TP : Traitement d'images noir et blanc avec Numpy

1 Images en noir et blanc et Numpy

Une image grise au format PNG est simplement donnée par la luminosité de ses pixels, qui est un entier entre 0 et 255 (donc encodable sur 8 bits). Pour une image couleur, il y a trois composantes (rouge, verte et bleue) par pixel, ici nous n'aurons qu'une composante car nous traitons d'images en noir et blanc. Voici comment récupérer les pixels d'une image sous forme d'un tableau Numpy :

```
from PIL import Image
import numpy as np
im=Image.open("lena_gris.png") #à télécharger sur le site web. Adapter le chemin sous Spyder.
T=np.array(im)
h,l=T.shape #hauteur, largeur de l'image
```

Le résultat est un tableau Numpy, dont les éléments sont de type `uint8` (pour *unsigned integer*, entiers non signés, sur 8 bits). `T[i][j]` donne la valeur du pixel à la i-ème ligne, et la j-ème colonne, indexées à partir de 0 (le pixel $(0, 0)$ est le coin en haut à gauche).

À l'inverse, on peut créer une image à partir d'un tableau Numpy au format `uint8` via :

```
Image.fromarray(tableau)
```

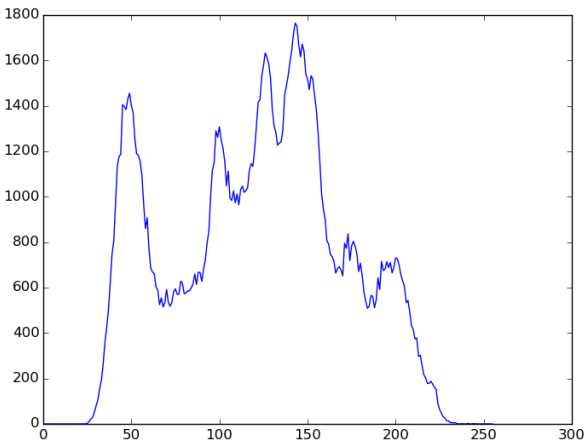
Souvent, le tableau en question est obtenu à partir d'un tableau existant, mais on peut en créer avec `np.zeros` : avec `h` et `l` hauteur et largeur de l'image à créer, on écrirait `np.zeros((h,l), dtype="uint8")` pour créer un tableau idoine, qu'il suffit de remplir.

Pour afficher une image, il suffit d'utiliser `show`. Par exemple `im.show()` via le code situé plus haut vous affiche l'image de Léna.

2 Quelques manipulations élémentaires

2.1 Histogramme

Exercice 1. *Histogramme d'une image*. Écrire une fonction `histo(im)`, qui prend en argument une image en 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_gris.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
```

2.2 Modifier la luminosité d'une image

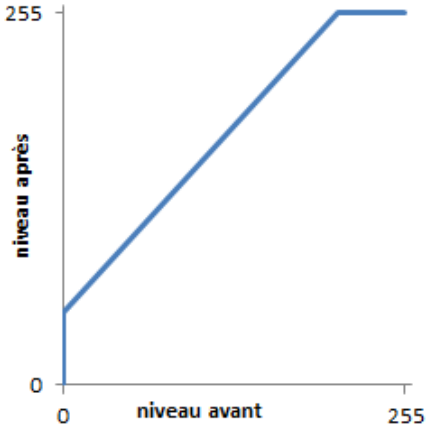
Pour augmenter la luminosité, il suffit d'ajouter une valeur fixe à tous les niveaux. Pour diminuer la luminosité il faudra au contraire soustraire une valeur fixe à tous les niveaux.

Exercice 2. Écrire une fonction `change_luminosite(im,d)`, qui prend en argument une image en 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.

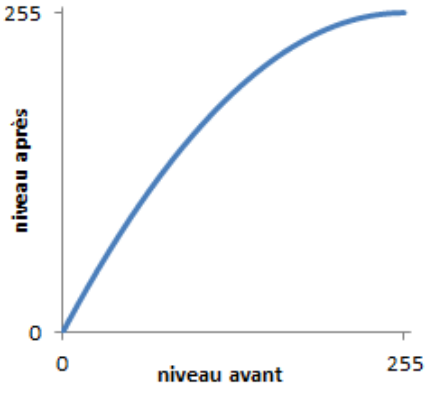
Le résultat attendu pour `lena_gris.png` avec un décalage de 50 est donné ci-dessous.


$\rightsquigarrow$


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

Exercice 3. *Si le TP est terminé.* Chercher une meilleur transformation !

2.3 Augmentation du contraste par dilatation de l'histogramme

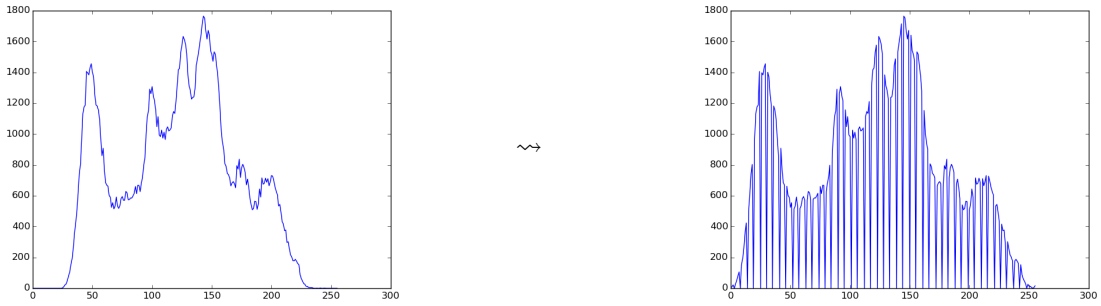
Une méthode relativement simple pour augmenter les contrastes est de s'arranger pour que l'histogramme de la nouvelle image occupe toute la plage de valeurs $[\![0,255]\!]$. Pour ce faire, on peut, en considérant l'histogramme comme une fonction $H : [\![0,255]\!] \to \mathbb{N}$:

— 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 [1] à l'entier le plus proche et en remplaçant les valeurs négatives par 0 et les valeurs supérieures à 255 par 255.

En appliquant cette transformation à l'image `lena_gris.png`, avec seuil $s = 3$, on obtient le résultat suivant [2] :

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



Exercice 4. Écrire une fonction `augmente_contraste(im,s)` prenant en entrée une image en niveau de gris et le seuil s, et renvoyant l'image obtenue par le procédé décrit précédemment. Testez votre fonction avec l'image `lena_gris.png`.

3 Images en fichier texte

Les formats d'image sont variés, dans cette section, on va s'intéresser à deux formats textuels : PBM et PGM. On pourra ouvrir le contenu avec le bloc note, par exemple.

3.1 Format PBM

Le format PBM est un format textuel (ASCII) permettant d'encoder des images binaires (chaque pixel est soit noir, soit blanc). Voici un exemple de fichier au format PBM, encodant une image représentant la lettre « J » [3]

```
P1
# Un exemple bitmap de la lettre "J"
7 10
0 0 0 0 0 0 0
0 0 0 0 0 1 0
0 0 0 0 0 1 0
0 0 0 0 0 1 0
0 0 0 0 0 1 0
0 0 0 0 0 1 0
0 0 0 0 0 1 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
```

Quelques précisions :

— La première ligne contient simplement `P1`.

— Tout ce qui suit un `#` est ignoré : ceci permet les commentaires.

1. la fonction `round` est là pour ça.
2. Mieux que l'originale !
3. L'exemple a été honteusement pris sur Wikipedia.

— La deuxième ligne (sans compter les lignes de commentaires) contient largeur et hauteur de l'image.

— À la suite se trouvent les valeurs des pixels, prise de haut en bas puis de gauche à droite, séparés par des espaces et des sauts de lignes. Les 1 correspondent à des pixels noirs, les 0 à des pixels blancs. Rien n'impose d'écrire une ligne de l'image par ligne du fichier, d'ailleurs les lignes sont normalement limitées à 70 caractères (caractères d'espacement inclus). Par exemple :

```
P1
3 5
1    0 1 0 1 0
0 0  1    1    1 1 1 0
0
```

est un encodage tout à fait valide d'une image de hauteur 5 et de largeur 3.

Rappel. Pour ouvrir un fichier en écriture, on utilise `f=open("nom_du_fichier", "w")` (le `w` signifie « write »). Pour écrire dans le fichier, on utilisera dans ce TP `f.write(s)` où `s` est une variable contenant une chaîne de caractères. Pour transformer un entier (ou autre chose) en chaîne, utiliser `str`. Pour concaténer des chaînes, utiliser `+`. Enfin, le saut de ligne s'encode comme `"\n"`. Une fois votre fichier écrit, fermez-le[4] avec `f.close()`.

Exercice 5. *Conversion PNG vers PBM*. Écrire une fonction `conversion_PNG_PBM(im, f)` prenant en entrée une image au format PNG (dont les pixels sont blancs (255) ou noirs (0), et écrivant dans un fichier `f` ouvert en écriture l'image au format PBM. En pratique, il faut ici ouvrir en écriture un fichier du style `image.pbm` (en dehors de la fonction), dans lequel on va écrire des lignes de texte comme ci-dessus. Essayez avec l'image `QR_code.png`. Modifier ensuite la fonction pour transformer une image en niveau de gris en image PBM : un pixel de valeur inférieure à 120 sera converti en pixel noir, autrement il sera converti en pixel blanc.

Remarque : il n'est pas très évident de faire l'inverse (nombre arbitraire d'espaces, commentaires...), mais ce serait un bon exercice !

3.2 Format PGM

Vous pouvez passer cette section dans un premier temps. Le format PGM est très similaire au format PBM, mais il encode des images en niveau de gris. Voici le texte d'un fichier PGM[5] :

```
P2
# Affiche le mot "FEEP" (exemple de la page principale de Netpbm à propos de PGM)
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

4. C'est essentiel : le fichier n'est *réellement* écrit qu'à la fermeture.
5. Qui provient, là encore, de Wikipedia.

— le fichier commence par `P2` ;
— après le couple largeur, hauteur se trouve une borne sur la luminosité.
— Les lignes suivantes sont les pixels, suivant les mêmes règles qu'un fichier au format PBM.

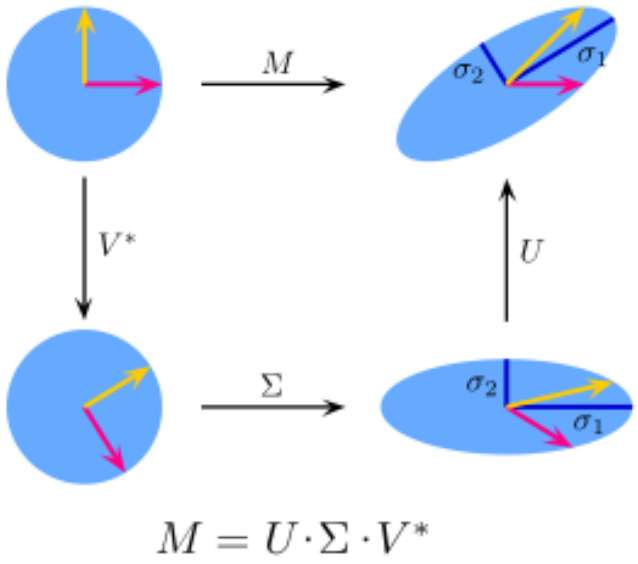
Exercice 6. *Conversion PNG vers PGM.* Écrire une fonction `conversion_PNG_PGM(im, f)` prenant en entrée une image au format PNG, et écrivant dans un fichier `f` ouvert en écriture l'image au format PGM. Tester avec Lena.

4 Compression d'une image par utilisation de la décomposition SVD

Une matrice M, de taille $n \times m$, à coefficients dans $\mathbb{R}$ admet une décomposition SVD (pour *singular values decomposition*) de la forme $M = U \times \Sigma \times {}^tV$ où :

— La matrice orthogonale V contient un ensemble de vecteurs de base orthonormés de $\mathbb{R}^m$, dits « d'entrée » ;
— La matrice orthogonale U contient un ensemble de vecteurs de base orthonormés de $\mathbb{R}^n$, dits « de sortie » ;
— La matrice $\Sigma = (s_{i,j})_{0 \leq i \leq n-1, 0 \leq j \leq m-1}$ est une matrice de taille $n \times m$ « diagonale » (seuls les éléments $s_{i,i}$ sont non nuls), à coefficients positifs, dont les éléments diagonaux sont les valeurs singulières de la matrice M. Les valeurs singulières sont décroissantes : $s_{0,0} \geq s_{1,1} \geq \cdots \geq s_{r-1,r-1} \geq 0$ avec $r = \min(n, m)$.

Cette décomposition généralise la diagonalisation des matrices symétriques réelles positives en base orthonormée, que vous connaissez (dans ce cas, $U = V$ et les valeurs singulières sont les valeurs propres), d'ailleurs l'existence de la décomposition peut se montrer via le théorème spectral appliqué à ${}^tMM$. L'interprétation géométrique de la décomposition se visualise dans le schéma ci-dessous [6]



Un moyen simple [7] de compresser une image est d'utiliser la décomposition en valeurs singulières : les grandes valeurs singulières (les premières dans la matrice Σ) sont les plus pertinentes. Si on en garde seulement k (virtuellement, on met à zéro les coefficients $s_{k,k}, \ldots, s_{r-1,r-1}$), on peut se contenter de ne stocker que les k premières lignes de ${}^tV$ (donc les k premières colonnes de V) et les k premières colonnes de U. À cela on rajoute les k valeurs singulières. Ceci fournit donc un moyen de compresser une image.

On peut obtenir une décomposition SVD en Python via la fonction `svd` du sous-module `numpy.linalg`, qu'on importera comme suit :

```
import numpy.linalg as alg
```

Exercice 7. Après avoir lu l'aide de la fonction `svd` (via `help(alg.svd)`), écrire une fonction `compression(M,k)` prenant en entrée une matrice `M` et un entier k, et retournant la matrice $U\Sigma_k{}^tV$ où Σ_k est la matrice Σ où les $s_{k,k}, s_{k+1,k+1}, \cdots, s_{r-1,r-1}$ ont été mis à zéro. Afficher l'image associée pour Lena, pour $k = 30$.

6. Tiré de Wikipédia !
7. Il en existe de meilleurs.

Exercice 8. En pratique, on stockerait uniquement les coefficients « utiles » : quel taux de compression obtient-on en fonction de n, m et k ? Application numérique : rajouter l'affichage à l'écran du taux de compression dans la fonction précédente. Faire une boucle pour $k \in \{10, 20, 30, ..., 150\}$, en affichant l'image obtenue.

FIGURE 1: Compression via SVD, pour $k \in \{10, 20, 30, 40, 50, 60\}$. $k = 395$ correspond à l'image sans perte, mais déja avec $k = 60$ on est très proche de l'image originelle.

Exercice 9. On peut appliquer une compression via SVD à une image au format RGB (rouge, vert, bleu) : il suffit d'appliquer l'approche au trois composantes de l'image. Le faire (chercher `lena.png` sur mon site).

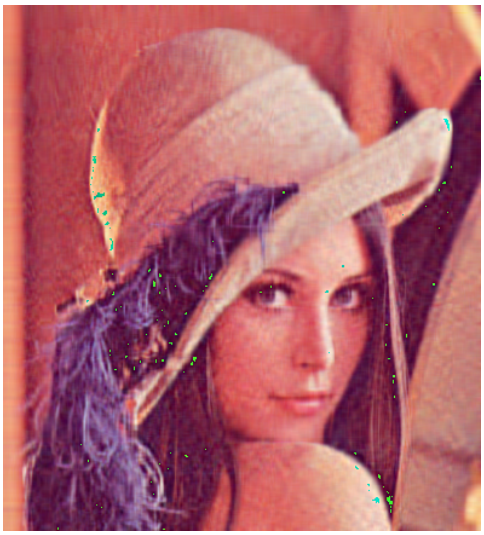


FIGURE 2: Compression de Léna avec $k = 40$. Certains pixels sont verts, et je ne sais pas pourquoi !