

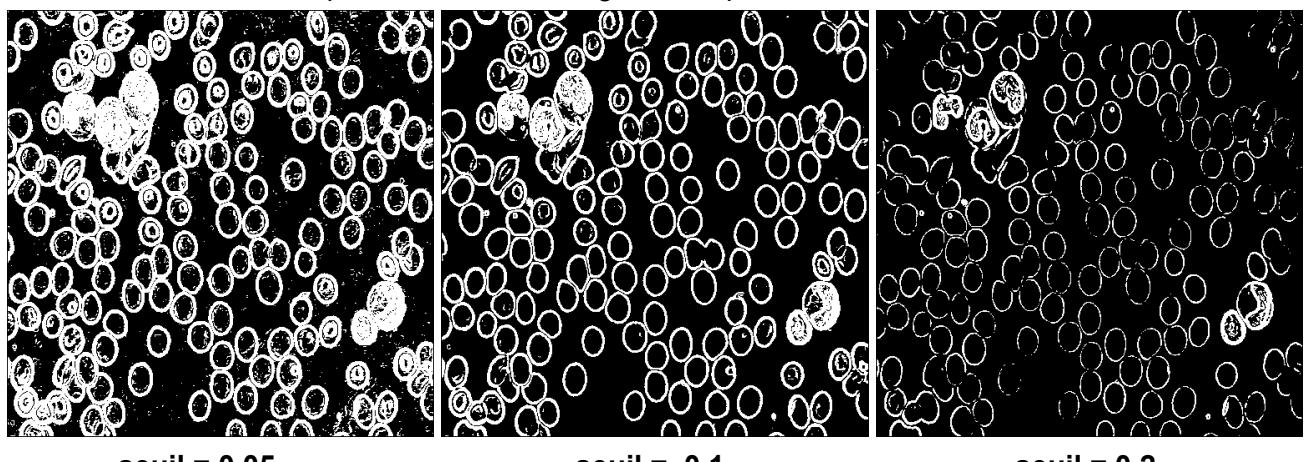
## 1 Détection de contours

### 1.1 Filtre de gradient local par masque

Le noyau de convolution du filtre de Sobel est de taille 3x3, il utilise donc le voisinage d'un pixel pour calculer son gradient, ce qu'on ne fait pas lors du calcul d'une dérivée par la simple différence entre deux pixels voisins. En plus, le filtre de Sobel calcule le gradient dans une direction et un moyenne pondérée dans l'autre direction, ce qui le rend plus robuste et moins sensible au bruit.

Pour des bruits à intensité faible, le filtre de Sobel lui même est suffisant pour réduire le bruit dans l'image. Si le bruit dans l'image est fort, on devra appliquer un filtre passe-bas avant d'appliquer le filtre de Sobel pour calculer le gradient.

Lorsqu'on fixe un seuil très bas, les contours détectés sont continus, cependant ils sont très sensibles au bruit et assez épais, ce qui implique une incertitude sur leur véritable localisation. En augmentant le seuil, les contours deviennent plus fins et plus résistants au bruit, au détriment de la continuité. De plus, bien qu'il y ait une réduction du nombre de faux contours détectés, une partie des vrais est également perdue.

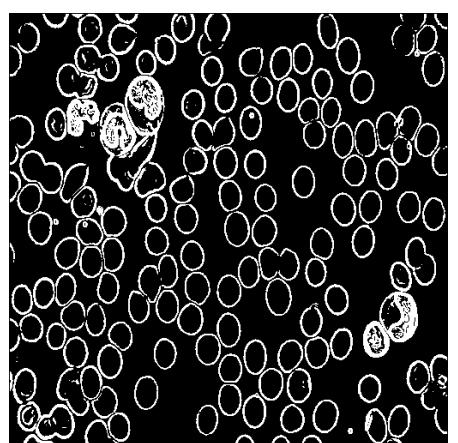


### 1.2 Maximum du gradient filtré dans la direction du gradient

En prenant les maxima du gradient dans la direction du gradient, on obtient des contours plus fins et bien localisés car les maxima se trouvent sur un seul pixel à chaque fois.

Comme vu dans la partie 1.1, quand on met un seuil très bas, le bruit apparaît encore avec les contours. Quand on met un seuil très haut, le bruit est supprimé mais des discontinuités dans les contours apparaissent.

En faisant varier le seuil, un bon compromis entre robustesse au bruit et continuité des contours semble être trouvé pour un seuil de 0,15 (figure ci-contre). La plupart des contours parasites sont éliminés, même si certaines microstructures subsistent, et la majorité des cellules sont détectées.



### 1.3 Filtre récursif de Deriche

Correction des fonctions **dericheGradX** et **dericheGradY** :

$$\begin{aligned} b1[j] &= l[j-1] + 2 * \alpha * b1[j-1] - (\alpha * \alpha) * b1[j-2] \\ b2[j] &= l[j+1] + 2 * \alpha * b2[j+1] - (\alpha * \alpha) * b2[j+2] \end{aligned}$$

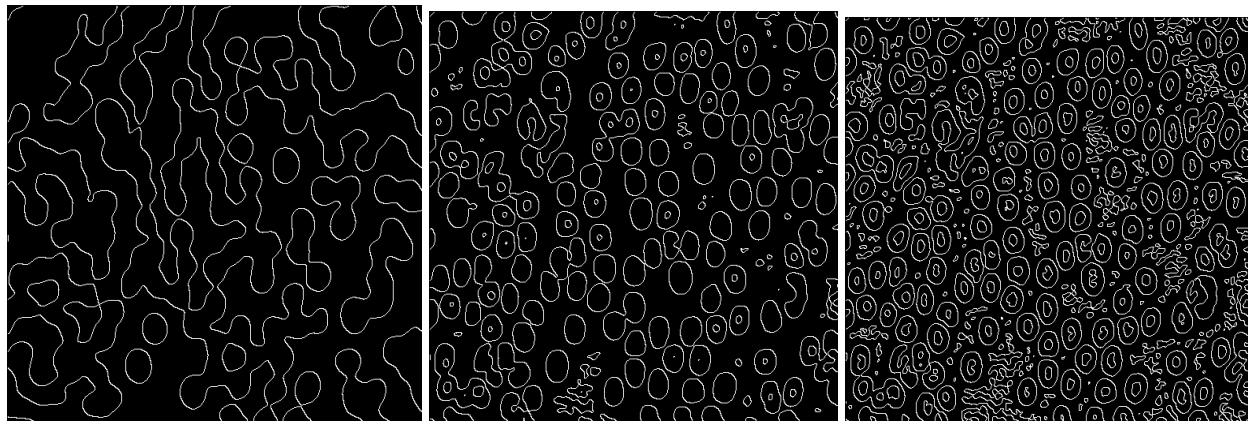
En réduisant le paramètre  $\alpha$ , nous observons une réduction du bruit et de la détection des mauvais contours. Cependant, en le rendant trop petit, les bords des différents objets commencent à fusionner en un seul contour.

Le temps de calcul ne dépend pas de alpha, puisqu'il est uniquement utilisé pour calculer les coefficients de l'expression récursive donc il ne change pas le nombre d'opérations effectuées sur chaque pixel.

Les fonctions **dericheSmoothX** et **dericheSmoothY** servent à lisser l'image grâce à l'application d'un filtre passe-bas avant d'appliquer la détection des contours.

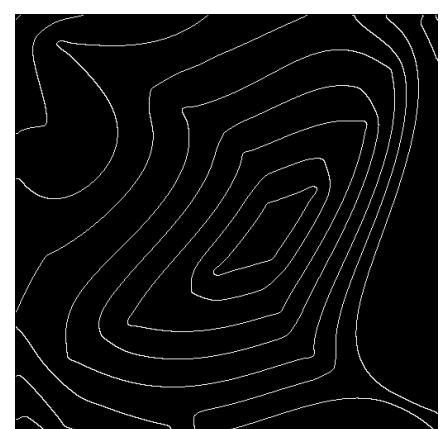
### 1.4 Passage par zéro du laplacien

On remarque que plus  $\alpha$  diminue, plus la réduction du bruit et de la détection de contours parasites, cependant, en le rendant trop petit, les bords des différents objets commencent à fusionner en un seul contour.



Les autres opérateurs vus précédemment (contours, Deriche) permettent de ressortir les contours de l'image cell.tif mieux que la méthode du passage par zéro du laplacien.

L'approche du Laplacienne présente l'inconvénient que le laplacien des points d'inflexion d'une fonction est également égal à zéro, ce qui entraîne la détection de faux contours sur des images en escalier, comme pyramide.tif (figure à droite). Pour éliminer ces faux contours, il est possible de ne pas prendre en compte les contours dont la norme est inférieure à un seuil donné car comme ils ne correspondent pas à des bords réels, la norme du gradient sera très faible dans de telles régions.

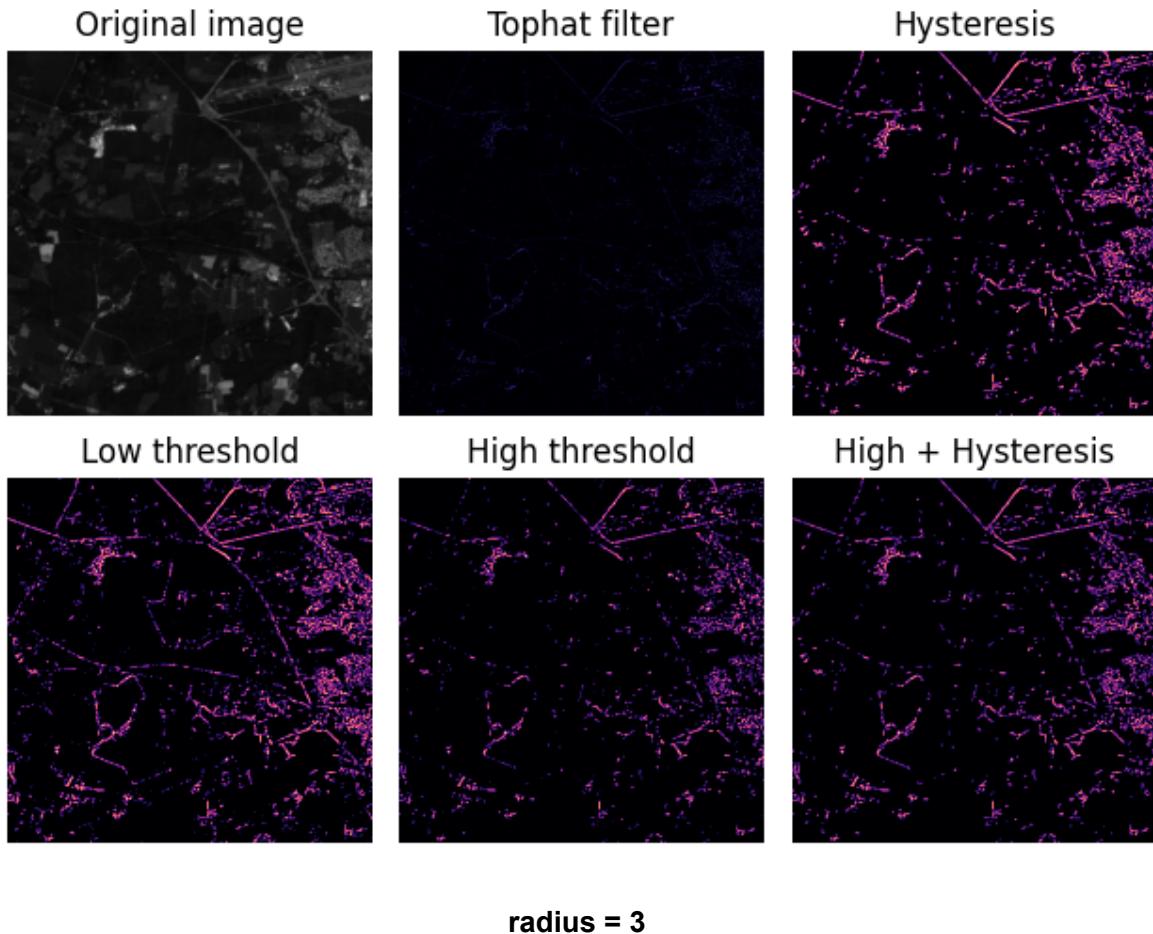


## 1.5 Changez d'image

Puisque l'image pyra-gauss.tif est bruitée et en escalier, on ne peut pas choisir Laplacien puisqu'il est sensible au bruit et peut nous donner des contours faux sur ce genre d'image. Je choisirais plus tôt l'opérateur de Deriche car il est moins sensible au bruit.

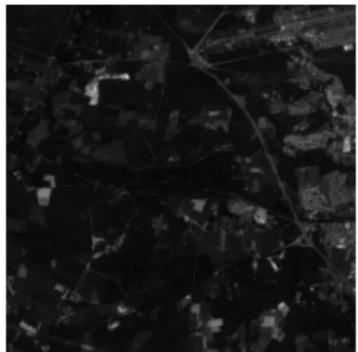
L'image pyra-gauss.tif étant bruitée, on doit appliquer un filtre passe bas pour lisser l'image comme pré-traitement . Le post-traitement peut se faire par la détection du maximum de gradient dans la direction du gradient combiné avec une suppression de contour basée sur un seuillage de la norme de gradient.

## 2 Seuillage avec hystérésis

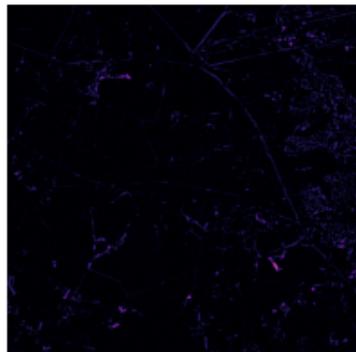


En augmentant le rayon, les lignes détectées deviennent plus épaisses et plus continues, mais il existe également un augmentation de la détection des faux contours.

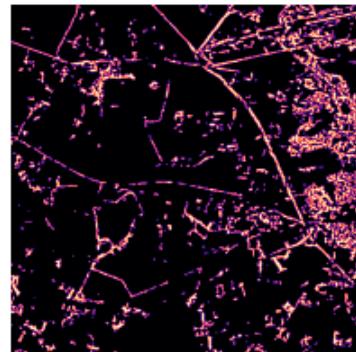
Original image



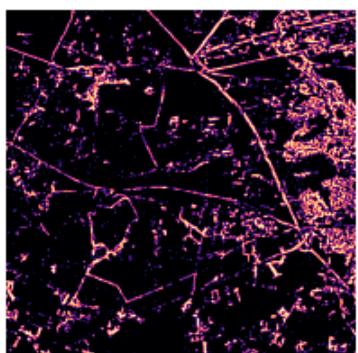
Tophat filter



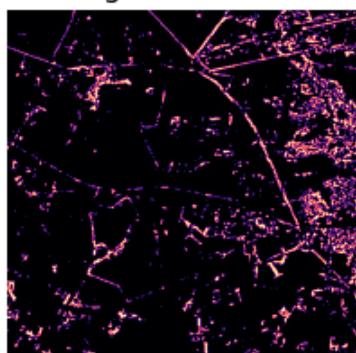
Hysteresis



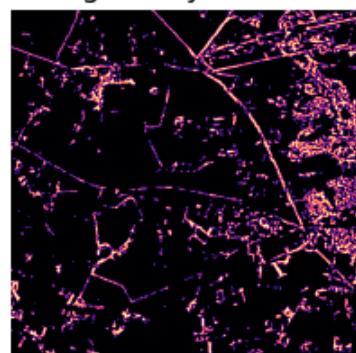
Low threshold



High threshold

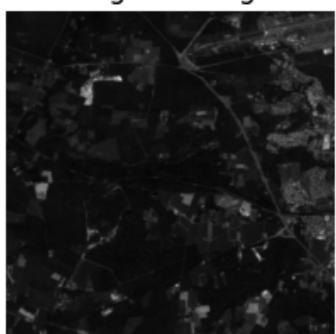


High + Hysteresis

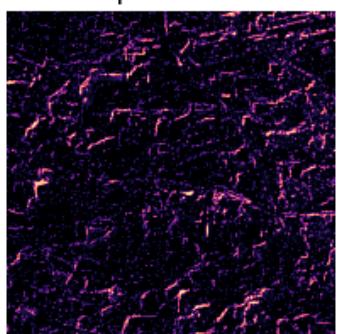


**radius = 5**

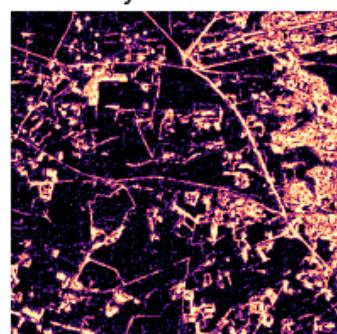
Original image



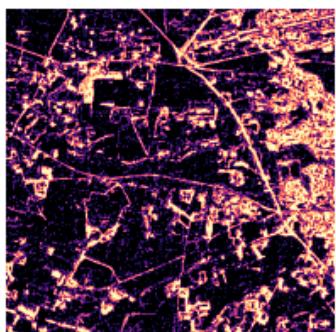
Tophat filter



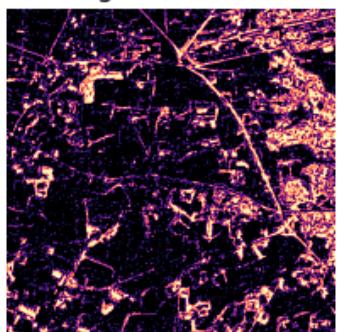
Hysteresis



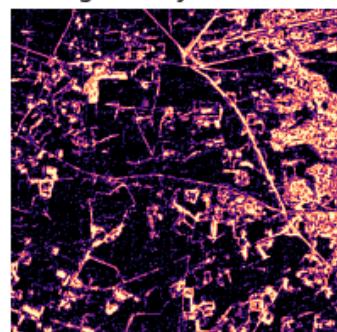
Low threshold



High threshold



High + Hysteresis

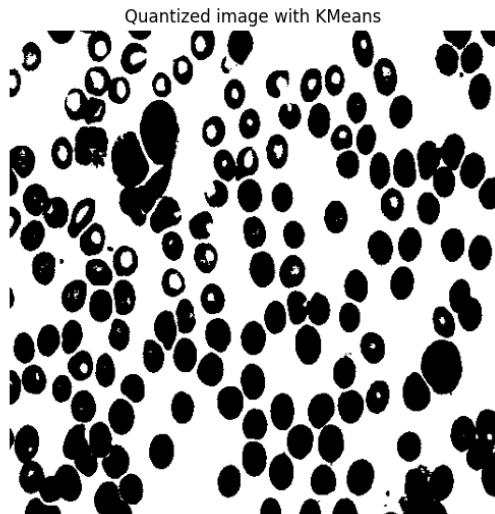


**radius = 8**

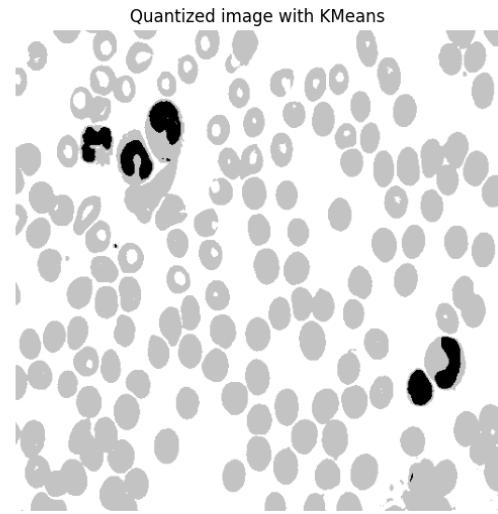
## 3 Segmentation par classification : K-moyennes

### 3.1 Image à niveaux de gris

L'algorithme des k-moyennes, pour  $k = 2$ , ne fait pas la différence entre les deux types de cellules différents, car l'une des classes est affectée aux cellules et l'autre à l'arrière-plan. Pour remédier à ce problème, nous augmentons le nombre de classes à 3.



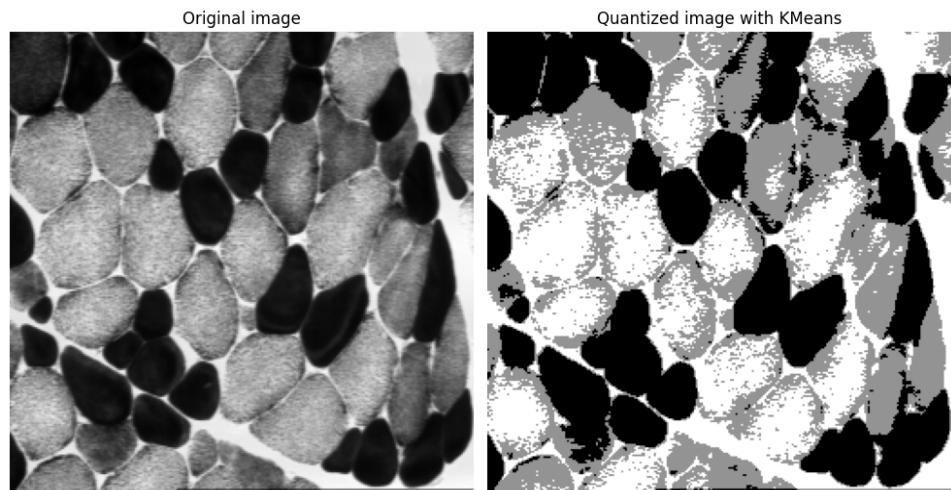
`n_classe = 2`



`n_classes=3`

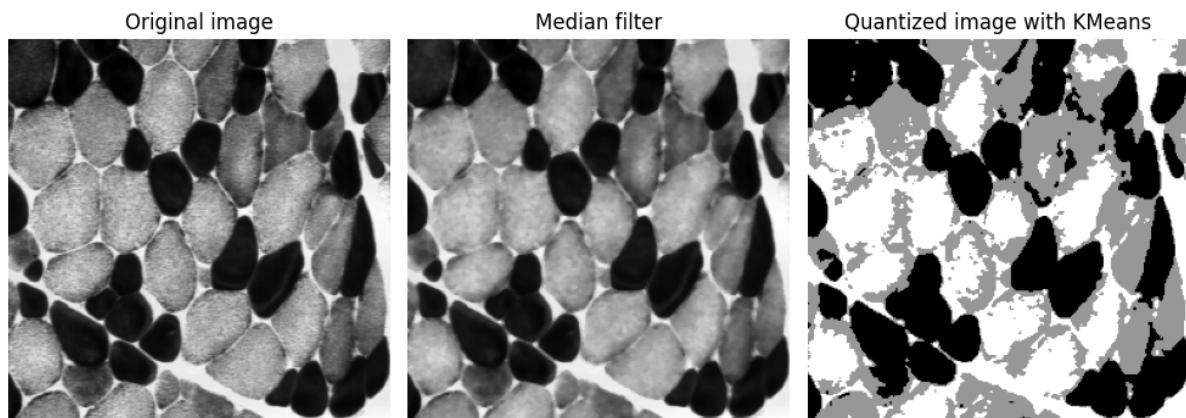
En exécutant l'algorithme k-means plusieurs fois après l'ajout de la 3ème classe, avec une initialisation aléatoire, nous observons que la classification obtenue ne change pas. Elle est stable.

Pour l'image muscle.tif, même si on la segmente en trois classes, dans le but de différencier le fond blanc, les fibres de couleur plus claire et celles de couleur plus foncée, l'algorithme des k-moyennes ne parvient pas à segmenter avec précision les fibres de couleur plus claire. Cela se produit parce qu'elles ont une structure granuleuse, par conséquent l'algorithme ne parvient pas à l'identifier comme un objet unique.



**n\_classes = 3**

Le filtrage de l'image avant la classification améliore les performances de l'algorithme k-means car cela permet de supprimer le bruit et lisser les textures les ce qui rend les objets plus homogènes.



### 3.2 Image en couleur

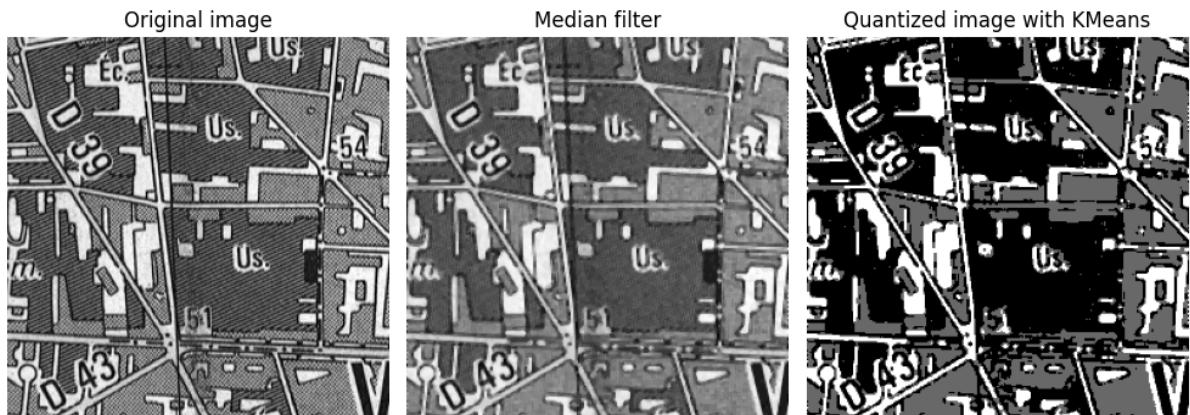
Après quantification, l'image est codée sur 10 couleurs seulement, donc même si l'on peut encore distinguer les fleurs, certaines informations sont perdues par rapport à l'image originale, par exemple des dégradés de couleurs se discrétilisent. De plus, seules les couleurs les plus fréquentes ont été conservées, tandis que les rares, comme le bleu, ont été perdus.



Pour un nombre de classe de 256, on obtient un rendu visuel similaire à celui de l'image codée sur 3 octets.



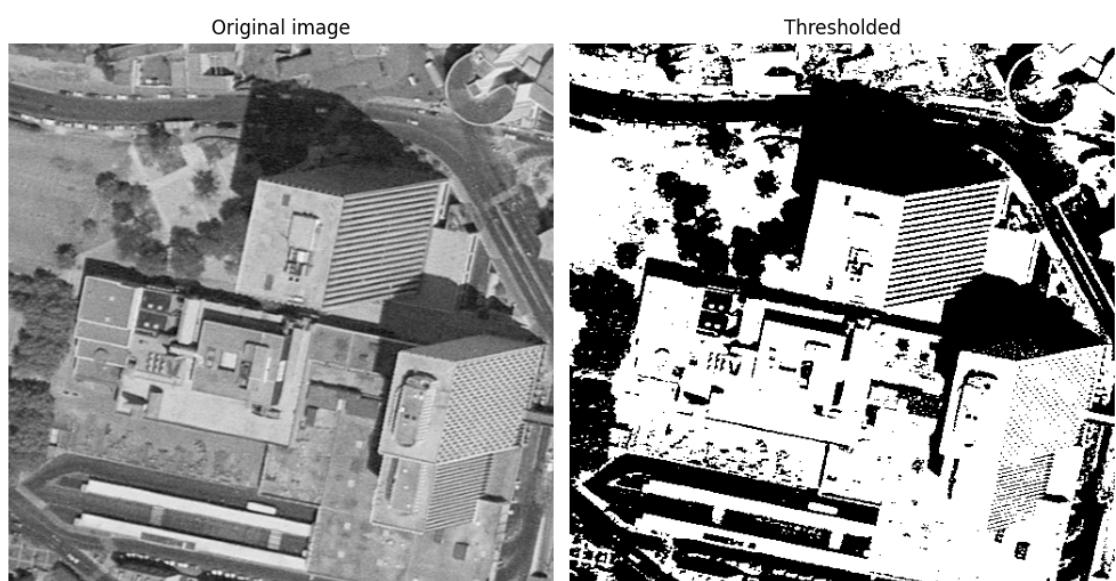
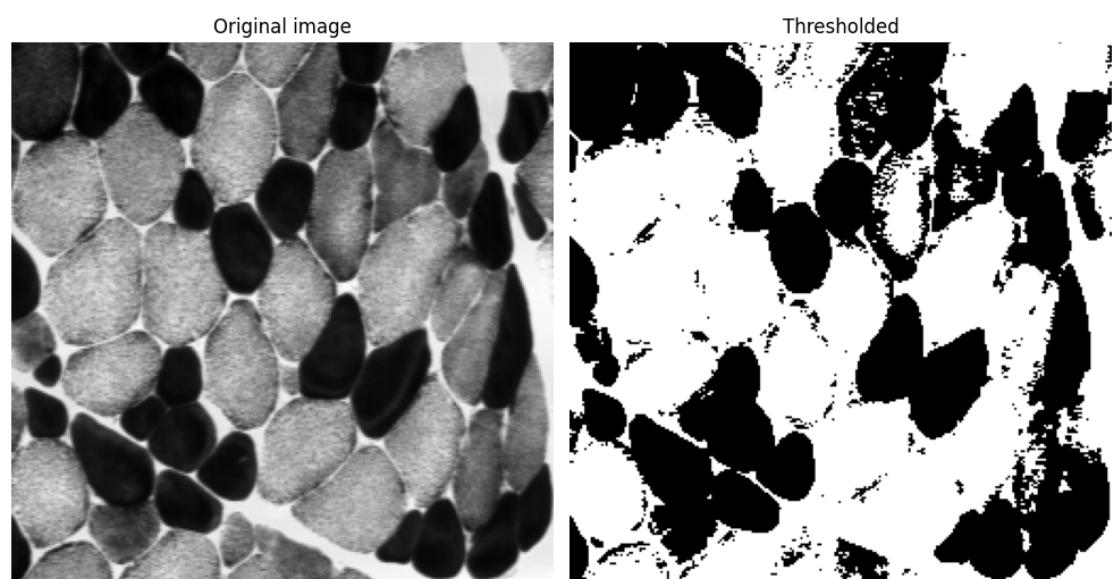
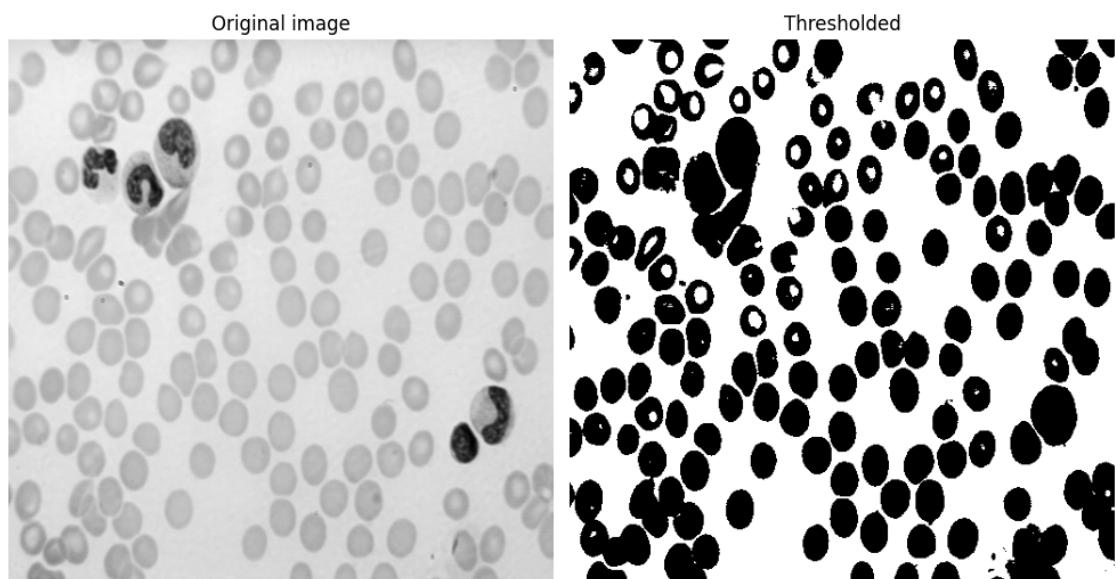
Pour retrouver les planches-mères de la carte carte.tif, on pourrait d'abord appliquer un filtre moyen pour lisser les textures hachurées et pointillées, rendant homogènes les zones appartenant à une même classe. Puis on applique l'algorithme k-means pour segmenter la carte en trois classes.



## 4 Seuillage automatique : Otsu

Dans le script python, le critère qu'on cherche à optimiser est la minimisation de la variance intra-classe.

On applique la méthode de segmentation d'Otsu à deux classes à différentes images en niveaux de gris, nous voyons qu'elle ne réussit pas toujours à distinguer les objets de l'arrière-plan ( exemple de 'muscle.tif').



Cette méthode semble marcher pour seuiller une image de norme du gradient :

