Traitement des images

Dans ce TP, vous découvrirez les outils que possède Python pour manipuler des images.
Pour cela, nous aurons besoin de la bibliothèque matplotlib.pyplot (ou pylab) et la bibliothèque numpy que nous avons déjà rencontrée .

1. Méthodes de bases

a) Récupération d'une image

Saisissez et exécutez le script suivant :

```
1  import numpy as np      # importation de la bibilothèque numpy
2
3  import matplotlib.pyplot as plt     # importation de la bibilothèque  matplotlib.pyplot
4
5  img=plt.imread('image1.png')         # récupération de 'image1.png'
6
7  plt.imshow(img)         # visualisation de l'image en couleur
8
9  plt.show()              # affichage de l'image
```

- En utilisant la fonction **`type()`**, de quel type est la variable **`img`** ?

- Comment pouvez-vous expliquer la signification de les captures d'écran suivantes :

```
x=778. y=52.
[1, 1, 1]
```

```
x=39. y=578.
[0, 0, 0]
```

b) Récupération des caractéristiques

- **d'une image**
  Compléter le script précédent en ligne 7 avec : **`print(img.shape)`**
- **d'un pixel**
  Compléter le script précédent en ligne 8 avec : **`print(img[200,300])`**
  Compléter le script précédent en ligne 9 avec : **`print(img[200,300,0])`**

  Que permettent de faire les instructions **`img.shape`**, **`img[200,300]`** et **`img[200,300,0]`** ?

c) Création d'une image

Une image est assimilée à un tableau à trois dimensions.

Pour créer une image sous python il faut donc dans un premier temps créer un tableau.

Cela se fait grâce à la fonction **`zeros([l,c,p])`** où :

- **`l`** est le nombre de lignes voulues
- **`c`** est le nombre de colonnes voulues
- **`p`** est la profondeur voulue :
  - **`p = 1`** : image en niveaux de gris
  - **`p = 3`** : image en couleurs sans transparence en codage Rouge Vert Bleu (RVB) pour un format d'image .png
  - **`p = 4`** : image en couleurs sans transparence

Exécutez le script suivant :

```
1  import numpy as np
2
3  import matplotlib.pyplot as plt
4
5
6  imageX=np.zeros([5,5,3])
7
8  for i in range(0,5):
9      imageX[i,i,0]=1
10     imageX[i,4-i,1]=1
11
12 plt.imshow(imageX)
13
14 plt.show()
```

- Au vu des résultats, commentez les lignes 6 et 8 à 10.

- Au vu des résultats, quelles sont les caractéristiques de la couleur jaune ?

d) Sauvegarde d'une image

Compléter le script précédent en ligne 11 avec : **`imsave("imageX.png",imageX)`**
Cela permet d'enregistrer en couleur l'image (de type array) `imageX` sous un format classique d'image (.png) dans le répertoire de travail (le même que celui dans lequel vous avez sauvegardé votre programme).

2. Modification d'une image

a) Convertir une image en couleurs en niveaux de gris

i) Compléter le script suivant en proposant une fonction **nivgris** permettant de convertir une image couleurs en niveaux de gris.
Une méthode intuitive consiste à faire la moyenne des 3 couleurs (R, V, B) de chaque pixel et ainsi obtenir la valeur du nouveau pixel en niveaux de gris.

```
import numpy as np

import pylab as plt   # importation de la bibilothèque  matplotlib.pyplot

def nivgris(img):









    return imgris

## Programme principal

img=plt.imread('image1.png')

imgris=nivgris(img)

print(imgris.shape)

plt.imsave("nivgris.png",imgris) # sauvegarde de l'image dans le répertoire de travail

plt.imshow(imgris)    # visualisation de l'image

plt.show()    # affichage de l'image
```

Exécuter le script. Que remarquez-vous ?

ii) La sensibilité de l'œil humain est plus grande à la couleur verte qu'au bleu. La pondération reste un peu arbitraire. La commission internationale de l'éclairage propose d'obtenir la luminance (valeur du gris) d'un pixel à partir de ses composantes RVB par la formule :

$$Y = 0{,}2125 \times R + 0{,}7154 \times V + 0{,}0721 \times B$$

Modifier le script précédent en proposant une fonction **nivgrisnaturel** permettant de convertir une image couleurs en niveaux de gris en suivant la méthode décrite ci-dessus.

b) Réduction d'une image

Compléter le script suivant en proposant une fonction permettant de créer une image couleurs quatre fois plus petite en ne gardant qu'un pixel sur quatre.

Un conseil faites un dessin avant de commencer avec une image de taille 6 sur 6.

```
import numpy as np

import matplotlib.pyplot as plt

def reduction(img):









## programme principal

img=plt.imread('image1.png')

reduc=reduction(img)

plt.imsave("reduction.png",reduc)   # sauvegarde de l'image

plt.imshow(reduc)                  # visualisation de l'image

plt.show()                        # affichage de l'image
```

c) L'inversion de couleur dans le système RVB

Il faut pour cela extraire l'information des trois couleurs dans le système RVB et les symétriser dans l'ensemble des valeurs qu'elles décrivent (entre 0 et 1)
Pour cela il suffit de remplacer la couleur d'un pixel `[a,b,c]` par `[1-a,1-b,1-c]`
Compléter le script suivant en proposant une fonction contraste permettant de créer une image avec les couleurs inversées.

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

def contraste(img):




## Programme principal

img=plt.imread('image1.png')

imcontrast=contraste(img)

plt.imsave("contraste.png",imcontrast,cmap="gray")

plt.imshow(imcontrast)

plt.show()
```

d) Compléments

i) **Noir et blanc**

Créer une fonction NB() prenant en argument une image et qui en crée la version en noir et blanc. Pour ce faire, on choisira judicieusement une valeur frontière (ou seuil).

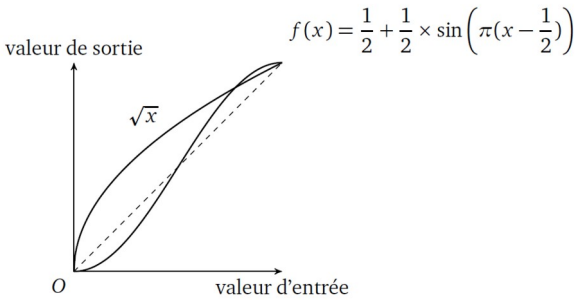
ii) **Amélioration du contraste**

Pour améliorer le contraste, on transforme le niveau de gris d'entrée en un nouveau niveau de gris dans le but d'exagérer les différences entre les niveaux. Pour cela on applique une fonction « bien choisie ».

a. Améliorer le contraste en utilisant une fonction racine carrée : on remplace chaque niveau d'entrée x par x.

b. Comparer avec l'usage d'un courbe « en S »(un morceau de sin convenablement` transformé).

c. La transformation précédente améliore-t-elle le contraste de l'image.



iii) **Flou et netteté**

Créer une fonction flou() prenant en argument une image et qui en renvoie une version floue.
Pour ce faire, on attribue à chaque composante d'un pixel la moyenne des composantes des neuf pixels voisins. On ne modifiera pas les pixels situés sur le bord.
Plus généralement, pour flouter une image, ou au contraire en accentuer les détails, la technique la plus simple consiste à remplacer la valeur de chaque pixel par une valeur calculée sur une petite fenêtre (3×3 typiquement) centrée autour de ce pixel. Ce calcul est basé sur une matrice qui varie suivant l'effet que l'on veut obtenir.

Par exemple, en prenant $M = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$, on remplacera la valeur d'un pixel par la moyenne coefficientée des valeurs pour les 9 pixels alentour, affectées des coefficients donnés par la matrice.
Écrire une fonction applique_filtre(im,filtre) qui prend en entrée une image et un filtre sous la forme d'une matrice 3×3 et effectue l'opération décrite ci-dessus.
On pourra essayer avec la matrice M (qui produira un flou assez léger).

Tester la fonction applique_filtre pour différentes matrices M afin d'augmenter le flou ou la netteté.