TP 03 IMA

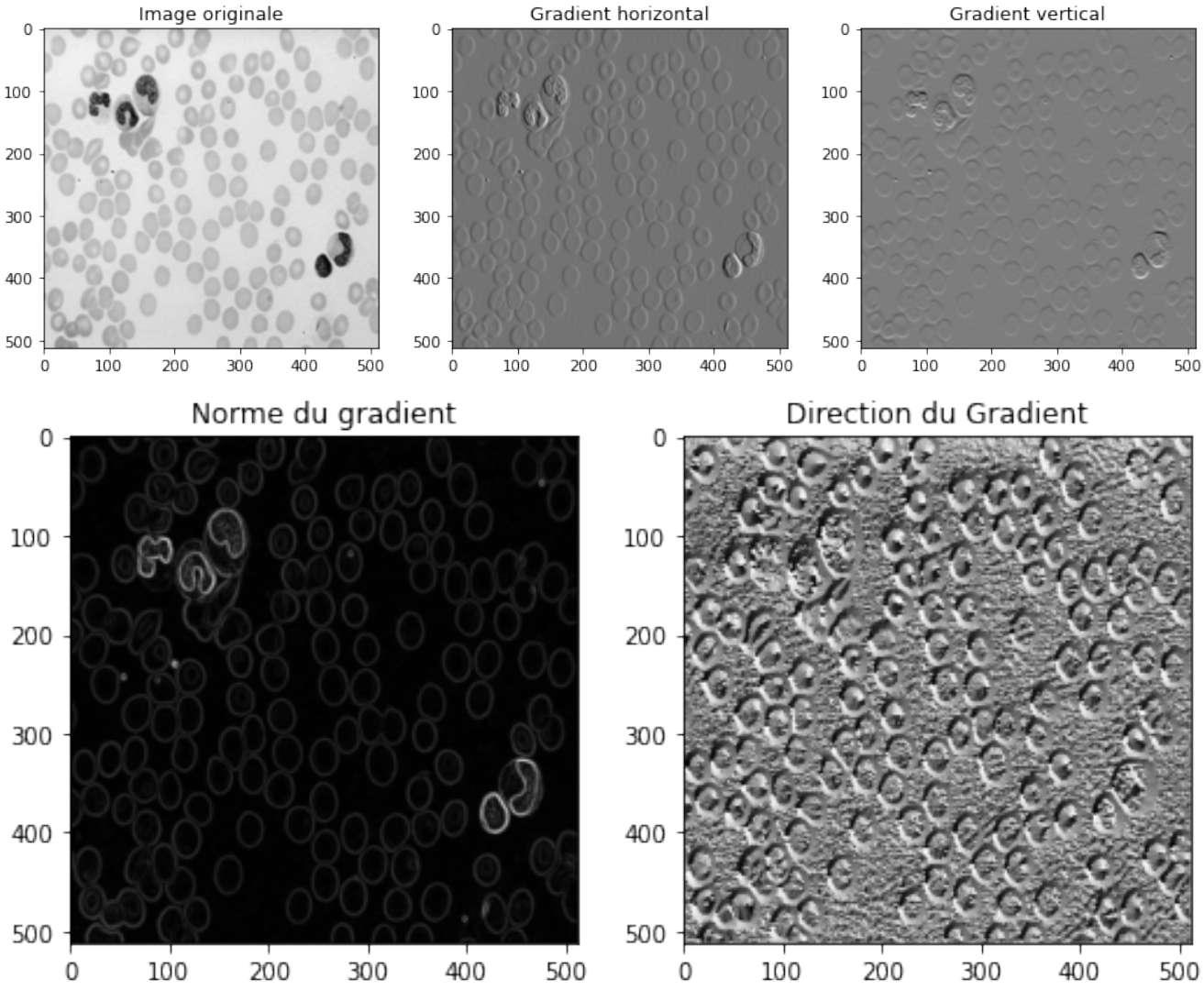
Luiz Augusto Facury de Souza

October 10, 2022

1 Détection de contours

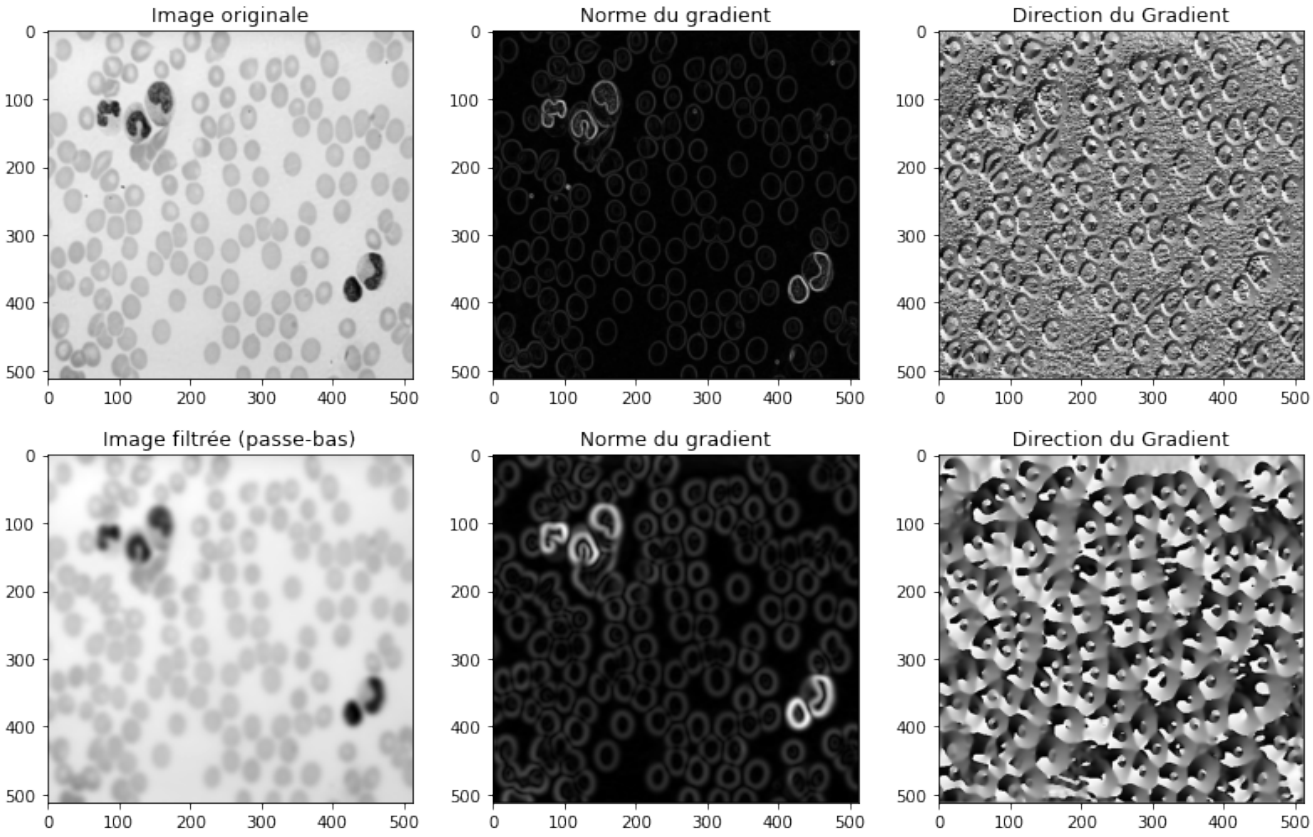
1.1 Filtre de gradient local par masque

Les images suivantes ont été générées sur la base du script sobel.py:

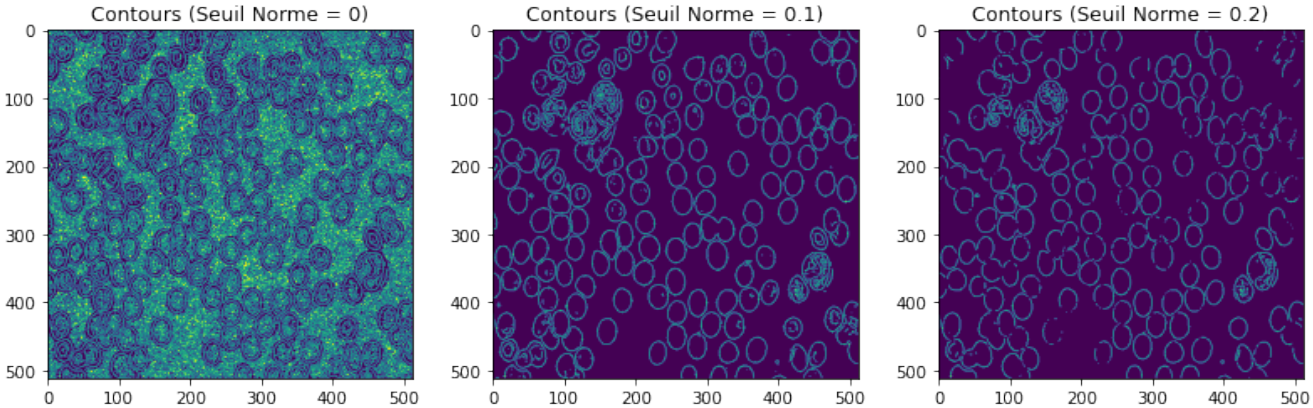


- Le filtre Sobel fait le calcul du gradiente dans une direction et effectue un lissage dans la direction orthoganale. De cette façon, le masque est moins sensible au bruit.

- Il n'est pas nécessaire d'utiliser un filtre passe-bas avant d'utiliser le filtre de Sobel, car il applique déjà un filtrage passe-bas dans l'image. Cependant, il est possible d'utiliser un filtre gaussien avant Sobel, de manière à ce que l'image soit plus lisse. Si l'image a beaucoup de bruit, il est possible d'utiliser le filtre gaussien, cependant la photo sera plus floutée, donc le contour sera moins défini, ce qui n'est pas souhaitable.



- C'est possible d'observer que le seuil norme en 0 produire une image avec un contour plus indiscernable que le seuil norme en 0.1 et 0.2. La norme en 0.1 et 0.2 produire contours continus et plus mince, toutefois, en 0.2 une partie des contours est effacée. Par conséquent, le norme en 0.1 est le meilleur résultat



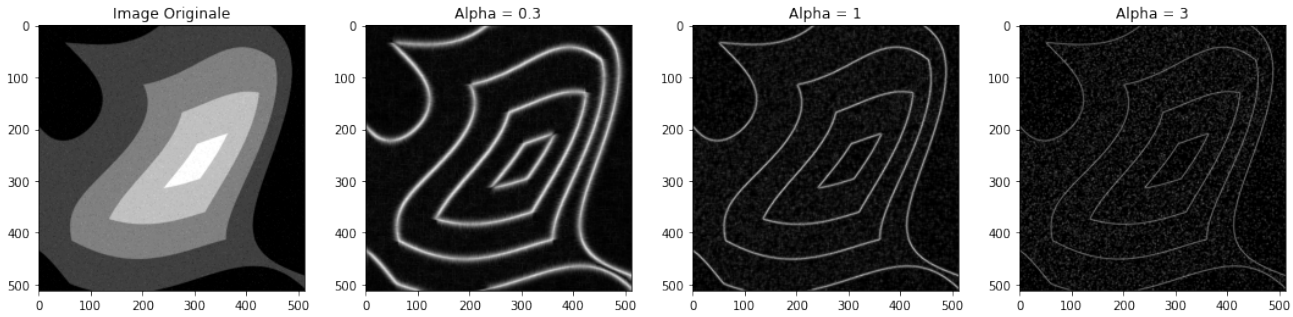
1.2 Maximum du gradient filtré dans la direction du gradient

- Le critère optimisé par la fonction est l'interpolation linéaire entre la normale et le gradient, générant un meilleur contour.
- Si le seuil est augmenté, le contour peut ne pas être continu, c'est-à-dire que la détection n'obtient pas le contour complet de la bobine. Par contre, si le seuil est diminué, la détection peut capter un bruit qui ne fait pas vraiment partie du contour.
- Le seuil en 0.1 offre un bon compromis entre la continuité des contours et la robustesse au bruit.

1.3 Filtre récursif de Deriche

```
def dericheGradY(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]   # LIGNE A MODIFIER
    b1[0]=b1[2]
    b1[1]=b1[2]

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

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


return grady
```

- C'est possible d'observer que, quand α est augmenté, l'algorithme est plus sensible pour la détection du contour, c'est-à-dire que il est plus sensible au bruit. D'autre part, quand α est diminué, les bordes sont plus fort et lisse

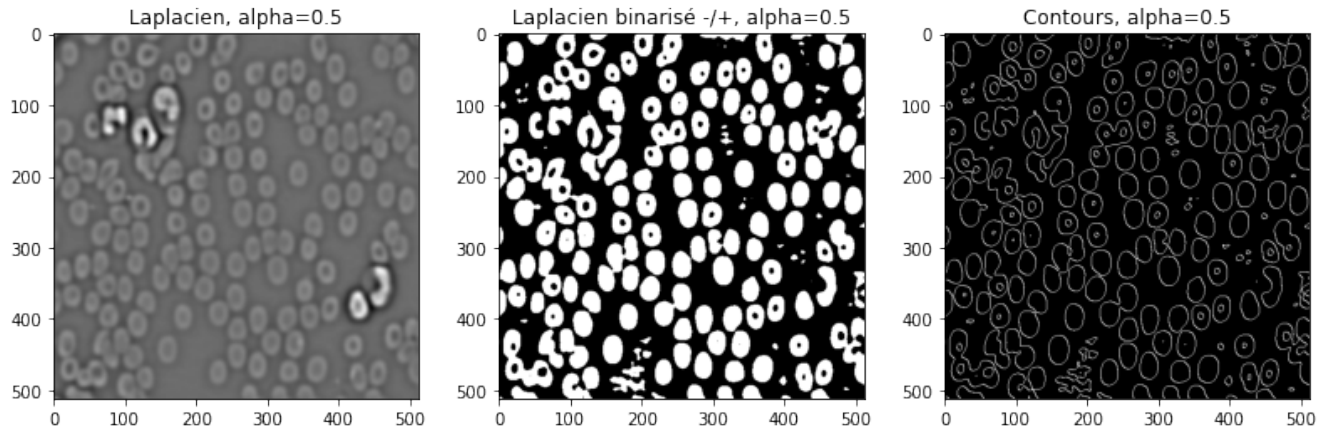


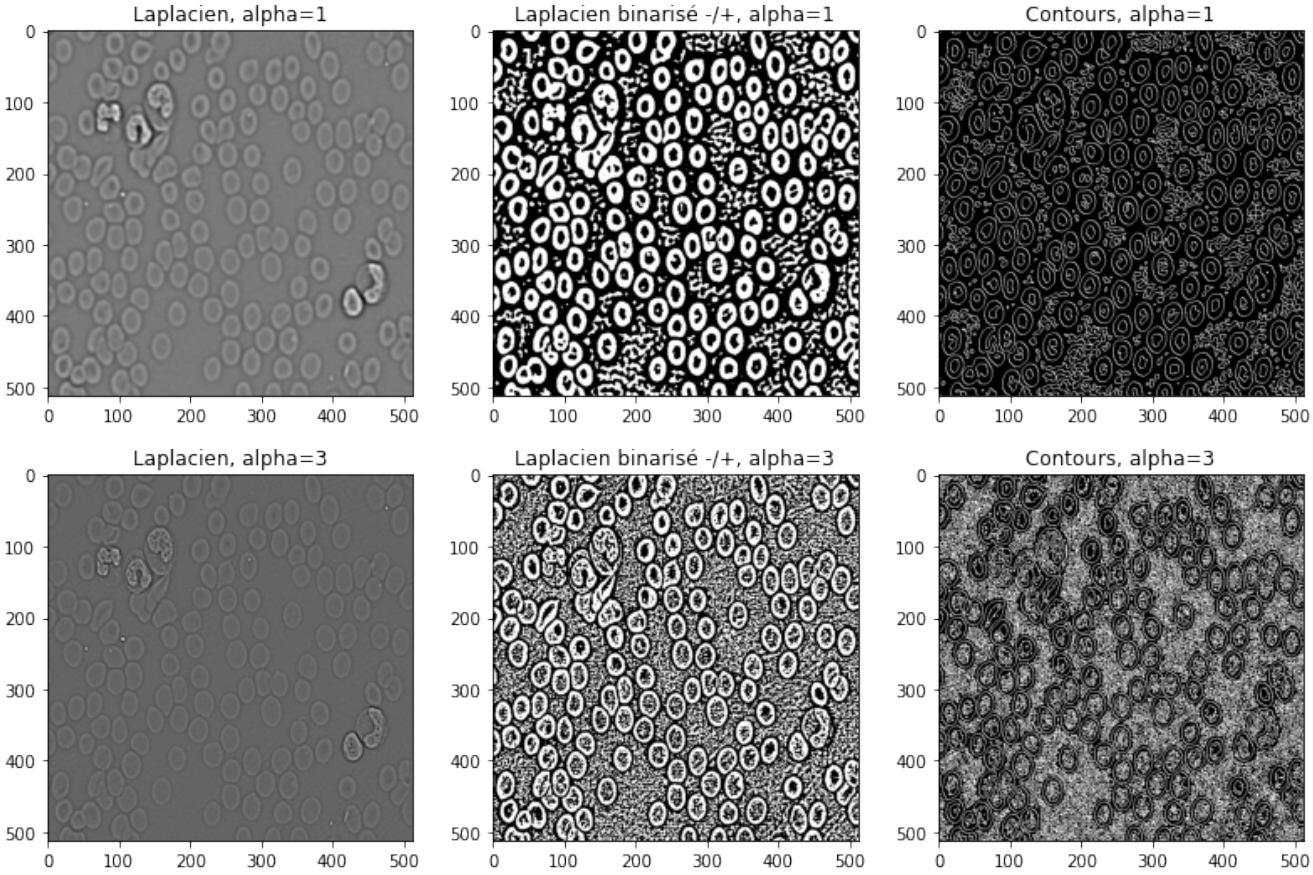
- Le temps de calcul ne dépend pas de la valeur de α, puisque la complexité de l'algorithme est donné par le número de lignes fois le número de columens de l'image, donc O(n^2). Le paramètre α est seulement un paramètre de l'exponentiel, O(1).

- Les fonctions dericheSmoothX et dericheSmoothY appliquent un filtre passe-bas dans l'un des axes.

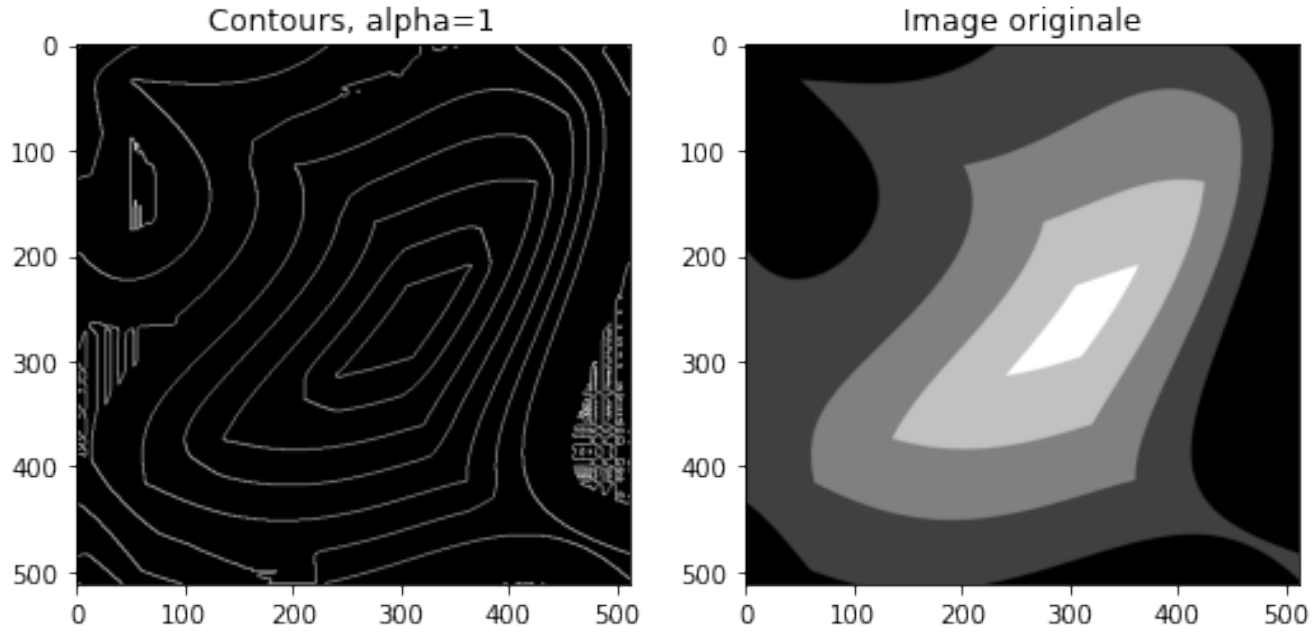
1.4 Passage par zéro du laplacien

- Un grand α donne un contour avec trop de bruit, alors qu'un petit α peut ne capter pas les contours corrects.





- L'opérateur Laplacien produit normalement un contour avec des lignes fermées, tandis que les autres opérateurs peuvent renvoyer un contour avec des lignes plus ouvertes.
- Il est possible de corriger les contours faux avec l'application d'un filtre gaussien avant l'opérateur.



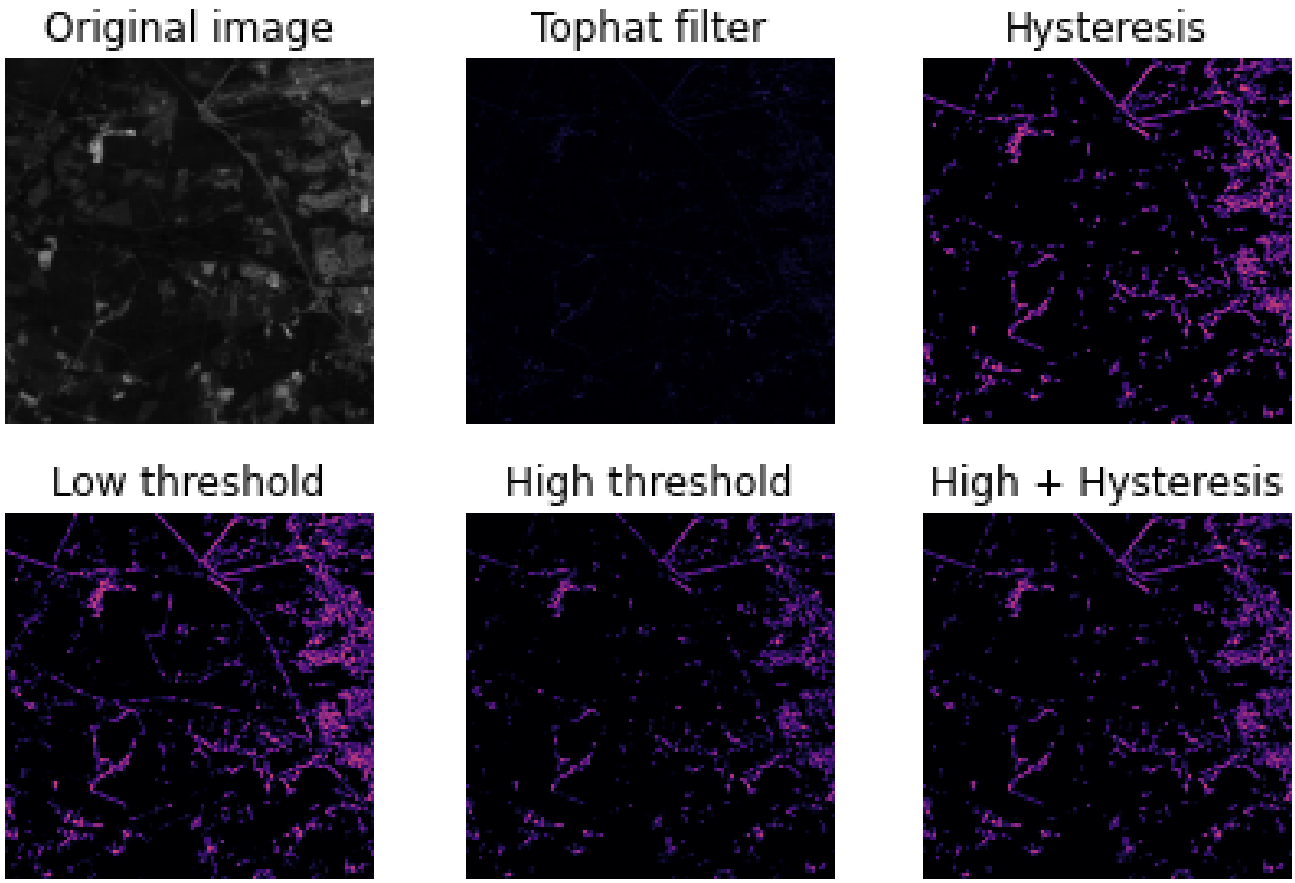
1.5 Changez d'image

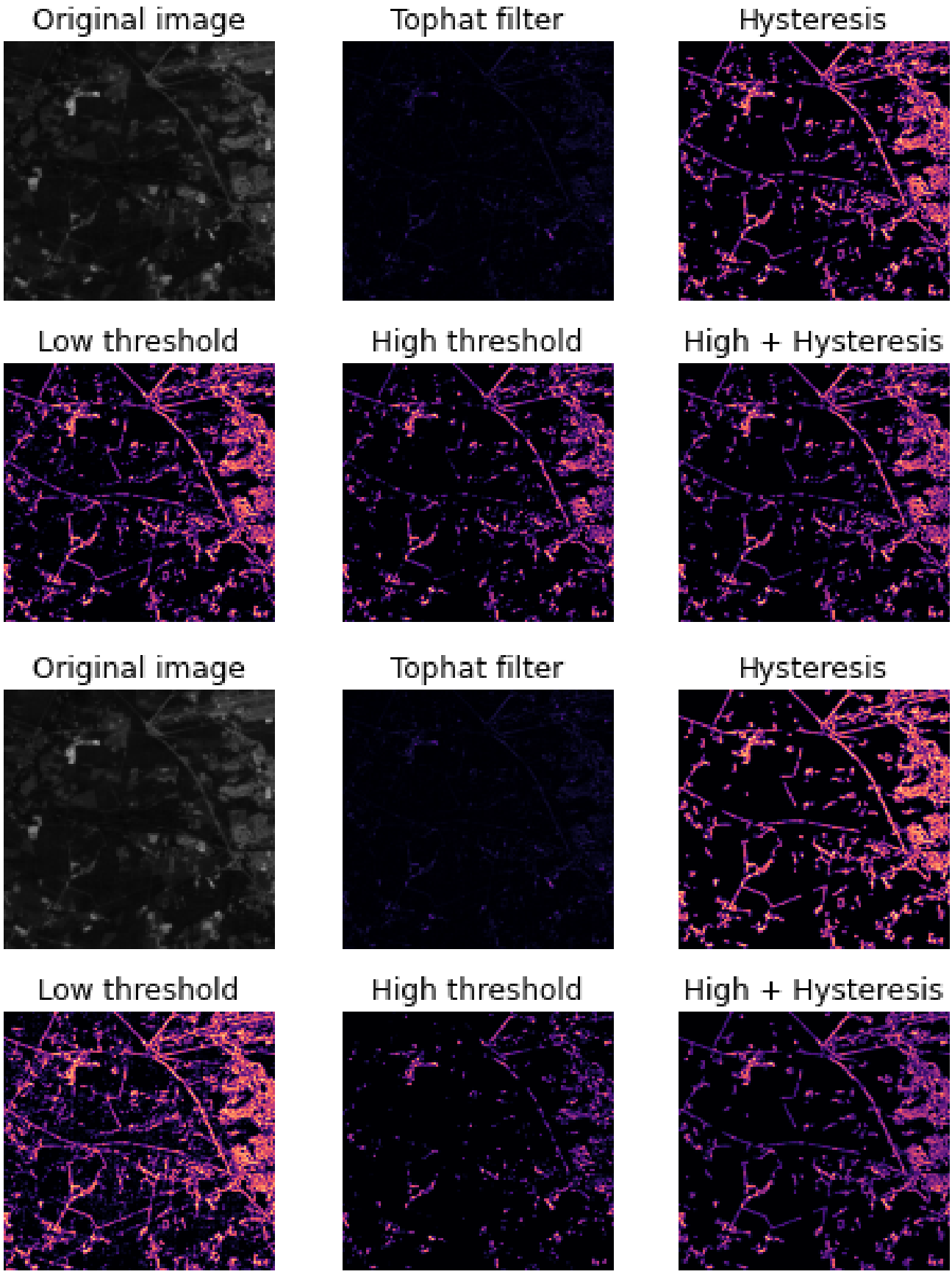
- Pour segmenter pyra-gauss, je choisirais le filtre de Deriche.
- Pour le pré-traitements, c'est possible d'utiliser un filtre, comme médian, gaussian, etc.
- Pour le post-traitements: max local du gradient, seuillage, hystérésis et fermeture.

2 Seuillage avec hystérésis

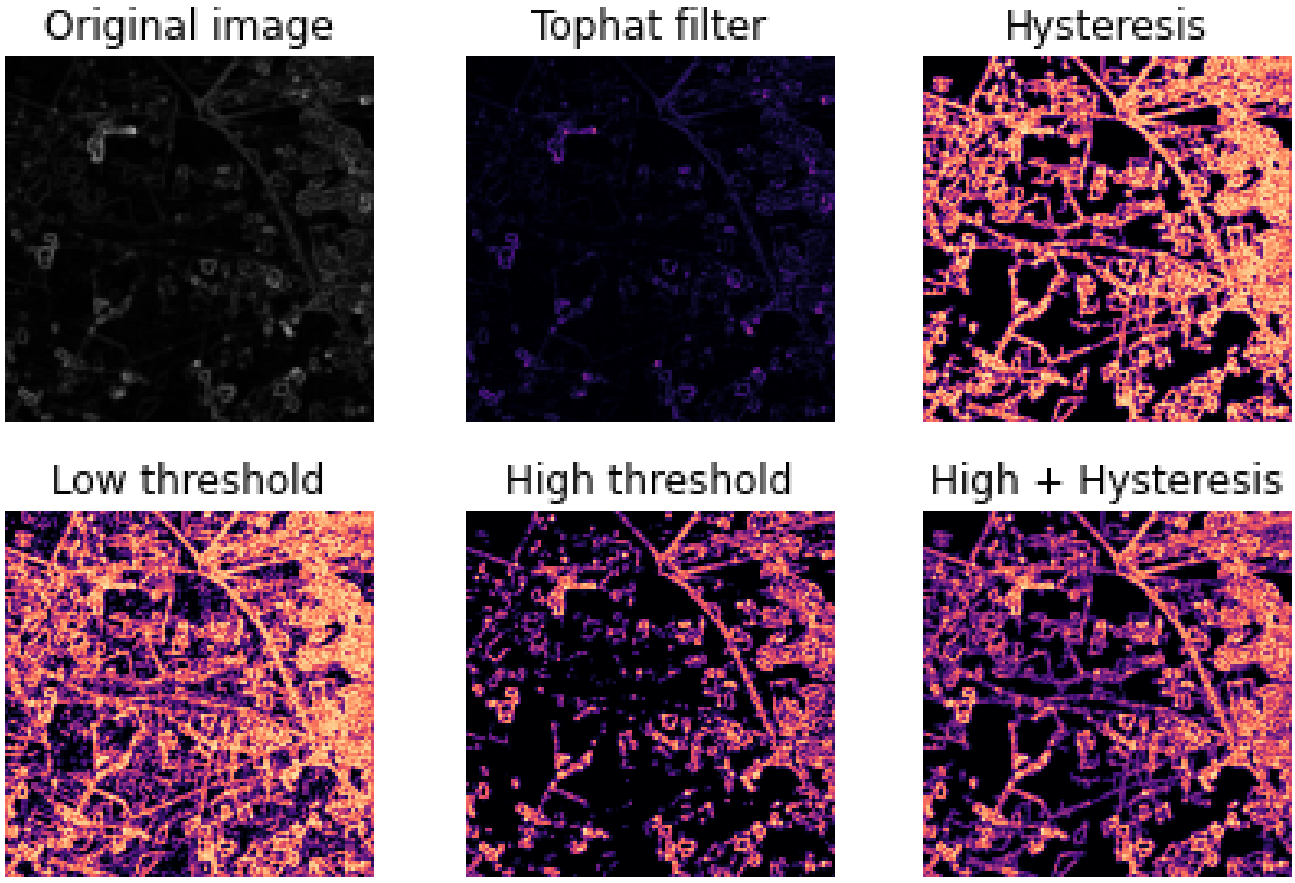
2.1 Application à la détection de lignes

- Dans les figures ci-dessous, il est possible d'observer la même image filtrée avec le filtre du Chapeau. Les valeurs utilisées étaient, respectivement:
  1. Low seuilles = 3, High seuilles = 5, Rayon = 3.
  2. Low seuilles = 3, High seuilles = 5, Rayon = 5.
  3. Low seuilles = 2, High seuilles = 9, Rayon = 5.



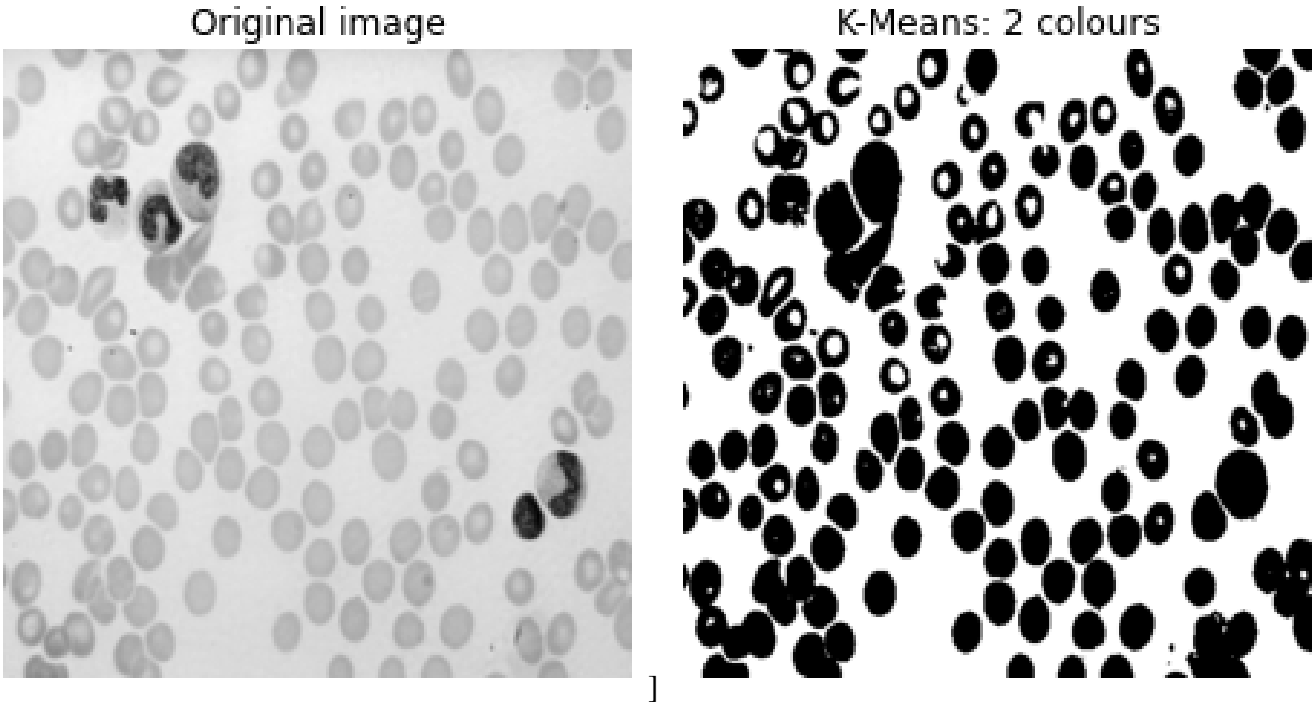

Original image
Tophat filter
Hysteresis
Low threshold
High threshold
High + Hysteresis
Original image
Tophat filter
Hysteresis
Low threshold
High threshold
High + Hysteresis

- L'image avec Low seuilles = 2, High seuilles = 9, Rayon = 5 était le meilleur à mon avis
- En appliquant le filtre Sobel avant, il est possible d'observer que les contours de l'image sont devenus plus continus et il est plus facile de différencier les couleurs de l'image, le résultat était en effet très différent.



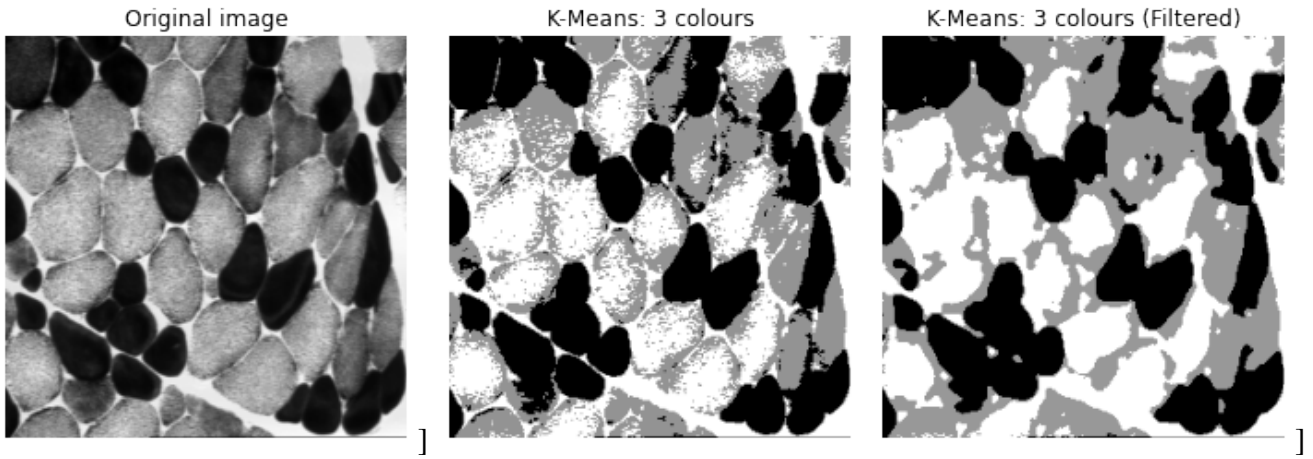
3 Segmentation par classification : K-moyennes

3.1 Image à niveaux de gris



]

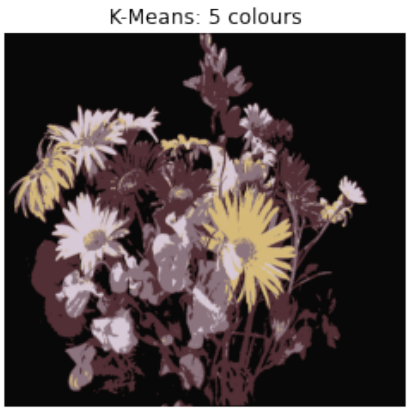
- K means fait un travail décent en classant les cellules, cependant, il échoue avec 2 classes losque l'intérieur des cellules est trop clair. C'est possible d'appliquer une méthode de fermeture pour résoudre ce problème.
- C'est possible d'initialiser les classes de forme aléatoire ou définir les centres manuellement. Généralement, le "random" donne de meilleurs résultats. Cependant, il y a le methode kmeans++, que fait le calcul suivant: si le point est loin des centroids que on déjà choisir, il y a une grande probabilité, si non, une petite probabilité. Kmeans++ donne normalement un meilleur résultat, car il y a une chance de démarrer l'algorithme avec 1 centre dans chaque cluster.
- Pour les images cellulaires, la classification est stable, puisqu'il n'y a que 2 classes très bien définies. Cependant, dans le cas d'une image dans laquelle les classes ne sont pas très bien définies ou il y a plus de classes, elle devient plus instable.
- Il y a 3 couleurs différentes dans l'image : les fibres grises, les fibres noires et l'espace blanc, où il n'y a pas de fibres. Les 2 derniers sont faciles à classifier, puisque les couleurs sont constantes. Cependant, pour les fibres grises, la couleur n'est pas constante, c'est-à-dire que le niveau de gris change entre chaque fibre, ce qui rend plus difficile pour le classificateur de distinguer les niveaux de gris et de comprendre qu'ils sont de la même classe.
- Les filtres peuvent rendre l'image plus lisse, réduire le bruit et, par conséquent, rendre la couleur grise plus constante.

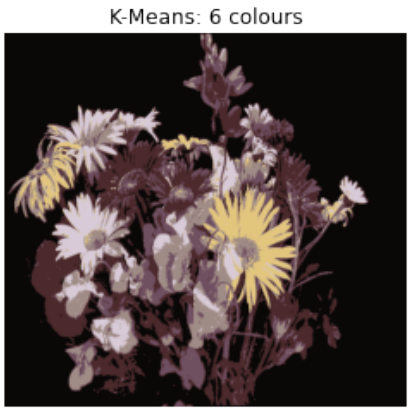


3.2 Image en couleur

- Il est possible de voir que 10 classes c'est une bonne approximation des différentes couleurs de l'image. Cependant, l'image semble plus plate et certaines couleurs ne sont pas très fidèles, depuis qu'il a perdu une certaine gamme de couleurs.

- 5 classes ont donné une classification décente, mais 6 classes serait probablement le nombre minimum de classes qui donne un rendu visuel similaire à celui de l'image codée.
- Il est possible de filtrer l'image afin d'égaliser le niveau de gris avant l'application des k means.



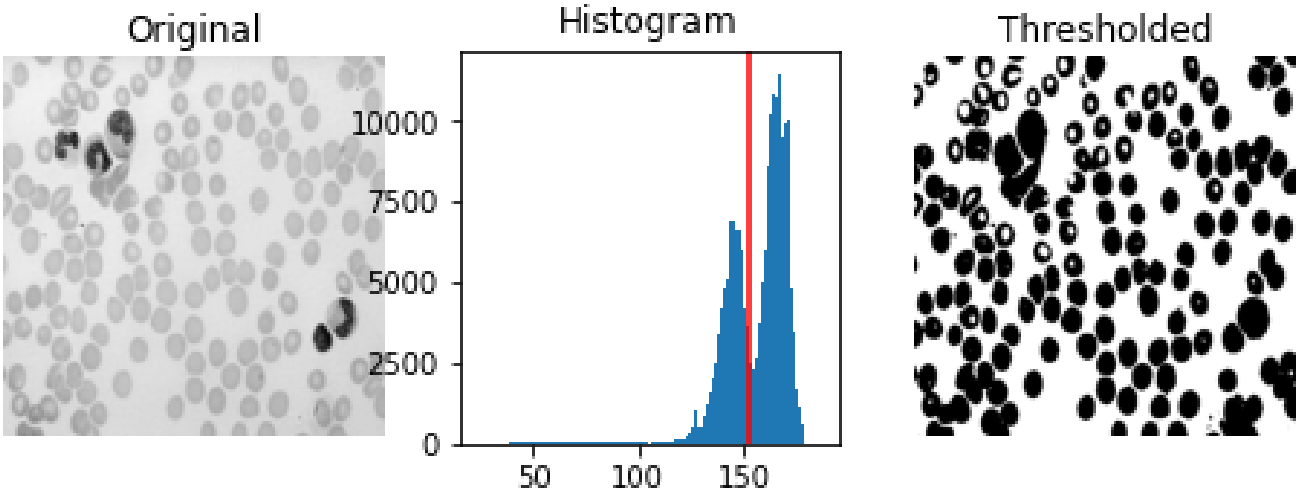


]

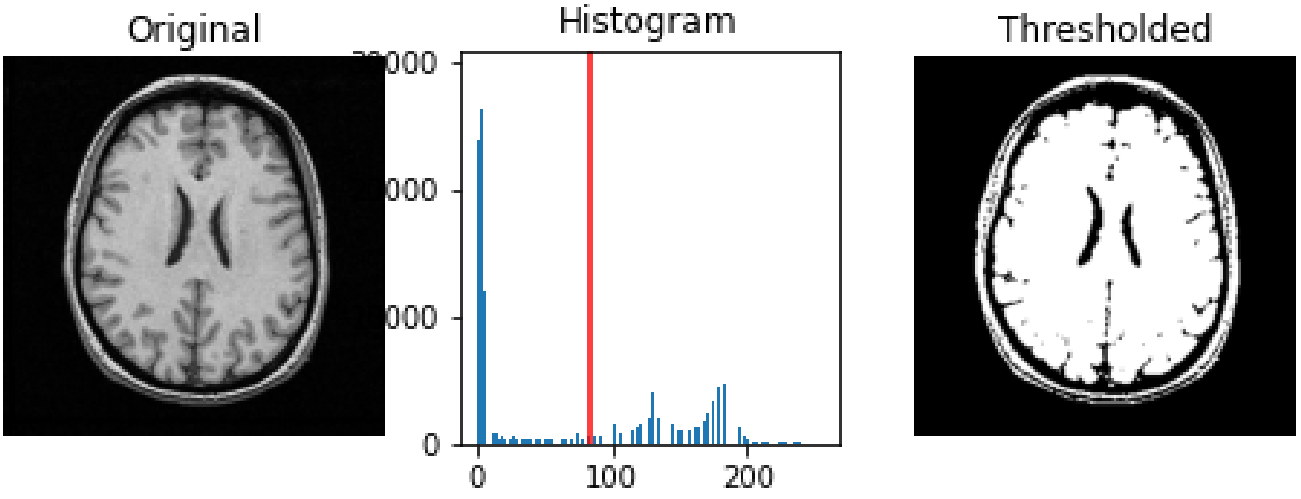
]

4 Seuillage automatique : Otsu

- Le critère à optimiser est le meilleur seuil qui donne le plus grand K.



]



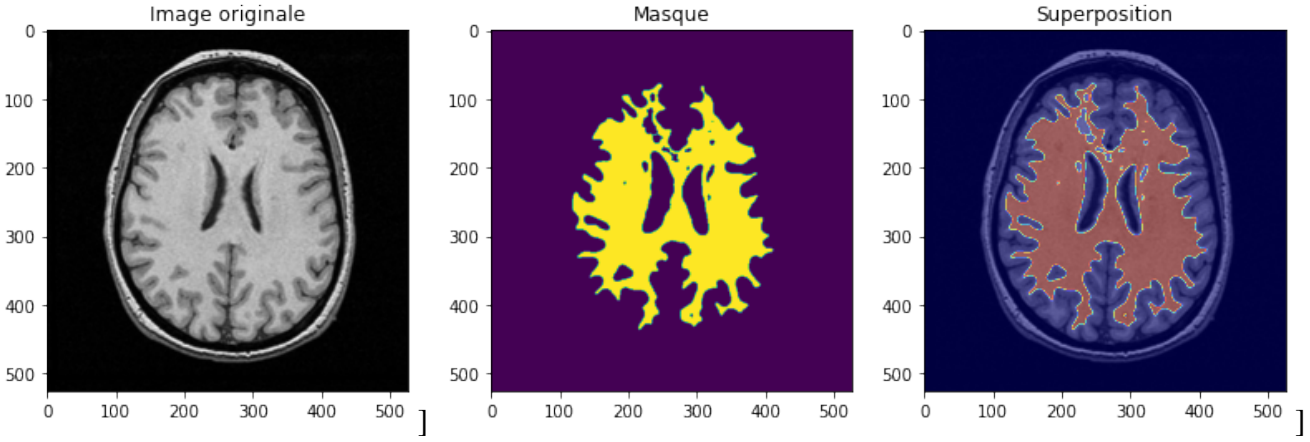
- Avec les tests, il est possible d'observer que l'algorithme a un bon résultat pour les images qui ont 2 classes.
- C'est possible de seuiller correctament une image de norme du gradient, puisque les normes les plus grandes (contours) seraient séparées des normes les plus basses, ce qui permet d'utiliser la méthode.
- Le code suivant applique les modifications demandées:

```
import matplotlib.pyplot as plt
from skimage import data
from skimage import io as skio
from skimage.filters import threshold_otsu
import numpy as np

def histogram(im):

    nl,nc=im.shape

    hist=np.zeros(256)

    for i in range(nl):
        for j in range(nc):
            hist[im[i][j]]=hist[im[i][j]]+1

    for i in range(256):
        hist[i]=hist[i]/(nc*nl)

    return(hist)


def otsu_thresh(im):

    h=histogram(im)

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

    maxt=0
    maxk=0


    for t in range(256):
        w0=0
        w1=0
        m0=0
```

```
        m1=0
        m2=0
        for i in range(t):
            w0=w0+h[i]
            m0=m0+i*h[i]
        if w0 > 0:
            m0=m0/w0

        for i in range(t,256):
            w1=w1+h[i]
            m1=m1+i*h[i]
        if w1 > 0:
            m1=m1/w1

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

        if k > maxk:
            maxk=k
            maxt=t


    thresh=maxt

    return(thresh)


image = skio.imread('cell.tif')
thresh = threshold_otsu(image)
print(thresh)
binary = image > thresh

thresh=otsu_thresh(image)
binary = image > thresh
print(thresh)

fig, axes = plt.subplots(ncols=3, figsize=(8, 2.5))
ax = axes.ravel()
ax[0] = plt.subplot(1, 3, 1)
ax[1] = plt.subplot(1, 3, 2)
ax[2] = plt.subplot(1, 3, 3, sharex=ax[0], sharey=ax[0])

ax[0].imshow(image, cmap=plt.cm.gray)
ax[0].set_title('Original')
ax[0].axis('off')

bins=np.max(image)-np.min(image)+1
```

```
ax[1].hist(image.ravel(), bins=bins)
ax[1].set_title('Histogram')
ax[1].axvline(thresh, color='r')

ax[2].imshow(binary, cmap=plt.cm.gray)
ax[2].set_title('Thresholded')
ax[2].axis('off')

plt.show()
```

5 Croissance de régions



- Le contraiste en question c'est si la différance absolute entre la moyenne du voisinage moins le moyenne de l'image c'est moins que un seuille fois l'écart type.
- Lorsque le thresh est augmenté, il est possible d'ajouter plus de pixels à l'objet. En revanche, lorsque le thresh est diminué, on ajout moins de pixels.
- La position du germe initiale permet de segmenter correctement la matière blanche.
- Si le germe initiale est dans la matière grise, c'est possible de segmenter.