Department of Electrical Engineering
Indian Institute of Technology Kharagpur

Signal Processing and System Design Laboratory
(EE69205)
Autumn, 2024-25

Experiment 2: Operations on 2D Signals- Image Filtering and Processing

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.

Introduction

Assignments to Solve and Report

Image filtering is a crucial aspect of image processing, used for tasks such as noise reduction, edge detection, and feature extraction. In this experiment, students will explore different filtering techniques and understand their effects on images.
The grayscale image can be downloaded from here

Part 1: Applying Low-Pass Filters

In this part of the experiment, you will add salt and pepper noise to a grayscale image and then apply three different low-pass filters. The filters to be used are:

1. **Mean Filter**
2. **Gaussian Filter**
3. **Median Filter**

Step-by-Step Instructions

1. Read the grayscale image provided from the above link using the below code snippet:

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from skimage.util import random_noise
from scipy.ndimage import gaussian_laplace, maximum_filter
from skimage.feature import canny

file_path = r"path_to/sample_image.tif"
img = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)

# Display the original image
plt.imshow(img, cmap='gray')
plt.title('Original Image')
```

```
plt.axis('off')
plt.show()
```

Listing 1: Load and display Grayscale image

2. Add salt and pepper noise to the image using the given code snippet:

```
# Add noise to the image
# Add salt and pepper noise to the image
salt_prob = 0.02
pepper_prob = 0.05
noisy_img = random_noise(img, mode='s&p', amount=salt_prob +
    pepper_prob)
noisy_img = (255 * noisy_img).astype(np.uint8)  # Convert to 8-bit
    image
plt.imshow(noisy_img, cmap='gray')
plt.title('Noisy Image')
plt.axis('off')
plt.show()
```

Listing 2: Adding Salt and Pepper Noise to the Image

You can try adding other noises such as 'poisson', 'salt', 'pepper', 'speckle'. Refer to the documentation.

3. **Mean Filter**

   - Apply a mean filter to the noisy image using the following 3x3 kernel:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

   - Display the original, noisy, and filtered images side by side.
   - Calculate and comment on the PSNR values before and after denoising the images.

4. **Gaussian Filter**

   - Apply a Gaussian filter to the noisy image using the following 3x3 kernel with standard deviation $\sigma = 1$:

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

   - Display the original, noisy, and filtered images side by side.
   - Calculate and comment on the PSNR values before and after denoising the images.

5. **Median Filter**

   - Apply a median filter to the noisy image. Note that the median filter does not use a specific kernel like mean or Gaussian filters. Instead, it replaces each pixel value with the median value of the 3x3 neighborhood.
   - Display the original, noisy, and filtered images side by side.
   - Calculate and comment on the PSNR values before and after denoising the images.

Part 2: Applying High-Pass Filters

In this part of the experiment, you will add salt and pepper noise to the grayscale image and then apply three different high-pass filters. The filters to be used are:

1. **Laplacian Filter**
2. **Sobel Filter**
3. **Prewitt Filter**

Step-by-Step Instructions

1. Read the grayscale image provided at the link below.
2. Add salt and pepper noise to the image.
3. **Laplacian Filter**
   - Apply a Laplacian filter to the noisy image using the following 3x3 kernel:

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

   - Display the original, noisy, and filtered images side by side.
   - Calculate and comment on the PSNR values before and after denoising the images. .
4. **Sobel Filter**
   - Apply a Sobel filter to the original grayscale image using the following 3x3 kernels for the x and y directions, then compute the gradient magnitude of the resulting image:

$$\text{Sobel-X: } \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{Sobel-Y: } \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

     The magnitude of the gradient G can be computed using the following formula:

$$G = \sqrt{G_x^2 + G_y^2}$$

     where G_x and G_y are the results of applying the Sobel-X and Sobel-Y filters, respectively.
   - Display the original image, the x-gradient, the y-gradient, and the resulting gradient magnitude images side by side.
   - Report your observations for the outputs generated.
5. **Prewitt Filter**
   - Apply a Prewitt filter to the original grayscale image using the following 3x3 kernels for the x and y directions, then compute the gradient magnitude of the resulting image:

$$\text{Prewitt-X: } \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{Prewitt-Y: } \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

The magnitude of the gradient G can be computed using the following formula:

$$G = \sqrt{G_x^2 + G_y^2}$$

where G_x and G_y are the results of applying the Prewitt-X and Prewitt-Y filters, respectively.

- Display the original image, the x-gradient, the y-gradient, and the resulting gradient magnitude images side by side.
- Report your observations for the outputs generated.

Part 3: Using Canny Edge Detector, LoG, and Max Pooling

In this part of the experiment, you will apply the Canny edge detector, Laplacian of Gaussian (LoG), and max pooling to the grayscale image. Each task is independent and you are required to use inbuilt functions from libraries like OpenCV and NumPy.

Canny Edge Detector

- Read the grayscale image provided at the link below.
- Apply the Canny edge detector using the inbuilt function from OpenCV:

```
import cv2
# Apply Canny edge detection
canny_edges = canny(img, sigma=sigma)

# Display the Canny edge detected image
plt.imshow(canny_edges, cmap='gray')
plt.title('Canny Edge Detected Image')
plt.axis('off')
plt.show()

```

- Display the original and edge-detected images side by side.
- Report your observation on the effect of the Canny edge detector on the image.

Laplacian of Gaussian (LoG)

- Read the grayscale image provided at the above link.
- Apply the Laplacian of Gaussian using the inbuilt function from OpenCV:

```
# Apply Laplacian of Gaussian (LoG) filter
sigma = 1
log_filtered_img = gaussian_laplace(img, sigma=sigma)

# Display the LoG filtered image
plt.imshow(log_filtered_img, cmap='gray')
plt.title('LoG Filtered Image')
plt.axis('off')
plt.show()

```

- Display the original and LoG images side by side.
- Report your observation on the effect of the LoG filter on the image.

Max Pooling

- Read the grayscale image provided at the link below.
- Apply max pooling using the inbuilt function from NumPy:

```
# Apply Max Pooling

pool_size = 5
max_pooled_img = maximum_filter(img, size=pool_size)

# Display the Max Pooled image
plt.imshow(max_pooled_img, cmap='gray')
plt.title('Max Pooled Image')
plt.axis('off')
plt.show()

```

- Display the original and pooled images side by side.
- Report your observation on the effect of max pooling on the image.

References

[1] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed., Pearson, 2018.

[2] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis, and Machine Vision*, 4th ed., Cengage Learning, 2014.

[3] S. K. Mitra, *Digital Signal Processing: A Computer-Based Approach*, 4th ed., McGraw-Hill, 2010.

[4] J. Canny, "A Computational Approach to Edge Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986.

[5] W. K. Pratt, *Digital Image Processing: PIKS Scientific Inside*, 4th ed., Wiley-Interscience, 2007.