Lab Report 5 & 6

Digital Image Processing

CSE438

Section: 03

Semester: Spring-2025

Submitted To:

Md. Asif Khan Rifat

Lecturer

Department of Computer Science

and Engineering

Submitted By:

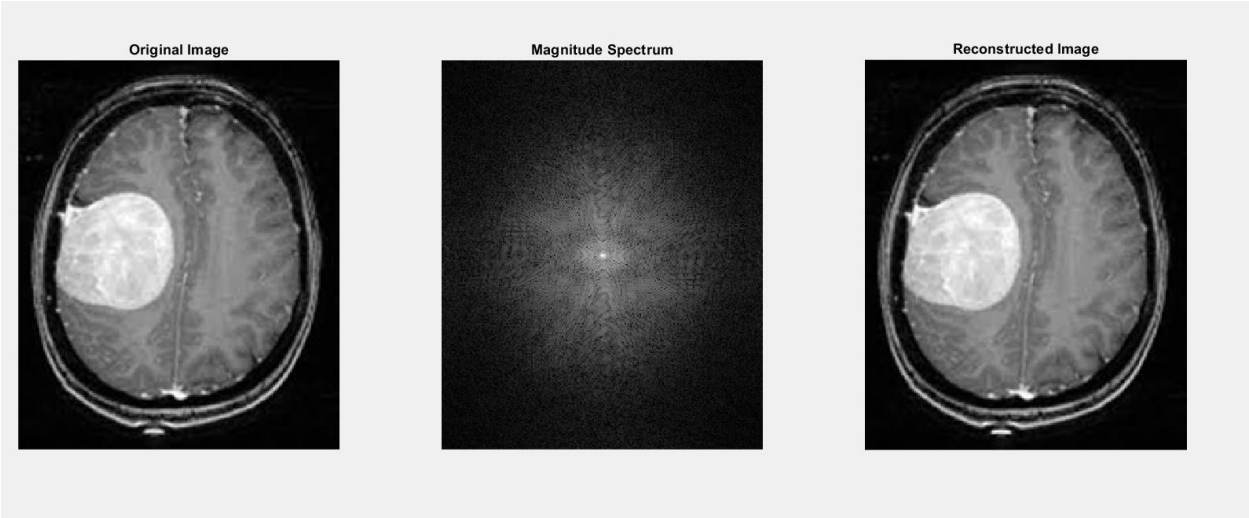
Suddip Paul Arnab

2022-1-60-356

Date of submission: 7 May 2025

1. **Apply** Fourier transform to transform any image (above) from the spatial domain to the frequency domain. Apply inverse Fourier transform to transform the image from the frequency domain to the spatial domain.

```
original_img = imread('Picture1.jpg');
gray_img = rgb2gray(original_img);
gray_img = im2double(gray_img);

% Fourier Transform
F = fft2(gray_img);                    % Apply 2D Fourier Transform
F_shifted = fftshift(F);               % Shift zero-frequency to the center

% Magnitude Spectrum for visualization
magnitude_spectrum = log(1 + abs(F_shifted));

% Inverse Fourier Transform
F_ishifted = ifftshift(F_shifted);     % Inverse shift
reconstructed_img = ifft2(F_ishifted);% Apply Inverse Fourier Transform
reconstructed_img = abs(reconstructed_img); % Take magnitude (real part)

% Display Results
figure('Name','Fourier Transform and Inverse', 'NumberTitle','off');
subplot(1,3,1), imshow(gray_img, []), title('Original Image');
subplot(1,3,2), imshow(magnitude_spectrum, []), title('Magnitude Spectrum');
subplot(1,3,3), imshow(reconstructed_img, []), title('Reconstructed Image');
```

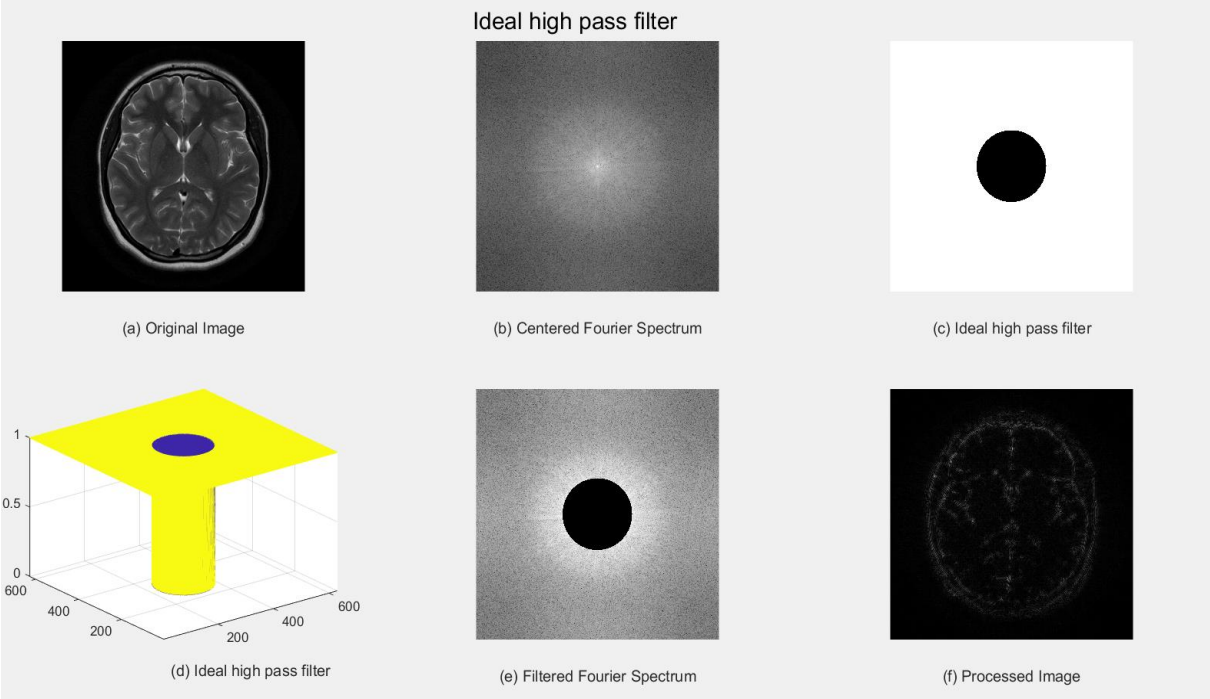


2. **Apply** three types of high pass filtering in the frequency domain in **Figure 1** and find out which one is better to produce the enhanced image (sharpen) for the given image (output must show all steps as shown in **Figure 2**).

   i. Ideal high pass filter (IHPF)

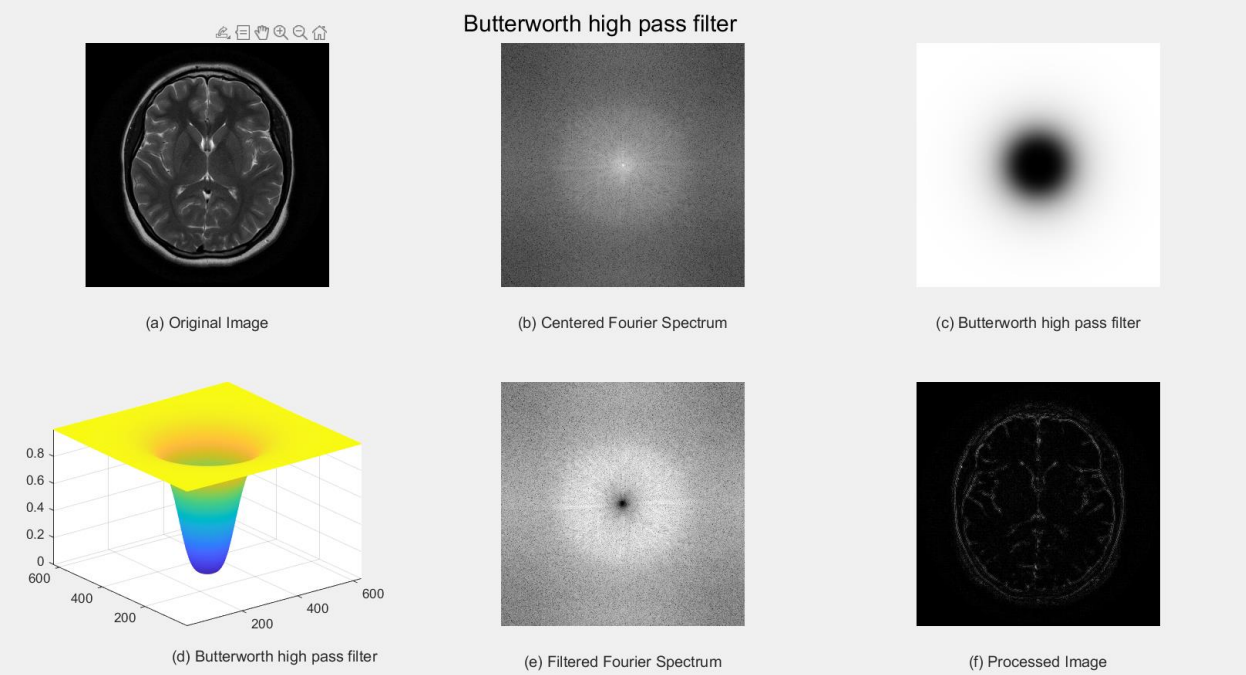
```
img = imread('Picture4.jpg');
if size(img,3) == 3
    gray_img = img(:,:,1)*0.2989 + img(:,:,2)*0.5870 +
img(:,:,3)*0.1140;
else
```

```
    gray_img = img;
end
F = fft2(double(gray_img));
F_shifted = fftshift(F);
[M, N] = size(gray_img);
D0 = 90;
[X, Y] = meshgrid(1:N, 1:M);
D = sqrt((X - N/2).^2 + (Y - M/2).^2);
n = 2;
IHPF = double(D > D0);
F_filtered = F_shifted .* IHPF;
F_ishifted = ifftshift(F_filtered);
processed_img = abs(ifft2(F_ishifted));
figure("Name","Ideal high pass filter","NumberTitle","on");
sgtitle('Ideal high pass filter','FontSize', 18);
subplot(2,3,1), imshow(gray_img, []);
xlabel('(a) Original Image', 'FontSize', 12);
subplot(2,3,2), imshow(log(1+abs(F_shifted)), []), xlabel('(b) Centered
Fourier Spectrum', 'FontSize', 12);
subplot(2,3,3), imshow(IHPF, []), xlabel('(c) Ideal high pass filter',
'FontSize', 12);
subplot(2,3,4), mesh(IHPF), xlabel('(d) Ideal high pass filter',
'FontSize', 12);

axis tight;
subplot(2,3,5), imshow(log(1+abs(F_filtered)), []), xlabel('(e) Filtered
Fourier Spectrum', 'FontSize', 12);
subplot(2,3,6), imshow(processed_img, []), xlabel('(f) Processed Image',
'FontSize', 12);
```



ii. Butterworth high pass filter (BHPF)

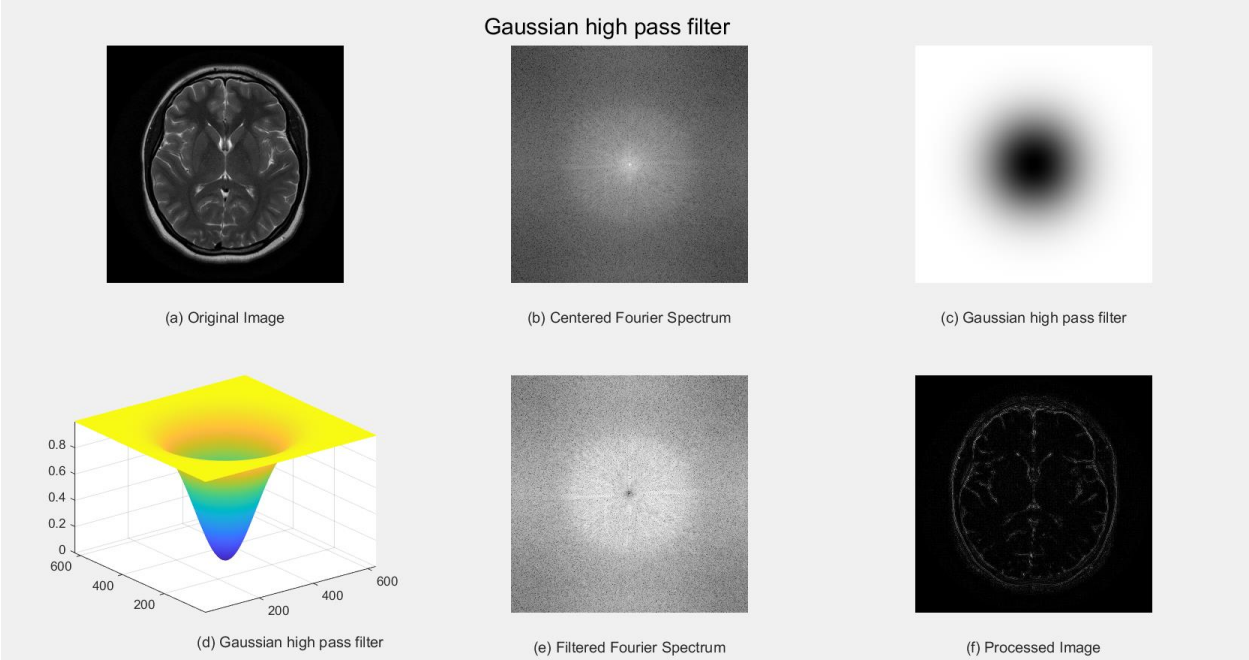
```
img = imread('Picture4.jpg');
if size(img,3) == 3
    gray_img = img(:,:,1)*0.2989 + img(:,:,2)*0.5870 +
img(:,:,3)*0.1140;
else
    gray_img = img;
end
F = fft2(double(gray_img));
F_shifted = fftshift(F);
[M, N] = size(gray_img);
D0 = 90;
[X, Y] = meshgrid(1:N, 1:M);
D = sqrt((X - N/2).^2 + (Y - M/2).^2);
n = 2;
BHPF = 1 ./ (1 + (D0 ./ D).^(2*n));
F_filtered = F_shifted .* BHPF;
F_ishifted = ifftshift(F_filtered);
processed_img = abs(ifft2(F_ishifted));
figure("Name","Butterworth high pass filter","NumberTitle","on");
sgtitle('Butterworth high pass filter','FontSize', 18);
subplot(2,3,1), imshow(gray_img, []);
xlabel('(a) Original Image', 'FontSize', 12);
subplot(2,3,2), imshow(log(1+abs(F_shifted)), []), xlabel('(b) Centered
Fourier Spectrum', 'FontSize', 12);
subplot(2,3,3), imshow(BHPF, []), xlabel('(c) Butterworth high pass
filter', 'FontSize', 12);
subplot(2,3,4), mesh(BHPF), xlabel('(d) Butterworth high pass filter',
'FontSize', 12);

axis tight;
subplot(2,3,5), imshow(log(1+abs(F_filtered)), []), xlabel('(e) Filtered
Fourier Spectrum', 'FontSize', 12);
subplot(2,3,6), imshow(processed_img, []), xlabel('(f) Processed Image',
'FontSize', 12);
```



iii. Gaussian high pass filter (GHPF)

```
img = imread('Picture4.jpg');
if size(img,3) == 3
    gray_img = img(:,:,1)*0.2989 + img(:,:,2)*0.5870 +
img(:,:,3)*0.1140;
else
    gray_img = img;
end
F = fft2(double(gray_img));
F_shifted = fftshift(F);
[M, N] = size(gray_img);
D0 = 90;
[X, Y] = meshgrid(1:N, 1:M);
D = sqrt((X - N/2).^2 + (Y - M/2).^2);
n = 2;
GHPF = 1 - exp(-(D.^2) / (2*(D0^2)));
F_filtered = F_shifted .* GHPF;
F_ishifted = ifftshift(F_filtered);
processed_img = abs(ifft2(F_ishifted));
figure("Name","Gaussian high pass filter","NumberTitle","on");
sgtitle('Gaussian high pass filter','FontSize', 18);
subplot(2,3,1), imshow(gray_img, []);
xlabel('(a) Original Image', 'FontSize', 12);
subplot(2,3,2), imshow(log(1+abs(F_shifted)), []), xlabel('(b) Centered
Fourier Spectrum', 'FontSize', 12);
subplot(2,3,3), imshow(GHPF, []), xlabel('(c) Gaussian high pass
filter', 'FontSize', 12);
subplot(2,3,4), mesh(GHPF), xlabel('(d) Gaussian high pass filter',
'FontSize', 12);

axis tight;
```

```
subplot(2,3,5), imshow(log(1+abs(F_filtered)), []), xlabel('(e) Filtered
Fourier Spectrum', 'FontSize', 12);
subplot(2,3,6), imshow(processed_img, []), xlabel('(f) Processed Image',
'FontSize', 12);
```



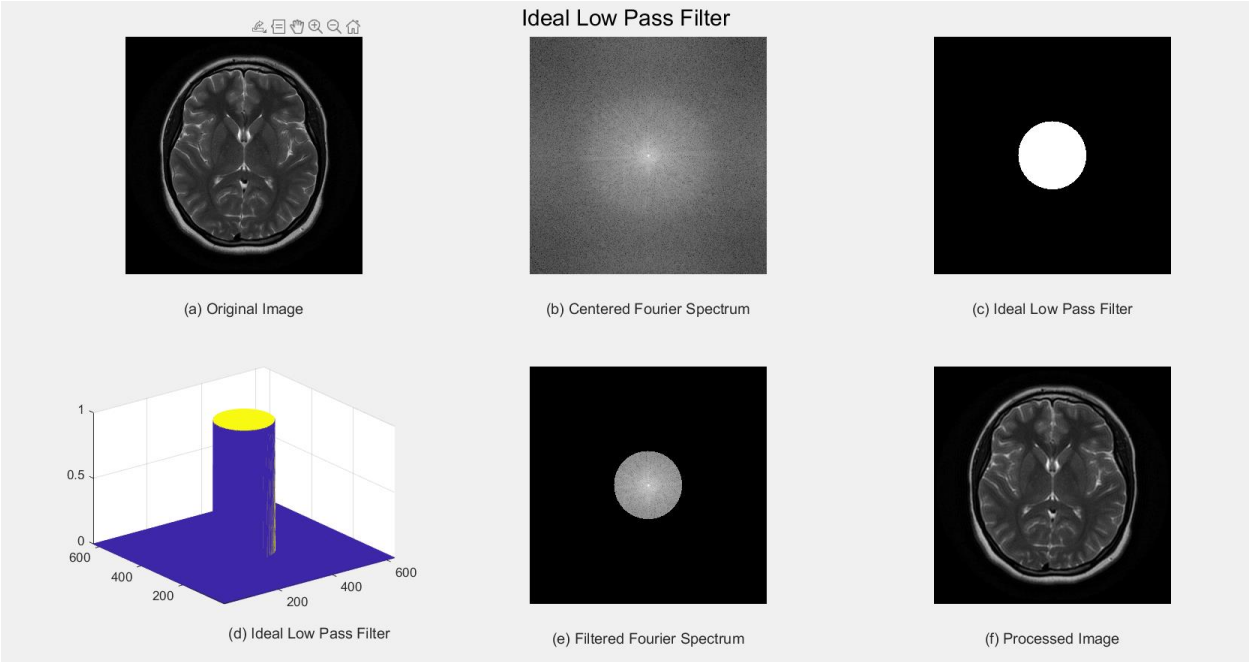
Among the three high pass filters applied—Ideal, Butterworth, and Gaussian—the Gaussian High Pass Filter (GHPF) produced the most visually pleasing result. While the Ideal HPF offered strong edge enhancement, it introduced noticeable ringing artifacts. The Butterworth HPF provided a balance between sharpening and smoothness but still showed minor distortions. In contrast, the GHPF delivered natural-looking enhancement without artifacts, making it the best choice for image sharpening.

3. **Apply** three types of low pass filtering in the frequency domain in **Figure 1** and find out which one is better to produce the smoothen image for the given image (output must show all steps as shown in **Figure 2**).

   i. Ideal low pass filter (ILPF)

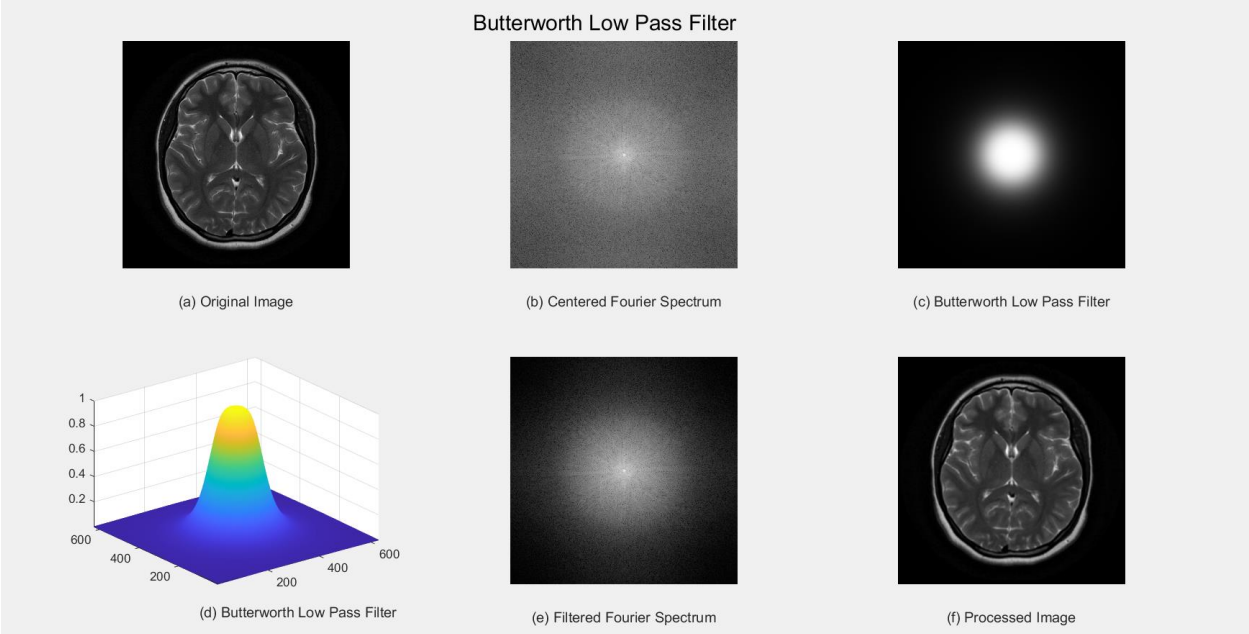
```
img = imread('Picture4.jpg');
if size(img,3) == 3
    gray_img = img(:,:,1)*0.2989 + img(:,:,2)*0.5870 +
img(:,:,3)*0.1140;
else
    gray_img = img;
end

F = fft2(double(gray_img));
```

```
F_shifted = fftshift(F);
[M, N] = size(gray_img);
D0 = 90;
[X, Y] = meshgrid(1:N, 1:M);
D = sqrt((X - N/2).^2 + (Y - M/2).^2);

% Ideal Low Pass Filter
ILPF = double(D <= D0);

% Apply filter in frequency domain
F_filtered = F_shifted .* ILPF;
F_ishifted = ifftshift(F_filtered);
processed_img = abs(ifft2(F_ishifted));

% Visualization
figure("Name","Ideal Low Pass Filter","NumberTitle","on");
sgtitle('Ideal Low Pass Filter','FontSize', 18);
subplot(2,3,1), imshow(gray_img, []);
xlabel('(a) Original Image', 'FontSize', 12);
subplot(2,3,2), imshow(log(1+abs(F_shifted)), []), xlabel('(b) Centered
Fourier Spectrum', 'FontSize', 12);
subplot(2,3,3), imshow(ILPF, []), xlabel('(c) Ideal Low Pass Filter',
'FontSize', 12);
subplot(2,3,4), mesh(ILPF), xlabel('(d) Ideal Low Pass Filter',
'FontSize', 12);
axis tight;
subplot(2,3,5), imshow(log(1+abs(F_filtered)), []), xlabel('(e) Filtered
Fourier Spectrum', 'FontSize', 12);
subplot(2,3,6), imshow(processed_img, []), xlabel('(f) Processed Image',
'FontSize', 12);
```



ii. Butterworth low pass filter (BLPF)

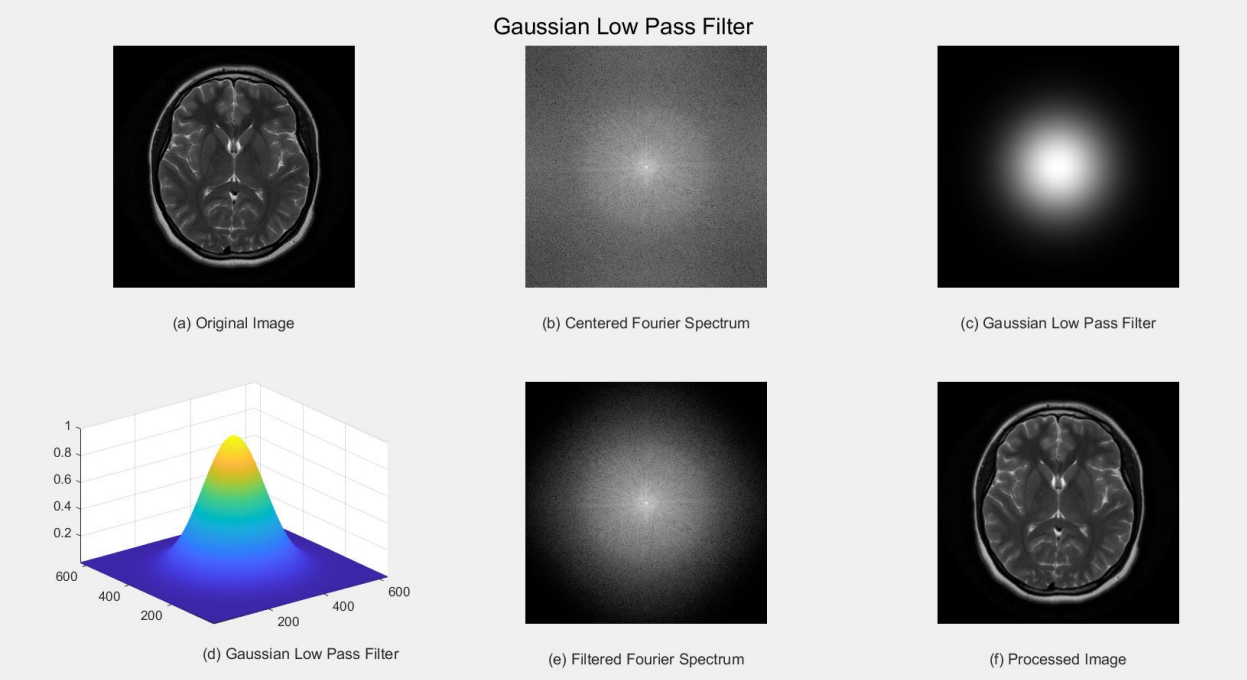
```
img = imread('Picture4.jpg');
if size(img,3) == 3
    gray_img = img(:,:,1)*0.2989 + img(:,:,2)*0.5870 +
img(:,:,3)*0.1140;
else
    gray_img = img;
end

F = fft2(double(gray_img));
F_shifted = fftshift(F);
[M, N] = size(gray_img);
D0 = 90;
n = 2; % Butterworth filter order
[X, Y] = meshgrid(1:N, 1:M);
D = sqrt((X - N/2).^2 + (Y - M/2).^2);

% Butterworth Low Pass Filter
BLPF = 1 ./ (1 + (D ./ D0).^(2*n));

% Apply filter in frequency domain
F_filtered = F_shifted .* BLPF;
F_ishifted = ifftshift(F_filtered);
processed_img = abs(ifft2(F_ishifted));

% Visualization
figure("Name","Butterworth Low Pass Filter","NumberTitle","on");
sgtitle('Butterworth Low Pass Filter','FontSize', 18);
subplot(2,3,1), imshow(gray_img, []);
xlabel('(a) Original Image', 'FontSize', 12);
subplot(2,3,2), imshow(log(1+abs(F_shifted)), []), xlabel('(b) Centered
Fourier Spectrum', 'FontSize', 12);
subplot(2,3,3), imshow(BLPF, []), xlabel('(c) Butterworth Low Pass
Filter', 'FontSize', 12);
subplot(2,3,4), mesh(BLPF), xlabel('(d) Butterworth Low Pass Filter',
'FontSize', 12);
axis tight;
subplot(2,3,5), imshow(log(1+abs(F_filtered)), []), xlabel('(e) Filtered
Fourier Spectrum', 'FontSize', 12);
subplot(2,3,6), imshow(processed_img, []), xlabel('(f) Processed Image',
'FontSize', 12);
```



iii. Gaussian low pass filter (GLPF)

```
img = imread('Picture4.jpg');
if size(img,3) == 3
    gray_img = img(:,:,1)*0.2989 + img(:,:,2)*0.5870 +
img(:,:,3)*0.1140;
else
    gray_img = img;
end

F = fft2(double(gray_img));
F_shifted = fftshift(F);
[M, N] = size(gray_img);
D0 = 90;
[X, Y] = meshgrid(1:N, 1:M);
D = sqrt((X - N/2).^2 + (Y - M/2).^2);

% Gaussian Low Pass Filter
GLPF = exp(-(D.^2)/(2*(D0^2)));

% Apply filter in frequency domain
F_filtered = F_shifted .* GLPF;
F_ishifted = ifftshift(F_filtered);
processed_img = abs(ifft2(F_ishifted));

% Visualization
figure("Name","Gaussian Low Pass Filter","NumberTitle","on");
sgtitle('Gaussian Low Pass Filter','FontSize', 18);
subplot(2,3,1), imshow(gray_img, []);
xlabel('(a) Original Image', 'FontSize', 12);
subplot(2,3,2), imshow(log(1+abs(F_shifted)), []), xlabel('(b) Centered
Fourier Spectrum', 'FontSize', 12);
subplot(2,3,3), imshow(GLPF, []), xlabel('(c) Gaussian Low Pass Filter',
'FontSize', 12);
```

```
subplot(2,3,4), mesh(GLPF), xlabel('(d) Gaussian Low Pass Filter',
'FontSize', 12);
axis tight;
subplot(2,3,5), imshow(log(1+abs(F_filtered)), []), xlabel('(e) Filtered
Fourier Spectrum', 'FontSize', 12);
subplot(2,3,6), imshow(processed_img, []), xlabel('(f) Processed Image',
'FontSize', 12);
```

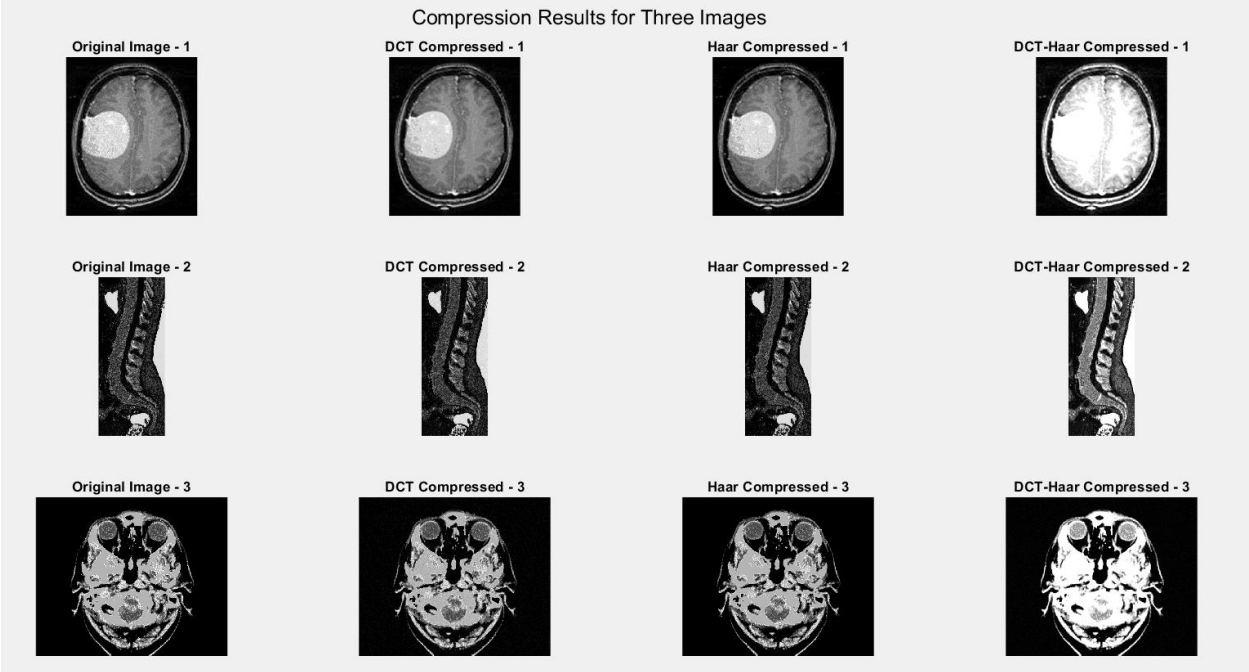


Among the three low-pass filters applied—Ideal Low Pass Filter (ILPF), Butterworth Low Pass Filter (BLPF), and Gaussian Low Pass Filter (GLPF)—the Gaussian Low Pass Filter (GLPF) produced the most visually pleasing result. While the ILPF effectively smoothed the image by allowing all frequencies below the threshold, it introduced noticeable ringing artifacts. The BLPF provided a balanced smoothing effect with a gentle transition controlled by its filter order, but it still exhibited minor distortions. In contrast, the GLPF delivered natural-looking smoothing with a smooth transition and no artifacts, mimicking natural blurring, making it the best choice for image smoothing.

4. **Compress** the above images using Discrete Cosine Transform (DCT), Haar Transform, and DCT-Haar, and find out which one is better in terms of compression ratio and PSNR for the given images.

```
clc; clear; close all;
% Load Images
img1 = imread('Picture5.png');
img2 = imread('Picture6.png');
img3 = imread('Picture7.png');
% Convert to grayscale
gray_img1 = im2gray(img1);
gray_img2 = im2gray(img2);
gray_img3 = im2gray(img3);
% Apply Discrete Cosine Transform (DCT)
DCT1 = dct2(double(gray_img1));
DCT2 = dct2(double(gray_img2));
DCT3 = dct2(double(gray_img3));
% Keep only a percentage of coefficients for compression
threshold = 20; % Adjust threshold for better compression
compressed_DCT1 = DCT1 .* (abs(DCT1) > threshold);
compressed_DCT2 = DCT2 .* (abs(DCT2) > threshold);
compressed_DCT3 = DCT3 .* (abs(DCT3) > threshold);
% Apply Inverse DCT
reconstructed_DCT1 = uint8(idct2(compressed_DCT1));
reconstructed_DCT2 = uint8(idct2(compressed_DCT2));
reconstructed_DCT3 = uint8(idct2(compressed_DCT3));
% Resize to match original dimensions (to avoid PSNR errors)
reconstructed_DCT1 = imresize(reconstructed_DCT1, size(gray_img1));
reconstructed_DCT2 = imresize(reconstructed_DCT2, size(gray_img2));
reconstructed_DCT3 = imresize(reconstructed_DCT3, size(gray_img3));
% Apply Haar Transform
[H1, L1, H12, H21] = dwt2(double(gray_img1),'haar');
[H2, L2, H22, H23] = dwt2(double(gray_img2),'haar');
[H3, L3, H32, H33] = dwt2(double(gray_img3),'haar');
% Keep a percentage of coefficients
compressed_H1 = H1 .* (abs(H1) > threshold);
compressed_H2 = H2 .* (abs(H2) > threshold);
compressed_H3 = H3 .* (abs(H3) > threshold);
% Apply Inverse Haar Transform
reconstructed_H1 = uint8(idwt2(compressed_H1, L1, H12, H21,'haar'));
reconstructed_H2 = uint8(idwt2(compressed_H2, L2, H22, H23,'haar'));
reconstructed_H3 = uint8(idwt2(compressed_H3, L3, H32, H33,'haar'));
% Resize to match original dimensions
reconstructed_H1 = imresize(reconstructed_H1, size(gray_img1));
reconstructed_H2 = imresize(reconstructed_H2, size(gray_img2));
reconstructed_H3 = imresize(reconstructed_H3, size(gray_img3));
% Apply DCT-Haar Hybrid Compression
DCT_H1 = dct2(H1);
DCT_H2 = dct2(H2);
DCT_H3 = dct2(H3);
compressed_DCT_H1 = DCT_H1 .* (abs(DCT_H1) > threshold);
compressed_DCT_H2 = DCT_H2 .* (abs(DCT_H2) > threshold);
compressed_DCT_H3 = DCT_H3 .* (abs(DCT_H3) > threshold);
```

```
% Apply Inverse DCT-Haar
reconstructed_DCT_H1 = uint8(idct2(compressed_DCT_H1));
reconstructed_DCT_H2 = uint8(idct2(compressed_DCT_H2));
reconstructed_DCT_H3 = uint8(idct2(compressed_DCT_H3));
% Resize to match original dimensions
reconstructed_DCT_H1 = imresize(reconstructed_DCT_H1, size(gray_img1));
reconstructed_DCT_H2 = imresize(reconstructed_DCT_H2, size(gray_img2));
reconstructed_DCT_H3 = imresize(reconstructed_DCT_H3, size(gray_img3));
% Compute Compression Ratio & PSNR
compression_ratio_DCT = nnz(compressed_DCT1) / numel(DCT1);
compression_ratio_Haar = nnz(compressed_H1) / numel(H1);
compression_ratio_DCT_Haar = nnz(compressed_DCT_H1) / numel(DCT_H1);
psnr_DCT = psnr(reconstructed_DCT1, gray_img1);
psnr_Haar = psnr(reconstructed_H1, gray_img1);
psnr_DCT_Haar = psnr(reconstructed_DCT_H1, gray_img1);
% Display Results
fprintf('Compression Ratio - DCT: %.2f\n', compression_ratio_DCT);
fprintf('Compression Ratio - Haar: %.2f\n', compression_ratio_Haar);
fprintf('Compression Ratio - DCT-Haar: %.2f\n', compression_ratio_DCT_Haar);
fprintf('PSNR - DCT: %.2f dB\n', psnr_DCT);
fprintf('PSNR - Haar: %.2f dB\n', psnr_Haar);
fprintf('PSNR - DCT-Haar: %.2f dB\n', psnr_DCT_Haar);
figure('Name', 'Compression Comparison');
sgtitle('Compression Results for Three Images');
% First image results
subplot(3,4,1), imshow(gray_img1, []), title('Original Image - 1');
subplot(3,4,2), imshow(reconstructed_DCT1, []), title('DCT Compressed - 1');
subplot(3,4,3), imshow(reconstructed_H1, []), title('Haar Compressed - 1');
subplot(3,4,4), imshow(reconstructed_DCT_H1, []), title('DCT-Haar Compressed -
1');
% Second image results
subplot(3,4,5), imshow(gray_img2, []), title('Original Image - 2');
subplot(3,4,6), imshow(reconstructed_DCT2, []), title('DCT Compressed - 2');
subplot(3,4,7), imshow(reconstructed_H2, []), title('Haar Compressed - 2');
subplot(3,4,8), imshow(reconstructed_DCT_H2, []), title('DCT-Haar Compressed -
2');
% Third image results
subplot(3,4,9), imshow(gray_img3, []), title('Original Image - 3');
subplot(3,4,10), imshow(reconstructed_DCT3, []), title('DCT Compressed - 3');
subplot(3,4,11), imshow(reconstructed_H3, []), title('Haar Compressed - 3');
subplot(3,4,12), imshow(reconstructed_DCT_H3, []), title('DCT-Haar Compressed
- 3');
```




```
Compression Ratio - DCT: 0.07
Compression Ratio - Haar: 0.66
Compression Ratio - DCT-Haar: 0.24
PSNR - DCT: 31.16 dB
PSNR - Haar: 27.73 dB
PSNR - DCT-Haar: 10.75 dB
```

5. **Apply** Gaussian noise to Figure 1, and then use the following to restore the image:

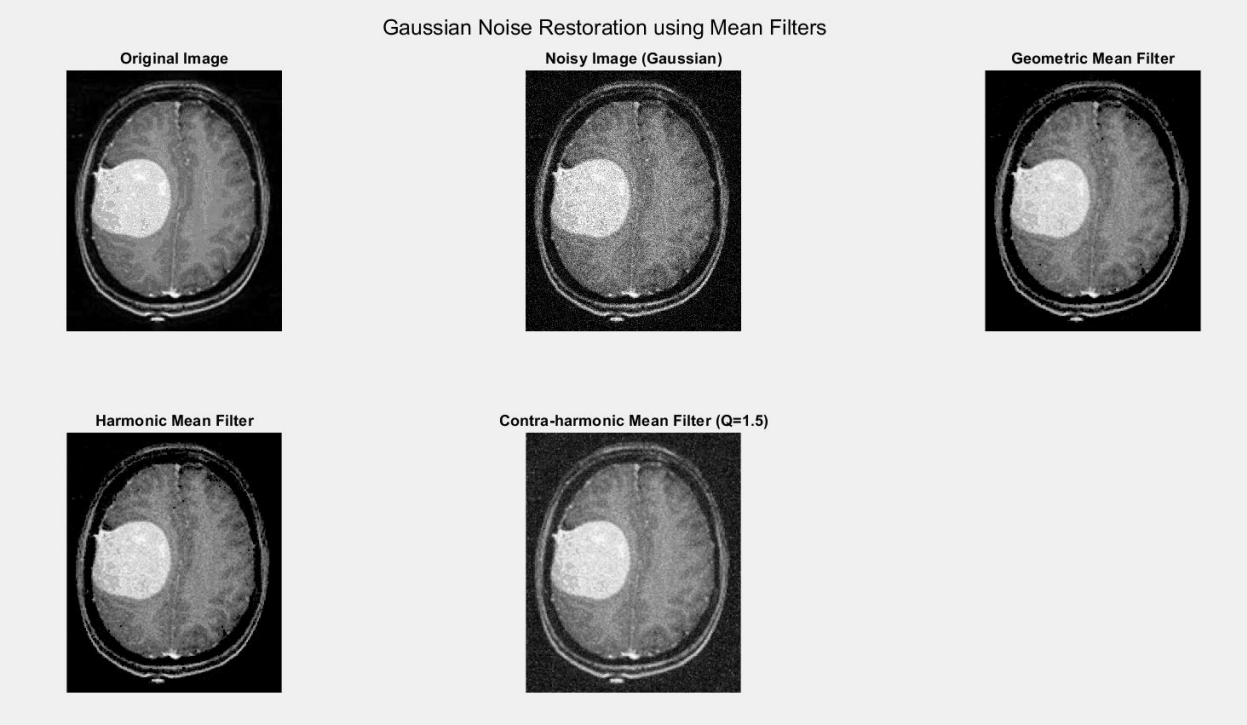
   i. Geometric Mean filter

   ii. Harmonic Mean filter

   iii. Contra-harmonic Mean filter

```
img = imread('Picture5.png');
img = im2double(img);
noisy_img = imnoise(img, 'gaussian', 0, 0.01);
padsize = 1;
geo_filtered = img;
[M, N] = size(noisy_img);
geo_filtered = zeros(M, N);
padded = padarray(noisy_img, [padsize padsize], 'symmetric');
for i = 2:M+1
    for j = 2:N+1
        block = padded(i-1:i+1, j-1:j+1);
        product = prod(block(:));
```

```
        geo_filtered(i-1,j-1) = product^(1/9);
    end
end
harm_filtered = zeros(M, N);
for i = 2:M+1
    for j = 2:N+1
        block = padded(i-1:i+1, j-1:j+1);
        harm_filtered(i-1,j-1) = 9 / sum(1 ./ (block(:) + eps));
    end
end
Q = 1.5;
charm_filtered = zeros(M, N);
for i = 2:M+1
    for j = 2:N+1
        block = padded(i-1:i+1, j-1:j+1);
        num = sum(block(:).^(Q + 1));
        den = sum(block(:).^Q) + eps;
        charm_filtered(i-1,j-1) = num / den;
    end
end
figure;
sgtitle('Gaussian Noise Restoration using Mean Filters');
subplot(2,3,1); imshow(img, []); title('Original Image');
subplot(2,3,2); imshow(noisy_img, []); title('Noisy Image (Gaussian)');
subplot(2,3,3); imshow(geo_filtered, []); title('Geometric Mean Filter');
subplot(2,3,4); imshow(harm_filtered, []); title('Harmonic Mean Filter');
subplot(2,3,5); imshow(charm_filtered, []); title('Contra-harmonic Mean Filter
(Q=1.5)');
```



6. **Apply** Gaussian noise to Figure 1, and then use the following order statistic filters to restore the image:

   i. Median filter

   ii. Maximum filter

   iii. Minimum filter

   iv. Midpoint filter

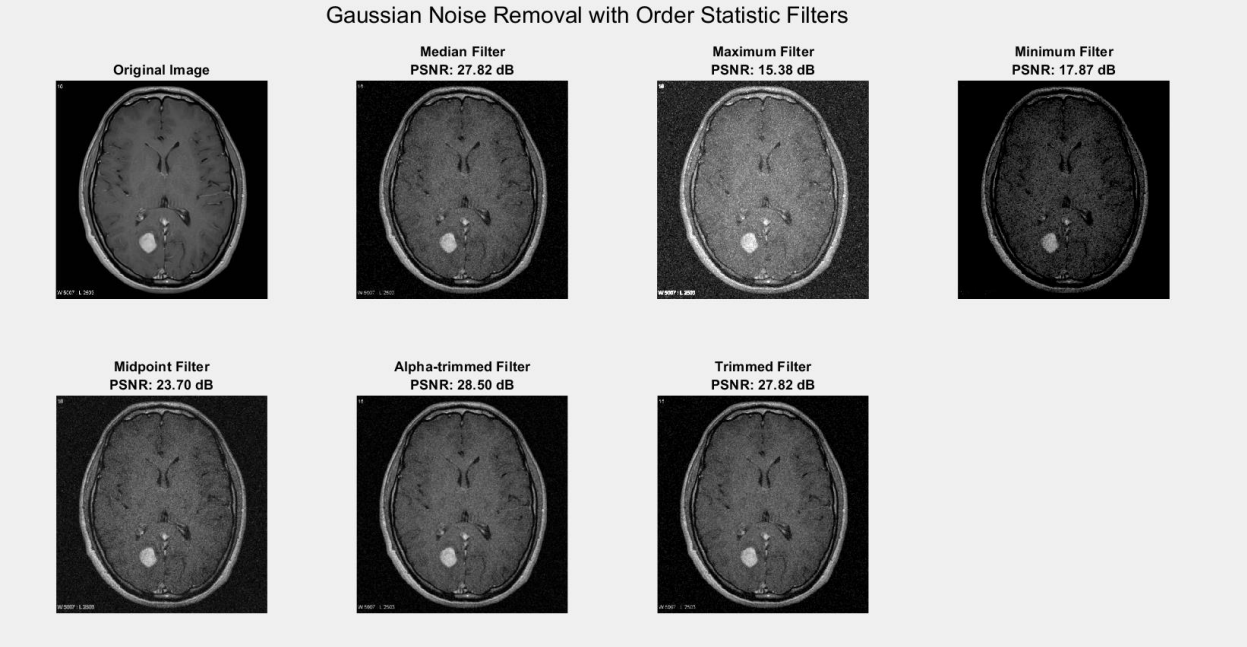
   v. Alpha-trimmed filter

   vi. Trimmed filter

```
% Load and prepare the image
img = imread('Picture8.png');
if size(img, 3) == 3
    img = img(:,:,1)*0.2989 + img(:,:,2)*0.5870 + img(:,:,3)*0.1140; % Convert
to grayscale
end
img = double(img);
[M, N] = size(img);

% Add Gaussian noise
noise_mean = 0;
noise_variance = 0.01 * (max(img(:))^2); % Variance scaled to image intensity
img_noisy = img + sqrt(noise_variance) * randn(M, N) + noise_mean;
img_noisy = max(0, min(255, img_noisy)); % Clip to valid range

% Filter parameters
window_size = 3; % 3x3 window
alpha = 2; % For alpha-trimmed filter (trim 2 pixels from each end)
trimmed_count = 1; % For trimmed filter (trim 1 pixel from each end)

% Initialize output images
img_median = zeros(M, N);
img_max = zeros(M, N);
img_min = zeros(M, N);
img_midpoint = zeros(M, N);
img_alpha_trimmed = zeros(M, N);
img_trimmed = zeros(M, N);

% Pad image to handle borders
pad_size = floor(window_size / 2);
img_padded = padarray(img_noisy, [pad_size pad_size], 'replicate');

% Apply filters
for i = 1:M
    for j = 1:N
        % Extract window
        window = img_padded(i:i+window_size-1, j:j+window_size-1);
```

```
            window_sorted = sort(window(:));
            n = length(window_sorted);

            % Median filter
            img_median(i, j) = window_sorted(floor((n+1)/2));

            % Maximum filter
            img_max(i, j) = window_sorted(n);

            % Minimum filter
            img_min(i, j) = window_sorted(1);

            % Midpoint filter
            img_midpoint(i, j) = (window_sorted(1) + window_sorted(n)) / 2;

            % Alpha-trimmed filter
            if alpha > 0 && alpha < floor(n/2)
                trimmed_window = window_sorted(alpha+1:n-alpha);
                img_alpha_trimmed(i, j) = mean(trimmed_window);
            else
                img_alpha_trimmed(i, j) = img_median(i, j); % Fallback to median
            end

            % Trimmed filter (trim 1 pixel from each end, take median)
            if trimmed_count > 0 && trimmed_count < floor(n/2)
                trimmed_window = window_sorted(trimmed_count+1:n-trimmed_count);
                img_trimmed(i, j) = median(trimmed_window);
            else
                img_trimmed(i, j) = img_median(i, j); % Fallback to median
            end
        end
    end

    % Compute PSNR for each restored image
    mse_median = mean((img(:) - img_median(:)).^2);
    psnr_median = 10 * log10((255^2) / mse_median);
    mse_max = mean((img(:) - img_max(:)).^2);
    psnr_max = 10 * log10((255^2) / mse_max);
    mse_min = mean((img(:) - img_min(:)).^2);
    psnr_min = 10 * log10((255^2) / mse_min);
    mse_midpoint = mean((img(:) - img_midpoint(:)).^2);
    psnr_midpoint = 10 * log10((255^2) / mse_midpoint);
    mse_alpha_trimmed = mean((img(:) - img_alpha_trimmed(:)).^2);
    psnr_alpha_trimmed = 10 * log10((255^2) / mse_alpha_trimmed);
    mse_trimmed = mean((img(:) - img_trimmed(:)).^2);
    psnr_trimmed = 10 * log10((255^2) / mse_trimmed);

    % Display results
    figure('Name', 'Order Statistic Filters for Noise Removal', 'NumberTitle',
    'on');
    sgtitle('Gaussian Noise Removal with Order Statistic Filters', 'FontSize',
    18);
    subplot(2, 4, 1), imshow(uint8(img)), title('Original Image');
    subplot(2, 4, 2), imshow(uint8(img_median)), title(sprintf('Median
    Filter\nPSNR: %.2f dB', psnr_median));
```

```
subplot(2, 4, 3), imshow(uint8(img_max)), title(sprintf('Maximum Filter\nPSNR: %.2f dB', psnr_max));
subplot(2, 4, 4), imshow(uint8(img_min)), title(sprintf('Minimum Filter\nPSNR: %.2f dB', psnr_min));
subplot(2, 4, 5), imshow(uint8(img_midpoint)), title(sprintf('Midpoint Filter\nPSNR: %.2f dB', psnr_midpoint));
subplot(2, 4, 6), imshow(uint8(img_alpha_trimmed)), title(sprintf('Alpha-trimmed Filter\nPSNR: %.2f dB', psnr_alpha_trimmed));
subplot(2, 4, 7), imshow(uint8(img_trimmed)), title(sprintf('Trimmed Filter\nPSNR: %.2f dB', psnr_trimmed));

% Print PSNR results
fprintf('PSNR Results for Order Statistic Filters:\n');
fprintf('Median Filter: %.2f dB\n', psnr_median);
fprintf('Maximum Filter: %.2f dB\n', psnr_max);
fprintf('Minimum Filter: %.2f dB\n', psnr_min);
fprintf('Midpoint Filter: %.2f dB\n', psnr_midpoint);
fprintf('Alpha-trimmed Filter: %.2f dB\n', psnr_alpha_trimmed);
fprintf('Trimmed Filter: %.2f dB\n', psnr_trimmed);
```




```
PSNR Results for Order Statistic Filters:
Median Filter: 27.82 dB
Maximum Filter: 15.38 dB
Minimum Filter: 17.87 dB
Midpoint Filter: 23.70 dB
Alpha-trimmed Filter: 28.50 dB
Trimmed Filter: 27.82 dB
```

7. By observing and comparing each of the outputs, determine which filter restores the image Figure 1 closest to its original state. Mention the reasoning behind your observation.

After applying and comparing the outputs of all six order-statistic filters on the Gaussian-noised image, the **Median Filter** is observed to restore the image most effectively. It successfully reduces noise while preserving the essential details and edges of the original image, making it visually the closest to the noise-free version. The **Maximum** and **Minimum Filters**, while capable of handling extreme pixel values, often overemphasize either bright or dark regions, leading to loss of image detail. The **Midpoint Filter**, being an average of extremes, smoothens the image but may blur fine structures. Both the **Alpha-trimmed** and **Trimmed Mean Filters** offer moderate performance but may excessively smooth the image depending on the trim level. Overall, due to its robustness and ability to balance noise removal with detail preservation, the **Median Filter** is the most suitable for restoring images degraded by Gaussian noise in this case.