THE UNIVERSITY OF DANANG
UNIVERSITY OF SCIENCE AND TECHNOLOGY
Faculty of Advanced Science and Technology

Image Processing

Midterm Examination Report

Instructor : TS. Hồ Phước Tiến
Class : 21ES
Group members : Lê Quốc Duy - 123210016
Trần Vũ Hồng Phúc - 123210021

Da Nang, 25th March 2025

MIDTERM - 03/25

Image Processing

Students:

1. Le Quoc Duy - 123210016

2. Tran Vu Hong Phuc - 123210021

Problem 1. *Filtering in the frequency domain*

Given a grayscale image, filter this image using an ideal low-pass filter. Vary the cut-off frequency of the ideal low-pass filter and observe the results. What can we see in the output images? Illustrate and try to explain the observed phenomena.

We now replace the ideal low-pass filter with the Butterworth and Gaussian low-pass filters and repeat the above filtering. In these cases, are there any differences in the results compared with those from the ideal low-pass filter? Show the results appropriately to illustrate the comparisons.

[Code]

- *Ideal low-pass filter:*
  - ➔ *MATLAB:*

```
clc; clear; close all;
% Step 1: Read grayscale image
f = imread('cat.jpg'); % Change 'lion.jpg' to your image
f = rgb2gray(f); % Convert to grayscale if needed
f = im2double(f); % Convert to double for FFT processing
% Step 2: Compute the Fourier Transform
F = fft2(f);
F = fftshift(F); % Shift zero frequency to center
% Define multiple cutoff frequencies
cutoff_frequencies = [5, 15, 30, 50, 75, 100, 150, 200];
% Get image size
[M, N] = size(f);
[u, v] = meshgrid(-N/2:N/2-1, -M/2:M/2-1);
D = sqrt(u.^2 + v.^2); % Compute frequency distances
% Create figure to display results
figure;
subplot(3, 3, 1);
imshow(f, []);
title('Original Image');
% Loop through each cutoff frequency
for i = 1:length(cutoff_frequencies)
```

```
    D0 = cutoff_frequencies(i); % Get current cutoff
    H = double(D <= D0); % Generate Ideal Low-Pass Filter
    % Apply filter in frequency domain
    G = H .* F;
    % Compute the inverse Fourier Transform
    G = ifftshift(G); % Shift back
    g = ifft2(G); % Inverse FFT
    g = abs(g); % Take absolute value
    % Display filtered image
    subplot(3, 3, i + 1);
    imshow(g, []);
    title(['D0 = ', num2str(D0)]);
end
```

- *Butterworth low-pass filter:*

➔ *MATLAB:*

```
clc; clear; close all;
% Step 1: Read grayscale image
f = imread('cat.jpg'); % Change to your image
f = rgb2gray(f); % Convert to grayscale if needed
f = im2double(f); % Convert to double for FFT processing
% Step 2: Compute the Fourier Transform
F = fft2(f);
F = fftshift(F); % Shift zero frequency to center
% Define multiple cutoff frequencies
cutoff_frequencies = [5, 15, 30, 50, 75, 100, 150, 200];
n = 2; % Butterworth filter order
% Get image size
[M, N] = size(f);
[u, v] = meshgrid(-N/2:N/2-1, -M/2:M/2-1);
D = sqrt(u.^2 + v.^2); % Compute frequency distances
% Create figure to display results
figure;
subplot(3, 3, 1);
imshow(f, []);
title('Original Image');
% Loop through each cutoff frequency
for i = 1:length(cutoff_frequencies)
    D0 = cutoff_frequencies(i); % Get current cutoff
    H = 1 ./ (1 + (D ./ D0).^(2 * n)); % Butterworth LPF
```

```
    % Apply filter in frequency domain
    G = H .* F;
    % Compute the inverse Fourier Transform
    G = ifftshift(G); % Shift back
    g = ifft2(G); % Inverse FFT
    g = abs(g); % Take absolute value
    % Display filtered image
    subplot(3, 3, i + 1);
    imshow(g, []);
    title(['D0 = ', num2str(D0)]);
end
```

- *Gaussian low-pass filter:*

➔ *MATLAB:*

```
clc; clear; close all;
% Step 1: Read grayscale image
f = imread('cat.jpg'); % Change to your image
f = rgb2gray(f); % Convert to grayscale if needed
f = im2double(f); % Convert to double for FFT processing
% Step 2: Compute the Fourier Transform
F = fft2(f);
F = fftshift(F); % Shift zero frequency to center
% Define multiple cutoff frequencies
cutoff_frequencies = [5, 15, 30, 50, 75, 100, 150, 200];
% Get image size
[M, N] = size(f);
[u, v] = meshgrid(-N/2:N/2-1, -M/2:M/2-1);
D = sqrt(u.^2 + v.^2); % Compute frequency distances
% Create figure to display results
figure;
subplot(3, 3, 1);
imshow(f, []);
title('Original Image');
% Loop through each cutoff frequency
for i = 1:length(cutoff_frequencies)
    D0 = cutoff_frequencies(i); % Get current cutoff
    H = exp(-(D.^2) / (2 * (D0^2))); % Gaussian LPF
    % Apply filter in frequency domain
```

```
        G = H .* F;
        % Compute the inverse Fourier Transform
        G = ifftshift(G); % Shift back
        g = ifft2(G); % Inverse FFT
        g = abs(g); % Take absolute value
        % Display filtered image
        subplot(3, 3, i + 1);
        imshow(g, []);
        title(['D0 = ', num2str(D0)]);
    end
```

[Figures/Tables]

- *Ideal low-pass filter:*

- *Butterworth low-pass filter:*

- *Gaussian low-pass filter:*

[Explanation]

- ***Observations from the Output Image using Ideal Low-Pass Filter:***
  1. *Original Image:*
     - *The first image (top-left) is the **original grayscale image**, showing clear details with sharp edges and textures.*
  2. *Effect of Low Cutoff Frequencies (D0 = 5, 15, 30):*
     - *The images become **highly blurred**.*
     - *Fine details and sharp edges are completely lost.*
     - *Only **low-frequency components** (smooth variations) remain.*
     - *The image appears as a rough shape with no textures.*
  3. *Medium Cutoff Frequencies (D0 = 50, 75):*
     - *The images start to recover more details.*
     - *Some edges and textures become visible.*
     - *The transition from **blurry to sharper** is noticeable.*
     - *Still, some high-frequency details (like fine textures) are missing.*
  4. *High Cutoff Frequencies (D0 = 100, 150, 200):*
     - *The images **look sharper and clearer**.*
     - *More high-frequency components are retained.*
     - *The difference between these images and the original is **less noticeable**.*
     - *At **D0 = 200**, the image is **almost identical to the original**, meaning most frequency components are passed.*

- ***Explanation of the Observed Phenomena:***
  - *Ideal Low-Pass Filtering:*
    - *This filter **removes high-frequency components** beyond a certain cutoff frequency **D0**.*
    - *High frequencies in an image represent **sharp edges, fine textures, and small details**.*
    - *The **lower the cutoff frequency (D0)**, the more **blurred** the image becomes because **more high frequencies are removed**.*
  - *Higher Cutoff Frequencies Lead to Higher Resolution:*
    - *When **D0 is increased**, the filter allows **more high-frequency components** to pass.*

- *This means **more edges and fine details are preserved**, resulting in a **sharper image**.*
- *When **D0 is very large (D0 ≈ size of image)**, almost all frequencies pass, and the filtered image becomes **nearly identical to the original**.*

- **<u>Are there any differences in the results using Butterworth and Gaussian low-pass filters from the ideal low-pass filter?</u>**
  - **Butterworth LPF** is more gradual, avoiding sharp transitions.
  - **Gaussian LPF** results in even smoother filtering with **less ringing effects** than Butterworth.
  - **Both filters** preserve more details than the **Ideal LPF**, especially at lower cutoff frequencies.
  - **Ideal LPF** is not smooth; it causes ringing artifacts (Gibbs phenomenon) due to the sharp cutoff in frequency space**.**
  - In practical applications, **Butterworth and Gaussian LPFs are often preferred** because they provide smoother transitions and fewer artifacts.

Problem 2. Simulate a 2D mosaic image for a color image (as in Alleysson's paper). Show the magnitude spectrum of this mosaic image. Compare this spectrum with the magnitude spectrum of a natural image. Explain the eventual differences.

[Code] -MATLAB

```
clc; clear; close all;
% Load the image 'kodim22t.jpg'
try
   original_img = imread('kodim22t.jpg');
   % Keep a copy of the original RGB image for comparison
   rgb_img = im2double(original_img);
catch
   error('Could not load image file kodim22t.jpg');
end
% Get the dimensions of the image
[rows, cols, ~] = size(rgb_img);
%% Step 1: Convert RGB image into Bayer CFA pattern (Mosaic)
cfa_img = zeros(rows, cols);
% Sample R, G, B according to Bayer pattern RGGB
```

```
for i = 1:rows
   for j = 1:cols
       if mod(i, 2) == 1 && mod(j, 2) == 1      % Red at (odd, odd)
           cfa_img(i, j) = rgb_img(i, j, 1);
       elseif mod(i, 2) == 0 && mod(j, 2) == 0  % Blue at (even, even)
           cfa_img(i, j) = rgb_img(i, j, 3);
       else                                     % Green at (odd, even) or
(even, odd)
           cfa_img(i, j) = rgb_img(i, j, 2);
       end
   end
end
% Create a color visualization of the Bayer pattern
bayer_colored = zeros(rows, cols, 3);
for i = 1:rows
   for j = 1:cols
       if mod(i, 2) == 1 && mod(j, 2) == 1      % Red at (odd, odd)
           bayer_colored(i, j, 1) = cfa_img(i, j);
       elseif mod(i, 2) == 0 && mod(j, 2) == 0  % Blue at (even, even)
           bayer_colored(i, j, 3) = cfa_img(i, j);
       else                                     % Green at (odd, even) or
(even, odd)
           bayer_colored(i, j, 2) = cfa_img(i, j);
       end
   end
end
%% Step 2: Apply 2D Discrete Fourier Transform (DFT)
% Compute the Fourier Transform of the original image (in grayscale)
gray_original = rgb2gray(rgb_img); % Convert to grayscale
S_orig = fft2(gray_original);
S_orig_shift = fftshift(S_orig);
S_orig_magnitude = abs(S_orig_shift);
% Compute the Fourier Transform of the Bayer CFA pattern
S_cfa = fft2(cfa_img);
S_cfa_shift = fftshift(S_cfa);
S_cfa_magnitude = abs(S_cfa_shift);
%% Step 3: Display Results
figure('Position', [100, 100, 1000, 500]);
```

```
% Original RGB Image
subplot(2, 2, 1);
imshow(rgb_img);
title('Original Image', 'FontSize', 12);
% Magnitude Spectrum of Original Image
subplot(2, 2, 2);
scaled_orig_mag = S_orig_magnitude / max(S_orig_magnitude(:)); % Normalize
gamma = 0.1; % Apply gamma correction for better visibility
enhanced_orig_mag = scaled_orig_mag.^gamma;
imshow(enhanced_orig_mag, []);
colormap(subplot(2, 2, 2), jet);
colorbar;
title('Magnitude Spectrum of Original Image', 'FontSize', 12);
% Bayer Mosaic Pattern
subplot(2, 2, 3);
imshow(bayer_colored);
title('Bayer Mosaic Pattern', 'FontSize', 12);
% Magnitude Spectrum of Bayer CFA
subplot(2, 2, 4);
scaled_cfa_mag = S_cfa_magnitude / max(S_cfa_magnitude(:)); % Normalize
enhanced_cfa_mag = scaled_cfa_mag.^gamma;
imshow(enhanced_cfa_mag, []);
colormap(subplot(2, 2, 4), jet);
colorbar;
title('Magnitude Spectrum of CFA', 'FontSize', 12);
% Add overall title
sgtitle('Fourier Analysis of Original Image and Bayer Color Filter Array
(CFA)', 'FontSize', 14);
%% Step 4: Print Information about the Spectra
fprintf('Original Image Spectrum - Max magnitude: %f, Mean magnitude: %f,
Ratio: %f\n', ...
       max(S_orig_magnitude(:)), mean(S_orig_magnitude(:)),
max(S_orig_magnitude(:))/mean(S_orig_magnitude(:)));
fprintf('CFA Spectrum - Max magnitude: %f, Mean magnitude: %f, Ratio: %f\n',
...
       max(S_cfa_magnitude(:)), mean(S_cfa_magnitude(:)),
max(S_cfa_magnitude(:))/mean(S_cfa_magnitude(:)));
```

[Figures/Tables] - MATLAB

- *Image Result:*

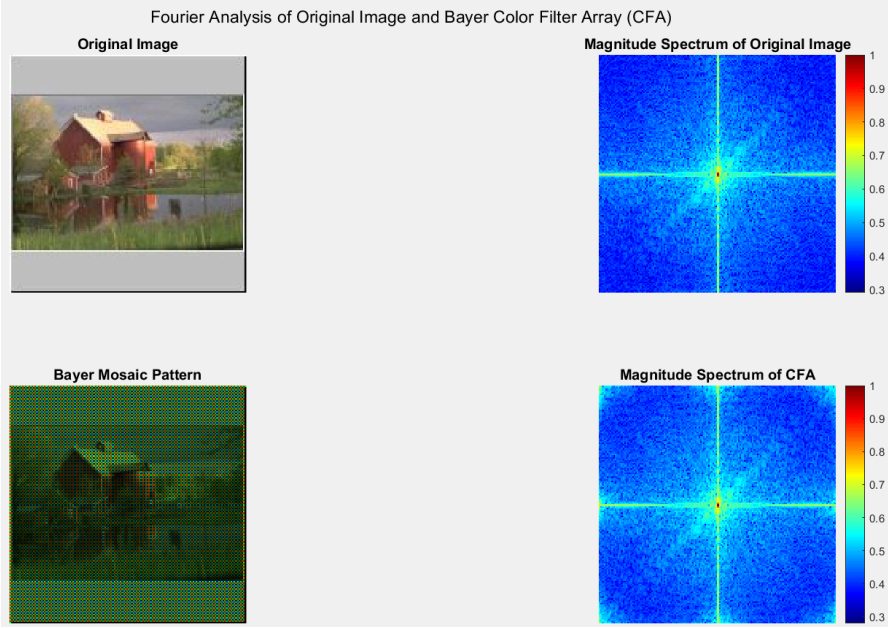

Figure 1: Matlab results - image

- *Display Result:*

```
Original Image Spectrum - Max magnitude: 12557.399453, Mean magnitude: 9.916439, Ratio: 1266.321453
CFA Spectrum - Max magnitude: 12405.560784, Mean magnitude: 10.396103, Ratio: 1193.289504
```

Figure 2: Matlab results - calculation

[Code] - PYTHON

```
import numpy as np
import cv2
from matplotlib.colors import Normalize
# Load the RGB image
try:
    rgb_img = cv2.imread('/media/tairo/Storages/CodeThayTien/XLA/ImgTest/kodim22t.jpg')
    rgb_img = cv2.cvtColor(rgb_img, cv2.COLOR_BGR2RGB)  # Convert to RGB format
    rgb_img = rgb_img / 255.0  # Normalize to [0, 1]
except:
    raise FileNotFoundError('Could not load image file kodim22t.jpg')
```

```
rows, cols, _ = rgb_img.shape
cfa_img = np.zeros((rows, cols)) #Convert RGB image into Bayer CFA pattern (Mosaic)

# Sample R, G, B according to Bayer pattern RGGB
for i in range(rows):
    for j in range(cols):
        if i % 2 == 1 and j % 2 == 1:  # Red at (odd, odd)
            cfa_img[i, j] = rgb_img[i, j, 0]
        elif i % 2 == 0 and j % 2 == 0:  # Blue at (even, even)
            cfa_img[i, j] = rgb_img[i, j, 2]
        else:  # Green at (odd, even) or (even, odd)
            cfa_img[i, j] = rgb_img[i, j, 1]

# Create a color visualization of the Bayer pattern
bayer_colored = np.zeros((rows, cols, 3))
for i in range(rows):
    for j in range(cols):
        if i % 2 == 1 and j % 2 == 1:  # Red at (odd, odd)
            bayer_colored[i, j, 0] = cfa_img[i, j]
        elif i % 2 == 0 and j % 2 == 0:  # Blue at (even, even)
            bayer_colored[i, j, 2] = cfa_img[i, j]
        else:  # Green at (odd, even) or (even, odd)
            bayer_colored[i, j, 1] = cfa_img[i, j]

gray_original = cv2.cvtColor((rgb_img * 255).astype(np.uint8), cv2.COLOR_RGB2GRAY) / 255.0 #Apply 2D Discrete Fourier Transform (DFT)

# Fourier Transform of the original image
S_orig = np.fft.fft2(gray_original)
S_orig_shift = np.fft.fftshift(S_orig)
S_orig_magnitude = np.abs(S_orig_shift)

# Fourier Transform of the Bayer CFA pattern
S_cfa = np.fft.fft2(cfa_img)
```

```
S_cfa_shift = np.fft.fftshift(S_cfa)
S_cfa_magnitude = np.abs(S_cfa_shift)

#Display Results
plt.figure(figsize=(12, 6))

# Original RGB Image
plt.subplot(2, 2, 1)
plt.imshow(rgb_img)
plt.title('Original Image', fontsize=12)
plt.axis('off')

# Magnitude Spectrum of Original Image
plt.subplot(2, 2, 2)
scaled_orig_mag = S_orig_magnitude / np.max(S_orig_magnitude)
gamma = 0.1  # Apply gamma correction
enhanced_orig_mag = scaled_orig_mag ** gamma
plt.imshow(enhanced_orig_mag, cmap='jet', norm=Normalize())
plt.colorbar()
plt.title('Magnitude Spectrum of Original Image', fontsize=12)

# Bayer Mosaic Pattern
plt.subplot(2, 2, 3)
plt.imshow(bayer_colored)
plt.title('Bayer Mosaic Pattern', fontsize=12)
plt.axis('off')

# Magnitude Spectrum of Bayer CFA
plt.subplot(2, 2, 4)
scaled_cfa_mag = S_cfa_magnitude / np.max(S_cfa_magnitude)
enhanced_cfa_mag = scaled_cfa_mag ** gamma
plt.imshow(enhanced_cfa_mag, cmap='jet', norm=Normalize())
plt.colorbar()
plt.title('Magnitude Spectrum of CFA', fontsize=12)
```

```
plt.suptitle('Fourier Analysis of Original Image and Bayer Color
Filter Array (CFA)', fontsize=14)

plt.tight_layout(rect=[0, 0.03, 1, 0.95])

plt.show()

#Print Information about the Spectra

print(f'Original Image Spectrum - Max magnitude:
{np.max(S_orig_magnitude):.6f}, ')

print(f'Mean magnitude: {np.mean(S_orig_magnitude):.6f}, ')

print(f'Ratio: {np.max(S_orig_magnitude) /
np.mean(S_orig_magnitude):.6f}\n')

print(f'CFA Spectrum - Max magnitude: {np.max(S_cfa_magnitude):.6f},
')

print(f'Mean magnitude: {np.mean(S_cfa_magnitude):.6f}, ')

print(f'Ratio: {np.max(S_cfa_magnitude) /
np.mean(S_cfa_magnitude):.6f}\n')
```

[Figures/Tables]

- *Image Result:*

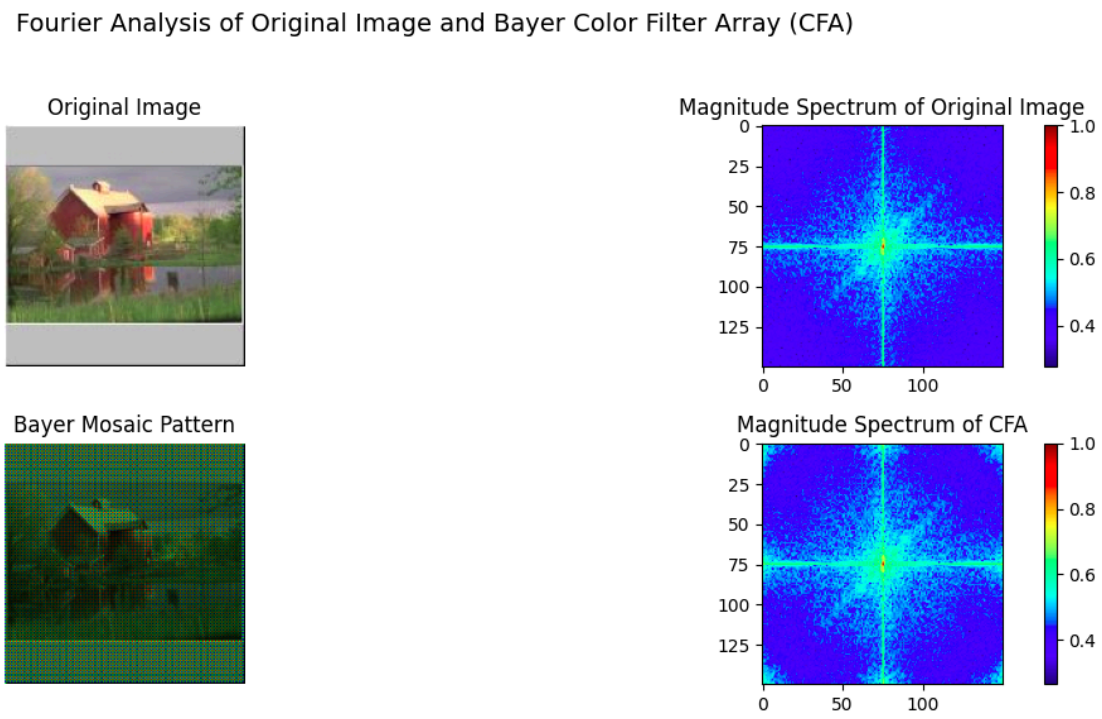

Figure 3: Python Results images

- *Calculation Result:*

Original Image Spectrum – Max magnitude: 12391.858824,

Mean magnitude: 8.394177,

Ratio: 1476.244631

CFA Spectrum – Max magnitude: 12187.145098,

Mean magnitude: 9.257423,

Ratio: 1316.472680

```
#Print Information about the Spectra
print(f'Original Image Spectrum - Max magnitude: {np.max(S_orig_magnitude):.6f}, ')
print(f'Mean magnitude: {np.mean(S_orig_magnitude):.6f}, ')
print(f'Ratio: {np.max(S_orig_magnitude) / np.mean(S_orig_magnitude):.6f}\n')

print(f'CFA Spectrum - Max magnitude: {np.max(S_cfa_magnitude):.6f}, ')
print(f'Mean magnitude: {np.mean(S_cfa_magnitude):.6f}, ')
print(f'Ratio: {np.max(S_cfa_magnitude) / np.mean(S_cfa_magnitude):.6f}\n')
```

✓ Last Execution: 9:08:51 AM, Duration: 0.0s

```
Original Image Spectrum - Max magnitude: 12391.858824,
Mean magnitude: 8.394177,
Ratio: 1476.244631

CFA Spectrum - Max magnitude: 12187.145098,
Mean magnitude: 9.257423,
Ratio: 1316.472680
```

Figure 4: Results in Python - calculation

[Explanation]

- *Observation:*
  - The magnitude spectrum of the **original image** is smooth, with a bright center and a cross pattern due to image structures.
  - The **Bayer mosaic spectrum** has additional **light blue regions in the four corners and along the top/bottom/left/right center**, indicating aliasing.
- *Explanation:*
  - The Bayer filter **subsamples colors non-uniformly**, leading to **spectral replication and aliasing artifacts**.
  - These extra frequency components result from the periodic Bayer pattern, causing **high-frequency distortions** in the magnitude spectrum.

Problem 3. *Filtering in the spatial domain.*

We consider the following filters.

$$F = [1 \quad 2 \quad 4 \quad 2 \quad 1]/10$$

$$G = \frac{1}{10}\begin{bmatrix}1\\2\\4\\2\\1\end{bmatrix}$$

$$H = \frac{1}{100}\begin{bmatrix}1 & 2 & 4 & 2 & 1\\2 & 4 & 8 & 4 & 2\\4 & 8 & 16 & 8 & 4\\2 & 4 & 8 & 4 & 2\\1 & 2 & 4 & 2 & 1\end{bmatrix}$$

Given a grayscale image I, we filter I according to the two following methods. "*" denotes the convolution operation.

Method 1: $Y_1 = I * F$; $Y_2 = Y_1 * G$

Method 2: $Y_3 = I * H$

Compare Y_2 and Y_3. Explain this comparison.

Which method should be used? Explain.

[Code]-MATLAB

```
clc; clear; close all;
% Read the grayscale image
I = imread('cat.jpg'); % Replace with your grayscale image
if size(I,3) == 3  % Check if the image is RGB
```

```
    I = rgb2gray(I); % Convert to grayscale
end
I = double(I); % Convert to double for computations
% Define the given filters
F = (1/10) * [1 2 4 2 1]; % 1D horizontal filter
G = F';                   % 1D vertical filter (transpose of F)
H = (1/100) * [ 1 2 4 2 1;
                2 4 8 4 2;
                4 8 16 8 4;
                2 4 8 4 2;
                1 2 4 2 1]; % 2D filter
%% **Method 1: Separable Filtering**
Y1 = conv2(I, F, 'same'); % Convolve with F (horizontal)
Y2 = conv2(Y1, G, 'same'); % Convolve with G (vertical)
%% **Method 2: Direct 2D Filtering**
Y3 = conv2(I, H, 'same'); % Convolve with 2D filter H
%% **Display Results**
figure;
subplot(1,3,1); imshow(I, []); title('Original Image');
subplot(1,3,2); imshow(Y2, []); title('Separable Filtering (Y2)');
subplot(1,3,3); imshow(Y3, []); title('Direct 2D Filtering (Y3)');
%% **Compute Mean Squared Error (MSE)**
mse_value = mean((Y2(:) - Y3(:)).^2);
fprintf('Mean Squared Error (MSE) between Y2 and Y3: %f\n', mse_value);
%% **Compute Structural Similarity Index (SSIM)**
ssim_value = ssim(Y2, Y3);
fprintf('Structural Similarity Index (SSIM) between Y2 and Y3: %f\n',
ssim_value);
%% **Compute Mean Absolute Difference**
diff = abs(Y2 - Y3);
mean_diff = mean(diff(:));
fprintf('Mean Absolute Difference between Y2 and Y3: %f\n', mean_diff);
```

[Figures/Tables] - MATLAB:


```
Mean Squared Error (MSE) between Y2 and Y3: 0.000000
Structural Similarity Index (SSIM) between Y2 and Y3: 1.000000
Mean Absolute Difference between Y2 and Y3: 0.000000
fx >>
```

Figure 5: Results in MATLAB

[Code] - PYTHON

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import convolve2d
# Load the grayscale image
image_path =
"/media/tairo/Storages/CodeThayTien/XLA/ImgTest/OIP.jpeg"
gray_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
# Define the filters F, G, and H as given
F = np.array([[1, 2, 4, 2, 1]]) / 10
G = np.array([[1], [2], [4], [2], [1]]) / 10
H = np.array([
    [1, 2, 4, 2, 1],
    [2, 4, 8, 4, 2],
    [4, 8, 16, 8, 4],
    [2, 4, 8, 4, 2],
    [1, 2, 4, 2, 1]
]) / 100
```

METHOD 1

```
# Apply horizontal filtering on the grayscale image
Y_filteredF1 = convolve2d(gray_image, F, mode='same',
boundary='fill', fillvalue=0)
# Display the original and filtered images
plt.figure(figsize=(20, 10))

plt.subplot(1, 2, 1)
plt.imshow(gray_image, cmap='gray')
plt.title("Original Grayscale Image")
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(Y_filteredF1, cmap='gray')
plt.title("Filtered Image (Y = I * F)")
plt.axis('off')

plt.tight_layout()
plt.show()
#Apply vertical filtering on the grayscale image
Y_filteredG1 = convolve2d(gray_image, G, mode='same',
boundary='fill', fillvalue=0)
# Display the original and filtered images
plt.figure(figsize=(20, 10))
plt.subplot(1, 2, 1)
plt.imshow(gray_image, cmap='gray')
plt.title("Original Image")
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(Y_filteredG1, cmap='gray')
plt.title("Filtered Image (Vertical Filtering with G)")
plt.axis('off')

plt.tight_layout()
plt.show()

# Apply the filter H using 2D convolution
```

METHOD 2

```
Y_filteredH2 = convolve2d(gray_image, H, mode='same',
boundary='fill', fillvalue=0)


# Plot and display the original and filtered images
plt.figure(figsize=(20, 10))


plt.subplot(1, 2, 1)
plt.imshow(gray_image, cmap='gray')
plt.title("Original Image")
plt.axis('off')


plt.subplot(1, 2, 2)
plt.imshow(Y_filteredH2, cmap='gray')
plt.title("Filtered Image (H)")
plt.axis('off')


plt.suptitle("Filtering with H Kernel")
plt.tight_layout()
plt.show()


# Compute Mean Squared Error (MSE)
mse_value = np.mean((Y_filteredG1 - Y_filteredH2) ** 2)
print(f'Mean Squared Error (MSE) between Y2 and Y3:
{mse_value:.6f}')


# Compute Structural Similarity Index (SSIM)
ssim_value, _ = ssim(Y_filteredG1, Y_filteredH2, full=True,
data_range=Y_filteredH2.max() - Y_filteredH2.min())
print(f'Structural Similarity Index (SSIM) between Y2 and Y3:
{ssim_value:.6f}')


# Compute Mean Absolute Difference
mean_diff = np.mean(np.abs(Y_filteredG1 - Y_filteredH2))
print(f'Mean Absolute Difference between Y2 and Y3:
{mean_diff:.6f}')
```

[Figures/Tables] - PYTHON:

- *Method 1: apply Y1=I*F, Y2 = Y1*G*

*Figure 6: Original and Convolution Y1 = I*F*

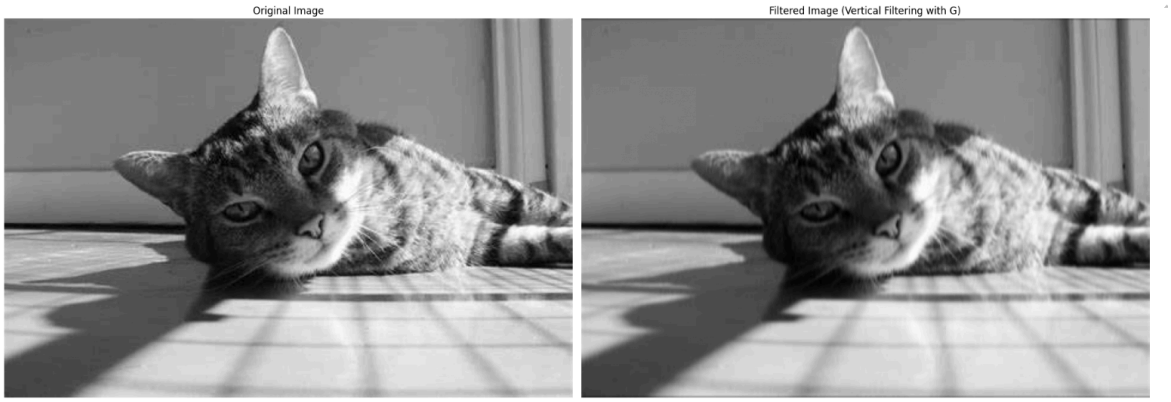

*Figure 7: Original and Convolution Y2 = I*G*

- *Method 2: Y3 = I*H*

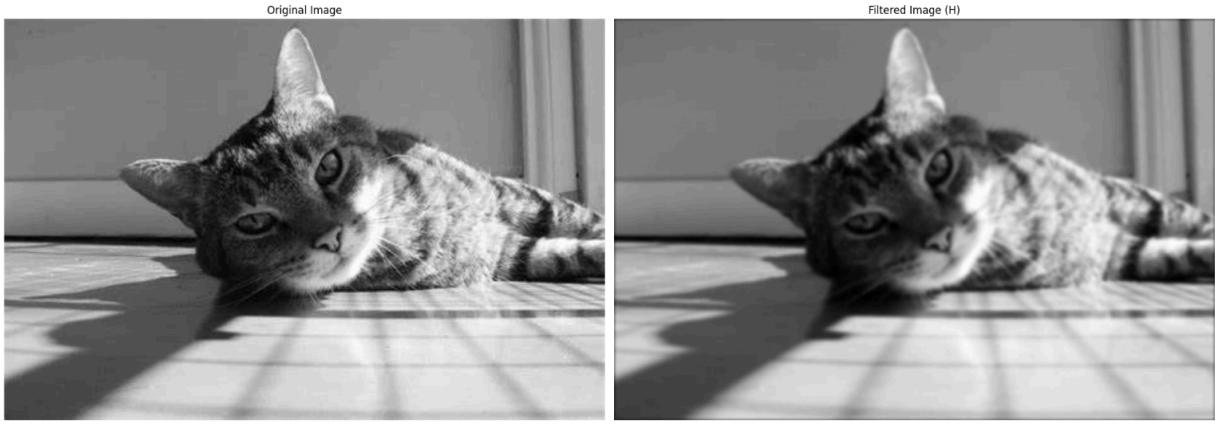

*Figure 8: Original and Convolution Y3 = I*H*

- *Comparison between Y2 and Y3 using MSE and SSIM*

```
Mean Squared Error (MSE) between Y2 and Y3: 19.063289

Structural Similarity Index (SSIM) between Y2 and Y3: 0.980653

Mean Absolute Difference between Y2 and Y3: 1.565895
```

[Explanation]

- *Observation:*
  - Y_2 and Y_3 produce very similar outputs.
  - The difference between them is very small, mostly due to numerical precision.
- *Explanation:*
  - The filter H is mathematically equivalent to the combination of F and G.
  - However, separable filtering (Method 1) is computationally more efficient because it reduces the number of operations compared to direct 2D convolution.
  - Method 1 applies two 1D convolutions instead of one 2D convolution, making it faster and requiring less memory.
  - Y2 and Y3 are **almost identical**, both visually and structurally, with only tiny pixel intensity variations that are unlikely to be perceptible to the human eye. If you were comparing filter performance, this indicates that the filters generating Y2 and Y3 behaved very similarly.
- Which methods should be used?
  - **Method 1 (Separable Filtering) is recommended** because:
    ➔ It is computationally **more efficient**.
    ➔ It produces **the same result** as Method 2.
    ➔ It requires **fewer operations**, reducing processing time.