L'objectif de ce TP est de manipuler ces images en Python. Basiquement, les images sont des matrices de quadruplet. On commence par quelques manipulations sur les images, et ensuite nous allons voir une très belle application d'un théorème d'algèbre linéaire (décomposition en valeurs singulières) à la compression des images.

1 Introduction

Image

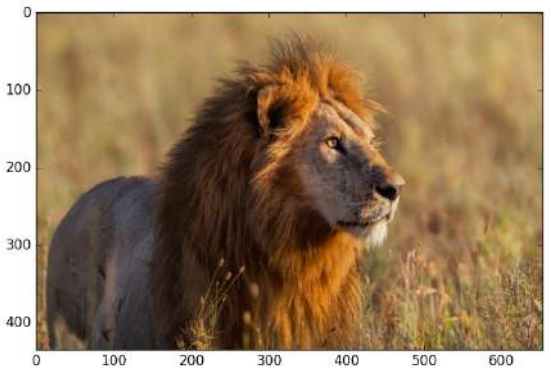
Une image RGBA (Red-Green-Blue-Alpha) est une matrice $N \times M$ dont chaque coefficient est de la forme $[r, g, b, \alpha]$ où :

- r, g et b sont trois nombres compris entre 0 et 1 (niveaux de couleur)
- α est un nombre entre 0 et 1 (transparence)

Un fichier image contient la matrice et d'autres informations. Chaque cellule de la matrice est appelée **un pixel**.

1 Testez le code suivant en expliquant le rôle de chaque ligne :

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
img=mpimg.imread('lion.png')
L=np.shape(img)
print(L)
plt.imshow(img)
plt.show()
```



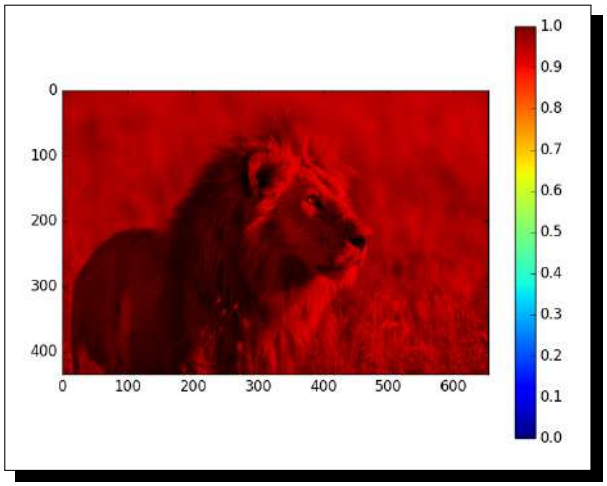
2 Modifier la transparence à 0.5 et afficher l'image.

3 Insérer la commande `img+=img`

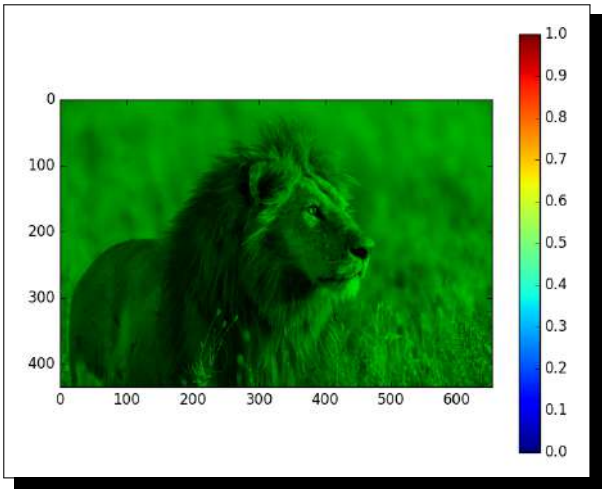
Un commentaire ?

4 Générer les trois couches R/G/B

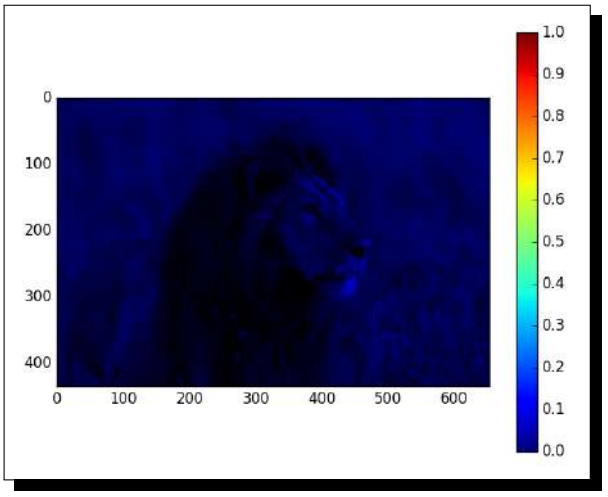
Par exemple le rouge est obtenu en remplaçant chaque pixel (r, g, b) par $(r, 0, 0)$



(a) Rouge



(b) Vert



(c) Bleu

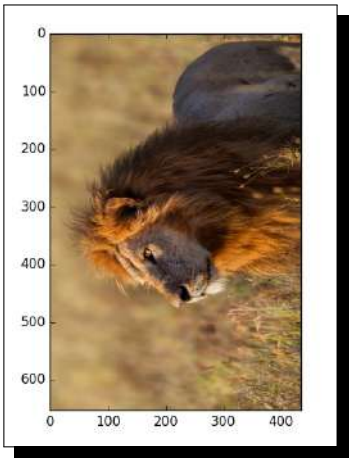
FIGURE 1

Note : la barre de couleur s'obtient avec la commande `plt.colorbar()`

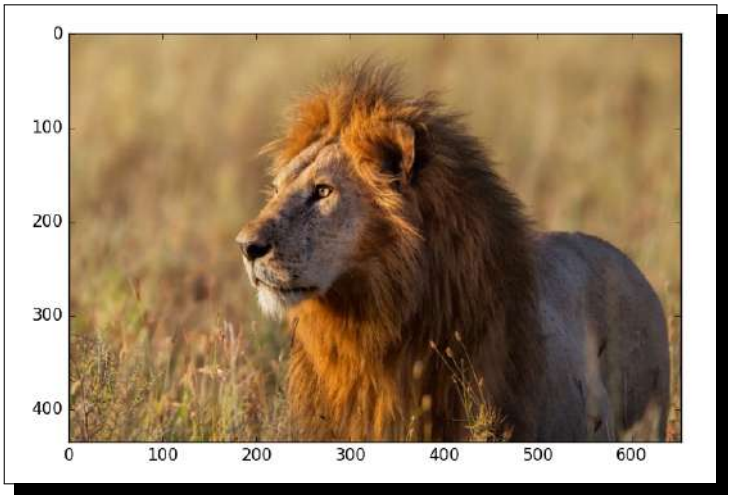
2 Exercices

Exercice 1 (*Opérations*)

Générer les images suivantes :



(a) Transposition



(b) Symétrie 1

(c) Symétrie 2

FIGURE 2 – Opérations sur les images

Exercice 2 (*Négatif*)

Générer le négatif du "lion" obtenu en remplaçant r, g, b par $1-r, 1-g, 1-b$.

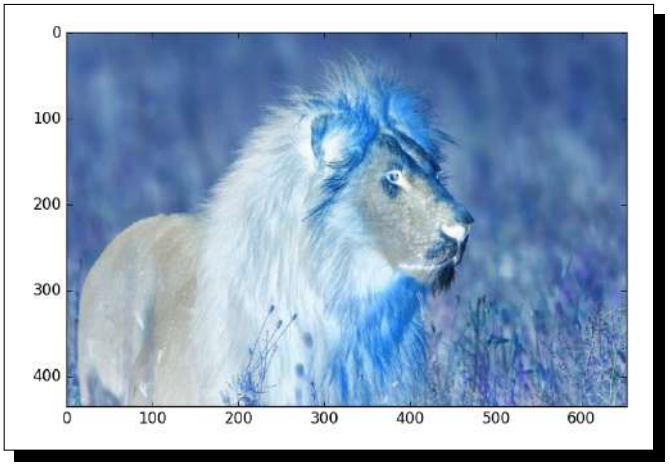


FIGURE 3 – Négatif

Exercice 3 (*Niveau de gris*)

On va créer une image en niveau de gris.

1. Créer un tableau de taille (N, M) `gris` obtenu en remplacant chaque pixel par sa moyenne du R, du G et du B. Comme on n'est plus en RGB, il faut spécifier que l'on souhaite associer à chaque nombre de $[0, 1]$ un niveau de gris. Ceci s'obtient avec la commande `cmap` et `plt.imshow(gris,cmap='gray')`
2. Tester en remplaçant plutôt le pixel par une moyenne coefficientée du type

$$0.9 * R + 0.05 * G + 0.05 * B$$

Exercice 4

Que fait le programme suivant ?

```
gris=np.array([[0.2125 * img[i,j,0] + 0.7154 * img[i,j,1] + 0.0721 * img
    [i,j,2]
 for j in range(M)] for i in range(N)])
plt.imshow(gris,cmap='Greys_r')
```

3 Compression

Exercice 5 (*Fusion*)

La fusion de deux images est obtenues de la manière suivante : Si (r_1, g_1, b_1) est un pixel de la première image et (r_2, g_2, b_2) est un pixel de la deuxième image, l'image fusionnée a pour pixel $(Max(r_1, r_2), Max(g_1, g_2), Max(b_1, b_2))$.

Générer l'image obtenu en fusionnant les images miroirs haut-bas et gauche-droite :

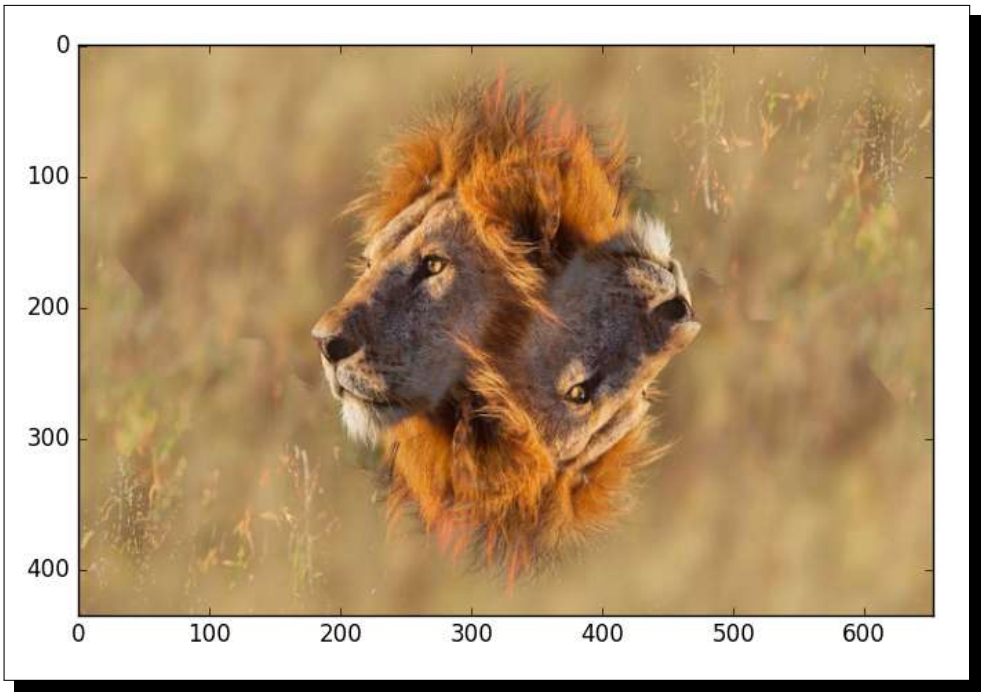


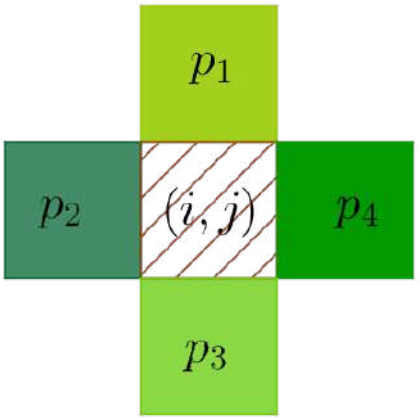
FIGURE 4

Exercice 6 (*Détection de contours*)

A partir de l'image en niveau de gris, on va appliquer un algorithme naïf de détection de contour pour créer une image en noir et blanc. L'idée est la suivante : si un pixel n'est pas si différent de ses voisins, c'est qu'il ne participe pas aux détails de l'image et on peut le remplacer par du blanc (1 en niveau de gris). Sinon on le remplace par du noir (0 en niveau de gris).

Créer une fonction `contour` qui, à partir d'une image en niveau de gris, renvoie une image en noir et blanc créée de la manière suivante :

Si $p_1 \sim p_3$ et $p_2 \sim p_4$ alors `contour(i,j)` vaudra 1 sinon il vaut 0.



La fonction contour prendra un argument seuil, car on traduira la condition $p_1 \sim p_3$ et $p_2 \sim p_4$ par $\sqrt{(p1-p3)^2+(p2-p4)^2} < seuil$

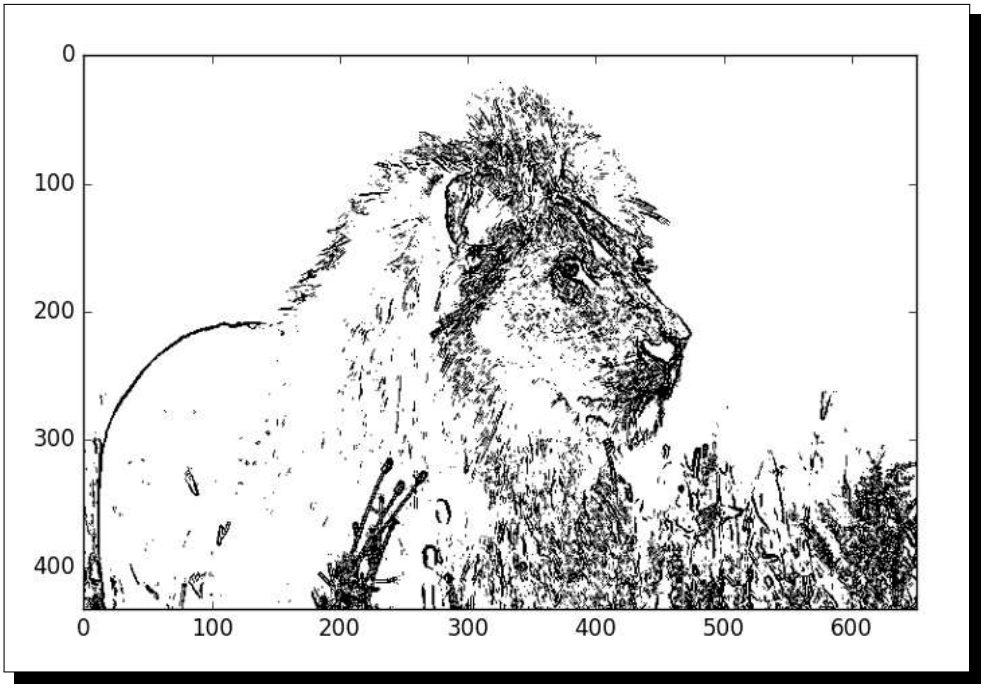


FIGURE 5 – Contour du lion gris au seuil=0.1

4 Un exemple de compression d'image

Théorème : Singular Value Decomposition

Si A est une matrice $N \times M$ réelle, alors il existe U orthogonale $N \times N$, il existe V orthogonale $M \times M$ et S une matrice $N \times M$ dont tous les éléments sont nuls sauf ceux de la "diagonale" qui s'énumèrent en une suite

$$s_1 \geq s_2 \geq s_3 \geq ... \geq s_k \geq 0$$

où $k = min(N, M)$ et tels que

$$A = USV$$

La commande `np.linalg.svd` renvoie le tuple composé des trois matrices U,S,V.
Essayer le code suivant :

```
>>>a = np.random.randn(9, 6)  #matrice aléatoire
>>>U, S, V = np.linalg.svd(a) #récupération de U,S,V
>>>s = np.zeros((9, 6)) #matrice nulle
>>>s[:6, :6] = np.diag(S) #que l'on remplie sur la diagonale
>>> np.dot(U, np.dot(s, V)) #matrice tronquée
```

Les (s_k) deviennent assez rapidement petit, ce qui incite à les négliger à partir d'un certain rang.

On remplace alors USV par UsV où la matrice s est obtenue en remplaçant les s_k par 0 à partir d'un certain rang et on génère l'image obtenue.

Si on note $N+1$ ce rang, voici ce que vous devez obtenir pour l'image compressée en noire et blanc (on notera le poids de chaque image)

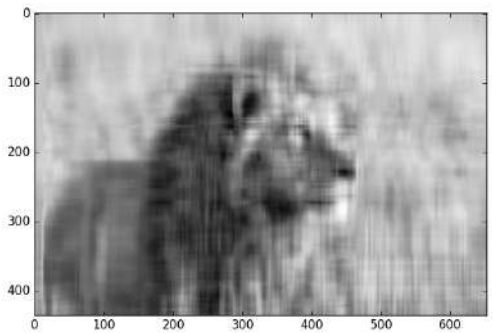


FIGURE 6 – N=10

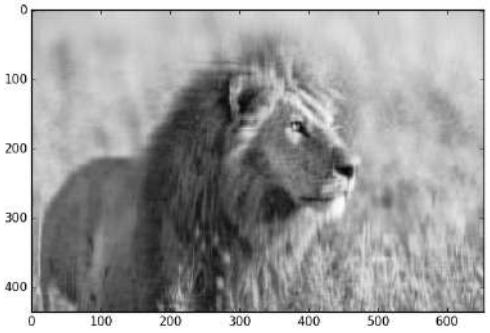


FIGURE 7 – N=30

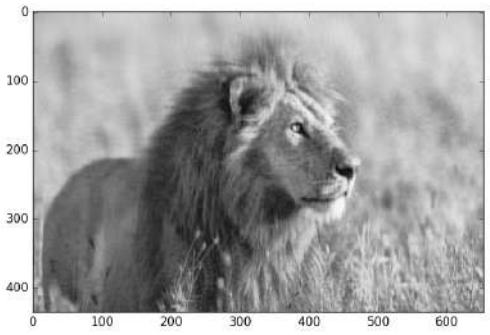


FIGURE 8 – N=50

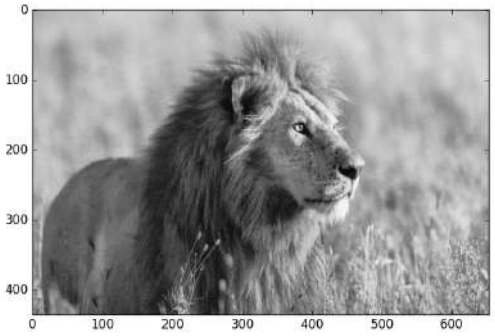


FIGURE 9 – N=100