TP-Cours d'Informatique 12

Traitement des images

ITC – PCSI^2 2024-2025 – Lycée Fabert (Metz) – L. Parise

La plupart des formats de fichier d'image utilisent de la compression de données (comme GIF, JPG, PNG ou plus récemment HEIC) : certains types de compression sont sans perte (PNG) alors que d'autres engendrent une perte d'information (JPG).

Cette compression facilite le stockage des images et leur envoi, enjeu crucial dans les années 90 et toujours d'actualité avec l'avènement de smartphones et des communications mobiles qui continuent (pour combien de temps encore ?) à être moins rapides que les transmissions filaires d'information.

La lecture et visualisation de ces images numériques sur un écran nécessite donc l'utilisation d'algorithmes de décodage implémentés dans vos ordinateurs, vos smartphones, etc. Une fois l'information décodée pour être affichée, la représentation informatique des images « colle » avec la réalité physique des écrans constitués de pixels disposés selon une grille, qui peuvent être « allumés » pour produire plus ou moins de lumière dans diverses tonalités.

Dans ce TP, nous utiliserons essentiellement des images en niveaux de gris. Une image sera modélisée par un tableau/une matrice I dont chaque élément I[x,y] représentera un niveau de gris compris entre 0 et 255. Comme de tels entiers (de type uint8 – **pour unsigned integers 8 bits** – dans numpy) peuvent être représentés sur 8 bits=1 octet, cela signifie que le poids en octets d'un tel fichier-image sera le nombre de pixels représentés par le tableau I c.-à-d. le produit du nombre de lignes par le nombre de colonnes de I.

1. Commencer par importer les bibliothèques numpy (np) et matplotlib.pyplot (plt). On définit un tableau numpy contenant les informations stockées dans un fichier-image en utilisant la fonction plt.imread.

```
sdFetes = plt.imread("sdFetes.jpg")

print(sdFetes[0,:])

hauteur, largeur = sdFetes.shape
print("Largeur de l'image en pixels :", largeur)
print("Hauteur de l'image en pixels :", hauteur)

print("Type de la variable générée par imread :", type(sdFetes))
print("Dimension du tableau :", sdFetes.ndim)

plt.figure("La salle des Fêtes du lycée Fabert")
plt.imshow(sdFetes,cmap=plt.cm.gray,vmin=0,vmax=255)
```

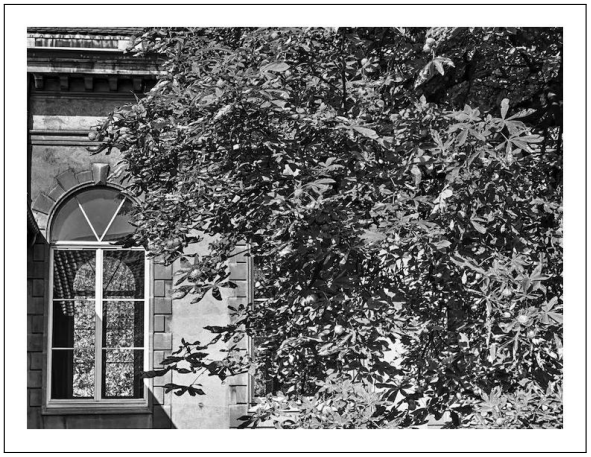
Listing 1 – Lecture d'un fichier-image et affichage

   Afficher la première ligne de ce tableau, récupérer sa taille : le nombre de lignes est la hauteur de l'image et le nombre de colonnes sa largeur. Afficher le type de ce tableau ainsi que sa « dimension ». L'affichage est effectué au moyen de la fonction plt.imshow ... et n'oubliez pas avant toute mise à l'écran, de définir et de nommer votre figure/fenêtre graphique !

2. Essayer de modifier le tableau sdFetes pour que le pixel en haut à gauche de l'image devienne blanc. Que se passe-t-il ?

Effectuer une copie empereur1 du tableau sdFetes à l'aide de deux boucles **for** imbriquées ou à l'aide de la fonction copy.deepcopy. Modifier empereur1 pour afficher une image contenant une ligne horizontale de pixels blancs à mi-hauteur. Effectuer le même travail avec une bande horizontale de 20 pixels blancs : on utilisera le **slicing** des tableaux numpy.

Pour les plus rapides : créer une nouvelle image/un nouveau tableau numpy pour **ajouter** un cadre blanc de 20 pixels autour de l'image de feu le marronnier du lycée Fabert.



3. À partir du tableau sdFetes, créer une image dont la largeur et la longueur sont divisées par 3 en ne prenant qu'une ligne et une colonne sur 3 dans ce tableau. Utiliser le slicing.

4. Il est possible de concaténer deux tableaux numpy pour en former un troisième. En fonction du nombre de lignes et de colonnes de chaque tableau, on peut agréger les images suivant l'axe horizontal ou verical :

```
GD = np.concatenate((sdFetes,sdFetes),axis=1)    # Gauche-Droite
plt.figure("Concaténation selon axe vertical")
plt.imshow(GD,cmap=plt.cm.gray,vmin=0,vmax=255)

HB = np.concatenate((sdFetes,sdFetes),axis=0)    # Haut-Bas
plt.figure("Concaténation selon axe horizontal")
plt.imshow(HB,cmap=plt.cm.gray,vmin=0,vmax=255)
```

Listing 2 – Exemple de concaténation de tableau numpy : 2 axes possibles

Utiliser cette technique pour produire une image avec effet miroir gauche-droite à partir de sdFetes puis découper des bandes horizontales de 90 lignes dans sdFetes, les stocker dans la liste bandesH, définir une liste contenant les entiers de 1 à 7 dans un ordre quelconque :

```
In : ordre = np.random.permutation(7) ; print(ordre)
Out: [2,1,4,7,6,5,3]
```

Reformer une image de même taille en agençant ces bandes horizontales suivant cet ordre. Afficher le résultat !

5. Lire le fichier entree.jpg à l'aide de plt.imread : le tableau numpy obtenu sera nommé entree. On souhaite diminuer le nombre de niveaux de gris de cette image dans laquelle les gris ont de valeurs allant de 0 à 255. Pour ce faire on va définir le nombre de niveaux souhaité, par exemple nbniveaux = 5 et on va créer une « subdivision régulière » de $[\![0;255]\!]$ sous la forme d'une liste

$$\text{Gris} = [0,51,102,153,204,255]$$

Remarquez que cette « subdivision » de $[\![0;255]\!]$ est constituée de nbniveaux intervalles qui sont ici $[\![0;51[\![$, $[\![51;102[\![$, $[\![102;153[\![$, $[\![153;204[\![$ et $[\![204;255]\!]$.

Créez une image/un tableau entree1 avec nbniveaux niveaux de gris en testant chaque élément entree[i,j] du tableau entree pour savoir dans quel intervalle de la subdivision cet élément se situe. Une fois l'intervalle déterminé on donnera à entree1[i,j] la valeur moyenne de l'intervalle.

6. Créez l'image/le tableau inverse, complément à 255 de entree (ce qui signifie que la somme entree+inverse doit former une image entièrement blanche).

7. À présent, on souhaite effectuer quelques statistiques sur les images que nous allons traiter, et cela dans le but, notamment, de créer un histogramme des niveaux de gris de chaque image. Le but de cette question est d'écrire la fonction

```
statsGris(image:numpy.ndarray)→
                grisMin:int, grisMax:int, moyenne:float, ecType:float, dictGris:dict
```

— Commencer par écrire la partie de code qui permet à statsGris de renvoyer le niveau de gris grisMin le plus bas du tableau image , le niveau le plus haut grisMax et la moyenne des gris.

— Poursuivre en calculant et en renvoyant l'écartype de toutes les niveaux de gris du tableau image.

— Terminer en formant le dictionnaire dictGris dont les clés sont les niveaux de gris de l'image (compris entre 0 et 255) et dont la valeur associée à chaque clé est le nombre de pixels de l'image ayant le niveau de gris de la clé.

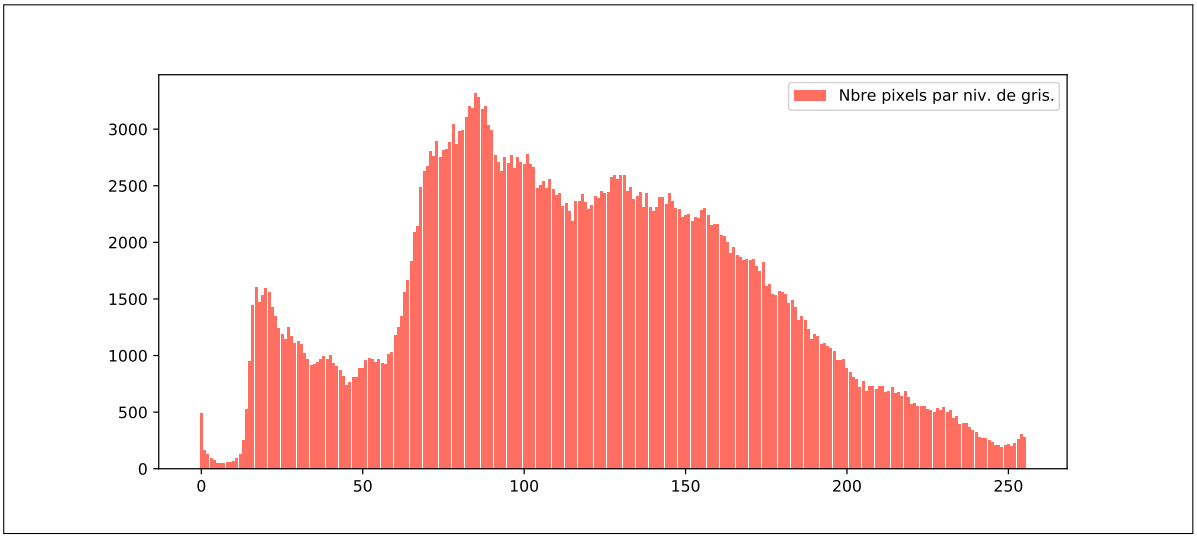
Test : Pour l'image entree.jpg :
grisMin, grisMax; moyenne, ecType =(0, 255, 117.8175720164609, 53.42083400733324)

8. Pour effectuer le tracé d'un histogramme avec matplotlib.pyplot, c'est très simple : si notes est la liste des notes du dernier DS, si nbeleves est une liste de même longueur que notes, alors on obtient l'histogramme donnant le nombre d'élèves par note par l'instruction

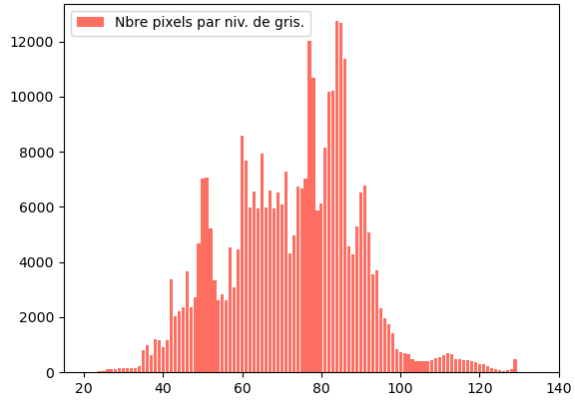
```
plt.bar(notes,nbeleves)
```

Écrire la fonction histogramme(image,nom)→None qui utilise la fonction statsGris pour obtenir le dictionnaire donnant le nombre de points de image ayant un niveau de gris donné et effectuer le tracé de l'histogramme correspondant à ce dictionnaire.

Ci-dessous, l'histogramme de l'image entree.jpg



9. Chargez et afficher l'image main.jpg :



Cette image est trop foncée et peu contrastée. Utilisez la fonction statsGris pour obtenir la moyenne, l'écart-type des gris puis tracer l'histogramme associé à cette image. On constate une faible plage [grisMin;grisMax] = [21; 134] et une moyenne = 72.696 faible. L'écart-type des gris est ecType = 16.536.

Vient alors l'idée de passer par une normalisation de la série de données (les niveaux de gris stockés dans le tableau numpy représentant l'image) : nous allons centrer-réduire les valeurs du tableau-image pour obtenir une moyenne nulle et un écart-type à 1. Ensuite, nous pourrons fixer l'écart-type souhaité (par dilatation des données) ainsi que la moyenne (par translation).

Écrire la fonction

ajusteHisto(image:**numpy.ndarray**,moyenne:**float**,ecType:**float**)→sortie :**numpy.ndarray**

qui associe à image le tableau-image sortie obtenu par le procédé décrit ci-dessus.

Remarque 1 : Comme nous allons diviser les données d'un tableau-image par l'écart-type des données et que la moyenne des données est en général un réel non entier, nous allons travailler sur une copie du tableau-image dont les éléments sont de type **float64**. Pour effectuer cette copie en modifiant le type des données, on utilisera la méthode astype associée à un objet de type numpy.ndarray : image.astype(`"float64"`) fournit la copie souhaitée. Par ailleurs, une fois le tableau modifié, sortie devra contenir des valeurs de type **uint8** à l'instar du tableau image.

Remarque 2 : Lors des modifications de niveaux de gris par homothétie-translation, et en fonction de la moyenne et de l'écart-type choisi, il est possible d'obtenir des valeurs négatives ou supérieures à 255. Pour éviter cet écueil, on appliquera à chaque élément du tableau la fonction cadrage définie par :

```python
def cadrage(x):
    if x>255:
        return 255
    if x<0:
        return 0
    return round(x)
```

Essayez les couples (moyenne,ecType) suivants :

$$(130, 35) \quad \text{et} \quad (130, 62)$$

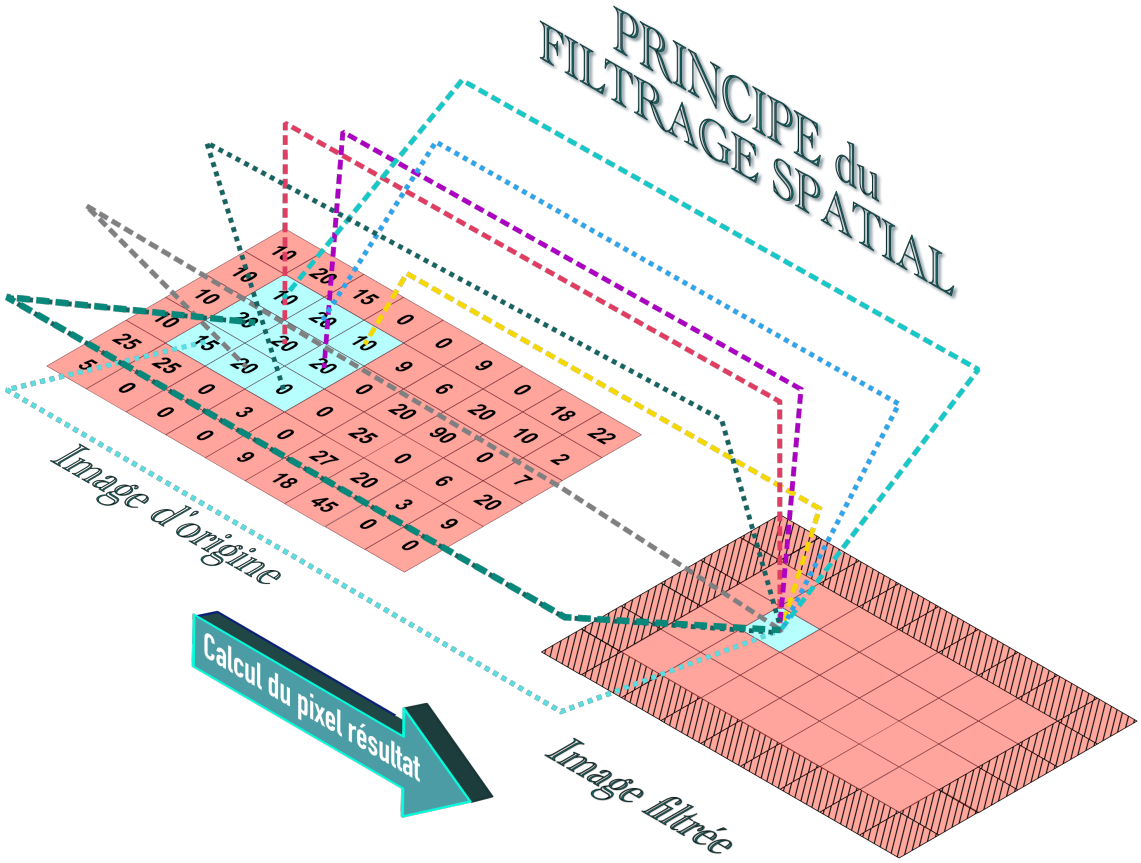
Expliquez la « qualité » de l'image en s'aidant de son histogramme.

10. **Filtrage spatial par masque de convolution** Le filtrage spatial convolutif correspond à des opérations permettant de modifier chaque pixel de l'image en fonction des valeurs des pixels de son voisinage. **Un filtre spatial est donc convolutif défini par :**

— le masque de convolution : taille et forme du voisinage considéré : c'est souvent une matrice carrée de taille impaire 3x3 ou 5x5 ou 7x7 ... etc,

— la position du pixel central dans le support,

— l'algorithme utilisé pour le calcul de la valeur de sortie à partir des valeurs du voisinage.



L'opération de convolution de l'image représentée par le tableau image de type numpy.ndarray de taille largeur x hauteur par le masque $h \in \mathrm{M}_{2n+1}(\mathbb{R})$ avec $n \in \mathbb{N}^*$ est donnée par :

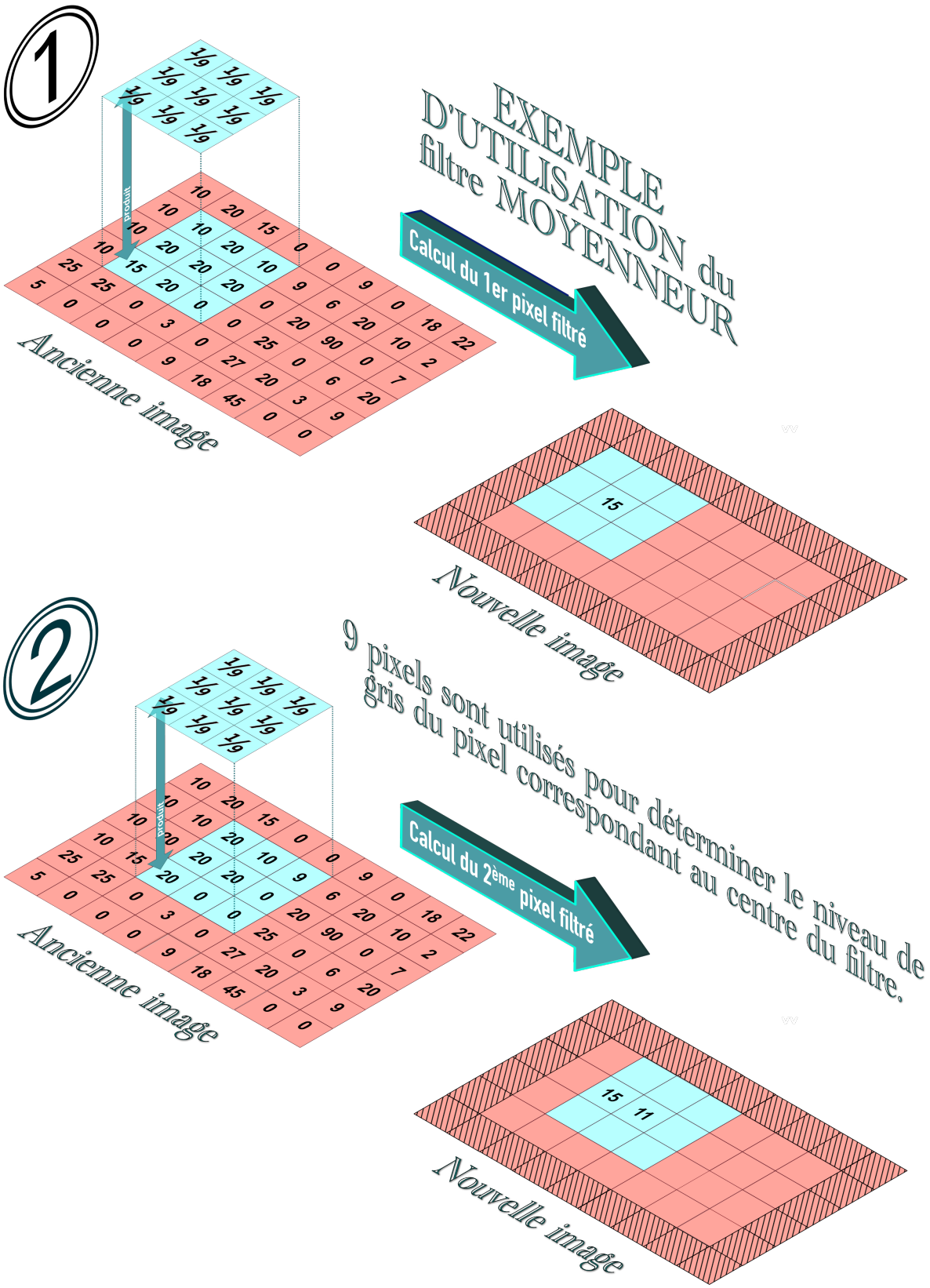
$$\widetilde{\text{image}}[x, y] = (\text{image} \bigstar h)[x, y] = \sum_{i=0}^{2n} \sum_{j=0}^{2n} \text{image}[x - n + i, y - n + j] \times h[i, j]$$

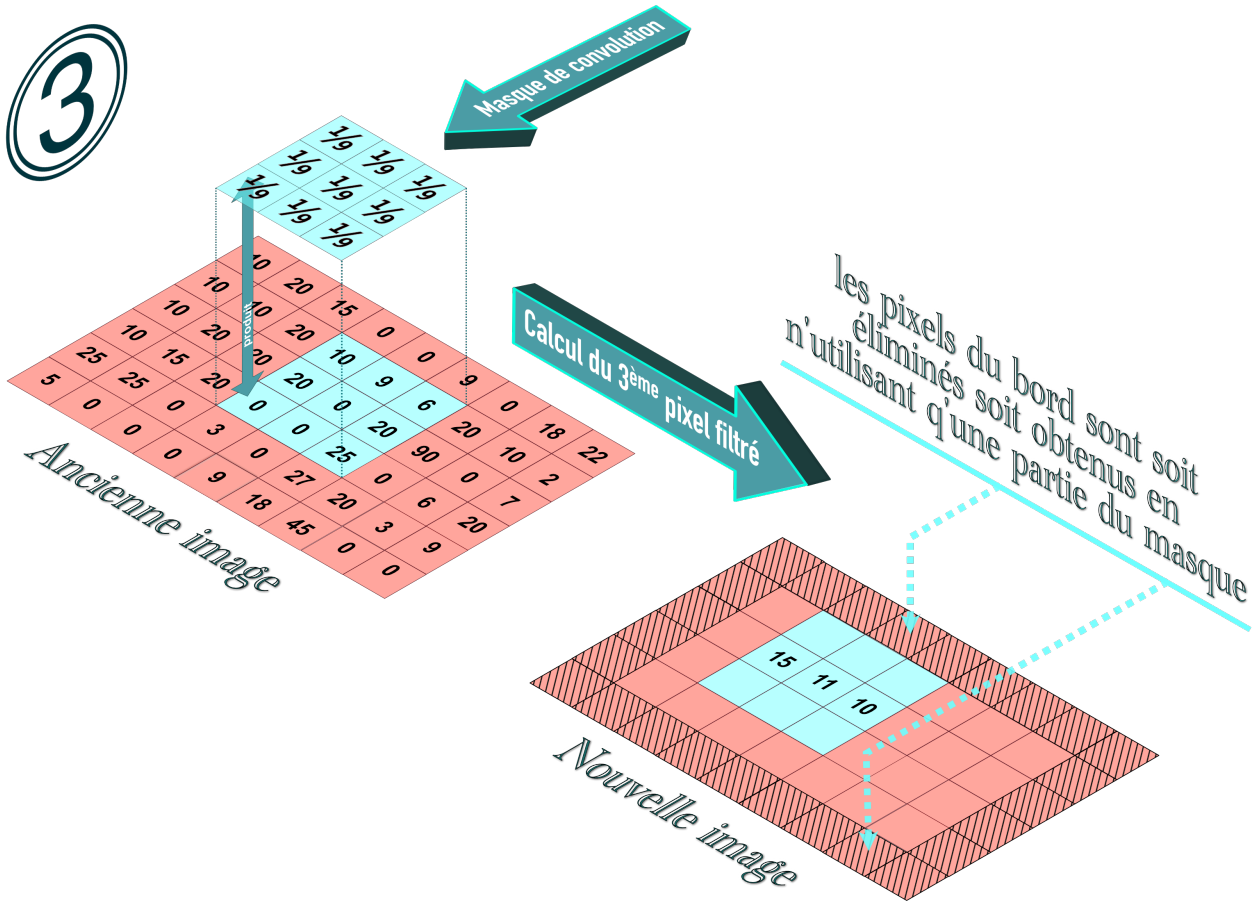
pour tout $x \in [\![1; \text{hauteur} - 2]\!]$ et tout $y \in [\![1; \text{largeur} - 2]\!]$. Dans les exemples qui vont suivre, on posera $n = 1$ c'est-à-dire que le masque de convolution sera une matrice de $\mathrm{M}_3(\mathbb{R})$.

Algorithme : balayage de l'image par le masque de convolution :

— le masque est superposé sur l'image, en le positionnant sur le voisinage du pixel courant,
— chaque coefficient du masque est multiplié avec le pixel qu'il recouvre, puis on ajoute toutes les valeurs pour obtenir la valeur de sortie du pixel courant,
— cette opération est répétée pour tous les pixels de l'image, sauf ceux du bord !

Les figures ci-dessous illustrent le processus du balayage de l'image par le masque de convolution, avec un exemple de filtre simple (filtre moyenneur) :





Définition 1 *Un filtre de convolution est dit normalisé lorsque la somme des coefficients du masque est égale à 1 c'est-à-dire (avec les notations précédentes) :*

$$\sum_{i=0}^{2n}\sum_{j=0}^{2n} h[i,j] = 1$$

L'intérêt des filtres normalisés est la conservation de la luminance globale de l'image. Inversement, un filtre non normalisé éclaircit ou assombrit globalement l'image.

Important : l'opération de filtrage par convolution est une somme pondérée, et donc c'est une opération linéaire : les valeurs des pixels de l'image filtrée sont des combinaisons linéaires des valeurs des pixels de l'image d'origine. C'est pourquoi les filtres réalisés par convolution sont appelés **filtres linéaires**.

Pour réaliser le lissage d'une image, plusieurs filtres spatiaux sont disponibles dont les filtres linéaires de convolution passe-bas, qui vont supprimer les hautes fréquences de l'image. Les principaux filtres de convolution passe-bas sont le **filtre moyenneur** et le **filtre gaussien** et des filtres non linéaires comme le **filtre médian**.

Écrire la fonction convolution(image:**numpy.ndarray**,H:**numpy.ndarray**)→sortie :**numpy.ndarray** qui applique le masque de convolution H au tableau numpy image et renvoie le tableau filtré. Dans un premier temps, on fera des essais avec le filtre moyenneur et le filtre gaussien.

Masque moyenneur

$$\frac{1}{9}\times\begin{array}{|c|c|c|}\hline 1&1&1\\\hline 1&1&1\\\hline 1&1&1\\\hline\end{array}$$

Masque gaussien

$$\frac{1}{16}\times\begin{array}{|c|c|c|}\hline 1&2&1\\\hline 2&4&2\\\hline 1&2&1\\\hline\end{array}$$