TP 11 : Traitement d'images avec Numpy

Dans ce TP, on va manipuler des images, à l'aide des tableaux Numpy.

1 Création d'un répertoire de travail

Avant de commencer ce TP, créez un répertoire spécifique dans votre espace personnel, par exemple `TP_Image`. Récupérer l'annexe sur le site web, que vous pouvez placer dans ce répertoire, ainsi que l'image `lena.png`. On va avoir besoin d'indiquer à Python où chercher les images pour les charger, le plus simple est de définir ce répertoire comme répertoire de travail. Veuillez indiquer dans l'annexe le chemin vers votre répertoire, de la forme :

```
os.chdir("U://Documents/chemin/vers/votre/répertoire/TP_Image") #à modifier dans l'annexe.
```

2 Syntaxe pour les images

fonction	description
`im=Image.open('image.png')`	Ouvre l'image `image.png`, stocke le résultat dans la variable `im`.
`im.show()`	affiche l'image stockée dans `im`
`tab=np.array(im)`	calcule le tableau de pixels associé à l'image, le résultat est stocké dans `tab`.
`tab.shape`	la forme du tableau (couple (h,ℓ) pour une image en noir et blanc, triplet $(h,\ell,3)$ pour une image couleur).
`Image.fromarray(tab)`	image associée à un tableau
`np.zeros((h,l), dtype=np.uint8)`	tableau 2D rempli de zéros de type `np.uint8`
`np.zeros((h,l,3), dtype=np.uint8)`	tableau 3D rempli de zéros de type `np.uint8`
`im.save('sauvegarde.png')`	sauvegarde une image.

3 À vous de jouer

Dans ce TP, par convention on écrit principalement des fonctions qui prennent en entrée des images et qui retournent des images. Par exemple la fonction suivante crée une image de la même taille que l'image passée en entrée, mais remplie de pixels noirs (elle est fournie) :

```
from PIL import Image #ces modules sont déja importés en haut de l'annexe
import numpy as np

def image_noire(im):
    t=np.array(im)
    h,l,r=t.shape #on sait que r=3
    for i in range(h):
        for j in range(l):
            for k in range(3):
                t[i][j][k]=0
    return Image.fromarray(t)
```

Remarquez que `0*t` aurait également créé un tableau de même taille rempli de zéros, mais on fera souvent usage de telles boucles imbriquées dans ce TP : i parcourt les indices des lignes, j parcourt les indices des colonnes, k parcourt $\{0,1,2\}$ qui sont les indices des pixels. Il faut que les boucles soient imbriquées pour faire parcourir à (i,j,k) toutes les valeurs $[\![0,h-1]\!]\times[\![0,\ell-1]\!]\times[\![0,2]\!]$. Une utilisation peut-être :

```
lena=Image.open("lena.png")
lena_noire=image_noire(lena)
lena_noire.show() #affichage
```

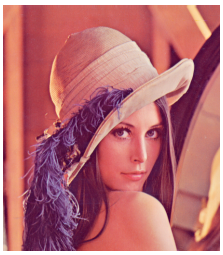
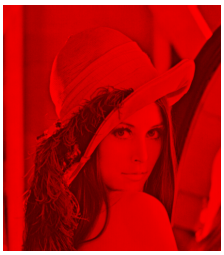
3.1 Décomposition d'une image en ses composantes

Question 1. Ecrire une fonction `composante_rouge(im)` retournant la composante rouge de l'image passée en paramètre. Servez-vous en pour calculer la composante rouge de `lena.png`. En travaillant sur le tableau associé à l'image `im`, il s'agit simplement de mettre à 0 les composantes 1 et 2 des pixels.

```
def composante_rouge(im):
    ...
    A COMPLETER
    ...

lena=Image.open("lena.png")
composante_rouge(lena).show()
```

Le résultat attendu est donné ci-dessous.

 ⇝ 

Faire de même avec les composantes vertes et bleues (on écrira les fonctions associées).

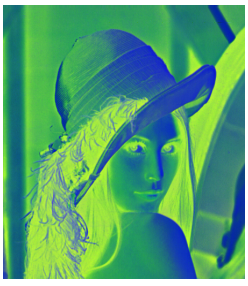
3.2 Négatif d'une image

Une image négative est une image dont les couleurs ont été inversées par rapport à l'originale ; par exemple le rouge devient cyan, le vert devient magenta, le bleu devient jaune et inversement. Les régions sombres deviennent claires, le noir devient blanc. Pour cela il suffit d'inverser les niveaux de chacune des couleurs primaires : un pixel initialement à (120, 10, 250) deviendra (135, 245, 5).

Question 2. Écrire une fonction `negatif(im)` retournant l'image négative de l'image passée en paramètre.

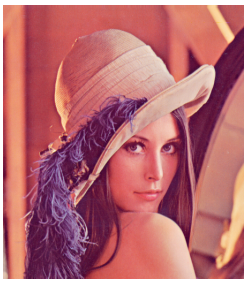
```
def negatif(im):
    ...
    A COMPLETER
    ...
```

Le résultat attendu pour Léna est le suivant :

 ⇝ 

3.3 Mise en place d'un cadre autour d'une image

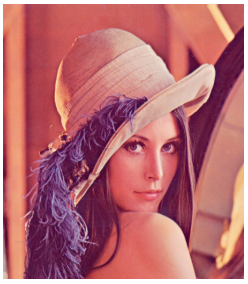
Question 3. Écrire une fonction `cadre_noir(im,ep)` prenant en entrée une image, ainsi qu'un (petit) entier `ep`, renvoyant une nouvelle image, identique à la précédente mais dont les pixels situés sur les bords de l'image ont été remplacés par des pixels noirs, sur une épaisseur `ep`.
Par exemple, le résultat attendu pour Léna et une épaisseur de 5 est donné ci-dessous.




Une fois votre fonction écrite, appliquez-la à Léna avec une épaisseur de 5.

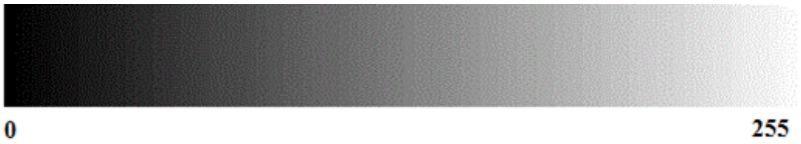
3.4 Rajout d'un cadre autour de l'image

Question 4. Écrire une fonction `rajout_cadre(im,ep)` fonctionnant comme précédemment, mais retournant une image de taille $(h + 2\text{ep}, \ell + 2\text{ep})$ (si l'image initiale a pour taille (h, ℓ)), le cadre ayant été ajouté à l'extérieur plutôt que par modification des pixels du bord. Voici le résultat avec une épaisseur de 20 pixels. Le plus simple est de créer un nouveau tableau à la bonne taille (avec `np.zeros`, dont le comportement a été expliqué plus haut, attention à avoir le type `uint8`), et de le remplir. Rappel : le pixel numéroté $(0, 0)$ correspond au coin en haut à gauche d'une image, et (i, j) au pixel situé à l'intersection de la i-ème ligne et de la j-ème colonne.




3.5 Conversion d'une image couleur en image en niveau de gris

Dans une image en niveaux de gris, les trois composantes R, V, B de chaque pixel ont la même valeur. Celle-ci vaut 0 pour un pixel noir, 255 pour un pixel blanc..

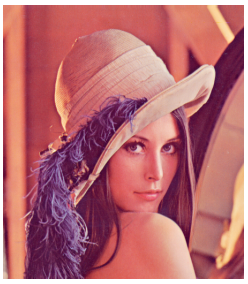


L'œil est plus sensible à certaines couleurs qu'à d'autres. Le vert (pur), par exemple, paraît plus clair que le bleu (pur). Pour tenir compte de cette sensibilité dans la transformation d'une image couleur en une image en niveaux de gris, on ne prend généralement pas la moyenne arithmétique des intensités de couleurs fondamentales, mais une moyenne pondérée. La formule standard donnant le niveau de gris en fonction des trois composantes est :

$$\text{gris} = \lfloor 0.299 \cdot \text{rouge} + 0.587 \cdot \text{vert} + 0.114 \cdot \text{bleu} \rfloor$$

où $\lfloor . \rfloor$ désigne la partie entière.

Question 5. Ecrire une fonction `niveau_gris(im)` renvoyant une nouvelle image, en niveau de gris. Les trois niveaux R, V, B d'un pixel sont égaux au niveau de gris donné par la formule ci-dessus.
Le résultat attendu pour Léna est donné ci-dessous.




Utilisez votre fonction pour obtenir une image que vous sauvegarderez sous le nom `lena_gris.png`

3.6 Image au format « B »

Dans une image en niveau de gris, les niveaux de rouge, vert et bleu étant identiques, les informations sont redondantes. Il existe un format pour stocker des images en noir et blanc (inutile de répéter trois fois la même valeur !). Il est possible de sauvegarder la matrice des pixels non pas en associant à chaque pixel un triplet de niveau (R,V,B) mais une valeur unique égale au niveau de gris. Le « B » du nom de format signifie simplement « black ».

On pourra créer une matrice numpy à 2 dimensions (et non plus 3), représentative d'une image noire à l'aide de l'instruction :

```
tabng=np.zeros((h, l), dtype="uint8") # h est la hauteur de l'image, l la largeur
```

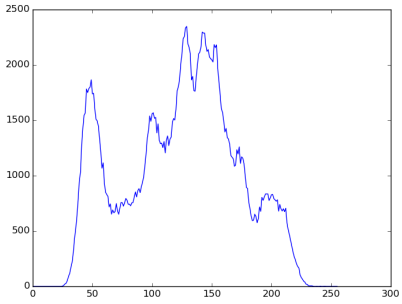
On placera ensuite à chaque emplacement du tableau `tabng[i][j]` le niveau de gris calculé du pixel de la ligne `i` et colonne `j`.

Question 6. Ecrire une fonction `niveau_gris2(im)` faisant la même chose que la fonction `niveau_gris(im)`, mais renvoyant une image au format « B » et non plus « RGB ». L'insruction `Image.fromarray` renvoie une image au bon format à partir d'un tableau de dimension 2.

Utilisez votre fonction pour obtenir une image que vous sauvegarderez sous le nom `lena_gris2.png` (le résultat attendu est visuellement identique au précédent). Cliquez droit sur chacune des deux images en niveau de gris créées et comparez leur taille en ko. Notez que la deuxième image est moins volumineuse que la première !

3.7 Histogramme d'une image

L'histogramme d'une photo permet de compter le nombre de pixel d'un niveau de gris donné. Celui de la photo `lena_gris2.png` est donné ci-dessous.

Question 7. Écrire une fonction `histo(im)`, qui prend en argument une image en niveau de gris au format « B » (décrite par une matrice dont chaque pixel est représenté seulement par le niveau de gris). Cette fonction doit renvoyer une liste de taille 256 : en première position (indice 0), le nombre de pixels noirs (gris 0), en deuxième position (indice 1), le nombre de pixels gris 1, ..., en dernière position (255), le nombre de pixels blancs (gris 255).

Appliquez votre code à l'image `lena_gris2.png`, préalablement chargée. Vous utiliserez ensuite le module `matplotlib` pour tracer l'histogramme. Petit rappel pour tracer un graphe :

```
import matplotlib.pyplot as plt
plt.plot(X,Y) #X et Y sont les listes (ou tableaux numpy) d'abscisses et d'ordonnées des points à tracer
plt.show() #pour afficher
```

Une image qui a un histogramme tout à gauche, c'est-à-dire une image très sombre, est image dite « bouchée ». Une image qui a un histogramme tout à droite, c'est-à-dire une image avec des lumières très vives, est image dite « brûlée ».

3.8 Modifier la luminosité d'une image

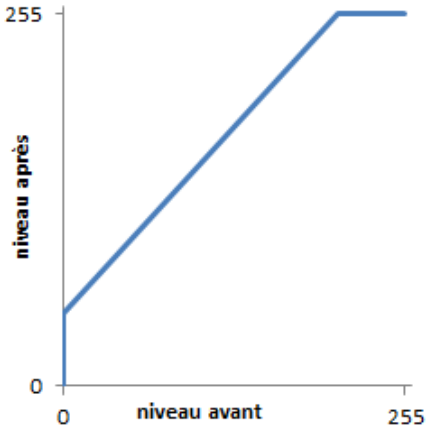
Pour augmenter la luminosité, il suffit d'ajouter une valeur fixe à tous les niveaux. Pour diminuer la luminosité il faudra au contraire soustraire une valeur fixe à tous les niveaux.

Question 8. Écrire une fonction `change_luminosite(im,d)`, qui prend en argument une image en niveau de gris décrite par une matrice dont chaque pixel est représenté seulement par le niveau de gris et un entier entre 0 et 255, valeur du décalage du niveau de gris. Cette fonction renvoie une nouvelle image. Attention : on prendra garde que si l'on essaie de mettre un entier dans un tableau dont les éléments sont de type `uint8`, celui-ci est pris modulo 256. On convient que pour une valeur de pixel p, si $p+d>255$ il faut stocker 255, et si $p+d<0$, il faut stocker 0.

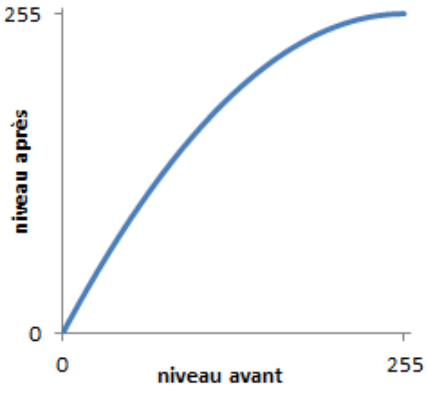
Le résultat attendu pour `lena_gris2.png` avec un décalage de 50 est donné ci-dessous.


$\rightsquigarrow$


Remarque : il est très mauvais d'augmenter ainsi la luminosité : avec un décalage de 50, il n'existera plus aucun point entre 0 et 50, et les points ayant une valeur supérieure à 205 deviendront des points parfaitement blancs, puisque la valeur maximale possible est 255. La nouvelle image contient des zones brûlées. Plutôt que d'utiliser la fonction $p \mapsto \begin{cases} p+50 & \text{si } p<205. \\ 255 & \text{sinon.} \end{cases}$, il vaut mieux utiliser une fonction « presque bijective » de forte croissance au voisinage de 0 et de très faible croissance au voisinage de 255, comme sur le graphe ci-dessous :



Transformation initiale



Une meilleure transformation

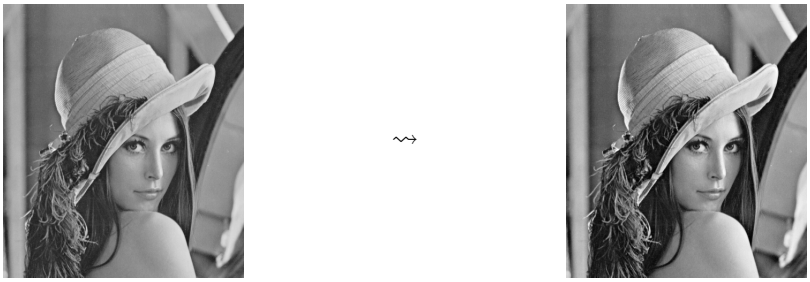
3.9 Augmentation du contraste par dilatation de l'histogramme

Une méthode relativement simple pour augmenter les contrastes est de s'arranger pour que l'histogramme de la nouvelle image occupe toute la plage de valeurs $[\![0,255]\!]$. Pour ce faire, on peut, en considérant l'histogramme comme une fonction $H : [\![0,255]\!] \to \mathbb{N}$:

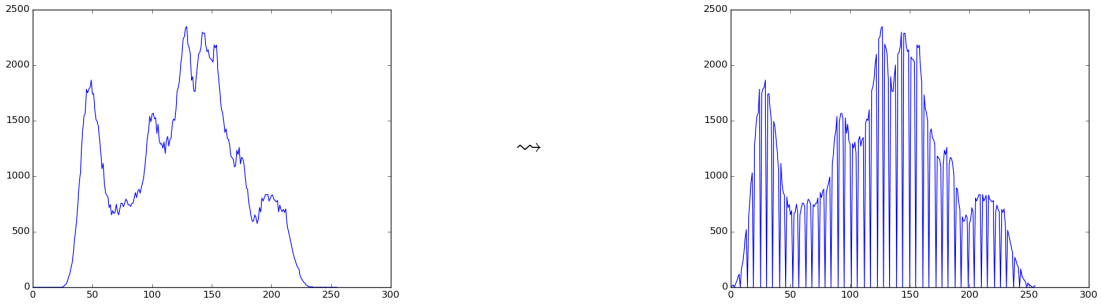
— repérer dans l'histogramme de l'image initiale les indices $i_{\min}$ et $i_{\max}$ tels que pour $i < i_{\min}$ ou $i > i_{\max}$, on ait $H(i) < s$, avec s un petit entier seuil (par exemple $s=3$). Autrement dit, il y a strictement moins de s pixels de couleur gris i pour tous les i strictement inférieurs à $i_{\min}$ ou strictement supérieurs à $i_{\max}$;

— appliquer la transformation affine $x \mapsto \frac{256\times(x-i_{\min})}{i_{\max}-i_{\min}}$ à chaque pixel gris x de l'image, en arrondissant à l'entier le plus proche et en remplaçant les valeurs négatives par 0 et les valeurs supérieures à 256 par 255.

En appliquant cette transformation à l'image `lena_gris2.png`, avec seuil $s = 3$, on obtient le résultat suivant :

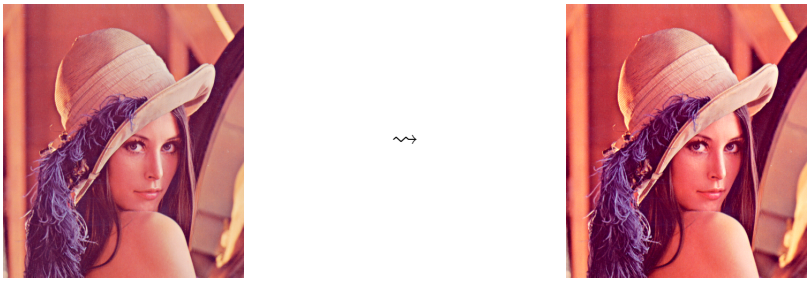


Ceci est très visible sur l'histogramme, voici comment celui-ci a été transformé :



Question 9. Écrire une fonction `augmente_contraste(im,s)` prenant en entrée une image au format « B » (donnée par niveaux de gris) et le seuil s, et renvoyant l'image obtenue par le procédé décrit précédemment. Testez votre fonction avec l'image `lena_gris2.png`.

Remarque : en fait, on peut augmenter également les contrastes sur une image au format RVB, en calculant d'abord l'image en niveau de gris associée, puis son histogramme, et en appliquant ensuite la transformation décrite ci-dessus à l'image au format RVB. Voici le résultat sur l'image de Léna originelle :



3.10 Ensemble de Mandelbrot

Cette section est volontairement moins guidée ! L'ensemble de Mandelbrot est l'ensemble des complexes c du plan tels que la suite $(z_n)_{n\in\mathbb{N}}$ définie par $z_0 = 0$ et $z_{n+1} = z_n^2 + c$ pour tout $n \geq 0$ soit bornée.

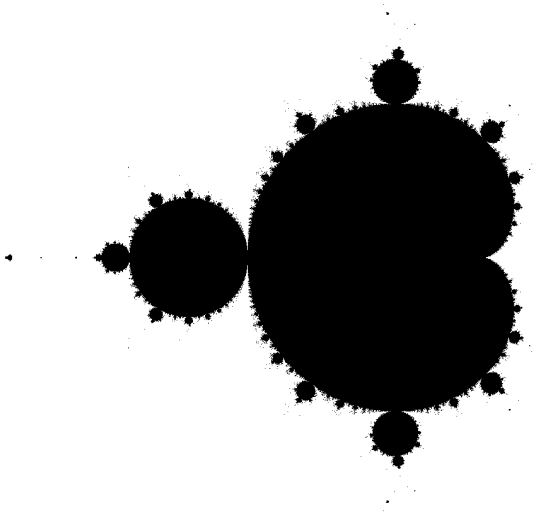
Question 10. Écrire une fonction `mandelbrot(N)` renvoyant une image en noir et blanc de taille $N \times N$, représentant le carré $\{x + \mathrm{i}y \mid -2 \leq x \leq 2 \text{ et } -2 \leq y \leq 2\}$, les points de l'ensemble de Mandelbrot étant noirs, les autres blancs. On admettra que si un terme de la suite a un module supérieur à 2, alors la suite n'est pas bornée.

Indications et rappels :

— Le complexe $x + \mathrm{i}y$ se construit via `complex(i,j)`. Les opérations usuelles (`+`, `*`) fonctionnent sur les complexes, et `abs` donne le module.

— si (i, j) est un pixel de l'image, quel est le complexe qu'on veut lui associer ?

— pour chaque complexe c, on effectuera au plus 100 itérations. Si aucun terme n'a un module qui dépasse 2, on considérera la suite bornée.



Question 11. Bonus : faire varier la luminosité des points en dehors de l'ensemble en fonction de la vitesse avec laquelle on calcule un terme de module supérieur à 2.