Traitement images (sans bibliothèque d'image)

1. Représentation et manipulations élémentaires

[1]. pixel : acronyme de PICTure ELement, élément d'image

1.1 Codage des images

✓ *Mosaïque de Pixels*

Une image est représentée sur les supports numériques, par une mosaïque de pixels[1] disposés en matrice de L lignes et C colonnes. La valeur de chaque pixel décrit la luminosité émise par le pixel.

✓ *Images Noir&Blanc*

Chaque pixel d'une image Noir& Blanc (plus exactement une image en niveaux de gris) est en général codé comme un entier non signé sur un octet (8 bits), le 0 codant le noir et le 255 le blanc (le maximum de lumière). Selon les modules utilisés, ce codage pourra être converti en un float compris entre 0 et 1.

✓ *Images Couleur*

Chaque pixel d'une image Couleur est en général codé par 3 entiers non signés, chacun codé sur un octet (8 bits), représentant dans l'ordre les intensités de chacune des 3 couleurs : dans l'ordre, Rouge, Vert, Bleu. Pour chaque couleur, le 0 code l'absence de luminosité, et le 255 le maximum d'intensité. Selon les modules utilisés, ce codage pourra être converti en 3 floats, chacun compris entre 0 et 1.

Première partie : utilisation de Numpy

1.2 Génération d'images à partir de tableaux numpy

On utilisera numpy pour manipuler les images comme des tableaux de valeurs. Pour ouvrir les images et les afficher, on utilisera le module matplotlib.pyplot. Pour importer ces modules :

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

On rappelle l'instruction vue en première année pour créer une matrice numpy de taille (L,C), pleine de zéros :
I=np.zeros((L,C)),
Il est pratique d'afficher simultanément plusieurs images dans une figure avec la fonction plt.subplot(). Les instructions suivantes affichent 2 images im1 et im2 dans chacun des 2 subplots d'une figure :

```
plt.close() # ferme la dernière figure
plt.figure(1) # ouvre un nouvelle figure
plt.subplot(211) #2 lignes,1 colonne,plot 1
plt.imshow(im1,interpolation="nearest")
plt.subplot(212) #2 lignes,1 colonne,plot 2
plt.imshow(im2,interpolation="nearest")
plt.show() # rend l'affichage effectif
```

Remarques

– l'argument interpolation="nearest" spécifie à plt de ne pas faire d'interpolation entre les pixels (utile lorsque la résolution image est différente de celle de l'écran).

– Par défaut, imshow normalise les données par leur min et leur max. On peut désactiver cette normalisation en spécifiant manuellement les bornes, en ajoutant à l'appel de imshow les arguments vmin et vmax . Essayer à l'occasion.

2.1 Des formes rectangulaires

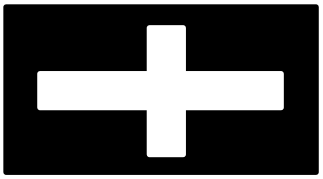
Question 1. Créer un tableau de zéros nommé im1 de taille L × C : avec L = 100 et C = 150. Exécuter la commande im1[20:61,100:121]=1, ce qui met à 1 tous les pixels (l,c), tels que l ϵ [20,60] et c ϵ [100,120]. Afficher en niveau de gris en utilisant la syntaxe suivante :

```
plt.imshow(im1 ,cmap = 'gray' ,interpolation="nearest",vmin=0, vmax=1)
```

Question 2. Créer un tableau de zéros de taille L×C×3, nommé im2. Exécuter la commande im2[40:61,20:131,0]=1, ce qui met à 1 la 1ère coordonnée (le Rouge) de tous les pixels (l,c), tels que l ϵ [40,60] et c ϵ [20,130]. Mettre à 1 la 2ème coordonnée (le Vert), de tous les pixels

(l,c), tels que l ∈ [20,80] et c ∈ [40,60]. Mettre à 1 le Bleu de tous les pixels (l,c), tels que l ∈ [20,80] et c ∈ [100,120]. Afficher im2 comme une image couleur, et im3=im2[:,:,0] comme une image de niveaux de gris: Que représente cette dernière?

Question 3. Représenter la figure suivante :



2.2 Des images aléatoires

Exécuter (avec par exemple L = 5 et C = 8) la commande imrand=np.random.rand(L,C,3)

Question 3. Que retourne cette commande ?

Question 4. Afficher imrand en couleur, et le plan rouge en niveaux de gris. (entier de 0 à 255)

Question 5. Sur quel type de variable est codé chaque pixel ?

On trouve l'information soit dans l'espace de travail (workspace), soit dans l'attribut dtype de imrand.

2.3 Des images sinusoïdales

Les pixels seront codés à partir des deux relations suivantes :

$$p(l,c) = \frac{1}{2}\left(1 + \sin\left(\frac{l}{\lambda_l} + \frac{c}{\lambda_c}\right)\right) \qquad (1)$$

$$p(l,c) = \frac{1}{2}\left(1 + \sin\left(\frac{l}{\lambda_l}\right)\sin\left(\frac{c}{\lambda_c}\right)\right) \qquad (2)$$

Question 6. En vue d'afficher une image de niveaux de gris, créer d'abord un tableau de zéros nommé im de taille L×C avec L = 100, C = 100.

Parcourir chaque pixel p(l; c) de ce tableau, pour lui affecter une valeur (son niveau de gris) conforme à la relation (1), où λl et λc sont les longueurs d'ondes respectives en ligne et en colonne. On prendra par exemple $\lambda_l = L/3$ et $\lambda_c = C/4$. Afficher le résultat. Même travail pour la relation 2

2.4 Animer une image

Question 7. Animer cette image, en insérant les calculs de la question précédente et leurs affichages dans une boucle, qui fera par exemple varier λc de 2C à C/5 par pas de C/10, ou encore qui déphasera l'onde à chaque itération. Pour que chaque image soit affichée, il faut insérer une petite pause dans la boucle :

```
plt.imshow(im ,cmap = 'gray' )
plt.show()
plt.pause(0.04) # temps de pause (s)plt.imshow(im ,cmap = 'gray' )
```

Quelques images sont disponibles dans le répertoire images. Les instructions suivantes permettent d'ouvrir une image dans l'espace de travail de python, sous forme d'un tableau numpy , sous le nom im, puis de l'afficher :

```
im = plt.imread('images/riz_nb.png')
plt.imshow(im,interpolation="nearest")
plt.show() # rend l'affichage effectif
```

Ouvrir dans python une image contenue dans le répertoire images.

Question 8.

Quelles sont les dimensions du tableau im ? Quel est le type des données de chaque pixel ? Quelles données indiquent s'il s'agit d'une image de gris ou des couleurs ?

Conversion d'une image couleur en gris. La conversion des intensités de couleur de chaque pixel (notées R, V , B pour Rouge, Vert, Bleu) en une seule intensité lumineuse (le niveau de gris noté I) pour le Noir & Blanc, est tout un art, et de nombreux algos existent.On se contentera d'une conversion très simple : I = (R+V +B)/3, et afficher les 2 images.

Ouvrir l'image couleur 'capture.png' contenue dans le répertoire images.

Question 9. Convertir les intensités de couleur de chaque pixel (notées R, V , B pour Rouge, Vert, Bleu) en une seule intensité lumineuse (le niveau de gris noté I). L'afficher en niveaux de gris.

Traitement d'images (Utilisation de labibliothèque PIL)

Dans le TP, on découvre quelques traitements d'images classiques, agissant sur l'orientation, la couleur, la résolution... autant de fonctions pré-programmées dans n'importe quel logiciel permettant de retoucher des images.
Nous travaillerons avec des images en 256 niveaux de gris, c'est-à-dire que les images sont représentées dans l'ordinateur par un tableau d'entiers de 0 à 255, 0 pour le noir et 255 pour le blanc. Les valeurs entières de l'intervalle [0; 255] sont autant de niveaux de gris, allant du noir au blanc.
Nous utilisons les images suivantes, au format bitmap ou jpeg, que vous trouverez dans le dossier de TP :
'lena.bmp', 'insecte.bmp', 'debut.bmp', 'fin.bmp', 'original.bmp', 'erreur.bmp',
'Mona_Lisa.jpg'
Commandes fondamentales sur les images :

- import PIL
  from PIL import Image # Importation du module Image de PIL
- img=Image.open('nomImage.extension ') # permet d'importer une image dans un tableau
- img.size # permet d'accéder à la taille de l'image, renvoie un couple (largeur, hauteur)
- img.mode # permet de connaitre le type d'image
- nouvelleimg=Image.new(mode,size) # crée en mémoire un tableau correspondant à un type d'image
- img.getpixel((x,y)) # renvoie la valeur du pixel de coordonnées (x,y) (x : indice de colonne, y : indice de ligne)

pour une image en 256 niveaux de gris, renvoie un entier entre 0 et 255
pour une image en RVB, renvoie un tuple (R,V,B) où R, V et B sont les niveaux de couleurs de 0 à 255 dans notre cas.

Rappel : on ne peut pas modifier les tuples, il faut d'abord les convertir en liste.

- list((1,2,3)) # convertit un tuple en liste [1,2,3]
- tuple([1,2,3]) # convertit une liste en tuple (1,2,3)

- img.putpixel((x,y),valeurpixel) # donne au pixel de coordonnées (x,y) la valeur
- valeurpixel pour une image en 256 niveaux de gris
- img.save('nomImage.extension') # sauvegarde l'image dans un fichier

Préliminaires :
Créer dans vos documents personnels un fichier TPXX, dans lequel vous enregistrerez vos scripts.
Il convient de placer les photos dans ce dossier pour leur appliquer les scripts programmés.
Il est conseillé de travailler avec Spyder, qui se comporte mieux vis-à-vis des images.
Si vous devez travailler dans l'interpréteur de Pyzo, installez le module Pillow en tapant : pip install Pillow

Exercice n°1 Un premier script "mystère"

```
import PIL
from PIL import Image
def TaQ(fichier,newname) :
    img=Image.open(fichier)
    newimg=Image.open(fichier)
    largeur=img.size[0]
    hauteur=img.size[1] # on peut écrire directement : largeur, hauteur=img.size
    for x in range(largeur):
        for y in range(hauteur):
            p=img.getpixel((x,hauteur-y-1))
            newimg.putpixel((x,y),p)
```

```
newimg.save(newname)
```

1. Recopier et tester ce programme sur l'image "lena.bmp" (ne pas oublier le '.bmp' lors de l'appel de la fonction pour fichier et newname)
2. Déterminer le nombre de pixels de l'image "lena.bmp"
3. Modifier ce script pour créer un fichier newname par une rotation d'angle 180°.

Le script TaQ est un modèle pour les programmes à créer, utilisez le copier-coller sans modération.

Exercice n° 2 Image en négatif

L'idée est de remplacer, dans un _chier image, tous les pixels de valeur n par leur complément à 255. On construit ainsi un fichier image en négatif.

1. Programmer, puis tester, une fonction 'negatif' qui prend en entrées un fichier image et un nouveau_nom de fichier et qui renvoie l'affichage de l'image en négatif, sauvegardée dans le fichier nouveau_nom.

Exercice n° 3 Détection de mouvement et jeu des erreurs

Une caméra capture des images toutes les 0.5 secondes ; comment déterminer si il y a eu un mouvement entre 2 images consécutives ?

Le principe est d'effectuer une soustraction pixel par pixel entre 2 images consécutives, et d'afficher le résultat. Si le résultat de la soustraction est nul, laisser le pixel à 0, sinon donner comme valeur 255.

Vous obtiendrez alors une image en noir et blanc.

1. Programmer une fonction soustraction effectuant cette action.
2. La tester sur 'original.bmp' et 'erreur.bmp'.
3. Améliorer la fonction pour éviter les défauts visibles à la question précédente

Exercice n° 4 Contraste

Le principe est d'étendre ou de réduire l'amplitude entre la plus petite valeur et la plus grande valeur de niveau de gris. On peut utiliser différents types d'interpolation, mais ici, on se limitera à une interpolation linéaire.

1. Écrire une fonction minmax, de paramètre un fichier image, et renvoyant un couple contenant la valeur minimale m et la valeur maximale M de niveau de gris de l'image.

On pourra utiliser les fonctions min() et max() qui, appliquées à une liste de flottants, renvoie respectivement leur minimum et leur maximum.

2. Une petite question de maths :

Soit a < b deux entiers entre 0 et 255. Donner l'expression de la fonction affine f vérifiant :
f(m) = a et f(M) = b.

- Programmer cette fonction f sous Python, en identifiant clairement les entrées et les sorties.

3. Programmer alors une fonction contraste qui prend en entrées :
   - un fichier image,
   - un nouveau_nom de fichier,
   - les paramètres a et b,

et qui renvoie l'affichage de l'image contrastée, sauvegardée dans le fichier nouveau_nom.

4. Tester votre fonction sur l'image 'lena.bmp', avec a = 0 et b = 255, puis a = 100 et b = 255, puis a = 0 et b = 150. Décrire l'action observée.

Exercice n° 6 Filtre couleur

Pour chaque pixel de l'image RVB, on garde la valeur de la couleur choisie et on met à zéro les 2 autres couleurs. Par exemple, pour effectuer un filtre rouge, le pixel deviendra (valeur de R, 0 , 0). Écrire une fonction 'filtrecouleur' prenant en paramètres une image RVB et une couleur sous forme de chaîne de caractères R, V ou B et renvoyant l'image filtrée. A tester sur l'image 'monalisa'.

Exercice n° 7 Pixellisation

Le principe est de découper l'image en carrés de 2 × 2 et de remplacer la valeur de chaque pixel de ce carré par la moyenne des valeurs de chaque pixel de ce carré.

1. Écrire une fonction pixel2 prenant en paramètres un fichier image et un nouveau_ nom, et renvoyant l'image pixellisée sauvegardée.

Tester votre fonction sur l'image 'lena.bmp'.

Remarque : on pourra laisser inchangés les pixels de certains bords, pour faciliter l'algorithme.

2. Généralisation : on peut augmenter la taille des carrés à 3 × 3, ou davantage, et pixelliser suivant cette nouvelle taille n.

Écrire une fonction pixel prenant en paramètres un fichier image, un entier naturel n et un nouveau_ nom, et renvoyant l'image pixellisée sauvegardée.