TD INFORMATIQUE : Traitements d'image

Ce TD comporte deux parties. La première a pour but de construire une image et effectuer des transformations simples sur cette image. La deuxième propose d'extraire des données scientifiques d'une photographie prise en microscopie optique.

I. Introduction.

1) Images en nuances de gris et images couleurs

– Une image numérique en nuances de gris est représentée par un tableau T (*n* lignes et *m* colonnes), dont le terme T[i,k] représente l'état de coloration du pixel (i,k) correspondant. Par convention, T[0,0] est situé en haut à gauche de l'image. Les termes associés aux pixels sont codés sur 256 valeurs entières de type uint8 (np.uint8 défini dans le module Numpy), allant de 0 (pour le noir) à 255 (pour le blanc).

– Dans le cadre du codage RGB (red, green, blue) des couleurs, chaque pixel de l'image numérique (*n* lignes, *m* colonnes) est souvent associé à un triplet de valeurs entières comprises entre 0 et 255 (type uint8). Les images couleurs sont représentées par un tableau T de taille *n* x *m* **x 3**, pouvant être décrit comme la superposition de 3 matrices de taille *n* x *m*, la première T[:, :,0] correspondant à la composante rouge, la deuxième T[:, :,1] correspondant à la composante verte, la troisième T[:, :,2] correspondant à la composante bleue de l'image numérique.

couleur	rouge	vert	bleu roi	blanc	noir	jaune citron	orange
Pixel RGB	(255,0,0)	(0,255,0)	(0,0,255)	(255,255,255)	(0,0,0)	(255,255,0)	(255,120,0)

2) Traitement d'images avec Python

– Plusieurs modules de Python permettent de lire, modifier, créer et/ou sauvegarder des fichiers d'images numériques dans différents formats d'image (.jpg, .tiff, .png,…). Nous utiliserons le **sous-module skimage.io.** On commence donc par importer les modules et sous-modules utiles.

```
import numpy as np
from skimage import io
```

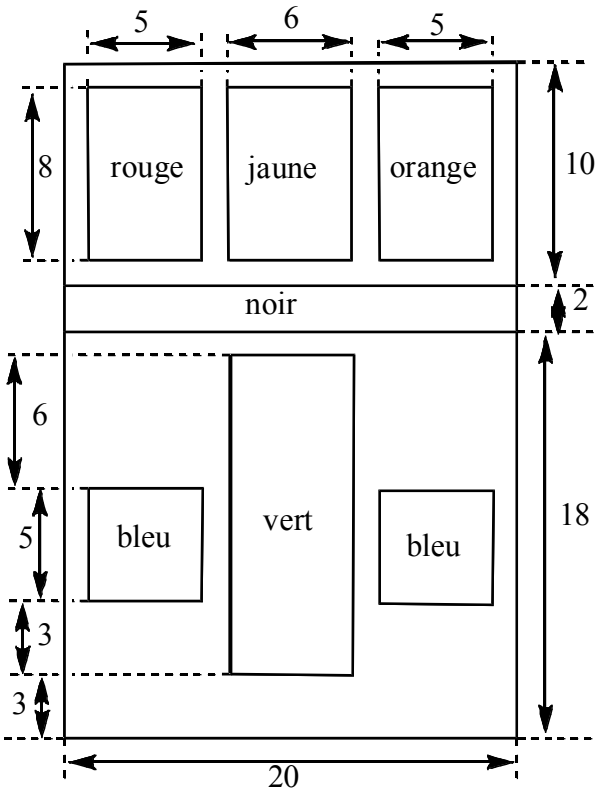
– Quelques commandes et fonctions de Python / Numpy / skimage.io :

np.zeros((10,20,3),dtype=np.uint8)	Crée un tableau 10x20x3 rempli de 0 de type entiers uint8
np.ones((10,20,3),dtype=np.uint8)	Crée un tableau 10x20x3 rempli de 1 de type entiers uint8
np.array(liste, dtype=np.uint8)	Crée un tableau d'entiers uint8 à partir d'une liste
np.array(liste, dtype=float)	Crée un tableau de flottants à partir d'une liste (ou d'un tableau)
np.uint8(M)	Crée un tableau d'entiers uint8 à partir d'un tableau M
np.float32(M)	Crée un tableau de flottants 32 bits à partir d'un tableau M
T.shape ou np.shape(T)	Renvoie les dimensions du tableau numpy T
np.copy(T)	Copie profonde du tableau numpy T
np.vectorize(f)	Permet de vectorialiser une fonction f.
T=io.imread('chemin de l'image')	Permet de lire le fichier image et de le stocker sous forme de tableau T. Le chemin d'accès doit être précisé en chaîne de caractères.
io.imshow(T) # ajouter plt.show() dans certaines versions de Python	affiche l'image d'un tableau numpy T dont les éléments sont des entiers de type np.uint8
io.imsave('/Users/Desktop/ /image.png',T)	Sauvegarde l'image d'un tableau numpy T (chemin d'accès, nom et extension du fichier image précisés en chaîne de caractère)

– Attention : l'arithmétique des entiers de type uint8 est particulière (modulo [256]). Par exemple : 60–100 = –40 ≡ 256 – 40 ≡ 216 [256]. Pour tout calcul utilisant les valeurs des pixels, il sera judicieux de les convertir au préalable en flottants, puis de les reconvertir en entiers uint8 pour l'affichage de l'image modifiée (obtenue par traitement numérique des pixels).

II. Création et modifications d'image.

1) Créer et afficher l'image numérique A comportant 300x200 pixels et schématisée ci-contre, en respectant les cotes relatives et les couleurs précisées sur fond blanc. Le pixel orange peut être obtenu par le triplet (255,120,0) dans le code RGB.



2) Afficher le négatif (couleurs complémentaires) de cette image A

3) La commission internationale de l'éclairage propose d'obtenir la luminance (valeur du gris) d'un pixel à partir de ses composantes RGB par la formule :
Gris naturel = 0,2125.Rouge + 0,7154.Vert + 0,0721.Bleu

Écrire une fonction ***gris*** qui prend un tableau représentant une image couleur RGB comme argument et renvoie un tableau représentant cette image en niveaux de gris.

Appliquer cette fonction à l'image A pour obtenir et afficher l'image en niveau de gris (notée B).

4) Détection de contours pour l'image en niveaux de gris.
On comparera simplement le **gradient d'intensité** entre un pixel et deux de ses voisins (à droite et en dessous), pour obtenir deux composantes (horizontale et verticale) du gradient d'intensité. On utilisera ensuite un seuil pour les pixels afin d'augmenter le contraste du tableau gradient ainsi obtenu. Si le gradient dépasse un certain seuil, le pixel est noirci dans une copie de l'image, sinon il est laissé en blanc.
On fera attention au type des pixels (entier de type np.uint8, entier de type int, flottant, etc.).

a) composante horizontale du gradient :
Écrire une fonction ***gradientH*** qui prend un tableau représentant une image en niveaux de gris comme argument et renvoie un tableau du gradient horizontal des pixels de l'image (comparaison entre pixels de 2 colonnes voisines).

b) composante verticale du gradient :de même, écrire une fonction ***gradientV*** qui renvoie un tableau du gradient vertical des pixels de l'image (entre 2 lignes voisines).

c) Appliquer ces deux fonctions à l'image B (image A en niveau de gris). Afficher l'image correspondant au gradient horizontal de l'image B, puis celle correspondant au gradient vertical de B. Tous les contours ont-ils été détectés par l'un ou l'autre gradient ? Conclure.

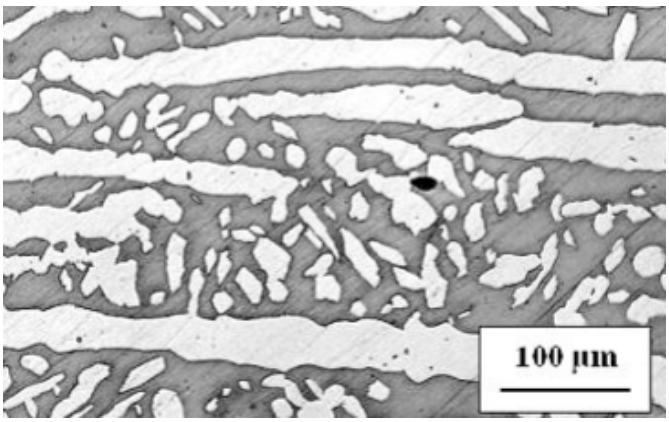
d) Combiner le gradient horizontal et le gradient vertical pour obtenir une approximation de la norme du gradient global et construire le tableau rempli des valeurs des gradients normalisés (et donc des contours). Afficher l'image correspondante. Conclure.

e) Choisir une valeur seuil pour noircir les vrais contours et blanchir le reste.
Remarque : pour choisir la valeur seuil du pixel, on peut s'aider du tracé de l'histogramme des pixels de l'image avec la syntaxe suivante :

```
Nombre_pixels,valeur=np.histogram(image,bins=256)
plt.plot(valeur[ :-1],Nombre_pixels)
plt.show()
```

III. Étude d'image d'un alliage métallique

L'image ci-dessous représente la microstructure d'un acier inoxydable X2CrNiMo 22.5.3 (C2%, Cr22%, Ni5%, Mo3%), obtenue par microscopie optique. Elle montre la présence de 2 phases : la ferrite (sombre, cubique centrée) et l'austénite (claire, cubique à faces centrées).



But : évaluer la proportion volumique de chacune des phases dans cet alliage et la surface de contact entre ces deux phases par unité de volume par traitement de cette image disponible dans le fichier image 'inox.jpg'. On supposera pour cela que l'échantillon d'inox étudié est identique sur toute une tranche d'épaisseur e. On pourra également utiliser les fonctions construites dans la partie précédente.

1) Sauver l'image 'inox.jpg' (ou 'inox.png') sur votre clé USB. Ouvrir l'image comme un tableau Numpy avec la fonction skimage.io.imread. Afficher l'image avec la fonction skimage.io.imshow.

2) Quelles sont les dimensions de l'image fournie ? Quelles sont les valeurs maximales et minimales des pixels ? On pourra proposer une fonction ou utiliser les fonctions np.max et np.min de Numpy.

3) Créer un tableau correspondant à l'image inox.jpg (ou inox.png) représentée en niveaux de gris.

4) Binarisation de l'image en niveau de gris : choisir une valeur seuil permettant de distinguer les pixels sombres des pixels clairs puis créer un tableau booléen, de même taille que celle de l'image en niveau de gris, dont les éléments valent 1 pour les pixels clairs et 0 pour les pixels sombres.

5) Déterminer le pourcentage volumique de chacune des phases dans l'échantillon d'inox photographié. Il faudra veiller à s'affranchir des effets de bord.

6) Détecter les contours de l'image et en déduire la surface de l'interface entre les deux phases par unité de volume pour l'échantillon d'inox étudié.

7) Les données scientifiques (pourcentage volumique des phases, interface entre les phases, etc…) extraites de l'analyse des photos peuvent être stockées dans une base de données permettant de garder leur évolution en fonction du temps (la proportion des phases de l'alliage peut varier au cours du temps). On suppose que la base de données est composée d'une table nommée evol_structure présentant, entre autres, les champs suivants :

– id : identifiant interne à la base de données,
– numéro_acier : identifiant de l'acier étudié
– temps : valeur temporelle correspondant à la date, l'heure, minute et seconde de la prise de la photographie
– pourcentage : pourcentage volumique de la phase austénite
– S : interface entre les phases austénite et ferrite par unité de volume

Un opérateur interroge la base de données pour obtenir un graphique entre deux dates qui seront stockées dans les variables time1 et time2. Il souhaite tracer l'évolution du pourcentage volumique de la phase austénite dans l'échantillon d'acier numéro X2CrNiMo 22.5.3 entre ces deux dates.

a) Donner la requête SQL à envoyer à la base de données permettant de récupérer la valeur des champs temps et pourcentage entre les deux instants time1 et time2.

b) La requête est envoyée et traitée à l'aide d'une fonction traitement_requete(requete) qui retourne une matrice resultat_requete dont la première colonne est le temps en secondes (avec pour origine la date time1) et la seconde la valeur du pourcentage volumique correspondant. Proposer un code pour représenter le pourcentage en fonction du temps, en ne traçant qu'une valeur sur 10. On légendera les deux axes du graphique.