```
from google.colab import drive
drive.mount('/content/drive')
import numpy as np
import scipy.io
import pandas as pd
from skimage import color
from skimage import io
from skimage.transform import radon, iradon, iradon_sart, rescale
from skimage.metrics import structural_similarity
from skimage.metrics import peak_signal_noise_ratio
import math
import matplotlib.pyplot as plt
```

```
Mounted at /content/drive
```

```
ctScans = scipy.io.loadmat('/content/drive/My Drive/CCE-AIMIA/ctscan_hw1.mat')
ctMasks = scipy.io.loadmat('/content/drive/My Drive/CCE-AIMIA/infmsk_hw1.mat')
```

```
(ms,ns,cs)= (ctScans['ctscan']).shape
(mm,nm,cm)= (ctMasks['infmsk']).shape
print((ms,ns,cs))
print((mm,nm,cm))
```

```
(512, 512, 3554)
(512, 512, 3554)
```

```
ctscansarray = []
ctmasksarray = []
for i in range(cm):
    ctscansarray.append((ctScans['ctscan'][:,:,i]))
    ctmasksarray.append((ctMasks['infmsk'][:,:,i]))
```

```
image = ctscansarray[3514]
image.shape
```

```
(512, 512)
```

```
class KmeansSegmentation:

    def segmentation_grey(self, image, k=2):
        """Performs segmentation of an grey level input image using KMeans algorithm, usin
        The function is the modified version adopted from the one produced by the below gi
        https://github.com/DSGeek24/Image-segmentation_KMeans/
        """
        #assigning cluster centroids clusters
        centroids = []
        clusters=[]
```

```
11
12          i=1
13          # Initializes k number of centroids for the clustering making sure no cluster cent
14
15          while(len(centroids)!=k):
16              cent = image[np.random.randint(0, image.shape[0]), np.random.randint(0, image.
17              if(len(centroids)>=1):
18                  if(cent not in centroids):
19                      centroids.append(cent)
20              else:
21                  centroids.append(cent)
22          print("Initial centroids are {}".format(centroids))
23
24          # Initializing k clusters
25          for m in range(0, k):
26              cluster=[]
27              clusters.append(cluster)
28
29          # Calling k means which returns the clusters with pixels
30          clusters = self.kmeans(clusters, image, centroids, k)
31          new_centroids=self.calculate_new_centroids(clusters,k)
32
33          # clustering and finding new centroids till convergence is reached
34          while(not(np.array_equal(new_centroids,centroids))) and i<=15:
35              centroids=new_centroids
36              clusters=self.kmeans(clusters,image,centroids,k)
37              new_centroids = self.calculate_new_centroids(clusters, k)
38              i=i+1
39          print("Convergence reached")
40
41          image=self.assignPixels(clusters,image,k)
42          return image
43
44      def findMinIndex(self,pixel, centroids):
45          d = []
46          for i in range(0, len(centroids)):
47              d1 = abs(int(pixel) - centroids[i])
48              d.append(d1)
49          minIndex = d.index(min(d))
50          return minIndex
51
52      def assignPixels(self,clusters,image,k):
53          cluster_centroids=[]
54          for i in range(0, k):
55              cent = np.nanmean(clusters[i])
56              cluster_centroids.append(cent)
57
58          for x in range(image.shape[0]):
59              for y in range(image.shape[1]):
60                  Value = round(cluster_centroids[self.findMinIndex(image[x,y], cluster_cent
61                  image[x, y] = Value
```

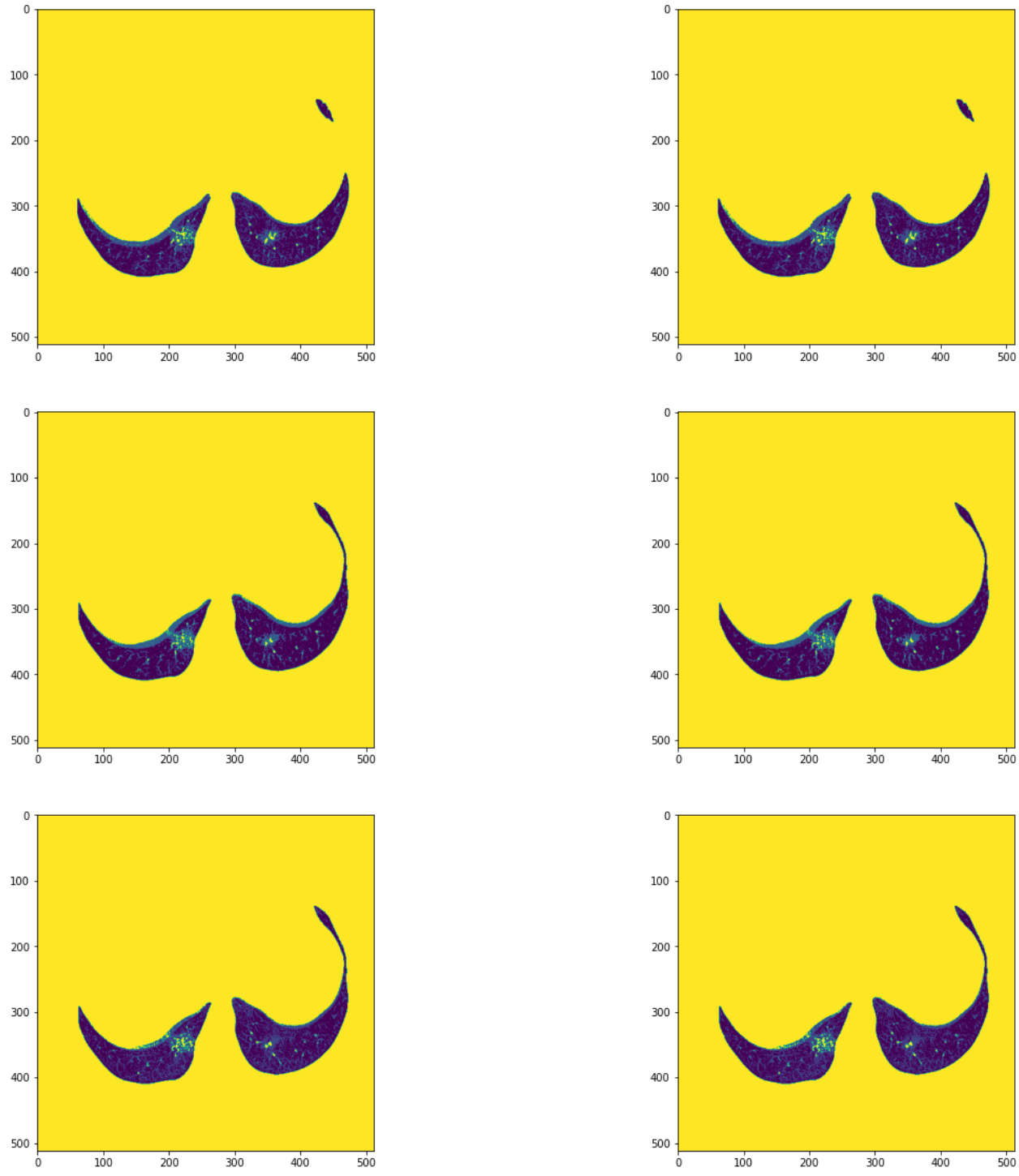
```
        return image

    def kmeans(self, clusters, image, centroids, k):

        def add_cluster(minIndex, pixel):
            try:
                clusters[minIndex].append(pixel)
            except KeyError:
                clusters[minIndex] = [pixel]
        for x in range(0, image.shape[0]):
            for y in range(0, image.shape[1]):
                pixel = image[x, y].tolist()
                minIndex = self.findMinIndex(pixel, centroids)
                add_cluster(minIndex, pixel)
        return clusters

    def calculate_new_centroids(self,clusters,k):
        new_centroids=[]
        for i in range(0, k):
            cent = np.nanmean(clusters[i])
            new_centroids.append(round(cent))
        return new_centroids
```

```
Segementation_object = KmeansSegmentation()
KmeanSegData = []
for i in range(len(ctscansarray)):
  KmeanSegData.append(Segementation_object.segmentation_grey(ctscansarray[i], 3))
```

```
Initial centroids are [224, 113, 243]
Convergence reached
Initial centroids are [255, 211, 227]
Convergence reached
Initial centroids are [253, 145, 29]
Convergence reached
Initial centroids are [255, 120, 64]
Convergence reached
Initial centroids are [255, 67, 108]
Convergence reached
Initial centroids are [230, 46, 84]
Convergence reached
Initial centroids are [226, 22, 63]
Convergence reached
Initial centroids are [253, 36, 48]
Convergence reached
Initial centroids are [255, 63, 54]
Convergence reached
Initial centroids are [255, 89, 86]
Convergence reached
Initial centroids are [245, 47, 85]
Convergence reached
Initial centroids are [235, 17, 91]
Convergence reached
Initial centroids are [255, 49, 101]
Convergence reached
Initial centroids are [236, 38, 55]
Convergence reached
Initial centroids are [241, 60, 64]
Convergence reached
Initial centroids are [230, 45, 34]
Convergence reached
Initial centroids are [232, 27, 12]
```

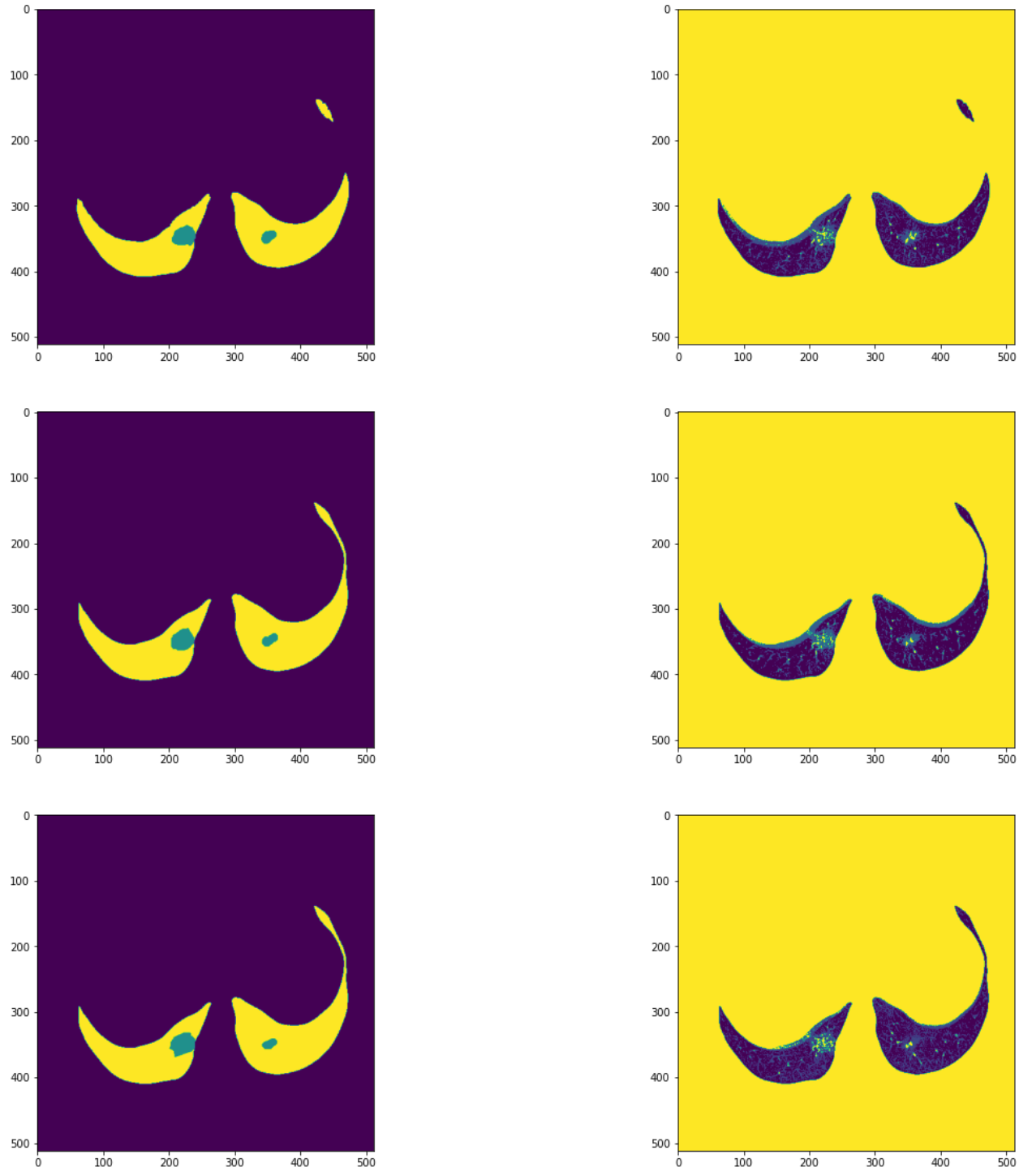
```
fig, ((ax1), (ax2), (ax3)) = plt.subplots(3, 2,figsize=(20, 20))
ax1[0].imshow((ctscansarray[900]))#,cmap="gray")
ax1[1].imshow((KmeanSegData[900]))#,cmap="gray")
ax2[0].imshow((ctscansarray[901]))#,cmap="gray")
ax2[1].imshow((KmeanSegData[901]))#,cmap="gray")
ax3[0].imshow((ctscansarray[902]))#,cmap="gray")
ax3[1].imshow((KmeanSegData[902]))#,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f79e44577d0>
```



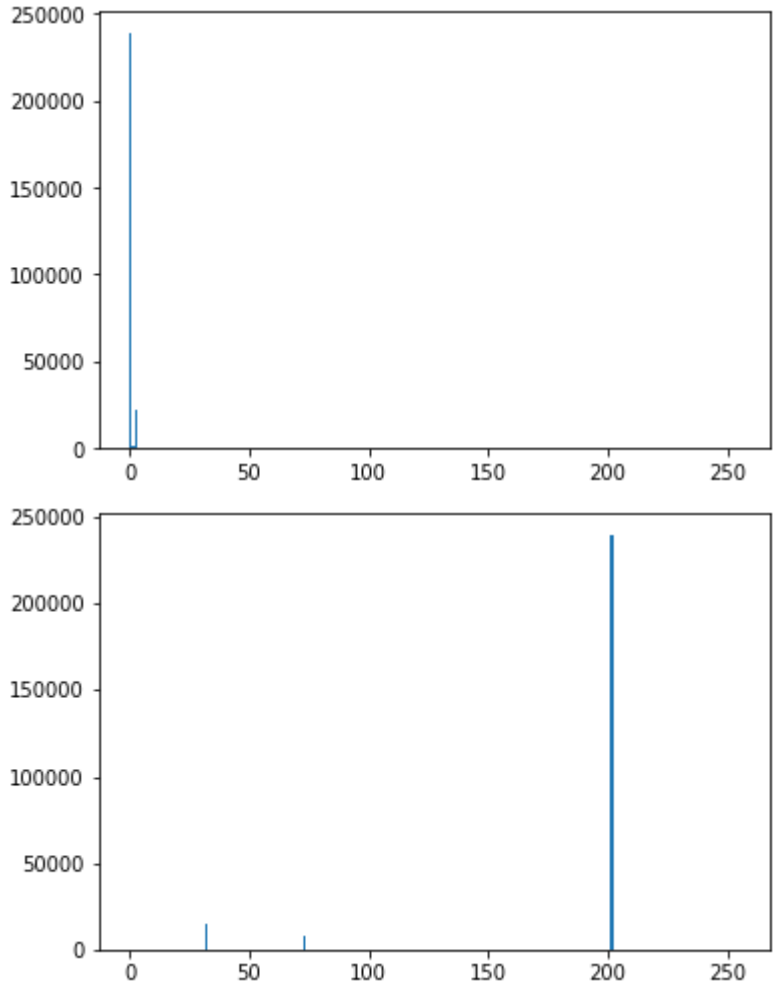
```
fig, ((ax1), (ax2), (ax3)) = plt.subplots(3, 2,figsize=(20, 20))
ax1[0].imshow((ctmasksarray[900]))#,cmap="gray")
ax1[1].imshow((KmeanSegData[900]))#,cmap="gray")
ax2[0].imshow((ctmasksarray[901]))#,cmap="gray")
ax2[1].imshow((KmeanSegData[901]))#,cmap="gray")
ax3[0].imshow((ctmasksarray[902]))#,cmap="gray")
ax3[1].imshow((KmeanSegData[902]))#,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f79e3ed9cd0>
```



```
1 #Historgram Test
```

```
img1 = ctmasksarray[900]
img2 = KmeanSegData[900]
#plt.hist(img.ravel(), bins=256, range=(0.0, 1.0), fc='k', ec='k') #calculating histogram
plt.hist(img1.ravel(),256,[0,255])
plt.show()
plt.hist(img2.ravel(),256,[0,255])
plt.show()
```




```
ctmaskinfected = []
Kmeansinfected = []
ctmaskhealthy = []
Kmeanshealthy = []
for i in range(len(KmeanSegData)):
  ctmaskinfected.append(np.sum((ctmasksarray[i])==1))
  Kmeansinfected.append(np.sum((KmeanSegData[i])==1))
  ctmaskhealthy.append(np.sum((ctmasksarray[i])==2))
  Kmeanshealthy.append(np.sum((KmeanSegData[i])==73))
```

```
tp = 0
tn = 0
fn = 0
fp = 0
```

```
 5
 6 for i in range(len(KmeanSegData)):
 7     if((ctmaskinfected[i] == Kmeansinfected[i])):
 8             tp = tp + 1
 9     if((ctmaskhealthy[i] == Kmeanshealthy[i])):
10             tn = tn + 1
11     if((ctmaskhealthy[i] == Kmeansinfected[i])):
12             fn = fn + 1
13     if((ctmaskinfected[i] == Kmeanshealthy[i])):
14             fp = fp + 1
15
16 print ('\n************Calculation of Tpr, Fpr, F-Score')
17 print(tp)
18 print(tn)
19 print(fn)
20 print(fp)
21 #TP rate = TP/TP+FN
22 tpr= float(tp)/(tp+fn)
23 print("\nTPR is:",tpr)
24
25 #fp rate is
26 fpr= float(fp)/(fp+tn)
27 print("\nFPR is:",fpr)
28
29 #F-score as 2TP/(2TP + FP + FN)
30 fscore = float(2*tp)/((2*tp)+fp+fn)
31 print("\nFscore:",fscore)
```

```
************Calculation of Tpr, Fpr, F-Score
423
0
0
423

TPR is: 1.0

FPR is: 1.0

Fscore: 0.6666666666666666
```