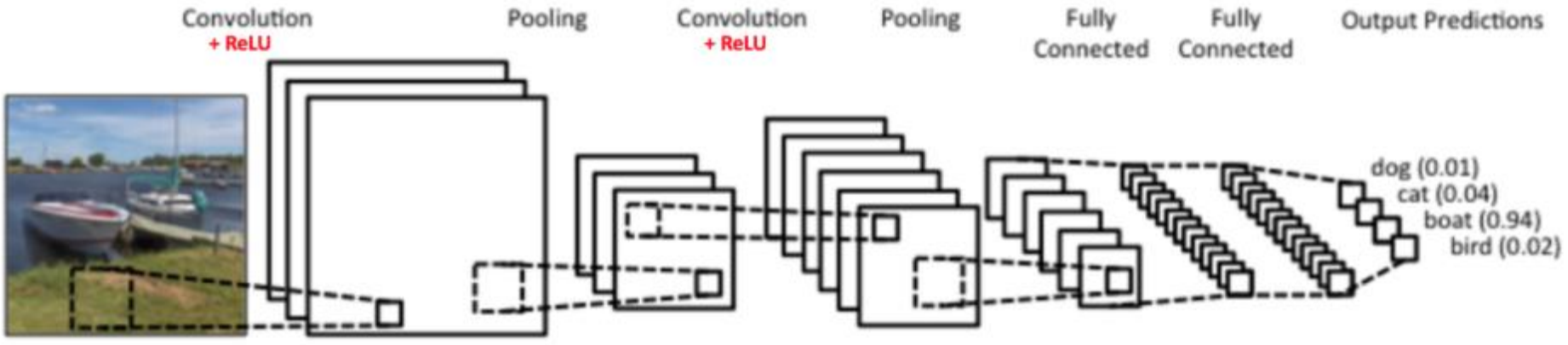
LES RÉSEAUX CONVOLUTIFS

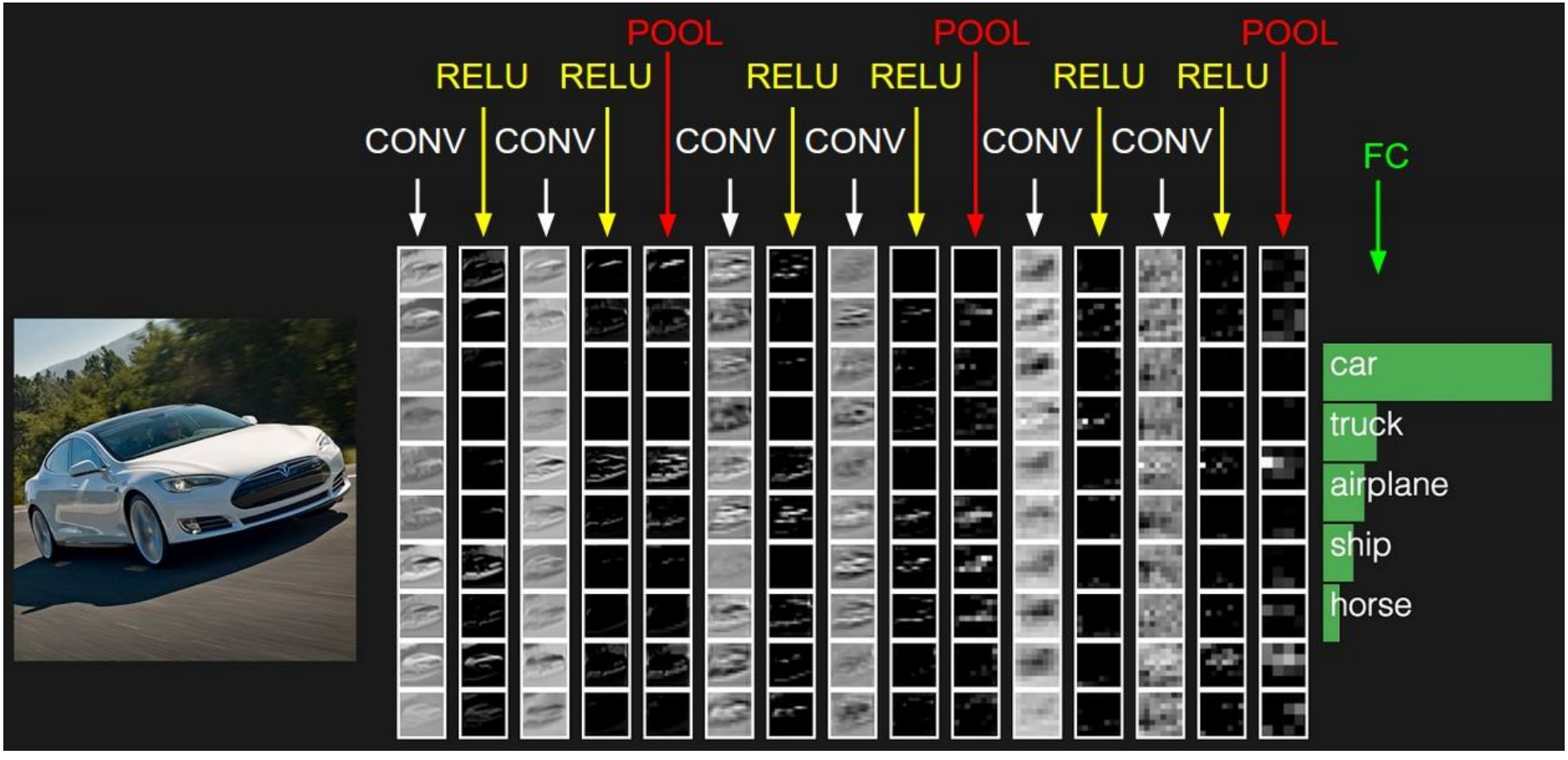
Réseaux Convolutifs

Modification du perceptron multi-couches, avec des opérations supplémentaires. Blocs possibles :

- couches convolutives (+ fonction non linéaire ReLU)
- couches d'agrégation ou "pooling"
- couches totalement connectées



EXEMPLE


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

La convolution

Le produit de convolution de 2 matrices revient à multiplier point par point chaque sous-matrice de la première par la seconde et additionner le total “au centre”.

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

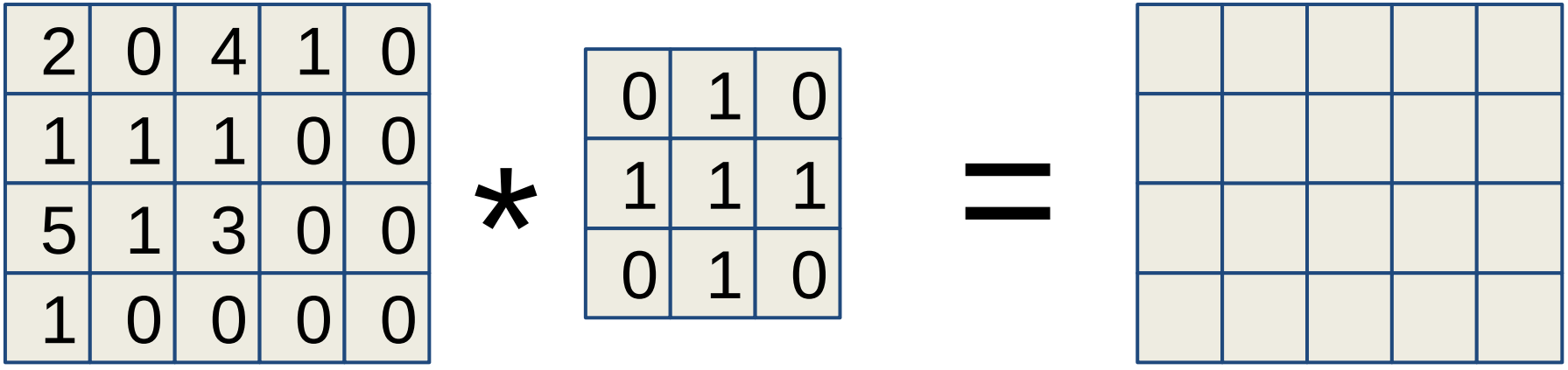
*

0	1	0
1	1	1
0	1	0

= ?

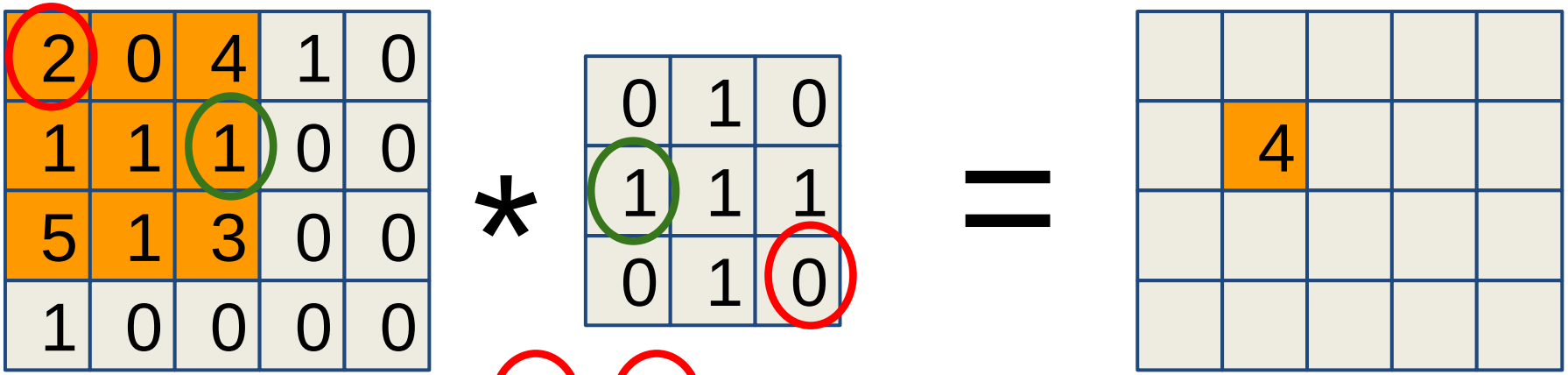
La convolution

Le produit de convolution de 2 matrices revient à multiplier point par point chaque sous-matrice de la première par la seconde et additionner le total “au centre”.



La convolution

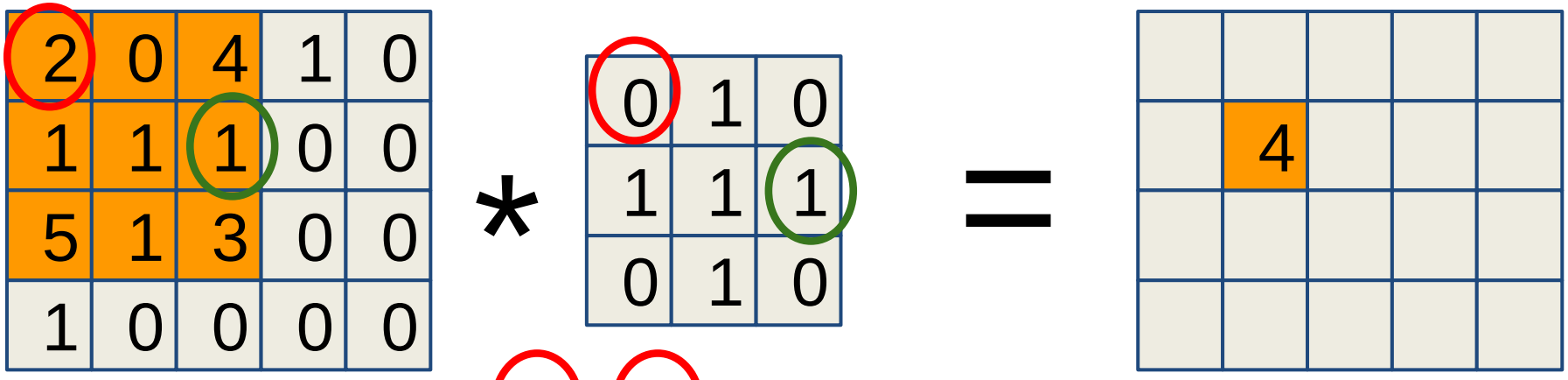
Le produit de convolution de 2 matrices revient à multiplier point par point chaque sous-matrice de la première par la seconde “retournée” et additionner le total “au centre”.



2 x 0 + 0 x 1 + 4 x 0
+ 1 x 1 + 1 x 1 + 1 x 1
+ 5 x 0 + 1 x 1 + 3 x 0 = 4

La convolution

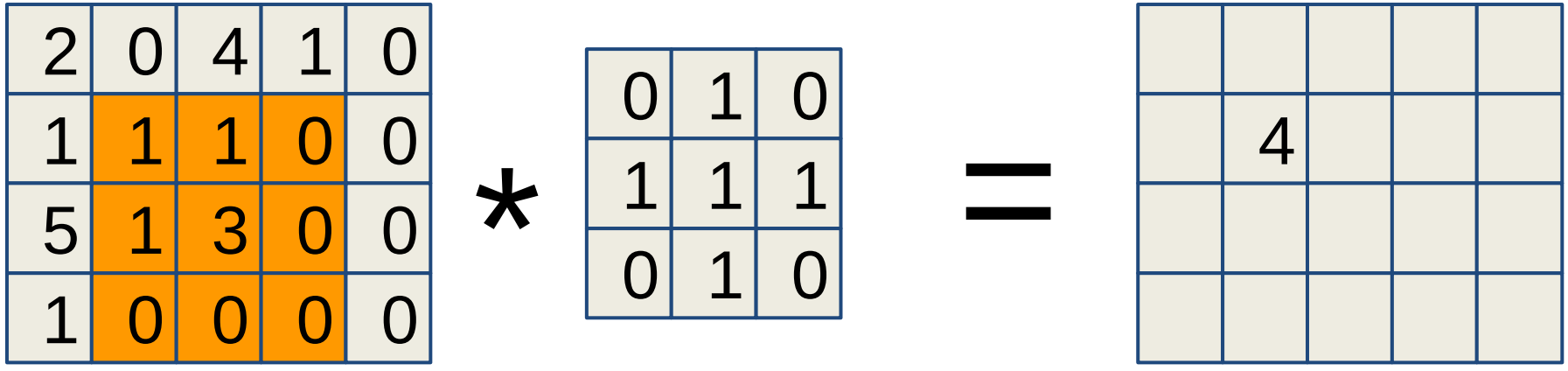
En pratique, la matrice par laquelle on multiplie, qu'on appelle "filtre" ou encore "noyau", est symétrique. Ce qui nous simplifie la vie:



2 x 0 + 0 x 1 + 4 x 0
+ 1 x 1 + 1 x 1 + 1 x 1
+ 5 x 0 + 1 x 1 + 3 x 0 = 4

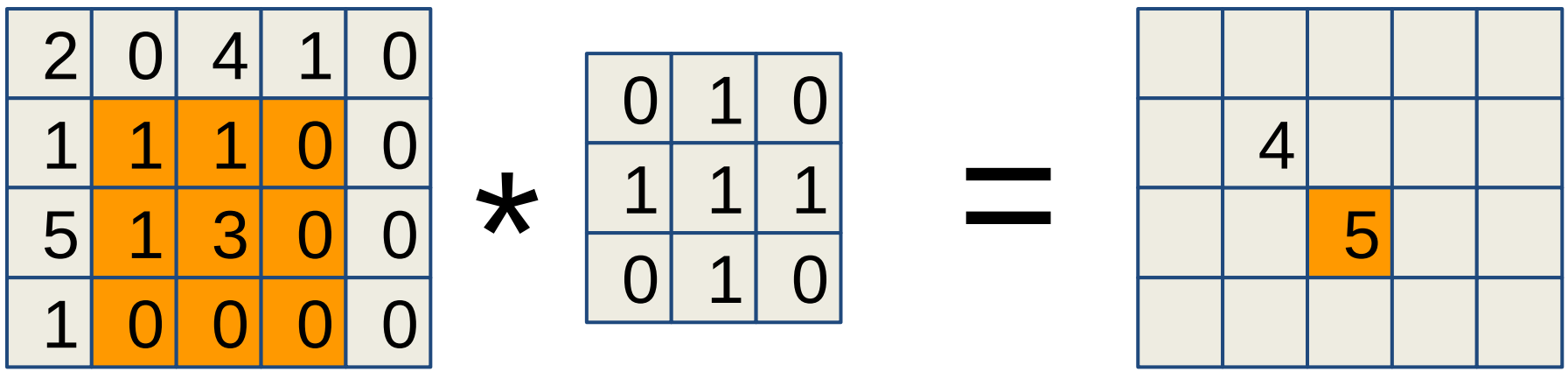
La convolution

A vous...



La convolution

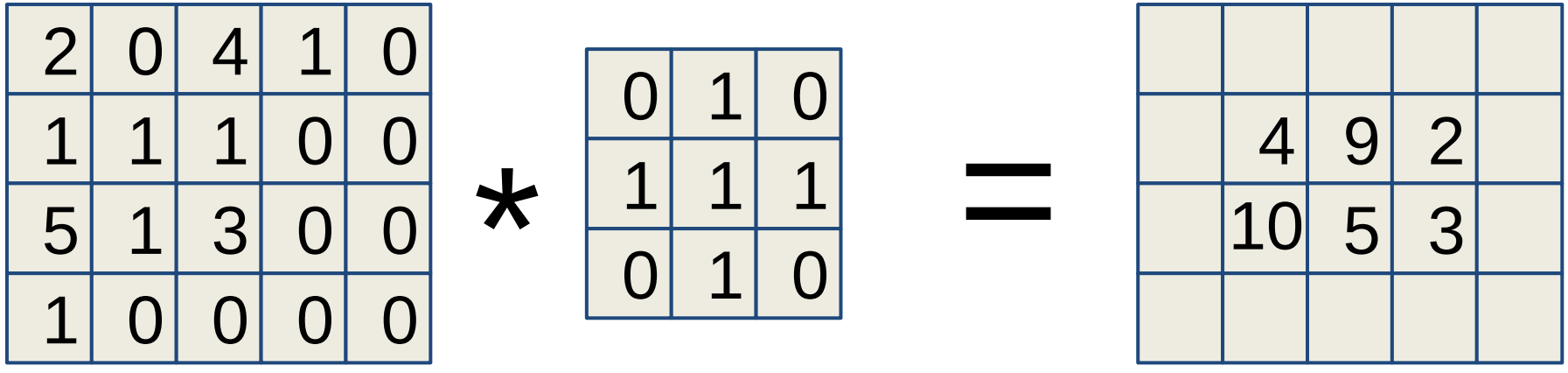
A vous...



1 x 0 + 1 x 1 + 0 x 0
+ 1 x 1 + 3 x 1 + 0 x 1
+ 0 x 0 + 0 x 1 + 0 x 0 = 5

La convolution

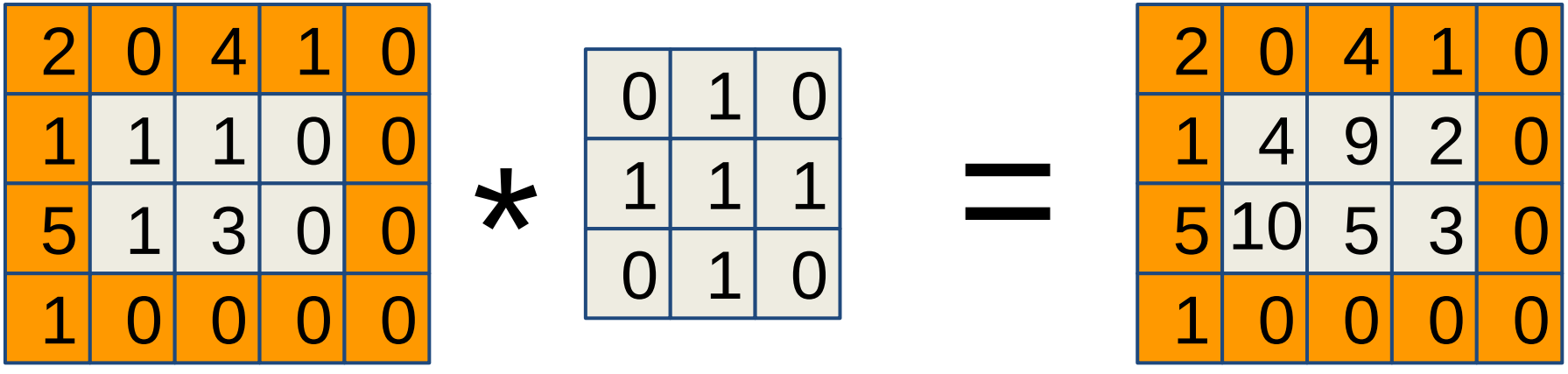
A vous...



La convolution

Et les bords ? Plusieurs solutions:

- Ne rien faire, garder les valeurs initiales.
- Appliquer un filtre spécial.
- Copier les valeurs obtenues sur les pixels adjacents.



La convolution

Application aux images:

Image de départ

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

*

Filtre / Noyau

0	1	0
1	1	1
0	1	0

=

Image modifiée

2	0	4	1	0
1	4	9	2	0
5	10	5	3	0
1	0	0	0	0

Filtres classiques

0	0	0
0	1	0
0	0	0

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

1	2	1
2	4	2
1	2	1

0	-1	0
-1	4	-1
0	-1	0

0	-1	0
-1	5	-1
0	-1	0

1	-1

-1	-1	-1
-1	8	-1
-1	-1	-1

Filtres classiques

Identité

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

1	2	1
2	4	2
1	2	1

0	-1	0
-1	4	-1
0	-1	0

0	-1	0
-1	5	-1
0	-1	0

1	-1

-1	-1	-1
-1	8	-1
-1	-1	-1

Filtres classiques

Identité

Flou moyen

1	2	1
2	4	2
1	2	1

0	-1	0
-1	4	-1
0	-1	0

0	-1	0
-1	5	-1
0	-1	0

1	-1

-1	-1	-1
-1	8	-1
-1	-1	-1

Filtres classiques

Identité

0	-1	0
-1	5	-1
0	-1	0

Flou moyen

1	-1

Flou gaussien

-1	-1	-1
-1	8	-1
-1	-1	-1

0	-1	0
-1	4	-1
0	-1	0

Filtres classiques

Identité

0	-1	0
-1	5	-1
0	-1	0

Flou moyen

1	-1

Flou gaussien

-1	-1	-1
-1	8	-1
-1	-1	-1

Laplacien

Filtres classiques

Identité

0	-1	0
-1	5	-1
0	-1	0

Flou moyen

1	-1

Flou gaussien

-1	-1	-1
-1	8	-1
-1	-1	-1

Laplacien

Laplacien normalisé

Filtres classiques

Identité

Flou moyen

Flou gaussien

Laplacien

0	-1	0
-1	5	-1
0	-1	0

1	-1

Laplacien II normalisé

Laplacien normalisé

Filtres classiques

Identité

Flou moyen

Flou gaussien

Laplacien

0	-1	0
-1	5	-1
0	-1	0

Vertical edge detection

Laplacien II normalisé

Laplacien normalisé

Filtres classiques

Identité

Flou moyen

Flou gaussien

Laplacien

Sharpen

Vertical edge detection

Laplacien II normalisé

Laplacien normalisé

Cumul de filtres

On peut créer des filtres en combinant d'autres filtres:

=

+

0	-1	0
-1	5	-1
0	-1	0

0	0	0
0	1	0
0	0	0

0	-1	0
-1	4	-1
0	-1	0

Normalisation

Une fois le filtre appliqué, on peut se retrouver avec des pixels négatifs, ou supérieurs à 255. Il faut donc normaliser la nouvelle matrice pour que les pixels se trouvent entre 0 et 255. En pratique, on applique à chaque pixel l'opération:

$$new = \frac{old - \min}{\max - \min} \times 255$$

Mais si l'image contient beaucoup d'outliers, ils vont avoir une trop grande influence. Pour y remédier, on peut remplacer max et min par les 95ème et 5ème quantiles, par exemple.

Normalisation

Non normalisé

Normalisé min/max

Normalisé 0.05/0.95

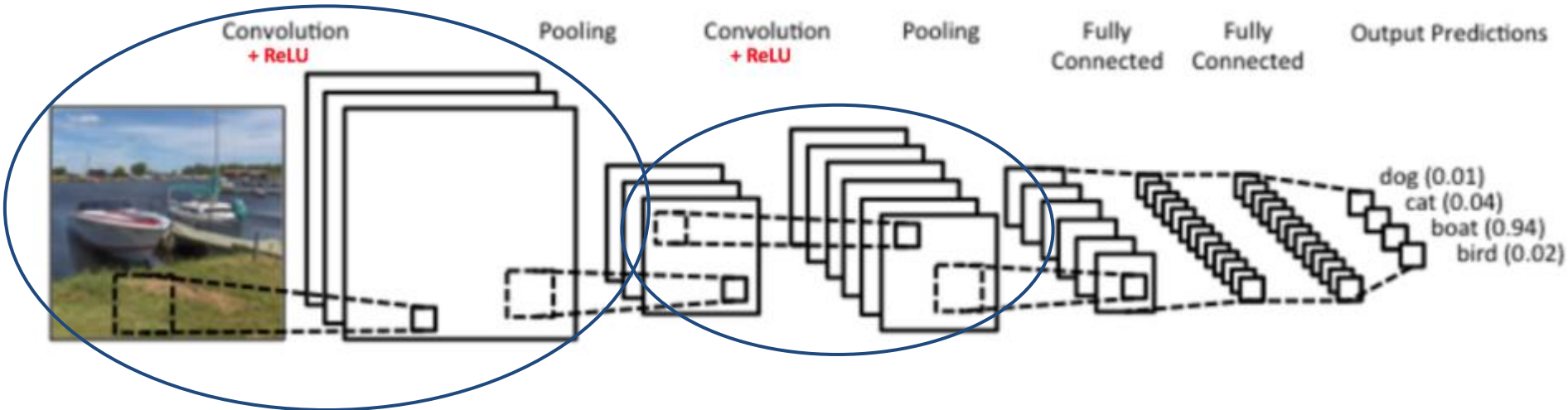
Et la couleur ?

Nous avons vu des filtres sur des images N&B. En pratique, vos images seront souvent en couleur.
Vos filtres seront donc en 3D.

DANS UN RÉSEAU

La convolution est donc un des 4 types d'opérations se produisant dans un réseau convolutionnel. Cela est fait de la manière suivante :

- Le réseau va en fait apprendre la valeur du filtre. Donc vous n'avez pas besoin de le choisir.
- Comme dans un perceptron multi-couches, on passe d'une couche à une autre par une opération. Cette fois l'opération est une convolution.



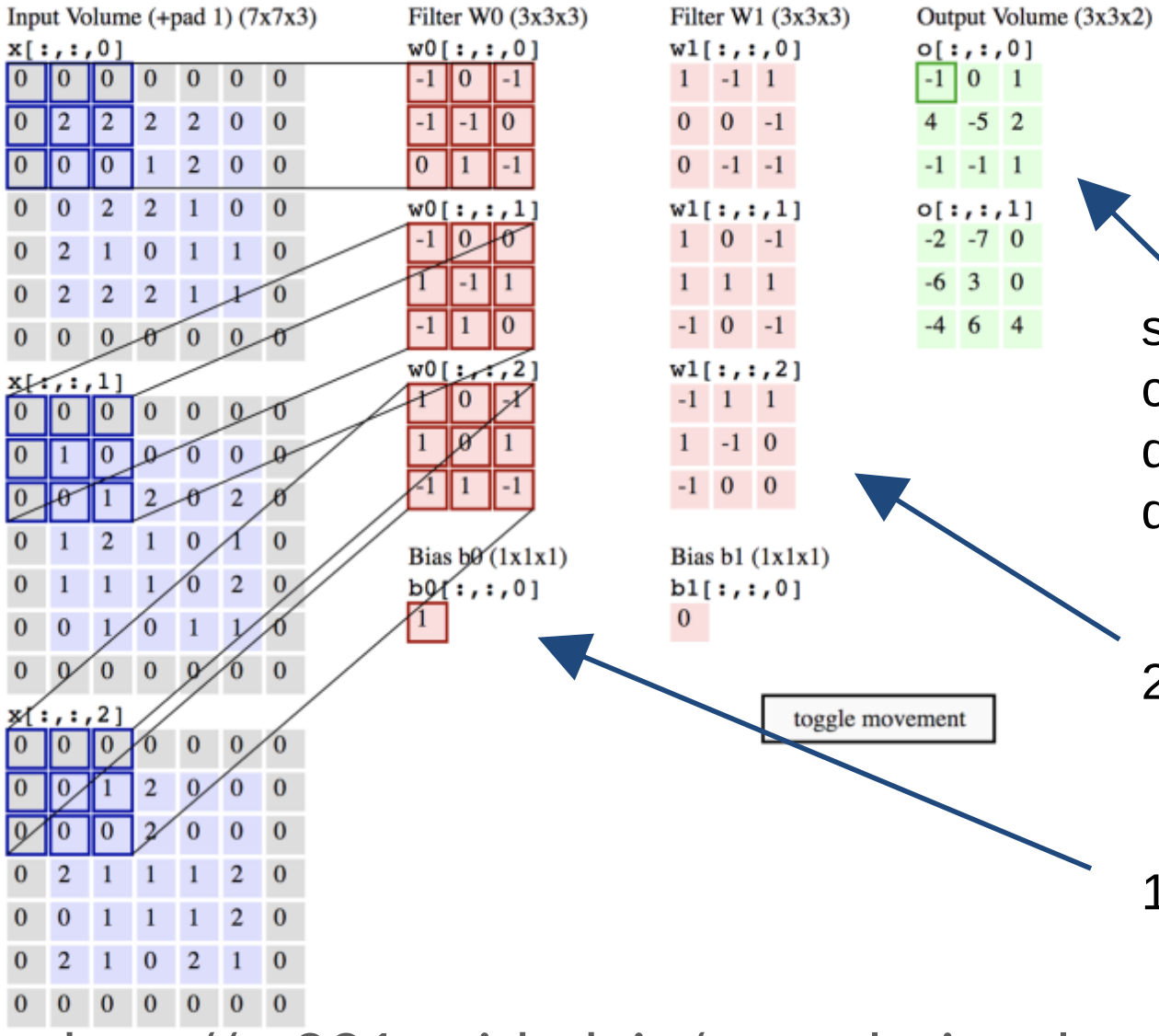
EN PRATIQUE

Comme on va le voir dans un exemple, en pratique on va appliquer plusieurs filtres sur les neurones de la couche précédente. Et potentiellement 1 par “channel” de couleur. Le nombre de filtres s’appelle la profondeur.

Un filtre est appliqué par exemple, tous les 2 neurones. Ce paramètre s’appelle le stride. De lui dépend la taille de la couche d’après.

Enfin, en pratique on ajoute souvent une rangée de 0 partout autour de la couche précédente. Cela s’appelle le zero-padding.

EXEMPLE



0-padding

couches R, G, B (par exemple)

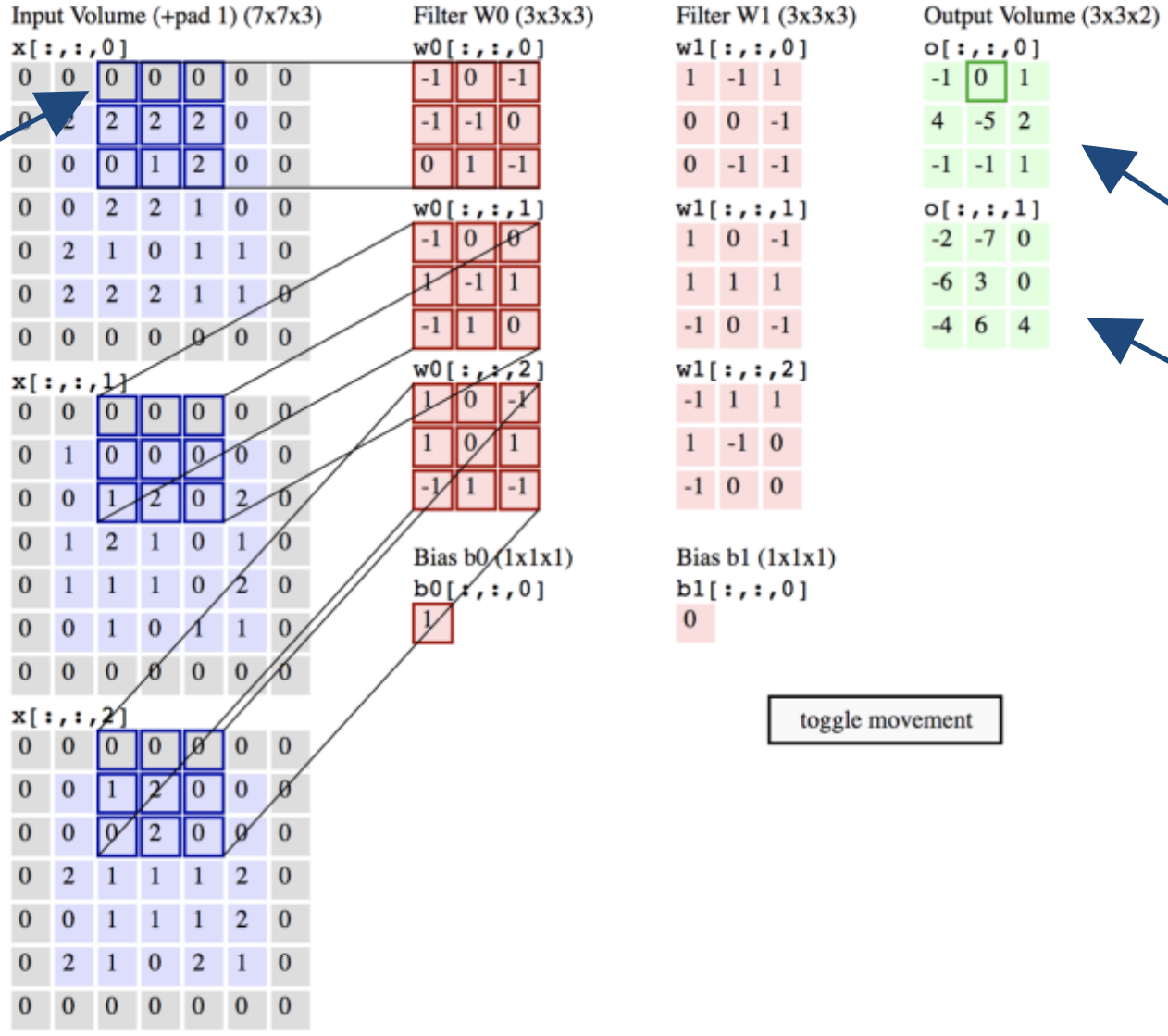
stride = 2 => chaque couche de sortie ne fait que 3 x 3.

2è filtre 3D

1er filtre 3D

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

EXEMPLE

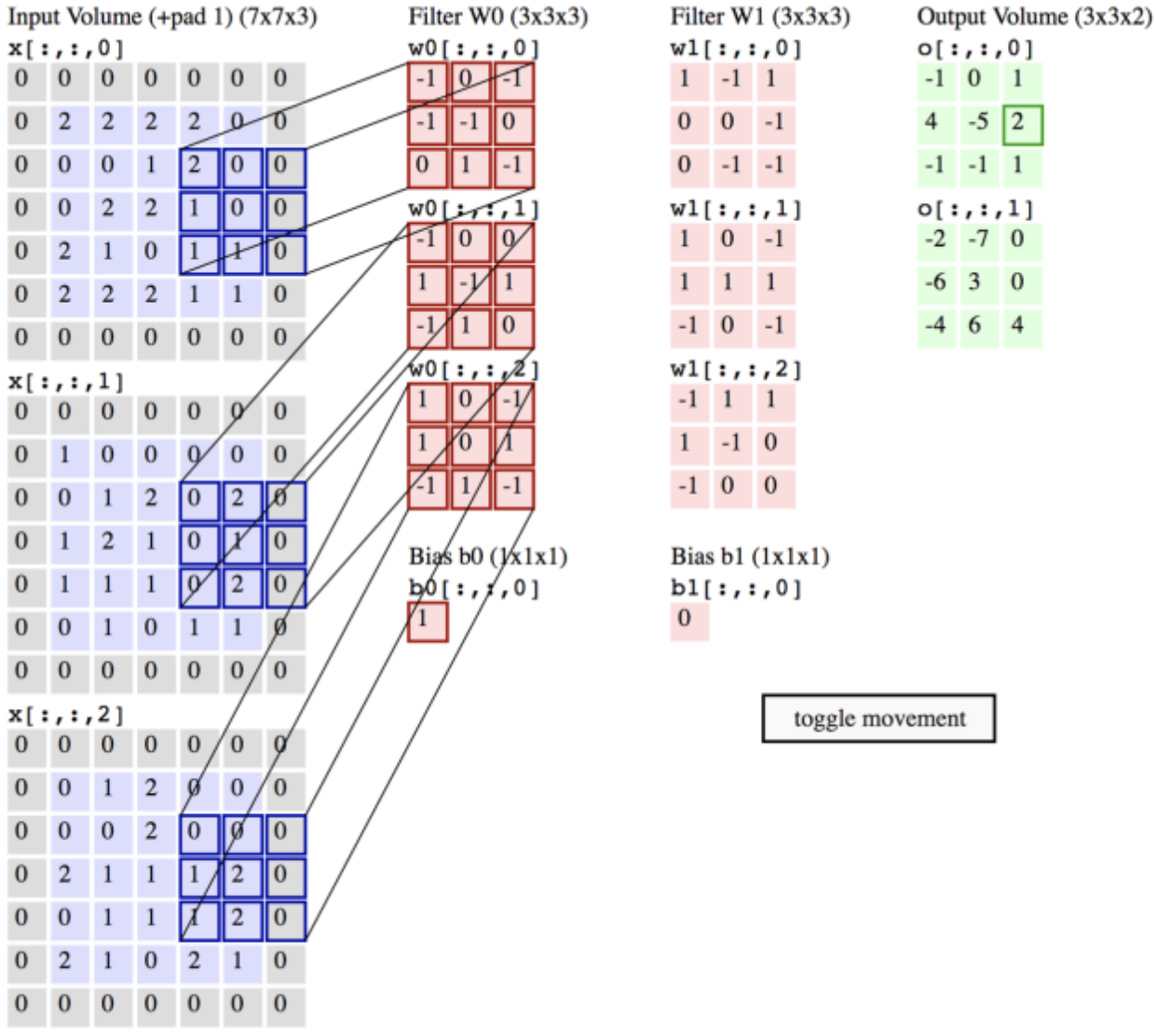


on passe directement à la 3e colonne (stride=2)

en sortie: 2 feature maps

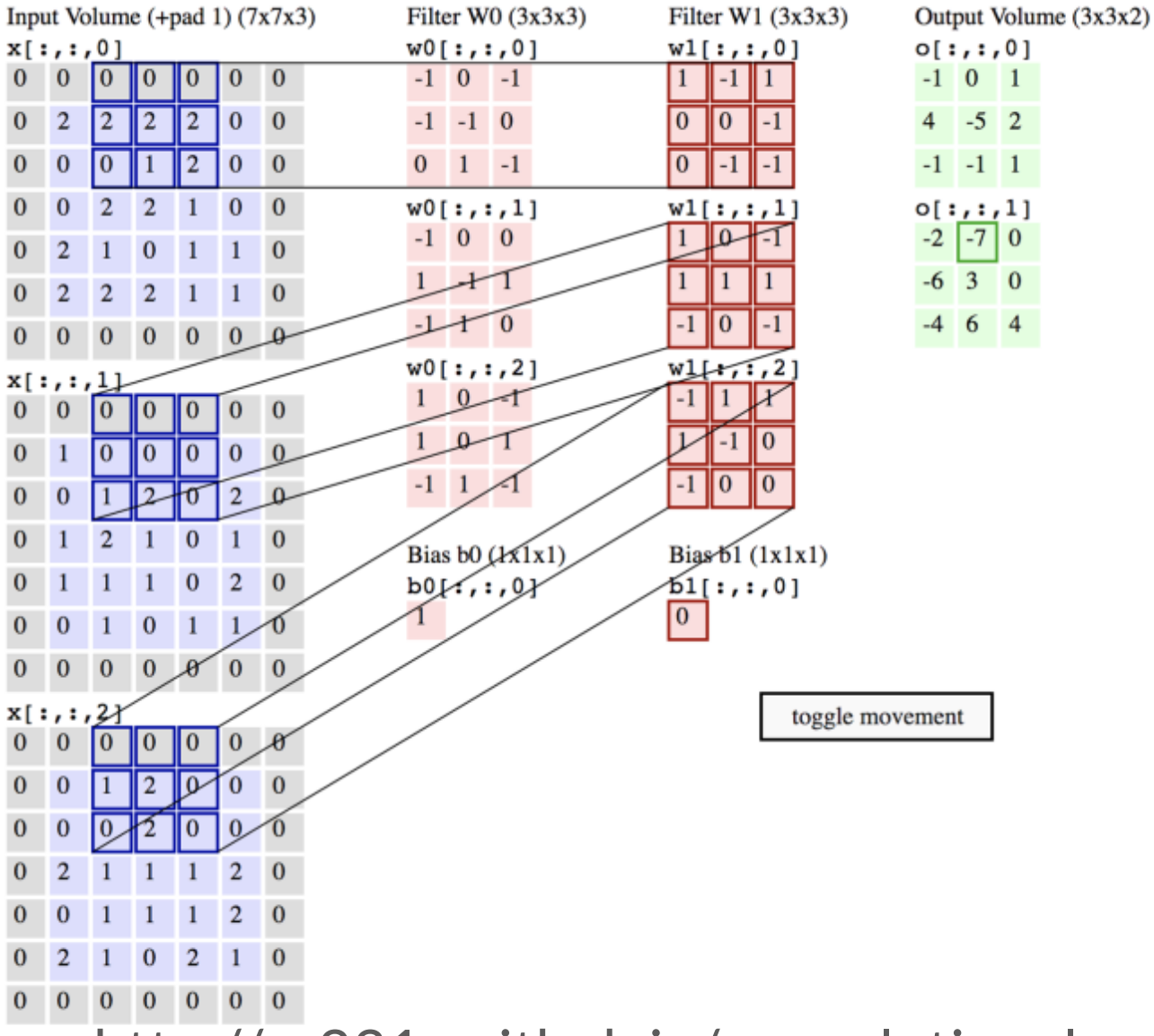
Source: http://cs231n.github.io/convolutional-networks/

EXEMPLE



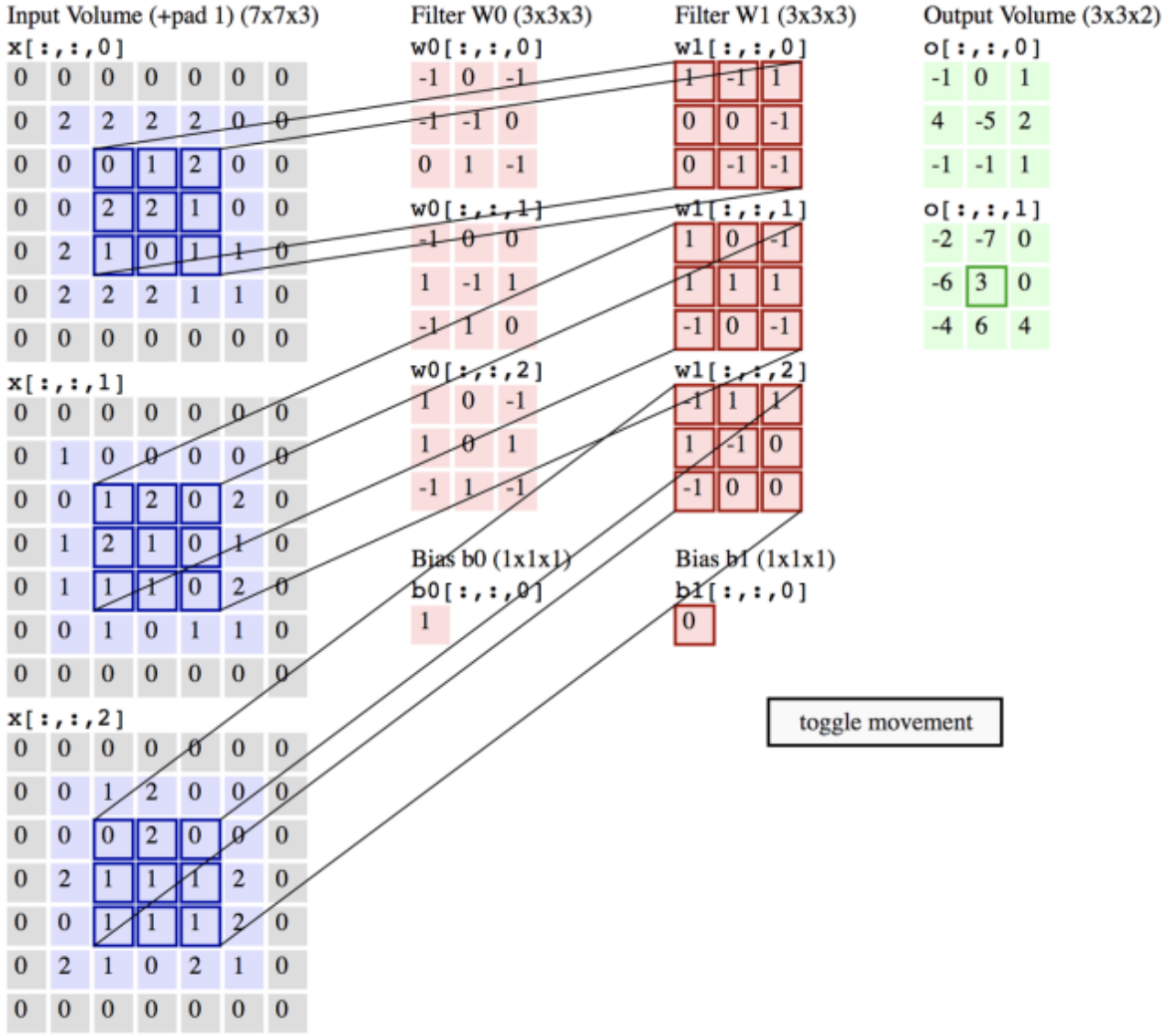
Source: http://cs231n.github.io/convolutional-networks/

EXEMPLE



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

EXEMPLE



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

QU'EST-CE QUE CA CHANGE ?

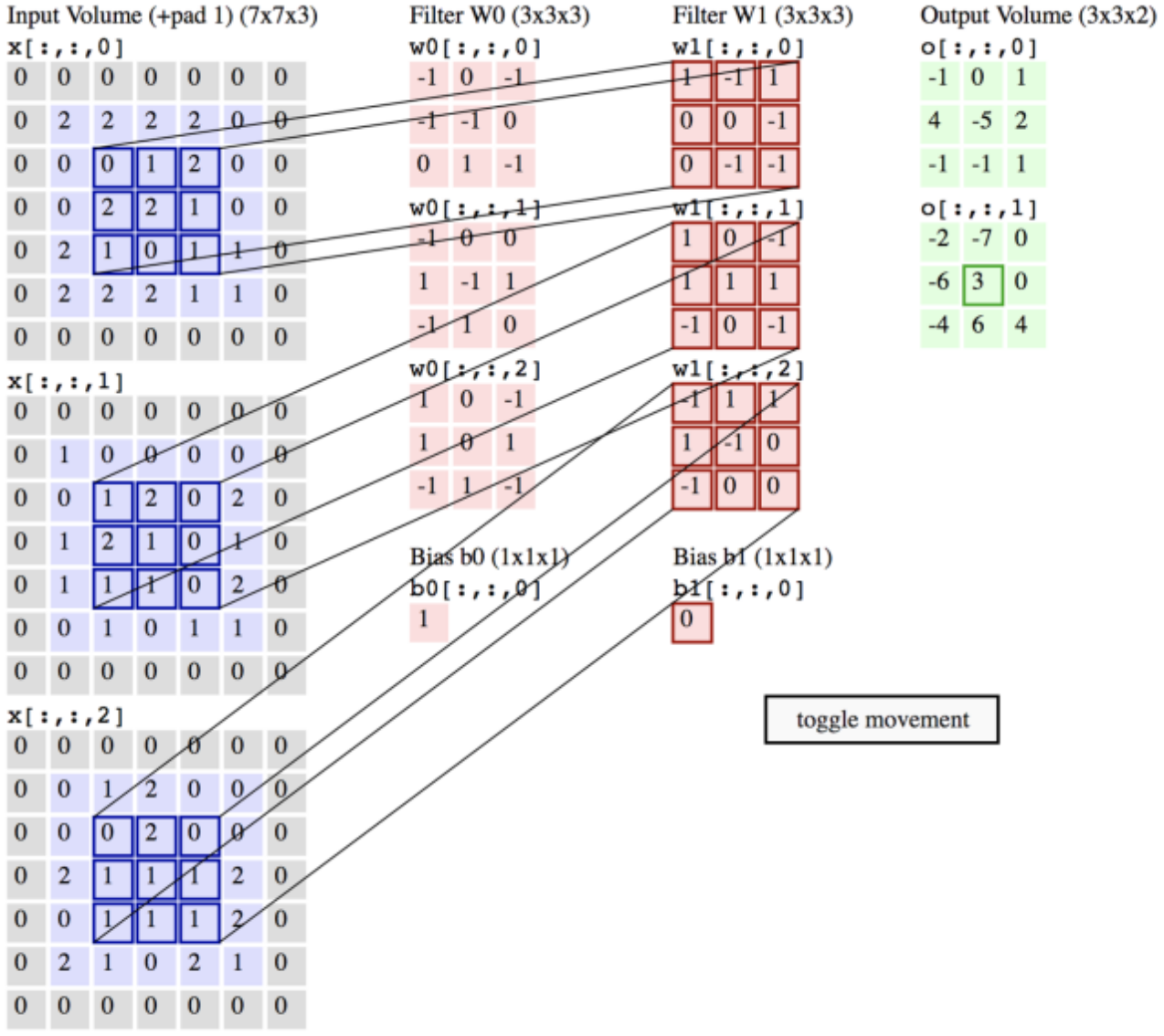
Selon vous, quels sont les 2 plus gros avantages des couches convolutives par rapport aux couches "totalement connectées" du perceptron multi-couches ?

QU'EST-CE QUE CA CHANGE ?

Selon vous, quels sont les 2 plus gros avantages des couches convolutives par rapport aux couches "totalement connectées" du perceptron multi-couches ?

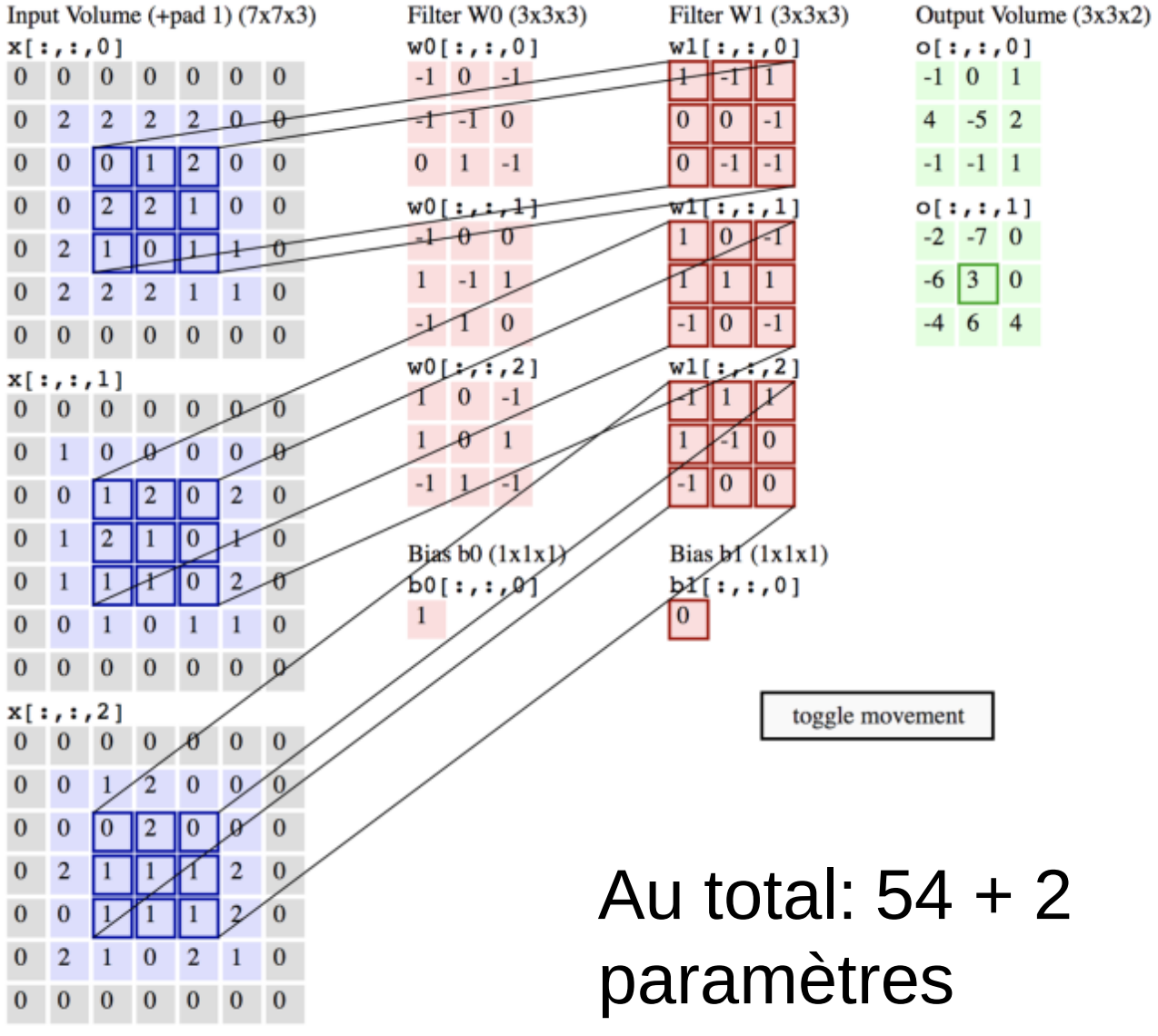
- Beaucoup moins de paramètres à apprendre. Avant: 1 poids par pixel. Maintenant: 9 poids par filtre.
- Les poids d'un filtre sont les mêmes sur toute l'image, ces features sont donc invariants par translation. Avant: la qualité de la détection dépendait de la position de l'objet dans l'image.

COMBIEN DE PARAMÈTRES ?



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

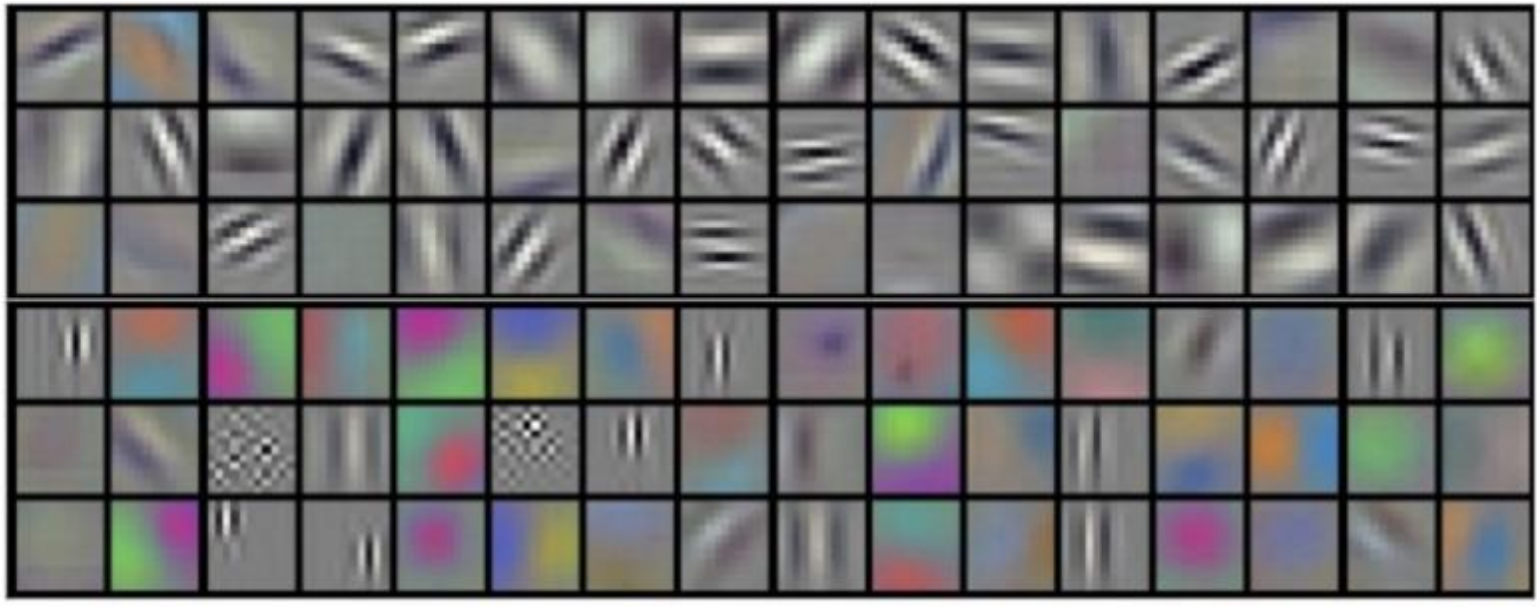
COMBIEN DE PARAMÈTRES ?



Au total: 54 + 2 paramètres

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

REPRÉSENTATION DES FILTRES

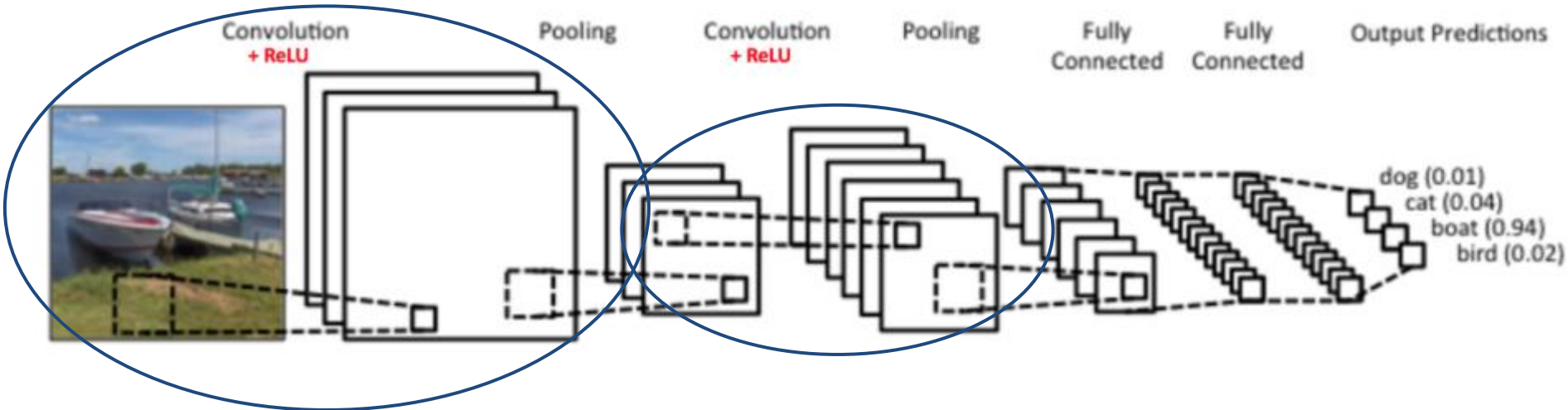


Source: [Krizhevsky et al, 2012]

Les filtres appris pendant la phase de convolution représentent des features. Ils sont très dépendants des images en entrée. Ils représentent ce qui est “intéressant” dans ces images. Vous n’avez pas la main dessus, c’est les calculs du réseau qui vous trouvent ces features.

CONVOLUTION : RÉSUMÉ

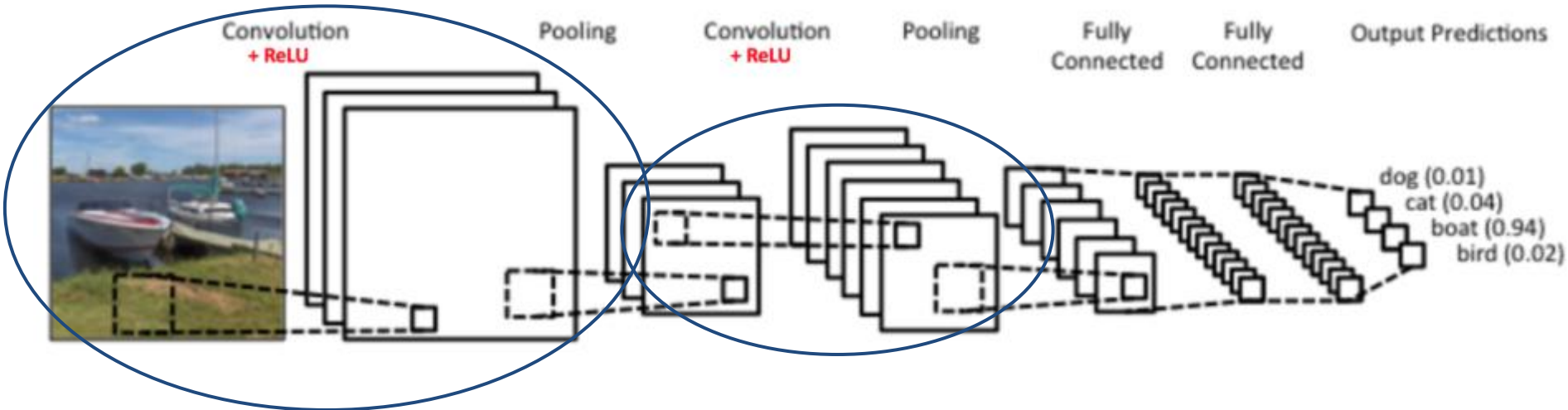
- chaque étape de convolution renvoie un volume formé d'un certain nombre de feature maps.
- la taille de ce volume dépend de:
  - nombre de filtres (hauteur du volume)
  - stride (longueur, largeur du volume = celles de l'input / stride)
  - 0-padding: peut également modifier la longueur et largeur du volume.
- le réseau va apprendre les filtres, il ne faut pas les choisir



Fonction ReLU

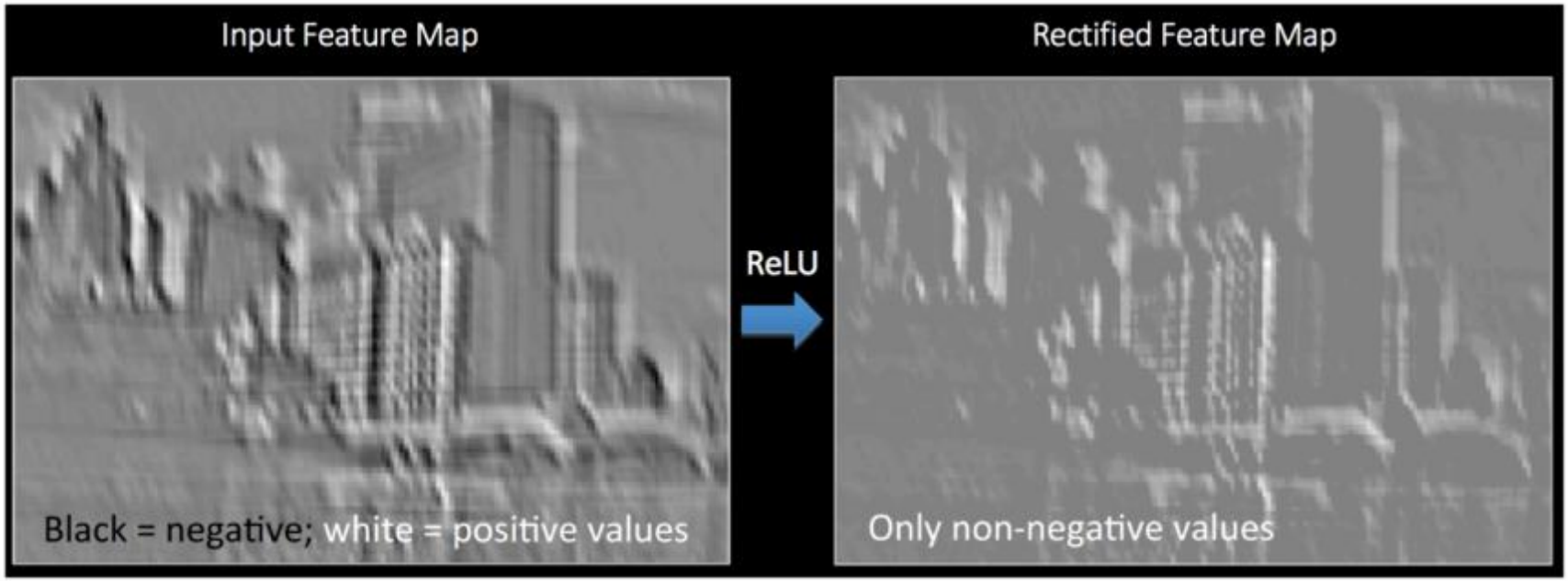
Cette opération se fait juste après la convolution, si bien qu'elle se confond un peu avec.

ReLU: Rectified Linear Unit. Opération permettant d'ajouter de la non-linéarité au réseau (= max(0, x)).



COUCHE ReLU

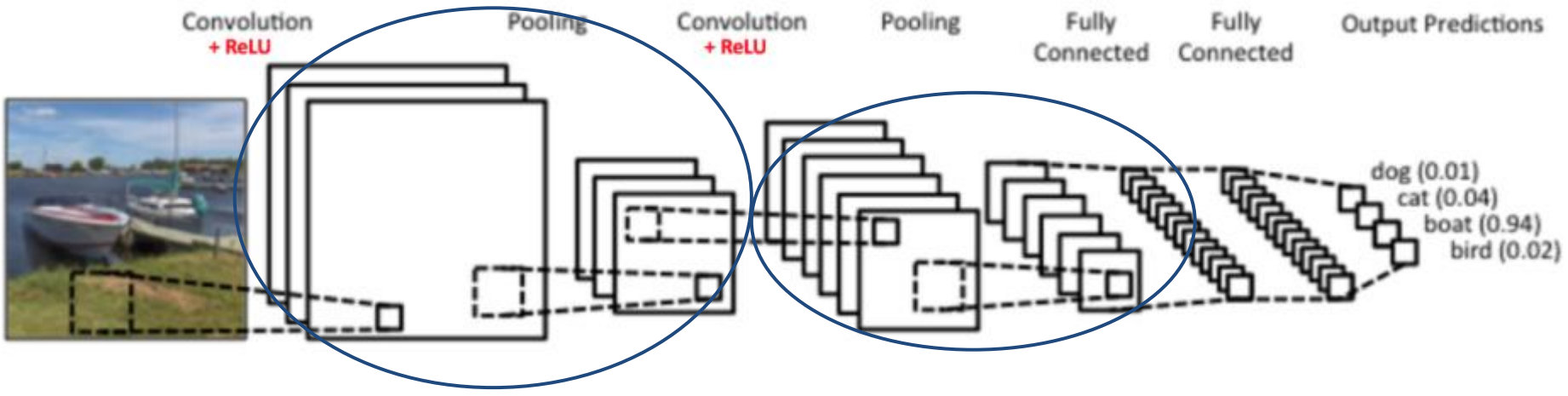
De manière extrêmement simple, on va appliquer à chaque neurone de sortie la fonction f(x) = max(x, 0), c'est-à-dire remplacer toutes les valeurs négatives par 0. C'est tout!



Source:
http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf

COUCHE DE POOLING

L'opération de pooling ou agrégation arrive en pratique après convolution + ReLU. Un volume qui passe par cette couche va sortir en plus petit! L'objectif de cette couche est de résumer et régulariser l'information contenu dans l'input.

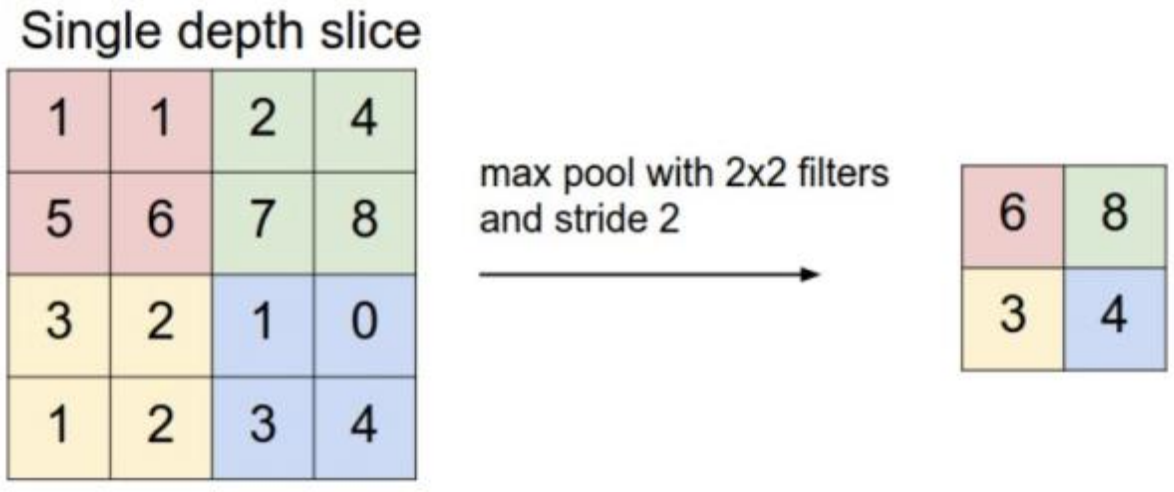


COUCHE DE POOLING

En pratique on choisit :

- un stride
- une taille de fenêtre

Et on fait glisser la fenêtre sur l'input en choisissant la plus grande valeur (max-pooling). Ou éventuellement une autre fonction comme la somme.



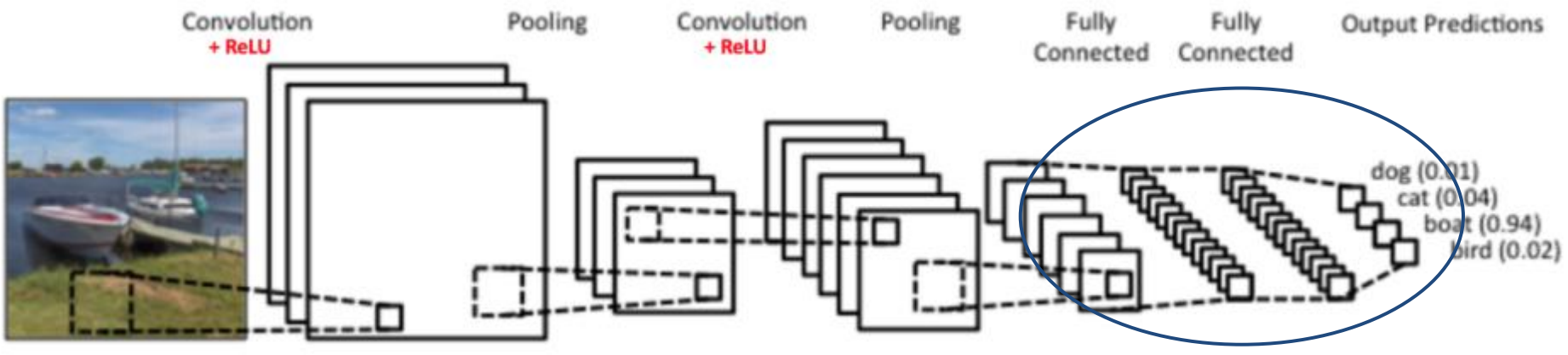
INTÉRÊTS DU POOLING

Le pooling, car il résume l'information de manière locale et qu'il fait diminuer le nombre de dimensions (réduction de dimension) permet en pratique de:

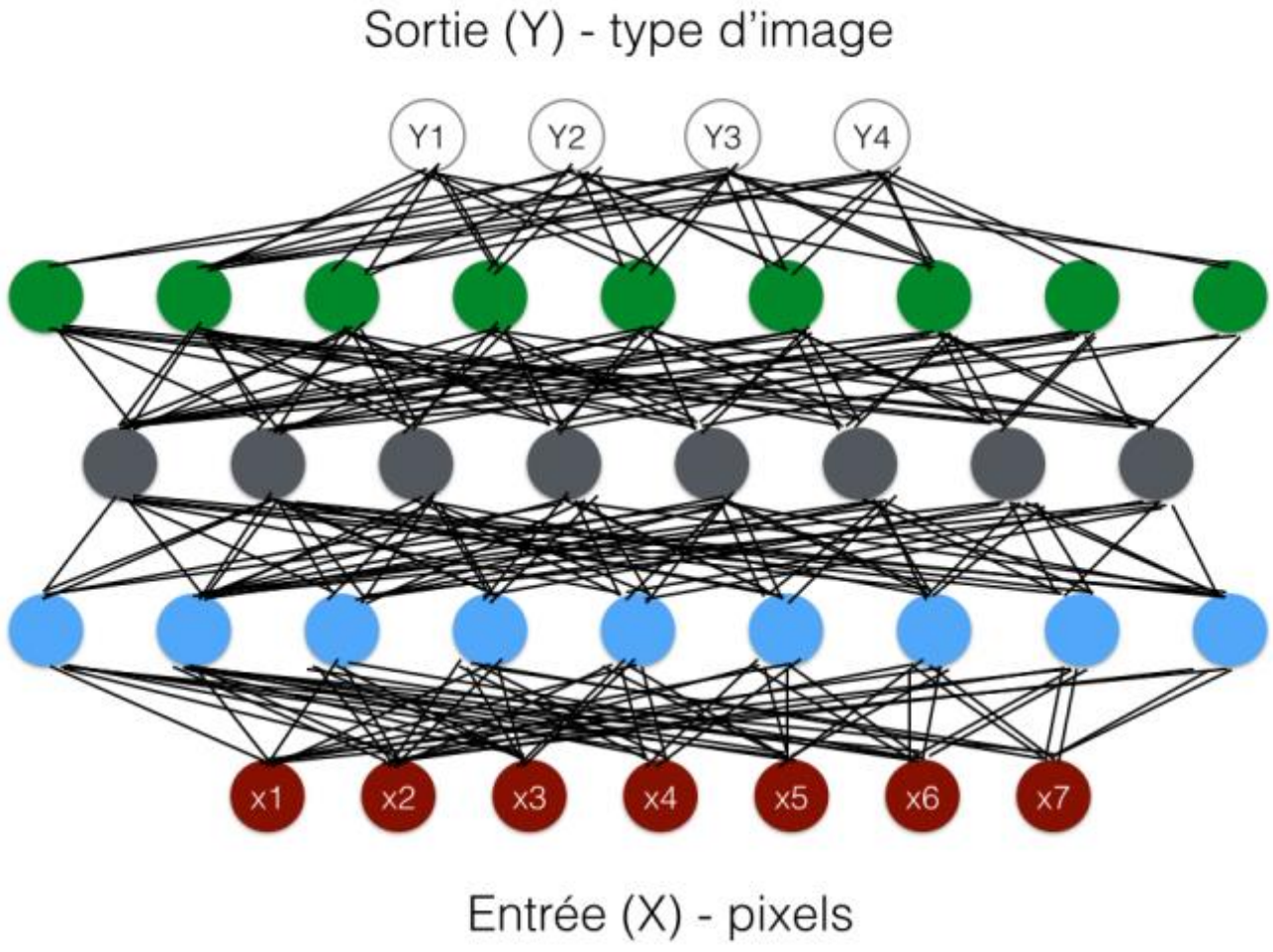
- limiter le sur-apprentissage : moins de paramètres
- dimension plus faible et donc plus facile d'interpréter les résultat de ces couches
- rend le réseau invariant au légères distorsions, car le résultat du pooling serait le même sur une entrée légèrement différente

COUCHE TOTALEMENT CONNECTÉE

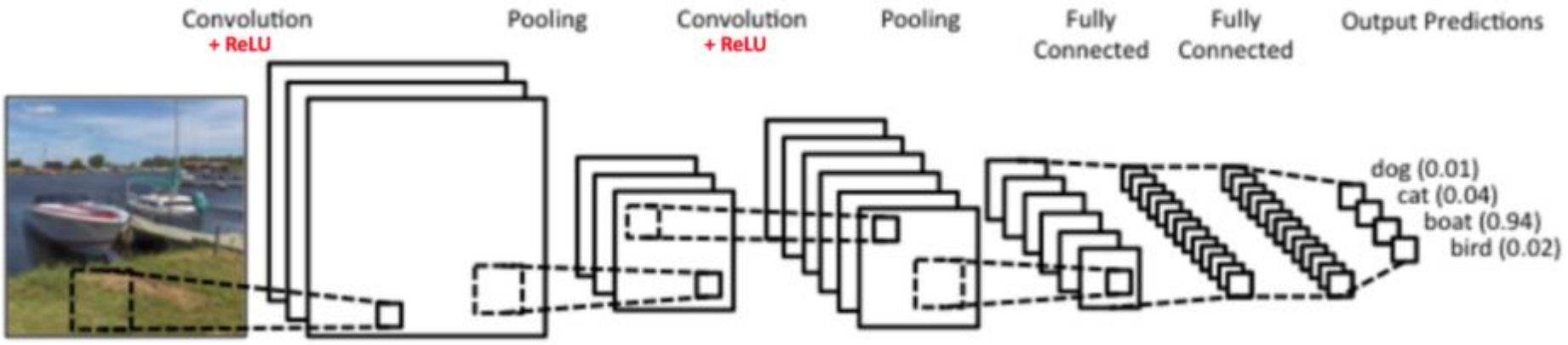
Ces couches sont les mêmes que dans un perceptron multi-couches, comme vu dans le chapitre des PMC.



COUCHE TOTALEMENT CONNECTÉE



RÉSUMÉ



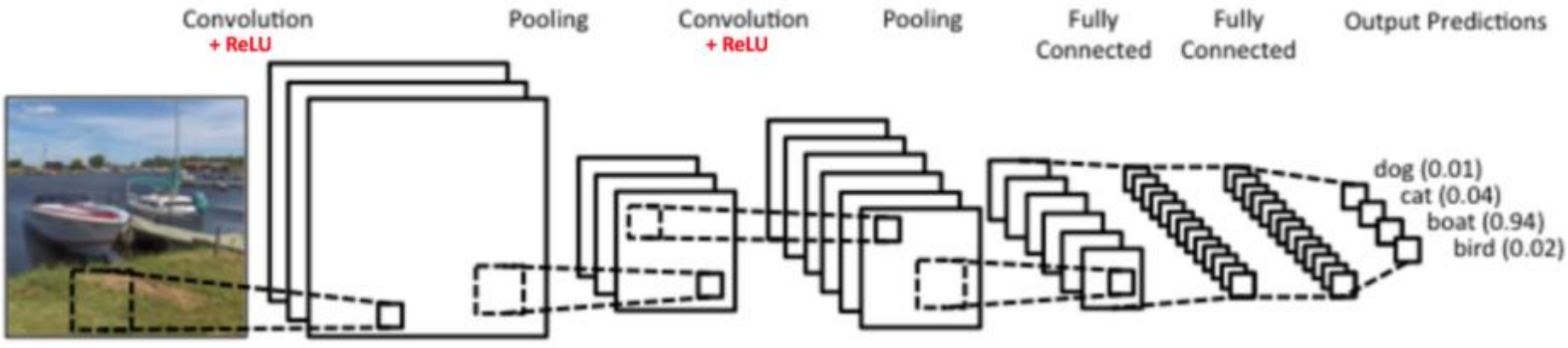
Input brut

Plusieurs filtres pour créer plusieurs feature maps

Résumé/régularisation

Algo classique de classification

RÉSUMÉ



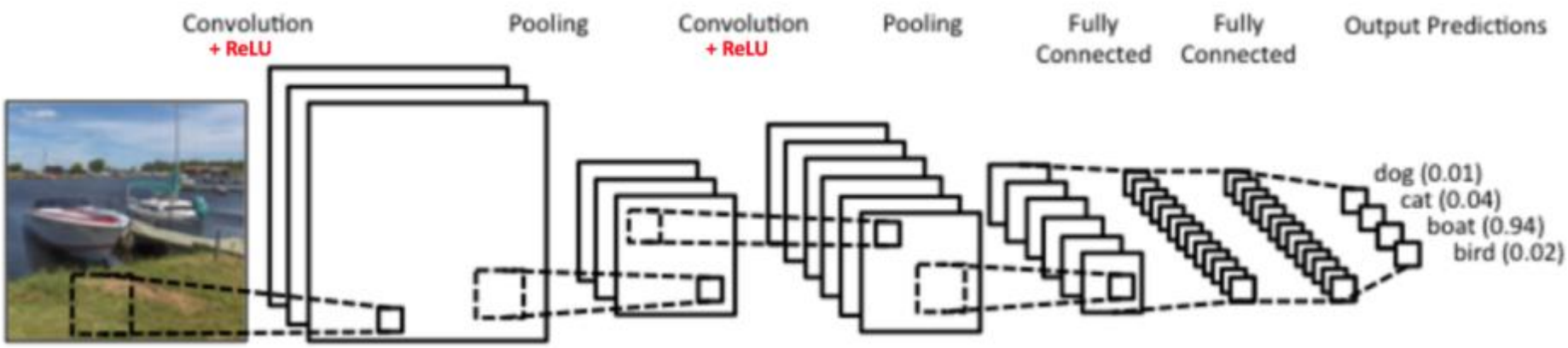
- nombre de filtres
- 0-padding
- stride

Choisir la fonction max ou sum.

Choisir les fonctions d'activation. En pratique, sigmoide.

Et c'est tout ?

RÉSUMÉ



- nombre de filtres
- 0-padding
- stride

Choisir la fonction max ou sum.

Choisir les fonctions d'activation. En pratique, sigmoide.

Il faut maintenant choisir le nombre de blocs, l'ordre dans lequel les mettre, et s'intéresser aux paramètres de l'optimisation!

RÉTRO-PROPAGATION

Souvent on a plusieurs classes (chient, chat, avion, camion…). On utilise alors la fonction de perte dite softmax qui est la généralisation de la fonction sigmoïde et renvoie pour chaque exemple un vecteur de valeurs sommant à 1.

La fonction associée qu'on veut minimiser (la log-vraisemblance) s'appelle dans ce cas la cross-entropy. Elle ressemble à:

$$\sum_{i=1}^{n} \sum_{c=1}^{C} y_{i,c} \ln(\hat{y}_{i,c})$$

1 si i est dans la classe c, 0 sinon

la proba prédite d'être dans la classe c

RÉTRO-PROPAGATION

Pour trouver la valeur des paramètres, rétro-propagation comme pour le MLP, c’est-à-dire via une descente de gradient sur une fonction heureusement dérivable, mais pas convexe!

RAPPEL : LEARNING RATE

Idée de la descente de gradient: faire des petits pas dans la direction de la pente jusqu'à atteindre le minimum.

La taille des pas est ce qu'on appelle le taux d'apprentissage ou encore learning rate. Il nous dit à quel point ce que nous venons de voir doit influencer la modification des poids.

Il est généralement noté η.

TROP DE PARAMETRES ?

Un “petit” réseau convolutif peut déjà atteindre le million de paramètres. Problème quand on a autant de paramètres : le sur-apprentissage. En particulier lorsqu’on a beaucoup moins d’exemples que de paramètres, on risque d’être très bon sur l’ensemble d’entraînement et très mauvais sur l’ensemble de test. Comment diminuer ce nombre ?

1. drop-out: on fixe un pourcentage p, par exemple p = 50%. Avec une probabilité p, chaque neurone est ignoré à chaque itération (comme s’il valait 0). De manière aléatoire, on diminue donc le nombre de paramètres à modifier.
2. weight decay: un autre paramètre qui va pousser certains poids vers 0. Plus il est grand, plus un grand nombre de poids va décroître fortement.

APPRENTISSAGE PAR BATCH

Les couches convolutives vont vite poser un problème de mémoire. Car plus leur profondeur est grande, plus on démultiplie le nombre d'images intermédiaires, qui doivent être gardées en mémoire le temps de la rétro-propagation.

Du coup, souvent on ne peut pas faire passer toutes les images dans le réseau en même temps et on est obligé de couper en batchs. Un batch :

- est un ensemble d'images d'entraînement
- est envoyé en haut du réseau et rétro-propagé
- est supprimé de la mémoire vive dès que les poids sont modifiés

APPRENTISSAGE PAR BATCH

En pratique, couper le problème en batch pose deux problèmes:

- on perd de l'information par rapport à l'ensemble du dataset, donc le gradient peut aller dans le mauvais sens, pour un batch donné
- pour la même raison, le modèle est donc plus long à entraîner

EN PRATIQUE

Il faut avoir en tête que la recherche avance moins vite que l'expérience sur le deep learning. En particulier, on ne sait pas encore pourquoi ces réseaux fonctionnent bien sur certains problèmes.
Et on ne sait pas encore non plus comment bien choisir les paramètres pour que ça marche!
Solution: essai / erreur / copie.

CE QUE VOUS DEVEZ RETENIR

- Le Machine Learning est un ensemble de méthodes, chacune adaptée (ou pas) à certaines situations.
- Si vous devenez Data Scientist: votre job est de comprendre quelle méthode utiliser, de pouvoir l'expliquer et de savoir optimiser ses paramètres.
- La complexité de la solution doit dépendre de celle du problème.
- Il n'y a qu'un moyen pour savoir si vous avez résolu le problème : le test!
- La plupart des méthodes sont déjà implémentées, ne réinventez pas la roue.
- Votre intuition et votre expérience sont pour beaucoup dans la qualité du modèle. L'ordinateur est un outil.