TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS
DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS

Individual Work Nr. 4

LINEAR PROCESSING OF IMAGES USING MATLAB

Elaborated: **Cristian Brinza**
st. gr. FAF-212

Verified: **Railean Serghei**
lect. univ

Chișinău, 2024

Purpose of the laboratory work:

The purpose of this lab work is to explore methods of linear image processing.

Theory notions:

Arithmetic Operations on Images: These operations involve applying a simple function to each grayscale value in the image. The functionf(x) maps the range 0 to 255 onto itself. Simple functions include adding or subtracting a constant value to each pixel, or multiplying each pixel by a constant. After these operations, it might be necessary to round the results to ensure that the outcomes are integers within the range 0 to 255. We can achieve this by first rounding the result (if necessary) to obtain an integer, and then "clipping" the values by setting:

Histogram Equalization: For a grayscale image, its histogram consists of the frequency of its gray levels. We can learn a lot about an image's appearance from its histogram—for example, in a dark image, the gray levels (and thus, the histogram) would be clustered at the lower end, and in a uniformly bright image, the gray levels would be grouped at the upper end. A well-contrasted image would have gray levels well spread across the range. The histogram of an image can be visualized in MATLAB using the imhist function. A poorly contrasted image would have its gray values clustered together in the center of the histogram.

Image Filtering: Image filtering can be seen as an extension of applying a function to each pixel's value. Spatial filtering involves applying a function across a neighborhood of each pixel using a "mask" or kernel. This operation generally requires positioning the mask over the current pixel, forming all the products of filter elements with corresponding neighborhood elements, and summing all these products. A typical example is using a mask to compute the average of all nine values in the mask, which then becomes the new gray value of the corresponding pixel in the new image.

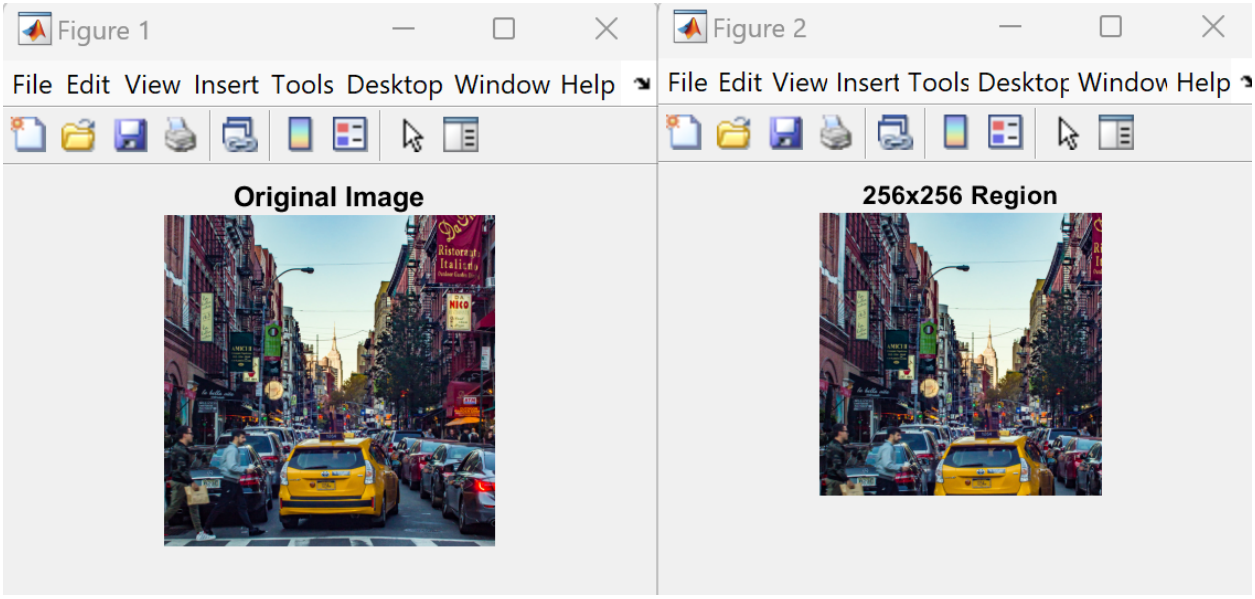
Low-Pass and High-Pass Filters: It's useful to have terminology to discuss a filter's effect on an image, particularly in choosing the right filter for a specific image based on the notion of frequencies. High-frequency components are characterized by rapid changes in gray values over short distances (e.g., edges and noise), whereas low-frequency components show little change (e.g., backgrounds, skin textures). A low-pass filter "passes" low-frequency components and reduces or eliminates high-frequency components, and vice versa for a high-pass filter

Implementation:

1. **GAMMA CURVE MANIPULATION**:

   1. ***Display an image of your choice using imshow, after previously saving this image. Read the image and select a region of 256x256.***

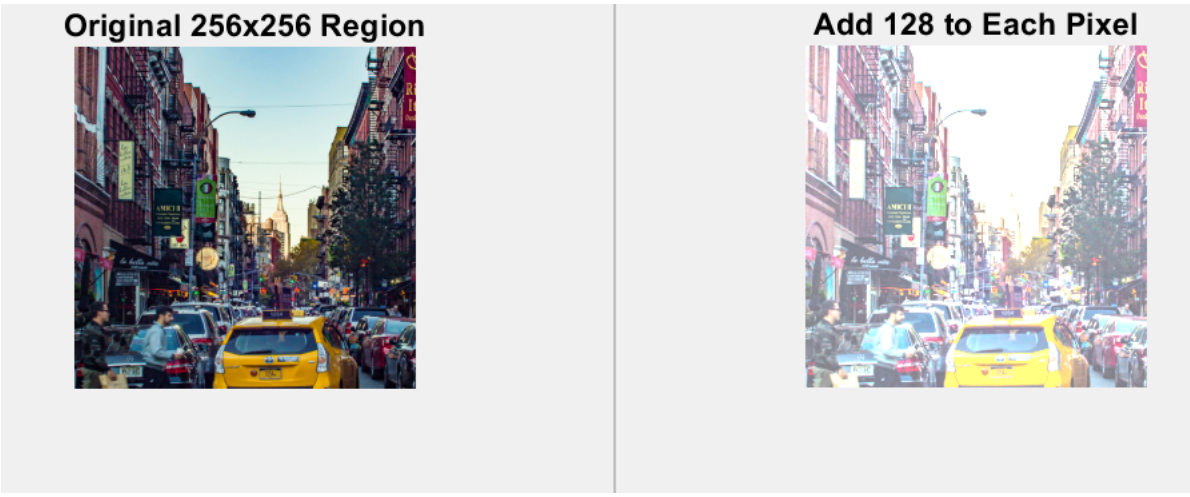
```
% Read the image from a file
c = imread('utm.tif');

% Display the dimensions of the image
sizeC = size(c);
fprintf('Dimensions of the image: %dx%dx%d\n', sizeC(1),
sizeC(2), sizeC(3));

% Ensure the image has three channels (RGB)
if sizeC(3) >= 3
    % Keep only the first three channels if there are
more than three
    c = c(:, :, 1:3);

    % Extract a 256x256 region from the original image
    if sizeC(1) >= 256 && sizeC(2) >= 256
        cc = c(1:256, 1:256, :); % Extracting the
specified region
    else
        error('The original image dimensions are too
small to extract a 256x256 region.');
    end

    % Display the original and cropped images
    figure; imshow(c); title('Original Image');
    figure; imshow(cc); title('256x256 Region');
else
    error('The image is not an RGB image. Required size
is MxNx3.');
end
```



This code loads an image and crops a 256x256 pixel region from the top-left corner. Two figures are displayed: the full image and the cropped region.

2. ***Perform the addition of 128 to each pixel value in the image.***

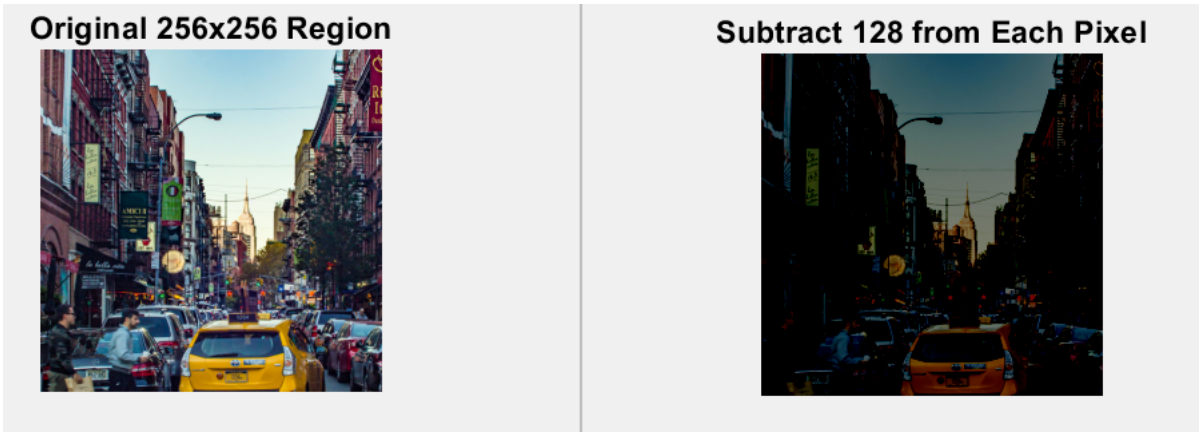
```
% Adding 128 to each pixel value in the selected region
c1 = imadd(cc, 128);
figure; imshow(cc); title('Original 256x256 Region');
figure; imshow(c1); title('Add 128 to Each Pixel');
```



This operation increases the brightness of each pixel by adding 128 to its value, potentially causing some pixel values to saturate at 255.

3. ***Perform the subtraction of 128 from each pixel value in the image.***

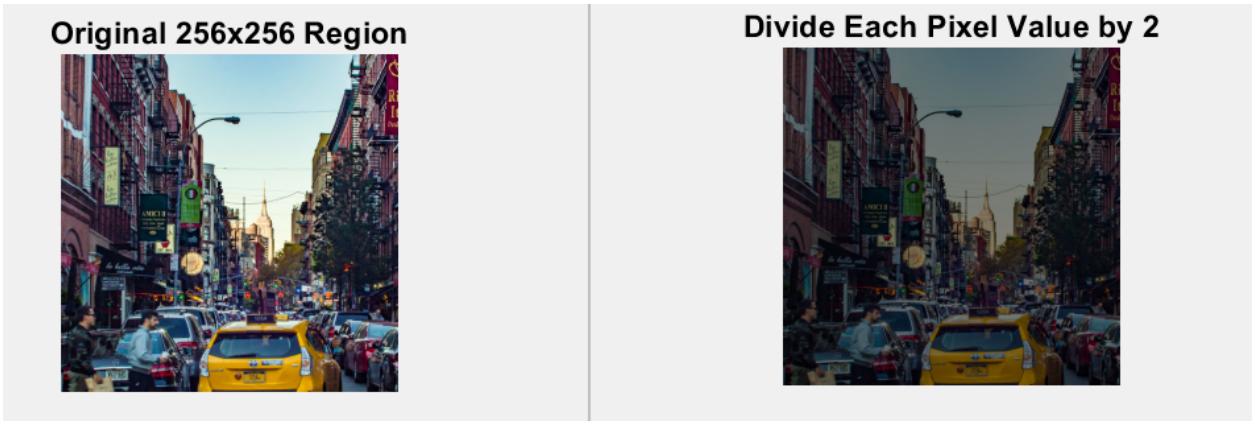
```
% Subtracting 128 from each pixel value in the selected
region
c2 = imsubtract(cc, 128);
figure; imshow(cc); title('Original 256x256 Region');
figure; imshow(c2); title('Subtract 128 from Each
Pixel');
```



This operation decreases the brightness by subtracting 128 from each pixel's value, which may result in many pixels approaching or hitting the minimum value of 0.

4. ***Divide each pixel value in the image by 2.***

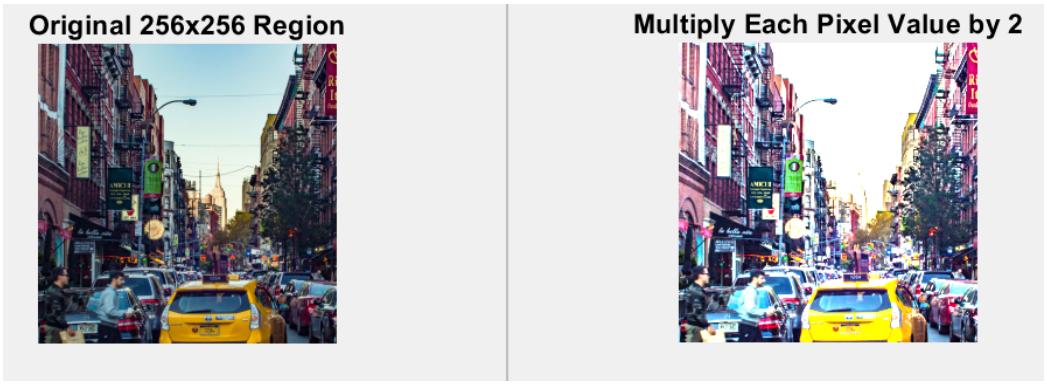
```
% Dividing each pixel value in the selected region by 2
c3 = imdivide(cc, 2);
figure; imshow(cc); title('Original 256x256 Region');
figure; imshow(c3); title('Divide Each Pixel Value by
2');
```



Dividing pixel values by 2 darkens the image, reducing the intensity of each pixel to half of its original value.

5. ***Multiply each pixel value in the image by 2.***

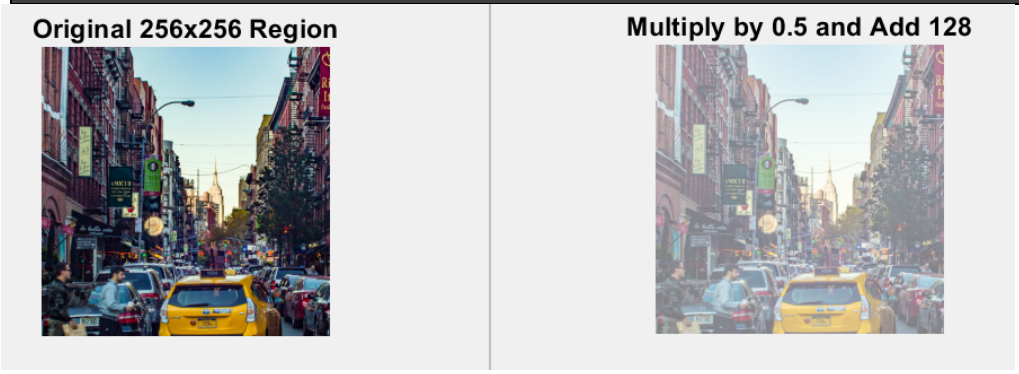
```
% Multiplying each pixel value in the selected region by
2
c4 = immultiply(cc, 2);
figure; imshow(cc); title('Original 256x256 Region');
figure; imshow(c4); title('Multiply Each Pixel Value by
2');
```



Multiplying each pixel value by 2 increases the brightness, potentially causing many pixels to saturate at the maximum value of 255.

6. ***Multiply each pixel value by 0.5 and add 128 to each pixel value in the image.***

```
% Multiplying each pixel value by 0.5 and then adding 128
c5 = imadd(immultiply(cc, 0.5), 128);
figure; imshow(cc); title('Original 256x256 Region');
figure; imshow(c5); title('Multiply by 0.5 and Add 128');
```



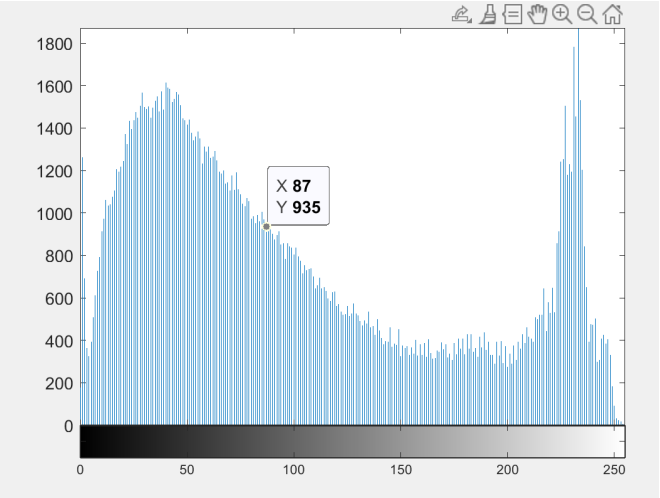
This operation first reduces each pixel value to half, then adds 128, balancing the overall brightness while maintaining a moderate contrast.

2. HISTOGRAM EQULIZATION:

1. Display the imge and the histogram

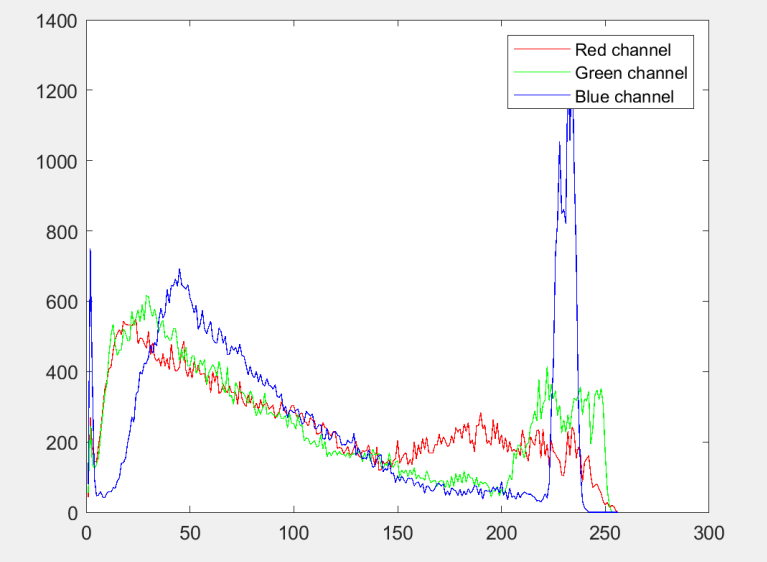
For the grayscale image:

```
imshow(cc);
figure;
imhist(cc);
axis tight;
```



For the color image:

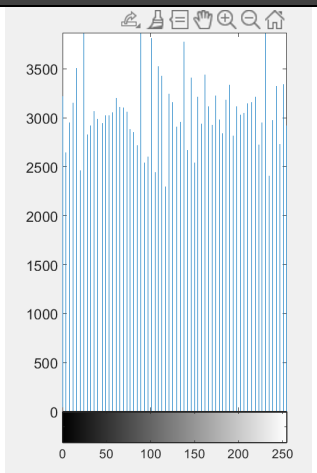
```
R = imhist(cc(:,:,1));
G = imhist(cc(:,:,2));
B = imhist(cc(:,:,3));
figure, plot(R, 'r')
hold on, plot(G, 'g')
plot(B, 'b'), legend('Red channel', 'Green channel',
'Blue channel');
```



2. ***Perform histogram equalization and display the image and histogram of the selected image***

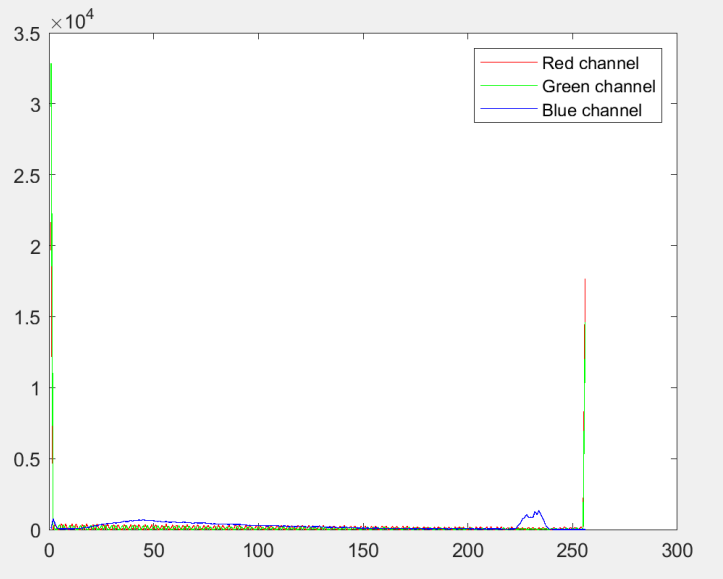
For the grayscale image:

```
h = histeq(cc);
figure
subplot(1,2,1)
imshow(cc)
subplot(1,2,1)
imhist(h)
```



For the color image:

```
I2 = imadjust(cc, [.2 .3 0; .6 .7 1], []);
R = imhist(I2(:,:,1));
G = imhist(I2(:,:,2));
B = imhist(I2(:,:,3));
figure, plot(R, 'r')
hold on, plot(G, 'g')
plot(B, 'b'), legend('Red channel', 'Green channel',
'Blue channel');

Display the original image and after histogram
equalization:
figure
subplot(1,2,1)
imshow(cc)
subplot(1,2,2)
imshow(I2)
```



3. **IMAGE FILTERING**:

   1. ***Filter the image with a 3x3 filter.***

```
```

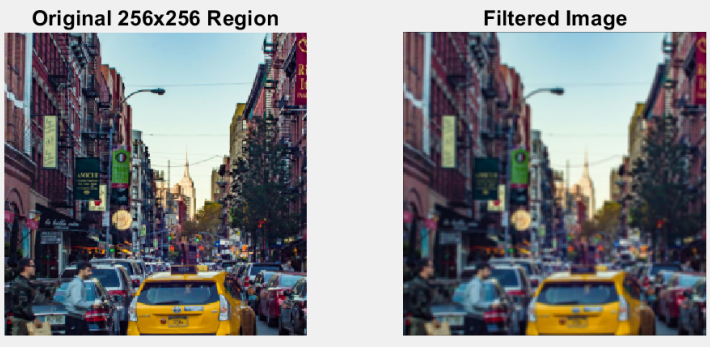
```
% Read the image from a file
c = imread('utm.tif');

% Display the dimensions of the image
sizeC = size(c);
fprintf('Dimensions of the image: %dx%dx%d\n', sizeC(1),
sizeC(2), sizeC(3));

% Ensure the image has three channels (RGB)
if sizeC(3) >= 3
    % Keep only the first three channels if there are
more than three
    c = c(:, :, 1:3);

    % Extract a 256x256 region from the original image
    if sizeC(1) >= 256 && sizeC(2) >= 256
        cc = c(1:256, 1:256, :); % Extracting the
specified region
    else
        error('The original image dimensions are too
small to extract a 256x256 region.');
    end

    % Apply a filter to each channel
    f1 = fspecial('average');
    cf1_r = filter2(f1, double(cc(:,:,1)));
    cf1_g = filter2(f1, double(cc(:,:,2)));
    cf1_b = filter2(f1, double(cc(:,:,3)));

    % Combine the filtered channels
    cf1 = cat(3, uint8(cf1_r), uint8(cf1_g),
uint8(cf1_b));

    % Display the original and filtered images
    figure;
    subplot(1,2,1);
    imshow(cc);
    title('Original 256x256 Region');

    subplot(1,2,2);
    imshow(cf1);
```

```
        title('Filtered Image');
    else
        error('The image is not an RGB image. Required size
    is MxNx3.');
    end
```

For the grayscale image:


```
IG = rgb2gray(cc);
figure;
imshow(IG)

Filter the image:
f1 = fspecial('average');
cf1 = filter2(f1, IG);
figure
subplot(1,2,1)
imshow(IG)
subplot(1,2,2)
imshow(cf1/255)
```

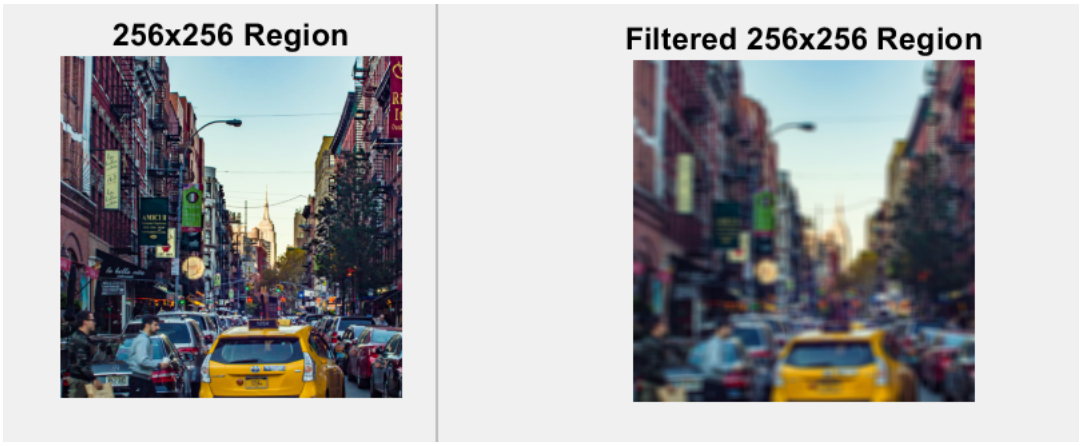
For the color image:



2. ***Filter the image with a 5x7 filter.***

```
% Read the image from a file
c = imread('utm.tif');

% Display the dimensions of the image
sizeC = size(c);
fprintf('Dimensions of the image: %dx%dx%d\n', sizeC(1),
sizeC(2), sizeC(3));

% Ensure the image has three channels (RGB)
if sizeC(3) >= 3
    % Keep only the first three channels if there are
more than three
    c = c(:, :, 1:3);

    % Extract a 256x256 region from the original image
    if sizeC(1) >= 256 && sizeC(2) >= 256
        cc = c(1:256, 1:256, :); % Extracting the
specified region
    else
        error('The original image dimensions are too
small to extract a 256x256 region.');
    end

    % Create a 5x7 averaging filter
    filter = ones(5, 7) / (5*7);

    % Apply the filter to the cropped image using
imfilter
    filtered_cc = imfilter(cc, filter, 'same',
'replicate');

    % Display the original and cropped images
    figure; imshow(c); title('Original Image');
    figure; imshow(cc); title('256x256 Region');
    figure; imshow(filtered_cc); title('Filtered 256x256
Region');
else
    error('The image is not an RGB image. Required size
is MxNx3.');
```

```
end
```

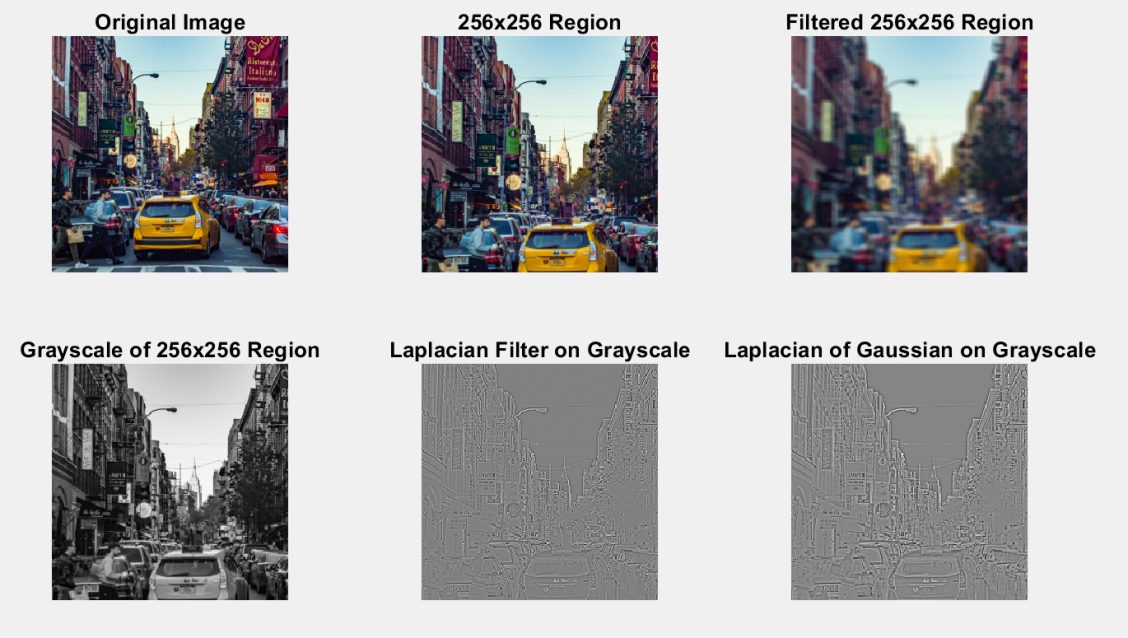


3. ***Detect edges in the image using a Laplacian and Laplacian of Gaussian filter.***

For grayscale image:

```
% Read the image from a file
c = imread('utm.tif');

% Display the dimensions of the image
sizeC = size(c);
fprintf('Dimensions of the image: %dx%dx%d\n', sizeC(1),
sizeC(2), sizeC(3));

% Ensure the image has three channels (RGB)
if sizeC(3) >= 3
    % Keep only the first three channels if there are
more than three
    c = c(:, :, 1:3);

    % Extract a 256x256 region from the original image
    if sizeC(1) >= 256 && sizeC(2) >= 256
        cc = c(1:256, 1:256, :); % Extracting the
specified region
    else
        error('The original image dimensions are too
small to extract a 256x256 region.');
    end
```

```
        % Convert the cropped image to grayscale
        gray_cc = rgb2gray(cc);

        % Create a 5x7 averaging filter
        filter = ones(5, 7) / (5*7);

        % Apply the filter to the cropped colored image using
    imfilter
        filtered_cc = imfilter(cc, filter, 'same',
    'replicate');

        % Apply Laplacian filter to the grayscale image
        laplacian_filter = fspecial('laplacian', 0);
        laplacian_gray = imfilter(double(gray_cc),
    laplacian_filter, 'replicate');

        % Apply Laplacian of Gaussian filter to the grayscale
    image
        log_filter = fspecial('log', [5 5], 0.5);
        log_gray = imfilter(double(gray_cc), log_filter,
    'replicate');

        % Display the original, filtered, and edge-detected
    images
        figure;
        subplot(2, 3, 1), imshow(c), title('Original Image');
        subplot(2, 3, 2), imshow(cc), title('256x256
    Region');
        subplot(2, 3, 3), imshow(filtered_cc),
    title('Filtered 256x256 Region');
        subplot(2, 3, 4), imshow(gray_cc), title('Grayscale
    of 256x256 Region');
        subplot(2, 3, 5), imshow(laplacian_gray, []),
    title('Laplacian Filter on Grayscale');
        subplot(2, 3, 6), imshow(log_gray, []),
    title('Laplacian of Gaussian on Grayscale');
    else
        error('The image is not an RGB image. Required size
    is MxNx3.');
    end
```



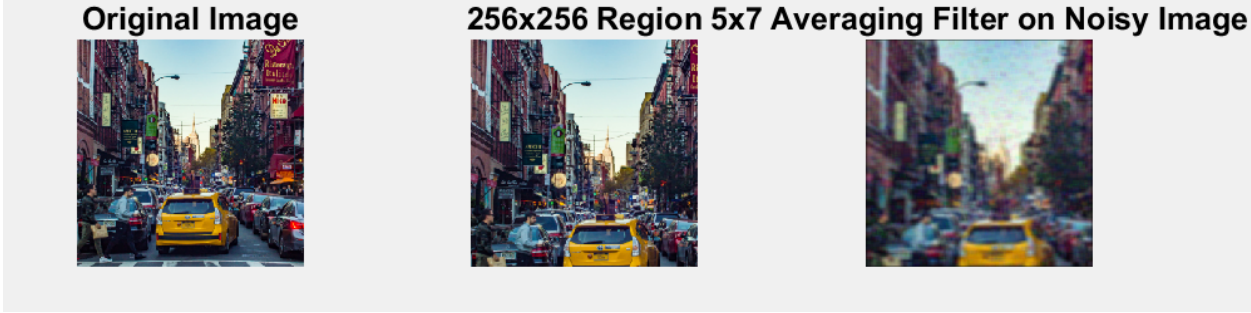
4. ***Add "salt and pepper" noise to the image.***

```
c_sp = imnoise(cc, 'salt & pepper');
imshow(c_sp)
```

5. ***Filter the image with a 3x3 filter.***

```
a3 = fspecial('average');
a4 = fspecial('average', [5,7]);
c_sp_f3 = filter2(a3, c_sp);
c_sp_f4 = filter2(a4, c_sp);
figure
```

```
subplot(1,2,1)
imshow(c_sp_f3/255)
subplot(1,2,2)
imshow(c_sp_f4/255)
```

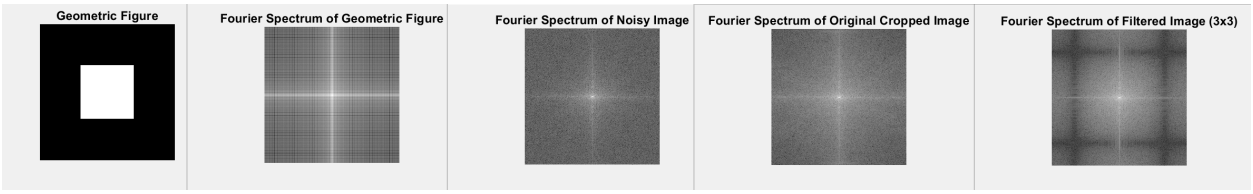


4. **FREQUENCY DOMAIN FILTERING**:

Generate a Geometric Figure and Display Its Fourier Spectrum:

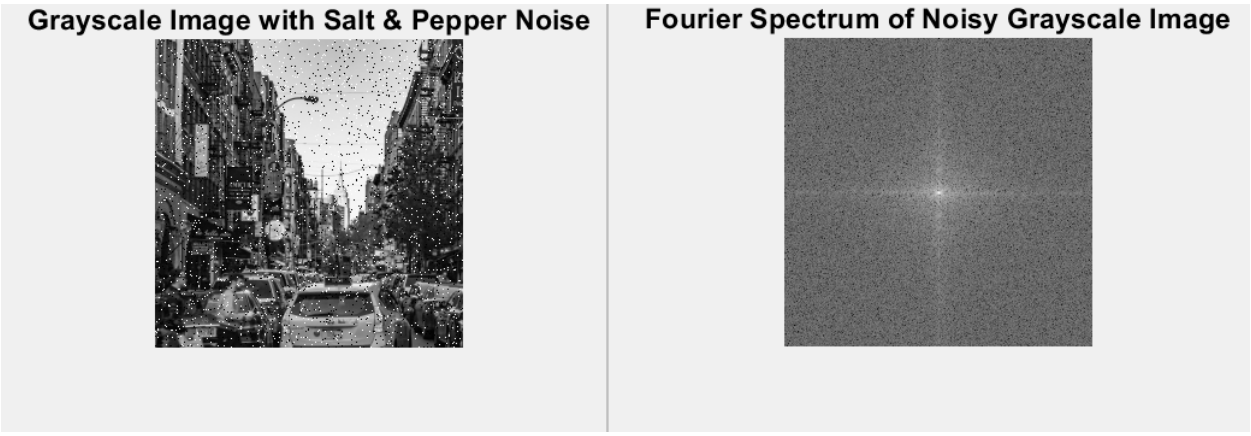
```
% Read the image from a file
c = imread('utm.tif');

% Display the dimensions of the image
sizeC = size(c);
fprintf('Dimensions of the image: %dx%dx%d\n', sizeC(1),
sizeC(2), sizeC(3));

% Ensure the image has three channels (RGB)
if sizeC(3) >= 3
    % Keep only the first three channels if there are
more than three
    c = c(:, :, 1:3);

    % Extract a 256x256 region from the original image
    if sizeC(1) >= 256 && sizeC(2) >= 256
        cc = c(1:256, 1:256, :); % Extracting the
specified region
    else
        error('The original image dimensions are too
small to extract a 256x256 region.');
    end
```

```
    % Generate a geometric figure and display its Fourier
spectrum
    a = zeros(256, 256);
    a(78:178, 78:178) = 1; % Create a square in the
center
    figure;
    imshow(a);
    title('Geometric Figure');

    % Compute the Fourier Transform and shift the zero-
frequency component to the center
    af = fftshift(fft2(a));
    figure, imshow(log(abs(af)), []);
    title('Fourier Spectrum of Geometric Figure');

    % Fourier Transform of the original cropped image
    cc_f = fftshift(fft2(rgb2gray(cc)));
    figure, imshow(log(abs(cc_f)), []);
    title('Fourier Spectrum of Original Cropped Image');

    % Add "salt and pepper" noise to the original cropped
image
    c_sp = imnoise(cc, 'salt & pepper');
    c_sp_f = fftshift(fft2(rgb2gray(c_sp)));
    figure, imshow(log(abs(c_sp_f)), []);
    title('Fourier Spectrum of Noisy Image');

    % Filter the noisy image using 3x3 average filter
    a3 = fspecial('average', 3);
    c_sp_f3 = imfilter(c_sp, a3, 'same');
    c_sp_f3_f = fftshift(fft2(rgb2gray(c_sp_f3)));
    figure, imshow(log(abs(c_sp_f3_f)), []);
    title('Fourier Spectrum of Filtered Image (3x3)');

else
    error('The image is not an RGB image. Required size
is MxNx3.');
end
```



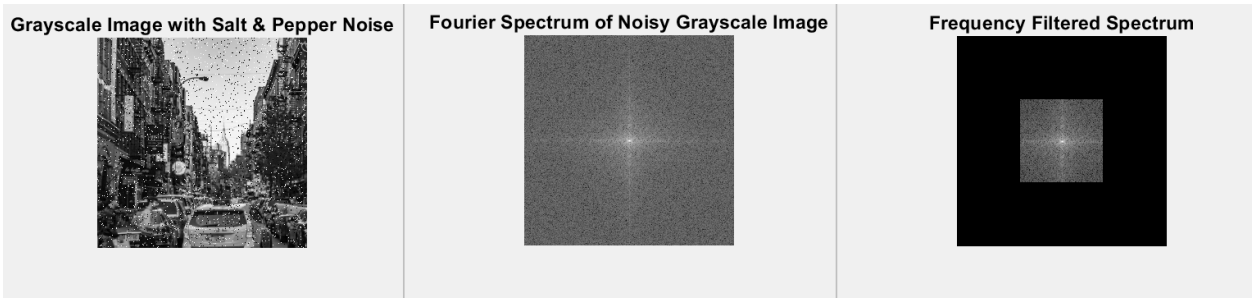
Add "Salt and Pepper" Noise to the Image and Display Its Fourier Spectrum:

```
% Ensure 'cc' is a grayscale image before filtering
if size(cc, 3) == 3
    IG = rgb2gray(cc); % Convert to grayscale if
necessary
else
    IG = cc; % If 'cc' is already grayscale
end

c_sp = imnoise(IG, 'salt & pepper'); % Add noise
imshow(c_sp);
title('Grayscale Image with Salt & Pepper Noise');

% Compute the Fourier Transform of the noisy image
cf = fftshift(fft2(c_sp));
figure;
imshow(log(abs(cf)), []); % Use log scaling for
visibility
title('Fourier Spectrum of Noisy Grayscale Image');
```



Perform Frequency Filtering by Multiplying the Geometric Figure with the Noise Spectrum to Eliminate High Frequencies:

```
% Multiply the Fourier spectrum of the geometric figure
with the noise spectrum
cf1 = cf .* a;

figure;
imshow(log(abs(cf1)), []); % Use log scaling for
visibility
title('Frequency Filtered Spectrum');
```



Perform the Inverse Fourier Transform of the Truncated Spectrum:

```
% Compute the inverse Fourier transform
cf2 = ifft2(ifftshift(cf1)); % Remember to use ifftshift
before ifft2

figure;
imshow(cf2, []);
title('Image After Inverse Fourier Transform');
```

Conclusions:

In this project, we explored the use of MATLAB for image processing, focusing on image manipulation in both the spatial and frequency domains. Initially, we learned techniques to modify image brightness and contrast through simple arithmetic operations, such as addition, subtraction, and multiplication, to adjust pixel values.

Subsequently, we experimented with various filters to reduce noise and enhance the sharpness of image details. We utilized histogram equalization to improve the contrast, thereby making subtle details more distinguishable. In dealing with the frequency domain, we analyzed Fourier spectra to gain insights into the distribution of image data across different frequencies. This approach also included practical exercises in adding and subsequently filtering out noise to explore noise reduction strategies.

The project culminated with the use of inverse Fourier transforms, allowing us to reconstruct images from their frequency spectra, which underscored our comprehensive study of essential image processing techniques.