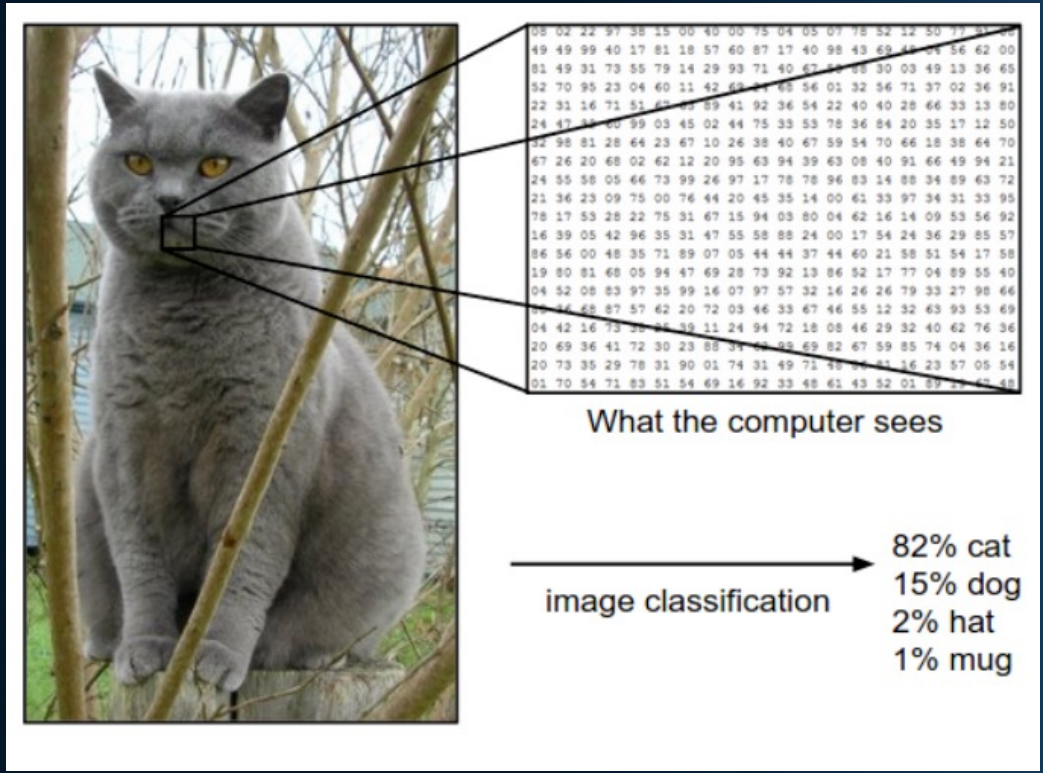
IA Géosciences - Deep learning: Réseaux de neurones convolutifs

Romain Wenger (Laboratoire Image Ville Environnement)

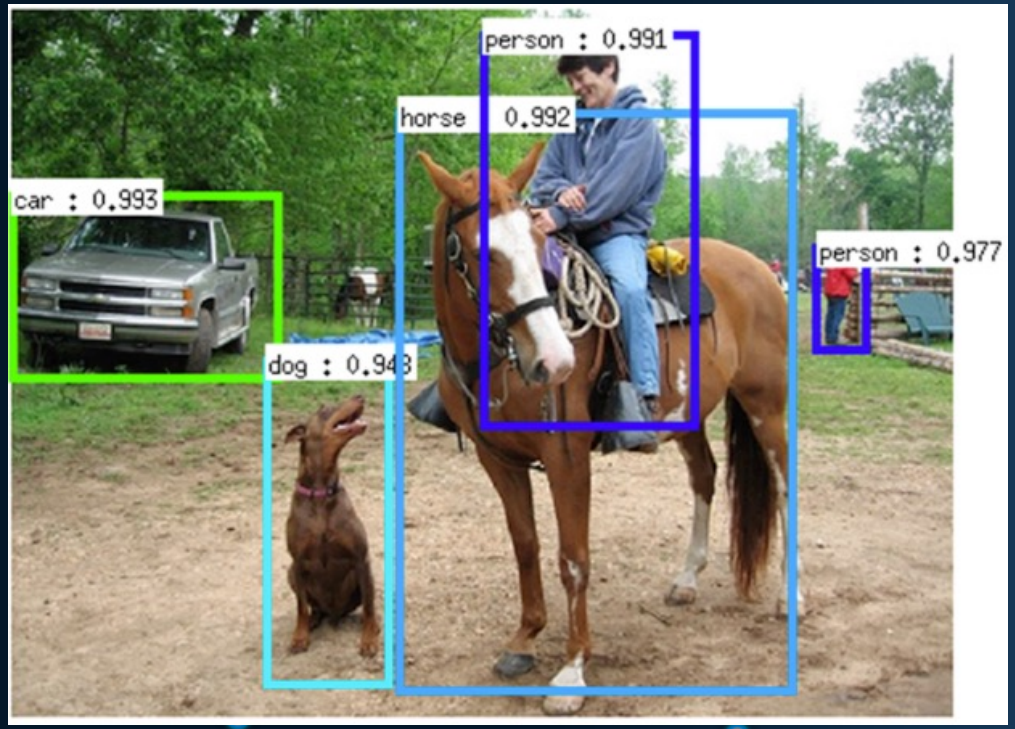
Table des matières

Table des matières

Reconnaissance d’image

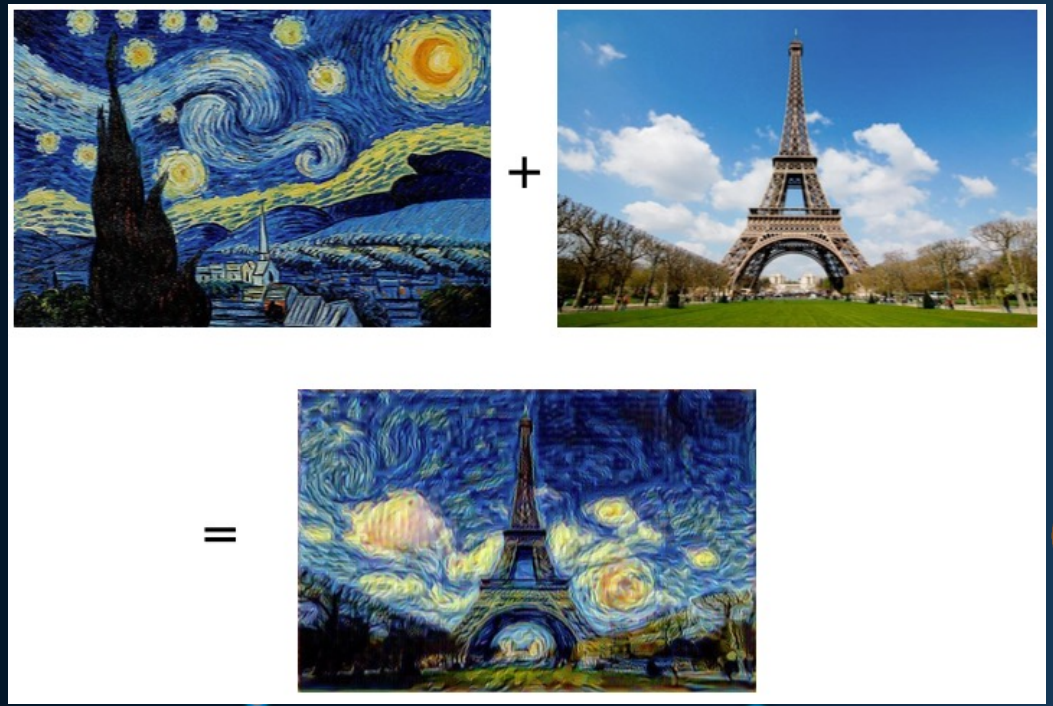


Détection d’objets

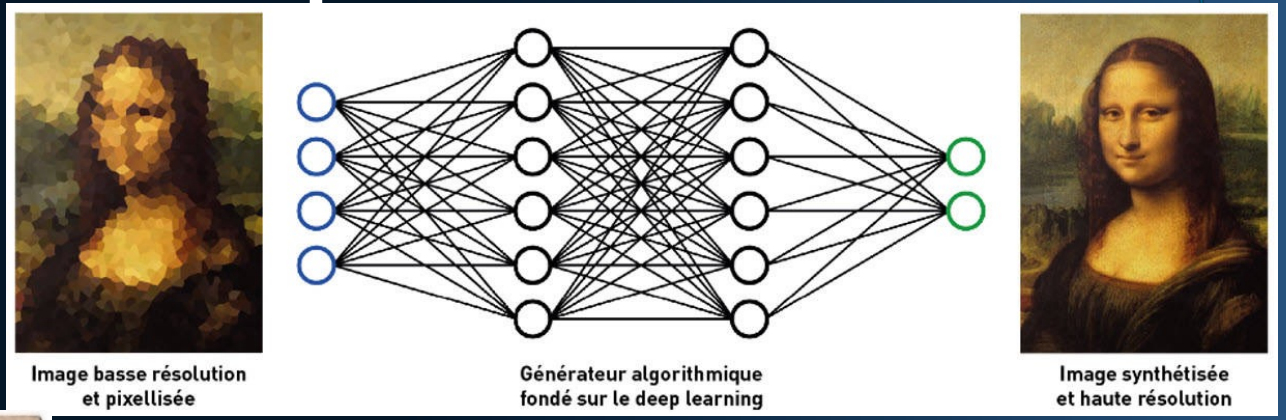


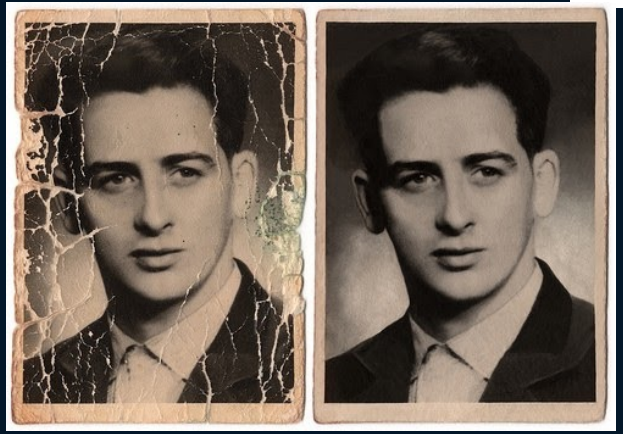
Segmentation sémantique

Transfert des styles d’images

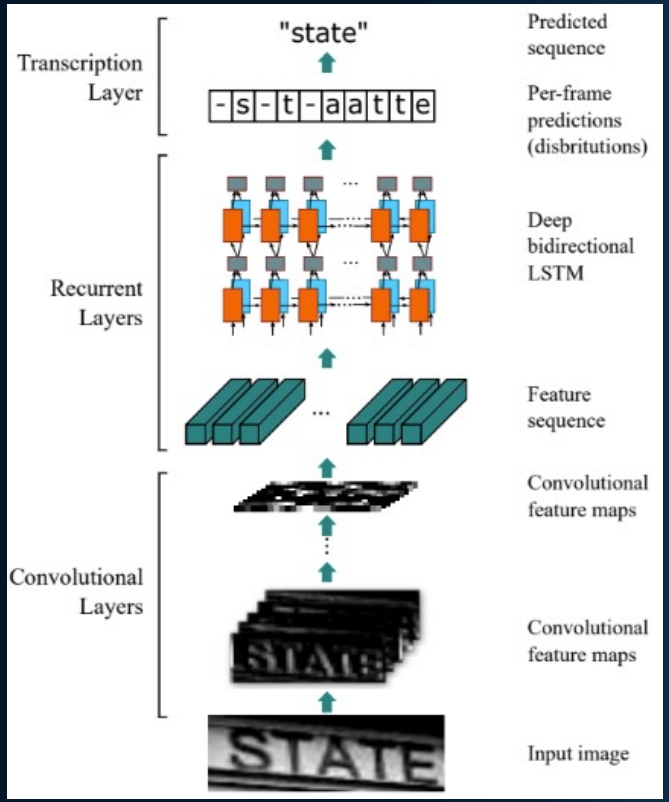


Restauration des pixels



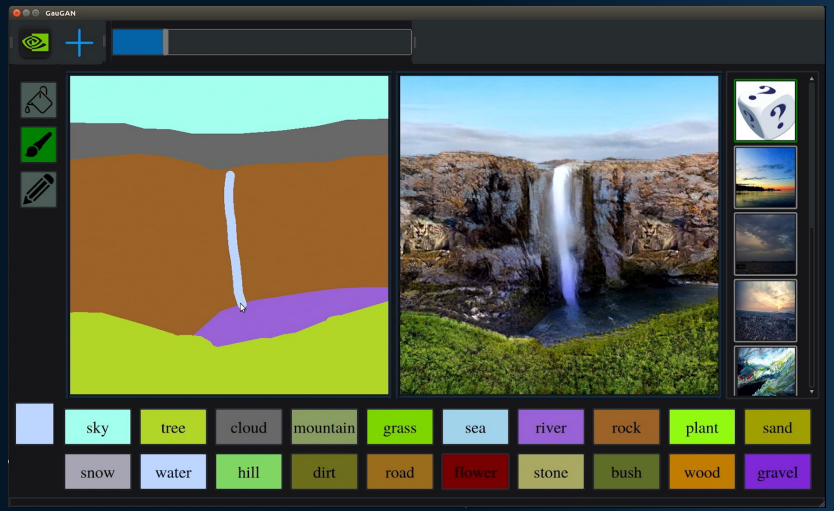


Extraction de textes dans les images



Et encore mieux ! Extraction + traduction + reconstruction

Création d’images

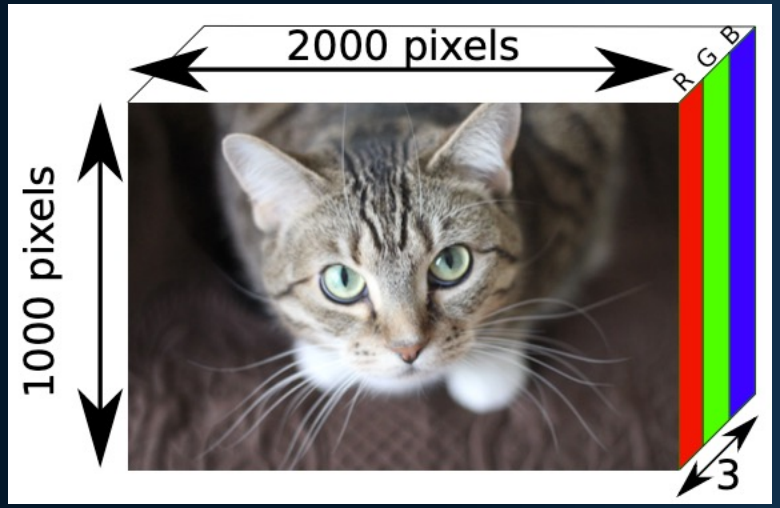


Nvidia Canvas App (fait parti de Nvidia AI Playground)

https://www.nvidia.com/en-us/studio/canvas/

https://www.nvidia.com/en-us/research/ai-playground/

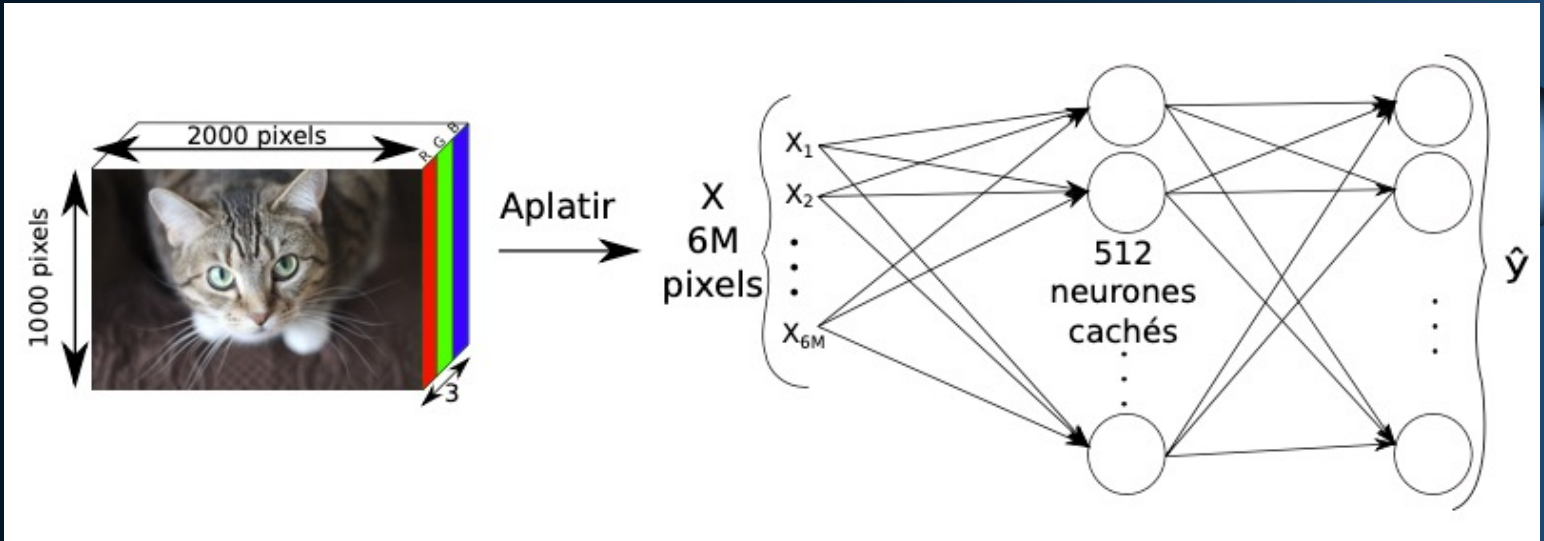
Qu’est-ce qu’une image ?



L’image c’est :

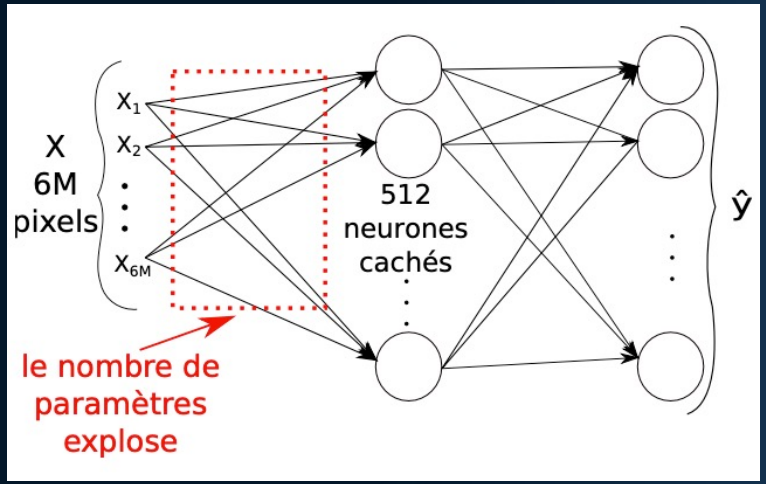
- Un tenseur : une matrice à plus de deux dimensions
- Dans le cas suivant : un cube de dimension (1000, 2000, 3)
- Au total 1000 * 2000 * 3 = 6M valeurs décimales

Une image pour un perceptron multi-couches ?



Perceptron multi-couches pour les images

Problème de stockage

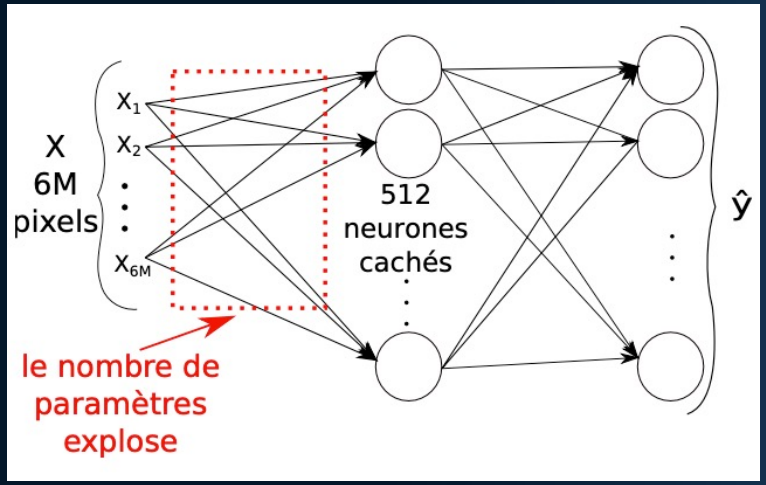


La matrice de la première couche sera :

$$W^{(1)} = \begin{bmatrix} w_{1,1} & \cdots & w_{1,6M} \\ \vdots & \ddots & \vdots \\ w_{512,1} & \cdots & w_{512,6M} \end{bmatrix}$$

Perceptron multi-couches pour les images

Problème de stockage

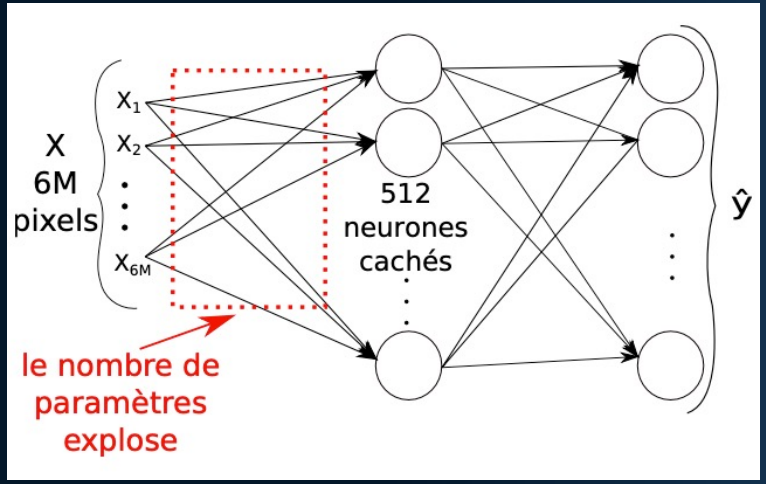


- La taille de la matrice sera de 6 millions de ligne et 512 colonnes
- Ce qui fera au total 6000000*512=3072000000 valeurs décimales
- Si chaque valeur décimale nécessite 4 octets

⇒ Au total on aura besoin de 12,28 Go pour stocker uniquement $W^{(1)}$

Perceptron multi-couches pour les images

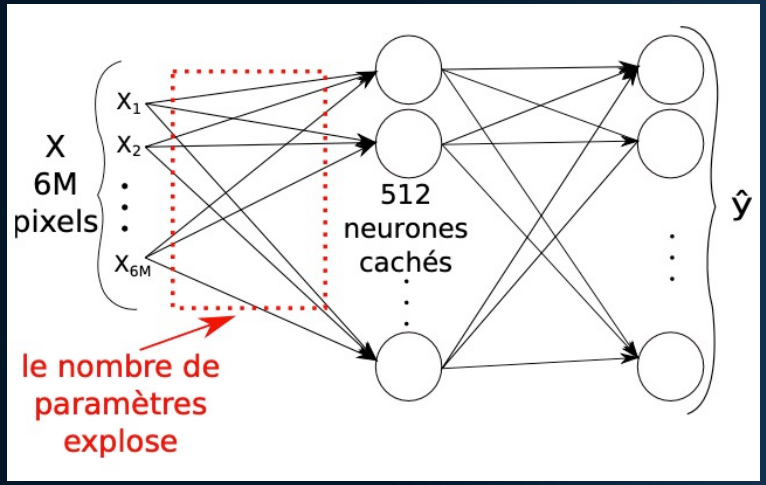
Problème de temps du traitement



- Avec une matrice aussi grande on a besoin de plusieurs heures pour faire une seule opération algébrique dessus
- Imaginez si on utilise un réseau à 100 couches cachées
- Donc le temps de calcul augmente de manière significative

Perceptron multi-couches pour les images

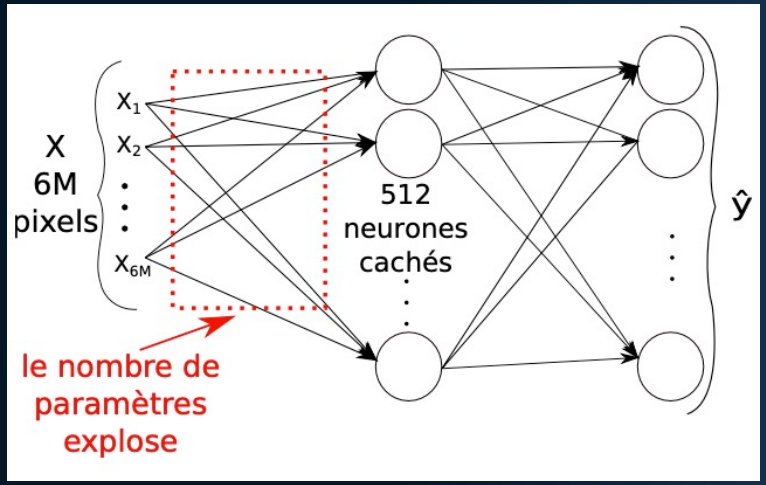
Problème de quantité de données



- Plus le modèle devient complexe (nombre de w augmente)
- Plus on a besoin des données pour l'entraîner
- Les données ne sont pas toujours disponibles dans tous les domaines

Perceptron multi-couches pour les images

Problème d'information spatiale



- Tous les neurones sont liés à tous les pixels
- Par contre un pixel est corrélé à ses pixels voisins
- Alors l'information spatiale (emplacement des pixels) est perdue

Perceptron multi-couches pour les images

Problèmes

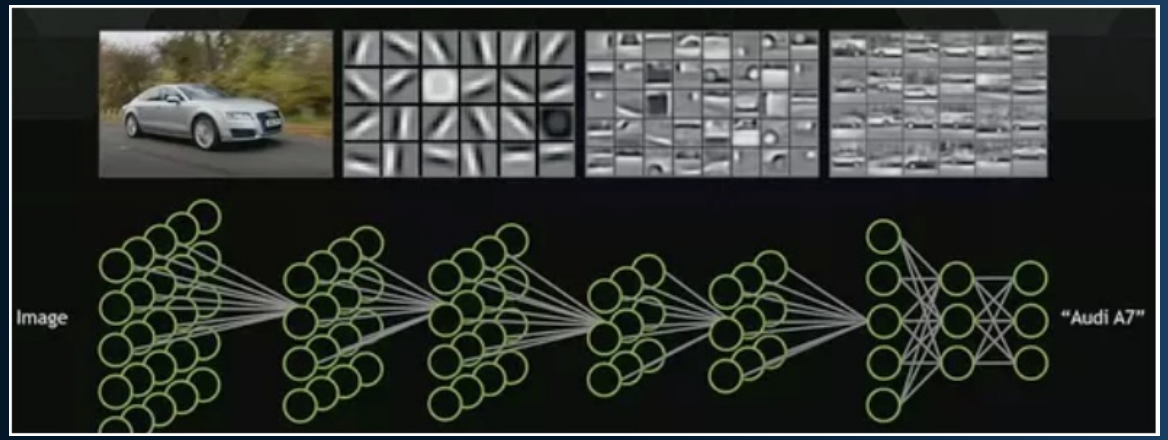
1. Explosion du nombre de paramètres
2. Augmentation du temps de calcul
3. Besoin d'une très grande quantité de données
4. Perte de l'information spatiale

⇒ **Réseaux convolutifs**

Table des matières

Les réseaux de neurones convolutifs

Convolutional Neural Networks (CNNs)

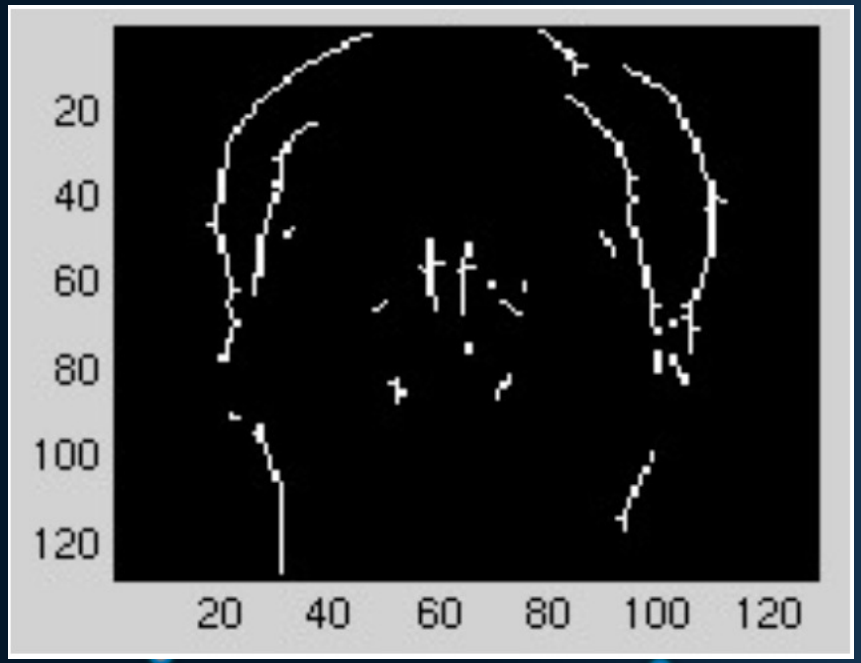


Par exemple les neurones:

- De la première couche détectent les contours
- De la deuxième couche identifient les sous-parties d'une voiture
- La dernière couche extrait les caractéristiques d'une marque

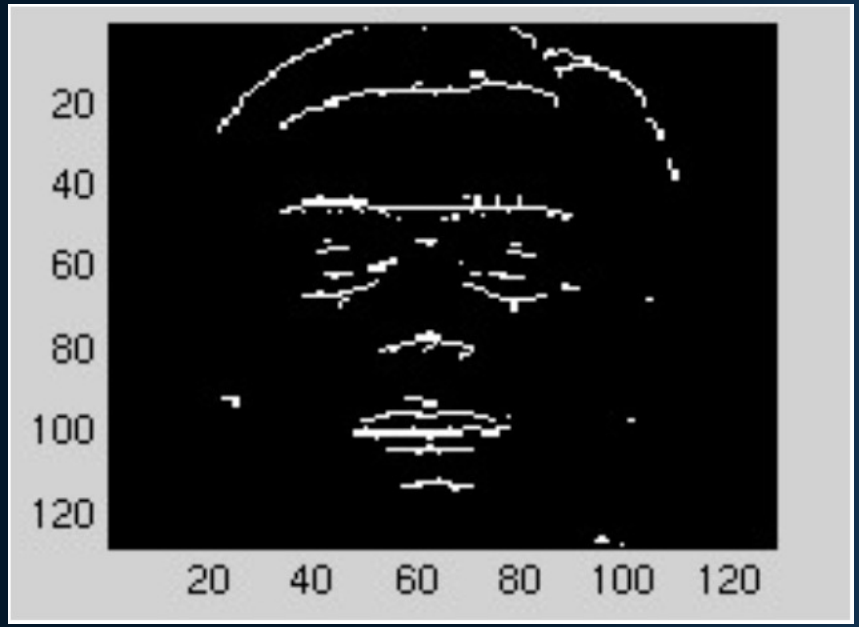
Détection de contours

Par exemple un neurone détectera les contours verticaux

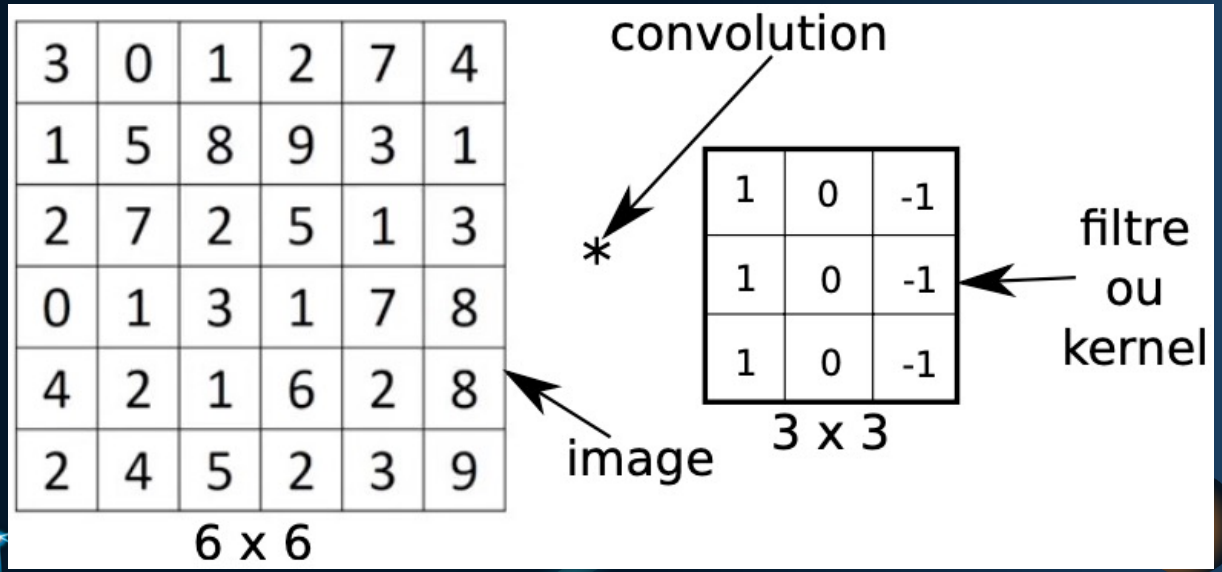


Détection de contours

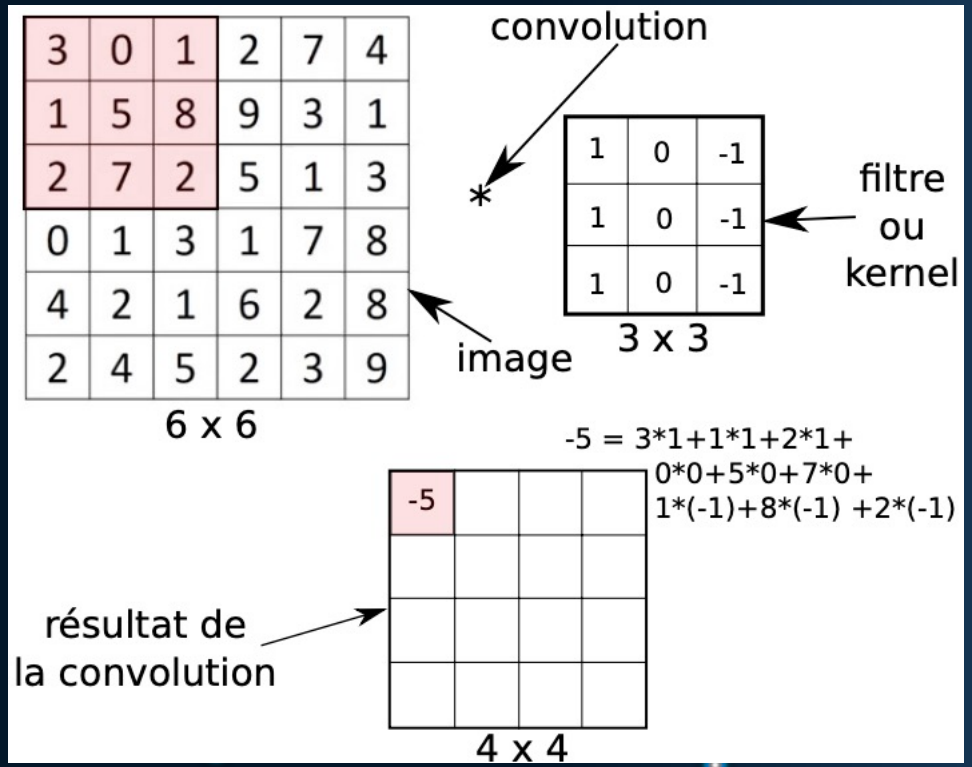
Par exemple un neurone détectera les contours horizontaux



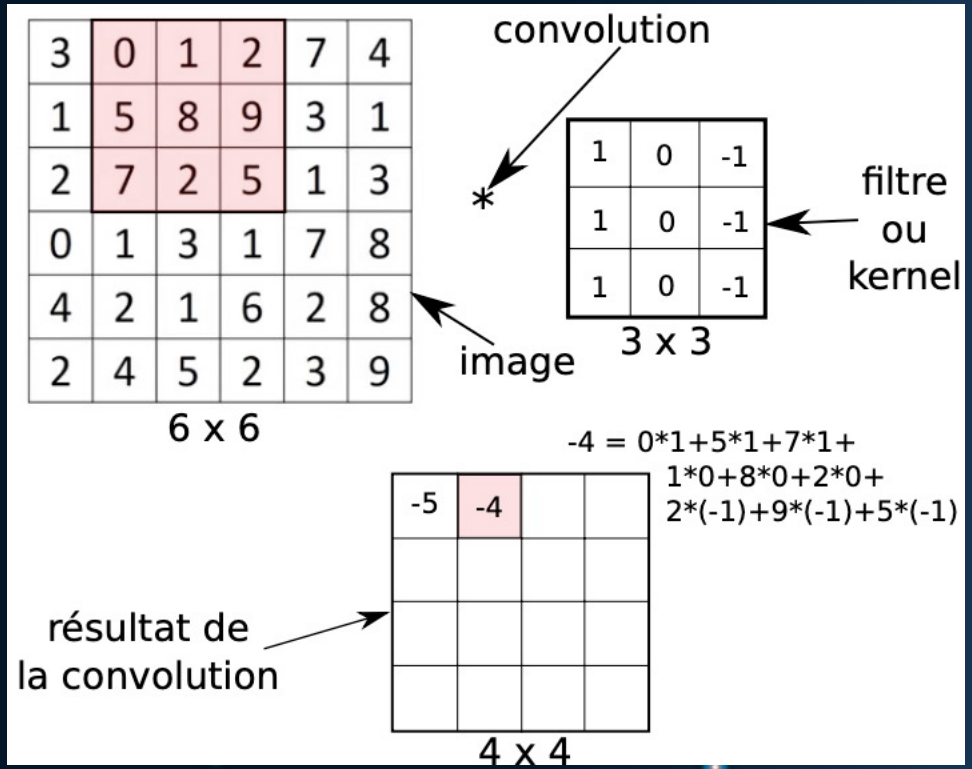
Exemple de convolution



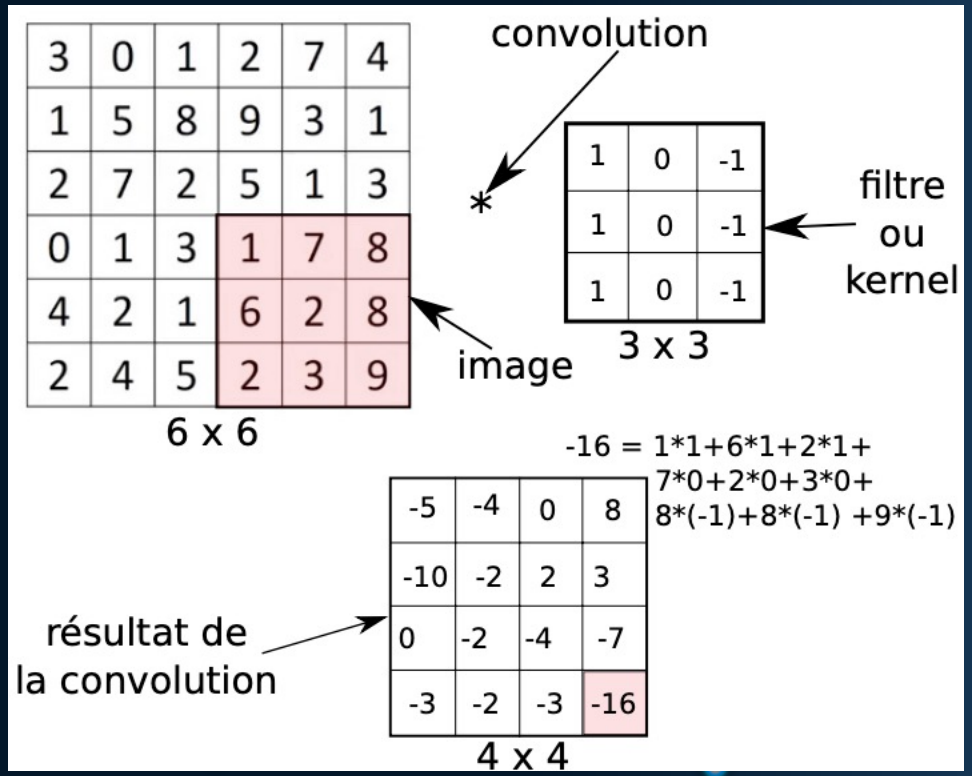
Exemple de détection de contour



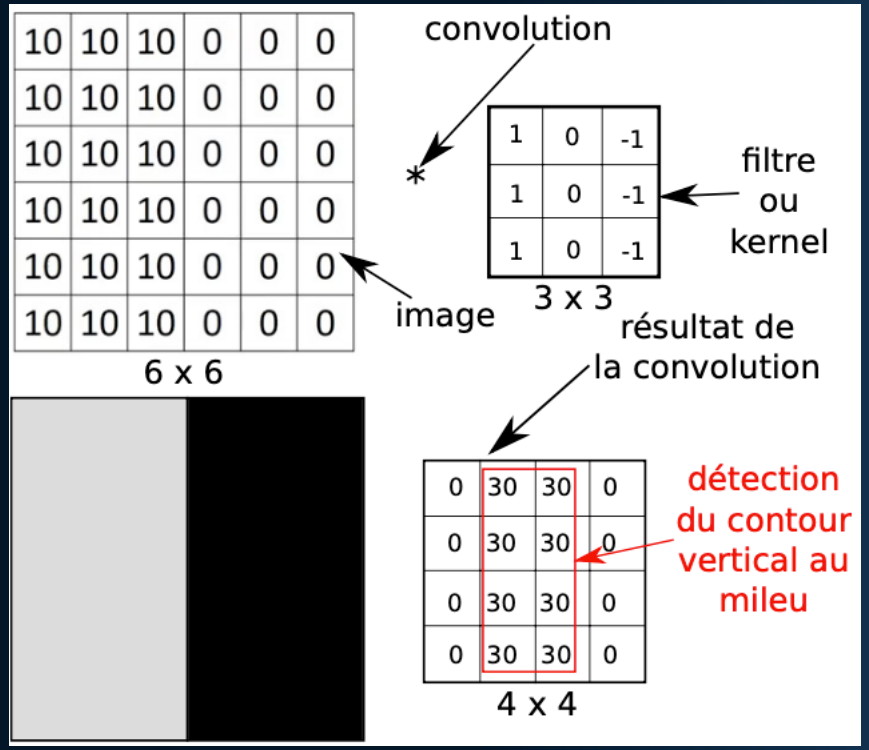
Exemple de détection de contour



Exemple de détection de contour

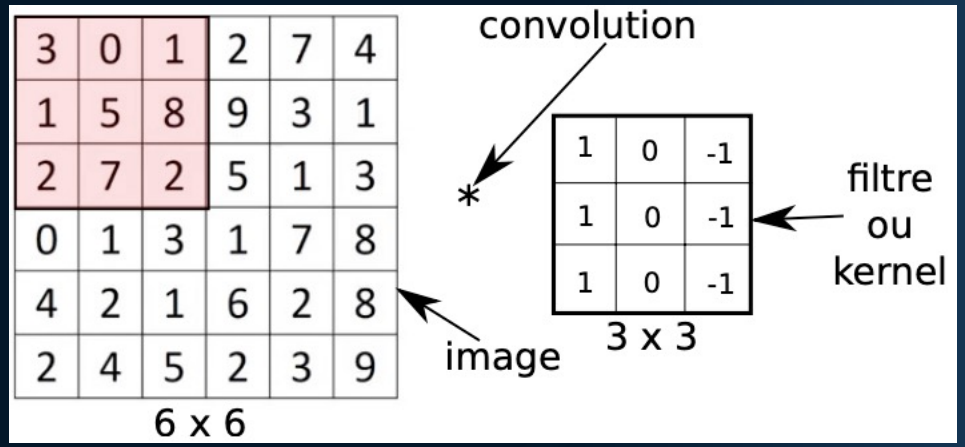


Exemple de détection de contour vertical



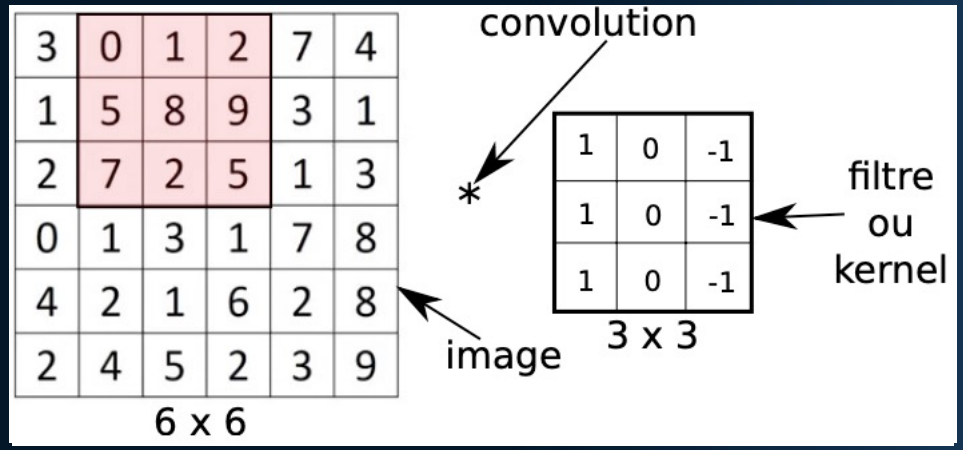
Le pas de la convolution (Stride)

Stride = 1



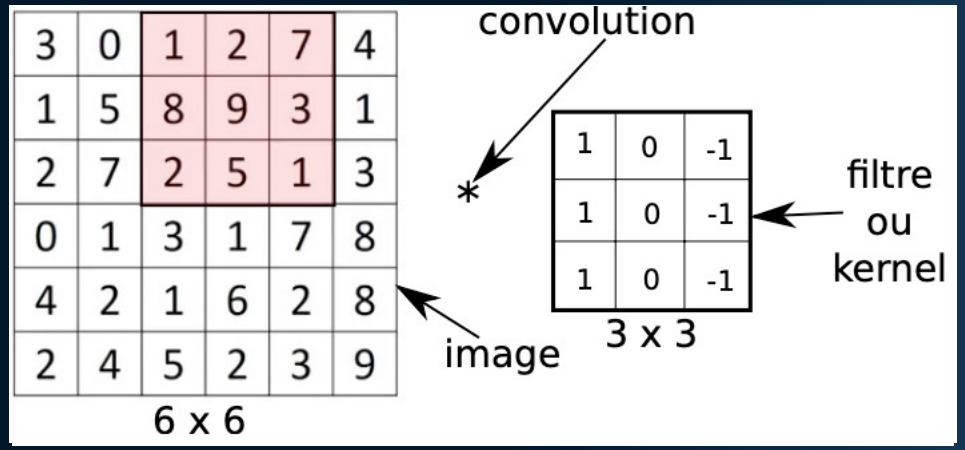
Le pas de la convolution (Stride)

Stride = 1



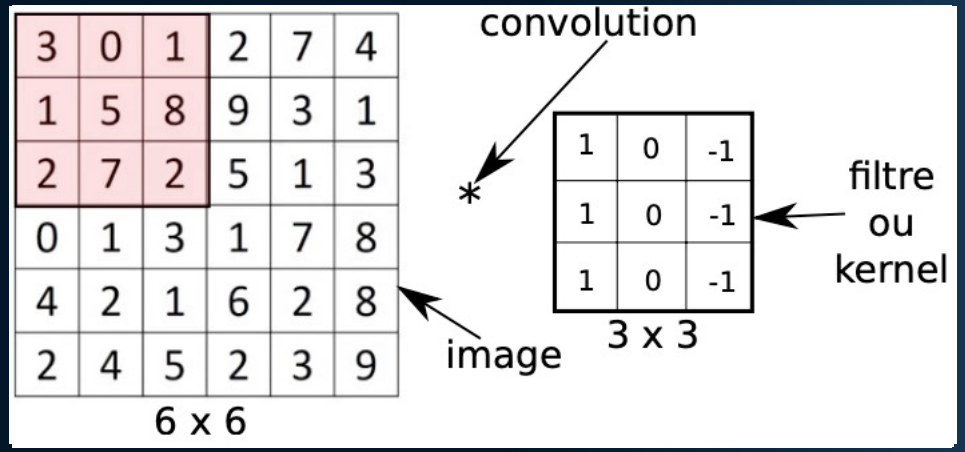
Le pas de la convolution (Stride)

Stride = 1



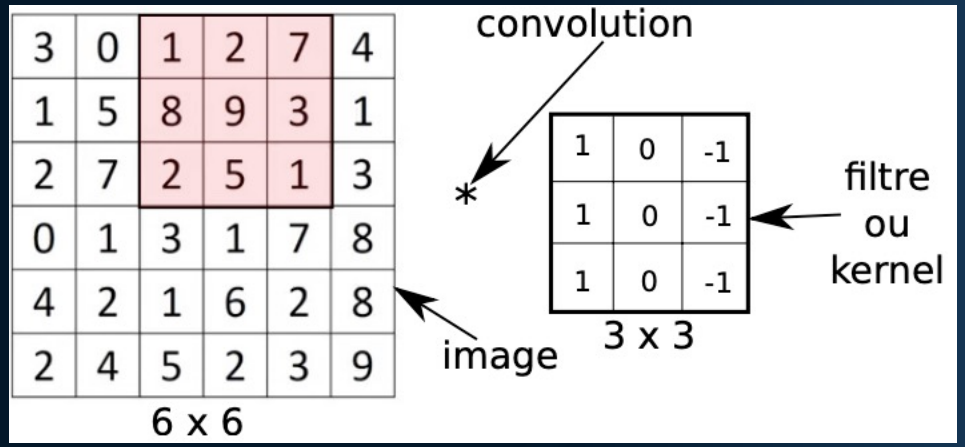
Le pas de la convolution (Stride)

Stride = 2



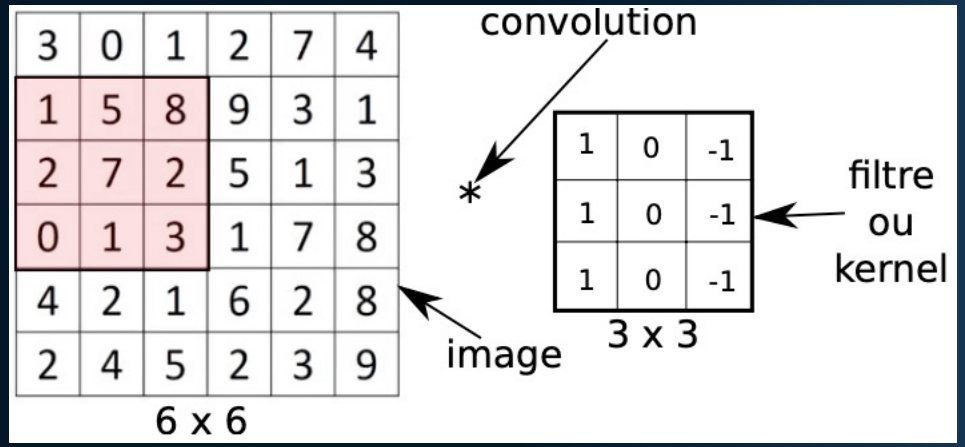
Le pas de la convolution (Stride)

Stride = 2

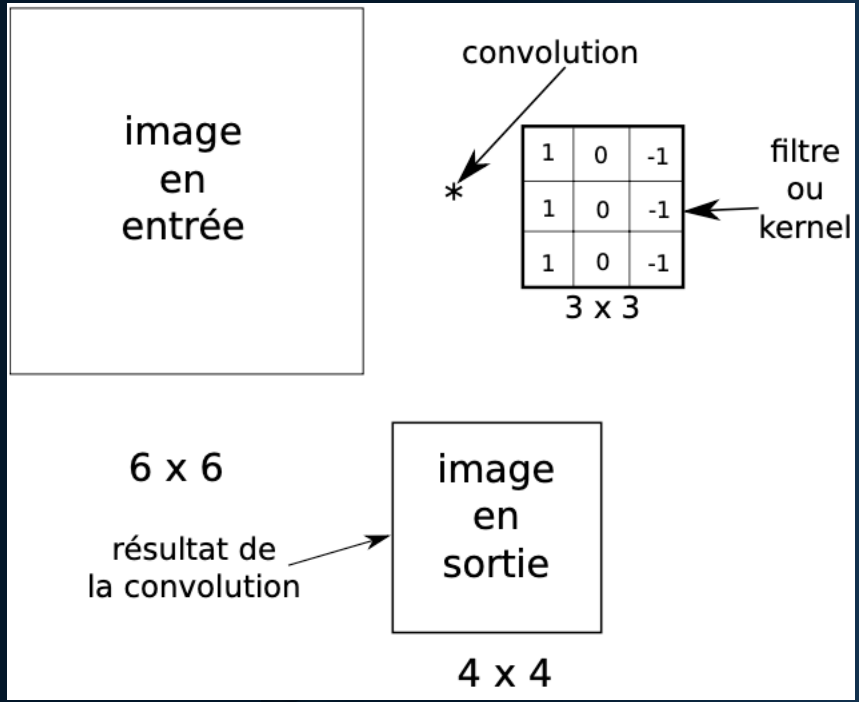


Le pas de la convolution (Stride)

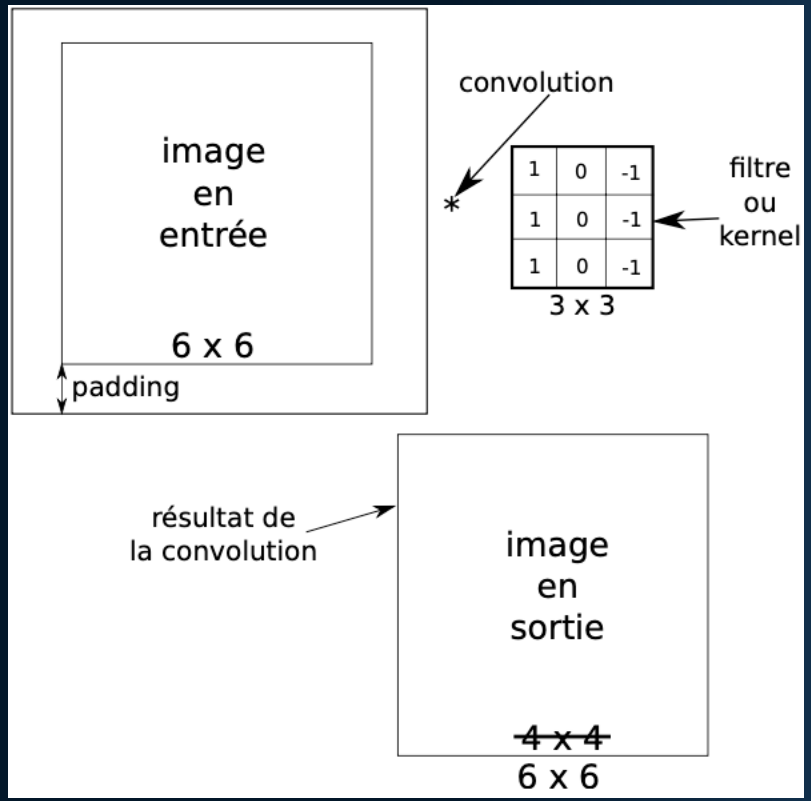
Stride = 2



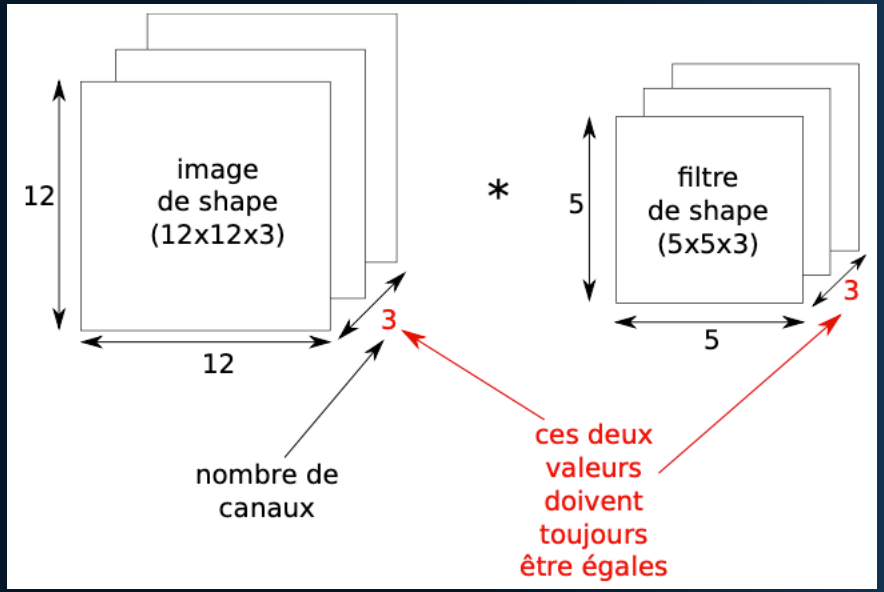
Remplissage de la bordure d’image (Padding)



Remplissage de la bordure d’image (Padding)



Convolution sur des images RGB



A noter que le résultat d'une convolution ne dépend pas du nombre de canaux

Résultat d'une convolution

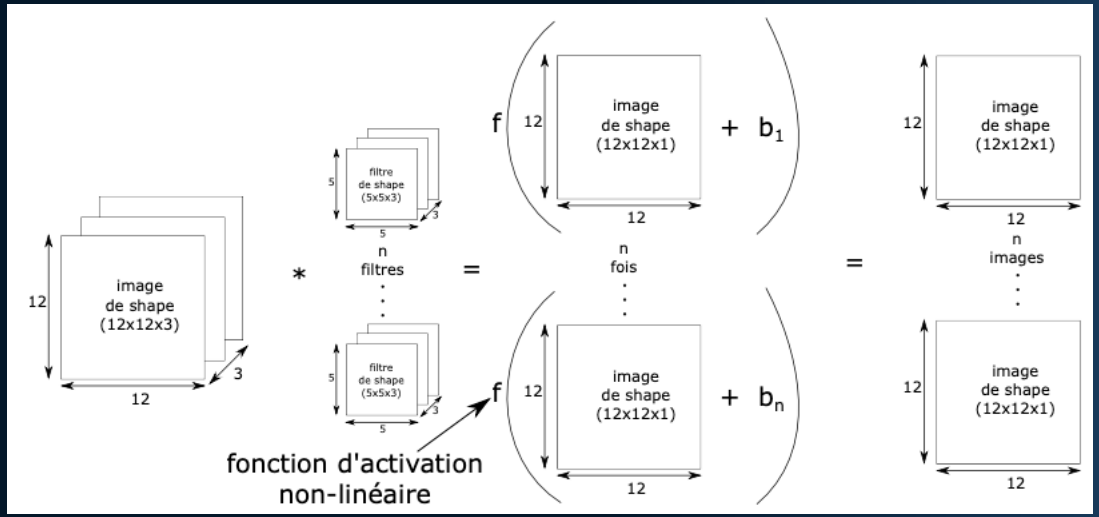
Le résultat d'une convolution sur une image est une image de taille

$$\left[\frac{n+2p-f}{s}+1\right] * \left[\frac{n+2p-f}{s}+1\right]$$

Avec :

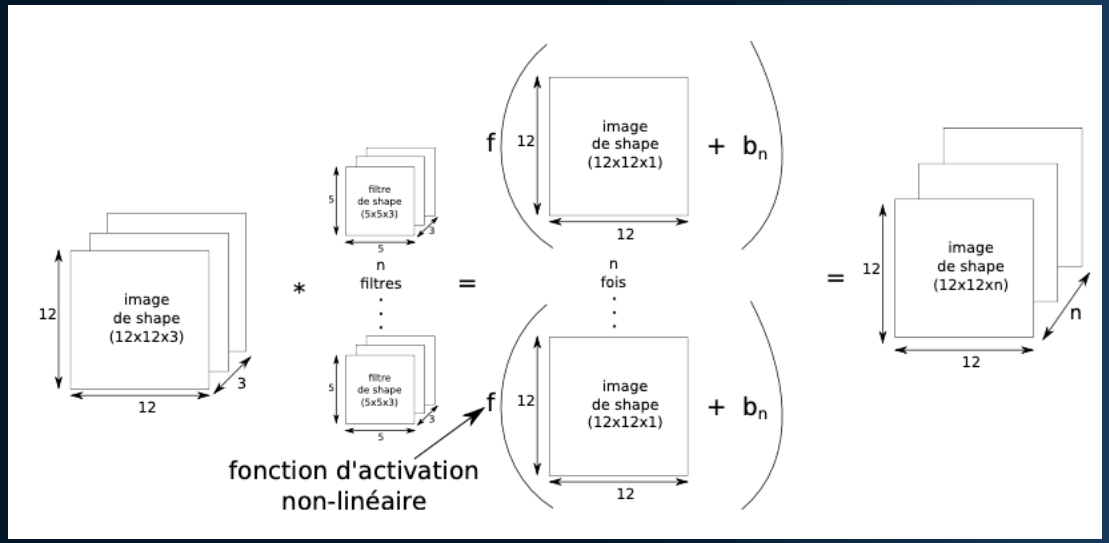
- L'image en entrée ayant une taille : $n * n$
- Le filtre a une dimension $f * f$
- Le stride a une valeur s
  - e.g. pour $s = 2$ on glisse le filtre de tous les deux pixels
- Le padding a une valeur p
  - e.g. pour $p = 2$ on remplit le contour de l'image avec 2 pixels
- $[x] = int(x)$

Une couche convolutive



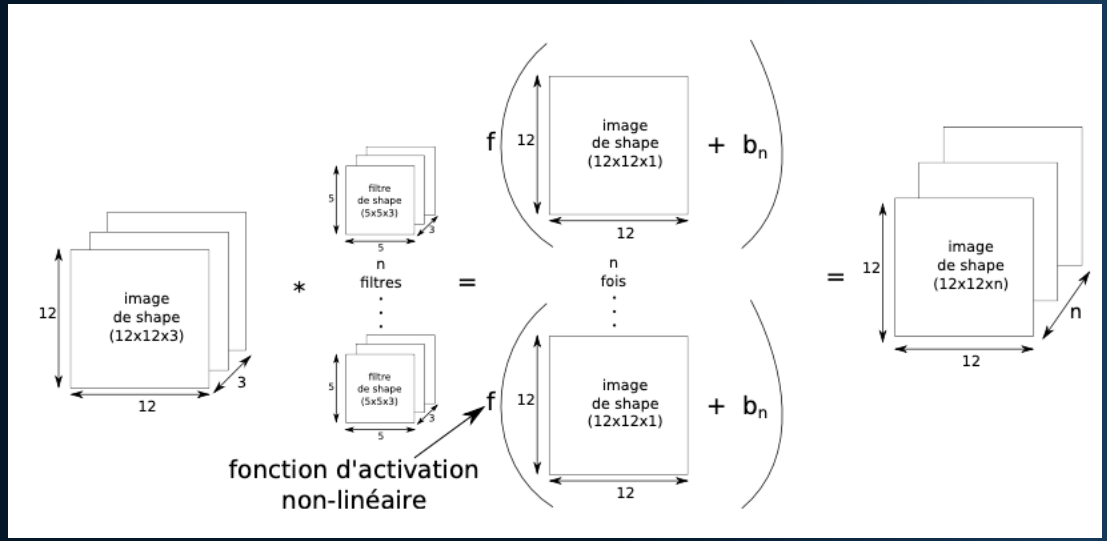
- Tout d'abord on applique n convolutions
- Le résultat des convolutions est une série de n images
- Chaque image passera par une fonction non-linéaire f
- Le résultat de f est une autre image de la même taille

Résultat d’une couche convolutive



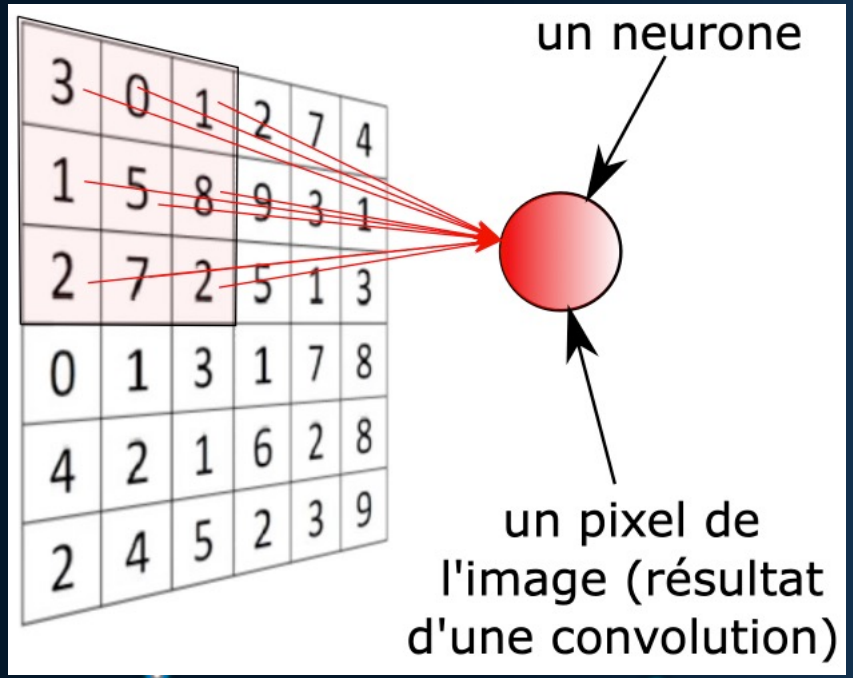
Donc le résultat d’une couche convolutive (n filtres) appliquée à une image sera considéré comme une autre image (qui a subi des opérations non-linéaires) avec n canaux.

Les paramètres à apprendre pour une convolution

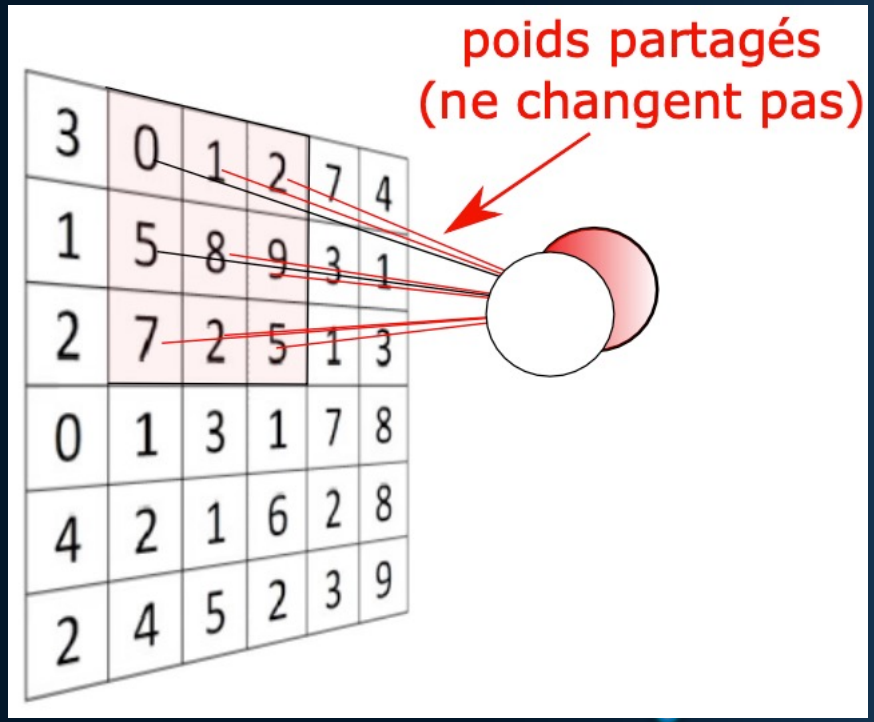


Donc pour un seul filtre de taille $(3 * 3 * 3)$ on a $27\ w$ à apprendre ainsi qu'un b. Alors pour une couche de 4 filtres de taille $(3 * 3 * 3)$ on aura $3 * 27 = 108w$ à apprendre ainsi que $4b \Rightarrow 112$ paramètres au total. Par contre un MLP aura $12 * 12 * 3 + 1 = 432$ paramètres à apprendre pour un neurone.

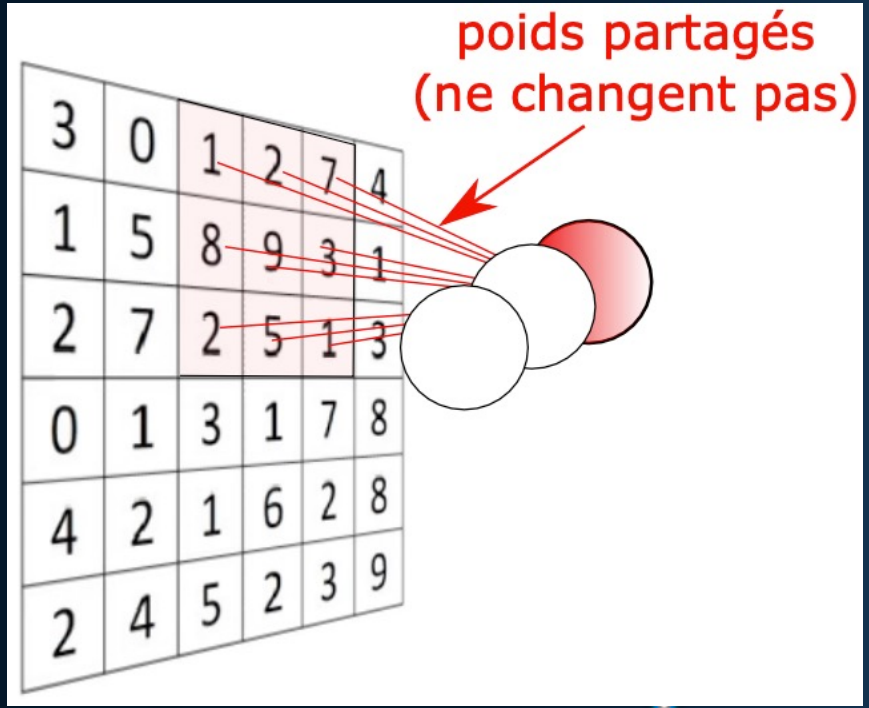
Une couche convolutive sous forme des neurones



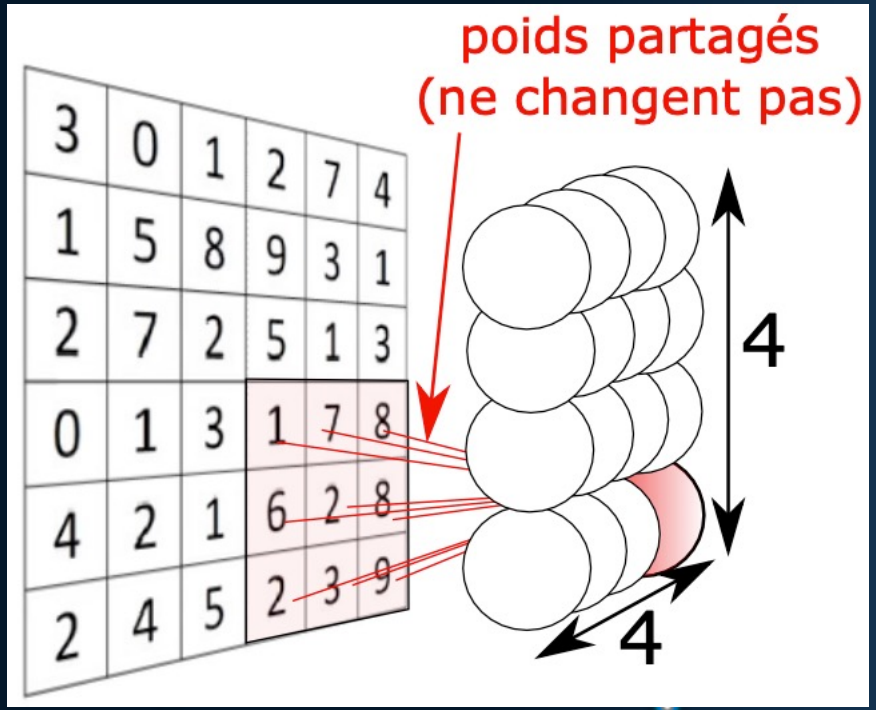
Une couche convolutive sous forme des neurones



Une couche convolutive sous forme des neurones



Une couche convolutive sous forme des neurones



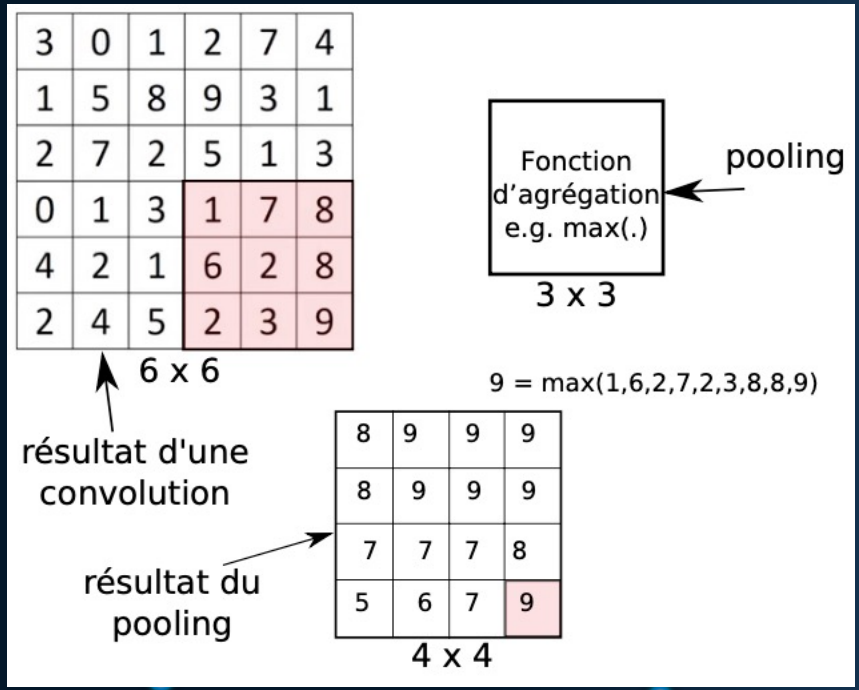
Avantage des CNNs par rapport aux MLPs

Avantage des CNNs par rapport aux MLPs :

- Moins de paramètres à apprendre avec les CNNs
- L'information spatiale est conservée
- Un filtre qui glisse sur une image extrait des caractéristiques invariantes à l'emplacement de l'objet dans l'image
    - E.g. on peut détecter un chat quel que soit sa position sur l'image

Toutes ces caractéristiques sont le résultat de la propriété de partage des poids (weight sharing)

Pooling


3 0 1 2 7 4
1 5 8 9 3 1
2 7 2 5 1 3
0 1 3 1 7 8
4 2 1 6 2 8
2 4 5 2 3 9
6 x 6
résultat d'une convolution
Fonction d'agrégation e.g. max(.)
pooling
3 x 3
9 = max(1,6,2,7,2,3,8,8,9)
8 9 9 9
8 9 9 9
7 7 7 8
5 6 7 9
résultat du pooling
4 x 4

Pooling

Opération de pooling :

- Le pooling consiste à appliquer une agrégation (e.g. max, avg, etc.)
- Il réduit la taille d'une image en entrée
- Cette opération ne contient pas de paramètres w à apporendre
- Le but est de rendre le réseau invariant aux petites variations dans l'image
- Cette opération possède un stride et un pool size, comme la convolution
- Global pooling, on fait l'agrégation sur toute l'imagen donc on transforme l'image en un seul point

Pooling

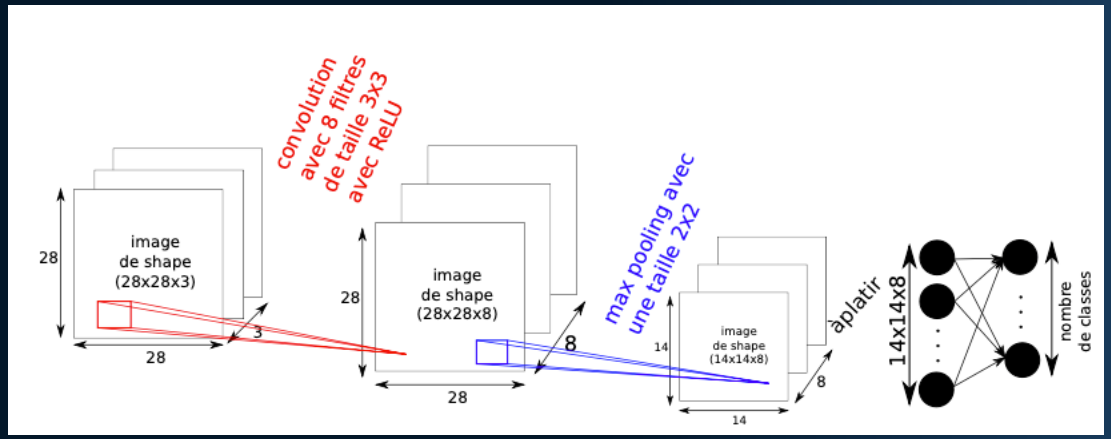
Le résultat d'un pooling sur une image est une image de taille réduite :

$$\left[\frac{n+2p-f}{s}+1\right] * \left[\frac{n+2p-f}{s}+1\right]$$

Avec :

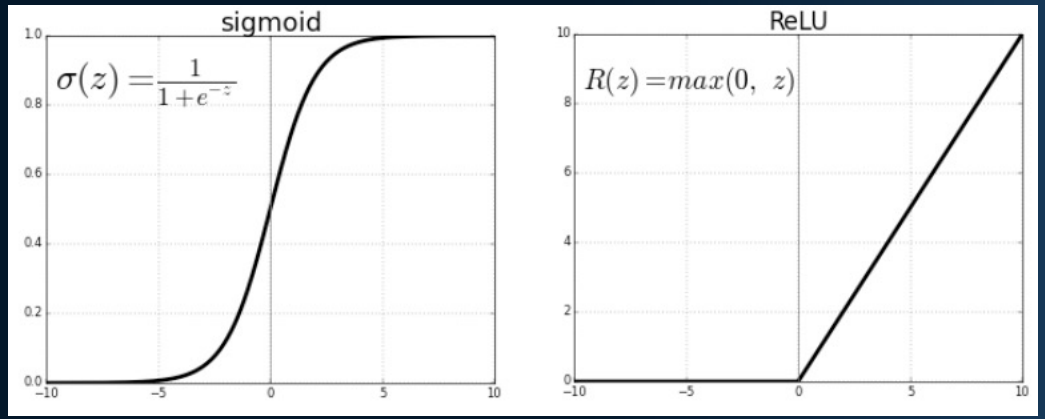
- L'image en entrée ayant une taille : $n * n$
- Le filtre a une dimension $f * f$
- Le stride a une valeur s (généralement égale à la taille du filtre)
- Le padding a une valeur p (généralement zéro)
- $[x] = int(x)$

Les réseaux convolutifs : une cascade de couches convolutives



L’entrainement est similaire à l’entrainement d’un MLP (sauf qu’ici on respecte la propriété du weight sharing)

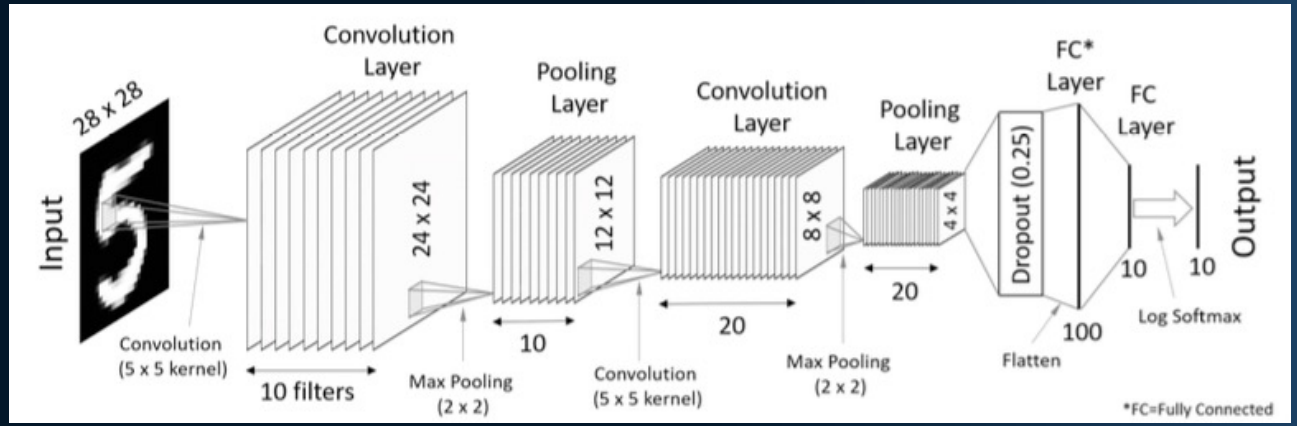
Fonction d’activation



- Généralement ReLu (Rectified Linear Unit) est plus utilisée
- Pour une grande valeur de z, ReLU donne toujours un gradient
- ReLU nécessite moins de temps de calcul que σ

Table des matières

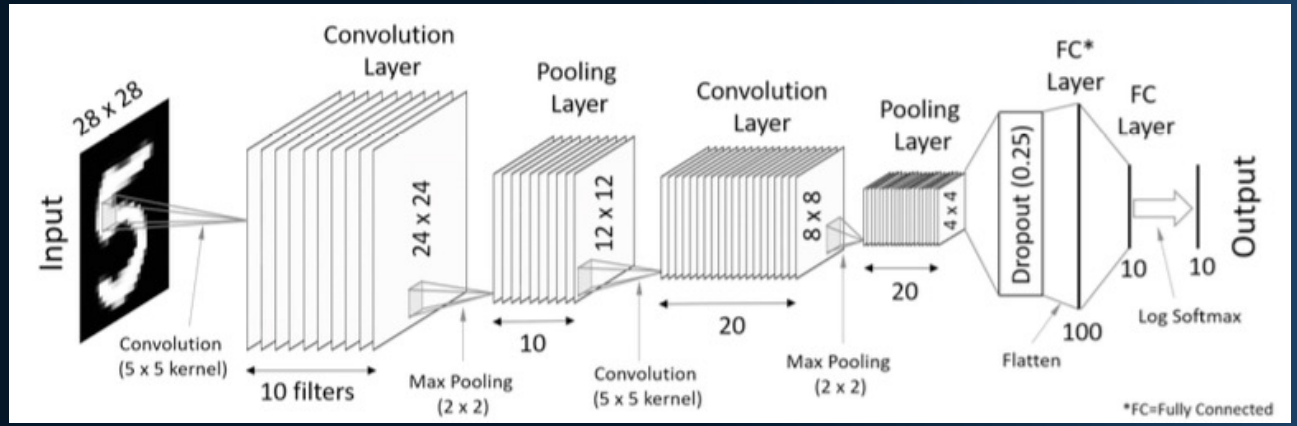
Exemple d’un CNN pour le jeu de données MNIST



Réseau profond convolutif avec 3 couches cachées et une sortie

- Première couche convolutive
  1. Contient 10 filtres de taille $5 * 5$ avec $s = 1$ et $p = 0$
  2. Utilise la fonction d’activation ReLU
  3. Suivie d’un pooling (max) de taille $2 * 2$

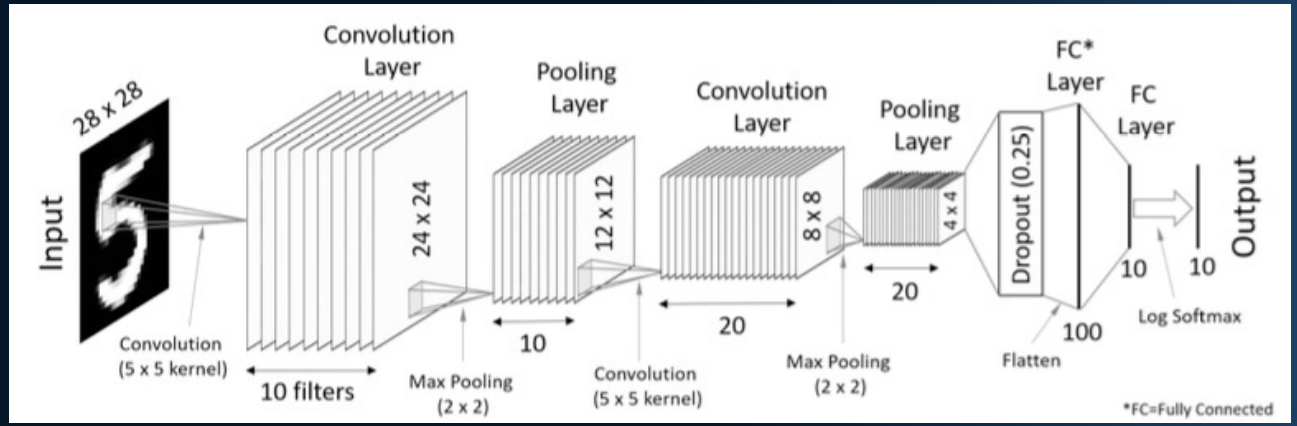
Exemple d'un CNN pour le jeu de données MNIST



Réseau profond convolutif avec 3 couches cachées et une sortie

- Deuxième couche convolutive
  1. Contient 20 filtres de taille $5 * 5 * 10$ avec $s = 1$ et $p = 0$
  2. Utilise la fonction d'activation ReLU
  3. Suivie d'un pooling (max) de taille $2 * 2$

Exemple d’un CNN pour le jeu de données MNIST



Réseau profond convolutif avec 3 couches cachées et une sortie

- Troisième couche est complètement connectée (Dense)
    1. Contient 100 neurones avec la fonction d’activation ReLU
- Dernière couche est complètement connectée (Dense)
    1. Contient 10 neurones avec la fonction d’activation softmax

De la théorie vers la pratique

Convolution

Import des librairies

In [ ]:

```
%tensorflow_version 2.x
from sklearn.preprocessing import normalize
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
import random

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.datasets import mnist
```

Charger les données MNIST

In [ ]:

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# réduire la taille du train pour accélérer les experiences
nnn=1000
x_train = x_train[:nnn]
y_train = y_train[:nnn]

print("x_train",x_train.shape)
print("Nombre d'image pour entrainer:",x_train.shape[0])
print("Nombre d'image pour tester:",x_test.shape[0])
print("La taille d'une image:",x_train.shape[1:],"ce qui fait au total:",
      x_train.shape[1]*x_train.shape[2],"pixels")
print("La liste des classes:",np.unique(y_train))
```

Télécharger le fichier (https://github.com/r-wenger/cours_m1-m2-OTG/blob/main/IA_geosciences_M2/CM/5_DL_Convolution.ipynb) et l'importer sur Google Colab