Merged_BMI260 2019 Problem Set 1

April 10, 2019

1 BIOMEDIN 260/RAD260: Problem Set 1 - Segmentation

1.1 Spring 2019

1.2 Group Work

Before we go any further: we encourage teamwork, and you can work by yourself or in pairs (2 people working together equally and submitting one notebook for the both of you). Who said radiology had to be lonely?

Person 1:
Xingchen Liu
Person 2:
Xi Yu

1.3 iPython Notebooks: A lesson in reproducibility

For those of you that are unfamiliar, iPython notebooks are interactive Python sessions that allow you to intersperse code, raw text, and markdown in a seamless manner. The next paragraph will teach you to use them.

The words you are reading right now are modifiable. Double-click me to change the text that you're reading. These modifiable boxes are called cells. They will be used to write answers to our written questions.

Double-click the box above me now, and go ahead and fill in your name (and your partner's as well, if applicable).

Cells can contain executable code as well. If you're running the Jupyter Notebook program, You can change the type of cell by highlighting the cell of interest and going to `Cell -> Cell Type -> Markdown/Code`, and then clicking the desired cell type. If you're running the newer JupyterLab program, just click the drop-down menu next to the play, stop, and refresh icons above this window (along the top of the tabs detailing the current files that are open) and change the cell type directly there.

You can add cells using Insert in the Jupyter Notebook program and the `'+'` icon in JupyterLab.

The best thing about notebooks is that you can run small components of your code in one cell to make sure they work before putting them together to make a larger component. Make as many cells as you want to play and experiment, but please delete them if they are not part of your final submission. To delete a cell, highlight the cell and going to `Edit -> Delete Cells`, or use the keyboard shortcut of pressing the 'D' button twice. Make sure to not do this by mistake, but if you do, you can `"Redo Cell Operation"` from the respective menus in the different programs.

Another nice thing about notebooks - the entire homework assignment is self-contained in this file! Please put all your functions and classes into the cells of this notebook, and please write clean code with at least some annotation to help us follow your thought process. Once you're done, export the notebook to a `.pdf` file and submit on Canvas.

In research, it is important that code is readable and reproducible. Notebooks are a natural first step toward both goals.

2 Setup

Run the code in the following cells by single clicking on them and then press CTRL+ENTER (SHIFT+ENTER on Mac) or click the play button in the menu bar (the one to the left of the stop button and to the right of the up and down arrows).

The following first cell loads all the Python dependencies for this homework. It's OK if you get an error at first.

```
In [1]: import os
        import numpy as np
        import pydicom as dicom
        import matplotlib.pyplot as plt
        import os
        import skimage
        %matplotlib inline
```

2.1 Did you get an error like this?

```
### ModuleNotFoundError: No module named 'skimage'/'numpy'/'pydicom'?
```

It means you need to install some more dependencies. For example, PyDicom is not usually included.

Here is how to install PyDicom given an Anaconda Python distribution, which we've asked you to get for this quarter, and here is some PyDicom documentation which might be useful later on in the assignment.

The other dependencies like scikit-image are installed similarly.

If you don't have an Anaconda Python distribution, you may have to use `sudo` like this:

`$ sudo pip3 install pydicom`

and likewise for `skimage`:

`$ sudo pip3 install scikit-image`

You can also install packages from within the Jupyter kernal. Try running the code in the cell below, which attempts to install NumPy:

```
In [2]: # Install a pip package in the current Jupyter kernel
        import sys
        !{sys.executable} -m pip install numpy
```

```
Requirement already satisfied: numpy in /anaconda3/envs/bmi260/lib/python3.6/site-packages (1.
```

3 The Data

We will be using data from the recent (2017) Kaggle Data Bowl challenge in this homework. You can download their data freely from Kaggle or here. I highly recommend downloading the "`sample images.7z`" file as this'll give you full CT scans, but won't take a day to download (total size < 1 GB).

Please download and unzip the data folder as soon as possible. Even the file for just the sample data is pretty big, coming in at 800 MB. You need the data to do the assignment. We will not accept your homework using any other data. Please try to download the data as soon as possible, and contact the TAs if you run into any problems. Do not wait until the last minute to download the data!

The data consist of sets of DICOM images that hold completely anonymized chest CT scans (see section "Reading in the Data"). DICOM (Digital Imaging and Communications in Medicine) is a standardized format for transmitting medical image data. Each DICOM file is composed of two parts: the image data as well as a header giving you a lot of metadata about the patient and specifics about the imaging parameters (e.g. space between slices).

You can choose any of the patients in the Kaggle dataset (sample, training, or validation) to prototype your code, but we expect you to generalize your algorithm to work for at least ten different patients to show robustness. The more you do, the more you'll prove your algorithm.

First, let's make sure you can read in the data.

In the cell below, replace the `scans_path` string variable with the (relative or absolute) path to the place where you put the data. Particularly, look for the folder that contains scans. Each scan is in its own folder and has a name, e.g. 0a0c32c9e08cc2ea76a71649de56be6d. `scans_path` should be the folder that contains these oddly named folders. Once you do this, you can run the cell below to look at all the metadata associated with that slice.

It should look like this:

```
In [3]: # each chest CT scan is a folder with a long weird name like 0a0c32c9e08cc2ea76a71649d
        # many .dcm files with similar long weird names, assign the path to such folder to the

        scans_path = "/Users/neoliu/Documents/bmi260/Problem Set 1/CT_chest_scans/"
        list_of_scans = os.listdir(scans_path)

        # for figuring out the controls lets experiment with slice 122 of slice 2
        scan_num = 3
        scan_path = os.path.join(scans_path,list_of_scans[scan_num])
        list_of_slices = os.listdir(scan_path)
        slice_num = 0
        slice_path = os.path.join(scan_path,list_of_slices[slice_num])

        # read in the full path to the file as ds
        ds=dicom.read_file(slice_path) # you may have to use pydicom instead of dicom
        print(ds)

        # for scan in list_of_scans:
        #     if scan.startswith('.DS'): continue
        #     scan_path = os.path.join(scans_path,scan)
        #     slices, num = get_slices(scan_path)
```

```
        #     a=[]
        #     for slice in slices:
        #         d = dicom.read_file(os.path.join(scan_path, slice))
        #         a.append(d.SliceLocation)
        #     a=np.asarray(a)
        #     a.sort()
        #     print(a-np.append(a[1:],a[0]))
```

```
(0008, 0005) Specific Character Set              CS: 'ISO_IR 100'
(0008, 0016) SOP Class UID                       UI: CT Image Storage
(0008, 0018) SOP Instance UID                    UI: 1.2.840.113654.2.55.164625047581783478378
(0008, 0060) Modality                            CS: 'CT'
(0008, 103e) Series Description                  LO: 'Axial'
(0010, 0010) Patient's Name                      PN: '0a0c32c9e08cc2ea76a71649de56be6d'
(0010, 0020) Patient ID                          LO: '0a0c32c9e08cc2ea76a71649de56be6d'
(0010, 0030) Patient's Birth Date                DA: '19000101'
(0018, 0060) KVP                                 DS: ''
(0020, 000d) Study Instance UID                  UI: 2.25.60037070027156423276159501017920151
(0020, 000e) Series Instance UID                 UI: 2.25.58703274222857573910779974742342423
(0020, 0011) Series Number                       IS: "1"
(0020, 0012) Acquisition Number                  IS: "1"
(0020, 0013) Instance Number                     IS: "26"
(0020, 0020) Patient Orientation                 CS: ''
(0020, 0032) Image Position (Patient)            DS: ['-160.100006', '-142.500000', '-45.91000
(0020, 0037) Image Orientation (Patient)         DS: ['1', '0', '0', '0', '1', '0']
(0020, 0052) Frame of Reference UID              UI: 2.25.10816424791949633522058525471061381
(0020, 1040) Position Reference Indicator        LO: 'SN'
(0020, 1041) Slice Location                      DS: "-45.910000"
(0028, 0002) Samples per Pixel                   US: 1
(0028, 0004) Photometric Interpretation          CS: 'MONOCHROME2'
(0028, 0006) Planar Configuration                US: 0
(0028, 0010) Rows                                US: 512
(0028, 0011) Columns                             US: 512
(0028, 0030) Pixel Spacing                       DS: ['0.664062', '0.664062']
(0028, 0100) Bits Allocated                      US: 16
(0028, 0101) Bits Stored                         US: 12
(0028, 0102) High Bit                            US: 11
(0028, 0103) Pixel Representation                US: 0
(0028, 0120) Pixel Padding Value                 US: 0
(0028, 1050) Window Center                       DS: "40"
(0028, 1051) Window Width                        DS: "400"
(0028, 1052) Rescale Intercept                   DS: "-1024"
(0028, 1053) Rescale Slope                       DS: "1"
(0028, 1054) Rescale Type                        LO: 'HU'
(7fe0, 0010) Pixel Data                          OW: Array of 524288 bytes
```

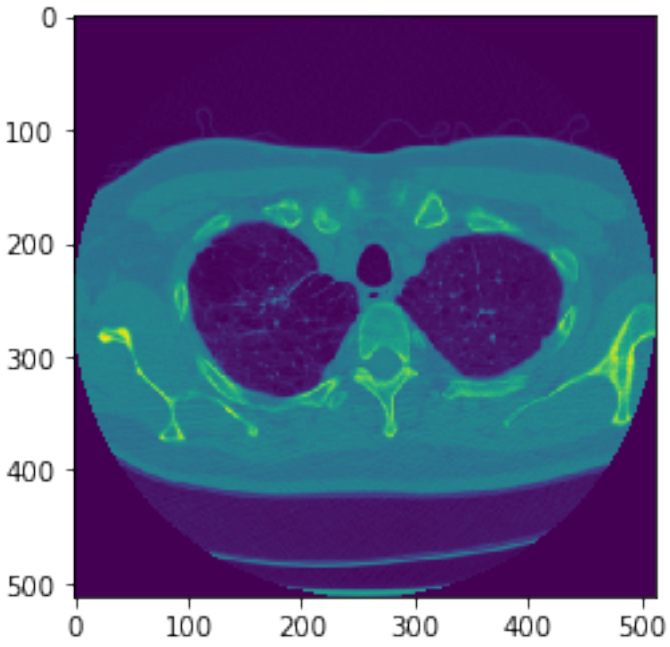
3.1 That's a lot of metadata, right? Don't be scared. Next, let's actually look at the slice.

We can see that there are many data fields in a DICOM file. There's a lot of patient information stored in a DICOM - name, birthdate, and so on. For obvious privacy reasons, this data has been completely anonymized. It's pretty evident that our data probably didn't come from Mr./Mrs. 0a0c32c9e08cc2ea76a71649de56be6d who was born on January 1st, 1900.

Run the following cell to view the image associated with this very old man/woman.

```
In [4]: rawimg= ds.pixel_array
        print(type(ds))
        print(type(ds.pixel_array))
        plt.imshow(rawimg, cmap='viridis')
        plt.show()
        print(type(rawimg), np.mean(ds.pixel_array), rawimg.dtype)
```

```
<class 'pydicom.dataset.FileDataset'>
<class 'numpy.ndarray'>
```




```
<class 'numpy.ndarray'> 508.58791732788086 uint16
```

We're all set. On to the problem set!

4 Introduction

Segmentation is the process of dividing a given image into sections. This can be binary segmentation (if you wanted to separate the image into foreground and background) or a multilabel segmentation if you wanted to label many different parts of an image (see figures below for examples of each). Segmentation is an extremely interesting problem in image analysis, and it has yet to be solved perfectly. From an algorithmic point of view, we can approach the problem using tools such as graph cuts, watershed, flood fill / region growing, statistical set separation, etc... From a machine learning point of view, we can think of the problem as a pixel classification problem that can be solved with some supervised learning algorithm (assuming we had some ground truth training data) or an unsupervised learning algorithm (such as k-means clustering).

(a) Binary Segmentation - A binary segmentation of some grains of rice. We can use segmentation to create a binary mask that allows us to separate the rice (foreground) from the darker background.

(b) Multilabel Segmentation - A multilabel segmentation of a coronal section from a magnetic resonance image of the brain. We can see that different colors represent different sections of the brain. For example, the white matter on the left side of the brain is labeled with a different color than the white matter on the right side of the brain.

A lot of radiological image analysis begins with segmentation. For example, to quantitatively describe a lung nodule, you may want to collect data on its edge sharpness, its total volume, and/or the mean intensity of voxels in the nodule. However, to do this, we have to first be able to segment the nodule and cleanly define its boundaries. This can get extremely hairy. For example, in the chest, it might be hard to teach a program to segment a nodule in the lung from a seed voxel. It's important to make sure the nodule segmentation doesn't bleed into the pericardium, as the voxel intensities for a nodule and the fleshy bits in the pericardium are very similar. We can try and restrict this through lung field segmentation (basically, we want to go through our chest CT scans and segment out the lungs) and thus define a region of interest, but this gets increasingly difficult as more and more problems pop up.

This assignment will help you understand many core concepts taught in the lectures and hopefully integrate the lectures into a medically relevant application. The goals of this project include:

a. Understanding medical image data format - specifically, DICOM.

b. Understanding thresholding techniques - specifically, Otsu's Threshold.

c. Understanding morphological image analysis techniques, and expanding this understanding into 3D.

d. Understanding image convolution and expanding it into efficient 3D.

e. Understanding visualization in 3D.

f. Embracing creativity and exploring image analysis.

These can seem daunting tasks for a first timer, but we promise the final results will be amazingly cool! Stick with us, and as always, don't hesitate to ask for help.

The main goal of this assignment is to segment out the lungs, and only the lungs, from these CT slices, and then create a 3D rendering of the lungs to give yourself some way to qualitatively assess how good your segmentation is. Basically, given a chest CT series, you will be creating something like this:

5 Step 1: (5 points)

Some important DICOM fields like *RescaleIntercept* and *RescaleSlope* determine how the image pixel values should be interpreted. These metadata are critical for quantitative imaging methods like CT. The default raw pixel values are arbitrary units returned from the actual machine used and may differ based on the scanner manufacturer. We'd like to convert these raw values to Hounsfield units, which you learned about in lecture, in order to have some standardization among all of our CT scans. The conversion formula is:

$$\text{Hounsfield Units} = \textit{RescaleSlope} \times \text{Raw Image} + \textit{RescaleIntercept}$$

5.0.1 Deliverable:

Read in the raw data for a CT slice, and convert its pixel values into Hounsfield units. Compute the max, min, mean, and standard deviation of both images (raw data and Hounsfield units) and compare them.

In [5]:
```
### WRITE CODE IN HERE. You can have up to 2 cells for this question, but only one is

        def printimage(img, name): # print image attrs
            print('{:30s} max={:8.2f}  min={:8.2f}  mean={:8.2f}  std={:8.2f}'.format(
                name+":", img.max(), img.min(), img.mean(), img.std()))

        printimage(ds.pixel_array, 'Original image')

        def get_hounsfield_img(ds):
            raw_img= ds.pixel_array
            converter = lambda x: x*ds.RescaleSlope + ds.RescaleIntercept
            return converter(raw_img)

        hounsfield_img = get_hounsfield_img(ds)

        printimage(hounsfield_img, 'Converted Hounsfield image')

        print("Comparing:")
        print(" HU_img_max = RescaleSlope*rawimg_max + RescaleIntercept\n",
            "HU_img_min = RescaleSlope*rawimg_min + RescaleIntercept\n",
            "HU_img_mean = RescaleSlope*rawimg_mean + RescaleIntercept\n",
            "HU_img_std = rawimg_std")

        ##########################################################################################
```

```
Original image:                max= 2437.00  min=    0.00  mean=  508.59  std=  518.57
Converted Hounsfield image:    max= 1413.00  min=-1024.00  mean= -515.41  std=  518.57
Comparing:
 HU_img_max = RescaleSlope*rawimg_max + RescaleIntercept
 HU_img_min = RescaleSlope*rawimg_min + RescaleIntercept
 HU_img_mean = RescaleSlope*rawimg_mean + RescaleIntercept
```

```
HU_img_std = rawimg_std
```

6 Step 2: (25 points)

As you may have noticed, we live in a 3D world. The next main part of your problem will be to go from all the individual slices of the CT scan (each one stored as a .dcm file) into a 3D volume. Basically, you'll need to figure out some way to order these slices in the right order. Read in a few of the DICOM images from your patient, and try using different DICOM fields as a sorting key. You may find one of them works well in sorting the slices in the correct order. Try looking up what that field means in the DICOM standard. *Hint: you may want to initialize a blank 3D matrix called `volCT` or something.*

Here is the pseudocode for one way (you are free to do your own) of creating this volume and filling it:

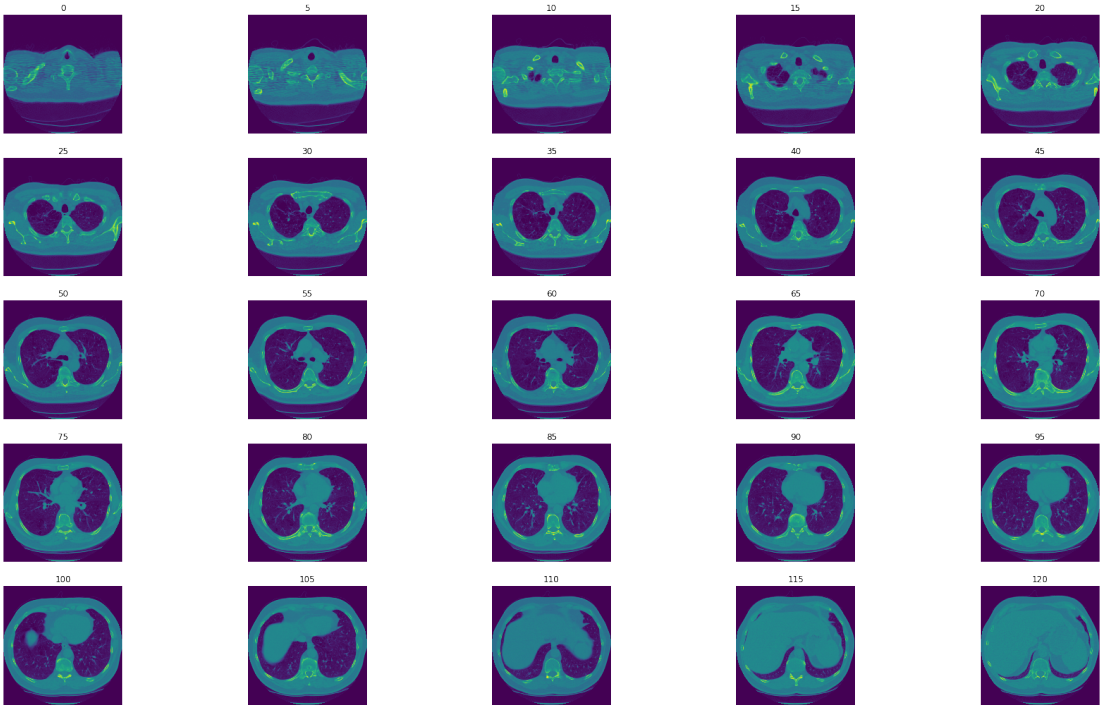
Visualize the NumPy volume you create from the stacks. Feel free to do this within Python with something like matplotlib.

6.0.1 Deliverable:

Make the 3D volume and display 25 of the slices in correct order for us, like this:

```
In [6]: ### WRITE CODE IN HERE. You can have up to 4 cells for this question, but only one is
        scan_path1 = "/Users/neoliu/Documents/bmi260/Problem Set 1/CT_chest_scans/0a0c32c9e08c
        #scan_path = "/Users/neoliu/Documents/bmi260/Problem Set 1/CT_chest_scans/0b20184e0cd4

        # Returns a 3D array of [#pixels, #pixels, #slices] for a givn CT scan
        def get_hounsfield_volume(scan_path):
            slices = os.listdir(scan_path)
            slices = [x for x in slices if x.endswith(".dcm")]
            num_slices = len(slices)

            # Get shape
            shape = None
            for slice in slices:
                f = (os.path.join(scan_path, slice))
                df = dicom.read_file(f)
                shape = df.pixel_array.shape
                break

            # Fill in volCT
            volCT = np.zeros(shape + (num_slices,))
            for slice in slices:
                f = (os.path.join(scan_path, slice))
                #print(f)
                df = dicom.read_file(f)
                num = df.InstanceNumber
                hounsfield_img = get_hounsfield_img(df)
                volCT[:,:, num-1] = hounsfield_img
```

```
    ## IMPORTANT! Remove the "black frame" from images to get a good threshold
    volCT = np.where(volCT < -1000, -1000, volCT)
    return volCT

volCT = get_hounsfield_volume(scan_path1)

plt.rcParams['figure.figsize'] = (30,18)
_fig, ax = plt.subplots(5,5)
for i in range(25):
    ax.flat[i].imshow(volCT[:,:,i*5])
    ax.flat[i].axis('off')
    ax.flat[i].set_title(i*5)
plt.show()

##########################################################################################
```



7 Step 3 (5 points)

If you dont already know about numerical types, take a look here and/or here.

7.0.1 Deliverable:

What is the default numerical type used to store raw image data in DICOM files? Answer in the cell below.

In the coding cell below that one, convert all the pixels in your volume to a float32 type number between 0.0 and 1.0; output the min, max, and dtype (datatype) of the pixels in your volume before and after you convert.

Answer: uint16 or int16

```
In [7]: ### WRITE CODE IN HERE. You can have up to 2 cells for this question, but only one is
        rt='/Users/neoliu/Documents/bmi260/Problem Set 1/CT_chest_scans/'
        f1=rt+'0a0c32c9e08cc2ea76a71649de56be6d/a6388f60224e46b59bc3ed57af292c11.dcm'
        f2=rt+'0a38e7597ca26f9374f8ea2770ba870d/9da9e6c77ee6a66c1fa5a7bad24902c5.dcm'

        print('Default dtype is: {} or {}'.format(
            dicom.read_file(f1).pixel_array.dtype, dicom.read_file(f2).pixel_array.dtype))

        print('{:10s} max={}  min={}  type={}'.format("before(HU):", volCT.max(), volCT.min(),

        def normalize(volCT):
            pos = volCT - volCT.min()
            return pos.astype(np.float32) / pos.max()
        vol = normalize(volCT)
        print('{:10s} max={}  min={}  type={}'.format("after     :", vol.max(), vol.min(), vol

        ##########################################################################################

Default dtype is: uint16 or int16
before(HU): max=1811.0  min=-1000.0  type=float64
after     : max=1.0  min=0.0  type=float32
```

8 Step 4 (10 points)

Now that we have our 3D matrix of lung CT data, we can try to segment out our lung. We know from lecture that the pixel values of CT scans are given in Hounsfield units, where lower Hounsfield units correspond to low density materials (like air) that are not highly attenuative for X-rays and higher Hounsfield units correspond to highly attenuative materials, like bone.

That red line separating the two groups of pixels is the Otsu's threshold. We can find a nice separating value of our two modes with this Otsu's method.

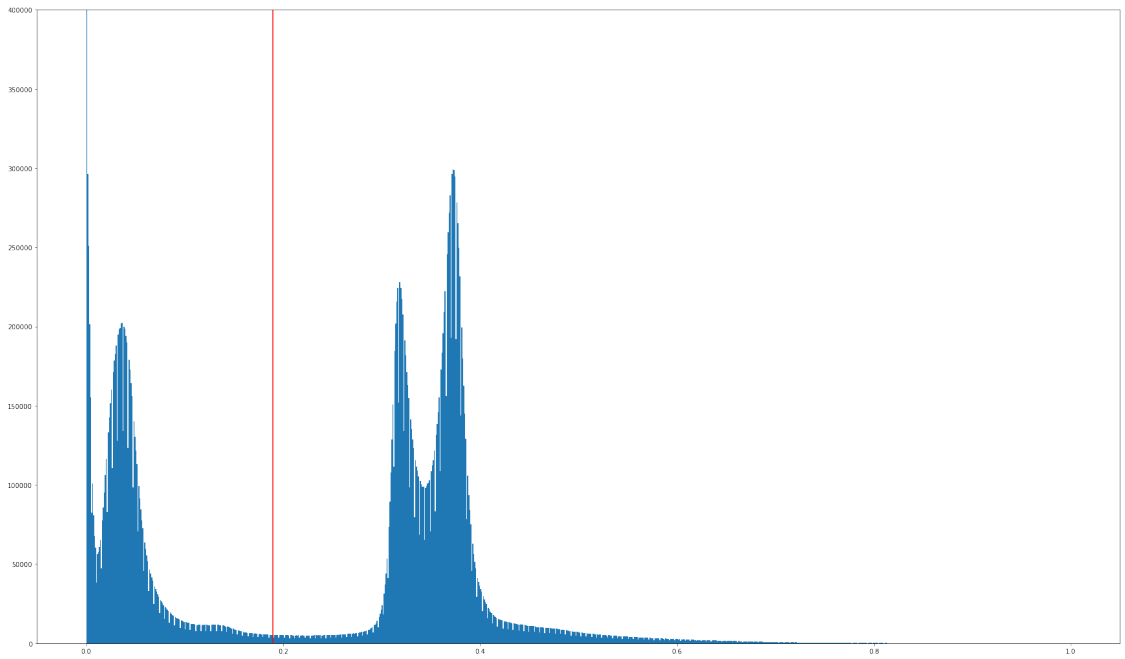
8.0.1 Deliverable:

A histogram of the Hounsfield units from a typical CT scan will be significantly bimodal! Why?

Answer: Maybe because the majority of pixels in CT scans either represent the air/lung which are lower density, or human tissue which is higher indensity???

Next, find the Otsu's threshold in your volume of pixels and plot the histogram of of your pixels with this line like the example figure. You will find these links (1, 2, and 3) useful if you have not done this before.

In [8]:
```
### WRITE CODE IN HERE. You can have up to 2 cells for this question, but only one is

_, ax = plt.subplots()
plt.hist(vol.ravel(), 1000)
plt.ylim([0,400000])

from skimage.filters import threshold_otsu
thresh = threshold_otsu(vol)
ax.axvline(x=thresh, color='r')
plt.show()

# # See Hounsfield distribution
# _, ax = plt.subplots()
# plt.hist(volCT.flatten(), 1000)
# plt.ylim([0,400000])
# ax.axvline(x=-960, color='g')
# plt.show()

##########################################################################################
```



9 Step 5 (10 points)

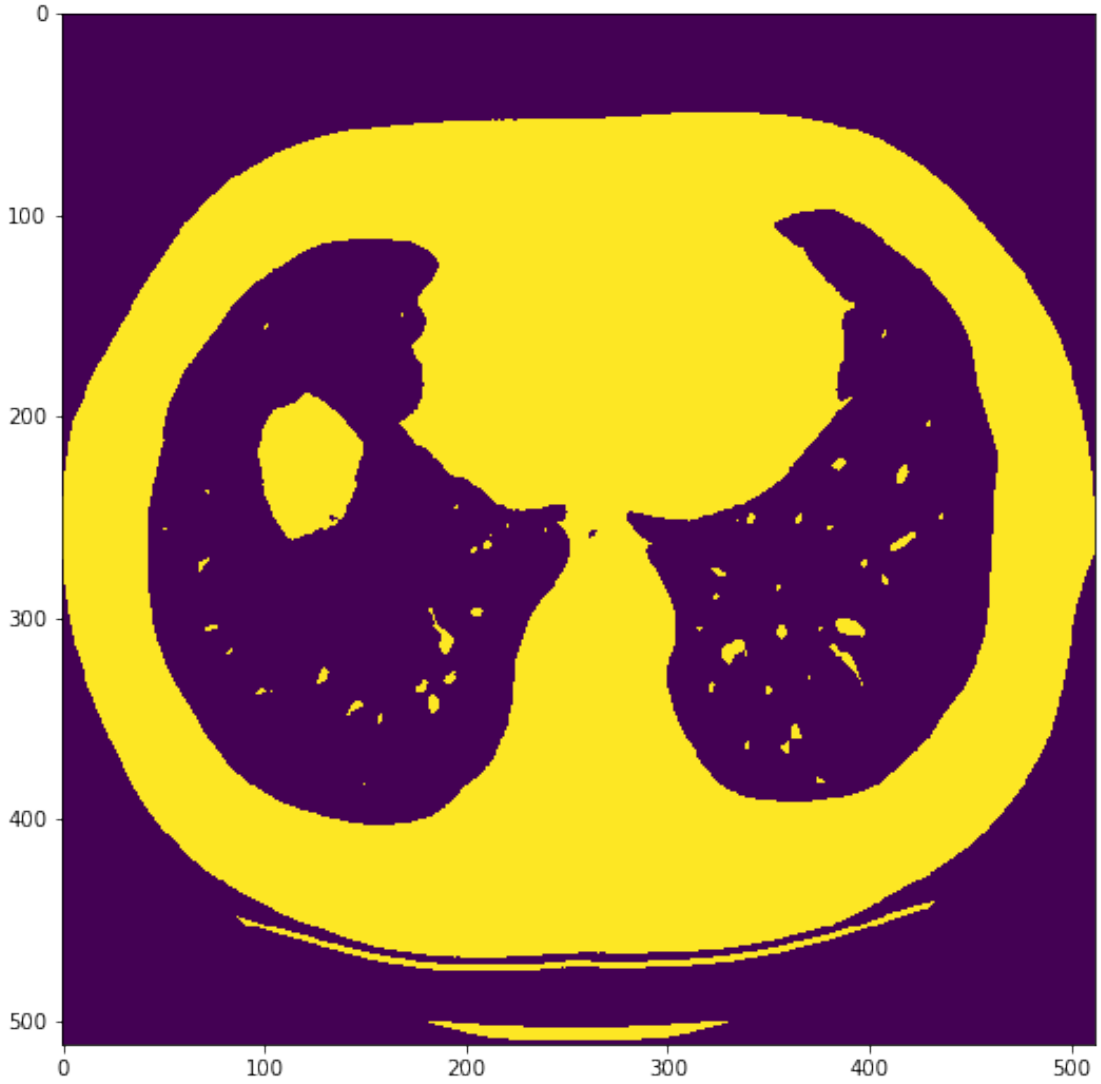
We can now take a look at the performance of Otsu's threshold. We want all the pixels less than Otsu's threshold since the lung is in the darker (low Hounsfield units) part of the image.

Despite the crude binary cutoff, we can see that this is really close to what we want for a final result, albeit a little noise.

9.0.1 Deliverable:

Display for us one slice of the CT scan showing a binary image after applying thresholding using Otsu's method.

In [9]:
```
### WRITE CODE IN HERE. You can have up to 2 cells for this question, but only one is

def show_slice(v):
    plt.rc('figure', figsize=(15,9))
    plt.imshow(v)
    plt.show()

thresh = threshold_otsu(vol)
show_slice(vol[:,:,100] > thresh)

# # try different thresholds
# _, axes = plt.subplots(5,10)
# i = 0
# for th in np.linspace(0,1,50):
#     bin = vol[:,:,25] > th
#     axes.flat[i].imshow(bin, cmap=plt.cm.gray)
#     axes.flat[i].set_title('th={:2.2f}'.format(th), fontsize=20)
#     i = i +1
# plt.show()


##########################################################################################
```



10 Step 6 (30 points)

Our dark lungs are surrounded by lighter-colored tissue, which is then again surrounded by darkness. Our lung segmentation is thus, interestingly, completely separated from the segmentation of the outside air. These two sets of binary connected components are completely unconnected. Can you think of a simple way of differentiating between the lungs and the air outside the body?

Here's one way to do it - we assume that the outside air will always be touching the edge of the image and the lungs will not:

However the segmentation is still rough around the edges. There are a few pixels of noise around the main lung (components that weren't touching the edges, but still aren't part of the lung). Luckily for us, these bits are quite small in size. We can therefore quickly filter them out based on their volume. We want to choose connected components that have a volume greater than some threshold V. You can choose V arbitrarily (most noise will have pixel counts in the double

digits or maybe low hundreds), as long as the lung volume should have a pixel count several orders of magnitude higher.

Here would be a possible pseudo-algorithm for this task:

After this, we just want to smooth out the edges. The segmentation of our lung looks very ragged because it wasn't able to pick out the lighter colored blood vessels in the lung. We can do this smoothing with a binary closing algorithm, provided by the `scikit-image` package (`skimage.morphology.binary_closing`). It works for both 2-dimensional and 3-dimensional binary matrices.

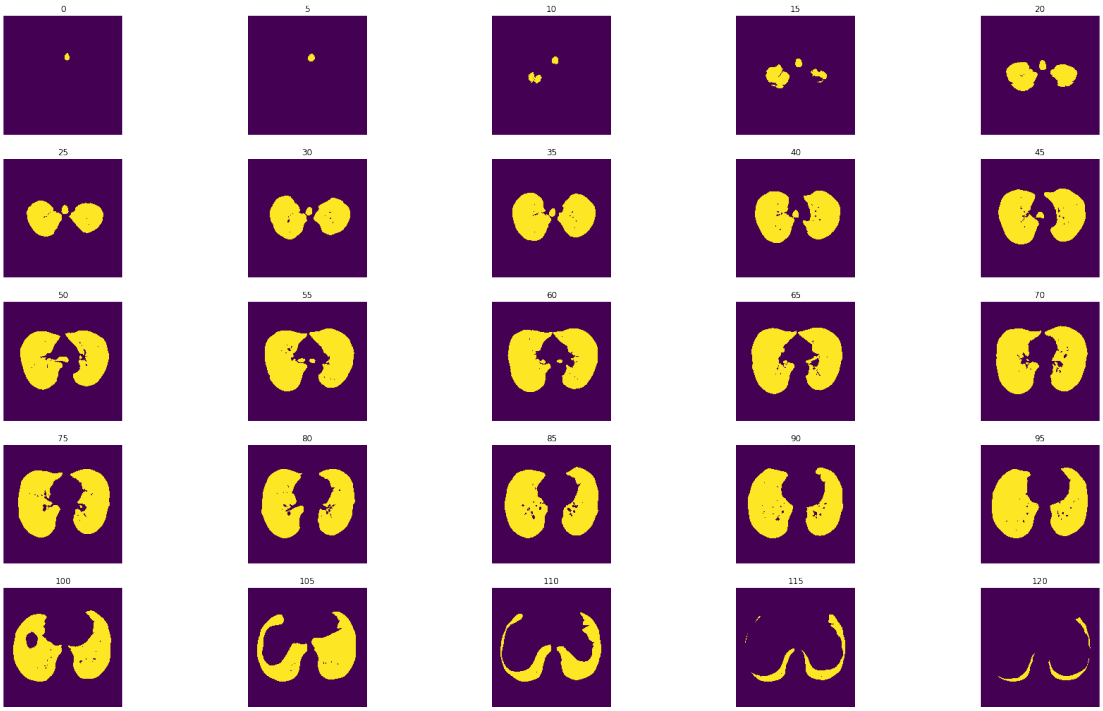
10.0.1 Deliverable:

Implement the steps described here and display for us an ordered sample of the cuts from your boolean CT scan volume that has been smoothed, with only the pixels of the lung and trachea as 1/true, and everything else as 0/False, like this:

```
In [10]: ### WRITE CODE IN HERE. You can have up to 5 cells for this question, but only one is

         from skimage import measure
         from skimage.morphology import binary_closing
         from collections import Counter
         plt.rc('figure', figsize=(30,18))

         def get_segmented(volBW, min_size=None, printCounter=False, smoothing=True):
             # Label components
             volLabel = measure.label(volBW)
             if printCounter: # for debugging
                 print(Counter(volLabel.flatten()).most_common()[:30])

             # Delete edge components
             edge_labels  = set(volLabel[[0,-1], :, :].flatten()) | set(volLabel[:, [0,-1], :]
             edge_mask = np.isin(volLabel, list(edge_labels))  # NOTE: 2nd param of isin() can
             volBW_edge_removed = np.logical_and(volBW, np.logical_not(edge_mask))

             # Delete small components
             DV=100000 # largest labels: {0: 16307700, 1: 13228194, 2: 5590203, 63: 245, 86: 6
             V= min_size if min_size else DV
             small_labels = [lab for (lab, cnt) in Counter(volLabel.flatten()).most_common() i
             small_mask = np.isin(volLabel, small_labels)
             volBW_small_removed = np.logical_and(volBW_edge_removed, np.logical_not(small_mas

             # smoothing
             return binary_closing(volBW_small_removed) if smoothing else volBW_small_removed

         volBW = vol < thresh
         volBW_smoothed = get_segmented(volBW)

         # E.g. show 5x5 slices, stepping 5
         def show_slices(vol, numrow=5, numcol=5, title=None):
```

```
        plt.rc('figure', figsize=(30,18))
        step=vol.shape[-1]//(numcol*numrow)
        print('numer of total slices: ', vol.shape[-1])
        _, axes = plt.subplots(numrow, numcol)
        if title:
            plt.suptitle(title, fontsize=30)
        for i in range(numrow * numcol):
            axes.flat[i].imshow(vol[:,:,i*step])
            axes.flat[i].set_axis_off()
            axes.flat[i].set_title(i*step)
        plt.show()

    show_slices(volBW_smoothed)


    ##########################################################################################
```

```
numer of total slices:  133
```



11 Step 7 (5 points each, 15 points total)

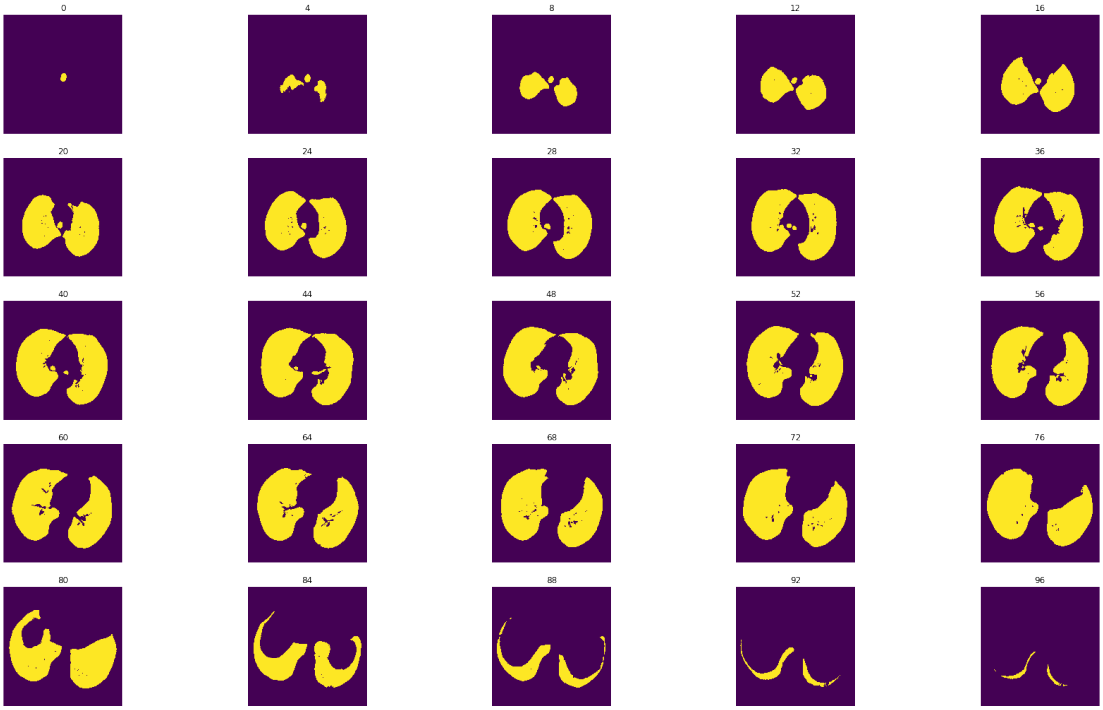
Show that the segmentation works on 3 more scans.

11.1 Deliverable:

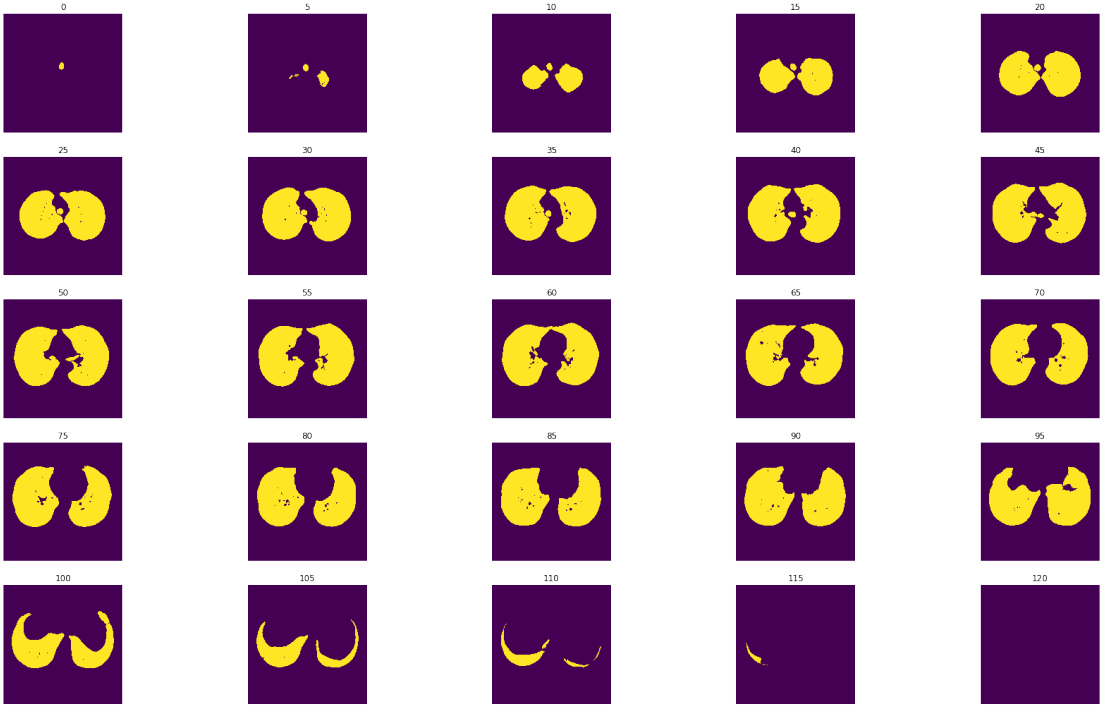
Same as Step 6, but with different scans.

```
In [11]: ### WRITE CODE IN HERE. #######

         scans_path = "/Users/neoliu/Documents/bmi260/Problem Set 1/CT_chest_scans/"
         list_of_scans = os.listdir(scans_path)
         list_of_scans = [x for x in list_of_scans if not x.startswith(".DS_Store")]

         def get_segmented_from_scan(scan_path, thresh=None, min_size=None, min_houns=None, pri
             volCT = get_hounsfield_volume(scan_path)
             vol_norm = normalize(volCT)

             th = thresh if thresh else threshold_otsu(vol_norm)
             volBW = vol_norm < th

             if min_houns:
                 volBW &= (volCT >= min_houns)

             return get_segmented(volBW, min_size, printCounter, smoothing)

         def get_slices(scan_path):
             slices = os.listdir(scan_path)
             slices = [x for x in slices if x.endswith(".dcm")]
             num_slices = len(slices)
             return slices, num_slices


         # Show the segmented CT scan in 25 slices
         def show_scan(scan_path):
             vol_segmented = get_segmented_from_scan(scan_path)
             show_slices(vol_segmented)

         # Show 3 scans
         cnt = 0
         for scan in list_of_scans:
             print ('Scan:', scan)
             scan_path = os.path.join(scans_path, scan)
             show_scan(scan_path)
             cnt += 1
             if cnt >= 3: break


         ##########################################################################################
```

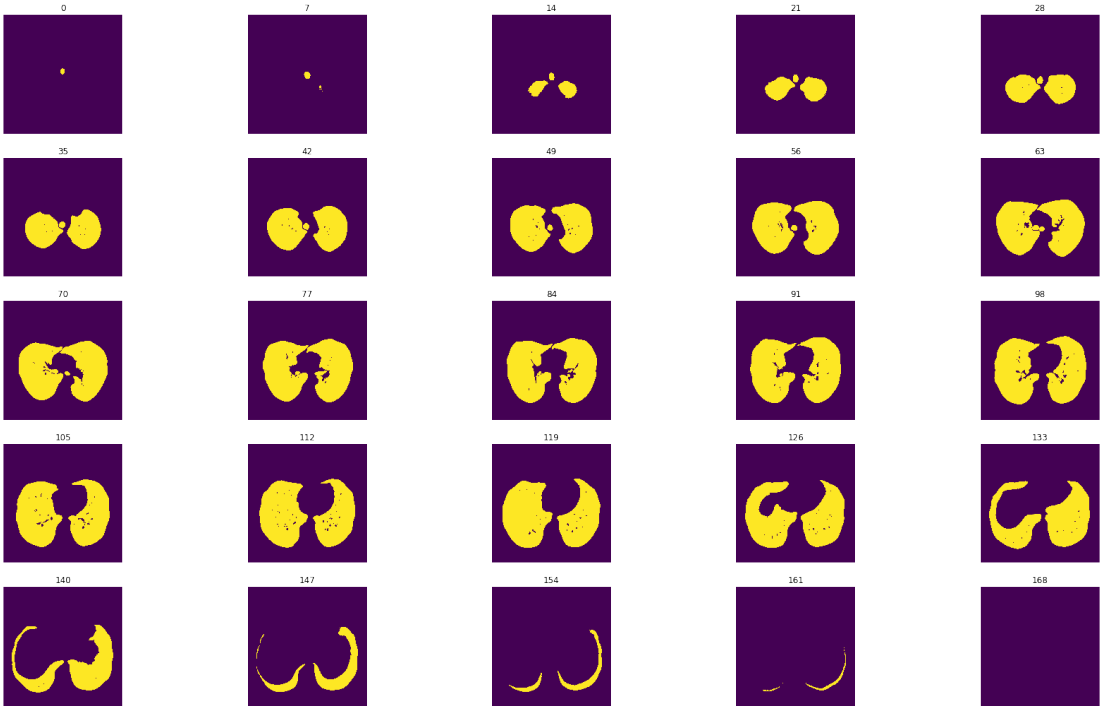
```
Scan: 0a38e7597ca26f9374f8ea2770ba870d
numer of total slices:  110
```




```
Scan: 00cba091fa4ad62cc3200a657aeb957e
numer of total slices:  134
```



```
Scan: 0b20184e0cd497028bdd155d9fb42dc9
numer of total slices:  196
```



12 Step 8 (Bonus 15 points, 5 points each)

If you've finished everything above, you've already done a full fledged segmentation. However, for an extra challenge, you can try your hand at:

1. Removing just the trachea so that your segmentation is just on the left and right lung without any of the trachea or bronchi.
2. Separating out the two halves of the lung into left/right lungs.
3. There are a ton of python packages that help you visualize the lung as a whole, but it might be a difficult to grasp. We highly recommend looking up some marching cube algorithms, but the easier way to go would probably use something like Fiji. Simply download Fiji (available for Windows, Mac, and Linux). You can save all your 3D segmentation as binary image slices. Then, using Fiji, simply read in your image sequence (be sure to name your images some numeric count or something so that Fiji knows what order to read the images in). Then, go to `Plugins -> 3D Viewer`. This should give you a figure like what you see below:

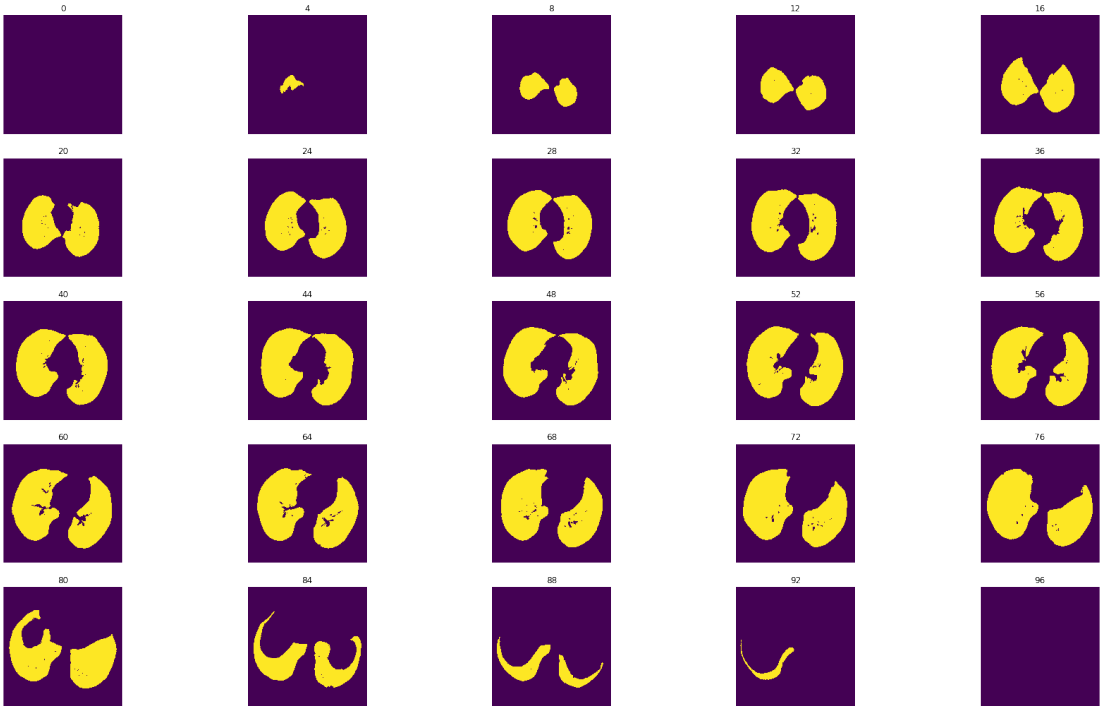
Show us that this works on at least two scans.

In [12]:
```
### WRITE CODE IN HERE. You can have up to 5 cells for this question, but only one is
import traceback

scans_path = "/Users/neoliu/Documents/bmi260/Problem Set 1/CT_chest_scans/"
list_of_scans = os.listdir(scans_path)
list_of_scans = [x for x in list_of_scans if not x.startswith(".DS_Store")]

# largest labels in sampled slices:
# [(0, 246006), (1, 8011), (3, 7492), (2, 634), (4, 1)]
# [(0, 243370), (1, 9505), (3, 8679), (2, 590)]
# [(0, 240754), (1, 10887), (4, 9927), (3, 575), (2, 1)]
threshold_trachea = 3000
def remove_trachea_by_slide_size(vol_bw):
    vol = vol_bw.copy()
    for i in range(vol.shape[-1]):
        img = vol[:, :, i]
        labels = measure.label(img)
        smalls = [lab for (lab, cnt) in Counter(labels.flatten()).most_common() if cnt
        vol[:, :, i] = np.where(np.isin(labels, smalls), 0, img)
    return vol

def separate_lungs(vol_bw):
    labels = measure.label(vol_bw, background=0)
    lung1, lung2 = [lab for (lab, cnt) in (Counter(labels.flatten()).most_common()[:3]
    y1s = np.where(labels == lung1)[1]
    y2s = np.where(labels == lung2)[1]
    if y1s.mean() > y2s.mean():
        left = lung1
        right = lung2
    else:
        left = lung2
        right = lung1
    return labels == left, labels == right

N = 2
vols_bw = [0] * N
vols_no_trachea = [0] * N
vols_left_lungs = [0] * N
vols_right_lungs = [0] * N
i = 0
tries = 1
for scan in list_of_scans:
    print('Scan#{}: {}'.format(tries, scan))
    scan_path = os.path.join(scans_path, scan)
    try:
        # Remove trachea without smoothing, for better separation results
```

```
                vols_bw[i] = get_segmented_from_scan(scan_path, smoothing=False)
                vols_no_trachea[i] = remove_trachea_by_slide_size(vols_bw[i])

                # Seprate left/right lungs
                vols_left_lungs[i], vols_right_lungs[i] = separate_lungs(vols_no_trachea[i])

                # Now do smoothing
                vols_no_trachea[i] = binary_closing(vols_no_trachea[i])
                vols_left_lungs[i] = binary_closing(vols_left_lungs[i])
                vols_right_lungs[i] = binary_closing(vols_right_lungs[i])

                show_slices(vols_no_trachea[i], title='scan#{} trachea removed'.format(tries))
                show_slices(vols_left_lungs[i], title='scan#{} left lung'.format(tries))
                show_slices(vols_right_lungs[i], title='scan#{} right lung'.format(tries))
                i += 1
            except ValueError as e:
                print('Skipping scan#{} due to left/right lungs connected'.format(tries))
                # print(traceback.format_exc())
                pass
            tries += 1
            if i >= N: break

        ##########################################################################################
```

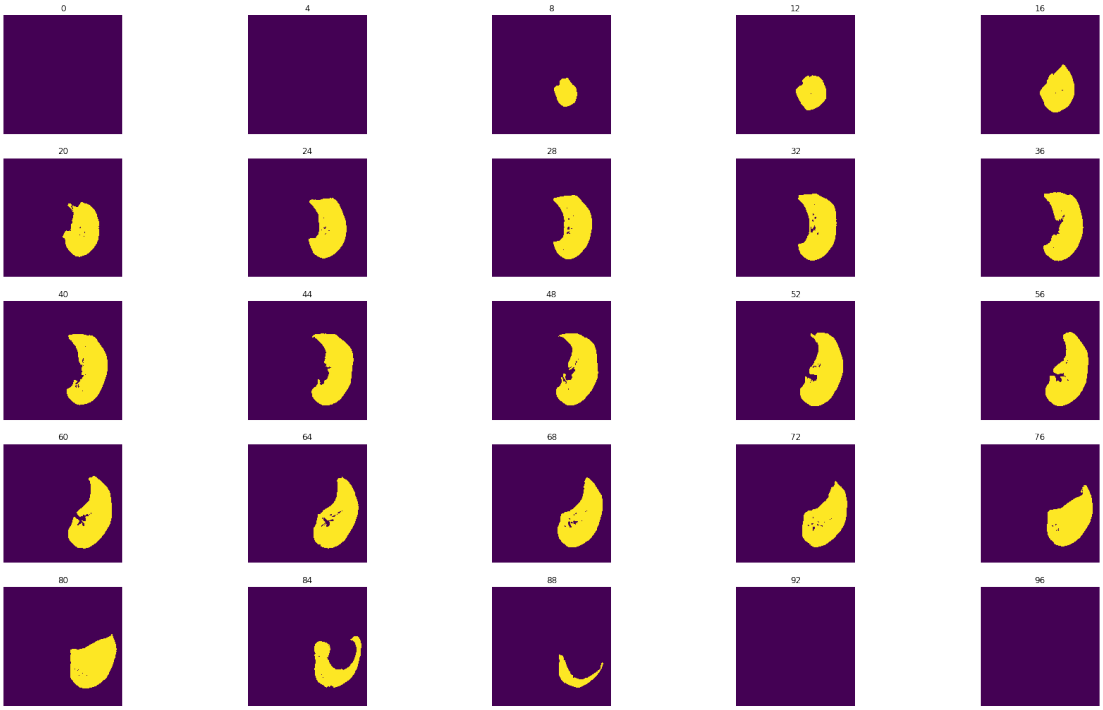
```
Scan#1: 0a38e7597ca26f9374f8ea2770ba870d
numer of total slices:  110
```

scan#1 trachea removed



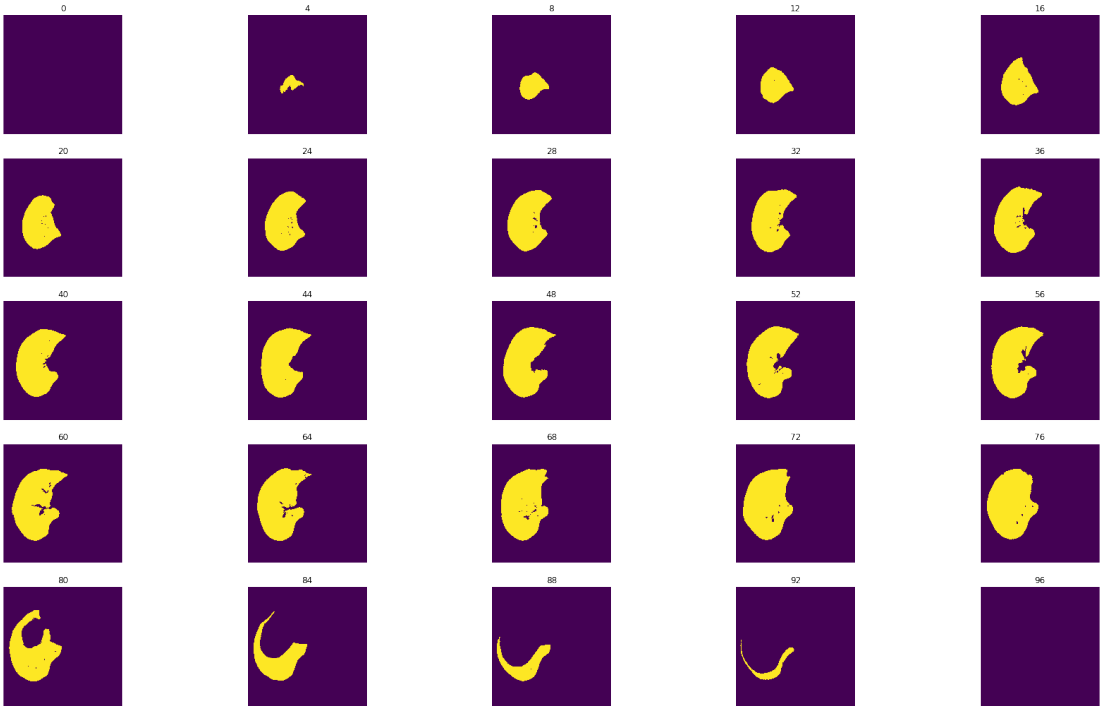

```
numer of total slices:  110
```

scan#1 left lung



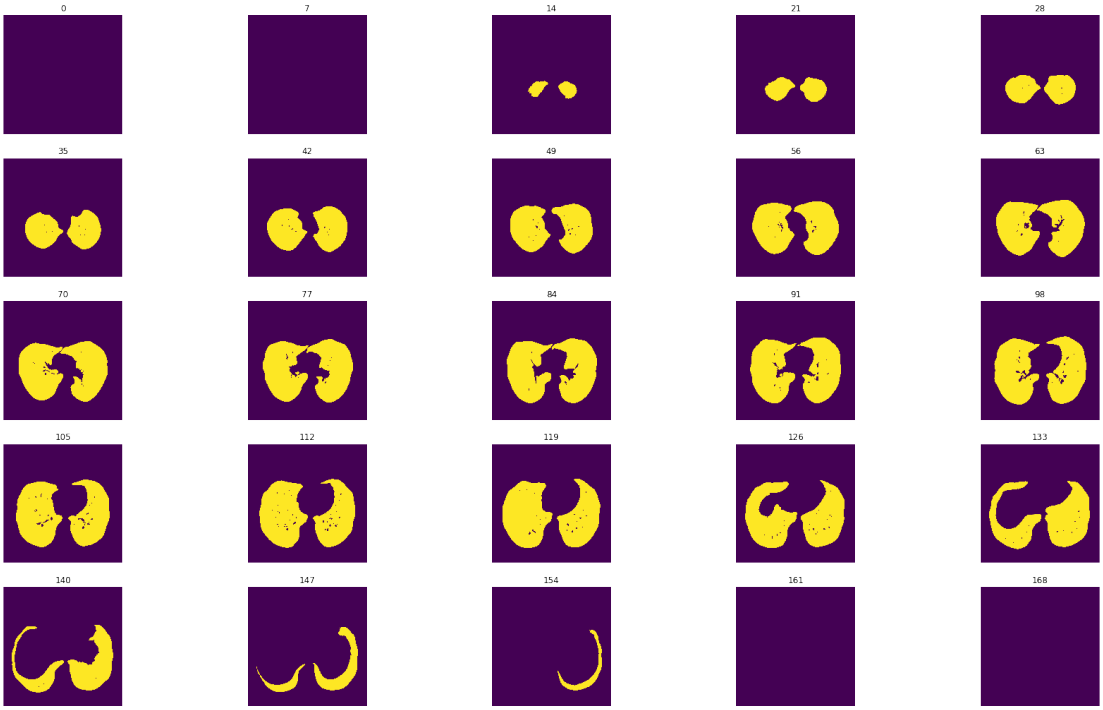

```
numer of total slices:  110
```

scan#1 right lung



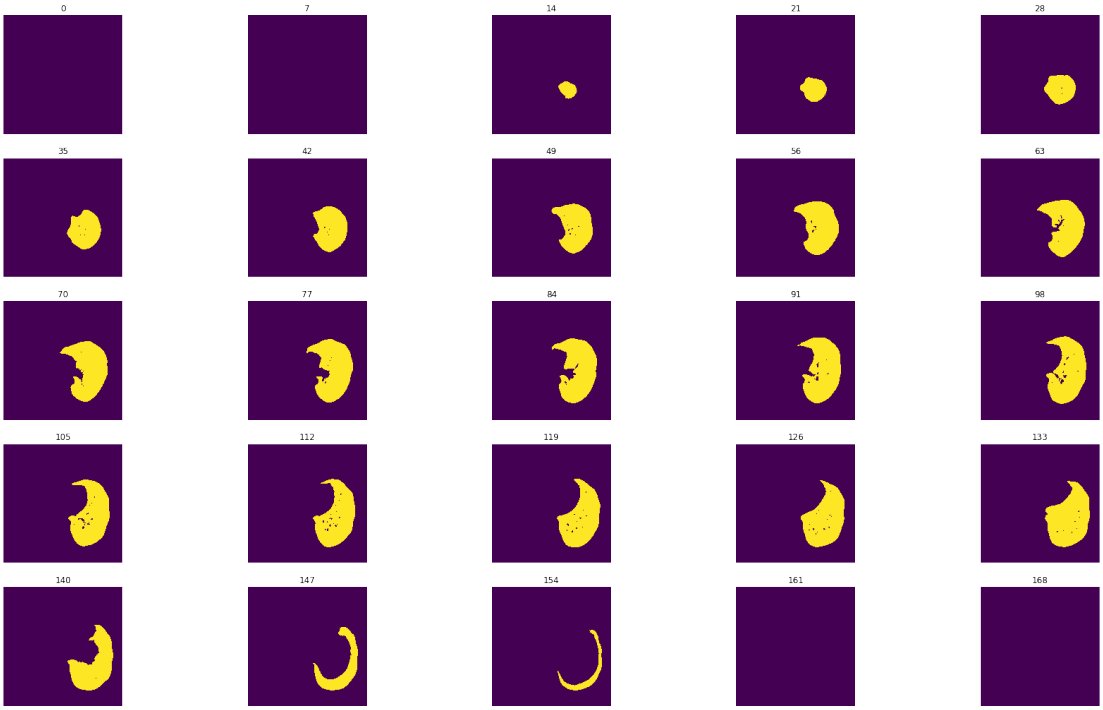

```
Scan#2: 00cba091fa4ad62cc3200a657aeb957e
Skipping scan#2 due to left/right lungs connected
Scan#3: 0b20184e0cd497028bdd155d9fb42dc9
numer of total slices:  196
```

scan#3 trachea removed



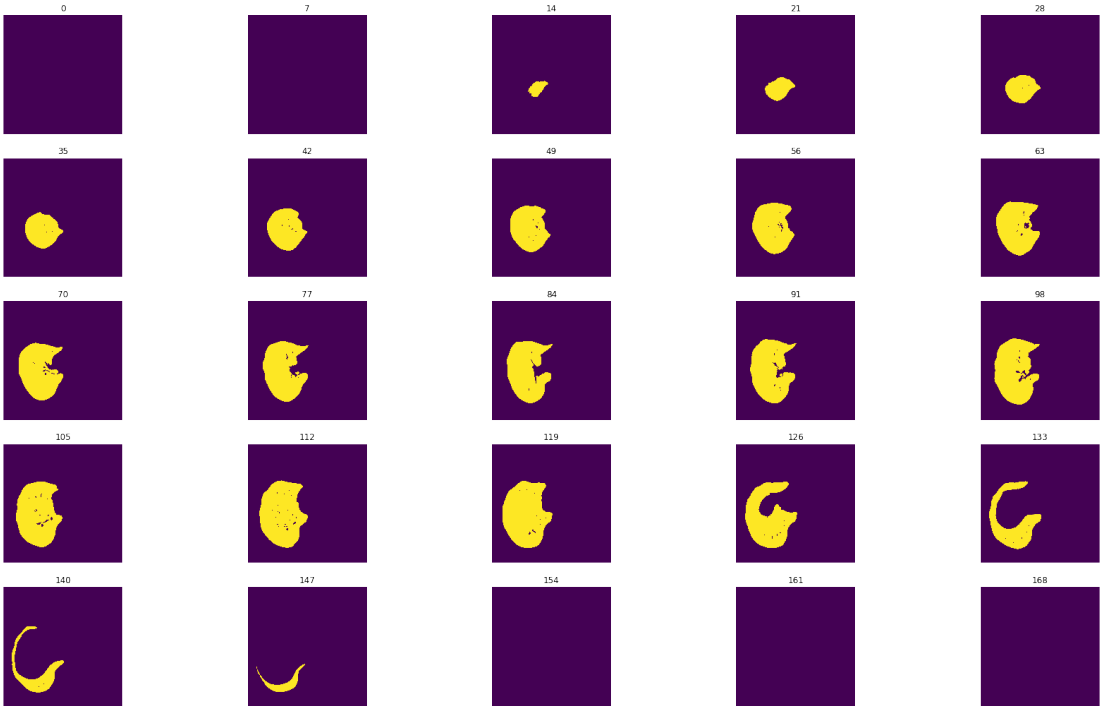

```
numer of total slices:  196
```

scan#3 left lung



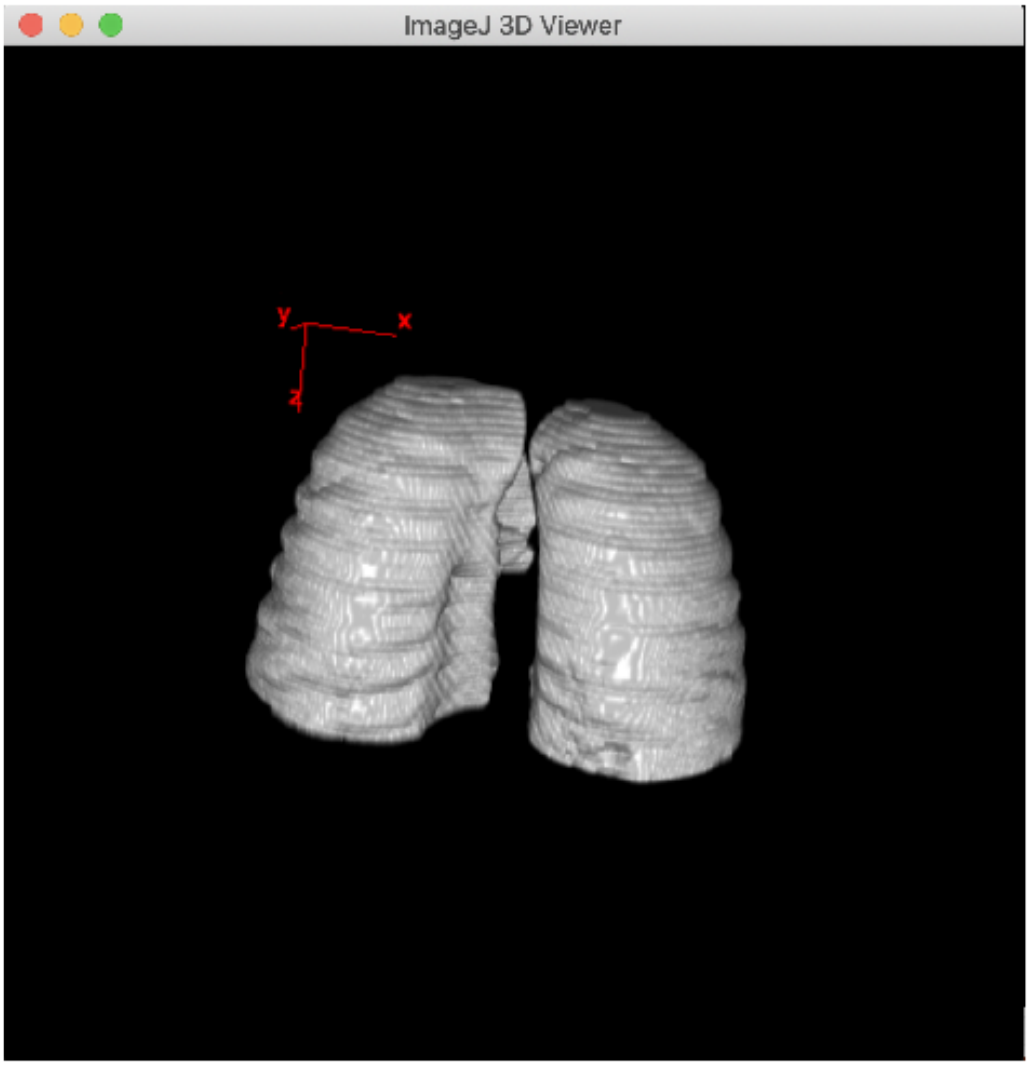

```
numer of total slices:  196
```

scan#3 right lung



In [13]:
```
### WRITE CODE IN HERE. You can have up to 5 cells for this question, but only one is

root_dir = '/Users/neoliu/tmp'

import cv2
 # float32
for i in range(N):
    for j in range(vols_no_trachea[i].shape[-1]):
        os.makedirs(os.path.join(root_dir, 'scan_'+str(i)), exist_ok=True)
        cv2.imwrite(os.path.join(root_dir, 'scan_'+str(i), 'slice_'+str(j)+'.png'),
                    vols_no_trachea[i][:,:,j].astype('uint8') * 255)

##########################################################################################
```

In [17]:
```
for i in range(N):
    image = plt.imread('scan_{}.png'.format(i))
    plt.rc('figure', figsize=(15,9))
    fig, ax = plt.subplots()
    ax.imshow(image)
    ax.axis('off')  # clear x-axis and y-axis
    plt.show()
```


ImageJ 3D Viewer
y
x
z

12.1 References

Correcting Non-Uniform Illumination - Rice Image Segmentation - Brain Image

We thank Darvin Yi for the content.