Convolutional Neural Networks (1)

Pourquoi ?

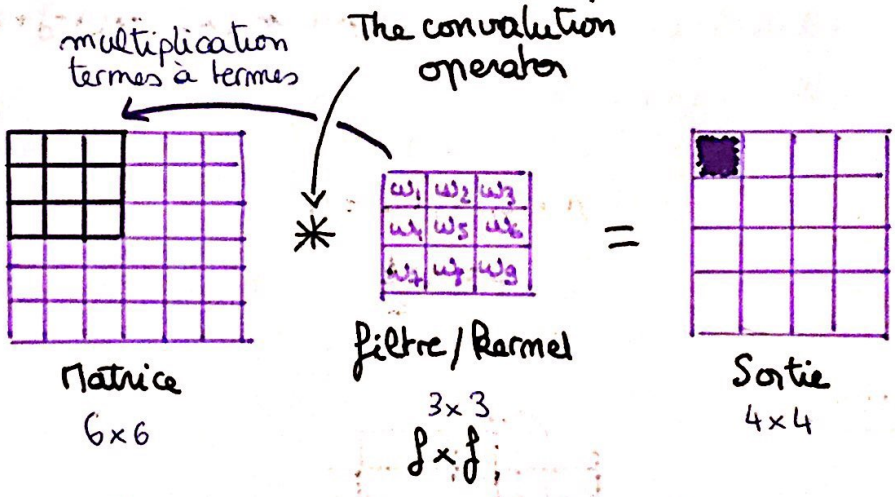
Si on utilisait des réseaux de neurones FC sur les images, on devrait traiter une matrice trop large (~n° de paramètres)

=> overfitting car on aurait du mal à avoir assez de données pour tous ces paramètres.

=> calcul/mémoire trop couteux, infaisable et on ne veut pas se limiter qu'aux petites images.

Solution = opération de convolution.

L'opération de convolution



=> il existe beaucoup de filtres différents qui permettent de faire de la détection de contours (ex: Vertical edge, Horizontal, Sobel, Schaar filter...).

Python : conv-forward
Tensorflow : tf.nn.conv2d
keras : Conv2D.

⚠ convolution = regular Kernel + flipping Kernel (V + H)

$(A * B) * C = A * (B * C)$ Associativité

cross-correlation = regular Kernel

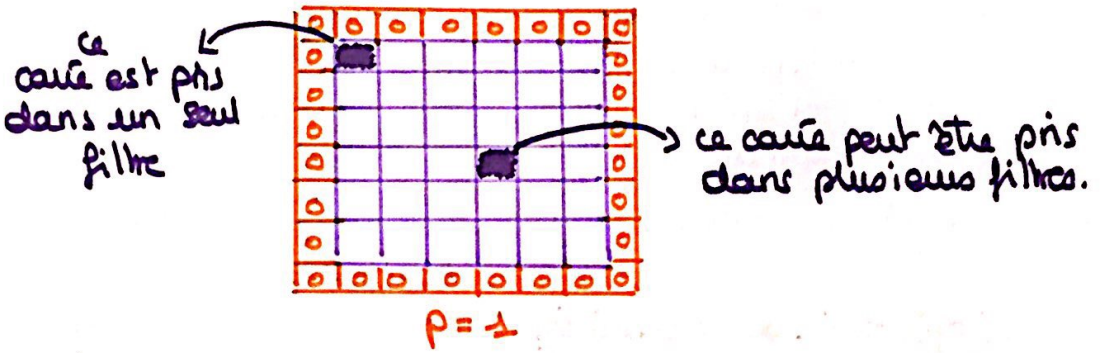
En pratique, on ne se préoccupe pas de l'étape "flipping".

Padding

=> des pixels du bord sont beaucoup moins utilisés dans la sortie (perte d'information)

=> On ne veut pas que l'image rétrécisse à chaque fois qu'on détecte des bords (↔ Shrinking image)

Solution : Rajouter un contour de zéros.



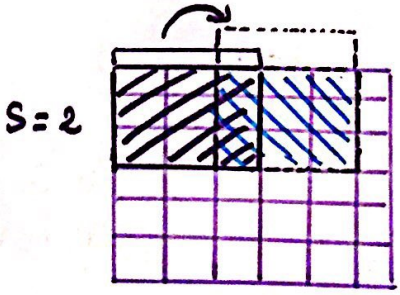
- Valid convolution : $n \times n * f \times f \rightarrow (n-f+1) \times (n-f+1)$ $p=0$
- Same convolution : la taille de la sortie est la même que la taille de l'entrée.

$$\Rightarrow n + 2p - f + 1 = n$$

$$p = \frac{f-1}{2}$$

f est en général impair.

Stride



= nombre de pas dont on déplace le filtre sur l'image.

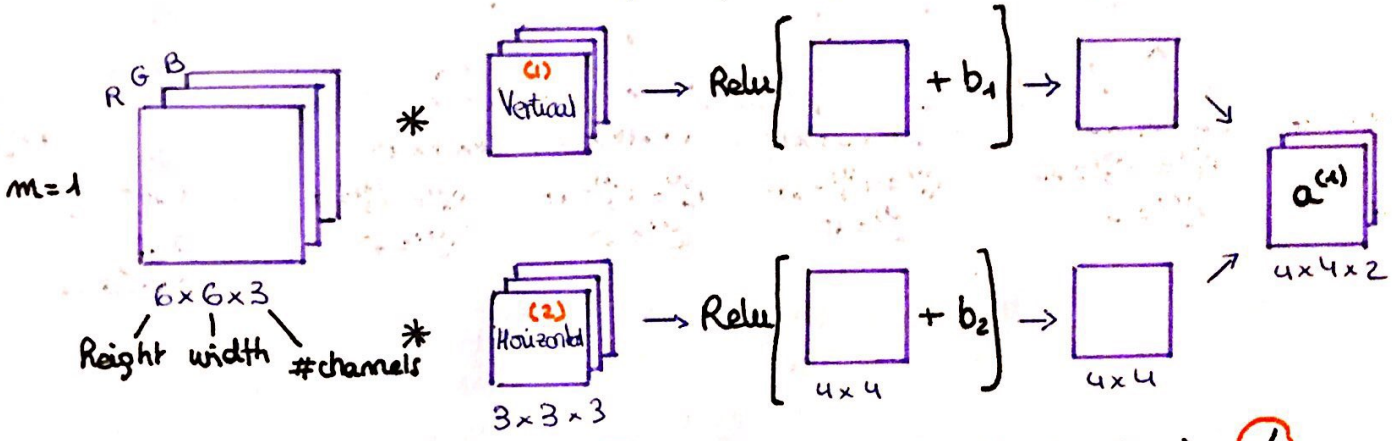
d'image finale (la matrice finale) a pour taille :

$$\left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor$$

Exemple de réseaux avec image 3D

$s=1$, $p=0$, $f=3$, nombre de filtres (n_c') = 2

=> on peut ajouter plusieurs détecteurs de features.



$n \times n \times n_c$ — $a^{(0)}$

$f \times f \times n_c$ — "$w^{(1)}$"

$(n-f+1) \times (n-f+1) \times n_c'$

$z^{(1)} = w^{(1)} a^{(0)} + b^{(1)}$

$a^{(1)} = g(z^{(1)})$

Pour une couche l, on a

Entrée

$n_H^{[l-1]} \times n_w^{[l-1]} \times n_c^{[l-1]}$

hauteur — largeur — nombre de channels.

Sortie

$n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

nombre de filtres de la sortie.

Filtre

$f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

taille de l'entrée

Activations

$A^{[l]} = m \times n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

nombre d'exemples

Poids

nombre de paramètres : $\left[f^{(l)} \times f^{(l)} \times n_c^{(l-1)} \quad \underset{\text{biais}}{+1} \right] \times n_c^{(l)}$

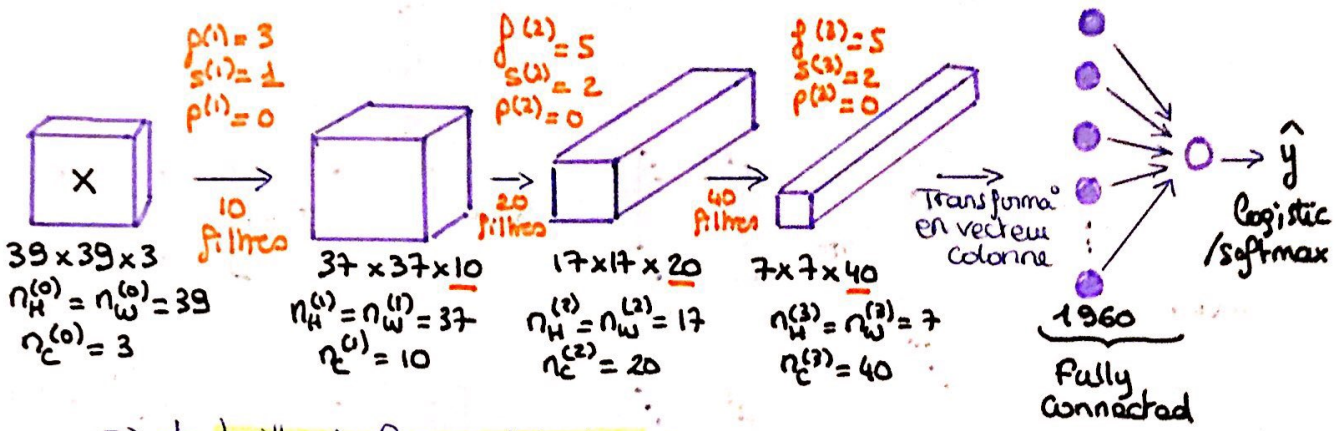
Biais

$n_c^{(l)} = [1, 1, 1, n_c^{(l)}]$

$$n_H^{(l)} = \left\lfloor \frac{n_H^{(l-1)} + 2p^{(l)} - f^{(l)}}{s^{(l)}} + 1 \right\rfloor$$

=> vrai pour l'épaisseur aussi

Exemple : ConvNet (voir aussi LeNet-5 avec Conv+Pool)



=> la taille de la matrice y diminue alors que le nombre de channels augmente au fur et à mesure que l'on rajoute des convolutions.

Trois types de couches

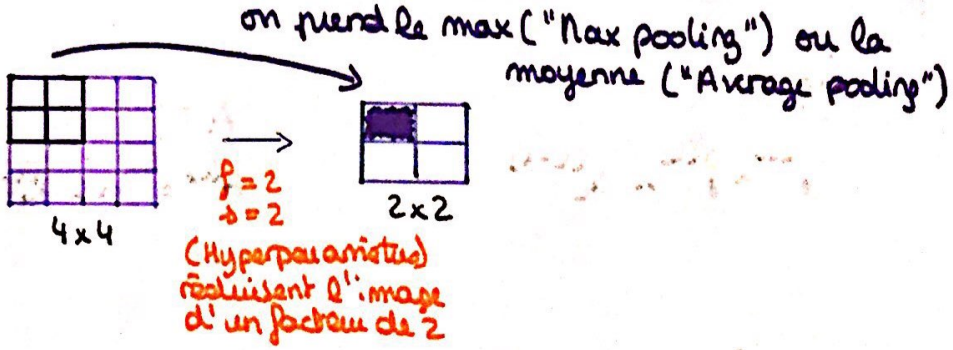
- Convolution (Conv)
- Pooling (Pool)
- fully connected (FC).

Pooling

Le pooling permet :

- ✱ de réduire la taille de la représentation
- ✱ d'accélérer les calculs.
- ✱ de faire des features qui détecte de façon + robuste.

Il s'applique indépendamment à chaque channel.



=> Il n'y a aucun paramètres à apprendre, ce sont juste des calculs.

=> permet de conserver les features de l'image.

1 couche = 1 Conv + 1 Pool (on ne compte que les couches qui ont des poids).

Choix des hyperparamètres

Il est bien de regarder ce qui existe déjà dans la littérature, les exemples qui ont bien marché.

Pourquoi les convolutions ?

Partage de paramètres : un filtre peut être utile sur plusieurs parties de l'image.
↳ overfitting.

Sparsity of connections : chaque sortie dépends seulement d'un petit nombre d'entrée.

Translation invariance : une image de chat reste une image de chat.