

STT3030 - Cours #8

Arthur Charpentier

Automne 2024

Réseaux de neurones

Un réseau de neurones est une fonction \hat{f} pour l'apprentissage supervisé, ils entrent alors dans le cadre des fonctions qu'on a vues à présent.

- ▶ On veut $\hat{f}(x_i) = \hat{y}_i$ avec la plus petite erreur de prédiction possible.
- ▶ Les réseaux de neurones sont extrêmement flexibles puisqu'ils la possibilité d'apprendre par eux-mêmes des relations non linéaires et des interactions.

Représentation graphique

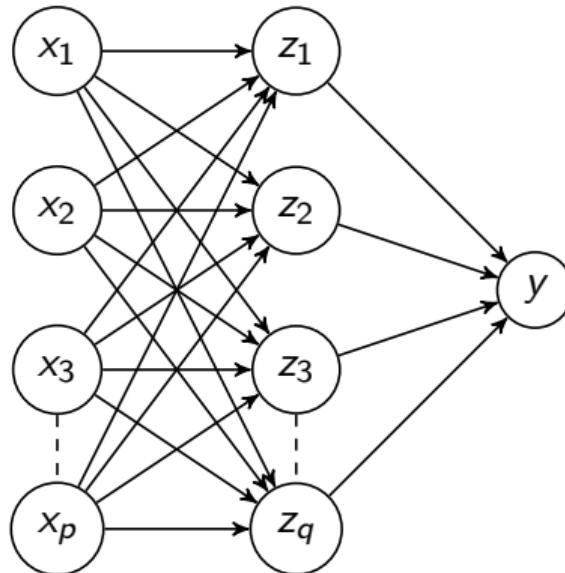


Figure 1: Représentation graphique d'un réseau de neurones. La première couche représente nos prédicteurs x (couche d'entrée). La deuxième couche z est une *couche cachée* de neurones. Chaque arrête représente un coefficient.

Représentation fonctionnelle

On utilise souvent la représentation graphique, celle-ci en met plein la vue pour expliquer quelque chose de relativement simple.

Chaque élément z_h de la couche caché (disons \mathbf{z}) est une combinaison linéaire des éléments de la couche précédente (ici les prédicteurs \mathbf{x}) sur laquelle nous appliquons une fonction non linéaire:

$$z_l = g \left(\sum_{j=1}^p b_{l,j}^{(1)} x_j \right)$$

où g est une fonction non linéaire et où les $b_j^{(1)}$ forment une collection de paramètres (coefficients à apprendre).

Représentation fonctionnelle

On peut séquentiellement passer l'information à travers plusieurs couches, par exemple, pour le modèle à une couche cachée, la prochaine couche est la réponse prédite:

$$\hat{f}(\mathbf{x}) = \hat{y} = h(\mathbf{z}_{1 \times q} \mathbf{B}_{q \times 1}^{(2)})$$

ce qui donne

$$\hat{f}(\mathbf{x}) = \hat{y} = h(g(\mathbf{x}_{1 \times p} \mathbf{B}_{p \times q}^{(1)} \mathbf{B}_{q \times 1}^{(2)}))$$

- ▶ On parle souvent d'architecture du réseau pour parler de sa forme: nombre de neurones, nombre de couches, type de couche, etc...
- ▶ Au cours précédent on a vu un simple réseau à une couche cachée, on peut facilement ajouter des couches!

Représentation graphique

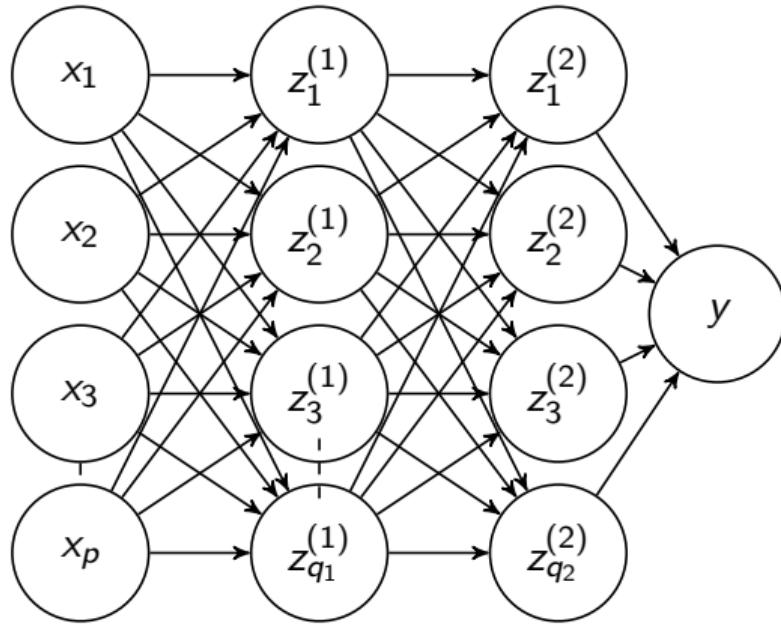


Figure 2: Représentation graphique d'un réseau de neurones. Remarquez qu'on indice q_j par la couche, chaque couche cachée peut avoir une différente quantité de neurones.

Représentation fonctionnelle

Ce qui donne

$$\hat{f}(\mathbf{x}) = \hat{y} = h(g_2(g_1(\mathbf{x}_{1 \times p} \mathbf{B}_{p \times q_1}^{(1)}) \mathbf{B}_{q_1 \times q_2}^{(2)}) \mathbf{B}_{q_2 \times 1}^{(3)})$$

Comment déterminer l'architecture optimale pour un réseau de neurones ?

- ▶ Question sans réponse (facile ou générale).
- ▶ Pour l'instant on fait une recherche d'hyperparamètre par validation croisée ou par sous-échantillonnage.
- ▶ C'est long et peu efficace.
- ▶ On utilise souvent des *tites règles du pouce* (rule of thumbs)

Architecture du réseau: nombre de couches

- ▶ En théorie, n'importe quelle fonction f peut être approximé avec une seule couche.
- ▶ Malgré tout en pratique il semble plus efficace d'utiliser plusieurs couches consécutives.
- ▶ Le terme *deep learning* fait référence aux *deep neural network*, aux réseaux de neurones avec un très grand nombre de couches.
- ▶ En pratique, ces modèles sont réservés pour des données avec plusieurs centaines de milliers d'observations.
- ▶ Ici, nous pouvons expérimenter avec 1,2 ou 3 couches et avec des données typiques de quelques milliers d'observations ce sera probablement suffisant.

Architecture du réseau: nombre de neurones

- ▶ Le nombre de neurones dans chaque couche dépend souvent du nombre de couches.
- ▶ Souvent le nombre de neurones est dicté par la taille de l'entrée et la taille de la sortie.
- ▶ On utilise les couches intermédiaires pour progressivement passer d'une taille à l'autre.
- ▶ Par exemple avec une taille 784 en entrée, deux couches cachées et une sortie de taille 10 (MNIST), on peut vouloir faire 784–392–196–10.
- ▶ Encore une fois ici, on peut vouloir faire de la validation croisée.

Architecture du réseau: fonction d'activation et type de couche.

- ▶ Les couches cachées sont traditionnellement complètement connectées telles qu'introduites au cours précédent.
- ▶ On peut choisir la fonction d'activation g par validation croisé, mais la fonction ReLu est le choix par défaut, Sigmoid est l'autre candidat traditionnel.
- ▶ La couche de sortie et la fonction d'activation qui en découle sont déterminées par la forme de la réponse.
- ▶ On a vu comment utiliser la fonction identité pour la régression ou bien logit pour une classification binaire.

Architecture du réseau: fonction d'activation et type de couche.

- ▶ Cela ne résout qu'une petite quantité de problèmes de prédition moderne.
- ▶ On doit donc concevoir des couches et/ou des fonctions d'activation spécialisées si nous observons d'autres structures de données
- ▶ Par exemple si la réponse une observation de survie (temps avant un évènement avec potentielle censure)
- ▶ Un exemple important que nous verrons aujourd'hui est la mise au point d'une couche d'entrée spécialisée pour les images. (matériel extra!)

La base MNIST

On va utiliser Keras et Tensor Flow,

- ▶ TensorFlow est une bibliothèque open source développée par Google pour le calcul numérique et l'apprentissage automatique, <https://www.tensorflow.org/install>
- ▶ Keras est une API pour construire et entraîner des modèles d'apprentissage profond. Il a été intégré à TensorFlow en tant que module officiel.
- ▶ PyTorch est une autre bibliothèque open source, développée par Facebook's AI Research lab (FAIR)

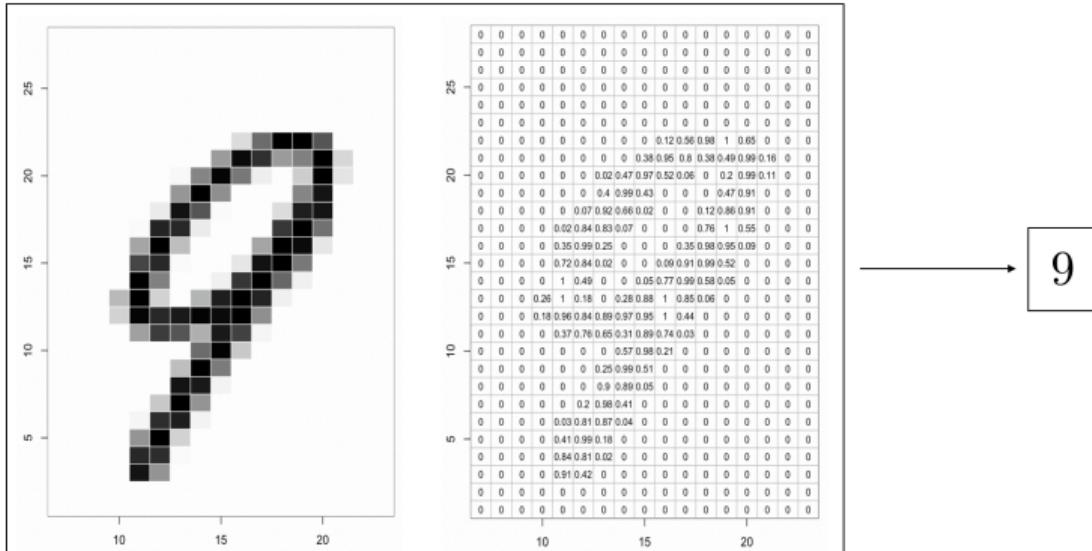
```
1 library(keras)
2 mnist = dataset_mnist()
```

(c'est un peu long à tourner...)

```
1 > table(mnist$train$y)
2
3   0    1    2    3    4    5    6    7    8    9
4 5923 6742 5958 6131 5842 5421 5918 6265 5851 5949
```

La base MNIST

Reconnaissance d'écriture manuscrite, $y \in \{0, 1, \dots, 9\}$



$$x_i \in \mathcal{M}_{28,28}$$

$$y_i \in \{0, 1, \dots, 9\}$$

La base MNIST

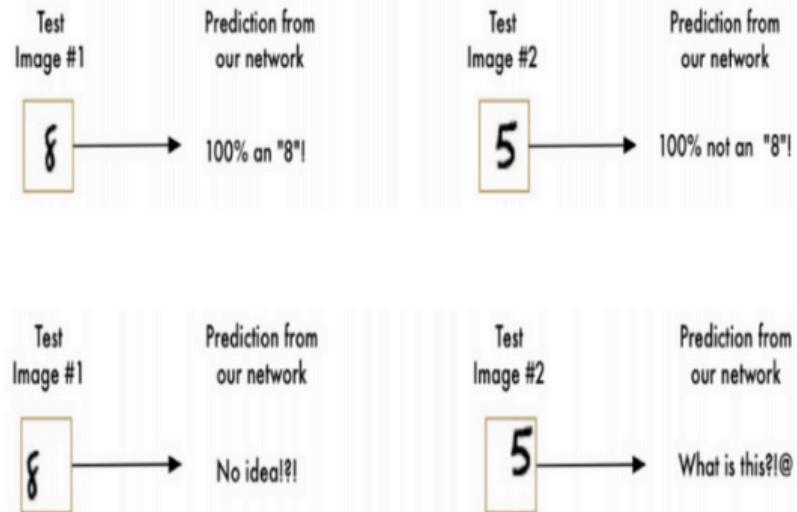


FIGURE 11.9. Examples of training cases from ZIP code data. Each image is a 16×16 8-bit grayscale representation of a handwritten digit.

La base MNIST

```
1 idx = which(mnist$train$y %in% c(3,8))
2 MV = NULL
3 for(i in idx){MV = cbind(MV,as.vector(mnist$train$x[i,,])) }
4 dim(MV)
5 [1] 784 11982
6 object.size(MV)
7 37575768 bytes
8 X_train = t(MV)
9 Y_train = mnist$train$y[idx]
10 df_train=data.frame(Y=Y_train,X_train)
11 idx = which(mnist$test$y %in% c(3,8))
12 MV = NULL
13 for(i in idx){MV = cbind(MV,as.vector(mnist$test$x[i,,])) }
14 X_test = t(MV)
15 Y_test = mnist$test$y[idx]
16 df_test=data.frame(Y=Y_test,X_test)
```

La base MNIST

Histoire d'avoir un élément de comparaison, on peut faire une régression

```
1 reg = glm((Y==8)~., data=df_train, family=binomial)
2 library(ROCR)
3 Y = as.numeric(Y_test)
4 S = as.numeric(predict(reg, newdata=df_test, type="response"))
5 pred = prediction(S,Y)
6 auc.perf = performance(pred, measure = "auc")
7 auc.perf@y.values[[1]]
8 [1] 0.9550989
9 plot(performance(pred,"tpr","fpr"))
```

Réseau de neurones convolutif

- ▶ On peut prendre les images de MNIST (28X28) et transformer les images (matrice de pixels) en vecteur (de taille $p = 784$).
- ▶ Cette réécriture “détruit” les liens de corrélation spatiale existants
- ▶ Nous avons endommagé les données en quelque sorte.
- ▶ Les réseaux de neurones convolutif sont conçus spécialement pour l'analyse d'images et respectent ainsi la corrélation spatiale des images.

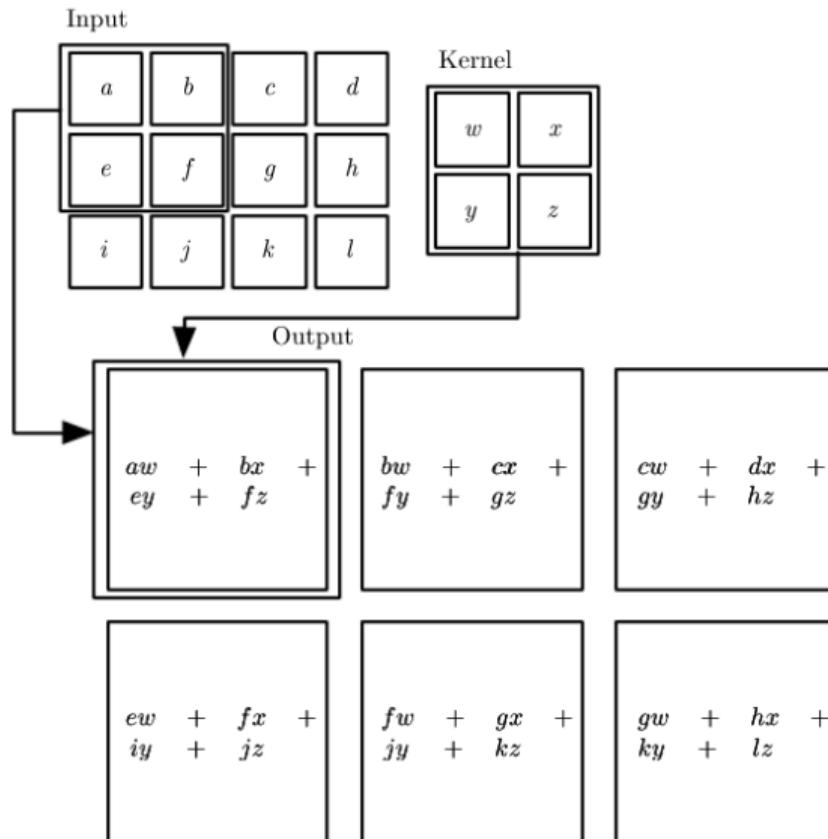
L'opérateur de convolution

Soit $*$ l'opérateur de la convolution 2 dimensions: il s'agit d'une combinaison linéaire entre une sous-matrice (un voisinage de l'image) et un noyau de convolution.

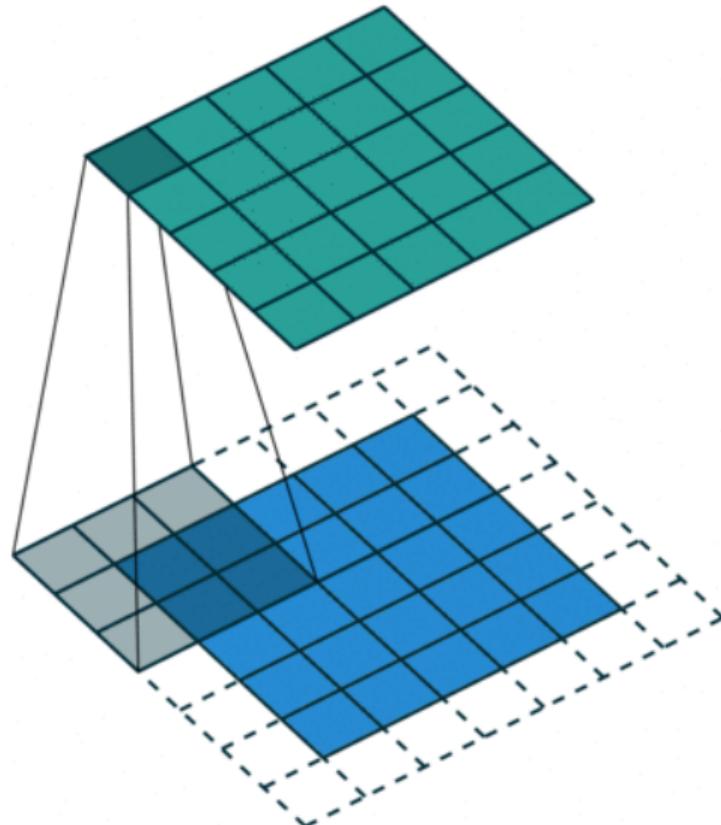
$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i + m, j + n) K(m, n)$$

On glisse le noyau K sur la matrice I et on obtient une nouvelle matrice S .

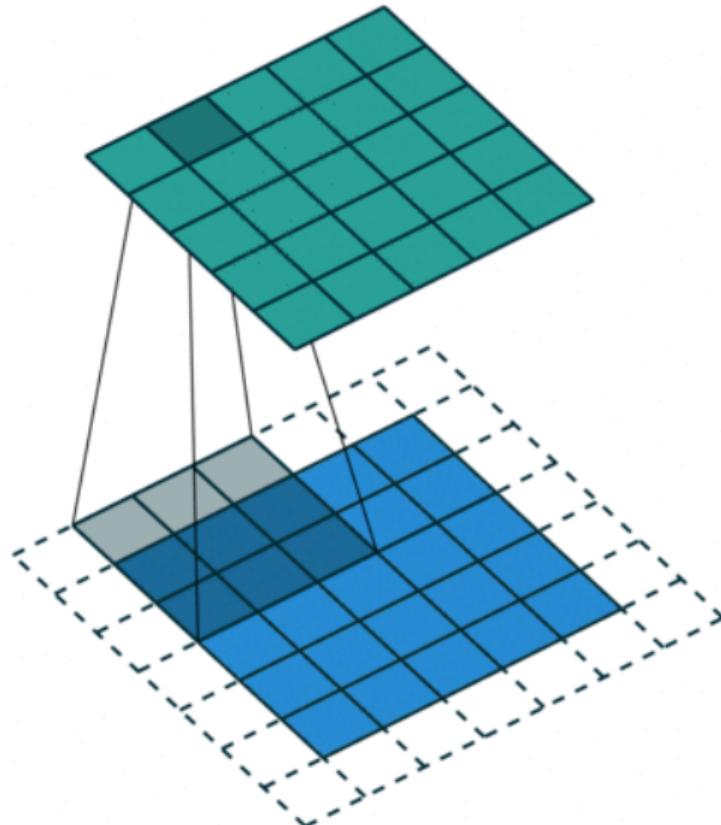
L'opérateur de convolution



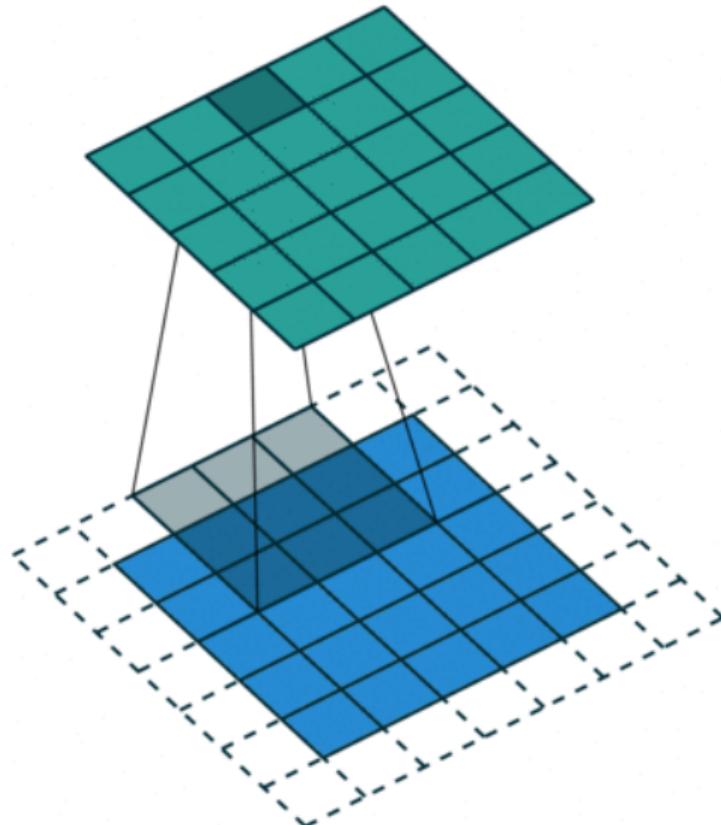
L'opérateur de convolution



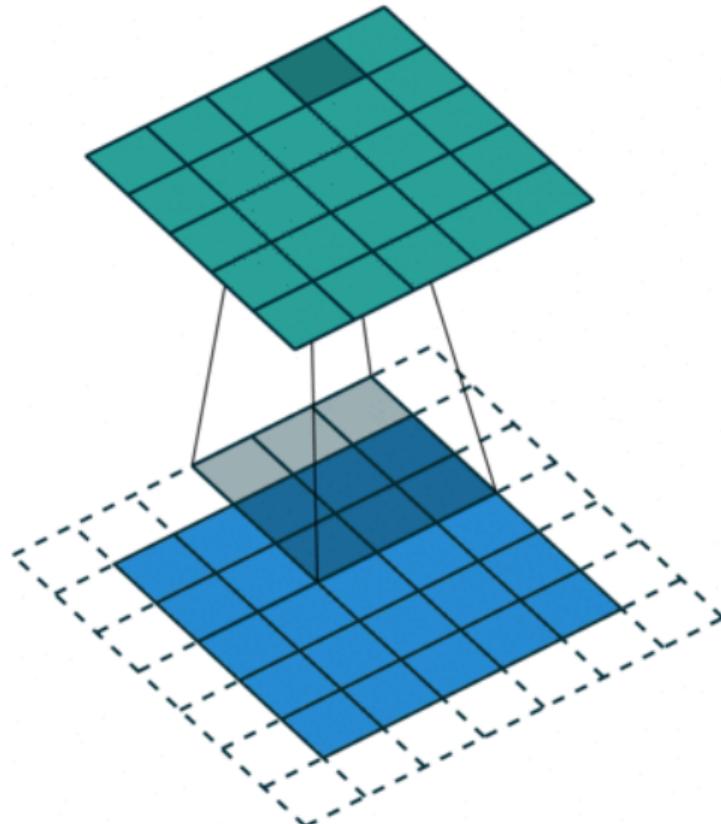
L'opérateur de convolution



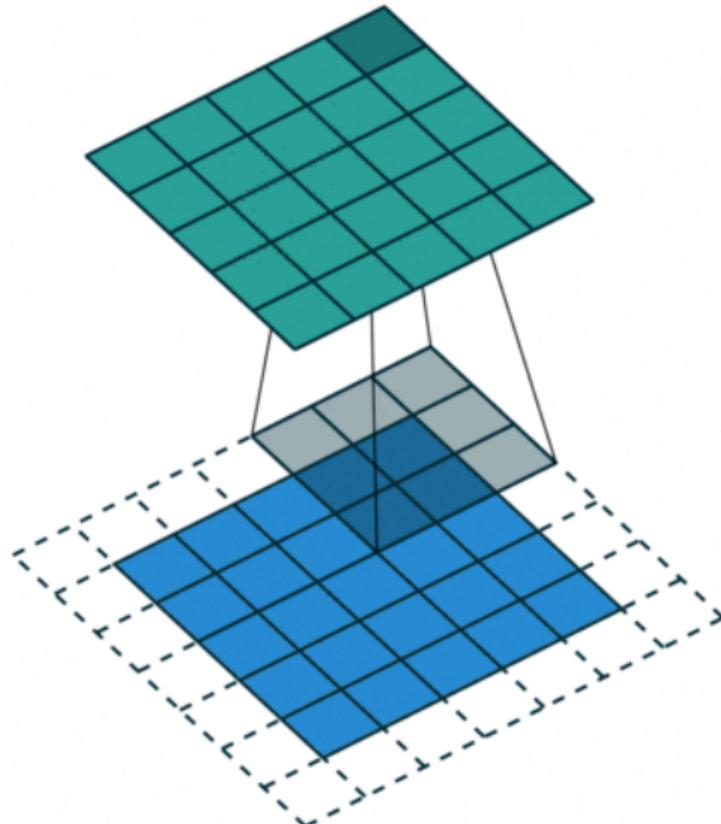
L'opérateur de convolution



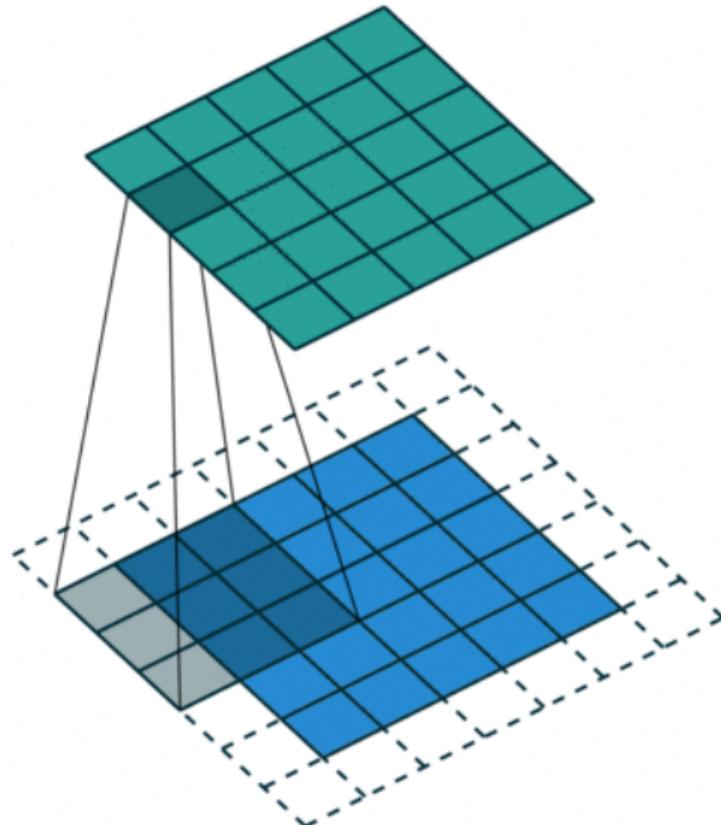
L'opérateur de convolution



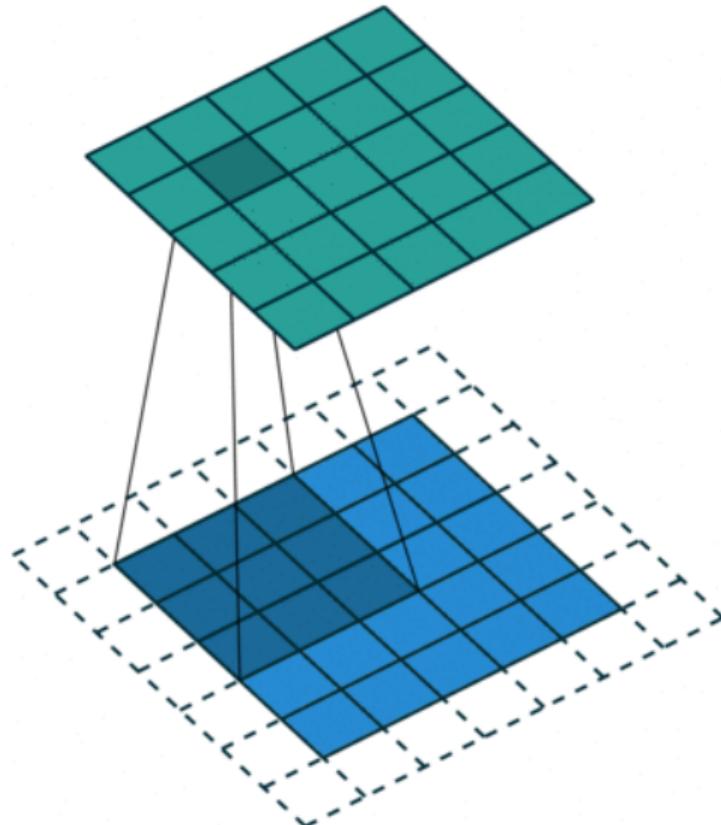
L'opérateur de convolution



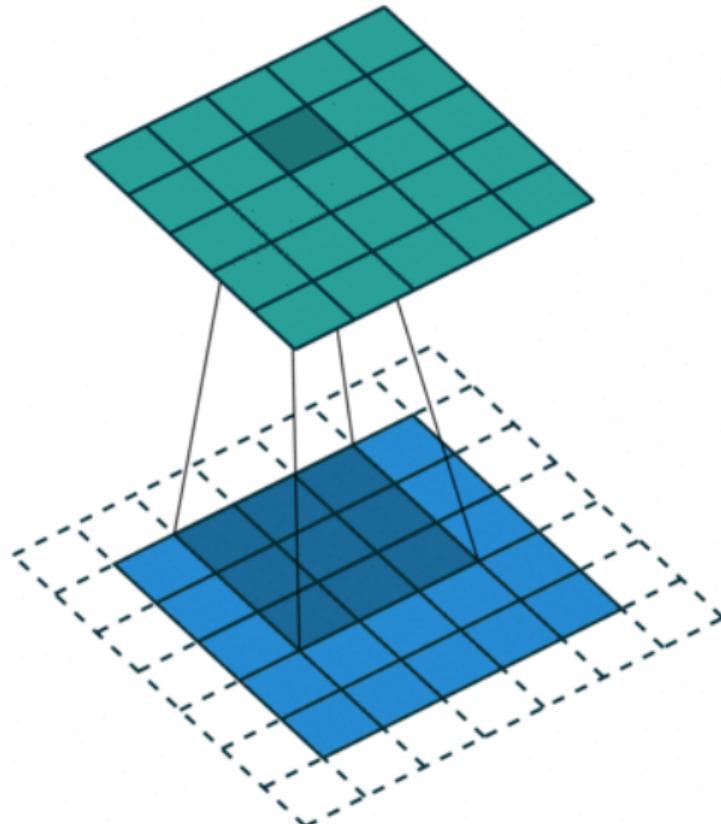
L'opérateur de convolution



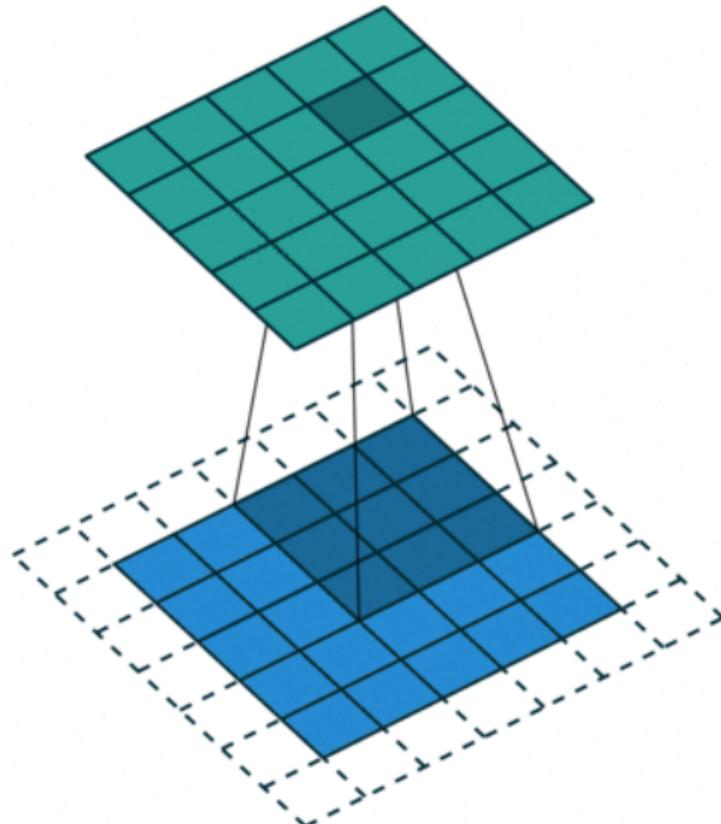
L'opérateur de convolution



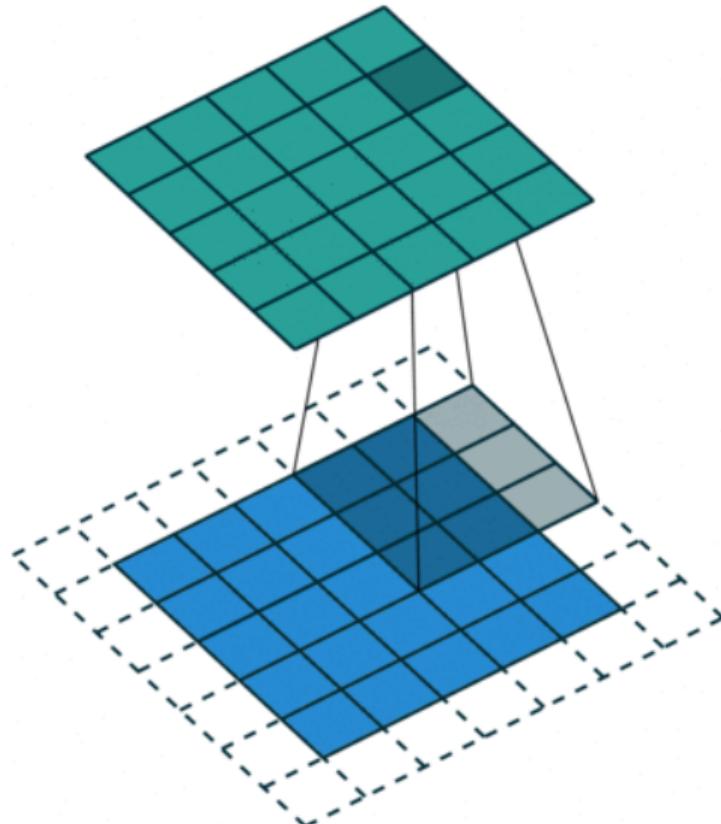
L'opérateur de convolution



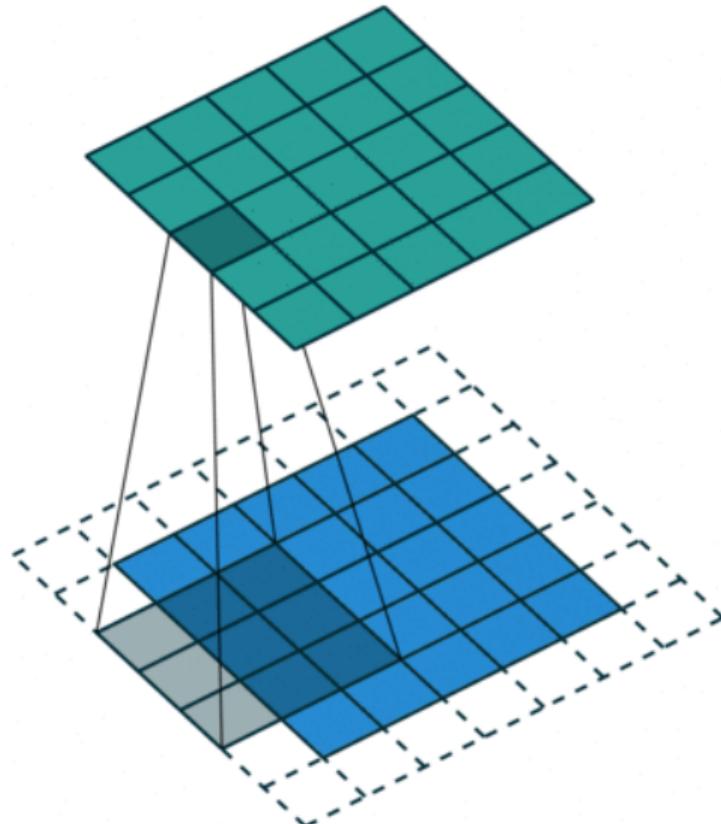
L'opérateur de convolution



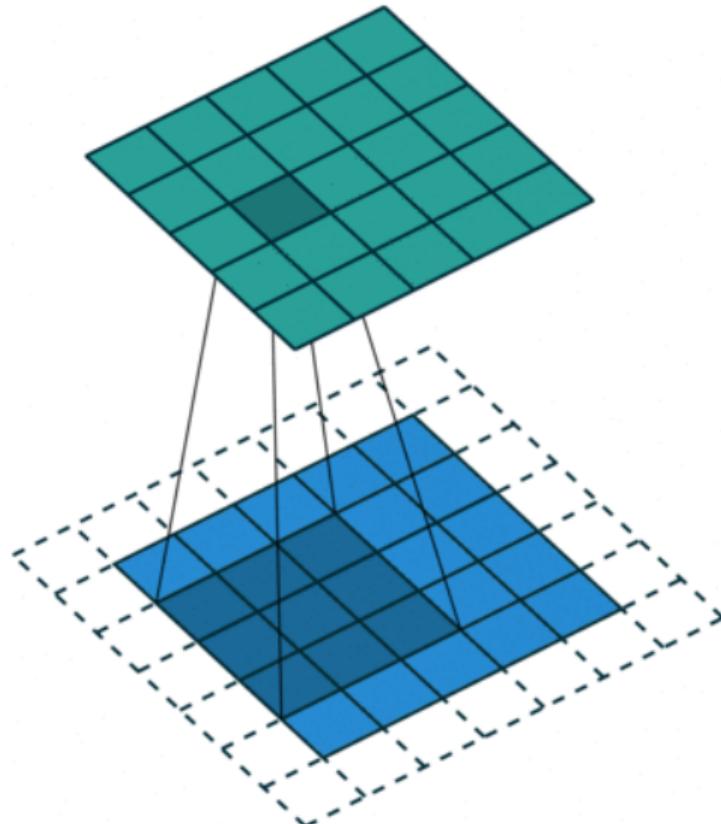
L'opérateur de convolution



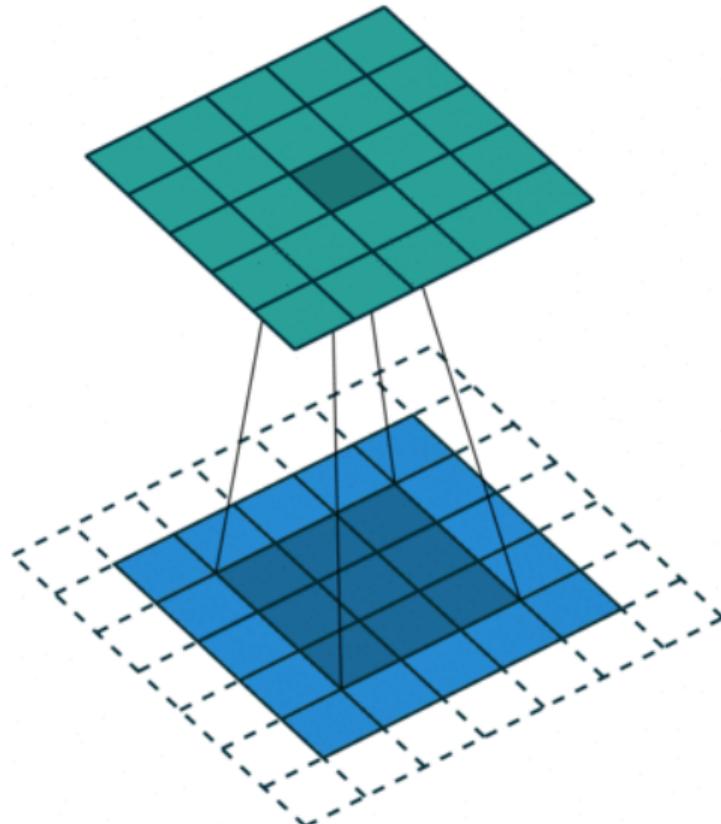
L'opérateur de convolution



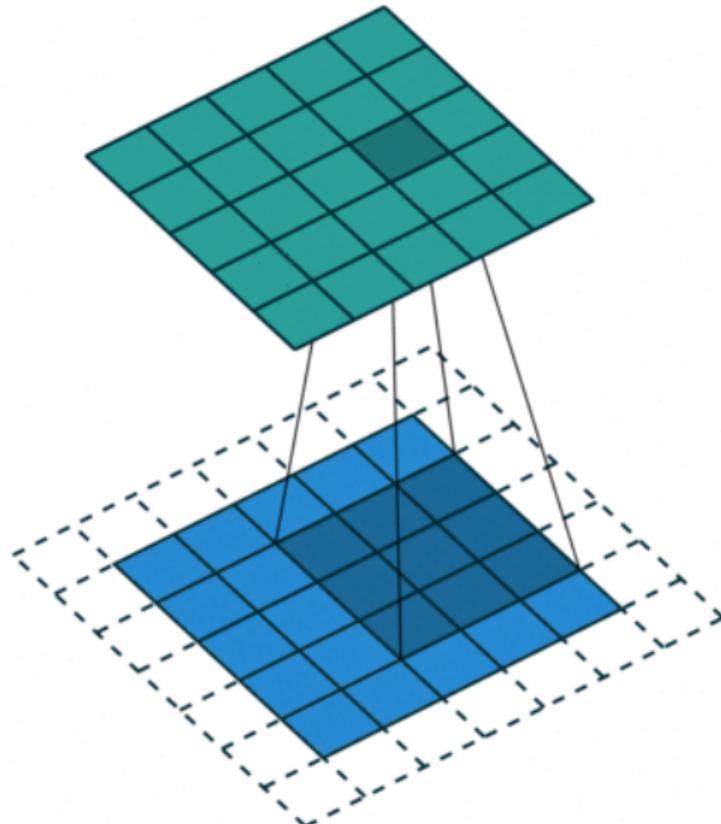
L'opérateur de convolution



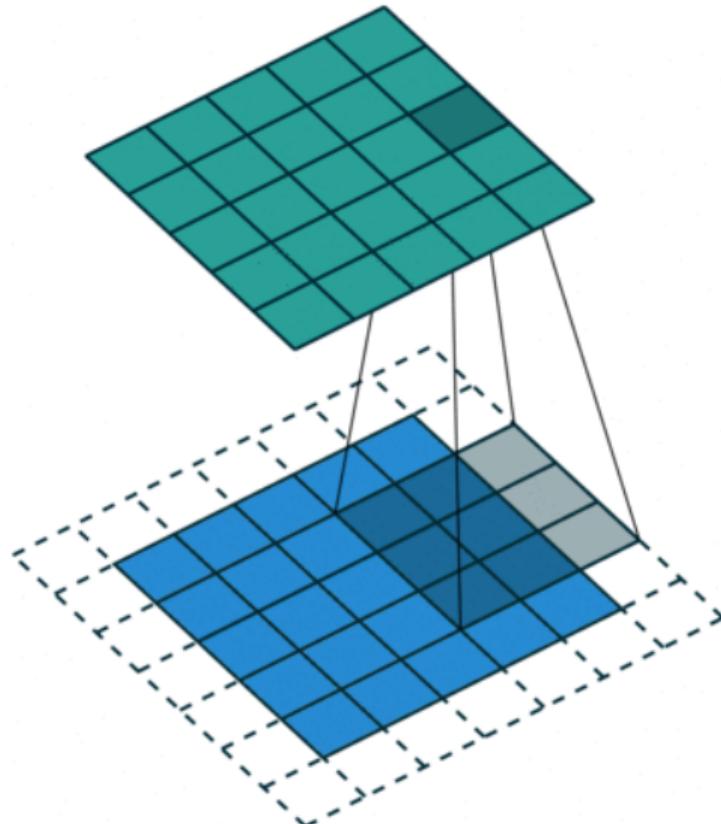
L'opérateur de convolution



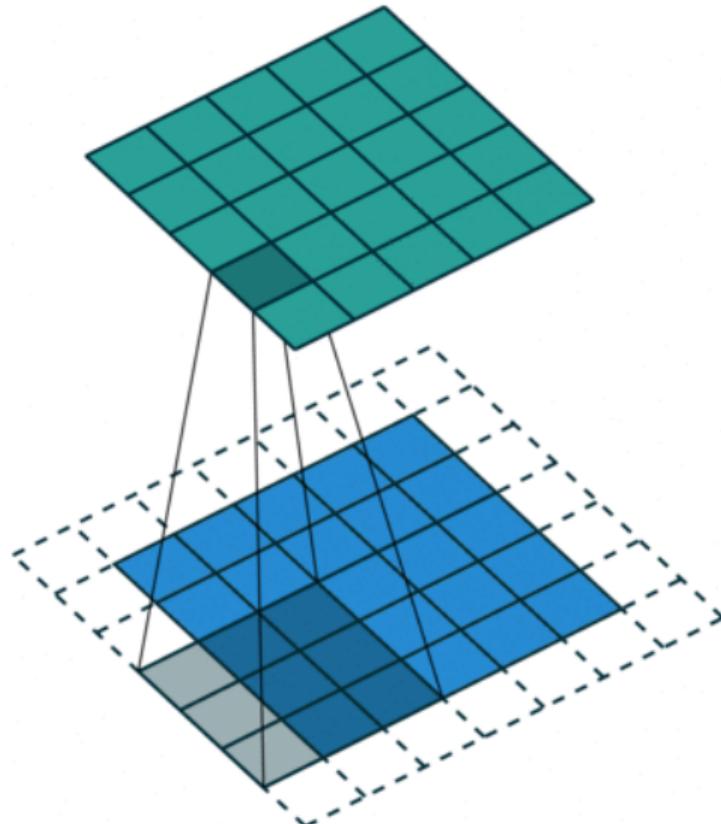
L'opérateur de convolution



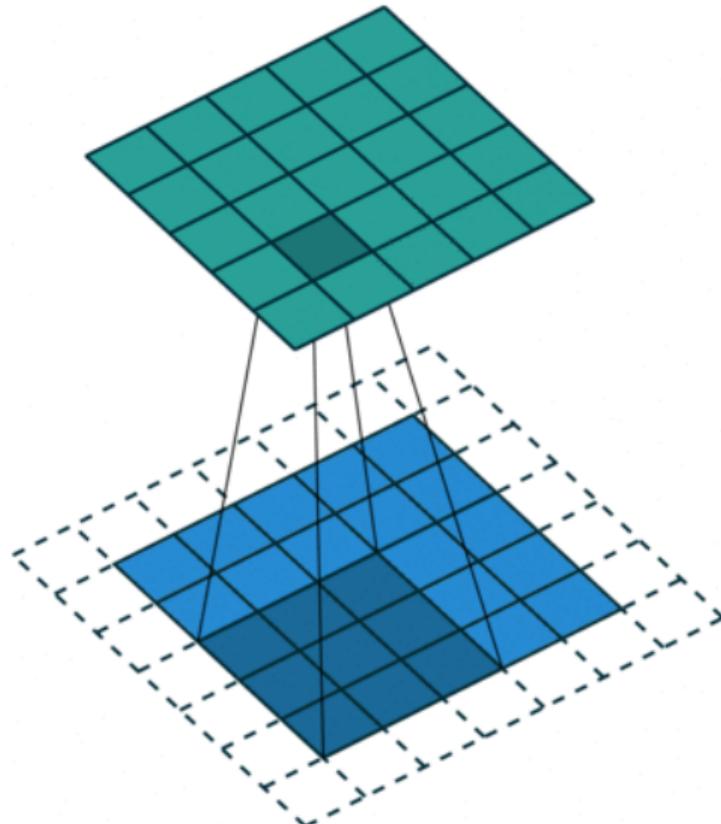
L'opérateur de convolution



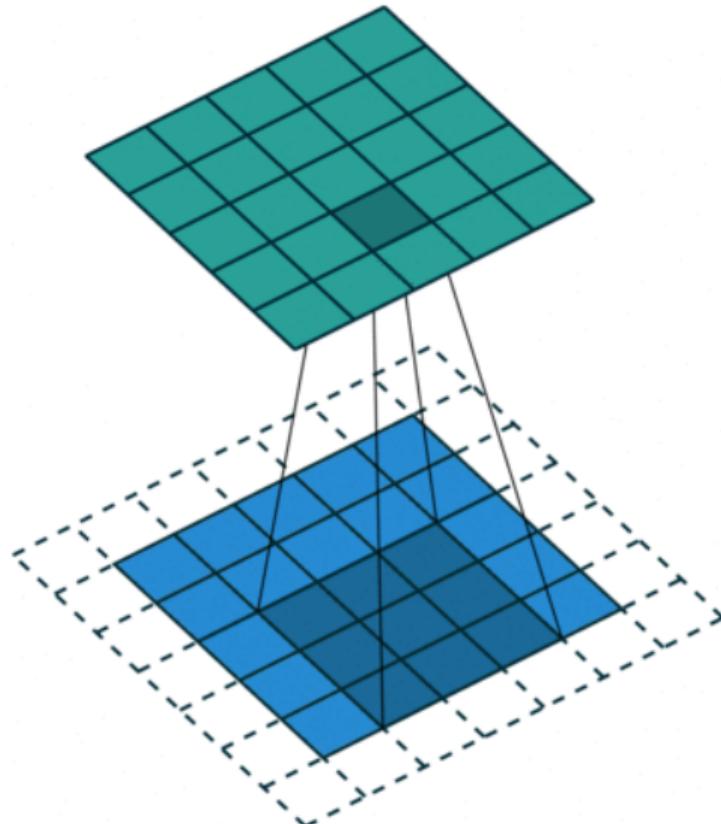
L'opérateur de convolution



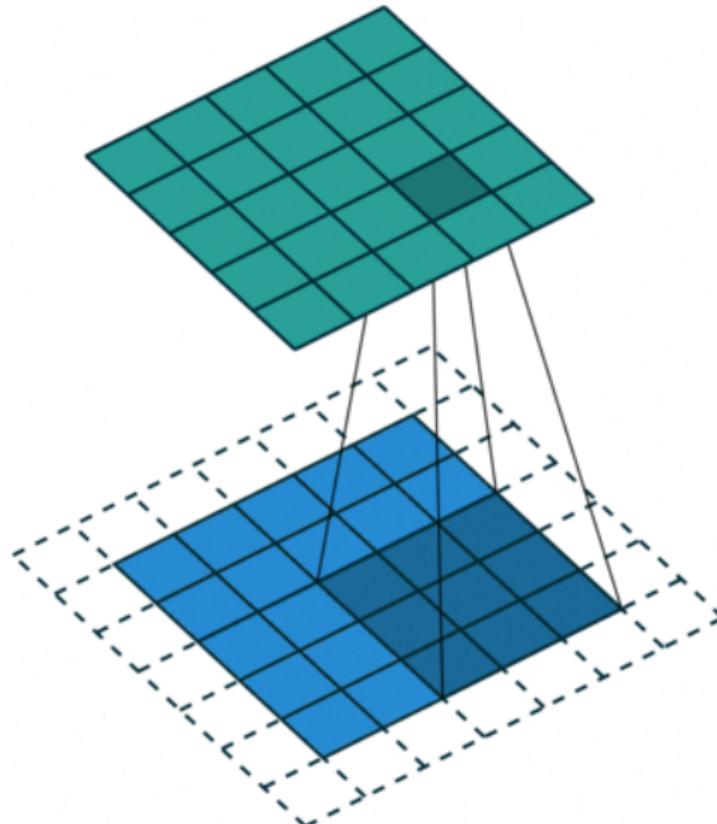
L'opérateur de convolution



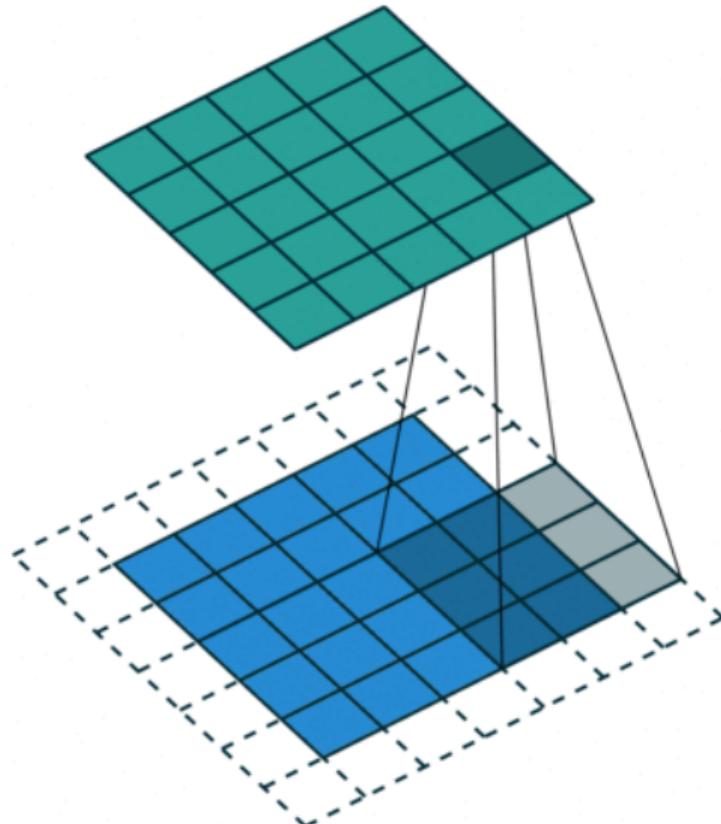
L'opérateur de convolution



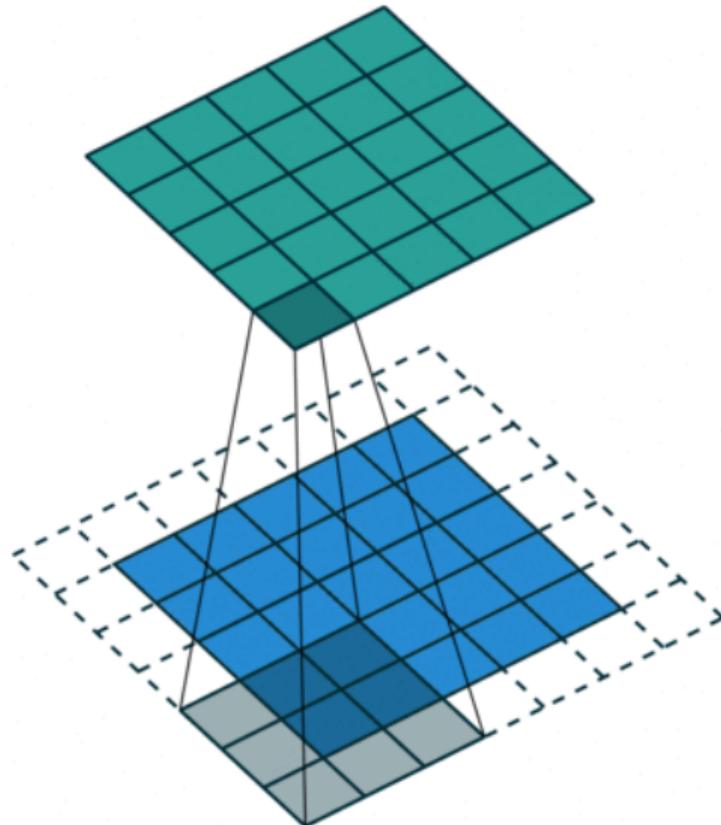
L'opérateur de convolution



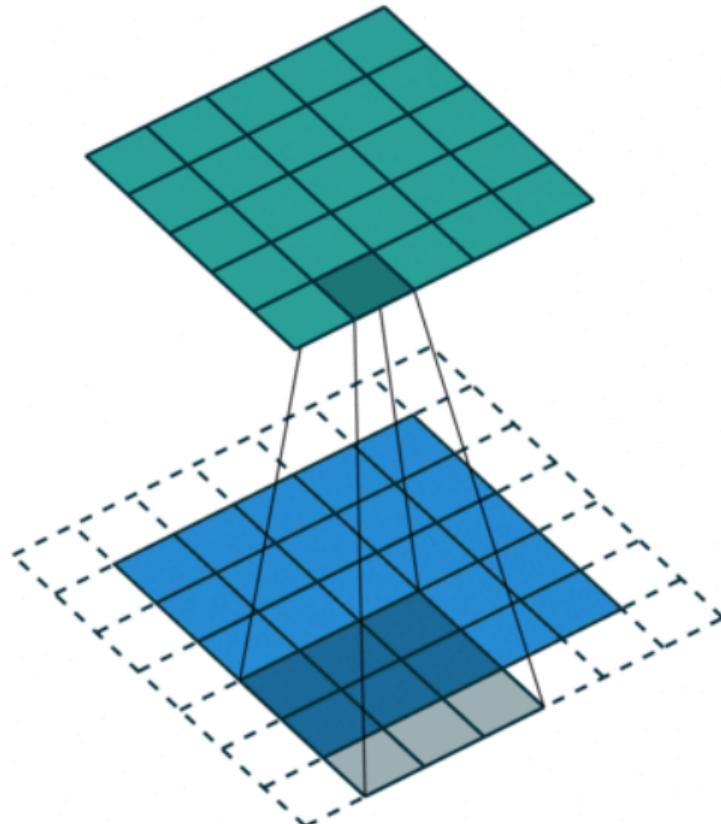
L'opérateur de convolution



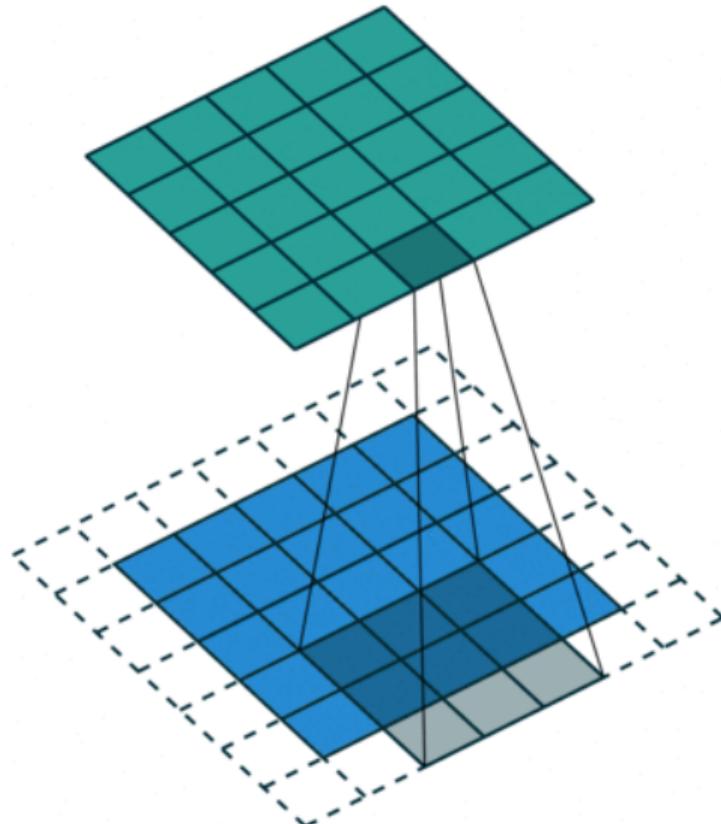
L'opérateur de convolution



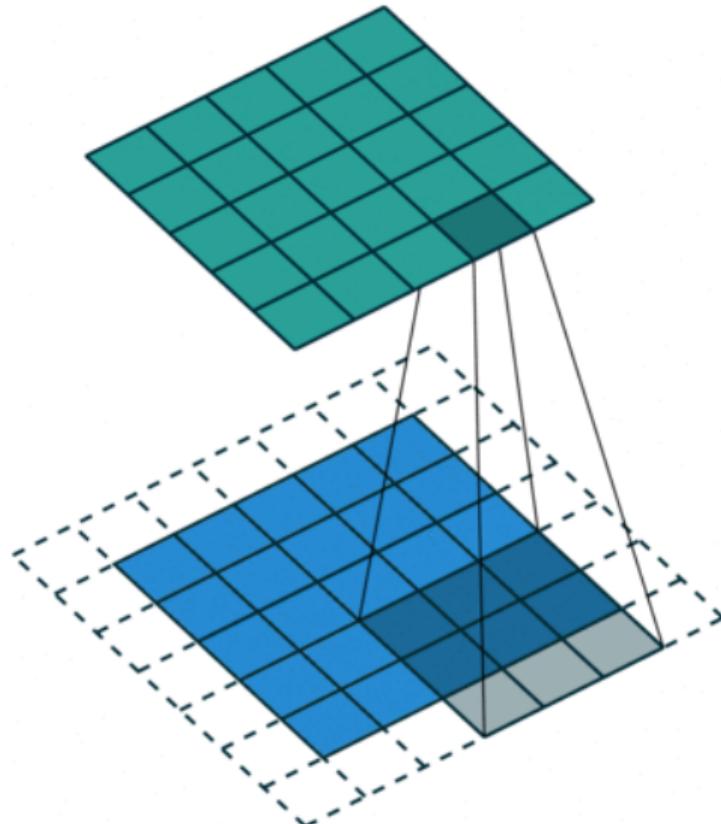
L'opérateur de convolution



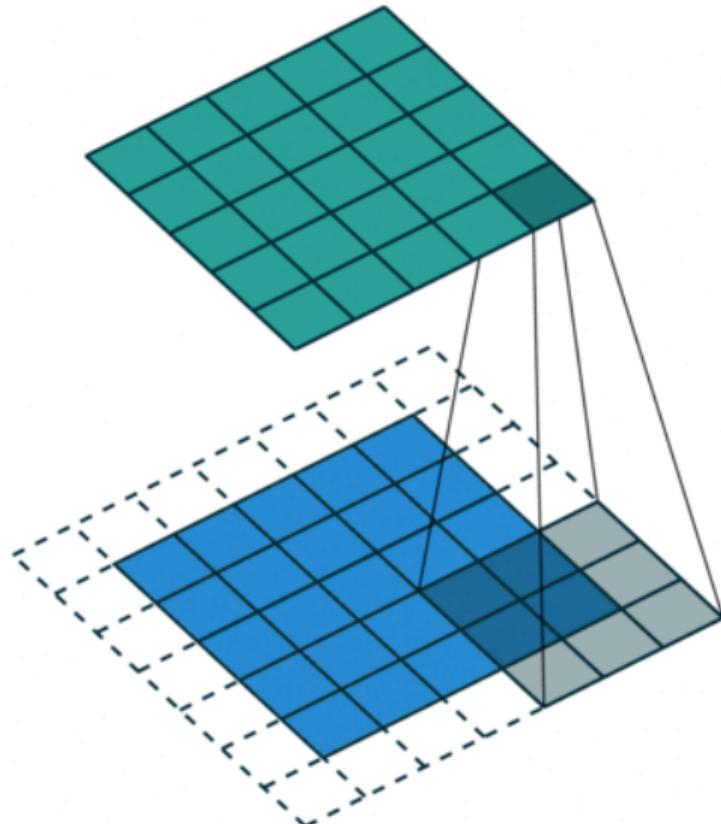
L'opérateur de convolution



L'opérateur de convolution



L'opérateur de convolution



L'opérateur de convolution

- ▶ La taille de noyau K est variable.
- ▶ En filtrage, on prend des noyaux carrés, de taille impaire. Ça permet donc d'être centré sur un pixel.
- ▶ Avec un *padding* de 1, et un glissement de taille un 1, on peut conserver l'image de la même taille.

Filtration d'image

- ▶ Depuis la numérisation de la photo, il est commun d'appliquer des filtres sur des images.
- ▶ Un filtre est un noyau (kernel) (de taille impaire afin d'être centré sur un pixel) qui permet de modifier l'image d'une certaine manière.
- ▶ Le filtre suivant:

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

fera une moyenne d'un pixel et de ces 8 voisins le plus proche, ce qui brouillera l'image.

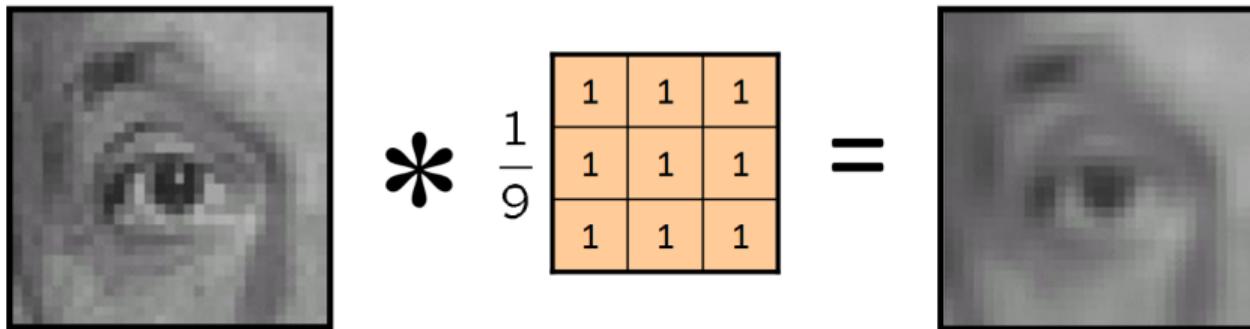
Filtration d'image

$$\begin{array}{c} \text{Image} \\ * \\ \text{Filter} \\ = \\ \text{Result} \end{array}$$

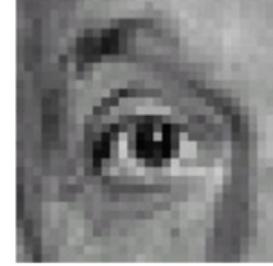
The diagram illustrates the convolution process. On the left is a grayscale image of an eye. In the center is a 3x3 filter kernel with a value of 1 in the middle position and 0s elsewhere. To the right is the result of applying this filter to the image, which is identical to the original image.

0	0	0
0	1	0
0	0	0

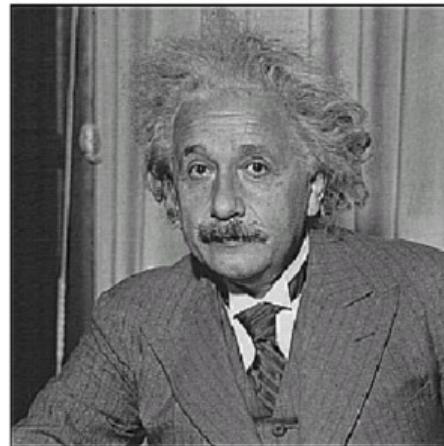
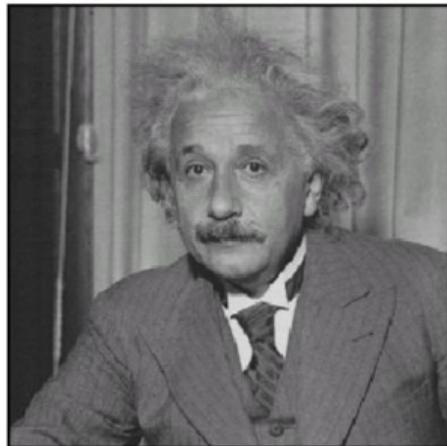
Filtration d'image

$$\text{Image} * \frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} = \text{Filtered Image}$$


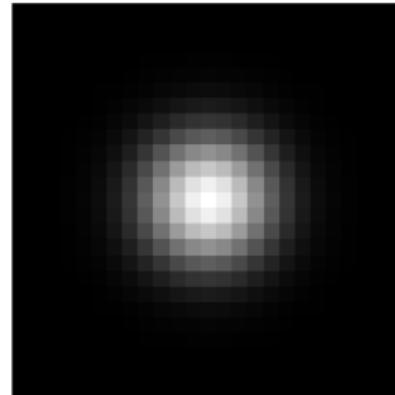
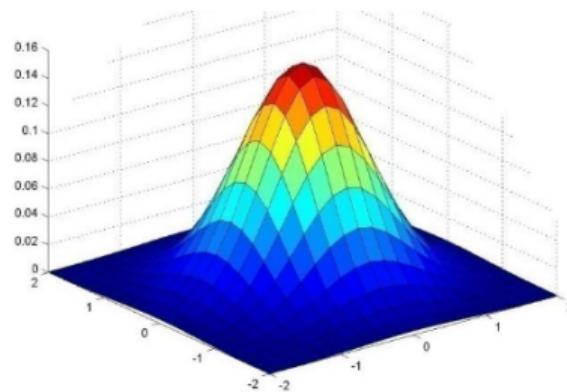
Filtration d'image


$$* \left(\begin{array}{ccc} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{array} - \frac{1}{9} \begin{array}{ccc} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{array} \right) = \text{blurred image}$$


Filtration d'image

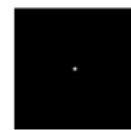
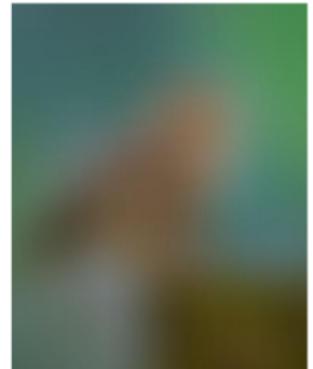
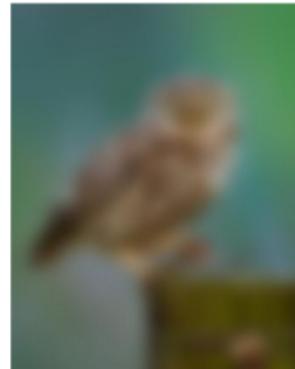


Filtration d'image

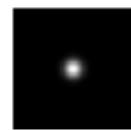


$$G_{\sigma} = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

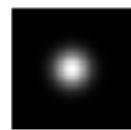
Filtration d'image



$\sigma = 1$ pixel



$\sigma = 5$ pixels



$\sigma = 10$ pixels

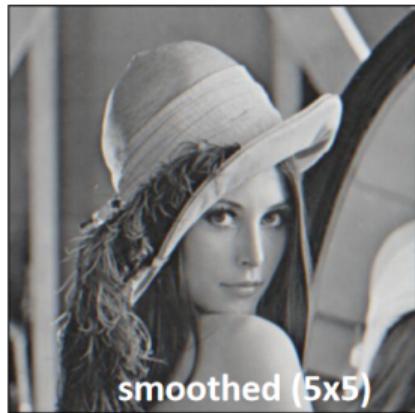


$\sigma = 30$ pixels

Filtration d'image



-



=

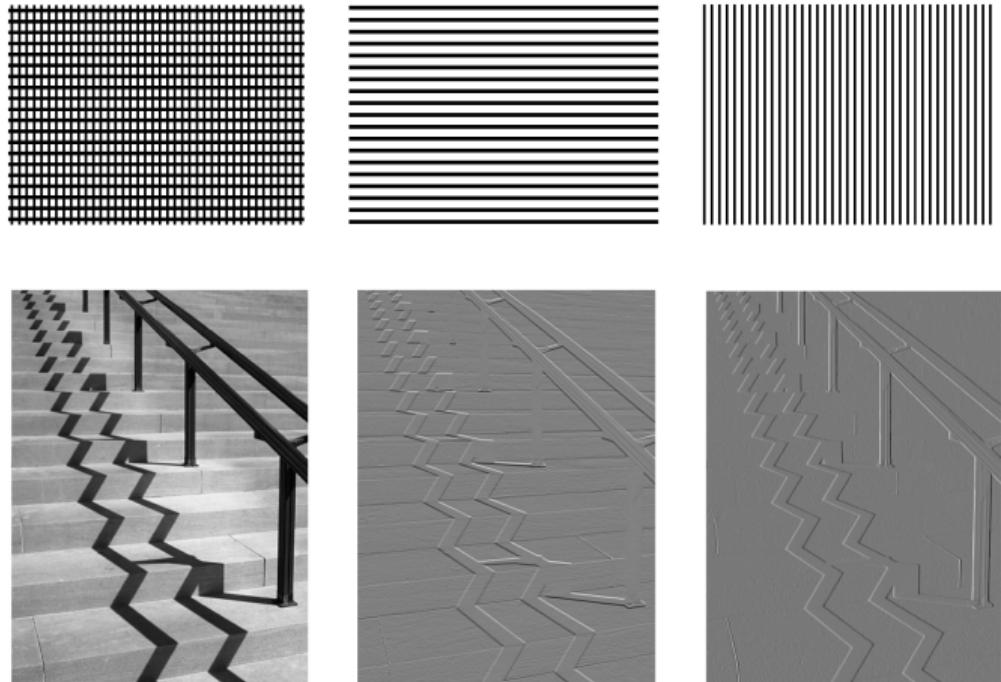


Filtration d'image

On parle d'**opérateur de Sobel**,

horizontal $\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$

vertical $\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$



Filtre d'image

- ▶ En résumé, avec de bons filtres on peut faire ressortir toutes sortes de propriétés des images.
- ▶ En supposant que certaines de ces propriétés sont des caractéristiques importantes pour la classification, on veut donc apprendre des filtres.
- ▶ Les réseaux des neurones convolutif visent donc l'apprentissage de noyaux/filtres.

Le filtre suivant:

$$\begin{bmatrix} \omega_{1,1} & \omega_{1,2} & \omega_{1,3} \\ \omega_{2,1} & \omega_{2,2} & \omega_{2,3} \\ \omega_{3,1} & \omega_{3,2} & \omega_{3,3} \end{bmatrix}$$

est maintenant un noyau de taille 3 par 3 où les ω sont des paramètres à apprendre à l'aide de données!

Réseau de neurones convolutif

- ▶ Entre la première couche et la seconde, on apprend en parallèle plusieurs filtres.
- ▶ Certaines configurations vont réduire la taille de l'image (grand *stride* par exemple).
- ▶ On peut ensuite répéter le processus sur ces petites images.
- ▶ Lorsque les images sont très petites, on les vectorise et finit le modèle comme avant, avec des couches complètement connectées.

Réseau de neurones convolutif

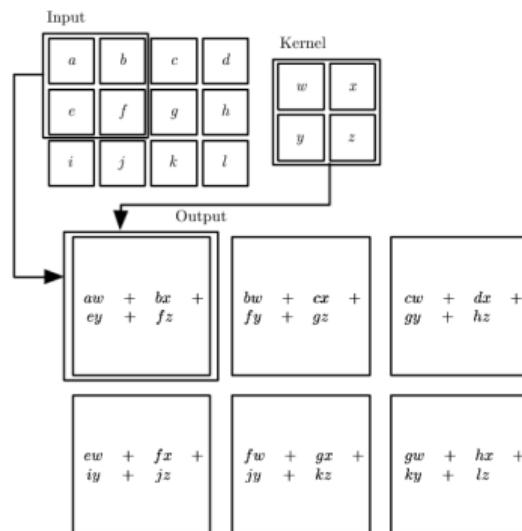
- ▶ On emploie souvent une couche de pooling entre les étapes de convolution.
- ▶ Le max pooling est un filtre qui extrait la valeur maximale de ces entrés.

Max pool

$$\begin{bmatrix} 1 & 2 & 5 & 3 \\ 3 & 0 & 1 & 2 \\ 2 & 1 & 3 & 4 \\ 1 & 1 & 2 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 5 \\ 2 & 4 \end{bmatrix}$$

Réseau de neurones convolutif

- ▶ La couche pooling, peut être un peu perçu comme l'application de la fonction non linéaire dans un réseau standard.
- ▶ C'est une fonction différentiable, mais non paramétrique; on n'apprend rien ici.
- ▶ Tout ce qu'on apprend dans ce modèle sont les éléments des filtres, qui en sorte encore une fois une fois une combinaison linéaire:



Réseau de neurones convolutif

Mais pourquoi ça fonctionne ?

- ▶ On a besoin de moins de paramètres par combinaison linéaire puisqu'on ne prend qu'une combinaison de pixel voisin.
- ▶ On peut donc apprendre BEAUCOUP plus de filtres.
- ▶ On apprend aussi des *patterns* qui ont une importance à travers l'image au complet.
- ▶ Comme on apprend un filtre pour plusieurs endroits sur l'image ET plusieurs images, on décuple le nombre de données d'entraînement.

Réseau de neurones convolutif

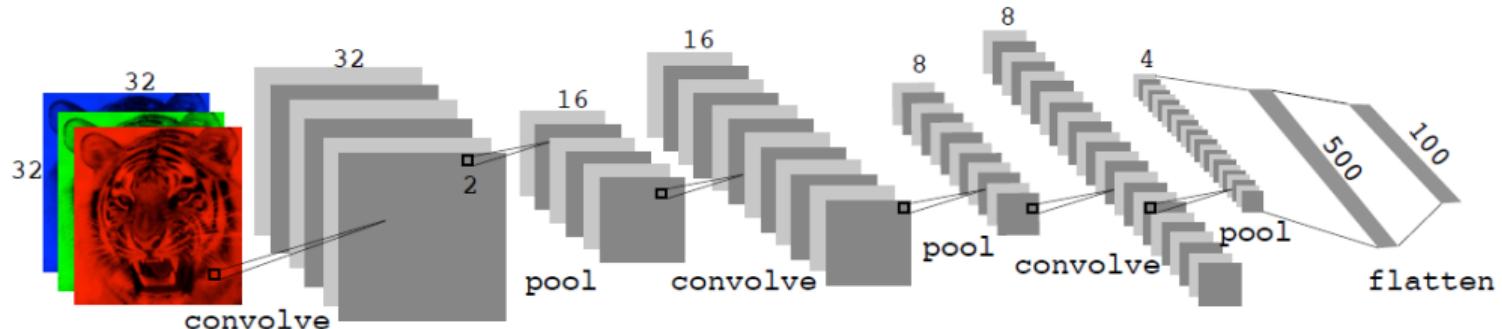
- ▶ On apprend plusieurs filtres:



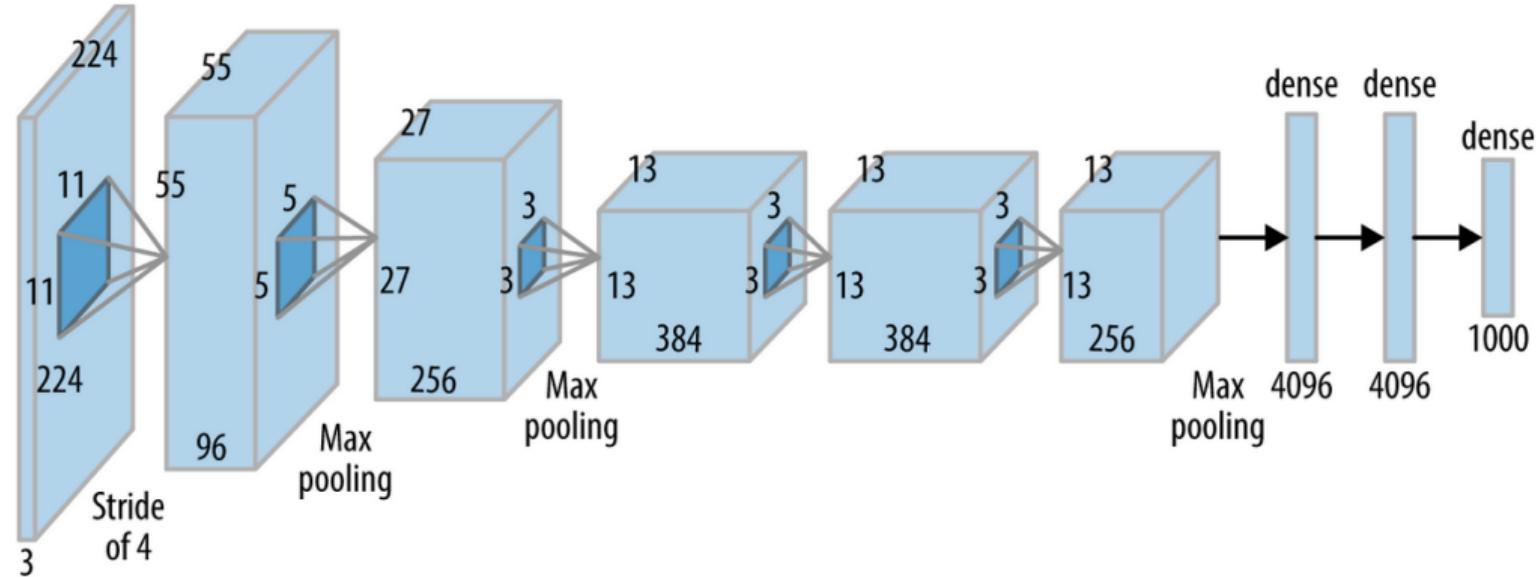
- ▶ On pool les images filtrées.
- ▶ Le stride est responsable de réduire la taille de l'image.
- ▶ Éventuellement on vectorise ce qui reste et conclut avec un réseau standard.

Réseau de neurones convolutif

La taille des images de la couche suivante est : $\frac{W - K + 2P}{S} + 1$ où W est la taille d'entrée, K la taille du noyau, P la padding et S le stride.



Réseau de neurones convolutif



Pour le prochain cours:

- ▶ Lecture: Deep Lerning Chapitre 9
- ▶ Ressource CNN: Belle animations, Ressources PyTorch, Crédit image
- ▶ Préparation: 12.1, 12.4.1 et 9.1 (PRML)
- ▶ Exercices: 10.10.4
- ▶ Lab : 10.9.3 (Difficile)
- ▶ Extra Lab : Entrainer un GAN! (Difficile)

Références