MathieuVandecasteele_MedIm

November 20, 2018

1 Mathieu VANDECASTEELE - Master SID - Medical IMAGING

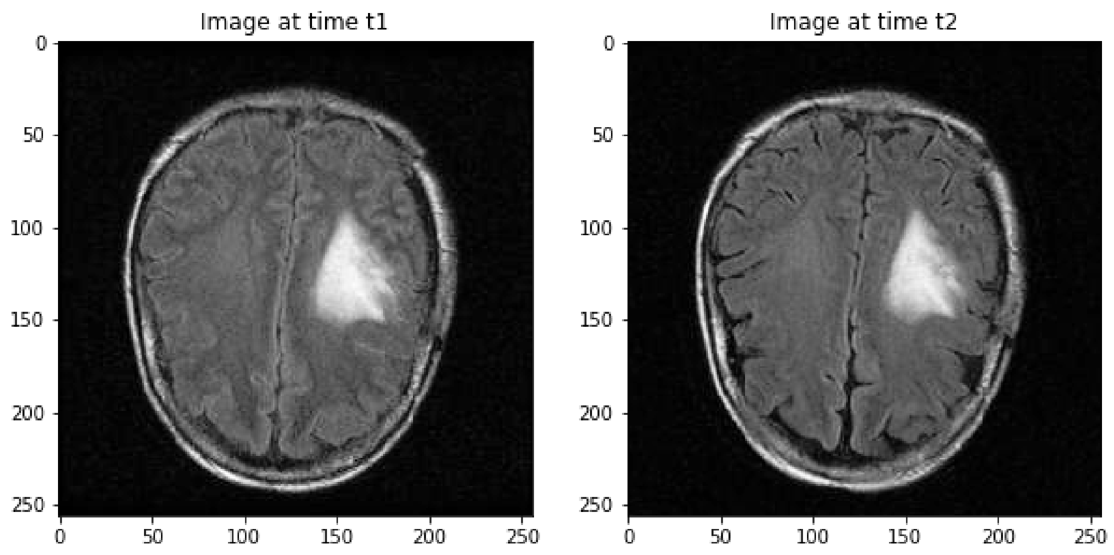
2 TP - Tumor Detection

The aim is to determine which method is the best for tumor detection by comparing results.
We have two images taken between a period t1 and a period t2

```
In [97]: # Import the primary needed packages
         import matplotlib.pyplot as plt
         import matplotlib.image as pimg
         import numpy as np
         import pandas as pd
         import PIL
         import cv2
         from scipy.ndimage.measurements import label
         import random
         import csv
         import operator
         from skimage import morphology, filters, measure,
         segmentation, color, io, data
```

Let's display the images :

```
In [144]: img_t1 = plt.imread('IRMcoupe17-t1.jpg')
          img_t2 = plt.imread('IRMcoupe17-t2.jpg')
          plt.figure(figsize=(10, 8))
          plt.subplot(1,2,1)
          plt.title("Image at time t1")
          plt.imshow(img_t1, cmap='Greys_r')
          plt.subplot(1,2,2)
          plt.title("Image at time t2")
          plt.imshow(img_t2, cmap='Greys_r')
          plt.show()
```



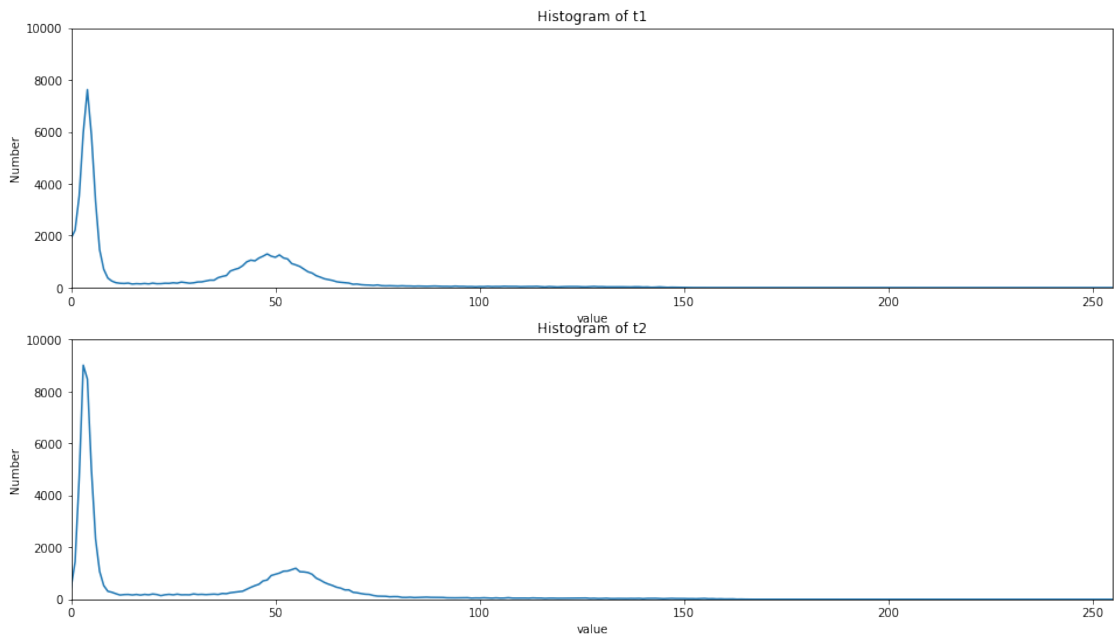
3 Before : some useful functions :

```
In [99]: # First : some useful functions

         # To plot histograms
         def histogramme(image):
             h = np.zeros(256,dtype=np.uint32)
             s = image.shape
             for j in range(s[0]):
                 for i in range(s[1]):
                     valeur = image[j,i]
                     h[valeur] += 1
             return h

         # To calculate evolutions at the end for each method
         def surface_evolution(tumor_t1, tumor_t2):
             surface_t1 = np.sum(tumor_t1)
             surface_t2 = np.sum(tumor_t2)
             print("Percentage of change (t2 - t1) : %2.2f%%" %
                   ((surface_t2 - surface_t1) / surface_t1 * 100))
             return (surface_t2 - surface_t1) / surface_t1 * 100

         def plot_contour_tumor(t1_tumor, t2_tumor, img_t1, img_t2):
             plt.figure(figsize=(10,4))
             plt.subplot(1,2,1)
             plt.title("Detected Tumor T1")
             plt.imshow(img_t1, cmap='Greys_r')
             plt.contour(t1_tumor, origin='lower')
             plt.subplot(1,2,2)
             plt.title("Detected Tumor T2")
             plt.imshow(img_t2, cmap='Greys_r')
             plt.contour(t2_tumor, origin='lower')
             plt.show()
```

4 METHOD 1 : BINARIZATION & LABELLING

First Method, we will use a simple binarization. It implies to find a thresold. To help us find it, we can plot the histograms for the images :

```
In [100]: plt.figure(figsize=(16,9))
          plt.subplot(2,1,1)
          plt.title("Histogram of t1")
          plt.plot(histogramme(img_t1))
          plt.axis([0,255,0,10000])
          plt.xlabel("value")
          plt.ylabel("Number")
          plt.subplot(2,1,2)
          plt.title("Histogram of t2")
          plt.plot(histogramme(img_t2))
          plt.axis([0,255,0,10000])
          plt.xlabel("value")
          plt.ylabel("Number")
          plt.show()
```

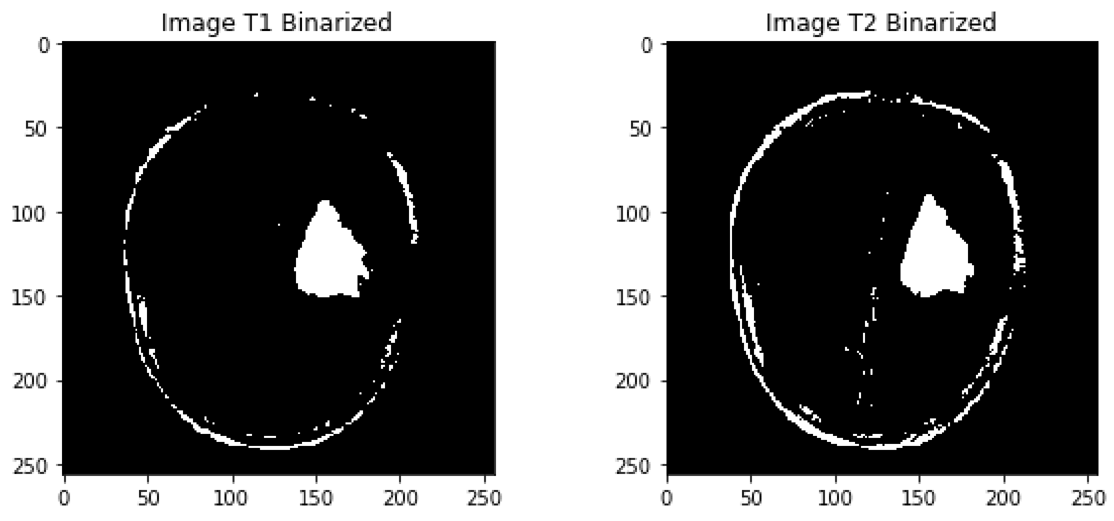


The tumor has a high grayscale value and a smaller surface than the other part of the brain, we can try a thresold = 80.

Binarization :

```
In [101]: # Function for binarization
          def binarize(image, thresold):
              n,p = image.shape
              result = np.zeros([n,p])
              for i in range(n):
                  for j in range(p):
                      if image[i,j] >= thresold:
                          result[i,j] = 256
                      else :
                          result[i,j] = 0
              return result
```

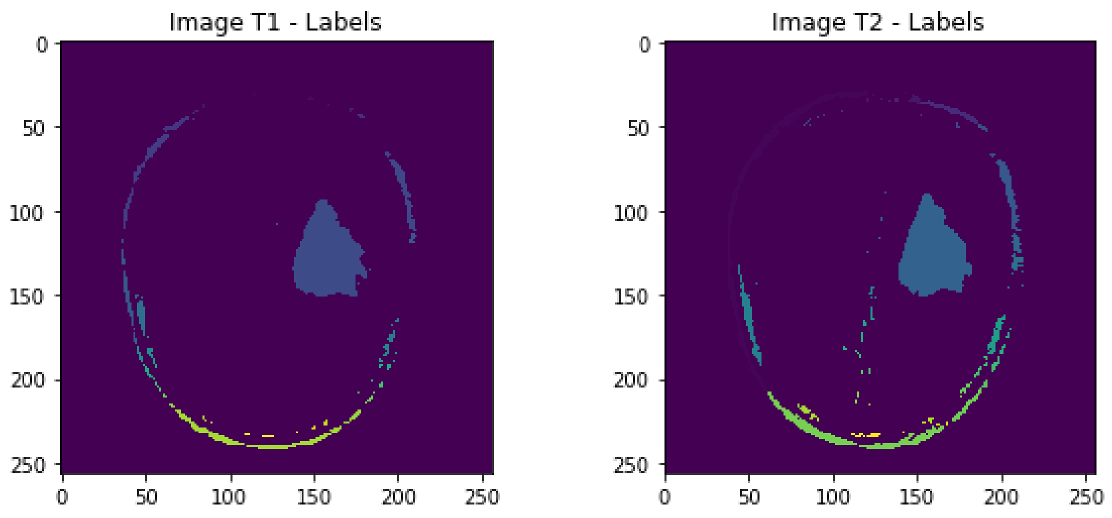
```
In [102]: # %% Binarization image 1
          # i choose 80, it seems to be a good value.
          thresold = 80
          imgt1b = binarize(img_t1,thresold)
          # %% Binarization image 2
          imgt2b = binarize(img_t2,thresold)
          # Display the binarized images
          plt.figure(figsize=(10, 4))
          plt.subplot(1,2,1)
          plt.title("Image T1 Binarized")
          plt.imshow(imgt1b, cmap='Greys_r')
          plt.subplot(1,2,2)
          plt.title("Image T2 Binarized")
          plt.imshow(imgt2b, cmap='Greys_r')
          plt.show()
```



We want to only keep the tumor, so we will use a labelling function to separate it from the other parts. The "Label" function will return an image labelled and a total number of the differents parts found.

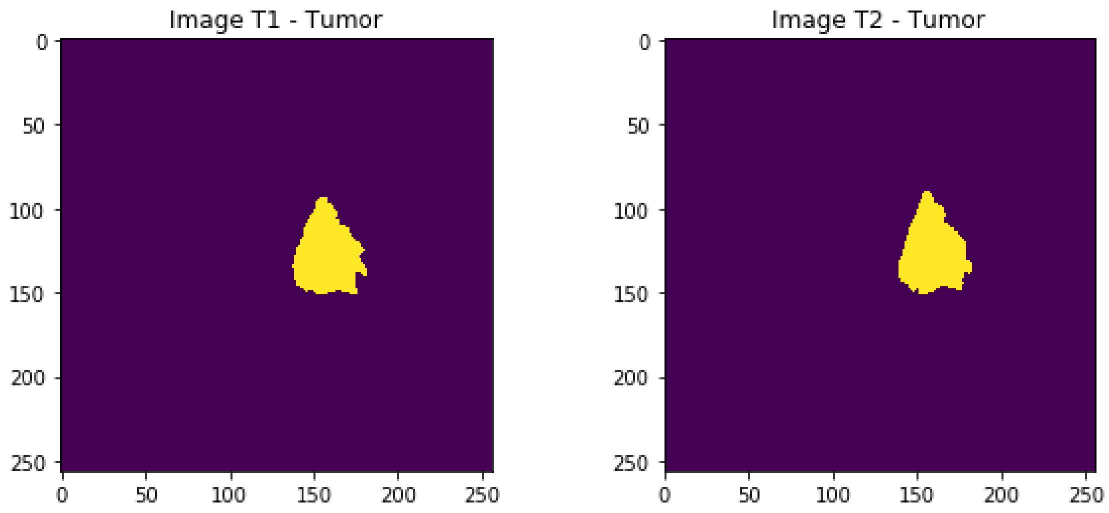
```
In [103]: # Use the label function :
          label_t1, nb_feature1 = label(imgt1b)
          label_t2, nb_feature2 = label(imgt2b)

          print("Number of features found for T1 : ",nb_feature1)
          print("Number of features found for T2 : ",nb_feature2)

          plt.figure(figsize=(10, 4))
          plt.subplot(1,2,1)
          plt.title("Image T1 - Labels")
          plt.imshow(label_t1)
          plt.subplot(1,2,2)
          plt.title("Image T2 - Labels")
          plt.imshow(label_t2)
          plt.show()
```

```
Number of features found for T1 :  60
Number of features found for T2 :  83
```



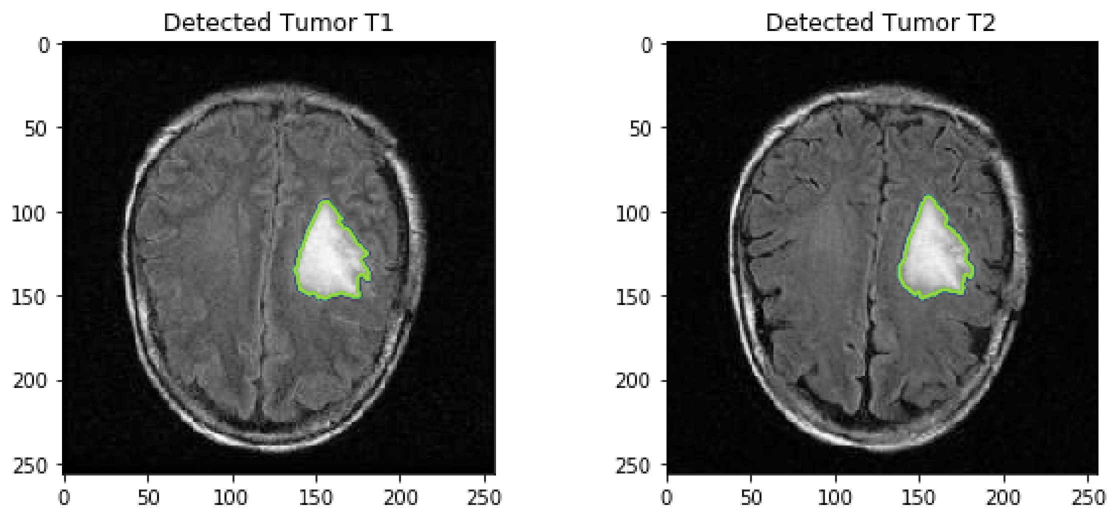
```
In [104]: # Display the tumor separated from the other part of the brain
          # we count pixels for each label and we take the highest
          # surface which is the tumor.

          def count_labels_and_return_tumor(labels,nb_feature):
              count = np.zeros((nb_feature+1, ))
```

```
        for i in range(nb_feature + 1):
            pos = labels == i
            count[i] = np.sum(pos.flatten())
        tumorlabel = np.argmax(count[1:])+1
        pos = labels == tumorlabel
        tumor = np.zeros((256, 256))
        tumor[pos] = 256
        return tumor

    # T1
    t1_tumor = count_labels_and_return_tumor(label_t1,nb_feature1)
    # T2
    t2_tumor = count_labels_and_return_tumor(label_t2,nb_feature2)


    # Display :
    plt.figure(figsize=(10, 4))
    plt.subplot(1,2,1)
    plt.title("Image T1 - Tumor")
    plt.imshow(t1_tumor)
    plt.subplot(1,2,2)
    plt.title("Image T2 - Tumor")
    plt.imshow(t2_tumor)
    plt.show()
```



To end with this method, we plot the contour of the tumor with the original images :

```
In [105]: plot_contour_tumor(t1_tumor, t2_tumor, img_t1, img_t2)
```



Let's make measurements and establish a percentage of the evolution of the tumor with thresold = 80:

```
In [106]: difference_binarization = surface_evolution(t1_tumor, t2_tumor)

Percentage of change (t2 - t1) : 1.16%
```

The method is efficient and simple but the problem is that it depends strongly from the thresold chosen and we can't really determine it with another method than an histogram. I have tried different thresolds and the results are quite different :

thresold	-> percentage (t2-t1)
70	-> 0.75%
75	-> 0.34%
80	-> 1.16%
85	-> 3.13%

5 METHOD 2 : FCM

We will use the FCM method which is a clustering method.

The cost function is defined as :

$$J = \sum_{i=1}^{C} \sum_{k=1}^{N} (\mu_{k,i})^m ||y_k - u_i||^2$$

with $Y = y_1, y_2, \ldots, y_N \in R^N$ the vector of all pixels.
with C the number of clusters
with N: pixel number
with u_i center of a cluster i
An iteration :
1.Calculate the new centers thanks to a matrix with degrees U

2.Update U with the new calculated centers.

Initialisation of U : random values. The decision label for each pixel is found with the maximum value.

```
In [107]: # Implementation of functions i will use for the FCM method.
          def initialize(n, p):
              U = np.random.random((n, p))
              U = (U - np.mean(U)) / np.std(U)
              return U

          def calc_center(U, x, m):
              C = np.zeros(U.shape[1])
              for i in range(U.shape[1]):
                  c_i = np.dot(np.power(U[:, i], m), x) / \
                  np.dot(np.power(U[:, i], m), np.ones((U[:, i].shape)))
                  C[i] = c_i
              print("centers : "+ str(C))
              return C

          def calc_u(U,C,m,x):
              d = np.zeros(U.shape)
              for k in range(len(x)):
                  for i in range(len(C)):
                      d[k, i] = np.linalg.norm(x[k] - C[i])
              for k in range(len(x)):
                  for i in range(len(C)):
                      U[k, i] = 1/np.sum(np.power(d[k,i]/d[k,j],2/(m - 1))
                                         for j in range(len(C)))
              return U

          def get_cluster(U):
              return np.argmax(U, axis=1)

          def calculate_the_loss(U, m, x, C):
              J = 0
              for i in range(len(C)):
                  J += np.dot(np.power(U[:, i], m), np.power(x - C[i], 2))
              return J

          def my_fcm(image,number_clusters, m_param,
                     epsilon, max_iteration):
              x = np.reshape(image, (image.shape[0] * image.shape[1],))
              U = initialize(x.shape[0], number_clusters)
              J_old = -1000
              J_new = 1000

              for i in range(max_iteration):
                  print("iteration : "+str(i))
```

```
            J_old = J_new
            C = calc_center(U,x,m_param)
            U = calc_u(U,C, m_param, x,)
            J_new = calculate_the_loss(U, m_param, x, C)
            if np.abs(J_new - J_old) <= epsilon:
                print("Found the optimal at the iteration " + str(i))
                break

        return U, C
```

According the previous histograms, we can determine 3 possible clusters (the tumor, the brain and the background), so we will consider C = 3 To explain the algorithm above, we stop if we reach an abs value smaller than a parameter epsilon, i have also decided to put a max_iteration number but it is not important because this number is never reached -> the optimal is found before.

We choose in general m=2. We will try to modify this parameter later.

```
In [108]: # parameters
          number_clusters = 3
          m_param = 2
          # parameters for the iterations
          epsilon = 1e-2
          max_iteration = 45

          #T1
          print("Let's train for image T1 : ")
          U_t1, C_t1 = my_fcm(img_t1,number_clusters, m_param,
                              epsilon, max_iteration)
          labels_t1 = get_cluster(U_t1)

          #T2
          print("Let's train for image T2 : ")
          U_t2, C_t2 = my_fcm(img_t2,number_clusters, m_param,
                              epsilon, max_iteration)
          labels_t2 = get_cluster(U_t2)
```

```
Let's train for image T1 :
iteration : 0
centers : [27.392668   27.42377613 27.48821081]


/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:24: DeprecationWarning: Calling n


iteration : 1
centers : [27.33072057 27.4551284  27.7128945 ]
iteration : 2
centers : [26.82786002 27.32202353 28.34776043]
iteration : 3
centers : [24.89949938 26.79659713 30.78625195]
```

```
iteration : 4
centers : [18.40894351 24.90785075 38.48170506]
iteration : 5
centers : [ 7.82136879 22.10594816 48.28581562]
iteration : 6
centers : [ 4.60182632 31.04576755 52.68520675]
iteration : 7
centers : [ 4.34688544 36.97923138 55.51209285]
iteration : 8
centers : [ 4.4026019  40.67751979 58.3200428 ]
iteration : 9
centers : [ 4.46272025 42.93779772 61.40247228]
iteration : 10
centers : [ 4.5066929  44.40999082 64.86685021]
iteration : 11
centers : [ 4.53920513 45.46336831 68.88333673]
iteration : 12
centers : [ 4.56511129 46.29202854 73.68417603]
iteration : 13
centers : [ 4.58775804 46.99179637 79.43519993]
iteration : 14
centers : [ 4.60947897 47.59893011 85.95953641]
iteration : 15
centers : [ 4.63171995 48.11460038 92.51005584]
iteration : 16
centers : [ 4.65443604 48.52740871 98.11610921]
iteration : 17
centers : [  4.67570083  48.83407957 102.2540345 ]
iteration : 18
centers : [  4.69320034  49.04569111 104.98837944]
iteration : 19
centers : [  4.70596458  49.18253229 106.66802256]
iteration : 20
centers : [  4.71444593  49.26658702 107.6548843 ]
iteration : 21
centers : [  4.71972254  49.31633532 108.21996304]
iteration : 22
centers : [  4.7228642   49.34505165 108.53888367]
iteration : 23
centers : [  4.72468272  49.36136396 108.7174457 ]
iteration : 24
centers : [  4.72571709  49.37053877 108.81698508]
iteration : 25
centers : [  4.72629922  49.3756684  108.8723406 ]
iteration : 26
centers : [  4.72662479  49.37852631 108.90308442]
iteration : 27
centers : [  4.7268062   49.38011531 108.92014692]
```

```
iteration : 28
centers : [  4.72690707  49.38099775 108.92961268]
iteration : 29
centers : [  4.72696309  49.38148749 108.93486287]
iteration : 30
centers : [  4.72699418  49.38175918 108.93777454]
iteration : 31
centers : [  4.72701143  49.38190987 108.93938919]
Found the optimal at the iteration 31
Let's train for image T2 :
iteration : 0
centers : [30.0370472  30.13675265 30.06728114]
iteration : 1
centers : [29.7518207  30.14658276 29.86899623]
iteration : 2
centers : [29.24178501 30.81441496 29.71007565]
iteration : 3
centers : [27.28750964 33.40116529 29.1020515 ]
iteration : 4
centers : [20.73911847 41.89534636 26.94565665]
iteration : 5
centers : [ 9.13827493 53.94218121 23.01621868]
iteration : 6
centers : [ 4.74434252 59.13463986 31.75411493]
iteration : 7
centers : [ 4.38709928 61.99178026 38.46285718]
iteration : 8
centers : [ 4.47231741 64.76409959 43.54474158]
iteration : 9
centers : [ 4.57633306 68.25068607 47.08762999]
iteration : 10
centers : [ 4.66040631 72.81721994 49.40933993]
iteration : 11
centers : [ 4.72246316 78.72539339 50.95738508]
iteration : 12
centers : [ 4.76881016 86.189507   52.05969561]
iteration : 13
centers : [ 4.80602426 94.77892568 52.9097263 ]
iteration : 14
centers : [  4.83963123 102.96751056  53.57615436]
iteration : 15
centers : [  4.87145854 109.37894482  54.0736521 ]
iteration : 16
centers : [  4.89918772 113.73213694  54.42375498]
iteration : 17
centers : [  4.92050926 116.43138084  54.65567046]
iteration : 18
centers : [  4.935226   118.0167085   54.80076282]
```

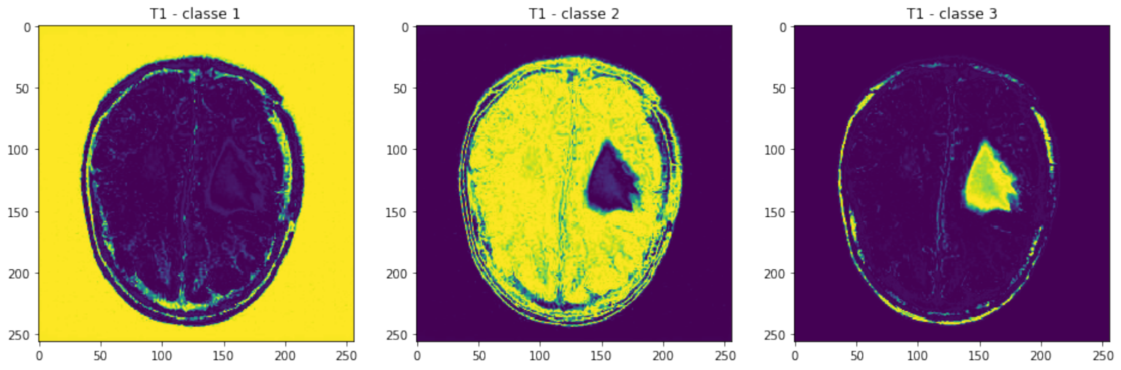
```
iteration : 19
centers : [  4.94459835 118.91952261  54.88749979]
iteration : 20
centers : [  4.95024444 119.42500467  54.93771421]
iteration : 21
centers : [  4.9535242  119.70543839  54.96617959]
iteration : 22
centers : [  4.95538623 119.86025575  54.98210498]
iteration : 23
centers : [  4.95642867 119.94550024  54.99094396]
iteration : 24
centers : [  4.95700743 119.99237082  54.99582673]
iteration : 25
centers : [  4.95732719 120.01812245  54.99851666]
iteration : 26
centers : [  4.95750336 120.03226512  54.99999622]
iteration : 27
centers : [  4.95760026 120.04003047  55.00080931]
iteration : 28
centers : [  4.95765351 120.04429371  55.00125592]
iteration : 29
centers : [  4.95768276 120.0466341   55.00150116]
iteration : 30
centers : [  4.95769882 120.04791886  55.00163581]
Found the optimal at the iteration 30
```

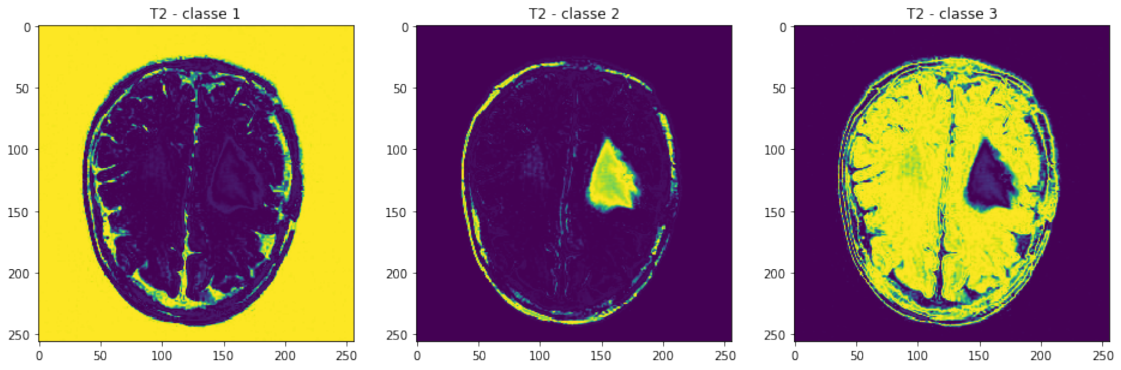
Let's plot now the matrices of appearance for each clusters.

```
In [109]: plt.figure(figsize=(16, 9))

          ax = plt.subplot(1, 3, 1)
          plt.imshow(np.reshape(U_t1[:, 0], (256, 256)))
          plt.title("T1 - classe 1")

          ax = plt.subplot(1, 3, 2)
          plt.imshow(np.reshape(U_t1[:, 1], (256, 256)))
          plt.title("T1 - classe 2")

          ax = plt.subplot(1, 3, 3)
          plt.imshow(np.reshape(U_t1[:, 2], (256, 256)))
          plt.title("T1 - classe 3")

          plt.figure(figsize=(16,9))
          ax = plt.subplot(1, 3, 1)
          plt.imshow(np.reshape(U_t2[:, 0], (256, 256)))
          plt.title("T2 - classe 1")
```

```
ax = plt.subplot(1, 3, 2)
plt.imshow(np.reshape(U_t2[:, 1], (256, 256)))
plt.title("T2 - classe 2")

ax = plt.subplot(1, 3, 3)
plt.imshow(np.reshape(U_t2[:, 2], (256, 256)))
plt.title("T2 - classe 3")
```

```
Out[109]: Text(0.5, 1.0, 'T2 - classe 3')
```

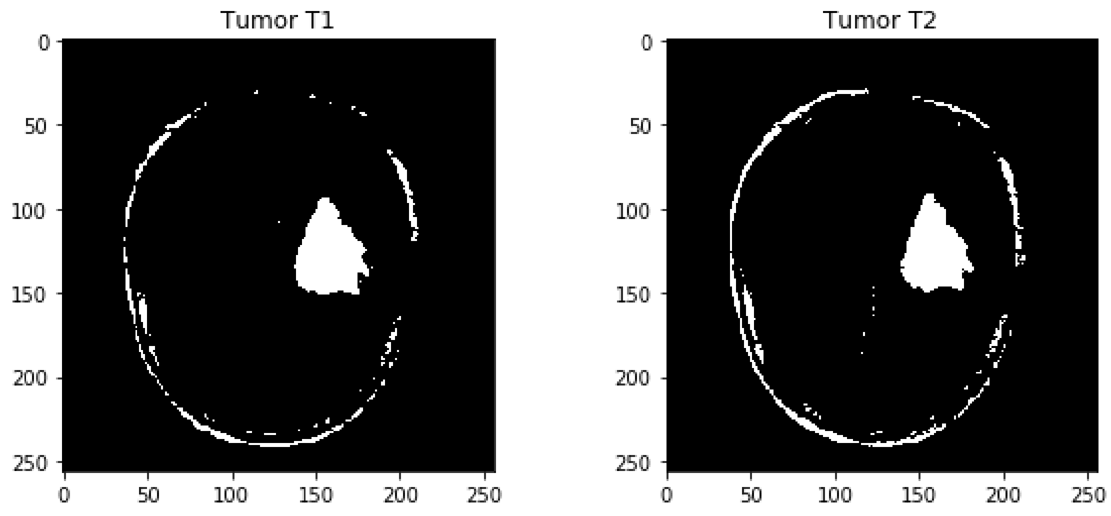




For t1 , the class 3 corresponds to the tumor and it's the class 2 for t2. We use them and the correct labels :

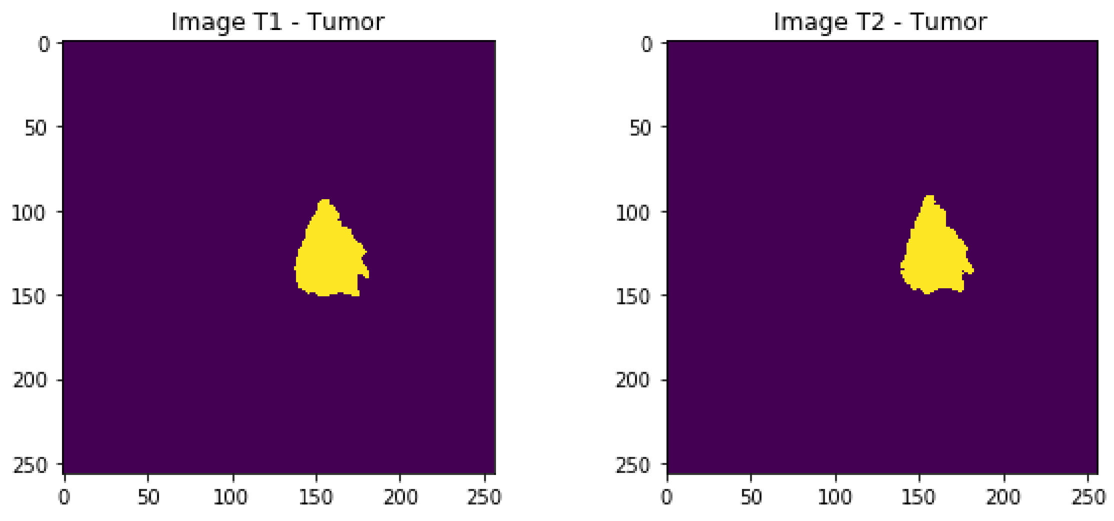
```
In [110]: labels_t1 = np.reshape(labels_t1, (256, 256))
          labels_t2 = np.reshape(labels_t2, (256, 256))
          plt.figure(figsize=(10, 4))
          plt.subplot(121)
          # I display for t1 the labels corresponding to the third class :
          # value : 2.
          plt.imshow(labels_t1 == 2, cmap='Greys_r')
          plt.title('Tumor T1')
```

```
         plt.subplot(122)
         # I display for t1 the labels corresponding to the second class :
         # value : 1.
         plt.imshow(labels_t2 == 1, cmap='Greys_r')
         plt.title('Tumor T2')

Out[110]: Text(0.5, 1.0, 'Tumor T2')
```

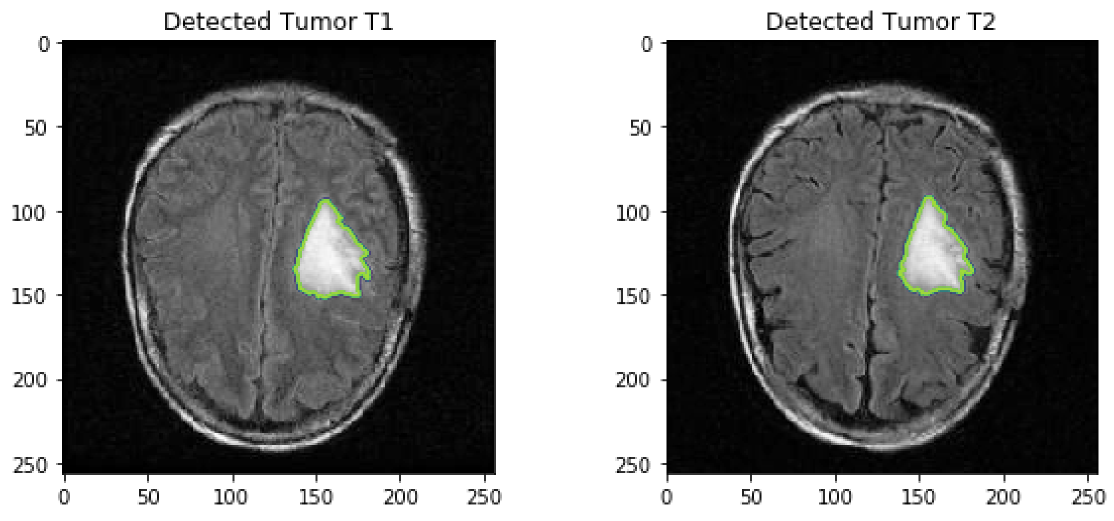


Now we are at the same state than Method 1 so let's do a labelling and measure the evolution between T1 and T2.

```
In [111]: t1_fcm = labels_t1 == 2
          t2_fcm = labels_t2 == 1
          label_t1, nb_feature1 = label(t1_fcm)
          label_t2, nb_feature2 = label(t2_fcm)

In [112]: # T1
          t1_tumor = count_labels_and_return_tumor(label_t1,
                                                   nb_feature1)
          # T2
          t2_tumor = count_labels_and_return_tumor(label_t2,
                                                   nb_feature2)


          # Display :
          plt.figure(figsize=(10, 4))
          plt.subplot(1,2,1)
          plt.title("Image T1 - Tumor")
          plt.imshow(t1_tumor)
          plt.subplot(1,2,2)
          plt.title("Image T2 - Tumor")
          plt.imshow(t2_tumor)
          plt.show()
```



```
In [113]: plot_contour_tumor(t1_tumor, t2_tumor, img_t1, img_t2)
```



```
In [114]: difference_fcm = surface_evolution(t1_tumor, t2_tumor)

Percentage of change (t2 - t1) : -6.23%
```

I tried to change m = 5 and the result is -6.98%. It is about the same and we can say that is is useless if we take a m > 2. Moreover we loose precision because when you plot the matrices of appearances, you can observe that there are more noise than m=2.

We found a tumor smaller than method 1 which was an increasing. Here, the tumor seems to be reduced.

6 METHOD 3 : MARKOV RANDOM FIELD

To save computer time, this method can re-use the previous centers calculated with FCM to initialise the matrix μ

Method to make the segmentation :
We will use σ as the standard deviation of the image
1.The energy of each cluster for each pixel is calculated.
2.The cluster of the pixel is the cluster with the minimum energy.
3.Update of μ and σ.

```
In [115]: #Implementation

          def calc_energy(image, labels, mu, sigma, x, y, beta):
              U1 = np.array([(image[x, y] - mu[i]) ** 2 / (2 * sigma[i])
                             for i in range(len(sigma))])
              U2 = []
              for i in range(len(mu)):
                  u = 8
                  u -= i == labels[max(x-1, 0), max(y-1, 0)]
                  u -= i == labels[max(x-1, 0), y]
                  u -= i == labels[max(x-1, 0), min(y+1, 255)]
                  u -= i == labels[x, max(y-1, 0)]
                  u -= i == labels[x, min(y+1, 255)]
                  u -= i == labels[min(x+1, 255), max(y-1, 0)]
                  u -= i == labels[min(x+1, 255), y]
                  u -= i == labels[min(x+1, 255), min(y+1, 255)]
                  U2.append(beta * u)
              U = np.array([U1[i] + U2[i] for i in range(len(sigma))])
              return U

          def mu_sigma(image, labels):
              mu = []
              sigma = []
              label_range = np.unique(labels)
              label_range = [int(i) for i in label_range]
              for i in label_range:
                  m = np.mean(image[labels == i])
                  mu.append(m)
                  s = np.std(image[labels == i])
                  sigma.append(s)
              return mu, sigma

          def markov_random_field(image, init_label, mu, sigma, beta,
                                  max_iteration, t, verbose=False):
              if verbose:
                  plt.figure()
                  plt.title("Evolution with iterations : " + str(t))
                  plt.imshow(init_label)
```

```
    old_labels = init_label
    for i in range(max_iteration):
        new_labels = np.zeros((256, 256))
        for x in range(image.shape[0]):
            for y in range(image.shape[0]):
                U = calc_energy(image, old_labels, mu,
                                sigma, x, y, beta)
                new_labels[x, y] = np.argmin(U)
        mu, sigma = mu_sigma(image, new_labels)
        print("iteration: " + str(i))
        if verbose:
            plt.figure()
            plt.imshow(new_labels)
        if np.sum(new_labels != old_labels) == 0:
            print('optimal found ! iteration : ' + str(i))
            break
        old_labels = new_labels.copy()
    return old_labels
```

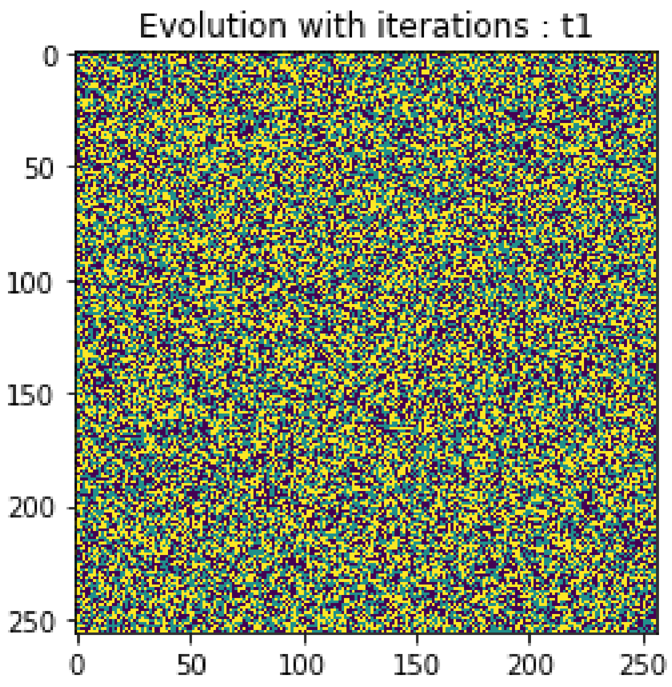
We consider that the optimal is reached when there is no change between the two last iterations. Let's use it, we will choose beta = 0.01 for now, we will study few values later :

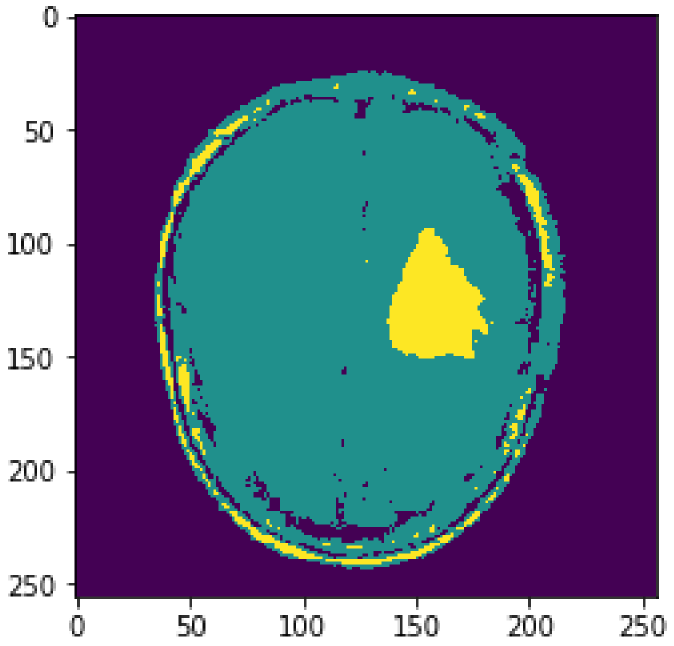
```
In [116]: # PARAMETERS
          old_labels_t1 = np.random.randint(0, 3, (256, 256))
          old_labels_t2 = np.random.randint(0, 3, (256, 256))
          # We take this beta, we will play with its value later.
          beta = 0.01
          # Standard Deviation
          sigma_t1 = 3*[np.std(img_t1)]
          sigma_t2 = 3*[np.std(img_t2)]
          # I use the previous calculated centers (see method 2)
          mu_t1 = np.array(np.sort(C_t1))
          mu_t2 = np.array(np.sort(C_t2))
          # I define a max_iteration but it is just a security.
          max_iteration = 10
```

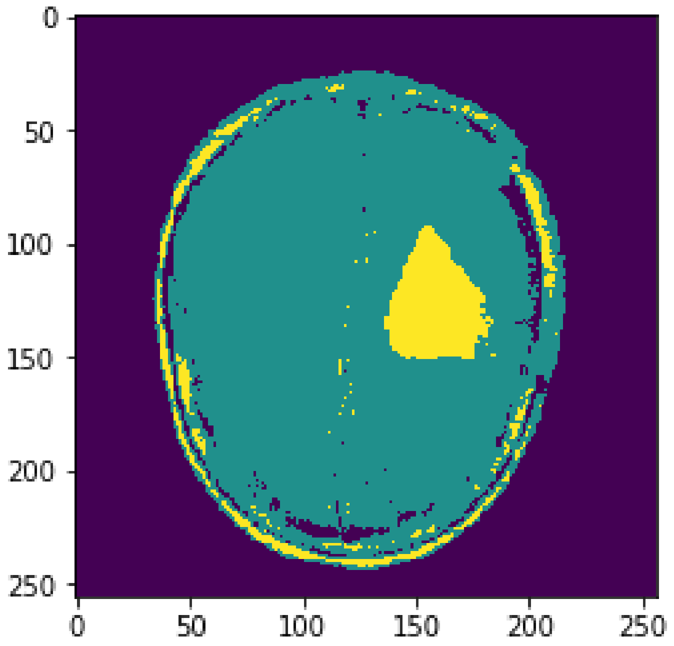
```
In [117]: # LAUNCH TRAINING
          print("wait a bit..")
          new_labels_t1 = markov_random_field(img_t1, old_labels_t1, mu_t1,
                                              sigma_t1, beta,
                                              max_iteration, 't1', verbose=True)
          print("wait a bit..")
          new_labels_t2 = markov_random_field(img_t2, old_labels_t2, mu_t2,
                                              sigma_t2, beta,
                                              max_iteration, 't2', verbose=True)
```

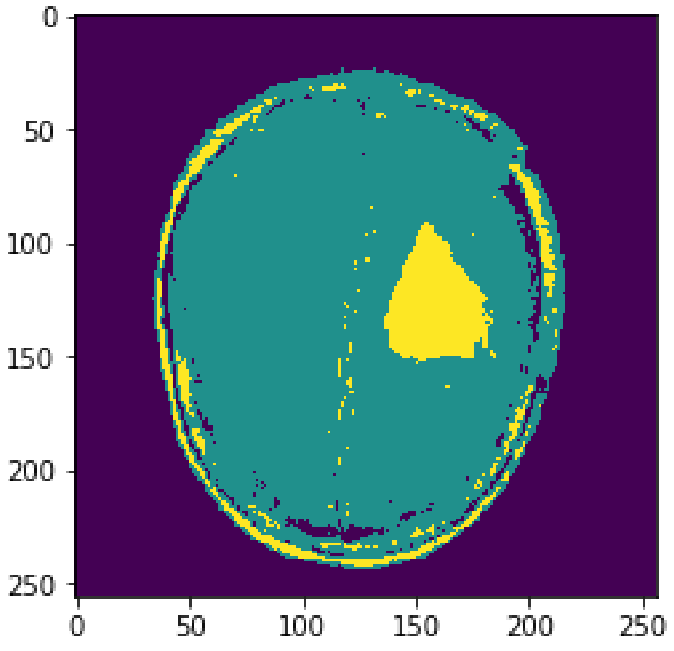
```
wait a bit..
iteration: 0
iteration: 1
```

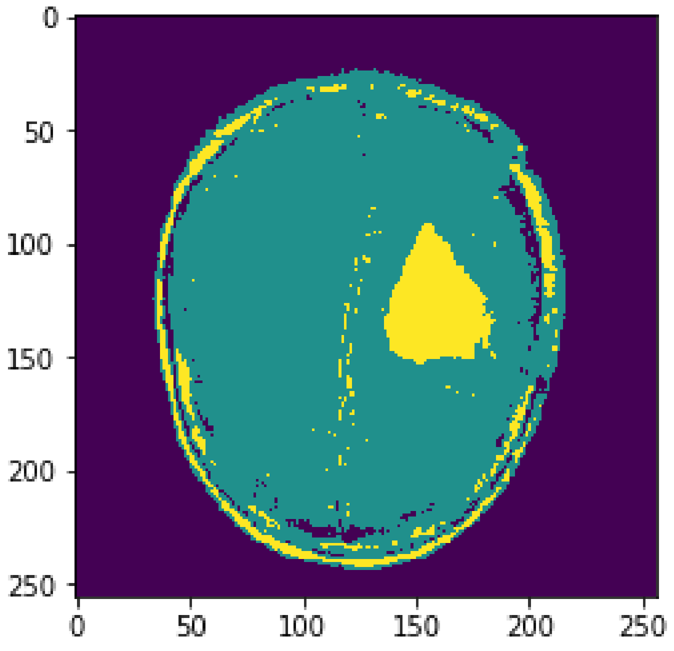
```
iteration: 2
iteration: 3
iteration: 4
iteration: 5
iteration: 6
optimal found ! iteration : 6
wait a bit..
iteration: 0
iteration: 1
iteration: 2
iteration: 3
iteration: 4
iteration: 5
optimal found ! iteration : 5
```

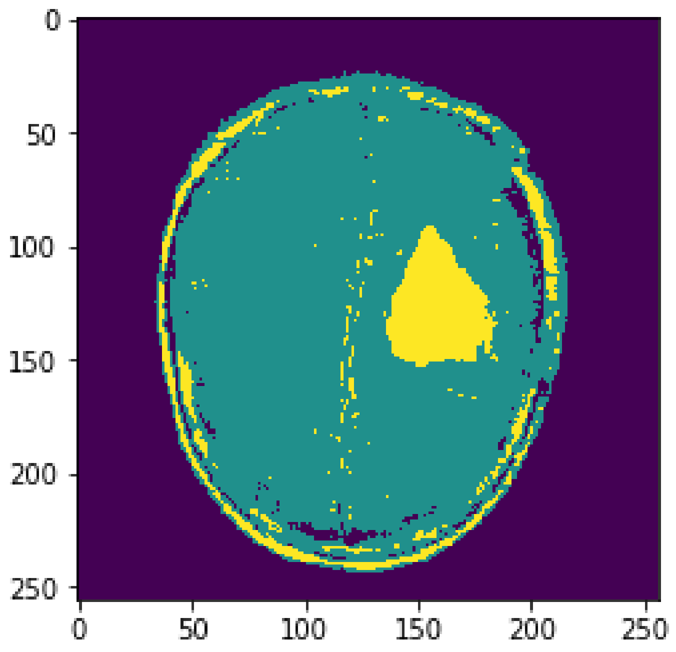


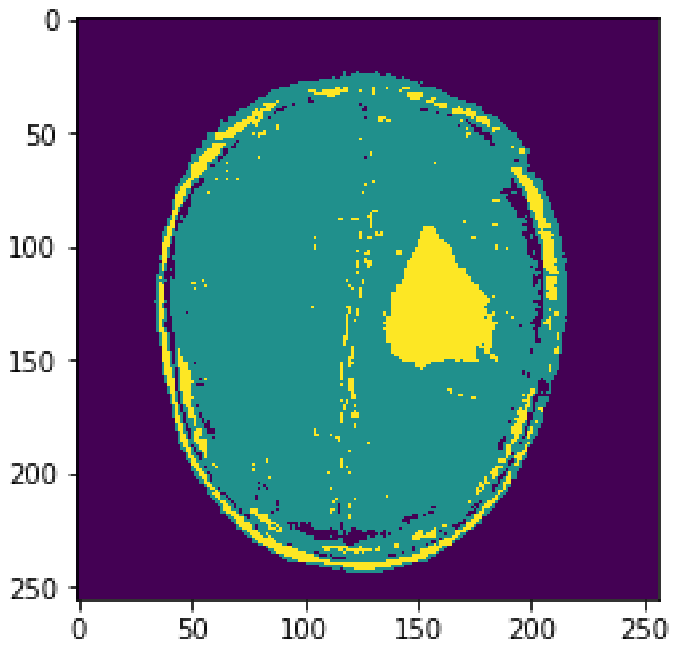

0
50
100
150
200
250
0
50
100
150
200
250

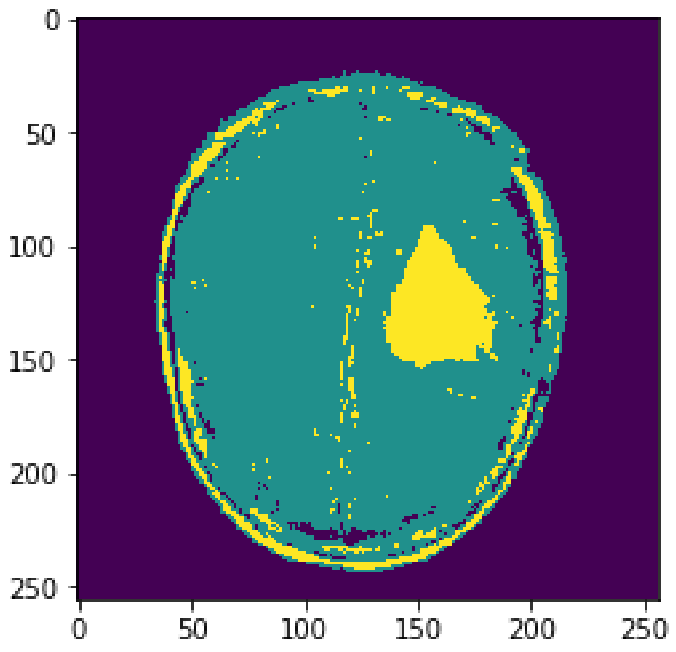

0
50
100
150
200
250
0
50
100
150
200
250

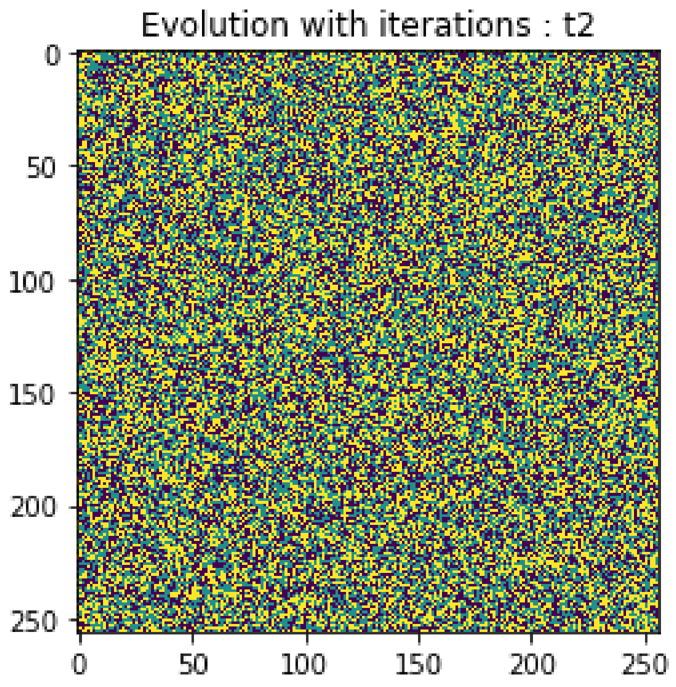


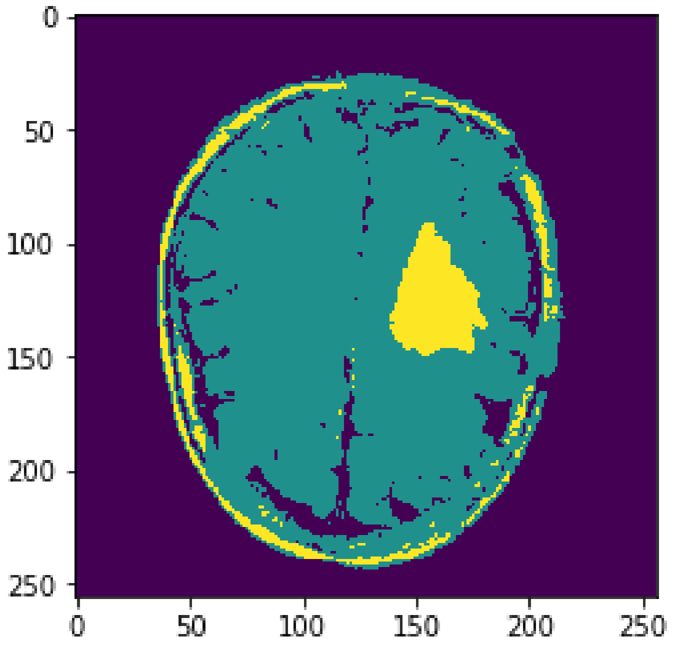


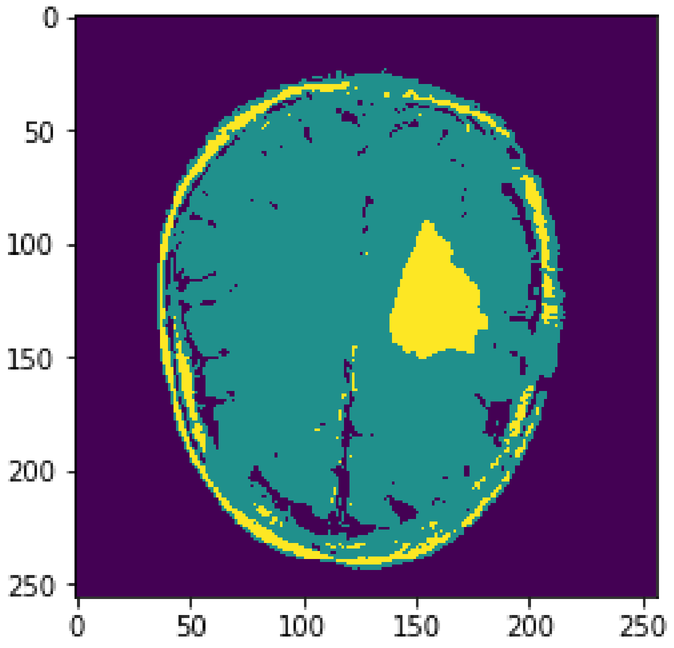

0
50
100
150
200
250
0
50
100
150
200
250

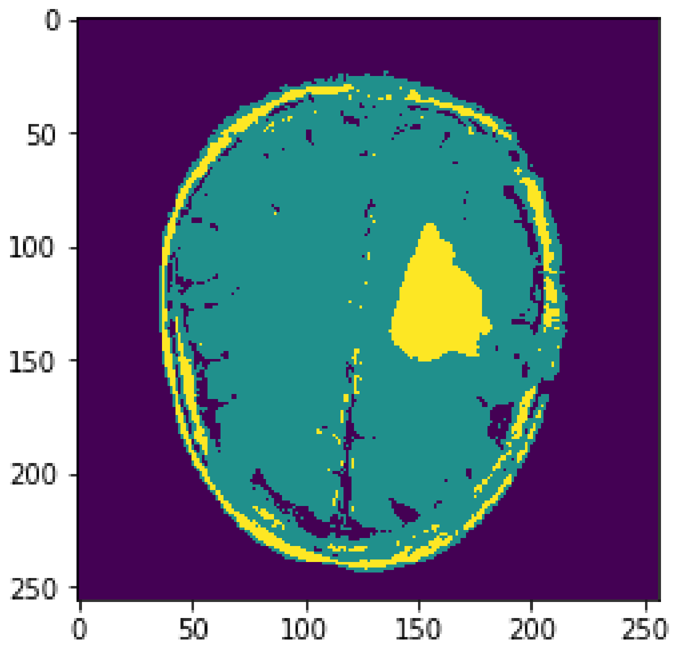

0
50
100
150
200
250
0
50
100
150
200
250


0
50
100
150
200
250
0
50
100
150
200
250

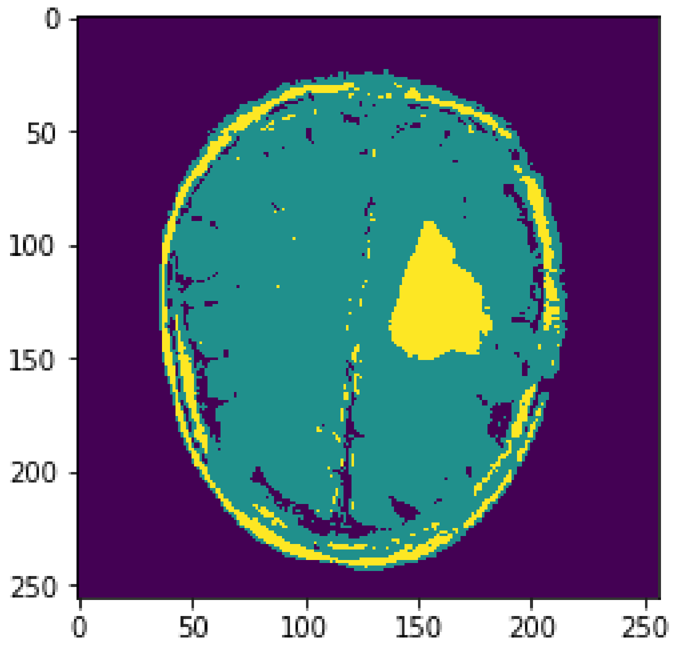

Evolution with iterations : t2
0
50
100
150
200
250
0
50
100
150
200
250


0
50
100
150
200
250
0
50
100
150
200
250

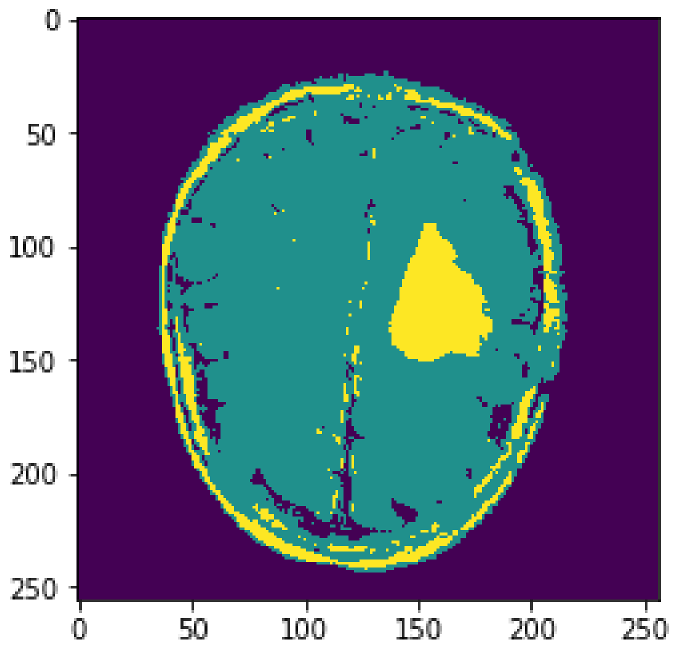

0
50
100
150
200
250
0
50
100
150
200
250

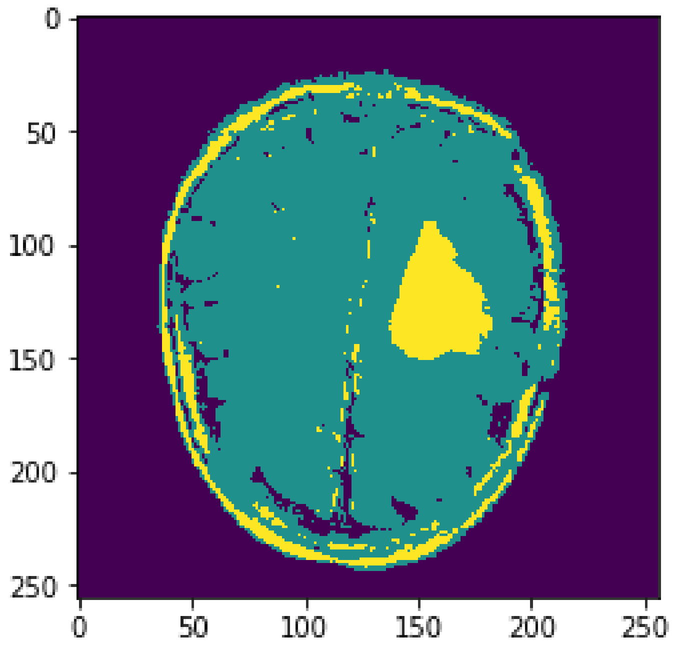


0
50
100
150
200
250
0
50
100
150
200
250



0
50
100
150
200
250
0
50
100
150
200
250

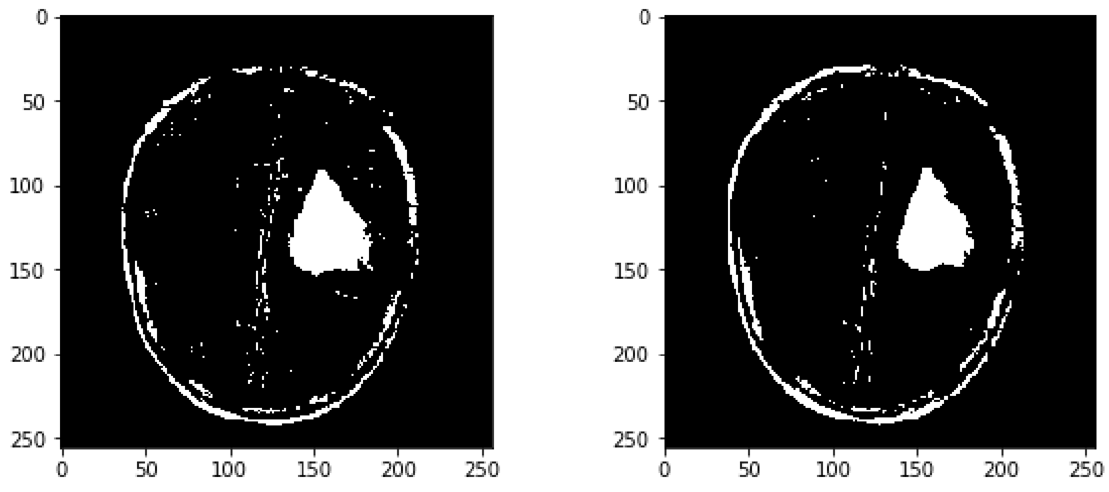




```
In [118]: # I take the label of the yellow color, let's display the result :
          t1_markov = new_labels_t1 == 2
          t2_markov = new_labels_t2 == 2
```

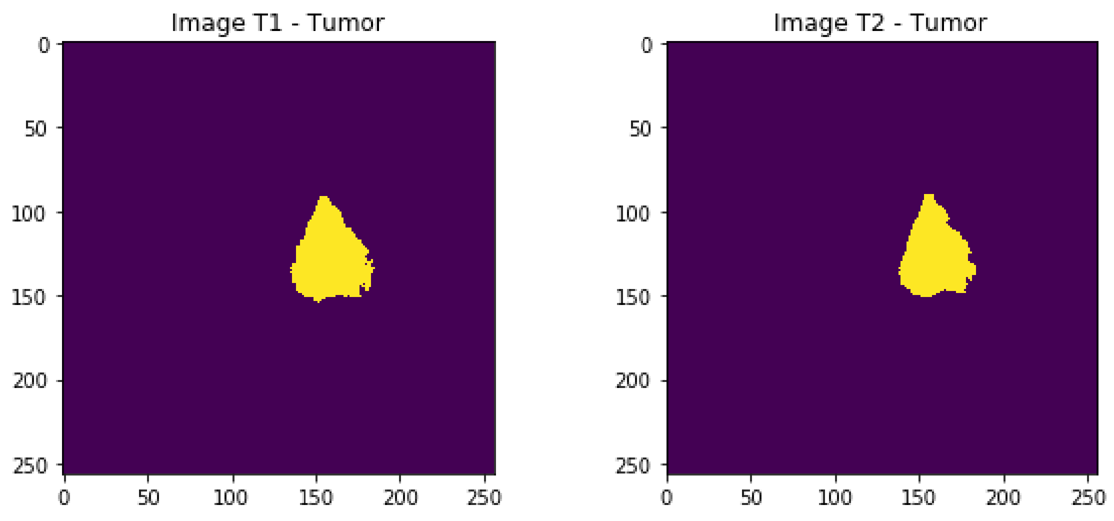
```
         label_t1, nb_feature_t1 = label(t1_markov)
         label_t2, nb_feature_t2 = label(t2_markov)
         plt.figure(figsize=(10, 4))
         plt.subplot(121)
         plt.imshow(t1_markov, cmap='Greys_r')
         plt.subplot(122)
         plt.imshow(t2_markov, cmap='Greys_r')
```

```
Out[118]: <matplotlib.image.AxesImage at 0x1c3ce3e550>
```

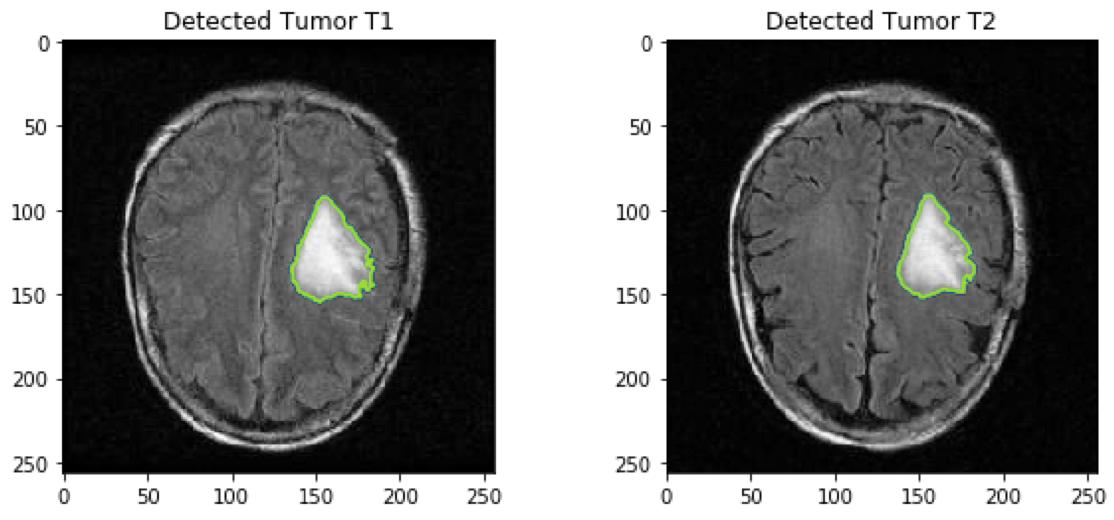


We just have to do a labelling and make the measures :

```
In [119]: # T1
          t1_tumor = count_labels_and_return_tumor(label_t1,nb_feature_t1)
          # T2
          t2_tumor = count_labels_and_return_tumor(label_t2,nb_feature_t2)

          # Display :
          plt.figure(figsize=(10, 4))
          plt.subplot(1,2,1)
          plt.title("Image T1 - Tumor")
          plt.imshow(t1_tumor)
          plt.subplot(1,2,2)
          plt.title("Image T2 - Tumor")
          plt.imshow(t2_tumor)
          plt.show()
```



```
In [120]: plot_contour_tumor(t1_tumor, t2_tumor, img_t1, img_t2)
```



```
In [121]: difference_markov = surface_evolution(t1_tumor, t2_tumor)

Percentage of change (t2 - t1) : -10.13%
```

It's about 10.13% of reduction! it is different than method 1 and method 2 but it raises a decreasing like method 2 so probably method 1 isn't goot at all.

I want to try differents values of beta, we can try beta = 0.001, beta = 0.1 and beta = 1

If i take a beta > 1 the computation time is very slow and it is about more than 60 iterations !

beta = 0.001 -> -10.13%
beta = 0.1 -> -10.81%
beta = 1 -> -8.62%
if we take a bigger beta, it will need more iterations to find an optimal. If you take a beta < 0.01, the result is the same and doesn't change.

7 METHOD 4 : ACTIVE CONTOUR

i've found an implementation of an active contour which uses the Chan Vese method, the library skimage has its own implementation. We can initialise the algorithm with a square which will be modified along the iterations to match the tumor. This method has two parameters to tune, lambda1 and lambda2. The documentation can be found here : http://scikit-image.org/docs/dev/api/skimage.segmentation.html#skimage.segmentation.morphological_chan_vese

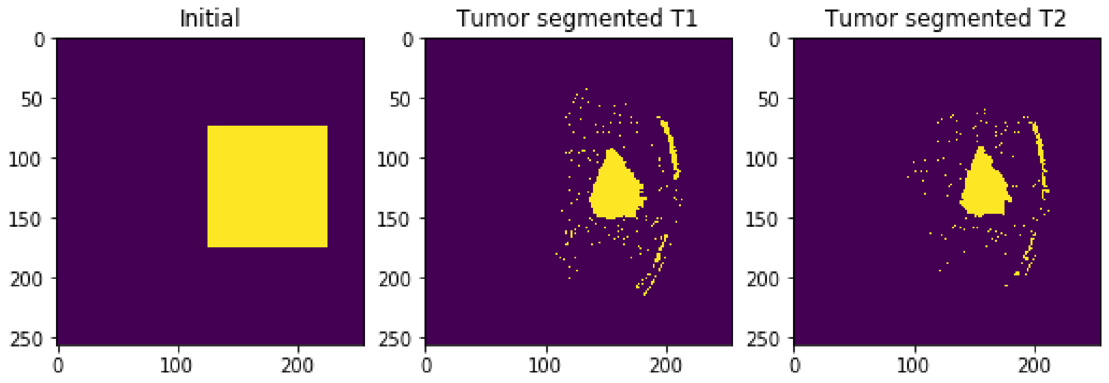
Example wich uses this method and function : http://scikit-image.org/docs/dev/auto_examples/segmentation/plot_morphsnakes.html#sphx-glr-auto-examples-segmentation-plot-morphsnakes-py

```
In [122]: # import
          from skimage.segmentation import checkerboard_level_set, morphological_chan_vese
          # Function to have to make the algorithm working
          def store_evolution_in(lst):
              """Returns a callback function to store the evolution of the level sets in
              the given list.
              """
              def _store(x):
                  lst.append(np.copy(x))
              return _store
```

```
In [138]: # PARAMETERS AND INITIALISATION
          threshold = 80
          white_t1 = img_t1 >= threshold
          t1_binary = np.zeros((256, 256))
          t1_binary[white_t1] = 1
          init_ls = np.zeros(img_t1.shape, dtype=np.int8)
          init_ls[75:175, 125:225] = 1

          evolution = []
          callback = store_evolution_in(evolution)
```

```
In [139]: # LET'S USE IT
          # T1
          segmentation_t1 = morphological_chan_vese(img_t1, 100, init_level_set=init_ls,
                                                   smoothing=0,lambda1=3, lambda2=1,
                                                   iter_callback=callback)


          # T2
          segmentation_t2 = morphological_chan_vese(img_t2, 100, init_level_set=init_ls,
                                                  smoothing=0,lambda1=5, lambda2=1)
```

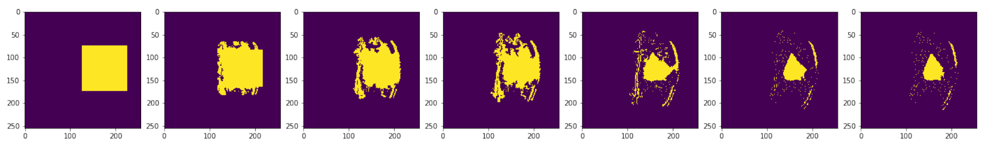
```
         plt.figure(figsize=(10,4))
         plt.subplot(131)
         plt.title("Initial")
         plt.imshow(init_ls)
         plt.subplot(132)
         plt.title('Tumor segmented T1')
         plt.imshow(seg1)
         plt.subplot(133)
         plt.title('Tumor segmented T2')
         plt.imshow(seg2)

Out[139]: <matplotlib.image.AxesImage at 0x1c3dcf5f60>
```



I have noticed 70 iterations, let's see the evolution every 10 iterations :

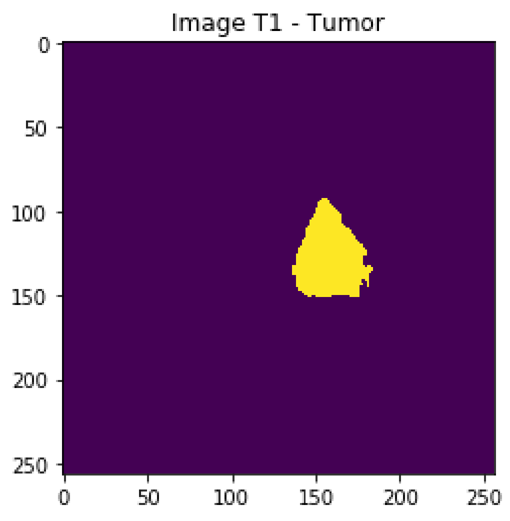
```
In [140]: plt.figure(figsize=(28, 10))
          plt.title("Evolution of t1")
          plt.subplot(1, 8, 1)
          plt.imshow(evolution[0])
          plt.subplot(1, 8, 2)
          plt.imshow(evolution[9])
          plt.subplot(1, 8, 3)
          plt.imshow(evolution[19])
          plt.subplot(1, 8, 4)
          plt.imshow(evolution[29])
          plt.subplot(1, 8, 5)
          plt.imshow(evolution[49])
          plt.subplot(1, 8, 6)
          plt.imshow(evolution[59])
          plt.subplot(1, 8, 7)
          plt.imshow(evolution[69])
          plt.subplot(1, 8, 8)
          plt.axis("off")
```

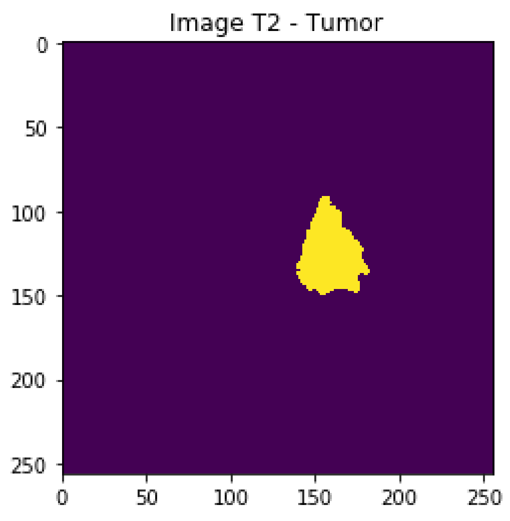
```
Out[140]: (0.0, 1.0, 0.0, 1.0)
```



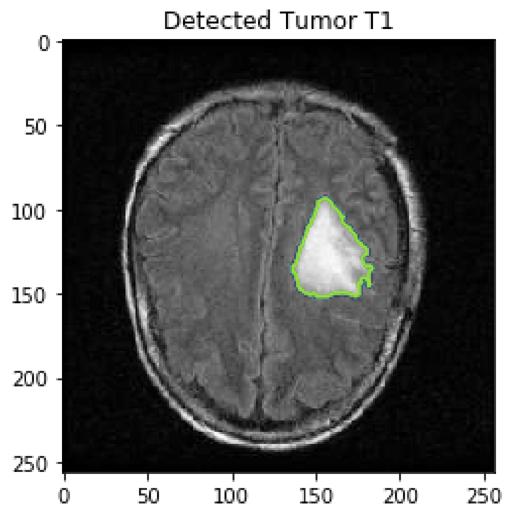
Now, let's do the traditional labelling and compare the 4 methods !

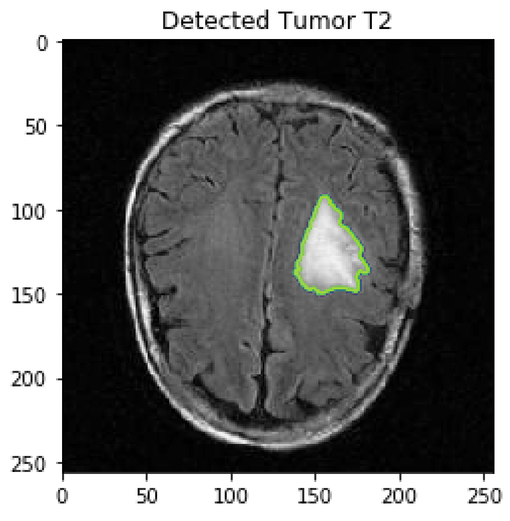
```
In [141]: label_t1, nb_feature_t1 = label(segmentation_t1)
          label_t2, nb_feature_t2 = label(segmentation_t2)

          # T1
          t1_tumor = count_labels_and_return_tumor(label_t1,nb_feature_t1)
          # T2
          t2_tumor = count_labels_and_return_tumor(label_t2,nb_feature_t2)

          # Display :
          plt.figure(figsize=(10, 4))
          plt.subplot(1,2,1)
          plt.title("Image T1 - Tumor")
          plt.imshow(t1_tumor)
          plt.subplot(1,2,2)
          plt.title("Image T2 - Tumor")
          plt.imshow(t2_tumor)
          plt.show()
```





```
In [142]: plot_contour_tumor(t1_tumor, t2_tumor, img_t1, img_t2)
```





```
In [143]: difference_activecontour = surface_evolution(t1_tumor, t2_tumor)
```

```
Percentage of change (t2 - t1) : -13.22%
```

8 CONCLUSION : COMPARISON BETWEEN METHODS

```
In [145]: print("EVOLUTION PERCENTAGE :")
          print("Binarization & Labelleing : "+str(difference_binarization)+"%")
```

```
        print("FCM : "+str(difference_fcm)+"%")
        print("MRF : "+str(difference_markov)+"%")
        print("Active Contour : "+str(difference_activecontour)+"%")
```

```
EVOLUTION PERCENTAGE :
Binarization & Labelleing : 1.1606597434331094%
FCM : -6.230910201588271%
MRF : -10.12919896640827%
Active Contour : -13.220338983050848%
```

The Active Contour gives to my mind the best results because the plot of the contour is more precise than the other methods. However it seems to be a less simple method than the 3 others because there are two hyperparameters to find, and find them is crucial to find a solution. Better solutions in general need more computation time (MRF) or are more difficult to optimize (Active Contour / Chan Vese).