TP2: Manipulation d'images

MP2I Option SI: Informatique tronc commun

On peut représenter une image via une *matrice de pixel*, c'est à dire un tableau 2D où chaque entrée est un pixel. Il existe plusieurs manières d'encoder les pixels [1], on s'intéresse dans ce TP à la plus simple : le codage RGB. Dans ce codage, un pixel est un triplet $(r, g, b) \in [\![0, 255]\!]^3$ correspondant à l'intensité du rouge, du vert et du bleu dans la couleur. Par exemple, $(255, 0, 0)$ représente un rouge pur, $(255, 255, 255)$ représente le blanc, et $(0, 0, 0)$ le noir. Dans la suite on note $\mathbb{P} = [\![0, 255]\!]^3$ l'ensemble des pixels. On peut donc représenter un pixel par un tableau de trois entiers 8 bits (car $2^8 = 256$). Sur python, la bibliothèque `matplotlib.pyplot` contient des fonctions pour charger et sauvegarder des images sous la forme de matrices de pixels, et plus précisément sous la forme de tableaux numpy :

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

a = plt.imread("images/poivrons.jpg") # charge l'image

plt.imshow(a) # dessine l'image dans la fenêtre de pyplot (sans l'afficher)
plt.show() # affiche la fenêtre pyplot

# a est une matrice numpy à trois dimensions.
# Les dimensions s'obtiennent avec a.shape:
n, m, k = a.shape # n lignes et m colonnes de pixels.
# Chaque pixel est composé de k valeurs. Dans la suite, on supposera que
# k vaut toujours 3 et que a[i, j] contient un triplet [r,g,b] représentant
# un pixel par codage RGB. r, g, et b sont des np.uint8, i.e. des entiers 8 bits.

# On ne peut pas modifier a, mais on peut en créer une copie, qui elle est modifiable
b = a.copy() # copie intégrale de a
b[0, 0, 2] = 0 # enlève la composante bleue du pixel en haut à gauche
b[36,45] = np.uint8([255,121,234]) # remplace le pixel ligne 36, colonne 45

b[:, 500] = np.uint8([255, 255, 255]) # remplace toute la colonne 500 par des pixels blanc
b[300, :] = np.uint8([255, 0, 0])     # remplace toute la ligne 300 par des pixels rouges
plt.imsave("poivrons2.jpg", b)        # sauvegarde l'image de la matrice b
```

Une archive est disponible sur Cahier de Prépa pour ce TP, contenant quelques images de test ainsi qu'un fichier Python avec le code précédent.

Attention : les matrices générées par `plt.imread` stockent des entiers 8 bits. Au cours du TP, on se retrouvera à faire des calculs qui peuvent dépasser de 8 bits momentanément (par exemple une somme au cours du calcul d'une moyenne). On pourra utiliser la fonction `np.int16` pour créer une copie d'une matrice en utilisant des entiers 16 bits, sur lesquels les opérations auront moins de chance de causer des dépassements :

```python
a = plt.imread("poivrons.png")
b = np.int16(a) # identique à a, mais chaque valeur est stockée sur 16 bits au lieu de 8
```

1. en.wikipedia.org/wiki/Color_model

1 Manipulation pixel par pixel

Un type de manipulation d'image très simple est la manipulation pixel par pixel. L'idée est de modifier l'image localement, sur chaque pixel, sans tenir compte des valeurs des autres pixels de l'image.

Q1. Écrire une fonction `noR` telle que pour a une matrice de pixels RGB, `noR(a)` est une matrice identique à a mais sans rouge.

Q2. Écrire une fonction `negatif` telle que pour a une matrice de pixels RGB, `negatif(a)` est la matrice représentant l'image négative de a, i.e. où chaque valeur de couleur x a été changée en $255 - x$.

Q3. Écrire une fonction permettant de changer les valeurs RGB de chaque pixel par permutation cyclique :

$$R \to G \quad G \to B \quad B \to R$$

Q4. Une couleur $p \in \mathbb{P}$ est un *niveau de gris* si ses trois valeurs R, G et B sont égales. Écrire une fonction permettant de transformer une image en niveaux de gris en prenant la moyenne des valeurs R, G et B de chaque pixel comme nuance de gris.

Q5. Plutôt que de prendre la moyenne, on considère maintenant la moyenne pondérée par les coefficients $c_R = 0.2126, c_G = 0.7152, r_B = 0.0722$[2]. Coder la fonction correspondante, et comparer les images obtenues avec les deux fonctions de niveaux de gris.

Le but des questions suivantes est de modifier les pixels d'une image selon une palette, c'est à dire un ensemble restreint de couleurs. Le principe est de remplacer chaque pixel de l'image par la couleur de la palette la plus similaire.

Q6. Écrire une fonction `plus_proche_couleur(palette, p)` prenant en entrée une liste de couleurs `palette` dans $\mathbb{P}$ et une couleur $p \in \mathbb{P}$, renvoyant la couleur de `palette` la plus proche de p selon la norme euclidienne dans $\mathbb{P}$.

Q7. Écrire une fonction `reduction_palette(palette, a)` qui modifie la matrice de pixels a pour n'utiliser que les couleurs présente dans la liste *palette*.

Q8. Tester cette fonction avec la palette contenant tous les triplets (r, g, b) avec $r, g, b \in \{0, 128, 255\}$, puis avec une palette faite à la main d'une dizaine de couleurs. Vous pouvez trouver sur internet des sites permettant de choisir une couleur et d'obtenir son code RGB Vous pouvez aussi demander à ChatGPT de vous générer une palette de couleurs sous la forme d'une liste python.

2. Ces valeurs font partie de la "recommandation 709", qui fixe certains standards pour les transmissions audiovisuelles, elles sont choisies par rapport à la façon dont nos yeux perçoivent les couleurs

2 Transformations

On s'intéresse maintenant à des transformations plus globales

Q9. Écrire des fonctions permettant de transformer une image par symétries verticale et horizontale. Ces fonctions renverront une **nouvelle** matrice numpy, sans modifier celles d'entrée.

Q10. Écrire une fonction permettant d'appliquer une rotation de 90° dans le sens horaire, par rapport au centre de l'image.

On fixe $R > 0$, et on considère pour chaque pixel sa distance d au centre de l'image :

— Si $d < R$, on laisse le pixel intact ;

— Si $R \leq d < R\sqrt{2}$ on inverse les couleurs du pixel ;

— Si $R\sqrt{2} \leq d < R\sqrt{3}$ on laisse le pixel intact ;

— Si $R\sqrt{3} \leq d < R\sqrt{4}$ on inverse les couleurs du pixel (i.e. on prend le pixel négatif) ;

— Etc...

Cette transformation aura pour effet de dessiner des anneaux concentriques dans lesquels les couleurs seront alternativement inversés ou laissées intactes.

Q11. Écrire une fonction `anneaux_concentriques` implémentant la transformation précédente.

On veut maintenant appliquer un filtre permettant de pixeliser une image. Étant donné une image source a de taille $n \times m$, et une taille de bloc t, on procède comme suit :

— On divise a en $\lfloor \frac{n}{t} \rfloor \times \lfloor \frac{m}{t} \rfloor$ blocs carrés de taille $t \times t$, de sorte que le carré d'indice (i, j) a pour pixel d'origine (celui en haut à gauche) le pixel d'indice (ti, tj) de a ;

— On réduit chaque bloc à la couleur moyenne de ses pixels ;

— On construit à partir des couleurs moyennes une image de taille $\lfloor \frac{n}{t} \rfloor \times \lfloor \frac{m}{t} \rfloor$.

Q12. Implémenter la transformation décrite ci-dessus.

Transformation du photomaton La transformation du photomaton consiste à diviser l'image en quatre, en répartissant les pixels selon le schéma suivant :

0, 0	0, 1	0, 2	0, 3	0, 4	0, 5	0, 6	0, 7
1, 0	1, 1	1, 2	1, 3	1, 4	1, 5	1, 6	1, 7
2, 0	2, 1	2, 2	2, 3	2, 4	2, 5	2, 6	2, 7
3, 0	3, 1	3, 2	3, 3	3, 4	3, 5	3, 6	3, 7
4, 0	4, 1	4, 2	4, 3	4, 4	4, 5	4, 6	4, 7
5, 0	5, 1	5, 2	5, 3	5, 4	5, 5	5, 6	5, 7

$\longmapsto$

0, 0	0, 2	0, 4	0, 6	0, 1	0, 3	0, 5	0, 7
2, 0	2, 2	2, 4	2, 6	2, 1	2, 3	2, 5	2, 7
4, 0	4, 2	4, 4	4, 6	4, 1	4, 3	4, 5	4, 7
1, 0	1, 2	1, 4	1, 6	1, 1	1, 3	1, 5	1, 7
3, 0	3, 2	3, 4	3, 6	3, 1	3, 3	3, 5	3, 7
5, 0	5, 2	5, 4	5, 6	5, 1	5, 3	5, 5	5, 7

Autrement dit, sur une ligne donnée, les pixels de numéro de colonne pair sont regroupés à gauche, et ceux de numéro de colonne impair sont regroupés à droite, et de même, sur une colonne donnée, les pixels sont séparés selon leur indice de ligne. Ceci donne des résultats proches des photos d'identité que l'on tire dans un photomaton (voir Fig. 1 et 2).

FIGURE 1 – Image source

FIGURE 2 – Image après transformation

Q13. Implémenter la transformation du photomaton.

Q14. Qu'obtient-on si l'on applique cette transformation deux fois d'affilée ?

On peut montrer qu'après un nombre fini d'itérations, on retrouve l'image originale.

Intéressons-nous maintenant au contraste d'une image. Une image est contrastée si elle contient à la fois des zones très sombres et des zones très lumineuses. Le but des questions suivantes est d'arriver à rendre une image contrastée, sans trop la dénaturer.

Étant donné une image en niveaux de gris A à contraster, on procède comme suit :

1. On calcule un tableau T de taille 256, tel que pour tout $i \in [\![0, 255]\!]$, $T[i]$ contient le nombre de pixels de niveau de gris i dans A.
2. On calcule deux indices $0 \leq i \leq j < 256$ tels que pour $k < i$ et pour $k > j$, $T[k] < S$ où S est une certaine valeur seuil.
3. Pour chaque pixel de A, de niveau de gris x, on le remplace par $e(x) = \lfloor 256 * \frac{x-i}{j-i} \rfloor$, en remplaçant les valeurs négatives par 0 et les valeurs supérieures à 255 par 255.

La formule $e(x) = \lfloor 255 * \frac{x-i}{j-i} \rfloor$ est telle que $e(i) = 0$, $e(j) = 255$. Autrement dit, on identifie toutes les couleurs entre 0 et i à 0, toutes les couleurs entre j et 255 à 255, et on étale les couleurs entre i et j pour couvrir toutes les valeurs $0 - 255$.

Q15. Écrire une fonction `clamp(x)` qui renvoie $\lfloor x \rfloor$ si $0 \leq x < 256$, 0 si $x < 0$ et 255 si $x \geq 256$.

Q16. Écrire une fonction `liste_gris(a)` qui renvoie le tableau des occurrences des niveaux de gris d'une image a.

Q17. En utilisant la fonction `plt.bar`[3], écrire une fonction `profil_gris(a)` qui affiche le profil de niveau de gris de l'image a. Autrement dit, on veut un graphique à barres, avec 256 barres, une pour chaque niveau de gris, et tel que la hauteur de la barre numéro i est le pourcentage de pixels de a étant du niveau de gris i.

Q18. Écrire une fonction `indices_seuil(L, S)` qui calcule un couple d'indices (i, j) tels que $L[k] < S$ pour $k < i$ et $k > j$.

Q19. Implémenter une fonction `contraster_gris(a, seuil)` qui applique la procédure de contraste décrite plus haut sur une image a. La tester sur l'image "paysage_fleur.jpg", qui est très peu contrastée.

Pour adapter cette méthode aux images en couleurs, on calcule la liste des niveaux de gris de l'image, on calcule les deux indices i, j décrits plus haut, puis on applique la transformation $x \mapsto \lfloor 256 \times \frac{x-i}{j-i} \rfloor$ à chaque couleur de chaque pixel (i, j) de l'image (sans oublier d'appliquer la fonction `clamp` pour ramener les valeurs dans l'intervalle $[\![0, 255]\!]$).

Q20. Implémenter une fonction `contraster_rgb(a, seuil)` permettant de contraster une image en couleurs. Testez votre fonction sur "paysage_fleur.jpg".

3. Documentation sur matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.bar.html