E969205: Signal Processing System Design
Indian Institute of Technology, Kharagpur

Image Processing and Filtering

Anirvan Krishna | Roll no. 21EE38002

1. Objective

The objective of this experiment is to understand and implement various image filtering techniques on a conventional 2D grayscale image. Students will learn to apply low-pass and high-pass filters, Gaussian filter, Laplacian of Gaussian (LoG) filter, Canny edge detection, and max pooling using both scratch implementations and inbuilt functions.

2. Noise Addition

Salt and Pepper noise is a type of image noise that randomly alters the pixel values of an image to either the maximum (white) or the minimum (black) intensity, resembling salt and pepper sprinkled over the image. This noise is often caused by sharp, sudden disturbances in the image signal, such as faulty memory locations or analog-to-digital conversion errors.

The mathematical model for Salt and Pepper noise is represented as:

$$\mathbf{I}_{\text{noisy}}(x,y) = \begin{cases} 0, & \text{with probability } p_1 \\ 255, & \text{with probability } p_2 \\ \mathbf{I}(x,y), & \text{with probability } 1-p_1-p_2 \end{cases}$$

where $\mathbf{I}(x,y)$ is the original intensity at pixel location (x,y), and p_1 and p_2 are the probabilities of the pixel being set to 0 (pepper) and 255 (salt), respectively.

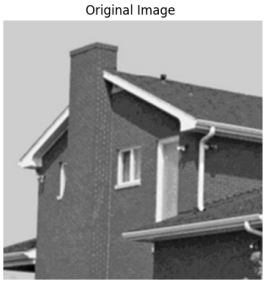

Fig. 1. Salt-Pepper Noise-I

Fig. 2. Salt-Pepper Noise-II

3. Low Pass Filtering

Filtering is an essential image processing technique used to enhance images, reduce noise, and perform various other transformations. Three commonly used filters are Mean, Gaussian, and

Median filters. Each filter operates differently and is suitable for different types of noise and image processing tasks.

3.1. Mean Filtering—Mean filtering is a simple, linear smoothing technique that reduces noise in an image. The filter replaces each pixel value with the average (mean) of its neighboring pixel values, including itself. This process smooths the image by reducing intensity variations between adjacent pixels, effectively reducing noise but also blurring the image.

The mathematical representation of Mean Filtering is:

$$\mathbf{I}_{\text{filtered}}(x,y) = \frac{1}{N} \sum_{(i,j)\in\mathcal{N}} \mathbf{I}(x+i, y+j)$$

where $\mathbf{I}(x,y)$ is the intensity of the pixel at location (x,y), $\mathcal{N}$ represents the neighborhood of the pixel (usually a square window centered around (x,y)), and N is the number of pixels in the neighborhood. In this example we are using a neighbourhood of size 3×3. Therefore, the Mean Filtering kernel is represented as:

$$\mathbf{K}_{\text{mean}} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

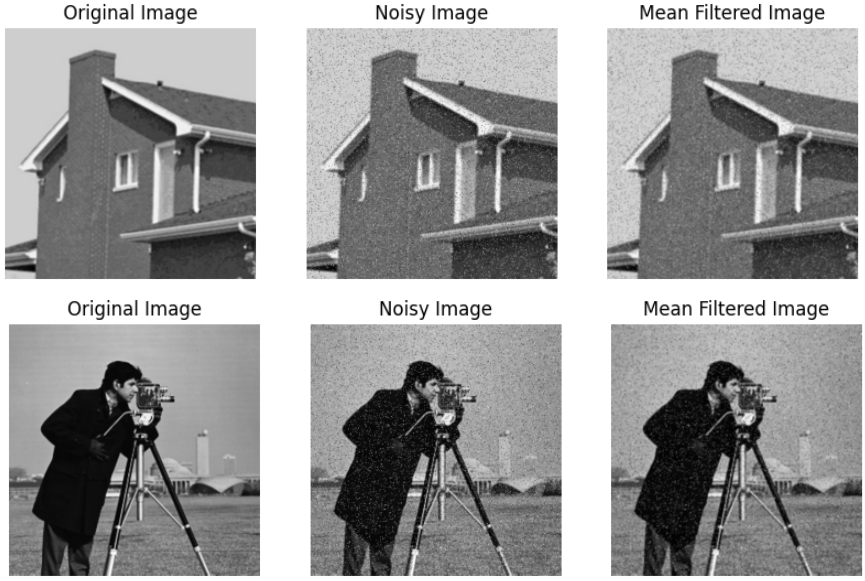

Fig. 3. Mean Filter Application Examples

3.2. Gaussian Filtering—Gaussian filtering is a weighted mean filter where the weights are determined by a Gaussian distribution. The filter is used to smooth images and reduce noise while preserving edges better than the Mean filter. The Gaussian filter applies a convolution between the image and a Gaussian kernel, giving more weight to the central pixels in the neighborhood.

The Gaussian function used to create the kernel is given by:

$$G(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

where σ is the standard deviation of the Gaussian distribution, determining the extent of

smoothing. The filtered image is obtained by convolving the original image with this Gaussian kernel. In this case we are using a Gaussian Kernel of size 3×3:

$$\mathbf{K}_{\text{gaussian}} = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

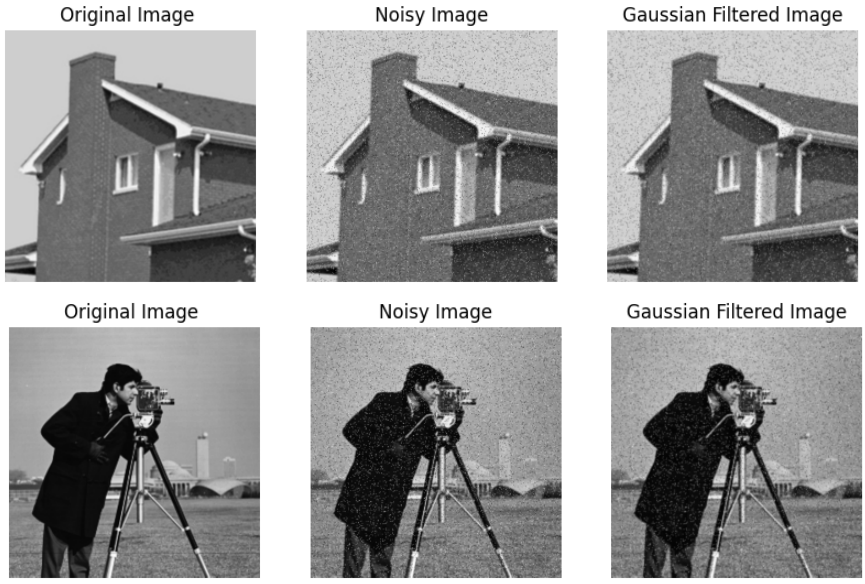

Fig. 4. Application of Gaussian Filter

3.3. Median Filtering—Median filtering is a non-linear filter that replaces each pixel value with the median value of its neighboring pixels. This filter is particularly effective for removing Salt and Pepper noise, as it preserves edges better than linear filters like Mean filtering.

The Median Filter operation can be expressed as:

$$\mathbf{I}_{\text{filtered}}(x,y) = \text{median}\left(\mathbf{I}(x+i, y+j)\right) \quad \text{for} \quad (i,j) \in \mathscr{N}$$

where $\mathscr{N}$ is the neighborhood of the pixel (x,y). In this experiment we are using a kernel size of 3×3 for the median filter.

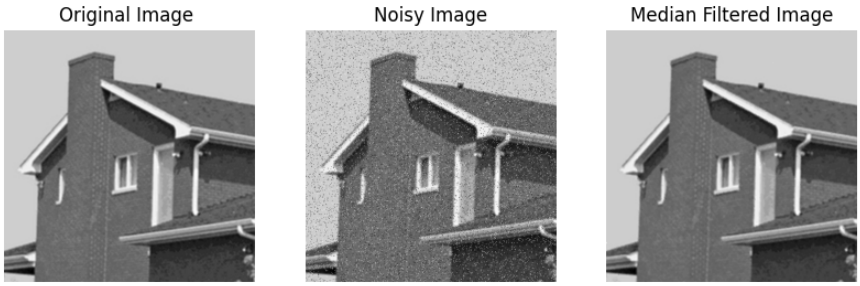

Fig. 5. Example-1: Application of Median Filter

Peak Signal-to-Noise Ratio (PSNR) : Peak Signal-to-Noise Ratio (PSNR) is a metric used to evaluate the quality of a reconstructed or filtered image compared to its original version. It measures the ratio between the maximum possible power of a signal (image) and the power of corrupting noise. Higher PSNR values indicate better image quality.

Fig. 6. Example-2: Application of Median Filter

The formula for PSNR is:

$$\text{PSNR} = 10\log_{10}\left(\frac{L^2}{\text{MSE}}\right)$$

where L is the maximum pixel intensity value (usually 255 for 8-bit images) and MSE is the Mean Squared Error between the original and filtered images. For the noisy house image, PSNR = 39.130 dB. For the noisy cameraman image, PSNR = 39.110 dB

Table 1. PSNR Values for Different Filters

Filter Type	**PSNR house(in dB)**	**PSNR Cameraman (in dB)**
Mean Filter	32.061	30.974
Gaussian Filter	31.541	31.208
Median Filter	43.898	38.978

The given table discusses the Peak Signal to Noise Ratio (PSNR) for application of different filtering techniques. We conclude that Median Filter provides the best image quality and filtering after application.

4. High Pass Filtering and Applications in Edge Detection

Laplacian, Sobel, and Prewitt filters are commonly used edge detection techniques in image processing. These filters highlight regions of an image where there are significant changes in intensity, which correspond to edges. Each filter has its own unique method of calculating the intensity changes.

4.1. Laplacian Filter—The Laplacian filter is a second-order derivative filter used to detect edges by calculating the second derivatives of the image. It highlights regions where the intensity changes rapidly and is sensitive to noise. The Laplacian filter is often used in conjunction with a smoothing filter (like Gaussian) to reduce noise before edge detection.

The Laplacian operator in 2D is defined as:

$$\nabla^2\mathbf{I}(x,y) = \frac{\partial^2\mathbf{I}(x,y)}{\partial x^2} + \frac{\partial^2\mathbf{I}(x,y)}{\partial y^2}$$

The corresponding convolution kernel (for a common discrete Laplacian) is:

$$\mathbf{L} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

This kernel is convolved with the image to produce the Laplacian-filtered image.

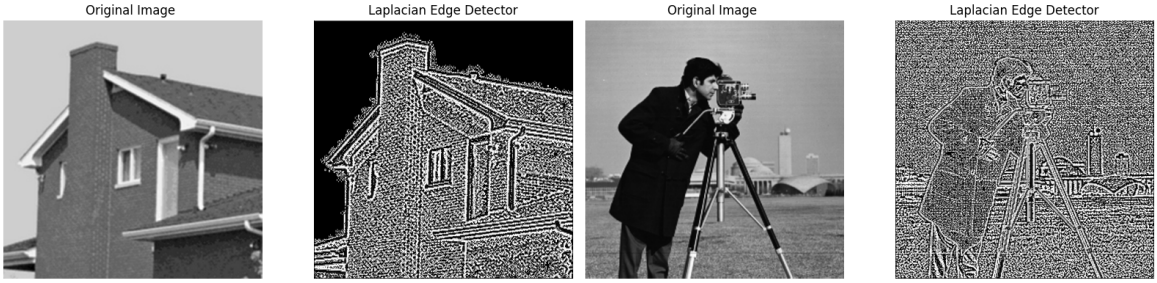

Fig. 7. Edge Detection using Laplacian Filtering

4.2. Sobel Filter—The Sobel filter is a first-order derivative filter used for edge detection, particularly to detect vertical and horizontal edges. It uses two 3x3 convolution kernels to calculate the gradients in the x and y directions.

The Sobel operator kernels are defined as:

$$\mathbf{S_x} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{S_y} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

The gradient magnitude at each pixel is then calculated as:

$$\mathbf{G}(x,y) = \sqrt{\mathbf{G_x}^2 + \mathbf{G_y}^2}$$

where $\mathbf{G_x}$ and $\mathbf{G_y}$ are the gradients in the x and y directions, respectively.

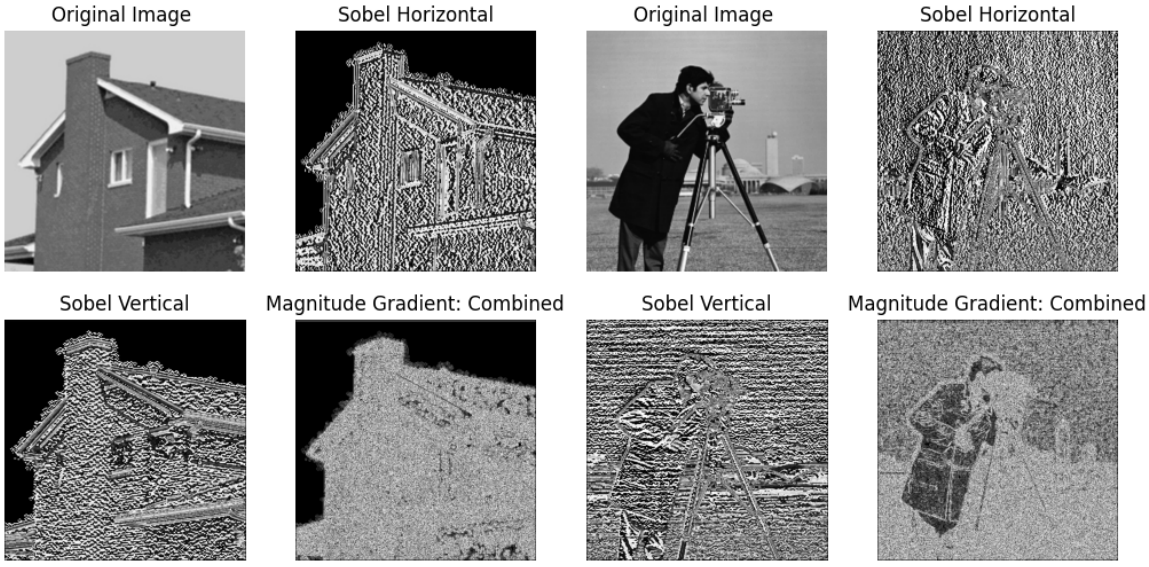

Fig. 8. Edge Detection using Sobel Kernel

4.3. Prewitt Filter—The Prewitt filter is similar to the Sobel filter but with different convolution kernels. It also calculates the gradient of the image intensity to find edges but uses simpler kernels.

The Prewitt operator kernels are defined as:

$$\mathbf{P_x} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{P_y} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

The gradient magnitude is similarly calculated as:

$$\mathbf{G}(x,y) = \sqrt{\mathbf{G_x}^2 + \mathbf{G_y}^2}$$

where $\mathbf{G_x}$ and $\mathbf{G_y}$ are the gradients in the x and y directions, respectively.

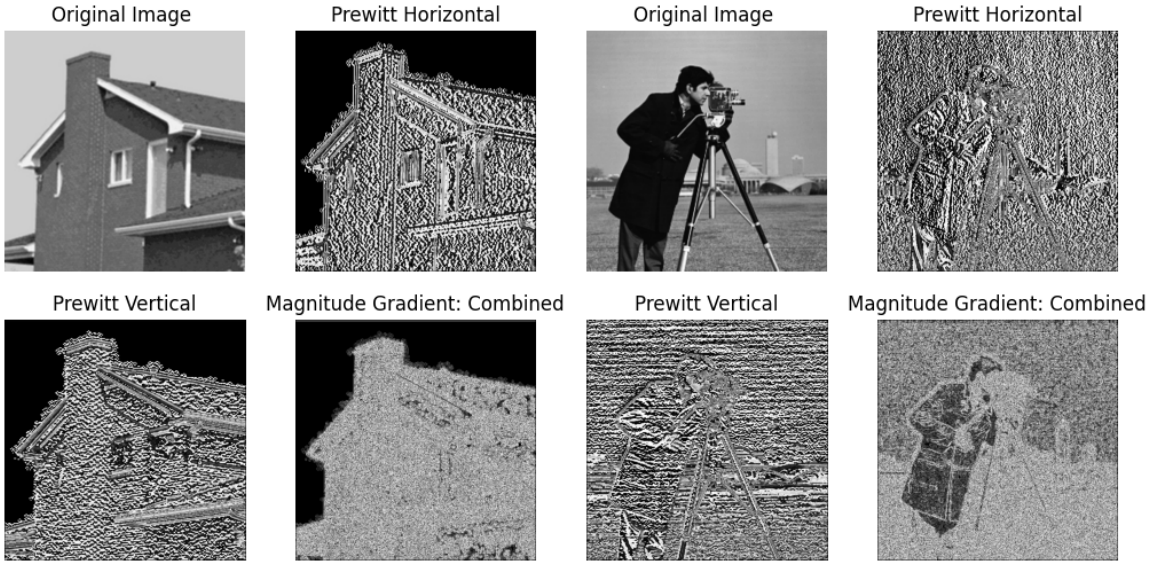

Fig. 9. Edge Detection using Prewitt Kernel

5. Advanced Edge Detection Techniques

Edge detection is a crucial technique in image processing for identifying the boundaries of objects within an image. The Canny and Laplacian of Gaussian (LoG) edge detectors are two widely used methods for detecting edges with high accuracy and efficiency.

5.1. Canny Edge Detection—Canny Edge Detection is a multi-stage algorithm designed to detect a wide range of edges in images. It is widely regarded as one of the most effective edge detection algorithms. It involves these following steps:

(1) Noise Reduction:
- The image is first smoothed using a Gaussian filter to reduce noise and spurious gradients.

(2) Gradient Calculation:
- The gradient intensity and direction are computed using derivative filters, often Sobel operators.

- The gradient magnitude G and direction θ are then computed as:

$$G = \sqrt{G_x^2 + G_y^2} \quad \text{and} \quad \theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

(3) Non-Maximum Suppression:
- Thin out the edges by suppressing non-maximum pixels in the gradient direction.

(4) Double Thresholding:
- Apply high and low thresholds to determine strong and weak edges.

(5) Edge Tracking by Hysteresis:
- Final edges are determined by suppressing all edges that are not connected to strong edges.

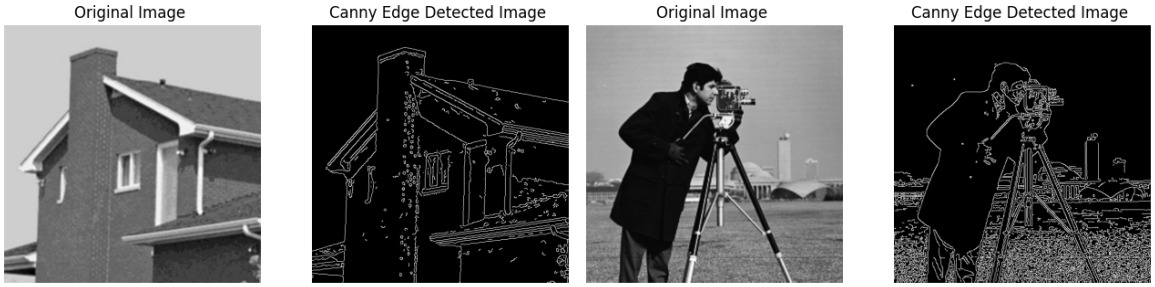

Fig. 10. Edge Detection using Canny Edge Detector

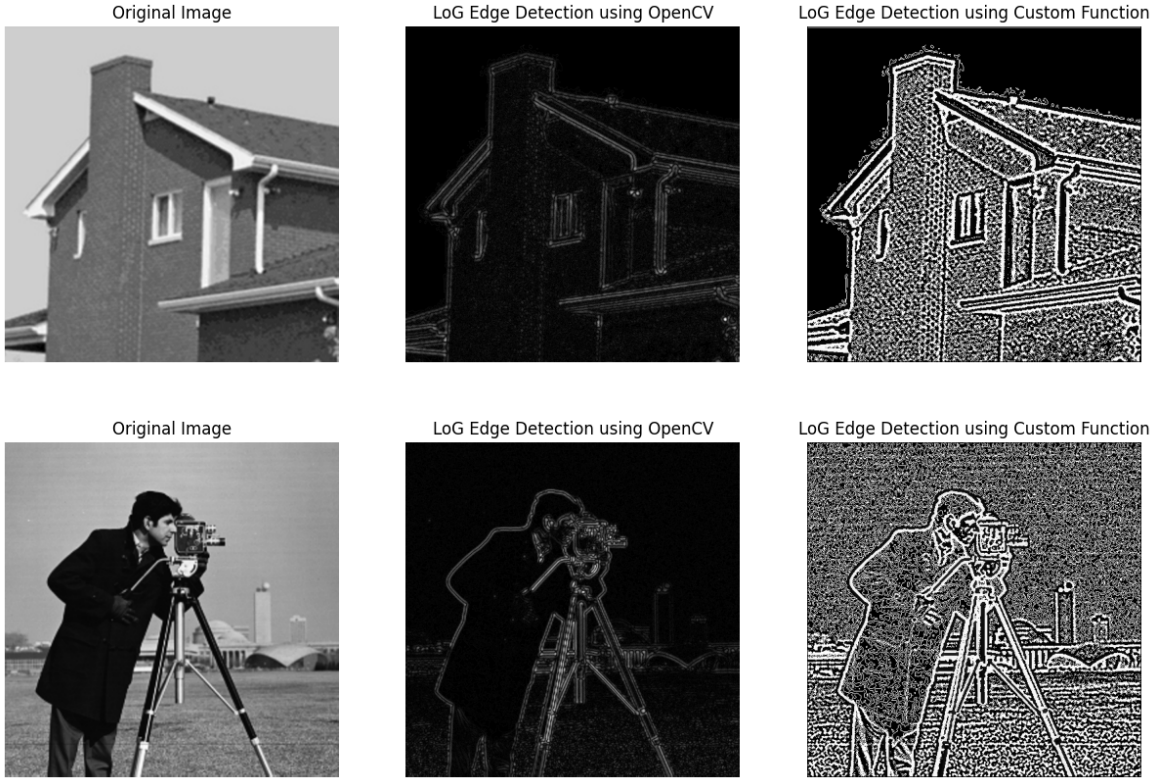

Fig. 11. Edge Detection using Laplacian of Gaussian

5.2. Laplacian of Gaussian (LoG)—The Laplacian of Gaussian (LoG) is used to detect edges by identifying regions in the image where the intensity changes rapidly. It combines Gaussian smoothing with the Laplacian operator to detect edges. It involves the following steps.

(1) Gaussian Smoothing:
    - The image is smoothed using a Gaussian filter to reduce noise.
(2) Laplacian Filtering:
    - Apply the Laplacian operator to the smoothed image to highlight regions of rapid intensity change.
(3) Zero-Crossing Detection:
    - Detect edges by finding zero crossings in the filtered image.
    - The overall LoG operation can be represented as:

$$LoG(x,y) = \frac{\partial^2}{\partial x^2} G(x,y) + \frac{\partial^2}{\partial y^2} G(x,y)$$

6. Downsampling of Images

Downsampling is a process used to reduce the resolution of an image, which can be useful for various applications such as image compression and computational efficiency.

6.1. Max Pooling—Max Pooling is a downsampling technique that reduces the size of the image by selecting the maximum value from a group of neighboring pixels. It is commonly used in CNNs to retain the most significant features and reduce the spatial dimensions of the image.

Process:

(1) Divide the image into non-overlapping rectangular regions (often 2×2 or 3×3).
(2) For each region, select the maximum pixel value.
(3) Form a new image with the selected maximum values.

Mathematical Representation: Given a pooling window size of 2×2, the max pooling operation can be expressed as:

$$\text{MaxPooling}(x,y) = \max\left(\begin{bmatrix} I(x,y) & I(x+1,y) \\ I(x,y+1) & I(x+1,y+1) \end{bmatrix}\right)$$

where $I(x,y)$ represents the pixel value at position (x,y) in the original image. The figure below shows the result of Max Pooling applied on the original image with different kernel sizes.

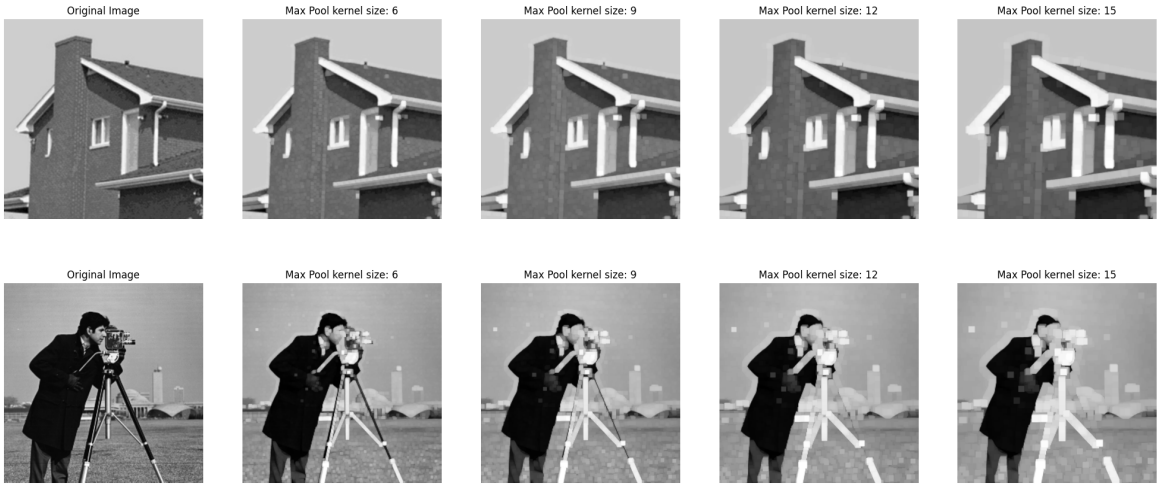

Fig. 12. Downsampling using Max Pooling

6.2. Average Pooling—Average Pooling is another downsampling technique that reduces the image size by computing the average value of pixels within each pooling window. It is used to smooth the image and capture average features over the region.

Process:

(1) Divide the image into non-overlapping rectangular regions (often 2x2 or 3x3).

(2) For each region, calculate the average pixel value.

(3) Form a new image with the computed average values.

Mathematical Representation: Given a pooling window size of 2×2, the average pooling operation can be expressed as:

$$\text{AveragePooling}(x,y) = \frac{1}{4}\sum_{i=0}^{1}\sum_{j=0}^{1} I(x+i,y+j)$$

where $I(x+i,y+j)$ represents the pixel values within the pooling window. The figure below shows result of average pool for different kernel sizes.

Fig. 13. Downsampling using Average Pooling

7. Discussion Questions

7.1. ***Advantages and Disadvantages of Median Filters Compared to Other LP Filters***—

7.1.1. Advantages of Median Filters—

- Noise Reduction: Median filters are highly effective at removing impulse noise (also known as salt-and-pepper noise) without blurring the image, unlike linear LP filters.
- Edge Preservation: Median filters preserve edges better than other LP filters like the averaging filter, which tends to blur edges while smoothing noise.
- Non-Linearity: Since median filters are non-linear, they are better at preserving the overall structure and sharpness of the image while filtering out extreme values.

7.1.2. Disadvantages of Median Filters—

- Computational Complexity: Median filtering requires sorting the pixel values in the neighborhood, which can be computationally intensive, especially for large kernel sizes.
- Not Suitable for All Noise Types: While effective against impulse noise, median filters may not perform as well as other LP filters in removing Gaussian noise, where a linear LP filter might be more appropriate.

- Edge Artifacts: For larger kernel sizes, median filters might introduce artifacts or distortions at the edges, as the filter does not take into account the local image structure.

7.2. ***Difference between Low Pass and High Pass Filter Kernels:*** —Low-pass (LP) and high-pass (HP) filters are used in image processing to manipulate the frequency components of an image. The primary differences between their kernels are:

Feature	**Low-Pass Filter (LPF)**	**High-Pass Filter (HPF)**
Purpose	Retains low-frequency components, suppresses high frequencies (blurring)	Retains high-frequency components, suppresses low frequencies (sharpening)
Kernel Characteristics	Positive weights that sum to 1, symmetrically distributed	Includes both positive and negative weights, sum close to zero
Effect	Smooths the image by averaging neighboring pixels	Enhances edges and fine details by emphasizing intensity differences
Common Example	$\frac{1}{9}\begin{pmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{pmatrix}$	$\begin{pmatrix}0 & -1 & 0\\ -1 & 4 & -1\\ 0 & -1 & 0\end{pmatrix}$

Table 2. Comparison of Low-Pass and High-Pass Filters

7.3. ***Proposed Kernel Filter for Detecting Diagonal Edges Only***—To detect diagonal edges, you can design a filter kernel that responds strongly to changes in pixel intensity along diagonal directions. Here's an example of a kernel that detects diagonal edges from top-left to bottom-right:

Feature	**Leading Diagonal**	**Trailing Diagonal**
Example	$\begin{pmatrix}-1 & 0 & 0\\ 0 & 2 & 0\\ 0 & 0 & -1\end{pmatrix}$	$\begin{pmatrix}0 & 0 & -1\\ 0 & 2 & 0\\ -1 & 0 & 0\end{pmatrix}$
Kernel Characteristics	Computes numerical gradient from top-left to bottom-right	Computes numerical gradient from top-right to bottom-left

Table 3. Diagonal Edge detection Kernels

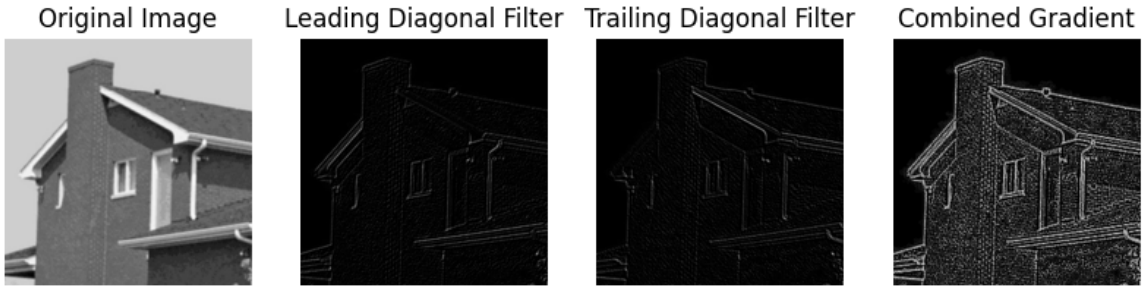

Fig. 14. Diagonal Edge Detection

7.4. ***Kernel Design for Corner Detection:***—Corner detection is a crucial technique in image processing, used to identify points in an image where the intensity changes sharply in multiple directions. These points, or "corners," are significant features that can be used in various applications, such as image matching, motion detection, and 3D reconstruction. *Corners* are points in an image where two or more edges meet. They are characterized by a large intensity variation when observed from multiple directions. The kernels used are as follows:

$$\mathbf{K_{trailing}} = \begin{pmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{K_{leading}} = \begin{pmatrix} -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \end{pmatrix}$$

Both the kernels are applied on the image in cascade and the ouput is as follows:

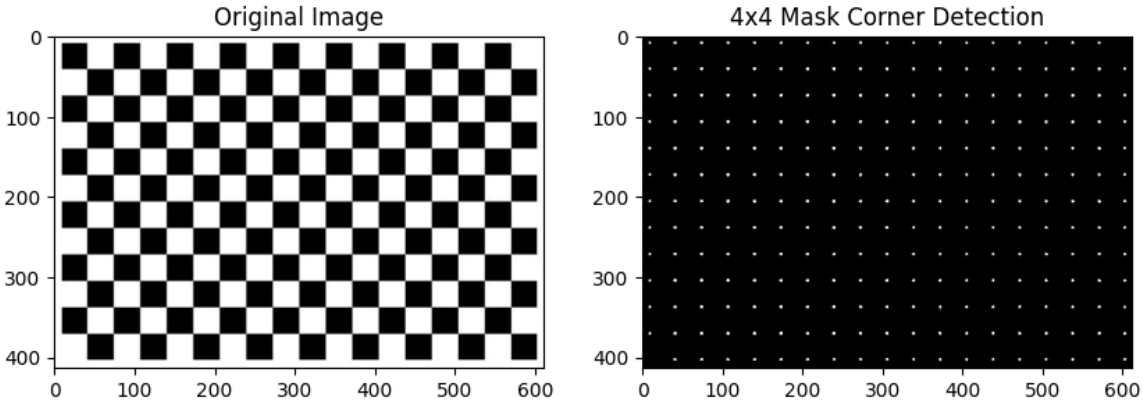

Fig. 15. Corner detection using 4×4 mask

7.5. ***Edge Orientations of Sobel Filter***—For edge detection using sobel filtering, we have:

$$\mathbf{G_x} = \mathbf{S_x} * \mathbf{I}, \quad \mathbf{G_y} = \mathbf{S_y} * \mathbf{I} \quad \text{and} \quad \Theta = \arctan(\frac{G_y}{G_x})$$

Here, $\mathbf{G_x}$ and $\mathbf{G_y}$ represent the gradients along x and y directions respectively. Θ represents the orientation matrix. In the given result, the orientation angle is represented in degrees.

Fig. 16. Orientation of Sobel Filtered Image

7.6. ***Spatial Domain vs. Frequency Domain Filters***—The application of Spatial Domain Filters are discussed in the earlier sections of the report where the filtering involves convolving the image with the filter kernel. This section discusses the application of filters in the frequency domain, with an example of High-Pass filter application in hand. The following table lists the major differences between spatial and frequency domain filters.

Aspect	**Spatial Domain Filters**	**Frequency Domain Filters**
Basic Concept	Operate directly on the pixel values of the image.	Operate on the Fourier transform of the image.
Procedure	Convolution of the image with a filter kernel.	Multiply the Fourier transform of the image by a filter.
Kernels Used	Typically small, localized kernels like 3x3, 5x5 matrices.	Large filters, often the same size as the image's frequency domain representation.
Types of Filters	Smoothing (e.g., Gaussian, Mean), Sharpening (e.g., Laplacian)	Low-pass (e.g., Gaussian in frequency) High-pass (e.g., Butterworth)
Edge Handling	Special techniques like padding or reflection required to handle edges.	Natural edge preservation by circular convolution in the frequency domain.
Applications	Used for tasks like noise reduction, edge detection, and simple smoothing.	Ideal for large-scale filtering tasks like image compression and global transformations.
Effect on Image	Changes are localized to specific areas, depending on the filter used.	Global changes affect the entire image uniformly in the spatial domain.

Table 4. Comparison between Spatial and Frequency Domain Filters

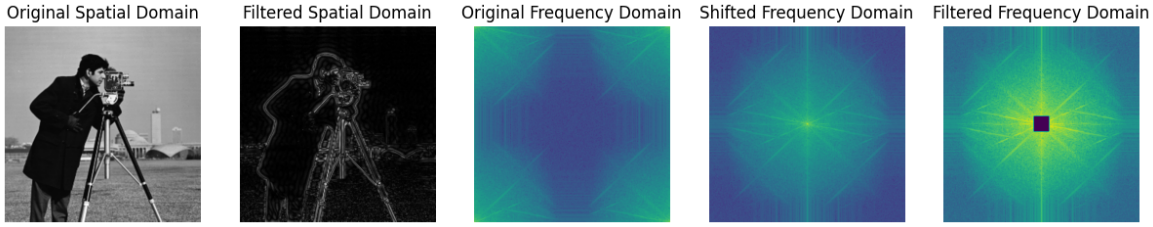

Fig. 17. High Pass Filtering in Frequency Domain

To process the image, we first apply a Fast Fourier Transform (FFT) to convert the image from the spatial domain to the frequency domain. This is expressed mathematically as:

$$\text{FFT}(f(x,y)) = F(u,v)$$

where $f(x,y)$ is the original image and $F(u,v)$ is its frequency representation.

Next, we shift the zero-frequency component to the center of the frequency spectrum using:

$$\text{FFTShift}(F(u,v)) = F'(u,v)$$

where $F'(u,v)$ represents the shifted frequency spectrum.

We then define a circular high-pass filter with a specified radius r. The mask $M(u,v)$ is defined as:

$$M(u,v) = \begin{cases} 0 & \text{if } \sqrt{(u-u_0)^2 + (v-v_0)^2} \leq r \\ 1 & \text{otherwise} \end{cases}$$

where (u_0, v_0) is the center of the frequency spectrum. Applying this mask to the shifted frequency spectrum removes low-frequency components:

$$F''(u,v) = F'(u,v) \times M(u,v)$$

We then shift the zero-frequency component back to its original position:

$$\text{IFFTShift}(F''(u,v)) = F_{\text{filtered}}(u,v)$$

where $F_{\text{filtered}}(u,v)$ is the filtered frequency spectrum. We perform an inverse FFT to convert the filtered image back to the spatial domain:

$$f_{\text{filtered}}(x,y) = \text{IFFT}(F_{\text{filtered}}(u,v))$$

Finally, we take the absolute value to get the magnitude of the complex result and convert it to an 8-bit unsigned integer format:

$$f_{\text{filtered}}(x,y) = \text{abs}(f_{\text{filtered}}(x,y))$$

This process removes low-frequency components and enhances high-frequency details in the image.

8. Conclusion

This report examined various image processing techniques, including filtering methods and edge detection. The Median filter excelled at removing Salt and Pepper noise and preserving edges, outperforming Mean and Gaussian filters in PSNR. High-pass filters like Sobel and Prewitt were effective for edge detection, while the Laplacian filter detected rapid intensity changes but was sensitive to noise. Downsampling techniques, Max Pooling and Average Pooling, were compared, with Max Pooling retaining key features. Corner detection and frequency domain filtering showed effective results for feature enhancement. Understanding these methods allows for optimized image processing tailored to specific tasks.