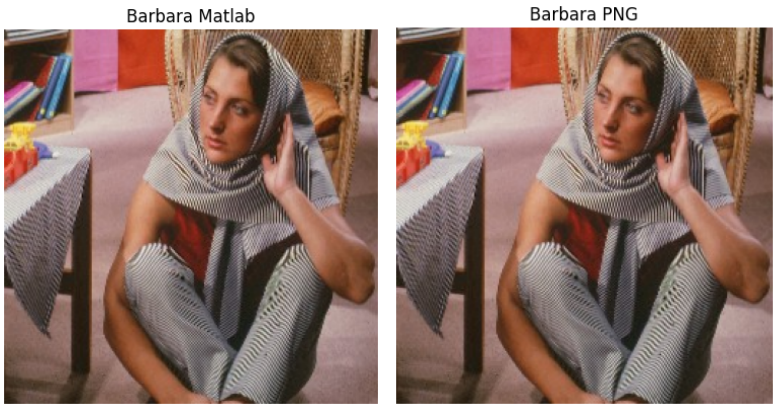
Computer vision
Lab 1

Task 1: Image Exploration

Load and Display the Image Barbara in different formats

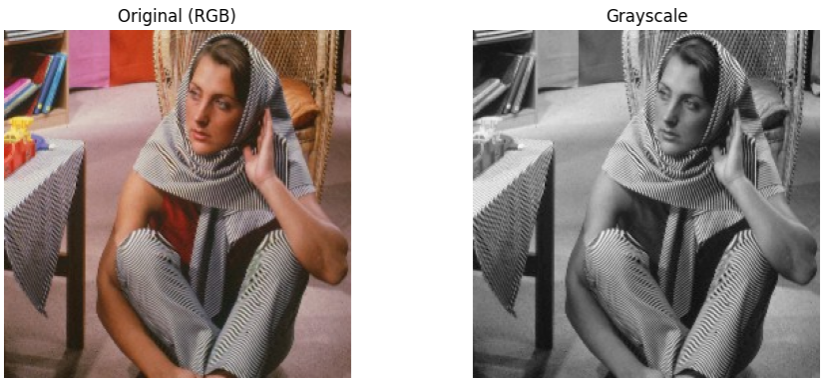
- Load the image from the `.mat` file.
- Load and display the `.png` image using:



Output of different formats is the same.

Image Conversion and Inspection

- Convert the image to grayscale.
- Check image dimensions and number of channels.
- Convert between data types: `double` ↔ `uint8`.




```
color: (256, 256, 3)
uint8
gray: (256, 256)
uint8
Color: dtype=uint8, range=(0.000, 255.000)
Gray: dtype=uint8, range=(10.000, 237.000)

Color: dtype=float64, range=(0.000, 1.000)
```

```
Gray: dtype=float64, range=(0.039, 0.929)

Color (back to uint8): dtype=uint8, range=(0, 255)
Gray (back to uint8): dtype=uint8, range=(10, 237)
```

Convert to unit8: Take the uint8 image (values 0-255) and convert it to float64 in the range [0,1] by dividing by 255. Then convert it back to uint8 by multiplying by 255 and casting. But for the grayscale the range is different.

$$I_{gray} = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

In barbara.png case, the grayscale image itself did not use the full 0–255 range. Its darkest pixel had a value of 10, and its brightest pixel had a value of 237. That happens because the original photo did not contain pure black or pure white areas, meaning it had limited contrast. After converting to float64 and then back to uint8, the same brightness range (10–237) remains unchanged.

The conversion only changes how the data are represented in memory — not how bright or dark the image actually is.

Image Resizing

- Half the original size
- One-quarter the original size

We used INTER_AREA method, which gives a better result in downscaling.

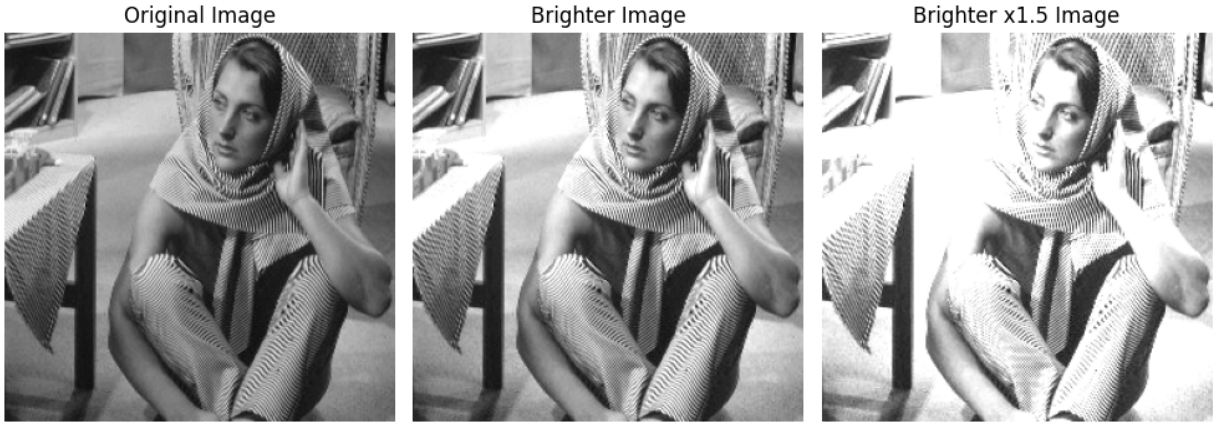
Method	What it does	Visual effect
INTER_NEAREST	Takes the value of the nearest pixel without averaging.	Very fast, but produces visible “steps” and “blocks” (pixelation).
INTER_LINEAR	Linearly averages neighboring pixels.	Slightly blurs the image but makes it look smooth.
INTER_CUBIC	Uses weighted averaging of 16 neighboring pixels (cubic interpolation).	Produces smoother results, a bit “soft,” best for upscaling.
INTER_AREA	Computes the average of all pixels that fall inside the new pixel area.	Ideal for downscaling because it reduces aliasing (moire or ripple effects).

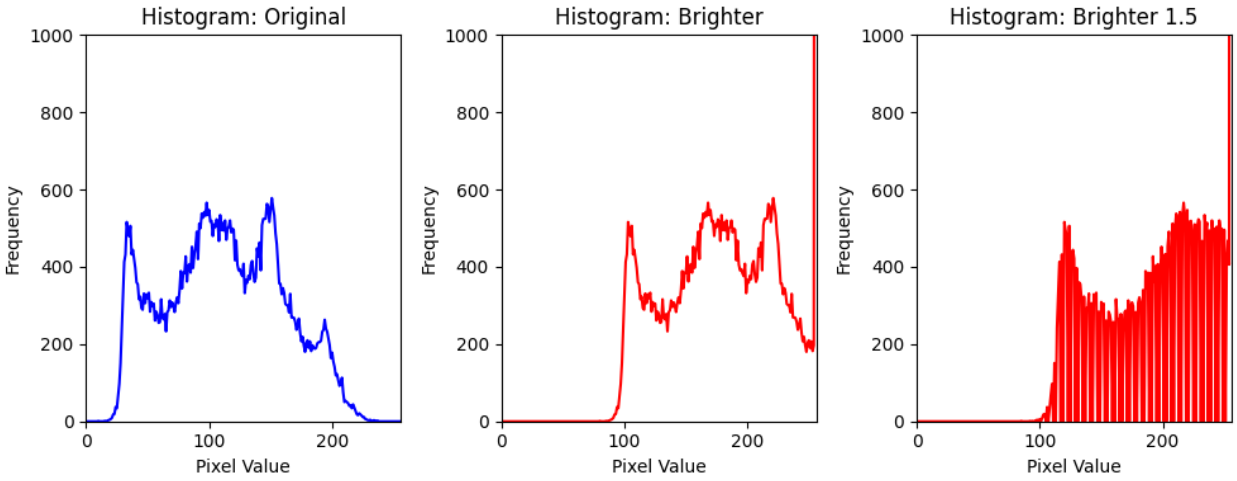
After resizing the original 256×256 image to 128×128 and then to 64×64, the one-quarter size version appeared blurry. We see how the sharpness is decreasing with downscaling.

Brightness and Histogram Analysis

- Adjust the image brightness.
- Visualize the image histogram before and after adjustment.

In the first case, we only increase brightness, while in the second case we both add brightness and multiply by 1.5.





We increased brightness by +70, but using large values for `alpha` or `beta` can cause clipping, because any pixel value above 255 is capped at the maximum intensity (255).

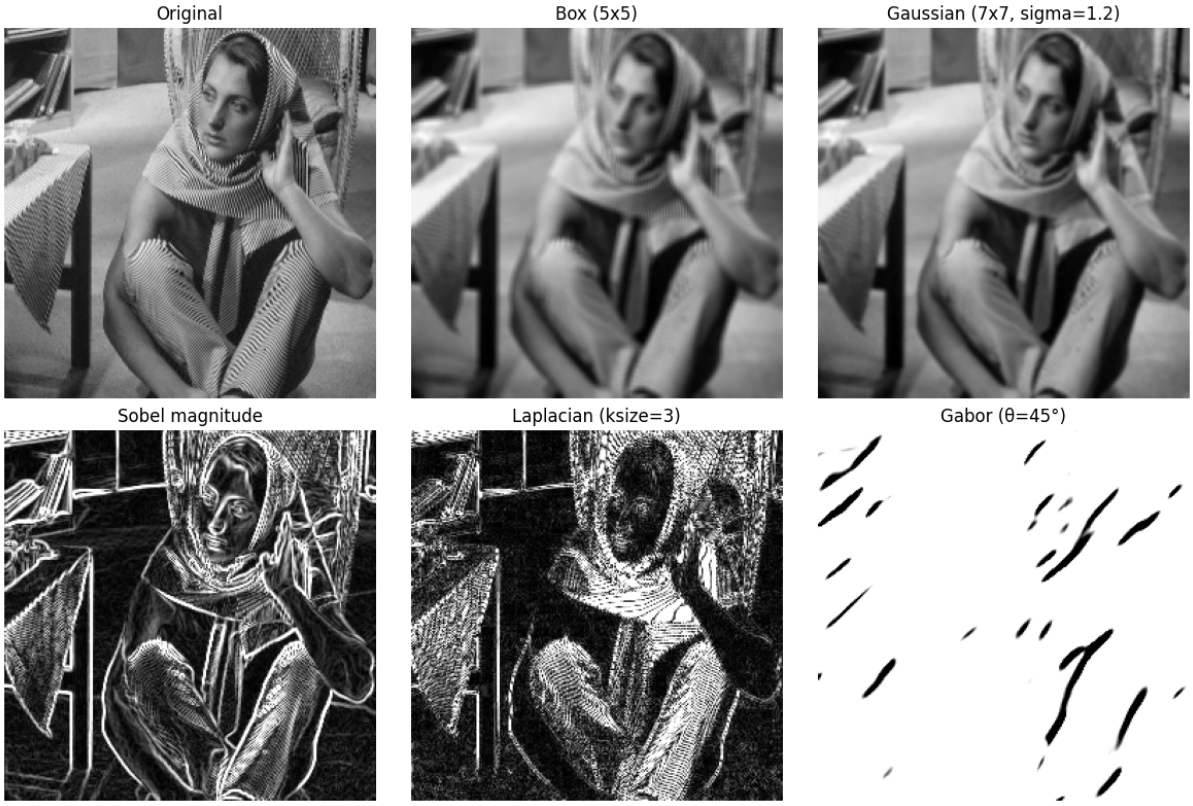
The histogram becomes wider for the image with higher brightness and contrast because multiplication amplifies the differences between pixel values. For example, if two pixels originally have values of 100 and 110 (a difference of 10), multiplying them by 1.5 changes them to 150 and 165 (a difference of 15). This stretching effect increases the contrast in the image).

Function Exploration Use the following functions to explore their effects:

- `fspecial` – Create predefined filters
- `imfilter` – Apply filtering
- `imerode` – Perform morphological erosion
- `imadjust` – Adjust image intensity
- `imhist` – Display histogram
- `histeq` – Histogram equalization
- `edge` – Edge detection
- `im2bw` – Convert image to binary (thresholding)

These functions are built-in to MATLAB; however, since we use Python in this lab, we will use their Python equivalents instead.

- "Low-frequency" = slow changes → smooth areas, gradual brightness transitions.
- "High-frequency" = fast changes → edges, fine textures, noise.



The **box (or mean) filter** replaces each pixel with the average of its neighbors inside a 5×5 window. It's a simple low-pass (smoothing) filter that reduces noise and fine details. The image becomes blurred**,** especially around edges.

Gaussian filter applies a weighted average, where closer pixels contribute more than distant ones (Gaussian distribution). It's also a low-pass filter, but smoother and more natural than the box filter. The image is softened, but edges are preserved better than in the box filter.

The Sobel filter calculates the gradient, which measures how quickly brightness changes in the image. It does this in both the horizontal (X) and vertical (Y) directions. Large gradients

correspond to strong edges, where the image changes sharply from dark to light. The output highlights edges and contours – areas with strong brightness transitions appear as bright lines.

The **Laplacian filter** measures the second derivative of the image intensity, detecting regions where brightness changes rapidly. It finds edges by looking for points where the intensity curve bends sharply.We see, that this method produces sharper and more detailed edge maps than Sobel, that's why we see strong outlines along the scarf and facial features.

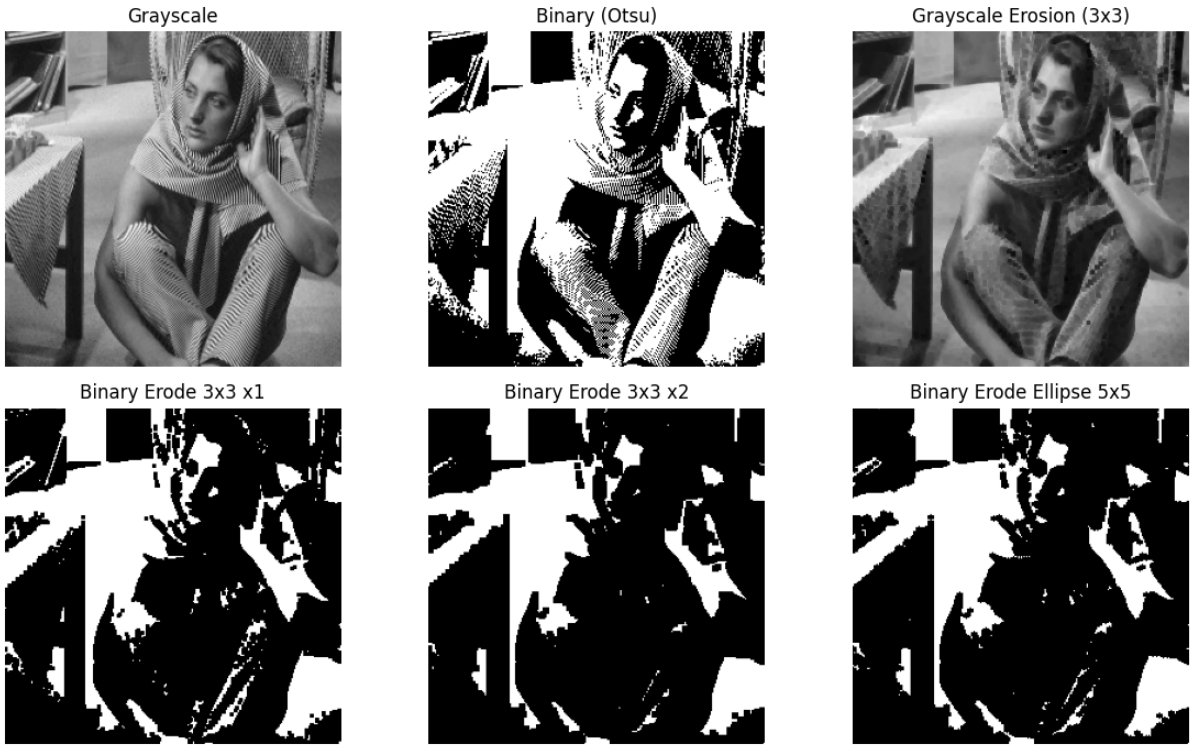
The **Gabor filter** detects patterns or textures that match a specific orientation and frequency. It responds strongly to edges or stripes that align with its defined angle – here, 45 degrees. We can see diagonal features and textures that follow a 45° direction stand out, while other directions are suppressed.

Morphological erosion

Erosion shrinks bright (white) regions and grows dark (black) regions. It's the opposite of dilation, which expands white areas.
For each pixel:

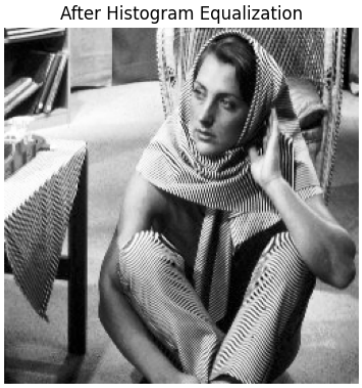
- If all pixels under the kernel are white (255), the output pixel stays white.
- If any pixel under the kernel is black (0), the output pixel becomes black.

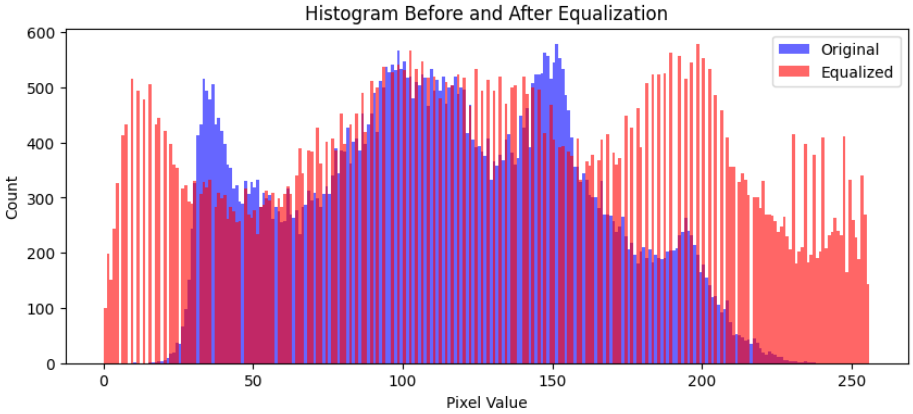
So, white objects shrink, and small white spots or thin lines disappear. Grayscale erosion replaces each pixel with the minimum value in its neighborhood.



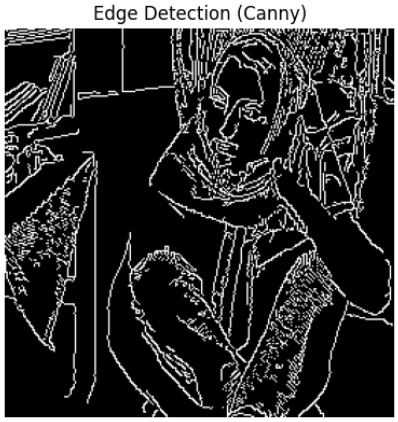
Histogram equalization

This method spreads out pixel intensities to improve contrast. Histogram equalization stretches and redistributes pixel values so that they cover the entire range (0–255) more evenly. Very common brightness levels (where histogram has peaks) are spread out. Rare brightness levels (where histogram is empty) are filled in.





Edge detection Canny



Canny uses a series of steps to find edges reliably. The Canny algorithm produces clean, thin, and continuous edges. It avoids detecting random noise as edges and keeps only the meaningful contours. This makes it one of the most accurate and widely used edge detection methods in computer vision.

Treshold method

With a fixed treshold Every pixel value above 127 becomes white (255); everything below becomes black (0). We can adjust this cutoff to control how much of the image turns white.

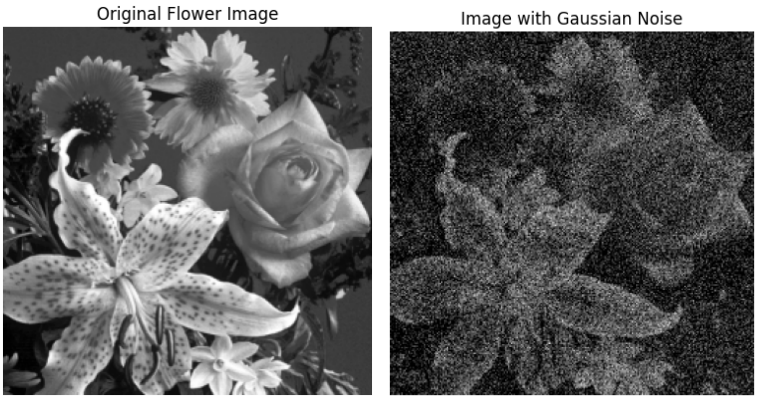
Otsu method utomatically chooses the best threshold by analyzing the image histogram. It separates dark and bright regions so that each group is as consistent as possible.

Task 2: Noise Filtering

OpenCV is mainly focused on image processing and computer vision, not image synthesis or simulation. So, when we want to add Gaussian noise, typically NumPy is used.

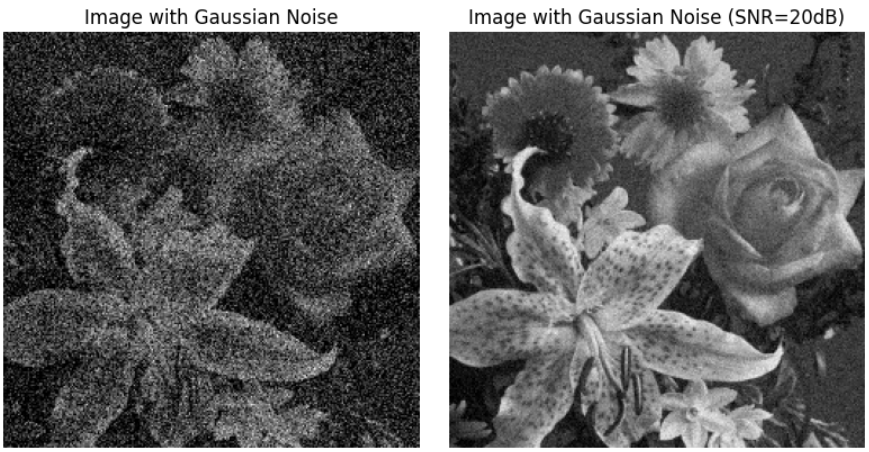
Add Gaussian Noise with:

- Zero mean



The original flower image becomes corrupted with random bright and dark speckles.

- Signal-to-noise ratio (SNR) of 20 dB



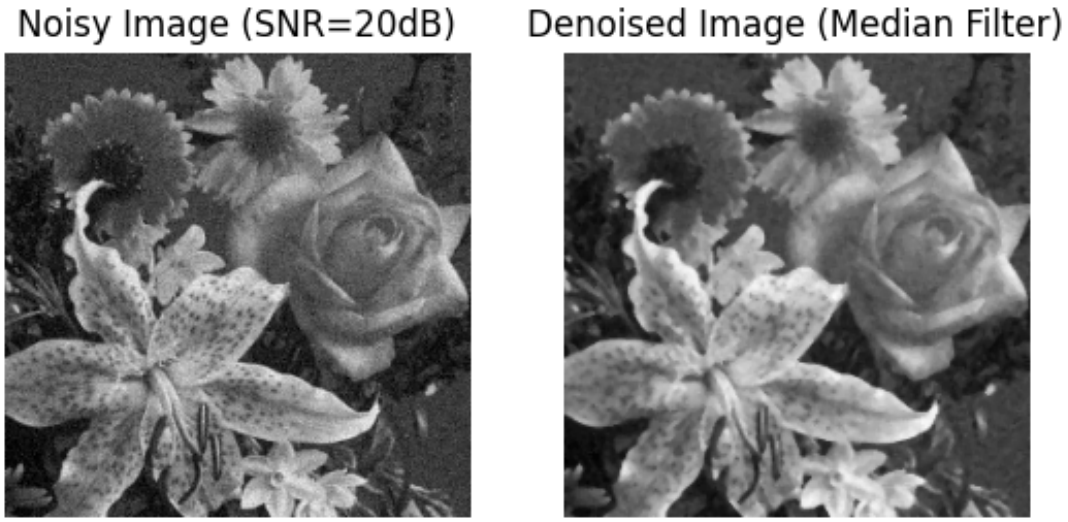
In the **standard Gaussian noise** model, the noise variance (σ^2) is fixed manually, so the amount of noise is absolute. In the **SNR-based Gaussian noise** model, the noise variance is computed relative to the image's average signal power, ensuring a consistent noise level across different images.

Apply the following filters to reduce noise:

- Moving average filter (box filter)

The filter smooths the image by averaging neighboring pixels. Noise is reduced, but fine details and edges become blurred.

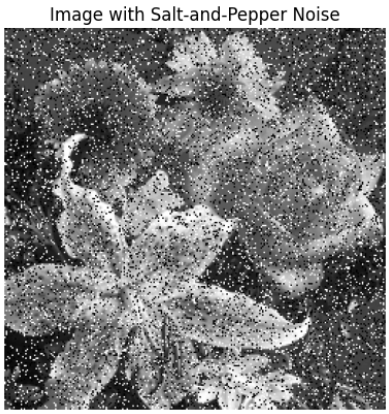
- Median filter

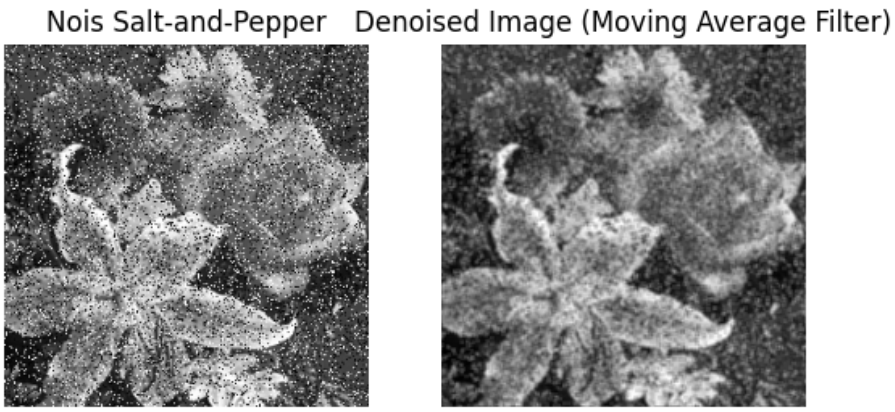


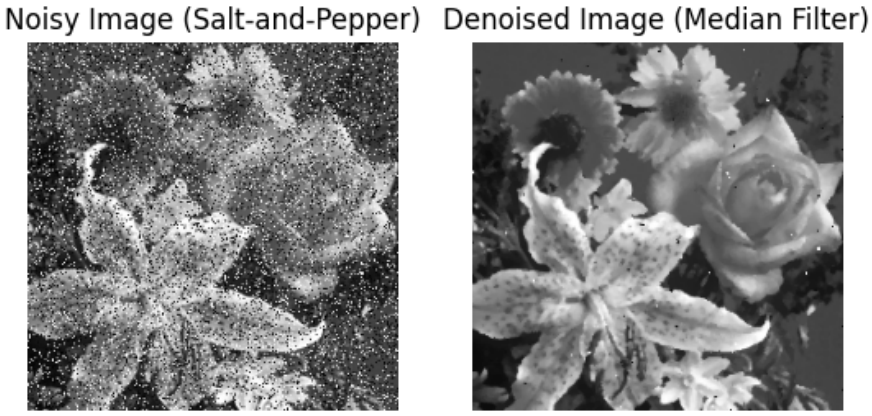
The median filter replaces each pixel with the median of its neighborhood. It removes noise more effectively than the box filter, especially isolated bright or dark specks.

Add Salt-and-Pepper Noise

- Add salt-and-pepper noise affecting **20%** of the pixels.
- Apply both filters again:
    - Moving average
    - Median



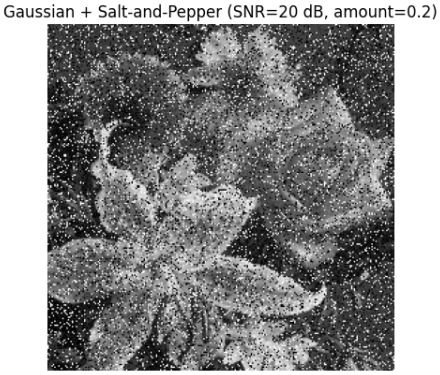




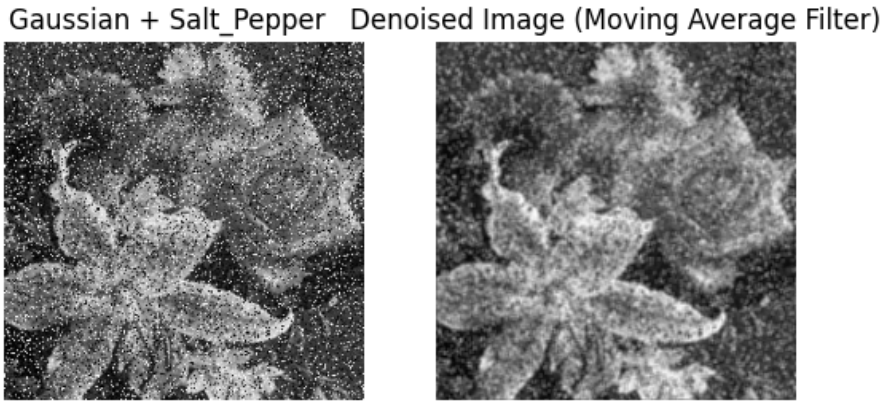
Median filter removes noise more effectively than the Moving Average filter.

Combine Both Noise Types

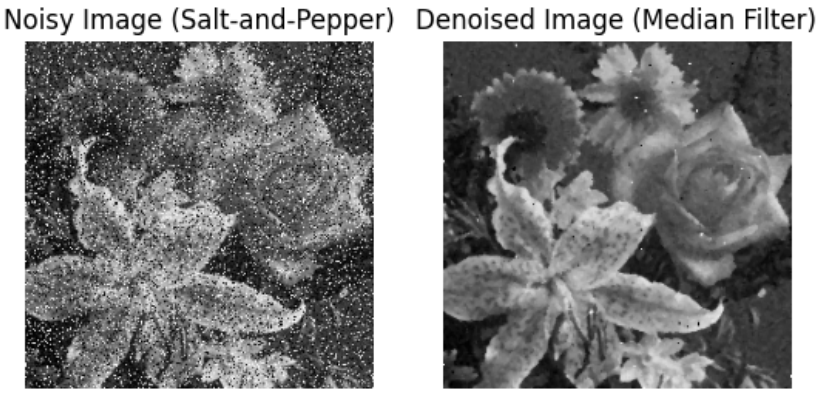
- Add **both Gaussian and salt-and-pepper noise** to the image.



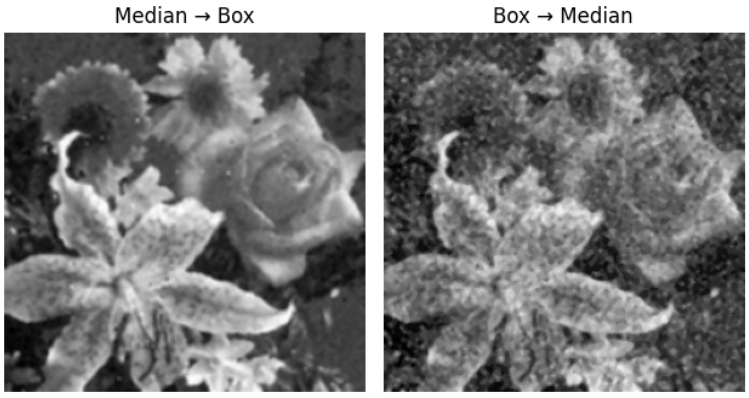
- Restore the image using:
    - a) “Box filter” = “Moving average filter” = “Mean filter”



    - b) Median filter



    - c) Sequential filtering (we choose the order of filters)



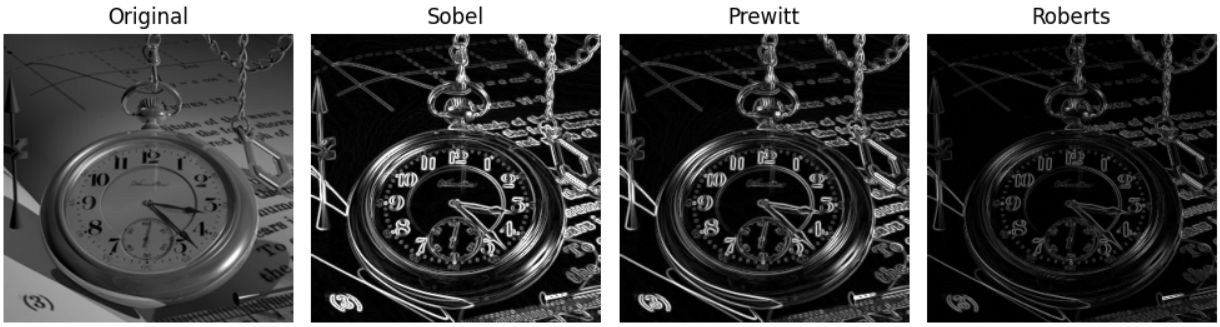
The median filter is more effective than the moving average, so it should be applied first in sequence to preserve sharpness.

Trade-off: median preserves sharpness but may lose subtle details; moving average smooths more uniformly but at the cost of edge clarity.

Task 3: Edge Detection

Apply Edge Detection Kernels

- Use **three different edge detection kernels** (e.g., Sobel, Prewitt, Roberts).
- Generate edge maps for each method.



Effect of kernel choice

Method	Kernel Size	Edge Type	Notes
Sobel	3×3	Thick, smooth edges	Robust, good general choice
Prewitt	3×3	Sharper edges	Slightly noisier
Roberts	2×2	Thin, fine edges	Sensitive to noise

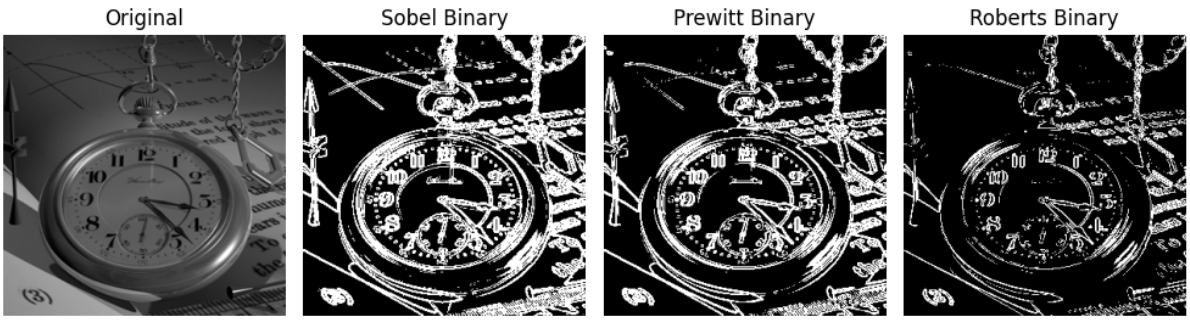
Sobel uses weighted gradients (larger emphasis on central pixels), which makes it more robust to noise and produces thicker, clearer edges, especially along vertical and horizontal directions.

Prewitt is simpler (uniform weights), so it responds more evenly but is also more sensitive to noise, resulting in noisier edge maps.

Roberts uses very small diagonal kernels, making it good for fine detail and diagonal edges, but it is also highly sensitive to noise and often produces scattered results.

Convert Edge Maps to Binary (this step is applicable when we perform the previous step with convolution function that do not provide directly binary map output)

Treshold = 50



Effect of thresholding

A low threshold keeps more edges but introduces noise (background and weak edges appear). A high threshold removes noise but loses fine details and weak edges.

For the clock image, Sobel gives the most meaningful edges. It clearly outlines the clock's circular frame, numbers. Balances noise suppression with edge clarity better than Prewitt and Roberts. Prewitt detects weaker gradients and adds noise. Roberts captures fine diagonals but produces fragmented, noisy maps.

Task 4: Template Matching

Perform Template Matching

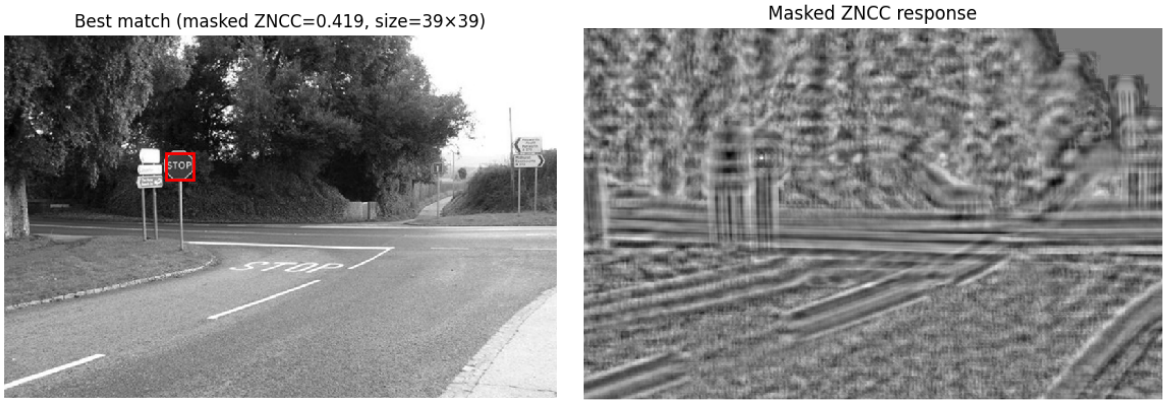
- Use **normalized cross-correlation** to find the location of the template within the original image.
- Visualize the match.


```
Street: (790, 520) | Template original: (1024, 1024)
```

Both images are converted to grayscale to simplify processing and then to NumPy arrays for numerical operations. Because the object may appear at different sizes, the template is resized to several scales (around ±30% of a base width). The resizing uses the Lanczos method, which keeps details sharp.

The main image is normalized by subtracting its mean and dividing by its standard deviation, so brightness and contrast differences do not affect matching. Finally, the code shows the normalized image and a few scaled templates to confirm the setup.

Now we performs template matching using a method called masked zero-mean normalized cross-correlation (masked ZNCC). It finds where a smaller image (the template) best matches a region of a larger image, while ignoring irrelevant parts of the template.



The function `conv2_valid_fft` computes convolution using the Fast Fourier Transform. This is a faster way to slide the template across the image, especially for large arrays. The function

reverses the template, transforms both images to the frequency domain, multiplies them, and converts the result back to the spatial domain.

The `masked_zncc` function computes a correlation map between the image and the template. It normalizes both so that differences in brightness or contrast do not affect the result. The mask defines which pixels of the template are meaningful and which should be ignored, such as uniform background areas. For every possible location, it measures how well the masked template pattern matches the image patch, producing correlation values between -1 and 1. Higher values mean better matches.

The `build_mask` function automatically creates a mask from the template. It keeps the darkest and brightest pixels (where the object has strong features) and discards mid-gray areas that belong to the background. This focuses the matching process on the distinctive parts of the object.

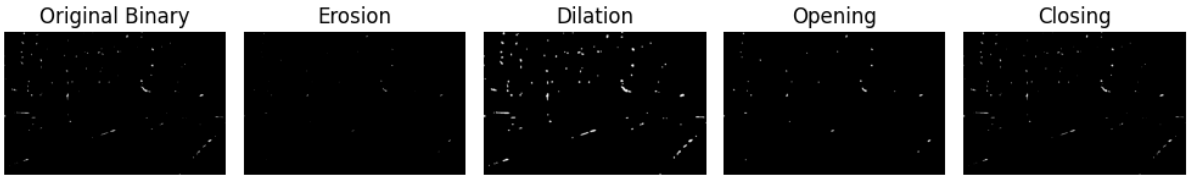
The loop runs this masked ZNCC process for several versions of the template at different scales. For each template size, it computes a correlation map, finds the location with the highest score, and keeps the best overall result. The position and size of the best match are saved.

The result draws a red rectangle on the main image to show where the template was found, and displays the correlation response map. The bright area in the response map indicates where the best match occurred.

Optional: Refine with Morphological Operations

- Apply morphological operations (e.g., `imerode`, `imdilate`, `imopen`, `imclose`) to clean up the correlation map or improve detection.
- Discuss how these operations affect the result.

Morphological operations modify shapes in binary images. They are based on a small matrix called a structuring element (or kernel), which slides over the image. Each of these frames (Original Binary, Erosion, Dilation, Opening, Closing) is a processed version of the NCC map. This map shows where, in the large image, the NCC algorithm found strong matches with the template.



- **Erosion** → removes small noise, but shrinks detections
- **Dilation** → enlarges detections, connects regions
- **Opening** → cleans isolated noise, keeps shape
- **Closing** → fills small gaps, strengthens object continuity

In this task, normalized cross-correlation was used to locate the template within a larger image. The method accurately detected the matching area, even with lighting or scale differences. Applying morphological operations helped clean the correlation map and improve the clarity of the final detection.