TP 6 : images

1 Les images

• Une image numérique est un tableau dont les cases sont de petits carrés appelés *pixels*.
• En informatique, une image est donc un tableau (ou une matrice) dont chaque case correspond à un pixel et contient sa couleur.
• La couleur d'un pixel est un triplet d'entiers compris entre 0 et 255.
• Un triplet donnant la couleur est donné selon le codage RGB (Red, Green, Blue). Chaque entier est d'autant plus grand que la couleur est grande dans le pixel.
• Le triplet $(0, 0, 0)$ donne la couleur noire et le triplet $(255, 255, 255)$ donne le blanc.
• Si les trois composantes sont égales (à un entier $i \in \{0, \dots, 255\}$), on dit que la couleur obtenue est grise de niveau i. Si tous les pixels de l'image sont grises, on parle d'une image en niveau de gris.

2 Avec Python - en utilisant Python Image Library

ATTENTION : la numérotation des cases (pixels) n'est pas celle des matrices :

1. Si i et j sont deux entiers naturels, (i, j) correspond au pixel situé **en colonne** i **et ligne** j.
2. La case située en haut à gauche est numérotée $(0, 0)$.

2.1 Bibliothèque PIL

On commence par importer le module `Image` (attention la majuscule) de la bibliothèque `PIL` (pour **P**ython **I**mage **L**ibrary) :

```
from PIL import Image
```

2.2 Création d'une image

Pour une image à n colonnes et p lignes, on utilise la commande (l'image sera blanche ici) :

```
mon_image = Image.new("RGB",(n,p),(255,255,255))
```

2.3 Import d'une image

Pour importer une image existante :
• en utilisant l'icône répertoire ou l'explorateur de fichiers, choisir pour répertoire de travail celui contenant l'image à importer ;
• importer l'image en utilisant la commande :

```
mon_image = Image.open("nom_du_fichier.png")
```

2.4 Accéder à un pixel et le modifier

• Pour accéder à la taille d'une image, on utilise la commande :

```
mon_image.size
```

qui retourne une liste. Ainsi, `mon_image.size[0]` est le nombre de colonnes de l'image et `mon_image.size[1]` est le nombre de lignes.
• Pour accéder au triplet donnant la couleur du pixel en colonne i et ligne j, on utilise la commande :

```
pixel = mon_image.getpixel(i,j)
```

• On récupère alors un triplet, et pour accéder à la composante rouge (resp. verte, resp. bleue), on saisit alors :

`pixel[0]\ (resp. pixel[1], resp. pixel[2])`

• Pour modifier le triplet RGB du pixel en colonne i et ligne j, on utilise la commande :

```
mon_image.putpixel((i,j),(nouveau_R,nouveau_G,nouveau_B))
```

2.5 Afficher et enregistrer l'image

• L'affichage de l'image se fait en utilisant la commande :

```
mon_image.show()
```

• L'enregistrement de l'image se fait dans le répertoire courant en utilisant la commande :

```
mon_image.save("nom_du_fichier.png")
```

3 Avec Python - sans utiliser Python Image Library

Dans ce cas, on travaille avec des tableaux, c'est-à-dire les structures de données `array` de `numpy`

3.1 Bibliothèques matplotlib et numpy

On commence par importer le module `pyplot` de la bibliothèque `matplotlib` :

```
import matplotlib.pyplot as plt
import numpy as np
```

Seuls les fichiers .png sont utilisables, et les niveaux de couleurs qui a priori sont des octets non signés sont convertis en flottants dans [

3.2 Création d'une image

Pour une image à n colonnes et p lignes, on utilise l'une des commandes commande (l'image sera blanche ici) :

```
image_NB = np.zeros((p,n), dtype = np.uint8)
image_couleur = np.zeros((a,b,3), dtype = np.uint8)
```

3.3 Import d'une image

Pour importer une image existante :
• en utilisant l'icône répertoire ou l'explorateur de fichiers, choisir pour répertoire de travail celui contenant l'image à importer ;
• importer l'image en utilisant la commande :

```
mon_image = plt.imread("nom_du_fichier.png")
```

ce qui a pour effet de transformer l'image en tableau.
Il est parfois préférable de saisir :

```
mon_image=np.uint8(plt.imread("nom_du_fichier.png") * 255)
```

3.4 Accéder à un pixel et le modifier

• Comme une image est dans ce cas de type `array`, on fonctionne comme avec les matrices.

3.5 Afficher et enregistrer l'image

• L'affichage de l'image se fait en utilisant la commande :

```
plt.imshow(mon_image,interpolation='nearest')
plt.show()
```

• L'enregistrement de l'image se fait dans le répertoire courant en utilisant la commande :

```
plt.savefig("nom_du_fichier.png")
```

4 Manipulations d'images

4.1 Niveaux de gris, noir et blanc

1. Écrire une fonction qui prend en argument une image et qui retourne la version en niveaux de gris de l'image. Pour cela, on choisira pour niveau de gris la moyenne (rendue entière) des trois composantes RGB d'origine.
2. Écrire une fonction qui prend en argument une image et un seul et qui retourne la version en noir et blanc, ou un gris au delà du seul fixé est blanc.

4.2 Effet miroir, négatif

3. Écrire une fonction qui prend en argument une image et retourne sa représentation dans un miroir.
4. Écrire une fonction qui prend en argument une image et retourne son négatif.

4.3 Luminosité et contraste

5. Écrire une fonction qui prend en argument une image et un entier a compris entre -255 et 255. Elle augmentera lorsque c'est possible chaque composante d'un pixel de a.
6. Écrire une fonction qui prend en argument une image et un entier a compris entre 0 et 255. Elle transformera chaque composante d'un pixel en utilisant la fonction $f : x \longmapsto \begin{cases} 0 & \text{si } x \leqslant a \\ \frac{255}{255-2a}(x-a) & \text{si } x \in [a, 255-a] \\ 255 & \text{si } x > 255-a \end{cases}$.

4.4 Détecter des contours

7. Écrire une fonction prenant en argument une image et un seuil s et qui en renvoie une image des contours en noir et blanc.
   Pour ce faire : • introduire une variable `gris` contenant l'image en niveau de gris,
   • créer une image blanche de la taille de l'image de départ,
   • pour les pixels qui ne sont pas sur les bords, calculer la moyenne de gris des 8 pixels voisins.
   si l'écart entre cette moyenne de gris et le gris du pixel considéré est supérieur au seuil s, on pense que ce pixel représente un bord et on le place en noir.
   si l'écart est inférieur au seuil, on laisse le pixel en blanc.

4.5 Flou et pixellisation

8. Écrire une fonction prenant en argument une image et qui la floute.
   Pour les pixels qui ne sont pas sur les bords, on attribue à chaque composante d'un pixel la moyenne des composantes des huit pixels voisins.
9. Écrire une fonction prenant en argument une image et un entier c et qui en renvoie une version pixellisée.
   Pour ce faire : • On partitionne l'image en sous-images carrées de côté c.
   • On calcule la moyenne des composantes des pixels de la sous-image.
   • On attribue cette valeur à l'ensemble des pixels de la sous-image.