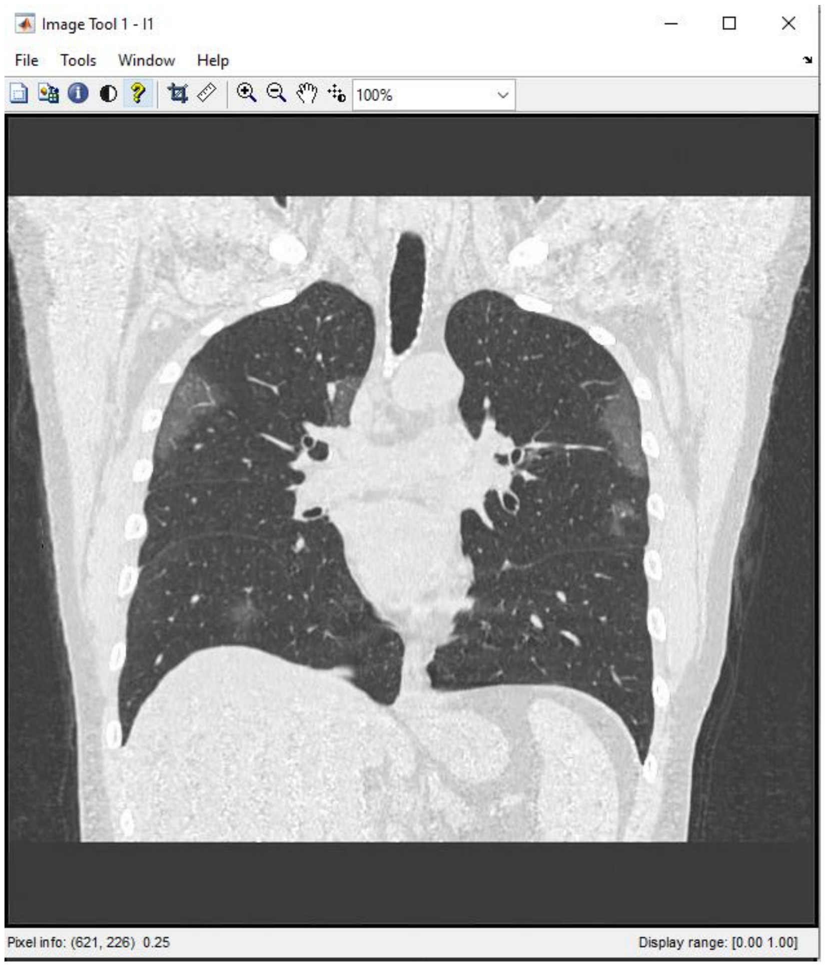
ENSC 474 Final Project
COVID-19 Ground Glass Opacity Lung Segmentation

Stefan Ungurean

301316291

Section 1: Exploration

April 12th: Opened the project description and read the deliverables for the first time. My thoughts instantly went to figuring out a method to sort the dark areas of the CT-Scans from the light ones and finally create some sort of pixel threshold to determine the gray ground glass opacity. So, I started by trying an idea on the first image, “Patient001”. I thought that the results on “Patient 001” would be the same as the results on other images. By using a function called “imtool” which allowed me to hover over an image and extract the pixel intensities at different pixel locations. Next, I manually came up with a range of acceptable pixel intensities for the ground glass opacities. The tool showed the pixel intensity next to the location in the bottom left of the image (as shown below). I came to a range of intensities between 0.4 and 0.7 since the image was converted to double. Then I was able to crop out the image values of intensity values between the threshold.



Pixel info: (621, 226) 0.25

Display range: [0.00 1.00]

Figure 1.1 Imtool Demo

Unfortunately, I did not get a chance to save the output of this result. I soon realized that I made a pretty big mistake. I had not cropped the image properly and there were plenty of other pixel values that fit the range I set before. I also tested to see if this method would work with other images, which it did not, because each image had a slightly different range of pixel intensities for the ground glass opacity.

April 13th: After my mistakes the previous day I decided on creating a good crop function that got a tight view of the lungs. However, I quickly realized that each set of lungs was not the same and a generic crop function would not work perfectly with each image. Nevertheless, I eyeballed a range of pixel values to crop the image that fit best for all the images. The results of my crop are shown below.

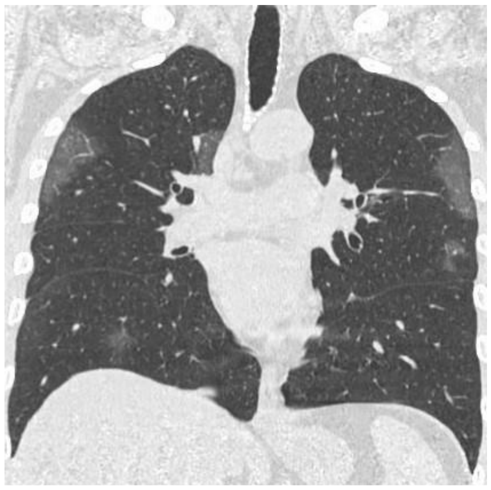


Figure 1.2 Crop Function Patient001

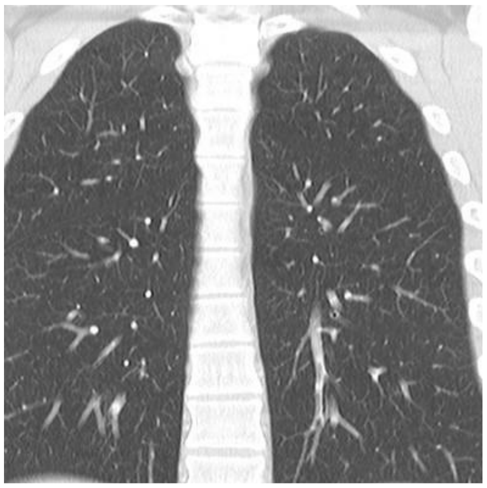
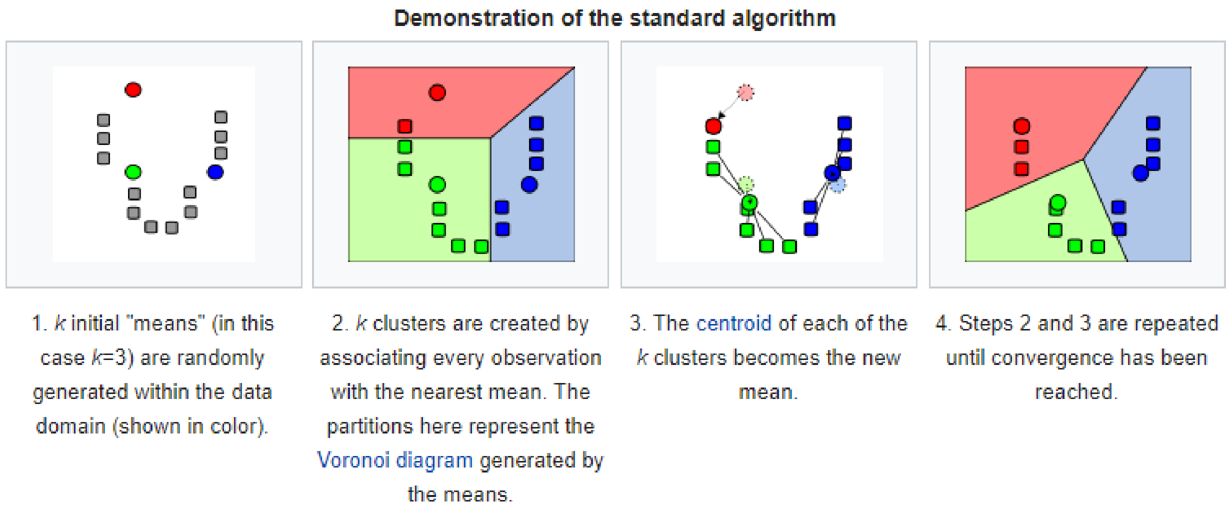


Figure 1.3 Crop Function Patient005

The crop did not work properly for each lung, but I decided to return to the crop later. After reading the document again and doing some reading online, I came across image segmentation using K-mean segmentation. This seemed to be exactly what I was looking for. This algorithm partitions x number of observations into k clusters based on the mean of different clustered centroids [1].


Demonstration of the standard algorithm

1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).

2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

3. The centroid of each of the *k* clusters becomes the new mean.

4. Steps 2 and 3 are repeated until convergence has been reached.

Figure 1.3 K-Means Clustering [1]

Implementing this algorithm using the function “imsegkmeans” which segments an image I into k different clusters. I received the following output (using the left lung as an example).

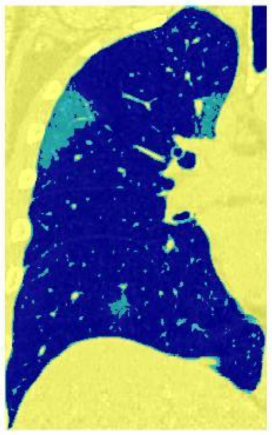


Figure 1.4 K-Means Clustering Patient001

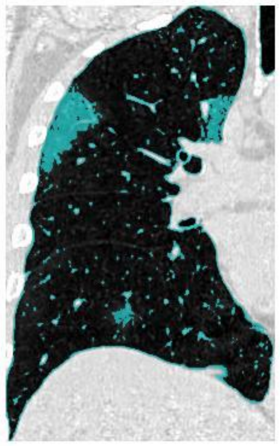


Figure 1.5 Cluster Selected Patient001

This seemed to be exactly what I wanted. However, I quickly ran into another issue. When I selected the clusters that represented the ground glass opacity, I did it manually by choosing the second level of the segmentation. As seen above, the I told the segmentation to run in the levels. For “Patient001,” the second level of the segmentation was the GGO (ground glass opacity). However, that was not the case for other images like “Patient010,” for example.

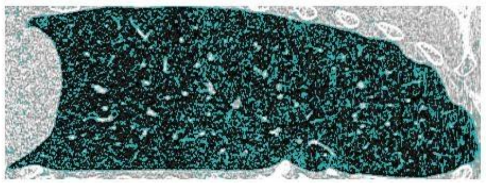


Figure 1.6 K-Mean 2nd Level Patient010

As seen in figure 1.6, the second level is not the same as the second level in patient 001. So, I had to hunt for a different method. I was disappointed until I ran across Otsu’s method.

Otsu’s method was similar to K-mean segmentation however it uses the image’s histogram and a threshold to separate the image into classes. It then minimizes the weighted sum of the variances of the classes which are separated by the threshold, i.e. “foreground” and “background” [2]. Using a function called “multithresh” which computes the necessary threshold for image A based on Otsu’s method and “imquantize” which converts the image to a two-level(binary) image I managed to extract the GGO. Interestingly, “multithresh” can generate thresholds for N levels and “imquantize” then returns an image with N+1 discrete levels. To show this clearly, I changed the image to RGB to show the levels.

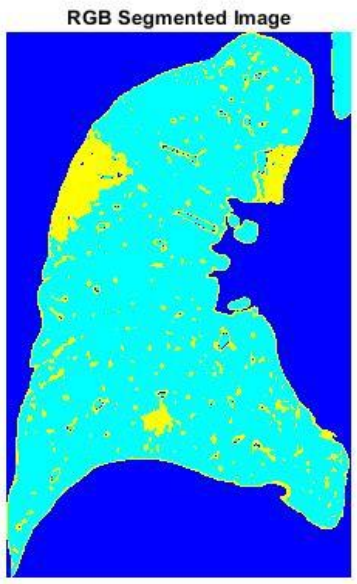


Figure 1.7 Otsu's Method Patient001

This looks like the results from the k-mean segmentation however, for Otsu's method, all the GGO was segmented to level 3 of the quantized image (more in depth in section 2 of report).

April 14th: I was content with the results Otsu's method gave me and I decided to try and figure out a way to better crop the image. After doing some reading online, I found a better method of cropping the lungs based on object detection. Essentially, I had to figure out a way to detect the large lung elements in each image without cropping any of it out like the current crop function was doing. Thankfully, there is a very useful pair functions called "bwlabel" and "regionprops" which can be used to select objects in an image (more info in section 2). I included these functions in my crop function and the results were brilliant for every image. After finding a better way of cropping, the segmentation from the day before turned out better and seemed more accurate. Finally, I decided on determining the severity of GGO in the lungs by the percentage, i.e dividing the segmented area pixels by the total number of pixels that defined the lungs. Overall the methodology I developed throughout the past few days seemed to work very well.

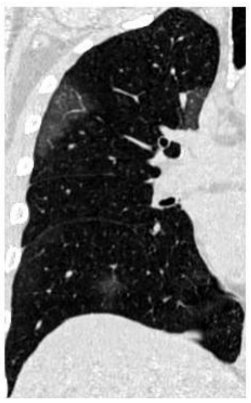


Figure 1.8 Left Lung Fully Cropped

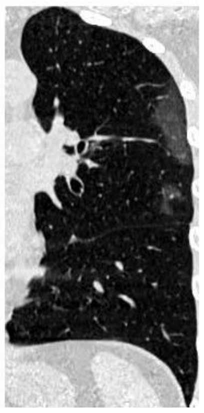


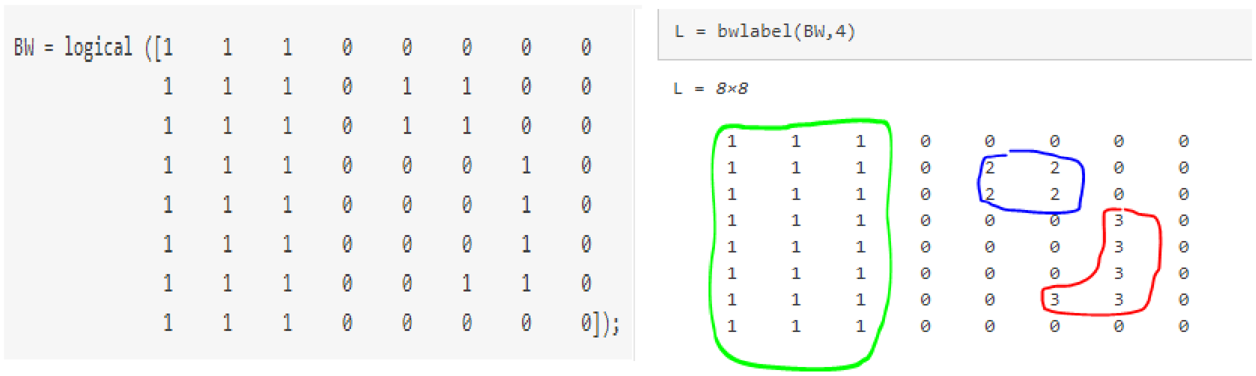
Figure 1.9 Right Lung Fully Cropped

Section 2

Based on the results I received from the previous days I was happy with Otsu's method for ground glass opacity and lung segmentation as well as the crop function I developed. In this section, I will give a detailed account of every function and method used for the final results with a reflection on how well the algorithm performed after every analysis. For ease of investigation, I will be using the first patient to demonstrate the algorithm.

Cropping:

After my trial and error with cropping each CT Scan, the 'bwlabel' and 'regionprops' functions worked very well together in cropping each lung individually. I thought having each lung individually would make for more accurate results. Bwlabel returns a matrix of 'connectivity' [4] which essentially means an image that has connected values will be grouped together by different values.



```
BW = logical ([1    1    1    0    0    0    0    0
               1    1    1    0    1    1    0    0
               1    1    1    0    1    1    0    0
               1    1    1    0    0    0    1    0
               1    1    1    0    0    0    1    0
               1    1    1    0    0    0    1    0
               1    1    1    0    0    1    1    0
               1    1    1    0    0    0    0    0]);
```

```
L = bwlabel(BW,4)

L = 8×8

     1     1     1     0     0     0     0     0
     1     1     1     0     2     2     0     0
     1     1     1     0     2     2     0     0
     1     1     1     0     0     0     3     0
     1     1     1     0     0     0     3     0
     1     1     1     0     0     0     3     0
     1     1     1     0     0     3     3     0
     1     1     1     0     0     0     0     0
```

Figure 2.1 BWLabelFunctionality [4]

Then regionprops is a function that selects certain region properties from the bwlabel matrix such as 'Area' and 'BoundingBox'. 'BoundingBox' selects the smallest box containing a region while 'Area' selects the number of pixels in that region. Using this idea and creating a threshold for the area to only select the lung elements I managed to segment the lungs fully. I had to create a threshold since the 'BoundingBox' property created many different segments in the image of which the lungs were a part of, so I needed to create a threshold of pixels so it would filter out all the unwanted areas. After the crop I decided to also adjust the image to increase the contrast so the GGO would become more apparent.

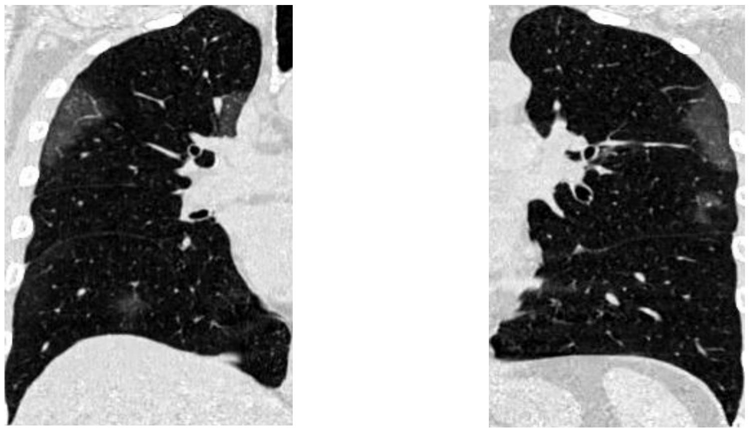


Figure 2.2 Left and Right Lung Cropped

The crop function was algorithm was fairly robust. The only thing I would improve is to figure out a way to automatically measure the areas of the lungs and create the threshold for cropping, rather than manually selecting the areas through trial and error. Nevertheless, the function served its purpose really well.

Lung Segmentation:

The next important function to inspect is 'LungSegment' which segments the actual lung itself from the white background and sides of the CT Scan. This method implements Otsu's method by using a function called 'graythresh' to determine the appropriate threshold that minimizes intra-class variance of white and black pixels. It takes the histogram of the image, in this case either lung, and calculates probabilities of each class separated by a threshold t. Finally, the algorithm performs a weighted sum of variances of the two classes which is the intraclass variance [2].

$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t)$$

Equation 2.1 Minimizing Intraclass Variance [1]

Essentially, if a threshold is chosen there are two things that need to happen to ensure the threshold has been chosen properly. First, say the threshold separates the image to background and foreground, so two classes, then the variance within each class needs to be minimized as to ensure the pixels within that threshold actually belongs there. Second, the variance between each class must be maximized to ensure the two classes have pixel values that are more dispersed.

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i)$$

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i)$$

$$\begin{aligned}\sigma_b^2(t) = \sigma^2 - \sigma_w^2(t) &= \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2 \\ &= \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2\end{aligned}$$

Equation 2.2 Class Probabilities [2] Equation 2.3 Maximizing Interclass Variance [2]

Equations (2.2) represent the class probabilities which is essentially the number of pixels within that particular class divided by the total number of pixels in the image or both classes. Above is also an equivalent form of equation (2.1) which is maximizing interclass variance. For a smaller case, a calculation of the intraclass variance is show in figure (2.4).

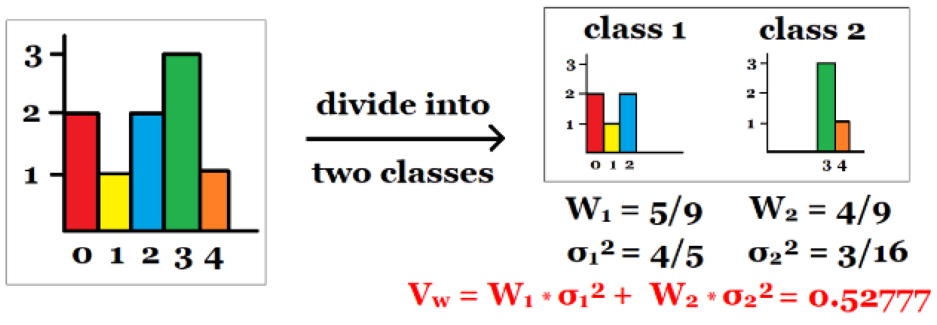


Figure 2.3 Intraclass Variance [3]

Implementing this algorithm using ‘graythresh’ and then using ‘imbinarize’ to change every value above the threshold to 1 and everything below to 0 to fully convert the image to a binary one I received the following output.

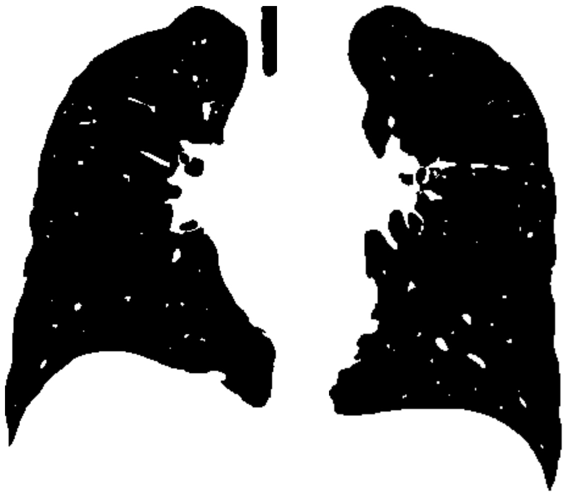


Figure 2.4 Binary Lungs

After filling the holes in the image and taking the complement I managed to get the full binary lung and extract the lung itself from the original image by multiplying the complement by the figure (2.2). Then, I subtracted figure (2.4) from the multiplied image to get rid of the very bright parts.

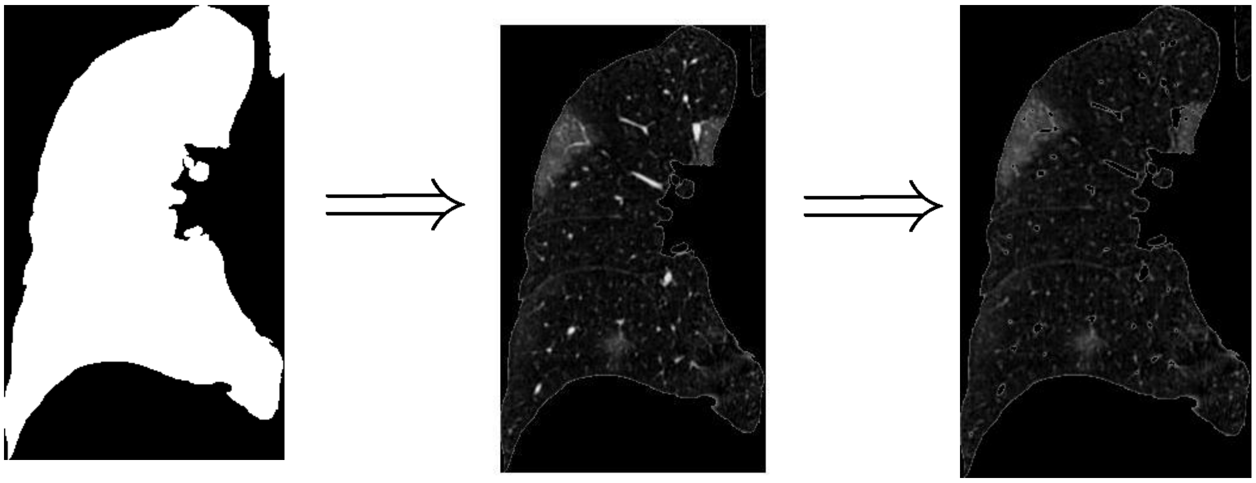


Figure 2.5 Binary Lung to Segmented Lung (a,b,c)

Ground Glass Opacity Segmentation:

The next step was to segment the GGO from figure (2.6). To do this I had to implement Otsu’s method again, but this time I had to create two different thresholds to segment the lungs into three different sections: the black outside, the GGO itself, and the remaining bright pixels. To do this, I used a separate function called ‘multithresh’ which allowed me to specify the number of thresholds I wanted. As stated in my exploration, ‘imquantize’ returns an image with N+1 discrete levels if multithresh is called with N levels. In my case, I called multithresh with 2 levels and the quantized output gave 3. Turning the image to an RGB one for easier visualization the GGO seemed to be quite isolated in yellow.

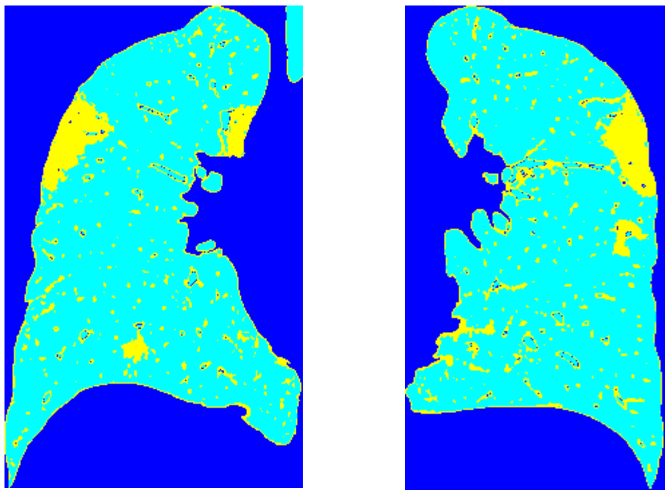


Figure 2.6 RGB of Segmented GGO

After doing some pixel intensity filtering by manually getting rid of values above a certain threshold and filling whatever holes there are in the image, I call another function called ‘bwareaopen’. This function scans for any cluster of pixels less than 100 pixels in area and ‘opens’ them, making them zero. The result is quite good. If you overlay figure (2.7) on the original, the GGO has been fully segmented.

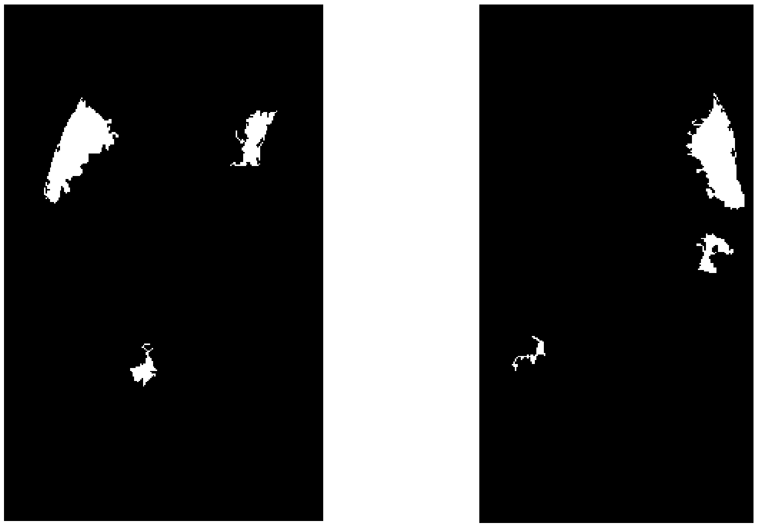


Figure 2.7 GGO in Binary

For some cases, the algorithm is sufficient up to this point to segment the GGO, however, the image is passed through a final function called ‘ExtractGroundGlass’. All this function does is fine tuning by filtering the GGO based on solidity and area using the ‘regionprops’ function again. Solidity is what fraction of the actual area your region is. The difference is not apparent in most cases, but it does help. As seen in ‘Patient005 below’, ‘ExtractGroundGlass’ removes the bottom line which would have affected the overall percentage calculation.

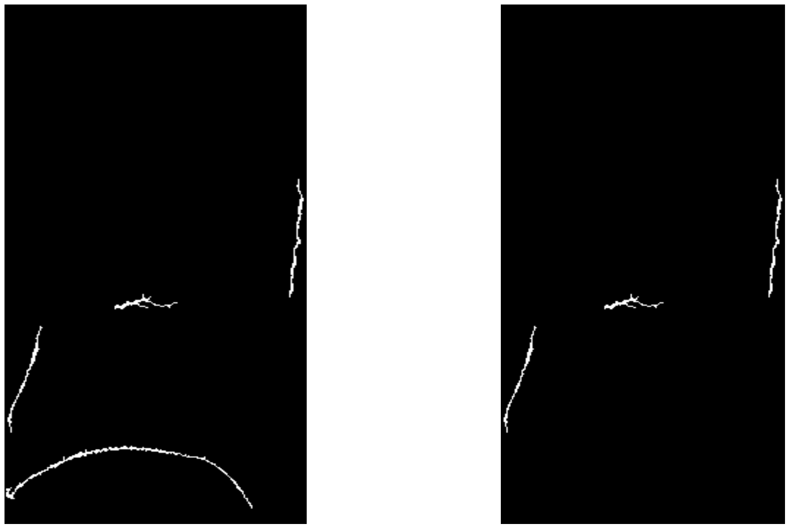
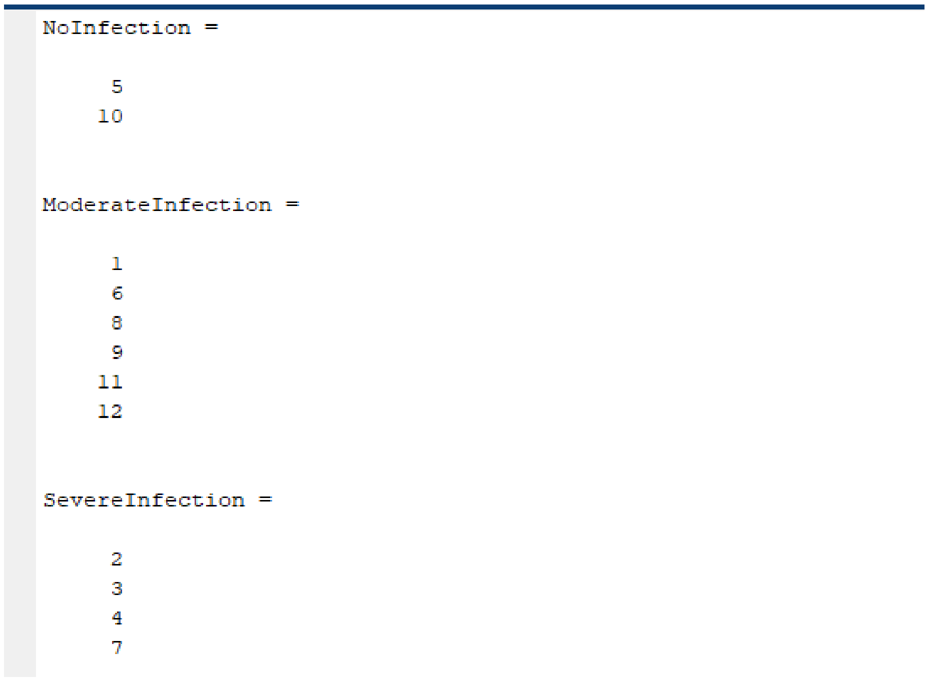


Figure 2.8 Fine Tuning in Patient005 Left Lung

Results:

The overall severity of ground glass opacity in the lungs was calculated by summing all the non-zero pixels in the final GGO output like figure (2.8) or (2.7) and dividing it by the total number of pixels in figure (2.5 b). By analyzing all of the percentages at the end of all the outputs, I decided that anything under less 1% was considered healthy, between 1% and 10% was infected/early onset whereas anything above 10% was infected/severe. The results are summarized in figure (2.9) or in the command window.



```
NoInfection =

     5
    10


ModerateInfection =

     1
     6
     8
     9
    11
    12


SevereInfection =

     2
     3
     4
     7
```

Figure 2.8 Fine Tuning in Patient005 Left Lung

Analysis & Conclusion

The algorithm was successful in determining infection and severity in eleven out of the twelve cases. The only case that was difficult was ‘Patient006’. Here the script says the patient suffers from moderate infection. However, this is not the case when someone looks at the original CT scan. The issue lies with the circled red areas, the algorithm cannot discern between the blurred parts and GGO.

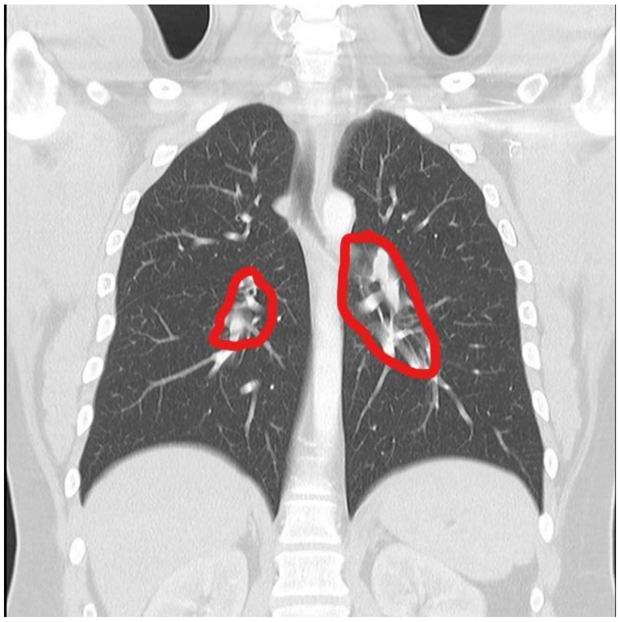


Figure 2.9 Patien006

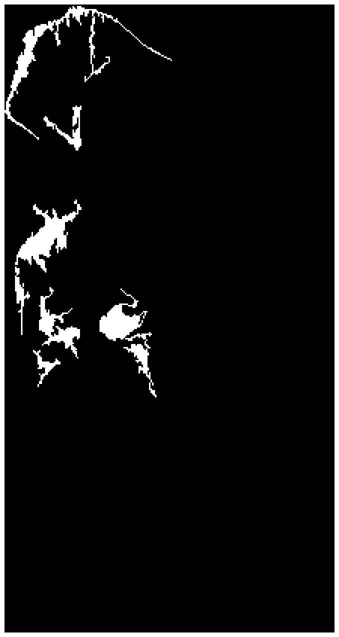


Figure 2.9 Patien006 GGO Right Lung

Now the actual percentage of affected tissue the algorithm gives for this patient is 3.9083% which is not a lot, but is above the threshold of 1%. I could not figure out how to fix the algorithm for this case. I think this is due to the CT Scan itself not being as clear as the other eleven. Maybe issues like this are the reason why radiologists are still needed in medicine today. The algorithm analyzes correctly, but it cannot pass judgement as an end all, be all.

For all other patients, the algorithm gave stellar results and GGO segmentation. I strongly believe the cropping of individual lungs and repeating Otsu’s method more than once during segmentation allowed me to come to accurate results.

Overall, I am content with my results and I really enjoyed this whole project. I found it extremely interesting to work on something relevant while learning new image processing techniques.

References

[1] “K-Means Clustering”, April 13th 2020, https://en.wikipedia.org/wiki/K-means_clustering

[2] “Otsu’s Method”, December 3rd 2019, https://en.wikipedia.org/wiki/Otsu%27s_method

[3] HBY coding academic, “Otsu Thresholding – Image Binarization”, Jan 18th 2019, https://medium.com/@hbyacademic/otsu-thresholding-4337710dc519

[4] “bwlabel”, Mathworks, https://www.mathworks.com/help/images/ref/bwlabel.html