National University of Computer & Emerging Sciences, Karachi
Artificial Intelligence-School of Computing
Fall 2024, Lab Manual - 03

Course Code (AI4002)	**Course: Computer Vision Lab**
Instructor(s):	**Muhammad Khalid**

Objectives:

1. Understand the concept of Linear & non-linear filtering and its applications in image processing.

1. Linear Filtering

Linear filtering is a fundamental concept in digital image processing that involves modifying an image by applying a linear operation called a filter or kernel. These filters are used to enhance or suppress certain features in an image, such as noise reduction, edge detection, and image sharpening. Linear filtering works by convolving the image with the filter kernel, which involves sliding the kernel over the image and computing a weighted sum of pixel values at each position.

Some key aspects of linear filtering in digital image processing:

Filter Kernel: A filter kernel is a small matrix, usually square, that defines the weights to be applied to the pixels in the image during convolution. The size of the kernel determines the extent of the neighborhood around each pixel that is considered during the filtering process. Common filter sizes include 3x3 and 5x5.

Convolution Operation: To apply a filter to an image, you slide the kernel over the entire image. At each position, you perform a point-wise multiplication between the kernel and the corresponding pixel values in the image and then sum up the results. This sum becomes the new value of the pixel at the center of the kernel.

Filter Types

- **Smoothing Filters:** These filters are used to reduce noise and blur an image. Common smoothing filters include the Gaussian filter and the mean filter.
- **Edge Detection Filters:** These filters highlight edges and boundaries in an image. Examples include the Sobel and Prewitt filters.
- **Sharpening Filters:** Sharpening filters enhance edges and fine details in an image. The Laplacian filter is an example.
- **Custom Filters:** You can create custom filters with specific weightings to achieve desired image processing effects.
- **Border Handling:** Handling image borders during convolution is important. There are different methods, such as zero-padding (setting border pixels to zero), mirror padding, or using only the valid part of the convolution result.
- **Filtering in Frequency Domain:** Convolution in the spatial domain can be computationally expensive, especially for large images and kernels. In some cases, it is more efficient to perform filtering in the frequency domain using techniques like the Fourier Transform.

- **Filtering Libraries:** Various programming languages and libraries provide functions for applying linear filters to images. Popular choices include OpenCV (Python), MATLAB, and image processing libraries in languages like C++ and Java.

Applications: Linear filtering is widely used in image enhancement, computer vision, medical image processing, and various other fields to extract valuable information from images or prepare them for further analysis.

2. Smoothing Filters

Smoothing linear filters in digital image processing are used to reduce noise, blur an image, and eliminate fine details. These filters work by averaging or weighting the pixel values in the neighborhood of each pixel to create a smoother and less noisy image.

1. **Mean Filter (Box Filter):**
   Working Mechanism: The mean filter replaces each pixel's value with the average of the pixel values in its neighborhood. It uses a square kernel (usually 3x3 or 5x5) and computes the average of the pixel values covered by the kernel.
2. **Gaussian Filter:**
   Working Mechanism: The Gaussian filter applies Gaussian smoothing to the image. It uses a Gaussian kernel, which emphasizes the central pixel and diminishes the influence of distant pixels. This creates a smoothing effect while preserving edges.

3. Working Mechanism for Different Edge Detection Filters

1. **Sobel Filter:**
   Working Mechanism: The Sobel filter calculates the gradient of an image to detect edges. It uses two 3x3 convolution kernels, one for detecting vertical edges and the other for horizontal edges. The gradient magnitude is computed as the square root of the sum of squares of the two gradients.

2. **Canny Edge Detector:**
   Working Mechanism: The Canny edge detector involves several steps, including Gaussian smoothing, gradient calculation, non-maximum suppression, and edge tracking by hysteresis. It identifies edges as regions where the gradient magnitude is above a certain threshold and connects them to form continuous edges.

3. **Laplacian of Gaussian (LoG):**
   Working Mechanism: The LoG filter first applies Gaussian smoothing to the image to reduce noise and then calculates the Laplacian (second derivative) of the smoothed image to find areas of rapid intensity change, which correspond to edges.

Smoothing Filters	Edge Detection Filters
import cv2	import cv2

# Load an image 	
image = cv2.imread('image.jpg')
 	
# Apply a 5x5 Mean (Box) Filter 	
mean_filtered_image = cv2.blur(image, (5, 5))
 	
# Apply Gaussian Smoothing 	
sigma = 1.5 # Adjust the sigma parameter for smoothing strength
gaussian_filtered_image = cv2.GaussianBlur(image, (0, 0), sigma)
 	
# Display the original and filtered images 	
cv2.imshow('Original Image', image)
cv2.imshow('Mean Filtered Image', mean_filtered_image)
 	
cv2.waitKey(0)
cv2.destroyAllWindows()
	# Load an image
	image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE) # Convert to grayscale for edge detection
	
	# Apply Sobel Filter to Detect Edges
	sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=5)
	sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=5)
	
	# Calculate the magnitude of the gradient
	edge_image = cv2.magnitude(sobel_x, sobel_y)
	
	# Apply Canny Edge Detector
	canny_edge_image = cv2.Canny(image, 100, 200) # Adjust threshold values as needed
	
	# Apply Gaussian Smoothing
	sigma = 1.5 # Adjust the sigma parameter for smoothing strength
	smoothed_image = cv2.GaussianBlur(image, (0, 0), sigma)
	
	# Apply Laplacian Filter for Edge Detection
	laplacian = cv2.Laplacian(smoothed_image, cv2.CV_64F)
	
	# Display the original and edge-detected images
	cv2.imshow('Original Image', image)
	cv2.imshow('Edge Image (Sobel)', edge_image.astype(np.uint8))
	
	cv2.waitKey(0)
	cv2.destroyAllWindows()

4. Sharpening Liner Filters and their Working Mechanism

Sharpening linear filters in digital image processing are used to enhance the fine details and edges in an image, making them appear more pronounced and crisp. These filters work by emphasizing the high-frequency components (such as edges) in the image while reducing the low-frequency components (such as smooth areas).

1. Laplacian Filter:

Working Mechanism: The Laplacian filter enhances edges by highlighting areas where there is a rapid change in intensity values. It calculates the second derivative of the image, emphasizing regions with abrupt intensity transitions.

2. **Unsharp Masking (High-pass Filter):**

   Working Mechanism: Unsharp masking enhances details by subtracting a blurred version of the image from the original image. It accentuates the differences between neighboring pixels, effectively increasing contrast. The unsharp masking process involves convolving the image with a smoothing (blurring) kernel and then subtracting the smoothed image from the original image.

3. **High-Boost Filtering:**

   Working Mechanism: High-boost filtering is an extension of unsharp masking. It applies a weighted version of the Laplacian filter to the original image. The weight factor controls the degree of sharpening.
   High-Boost = A * Original - Blurred
   Here, "A" is a user-defined constant that determines the strength of sharpening.

4. **Gradient-based Filters (Sobel, Prewitt, Scharr):**

   Working Mechanism: Gradient-based filters, like Sobel, Prewitt, and Scharr, can also be used for sharpening. By applying these filters, you calculate the gradient of the image, which represents the rate of change of intensity. Edges are enhanced because they have high gradients.

import cv2
import numpy as np
 	
# Load an image 	
image = cv2.imread('image.jpg')
 	
# Laplacian Filter for Sharpening 	
laplacian = cv2.Laplacian(image, cv2.CV_64F)
sharpened_image_laplacian = cv2.add(image, laplacian)
sharpened_image_laplacian = np.clip(sharpened_image_laplacian, 0, 255).astype(np.uint8)
 	
# Unsharp Masking (High-pass Filter) for Sharpening 	
blurred_image = cv2.GaussianBlur(image, (5, 5), 0)
sharpened_image_unsharp = cv2.addWeighted(image, 2, blurred_image, -1, 0)
 	
# High-Boost Filtering for Sharpening 	
A = 2 # Adjust this value for the desired sharpening strength
sharpened_image_high_boost = cv2.addWeighted(image, A + 1, blurred_image, -A, 0)
	The Laplacian filter enhances edges by calculating the Laplacian of the image.
	Unsharp masking subtracts a blurred version of the image from the original to enhance details.
	High-boost filtering allows you to adjust the sharpening strength using the "A" parameter.
	You can adjust the parameters and experiment with different values of "A" to achieve the desired sharpening effect.

# Display the original and sharpened images 	
cv2.imshow('Original Image', image)
cv2.imshow('Laplacian Sharpened Image', sharpened_image_laplacian)
cv2.imshow('Unsharp Masking Sharpened Image', sharpened_image_unsharp)
cv2.imshow('High-Boost Sharpened Image', sharpened_image_high_boost)
 	
cv2.waitKey(0)
cv2.destroyAllWindows()	

5. Border Handling Linear filtering

When applying linear filters to images in Python using libraries like OpenCV, you may need to handle the borders of the image. The border handling methods determine how to handle pixels near the image edges where the filter kernel extends beyond the image boundaries.

```
import cv2
import numpy as np

# Load an image
image = cv2.imread('image.jpg')

# Define a simple 3x3 kernel
kernel = np.array([[0, 1, 0],
                   [1, 5, 1],
                   [0, 1, 0]], dtype=np.float32)  # Example kernel, you can define your own

# Normalize the kernel to ensure the sum of its elements is 1
kernel /= kernel.sum()

# Linear filtering with various border handling methods
filtered_image_replicate = cv2.filter2D(image, -1, kernel, borderType=cv2.BORDER_REPLICATE)
filtered_image_constant = cv2.filter2D(image, -1, kernel, borderType=cv2.BORDER_CONSTANT, borderValue=(0, 0, 0))
filtered_image_wrap = cv2.filter2D(image, -1, kernel, borderType=cv2.BORDER_WRAP)
filtered_image_reflect = cv2.filter2D(image, -1, kernel, borderType=cv2.BORDER_REFLECT)

# Display the results
cv2.imshow('Original Image', image)
```

BORDER_REPLICATE: It replicates the border pixels to extend the image. This method is useful when you want to maintain the border pixel values.

BORDER_CONSTANT: It pads the image with a constant value (specified by borderValue). This is useful when you want to set a specific background color around the image.

BORDER_WRAP: It wraps the image around as if it's a torus, allowing the filter to continue from one edge to the opposite edge.

BORDER_REFLECT: It reflects the border pixels, which can help reduce artifacts in the filtered image.

```
cv2.imshow('Filtered (Replicate)', filtered_image_replicate)
cv2.imshow('Filtered (Constant)', filtered_image_constant)
cv2.imshow('Filtered (Wrap)', filtered_image_wrap)
cv2.imshow('Filtered (Reflect)', filtered_image_reflect)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

6. Non-Linear Filtering:

Explanation: Non-linear filters are image processing techniques that use a non-linear mathematical function to process pixel values. Unlike linear filters (e.g., blurring, sharpening), non-linear filters consider neighboring pixel values in a non-linear way.

Real-Life Example: Removing salt-and-pepper noise from a photograph is a common real-life application of non-linear filtering.

1. **Median Filter:**
   The median filter replaces each pixel's value with the median value of the pixel values in its neighborhood. It's effective for salt-and-pepper noise removal
2. **Minimum Filter (Min Filter):**
   The minimum filter replaces each pixel's value with the minimum value among the pixel values in its neighborhood.
3. **Maximum Filter (Max Filter):**
   The maximum filter replaces each pixel's value with the maximum value among the pixel values in its neighborhood.
4. **Bilateral Filter:**
   The bilateral filter is a non-linear filter that smooths an image while preserving edges.

```
# Apply Median Filter
median_filtered_image = cv2.medianBlur(image, 5)  # Adjust kernel size as needed

kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
# Apply Minimum Filter
min_filtered_image = cv2.erode(image, kernel)

# Apply Maximum Filter
max_filtered_image = cv2.dilate(image, kernel)

# Apply Bilateral Filter
bilateral_filtered_image = cv2.bilateralFilter(image, d=9, sigmaColor=75, sigmaSpace=75)
```

Characteristic	Linear Filters	Non-Linear Filters

Linearity	Linear filters apply weighted sums of pixel values. The output at a pixel is a linear combination of its neighbors. Examples include mean, Gaussian, Sobel, etc.	Non-linear filters use non-linear operations on pixel values, making the output dependent on pixel ranking or other non-linear criteria. Examples include median, min, max, bilateral, etc.
Noise Reduction	Effective for reducing Gaussian noise and blurring an image while preserving linear structures.	Effective for removing non-Gaussian noise like salt-and-pepper noise and preserving details and edges.
Edge Preservation	Linear filters may smooth or blur edges in an image, making them less distinct.	Non-linear filters are better at preserving edges and details while reducing noise.
Computational Complexity	Generally computationally less expensive.	Can be more computationally intensive, especially with large kernels or adaptive filtering.
Common Examples	Mean filter, Gaussian filter, Sobel filter, Laplacian filter, etc.	Median filter, Min filter, Max filter, Bilateral filter, Adaptive median filter, Non-Local Means (NLM) filter, etc.
Application	Commonly used for tasks like basic smoothing, blurring, and gradient calculation.	Used for tasks like noise reduction, detail preservation, and edge enhancement.
Sensitivity to Noise	May not be effective against impulse noise (e.g., salt-and-pepper noise).	Effective against impulse noise and other non-Gaussian noise types.
Control Parameters	Typically controlled by filter size and kernel weights.	Controlled by filter size and the specific non-linear operation (e.g., median, min, max).

Tasks

1. Complete the above codes where the lines are missing.
2. **Linear Filtering:**
    - Implement a Gaussian blur filter using convolution for image smoothing.
    - Apply a Sobel filter to perform edge detection on a grayscale image.
    - Perform image sharpening using the Laplacian filter.
    - Implement a mean filter for noise reduction in an image.
3. **Non-Linear Filtering:**
    - Develop a median filter for removing salt-and-pepper noise from an image.
    - Apply a max filter to perform dilation on a binary image.
    - Implement a min filter to perform erosion on a binary image.
    - Create a bilateral filter for edge-preserving smoothing.
    - Implement an adaptive median filter for noise reduction while preserving edges.