# Super Pixel Para Segmentação da Rocha

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from matplotlib.colors import ListedColormap, LinearSegmentedColormap
from matplotlib import cm
from skimage.segmentation import slic
from skimage.segmentation import mark_boundaries
from skimage.measure import regionprops
from mylib import *
from copy import deepcopy
# %matplotlib notebook
```

In [2]:

```
fatia = np.load("secao_do_plug.npy")
show_npy(fatia)
```

```
shape=(925, 920), type=uint16
min= 0, max=10184
```

In [3]:

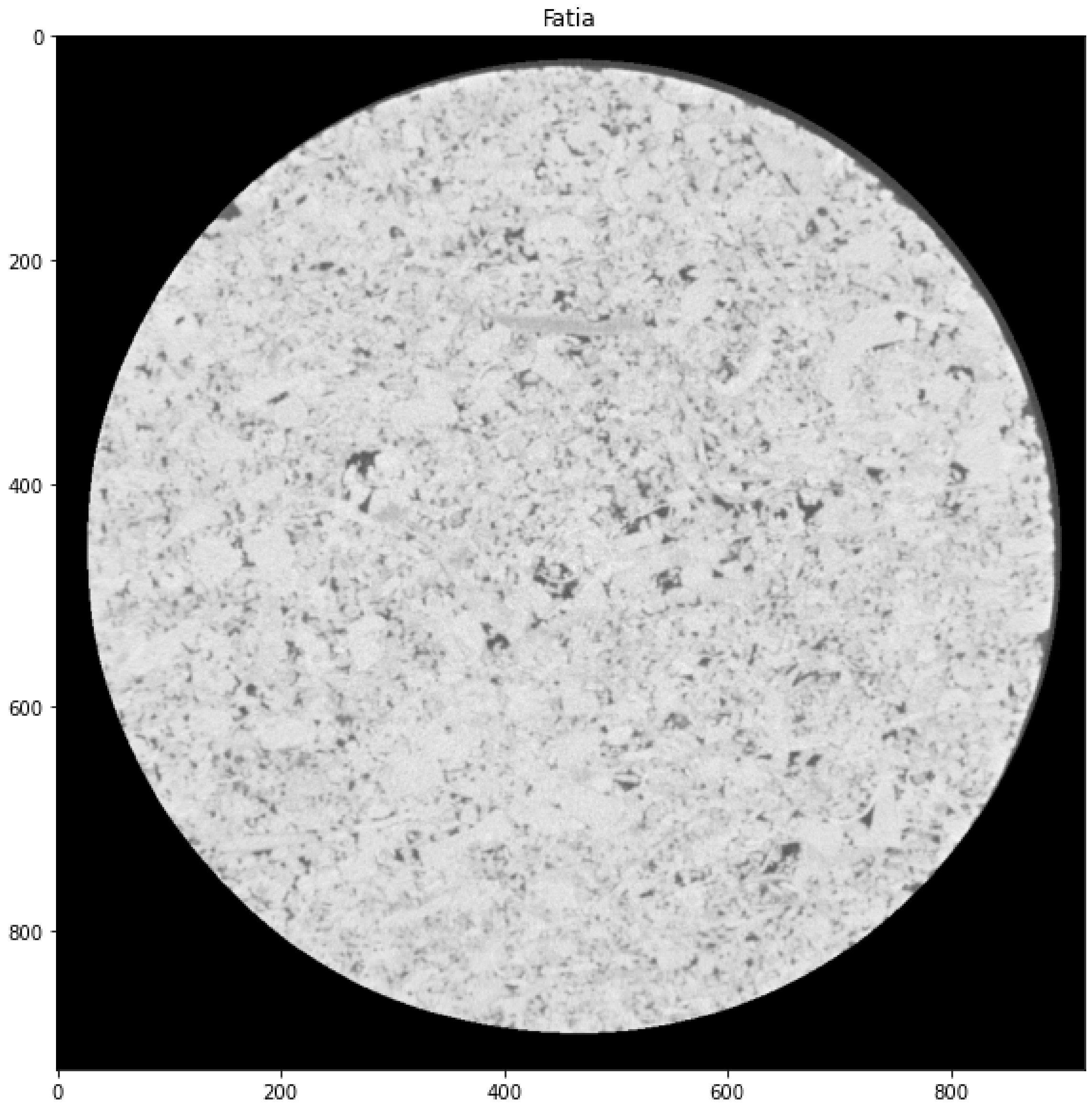
```
def normalise_image(img):
    max_val = np.max(img)
    img = img/max_val
    return img
```

In [4]:

```
fatia = normalise_image(fatia)
```

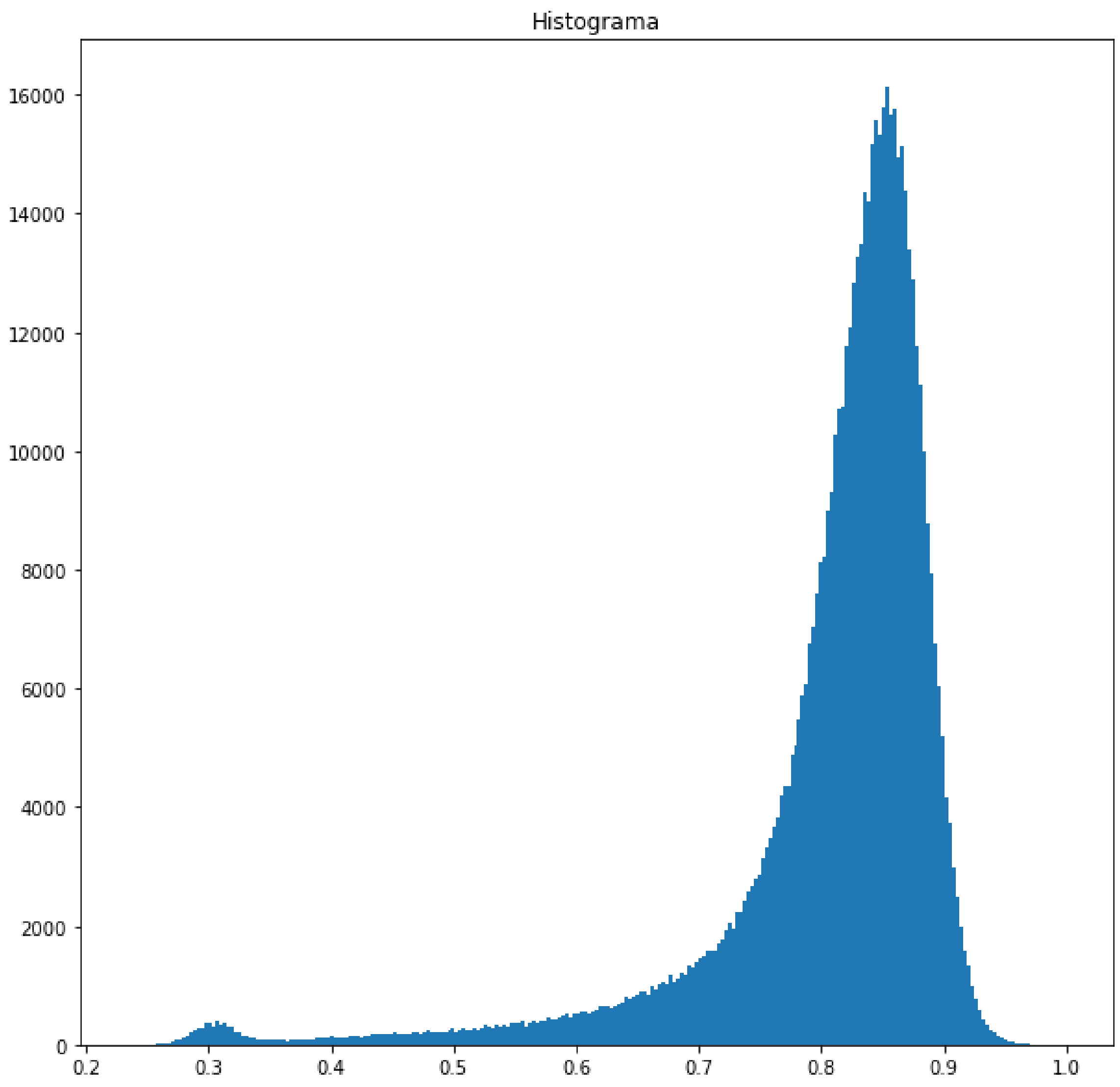
In [5]:

```
show_gray(fatia,"Fatia")
```



In [6]:

```
mask = fatia > 0
data = fatia[fatia>0]
show_hist(data.ravel(),"Histograma")
```



## Separação em Super Pixels

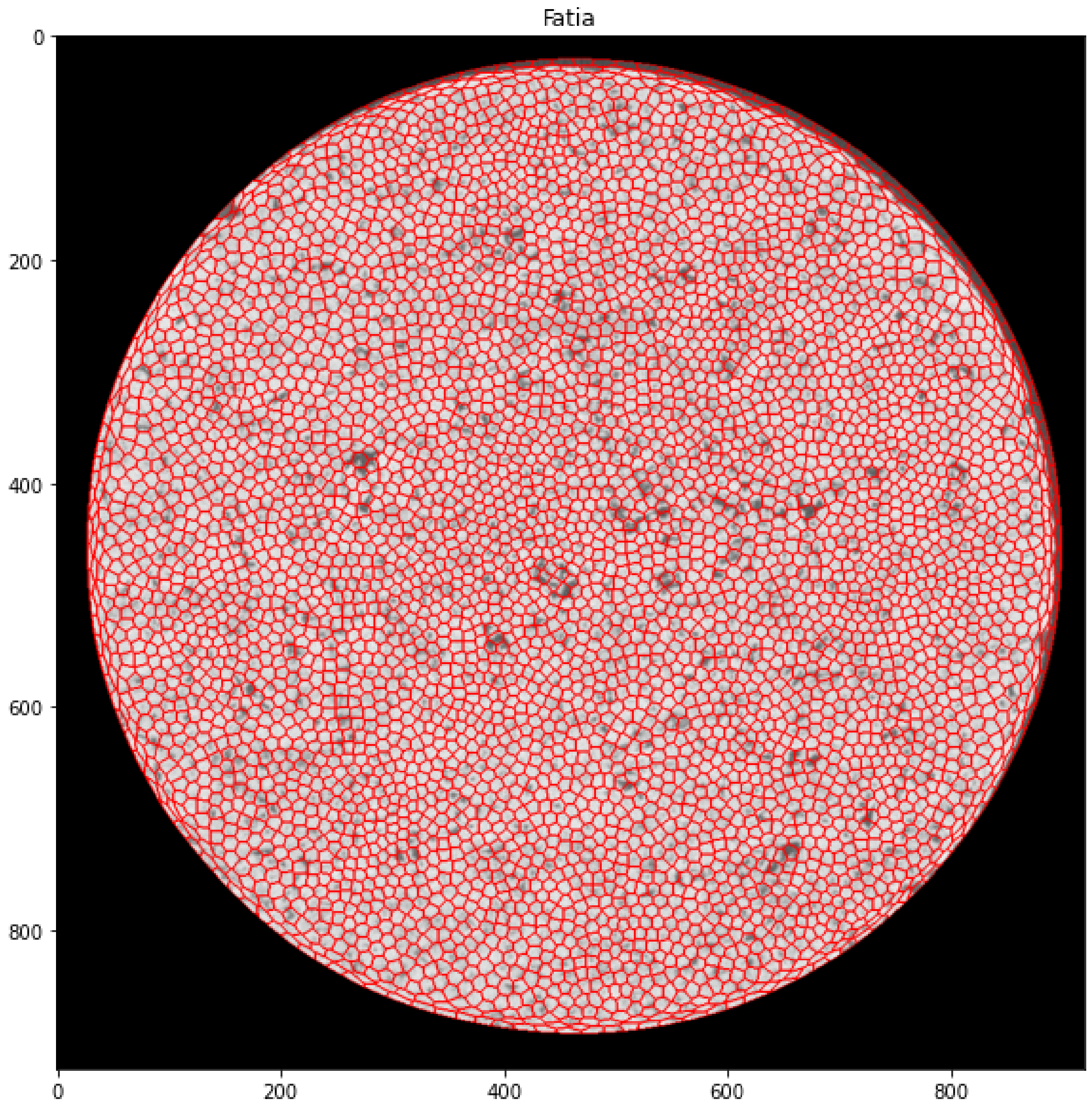
### Testes com Parâmetros

In [7]:

```
numSegments = 5000
```

In [8]:

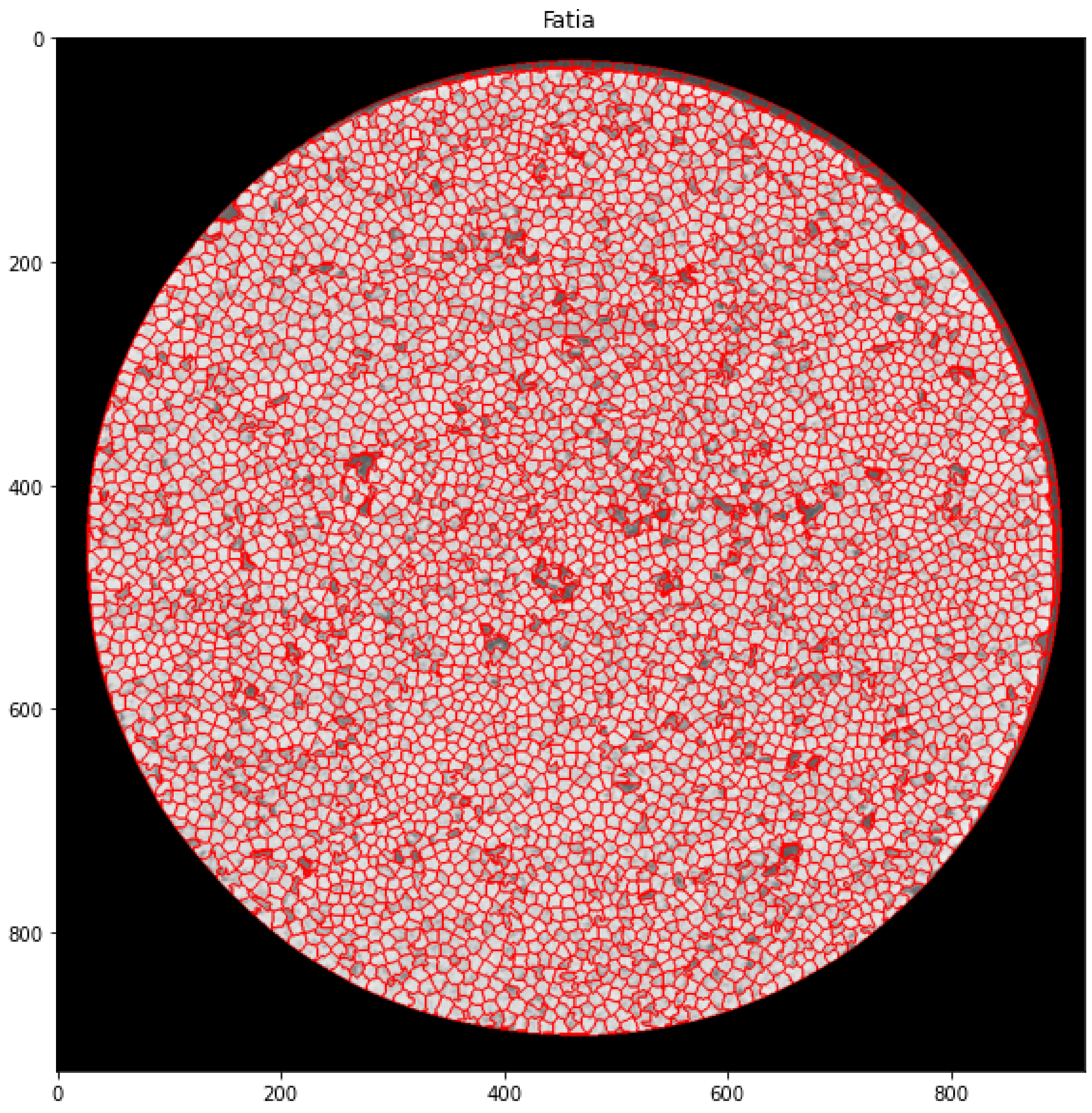
```
segments = slic(fatia, n_segments = numSegments, compactness = 0.06, start_label = 1, s
lic_zero = False, sigma = 10, mask = mask)
fatia_bound = mark_boundaries(fatia, segments, (1, 0, 0))
show_gray(fatia_bound,"Fatia")
```



In [9]:

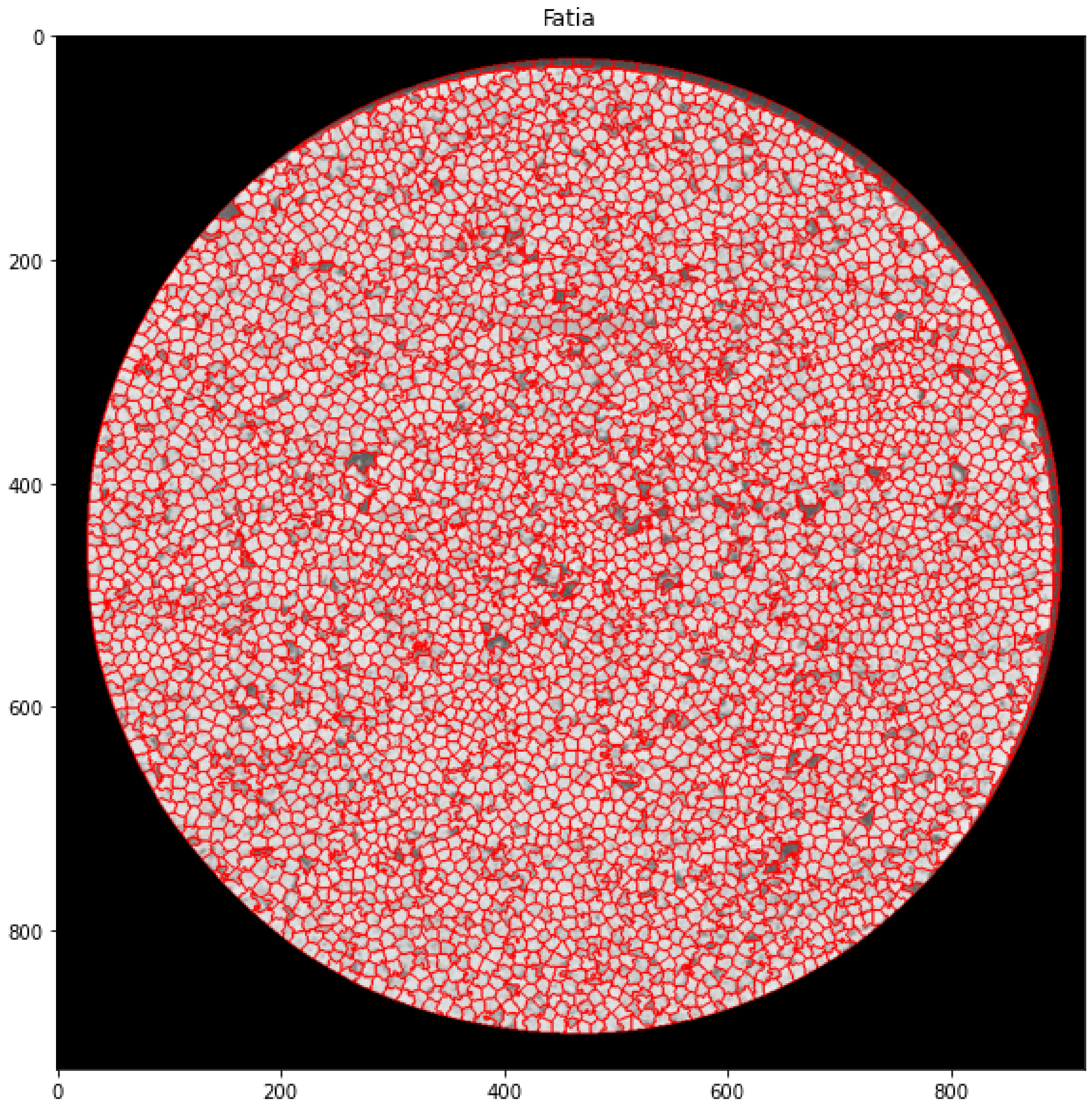
```
segments = slic(fatia, n_segments = numSegments, compactness = 0.06, start_label = 1, s
lic_zero = False, sigma = 1, mask = mask)
fatia_bound = mark_boundaries(fatia, segments, (1, 0, 0))
show_gray(fatia_bound,"Fatia")

best_segments = segments
best_fatia_bound = fatia_bound
```



In [10]:

```
segments = slic(fatia, n_segments = numSegments, compactness = 0.06, start_label = 1, s
lic_zero = False, sigma = 0.1, mask = mask)
fatia_bound = mark_boundaries(fatia, segments, (1, 0, 0))
show_gray(fatia_bound,"Fatia")
```



## Cálculo dos Centroides e Valor Médio

In [11]:

```
def fill_region(img, region, value):
    for i in range(region.shape[0]):
        img[region[i][0]][region[i][1]] = value

    return img
```

In [12]:

```python
def mse(matrix, value):
    return (np.square(matrix - value)).mean(axis=None)
```

In [13]:

```python
regions = regionprops(best_segments, intensity_image=fatia)

cx = []
cy = []
means_image = deepcopy(fatia)
mse_image = deepcopy(fatia)

for props in regions:
    y, x = props.centroid
    values = props.intensity_image
    cx.append(x)
    cy.append(y)
    mean = props.mean_intensity
    means_image = fill_region(means_image, props.coords, mean)
    mse_image =  fill_region(mse_image, props.coords, mse(values, mean))
```

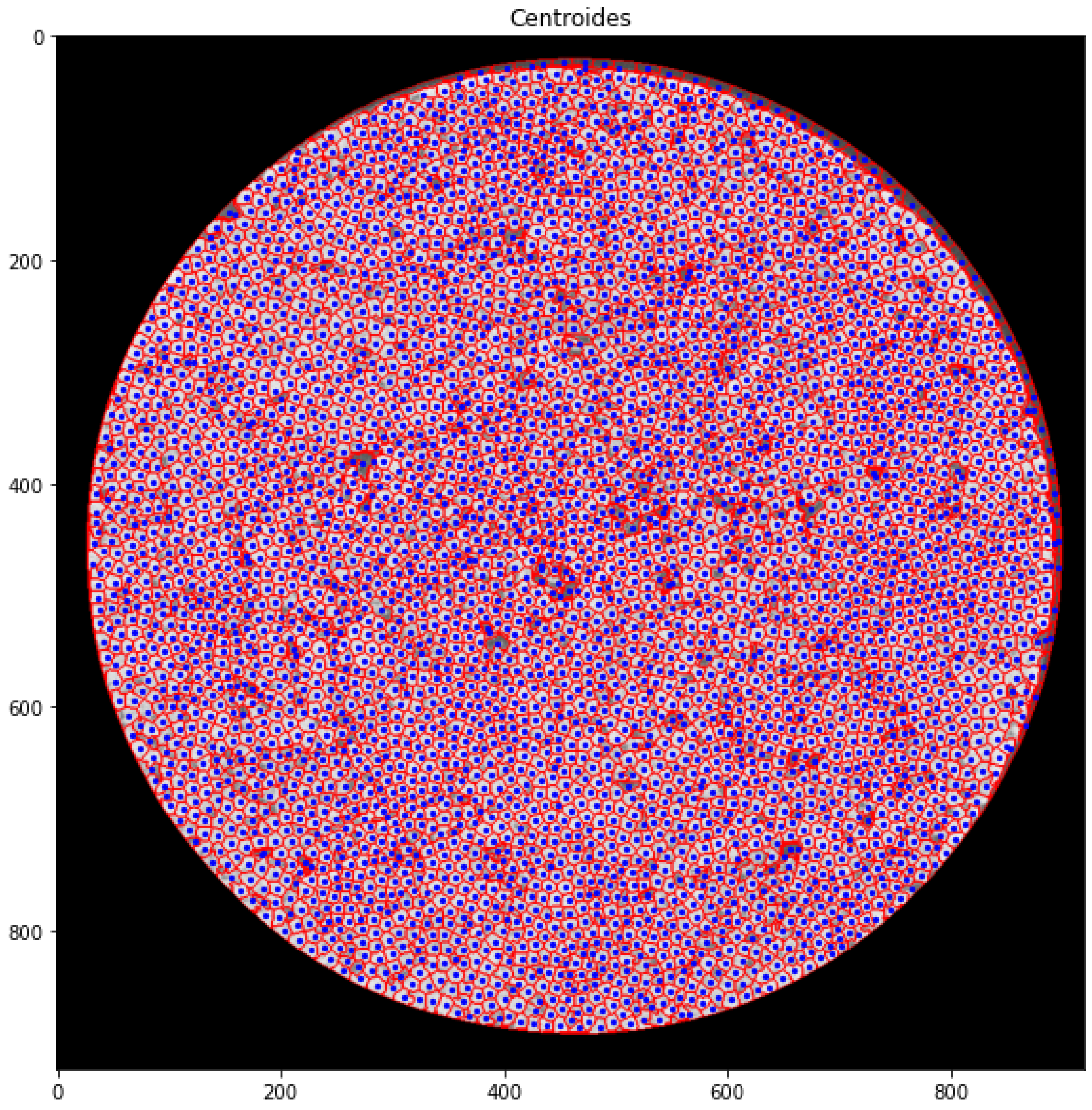
In [14]:

```python
def show_centroids(img, cx, cy, title):
    plt.figure(figsize=(10, 10))
    plt.title(title)
    plt.imshow(img, cmap="gray")
    plt.scatter(x=cx, y=cy, c='b', s=5)
    plt.show()
```

## Imagem com Centroides Sobrepostos

In [15]:

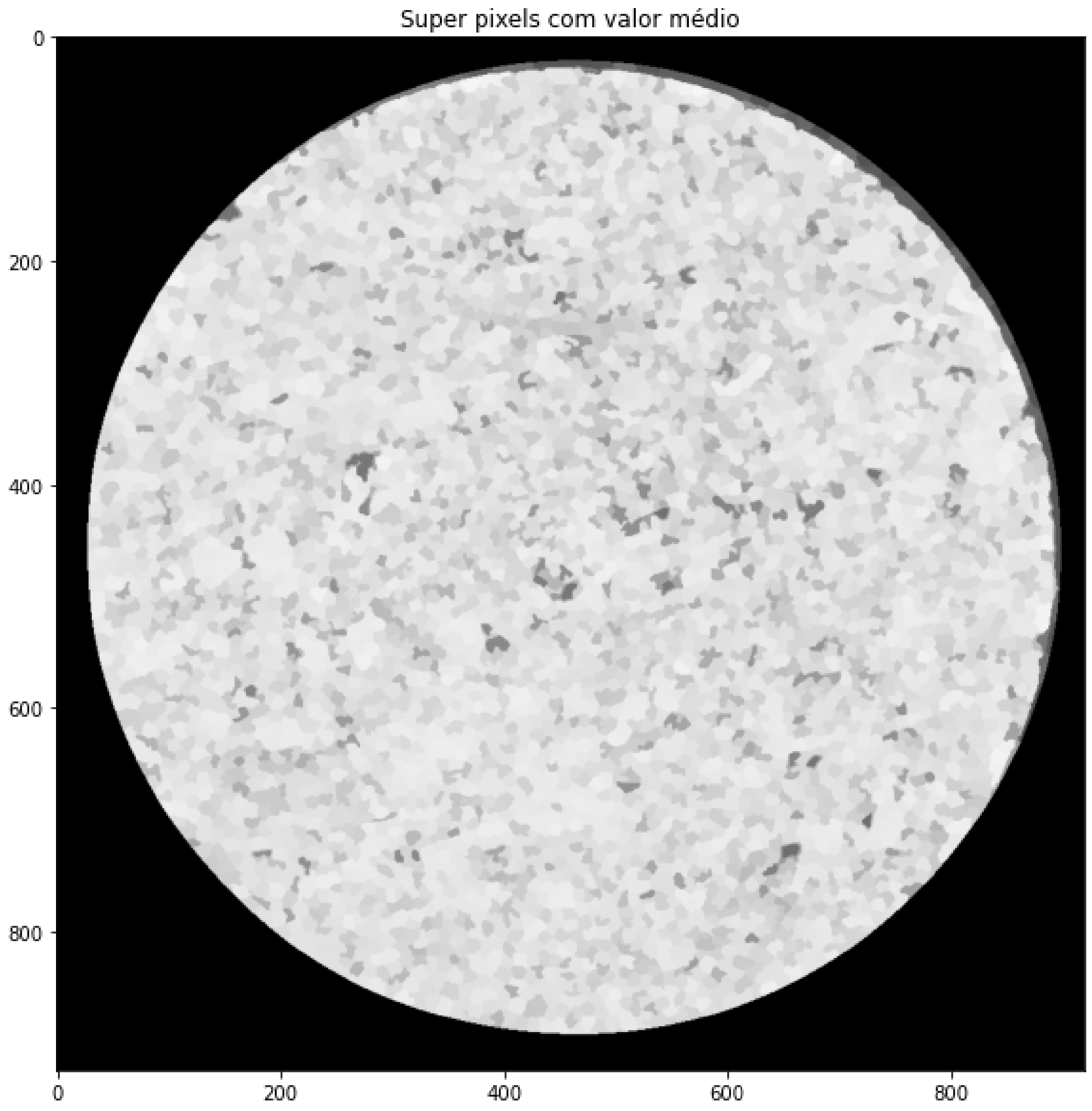
```
show_centroids(best_fatia_bound, cx, cy, "Centroides")
```



## Imagem com Valores Médios dos Super Pixels

In [16]:

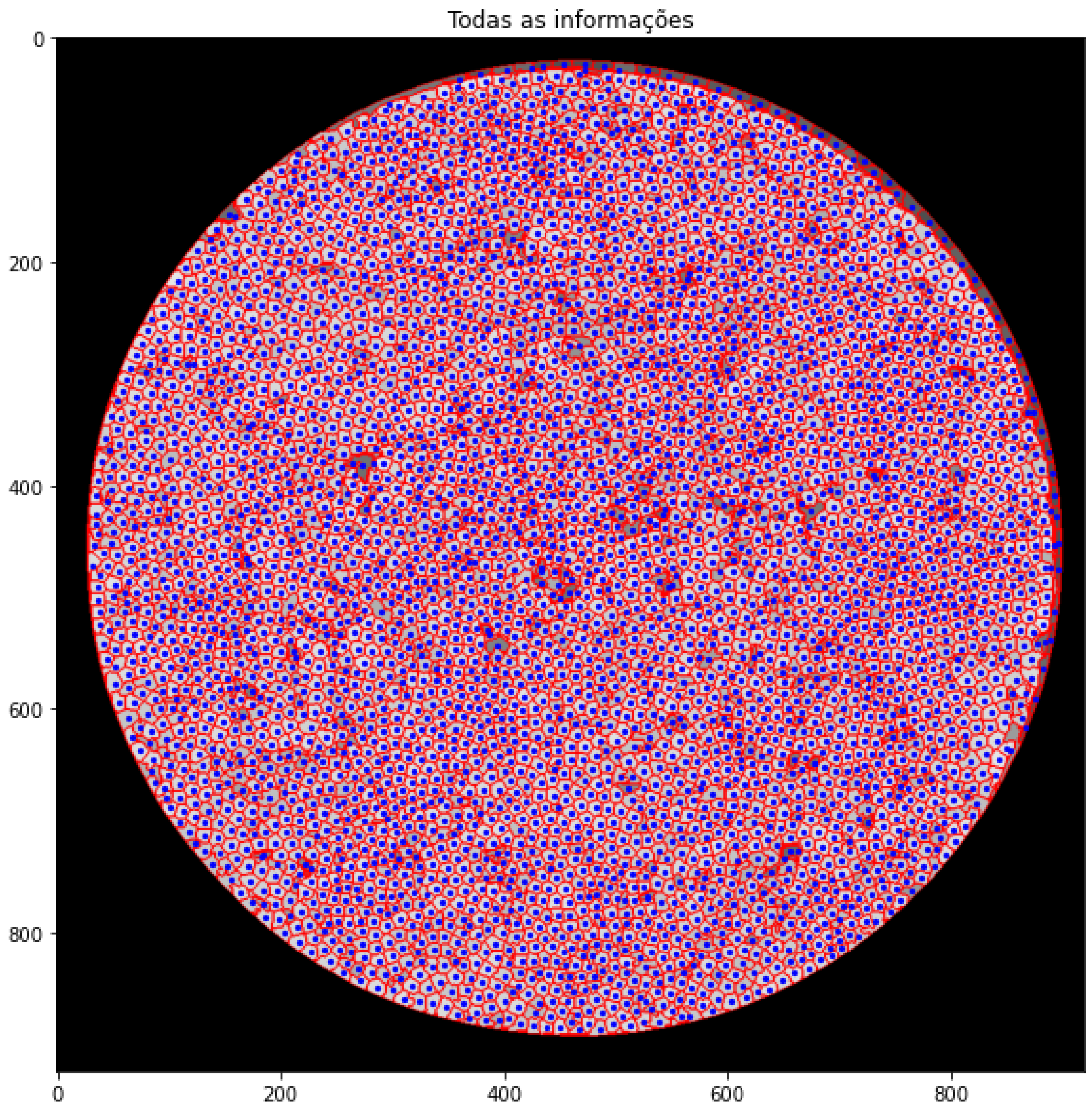
```
show_gray(means_image,"Super pixels com valor médio")
```



## Imagem com Todas as Informações

In [17]:

```
show_centroids(mark_boundaries(means_image, best_segments, (1, 0, 0)), cx, cy,"Todas as informações")
```



## Cálculo do Erro Médio

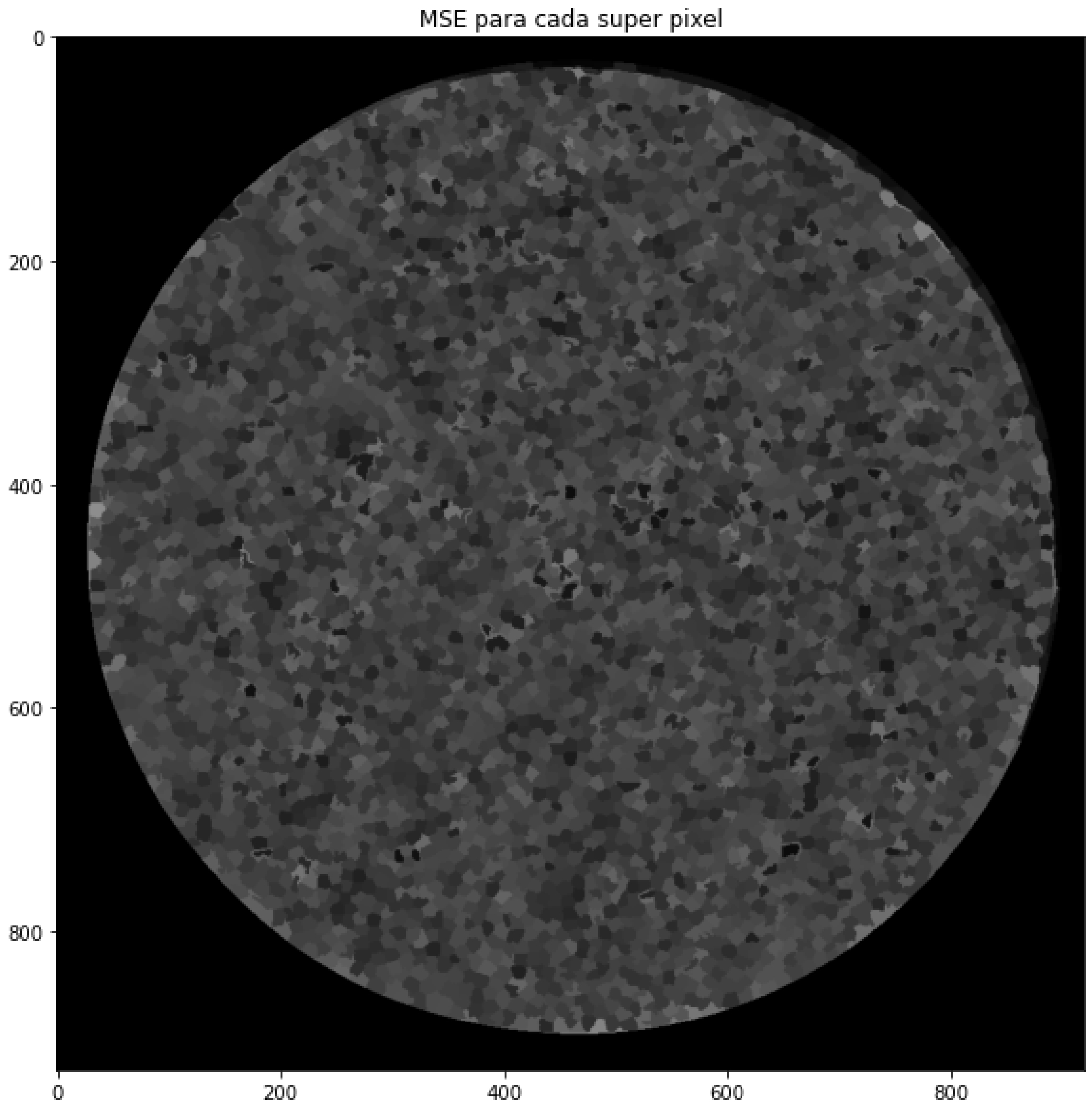
In [18]:

```
mse = (np.square(means_image - fatia)).mean(axis=None)
print(f'MSE = {mse}')
```

```
MSE = 0.0017751555923630708
```

In [19]:

```
show_gray(mse_image,"MSE para cada super pixel")
```


MSE para cada super pixel