Cairo university

Faculty of Engineering

Systems and Biomedical Department

Computer Vision (SBE 3230)

Assignment 1

"Filtering and Edge Detection"

Name	B.N
Eman Abdelazeem	**12**
Hassnaa Hossam	**20**
Abdelrahman Alaa	**36**
Farha Elsayed	**3**

Supervised by: Dr. Ahmed Badawy

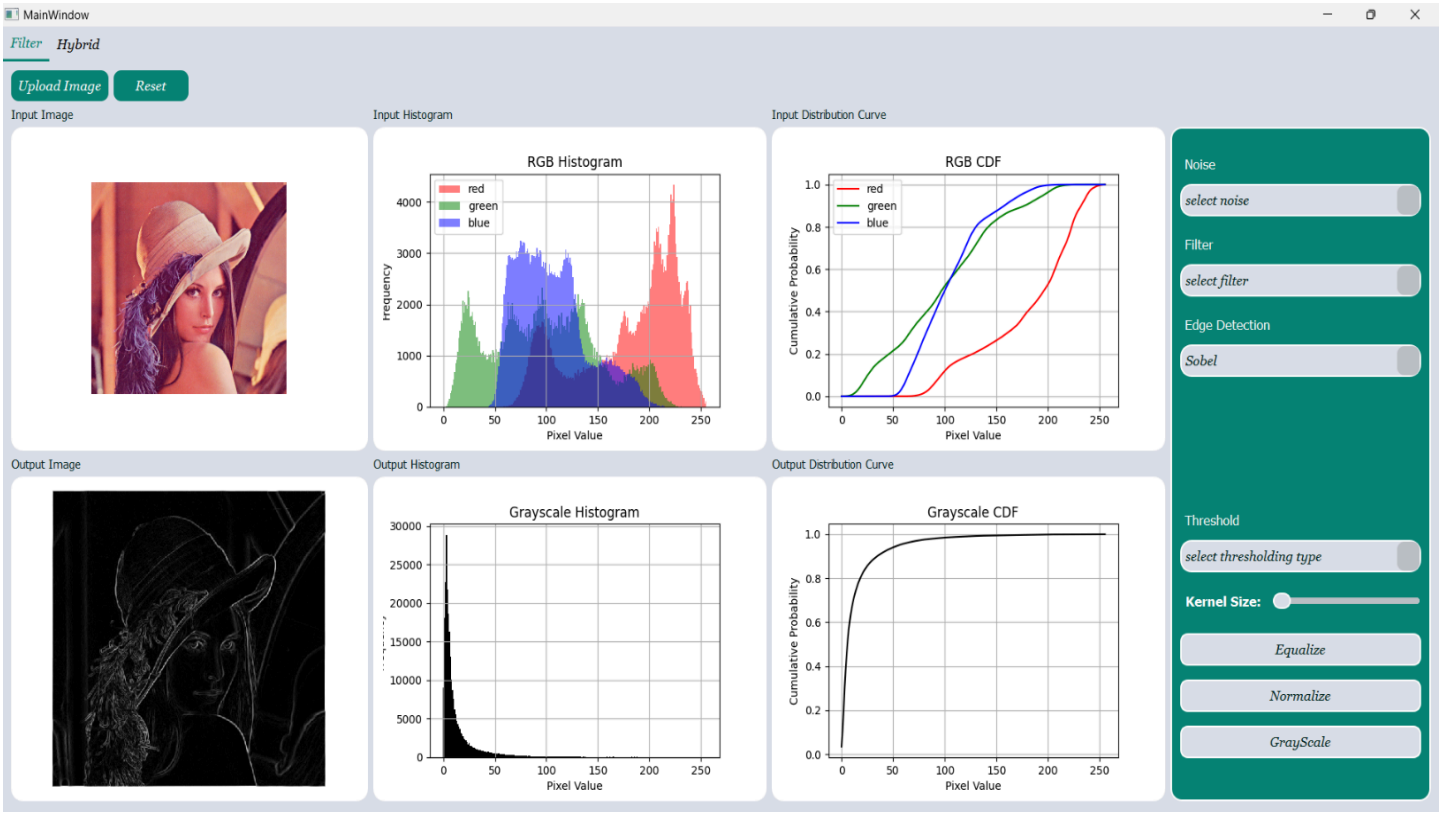
Submitted to: Eng. Yara Wael &

Eng. Omar Dawah

Introduction:

This report presents an image processing application developed using Python and PyQt5, offering a range of essential features, including noise addition, filtering, edge detection, histogram analysis, equalization, normalization, thresholding, frequency domain filtering, and creating hybrid images.The application provides users with an interactive platform to enhance, analyze, and manipulate images effectively.

UI:



Features

1. Noise

1- Salt and Pepper Noise

Salt & Pepper noise randomly alters pixel values in an image by setting some pixels to either white (255) or black (0). This simulates noise commonly seen in transmission errors or corrupted images.

Process:

1. Compute the number of salt and pepper pixels using the given probability and ratio.

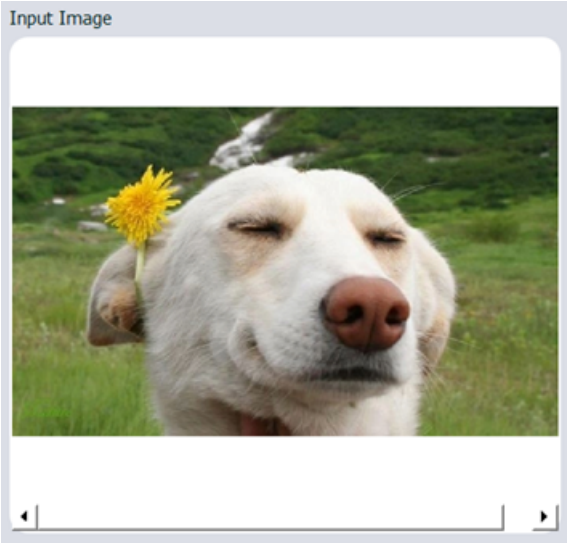
2. Generate random pixel coordinates for noise placement.

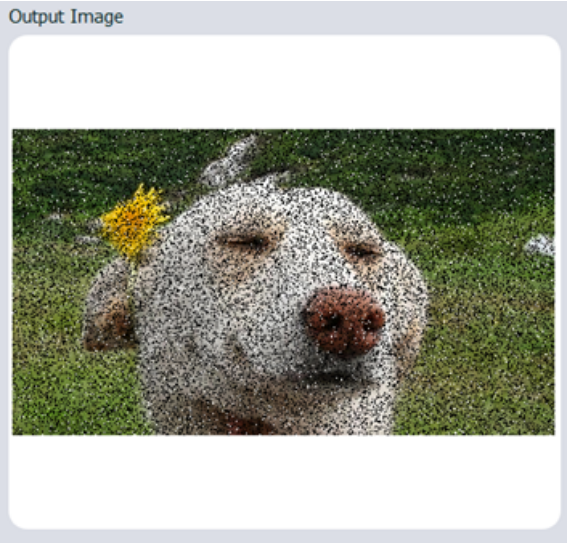
3. Modify the pixel values:

- 'Salt' pixels are set to 255 (white).

- 'Pepper' pixels are set to 0 (black).

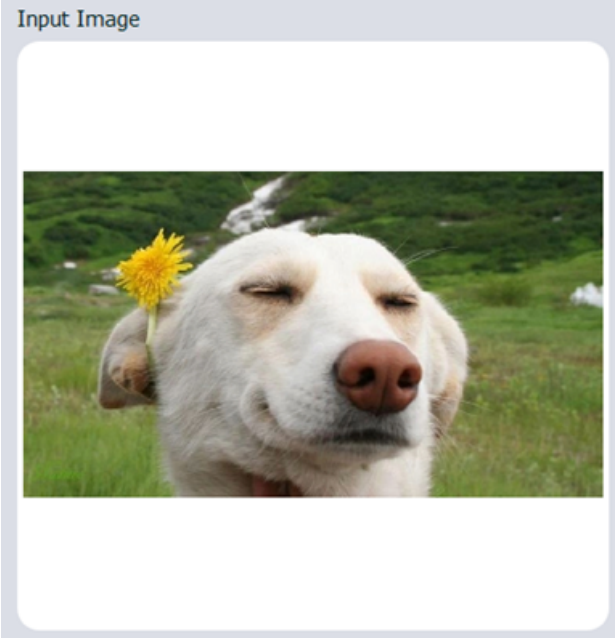
4. The process is applied differently for grayscale (2D) and colored (3D) images.

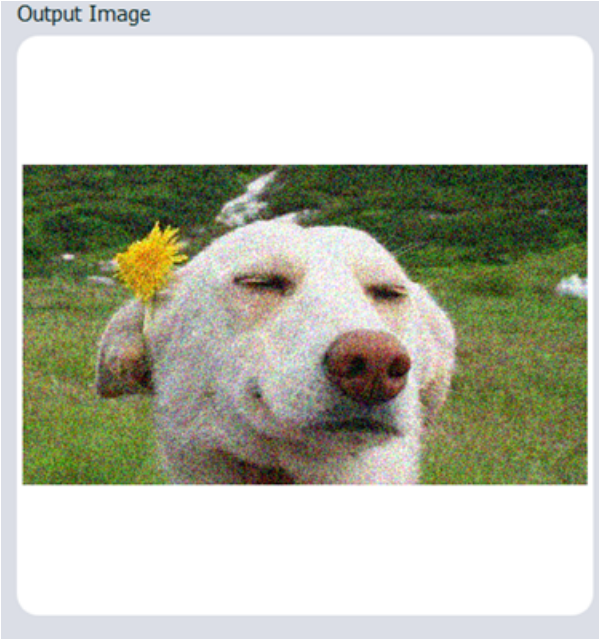




2- Uniform Noise

The approach for adding Uniform noise involves generating random values uniformly distributed within a specified range and adding them to the original image. A matrix noise with the same size and data type as the input image is initialized to hold the uniform noise. The ‘np.random.randint()’ function provided by numPy is used to generate random values uniformly distributed between 0 and 50. These random values are stored in the noise matrix. Next, the generated uniform noise matrix noise is added to the duplicate image result. This addition operation is performed element-wise, meaning each pixel in the noise matrix is added to the corresponding pixel in the duplicate image. Finally, the resulting image result, now containing the added uniform noise, is returned. This process introduces random variations throughout the image, simulating the appearance of uniform noise.

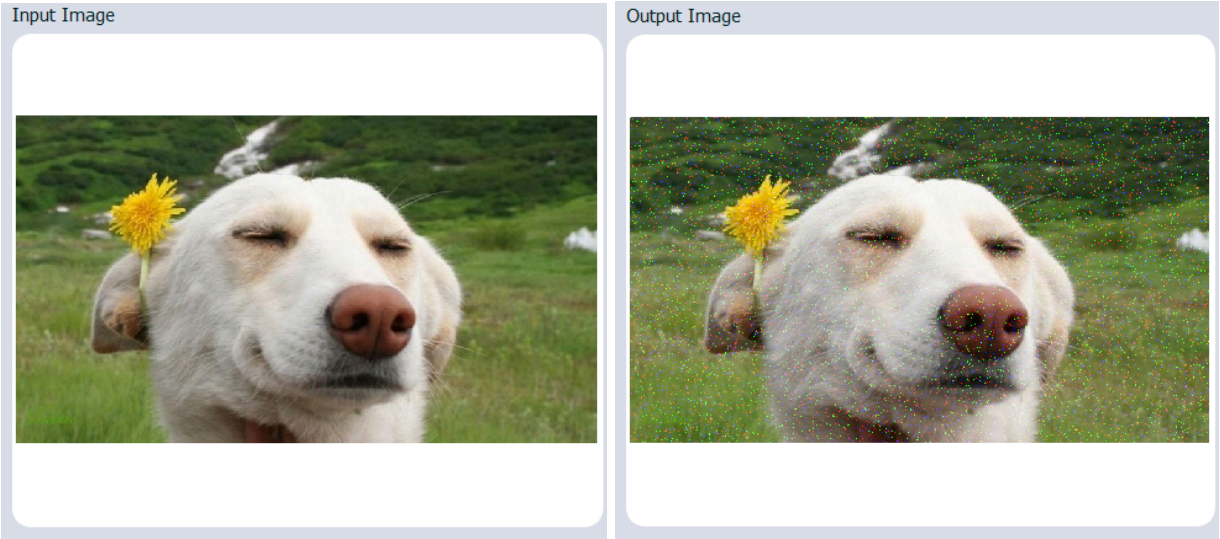




3- Gaussian noise

Adding Gaussian noise involves generating random values from a Gaussian distribution with specified mean and standard deviation parameters and adding them to the original image. A noise matrix gauss is generated using np.random.normal(mean, sigma, self.image_array.shape), where values follow a normal distribution with the specified mean and sigma. The noise is converted to uint8 for compatibility with the image format. The cv2.add() function is used to

add the noise to the image element-wise, ensuring proper handling of pixel overflow.



Observations:

Noise Type	**Salt & Pepper**	**Uniform**	**Gaussian**
Parameters	Noise density (probability of black/white pixels appearing) and The ratio of white to black.	Minimum and maximum values defining the range.	Mean and standard deviation
Effect on Image	Causes sudden, high-intensity changes at some pixels. (black&white) pixels.	Introduces a consistent level of noise across the image.	Blends more naturally with the image but adds variance.
Filtering Approach	Best removed using median filters.	Can be reduced using average filters or low-pass filters.	Best reduced using Gaussian filters.

2. Filters:

1- Median Filter:

The Median filter is a non-linear filtering technique that reduces noise while preserving edges. It is particularly effective against Salt & Pepper noise.

Implementation Steps:

1. The image is padded to handle edge pixels.
2. A sliding window (kernel) moves across the image.
3. For each region covered by the kernel:
    - All pixel values are collected.
    - The median value is computed and assigned to the central pixel.
4. The final result is a noise-free image with preserved edges, making it useful for medical and biometric images.

Salt & Pepper noisy image before and after applying a Median filter
Kernel size: 5x5, Salt & Pepper Noise(probability: 0.1, ratio:0.1)

2- Gaussian Filter:

The Gaussian filter is a weighted averaging filter that smooths the image while reducing high-frequency noise. Unlike the Average filter, it gives higher importance (weight) to pixels near the center of the kernel.

Implementation Steps:

1. Generating a Gaussian Kernel (get_gaussian_kernel)
    - Computes values using the Gaussian function.
      $G(x,y) = 1 / (2\pi\sigma^2) * \exp(-(x^2+y^2) / 2\sigma^2))$
    - Gives higher weight to center pixels for natural blurring.
    - Normalizes the kernel to maintain brightness.
2. Applying the Kernel (apply_kernel)
    - Convolves the kernel with the image.
    - Replaces each pixel with a weighted sum of its neighbors.

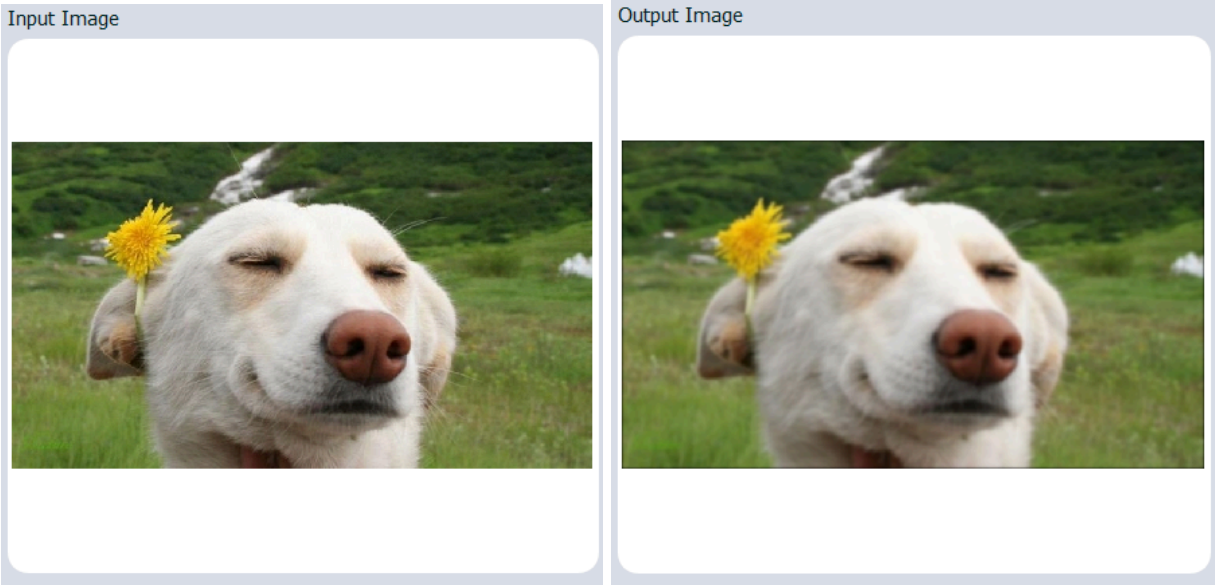
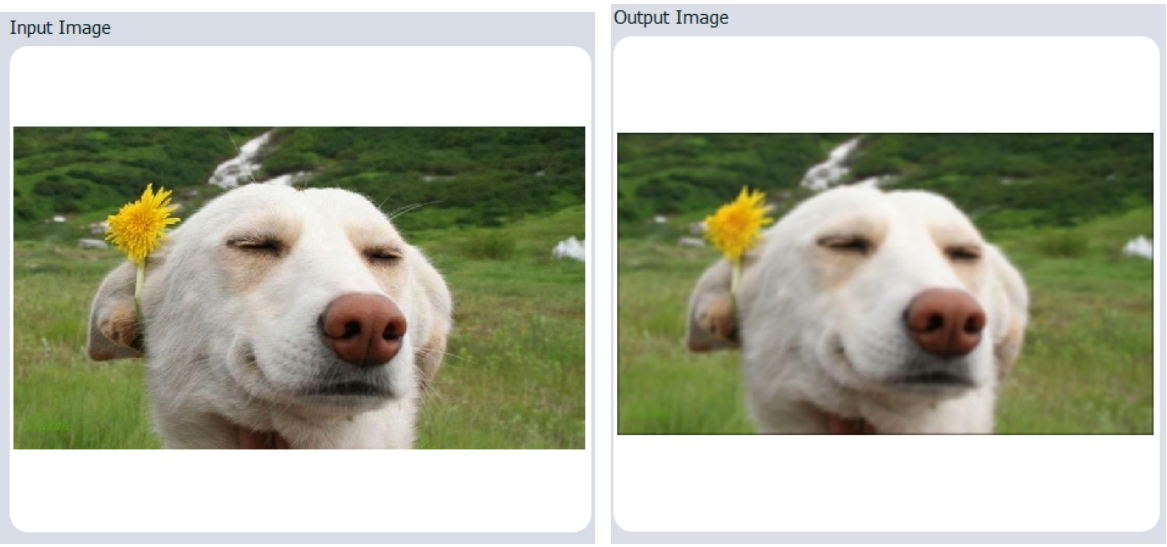


Image before and after applying a Gaussian filter
Kernel size: 7x7, Gaussian Filter (sigma: 2)

3- Average Filter:

The Average filter is a simple blurring technique that replaces each pixel's value with the average of its surrounding pixels. This helps in reducing noise and smoothing the image. The kernel is a small square matrix (of size kernel_size × kernel_size) that slides over the image. Each element in the kernel has the same value This ensures that all neighboring pixels contribute equally to the final averaged pixel value.



Kernel size: 7x7, Average Filter

Observations:

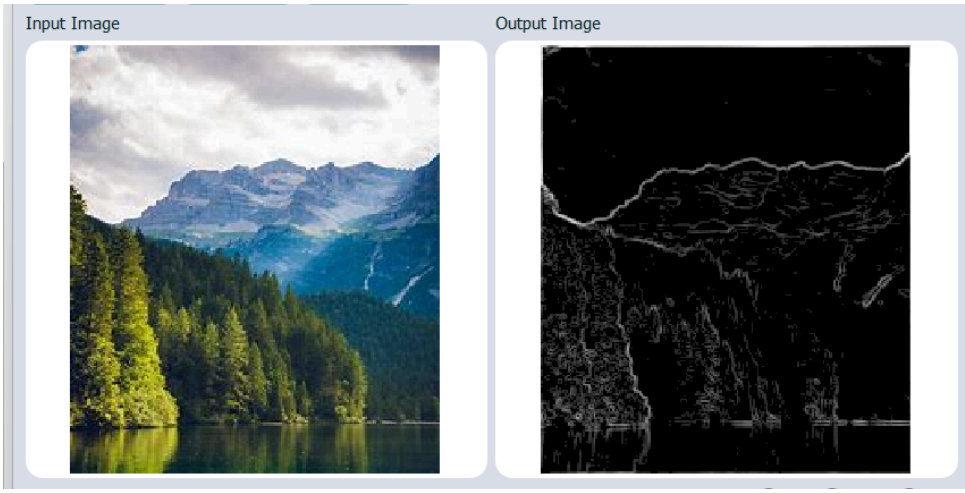
Filter Type	**Average**	**Gaussian**	**Median**
Best For	Basic noise reduction, smooths images.	Smooth blurring, reducing Gaussian noise.	Removing salt-and-pepper noise.
Drawbacks	Blurs edges, reduces sharpness.	Can distort fine details.	Slightly slower, can still blur edges.

3. Edge Detection

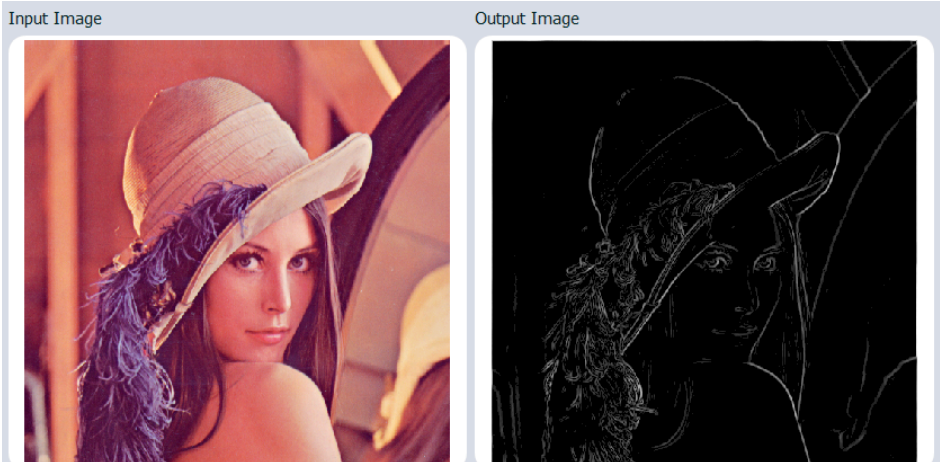
is a technique used to identify boundaries within an image by detecting changes in intensity. It helps in extracting significant features such as object outlines , The input image is converted to grayscale if it is colored and The selected edge detection kernel is applied using a convolution operation and then The magnitude of the gradient is computed after that The output is normalized and converted to a displayable format.

We implemented four well-known edge detection filters:

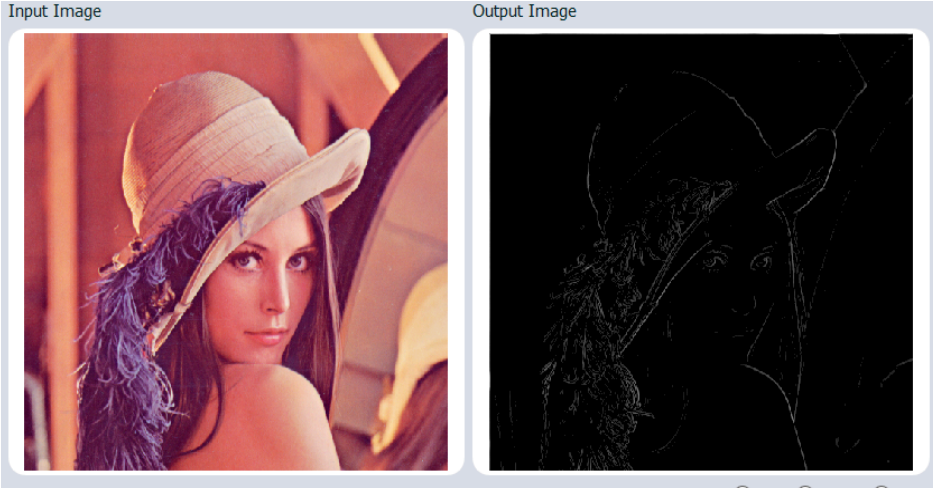
- Sobel Filter: Uses two 3×3 kernels (horizontal and vertical) to compute intensity gradients in both directions.



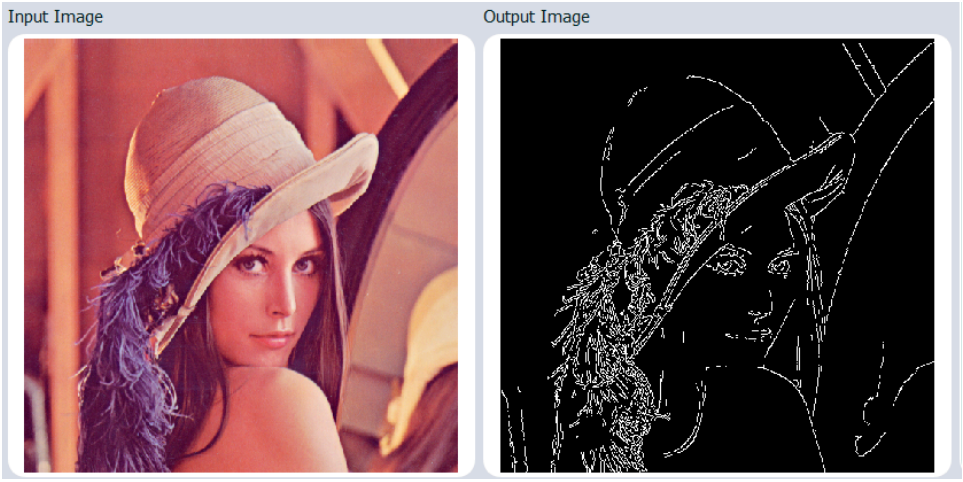
- Prewitt Filter: Similar to Sobel but with different weight distribution.



- Roberts Filter: A simple 2×2 kernel for detecting edges and sensitive to noise



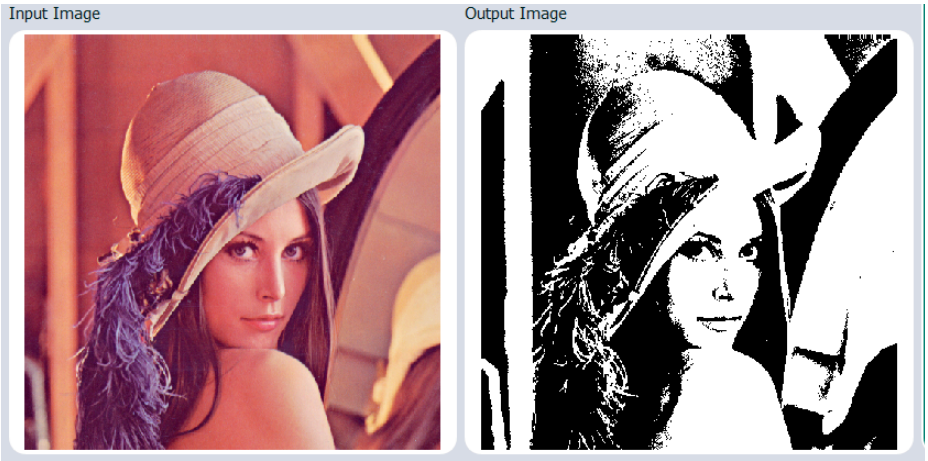
- Canny Filter: A multi-stage process that involves noise reduction, gradient calculation, non-maximum suppression, and hysteresis thresholding.



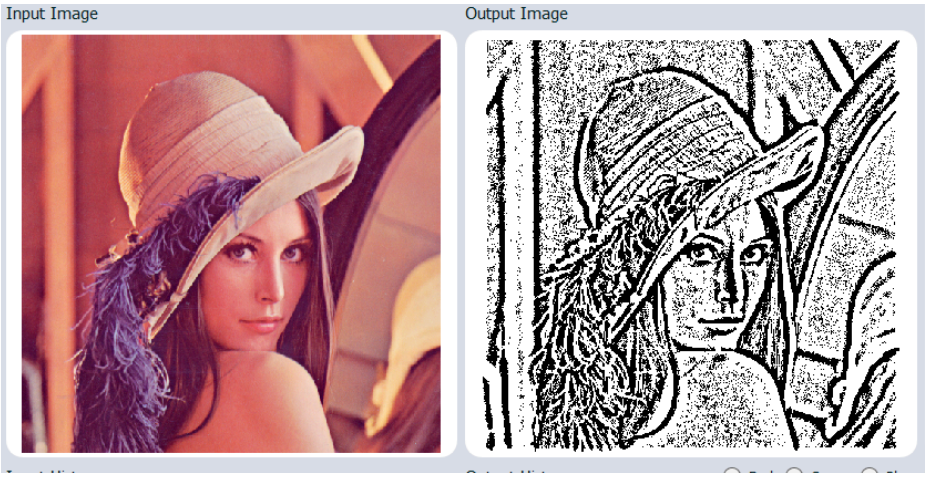
Thresholding

Thresholding is a technique for segmenting an image into foreground and background by setting intensity values above a certain threshold to one category (white) and values below to another category (black).

- **Global Thresholding**: A fixed threshold value of **128** is applied across the entire image. The algorithm iterates over all pixels, and any pixel with an intensity greater than 128 is set to **255** (white), while any pixel with an intensity lower than 128 is set to **0** (black).



- **Local Thresholding**: Instead of using a fixed value, a sliding window moves across the image. For each pixel, the mean intensity of the surrounding pixels within the window is calculated and used as a threshold. This allows the threshold to adapt dynamically to different regions of the image.



Normalization

The normalization process in this application ensures consistent brightness and contrast by adjusting pixel intensity values. The implementation follows these steps:

1. **Grayscale Conversion:** If the image is in RGB format, it is converted to grayscale to simplify intensity-based processing.
2. **Finding Intensity Range:** The minimum and maximum pixel intensity values in the grayscale image are identified.
3. **Normalization Process:** Each pixel's intensity is adjusted to span the full brightness range, enhancing contrast and improving visibility. The pixel values are then converted to an 8-bit format for proper display.

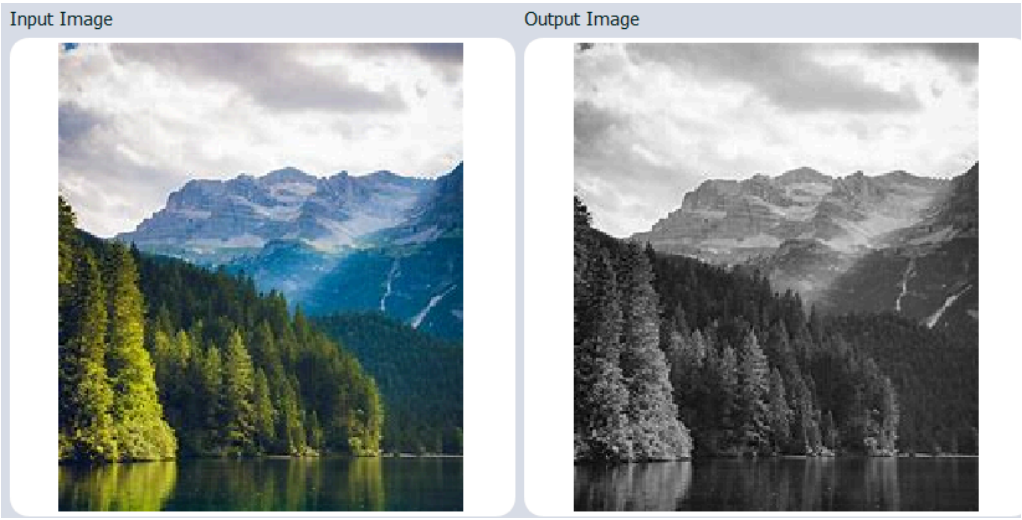


Image before and after Normalization

Equalization

Histogram equalization enhances image contrast by redistributing pixel intensity values for better visibility, especially in low-contrast images. The process involves computing the histogram, generating and normalizing the cumulative distribution function (CDF), and using it to adjust pixel values. This ensures a more balanced intensity distribution, improving image clarity.

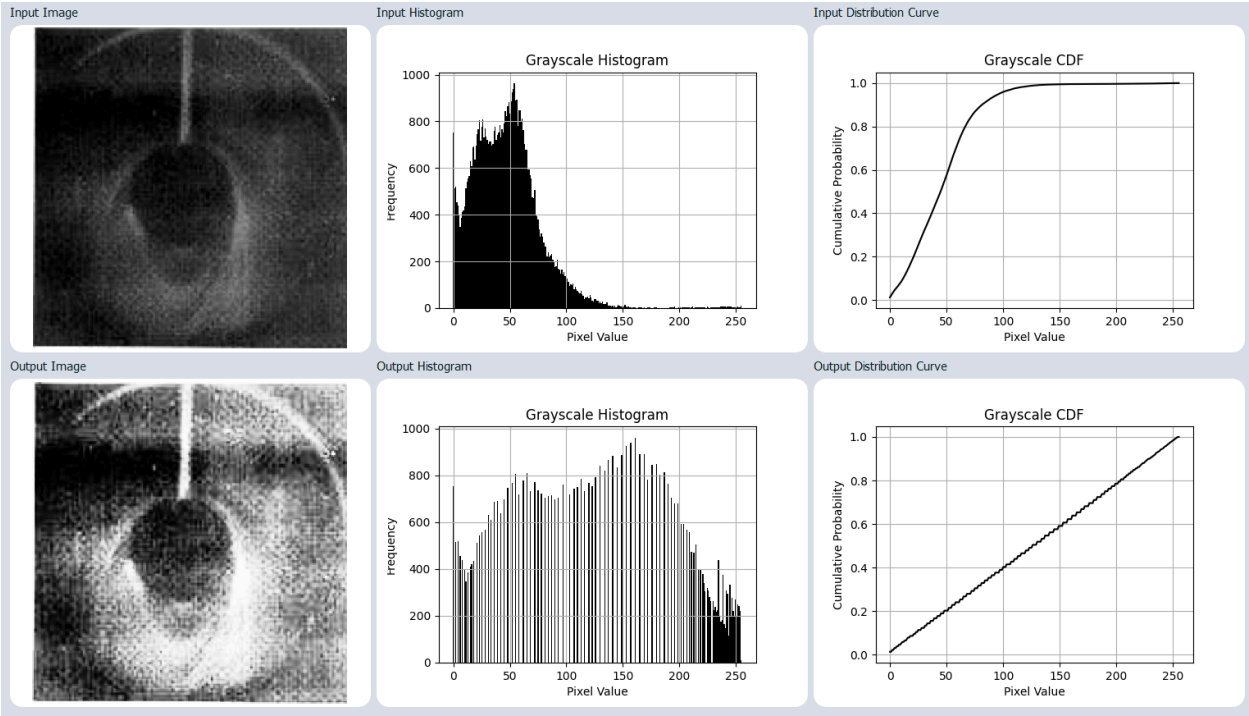


Image before and after histogram equalization

Histogram and Cumulative Distribution Function:

For each input image, its histogram and cumulative distribution function (CDF) were plotted to visualize the pixel intensity distribution and cumulative probability. The analysis was performed for both grayscale and RGB images. In the case of grayscale images, a single histogram and CDF were plotted since they contain only one intensity channel. For RGB images, histograms and CDFs for the red, green, and blue channels were plotted in the same figure, allowing for a comprehensive understanding of color distribution.

Additionally, whenever modifications such as edge detection, filtering, or histogram equalization were applied, the processed image was analyzed in the same manner. The histograms and CDFs of the modified images were generated to

observe changes in intensity distribution and contrast. This approach provides a clear visualization of how different transformations impact image characteristics, helping to assess the effectiveness of various image processing techniques.

Histogram and Cumulative Distribution Function (CDF) Computation

To analyze the intensity distribution of images, functions were implemented to compute both the histogram and the cumulative distribution function (CDF) for grayscale and RGB images. These functions utilize the OpenCV and NumPy libraries for efficient computation.

Histogram Computation

The histogram represents the frequency distribution of pixel intensities in an image. It provides insights into the image's contrast and brightness. The function `compute_histogram()` determines the histogram using OpenCV's `cv2.calcHist()` method:

- For **grayscale images**, a single histogram is computed.
- For **RGB images**, the histogram is calculated separately for each of the three color channels (Red, Green, and Blue).

The histogram H(i)H(i) at intensity ii is mathematically defined as:

$$H(i)=\sum x,y\ [\ \delta(I(x,y)-i)]$$

where I(x,y)I(x,y) represents the intensity at pixel (x,y)(x,y), and δ(k)\delta(k) is 1 if k=0k = 0, otherwise 0.

CDF Computation

The CDF provides a cumulative representation of pixel intensities, which is particularly useful for image enhancement techniques such as histogram equalization. The function `compute_CDF()` calculates the CDF by summing the histogram values and normalizing the result.

- If the image is **grayscale**, a single CDF is computed.
- If the image is **RGB**, a separate CDF is calculated for each color channel.

The CDF at intensity ii is given by:

$$CDF(i)=\sum_{j=0 \rightarrow i} [\ H(j)\ ]$$

Implementation Details

The histogram and CDF computations rely on:

- **OpenCV (`cv2.calcHist()`)** for histogram generation.
- **NumPy (`cumsum()`)** for efficient cumulative summation.
- **Normalization** ensures the CDF values are scaled between 0 and 1 for consistency in analysis.

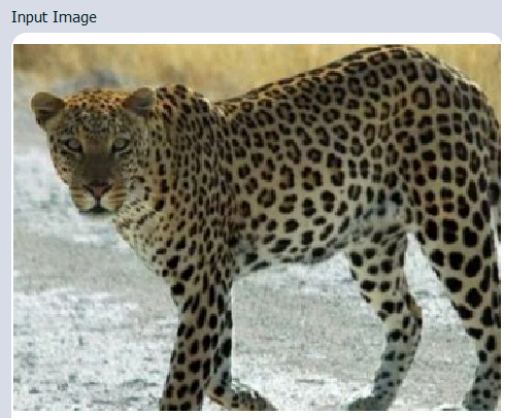
By computing these functions for both original and modified images, the effects of various image processing techniques, such as edge detection, filtering, and equalization, can be visualized and compared effectively.

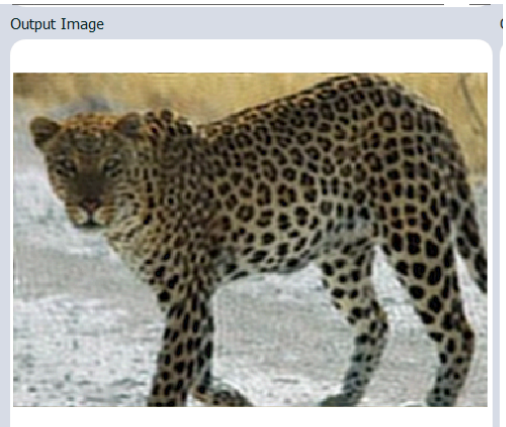
Frequency Domain Filters:

Low-Pass and High-Pass Filters

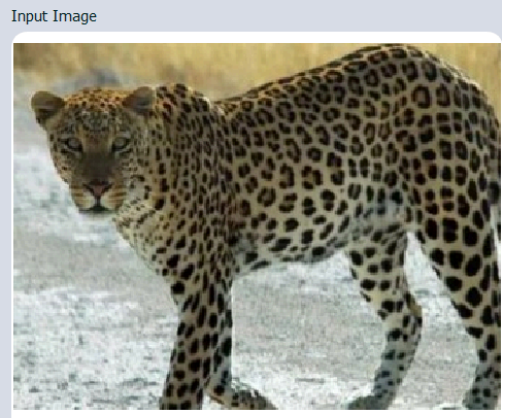
Images are converted to the frequency domain using the Discrete Fourier Transform (DFT), allowing for targeted filtering. Circular masks are applied to modify the frequency components:

- Low-pass filtering retains low-frequency content within a specified radius, eliminating fine details for a smoothing effect.





- High-pass filtering preserves high-frequency components beyond the defined radius, emphasizing edges and sharp features.
  Once the filtering process is complete, the Inverse DFT is used to transform the modified frequency spectrum back into the spatial domain, reconstructing the processed image.



Hybrid Image :

Hybrid images combines the low-frequency components of one image with the high-frequency components of another image to create an interesting perceptual effect .The fundamental idea behind hybrid images is to create an image that appears as one image when viewed from a distance but transforms into another image when viewed up close , This effect is achieved by combining the low-frequency information (smooth features, overall structure) of one image with the high-frequency information (fine details, texture) of another image.

For Image A, extract the low-frequency components using a low-pass filter and extract the high-frequency components using a high-pass filter.

For Image B, do the same: extract low-frequency components with a low-pass filter and high-frequency components with a high-pass filter.

Combine the low-frequency components of Image A with the high-frequency components of Image B to create the hybrid image.