ASSIGNMENT 1

Name: 李采蓉, Student ID: 107062117 03/26/2021

1 Part 1

1.1 Read in and print out all the data fields in a DICOM file

First, I imported pydicom as pyd, and used pyd.dcmread() to read the dicom file. After reading in, I printed out the information.

```
# Part1: read Dicom Information
path = "CT_chest_scans"
dirList = os.listdir(path)
file = os.listdir(path+'/'+dirList[0])[0]
filepath = path+'/'+dirList[0]+'/'+file
printDicomInfo(filepath)
```

```
def printDicomInfo(filepath):
    ds = pyd.dcmread(filepath)
    print(ds)
```

```
Dataset.file_meta -------------------------------
(0002, 0000) File Meta Information Group Length  UL: 192
(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: 1.2.840.113654.2.55.249139163741242644304020243485943527041
(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: 1.2.840.113654.2.55.249139163741242644304020243485943527041
(0008, 0060) Modality                            CS: 'CT'
(0008, 103e) Series Description                  LO: 'Axial'
(0010, 0010) Patient's Name                      PN: '00cba091fa4ad62cc3200a657aeb957e'
(0010, 0020) Patient ID                          LO: '00cba091fa4ad62cc3200a657aeb957e'
(0010, 0030) Patient's Birth Date                DA: '19000101'
(0018, 0060) KVP                                 DS: None
(0020, 000d) Study Instance UID                  UI: 2.25.862087301405397123827718905017727342779506923977090073054 73
```

Figure 1: Dicom information output result

1.2 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

After reading the dicom file, I dealt with the padding value first. I set pixel values those were smaller than -2000 to zero, and then got the attributes, including RescaleIntercept, RescaleSlope, and pixelArray. Then I converted values with formula:

$$HounsfieldUnits = RescaleSlope * pixelArray + RescaleIntercept.$$

After converting, print out max, min, mean, standard deviation values with Numpy's API.

```
def dicom2HU(file):
    image = file.pixel_array
    image_HU = np.copy(image)

    #deal with padding value
    image_HU[image_HU == -2000] = 0
    #transport to HU
    intercept = file.RescaleIntercept
    slope = file.RescaleSlope

    image_HU = slope * image_HU + intercept

    #compute mean, var, max, min
    print("mean: HU ", np.mean(image_HU), "origin ", np.mean(image))
    print("standard deviation: HU ", np.sqrt(np.var(image_HU)), "origin ",np.sqrt(np.var(image)))
    print("max: HU ", np.max(image_HU), "origin ",np.max(image))
    print("min: HU ", np.min(image_HU), "origin ",np.min(image))
    return image_HU
```

```
mean: HU  -535.1012725830078 origin  63.39213562011719
standard deviation: HU  505.11585972297553 origin  1158.0840289672014
max: HU  1488.0 origin  2512
min: HU  -1024.0 origin  -2000
```

Figure 2: Max, Min, Mean and standard deviation output result

2 Part 2

2.1 Read in a 3D volume of dicom

As mentioned above (1.1), I used pyd.dcmread() and ran a for loop to read all of the files in one directory.

```
path = "CT_chest_scans"
dirList = os.listdir(path)
slices = [pyd.read_file(path+'/'+dirList[0]+'/'+s) for s in os.listdir(path+'/'+dirList[0])]
```

2.2 Sort all the slices to make it into correct order

I sent all dicom files into function "sort(slices)". In the function, I sorted the files with the attribute "InstanceNumber".

```
# sort: use InstanceNumber
def sort(slices):
    sorted_slices = (sorted(slices, key = lambda s: s.InstanceNumber))

    return sorted_slices
```

2.3 Normalize all the pixel from Hounsfield Units to float32 type number between 0.0 to 1.0 and display 25 slices in correct order.

After sorting, I ran a for loop to convert all of the dicom files' image values into Hounsfield Units, and sent the results to function "normalize(imgs)", which was used to normalize images values.

```
images_HU = np.stack([s.pixel_array for s in sorted_slices])
for i in range(len(sorted_slices)):
    images_HU[i] = dicom2HU(sorted_slices[i])

images_norm = normalize(images_HU)
```

In function "normalize(imgs)", I got MINBOUND and MAXBOUND of images by Numpy's API : np.min() and np.max(), and then applying the formula :

$$Normalization = (img - MINBOUND)/float(MAXBOUND - MINBOUND)$$

to all of the images. In the end, I plotted out 25 normalized images in sorted sequence.

```
def normalize(imgs):
    img_norm = np.copy(imgs)
    for i in range(len(imgs)):
        img = imgs[i]
        MIN_BOUND = np.min(img)
        MAX_BOUND = np.max(img)
        img_norm[i] = (img - MIN_BOUND) / float(MAX_BOUND - MIN_BOUND)


    for i in range(25):
        plt.subplot(5, 5, i+1)
        plt.imshow(imgs[i], cmap=plt.cm.gray)
    plt.show()
    return img_norm
```

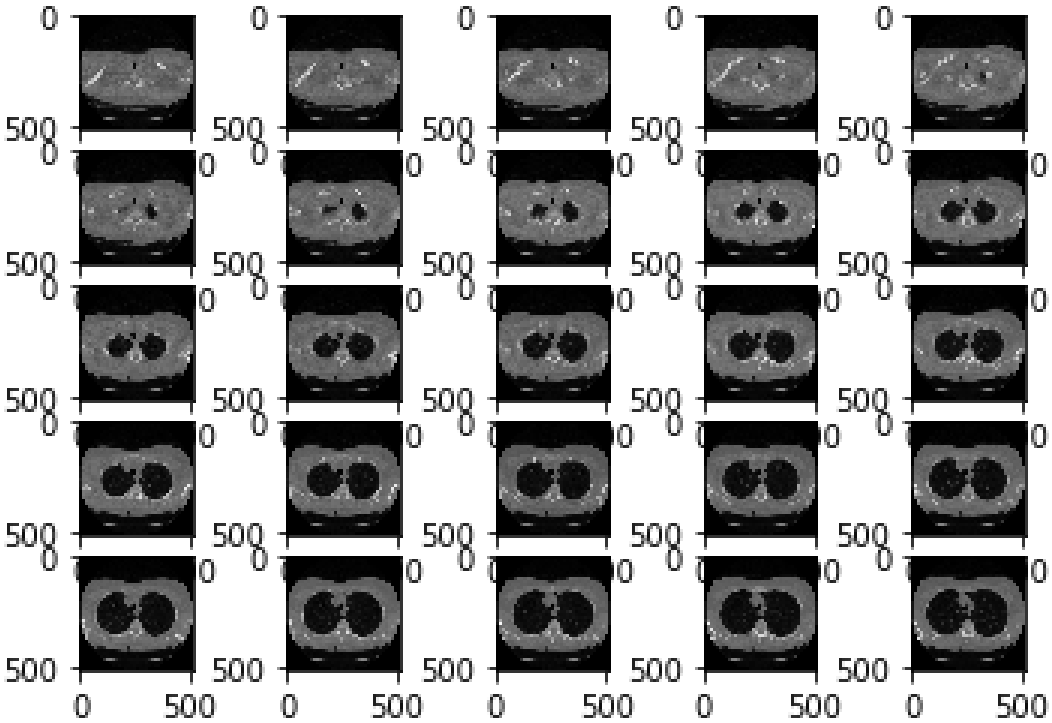

Figure 3: Normalization result in sorted sequence

3 Part 3

3.1 Segmentation: Using local mean

I got local median by using Numpy's API np.mean(), and using local mean as threshold. I plotted out the masked image and the histogram to see the relationship between threshold(the black line) and the input image. For each pixel value which is bigger than the threshold, set it to 2000, otherwise, set it to -2000.

```
def localMeanThreshold(img):
    threshold = np.mean(img)
    image = img.copy()

    plt.hist(image.flatten(), bins=80, color='c')
    plt.xlabel("Hounsfield Units (HU)")
    plt.ylabel("Frequency")
    plt.axvline(threshold , color='k', linewidth=1)
    plt.show()

    image[image > threshold]= 2000
    image[image < threshold]= -2000
    plt.imshow(image, cmap=plt.cm.gray, vmin = -2000, vmax= 2000)
    plt.show()
    return image
```

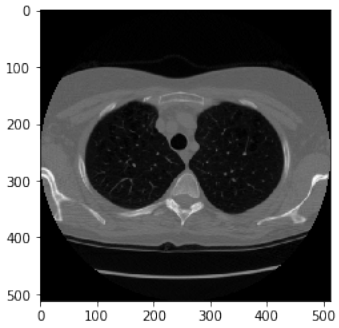

Figure 4: Original Image

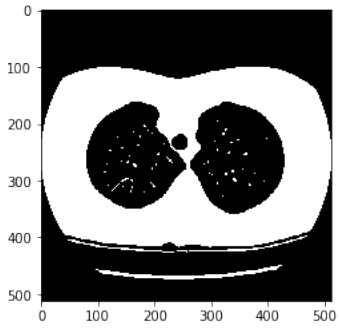

Figure 5: Segmented with Mean threshold

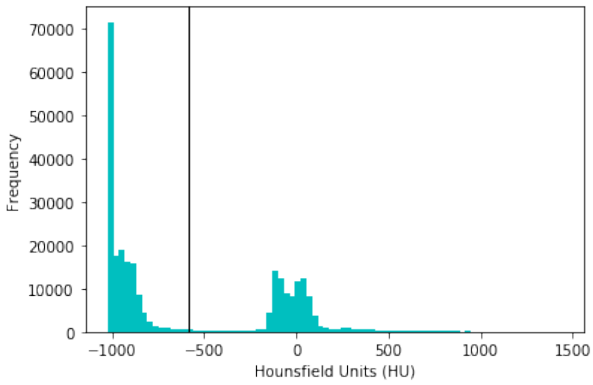

Figure 6: Histogram for original pixel values and threshold (black line)

3.2 Segmentation: Using local median

Such as the method mentioned above, in this case just changing the threshold from np.mean() to np.median.

I got local median by using Numpy's API np.median(), and using local mean as threshold. I plotted out the masked image and the histogram to see the relationship between threshold(the black line) and the input image. For each pixel value which is bigger than the threshold, set it to 2000, otherwise, set it to -2000.

```
def localMedianThreshold(img):
    threshold = np.median(img)
    image = img.copy()

    plt.hist(image.flatten(), bins=80, color='c')
    plt.xlabel("Hounsfield Units (HU)")
    plt.ylabel("Frequency")
    plt.axvline(threshold , color='k', linewidth=1)
    plt.show()

    image[image > threshold]= 2000
    image[image < threshold]= -2000
    plt.imshow(image, cmap=plt.cm.gray, vmin = -2000, vmax= 2000)
    plt.show()
```

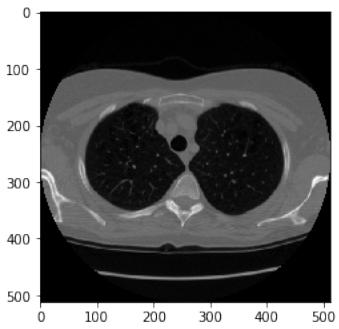

Figure 7: Original Image

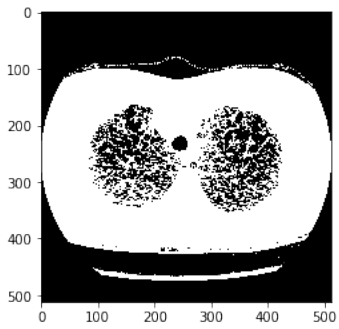

Figure 8: Segmented with Median threshold

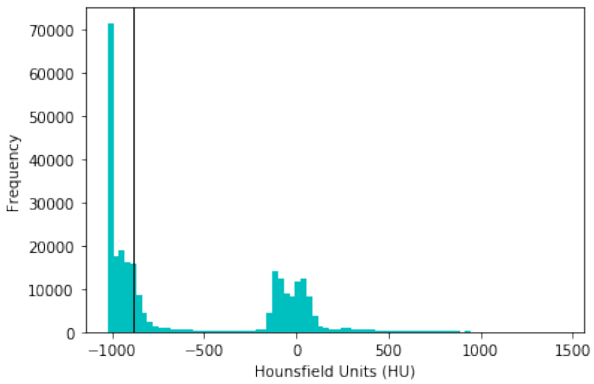

Figure 9: Histogram for original pixel values and threshold (black line)

4 Bonus: show 3D segmented result

First, in function "saveSegmented()", I used LocalMean threshold to segment all files in one directory and stored all of the results in another directory. With pixel values bigger than threshold, set it to -2000, otherwise, set it to 2000.

```
def saveSegmented(slices):
    if not os.path.exists("CT_chest_scans_segmented"):
        os.makedirs("CT_chest_scans_segmented")
    for i in range(len(slices)):
        file = os.listdir(path+'/'+dirList[0])[i]
        ds = pyd.read_file(path+'/'+dirList[0]+'/'+file)
        img = dicom2HU(ds)
        threshold = np.mean(img)
        img = np.where(img < threshold, 2000, -6000)
        img = img.astype(np.uint16)
        ds.PixelData = img.tobytes()
        ds.save_as("CT_chest_scans_segmented/"+file)
```

Then I used an online Dicom viwer: med3web (https://med3web.opensource.epam.com/) to get my reconstructed 3D model.

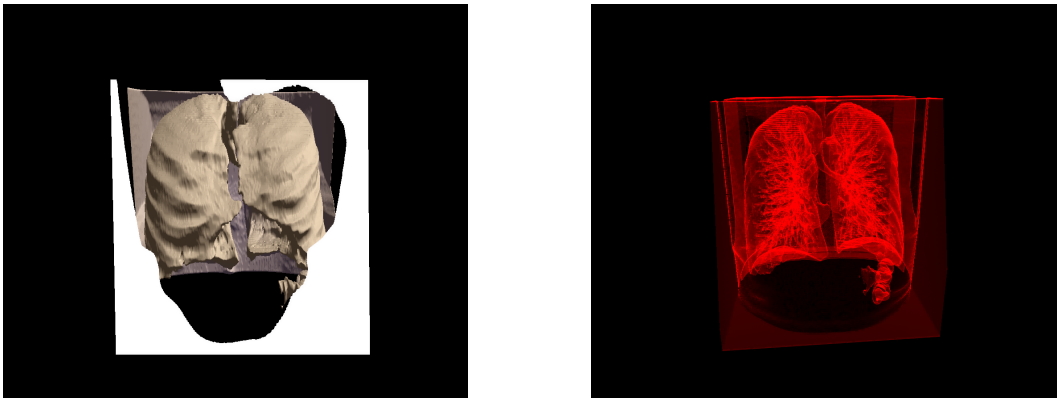

Figure 10: 3D reconstruction Output result

5 Question and Summary

Actually, I seldom program with python in the past, so the first problem for me was leaning how to program with python. Although it took a lot of time, I surely got much more familiar with python after completing this assignment. Such as knowing how to read files in directories, the usage of matplotlib, and using command lines in python, etc.

Another difficulty I have conquered this time is to find out the way to save segmented pixel data. Since I was not familiar with the APIs of Pydicom, I spent a lot of time searching for the way to save the pixel data correctly. At the beginning, the segmented pixel data was overwritten by the original data. I struggled at this problem for a long time, and finally found out the mistake I made when saving data.

As to the segmentation problem, I also tried using Ostu's method to get the segmented images. I used CV2's API "cv2.threshold()" to do so. Because of the image type difference

problem, I need to rewrite the pixelArray to PNG image and read it again using cv2.imread(). I'm not really sure whether the transformation from array to PNG image would make some differences or not, so I didn't put the result above.

```
# Otsu's thresholding
def OstuThreshold(img):

    image = img.copy()
    MIN_BOUND = np.min(image)
    MAX_BOUND = np.max(image)
    image = (image - MIN_BOUND) / float(MAX_BOUND - MIN_BOUND)
    image*=255
    cv2.imwrite('examples.png', image)
    image = cv2.imread('examples.png', 0)

    threshold,image1 = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)

    plt.hist(image.flatten(), bins=80, color='c')
    plt.xlabel("gray scale value")
    plt.ylabel("Frequency")
    plt.axvline(threshold , color='k', linewidth=1)
    plt.show()

    plt.imshow(image1, cmap=plt.cm.gray)
    plt.show()
```

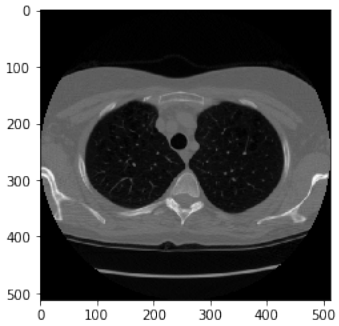

Figure 11: Original Image

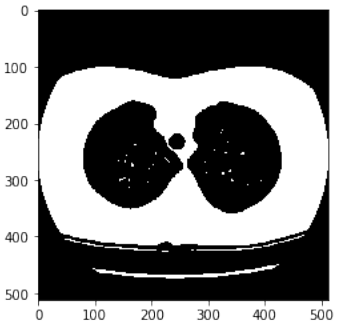

Figure 12: Segmented with Ostu's method

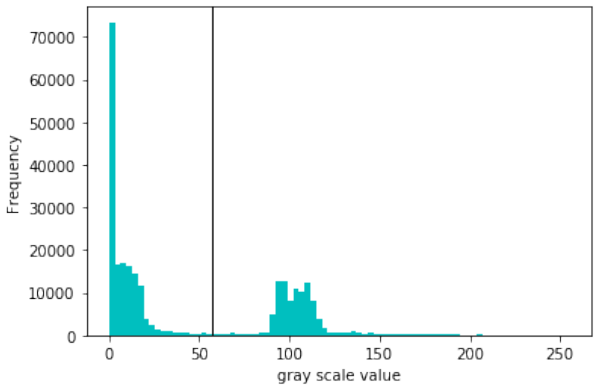

Figure 13: Histogram for Ostu's method