Éleve: Rafael Senna Benatti
Discipline: IMA201(b)
TP Filtrage et restauration

1. Détection de contours

1.1. Filtre de gradient local par masque

- Rappelez l'intérêt du filtre de Sobel, par rapport au filtre différence, qui calcule une dérivée par la simple différence entre deux pixels voisins.

Le filtre de Sobel a une composante pondérée au centre de la matrice 3x3, alors, si nous avons une filtre de Sobel qui approxime le dérivé à l'horizontale, le noyau applique aussi un noyau pour avoir de blur avant la détection de contours.

- Est-il nécessaire de faire un filtre passe-bas de l'image avant d'utiliser le filtre de Sobel?

Non, le filtre passe-bas n'est pas nécessaire, parce que la composante centrale pondérée a déjà la fonction de faire un filtrage et atténuer le bruit.

- Le seuillage de la norme du gradient permet d'obtenir des contours. Commentez la qualité des contours obtenus (robustesse au bruit, continuité, épaisseur, position...) quand l'on fait varier ce seuil.

Alors, quand on change le seuil si on le réduit beaucoup c'est possible de voir beaucoup de bruit, malgré le bruit avec un petit seuil la continuité n'est pas faible, les contours sont plus épaisses et sont tous visibles.

Si le seuil est à un valeur appropriée, c'est possible de voir les conteurs bien définis, bonne épaisseur et sans bruit, avec bonne continuité. Mais si on engrand le seuil trop, on va perdre beaucoup de contours, les autres seront minces et faibles, mais il n'y aura pas de bruit.

1.2 Maximum du gradient filtré dans la direction du gradient

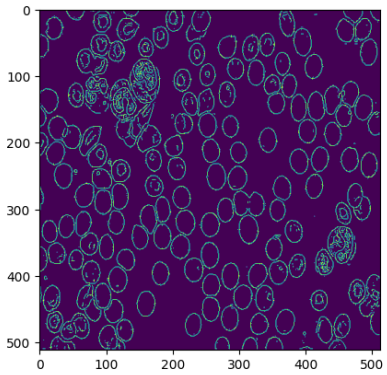
La fonction maximaDirectionGradient permet de déterminer les pixels de l'image qui sont des maxima du gradient dans la direction du gradient.

- Quel critère de qualité est optimisé par ce procédé ? Il est possible d'éliminer les contours dont la norme est inférieure à un seuil donné. Commentez les résultats obtenus en terme de position et de continuité des contours, et de robustesse au bruit en faisant varier ce seuil. Cherchez à fixer le seuil sur la norme de façon à obtenir un compromis entre robustesse au bruit et continuité des contours.

Ce procédé permet d'optimiser la robustesse au bruit puisque seulement les valeurs de maxima seront utilisés de façon que les petites gradient générés par le bruit ne seront pas pris en compte pour trouver les contours. Alors, quand on change le seuil si on le réduit beaucoup c'est possible de voir beaucoup de bruit, malgré le bruit avec un petit seuil la continuité n'est pas faible, les contours sont plus épaisses et sont tous visibles.

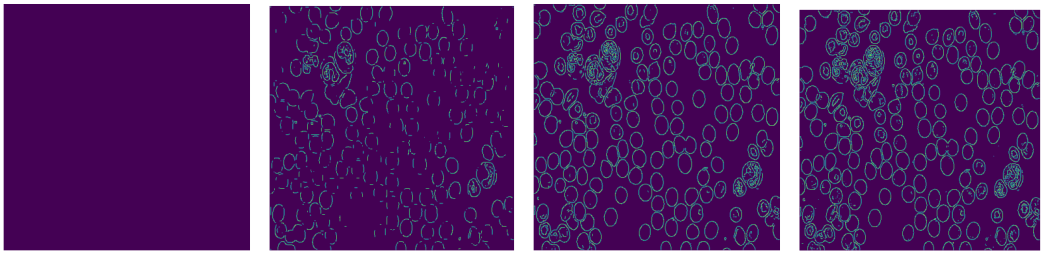
Si le seuil est à un valeur appropriée, c'est possible de voir les conteurs bien définis, bonne épaisseur et sans bruit, avec bonne continuité. Mais si on engrand le seuil trop, on va perdre beaucoup de contours, les autres seront minces et faibles, mais il n'y aura pas de bruit.

Le valeur de seuil sur la norme égal a 0.09 a abouti à un bon rapport entre la continuité des contours et la robustesse au bruit.



1.3 Filtre récursif de Deriche

- Testez la détection de contours avec ce filtre sur plusieurs images. Décrivez l'effet du paramètre α sur les résultats de la segmentation (faites varier ce paramètre sur l'intervalle 0, 3...3, 0).



On peut voir que le plus grand l'alpha, plus continus sont les contours de la segmentation et moins de robustesse au bruit, en créant faux contours.

- Le temps de calcul dépend-il de la valeur de α ? Expliquez pourquoi.

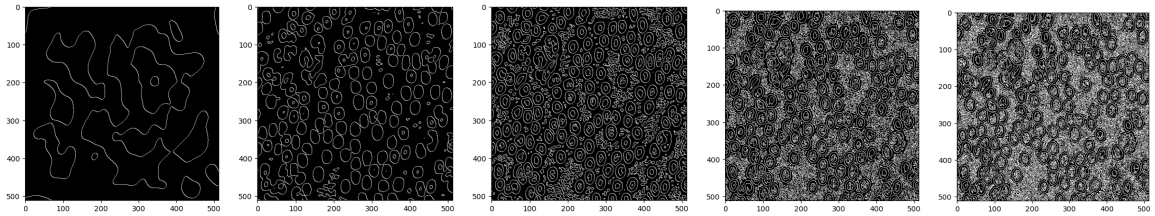
Non, le valeur de alpha est utilisé pour faire de calculs exponentiels, les algorithmes modernes utilisés pour faire des calculs exponentiels n'ont pas une relation direct avec le valeur d'alpha

- Comment et dans quel but les fonctions dericheSmoothX et dericheSmoothY sont-elles utilisées (cf. le filtre de Sobel).

Ces fonctions sont utilisées avant de calculer les gradients dans le but de faire une filtrage pour débruiter l'image.

1.4 Passage par zéro du Laplacien

- Quel est l'effet du paramètre α sur les résultats?



Le plus alpha accroît, moins la robustesse au bruit de la segmentation, mais le valeur d'alpha n'influence pas la continuité des contours. Un valeur très petite d'alpha nous donne de mauvais contours et plus grandes que prévu.

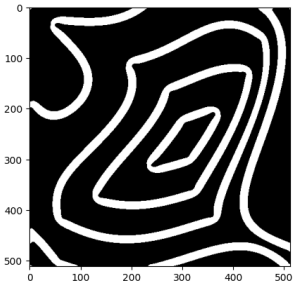
- Sur l'image cell.tif, quelles sont les principales différences par rapport aux résultats fournis par les opérateurs vus précédemment (contours, Deriche) ?

  La passage par zéro du Laplacien permet une plus grand continuité des contours, mais c'est possible de donner de mauvais formes, les autres peuvent offrir des descontinuités à ses formes, mais pas une forme très différente.

- Sur l'image pyramide.tif, comment est-il possible de supprimer les faux contours créés par cette approche ?

  C'est possible de le faire en utilisant le Laplacien de Difference. Dans le code c'est possible en changeant le valeur du threshold de posneg à un valeur égal ou plus grande que 1. Par exemple:

```
posneg=(lpima>=2)
```

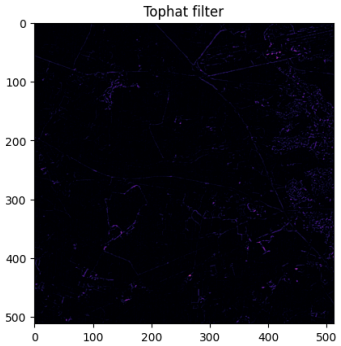


1.5 Changez d'image

Je choisirais l'opérateur de Deriche, parce que ce filtre ne change pas la forme si le valeur de alpha n'est pas bon, et cet operateur présente robustesse au bruit. C'est important parce que cette image a de bruit gaussien.

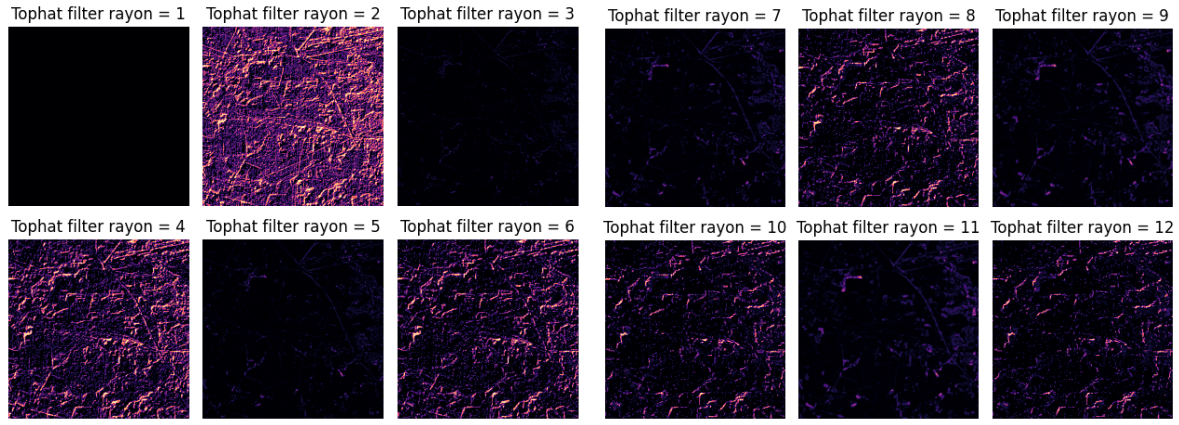
Pour le pré-traitement j'utiliserais une filtre linéaire, parce que ce filtre marche très bien pour le bruit gaussien. Pour le post-traitement j'utiliserais l'hystérésis pour assurer que les contours sont continus et tous fermés.

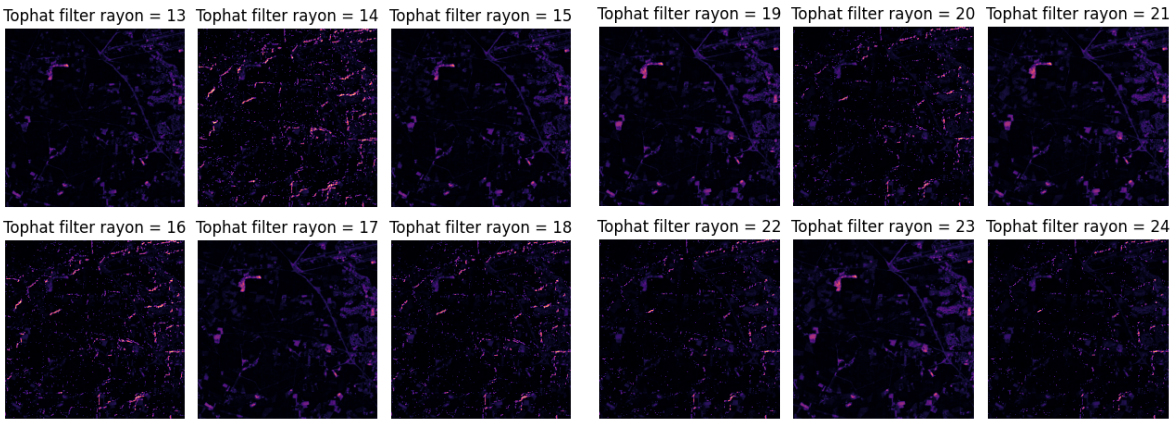
- Appliquez le filtre du Chapeau haut de forme (tophat) à une image SPOT pour effectuer une détection de lignes:



- Modifiez le rayon de l'élément structurant utilisé pour calculer le filtre tophat, et indiquez comment évoluent les lignes détectées.

```
# %%
rayons = [1, 2, 3, 4, 5, 6]
topList = []
counter = 0
for rayon in rayons:
    top = tophat(ima,rayon)
    topList.append(top)
    counter = counter + 1
fig, ax = plt.subplots(nrows=2, ncols=3)
chart = [ax[0, 0], ax[0, 1], ax[0, 2], ax[1, 0], ax[1, 1], ax[1, 2]]
counter = 0
for pos in chart:
    pos.imshow(topList[counter], cmap='magma')
    title = "Tophat filter rayon = " + str(rayons[counter])
    pos.set_title(title)
    for a in ax.ravel():
        a.axis('off')
    plt.tight_layout()
    counter = counter + 1
plt.show()
```

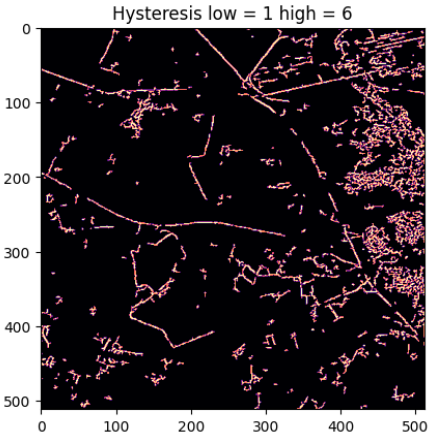




Par les images on peut voir que le valeur du rayon n'a pas une relation LINÉAIRE avec la qualité de détection des contours, par contre c'est possible voir que les valeurs petits et pairs ont petite robustesse au bruit. Les valeurs plus grandes permettent de voir que les lignes plus importantes

- Modifiez les valeurs des deux seuils, et examinez comment les lignes sont supprimées ou préservées. Quels sont les seuils qui donnent, à votre avis, le meilleur résultat?

  Plus les seuils accroît le plus de lignes sont supprimés. En testant expérimentalement les valeurs qui donnent les meilleurs résultats sont 1 pour le seuil bas et 6 pour le seuil haut.



3. Segmentation par classification : K-moyennes

3.1 Image à niveau de gris

- Testez l'algorithme des k-moyennes sur l'image cell.tif pour une classification en 2 classes. Cette classification segmente-t-elle correctement les diff´erents types de cellules ? Si non, que proposez-vous ?

Nous avons deux types de cellules, alors seulement deux classes n'est pas assez, parce qu'une classe serait utilisée pour le fond (parce que le fond est uniforme) et l'autre pour les cellules. Comme nous avons deux types de cellules pour bien détecter les contours il faut utiliser deux classes.

- Testez les différentes possibilités pour initialiser les classes. Décrivez si possible ces différentes méthodes.

- La classification obtenue est-elle stable (même position finale des centres des classes) avec une initialisation aléatoire ? Testez sur diff´erentes images `a niveaux de gris et diff´erents nombres de classes.
- Quelles sont les difficult´es rencontr´ees pour la segmentation des diff´erentes fibres musculaires dans l'image muscle.tif ?
- Expliquez pourquoi le filtrage de l'image originale (filtre de la moyenne ou filtre median) permet d'am´eliorer la classification.

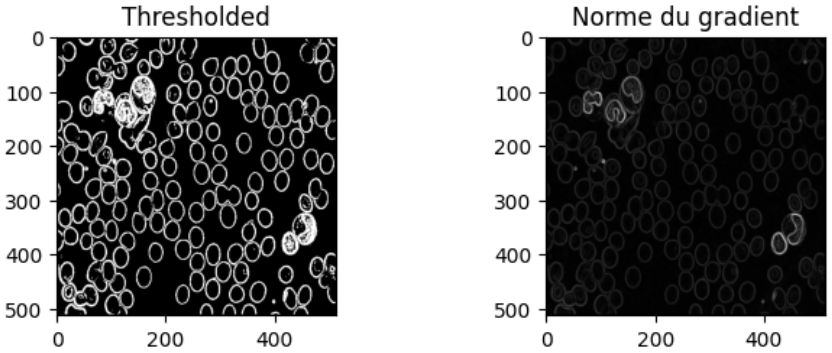
```
randNumber = np.random.randint(0, high=42)
image_array_sample= shuffle(image_array, random_state=randNumber)[:100]
kmeans=KMeans(n_clusters=n_class,random_state=randNumber, max_iter=10).fit(image_array_sample)
```

Nous pouvons utiliser une valeur aléatoire pour générer la seed du calcul du point central initial aléatoire, c'est bon pour créer des contours sans biais, par contre les résultats ne sont pas toujours pareil, quelquefois il y aura de petites différences aux contours. Si on décroît la taille de notre échantillon beaucoup plus, les résultats seront instables et c'est possible d'être un bon résultat ou même pas. En faisant ça, c'est possible réduire le temps que les opérations en prennent. C'est possible aussi de changer le numéro d'interactions, mais cela n'a eu aucun impact pertinent sur les résultats.

Pour l'image muscle.tif le bruit dans les fibres rend la segmentation plus difficile. De plus, il y a 3 types de cellules, et si on utilise 3 ou 4 classes le bruit est pris en compte comme une classe. Si on utilise plusieurs classes , l'image retourne à sa forme originale. Pour ces raisons, il est encore possible d'utiliser la méthode KMeans pour la segmentation si on utilise un filtre pour réduire le bruit.

4. Seuillage Automatique: Otsu

Dans ce script on cherche à réduire la variance intra-classe. Cette méthode a présenté de bons résultats dans plusieurs images en gris.
Si on applique le seuillage automatique de Otsu à une norme du gradient, la méthode retourne les contours entre les classes. Comme c'est possible à voir dans le image dessous:



- **Modifiez ls script otsu.py pour traiter le problème à trois classes, i.e. la recherche de deux seuils.**

```
def tri_class_otsu(image):
    t1 = otsu_thresh(image)
    print(t1)
    h=histogram(image)
```

```
    imageArray = np.uint32(image)
    above_t1 = imageArray[imageArray>=t1]
    below_t1 = imageArray[imageArray<t1]
    m1_lower = np.mean(below_t1)
    m1_upper = np.mean(above_t1)
    TBD = imageArray[(imageArray>= m1_lower) & (imageArray <= m1_upper)]
    t2, m2_lower, m2_upper, TBD = second_iteration_tri_class_otsu(image, m1_lower, m1_upper, TBD)
    minimunDelta = 0.01 # Stop condition: difference between thresholds
    deltaThresholds = t2 - t1
    t = []
    m_lower = []
    m_upper = []
    t.append(t2)
    m_lower.append(m2_lower)
    m_upper.append(m2_upper)
    x = 0
    while (abs(deltaThresholds) > minimunDelta):
        t_temp, m_lower_temp, m_upper_temp, TBD = second_iteration_tri_class_otsu(image, m_lower[x],
m_upper[x], TBD)
        t.append(t_temp)
        m_lower.append(m_lower_temp)
        m_upper.append(m_upper_temp)
        x = x + 1
        deltaThresholds = t[x] - t[x-1]
    return t[x]
def second_iteration_tri_class_otsu(im, m1_lower, m1_upper, TBD):
    h=histogram(im)
    m1_lower = int(m1_lower)
    m1_upper = int(m1_upper)

    m=0
    for i in range(256):
        m=m+i*h[i]

    maxt=0
    maxk=0
    for t in range(m1_lower, m1_upper):
        w2_lower=0
        w2_upper=0
        m2_lower=0
        m2_upper=0
        for i in range(t):
            w2_lower=w2_lower+h[i]
            m2_lower=m2_lower+i*h[i]
        if w2_lower > 0:
            m2_lower=m2_lower/w2_lower

        for i in range(t,256):
            w2_upper=w2_upper+h[i]
            m2_upper=m2_upper+i*h[i]
        if w2_upper > 0:
            m2_upper=m2_upper/w2_upper

        k=w2_lower*w2_upper*(m2_lower-m2_upper)*(m2_lower-m2_upper)
        if k > maxk:
            maxk=k
            maxt=t


    thresh=maxt
```

```
    m2_lower = np.mean(TBD[TBD<=thresh])
    m2_upper = np.mean(TBD[TBD>=thresh])
    TBD = TBD[(TBD>=m2_lower) & (TBD <=m2_upper)]
    print(f"n2 = {thresh}")
    return(thresh, m2_lower, w2_lower, TBD)
```

5. Croissance de régions

Dans le code on peut voir la partie responsable pour sélectionner les pixels qui seront ajoutés au objet.

```
            if np.abs(m0-m) < thresh * s0 :
                mask[y][x]=255
                modif=1
```

La condition ci-dessus est le prédicat.

Cela signifie que la multiplication du écart-type par le seuil doit être plus grand différence absolue entre la moyenne de l'image de l'objet existant et la région du pixel.

Si le seuil est très grand, l'objet ressemblera à l'image complète. Pour bien segmenter la matière blanche il faudra augmenter le seuil.

La seuillage d'Otsu donne un résultat proche de la croissance de région.