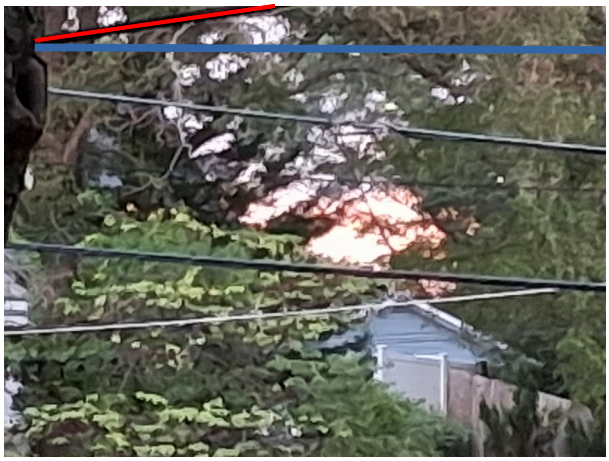
Report 1

Project 1: Histogram and Filtering

TJ DiMeola
Course: Computer Vision CSCI 581
Instructor: Dr. Hawk Wang
Date: September 25, 2025

Part 1: Perspective and orthographic projections

This question aims to assist understanding of how projection affects images. Using a camera take two photos of the same scene:



- Perspective projection: Take a normal photo, where lines appear to converge due to perspective (e.g., a desk or the edges of a building).

- Orthographic projection (approximate): Either zoom in from farther away or crop the central part of the image so that lines appear more parallel and less converging. Choose any scene you like (indoor or outdoor), but make sure there are some straight edges in the images.

Question 1 (and submission requirements):

- Submit both images (perspective and approximate orthographic).
- On each image, draw or overlay lines along a few straight edges.
- Write a brief note (a few sentences) describing the difference you see between the two projections.

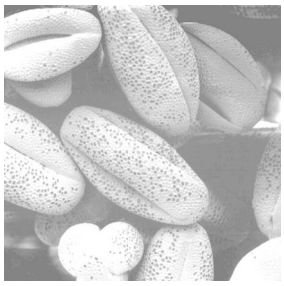
Discussion:
The images above are not great examples because I could not get sufficiently close to the wires to really show the change in angle. That said, even so, there is enough of a difference in the angles between them to make the point. "close up" = smaller angle, further away = larger angle.

Part 2: Histogram Manipulation & Linear Filtering

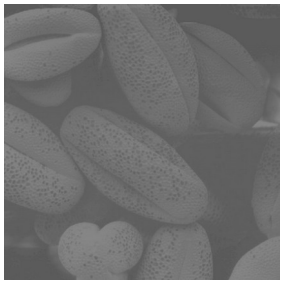
The purpose of this part is to help you understand and implement histogram equalization, one of the fundamental point operations, which is useful to enhance image contrast.

(a) Histogram Equalization on Provided Images (required)

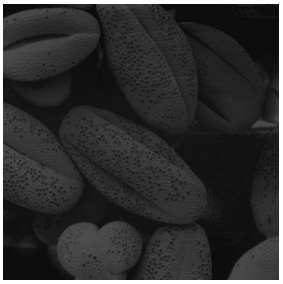
Given the three test images (grayscale test image) with low or uneven contrast (see below).



test image 1.png



test image 2.png



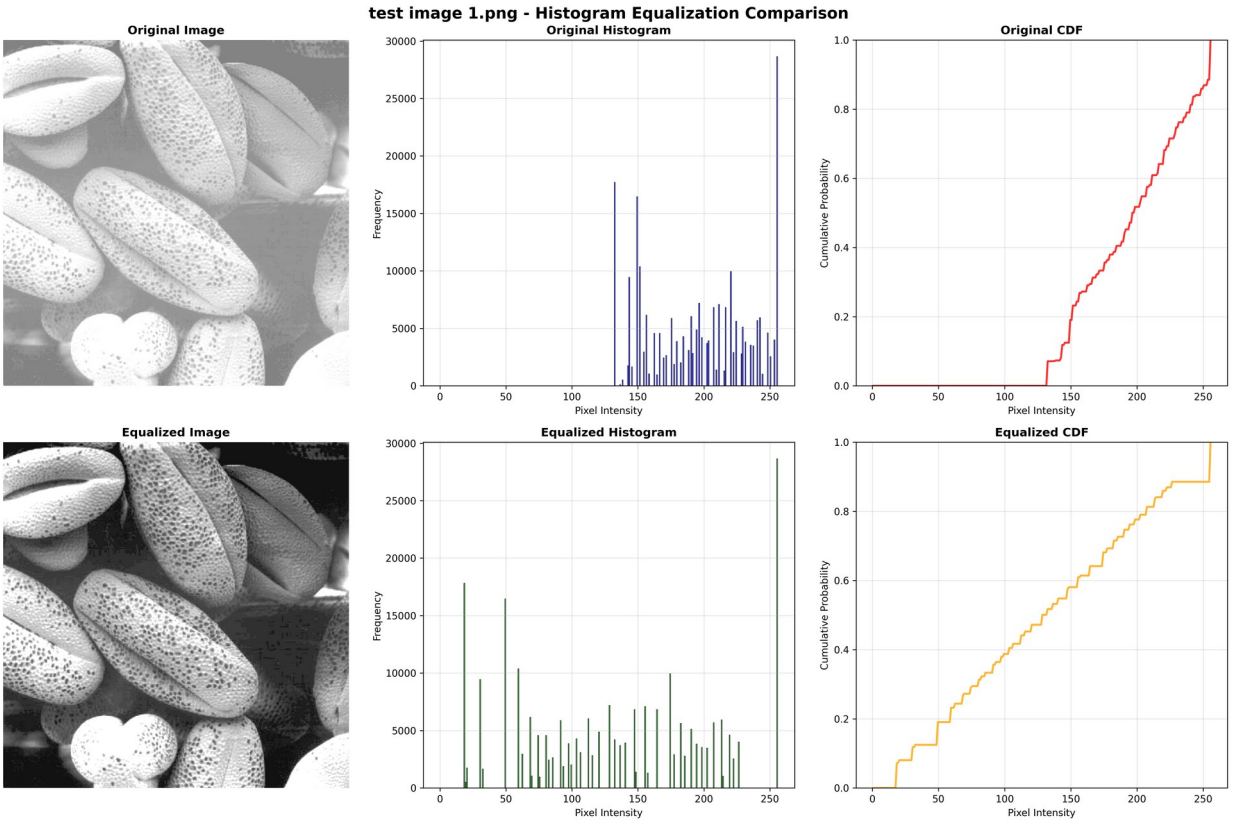
test image 3.png

Question 1:
CODE: *part2_ab.py*

- Compute the histogram of each image.
- Compute the cumulative distribution function (CDF) from the histogram.
- Use the CDF to perform histogram equalization.

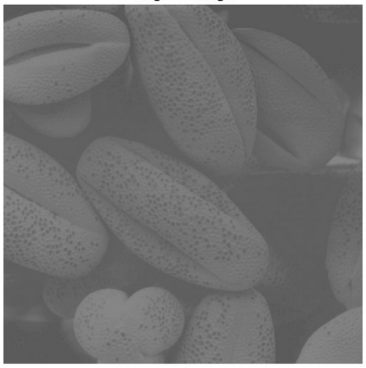
Display, for each test image:

- The original image and its histogram
- The equalized image and its histogram

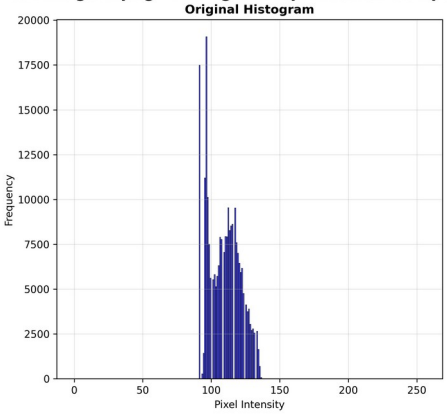


test image 2.png - Histogram Equalization Comparison

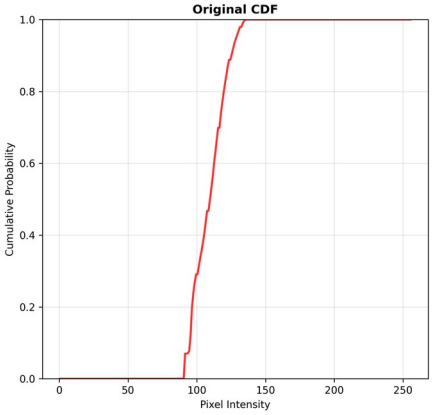
Original Image



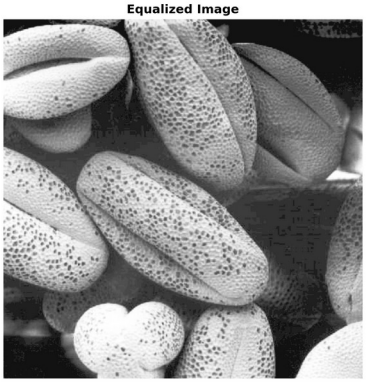
Original Histogram



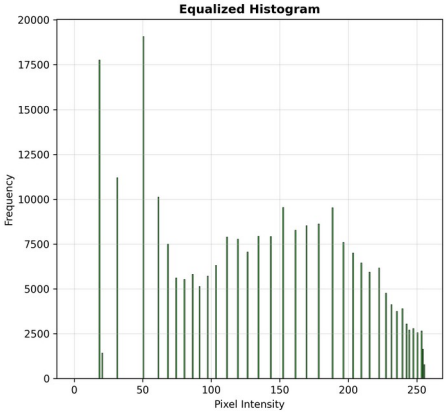
Original CDF



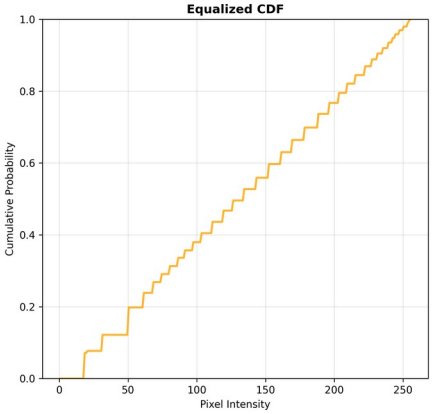
Equalized Image



Equalized Histogram

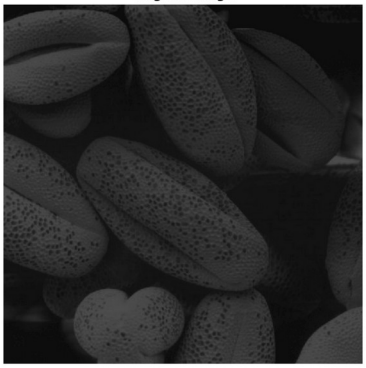


Equalized CDF

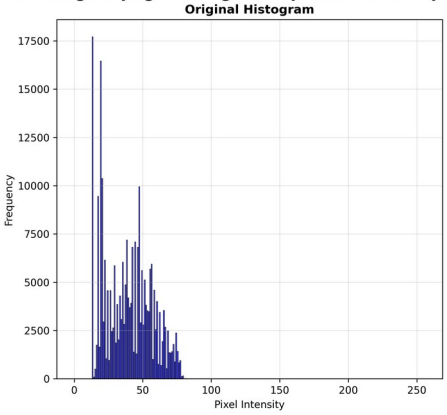


test image 3.png - Histogram Equalization Comparison

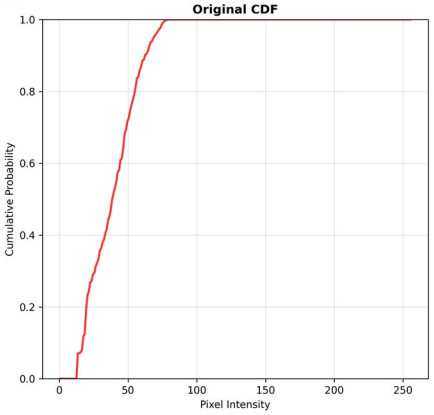
Original Image



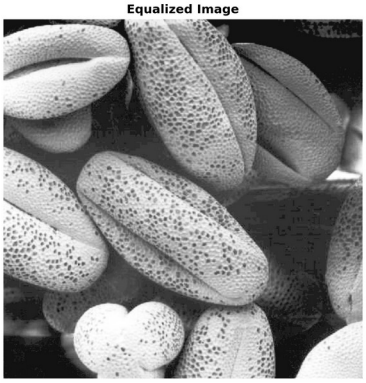
Original Histogram



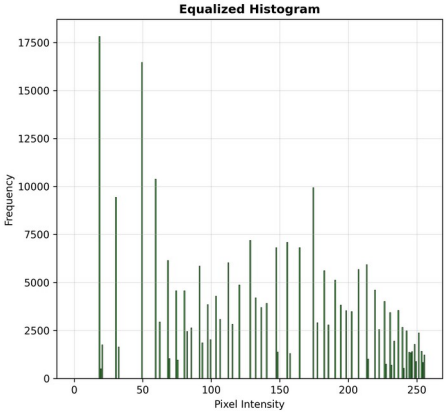
Original CDF



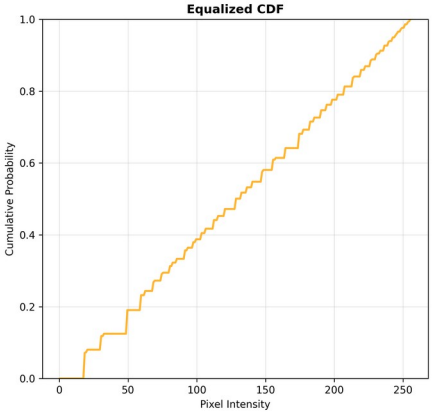
Equalized Image



Equalized Histogram



Equalized CDF



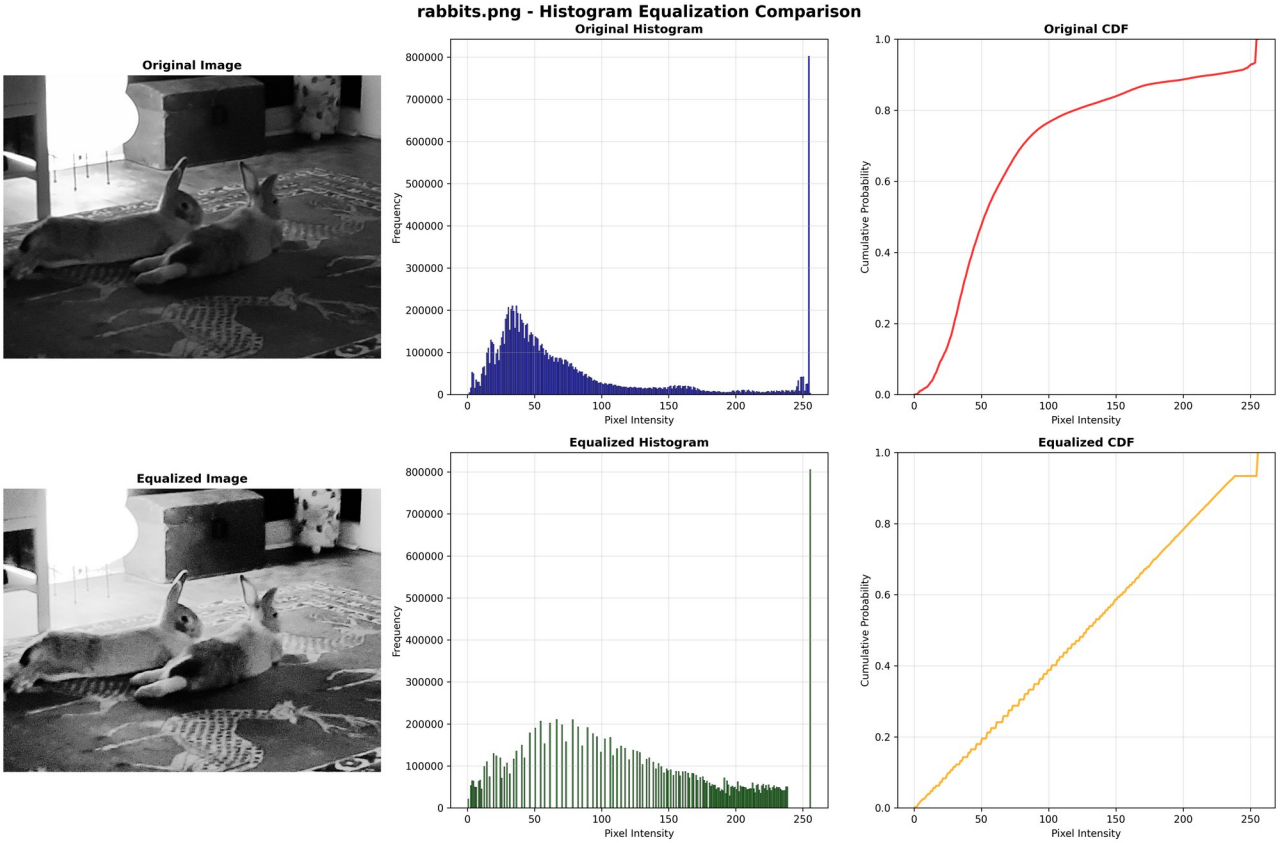
(b) Creative Task: Apply to Your Own Images (required)

Question 2

CODE: *part2_ab.py*

Find at least one of your own images with poor contrast (too dark, too bright, or low contrast). Apply your histogram equalization implementation to this image and show the result.

- Submit the original and equalized versions side by side (image and its histogram).



- Briefly describe how equalization changed the image and whether you think it improved the quality.

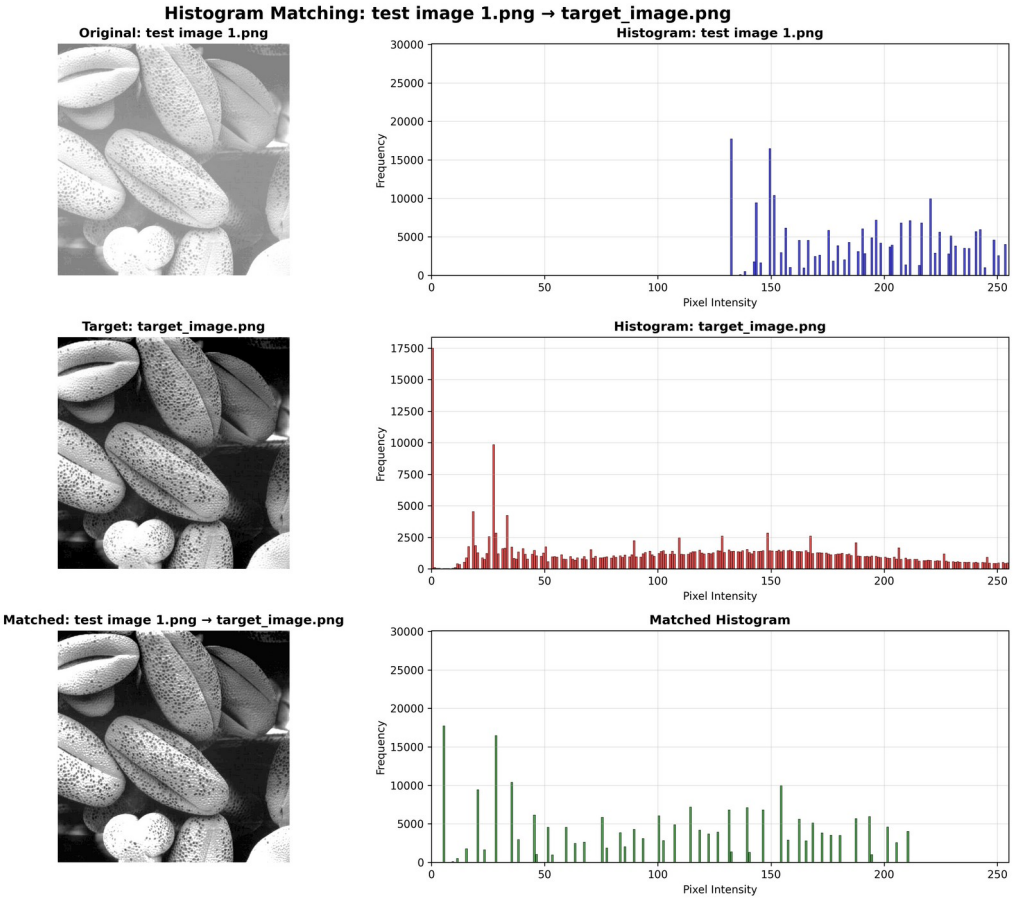
Discussion: The original image has (for all intents and purposes) a single bright light source in the background. This leads the foreground, the focal point of the image, to look dark. In other words, as seen by the histogram of the original image, the peak is resides at a low intensity, while there exists a small spike at the high intensity which represents the light source. This odd relationship between light source and focal point can be seen in the CDF which is not linear and includes that discontinuity at the end.

Following equalization, the intensity values are smoothed across the image such that our focal point is now clearly visible. While the point light source still appears as a spike in the histogram and a discontinuity in the equalized CDF, its contribution does much less to damage the quality of the image.

(c) Optional Challenge: Histogram Matching (optional)

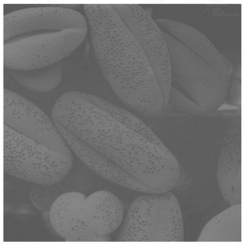
Question 3
CODE: *part2_c.py*
As an optional extension, try implementing histogram matching. The goal is to transform the histogram of a given image so that it matches a target histogram. You are provided with a target image.

- Apply histogram matching to the three test images provided above so that their histograms resemble that of the target.
- Submit the original image and its histogram, the target image and its histogram, and the matched image and its histogram.

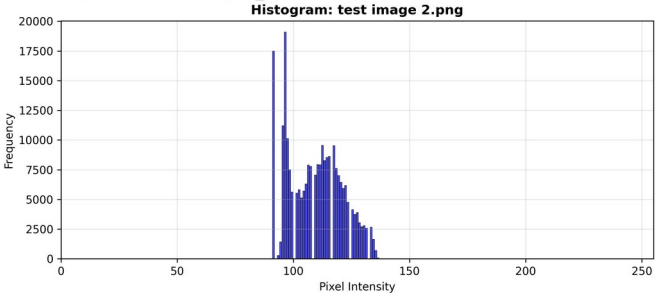


Histogram Matching: test image 2.png → target_image.png

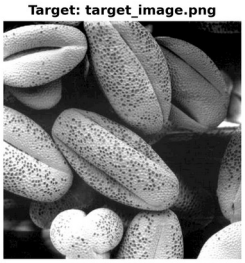
Original: test image 2.png



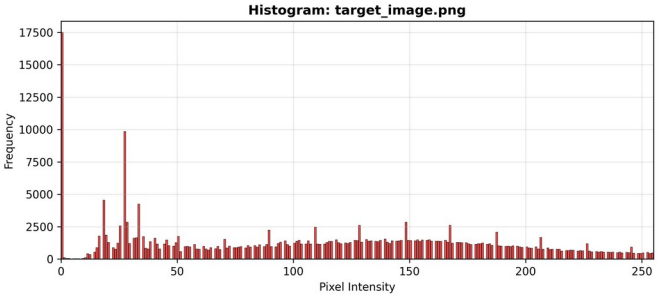
Histogram: test image 2.png



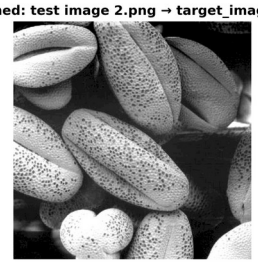
Target: target_image.png



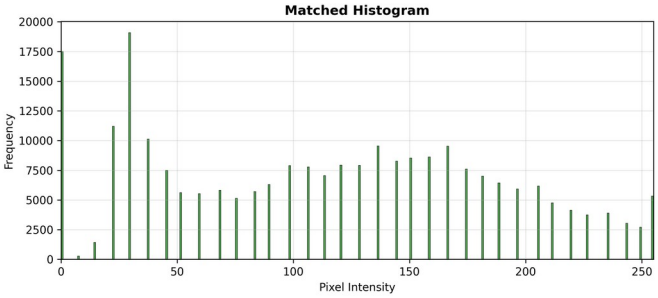
Histogram: target_image.png



Matched: test image 2.png → target_image.png

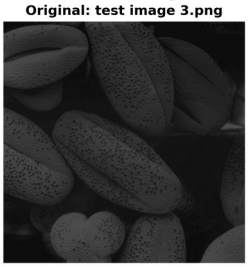


Matched Histogram

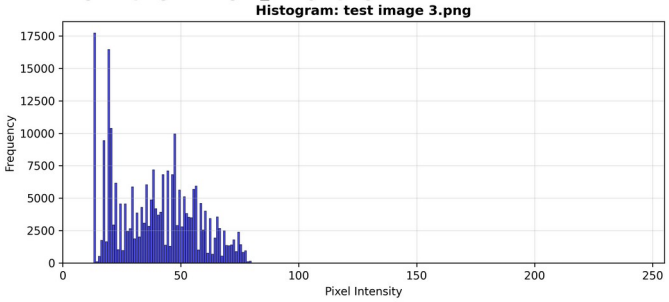


Histogram Matching: test image 3.png → target_image.png

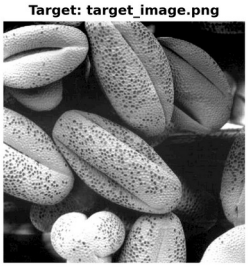
Original: test image 3.png



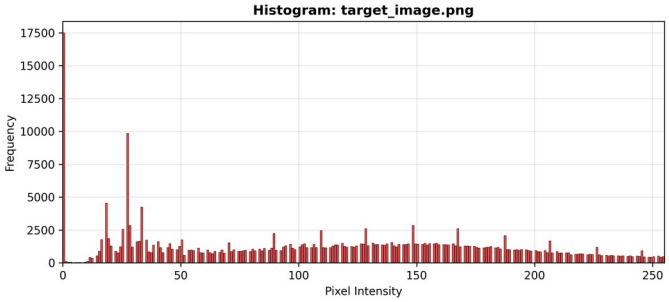
Histogram: test image 3.png



Target: target_image.png

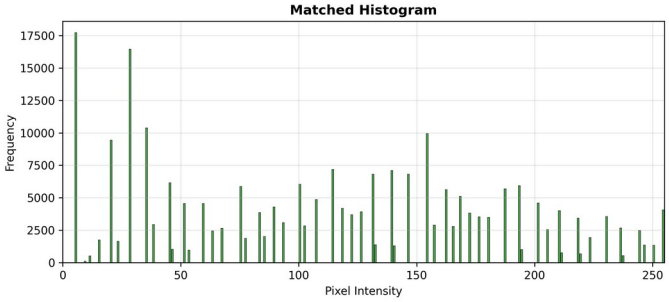


Histogram: target_image.png



Matched: test image 3.png → target_image.png

Matched Histogram

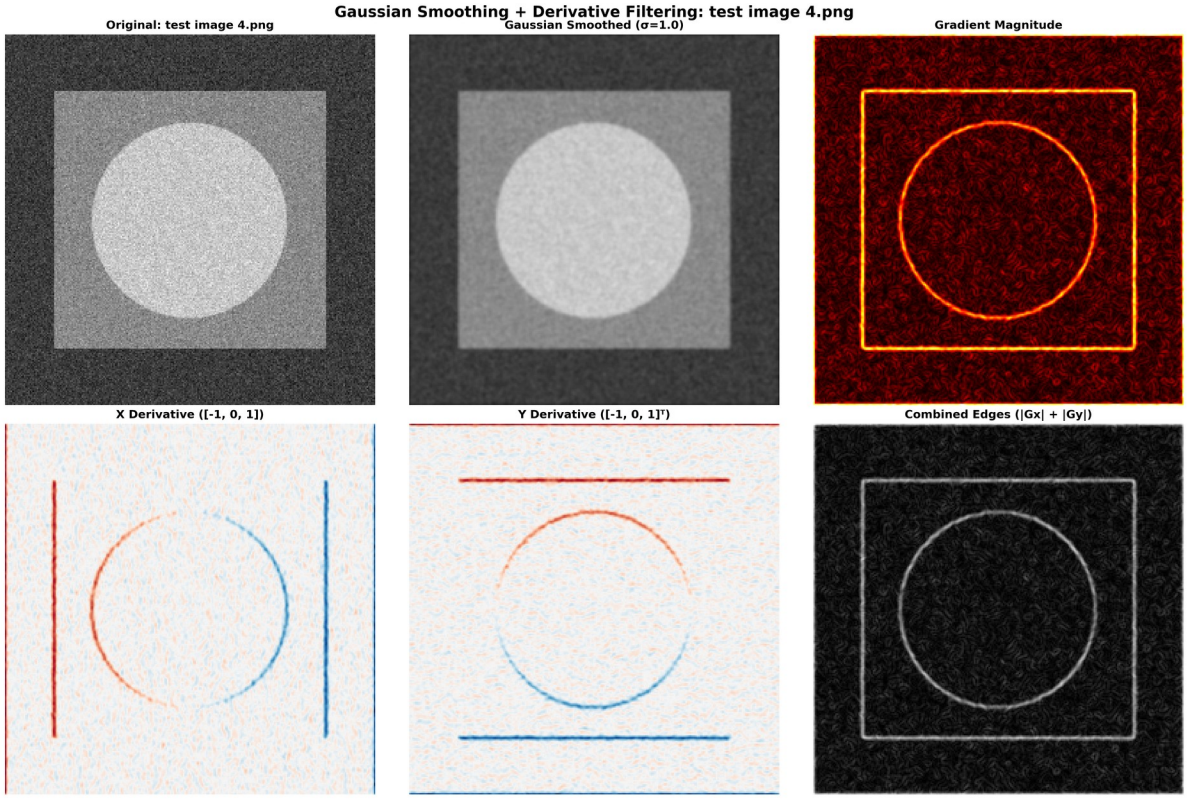


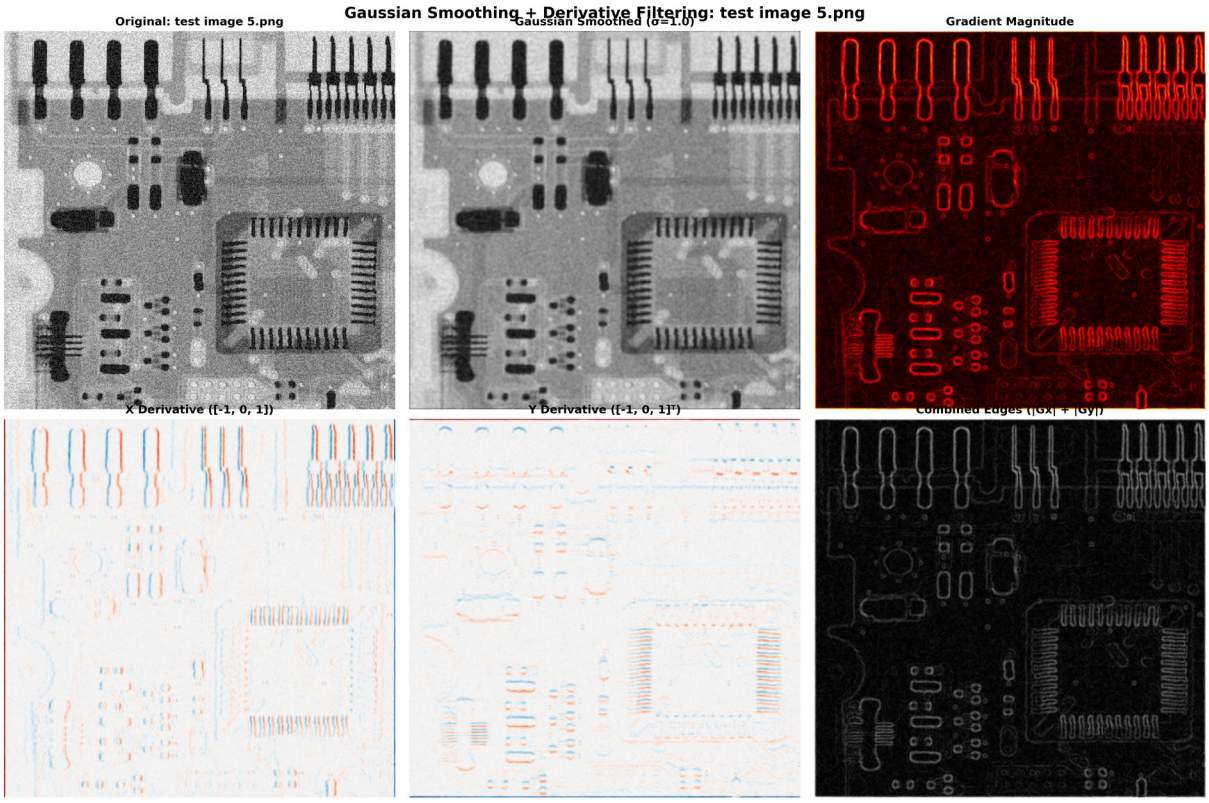
(d) Derivative of Gaussian (Required)

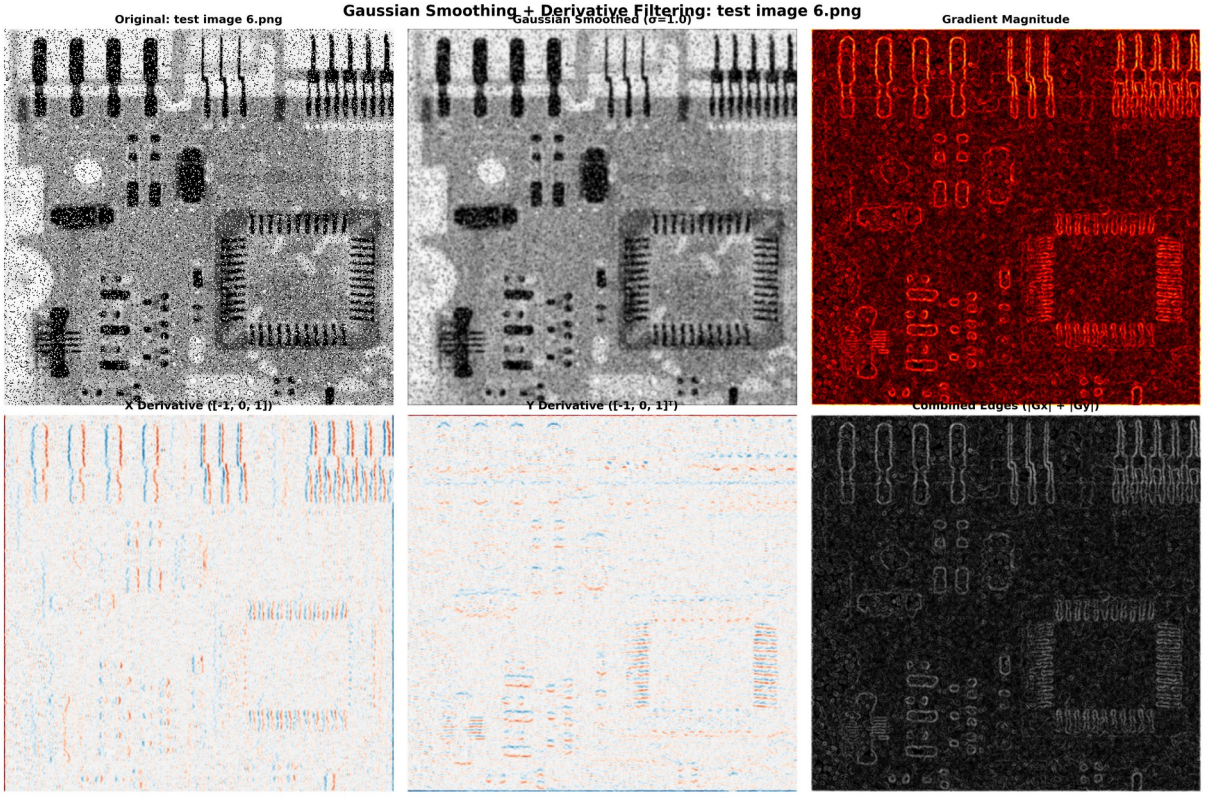
In class, we saw that applying a Gaussian filter followed by a derivative filter is mathematically equivalent to directly applying a derivative of Gaussian (DoG) filter. Here, you will implement both approaches and compare the results.

Question 4:
CODE: *part2_d.py*
You are given three test images (grayscale test image), the task is to:

- **Naive Approach (Two Steps)**
  - Smooth the image using a Gaussian filter.
  - Apply a simple derivative filter (e.g., [-1, 0, 1]) in the x-direction and y-direction separately.
  - Compute and visualize the results for both directions.




Gaussian Smoothing + Derivative Filtering: test image 5.png
Original: test image 5.png
Gaussian Smoothed (σ=1.0)
Gradient Magnitude
X Derivative ([-1, 0, 1])
Y Derivative ([-1, 0, 1]ᵀ)
Combined Edges (|Gx| + |Gy|)

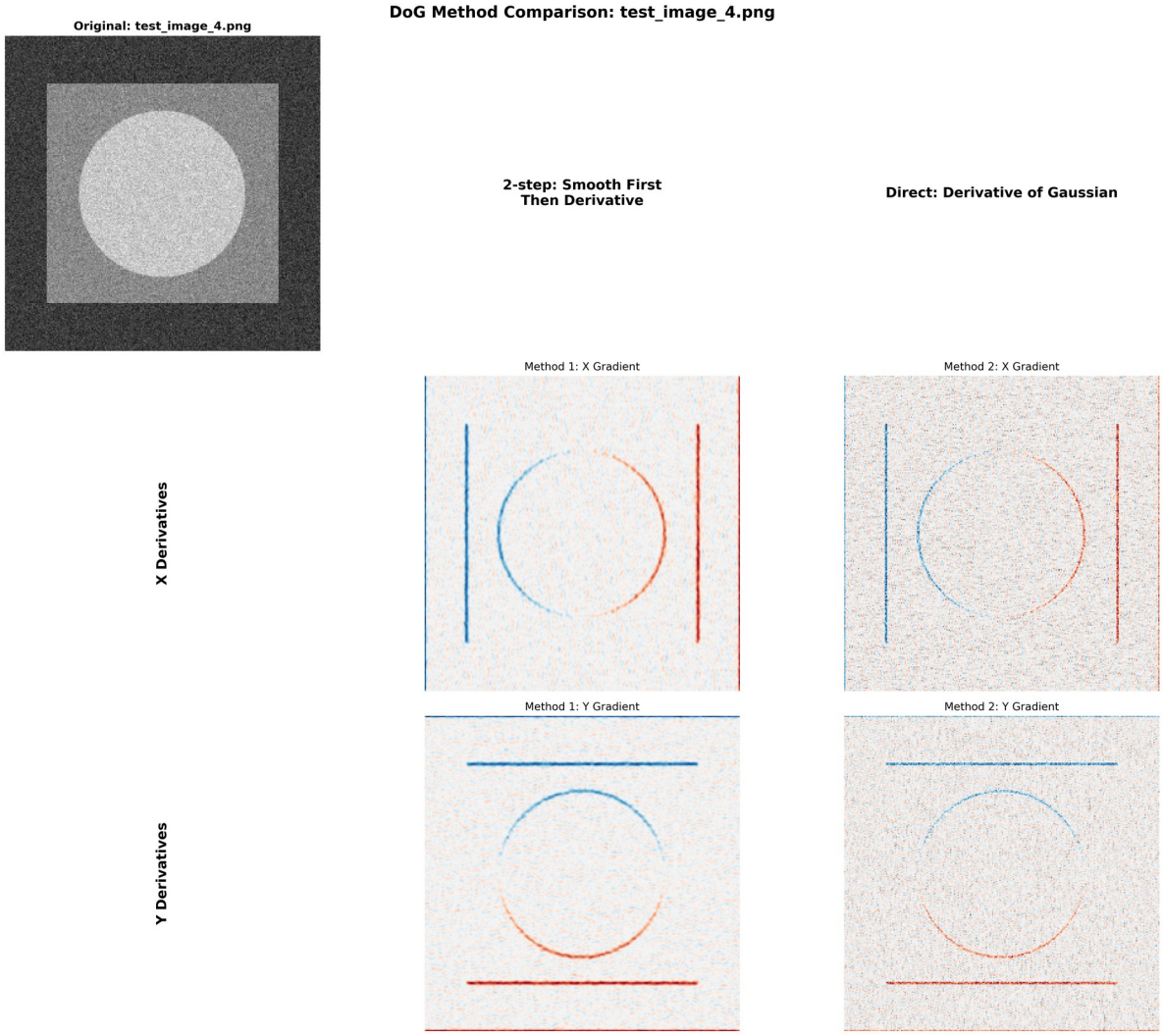

Gaussian Smoothing + Derivative Filtering: test image 6.png
Original: test image 6.png
Gaussian Smoothed (σ=1.0)
Gradient Magnitude
X Derivative ([-1, 0, 1])
Y Derivative ([-1, 0, 1]ᵀ)
Combined Edges (|Gx| + |Gy|)

- **Direct Approach (One Step)**
  - Construct a derivative of Gaussian filter in the x-direction and y-direction
  - Convolve the image directly with these filters.
  - Visualize the results.

- **Comparison**
  - Display the outputs from both approaches side by side.
  - Briefly describe your observations: are the results nearly the same?
  - Why might small differences appear?

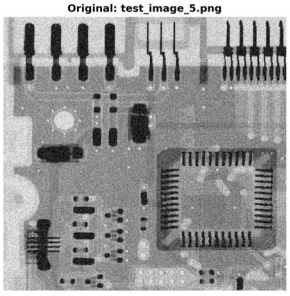
Discussion: (See images below)

The 2 step approach seems to work better for images that are already noisy (image 4 and 6). Where one is starting with a “good” image (image 5), the direct approach provides significantly more nuanced edge discrimination.



DoG Method Comparison: test_image_5.png

Original: test_image_5.png

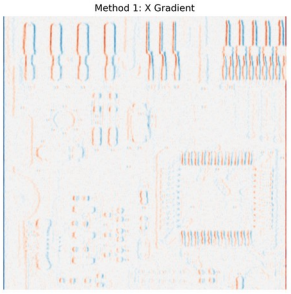


2-step: Smooth First Then Derivative

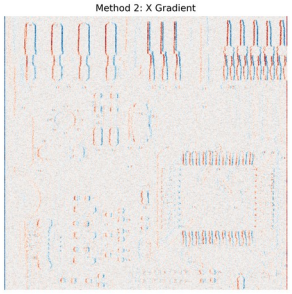
Direct: Derivative of Gaussian

X Derivatives

Method 1: X Gradient

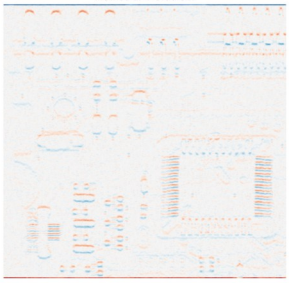


Method 2: X Gradient

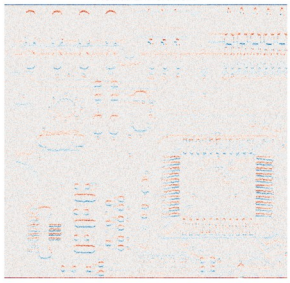


Y Derivatives

Method 1: Y Gradient

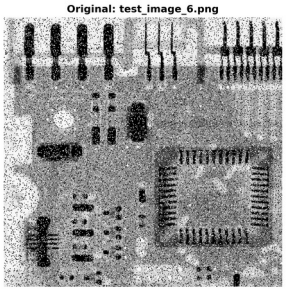


Method 2: Y Gradient



DoG Method Comparison: test_image_6.png

Original: test_image_6.png

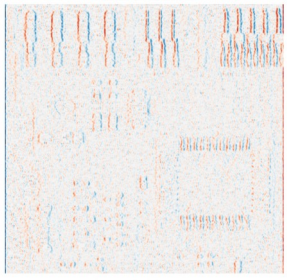


2-step: Smooth First Then Derivative

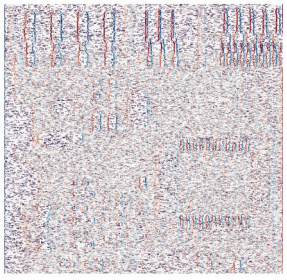
Direct: Derivative of Gaussian

X Derivatives

Method 1: X Gradient

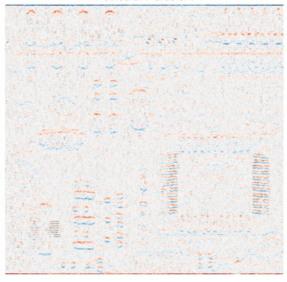


Method 2: X Gradient

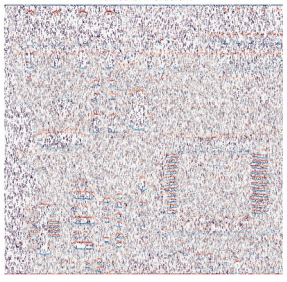


Y Derivatives

Method 1: Y Gradient



Method 2: Y Gradient

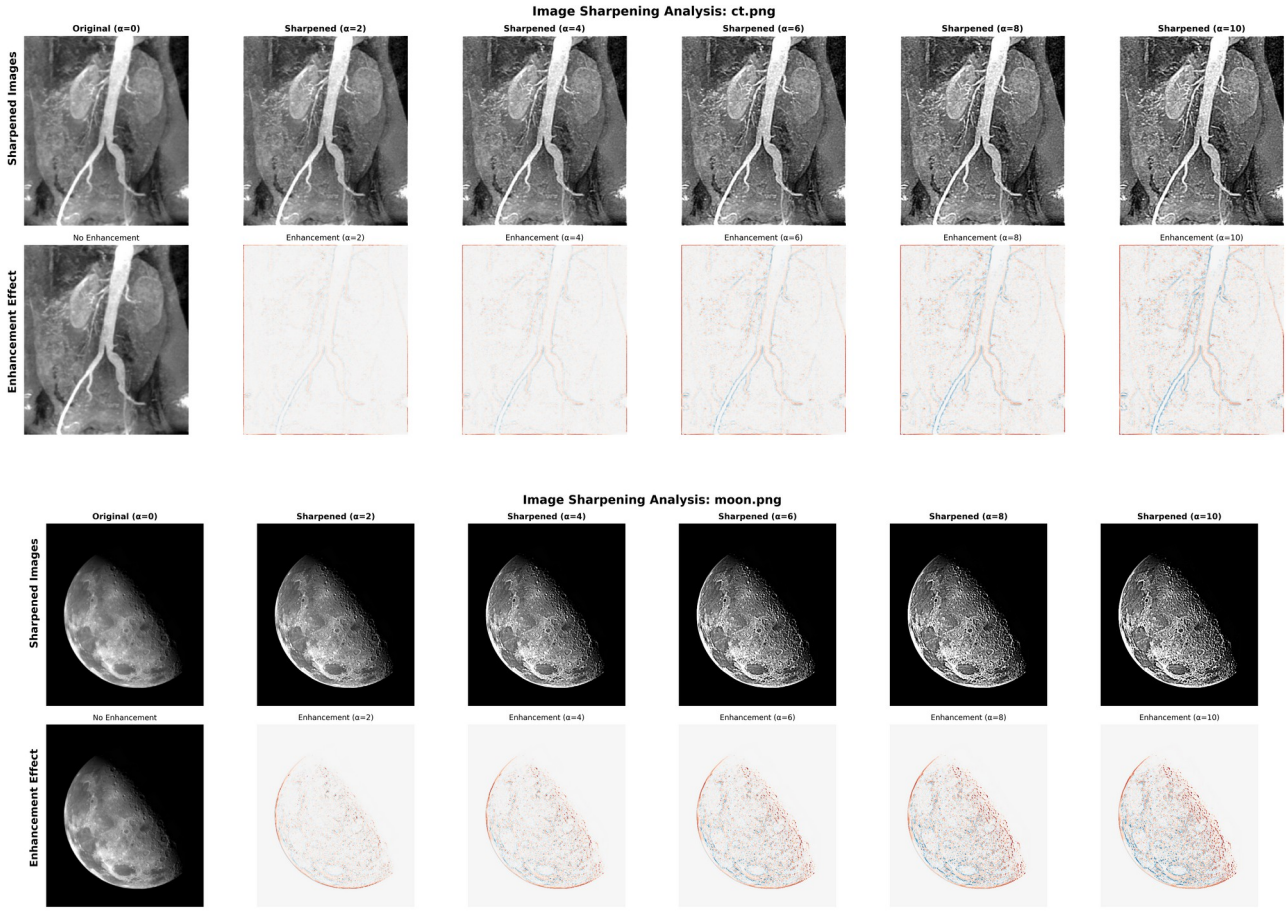


(e) Creative Task: Image Sharpening (Required)
CODE: *part2_e.py*
In class, we discussed that by subtracting a smoothed (low-pass) version of an image from the original, we can extract the edges and details (high-frequency details). Adding these details back into the original image enhances sharpness:

Question 5
You are given two test images (grayscale test image ct.png and moon.png), the task is to:

- **Extract High-Frequency Details**
  - Smooth the image using a Gaussian filter.
  - Subtract the smoothed version from the original image to obtain the high-frequency component.

- **Sharpen the Image**
  - Add the high-frequency component back to the original image.
  - Experiment with different blending weights (e.g., original + α × high-frequency, with α ranging from 1 to 10).
  - Observe how the sharpness changes as you vary α.



- **Creative Exploration**
  - Apply your sharpening method to your own images.
  - Pick at least one example where sharpening makes the image look noticeably better, and one example where too much sharpening creates artifacts (e.g., amplified noise).

Discussion: Viewed close up the top images (flowers) do not appear to contain artifacts, regardless of the sharpening amount, while the lower images (landscape) begin to have significant artifacts at about alpha = 6. This can be seen through the lens of the high-frequency artifacts, where there is really nothing to see for the flowers, while high-frequency values for the landscape are clearly visible along the edges where the intensity values become mixed (vegetation and sky, vegetation and field). The flowers are distinct from the background, so do not evince such artifacts (clear boundary between pixel values).

Part 3: Anisotropic Diffusion
CODE: *part3.py*

This part introduces you to anisotropic diffusion, a classic image filtering technique (also called Perona–Malik diffusion) proposed by Pietro Perona and Jitendra Malik in their 1990 IEEE Transactions on Pattern Analysis and Machine Intelligence paper. The goal is to reduce noise in images while preserving important structures such as edges, something that ordinary Gaussian smoothing cannot achieve.

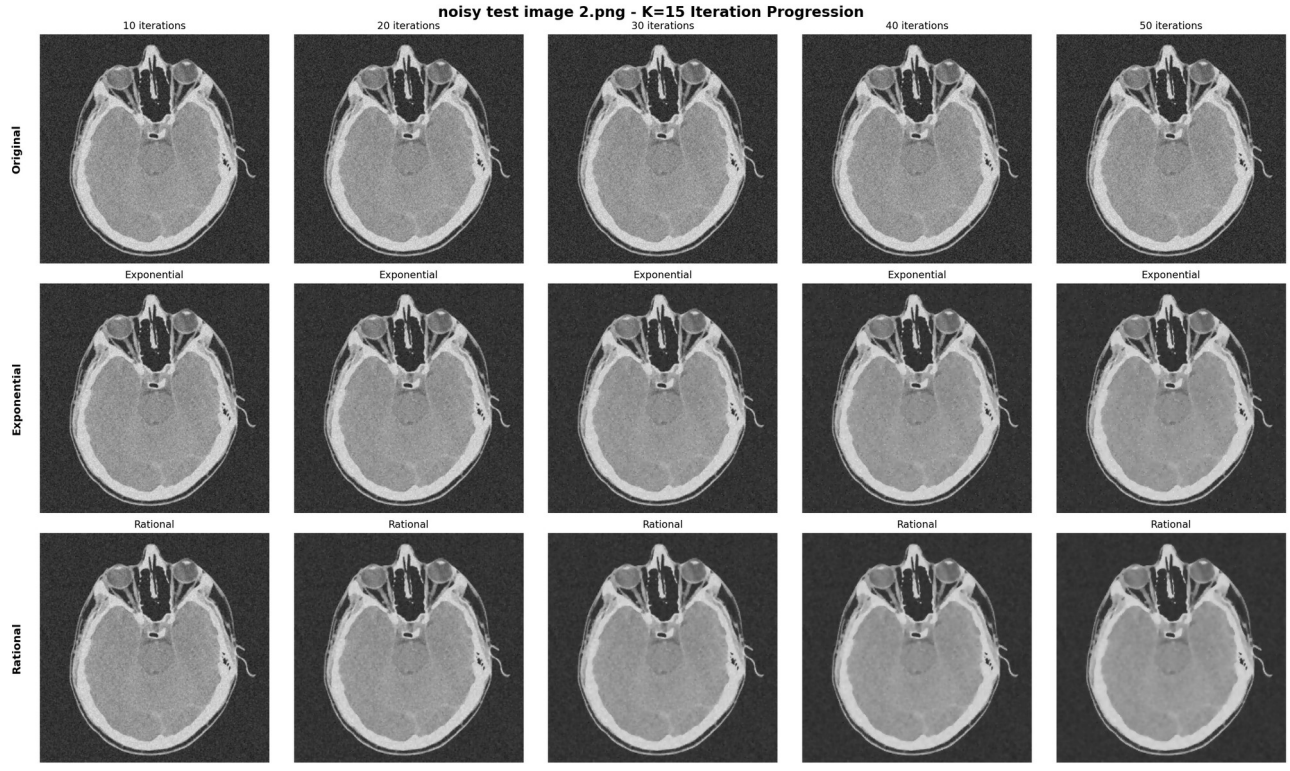
Question 1: Implementation of Anisotropic Diffusion

where $\frac{\partial I}{\partial t} = \nabla \cdot (c(x, y, t) \nabla I)$

is the diffusion coefficient that controls the amount of smoothing depending on the local image gradient. Use the following two diffusion functions:

$$c(|\nabla I|) = e^{-\left(\frac{|\nabla I|}{K}\right)^2} \text{ and } c(|\nabla I|) = \frac{1}{1+\left(\frac{|\nabla I|}{K}\right)^2}$$

Run the implementation for different values of the parameter K different numbers of iterations. How does the choice of K affect the result? What happens as you increase the number of iterations?



Discussion: Since K handles (local) edge detection, its impact on the images is quite subtle compared to lambda and the number of iterations which more or less control the *rate* at which the smoothing takes place. This means that even if you get great smoothing, you may have edges running into each other as discussed in the next question (2). The number of iterations can be safely increased to very high numbers, but of course there are computational considerations, and the changes may not be significant after a certain threshold. Sigma is tricky: it is recommended not to increase sigma > .25 or risk non-convergence. But I did it and the results were interesting (see below, and in the results folder, results_diffusion).

Question 2: Comparison with Gaussian Smoothing

- Apply Gaussian smoothing to the same images.
- Compare results side-by-side: how does anisotropic diffusion preserve edges differently? Discuss the results and the differences you observed.

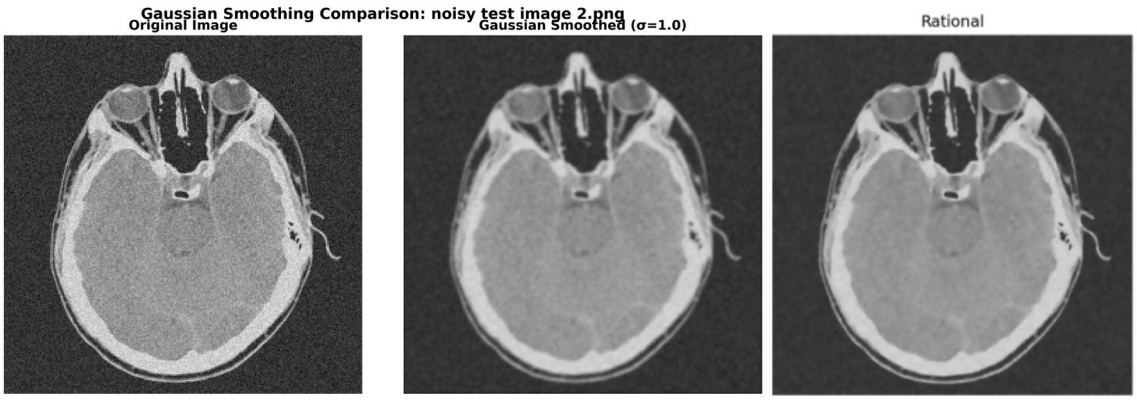
Above, original image (left) Gaussian smoothing (right). To the right, exponential and rational diffusion carried out at lambda = .25, K = 95, # of iterations = 50.

Rational

General observations:

- Better control of smoothing with PM diffusion (can control lambda, K, and iterations, where Gaussian has only sigma.
- Rational *may* have a slight edge over exponential in retaining features of significance.
- PM began to loose certain features where pixels were very close. Particularly evident in more complex image (Lena, right), at the point where the smoothing seemed almost optimal.
- Gaussian begins to loose significant features during smoothing. This is particularly evident in the brain scan below, where the small features in the Gaussian begin to look smudged, while PM remains crisp.

Below, original image (left) Gaussian smoothing (right). To the right, rational diffusion carried out at lambda = .25, K = 95, # of iterations = 50.



Question 3 (optional): Think critically about the limitations of anisotropic diffusion. Suggest one possible improvement or alternative approach, drawing from your own intuition or knowledge from other methods.

Discussion: While the number of controllable parameters available with Perona-Malik diffusion makes it attractive for precise smoothing, they are also quite sensitive. Convergence to an acceptable smoothness may take many iterations for low lambda, but if the lambda is increased too much, there is no convergence at all.

It would make sense to tie lambda to the local high-frequency components of the image. Where they were significant, increase lambda, where insignificant, keep lambda low. This merits testing.