

AIV1 - TP12

# Segmentation d'une image couleur par classification non supervisée de pixels

Suliac Lavenant et Antoine Nollet

8th April 2022

# Table des matières

<b>1</b>	<b>Contexte</b>	<b>3</b>
<b>2</b>	<b>SEGMENTATION DES IMAGES par K-Means en RGB</b>	<b>3</b>
2.1	K-Means . . . . .	3
2.2	Cas du 73 . . . . .	4
2.3	Réutilisation de la classification pour d'autres cas . . . . .	5
<b>3</b>	<b>SEGMENTATION DE L'IMAGE 'cas_1_dalton42.bmp' par K-Means en RGB</b>	<b>6</b>
3.1	Cas du 42 . . . . .	6
3.2	Réutilisation de la classification pour d'autres cas . . . . .	7
<b>4</b>	<b>K-MEANS et Analyse en Composantes Principales</b>	<b>7</b>
<b>A</b>	<b>Annexe - Macro Question 1</b>	<b>10</b>
<b>B</b>	<b>Annexe - Macro Question 3</b>	<b>14</b>

# 1 Contexte

Dans le cadre du TP précédent, nous avons déterminé qu’une approche par Analyse en Composante Principales (ACP) amenait à des résultats plus probants (de meilleures segmentations) qu’avec une approche des couleurs RGB. Nous classifions les images par une segmentation par la méthode d’Otsu... Dans le cadre de ce TP, il sera question d’utiliser une nouvelle méthode de classification, le Clustering par K-Means, et de constater si les conclusions du précédent TP sont toujours vérifiées ou non. Le choix de la méthode de classification pourrait influencer nos résultats.. ?

## 2 SEGMENTATION DES IMAGES par K-Means en RGB

Dans le but de comparer de nouveau l’approche ACP et RGB mais avec une méthode de classification différente qu’est le clustering par K-Means, nous allons donc utiliser cette méthode avec une approche RGB d’une image.

### 2.1 K-Means

Il faut tout d’abord clarifier ce qu’est le Clustering par K-Means : en parallèle à la segmentation par méthode d’Otsu, il s’agit également ici d’une classification par apprentissage non-supervisé. Leur différence réside en la définition d’un nombre K de classes à déterminer, nombre K qui sera défini par l’utilisateur. Une fois ce nombre défini, le principe du K-Means sera de déterminer les k centres de gravité  $\hat{\mu}_k$  dans l’espace d’attributs. Ces  $\hat{\mu}_k$  centres de gravité dans l’espace d’attributs permettront d’obtenir les k classes. En effet, chaque point dans l’espace d’attributs sera plus proche d’un centre de gravité que des autres, ce point (correspondant à un pixel de l’image) appartiendra donc à la classe défini par le centre de gravité.

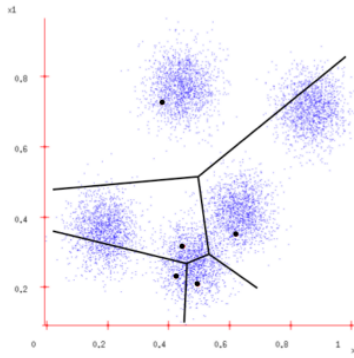


FIGURE 1 – Exemple d’assignation de centres de gravité

Le but du K-means sera de déterminer ces  $\hat{\mu}_k$  centres de gravité de telle sorte à minimiser la distance des points de l’espace d’attributs et de leurs centres de gravité. Ainsi le K-Means minimisera la distance totale D suivante :

$$D = \frac{1}{N} \sum_x (x - \mu_{\hat{\omega}(x)})^T \cdot (x - \mu_{\hat{\omega}(x)})$$

où N est le nombre total de données (les pixels de l’image), x est une donnée (un pixel de l’image) et  $\mu_{\hat{\omega}(x)}$  est le centre de gravité le proche correspondant à la donnée x. X appartiendra donc à la classe  $\hat{\omega}(x)$ .

## 2.2 Cas du 73

Voici l'image que nous utilisons ici :

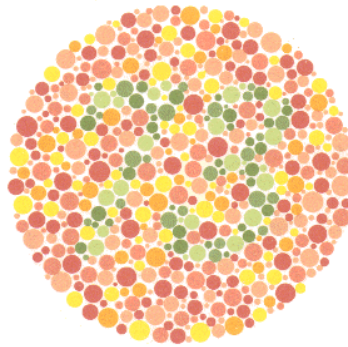


FIGURE 2 – Image Couleur de Base

En utilisant la méthode de classification du clustering par K-Means (voir l'implémentation en Annexe - Macro Question 1) et variant les valeurs de  $k$ , voici les résultats obtenus :

K utilisé	4	6	7
résultat segmentation			

K utilisé	8	9	10
résultat segmentation			

En vue des résultats, on voit un début d'apparition du nombre 73 à partir du moment où on cherche 6 classes (apparition du vert foncé). Cependant, on préférera donner la valeur de 8 au nombre  $k$  car l'apparition du vert clair permet une meilleure reconnaissance du nombre 73.

## 2.3 Réutilisation de la classification pour d'autres cas

Nous avons maintenant déterminé les centres de gravité par le clustering par K-Means, et donc effectué une classification. Essayons d'utiliser les mêmes centres de gravité mais pour d'autres images du même type que notre image de base du 73. Voici donc les deux images ressemblants à notre image de base :

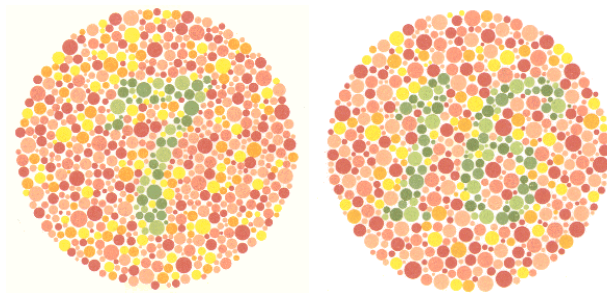


FIGURE 3 – Images du 7 et du 16 de base

Ainsi, en reprenant la même classification avec les mêmes centres de gravité que précédemment avec le 73, nous obtenons les segmentations suivantes :

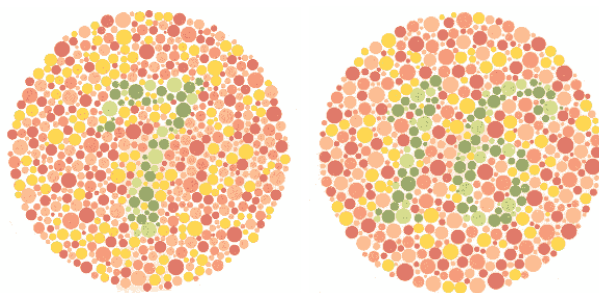


FIGURE 4 – Images du 7 et du 16 segmentées

On constate donc que la classification par Clustering par K-Means mène à une segmentation, bien que minime, qui fait ressortir les nombres dans les images. Mais est-ce que cette méthode de classification fonctionne avec d'autres images ?

### 3 SEGMENTATION DE L'IMAGE 'cas\_1\_dalton42.bmp' par K-Means en RGB

Nous allons donc réutiliser la classification par clustering par K-Means afin de segmenter une nouvelle image RGB.

#### 3.1 Cas du 42

Voici donc l'image que nous utilisons ici :

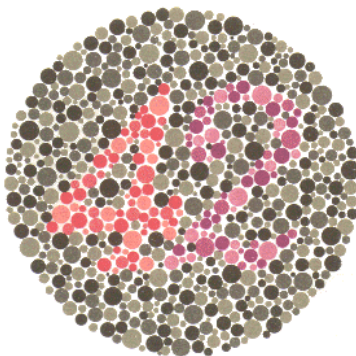
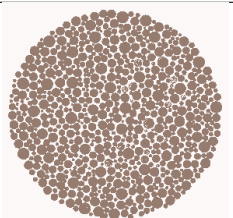
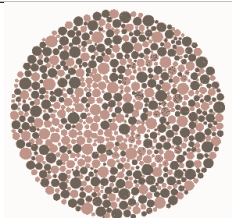
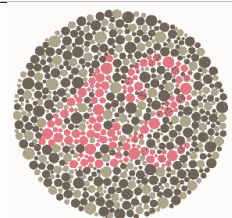
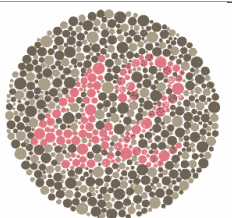
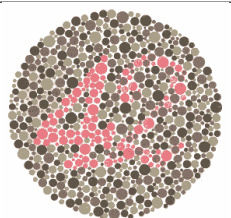


FIGURE 5 – Image Couleur de Base

Comme précédemment, on utilise la méthode de classification du clustering par K-Means (voir l'implémentation en Annexe - Macro Question 1) et variant les valeurs de  $k$ , voici les résultats obtenus :

K utilisé	2	3	4
résultat segmentation			

K utilisé	5	6
résultat segmentation		

Comme précédemment, on détermine le  $k$  qui nous intéresse le plus selon la situation (apprentissage supervisé), et on choisira un  $k$  égal à 4. En dessous de 4, le nombre n'est pas dissocié du fond. Au dessus de 4, la segmentation du 42 perd de la précision souhaitée. Un  $k$  égal à 4 permettra une segmentation idéal pour notre cas.

### 3.2 Réutilisation de la classification pour d'autres cas

Réutilisons notre classification pour une image ressemblante à la précédente. La voici :

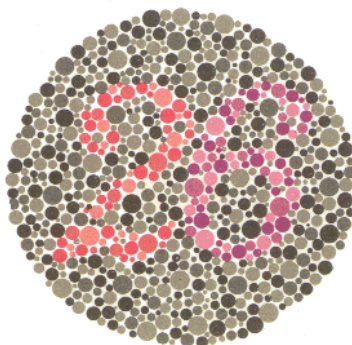


FIGURE 6 – Image du 26 de Base

Et voici le résultat obtenu avec la même classification qu'avec l'image du 42 :

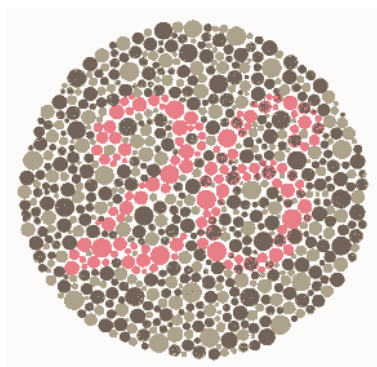


FIGURE 7 – Image du 26 segmentée

Nous obtenons là également une segmentation adéquate du nombre dans l'image. Utilisons donc cette méthode de classification avec une approche en analyse des composantes principales des images...

## 4 K-MEANS et Analyse en Composantes Principales

Nous allons donc pouvoir utiliser une classification par Clustering par K-Means sur une approche ACP des images afin de comparer les résultats obtenus avec une approche RGB.

Nous allons donc réutiliser notre image du 42 :

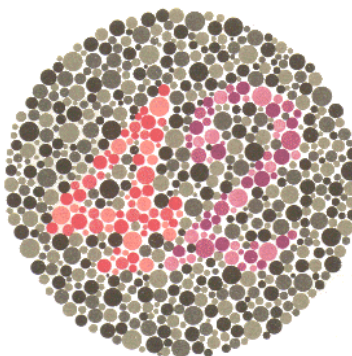


FIGURE 8 – Image Couleur de Base du 42



Voici sa conversion en image ACP (voir Annexe - Macro Question 3 pour plus de détails de l'implémentation utilisée) :

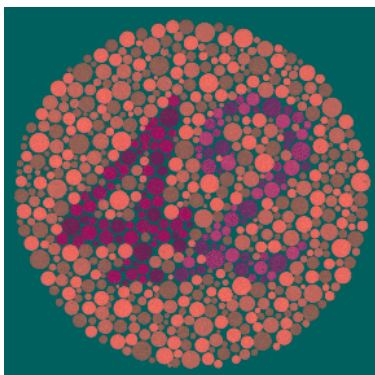


FIGURE 9 – Image ACP de Base du 42

Nous avons utilisé un  $k=4$  pour notre précédent K-Means, nous faisons donc de même pour obtenir le résultat suivant :

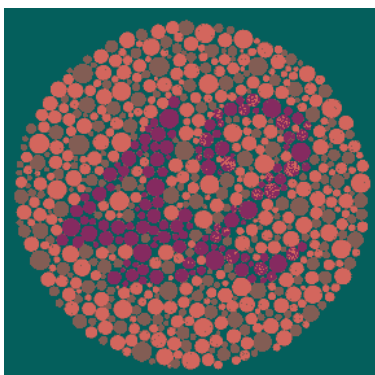


FIGURE 10 – Image segmentée du 42

On peut constater que cette segmentation produit davantage classe que ce que l'on en veut vraiment. On voudrait segmenter l'image, dans l'idéal, en 2 classes : le fond et le nombre. En vue même d'un résultat similaire que l'approche RGB, on se pose donc la question de l'utilité de l'analyse en composantes principales (ACP). L'intérêt de l'approche ACP résidera dans le choix des composantes utilisées. Ainsi voici les différentes composantes (canaux) de l'image ACP :

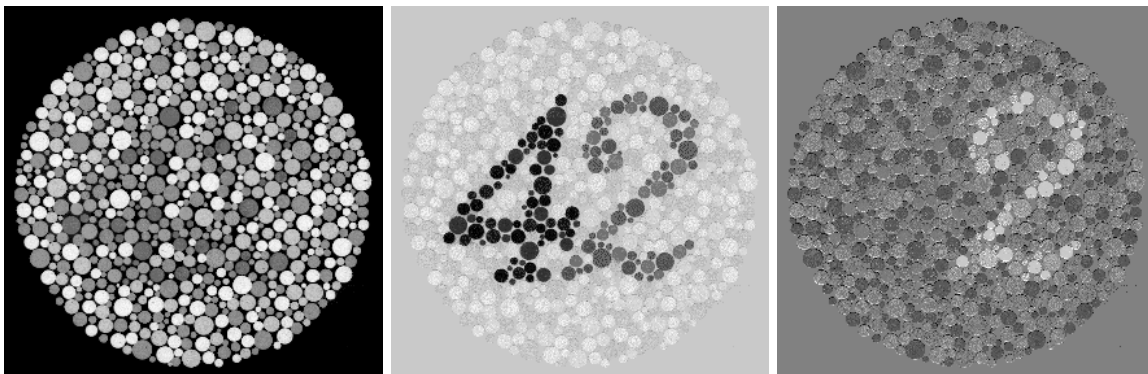


FIGURE 11 – composantes de l'image ACP

La composante numéro 2 est la plus intéressante avec un fond clair et une forme foncée. On va donc appliquer k-means sur cette image avec un  $k$  égal à 2 :



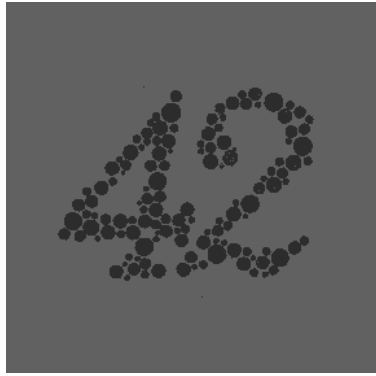


FIGURE 12 – image segmentée

Ainsi le résultat est bien plus satisfaisante car nous n'avons plus que 2 classes de pixels : le nombre et le fond.

Essayons donc la même classification, avec le même  $k$  (2), sur l'image ACP du 26 qui ressemble à celle du 42 (en utilisant le 2e canal ACP pour effectuer le k-means) :

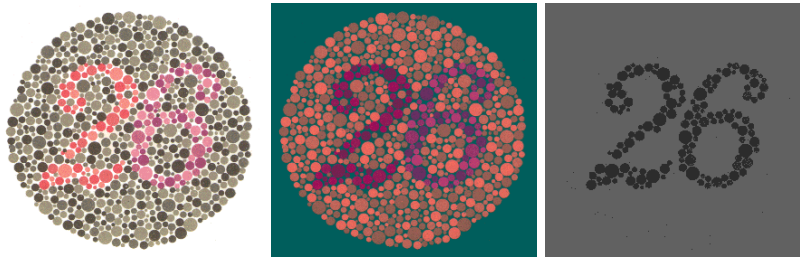


FIGURE 13 – Image Couleur, ACP et segmentée du 26

On peut conclure de l'intérêt d'utiliser l'analyse en composantes principales pour une bonne segmentation de nos images, car même en changeant de méthode de classification, nous obtenons des résultats toujours aussi probants. À condition, bien sûr, de choisir judicieusement les canaux ACP utilisés...

## A Annexe - Macro Question 1

```
1 #Suliac Lavenant et Antoine Nollet
2
3 # les imports nécessaires
4 import numpy as np
5 import cv2
6 import matplotlib.pyplot as plt
7 from sklearn.metrics import confusion_matrix
8 from sklearn.cluster import KMeans
9
10 #ouvre une image en niveau de gris et vérifie qu'elle est bien de dimension 2
11 def openImg(name):
12     image = plt.imread( 'images/' + name + '.png' )
13     if image.ndim > 2:
14         image = image[:, :, 0]
15
16     image[:, :] = (image[:, :] * 255).astype(np.uint8)
17
18     return image
19
20 #affiche une image en niveau de gris
21 def showGreyImage(image):
22     plt.imshow(image, cmap='gray' )
23     plt.show()
24
25 #affiche une image
26 def showImage(image):
27     plt.imshow(image)
28     plt.show()
29
30 #affichage de l'histogramme de image
31 def showHistogramOf(image):
32     nbins=255
33     countsg, edgesg = np.histogram(image, bins=nbins, range=(0,255))
34     plt.hist(edgesg[:-1], nbins, weights=countsg)
35     plt.show()
36
37 #seuille l'image image selon le seuil seuil
38 def seuillage(image, seuil):
39     return np.where(image > seuil, 255, 0)
40
41 #seuille l'image image selon le seuil seuil
42 def seuillageInv(image, seuil):
43     return np.where(image > seuil, 0, 255)
44
45 #seuille l'image image selon les seuils seuil1 et seuil2
46 def doubleSeuillage(image, seuil1, seuil2):
47     imageSeuil1 = np.where(image > seuil1, 127, 0)
48     imageSeuil2 = np.where(image > seuil2, 255, 0)
49     return np.maximum(imageSeuil1, imageSeuil2)
50
51 #calcule la matrice de confusion de l'image image
52 def calculateConfusionMatrice(image, name):
53     ImageSeuil_1D = image.flatten()
54     GT_1D = (openImg(name+'_GT')).flatten()
55     cm = confusion_matrix(GT_1D, ImageSeuil_1D)
56     return cm
57
58 def ostsu2(image):
59     imageFlat = image.flatten()
60
61     # l'histogramme des niveaux de gris
62     h = [np.count_nonzero(imageFlat==i) for i in range(256)]
63
64     N = len(imageFlat)
65
66     dispersionMax = 0
67     tMax = 0
```

```

68     for t in range(255):
69
70         # calcul des P(w1(t)) et N(w1(t))
71         pw1 = 0
72         nw1 = 0
73         for i in range(t+1):
74             pw1 += h[i] / N
75             nw1 += h[i]
76         # duction des P(w2(t)) et N(w2(t))
77         pw2 = 1-pw1
78         nw2 = N-nw1
79
80         # On ne retient pas les cas où une classe est vide
81         # Cela reviendrait à traiter une classe et non deux et cela n'a pas de sens
82         if (nw1!=0 and nw2!=0):
83             #calcul 1
84             uw1=0
85             for i in range(t+1):
86                 uw1+=(i*h[i])
87             uw1 = uw1/nw1
88             #calcul 2
89             uw2=0
90             for i in range(t+1,256):
91                 uw2+=(i*h[i])
92             uw2 = uw2/nw2
93             #calcul de la dispersion et mis à jour (maximisation) des valeurs courantes
94             dispersion = pw1*pw2*((uw1-uw2)**2)
95             if (dispersion>dispersionMax):
96                 dispersionMax=dispersion
97                 tMax=t
98
99         print("Dispersion maximale: "+str(dispersionMax))
100        print("Valeur de seuil optimale: "+str(tMax))
101
102        imageBin = seuillage(image, tMax)
103
104        return imageBin
105
106
107    # fonction de la sous section "Transformation en matrice de données"
108    def matrice_donnes(image):
109        # on récupère les composantes couleur de l'image
110        r = image[:, :, 0]
111        g = image[:, :, 1]
112        b = image[:, :, 2]
113
114        # on s'en sert pour déterminer les vecteurs d'attributs RGB, on les utilise en 1 dimension chacun
115        rFlat = r.flatten()
116        gFlat = g.flatten()
117        bFlat = b.flatten()
118
119        # on construit notre matrice de données
120        x = np.zeros((len(image)*len(image[0]),3), dtype=int)
121        x[:,0] = rFlat[:]
122        x[:,1] = gFlat[:]
123        x[:,2] = bFlat[:]
124
125        return x
126
127
128    # Projection en ACP
129    def analyseEnComposantesPrincipales(x):
130        # vecteur M des moyennes de valeurs
131        x_meaned = x - np.mean(x, axis=0)
132        # calcul de la matrice de co-variance (dimension 3x3)
133        cov_mat = np.cov(x_meaned, rowvar=False)
134
135        # valeurs propres et vecteurs propres de la matrice de co-variance
136        eigen_values, eigen_vectors = np.linalg.eigh(cov_mat)
137
138        # tri des indices selon les valeurs propres les plus grandes

```

```

139 sorted_index = np.argsort(eigen_values)[::-1]
140 # on trie les valeurs propres selon les indices
141 sorted_eigenvalue = eigen_values[sorted_index]
142 # on trie les vecteurs propres selon les indices
143 sorted_eigenvectors = eigen_vectors[:, sorted_index]
144
145
146 #####
147 #sorted_eigenvectors[:,0] = sorted_eigenvectors[:,0] * -1 #bidouillage si python < 3.9
148 #####
149
150 # calcul de la projection par ACP des données grâce à la matrice W des vecteurs propres :  $Y = W^T \cdot X$ 
151 y_projected = np.dot(sorted_eigenvectors.transpose(), x_meaned.transpose()).transpose()
152
153 return y_projected
154
155 # On termine les 3 canaux grâce à la matrice des données projetée par ACP (qui est de même dimension que l'originale)
156 def projectedMatriceOn3Canal(xPCA):
157     #récupération du min et du max pour normalisation
158     min = xPCA.min()
159     max = xPCA.max()
160
161     #normalisation des valeurs entre 0 et 255
162     xPCA = ((xPCA-min)/(max-min))*255
163
164     #enlève les chiffres après la virgule
165     xPCA = np.trunc(xPCA)
166     #change les valeurs de float vers int
167     xPCA=xPCA.astype('uint8')
168
169     # définition des différents canaux ACP
170     c0PCA=np.reshape(xPCA[:,0], (len(xPCA),1))
171     c1PCA=np.reshape(xPCA[:,1], (len(xPCA),1))
172     c2PCA=np.reshape(xPCA[:,2], (len(xPCA),1))
173
174     return c0PCA, c1PCA, c2PCA
175
176
177 def otsu(c0, c1, c2):
178
179     # Binarisation sous Otsu de chaque composante
180     c0Bin=otsu2(c0)
181     #showGreyImage(c0Bin)
182     #plt.imshow("c0Bin.png", c0Bin, cmap='gray')
183     c1Bin=otsu2(c1)
184     #showGreyImage(c1Bin)
185     #plt.imshow("c1Bin.png", c1Bin, cmap='gray')
186     c2Bin=otsu2(c2)
187     #showGreyImage(c2Bin)
188     #plt.imshow("c2Bin.png", c2Bin, cmap='gray')
189
190     #Fusion des composantes pour reformer une image couleur
191     finalBin=np.zeros((len(c0), len(c0[0]), 3), dtype=np.uint8)
192     finalBin[:, :, 0]=c2Bin[:, :]
193     finalBin[:, :, 1]=c1Bin[:, :]
194     finalBin[:, :, 2]=c0Bin[:, :]
195
196     showImage(finalBin)
197     #plt.imshow("finalBinc200.png", finalBin)
198
199
200 def kmeans(k, image, show=True, save=True):
201     # les données relevées de l'image
202     x=matrice_donnes(image)
203     # création du kmeans avec k classes à terminer
204     k_means = KMeans(n_clusters=k)
205     k_means.fit(x)
206     # centroids : vecteur de dimension n x 3
207     centroids = k_means.cluster_centers_
208     centroids = np.asarray(centroids, dtype=np.uint8)
209     # labels: vecteur de dimension nbpixels x 1

```

```

210     labels = k_means.labels_
211     # centroid_vector: vecteur de dimension nbpixels x 3
212     centroid_vector = centroids[labels]
213     centroid_array_2D = centroid_vector.reshape(len(image), len(image[0]), len(image[0][0]))
214
215     # affichage de l'image
216     if show:
217         showImage(centroid_array_2D)
218
219     #enregistrement de l'image
220     if save:
221         plt.imsave("cas_2_dalton73-k"+str(k)+".png", centroid_array_2D)
222
223     # r utilisation pour autres images
224     return k_means, centroids
225
226
227 def autoSegment(k_means, centroids, nameAndFormat, show=True, save=True):
228     # ouverture de l'image
229     image = plt.imread('images/' + nameAndFormat)
230     #plt.imsave(nameAndFormat.split(".")[0]+".png", image)
231
232     # matrice de données relative à l'image
233     test_color_image_vector = matrice_donnes(image)
234     # r utilisation du k means précédent pour effectuer la classification
235     test_labels = k_means.predict(test_color_image_vector)
236     test_centroid_vector = centroids[test_labels]
237     # remise en image 2D
238     centroid_array_2D = test_centroid_vector.reshape(len(image), len(image[0]), len(image[0][0]))
239
240     # affichage de l'image
241     if show:
242         showImage(centroid_array_2D)
243
244     #enregistrement de l'image
245     if save:
246         plt.imsave(nameAndFormat.split(".")[0]+"seg.png", centroid_array_2D)
247
248
249 def question1():
250     #ouverture de l'image
251     nameAndFormat = "cas_2_dalton73.bmp"
252     image = plt.imread('images/' + nameAndFormat)
253     #plt.imsave("cas_2_dalton73.png", image)
254
255     # n : nombre de classes
256     n = 8
257
258     # utilisation du k means pour classifier
259     k_means, centroids = kmeans(n, image)
260
261     # r utilisation de la classification pour d'autres images
262     listeName = ["cas_2_dalton7.bmp", "cas_2_dalton16.bmp"]
263     for name in listeName:
264         autoSegment(k_means, centroids, name)
265
266
267 question1()

```

## B Annexe - Macro Question 3

```
1
2 #Suliac Lavenant et Antoine Nollet
3
4 # les imports n ecessaires
5 import numpy as np
6 import cv2
7 import matplotlib.pyplot as plt
8 from sklearn.metrics import confusion_matrix
9 from sklearn.cluster import KMeans
10
11 #ouvre une image en niveau de gris et v rifie qu'elle est bien de dimension 2
12 def openImg(name):
13     image = plt.imread( 'images/' + name + '.png' )
14     if image.ndim > 2:
15         image = image[:, :, 0]
16
17     image[:, :] = (image[:, :] * 255).astype(np.uint8)
18
19     return image
20
21 #affiche une image en niveau de gris
22 def showGreyImage(image):
23     plt.imshow(image, cmap='gray' )
24     plt.show()
25
26 #affiche une image
27 def showImage(image):
28     plt.imshow(image)
29     plt.show()
30
31 #affichage de l'histogramme de image
32 def showHistogramOf(image):
33     nbins=255
34     countsg, edgesg = np.histogram(image, bins=nbins, range=(0,255))
35     plt.hist(edgesg[:-1], nbins, weights=countsg)
36     plt.show()
37
38 #seuille l'image image selon le seuil seuil
39 def seuillage(image, seuil):
40     return np.where(image > seuil, 255, 0)
41
42 #seuille l'image image selon le seuil seuil
43 def seuillageInv(image, seuil):
44     return np.where(image > seuil, 0, 255)
45
46 #seuille l'image image selon les seuils seuil1 et seuil2
47 def doubleSeuillage(image, seuil1, seuil2):
48     imageSeuil1 = np.where(image > seuil1, 127, 0)
49     imageSeuil2 = np.where(image > seuil2, 255, 0)
50     return np.maximum(imageSeuil1, imageSeuil2)
51
52 #calcule la matrice de confusion de l'image image
53 def calculateConfusionMatrice(image, name):
54     ImageSeuil_1D = image.flatten()
55     GT_ID = (openImg(name+'.GT')).flatten()
56     cm = confusion_matrix(GT_ID, ImageSeuil_1D)
57     return cm
58
59 # fonction de la sous section "Transformation en matrice de donn es"
60 def matrice_donnes(image):
61     # on r ecup re les composantes couleur de l'image
62     r = image[:, :, 0]
63     g = image[:, :, 1]
64     b = image[:, :, 2]
65
66     # on s'en sert pour d terminer les vecteurs d'attributs RGB, on les utilise en 1 dimension chacun
67     rFlat = r.flatten()
```

```

68     gFlat = g.flatten()
69     bFlat = b.flatten()
70
71     # on construit notre matrice de donn es
72     x = np.zeros((len(image)*len(image[0]),3), dtype=int)
73     x[:,0] = rFlat[:]
74     x[:,1] = gFlat[:]
75     x[:,2] = bFlat[:]
76
77     return x
78
79
80 # Projection en ACP
81 def analyseEnComposantesPrincipales(x):
82     # vecteur M des moyennes de valeurs
83     x_meaned = x - np.mean(x, axis=0)
84     # calcul de la matrice de co-variance (dimension 3x3)
85     cov_mat = np.cov(x_meaned, rowvar=False)
86
87     # valeurs propres et vecteurs propres de la matrice de co-variance
88     eigen_values, eigen_vectors = np.linalg.eigh(cov_mat)
89
90     # tri des indices selon les valeurs propres les plus grandes
91     sorted_index = np.argsort(eigen_values)[::-1]
92     # on trie les valeurs propres selon les indices
93     sorted_eigenvalue = eigen_values[sorted_index]
94     # on trie les vecteurs propres selon les indices
95     sorted_eigenvectors = eigen_vectors[:,sorted_index]
96
97
98     #####
99     #sorted_eigenvectors[:,0] = sorted_eigenvectors[:,0] * -1 #bidouillage si python < 3.9
100    #####
101
102    # calcul de la projection par ACP des donn es gr ce la matrice W des vecteurs propres :  $Y = W^T \cdot X$ 
103    y_projected = np.dot(sorted_eigenvectors.transpose(), x_meaned.transpose()).transpose()
104
105    return y_projected
106
107 # On d termine les 3 canaux gr ce la matrice des donn es projet e par ACP (qui est de m me dimension c
108 def projectedMatriceOn3Canal(xPCA):
109     #r cuperation du min et du max pour normalisation
110     min = xPCA.min()
111     max = xPCA.max()
112
113     #normalisation des valeurs entre 0 et 255
114     xPCA = ((xPCA-min)/(max-min))*255
115
116     #enleve les chiffres apr s la virgule
117     xPCA = np.trunc(xPCA)
118     #change les valeur de float vers int
119     xPCA=xPCA.astype('uint8')
120
121     # d finition des diff rents canaux ACP
122     c0PCA=np.reshape(xPCA[:,0], (len(xPCA),1))
123     c1PCA=np.reshape(xPCA[:,1], (len(xPCA),1))
124     c2PCA=np.reshape(xPCA[:,2], (len(xPCA),1))
125
126     return c0PCA, c1PCA, c2PCA
127
128 def projectedMatriceOn1Canal(xPCA):
129     #r cuperation du min et du max pour normalisation
130     min = xPCA.min()
131     max = xPCA.max()
132
133     #normalisation des valeurs entre 0 et 255
134     xPCA = ((xPCA-min)/(max-min))*255
135
136     #enleve les chiffres apr s la virgule
137     xPCA = np.trunc(xPCA)
138     #change les valeur de float vers int

```



```

139 xPCA=xPCA.astype('uint8')
140
141 return xPCA
142
143
144 def kmeans(k,x):
145     # cr ation du kmeans avec k classes d terminer
146     k_means = KMeans(n_clusters=k)
147     k_means.fit(x)
148     # centroids : vecteur de dimension n x 3
149     centroids = k_means.cluster_centers_
150     centroids = np.asarray(centroids,dtype=np.uint8)
151     # labels: vecteur de dimension nbpixels x 1
152     labels = k_means.labels_
153     # centroid_vector: vecteur de dimension nbpixels x 3
154
155     # r utilisation pour autres images
156     return k_means, centroids, labels
157
158
159 def autoSegment(k_means, centroids, nameAndFormat,show=True,save=True):
160     # ouverture de l'image
161     image = plt.imread('images/' + nameAndFormat)
162     #plt.imshow(nameAndFormat.split(".")[0]+".png", image)
163
164     # matrice de donn es relative l'image projet e ensuite en ACP
165     test_color_image_vector = matrice_donnes(image)
166     test_color_image_vector = analyseEnComposantesPrincipales(test_color_image_vector)
167
168     # On utilisait tout les canaux
169     #test_color_image_vector = projectedMatriceOn1Canal(test_color_image_vector)
170     # On va pr f rer utiliser le 2e canal ACP
171     c0,c1,c2 = projectedMatriceOn3Canal(test_color_image_vector)
172     test_color_image_vector=np.zeros((len(c0), 3), dtype=np.uint8)
173     test_color_image_vector[:,0]=c1[:,0]
174     test_color_image_vector[:,1]=c1[:,0]
175     test_color_image_vector[:,2]=c1[:,0]
176
177     # r utilisation du k means pr c dent pour effectuer la classification
178     test_labels = k_means.predict(test_color_image_vector)
179     test_centroid_vector = centroids[test_labels]
180     # remise en image 2D
181     centroid_array_2D = test_centroid_vector.reshape(len(image),len(image[0]),len(image[0][0]))
182
183     # affichage de l'image
184     if show:
185         showImage(centroid_array_2D)
186
187     #enregistrement de l'image
188     if save:
189         plt.imshow(nameAndFormat.split(".")[0]+"seg.png", centroid_array_2D)
190
191
192 # fonction d'affichage
193 def afficheACP(xPCA,shape,show=True,save=True):
194
195     finalBin = xPCA.reshape(shape)
196
197     if show:
198         showImage(finalBin)
199     if save:
200         plt.imshow("cas_1_dalton42acp.png", finalBin)
201
202
203
204 def question3():
205     #ouverture de l'image
206     nameAndFormat = "cas_1_dalton42.bmp"
207     image = plt.imread('images/' + nameAndFormat)
208
209     #matrice des donn es

```

```

210     x = matrice_donnes(image)
211
212     # Projection en ACP
213     xPCA = analyseEnComposantesPrincipales(x)
214
215     # On va pr f rer utiliser des canaux sp cifiques plut t qu utiliser tous les canaux
216     #xPCA = projectedMatriceOn1Canal(xPCA)
217     c0,c1,c2 = projectedMatriceOn3Canal(xPCA)
218
219     # on utilise le 2e canal
220     finalBin=np.zeros((len(c0), 3), dtype=np.uint8)
221     finalBin[:,0]=c1[:,0]
222     finalBin[:,1]=c1[:,0]
223     finalBin[:,2]=c1[:,0]
224     xPCA = finalBin
225
226     # En choisissant un canal sp cifique , on aura moins de classes
227     k=2
228     #k=4
229
230     # Utilisation de K-Means
231     k_means, centroids, labels = kmeans(k,xPCA)
232
233     centroid_vector = centroids[labels]
234     centroid_array_2D = centroid_vector.reshape(len(image),len(image[0]),len(image[0][0]))
235     #plt.imsave(nameAndFormat.split(".")[0]+"segAcp0.png", centroid_array_2D[:, :, 0], cmap='gray')
236     #plt.imsave(nameAndFormat.split(".")[0]+"segAcp1.png", centroid_array_2D[:, :, 1], cmap='gray')
237     #plt.imsave(nameAndFormat.split(".")[0]+"segAcp2.png", centroid_array_2D[:, :, 2], cmap='gray')
238     centroid_array_2D[:, :, 0]=centroid_array_2D[:, :, 1]
239
240     plt.imsave(nameAndFormat.split(".")[0]+"seg.png", centroid_array_2D)
241     showImage(centroid_array_2D)
242
243     # R utilisation de la classification pour une autre image
244     listeName = ["cas_1_dalton26.bmp"]
245     for name in listeName:
246         autoSegment(k_means, centroids, name)
247
248 question3()

```