ICT for Health
Laboratory # 3
CT scan - Covid interstitial pneumonia

Monica Visintin

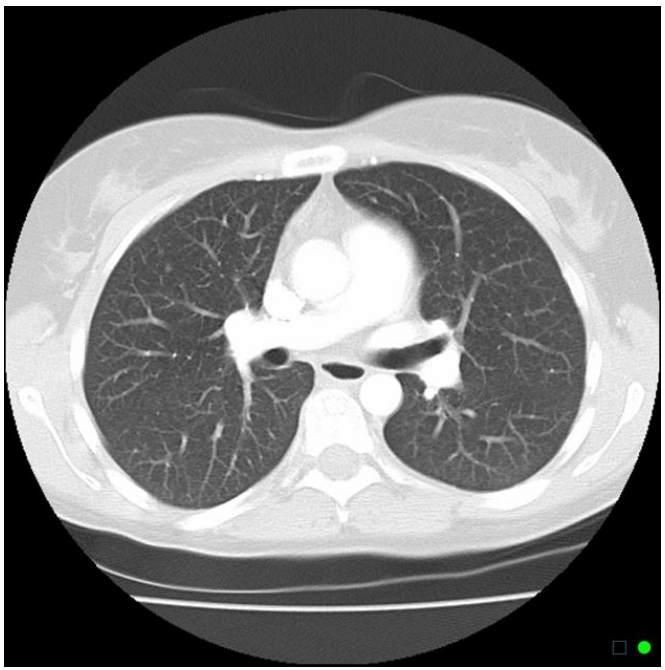
Politecnico di Torino

November 19th 2020

Table of Contents

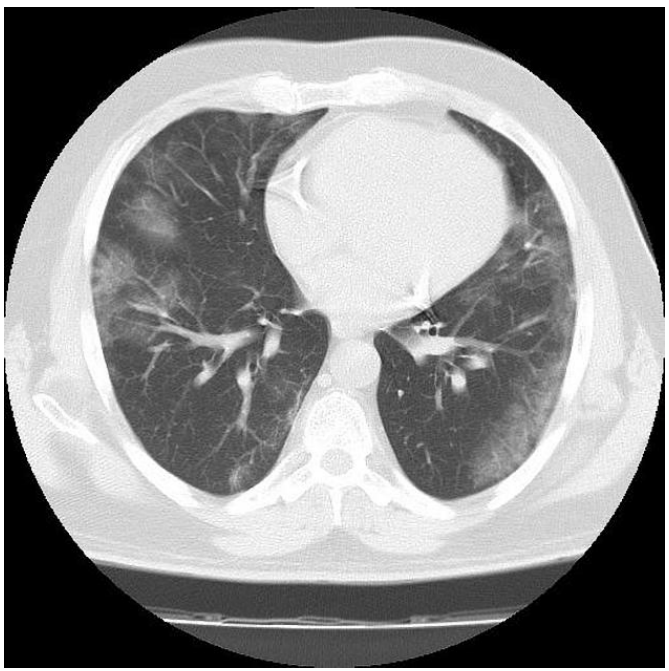
Description [1]

Covid-19 in some cases causes the so-called interstitial pneumonia, which can be distinguished from radiographies or CT-scans of lungs.
Normal lung CT-scans can be seen at link
`https://radiopaedia.org/cases/normal-chest-ct-lung-window-1?lang=us`
One image is



Description [2]

Lungs with Covid-19 interstitial pneumonia can be seen in the CT-scan at link:
`https://radiopaedia.org/cases/covid-19-pneumonia-4`
One image is the following. Note the presence of ground glass opacities (”frosted” areas) with a lighter grey color inside the lungs.



Description [3]

Aim of the lab is to isolate the ground glass opacities inside the lungs.
The analysis is performed on the specific CT-scan in folder `lab3`, subfolder `ct_scans` in Drobox. The CT-scan is saved in **NII format**, which can be read using a specific Python library. Use: `import nibabel as nib`. Of course, you first need to install package nibabel.
There are several ways to solve the problem. We will use clustering algorithms (**K-means** and **DBSCAN**) and **image filtering**.
K-means and DBSCAN are available in **scikit learn library of Python** and they must be imported.

Table of Contents

Read the data [1]

```
1  import numpy    as np
2  import nibabel as nib # to read NII files
3  import matplotlib.pyplot as plt
4  from sklearn.cluster import KMeans
5  from sklearn.cluster import DBSCAN
6
7  #%% methods
8  def read_nii(filepath):
9      '''
10     Reads .nii file and returns pixel array
11     '''
12     ct_scan = nib.load(filepath)
13     array   = ct_scan.get_fdata()
14     array   = np.rot90(np.array(array))
15     return(array)
```

Read the data [2]

```
17 def plotSample(array_list, color_map = 'nipy_spectral'):
18     '''
19     Plots a slice with all available annotations
20     '''
21     plt.figure(figsize=(18,15))
22     plt.subplot(1,4,1)
23     plt.imshow(array_list[0], cmap='bone')
24     plt.title('Original Image')
25     plt.subplot(1,4,2)
26     plt.imshow(array_list[0], cmap='bone')
27     plt.imshow(array_list[1], alpha=0.5, cmap=color_map)
28     plt.title('Lung Mask')
29     plt.subplot(1,4,3)
30     plt.imshow(array_list[0], cmap='bone')
31     plt.imshow(array_list[2], alpha=0.5, cmap=color_map)
32     plt.title('Infection Mask')
33     plt.subplot(1,4,4)
34     plt.imshow(array_list[0], cmap='bone')
35     plt.imshow(array_list[3], alpha=0.5, cmap=color_map)
36     plt.title('Lung and Infection Mask')
37     plt.show()
```

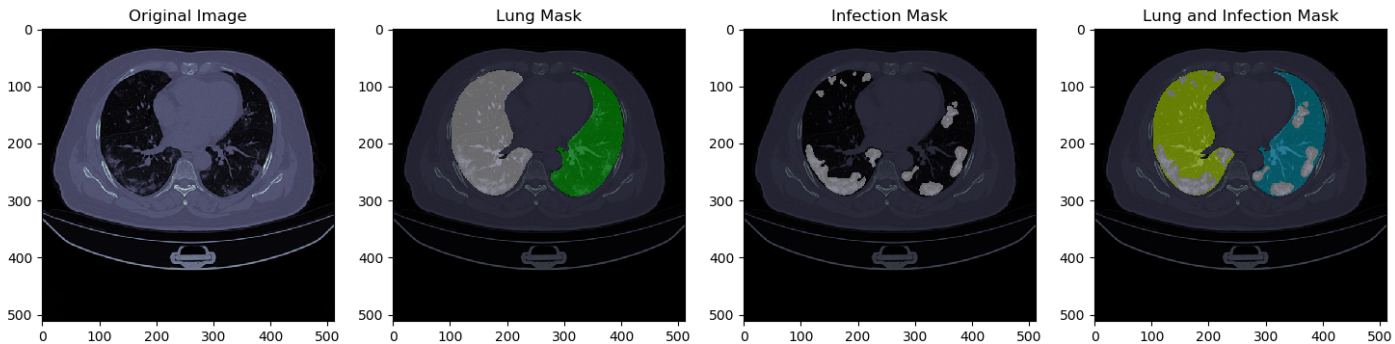
Read the data [3]

Main part: read and show a slice

```
39 #%% main part
40 # Read sample
41 plt.close('all')
42 fold1='./ct_scans'
43 fold2='./lung_mask'
44 fold3='./infection_mask'
45 fold4='./lung_and_infection_mask'
46 f1='/coronacases_org_001.nii'
47 f2='/coronacases_001.nii'
48 sample_ct   = read_nii(fold1+f1+f1)
49 sample_lung = read_nii(fold2+f2+f2)
50 sample_infe = read_nii(fold3+f2+f2)
51 sample_all  = read_nii(fold4+f2+f2)
52
53 Nr,Nc,Nimages=sample_ct.shape# Nr=512,Nc=512,Nimages=301
54 # Examine one slice of a ct scan and its annotations
55 index=140 # index of the slice
56 sct=sample_ct[...,index]
57 sl=sample_lung[...,index]
58 si=sample_infe[...,index]
59 sa=sample_all[...,index]
60 plotSample([sct,sl,si,sa])
```

Read the data [4]

This is what you see:



The first image is the original one (`sct`), the others were obtained using software which is not available. We should get similar results using K-means and DBSCAN. In the third image, the ground glass opacities (interstitial pneumonia) are light grey.

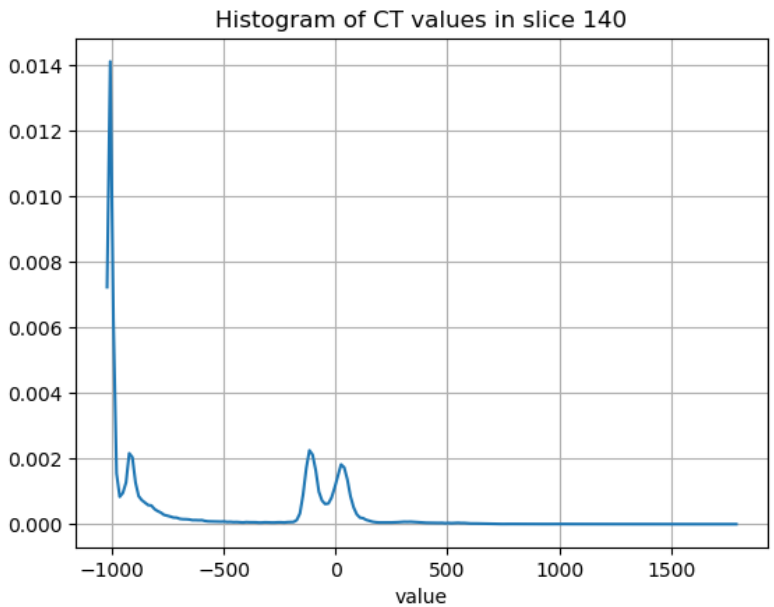
Table of Contents

K-Means [1]

Notice that in the original image you have,essentially, 4 different grey colors (see the histogram, next slide) and infection appears as a dark grey (not the darkest). Values of `sct` range from -1021 to 1806; smaller values correspond to darker pixels. Results are shown for slice 140.

K-Means [2]

```
62 a=np.histogram(sct,200,density=True)
63 plt.figure()
64 plt.plot(a[1][0:200],a[0])
65 plt.title('Histogram of CT values in slice '+str(index))
66 plt.grid()
67 plt.xlabel('value')
```



K-Means [3]

We can then ask K-means to quantize the image colors into 4 different clusters, but in some slices it is convenient to have 5 quantized colors to identify lungs in the image. It is first necessary to instantiate object KMeans:

```
62 Ncluster=5
63 kmeans = KMeans(n_clusters=Ncluster,random_state=0)# instantiate Kmeans
```

To run the algorithm, you must write `kmeans.fit(A)`
where `A` must be a 2D array. You cannot directly pass the image (512×512 pixels), because KMeans would try and find 5 groups of rows of pixels (one entire row would be mapped into an "average" row of the cluster it belongs to). This is not what we want. We want 5 groups of colors and each pixel should be mapped into a color which is the average color of the cluster it belongs to.

K-Means [4]

Therefore, what we need to do is to reshape the image to get just one column and then use `kmeans.fit(A)`

```
64 A=sct.reshape(-1,1)# Ndarray, Nr*Nc rows, 1 column
65 kmeans.fit(A)# run Kmeans on A
```

The interesting attributes of `kmeans` are

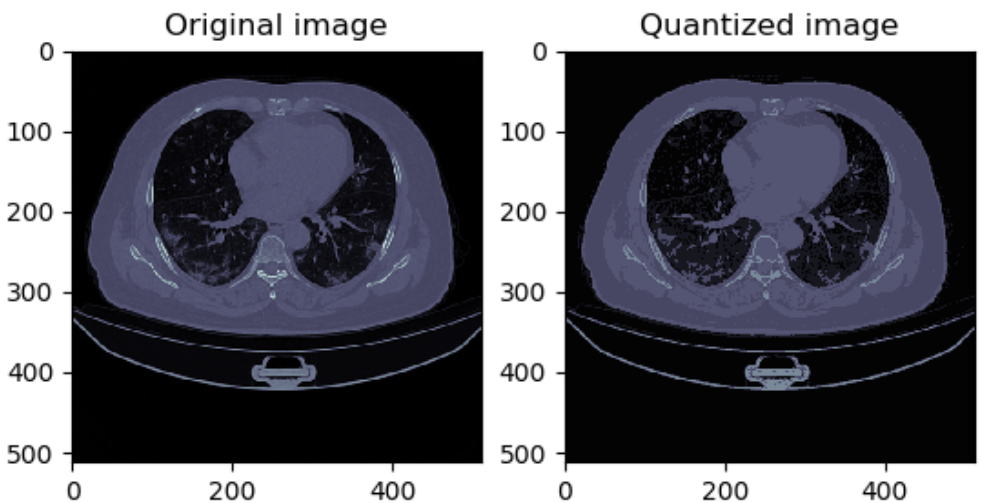
- `kmeans.cluster_centers_` that stores the five quantized colors (size equal to 5)
- `kmeans.labels_` that stores the cluster each pixel belongs to (size equal to `Nr*Nc`)

To generate the color-quantized new image, write:

```
66 kmeans_centroids=kmeans.cluster_centers_.flatten()# centroids/quantized colors
67 for k in range(Ncluster):
68     ind=(kmeans.labels_==k)# indexes for which the label is equal to k
69     A[ind]=kmeans_centroids[k]# set the quantized color
70 sctq=A.reshape(Nr,Nc)# quantized image
```

K-Means [5]

```
71 plt.figure()
72 plt.imshow(sct, cmap='bone')
73 plt.title('Original image')
74 vm=sct.min()
75 vM=sct.max()
76 plt.figure()
77 plt.imshow(sctq,vmin=vm,vmax=vM, cmap='bone')
78 plt.title('Quantized image')
```

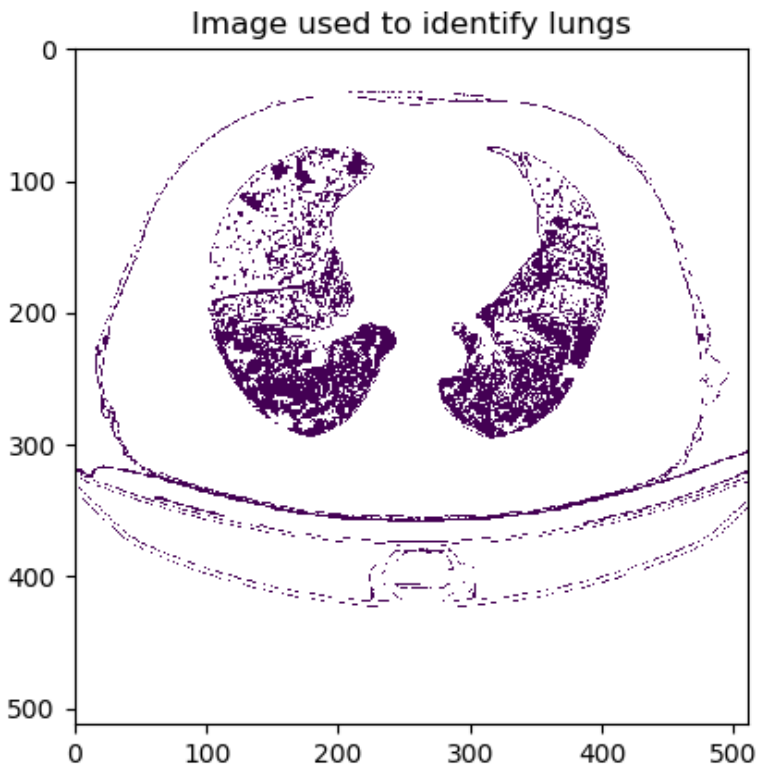


K-Means [6]

The image with quantized colors is not so different from the original one, but only 5 colors are present, and it is much easier to identify the ground glass opacities, whose label is that of the second darkest color.
Note that the mapping between label and quantized color is random.
Therefore, it is first necessary to sort the quantized colors (i.e. the centroids) and then find the label of the second smaller centroid:

```
79 ifind=1# second darkest color
80 ii=kmeans_centroids.argsort()# sort centroids from lowest to highest
81 ind_clust=ii[ifind]# get the index of the desired cluster
82 ind=(kmeans.labels_==ind_clust)# get the indexes of the pixels having the desire
83 D=A*np.nan
84 D[ind]=1# set the corresponding values of D to 1
85 D=D.reshape(Nr,Nc)# make D an image/matrix through reshaping
86 plt.figure()
87 plt.imshow(D)
88 plt.title('Image used to identify lungs')
```

K-Means [7]



Note that pixels whose value is `nan` are white in the images; pixels whose value is 1 are purple.

K-Means [8]

Unfortunately not only the lungs have this color, but also the cradle where the patient lies on and other areas of the image.
DBSCAN can be used to separate these regions, provided that parameters `eps` and `min_samples` are correctly set. DBSCAN will generate one cluster corresponding to the cradle alone, another cluster for the left lung, another cluster for the right lung, etc, starting from the image in the previous slide.

DBSCAN [1]

It is convenient to write a function, because DBSCAN will be used several times in the code.
To instatiate **DBSCAN**, use
`cluster=DBSCAN(eps=epsval,min_samples=minval,metric='euclidean')`
where `eps=epsval` is the radius of the hypersphere that includes points directly reachable from a given core point **P** (at the center of the hypersphere), and `min_samples=minval` is the minimum number of points included in the hypersphere to get a core point instead of an edge point. To run the algorithm, use
`cluster.fit(A)`
where `A` must be a 2D array.

DBSCAN [2]

In our case, reasonable parameters for DBSCAN are:
`eps=2,min_samples=5`.
The input of DBSCAN now is the set of coordinates (x, y) of the pixels that belong to the desired cluster of K-means (the Euclidean distance between two points is the distance in pixels between the two points).
DBSCAN might find isolated points (detected as outliers and labelled with -1), and it finds other clusters, the number depending on the specific slice of CT-scan.
In general, the **two most populated clusters** generated by BSCAN correspond to the two lungs. Of course the centroid/mean of the right lung has a second coordinate that is higher than that of the right lung. This detail can be used to identify left and right lungs.

DBSCAN [3]

```
def useDBSCAN(D,z,epsv,min_samplesv):
    Nr,Nc=D.shape
    clusters =DBSCAN(eps=epsv,min_samples=min_samplesv,metric='euclidean')
    clusters.fit(z)
    Nclust_DBSCAN=np.max(clusters.labels_)+1
    Npoints_per_cluster=np.zeros((Nclust_DBSCAN,),dtype=int)
    for k in range(np.max(clusters.labels_)):
        ii=(clusters.labels_==k)
        Npoints_per_cluster[k]=np.sum(ii)
    ii=np.argsort(-Npoints_per_cluster)# from the most to the less populated
    Npoints_per_cluster=Npoints_per_cluster[ii]
    C=np.zeros((Nr,Nc,Nclust_DBSCAN))*np.nan # one image for each cluster
    info=np.zeros((Nclust_DBSCAN,5),dtype=float)
    for k in range(Nclust_DBSCAN):
        i1=ii[k]
        index=(clusters.labels_==i1)
        jj=z[index,:] # coordinates of cluster k
        C[jj[:,0],jj[:,1],k]=1 # Ndarray k stores cluster k
        a=np.mean(jj,axis=0).tolist()
        b=np.var(jj,axis=0).tolist()
        info[k,0:2]=a #  store coordinates of the centroid
        info[k,2:4]=b # store variance
        info[k,4]=Npoints_per_cluster[k] # store points in cluster
    return C,info,clusters
```

DBSCAN [4]

In the main part of the script, write

```
89 C, centroids , clust=useDBSCAN(D, np.argwhere(D==1),2,5)
```

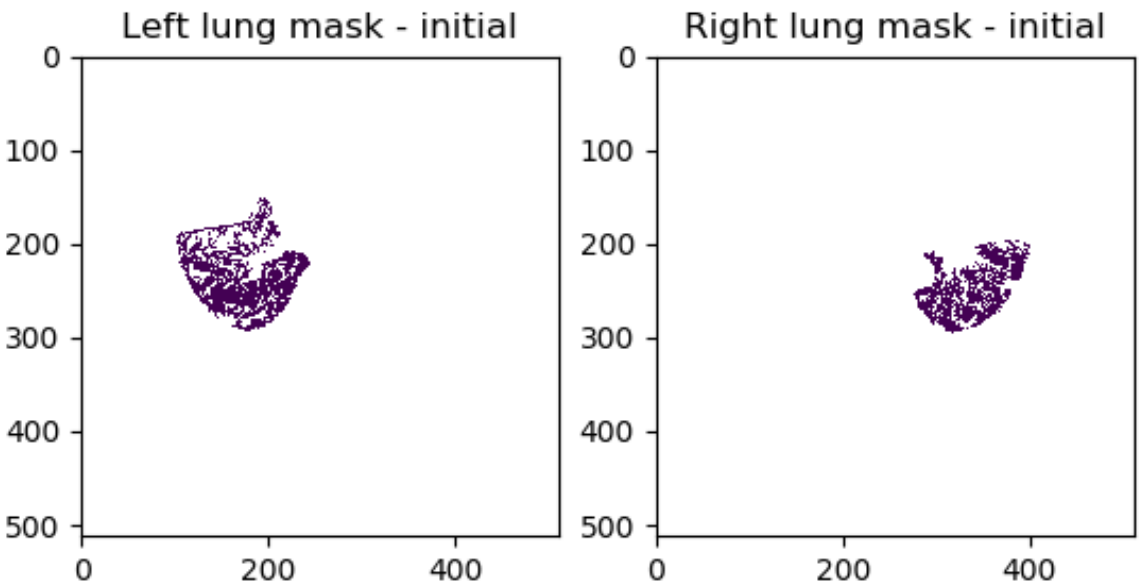
We want to identify left and right lungs. We know that Ndarray `C` stores the images of the two lungs in `C[:,:,0]` and `C[:,:,1]`. Of course left lung is on the left and the right lung is on the right, which means the lung in `C[:,:,0]` is the left one if `centroids[0,1]<centroids[1,1]`, otherwise, we swap them.

```
90 if centroids[1,1]<centroids[0,1]:# swap the two subimages
91     print('swap')
92     tmp = C[:,:,0]*1
93     C[:,:,0] = C[:,:,1]*1
94     C[:,:,1] = tmp
95     tmp=centroids[0,:]*1
96     centroids[0,:]=centroids[1,:]*1
97     centroids[1,:]=tmp
98 LLung = C[:,:,0].copy()  # left lung
99 RLung = C[:,:,1].copy()  # right lung
100 fig, (ax1,ax2) = plt.subplots(1,2)
101 ax1.imshow(LLung)
```

DBSCAN [5]

```
102 ax1.set_title('Left lung mask - initial')
103 ax2.imshow(RLung)
104 ax2.set_title('Right lung mask - initial')
```

DBSCAN [6]


Left lung mask - initial
Right lung mask - initial
0
100
200
300
400
500
0
200
400

DBSCAN [7]

Result is not so good: only a portion of the lung is found and ”holes” are present inside. However left and right lungs are separated. To improve the results, it is convenient to start again from the beginning, using as input of DBSCAN not only the positions of the second darkest color, but also of the darkest color (the upper part of lungs is black, actually). But now we approximately know the lung positions.

DBSCAN [8]

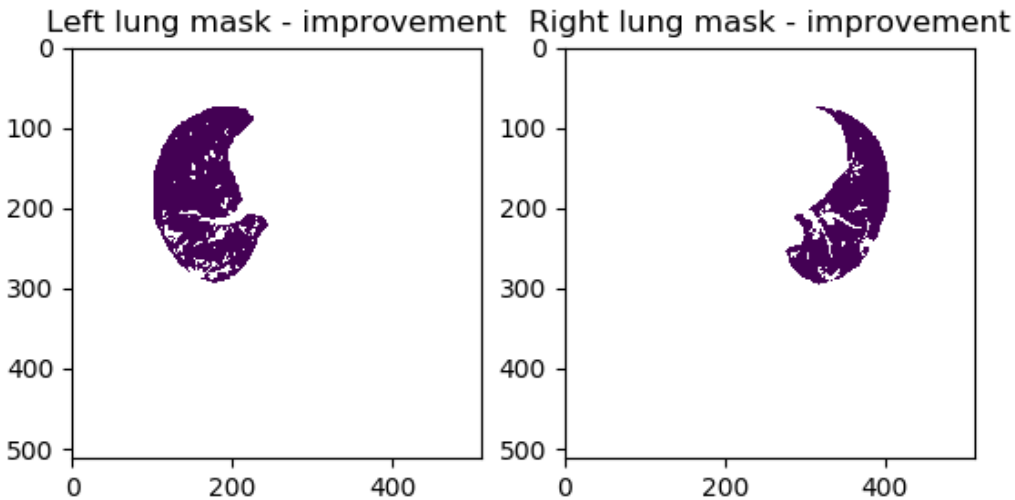
```
105 D=A*np.nan
106 ii=kmeans_centroids.argsort()# sort centroids from lowest to highest
107 ind=(kmeans.labels_==ii[0])# indexes of pixels with darkest color
108 D[ind]=1# set the corresponding values of D to 1
109 ind=(kmeans.labels_==ii[1])# indexes of pixels with the 2nd darkest color
110 D[ind]=1# set the corresponding values of D to 1
111 D=D.reshape(Nr,Nc)# make D an image/matrix through reshaping
112 C,centers2,clust=useDBSCAN(D,np.argwhere(D==1),2,5)
113 ind=np.argwhere(centers2[:,4]<1000) # remove small clusters
```

DBSCAN [9]

Among the found clusters we must select the lungs, whose centroids are already available (at least approximatively).

```
114 centers2=np.delete(centers2,ind,axis=0)
115 distL=np.sum((centroids[0,0:2]-centers2[:,0:2])**2,axis=1)
116 distR=np.sum((centroids[1,0:2]-centers2[:,0:2])**2,axis=1)
117 iL=distL.argmin()
118 iR=distR.argmin()
119 LLungMask=C[:,:,iL].copy()
120 RLungMask=C[:,:,iR].copy()
121 fig, (ax1,ax2) = plt.subplots(1,2)
122 ax1.imshow(LLungMask)
123 ax1.set_title('Left lung mask - improvement')
124 ax2.imshow(RLungMask)
125 ax2.set_title('Right lung mask - improvement')
```

DBSCAN [10]



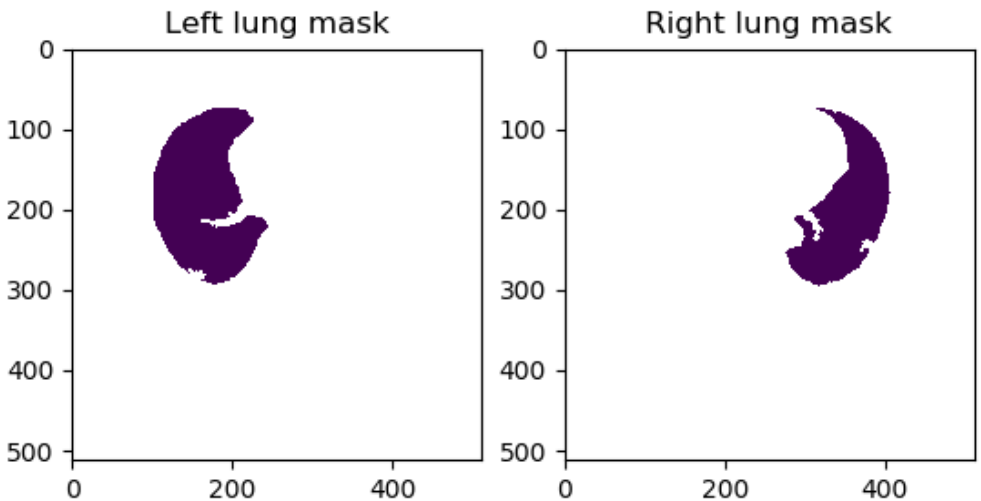
The two lungs are better identified, but still holes are present inside them.

DBSCAN [11]

We can remove holes by applying DBSCAN on the coordinates of the above images/Ndarrays where pixels are equal to `nan`: there will be a very big cluster that surrounds each lung and small clusters inside the lungs. By picking only the largest cluster we find what is outside the lungs, and therefore we identify the lungs.

```
126 C,centers3,clust=useDBSCAN(LLungMask,np.argwhere(np.isnan(LLungMask)),1,5)
127 LLungMask=np.ones((Nr,Nc))
128 LLungMask[C[:,:,0]==1]=np.nan
129 C,centers3,clust=useDBSCAN(RLungMask,np.argwhere(np.isnan(RLungMask)),1,5)
130 RLungMask=np.ones((Nr,Nc))
131 RLungMask[C[:,:,0]==1]=np.nan
132 fig, (ax1,ax2) = plt.subplots(1,2)
133 ax1.imshow(LLungMask)
134 ax1.set_title('Left lung mask')
135 ax2.imshow(RLungMask)
136 ax2.set_title('Right lung mask')
```

DBSCAN [12]



The masks are now substantially correct, apart from a couple of notches (very light color in the lung border in the original image). Note that any measuring instrument makes error, and what we are developing is a software instrument.

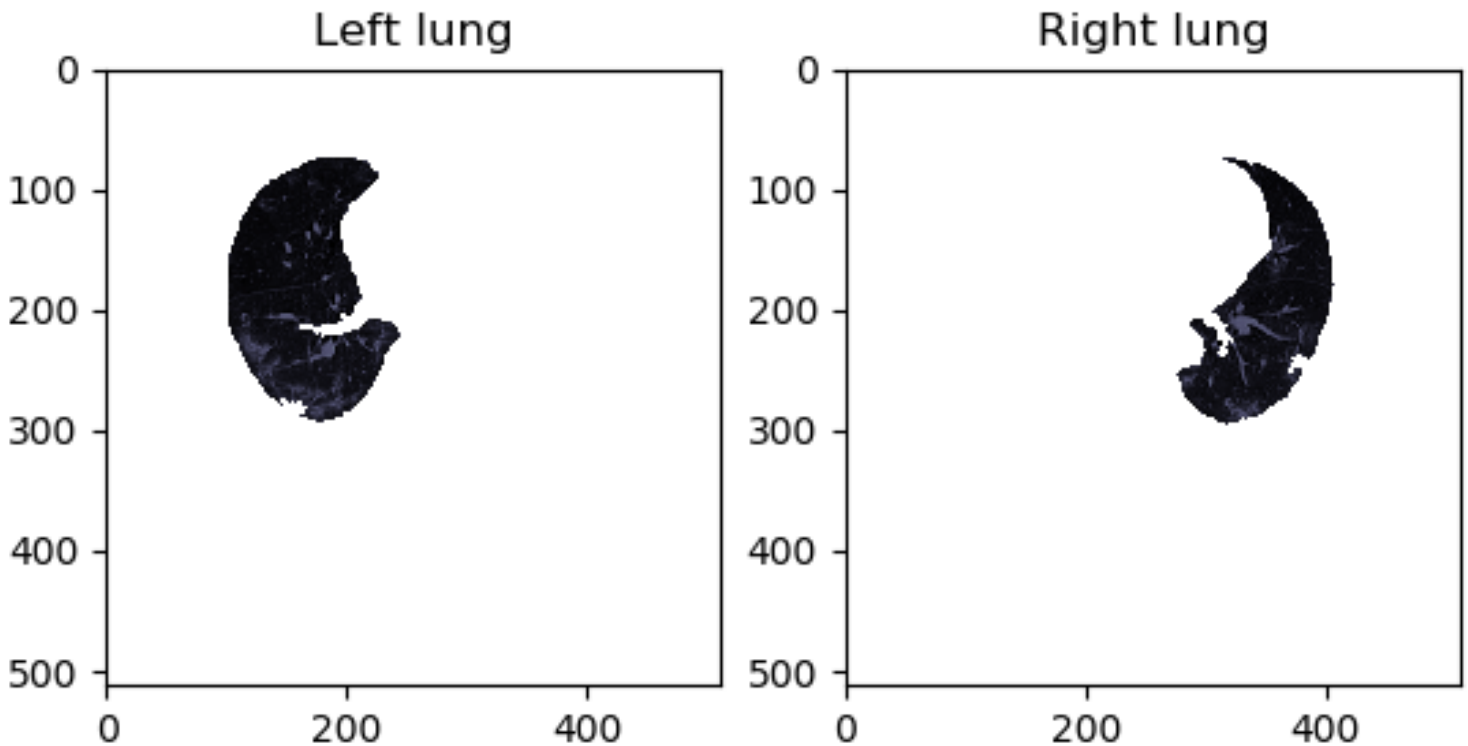
Table of Contents

Infection mask [1]

Let us look at the original image, corresponding to the found lung masks, using the 'bone' map and the original color scale:

```
137 fig, (ax1,ax2) = plt.subplots(1,2)
138 ax1.imshow(LLungMask*sct,vmin=vm,vmax=vM, cmap='bone')
139 ax1.set_title('Left lung')
140 ax2.imshow(RLungMask*sct,vmin=vm,vmax=vM, cmap='bone')
141 ax2.set_title('Right lung')
```

Infection mask [2]

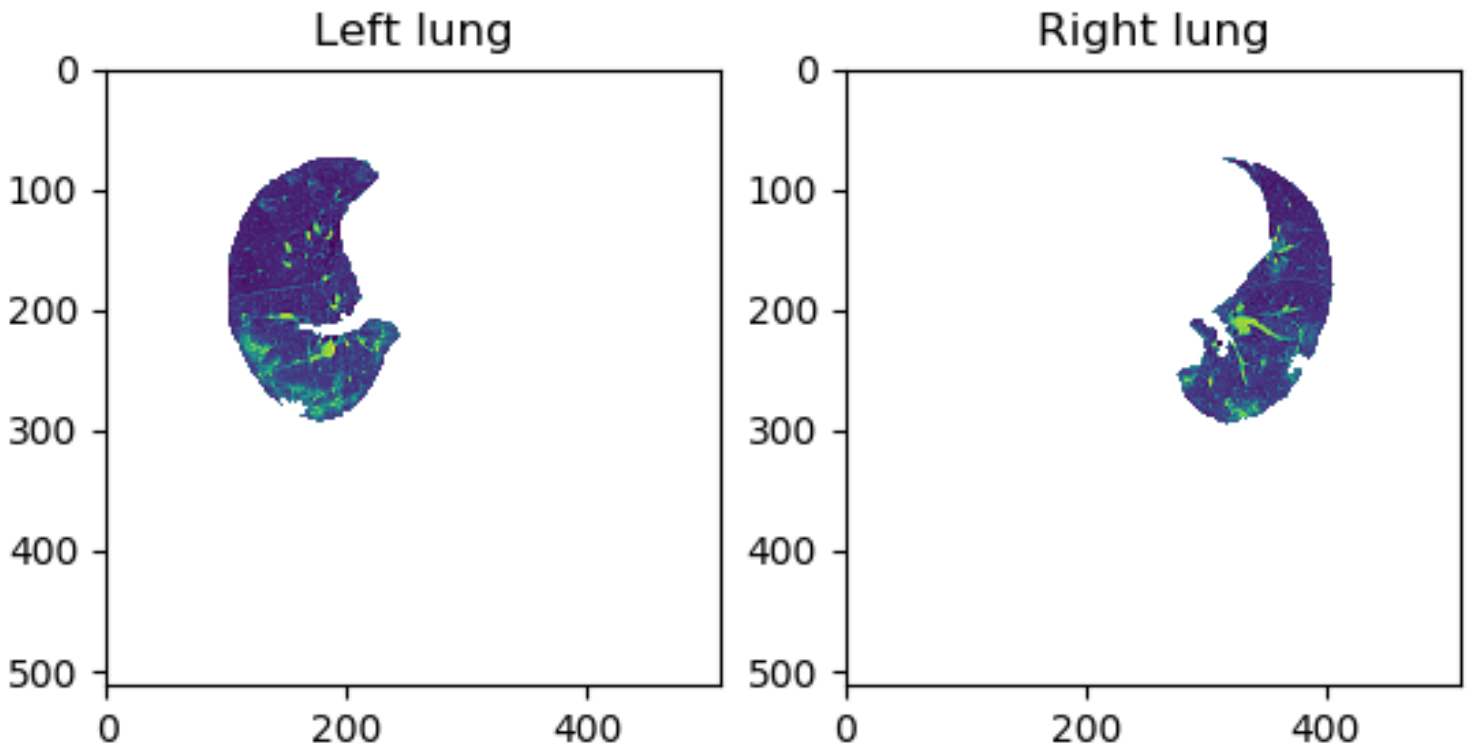


Infection mask [3]

Let us look at the lungs without the 'bone' colormap:

```
141 fig, (ax1,ax2) = plt.subplots(1,2)
142 ax1.imshow(LLungMask*sct)
143 ax1.set_title('Left lung')
144 ax2.imshow(RLungMask*sct)
145 ax2.set_title('Right lung')
```

Infection mask [4]



Infection mask [5]

Lungs in the previous slide more clearly show that the ground glass opacities simply correspond to a range of color values. In particular, a reasonable range is from -650 to -300. The obtained infection mask, however, is too scattered, and it is convenient to filter it, i.e. substitute to the value in position i, j the average of the values in a square of 9 pixels, having i, j in the center.

Infection mask [6]

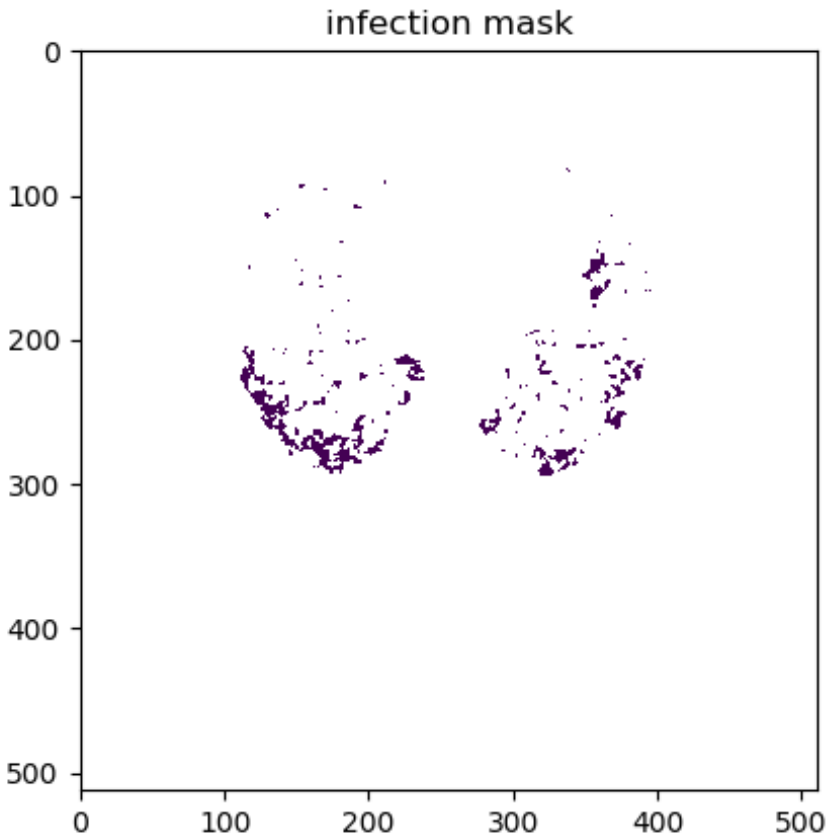
Function to implement filtering:

```
def filterImage(D,NN):
    """D = image (matrix) to be filtered, Nr rows, N columns,
    scalar values (no RGB color image)
    The image is filtered using a square kernel/impulse response with
    side 2*NN+1"""
    from itertools import product
    E=D.copy()
    E[np.isnan(E)]=0
    Df=E*0
    Nr,Nc=D.shape
    rang=np.arange(-NN,NN+1)
    square=np.array([x for x in product(rang, rang)])
    #square=np.array([[1,1],[1,0],[1,-1],[0,1],[0,0],
    #[0,-1],[-1,1],[-1,0],[-1,-1]])
    for kr in range(NN,Nr-NN):
        for kc in range(NN,Nc-NN):
            ir=kr+square[:,0]
            ic=kc+square[:,1]
            Df[kr,kc]=np.sum(E[ir,ic])
    return Df/len(square)
```

Infection mask [7]

```
146 # it is necessary to substitute nan with a number
147 LLungMask[np.isnan(LLungMask)]=0
148 RLungMask[np.isnan(RLungMask)]=0
149 LungsMask=LLungMask+RLungMask #  overall mask
150 B=LungsMask*sct #  original image of lungs
151 inf_mask=1*(B>-650)&(B<-300) #  selected color interval
152 InfectionMask=filterImage(inf_mask,1) # filter
153 InfectionMask=1.0*(InfectionMask>0.5)
154 InfectionMask[InfectionMask==0]=np.nan
155 plt.figure()
156 plt.imshow(InfectionMask)
157 plt.title('infection mask')
```

Infection mask [8]



This infection mask is less smooth than that given in the dataset, but it might be more realistic, in the end.

Infection mask [9]

Final plot:

```
158 color_map = 'spring'
159 plt.figure()
160 plt.imshow(sct,alpha=0.8,vmin=vm,vmax=vM, cmap='bone')
161 plt.imshow(InfectionMask*256,alpha=1,vmin=0,vmax=256, cmap=color_map)
162 plt.title('Original image with ground glass opacities in yellow')
```

Infection mask [10]

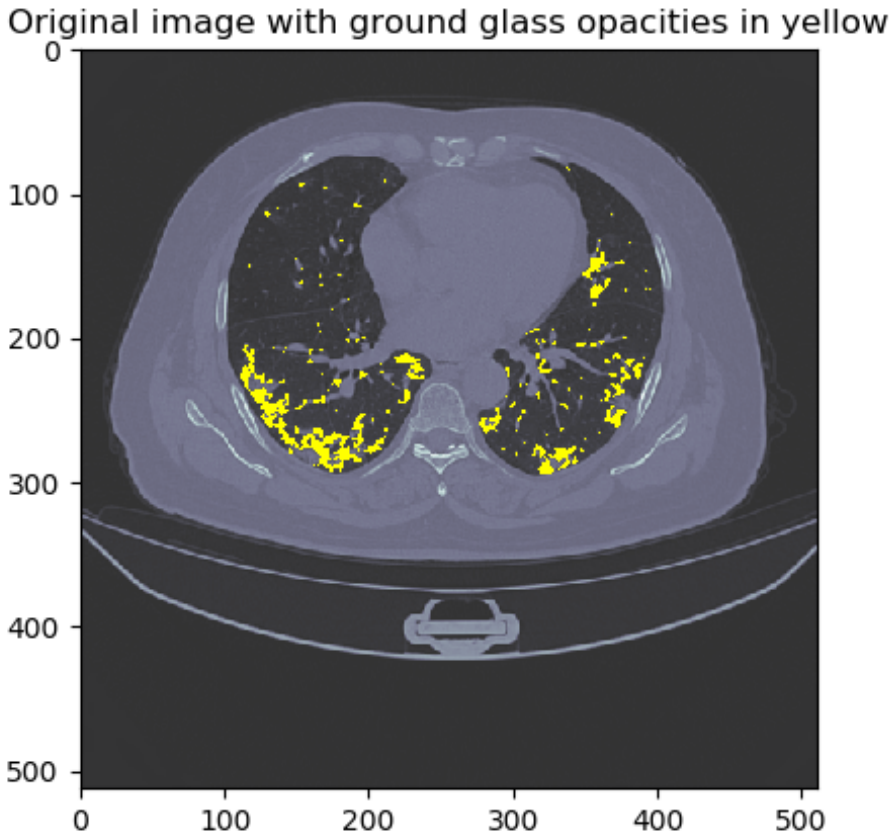


Table of Contents

Your job

1. Run the code for a slice from 125 to 135 (one slice, choose the one you like the most in this range).
2. Invent a feature that can be given to the radiologist to understand how much of the lungs is infected (one number, a measurement).
3. Include in the code the lines that are necessary to measure your selected feature.
4. Complete the report, including your own images. Describe the feature you decided to use and include the value of this feature for your slice. Critically discuss the obtained results.
5. Report due in 14 days, i.e. December 3rd 2020, 11.59 PM.