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

21AD61 : DIGITAL IMAGE PROCESSING

NAME : _____

USN : _____

YEAR/SEM : _____

SECTION : _____

BRANCH : _____

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

CERTIFICATE

This is to certify that Mr./Mrs. _____studying in ____semester _____Branch has satisfactorily completed the course of experiments in _____Laboratory for the academic session 2023-2024 as prescribed by the Visvesvaraya Technological University.

Register Number of the Candidate:

Signature of the Staff Incharge

Date:_____

Signature of HOD

Date:

Sl. No	**Program / Exercise**
1.	**Implementation of Relationships between Pixels Neighbour of 4,8 and Diagonal point.**
2.	**(a).Display color Image, find its complement and convert to gray scale.** **(b).Simulation and Display of an Image, Negative of an Image(Binary & Gray Scale.**
3.	(a).**Implementation of Transformations of an Image i)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 plane 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.	**mplementation of Relationships between Pixels Neighbour of 4,8 and Diagonal point.**		02/11/2023	
2	**(a).Display color Image, find its complement and convert to gray scale.** **(b).Simulation and Display of an Image, Negative of an Image(Binary & Gray Scale.**		07/11/2023	
3.	(a).**Implementation of Transformations of an Image i)Scaling & Rotation.** **(b).Display the color image and its Resized images by different methods.**		21/11/2023	
4.	**Contrast stretching of a low contrast image, Histogram, and Histogram Equalization.**		28/11/2023	
5.	**Display of bit plane of an image.**		04/12/2023	
6.	**Display of FTT(1D, 2D) of an image.**		12/12/2023	
7.	**Computation of mean, Standard Deviation, Correlation coefficient of the given Image.**		19/12/2023	
8.	**Implementation of Image Smoothening Filters(Mean and Median filtering of an Image.**		26/12/2023	
9.	**Implementation of image sharpening filters and Edge Detection using Gradient Filters.**		16/01/2024	

DIGITAL IMAGE PROCESSING AND COMPUTER VISION

LAB PROGRAMS

PROGRAM 1

Implementation of Relationships between Pixels Neighbour of 4,8 and Diagonal point

CODE :

```
import numpy as np

# Create a sample matrix (using a magic square for demonstration)
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)

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

# 4-Point Neighbours
N4 = [a[b+1, c] if b+1 < a.shape[0] else None, # Below
      a[b-1, c] if b-1 >= 0 else None, # Above
      a[b, c+1] if c+1 < a.shape[1] else None, # Right
      a[b, c-1] if c-1 >= 0 else None] # Left

print("N4 =")
print(N4)

# 8-Point Neighbours
N8 = [a[b+1, c] if b+1 < a.shape[0] else None, # Below
      a[b-1, c] if b-1 >= 0 else None, # Above
      a[b, c+1] if c+1 < a.shape[1] else None, # Right
      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, # Below-Right
      a[b+1, c-1] if b+1 < a.shape[0] and c-1 >= 0 else None, # Below-Left
      a[b-1, c-1] if b-1 >= 0 and c-1 >= 0 else None, # Above-Left
      a[b-1, c+1] if b-1 >= 0 and c+1 < a.shape[1] else None] # Above-Right

print("N8 =")
print(N8)
```

```
# Diagonal Neighbours
ND = [a[b+1, c+1] if b+1 < a.shape[0] and c+1 < a.shape[1] else None, # Below-Right
      a[b+1, c-1] if b+1 < a.shape[0] and c-1 >= 0 else None, # Below-Left
      a[b-1, c-1] if b-1 >= 0 and c-1 >= 0 else None, # Above-Left
      a[b-1, c+1] if b-1 >= 0 and c+1 < a.shape[1] else None] # Above-Right

print("ND =")
print(ND)
```

OUTPUT :

```
print(N4)
```

```
[21, 14, 22, 13]
```

```
print(N8)
```

```
[21, 14, 22, 13, 3, 19, 7, 16]
```

```
print(ND)
```

```
[3, 19, 7, 16]
```

PROGRAM 2 (a)

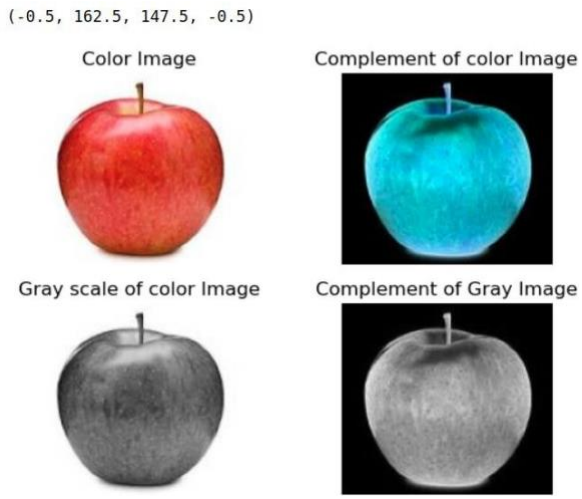
Display color Image, find its complement and convert to gray scale

CODE :

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

# Read the color image
I = cv2.imread('C:/Users/anuam/Downloads/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')
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()
```

OUTPUT :



PROGRAM 2(b)

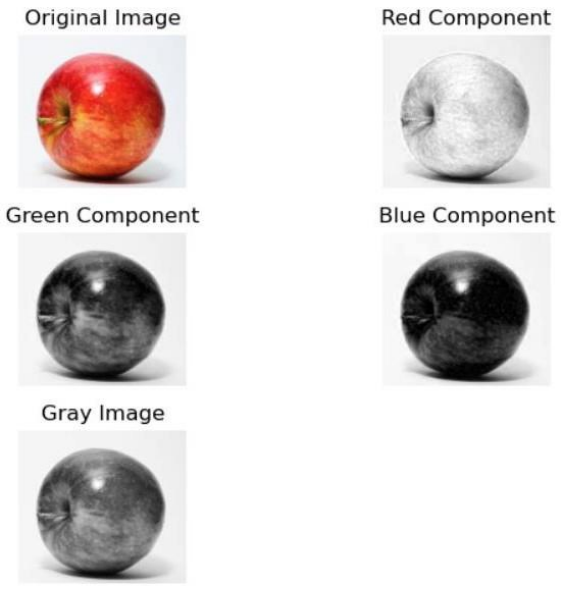
Simulation and Display of an Image,
Negative of an Image(Binary & Gray Scale

CODE :

```
import cv2
import matplotlib.pyplot as plt

# Read the image
i = cv2.imread('C:/Users/anuam/Downloads/apple.jpg')

# Convert BGR to RGB (OpenCV reads images in BGR format)
i = cv2.cvtColor(i, cv2.COLOR_BGR2RGB)

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

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

# Displaying in grayscale
plt.title('Red Component')
plt.axis('off')

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

# Displaying in grayscale
```

```
plt.title('Green Component')
plt.axis('off')

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

# Displaying in grayscale
plt.title('Blue Component')
plt.axis('off')

# Convert to grayscale
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()
```

OUTPUT :



PROGRAM 3 (a)

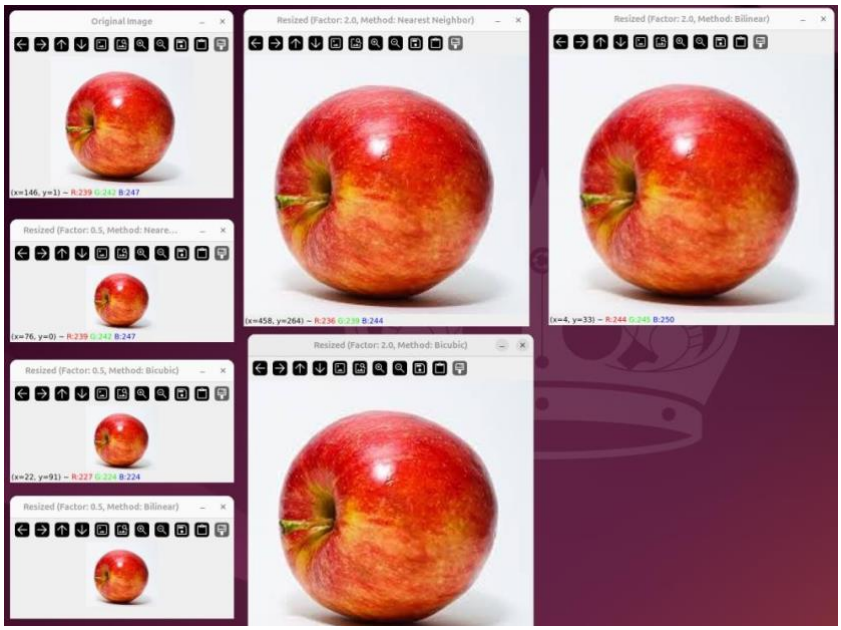
Implementation of Transformations of an Image i)Scaling & Rotation

CODE :

```
import cv2
import numpy as np

# Load the image
image = cv2.imread('C:/Users/anuam/Downloads/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()
```

OUTPUT :



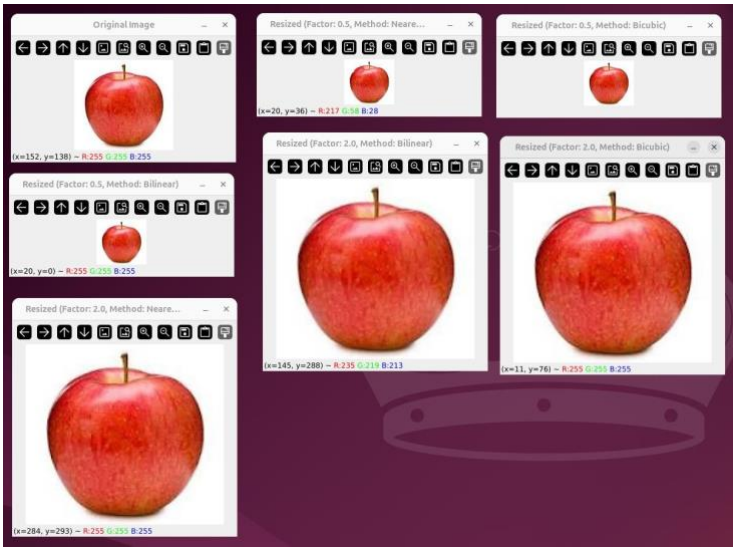
PROGRAM 3 (b)

Display the color image and its Resized images by different methods

CODE :

```
import cv2
import numpy as np

# Load the image
image = cv2.imread('input_image.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()
```

OUTPUT :



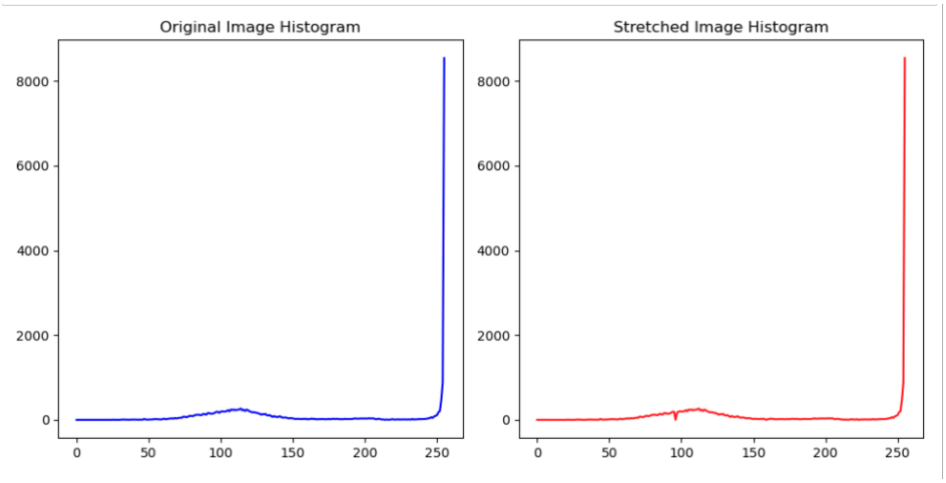
PROGRAM 4

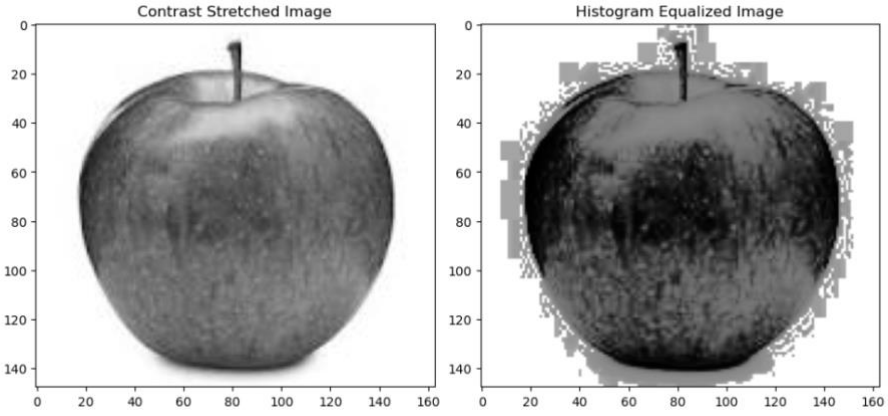
Contrast stretching of a low contrast image, Histogram, and Histogram Equalization

CODE :

```
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()
```

OUTPUT :


Original Image Histogram
Stretched Image Histogram
8000
6000
4000
2000
0
0
50
100
150
200
250


Contrast Stretched Image
Histogram Equalized Image
0
20
40
60
80
100
120
140
160

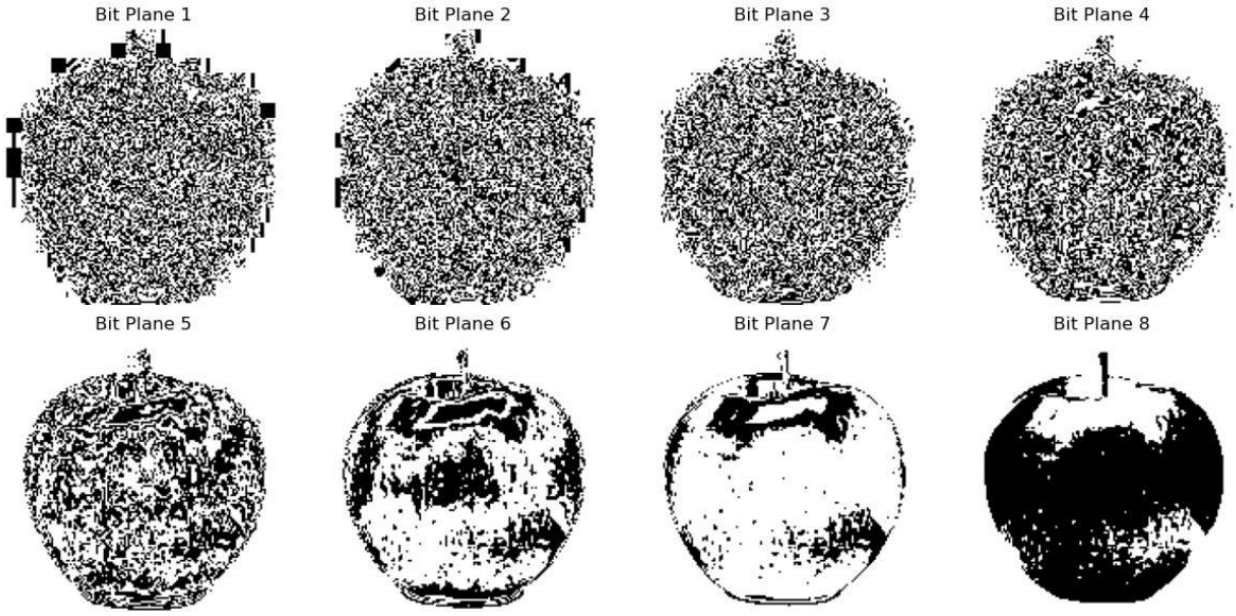
PROGRAM 5

Display of bit plane of an image.

CODE :

```
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()
```

OUTPUT :



PROGRAM 6

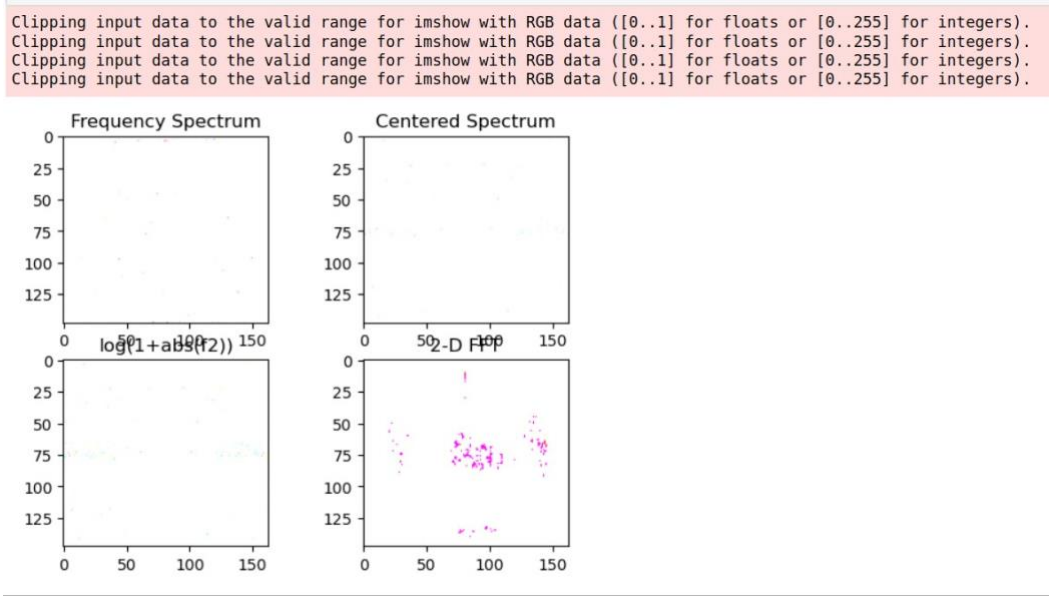
Display of FTT(1D, 2D) of an image

CODE :

```
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()
```

OUTPUT :



PROGRAM 7

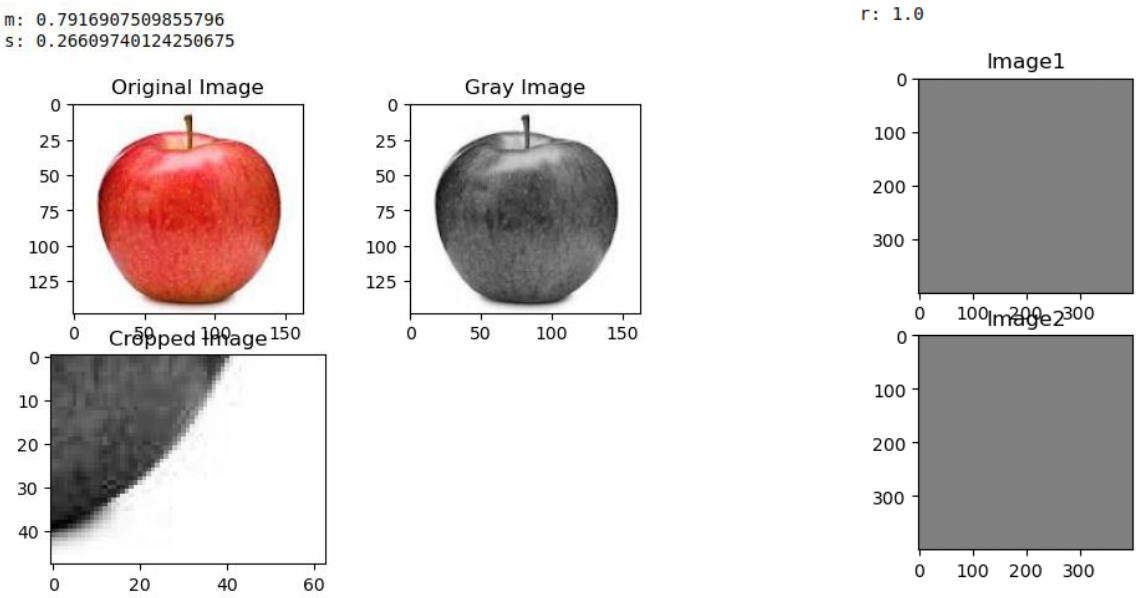
Computation of mean, Standard Deviation, Correlation coefficient of the given Image

CODE :

```
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()
```

OUTPUT :



PROGRAM 8

Implementation of Image Smoothening Filters(Mean and Median filtering of an Image

CODE :

```
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()
```

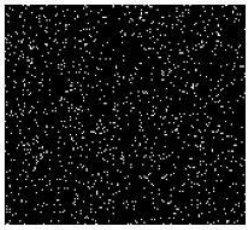
OUTPUT :

(-0.5, 162.5, 147.5, -0.5)

Original Image

Gray Image

Noise added Image



3x3 Median Filter

10x10 Median Filter

(-0.5, 162.5, 147.5, -0.5)

Original Image

Gray Image

3x3 Average Filter

10x10 Average Filter

(-0.5, 162.5, 147.5, -0.5)

Original Image

Filtered Image 2

PROGRAM 9

Implementation of image sharpening filters and Edge Detection using Gradient Filters.

CODE :

```
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)
```

OUTPUT :