1. Détection de Contours

1.1. Filtre de Gradient Local par Masque

Le filtre **sobel** fait un gradient directionnelle, alors il va être plus sensible aux changements intenses dans l'image, par contraire le filtre différence est moins sensible à des changements intenses

Oui, il será nécessaire parce que sans filtre, on aura beaucoup d'information, que sera inutile quand on va détecter les bordes, et cette information va générer de bruit

Quand on varie le seuil, on va obtenir des contours plus gros et continus quand le valeur est plus petit, et on va obtenir des petits contours et moin continuité quand on augmente le seuil, cependant quand est plus grand le valeur, il y aura plus de bruit dans les contours

1.2. Maximum du gradient filtré

Le critère de qualité est optimisé pour identifier les points qui sont les contours, clairement, mais il le fait à travers de l'optimisation linéaire, en cherchant les valeurs du gradient dans les différentes directions.

Oui, avec une modification du mode on peut élire le seuil minimum que le méthode va reconnaître comme un contour,

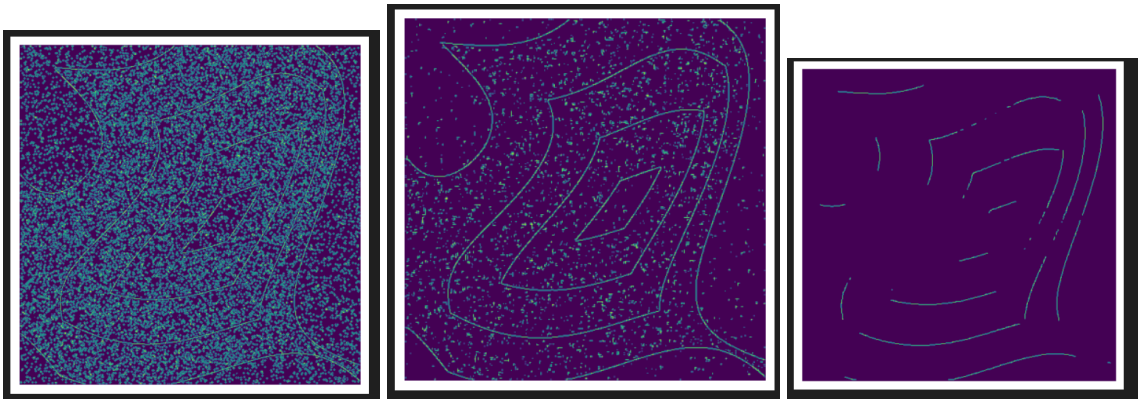
Avec 0.1, on trouve le meilleur résultat pour le rapport parmi bruit et continuité

1.3. FIltre Récursif de Deriche

```
def deriche GradY(ima,alpha):

    nl,nc=ima.shape
    ae=math.exp(-alpha)
    c=-(1-ae)*(1-ae)/ae

    b1=np.zeros(nl)
    b2=np.zeros(nl)

    grady=np.zeros((nl,nc))

    for i in range(nc):

        l=ima[:,i].copy()

        for j in range(2,nl):
            b1[j] = l[j-1] + 2*ae*b1[j-1] - ae*ae*b1[j-2]
        b1[0]=b1[2]
```

```
        b1[1]=b1[2]

        for j in range(nl-3,-1,-1):
            b2[j] = l[j+1] + 2*ae*b1[j+1] - ae*ae*b1[j+2]

        b2[nl-1]=b2[nl-3]
        b2[nl-2]=b2[nl-3]


        grady[:,i]=c*ae*(b1-b2);


    return grady
```

Dans les images on peut noter que pendant on réduit alpha, les contours auront moins de bruit, et les contours, arriveront dans un point ou il ne serions mieux, dans 0.3 les contours sont non continus, et à 3 ils sont pas visibles avec le bruit



Le temps de calcule ne dépend pas de alpha car la complexité est O(n²) peu importe le valeur d'alpha

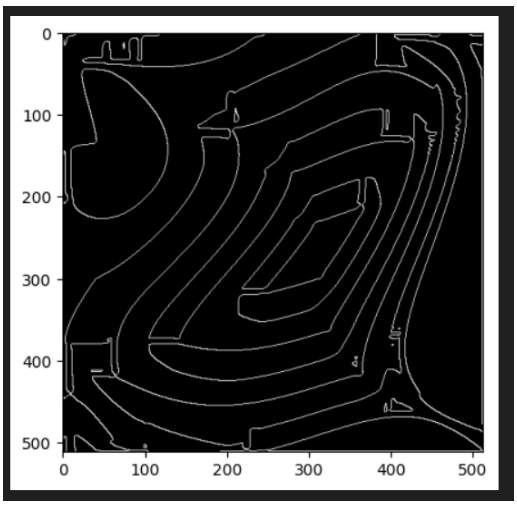
Les fonctions **dericheSmoothX** et **deriche SmoothY** sont utilisées pour appliquer un effet similaire que le filtre passe bas dans le filtre **sobel**

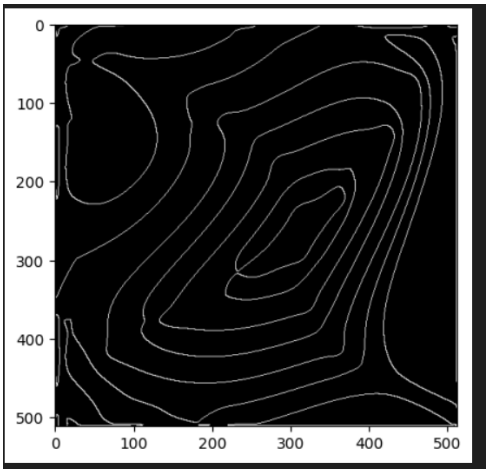
1.4. Passage par zéro du laplacien

Alpha change les résultats des contours, de manière que si on l'augmente, il va réduire les contours dans l'image, par contraire si on réduit alpha, l'image aura plusieurs de contours qui ne sont pas réelles

Le principal différence parmi le filtre laplacien et les filtres deriche et sobel est que avec le filtre laplacien les contours sont plus carrées, et sont fermes, avec les autres; les contours sont seulement des lignes, que peuvent être ou non continus

Si on applique un filtre gaussien, on aura des meilleures contours, parce que l'image va réduire les petits détails, et les petits contours, alors, il va montrer seulement les contours importants dans l'image





1.5. Changez d'image

Comme c'est une image avec bruit je choisirai un filtre sobel
Pré-traitements (filtre gaussien)
Post-traitements ()

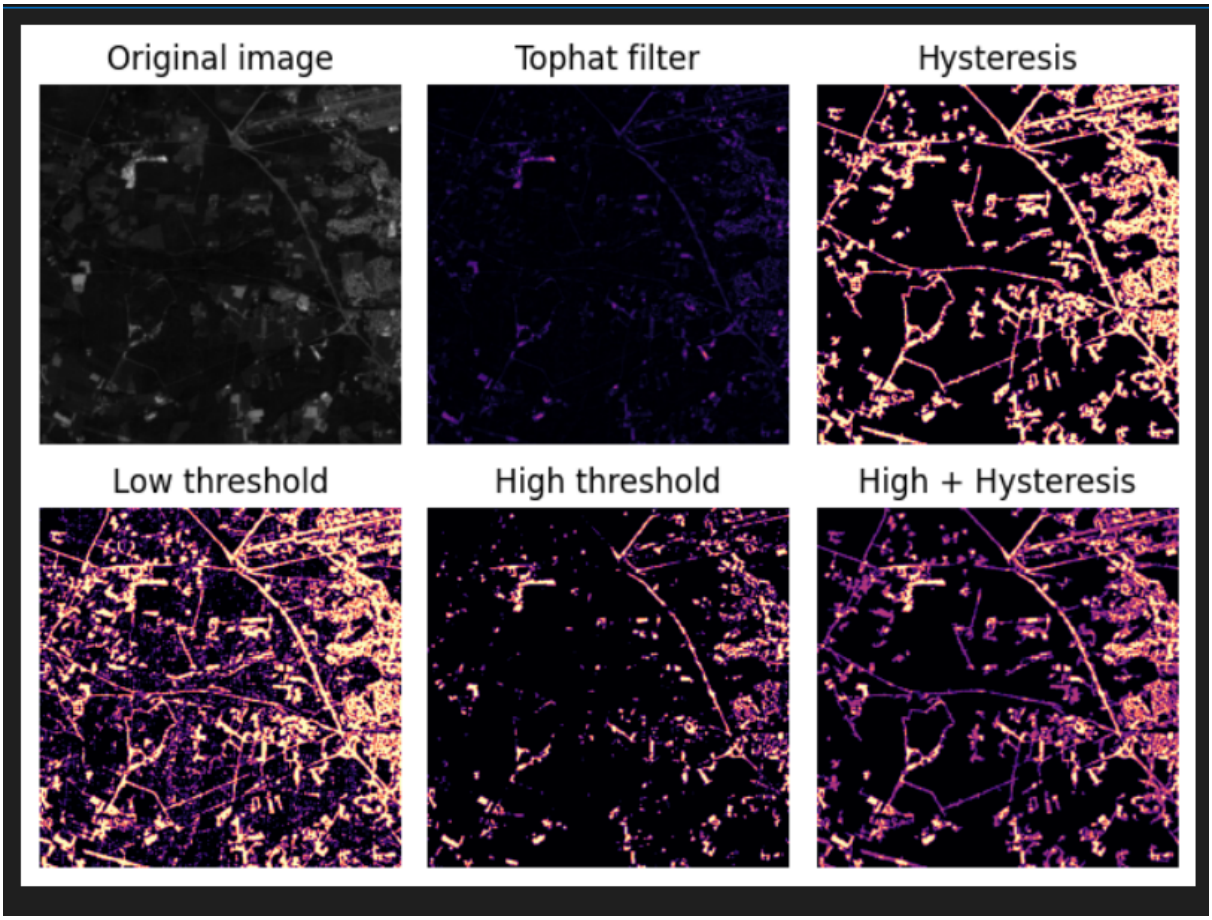
2. Seuillage avec hystérésis

2.1. Application à la détection de lignes

Après la modification de le rayon on peut noter que si on l'augmente les lignes détectées seront plus proches de las lignes réelles, et si on diminue le rayon, il aura moins de détails dans l'identification des lignes

Aprés plusieurs essais, le meilleur option est utiliser

```
low = 2
high = 9
```

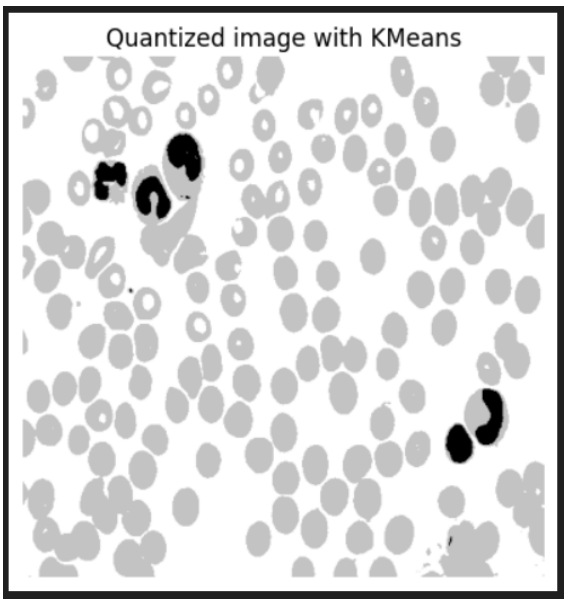


On peut noter que avec l'application d'un opérateur sobel on a des meilleures résultats dans l'identification des lignes

3. Segmentation par classification : K-moyennes

3.1. Image à niveaux de gris

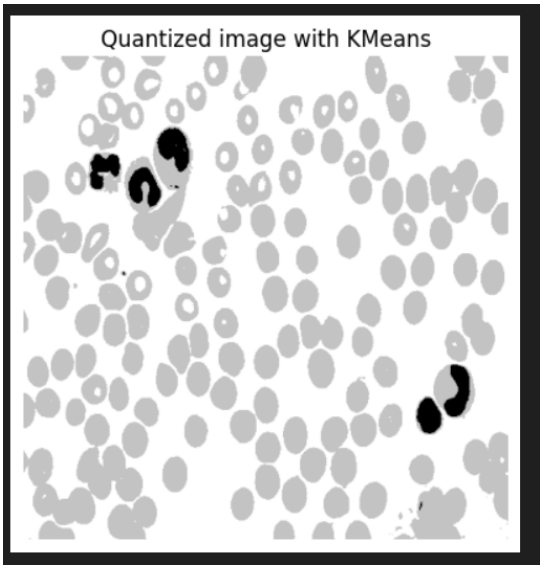
Avec 2 classes on n'a un bon classification des cellules, avec 3, on a l'arrière-plan, les cellules clairs, et les cellules un peu plus grises

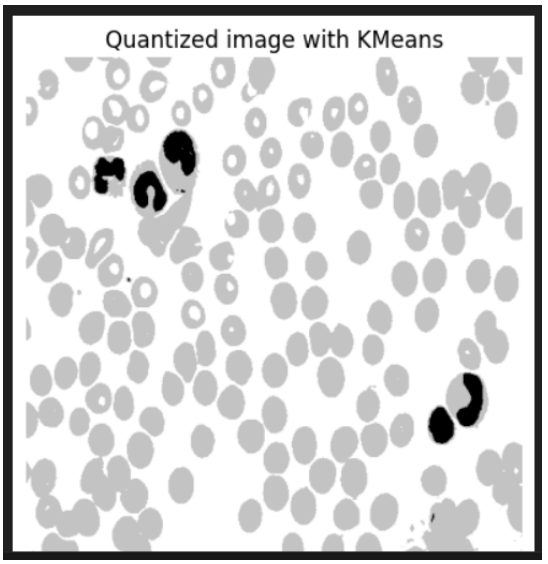


Pour calculer le número de classes, on a différentes options :

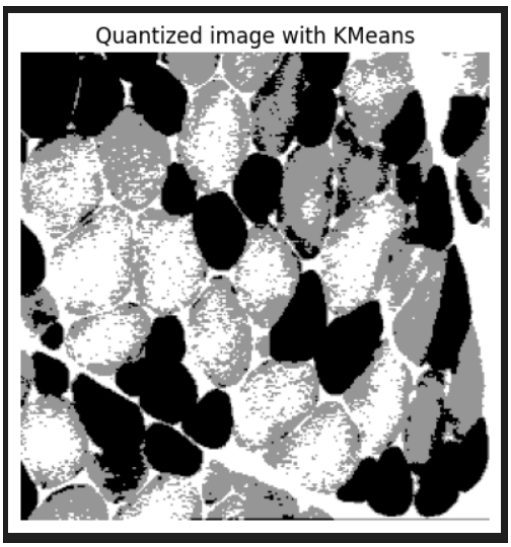
- K Means ++ : Il fait un calcule de probabilité pour sélectionner les meilleures centres dans la classification
- random : Les centres sont définis par définition aléatoire
- ndarray : on peut définir les centres

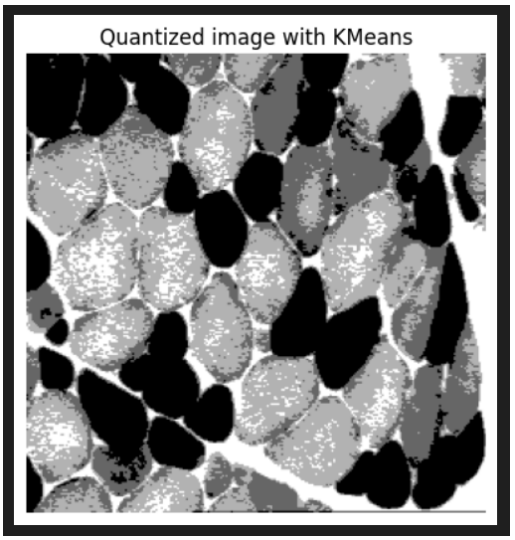
Le résultat avec l'initialisation random et un petit número de classes est stable, avec plus des classes est moins stable, c,est le même logique pour l'initialisation avec k means ++





La principal difficulté pour avoir un bon segmentation avec cette image est que l’image a des points dans le muscle qui sont presque le même couleur que l'arrière-plan, alors la segmentation será comme si l'arrière plan et le muscle soit la meme class

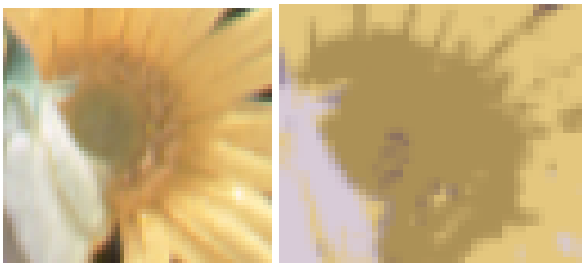




Si on applique un filtre moyenne, on aura meilleures résultats parce dans les fibres de le muscle qui n'ont pas le même couleur, avec le filtre moyenne, se réduira cette différence, et le muscle será plus défini, ainsi dans la classification k means on aura aussi cette segmentation

3.2. Image en couleur

On peut noter dans l'image original et l'image segmentee, on perd des details de coleur, par example dans le centre de le fleur jaune, on voit que dans l'image original, elle a 3 ou 4 differents couleur jaunes, dans la segmentation, on a une seul coleur, alors en general on perd detail de coleur



Avec 10, on a un bon résultat, cependant avec 64 couleurs, on a un très bon similarité, on peut optimiser selon la nécessité

On peut appliquer un filtre pour retirer les rayons et les points, après obtenir la segmentation avec 2 ou 3 classes

4. **Seuillage Automatique : Otsu**

Le méthode est optimisée pour trouver la major variance parmi les deux classes de pixeles (arrière-plan et objet)

Le méthode est util quand on a seulement un objet, quand on doit diviser parmi 3 (deux objets et le fond) est moins efficace

Oui; le méthode peut seuiller correctement une image norme du gradient, car l'image aura une distribution qui généralement permettra d'avoir la distinction claire parmi les objets ci sont bordés ou pas bordés

```
def otsu_thresh_3(im):

    h=histogram(im)

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

    maxt1=0
    maxt2=0
    maxk=0


    for t1 in range(256):
        for t2 in range(256):
            w0=0
            w1=0
            w2=0

            m0=0
            m1=0
            m2=0

            for i in range(t1):
                w0=w0+h[i]
                m0=m0+i*h[i]
            if w0 > 0:
                m0=m0/w0

            for i in range(t1,t2):
```

```
            w1=w1+h[i]
            m1=m1+i*h[i]
        if w1 > 0:
            m1=m1/w1

        for i in range(t1,256):
            w2=w2+h[i]
            m2=m2+i*h[i]
        if w2 > 0:
            m2=m2/w2

        k=w0*w1*((m0-m1)*(m0-m1))+ w1*w2*((m1-m2)*(m1-m2))

        if k > maxk:
            maxk=k
            maxt1=t1
            maxt2=t2


    return(maxt1,maxt2)
```

5. Croissance de régions

Pour ajouter un pixel à l'objet existant on doit vérifier que la différence parmi la moyen initial et la moyen du pixel est mineur que le seuil défini,

Si on augmente le seuil, on aura plus des pixels dans l'objet, on peut générer des bordes plus continus

On doit changer les paramètres thresh, rayon et (x0,y0), pour des bons résultats on a `thresh=6 , rayon=3 et y0=300 , x0=300`

On peut segmenter la matière grise si on mettre le pixel initial dans le partie de matière gris

Le prédicat dans ce script

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

Si on a une différence absolue parmi les deux moyennes mineur au seuil multiplié par la déviation de le premier pixel on aura une partie de l' objet cherché