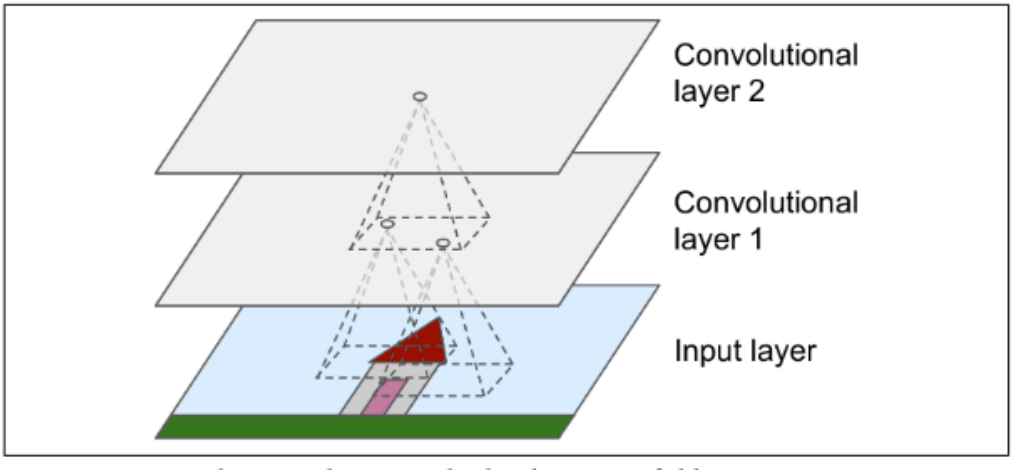
CNN Convolutional Neural Networks

On utilise les CNN pour des tâches difficiles comme reconnaître une photo de maison.

⚡ Mais pourquoi ne pas simplement le faire avec un Artificial Neural Network ?

On aurait besoin de trop de poids, c'est juste limite pour le MNIST.



🔥 L'idée du CNN

Contrairement au MLP, les layers des CNN ne sont pas tous connectés a chaque pixel de l'image mais qu'a une partie. Chaque neurone de la deuxième couche convolutionnelle est connecté à une petite région de la couche précédente, ce qui permet au réseau de détecter d’abord des caractéristiques simples, puis de les combiner progressivement en motifs plus complexes. Cette structure hiérarchique, similaire à celle des images réelles, rend les CNN particulièrement efficaces pour la reconnaissance d’images.

🔥 Filtres et feature maps

Les poids d’un neurone dans une couche convolutionnelle peuvent être vus comme une petite image (matrice) de la même taille que son **champ réceptif** (receptive field).

Par exemple un filtre de **7×7** avec **des 1 au centre vertical** détectera uniquement les **lignes verticales**.

Dans une couche convolutionnelle, tous les neurones appliquent plusieurs filtres en parallees. mais elles partagent **tous** les mêmes poids pour chaque filtre ! Chaque neurone, pour chaque filtre :

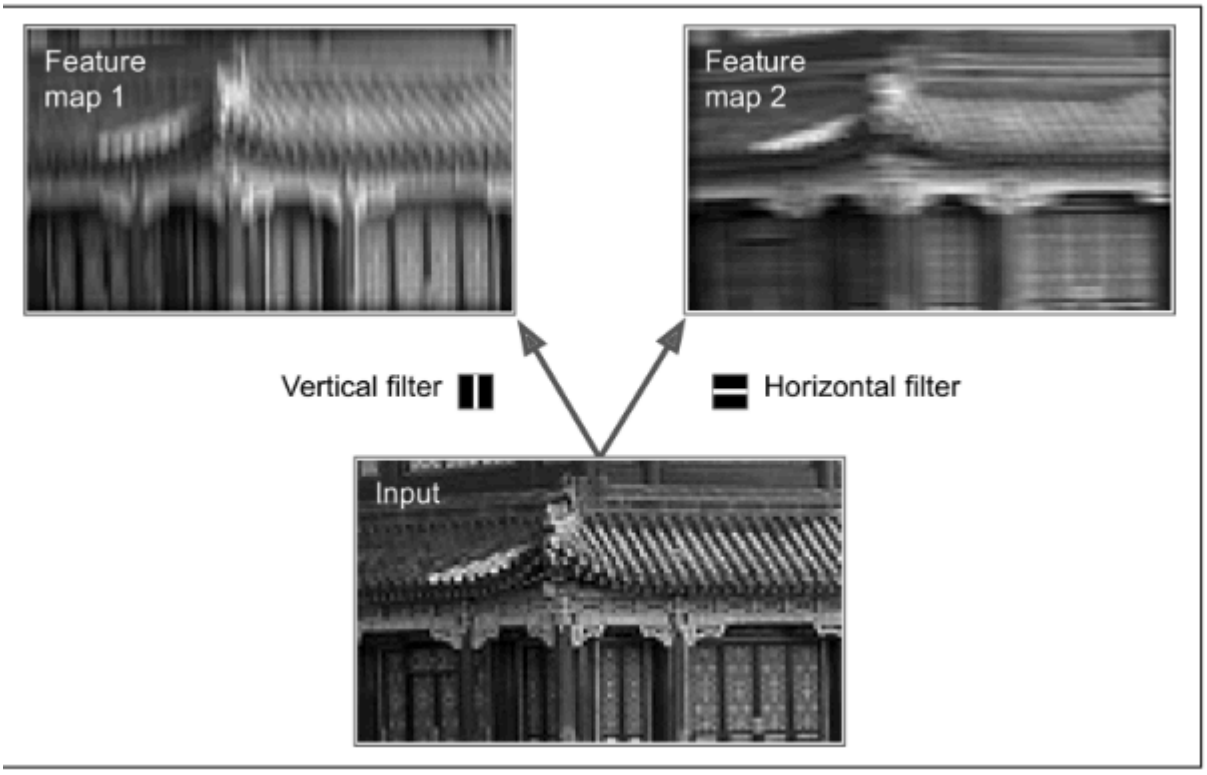
- Regarde une petite région de l'image (le **champ réceptif**),
- Applique **le même filtre** (les mêmes poids),
- Produit **une valeur** qui dit : *« est-ce que le motif que je cherche est présent ici ? »*

Ce partage des poids rend le modele beaucoup plus petit.

Le résultat est appelé une **carte de caractéristiques** (*feature map*), qui montre les endroits de l'image où le filtre détecte quelque chose d'important.

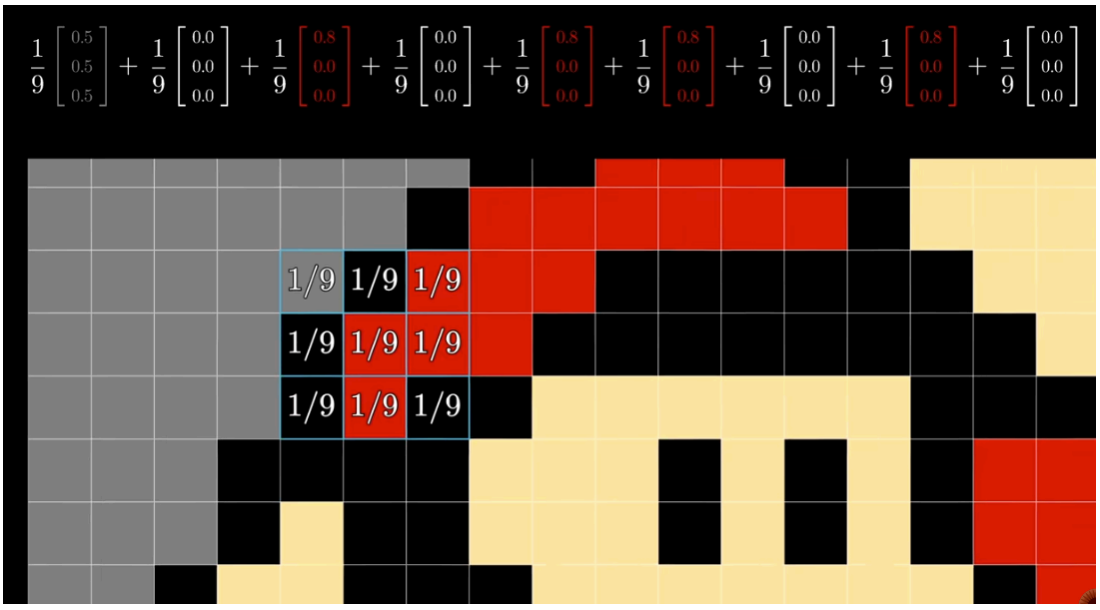
Pas besoin de programmer manuellement ces filtres :

- Pendant **l'entraînement**, le réseau apprend **tout seul** quels filtres sont les plus utiles pour la tâche (comme la reconnaissance d'image).
- Les couches supérieures du réseau vont alors combiner ces filtres simples (lignes, bords...) pour reconnaître des motifs plus complexes (formes, objets...).



Formule de la convolution (sur les)

$$z_{i,j,k} = b_k + \sum_{u=0}^{fh-1} \sum_{v=0}^{fw-1} \sum_{k'=0}^{fn'-1} x_{i',j',k'} \cdot w_{u,v,k',k}$$

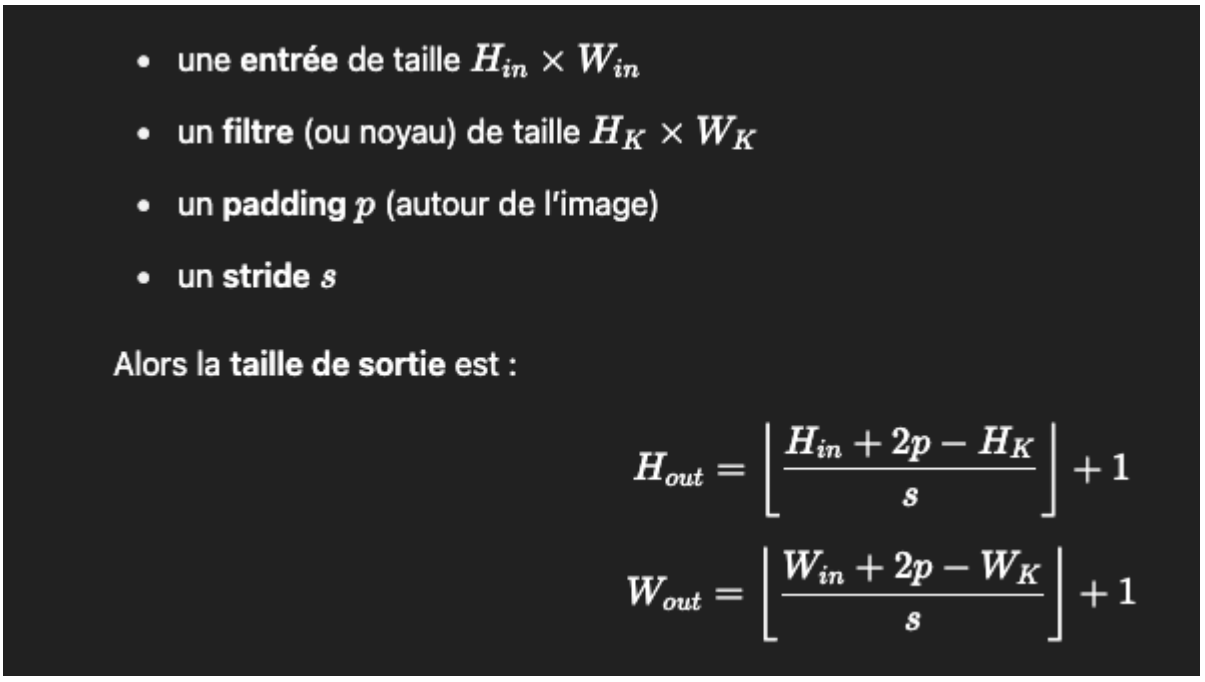


- **Stride** s : nombre de pixels à sauter entre chaque application du filtre
- **Padding** p : nombre de pixels à ajouter autour de l'image pour gérer les bords

-1	**1**	**1**	**1**	-1
-1	**1**	-1	**1**	-1
-1	**1**	**1**	**1**	-1
-1	-1	-1	**1**	-1
-1	-1	-1	**1**	-1
-1	-1	**1**	-1	-1
-1	**1**	-1	-1	-1

-1	**1**	**1**	**1**	-1
-1	**1**	-1	**1**	-1
-1	**1**	**1**	**1**	-1
-1	-1	-1	**1**	-1
-1	-1	-1	**1**	-1
-1	-1	**1**	-1	-1
-1	**1**	-1	-1	-1

Disadvantage: corner pixels don't contribute as much in feature detection



Les 2p c'est parce qu'on ajoute p pixels en haut, et p pixel en bas. H_in - H_k parce que plus le filtre est grand plus on doit commencer loin du bord pour être dans une plage valide.

Calculer le nombre d'opérations ?

Chaque output pixel est le résultat d'une convolution (c'est-a-dire du dot product entre une région de taille $H_K \cdot W_K \cdot C_{\text{in}}$ et un noyau de la même taille), et il faut le faire pour chaque pixel de sortie soit $H_K \cdot W_K \cdot C_{\text{in}} \cdot H_{\text{out}} \cdot W_{\text{out}} \cdot C_{\text{out}}$ multiplications.

Si on veut comparer avec un fully-connected layer, ou la taille du kernel est la taille de l'image, on voit que c'est bien plus petit.

average pooling

L'**Average pooling** consiste à **diviser l'image en petites régions (souvent carrées)** et à **calculer la moyenne des valeurs** dans chacune de ces régions.

Max pooling divise une image (ou carte de caractéristiques) en petits blocs (ex. 2×2) et **retient uniquement la valeur maximale** de chaque bloc.