Assignment no:4

1. **Aim:** To implement denoising in spatial domain using Gray Image.
2. **Software Tool Used:** MATLAB
3. **Theory:**

Noise:

1. Gaussian Noise: Gaussian noise affects all image pixels and follows a normal distribution (bell curve). The noise varies randomly but centers around a mean value with a certain variance.

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

- where μ is the mean, σ^2 is the variance, and x is the intensity value.

2. Salt and Pepper Noise: Salt and pepper noise randomly turns some pixels black (0) or white (255) in a grayscale image. It is a form of impulse noise, often caused by data transmission errors.

$$I(x,y) = \begin{cases} 0 & \text{with probability } p_{salt} \\ 255 & \text{with probability } p_{pepper} \\ I(x,y) & \text{otherwise} \end{cases}$$

- where p_{salt} and p_{pepper} are the probabilities of salt and pepper noise.

3. Speckle Noise: Speckle noise is a multiplicative noise common in coherent imaging systems like radar and ultrasound. The noise varies based on the intensity of the pixel.

$$I_{noisy}(x,y) = I_{original}(x,y) + n(x,y) \cdot I_{original}(x,y)$$

where $n(x,y)$ is the noise and $I_{original}(x,y)$ is the original pixel value.

Different type of filter:

1. Arithmetic Mean Filter: This filter replaces the value of each pixel by the arithmetic mean (average) of the pixel values in its neighbourhood. It smooths the image but can blur edges. Reduces Gaussian noise but causes blurring.

$$I_{filtered}(x,y) = \frac{1}{N}\sum_{i,j} I(x+i, y+j)$$

Where N is the size of kernel m*n, and |(i, j)| <= |(m\2,n\2)|.

2. Geometric Mean Filter: The geometric mean filter computes the geometric mean of the pixel values in the neighbourhood. It provides a more subtle smoothing effect compared to the arithmetic mean filter. Reduces noise while preserving some sharpness in the image.

$$I_{filtered}(x,y) = \left(\prod_{i,j} I(x+i, y+j)\right)^{1/N}$$

Where N is the size of kernel m*n, and |(i, j)| <= |(m\2,n\2)|.

3. Weighted Average Filter: In this filter, different weights are assigned to the pixels in the neighbourhood, usually giving more weight to the centre pixel, resulting in better smoothing while reducing edge blurring. Improves noise reduction while slightly preserving edges.

$$I_{filtered}(x, y) = \sum_{i,j} w(i, j) \cdot I(x + i, y + j)$$

Where w (i, j) is weight kernel is the size of kernel m*n, and |(i, j)| <= |(m\2,n\2)|.

$$\text{3x3 Kernel} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad \text{5x5 Kernel} = \frac{1}{52}\begin{bmatrix} 1 & 1 & 2 & 1 & 1 \\ 1 & 2 & 4 & 2 & 1 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \\ 1 & 1 & 2 & 1 & 1 \end{bmatrix}$$

4. Minimum Filter: The minimum filter replaces each pixel with the minimum value from its neighbourhood. It is effective at removing salt noise (white pixels). Preserves darker areas, removing bright outliers like salt noise.

$$I_{filtered}(x, y) = \min\{I(x + i, y + j)\}$$

Where the minimum is taken over of neighbourhood of kernel size, and |(i, j)| <= |(m\2,n\2)|.

5. Maximum Filter: The maximum filter replaces each pixel with the maximum value from its neighbourhood. It is effective at removing pepper noise (black pixels). Preserves brighter areas, removing dark outliers like pepper noise..

$$I_{filtered}(x, y) = \max\{I(x + i, y + j)\}$$

Where the maximum is taken over of neighbourhood of kernel size, and |(i, j)| <= |(m\2,n\2)|.

6. Median Filter: This filter replaces the pixel value with the median of the pixel values in the neighbourhood. It is highly effective for removing salt and pepper noise while preserving edges. Reduces impulse noise and maintains edges well.

Formula: where the median is taken over the neighbourhood of kernel size.

$$I_{filtered}(x, y) = \text{median}\{I(x + i, y + j)\}$$

7. Alpha-Trimmed Mean Filter: The alpha-trimmed mean filter removes a certain number of extreme pixel values (both highest and lowest) and computes the average of the remaining values. This makes it robust to both Gaussian and impulse noise. Offers a compromise between median and mean filtering by handling both Gaussian and impulse noise.

- **Formula**: after trimming d maximum and d minimum values from the neighbourhood of kernel size. Where 2d is the alpha value.

$$I_{filtered}(x, y) = \frac{1}{N - 2d} \sum_{i,j} I(x + i, y + j)$$

8. Mid-Point Filter: The mid-point filter takes the average of the maximum and minimum pixel values in the neighbourhood. It is useful for removing noise in images with uniform noise distribution. Provides noise reduction while preserving detail in images with both Gaussian and impulse noise.

$$I_{filtered}(x, y) = \frac{1}{2}\left(\max\{I(x + i, y + j)\} + \min\{I(x + i, y + j)\}\right)$$

where the max and min are taken over the neighbourhood in size of kernel.

4. Result:

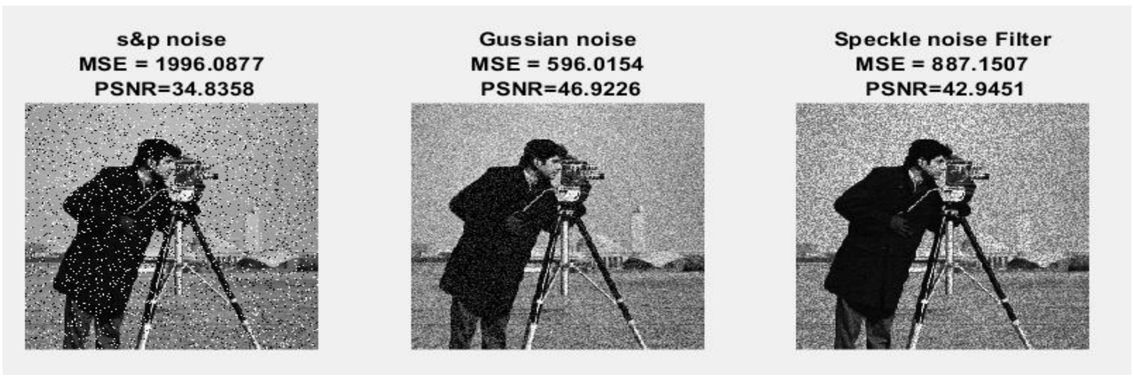


Fig (1): effect of different noise on image “cameraman.tif”

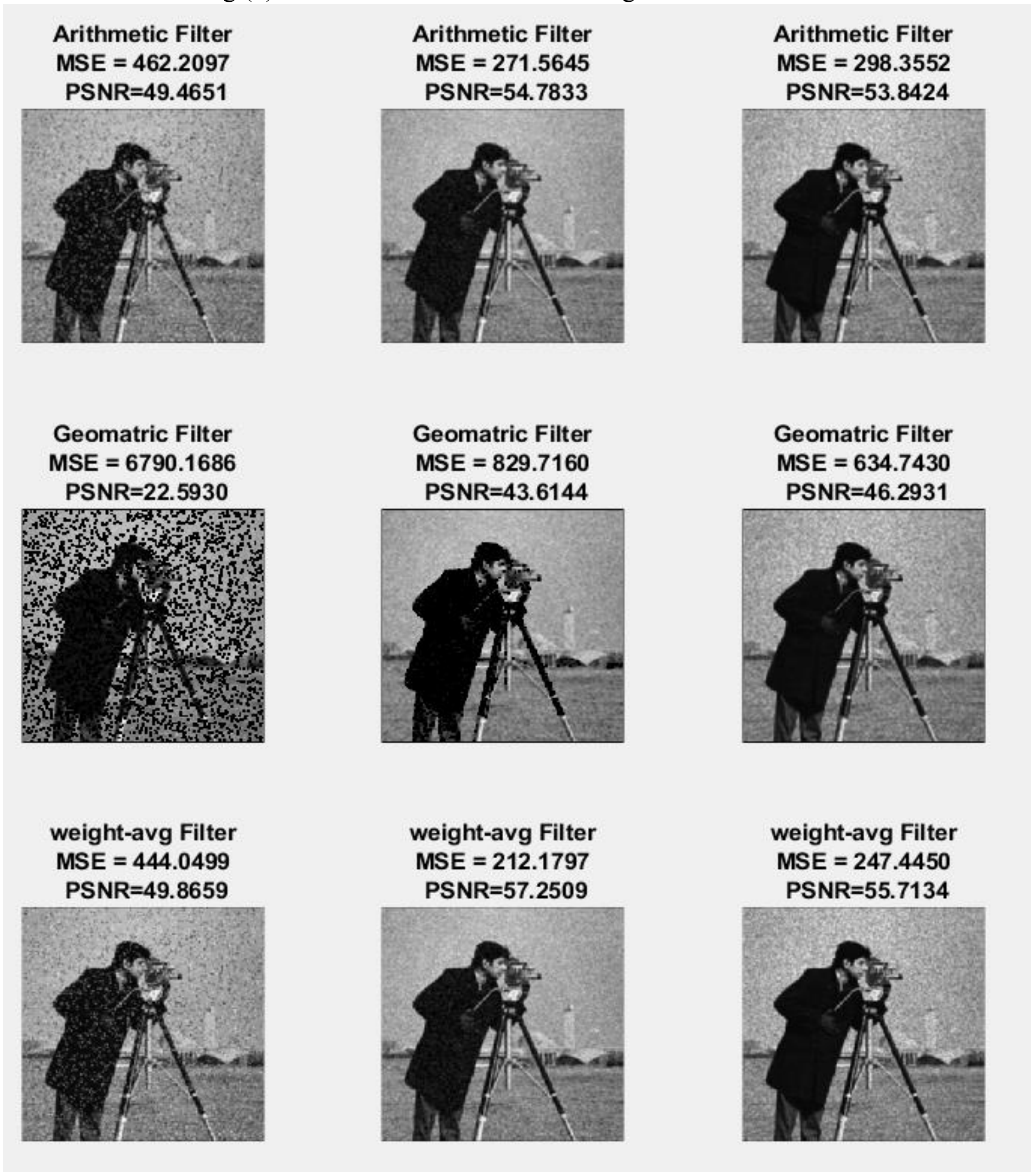


fig (2): effect of arithmetic, geometric and weight average filter using 3*3 kernel

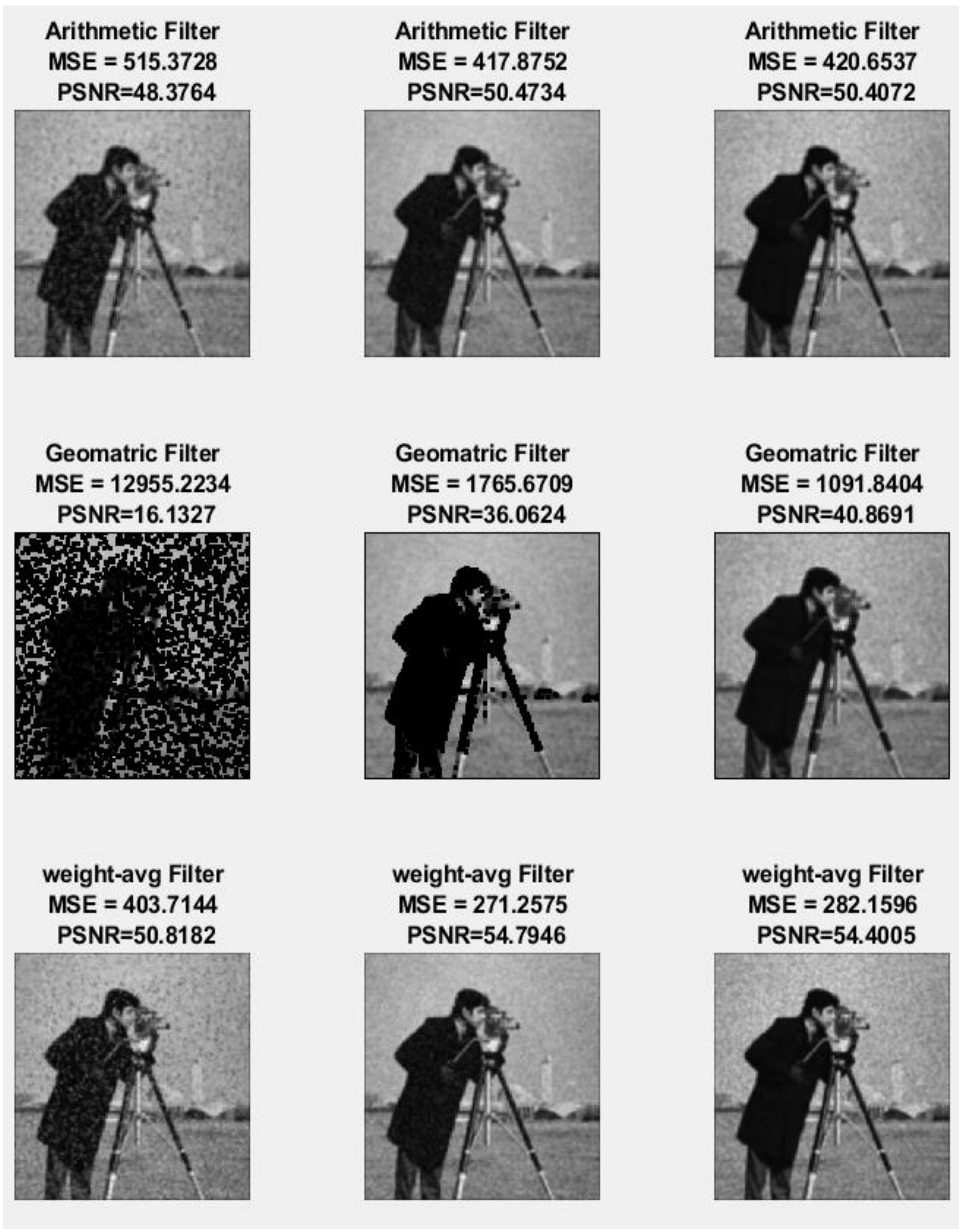


fig (3): effect of arithmetic, geometric and weight average filter using 5*5 kernel

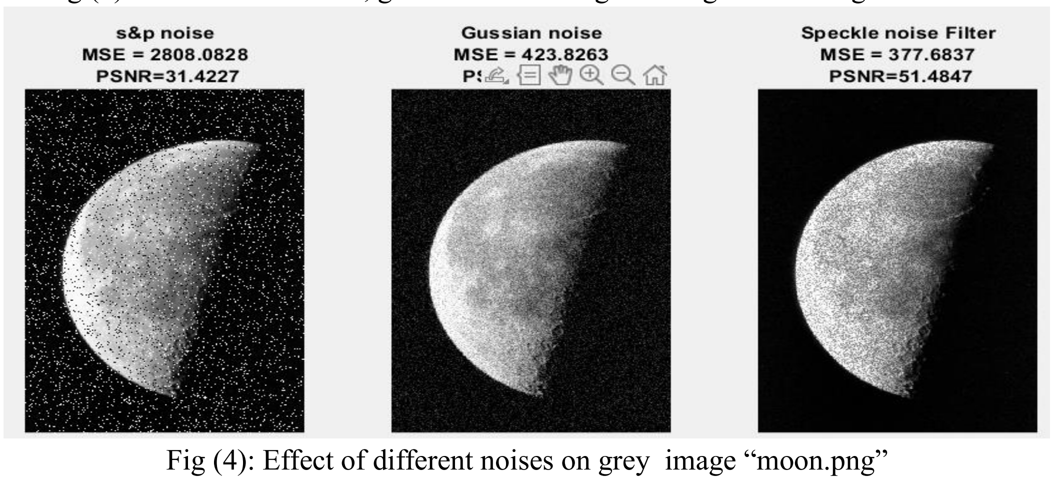
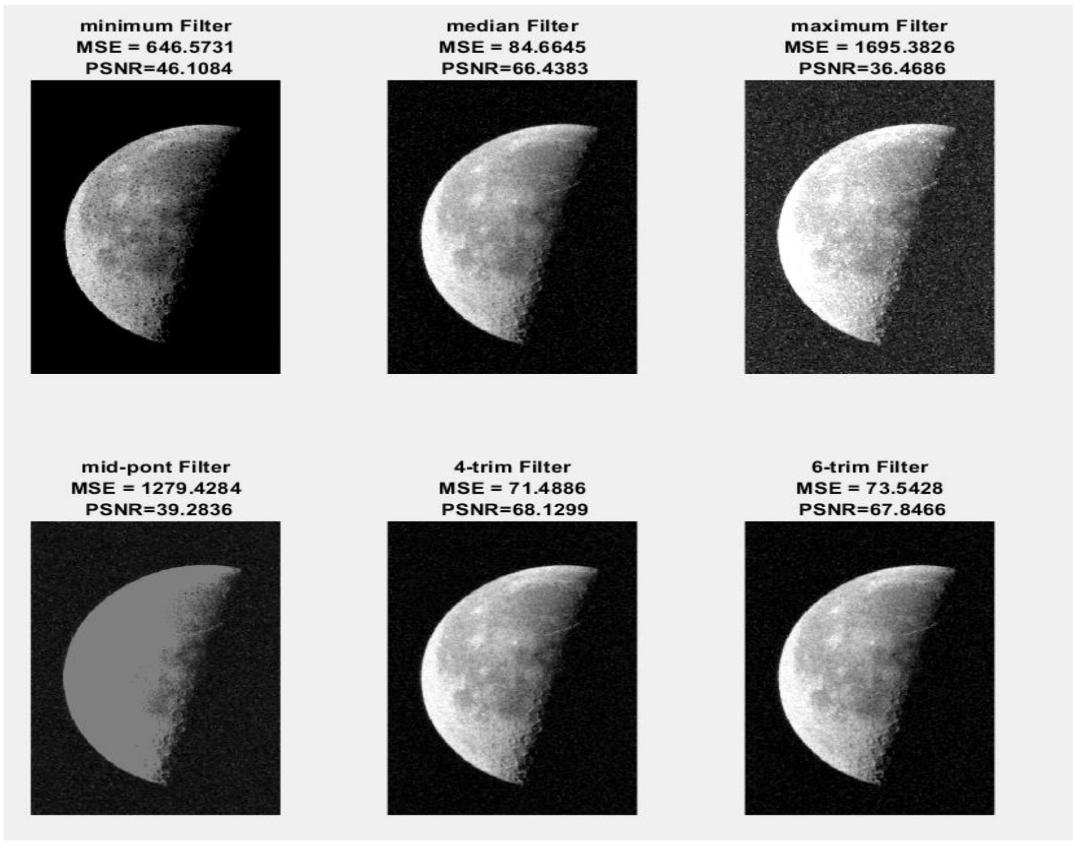


Fig (4): Effect of different noises on grey image "moon.png"



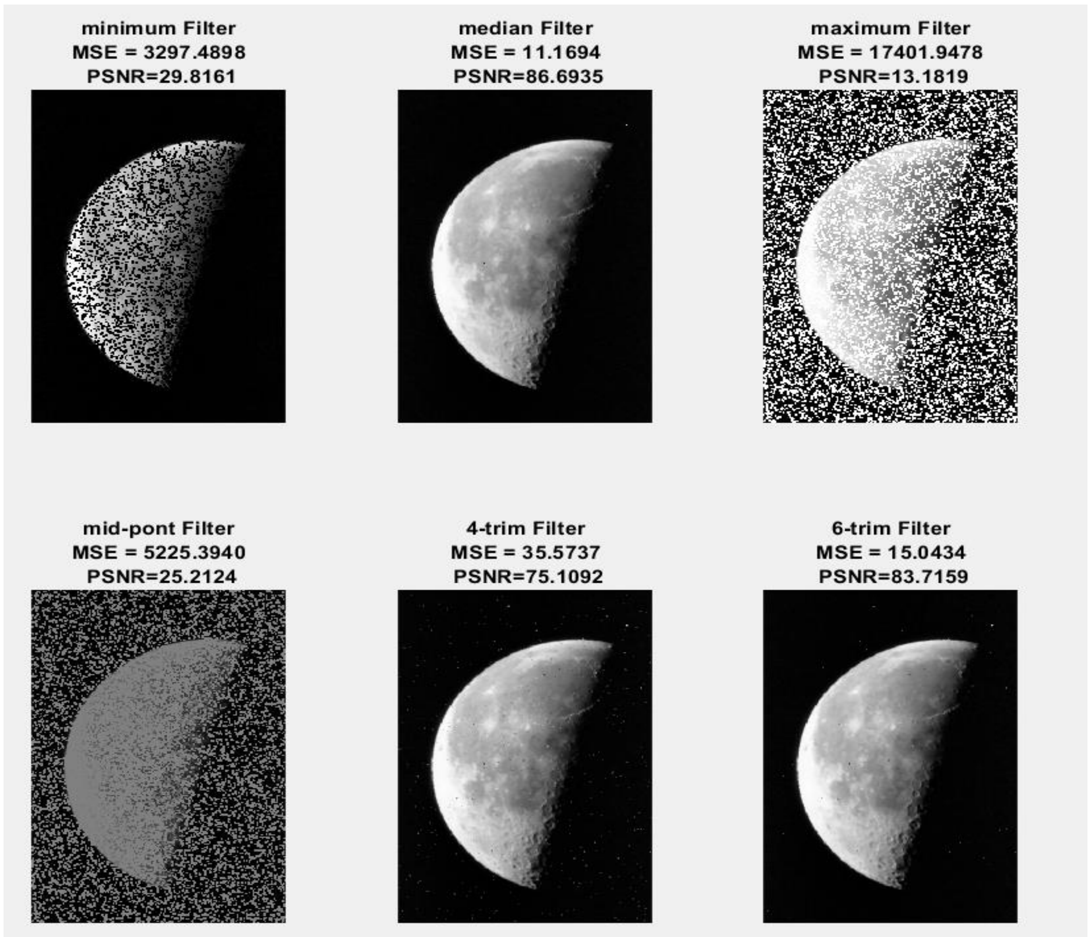


fig (6): effect of different filter on Gaussian noise Image the using the 3*3 kernel

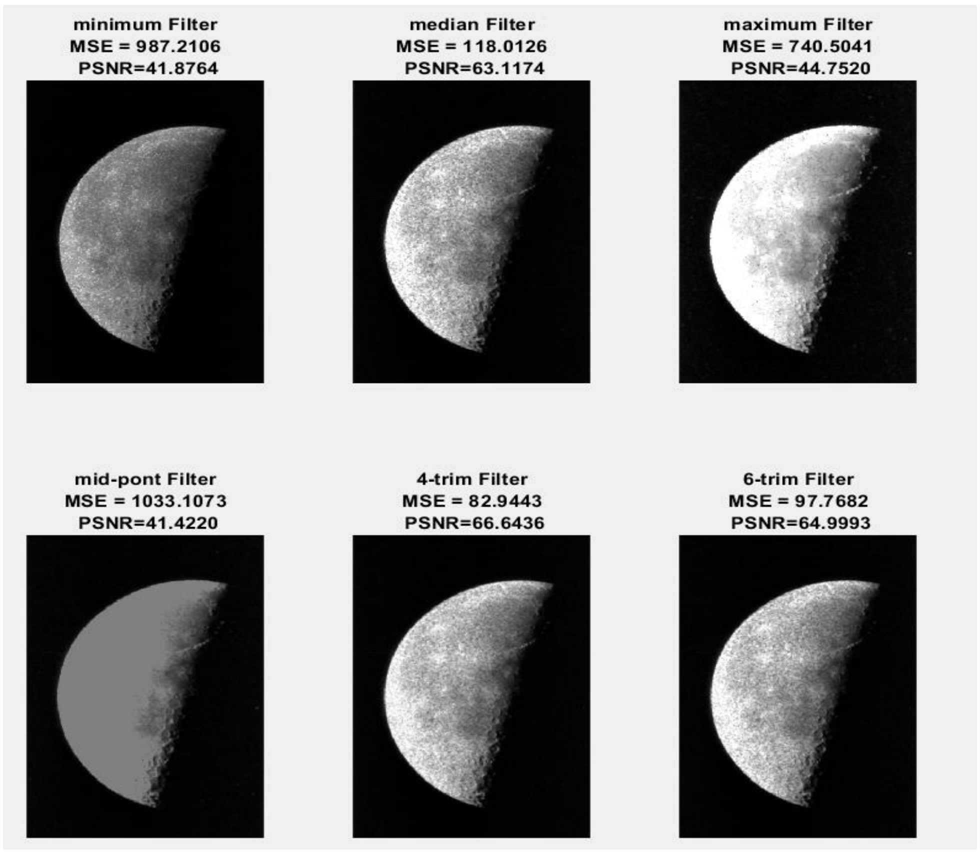


fig (7): effect of different filter on speckle noise Image the using the 3*3 kernel

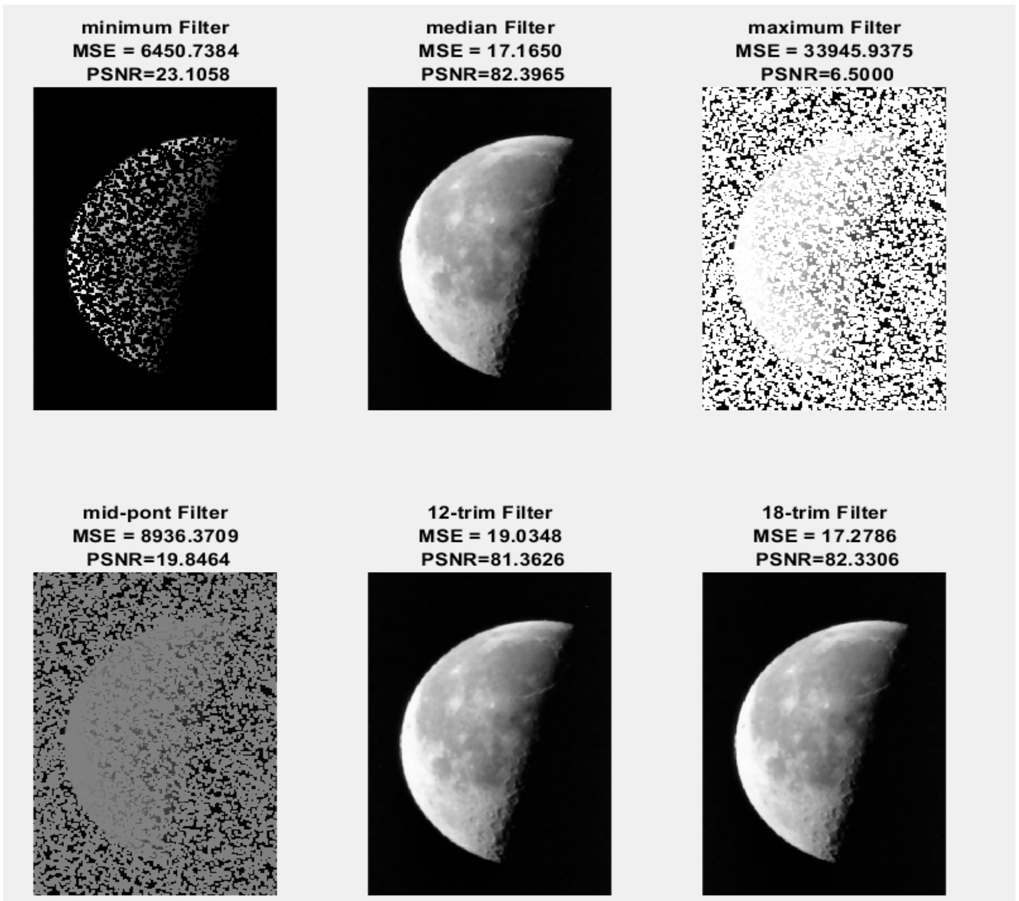
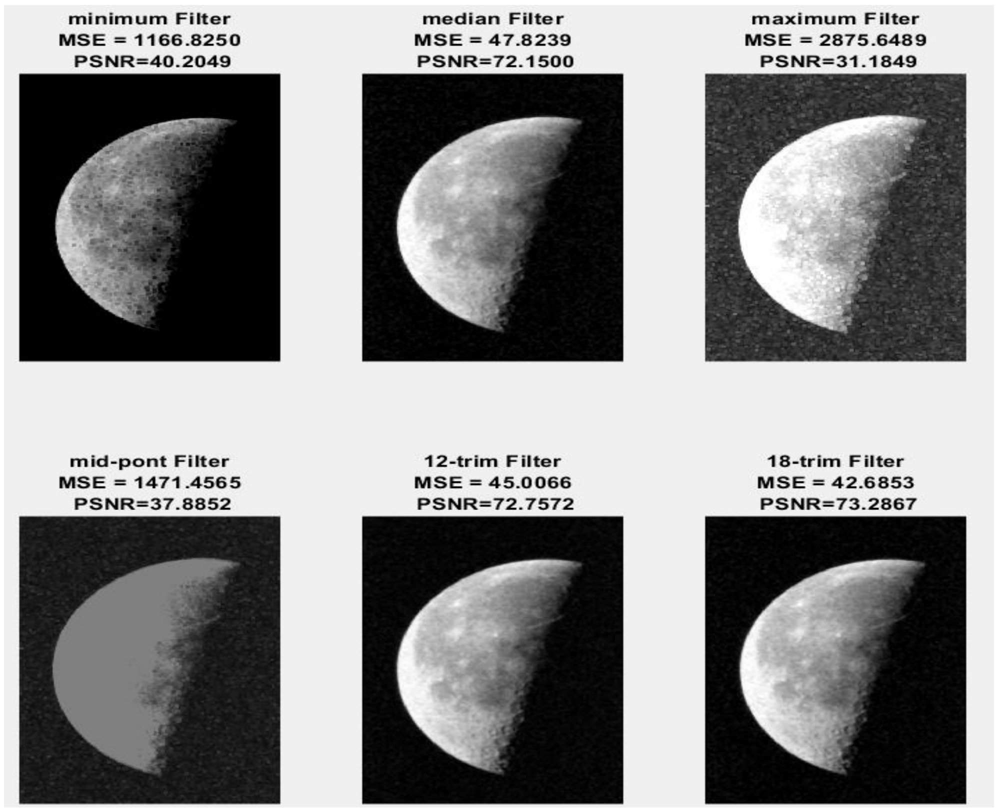
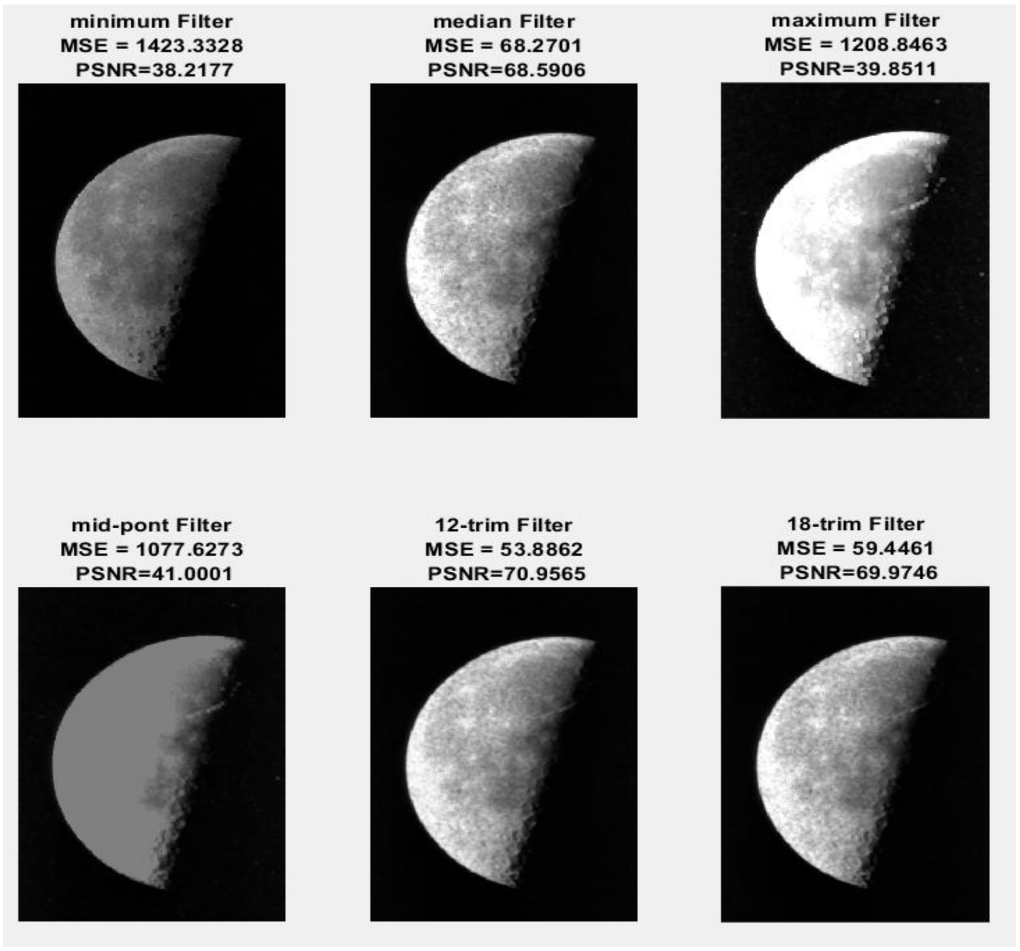


fig (8): effect of different filter on salt and pepper noise Image the using the 5*5 kernel



fig(9): effect of different filter on gaussian noise Image the using the 5*5 kernel



fig(10): effect of different filter on speckle noise Image the using the 5*5 kernel

5. <u>**Discussion:**</u>

   1. **Arithmetic Mean Filter**

- **3x3 Kernel**:
  - **Performance**: Smooths the image by averaging pixel values within the kernel. This filter effectively reduces noise but may blur edges and fine details.
  - **Noise Reduction**: Effective for Gaussian and speckle noise. Less effective for salt and pepper noise as it cannot distinguish between noisy and clean pixels very well.
- **5x5 Kernel**:
  - **Performance**: Provides better noise reduction compared to the 3x3 kernel but introduces more blurring.
  - **Noise Reduction**: Improved noise reduction for all types of noise, but edge sharpness and fine details are significantly reduced.

   2. **Geometric Mean Filter**

- **3x3 Kernel**:
  - **Performance**: Uses the geometric mean of pixel values, which helps in preserving details better than the arithmetic mean filter. It is effective in handling multiplicative noise like speckle noise.
  - **Noise Reduction**: Excellent for Gaussian and speckle noise, and somewhat effective for salt and pepper noise, but not as robust as the median filter.
- **5x5 Kernel**:
  - **Performance**: Provides better noise reduction but at the cost of slight detail loss.
  - **Noise Reduction**: Enhanced noise reduction, especially for speckle noise, but with some loss of sharpness.

   3. **Weighted Average Filter**

- **3x3 Kernel**:
  - **Performance**: Assigns different weights to pixels, giving more importance to central pixels. It strikes a balance between noise reduction and detail preservation.
  - **Noise Reduction**: Effective for Gaussian and speckle noise, offers improved detail preservation compared to the arithmetic mean filter.
- **5x5 Kernel**:
  - **Performance**: More aggressive in noise reduction but can lead to significant blurring and loss of details.
  - **Noise Reduction**: Better noise reduction overall, but results in more noticeable image smoothing.

   4. **Minimum Filter**

- **3x3 Kernel**:
  - **Performance**: Replaces the central pixel with the minimum value within the neighbourhood. Very effective against salt and pepper noise but darkens the image.

    - **Noise Reduction**: Efficient in removing salt noise (pepper), but less effective for Gaussian and speckle noise.
- **5x5 Kernel**:
    - **Performance**: Further reduces noise but at the expense of image darkening.
    - **Noise Reduction**: Improved salt noise removal but less effective in preserving image brightness.

5. **Maximum Filter**

- **3x3 Kernel**:
    - **Performance**: Replaces the central pixel with the maximum value, effective against pepper noise.
    - **Noise Reduction**: Useful for removing pepper noise but can result in a loss of bright details.
- **5x5 Kernel**:
    - **Performance**: Better at removing pepper noise, though it can cause a loss of finer bright details.
    - **Noise Reduction**: More effective for salt noise (bright areas) but can introduce some unwanted brightening in the image.

6. **Median Filter**

- **3x3 Kernel**:
    - **Performance**: Replaces the central pixel with the median of pixel values in the neighbourhood. Highly effective in preserving edges and removing salt and pepper noise.
    - **Noise Reduction**: The most effective for salt and pepper noise. Also works well for Gaussian noise, although less so for speckle noise.
- **5x5 Kernel**:
    - **Performance**: Offers superior noise reduction, especially for high levels of salt and pepper noise, with a slight increase in blurring.
    - **Noise Reduction**: Improved removal of noise but with some loss of sharpness.

7. **Alpha-Trimmed Mean Filter**

- **3x3 Kernel**:
    - **Performance**: Discards a certain percentage of extreme values before averaging, making it robust against impulsive noise.
    - **Noise Reduction**: Effective for all types of noise, particularly for salt and pepper, with good balance between noise reduction and detail preservation.
- **5x5 Kernel**:
    - **Performance**: Provides even better noise reduction but can cause more significant smoothing.
    - **Noise Reduction**: Enhanced performance in noise reduction but with increased image blurring.

Small Alpha Value:

- **Fewer extreme values are discarded.**
- **Less aggressive noise reduction; better preservation of image details.**
- **Small effect across different kernel sizes, but larger kernels might still include some noise.**

Large Alpha Value:

- **More extreme values are discarded.**
- **More effective noise reduction; potential for increased blurring and loss of details.**
- **Kernel Size Influence: Larger kernels with high alpha values may cause excessive blurring, while smaller kernels provide moderate noise reduction with less detail loss.**

8. Mid-Point Filter

- **3x3 Kernel**:
  - **Performance**: Computes the average of the maximum and minimum pixel values in the neighbourhood. Balances between noise reduction and detail preservation.
  - **Noise Reduction**: Effective for salt and pepper noise and offers decent performance for Gaussian and speckle noise.
- **5x5 Kernel**:
  - **Performance**: Provides stronger noise reduction, especially for higher levels of noise, but at the cost of some detail loss.
  - **Noise Reduction**: Better noise reduction with increased smoothing, balancing between detail loss and noise removal.

6. Conclusion:

Different filters and kernel sizes have varying impacts on image quality and noise reduction. The **median filter** is superior for salt and pepper noise, while the **geometric mean filter** and **alpha-trimmed mean filter** are better suited for Gaussian and speckle noise. Kernel size plays a critical role in balancing noise reduction with image detail preservation, with smaller kernels offering better detail retention and larger kernels providing stronger noise reduction.

Filters like the median and alpha-trimmed mean showed superior performance in preserving details while effectively removing noise. The 3x3 kernels generally balanced noise reduction and detail preservation better than the 5x5 kernels, which tended to introduce more blurring. The choice of filter and kernel size should be tailored to the specific type of noise and the level of detail retention required, with alpha-trimmed mean filters offering flexibility in noise reduction versus detail preservation.

7. Reference:

1.Gonzalez, R.C., & Woods, R.E. (2018). *Digital Image Processing* (4th ed.). Pearson.

2.Pratt, W.K. (2007). *Digital Image Processing: PIKS Scientific Inside* (4th ed.). Wiley-Interscalene.

8. Code:

```
r = imread("cameraman.tif");
[rows, cols] = size(r);
sp_noise      = imnoise(r, 'salt & pepper', 0.1);
gussian_noise  = imnoise(r, 'gaussian');
speckle_noise  = imnoise(r, 'speckle');

figure('Name',"noise Image");
subplot(1,3,1);   imshow(sp_noise);
mse = sum((double(r(:))-double(sp_noise(:))).^2)/(rows*cols);
psnr = 10*log(255*255/mse);
title(sprintf('s&p noise\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));
subplot(1,3,2);   imshow(gussian_noise);
mse = sum((double(r(:))-double(gussian_noise(:))).^2)/(rows*cols);
psnr = 10*log(255*255/mse);
title(sprintf('Gussian noise\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));
subplot(1,3,3);   imshow(speckle_noise);
mse = sum((double(r(:))-double(speckle_noise(:))).^2)/(rows*cols);
psnr = 10*log(255*255/mse);
title(sprintf('Speckle noise Filter\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));


filtered_img1 = zeros(rows, cols, 'uint8'); % for arithmetic filter
filtered_img2 = zeros(rows, cols, 'uint8'); % for geometric filter
filtered_img3 = zeros(rows, cols, 'uint8'); % for weighted average filter

sp_pad      = padarray(sp_noise, [1, 1], 0, 'both');
gussian_pad  = padarray(gussian_noise, [1, 1], 0, 'both');
speckle_pad  = padarray(speckle_noise, [1, 1], 0, 'both');
noise_array = {sp_pad, gussian_pad, speckle_pad};

arith_kernal = ones(3, 3) / 9;
weight_kernal = [1, 2, 1; 2, 4, 2; 1, 2, 1] / 16;
figure('Name',"for 3*3 kernal");
for k = 1:3
   current_noise_img = noise_array{k}; % Get the padded noisy image for current type
   for i = 2:rows+1
      for j = 2:cols+1
         subMatrix = current_noise_img(i-1:i+1, j-1:j+1);
         filtered_img1(i-1, j-1) = sum(double(subMatrix(:)) .* double(arith_kernal(:)));
         filtered_img2(i-1, j-1) = nthroot(prod(double(subMatrix(:))), 9);
         filtered_img3(i-1, j-1) = sum(double(subMatrix(:)) .* double(weight_kernal(:)));
      end
   end
```

```
    subplot(3, 3, k); imshow(filtered_img1);
    mse = sum((double(r(:))-double(filtered_img1(:))).^2)/(rows*cols);
    psnr = 10*log(255*255/mse);
    title(sprintf('Arithmetic Filter\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));
    subplot(3, 3, k+3); imshow(filtered_img2);
    mse = sum((double(r(:))-double(filtered_img2(:))).^2)/(rows*cols);
    psnr = 10*log(255*255/mse);
    title(sprintf('Geomatric Filter\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));
    subplot(3, 3, k+6); imshow(filtered_img3);
    mse = sum((double(r(:))-double(filtered_img3(:))).^2)/(rows*cols);
    psnr = 10*log(255*255/mse);
    title(sprintf('weight-avg Filter\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));
end

figure('Name',"5*5 kernal");
sp_pad       = padarray(sp_noise, [2, 2], 0, 'both');
gussian_pad  = padarray(gussian_noise, [2, 2], 0, 'both');
speckle_pad  = padarray(speckle_noise, [2, 2], 0, 'both');
noise_array = {sp_pad, gussian_pad, speckle_pad};

arith_kernal = ones(5, 5) / 25;
weight_kernal = [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
    current_noise_img = noise_array{k}; % Get the padded noisy image for current type
    for i = 3:rows+2
      for j = 3:cols+2
        subMatrix = current_noise_img(i-2:i+2, j-2:j+2);
        filtered_img1(i-2, j-2) = sum(double(subMatrix(:)) .* double(arith_kernal(:)));
        filtered_img2(i-2, j-2) = nthroot(prod(double(subMatrix(:))), 25);
        filtered_img3(i-2, j-2) = sum(double(subMatrix(:)).* double(weight_kernal(:)));
      end
    end
    subplot(3, 3, k); imshow(filtered_img1);
    mse = sum((double(r(:))-double(filtered_img1(:))).^2)/(rows*cols);
    psnr = 10*log(255*255/mse);
    title(sprintf('Arithmetic Filter\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));
    subplot(3, 3, k+3); imshow(filtered_img2);
    mse = sum((double(r(:))-double(filtered_img2(:))).^2)/(rows*cols);
    psnr = 10*log(255*255/mse);
    title(sprintf('Geomatric Filter\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));
    subplot(3, 3, k+6); imshow(filtered_img3);
    mse = sum((double(r(:))-double(filtered_img3(:))).^2)/(rows*cols);
    psnr = 10*log(255*255/mse);
    title(sprintf('weight-avg Filter\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));
end
```

```
r = imread("moon.tif");
[rows, cols] = size(r);
sp_noise      = imnoise(r, 'salt & pepper', 0.1);
gussian_noise  = imnoise(r, 'gaussian');
speckle_noise  = imnoise(r, 'speckle');

figure('Name',"noise Image");
subplot(1,3,1);   imshow(sp_noise);
mse = sum((double(r(:))-double(sp_noise(:))).^2)/(rows*cols);
psnr = 10*log(255*255/mse);
title(sprintf('s&p noise\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));
subplot(1,3,2);   imshow(gussian_noise);
mse = sum((double(r(:))-double(gussian_noise(:))).^2)/(rows*cols);
psnr = 10*log(255*255/mse);
title(sprintf('Gussian noise\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));
subplot(1,3,3);   imshow(speckle_noise);
mse = sum((double(r(:))-double(speckle_noise(:))).^2)/(rows*cols);
psnr = 10*log(255*255/mse);
title(sprintf('Speckle noise Filter\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));

filtered_min = zeros(rows, cols, 'uint8'); %minimum filter
filtered_med = zeros(rows, cols, 'uint8'); %median filter
filtered_max = zeros(rows, cols, 'uint8');  %maximum filter
filtered_trim_4 = zeros(rows, cols, 'uint8'); %4_trimmed filter
filtered_trim_6 = zeros(rows, cols, 'uint8');  % 6_trimmed filter
filtered_mid_point = zeros(rows, cols, 'uint8'); %  mid-point filter

sp_pad      = padarray(sp_noise, [1, 1], 0, 'both');
gussian_pad  = padarray(gussian_noise, [1, 1], 0, 'both');
speckle_pad  = padarray(speckle_noise, [1, 1], 0, 'both');
noise_array = {sp_pad, gussian_pad, speckle_pad};

for k = 1:3
   current_noise_img = noise_array{k}; % Get the padded noisy image for current type
   for i = 2:rows+1
      for j = 2:cols+1
         submatrix = current_noise_img(i-1:i+1, j-1:j+1);
         filtered_min(i-1,j-1) = min(submatrix(:));
         filtered_med(i-1,j-1) = median(submatrix(:));
         filtered_max(i-1,j-1) = max(submatrix(:));
         filtered_mid_point(i-1,j-1)  =  round((double(filtered_min(i-1,j-1)  +filtered_max(i-1,j-
1)))/2);
         vector_A = submatrix(:);
         sort_a = sort(vector_A);
         filtered_trim_4(i-1,j-1) = round((sum(sort_a(3:7))/5));
```

```
            filtered_trim_6(i-1,j-1) = round((sum(sort_a(4:6))/3));
        end
    end
    figure('Name',"for 3*3 kernal");
    subplot(2,3,1); imshow(filtered_min);
    mse = sum((double(r(:))-double(filtered_min(:))).^2)/(rows*cols);
    psnr = 10*log(255*255/mse);
    title(sprintf('minimum Filter\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));
    subplot(2, 3,2); imshow(filtered_med);
    mse = sum((double(r(:))-double(filtered_med(:))).^2)/(rows*cols);
    psnr = 10*log(255*255/mse);
    title(sprintf('median Filter\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));
    subplot(2, 3,3); imshow(filtered_max);
    mse = sum((double(r(:))-double(filtered_max(:))).^2)/(rows*cols);
    psnr = 10*log(255*255/mse);
    title(sprintf('maximum Filter\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));
    subplot(2,3,4); imshow(filtered_mid_point);
    mse = sum((double(r(:))-double(filtered_mid_point(:))).^2)/(rows*cols);
    psnr = 10*log(255*255/mse);
    title(sprintf('mid-pont Filter\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));
    subplot(2, 3, 5); imshow(filtered_trim_4);
    mse = sum((double(r(:))-double(filtered_trim_4(:))).^2)/(rows*cols);
    psnr = 10*log(255*255/mse);
    title(sprintf('4-trim Filter\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));
    subplot(2, 3, 6); imshow(filtered_trim_6);
    mse = sum((double(r(:))-double(filtered_trim_6(:))).^2)/(rows*cols);
    psnr = 10*log(255*255/mse);
    title(sprintf('6-trim Filter\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));

end

filtered_min = zeros(rows, cols, 'uint8');
filtered_med = zeros(rows, cols, 'uint8');
filtered_max = zeros(rows, cols, 'uint8');
filtered_trim_4 = zeros(rows, cols, 'uint8');
filtered_trim_6 = zeros(rows, cols, 'uint8');
filtered_mid_point = zeros(rows, cols, 'uint8');

sp_pad      = padarray(sp_noise, [2, 2], 0, 'both');
gussian_pad  = padarray(gussian_noise, [2, 2], 0, 'both');
speckle_pad  = padarray(speckle_noise, [2, 2], 0, 'both');
noise_array = {sp_pad, gussian_pad, speckle_pad};

for k = 1:3
    current_noise_img = noise_array{k}; % Get the padded noisy image for current type
    for i = 3:rows+2
        for j = 3:cols+2
            submatrix = current_noise_img(i-2:i+2, j-2:j+2);
            filtered_min(i-2,j-2) = min(submatrix(:));    %minimum filter
            filtered_med(i-2,j-2) = median(submatrix(:)); %median filter
            filtered_max(i-2,j-2) = max(submatrix(:));    %maximum filter
            filtered_mid_point(i-2,j-2)  =  round((double(filtered_min(i-2,j-2)  +filtered_max(i-2,j-
2)))/2); %mid point filter
            vector_A = submatrix(:);  sort_a = sort(vector_A);
            filtered_trim_4(i-2,j-2) = round((sum(sort_a(7:19))/13)); % 12_trimmed filter
            filtered_trim_6(i-2,j-2) = round((sum(sort_a(10:16))/7)); % 18_trimmed filter
        end
    end
    figure('Name',"for 5*5 kernal");
    subplot(2,3,1); imshow(filtered_min);
    mse = sum((double(r(:))-double(filtered_min(:))).^2)/(rows*cols);
    psnr = 10*log(255*255/mse);
    title(sprintf('minimum Filter\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));
    subplot(2, 3,2); imshow(filtered_med);
    mse = sum((double(r(:))-double(filtered_med(:))).^2)/(rows*cols);
    psnr = 10*log(255*255/mse);
    title(sprintf('median Filter\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));
    subplot(2, 3,3); imshow(filtered_max);
    mse = sum((double(r(:))-double(filtered_max(:))).^2)/(rows*cols);
    psnr = 10*log(255*255/mse);
    title(sprintf('maximum Filter\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));
    subplot(2,3,4); imshow(filtered_mid_point);
    mse = sum((double(r(:))-double(filtered_mid_point(:))).^2)/(rows*cols);
    psnr = 10*log(255*255/mse);
    title(sprintf('mid-pont Filter\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));
    subplot(2, 3, 5); imshow(filtered_trim_4);
    mse = sum((double(r(:))-double(filtered_trim_4(:))).^2)/(rows*cols);
    psnr = 10*log(255*255/mse);
    title(sprintf('12-trim Filter\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));
    subplot(2, 3, 6); imshow(filtered_trim_6);
    mse = sum((double(r(:))-double(filtered_trim_6(:))).^2)/(rows*cols);
    psnr = 10*log(255*255/mse);
    title(sprintf('18-trim Filter\nMSE = %.4f\n PSNR=%0.4f', mse,psnr));
end
```