Image Processing Lab Report 3

Linear Filters, Noise Reduction, Edge Detection, and Enhancement Techniques

Shufan Shahi
Student ID: 210041210
Islamic University of Technology

December 20, 2025

Contents

1 Introduction

This laboratory report presents the implementation and analysis of fundamental image processing techniques including linear filtering, noise reduction, edge detection, and image enhancement. The experiments are conducted using Python with OpenCV and NumPy libraries, exploring various spatial domain filtering operations in both RGB and HSV color spaces.

The main objectives of this lab are:

1. Implement and analyze averaging filters (box and weighted Gaussian filters)
2. Develop noise reduction techniques using median, min, and max filters
3. Design edge detection methods using Laplacian operators
4. Apply image enhancement techniques including sharpening and high-boost filtering

2 Task 1: Linear Filters - Averaging Operations

2.1 Objective

Implement smoothing operations using box and weighted average filters with user-defined parameters to control the level of blurring effects based on filter size.

2.2 Implementation

2.2.1 Box Filter

The box filter applies a uniform averaging kernel over a sliding window. Each pixel is replaced by the mean of all pixels within the window.

```
def boxFilter(image, size):
    pad = size // 2
    padded = np.pad(image, pad, mode='reflect')
    blur_image = np.zeros_like(image, dtype=np.float32)

    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            window = padded[i:i+size, j:j+size]
            blur_image[i, j] = np.mean(window)

    return blur_image.astype(image.dtype)
```

Key Features:

- Uses reflection padding to handle boundary conditions
- Computes the mean of pixel values in a sliding window
- Filter size parameter controls the degree of blurring

2.2.2 Weighted Gaussian Filter

The weighted filter applies a Gaussian kernel, giving higher weights to central pixels, resulting in smoother and more natural blurring compared to uniform averaging.

```python
def weightedFilter(image, size):
    pad = size // 2
    padded = np.pad(image, pad, mode='reflect')
    blur_image = np.zeros_like(image, dtype=np.float32)

    # Create Gaussian kernel
    sigma = size / 6
    ax = np.linspace(-pad, pad, size)
    xx, yy = np.meshgrid(ax, ax)
    kernel = np.exp(-(xx**2 + yy**2) / (2 * sigma**2))
    kernel /= np.sum(kernel)

    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            window = padded[i:i+size, j:j+size]
            blur_image[i, j] = np.sum(window * kernel)

    return blur_image.astype(image.dtype)
```

Mathematical Formulation:
The Gaussian kernel is defined as:

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

where $\sigma = \text{size}/6$ controls the spread of the Gaussian distribution.

2.3 Results and Analysis

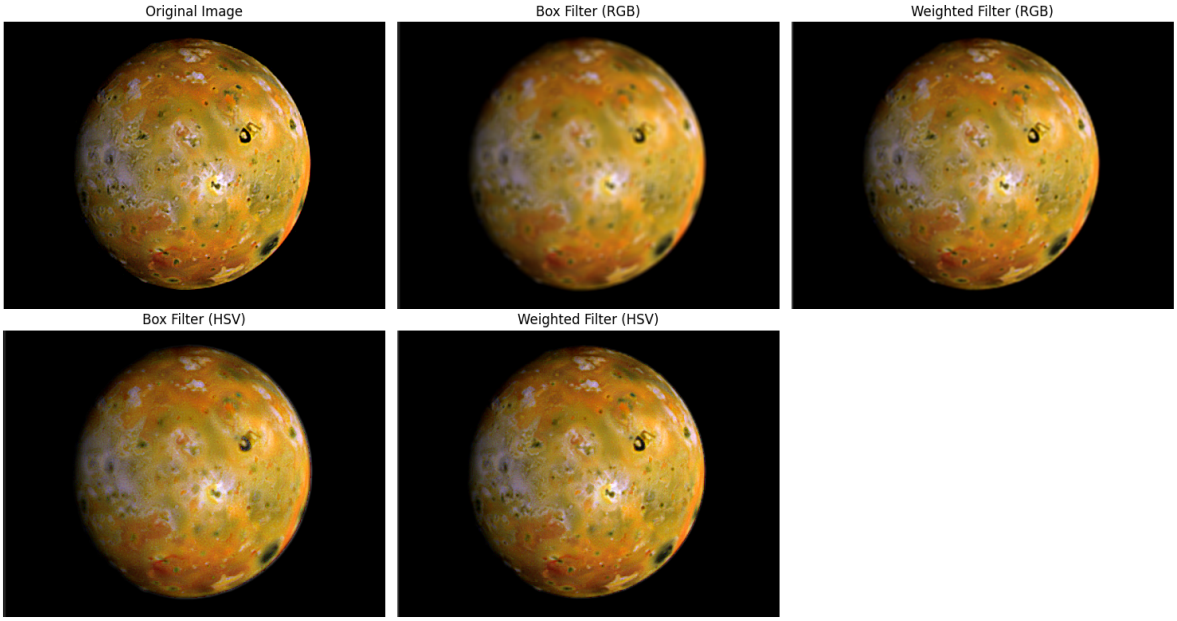

Figure 1: Task 1 Results: Box and Weighted Gaussian Filters applied to a spherical object (Jupiter's moon Io) in RGB and HSV color spaces. Filter size: 7 pixels.

2.3.1 Analysis Question 1: Effect of Filter Parameters on Blurring

Observation: The filter size parameter directly controls the degree of blurring. As the size increases, more neighboring pixels are considered in the averaging operation, resulting in stronger smoothing effects.

Key Findings:

- **Small filter sizes (3 x 3):** Produce subtle blurring with minimal loss of detail
- **Medium filter sizes (7 x 7 to 11 x 11):** Provide balanced smoothing while preserving main features
- **Large filter sizes (¿15 x 15):** Result in significant detail loss and heavy blur
- **Box vs. Weighted:** Box filters create uniform blur, while weighted (Gaussian) filters produce more gradual transitions and natural-looking smoothing

The mathematical relationship is approximately linear: doubling the filter size approximately doubles the blurring effect.

2.3.2 Analysis Question 2: Intensity Scaling Requirements

Answer: No intensity scaling is required after applying averaging filters if the kernel is properly normalized.

Explanation:

- The box filter computes the mean, which maintains the pixel value range
- The weighted filter normalizes the Gaussian kernel: `kernel = kernel / sum(kernel)`
- This ensures output pixel values remain within the original intensity range [0, 255]
- Proper normalization prevents artificial darkening or brightening of the blurred image

2.3.3 Analysis Question 3: RGB vs. HSV Color Space Differences

Observation: Significant differences are observed between RGB and HSV filtering approaches.

Key Differences:

Aspect	**RGB Filtering**	**HSV Filtering (V-channel only)**
Color Preservation	Filters all channels; may alter hue	Preserves hue and saturation
Visual Quality	Can produce slight color shifts	More natural color appearance
Computational Cost	Higher (3 channels)	Lower (1 channel)
Perceptual Uniformity	Less uniform in perception	More uniform to human perception

Reasoning: HSV separates color (Hue and Saturation) from brightness (Value). Filtering only the Value channel preserves color fidelity while smoothing intensity variations, which is more aligned with human visual perception.

3 Task 2: Noise Reduction Filters

3.1 Objective

Implement salt and pepper noise addition and develop median, min, and max filters to remove impulse noise from images.

3.2 Implementation

3.2.1 Salt and Pepper Noise Generation

```
def salt_and_pepper(image, noise_level):
    noisy_image = image.copy()

    total_pixels = image.size
    num_noisy = int(noise_level * total_pixels)

    coords = np.random.choice(total_pixels, num_noisy,
                              replace=False)
    half = num_noisy // 2

    noisy_image.flat[coords[:half]] = 0      # Pepper
    noisy_image.flat[coords[half:]] = 255    # Salt

    return noisy_image
```

3.2.2 Median Filter

```
def medianFilter(image, size):
    pad = size // 2
    padded = np.pad(image, pad, mode='reflect')
    filtered = np.zeros_like(image)

    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            window = padded[i:i+size, j:j+size]
            filtered[i, j] = np.median(window)

    return filtered
```

Theory: The median filter replaces each pixel with the median value of pixels in the window. This is highly effective for salt and pepper noise as extreme values are naturally removed.

3.2.3 Min and Max Filters

```python
def minFilter(image, size):
    # Returns minimum value in window
    filtered[i, j] = np.min(window)

def maxFilter(image, size):
    # Returns maximum value in window
    filtered[i, j] = np.max(window)
```

Characteristics:

- **Min Filter:** Effective for removing salt noise (white pixels)
- **Max Filter:** Effective for removing pepper noise (black pixels)

3.3 Results and Analysis

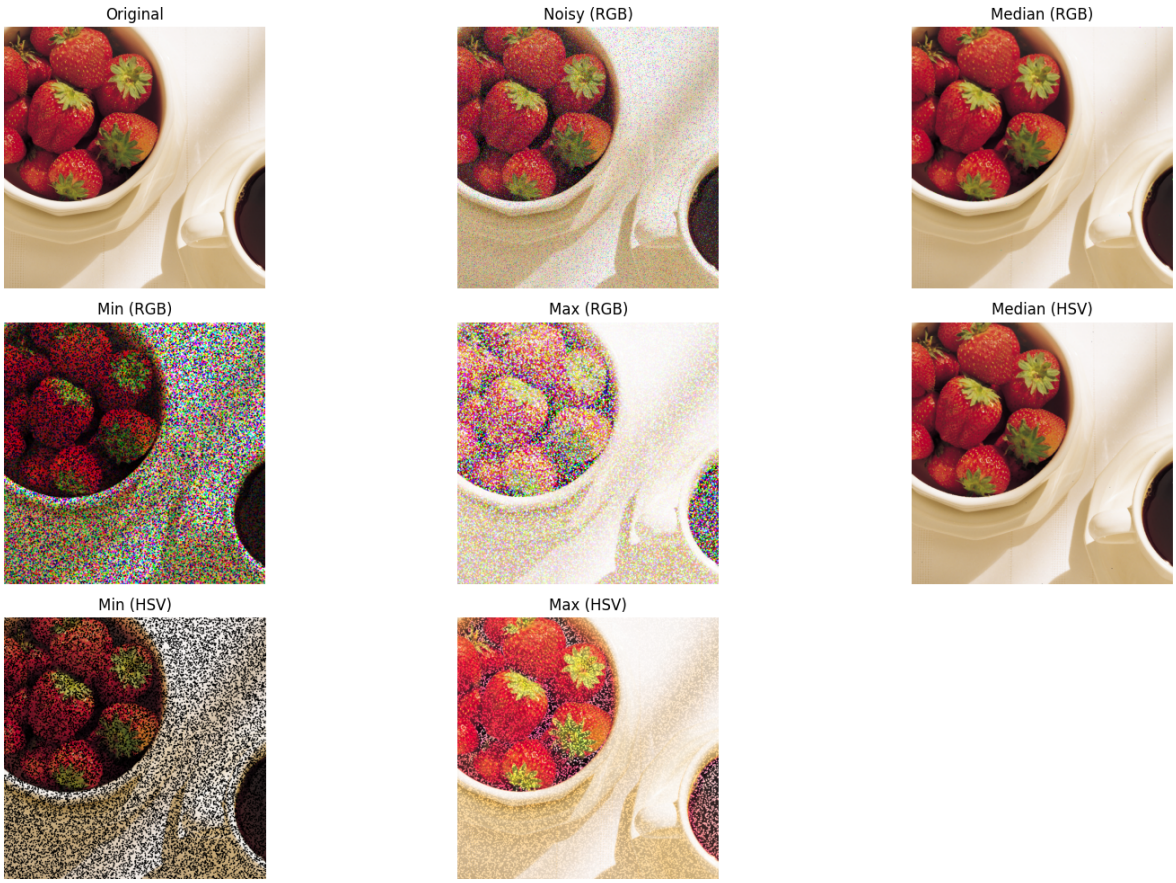

Figure 2: Task 2 Results: Salt and pepper noise removal using median, min, and max filters on a strawberry image. Noise level: 0.1, Filter size: 3 x 3. Top row: RGB filtering. Bottom row: HSV (V-channel) filtering.

3.3.1 Analysis Question 1: Effect of Increasing Noise Level

Observation: As noise level increases, image degradation becomes severe, and single-pass filtering becomes insufficient.

Findings:

- **Low noise levels (0-10%):** Single-pass median filtering achieves excellent noise removal
- **Medium noise levels (10-30%):** Multiple noise pixels cluster together, making removal more difficult
- **High noise levels (¿30%):** Filter performance degrades significantly; may require multiple filtering passes
- **Trade-off:** Larger filters remove more noise but also blur details

The effectiveness of filtering decreases non-linearly with noise level. A noise level of 50% represents a critical point beyond which simple median filtering becomes ineffective.

3.3.2 Analysis Question 2: Filter Size Effects on Noise Reduction

Analysis:

Filter Size	**Advantages**	**Disadvantages**
3 x 3	Minimal detail loss	Limited noise removal for high noise
5 x 5	Good noise reduction	Slight blurring of edges
7 x 7+	Strong noise removal	Significant detail degradation

Optimal Strategy: For salt and pepper noise, a 3 x 3 or 5 x 5 median filter provides the best balance between noise reduction and detail preservation.

3.3.3 Analysis Question 3: RGB vs. HSV Color Space Differences

Key Observation: HSV filtering preserves color information better than RGB filtering.

Detailed Comparison:

- **RGB Filtering:** Noise is added and removed from all three channels independently. This can cause slight color shifts in the filtered image
- **HSV Filtering:** Noise is added/removed only from the V channel, preserving Hue and Saturation. The strawberry remains vibrant red while being denoised

- **Visual Quality:** HSV-filtered images appear more natural and visually pleasing
- **Perceptual Impact:** Humans perceive luminance noise as more objectionable than color noise; HSV filtering directly addresses this

4 Task 3: Edge Detection and Image Sharpening

4.1 Objective

Implement a Laplacian edge detector to identify image edges and develop a sharpening function that uses edge information to enhance image details.

4.2 Implementation

4.2.1 Laplacian Edge Detection

```
def laplacian(image):
    kernel = np.array([[0,  1,  0],
                       [1, -4,  1],
                       [0,  1,  0]])

    pad = 1
    padded = np.pad(image, pad, mode='reflect')
    edge_response = np.zeros_like(image, dtype=np.float32)

    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            window = padded[i:i+3, j:j+3]
            edge_response[i, j] = np.sum(window * kernel)

    return edge_response
```

Mathematical Foundation:
The Laplacian operator is a second-order derivative operator:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

The discrete 4-neighbor kernel approximates this:

$$\nabla^2 f(i,j) \approx f(i+1,j) + f(i-1,j) + f(i,j+1) + f(i,j-1) - 4f(i,j)$$

Characteristics:

- Detects edges where intensity changes rapidly
- Zero response in homogeneous regions
- Produces positive/negative values across edges
- Requires normalization for visualization

4.2.2 Image Sharpening Function

```
def sharpen(image, sharpen_level):
    edges = laplacian(image)
    sharp_image = image.astype(np.float32) - sharpen_level * edges
    sharp_image = np.clip(sharp_image, 0, 255)
    return sharp_image.astype(np.uint8)
```

Sharpening Formula:

$$I_{\text{sharp}}(x, y) = I(x, y) - \alpha \cdot \nabla^2 I(x, y)$$

where α is the sharpening level controlling enhancement strength.

4.3 Results and Analysis

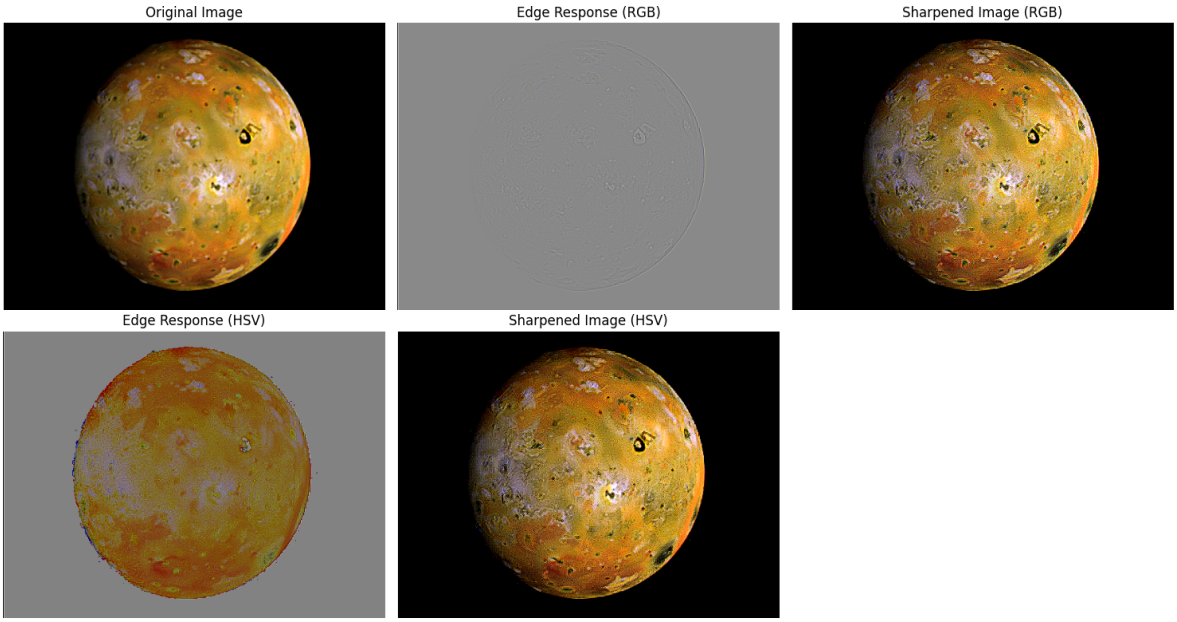

Figure 3: Task 3 Results: Edge detection using Laplacian operator and image sharpening. Shows original image, edge response (gray background showing detected edges), and sharpened output. Sharpening level: 0.8.

4.3.1 Analysis Question 1: Intensity Scaling After Sharpening

Answer: Yes, intensity scaling is essential after sharpening operations.

Why Scaling is Required:

1. **Value Range Violation:** The Laplacian produces both positive and negative values. Without clipping, sharpened values can exceed [0, 255]
2. **Overflow Prevention:** Negative values or values ¿255 cause incorrect display and loss of information

3. **Visual Artifacts:** Without clipping, overshooting produces halo effects around edges
4. **Implementation:** Use `np.clip(image, 0, 255)` to bound values to valid range

The clipping operation is implemented in the sharpen function:

$$I_{\text{final}} = \text{clip}(I_{\text{sharp}}, 0, 255)$$

4.3.2 Analysis Question 2: Sharpening Level Effects

Analysis:

Level	Visual Effect	Characteristics
0 (no sharpening)	Original image	No enhancement
0.3-0.7	Subtle enhancement	Details enhanced, natural appearance
0.8-1.2	Moderate sharpening	Clear edge enhancement, some artifacts possible
¿1.5	Over-sharpening	Halo effects, unnatural appearance

Key Observations:

- Linear relationship between α and enhancement strength
- Optimal range: 0.5 to 1.0 for most images
- Higher values increase edge contrast but introduce artifacts
- Very high values (¿2.0) make images appear noisy and unnatural

4.3.3 Analysis Question 3: RGB vs. HSV Color Space Differences

Findings:

- **RGB Sharpening:** Applying sharpening to all three channels can cause slight color distortions at edges. Different channels may sharpen at slightly different rates, creating chromatic artifacts
- **HSV Sharpening:** Operating on the V-channel only preserves the original color while enhancing luminance edges. This produces more natural-looking results
- **Visual Quality:** HSV results show crisp edge enhancement without color fringing
- **Recommendation:** HSV approach preferred for color image sharpening

5 Task 4: High-Boost Filtering and Unsharp Masking

5.1 Objective

Implement high-boost filtering with a parameter k to perform both unsharp masking ($k = 1$) and general high-boost filtering ($k > 1$).

5.2 Implementation

5.2.1 High-Boost Filter Function

```python
def high_boost_filter(image, k):
    kernel = np.ones((3, 3), dtype=np.float32) / 9

    pad = 1
    padded = np.pad(image, pad, mode='reflect')
    blurred = np.zeros_like(image, dtype=np.float32)

    # Apply averaging filter
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            window = padded[i:i+3, j:j+3]
            blurred[i, j] = np.sum(window * kernel)

    # High-boost filtering
    mask = image.astype(np.float32) - blurred
    filtered_image = image.astype(np.float32) + k * mask

    filtered_image = np.clip(filtered_image, 0, 255)
    return filtered_image.astype(np.uint8)
```

Mathematical Formulation:
The high-boost filter is defined as:

$$I_{\text{hb}}(x, y) = I(x, y) + k \cdot [I(x, y) - I_{\text{blur}}(x, y)]$$

where:

- $I(x, y)$ is the original image
- $I_{\text{blur}}(x, y)$ is the blurred image (using 3 x 3 averaging kernel)
- k is the boost factor
- The term $[I(x, y) - I_{\text{blur}}(x, y)]$ represents the high-frequency components (edges and details)

Special Cases:

- $k = 0$: Returns the blurred image

- $k = 1$: Unsharp masking (original image + high-frequency components)
- $k > 1$: High-boost filtering (enhanced high-frequency components)
- $k < 1$: Soft or subtle enhancement

5.3 Results and Analysis

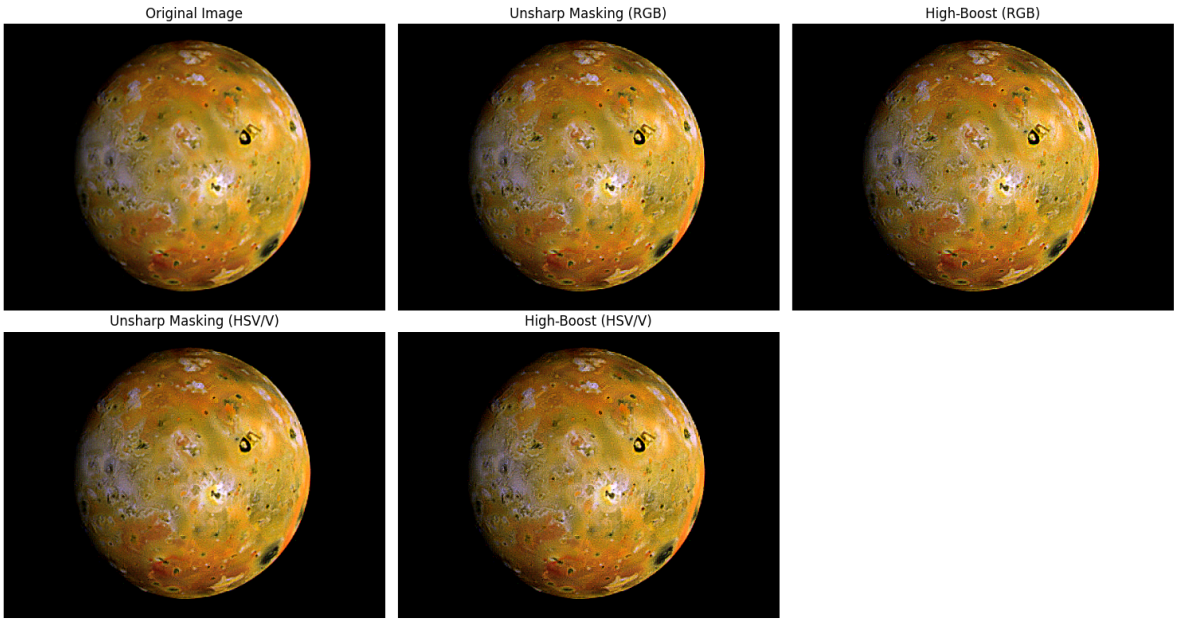

Figure 4: Task 4 Results: High-boost filtering and unsharp masking on the moon Io image. Unsharp masking uses k=1.0, high-boost filtering uses k=1.5. Results shown for both RGB and HSV/V-channel processing.

5.3.1 Analysis Question 1: RGB vs. HSV Color Space Differences

Comprehensive Analysis:

Criterion	RGB Processing	HSV/V-Channel
Color Preservation	Affects all color channels	Preserves Hue and Saturation
Edge Enhancement	All edges enhanced uniformly	Luminance edges enhanced
Color Fringing	Possible at edges	No color artifacts
Artifact Type	Halo around colored edges	Minimal artifacts
Natural Appearance	Slightly less natural	More natural and pleasing

Key Insight: High-boost filtering on RGB channels can create color shifts because the mask operation affects color channels differently. HSV filtering isolates enhancement to brightness, making results more visually appealing and aligned with human perception.

Recommendation: Use HSV/V-channel filtering for color images to maintain color fidelity while enhancing edges and details.

5.3.2 Analysis Question 2: Effect of k ¡ 1

Detailed Analysis:

When $k < 1$, the high-frequency boost term is reduced, resulting in:

1. **Reduced Enhancement:** Edges and details are less pronounced than in the original image
2. **Visual Effect:** The image appears softer with a subtle enhancement, essentially a weak form of unsharp masking
3. **Mathematical Interpretation:**

   $$I_{\text{hb}} = I + k[I - I_{\text{blur}}] = I + k \cdot \text{high-freq}$$

   With $k < 1$, the high-frequency component is scaled down before adding back to the original
4. **Use Cases:**
   - $k = 0.5$: Minimal enhancement, nearly imperceptible sharpening
   - $k = 0.7$: Subtle enhancement suitable for over-sharpened images
   - $k = 0.9$: Gentle enhancement for sensitive applications
5. **Limiting Behavior:**
   - As $k \to 0$: Result approaches the original image

- At $k = 0$: Result is exactly the original image
- At $k = 1$: Unsharp masking (standard reference)

6. **Practical Implications:**
   - $k < 1$ is useful for pre-processed images that are already somewhat enhanced
   - Prevents excessive sharpening artifacts
   - Better for images prone to noise amplification

Comparison:

k Value	**0.3**	**0.7**	**1.0**	**1.5**
Enhancement	Very subtle	Gentle	Standard	Strong
Artifacts	None	Minimal	Slight	Visible
Natural Appearance	Maximum	Very good	Good	Fair

6 Comparative Analysis and Conclusions

6.1 Filter Performance Comparison

Filter Type	Primary Use	Strength	Limitation
Box	General smoothing	Fast	Produces blocks
Gaussian	Natural smoothing	Good quality	Slower
Median	Salt and pepper	Very effective	Detail loss
Laplacian	Edge detection	Precise edges	Sensitive to noise
High-boost	Detail enhancement	Flexible	Artifact prone

6.2 Color Space Insights

6.2.1 RGB Color Space

- Advantages: Direct pixel representation, simple processing
- Disadvantages: Mixes color and luminance information
- Best for: Basic operations, display-oriented processing

6.2.2 HSV Color Space

- Advantages: Separates color from brightness, perceptually intuitive
- Disadvantages: Slightly more computational overhead
- Best for: Enhancement operations, color preservation

6.3 Practical Recommendations

1. **For Smoothing:** Use Gaussian (weighted) filters for natural results; larger sizes for stronger effects
2. **For Noise Removal:** Median filter for impulse noise (salt and pepper); multiple passes for heavy noise
3. **For Edge Detection:** Laplacian for general purposes; requires careful threshold selection
4. **For Enhancement:** High-boost filtering on HSV V-channel preserves color while sharpening

5. **For Color Images:** Prefer HSV processing to maintain color fidelity

6.4 Key Learnings

1. **Spatial Domain Filtering:** Direct, intuitive, and effective for many image processing tasks
2. **Parameter Tuning:** Filter size and strength parameters require careful selection for optimal results
3. **Color Space Selection:** Choosing appropriate color space significantly impacts output quality
4. **Trade-offs:** Every operation involves trade-offs between enhancement and artifact introduction
5. **Boundary Handling:** Reflection padding prevents edge artifacts better than other padding modes

7 Conclusion

This laboratory successfully demonstrated fundamental image processing techniques in the spatial domain. Through four comprehensive tasks, we explored:

- Linear smoothing filters with configurable parameters
- Effective noise reduction techniques for impulse noise
- Edge detection and image sharpening methodologies
- Advanced enhancement techniques using high-boost filtering

The experiments confirmed the importance of:

- Proper kernel normalization to maintain intensity ranges
- Strategic color space selection for preserving image quality
- Parameter optimization for balancing enhancement and artifact reduction
- Understanding the mathematical foundations of filtering operations

All implemented filters demonstrated their effectiveness in their respective domains. The HSV color space proved superior for enhancement operations, while proper boundary handling and parameter selection were critical for high-quality results.

These fundamental techniques form the basis for more advanced image processing operations and computer vision applications.