

# Stephen\_Cohen\_PCA\_Kmean

March 22, 2024

## 1 SI221 - TP PCA & K-means

### 1.0.1 Données

Nous allons travailler sur des images satellitaire: celles-ci sont capturées via différent capteurs, chacun correspondant à un domaine de longueur d'ondes. Nous avons donc, pour chaque image, un certain nombre de *canaux* - chacun correspondant à un *domaine spectral*.

- Données SPOT sur Tarascon: La taille du pixel est de 20m. Il y a 3 canaux:
  - 0.50-0.59  $\mu$
  - 0.61-0.68  $\mu$
  - 0.78-0.89  $\mu$
- Données LANDSAT sur Tarascon: La taille du pixel est de 30m. Il y a 8 canaux:
  - 0.45-0.52  $\mu$
  - 0.52-0.60  $\mu$
  - 0.63-0.69  $\mu$
  - 0.75-0.90  $\mu$  (Proche InfraRouge : PIR)
  - 2.08-2.35  $\mu$  (Moyen InfraRouge : MIR)
  - 10.40-12.50  $\mu$  (infra rouge thermique)
  - 1.55-1.75  $\mu$  (Moyen InfraRouge : MIR)
  - 0.52-0.90  $\mu$  (panchromatique)
- Données LANDSAT sur Kedougou: La taille du pixel est de 30m; mais, à la différence des données acquises sur Tarascon, il y a un problème d'acquisition de données inhérent au capteur. Il y a ici 6 canaux:
  - 0.45-0.52  $\mu$
  - 0.52-0.60  $\mu$
  - 0.63-0.69  $\mu$
  - 0.76-0.90  $\mu$
  - 2.08-2.35  $\mu$
  - 1.55-1.75  $\mu$

STEPHEN COHEN

## 1.0.2 PCA

Le recours à l'ACP va permettre de réduire la dimension de l'espace de représentation et ici de réduire le nombre de canaux nécessaires pour conserver l'essentiel de l'information. De manière générale, le but de l'ACP est de visualiser dans un espace de plus petite dimension les proximités entre les observations (ici les pixels) et ainsi les corrélation entre les variables (les valeurs des pixels sur les différents canaux).

**Rappel sur l'ACP:** Avec  $N$  individus, pour lesquels on dispose de  $p$  caractéristiques (ce qui veut dire que son vecteur est de dimension  $p$ ), on notera  $\mathbf{x}_k^i$  la  $k$  ième caractéristique de l'individu  $i$ . On note donc  $\mathbf{X}$  la matrice dans laquelle chaque ligne est constituée par un individu et chaque colonne représente une variable.

L'ACP est constituée par les étapes suivantes : - Centrage et réduction des données: Notant  $m_k$  et  $\sigma_k$ , les moyennes et écarts-type de la  $k$  ième variable, on notera  $\mathbf{x}'_k^i = \frac{\mathbf{x}_k^i - m_k}{\sigma_k}$  la donnée centrée réduite. - Calcul de la matrice de covariance des données centrées réduites. - Calcul des valeurs propres  $\lambda_j$  et vecteurs propres  $\mathbf{u}_j$  de la matrice de covariance; - Vérification de l'ordre des vecteurs propres selon les valeurs propres croissantes; - Calcul des composantes principales  $\mathbf{x}''_q^i$  exprimées dans la base des vecteurs propres:

$$\mathbf{x}''_q^i = \mathbf{x}'_q^T \mathbf{u}_q$$

On obtient donc de nouvelles variables constituées par des combinaisons linéaires des anciennes. Les composantes principales contiennent une quantité d'information proportionnelle à la valeur propre correspondante. On définit ainsi le pourcentage d'inertie par  $\frac{\lambda_i}{\sum_{j=1}^p \lambda_j}$ .

Le but premier de ce TP est **l'application du PCA (ou ACP, Analyse en composantes principales)** à ces images satellitaires, où les individus sont chaque canal de l'image, et les variables chacun des pixels.

```
[ ]: import matplotlib.pyplot as plt
      import numpy as np
```

Les données proviennent de fichier .mat, avec un fichier par canal. On peut commencer par travailler avec SPOT, qui n'a que 3 canaux. Commençons par charger les images et les réunir sur une même matrice.

```
[ ]: from scipy.io import loadmat

def load_image(path_img):
    im_loaded = loadmat('./'+ path_img +'.mat')
    return im_loaded['imm']
```

```
[ ]: # First image: SPOT
path_img = 'SPOT/cam'
n_bands = 3

# Let's take a look at the first channel's size
img = load_image(path_img + '1')
print(img.shape)
```

```
(512, 512)
```

```
[ ]: img = np.zeros((img.shape[0], img.shape[1], n_bands))
print(img.shape)
# Stacking up images into the numpy array:
# A compléter
```

```
(512, 512, 3)
```

```
[[[29. 23. 29.]
```

```
 [30. 24. 29.]
```

```
 [32. 27. 30.]
```

```
...
```

```
 [43. 31. 39.]
```

```
 [36. 24. 32.]
```

```
 [29. 20. 26.]]
```

```
[[35. 28. 36.]
```

```
 [37. 29. 33.]
```

```
 [38. 29. 32.]
```

```
...
```

```
 [40. 28. 38.]
```

```
 [32. 23. 30.]
```

```
 [28. 21. 25.]]
```

```
[[32. 24. 32.]
```

```
 [35. 24. 33.]
```

```
 [34. 22. 31.]
```

```
...
```

```
 [32. 25. 31.]
```

```
 [32. 25. 31.]
```

```
 [34. 24. 31.]]
```

```
...
```

```
[[31. 27. 34.]
```

```
 [32. 28. 35.]
```

```
 [32. 28. 35.]
```

```
...
```

```
 [29. 19. 32.]
```

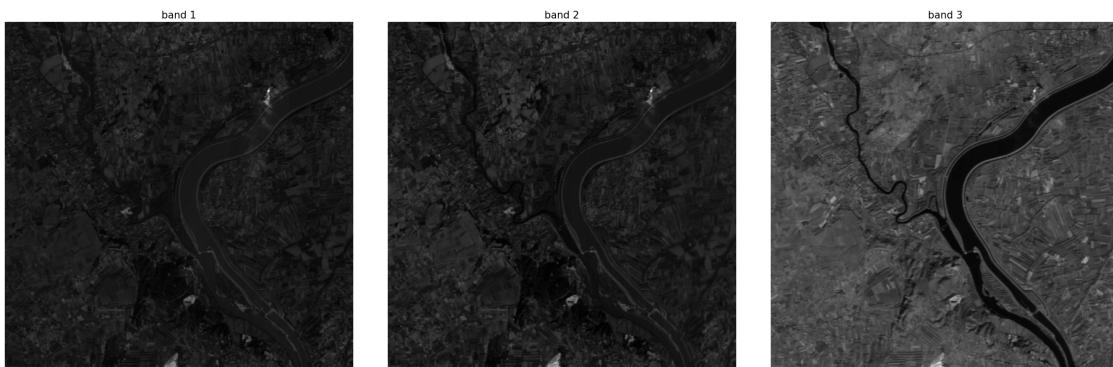
```
[28. 18. 37.]  
[26. 17. 32.]]
```

```
[[32. 28. 34.]  
[33. 29. 34.]  
[33. 29. 35.]  
...  
[37. 25. 37.]  
[31. 21. 38.]  
[24. 16. 29.]]
```

```
[[33. 29. 34.]  
[34. 30. 34.]  
[33. 30. 35.]  
...  
[39. 28. 35.]  
[35. 23. 37.]  
[26. 17. 37.]]]
```

Essayons d'abord de comprendre, en regardant les images, les caractéristiques des différents canaux et les informations qu'ils comportent.

```
[ ]: import matplotlib.pyplot as plt  
import matplotlib.gridspec as grid  
  
fig,axes = plt.subplots(1,n_bands, figsize=(n_bands*10,12), sharex='all',  
                      sharey='all')  
fig.subplots_adjust(wspace=0.1, hspace=0.15)  
axes = axes.ravel()  
  
for i in range(n_bands):  
    # Use floats instead of 8-bit integers for a clearer view of the image  
    axes[i].imshow(img[:, :, i] / 255, cmap='grey')  
    axes[i].set_title('band '+str(i+1), fontsize=15)  
    axes[i].axis('off')
```



Qu'observez-vous ?

Chaque image est un individu: on va d'abord redimensionner les images en vecteurs (pas besoin d'information de structure spatiale !) et on calcule la moyenne et l'écart-type de chacun de ces vecteurs pour les centrer.

```
[ ]: # Reshape the image for practicality:  
# A compléter  
vectors=[]  
for i in range(3):  
    vectors.append(img[:, :, i].ravel())  
print(vectors[2])
```

[29. 29. 30. ... 35. 37. 37.]

```
[ ]: # Standardization:  
for i in range(3):  
    vectors[i]=(vectors[i]-np.mean(vectors[i]))/np.sqrt(np.var(vectors[i]))  
print(vectors[1])  
  
# A compléter
```

[ 0.31463317 0.5357671 1.19916889 ... 1.42030282 0.31463317  
-1.01217042]

**Vecteurs propres et valeurs propres:** on concatène les vecteurs centrés en une seule matrice dont on calcule la matrice de covariance, qui mesure la relation entre les images. On calcule ensuite les valeurs propres et les vecteurs propres de la matrice de covariance grâce à la fonction `eig`. On obtient les vecteurs propres, concaténés.

```
[ ]: matrix=np.matrix(vectors)  
print(matrix)  
  
# Compte the covariance matrix:  
# A compléter  
  
cov_matrix=np.cov(matrix)  
print(cov_matrix)  
  
# Compute eigenvalues and vectors with eig:  
# A compléter  
  
val_propre,vectors_propre=np.linalg.eig(cov_matrix)
```

[[-0.33621727 -0.06011854 0.49207891 ... 2.42476999 1.32037509  
-1.16451345]  
[ 0.31463317 0.5357671 1.19916889 ... 1.42030282 0.31463317  
-1.01217042]]

```
[ 0.12187387  0.12187387  0.26818369 ...  0.99973279  1.29235244
 1.29235244]]
[[1.00000381  0.91281965  0.44564374]
 [0.91281965  1.00000381  0.53484936]
 [0.44564374  0.53484936  1.00000381]]
```

```
[ ]: # Lets sort the eigenvalues to be able to choose the components by importance:
print(np.sort(val_propre))
```

```
[0.0811231  0.63201105 2.28687729]
```

**Calcul des composantes principales:** On multiplie la matrice concaténée par la matrice des vecteurs propres pour obtenir les composantes principales dans la base des vecteurs propres; ainsi, on va pouvoir trouver le sous-espace optimal pour visualiser les proximités entre les observations, et les corrélations entre les variables.

```
[ ]: # Lets get the components in separate matrices, so we can display them easily:
```

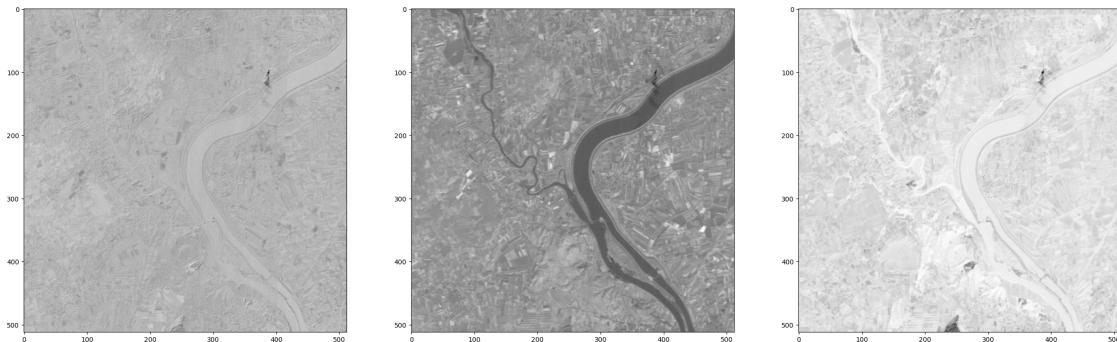
```
sorted_indices = np.argsort(val_propre)

val_propre = val_propre[sorted_indices]
vectors_propre = vectors_propre[sorted_indices]
acp = [np.dot(np.transpose(matrix),vectors_propre[i]) for i in range(3)]
print(len(acp[0]))
```

```
1
```

```
[ ]: plt.figure(figsize=(n_bands*10, 12))
```

```
# Visualize them with imshow:
for i in range(n_bands):
    plt.subplot(1,n_bands,i+1)
    plt.imshow(acp[i].reshape(512,512),cmap='grey')
```



Visuellement, on constate que la première composante détecte un certain nombre d'emplacements

précis, ainsi que des limites - la seconde semble détecter l'eau et les champs, et la dernière indique des artefacts verticaux sans doute liés aux capteurs.

```
[ ]: # Compute the percentage of inertia of each component:
```

```
inertia = [x/sum(val_propre) for x in val_propre]
print(inertia)
```

```
[0.027040930652252458, 0.21066954636873023, 0.7622895229790173]
```

**ACP sur les données LANDSAT sur Tarascon:** Effectuez l'analyse en composantes principales sur les données LANDSAT sur Tarascon. Affichez les images résultats. Calculez le pourcentage d'inertie associé à chaque image et interprétez qualitativement les résultats. Combien d'images faut-il garder pour conserver au moins 95% de l'information ?

```
[ ]: path_img = 'Landstat_Tarascon/landsattarasconC'
n_bands = 8
```

```
def full(path_img,n_bands):
    img = load_image(path_img +'1')
    img = np.zeros((img.shape[0], img.shape[1], n_bands))

    for i in range(1,n_bands+1):
        new_im=load_image(path_img + str(i))
        img[:, :, i-1]=new_im
    vectors=[]

    for i in range(n_bands):
        vectors.append(img[:, :, i].ravel())
    for i in range(n_bands):
        vectors[i]= (vectors[i]-np.mean(vectors[i]))/np.sqrt(np.var(vectors[i]))

    matrix=np.matrix(vectors)

    cov_matrix=np.cov(matrix)

    val_propre,vectors_propre=np.linalg.eig(cov_matrix)
    sorted_indices = np.argsort(val_propre)
    val_propre = val_propre[sorted_indices]
    vectors_propre = vectors_propre[sorted_indices]

    acp = [np.dot(np.transpose(matrix),vectors_propre[i]) for i in
    range(n_bands)]

    plt.figure(figsize=(n_bands*10, 12))
    for i in range(n_bands):
        plt.subplot(1,n_bands,i+1)
        plt.imshow(acp[i].reshape(img.shape[0],img.shape[1]),cmap='grey')
```

```

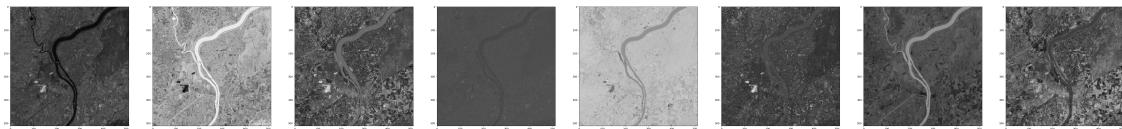
inertia = [x/sum(val_propre) for x in val_propre]
print(inertia)
y=0
i=0
while(y<=0.95):
    y+=inertia[-i-1]
    i+=1
    print("On a besoin de " +str(i)+" images différentes pour conserver 95% de l'information")

full(path_img,n_bands)

```

[0.0012756117698587992, 0.0024915802799023736, 0.0036470798765829257,  
0.016774826455410893, 0.02665518840229342, 0.09404482308887205,  
0.16981092985227558, 0.685299960274804]

On a besoin de 4 images différentes pour conserver 95% de l'information



Les images semblent identifier des objets différents. Par exemple : - Certaines identifient de manière très précise l'eau (images 2 et 7) - D'autres les objets plus précis, comme les champs (image 3) - D'autre encore les hauts reliefs (image 6)

On remarque en effet, que plus l'inertie est importante plus, l'image apporte d'informations via ses contrastes (ie variance de la couleur).

**ACP sur les données LANDSAT sur Kedougou:** Effectuez une ACP sur ce jeu de données. Comparez le résultat avec le dernier capteur défectueux (CANAL6bruit) et l'original.

```

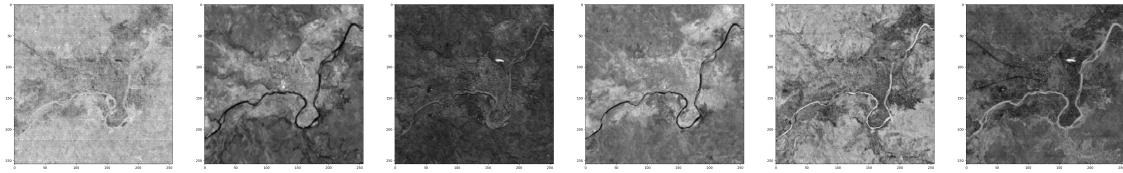
[ ]: # Capteur normal:
path_img = 'Landstat_Kedougou/landsatKedougouC'
n_bands = 6

full(path_img,n_bands)

```

[0.004107167656504177, 0.012708002279120419, 0.013837605764685136,  
0.042149120579943675, 0.10657860465829956, 0.820619499061447]

On a besoin de 3 images différentes pour conserver 95% de l'information

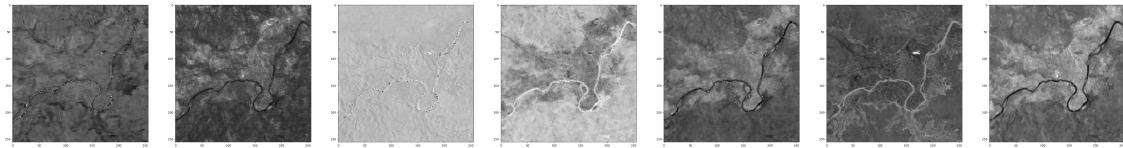


[ ]: # Avec le bruit:

```
full(path_img,n_bands+1)
```

[0.0035136785276456695, 0.010703879876805367, 0.011842091026546606,  
0.03602660192577217, 0.07830584728722105, 0.10856650150835825,  
0.7510413998476508]

On a besoin de 4 images différentes pour conserver 95% de l'information



On observe que l'image avec le bruit porte le moins d'information donc ne pose pas de problème dans notre modèle. Cependant il faut une image de plus pour conserver 95% de l'information.

### 1.0.3 K-means

Dans cette partie, on souhaite réaliser un classificateur automatique pour réaliser une segmentation simple d'images : par exemple, pour séparer terres et rivière. - Tracer l'histogramme canal par canal est instructif ! - La recherche de prototypes nécessite un peu de patience : il faut rechercher sur l'image les coordonnées de points dont on a pu (ou su) voir l'originalité.

**Classification rivière-terre:** Commencez par tester la procédure sur *SPOT* à l'aide des valeurs exemples pour l'eau et sur la terre.

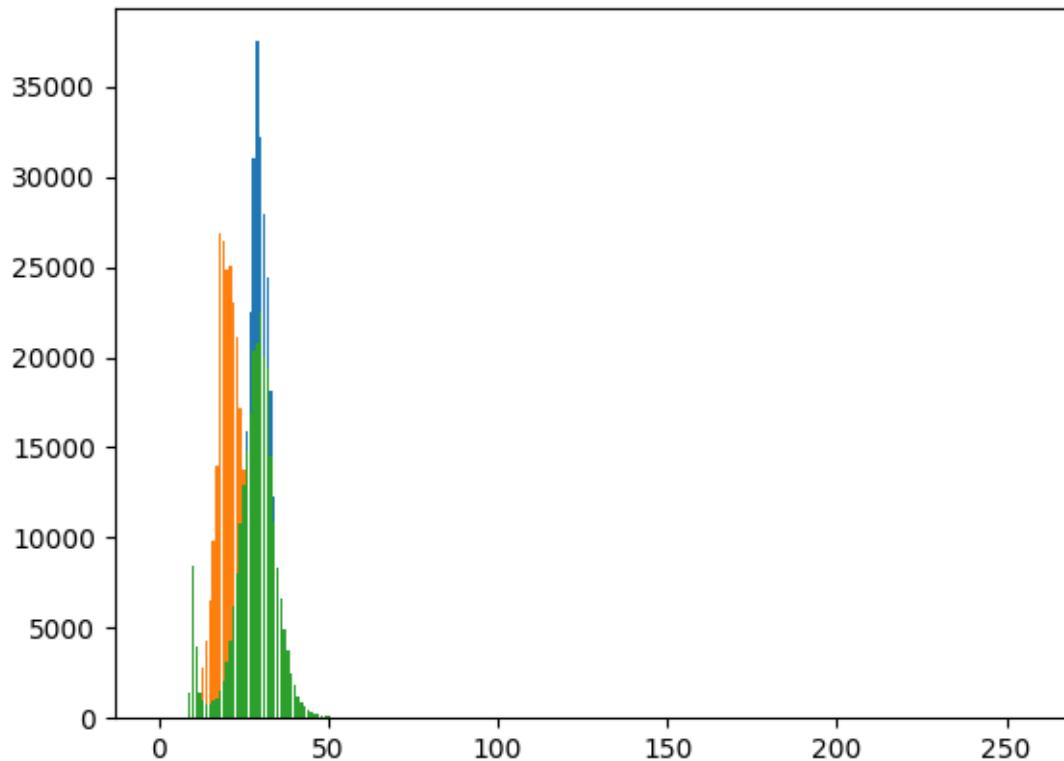
[ ]: 

```
def histogramme(image):
    hist = np.zeros(256, dtype=np.float32)
    shap = image.shape
    for j in range(shap[0]):
        for i in range(shap[1]):
            valeur = image[j,i]
            hist[int(valeur)] += 1
    return hist
```

```
[ ]: path_img = 'SPOT/cam'
n_bands = 3
size = 512

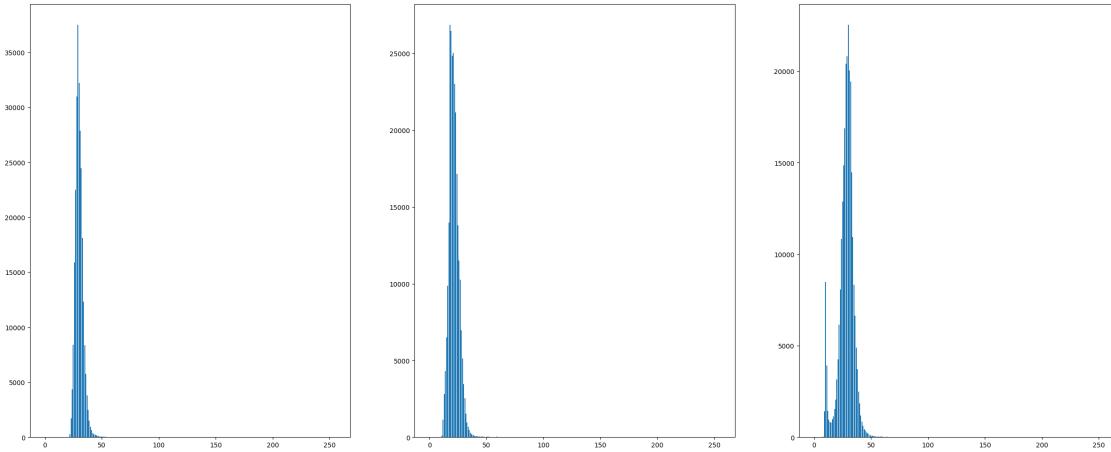
img=[load_image(path_img + str(i)) for i in range(1,n_bands+1)]
for i in range(len(img)):
    plt.bar([i for i in range(256)],histogramme(img[i]))

plt.show()
```



```
[ ]: plt.figure(figsize=(n_bands*10, 12))

# Use the "histogramme" function to have an intuition of pixel repartition on
each image
for i in range(n_bands):
    plt.subplot(1,n_bands,i+1)
    plt.bar([i for i in range(256)],histogramme(img[i]))
```



```
[ ]: # Lieu du pixel caractéristique de l'eau
p1=[210, 150]
# Lieu du pixel caractéristique de la terre
p2=[50, 50]
```

Utilisez la classe KMeans de `sklearn`. Choisissez le nombre de classes avec `n_clusters` et les centres initiaux des classes avec `init`. Enfin, faites tourner l'algorithme sur l'image choisie avec `.fit()` et obtenez les classes de chaque pixel avec `.predict()`. N'oubliez pas d'adapter la structure des données: nous n'avons ici qu'un exemple pour la procédure (un seul canal) avec un grand nombre de caractéristiques à classer (les pixels) - choisissez les tailles de vos entrées en conséquence !

```
[ ]: from sklearn.cluster import KMeans

n_clusters=2

kmeans = KMeans(n_clusters=n_clusters, init=[[img[0][p1[0]][p1[1]]], [img[0][p2[0]][p2[1]]]])
img_reshaped = np.reshape(img[0], (img[0].shape[0] * img[0].shape[1], 1))
kmeans.fit(img_reshaped)
labels = kmeans.predict(img_reshaped)
image_segmented = np.reshape(labels, img[0].shape)

kmeans = KMeans(n_clusters=n_clusters, init=[[img[1][p1[0]][p1[1]]], [img[1][p2[0]][p2[1]]]])
img_reshaped = np.reshape(img[1], (img[1].shape[0] * img[1].shape[1], 1))
kmeans.fit(img_reshaped)
labels = kmeans.predict(img_reshaped)
image_segmented2 = np.reshape(labels, img[1].shape)

kmeans = KMeans(n_clusters=n_clusters, init=[[img[2][p1[0]][p1[1]]], [img[2][p2[0]][p2[1]]]])
```

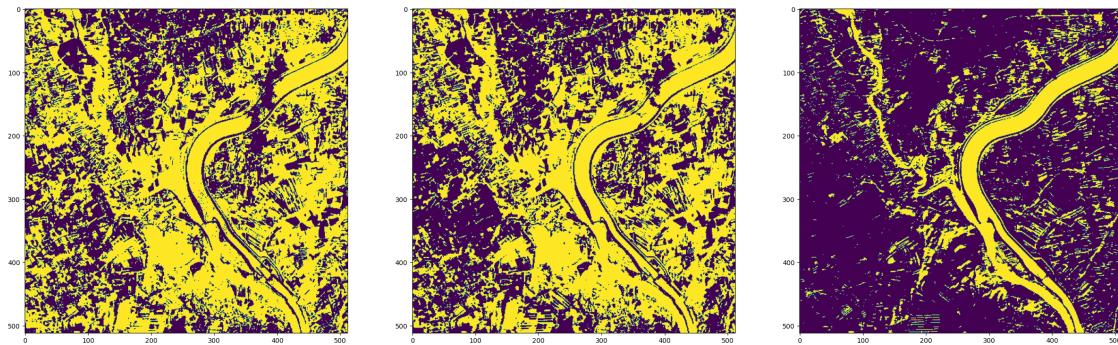
```

img_reshaped = np.reshape(img[2], (img[2].shape[0] * img[2].shape[1], 1))
kmeans.fit(img_reshaped)
labels = kmeans.predict(img_reshaped)
image_segmented3 = np.reshape(labels, img[2].shape)

plt.figure(figsize=(n_bands*10, 12))
plt.subplot(1,n_bands,1)
plt.imshow(image_segmented)
plt.subplot(1,n_bands,2)
plt.imshow(image_segmented2)
plt.subplot(1,n_bands,3)
plt.imshow(image_segmented3)

```

[ ]: <matplotlib.image.AxesImage at 0x7f4581c2b2b0>



Faites de même avec les deux autres images. Avec quels canaux pouvez vous obtenir les meilleures segmentation ? Jusqu'à combien de classes ?

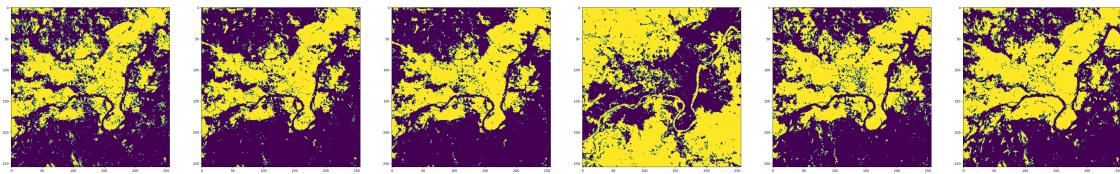
```

[ ]: path_img = 'Landstat_Kedougou/landsatkedougouC'
n_bands = 6
size = 256

def full2(path_img, n_bands):
    plt.figure(figsize=(n_bands*10, 12))
    for i in range(1, n_bands + 1):
        n_clusters=2
        img = load_image(path_img + str(i))
        kmeans = KMeans(n_clusters=n_clusters, init=[[img[p1[0]][p1[1]]],[img[p2[0]][p2[1]]]])
        img_reshaped = np.reshape(img, (img.shape[0] * img.shape[1], 1))
        kmeans.fit(img_reshaped)
        labels = kmeans.predict(img_reshaped)
        image_segmented = np.reshape(labels, img.shape)
        plt.subplot(1,n_bands,i)
        plt.imshow(image_segmented)

```

```
full2(path_img, n_bands)
```



```
[ ]: path_img = 'Landstat_Tarascon/landsattarasconC'  
n_bands = 8  
size = 512
```

```
full2(path_img,n_bands)
```

