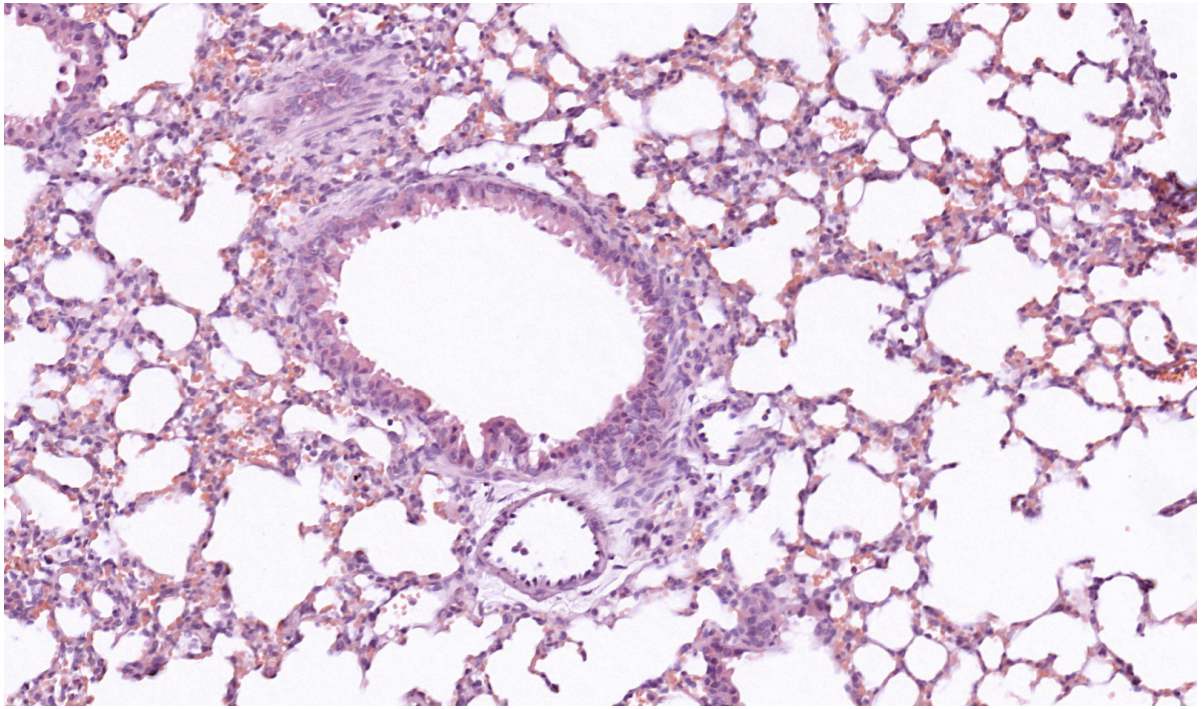
BIOM 5202 / SYSC 5202 - Fall 2024: Assignment #1: Image Enhancement
Name: Amirhossein Ebrahimi (300396640)

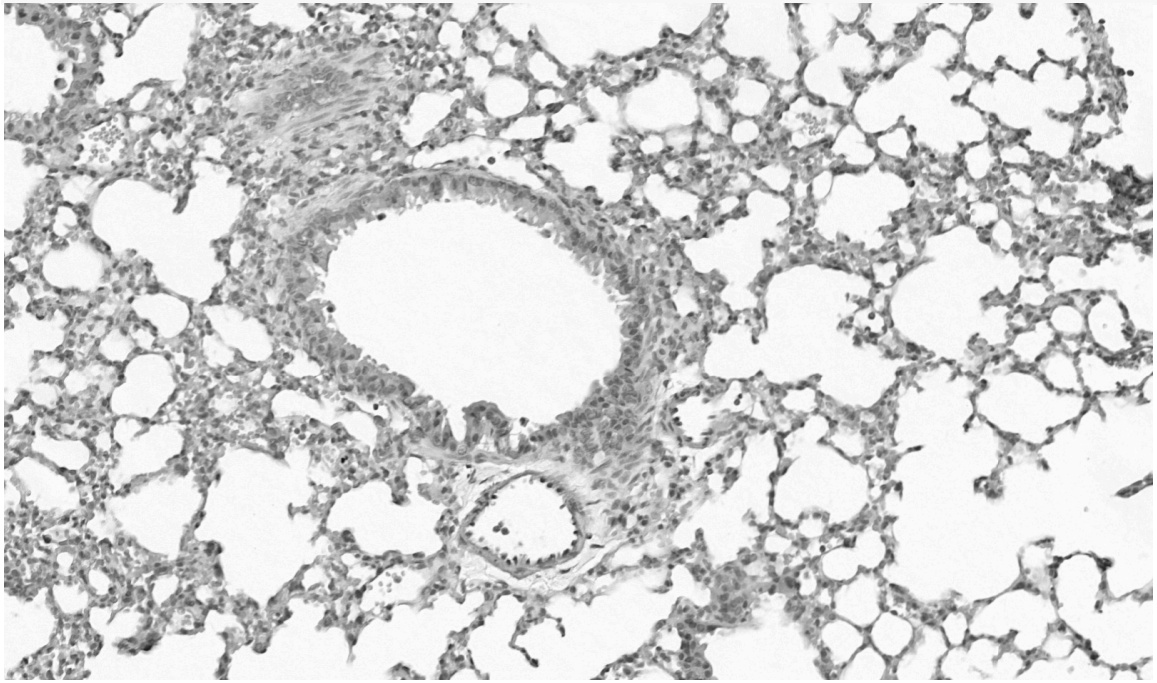
Source code Link: (I only put the main part of the codes, you can access and run everything from scratch in Google Colab). In each section, I have hyperlinked each question title to its relevant part of the code in Colab.

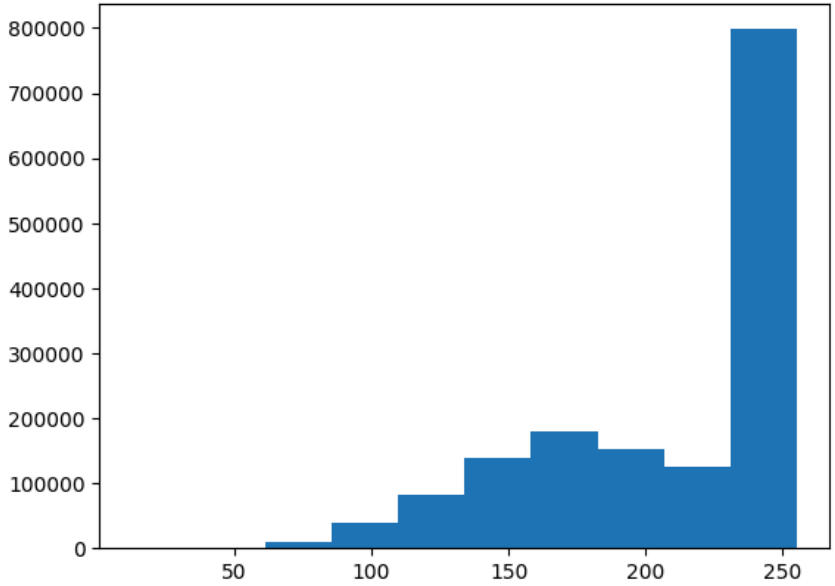
Instructions

- This assignment must be completed individually. It is okay to discuss things with your colleagues, but do not copy any of their answers, codes or notes. Plagiarism is a serious instructional offence that will not be tolerated. Please cite all sources when referring to external resources (e.g. course notes, books, papers, etc).
- For this assignment, it is recommended that you work with Matlab (or equivalent software). Carleton students can install Matlab on their personally owned computers for learning purposes (see http://carleton.ca/ccs/matlab/). Python is also acceptable provided the students submit codes with clear instructions on how to run their codes. All codes must be appropriately commented. Do not simply copy codes (or parts of it) from the internet! All sources must be cited.
- The deliverable for this assignment is a short report that must include: i) the answers to each question; ii) the images and plots generated in each step, carefully captioned; iii) discussion of all information requested in each section; iv) the Matlab codes, properly commented (.m file) and the image(s) you used to generate the results presented (yes! I will run your script and check the results).
- All submitted files must be named as follows: F24_5202_A1_FirstnameLastname.pdf for the report and F24_5202_Image1_FirstnameLastname.jpg, etc).
- Due date and time: October 2nd, 2024 at 23:59.

SECTION 1 - Image Histogram

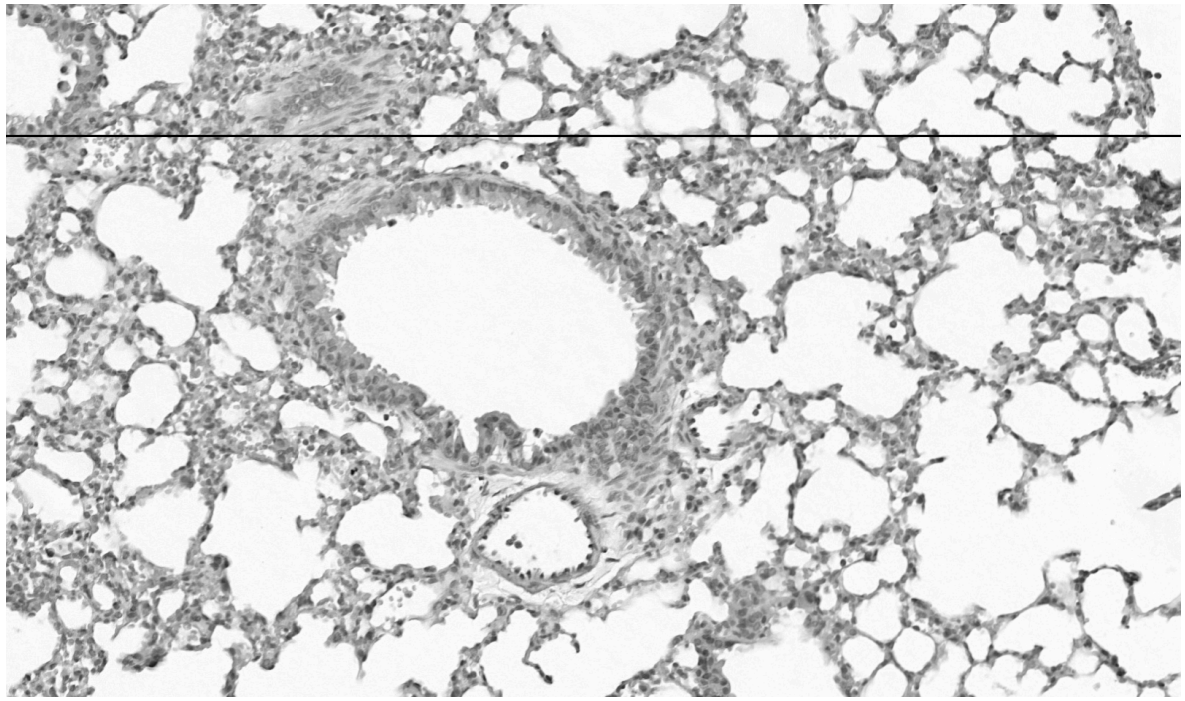
Question:	Answer / Code / Image
1A Choose a 2D gray level medical / microscopy image or a slice of a 3D gray level medical / microscopy image	
1B Considering the underlying medical imaging technology, what does each pixel in the image represent? If the images are in colour, what do the colours represent?	*Each pixel in the image represents a very small area of the tissue sample* ***Purple/Blue**: These colors represent cell nuclei, stained by dyes such as hematoxylin. Hematoxylin binds to the nucleic acids (DNA/RNA), marking the cell's nucleus.* ***Pink/Red**: Often is stained pink or red by eosin. Eosin binds to proteins and other cell structures outside the nucleus.* ***White Spaces**: The white or lighter regions represent spaces in the tissue, such as air spaces (in lung tissue), fat droplets, or areas devoid of cells.*
1C Open it with Matlab/Python and display it in a window using (for example) the “imshow” function	```# Setting the image name and path file_name = "lung.png" # Loading the image using OpenCV img = cv.imread(file_name, 0) # converting the image to numpy array for better access to the row```

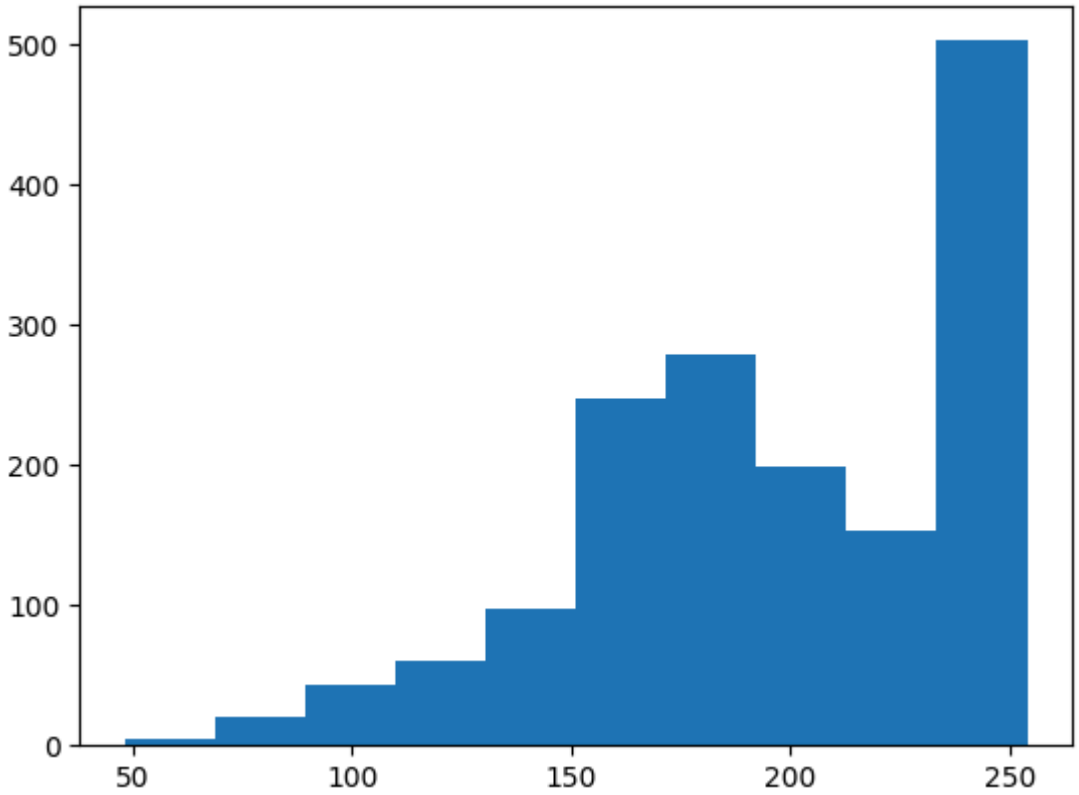
	`` image_array = np.array(img) # Displaying the image (Note only runs in Google Colab, in local system use cv.imshow("Gray Scale", img)) cv2_imshow(img) ``` 
1D Display the image's histogram.	``` plt.hist(image_array.flatten()) ```



1E Choose an interesting row or a column of the image (i.e. containing homogeneous and inhomogeneous areas, soft and sharp edges), and display it in a window using the “plot” function

```
# Select the row of interest (Compared to other parts, it is more darker with more detailed objects)
y = 180
start_point = (0, y)
end_point = (img.shape[1] - 1, y)
# Draw a line on the selected row
cv.line(img, start_point, end_point, color=(0, 255, 0), thickness=2)
# Show the image with the drawn line
cv2_imshow( img)
cv.waitKey(0)
cv.destroyAllWindows()
# Extract the pixel values along the selected row (row 180)
plt.hist(image_array[y,:])
```





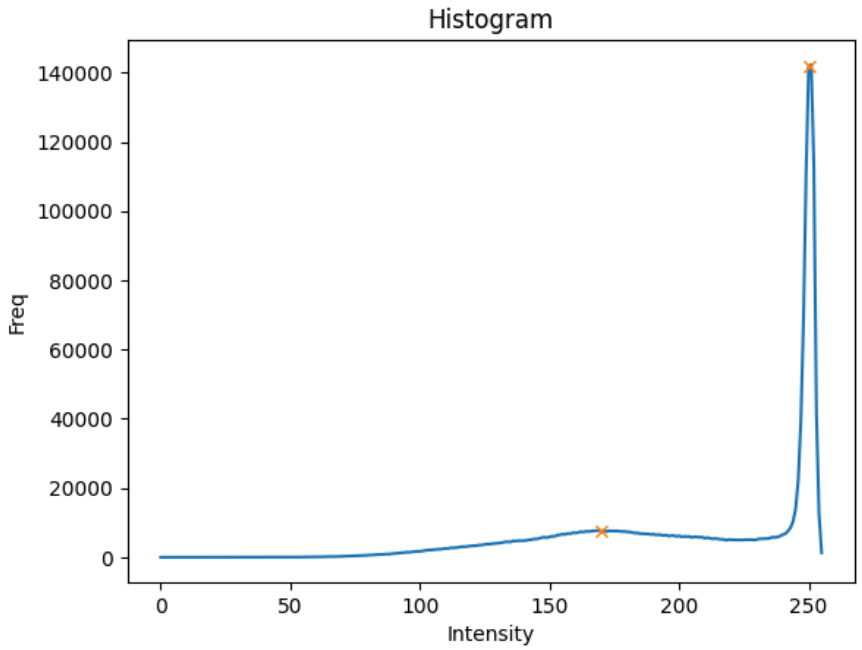
As expected more darker values are presented in histogram.

1F How many modes does the histogram have? Can you identify each mode with specific structures in the image? Explain.

So each peak that can be seen in a histogram is a mode indicating different regions or structures in the image. Visually observing I can only see two mode:

```
# Plot the histogram
hist, bins = np.histogram(image_array.flatten(), bins=256, range=(0, 256))

# Plotting the histogram
plt.plot(hist)
plt.title('Histogram')
plt.xlabel('Intensity')
plt.ylabel('Freq')
```

```
# Mark the peaks on the histogram, I found the index by eyes
plt.plot([250,170], hist[[250,170]], "x")
plt.show()
```



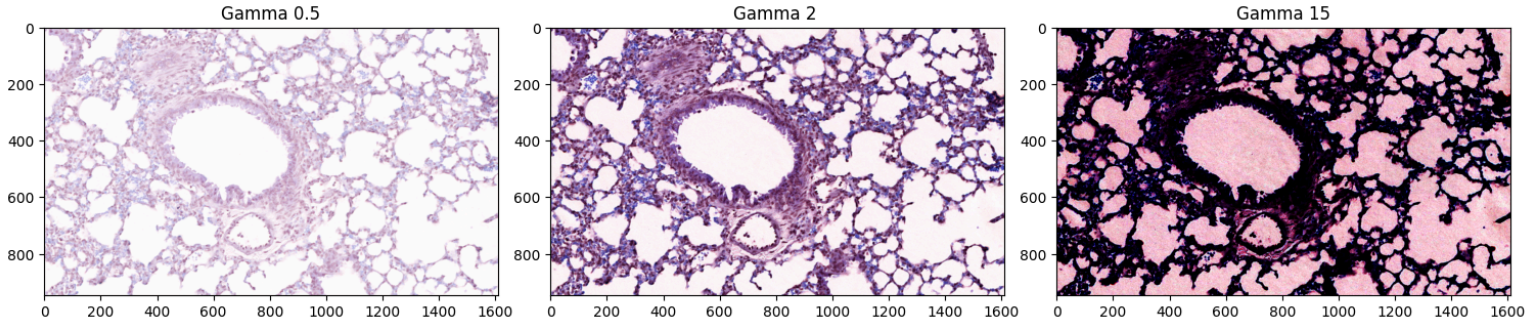
So the one around 250 is a peak or a mode, representing a specific feature in the image that the majority of pixels have a value around 250. That's our white spaces.
The other one between 200 and 150 is tissue structures (nuclei, cytoplasm, etc., which may appear in different shades depending on staining).

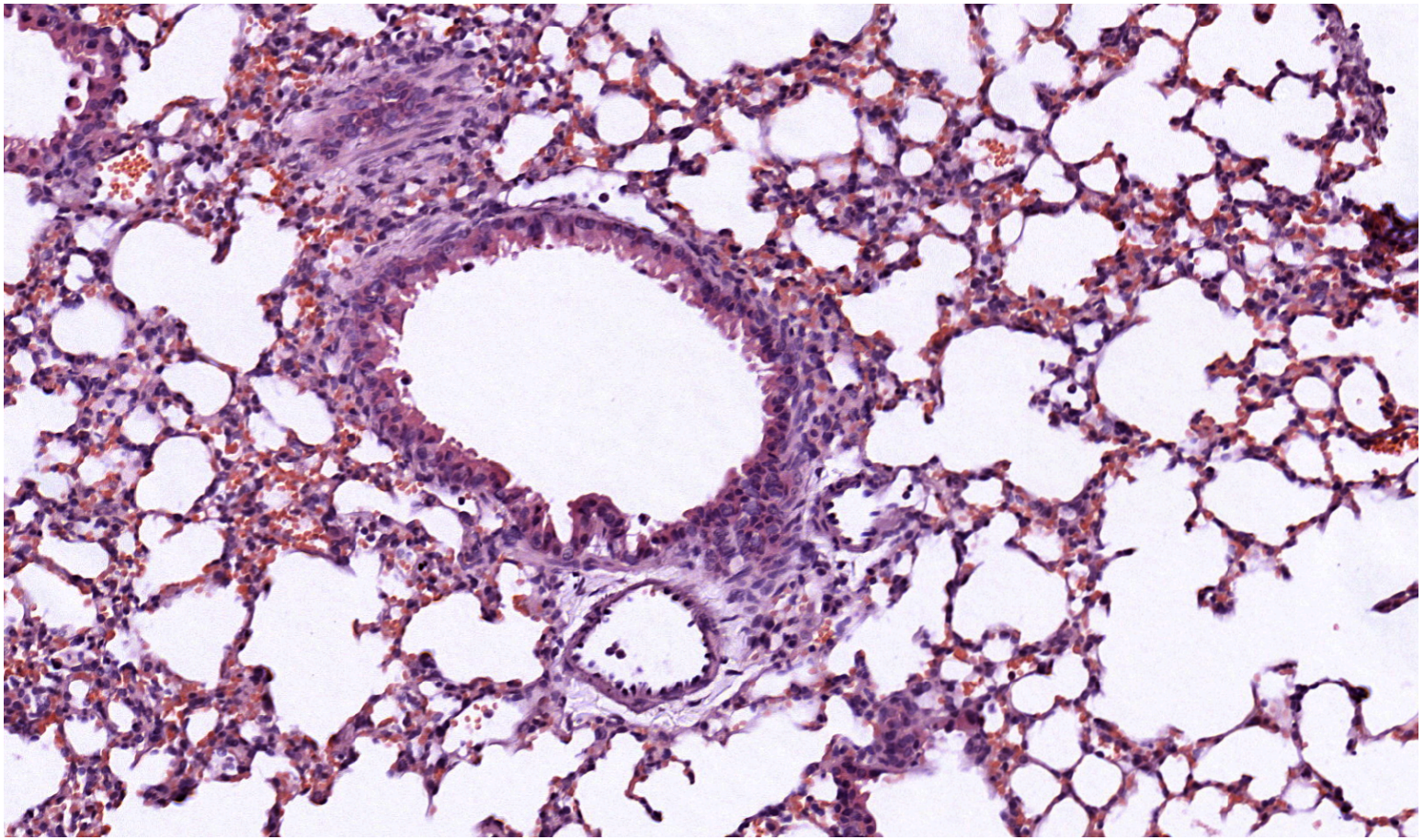
We should've been able to see ***Pink/Red*** *and* ***Purple/Blue.*** However, since it is in grayscale we can not distinguish it in two peaks or modes. **But,** the variance of our peak around 170 is high, illustrating that ***Pink/Red*** *and* ***Purple/Blue*** exist.

1G Try three different levels of gamma correction. Show the resulting images and the code to generate them.

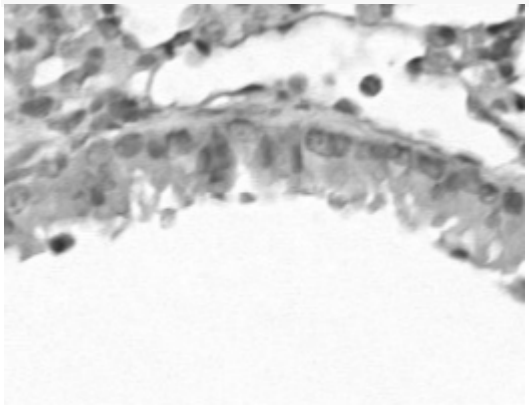
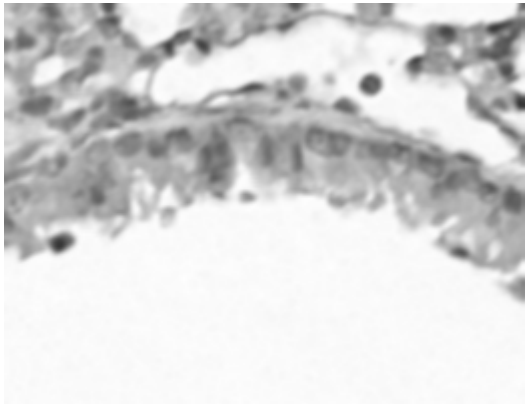
```
def apply_gamma_correction(img, gamma):
 lookUpTable = np.empty((1,256), np.uint8)
 for i in range(256):
     lookUpTable[0,i] = np.clip(pow(i / 255.0, gamma) * 255.0, 0, 255)
 return cv.LUT(img, lookUpTable)
```

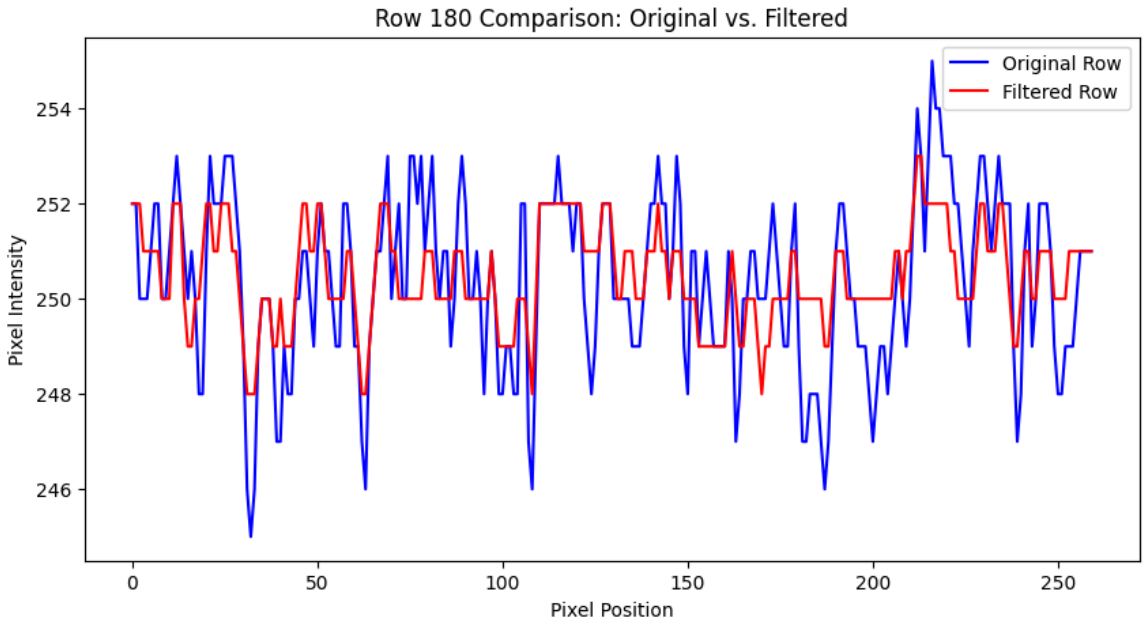
```
img_gamma_05 = apply_gamma_correction(img, gamma=0.5)
img_gamma_2 = apply_gamma_correction(img, gamma=2)
img_gamma_15 = apply_gamma_correction(img, gamma=15)

plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
plt.imshow(img_gamma_05)
plt.title("Gamma 0.5")

plt.subplot(1, 3, 2)
plt.imshow(img_gamma_2)
plt.title("Gamma 2 ")

plt.subplot(1, 3, 3)
plt.imshow(img_gamma_15)
plt.title("Gamma 15")

plt.tight_layout()
plt.show()
```

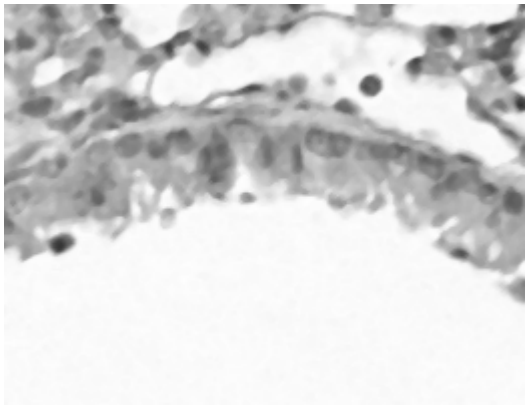


1H Which level of gamma correction do you consider best? Why? What does this say about the underlying imaging technology	 Choosing the optimal gamma value is highly application-specific. In my case, while working on lung injury research, I selected a gamma threshold of 2. This setting significantly enhances contrast, allowing for better differentiation of objects within the tissue. It also refines colour distinctions, particularly between red and blue stains, which is crucial for accurately identifying cellular structures. This choice highlights how gamma correction helps make important details in medical imaging clearer, which can improve diagnostic accuracy. I'm actually surprised how this assignment helped me to enhance my basic computer vision skills. This gamma correction helped me much more than I expected. I didn't check gamma correction before taking this course. It really helped me in detecting the number of cells.

SECTION 2 - Smoothing - Linear and Nonlinear Filtering

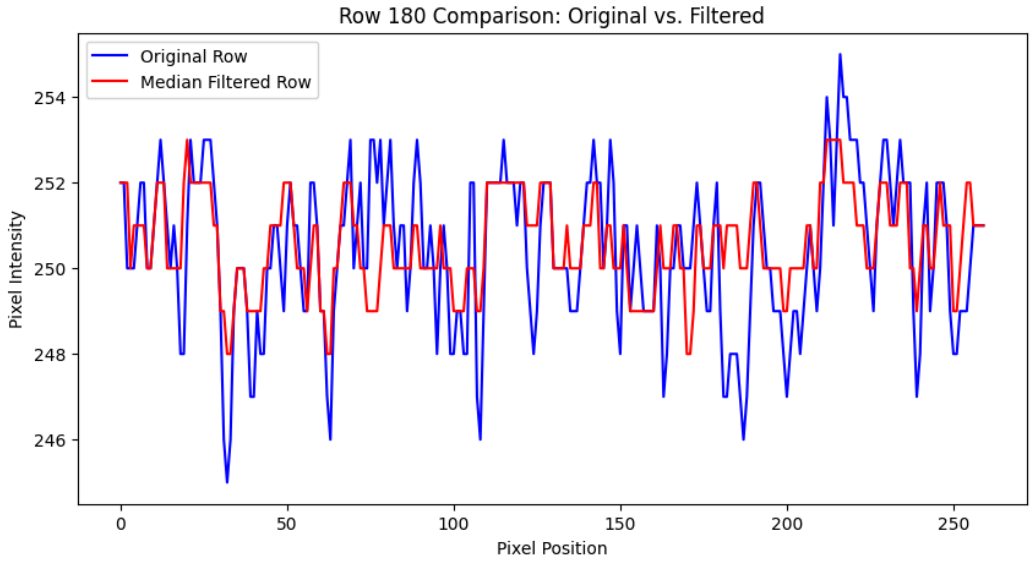
Question:	Answer / Code / Image
2A Select an “interesting” 50×50 pixel region of the image from Section 1. It should contain contrasts and edges and include the row from 1E.. Show the region	The 50x50 in my image was too small, so Instead I’m operating on 200x200. `img_200x200 = img[180:380,500:760]` `plt.imshow(img_200x200)` 
2B Apply a 3x3 averaging filter to the original image. Display the filtered image.	`kernel = np.ones((3,3),np.float32)/9` `avg_img = cv.filter2D(img_200x200,-1,kernel)` `cv2_imshow(avg_img` 
2C Extract the same row or column as you did before from the original image, and display it using the “hold”	`original_row = img_200x200[y, :]` `filtered_row = avg_img[y, :]` `plt.figure(figsize=(10, 5))`

and “plot” functions on the same graph as the original row or column but with a distinct colour.	`plt.plot(original_row, label='Original Row', color='blue')` `plt.plot(filtered_row, label='Filtered Row', color='red')` `plt.show()`  *We can easily see how it’s more smoother (Red line)*
2D Apply a 3x3 median filter to the original image. Display the filtered image.	`med_img = cv.medianBlur(img_200x200,3)` `plt.imshow(med_img)`

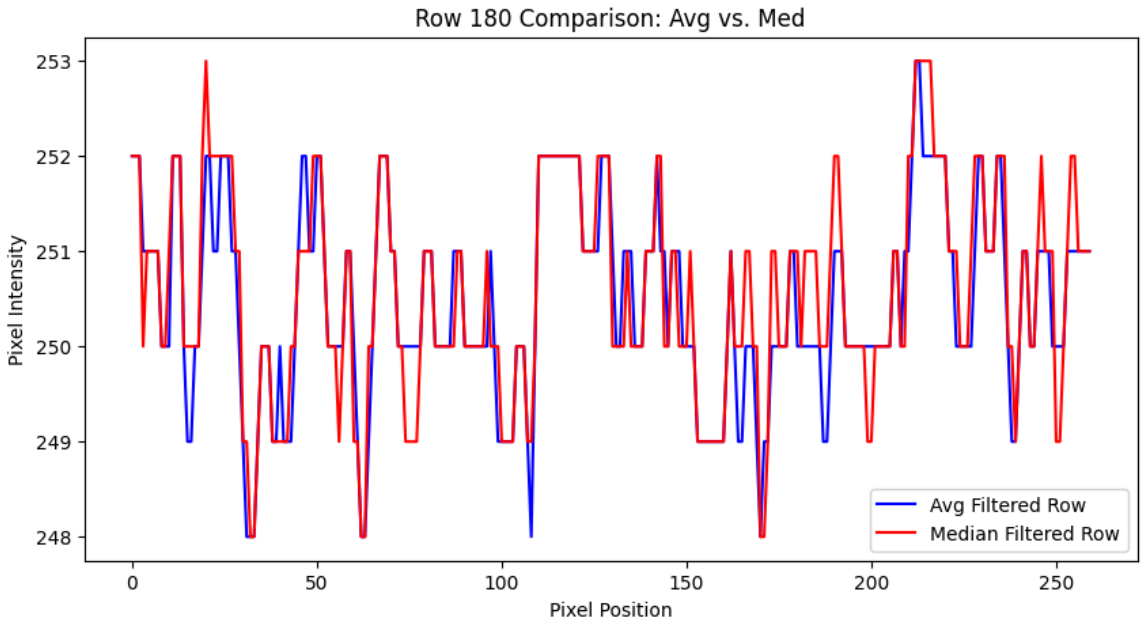


2E Extract the same row or column as you did before from the original and averaged images, and display it using the “hold” and “plot” functions on the same graph as the original row or column but with a distinct colour.

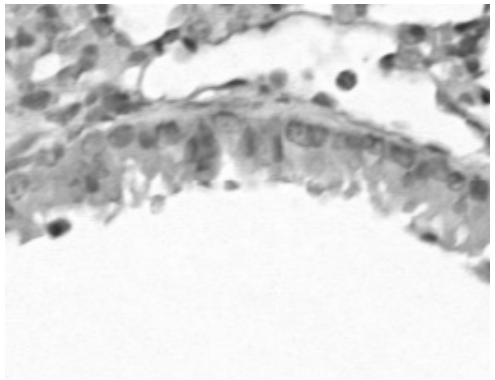
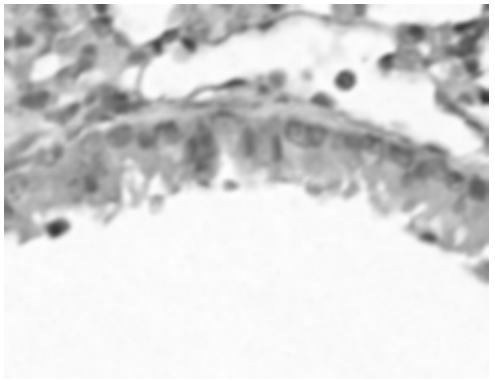
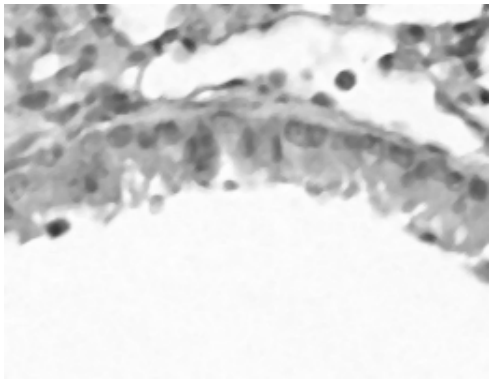
```
med_filtered_row = med_img[y, :]

plt.figure(figsize=(10, 5))
plt.plot(original_row, label='Original Row', color='blue')
plt.plot(med_filtered_row, label='Median Filtered Row', color='red')
plt.show()
```



Comparison between Avg and Med filter:



2F Based on both the image and the extracted row or column, compare the original image, the average-filtered image and the median-filtered image. What happens to homogeneous areas of the image? What happens to noisy areas of the image? What happens to edges? Briefly explain.

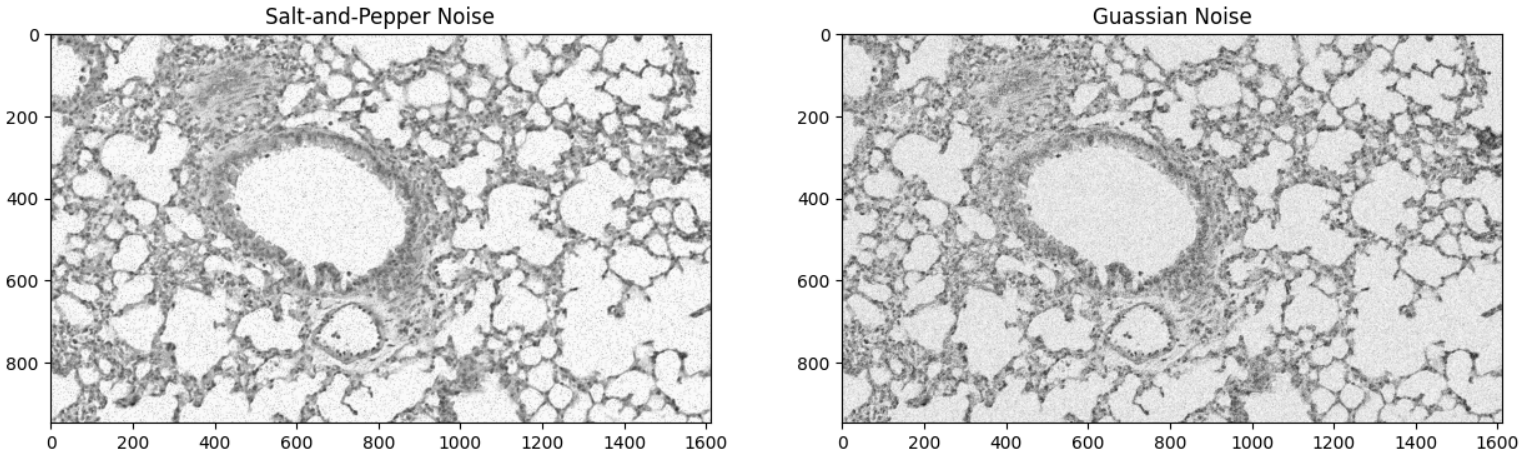
Main	Average	Median
		

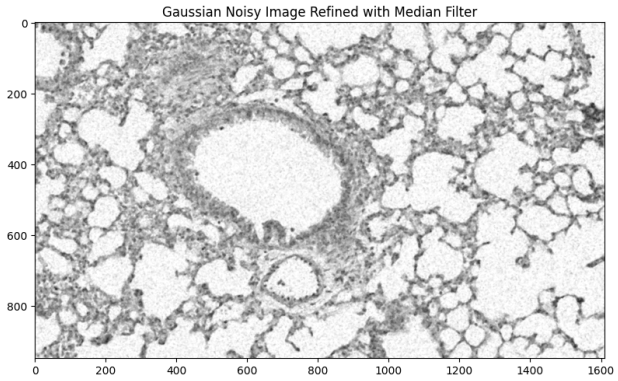
Average-Filtered Image:

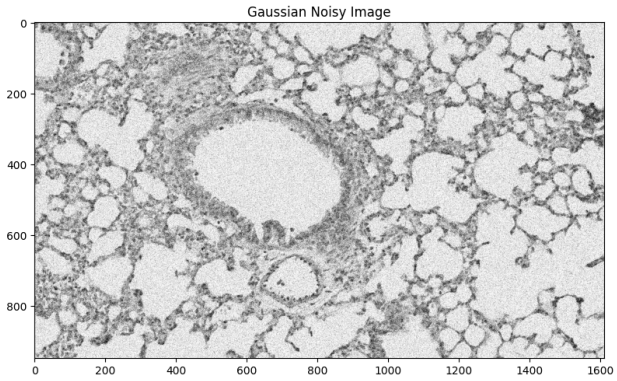
	The average filter smooths homogeneous regions by averaging nearby pixels. This can reduce noise in these regions but can also make them blurred. Also *for the noisy area*, it can reduce noise as well since it is averaging the intensity values, but we're making the other clear objects blur. Finally, since edges have distinguishable contrasts, by averaging we are actually making the edges/contrasts less distinguishable. **Median-Filtered Image:** On the other hand, the median filter keeps homogeneous areas very well since it replaces each pixel with the median of its neighbours. So it does not smooth the pixel intensity as much as the average filter. For *the noisy area*, Median filters are great for the salt_and_peper noise because these noises have the highest or the lowest intensity, and since we're doing the median, it ignores those noises without blurring too much. Finally for the edges, with the same explanation, since we're not averaging, we are not distinguishing contrasts between two regions too much, so it acts better than avg filter in preserving edges. **Overall:** Average filter is good for smoothing homogeneous areas but blurring other areas as well. Median filter is good for removing noise and preserving edges.

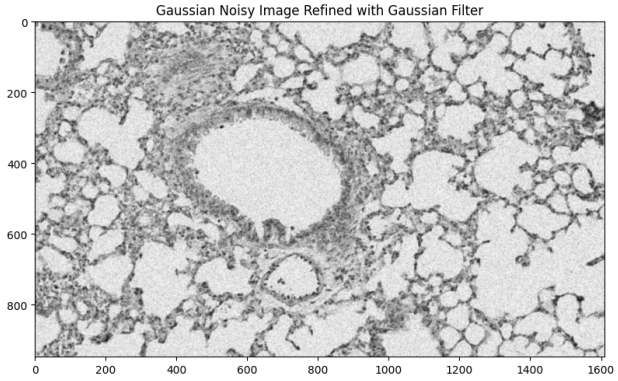
SECTION 3 - Image Noise

Question:	**Answer / Code / Image**
3A Use the Matlab command "imnoise" to generate two noisy versions of the image, one with salt and pepper noise and the other with Gaussian white noise.	I'm using `skimage` in python `from skimage import util` `# Add salt-and-pepper noise` `salt_pepper_noise_img = util.random_noise(img, mode='s&p', amount=0.05)` `# Add gaussian noise` `gaussian_noise_img = util.random_noise(img, mode='gaussian', var=0.05)` `plt.figure(figsize=(15, 5))` `plt.subplot(1, 2, 1)`

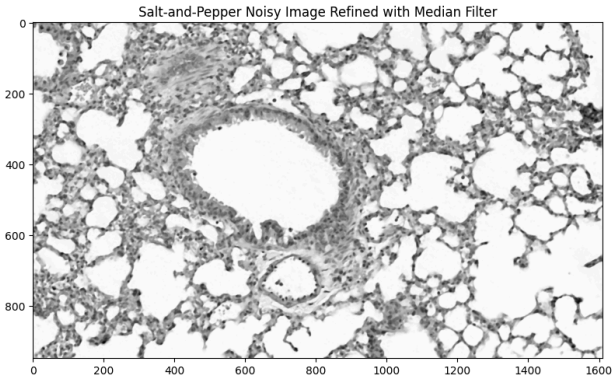
	`` plt.imshow(salt_pepper_noise_img, cmap='gray') plt.title("Salt-and-Pepper Noise") plt.subplot(1, 2, 2) plt.imshow(gaussian_noise_img, cmap='gray') plt.title("Guassian Noise") plt.show() ``` 
3B Apply a 5×5 Gaussian and median filter using the same mask size to the noisy images.	``` guas_noise_med_ref = cv.medianBlur(gaussian_noise_img_uint8,5) guas_noise_guas_ref = cv.GaussianBlur(np.array(gaussian_noise_img_uint8),(5,5),0) ```

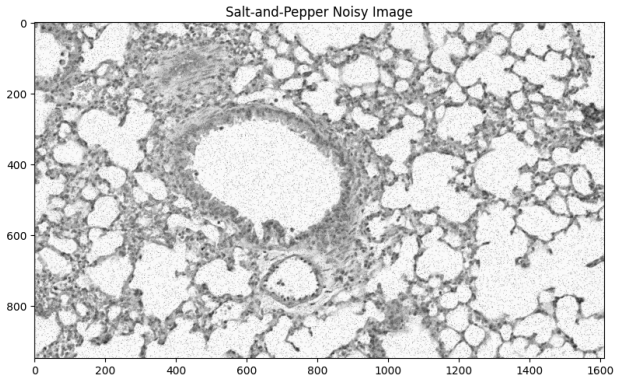


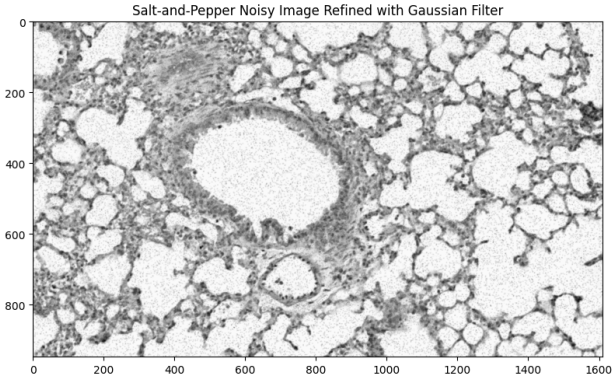





```
sp_noise_med_ref = cv.medianBlur(salt_pepper_noise_img_uint8,5)


sp_noise_guas_ref = cv.GaussianBlur(np.array(salt_pepper_noise_img_uint8),(5,5),0)
```







It can be observed that how median filter completely removed salt and pepper noise.

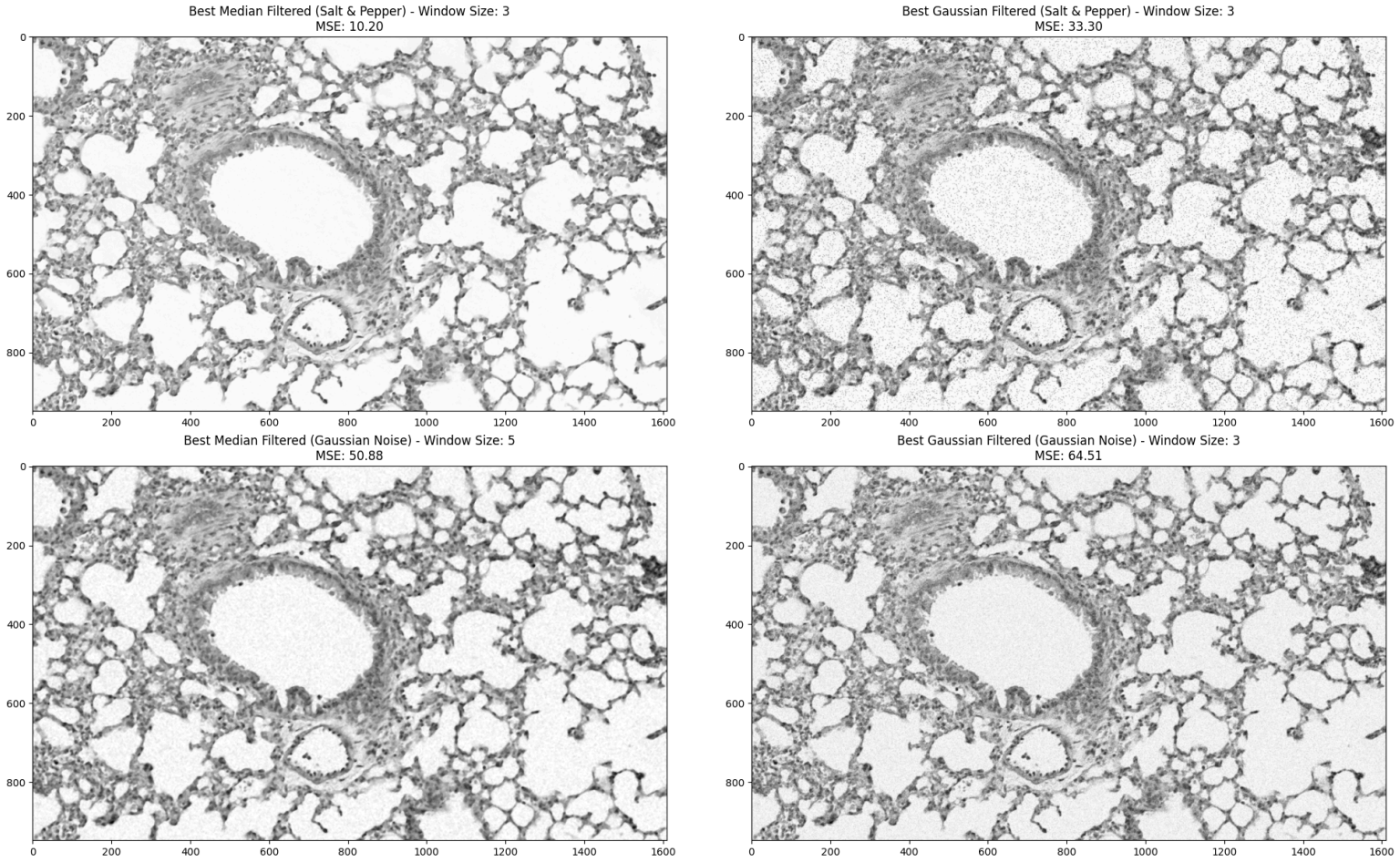
3C: Compare the performance of the filters and windows sizes in terms of: mean squared error and general visual properties.

[For the sake of place, I have not put the code here]

I'm applying 4 different window sizes and calculated the MSE for each and reported here, but only plotted the best ones.

```
Window Size: 3
  MSE for Median Filter on Salt-and-Pepper Noise: 10.20
  MSE for Gaussian Filter on Salt-and-Pepper Noise: 33.30
  MSE for Median Filter on Gaussian Noise: 57.36
  MSE for Gaussian Filter on Gaussian Noise: 64.51
Window Size: 5
  MSE for Median Filter on Salt-and-Pepper Noise: 24.95
  MSE for Gaussian Filter on Salt-and-Pepper Noise: 44.27
  MSE for Median Filter on Gaussian Noise: 50.88
  MSE for Gaussian Filter on Gaussian Noise: 65.16
Window Size: 7
```

	MSE for Median Filter on Salt-and-Pepper Noise: 35.54 MSE for Gaussian Filter on Salt-and-Pepper Noise: 57.50 MSE for Median Filter on Gaussian Noise: 52.91 MSE for Gaussian Filter on Gaussian Noise: 70.36 Window Size: 9 MSE for Median Filter on Salt-and-Pepper Noise: 42.18 MSE for Gaussian Filter on Salt-and-Pepper Noise: 63.71 MSE for Median Filter on Gaussian Noise: 55.91 MSE for Gaussian Filter on Gaussian Noise: 74.44 It can be observed that Median Filter did an amazing job on salt and pepper noise. I already described why, but since the median filter takes the med, highest and lowest noises will be discarded. For the Gaussian Noise, It is observable that the Gaussian Filter did a better job. The reason is obvious: we applied gaussian noise and then denosied it with a gaussian filter.

SECTION 4 - Edge Detection

Question:	**Answer / Code / Image**
4A Show the 3×3 convolution kernels for: i) Prewitt, ii) Sobel, ii) Gaussian, iii) Laplace filters	1. Prewitt: `prewitt_kernel_x = [1, 0, -1]` `[1, 0, -1]` `[1, 0, -1]`

```
prewitt_kernel_y = [1,   1,  1]
                   [0,   0,  0]
                   [-1, -1, -1]
```

2. Sobel:

```
sobel_kernel_x =   [1, 0, -1]
                   [2, 0, -2]
                   [1, 0, -1]

sobel_kernel_y =   [1,   2,  1]
                   [0,   0,  0]
                   [-1, -2, -1]
```

3. Guassain:

```
guassian_kernel =   [0.0625,   0.125,  0.0625]
                    [0.125,    0.25,   0.125]
                    [0.0625,   0.125,  0.0625]
```

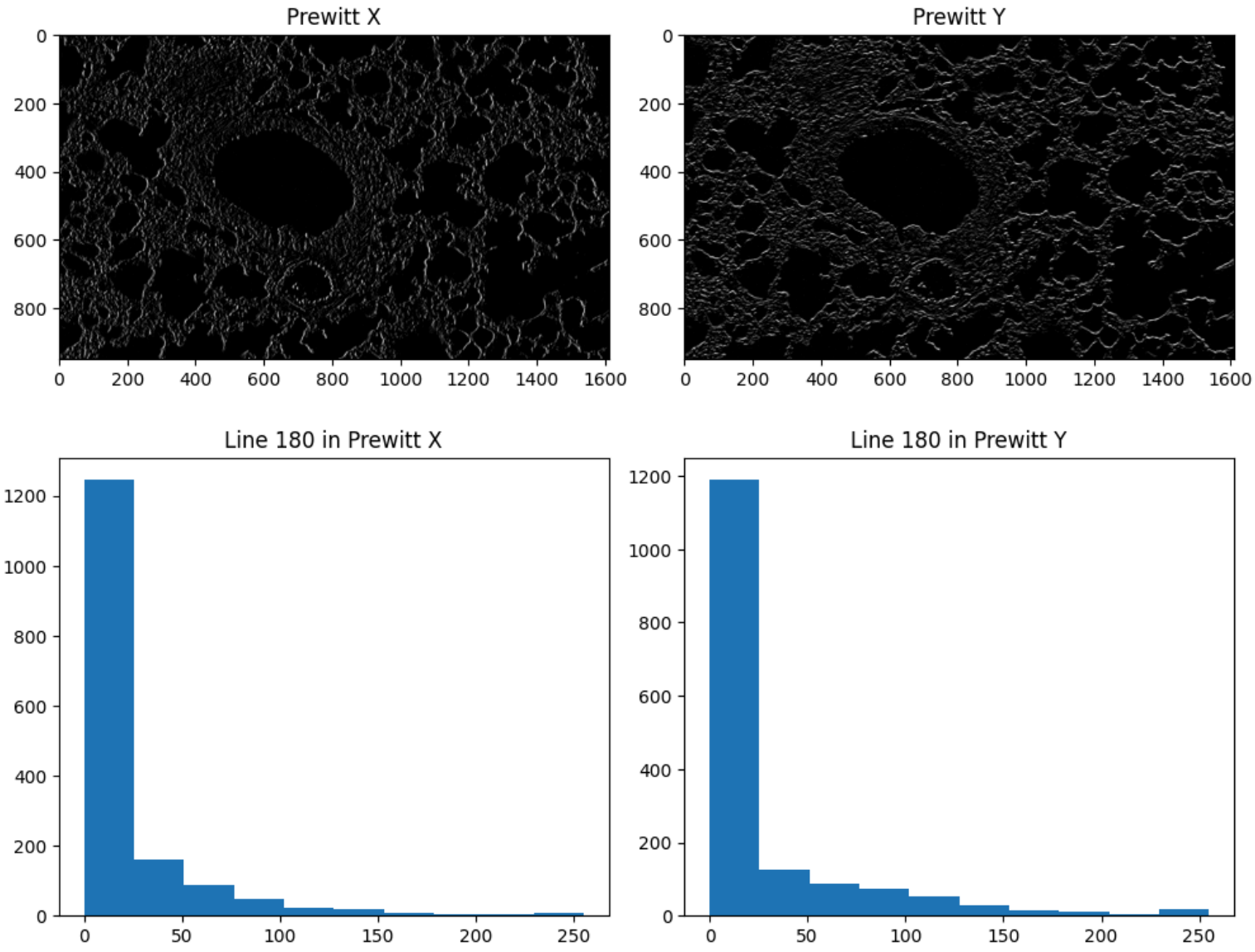
4. Laplace:

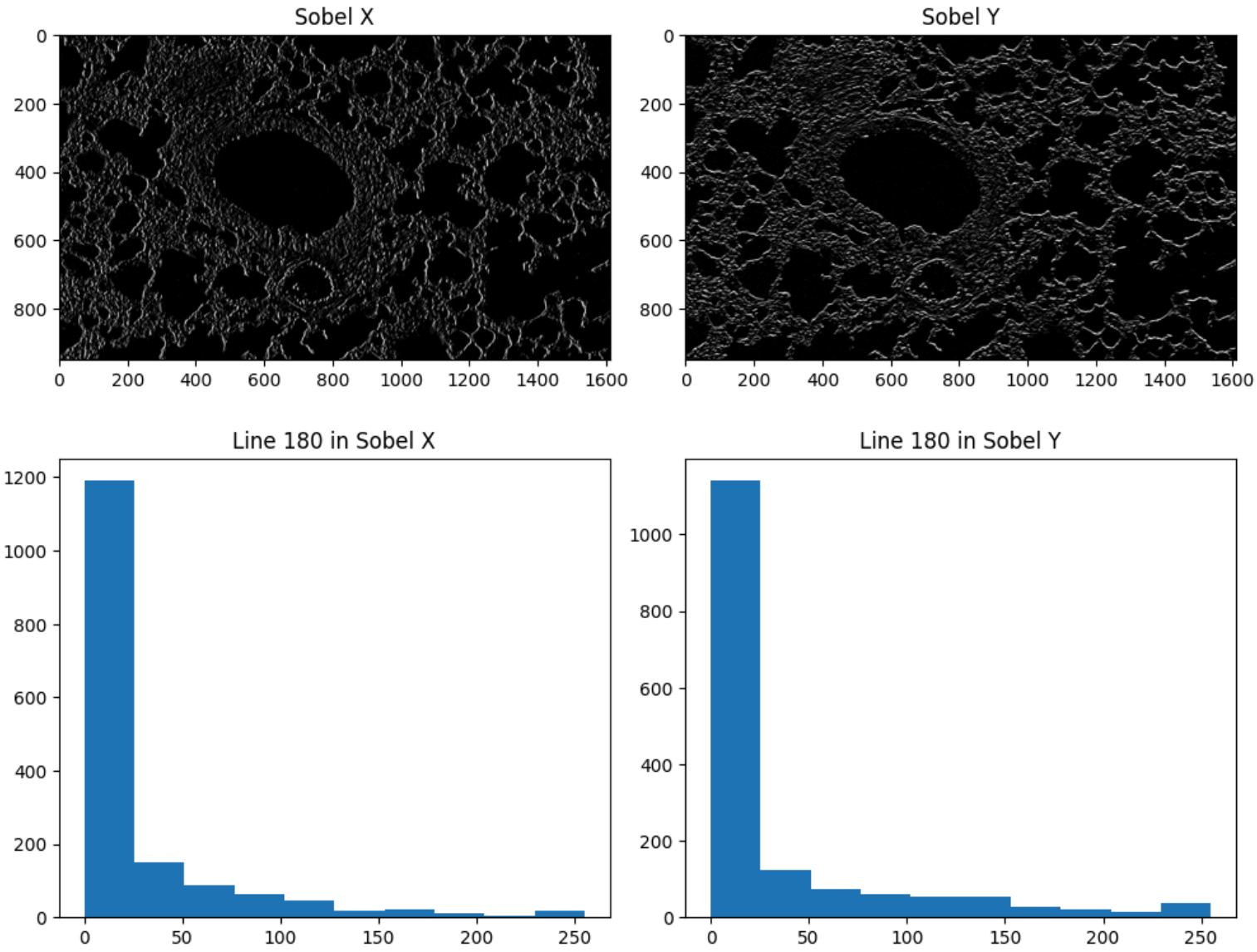
```
laplace_kernel =   [0,   -1,  0]
                   [-1,   4, -1]
                   [0,   -1,  0]
```

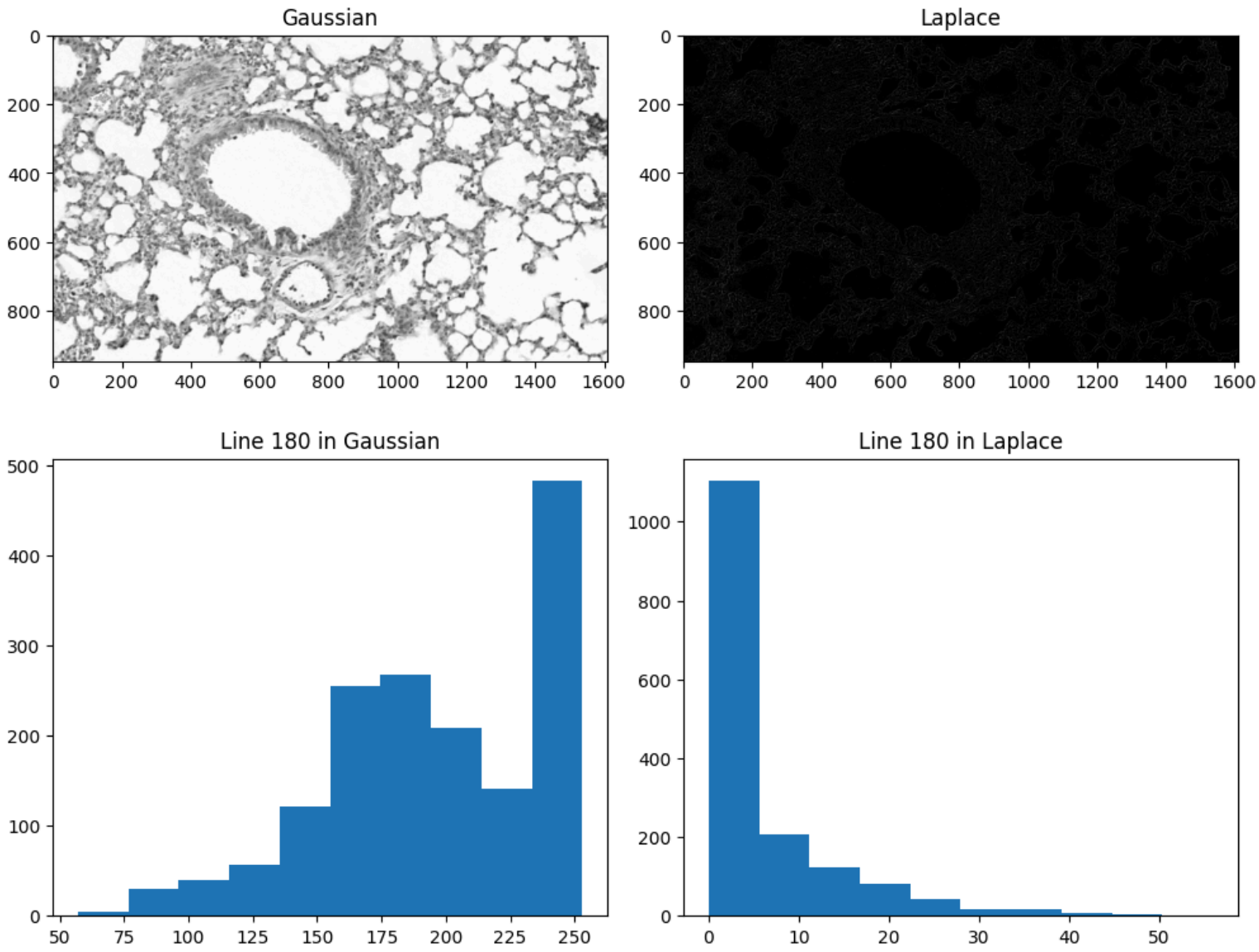
4B Apply each filter to the image. Show the image and a line through the image.

```
prewitt_kernel_x = np.array([[1, 0, -1],
                             [1, 0, -1],
                             [1, 0, -1]])

prewitt_kernel_y = np.array([[1, 1, 1],
                             [0, 0, 0],
                             [-1, -1, -1]])

sobel_kernel_x = np.array([[1, 0, -1],
                           [2, 0, -2],
                           [1, 0, -1]])
```

```
sobel_kernel_y = np.array([[1, 2, 1],
                           [0, 0, 0],
                           [-1, -2, -1]])

gaussian_kernel = cv.getGaussianKernel(3, 0)
gaussian_kernel = gaussian_kernel * gaussian_kernel.T

laplace_kernel = np.array([[0, -1, 0],
                           [-1, 4, -1],
                           [0, -1, 0]])

# Apply the filters
prewitt_x = cv.filter2D(img, -1, prewitt_kernel_x)
prewitt_y = cv.filter2D(img, -1, prewitt_kernel_y)
sobel_x = cv.filter2D(img, -1, sobel_kernel_x)
sobel_y = cv.filter2D(img, -1, sobel_kernel_y)
gaussian = cv.filter2D(img, -1, gaussian_kernel)
laplace = cv.filter2D(img, -1, laplace_kernel)
```


Prewitt X
Prewitt Y
Line 180 in Prewitt X
Line 180 in Prewitt Y


Sobel X
Sobel Y
Line 180 in Sobel X
Line 180 in Sobel Y

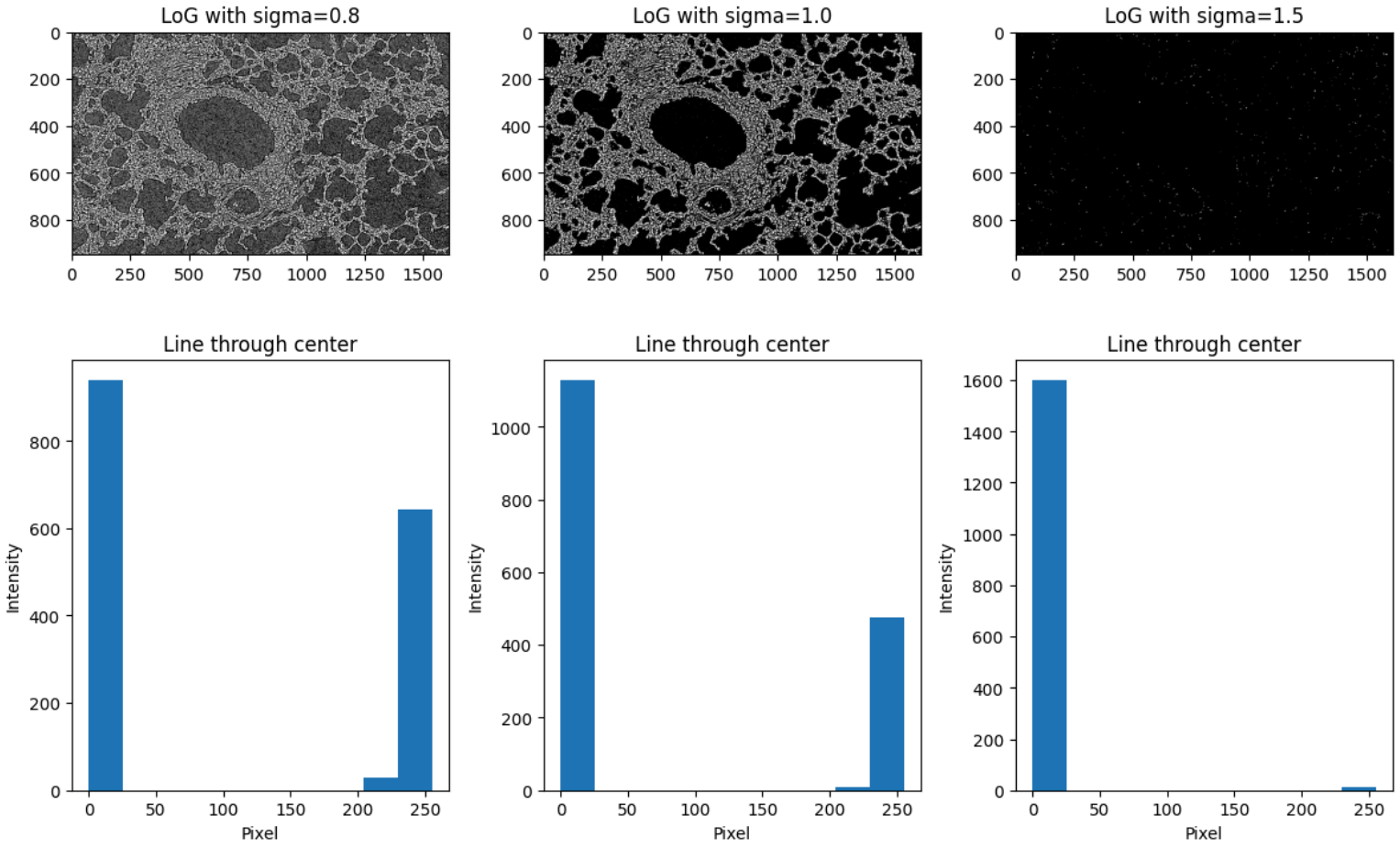


4C Calculate a 7×7 Laplacian of Gaussian kernel (LoG) (you chose the parameters)

```
def laplacian_of_gaussian(size, sigma):
    ax = np.linspace(-(size // 2), size // 2, size)
    xx, yy = np.meshgrid(ax, ax)
    norm = (xx**2 + yy**2) / (2.0 * sigma**2)
    kernel = (1.0 / (np.pi * sigma**4)) * (1.0 - norm) * np.exp(-norm)
```

```
    return -1*kernel
```

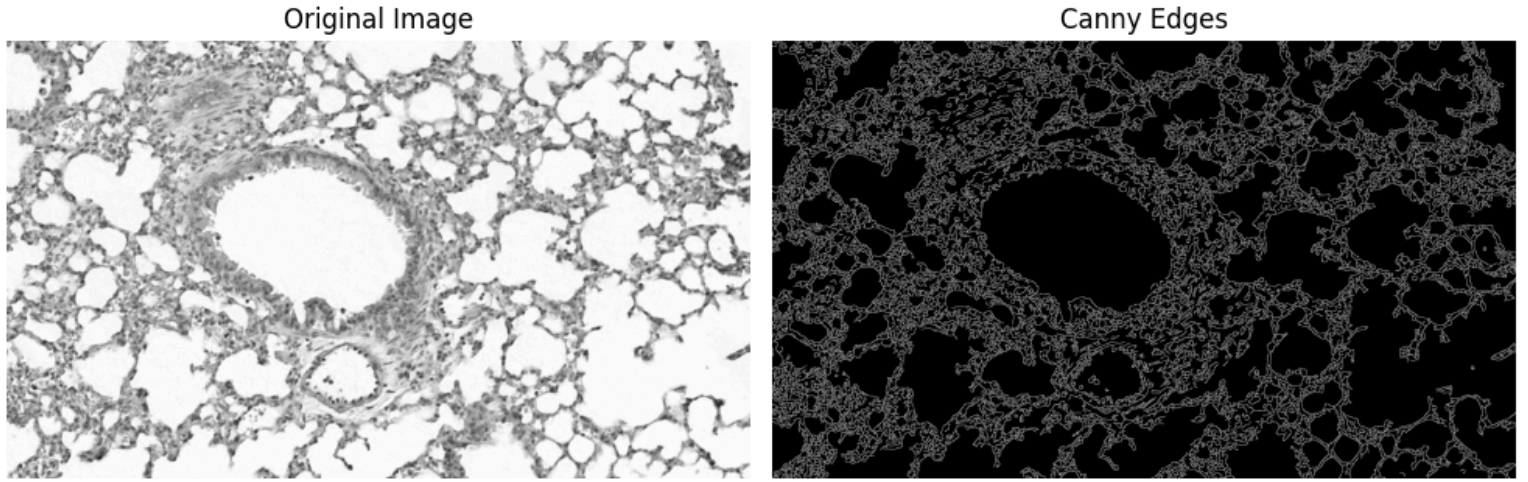
4D Apply the filter to the image for three different parameter values. Show the image and a line through the image.

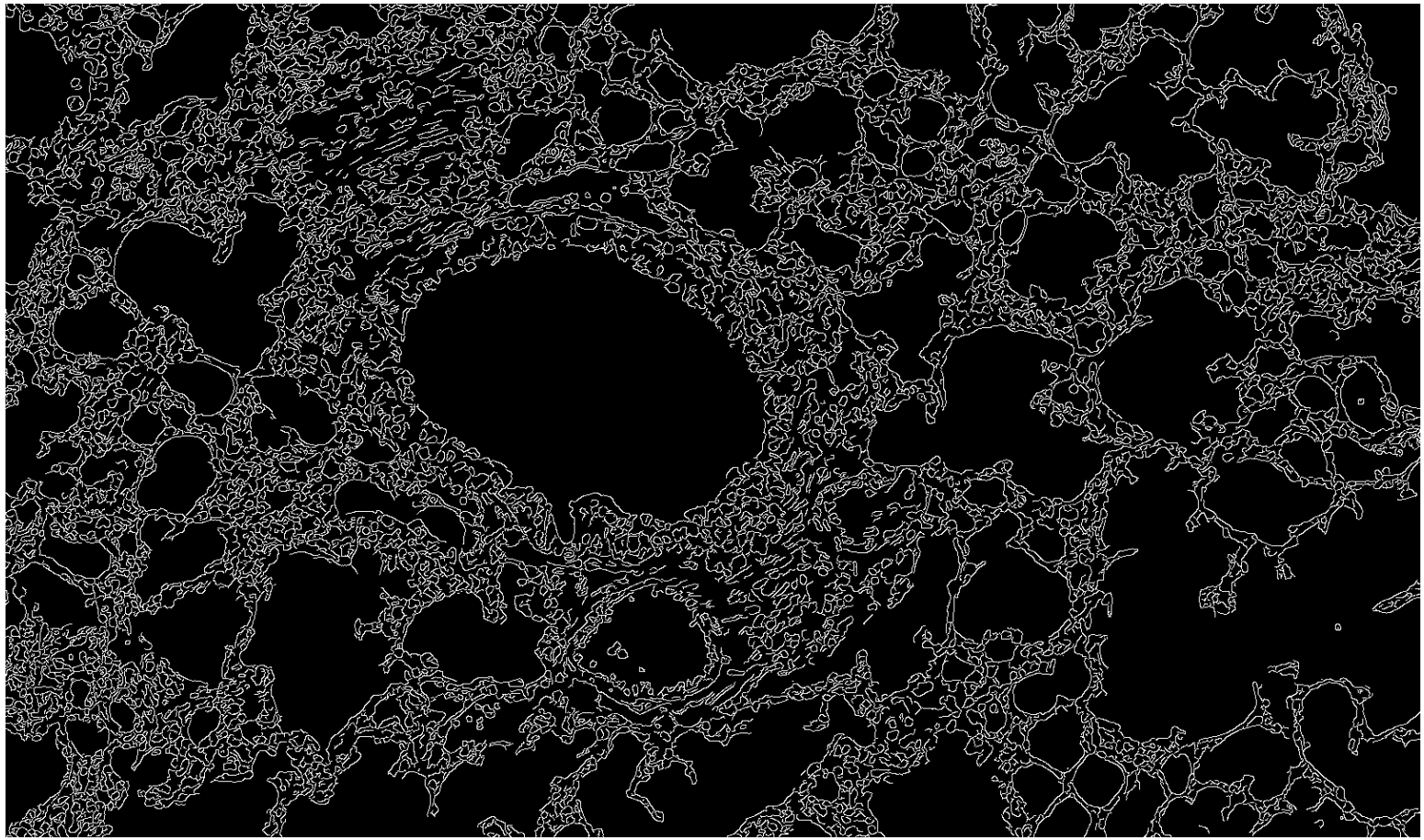


4E Apply the Canny edge detection algorithm (use matlab “edge(IMG, ‘canny’)” or similar in python)

```
canny_edges = cv.Canny(img, 100, 200)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.axis('off')
```

```
plt.subplot(1, 2, 2)
plt.imshow(canny_edges, cmap='gray')
plt.title('Canny Edges')
plt.axis('off')

plt.tight_layout()
plt.show()
```



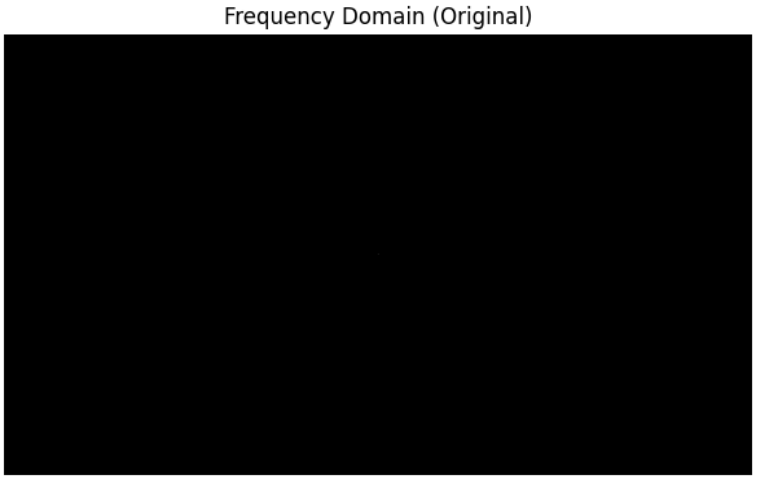
4F: Compare the performance of the filters and in terms of their ability to detect edges in the images. Which do you think worked best on your image?	 Since it's application-specific, I believe the Canny edge performed best for my needs. I don't need to distinguish between horizontal and vertical edges; instead, I require an algorithm that detects edges in all directions effectively, which is crucial for cell detection.

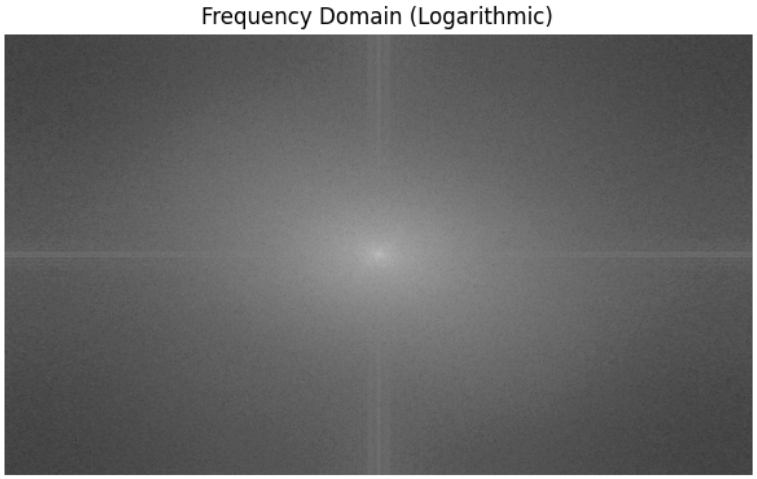
SECTION 5 - Frequency Domain

Question:	**Answer / Code / Image**
5A Show the Frequency Domain representation of your image with: i) original	`f_transform = np.fft.fft2(img)` `f_shift = np.fft.fftshift(f_transform)`

greyscale values, ii) logarithmic greyscale values	

```
# Magnitude spectrum with original grayscale values
magnitude_spectrum = np.abs(f_shift)

# Magnitude spectrum with logarithmic grayscale values
magnitude_spectrum_log = np.log(1 + magnitude_spectrum)

# Plot the results
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.imshow(magnitude_spectrum, cmap='gray')
plt.title('Frequency Domain (Original)')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(magnitude_spectrum_log, cmap='gray')
plt.title('Frequency Domain (Logarithmic)')
plt.axis('off')

plt.tight_layout()
plt.show()
```

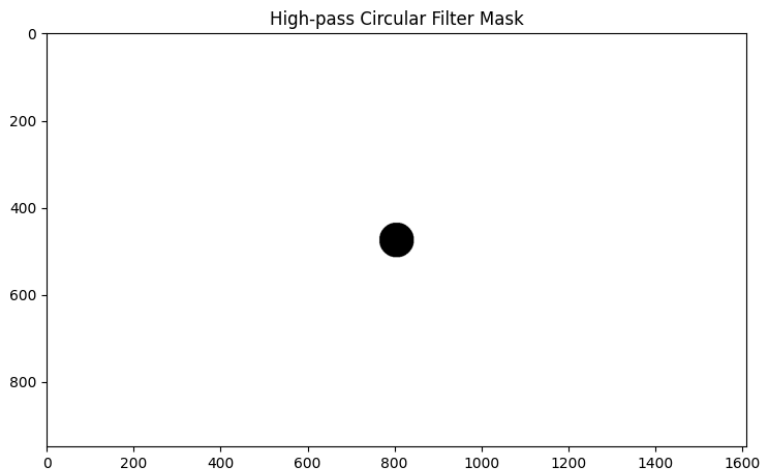
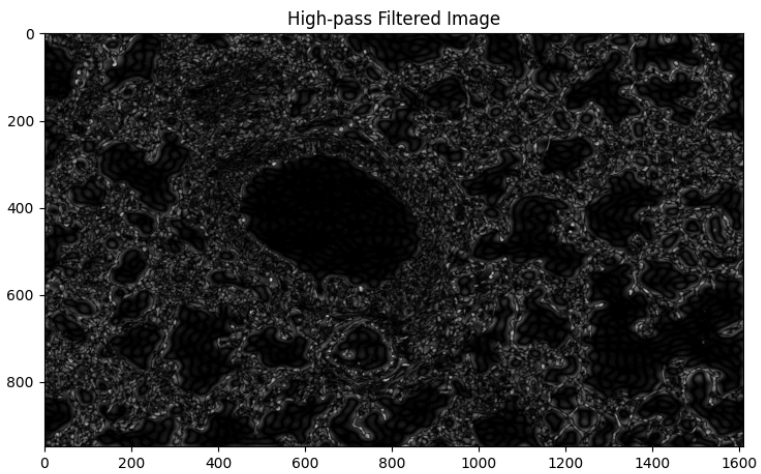
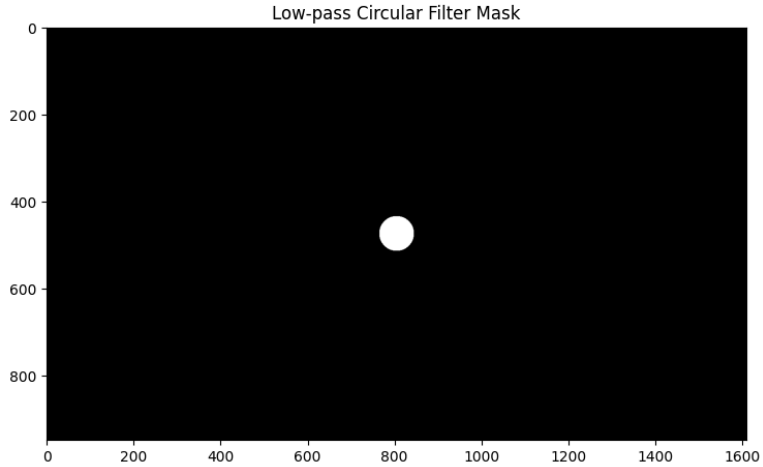
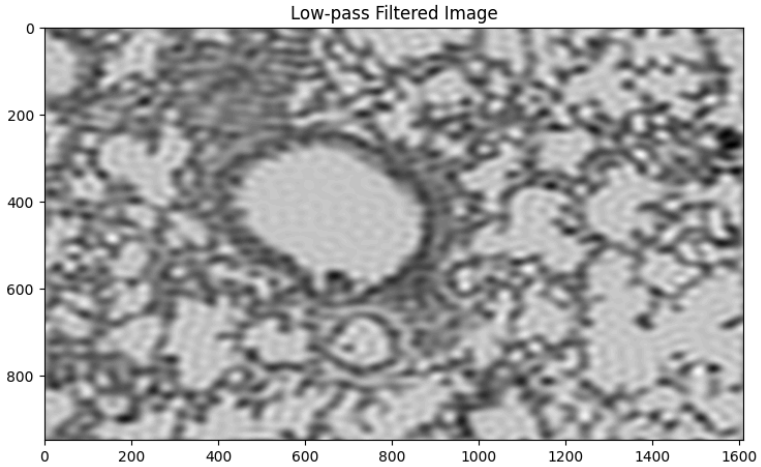




5B Calculate frequency domain filters for a high-pass and low-pass filter

To create a low-pass or high-pass, we first define an empty matrix with the same size as our image. Then we take the center of these filters to 0 and 1 with a fixed radius of r.

```
# Get the image dimensions
rows, cols = img.shape
crow, ccol = rows // 2 , cols // 2  # Center of the image

x = np.arange(0, cols)
y = np.arange(0, rows)
xv, yv = np.meshgrid(x, y)

distance_from_center = np.sqrt((xv - ccol)**2 + (yv - crow)**2)

r = 40
#High-pass Filter (HPF)
hpf = np.ones((rows, cols), np.uint8)
hpf[distance_from_center < r] = 0

#Low-pass Filter (LPF)
lpf = np.zeros((rows, cols), np.uint8)
lpf[distance_from_center < r] = 1
```

5C Apply the filters to your image	    ``` # (FFT) convertg image to frequency domain f = np.fft.fft2(img) fshift = np.fft.fftshift(f) # Shift the zero frequency component to the center # High-pass Filter fshift_hpf = fshift * hpf # Inverse FFT to get the filtered image back f_ishift_hpf = np.fft.ifftshift(fshift_hpf) # Inverse FFT shift ```

```
img_hpf = np.fft.ifft2(f_ishift_hpf)
img_hpf = np.abs(img_hpf)  # (real part)

# Low-pass Filter
fshift_lpf = fshift * lpf
# Inverse FFT to get the filtered image back
f_ishift_lpf = np.fft.ifftshift(fshift_lpf)  # Inverse FFT shift
img_lpf = np.fft.ifft2(f_ishift_lpf)
img_lpf = np.abs(img_lpf)  #(real part)
```

5D: Compare the performance of the filters to convolution kernel-based filters (from section 3). What could account for differences?

Aspect	Frequency Domain (FFT-based)	Spatial Domain (Convolution Kernel)
Edge Preservation	High-pass filters enhance edges but can also amplify noise.	Median filters preserve edges well; Gaussian filters soften them.
Noise Reduction	Low-pass filters effectively reduce noise but blur the image.	Median filters are great at removing salt-and-pepper noise; Gaussian filters reduce noise at the cost of some blurring.
Homogeneous Areas	Low-pass filters effectively smooth homogeneous areas.	Gaussian filters smooth areas well; median filters handle salt-and-pepper noise better.
Computational Cost	Efficient for large images or global filtering.	Efficient for small kernel sizes or localized filtering.

Frequency domain filters apply global changes to the entire image based on frequency content. This means edges, noise, and smooth regions are all treated globally based on the frequency.
Convolution kernel filters operate locally, using a fixed neighborhood (kernel) size, allowing for more control of how edges and noise are treated at a pixel level.

Again they are all application specific, but these knowledges are needed when trying to decide which approach to choose.