TP n° 08 – Découverte du traitement d'images et manipulation de tableaux numpy

I Traitement d'une image très simple

L'image matricielle (ou bitmap) : Elle est composée de petits points appelés « pixels » que l'on ne voit pas forcément à l'œil nu. Lors de l'agrandissement d'une image matricielle, cette dernière devient floue car les pixels ressortent, ce sont les carrés qui apparaissent sur l'écran.
La couleur de chaque pixel est codée en code Rouge, vert, bleu, abrégé en RVB ou en RGB (de l'anglais « red, green, blue »), car ce codage est proche du matériel. Les écrans d'ordinateurs reconstituent une couleur par synthèse additive à partir de trois couleurs primaires, un rouge, un vert et un bleu, formant sur l'écran une mosaïque trop petite pour être aperçue. Le codage RVB indique une valeur (de 0 à 1 dans ce TP, mais peut aussi être de 0 à 255) pour chacune de ces couleurs primaires.
La matrice qui permet de définir l'image doit être mise sous la forme d'une `array` numpy. Le type `array` est similaire à une liste (`list`) avec la condition supplémentaire que tous les éléments sont du même type. Nous concernant ce sera donc un tableau d'entiers (`int`) ou de flottants (`float`). Il sera alors possible d'accéder à une valeur du tableau `numpy` directement à l'aide d'un `tuple ligne,colonne`.

I.1 Manipulation d'une array numpy

Tester le code suivant :

```
import numpy as np
liste=[[1,2],[3,4]]
array=np.array(liste)

print(liste[0][1],array[0][1],array[0,1])
```

I.2 Manipulation d'une image simple

Le code suivant permet de générer une image de 2x2 pixels.

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

# Création d'une image avec 4 pixels
imagebase = np.ones([2,2,3])
imagebase[1,1] = [0.4,0.6,0.8]
imagebase[0,0] = [0.8,0.4,0.2]
plt.imshow(imagebase)
plt.show()
```

Exercice 1. Modifier ce code pour faire apparaître les éléments `imagebase[0]`, `imagebase[0][0]` et `imagebase[0][0][0]`. Identifier à quoi ils correspondent.

Exercice 2. Modifier ce code pour modifier les couleurs de l'image comme vous le souhaitez.

I.3 Gestion de la transparence

En ajoutant une 4ème valeur pour chaque pixel après le RVB, il est possible de gérer la transparence de l'image comme pour un format PNG par exemple. On parle alors de codage RVBA. En infographie, RVBA (sigle signifiant Rouge Vert Bleu Alpha) ou RGBA (Red Green Blue Alpha) est une extension du format de codage des couleurs RVB qui lui ajoute un canal alpha qui détermine l'opacité, pour calculer une image numérique composée de calques virtuels, superposés.
Par exemple, le code suivant génère une image avec 4 pixels noirs, mais 4 valeurs de transparence différentes.

```
# Création d'une image avec 4 pixels
imagebase = np.zeros([2,2,4])
imagebase[0,0,3] = 1
imagebase[0,1,3] = 0.75
imagebase[1,0,3] = 0.5
imagebase[1,1,3] = 0.25
plt.imshow(imagebase)
plt.show()
```

Exercice 3. Modifier le code précédent pour avoir des pixels à 10% d'opacité à gauche et 90% d'opacité à droite.

II Traitement d'une image plus complexe

Nous allons dans la suite ouvrir une image au format PNG. Sa structure sera strictement la même que celle de l'image précédente, il y aura évidemment beaucoup plus de pixels. L'image est disponible dans le dossier « Ressources/PTSI/TP/TP08 ».
L'image choisie est un tableau de **David Hockney** intitulé *Portrait of an Artist.*

FIGURE 1 – Portrait of an Artist - David Hockney

II.1 Ouverture de l'image

```
image = img.imread('David_Hockney_Portrait_of_an_Artist.png')
plt.imshow(image)
plt.show()

print(image.shape)
```

Exercice 4. Après avoir exécuté ce code, expliquer le résultat de la ligne `print(image.shape)`.

II.2 Extraction des couleurs

Le code suivant permet d'afficher 3 images sur lesquelles seule la composante bleue de l'image originale apparaît.

```
def extraire_couleur(image,id_couleur):
    image_out=image.copy()
    image_out[:,:,0]=0
```

```
    image_out[:,:,1]=0
    return image_out

f, axarr = plt.subplots(1,3)
for i in range(3):
    axarr[i].imshow(extraire_couleur(image,i))
plt.show()
```

Exercice 5. Proposer une modification de ce code afin qu'il afficher 3 images sur lesquelles apparaissent respectivement les composantes rouge, verte et bleue. Il faudra automatiser cette tâche en utilisant une boucle `for`.

II.3 Inversion des couleurs

Il est possible d'inverser les couleurs d'une image afin d'obtenir son « négatif ». Le principe est de prendre le complément à 1 de chacune des valeurs des couleurs. Ainsi, si une couleur d'un pixel est affectée de la valeur a, il faut lui attribuer la valeur $1-a$ pour inverser l'image.
La solution suivante a été envisagée afin de réaliser cette opération, elle consiste à utiliser un tableau rempli de 1 de la taille de l'image et de lui soustraire l'image. Le résultat n'est pourtant pas celui attendu.

```
def inverser_couleur(image):
    image_out=image.copy()
    return np.ones(np.shape(image_out))-image_out

f, axarr = plt.subplots(1,2)
axarr[0].imshow(image)
axarr[1].imshow(inverser_couleur(image))
plt.show()
```

Exercice 6. Modifier le code afin de corriger cet erreur, cela permettra d'afficher l'image et son inverse.

II.4 Conversion en niveaux de gris

Une possibilité de conversion en niveaux de gris s'effectue en faisant la moyenne des trois couleurs et en affectant cette moyenne aux trois couleurs du pixel. Par exemple, si le codage RVBA d'un pixel est $[a,b,c,1]$, alors, sa conversion en niveaux de gris donnera le résultat suivant $\left[\frac{a+b+c}{3},\frac{a+b+c}{3},\frac{a+b+c}{3},1\right]$.
Le code suivant permet d'afficher ce résultat.

```
def convert_nb_base(image):
    image_out=np.zeros(np.shape(image))
    image_out[:,:,3]=1
    for j in range(3):
        for i in range(3):
            image_out[:,:,j]+=image[:,:,i]/3
    return image_out

plt.imshow(convert_nb_base(image))
plt.show()
```

Le résultat n'est pourtant pas optimal car les zones sombres sont trop « noires ». Il a été montré qu'une meilleure solution consiste à utiliser la répartition suivante : $0.2125*rouge+0.7154*vert+0.0721*bleu$, on constate que $0.2125+0.7154+0.0721=1$.

Exercice 7. Créer une fonction `convert_nb_evol` qui prend en compte cette répartition et l'utiliser afin de convertir l'image en niveaux de gris. Cette fonction devra être optimisée au maximum grâce à l'utilisation de boucles `for`. Afficher les résultats des deux solutions côte à côte afin de valider le résultat.

II.5 Contraste et luminosité

La fonction Contraste et luminosité améliore l'apparence d'une image. La luminosité améliore la clarté globale de l'image (par exemple, pour rendre plus claires des couleurs sombres et pour rendre plus pâles des couleurs claires). Le contraste règle la différence entre les couleurs les plus sombres et les couleurs les plus claires.
Pour modifier ces paramètres, il faut appliquer la formule suivante à chaque valeur de couleur de chaque pixel : valeur=contraste*valeur+luminosite, avec `contraste` et `luminosite` des valeurs à modifier selon le réglage souhaité.
Attention, il faut veiller à ce que la valeur issue de ce calcul soit toujours comprise entre 0 et 1, sinon il y a aura une erreur à l'exécution. Pour vous aider à cela, vous pouvez aller regarder sur internet la fonction `numpy.clip`.

Exercice 8. Créer une fonction `contraste_luminosite` qui permet d'appliquer cet effet à l'image. Afficher l'image initiale et l'image modifiée côte à côte afin de voir l'impact de cet effet.

II.6 Rotation de l'image de 90°

Le code suivant permet d'effectuer une rotation de 90° de l'image.

```
def rotation(image):
    return np.rot90(image)

image_rot=image.copy()
f, axarr = plt.subplots(1,4, gridspec_kw={'width_ratios': [3/2, 1,3/2, 1]})
for i in range(4):
    axarr[i].imshow(image_rot)
    axarr[i].title.set_text('Rotation {}°'.format(i*90))
    image_rot=rotation(image_rot)
plt.show()
```

Le principe de la rotation d'une image est présenté sur la figure 2.

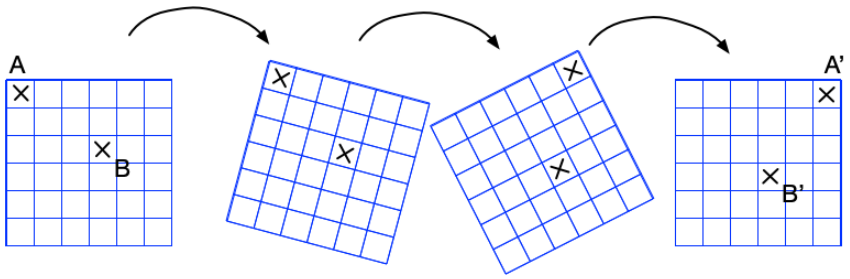

FIGURE 2 – Principe de la rotation d'une image pixelisée

Exercice 9. D'après le principe présenté sur la figure 2, proposer une solution pour coder vous-même la fonction `rotation` sans l'utilisation de `np.rot90`.

II.7 Réduire la taille d'une image

Il est parfois utile de réduire la taille d'une image si celle-ci est trop grande pour être manipulée, traitée ou transmise. Le principe consiste à supprimer des pixels.
La fonction suivante permet de le faire.

```
def reduire(image, facteur):
    nb_lig,nb_col,nb_coul=image.shape
    image2=np.zeros((nb_lig//facteur,nb_col//facteur,nb_coul))
    for i in range(nb_lig//facteur):
        for j in range(nb_col//facteur):
            for k in range(nb_coul):
```

```
                image2[i,j,k]=image[i*facteur,j*facteur,k]
    return image2

plt.imshow(reduire(image,5))
plt.show()
```

II.8 Augmenter la taille d'une image

Il peut aussi être intéressant d'augmenter la taille d'une image. Attention, cela n'augmente pas sa qualité car l'information ajoutée est déterminée à partir de l'image de petite taille. L'idée consiste à ajouter des pixels entre les pixels existants et de déterminer les valeurs de leurs couleurs à partir des caractéristiques des pixels proches.

Exercice 10. S'inspirer de la fonction `reduire` afin de créer une fonction `agrandir` qui permet d'augmenter la taille d'une image.

II.9 Détection de contours

La détection de contours est une technique qui peut être utilisée dans le traitement d'image, afin de faire de la détection d'objets,...
Cela consiste à détecter les gros écarts de contraste entre des pixels à proximité. Pour cela, pour chaque pixel, on regarde la valeur des niveaux de gris de celui décalé de 2 vers la gauche ($p1$), deux en dessous ($p2$), deux vers la droite ($p3$) et deux au-dessus ($p4$), et on calcule la norme $\sqrt{\frac{(p1-p3)^2+(p2-p4)^2}{2}}$. Si cette norme est plus grande qu'une valeur `seuil` à définir, on trace un pixel noir sur une nouvelle image blanche. Une fois tous les pixels parcourus les contours apparaissent sur cette image. Il peut alors être intéressant de modifier la valeur `seuil` afin de faire varier la sensibilité aux contours.

Exercice 11. Créer la fonction `contour` qui renvoie une image blanche avec les contours en noir dont le fonctionnement est décrit dans le paragraphe précédent.