DURET Guillaume
BURGEVIN Valentin

GR IMA

TP de lignes de partage des eaux :

2018-2019

Introduction

Notre objectif est d'élaborer un algorithme de segmentation par ligne de partage des eaux via la méthode des files d'attentes hiérarchiques. Cet algorithme doit être capable de séparer et compter les grains de cafés sombres sur un fond claire (voir Figure 1). Sur cette image, les grains de café se touchent les uns les autres, et certains sont en partie en dehors de l'image ce qui rend difficile pour un algorithme de compter correctement le nombre de grains de café présents sur l'image. Il va donc falloir dans un premier temps créer des marqueurs à l'intérieur de chaque grain qui servira d'initialisation pour notre algorithme de segmentation par ligne de partage des eaux.

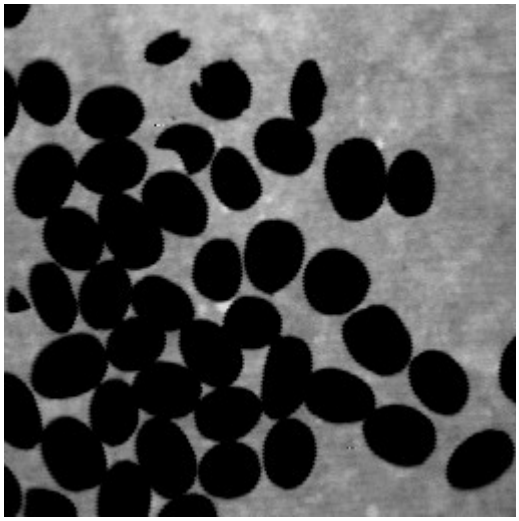

Figure 1: Image des grains de café à segmenter

I) Obtention des marqueurs

Nous devons donc d'abord obtenir les marqueurs initiaux de notre algorithme, c'est à dire un ensemble de point à l'intérieur de chaque grain que l'on affichera d'une couleur différente et auquel on associera un label unique pour chaque grain, ainsi qu'au fond plus clair de l'image qui aura lui aussi son propre marqueur.

Nous devons d'abord différencier le fond de l'image plus clair des grains de café. Comme les grains sont noirs sur un fond plus clair, nous pouvons simplement différencier les deux à l'aide d'un seuillage binaire.

```
#Variables
Seuil=20
#Import de l'image
img_grains = cv.imread('Image.png',cv.CV_8UC1)
#Seuillage
ret,img_seuil=cv.threshold(img_grains,Seuil,1,cv.THRESH_BINARY_INV)
```

La variable `Seuil` est la variable qui définit à partir de quelle valeur un pixel devient noir ou blanc. Comme les grains de café sont très sombres, on peut prendre un seuil très faible. Nous utilisons ensuite la fonction `threshold` d'opencv pour effectuer le seuillage, le dernier argument de la fonction permettant de faire un seuillage binaire (soit la valeur du pixel est au-dessus du seuil soit elle est en-dessous) et inversé afin d'avoir les grains en blanc(Figure 2).

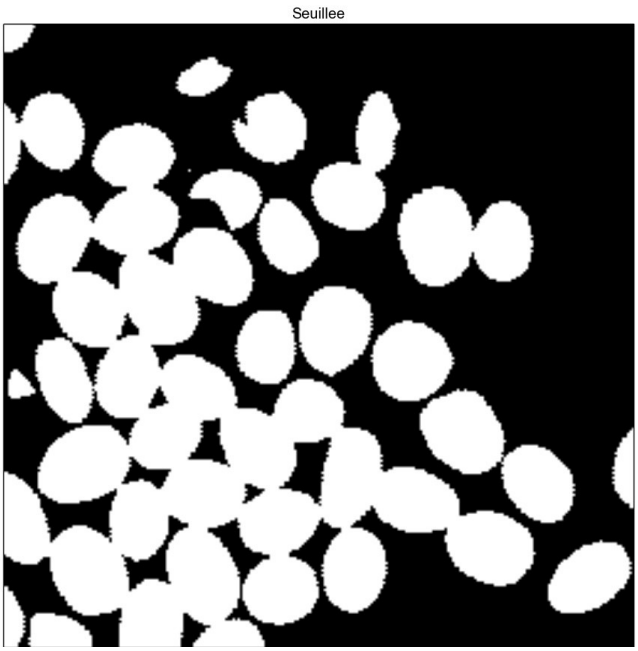

Figure 2: Image seuillée

À partir de cette image seuillée, nous allons chercher à obtenir la carte des distances de l'image, c'est à dire afficher en plus sombre les pixels de grains les plus éloignés des contours de ce dernier et en plus clairs les pixels les plus proches des contours.

```
#Carte des distances
img_distance=cv.distanceTransform(img_seuil,cv.DIST_L2,
cv.DIST_MASK_PRECISE) #Calcul de la carte des distances
img_distance_inv=(np.amax(img_distance)-img_distance) #Inversion de la
carte des distances
img_distance_final=cv.multiply(img_distance_inv.astype(np.uint8),img_seuil)
```

Nous utilisons la fonction `distanceTransform` d'opencv afin de réaliser cette carte des distances. Cependant nous obtenons cette carte des distances sur un fond blanc (Figure 3) ce qui ne permet pas de distinguer les contours.

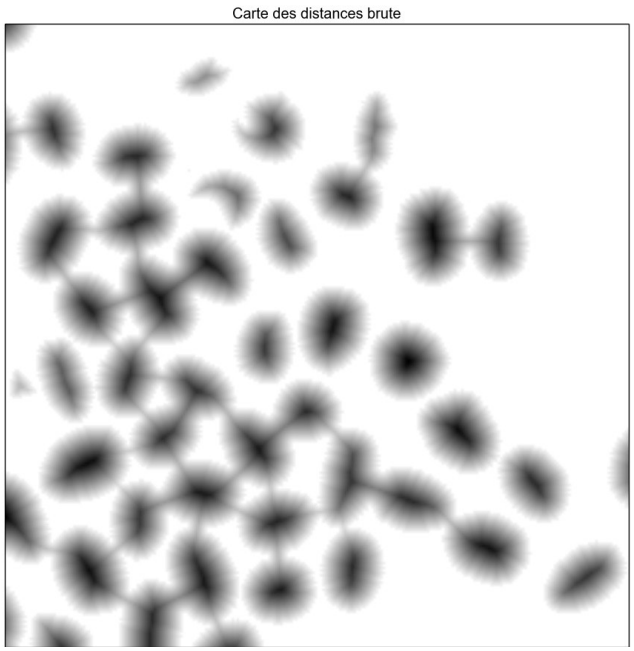

Figure 3: Carte des distances brute

Pour palier à ce problème nous inversons les couleurs de cette image (le blanc devient noir et le noir devient blanc) puis nous multiplions la matrice de l'image seuillée qui sert alors de masque et la matrice de la carte des distances inversée. Nous obtenons donc finalement la carte des distances sur fond noir (Figure 4) que nous utiliserons pour la suite.

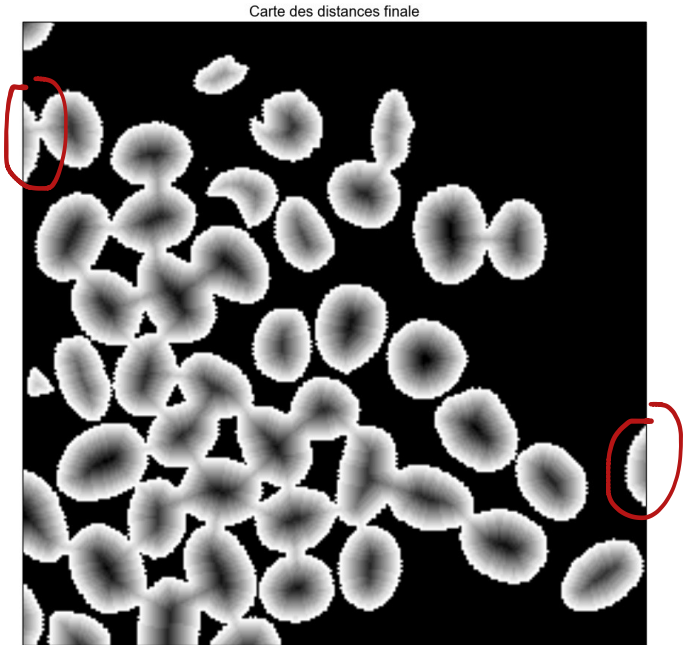

Figure 4: Carte des distances finale

Nous pouvons maintenant à partir de cette carte des distances obtenir les marqueurs de nos grains en prenant les point les plus sombres de chaque grain. Cependant, comme les grains ne sont pas tous de la même taille, les points les plus sombres pour chaque grain ne sont pas tous de la même valeur, nous ne pouvons donc pas faire un seuillage classique. La solution est alors d'utilisé le seuillage adaptatif, qui va séparer l'image en un certain nombre de carrés, et va adapter son seuil en fonction de l'intensité des pixels de chaque carré.

```
#Obtention des marqueurs des grains deimg_grains
img_dist_seuil_adapt=cv.adaptiveThreshold(img_distance_inv.astype(np.uin
t8),np.amax(img_distance_final),cv.ADAPTIVE_THRESH_GAUSSIAN_C,cv.THRESH_
BINARY_INV,19,2)
kernel_open = np.ones((2,2),np.uint8)
img_dist_open=cv.morphologyEx(img_dist_seuil_adapt,cv.MORPH_OPEN,kernel_
open)
kernel_dilate = np.ones((3,3),np.uint8)
img_dist_open=cv.dilate(img_dist_open,kernel_dilate,10)
```

Pour faire le seuillage adaptatif, nous utilisons la fonction `adaptativeThreshold` d'opencv sur la carte des distances. Il est à noter que l'avant-dernier paramètre de la fonction définit la taille des carrés, ici nous avons choisi 19 pixels afin d'avoir au moins un point blanc dans chaque grain (Figure 5).

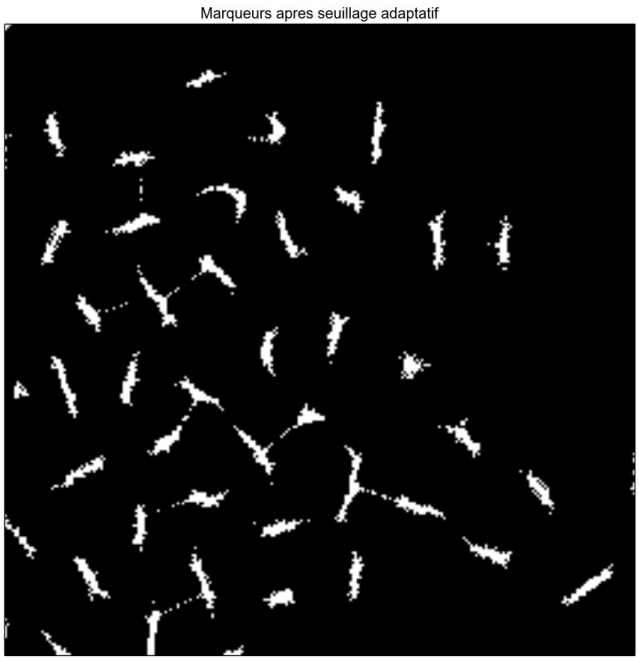

Figure 5: Marqueurs après seuillage adaptatif de la carte des distances

Cependant, le seuillage adaptatif n'est pas parfait et certains points sont isolés ou relient deux marqueurs. Pour les faire disparaître nous effectuons deux opérations morphologiques : une ouverture à l'aide de la fonction `morphologyEx` (en mettant `cv.MORPH_OPEN` comme second paramètre) pour faire disparaître les pixels blancs isolés et une dilatation via la fonction `dilate`

pour augmenter un peu la taille des marqueurs(Figure 6). Nous devons au préalable créer un élément structurant pour chacune de ces opérations. Pour l'ouverture nous choisissons un carré de deux sur deux, donc le plus petit possible afin de n'enlever que les points isolés. Pour la dilatation nous prenons un carré légèrement plus grand pour bien agrandir les marqueurs sans dépasser les contours.

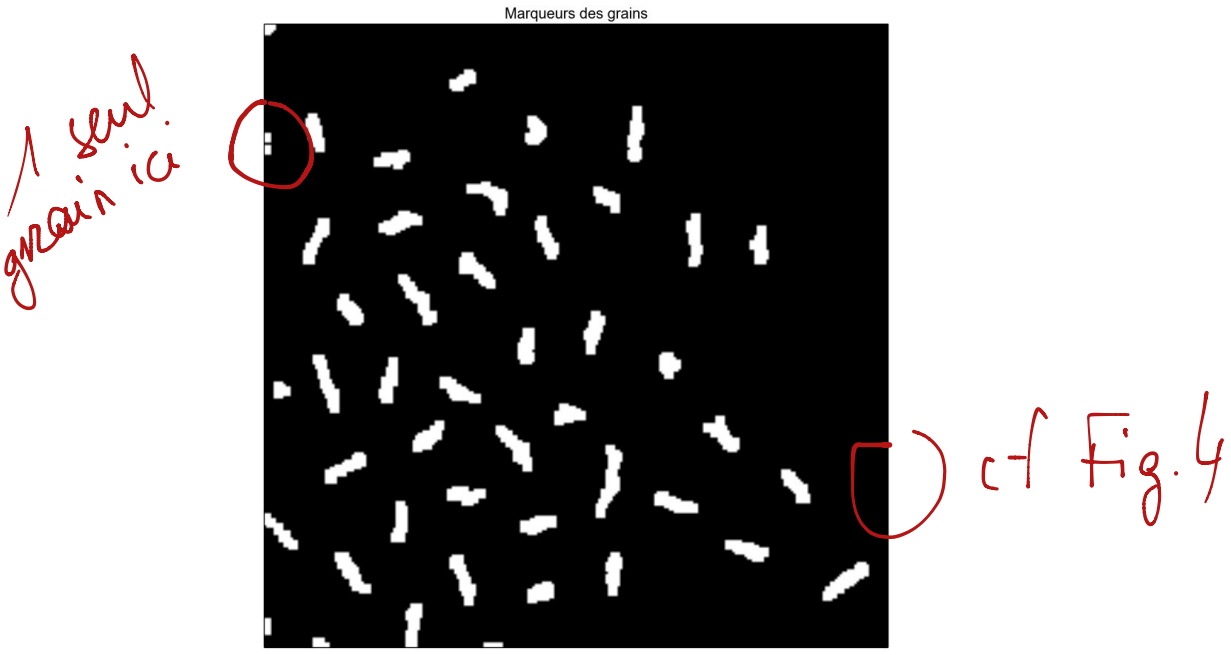

Figure 6: Marqueurs des grains après ouverture et dilatation

Il ne nous manque maintenant plus que les marqueurs du fond afin que nous puissions reconnaître l'espace entre les grains. Pour cela, nous utilisons simplement le seuillage inverse au premier que nous avons effectué (les grains en noir avec le fond blanc), et nous faisons une opération morphologique d'érosion afin de faire grossir les grains pour être sur que les marqueurs du fond ne chevauchent pas les grains.

```
#Obtention des marqueurs du fond
kernel_erode = np.ones((3,3),np.uint8)
ret,img_seuil_inv=cv.threshold(img_grains,Seuil,255,cv.THRESH_BINARY)
img_marq_fond=cv.erode(img_seuil_inv,kernel_erode,15)
```

Nous choisissons comme élément structurant pour l'érosion un carré de trois sur trois, il faut faire attention à ne pas faire trop grossir les grains et que le fond entre deux grains ne soit comblé.

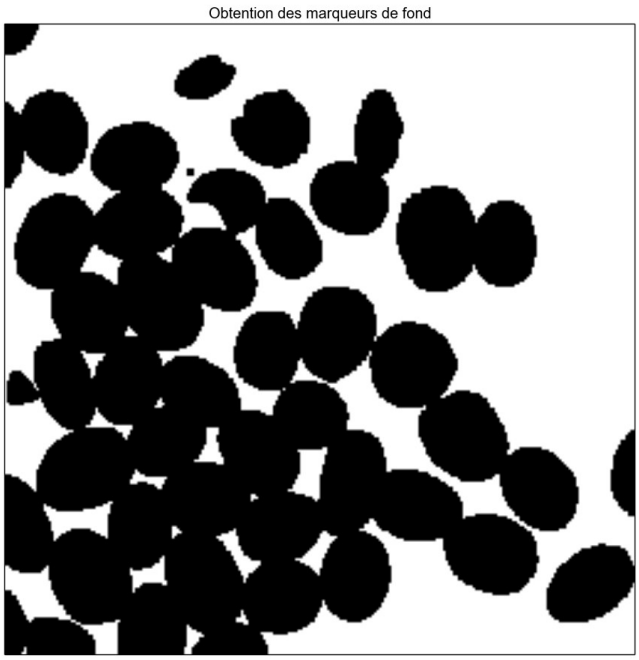

Figure 7: Marqueur du fond de l'image

La dernière étape pour obtenir nos marqueurs est de donner des labels unique à chaque marqueur des grains de café et un dernier pour tous les marqueurs du fond.

```
#attribution des labels des grains
ret, labels_grains = cv.connectedComponents(img_dist_open)
#Fusion avec le fond
img_final=labels_grains+((img_marq_fond/255)*(np.amax(labels_grains)+1))
```

La fonction `connectedComponents` identifie les différents éléments unis en fonction de leurs couleurs (ici blanc ou noir) et attribut à ces éléments un label unique.

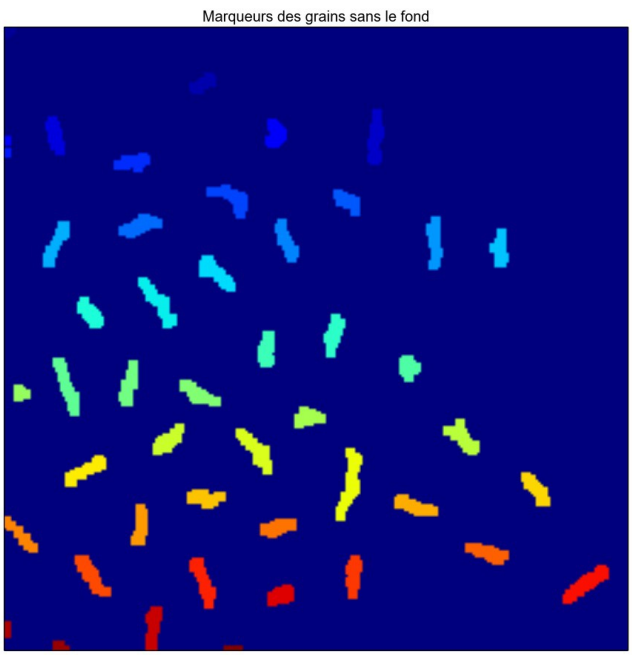

Figure 8 : Marqueurs des grains labellisés

Pour différencier le fond du reste des grains, on utilise l'image des marqueurs du fond comme un masque : en divisant par 255, les pixels des grains valent 0 et les pixels du fond valent 255. Il suffit de leur ajouter un nouveau label (par exemple le label maximum des grain plus un) et on a le fond dans un label différent.

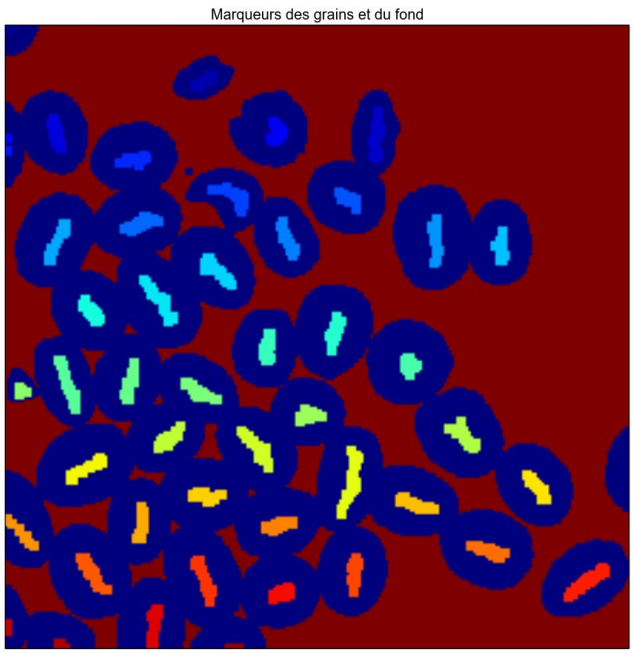

Figure 9: Marqueurs des grains et du fond labellisés

Nous avons finalement obtenus nos marqueurs ce qui correspond à l'initialisation de notre algorithme. La deuxième étape est donc de créer l'algorithme de ligne de partage des eaux.

II)Algorithme LPE

On applique ensuite l'algorithme de la ligne de partage des eaux à l'aide des marqueurs obtenue précédemment selon l'algorithme suivant :

Tant que la FAH n'est pas vide, faire:
{- extraire un jeton x de la FAH
- déterminer les pixels voisins de x d'étiquette nulle dans g
- pour chaque voisin y d'étiquette nulle, faire:
{ - assigner à y dans l'image g la même étiquette que x.
- insérer le jeton y dans la file d'attente de la FAH de priorité correspondant au niveau de gris de y dans f (si elle existe) ou à la file d'attente de plus forte priorité existant encore. }
}

On parcourt donc l'image en commençant par les pixels de teinte foncée et on repère autour de ce pixel les pixels qui n'ont pas de marqueur (=0) pour leur attribuer le marqueur du pixel central (voir code).

```
#initialisation de la FAH
FAH_x = []
FAH_y = []
for i in range(0,256):
    FAH_x.append([])
    FAH_y.append([])


for i in range(0, len(img_final)):
    for j in range (0, len(img_final[1])):
        if img_final[i,j]!=0:
            FAH_x[img_distance_final[i,j]].append(i)
            FAH_y[img_distance_final[i,j]].append(j)



vide=0
com=0
plt.ion()
while vide != 1:
    for i in range(0,256):
        if len(FAH_x[i])!=0:
            x=FAH_x[i][0]
            y=FAH_y[i][0]
            del FAH_x[i][0]
            del FAH_y[i][0]
            for k in range(-1,2):
                for l in range(-1,2):
                    if x+k<len(img_distance_final[1]) and x+k>=0 and
y+l<len(img_distance_final) and y+l>=0:
                        if img_final[x+k,y+l]==0:
                            img_final[x+k,y+l]=img_final[x,y]
                            FAH_x[img_distance_final[x+k,
y+l]].append(x+k)
                            FAH_y[img_distance_final[x+k,
y+l]].append(y+l)
                            com+=1
                            if com%500==0:
                                plt.imshow(img_final, 'jet')
                                plt.show()
                                plt.pause(0.00001)

            break
    vide=1
```

```
    for i in range (0,256):
        if len(FAH_x[i])!=0:
            vide=0
plt.ioff()
plt.imshow(img_final,'jet')
plt.show()
print ("Nombre de grains de cafe: ")
print (np.max(img_final)-1) #On retranche 1 car le marqueur 1 correspond
au fond et non a un grain de cafe
```

On obtient finalement la figure suivante :

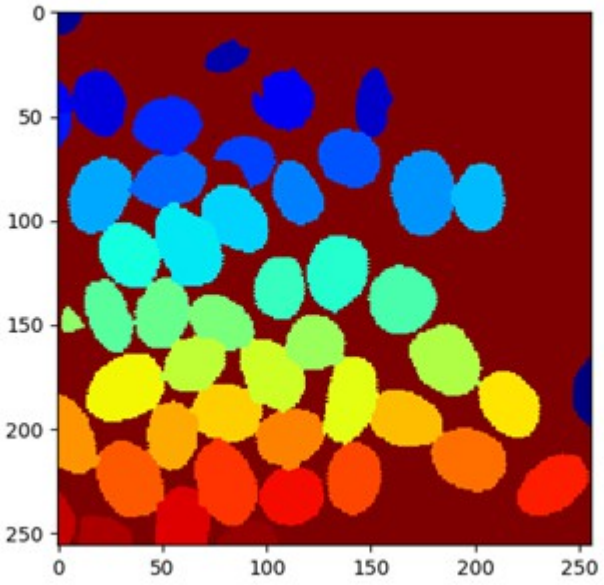

Figure 10 : Grains de café après LPE

On observe donc bien que chaque grain de café est associé à un marqueur différant (représenté par des couleurs différentes) ce qui est bien le but recherché

On peut ensuite à l'aide de la figure précédente on peut facilement déterminer les contours (dès qu'il y a des changements de teinte)

On obtient donc finalement la figure ci-dessous :

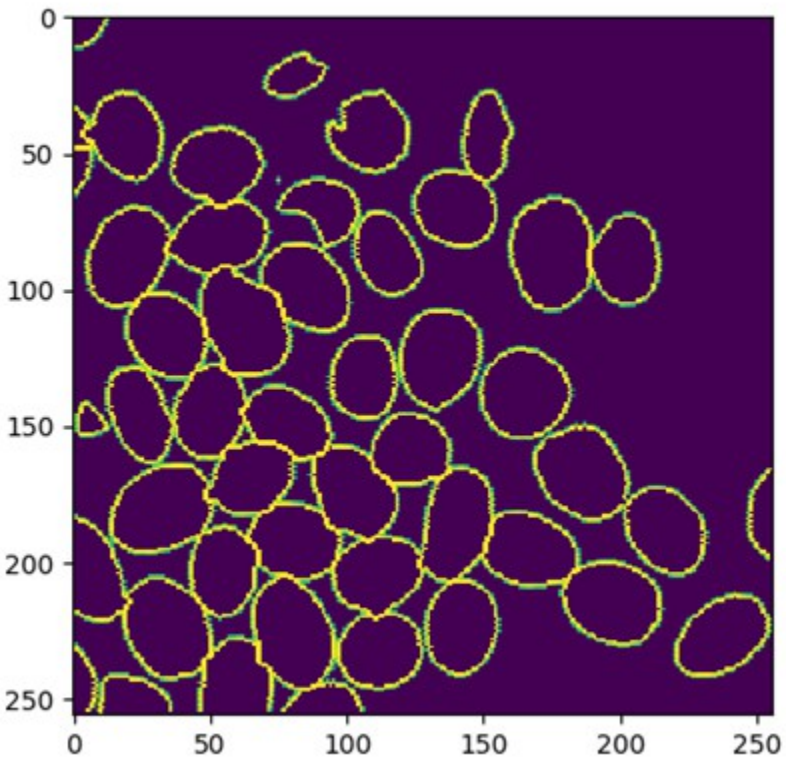

Figure 11 : Contours des grains de café obtenus par LPE

Nb de grains ?

Conclusion :

Pour conclure la LPE est une méthode globale qui peut être efficace s'il y a bcp d'objet qui sont parfois chevauché. Cependant la méthode met quand même du temps) à s'exécuter et elle nécessite au préalable des marqueur pour éviter la sur-segmentation .