NITTE EDUCATION TRUST

NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY
An Autonomous Institution Approved by UGC/AICTE/Govt. of Karnataka
Accredited by NBA (Tier-I) and NAAC 'A+' Grade
Affiliated to Visvesvaraya Technological University, Belagavi
Post Box No. 6429, Yelahanka, Bengaluru-560064, Karnataka, INDIA

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

PRACTIAL RECORD BOOK

DIGITAL IMAGE PROCESSING AND COMPUTER VISION
21ADL66

NAME	: ABHISHEK VERMA
USN	: 1NT21AD002
YEAR/SEM	: 3RD / 6TH
SECTION	: A
BRANCH	: ARTIFICIAL INTELLIGENCE and DATA SCIENCE

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

CERTIFICATE

This is to certify that Mr./Mrs. ABHISHEK VERMA

studying in 6TH semester AI&DS Branch has satisfactorily completed the course of experiments in DIGITAL IMAGE PROCESSING AND COMPUTER VISION Laboratory for the academic session 2023-2024 as prescribed by the Visvesvaraya Technological University.

Register Number of the Candidate:

1NT21AD002

Signature of the Staff In-charge

Date:

Signature of HOD

Date:

Sl. No	**Program / Exercise**
1.	Implementation of Relationships between Pixels Neighbor of 4,8 and Diagonal point
2.	a. Simulation and Display of an Image, Negative of an Image (Binary & Gray Scale) b. Display color Image, find its complement and convert to gray scale
3.	a. Implementation of Transformations of an Image (Scaling & Rotation) b. Display the color image and its Resized images by different methods
4.	Contrast stretching of a low contrast image, Histogram, and Histogram Equalization
5.	Display of bit planes of an Image
6.	Display of FTT (1D, 2D) of an image
7.	Computation of mean, Standard Deviation, Correlation coefficient of the given Image
8.	Implementation of Image Smoothening Filters (Mean and Median filtering of an image)
9.	Implementation of image sharpening filters and Edge Detection using Gradient Filters

INDEX SHEET

SL.NO	Programs	Page no	Date of Execution	Staff Signature
1.	Implementation of Relationships between Pixels Neighbor of 4,8 and Diagonal points.	1	02/04/2024	
2	a. Simulation and Display of an Image, Negative of an image (Binary & Gray Scale) b. Display color Image, find its complement and convert to gray scale	2	02/04/2024	
3.	a. Implementation of Transformations of an image (Scaling & Rotation). b. Display the color image and its Resized images by different methods.	5	16/04/2024	
4.	Contrast stretching of a low contrast image, Histogram, and Histogram Equalization.	7	16/04/2024	
5.	Display of bit planes of an Image.	8	07/05/2024	
6.	Display of FTT (1D, 2D) of an image.	9	07/05/2024	
7.	Computation of mean, Standard Deviation, Correlation coefficient of the given Image.	10	07/05/2024	
8.	Implementation of Image Smoothening Filters (Mean and Median filtering of an Image).	12	21/05/2024	
9.	Implementation of image sharpening filters and Edge Detection using Gradient Filters.	16	21/05/2024	

Implementation of Relationships between Pixels Neighbor of 4,8 and Diagonal points.

PROGRAM 1

```
a = np.array([[17, 24, 1, 8, 15],
              [23, 5, 7, 14, 16],
              [ 4, 6, 13, 20, 22],
              [10, 12, 19, 21, 3],
              [11, 18, 25, 2, 9]])
print("a =")
print(a)

b = int(input("Enter the row < size of the Matrix: "))
c = int(input("Enter the column < size of matrix: "))
print("Element:", a[b, c])

N4 = [a[b+1, c] if b+1 < a.shape[0] else None,
      a[b-1, c] if b-1 >= 0 else None,
      a[b, c+1] if c+1 < a.shape[1] else None,
      a[b, c-1] if c-1 >= 0 else None]
print("N4 =")
print(N4)

N8 = [a[b+1, c] if b+1 < a.shape[0] else None,
      a[b-1, c] if b-1 >= 0 else None,
      a[b, c+1] if c+1 < a.shape[1] else None,
      a[b, c-1] if c-1 >= 0 else None, # Left
      a[b+1, c+1] if b+1 < a.shape[0] and c+1 < a.shape[1] else None,
      a[b+1, c-1] if b+1 < a.shape[0] and c-1 >= 0 else None,
      a[b-1, c-1] if b-1 >= 0 and c-1 >= 0 else None,
      a[b-1, c+1] if b-1 >= 0 and c+1 < a.shape[1] else None] #
print("N8 =")
print(N8)

ND = [a[b+1, c+1] if b+1 < a.shape[0] and c+1 < a.shape[1] else None,
      a[b+1, c-1] if b+1 < a.shape[0] and c-1 >= 0 else None,
      a[b-1, c-1] if b-1 >= 0 and c-1 >= 0 else None,
      a[b-1, c+1] if b-1 >= 0 and c+1 < a.shape[1] else None]
print("ND =")
print(ND)
```

```
a =
[[17 24  1  8 15]
 [23  5  7 14 16]
 [ 4  6 13 20 22]
 [10 12 19 21  3]
 [11 18 25  2  9]]
Element: 9
N4 =
[None, 3, None, 2]
```

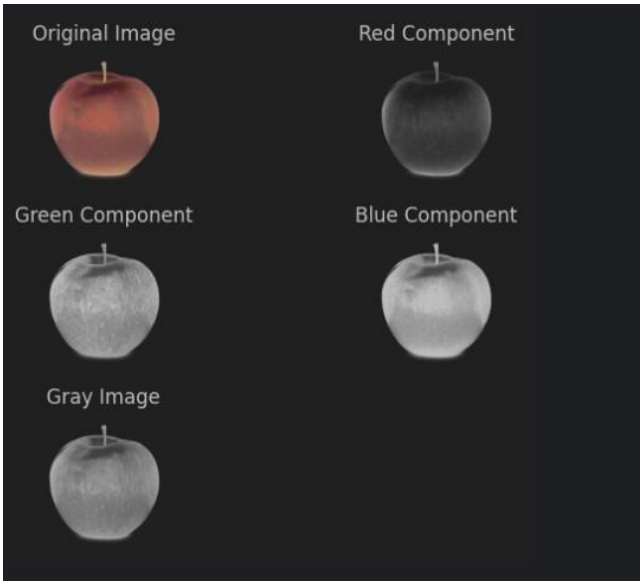
```
import numpy as np

# Create the 5x5 array
a = np.array([
    [17, 24, 1, 8, 15],
    [23, 5, 7, 14, 16],
    [4, 6, 13, 20, 22],
    [10, 12, 19, 21, 3],
    [11, 18, 25, 2, 9]
])

# Print the array
print("a =")
print(a)

# Get user input for row and column
x = int(input("Enter the row (0 to 4): "))
y = int(input("Enter the column (0 to 4): "))

# Ensure the input is within bounds
if 0 <= x < 5 and 0 <= y < 5:
    print("Element:", a[x, y])

    # Get 4-connected neighbors
    N4 = [
        a[x+1, y] if x+1 < 5 else None,  # Down
        a[x-1, y] if x-1 >= 0 else None,  # Up
        a[x, y+1] if y+1 < 5 else None,  # Right
        a[x, y-1] if y-1 >= 0 else None  # Left
    ]
    print("N4 =", N4)

    # Get 8-connected neighbors
    N8 = [
        a[x+1, y] if x+1 < 5 else None,  # Down
        a[x-1, y] if x-1 >= 0 else None,  # Up
        a[x, y+1] if y+1 < 5 else None,  # Right
        a[x, y-1] if y-1 >= 0 else None,  # Left
        a[x+1, y+1] if x+1 < 5 and y+1 < 5 else None,  # Bottom-right
        a[x+1, y-1] if x+1 < 5 and y-1 >= 0 else None,  # Bottom-left
        a[x-1, y-1] if x-1 >= 0 and y-1 >= 0 else None,  # Top-left
        a[x-1, y+1] if x-1 >= 0 and y+1 < 5 else None  # Top-right
    ]
    print("N8 =", N8)

    # Get diagonal neighbors
    ND = [
        a[x+1, y+1] if x+1 < 5 and y+1 < 5 else None,  # Bottom-right
        a[x+1, y-1] if x+1 < 5 and y-1 >= 0 else None,  # Bottom-left
        a[x-1, y-1] if x-1 >= 0 and y-1 >= 0 else None,  # Top-left
        a[x-1, y+1] if x-1 >= 0 and y+1 < 5 else None  # Top-right
    ]
    print("ND =", ND)
else:
    print("Invalid row or column. Please enter values within the range of the matrix.")
```

```
N8 =
[None, 3, None, 2, None, None, 21, None]
ND =
[None, None, 21, None]
```

PROGRAM 2A

```
import cv2
import matplotlib.pyplot as plt

i = cv2.imread('apple.jpg')

i = cv2.cvtColor(i, cv2.COLOR_BGR2RGB)

plt.subplot(3, 2, 1)
plt.imshow(i)
plt.title('Original Image')
plt.axis('off')

r = i[:, :, 0]
plt.subplot(3, 2, 2)
plt.imshow(r, cmap='gray')
plt.title('Red Component')
plt.axis('off')

g = i[:, :, 1]
plt.subplot(3, 2, 3)
plt.imshow(g, cmap='gray')
plt.title('Green Component')
plt.axis('off')

b = i[:, :, 2]
plt.subplot(3, 2, 4)
plt.imshow(b, cmap='gray')
plt.title('Blue Component')
plt.axis('off')

rg = cv2.cvtColor(i, cv2.COLOR_RGB2GRAY)
plt.subplot(3, 2, 5)
plt.imshow(rg, cmap='gray')
plt.title('Gray Image')
plt.axis('off')
plt.tight_layout()
plt.show()
```

```
import cv2 as cv
import matplotlib.pyplot as plt
i = cv.imread("apple.jpg")

plt.subplot(1,1,1)## row , colum , plot num
i = cv.cvtColor(i,cv.COLOR_BGR2RGB)
plt.imshow(i)
plt.title("Original Image")

r = image[:,:,0]
plt.imshow(r,cmap = "gray")
plt.title("Red Components")

g = image[:,:,1]
plt.imshow(g,cmap = "gray")
plt.title("Green Components")

b = image[:,:,2]
plt.imshow(b,cmap = "gray")
plt.title("Blue Components")

gray_image = cv.cvtColor(image,cv.COLOR_RGB2GRAY)
plt.imshow(gray_image,cmap = "gray")
plt.title("Gray Image")
```



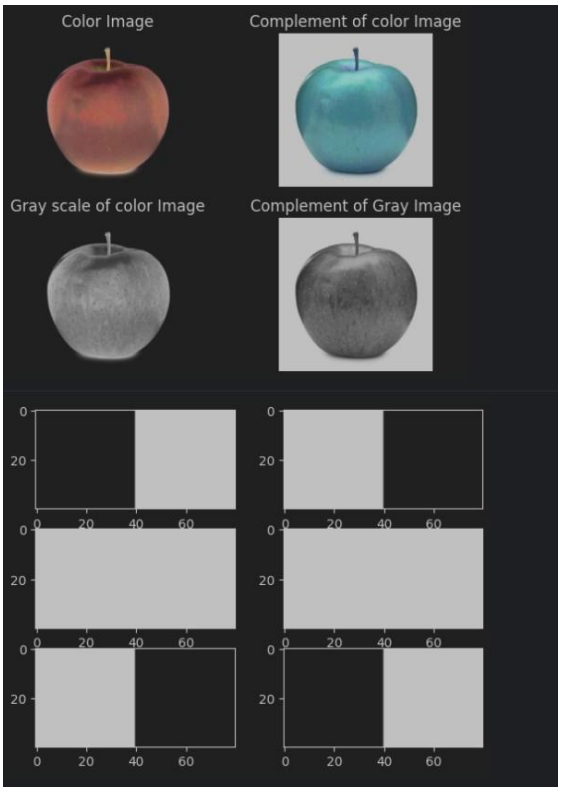
Display color Image, find its complement and convert to gray scale

PROGRAM 2B

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Read the color image
I = cv2.imread('apple.jpg')
# Convert BGR to RGB (OpenCV reads images in BGR format)
I_rgb = cv2.cvtColor(I, cv2.COLOR_BGR2RGB)
# Plot original color image
plt.subplot(2, 2, 1)
plt.imshow(I_rgb)
plt.title('Color Image')
plt.axis('off')
# Find complement of color image
c = cv2.bitwise_not(I_rgb)
plt.subplot(2, 2, 2)
plt.imshow(c)
plt.title('Complement of color Image')
plt.axis('off')
# Convert color image to grayscale
r = cv2.cvtColor(I_rgb, cv2.COLOR_RGB2GRAY)
plt.subplot(2, 2, 3)
plt.imshow(r, cmap='gray')
plt.title('Gray scale of color Image')
plt.axis('off')
# Find complement of grayscale image
b = cv2.bitwise_not(r)
plt.subplot(2, 2, 4)
plt.imshow(b, cmap='gray')
plt.title('Complement of Gray Image')
```

```
import matplotlib.pyplot as plt
import numpy as np
import cv2 as cv

def plot_img(img,title,cmap = None):
    plt.figure()
    plt.imshow(img,cmap = cmap)
    plt.title(title)
    plt.show()

image = cv.imread("apple.jpg")
image = cv.cvtColor(image,cv.COLOR_BGR2RGB)
plot_img(image,"Color Image")

c = cv.bitwise_not(image)
plot_img(c,"Complement of Image")

gray = cv.cvtColor(image,cv.COLOR_RGB2GRAY)
plot_img(gray,"gray scale",cmap = "gray"

gray_complement = cv.bitwise_not(gray)
plot_img(gray_complement,"Gray Complement image", cmap = "gray")

# Simulation of an Image (Arithmetic & Logic Operation)
a = np.ones((40, 40), dtype=np.uint8)
b = np.zeros((40, 40), dtype=np.uint8)
c = np.hstack((a, b))
d = np.hstack((b, a))
e = np.vstack((c, d))
A = 10 * (c + d)
M = c * d
S = np.abs(c - d)
D = c / 4

plot_img(c,"image",cmap = "gray")

plot_img(d,"image",cmap = "gray")

plot_img(A,"image",cmap = "gray")

plot_img(M,"image",cmap = "gray")

plot_img(S,"image",cmap = "gray")

plot_img(D,"image",cmap = "gray")
```

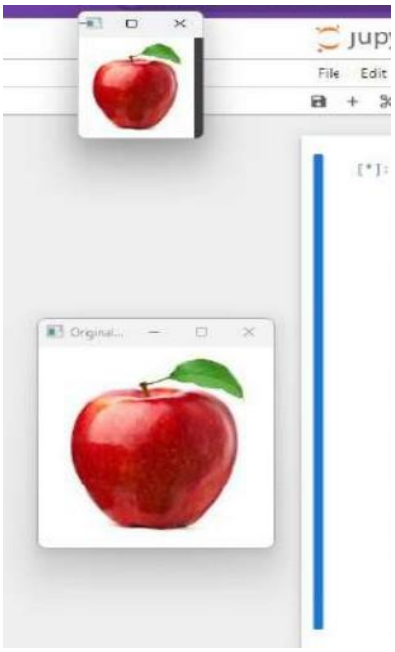
```
plt.axis('off')
# Simulation of an Image (Arithmetic & Logic Operation)
a = np.ones((40, 40), dtype=np.uint8)
b = np.zeros((40, 40), dtype=np.uint8)
c = np.hstack((a, b))
d = np.hstack((b, a))
e = np.vstack((c, d))
A = 10 * (c + d)
M = c * d
S = np.abs(c - d)
D = c / 4
plt.figure()
plt.subplot(3, 2, 1)
plt.imshow(c, cmap='gray')
plt.subplot(3, 2, 2)
plt.imshow(d, cmap='gray')
plt.subplot(3, 2, 3)
plt.imshow(A, cmap='gray')
plt.subplot(3, 2, 4)
plt.imshow(M, cmap='gray')
plt.subplot(3, 2, 5)
plt.imshow(S, cmap='gray')
plt.subplot(3, 2, 6)
plt.imshow(D, cmap='gray')
plt.show()
```



PROGRAM 3A

```
import cv2
import numpy as np

# Load the image
image = cv2.imread('apple.jpg')

# Define scaling factor
scaling_factor = 0.5  # You can change this value

# Perform scaling
scaled_image = cv2.resize(image, None, fx=scaling_factor, fy=scaling_factor,
interpolation=cv2.INTER_LINEAR)

# Define rotation angle (in degrees)
rotation_angle = 45  # You can change this value

# Perform rotation
height, width = scaled_image.shape[:2]
rotation_matrix = cv2.getRotationMatrix2D((width/2, height/2), rotation_angle, 1)
rotated_image = cv2.warpAffine(scaled_image, rotation_matrix, (width, height))

# Display the original, scaled, and rotated images
cv2.imshow('Original Image', image)
cv2.imshow('Scaled Image', scaled_image)
cv2.imshow('Rotated Image', rotated_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
import matplotlib.pyplot as plt
import numpy as np
import cv2 as cv

image = cv.imread("apple.jpg")
image = cv.cvtColor(image,cv.COLOR_BGR2RGB)
plt.imshow(image)
plt.title("Original Image")

scaled_img = cv.resize(image,None,fx = 0.5,fy = 0.5,interpolation = cv.INTER_LINEAR)
plt.imshow(scaled_img)

from scipy import ndimage
rotated_image = ndimage.rotate(image,45)
plt.imshow(rotated_image)
```

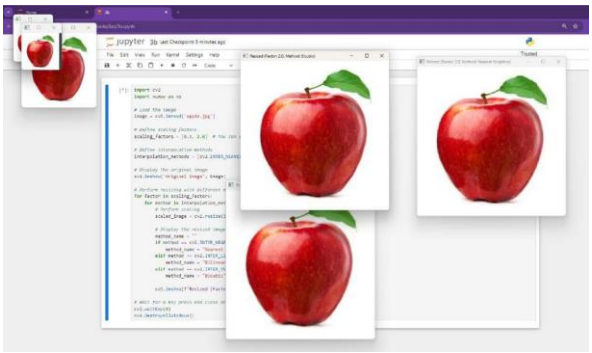


Display the color image and its Resized images by different methods.

PROGRAM 3B

```
import cv2
import numpy as np

# Load the image
image = cv2.imread('apple.jpg')

# Define scaling factors
scaling_factors = [0.5, 2.0]  # You can add more scaling factors as needed

# Define interpolation methods
interpolation_methods = [cv2.INTER_NEAREST, cv2.INTER_LINEAR, cv2.INTER_CUBIC]

# Display the original image
cv2.imshow('Original Image', image)

# Perform resizing with different methods
for factor in scaling_factors:
    for method in interpolation_methods:
        # Perform scaling
        scaled_image = cv2.resize(image, None, fx=factor, fy=factor, interpolation=method)

        # Display the resized image
        method_name = ""
        if method == cv2.INTER_NEAREST:
            method_name = "Nearest Neighbor"
        elif method == cv2.INTER_LINEAR:
            method_name = "Bilinear"
        elif method == cv2.INTER_CUBIC:
            method_name = "Bicubic"

        cv2.imshow(f'Resized (Factor: {factor}, Method: {method_name})', scaled_image)

# Wait for a key press and close all windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

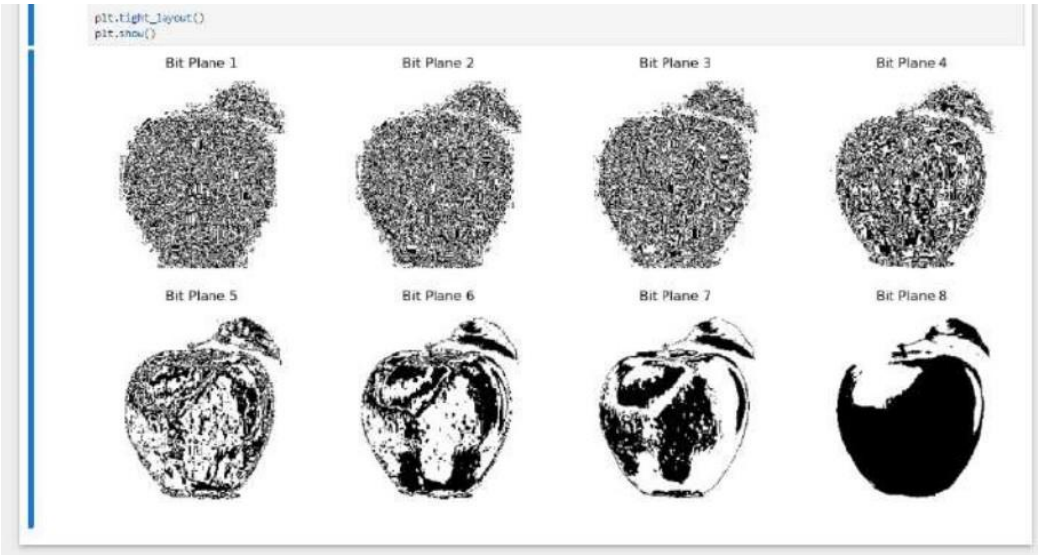
```
import matplotlib.pyplot as plt
import numpy as np
import cv2 as cv

scaling_factor = [0.5,1.5,2.0]
interpolation = [cv.INTER_NEAREST,cv.INTER_LINEAR,cv.INTER_CUBIC]

image = cv.imread("apple.jpg")
image = cv.cvtColor(image,cv.COLOR_BGR2RGB)
plt.imshow(image)

for i in scaling_factor:
    for j in interpolation:
        scaled_img = cv.resize(image,None,fx = i,fy=i,interpolation=j)
        method_name = ""

        if j == cv.INTER_NEAREST:
            method_name = "Nearest Neighbor"

        elif j== "cv.INTER_LINEAR":
            method_name = "Bilinear"

        else:
            method_name = "Bicubic"

        cv.imshow(f"Resized(factor:{i}, Method = {method_name}",scaled_img)

cv.waitKey(0)
cv.destroyAllWindows()
```



PROGRAM 4

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image
image = cv2.imread('input_image.jpg', cv2.IMREAD_GRAYSCALE)

# Apply contrast stretching
min_intensity = np.min(image)
max_intensity = np.max(image)
stretched_image = cv2.normalize(image, None, 0, 255, norm_type=cv2.NORM_MINMAX)

# Calculate and plot histograms
hist_original = cv2.calcHist([image], [0], None, [256], [0, 256])
hist_stretched = cv2.calcHist([stretched_image], [0], None, [256], [0, 256])

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(hist_original, color='b')
plt.title('Original Image Histogram')

plt.subplot(1, 2, 2)
plt.plot(hist_stretched, color='r')
plt.title('Stretched Image Histogram')

plt.tight_layout()
plt.show()

# Apply histogram equalization
equalized_image = cv2.equalizeHist(image)

# Calculate and plot histograms for equalized image
hist_equalized = cv2.calcHist([equalized_image], [0], None, [256], [0, 256])

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(stretched_image, cmap='gray')
plt.title('Contrast Stretched Image')

plt.subplot(1, 2, 2)
plt.imshow(equalized_image, cmap='gray')
plt.title('Histogram Equalized Image')

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

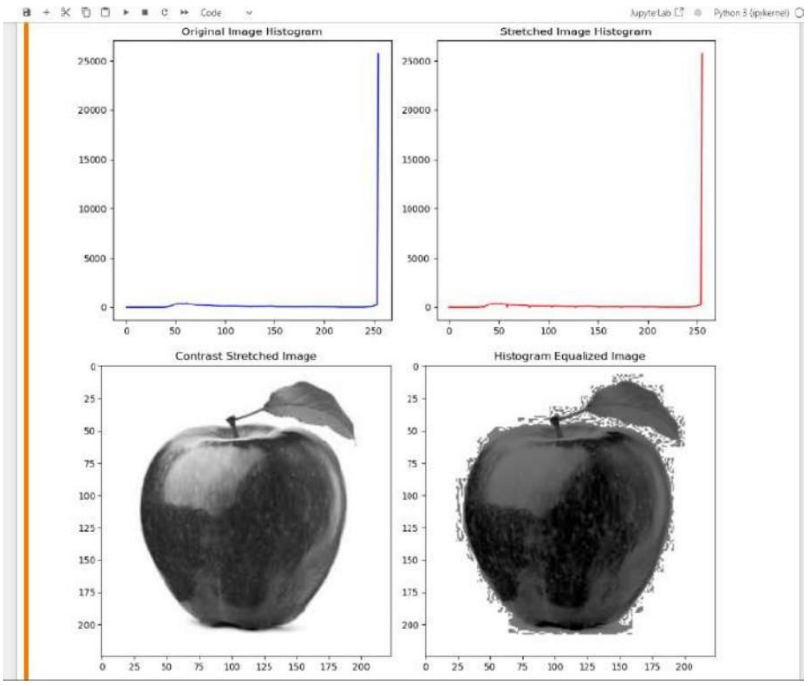
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Helper function to plot images
def plot_image(image, title, cmap='gray'):
    plt.figure()
    plt.imshow(image, cmap=cmap)
    plt.title(title)
    plt.axis('off')
    plt.show()

image = cv2.imread('apple.jpg', cv2.IMREAD_GRAYSCALE)
min_intensity = np.min(image)
max_intensity = np.max(image)
stretched_image = cv2.normalize(image, None, 0, 255,
norm_type=cv2.NORM_MINMAX)

hist_original = cv2.calcHist([image], [0], None, [256], [0, 256])
hist_stretched = cv2.calcHist([stretched_image], [0], None, [256], [0, 256])

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(hist_original, color='b')
plt.title('Original Image Histogram')
plt.subplot(1, 2, 2)
plt.plot(hist_stretched, color='r')
plt.title('Stretched Image Histogram')
plt.tight_layout()
plt.show()

# Apply histogram equalization
equalized_image = cv2.equalizeHist(image)

# Calculate and plot histograms for equalized image
hist_equalized = cv2.calcHist([equalized_image], [0], None, [256], [0, 256])

# Plot images using the plot_image function
plot_image(stretched_image, 'Contrast Stretched Image')
plot_image(equalized_image, 'Histogram Equalized Image')
```



PROGRAM 5

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image in grayscale
image = cv2.imread('input_image.jpg', cv2.IMREAD_GRAYSCALE)

# Get the dimensions of the image
height, width = image.shape

# Create an array to store the bit planes
bit_planes = np.zeros((8, height, width), dtype=np.uint8)

# Calculate the bit planes
for i in range(8):
    bit_planes[i] = (image >> i) & 1  # Extract ith bit plane

# Display the bit planes
plt.figure(figsize=(12, 6))
for i in range(8):
    plt.subplot(2, 4, i+1)
    plt.imshow(bit_planes[i], cmap='gray')
    plt.title(f'Bit Plane {i+1}')
    plt.axis('off')

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

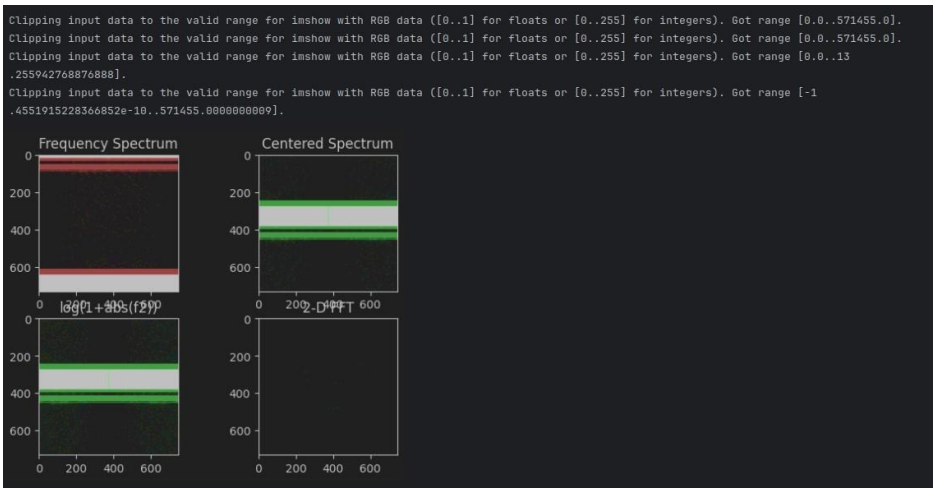
```
image = cv.imread("apple.jpg")
image = cv.cvtColor(image , cv.COLOR_BGR2GRAY)
plt.imshow(image,cmap = "gray")

image.shape

bit_plane = np.zeros((8,height,width),dtype = np.uint8)

for i in range(8):
    bit_plane[i] = (image >> i ) &1
for i in range(8):
    plt.subplot(2,4,i+1)
    plt.imshow(bit_plane[i],cmap = "gray")

plt.show()
```



PROGRAM 6

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft2, fftshift

# Read the image and convert to double precision array
l = plt.imread('cancercell.jpg').astype(float)

# Perform 2-D FFT
f1 = np.fft.fft2(l)

# Shift zero frequency component to the center
f2 = np.fft.fftshift(f1)

# Display magnitude of frequency spectrum
plt.subplot(2, 2, 1)
plt.imshow(np.abs(f1))
plt.title('Frequency Spectrum')

# Display magnitude of centered spectrum
plt.subplot(2, 2, 2)
plt.imshow(np.abs(f2))
plt.title('Centered Spectrum')

# Compute log(1 + abs(f2))
f3 = np.log(1 + np.abs(f2))
```

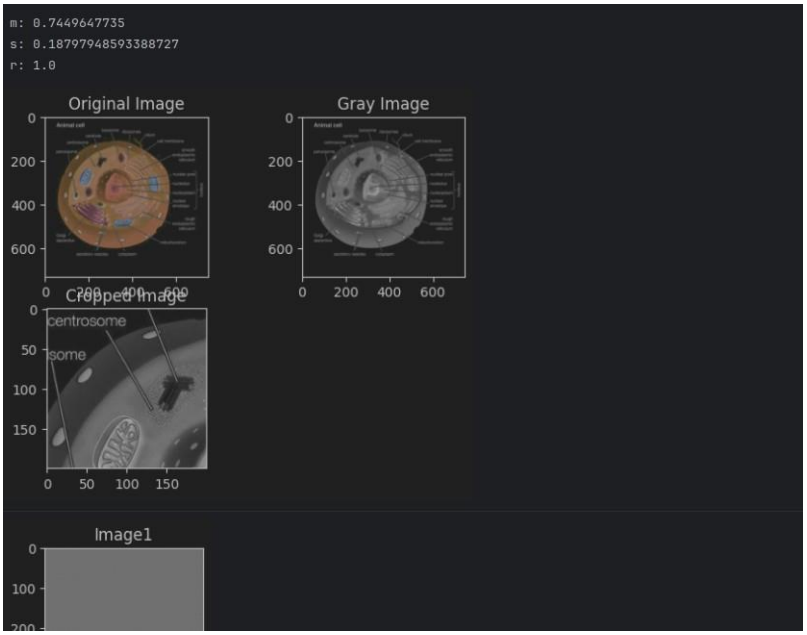
```
# Display log(1 + abs(f2))
plt.subplot(2, 2, 3)
plt.imshow(f3)
plt.title('log(1+abs(f2))')

# Perform 2-D FFT on f1
l_fft = fft2(f1)

# Take real part of the result
l1 = np.real(l_fft)

# Display real part of 2-D FFT
plt.subplot(2, 2, 4)
plt.imshow(l1)
plt.title('2-D FFT')

plt.show()
```



PROGRAM 7

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import io, color
from scipy.stats import pearsonr

# Read the image
i = io.imread('cancercell.jpg')

# Display original image
plt.subplot(2, 2, 1)
plt.imshow(i)
plt.title('Original Image')
```

```
# Convert to grayscale
g = color.rgb2gray(i)

# Display grayscale image
plt.subplot(2, 2, 2)
plt.imshow(g, cmap='gray')
plt.title('Gray Image')

# Crop the image
c = g[100:300, 100:300]

# Display cropped image
plt.subplot(2, 2, 3)
plt.imshow(c, cmap='gray')
plt.title('Cropped Image')

# Calculate mean and standard deviation of the cropped image
m = np.mean(c)
s = np.std(c)
print('m:', m)
print('s:', s)

# Generate checkerboard patterns
checkerboard = np.indices((400, 400)).sum(axis=0) % 2

# Create checkerboard images with different thresholds
k = checkerboard > 0.8
k1 = checkerboard > 0.5

# Display checkerboard images
plt.figure()
plt.subplot(2, 1, 1)
plt.imshow(k, cmap='gray')
plt.title('Image1')

plt.subplot(2, 1, 2)
plt.imshow(k1, cmap='gray')
plt.title('Image2')

# Calculate Pearson correlation coefficient between the two images
r, _ = pearsonr(k.flatten(), k1.flatten())
print('r:', r)

plt.show()
```

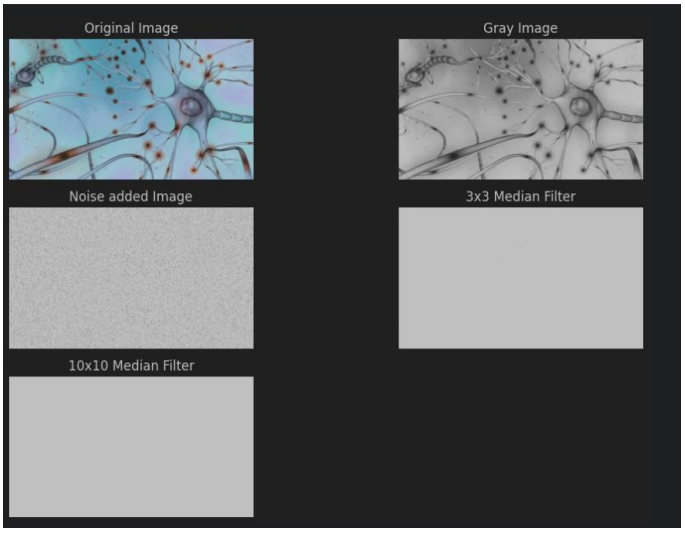


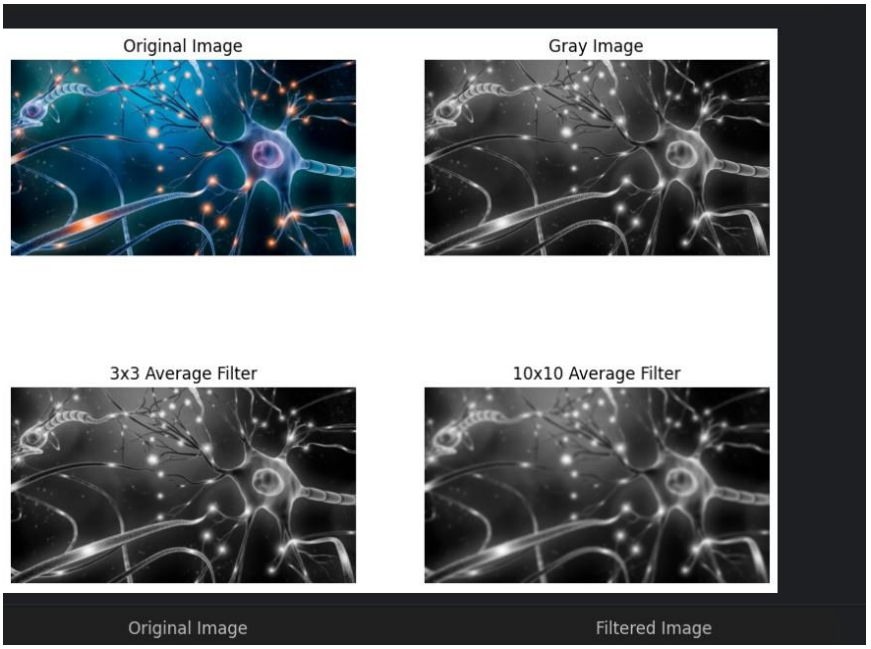
PROGRAM 8

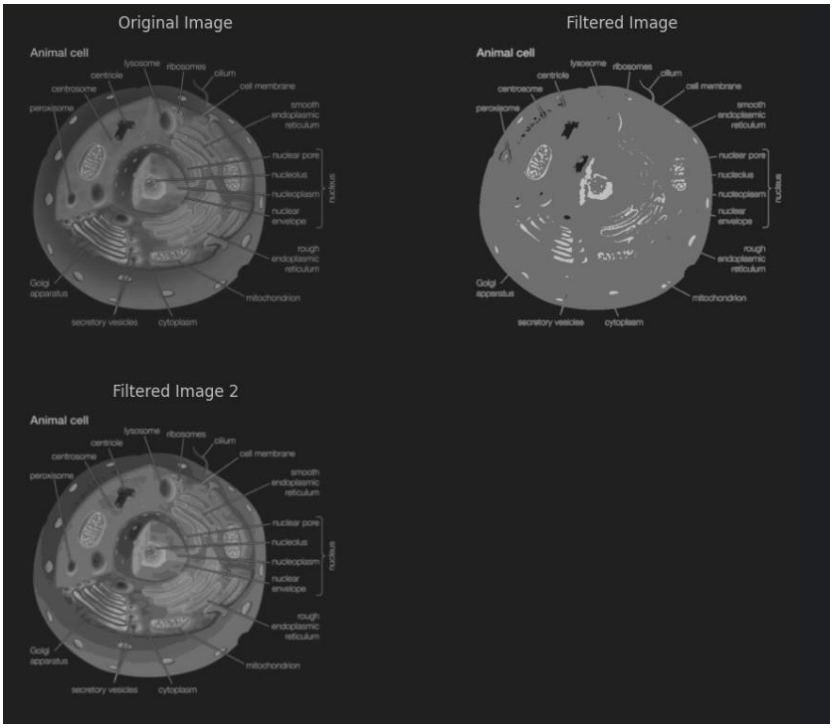
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import convolve
from scipy.ndimage import median_filter

# Read the image
I = cv2.imread('C:/Users/anuam/OneDrive/Desktop/neuron.jpg')
K = cv2.cvtColor(I, cv2.COLOR_BGR2GRAY)

# Add salt and pepper noise
J = cv2.randu(K.copy(), 0, 255)
noise = np.random.choice([0, 255], K.shape, p=[0.95, 0.05])
J[noise == 255] = 255
J[noise == 0] = 0

# Apply median filters
f = median_filter(J, size=(3, 3))
f1 = median_filter(J, size=(10, 10))

# Display results
plt.figure(figsize=(12, 8))
plt.subplot(3, 2, 1)
plt.imshow(cv2.cvtColor(I, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')

plt.subplot(3, 2, 2)
```

```
plt.imshow(K, cmap='gray')
plt.title('Gray Image')
plt.axis('off')

plt.subplot(3, 2, 3)
plt.imshow(J, cmap='gray')
plt.title('Noise added Image')
plt.axis('off')

plt.subplot(3, 2, 4)
plt.imshow(f, cmap='gray')
plt.title('3x3 Median Filter')
plt.axis('off')

plt.subplot(3, 2, 5)
plt.imshow(f1, cmap='gray')
plt.title('10x10 Median Filter')
plt.axis('off')

# Mean Filter and Average Filter
plt.figure(figsize=(10, 8))

i = cv2.imread('C:/Users/anuam/OneDrive/Desktop/neuron.jpg')
g = cv2.cvtColor(i, cv2.COLOR_BGR2GRAY)

# 3x3 Average filter
g1 = np.ones((3, 3)) / 9.0
b1 = convolve(g, g1)

plt.subplot(2, 2, 1)
plt.imshow(cv2.cvtColor(i, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')

plt.subplot(2, 2, 2)
plt.imshow(g, cmap='gray')
plt.title('Gray Image')
plt.axis('off')

plt.subplot(2, 2, 3)
plt.imshow(b1, cmap='gray')
plt.title('3x3 Average Filter')
plt.axis('off')

# 10x10 Average filter
g2 = np.ones((10, 10)) / 100.0
b2 = convolve(g, g2)
```

```
plt.subplot(2, 2, 4)
plt.imshow(b2, cmap='gray')
plt.title('10x10 Average Filter')
plt.axis('off')

# Implementation of filter using Convolution
plt.figure(figsize=(10, 8))

I = cv2.imread('C:/Users/anuam/OneDrive/Desktop/earcell.jpg', cv2.IMREAD_GRAYSCALE)
plt.subplot(2, 2, 1)
plt.imshow(I, cmap='gray')
plt.title('Original Image')
plt.axis('off')

# Convolution with filter a
a = np.array([[0.001, 0.001, 0.001], [0.001, 0.001, 0.001], [0.001, 0.001, 0.001]])
R = convolve(I, a)

plt.subplot(2, 2, 2)
plt.imshow(R, cmap='gray')
plt.title('Filtered Image')
plt.axis('off')

# Convolution with filter b
b = np.array([[0.005, 0.005, 0.005], [0.005, 0.005, 0.005], [0.005, 0.005, 0.005]])
R1 = convolve(I, b)

plt.subplot(2, 2, 3)
plt.imshow(R1, cmap='gray')
plt.title('Filtered Image 2')
plt.axis('off')
plt.tight_layout()
plt.show()
```




Original Image
Gray Image
3x3 Average Filter
10x10 Average Filter
Original Image
Filtered Image


Original Image
Filtered Image
Filtered Image 2
Animal cell
lysosome
ribosomes
cilium
centriole
centrosome
cell membrane
peroxisome
smooth endoplasmic reticulum
nuclear pore
nucleolus
nucleoplasm
nuclear envelope
nucleus
rough endoplasmic reticulum
Golgi apparatus
mitochondrion
secretory vesicles
cytoplasm

PROGRAM 9

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import convolve
from scipy import ndimage
import os

# Define function to read image safely
def safe_imread(filename):
    if not os.path.exists(filename):
        raise FileNotFoundError(f"File '{filename}' not found.")
    return cv2.imread(filename)

# Define the Laplacian filter
def laplacian_filter(img, alpha=0.05):
    kernel = np.array([[0, 1, 0], [1, -4 + alpha, 1], [0, 1, 0]])
    return convolve(img, kernel)

# Main script
try:
    # Read the image
    i = safe_imread('C:/Users/anuam/OneDrive/Desktop/neuron.jpg')

    # Display the original image
    plt.subplot(4, 2, 1)
    plt.imshow(cv2.cvtColor(i, cv2.COLOR_BGR2RGB))
    plt.title('Original Image')
    plt.axis('off')

    # Convert to grayscale
    g = cv2.cvtColor(i, cv2.COLOR_BGR2GRAY)

    # Display the grayscale image
    plt.subplot(4, 2, 2)
    plt.imshow(g, cmap='gray')
    plt.title('Gray Image')
    plt.axis('off')

    # Apply Laplacian filter
    f = laplacian_filter(g, alpha=0.05)

    # Display the Laplacian filtered image
    plt.subplot(4, 2, 3)
    plt.imshow(f, cmap='gray')
    plt.title('Laplacian')
    plt.axis('off')
```

```
# Apply Sobel edge detection
s = cv2.Sobel(g, cv2.CV_64F, 1, 0, ksize=3) + cv2.Sobel(g, cv2.CV_64F, 0, 1, ksize=3)

# Display the Sobel edge detected image
plt.subplot(4, 2, 4)
plt.imshow(s, cmap='gray')
plt.title('Sobel')
plt.axis('off')

# Apply Prewitt edge detection
kernelx = np.array([[1, 0, -1], [1, 0, -1], [1, 0, -1]])
kernely = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]])
px = convolve(g, kernelx)
py = convolve(g, kernely)
p = np.sqrt(px*2 + py*2)

# Display the Prewitt edge detected image
plt.subplot(4, 2, 5)
plt.imshow(p, cmap='gray')
plt.title('Prewitt')
plt.axis('off')

# Apply Roberts edge detection
kernelx = np.array([[1, 0], [0, -1]])
kernely = np.array([[0, 1], [-1, 0]])
rx = convolve(g, kernelx)
ry = convolve(g, kernely)
r = np.sqrt(rx*2 + ry*2)

# Display the Roberts edge detected image
plt.subplot(4, 2, 6)
plt.imshow(r, cmap='gray')
plt.title('Roberts')
plt.axis('off')

# Apply Sobel edge detection (horizontal)
sobel_horizontal = cv2.Sobel(g, cv2.CV_64F, 1, 0, ksize=3)

# Display the Sobel horizontal edge detected image
plt.subplot(4, 2, 7)
plt.imshow(sobel_horizontal, cmap='gray')
plt.title('Sobel Horizontal')
plt.axis('off')

# Apply Sobel edge detection (vertical)
sobel_vertical = cv2.Sobel(g, cv2.CV_64F, 0, 1, ksize=3)
```

```
    # Display the Sobel vertical edge detected image
    plt.subplot(4, 2, 8)
    plt.imshow(sobel_vertical, cmap='gray')
    plt.title('Sobel Vertical')
    plt.axis('off')

    plt.tight_layout()
    plt.show()

except FileNotFoundError as e:
    print(e)
```