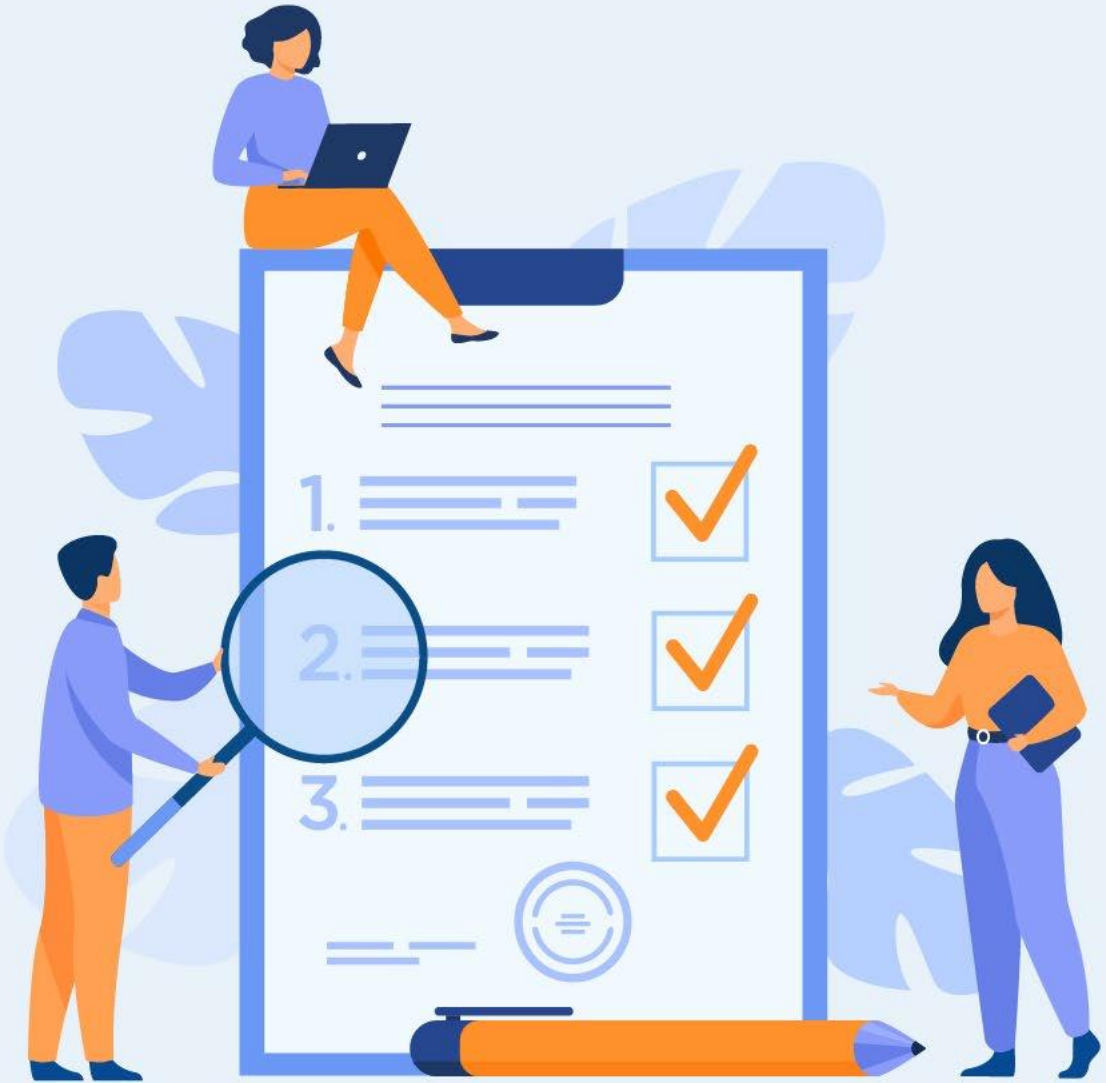
PARTIE 2

MANIPULER LES RÉSEAUX DE NEURONES AVANCÉS

Dans ce module, vous allez :

- Construire des réseaux à convolution
- Optimiser les réseaux à convolution

33 heures



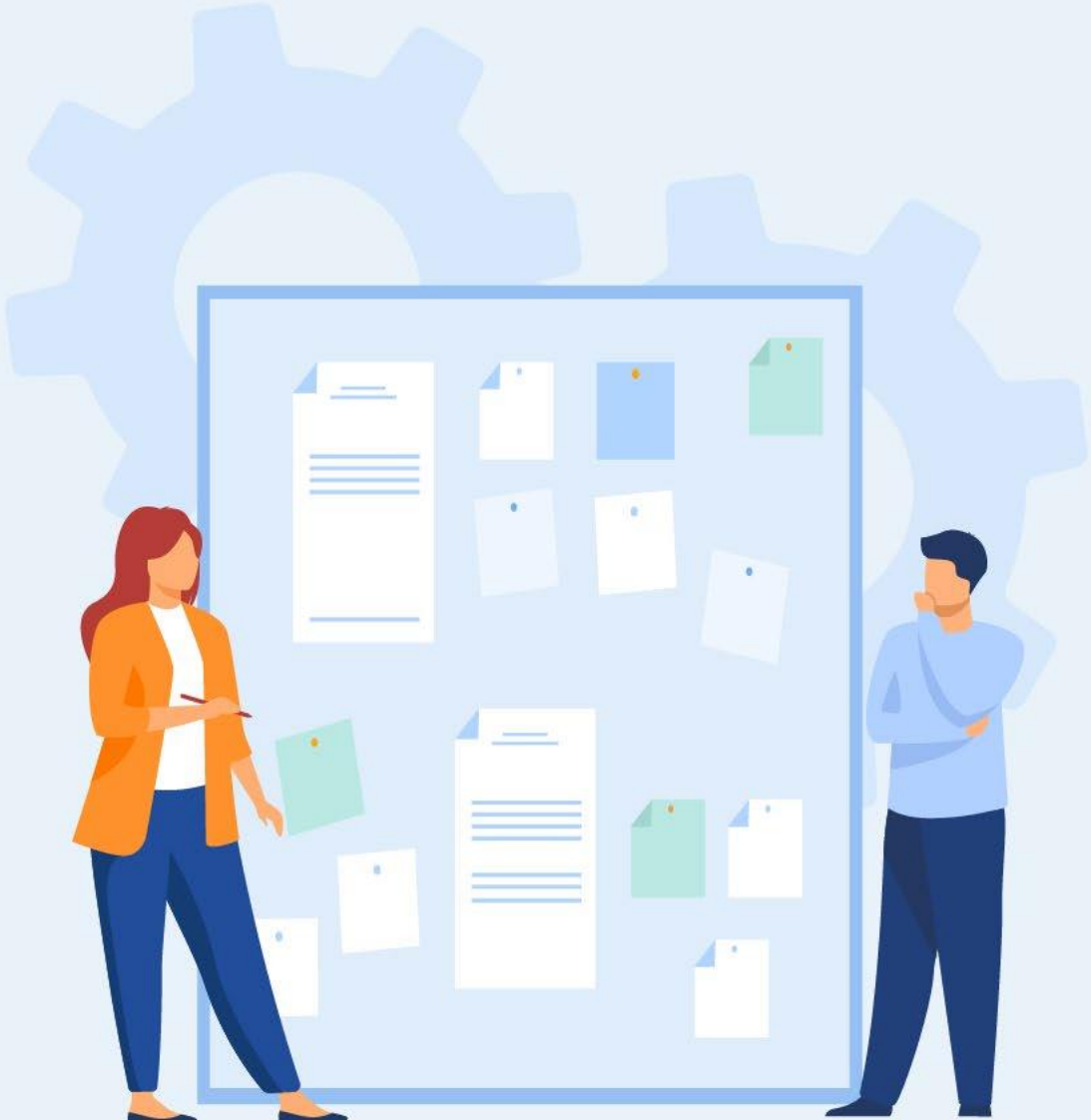
CHAPITRE 1

CONSTRUIRE DES RÉSEAUX À CONVOLUTION

Ce que vous allez apprendre dans ce chapitre :

- Comprendre les réseaux à convolution
- Assimiler les différentes couches d’un réseau de neurones à convolution

23 heures



CHAPITRE 1

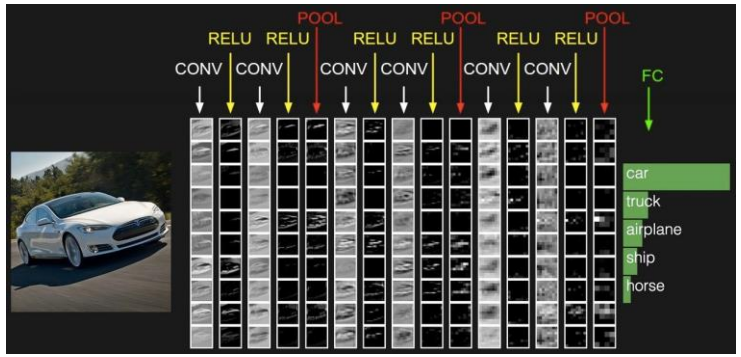
CONSTRUIRE DES RÉSEAUX À CONVOLUTION

1. **Définition**
2. Architecture des réseaux à convolution
3. Apprentissage des réseaux à convolution
4. Quelques exemples des réseaux à convolution

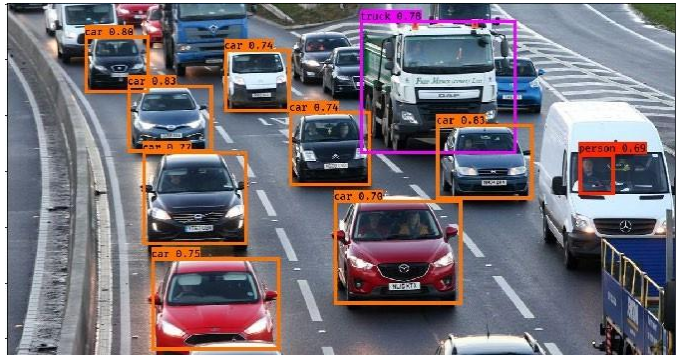
01 - CONSTRUIRE DES RÉSEAUX À CONVOLUTION

Définition

Récemment, les progrès scientifiques et technologiques ont donné naissance à plusieurs applications telles que la classification d'images, la détection d'objets, la segmentation sémantique, etc.



Classification



Détection

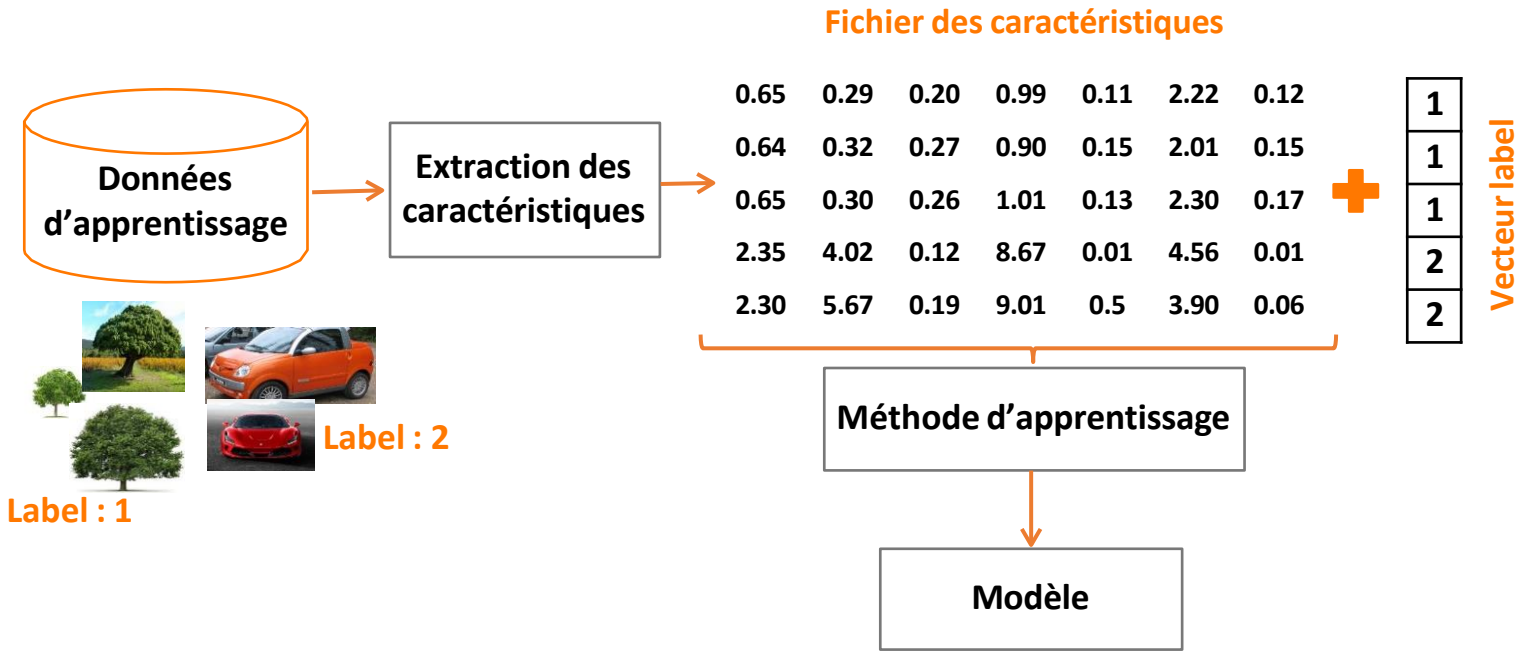
Segmentation sémantique

PARTIE 2

Pipeline traditionnelle de la classification

Traditionnellement, la classification comporte deux parties primordiales :

- ✓ la première partie c'est l'extraction des caractéristiques ;
- ✓ la deuxième partie c'est l'apprentissage.



PARTIE 2

Pipeline traditionnelle de la classification

Une fois qu'un modèle d'apprentissage a été entraîné, il peut être utilisé pour faire des prédictions sur de nouvelles données non vues auparavant. La prédiction après l'apprentissage implique d'appliquer le modèle entraîné à ces nouvelles données pour obtenir des sorties ou des étiquettes prévues.

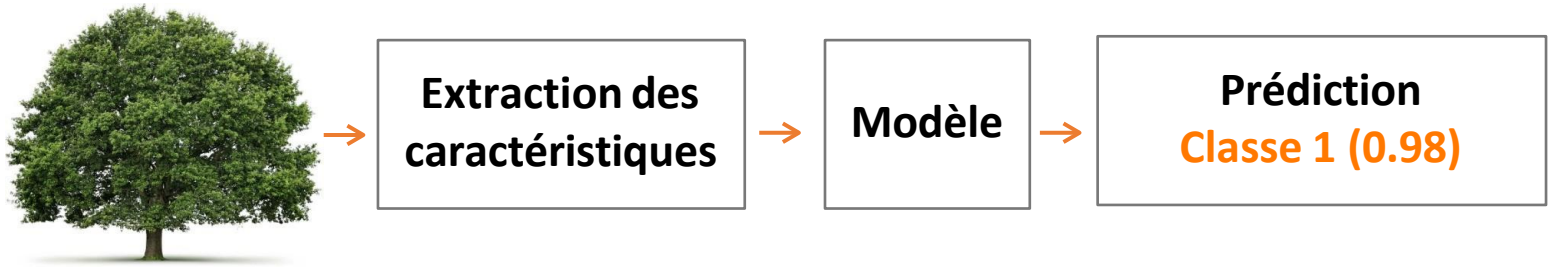
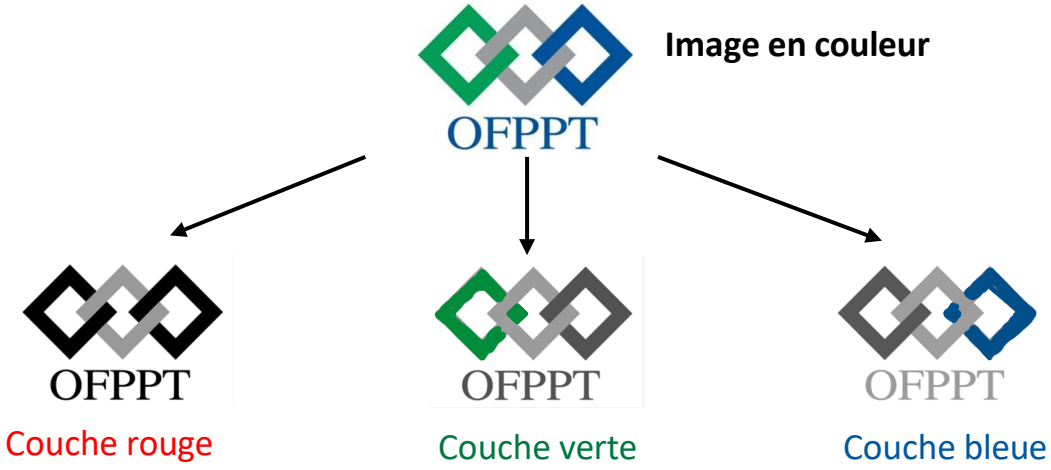


Image en couleur

Une image numérique est constituée par des milliers de points que nous appelons pixels. Le pixel est la plus petite composante carrée d'une image numérique et il peut prendre seulement une seule couleur. Il est possible de distinguer chaque pixel en grossissant fortement l'image. Logiquement, plus une image comporte de pixels, plus elle est détaillée.

Une image en couleur est constituée de trois couches (ou canaux) : une couche rouge (R), une couche verte (V), une couche bleue (B).

Soit *Nx* le nombre de colonnes de l'image et *Ny* le nombre de lignes. Le nombre de pixels total est *N=Nx*Ny*. Chaque couche est une matrice comportant *Ny* lignes et *Nx* colonnes. Le plus souvent, cette matrice contient des entiers codés sur 8 bits (les valeurs vont de 0 à 255).



PARTIE 2

Image niveaux de gris

Une image en niveaux de gris est une image dans laquelle la valeur de chaque pixel représente non une couleur mais uniquement une quantité de lumière. c'est-à-dire qu'il ne contient que des informations d'intensité de lumière. Les images en niveaux de gris sont composées exclusivement de nuances de gris. Le contraste va du noir qui a l'intensité la plus faible au blanc qui représente l'intensité la plus forte.

Une image couleur peu facilement être convertit en image en niveaux de gris en calculons pour la moyenne des valeurs des trois couches (rouge, verte et bleue) de chaque pixel.

Remarque importante: il ne faut pas confondre une image en niveaux de gris avec une image en noir et blanc. Les image en noir et blanc sont des images binaires où chaque pixel ne peut prendre qu'une de deux états soit noir ou blanc mais pas des valeurs entre les deux.

Image couleur

Image en niveaux de gris

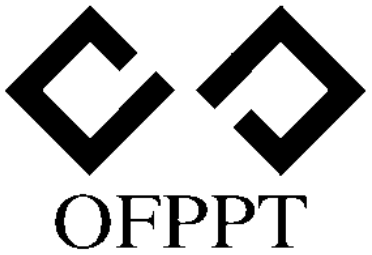
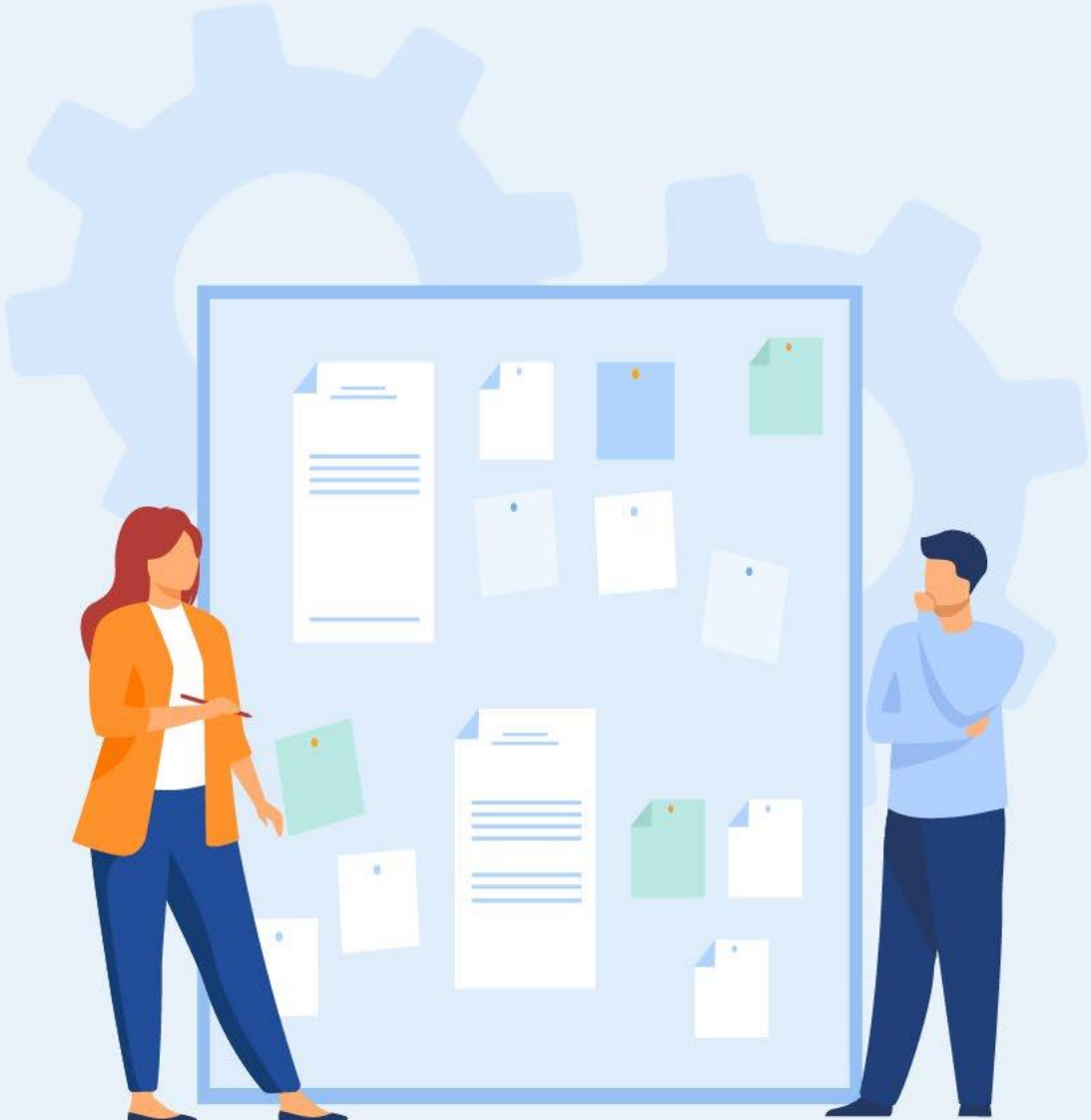


Image en noir et blanc



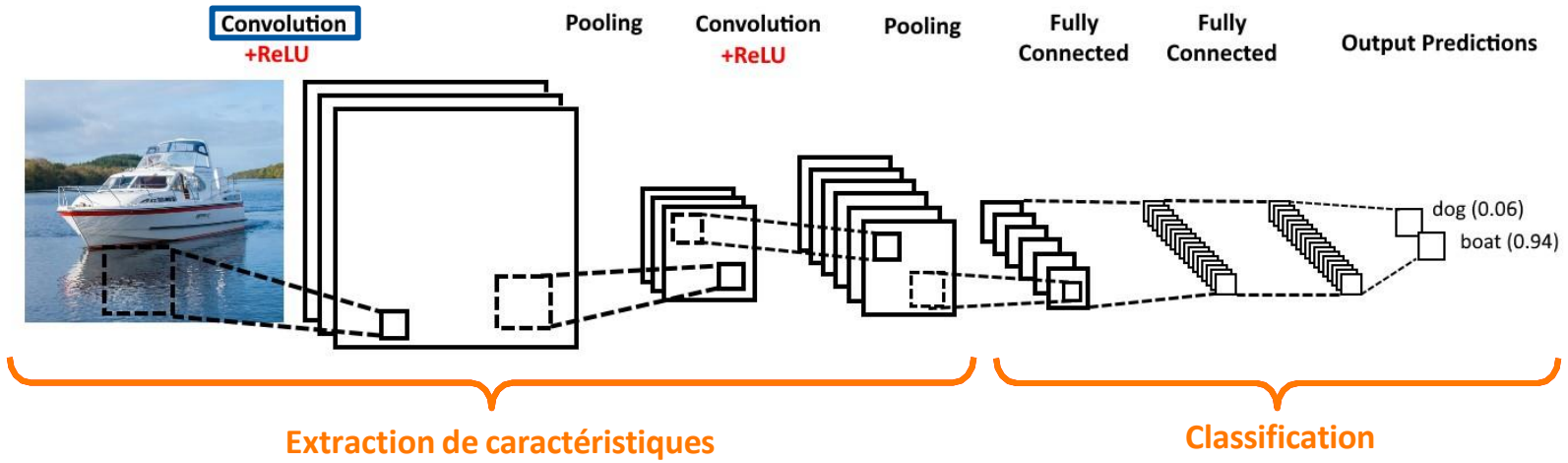
CHAPITRE 1

CONSTRUIRE DES RÉSEAUX À CONVOLUTION

1. Définition
2. **Architecture des réseaux à convolution**
3. Apprentissage des réseaux à convolution
4. Quelques exemples des réseaux à convolution

Architecture des réseaux à convolution

Les réseaux de neurones à convolution (Convolutional Neural Networks, CNN en anglais) sont des architectures spécifiquement conçues pour le traitement efficace des données structurées. Ce type de réseaux a révolutionné le traitement d'images et a surmonté l'extraction manuelle des caractéristiques. CNN est calculé directement sur les images et consiste en une ou plusieurs couches de convolution alternées avec des couches de sous-échantillonnage. Il comprend deux parties principales : la partie extraction des caractéristiques et la partie classification.



PARTIE 2

Couche de convolution

La convolution est une opération mathématique fondamentale utilisée dans divers domaines, notamment en traitement du signal, en vision par ordinateur et en apprentissage automatique. Dans le contexte des réseaux de neurones, la convolution est une opération clé effectuée par les couches de convolution des réseaux de neurones à convolution (CNN). La convolution est une opération qui combine deux fonctions pour produire une troisième fonction. Dans le cas des réseaux de neurones, la convolution est appliquée entre une entrée et un filtre (ou noyau) pour produire une carte de caractéristiques.

Détection des bords

Filtre

$$* \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$$

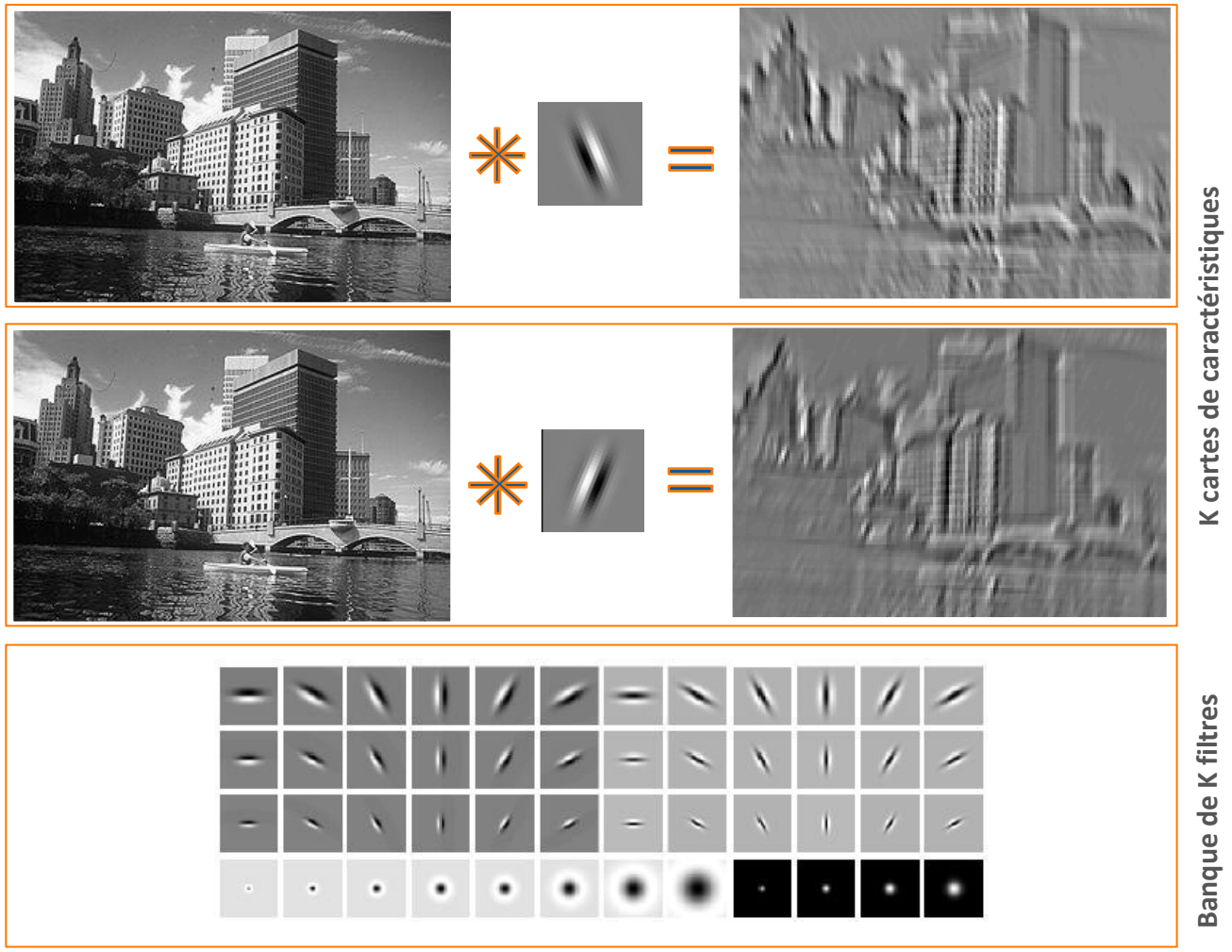
Sharpen

$$* \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} =$$

PARTIE 2

Couche de convolution

- ✓ Les images représentent les cartes de caractéristiques, également appelées « feature maps » en anglais.
- ✓ Elles sont des représentations spatiales des caractéristiques extraites d'une entrée à l'aide de filtres lors de la convolution dans un réseau de neurones à convolution.
- ✓ Lorsqu'un CNN est appliqué à une image ou à une autre forme de données structurées en grille, chaque couche de convolution produit une carte de caractéristiques en appliquant des filtres à l'entrée.
- ✓ Chaque pixel dans une carte de caractéristiques représente la réponse du filtre à une caractéristique spécifique dans une région de l'entrée.



filter 1

-1	-1	-1
1	1	1
0	0	0

filter 2

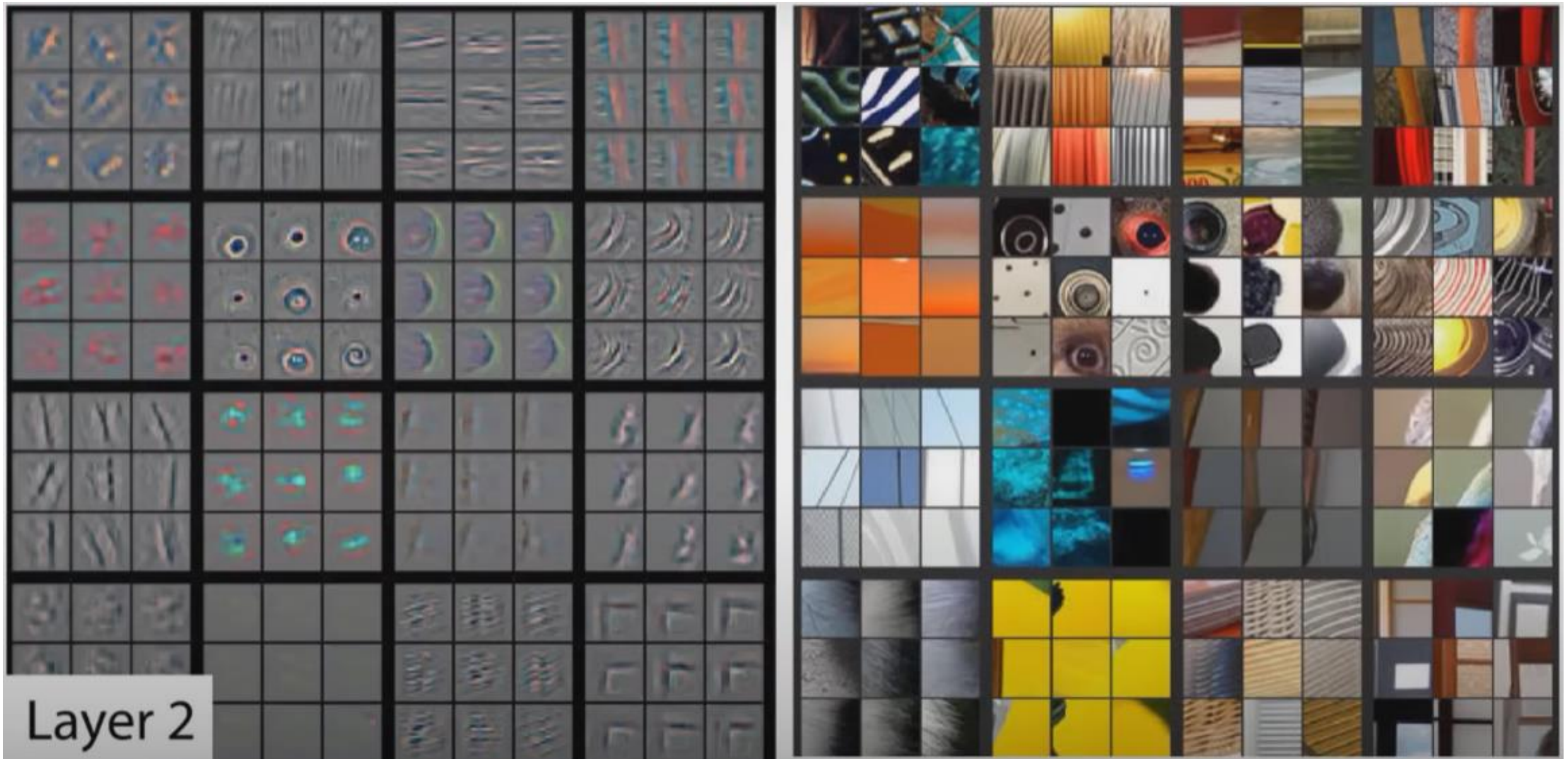
-1	1	0
-1	1	0
-1	1	0

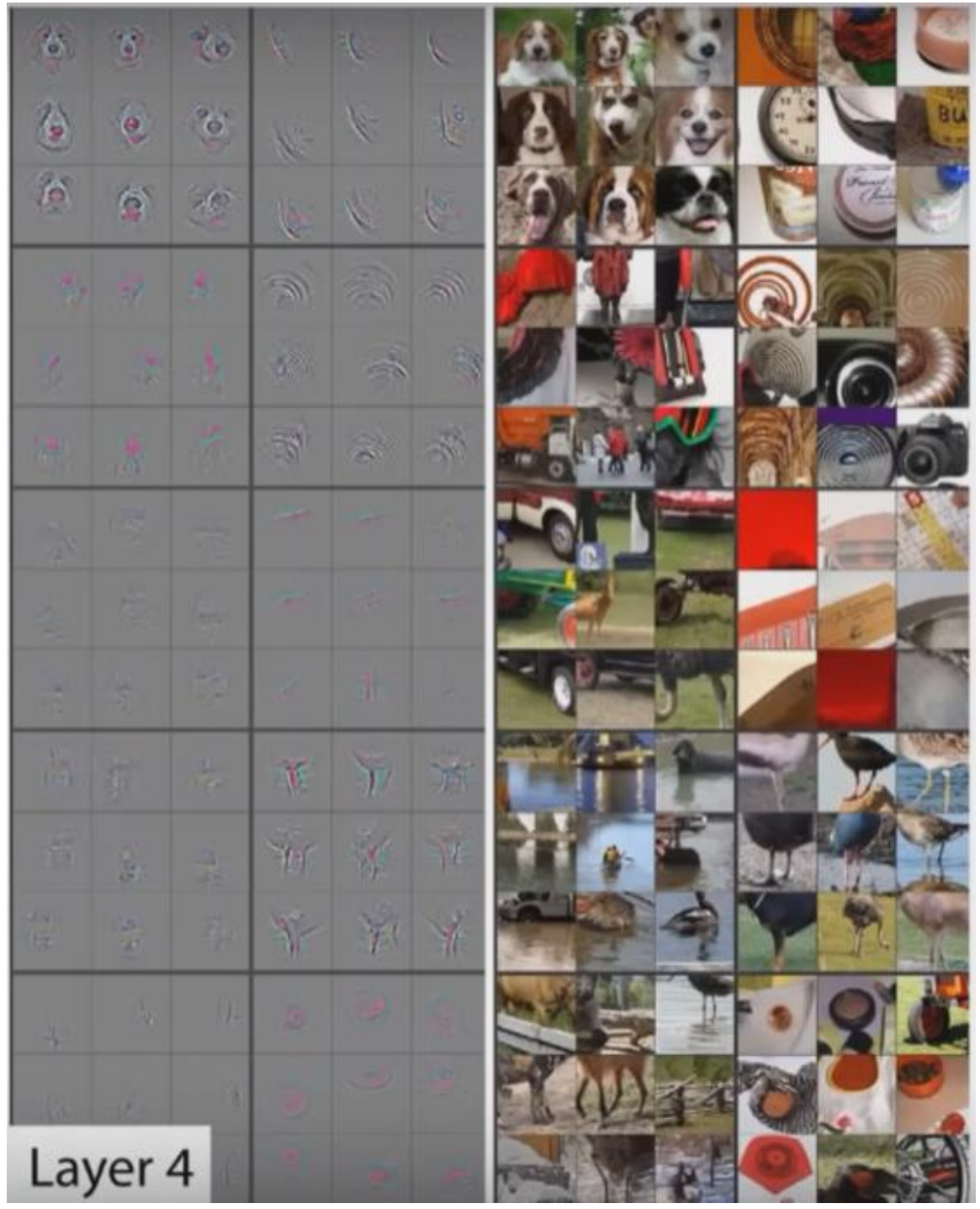
filter 3

0	0	0
1	1	1
-1	-1	-1

filter 4

0	1	-1
0	1	-1
0	1	-1


Layer 2


Layer 4

Couche de convolution 1D

Le filtre est appliqué à l'entrée en effectuant une opération de produit scalaire entre le filtre et une fenêtre glissante de l'entrée. Cette fenêtre glissante parcourt l'entrée avec un certain pas (stride) et multiplie les valeurs correspondantes de l'entrée par les valeurs du filtre.

On considère un vecteur d'entrée de taille 7X1 et un filtre 3X1, le calcul de la convolution 1D est comme suit :

Vecteur d'entrée **Filtre 1D** **Résultat de la convolution**

PARTIE 2

Couche de convolution 2D

On traite maintenant le cas d'une matrice. On considère une matrice d'entrée 5X5 et un filtre 3X3 :

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image d'entrée de taille 5x5

1	0	1
0	1	0
1	0	1

Filtre/kernel de taille 3x3

Pour la première case, on obtient :

1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

4		

Résultat

PARTIE 2

Couche de convolution 2D

1	1x1	1x0	0x1	0
0	1x0	1x1	1x0	0
0	0x1	1x0	1x1	1
0	0	1	1	0
0	1	1	0	0

4	3	

Résultat

1	1	1x1	0x0	0x1
0	1	1x0	1x1	0x0
0	0	1x1	1x0	1x1
0	0	1	1	0
0	1	1	0	0

4	3	4

Résultat

1	1	1	0	0
0x1	1x0	1x1	1	0
0x0	0x1	1x0	1	1
0x1	0x0	1x1	1	0
0	1	1	0	0

4	3	4
2		

Résultat

1	1	1	0	0
0	1x1	1x0	1x1	0
0	0x0	1x1	1x0	1
0	0x1	1x0	1x1	0
0	1	1	0	0

4	3	4
2	4	

Résultat

PARTIE 2

Couche de convolution 2D

1	1	1	0	0
0	1	1x1	1x0	0x1
0	0	1x0	1x1	1x0
0	0	1x1	1x0	0x1
0	1	1	0	0

4	3	4
2	4	3

Résultat

1	1	1	0	0
0	1	1	1	0
0x1	0x0	1x1	1	1
0x0	0x1	1x0	1	0
0x1	1x0	1x1	0	0

4	3	4
2	4	3
2		

Résultat

1	1	1	0	0
0	1	1	1	0
0	0x1	1x0	1x1	1
0	0x0	1x1	1x0	0
0	1x1	1x0	0x1	0

4	3	4
2	4	3
2	3	

Résultat

1	1	1	0	0
0	1	1	1	0
0	0	1x1	1x0	1x1
0	0	1x0	1x1	0x0
0	1	1x1	0x0	0x1

4	3	4
2	4	3
2	3	4

Résultat

PARTIE 2

Couche de convolution 2D

Lorsqu'il s'agit de traiter des images RGB (Red-Green-Blue), la convolution est effectuée sur chaque canal de couleur (R, G, B) séparément, et les résultats sont ensuite combinés pour former la carte de caractéristiques finale. Le filtre utilisé pour la convolution est également une matrice bidimensionnelle séparée pour chaque canal de couleur. Par conséquent, il y aura un filtre distinct pour le canal R, un pour le canal G et un pour le canal B. Chaque filtre est appris par le réseau lors de l'entraînement.

11	23	14	0	0
0	2	15	10	0
0	0	3	11	1
0	0	1	1	0
0	11	1	0	0

1	0	1
0	1	0
1	0	1

Filtre du canal **R**

0	0	12	0	0
0	20	18	11	0
0	0	6	4	1
0	0	3	1	0
0	90	0	0	0

-1	1	1
0	0	0
1	0	1

Filtre du canal **V**

0	12	98	0	0
0	0	0	56	0
3	0	10	11	16
0	0	0	1	0
0	90	12	0	0

1	0	1
-1	0	0
1	0	1

Filtre du canal **B**

159		

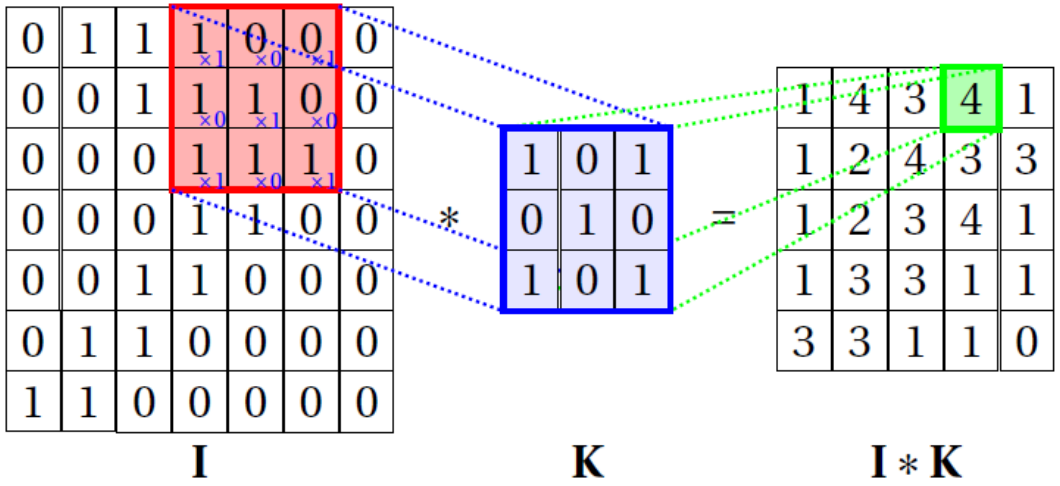
30 + **18** + **111** = 159

PARTIE 2

Couche de convolution 2D

Soit I une image de taille $N \times N$ et Kun filtre de taille $F \times F$, la taille de la sortie O est calculée par :

$$O = I * K \rightarrow (N - F + 1) \times (N - F + 1)$$

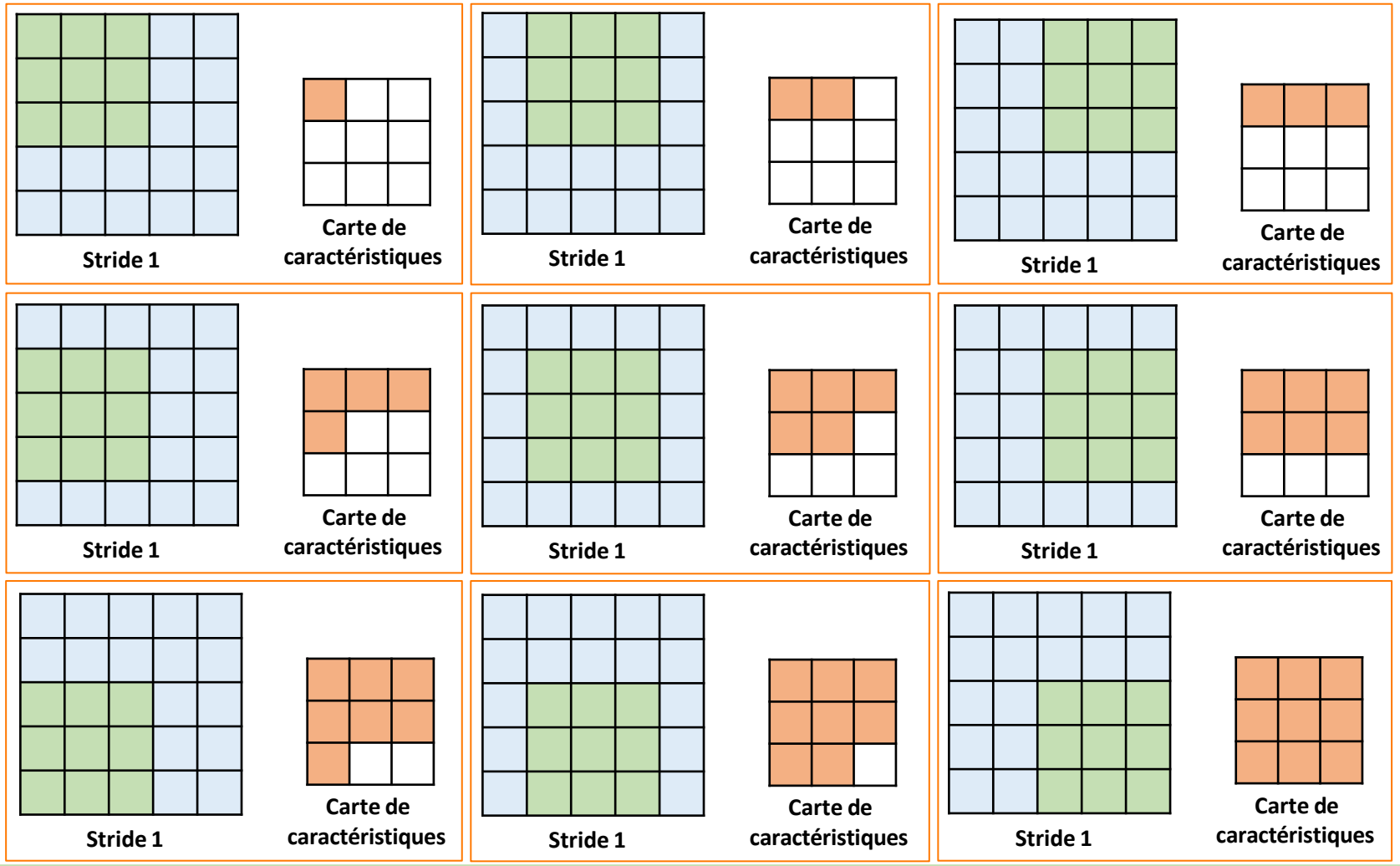


Exemple :

$$N = 7 \times 7$$
$$K = 3 \times 3$$
$$O = (7 - 3 + 1) \times (7 - 3 + 1) = 5 \times 5$$

Couche de convolution 2D : Pas de convolution

Le pas de la convolution ou « stride » en anglais est un paramètre qui contrôle le décalage du filtre lors de la convolution. Il définit le nombre de pixels (ou de pas) de déplacement horizontal et vertical entre chaque application du filtre. Un stride de 1 signifie que le filtre se déplace d'un pixel à la fois, tandis qu'un stride de 2 signifie que le filtre se déplace de deux pixels à la fois.



PARTIE 2

Couche de convolution 2D : Pas de convolution

Soit I une image de taille $N \times N$ et K un filtre de taille $F \times F$ et S le pas de la convolution. La taille de la sortie O est calculée par :

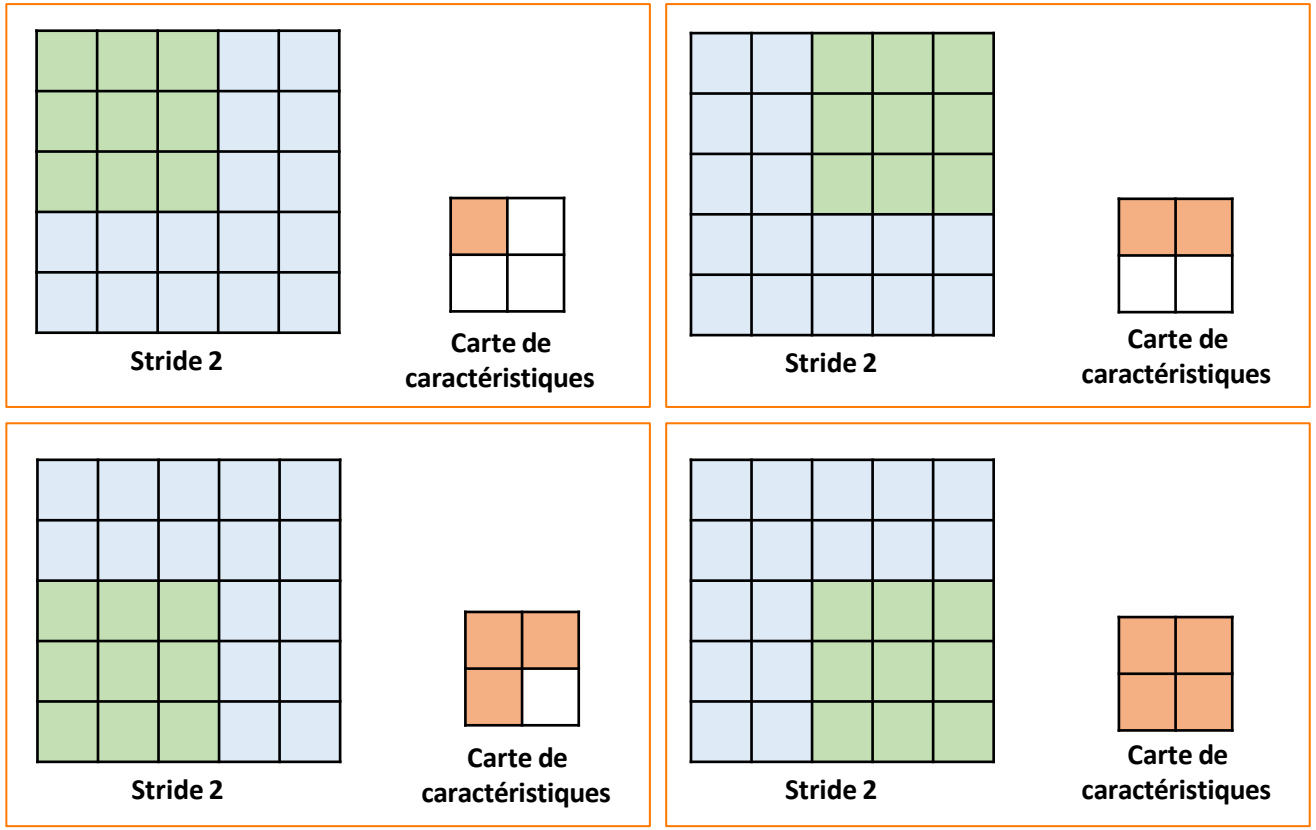
Taille de la carte

$$O = \frac{(N - F)}{S} + 1$$

Exemple :

$$\mathrm{N} = 5 \times 5$$
$$\mathrm{K} = 3 \times 3$$
$$\mathrm{S} = 2$$

$$\mathrm{O} = \frac{(5 - 3)}{2} + 1$$



PARTIE 2

Couche de convolution 1D : Pas de convolution

On considère un vecteur d'entrée 7X1, et un filtre 3X1:

PARTIE 2

Couche de convolution 2D : Remplissage

Le remplissage (ou bien padding en anglais) est l'ajout de zéros autour des bords de l'entrée avant d'appliquer la convolution. Il permet de conserver les dimensions spatiales de l'entrée après la convolution, en particulier lorsque le filtre est plus grand que 1x1.

Au début, on calcule le nombre de zéros P qu'on peut ajouter dans le remplissage :

$$P = \frac{F-1}{2}$$

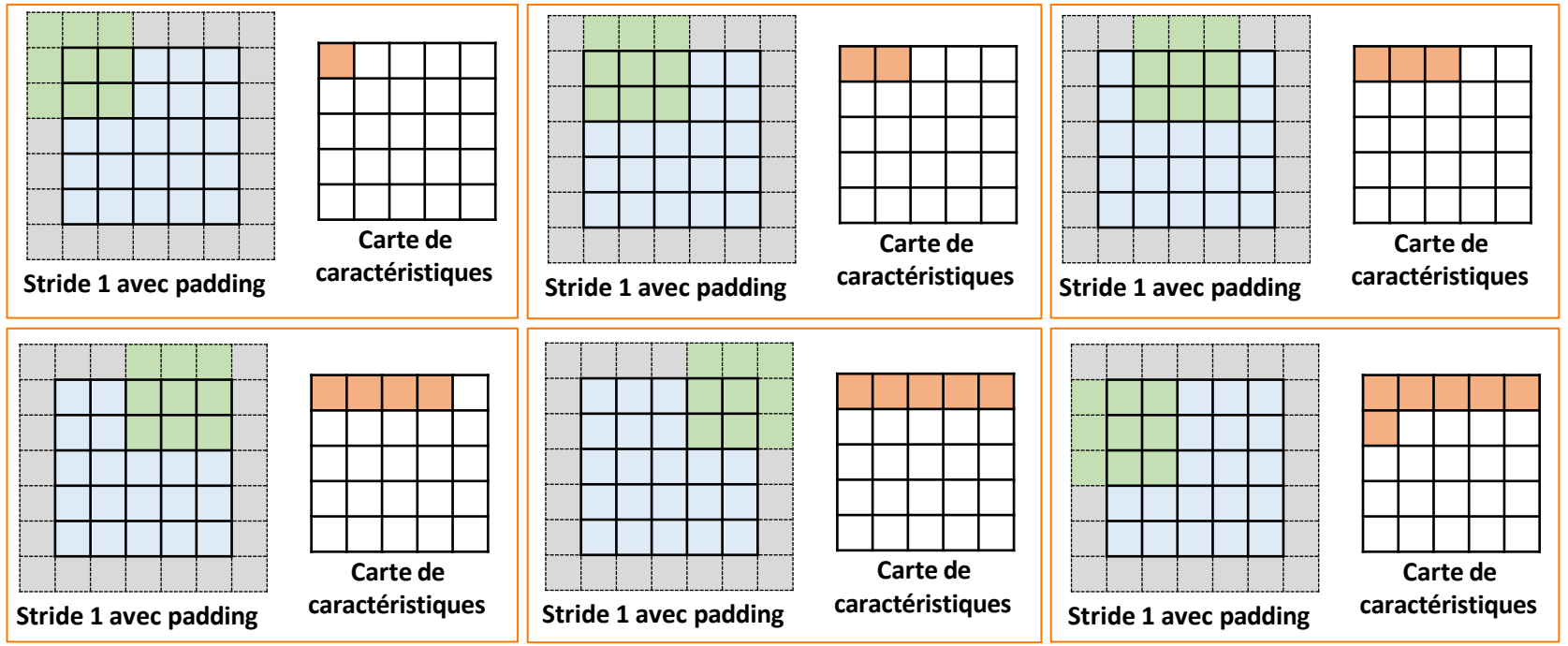
Taille de la carte

$$O = \frac{(N - F + 2P)}{S} + 1$$

Exemple :

$$P = \frac{3-1}{2} = 1$$

$$O = \frac{(5-3+2)}{1} + 1 = 5$$



01 - CONSTRUIRE DES RÉSEAUX À CONVOLUTION

Architecture des réseaux à convolution

```
from tensorflow.keras.layers import Conv2D
Conv2D(filters=32,              # nombre de filtres (noyaux)
       kernel_size=(3,3),       # taille de chaque filtre
       strides=(1,1),           # déplacement du filtre (sx,sy)
       padding='same',          # 'same' (même taille que l'entré), 'valid'#pas de padding(réduction )
       activation='relu',       # fonction d'activation
       input_shape=(28,28,1) # taille de l'entrée (uniquement dans la 1re couche)
)
```

PARTIE 2

Couche de convolution 1D : Remplissage

On considère un vecteur d'entrée de taille 7 et un filtre de taille 3X1 :

Entrée de taille 7 stride 1

Filtre/kernel de taille 3

PARTIE 2

Couche de convolution 2D : Profondeur

La profondeur ou « depth » en anglais dans le contexte d'une couche de convolution se réfère au nombre de filtres (ou noyaux) utilisés dans cette couche. Chaque filtre extrait des caractéristiques spécifiques de l'entrée, et le nombre total de filtres détermine la profondeur de la couche de convolution.

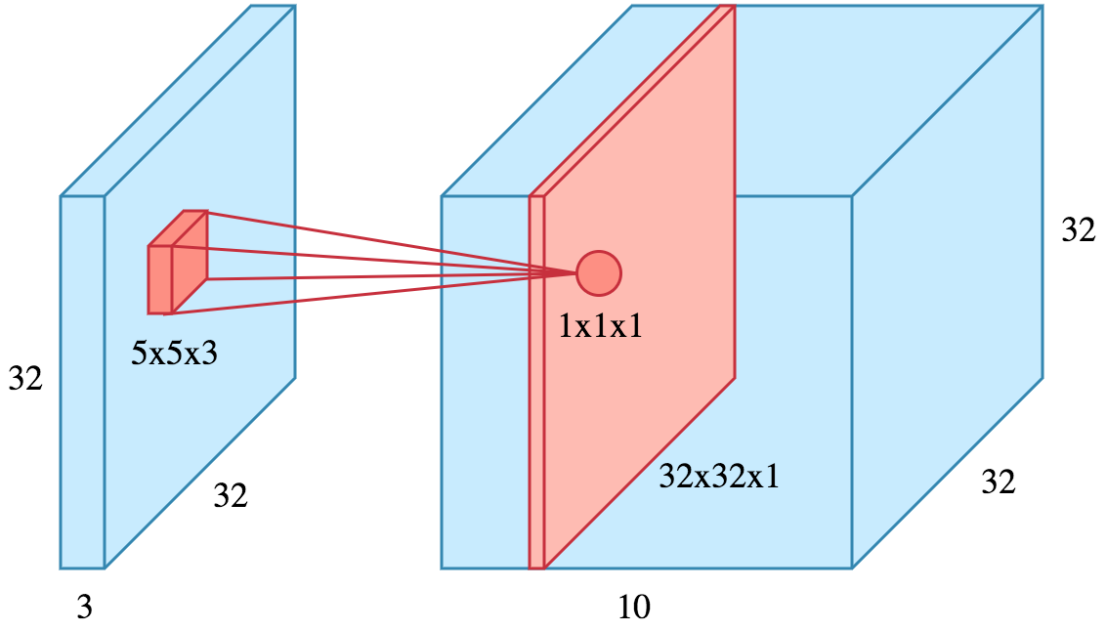
La profondeur d'une couche de convolution correspond au nombre de cartes de caractéristiques en sortie. Chaque carte de caractéristiques représente la réponse d'un filtre spécifique à travers l'ensemble de l'entrée. Par exemple, si une couche de convolution a une profondeur de 10, cela signifie qu'elle utilise 10 filtres et produit 10 cartes de caractéristiques en sortie.

Pour les images RGB (3 canaux)

Les filtres sont de tailles 5x5x**3**

Exemple :

Élément	Description
I = 32×32×3	Image d'entrée : 32×32 pixels, 3 canaux (RGB)
F = 5×5×3	Chaque filtre a la même profondeur que l'entrée
1 filtre = 1 carte en sortie	Taille 32×32×1 (si padding='same')
K = 10 filtres	Donc on aura 10 cartes en sortie → **32×32×10**

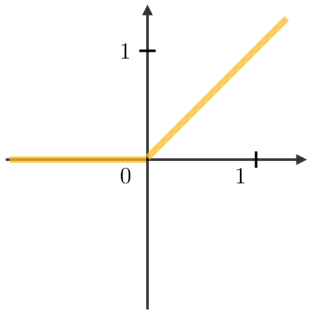


PARTIE 2

Fonction d'activation

La fonction d'activation ReLU (Rectified Linear Unit en anglais) est une fonction non linéaire largement utilisée dans les réseaux de neurones, y compris les réseaux de neurones à convolution (CNN). Elle est appliquée élément par élément à la sortie d'une couche de convolution (ou à toute autre couche) pour introduire de la non-linéarité dans le modèle. La fonction ReLU est définie comme suit :

$f(x) = \max(0, x)$



15	20	-10	35
18	-110	25	100
20	-15	25	-10
101	75	18	23

ReLU

15	20	0	35
18	0	25	100
20	0	25	0
101	75	18	23

Elle prend une valeur d'entrée x et renvoie x si x est positif ou zéro, et renvoie 0 si x est négatif. En d'autres termes, la fonction ReLU supprime les valeurs négatives et laisse les valeurs positives inchangées.

PARTIE 2

Fonction d'activation

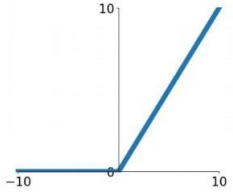
La fonction ReLU présente plusieurs avantages :

1. Non-linéarité : La fonction ReLU est non linéaire, ce qui lui permet de modéliser des relations non linéaires entre les caractéristiques extraites par le réseau. Cela permet au modèle de capturer des motifs et des interactions complexes dans les données.
2. Sparsité : La fonction ReLU a une propriété intéressante appelée "sparsité". Elle met à zéro les valeurs négatives, ce qui signifie que seules les activations positives sont transmises aux couches suivantes. Cela peut favoriser la sélection de caractéristiques importantes et contribuer à une meilleure représentation des données.
3. Calculs efficaces : La fonction ReLU est simple et rapide à calculer par rapport à d'autres fonctions d'activation plus complexes telles que la fonction sigmoïde ou la tangente hyperbolique.

Cependant, il convient de noter que la fonction ReLU présente également quelques limitations, notamment la disparition du gradient pour les valeurs négatives. Lorsque le gradient devient nul, les neurones associés ne reçoivent plus de mises à jour lors de la rétropropagation du gradient, ce qui peut ralentir ou arrêter l'apprentissage du réseau. Pour atténuer ce problème, des variantes de ReLU, comme la Leaky ReLU ou la ELU, ont été proposées.

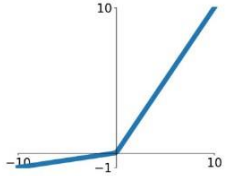
ReLU

$$\max(0, x)$$



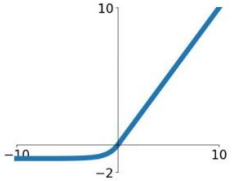
Leaky ReLU

$$\max(0.1x, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



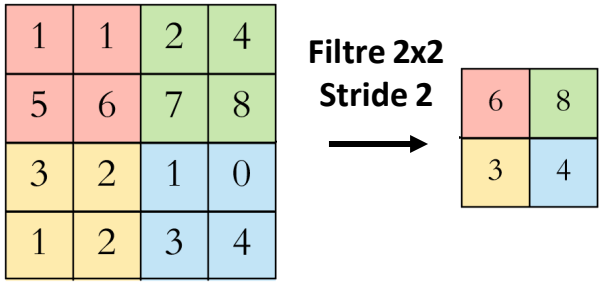
PARTIE 2

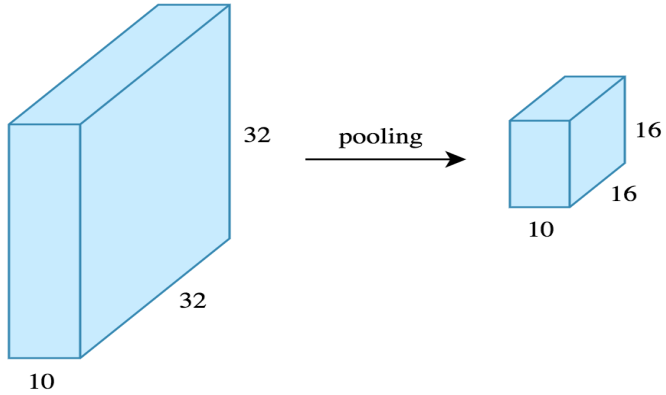
Couche de pooling

Ce type de couche réduit le nombre de paramètres lorsque les images sont trop grandes. Elle est appelée également couche de sous-échantillonnage puisqu'elle réduit la dimensionnalité de chaque carte de caractéristiques en conservant les informations importantes.

Le pooling spatial peut être de différents types:

- ✓ Max Pooling
- ✓ Average pooling
- ✓ Sum pooling





Taille de la sortie

$$Ox = \frac{(Nx - Fx)}{Sx} + 1 \qquad Oy = \frac{(Ny - Fy)}{Sy} + 1$$

Exemple :

$$O = \frac{(4-2)}{2} + 1 = 2$$

La taille de la sortie est : 2x2

PARTIE 2

```
from tensorflow.keras.layers import MaxPooling2D

MaxPooling2D(pool_size=(2,2),    # taille de la fenêtre
             strides=(2,2),      # déplacement (souvent égal à pool_size)
             padding='valid')    # 'valid' (réduction), 'same' (taille conservée)
```

```
from tensorflow.keras.layers import AveragePooling2D

AveragePooling2D(pool_size=(2,2),
                 strides=(2,2),
                 padding='valid')
```

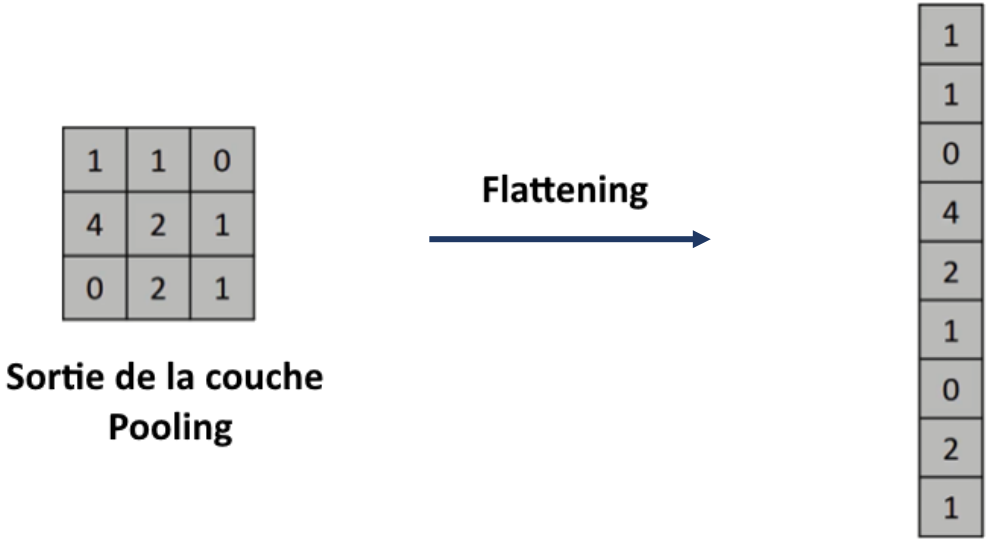
```
from tensorflow.keras.layers import MaxPooling2D

MaxPooling2D(pool_size=(2,2),    # taille de la fenêtre
             strides=(2,2),      # déplacement (souvent égal à pool_size)
             padding='valid')    # 'valid' (réduction), 'same' (taille conservée)
```

PARTIE 2

Couche « Flatten »

La couche « Flatten » est une couche utilisée dans les réseaux de neurones pour convertir une entrée multidimensionnelle en un vecteur unidimensionnel, souvent utilisé comme entrée pour les couches entièrement connectées. Lorsqu'une couche Flatten est ajoutée après les couches de convolution et de pooling dans un réseau de neurones à convolution (CNN), elle transforme la sortie de ces couches en un vecteur 1D. Cela permet de passer des informations extraites localement par les couches précédentes à une représentation linéaire globale des caractéristiques.

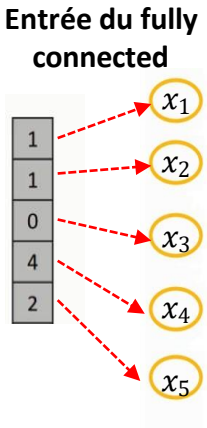
Une fois les caractéristiques aplaties, ce vecteur peut être utilisé comme entrée pour les couches entièrement connectées, qui sont des couches traditionnelles de réseaux de neurones où chaque neurone est connecté à tous les neurones de la couche précédente. Les couches entièrement connectées sont souvent utilisées pour effectuer des tâches de classification ou de prédiction finales.

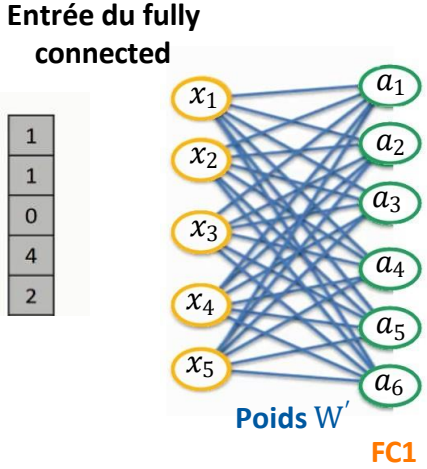


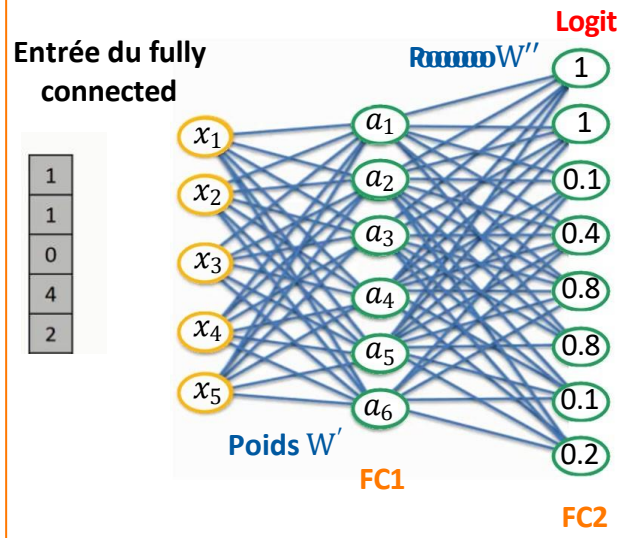
PARTIE 2

Couche entièrement connectée

La couche entièrement connectée (FC- Fully Connected en anglais), également appelée couche dense, est une couche de réseau de neurones où chaque neurone est connecté à tous les neurones de la couche précédente. Dans cette couche, chaque neurone effectue une combinaison linéaire des entrées provenant de la couche précédente, suivi d'une fonction d'activation non linéaire.



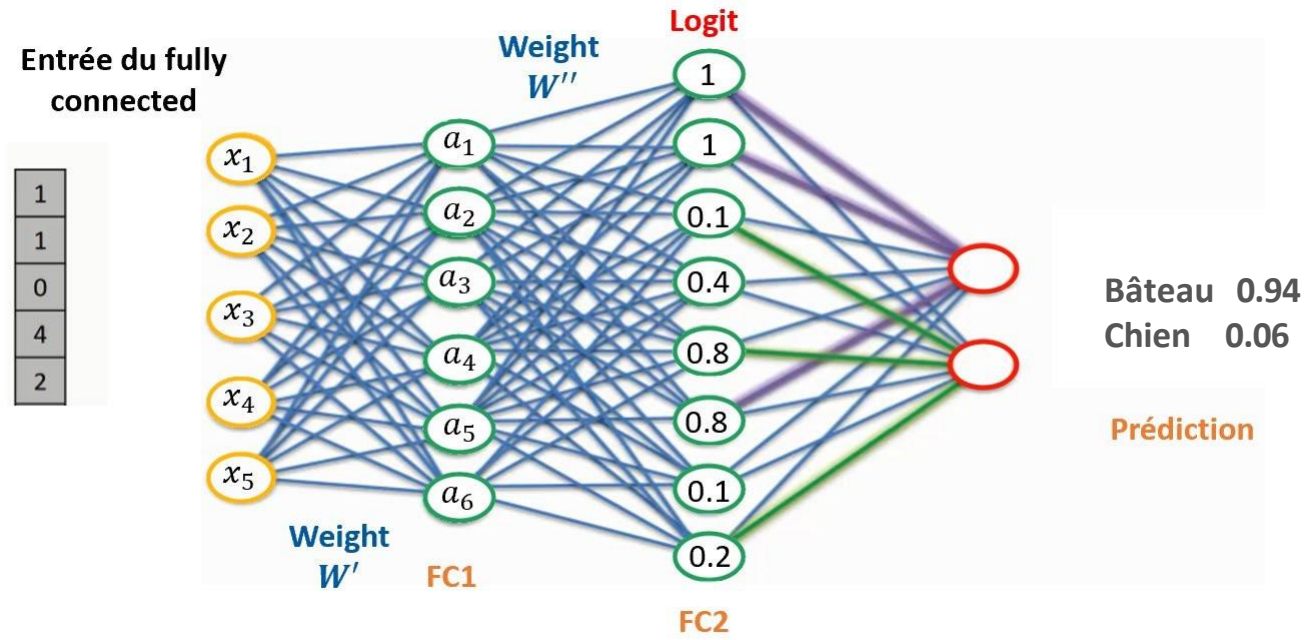




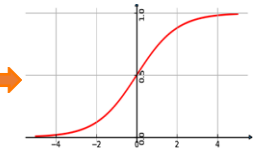
PARTIE 2

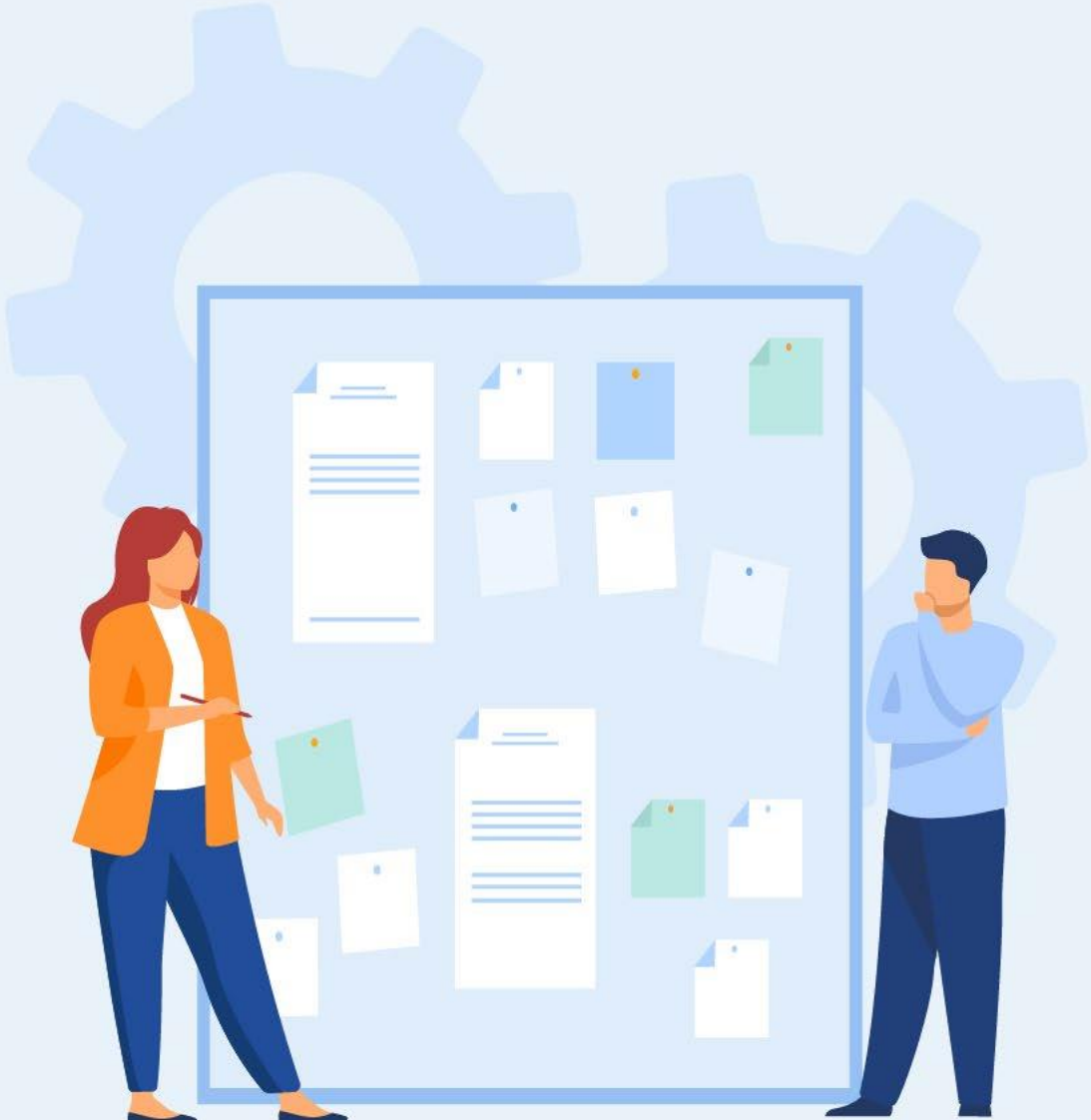
Couche entièrement connectée

Après la combinaison linéaire, une fonction d'activation non linéaire (exemple: fonction softmax) est appliquée à la sortie de chaque neurone. Cela introduit de la non-linéarité dans le modèle et permet de capturer des relations complexes entre les caractéristiques. La sortie de la couche entièrement connectée est généralement transmise à une autre couche entièrement connectée ou à la couche de sortie du modèle. Cela peut être une sortie de classification, de régression ou d'autres tâches spécifiques. Les couches entièrement connectées sont souvent utilisées dans les parties finales des réseaux de neurones, où la tâche principale est la classification, la prédiction ou la génération. Elles permettent de combiner et de transformer les caractéristiques extraites des couches précédentes en une représentation finale qui peut être utilisée pour la tâche spécifique.



$$Softmax(z)_i = \frac{e^{z_i}}{\sum_{l=1}^{k} e^{z_l}}$$





CHAPITRE 1

CONSTRUIRE DES RÉSEAUX À CONVOLUTION

1. Définition
2. Architecture des réseaux à convolution
3. **Apprentissage des réseaux à convolution**
4. Quelques exemples des réseaux à convolution

PARTIE 2

Entropie croisée

L'entropie croisée, également connue sous le nom de perte de log-vraisemblance, est une mesure couramment utilisée pour quantifier la différence entre deux distributions de probabilités. Dans le contexte de l'apprentissage automatique, l'entropie croisée est souvent utilisée comme fonction de perte pour entraîner des modèles de classification. Lorsqu'elle est utilisée comme fonction de perte, l'entropie croisée mesure la divergence entre la distribution de probabilités prédite par le modèle et la distribution de probabilités réelle (étiquettes réelles) des données d'entraînement. L'objectif est de minimiser cette divergence pour améliorer la capacité du modèle à prédire correctement les classes.

Exemple :

Animal	Label
Chien	[1, 0, 0, 0, 0]
Renard	[0, 1, 0, 0, 0]
Cheval	[0, 0, 1, 0, 0]
Aigle	[0, 0, 0, 1, 0]
Écureuil	[0, 0, 0, 1]

Lorsque nous n'avons pas suffisamment entrainé le modèle, il peut classer la première image (chien) comme suit:

$$Q_1 = [0.4 \quad 0.3 \quad 0.05 \quad 0.05 \quad 0.2]$$

$$P_1 = [1 \quad 0 \quad 0 \quad 0 \quad 0]$$

$$H(P_1, Q_1) = -\sum_i P_1(i) \log Q_1(i)$$

$$= -(1 \log 0.4 + 0 \log 0.3 + 0 \log 0.05 + 0 \log 0.05 + 0 \log 0.2)$$

$$= -\log 0.4$$

$$\approx 0.916$$

?

Avec:

- Q_1: La distribution de probabilité
- P_1: Vrai label
- H: validation croisée

PARTIE 2

Entropie croisée

Le modèle indique que la première image est de 40% pour un chien, 30% pour un renard, 5% pour un cheval, 5% pour un aigle et 20% pour un écureuil. Cette estimation n'est pas très précise ou confiante.

En revanche, le label nous donne la distribution exacte de la classe animale de la première image.

Une fois le modèle bien entrainé, il peut produire la prédiction suivante pour la première image.

$$Q_1 = [0.98 \quad 0.01 \quad 0 \quad 0 \quad 0.01]$$

$$P_1 = [1 \quad 0 \quad 0 \quad 0 \quad 0]$$

$$\begin{aligned} H(P_1, Q_1) &= -\sum_i P_1(i) \log Q_1(i) \\ &= -(1 \log 0.98 + 0 \log 0.01 + 0 \log 0 + 0 \log 0 + 0 \log 0.01) \\ &= -\log 0.98 \\ &\approx 0.02 \end{aligned}$$

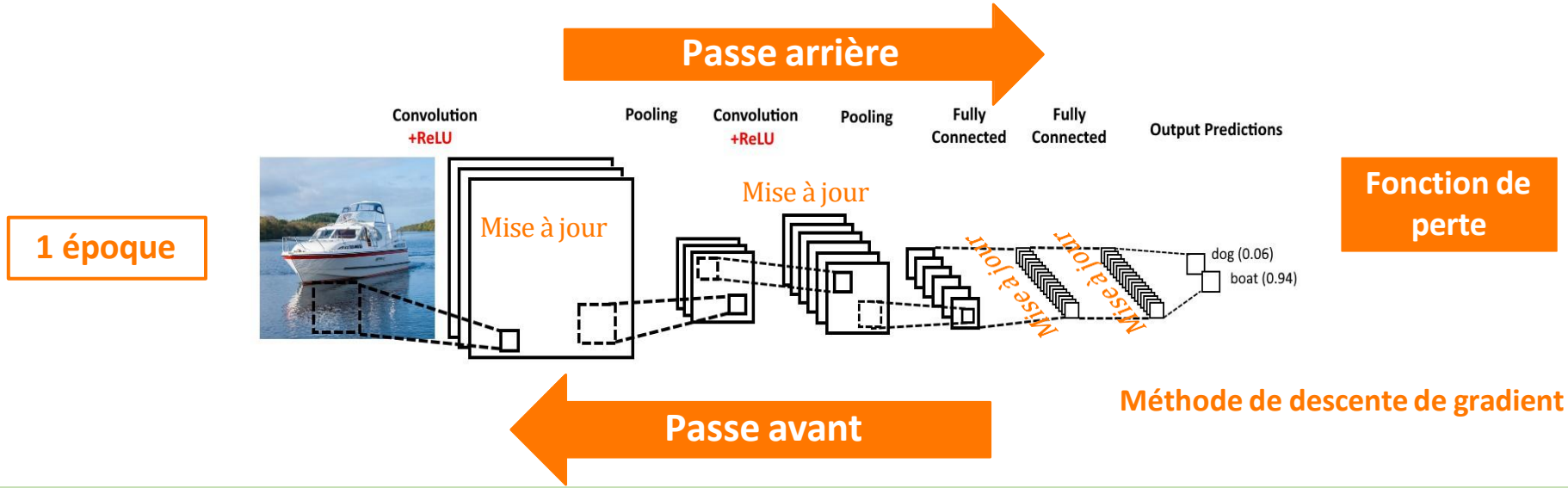
L'entropie croisée diminue et la prédiction devient de plus en plus précise. elle devient nul si la prédiction est parfaite.

PARTIE 2

Dans les réseaux de neurones, les époques (epochs en anglais) font référence au nombre de fois où l'ensemble complet des données d'entraînement est passé à travers le réseau pendant le processus d'apprentissage. Chaque époque consiste en une passe avant et en arrière (forward pass et backward pass) à travers le réseau pour ajuster les poids et les biais en fonction de l'erreur de prédiction.

Dans un premier temps, le CNN commence sa passe avant. Dans cette étape, les filtres ainsi que les poids des couches entièrement connectées sont initialisés aléatoirement. L'entraînement d'un CNN consiste alors à optimiser les coefficients du réseau pour minimiser l'erreur de classification en sortie (fonction de perte).

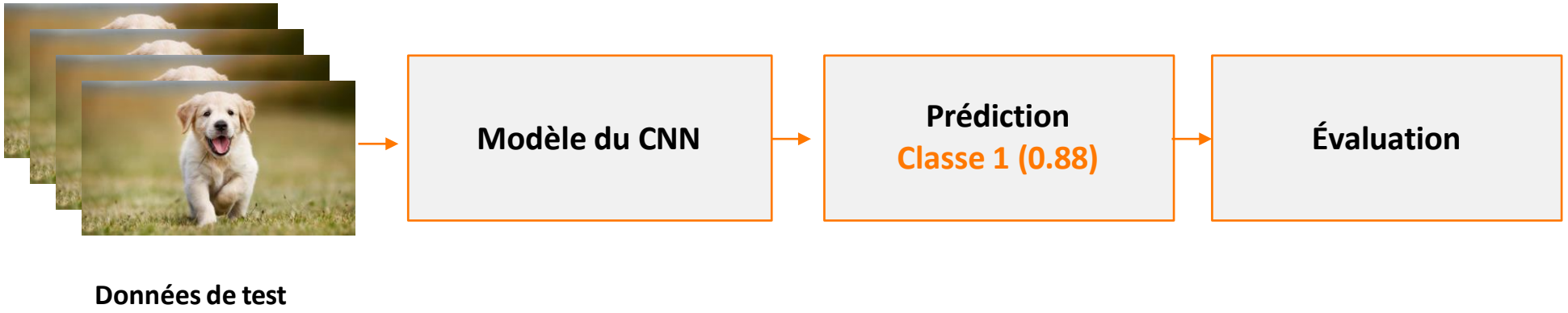
En pratique, les coefficients du réseau sont modifiés de façon à corriger les erreurs de classification rencontrées, selon une méthode de descente de gradient. Ces gradients sont rétro-propagés dans le réseau depuis la couche de sortie.

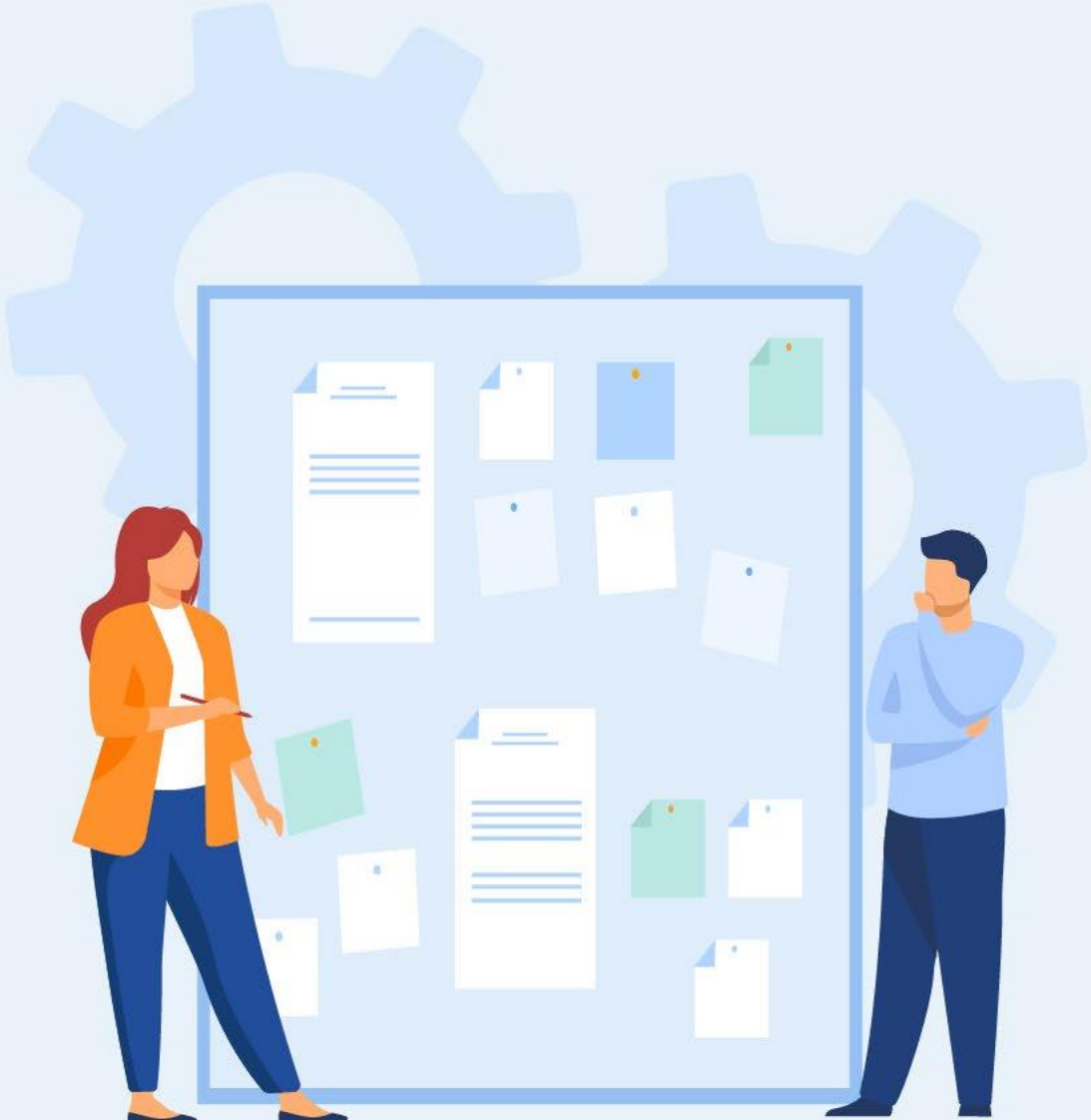


PARTIE 2

Phase de test (prédiction)

Dans un réseau de neurones, la phase de test fait référence à l'étape où le modèle entraîné est évalué sur un ensemble de données de test indépendantes. Pendant cette phase, le réseau de neurones est utilisé pour effectuer des prédictions sur des exemples de données qu'il n'a pas encore rencontrés. La phase de test dans un réseau de neurones consiste à évaluer les performances du modèle entraîné sur un ensemble de données de test indépendantes. Cela permet d'évaluer la capacité de généralisation du modèle et de détecter tout surapprentissage éventuel. Comme tous les algorithmes d'apprentissage, le réseau de neurones à convolution est déterminé dans la phase d'apprentissage. Après cette phase, le modèle est sauvegardé pour être utilisé dans la phase de test.





CHAPITRE 1

CONSTRUIRE DES RÉSEAUX À CONVOLUTION

1. Définition
2. Architecture des réseaux à convolution
3. Apprentissage des réseaux à convolution
4. **Quelques exemples des réseaux à convolution**

PARTIE 2

LeNet (1990)

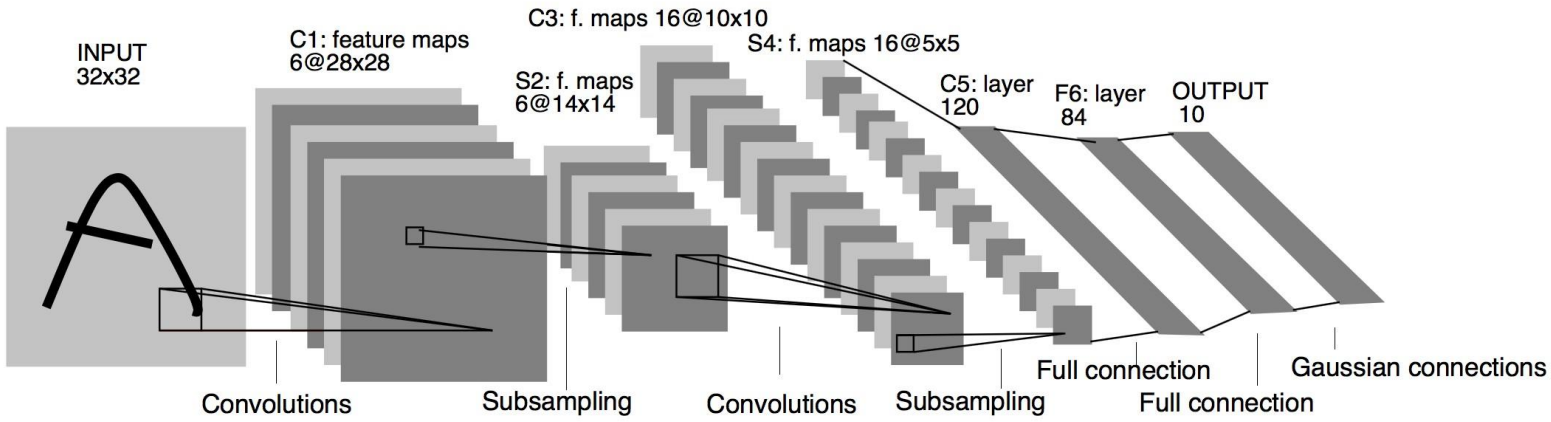
LeNet, également connu sous le nom de LeNet-5, est largement connu et considéré comme l'un des premiers modèles CNN à avoir eu un impact significatif dans le domaine de la vision par ordinateur. Il a été développé par Yann LeCun, Léon Bottou, Yoshua Bengio et Patrick Haffner dans les années 1990. LeNet est souvent utilisé pour des tâches de classification, en particulier pour la reconnaissance de chiffres manuscrits. La dernière couche du réseau est généralement une couche entièrement connectée avec des neurones correspondant aux classes de sortie, suivie d'une fonction d'activation softmax pour obtenir des probabilités de classe.

Pooling : moyenne

Non-linéarité : sigmoïde ou tanh

Couches entièrement connectées à la fin

Entrainé sur la base de données MNIST avec des exemples d'entraînement de 60000



Challenge ImageNet

Le challenge ImageNet est une compétition annuelle de reconnaissance d'images qui se concentre sur la classification des images en plusieurs catégories. Elle a été initiée en 2010 et est devenue l'une des compétitions les plus prestigieuses dans le domaine de la vision par ordinateur.

- ✓ ~ 14 millions d'images étiquetées, 20 000 classes
- ✓ Images récoltées de Internet
- ✓ Étiquettes humaines via Amazon Mturk
- ✓ Défi de reconnaissance visuelle à grande échelle ImageNet (ILSVRC): 1,2 million d'images d'entraînement, 1000 classes

Lien de téléchargement : www.image-net.org/challenges/LSVRC/

PARTIE 2

AlexNet (2012)

AlexNet est une architecture développée par Alex Krizhevsky, Ilya Sutskever et Geoffrey Hinton. Elle a remporté le challenge ImageNet en 2012 et a marqué une avancée majeure dans le domaine de la vision par ordinateur en utilisant des réseaux de neurones profonds.

Pooling: maximum

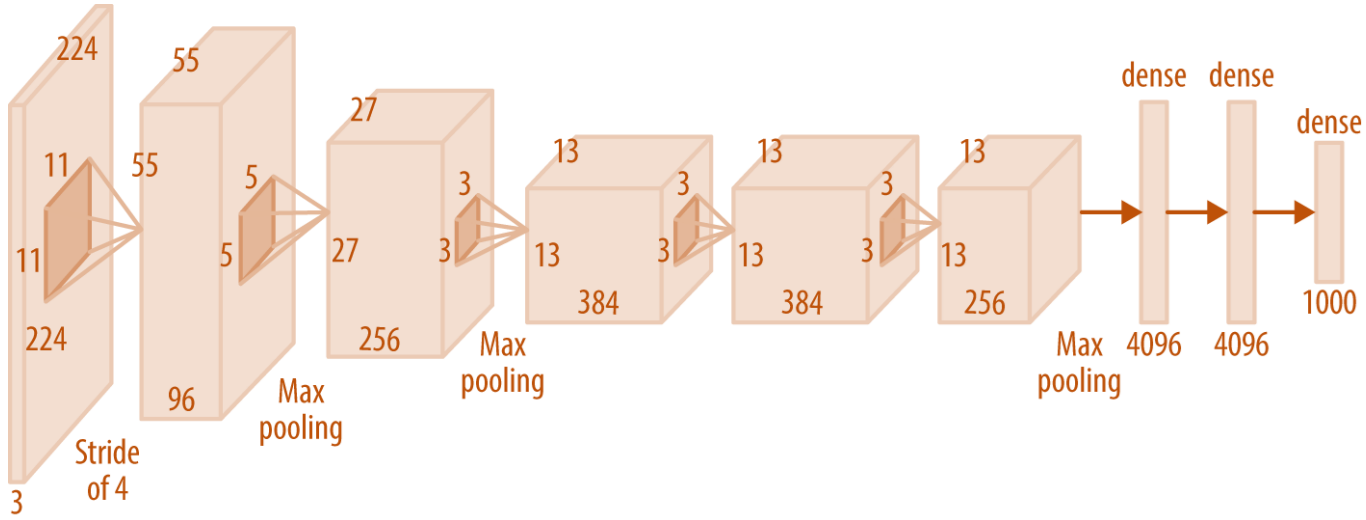
Non-linéarité: ReLU

Plus de données et un modèle plus grand (unités 650K, paramètres 60M)

Implémentation GPU (accélération 50x sur CPU)

Entraîné sur deux GPU pendant une semaine

Régularisation: Dropout



PARTIE 2

VGG (2014)

VGG (Visual Geometry Group) a été développé par l'équipe de recherche Visual Geometry Group de l'Université d'Oxford. VGG a été présenté en 2014 et est devenu un modèle très populaire dans le domaine de la vision par ordinateur.

L'architecture de VGG est connue pour sa simplicité et sa profondeur. Bien qu'elle soit plus coûteuse en termes de ressources computationnelles en raison de son nombre de paramètres plus élevé, elle a montré de très bonnes performances en termes de précision de classification sur diverses tâches de vision par ordinateur. Les principes de conception de VGG ont également influencé le développement d'autres architectures de réseaux de neurones convolutifs.

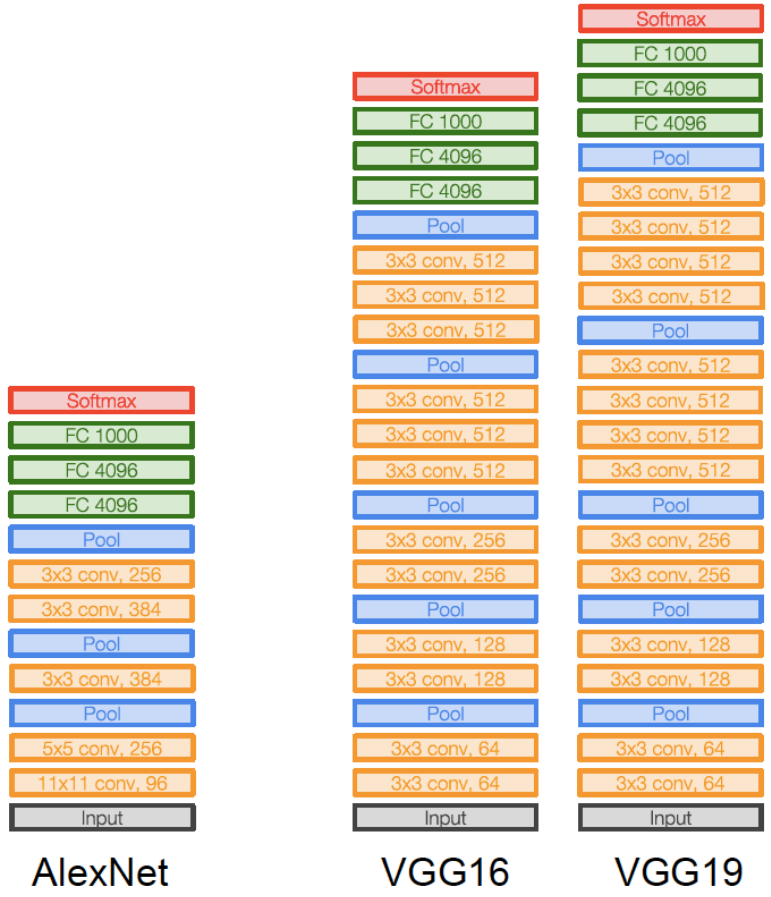
Pooling: maximum

Non-linéarité: ReLU

Architecture uniforme : toutes les couches de convolution sont suivies d'une couche de sous-échantillonnage et les couches entièrement connectées sont situées à la fin.

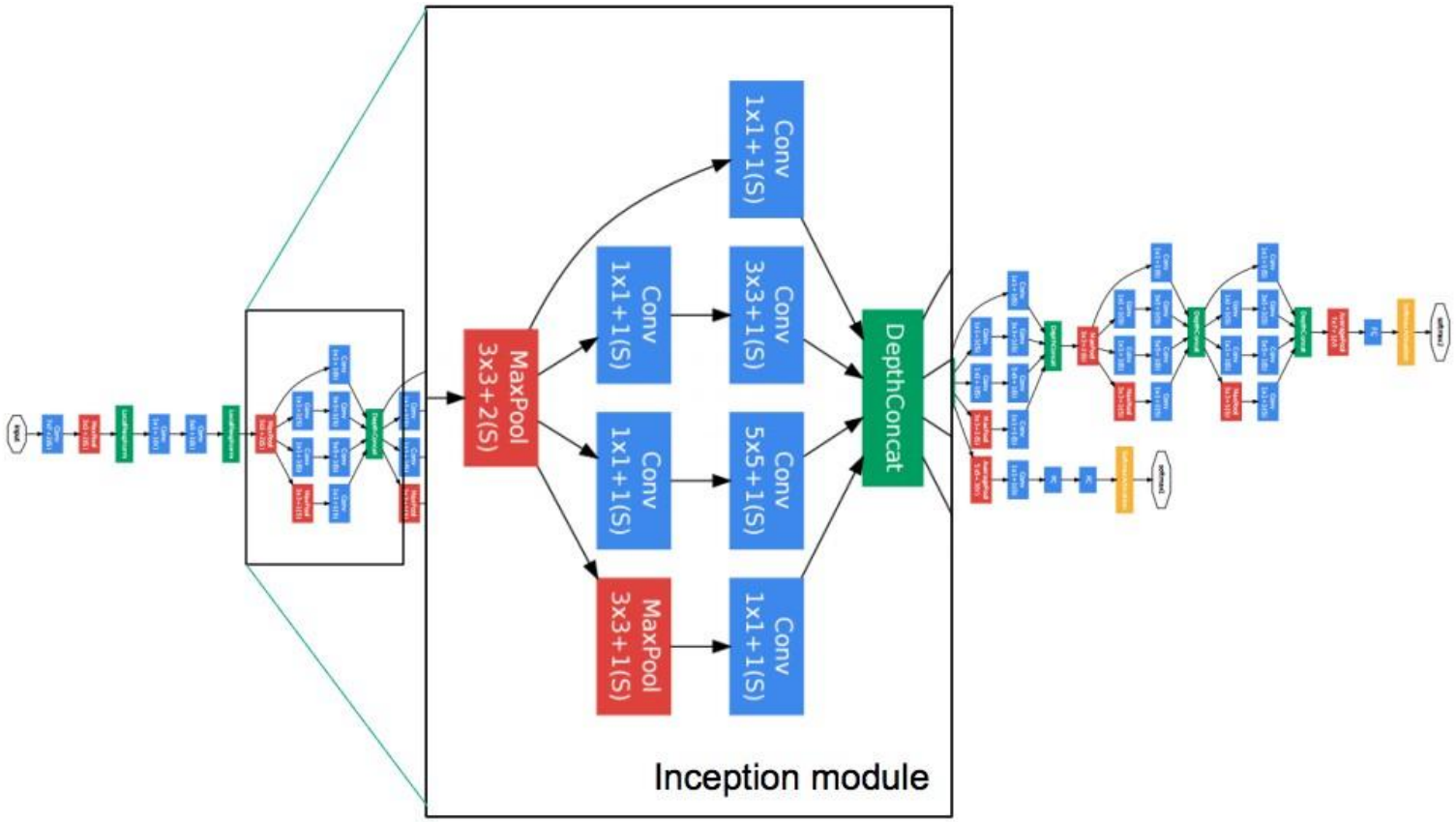
Filtres : de plus petites tailles (2x2 et 3x3).

Nombre de paramètres : 138 millions de paramètres et occupe environ 500 Mo d'espace de stockage



GoogLeNet (2014)

GoogLeNet, également connu sous le nom de Inception v1, est une architecture développée par l'équipe de recherche de Google (Google Brain) en 2014. Il a été conçu pour améliorer à la fois la précision et l'efficacité des modèles de reconnaissance d'images.



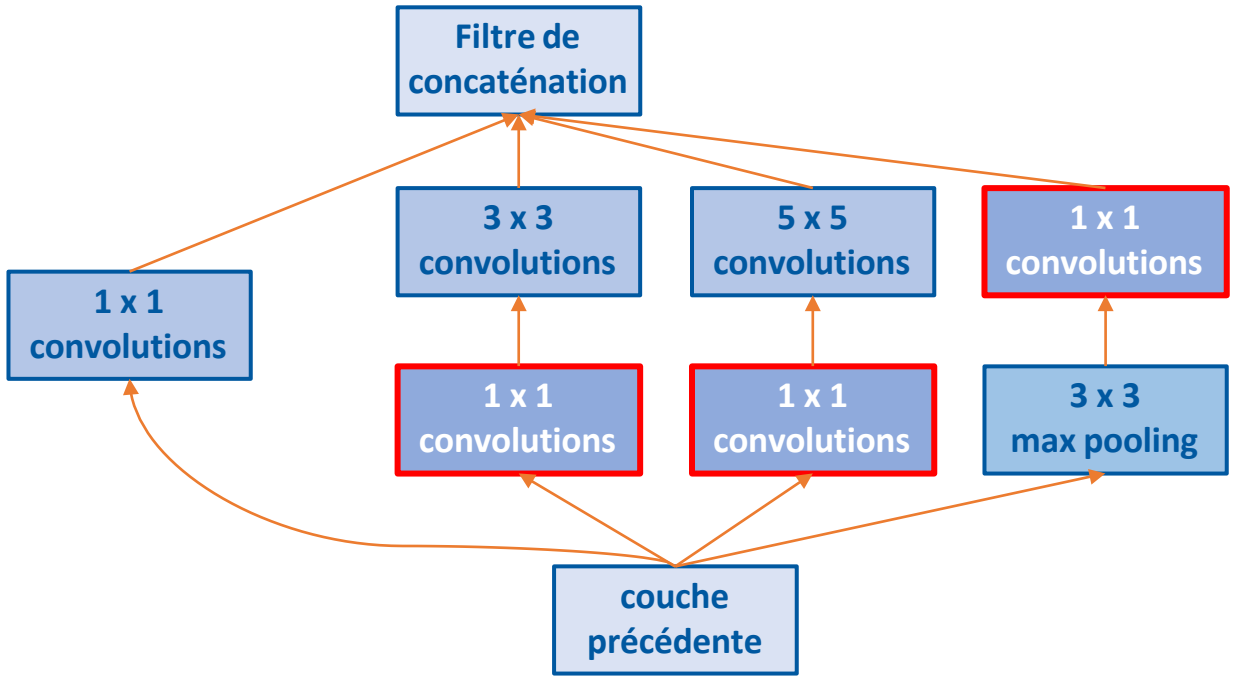
PARTIE 2

GoogLeNet (2014)

Réduction de la dimension spatiale : GoogLeNet utilise des opérations de sous-échantillonnage (pooling) avec un pas de 2 et des convolutions 1x1 pour réduire la dimension spatiale des caractéristiques. Cela permet de réduire le coût computationnel et d'introduire une certaine invariance aux translations et déformations.

Utilisation d'une convolution 1x1 : GoogLeNet utilise des convolutions 1x1 pour réduire la dimensionnalité des cartes de caractéristiques et contrôler le nombre de paramètres. Ces convolutions 1x1 sont utilisées pour effectuer des projections linéaires afin de réduire les coûts computationnels sans sacrifier les performances.

Utilisation de blocs auxiliaires : GoogLeNet utilise des blocs auxiliaires, composés de couches de convolution et de sous-échantillonnage, à des étapes intermédiaires du réseau. Ces blocs auxiliaires aident à la régularisation du modèle et à la propagation du gradient pendant l'entraînement.



PARTIE 2

Inception V2 (2015)

Inception v2 est une amélioration de l'architecture Inception v1 (GoogLeNet) développée par l'équipe de recherche de Google (Google Brain) en 2015. Inception v2 utilise :

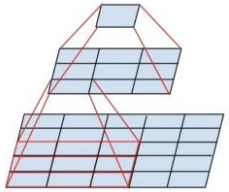
La factorisation des convolutions : Inception v2 utilise des convolutions factorisées pour réduire la complexité computationnelle.

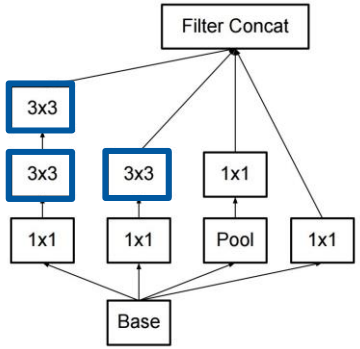
La factorisation de la convolution 3x3 : Inception v2 utilise également une factorisation de la convolution 3x3 en deux convolutions séquentielles de 3x1 et 1x3, ce qui réduit le nombre de paramètres et améliore la représentation des caractéristiques.

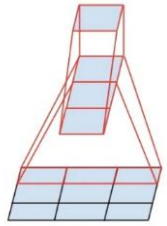
La normalisation par lots (batch normalization) : Inception v2 utilise la normalisation par lots (batch normalization) pour accélérer l'entraînement et stabiliser le processus d'apprentissage.

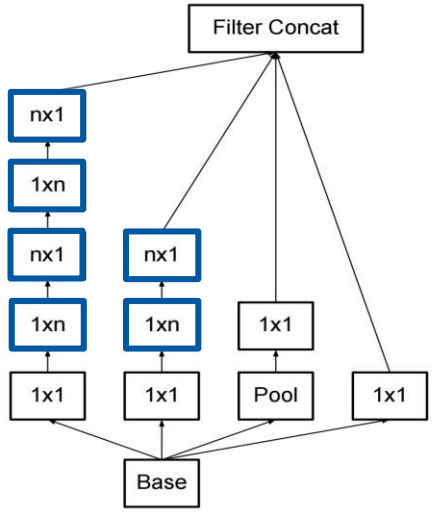
La réduction de dimension : Inception v2 utilise des convolutions 1x1 pour réduire la dimensionnalité des cartes de caractéristiques avant d'appliquer des convolutions plus grandes.

La régularisation améliorée : Inception v2 utilise des techniques de régularisation avancées, telles que la pénalis réduire le surapprentissage.









ResNet (2015)

ResNet, également connu sous le nom de Residual Network a été introduit par Kaiming He, Xiangyu Zhang, Shaoqing Ren et Jian Sun en 2015. ResNet est célèbre pour son innovation dans l'utilisation de blocs résiduels pour faciliter l'apprentissage en profondeur.

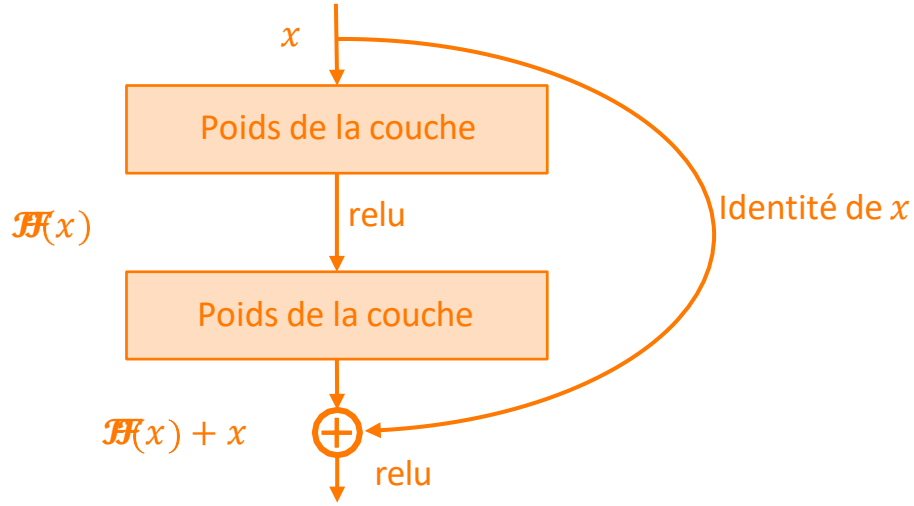
PARTIE 2

ResNet (2015)

ResNet a introduit l'idée de blocs résiduels, qui permettent de surmonter le problème du dégradement de la performance lorsqu'on entraîne des réseaux très profonds. Au lieu d'essayer d'apprendre des caractéristiques directement, les blocs résiduels utilisent des connexions de saut (skip connections) pour transmettre les résidus de l'entrée aux sorties des couches subséquentes. Cela facilite le passage du gradient et facilite l'apprentissage en profondeur.

Les blocs résiduels sont composés de couches de convolution empilées avec des connexions de saut (skip connections) qui ajoutent l'entrée d'origine à la sortie des couches convolutives. Cela permet aux blocs résiduels d'apprendre des représentations résiduelles par rapport à l'entrée d'origine, facilitant ainsi l'apprentissage des détails fins.

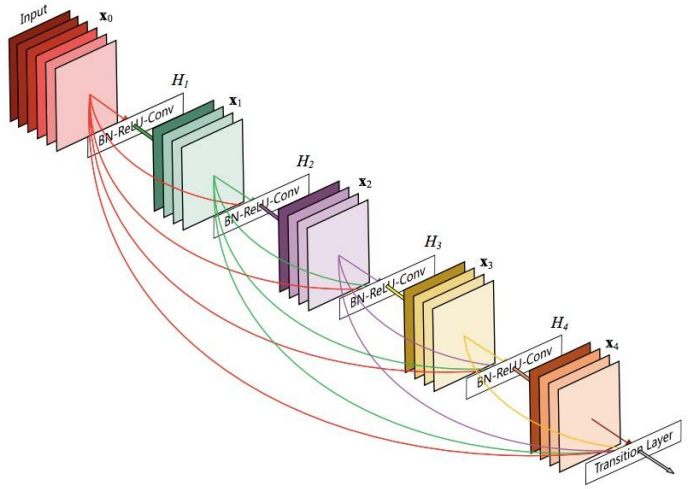
$$\mathcal{H}(x) = \mathcal{F}(x) + x$$



PARTIE 2

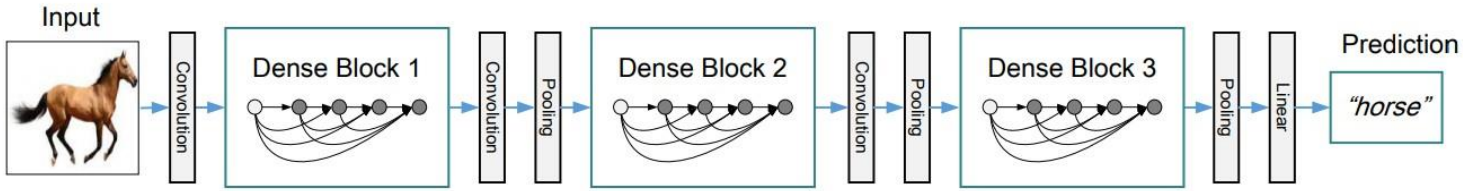
DenseNet (2016)

DenseNet, abréviation de Dense Convolutional Network, est une architecture introduite par Gao Huang, Zhuang Liu, Laurens van der Maaten et Kilian Q. Weinberger en 2016. DenseNet est célèbre pour son approche innovante de connectivité dense, où chaque couche est connectée à toutes les couches précédentes, favorisant ainsi un meilleur échange d'informations entre les couches du réseau.



DenseNet utilise une connectivité dense entre les couches, ce qui signifie que chaque couche est connectée à toutes les couches précédentes. Contrairement aux architectures traditionnelles où les couches sont connectées linéairement, DenseNet exploite des connexions directes entre toutes les couches, ce qui permet un flux d'informations complet et dense à travers le réseau.

DenseNet est construit à partir de blocs de densité, où chaque bloc est composé de plusieurs couches de convolution empilées. Les caractéristiques de chaque couche sont concaténées et transmises en tant qu'entrée à toutes les couches subséquentes. Cela permet aux couches ultérieures d'accéder à toutes les informations extraites par les couches précédentes, favorisant ainsi une meilleure réutilisation des caractéristiques et une propagation du gradient plus fluide.



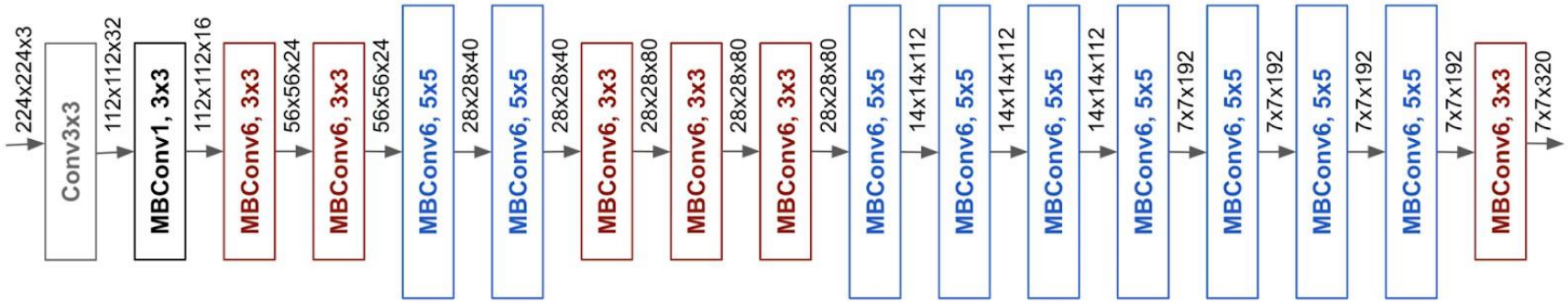
EfficientNet (2019)

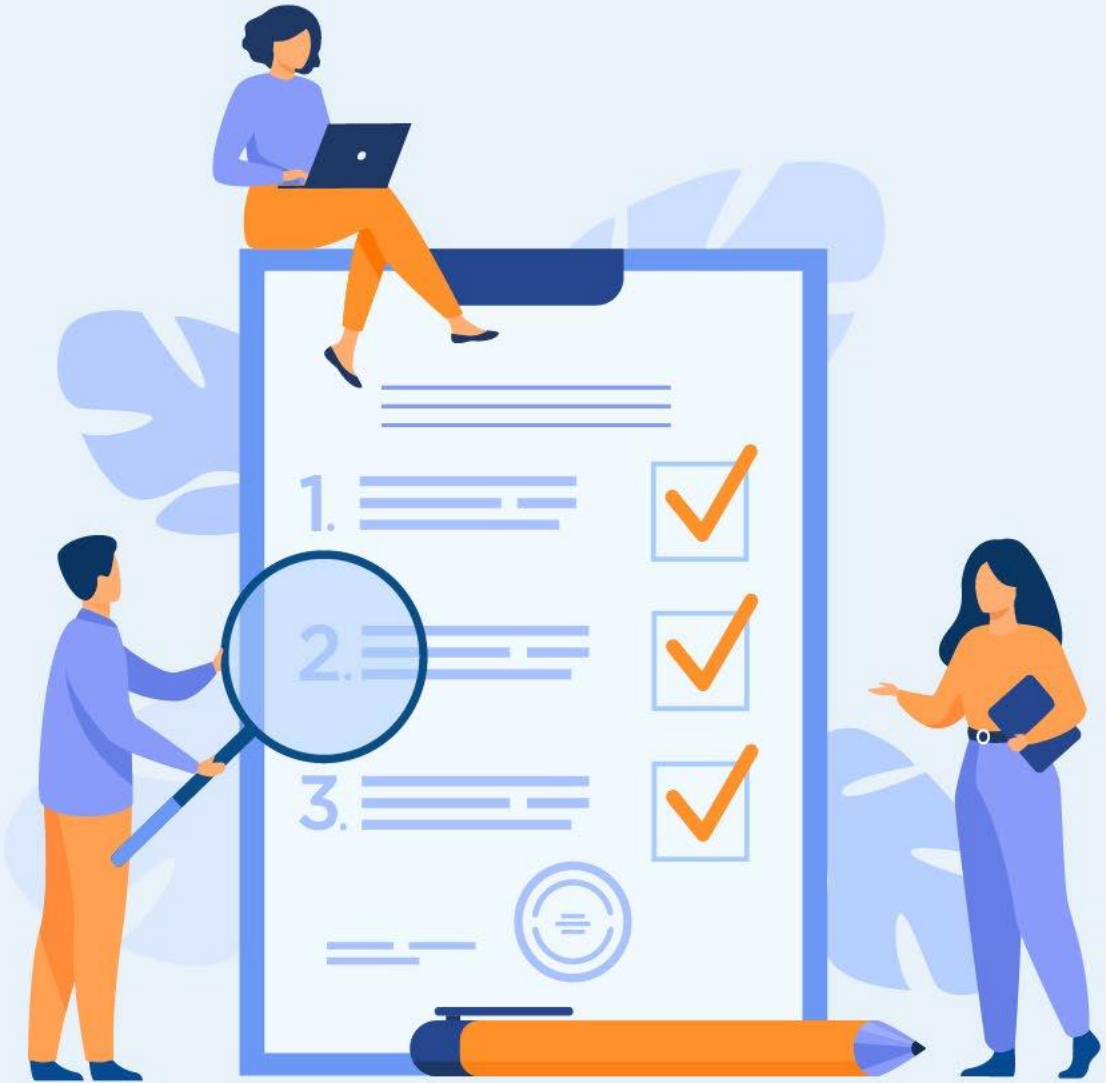
EfficientNet est une architecture développée par Mingxing Tan et Quoc V. Le. EfficientNet a pour objectif de fournir un modèle CNN performant et efficace en optimisant conjointement la profondeur, la largeur et la résolution spatiale du réseau.

EfficientNet propose un schéma de scaling compound qui optimise simultanément la profondeur, la largeur et la résolution spatiale du réseau. Cela est réalisé en utilisant des coefficients de mise à l'échelle pour contrôler la taille des différentes dimensions du réseau, permettant ainsi de trouver un bon équilibre entre la performance et l'efficacité.

EfficientNet utilise des blocs Mobile Inverted Residual Bottleneck (MBConv) comme élément de base de son architecture. Les blocs MBConv combinent des convolutions profondes et des convolutions à faible coût pour capturer efficacement les informations dans différentes échelles spatiales.

EfficientNet introduit des coefficients de mise à l'échelle (phi) pour contrôler la taille globale du modèle. Les valeurs de phi déterminent les dimensions du modèle, y compris la profondeur du réseau, la largeur des couches (nombre de canaux) et la résolution spatiale des images en entrée.





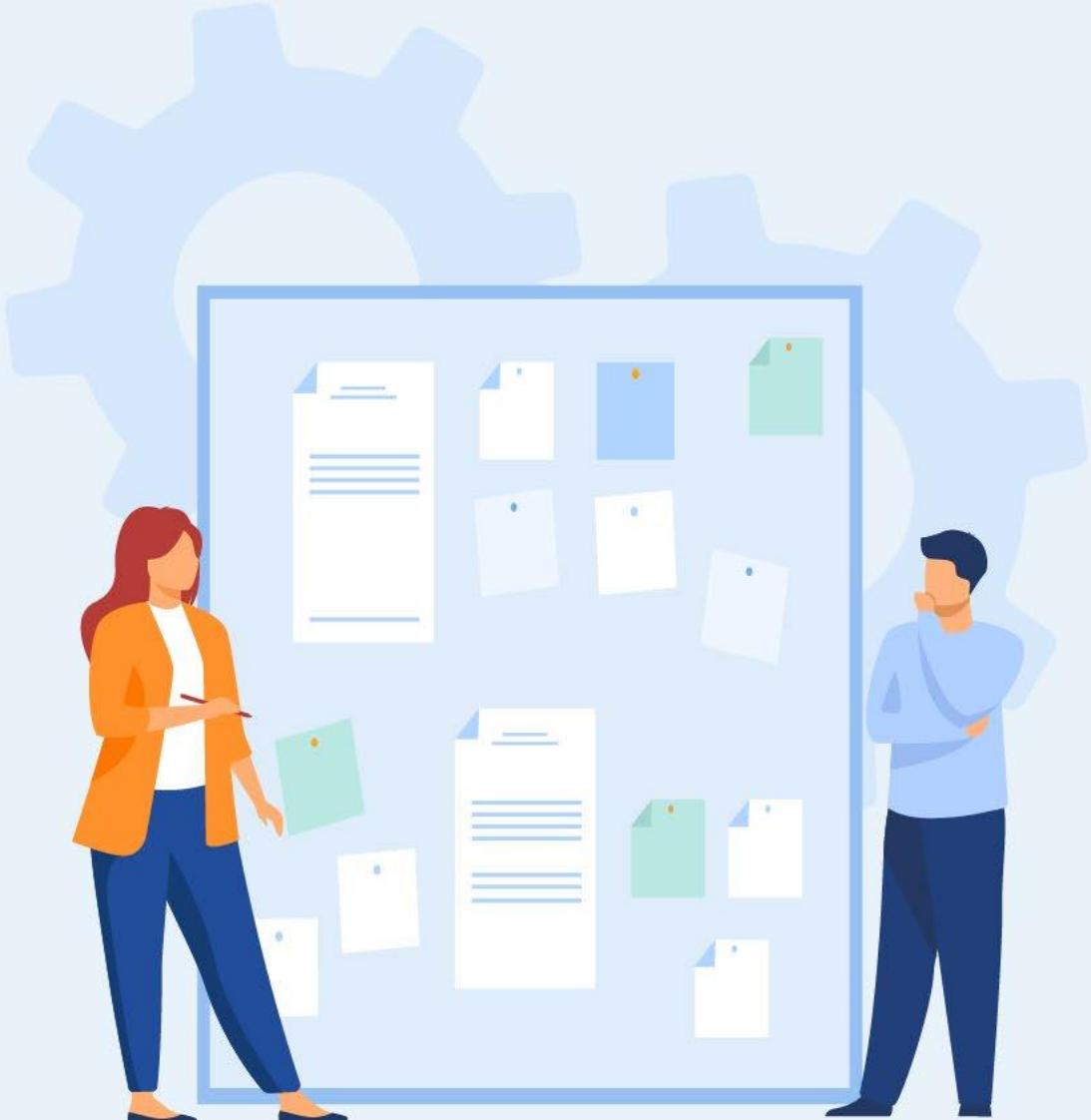
CHAPITRE 2

OPTIMISER LES RÉSEAUX À CONVOLUTION

Ce que vous allez apprendre dans ce chapitre :

- Les différents scénarios d’apprentissage

10 heures



CHAPITRE 2

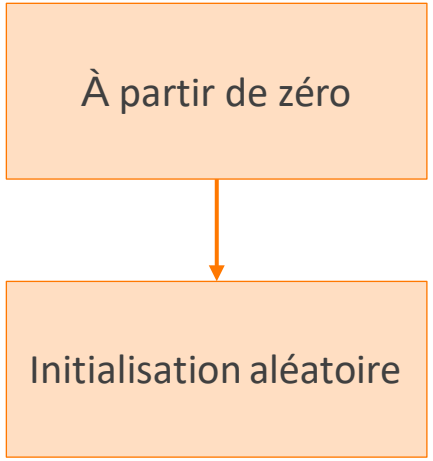
OPTIMISER LES RÉSEAUX À CONVOLUTION

1. **Scénarios d’entraînement**
2. Avantages et inconvénients

À partir de zéro

L'entraînement d'un réseau à convolution à partir de zéro, également appelé « from scratch » en anglais, implique l'initialisation du réseau avec des poids aléatoires et l'optimisation de ces poids en utilisant un ensemble de données d'entraînement pour résoudre une tâche spécifique.

Il est à noter que l'entraînement d'un CNN à partir de zéro peut être computationnellement intensif, surtout si le modèle est profond et les données d'entraînement sont volumineuses. Dans certains cas, le transfert d'apprentissage (fine-tuning) à partir de modèles pré-entraînés peut être préférable pour tirer parti des connaissances préalables et accélérer le processus d'apprentissage.

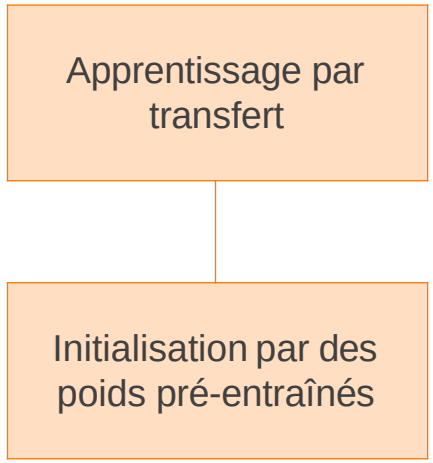


Apprentissage par transfert

L'apprentissage par transfert (transfer learning en anglais) est une technique couramment utilisée dans l'apprentissage automatique, en particulier dans le contexte des réseaux de neurones convolutifs (CNN). Plutôt que de former un modèle à partir de zéro, l'apprentissage par transfert consiste à utiliser un modèle pré-entraîné sur une tâche similaire comme point de départ, puis à l'adapter à une nouvelle tâche spécifique.

- ✓ Utiliser des modèles pré-entraînés
  - Entraînés sur un grand ensemble de données de référence pour résoudre un problème similaire à celui que nous voulons résoudre

- ✓ Utilisation dans le cas de:
  - Peu de données
  - Limite des ressources matérielles



1. CNN comme extracteur de caractéristiques fixes: CNN pré-entrainé sur ImageNet: on supprime la dernière couche entièrement connectée (les sorties de cette couche sont les 1000 scores de classe), puis on traite le reste du CNN comme un extracteur de caractéristiques fixes pour le nouvel ensemble de données.

2. Fine-tuning (réajustement des poids) CNN: Affiner les poids du réseau pré-entrainé en poursuivant la rétropropagation. Il est possible de régler avec précision toutes les couches du CNN, ou il est possible d'affiner toutes les couches ou seulement les couches supérieures du CNN.

3. Modèle pré-entraînés : Les chercheurs partagent leurs modèles comme dans Model Zoo.

Source: https://github.com/BVLC/caffe/wiki/Model-ZooTable of Contents

Exemple : Modèle pré-entraînés

Berkeley-trained models

Network in Network model

ResFace101: ResNet-101 for Face Recognition

Places-CNN model from MIT

GoogLeNet GPU implementation from Princeton

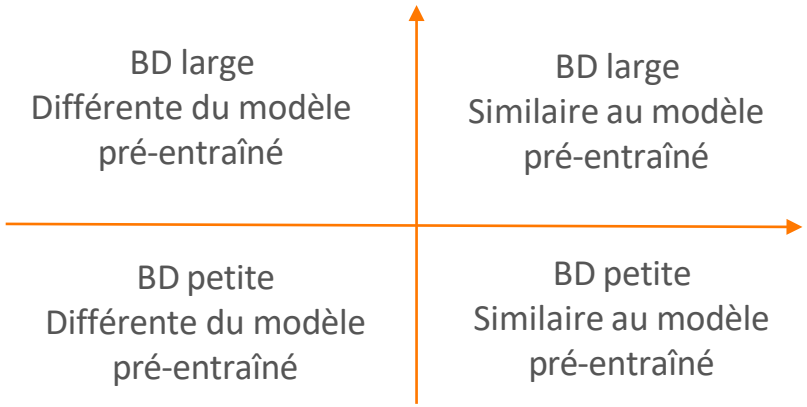
Fully Convolutional Networks for Semantic Segmentation (FCNs)

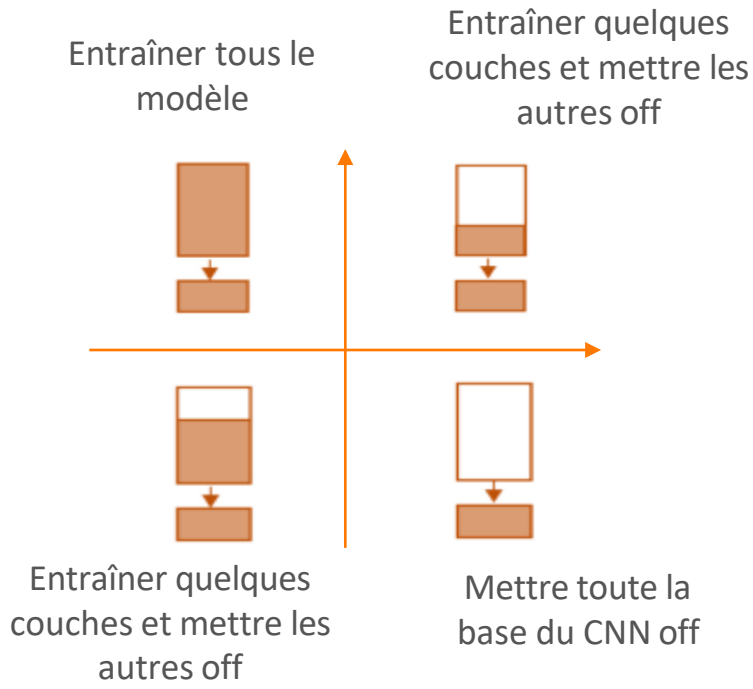
- Cascaded Fully Convolutional Networks for Biomedical Image Segmentation
- Deep Networks for Earth Observation

Récapitulatif

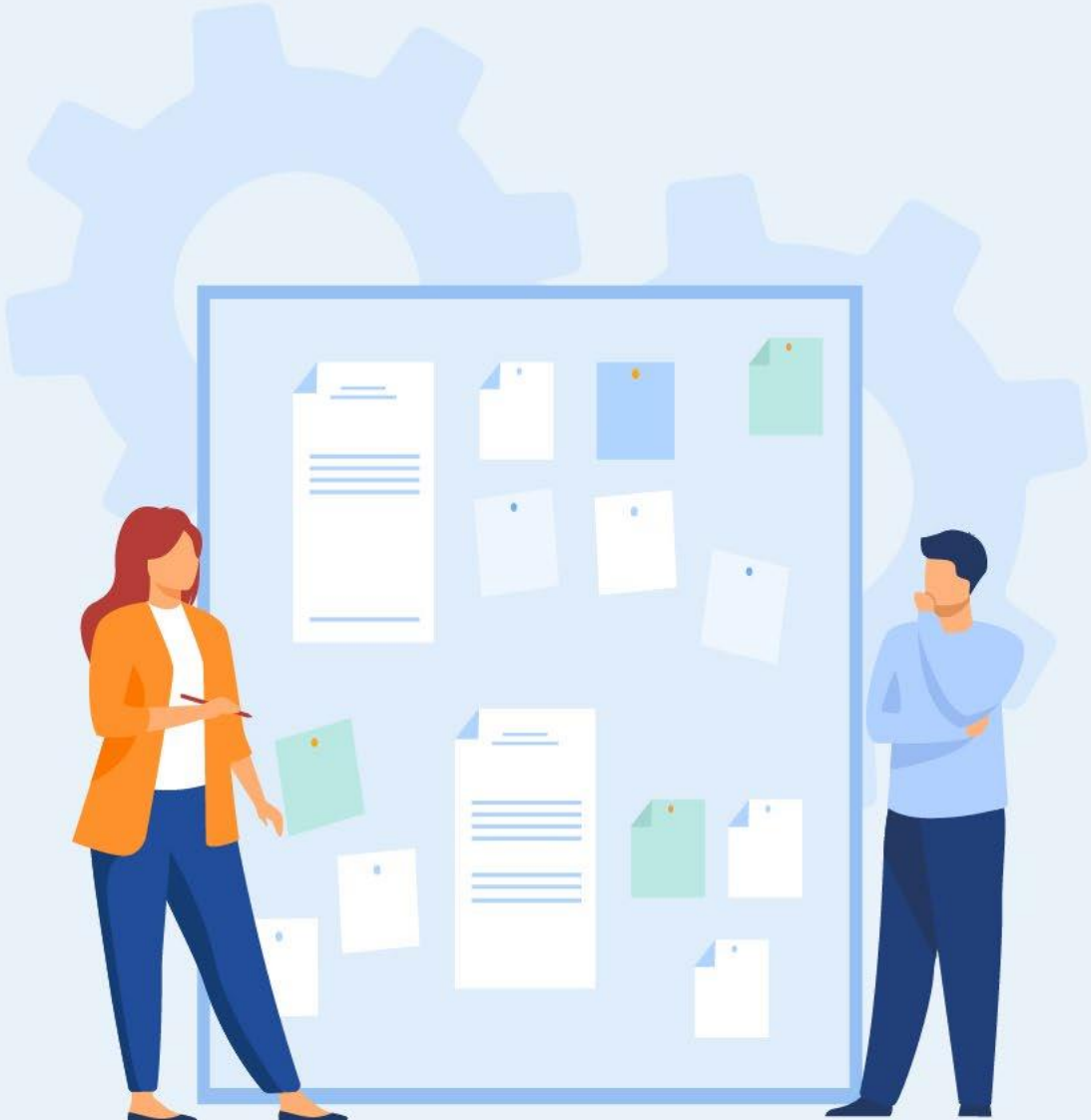
Les scénarios d'apprentissage d'un réseau à convolution dépendent de deux contraintes :

- ✓ La taille de la base de données
- ✓ La similarité de la base de données avec celle des modèles pré-entraînés





PARTIE 2



CHAPITRE 2

OPTIMISER LES RÉSEAUX À CONVOLUTION

1. Scénarios d’entraînement
2. **Avantages et inconvénients**

Avantages

- ✓Adaptation aux données spatiales
- ✓Invariance aux translations et déformations
- ✓Extraction automatique des caractéristiques
- ✓Apprentissage hiérarchique
- ✓Capacité à gérer de grandes quantités de données

Inconvénients

- ✓Besoin de données étiquetées
- ✓Besoin de puissance de calcul
- ✓Risque de surapprentissage
- ✓Interprétabilité limitée
- ✓Besoin de grandes quantités de données