Michael Ungureanu
301338428
mungurea@sfu.ca

ENSC 474
Final Project Report

Michael Ungureanu 301338428 mungurea@sfu.ca

Table of Contents:

Michael Ungureanu 301338428 mungurea@sfu.ca

Section 1:

Exploration

Beginning the project:

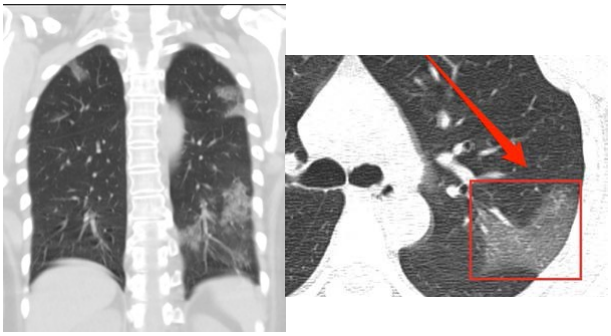

Figure 1
Lung scans of lungs infected with COVID-19, indicating the swellings

Planning:

In this project, finding the area within the lungs that are infected will require some methods to do the following:

-Find the lungs
-Find the area the lungs occupy in the image (pixels of lung)
-Find the swellings (differentiate from the light parts of the lungs)
-Find the area the swellings occupy in the image (pixels of swell)

Find the Lungs:

Noticing the lungs are darker than the image lends itself to wanting to find the dark parts of the image, several images have black backgrounds. This means finding the lungs is about finding the lungs, not finding the dark spots.

For finding the lungs:

Methods proposed in project description:

- Ostu's Method:

Used for thresholding the image, transforming an image from 0-255 to effectively 0-1 (black or white). This method takes the histogram of an image, superimposed with the inter-class variance (which creates a distribution over the histogram), bisects this distribution and then the upper half of the histogram becomes the whites, the bottom half the blacks.

This method would be useful for simplifying the images, making them easier to work with by having them be much harsher edges.

Pros:

- Creates a binary image that is easy to work with
- Matlab has built-in functions to do this
- quick

Cons:

- If the swellings and the white parts of the lungs are both above the threshold they could both be grouped into white. This could cause issues not only in distinguishing the healthy parts of the lungs from the swollen ones, but if the swellings are up against the edges of the lungs, it could cause the perceived shape of the lungs to change.

Possible alternatives:

Do it myself. Then I could adjust where in the distribution the threshold would be. This can then be adjusted for best results. However this would only work for scans where the intensities do not match. If the swellings are brighter in one image than another, they may end up on the wrong side of the threshold.

- Active Contours:

This method involves creating a "snake" which is a line of n points, and then moving it from one side of the image to the other. As the snake contacts strong edges, the points stick. By ensuring the distance between points isn't too great, the snake can wrap around shapes and not get stretched across the entire image.

Pros:

- Creates a contour of strong edges that can then be used to identify shapes
- Effectively finds shapes in an image
- Can be done recursively

Cons:

- Requires strong edges, so the image would have to be somewhat thresholded
- Complicated, involving properties such as "energy" that could over complicate
- Depends on the density of the points on the lines, so can miss small details. This could cause the shapes to inaccurate.
- Changing the direction of travel (from x to y, for example) could be difficult
- Could be slow

An alternative would be to start in the middle of the image, rather than on the side, and work outwards. Given that the lungs are always centered in the image, the first dark shapes that will be met working out from the middle will be the lungs. Then, when the shape is found, wrap around as the method prescribes. Rather than having point-properties such as energy that have to constantly be checked, simply define the distance to points in terms of the size of the image.

If a point passes, for example, 1/5th of the image without finding a desired edge, discard the point.

This will easily find the inside edges of the lungs but the outside edge remains unfound. Furthermore this method would still benefit from a thresholded image.

Attempt 1:

Using this method here:
https://www.mathworks.com/help/images/detecting-a-cell-using-image-segmentation.html
in an attempt to create a mask that could be overlayed over the lungs.

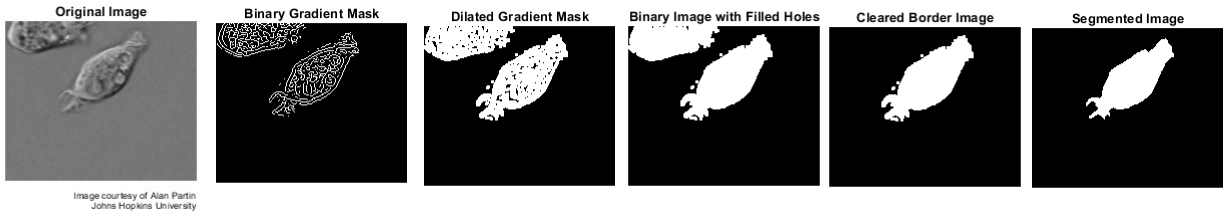

Figure 2
A series of images exemplifying the method in the link above

Having this mask would firstly allow for counting the pixels inside the lungs, and second could aid in isolating the lungs by multiplying the mask by the image, where everything outside of the lungs would be zeroed and everything within would remain.

This would have the drawback of making the lungs the same as the background, but it could aid in isolating the swellings. If this was not desired, an inverted image could be used (so the lungs would be white against a black background).

An example of the code used:

```
I = imread('Patient008.jpg');
I = rgb2gray(I);
I = double(I);

[~, threshold] = edge(I, 'sobel');
fudgefactor = 0.8;    %the closer to 1, the fewer lines appear.  The closer to 0,
the more
BinaryMask = edge(I, 'sobel', threshold * fudgefactor);

structure90 = strel('line', 6, 90);     %structure('type', thickness, angle)
structure00 = strel('line', 6, 0);      %structure('type', thickness, angle)

MaskDilate = imdilate(BinaryMask, [structure90, structure00]);      %dilate using
the line structures as guides

DilateFill = imfill(MaskDilate, 'holes');       %fill in the holes  in the image
imshow(DilateFill)
```

Figure 3
A fuzzy contour image of the lungs

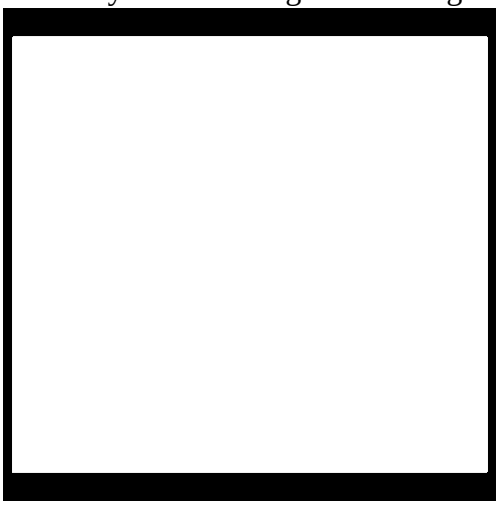

Figure 4
The “fill” step of the method

Figure 5

The same method applied to a grainy image

Figure 3 shows an early attempt at the method. Initially, the original code had a 'fudgefactor' of 0.5. With the intent of having the lungs (and the sides of the image) being the only closed shapes, this was moved closer to 1 to avoid the ribs creating closed shapes. Then, the thickness of the line structures was increased, to close a gap in the lung structure.

This method failed because some of the images have a border. This means that the part where the method fills the gaps causes the entire image to be white because that is the first 'hole' that the image sees. This failure can be seen in Figure 4.

This method failed again when applied to grainy images, as can be seen in Figure 5. There are too many edges in that image compared to the image used to create figure 3. Because the line thickness and the edge sensitivity are kept the same, the result of Figure 5 is unusable.

It may be possible to avoid the border issue by breaking it with a black rectangle positioned at centre bottom of the image (it is unlikely lungs will be in this part of the image). The images with no borders could be zero-padded to match. However this would still not account for the issue in Figure 5, where the line thickness and edge sensitivity required to create Figure 3 made an unusable image in Figure 5.

Attempt 2:

In an attempt to get the method from Attempt 1 to work, thresholding was attempted. The idea was to create an image where the lungs and the background are the black parts, however the rest of the image is white. This would then allow there to be no edge leading into the image (from border to image), and hopefully clarify the grainy image.

For this attempt the built-in Matlab implementation of Otsu's method was used. The resulting thresholded image shows very quickly why this will not work. The image used was the same as in Figure 3.

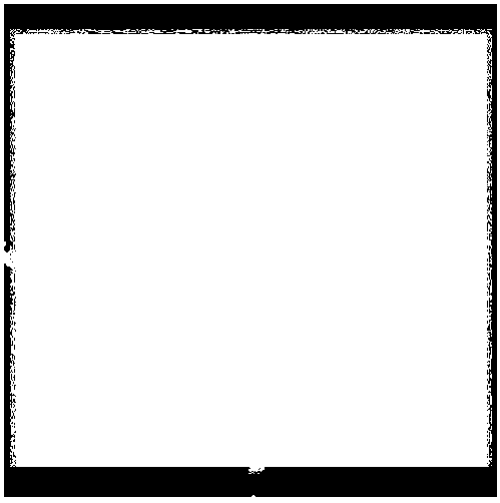

Figure 6
A failed attempt at thresholding

Figure 6 is the result of the following code snippet:

```
threshold = graythresh(I);
IBinary = imbinarize(I, threshold);
imshow(IBinary)
```

Most likely the issue is in the black border skewing the distribution and because the lungs are not pure black (0) while the border is, the whole image becomes above the threshold and thus white.

Attempt 3:

After some further testing, it was realized that this method should work, but wasn't. It eventually was found to be the conversion to double. Any image that was converted to double would have issues with this. Something does not seem to work when the image is doubles.

The results are

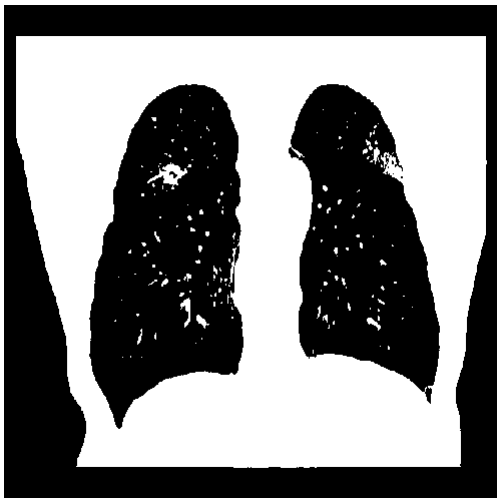

Figure 7
A properly thresholded image

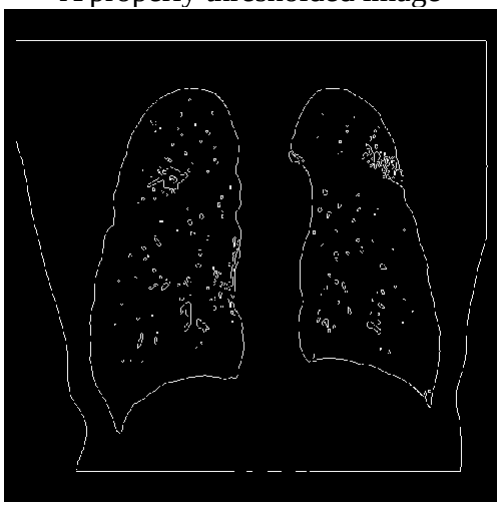

Figure 8
An image with proper edge finding

This method does fairly well with the grainy image, the results of which can be seen in Figure 9

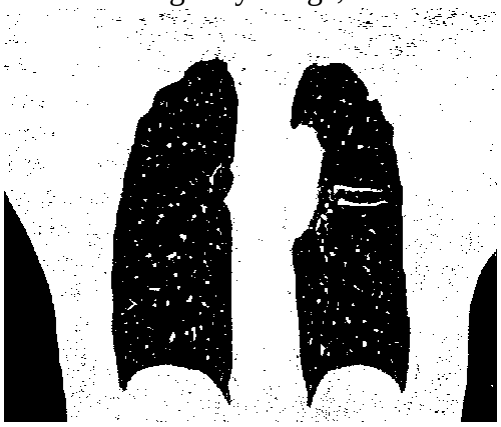

Figure 9
A properly thresholded grainy image

Figure 10
The edges of the thresholded image

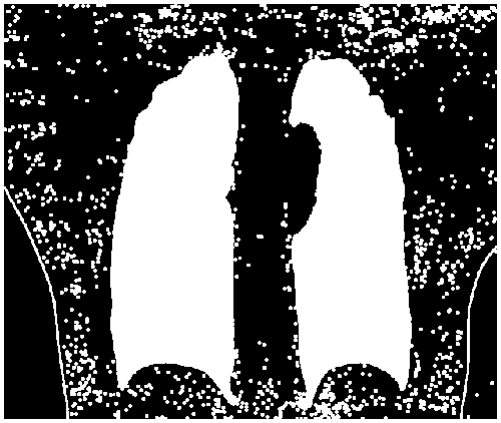

Figure 11

The lungs have been properly filled

This method still has issues when there is still a border in the image.

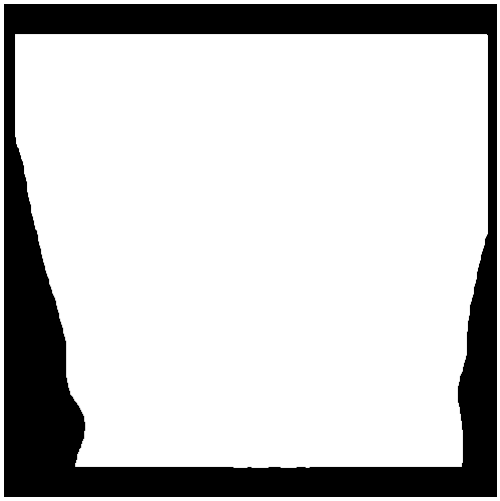

Figure 12
The results of the filling of an image that had a border

If there is a border, it must be either removed or broken.

There are test images with only borders on one side, and the results in this method can be seen in Figure 13. This is after the items touching the edge of the image are removed. This is done using the matlab built in function imclearborder() function.

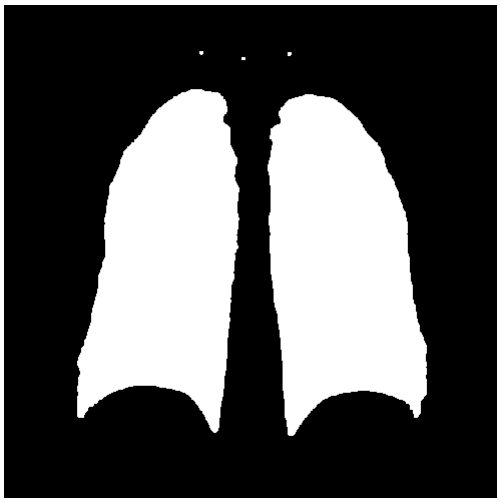

Figure 13
Lungs isolated in an image that only had a border on the sides

However in this image there are unwanted dots.

These dots point towards a greater issue that is plaguing this technique, and it is best exemplified when trying to use the imclearborder() function on the grainy image in Figure 9. As can be seen in Figure 9 and more clearly in Figure 11, there are enough miscellaneous edges and dots to connect the lungs to the border and this results in a removal of a lung from the desired mask, Figure 14.

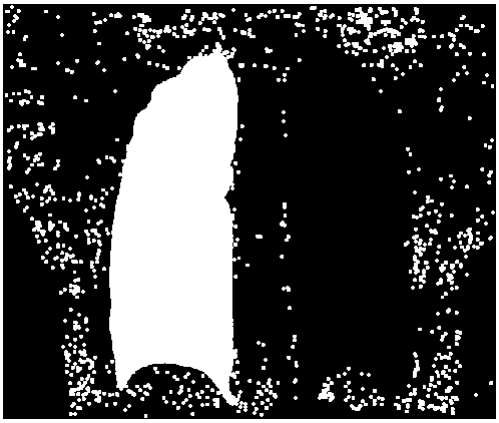

Figure 14
Using imclearborder() on a grainy image

This clearly is an issue and so these images need to be made less grainy for the desired results.

Using the Matlab built-in median filter, medfilt2(), on the binary image of the grainy image (Figure 9) gives the result shown in Figure 15. This filter cleans up the image and provides a much less noisy image. The median filter on the grainy (noisy) image has the desired effect of blending the grays into each other while still allowing the edges on the sharp contrasts (against the lungs and sides of the body) to show through. This is because the smaller noise blends into one gray mass while the sharp edges get blurred, and these sharp-to-blurred edges still show as the same edge when taking the edge() function edges.

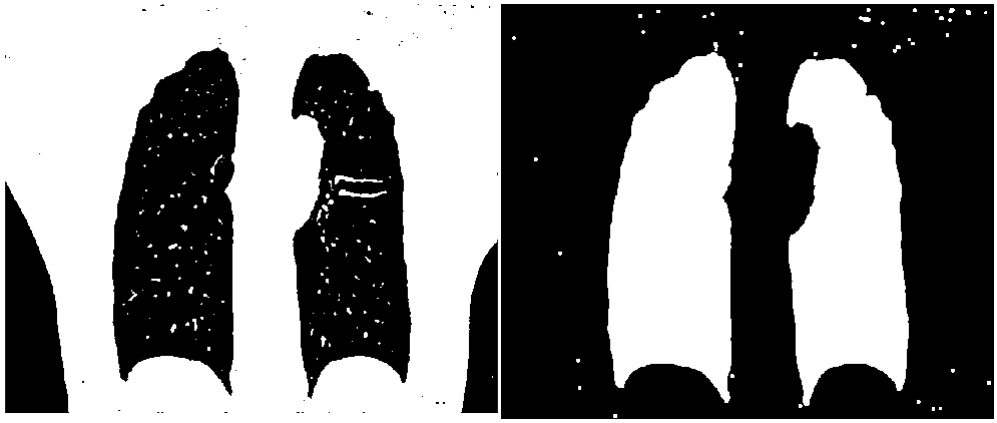

Figure 15
left: the smoothed binary image. Right: the filled and imclearborder() image

This still leaves some specks on the grainy image, however it does clear out the dots from Figure 13. This result is seen in Figure 16.

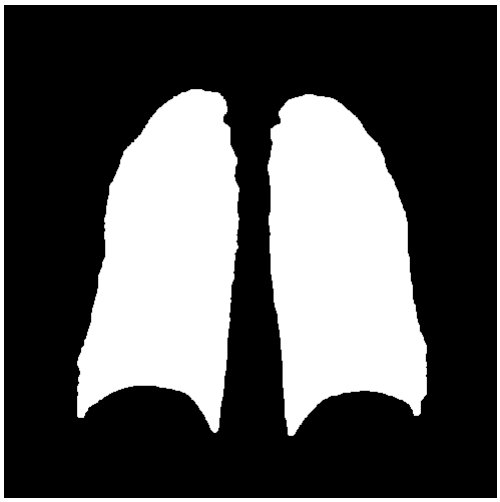

Figure 16
filtered version of image used for Figure 13

Attempt 4:

While this is still a continuation of Attempt 3, there is now a game plan. The goal is to use the masks created by Attempt 3, multiply them by the original image, thus creating an image that is black everywhere except the lungs, where it is nearly black, with white patches (blood vessels) and gray patches (swellings). Additionally, the white areas in the masks will be counted so there is a number of pixels that denotes the lung area. This will be used in the calculation of the % swollen number.

There are still some spots lingering on the noisiest images, and if there is time then a method or eliminating these may be found, but otherwise they are not going to seriously affect the calculations.

Once the image is black except for the white and gray sections, another thresholding shall be done, perhaps manually, and this should lump the grays with the blacks, leaving only the blood vessels. These can then be made to eliminate the blood vessels in the mostly-black-image. Then, all that will be left that is not black is the swellings. These will then be counted, compared to the lungs area, and this number will be reported.

In testing this, it is important to know if the dilation from the method in Attempt 3 is worth doing. This would apply to the clearest images, however it is hard to test these images as most of them have a border. So, the first goal in Attempt 4 will be to eliminate the border from the top of an image. Once there is one border gone, the rest will be eliminated in the imclearborder() function.

Trimming the border was fairly straightforwards. It was done by comparing the rows of the image, going row by row, confirming that the entire row was identical to the next. If it was, it was a border and thus should be removed.

The only hitch happened in copying the image, without the desired border, into another image. Although there was a semicolon, it continues to output the results of the copying to command window. If there is time this will be addressed.

After removing the border, the image results in Figure 17

Figure 17
Image with border trimmed.

While the border is successfully trimmed, the image presents other issues as can be seen in Figure 18.

Figure 18
Improperly filled lungs.

The issue in Figure 18 is the presence of smaller complete shapes which result in the filling function to fill them but not fill the lungs. This means that it then does not fill the lungs and they do not make a proper mask.

The smaller shapes will need to be removed.

Using the bwareaopen() built-in function, any shapes smaller than a certain number of pixels can be removed. This size will have to be adjusted based on the size of the image.

In addition, some blurring of the lines is helpful. This method was used in Attempt 1, albeit with more extreme dilation. For this attempt, the goal is go close any 1-pixel gaps that are sometimes left in the edges of the lungs where they double back or have many notches. By dilating the lines from 1 pixel to 2, the edge of the image is still preserved however the 1 pixel gaps are filled.

Then, the shapes smaller than 1/200th of the overall size of the image are removed. As the lungs are large, they should take up more than 1/200th of the overall image. They do for all of the test set. If they do not, this value will have to be adjusted. The results of successfully removing the small areas is shown in Figure 19.

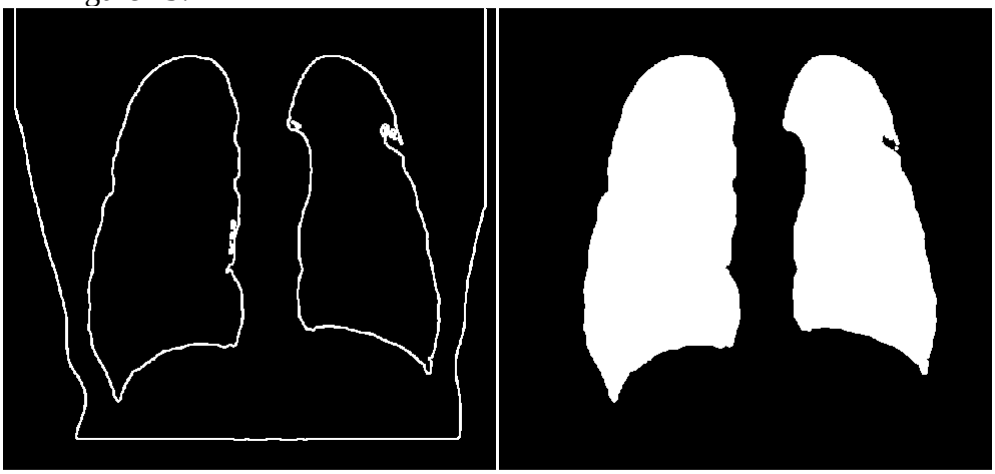

Figure 19
Lungs with smaller shapes removed, and the resulting mask.

When the mask is applied to the image, the resulting image is black except for the lungs. This can be seen in Figure 20.

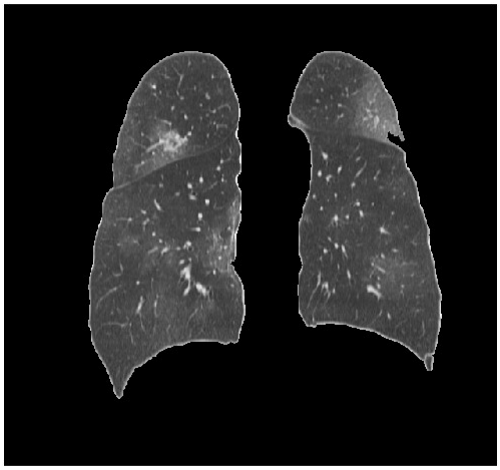

Figure 20
Only lungs.

To further clarify this image, making it so the lungs are even darker would be advantageous. As a result, the mask made in Figure 21, and the resulting image, were created.

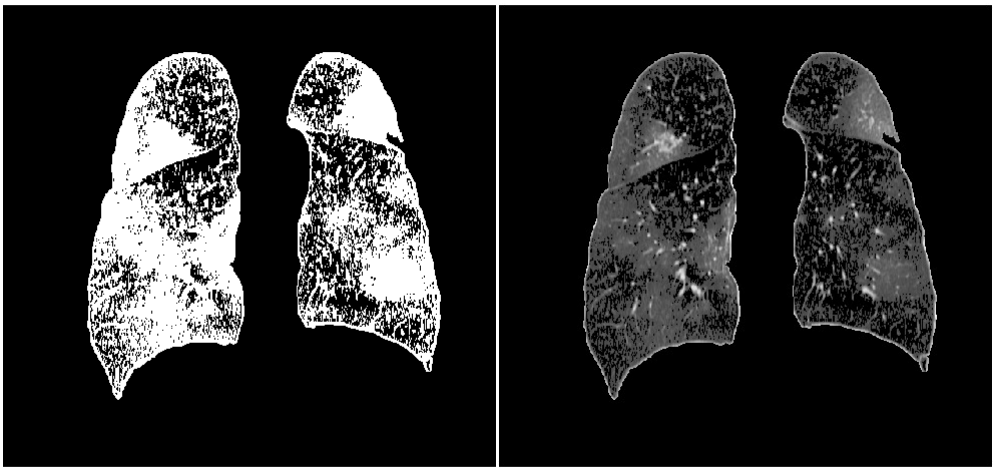

Figure 21
A mask and the resulting lung images, encouraging a more clear delineation between lungs and vessels or swellings.

Once the lungs have been isolated, the next step in the plan is to isolate the blood vessels and remove these from the lungs. The right side of Figure 21 is a lung image where the grey of the lung from the original image is blackened. This photo would be difficult to apply the original method to, because the image has far too much black. The threshold of the original image still applies well to this image, and so using that threshold, a binary image seen in Figure 22 can be created. Figure 22 shows the binary image, which will be used as a mask, as well as the isolated lungs.

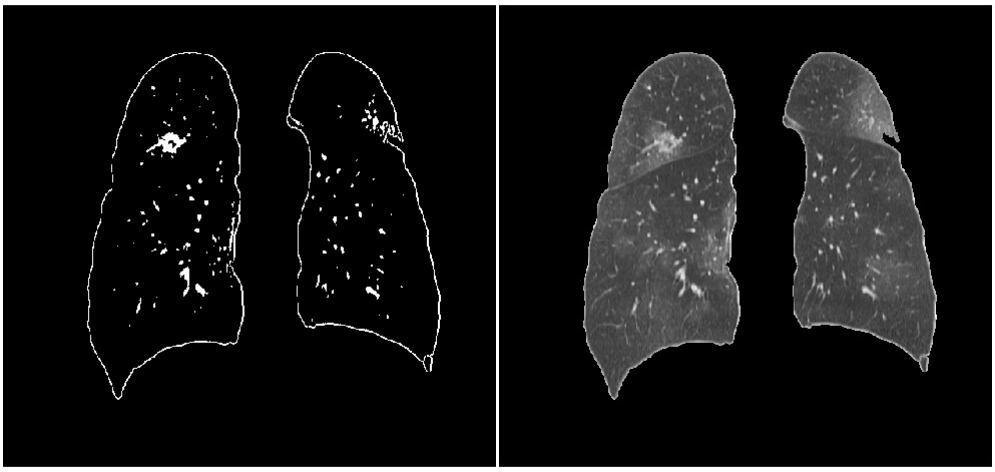

Figure 22
A mask (left) derived from the threshold of the original image and an isolated lung image (right)

The only difference is that instead of using the mask as is, the complement will be used. This is because the mask is of the white parts, which need to be removed from the lung image to properly find the swellings.

The result of this can be seen in Figure 23.

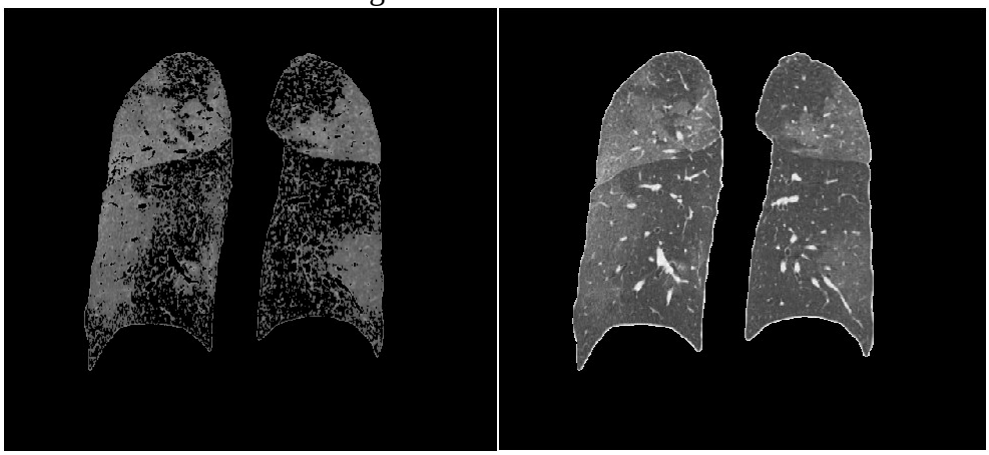

Figure 23
A lung without the white (left) and the lung image that the white was removed from (right).

An unfortunate artifact of this method is a border around the lungs. This is not desired and could conflict with any further computation or modification. As such, removing it is best. To do this, the dilation technique works very well. The mask image from Figure 22 has a border only 2 pixels wide, however the border in the isolated lungs (seen on the right in Figure 23) is 4 pixels wide. Dilating it to 4 pixels wide and using it in the same mask as left of Figure 23 results in Figure 24.

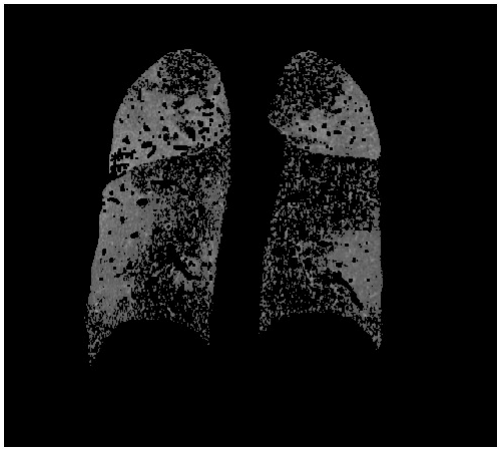

Figure 24
An isolated, white-less lung image.

Now the goal is to remove the noise from the image that is not the swellings. The noise is, for the most part, small clusters of pixels compared to the large areas of solid colour that is the result of the swellings.

This becomes a real issue in healthy lungs, where there is no swelling however there still remains significant noise. An example of this is seen in Figure 25.

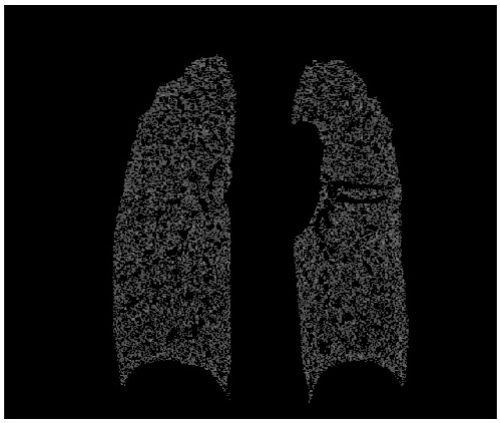

Figure 25
A white-less, healthy lung with significant noise.

Some options for this could be to use the same technique as before, where any shapes of a size smaller than a certain amount of pixels are removed. This could have issues in that the size of the shapes would be fixed and could have adverse effects.

Another option would be to proceed with the healthy lungs as if they were unhealthy, and in the last step when a comparison between lung area and final “swellings” pixels is calculated, if the value is too high it can be ignored. This option is clearly not recommended, as it assumes that no lungs can be 100% unhealthy which is not true.

A possible option would be to erode the images. Eroding the images using imerode() would reduce the noise in lungs such as Figure 25 by removing pixels from every edge. This method would shrink all shapes in the image by some number of pixels, allowing the larger shapes (swellings) to remain.

Given the noise being constant throughout the image, removing pixels from the swellings should not actually affect the comparison with healthy lungs. Because the noise is throughout the entire image, the swellings will have some number of extra pixels equal to the number of noise pixels adjacent to the swellings. This number would be reduced with this method, further bringing the final number closer to the reality.

Figure 26 shows a comparison between the white-less lung, and lung eroded by 2 pixels.

Figure 26
White-less lung and its eroded counterpart.

This method is effective as it successfully fully removes everything from a healthy lung, for example Figure 27. Yet another example is in Figure 28.

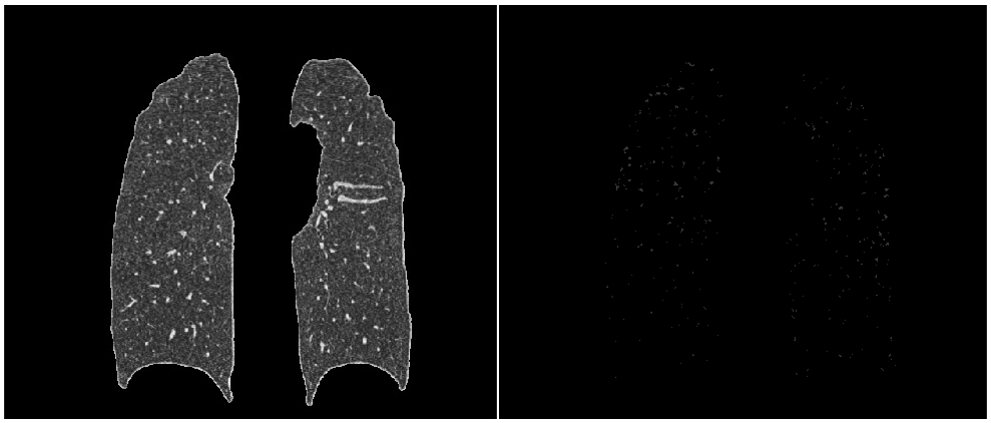

Figure 27
A healthy lung, and everything except for swellings in a lung that has no swellings (so there is nothing left).

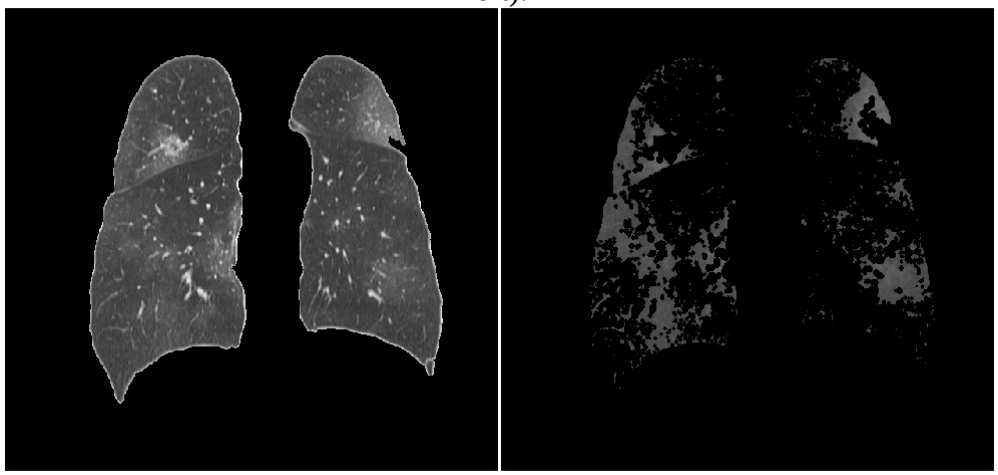

Figure 28
Unhealthy lung, left, and the non-swellings removed, right.

The results from this last part of the process can be used as a mask to show a final image of only lungs. This process results in a binary image that only contains the overall area that is covered by swellings. In a healthy lung, this image will be black. If it is unhealthy, it will look like Figure 29.

Figure 29
Only the swellings.

Finally, to count the % swollen, the swellings are binarized, so as to be only black and white, then the whites are counted by a simple sum. This is compared to the lung mask. These two masks are seen in Figure 30.

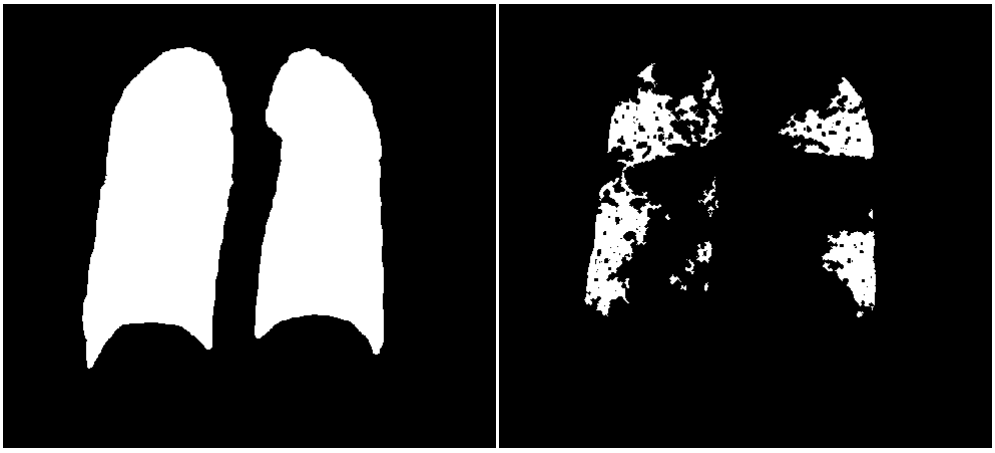

Figure 30
Binary mask of lungs (left) and swellings (right) used to copmare pixel counts

Section 2:

Proposed Solution

To find the swellings in a CT scan of lungs, in an effort to determine if they are infected by COVID-19, a series of masks and mask-based image processing techniques were used.

COVID-19 is a disease that affects the respiratory system, primarily manifesting in the lungs as swellings and mucus secretion similar to pneumonia. This tends to leave scar tissue in the lungs, as well as swellings. These appear in CT scans as grey areas within the lungs. Figure 31 shows a healthy lung compared to a lung infected with COVID-19.

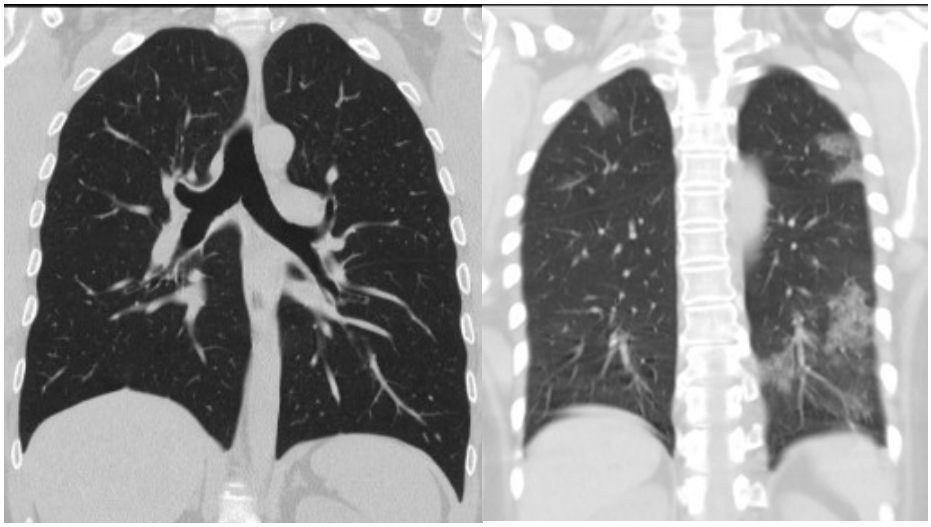

Figure 31
A healthy CT scan (left) compared to a COVID-19 infected CT scan (right)

From these images, as well as the provided test set, some key observations can be made:
1. The lungs are always a dark colour
2. The surrounding image is of many shades of grey and white and even sometimes black
3. The swellings/scar tissues are darker than the blood vessels that already exist in the lungs

To isolate the area of the lungs affected by COVID-19, a mask-based system will be used. These masks will be divided into sections, as each mask presents its own series of challenges.

Mask 1: Isolating the Lungs

To find the lungs, a combination of thresholding and edge detection is used. First, the image is thresholded using Matlab's built-in implementation of Otsu's method.

Otsu's method involves finding a threshold within an image that minimizes variance. Defined as a weighted sum of variance between the two classes, it passes over an image and creates a distribution of the variance between pixels intensities as well as a distribution of pixel intensities. By

bisecting the variance distribution, it finds an intensity above which the image will be white and below it will be 0.

For the purpose of finding the lungs, this is incredibly helpful. Because the lungs are always very dark on an image, they become black, while the rest becomes white.

This creates a binary image, on which edge detection can be applied.

Using Matlab's built-in edge detection function creates a binary image consisting of only edges usiung the following code snippet:

```
[~, threshold] = edge(Image, 'sobel');
fudgefactor = 0.2;    %the closer to 1, the fewer lines appear.  The
closer to 0, the more
BinaryEdge = edge(image, 'sobel', threshold * fudgefactor);
```

The binary image of only edges is not perfect for the mask yet. The edges are 1 pixel thick, and often have some disconnections in the edge that defines the shape of the lungs. To remedy this, the edges are dilated to 2 pixels by the following code snippet:

```
structure90 = strel('line', 2, 90);     %structure('type', thickness,
angle)
structure00 = strel('line', 2, 0);      %structure('type', thickness,
angle)

MaskDilate = imdilate(BinaryEdge, [structure90, structure00]);
%dilate using the line structures as guides
```

After which, if the shapes are not closed then they are assumed to not be the lungs. The intent is to use Matlab's imfill() function to fill in the lungs. However, if there are shapes within the lungs, this function will fill those and not the lungs. To remove shapes within the image, anything smaller than 1/200th of the image area are removed with the bwareaopen() function.

This function takes a binary image and an integer as input. It then removes any shapes containing fewer pixels than the integer. A third input can be used to define connectivity of the shapes, however the default connectivity is sufficient for this application.

Once the small shapes are removeed, the only shapes remaining are lungs or torso. Because the torso is typically touching the edges of the image, using imclearborder() effectively gets rid of these and imfill() can then be used. The results are seen in Figure 32

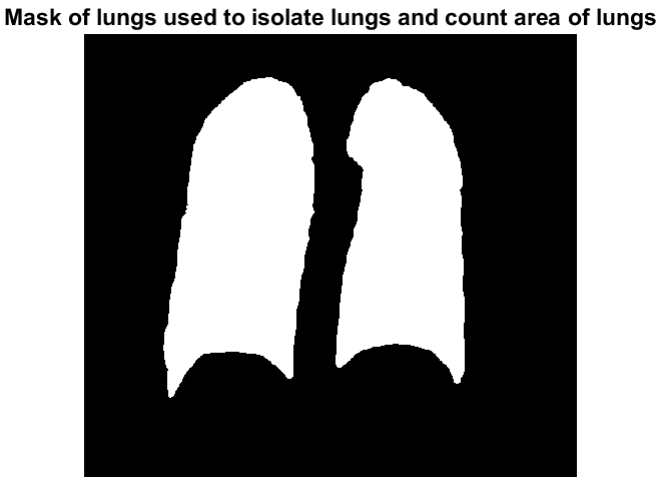

Figure 32

This binary mask, when multiplied by the original image, creates an image of only the lungs, seen in Figure 33.

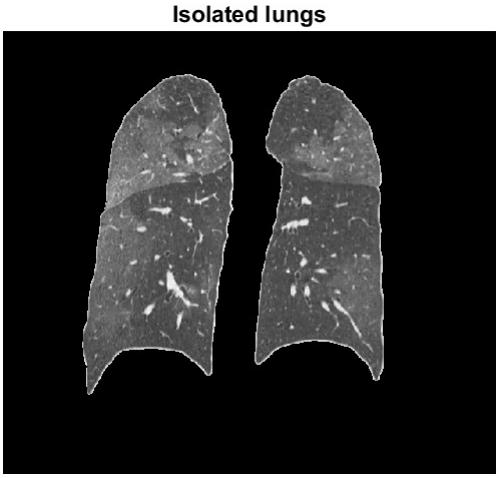

Figure 33

Mask 2: Isolating Blood Vessels and Noise

In Figure 33, anything outside of the lungs has been removed. Next, anything that is not the swellings/scar tissues need to be removed. To do this, the same method could be applied, however with some modification.

First of all, using Otsu's method to find a threshold in an image of only lungs is inneffective. There is too many black pixels in the image, causing the thresholding to output a binary image matching Figure 32. This is a step backwards.

The goal is for the background of the lungs to be black while the rest of the lungs (blood vessels, swellings, etc.) remains white. To do this, a histogram filter was used to create an image where the background is no longer black, however the white becomes even more white. This image is a perfect candidate for Otsu's method.

This results in a binary image which is then used as a mask, exactly like Mask 1. The results are an image where the background of the lungs is black, meaning the only things in the image that are not black are blood vessels, swellings, scar tissue, and noise. This can be sen in Figure 34.

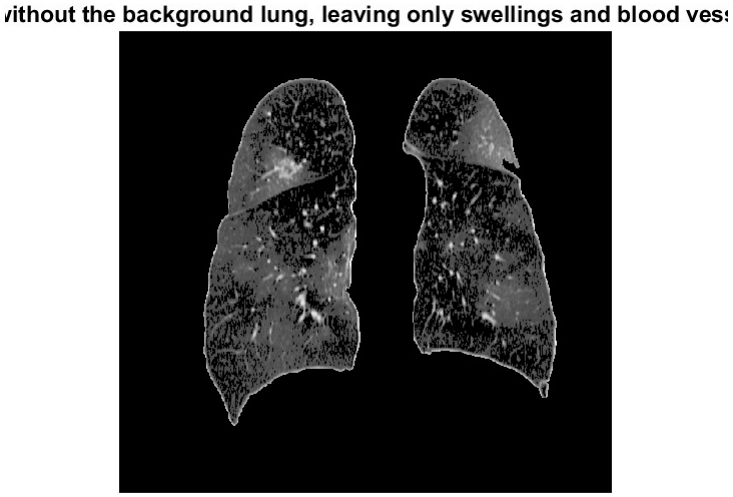

Figure 34

Mask 3: Removing Blood Vessels and Noise

In Figure 34, it is clear which parts of the lung should be removed. There is a white border, and nearly-white blood vessels.

To create a mask to remove these, the threshold used in Mask 1 can be applied. In Mask 1, one of the issues was the presence of small shapes within the lungs. In this case, those shapes are desired, and so the same threshold is used to generate the same binary image.

Using the same method, the edges of this binary image are found and then dilated and filled. The results are seen in Figure 35.

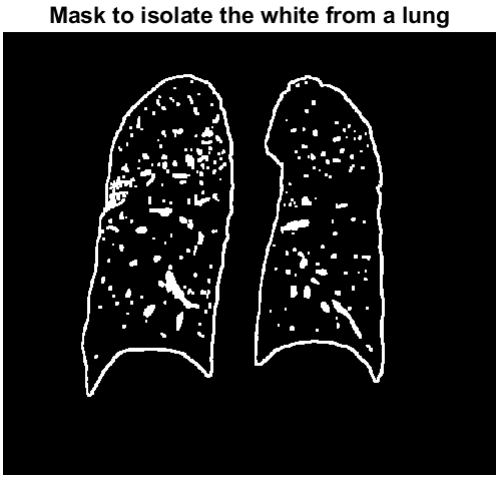

Figure 35

This mask is applied to the isolated lung image to create an image that should have only swellings/scar tissues, and some noise, seen in Figure 36.

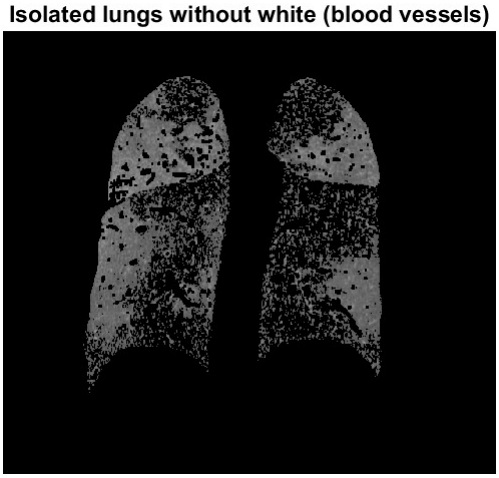

Figure 36

Mask 4: Isolating the Swellings

To isolate the swellings, no thresholds are needed. The image in Figure 36 already contains only black except for the desired areas. The goal is to remove the noise remaining in the image.

Mask 3 will result in an image of only noise if the lungs are healthy, so Mask 4 needs to be able to differentiate between healthy and infected lungs. This is done in two steps.

First, the image in Figure 36 is eroded, effectively moving the edges of every shape inwards 2 pixels. This should not seriously affect the overall pixel counts because it is the same amount of pixels added when the edges are dilated. When the edges were dilated, 1 extra pixel was included in each shape each time. It was done twice so every shape should have approximately two extra pixels of width in each direction.

Next, any shapes including fewer than 25 pixels are removed. This eliminates the noise, as most of the shapes are fewer than 25 pixels especially after eroding disconnects a few of them.

Finally, the holes are filled much like in the other masks. This preserves some of the pixels that may have been removed from the inside edges of the swellings. Additionally, it creates the final mask to apply to the image. Once it is applied, it results in Figure 37.

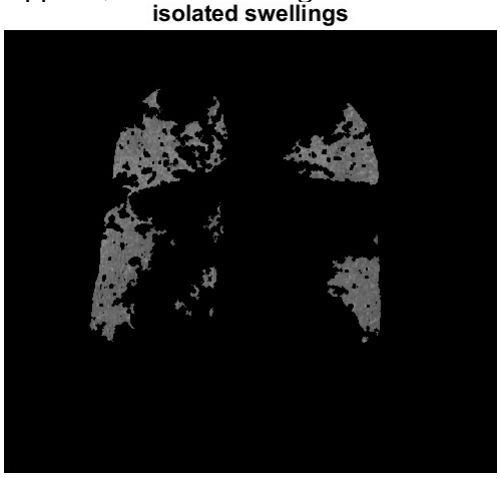

Figure 37

This final image is then converted to binary so that the number of non-black pixels can easily be counted by simply summing the image.

Comparing this binary image seen in Figure 38 with the mask from Mask 1, seen in Figure 32, gives a percentage of the lung area that is covered by swellings/scar tissues.

A few examples are seen below:

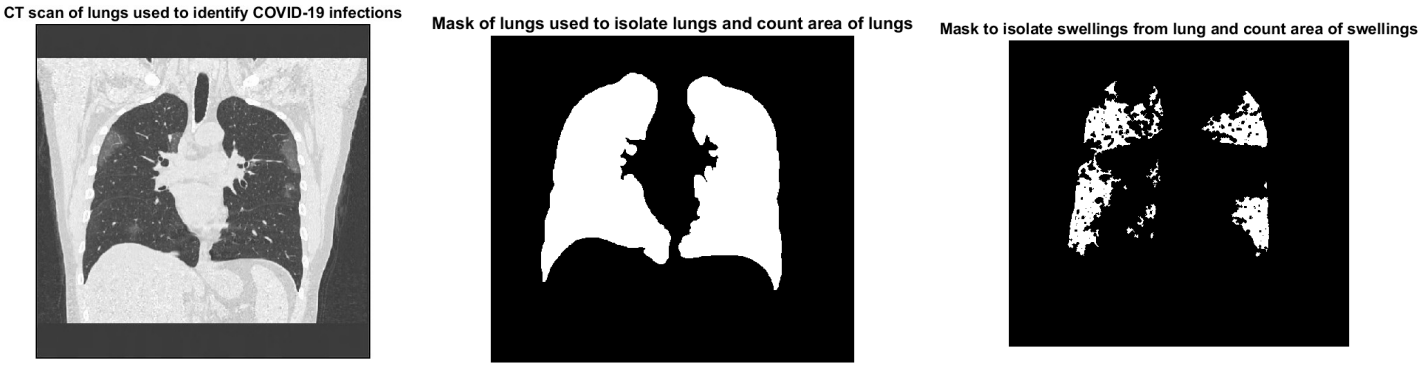

Figure 38
A comparison of CT scan, lung mask, and swelling mask, outputting 9.1268% of lung swollen

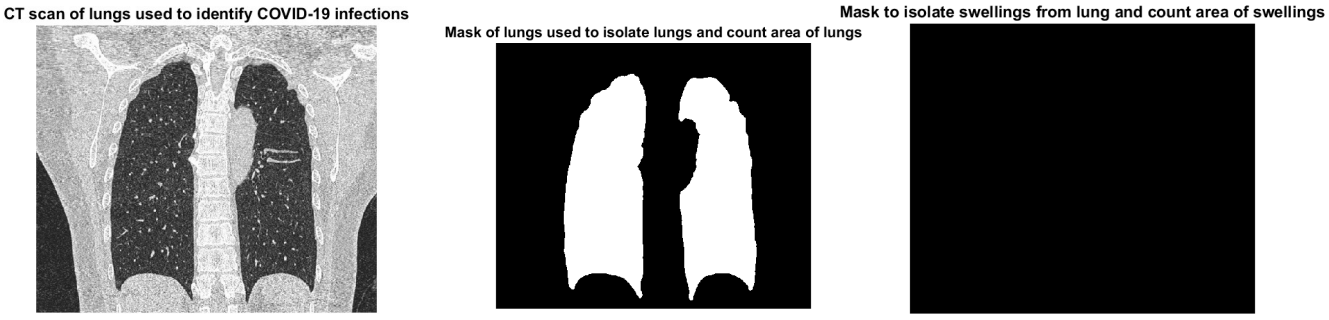

Figure 39
A comparison of CT scan, lung mask, and swelling mask, outputting 0% of lung swollen

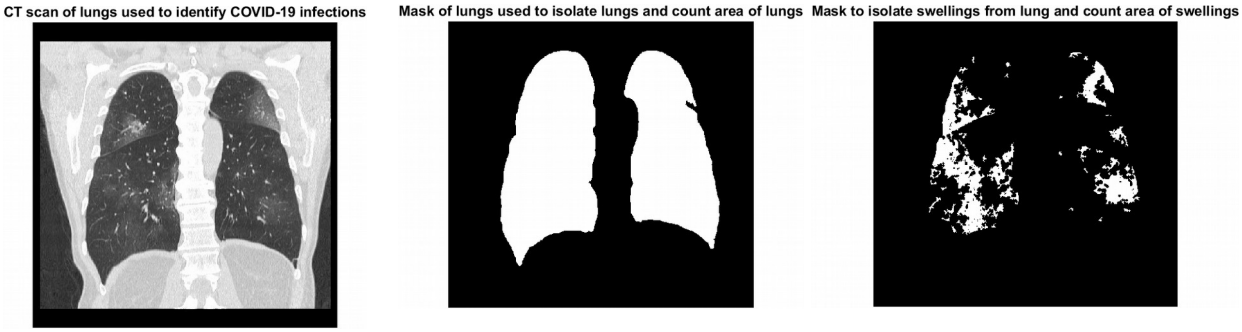

Figure 40
A comparison of CT scan, lung mask, and swelling mask, outputting 20.4403% of lung swollen

Within the code, there are many imshow() statements that are commented-out. These show intermediate steps and can be used to demonstrate the process more in depth if required.

Michael Ungureanu 301338428 mungurea@sfu.ca

Sources used:

"Detecting A Cell Using Image Segmentation" *MATLAB & Simulink Example*, www.mathworks.com/help/images/detecting-a-cell-using-image-segmentation.html.

"Otsu's Method." *Wikipedia*, Wikimedia Foundation, 3 Dec. 2019, en.wikipedia.org/wiki/Otsu's_method.