Traitement d'images
TP 2 : Segmentation

Dans le domaine du traitement d'image, la segmentation d'une image est le processus qui consiste à diviser une image en plusieurs segments. L'objectif de la segmentation d'une image est de changer la représentation d'une image en quelque chose de plus significatif et de plus facile à analyser. Elle est généralement utilisée pour localiser des objets et créer des frontières.

Segmentation approche pixels

Seuillage

Dans cette partie, nous allons tester plusieurs techniques de seuillage d'image. Nous allons tester le seuillage simple (global), le seuillage adaptatif et le seuillage à seuil automatique.
Dans le cas du seuillage simple, la valeur globale du seuil est utilisée et reste constante tout au long du processus. Dans le cas du seuillage adaptatif, la valeur du seuil est calculée pour des régions locales plus petites, avec des valeurs de seuil différentes pour différentes régions en fonction du changement d'éclairage.
Dans le seuillage automatique (ici on va traiter la méthode de Otsu), la valeur du seuil n'est pas choisie mais déterminée automatiquement. Une image bimodale (deux valeurs d'image distinctes) est considérée. L'histogramme généré contient deux pics. Une condition générique consisterait donc à choisir une valeur seuil qui se situe au milieu des deux valeurs de crête de l'histogramme.

1. Réaliser un seuillage de l'image 'cameraman.tif' avec les trois techniques. Nous utiliserons :
   - La fonction ***cv2.threshold*** avec des valeurs de seuils choisis manuellement pour le seuillage simple.
   - La fonction ***cv2.adaptiveThreshold*** pour le seuillage adaptatif.
   - La fonction ***cv2.threshold*** avec un flag ***cv2.THRESH_OTSU***

Exercice :

Dans cet exercice, nous utilisons une image importée du module data de la bibliothèque Scikit image.

```
from skimage import data

image = data.astronaut()
plt.imshow(image)
plt.show()
```

Question : Générer une image pour chaque canal (rouge, vert, bleu), puis effectuer des segmentations avec les trois seuils suivant : [100, 150, 200]. Afficher le résultat.

K-moyennes

Dans cette partie, nous allons procéder à la segmentation d'une image à l'aide d'une méthode de regroupement appelée k-moyennes (k-Means).
Les algorithmes de regroupement sont des algorithmes non supervisés, ce qui signifie qu'il n'y a pas de données étiquetées disponibles. Ils sont utilisés pour identifier différentes classes ou groupes dans les données sur la base de leurs similarités. Les points de données d'un même groupe sont plus semblables aux autres points de données de ce même groupe qu'à ceux des autres groupes.
Le regroupement par K-moyennes est l'un des algorithmes de regroupement les plus couramment utilisés. Le k représente le nombre de groupes.

Le regroupement par K-moyennes fonctionne comme suit :

1. Choisissez le nombre de clusters que vous souhaitez trouver, soit k.
2. Assignez aléatoirement les points de données à l'une des k clusters.
3. Calculez ensuite le centre des clusters.
4. Calculez la distance entre les points de données et les centres de chaque cluster.
5. En fonction de la distance de chaque point de données par rapport au cluster, réaffectez les points de données aux clusters les plus proches.
6. Calculez à nouveau le nouveau centre du cluster.
7. Répétez les étapes 4, 5 et 6 jusqu'à ce que les points de données ne changent pas de cluster ou jusqu'à ce que le nombre d'itérations assigné soit atteint.

On commence par importer les bibliothèques nécessaires :

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

On choisit une image couleur et on procède à sa lecture :

```
image = cv2.imread(image_filename)
 image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
 plt.imshow(image)
```

Nous devons maintenant préparer les données pour l'application des moyennes K. L'image est une forme tridimensionnelle, mais pour appliquer le regroupement par k-means, nous devons la transformer en un tableau bidimensionnel.

```
pixel_vals = image.reshape((-1,3))
pixel_vals = np.float32(pixel_vals)
```

On fixe les paramètres pour notre algorithme de regroupement et on l'applique.
Les critères de l'arrêt de l'algorithme : soit 100 itérations max, soit la précision approche 85%.

```
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.85)
```

Il faut fixer le nombre clusters à chercher :

```
k = 3
```

Application du k-means sur nos données pixel_vals avec k clusters à trouver en utilisant les critères fixés ci-dessus et une initialisation aléatoire des centres :

```
retval, labels, centers = cv2.kmeans(pixel_vals, k, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)
```

Récupération des résultats :

```
centers = np.uint8(centers)
segmented_data = centers[labels.flatten()]
segmented_image = segmented_data.reshape((image.shape))
plt.imshow(segmented_image)
```

Segmentation approche contour

Définition de la détection des contours en traitement d'image

La détection des contours est une méthode utilisée dans le traitement des images pour déterminer les limites (contours) d'objets ou de zones dans une image. Les bords sont l'un des aspects les plus significatifs des images.

Un bord d'image est un changement local important dans l'intensité de l'image qui est souvent lié à une discontinuité dans l'intensité de l'image ou dans la dérivée première (gradient). Les discontinuités d'intensité de l'image peuvent être dues à des discontinuités de pas ou à des discontinuités de ligne.

- Les discontinuités de pas (en marche d'escalier se produisent lorsque l'intensité de l'image passe rapidement d'une valeur d'un côté de la discontinuité à une valeur différente de l'autre côté de la discontinuité.
- Les discontinuités linéaires se produisent lorsque l'intensité visuelle change rapidement mais revient à sa valeur initiale après une courte distance.

Toutefois, les bordures en escalier et en ligne sont rares sur les images du monde réel. Les discontinuités nettes dans les signaux réels sont rares en raison des composantes à basse fréquence ou du lissage introduit par la plupart des équipements de détection. Les contours en escalier deviennent des contours de rampe et les contours des lignes deviennent des contours de toit si les changements d'intensité se produisent sur une distance limitée plutôt qu'instantanément.

Techniques de détection des contours

L'opération consistant à reconnaître les changements locaux substantiels dans une image est connue sous le nom de détection des contours. Un contour en escalier dans une dimension correspond à un pic local dans la dérivée première. Le gradient est une mesure de changement de fonction et une image peut être considérée comme un tableau d'échantillons d'une fonction continue de l'intensité de l'image. Une approximation discrète du gradient peut être utilisée pour identifier des changements substantiels dans les niveaux de gris d'une image. Le gradient est défini comme le vecteur et est l'équivalent bidimensionnel de la dérivée première. Le gradient possède deux propriétés essentielles :

- Le vecteur pointe dans la direction du taux de croissance le plus élevé de la fonction des coordonnées.
- La magnitude du gradient est égale au taux le plus élevé d'augmentation de la fonction des coordonnées par unité de distance dans la direction du vecteur.

Toutefois, il est d'usage d'approximer l'intensité du gradient en utilisant des valeurs absolues. La direction du gradient est déterminée par l'analyse vectorielle comme l'angle mesuré par rapport à l'axe des x.

Trois opérateurs de détection des contours se basant sur la dérivée première : l'opérateur de Sobel, l'opérateur de Prewitt et l'opérateur de Robert.

Par ailleurs, ces détecteurs de contours calculent la dérivée première et supposent la présence d'un point de contour si elle est supérieure à un certain seuil. Par conséquent, un nombre excessif de points de contour est détecté. Une meilleure stratégie consisterait à découvrir et à examiner uniquement les endroits présentant des maxima locaux dans les valeurs de gradient.

Cela signifie qu'aux points de contour, la dérivée première présentera un pic et la dérivée seconde un passage à zéro. Les points de contour peuvent donc être identifiés en localisant les points de passage à zéro de la dérivée seconde de l'intensité de l'image.

La dérivée seconde est représentée par deux opérateurs bidimensionnels : le laplacien de Gauss et le détecteur de contours de Canny.

Implémentation sur Python

Nous allons travailler sur une image en niveaux de gris. Prenons par exemple l'image cameraman.tif.

La première chose à faire, comme toujours, est d'importer les bibliothèques nécessaires.

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
```

Ensuite, un prétraitement de l'image est nécessaire. On applique un filtre gaussien pour réduire le bruit dans l'image. Cependant, la réduction du bruit a un coût sur l'intensité des contours. Un filtrage plus poussé pour éliminer le bruit réduit l'intensité des contours.

```
original_img = cv2.imread('cameraman.tif')
blur_img = cv2.GaussianBlur(original_img,(3,3),0)
```

On peut par la suite afficher l'image

```
plt.figure(figsize=(10,10))
plt.imshow(blur_img)
plt.title("blured original image ")
plt.show()
```

Dans la suite de cet exercice, nous allons tester et comparer trois méthodes pour la détection des contours : le filtre Sobel, l'opérateur Canny et le détecteur Laplacien.

Ces techniques sont implémentées sous la bibliothèque OpenCV avec les fonctions : **cv2.Sobel(), cv2.Canny()** et **cv2.Laplacian().**

Appliquer ces techniques sur l'image cameraman.tif et afficher leurs résultats sur la même figure.
Utiliser l'erreur quadratique moyenne MSE pour comparer leurs performances.