

La **définition d'une image** correspond à sa dimension, exprimée en pixels. Le pixel est la plus petite composante d'une **image** numérique. Il est possible de distinguer chaque pixel en grossissant fortement l'**image**. Logiquement, plus une **image** comporte de pixels, plus elle est détaillée, par exemple une image de définition 800 x 600 (800 par 600), signifie que cette image est composée de 800 pixels en largeur et de 600 pixels en hauteur, soit en tout $800 \times 600 = 480000$ pixels.

Un pixel est composé de trois parties : une partie rouge, une partie verte et une partie bleue. À chaque pixel on associe donc 3 couleurs : le rouge, le vert et le bleu. On parle du canal rouge, du canal vert et du canal bleu d'un pixel (on parle de système RVB ou RGB en anglais). La variation de l'intensité lumineuse de chaque canal permet d'obtenir un très grand nombre de couleurs.) La valeur de l'intensité lumineuse associée à chaque canal de chaque pixel d'une image est très souvent comprise entre 0 et 255 (256 valeurs possibles). On codera donc un pixel à l'aide d'un triplet de valeur (par exemple "247,56,98"). La première valeur donnant l'intensité du canal rouge, la deuxième valeur donnant l'intensité du canal vert et la troisième valeur donnant l'intensité du canal bleu.

Un pixel est tellement petit que notre œil superposera la partie rouge, la partie verte et la partie bleue du pixel, voilà pourquoi nous voyons des pixels de différentes couleurs.

Avant de commencer à écrire un programme qui nous permettra de travailler sur les pixels d'une image, il est nécessaire de préciser que chaque pixel a des coordonnées x,y.

Travailler avec des images en Python

Cette section a été mise à jour en fonction des méthodes actuellement utilisées pour manipuler et afficher des images.

Ici l'objectif sera toujours de transformer une image en tableau numpy, pour pouvoir ensuite la manipuler.

Lecture de l'image

PIL permet de lire une image enregistrée localement dans de nombreux formats (il n'est pas forcément disponible sur vos machines).

matplotlib.image ne permet que de charger des .png . Mais vous avez directement un tableau numpy (pas besoin de transformation).

Avec matplotlib.image

```
import matplotlib.image as mpimg
import numpy as np
img = mpimg.imread("monimage.png")
```

Le résultat, img est un tableau numpy. C'est parfois un tableau de float (entre 0 et 1) ou parfois un tableau d'octets. On peut être amenés à faire une transformation :

```
if img.dtype == np.float32: # Si le résultat n'est pas un tableau d'entiers
img = (img * 255).astype(np.uint8)
```

Avec PIL

```
from PIL import Image
import numpy as np
imgpil = Image.open("monimage.png")
# anciennement np.asarray
img = np.array(imgpil) # Transformation de l'image en tableau numpy
```

Affichage de l'image

Matplotlib permet d'afficher une image (si c'est un tableau numpy).

```
import matplotlib.pyplot as plt
plt.imshow(img)
plt.show()
```

Sauvegarde des images

Pour enregistrer des images, on utilise une méthode similaire au chargement. Matplotlib permet ainsi de sauvegarder directement un tableau numpy au format PNG uniquement. PIL permettra de sauvegarder dans n'importe quel format, pourvu qu'on ait transformé le tableau numpy en Image PIL.

Avec Matplotlib

```
import matplotlib.image as mpimg
mpimg.imsave("resultat.png", img)
```

Avec PIL

Avec PIL, il faut d'abord transformer le tableau numpy en image PIL.

```
from PIL import Image
imgpil = Image.fromarray(img) # Transformation du tableau en image PIL
imgpil.save("resultat.png")
```

Créer une image

Dans cette partie, nous allons générer des images en utilisant numpy. Pour créer une image en mode RGB, il faut créer un vecteur de 3 dimensions dont la première dimension est la hauteur, la seconde est la largeur et la troisième représente les canaux RGB (chaque canal est représenté par une valeur de type numpy uint8). Dans le script ci-dessous, nous allons créer un vecteur numpy de taille (100, 300, 3) dont les trois canaux ont tous la valeur 0 pour tous les pixels (ce qui représente une image en noir) :

```
hauteur = 100
largeur = 300
# mode RGB
canal = 3
new_array_RGB = np.zeros([hauteur, largeur, canal], dtype =np.uint8)
plt.imshow( new_array_RGB )
plt.show()
```

Remplissons maintenant le vecteur numpy avec des pixels de couleur rouge (rouge =255, vert = 0, bleu = 0), vert (rouge = 0, vert = 255, bleu = 0) et bleu (rouge = 0, vert = 0,bleu = 255). Pour réaliser cette tâche, nous décomposons notre vecteur en trois tranches. Les trois valeurs de chaque couleur sont diffusées sur toutes les lignes et colonnes de chaque tranche : la première tranche sera en rouge, la deuxième en vert et la troisième en bleu :

```
# rouge : première tranche 0, 100
new_array_RGB[:,:100,:] = (255, 0, 0)
```

```
# vert : deuxième tranche 100, 200
new_array_RGB[:,100:200,:] = (0, 255, 0)
# bleu : Troisième tranche 200, 300
new_array_RGB[:,200:,:] = (0, 0, 255)
plt.imshow(new_array_RGB)
plt.show()
```

Pour créer une image en nuance de gris (mode L : un seul canal), il faut seulement créer un vecteur à deux dimensions représentant la hauteur et la largeur de l'image. Il faut également ajouter un paramètre à plt.imshow() pour préciser que nous voulons afficher une image en nuance de gris :

```
hauteur = 100
largeur = 300
# mode L
new_array_grey = np.zeros([hauteur, largeur], dtype=np.uint8)
# Noir
new_array_grey[:,:100] = 0
# Gris
new_array_grey[:,100:200] = 150
# Blanc
new_array_grey[:,200:] = 255
plt.imshow(new_array_grey, cmap = 'Greys_r')
plt.show()
```

Enregistrer un vecteur NumPy sous forme d'image

Vous pouvez facilement faire le chemin inverse et créer une image PIL à partir d'un vecteur numpy en utilisant Image.fromarray() (Si le type de données du vecteur numpy n'est pas un entier une erreur se produira, il est donc nécessaire de le convertir en utilisant np.uint8()) :

```
PIL_image = Image.fromarray( np.uint8( image_array ) )
print(PIL_image)
```

```
PIL_image.save("/path/nom.png")
```

Histogramme des pixels

Nous affichons l'histogramme de notre image qui représente la répartition des pixels selon leur intensité (la fonction flatten() transforme le vecteur numpy en un autre vecteur numpy avec une seule colonne) :

```
plt.hist(image_array.flatten(), bins = 20, density = True , alpha = .5 , edgecolor = 'black', color = 'red')
plt.show()
```

Image en nuance de gris

Nous pouvons aussi facilement transformer l'image en nuance de gris. Il existe plusieurs façons de le faire, mais une façon simple est de prendre la moyenne pondérée des valeurs RGB de l'image originale. Dans le script suivant, nous utilisons les poids suivant : 0.3 pour le canal rouge, 0.5 pour le canal vert et 0.2 pour le bleu. Vous pouvez changer les poids et voir leur effet sur l'image en nuance de gris générée.

```
image_array = mpimg.imread("g:\rss.png")
L = image_array.shape[0]
H = image_array.shape[1]
# Créer un vecteur numpy de taille (L, H)
array_grey = np.zeros((L,H))
# Moyenne pondérée
for i in range(L):
    for j in range(H):
        array_grey[i,j] = ( 0.3*image_array[i,j,0] +
            0.5*image_array[i,j,1] + 0.2*image_array[i,j,2] )
plt.imshow(array_grey, cmap = 'Greys_r')
plt.show()
```

Vous pouvez afficher l'image segmentée directement à partir du vecteur précédant (True prend automatiquement la valeur maximale 255 ce qui correspond à la couleur blanche et False prend la valeur minimal 0 ce qui correspond à la couleur noire) :

```
plt.imshow(array_seg, cmap='Greys_r')
plt.show()
```

Exercice

Question : Générer une image d'échiquier en noir et blanc de taille (200,200) avec numpy.

Solution

```
import numpy as np
import matplotlib.pyplot as plt
echiquier_array = np.zeros([200, 200], dtype = np.uint8)
for x in range(200):
    for y in range(200):
        if (x % 50) // 25 == (y % 50) // 25:
            echiquier_array[x, y] = 0
        else:
            echiquier_array[x, y] = 255
plt.imshow(echiquier_array, cmap='Greys_r')
plt.show()
```