AIV1 - TP10
Segmentation non supervisée par analyse d'histogramme

Suliac Lavenant et Antoine Nollet

25 March 2022

Table des matières

1 Introduction

Dans le cadre de ce TP l'objectif sera d'utiliser l'apprentissage non-supervisé (contrairement au précédent TP) via seuillage par la méthode d'Otsu des niveaux de gris afin de segmenter les différentes parties d'une image.

2 Binarisation non supervisée par méthode d'OTSU

Nous reprenons une image utilisée lors du précédent TP contenant deux classes distinctes :

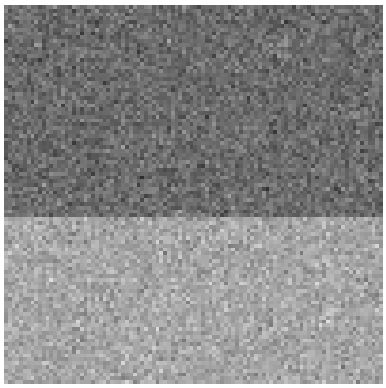

FIGURE 1 – Image 2classes_100_100_8bits

La méthode d'Otsu, avec deux classes, cherchera à trouver le seuil t qui maximisera la dispersion entre les classes.

Voici les probabilités à priori P ainsi que les tailles N des classes $\omega_k(t)$:

$P(\omega_1(t)) = \sum_{i=0}^{t} \frac{h(i)}{N}$

$N(\omega_1(t)) = \sum_{i=0}^{t} h(i)$

$P(\omega_2(t)) = 1 - P(\omega_1(t))$

$N(\omega_2(t)) = N - N(\omega_1(t))$

Les moyennes des classes $\omega_k(t)$:

$\mu(\omega_1(t)) = \frac{\sum_{i=0}^{t} i.h(i)}{N(\omega_1(t))}$

$\mu(\omega_2(t)) = \frac{\sum_{i=t+1}^{255} i.h(i)}{N(\omega_2(t))}$

On cherche ainsi à maximiser la dispertion entre les classes via un seuil $\hat{t}$ optimal, c'est à dire où $\hat{t} = argmax(\sigma(t))$.

Et où $\sigma(t) = P(\omega_1(t)).P(\omega_2(t)).(\mu(\omega_1(t)) - \mu(\omega_2(t)))^2$

L'exécution de la méthode d'Otsu sur notre image (voir l'Annexe "Macro Exercice 1") retourne une dispersion (variance) maximale de 859.43 pour un seuil de 135.

Avec ce seuil, que nous avons obtenu grâce à la méthode d'Otsu, nous obtenons l'image segmentée suivante :

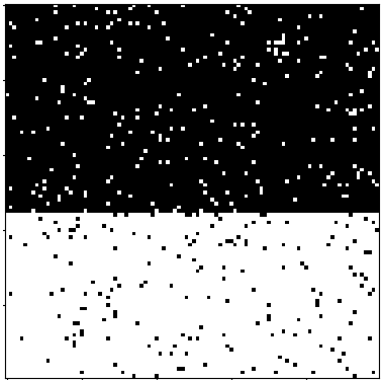

FIGURE 2 – Image 2classes_100_100_8bits segmentée avec le seuil 135

Cette image ainsi segmentée dispose d'un taux de bonne classification de 95.88%.

Lors du tp précédent, avec l'apprentissage supervisé via le seuillage automatique de Bayes, nous avions trouvé un seuil optimal de 141 avec un taux de bonne classification de 95.78%.

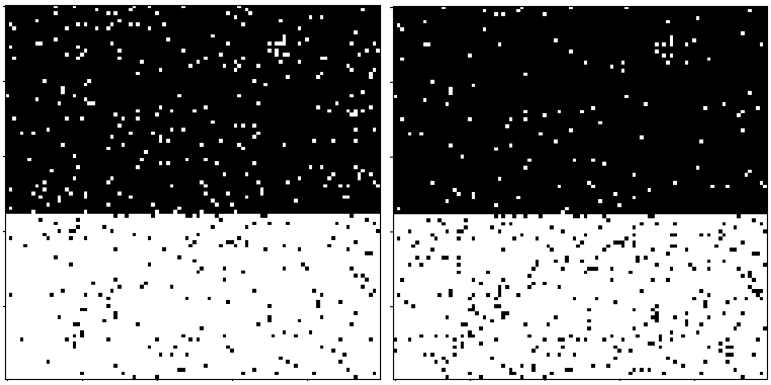

FIGURE 3 – A gauche l'image segmenté selon Otsu et a droite celle segmenté du premier tp

L'apprentissage non-supervisé est donc ici bénéfique car en plus de ne pas avoir besoin d'images de références (des classes que nous voulons reconnaître), nous obtenons un meilleur taux de bonne classification.

3 Classification non supervisée en 3 classes de l'image par la méthode d'Otsu

L'apprentissage non-supervisé via méthode d'Otsu a été bénéfique pour une image avec deux classes différentes, il reste à constater de ce qu'il en est pour une image à trois classes différentes... Voici donc l'image à trois classes utilisée lors du précédent TP :

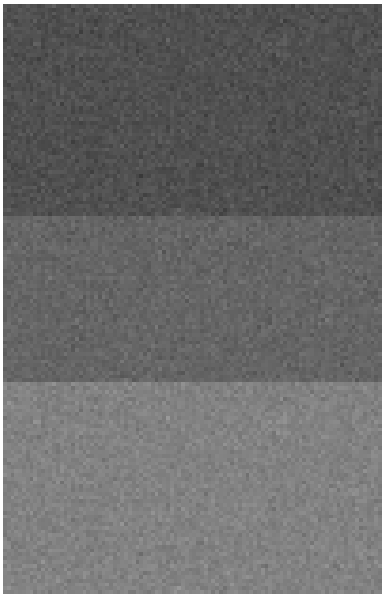

FIGURE 4 – Image 3classes_100_156_8bits

La méthode d'Otsu, avec plus de deux classes, cherchera à minimiser la dispersion dans les classes une à une (dipersion intra-classe). Précédemment on maximisait la dispersion entre les deux classes (inter-classe), qu'elles soient éloignées et donc bien distinctes. Maintenant, on veut minimiser la dispersion au sein de chaque classe (intra-classe), que les classes soient le plus compacte que possible afin d'inclure le moins possible les autres classes.

Ici nous considérons 3 classes séparées par 2 seuils. Ainsi, voici les probabilités selon les seuils t1 et t2 pour qu'un pixel de l'image appartiennent à l'une des trois classes ω_1, ω_2 ou ω_3 :

$$P(\omega_1(t1)) = \sum_{i=0}^{t1} \frac{h(i)}{N}$$

$$P(\omega_2(t1,t2)) = \sum_{i=t1+1}^{t2} \frac{h(i)}{N}$$

$$P(\omega_3(t2)) = \sum_{i=t2+1}^{255} \frac{h(i)}{N}$$

Nous considérons les tailles de ces classes (nombres de pixels appartenants à la classe) comme étant les suivants :

$$N(\omega_1(t1)) = \sum_{i=0}^{t1} h(i)$$

$$N(\omega_2(t1,t2)) = \sum_{i=t1+1}^{t2} h(i)$$

$$N(\omega_3(t2)) = \sum_{i=t2+1}^{255} h(i)$$

Avec ces tailles, on peut déterminer la valeur moyenne de chaque classes :

$$\mu_1(t1) = \sum_{i=0}^{t1} \frac{i * h(i)}{N(\omega_1(t1)}$$

$$\mu_2(t1,t2) = \sum_{i=t1+1}^{t2} \frac{i * h(i)}{N(\omega_2(t1,t2)}$$

$$\mu_3(t2) = \sum_{i=t2+1}^{255} \frac{i * h(i)}{N(\omega_3(t2)}$$

Avec ces valeurs moyennes, nous pouvons donc calculer les variances de chaque classe comme ceci :

$$\sigma_1(t1) = \sum_{i=0}^{t1} \frac{(i - \mu_1(t1))^2.h(i)}{N(\omega_1(t1))}$$

$$\sigma_2(t1,t2) = \sum_{i=t1+1}^{t2} \frac{(i - \mu_2(t1,t2))^2.h(i)}{N(\omega_2(t1,t2))}$$

$$\sigma_3(t2) = \sum_{i=t2+1}^{255} \frac{(i - \mu_3(t2))^2.h(i)}{N(\omega_3(t2))}$$

Avec les variances de chaque classe de l'image, accompagnées de leur probabilité d'apparition dans l'image, nous pouvons calculer la variance intra-classe $\sigma(t1,t2)$ de l'image :

$$\sigma(t1,t2) = P(\omega_1(t1)).\sigma_1(t1) + P(\omega_2(t1,t2)).\sigma_2(t1,t2) + P(\omega_3(t2)).\sigma_3(t2)$$

La méthode d'Otsu consistera donc à déterminer les seuils optimaux $\hat{t1}$ et $\hat{t2}$ qui minimiseront la valeur de $\sigma(t1,t2)$.

$$(\hat{t1},\hat{t2}) = argmin(\sigma(t1,t2))$$

L'exécution de la méthode d'Otsu sur notre image (voir l'Annexe "Macro Exercice 2") retourne une dispersion (variance) minimale de 33.69308536001359 pour un seuil de 90 et de 115.

Avec ce seuil, que nous avons obtenu grâce à la méthode d'Otsu, nous obtenons l'image segmentée suivante :

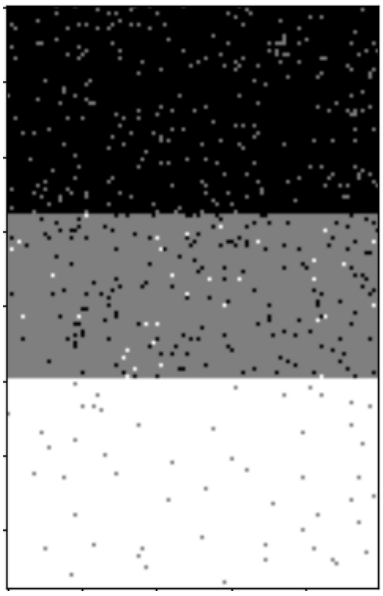

FIGURE 5 – Image 3classes_100_156_8bits segmentée avec les seuils 90 et 115

Cette image ainsi segmentée dispose d'un taux de bonne classification de 95.86%.
Nous avions, avec l'apprentissage supervisé par seuillage de Bayes, un taux de bonne classification de 95.95% . Tout ceci reste cohérent car nous obtenons approximativement les mêmes taux et surtout qu'en général et qu'intuitivement la bonne classification est plus simple lorsque nous avons déjà une connaissance des classes à reconnaître.

4 Segmentation d'une couleur par binarisation des canaux

Si nous réussissons à segmenter une image en niveaux de gris grâce à la détermination d'un seuil par la méthode d'Otsu, constatons ce qu'il en est pour des images couleurs...

On part de l'image IMAGE3D.TIF suivante :

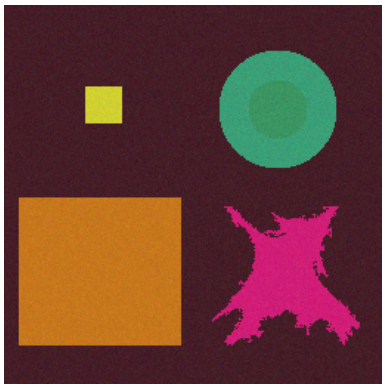

FIGURE 6 – Image IMAGE3D

On la sépare ensuite en 3 canaux : R G et B :

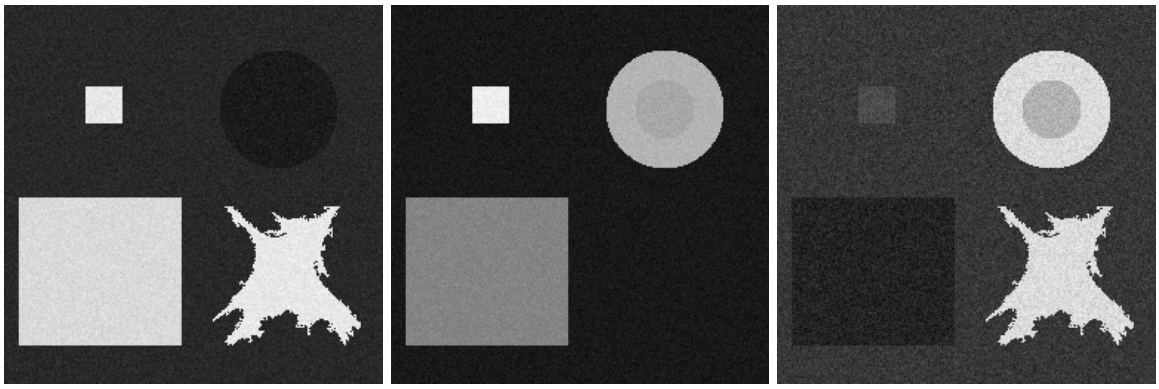

FIGURE 7 – les 3 canaux de l'image IMAGE3D (R,G,B)

On détermine leurs histogrammes de niveaux de gris qui sont donc les suivants :

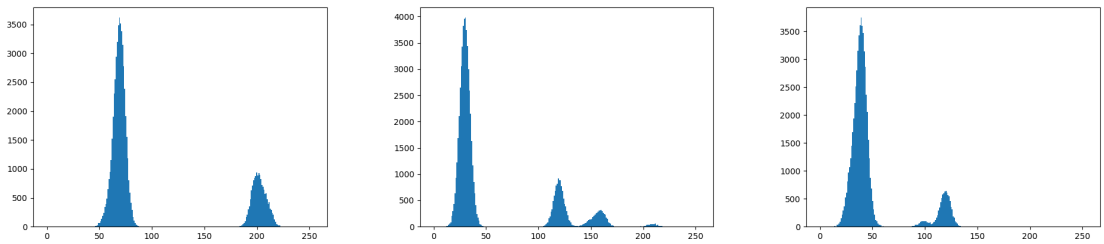

FIGURE 8 – les histogrames des 3 canaux de l'image IMAGE3D (R,G,B)

On binarise ensuite ces trois composantes de manière non supervisée avec la méthode d'Otsu (comme vu dans la section "Binarisation non supervisée par méthode d'OTSU", voir l'Annexe "Macro Exercice 3") :

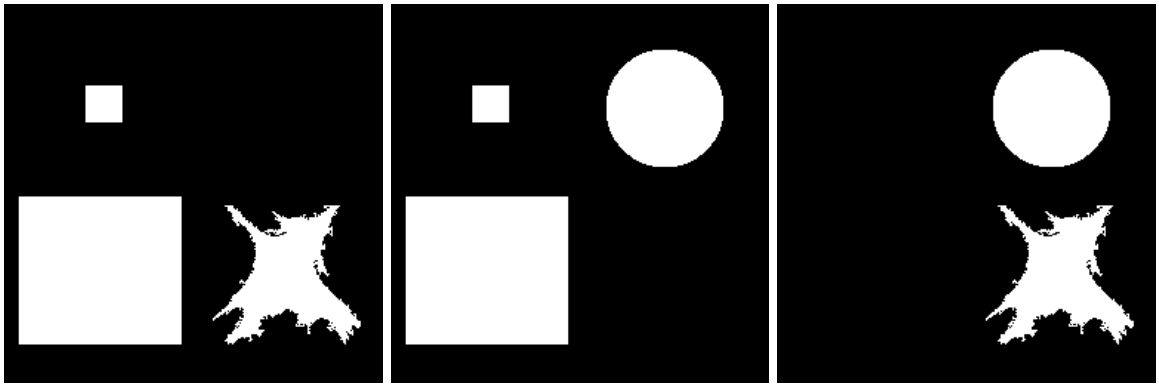

FIGURE 9 – les 3 canaux binarisé de l'image IMAGE3D (R,G,B)

Les seuils ici retenu pour ces trois composantes sont respectivement 89, 49 et 64.

On peut maintenant refusionner ces trois composantes pour revenir a une image couleur :

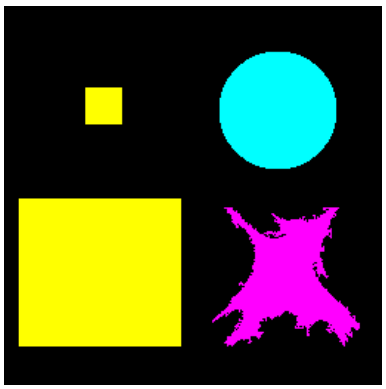

FIGURE 10 – Image finale

Ainsi, nous pouvons constater que la méthode d'Otsu est tout à fait utilisable aussi bien avec des images en niveaux de gris qu'avec des images couleurs.

A Macro Exercice 1

```python
#Suliac Lavenant et Antoine Nollet

import numpy as np
import cv2
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

#ouvre une image en niveau de gris et v rifie qu'elle est bien de dimension 2
def openImg(name):
    image = plt.imread( 'images/' + name + '.png' )
    if image.ndim > 2:
        image = image[:,:,0]

    image[:,:]=(image[:,:]*255).astype(np.uint8)

    return image

#affiche une image en niveau de gris
def showGreyImage(image):
    plt.imshow(image, cmap='gray' )
    plt.show()

#affichage de l'histogramme de image
def showHistogrameOf(image):
    nbins=255
    counttsg, edgesg = np.histogram(image, bins=nbins, range=(0,255))
    plt.hist(edgesg[:-1], nbins, weights=counttsg)
    plt.show()

#seuille l'image image selon le seuil seuil
def seuillage(image, seuil):
    return np.where(image > seuil, 255, 0)

#seuille l'image image selon le seuil seuil
def seuillageInv(image, seuil):
    return np.where(image > seuil, 0, 255)

#seuille l'image image selon les seuils seuil1 et seuil2
def doubleSeuillage(image, seuil1, seuil2):
    imageSeuil1 = np.where(image > seuil1, 127, 0)
    imageSeuil2 = np.where(image > seuil2, 255, 0)
    return np.maximum(imageSeuil1,imageSeuil2)

#calcule la matrice de confusion de l'image image
def calculateConfusionMatrice(image, name):
    ImageSeuil_1D = image.flatten()
    GT_1D = (openImg(name+'_GT')).flatten()
    cm = confusion_matrix(GT_1D, ImageSeuil_1D)
    return cm

def question1():
    name = "2classes_100_100_8bits"

    image = openImg(name)
    imageFlat = image.flatten()

    # l'histogramme des niveaux de gris
    h = [np.count_nonzero(imageFlat==i) for i in range(256)]

    # le nombre de pixels de l'image
    N = len(imageFlat)

    # D termination du seuil par la m thode d'Otsu
    dispersionMax = 0
    tMax = 0
    for t in range(255):

```

```
        # calcul des P(w1(t)) et N(w1(t))
        pw1 = 0
        nw1 = 0
        for i in range(t+1):
            pw1 += h[i] / N
            nw1 += h[i]
        # d duction des P(w2(t)) et N(w2(t))
        pw2 = 1-pw1
        nw2 = N-nw1

        # On ne retient pas les cas o  une classe est vide
        # Cela reviendrait  traiter une classe et non deux et cela n'a pas de sens
        if(nw1!=0 and nw2!=0):
            #calcul 1
            uw1=0
            for i in range(t+1):
                uw1+=(i*h[i])
            uw1 = uw1/nw1
            #calcul 2
            uw2=0
            for i in range(t+1,256):
                uw2+=(i*h[i])
            uw2 = uw2/nw2
            #calcul de la dispersion et mis  jour (maximisation) des valeurs courantes
            dispersion = pw1*pw2*((uw1-uw2)**2)
            if(dispersion>dispersionMax):
                dispersionMax=dispersion
                tMax=t

    print("Dispersion maximale : "+str(dispersionMax))
    print("Valeur de seuil optimale : "+str(tMax))

    imageBin = seuillage(image, tMax)

    confusionMatrice = calculateConfusionMatrice(imageBin, name)
    print("taux de bonne classification des pixels de : "+str((confusionMatrice[0,0]+confusionMatrice[1,1])/(

    showGreyImage(imageBin)



question1()
```

B Macro Exercice 2

```
#Suliac Lavenant et Antoine Nollet

import numpy as np
import cv2
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

#ouvre une image en niveau de gris et v rifie qu'elle est bien de dimension 2
def openImg(name):
    image = plt.imread( 'images/' + name + '.png' )
    if image.ndim > 2:
        image = image[:,:,0]

    image[:,:]=(image[:,:]*255).astype(np.uint8)

    return image

#affiche une image en niveau de gris
def showGreyImage(image):
    plt.imshow(image, cmap='gray' )
    plt.show()
```

```
22
23 #affichage de l'histogramme de image
24 def showHistogrameOf(image):
25     nbins=255
26     counttsg, edgesg = np.histogram(image, bins=nbins, range=(0,255))
27     plt.hist(edgesg[:-1], nbins, weights=counttsg)
28     plt.show()
29
30 #seuille l'image image selon le seuil seuil
31 def seuillage(image, seuil):
32     return np.where(image > seuil, 255, 0)
33
34 #seuille l'image image selon le seuil seuil
35 def seuillageInv(image, seuil):
36     return np.where(image > seuil, 0, 255)
37
38 #seuille l'image image selon les seuils seuil1 et seuil2
39 def doubleSeuillage(image, seuil1, seuil2):
40     imageSeuil1 = np.where(image > seuil1, 127, 0)
41     imageSeuil2 = np.where(image > seuil2, 255, 0)
42     return np.maximum(imageSeuil1,imageSeuil2)
43
44 #calcule la matrice de confusion de l'image image
45 def calculateConfusionMatrice(image, name):
46     ImageSeuil_1D = image.flatten()
47     GT_1D = (openImg(name+'_GT')).flatten()
48     cm = confusion_matrix(GT_1D, ImageSeuil_1D)
49     return cm
50
51 def question2():
52     name = "3classes_100_156_8bits"
53
54     image = openImg(name)
55     imageFlat = image.flatten()
56
57     # l'histogramme des niveaux de gris
58     h = [np.count_nonzero(imageFlat==i) for i in range(256)]
59
60     # le nombre de pixels de l'image
61     N = len(imageFlat)
62
63     # D termination du seuil par la m thode d'Otsu
64     dispersionMin = float("inf")
65     t1Min = 0
66     t2Min = 0
67     for t1 in range(255):
68         for t2 in range(t1+1,256):
69
70             # calcul de P(w1(t)) et de N(w1(t))
71             pw1 = 0
72             nw1 = 0
73             for i in range(t1+1):
74                 pw1 += h[i] / N
75                 nw1 += h[i]
76
77             # calcul de P(w2(t)) et de N(w2(t))
78             pw2 = 0
79             nw2 = 0
80             for i in range(t1+1,t2+1):
81                 pw2 += h[i] / N
82                 nw2 += h[i]
83
84             # calcul de P(w3(t)) et de N(w3(t))
85             pw3 = 0
86             nw3 = 0
87             for i in range(t2+1,256):
88                 pw3 += h[i] / N
89                 nw3 += h[i]
90
91             # on souhaite 3 classes, donc aucune doivent  tre  vide
92             if(nw1!=0 and nw2!=0 and nw3!=0):
```

```
                #calcul  1
                uw1=0
                for i in range(t1+1):
                    uw1+=(i*h[i])/nw1

                #calcul  3
                uw2=0
                for i in range(t1+1,t2+1):
                    uw2+=(i*h[i])/nw2

                #calcul  3
                uw3=0
                for i in range(t2+1,256):
                    uw3+=(i*h[i])/nw3

                # calcul variance de la classe w1
                somme1=0
                for i in range(t1+1):
                    somme1+=((((i-uw1)**2)*h[i])/nw1)
                # calcul variance de la classe w2
                somme2=0
                for i in range(t1+1,t2+1):
                    somme2+=((((i-uw2)**2)*h[i])/nw2)
                # calcul variance de la classe w3
                somme3=0
                for i in range(t2+1,256):
                    somme3+=((((i-uw3)**2)*h[i])/nw3)

                # calcul variance de l'image enti re
                dispersion=pw1*somme1+pw2*somme2+pw3*somme3

                # mis    jour des valeurs courantes, minimisation de la variance
                if dispersion < dispersionMin:
                    t1Min = t1
                    t2Min = t2
                    dispersionMin = dispersion



    print("Dispersion minimale : "+str(dispersionMin))
    print("Valeur de seuil optimale : "+str(t1Min)+" et "+str(t2Min))

    imageBin = doubleSeuillage(image, t1Min, t2Min)

    confusionMatrice = calculateConfusionMatrice(imageBin, name)
    print("taux de bonne classification des pixels de : "+str((confusionMatrice[0,0]+confusionMatrice[1,1])/(

    showGreyImage(imageBin)



question2()
```

C Macro Exercice 3

```
#Suliac Lavenant et Antoine Nollet

import numpy as np
import cv2
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

#ouvre une image en niveau de gris et v rifie qu'elle est bien de dimension 2
def openImg(name):
    image = plt.imread( 'images/' + name + '.png' )
```

```
    if image.ndim > 2:
        image = image[:,:,0]

    image[:,:]=(image[:,:]*255).astype(np.uint8)

    return image

#affiche une image en niveau de gris
def showGreyImage(image):
    plt.imshow(image, cmap='gray' )
    plt.show()

#affiche une image
def showImage(image):
    plt.imshow(image)
    plt.show()

#affichage de l'histogramme de image
def showHistogrameOf(image):
    nbins=255
    counttsg, edgesg = np.histogram(image, bins=nbins, range=(0,255))
    plt.hist(edgesg[:-1], nbins, weights=counttsg)
    plt.show()

#seuille l'image image selon le seuil seuil
def seuillage(image, seuil):
    return np.where(image > seuil, 255, 0)

#seuille l'image image selon le seuil seuil
def seuillageInv(image, seuil):
    return np.where(image > seuil, 0, 255)

#seuille l'image image selon les seuils seuil1 et seuil2
def doubleSeuillage(image, seuil1, seuil2):
    imageSeuil1 = np.where(image > seuil1, 127, 0)
    imageSeuil2 = np.where(image > seuil2, 255, 0)
    return np.maximum(imageSeuil1,imageSeuil2)

#calcule la matrice de confusion de l'image image
def calculateConfusionMatrice(image, name):
    ImageSeuil_1D = image.flatten()
    GT_1D = (openImg(name+'_GT')).flatten()
    cm = confusion_matrix(GT_1D, ImageSeuil_1D)
    return cm

def ostsu2(image):
    imageFlat = image.flatten()

    # l'histogramme des niveaux de gris
    h = [np.count_nonzero(imageFlat==i) for i in range(256)]

    N = len(imageFlat)

    dispersionMax = 0
    tMax = 0
    for t in range(255):

        # calcul des P(w1(t)) et N(w1(t))
        pw1 = 0
        nw1 = 0
        for i in range(t+1):
            pw1 += h[i] / N
            nw1 += h[i]
        # d duction des P(w2(t)) et N(w2(t))
        pw2 = 1-pw1
        nw2 = N-nw1

        # On ne retient pas les cas o  une classe est vide
        # Cela reviendrait  traiter une classe et non deux et cela n'a pas de sens
        if(nw1!=0 and nw2!=0):
            #calcul  1
```

```
            uw1=0
            for i in range(t+1):
                uw1+=(i*h[i])
            uw1 = uw1/nw1
            #calcul  2
            uw2=0
            for i in range(t+1,256):
                uw2+=(i*h[i])
            uw2 = uw2/nw2
            #calcul de la dispersion et mis    jour (maximisation) des valeurs courantes
            dispersion = pw1*pw2*((uw1-uw2)**2)
            if(dispersion>dispersionMax):
                dispersionMax=dispersion
                tMax=t

    print("Dispersion maximale : "+str(dispersionMax))
    print("Valeur de seuil optimale : "+str(tMax))

    imageBin = seuillage(image, tMax)

    return imageBin

def question3():
    name = "IMAGE3D"

    # lecture de l'image principale
    image = plt.imread( 'images/' + name + '.TIF' )
    #showImage(image)

    # s paration par les composantes RGB
    r=image[:,:,0]
    #showGreyImage(r)
    #showHistogrameOf(r)
    g=image[:,:,1]
    #showGreyImage(g)
    #showHistogrameOf(g)
    b=image[:,:,2]
    #showGreyImage(b)
    #showHistogrameOf(b)

    # applatissement des composantes
    rFlat = r.flatten()
    gFlat = g.flatten()
    bFlat = b.flatten()

    # l'histogramme des niveaux de gris de chaque composante
    rh = [np.count_nonzero(rFlat==i) for i in range(256)]
    gh = [np.count_nonzero(gFlat==i) for i in range(256)]
    bh = [np.count_nonzero(bFlat==i) for i in range(256)]

    # Binarisation sous Ostsu de chaque composante
    rBin=ostsu2(r)
    #showGreyImage(rBin)
    gBin=ostsu2(g)
    #showGreyImage(gBin)
    bBin=ostsu2(b)
    #showGreyImage(bBin)

    #Fusion des composantes pour reformer une image couleur
    finalBin=np.zeros((image.shape), dtype=np.uint8)
    finalBin[:,:,0]=rBin[:,:]
    finalBin[:,:,1]=gBin[:,:]
    finalBin[:,:,2]=bBin[:,:]

    showImage(finalBin)
    #plt.imsave("finalBin.png", finalBin)


question3()
```