安徽大学《数字图像处理（双语）》实验报告（5）

学号：WA2214014 专业：人工智能 姓名及签字：杨跃浙

实验日期：2024 年 12 月 26 日 实验成绩： 教师签字：

【实验名称】： 基于空间滤波的图像复原和彩色模型

【实验目的】：

1. 熟悉和掌握如何为灰度图像添加噪声

2. 通过 MATLAB 编程实现基于均值和中值滤波的图像去噪

3. 通过 MATLAB 编程实现基于自适应中值滤波的图像去噪，并通过理论和实验分析掌握其与传统的中值滤波相比的优势

4. 通过 MATLAB 编程实现彩色图像和灰度图像的转换以及在不同颜色空间编辑图像的技术

【实验内容】

PROJECT 05-01

Noise Generators

See Fig. 5.2 for the shapes and parameters of the following noise probability density functions.

(a) Using function "imnoise" to add Gaussian noise to an image. You must be able to specify the noise mean and variance.

(b) Using function "imnoise" to add salt-and-pepper (impulse) noise to an image. You must be able to specify the probabilities of each of the two noise components.

PROJECT 05-02

Noise Reduction Using mean Filters and median filters

(a) Add salt-and-pepper noise to 'ckt-board-orig.tif', with $P_a = P_b = 0.1$ and show the noised image

(b) Program a 5 x 5 median filtering on (a) and demonstrate the restored result

(c) Further add Gaussian noise with specified mean and variance onto (a)

(d) Apply 5 x 5 Arithmetic Mean Filter, Geometric Mean Filter, Median Filter and Alpha-trimmed mean filter (adjusting the parameter d for your filter) on (c) respectively. Explain differences between your results

(e) Using the image difference and histogram technology to access the the performance of Alpha-trimmed mean filter and the Median Filter during the step (d)

PROJECT 05-03

Noise Reduction adaptive median filters

(a) Add salt-and-pepper noise to 'ckt-board-orig.tif', with $P_a = P_b = 0.25$ and show the noised image

(b) Apply 7 x 7 median filtering on (a) and show the processed image

(c) Design an adaptive median filter with maximum size of the subwindow as 7 x 7 on (a) and show the processed image

(d) Explain differences between your results.

PROJECT 06-01

Convert the RGB color image into gray image

(a) Read the color image" top_ left_flower.tif" in Matlab, you will obtain a 3-D array which denote the Red, Green and Blue component respectively.

(b) Use the equation I=(R+G+B)/3 to calculate the intensity. This is actually a simple way to convert the RGB color image into a gray image.

(c) Display the gray image.

(d) Adjust the proportion of R, G and B and re-calculate intensity I. Comparing with the result from (b) and discover the best proportions.

PROJECT 06-02

Intensity modification in color images

Now we wish to modify the intensity of the color image "Fig_strawberries" by suppressing it with a scalar of k=0.7 (which means I = 0.7*I) in both RGB color model and CMY color model

(a) Figure out how to suppress the intensity of the color image in RGB model according to PROJECT 06-01 and demonstrate the result

(b) Figure out how to suppress the intensity of the color image in CMY model according to the relationship between RGB model and CMY model, and demonstrate the result

【实验代码和结果】

Project05-01：

Code:

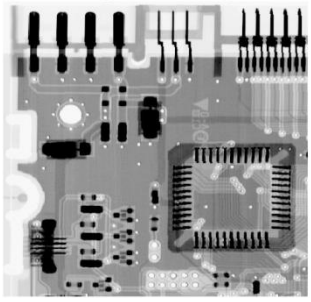
Project1.m:

```
img =
imread('/Users/youngbean/Documents/Github/Misc-Projects/D
igital Image
Processing/Class5/IMAGES/ckt-board-orig.tif');

mean = 0;
variance = 0.01;
noisy_gaussian = imnoise(img, 'gaussian', mean, variance);

salt_pepper_density = 0.05; % Density of salt-and-pepper
noise (proportion of noisy pixels)
noisy_salt_pepper = imnoise(img, 'salt & pepper',
salt_pepper_density);

output_dir =
'/Users/youngbean/Documents/Github/Misc-Projects/Digital
Image Processing/Class5/Figures/';


figure;
imshow(img);
title('Original Image');
```

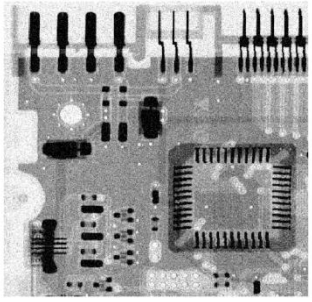
```
exportgraphics(gcf, fullfile(output_dir,
'original_image.png'), 'Resolution', 300);
close;


figure;
imshow(noisy_gaussian);
title('Gaussian Noise');
exportgraphics(gcf, fullfile(output_dir,
'noisy_gaussian.png'), 'Resolution', 300);
close;

figure;
imshow(noisy_salt_pepper);
title('Salt-and-Pepper Noise');
exportgraphics(gcf, fullfile(output_dir,
'noisy_salt_pepper.png'), 'Resolution', 300);
close;


figure;
subplot(1, 3, 1);
imshow(img);
title('Original Image');

subplot(1, 3, 2);
imshow(noisy_gaussian);
title('Gaussian Noise');

subplot(1, 3, 3);
imshow(noisy_salt_pepper);
title('Salt-and-Pepper Noise');

exportgraphics(gcf, fullfile(output_dir,
'all_images_displayed.png'), 'Resolution', 300);
```

Result:

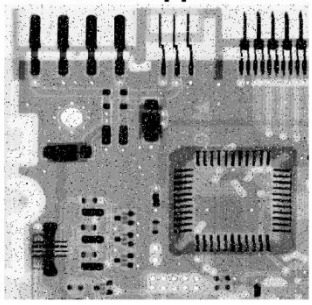
Original Image



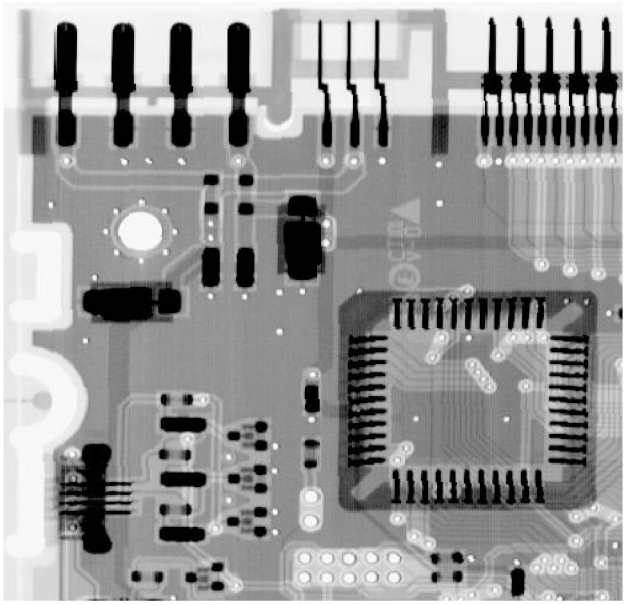
Gaussian Noise



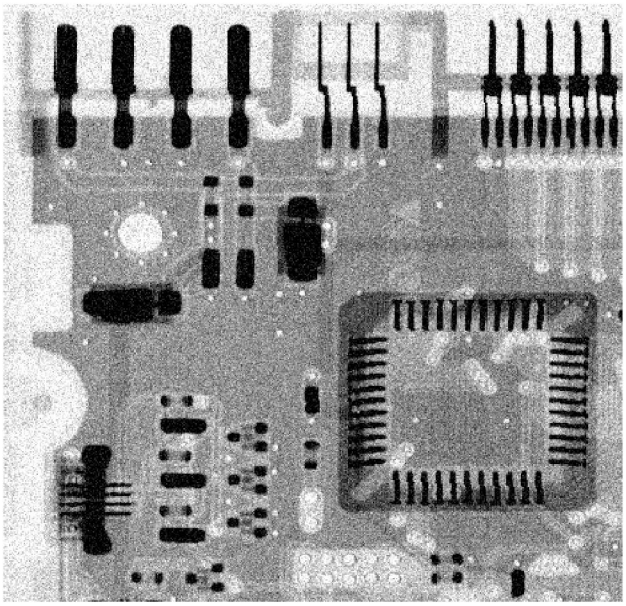
Salt-and-Pepper Noise

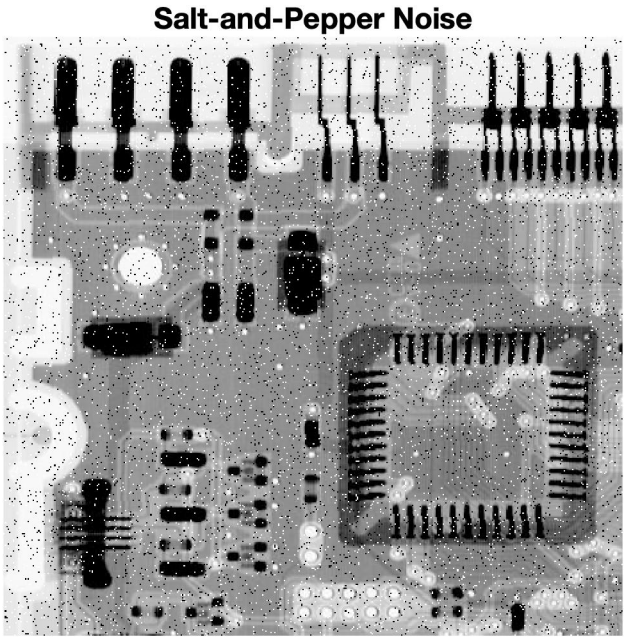


Original Image



Gaussian Noise





Project05-02:

Code:

alpha_trimmed_filter.m:

```
function output = alpha_trimmed_filter(img, window_size, d)
[rows, cols] = size(img);
pad_size = floor(window_size / 2); % Padding size for borders
padded_img = padarray(double(img), pad_size,
'symmetric'); % Symmetric padding
output = zeros(size(img)); % Initialize output

% Ensure d is valid
total_window_size = prod(window_size);
if d >= total_window_size
error('Trim value d must be less than the total number of
elements in the window');
end

% Process each pixel
for i = 1:rows
for j = 1:cols
% Extract the local window
```

```
window = padded_img(i:i+window_size(1)-1,
j:j+window_size(2)-1);
sorted_window = sort(window(:)); % Sort pixel values in the
window
trimmed_window = sorted_window((d/2)+1:end-(d/2)); % Trim
d/2 values from both ends
output(i, j) = mean(trimmed_window); % Calculate the mean of
the trimmed window
end
end

output = uint8(output); % Convert back to uint8 format
end
```

Project2.m:

```
img =
imread('/Users/youngbean/Documents/Github/Misc-Projects/D
igital Image
Processing/Class5/IMAGES/ckt-board-orig.tif');
output_dir =
'/Users/youngbean/Documents/Github/Misc-Projects/Digital
Image Processing/Class5/Figures/';

d_values = [2, 6, 10, 14, 20];
results = {
'Original Image', img;
'Salt & Pepper Noise', imnoise(img, 'salt & pepper', 0.1);
'Median Filter (Salt & Pepper)', medfilt2(imnoise(img, 'salt
& pepper', 0.1), [5, 5]);
'Salt & Pepper + Gaussian Noise', imnoise(imnoise(img, 'salt
& pepper', 0.1), 'gaussian', 0, 0.01);
};

h = fspecial('average', [5, 5]);
results{end+1, 1} = 'Arithmetic Mean Filter';
results{end, 2} = imfilter(results{4, 2}, h, 'replicate');

% Add Geometric Mean Filter
```

```
geo_filtered = exp(imfilter(log(double(results{4, 2}) + 1),
ones(5, 5), 'replicate')) .^ (1 / 25);
results{end+1, 1} = 'Geometric Mean Filter';
results{end, 2} = uint8(geo_filtered - 1);

% Add Median Filter (Gaussian)
results{end+1, 1} = 'Median Filter (Gaussian)';
results{end, 2} = medfilt2(results{4, 2}, [5, 5]);

% Add Alpha-Trimmed Filters for Different d Values
for i = 1:numel(d_values)
d_current = d_values(i);
results{end+1, 1} = sprintf('Alpha-Trimmed Mean Filter (d
= %d)', d_current);
results{end, 2} = alpha_trimmed_filter(results{4, 2}, [5, 5],
d_current);
end


for i = 1:size(results, 1)
image_name = results{i, 1};
current_image = results{i, 2};
% Save image with title
figure;
imshow(current_image, []);
title(image_name, 'Interpreter', 'none');
exportgraphics(gcf, fullfile(output_dir,
sprintf('%s_image.png', strrep(image_name, ' ', '_'))),
'Resolution', 300);
close;

% Save histogram with title
figure;
imhist(current_image);
title(sprintf('Histogram of %s', image_name), 'Interpreter',
'none');
exportgraphics(gcf, fullfile(output_dir,
sprintf('%s_histogram.png', strrep(image_name, ' ', '_'))),
'Resolution', 300);
```

```
close;
end

num_images = size(results, 1);
rows = ceil(num_images / 4); % Adjust rows for a max of 4 columns

% Figure for all images
figure('Name', 'All Processed Images', 'NumberTitle',
'off');
set(gcf, 'Position', [100, 100, 1800, 1200]); % Adjust figure
size

for i = 1:num_images
subplot(rows, 4, i);
imshow(results{i, 2}, []);
title(results{i, 1}, 'Interpreter', 'none');
end

exportgraphics(gcf, fullfile(output_dir,
'all_images_displayed.png'), 'Resolution', 300);

figure('Name', 'All Histograms', 'NumberTitle', 'off');
set(gcf, 'Position', [100, 100, 1800, 1200]); % Adjust figure
size

for i = 1:num_images
subplot(rows, 4, i);
imhist(results{i, 2});
title(sprintf('Histogram of %s', results{i, 1}),
'Interpreter', 'none');
end

exportgraphics(gcf, fullfile(output_dir,
'all_histogram_displayed.png'), 'Resolution', 300);

% Extract original image
original_image = results{1, 2}; % Original image

% Collect images for comparison
```

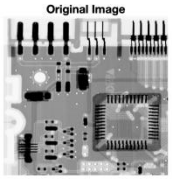
```
comparison_images = { ...
'Median Filter (Gaussian)', results{7, 2}; % Median Filter
sprintf('Alpha-Trimmed (d = %d)', d_values(1)), results{8,
2};
sprintf('Alpha-Trimmed (d = %d)', d_values(2)), results{9,
2};
sprintf('Alpha-Trimmed (d = %d)', d_values(3)), results{10,
2};
sprintf('Alpha-Trimmed (d = %d)', d_values(4)), results{11,
2};
sprintf('Alpha-Trimmed (d = %d)', d_values(5)), results{12,
2};
};

% Compute differences with the original image
diff_images = cell(size(comparison_images, 1), 1);
for i = 1:size(comparison_images, 1)
diff_images{i} = imabsdiff(original_image,
comparison_images{i, 2});
end

figure('Name', 'Differences: Filters vs Original',
'NumberTitle', 'off');
set(gcf, 'Position', [100, 100, 1800, 800]); % Adjust figure
size

rows = 2;
cols = 3;

for i = 1:size(diff_images, 1)
subplot(rows, cols, i);
imshow(diff_images{i}, []);
title(sprintf('Difference: %s', comparison_images{i, 1}),
'Interpreter', 'none');
end

% Save the combined difference image
exportgraphics(gcf, fullfile(output_dir,
'all_difference_images.png'), 'Resolution', 300);
```

```
close;

% Save individual difference images
for i = 1:size(diff_images, 1)
figure;
imshow(diff_images{i}, []);
title(sprintf('Difference: %s', comparison_images{i, 1}),
'Interpreter', 'none');
exportgraphics(gcf, fullfile(output_dir,
sprintf('difference_%s.png', strrep(comparison_images{i,
1}, ' ', '_'))), 'Resolution', 300);
close;
end

original_image = results{1, 2};
original_histogram = imhist(original_image);

% Collect histograms for comparison
comparison_images = { ...
'Median Filter (Gaussian)', results{7, 2}; % Median Filter
sprintf('Alpha-Trimmed (d = %d)', d_values(1)), results{8,
2};
sprintf('Alpha-Trimmed (d = %d)', d_values(2)), results{9,
2};
sprintf('Alpha-Trimmed (d = %d)', d_values(3)), results{10,
2};
sprintf('Alpha-Trimmed (d = %d)', d_values(4)), results{11,
2};
sprintf('Alpha-Trimmed (d = %d)', d_values(5)), results{12,
2};
};

% Compute histogram differences
histogram_differences = cell(size(comparison_images, 1),
1);
for i = 1:size(comparison_images, 1)
comparison_histogram = imhist(comparison_images{i, 2});
histogram_differences{i} = (original_histogram -
comparison_histogram);
```

```
end

figure('Name', 'Histogram Differences: Filters vs Original',
'NumberTitle', 'off');
set(gcf, 'Position', [100, 100, 1800, 800]); % Adjust figure
size

rows = 2;
cols = 3;

for i = 1:size(histogram_differences, 1)
subplot(rows, cols, i);
bar(histogram_differences{i}, 'k'); % Use a bar plot to show
histogram difference
title(sprintf('Histogram Difference: %s',
comparison_images{i, 1}), 'Interpreter', 'none');
xlabel('Pixel Intensity');
ylabel('Difference');
end

exportgraphics(gcf, fullfile(output_dir,
'all_histogram_differences.png'), 'Resolution', 300);
close;

for i = 1:size(histogram_differences, 1)
figure;
bar(histogram_differences{i}, 'k');
title(sprintf('Histogram Difference: %s',
comparison_images{i, 1}), 'Interpreter', 'none');
xlabel('Pixel Intensity');
ylabel('Difference');
exportgraphics(gcf, fullfile(output_dir,
sprintf('histogram_difference_%s.png',
strrep(comparison_images{i, 1}, ' ', '_'))), 'Resolution',
300);
close;
end
```

Result:

Original Image

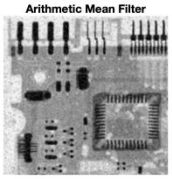


Salt & Pepper Noise

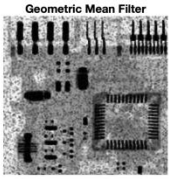
Median Filter (Salt & Pepper)

Salt & Pepper + Gaussian Noise

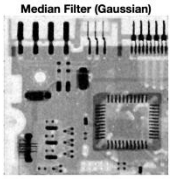
Arithmetic Mean Filter



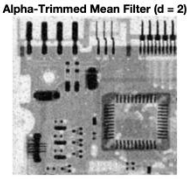
Geometric Mean Filter



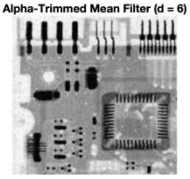
Median Filter (Gaussian)



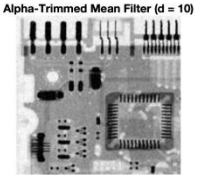
Alpha-Trimmed Mean Filter (d = 2)



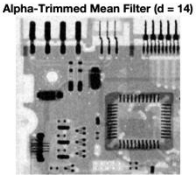
Alpha-Trimmed Mean Filter (d = 6)



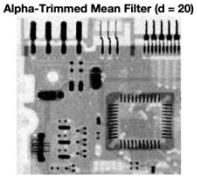
Alpha-Trimmed Mean Filter (d = 10)



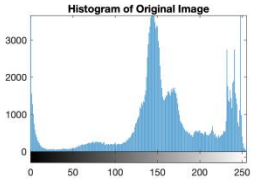
Alpha-Trimmed Mean Filter (d = 14)



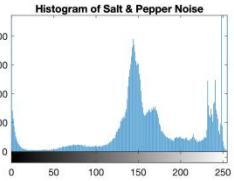
Alpha-Trimmed Mean Filter (d = 20)



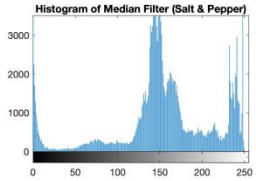
Histogram of Original Image



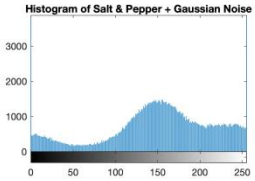
Histogram of Salt & Pepper Noise



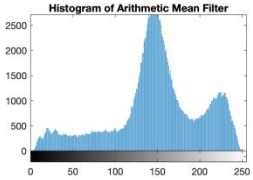
Histogram of Median Filter (Salt & Pepper)



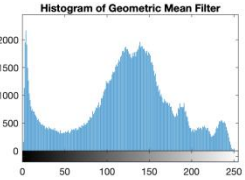
Histogram of Salt & Pepper + Gaussian Noise



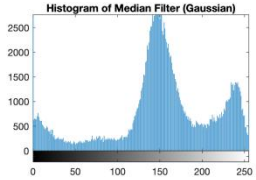
Histogram of Arithmetic Mean Filter



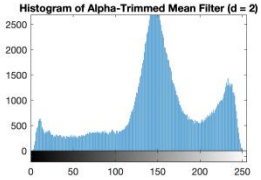
Histogram of Geometric Mean Filter



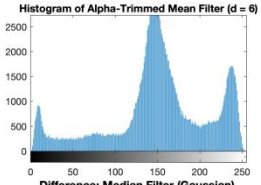
Histogram of Median Filter (Gaussian)



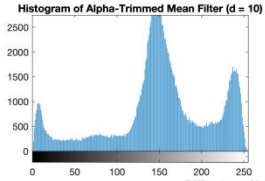
Histogram of Alpha-Trimmed Mean Filter (d = 2)



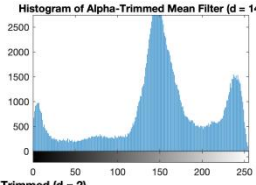
Histogram of Alpha-Trimmed Mean Filter (d = 6)



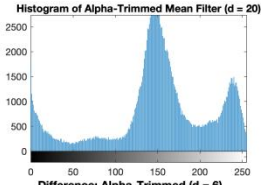
Histogram of Alpha-Trimmed Mean Filter (d = 10)



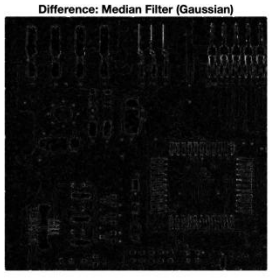
Histogram of Alpha-Trimmed Mean Filter (d = 14)



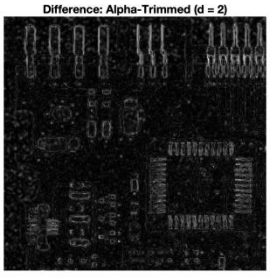
Histogram of Alpha-Trimmed Mean Filter (d = 20)



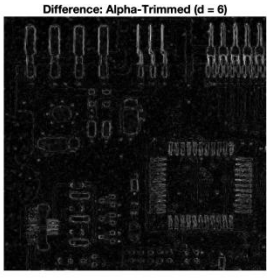
Difference: Median Filter (Gaussian)



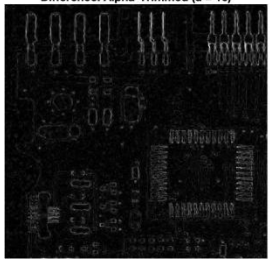
Difference: Alpha-Trimmed (d = 2)



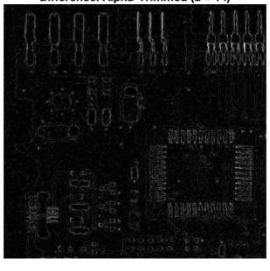
Difference: Alpha-Trimmed (d = 6)



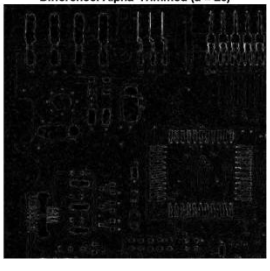
Difference: Alpha-Trimmed (d = 10)

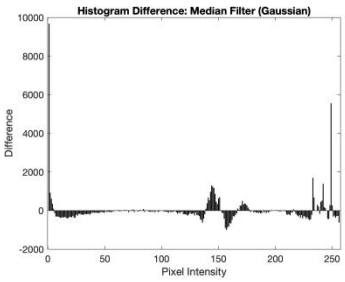


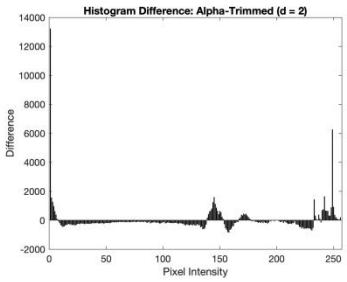
Difference: Alpha-Trimmed (d = 14)

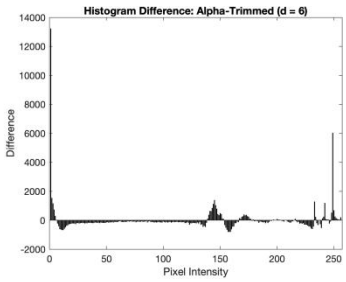


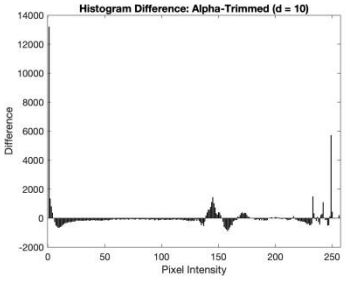
Difference: Alpha-Trimmed (d = 20)

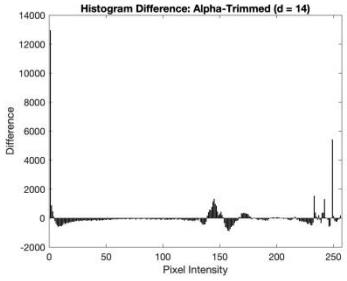


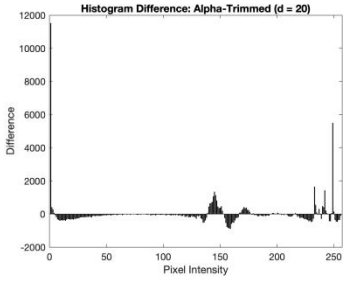




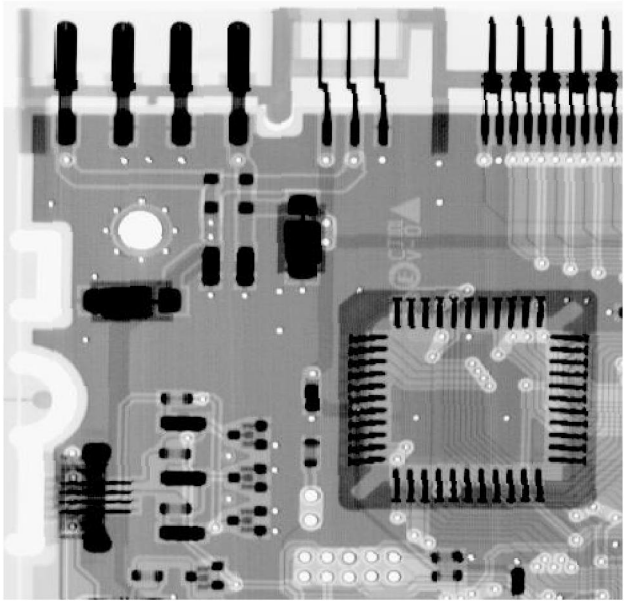




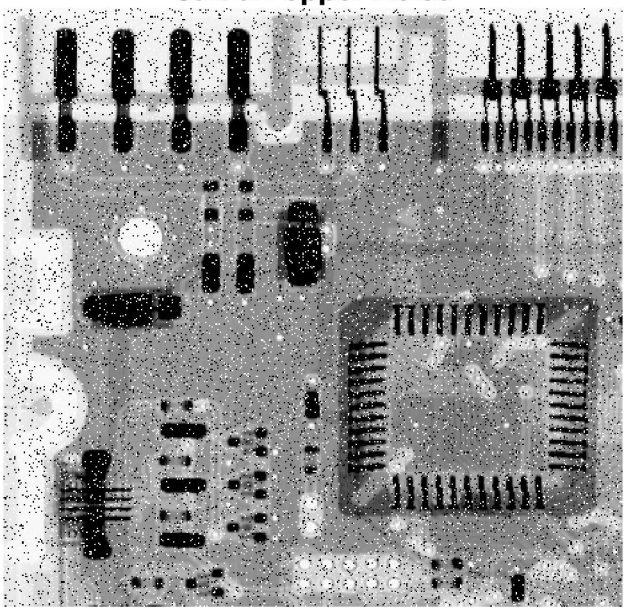




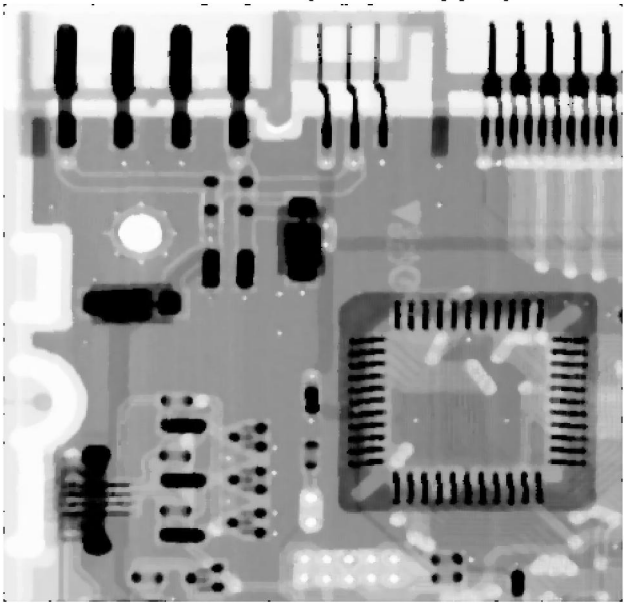
Original Image



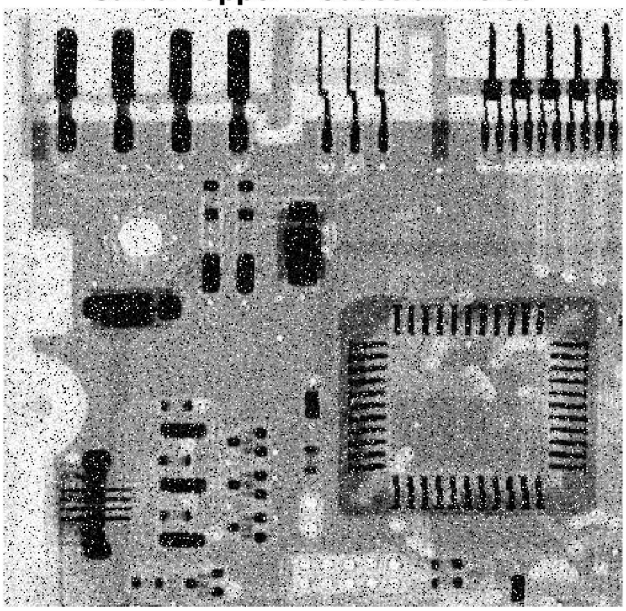
Salt & Pepper Noise



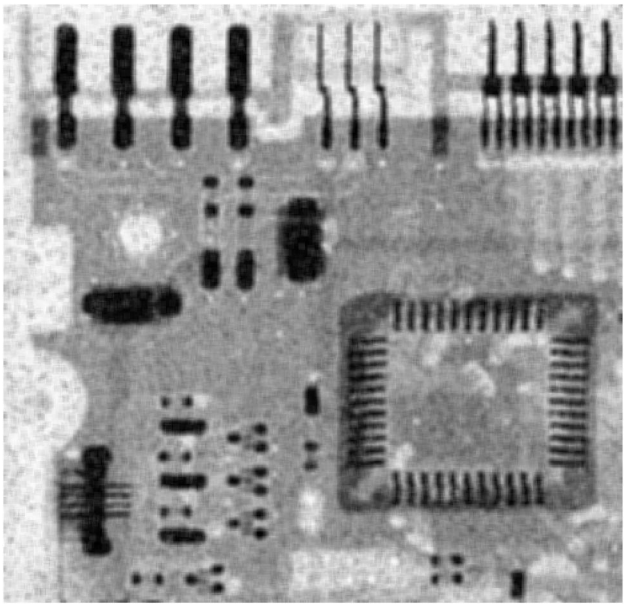
Median Filter (Salt & Pepper)



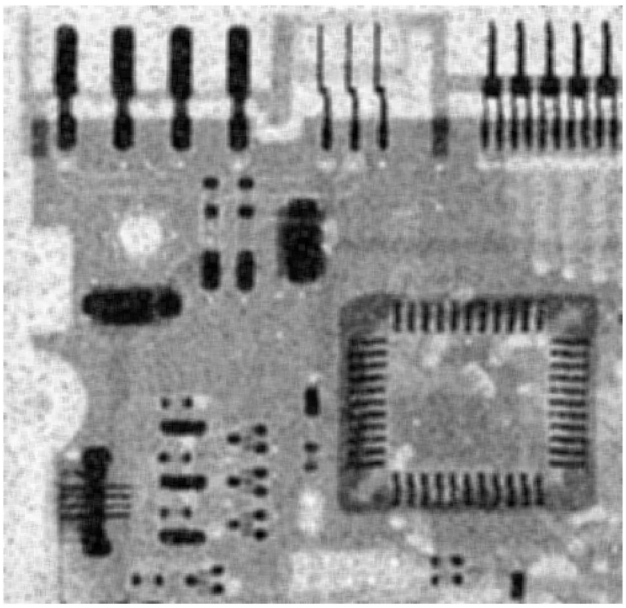
Salt & Pepper + Gaussian Noise



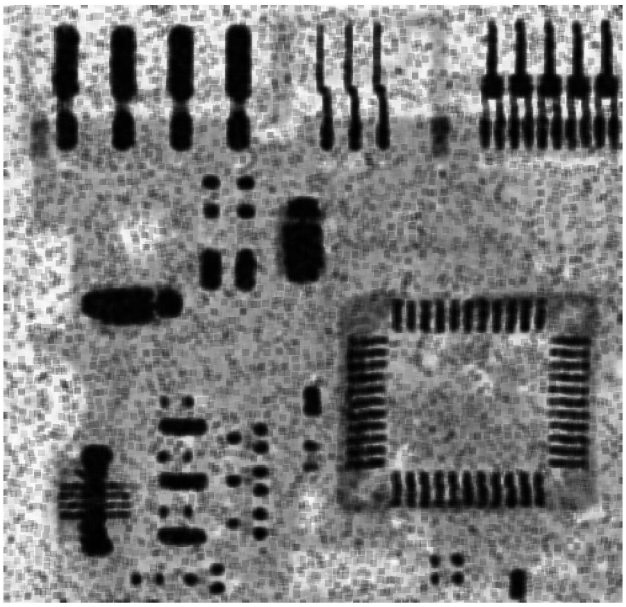
Arithmetic Mean Filter



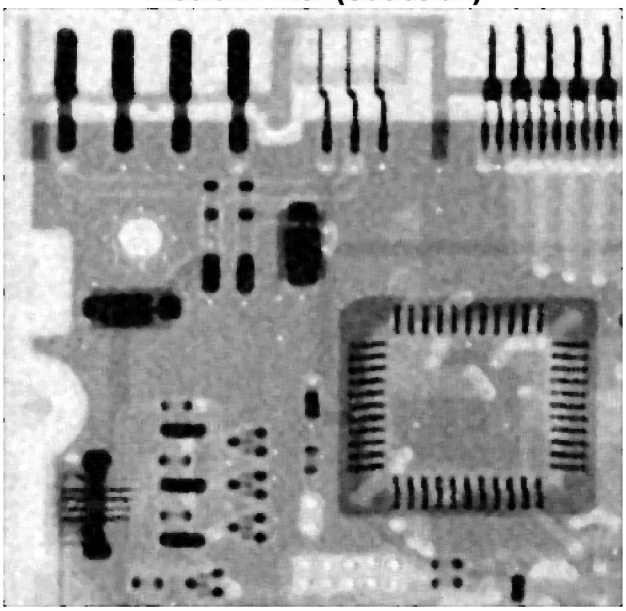
Arithmetic Mean Filter



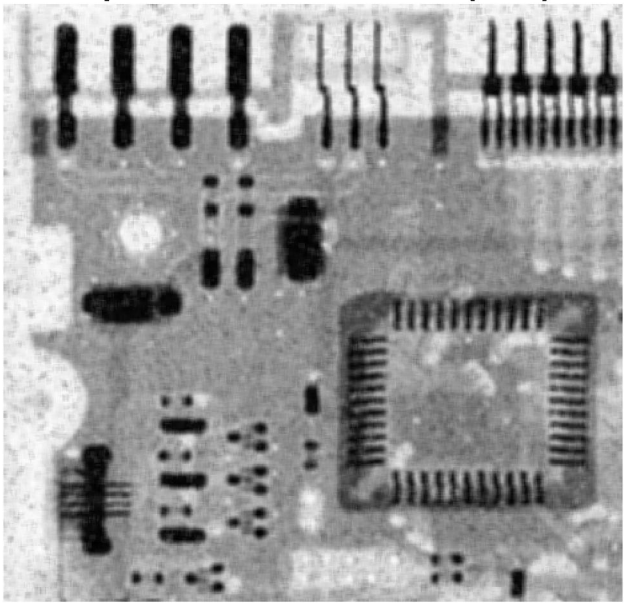
Geometric Mean Filter



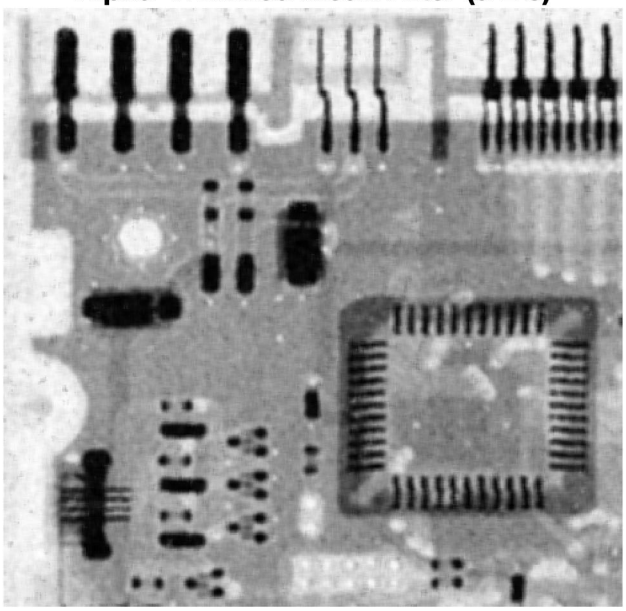
Median Filter (Gaussian)



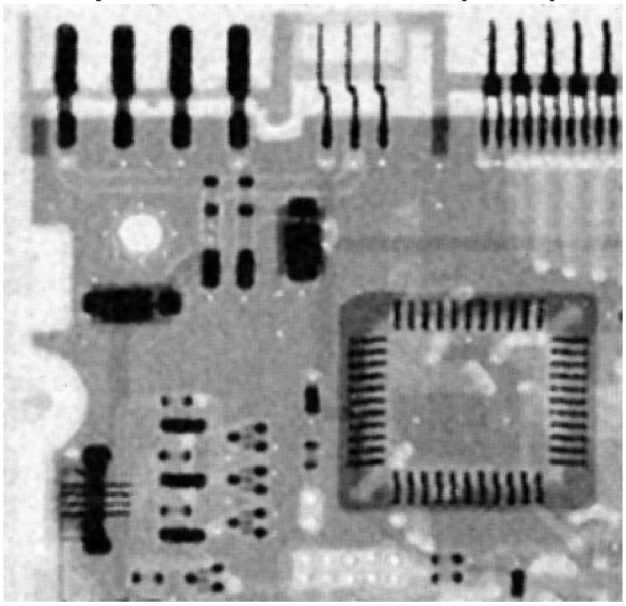
Alpha-Trimmed Mean Filter (d = 2)



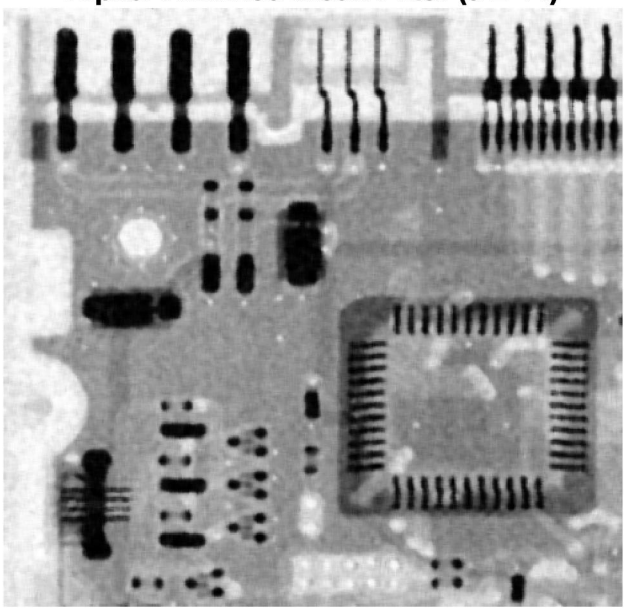
Alpha-Trimmed Mean Filter (d = 6)



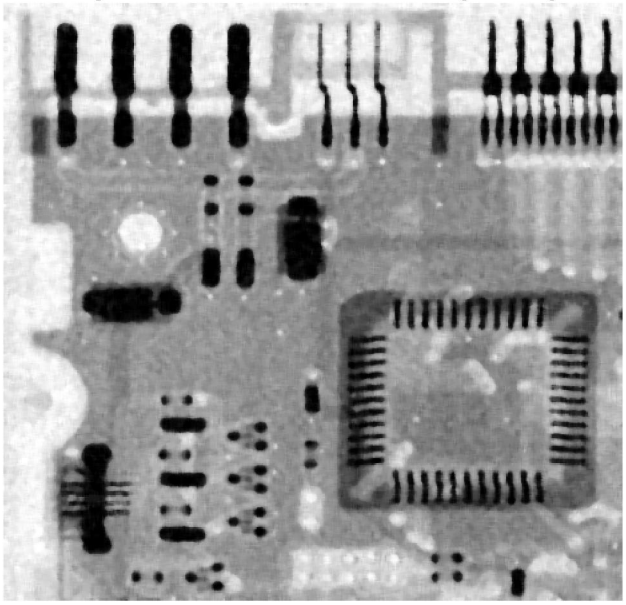
Alpha-Trimmed Mean Filter (d = 10)

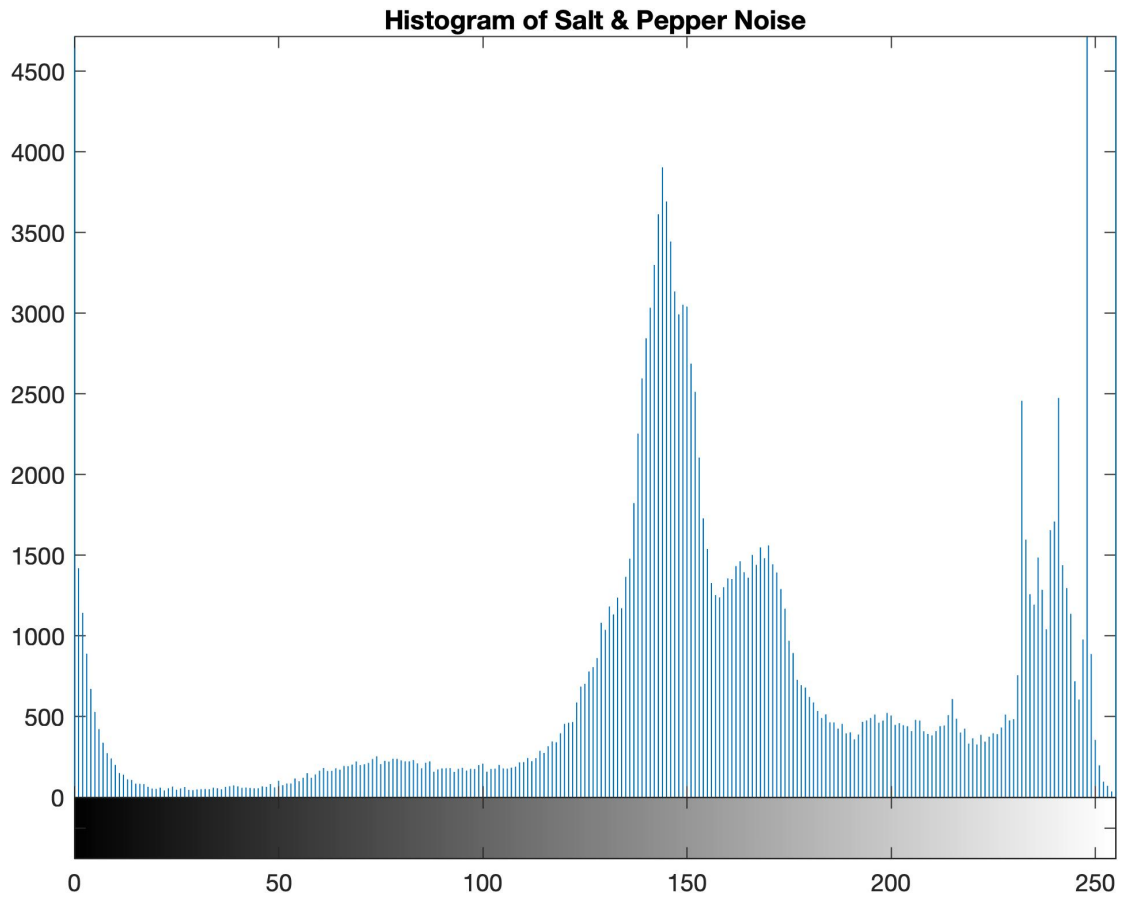


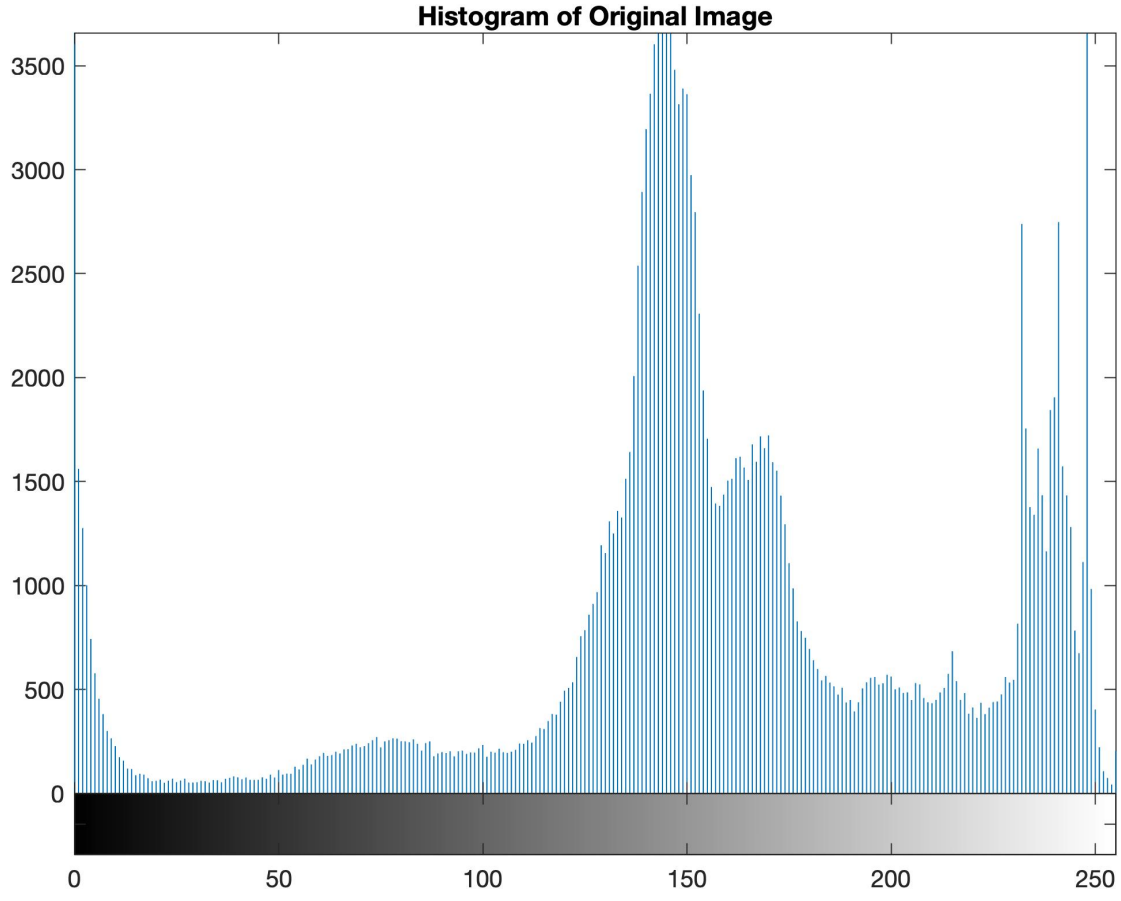
Alpha-Trimmed Mean Filter (d = 14)

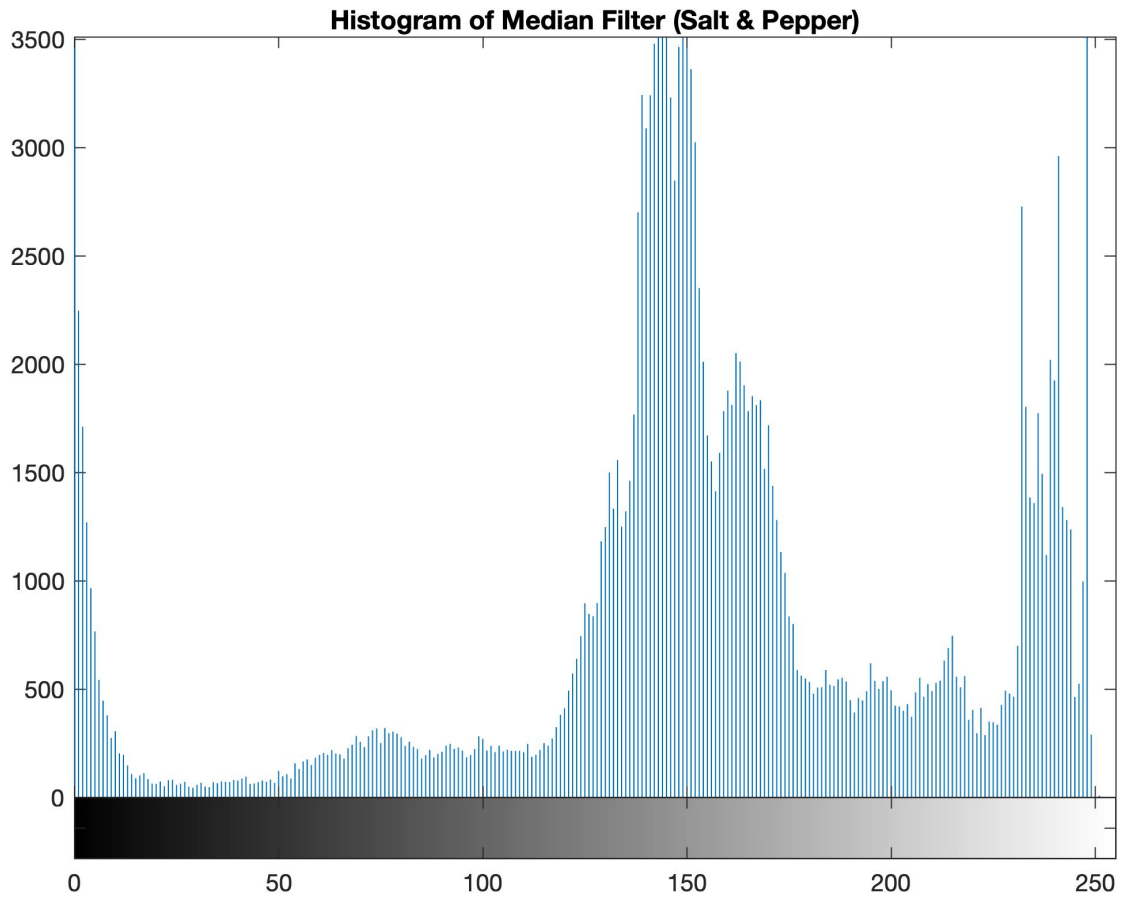


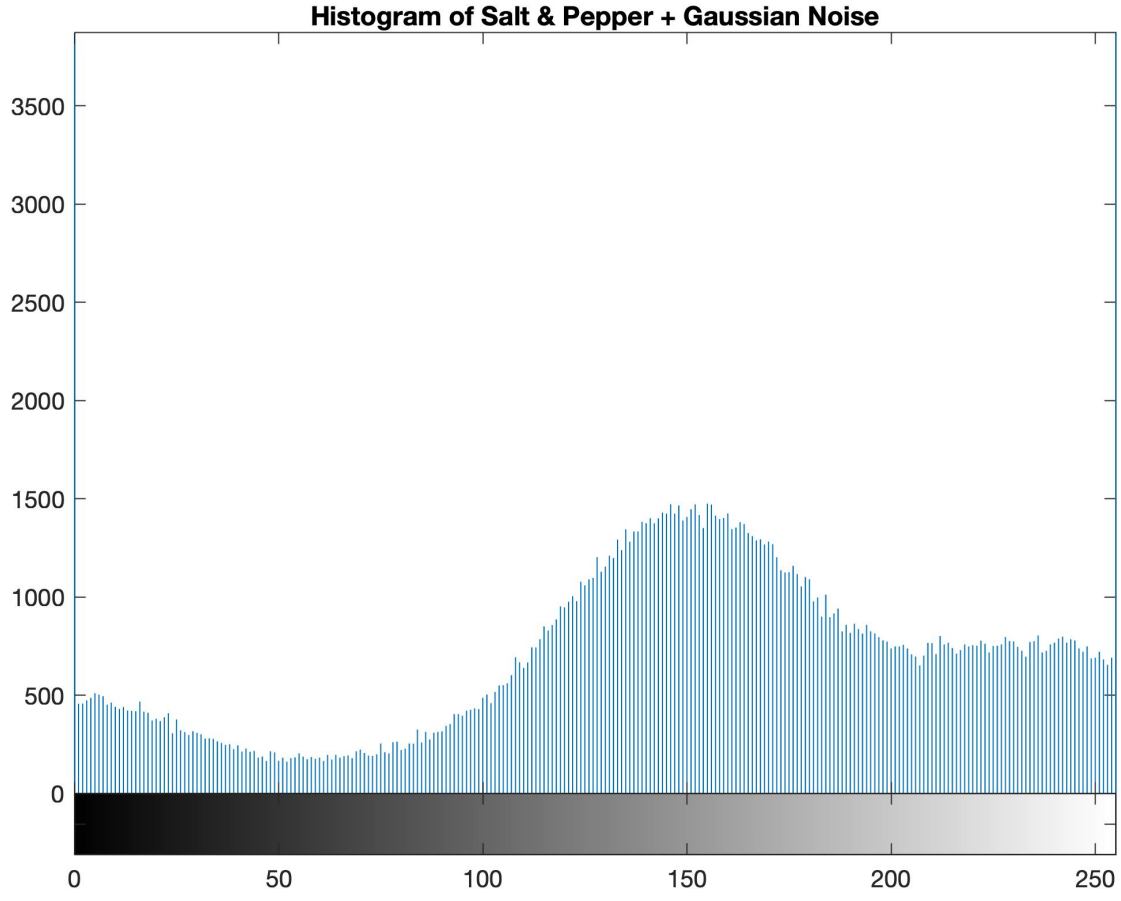
Alpha-Trimmed Mean Filter (d = 20)

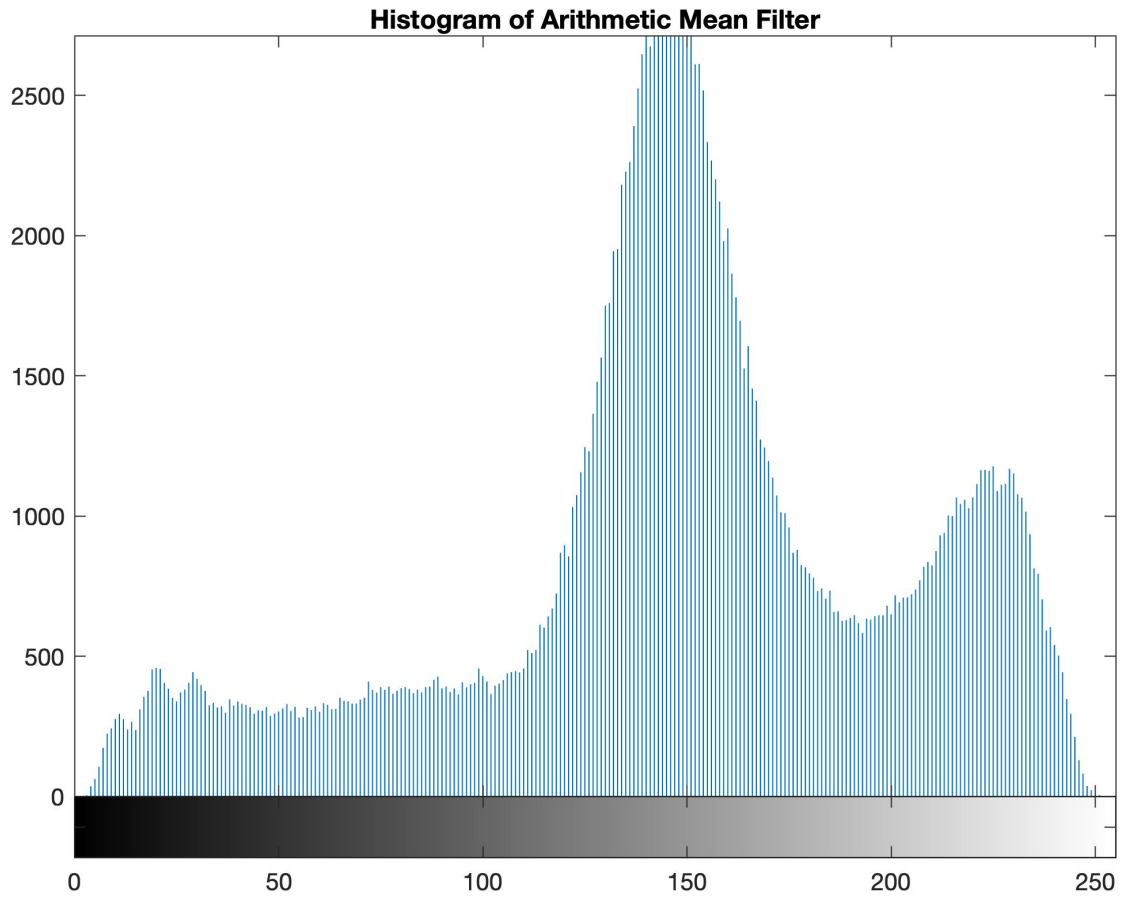


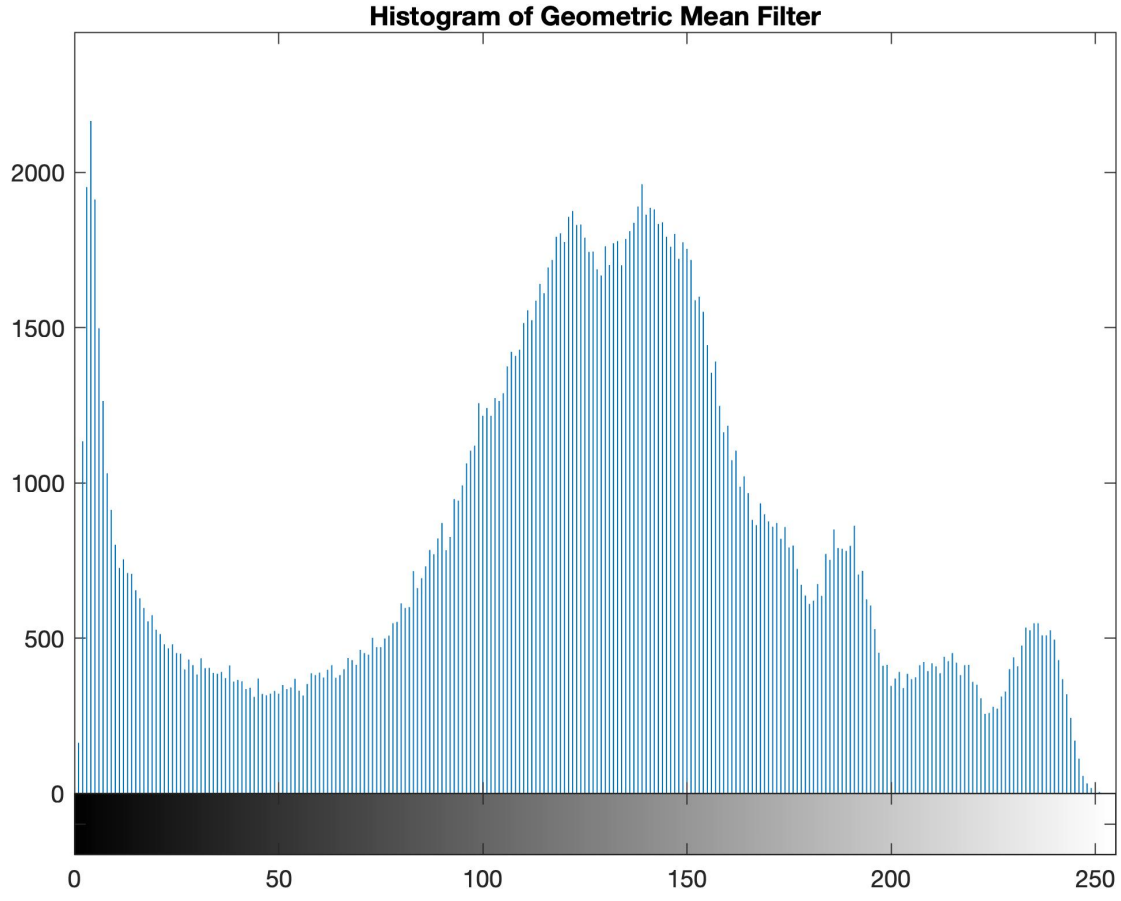

Histogram of Salt & Pepper Noise
4500
4000
3500
3000
2500
2000
1500
1000
500
0
0
50
100
150
200
250

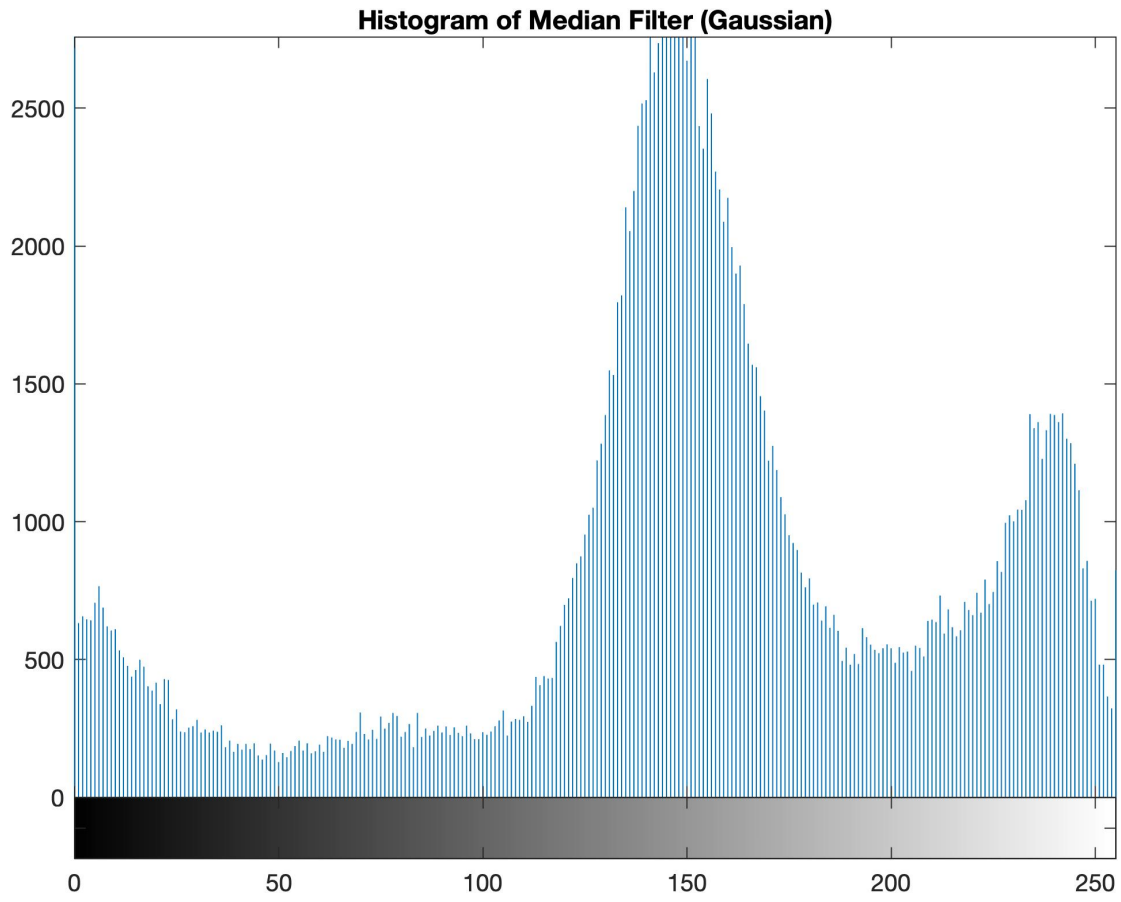

Histogram of Original Image
3500
3000
2500
2000
1500
1000
500
0
0
50
100
150
200
250

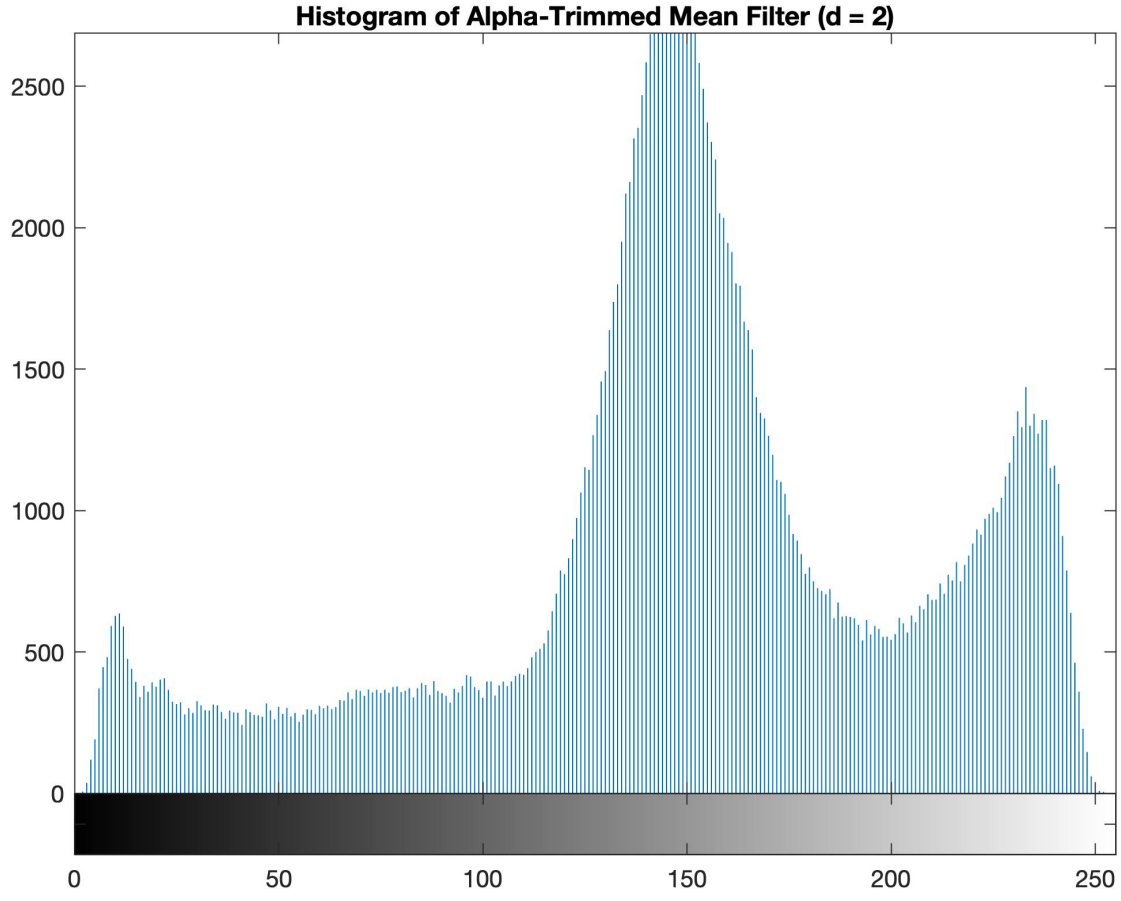

Histogram of Median Filter (Salt & Pepper)
3500
3000
2500
2000
1500
1000
500
0
0
50
100
150
200
250

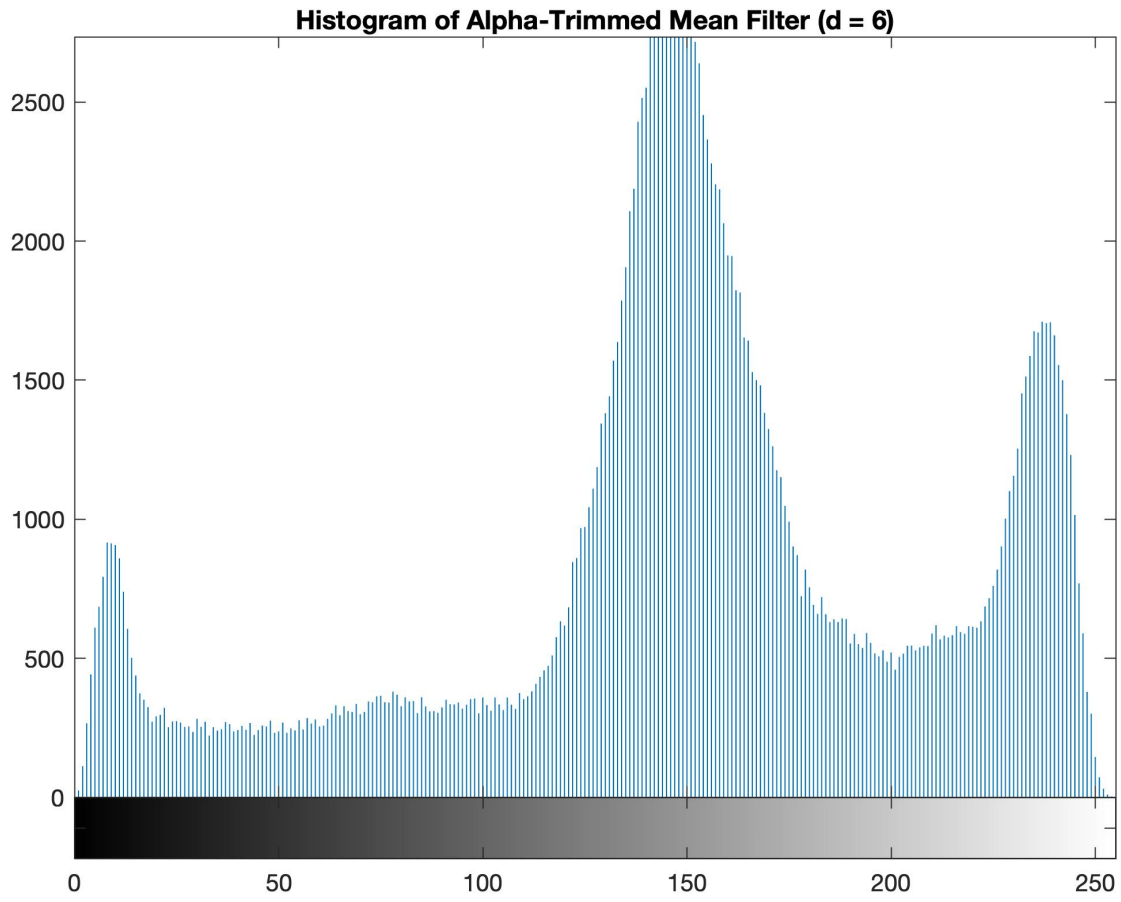

Histogram of Salt & Pepper + Gaussian Noise
3500
3000
2500
2000
1500
1000
500
0
0
50
100
150
200
250

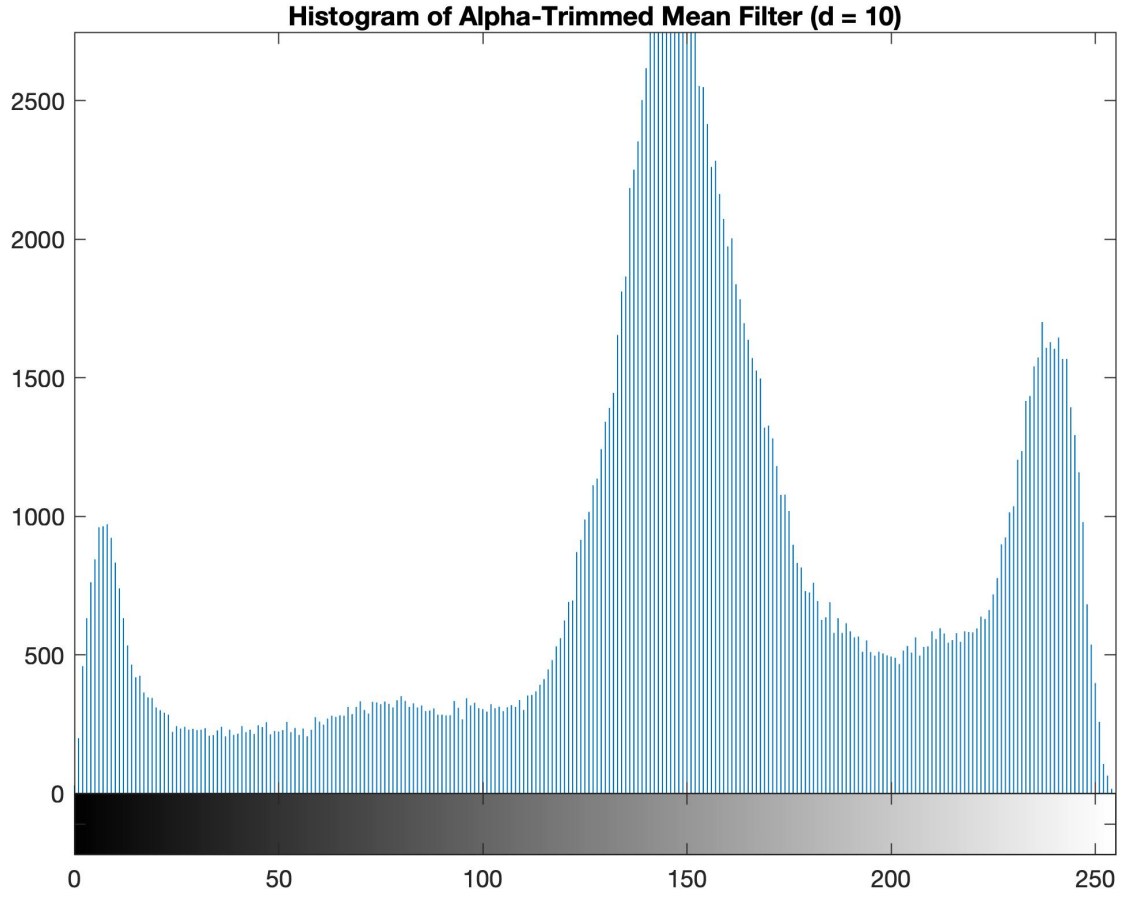

Histogram of Arithmetic Mean Filter
2500
2000
1500
1000
500
0
0
50
100
150
200
250

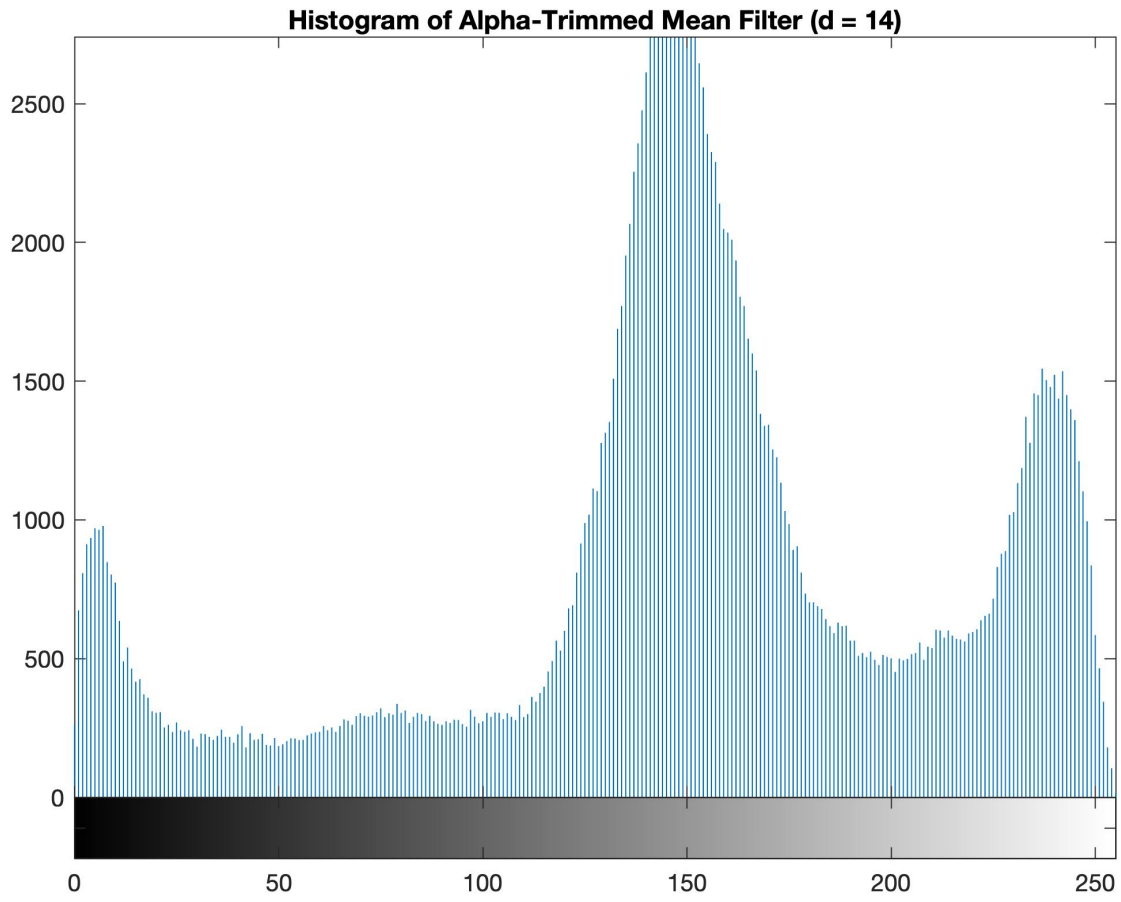

Histogram of Geometric Mean Filter
2000
1500
1000
500
0
0
50
100
150
200
250

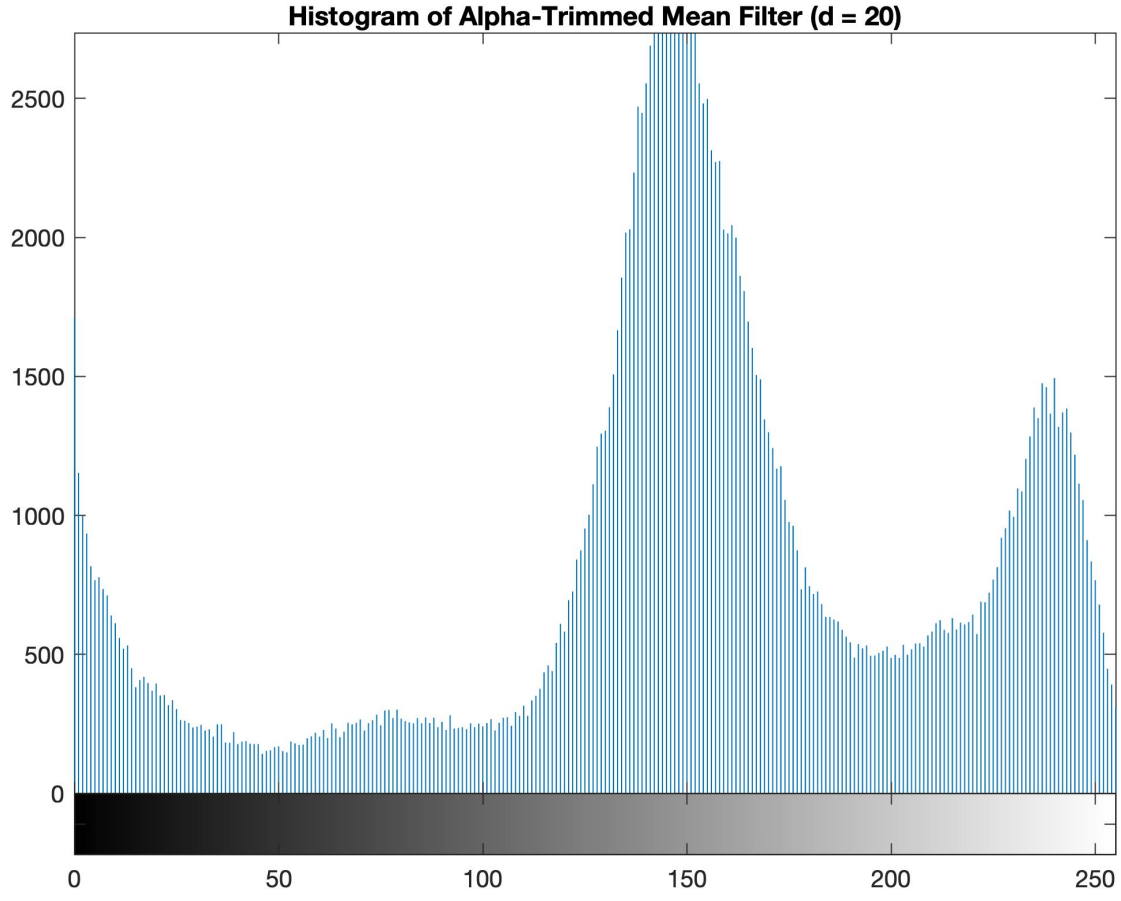

Histogram of Median Filter (Gaussian)
2500
2000
1500
1000
500
0
0
50
100
150
200
250


Histogram of Alpha-Trimmed Mean Filter (d = 2)
2500
2000
1500
1000
500
0
0
50
100
150
200
250

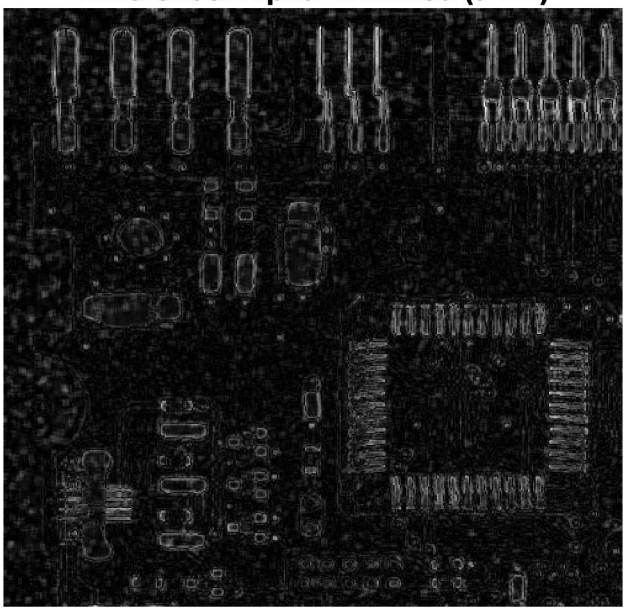

Histogram of Alpha-Trimmed Mean Filter (d = 6)
2500
2000
1500
1000
500
0
0
50
100
150
200
250


Histogram of Alpha-Trimmed Mean Filter (d = 10)
2500
2000
1500
1000
500
0
0
50
100
150
200
250

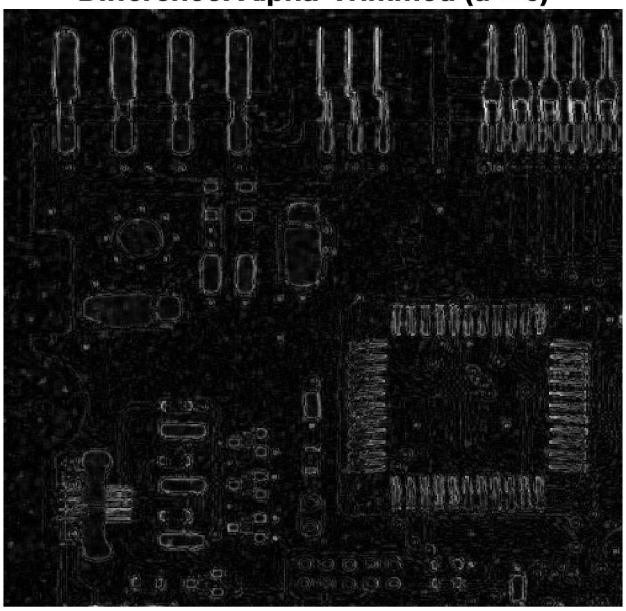

Histogram of Alpha-Trimmed Mean Filter (d = 14)
2500
2000
1500
1000
500
0
0
50
100
150
200
250


Histogram of Alpha-Trimmed Mean Filter (d = 20)
2500
2000
1500
1000
500
0
0
50
100
150
200
250

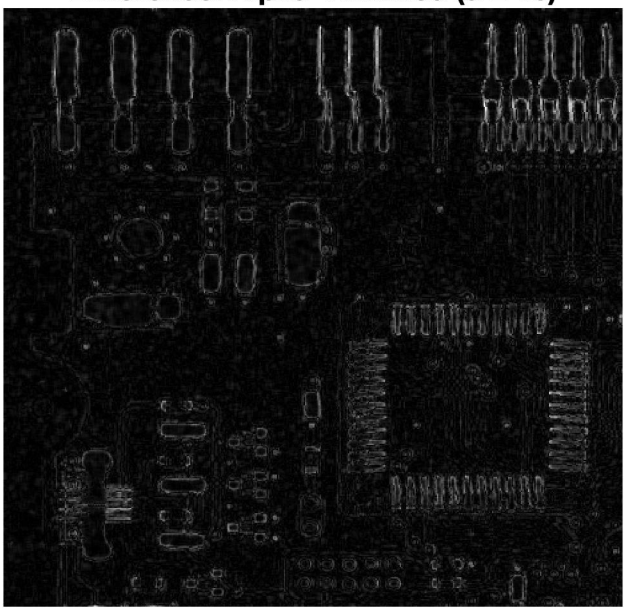
Difference: Alpha-Trimmed (d = 2)



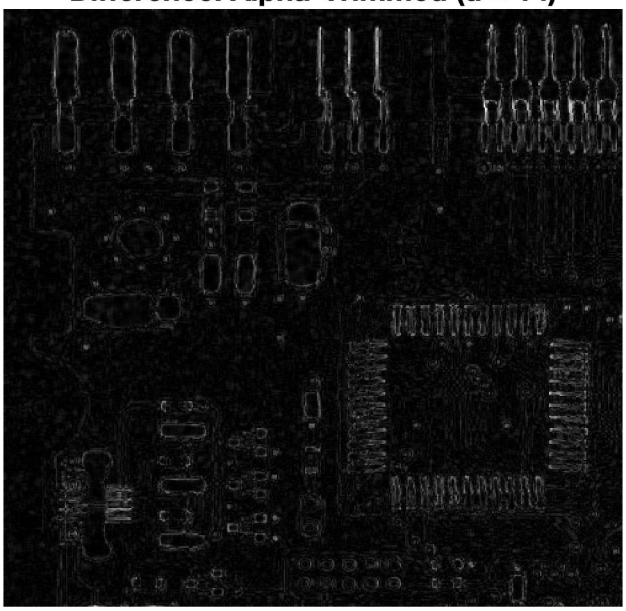
Difference: Alpha-Trimmed (d = 6)



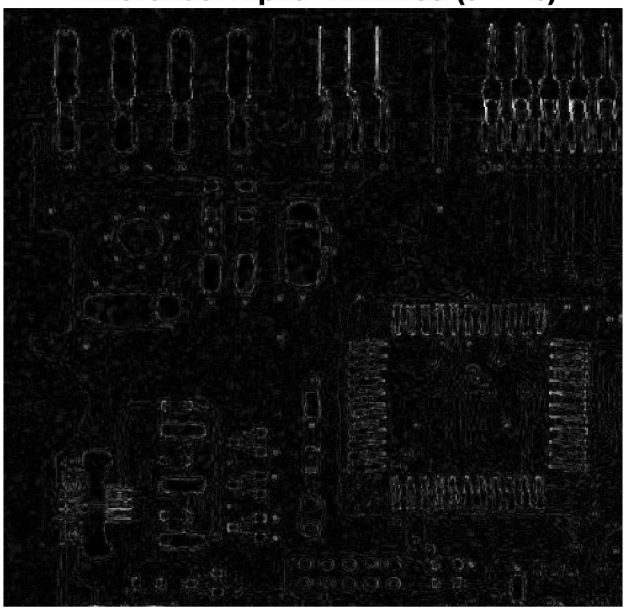
Difference: Alpha-Trimmed (d = 10)



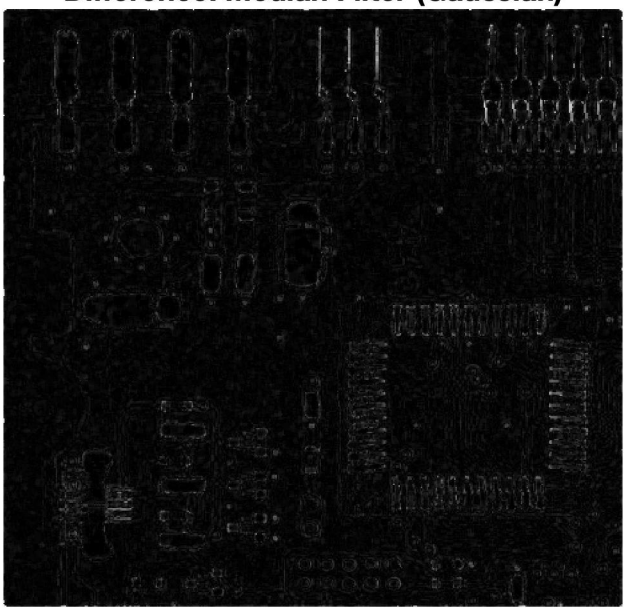
Difference: Alpha-Trimmed (d = 14)

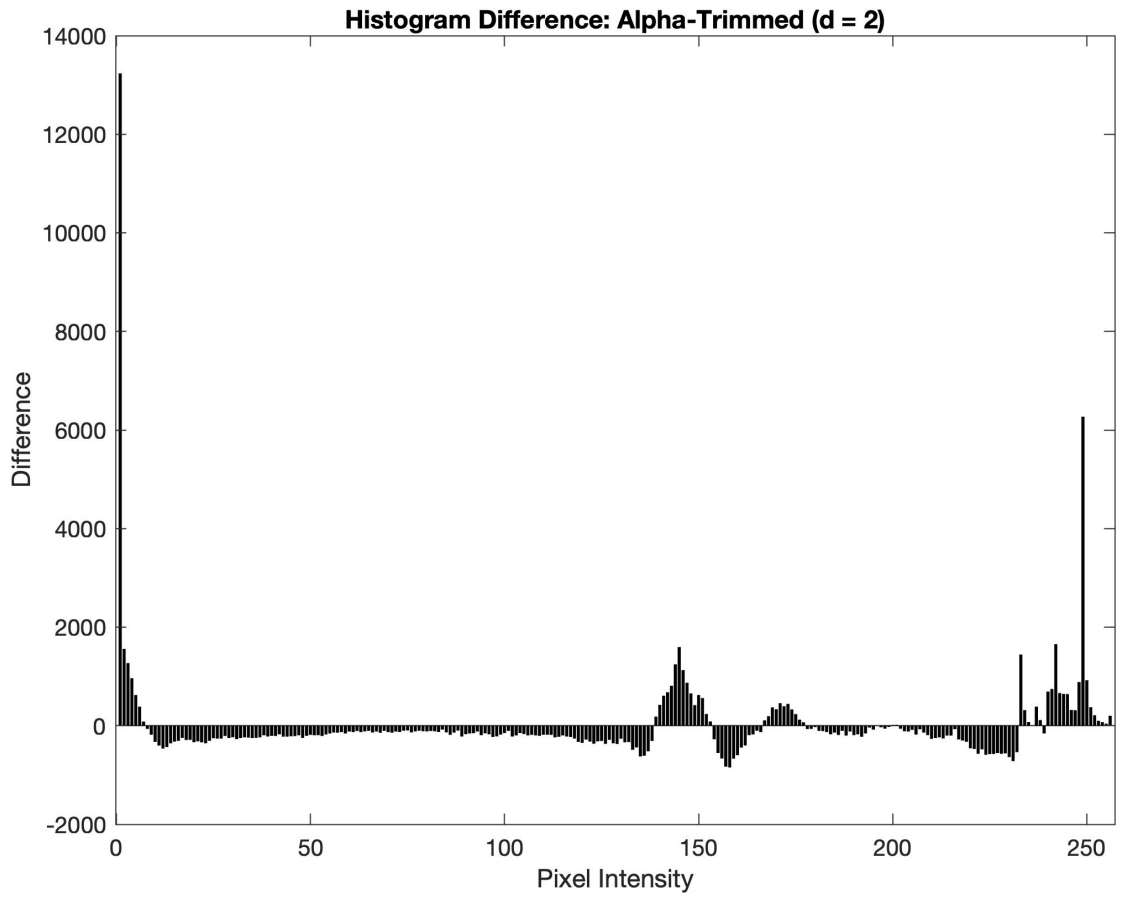


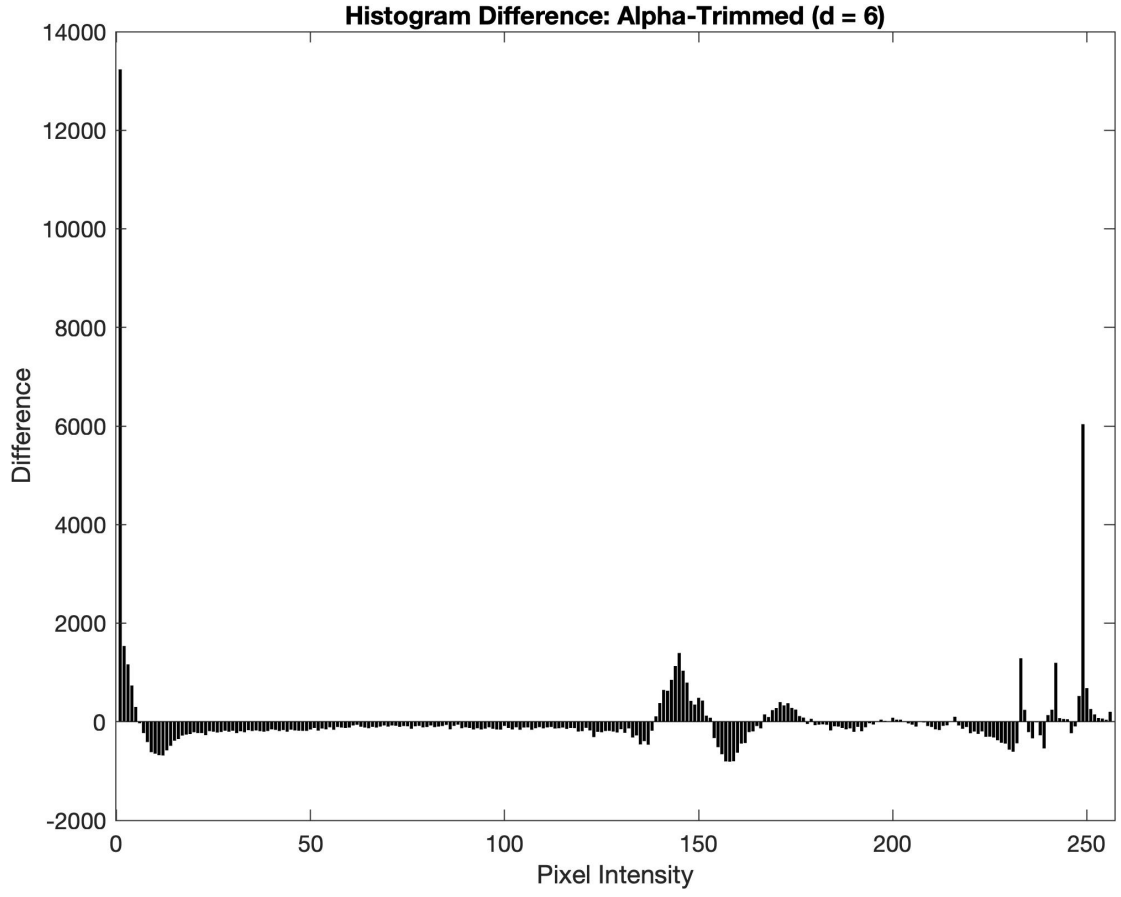
Difference: Alpha-Trimmed (d = 20)

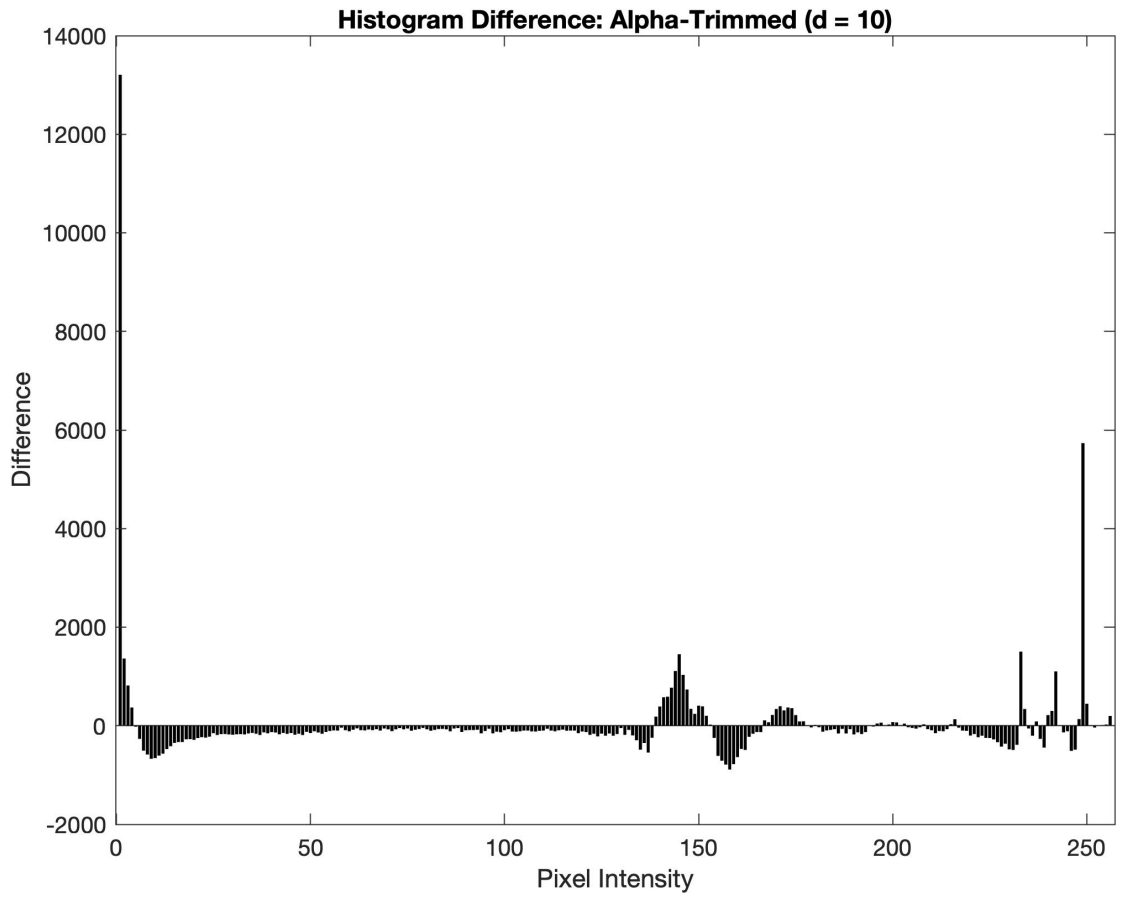


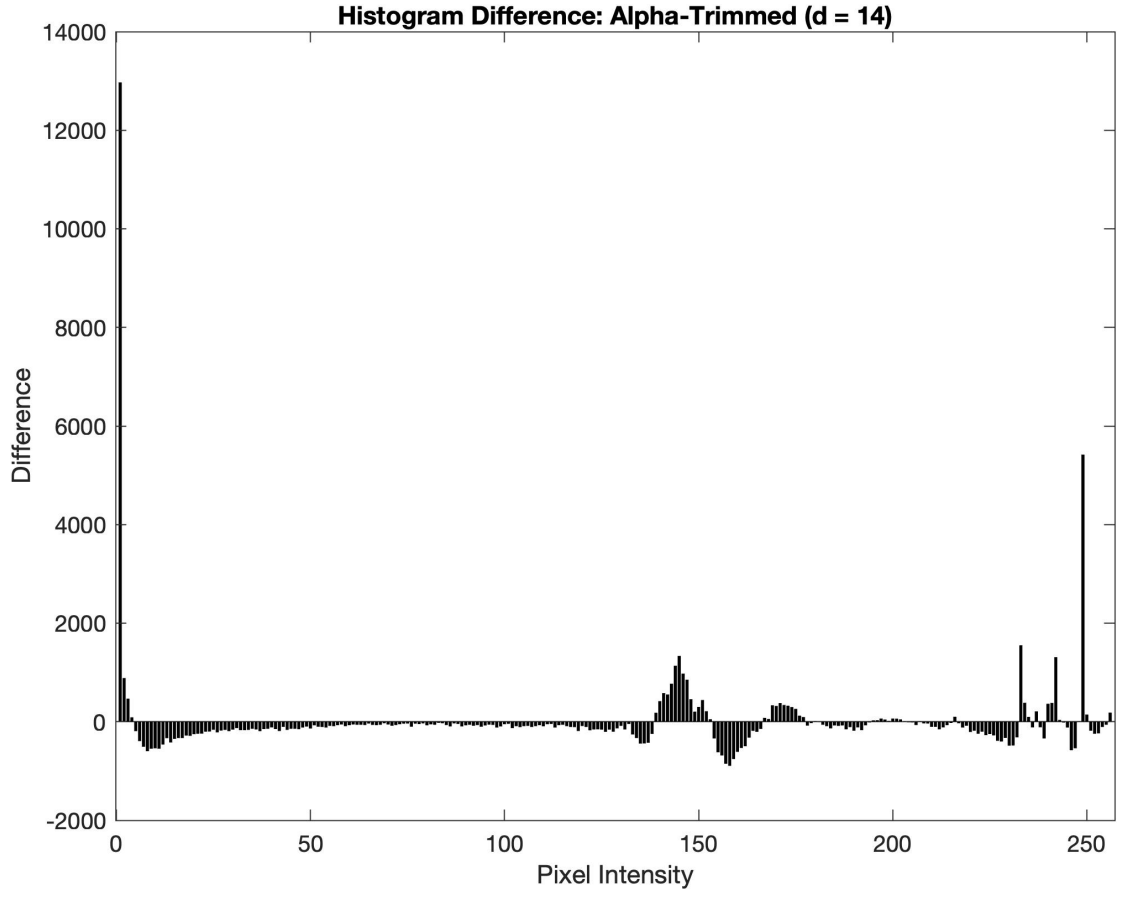
Difference: Median Filter (Gaussian)

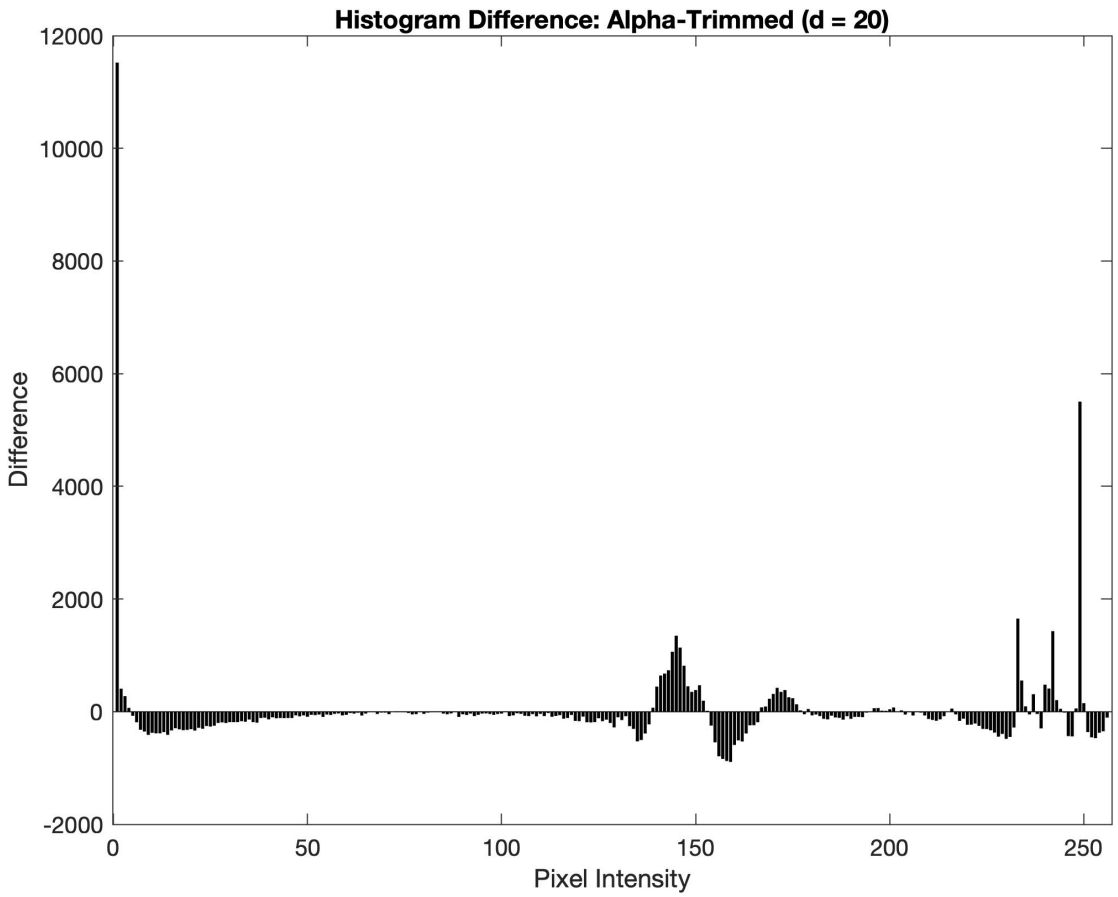


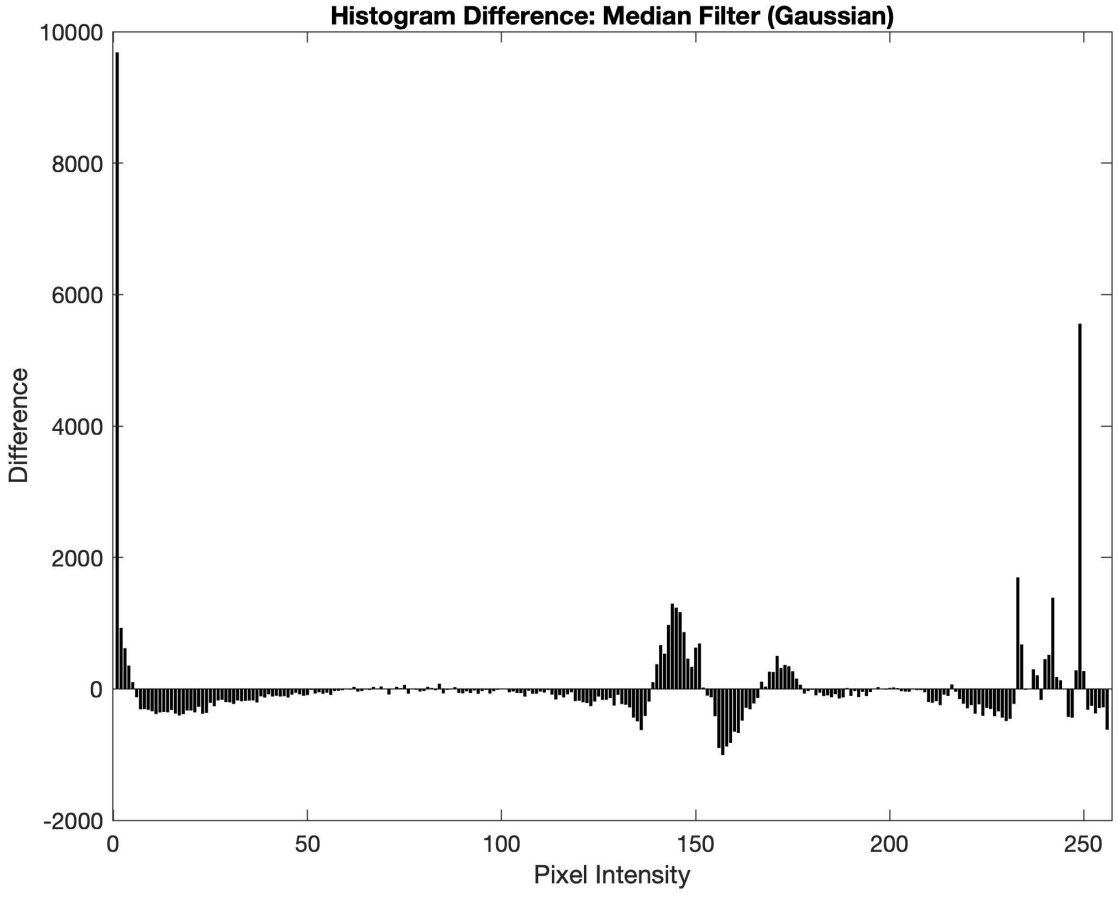

Histogram Difference: Alpha-Trimmed (d = 2)
Difference
Pixel Intensity
14000
12000
10000
8000
6000
4000
2000
0
-2000
0
50
100
150
200
250


Histogram Difference: Alpha-Trimmed (d = 6)
Difference
Pixel Intensity
14000
12000
10000
8000
6000
4000
2000
0
-2000
0
50
100
150
200
250


Histogram Difference: Alpha-Trimmed (d = 10)
Difference
Pixel Intensity
14000
12000
10000
8000
6000
4000
2000
0
-2000
0
50
100
150
200
250


Histogram Difference: Alpha-Trimmed (d = 14)
Difference
Pixel Intensity
14000
12000
10000
8000
6000
4000
2000
0
-2000
0
50
100
150
200
250


Histogram Difference: Alpha-Trimmed (d = 20)
Difference
Pixel Intensity
12000
10000
8000
6000
4000
2000
0
-2000
0
50
100
150
200
250


Histogram Difference: Median Filter (Gaussian)
Difference
Pixel Intensity
10000
8000
6000
4000
2000
0
-2000
0
50
100
150
200
250

Project05-03:

Code:

adaptive_median_filter.m:

```
function output = adaptive_median_filter(img,
max_window_size)
[rows, cols] = size(img);
output = img; % Initialize output image
half_max_size = floor(max_window_size / 2);

for i = 1:rows
for j = 1:cols
window_size = 3; % Start with a 3x3 window
while window_size <= max_window_size
half_size = floor(window_size / 2);

% Get subwindow
r_min = max(i - half_size, 1);
r_max = min(i + half_size, rows);
c_min = max(j - half_size, 1);
c_max = min(j + half_size, cols);
subwindow = img(r_min:r_max, c_min:c_max);

% Compute Zmin, Zmax, Zmed
Zmin = min(subwindow(:));
Zmax = max(subwindow(:));
Zmed = median(subwindow(:));

% Check Level A
if Zmed > Zmin && Zmed < Zmax
% Check Level B
if img(i, j) > Zmin && img(i, j) < Zmax
output(i, j) = img(i, j); % Keep original pixel
else
output(i, j) = Zmed; % Replace with median
end
```

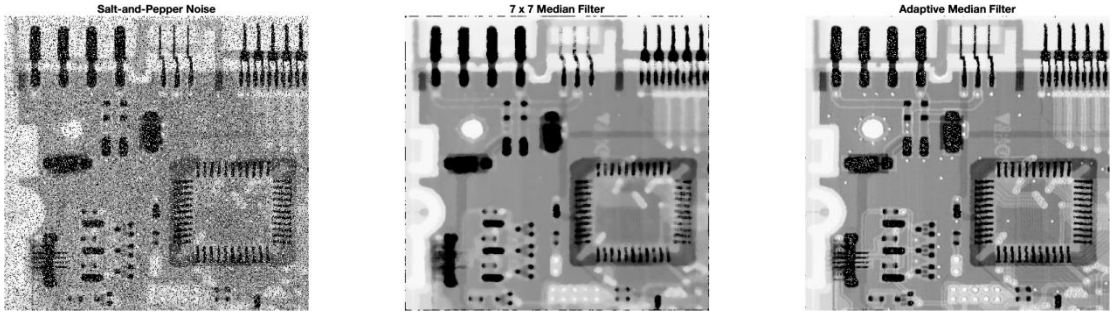
```
break; % Window size is sufficient
else
window_size = window_size + 2; % Expand the window
end
end
end
end
end
```

Project3.m:

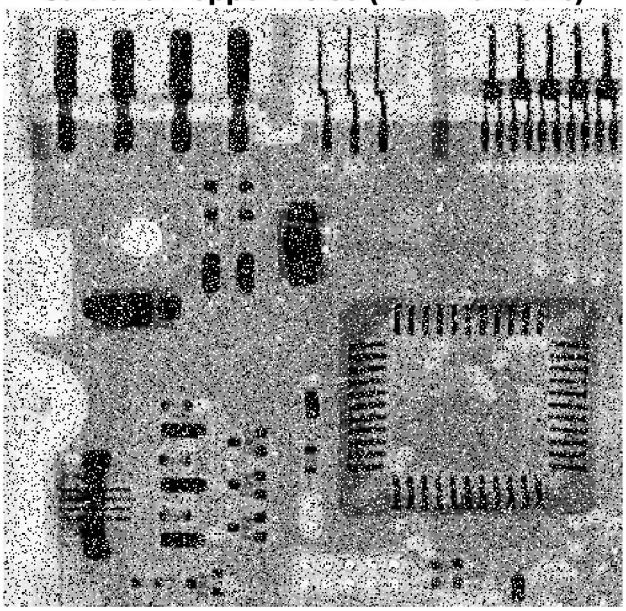
```
img_path =
'/Users/youngbean/Documents/Github/Misc-Projects/Digital
Image Processing/Class5/IMAGES/ckt-board-orig.tif';
output_dir =
'/Users/youngbean/Documents/Github/Misc-Projects/Digital
Image Processing/Class5/Figures/';

img = imread(img_path);
noisy_sp = imnoise(img, 'salt & pepper', 0.25);

figure;
imshow(noisy_sp);
title('Salt-and-Pepper Noise (Pa = Pb = 0.25)');
exportgraphics(gcf, fullfile(output_dir,
'noisy_salt_pepper_0.25.png'), 'Resolution', 300);
close;


% Apply 7 x 7 median filtering
median_filtered = medfilt2(noisy_sp, [7, 7]);

figure;
imshow(median_filtered);
title('7 x 7 Median Filter');
exportgraphics(gcf, fullfile(output_dir,
'median_filtered.png'), 'Resolution', 300);
close;

% Adaptive median filter with maximum subwindow size 7 x 7
```

```
adaptive_filtered = adaptive_median_filter(noisy_sp, 7);

figure;
imshow(adaptive_filtered);
title('Adaptive Median Filter (Max Subwindow 7 x 7)');
exportgraphics(gcf, fullfile(output_dir,
'adaptive_median_filtered.png'), 'Resolution', 300);
close;


figure('Name', 'All Results', 'NumberTitle', 'off');
set(gcf, 'Position', [100, 100, 1800, 600]);

subplot(1, 3, 1);
imshow(noisy_sp);
title('Salt-and-Pepper Noise');

subplot(1, 3, 2);
imshow(median_filtered);
title('7 x 7 Median Filter');

subplot(1, 3, 3);
imshow(adaptive_filtered);
title('Adaptive Median Filter');

exportgraphics(gcf, fullfile(output_dir,
'all_results.png'), 'Resolution', 300);
```

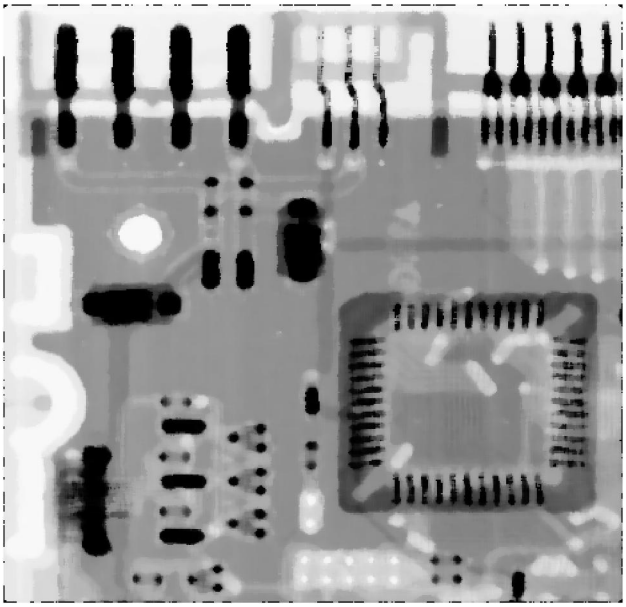
Result:



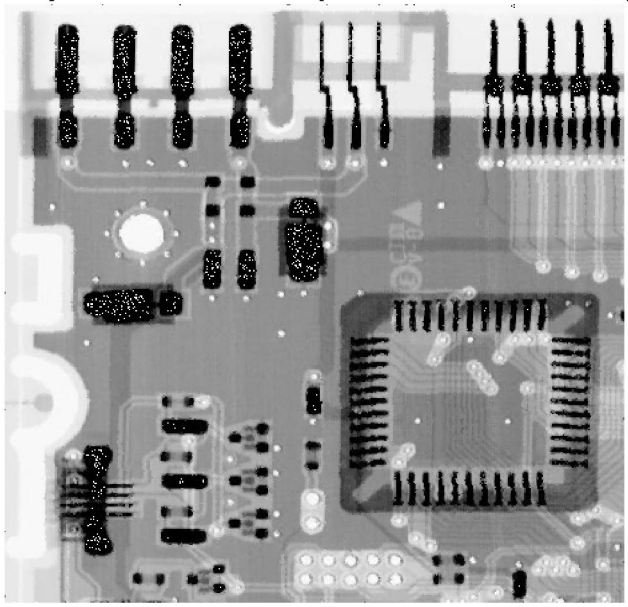
Salt-and-Pepper Noise (Pa = Pb = 0.25)



7 x 7 Median Filter



Adaptive Median Filter (Max Subwindow 7 x 7)



Differences Between 7 × 7 Median Filter and Adaptive Median Filter

1. Window Size:

The 7 × 7 median filter uses a fixed window size throughout the image. In contrast, the adaptive median filter starts with a smaller window (e.g., 3 × 3) and dynamically expands the window size up to the maximum (7 × 7) based on the noise characteristics of the local region.

2. Edge Preservation:

The fixed 7 × 7 median filter may cause over-smoothing in areas with fine details or sharp edges, as it applies the same window size regardless of the local image structure. The adaptive median filter, by using a smaller window in regions with less noise, preserves edges and finer details better.

3. Performance in High-Density Noise:

In regions with high-density noise, the fixed median filter might fail to remove all noise if the noisy pixels dominate the fixed window. The adaptive median filter expands the window size in such regions, allowing it to include more valid pixels for noise removal, leading to better performance.

4. Flexibility:

The adaptive median filter is more versatile, as it tailors the window size to the noise distribution. This adaptability makes it more effective in handling varying noise densities across the image, whereas the fixed median filter applies a uniform approach that might not work optimally for all regions.

In summary, while the 7 × 7 median filter is simpler and effective for moderate noise, the adaptive median filter provides superior results by balancing noise reduction and detail preservation, especially in images with uneven noise distributions.

Project06-01：

Code:

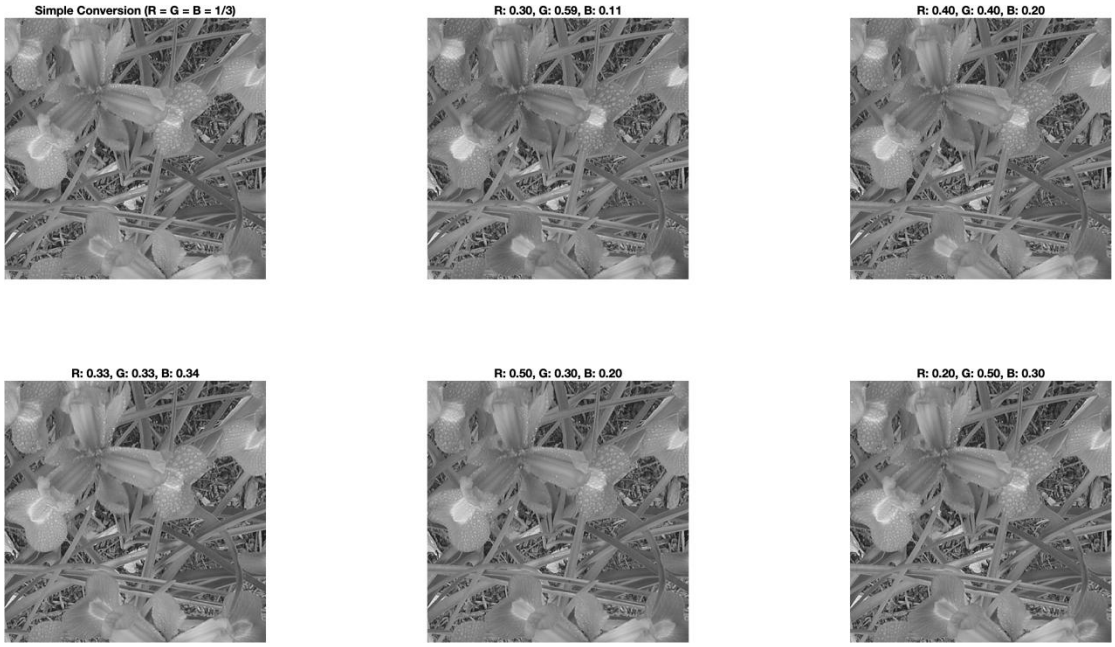
Project4.m:

```
img_path =
'/Users/youngbean/Documents/Github/Misc-Projects/Digital
Image Processing/Class5/IMAGES/top_ left_flower.tif';
output_dir =
'/Users/youngbean/Documents/Github/Misc-Projects/Digital
Image Processing/Class5/Figures/';

rgb_img = imread(img_path);

figure;
imshow(rgb_img);
title('Original RGB Image');
exportgraphics(gcf, fullfile(output_dir,
'original_rgb_image.png'), 'Resolution', 300);
close;

% (b) Convert to gray image using I = (R + G + B) / 3
% Separate the Red, Green, and Blue channels
R = double(rgb_img(:, :, 1));
G = double(rgb_img(:, :, 2));
B = double(rgb_img(:, :, 3));

% Calculate intensity
I_simple = (R + G + B) / 3;
I_simple = uint8(I_simple);
figure;
imshow(I_simple);
title('Gray Image (Simple Conversion)');
exportgraphics(gcf, fullfile(output_dir,
'gray_image_simple.png'), 'Resolution', 300);
close;

% (c) Adjust proportions of R, G, B and compare results
% Define different proportions
proportions = [
0.3, 0.59, 0.11; % Common grayscale conversion weights
0.4, 0.4, 0.2; % Another example
0.33, 0.33, 0.34; % Balanced
0.5, 0.3, 0.2; % Stronger weight on R
```

```
0.2, 0.5, 0.3 % Stronger weight on G
];

% Initialize storage for results
gray_images = cell(size(proportions, 1) + 1, 1);

% Add simple conversion result as the first image
gray_images{1} = I_simple;

% Process and save results
for i = 1:size(proportions, 1)
% Get current proportions
r_weight = proportions(i, 1);
g_weight = proportions(i, 2);
b_weight = proportions(i, 3);
% Calculate intensity with weighted proportions
I_weighted = r_weight * R + g_weight * G + b_weight * B;
I_weighted = uint8(I_weighted); % Convert back to uint8 for
display
% Store the result
gray_images{i + 1} = I_weighted;
% Save each image separately
figure;
imshow(I_weighted);
title(sprintf('R: %.2f, G: %.2f, B: %.2f', r_weight, g_weight,
b_weight));
exportgraphics(gcf, fullfile(output_dir,
sprintf('gray_image_r%.2f_g%.2f_b%.2f.png', r_weight,
g_weight, b_weight)), 'Resolution', 300);
close;
end

figure('Name', 'Comparison of Gray Images with Sample
Conversion', 'NumberTitle', 'off');
set(gcf, 'Position', [100, 100, 1400, 800]);

subplot(2, ceil((size(proportions, 1) + 1) / 2), 1);
imshow(gray_images{1});
title('Simple Conversion (R = G = B = 1/3)');
```

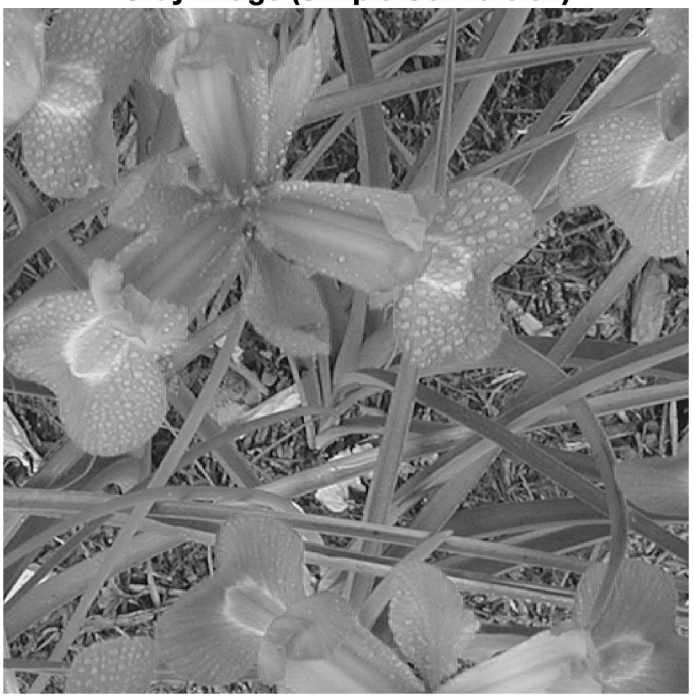
```
for i = 1:size(proportions, 1)
subplot(2, ceil((size(proportions, 1) + 1) / 2), i + 1);
imshow(gray_images{i + 1});
title(sprintf('R: %.2f, G: %.2f, B: %.2f', proportions(i, 1),
proportions(i, 2), proportions(i, 3)));
end

exportgraphics(gcf, fullfile(output_dir,
'gray_image_comparison.png'), 'Resolution', 300);
```

Result:

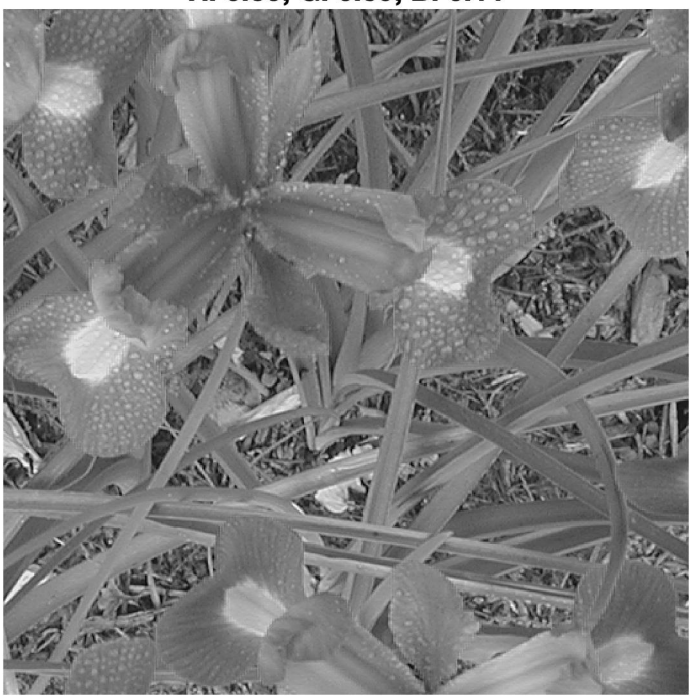


Original RGB Image

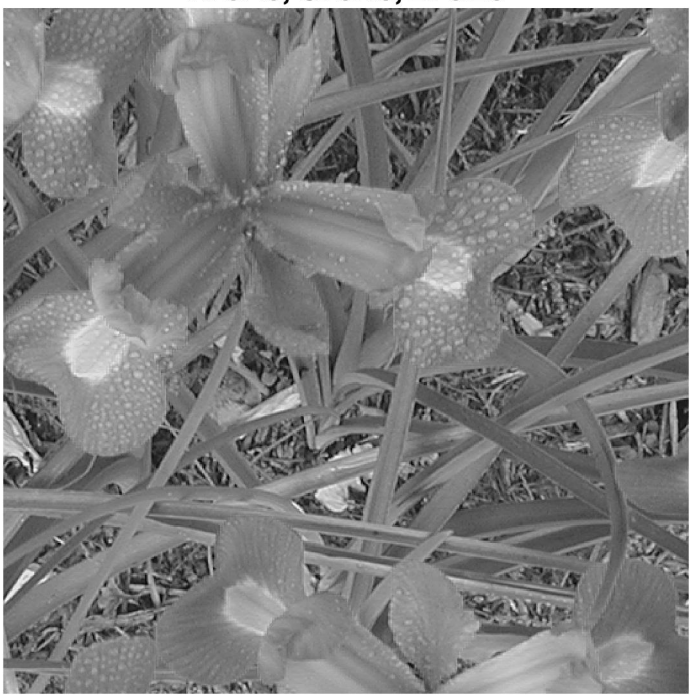
Gray Image (Simple Conversion)



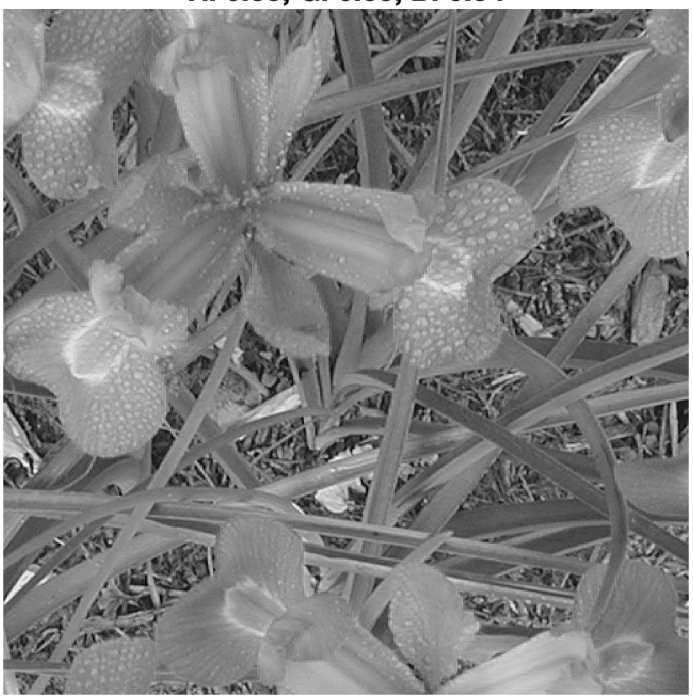
R: 0.30, G: 0.59, B: 0.11



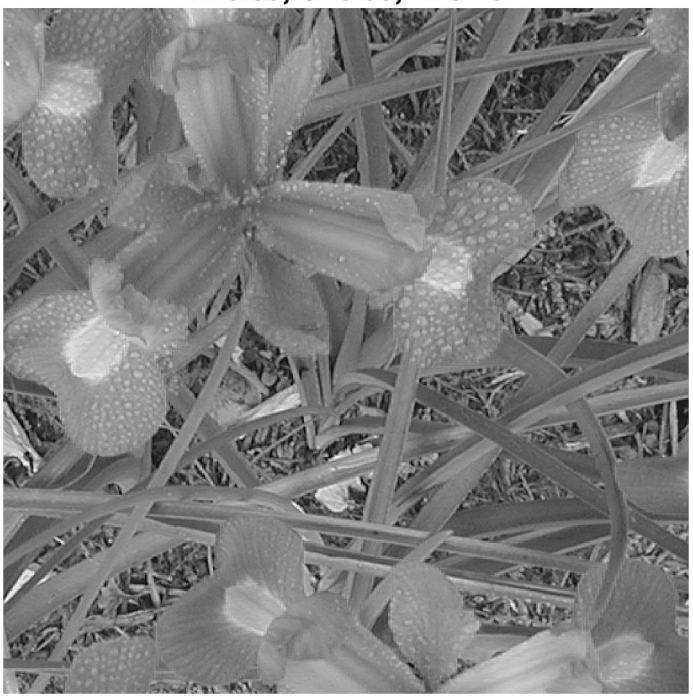
R: 0.40, G: 0.40, B: 0.20



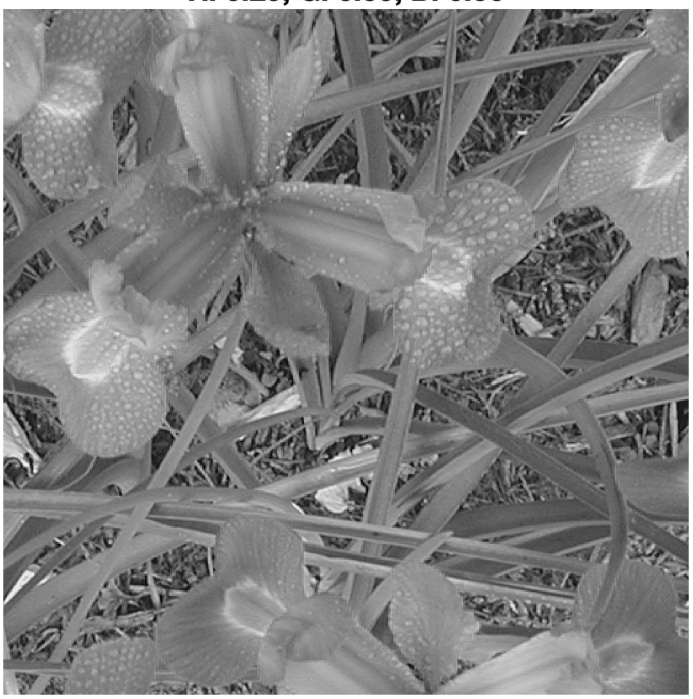
R: 0.33, G: 0.33, B: 0.34



R: 0.50, G: 0.30, B: 0.20



R: 0.20, G: 0.50, B: 0.30



In **Project 06-01 Part (d)**, I compared the results of grayscale conversion by adjusting the proportions of the RGB components and determined that the proportion $\mathrm{R}: 0.3, \mathrm{G}: 0.59, \mathrm{B}: 0.11$ is the best choice. The rationale for this decision is as follows:

1.Adaptation to Human Perception:

The human eye is most sensitive to green (G), followed by red (R), and least sensitive to blue (B). Assigning a higher weight to the green component (0.59), a moderate weight to red (0.3), and a smaller weight to blue (0.11) aligns well with the natural brightness perception of the human visual system.

2. Limitations of Simple Averaging:

The formula $\mathrm{I} = (\mathrm{R} + \mathrm{G} + \mathrm{B})/3$ uses equal weights for all three

channels, ignoring the varying sensitivities of the human eye to different colors. As a result, the grayscale image produced by this method may appear less natural in brightness and contrast, particularly for images with vibrant colors.

3. Advantages of the Selected Proportion:

Using $R: 0.3, G: 0.59, B: 0.11$ produces a grayscale image that more accurately reflects the brightness and contrast characteristics of the original RGB image. Experimentally, this proportion showed clearer details and a more natural brightness distribution compared to other tested proportions.

4. Comparison with Other Proportions:

With $R: 0.4, G: 0.4, B: 0.2$, the weights are more balanced, but the lack of emphasis on green sensitivity results in a grayscale image with less natural brightness.

The $R: 0.5, G: 0.3, B: 0.2$ proportion gives excessive weight to the red channel, resulting in an overly bright grayscale image that does not accurately represent scene details.

$R: 0.2, G: 0.5, B: 0.3$ places more emphasis on the green channel, but the additional weight on blue can distort brightness distribution in some regions.

Through comparison, the proportion $R: 0.3, G: 0.59, B: 0.11$

proved to strike the best balance between brightness and detail preservation, resulting in a grayscale image that closely matches human visual perception. This proportion is widely used in practical applications and is suitable for various types of image grayscale conversion tasks.

Project06-02:

Code:

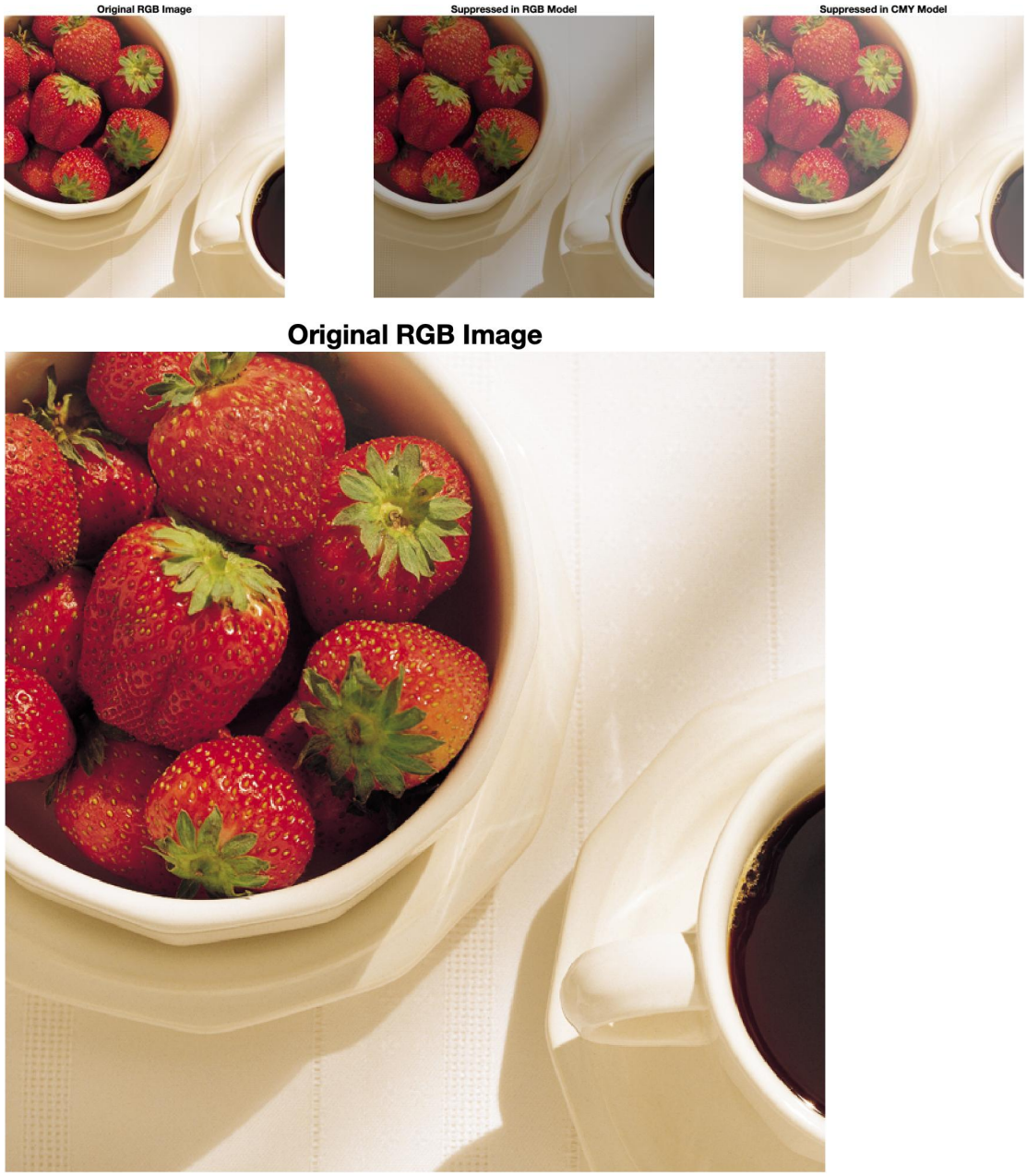
Project5.m:

```
img_path =
'/Users/youngbean/Documents/Github/Misc-Projects/Digital
Image Processing/Class5/IMAGES/Fig_strawberries.tif';
output_dir =
'/Users/youngbean/Documents/Github/Misc-Projects/Digital
Image Processing/Class5/Figures/';

rgb_img = imread(img_path);

figure;
imshow(rgb_img);
title('Original RGB Image');
exportgraphics(gcf, fullfile(output_dir,
'original_rgb_image.png'), 'Resolution', 300);
close;


% (a) Suppress intensity in RGB model
k = 0.7; % Suppression factor

% Suppress intensity in RGB model
rgb_suppressed = uint8(k * double(rgb_img)); % Scale
intensity


figure;
```

```
imshow(rgb_suppressed);
title('Intensity Suppressed in RGB Model');
exportgraphics(gcf, fullfile(output_dir,
'intensity_suppressed_rgb.png'), 'Resolution', 300);
close;

% (b) Suppress intensity in CMY model
% Convert RGB to CMY
cmy_img = 1 - im2double(rgb_img); % Normalize to [0, 1] range
and invert

% Suppress intensity in CMY model
cmy_suppressed = cmy_img * k;

% Convert back to RGB
rgb_from_cmy = uint8((1 - cmy_suppressed) * 255);

figure;
imshow(rgb_from_cmy);
title('Intensity Suppressed in CMY Model');
exportgraphics(gcf, fullfile(output_dir,
'intensity_suppressed_cmy.png'), 'Resolution', 300);
close;

figure('Name', 'Comparison of Intensity Suppression',
'NumberTitle', 'off');
set(gcf, 'Position', [100, 100, 1600, 600]);

subplot(1, 3, 1);
imshow(rgb_img);
title('Original RGB Image');

subplot(1, 3, 2);
imshow(rgb_suppressed);
title('Suppressed in RGB Model');

subplot(1, 3, 3);
imshow(rgb_from_cmy);
title('Suppressed in CMY Model');
```

```
exportgraphics(gcf, fullfile(output_dir,
'intensity_suppression_comparison.png'), 'Resolution',
300);
```

Result:



Intensity Suppressed in RGB Model

Intensity Suppressed in CMY Model

【小结或讨论】

在本次实验中，我系统地探索了图像处理中的噪声添加、滤波去噪以及颜色模型的转换与强度调整等关键技术。

在 PROJECT 05-01 中，我通过 MATLAB 的 imnoise 函数实现了噪声添加，包括高斯噪声和盐椒噪声。高斯噪声由均值 μ 和方差 σ^2 定义，其概率密度函数为:

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2x}}$$

实验中，我设置 $\mu = 0$ 和 $\sigma^2 = 0.01$，生成了零均值的高斯噪声，模拟了图像中的随机噪声。而盐椒噪声通过指定概率 $P_\alpha = P_b$，分别以概率 P_α 和 P_b 替换像素为白 (1) 或黑 (0)，产生了明显的 "白点" 和 "黑点"。这些噪声的添加为后续的去噪处理提供了实验基础。

在 PROJECT 05-02 中，我使用多种滤波器对噪声图像进行了去噪处理。首先，我通过 5×5 均值滤波器对噪声图像进行平滑处理，其公式为:

$$I_{menh}(x,y) = \frac{1}{m \cdot n} \sum_{i=-\lfloor m/2 \rfloor}^{\lfloor m/2 \rfloor} \sum_{j=-\lfloor n/2 \rfloor}^{\lfloor n/2 \rfloor} I(x+i, y+j)$$

其中 $m = 5, n = 5$ 。结果显示，均值滤波对噪声的平滑效果显著，但也导致了边缘模糊和细节丢失。

接着，我应用了 5×5 中值滤波器，其原理是用局部窗口中像素的中值代替当前像素值:

$$I_{median}(x, y) = \text{median}\{I(x+i, y+j): -\lfloor m/2 \rfloor \leq i, j \leq \lfloor m/2 \rfloor\}$$

实验结果表明，中值滤波在去除盐椒噪声方面表现优异，同时较好地保留了图像的边缘细节。此外，我实现了 Alpha 修剪均值滤波器，该滤波器通过调整修剪参数 d，从窗口中移除两端的极值像素后计算均值:

$$I_{alphia}(x, y) = \frac{1}{T} \sum_{k=(d/2)+1}^{N-(d/2)} S_k$$

其中 S_k 是排序后的窗口像素值，$T = N - d_0$ 调整 d 值让我在去噪能力和细节保留之间找到了平衡点。通过图像差异和直方图分析，我发现 Alpha 修剪均值滤波器在噪声强度较大时表现出色。

在 PROJECT 05-03 中，我设计并实现了自适应中值滤波器，与固定窗口的 7×7 中值滤波器进行了对比。自适应中值滤波器的特点是根据噪声区域的复杂性动态调整窗口大小，其核心算法分为两步:

确定窗口中值 Z_{med} 、最小值 Z_{min} 和最大值 Z_{max} :

$$Z_{min} \leq Z_{med} \leq Z_{max}$$

根据中值是否介于最小值和最大值之间，以及当前像素是否在噪声范围内，决定保留原像素还是替换为中值。实验表明，自适应中值滤波在去除高密度盐椒噪声时，能够更好地保留图像细节。

在 PROJECT 06-01 中，我实现了 RGB 彩色图像到灰度图像的转换。按照公式:

$$I = \frac{R + G + B}{3}$$

我计算了简单灰度值，并进一步通过调整 RGB 三个通道的权重 (如 $R: 0.3, G: 0.59, B: 0.11$ 等)，生成了不同灰度图像。常用比例更符合人眼对亮度的感知，实验结果清晰展示了不同权重设置对灰度图像亮度和对比度的影响。

在 PROJECT 06-02 中，我分别在 RGB 和 CMY 模型中实现了图像强度的抑制 (抑制因子 $k = 0.7$)。在 RGB 模型中，我直接缩放 RGB 值:

$$I_{RGB} = k \cdot I$$

而在 CMY 模型中，先将 RGB 转换为 CMY 模型:

$$CMY = 1 - \frac{RGB}{255}$$

抑制强度后再转换回 RGB 模型。两种模型的结果相似，但 CMY 模型在颜色调整方面更加灵活。

通过本次实验，我不仅掌握了图像噪声生成与去噪的理论知识和编程实现，还深入理解了彩色图像的灰度化和颜色模型转换的应用场景。这些技术为图像预处理、增强和分析奠定了重要的基础。