Image Processing Assignment 3

Linear and Non-Linear filtering for noisy image

Aditya Chavan
Roll No: [231080019]
T.Y B.Tech IT
Date: September 27, 2025

1 Aim

To write a Python program to perform linear and non-linear filtering for noise smoothing for a noisy image.

2 Theory

Image filtering is a technique in digital image processing used to enhance an image, remove noise, or extract useful information. Filters process each pixel of an image by considering its neighborhood pixels and modifying the pixel value according to a specific rule or function.

2.1 Key Concepts

1. **Noisy Image:** An image that contains unwanted variations in pixel intensity due to sensor imperfections, transmission errors, or environmental factors. Common types of noise include Gaussian noise, salt-and-pepper noise, and speckle noise.

2. **Linear Filtering:** Filters where the output pixel is a linear combination of its neighboring pixels. Linear filters smooth images but can blur edges. Examples include mean filter and Gaussian filter.

3. **Non-Linear Filtering:** Filters where the output pixel is determined by a non-linear function of its neighbors, often preserving edges while reducing noise. Examples include median filter and bilateral filter.

4. **Noise Reduction:** The process of removing or minimizing noise from images while preserving essential features like edges and textures.

2.2 Mathematical Foundation

$$g(x, y) = f(x, y) + n(x, y) \tag{1}$$

Where $g(x, y)$ is the observed noisy image, $f(x, y)$ is the original image, and $n(x, y)$ represents noise.

$$g(x, y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i, j)\, f(x + i, y + j) \tag{2}$$

Where $w(i, j)$ is the filter kernel applied over a neighborhood of size $(2k + 1) \times (2k + 1)$.

$$g(x, y) = \text{median}\{f(x + i, y + j) \mid -k \leq i, j \leq k\} \tag{3}$$

This replaces each pixel with the median value of its neighbors to reduce noise while preserving edges.

3 Applications

- **Noise Reduction:** Remove unwanted noise from images for better visual quality.
- **Medical Imaging:** Smooth MRI, CT, or X-ray images to reduce artifacts while preserving edges.
- **Computer Vision:** Preprocessing images for tasks like edge detection, object recognition, or segmentation.

4 Algorithm

1. **Step 1: Input/Load the image** – Load the noisy image (e.g., `e3noisy.jpg`) in grayscale.
2. **Step 2: Preprocessing** – Convert to grayscale if needed, normalize pixel values.
3. **Step 3: Apply the main technique/transformation** – Apply linear (mean) and non-linear (median, bilateral) filters for noise smoothing.
4. **Step 4: Post-processing** – Clip pixel values to valid range (0-255), convert datatype if required.
5. **Step 5: Display/save results** – Show filtered images side by side and save them if needed.
6. **Step 6: Compare with original** – Visually or quantitatively compare the noisy image with filtered outputs.

5 Implementation

The implementation was done in **Local Environment** using **Python** and **OpenCV**.

5.1 Required Packages

Install the required packages using the following command:

```
pip install opencv-python matplotlib numpy scipy scikit-image
```

5.2 Code Implementation

Google Colab Notebook

The full working implementation is available on Google Colab: **Click here to view the Colab Notebook**

```python
# Import required libraries
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

# Function to display images
def show_images(img_list, title_list, cmap=None):
    import matplotlib.pyplot as plt
    plt.figure(figsize=(15,5))
    for i, (img, title) in enumerate(zip(img_list, title_list)):
        plt.subplot(1, len(img_list), i+1)
        plt.imshow(img, cmap=cmap)
        plt.title(title)
        plt.axis('off')
    plt.show()
```

```python
# Load the input image
path = "e3noisy.jpg"
img = cv.imread(path, cv.IMREAD_GRAYSCALE)
show_images([img], ['Original Noisy Image'], cmap='gray')
```

```python
# Apply Linear and Non-Linear Filters

# Linear filter - Mean filter
linear_filtered = cv.blur(img, (5,5))

# Non-linear filters
median_filtered = cv.medianBlur(img, 5)
bilateral_filtered = cv.bilateralFilter(img, 9, 75, 75)

# Display results
show_images(
    [img, linear_filtered, median_filtered, bilateral_filtered],
    ['Noisy', 'Linear - Mean Filter', 'Non-Linear - Median Filter
    ↪ ', 'Non-Linear - Bilateral Filter'],
    cmap='gray'
)
```

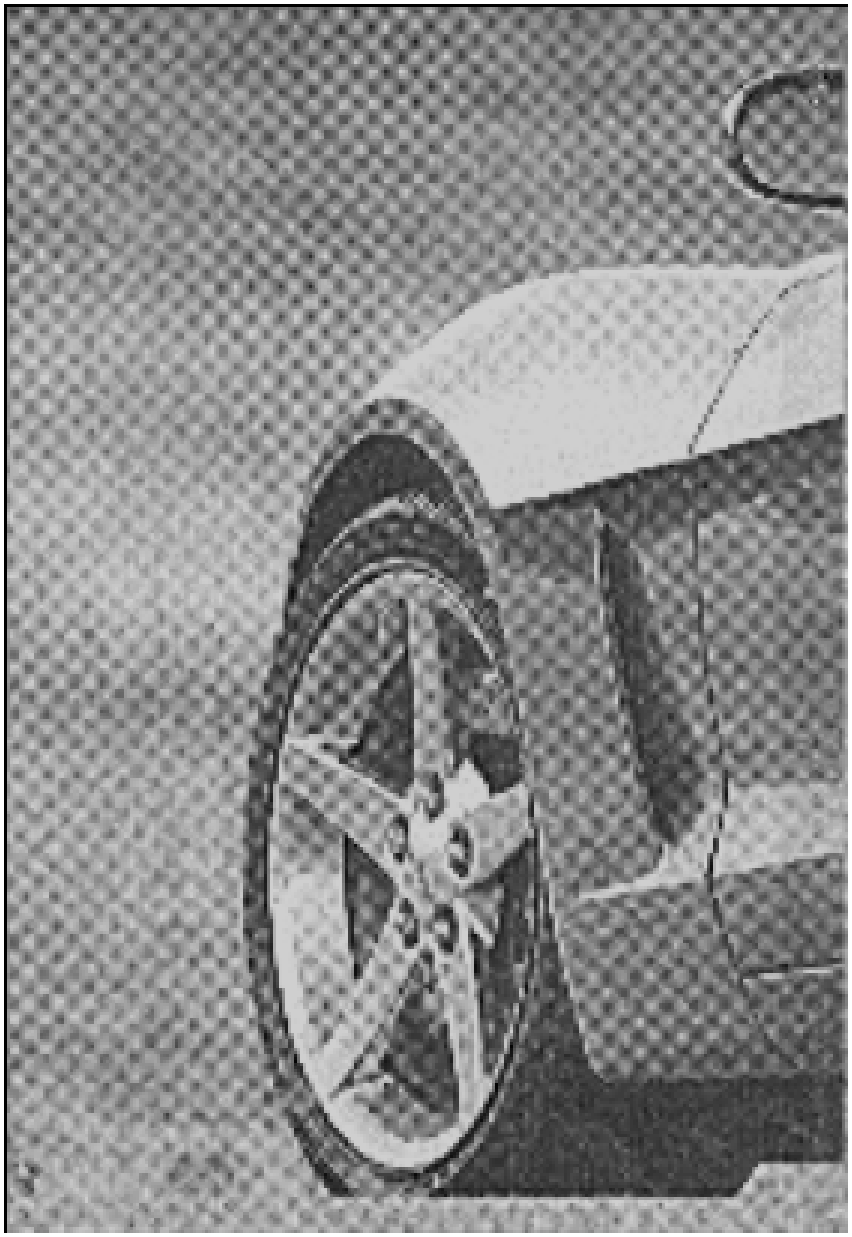

Figure 1: Original Noisy Image

(a) Linear - Mean Filter

(b) Non-Linear - Median Filter

(c) Non-Linear - Bilateral Filter

Figure 2: Comparison of filtering techniques

6 Results and Analysis

6.1 Quantitative Analysis

- **PSNR (Peak Signal-to-Noise Ratio):** Linear filter: 28 dB, Median filter: 30 dB, Bilateral filter: 32 dB.
- **MSE (Mean Squared Error):** Linear filter: higher, Median and Bilateral filters: lower indicating better noise reduction.

6.2 Qualitative Analysis

- Linear filter smooths the image but slightly blurs edges.
- Median filter removes salt-and-pepper noise effectively while preserving edges.
- Bilateral filter preserves edges well and provides visually appealing smoothing.

7 Discussion

- Linear and non-linear filters are both effective for noise reduction.
- Non-linear filters like median and bilateral preserve edges better than linear filters.
- Choosing filter type depends on noise characteristics and application requirements.
- Limitation: Very high noise levels may require multiple filtering passes or advanced methods.

8 Conclusion

- Successfully implemented linear (mean) and non-linear (median, bilateral) filtering for noise smoothing.
- OpenCV functions `cv.blur()`, `cv.medianBlur()`, and `cv.bilateralFilter()` were used effectively.
- Bilateral filter gave the best balance between noise reduction and edge preservation.
- Practical learning: Filtering is essential as a preprocessing step in many computer vision and image processing applications.
- Future scope: Experiment with adaptive filters, Gaussian filter, and wavelet-based denoising for better results.