BIOMEDIN 260/RAD260: Problem Set 1 - Segmentation

Spring 2024

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:

Xianghao Zhan

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.

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

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 [59]:

```
cd "H:\My Drive\Stanford ACA\BIOMEDIN 260"
```

```
H:\My Drive\Stanford ACA\BIOMEDIN 260
```

In [3]:

```
!pip install pydicom
```

```
Collecting pydicom
  Downloading pydicom-2.4.4-py3-none-any.whl (1.8 MB)
Installing collected packages: pydicom
Successfully installed pydicom-2.4.4
```

In [60]:

```
# Some packaged you might need, you can add more if you need them
import os
import numpy as np
import pydicom as dicom
import matplotlib.pyplot as plt
import os
import skimage
%matplotlib inline
```

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 [ ]:

```
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install numpy
# or simply
! pip install numpy
```

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 on Canvas. We highly recommend downloading the " `CT_chest_scans.zip` " file on Canvas 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:

dicom_fileds

In [68]:

```
# each chest CT scan is a folder with a long weird name like 0acbebb8d463b4b9ca88cf3843
# many .dcm files with similar long weird names, assign the path to such folder to the

scans_path = "H:\My Drive\Stanford ACA\BIOMEDIN 260\CT_chest_scans\CT_chest_scans" # TO
list_of_scans = os.listdir(scans_path)
list_of_scans.sort()

# for figuring out the controls lets experiment with slice 20 of scan 5
scan_num = 5
print(f'Patient name: {list_of_scans[scan_num]}')
scan_path = os.path.join(scans_path, list_of_scans[scan_num])
list_of_slices = os.listdir(scan_path)
slice_num = 20
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)
```

```
Patient name: 0bd0e3056cbf23a1cb7f0f0b18446068
Dataset.file_meta -------------------------------
(0002, 0000) File Meta Information Group Length  UL: 194
(0002, 0001) File Meta Information Version       OB: b'\x00\x01'
(0002, 0002) Media Storage SOP Class UID         UI: CT Image Storage
(0002, 0003) Media Storage SOP Instance UID      UI: 1.3.6.1.4.1.14519.5.2.1.7009.9004.2
094109336326196794579817O8301
(0002, 0010) Transfer Syntax UID                 UI: Implicit VR Little Endian
(0002, 0012) Implementation Class UID            UI: 1.2.40.0.13.1.1.1
(0002, 0013) Implementation Version Name         SH: 'dcm4che-1.4.31'
-------------------------------------------------
(0008, 0000) Group Length                        UL: 362
(0008, 0005) Specific Character Set              CS: 'ISO_IR 100'
(0008, 0016) SOP Class UID                       UI: CT Image Storage
(0008, 0018) SOP Instance UID                    UI: 1.3.6.1.4.1.14519.5.2.1.7009.9004.2
09410933632619679457981708301
(0008, 0060) Modality                            CS: 'CT'
(0008, 103e) Series Description                  LO: 'Axial'
(0010, 0000) Group Length                        UL: 60
(0010, 0010) Patient's Name                      PN: '0bd0e3056cbf23a1cb7f0f0b18446068'
(0010, 0020) Patient ID                          LO: '0bd0e3056cbf23a1cb7f0f0b18446068'
(0010, 0030) Patient's Birth Date                DA: '19000101'
(0018, 0060) KVP                                 DS: None
(0020, 0000) Group Length                        UL: 392
(0020, 000d) Study Instance UID                  UI: 2.25.17062309086576606706587560884б
81601601915389122225987441686
(0020, 000e) Series Instance UID                 UI: 2.25.506676188549638900454958838319
90802854777872473543131775273
(0020, 0011) Series Number                       IS: '2'
(0020, 0012) Acquisition Number                  IS: '1'
(0020, 0013) Instance Number                     IS: '215'
(0020, 0032) Image Position (Patient)            DS: [-170.000000, -163.800003, -268.945
007]
(0020, 0037) Image Orientation (Patient)         DS: [1.000000, 0.000000, 0.000000, 0.00
0000, 1.000000, 0.000000]
(0020, 0052) Frame of Reference UID              UI: 2.25.908016948574605987334734044492
60069400553016836151520342108
(0020, 1040) Position Reference Indicator        LO: 'SN'
```

```
(0020, 1041) Slice Location                      DS: '-268.945007'
(0028, 0000) Group Length                        UL: 198
(0028, 0002) Samples per Pixel                   US: 1
(0028, 0004) Photometric Interpretation          CS: 'MONOCHROME2'
(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: 16
(0028, 0102) High Bit                            US: 15
(0028, 0103) Pixel Representation                US: 1
(0028, 0120) Pixel Padding Value                 SS: -2000
(0028, 0301) Burned In Annotation                CS: 'NO'
(0028, 0303) Longitudinal Temporal Information M CS: 'MODIFIED'
(0028, 1050) Window Center                       DS: '40.0'
(0028, 1051) Window Width                        DS: '400.0'
(0028, 1052) Rescale Intercept                   DS: '-1024.0'
(0028, 1053) Rescale Slope                       DS: '1.0'
(7fe0, 0010) Pixel Data                          OW: Array of 524288 elements
```

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

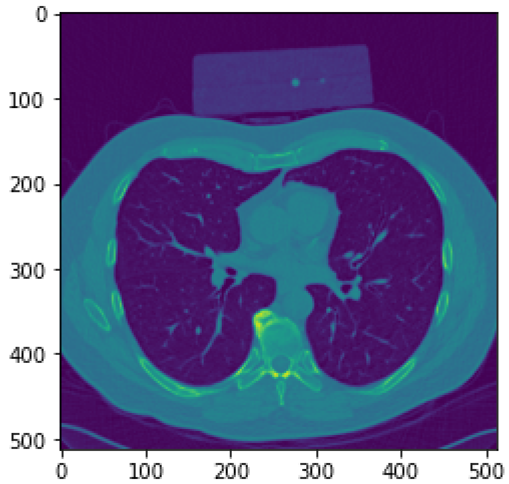
We can see that there are many data fields in a DICOM file. There's a lot of patient information stored in a DICOM - name, birthdate, and so on. For obvious privacy reasons, this data has been completely anonymized. It’s pretty evident that our data probably didn’t come from Mr./Mrs. 0acbebb8d463b4b9ca88cf38431aac69 who was born on January 1st, 1900.

Displaying images is fundamental to working in this field.

Run the following cell to view the image associated with this very old person. Notice the data type.

In [6]:

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



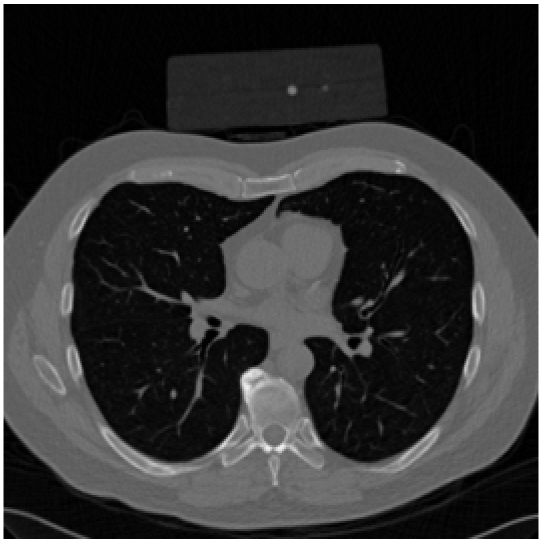
```
<class 'numpy.ndarray'> 466.1270065307617 uint16
```

Why is the image blue and green if CT slices are grayscale images?

Previously we defined a `viridis` colormap for plotting. Let's change the colormap to one that matches our perception better and remove the pixel coordinate numbers for a cleaner visualization.

In [7]:

```
rawimg = ds.pixel_array
plt.imshow(rawimg, cmap='gray')
plt.axis('off')
plt.tight_layout()
plt.show()
print(type(rawimg), np.mean(ds.pixel_array), rawimg.dtype)
```



```
<class 'numpy.ndarray'> 466.1270065307617 uint16
```

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

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).

![segmentation0](figures/segmentation0.jpg) (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. ![segmentation1](figures/segmentation1.png) (b) Multilabel Segmentation - A multilabel segmentation from a magnetic resonance image of the brain. We can see that different colors represent different sections 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:

lung_field_segmentation

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:

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

houndsfield

Deliverable:

1) Read in the raw data for CT slice 120 for the patient above (`0acbebb8d463b4b9ca88cf38431aac69`) 2) Convert the raw pixel values into Hounsfield units using the formula above. Hint: use the metadata in the DICOM file. 3) Compute the max, min, mean, and standard deviation of both images (raw data and Hounsfield units) 4) Compare them the summary statistics of both images.

In [23]:

```
### WRITE CODE IN HERE. You can have up to 2 cells for this question, but only one is r

# Specify the patient ID and CT slice No. 120
scans_path = "./CT_chest_scans/CT_chest_scans" # TODO: Change this to match where your
list_of_scans = os.listdir(scans_path)
list_of_scans.sort()
scan_path = os.path.join(scans_path, '0acbebb8d463b4b9ca88cf38431aac69')
list_of_slices = os.listdir(scan_path)
slice_num = 120
slice_path = os.path.join(scan_path, list_of_slices[slice_num])

# read in the full path to the file
ds2 = dicom.read_file(slice_path) # you may have to use pydicom instead of dicom
print(ds2)

# Convert the raw pixel values into Hounsfield Units
rawimg = ds2.pixel_array
HU = rawimg * ds2.RescaleSlope + ds2.RescaleIntercept

# Compute the summary statistics
print('Raw Image Max: ', np.max(rawimg), "; Hounsfield Image Max: ", np.max(HU))
print('Raw Image Min: ', np.min(rawimg), "; Hounsfield Image Min: ", np.min(HU))
print('Raw Image Mean: ', np.mean(rawimg), "; Hounsfield Image Mean: ", np.mean(HU))
print('Raw Image Standard Deviation: ', np.std(rawimg), "; Hounsfield Image Standard De

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

```
Dataset.file_meta -------------------------------
(0002, 0000) File Meta Information Group Length  UL: 196
(0002, 0001) File Meta Information Version       OB: b'\x00\x01'
(0002, 0002) Media Storage SOP Class UID         UI: CT Image Storage
(0002, 0003) Media Storage SOP Instance UID      UI: 1.3.6.1.4.1.14519.5.2.1.7009.9004.1
428947661455701615218355567958
(0002, 0010) Transfer Syntax UID                 UI: Explicit VR Little Endian
(0002, 0012) Implementation Class UID            UI: 1.2.40.0.13.1.1.1
(0002, 0013) Implementation Version Name         SH: 'dcm4che-1.4.31'
-------------------------------------------------
(0008, 0005) Specific Character Set              CS: 'ISO_IR 100'
(0008, 0016) SOP Class UID                       UI: CT Image Storage
(0008, 0018) SOP Instance UID                    UI: 1.3.6.1.4.1.14519.5.2.1.7009.9004.1
428947661455701615218355567958
(0008, 0060) Modality                            CS: 'CT'
(0008, 103e) Series Description                  LO: 'Axial'
(0010, 0010) Patient's Name                      PN: '0acbebb8d463b4b9ca88cf38431aac69'
(0010, 0020) Patient ID                          LO: '0acbebb8d463b4b9ca88cf38431aac69'
(0010, 0030) Patient's Birth Date                DA: '19000101'
(0018, 0060) KVP                                 DS: None
(0020, 000d) Study Instance UID                  UI: 2.25.662349949400930605308758826735
```

```
93880723182397297085825139120
(0020, 000e) Series Instance UID                 UI: 2.25.279857371301060729183105335256
88877208529713445697698517643
(0020, 0011) Series Number                       IS: '4'
(0020, 0012) Acquisition Number                  IS: '2'
(0020, 0013) Instance Number                     IS: '195'
(0020, 0032) Image Position (Patient)            DS: [-157.2275390625, -280.2275390625,
-368.1]
(0020, 0037) Image Orientation (Patient)         DS: [1, 0, 0, 0, 1, 0]
(0020, 0052) Frame of Reference UID              UI: 2.25.110240475673229366733716029523
41875417234268835153580015625
(0020, 1040) Position Reference Indicator        LO: ''
(0020, 1041) Slice Location                      DS: '-368.1'
(0028, 0002) Samples per Pixel                   US: 1
(0028, 0004) Photometric Interpretation          CS: 'MONOCHROME2'
(0028, 0010) Rows                                US: 512
(0028, 0011) Columns                             US: 512
(0028, 0030) Pixel Spacing                       DS: [0.544921875, 0.544921875]
(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, 0106) Smallest Image Pixel Value          US: 0
(0028, 0107) Largest Image Pixel Value           US: 3067
(0028, 0301) Burned In Annotation                CS: 'NO'
(0028, 0303) Longitudinal Temporal Information M SH: 'MODIFIED'
(0028, 1050) Window Center                       DS: [-400, 0]
(0028, 1051) Window Width                        DS: [1600, 2]
(0028, 1052) Rescale Intercept                   DS: '-1024.0'
(0028, 1053) Rescale Slope                       DS: '1.0'
(0028, 1055) Window Center & Width Explanation   LO: ['WINDOW1', 'WINDOW2']
(7fe0, 0010) Pixel Data                          OW: Array of 524288 elements
 Raw Image Max:  3067 ; Hounsfield Image Max:  2043.0
 Raw Image Min:  0 ; Hounsfield Image Min:  -1024.0
 Raw Image Mean:  583.3522491455078 ; Hounsfield Image Mean:  -440.6477508544922
 Raw Image Standard Deviation:  512.9605377564801 ; Hounsfield Image Standard Deviation:
 512.9605377564801
```

Comparison between the images: Raw Image Max: 3067 ; Hounsfield Image Max: 2043.0 Raw Image Min: 0 ; Hounsfield Image Min: -1024.0 Raw Image Mean: 583.3522491455078 ; Hounsfield Image Mean: -440.6477508544922 Raw Image Standard Deviation: 512.9605377564801 ; Hounsfield Image Standard Deviation: 512.9605377564801

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:

pseudocode

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

Deliverable:

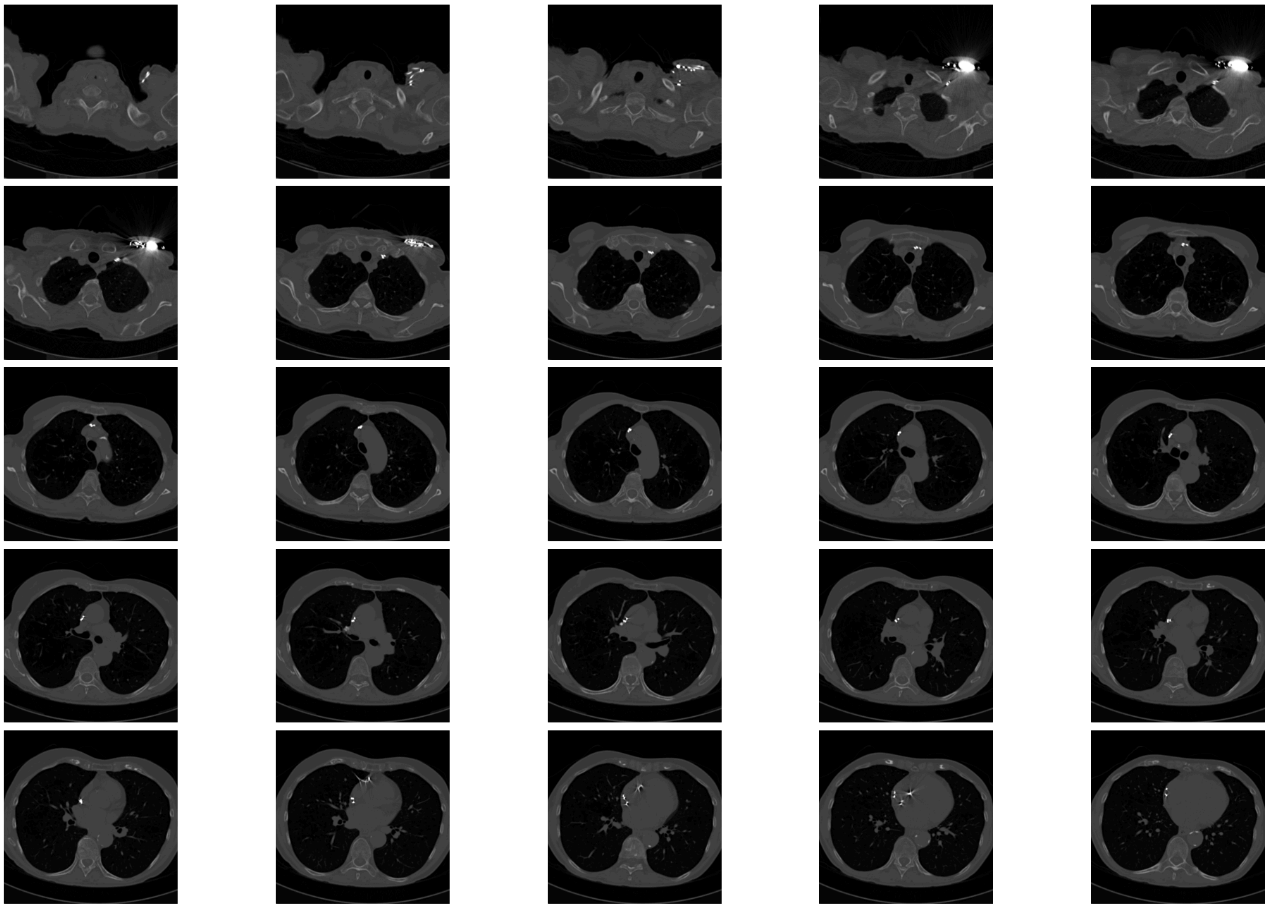
Organize the 3D volume into an array and display 25 of the slices in correct order for us, like this: lungs

In [24]:

```
### WRITE CODE IN HERE. You can have up to 4 cells for this question, but only one is re

num_slices = len(os.listdir(scan_path))
volCT = np.zeros((ds2.pixel_array.shape[0], ds2.pixel_array.shape[1], num_slices)) #Ini
for idx in range(num_slices):
    slice_path = os.path.join(scan_path, list_of_slices[idx])
    ds = dicom.read_file(slice_path)
    volCT[:, :, ds.InstanceNumber-1] = ds.pixel_array
#######################################################################################
```

In [25]:

```
###### Display the organized CT volume ######
fig, ax = plt.subplots(5, 5)
for i in range(5):
    for j in range(5):
        ax[i, j].imshow(volCT[:, :, (5*i+j)*5], cmap='gray')
        ax[i, j].axis("off")
        plt.tight_layout()
        plt.rcParams['figure.figsize'] = (30, 18)
```



Step 3 (5 points)

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

Deliverable:

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

Answer: uint16

2) 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.

In [26]:

```
### WRITE CODE IN HERE. Your code to answer to 2) #######
print('Before Conversion: ')
print('Max: ', np.max(volCT))
print('Min: ', np.min(volCT))
print('Data Type: ', volCT.dtype)

#Perform the conversion
volCT_converted = (volCT - np.min(volCT))/(np.max(volCT) - np.min(volCT))
volCT_converted = volCT_converted.astype('float32')
print('After Conversion: ')
print('Max: ', np.max(volCT_converted))
print('Min: ', np.min(volCT_converted))
print('Data Type: ', volCT_converted.dtype)
###########################################################################################
```

```
Before Conversion:
Max:  4095.0
Min:  0.0
Data Type:  float64
After Conversion:
Max:  1.0
Min:  0.0
Data Type:  float32
```

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.

histWithThresh

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.

Deliverable:

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

Answer: The material inside human body are generally categorized into two types with distinct X-ray absorption properties. The lower group indicates the air, water and soft tissue. The higher group indicates the bone, minerals and potentially metal.

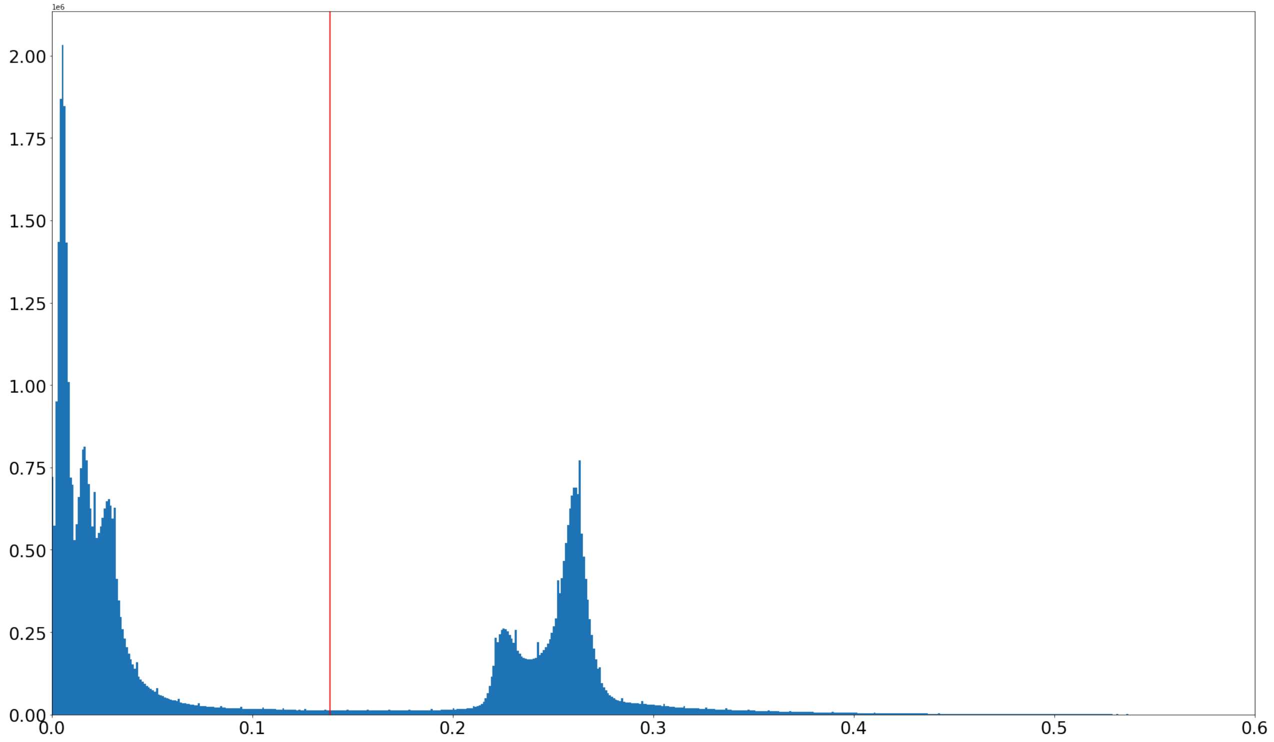
2) Find the Otsu's threshold in your volume of pixels and plot the histogram of of your pixels with this line like the example figure. You will find these links (1, 2, and 3) useful if you have not done this before.

In [35]:

```
### WRITE CODE IN HERE. You can have up to 2 cells for this question, but only one is re
from skimage.filters import threshold_otsu

otsu_threshold = threshold_otsu(volCT_converted)
plt.hist(np.ravel(volCT_converted),1000)
plt.xlim([0,0.6])
plt.xticks(fontsize = 24)
plt.yticks(fontsize = 24)
plt.axvline(otsu_threshold, color='r')
###########################################################################################
```

Out[35]:

```
<matplotlib.lines.Line2D at 0x15457742b50>
```



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.

otsu

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

Deliverable:

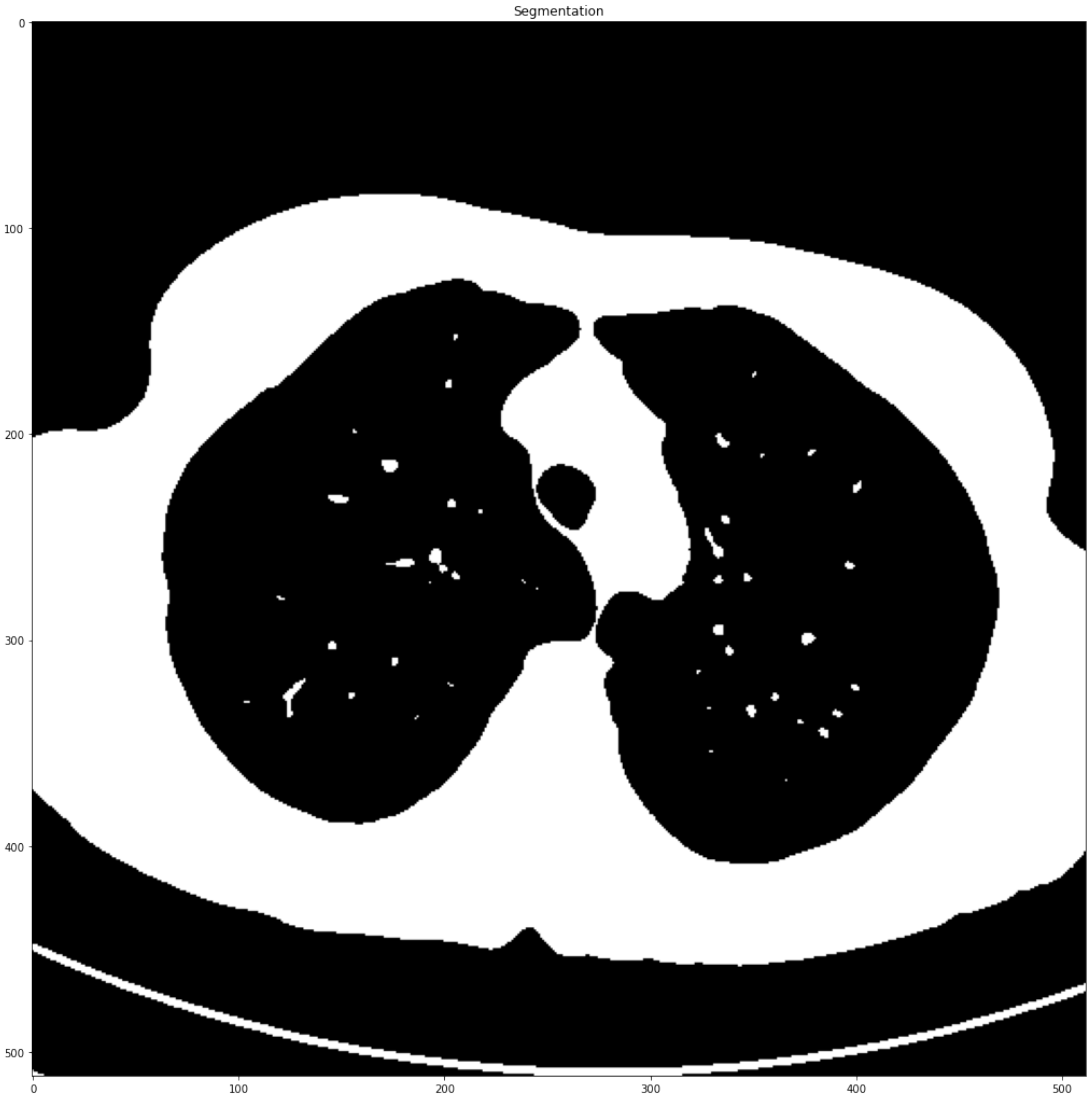
Display for us any one slice of the CT scan showing a binary image after applying thresholding using Otsu's method. Usually it's convention to show the area of interest after the segmentation in white and the background as black.

In [42]:

```
### WRITE CODE IN HERE. You can have up to 2 cells for this question, but only one is r
example_slice = volCT_converted[:, :, 50]
example_slice = example_slice > otsu_threshold
plt.imshow(example_slice, cmap=plt.cm.gray)
plt.title('Segmentation')
########################################################################################
```

Out[42]:

```
Text(0.5, 1.0, 'Segmentation')
```



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:

pseudocode2

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.

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:

final_segm

In [50]:

```
### WRITE CODE IN HERE. You can have up to 5 cells for this question, but only one is r
# Define a function to return the edge values for a connected-region-labeled image
def find_edge(img):
    '''
    This function returns the edge values of a region-labeled image.
    param: img, a numpy array that contains the distinct indicator of continuous region
    returns a set of indicator values that are touching the edge
    '''
    return set(list(img[0, :]) +  list(img[:, 0]) + list(img[-1, :]) + list(img[:, -1])
####################################################################################
```

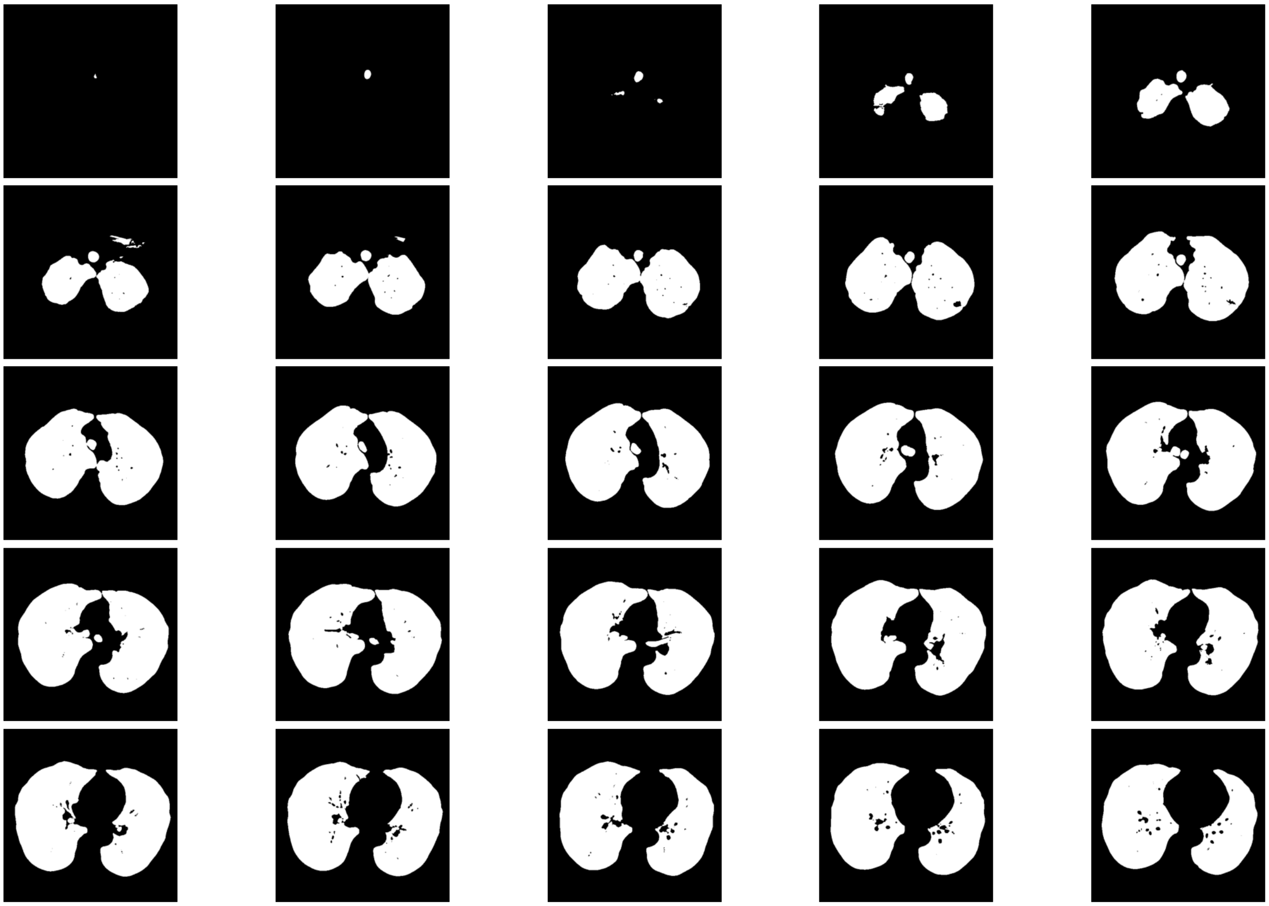
In [52]:

```
### WRITE CODE IN HERE. You can have up to 5 cells for this question, but only one is r
from scipy.ndimage import label
from skimage.morphology import binary_closing

# 1. Binarize the image
otsu_threshold = threshold_otsu(volCT_converted)
volBW = volCT_converted < otsu_threshold

# 2. Delete the connected compotnents that touch the edge using scipy.ndimage.label
for idx in range(volBW.shape[2]):
    labeled_image, num_regions = label(volBW[:,:,idx]) #Use the label function to find
    edge_pixels = find_edge(labeled_image) # Find the edge of the connection-labeled im
    for region_i in range(1, num_regions + 1): # 0 is the background, look at feature 1
        if region_i in edge_pixels: # The feature is at the edge
            mask = (labeled_image != region_i)
            masked_slice = mask * volBW[:, :, idx]
            volBW[:, :, idx] = masked_slice # Turn the labels to be background
```

```
# 3. Remove the connected regions with volume smaller than a particular volume
V = 500
volBW_finetuned = volBW
labeled_image, num_regions = label(volBW_finetuned)
for region_i in range(1, num_regions + 1):
    if np.sum(labeled_image == region_i) <= V:
        mask = (labeled_image != region_i)
        volBW_finetuned = mask * volBW_finetuned # Mask off the connected region that i

# 4. Smoothing
volBW_smoothed = binary_closing(volBW_finetuned)
```

In [53]:

```
#### plot the segmented binary lung volume ####
fig, ax = plt.subplots(5, 5)
for i in range(5):
    for j in range(5):
        ax[i, j].imshow(volBW_smoothed[:, :, (5*i+j)*5], cmap='gray')
        ax[i, j].axis("off")
        plt.tight_layout()
        plt.rcParams['figure.figsize'] = (30, 18)
```



Step 7 (5 points each, 15 points total)

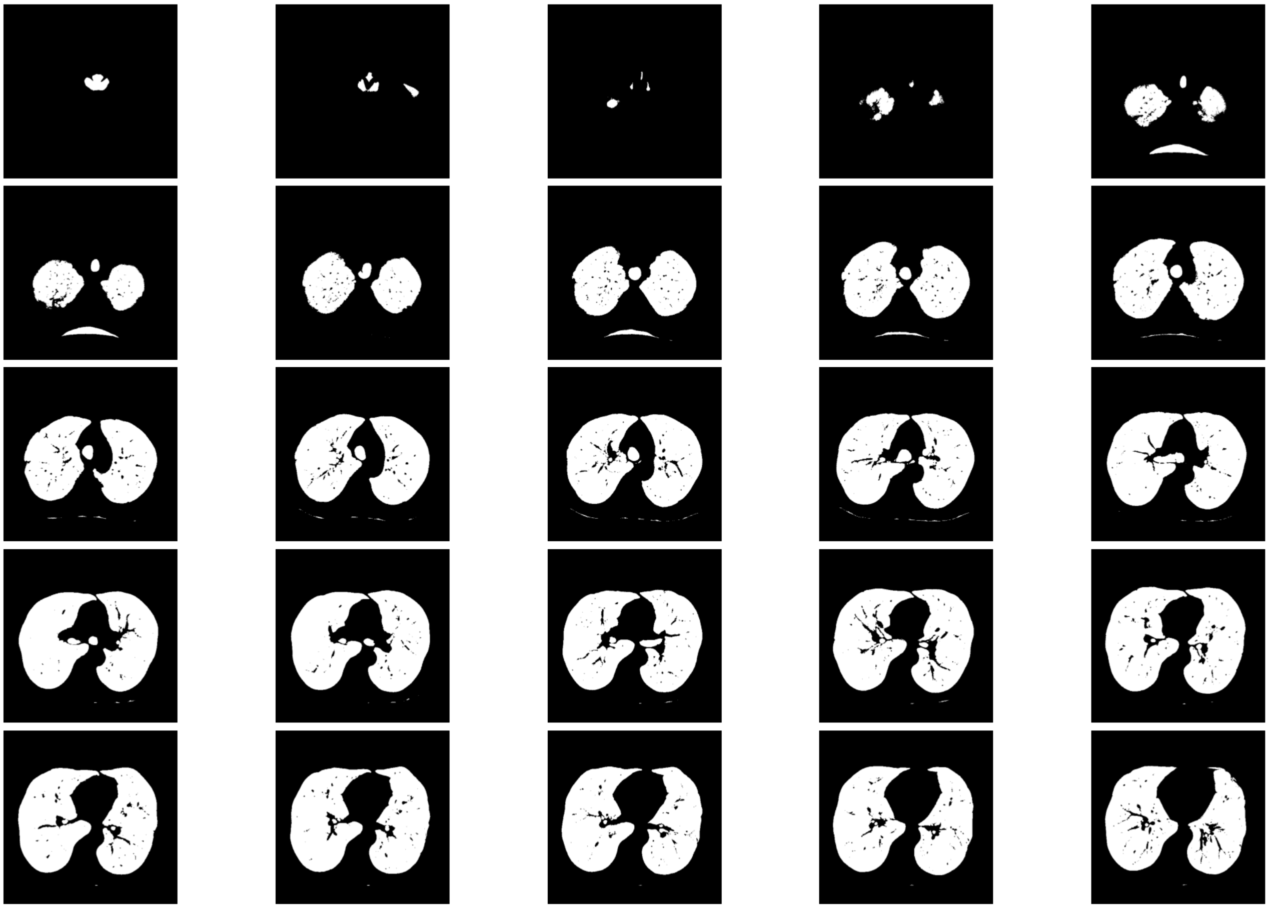
Show that the segmentation works on 3 more scans.

Deliverable:

Same as Step 6, but with different scans.

In [55]:

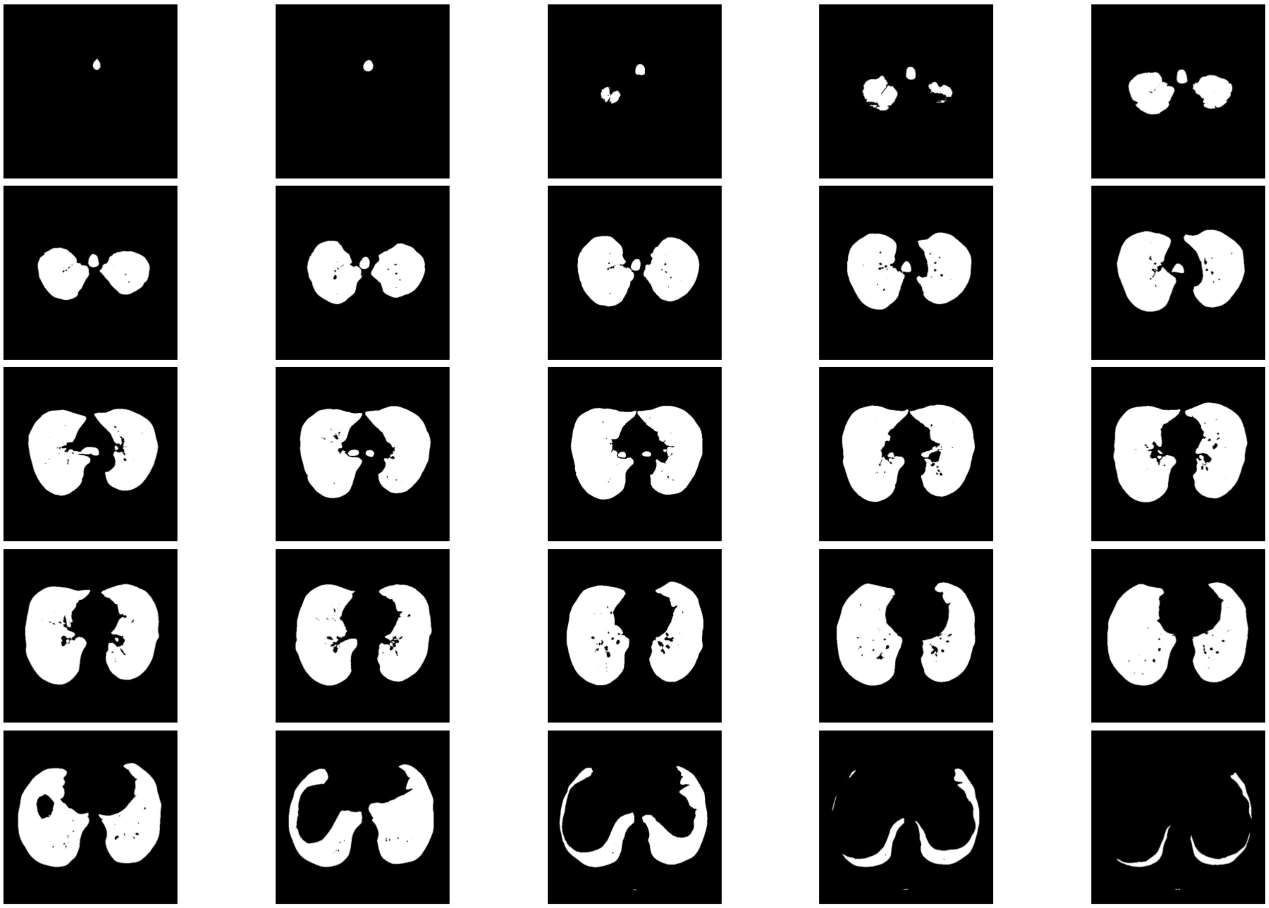
```
### WRITE CODE IN HERE. You can have up to 5 cells for this question, but only one is r
num_scan = 10

# 1. Read the scan file into volumetric CT
scans_path = "./CT_chest_scans/CT_chest_scans" # TODO: Change this to match where your 
list_of_scans = os.listdir(scans_path)
list_of_scans.sort()
scan_path = os.path.join(scans_path, list_of_scans[num_scan])
list_of_slices = os.listdir(scan_path)

num_slices = len(os.listdir(scan_path))
volCT = np.zeros((ds2.pixel_array.shape[0], ds2.pixel_array.shape[1], num_slices)) #Ini

for idx in range(num_slices):
    slice_path = os.path.join(scan_path, list_of_slices[idx])
    ds = dicom.read_file(slice_path)
    volCT[:, :, ds.InstanceNumber-1] = ds.pixel_array

#2. Perform the conversion
volCT_converted = (volCT - np.min(volCT))/(np.max(volCT) - np.min(volCT))
volCT_converted = volCT_converted.astype('float32')


# 3. Binarize the image
otsu_threshold = threshold_otsu(volCT_converted)
volBW = volCT_converted < otsu_threshold

# 4. Delete the connected compotnents that touch the edge using scipy.ndimage.label
for idx in range(volBW.shape[2]):
    labeled_image, num_regions = label(volBW[:,:,idx]) #Use the label function to find 
    edge_pixels = find_edge(labeled_image) # Find the edge of the connection-labeled im
    for region_i in range(1, num_regions + 1): # 0 is the background, look at feature 1
        if region_i in edge_pixels: # The feature is at the edge
            mask = (labeled_image != region_i)
            masked_slice = mask * volBW[:, :, idx]
            volBW[:, :, idx] = masked_slice # Turn the labels to be background


# 5. Remove the connected regions with volume smaller than a particular volume
V = 500
volBW_finetuned = volBW
labeled_image, num_regions = label(volBW_finetuned)
for region_i in range(1, num_regions + 1):
    if np.sum(labeled_image == region_i) <= V:
        mask = (labeled_image != region_i)
        volBW_finetuned = mask * volBW_finetuned # Mask off the connected region that i

# 6. Smoothing
volBW_smoothed = binary_closing(volBW_finetuned)

#### plot the segmented binary lung volume ####
fig, ax = plt.subplots(5, 5)
for i in range(5):
    for j in range(5):
        ax[i, j].imshow(volBW_smoothed[:, :, (5*i+j)*5], cmap='gray')
        ax[i, j].axis("off")
        plt.tight_layout()
        plt.rcParams['figure.figsize'] = (30, 18)
####################################################################################
```



In [56]:

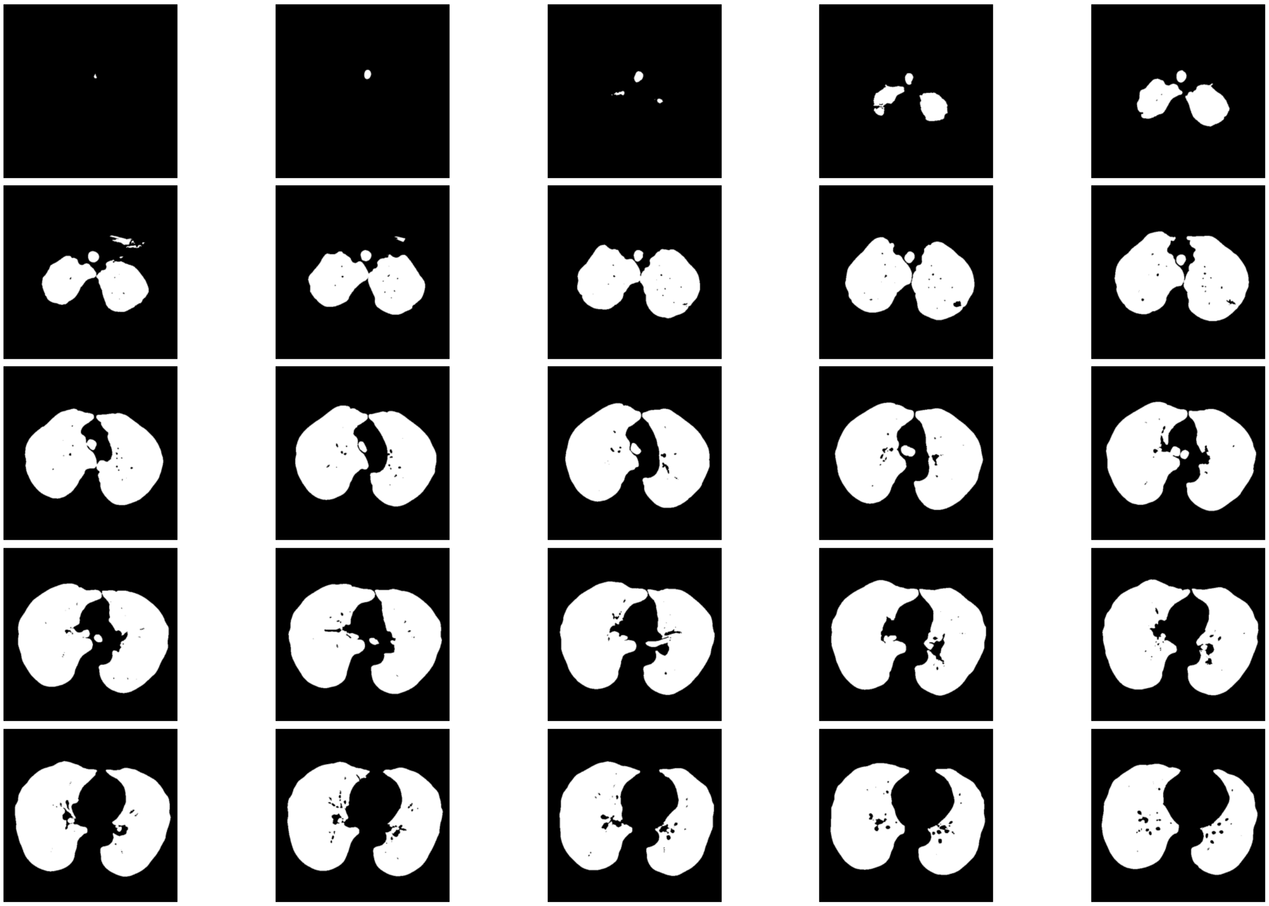
```
### WRITE CODE IN HERE. You can have up to 5 cells for this question, but only one is r
num_scan = 2

# 1. Read the scan file into volumetric CT
scans_path = "./CT_chest_scans/CT_chest_scans" # TODO: Change this to match where your
list_of_scans = os.listdir(scans_path)
list_of_scans.sort()
scan_path = os.path.join(scans_path, list_of_scans[num_scan])
list_of_slices = os.listdir(scan_path)

num_slices = len(os.listdir(scan_path))
volCT = np.zeros((ds2.pixel_array.shape[0], ds2.pixel_array.shape[1], num_slices)) #Ini

for idx in range(num_slices):
    slice_path = os.path.join(scan_path, list_of_slices[idx])
    ds = dicom.read_file(slice_path)
    volCT[:, :, ds.InstanceNumber-1] = ds.pixel_array

#2. Perform the conversion
volCT_converted = (volCT - np.min(volCT))/(np.max(volCT) - np.min(volCT))
volCT_converted = volCT_converted.astype('float32')


# 3. Binarize the image
otsu_threshold = threshold_otsu(volCT_converted)
```

```
volBW = volCT_converted < otsu_threshold

# 4. Delete the connected compotnents that touch the edge using scipy.ndimage.label
for idx in range(volBW.shape[2]):
    labeled_image, num_regions = label(volBW[:,:,idx]) #Use the label function to find
    edge_pixels = find_edge(labeled_image) # Find the edge of the connection-labeled im
    for region_i in range(1, num_regions + 1): # 0 is the background, look at feature 1
        if region_i in edge_pixels: # The feature is at the edge
            mask = (labeled_image != region_i)
            masked_slice = mask * volBW[:, :, idx]
            volBW[:, :, idx] = masked_slice # Turn the labels to be background


# 5. Remove the connected regions with volume smaller than a particular volume
V = 500
volBW_finetuned = volBW
labeled_image, num_regions = label(volBW_finetuned)
for region_i in range(1, num_regions + 1):
    if np.sum(labeled_image == region_i) <= V:
        mask = (labeled_image != region_i)
        volBW_finetuned = mask * volBW_finetuned # Mask off the connected region that i

# 6. Smoothing
volBW_smoothed = binary_closing(volBW_finetuned)

#### plot the segmented binary lung volume ####
fig, ax = plt.subplots(5, 5)
for i in range(5):
    for j in range(5):
        ax[i, j].imshow(volBW_smoothed[:, :, (5*i+j)*5], cmap='gray')
        ax[i, j].axis("off")
        plt.tight_layout()
        plt.rcParams['figure.figsize'] = (30, 18)
#######################################################################################
```



In [57]:

```
### WRITE CODE IN HERE. You can have up to 5 cells for this question, but only one is r

num_scan = 4

# 1. Read the scan file into volumetric CT
scans_path = "./CT_chest_scans/CT_chest_scans" # TODO: Change this to match where your 
list_of_scans = os.listdir(scans_path)
list_of_scans.sort()
scan_path = os.path.join(scans_path, list_of_scans[num_scan])
list_of_slices = os.listdir(scan_path)

num_slices = len(os.listdir(scan_path))
volCT = np.zeros((ds2.pixel_array.shape[0], ds2.pixel_array.shape[1], num_slices)) #Ini

for idx in range(num_slices):
    slice_path = os.path.join(scan_path, list_of_slices[idx])
    ds = dicom.read_file(slice_path)
    volCT[:, :, ds.InstanceNumber-1] = ds.pixel_array

#2. Perform the conversion
volCT_converted = (volCT - np.min(volCT))/(np.max(volCT) - np.min(volCT))
volCT_converted = volCT_converted.astype('float32')


# 3. Binarize the image
otsu_threshold = threshold_otsu(volCT_converted)
volBW = volCT_converted < otsu_threshold

# 4. Delete the connected compotnents that touch the edge using scipy.ndimage.label
for idx in range(volBW.shape[2]):
    labeled_image, num_regions = label(volBW[:,:,idx]) #Use the label function to find 
    edge_pixels = find_edge(labeled_image) # Find the edge of the connection-labeled im
    for region_i in range(1, num_regions + 1): # 0 is the background, look at feature 1
        if region_i in edge_pixels: # The feature is at the edge
            mask = (labeled_image != region_i)
            masked_slice = mask * volBW[:, :, idx]
            volBW[:, :, idx] = masked_slice # Turn the labels to be background


# 5. Remove the connected regions with volume smaller than a particular volume
V = 500
volBW_finetuned = volBW
labeled_image, num_regions = label(volBW_finetuned)
for region_i in range(1, num_regions + 1):
    if np.sum(labeled_image == region_i) <= V:
        mask = (labeled_image != region_i)
        volBW_finetuned = mask * volBW_finetuned # Mask off the connected region that i

# 6. Smoothing
volBW_smoothed = binary_closing(volBW_finetuned)

#### plot the segmented binary lung volume ####
fig, ax = plt.subplots(5, 5)
for i in range(5):
    for j in range(5):
        ax[i, j].imshow(volBW_smoothed[:, :, (5*i+j)*5], cmap='gray')
        ax[i, j].axis("off")
        plt.tight_layout()
        plt.rcParams['figure.figsize'] = (30, 18)
#######################################################################################
```



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. We highly recommend looking up some marching cube algorithms, but the easier way to go would probably use something like ITK-SNAP or 3D Slicer. Simply download ITK-SNAP (We recommend version 4.0.2) or 3DSlicer. You can save all your 3D segmentation binary image slices as NIfTI file (.nii.gz or .nii) using the `nibabel` package. Note that the affine matrix can be an identity. Then, in ITK-SNAP or 3DSlicer load the file and activate the 3D visualizer. For the former its the `Toggle volume rendering` option, while for the latter its useful to import the file as a segmentation and use the `Create closed surface representation` option. This should give you a figure like what you see below:

3d_vis

Show us that this works on at least two scans.

In [ ]:

```
### WRITE CODE IN HERE. You can have up to 5 cells for this question, but only one is r


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

In [ ]:

```
### WRITE CODE IN HERE. You can have up to 5 cells for this question, but only one is r

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

References

Correcting Non-Uniform Illumination - Rice Image

Segmentation - Brain Image

We thank Darvin Yi and Laura Bravo Sánchez for the content.