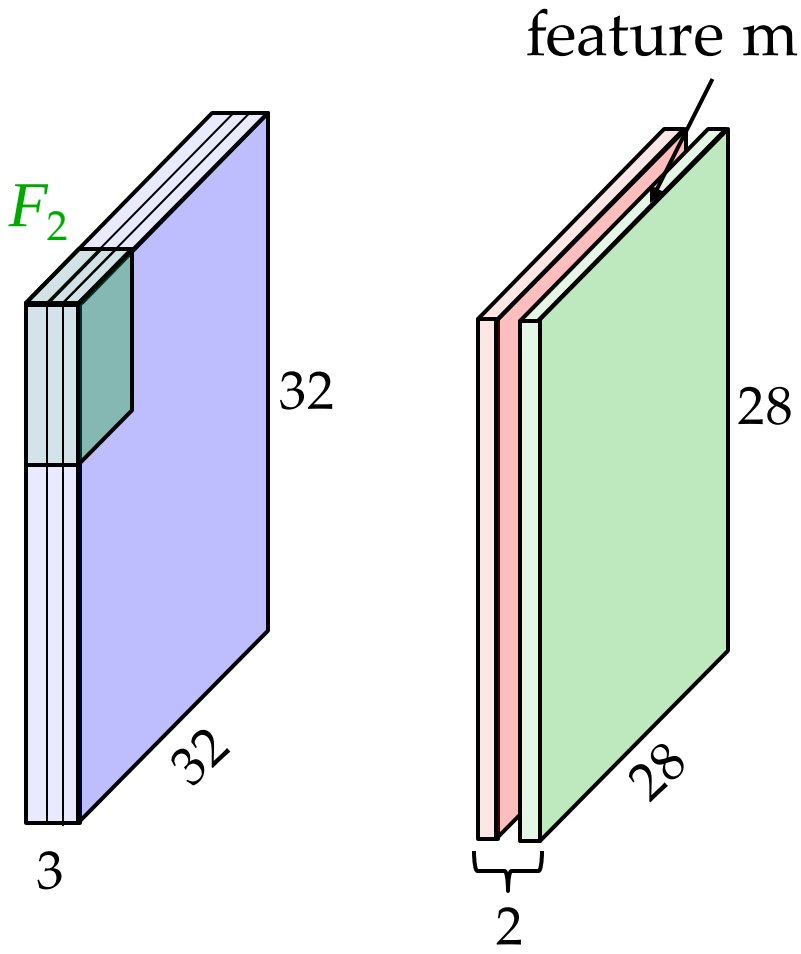


« combien présent est le feature F_1 à cet endroit? »

adapté de : cs231n, Université Stanford

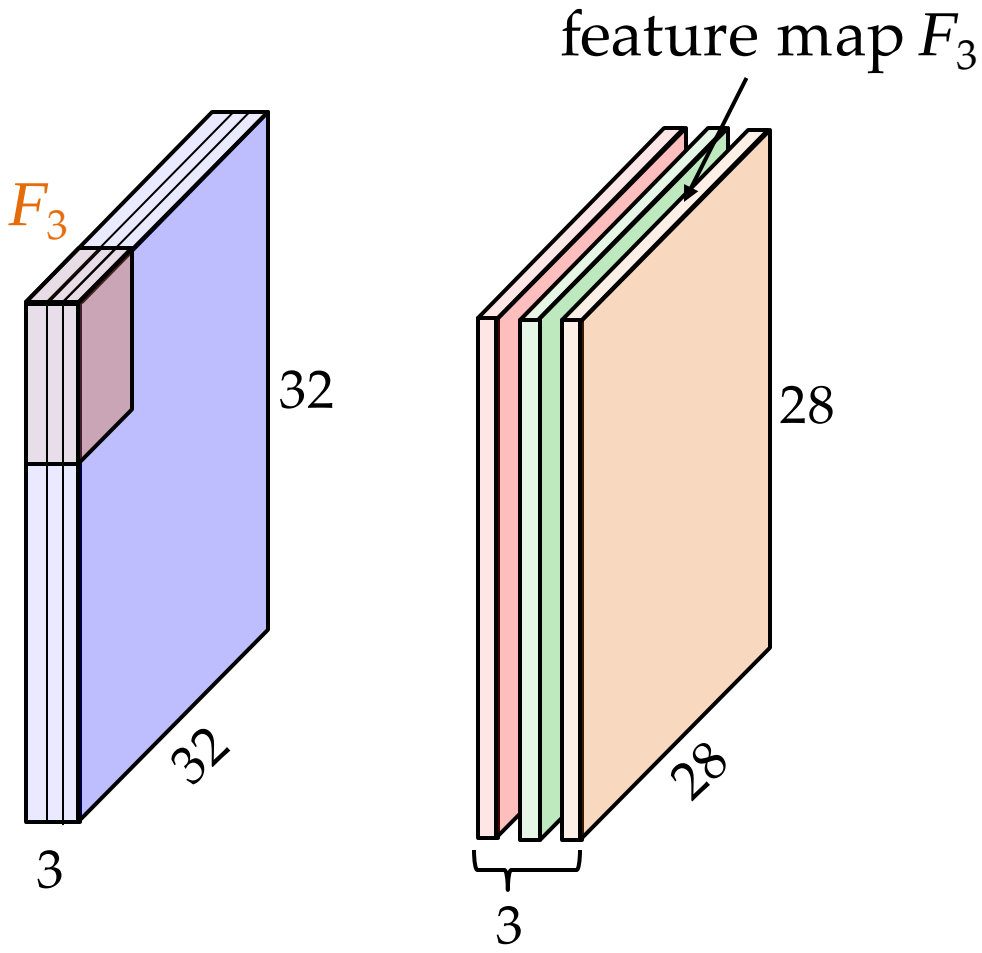
(Note : la taille de sortie est différente ici…)

Banque de filtres convolutifs



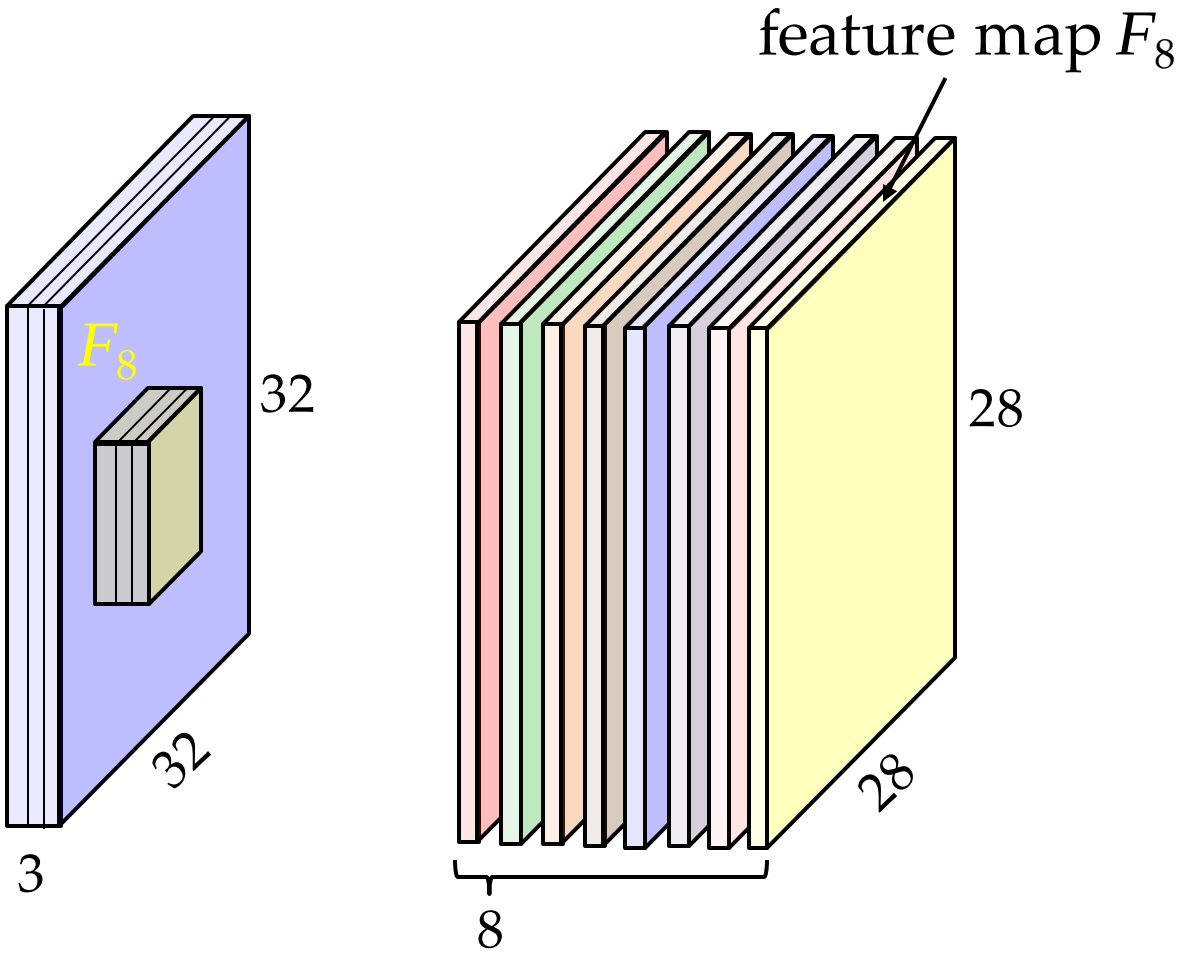
adapté de : cs231n, Université Stanford

Banque de filtres convolutifs



adapté de : cs231n, Université Stanford

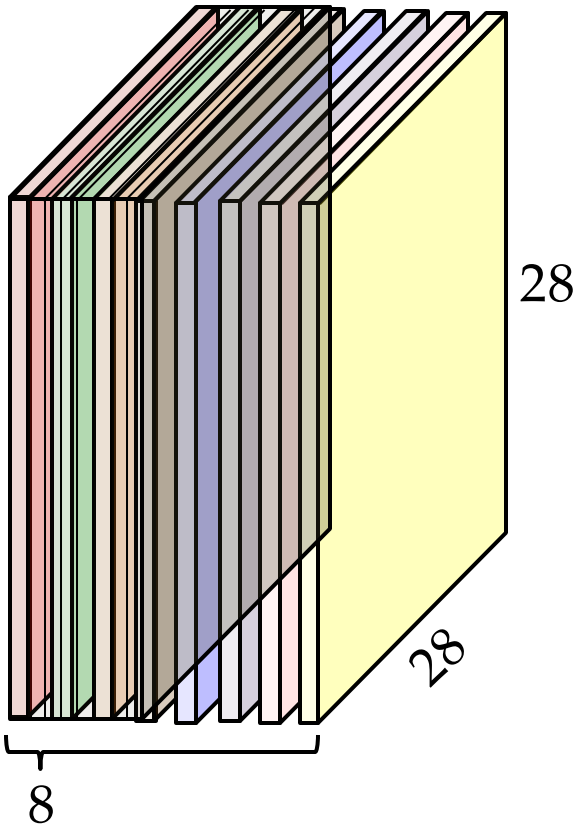
Banque de filtres convolutifs



adapté de : cs231n, Université Stanford

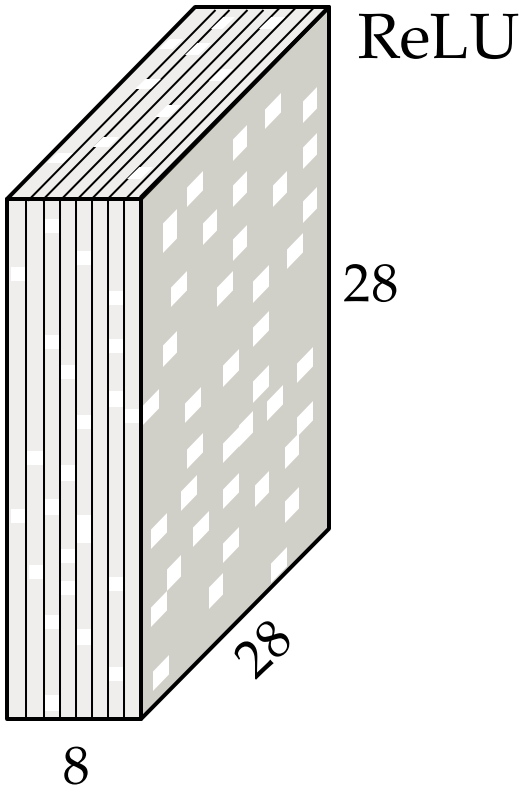
Résultante

Tenseur ordre 3

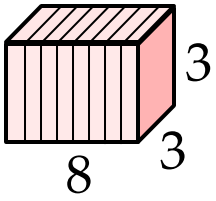


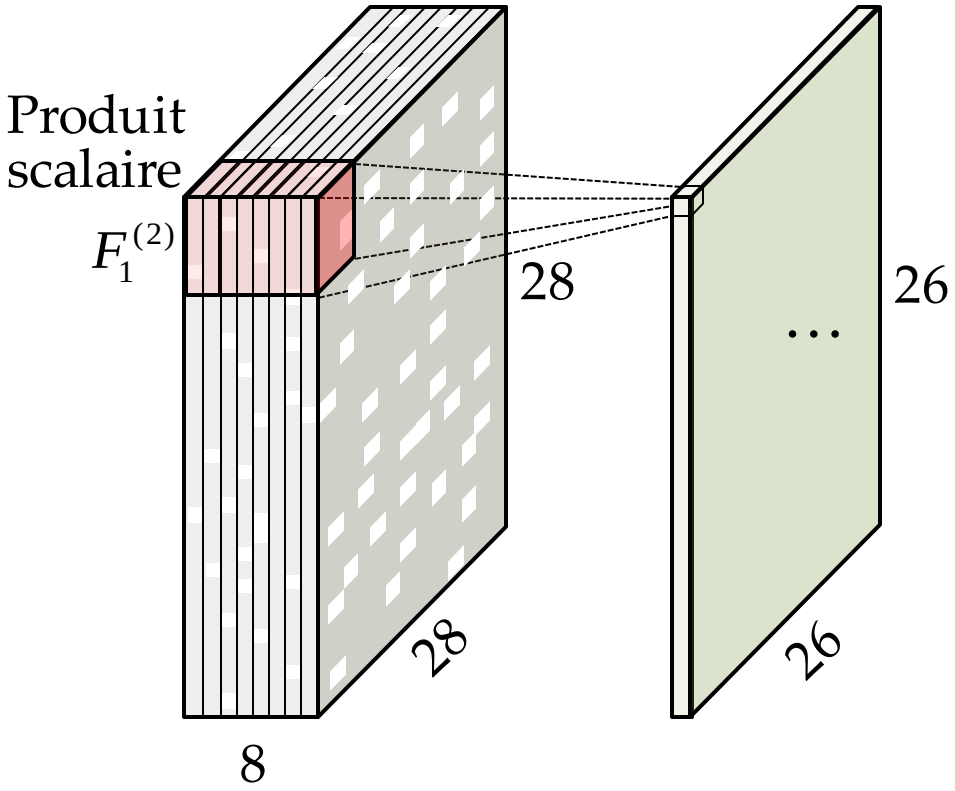
Résultante

- Applique la non-linéarité sur chaque entrée, individuellement
- Appelé *feature activation map*

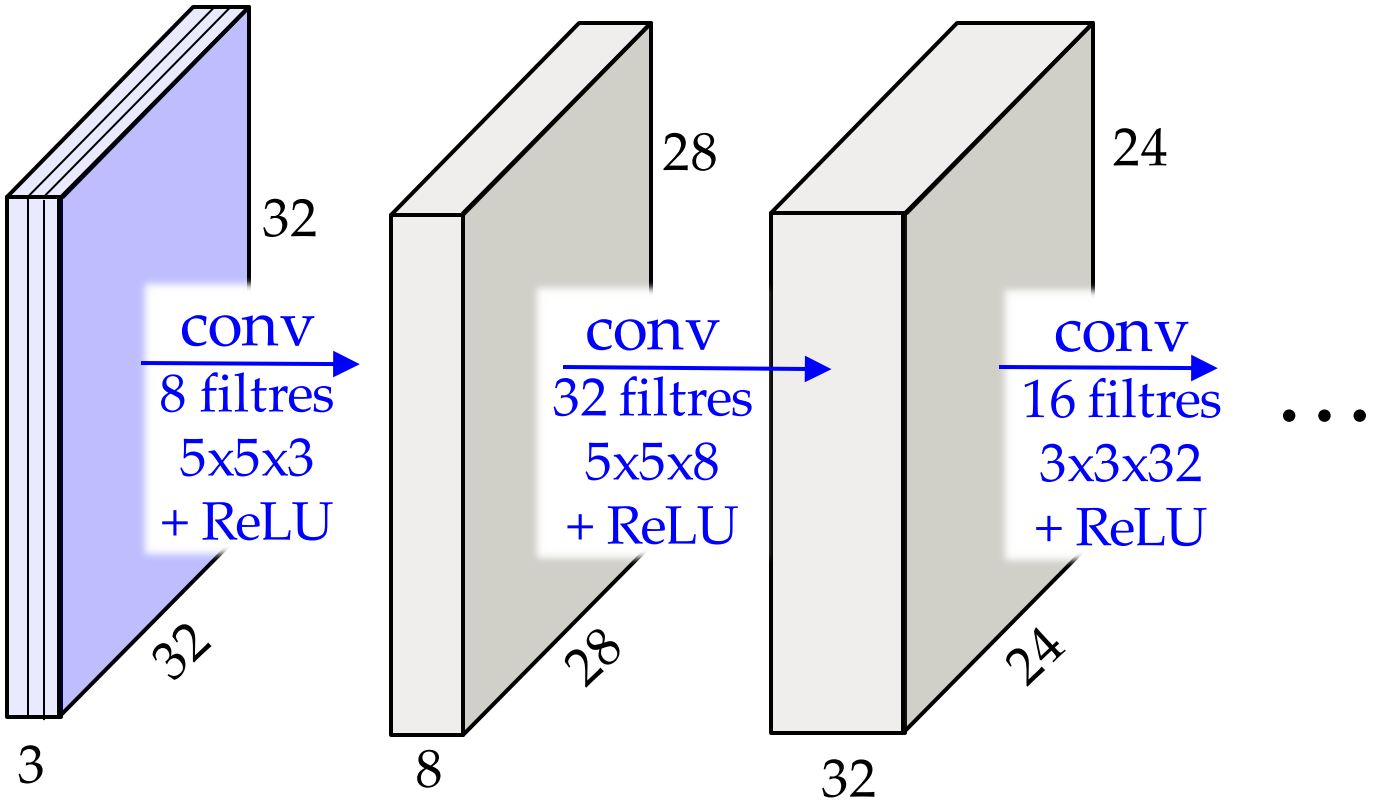


On recommence, avec une banque de filtres différents



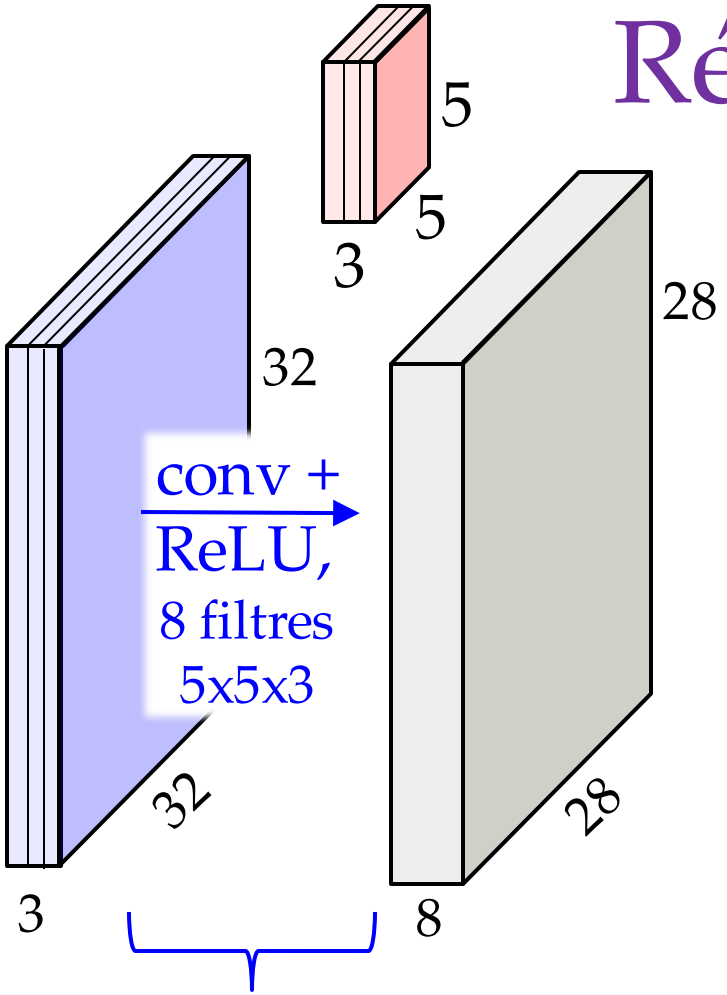


Idée de base



adapté de : cs231n, Université Stanford

Réduction # paramètres



- Perte de capacité d'expression vs. fully-connected
  - on ne peut pas établir de lien entre les pixels très éloignés
- Gros gain en réduction du nombre de paramètre

Combien de paramètres pour cette couche?

$(5x5x3 + 1) \times 8 = 608$

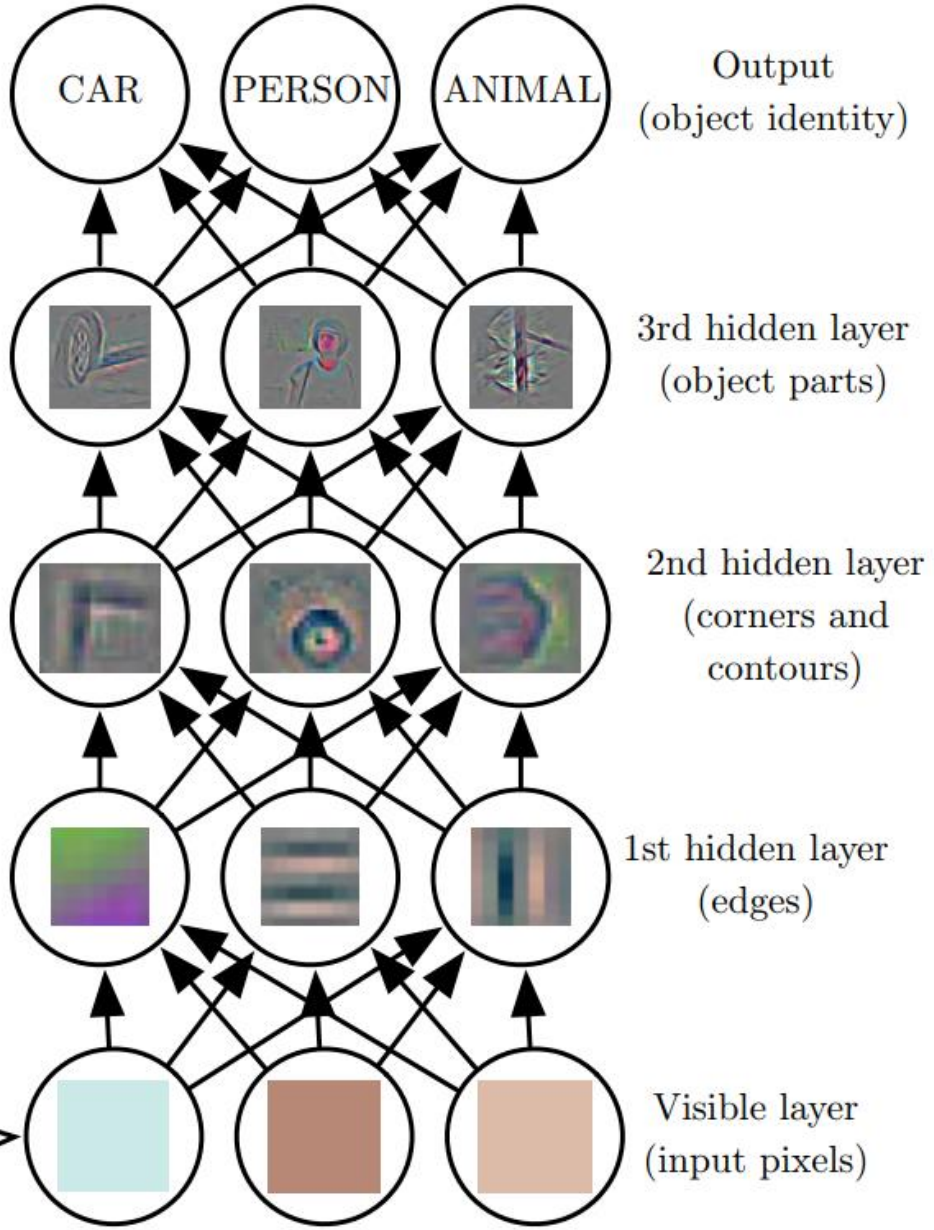
(1 : biais, 8 : filtres)

Combien de paramètres si c'était *fully-connected*?
28x28x8 = 6272 neurones
32x32x3 + 1 : 3073 param./neurones
Total : 19,273,856 paramètres

adapté de : cs231n, Université Stanford

Hiérarchie de filtres

Va établir des liens entre des pixels de plus en plus éloignés



(I. Goodfellow) deeplearningbook.org

Visualisation des *features* (Yosinski et al.)

Deep Visualization Toolbox

yosinski.com/deepvis

#deepvis


Jason Yosinski


Jeff Clune


Anh Nguyen


Thomas Fuchs


Hod Lipson

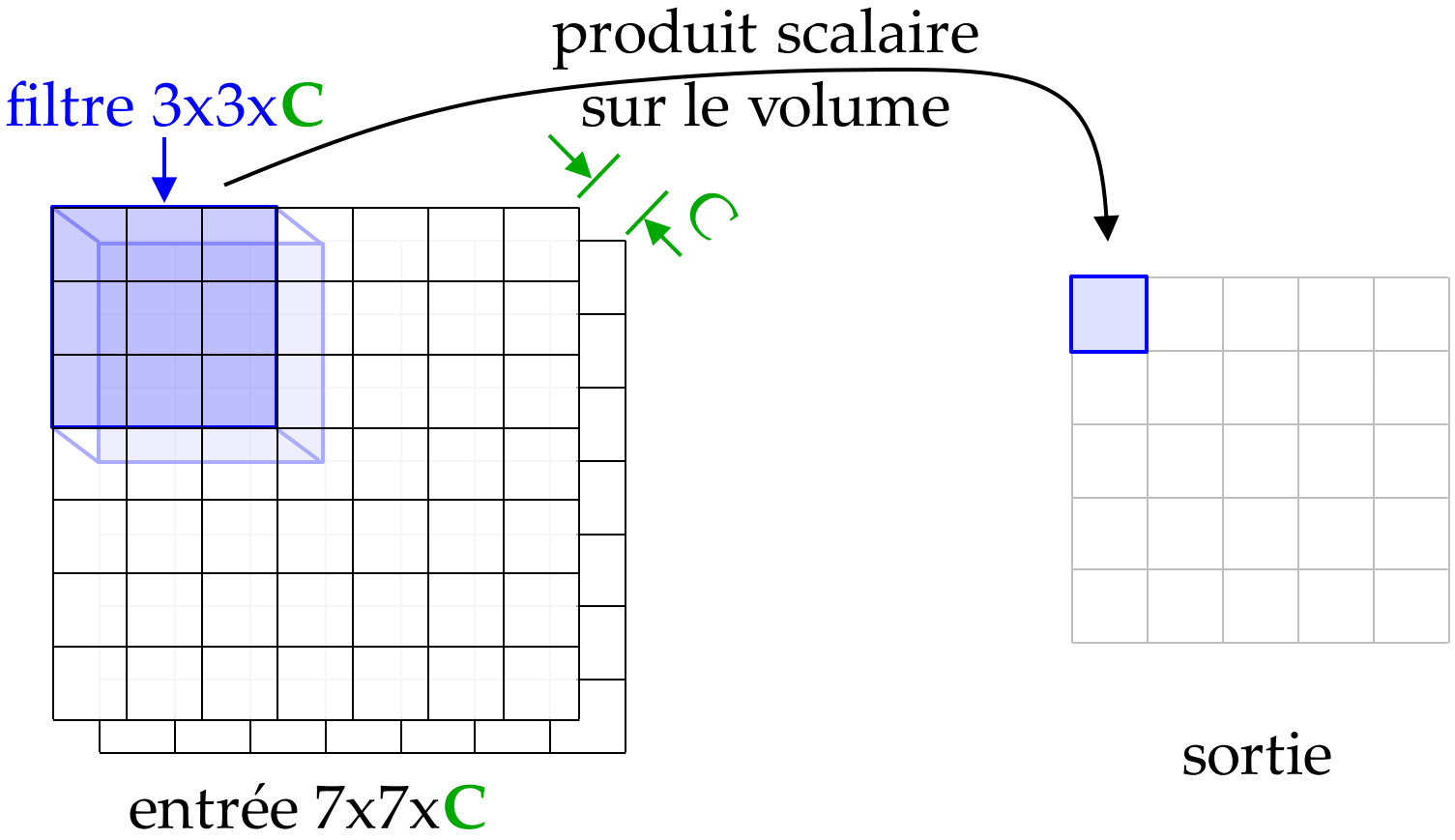
Cornell University

University of Wyoming

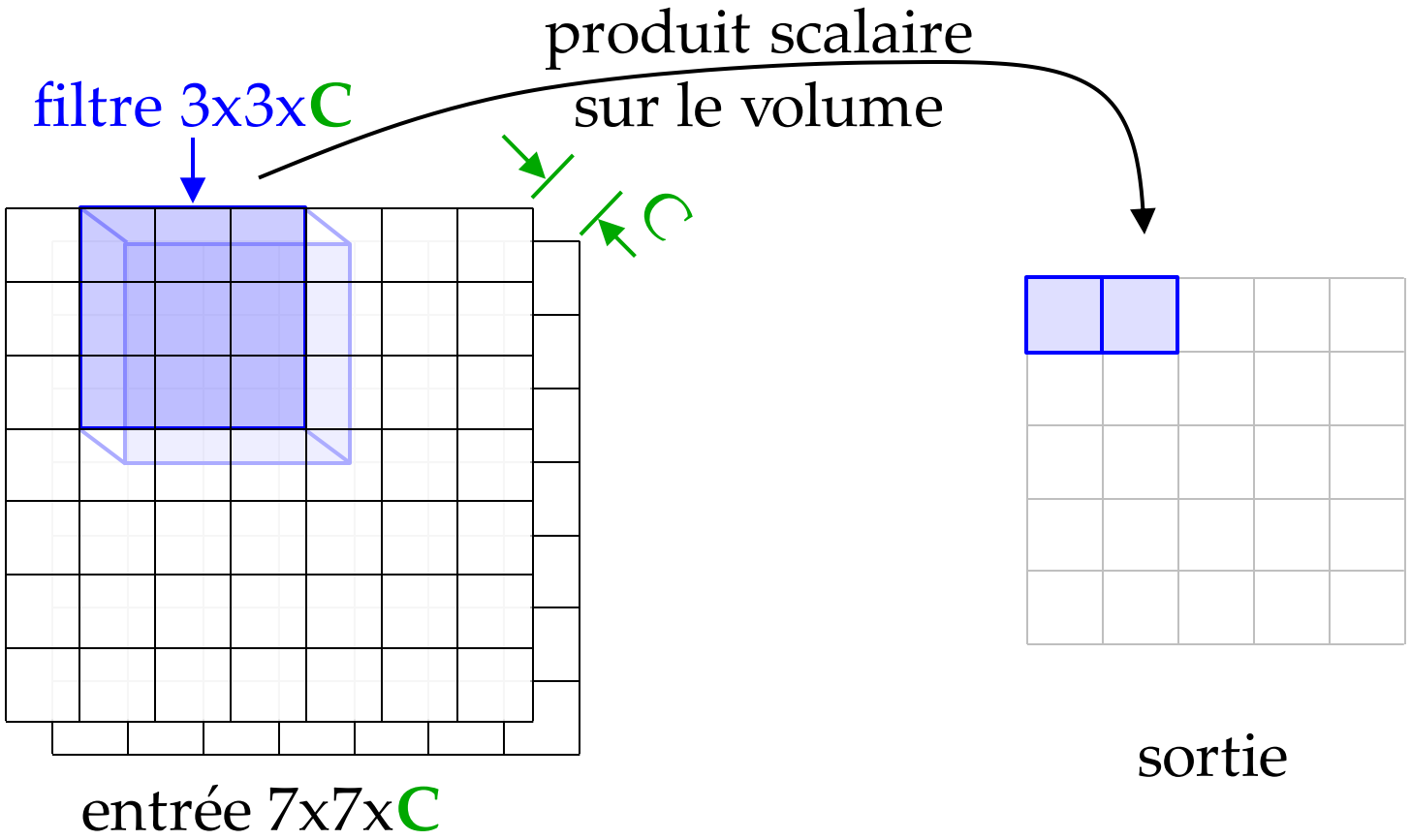
NASA Jet Propulsion Laboratory
California Institute of Technology

https://www.youtube.com/watch?v=AgkfIQ4IGaM

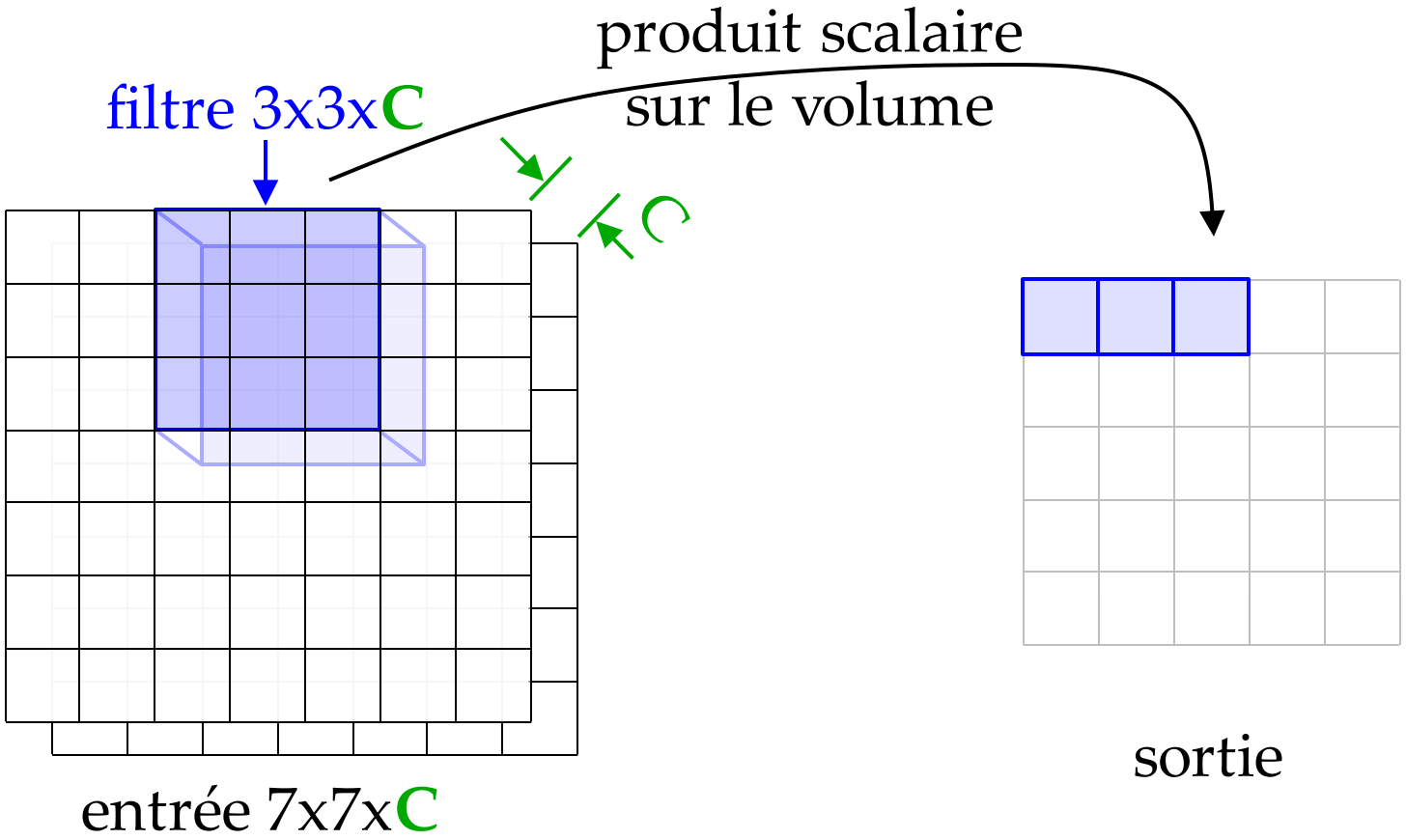
Effet de bord : redimensionalisation



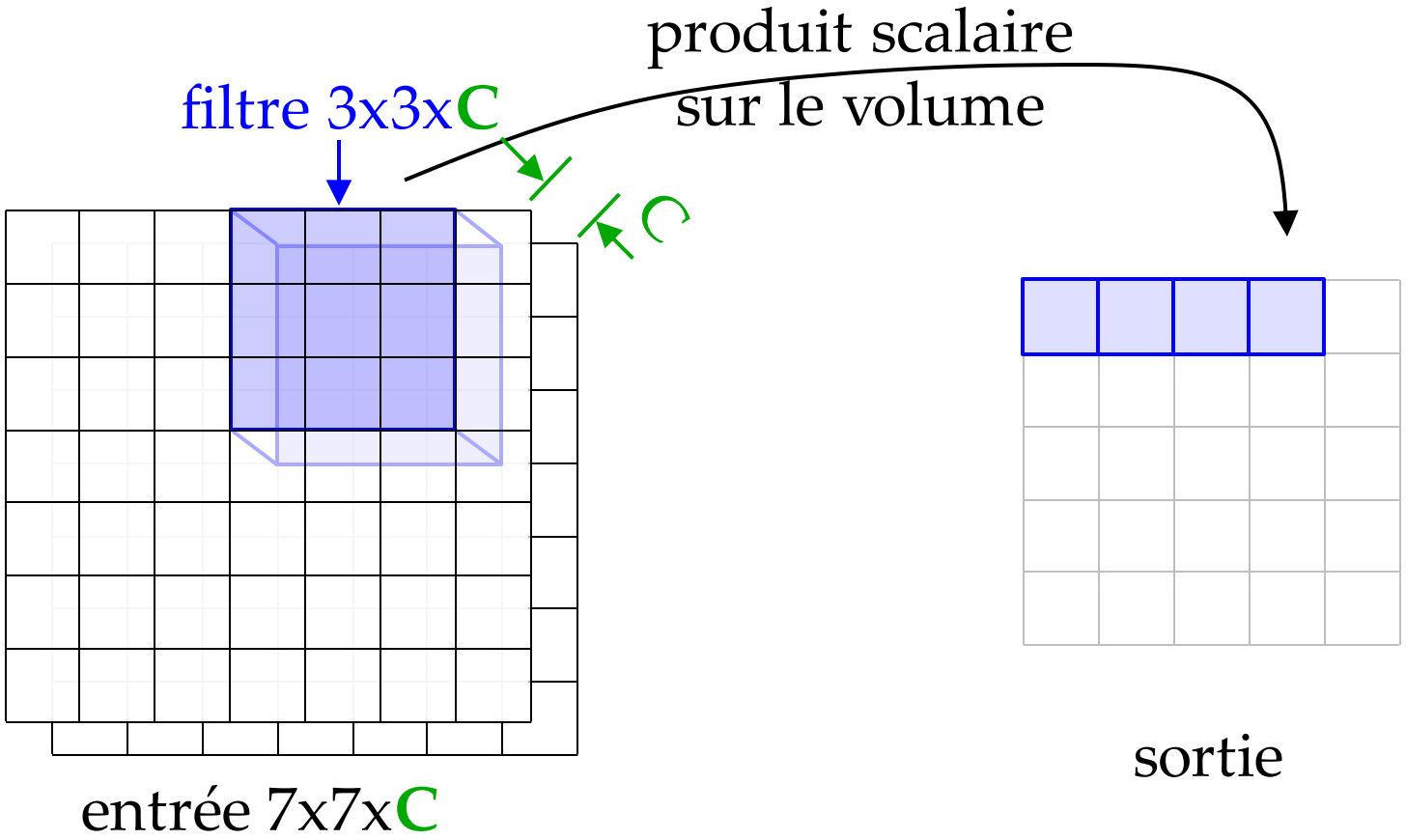
Effet de bord : redimensionalisation



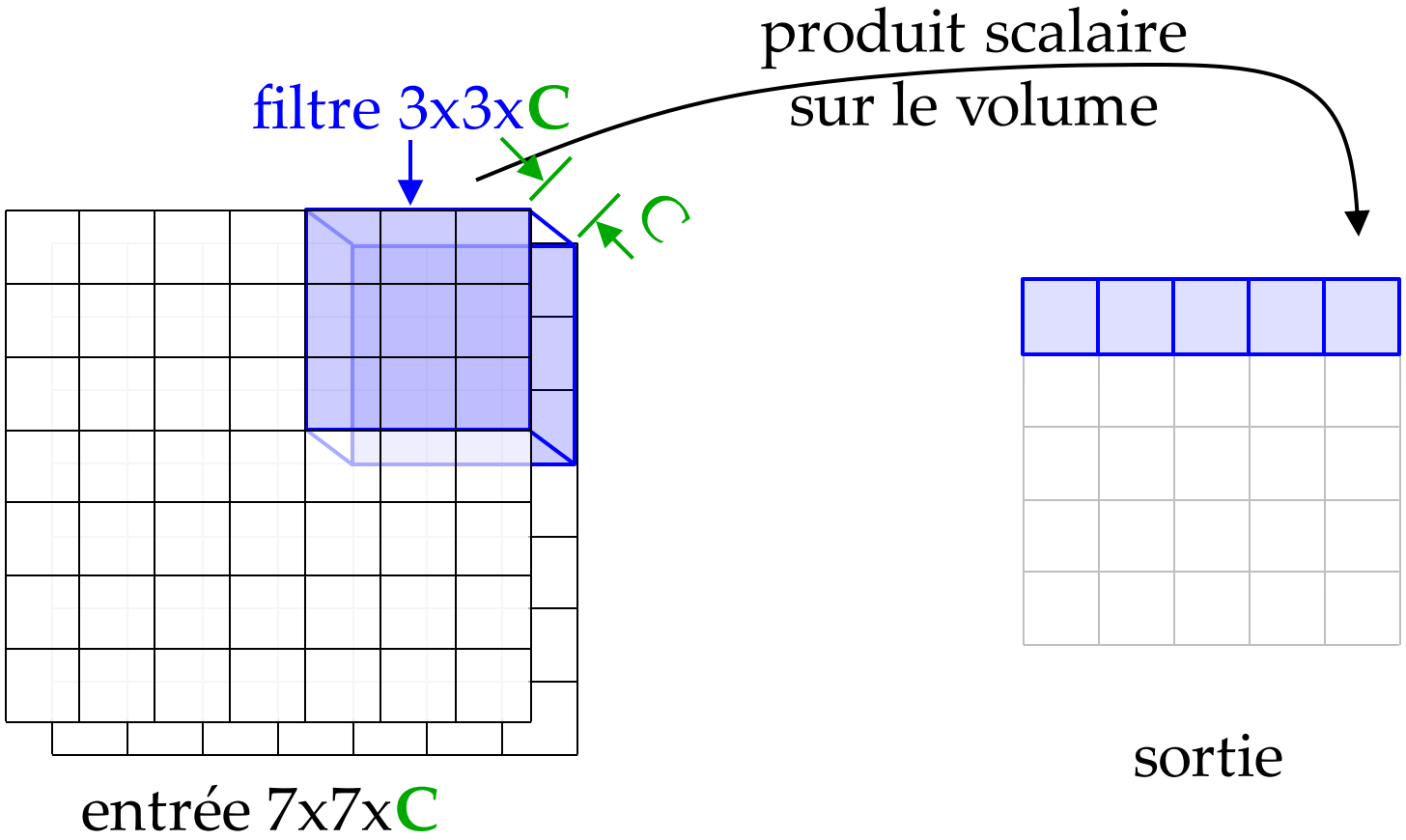
Effet de bord : redimensionalisation



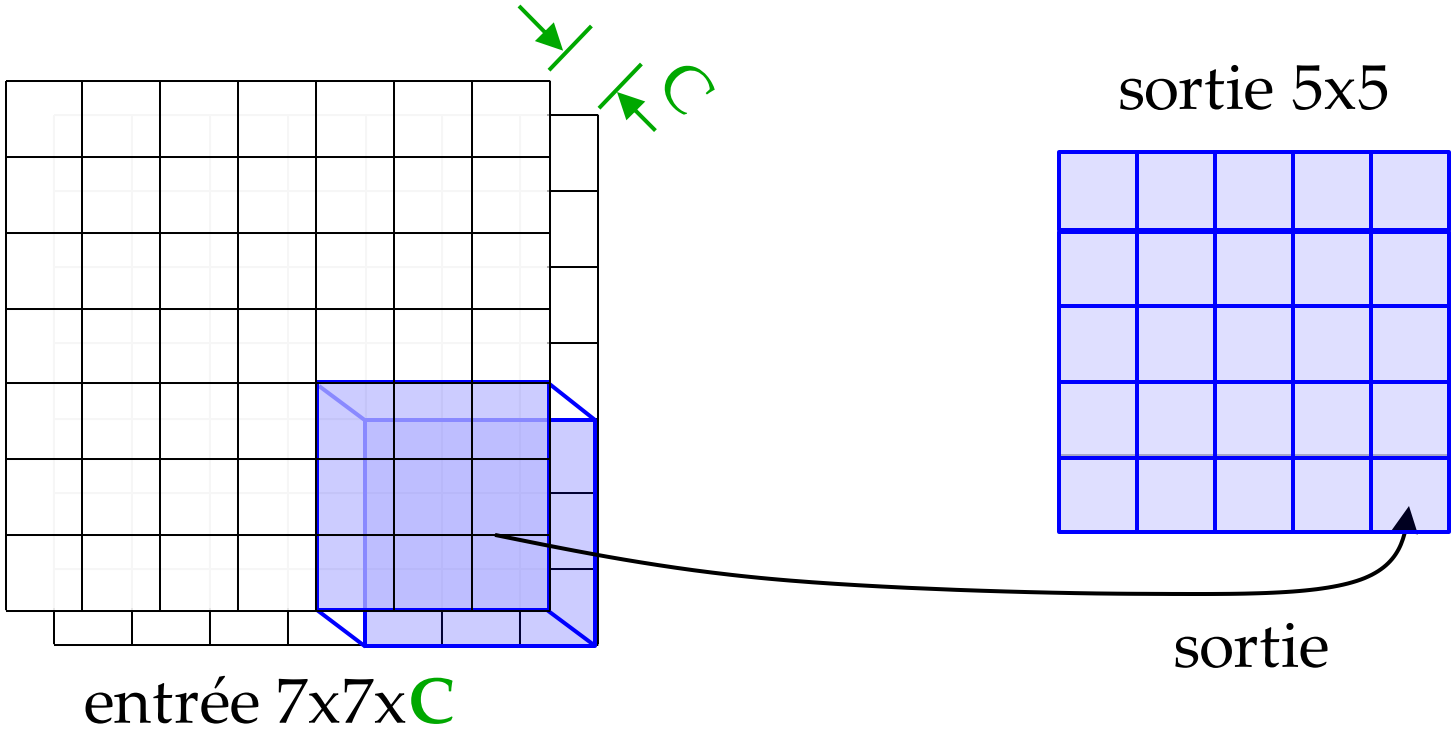
Effet de bord : redimensionalisation



Effet de bord : redimensionalisation

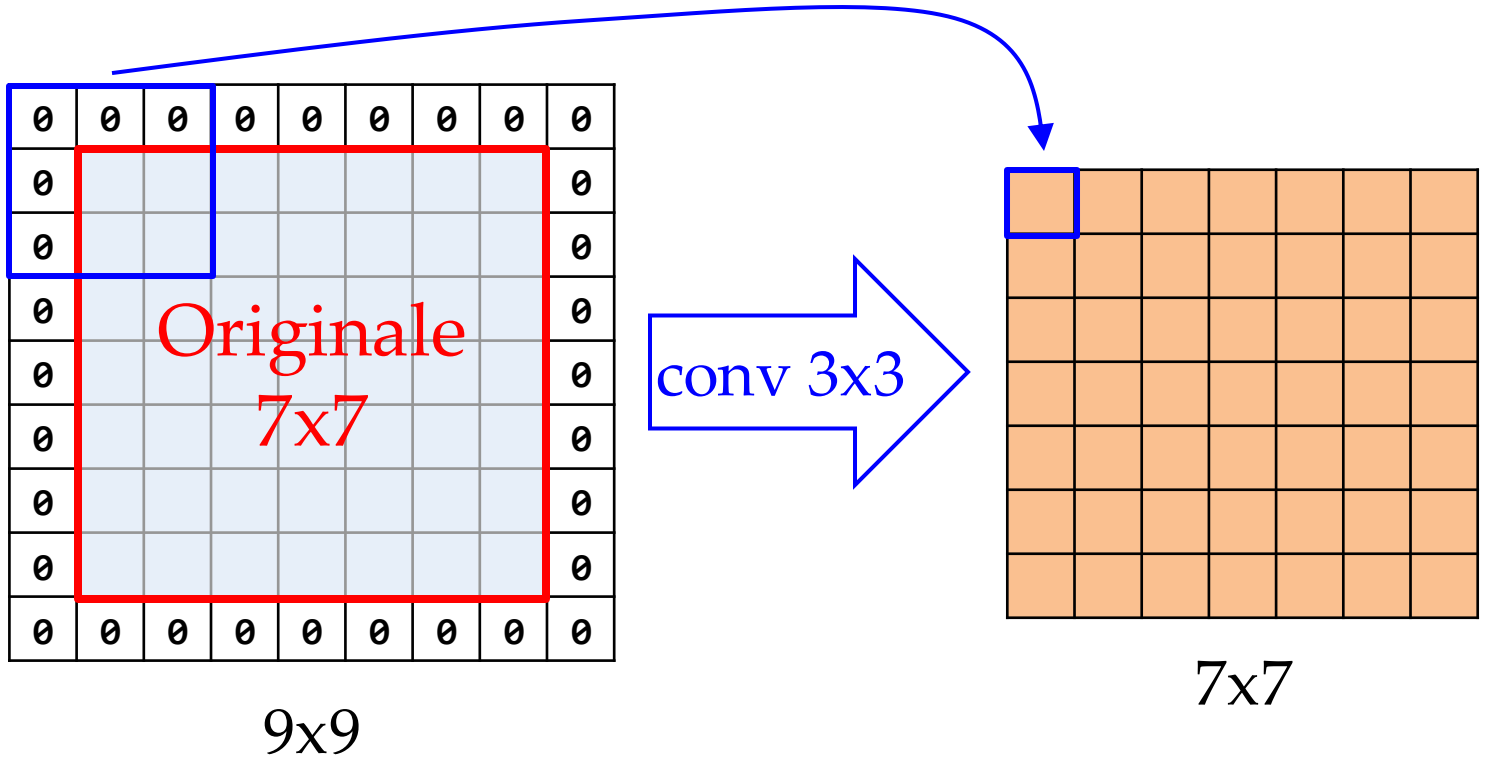


Effet de bord : redimensionalisation



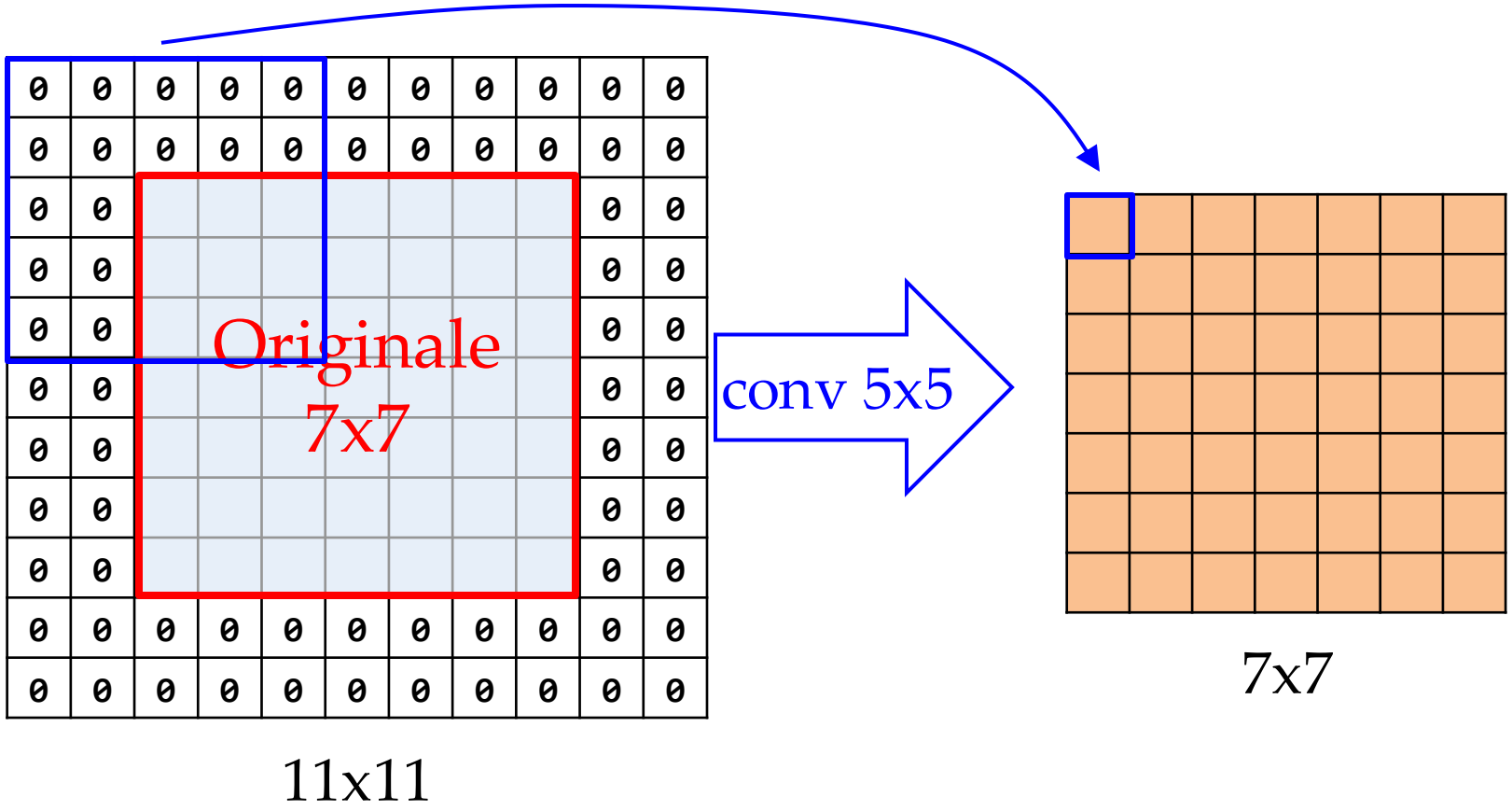
Ajout de zéros : *zero padding*

- Ajoute des entrées à 0 en bordure



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

Ajout de zéros : *zero padding*



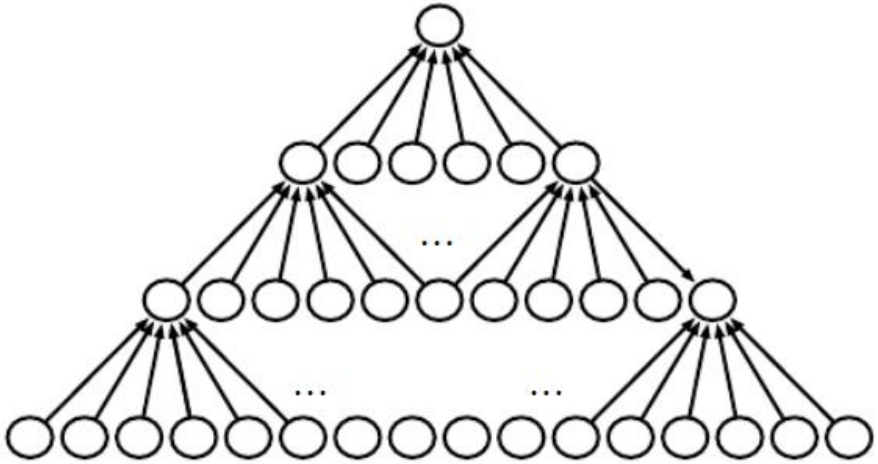
Pour filtre 3x3, padding de largeur 1
Pour filtre 5x5, padding de largeur 2
Pour filtre 7x7, padding de largeur 3

Zero-padding

- Conserve la largeur du pipeline
- Attention aux effets de bord, où les entrées sont moins informatives à cause du 0-padding

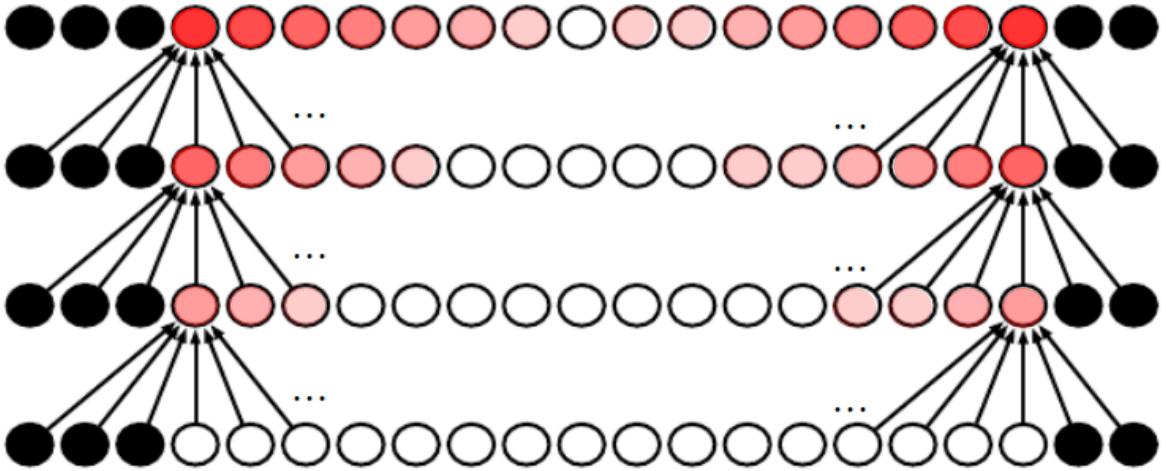
(ne semble pas trop problématique)

sans 0-padding



rougeur : impact du 0-padding

avec 0-padding



Quelques questions!

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

Réponse : 5x5x**3**

(Comme la 3ème dimension du filtre doit toujours être égale au nombre de canaux entrant, on la laisse souvent tomber dans la notation)

- Si j'ai 32 filtres 5x5 appliqués sur cette image 224x224x3, sans 0-padding, quelle sont les dimensions du tenseur de sortie?

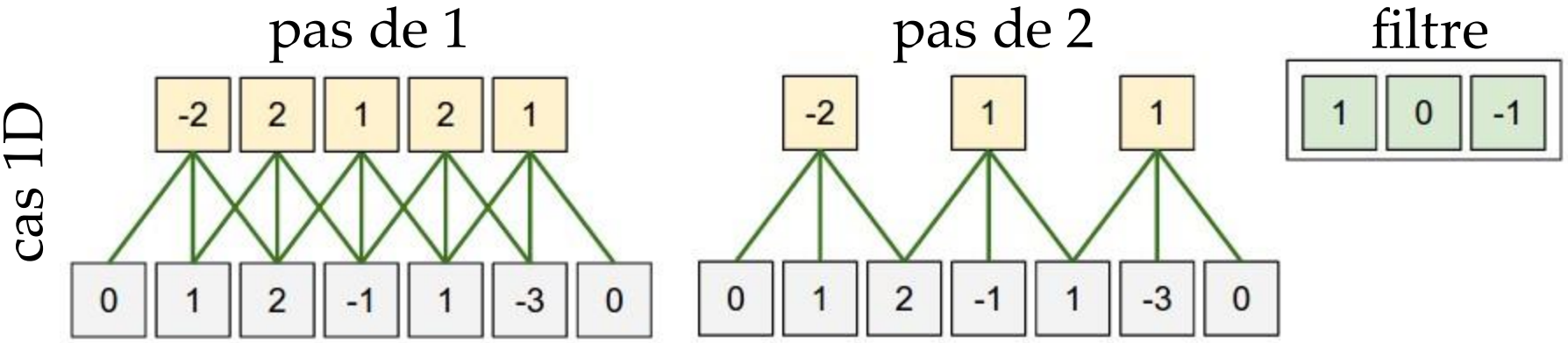
Réponse : 220x220x32

- Quelle est la largeur du 0-padding pour un tenseur de 64x64x10, si on applique un filtre 9x9?

Réponse : largeur de 4

Pas (*stride*)

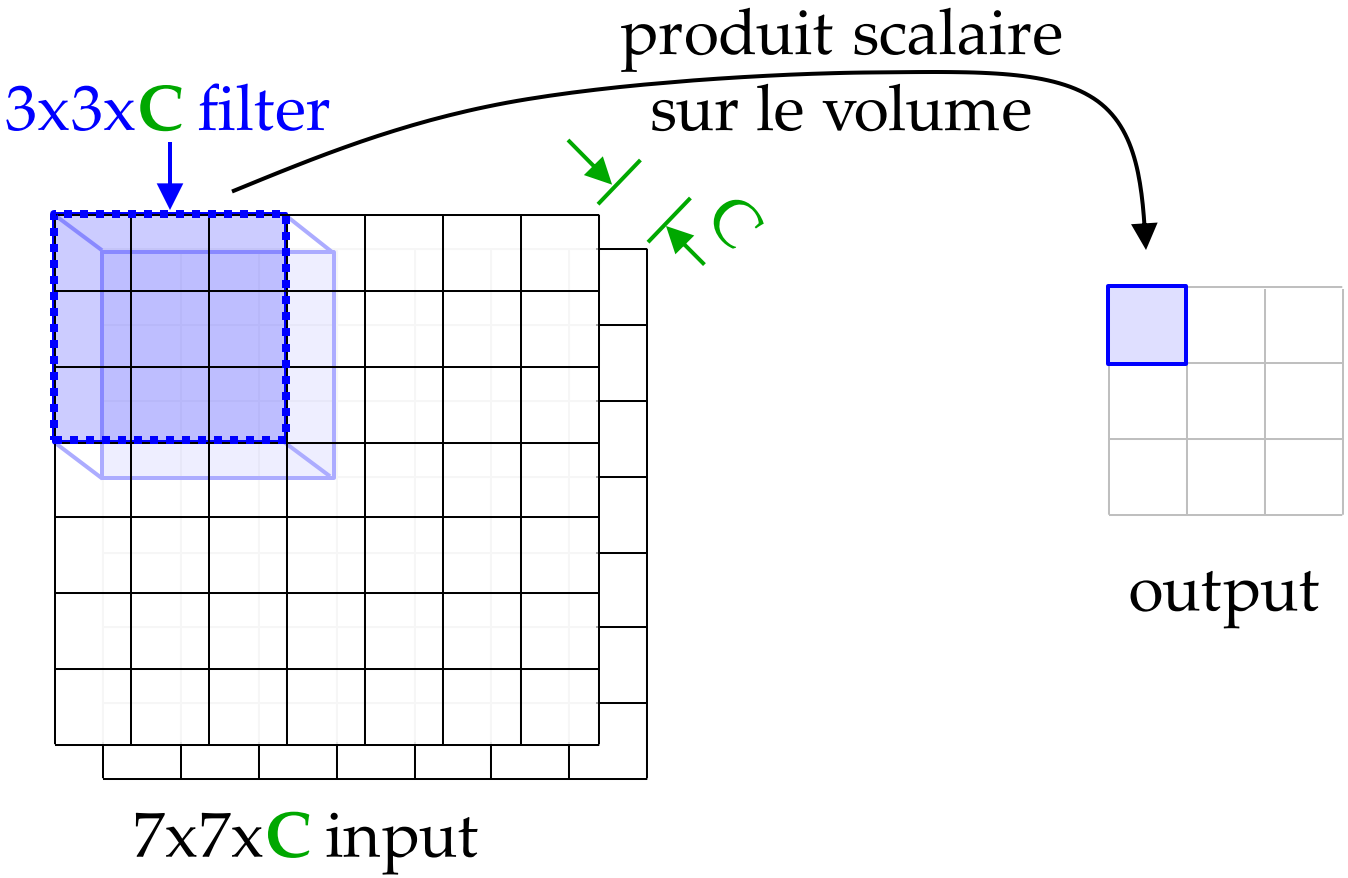
- ***Pas*** (stride) : saut dans l'application de la convolution



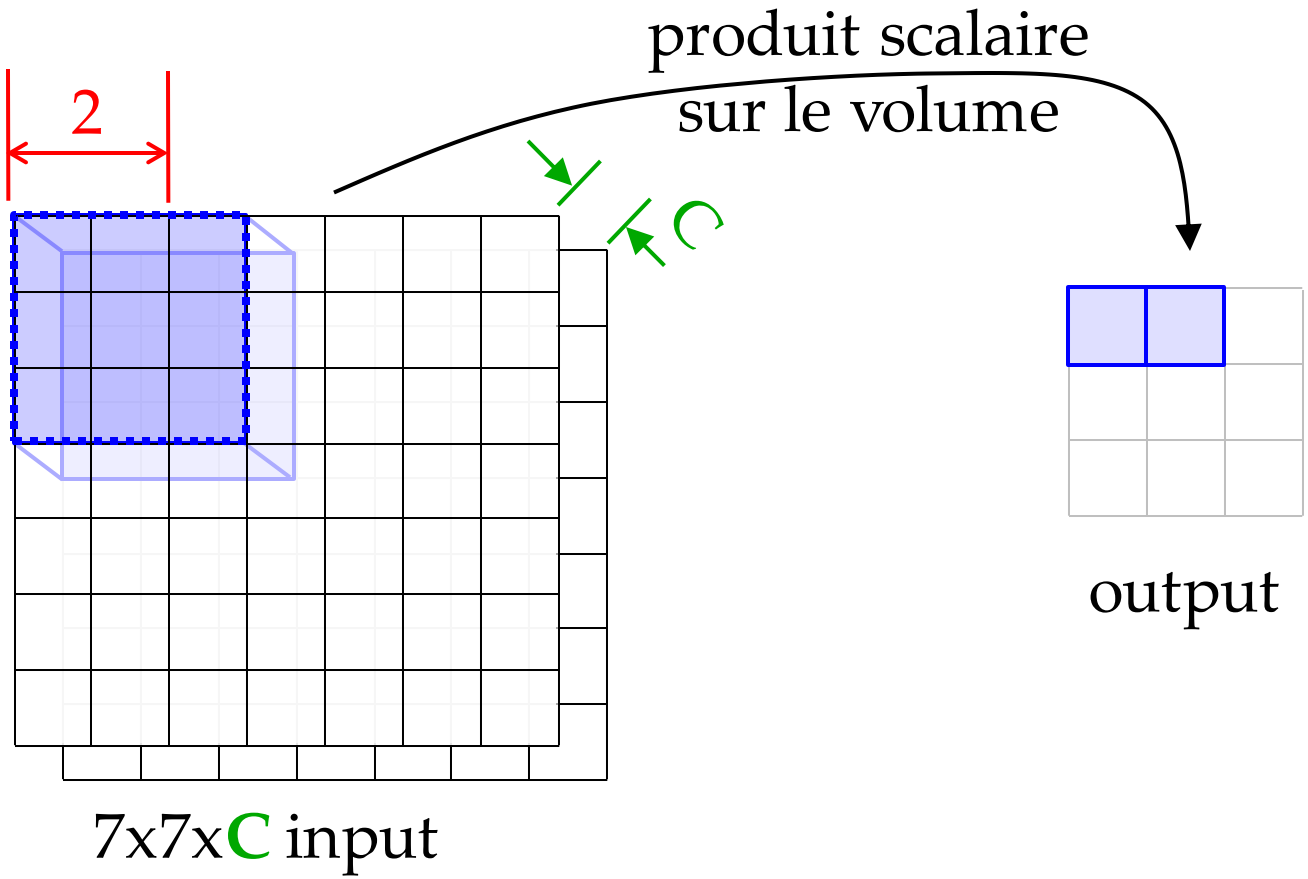
- Le ***pas*** est rarement plus de 3
  - Si plus d'1, réduit grandement la taille de sortie HxW
- Pas toujours possible d'avoir un nombre entier d'application de convolution, si le ***pas*** n'est pas 1
  - Par exemple, entrée 7x7, filtre 3x3, ***pas*** de 3
  - Libraire peut automatiquement faire du 0-padding, couper l'image (*crop*) ou lancer une exception

http://cs231n.github.io/convolutional-networks/

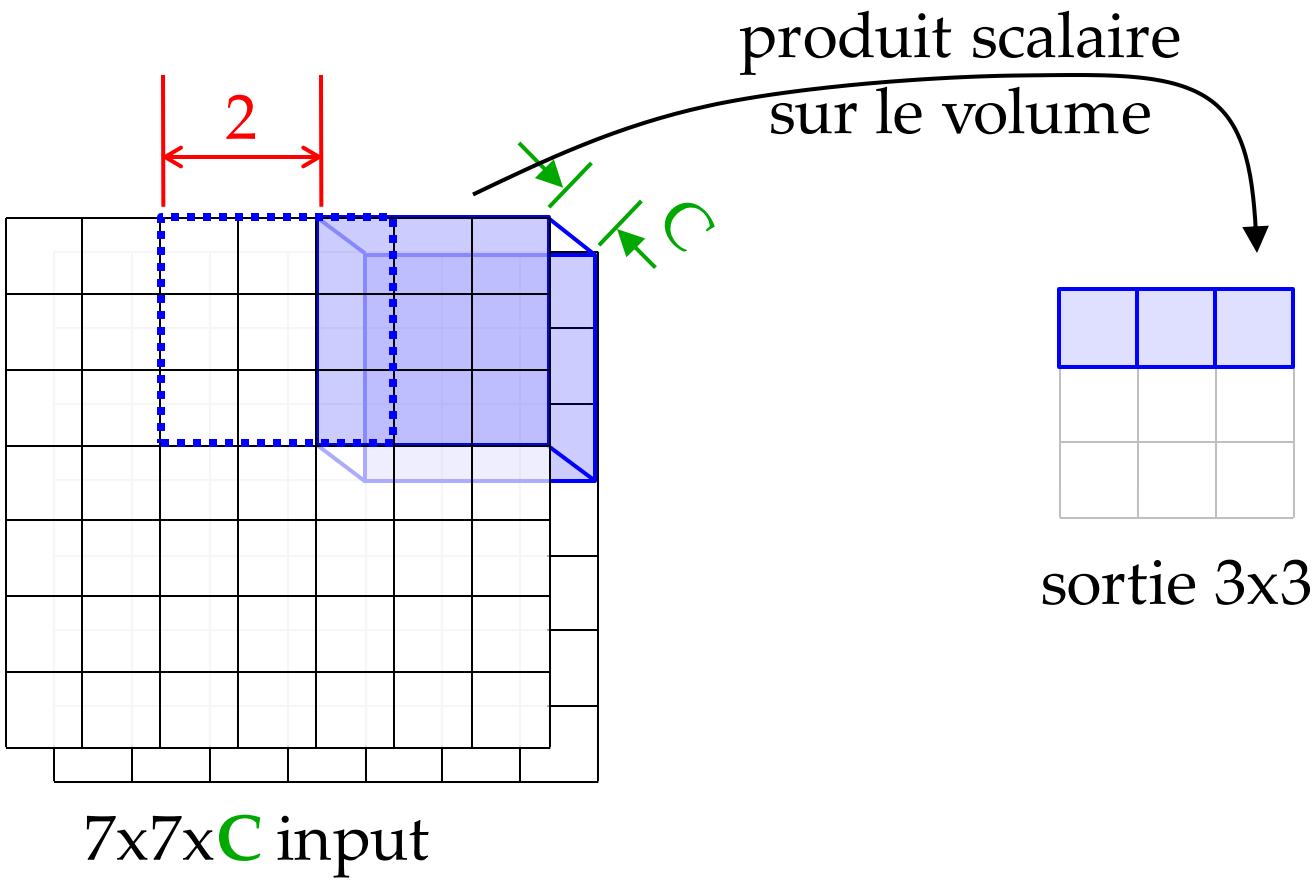
Pas de 2


produit scalaire
sur le volume
3x3xC filter
C
output
7x7xC input

Pas de 2



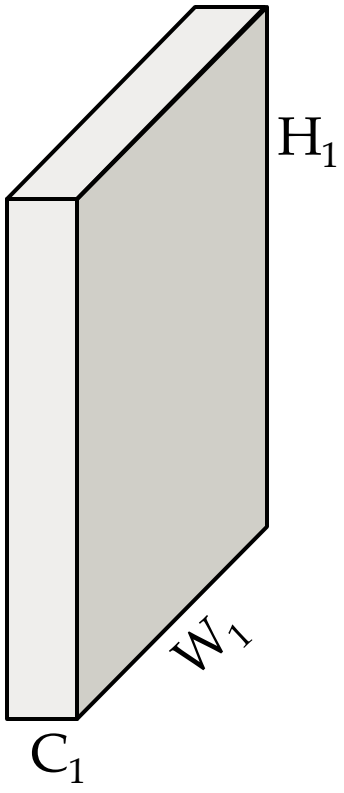
Pas de 2

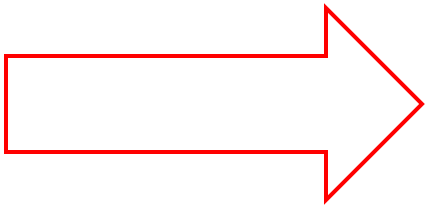


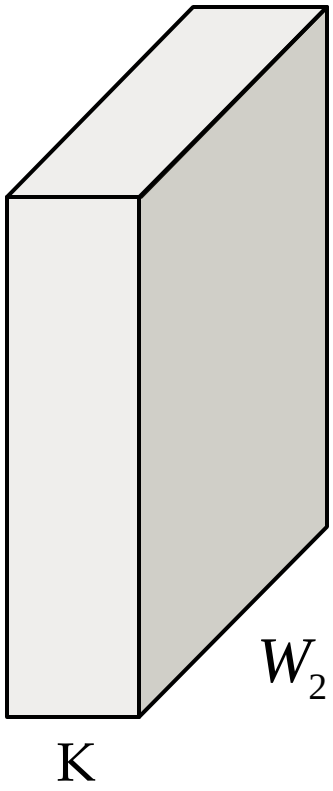
Tailles entrée et sortie

K : nombre de filtres
F : taille du filtre (FxF)
S : pas (stride)
P : quantité de 0-padding

Valeurs typiques (cs231n)
K : puissances de 2
F=3, S=1, P=1
F=5, S=1, P=2
F=5, S=2, P=autant que nécessaire
F=1, S=1, P=0





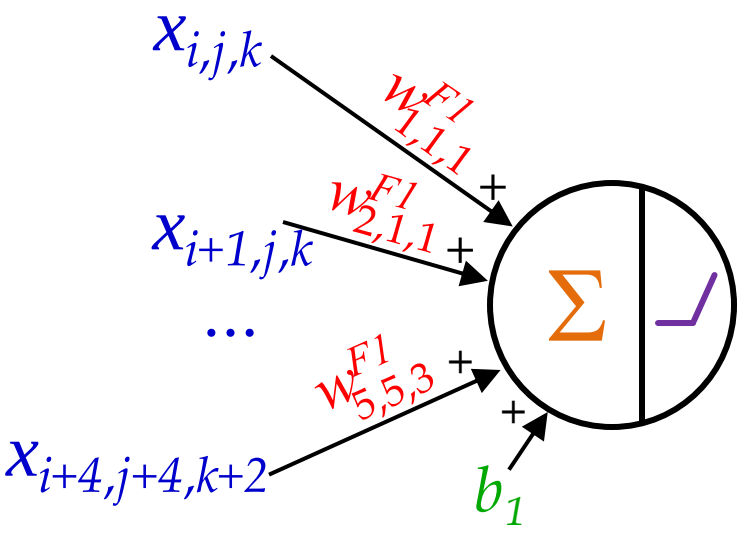


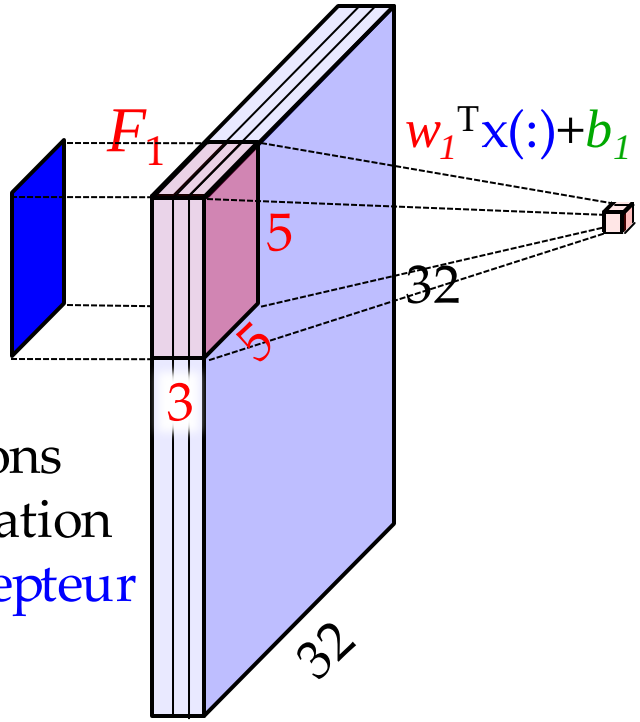
$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

paramètres : $F \cdot F \cdot C_1 \cdot K$ poids + K biais

Point de vue d'un neurone



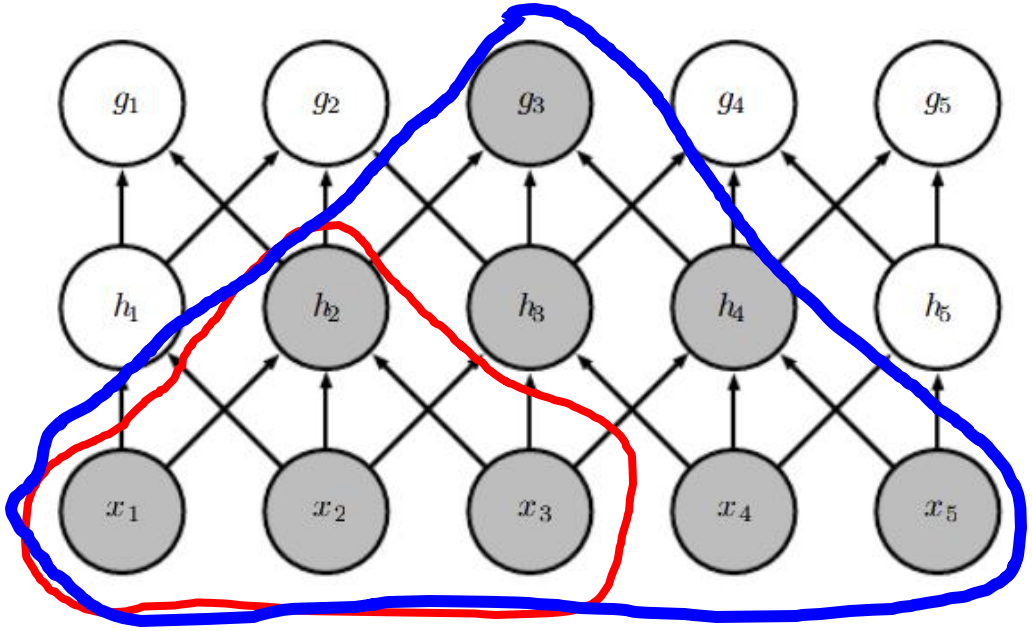


champ récepteur

prend des décisions basées sur l'information dans son champ récepteur

Champ récepteur

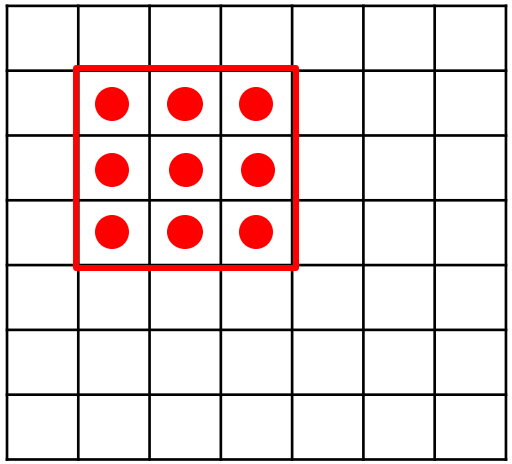
- Augmente avec la profondeur
- Les neurones plus haut peuvent prendre des décisions basées sur des champs plus grands



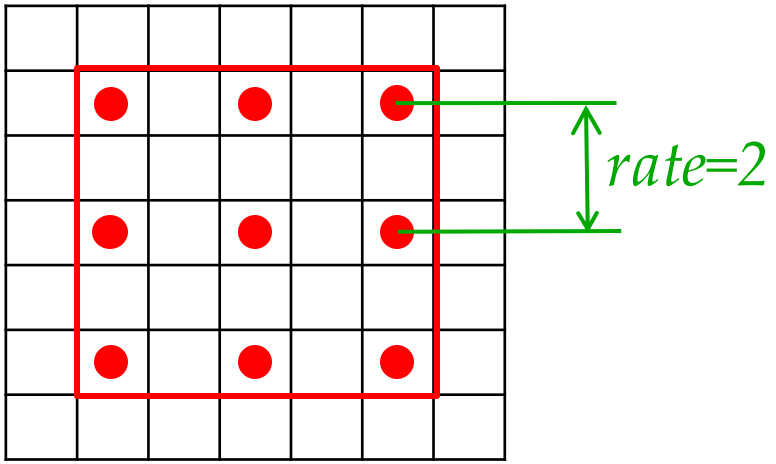
(I. Goodfellow) deeplearningbook.org

Convolution « à trous »

- En anglais : « *atrous* », *dilated convolution*



Classique 3x3



À trous 3x3 *Encore 9 paramètres*

- Permet d'étendre la taille du champ récepteur, sans augmenter le nombre de paramètres
- Remplace *maxpooling* en quelque sorte
- Utilisé dans des concepts comme *spatial pyramid*

Conv2d

CLASS `torch.nn.Conv2d(`*`in_channels`*`, `*`out_channels`*`, `*`kernel_size`*`, `*`stride=1`*`, `*`padding=0`*`, `*`dilation=1`*`, `*`groups=1`*`, `*`bias=True`*`)` [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_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

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

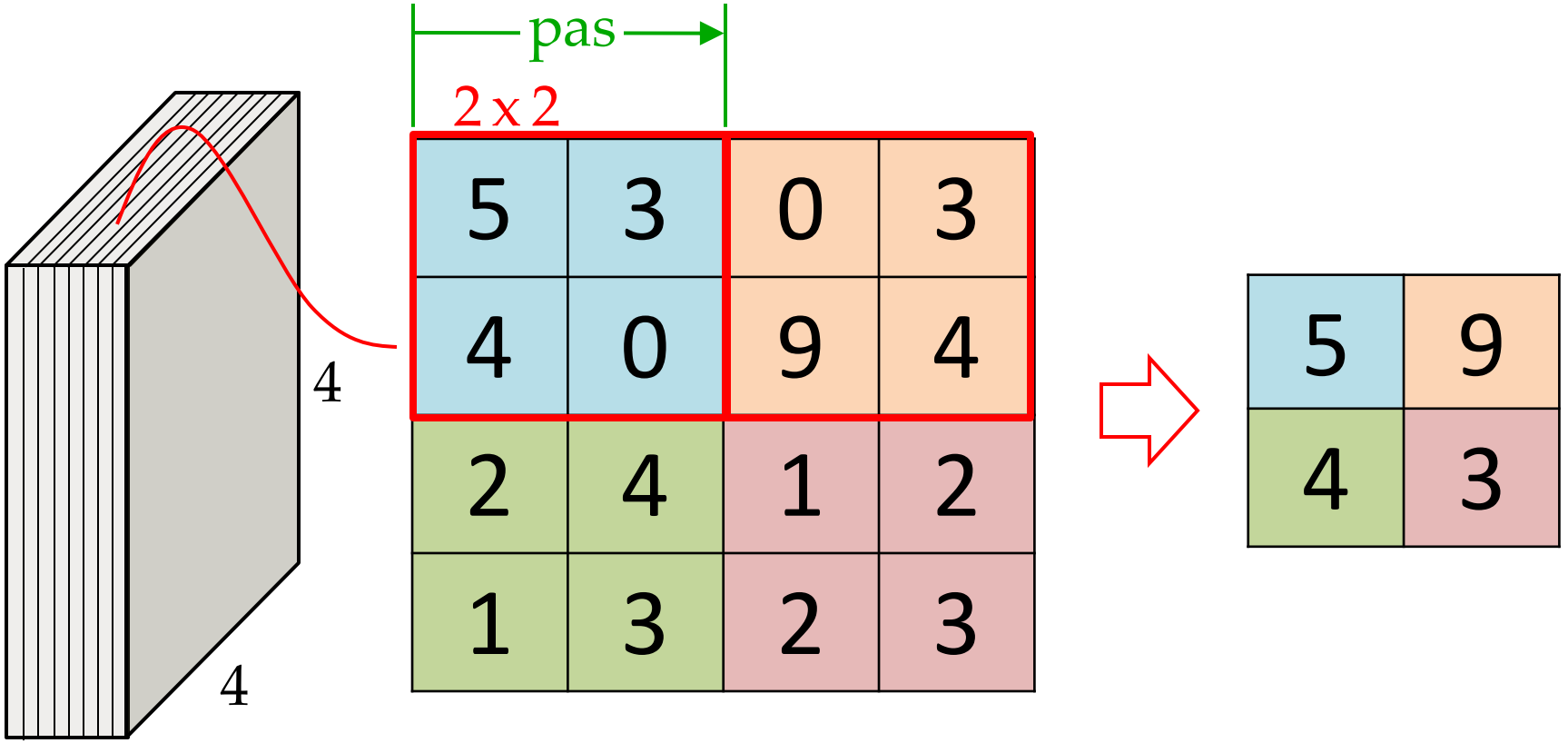
where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for `padding` number of points for each dimension.
- `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.

Pooling

Max Pooling

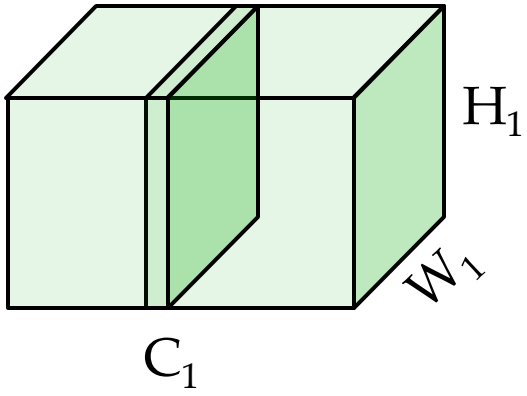
- Appliqué pour chaque tranche, indépendamment

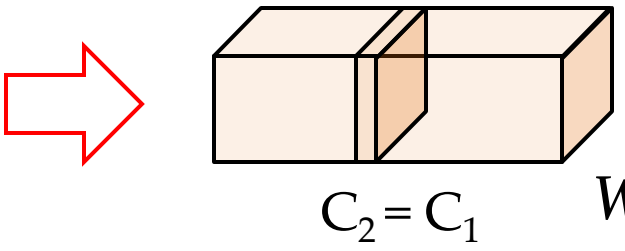


Doit spécifier :
- taille
- pas

Max Pooling

F : taille
S : pas (*stride*)





$$H_2 = \frac{H_1 - F}{S} + 1$$

$$W_2 = \frac{W_1 - F}{S} + 1$$

Valeurs usuelles : F = 2, S=2
F = 3, S=2

N'ajoute aucun paramètre entrainable ☺

Max Pooling

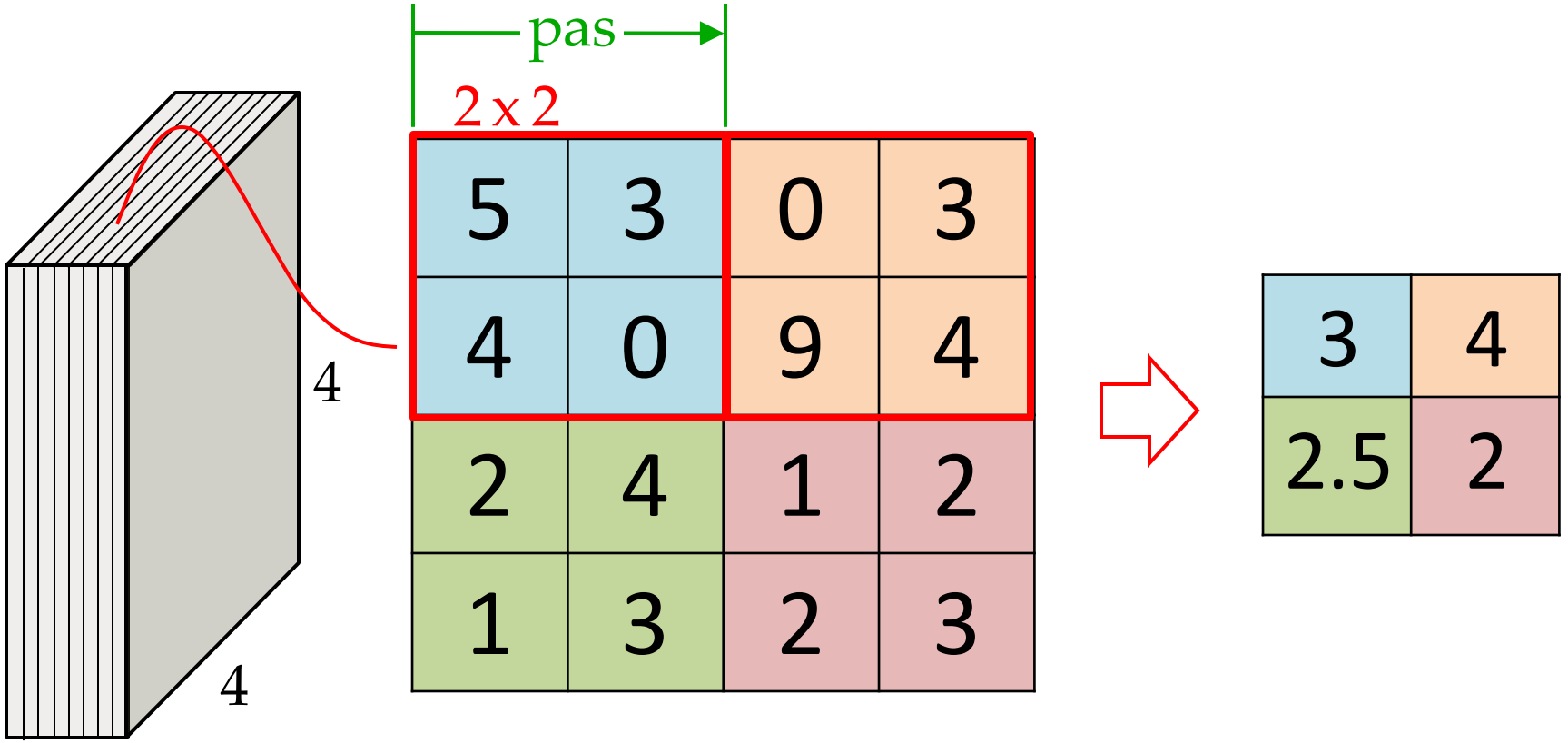
- Réduit la dimension du feature map
  - se fait aussi avec **conv+*stride*≠1**
- Souvent, on en profite pour augmenter le nombre de filtre
  - la « quantité » d'information reste similaire
  - augmente la richesse de la représentation / abstraction
- Pourquoi maxpool au lieu de faire une moyenne?
  - maxpool: détecte la présence d'un feature dans une région
  - avgpool: va en partie noyer cette valeur (ou compter le nombre)

Max Pooling

- Question : comment se propage le gradient avec maxpooling ?

Average Pooling

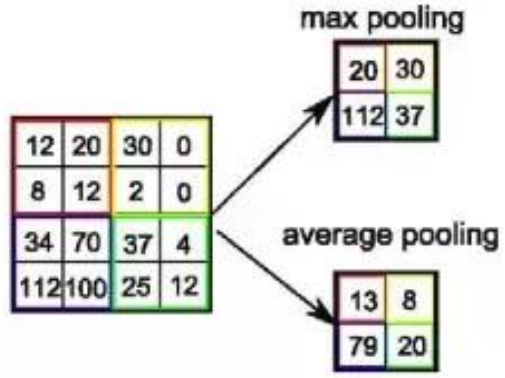
- On fait la moyenne sur chaque fenêtre

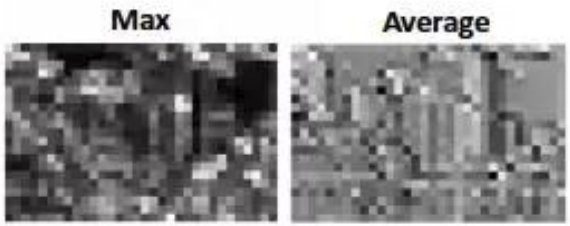


Doit spécifier :
- taille
- pas

Average Pooling

- Contrairement à maxpooling, on ne sélectionne pas de features en particulier
- Va avoir tendance à lisser les features (filtre passe-bas)
- Gradient va se propager à toutes les cellules



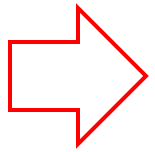


- Va voir plus loin une utilisation particulière en fin de pipeline (*global averaging pooling*)

Stochastic Pooling

- On pige la sortie au hasard durant l'entraînement, avec probabilité proportionnelle à l'activation

5	3	0	3
4	0	9	4
2	4	1	2
1	3	2	3



5/12	3/12	0	3/16
4/12	0	9/16	4/16
2/10	4/10	1/8	2/8
1/10	3/10	2/8	3/8

Probabilités p

Exemple sortie pigée:

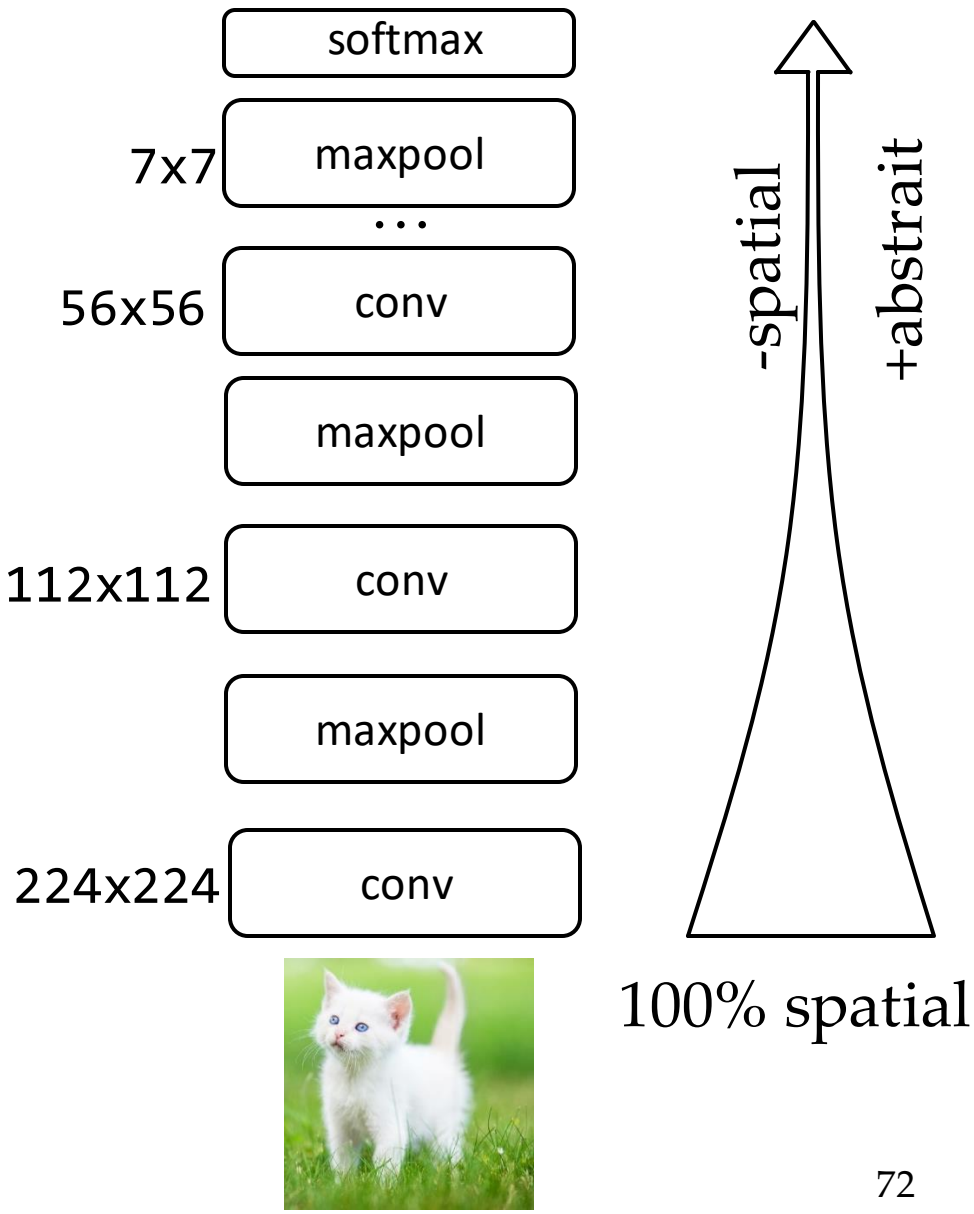
4	9
3	2

- **Pour les tests**, on prend la moyenne pondérée par p
- Semble offrir une forme de régularisation

https://arxiv.org/pdf/1301.3557.pdf

Pooling

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



Conv2d

CLASS `torch.nn.Conv2d(`*`in_channels`*`, `*`out_channels`*`, `*`kernel_size`*`, `*`stride=1`*`, `*`padding=0`*`, `*`dilation=1`*`, `*`groups=1`*`, `*`bias=True`*`)` [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_{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.

- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for `padding` number of points for each dimension.
- `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.

Conv1d

CLASS `torch.nn.Conv1d(`*`in_channels`*, *`out_channels`*, *`kernel_size`*, *`stride=1`*, *`padding=0`*, *`dilation=1`*, *`groups=1`*, *`bias=True`*`)` [SOURCE]

Applies a 1D convolution over an input signal composed of several input planes.

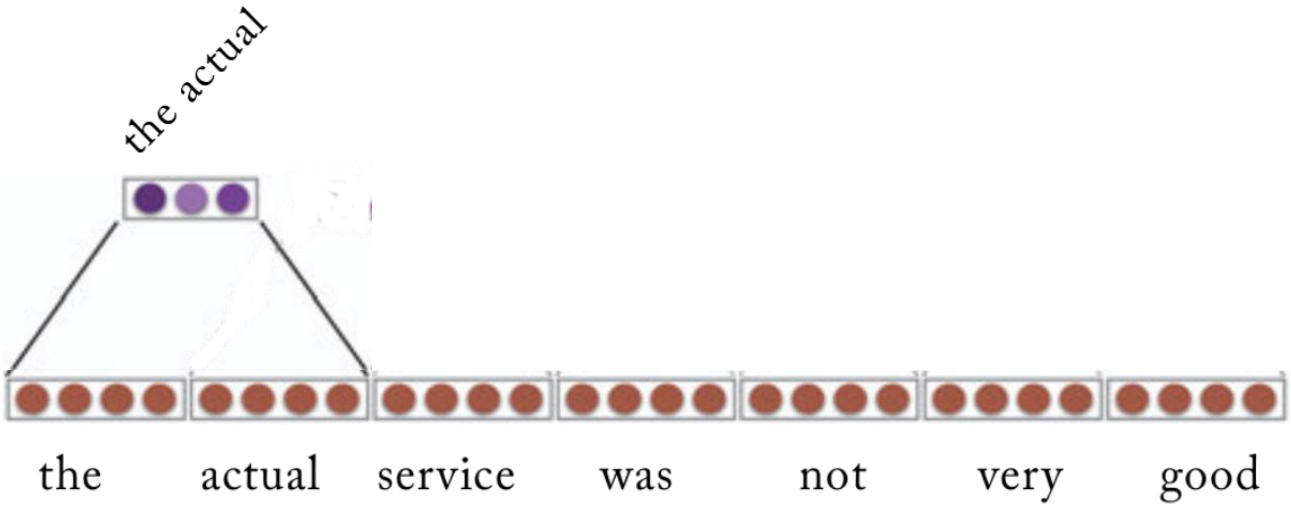
In the simplest case, the output value of the layer with input size (N, C_{in}, L) and output $(N, C_{\text{out}}, L_{\text{out}})$ can be precisely described as:

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

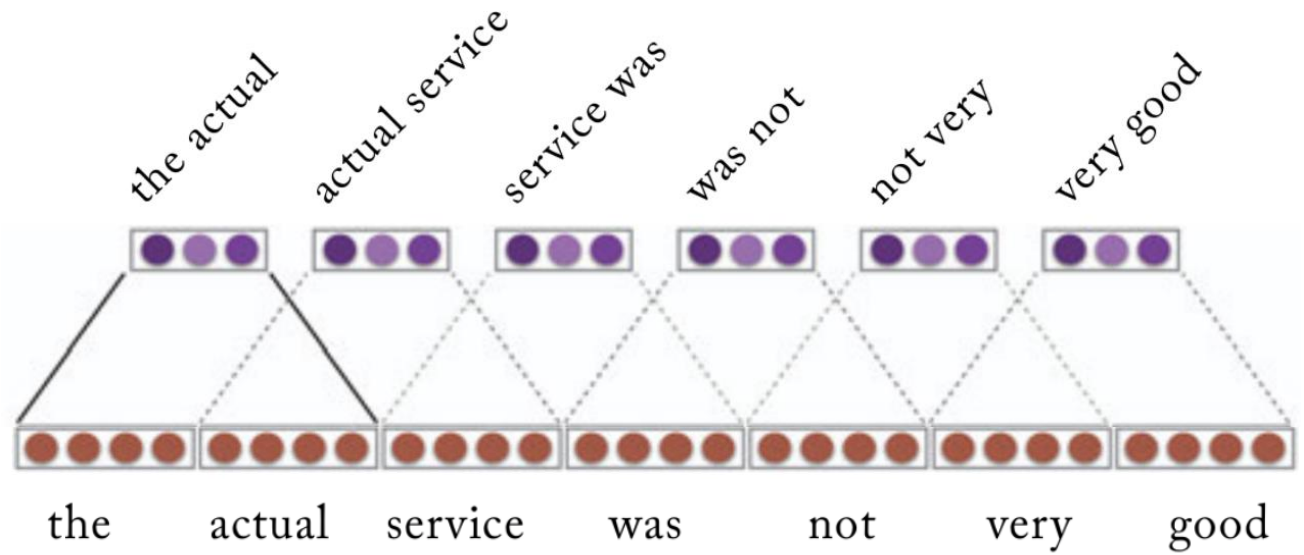
where $\star$ is the valid cross-correlation operator, N is a batch size, C denotes a number of channels, L is a length of signal sequence.

- `stride` controls the stride for the cross-correlation, a single number or a one-element tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for `padding` number of points.
- `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

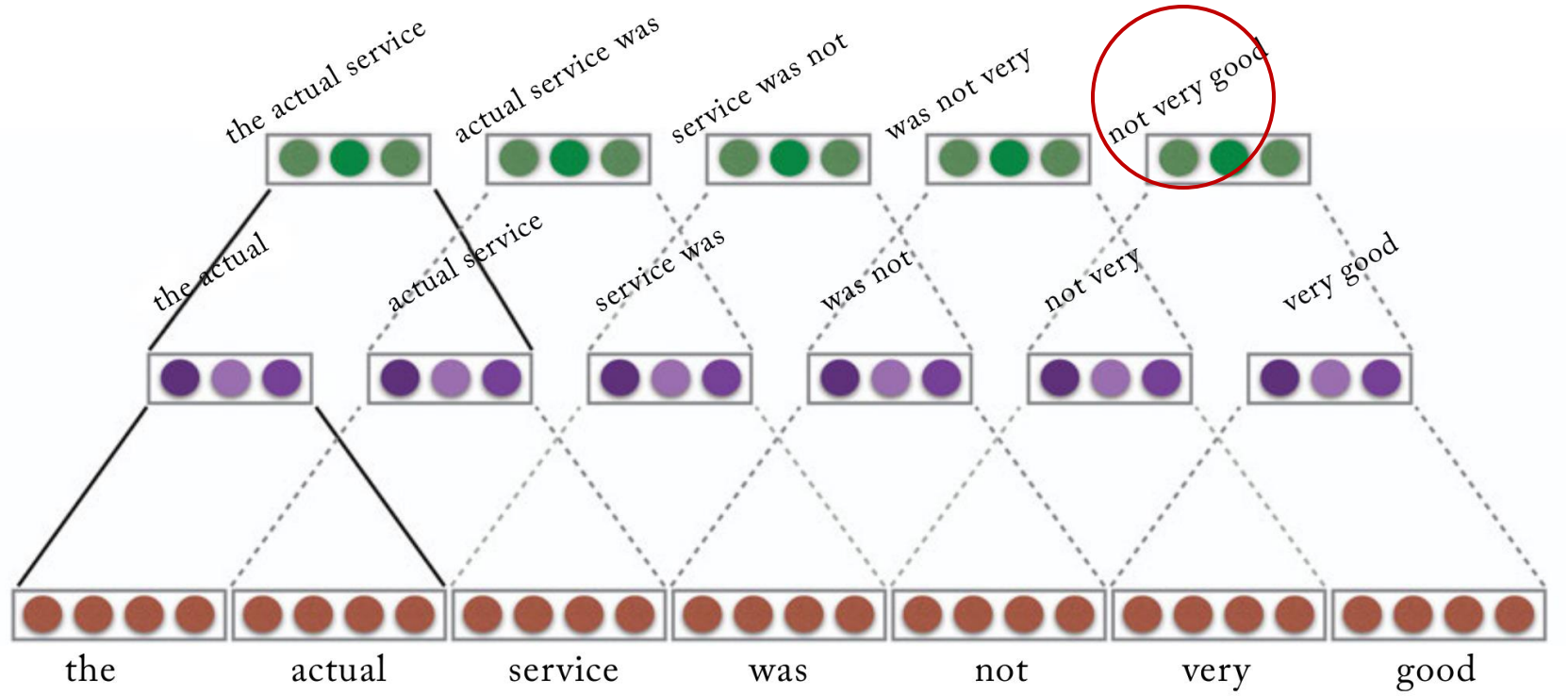
Utiliser des CNNs pour autre chose?



Utiliser des CNNs pour autre chose?



Utiliser des CNNs pour autre chose?



Convolutions 1D sur du texte

- Utilisation de word embeddings (que l'on verra bientôt)
- Extracteurs de "n-grams"
- On a un petit problème lorsque l'on fait de la classification...

Conv1d

CLASS `torch.nn.Conv1d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True)` [SOURCE]

Applies a 1D 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_{in}, L) and output $(N, C_{\text{out}}, L_{\text{out}})$ can be precisely described as:

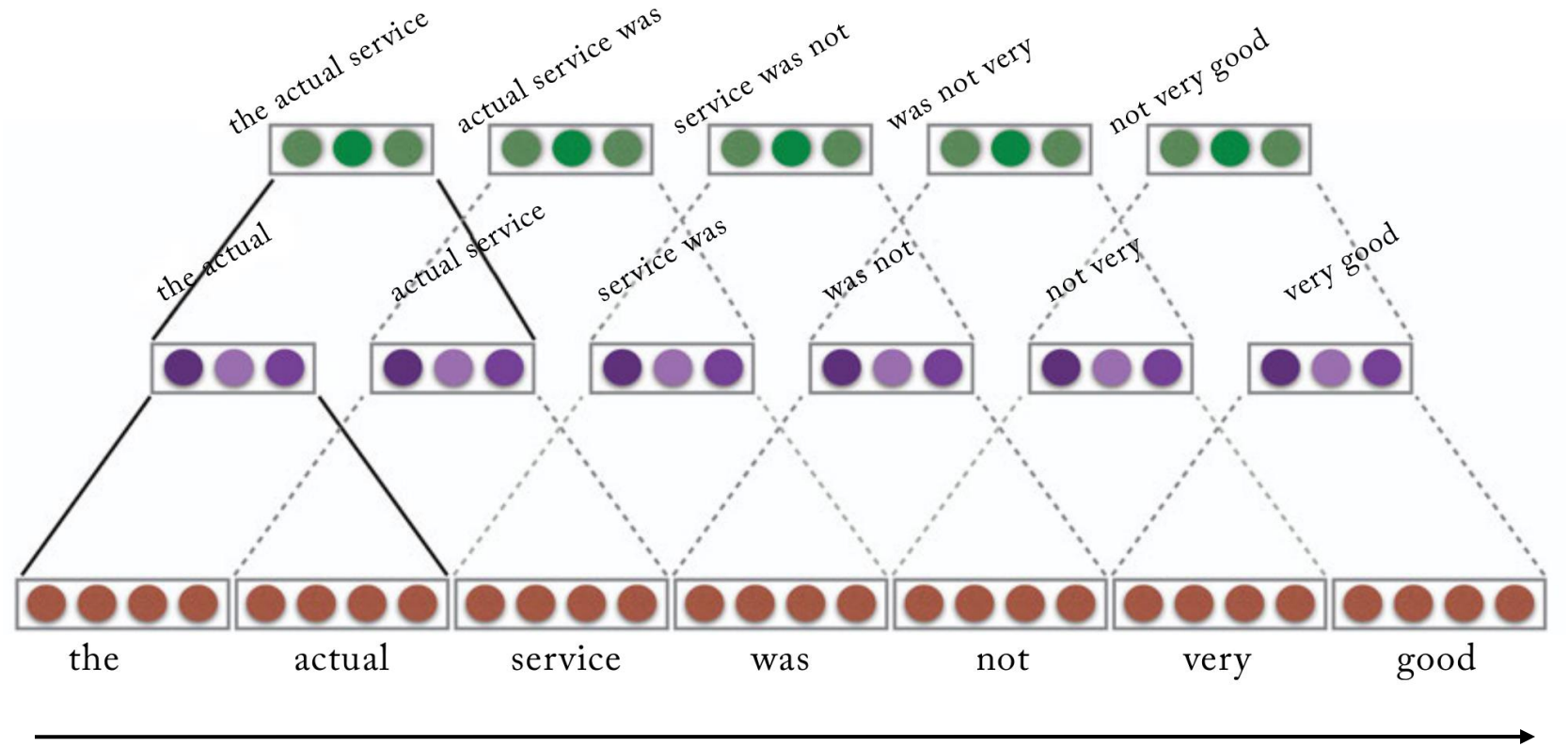
$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{in}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid cross-correlation operator, N is a batch size, C denotes a number of channels, L is a length of signal sequence.

- `stride` controls the stride for the cross-correlation, a single number or a one-element tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for `padding` number of points.
- `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

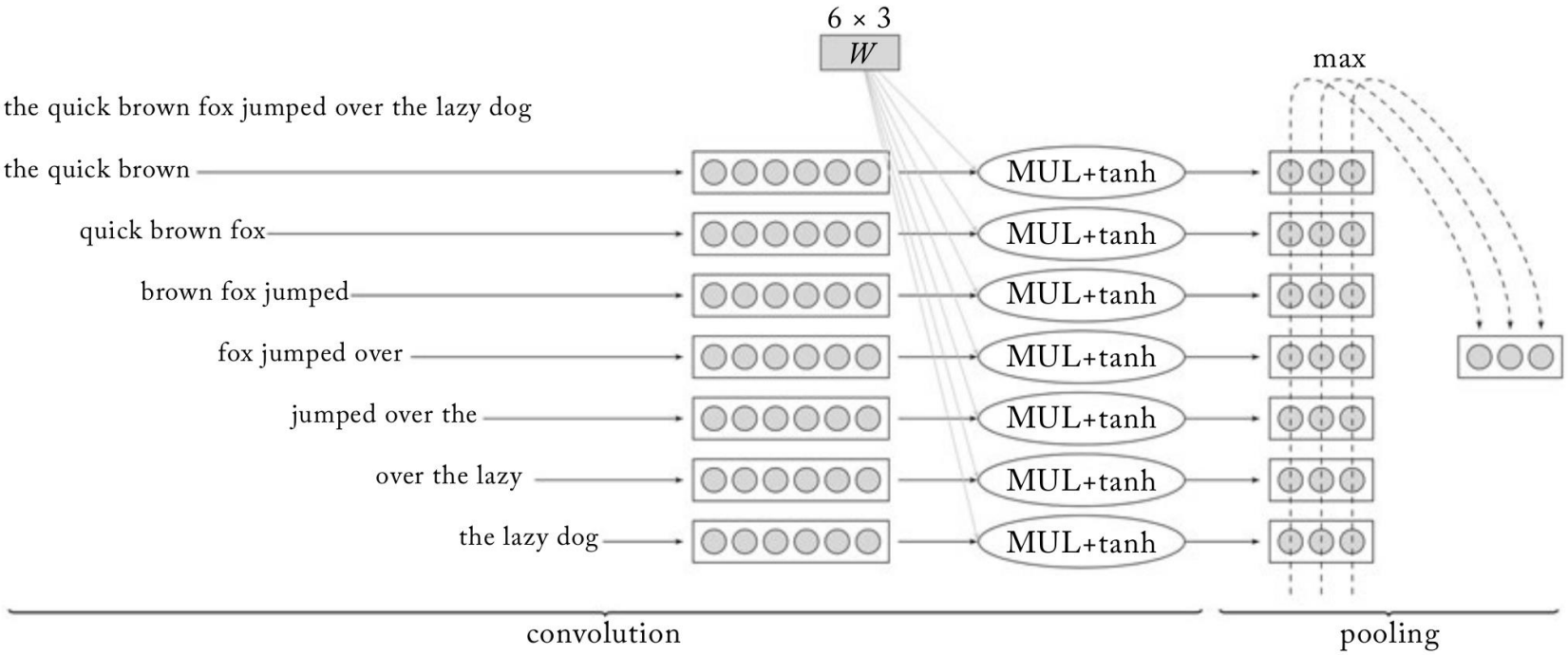
Utiliser des CNNs pour autre chose?

5 "mots" en sortie



7 mots en entrée

Max Pooling Over Time



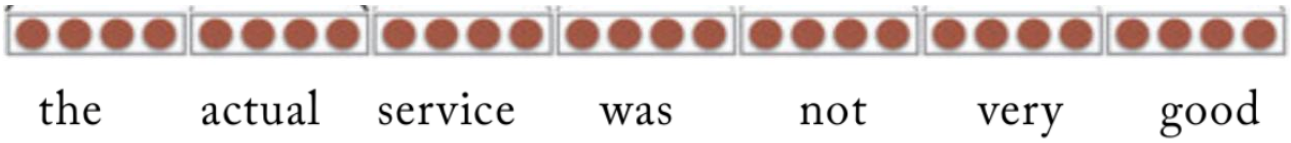
Embeddings

Embedding

CLASS `torch.nn.Embedding(num_embeddings, embedding_dim, padding_idx=None, max_norm=None, norm_type=2.0, scale_grad_by_freq=False, sparse=False, _weight=None)` [SOURCE]

A simple lookup table that stores embeddings of a fixed dictionary and size.

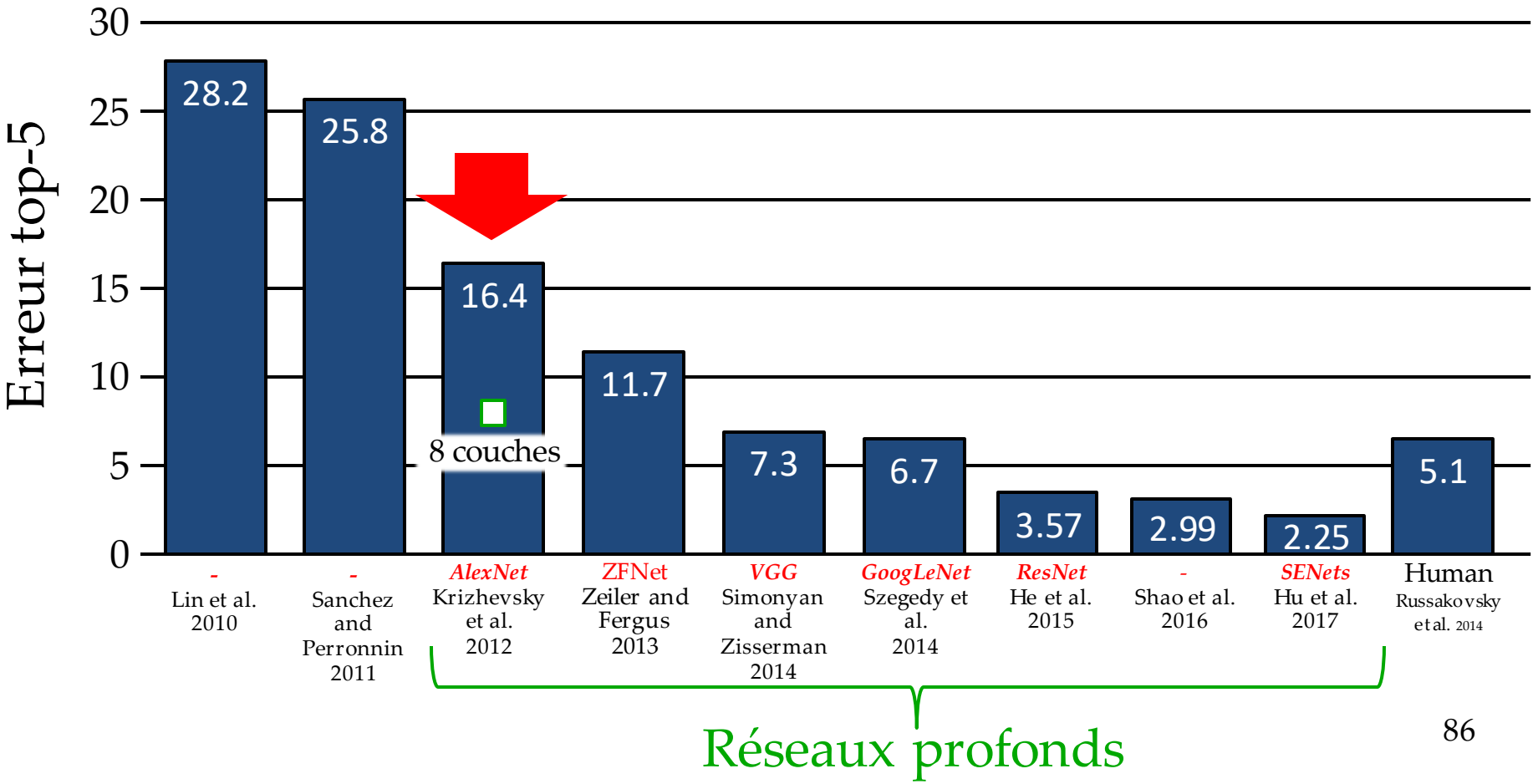
This module is often used to store word embeddings and retrieve them using indices. The input to the module is a list of indices, and the output is the corresponding word embeddings.



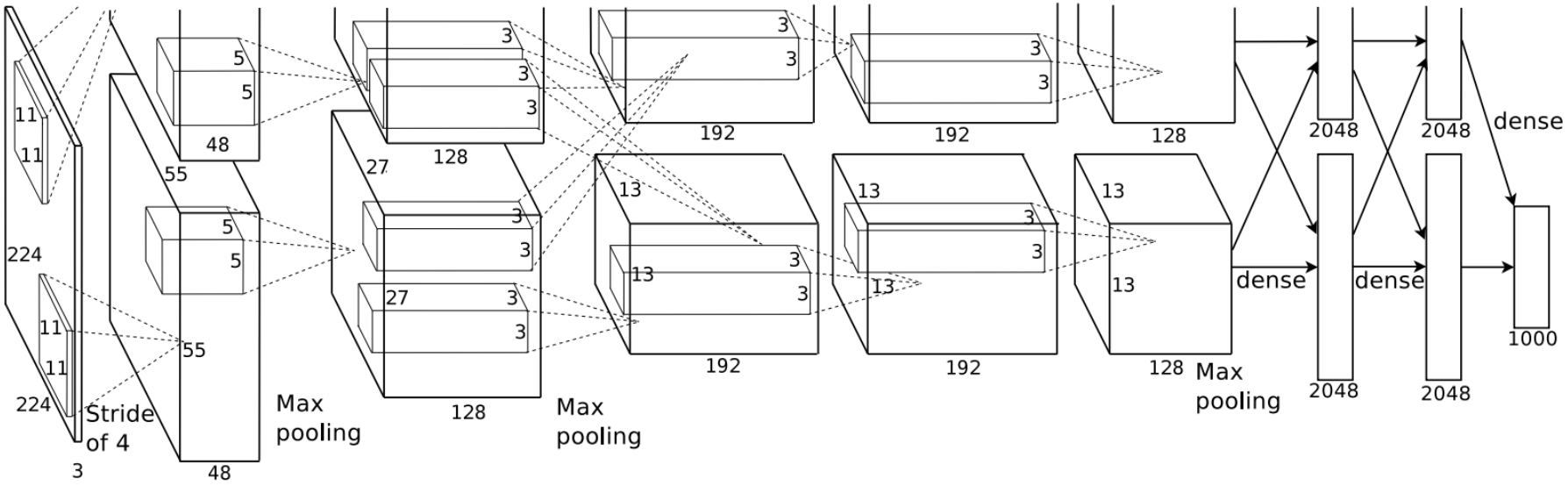
Architectures

Large Scale Visual Recognition Challenge

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



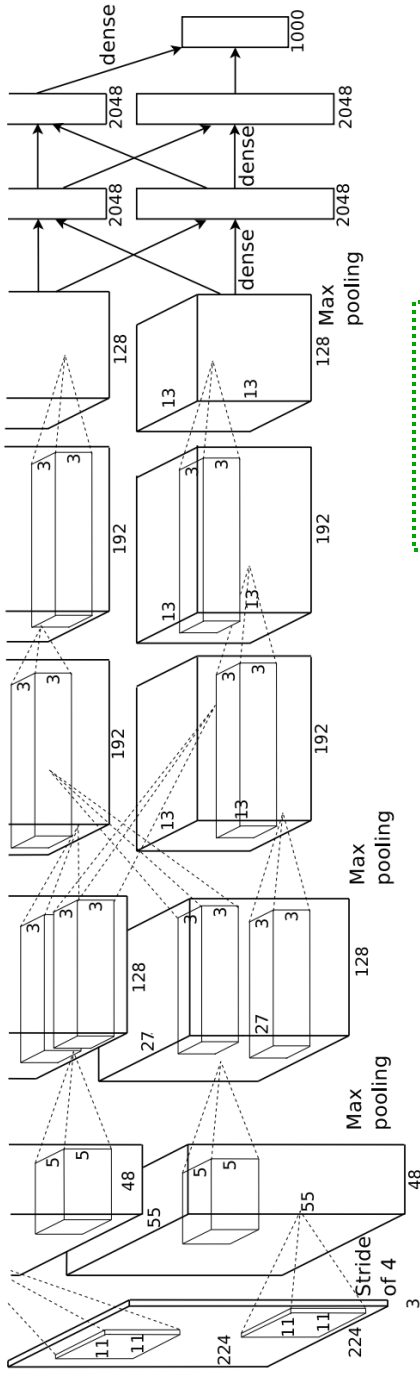
AlexNet



- 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



adapté de : cs231n

majorité des paramètres

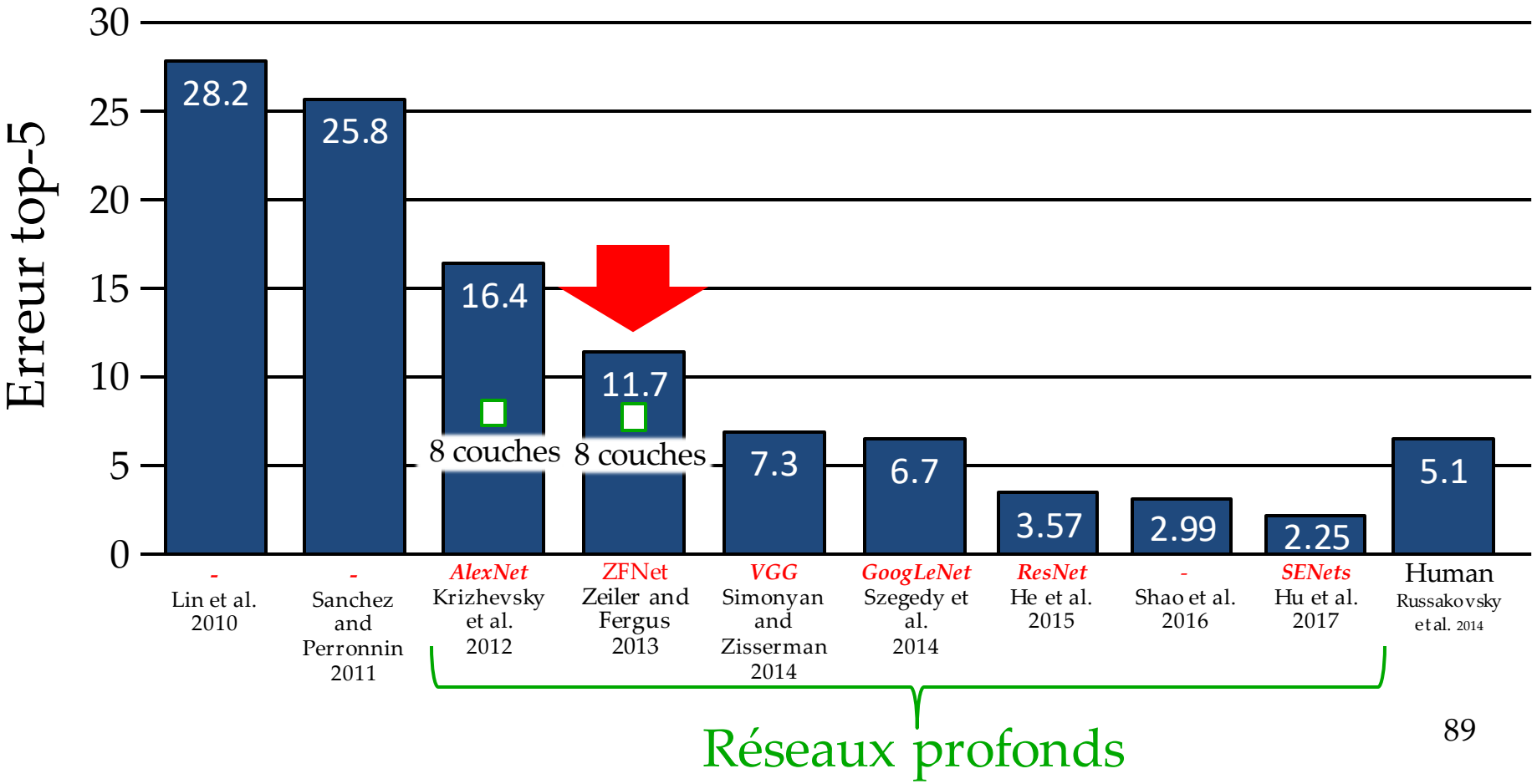
[1000] FC8: 1000 neurons (class scores)
[4096] FC7: 4096 neurons
[4096] FC6: 4096 neurons

Classificateur puissant (besoin dropout)

[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

réduction rapide

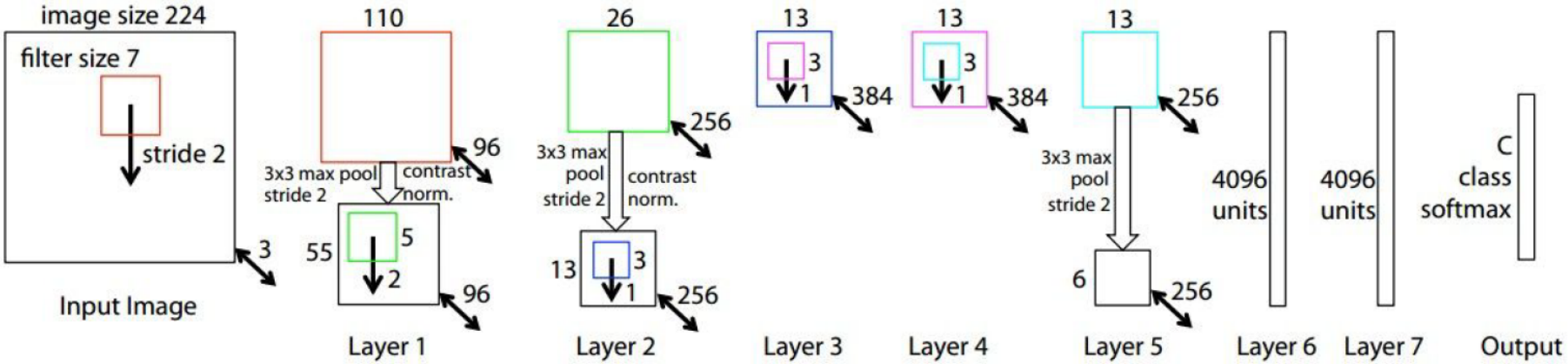
Large Scale Visual Recognition Challenge



ZFNet

ZFNet

[Zeiler and Fergus, 2013]



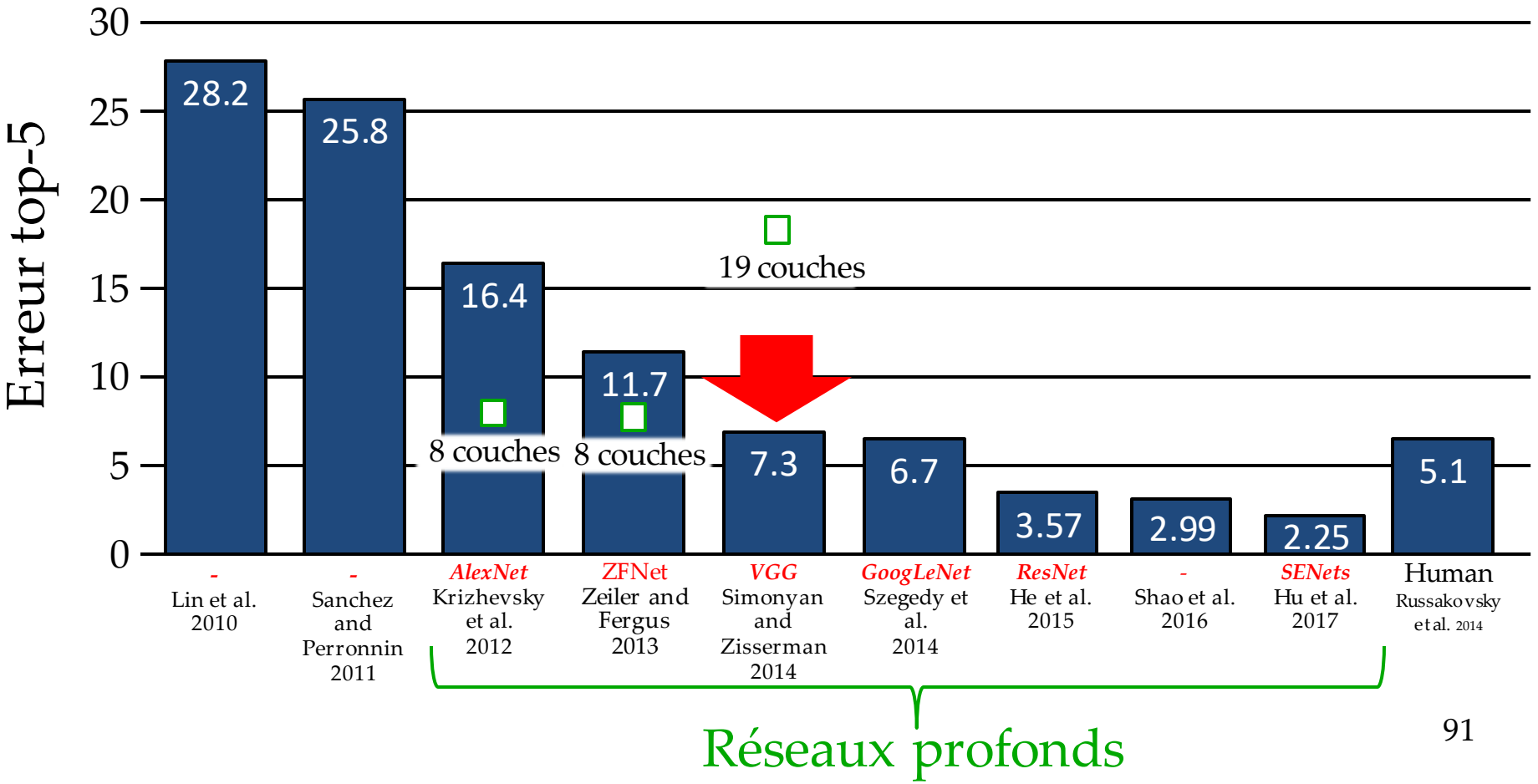
TODO: remake figure

AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 16.4% -> 11.7%

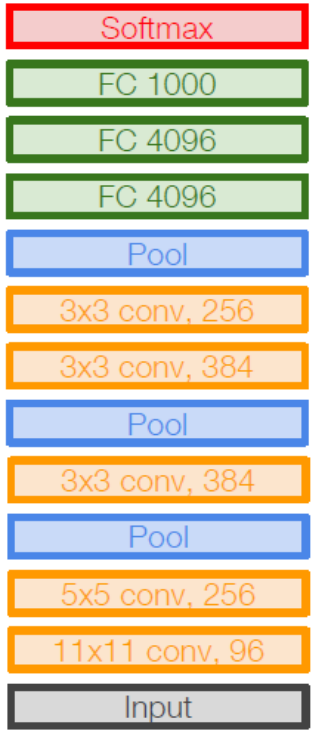
tiré de : cs231n

Large Scale Visual Recognition Challenge



VGGNet

- Toujours 3 couches fully-connected comme classificateur
- 16-19 couches
- 138M paramètres
- Que des convolutions 3x3
- Empilement de 3 convolution 3x3 a le même champ récepteur qu'un filtre 7x7
  - Mais plus de non-linéarité (si ReLU)
  - Moins de paramètres : $3(3^2C^2)$ vs. 7^2C^2 , avec C channels en entrée-sortie
    (économie de 45%)



AlexNet

VGG16

VGG19

adapté de : cs231n

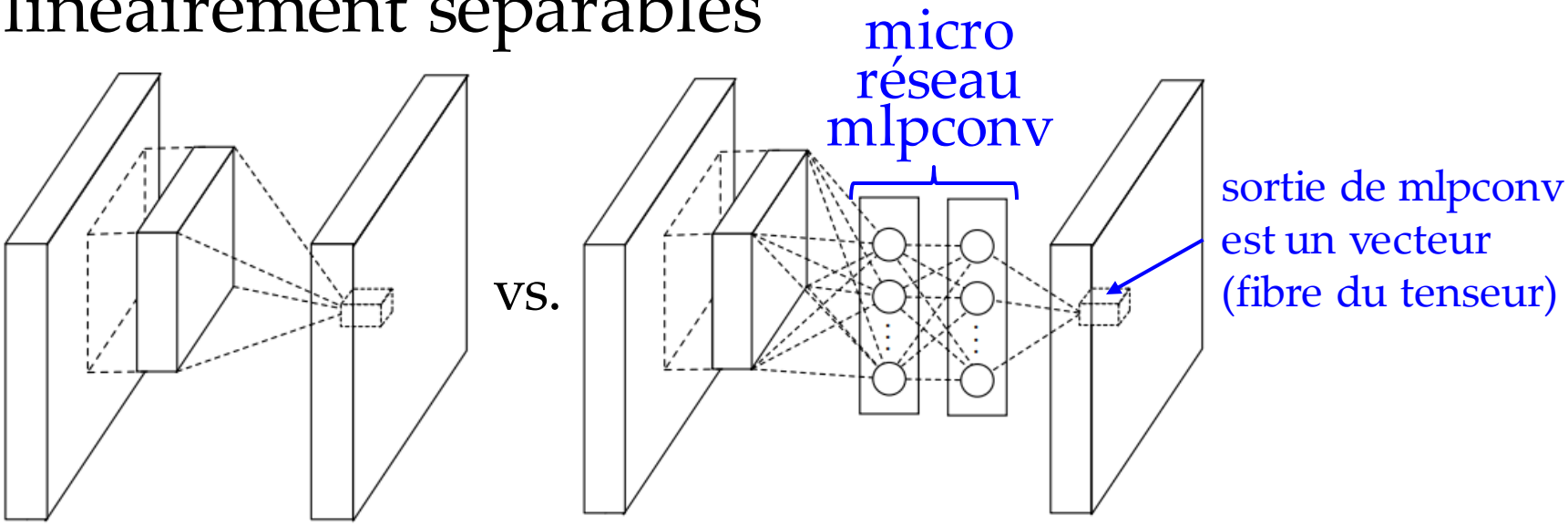
VGGNet

- Procédure complexe d'entraînement
  - entraine petit réseau
  - puis insère des nouvelles couches au milieu, initialisées au hasard
- Procédure inutile :

  It is worth noting that after the paper submission we found that it is possible to initialise the weights without pre-training by using the initialisation procedure of Gloriot & Bengio (2010).

Network In Network (NIN)

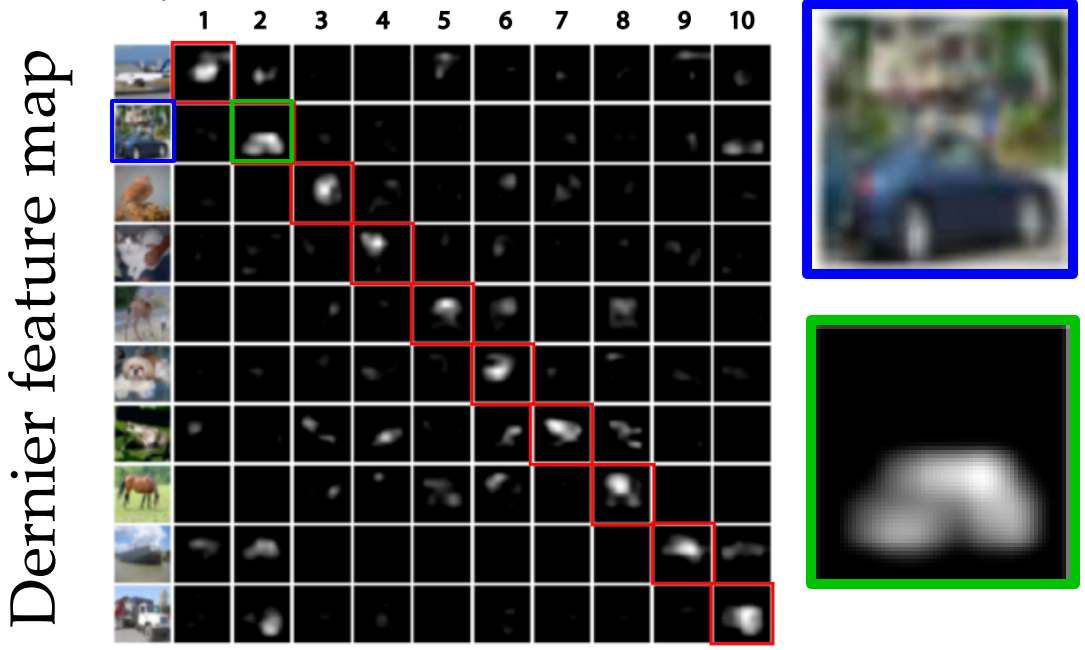
- Les filtres CNN supposent que les *features* sont linéairement séparables

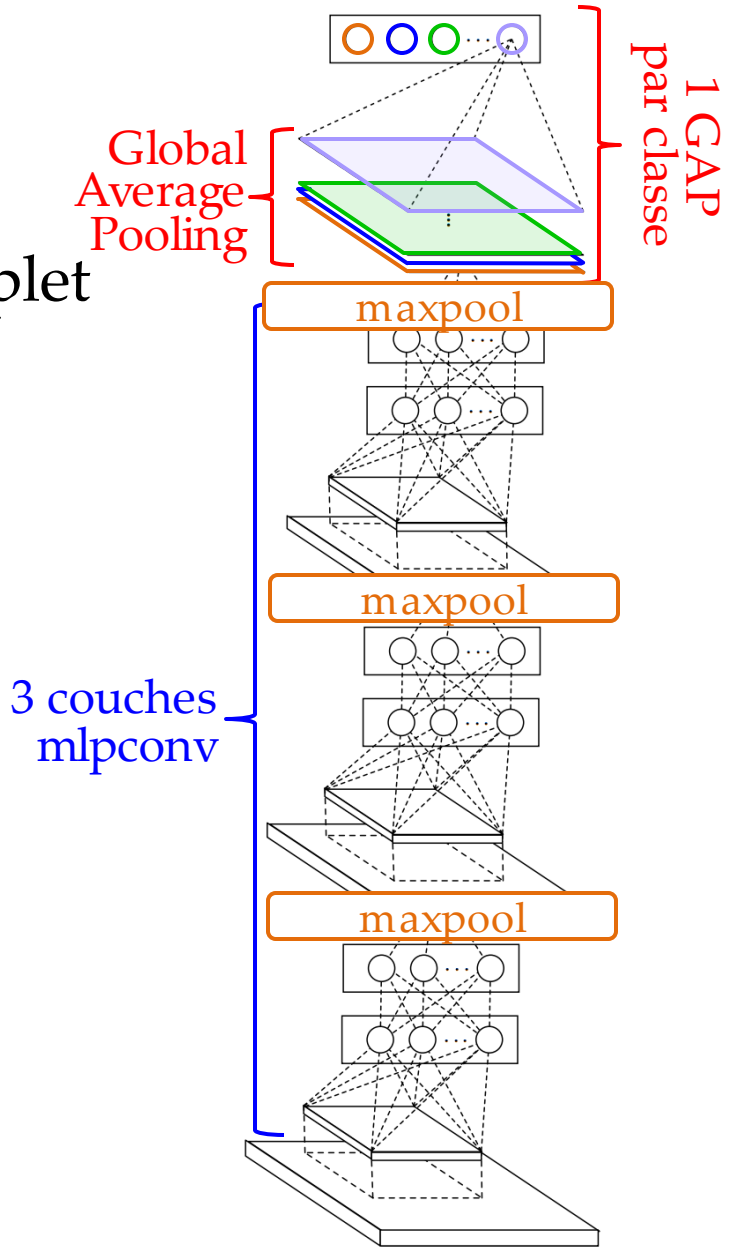


- Remplacé par un micro-réseau de neurones (**`mlpconv`**), qui peut exprimer des fonctions non-linéaires
- Partagés, comme dans les filtres CNN
- Utilisation des convolutions 1x1

NiN

- Introduit le Global Average Pooling (GAP) sur les features map finaux
- Moyenne d'un feature map au complet
- 1 par classe, connecté au softmax
- Force la corrélation entre un *feature map* et une classe :





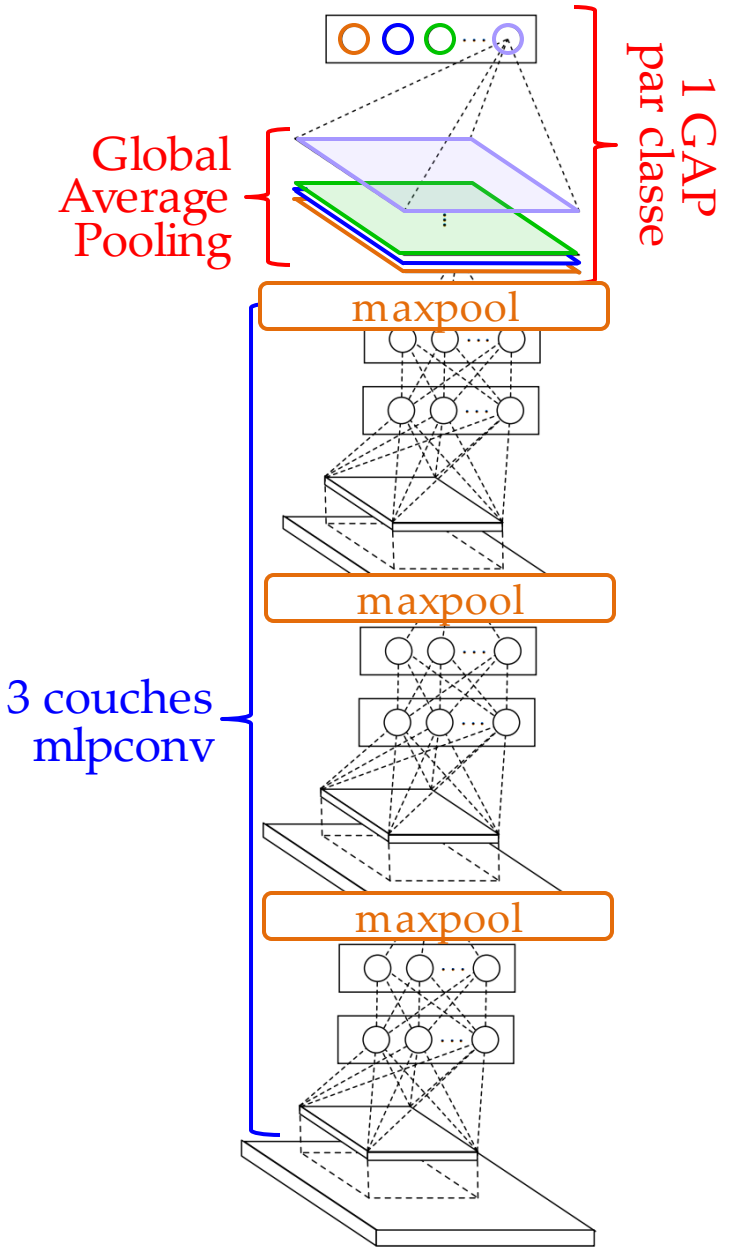
- Facilite l'interprétation des erreurs du réseau

NiN

- GAP agit comme régularisateur structurel

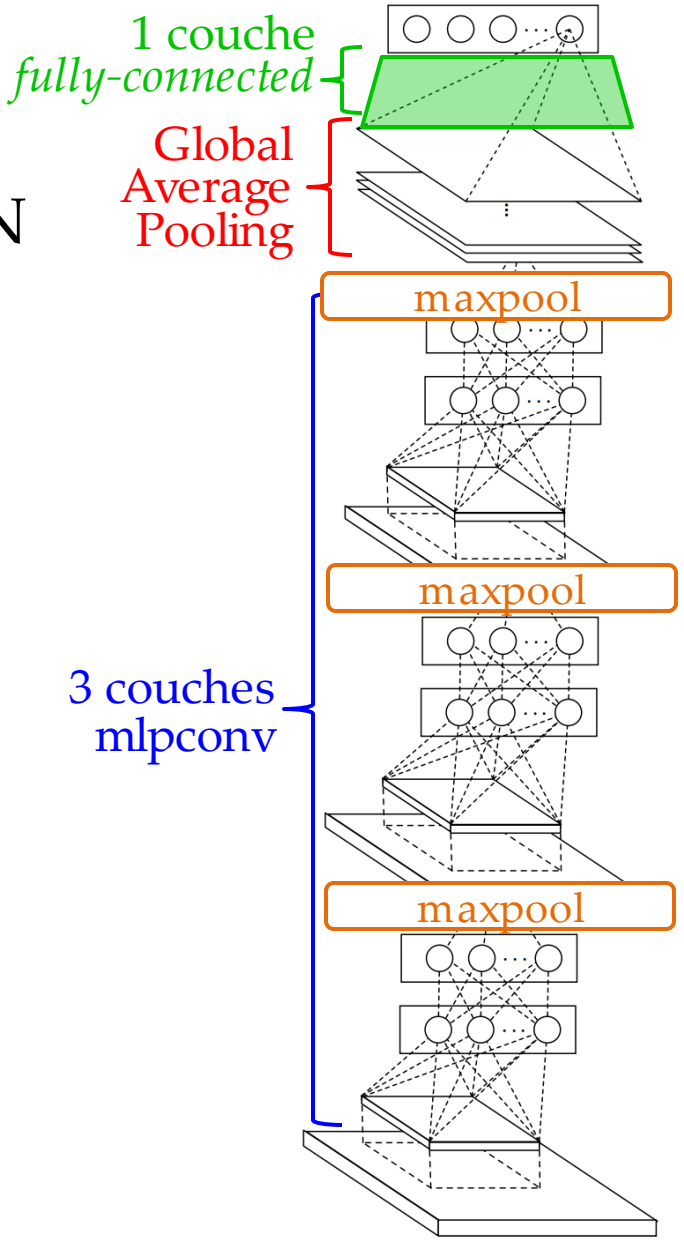
Method	Testing Error
mlpconv + Fully Connected	11.59%
mlpconv + Fully Connected + Dropout	10.88%
mlpconv + Global Average Pooling	10.41%

- Puissance d'extraction des filtres micro-réseaux améliore tellement la qualité des *features* que le classificateur n'est plus nécessaire
- Dropout sur les sorties mlpconv 1 et 2



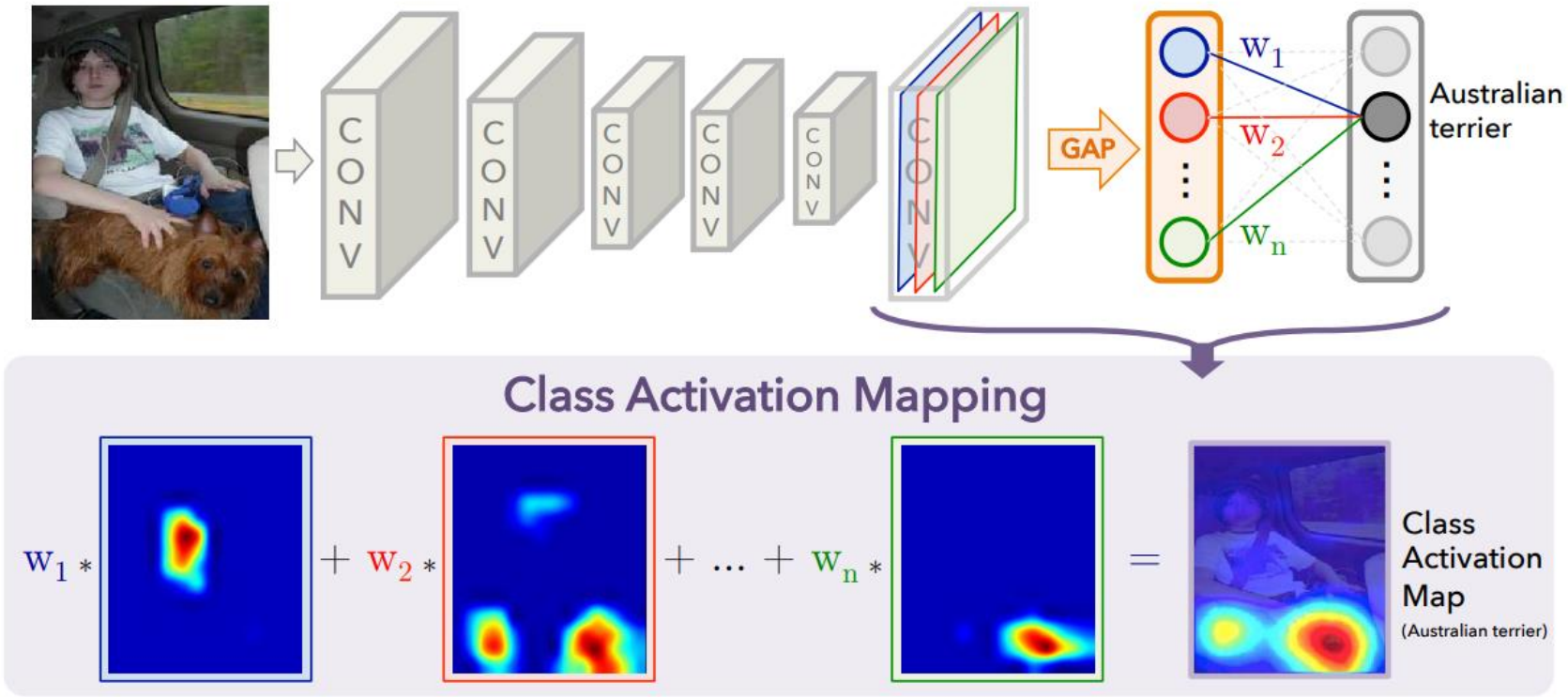
NiN

- Certaines implémentation de NiN semblent utiliser une couche de fully connected comme classificateur (à vérifier!)
  - Beaucoup moins de paramètres
  - Beaucoup moins d'overfit
- Take-home message reste le même : plus besoin d'un classificateur puissant en sortie
- Tendance forte des prochaines architectures



GAP : localisation d'objet gratuite!

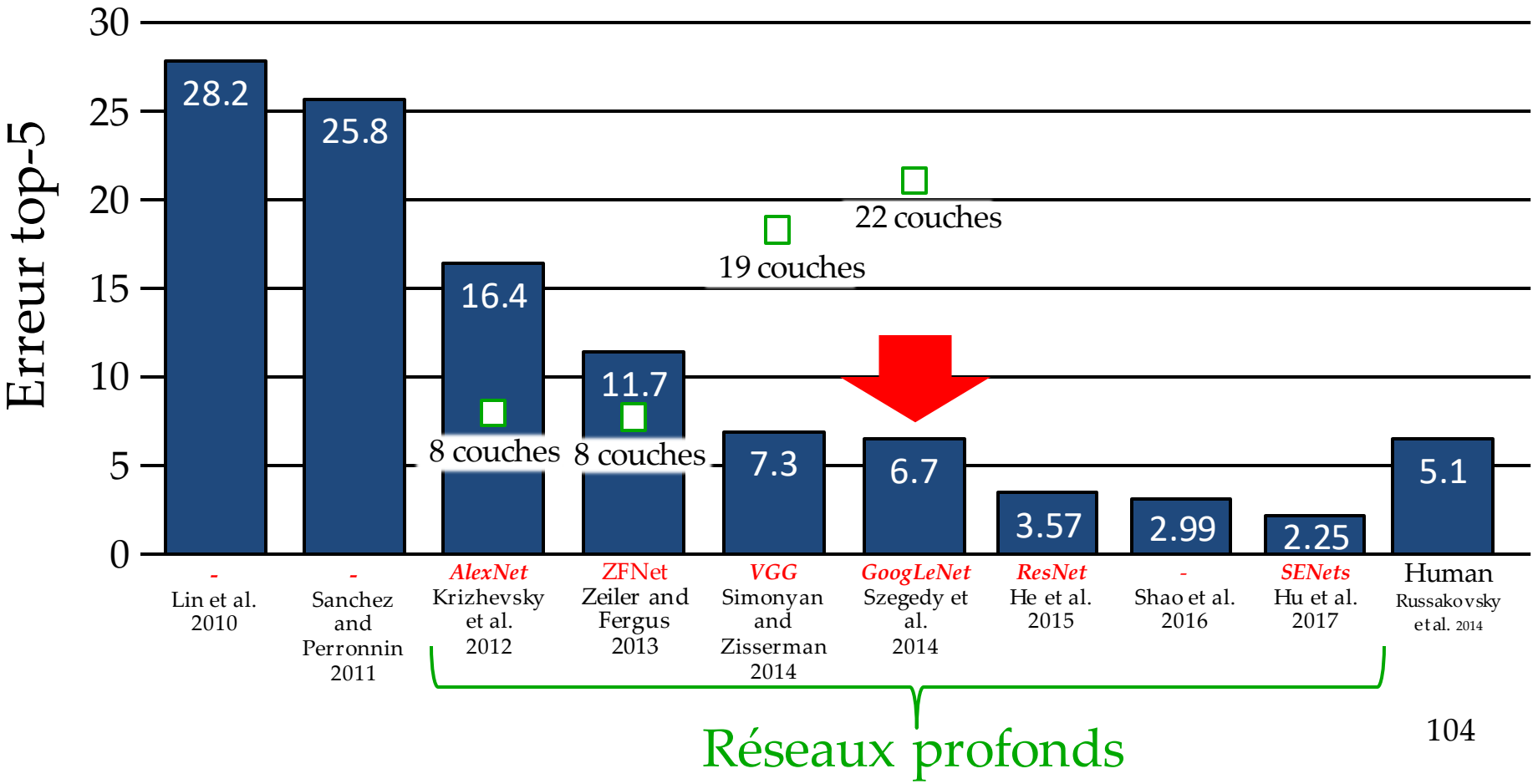
- http://cnnlocalization.csail.mit.edu/



- Donne une certaine interprétabilité aux résultats

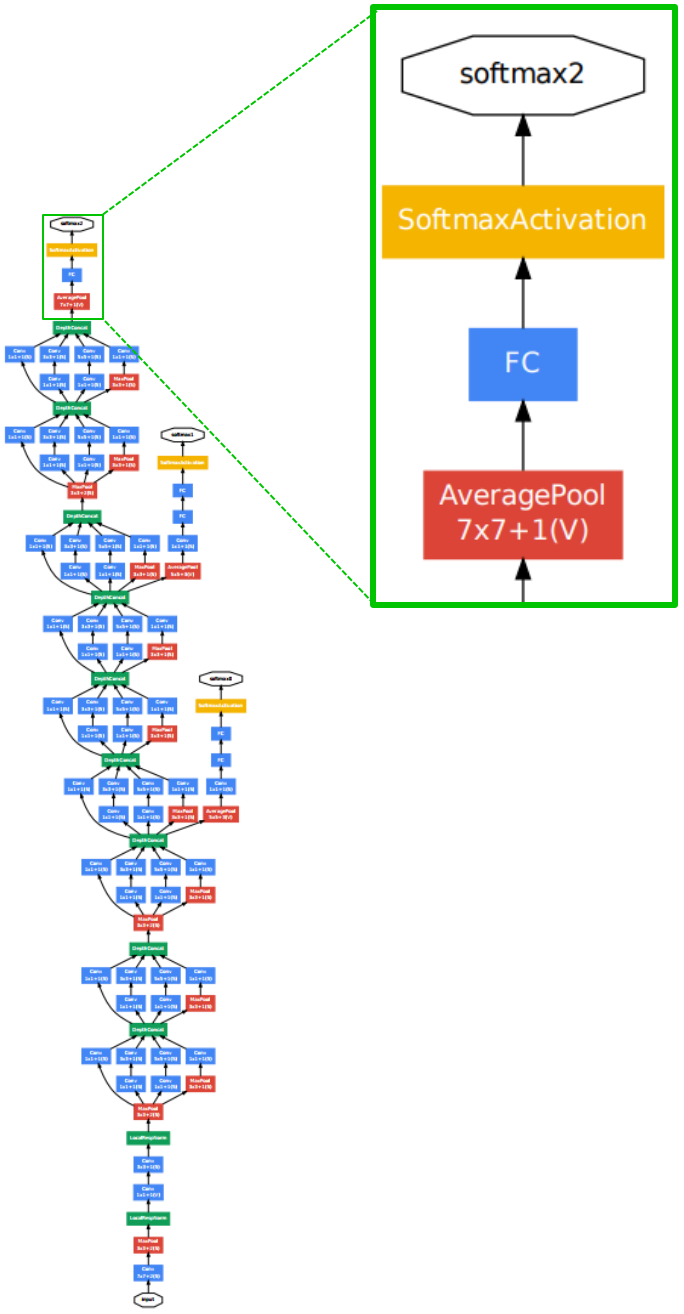
Zhou et al., Learning Deep Features for Discriminative Localization, CVPR 2016.

Large Scale Visual Recognition Challenge



GoogLeNet

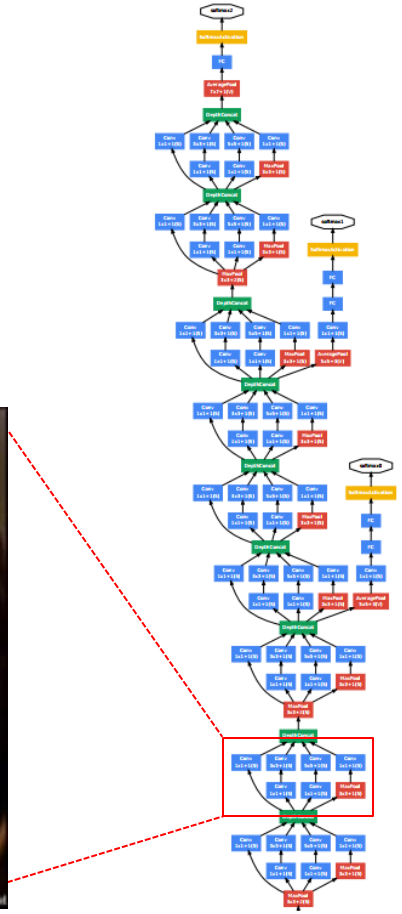
- Réseau plus profond (22 couches)
- Seulement 5 millions de paramètres, 12 fois moins qu'AlexNet
- Toujours pas de batch norm
- GAP + une couche fully-connected (tendance classificateur faible)
- La couche *fully connected* ne sert (au dire des auteur) qu'à adapter les *features* finaux vers le nombre de sorties (*labels*) désirées.



Szegedy et al., Going Deeper with Convolutions, CVPR 2015.

GoogLeNet

- Emphase sur minimisation des calculs via modules *Inception*

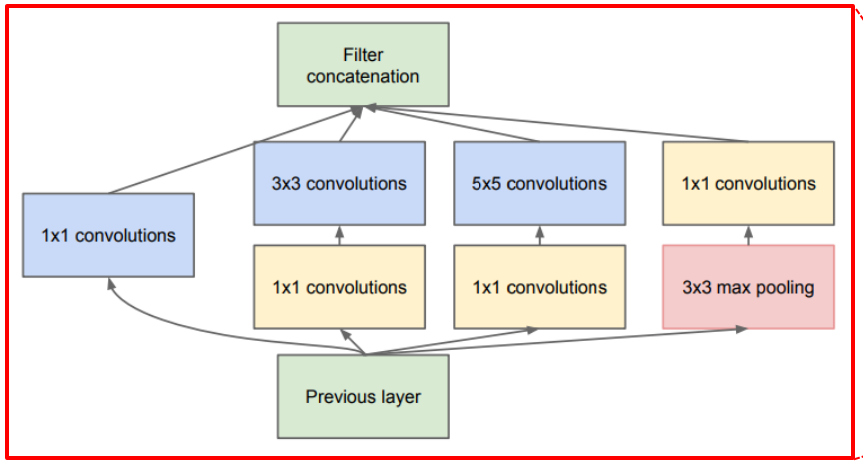


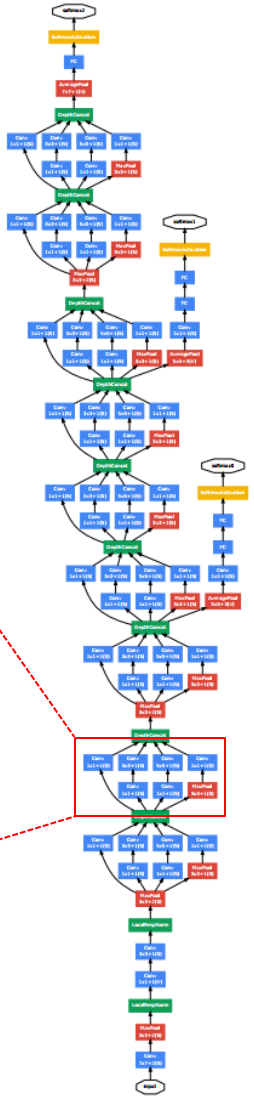
References

[1] Know your meme: We need to go deeper. http://knowyourmeme.com/memes/we-need-to-go-deeper. Accessed: 2014-09-15.

GoogLeNet

- Emphase sur minimisation des calculs via modules *Inception*

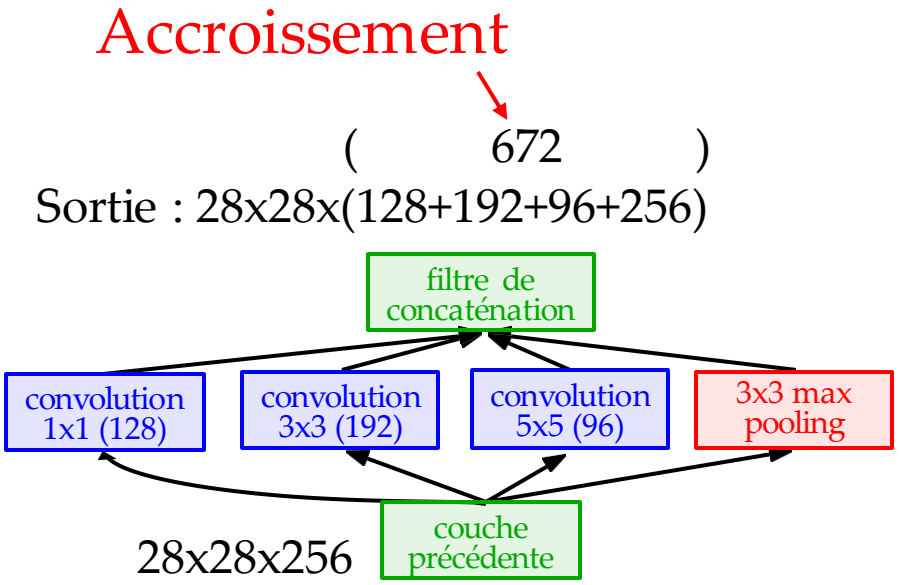




- S'éloigne ainsi de l'approche convolution 1 taille de filtre suivie de maxpool

GoogLeNet

- Idée de base : avoir des filtres en parallèle avec des champs récepteurs de taille multiple (pyramide spatiale)
- La couche suivante a donc accès à des *features* à plusieurs échelles spatiales
- Version naïve :



Coût en calcul

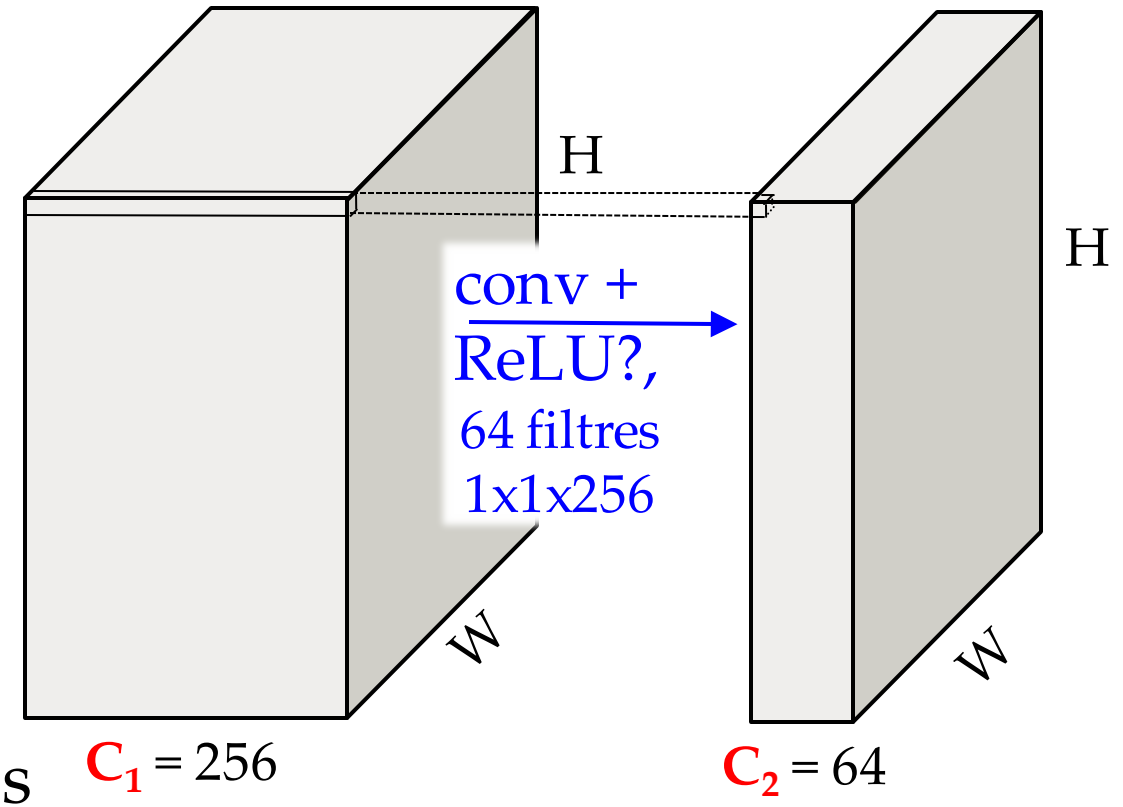
Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256
Total: 854M ops

Détail des calculs dans la vidéo Stanford
https://youtu.be/DAOcjicFr1Y?t=1717

adapté de : cs231n, Université Stanford

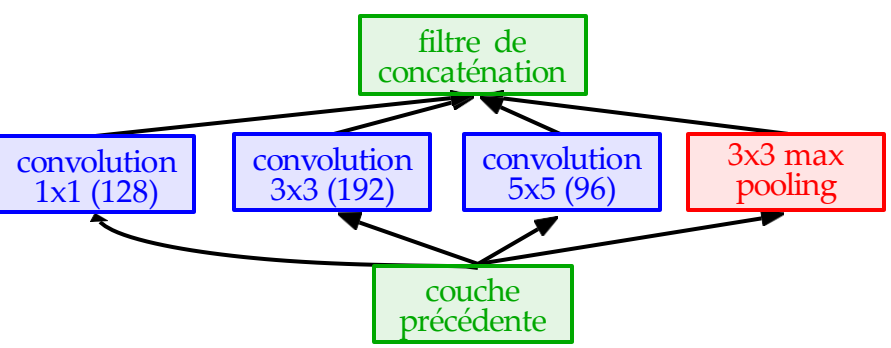
Convolution 1x1

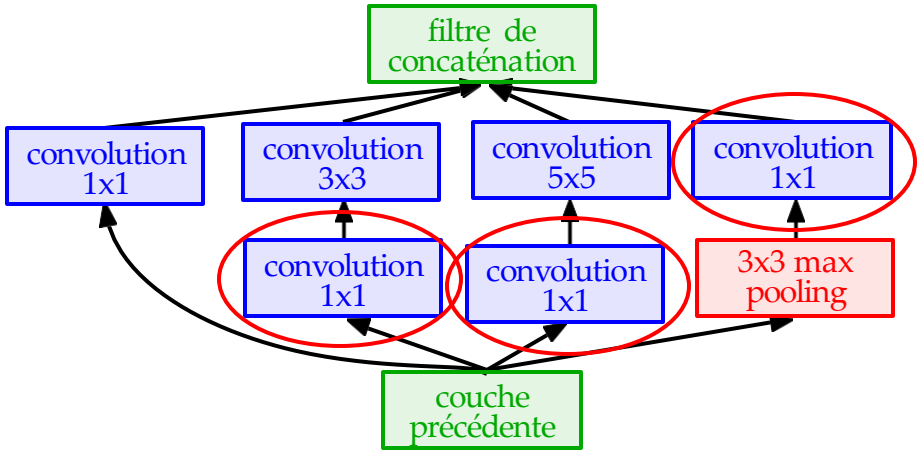
- Origine dans NiN
- Popularisées par GoogLeNet (prochaine architecture)
- Semble inutile…
- Rappel : 1x1x**C**
- Préserve les dimensions H, W
- Sert à réduire le nombre de dimensions **C**, via une projection linéaire (style PCA)
- *Fully-connected* sur les features
- Forme de bottleneck



GoogLeNet

- Ajout de convolutions 1x1 comme bottleneck
- Permet de choisir la dimension d'entrée des opérations de convolution couteuses

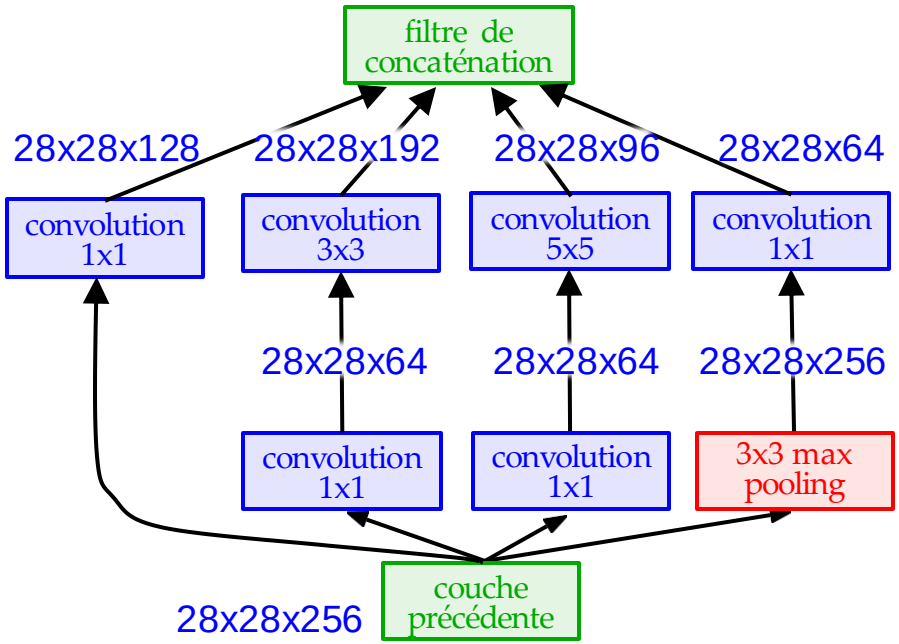




adapté de : cs231n, Université Stanford

GoogLeNet

- Fera diminuer :
  - nombre de calcul
  - dimension en sortie



Coût en calcul

Conv Ops:
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: 358M ops

Passe de 854Mops à 358 Mops pour cet exemple

Détail des calculs dans la vidéo Stanford

adapté de : cs231n, Université Stanford

GoogLeNet

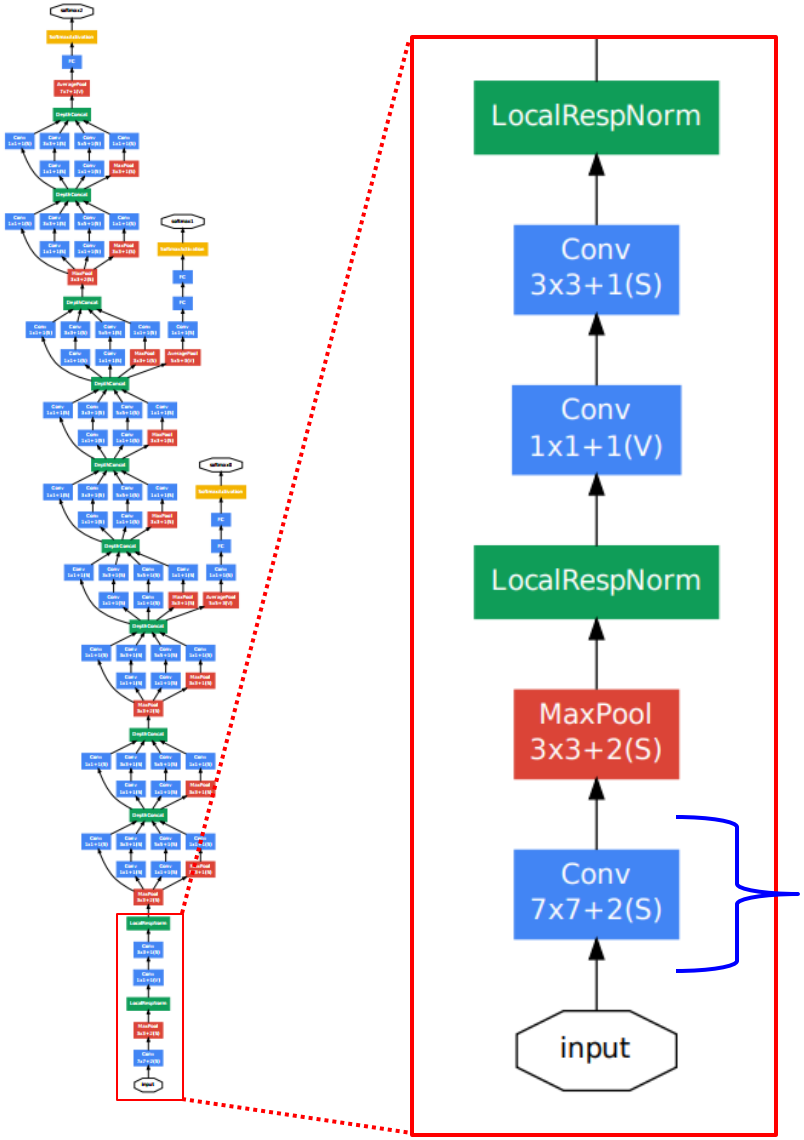
Entrée

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Augmente le nombre de filtre selon la distance de l'entrée

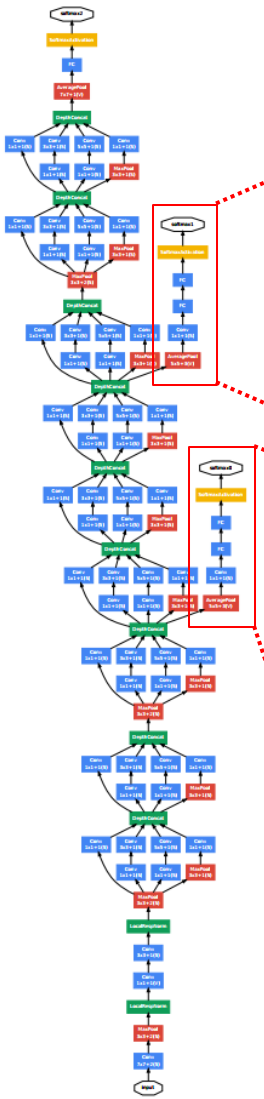
Sortie

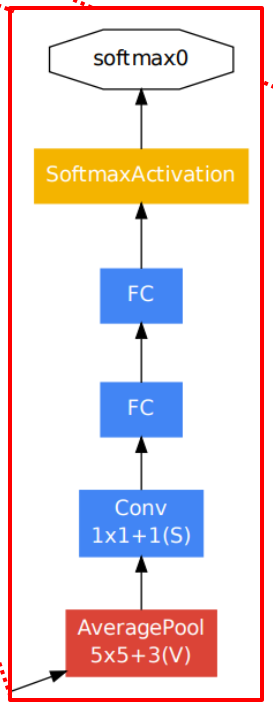
GoogLeNet : base du réseau

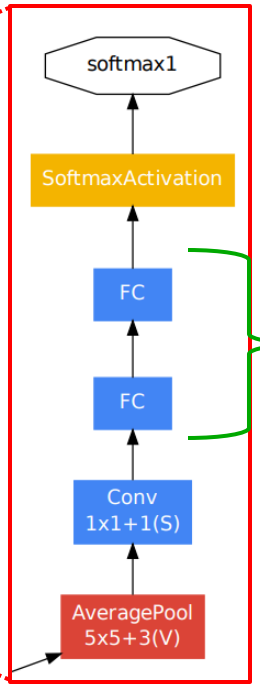


convolution plus large à la base
AlexNet : 11x11

GoogLeNet : têtes auxiliaires







deux couches fully-connected, besoin d'un classificateur plus puissant car features moins évolués

- Pour combattre le vanishing gradient
- Fonction des pertes sur softmax0 et softmax1 multipliée par 0.3
- Ne contribue que pour 0.5% de gain

Fin Partie I