« CORRIGÉ » du TD Informatique : Traitements d'image

II. Création et modifications d'image.

1) et 2) Création et affichage de l'image numérique A + le négatif

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import io

# image A : dessin géométrique
dessin=255*np.ones((300,200,3),dtype=np.uint8) # création d'une image blanche
dessin[100:120,:,:]=0 # bande noire horizontale
dessin[10:90,10:60,1:]=0 # rectangle rouge à gauche
# rectangle orange à droite
dessin[10:90,140:190,1],dessin[10:90,140:190,2]=120,0 # ou écrire :   dessin[10:90,140:190,:]=(255,120,0)

dessin[10:90,70:130,2]=0 # rectangle jaune citron au centre
dessin[130:270,70:130,0],dessin[130:270,70:130,2]=0,0 # rectangle vert au centre
dessin[190:240,10:60,:2],dessin[190:240,140:190,:2]=0,0 # carrés bleus à gauche et à droite
io.imshow(dessin)
#plt.show() nécessaire pour certaines versions de Python
#négatif du dessin
io.imshow(255-dessin)
```

3) image en nuances de gris

```
# fonction pour obtenir un tableau représentant l'image A en niveau de gris
def gris(T):
    lx,ly,lz=T.shape
    M=255*np.ones((lx,ly))   # création du tableau correspondant à l'image en niveaux de gris
    N=np.array(T,dtype=np.float) # conversion pour les calculs en flottants
    for i in range(lx):
        for k in range(ly):
            M[i,k]=0.2125*N[i,k,0]+0.7154*N[i,k,1]+0.0721*N[i,k,2] # les valeurs seront toutes
                                                                    # comprises entre 0 et 255
    return np.array(M,dtype=np.uint8) # conversion des flottants en entiers uint8 pour l'image

B=gris(dessin)      # B est l'image de A (appelé dessin dans ce code ) en niveau de gris
io.imshow(B)


# autre fonction possible utilisant le calcul vectoriel sous numpy (plus efficace)
def gris2(T):
    lx,ly,lz=T.shape
    N=np.array(T,dtype=np.float) # conversion pour les calculs en flottants
    M=0.2125*N[ :, :,0]+0.7154*N[ :, :,1]+0.0721*N[ :, :,2] # tableau correspondant à l'image en niveaux de gris
    return np.array(M,dtype=np.uint8) # conversion des flottants en entiers uint8 pour l'image

B=gris2(dessin)    # B est l'image de A (appelé dessin dans ce code ) en niveau de gris
io.imshow(B)
```

4) a) et b) Détection de contours pour l'image en niveaux de gris.

```
# gradient horizontal pour une image en niveau de gris (comparaison de pixels voisins entre 2 colonnes)
def gradientH(a):
    lx,ly=a.shape
    H=255*np.ones((lx,ly))
    fichier_gris=np.array(a, dtype=np.float) # conversion des pixels en float pour calcul du gradient
    for i in range(lx):
        for k in range(ly-1):
            H[i,k]=abs(fichier_gris[i,k+1]-fichier_gris[i,k])
    return H      # attention : H est un tableau de flottants
```

```
# gradient vertical pour une image en niveau de gris (entre 2 lignes)
def gradientV(a):
	lx,ly=a.shape
	V=255*np.ones((lx,ly))
	fichier_gris=np.array(a, dtype=np.float) # conversion des pixels en float pour calcul du gradient
	for k in range(ly):
		for i in range(lx-1):
			V[i,k]=abs(fichier_gris[i+1,k]-fichier_gris[i,k])
	return V    # V est un tableau de flottants
```

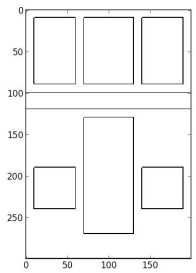
4c) L'affichage des gradients horizontaux et verticaux montre qu'il n'est pas possible de détecter tous les contours en ne prenant en compte que la composante horizontale ou verticale du gradient d'intensité. Il est nécessaire de combiner ces deux composantes pour obtenir la norme du gradient global.

```
# Affichage des images des gradients horizontaux et verticaux de l'image B
B=gris(dessin)
HB=gradientH(B)
VB=gradientV(B)
io.imshow(np.array(255-HB,dtype=np.uint8)) # conversion flottant en entier uint8 (gradient faible=>noir)
io.imshow(np.array(255-VB,dtype=np.uint8)) # afficher le négatif de l'image pour visualiser les contours
```

4d) En utilisant les deux composantes du gradient, on a effectivement détecté les contours.

```
#-------------------------------------------------------------
# matrice du gradient global sqrt(H**2+V**2) et visualisation des contours (avec seuil)
#-------------------------------------------------------------
gradient=np.sqrt(HB**2+VB**2) # np.max(gradient) donne 360,62 et np.min(gradient) donne 0
gradient_norm=(255/np.max(gradient))*gradient # normalisation du gradient
gradient_norm=np.array(gradient_norm,dtype=np.uint8)
io.imshow(255-gradient_norm) # pour visualiser le gradient normalisé en niveaux de gris
```

4e) Après le choix du seuil, on peut obtenir des contours plus nets (bien contrastés)




```
# choix du seuil
nombre,valeur=np.histogram(gradient_norm,bins=256)
plt.plot(valeur[:-1],nombre)
plt.show() # choix du seuil=10 pour enlever les parties sombres du tableau gradient (gradient très faible)

# tableau du gradient normalisé avec seuil=10 ----------------------------------------
# première possibilité : avec des boucles pour sélectionner les pixels des contours (à noircir)
lx,ly=gradient_norm.shape
G_seuil=255*np.ones((lx,ly),dtype=np.uint8) # tableau image de même dimension que gradient_norm
seuil=10
for i in range(lx):
	for k in range(ly):
		if gradient_norm[i,k]>seuil: # il s'agit d'un contour => à noircir
			G_seuil[i,k]=0
		else: # pas de contour => à blanchir
			G_seuil[i,k]=255
io.imshow(G_seuil)

# autre possibilité : passage par un tableau de booléen
seuil=10
G_seuil=np.array(gradient_norm)<seuil #renvoie 1 tableau de booléen avec True pour les parties noires
                                      # de gradient (absence de contours à blanchir)
G_seuil=np.array(255*G_seuil,dtype=np.uint8) # conversion en image : 255*True=255 et 255*False=0
```

```
io.imshow(G_seuil)
```

III. Étude d'image d'un alliage métallique

1 et 2) Pour chercher le chemin d'accès du fichier, sélectionner le fichier avec la souris, clic droit puis propriétés...

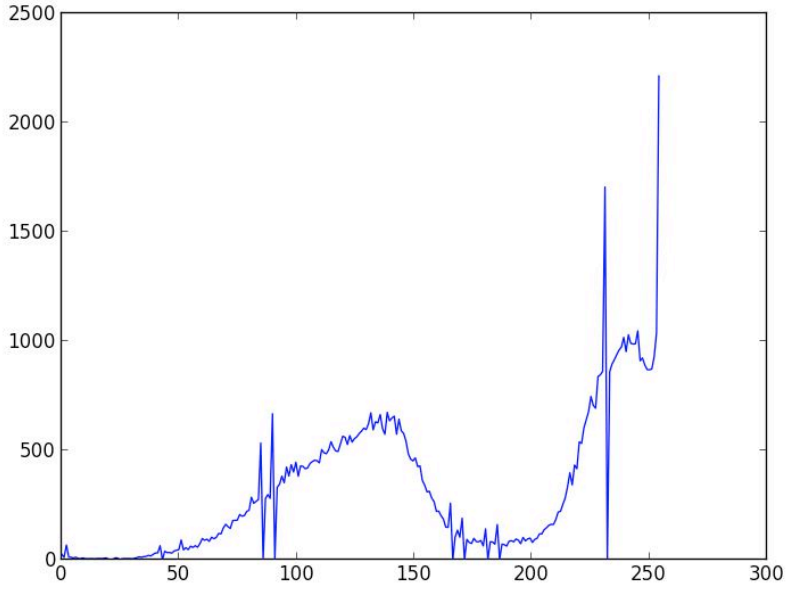
```
inox=io.imread('/Users/Desktop/TD-info-image/inox.jpg')
print inox.shape #(224,363,3) => image couleur RGB !
print np.max(inox) # 255
print np.min(inox) # 0
io.imshow(inox)

# fonctions pour déterminer le maximum et le minimum d'un tableau à 3 dimensions
def maxi(t):
    m=t[0,0,0]
    lx,ly,lz=t.shape
    for i in range(lx):
        for k in range(ly):
            for p in range(lz):
                if t[i,k,p]>m:
                    m=t[i,k,p]
    return m
print maxi(inox) # 255
print mini(inox) # 0
```

```
def mini(t):
    m=t[0,0,0]
    lx,ly,lz=t.shape
    for i in range(lx):
        for k in range(ly):
            for p in range(lz):
                if t[i,k,p]<m:
                    m=t[i,k,p]
    return m
```

3) On peut soit appliquer la fonction *gris* au fichier inox, soit ne prendre qu'une composante (R,G ou B) du tableau inox : gris(inox) ou inox[:, :,1] par exemple.

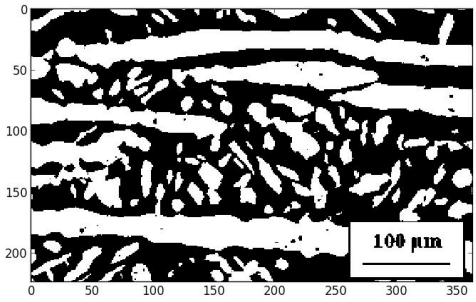
4) Binarisation de l'image

```
# image en niveau de gris
inox1=gris(inox)   # ou utiliser qu'une seul composante, composante verte par exemple : inox2=inox[:,:,1]
# choix du seuil
nombre,valeur=np.histogram(inox1,bins=256)
plt.plot(valeur[:-1],nombre)
plt.show()    # choix du seuil=190 pour séparer les zones claires (pixels>200) et sombres (pixels<160)
```



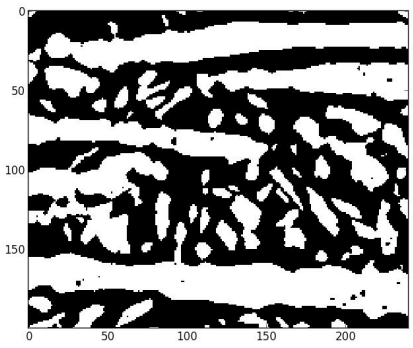

```
# binarisation de l'image : deux exemples de fonction
def binarisation_bool(image_gris,seuil):
    t=image_gris>seuil  # tableau contenant true si pixel>seuil (partie claire)
    return t

def binarisation(image_gris,seuil):
```

```
    lx,ly=image_gris.shape
    t=np.ones((lx,ly))
    for i in range(lx):
        for k in range(ly):
            if image_gris[i,k]<seuil:   # pour partie sombre (pixel<seuil)
                t[i,k]=0
    return t

inox_bool=binarisation_bool(inox1,190)
inox_bin=binarisation(inox1,190)
io.imshow(inox_bool) # image en noir et blanc
#io.imshow(inox_bin)
```



5) On a : % volumique de l'austénite = $V_{\text{austénite}}/V_{\text{total}}=e.S_{\text{claire}}/(e.S_{\text{totale}})$
=> % austénite = nbre pixels clairs/nbre total de pixels de l'image inox.jpg originale

```
#-----pourcentages volumiques des deux phases : pixel=0 pour ferrite et pixel=1 pour austénite----
# sélection d'une partie de l'image binarisée, pour s'affranchir des effets de bord
inox_bool_extrait=inox_bool[10:210,10:250]
inox_bin_extrait=inox_bin[10:210,10:250]
io.imshow(inox_bool_extrait)
#io.imshow(inox_bin_extrait)
```




```
lx,ly=inox_bool_extrait.shape
print ('le pourcentage volumique en austénite vaut '+str(float(np.sum(inox_bool_extrait))/(lx*ly)))
print ('le pourcentage volumique en ferrite vaut '+str(1.0-float(np.sum(inox_bool_extrait))/(lx*ly)))
# attention à convertir les entiers en flottants
```

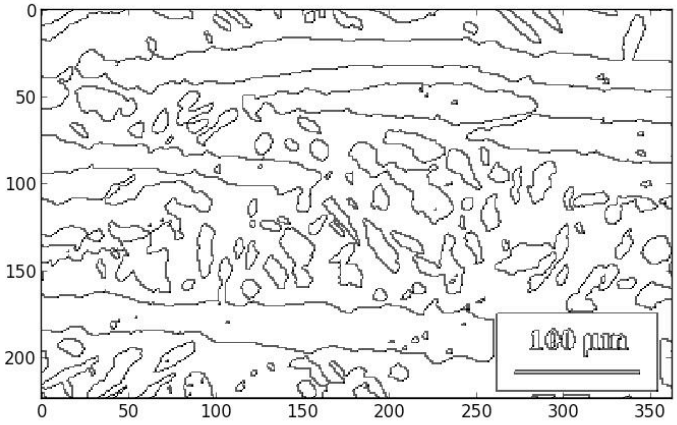
 # réponse : 46,3% en austénite (claire) et 53,7% en ferrite (sombre)

```
# au lieu d'utiliser la fonction np.sum, on peut proposer une fonction somme applicable à un tableau
def somme_t(t):
    a,b=t.shape
    s=0
    for i in range(a):
        for k in range(b):
            s+=t[i,k]
    return s
```

```
print ('le pourcentage volumique en austénite vaut '+str(float(somme_t(inox_bool_extrait))/(lx*ly)))
print ('le pourcentage volumique en ferrite vaut '+str(1.0-float(somme_t(inox_bool_extrait))/(lx*ly)))
```

6) On a : $S_{interface}/V_{total}$ = e.contours / (e.S_{totale}) = nbre pixels des contours/nbre de pixels total, ce qui donnera une réponse dépendant du pixel. On utilisera la réglette 100 µm, correspondant à 75 pixels, présente sur la photo pour obtenir un rapport en μm^{-1} ou m^{-1}.

```
#---- détection des contours (sur des images en noir et blanc au départ) et surface de l'interface
# on peut utiliser les fonctions gradient précédentes après conversion du tableau en niveaux de gris
inox_gradH=gradientH(np.array(255*inox_bool,dtype=np.uint8))
inox_gradV=gradientV(np.array(255*inox_bool,dtype=np.uint8))
inox_gradient=np.sqrt(inox_gradH**2+inox_gradV**2)
norm=(255/np.max(inox_gradient))*inox_gradient # normalisation du gradient
norm=np.array(norm,dtype=np.uint8)
#io.imshow(255-norm) # pour visualiser le gradient normalisé en niveaux de gris

# choix du seuil
nombre,valeur=np.histogram(norm,bins=256)
plt.plot(valeur[:-1],nombre)
plt.show() # choix du seuil=150 pour enlever les parties sombres du tableau gradient (gradient très faible)
seuil=150
norm_seuil=np.array(norm)<seuil #tableau de booléen avec True pour les parties noires de gradient (absence de contours à
blanchir)
norm_seuil=np.array(255*norm_seuil,dtype=np.uint8) # conversion en image : 255*True=255 et 255*False=0
io.imshow(norm_seuil) # pour visualiser les contours
```




```
# sélection d'une partie de l'image binarisée, pour s'affranchir des effets de bord
contours_extrait=norm_seuil[10:210,10:250]
lx,ly=contours_extrait.shape
# attention : le tableau contours_extrait est rempli de 0 (contours) et de 255 (absence de contours)
# pour obtenir le nombre de pixels sombres des contours, faire la somme du négatif du tableau extrait, puis diviser par 255
a=float(np.sum(255-contours_extrait))/(255*lx*ly)
print ('le rapport interface sur volume total vaut '+str(a)+' par pixel')
```

 # réponse : S/V vaut 0,11 par pixel

```
# utilisation du témoin indiquant 100 micromètres pour 345-270=75 pixels
print('le rapport interface sur volume total vaut '+str(75.0*a/100.0)+' par micromètre')
```

 # réponse : S/V vaut 0,083 μm^{-1} ou 8,3.10^4 m^{-1}.

7) a) requete = SELECT temps,pourcentage FROM evol_structure WHERE numéro_acier = X2CrNiMo 22.5.3 AND temps > time1 AND temps < time2

b)
```
import matplotlib.pyplot as plt
plt.plot(resultat_requet[0 :–1 :10,0], resultat_requete[0 :–1 :10,1],'k+')
```

```
plt.xlabel(‘Temps (secondes)’)
plt.ylabel(‘Pourcentage austénite’)
plt.show()
```