TD 1 : INTRODUCTION

Dans ce TD, on utilisera la bibliothèque python de traitement d'images scikit-image pour charger les images et les packages de Matplotlib pour la visualisation :

Les images seront traitées comme un tableau numpy (classe numpy.ndarray)

- Importations :

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

EXERCICE 1 : Manipulation des images en niveaux de gris

Compléter le fichier de commande TD1Ex1.py selons les instructions ci-dessous.

Manipulation des images en niveaux de gris

- Charger l'image autumng.png, afficher les informations principales sur cette image et l'afficher et la visualiser :

```
im = ski.io.imread("autumng.tif")
print("Informations sur l\'image chargée:")
print("type de im:",type(im))
print("\tType des éléments:",im.dtype)
print('\tIntervalle des valeurs:[',im.min(),',',im.max(),']')
print("\tDimension:",im.ndim)
print("\tTaille:",im.shape)

plt.imshow(im,cmap='gray')
plt.colorbar()
plt.show()
```

↳ *Est-ce une image en couleur ou en niveaux de gris ? Quel est le nombre de couleurs ou de niveaux de gris ? Quelles sont les dimensions de cette image ? Quel est le rôle du paramètre cmap='gray' ?*

↳ *Quelle est la valeur du pixel de coordonnées* $(62,121)$ *?*

↳ *Transformer cette image pour un codage en nombre flottants.*

```
im2 = im/ 255
```

NB : on peut aussi utiliser la fonction .util.img_as_float du package util :

```
im2 =ski.util.img_as_float(im)
```

↳ *Quelle est la dynamique de l'image, est-ce que les niveaux de gris sont quantifiés ?*

Manipulation d'une table des couleurs

On va maintenant créer notre propre table des couleurs pour l'affichage des images avec la fonction `matplotlib.colors.listedColormap`. Ici notre table des couleurs définit des niveaux de gris.

➢ Taper

```
from matplotlib.colors import ListedColormap


nbCouleurs = 256
myColors = np.zeros((nbCouleurs,3),float)
for p in range(3):
    myColors[:,p]=np.linspace(0,1,nbCouleurs)
newcmp = ListedColormap(myColors)
plt.figure()
plt.imshow(im,cmap=newcmp)
plt.colorbar()
plt.title('Affichage sur '+ str(nbCouleurs) + ' couleurs')
```

↳ *Faire varier `nbCouleurs` (e.g. 16), que constatez-vous ? Est-ce que l'image a été modifiée ou juste l'affichage ?*

Affichage en fausses couleurs

➢ Refaire une table des couleurs de 256 valeurs et modifier les 128 dernière lignes pour avoir une dominante rouge :

```
myColors[-128:,0]=1
```

↳ *Que constatez-vous ?*

EXERCICE 2 : Manipulation des images en couleurs

➢ Charger l'image `fleur.jpg`, afficher les informations sur cette image et l'afficher

```
im = ski.io.imread('fleur.jpg')
plt.figure()
plt.imshow(im)
print("Type de l'image : ", im.dtype)
print("Dimensions      : ", im.ndim)
print("Taille          : ", im.shape)
```

↳ *Quelles sont les caractéristiques de cette image ?*

➢ Afficher les 3 canaux en niveaux de gris

```
H,W,P = im.shape

R = -- to be completed
```

```
G = -- to be completed
B = -- to be completed

plt.figure(figsize=(15,5))
plt.subplot(1,3,1)
plt.imshow(R,cmap='gray')
plt.title('Canal R')
plt.subplot(1,3,2)
plt.imshow(G,cmap='gray')
plt.title('Canal V')
plt.subplot(1,3,3)
plt.imshow(B,cmap='gray')
plt.title('Canal B')
```

- Convertir cette image en niveaux de gris avec un codage en nombres flottants. Utiliser la transformation du cours.

```
img = -- to be completed      # cours
img = -- to be completed

plt.figure(fig size=(10,5))
plt.subplot(1,2,1)
plt.imshow(img,cmap = 'gray')
plt.title('Ma conversion')
print("Type de l'image : ", img.dtype)
print("Dimensions      : ", img.ndim)
print("Taille          : ", img.shape)
```

- Utiliser maintenant la fonction `color.rgb2gray` de scikit-Image.

```
img2 = ski.color.rgb2gray(im)
plt.subplot(1,2,2)
plt.imshow(img2,cmap = 'gray')
plt.title('scikit-Image conversion')
print("Type de l'image : ", img2.dtype)
print("Dimensions      : ", img2.ndim)
print("Taille          : ", img2.shape)
```

↳ *Obtenez-vous le même résultat ? Interpréter en vous référant à la documentation. Vous pouvez tester par :*

```
print('img2 = img :' + str(np.array_equal(img,img2 )))
```

EXERCICE 3 : Modification d'une image

On va effectuer quelques manipulations simples sur les pixels des images en niveaux de gris.

- Charger l'image de test cameraman :

```
im = ski.data.camera()
print("Informations sur l\'image chargée:")
print("type de im2:",type(im))
print("\tType des éléments:",im.dtype)
print('\tIntervalle des valeurs:[',im.min(),',',im.max(),']')
print("\tDimension:",im.ndim)
print("\tTaille:",im.shape)
plt.figure(figsize=(15,5))
plt.imshow(im,cmap='gray')
plt.colorbar()
```

```
plt.title('Cameraman')
plt.show()
```

Codage en nombre flottants

Les traitements sur les images ne retournant généralement pas des valeurs entières, il est d'usage de transformer l'image source pour la coder en nombre flottants avant tout traitement.

- Transformer l'image en nombres flottants codés sur $[0,1]$

```
im = -- to be completed # change en type float64
print("Informations sur l\'image chargée:")
print("type de im2:",type(im))
print("\tType des éléments:",im.dtype)
print('\tIntervalle des valeurs:[',im.min(),',',im.max(),']')
print("\tDimension:",im.ndim)
print("\tTaille:",im.shape)
plt.figure(figsize=(15,5))
plt.imshow(im,cmap='gray')
plt.colorbar()
plt.title('Cameraman')
plt.show()
```

Modifier directement la valeurs de pixels

- Copier cette image dans im2 et modifier im2 pour créer un rectangle blanc entre les coins de coordonnées $(100,350)$ et $(200,400)$, respectivement coin en haut à gauche et en bas à droite.

```
im2 = im.copy()
i1, i2 = 100, 200
j1, j2 = 350, 400
im2[-- to be completed]= [-- to be completed
```

Binariser une image

Il est fréquent de binariser une image pour détecter les pixels qui prennent une valeur supérieure ou inférieure à un seuil. Ecrire un programme pour cela. On pourra utiliser de boucles for pour traiter chaque pixel, ou écrire directement une relation logique sur l'image.

- Charger l'image à traiter

```
im = ski.data.page()
im = ski.util.img_as_float(im)
```

- Traitement par pixel

```
im3 = np.empty_like(im)
H,W=im3.shape
T = 0.3
for i in range(H):
    for j in range(W):
        if -- to be completed:
            im3[i,j]= -- to be completed
        else:
            im3[i,j]= -- to be completed
```

- Traitement de la matrice

```
im3 = -- to be completed
```

- Affichage

```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.imshow(im,cmap='gray')
plt.title('im')
plt.subplot(1,2,2)
plt.imshow(im3,cmap='gray')
plt.title('im3 obtenue par seuillage – méthode for')
plt.colorbar()
plt.show()
```

Utilisation de masques

L'idée est de modifier uniquement les pixels définis par un masque. Une image binaire peut directement servir de masque.

- Taper le code suivant et interpréter

```
im = ski.data.cat()
im = ski.color.rgb2gray(im)
T = 0.3
im5 = im.copy()
# méthode 1
masque  = im>T
im5[masque]=0
```

- Alternativement on peut écrire

```
im5 = im.copy()
im5[np.argwhere(im>T)]=1
```

- Afficher les résultats

```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.imshow(im,cmap='gray')
plt.subplot(1,2,2)
plt.imshow(im5,cmap='gray')
plt.title('Image avec masque')
plt.show()
```

La fonction `numpy.mgrid` permet de créer des tableaux représentant les coordonnées de chaque pixel puis de créer un masque. Par exemple :

- Taper le code suivant et interpétrer :

```
H,W = im.shape
G = np.mgrid[0:H,0:W]
X = G[0,:,:]-H/2
Y = G[1,:,:]-W/2
R = np.sqrt(X**2+Y**2)
R0 = 0.4*min(H,W)
masque = R>R0
im7 = im.copy()
im7[masque]=0

plt.figure()
```

```
plt.imshow(im7,cmap='gray')
plt.title('Image avec masque circulaire')
plt.show()
```

↳ *Que fait ce code ?*

➢ Modifier le code pour faire un éclairage circulaire (spot) centré sur le nez du chat en assombrissant la périphérie.

```
im8 = im.copy()
im8[masque] = -- to be completed
plt.figure()
plt.imshow(im8,cmap='gray')
plt.title('Image avec masque circulaire')
plt.show()
```