Manipulation des images avec Pyzo

La manipulation des répertoires d'utilisation de Pyzo.

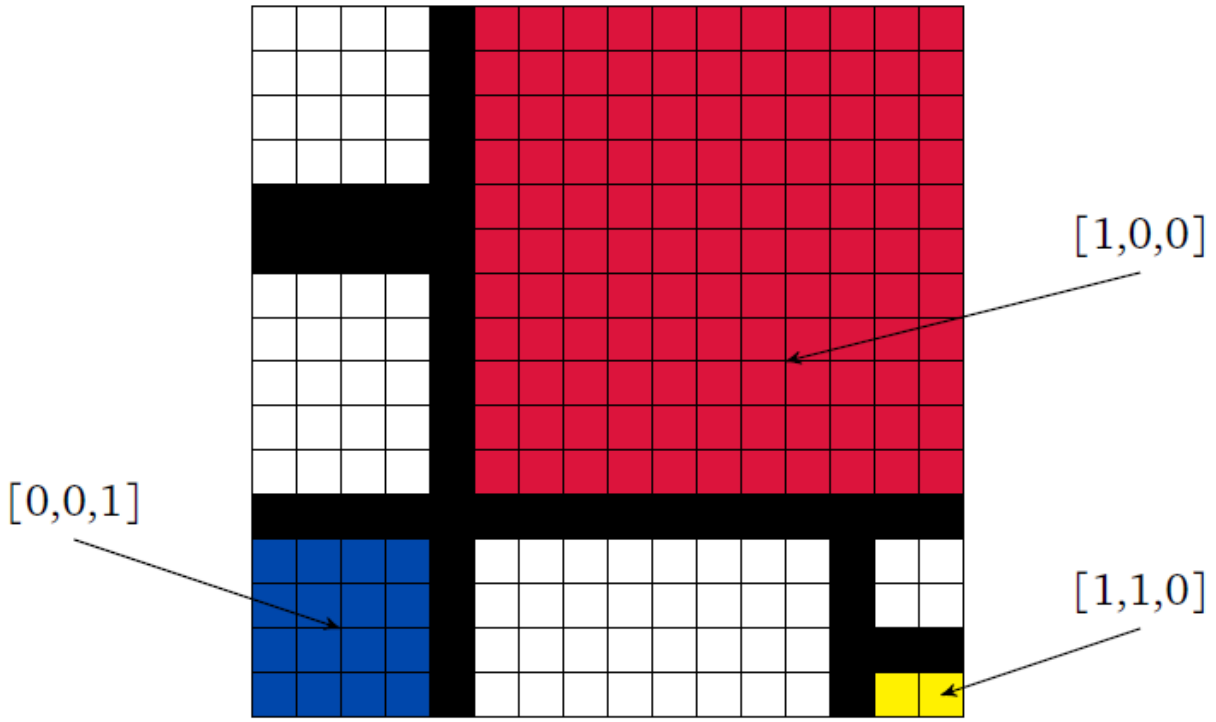
```
import os
os.chdir("C:\\Users\\Maison\\Desktop") # pour changer de répertoire
```

Cette fiche va utiliser principalement le module Pylab présent dans la distribution Pyzo.

```
from pylab import *
```

Format numérique d'image

Une image peut être numérisée sous forme d'image matricielle (en anglais « bitmap ») par une matrice de points colorés.
Cette matrice a *n* lignes (la hauteur de l'image) et *p* colonnes (la largeur).
L'élément (i, j) représente un pixel, c'est-à-dire une portion d'image considérée de couleur constante.

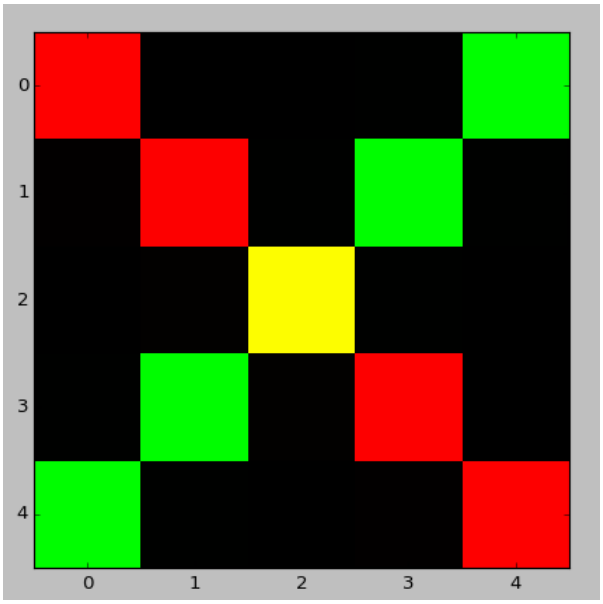


Il existe plusieurs manières de coder la couleur. La méthode accessible avec la bibliothèque matplotlib.pyplot est la méthode RGB pour «Red-Green-Blue ». Chaque couleur est représenté par une liste à trois entrées [r,g,b] où r, g et b sont trois réels représentant respectivement la quantité de rouge, de vert et de bleu que contient la couleur. La méthode de mélange des couleurs est la synthèse additive : on peut penser à trois spots de couleurs qui éclairent un fond noir. Le mélange des trois lumière crée la couleur désirée.
Si on appelle A la matrice associée à l'image ci-dessus, elle est de taille $16 \times 16 \times 3$ et le pixel en bas à droite de l'image correspond au triplet [A[15,15,0],A[15,15,1],A[15,15,2]] c'est-à-dire ici à [1,1,0].

Construction d'une image

Création d'une matrice 5×5×3 de nombres dans [0;1[: la bibliothèque Numpy le permet, en particulier l'instruction zeros([5,5,3]), puis l'utilisateur indique les valeurs du codage RGB attendues.

```
import numpy as np
A=np.zeros([5,5,3])
for i in range(0,5):
    A[i,i,0]=1
    A[i,4-i,1]=1
```



Préparation des propriétés de l'image :

```
figure(1)
imshow(A, interpolation='none')
# sinon, prendre interpolation='nearest'
grid(True)
```

Affichage de l'image :

```
show()
```

Quelles couleurs ont été données sur la figure ci-dessus ?

Chargement, modification et sauvegarde d'une image

Chargement de l'image dans Pyzo (de type png) :

```
image = imread('plage.png')
```

En appliquant la commande « print » à l'image plage.png, print(image), vous verrez que l'image est stockée sous forme d'une liste de dimension 3 composée de flottants entre 0 et 1. Les deux premières dimensions donnent les coordonnées x et y des pixels, et la troisième dimension contient les niveaux des trois couleurs RGB, codées sur un octet (soit un entier entre 0 et 255 pour chaque couleur, ce qui permet d'obtenir plus de 16 millions de couleurs différentes), print(image*255) permet de retrouver les entiers associés aux couleurs primaires. La quatrième valeur, pour les images de type *.png donne la transparence du pixel.
Pour afficher les 3 couleurs RGB du pixel de coordonnées (20,40), taper :

```
print(image[20,40])
```

L'instruction suivante donnera la taille de l'image chargée.

```
print(image.shape)
print(image.shape[0]) #permet de connaître le nombre de lignes de l'image
print(image.shape[1]) #permet de connaître le nombre de colonnes de l'image
```

Comment transformer une telle image en niveau du gris? Quelques-uns pensent que on doit simplement prendre la moyenne des trois plans de couleurs : Rouge / Vert / Bleu

```
image2 = (image[:,:,0]+image[:,:,1]+image[:,:,2])/3.0
```

Mais ceci est inexact. 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 RGB par la formule :

$$\text{Gris naturel} = 0{,}2125 \times \text{Rouge} + 0{,}7154 \times \text{Vert} + 0{,}0721 \times \text{Bleu}$$

```
image2 = 0.2125*image[:,:,0] + 0.7154*image[:,:,1] + 0.0721*image[:,:,2].
```

Mais la visualisation de image2 par imshow() reste étonnante. Pour qu'un tableau 2D corresponde à une image, il faut lui associer une palette de couleur, ici "gray" pour le gris.

```
imshow(image2, cmap="gray")
colorbar() # dessiner la barre de niveau des nuances de gris
```

Pour copier ou sauvegarder l'image

```
image3=copy(image)
imsave("image3.png",image2,cmap="gray")
```

Une application : L'inversion de couleur dans le système RGB :
Il faut pour cela extraire l'information des trois couleurs dans le système RGB et les symétriser dans l'ensemble des valeurs qu'elles décrivent (entre 0 et 255).

```
image = imread('maths.png')
image2=copy(image)
for y in range (image.shape[1]) :
   #pour chaque colonne :
   for x in range (image.shape[0]) :
         # code du pixel (niveau de gris) puis inversion du niveau de gris :
         # création du pixel correspondant dans la nouvelle image :
         p1=image2[x,y,0];q1=1-p1;image2[x,y,0]=q1
         p2=image2[x,y,1];q2=1-p2;image2[x,y,1]=q2
         p3=image2[x,y,2];q3=1-p3;image2[x,y,2]=q3
         # une automatisation de ces trois lignes serait préférable.
imshow(image2, interpolation='none') # Par défaut : quadratique
imsave("image2.png",image2)
show()
```

Premiers exercices

1) Créer une image de taille adéquate permettant d'afficher le drapeau français.

2) Obtention d'une image en niveaux de gris ou en noir et blanc

a) Créer une fonction gris() prenant en argument une image et qui en crée la version en niveaux de gris. Pour ce faire, le niveau de gris attribué à chaque pixel est obtenu en suivant la recommandation de la commission internationale de l'éclairage.

b) 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).

3) Amélioration du contraste

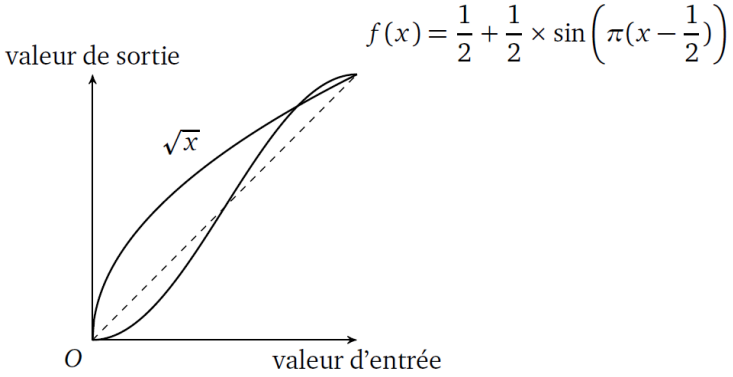
Pour améliorer le contraste, on transforme le niveau de gris d'entrée en un nouveau niveau de gris dans le but d'exagérer les différences entre les niveaux. Pour cela on applique une fonction « bien choisie ».

a) Améliorer le contraste en utilisant une fonction racine carrée : on remplace chaque niveau d'entrée x par $\sqrt{x}$.

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

c) Expliquer en quoi la transformation précédente améliore le contraste de l'image.

d) Quel type de fonction est susceptible d'être utile ?



4) Flou et pixellisation

a) 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) et avec $M' = \begin{pmatrix} -1 & -2 & -1 \\ -2 & 16 & -2 \\ -1 & -2 & -1 \end{pmatrix}$, qui produira une accentuation assez marquée des détails.

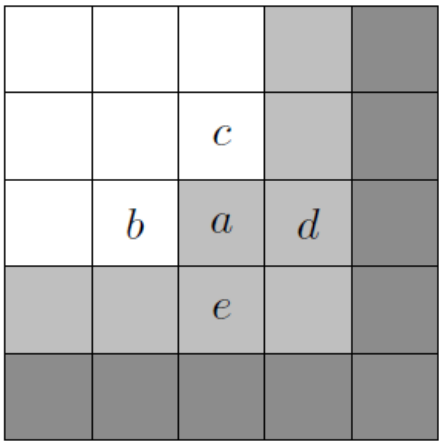
b) Créer une fonction pixellisation() prenant en argument une image et un entier c et qui en renvoie une version pixellisée. Pour ce faire :

- 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.

5) Détection des bords

Dans une image, les bords des objets correspondent à des zones où les valeurs des pixels changent rapidement. C'est le cas par exemple lorsque l'on passe d'un objet clair (avec des valeurs grandes) à un arrière-plan sombre (avec des valeurs petites).

Afin de savoir si un pixel avec une valeur a est le long d'un bord d'un objet de l'image, on prend en compte les valeurs *b*, *c*, *d*, et *e* de ses quatre voisins (deux horizontalement et deux verticalement).



On calcule une valeur L suivant la formule (gradient discret d'intensité) :

$$L = \sqrt{(b - d)^2 + (c - e)^2}$$

On peut remarquer que si $L = 0$, alors on a $b = d$ et $c = e$: le pixel n'est pas sur un bord de l'objet de l'image. Au contraire, si L est grand, ceci signifie que les pixels voisins ont des valeurs très différentes : le pixel central est donc probablement sur le bord d'un objet.

On transforme ensuite l'image en affichant du noir quand L est faible et du blanc quand L est elevé. Il y a donc une valeur de seuil à déterminer.

a) Composante horizontale du gradient :

Écrire une fonction **gradientH()** qui prend un tableau représentant une image en niveaux de gris comme argument et renvoie un tableau du gradient horizontal des pixels de l'image (comparaison absolue entre pixels des 2 colonnes voisines). On ne prendra que la différence à droite ou à gauche suivant si on se trouve sur la première colonne ou la dernière colonne de l'image.

b) Composante verticale du gradient :

De même, écrire une fonction **gradientV()** qui renvoie un tableau du gradient vertical des pixels de l'image (entre les 2 lignes voisines). On ne prendra que la différence en dessous ou au dessus suivant si on se trouve sur la première ligne ou la dernière ligne de l'image.

c) Appliquer ces deux fonctions à l'image B (image A en niveau de gris). Afficher l'image correspondant au gradient horizontal de l'image B, puis celle correspondant au gradient vertical de B.

Tous les contours ont-ils été détectés par l'un ou l'autre gradient ? Conclure.

d) Combiner le gradient horizontal et le gradient vertical pour obtenir une approximation de la norme du gradient global et construire le tableau rempli des valeurs des gradients normalisés (et donc des contours). Afficher l'image correspondante. Conclure.

e) Choisir une valeur seuil pour noircir les vrais contours et blanchir le reste.

Remarque : pour choisir la valeur seuil du pixel, on peut s'aider du tracé de l'histogramme des pixels de l'image avec la syntaxe suivante :

```
Nombre_pixels,valeur=np.histogram(image,bins=256)
plt.plot(valeur[ :-1],Nombre_pixels)
plt.show()
```