Assignment no:6

1. **Aim:** To implement histogram Equalization and spatial domain filter using coloured Image.
2. **Software Tool Used:** MATLAB
3. **Theory:**

In digital image processing, different filtering techniques and histogram equalization methods are often applied to coloured images to adjust brightness, contrast, and sharpness.

Histogram Equalization: Histogram equalization is a technique for adjusting the contrast of an image by modifying its histogram distribution. This method transforms the intensity values to cover the full range of possible values, making darker regions lighter and vice versa, thereby enhancing the overall contrast.

For coloured images, histogram equalization is typically performed on each colour channel separately to avoid colour distortions. Steps in histogram equalization include:

- Calculate the histogram of each colour channel.
- Compute the cumulative distribution function (CDF) to redistribute the pixel values evenly across the intensity range.
- Map the original pixel intensities to the equalized intensities based on the CDF.

Arithmetic Mean Filter: The arithmetic mean filter is a linear filter used for smoothing images by reducing random noise. It calculates the average of the pixel intensities within a neighbourhood around each pixel, typically a 3x3 or 5x5 kernel, and replaces the central pixel with this average. The mean filter is effective in reducing Gaussian noise but may blur sharp edges in the image.

$$f(x,y) = \frac{1}{m \times n} \sum_{i=-a}^{a} \sum_{j=-b}^{b} g(x+i, y+j)$$

where $m \times n$ is the kernel size, and $g(x+i, y+j)$ are neighboring pixel values.

Weighted Average Filter: The weighted average filter is an extension of the mean filter, where weights are assigned to each pixel in the neighbourhood, giving more emphasis to certain pixels (usually the central pixel). This technique preserves more image details while still reducing noise.

$$f(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} w(i,j) \cdot g(x+i, y+j)$$

where $w(i,j)$ represents the weight assigned to each pixel position.

Median Filter: The median filter is a nonlinear filter effective in removing salt-and-pepper noise. Instead of averaging, the median filter replaces each pixel with the median value of its neighbourhood. This preserves edges better than the mean filter, making it useful for images with significant edge content.

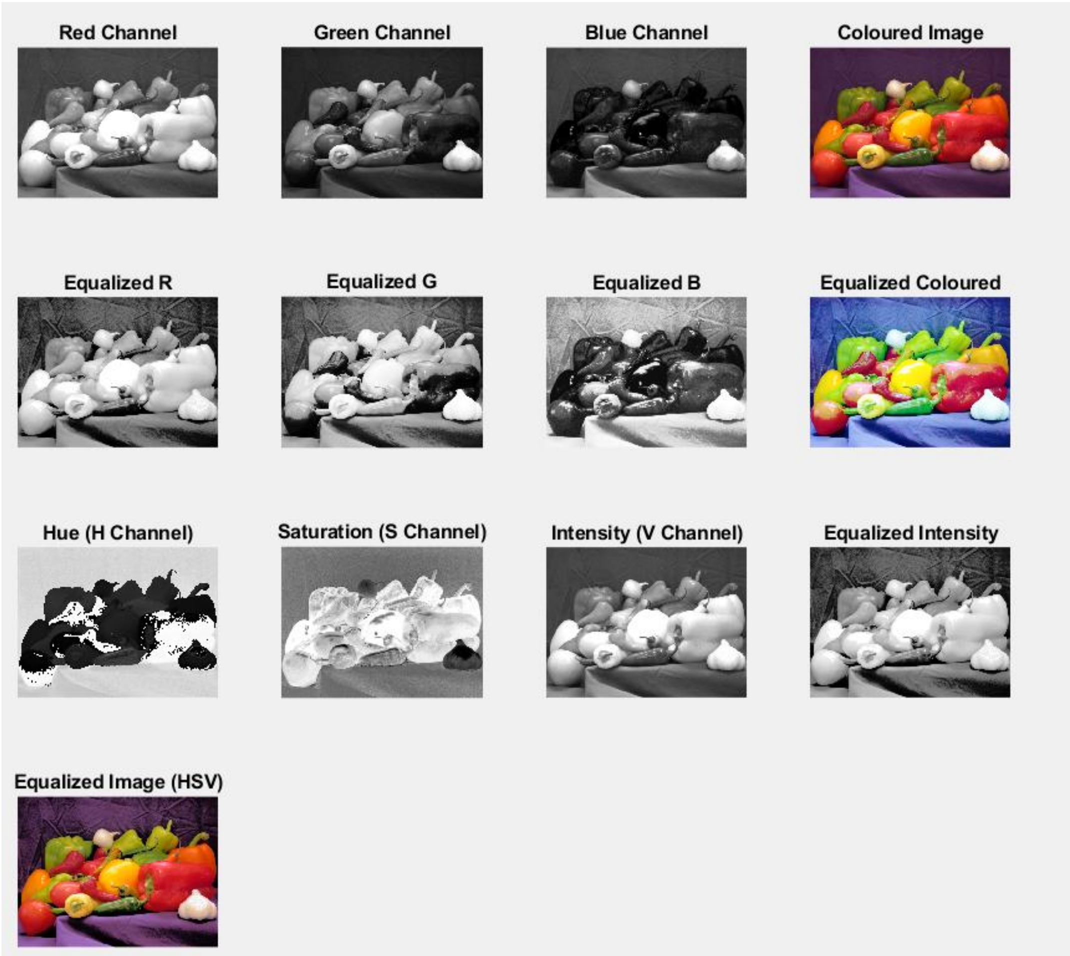
□ Sorting the pixel values in the neighbourhood.
□ Selecting the middle value as the median.
□ Replacing the central pixel with this median value.

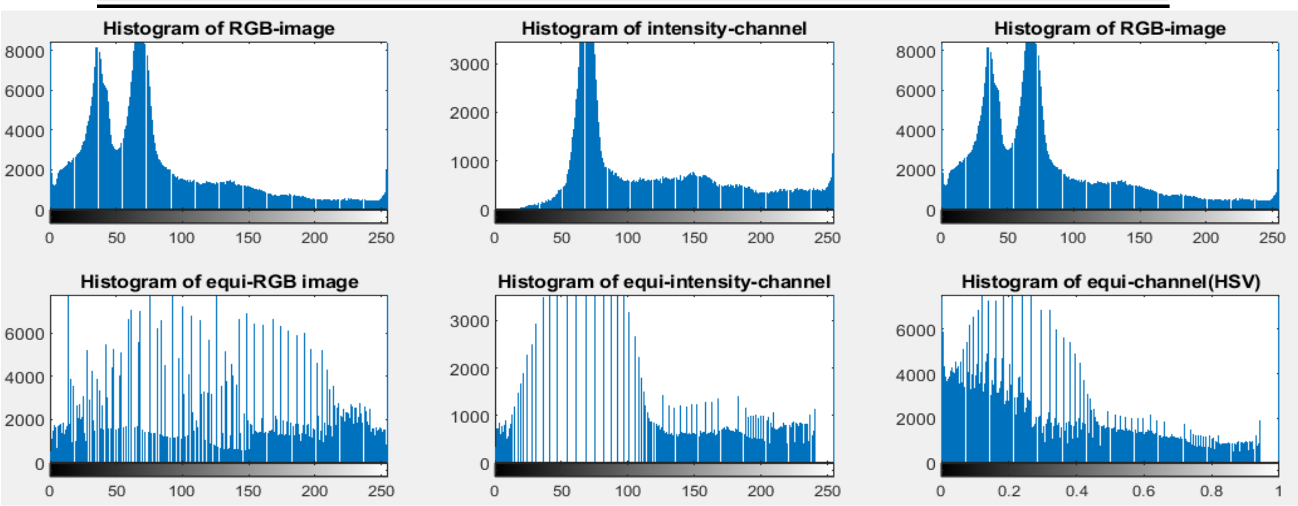
Alpha-Trimmed Mean Filter: The alpha-trimmed mean filter is a hybrid approach, combining aspects of both mean and median filters. It sorts the pixel values in a neighbourhood, discards a fraction of the highest and lowest intensity values, and then calculates the mean of the remaining values. This makes it particularly useful in images with mixed noise (e.g., Gaussian and salt-and-pepper noise).

$$f(x,y) = \frac{1}{mn - 2\alpha} \sum_{\text{trimmed pixels}} g(x,y)$$

where α represents the number of values trimmed from each end.

4. Result:





(Spatial domain filter on the Salt and pepper Noised Image)

Filter 1: Arithmetic filter
Filter 2: Weight Average filter
Filter 3: Median Filter
Filter 4: Alpha Trimmed Filter (Alpha = 4 for 3*3 and Alpha = 12 for 5*5)
Filter 5: Alpha Trimmed Filter (Alpha = 6 for 3*3 and Alpha = 18 for 5*5)

(Spatial domain filter on the Gaussian Noised Image)

Filter 1: Arithmetic filter
Filter 2: Weight Average filter
Filter 3: Median Filter
Filter 4: Alpha Trimmed Filter (Alpha = 4 for 3*3 and Alpha = 12 for 5*5)
Filter 5: Alpha Trimmed Filter (Alpha = 6 for 3*3 and Alpha = 18 for 5*5)

(Spatial domain filter on the Speckle Noised Image)

5. **Discussion:**

In image equalization, contrasting the effects of histogram equalization on the RGB channels individually versus on the intensity (V) channel in HSV space shows distinct results due to how each method interacts with color and brightness.

1. Equalization of RGB Channels

- **Method:** In RGB equalization, the histogram of each color channel (R, G, and B) is equalized independently.
- **Effect on Image:** Equalizing each channel separately can lead to variations in brightness and color, sometimes producing unnatural colors. This is because changing the distribution of each channel alters their relative intensities, which can cause shifts in hue and saturation.

- **Advantages:** RGB equalization enhances contrast in each channel, which can reveal more detail in low-contrast regions for each color independently.
- **Disadvantages:** It often introduces color distortions, particularly if the original image has strong color biases or if specific channels have very different distributions.

2. Equalization of Intensity (V Channel) in HSV Space

- **Method:** In HSV equalization, only the V (intensity/brightness) channel is equalized, leaving the H (hue) and S (saturation) channels unaltered.
- **Effect on Image:** This approach enhances the brightness and contrast of the image without affecting the colors, resulting in a more natural-looking enhancement. Since hue and saturation remain the same, the overall color balance is preserved.
- **Advantages:** Preserves color fidelity, making it more visually pleasing. It enhances the brightness while maintaining the original color characteristics, providing a balanced improvement in contrast.
- **Disadvantages:** The enhancement is limited to the intensity channel, so it may not be as pronounced in contrast details compared to individual channel equalization in RGB.

the **arithmetic mean**, **weighted average**, **median**, and **alpha-trimmed mean** spatial filters perform when applied using 3×33 \times 33×3 and 5×55 \times 55×5 kernels on an image, specifically using **alpha values of 4 and 6 for 3×33 \times 33×3** and **12 and 18 for 5×55 \times 55×5** kernels in the alpha-trimmed filter. Each filter has unique properties that influence the outcome in terms of noise reduction, edge preservation, and smoothness. Let's look at each in turn:

1. Arithmetic Mean Filter

- **Description:** The arithmetic mean filter averages all pixel values within the kernel.
- **Effect of Kernel Size:**
  - **3×33 \times 33×3 Kernel:** Provides mild smoothing, reducing noise but with limited blur. Smaller details and edges are generally preserved better.
  - **5×55 \times 55×5 Kernel:** Provides more aggressive smoothing, resulting in a stronger blur effect. While it reduces noise more effectively, it also causes more blurring, making the image look softer and potentially losing finer details.
- **Strengths:** Good at reducing random noise (Gaussian noise) and is computationally simple.
- **Weaknesses:** Not effective against salt-and-pepper noise, as outliers still influence the mean.

2. Weighted Average Filter

- **Description:** This filter assigns different weights to pixels in the kernel, giving more weight to the central pixel or nearby pixels.
- **Effect of Kernel Size:**
  - **3×33 \times 33×3 Kernel:** Provides smooth blurring while preserving the central pixel's influence. Noise reduction is moderate and is generally better than a simple mean filter.
  - **5×55 \times 55×5 Kernel:** The larger kernel increases the smoothing effect, creating a blurrier result with increased noise suppression.
- **Strengths:** Can be tuned to emphasize certain regions in the kernel (e.g., center) and can be more effective at preserving edges.
- **Weaknesses:** More complex than the arithmetic mean filter, and excessive smoothing can blur fine details.

3. Median Filter

- **Description:** The median filter replaces the center pixel with the median value of all pixels in the kernel. This is particularly effective for salt-and-pepper noise.
- **Effect of Kernel Size:**
  - **3×33 \times 33×3 Kernel:** Very effective at removing small amounts of salt-and-pepper noise while preserving edges. Limited smoothing effect, so fine details are mostly retained.
  - **5×55 \times 55×5 Kernel:** Increased noise removal power, but also blurs small details more significantly. Still preserves edges better than mean filters.
- **Strengths:** Excellent for removing salt-and-pepper noise and preserving edges.
- **Weaknesses:** Not as effective for Gaussian noise. Larger kernels can reduce detail due to median blurring.

4. Alpha-Trimmed Mean Filter

- **Description:** This filter discards the alpha largest and alpha smallest pixel values within the kernel before averaging the remaining values.
- **Effect of Kernel Size and Alpha Values:**
  - **3×33 \times 33×3 Kernel (alpha = 4 or 6):**
    - **Alpha = 4:** Eliminates two extreme values (highest and lowest) from consideration, offering balanced noise reduction without extreme outliers. Good for salt-and-pepper noise without too much blurring.
    - **Alpha = 6:** Discards more values, resulting in stronger noise suppression but slightly higher blurring, reducing image sharpness and possibly affecting edges.
  - **5×55 \times 55×5 Kernel (alpha = 12 or 18):**
    - **Alpha = 12:** Larger kernel with moderate trimming provides good balance between noise removal and detail preservation.
    - **Alpha = 18:** Eliminates most extreme values, achieving significant noise reduction and blur effect. This level is often more effective for strong salt-and-pepper noise, but fine details and edges may suffer.
- **Strengths:** Highly flexible filter, effective for images with mixed types of noise (salt-and-pepper, Gaussian).
- **Weaknesses:** Requires tuning of alpha parameter; excessive trimming can reduce detail and edge sharpness.

6. <u>Conclusion:</u>

In conclusion, this study compared various spatial filters and histogram equalization techniques on noisy images. Equalizing the RGB channels enhanced brightness but often altered color balance, while equalizing the intensity channel in HSV preserved color fidelity. The arithmetic mean and weighted average filters provided smoothing, with the latter retaining more edge detail. The median filter effectively removed salt-and-pepper noise, preserving structural details, and the alpha-trimmed filter offered a balance between noise reduction and edge clarity, especially with higher alpha values. Using 5x5 kernels improved noise reduction but slightly softened details. Each approach demonstrated distinct advantages, underscoring the importance of filter and channel selection based on noise type and image requirements.

7. **Code:**

```
I = imread('peppers.png');
[R, G, B] = imsplit(I);
figure;
display_image_with_histogram(R, 'Red Channel', 1);
display_image_with_histogram(G, 'Green Channel', 2);
display_image_with_histogram(B, 'Blue Channel', 3);
display_image_with_histogram(I, 'Coloured Image', 4);
equalized_R = histogram_equalization(R);
equalized_G = histogram_equalization(G);
equalized_B = histogram_equalization(B);
equalized_RGB = cat(3, equalized_R, equalized_G, equalized_B);
display_image_with_histogram(equalized_R, 'Equalized R', 5);
display_image_with_histogram(equalized_G, 'Equalized G', 6);
display_image_with_histogram(equalized_B, 'Equalized B', 7);
display_image_with_histogram(equalized_RGB, 'Equalized Coloured', 8);
HSV = rgb2hsv(I);
[h, s, v] = imsplit(HSV);
h_uint8 = uint8(h * 255),s_uint8 = uint8(s * 255),v_uint8 = uint8(v * 255);
equalized_v = histogram_equalization(v_uint8);
HSV_equalized = cat(3, h, s, double(equalized_v) / 255);
equalized_colored_image = hsv2rgb(HSV_equalized);
display_image_with_histogram(h_uint8, 'Hue (H Channel)',9);
display_image_with_histogram(s_uint8, 'Saturation (S Channel)', 10);
display_image_with_histogram(v_uint8, 'Intensity (V Channel)', 11);
display_image_with_histogram(equalized_v, 'Equalized Intensity', 12);
display_image_with_histogram(equalized_colored_image, 'Equalized Image (HSV)',13);
function display_image_with_histogram(image, title_text, position)
  subplot(4, 4, position), imshow(image), title(title_text);
end
figure;
subplot(2,3,1), imhist(I), title('Histogram of RGB-image');
subplot(2,3,4), imhist(equalized_RGB), title('Histogram of equi-RGB image');
subplot(2,3,2), imhist(v_uint8), title('Histogram of intensity-channel');
subplot(2,3,5), imhist(equalized_v), title('Histogram of equi-intensity-channel');
subplot(2,3,3), imhist(I), title('Histogram of RGB-image');
subplot(2,3,6), imhist(equalized_colored_image), title('Histogram of equi-channel(HSV)');
function equi_img = histogram_equalization(input_image)
  cdf = cumsum(imhist(input_image) / numel(input_image));
  sk = uint8(round(cdf * 255));
  equi_img = sk(double(input_image) + 1);
end
img = imread("peppers.png");
noise_types = {'salt & pepper', 'gaussian', 'speckle'};
noises = {imnoise(img, 'salt & pepper', 0.1), imnoise(img, 'gaussian'), imnoise(img, 'speckle')};
figure();
for k = 1:3
  subplot(1, 3, k); imshow(noises{k});
  [mse, psnr] = calculateMSE_PSNR(noises{k}, img);
  title(sprintf('%s \nMSE = %.4f\n PSNR = %.4f', noise_types{k}, mse, psnr));
```

```
end
arith_kernal_3x3 = ones(3, 3) / 9;
weight_kernal_3x3 = [1, 2, 1; 2, 4, 2; 1, 2, 1] / 16;
arith_kernal_5x5 = ones(5, 5) / 25;
weight_kernal_5x5 = [1,1,2,1,1;1,2,4,2,1;2,4,8,4,2;1,2,4,2,1;1,1,2,1,1] / 52;
for k = 1:3
    noise_img = noises{k};
    hsv_img = rgb2hsv(noise_img);  % Convert to HSV for HSV-based filtering
    intensity_channel = hsv_img(:,:,3);  % Extract the V (intensity) channel
    [rows, cols, ~] = size(noise_img);
    filter_results_rgb_3x3 = zeros(rows, cols, 3, 5, 'double');
    filter_results_rgb_5x5 = zeros(rows, cols, 3, 5, 'double');
    filter_results_hsv_3x3 = zeros(rows, cols, 5, 'double');
    filter_results_hsv_5x5 = zeros(rows, cols, 5, 'double');
    for ch = 1:3
        channel = double(noise_img(:,:,ch));
        pad_3x3 = padarray(channel, [1, 1], 0, 'both');
        pad_5x5 = padarray(channel, [2, 2], 0, 'both');
        for i = 2 : rows + 1
            for j = 2 : cols + 1
                submatrix_3x3 = pad_3x3(i-1:i+1, j-1:j+1);
                vector_3x3 = sort(submatrix_3x3(:));
                filter_results_rgb_3x3(i-1, j-1, ch, 1) = sum(submatrix_3x3(:) .* arith_kernal_3x3(:));
                filter_results_rgb_3x3(i-1, j-1, ch, 2) = sum(submatrix_3x3(:) .* weight_kernal_3x3(:));
                filter_results_rgb_3x3(i-1, j-1, ch, 3) = median(vector_3x3);                % Median
                filter_results_rgb_3x3(i-1, j-1, ch, 4) = mean(vector_3x3(3:7));             % 4-trimmed
                filter_results_rgb_3x3(i-1, j-1, ch, 5) = mean(vector_3x3(4:6));             % 6-trimmed
            end
        end
        for i = 3 : rows + 2
            for j = 3 : cols + 2
                submatrix_5x5 = pad_5x5(i-2:i+2, j-2:j+2);
                vector_5x5 = sort(submatrix_5x5(:));
                filter_results_rgb_5x5(i-2, j-2, ch, 1) = sum(submatrix_5x5(:) .* arith_kernal_5x5(:));
                filter_results_rgb_5x5(i-2, j-2, ch, 2) = sum(submatrix_5x5(:) .* weight_kernal_5x5(:));
                filter_results_rgb_5x5(i-2, j-2, ch, 3) = median(vector_5x5);                % Median
                filter_results_rgb_5x5(i-2, j-2, ch, 4) = mean(vector_5x5(7:19));            % 12-trimmed
                filter_results_rgb_5x5(i-2, j-2, ch, 5) = mean(vector_5x5(10:16));           % 18-trimmed
            end
        end
    end
    pad_3x3 = padarray(intensity_channel, [1, 1], 0, 'both');
    pad_5x5 = padarray(intensity_channel, [2, 2], 0, 'both');
    for i = 2 : rows + 1
        for j = 2 : cols + 1
            submatrix_3x3 = pad_3x3(i-1:i+1, j-1:j+1);
            vector_3x3 = sort(submatrix_3x3(:));
            filter_results_hsv_3x3(i-1, j-1, 1) = sum(submatrix_3x3(:) .* arith_kernal_3x3(:));
            filter_results_hsv_3x3(i-1, j-1, 2) = sum(submatrix_3x3(:) .* weight_kernal_3x3(:));
            filter_results_hsv_3x3(i-1, j-1, 3) = median(vector_3x3);                % Median
            filter_results_hsv_3x3(i-1, j-1, 4) = mean(vector_3x3(3:7));             % 4-trimmed
```

```
            filter_results_hsv_3x3(i-1, j-1, 5) = mean(vector_3x3(4:6));                    % 6-trimmed
        end
    end
    for i = 3 : rows + 2
        for j = 3 : cols + 2
            submatrix_5x5 = pad_5x5(i-2:i+2, j-2:j+2);
            vector_5x5 = sort(submatrix_5x5(:));
            filter_results_hsv_5x5(i-2, j-2, 1) = sum(submatrix_5x5(:) .* arith_kernal_5x5(:));
            filter_results_hsv_5x5(i-2, j-2, 2) = sum(submatrix_5x5(:) .* weight_kernal_5x5(:));
            filter_results_hsv_5x5(i-2, j-2, 3) = median(vector_5x5);                         % Median
            filter_results_hsv_5x5(i-2, j-2, 4) = mean(vector_5x5(7:19));                    % 12-trimmed
            filter_results_hsv_5x5(i-2, j-2, 5) = mean(vector_5x5(10:16));                   % 18-trimmed
        end
    end
    figure('Name', ['Noise Type: ', noise_types{k}]);
    for f = 1:5
        filtered_img_rgb_3x3 = uint8(cat(3, filter_results_rgb_3x3(:,:,1,f), filter_results_rgb_3x3(:,:,2,f),
filter_results_rgb_3x3(:,:,3,f)));
        filtered_img_rgb_5x5 = uint8(cat(3, filter_results_rgb_5x5(:,:,1,f), filter_results_rgb_5x5(:,:,2,f),
filter_results_rgb_5x5(:,:,3,f)));
        [mse_3x3, psnr_3x3] = calculateMSE_PSNR(filtered_img_rgb_3x3, img);
        [mse_5x5, psnr_5x5] = calculateMSE_PSNR(filtered_img_rgb_5x5, img);
        hsv_img_3x3 = hsv_img;
        hsv_img_3x3(:,:,3) = filter_results_hsv_3x3(:,:,f);
        filtered_img_hsv_3x3 = hsv2rgb(hsv_img_3x3);
        [mse_hsv_3x3, psnr_hsv_3x3] = calculateMSE_PSNR(filtered_img_hsv_3x3, img);
        hsv_img_5x5 = hsv_img;
        hsv_img_5x5(:,:,3) = filter_results_hsv_5x5(:,:,f);
        filtered_img_hsv_5x5 = hsv2rgb(hsv_img_5x5);
        [mse_hsv_5x5, psnr_hsv_5x5] = calculateMSE_PSNR(filtered_img_hsv_5x5, img);
        subplot(4, 5, f); imshow(filtered_img_rgb_3x3);
        title(sprintf('RGB 3x3 Filter %d\nMSE = %.4f\nPSNR = %.4f', f, mse_3x3, psnr_3x3));
        subplot(4, 5, f + 5); imshow(filtered_img_hsv_3x3);
        title(sprintf('HSV 3x3 Filter %d\nMSE = %.4f\nPSNR = %.4f', f, mse_hsv_3x3, psnr_hsv_3x3));
        subplot(4, 5, f + 10); imshow(filtered_img_rgb_5x5);
        title(sprintf('RGB 5x5 Filter %d\nMSE = %.4f\nPSNR = %.4f', f, mse_5x5, psnr_5x5));
        subplot(4, 5, f + 15); imshow(filtered_img_hsv_5x5);
        title(sprintf('HSV 5x5 Filter %d\nMSE = %.4f\nPSNR = %.4f', f, mse_hsv_5x5, psnr_hsv_5x5));
    end
end
function [mse, psnr] = calculateMSE_PSNR(filtered_img, original_img)
    mse = mean((double(filtered_img(:)) - double(original_img(:))).^2);
    psnr = 10 * log10(255^2 / mse);
end
```