Compte Rendu de TP 04

Diagnostic par apprentissage

Segmentation d'images microscopiques

Nom : BOUARICHE

Prénom : Iheb

Année : 2023/2024

Spécialité : Instrumentation an2

Encadré par : Mme. Anissa MOKRAOUI

B) Extraction des attributs et preparation du jeu de données

Question 01:

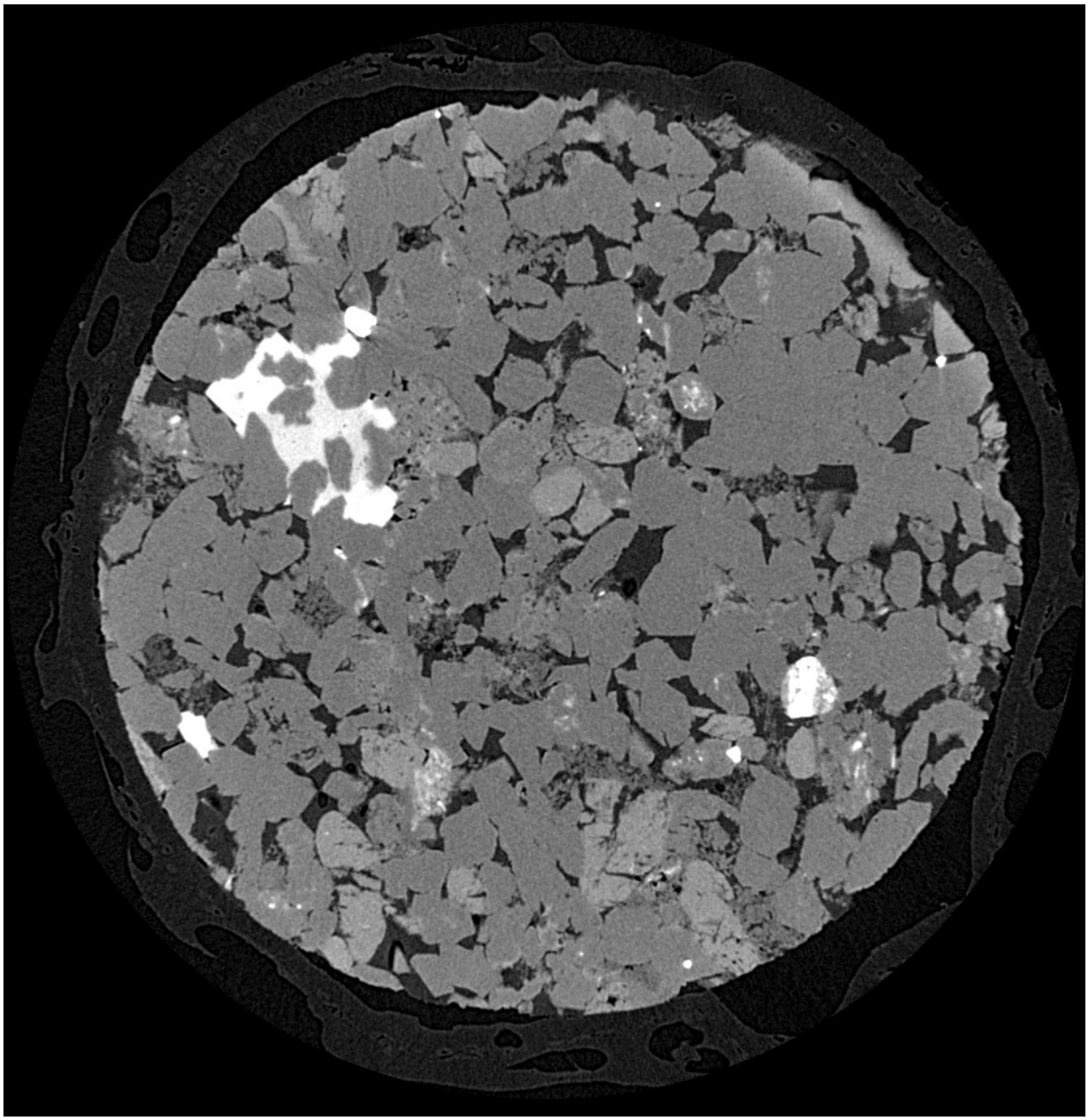
```
import numpy as np
import cv2
import pandas as pd
```

Commentaire: Dans cette partie on a fais l'importation des bibliothques. Numpy est utilisé pour la définition des vecteurs et des matrices. Pandas est utilisé pour la manipulation des données, et cv2 est employé pour la manipulation des images.

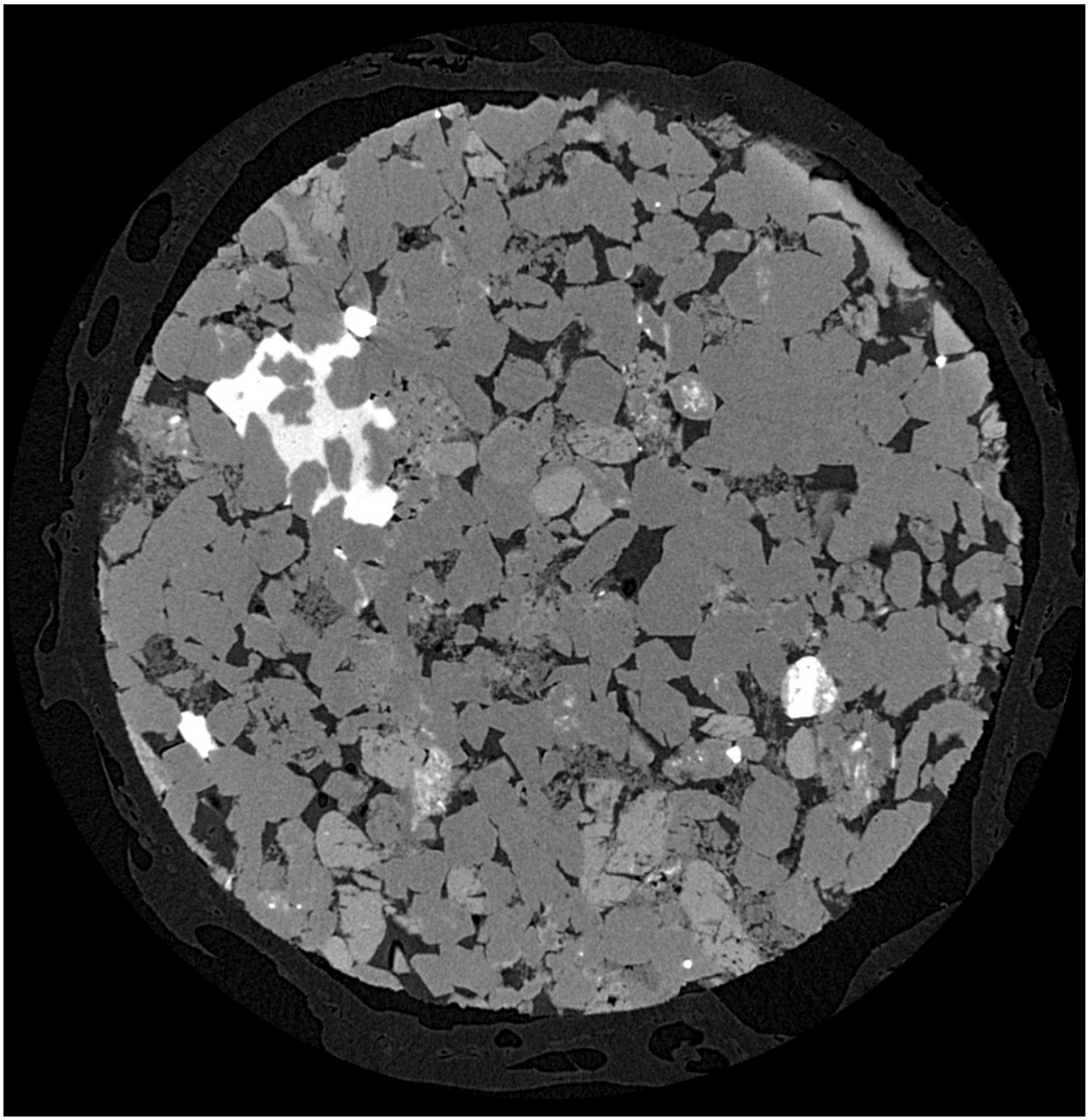
Question 02:

```
img = cv2.imread("Sandstone_Versa0000_train.tif")

from google.colab.patches import cv2_imshow
img2 = cv2.imread("Sandstone_Versa0000_train.tif",
cv2.IMREAD_UNCHANGED)
print(img2)
cv2_imshow(img2)
```

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```



```
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2_imshow(img)
```



```
img2.shape
```

```
(1024, 996)
```

```
Img2 =img2.reshape(-1)
```

Commentaire: On a commencé par l'affichage de l'image qui est de dimension de (1024, 996) et puis on a platé l'image pour avoir un vecteur unidimensionel de 1019904 valeurs.

Question 03:

a) Commentaire : Ces lignes de code visent à créer un DataFrame vide avec une colonne nommée "Original Imag" et mettre les valeurs de chaque pixel de l'image précédante dans une ligne.

```
print("le nombre des pixels dans l'image est: ",
img2.shape[0]*img2.shape[1])
```

```
le nombre des pixels dans l'image est:  1019904
```

```
df = pd. DataFrame()
df["Original Imag"] = Img2
df.head
```

```
<bound method NDFrame.head of          Original Imag
0                     0
1                     0
2                     0
3                     0
4                     0
...                 ...
1019899               0
1019900               0
1019901               0
1019902               0
1019903               0

[1019904 rows x 1 columns]>
```

b) Filtrage de Gabor

```
num = 1
kernels = []
for theta in range(2):
  theta = theta / 4. * np.pi
  for sigma in (1, 3):
    for lamda in np.arange(0, np.pi, np.pi / 4):
      for gamma in (0.05, 0.5):
        gabor_label = 'Gabor' + str(num)
        print(gabor_label)
        ksize=9
        kernel = cv2.getGaborKernel((ksize, ksize), sigma,
theta,lamda, gamma, 0, ktype=cv2.CV_32F)
        kernels.append(kernel)
        fimg = cv2.filter2D(img2, cv2.CV_8UC3, kernel)
        filtered_img = fimg.reshape(-1)
        df[gabor_label] = filtered_img
        print(gabor_label, ': theta=', theta, ': sigma=', sigma, ':
lamda=',lamda, ': gamma=', gamma)
        num += 1
```

```
Gabor1
Gabor1 : theta= 0.0 : sigma= 1 : lamda= 0.0 : gamma= 0.05
Gabor2
Gabor2 : theta= 0.0 : sigma= 1 : lamda= 0.0 : gamma= 0.5
Gabor3
Gabor3 : theta= 0.0 : sigma= 1 : lamda= 0.7853981633974483 : gamma=
0.05
Gabor4
Gabor4 : theta= 0.0 : sigma= 1 : lamda= 0.7853981633974483 : gamma=
0.5
Gabor5
Gabor5 : theta= 0.0 : sigma= 1 : lamda= 1.5707963267948966 : gamma=
0.05
Gabor6
Gabor6 : theta= 0.0 : sigma= 1 : lamda= 1.5707963267948966 : gamma=
0.5
Gabor7
Gabor7 : theta= 0.0 : sigma= 1 : lamda= 2.356194490192345 : gamma=
0.05
Gabor8
Gabor8 : theta= 0.0 : sigma= 1 : lamda= 2.356194490192345 : gamma= 0.5
Gabor9
Gabor9 : theta= 0.0 : sigma= 3 : lamda= 0.0 : gamma= 0.05
Gabor10
Gabor10 : theta= 0.0 : sigma= 3 : lamda= 0.0 : gamma= 0.5
Gabor11
Gabor11 : theta= 0.0 : sigma= 3 : lamda= 0.7853981633974483 : gamma=
0.05
Gabor12
Gabor12 : theta= 0.0 : sigma= 3 : lamda= 0.7853981633974483 : gamma=
0.5
Gabor13
Gabor13 : theta= 0.0 : sigma= 3 : lamda= 1.5707963267948966 : gamma=
0.05
Gabor14
Gabor14 : theta= 0.0 : sigma= 3 : lamda= 1.5707963267948966 : gamma=
0.5
Gabor15
Gabor15 : theta= 0.0 : sigma= 3 : lamda= 2.356194490192345 : gamma=
0.05
Gabor16
Gabor16 : theta= 0.0 : sigma= 3 : lamda= 2.356194490192345 : gamma=
0.5
Gabor17
Gabor17 : theta= 0.7853981633974483 : sigma= 1 : lamda= 0.0 : gamma=
0.05
Gabor18
Gabor18 : theta= 0.7853981633974483 : sigma= 1 : lamda= 0.0 : gamma=
0.5
Gabor19
```

```
Gabor19 : theta= 0.7853981633974483 : sigma= 1 : lamda=
0.7853981633974483 : gamma= 0.05
Gabor20
Gabor20 : theta= 0.7853981633974483 : sigma= 1 : lamda=
0.7853981633974483 : gamma= 0.5
Gabor21
Gabor21 : theta= 0.7853981633974483 : sigma= 1 : lamda=
1.5707963267948966 : gamma= 0.05
Gabor22
Gabor22 : theta= 0.7853981633974483 : sigma= 1 : lamda=
1.5707963267948966 : gamma= 0.5
Gabor23
Gabor23 : theta= 0.7853981633974483 : sigma= 1 : lamda=
2.356194490192345 : gamma= 0.05
Gabor24
Gabor24 : theta= 0.7853981633974483 : sigma= 1 : lamda=
2.356194490192345 : gamma= 0.5
Gabor25
Gabor25 : theta= 0.7853981633974483 : sigma= 3 : lamda= 0.0 : gamma=
0.05
Gabor26
Gabor26 : theta= 0.7853981633974483 : sigma= 3 : lamda= 0.0 : gamma=
0.5
Gabor27
Gabor27 : theta= 0.7853981633974483 : sigma= 3 : lamda=
0.7853981633974483 : gamma= 0.05
Gabor28
Gabor28 : theta= 0.7853981633974483 : sigma= 3 : lamda=
0.7853981633974483 : gamma= 0.5
Gabor29
Gabor29 : theta= 0.7853981633974483 : sigma= 3 : lamda=
1.5707963267948966 : gamma= 0.05
Gabor30
Gabor30 : theta= 0.7853981633974483 : sigma= 3 : lamda=
1.5707963267948966 : gamma= 0.5
Gabor31
Gabor31 : theta= 0.7853981633974483 : sigma= 3 : lamda=
2.356194490192345 : gamma= 0.05
Gabor32
Gabor32 : theta= 0.7853981633974483 : sigma= 3 : lamda=
2.356194490192345 : gamma= 0.5
```

Commentaire: Ce script est pour créer des filtres de type Gabor pour notre image. Il utilise différentes valeurs pour les paramètres theta, sigma, lambda et gamma.

À chaque itération, un filtre Gabor est créé à l'aide de la fonction cv2.getGaborKernel() de OpenCV, puis on l'applique sur notre image avec l'instruction cv2.filter2D(). Les résultats de ces filtres sont ensuite aplatis en un vecteur unidimensionnel et ajoutés en tant qu'attribute a notre tableau de donnée, avec des étiquettes distinctes pour chaque filtre Gabor.

Le numéro "num" est utilisé pour attribuer des étiquettes uniques à chaque filtre Gabor dans notre tableau de données. Le processus est accompagné d'impressions détaillant les valeurs des paramètres pour chaque filtre appliqué.

c) Application de differents filtre sur notre image

```
edges = cv2.Canny(img, 100,200)
edges1 = edges.reshape(-1)
df['Canny Edge'] = edges1


from skimage.filters import roberts, sobel, scharr, prewitt
edge_roberts = roberts(img)
edge_roberts1 = edge_roberts.reshape(-1)
df['Roberts'] = edge_roberts1
edge_sobel = sobel(img)
edge_sobel1 = edge_sobel.reshape(-1)
df['Sobel'] = edge_sobel1
edge_scharr = scharr(img)
edge_scharr1 = edge_scharr.reshape(-1)
df['Scharr'] = edge_scharr1
edge_prewitt = prewitt(img)
edge_prewitt1 = edge_prewitt.reshape(-1)
df['Prewitt'] = edge_prewitt1

from scipy import ndimage as nd
gaussian_img = nd.gaussian_filter(img, sigma=3)
gaussian_img1 = gaussian_img.reshape(-1)
df['Gaussian s3'] = gaussian_img1
gaussian_img2 = nd.gaussian_filter(img, sigma=7)
gaussian_img3 = gaussian_img2.reshape(-1)
df['Gaussian s7'] = gaussian_img3
median_img = nd.median_filter(img, size=3)
median_img1 = median_img.reshape(-1)
df['Median s3'] = median_img1
```

Commentaire: dans cette partie on applique diverses opérations de filtrage sur notre image, et stocker les résultats dans notre tableau de données "df" en utilisant la bibliothèque Pandas en Python.

Dans cette etape on a les données qui vont etre utiliser comme des entrées de notre model pour l'application de segmentation, maintenant il ne reste que d'ajouter une column de donnée labels (les pixels de l'image segmentée).

d) L'ajout des labels:

```
labeled_img = cv2.imread('Sandstone_Versa0000_mask.tif')
labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_BGR2GRAY)
labeled_img1 = labeled_img.reshape(-1)
df['Labels'] = labeled_img1
```

Commentaire : Ce code charge une image à partir du fichier 'Sandstone_Versa0000_mask.tif' en utilisant la fonction cv2.imread() d'OpenCV. Par la suite, l'image est transformée en niveaux de gris via cv2.cvtColor() en spécifiant cv2.COLOR_BGR2GRAY. Ensuite, l'image en niveaux de gris est transformée en un vecteur unidimensionnel à l'aide de la méthode reshape(-1). Enfin, ce vecteur est ajouté à notre tableau de donnée "df" sous la colonne "Labels".

Cette partie du code prend une image segmentée, la convertit en un vecteur unidimensionnel pour l'utiliser comme labels pour notre jeu de donnée.

e) La séparation de jeu donnée a des donnée pour l'entrainement et des données pour le test:

```
df.head()
```

```
   Original Imag  Gabor1  Gabor2  Gabor3  Gabor4  Gabor5  Gabor6
Gabor7  \
0              0       0       0       0       0       0       0
0
1              0       0       0       0       0       0       0
0
2              0       0       0       0       0       0       0
0
3              0       0       0       0       0       0       0
0
4              0       0       0       0       0       0       0
0

   Gabor8  Gabor9  ...  Gabor32  Canny Edge  Roberts  Sobel  Scharr
Prewitt  \
0       0       0  ...        0           0      0.0    0.0     0.0
0.0
1       0       0  ...        0           0      0.0    0.0     0.0
0.0
2       0       0  ...        0           0      0.0    0.0     0.0
0.0
3       0       0  ...        0           0      0.0    0.0     0.0
0.0
4       0       0  ...        0           0      0.0    0.0     0.0
0.0

   Gaussian s3  Gaussian s7  Median s3  Labels
0            0            0          0      29
1            0            0          0      29
2            0            0          0      29
3            0            0          0      29
4            0            0          0      29

[5 rows x 42 columns]
```

```
Y = df['Labels']
X = df.drop('Labels', axis=1)
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.2, random_state=42)
```

Commentaire: dans cette partie on a séparé le jeu de données a deux parties, la premiere pour l'entrainement "X_train" et la deuxieme partie pour le test "X_test". On a pris un pourcentage de 20% de jeu de données pour le test et le reste pour l'entrainement.

C) Entrainement du modele:

1) l'importation des bibliotheques

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators = 100, random_state = 42)
```

Ici on a importer la bibliotheque qui contient l'algorithme d'apprentissage "RandomForestClassifier". On a défini un nombre d'arbre de 100, et un parametre de génération de nombres aléatoires égale à 42.

n_estimators : C'est le nombre d'arbres de décision à construire dans la forêt aléatoire. Plus ce nombre est élevé, plus le modèle peut capturer de complexité, mais cela peut aussi augmenter le temps d'entraînement et la consommation de ressources. En règle générale, un nombre plus élevé d'estimateurs peut améliorer les performances du modèle, mais il convient de trouver un équilibre en fonction des données et du temps d'exécution.

random_state : Il s'agit d'une graine (seed) pour l'initialisation de la génération de nombres aléatoires dans l'algorithme. Fixer cette valeur garantit la reproductibilité des résultats. Utiliser la même valeur pour random_state permet de reproduire les mêmes résultats d'exécution à chaque fois que le modèle est entraîné.

2) L'entrainement du model

```
model.fit(X_train,Y_train)

RandomForestClassifier(random_state=42)
```

On a lancé l'apprentissage de model par cette ligne de code. on a utilisé les valeurs de X_train et Y_train comme des données pour l'apprentissage.

3) Le test du modele

```
model.score(X_test,Y_test)

0.9829886116844216
```

Commentaire: On a obtenu une précision de 98% sur les données de test, indiquant que notre modèle s'est bien entraîné avec notre jeu de données et les parametres choisis et il a appris à effectuer des prédictions précises sur des données qu'il n'a pas jamais vus.

4) Analyse de la précision de données d'entrainement et de test:

```
from sklearn import metrics

from sklearn import metrics

predictions_train = model.predict(X_train)
accuracy_train = metrics.accuracy_score(Y_train, predictions_train)
print("Précision sur les données d'entraînement :", accuracy_train)

# Prédiction sur les données de test
predictions_test = model.predict(X_test)
accuracy_test = metrics.accuracy_score(Y_test, predictions_test)
print("Précision sur les données de test :", accuracy_test)
```

```
Précision sur les données d'entraînement : 0.9999987743941524
Précision sur les données de test : 0.9829886116844216
```

Commentaire : On a un score de prédiction de 99.99% sur les données d'entraînement et de 98.29% sur les données de test. Notre modèle montre une excellente performance. Cela indique qu'il s'est bien adapté aux données d'entraînement et a réussi à généraliser son apprentissage pour effectuer des prédictions précises même sur des données de test qu'il n'avait jamais vu.

5) Analyse de la contribution des attributs:

```
features_list = list(X_train.columns)
features_imp = pd.Series(model.feature_importances_,index =
features_list).sort_values(ascending=False)
print(features_imp)
```

```
Median s3         1.570393e-01
Gaussian s3       1.388735e-01
Original Imag     1.226133e-01
Gabor4            9.898186e-02
Gaussian s7       6.947841e-02
Gabor3            6.044985e-02
Gabor6            5.710451e-02
Gabor5            5.107034e-02
Gabor23           4.249738e-02
Gabor7            3.041683e-02
Gabor24           2.915298e-02
Gabor12           2.869697e-02
Gabor11           2.153773e-02
Gabor21           1.730256e-02
Gabor8            1.121452e-02
Gabor29           1.062140e-02
```

```
Gabor31          8.964347e-03
Prewitt          8.152386e-03
Sobel            7.986599e-03
Scharr           7.595907e-03
Gabor20          5.524420e-03
Roberts          5.383112e-03
Gabor32          3.020886e-03
Gabor30          2.533671e-03
Canny Edge       1.417029e-03
Gabor22          1.230777e-03
Gabor28          7.368440e-04
Gabor27          3.671902e-04
Gabor14          1.891334e-05
Gabor13          1.351281e-05
Gabor15          2.171831e-06
Gabor16          4.924385e-07
Gabor19          2.610953e-07
Gabor26          0.000000e+00
Gabor25          0.000000e+00
Gabor17          0.000000e+00
Gabor18          0.000000e+00
Gabor10          0.000000e+00
Gabor2           0.000000e+00
Gabor9           0.000000e+00
Gabor1           0.000000e+00
dtype: float64
```

Commentaire Les valeurs obtenues, classées de manière descendante, représentent la contribution relative de chaque attribut à la prédiction du modèle. Les caractéristiques les plus importantes ont des valeurs plus élevées, indiquant qu'elles ont une plus grande influence sur les prédictions du modèle. Par exemple, 'Median s3', 'Gaussian s3' et 'Original Imag' sont parmi les caractéristiques les plus significatives pour la prédiction.

Les caractéristiques avec des valeurs proches de zéro ou nulles ont un influence négligeable sur les prédictions du modèle.

Note : L'analyse de la contribution des attributs permet d'identifier le taux d'influence des attributs sur les prédictions du modèle. On peut eliminer les attributs les moins significatifs, lors d'un nouvel entraînement sur d'autres images, comme ça on peut réduire le temps nécessaire à l'entraînement du modèle ainsi que la taille du jeu de données.

D) Sauvegarde du modèle

```
import pickle
filename = "sandstone_model"
pickle.dump(model, open(filename, 'wb'))
```

Commentaire : dans cette partie on a sauvegarder le modele et ses parametres dans un fichier "sandstone_model".

E) Test du modèle sur de nouvelles données

1) Telechargement de modele (chargement de modele)

```
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.predict(X)
```

Commentaire: Dans cette section, nous avons importé le modèle entraîné. Il est possible de conclure que la sauvegarde d'un modèle permet de le télécharger ultérieurement sur n'importe quelle machine ou dans n'importe quel programme, offrant ainsi une portabilité et une réutilisabilité du modèle sans perdre ses performances. Puis on a utilisé notre modele entrainé pour prédire les résultats de segmentation et avec notre image comme entrée.

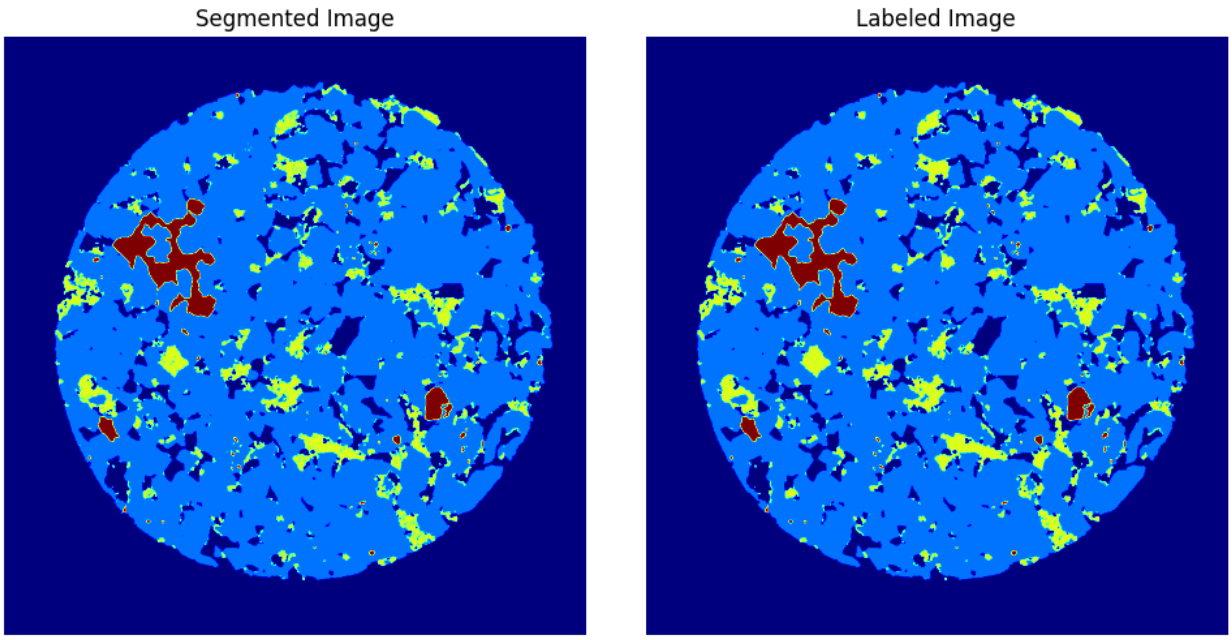
2) Redimensionnement de l'image

```
segmented = result.reshape((img.shape))
```

Commentaire: Dans cette partie, nous avons utilisé la fonction ".reshape" pour transformer le résultat obtenu de vecture unidimensionel à la dimension initiale de notre image.

3) Affichage de l'image segmentée:

```
from matplotlib import pyplot as plt

fig, axes = plt.subplots(1, 2, figsize=(10, 5))

axes[0].imshow(segmented, cmap='jet')
axes[0].set_title('Segmented Image')

axes[1].imshow(labeled_img, cmap='jet')
axes[1].set_title('Labeled Image')

for ax in axes:
    ax.axis('off')

plt.tight_layout()
plt.show()
```



Commentaire: ces deux figures représentent l'image prédite par notre modèle entraîné, appelée "Segmented Image", et l'image segmentée originale utilisée comme référence, nommée "Labeled Image". On peut constater qu'il n'y a pas de différence majeure entre les deux images, à l'exception de quelques détails très minimes. Cela confirme les résultats obtenus par notre modèle entraîné, et qui est très précis.

Test du modèle:

Dans cette partie, on va tester le modèle sur d'autre image que celui qu'on a utilisé pour l'entraînement. on va répéter toutes les étapes de filtrage et de préparation des données, puis conclure par une évaluation de notre modèle. Et comme ça on evalue la capacité de notre modele de génaraliser et de produire des prédictions sur des données totalement inconnues.

```
img = cv2.imread("Sandstone_Versa0050_train.tif")
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

from google.colab.patches import cv2_imshow
img2 = cv2.imread("Sandstone_Versa0050_train.tif",
cv2.IMREAD_UNCHANGED)
Img2 =img2.reshape(-1)

num = 1
kernels = []
for theta in range(2):
  theta = theta / 4. * np.pi
  for sigma in (1, 3):
    for lamda in np.arange(0, np.pi, np.pi / 4):
      for gamma in (0.05, 0.5):
```

```
        gabor_label = 'Gabor' + str(num)
        print(gabor_label)
        ksize=9
        kernel = cv2.getGaborKernel((ksize, ksize), sigma,
theta,lamda, gamma, 0, ktype=cv2.CV_32F)
        kernels.append(kernel)
        fimg = cv2.filter2D(img2, cv2.CV_8UC3, kernel)
        filtered_img = fimg.reshape(-1)
        df[gabor_label] = filtered_img
        print(gabor_label, ': theta=', theta, ': sigma=', sigma, ':
lamda=',lamda, ': gamma=', gamma)
        num += 1

edges = cv2.Canny(img, 100,200)
edges1 = edges.reshape(-1)
df['Canny Edge'] = edges1


from skimage.filters import roberts, sobel, scharr, prewitt
edge_roberts = roberts(img)
edge_roberts1 = edge_roberts.reshape(-1)
df['Roberts'] = edge_roberts1
edge_sobel = sobel(img)
edge_sobel1 = edge_sobel.reshape(-1)
df['Sobel'] = edge_sobel1
edge_scharr = scharr(img)
edge_scharr1 = edge_scharr.reshape(-1)
df['Scharr'] = edge_scharr1
edge_prewitt = prewitt(img)
edge_prewitt1 = edge_prewitt.reshape(-1)
df['Prewitt'] = edge_prewitt1

from scipy import ndimage as nd
gaussian_img = nd.gaussian_filter(img, sigma=3)
gaussian_img1 = gaussian_img.reshape(-1)
df['Gaussian s3'] = gaussian_img1
gaussian_img2 = nd.gaussian_filter(img, sigma=7)
gaussian_img3 = gaussian_img2.reshape(-1)
df['Gaussian s7'] = gaussian_img3
median_img = nd.median_filter(img, size=3)
median_img1 = median_img.reshape(-1)
df['Median s3'] = median_img1

labeled_img = cv2.imread('Sandstone_Versa0050_mask.tif')
labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_BGR2GRAY)
labeled_img1 = labeled_img.reshape(-1)
df['Labels'] = labeled_img1

Y = df['Labels']
X = df.drop('Labels', axis=1)
```

```
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.predict(X)
segmented = result.reshape((img.shape))

Gabor1
Gabor1 : theta= 0.0 : sigma= 1 : lamda= 0.0 : gamma= 0.05
Gabor2
Gabor2 : theta= 0.0 : sigma= 1 : lamda= 0.0 : gamma= 0.5
Gabor3
Gabor3 : theta= 0.0 : sigma= 1 : lamda= 0.7853981633974483 : gamma=
0.05
Gabor4
Gabor4 : theta= 0.0 : sigma= 1 : lamda= 0.7853981633974483 : gamma=
0.5
Gabor5
Gabor5 : theta= 0.0 : sigma= 1 : lamda= 1.5707963267948966 : gamma=
0.05
Gabor6
Gabor6 : theta= 0.0 : sigma= 1 : lamda= 1.5707963267948966 : gamma=
0.5
Gabor7
Gabor7 : theta= 0.0 : sigma= 1 : lamda= 2.356194490192345 : gamma=
0.05
Gabor8
Gabor8 : theta= 0.0 : sigma= 1 : lamda= 2.356194490192345 : gamma= 0.5
Gabor9
Gabor9 : theta= 0.0 : sigma= 3 : lamda= 0.0 : gamma= 0.05
Gabor10
Gabor10 : theta= 0.0 : sigma= 3 : lamda= 0.0 : gamma= 0.5
Gabor11
Gabor11 : theta= 0.0 : sigma= 3 : lamda= 0.7853981633974483 : gamma=
0.05
Gabor12
Gabor12 : theta= 0.0 : sigma= 3 : lamda= 0.7853981633974483 : gamma=
0.5
Gabor13
Gabor13 : theta= 0.0 : sigma= 3 : lamda= 1.5707963267948966 : gamma=
0.05
Gabor14
Gabor14 : theta= 0.0 : sigma= 3 : lamda= 1.5707963267948966 : gamma=
0.5
Gabor15
Gabor15 : theta= 0.0 : sigma= 3 : lamda= 2.356194490192345 : gamma=
0.05
Gabor16
Gabor16 : theta= 0.0 : sigma= 3 : lamda= 2.356194490192345 : gamma=
0.5
```

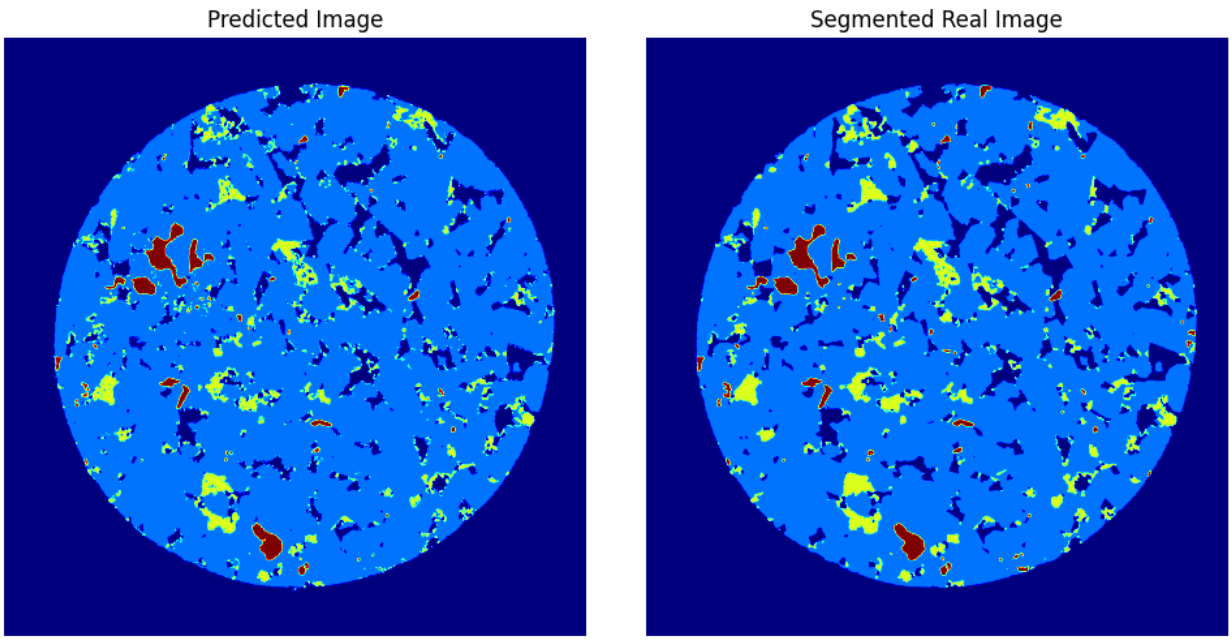
```
Gabor17
Gabor17 : theta= 0.7853981633974483 : sigma= 1 : lamda= 0.0 : gamma=
0.05
Gabor18
Gabor18 : theta= 0.7853981633974483 : sigma= 1 : lamda= 0.0 : gamma=
0.5
Gabor19
Gabor19 : theta= 0.7853981633974483 : sigma= 1 : lamda=
0.7853981633974483 : gamma= 0.05
Gabor20
Gabor20 : theta= 0.7853981633974483 : sigma= 1 : lamda=
0.7853981633974483 : gamma= 0.5
Gabor21
Gabor21 : theta= 0.7853981633974483 : sigma= 1 : lamda=
1.5707963267948966 : gamma= 0.05
Gabor22
Gabor22 : theta= 0.7853981633974483 : sigma= 1 : lamda=
1.5707963267948966 : gamma= 0.5
Gabor23
Gabor23 : theta= 0.7853981633974483 : sigma= 1 : lamda=
2.356194490192345 : gamma= 0.05
Gabor24
Gabor24 : theta= 0.7853981633974483 : sigma= 1 : lamda=
2.356194490192345 : gamma= 0.5
Gabor25
Gabor25 : theta= 0.7853981633974483 : sigma= 3 : lamda= 0.0 : gamma=
0.05
Gabor26
Gabor26 : theta= 0.7853981633974483 : sigma= 3 : lamda= 0.0 : gamma=
0.5
Gabor27
Gabor27 : theta= 0.7853981633974483 : sigma= 3 : lamda=
0.7853981633974483 : gamma= 0.05
Gabor28
Gabor28 : theta= 0.7853981633974483 : sigma= 3 : lamda=
0.7853981633974483 : gamma= 0.5
Gabor29
Gabor29 : theta= 0.7853981633974483 : sigma= 3 : lamda=
1.5707963267948966 : gamma= 0.05
Gabor30
Gabor30 : theta= 0.7853981633974483 : sigma= 3 : lamda=
1.5707963267948966 : gamma= 0.5
Gabor31
Gabor31 : theta= 0.7853981633974483 : sigma= 3 : lamda=
2.356194490192345 : gamma= 0.05
Gabor32
Gabor32 : theta= 0.7853981633974483 : sigma= 3 : lamda=
2.356194490192345 : gamma= 0.5
```

```
from sklearn import metrics

predictions_train = model.predict(X)
accuracy_train = metrics.accuracy_score(Y, predictions_train)
print("Précision de modele :", accuracy_train)
```

```
Précision de modele : 0.9787068194653614
```

Commentaire : Nous avons obtenu une précision de 97.87% sur une nouvelle image que le modèle n'avait jamais vue. En comparaison, dans le cas précédent, le modèle a été entraîné sur 80% des pixels de l'image, ne laissant que 20% pour l'évaluation du modèle. Ce résultat constitue une évaluation complète sur la totalité de l'image, démontrant ainsi la capacité du modèle à généraliser et à produire des prédictions précises même sur des données entièrement inconnues.

```
from matplotlib import pyplot as plt

fig, axes = plt.subplots(1, 2, figsize=(10, 5))

axes[0].imshow(segmented, cmap='jet')
axes[0].set_title('Predicted Image')

axes[1].imshow(labeled_img, cmap='jet')
axes[1].set_title('Segmented Real Image')

for ax in axes:
    ax.axis('off')

plt.tight_layout()
plt.show()
```



Commentaire : Les deux figures représentent l'image prédite par notre modèle entraîné, nommée "Predicted Image", et l'image segmentée réelle fournie, appelée "Segmented Real Image". On peut constater que les deux images sont identiques, à l'exception de quelques détails très minimes. Cela confirme que notre modèle est bien entraîné et généralisé, capable de produire des prédictions précises dans des domaines de la même catégorie (par exemple, dans notre cas, des images microscopiques).

Conclusion:

Ce travail présente un processus complet de segmentation d'images utilisant un modèle d'apprentissage qui s'appele RandomForestClassifier. On suivi les étapes suivantes:

- **Préparation des données :** Création et prépararion de notre jeu de donnée qui contient des pixels de notre image originale et les pixels des images filtrés
- **Entraînement du modèle :** Utilisation de l'algorithme RandomForestClassifier pour l'apprentissage. Et la séparation des données en ensembles d'entraînement et de test. Et finalement l'entraînement du modèle qui a montré des performances élevées.
- **Analyse post-entraînement :** Identification de l'importance des attributs, mettant en évidence les caractéristiques les plus influentes dans les prédictions du modèle.
- **Test sur de nouvelles données :** Le modèle entraîné a été utilisé pour prédire la segmentation d'une nouvelle image. La visualisation des images segmentées a montré une similarité élevée entre l'image prédite et l'image segmentée originale. et ça a confimé la generalisation de l'apprentissage de notre modele.

Ce travail démontre la capacité d'un modèle basé sur RandomForestClassifier à segmenter des images avec une précision élevée, en utilisant uniquement une seule image pour l'entraînement. Cette approche s'avère économique en termes de ressources et de temps de calcul, comparativement à d'autres méthodes d'apprentissage. La capacité du modèle à généraliser a été confirmée en le testant sur une nouvelle image, démontrant une performance cohérente avec les données d'entraînement.