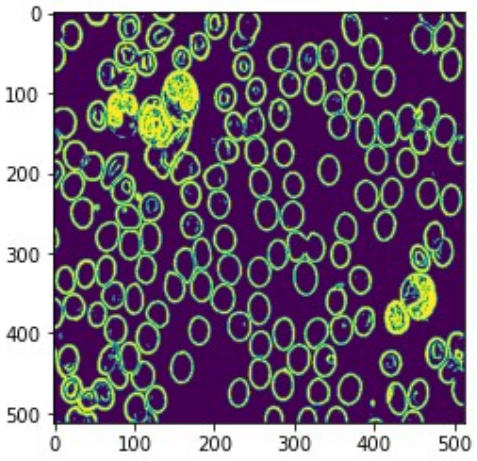
TP3 – Chloé Court

1.1 -

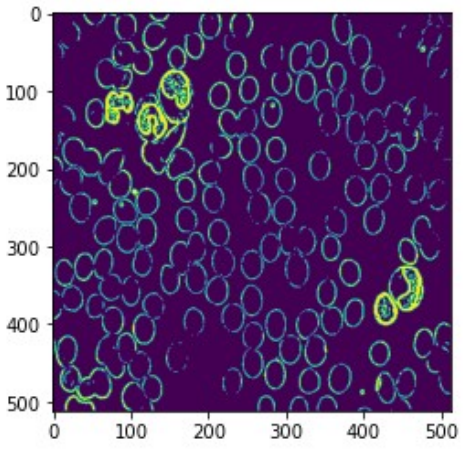
- Le filtre de Sobel a l'avantage de lisser le bruit car ce dernier va réaliser une moyenne autour d'un pixel. A l'inverse, le filtre différence va faire une différence entre deux pixels, ce qui peut être un problème s'il y a du bruit.

- Ainsi, il n'est pas nécessaire de réaliser un filtre-passe bas avant d'appliquer le filtre de Sobel puisque ce dernier permet déjà de filtrer le bruit.

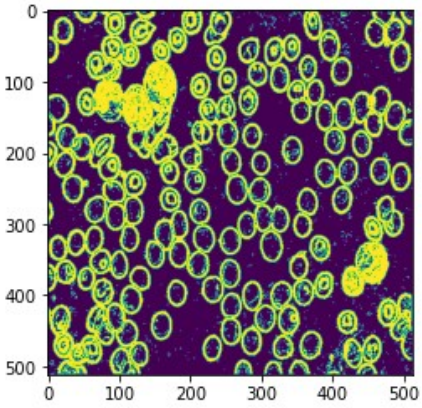
- Avec un seuil de 0.1, pour la majorité des cellules, les contours sont bien continus, robustes au bruit, d'une épaisseur cohérente et à la bonne position. Certaines cellules semblent remplies, c'est-à-dire que trop de contours sont détectés au milieu de celles-ci. Lorsque l'on diminue le seuil, les bords deviennent plus épais et le problème précédent s'accentue. A l'inverse, si on augmente ce seuil, l'épaisseur diminue jusqu'à ce que certains contours ne soient plus continus. Il n'y a pas de changement au niveau de la robustesse au bruit et de la position. La valeur optimale du seuil semble se trouver entre 0,1 et 0,2.



Norme du gradient seuillée à 0,1



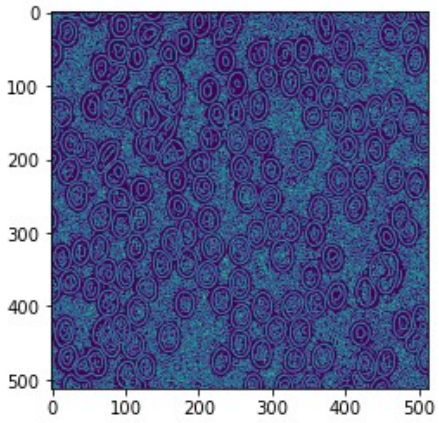
Norme du gradient seuillée à 0,2



Norme du gradient seuillée à 0,05

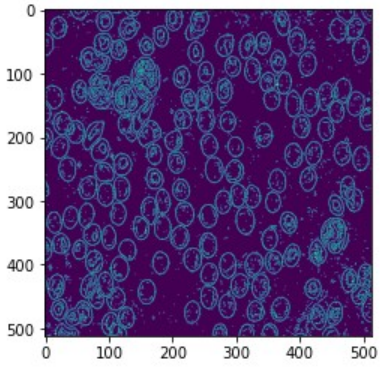
1.2 -

- Pour bien observer les maxima dans la direction du gradient, on a besoin de filtrer en passe-bas pour diminuer le bruit. Le critère de qualité optimisé est la taille des contours, qui sont fins (larges de 1 pixel).

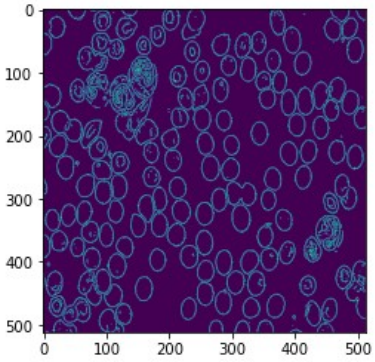


Maxima dans la direction du gradient avec l'image filtrée avec sigma = 0,8

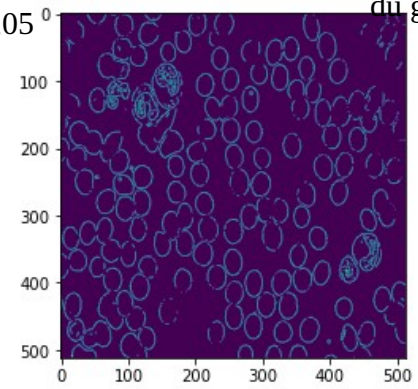
- En éliminant les contours dont la norme est inférieure à un seuil donné, on observe que la robustesse au bruit n'est pas bonne avec un seuil trop faible (ex avec 0,05). En revanche, la continuité des contours est affectée si on seuille trop (ex 0,2). La position des contours ne change pas, mais certains disparraissent avec le seuillage.



Maxima dans la direction du gradient seuillé à 0,05

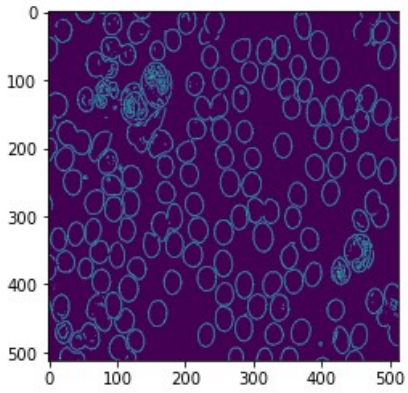


Maxima dans la direction du gradient seuillé à 0,1



Maxima dans la direction du gradient seuillé à 0,2

- On souhaite alors obtenir le meilleur compromis entre le bruit et la continuité des contours. Il semble que ce soit optimal pour un seuil entre 0,1 et 0,2. En faisant varier le seuil, cela semble etre optimal pour un seuil à 0,15.

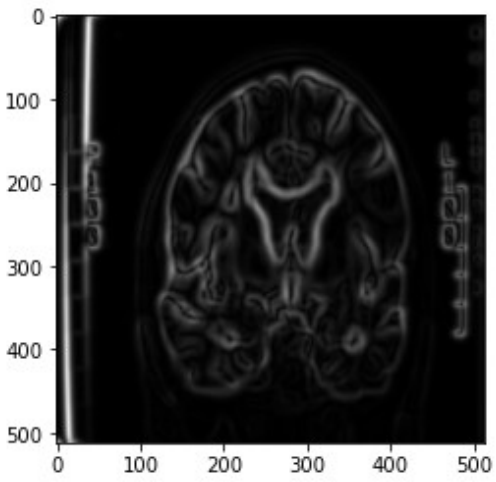


Maxima dans la direction du gradient seuillé à 0,15

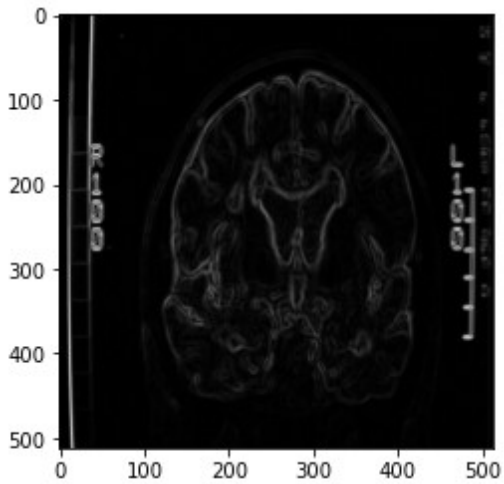
1.3 -

La fonction gradient modifiée devient :

```
def dericheGradX(ima,alpha):

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

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

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


#gradx=np.zeros(nl,nc)
    for i in range(nl):

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

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

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

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

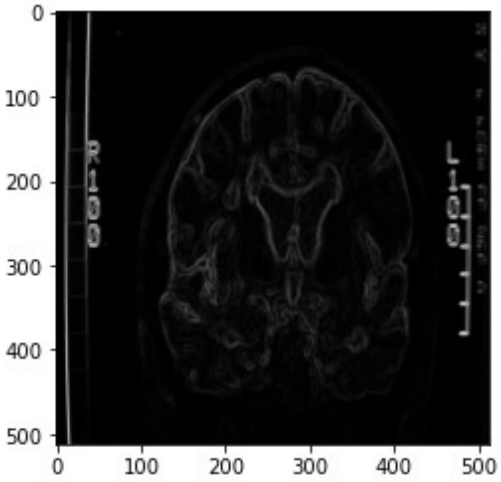


return gradx



Filtre dérich avec alpha=0,5

Filtre dérich avec alpha=2



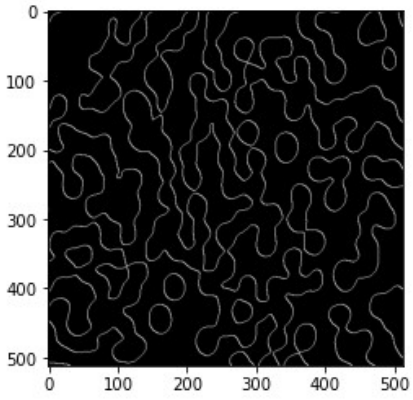
Filtre dérich avec alpha=3

- Modifier alpha sur l'intervalle [0,3;3] modifie le contraste et la netteté. En effet, plus alpha est proche de 3 et plus l'image est floue. En revanche, plus alpha est grand et moins on observe de contrastes.

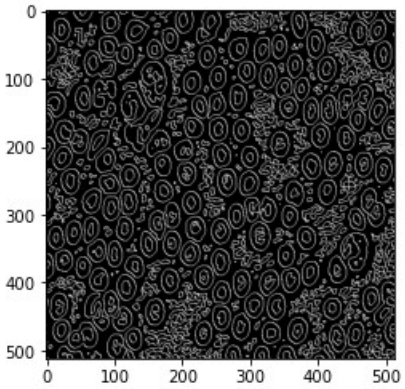
- La valeur de alpha n'a pas d'influence sur le temps de calcul. En effet, dans le calcul de gradX et gradY, le temps de calcul dépend des boucles est en $O(n^2)$ et ne dépend pas de la valeur de alpha (car les valeurs des exponentielles sont calculées en O(1)).

- Les fonctions dericheSmoothX et dericheSmoothY sont utilisées dans le but de lisser l'image (et donc d'en réduire le bruit). Cela s'observe de plusieurs manière dans la nouvelle fonction, d'abord dans le calcul des valeurs des b1[j] et b2[j] et ensuite dans le calcul de smoothx[i,:]=b1+b2 qui désormais est une somme alors qu'avant on faisait une différence. Cela a pour résultat de faire une moyenne ce qui réduit le bruit.

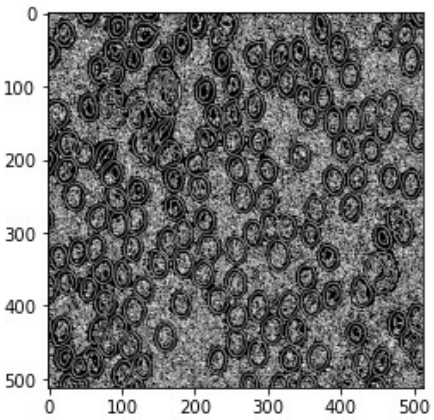
1.4 -



Laplacien binarisé avec alpha=0,2



Laplacien binarisé avec alpha=1



Laplacien binarisé avec alpha=3

- On observe qu'alpha a une influence sur le bruit et sur les contours. La réduction du bruit affecte la précision de la position des contours. Par exemple pour alpha = 0,2, les contours ne correspondent pas aux vrais contours des cellules.

- Quand on augmente alpha, on observe du bruit, quand on le diminue, les contours ont une position de moins en moins fidèle à l'image d'origine. Une des différences avec les filtres précédents est le fait que les contours sont toujours fermés, et donc continus.

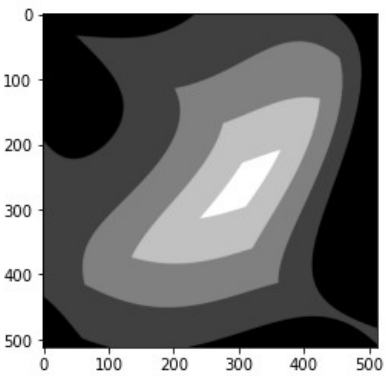
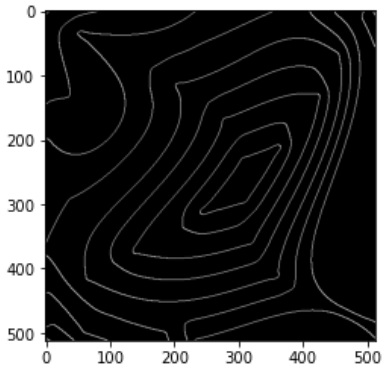


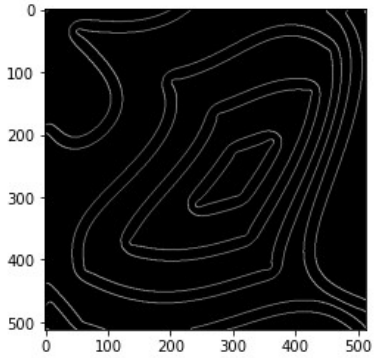
Image originale



Laplacien binarisé avec alpha = 0,5

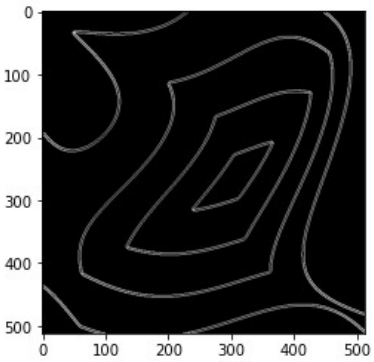
- On observe l'apparition de contours qui n'existent pas sur l'image originale.

Pour pallier ce problème, on peut jouer sur alpha et sur le seuil. En effet, avec lpima>=1, on observe comme suit que la distance des faux contours avec les contours de l'image d'origine diminue. Cependant, cette distance ne devient jamais nulle en augmentant ce seuil.



Laplacien binarisé avec
alpha = 0,5 et lpima>=1

On peut ensuite augmenter alpha pour n'avoir que les contours correspondant à l'image d'origine.



Laplacien binarisé avec
alpha = 3,5 et lpima>=1

1.5 -

- On observe que l'image originale est bruitée. Ainsi, il faut choisir un algorithme pour le filtrer qui soit performant pour diminuer le bruit sans en affecter les contours. Ainsi, le dernier algorithme ne peut etre choisi.

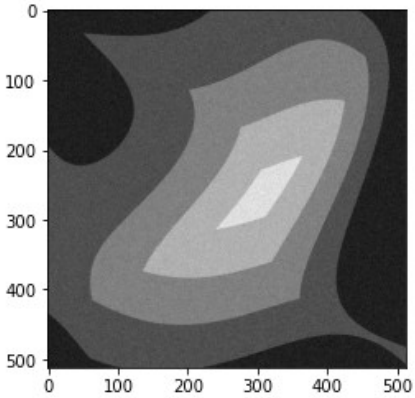


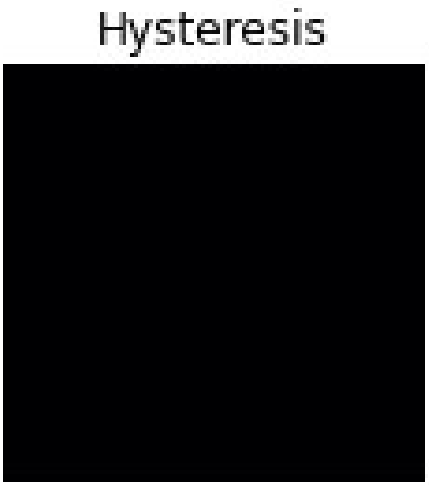
Image originale

En ce sens, le filtrage qui paraît le plus pertinent semble être le filtrage de sobel, car il sera performant dans la diminution de bruit sans altérer les contours.

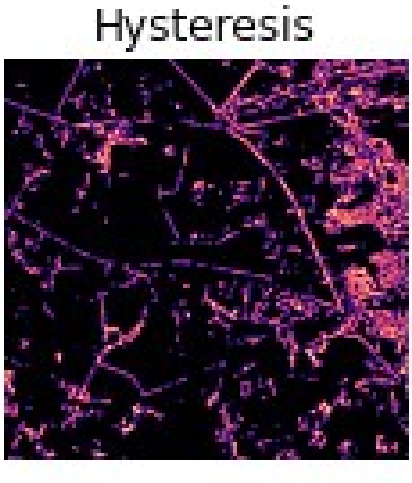
- Il faudrait effectuer un filtrage passe-bas en pré-traitement. En post-traitement, on pourrait calculer le maximum du gradient ou le passage par zéro du laplacien pour respectivement optimiser l'épaisseur des contours et améliorer la netteté et les contrastes.

2.1 -

- Lorsque l'on augmente le rayon de l'élément structurant, les lignes sont de mieux en mieux détectées. En revanche, le temps d'exécution augmente.



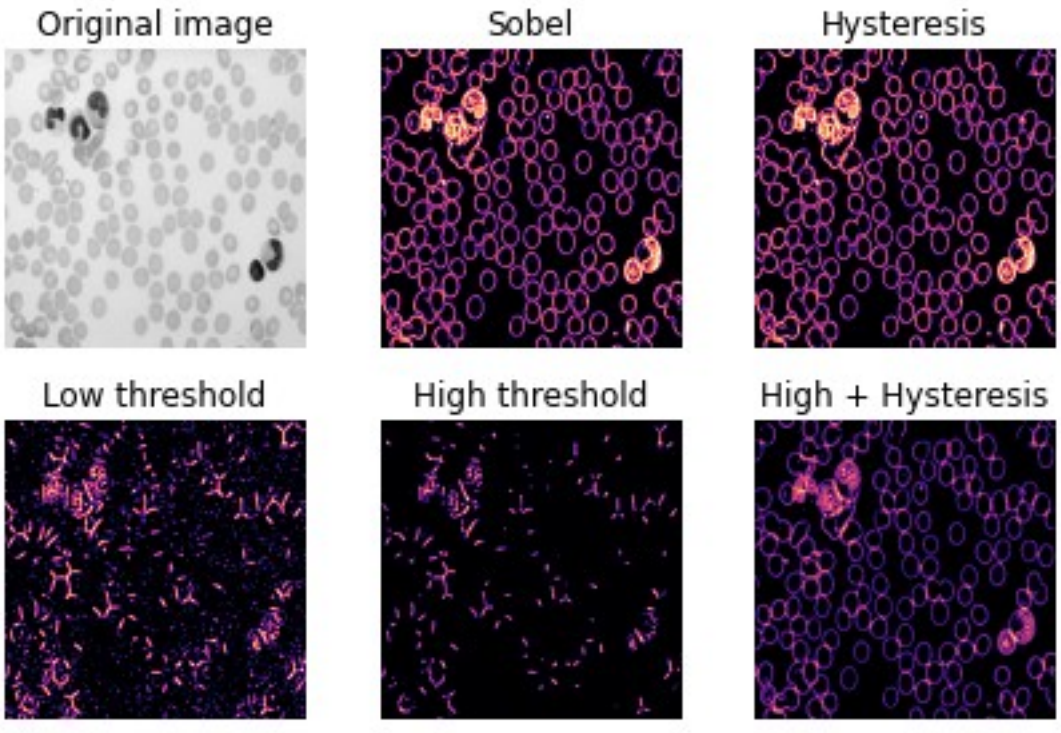
R=1



R=5

R=10

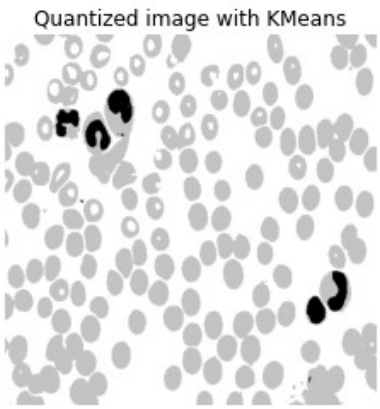
- Un seuil bas trop bas provoque une image trop bruitée. On souhaite tout de même low suffisamment bas pour que les lignes soient continues. Ce n'est pas le cas pour low=3. La valeur optimale de low semble être 2 car les lignes sont continues. On ne prend pas moins car on aurait plus de bruit.
  Le seuil haut optimal ne doit contenir que des points donnant des lignes incomplètes, mais ces points doivent appartenir à des lignes existantes. Pour cela on prend high=6



- On applique le filtre Sobel à l'image cell puis le filtre à Hystérésis. On observe que le résultat est légèrement meilleur avec le filtre à Hystérésis que les résultats obtenus précédemment.

3.1 -

- Pour une classification en deux classes, l'algorithme segmente bien les cellules et restitue l'information des noyaux clairs. Cependant, on perd l'information des noyaux noirs. On peut essayer cet algorithme pour 3 classes :



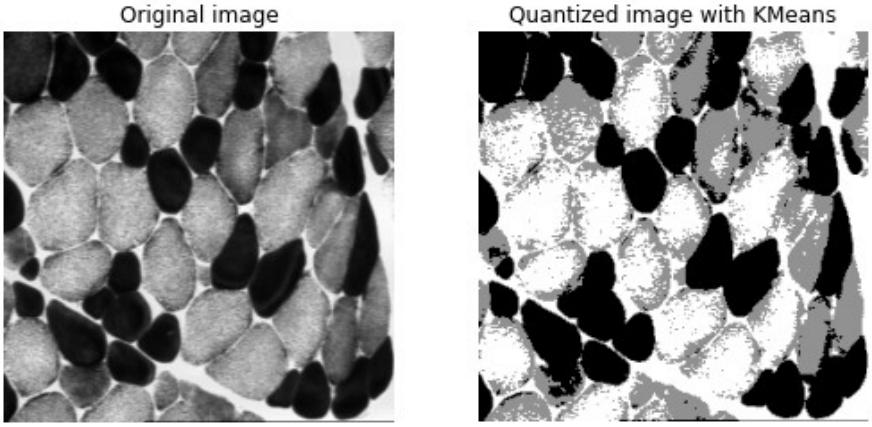
Dans ce cas là, on retrouve bien toutes les informations souhaitées.

- Il existe deux initialisations possibles pour initialiser les classes. En effet, il y a kmeans++ et random. Il est aussi possible de définir les centres.
  - K-Means++ est une méthode d'initialisation qui vise à améliorer la convergence et la qualité des clusters en sélectionnant les centres de manière stratégique.
  - L'initialisation aléatoire est plus simple, mais elle est moins prédictible et peut nécessiter plus d'itérations pour converger vers une solution acceptable.

On préfèrera donc souvent la méthode K-means ++ car elle produit des résultats plus stables et de meilleure qualité.

kmeans = KMeans(n_clusters=n_class, init='random', random_state=0).fit(image_array_sample)

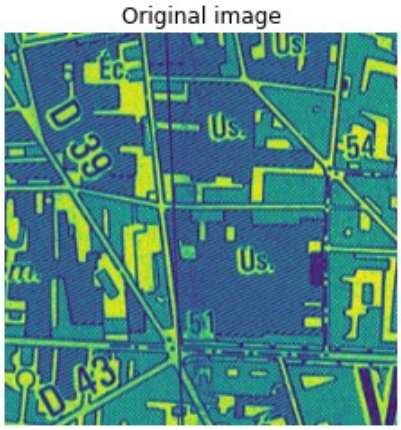
- Avec une initialisation aléatoire, la classification obtenue est toujours stable, peu importe le nombre de classes et le nombre de niveaux de gris différents sur l'image.

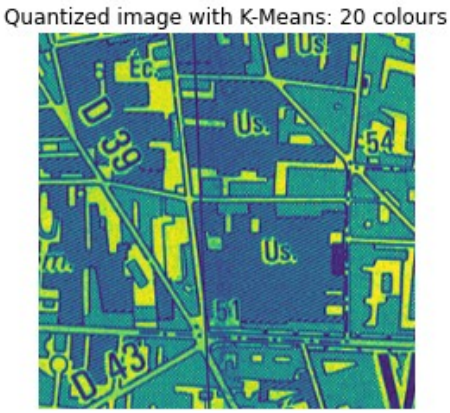


- Ici, l'image originale semble avoir 3 niveaux de gris, donc une classe n=3 devrait en théorie convenir. Or le problème est que tous les pixels de chaque portion n'ont pas forcément le même niveau de gris ce qui fait que les portions ne sont au final pas de la même couleur après cette classification. Ainsi sur des images où les niveaux de gris ne sont pas constants sur un voisinage, cet algorithme ne paraît pas le plus performant.
- Le filtrage moyen ou médian permettra de réduire le bruit et donc les variations brutales de niveaux de gris des pixels voisins. Ainsi, le résultat de la segmentation par classification sera meilleur.

3.2 -

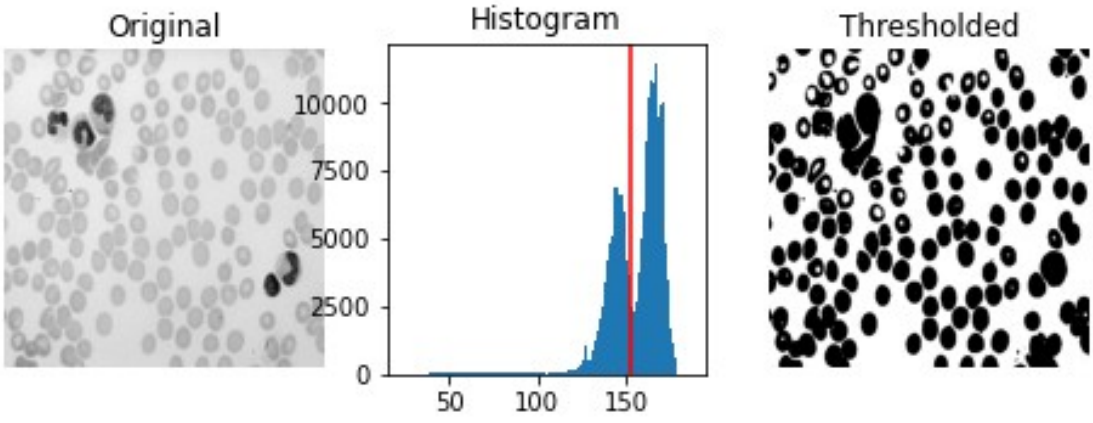
- L’image quantifiée avec 10 couleurs est très similaire à l’image originale. On distingue parfaitement chaque fleur. Il y a tout de même évidemment une perte d’informations au niveau des couleurs qui paraissent beaucoup plus ternes. De plus, on perd aussi une impression de profondeur, la texture et les dégradés de couleur au niveau d’une fleur.
- Pour que le rendu soit similaire à celui de l’image codée sur 3 octets, on peut prendre n=20. En effet, pour n=19, on a encore certains dégradés qui sont abrupts, notamment sur les fleurs violettes en premier plan. Or comme il y a en pratique 256^3 couleurs différentes, il faudrait ce même nombre de classes pour restituer parfaitement l’image.
- Pour retrouver les vraies couleurs de la carte IGN, il faut augmenter progressivement le nombre de classes jusqu’à retrouver une image en couleur correspondant à l’image originale. Cela fonctionne pour n = 20 :

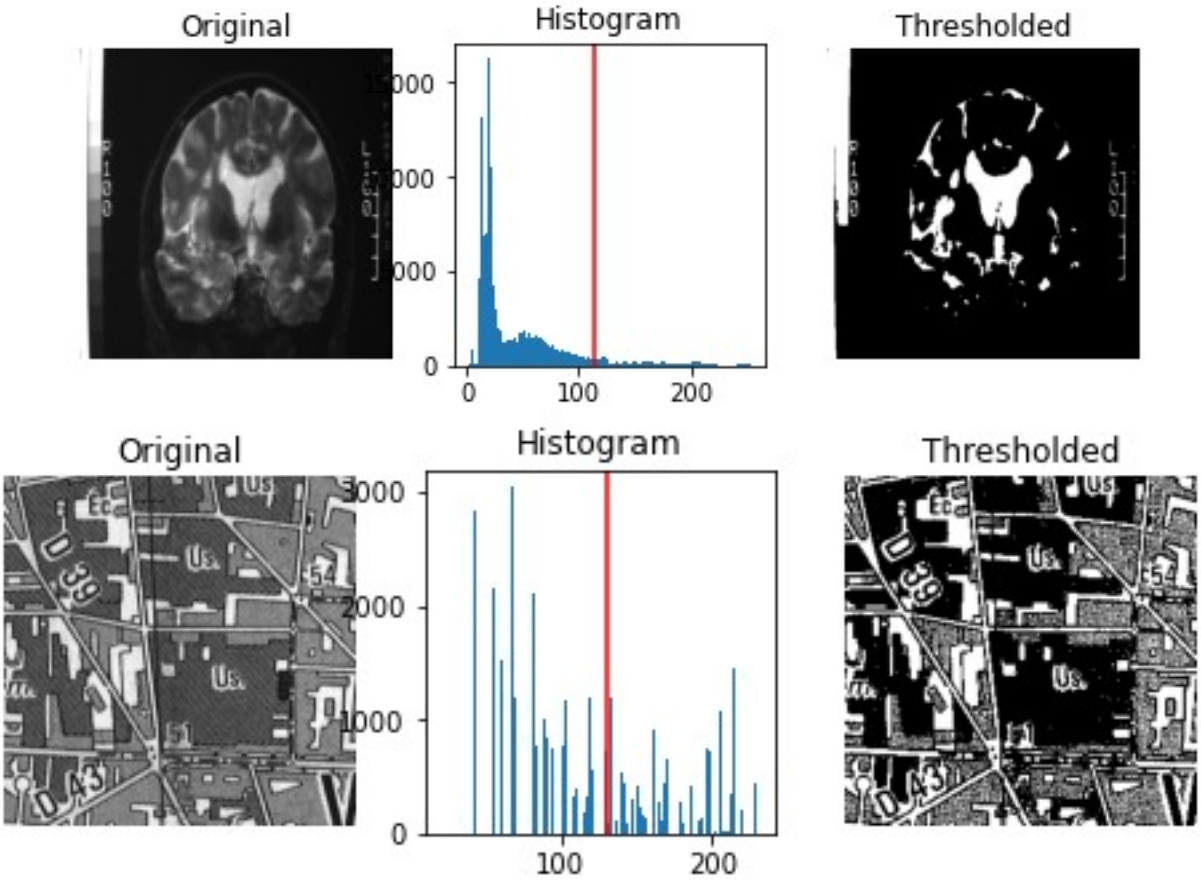




4 -

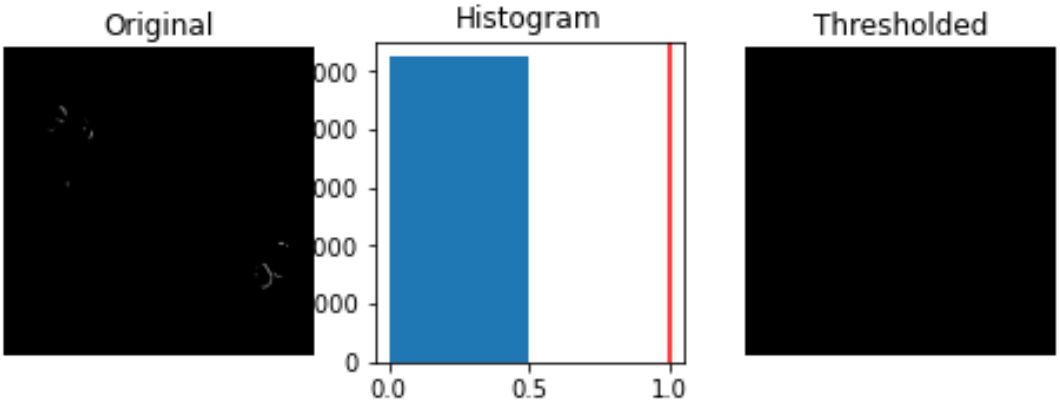
- On cherche à minimiser la dispersion intra-classe en déterminant un seuil optimal qui maximise la variance entre les deux classes. Ce seuil optimal est déterminé en parcourant tous les niveaux de gris possible et en calculant une valeur de critère (k) pour chaque seuil possible, puis en sélectionnant le seuil qui maximise cette valeur de critère.

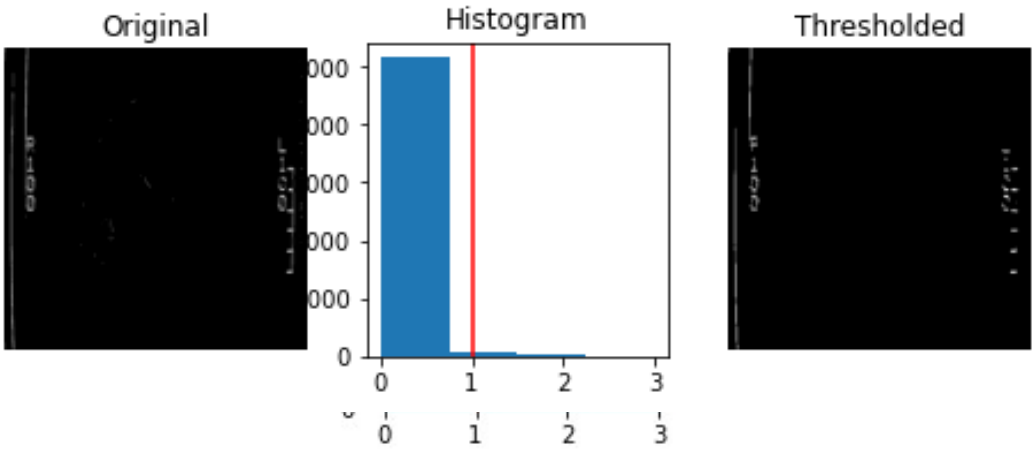




- Lorsque l'image possède plus de 2 niveaux de gris différents, il n'existe aucun seuil qui puisse convenir pour restituer toutes les classes. Ainsi, pour les cellules, on voit qu'on a perdu l'information des noyaux noirs, et dans la deuxième on a perdu la partie grise du cerveau. Un autre seuil aurait pu permettre d'avoir tout le cerveau en blanc mais on n'aurait jamais pu avoir l'information des trois niveaux de gris avec cet algorithme. Il faudrait au moins 3 classes pour ces deux images. Cependant, pour l'image carte, comme il y a du bruit il y a des pixels aux niveaux de gris très différents qui sont voisins et qui donc ne vont pas forcément être du même côté du seuil. Ainsi on a cette impression de gris. Pour résumer, si le niveau de gris est proche sur un voisinage (pas de bruit), plus de 2 niveaux de gris suffisent à perdre beaucoup d'informations sur l'image.

- Il est en effet possible de seuiller sur la norme du gradient, hors cela est bien moins efficace que de seuiller avec contours norme comme nous l'avons fait précédemment. En effet, sur les images qui ayant plus de 2 classes et n'ayant pas de bruit, les résultats ne sont pas du tout concluants (cf ci-dessous avec cell.tif, carte.tif et brain.tif).





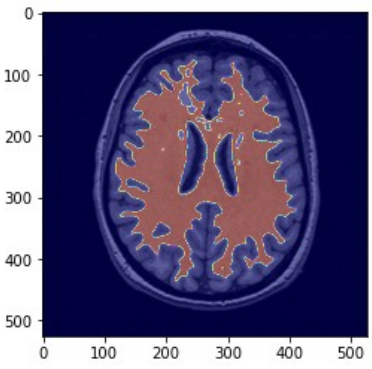
- Pour traiter le problème à trois classe, c'est-à-dire la recherche de deux seuils, il faut modifier l'algorithme comme suit :

```
def otsu_thresh(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
```

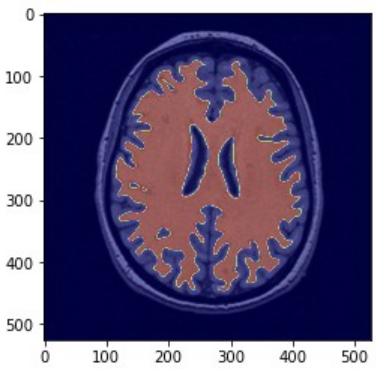
```
    for i in range(t1,t2):
        w1=w1+h[i]
        m1=m1+i*h[i]
    if w1 > 0:
        m1=m1/w1

    for i in range(t2, 256):
        w2=w2+h[i]
        m2=m2+i*h[i]
    if w2 > 0:
        m2=m2/w2

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

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

thresh=(maxt1,maxt2)

return(thresh)
```

5 -

- Pour qu'un pixel soit ajouté à l'objet existant, il faut qu'il soit voisin d'un pixel dans le périmètre et qu'il ait un niveau de gris similaire.
- Plus tresh est grand, et plus il y a de pixels dans l'objet existant. A l'inverse, plus le seuil est bas et moins il y a de pixels.

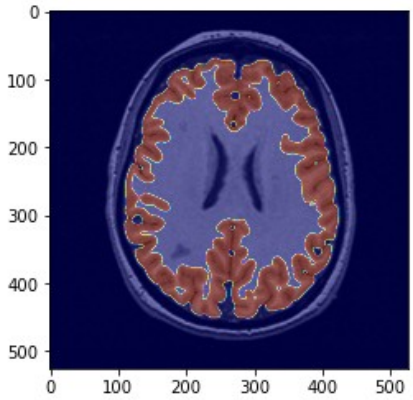


Segmentation de la matière blanche avec tresh =2



Segmentation de la matière blanche avec tresh =4

- Pour segmenter la matière blanche, il faut que le germe initial se trouve dans la matière blanche. Ainsi, x0 et y0 étant les positions du pixel étant le germe initial sont les paramètres qui permettent de segmenter la matière blanche.
- Il est donc possible de segmenter la matière grise en positionnant le germe initial dans la matière grise. En effet, en prenant y0 = 100 et x0 = 350, on a bien segmenté la matière grise.



- Le prédicat permet de mettre en place la condition pour qu'un pixel soit ajouté à l'objet existant. En effet, il faut que la différence de moyenne `m` du voisinage de ce pixel soit inférieur au seuil multiplié par l'écart-type initial (s0).
- Sans utiliser la croissance de région, on pourrait au lieu d'utiliser le périmètre comparer la moyenne du voisinage des pixels à la moyenne du pixel au centre.
- Pour que l'algorithme de croissance de région soit nécessaire, il faudrait comparer la moyenne du voisinage d'un pixel à la moyenne sur la région entière (et non pas d'un pixel sur le périmètre).