Agrégation, Convolution, Neurones convolutifs

Objectifs [a] *:*

- *Introduction des opérations d'agrégation et de convolution matricielle*
- *Application de ces opérations à des images*
- *Introduction des réseaux de neurones convolutifs*

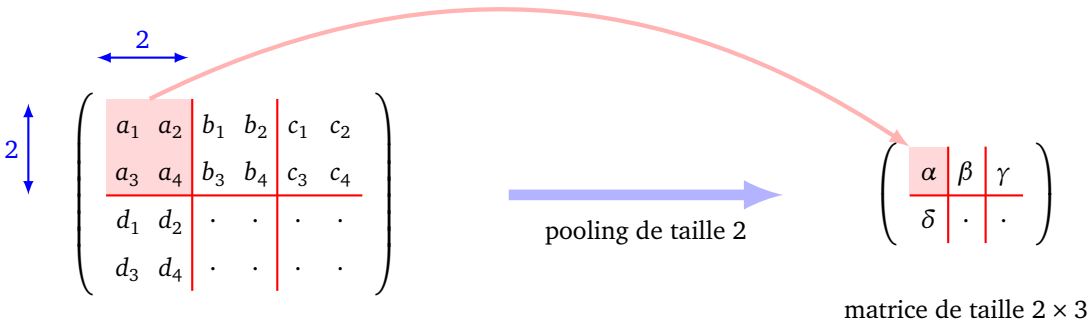
a. Version 2023 adapté BUT 3 Informatique (Orléans), inspiré du livre d'Arnaud Bodin et François Recher

1. Opérations sur les Matrices

Nous voyons dans ce paragraphe, la notion d'agrégation connue sous le terme anglais "pooling" et la notion de convolution pour des matrices.

1.1. Pooling (agrégation)

Le *pooling* (regroupement de termes) consiste à transformer une matrice en une matrice plus petite tout en essayant d'en garder les caractéristiques principales.
Un ***pooling*** de taille k transforme une matrice de taille $n \times p$ en une matrice de taille k fois plus petite, c'est-à-dire de taille $n//k \times p//k$. Une sous-matrice de taille $k \times k$ de la matrice de départ produit un seul coefficient de la matrice d'arrivée.

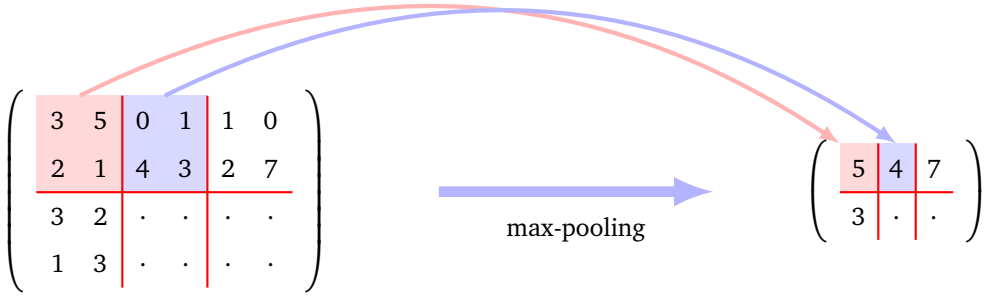


matrice de taille 4×6

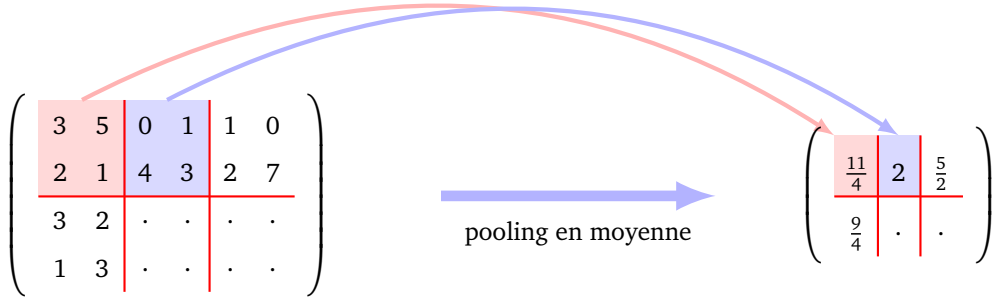
matrice de taille 2×3

On distingue deux types de pooling.

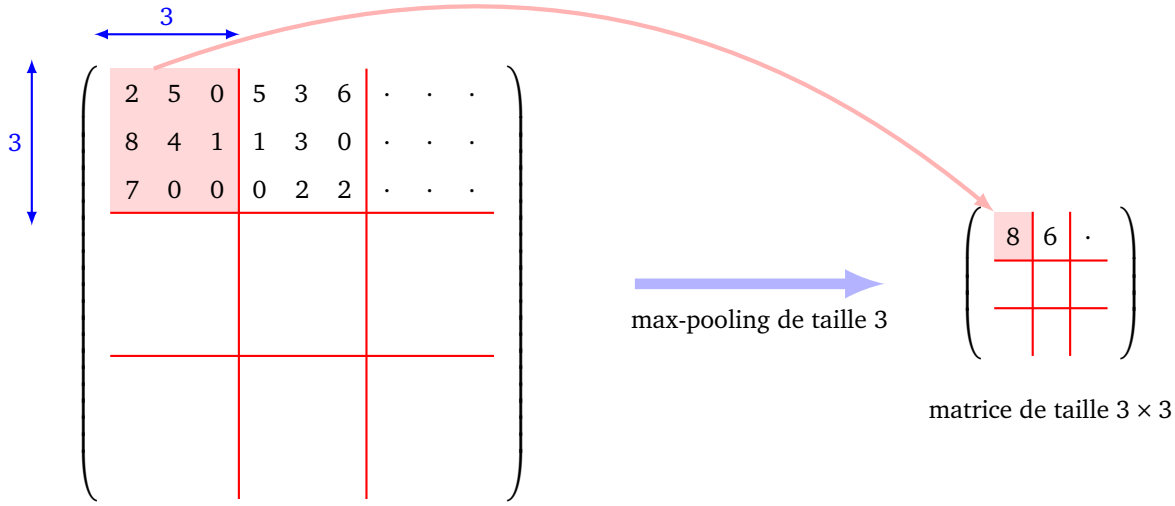
- Le ***max-pooling*** de taille k consiste à retenir le maximum de chaque sous-matrice de taille $k \times k$:



- Le ***pooling en moyenne*** de taille k (*average pooling*) consiste à retenir la moyenne des termes de chaque sous-matrice de taille $k \times k$:



Ci-dessous, voici le début d'un max-pooling de taille 3. La matrice de départ de taille 9×9 est transformée en une matrice de taille 3×3.



matrice de taille 9×9

matrice de taille 3×3

Le max-pooling, qui ne retient que la valeur la plus élevée par sous-matrice, permet de détecter la présence d'une caractéristique (par exemple un pixel blanc dans une image noire). Tandis que le pooling en moyenne prend en compte tous les termes de chaque sous-matrice (par exemple avec 4 pixels d'une image de ciel, on retient la couleur moyenne).

1.2. Essais sur une image

Dans le code ci dessous on effectue quatre "pooling" avec des tailles de fenêtres différentes.

```
import numpy as np
from skimage import data
```

```
from skimage.measure import block_reduce
import matplotlib.pyplot as plt


# Charge une image (par exemple, un cheval de la bibliothèque de données skimage)
image = data.horse()

# Affiche le nombre de pixels de l'image originale
print(f"Nombre de pixels de l'image originale : {horse_image.size}")

# Affiche l'image d'origine
plt.imshow(image, cmap=plt.get_cmap('gray'))
plt.title('Image originale')
plt.show()

# Applique le max pooling avec différentes tailles de fenêtre
pooling_sizes = [2, 3, 4,10]

for size in pooling_sizes:
    # Utilise la fonction block_reduce de skimage pour effectuer le max pooling
    pooled_image = block_reduce(image, (size, size), np.max)

    # Affiche le nombre de pixels de l'image après max pooling
    print(f"Nombre de pixels après Max Pooling {size}x{size} : {pooled_image.size}")

    # Affiche l'image résultante avec le titre correspondant
    plt.imshow(pooled_image, cmap=plt.get_cmap('gray'))
    plt.title(f'Max Pooling {size}x{size}')
    plt.show()
```

Exercice 1.
Dans le code précédent : changer d'image en remplaçant par exemple horse() par camera(). Aussi transformer ces max-pooling en mean-pooling.

2. Convolution matricielle

2.1. Définition

La convolution en deux dimensions est une opération qui :

- à partir d'une matrice d'entrée notée A,
- et d'une matrice d'un motif noté M,

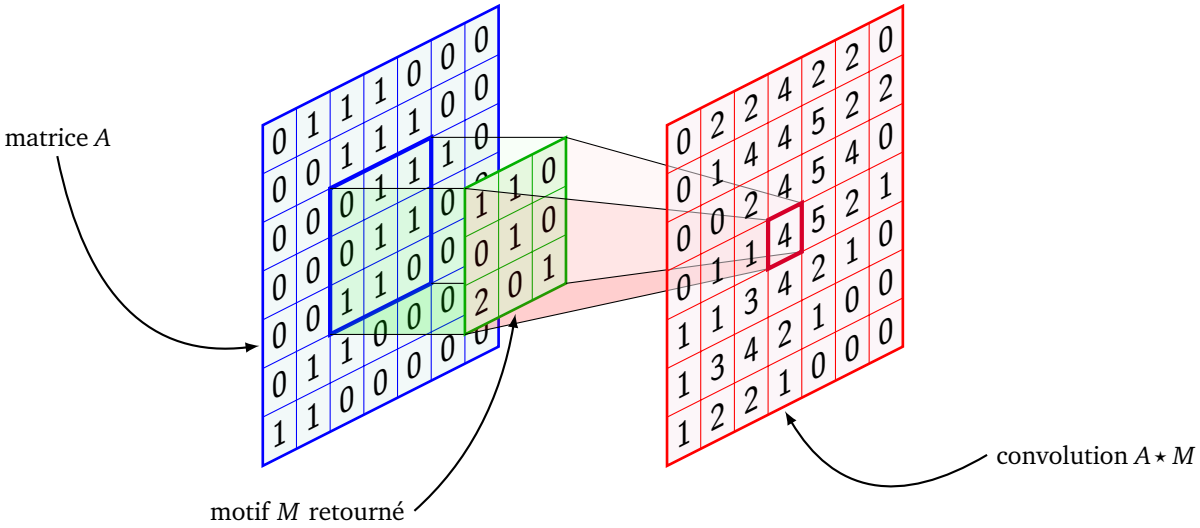
associe une matrice de sortie $A \star M$.
Tout d'abord, il faut retourner la matrice M :

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \qquad \begin{pmatrix} m_{33} & m_{32} & m_{31} \\ m_{23} & m_{22} & m_{21} \\ m_{13} & m_{12} & m_{11} \end{pmatrix}$$

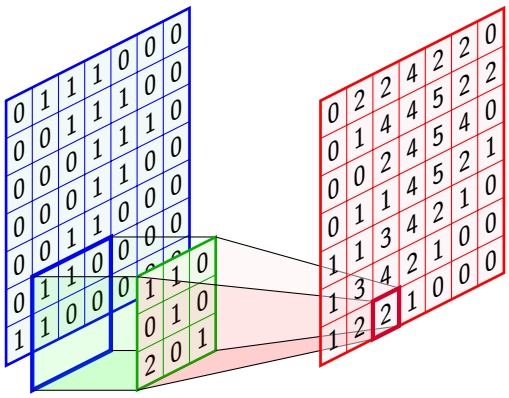
Motif de taille 3×2 **Motif retourné**

Le calcul de $A \star M$ s'effectue coefficient par coefficient :

- on centre le motif retourné sur la position du coefficient à calculer,
- on multiplie chaque coefficient de A par le coefficient du motif retourné en face (quitte à ajouter des zéros virtuels sur les bords de A),
- la somme de ces produits donne un coefficient de $A \star M$.



Remarque dans notre cas $M = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$. Voici un autre schéma pour le calcul d'un autre coefficient. Pour ce calcul, on rajoute des zéros virtuels autour de la matrice A.



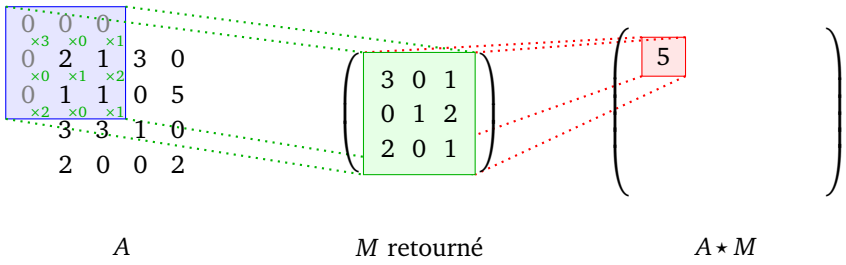
Exemple.
Calculons la convolution $A \star M$ définie par :

$$\begin{pmatrix} 2 & 1 & 3 & 0 \\ 1 & 1 & 0 & 5 \\ 3 & 3 & 1 & 0 \\ 2 & 0 & 0 & 2 \end{pmatrix} \star \begin{pmatrix} 1 & 0 & 2 \\ 2 & 1 & 0 \\ 1 & 0 & 3 \end{pmatrix}.$$

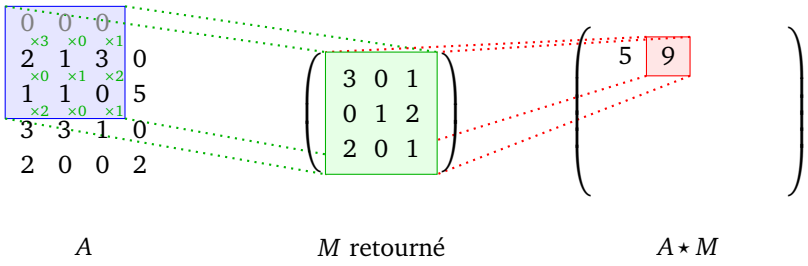
On commence par retourner M. Pour calculer le premier coefficient de $A \star M$, on centre le motif sur le premier coefficient de A, puis on rajoute des zéros virtuels à gauche et en haut (cette opération s'appelle padding ou rembourrage en français). Ensuite on calcule les produits des coefficients de la matrice M retournée avec les coefficients de A correspondants, et on les additionne :

$$0\times 3 + 0\times 0 + 0\times 1 + 0\times 0 + 2\times 1 + 1\times 2 + 0\times 2 + 1\times 0 + 1\times 1.$$

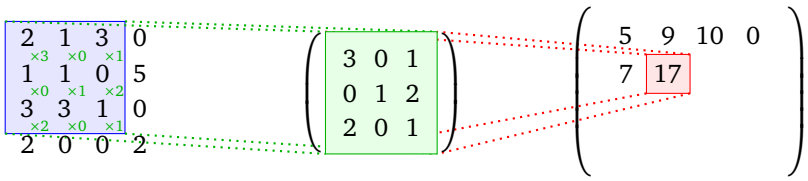
Cette somme vaut 5, c'est le premier coefficient de $A \star M$.



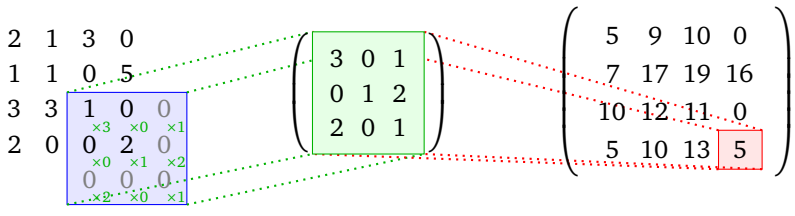
On continue avec le second coefficient.



Et ainsi de suite :

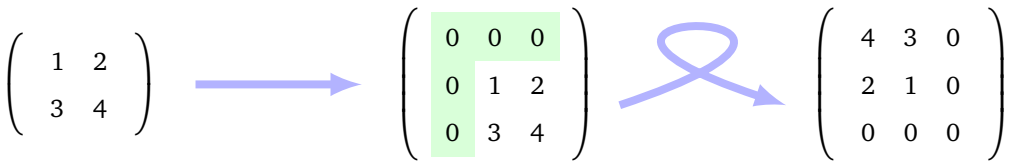


Jusqu'à calculer entièrement la matrice $A \star M$.



Remarque.

- Il ne faut pas confondre le fait de retourner la matrice avec une opération différente qui s'appelle la transposition.
- Dans la plupart de nos situations, le motif sera une matrice de taille 3×3. Par contre la matrice A pourra être de très grande taille (par exemple une matrice 600×400, codant une image).
- Cependant, si la matrice du motif possède une dimension paire on rajoute des zéros virtuels à gauche ou en haut (avant de la retourner).



Motif de taille 2×2 **Motif augmenté** **Motif retourné**

- Contrairement au pooling, les motifs disposés pour des calculs d'éléments voisins se superposent. Sur le dessin ci-dessous le motif est d'abord centré sur le coefficient 8, puis sur le coefficient 9.

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 & 17 & 18 \end{pmatrix}$$

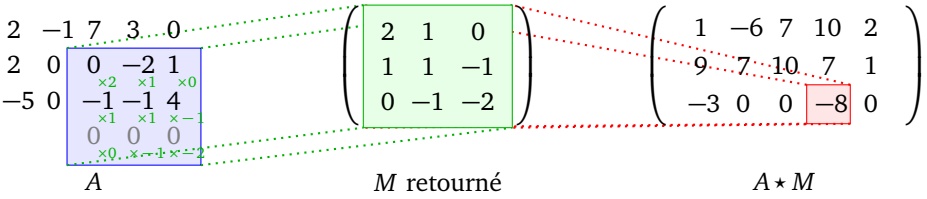
- Par définition, il faut retourner la matrice M avant de calculer les produits. Cela complique l'usage mais est cohérent avec la définition donnée pour la dimension 1 et cela permettra d'avoir de bonnes propriétés mathématiques (voir le chapitre « Convolution avec Python »).

2.2. Exemples simples

Voici un autre exemple de calcul :

$$\begin{pmatrix} 2 & -1 & 7 & 3 & 0 \\ 2 & 0 & 0 & -2 & 1 \\ -5 & 0 & -1 & -1 & 4 \end{pmatrix} \star \begin{pmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{pmatrix}.$$

Il ne faut pas oublier de retourner M avant de commencer !



A M retourné $A \star M$

Exercice 2.

Considérons la matrice : A$=\begin{pmatrix} 2 & 0 & -1 \\ 2 & 1 & 2 \\ 1 & 0 & 3 \end{pmatrix}$ et le motif M$=\begin{pmatrix} 0 & 0 & 1 \\ 2 & 0 & -1 \\ 1 & 2 & 1 \end{pmatrix}$

Calculer $A \star M$, *sans oublier de retourner* M *!*

Exemple particulier : translation des coefficients de la matrice d'entrée. Une convolution par la matrice

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

correspond à une translation des coefficients vers le bas. Par exemple :

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \star \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}.$$

De même pour une translation des coefficients vers la droite.

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \star \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 5 & 6 & 7 \\ 0 & 9 & 10 & 11 \\ 0 & 13 & 14 & 15 \end{pmatrix}.$$

Exemple particulier : moyennisation locale des coefficients de la matrice d'entrée. On effectue une moyenne locale des coefficients à l'aide du motif :

$$\frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}.$$

Par exemple :

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{pmatrix} \star \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \simeq \begin{pmatrix} 1.77 & 3 & 3.66 & 4.33 & 3.11 \\ 4.33 & 7 & 8 & 9 & 6.33 \\ 7.66 & 12 & 13 & 14 & 9.66 \\ 11 & 17 & 18 & 19 & 13 \\ 8.44 & 13 & 13.66 & 14.33 & 9.77 \end{pmatrix}.$$

On remarque des phénomènes de bords dus à l'ajout de zéros virtuels.

2.3. Application à l'image

Dans ce paragraphe nous fabriquons d'abord "à la main" des images simples. On leur applique des filtres c'est à dire que vous ferez le produit de convolutions de cette matrice image avec des matrices spécifiques. On commence par importer quelques librairies, notamment `signal` qui permet le calcul de convolution.

```
import numpy as np
from skimage import data
import matplotlib as plt
from scipy import signal
from matplotlib.pyplot import imshow, get_cmap
import matplotlib.pyplot as plt
```

La fonction suivante permet d'afficher des images côte à côte (dans le but de voir apparaître les transformations dues à des opérations de convolution).

```
def displayTwoBaWImages(img1, img2):
  _, axes = plt.subplots(ncols=2)
  axes[0].imshow(img1, cmap=plt.get_cmap('gray'))
  axes[1].imshow(img2, cmap=plt.get_cmap('gray'))
```

Nous créons maintenant une image extrêmement simple et nous l'affichons :

```
image = np.array([[0,0,0,0,0],
                  [0,0,1,0,0],
                  [0,1,1,1,0],
                  [0,0,1,0,0],
                  [0,0,0,0,0]])
cmp = get_cmap('gray')
imshow(image, cmap=cmp, vmin=0, vmax=1)
```

Nous rappelons en effet qu'une image noire et blanc peut être représentée par une matrice constituée de 0 (noir) et de 1 (blanc). Pour des nuances de gris il suffit de remplacer le 0 ou le 1 par une valeur entre 0 et 1.

Exercice 3.
Fabriquer une image de 8 par 8 pixels comprenant un carré gris à l'intérieur d'un carré noir.

Nous créons maintenant une matrice de convolution (ou filtre, ou motif ...) noté `kernel` ici et nous l'affichons :

```
kernel = np.ones((3,3), np.float32)/2
print(kernel)
```

Nous faisons maintenant le produit de convolution (en utilisant la fonction `signal.convolve2d`). Pour l'afficher en tant qu'image (à côté de l'image d'origine), le code est :

```
imgconvol = signal.convolve2d(image,
                              kernel,
                              mode='same',
                              boundary='fill',
                              fillvalue=0)
displayTwoBaWImages(image, imgconvol)
```

Que retourne `print(imgconvol)` ?

Pour les exercices suivants on utilisera le code de l'exercice 3.

Exercice 4.
Appliquer le filtre (kernel) suivant à l'image `image=data.horse()`

```
[[0,1,0],
 [1,-4,1],
 [0,1,0]]
```

Que remarquez vous ?

Exercice 5.
On va maintenant appliquer une convolution à l'image de camera-man (: `image=data.camera()`). Pour cela on commence par re-normaliser cette image :
`image_normalized = (image - np.min(image)) / (np.max(image) - np.min(image))`
On utilisera ensuite le filtre suivant, que l'on appliquera à `image_normalized`.

```
[[,0,-1,0,],
[-1,5,-1],
[0,-1,0]]
```

Que remarquez vous ?

Exercice 6.
même question avec `image=data.checkerboard()` et le filtre :

```
[[1,1,1],
 [1,1,1],
 [1,1,1]]/9
```

Exercice 7.
même question avec `image=data.checkerboard()` et le filtre :

```
[[-1,2,-1],
 [-1,2,-1],
 [-1,2,-1]]
```

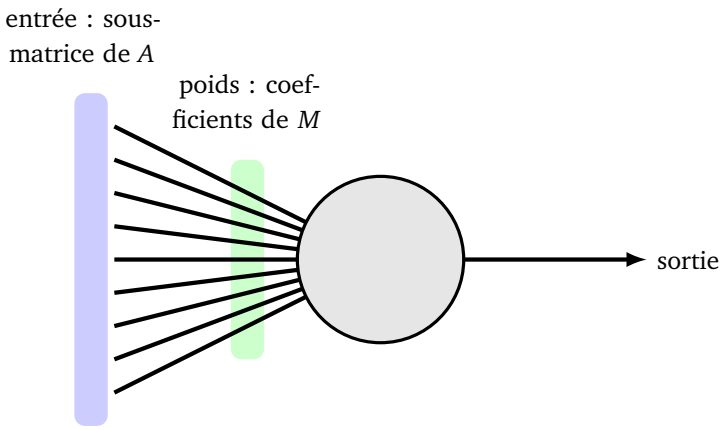
puis

```
[[-1,-1,-1],
 [2,2,2],
 [-1,-1,-1]].
```

3. Réseaux convolutifs

3.1. Neurone de convolution

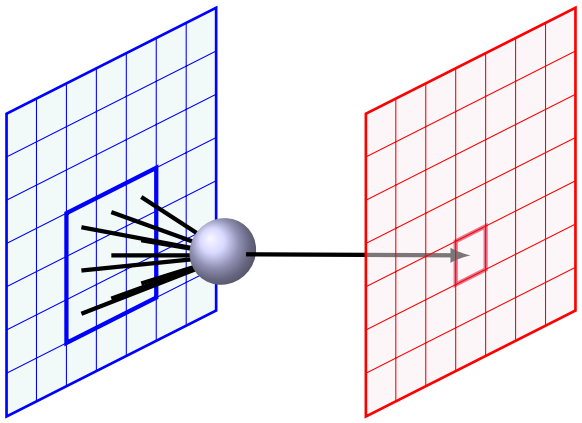
À l'aide de la convolution, nous allons définir un nouveau type de couche de neurones : une « couche de convolution ». Tout d'abord un ***neurone de convolution*** de taille 3×3 est un neurone classique ayant 9 entrées (pour l'instant la fonction d'activation est l'identité).



Les poids du neurone correspondent aux coefficients d'une matrice de convolution :

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix}.$$

Imaginons que l'entrée soit une image ou un tableau de dimension 2, pour nous ce sera une matrice A. Alors un neurone de convolution est relié à une sous-matrice 3×3 de A.



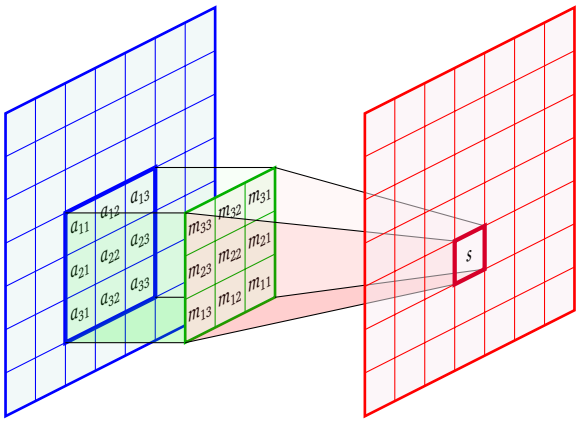
Ce neurone agit comme un élément de convolution. Par exemple si la sous-matrice de A est notée

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

alors le neurone produit la sortie :

$$s = a_{11} \cdot m_{33} + a_{12} \cdot m_{32} + a_{13} \cdot m_{31} + a_{21} \cdot m_{23} + a_{22} \cdot m_{22} + a_{23} \cdot m_{21} + a_{31} \cdot m_{13} + a_{32} \cdot m_{12} + a_{33} \cdot m_{11}$$

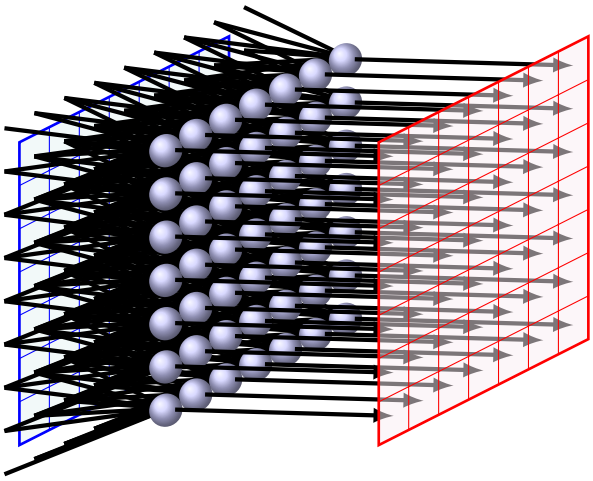
N'oublier pas que dans le produit de convolution, on commence par retourner la matrice M.



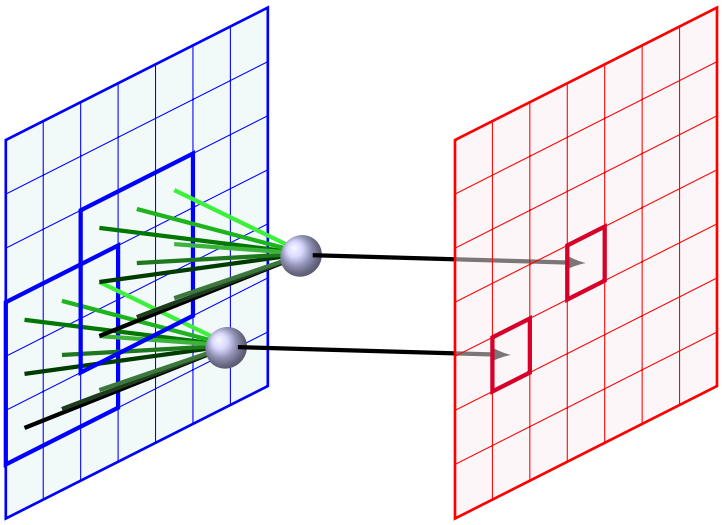
On pourrait aussi composer par une fonction d'activation au niveau du neurone, mais on le fera plus tard à l'aide d'une « couche d'activation ». Plus généralement un neurone de convolution de taille $p \times q$ possède pq arêtes et donc pq poids à définir ou à calculer.

3.2. Couche de convolution

Considérons une entrée A représentée par une matrice de taille $n \times p$.
Une ***couche de convolution*** (pour un seul motif) est la donnée d'une matrice M appelé ***motif*** (par exemple de taille 3×3) et qui renvoie en sortie les coefficients de la matrice $A \star M$.



Pour une entrée de taille $n \times p$, il y a donc np neurones de convolutions, chacun ayant 9 arêtes (car le motif est de taille 3×3). Il est très important de comprendre que les poids sont communs à tous les neurones (ce sont les coefficients de M). Ainsi, pour une couche de convolution, il y a seulement 9 poids à déterminer pour définir la couche de convolution (bien que le nombre de neurones puisse être très grand).
Sur le dessin ci-dessous on ne montre que deux neurones. Noter deux choses importantes : (a) deux neurones différents peuvent avoir certaines entrées communes (sur le dessin les deux carrés 3×3 s'intersectent) et (b) deux neurones différents ont les mêmes poids (sur le dessin les arêtes entrantes ayant la même couleur, ont les mêmes poids).



Remarque.

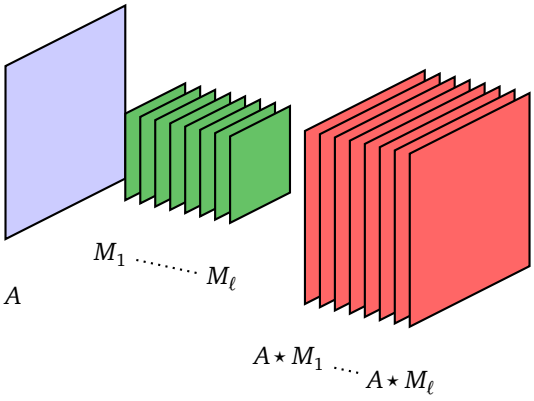
- Terminologie : le motif M s'appelle aussi le ***noyau*** (*kernel*), le ***filtre*** ou encore le ***masque***.
- Le motif peut être d'une taille différente de la taille 3×3 considérée dans les exemples qui est cependant la plus courante.
- Il existe également des variantes avec biais ou fonction d'activation.
- Combinatoire : dans le cas d'une couche complètement connectée, le nombre de poids à calculer serait énorme. En effet, une couche de np neurones complètement connectée à une entrée de taille $n \times p$ amènerait à calculer $(np)^2$ poids. Pour $n = p = 100$, cela fait 10 000 neurones et 100 000 000 poids. Pour rappel, notre couche de convolution est définie par 9 poids (quels que soient n et p).
- D'un point de vue mathématique, une couche de convolution associée au motif M est l'application :

$$\begin{array}{rccc} F : & \mathcal{M}_{n,p} & \longrightarrow & \mathcal{M}_{n,p} \\ & A & \longmapsto & A \star M \end{array}$$

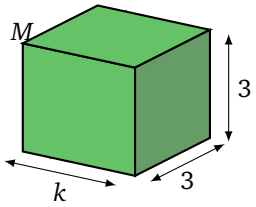
où $\mathcal{M}_{n,p}$ est l'ensemble des matrices de taille $n \times p$.
Si on transforme une matrice en un (grand) vecteur, alors on obtient une application $F : \mathbb{R}^{np} \to \mathbb{R}^{np}$.

3.3. Différentes couches d'un réseau de neurones

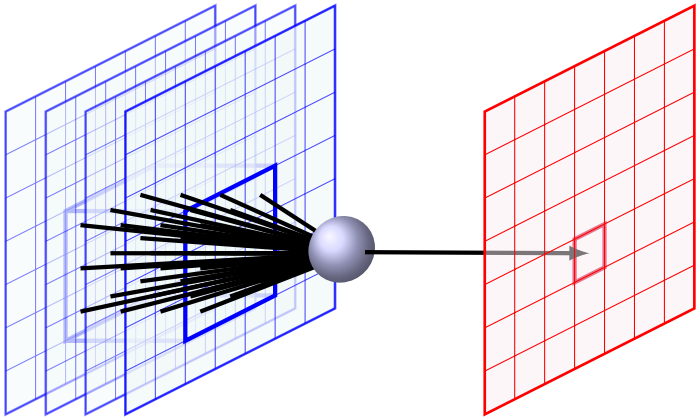
Les couches de convolution sont au cœur des réseaux de neurones modernes. Voici un petit survol des types de couches qui seront mises en pratique dans le chapitre « Reconnaissance d'images par un CNN ».
Plusieurs filtres. On verra dans la pratique qu'une couche de convolution est définie avec plusieurs motifs, c'est-à-dire un nombre ℓ de matrices $M_1, M_2, \ldots, M_\ell$. Ainsi pour une entrée A de taille $n \times p$, une couche de convolution à ℓ motifs renvoie une sortie de taille $n \times p \times \ell$, correspondant à $(A \star M_1, A \star M_2, \ldots, A \star M_\ell)$.



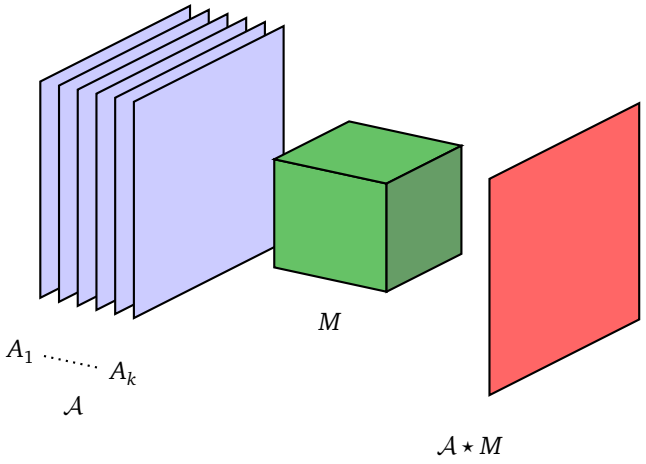
Convolution à partir de plusieurs canaux. Pour traiter une sortie de taille $n \times p \times k$ du type précédent, qui va être l'entrée de la couche suivante on doit définir une convolution ayant plusieurs canaux en entrée. Si les entrées sont les $A_1, A_2, \ldots, A_k$ alors un motif M associé à cette entrée de profondeur k est un 3-tenseur de taille $(3,3,k)$, c'est-à-dire un tableau à 3 dimensions de la forme $3 \times 3 \times k$ (on renvoie au chapitre « Tenseurs » pour la définition).



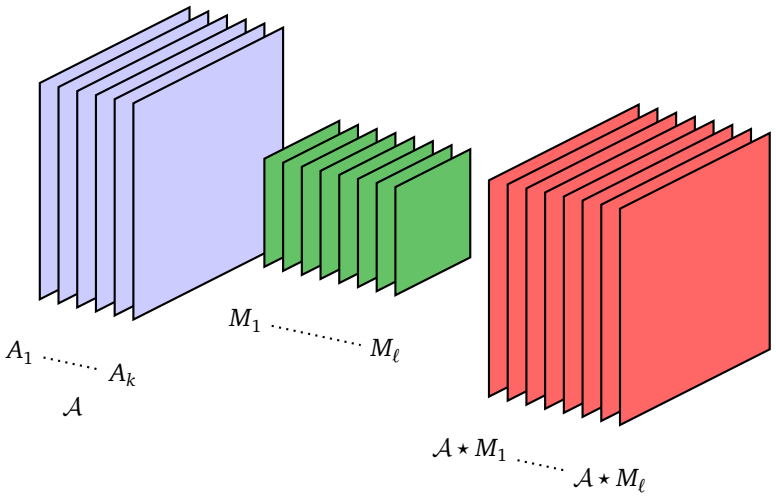
Ce motif M correspond donc à une couche de neurones où chaque neurone possède $3 \times 3 \times k$ arêtes. Chaque neurone est connecté localement à une zone 3×3 dans un plan, mais ceci sur toute la profondeur des k entrées. Le calcul de la sortie du neurone se fait en réalisant la somme de $3 \times 3 \times k$ termes. Pour réaliser une couche de neurones, la zone 3×3 se déplace dans le plan, mais s'étend toujours sur toute la profondeur.



On note $\mathcal{A} = (A_1, A_2, \ldots, A_k)$ et $\mathcal{A} \star M$ le produit de convolution. Si l'entrée $\mathcal{A}$ est de taille (n,p,k) alors la sortie $\mathcal{A} \star M$ associée au motif M est de taille (n,p).



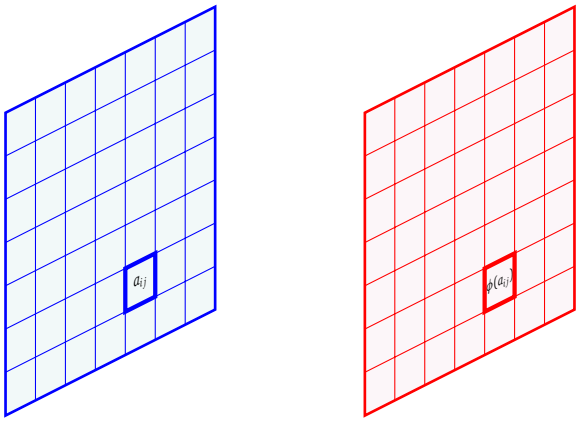
Convolution à plusieurs filtres à partir de plusieurs canaux. C'est le cas général dans la pratique. Une entrée donnée par plusieurs canaux $\mathcal{A} = (A_1, A_2, \ldots, A_k)$, associée à des motifs $M_1, M_2, \ldots, M_\ell$ (qui sont donc chacun des 3-tenseurs de taille $(3,3,k)$) produit une sortie de taille $n \times p \times \ell$, correspondant à $(\mathcal{A} \star M_1, \mathcal{A} \star M_2, \ldots, \mathcal{A} \star M_\ell)$. Si l'entrée $\mathcal{A}$ est de taille (n,p,k) alors la sortie est de taille (n,p,ℓ).



Noter que sur cette figure chaque motif M_i est représenté par carré 3×3, alors qu'en fait chacun devrait être une boîte en 3 dimensions de taille $3 \times 3 \times k$.

Activation. On peut composer par une fonction d'activation ϕ directement dans le neurone ou bien dans une couche à part (ce qui est équivalent). Comme d'habitude, la fonction d'activation est la même pour tous les neurones d'une couche. Ainsi une ***couche d'activation*** pour la fonction d'activation ϕ, transforme une entrée

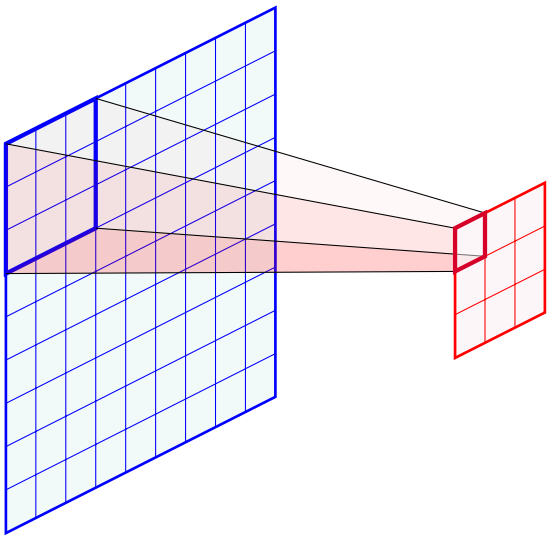
$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1p} \\ a_{21} & a_{22} & \dots & a_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{np} \end{pmatrix} \quad \text{en} \quad \begin{pmatrix} \phi(a_{11}) & \phi(a_{12}) & \dots & \phi(a_{1p}) \\ \phi(a_{21}) & \phi(a_{22}) & \dots & \phi(a_{2p}) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(a_{n1}) & \phi(a_{n2}) & \dots & \phi(a_{np}) \end{pmatrix}.$$



matrice A activation ϕ

Les fonctions d'activation sont les mêmes que celles rencontrées auparavant : ReLU, σ, tanh...

Pooling. Une couche de pooling de taille k transforme une entrée de taille $n \times p$ en une sortie de taille $n//k \times p//k$. Nous renvoyons à la première section de ce chapitre pour le max-pooling ou le pooling en moyenne. Cette opération permet de réduire les données d'un facteur k^2. Par exemple, une entrée de taille 100×80 (8000 données) avec un pooling de taille 4 fournit une sortie de taille 25×20 (500 données). Ci-dessous un pooling de taille 3 transforme une matrice 9×9 en une matrice 3×3.



Dropout. Le ***dropout*** (décrochage ou abandon en français) est une technique pour améliorer l'apprentissage d'un réseau et en particulier pour prévenir le sur-apprentissage. L'idée est de désactiver certains neurones d'une couche lors des étapes de l'apprentissage. Ces neurones sont choisis au hasard et sont désactivés temporairement pour une itération (par exemple on peut choisir à chaque itération de désactiver un neurone avec une probabilité $\frac{1}{2}$). Cela signifie que l'on retire toute arête entrante et toute arête sortante de ces neurones, ce qui revient à mettre les poids à zéro tant pour l'évaluation que pour la rétropropagation. Lors de l'itération, suivante on choisit de nouveau au hasard les neurones à désactiver. Voir le chapitre « Probabilités » pour plus de détails.