ICC/Programmation — Exercices semaine 10

27 avril 2026

Objectif. Dans ces exercices, vous apprendrez comment fonctionne la convolution et comment l'implémenter avec NumPy. La convolution est la base mathématique du filtrage d'images : le flou et le lissage en sont des cas particuliers. Vous construirez tout depuis zéro, puis découvrirez comment NumPy le fait pour vous.

Ces exercices sont autonomes. Vous n'avez besoin que de NumPy et Pillow.

Mise en place — créer un environnement virtuel. Un environnement virtuel est une installation Python isolée qui garde les bibliothèques de votre projet séparées du reste du système. Cela évite les conflits de versions et permet de reproduire exactement votre configuration sur une autre machine.

Ouvrez Visual Studio Code, ouvrez un terminal, naviguez vers votre dossier de projet, et exécutez :

```
python3 -m venv .venv
```

Cela crée un dossier caché `.venv` contenant une copie privée de Python et des bibliothèques que vous installerez. Rien n'est encore installé.

Activer l'environnement virtuel. Vous devez l'activer à chaque fois que vous ouvrez un nouveau terminal. La commande dépend du système d'exploitation :

- **macOS / Linux :** `source .venv/bin/activate`
- **Windows (PowerShell) :** `.venv\Scripts\Activate.ps1`
- **Windows (Invite de commandes) :** `.venv\Scripts\activate.bat`

Une fois activé, votre invite de terminal affiche `(.venv)` au début. **Si vous ne voyez pas ce préfixe, l'environnement n'est pas actif et `import numpy` échouera.**

Installer les bibliothèques nécessaires. Faites-le une seule fois, juste après la première activation :

```
python3 -m pip install numpy Pillow
```

Vous n'avez pas besoin de les réinstaller la prochaine fois — activer l'environnement suffit.

Créer votre fichier de travail. Créez un nouveau fichier `convolution.py` et placez les lignes suivantes en haut. Tous les extraits de code de ces exercices supposent que ces lignes sont présentes :

```python
import numpy as np
from PIL import Image

def save_image(arr, filename):
    """Sauvegarde un tableau numpy en fichier image."""
    Image.fromarray(arr.astype("uint8")).save(filename)
```

Exécutez votre fichier à tout moment en tapant la commande suivante dans le terminal :

```
python3 convolution.py
```

Exercice 1. Qu'est-ce qu'un noyau ?

Un **noyau** (aussi appelé filtre ou masque) est un petit tableau 2D de poids. Lors d'une convolution, le noyau glisse sur une image et, à chaque position, calcule une somme pondérée des pixels qu'il recouvre. Le résultat remplace le pixel central.

(a) Créez le noyau de **flou uniforme** 3×3. Tous les poids sont égaux et leur somme vaut 1, de sorte que la luminosité moyenne est conservée :

```python
box = np.ones((3, 3)) / 9
print(box)
print("Somme des poids :", box.sum())
```

Pourquoi les poids somment-ils à 1 ? Que se passerait-il pour la luminosité de l'image s'ils sommaient à 2 ?

(b) Créez le noyau **identité** — il laisse l'image inchangée car il lit uniquement le pixel central avec un poids 1 :

```
identity = np.array([[0, 0, 0],
                     [0, 1, 0],
                     [0, 0, 0]], dtype=np.float64)
print(identity)
```

Si vous appliquez ce noyau à n'importe quelle image, à quoi devrait ressembler le résultat ?

(c) Créez un noyau de flou uniforme 5×5 avec tous les poids égaux à $1/25$. Affichez-le et vérifiez que la somme vaut 1 :

```
box5 = np.ones((5, 5)) / 25
print(box5)
print("Somme :", box5.sum())
```

(d) **Question.** Un noyau $k \times k$ examine un voisinage de $k \times k$ pixels autour de chaque pixel. Combien de pixels un noyau 3×3 examine-t-il ? Si un noyau de flou 5×5 est appliqué à une image de 100×100 pixels, combien de multiplications sont effectuées au total ?

Exercice 2. Convolution manuelle sur un petit tableau

Avant d'écrire une fonction générale, travaillez les mécanismes à la main.

(a) Considérez ce signal 5×5 et le noyau de flou 3×3. La valeur de sortie de la convolution à la position **`[2,2]`** (le centre) est la somme pondérée du voisinage 3×3 centré là :

```
signal = np.array([[  0,   0,   0,   0,   0],
                   [  0, 100, 100, 100,   0],
                   [  0, 100, 200, 100,   0],
                   [  0, 100, 100, 100,   0],
                   [  0,   0,   0,   0,   0]], dtype=np.float64)

kernel = np.ones((3, 3)) / 9

region = signal[1:4, 1:4]        # voisinage 3x3 du centre
result = np.sum(region * kernel)  # multiplication élément par élément, puis somme
print("Sortie en [2,2] :", result)
```

Calculez la valeur attendue à la main d'abord, puis exécutez le code pour vérifier.

(b) Calculez la valeur de sortie à la position **`[1,1]`** de la même façon. Quels neuf pixels le voisinage couvre-t-il ?

```
region = signal[0:3, 0:3]
result = np.sum(region * kernel)
print("Sortie en [1,1] :", result)
```

(c) Lorsque le noyau est centré sur un pixel de bord, une partie de celui-ci dépasse en dehors de l'image. Deux stratégies courantes permettent de gérer ce cas :

- **Zero-padding :** traiter toute position hors de l'image comme valant 0. C'est ce que fait **`np.pad(..., constant_values=0)`**.
- **Réplication (clamp) :** répéter la valeur du pixel de bord le plus proche pour toute position en dehors de l'image.

Nous utilisons le zero-padding dans cette session. Vous en verrez l'effet sur les bords de l'image à l'exercice 4.

Exercice 3. Implémenter une fonction de convolution générale

Écrivez maintenant la fonction qui automatise le glissement du noyau sur toute l'image.

(a) Étudiez le squelette ci-dessous, puis complétez la ligne manquante marquée `...` :

```
def convolve2d(image, kernel):
    """
    Convolve une image en niveaux de gris 2D avec un noyau 2D.
    Les bords sont gérés par zero-padding.
    Renvoie un tableau float64 de même forme que l'image.
    """
    kh, kw = kernel.shape
    pad_h = kh // 2    # ex. 1 pour un noyau 3x3
    pad_w = kw // 2

    # Zero-padding pour centrer le noyau sur chaque pixel,
    # y compris ceux du bord.
    padded = np.pad(image, ((pad_h, pad_h), (pad_w, pad_w)),
                    mode='constant', constant_values=0)

    h, w = image.shape
    output = np.zeros((h, w), dtype=np.float64)

    for row in range(h):
        for col in range(w):
            region = padded[row : row + kh, col : col + kw]
            output[row, col] = ...   # à compléter

    return output
```

Indice : utilisez **`np.sum`** et l'opérateur `*`, exactement comme à l'exercice 2.

(b) Testez avec le noyau identité — le résultat doit correspondre à l'entrée :

```
test = np.array([[10, 20, 30],
                 [40, 50, 60],
                 [70, 80, 90]], dtype=np.float64)

identity = np.array([[0, 0, 0],
                     [0, 1, 0],
                     [0, 0, 0]], dtype=np.float64)

result = convolve2d(test, identity)
print(result)
print("Correspond à l'original :", np.allclose(result, test))
```

Si **`np.allclose`** affiche **`False`**, relisez votre corps de boucle.

(c) Appliquez le noyau de flou 3×3 au **`signal`** de l'exercice 2 et affichez le résultat. Comparez la valeur centrale avec votre calcul manuel de 2(a) :

```
box = np.ones((3, 3)) / 9
blurred = convolve2d(signal, box)
print(np.round(blurred, 1))
```

Les valeurs de bord (où le zero-padding a été utilisé) ont-elles du sens ?

Exercice 4. Noyaux de flou

Un noyau de flou remplace chaque pixel par une moyenne pondérée de ses voisins. Plus le noyau est large, plus le flou est fort.

(a) Chargez l'image fournie et appliquez le flou uniforme 3×3. Assurez-vous que **`original.jpg`** se trouve dans le même dossier que votre script :

```
img = np.array(Image.open("original.jpg").convert("L"), dtype=np.float64)
print("Forme de l'image :", img.shape)

box3 = np.ones((3, 3)) / 9
blurred3 = convolve2d(img, box3)
save_image(blurred3, "flou_3x3.png")
```

(b) Appliquez un noyau 5×5 et un noyau 9×9 et sauvegardez les résultats :

```
box5 = np.ones((5, 5)) / 25
blurred5 = convolve2d(img, box5)
save_image(blurred5, "flou_5x5.png")

box9 = np.ones((9, 9)) / 81
blurred9 = convolve2d(img, box9)
save_image(blurred9, "flou_9x9.png")
```

Ouvrez **`original.jpg`**, **`flou_3x3.png`**, **`flou_5x5.png`** et **`flou_9x9.png`** côte à côte. Que remarquez-vous quand le noyau s'agrandit ?

(c) **Question.** Votre fonction **`convolve2d`** utilise deux boucles Python imbriquées. Pour une image de 1000×1000 pixels avec un noyau 3×3, combien d'itérations la boucle interne effectue-t-elle au total ?

Exercice 5. Convolution avec NumPy

Vous avez construit **`convolve2d`** depuis zéro pour en comprendre le fonctionnement. NumPy fournit **`np.convolve`** pour les signaux 1D — une implémentation rapide et compilée de exactement la même opération. Dans cet exercice vous l'explorerez, puis vous l'appliquerez ligne par ligne à une image.

(a) **np.convolve sur un signal 1D.** Exécutez le code suivant et observez le résultat :

```
signal_1d = np.array([0, 0, 100, 200, 100, 0, 0], dtype=np.float64)
kernel_1d = np.array([1/3, 1/3, 1/3])   # moyenne glissante sur 3 points

result_same = np.convolve(signal_1d, kernel_1d, mode='same')

print("signal   :", signal_1d)
print("résultat :", np.round(result_same, 1))
```

`mode='same'` conserve la même longueur que le signal d'entrée en ajoutant des zéros aux bords — exactement ce que fait votre **`convolve2d`**.

(b) **Appliquer np.convolve ligne par ligne.** **`np.convolve`** ne fonctionne que sur des tableaux 1D, mais vous pouvez l'appliquer à chaque ligne d'une image 2D dans une boucle. Complétez la fonction ci-dessous en remplissant l'argument manquant :

```
def convolve_rows(image, kernel_1d):
    """Applique un noyau 1D à chaque ligne d'une image 2D."""
    h, w = image.shape
    output = np.zeros_like(image, dtype=np.float64)
    for row in range(h):
        output[row, :] = np.convolve(image[row, :], kernel_1d, mode=...)
    return output
```

(c) **Filtrage séparable.** Un flou uniforme 2D peut être décomposé en deux passes 1D : d'abord flouter chaque ligne, puis flouter chaque colonne du résultat. On appelle cela un filtre séparable et c'est beaucoup plus rapide pour les grands noyaux. Appliquez-le à votre image et sauvegardez le résultat :

```
k = np.array([1/5, 1/5, 1/5, 1/5, 1/5])   # moyenne glissante sur 5 points

# Passe 1 : flouter chaque ligne
after_rows = convolve_rows(img, k)

# Passe 2 : flouter chaque colonne en transposant, floutant les lignes,
#           puis en retransposant
after_cols = convolve_rows(after_rows.T, k).T

save_image(after_cols, "flou_separable.png")
```

Comparez **`flou_separable.png`** avec **`flou_5x5.png`**. Semblent-ils identiques ?

(d) **Vérifier que les résultats correspondent.** Vérifiez que votre filtre séparable donne le même résultat que **`convolve2d`** avec le noyau uniforme 5×5 :

```
result_2d  = convolve2d(img, box5)
result_sep = after_cols

print("Différence max       :", np.max(np.abs(result_2d - result_sep)))
print("Résultats identiques :", np.allclose(result_2d, result_sep))
```