AIV1 - TP11
Analyse en composantes principales pour la segmentation d'images couleur

Suliac Lavenant et Antoine Nollet

1st April 2022

Table des matières

1 Analyse en composantes principales

Dans ce TP, il sera question d'utiliser l'apprentissage non-supervisé via l'Analyse en Composantes Principales (ACP) afin de segmenter une image en diverses classes. En guise d'illustration, nous utiliserons dans cette section l'image suivante pour nos manipulations :

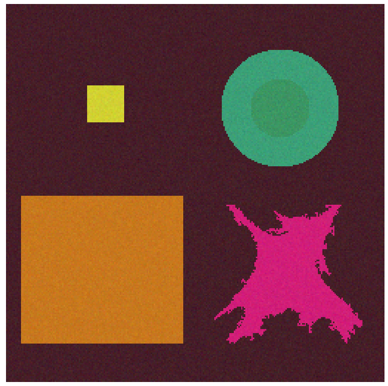

Figure 1 – Image Couleur de Base

Vous trouverez par la suite, en annexe de ce compte-rendu, l'implémentation de l'ACP utilisé dans le cadre du TP.

1.1 Transformation en matrice de données

Pour notre image couleur, ses composantes principales seront ses composantes de couleurs, c'est à dire ses canaux RGB (rouge-vert-bleue). Il y a donc 3 vecteurs d'attributs, un pour chaque composante couleur.

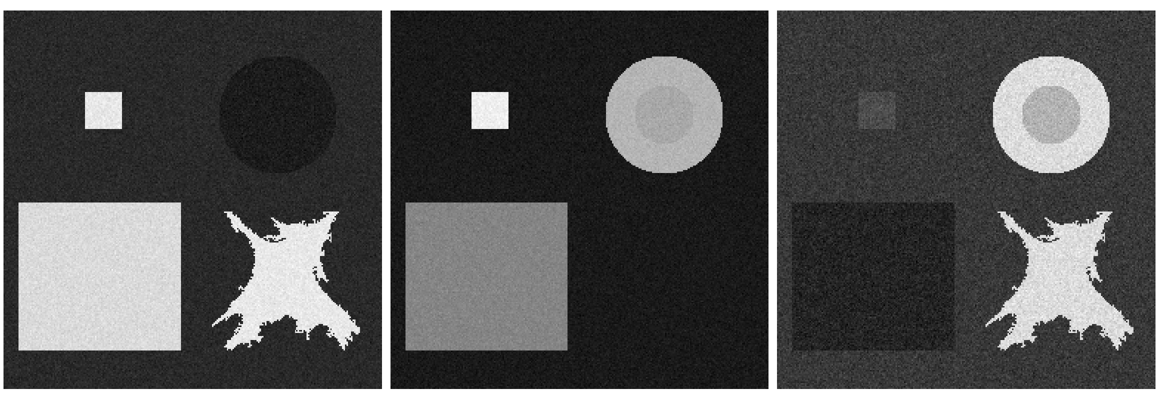

Figure 2 – Images des composantes RGB de l'image de base

La matrice de données sera donc une matrice de taille NxD, où N correspond au nombre de pixels de l'image de base et où D est le nombre de vecteurs d'attributs (ici 3 : les vecteurs R, G et B). La matrice sera donc de cette forme :

$$\begin{bmatrix} \mathrm{R}_1 & \mathrm{R}_2 & \mathrm{R}_3 & ... & \mathrm{R}_N \\ \mathrm{G}_1 & \mathrm{G}_2 & \mathrm{G}_3 & ... & \mathrm{G}_N \\ \mathrm{B}_1 & \mathrm{B}_2 & \mathrm{B}_3 & ... & \mathrm{B}_N \end{bmatrix}$$

où chaque valeur C_i correspond à la valeur de la composante couleur C du ième pixel.

1.2 Fonction ACP

Une fois notre image de base transformée en sa matrice de donnée, que nous nommerons X, nous pouvons la projeter par Analyse de ses Composantes Principales (ACP).

L'ACP consistera en ces diverses étapes :
- Calcul de la matrice de co-variance des données de X
- Recherche des vecteurs propres avec les valeurs propres les plus élevées
- Matrice Solution W à partir des vecteurs propres
- Projection des données

1.2.1 Matrice de Co-Variance

Pour déterminer la matrice de co-variance, il faut d'abord déterminer les moyennes des $D = 3$ attributs sur nos N pixels. Soit M définit comme ceci :

$$M = (\mu_R, \mu_G, \mu_B)$$

avec $\mu_i = \frac{1}{N}\sum_{j=1}^{N} X_{i,j}$
et où $X_{i,j}$ correspond à la valeur de la composante couleur i au j ième pixel. Par exemple, $X_{R,1}$ correspondra à la valeur R_1 de notre matrice de données

Ensuite, il sera possible de déterminer la co-variance Σ entre 2 attributs i et l de la manière suivante :

$$\Sigma_{i,l} = \frac{1}{N}\sum_{j=1}^{N}(X_{i,j} - \mu_i).(X_{l,j} - \mu_l)$$

Si cette valeur $\Sigma_{i,l}$ est positive, alors les deux attributs augmentent et diminuent en même temps. Si elle est négative, alors l'un augmente pendant que l'autre diminue. Enfin si elle est nulle alors les attributs sont indépendants.

Ainsi, dans notre cas, nous obtiendrons la matrice Σ de co-variance suivante :

$$\begin{bmatrix} \Sigma_{R,R} & \Sigma_{R,G} & \Sigma_{R,B} \\ \Sigma_{G,R} & \Sigma_{G,G} & \Sigma_{G,B} \\ \Sigma_{B,R} & \Sigma_{B,G} & \Sigma_{B,B} \end{bmatrix}$$

À noter que cette matrice de co-variance est symétrique, en effet : $\Sigma_{i,l} = \Sigma_{l,i}$.

1.2.2 Recherche Vecteurs Propres

Une fois la matrice Σ de co-variance obtenue, on va retenir les d vecteurs propres avec les valeurs propres les plus élevées. Comme il n'y a que 3 attributs et qu'ils sont naturellement indépendants (les couleurs ne sont pas dépendantes les unes des autres), on conservera les 3 vecteurs propres triés du vecteur à la valeur propre la plus grande au vecteur à la valeur propre la plus faible.

1.2.3 Matrice Solution

On considérera donc la matrice solution W qui sera égal à (w_1, w_2, w_3) où les w_i sont les vecteurs propres de la matrice et où w_1 est le vecteur propre à la valeur propre la plus grande.

1.2.4 Projection des données

Une fois la matrice solution W obtenue, nous pouvons déterminer la matrice Y de projection ACP de notre matrice de données X par la manière suivante :

$$Y = W^T.X$$

Comme nous conservons $d = D = 3$ et que donc la matrice W est de dimensions 3x3, alors la matrice Y de projection ACP sera des mêmes dimensions que la matrice de données X.

1.3 Transformation de la matrice projetée vers 3 canaux

Une fois la matrice Y des données projeté en ACP obtenue, comme elle est de même dimension que la matrice X des données initiales, nous pouvons traiter les canaux ACP de l'image de base comme étant des canaux RGB particuliers :

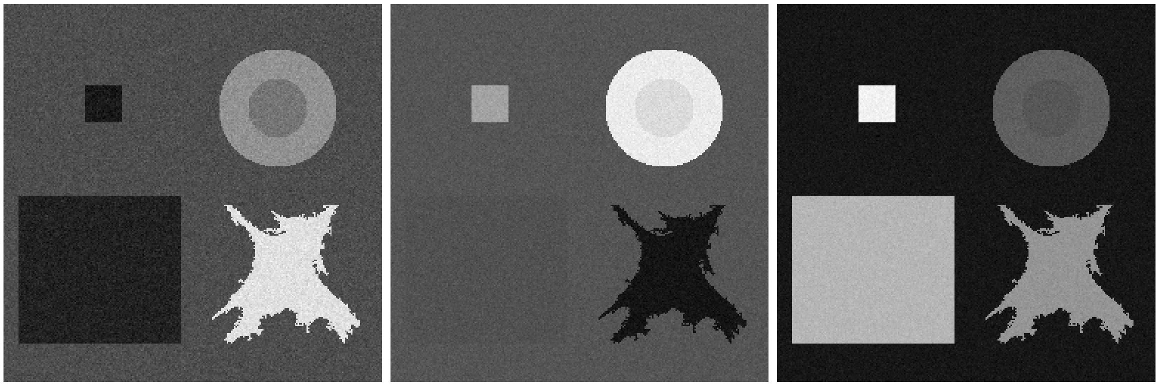

Figure 3 – les trois canaux ACP

1.4 Binarisation des 3 canaux

Une fois ces trois nouveaux canaux RGB obtenus (correspondants aux canaux ACP), nous pouvons les remettre en 2 dimensions pour qu'ils soient de nouveau de la taille de l'image d'origine (comme vu précédemment) avant d'appliquer sur chacun d'entre eux la binarisation par la méthode d'Otsu (vu dans le précédent TP).

Les trois canaux ainsi binarisés sont les suivants :

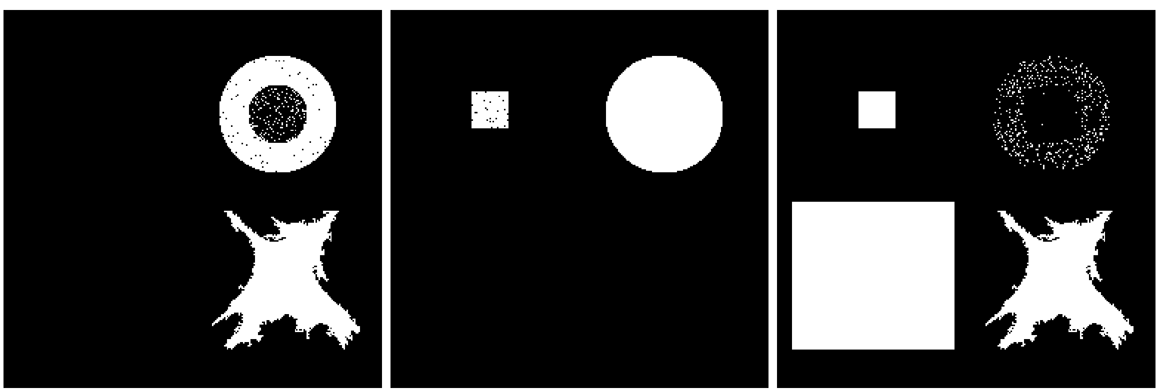

Figure 4 – les trois canaux segmentés (3eme, 2eme et 1er)

1.5 Fusion des 3 canaux

On peut ainsi fusionner ces trois canaux binarisés en une image couleur RGB (le 3eme canal sert de rouge, le 2eme sert de vert et le 1er sert de bleue) :

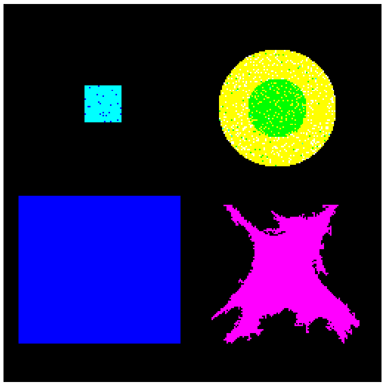

FIGURE 5 – image composé des trois canaux

Sur cette image, on peut voir nos 5 formes bien segmentées. Nous pouvons même appuyer le fait que les deux disques concentriques soient ici bien segmentés : ce n'était pas le cas lors de notre précédent TP qui utilisait la binarisation par méthode d'Otsu sur les canaux RGB de l'image de base. Notons d'ailleurs que nous avons ici 6 classes de pixels différentes (le fond en noir, la tâche en magenta, le grand carré en bleu, le petit carré en cyan, le disque central en vert et le disque externe en jaune).

1.6 Utilisation uniquement de 2 canaux

Il peut être pertinent de se poser la question de quels canaux ACP il est réellement utile d'utiliser. Bien que cela nous fasse sortir du cadre de l'apprentissage non-supervisé et entrer dans l'apprentissage supervisé (car ici on exprime empiriquement ce qu'on préfére obtenir), nous pouvons obtenir une segmentation de l'image en un nombre moindre de classes de pixels différentes.

On constate à partir des 3 canaux segmentées que le 1er canal est nécessaire afin de conserver la forme du grand carré qui est non présente dans les 2 autres. Ensuite, le 1er canal contient beaucoup d'informations sur les différentes formes, à l'exception de la bonne segmentation entre les deux disques concentriques. On choisira donc d'utiliser également le 3eme canal qui appuiera la bonne segmentation des diques (les disques ne sont pas séparés dans le 2eme canal) et qui ne faussera pas la segmentation du 1er canal (le 2eme canal rajoute du bruit dans le petit carré).

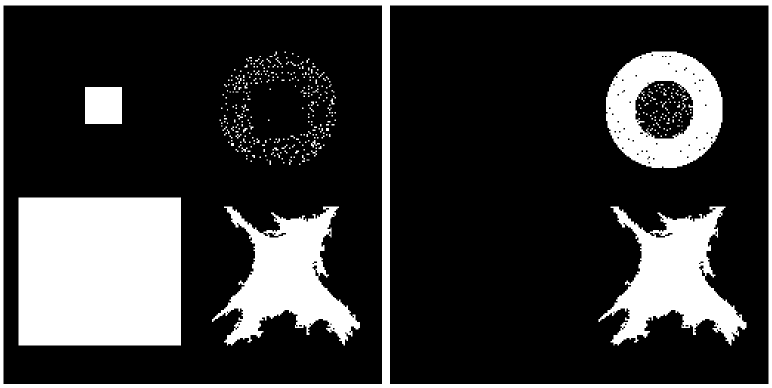

FIGURE 6 – les composantes segmentées des 1er et 3eme canaux

On peut voir que ces deux canaux suffisent pour segmenter toute les formes de l'image et on va donc essayer de recomposer l'image avec uniquement ces deux composantes. On met le 3eme canal dans la composante rouge et puis le 1er canal dans les composantes verte et bleue.

On obtient ainsi le résultat suivant :

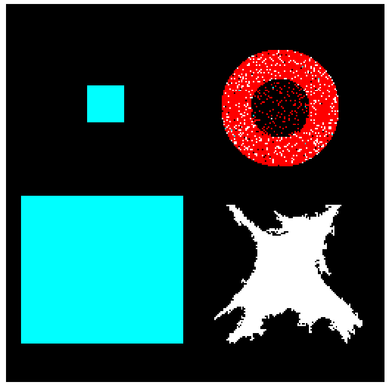

Figure 7 – image composée des 1er et 3eme canaux

Cette image, bien qu'utilisant que deux canaux, suffit à segmenter l'image. On retrouve bien encore une fois toute les formes et même les deux disques concentriques sont bien segmentés. Tout cela avec 4 classes différentes de pixels au lieu de 6 : les 2 carrés en cyan, la tâche en blanc, le disque externe en rouge et le disque interne et le fond en noir. (Il n'est ici pas gênant que le disque interne et le fond soient de la même couleur, l'important est que le disque externe soit séparé du fond et que le disuqe interne soit séparé du disque externe)

2 Application à 3 images

On veut maintenant appliquer ce même principe (de choisir quel(s) canal(canaux) ACP il est plus pertinent d'utiliser) aux images pour daltonien qui suivent afin de récupérer les nombres des images.

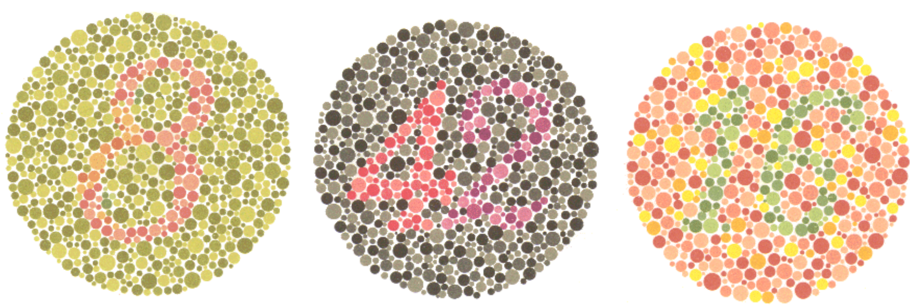

Figure 8 – les trois images de daltoniens traité

2.1 cas_4_dalton8

Voici notre première image :

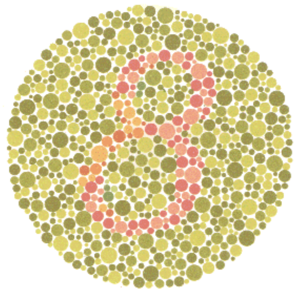

Figure 9 – image cas_4_dalton8

On a ensuite les trois canaux binarisés extraits de ses composantes ACP :

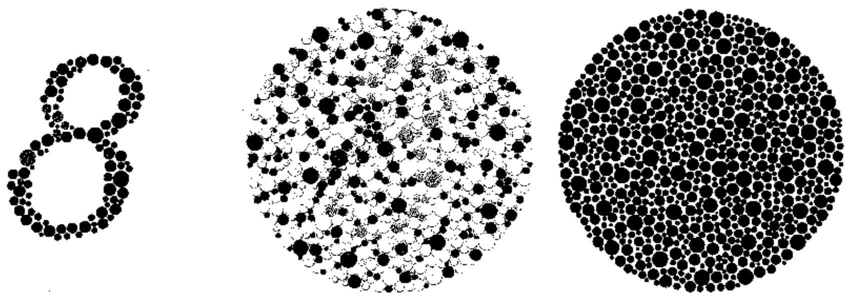

Figure 10 – les trois canaux segmentés (3eme, 2eme et 1er)

Une fusion naïve de ces composantes (on les considère toutes) donne ce résultat :

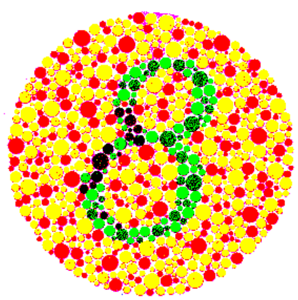

Figure 11 – image composée des trois canaux

Le résultat obtenu n'est pas celui voulu. En observant les composantes, on remarque que la 3eme contient juste le 8 et que les autres contiennent que du "bruit".

On va donc utiliser uniquement ce 3eme canal :

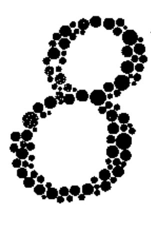

Figure 12 – image composée uniquement du 3eme canal

2.2 cas_1_dalton42

Voici notre seconde image :

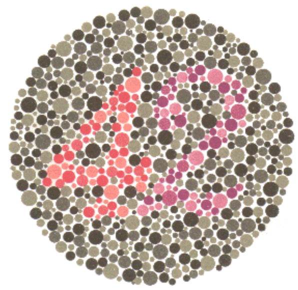

Figure 13 – image cas_1_dalton42

On a ensuite les trois canaux binarisés extraits de ses composantes ACP :

Figure 14 – les trois canaux segmentés (3eme, 2eme et 1er)

Une fusion naïve de ces composantes (on les considère toutes) donne ce résultat :

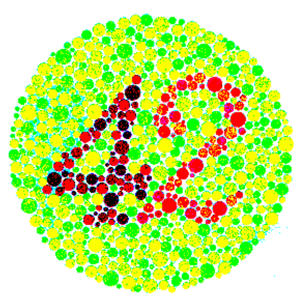

Figure 15 – image composée des trois canaux

Le résultat obtenu n'est pas celui voulu. En observant les composantes, on remarque que la 1eme contient juste le 42 et que les autres contiennent que du "bruit" (la 3eme contient toutefois le 2 du 42, mais l'utiliser rajouterait du bruit inutile car on peut utiliser le 2e canal).

On va donc utiliser uniquement ce 2eme canal :

Figure 16 – image composée uniquement du 2eme canal

2.3 cas_2_dalton16

Voici notre dernière image :

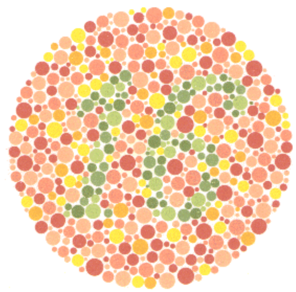

FIGURE 17 – image cas_2_dalton16

On a ensuite les trois canaux binarisés extraits de ses composantes ACP :

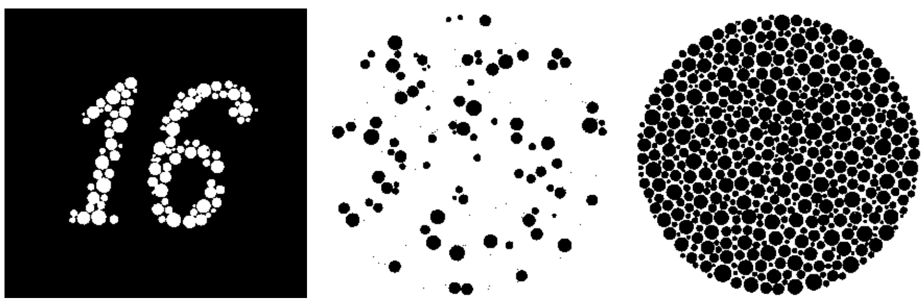

FIGURE 18 – les trois canaux segmentés (3eme, 2eme et 1er)

Une fusion naïve de ces composantes (on les considère toutes) donne ce résultat :

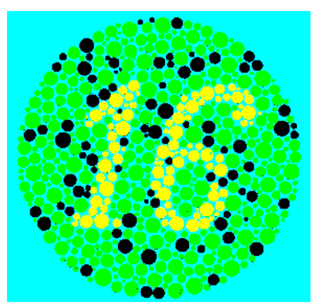

FIGURE 19 – image composée des trois canaux

Le résultat obtenu n'est pas celui voulu. En observant les composantes, on remarque que la 3eme contient juste le 16 et que les autres contiennent que du "bruit".
On va donc utiliser uniquement ce 3eme canal :

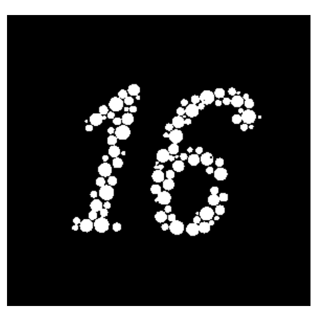

FIGURE 20 – image composée uniquement du 3eme canal

A Annexe - Implémentation de l'Analyse en Composantes Principales

```python
#Suliac Lavenant et Antoine Nollet

# les imports n cessaires
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
```

```python
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


# fonction de la sous section "Transformation en matrice de donn es"
def matrice_donnes(image):
    # on r cup re les composantes couleur de l'image
    r = image[:,:,0]
    g = image[:,:,1]
    b = image[:,:,2]

    # on s'en sert pour d terminer les vecteurs d'attributs RGB, on les utilise en 1 dimension chacun
    rFlat = r.flatten()
    gFlat = g.flatten()
    bFlat = b.flatten()

    # on construit notre matrice de donn es
    x = np.zeros((len(image)*len(image[0]),3), dtype=int)
    x[:,0] = rFlat[:]
    x[:,1] = gFlat[:]
    x[:,2] = bFlat[:]

    return x
```

```python
# Projection en ACP
def analyseEnComposantesPrincipales(x):
    # vecteur M des moyennes de valeurs
    x_meaned = x - np.mean(x, axis=0)
    # calcul de la matrice de co-variance (dimension 3x3)
    cov_mat = np.cov(x_meaned, rowvar=False)

    # valeurs propres et vecteurs propres de la matrice de co-variance
    eigen_values, eigen_vectors = np.linalg.eigh(cov_mat)

    # tri des indices selons les valeurs propres les plus grandes
    sorted_index = np.argsort(eigen_values)[::-1]
    # on trie les valeurs propres selon les indices
    sorted_eigenvalue = eigen_values[sorted_index]
    # on trie les vecteurs propres selon les indices
    sorted_eigenvectors = eigen_vectors[:,sorted_index]

    ####################################################################################
    #sorted_eigenvectors[:,0] = sorted_eigenvectors[:,0] * -1 #bidoullage si python < 3.9
    ####################################################################################

    # calcul de la projection par ACP des donn es gr ce   la matrice W des vecteurs propres
    y_projected = np.dot(sorted_eigenvectors.transpose(), x_meaned.transpose()).transpose()

    return y_projected

# On d termine les 3 canaux grace a la matrice des donn es projet e par ACP (qui est de m me dimension que
def projectedMatriceOn3Canal(xPCA):
    #r cuperation du min et du max pour normalisation
    min = xPCA.min()
    max = xPCA.max()

    #normalisation des valeurs entre 0 et 255
    xPCA = ((xPCA-min)/(max-min))*255

    #enleve les chiffres apr s la virgule
    xPCA = np.trunc(xPCA)
    #change les valeur de float vers int
    xPCA=xPCA.astype('uint8')

    # d finition des diff rents canaux ACP
    c0PCA=np.reshape(xPCA[:,0], (len(xPCA),1))
    c1PCA=np.reshape(xPCA[:,1], (len(xPCA),1))
    c2PCA=np.reshape(xPCA[:,2], (len(xPCA),1))

    return c0PCA, c1PCA, c2PCA


def otsu(c0, c1, c2):

    # Binarisation sous Ostsu de chaque composante
    c0Bin=ostsu2(c0)
    #showGreyImage(c0Bin)
    #plt.imsave("c0Bin.png", c0Bin, cmap='gray')
    c1Bin=ostsu2(c1)
    #showGreyImage(c1Bin)
    #plt.imsave("c1Bin.png", c1Bin, cmap='gray')
    c2Bin=ostsu2(c2)
    #showGreyImage(c2Bin)
    #plt.imsave("c2Bin.png", c2Bin, cmap='gray')

    #Fusion des composantes pour reformer une image couleur
    finalBin=np.zeros((len(c0), len(c0[0]), 3), dtype=np.uint8)
    finalBin[:,:,0]=c2Bin[:,:]
    finalBin[:,:,1]=c1Bin[:,:]
    finalBin[:,:,2]=c0Bin[:,:]

    showImage(finalBin)
    #plt.imsave("finalBinc200.png", finalBin)
```

```python
def question1():
    #ouverture de l'image
    nameAndFormat = "IMAGE3D.TIF"
    #nameAndFormat = "cas_4_dalton8.bmp"
    #nameAndFormat = "cas_1_dalton42.bmp"
    #nameAndFormat = "cas_2_dalton16.bmp"
    image = plt.imread( 'images/' + nameAndFormat)

    # R cup ration matrice de donn es
    x = matrice_donnes(image)
    # Projection des donn es par ACP
    pca = analyseEnComposantesPrincipales(x)
    # D termination des 3 diff rents canaux ACP
    pcaN = projectedMatriceOn3Canal(pca)

    # Remise en deux dimensions des differents canaux
    c0PCA = np.reshape(pcaN[0], (len(image),len(image[0])))
    c1PCA = np.reshape(pcaN[1], (len(image),len(image[0])))
    c2PCA = np.reshape(pcaN[2], (len(image),len(image[0])))

    # Binarisation par m thode d'Otsu et Fusion des composantes binaris es
    otsu(c0PCA, c1PCA, c2PCA)


question1()
```