Machine Vision

Linear and Nonlinear Spatial Filtering in MATLAB

Lab Activity Sheet 3

Author: Dr. Mostapha Kalami Heris

1 Introduction

You will implement spatial filtering using both correlation and convolution, examine how boundary padding choices affect results near borders, and apply nonlinear median filtering to combat impulse noise while preserving edges. Alongside the guided steps, you will find open-ended prompts inviting you to explore alternatives, test parameters, and justify your design decisions with evidence. Curiosity and experimentation are expected and rewarded.

2 Learning Outcomes

By completing this lab, you will be able to:

1. Distinguish correlation from convolution and identify when they yield identical outputs.
2. Apply and analyze linear filters (average, Gaussian, Laplacian and Laplacian-of-Gaussian).
3. Evaluate the influence of boundary padding (`zeros`, `replicate`, `symmetric`, `circular`) on visual appearance and measurements.
4. Implement nonlinear median filtering for robust noise removal with edge preservation.
5. Extend spatial filtering to RGB images and to gradient-based features.
6. Design and justify your own filtering strategies using qualitative and quantitative evidence.

3 Activity 1 — Correlation vs. Convolution

3.1 Objective

Understand the mathematical and operational difference between correlation and convolution in spatial filtering.

3.2 Description

In image processing, **correlation** measures the similarity between a kernel and regions of an image, whereas **convolution** flips the kernel before applying it. Although these two operations look similar, they produce different outputs unless the filter kernel is symmetric. MATLAB's `imfilter` performs correlation by default, but you can explicitly request convolution.

3.3 Steps

1. Read and display a grayscale image.

   ```
   I = imread('coins.png');
   I = im2double(I);
   imshow(I, []);
   title('Original Image');
   ```

2. Define a simple horizontal edge-detection kernel.

   ```
   h = [-1 0 1];
   ```

3. Apply correlation using `imfilter` (default behavior).

   ```
   Icorr = imfilter(I, h);
   imshow(Icorr, []);
   title('Result of Correlation');
   ```

4. Apply convolution using either `imfilter(...,'conv')` or `conv2`.

```
Iconv1 = imfilter(I, h, 'conv');
Iconv2 = conv2(I, h, 'same');
imshowpair(Iconv1, Iconv2, 'montage');
title('Convolution using imfilter and conv2');
```

Function Hint

`imfilter`: Performs correlation or convolution between an image and a kernel, with optional padding and output size control.
`conv2`: Computes explicit two-dimensional convolution; slower but educationally clear.
`fspecial`: Generates standard filter kernels such as averaging, Gaussian, or Laplacian for quick setup.

3.4 Discussion

Observe that correlation and convolution produce mirror-image results when the kernel is asymmetric. For symmetric kernels (e.g., Gaussian or mean filters), correlation and convolution yield identical outputs.

Exploration & Inquiry

Exploration & Inquiry

- Design and test an asymmetric kernel (for example, an emboss or directional-edge filter) and visualize its impact on the image.
- Identify a kernel where correlation and convolution produce identical results. Explain mathematically why this happens.
- Quantify differences by calculating mean absolute pixel deviation between correlation and convolution outputs.
- Experiment with multiple filter sizes and orientations; discuss how kernel shape affects sensitivity to direction.

4 Activity 2 — Linear Filtering for Smoothing and Enhancement

4.1 Objective

Apply and compare common linear filters (average, Gaussian, Laplacian/LoG) for noise reduction and detail control.

4.2 Description

You will construct standard linear filters, apply them to a grayscale image, and compare their effects visually and with simple diagnostics (for example, histograms or variance). Average and Gaussian filters suppress noise but blur edges to different extents. Laplacian or Laplacian-of-Gaussian (LoG) enhances edges and fine detail, which can also amplify noise.

4.3 Steps

1. Read and display a test image.

```
I = imread('coins.png');
I = im2double(I);
imshow(I, []); title('Original');
```

2. Create and apply a 5x5 averaging filter using `fspecial` and `imfilter`.

```
h_avg = fspecial('average', [5 5]);   % 5x5 normalized averaging kernel
I_avg = imfilter(I, h_avg);           % correlation by default
imshow(I_avg, []); title('Average (5x5)');
```

Function Hint

`fspecial('average', [m n])`: Returns a normalized mean filter of the size $m \times n$.
`imfilter(I, h)`: Applies correlation between image `I` and kernel `h`. Use optional arguments to control padding and to switch to convolution.

3. Create and apply a Gaussian filter via two routes:

```
% Route A: explicit kernel then imfilter
h_gauss = fspecial('gaussian', [5 5], 1);  % size = 5x5, sigma = 1
I_gA = imfilter(I, h_gauss);

% Route B: direct Gaussian smoothing
I_gB = imgaussfilt(I, 1);                  % sigma = 1
imshowpair(I_gA, I_gB, 'montage');
title('Gaussian: fspecial+imfilter (left) vs imgaussfilt (right)');
```

Function Hint

`fspecial('gaussian', [m n], sigma)`: Returns a Gaussian correlation kernel with standard deviation σ.
`imgaussfilt(I, sigma)`: Performs Gaussian smoothing directly on `I`. Accepts a scalar σ (isotropic) or a 2-element vector $[\sigma_x, \sigma_y]$ (anisotropic). Efficient and numerically stable for smoothing.

4. Try anisotropic Gaussian smoothing to see directional effects.

```
I_gX = imgaussfilt(I, [2 0.5]);  % more smoothing in x than y
I_gY = imgaussfilt(I, [0.5 2]);  % more smoothing in y than x
imshowpair(I_gX, I_gY, 'montage');
title('Anisotropic Gaussian: [2 0.5] (left) vs [0.5 2] (right)');
```

5. Edge emphasis with Laplacian or Laplacian-of-Gaussian (LoG).

```
% Pure Laplacian (alpha controls shape; try 0 to 0.4)
h_lap = fspecial('laplacian', 0.2);
I_lap = imfilter(I, h_lap);            % edge emphasis (can be negative)
imshow(I_lap, []); title('Laplacian response');

% Laplacian of Gaussian (LoG) - smoothing + second derivative
h_log = fspecial('log', [7 7], 1.0);   % size and sigma
I_log = imfilter(I, h_log);
imshow(I_log, []); title('LoG response (7x7, sigma=1.0)');
```

Function Hint

`fspecial('laplacian', alpha)`: Returns a discrete Laplacian kernel; parameter `alpha` modulates the central weight. It highlights regions of rapid intensity change and can produce negative outputs.
`fspecial('log', [m n], sigma)`: Returns a Laplacian-of-Gaussian kernel that smooths first, then applies a second derivative, enhancing edges while reducing small-scale noise.

6. Optional diagnostics: compare histograms or simple statistics.

```
figure; imhist(im2uint8(I));     title('Histogram: Original');
figure; imhist(im2uint8(I_gB));  title('Histogram: Gaussian (sigma=1)');
figure; imhist(im2uint8(I_avg)); title('Histogram: Average (5x5)');

% Simple variance proxy for residual noise (lower often means smoother)
vI    = var(I(:));
vAvg  = var(I_avg(:));
vG    = var(I_gB(:));
[vI, vAvg, vG]
```

Function Hint

`imhist(I8)`: Displays the histogram of an integer-type image (use `im2uint8` for conversion). Histograms help interpret dynamic range and contrast changes after filtering.
`var(I(:))`: Computes a scalar variance of all pixel values — a rough proxy for overall variation. Use cautiously, since a smaller variance may also indicate over-smoothing.

4.4 Discussion

Average and Gaussian both reduce noise, but Gaussian typically produces fewer artifacts and preserves structures better for the same effective scale. LoG enhances edges but can introduce ringing and amplify noise if σ is too small. Parameter choices should be justified by the task and by evidence such as visual inspection and simple metrics.

Exploration & Inquiry

Exploration & Inquiry

- Sweep kernel size and σ (for example, $\sigma \in \{0.5, 1, 2\}$, sizes $\in \{3, 5, 9\}$). For each setting, save the output and a histogram. Summarize which settings best suppress noise while preserving edges.
- Compare average vs. Gaussian at matched effective blur. Where do they diverge perceptually, and why.
- Use anisotropic Gaussian to smooth along one direction while preserving detail in the orthogonal direction. Propose a scenario where this is beneficial (for example, suppressing horizontal banding).
- Combine smoothing and sharpening: *Gaussian* $\rightarrow$ *Laplacian add-back*. Document when this improves detail without overamplifying noise.

5 Activity 3 — Boundary Padding Options

5.1 Objective

Investigate how different boundary padding strategies influence the output of spatial filtering operations.

5.2 Description

When filtering an image, pixels near the boundary lack neighbors outside the image region. MATLAB handles these cases by extending the image boundaries according to a specified padding mode. The padding choice can subtly or significantly affect the results near the edges, especially for large kernels. In this activity, you will explore and compare several padding options.

5.3 Steps

1. Read and display the image to be filtered.

```
I = imread('coins.png');
I = im2double(I);
imshow(I, []); title('Original');
```

2. Create a Gaussian filter.

```
h = fspecial('gaussian', [7 7], 1.0);
```

3. Apply the same filter using different padding modes.

```
    I_zeros     = imfilter(I, h, 'zeros');
    I_replicate = imfilter(I, h, 'replicate');
    I_symmetric = imfilter(I, h, 'symmetric');
    I_circular  = imfilter(I, h, 'circular');
```

4. Display and compare results side by side.

```
figure;
subplot(2,2,1); imshow(I_zeros, []);     title('Zeros Padding');
subplot(2,2,2); imshow(I_replicate, []); title('Replicate Padding');
subplot(2,2,3); imshow(I_symmetric, []); title('Symmetric Padding');
subplot(2,2,4); imshow(I_circular, []);  title('Circular Padding');
```

Function Hint

`imfilter(I, h, padding)`: Applies correlation (or convolution if specified) using the chosen `padding` mode. Common options include:

- `'zeros'` — pads with zeros (default), may darken borders.
- `'replicate'` — extends border pixels outward, preserves brightness near edges.
- `'symmetric'` — mirrors the image at the boundary, often produces the smoothest transitions.
- `'circular'` — wraps around, treating the image as periodic.

5.4 Discussion

Different padding methods produce distinct edge behaviors. `'zeros'` may introduce dark halos, `'replicate'` and `'symmetric'` tend to preserve natural edges, and `'circular'` can create artificial continuity across borders. The optimal padding strategy depends on the context — for instance, `'replicate'` for natural images or `'circular'` for frequency-domain consistency.

Exploration & Inquiry

Exploration & Inquiry

- Create a simple test pattern (for example, a white square on black) and visualize how each padding mode alters the result.
- Quantify the mean pixel intensity in a thin border region for each padding mode; interpret the differences.
- Propose guidelines for selecting padding strategies in denoising versus edge-detection tasks.
- Combine padding experimentation with varying kernel sizes to observe compounded border effects.

6 Activity 4 — Nonlinear Filtering for Noise Removal

6.1 Objective

Apply nonlinear filtering to reduce noise while preserving edges. Compare the performance of the median filter with the averaging filter under different noise conditions.

6.2 Description

Linear filters such as the averaging or Gaussian filters reduce random noise but often blur edges. Nonlinear filters, like the median filter, are more robust against outliers such as salt-and-pepper noise. In this activity, you will add synthetic noise to an image, apply different filters, and evaluate their effectiveness visually and quantitatively.

6.3 Steps

1. Read and display a grayscale image.

```
I = imread('coins.png');
I = im2double(I);
imshow(I, []); title('Original Image');
```

2. Add salt-and-pepper noise to the image using `imnoise`.

   ```
   I_noisy = imnoise(I, 'salt & pepper', 0.05);   % 5% noise density
   imshow(I_noisy, []); title('Noisy Image (Salt & Pepper)');
   ```

3. Apply a 3x3 averaging filter using `fspecial` and `imfilter`.

   ```
   h_avg = fspecial('average', [3 3]);
   I_avg = imfilter(I_noisy, h_avg);
   imshow(I_avg, []); title('Averaging Filter Result');
   ```

4. Apply a 3x3 median filter using `medfilt2`.

   ```
   I_med = medfilt2(I_noisy, [3 3]);
   imshow(I_med, []); title('Median Filter Result');
   ```

5. Compare all results visually.

   ```
   figure;
   subplot(1,3,1); imshow(I_noisy, []); title('Noisy');
   subplot(1,3,2); imshow(I_avg, []);   title('Averaged');
   subplot(1,3,3); imshow(I_med, []);   title('Median');
   ```

6. Optionally, evaluate numerically (if the clean image is known).

   ```
   rmse_avg = sqrt(mean((I(:) - I_avg(:)).^2));
   rmse_med = sqrt(mean((I(:) - I_med(:)).^2));
   [rmse_avg rmse_med]
   ```

Function Hint

`imnoise(I, 'salt & pepper', d)`: Adds salt-and-pepper noise with density `d` (fraction of pixels affected). Useful for testing robustness of nonlinear filters.
`fspecial('average', [m n])`: Creates an averaging filter for linear smoothing.
`medfilt2(I, [m n])`: Applies a median filter with neighborhood size $[m \times n]$; replaces each pixel with the median of its neighborhood, effectively removing isolated spikes while preserving edges.

6.4 Discussion

The median filter effectively removes impulse noise without significantly blurring edges, whereas the averaging filter smooths the entire image uniformly. The difference becomes more evident when noise density increases. Median filtering is nonlinear, meaning it cannot be represented as convolution but often provides perceptually superior results on images with discrete noise artifacts.

Exploration & Inquiry

Exploration & Inquiry

- Mix Gaussian and salt-and-pepper noise using multiple calls to `imnoise`. Compare the visual and numerical performance of average and median filters under each noise type.
- Experiment with different window sizes for `medfilt2` (for example, $[3\,3]$, $[5\,5]$, and $[7\,7]$) and observe trade-offs between smoothness and edge preservation.
- Try alternative padding modes for the median filter using the `'symmetric'` or `'replicate'` options. Observe border behavior and artifacts.
- Combine linear and nonlinear filters sequentially (for example, Gaussian smoothing followed by median filtering) and discuss the effects.

7 Activity 5 — RGB Filtering and Gradient-Based Features (Optional Challenge)

7.1 Objective

Extend spatial filtering concepts from grayscale images to color (RGB) images, and analyze how smoothing and sharpening influence gradient-based features.

7.2 Description

Color images consist of three channels (red, green, and blue). Filtering can be applied independently to each channel or to a luminance-converted grayscale image. This activity explores the effect of Gaussian smoothing and Laplacian sharpening on color images and examines the impact of filtering on gradient magnitude and direction.

7.3 Steps

1. Read and display a color image.

```
I = imread('peppers.png');
I = im2double(I);
imshow(I); title('Original RGB Image');
```

2. Apply Gaussian smoothing using `imgaussfilt`.

```
I_smooth = imgaussfilt(I, 1.5);  % sigma = 1.5
imshow(I_smooth); title('Smoothed RGB Image');
```

3. Apply sharpening using `fspecial('laplacian')` and `imfilter`.

```
h_lap = fspecial('laplacian', 0.2);
I_lap = imfilter(I_smooth, h_lap, 'replicate');
I_sharp = I_smooth - I_lap;
imshow(I_sharp); title('Sharpened RGB Image');
```

4. Compute gradient magnitude before and after smoothing.

```
I_gray = rgb2gray(I);
I_gray_smooth = rgb2gray(I_smooth);
[Gmag1, Gdir1] = imgradient(I_gray);
[Gmag2, Gdir2] = imgradient(I_gray_smooth);

figure;
subplot(1,2,1); imshow(Gmag1, []); title('Gradient: Original');
subplot(1,2,2); imshow(Gmag2, []); title('Gradient: After Smoothing');
```

5. Compare gradient statistics to quantify noise reduction.

```
meanG1 = mean(Gmag1(:));
meanG2 = mean(Gmag2(:));
[meanG1 meanG2]
```

Function Hint

`imgaussfilt(I, sigma)`: Performs Gaussian smoothing on image `I` with standard deviation `sigma`. Use a larger `sigma` for stronger blur.
`fspecial('laplacian', alpha)`: Creates a Laplacian kernel; smaller `alpha` increases edge sensitivity.
`imfilter(I, h)`: Applies the filter `h` to `I` with optional padding and convolution settings.
`imgradient(I)`: Returns gradient magnitude and direction of image `I`; useful for edge detection and texture analysis.
`rgb2gray(I)`: Converts RGB images to grayscale using luminance weighting.

7.4 Discussion

Gaussian smoothing before gradient computation reduces noise amplification, yielding cleaner and more stable edge maps. However, excessive smoothing can blur fine features. Sharpening enhances edges but may amplify noise if not balanced properly. In RGB processing, filtering each channel independently may cause minor color shifts, while grayscale-based filtering ensures uniformity but can sacrifice chromatic detail.

Exploration & Inquiry

Exploration & Inquiry

- Compare filtering each RGB channel separately versus converting to grayscale first. Analyze which produces more natural results.
- Try different filter orders: *smooth* then *sharpen*, versus *sharpen* then *smooth*. Discuss the perceptual and numerical differences.
- Compute gradients for each color channel independently and visualize them. Which channel contributes most to edge detection?
- Design your own three-step enhancement pipeline (for example, Gaussian $\to$ Laplacian $\to$ Gradient) and justify each stage.

8 Reflection and Discussion

8.1 Purpose

The reflection section encourages you to connect practical experimentation with theoretical understanding. You are expected to analyze your results critically, support your statements with visual or numerical evidence, and discuss the implications of your findings for real-world vision systems.

8.2 Guiding Questions

1. **Correlation vs. Convolution:** How do their results differ when using asymmetric kernels? Under what conditions do they produce identical outputs?
2. **Linear vs. Nonlinear Filtering:** In which cases does the median filter outperform the average filter? What is the cost of this improvement in terms of computation and texture preservation?
3. **Padding Strategies:** Which padding mode produced the most visually natural borders? How might padding choice affect quantitative tasks like edge counting or intensity statistics?
4. **Parameter Sensitivity:** How do kernel size and standard deviation (σ) influence smoothing and edge sharpness? How can one balance noise removal against detail loss?
5. **Color and Gradients:** How does filtering in RGB space differ from filtering in grayscale space? Which approach would you recommend for feature extraction or edge detection, and why?
6. **Personal Insights:** Which exploratory test surprised you the most, and what did it reveal about filter behavior or image structure?