TECHNICAL UNIVERSITY OF MOLDOVA FACULTY OF COMPUTERS INFORMATICS AND MICROELECTRONICS

DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATION

Laboratory work 4

Subject: Linear digital image processing methods

Done by: **Konjevic Alexandra, st. gr. FAF-213**

Verified by: **Railean Serghei, university lecturer**

CHIŞINĂU, 2024

TASK OF THE LABORATORY WORK

The purpose of this lab work is to explore various aspects of Fourier Transform and signal processing techniques. It involves tasks such as generating discrete signals, performing Fourier Transform, analyzing frequency spectra, determining magnitudes and phases, modeling rectangular pulses, filtering noise signals, and examining continuous Fourier Transform. The tasks cover practical applications of Fourier analysis in signal processing and aim to deepen understanding of frequency domain representations of signals.

THEORETICAL CONSIDERATIONS

Arithmetic operations on images involve applying a simple function to each pixel value. This function maps the range of pixel values (usually 0 to 255) onto itself. Simple functions include adding or subtracting a constant value from each pixel or multiplying each pixel by a constant. In each case, rounding might be necessary to ensure the results are integers within the 0 to 255 range. This can be achieved by rounding the result first (if necessary) to obtain an integer and then "clipping" the values by setting: pixel value = min(max(0,rounded value),255)

We can gain understanding of how these operations affect an image by visualizing the old grayscale values versus the new values. For instance, adding or subtracting 128 from each pixel brightens or darkens the image, respectively.

Histogram Equalization:

Histogram equalization enhances the contrast of an image by spreading out the intensity levels. It adjusts the intensity distribution of an image by transforming its histogram. A histogram represents the frequency of occurrence of each intensity level in an image. Histogram equalization can be visualized using the imhist function in MATLAB.

Spatial Filtering:

Spatial filtering is an extension of arithmetic operations, where a function is applied to a neighborhood of each pixel. It involves moving a "mask" or a rectangle (usually with odd-length sides) or another shape over the given image. As we do this, we create a new image whose pixel values are calculated from the grayscale values under the mask. The combination of the mask and the function is called a filter. If the function is a linear function of all the grayscale values in the mask, the filter is called a linear filter. We can implement a linear filter by multiplying all elements in the mask by the corresponding elements in the neighborhood and summing all these products.

Low-Pass and High-Pass Filters:

Low-pass filters reduce or eliminate high-frequency components, while high-pass filters reduce or eliminate low-frequency components. For example, a median filter is a low-pass filter as it tends to blur edges, while the Laplacian filter is a high-pass filter.

1. Gamma curve manipulations

1.1 Show an image from your file system.

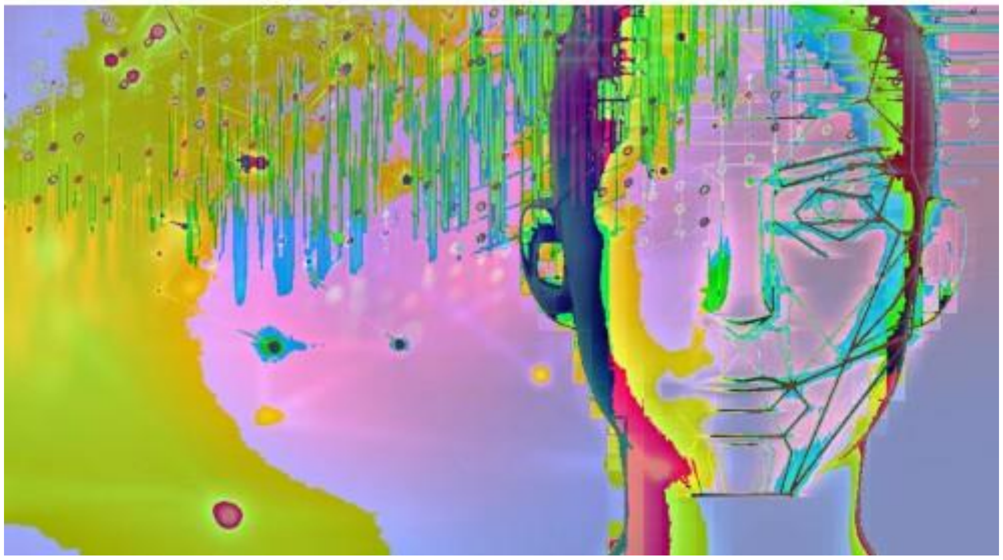
```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

original_image_path = './images/img.jpg'
saved_image_path  =  './images/my_image.tif'

image  =  Image.open(original_image_path)
image.save(saved_image_path)
image  =  Image.open(saved_image_path)

plt.figure()
plt.imshow(image)
plt.title('Original  Image')
plt.axis('off')

plt.show()
```

Original Image

1.2 Add 128 to each pixel of the image.

```
cc = np.array(image)
c1 = cc + 128

modified_image_c1  =  Image.fromarray(c1.astype(np.uint8))

plt.figure()
plt.imshow(modified_image_c1)
plt.title('Image with 128 Added to Each Pixel')
plt.axis('off')

plt.show()
```

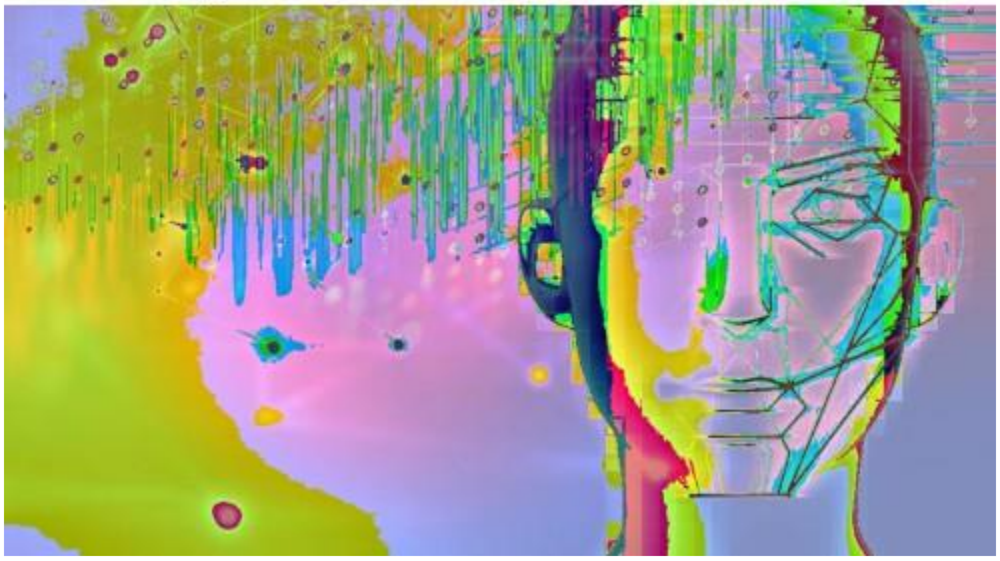
Image with 128 Added to Each Pixel



1.3 Subtract 128 from each pixel of the image.

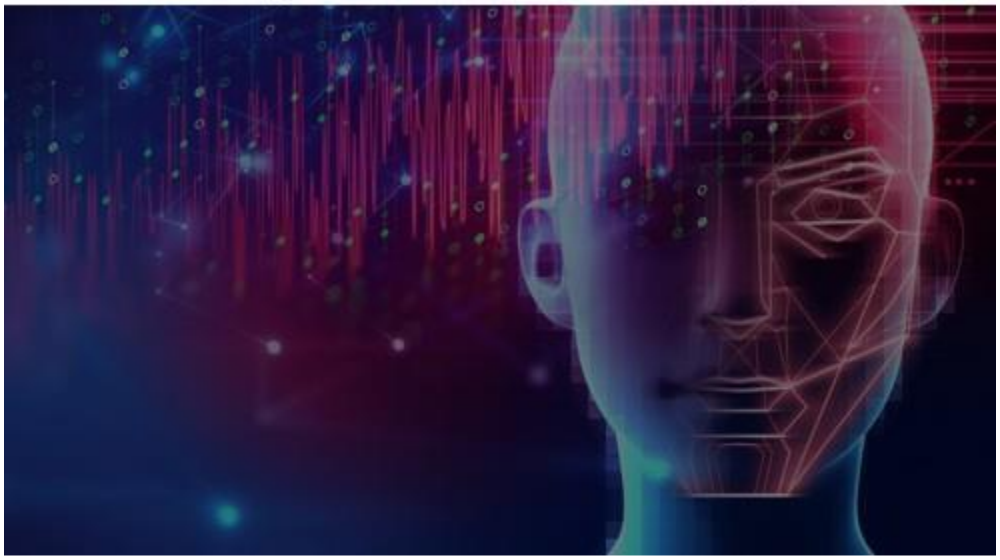
```
c2 = cc - 128

modified_image_c2  =  Image.fromarray(c2.astype(np.uint8))

plt.figure()
plt.imshow(modified_image_c2)
plt.title('Image with 128 Subtracted from Each Pixel')

plt.axis('off')
plt.show()
```

Image with 128 Subtracted from Each Pixel



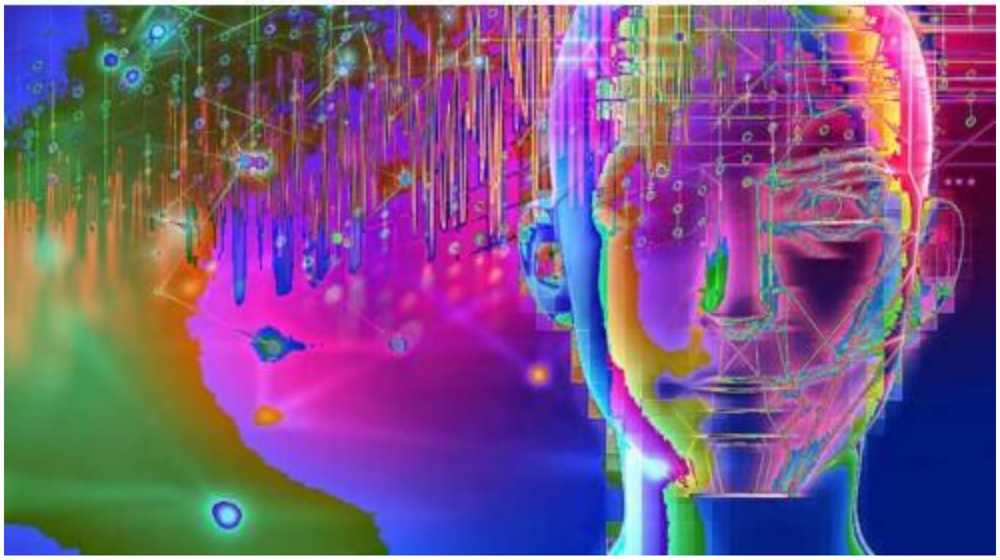
1.4 Divide each pixel of the image by 2

```
c3 = cc / 2
modified_image_c3 = Image.fromarray(c3.astype(np.uint8))
plot()
```



1.5 Multiply each pixel of the image by 2.

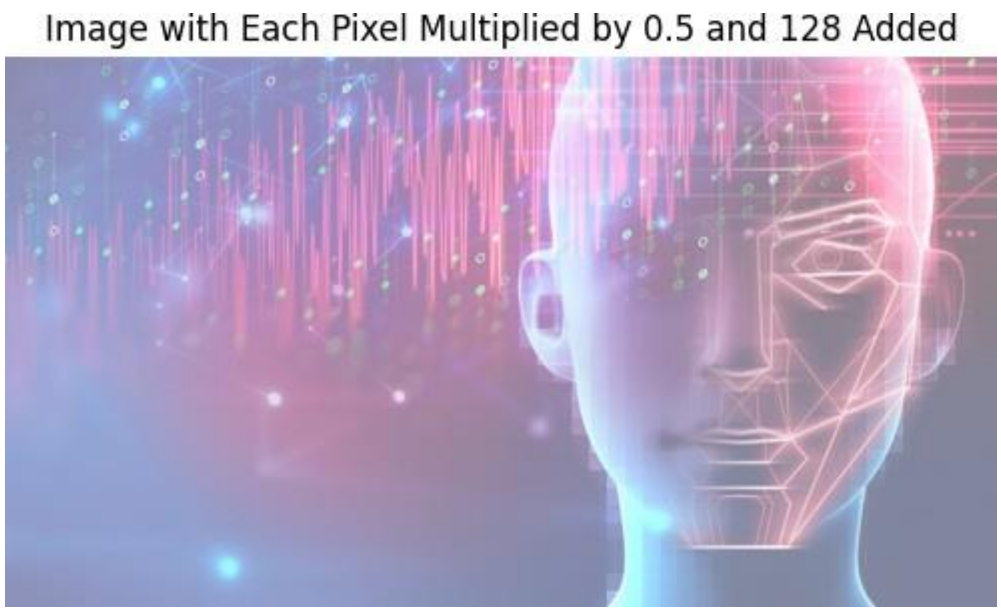
```
c4 = cc * 2

modified_image_c4  =  Image.fromarray(c4.astype(np.uint8))

plt.figure()
plt.imshow(modified_image_c4)
plt.title('Image with Each Pixel Multiplied by 2')
plt.axis('off')
plt.show()
```



1.6 Divide each pixel of the image by half then add 128

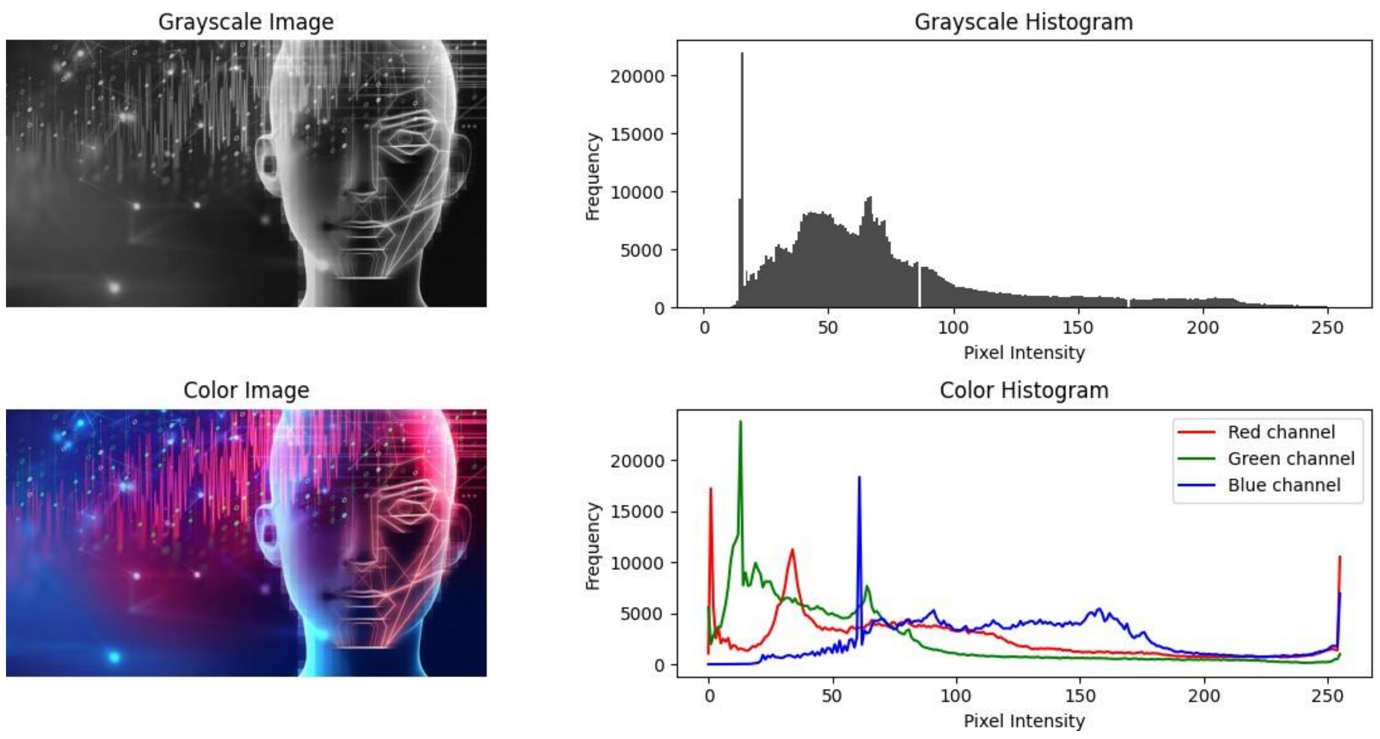
```
c5 = cc * 0.5 + 128
```

```
modified_image_c4  =  Image.fromarray(c5.astype(np.uint8))

plt.figure()
plt.imshow(modified
_image_c4)
plt.title('Image with Each Pixel Multiplied by 0.5 and
128 Added')plt.axis('off')
```



2.1 Show an image from your file system and its histogram

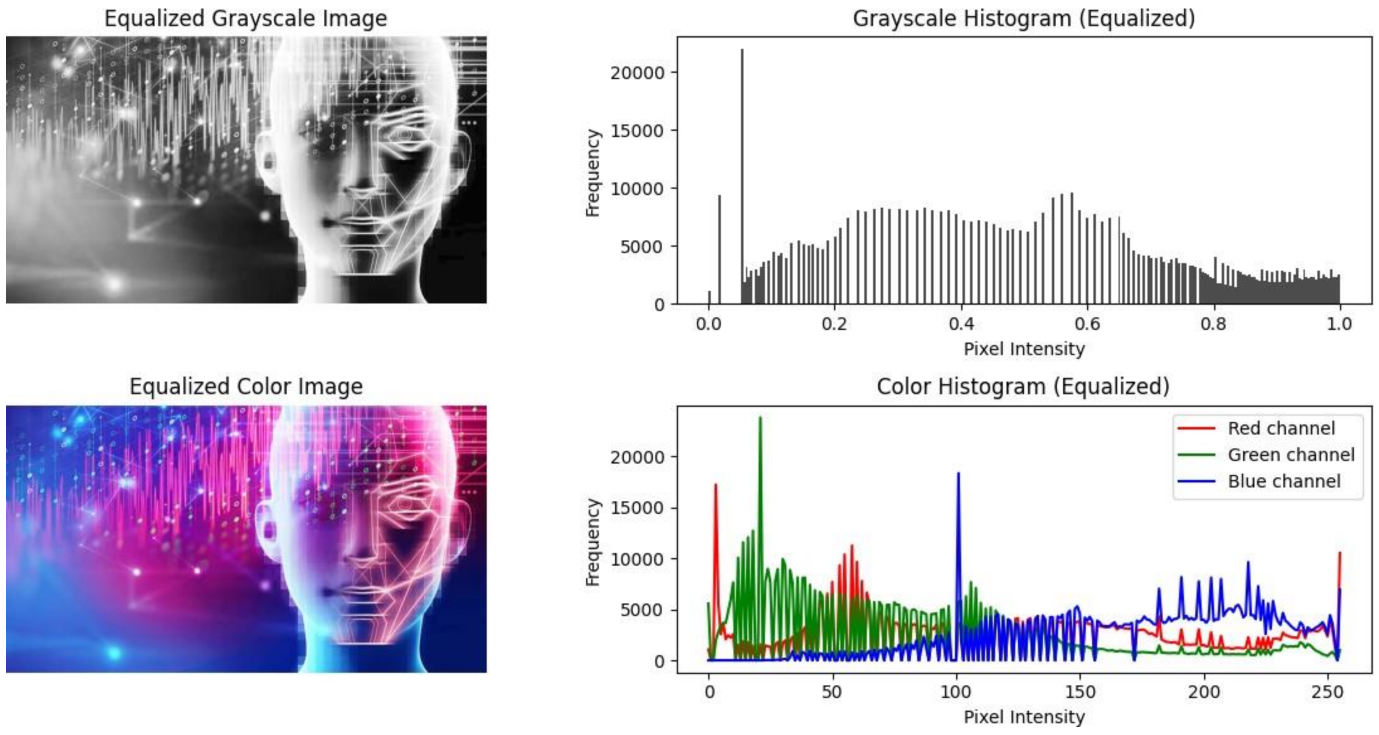
```
gray_image  =  image.convert('L')
cc_gray = np.array(gray_image)
# Display the grayscale image and its histogram
plt.figure(figsize=(12,  6))

# Grayscale Image and its histogram
plt.subplot(2,  2,  1)
plt.imshow(gray_image,  cmap='gray')
plt.title('Grayscale  Image')
plt.axis('off')

plt.subplot(2,  2,  2)
plt.hist(cc_gray.ravel(),  bins=256,  color='k',  alpha=0.7)
plt.title('Grayscale Histogram')
plt.xlabel('Pixel  Intensity')
plt.ylabel('Frequency')

# Color Image and its histogram
plt.subplot(2,  2,  3)
plt.imshow(image)
plt.title('Color Image')
plt.axis('off')

plt.subplot(2,  2,  4)
R  =  np.histogram(cc[:,:,0],  bins=256,  range=[0,256])[0]
G  =  np.histogram(cc[:,:,1],  bins=256,  range=[0,256])[0]
B  =  np.histogram(cc[:,:,2],  bins=256,  range=[0,256])[0]
plt.plot(R,  color='r')
plt.plot(G,  color='g')
plt.plot(B,  color='b')
plt.title('Color Histogram')
plt.xlabel('Pixel  Intensity')
plt.ylabel('Frequency')
plt.legend(['Red channel', 'Green channel', 'Blue channel'])

plt.tight_layout()
plt.show()
```



2.2 Equalize the histogram of the image and show the result.

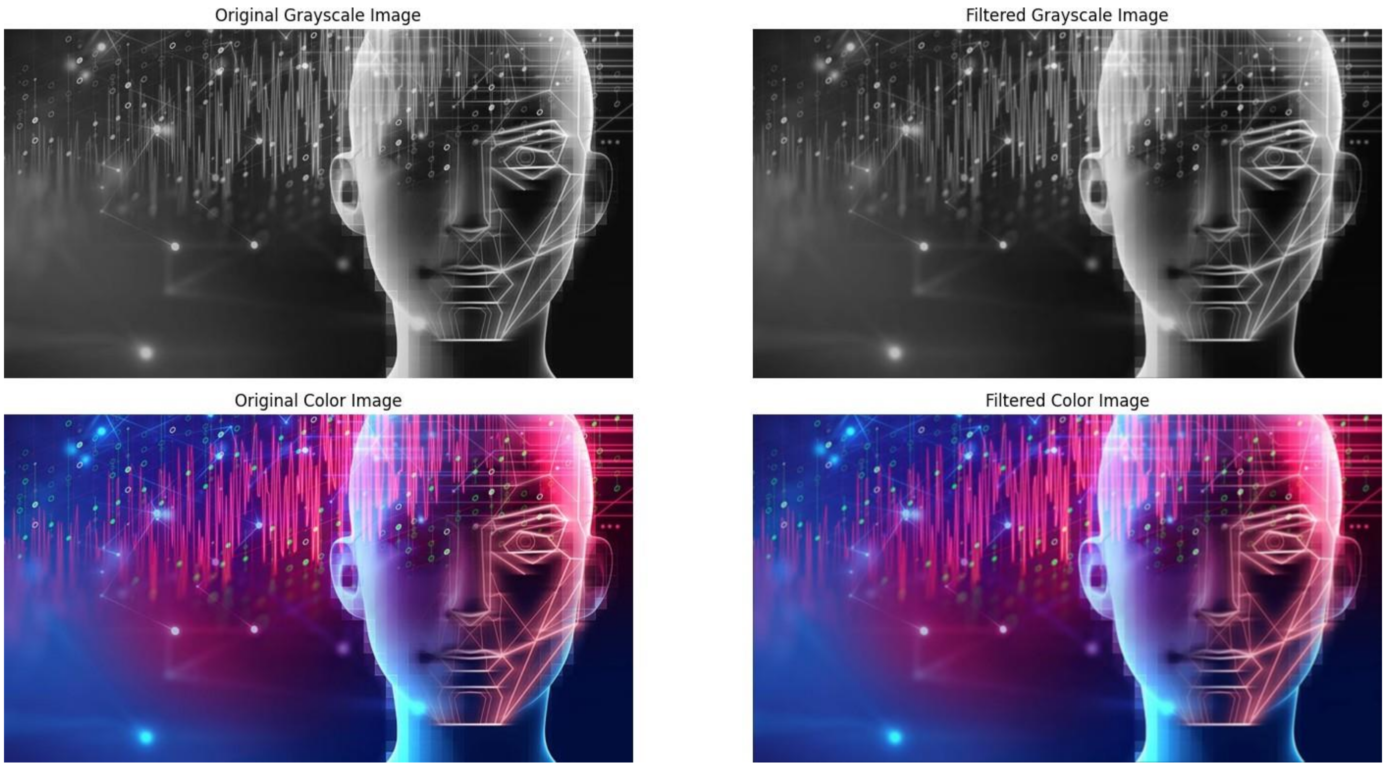
```
from skimage import exposure

# Perform histogram equalization for grayscale image
h_gray = exposure.equalize_hist(cc_gray)

# Perform histogram equalization for color image
I2 = exposure.equalize_hist(cc)

# Display the equalized grayscale image and its histogram
plt.figure(figsize=(12, 6))

# Equalized grayscale image
plt.subplot(2, 2, 1)
plt.imshow(h_gray, cmap='gray')
plt.title('Equalized Grayscale
Image')plt.axis('off')

# Grayscale histogram after equalization
plt.subplot(2, 2, 2)
plt.hist(h_gray.ravel(), bins=256, color='k',
alpha=0.7)plt.title('Grayscale Histogram
(Equalized)') plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')

# Display the equalized color image and its histogram
plt.subplot(2, 2, 3)
plt.imshow(I2)
plt.title('Equalized Color
Image')plt.axis('off')

# Color histogram after equalization
plt.subplot(2, 2, 4)
R = np.histogram(I2[:,:,0], bins=256, range=[0,1])[0]
```

```
G = np.histogram(I2[:,:,1], bins=256, range=[0,1])[0]
B = np.histogram(I2[:,:,2], bins=256, range=[0,1])[0]
plt.plot(R, color='r')
plt.plot(G,  color='g')
plt.plot(B,  color='b')
plt.title('Color Histogram (Equalized)')
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
plt.legend(['Red channel', 'Green channel', 'Blue channel'])

plt.tight_layout()
plt.show()
```



3.1 Apply a 3x3 filter to the image and show the result.

```
from scipy.signal import convolve2d

# Perform filtering for grayscale image
f1 = np.ones((3, 3)) / 9.0
cf1_gray = convolve2d(cc_gray, f1, mode='same')

# Separate channels for the color image
R = cc[:,:,0]
G = cc[:,:,1]
B = cc[:,:,2]

# Convolve each channel separately
cf1_color_R = convolve2d(R, f1,
mode='same')cf1_color_G = convolve2d(G,
f1, mode='same')cf1_color_B =
convolve2d(B, f1, mode='same')

# Stack the filtered channels back into a color image
cf1_color = np.stack([cf1_color_R, cf1_color_G, cf1_color_B], axis=2)

# Display original and filtered images
plt.figure(figsize=(16,  8))

# Original grayscale image
plt.subplot(2, 2, 1)
plt.title('Original Grayscale Image')
plt.axis('off')

# Filtered grayscale image
plt.subplot(2, 2, 2)
plt.imshow(cf1_gray, cmap='gray')
plt.title('Filtered Grayscale Image')
plt.axis('off')

# Original color image
plt.subplot(2, 2, 3)
plt.imshow(cc)
plt.title('Original Color Image')
plt.axis('off')

# Filtered color image
plt.subplot(2, 2, 4)
plt.imshow(cf1_color.astype(np.uint8))
plt.title('Filtered Color Image')
plt.axis('off')

plt.tight_layout()
plt.show()
```

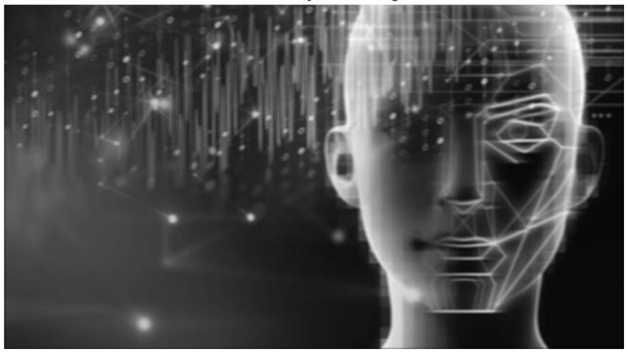


3.2 a Apply a 5x7 filter to the image and show the result

```
def filter(x, y):

    # Perform filtering for grayscale image
    f1_gray = np.ones((x, y)) / (x * y)
    cf1_gray = convolve2d(cc_gray, f1_gray, mode='same')

    # Perform filtering for color image
    f1_color = np.ones((5, 7)) / (5 * 7)
    cf1_color = np.zeros_like(cc,
    dtype=float)
    for i in range(3):  # Apply filter to each color channel separately
        cf1_color[:,:,i] = convolve2d(cc[:,:,i], f1_color, mode='same')
    plt.figure(figsize=(16, 8))

    # Original grayscale image
    plt.subplot(2, 2, 1)
    plt.imshow(cc_gray, cmap='gray')
    plt.title('Original Grayscale Image')
    plt.axis('off')

    # Filtered grayscale image
    plt.subplot(2, 2, 2)
    plt.imshow(cf1_gray, cmap='gray')
    plt.title('Filtered Grayscale Image')
    plt.axis('off')

    plt.tight_layout()
    plt.show()

    # Display filtered color image
    plt.figure(figsize=(12, 6))

    # Original color image
    plt.subplot(1, 2, 1)
    plt.imshow(cc)
    plt.title('Original Color Image')
    plt.axis('off')

    # Filtered color image
    plt.subplot(1, 2, 2)
    plt.imshow(cf1_color.astype(np.uint8))
    plt.title('Filtered Color Image')
    plt.axis('off')

    plt.tight_layout()
    plt.show()

filter(5, 7)
```

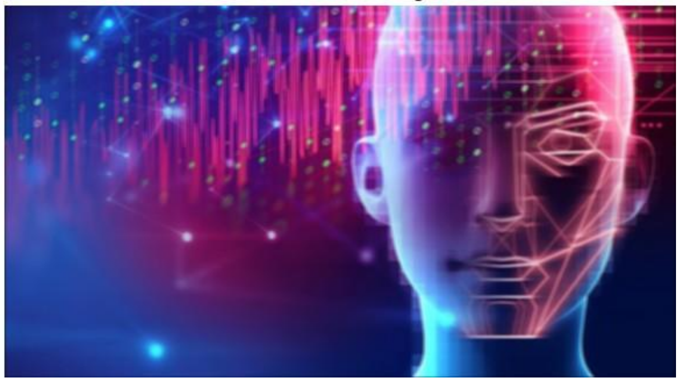
Original Grayscale Image

Filtered Grayscale Image



Original Color Image

Filtered Color Image

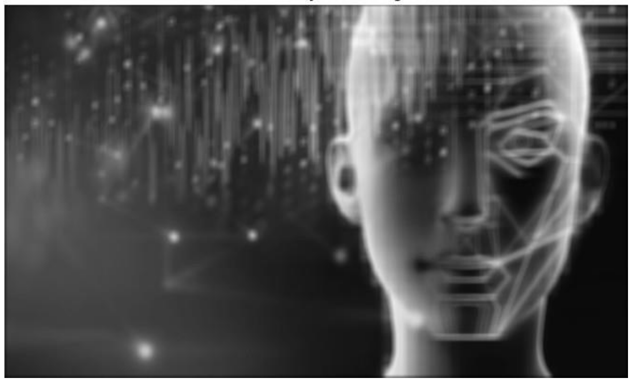


3.2 b Apply a 11x11 filter to the image and show the result

```
filter(11, 11)
```

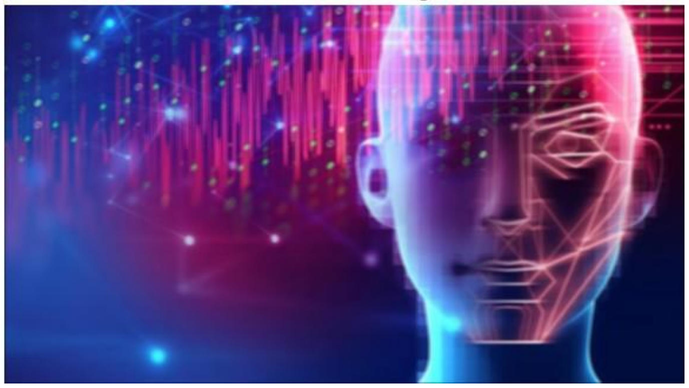
Original Grayscale Image

Filtered Grayscale Image



Original Color Image

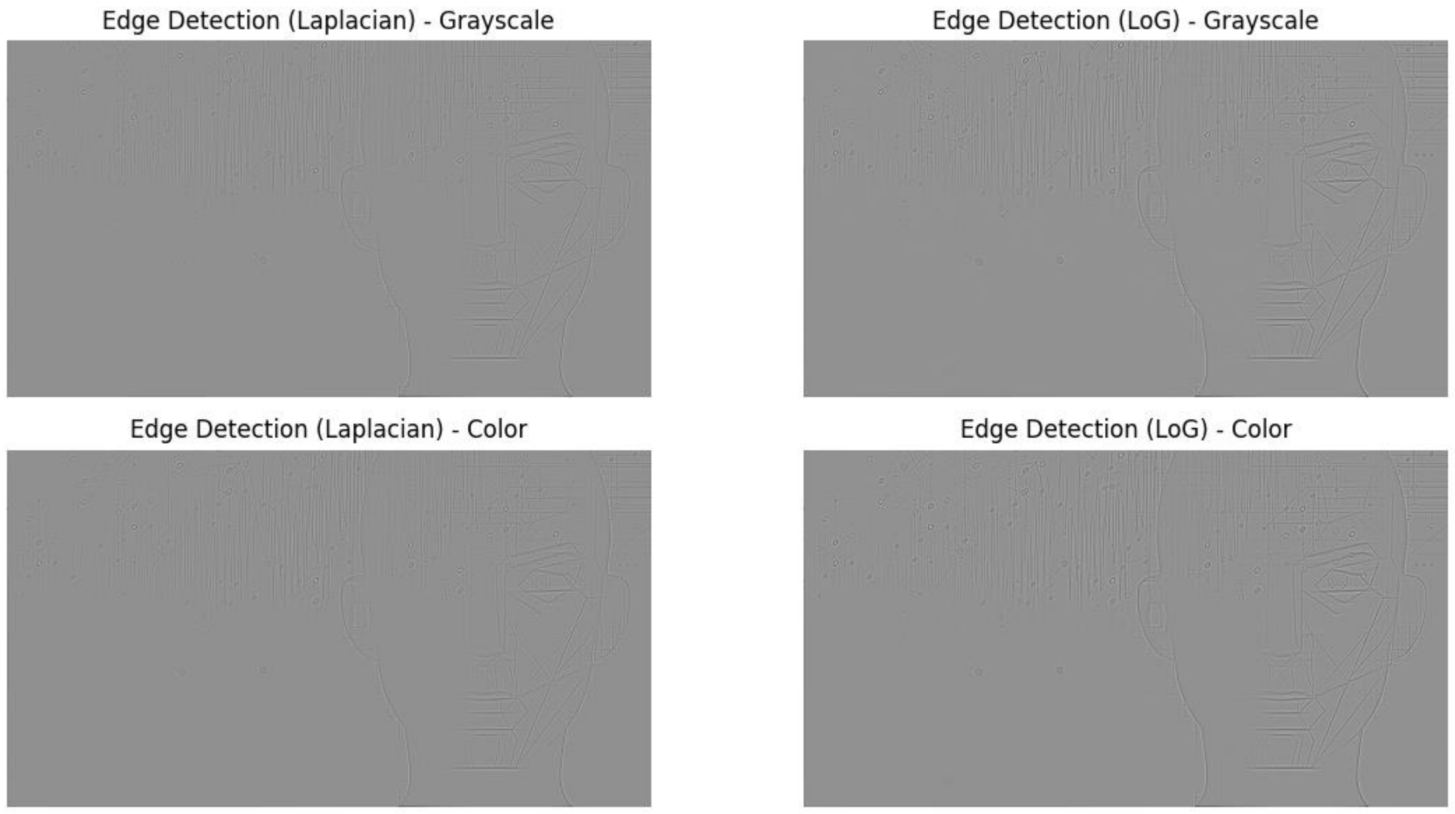
Filtered Color Image



3.3 Detect the edges of the images using a Laplacian and then a logarithmic Gaussian filter

```
from skimage import color

# Perform edge detection for grayscale
image
f_laplacian = np.array([[0, 1, 0], [1, -4,
1], [0, 1, 0]])
cf2_gray = convolve2d(cc_gray,
f_laplacian, mode='same')

# Perform Laplacian of Gaussian for
grayscale image
f_log = np.array([[0, 0, 1, 0, 0],
                  [0, 1, 2, 1, 0],
                  [1, 2, -16, 2, 1],
                  [0, 1, 2, 1, 0],
                  [0, 0, 1, 0, 0]])
cf3_gray = convolve2d(cc_gray, f_log,
mode='same')

# Convert the color image to grayscale
IG = color.rgb2gray(cc)

# Perform edge detection for color image
cf2_color = convolve2d(IG, f_laplacian,
mode='same')

# Perform Laplacian of Gaussian for color
image
cf3_color = convolve2d(IG, f_log,
mode='same')
```

```
# Display edge detection results
plt.figure(figsize=(12, 6))

# Edge detection for grayscale image
plt.subplot(2, 2, 1)
plt.imshow(cf2_gray / 155, cmap='gray')
plt.title('Edge Detection (Laplacian) -
Grayscale')
plt.axis('off')

plt.subplot(2, 2, 2)
plt.imshow(cf3_gray / 155, cmap='gray')
plt.title('Edge Detection (LoG) -
Grayscale')
plt.axis('off')

# Edge detection for color image
plt.subplot(2, 2, 3)
plt.imshow(cf2_color / 155, cmap='gray')
plt.title('Edge Detection (Laplacian) -
Color')
plt.axis('off')

plt.subplot(2, 2, 4)
plt.imshow(cf3_color / 155, cmap='gray')
plt.title('Edge Detection (LoG) - Color')
plt.axis('off')

plt.tight_layout()
plt.show()
```


Edge Detection (Laplacian) - Grayscale
Edge Detection (LoG) - Grayscale
Edge Detection (Laplacian) - Color
Edge Detection (LoG) - Color

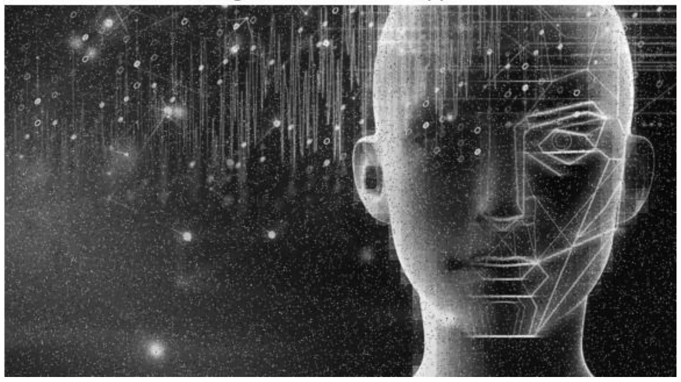
3.4 Add salt and pepper noise to the image and show the difference.

```
from skimage.util import random_noise

# Add "salt and pepper" noise to grayscale image
c_sp_gray = random_noise(cc_gray, mode='s&p')

# Convert the color image to grayscale
IG = color.rgb2gray(cc)

# Add "salt and pepper" noise to grayscale image
c_sp_color = random_noise(IG, mode='s&p')

# Display images with "salt and pepper" noise
plt.figure(figsize=(12,  6))

# Image with "salt and pepper" noise - grayscale
plt.subplot(1, 2, 1)
plt.imshow(c_sp_gray,
cmap='gray')
```

```
 plt.axis('off')

 # Image with "salt and pepper" noise - color
 plt.subplot(1, 2, 2)
 plt.imshow(c_sp_color,  cmap='gray')
 plt.title('Color Image with "Salt and Pepper" Noise')
 plt.axis('off')

 plt.tight_layout()
 plt.show()
```

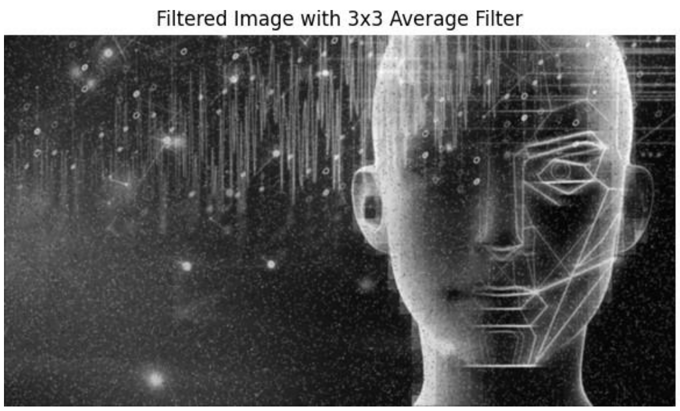
Grayscale Image with "Salt and Pepper" Noise

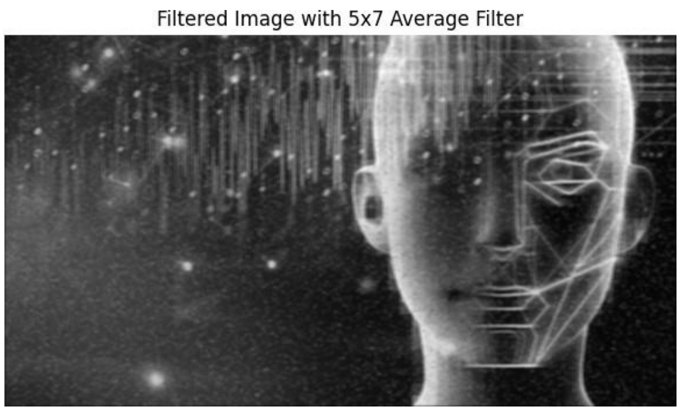
Color Image with "Salt and Pepper" Noise



3.5 Apply a 3x3 median filter to the image and show the result.

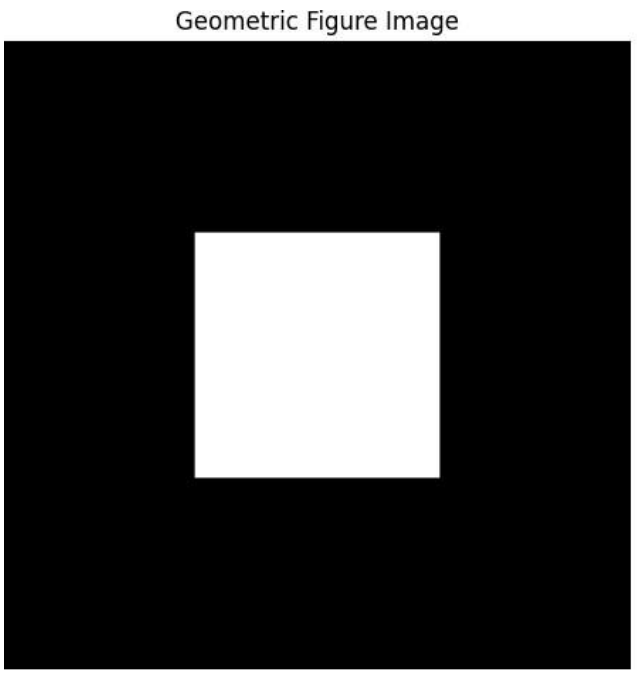
```
# Add "salt and pepper" noise to grayscale image
c_sp_gray = random_noise(cc_gray, mode='s&p')

# Convert the color image to grayscale
IG = color.rgb2gray(cc)

# Add "salt and pepper" noise to grayscale image
c_sp_color = random_noise(IG, mode='s&p')

# Define 3x3 and 5x7 average filters
a3 = np.ones((3, 3)) / 9
a4 = np.ones((5, 7)) / 35  # Adjusted to maintain sum of 1

# Filter the noisy images with the average
filtersc_sp_f3 = convolve2d(c_sp_gray, a3,
mode='same')c_sp_f4 = convolve2d(c_sp_gray, a4,
mode='same')

# Display filtered images
plt.figure(figsize=(12,  6))

# Filtered image with 3x3 average filter
plt.subplot(1, 2, 1)
plt.imshow(c_sp_f3 / 255,
cmap='gray')
plt.title('Filtered Image with 3x3 Average
Filter')plt.axis('off')

# Filtered image with 5x7 average filter
plt.subplot(1, 2, 2)
plt.imshow(c_sp_f4 / 255,
cmap='gray')
plt.title('Filtered Image with 5x7 Average
Filter')plt.axis('off')
```

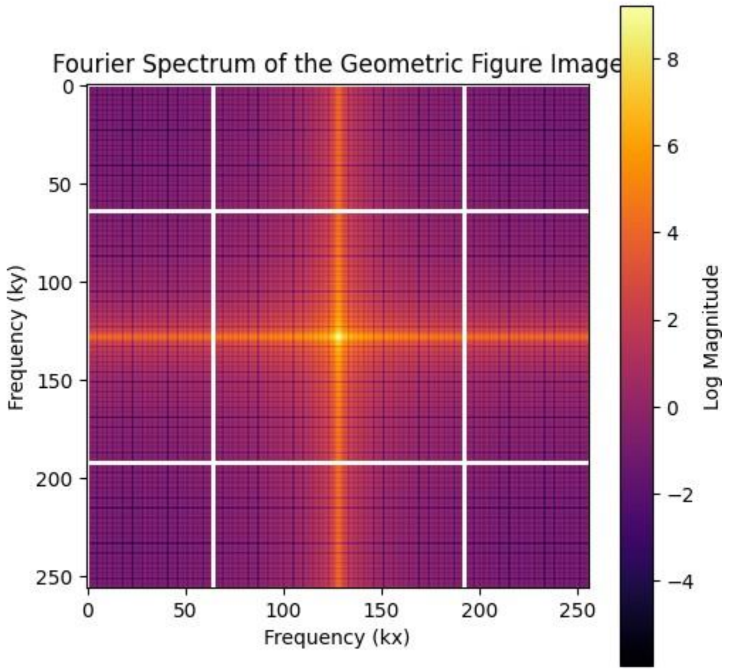




4.1 Generate a geometric pattern and apply a Fourier transform to it.

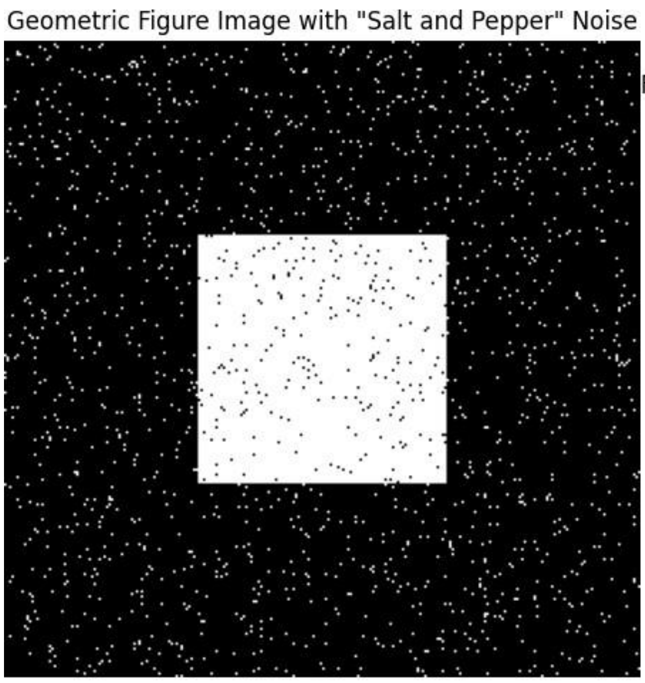
```
from scipy.fft import fftshift, fft2

# Generate the geometric figure image
a = np.zeros((256, 256))
a[78:178, 78:178] = 1

# Compute the Fourier transform and shift it
af = fftshift(fft2(a))

# Plot the original image
plt.figure(figsize=(10,  5))

plt.subplot(1, 2, 1)
plt.imshow(a,  cmap='gray')
plt.title('Geometric  Figure  Image')
plt.axis('off')

# Plot the Fourier spectrum using heatmap for a colored representation
plt.subplot(1, 2, 2)
plt.title('Fourier Spectrum of the Geometric Figure Image')
plt.xlabel('Frequency (kx)')
plt.ylabel('Frequency (ky)') plt.imshow(np.log(np.abs(af)),
cmap='inferno')plt.colorbar(label='Log Magnitude')

plt.tight_layout()
plt.show()
```

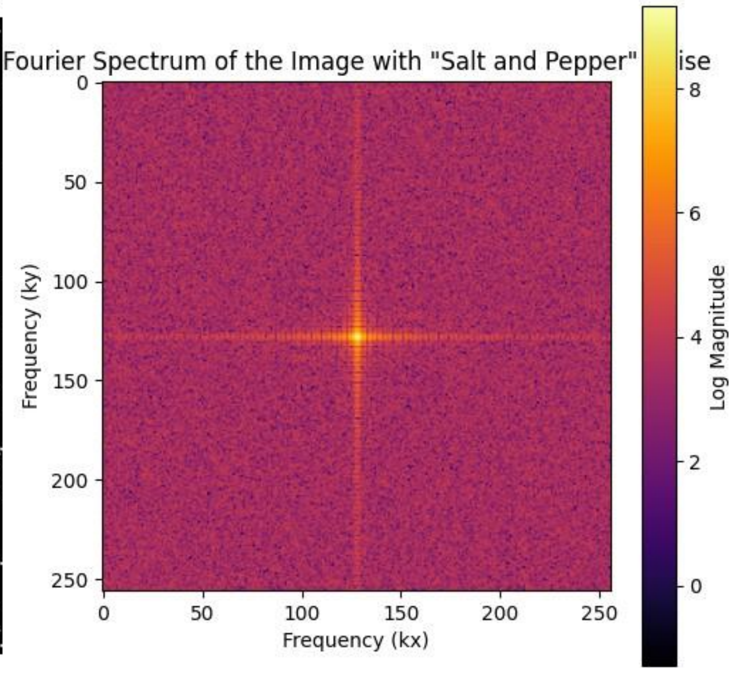




4.2 Add salt and pepper noise to the image and show the fft.

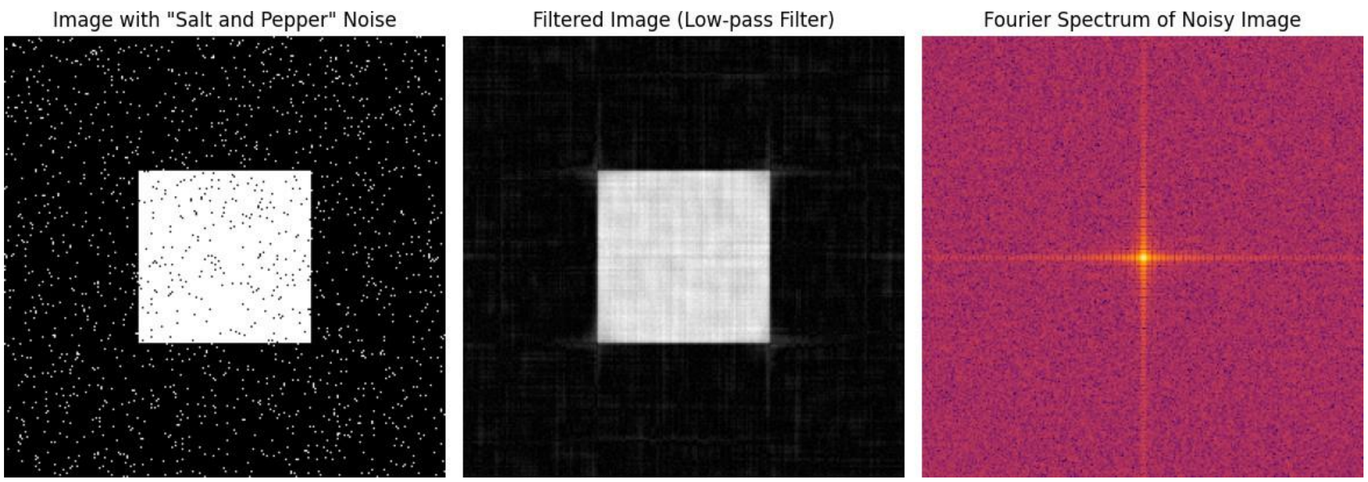
```
# Add "salt and pepper" noise to the image
c_sp = random_noise(a, mode='s&p')

# Compute the Fourier transform and shift it
cf = fftshift(fft2(c_sp))

# Plot the image with "salt and pepper" noise
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(c_sp, cmap='gray')
plt.title('Geometric Figure Image with "Salt and Pepper" Noise')
plt.axis('off')

# Plot the Fourier spectrum using heatmap for a colored representation
plt.subplot(1, 2, 2)
plt.title('Fourier Spectrum of the Image with "Salt and Pepper" Noise')
plt.xlabel('Frequency (kx)')
plt.ylabel('Frequency (ky)') plt.imshow(np.log(np.abs(cf)),
cmap='inferno')plt.colorbar(label='Log Magnitude')

plt.tight_layout()
plt.show()
```





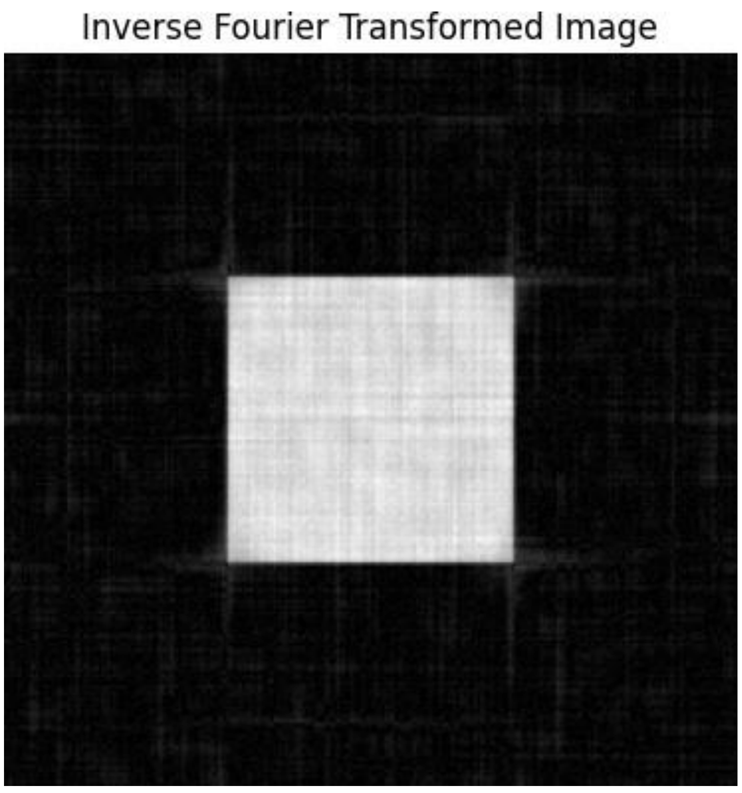
4.3 Apply a low-pass filter to the image and show the spectrum

```
from scipy.fft import ifft2

# Compute the absolute values of the Fourier coefficients
abs_cf = np.abs(cf)

# Perform frequency domain filtering by eliminating high frequencies
threshold = 100
filtered_cf = cf * (abs_cf > threshold)

# Shift the filtered Fourier spectrum back and compute the inverse Fourier transform
filtered_cf_shifted = fftshift(filtered_cf)
c_filtered  = np.abs(ifft2(filtered_cf_shifted))

# Plot the filtered image and its Fourier spectrum
plt.figure(figsize=(12, 6))

# Plot the original image with "salt and pepper" noise
plt.subplot(1, 3, 1)
plt.imshow(c_sp, cmap='gray')
plt.title('Image with "Salt and Pepper" Noise')
plt.axis('off')

# Plot the filtered image
plt.subplot(1, 3, 2) plt.imshow(c_filtered
/ 255, cmap='gray')
plt.title('Filtered Image (Low-pass Filter)')
plt.axis('off')

# Plot the Fourier spectrum of the noisy image
plt.subplot(1, 3, 3) plt.imshow(np.log(abs_cf),
cmap='inferno') plt.title('Fourier Spectrum of
Noisy Image')plt.axis('off')

plt.tight_layout()
plt.show()
```

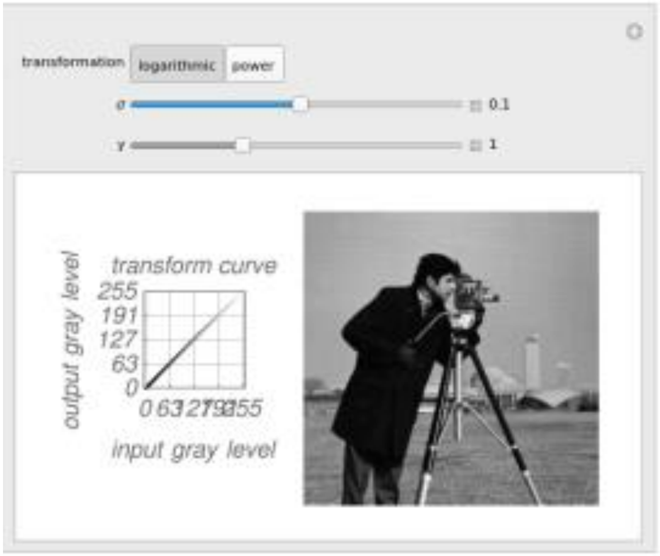


4.4 Perform the inverse Fourier transform and show the result.

```
# Compute the Fourier transform of the noisy image and shift it
cf = fftshift(fft2(c_sp))

# Perform frequency domain filtering by eliminating high frequencies
threshold = 100
filtered_cf = cf * (np.abs(cf) > threshold)

# Perform inverse Fourier transform of the truncated spectrum
cf2 = ifft2(filtered_cf)

# Plot the inverse Fourier transformed image
plt.imshow(np.abs(cf2) / 155, cmap='gray')
plt.title('Inverse Fourier Transformed Image')
plt.axis('off')
plt.show()
```



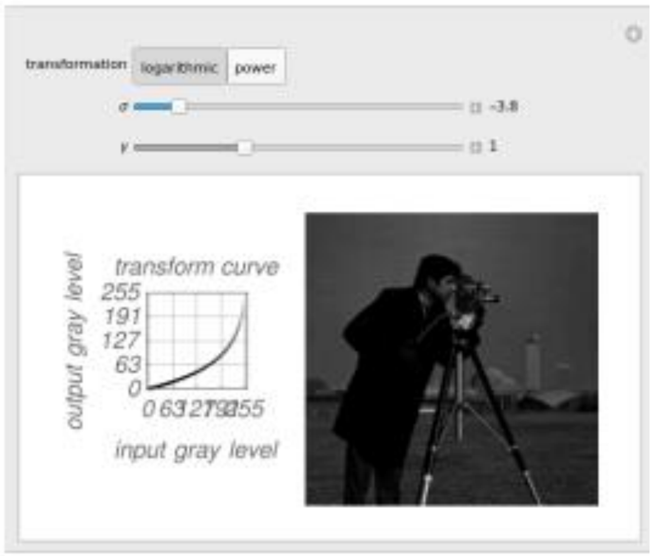
1.7 Analyze the gamma curve transform using the following tool: https://demonstrations.wolfram.com/TransformationsOfGrayLevelsInAnImage/



transformation [✓ logarithmic] power

σ 0.1

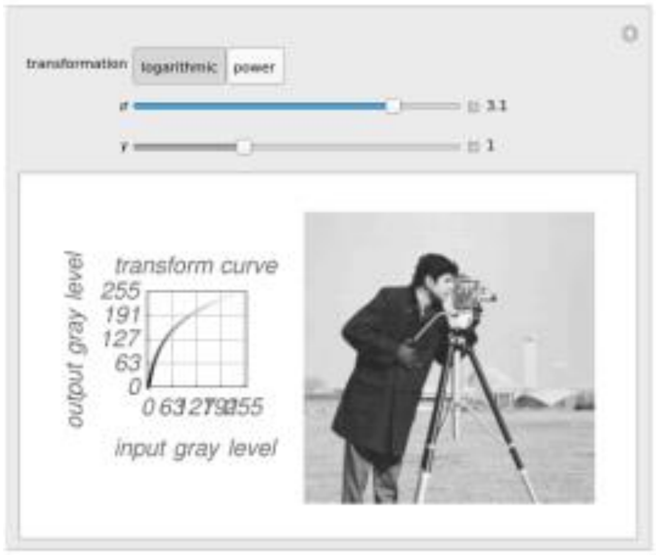
γ 1



transformation [✓ logarithmic] power

σ -3.8

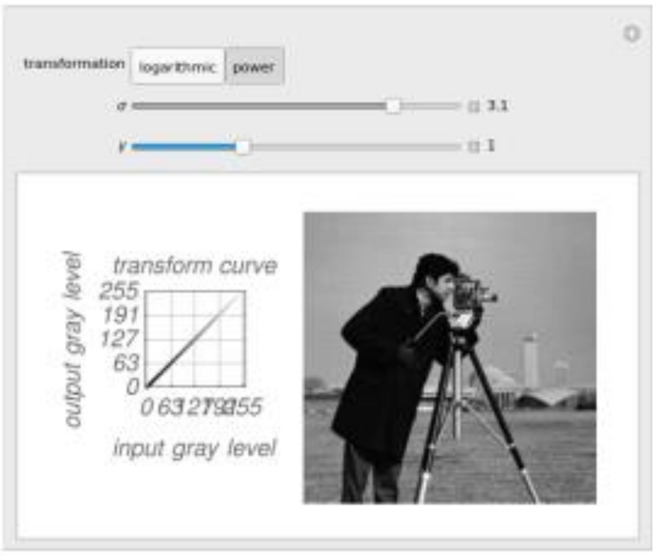
γ 1



transformation [✓ logarithmic] power

σ 3.1

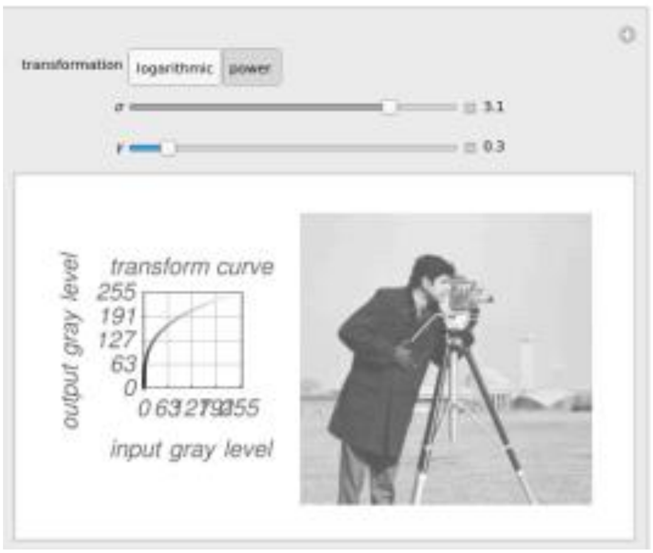
γ 1



transformation logarithmic [✓ power]

σ 3.1

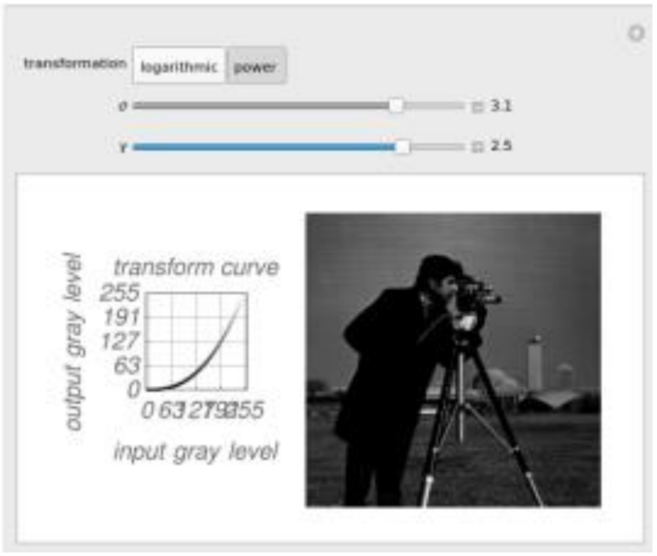
γ 1



transformation logarithmic [✓ power]

σ 3.1

γ 0.3



transformation logarithmic [✓ power]

σ 3.1

γ 2.5

CONCLUSION

In conclusion, image processing involves a variety of techniques aimed at enhancing, analyzing, and manipulating digital images. Arithmetic operations allow for simple modifications of pixel values, such as adjusting brightness and contrast. Histogram equalization enhances image contrast by redistributing intensity levels. Spatial filtering extends these operations by applying functions to neighborhoods of pixels, allowing for tasks like blurring or edge detection.

Frequency domain filtering involves transforming images into the frequency domain via the Fourier transform, where operations such as filtering out high frequencies can be performed to remove noise or enhance specific features. Combining both spatial and frequency domain techniques provides a powerful toolkit for image enhancement and analysis.

Understanding these fundamental concepts and techniques equips us with the necessary tools to process images for a wide range of applications, from medical imaging to satellite imagery analysis and beyond.