iNFO

IUT GRAND OUEST NORMANDIE

R 5.A.06

2023 – 2024

Sensibilisation à la programmation multimédia

CM n° 2
Traitements Images

ANNE Jean-François

Sensibilisation à la programmation multimédia
Programmation des images

Le but de ce TP est de se familiariser avec la programmation d'images en python.

A. Définitions

a) Notion de pixel (et quantification)

Un pixel (de l'anglais *Picture element*) est un point coloré élémentaire, petit constituant de base pour former les images numériques sur ordinateur :

- Quand chaque pixel est choisi soit noir soit blanc, cela donne une image monochrome ;
- Il peut aussi être choisi parmi les intermédiaires entre ces extrêmes, cela donne une image en niveaux de gris ;
- Enfin, il peut être choisi parmi les couleurs visibles par notre œil, cela donne une image en couleurs.

Ces valeurs possibles attribuées à un pixel représentent le résultat de la quantification du pixel (en traitement du signal).

Trois cas de quantification pour les pixels de l'image Lena : sur 2 valeurs, sur plusieurs niveaux de gris, en couleur

b) Matrice image (et échantillonnage)

L'image numérique est alors la subdivision de l'image d'origine en une mosaïque de pixels : on parle d'échantillonnage qui discrétise l'image continue d'origine en pixels élémentaires.

Si on agrandit l'image numérique, on peut percevoir les pixels : ce sont les petits rectangles de couleur apparaissant sur la figure suivante (dans le cas habituel où les pixels ont été choisis de cette forme rectangulaire).

Image matricielle (et agrandissement d'une partie) [source : Wikipédia]

Plus cet échantillonnage est dense (c.-à-d. le nombre de pixels induit par le nombre de lignes et de colonnes), plus la résolution de l'image est dite *haute* et moins cet effet de discrétisation en pixels est visible, mais rend l'image évidemment plus volumineuse en taille à enregistrer ou à transmettre sur le réseau.

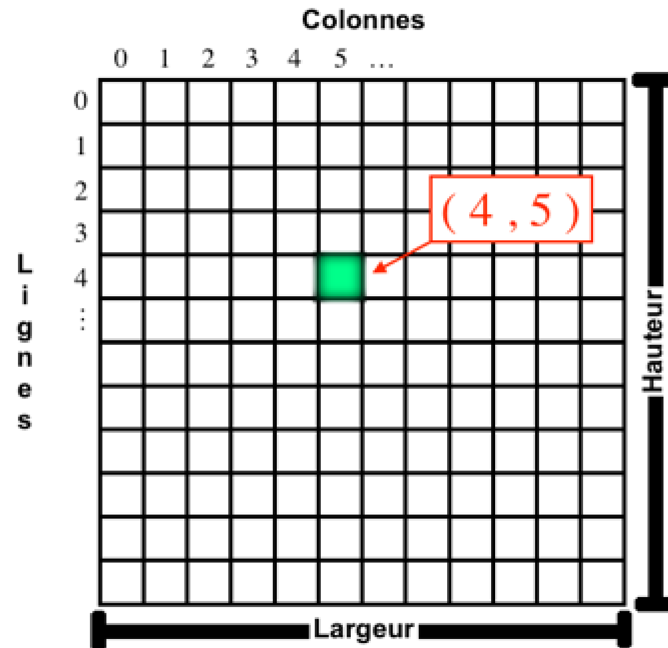
c) ***Coordonnées d'un pixel dans la matrice***

Pour désigner la position d'un pixel dans cette matrice, par exemple en parlant de pixel de coordonnées (4, 5), on s'inspire des mêmes conventions que pour les matrices mathématiques :

- La première coordonnée est un indice de ligne ;
- La deuxième coordonnée est un indice de colonne.

Ce pixel est donc celui situé sur la ligne 4, et la colonne 5.

N.B. : Sans doute par héritage d'habitudes informatiques, les matrices d'images démarrent cependant leurs indices à 0 au lieu de 1, comme usuellement en mathématiques. La première ligne est donc d'indice 0, la deuxième ligne est d'indice 1, etc.

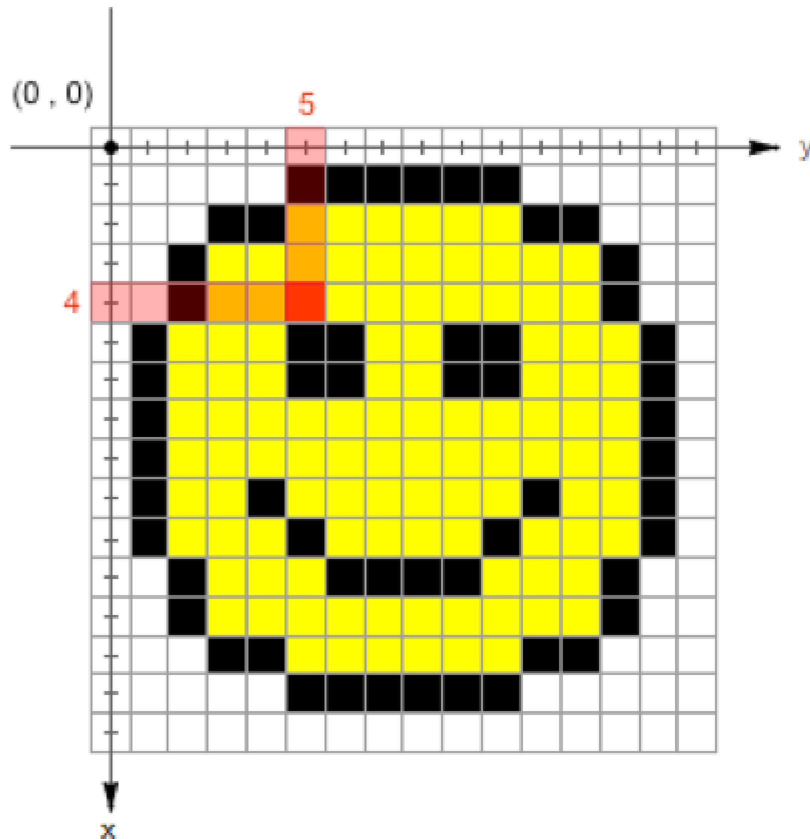


Système de coordonnées d'un pixel, en ligne/colonne

Parfois, il est tentant d'utiliser les notions d'abscisse et d'ordonnée, par analogie avec les coordonnées des points du plan en mathématiques (et donc d'utiliser les notations x et y). Mais il faut alors redoubler de vigilance, car cet usage entraîne quelques difficultés dans le cas des images :

Assez souvent, on trouve x utilisé comme coordonnée horizontale et y comme coordonnée verticale (ce qui est usuel) tout en conservant l'habitude informatique de l'origine (0,0) en haut à gauche de l'image. Bien que pratique, il ne faut pas perdre de vue que ceci permute les coordonnées et qu'ainsi la position (4, 5), où $x = 4$ et $y = 5$, n'est plus au même endroit mais en ligne 5 et colonne 4. De plus, un tel repère xy n'est plus un repère direct en mathématiques, ce qui entraîne des conséquences peu pratiques pour certains calculs.

Une autre convention consiste alors à utiliser x en vertical et y en horizontal. Bien que surprenant à première vue, cet usage évite les deux écueils de la convention précédente : le point de coordonnées (4, 5) reste bien celui de ligne 4 colonne 5, et le repère xy reste un repère direct.



Système de coordonnées d'un pixel en abscisse et ordonnée (x,y), conservant l'ordre ligne/colonne et un repère direct

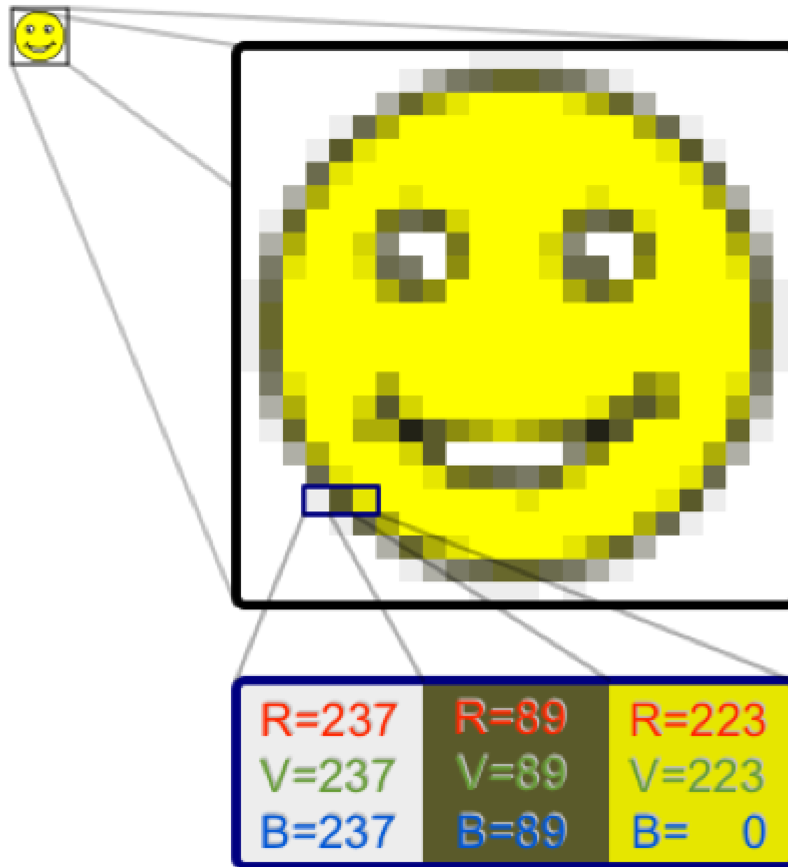
d) Représentation des couleurs

Comme vu précédemment, une valeur est attribuée pour chaque pixel afin de représenter sa « couleur ».

Le cas des **niveaux de gris** est le plus direct : classiquement, la valeur est mémorisée sur un octet, ce qui autorise des valeurs de 0 à 255. Cela permet de représenter l'intensité lumineuse sur 256 niveaux de gris, 0 étant le noir, 255 le blanc, et toute valeur entre les deux représentant un gris intermédiaire. Dans ce cas, l'image est une matrice d'octets.

Le cas **monochrome** pourrait donc être mémorisé avec des 0 et des 255, mais par économie on préfère ne pas représenter chacune de ses deux uniques couleurs par un octet (0 ou 255), mais par un seul bit (0 ou 1). Dans ce cas, l'image est une matrice binaire.

Le cas des pixels **couleurs** est moins direct : la représentation usuelle des couleurs y est directement liée au système visuel humain. On mémorise en effet la quantité de rouge, de vert et de bleu (comme le percevraient les trois différents types de cônes de la rétine de l'œil humain). Usuellement, comme pour les niveaux de gris, ces quantités respectives sont mémorisées par 3 octets, représentant chacun respectivement le niveau de rouge, le niveau de vert et le niveau de bleu d'un pixel. Dans ce cas, l'image est une matrice de triplets RVB.



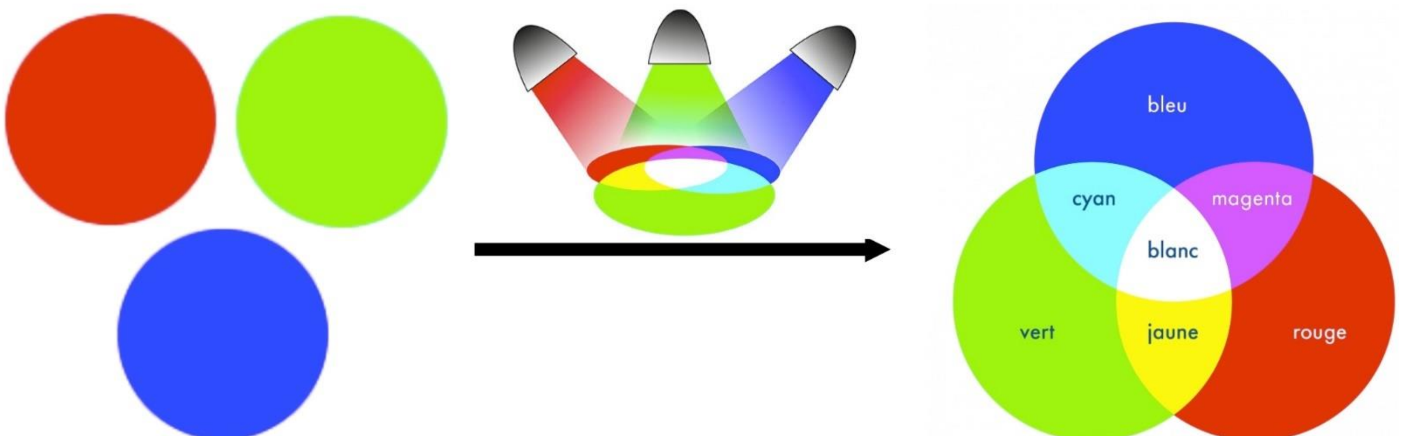
Triplets RVB constituant chaque pixel d'une image couleur.

On observe au passage qu'un niveau de gris est une proportion en quantités égales de rouge, vert et bleu

Il existe un cas un peu intermédiaire entre les images en niveaux de gris (un numéro de niveau par pixel) et les images en couleurs (un triplet RVB en chaque pixel) : les images avec palettes de couleurs (*colormap* en anglais). Le principe consiste à conserver un simple nombre pour quantifier chaque pixel (par exemple, encore un octet de 0 à 255), et cette valeur sera le numéro d'une couleur définie dans une palette de couleurs : par exemple 0=noir, 1=bleu, 2=vert clair, etc. Chacune de ces couleurs étant définie en RVB dans la palette. Lorsqu'une image couleur contient peu de teintes différentes, cela allège l'espace mémoire et de stockage nécessaire et les temps de traitements.

i.Synthèse additive des couleurs

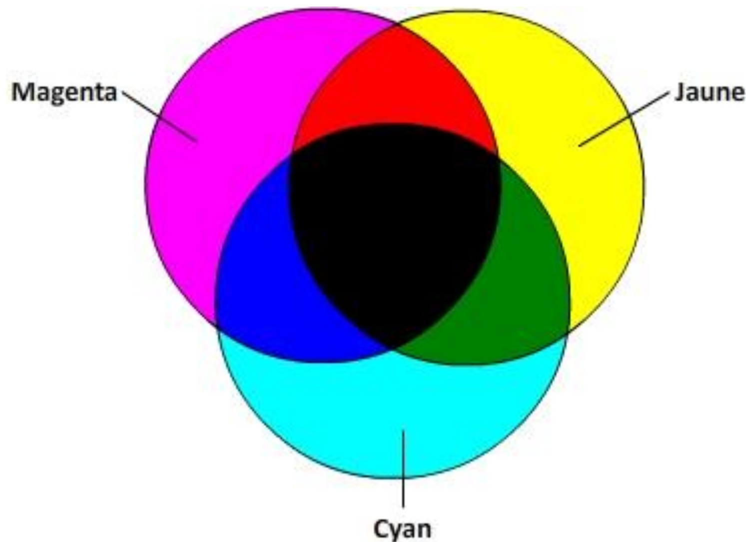
On observe sur l'image précédente qu'un pixel jaune est obtenu par « mélange » de rouge et de vert en quantités égales. Ce procédé, appelé synthèse additive, cela consiste à mélanger des proportions de lumière rouge, verte et bleue (qui servent de couleurs primaires) pour obtenir une lumière colorée (et ainsi reproduire les différentes couleurs), qui est par exemple émise par l'écran de notre ordinateur à destination de notre œil. Notons qu'on aboutit ainsi à du blanc intense quand on mélange les 3 lumières de base en quantité maximale.



Synthèse additive : des lumières de base (couleurs primaires) s'ajoutent pour former les différentes couleurs

ii.Synthèse soustractive des couleurs

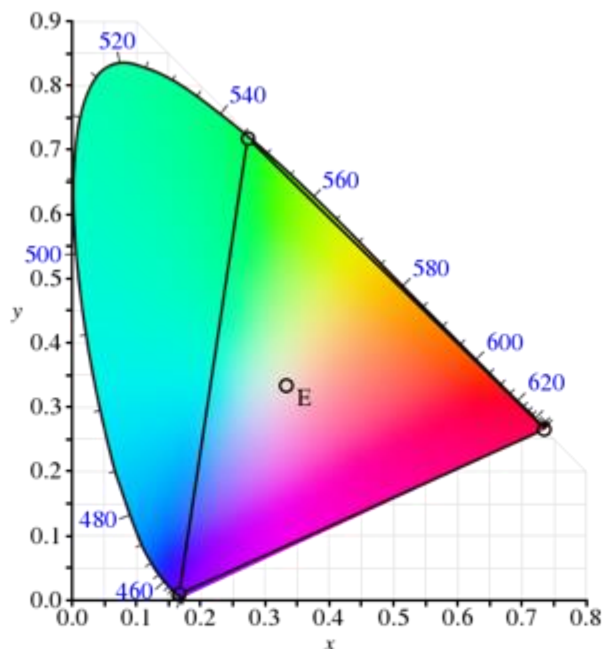
En synthèse soustractive, on part du blanc et on enlève, (on filtre) une ou plusieurs couleurs. La synthèse soustractive est utilisée dans l'imprimerie, les couleurs primaires sont le **cyan**, le **magenta**, le **jaune** (et le **noir** que l'on est obligé de rajouter car sa formation à partir des trois autres couleurs est, pour des raisons d'impureté, impossible).



iii.Autres systèmes de représentation des couleurs

En fait, d'autres systèmes de représentation des couleurs du spectre visible ont été formalisés, car toutes ne sont pas représentables par cette décomposition RVB.

C'est ainsi au sein des couleurs représentables que s'introduit la notion de gamut : c'est la partie de l'ensemble des couleurs visibles qu'un appareil ou un système est capable de représenter.



Exemple de gamut : les couleurs représentables en RVB (dans le triangle) par rapport à l'ensemble des couleurs visibles [source : Wikipédia]

e) Principes des fichiers image

Lors de l'enregistrement d'une image numérique (matrice de pixels) dans un fichier informatique, différents formats ont été standardisés, chacun avec des spécificités dont il est utile de connaître les principes généraux.

i. Les formats sans compression

Le réflexe le plus simple consiste à enregistrer un par un chacun des pixels de l'image dans le fichier informatique. Ainsi, le fichier résultat fait une taille similaire à la matrice de pixels : par exemple une matrice de pixels en niveaux de gris, de n lignes et n colonnes, représentait $n2$ octets dans l'ordinateur et occupera aussi $n2$ octets dans le fichier informatique.

• **Formats courants** : BMP, mais aussi des PNM (PPM, PGM, PBM) et variantes.

ii. Les formats de compression sans pertes

Une première idée pour économiser de l'espace lors du stockage est d'essayer d'exploiter les zones d'image où les pixels voisins sont identiques. Ainsi, au lieu de stocker n fois successivement la même couleur, on peut enregistrer la couleur et n (soit 2 valeurs seulement). C'est ce qu'on appelle le codage par plages (*run-length encoding*, en anglais). On devine que ce principe simple n'est pas gagnant quand tous les pixels voisins sont différents, mais en moyenne il peut s'avérer intéressant. Il existe d'autres principes comme le codage entropique (qui vise à représenter les valeurs d'origine par un code à longueur variable, d'autant plus court que la valeur est fréquente, et long quand la valeur est rare) ou aussi le codage par dictionnaire (qui remplace des combinaisons de valeurs d'origine par une marque courte, qui sera mémorisée dans un dictionnaire, servant au remplacement inverse pour reconstituer la matrice de pixels).

• **Formats courants** : GIF, TIFF, PNG…

iii. Les formats de compression avec pertes

Enfin, on peut pousser plus loin l'économie de place en permettant que quelques pixels soient modifiés durant l'enregistrement, du moment que cela ne se remarque pas facilement à l'œil (il y a donc perte d'information, car alors l'image relue après enregistrement ne fournit pas rigoureusement la même matrice de pixels, mais une matrice approchante). Pour ce faire, le principe ne consiste généralement pas directement à uniformiser des pixels voisins, mais plutôt à appliquer d'abord une transformée fréquentielle sur l'image (cf. chapitres suivants), puis opérer les transformations avec pertes dans cet espace (par exemple, ne pas enregistrer les hautes fréquences qui sont de petits détails presque imperceptibles).

• **Formats courants** : JPG, JPEG, JPEG 2000…

iv. Les formats d'image en vectoriel

Les images vectorielles sont basées sur des objets géométriques (lignes, courbes, formes) et des informations mathématiques décrivant leur position, leur taille et leurs propriétés. Au lieu d'enregistrer chaque pixel, elles enregistrent des instructions pour dessiner les objets. Par conséquent, elles sont indépendantes de la résolution, ce qui signifie qu'elles peuvent être redimensionnées à n'importe quelle taille sans perte de qualité. Les lignes et les formes restent nettes et précises.

Les images vectorielles sont idéales pour les logos, les icônes, les illustrations, les schémas, les cartes, etc. Elles sont particulièrement utiles lorsque vous avez besoin de redimensionner fréquemment une image sans perdre en qualité.

• **Formats courants** : SVG (Scalable Vector Graphics), AI (Adobe Illustrator), EPS (Encapsulated PostScript), PDF (Portable Document Format), etc.

2°) Synthèse :

1. **JPEG (Joint Photographic Experts Group)** :

- Avantages :
  - Compression avec perte : permet de réduire la taille du fichier.
  - Prise en charge de millions de couleurs.
  - Convient aux photographies et aux images avec des dégradés de couleurs.
- Inconvénients :
  - Perte de qualité lors de la compression.
  - Pas adapté aux images avec des zones de couleur unie ou des contours nets.
  - Ne prend pas en charge la transparence.

2. **PNG (Portable Network Graphics)** :
   - Avantages :
     - Compression sans perte : conserve la qualité de l'image.
     - Prend en charge la transparence (canal alpha).
     - Convient aux images avec des zones de couleur unie et des contours nets.
   - Inconvénients :
     - Fichiers plus volumineux que le JPEG.
     - Peut ne pas être le meilleur choix pour les photographies en raison de la taille des fichiers.
3. **GIF (Graphics Interchange Format)** :
   - Avantages :
     - Prend en charge l'animation.
     - Compression sans perte pour les images avec moins de couleurs.
   - Inconvénients :
     - Limite de 256 couleurs.
     - Ne convient pas aux photographies de haute qualité.
     - Taille de fichier généralement plus grande que le PNG pour des images similaires.
4. **BMP (Bitmap)** :
   - Avantages :
     - Aucune compression : qualité maximale.
     - Prise en charge de millions de couleurs.
   - Inconvénients :
     - Taille de fichier très grande.
     - Pas adapté au web en raison de sa taille.
5. **TIFF (Tagged Image File Format)** :
   - Avantages :
     - Compression avec ou sans perte (selon la configuration).
     - Prise en charge de plusieurs canaux (RVB, CMJN, etc.).
     - Haute qualité pour l'impression.
   - Inconvénients :
     - Taille de fichier généralement plus grande que le JPEG ou le PNG.
     - Pas aussi largement pris en charge par les navigateurs web.
6. **WebP** :
   - Avantages :
     - Compression avec ou sans perte.

        - Prend en charge la transparence.
        - Fichiers plus petits que le JPEG ou le PNG tout en maintenant une bonne qualité.
    - Inconvénients :
        - Pas encore pris en charge par tous les navigateurs.

7. **SVG (Scalable Vector Graphics)** :
    - Avantages :
        - Images vectorielles, redimensionnables sans perte de qualité.
        - Convient aux logos, icônes et graphiques.
    - Inconvénients :
        - Ne convient pas aux images complexes ou aux photographies.

Le choix du format d'image dépendra de l'utilisation prévue, de la qualité requise, de la taille du fichier et de la prise en charge par les navigateurs ou les logiciels. Il est courant d'utiliser une combinaison de formats en fonction des besoins spécifiques.

3°) Apprenez à représenter et afficher des images en langage Python

Pour la mise en pratique, nous utiliserons le langage Python qui permet une écriture aisée de la gestion d'images numériques, en s'appuyant sur les opérations très utiles des bibliothèques complémentaires comme OpenCV, Numpy et Matplotlib.

- https://koor.fr/Python/Tutoriel_Scipy_Stack/matplotlib_image.wp

a) Chargez une matrice de pixels

Nous allons faire nos premiers essais avec la photo de Lena (renommée Lenna par les Américains), qui est un classique historique des chercheurs en traitement d'images (téléchargeable sur la page https://en.wikipedia.org/wiki/Lenna et à enregistrer sous le nom `Lena.png`).

Tapez le programme suivant, et exécutez-le dans votre environnement Python.

```
from PIL import Image

# charge l'image
image = Image.open("Lena.png")
pixels = image.load()

# obtient les dimensions de l'image
largeur, hauteur = image.size
print(largeur, hauteur)

# accède à la valeur du premier pixel
pixel = pixels[0, 0]
print(pixel)
```

Il doit s'afficher :

```
512 512

(226, 137, 125)
```

En effet, l'image chargée par la fonction `image.load()` fait 512 lignes sur 512 colonnes, avec chaque pixel stocké sur 3 octets (car c'est une image couleur, en triplets RVB). Ensuite s'affiche la valeur du pixel de la ligne 0 colonne 0 qui est le triplet [226, 137, 125].

Pour convertir l'image couleur précédente en matrice de pixels en niveaux de gris, on peut ajouter les instructions suivantes :

```
from PIL import Image

# charge l'image
image = Image.open(Lena.png")

# Convertit l'image en niveaux de gris
image_grayscale = image.convert("L")

# Affiche l'image en niveaux de gris
image_grayscale.show()
pixels = image_grayscale.load()

# obtient les dimensions de l'image
largeur, hauteur = image_grayscale.size
print(largeur, hauteur)

# accède à la valeur du premier pixel
pixel = pixels[0, 0]
print(pixel)
```

Cette fois, il s'affiche :

```
(512, 512)

162
```

La nouvelle matrice fait aussi 512 lignes par 512 colonnes, mais sur de simples octets. Et la couleur RVB du pixel a été convertie en 162, qui correspond à la luminance.

b) Enregistrez une matrice de pixels

Cet appel de fonction `save` enregistre un fichier contenant la matrice de pixels, selon le type de matrice (niveaux de gris, ou couleurs) et selon le format indiqué par le nom choisi (ici PNG, mais on pourrait aussi choisir JPEG en terminant le nom du fichier par .jpg).

```
# Enregistre l'image modifiée au format JPG
image.save("chemin_vers_l_image_modifiee.jpg")

# Enregistre l'image modifiée au format PNG
image.save("chemin_vers_l_image_modifiee.png")
```

c) Visualisez une matrice de pixels

Il serait possible de visualiser l'image enregistrée grâce à un logiciel de traitement d'images (par exemple GIMP qui est un logiciel gratuit de retouche d'images, très réputé, appartenant à la famille des logiciels libres).

Il est cependant aussi possible d'afficher plus sommairement les images directement depuis le programme Python.

```
# Affiche l'image en niveaux de gris
image_grayscale.show()
```

d) Modifiez une matrice de pixels

i.Exemple 1 :

Ajoutons une ligne verticale blanche (en colonne 100) sur une image :

```
from PIL import Image

# Charge l'image
image = Image.open("Lena.png")

# Obtient les dimensions de l'image
largeur, hauteur = image.size

# Parcourt la colonne 100 et remplace les pixels par des pixels blancs
for y in range(hauteur):
    # (255, 255, 255) représente le blanc
    image.putpixel((100, y), (255, 255, 255))

# Affiche l'image modifiée
image.show()
```

`image.putpixel(i,j,(k))` permet donc d'écrire la valeur k du pixel aux coordonnées (i,j)

Donc pour accéder à un pixel :

- En lecture : `image.getpixel(i,j)` fournit la valeur du pixel en (i,j) ;
- En écriture : `image.putpixel ((i,j),valeur)` permet d'affecter la valeur souhaitée au pixel (i,j).

ii.Exemple 2 :

Modifions les composantes couleurs d'une image : le quart supérieur gauche du haut effacera la première composante de chaque triplet, le quart supérieur droit du haut effacera la deuxième composante de chaque triplet, le quart inférieur gauche du bas effacera la troisième composante, et le quart inférieur droit de l'image restera intact tel quel.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

# Charge l'image
image = Image.open("Lena.png")

# Convertit l'image en un tableau NumPy
image_array = np.array(image)

# Modifie les composantes couleur selon les quarts spécifiés
for i in range(image_array.shape[0] // 2):
    for j in range(image_array.shape[1] // 2):
        # Efface la 1re composante rouge du triplet RGB
        image_array[i, j, 0] = 0

plt.imshow(image_array)
plt.show()

for i in range(image_array.shape[0] // 2):
    for j in range(image_array.shape[1] // 2, image_array.shape[1]):
        # Efface la 2e composante verte du triplet RGB
        image_array[i, j, 1] = 0

plt.imshow(image_array)
```

```
plt.show()

for i in range(image_array.shape[0] // 2, image_array.shape[0]):
    for j in range(image_array.shape[1] // 2):
        # Efface la 3e composante bleue du triplet RGB
        image_array[i, j, 2] = 0

plt.imshow(image_array)
plt.show()
```

On constate sur le résultat que les couleurs de la photo originale de Lena contenaient essentiellement des mélanges de rouge et de vert (et très peu de bleu) car :

- La suppression du rouge faire apparaître une dominante de vert.
- La suppression du vert fait apparaître une dominante de rouge ;
- La suppression de la composante bleue ne change pas beaucoup les couleurs ;

`image_array [i,j, 0]` permet donc d'accéder à la composante d'indice 0 de la valeur du pixel aux coordonnées (i,j) c'est la composante rouge, et donc 1 pour le vert et 2 pour le bleu.

iii.Exemple 3 :

Passons d'une photo noire et blanche à son négatif.

Cette opération, dite d'inverse vidéo, renverse l'ordre des niveaux de gris de l'image (le noir devient blanc, le blanc devient noir, etc.).

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

# Charge l'image en niveaux de gris
image = Image.open("Lena.png").convert("L")

# Convertit l'image en un tableau NumPy
image_array = np.array(image)

# Effectue la transformation des pixels gris
image_array = 255 - image_array

# Affiche l'image résultante en niveaux de gris
plt.imshow(image_array, cmap='gray')
plt.show()
```

Plutôt que de répéter l'opération `image[i,j] = 255 - image[i,j]` en chaque pixel, on a utilisé ici les opérations matricielles pour faire globalement cette opération en une seule instruction, de manière plus concise et plus efficace.

4°) Maîtrisez la construction d'histogrammes et réalisez des modifications simples

Ce chapitre propose un certain nombre d'algorithmes de traitements d'image classiques ou amusants sous formes d'exercices pratiques, afin de vous plonger dans la marmite de pixels !

a) Convertissez une image couleur en noir & blanc (niveaux de gris)

Pour écrire un algorithme qui convertit une image couleur en niveaux de gris, exploitez la formule de luminance Y utilisée en télévision pour :

$$Y=0{,}299.R+0{,}587.V+0{,}114.B$$

```
from PIL import Image
import numpy as np

# Charge l'image
image = Image.open(
    "D:\Cours\Electronique\DUT Info\JFA\BUT\R5.A.06\TDs\TD2\Images\Lena.png")

# Convertit l'image en un tableau NumPy
image_array = np.array(image)

# Sépare les canaux RVB
r, v, b = image_array[:, :, 0], image_array[:, :, 1], image_array[:, :, 2]

# Effectue l'opération matricielle pour obtenir la luminance Y
y = 0.299 * r + 0.587 * v + 0.114 * b

# Convertit les valeurs en octets
y = y.astype(np.uint8)

# Affiche l'image de luminance Y
Image.fromarray(y).show()
```

Remarques :

On a utilisé des opérations matricielles (de multiplication et d'addition) pour éviter d'écrire deux boucles imbriquées répétant ce calcul sur chacun des pixels `y[i,j]`, en fonction de `r[i,j]`, `v[i,j]` et `b[i,j]`.

On a utilisé la fonction `image_array` pour « découper » l'image couleur en 3 tranches séparées pour chaque composante (celles de rouge, vert et bleu).

b) Calculez des histogrammes d'images

L'histogramme d'une image est le graphique qui représente le nombre de pixels existant pour chaque valeur.

Histogramme d'une image en niveaux de gris

Wikipedia : « Pour une image monochrome, c'est-à-dire à une seule composante, l'histogramme est défini comme une fonction discrète qui associe à chaque valeur d'intensité le nombre de pixels prenant cette valeur. La détermination de l'histogramme est donc réalisée en comptant le nombre de pixel pour chaque intensité de l'image. On effectue parfois une quantification, qui regroupe plusieurs valeurs d'intensité en une seule classe, ce qui peut permettre de mieux visualiser la distribution des intensités de l'image. »

Histogramme d'une image couleur

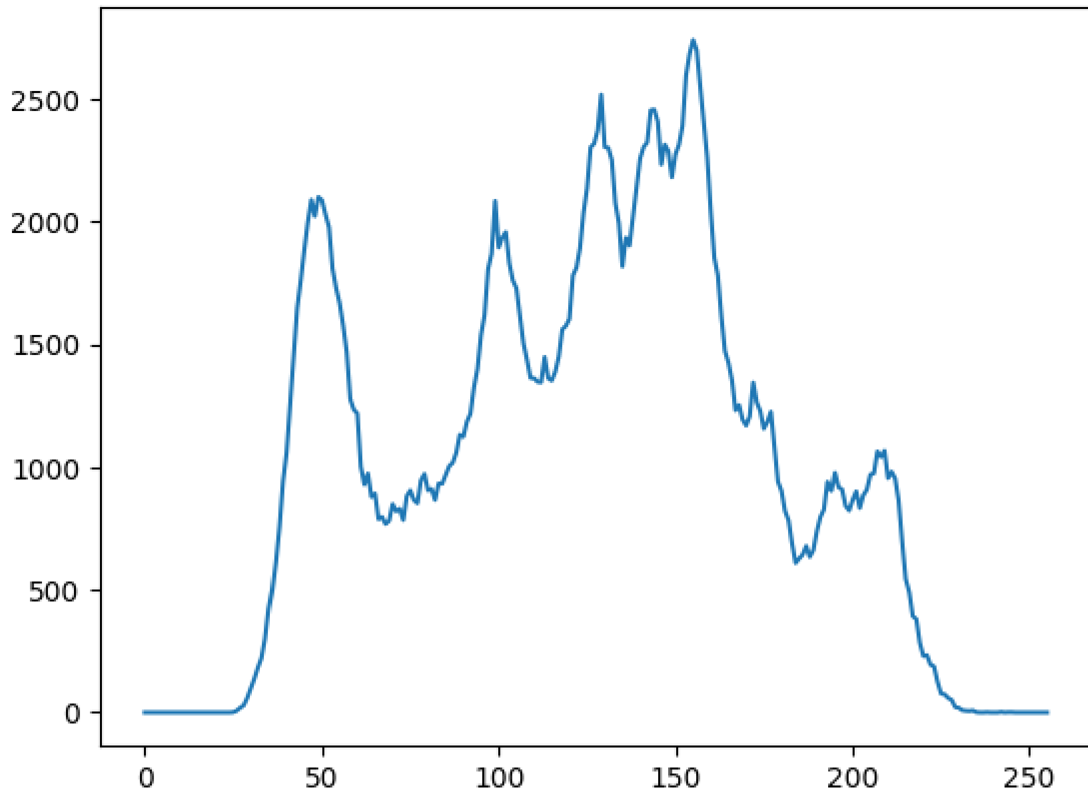
Wikipedia : « Pour les images couleurs, on peut considérer les histogrammes des 3 composantes indépendamment, mais cela n'est en général pas efficace. On construit plutôt un histogramme directement dans l'espace couleur. Les classes de l'histogramme correspondent désormais à une couleur (ou un ensemble de couleurs, en fonction de la quantification), plutôt qu'à une intensité. On parle alors parfois d'histogramme de couleur. »

Calculer l'histogramme de l'image en niveaux de gris, c'est en d'autres termes compter combien il y a de pixels pour chaque nuance de gris.

Voici un algorithme pour calculer, dans un vecteur de taille 256, l'histogramme d'une image en niveaux de gris.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

# Charge l'image en niveaux de gris
image = Image.open(
    "D:\Cours\Electronique\DUT
Info\JFA\BUT\R5.A.06\TDs\TD2\Images\Lena.png").convert("L")

# Convertit l'image en un tableau NumPy
image_array = np.array(image)

# Calcule l'histogramme de l'image
# prépare un vecteur de 256 zéros (pour chaque niveau de gris)
hist = np.zeros(256, dtype=int)
for i in range(image_array.shape[0]):      # énumère les lignes
    for j in range(image_array.shape[1]):  # énumère les colonnes
        hist[image_array[i, j]] += 1

print(hist)
plt.plot(hist)
plt.show()
```

```
[   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    1    7   19
   29   59   97  137  182  221  299  422  499  614  758  944 1060 1255
 1441 1648 1760 1884 1998 2090 2023 2101 2088 2030 1978 1808 1730 1668
 1577 1469 1278 1236 1220 1003  930  975  880  893  789  798  769  782
  850  821  831  785  883  906  867  854  947  973  906  912  867  934
  933  970 1005 1019 1060 1133 1125 1186 1219 1324 1400 1538 1619 1807
 1871 2085 1895 1935 1958 1831 1761 1729 1614 1504 1437 1363 1362 1347
 1345 1448 1362 1352 1388 1450 1561 1575 1604 1781 1813 1893 2037 2137
 2304 2319 2373 2519 2305 2302 2252 2076 1995 1818 1935 1903 2025 2140
 2258 2305 2324 2455 2458 2409 2234 2315 2288 2182 2275 2315 2391 2598
 2685 2742 2701 2556 2410 2261 2041 1853 1781 1613 1473 1425 1353 1233
 1254 1195 1171 1208 1344 1265 1230 1158 1185 1227 1086  942  904  822
  784  689  610  628  644  679  635  662  739  797  828  941  905  978
  919  910  843  824  867  902  834  886  909  970  976 1065 1043 1068
  956  982  955  864  697  544  491  393  381  284  230  234  195  188
  125   77   74   59   51   22   19    9    7    6    8    1    0    0
    1    0    0    0    2    0    1    1    0    0    0    0    0    0
    0    0    0    0]
```



Remarque :

La fonction `plot` de Matplotlib est utilisée pour tracer le graphique représentant la courbe de l'histogramme des valeurs calculées dans le vecteur.

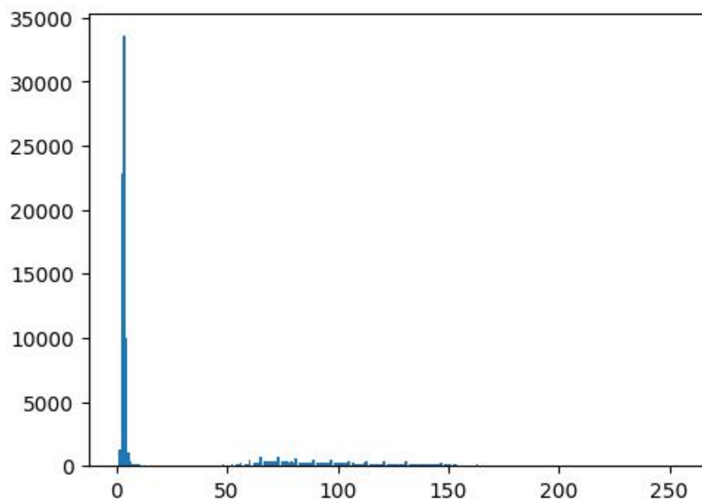
a) Fonction pour calculez des histogrammes d'images

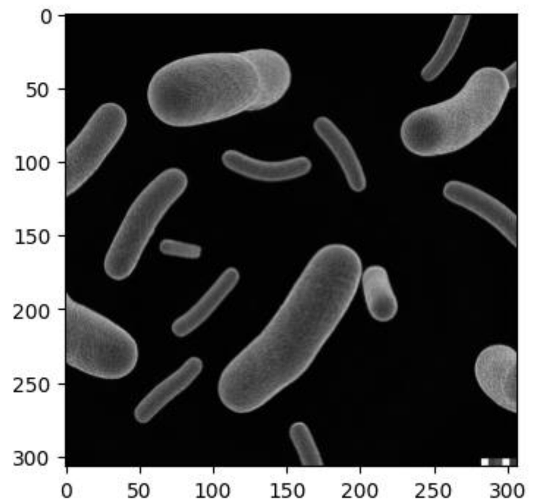
On peut utiliser une fonction MatPlotlib, pour tracer plus facilement l'histogramme d'une image :

```
import matplotlib.pyplot as plt
import numpy as np

# Chargement de l'image et affichage de ses dimensions
img = plt.imread("Images/bactéries.png")
print("Une image PNG est décodée avec des pixels de type",
      img.dtype, "compris entre 0 et 1")
# Nos pixels sont maintenant entre 0 et 255.
img = (img * 255).astype(np.uint8)
print("Après transformation : ", img.dtype, "compris entre 0 et 255")


# Récupération du plan de couleur pour la composante Red
# avec Numpy
img = img[:, :, 0]

# On divise l'espace disponible en une ligne et deux colonnes.
# On obtient deux systèmes d'axes dans le tuple.
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.set_size_inches(12, 4)

# Affichage de l'histogramme pour voir la répartition des pixels
pixels = img.ravel()   # On met l'image à plat
```

```
ax1.hist(pixels, bins=256)

# Affichage de l'image en niveaux de gris
ax2.imshow(img, cmap="gray")

# Affichage de la figure
plt.show()
```

```
Une image PNG est décodée avec des pixels de type float32 compris entre 0 et 1
Après transformation :  uint8 compris entre 0 et 255
```





On observe que les pixels relatifs aux bactéries sont compris entre 25 et 175. Les autres pixels (avant 25) correspondent au fond en noir (ce qui représente la majorité des pixels).

- *https://koor.fr/Python/Tutoriel_Scipy_Stack/matplotlib_hist.wp*

b) Égalisez un histogramme

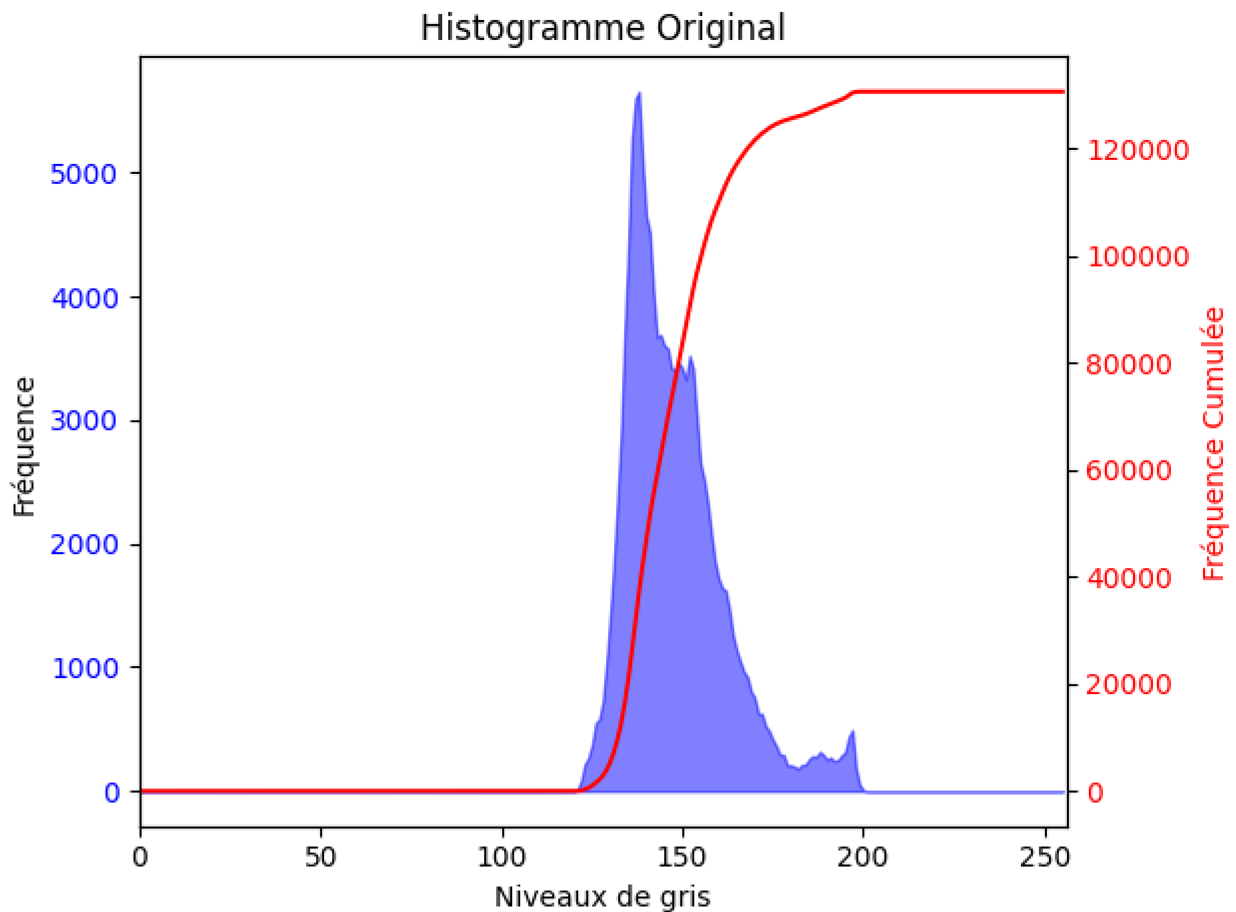
L'égalisation d'histogramme consiste à corriger une image qui manque de contraste : ses couleurs, ou ses niveaux de gris, se concentrent sur seulement quelques valeurs.

Cette transformation va faire une conversion de couleurs :

- Afin d'utiliser toute l'étendue des niveaux ;
- Afin d'avoir (à peu près) autant de pixels de chaque niveau.

L'astuce consiste à d'abord calculer l'histogramme cumulé (le principe est le même que pour l'histogramme, si ce n'est que pour toute valeur `i` on calcule non pas le nombre de pixels de cette valeur dans l'image, mais on cumule le nombre de pixels de valeur égale ou inférieure à `i` dans l'image).

Hawke's Bay : image peu contrastée [source : Wikipédia]



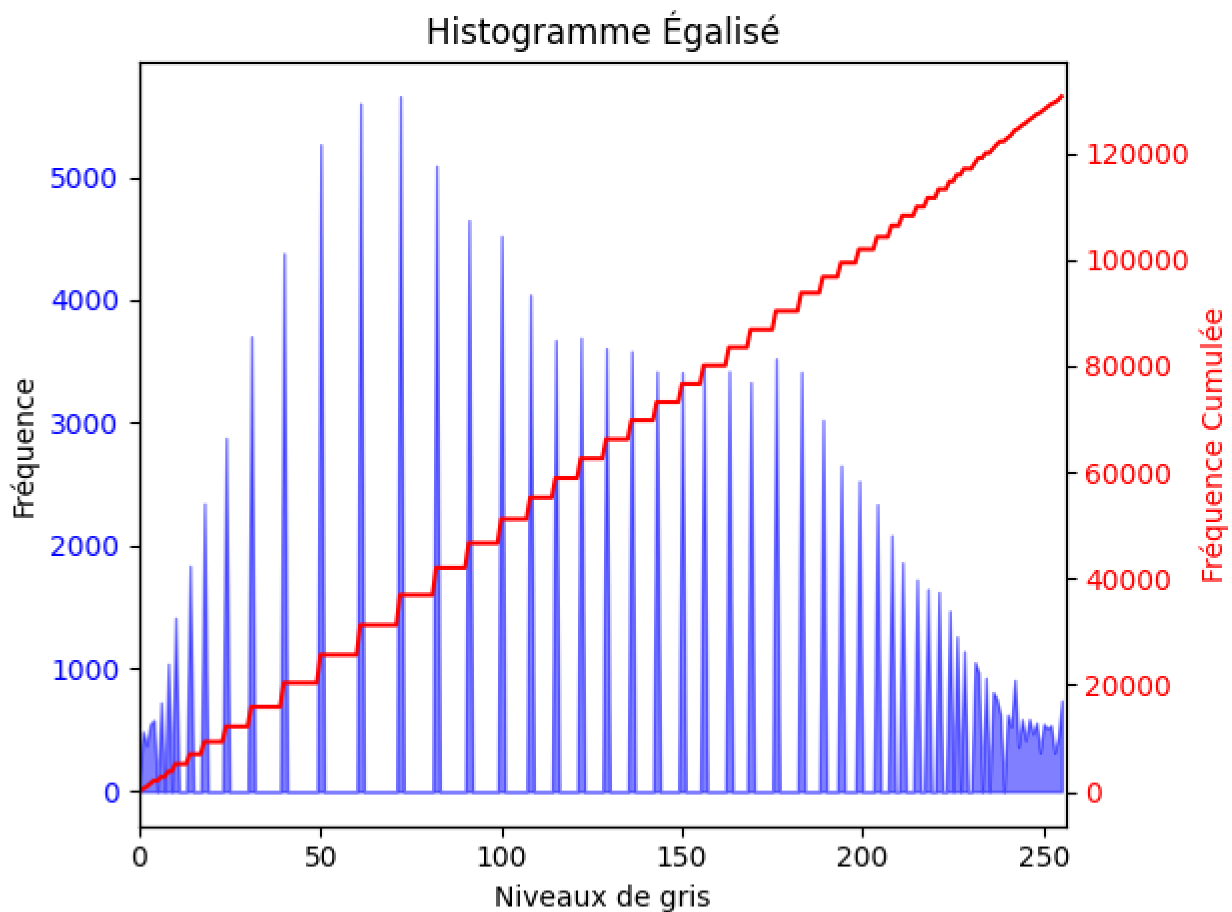
Hawke's Bay : histogramme correspondant (en rouge) ainsi que l'histogramme cumulé (en noir) normalisé en échelle pour rester dans les limites du graphique.

Ensuite, on utilise cet histogramme cumulé directement comme une table de conversion des niveaux de gris (pour cela, on normalise les valeurs de l'histogramme cumulé pour les ramener entre 0 et 255). L'idée de cette conversion étant la suivante :

- Tous les faibles niveaux inutilisés dans l'image (ici de 0 à 120 environ) seront ramenés à 0 ;
- Tous les hauts niveaux inutilisés (ici d'environ 200 à 255) seront ramenés à 255 ;

Les seuls niveaux utilisés seront étalés entre 0 et 255 suivant la progression de l'histogramme cumulé (tout un niveau étant déplacé vers un nouveau niveau, ce qui intercalera des « trous » de niveaux non utilisés dans l'image finale).

Hawke's Bay : image mieux contrastée grâce à l'égalisation de son histogramme [source : Wikipédia]



Le nouvel histogramme correspondant (en rouge, démontrant l'utilisation de toute l'étendue de la dynamique des niveaux du noir au blanc), ainsi que l'histogramme cumulé (en noir, démontrant la répartition régulière).

Voici un algorithme mettant en œuvre le principe précédent pour égaliser l'histogramme d'une image en niveaux de gris, afin d'en améliorer le contraste.

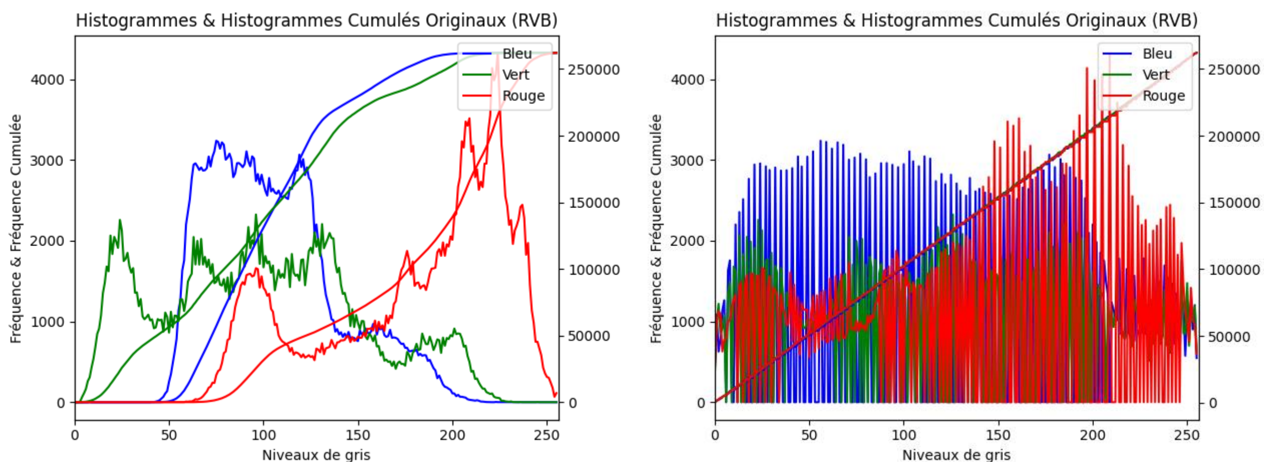
```
from PIL import Image
```

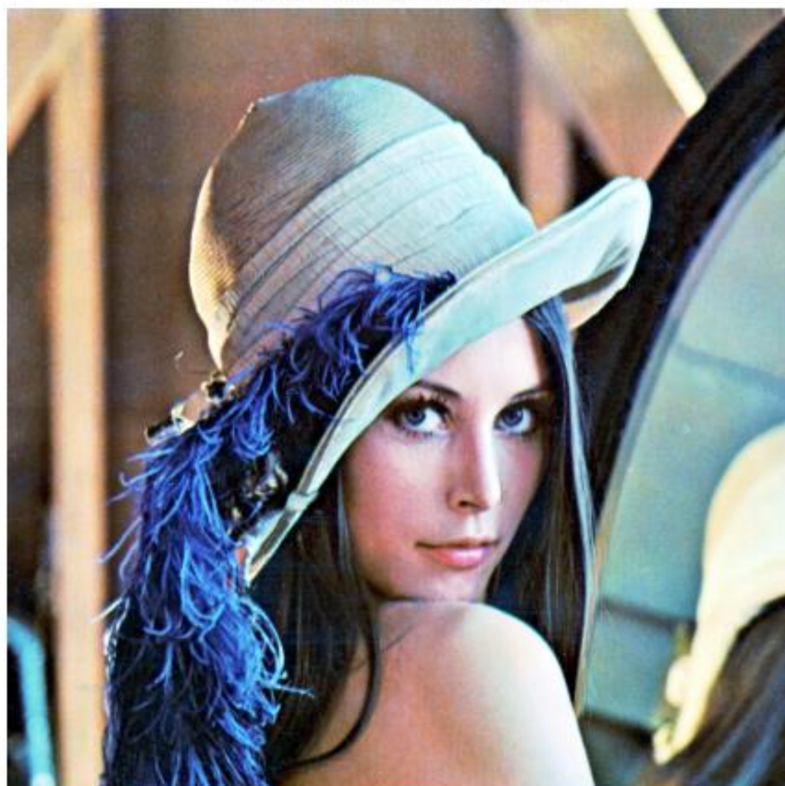
```
import numpy as np
import matplotlib.pyplot as plt

# Charge l'image en niveaux de gris
image = Image.open(
    "D:\Cours\Electronique\DUT Info\JFA\BUT\R5.A.06\TDs\TD2\Images\Hawkes
bay1.png").convert("L")

# Convertit l'image en un tableau NumPy
image_array = np.array(image)

# Calcule l'histogramme de l'image
histo = np.zeros(256, int)
for i in range(image_array.shape[0]):
    for j in range(image_array.shape[1]):
        histo[image_array[i, j]] += 1

# Calcule l'histogramme cumulé hc
hc = np.zeros(256, int)
hc[0] = histo[0]
for i in range(1, 256):
    hc[i] = histo[i] + hc[i-1]

# Normalise l'histogramme cumulé
nbpixels = image_array.size
hc = hc / nbpixels * 255

# Utilise hc comme table de conversion des niveaux de gris
equalized_image_array = hc[image_array]

# Crée une nouvelle image à partir du tableau égalisé
equalized_image = Image.fromarray(equalized_image_array)

# Convertit l'image égalisée en un tableau NumPy de type int
equalized_image_array = equalized_image_array.astype(int)

# Calcule l'histogramme de l'image égalisée
equalized_histo = np.zeros(256, int)
for i in range(equalized_image_array.shape[0]):
    for j in range(equalized_image_array.shape[1]):
        equalized_histo[equalized_image_array[i, j]] += 1

# Affiche l'image d'origine
plt.figure()
plt.imshow(image, cmap='gray')
plt.title('Original')

# Affiche l'histogramme
plt.figure()
plt.plot(histo)
plt.title('Histogram')

# Affiche l'image égalisée
```

```
plt.figure()
plt.imshow(equalized_image, cmap='gray')
plt.title('Equalized')

# Affiche l'histogramme
plt.figure()
plt.plot(equalized_histo)
plt.title('Histogram - Equalized')
plt.show()
```

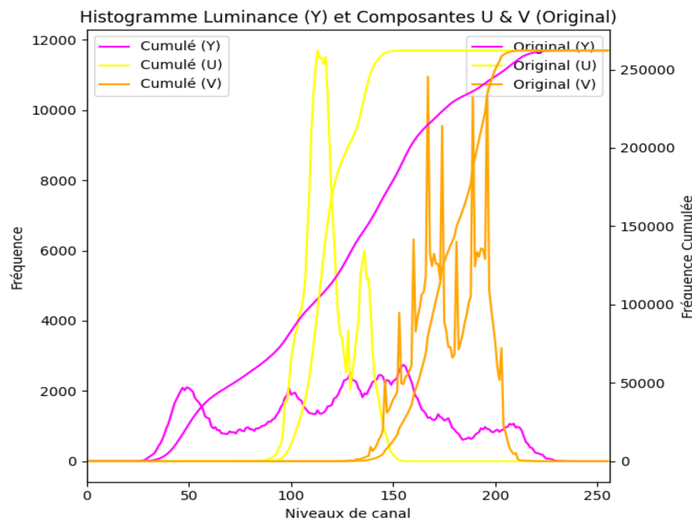
Remarques pour les images couleur :

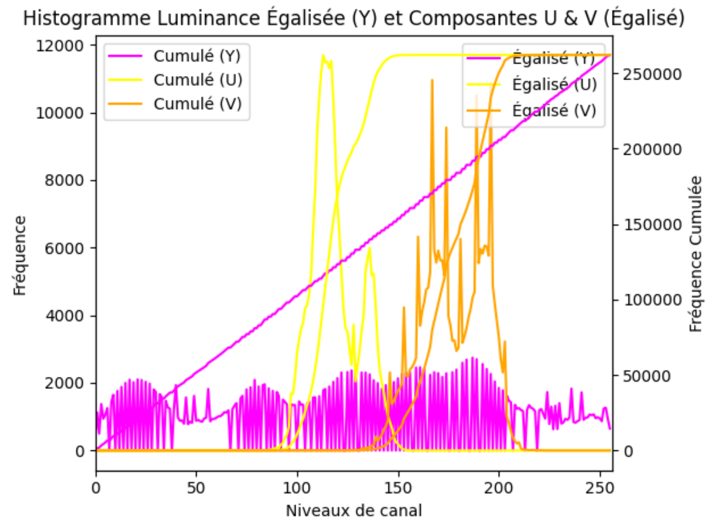
Dans le cas d'une image couleur, on pourrait penser répéter l'égalisation d'histogramme sur les 3 matrices des niveaux R, V et B individuellement. Cependant cette correction séparée va non seulement modifier le contraste final de la luminosité, mais peut aussi faire apparaître des couleurs qui n'existaient pas dans l'image originale (par exemple une zone avec uniquement des niveaux de gris pourra faire apparaître des couleurs, puisque R peut être corrigé différemment de V, lui-même différemment de B).





C'est pourquoi on préfère d'abord convertir l'image dans une représentation avec luminance et chrominance (par exemple YUV), puis effectuer l'égalisation uniquement sur la luminance Y.





B. Webographie

- https://openclassrooms.com/fr/courses/5060661-initiez-vous-aux-traitements-de-base-des-images-numeriques
- https://fr.wikipedia.org/wiki/%C3%89galisation_d%27histogramme
- https://zestedesavoir.com/tutoriels/1557/introduction-au-traitement-dimage/
- https://www.geeksforgeeks.org/opencv-python-program-analyze-image-using-histogram/
- https://koor.fr/Python/Tutoriel_Scipy_Stack/matplotlib_hist.wp
- https://koor.fr/Python/Tutoriel_Scipy_Stack/matplotlib_image.wp