# Digital Image Processing

## Journal

Submitted By

**Nagda Mihir Anupkumar**

Roll No: 31031523016

MSc CS – Part 1

**Department Of Computer Science**

**Somaiya Vidyavihar University**

**SK Somaiya college**

Index:

| **Sr No** | **Title** |
|---|---|
| **1.** | Perform the following:<br>1. Showing Histogram of an Image<br>2. Log Transformation<br>3. Power Log Transformation<br>4. Contrast Stretching<br>5. Thresholding Operations |
| **2.** | Implement Simple and Adaptive Histogram Equalization |
| **3.** | Perform the following:<br>1. Derivatives and Gradients<br>2. Laplacian Filter<br>3. Sharpening with Laplacian Plot<br>4. Unsharp Masking<br>5. Image Negatives |
| **4.** | Implement Sobel Image Detector & Canny Edge Detector using Scikit-Image |
| **5.** | Perform the following:<br>1. Erosion<br>2. Dilation<br>3. Opening and Closing<br>4. Skeletonization<br>5. Covex Hull<br>6. White and Black Top-Hats<br>7. Boundary Extraction |
| **6.** | Implement Bit-Plane Slicing |
| **7.** | Implement Basic Compression |
| **8.** | Implement LZW Compression |
| **9.** | Program for Upsampling and Downsampling of an image |
| **10.** | Perform the following:<br>1. Image Steganography<br>2. Visible Watermarking |
| **11.** | Program for 2D Convolution in frequency domain in an input image |
| **12.** | Implement Lowpass Filters in Frequency Domain. |

**Practical 1:**

Perform the following:

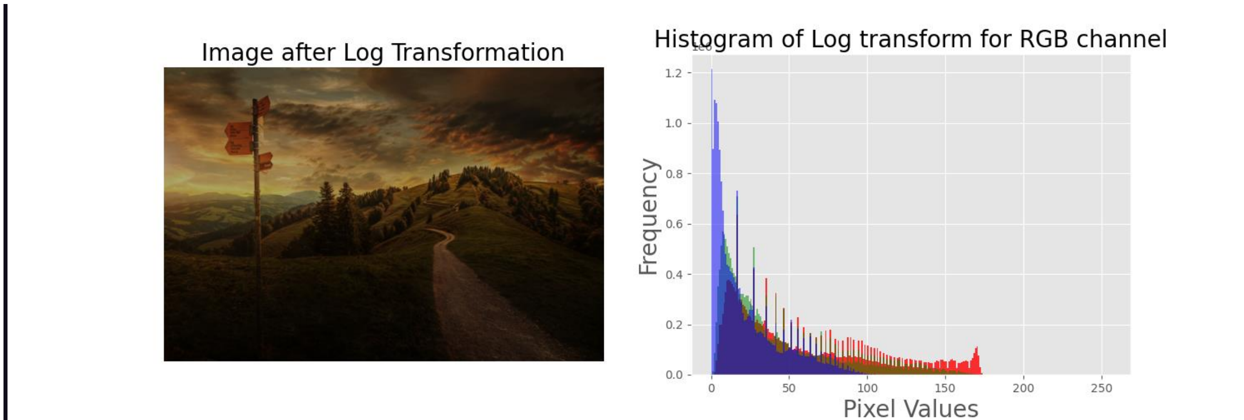
1. Showing Histogram of an Image
2. Log Transformation
3. Power Log Transformation
4. Contrast Stretching
5. Thresholding Operations

Code:

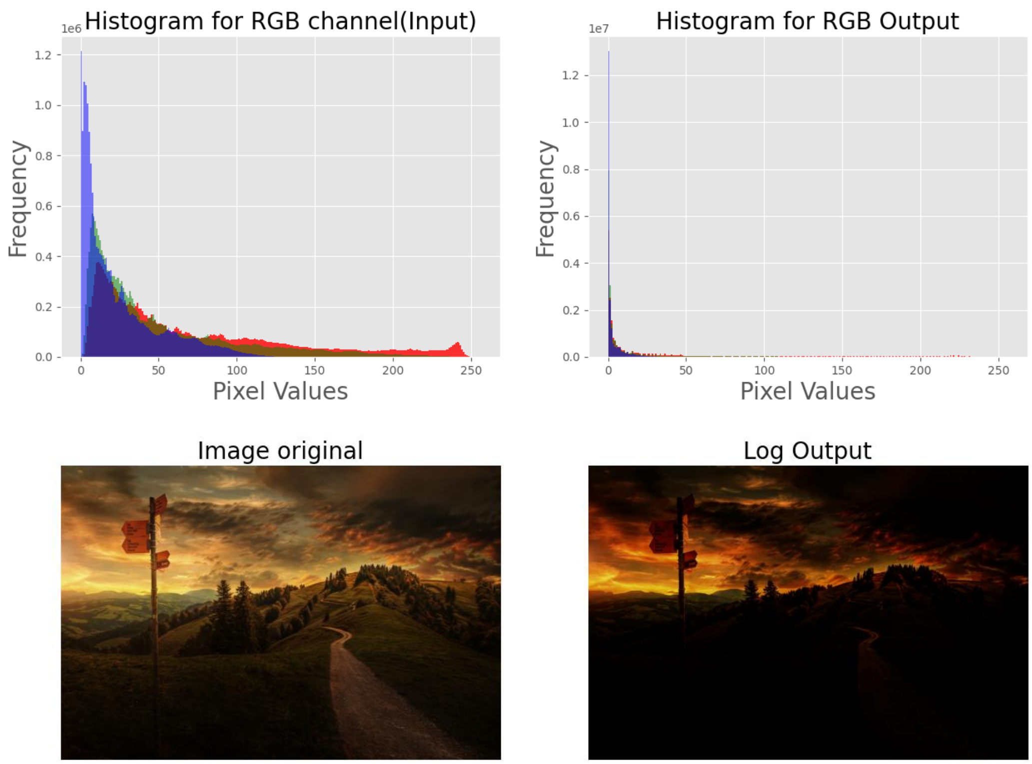
```
import numpy as np
from skimage import data, img_as_float, img_as_ubyte, exposure, io,
color
from skimage.io import imread
from skimage.exposure import cumulative_distribution
from skimage.restoration import denoise_bilateral, denoise_nl_means,
estimate_sigma

# from skimage.measure import compare_psnr
from skimage.util import random_noise
from skimage.color import rgb2gray
from PIL import Image, ImageEnhance, ImageFilter
from scipy import ndimage, misc
import matplotlib.pylab as pylab


def plot_image(image, title=""):
    pylab.title(title, size=20), pylab.imshow(image)
    pylab.axis("off")


def plot_hist(r, g, b, title=""):
    r, g, b = img_as_ubyte(r), img_as_ubyte(g), img_as_ubyte(b)
    pylab.hist(np.array(r).ravel(), bins=256, range=(0, 256),
color="r", alpha=0.8)
    pylab.hist(np.array(g).ravel(), bins=256, range=(0, 256),
color="g", alpha=0.5)
```

```
    pylab.hist(np.array(b).ravel(), bins=256, range=(0, 256),
color="b", alpha=0.5)
    pylab.xlabel("Pixel Values", size=20),
    pylab.ylabel("Frequency", size=20)
    pylab.title(title, size=20)


im = Image.open("image.jpeg")
im_r, im_g, im_b = im.split()
pylab.style.use("ggplot")
pylab.figure(figsize=(15, 5))
pylab.subplot(121), plot_image(im, "Original Image")
pylab.subplot(122), plot_hist(im_r, im_g, im_b, "Histpgram of RGB
Image")
pylab.show()
```


```
# Log Transformation

im = im.point(lambda i: 255 * np.log(1 + i / 255))
im_r, im_g, im_b = im.split()
pylab.style.use("ggplot")
pylab.figure(figsize=(15, 5))
pylab.subplot(121), plot_image(im, "Image after Log Transformation")
pylab.subplot(122), plot_hist(
    im_r, im_g, im_b, "Histogram of Log transform for RGB channel"
)
pylab.show()
```

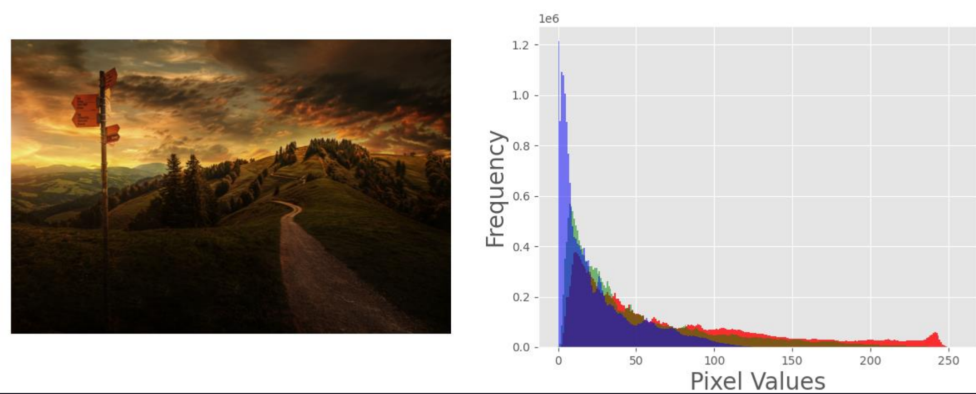


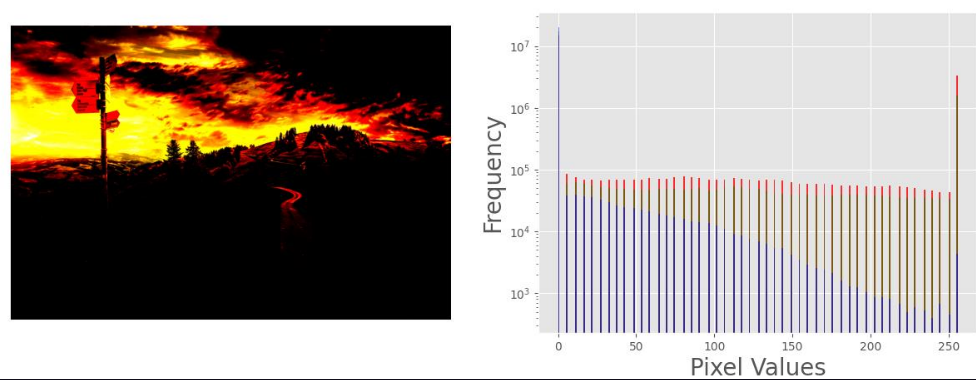

```
# Power Log Transformation

im = img_as_float(imread("image.jpeg"))
gamma = 2.5
im1 = im**gamma
pylab.style.use("ggplot")
pylab.figure(figsize=(15, 5))
pylab.subplot(121), plot_hist(
    im[..., 0], im[..., 1], im[..., 2], "Histogram for RGB
channel(Input)"
)
pylab.subplot(122), plot_hist(
    im1[..., 0], im1[..., 1], im1[..., 2], "Histogram for RGB Output"
)
pylab.show()
pylab.figure(figsize=(15, 5))
pylab.subplot(121), plot_image(im, "Image original")
pylab.subplot(122), plot_image(im1, "Log Output")
pylab.show()
```



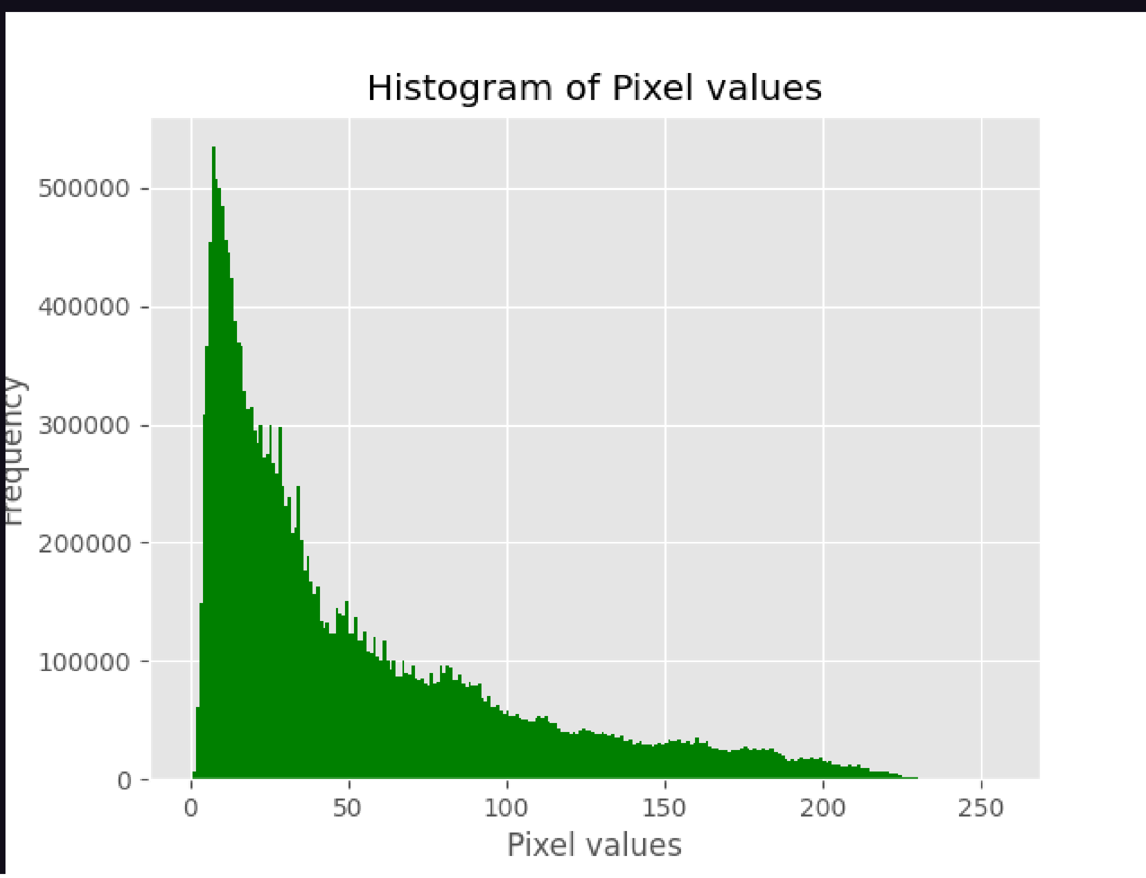

```
# Constrast Streching

im = Image.open("image.jpeg")
im_r, im_g, im_b = im.split()
pylab.style.use("ggplot")
pylab.figure(figsize=(15, 5))
pylab.subplot(121)
plot_image(im)
pylab.subplot(122)
plot_hist(im_r, im_g, im_b)
pylab.show()


def contrast(c):
    return 0 if c < 50 else (255 if c > 150 else (255 * c - 22950) /
48)
```

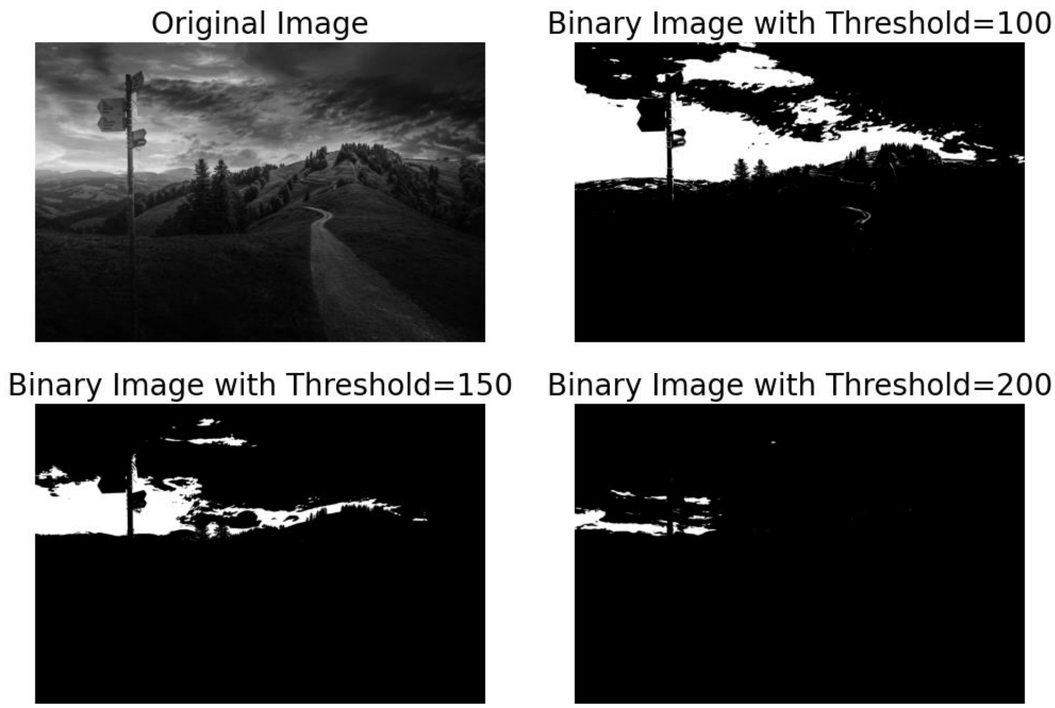
```
im1 = im.point(contrast)
im_r, im_g, im_b = im1.split()
pylab.style.use("ggplot")
pylab.figure(figsize=(15, 5))
pylab.subplot(121)
plot_image(im1)
pylab.subplot(122)
plot_hist(im_r, im_g, im_b)
pylab.yscale("log", base=10)
pylab.show()
```



```
# Thresholding Operations

im = Image.open("image.jpeg").convert("L")
pylab.hist(np.array(im).ravel(), bins=256, range=(0, 256), color="g")
pylab.xlabel("Pixel values"), pylab.ylabel("Frequency")
pylab.title("Histogram of Pixel values")
pylab.show()
pylab.figure(figsize=(12, 18))
pylab.gray()
pylab.subplot(221), plot_image(im, "Original Image")
pylab.axis("off")
th = [0, 50, 100, 150, 200]
for i in range(2, 5):
    im1 = im.point(lambda x: x > th[i])
    pylab.subplot(2, 2, i), plot_image(im1, "Binary Image with
Threshold=" + str(th[i]))

pylab.show()
```




Original Image
Binary Image with Threshold=100
Binary Image with Threshold=150
Binary Image with Threshold=200

**Practical 2:**

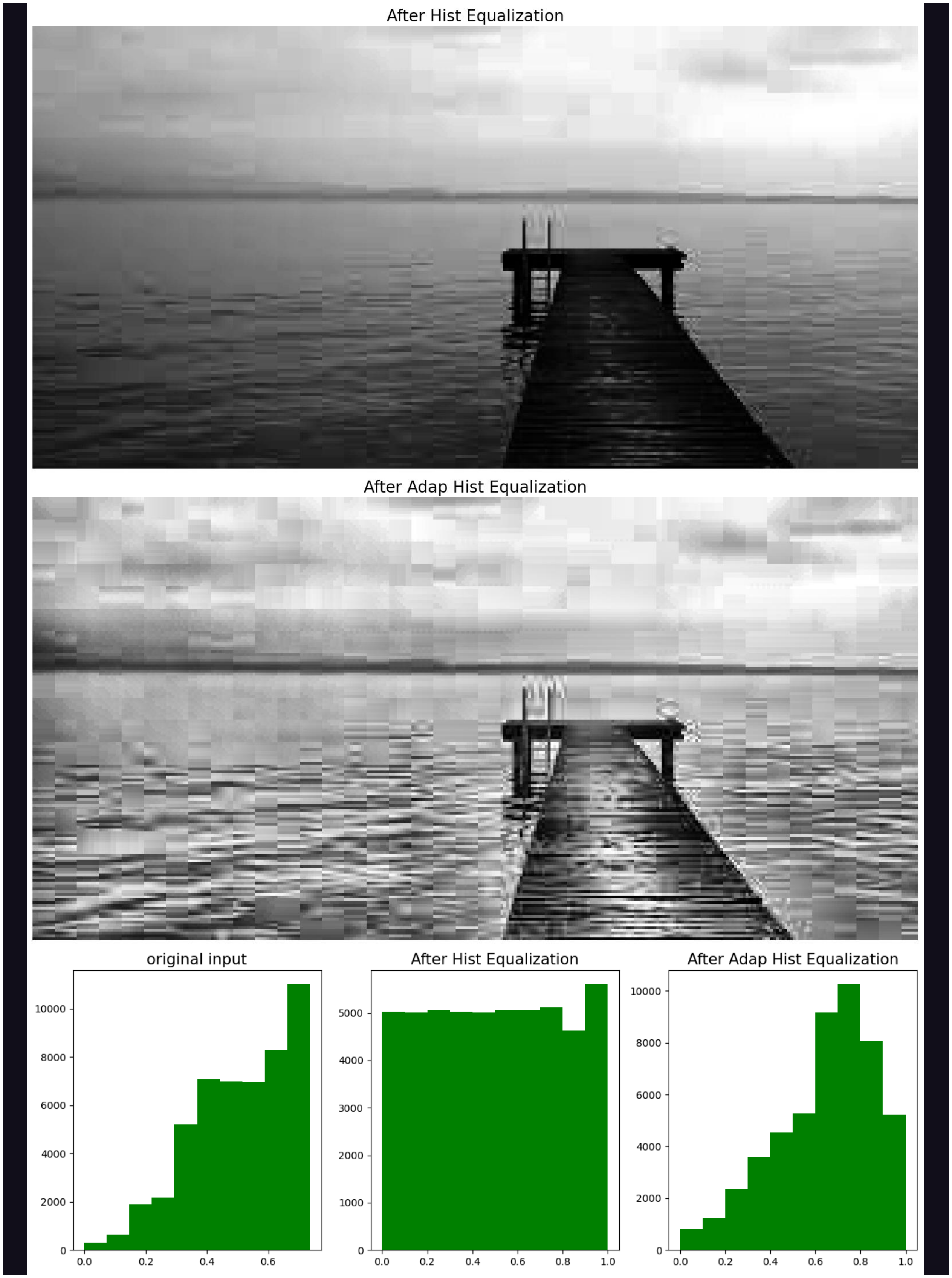
Implement Simple and Adaptive Histogram Equalization

Code:

```
import matplotlib.pylab as pylab
import numpy as np
from PIL import Image, ImageEnhance, ImageFilter
from scipy import misc, ndimage
from skimage import color, data, exposure, img_as_float, img_as_ubyte,
io
from skimage.color import rgb2gray
from skimage.exposure import cumulative_distribution
from skimage.io import imread
from skimage.restoration import (denoise_bilateral, denoise_nl_means,
                                 estimate_sigma)
from skimage.util import random_noise


def plot_image(image, title=""):
    pylab.title(title, size=20), pylab.imshow(image)
    pylab.axis("off")


def plot_hist(r, g, b, title=""):
    r, g, b = img_as_ubyte(r), img_as_ubyte(g), img_as_ubyte(b)
    pylab.hist(np.array(r).ravel(), bins=256, range=(0, 256),
color="r", alpha=0.8)
    pylab.hist(np.array(g).ravel(), bins=256, range=(0, 256),
color="g", alpha=0.5)
    pylab.hist(np.array(b).ravel(), bins=256, range=(0, 256),
color="b", alpha=0.5)
    pylab.xlabel("Pixel Values", size=20),
    pylab.ylabel("Frequency", size=20)
    pylab.title(title, size=20)


# Histogram Equalization (Simple and Adaptive)
```

```
img = rgb2gray(imread("image.jpg"))

# Histogram Equalization
img_eq = exposure.equalize_hist(img)

# Adaptive Histogram Equalization
img_adapteq = exposure.equalize_adapthist(img, clip_limit=0.03)
pylab.gray()
images = [img, img_eq, img_adapteq]

titles = ["original input", "After Hist Equalization", "After Adap
Hist Equalization"]
for i in range(3):
    pylab.figure(figsize=(20, 10))
    plot_image(images[i], titles[i])

pylab.figure(figsize=(15, 5))

for i in range(3):
    pylab.subplot(1, 3, i + 1)
    pylab.hist(images[i].ravel(), color="g"), pylab.title(titles[i],
size=15)

pylab.show()
```

original input


After Hist Equalization
After Adap Hist Equalization
original input
0 2000 4000 6000 8000 10000
0.0 0.2 0.4 0.6
After Hist Equalization
0 1000 2000 3000 4000 5000
0.0 0.2 0.4 0.6 0.8 1.0
After Adap Hist Equalization
0 2000 4000 6000 8000 10000
0.0 0.2 0.4 0.6 0.8 1.0

**Practical 3:**

Perform the following:

1. Derivatives and Gradients
2. Laplacian Filter
3. Sharpening with Laplacian Plot
4. Unsharp Masking
5. Image Negatives

Code:

```
import matplotlib.pylab as pylab
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image, ImageFilter, ImageOps
from scipy import misc, ndimage, signal
from skimage import feature, filters, img_as_float
from skimage.color import rgb2gray
from skimage.filters import laplace
from skimage.io import imread

def plot_image(image, title=""):
    pylab.title(title, size=20), pylab.imshow(image)
    pylab.axis("off")


# Derivatives and Gradients
ker_x = [[-1, 1]]
ker_y = [[-1], [1]]

im = rgb2gray(imread("image.jpeg"))
im_x = signal.convolve2d(im, ker_x, mode="same")
im_y = signal.convolve2d(im, ker_y, mode="same")

im_mag = np.sqrt(im_x**2 + im_y**2)
im_dir = np.arctan(im_y / im_x)
```

```
pylab.gray()
pylab.figure(figsize=(30, 20))
pylab.subplot(231), plot_image(im, "Original")
pylab.subplot(232), plot_image(im_x, "Grad_x")
pylab.subplot(233), plot_image(im_y, "Grad_y")
pylab.subplot(234), plot_image(im_mag, "||grad||")
pylab.subplot(235), plot_image(im_dir, r"$\theta$")
pylab.plot(range(im.shape[1]), im[0, :], "b-",
label=r"$f(x,y)|_{x=0}$", linewidth=5)
pylab.plot(range(im.shape[1]), im_x[0, :], "r-", label=r"$grad_x
(f(x,y))|_{x=0}$")
pylab.title(r"$grad_x (f(x, y))|_{x=0}$", size=30)
pylab.legend(prop={"size": 20})
pylab.show()
```

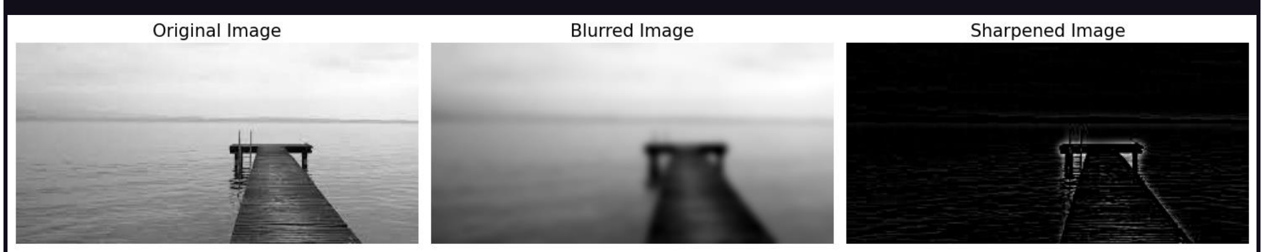

```
# Laplacian Filter
ker_laplacian = [[0, -1, 0], [-1, 4, -1], [0, -1, 0]]
im = rgb2gray(imread("image.jpeg"))
im1 = np.clip(signal.convolve2d(im, ker_laplacian, mode="same"), 0, 1)
pylab.gray()
pylab.figure(figsize=(20, 10))
pylab.subplot(121), plot_image(im, "Original")
```

```
pylab.subplot(122), plot_image(im1, "Laplacian Convolved Image")
pylab.show()
```


```
# Sharpening With laplacian plot
im = rgb2gray(imread("image.jpeg"))
im1 = np.clip(laplace(im) + im, 0, 1)
pylab.figure(figsize=(10, 15))
pylab.subplot(121), plot_image(im, "Original Image")
pylab.subplot(122), plot_image(im1, "Sharpened Image with Laplace")
pylab.tight_layout()
pylab.show()
```


```
# Unsharp Masking
def rgb2gray(im):
    return np.clip(
        0.2989 * im[..., 0] + 0.5870 * im[..., 1] + 0.1140 * im[...,
2], 0, 1
    )
```

```
im = rgb2gray(img_as_float(imread("image.jpeg")))
im_blurred = ndimage.gaussian_filter(im, 3)
im_detail = np.clip(im - im_blurred, 0, 1)

pylab.gray()

fig, axes = pylab.subplots(nrows=2, ncols=3, sharex=True, sharey=True,
figsize=(15, 15))
axes = axes.ravel()
axes[0].set_title("Original Image", size=15), axes[0].imshow(im)
axes[1].set_title("Blurred Image", size=15),
axes[1].imshow(im_blurred)
axes[2].set_title("Sharpened Image", size=15),
axes[2].imshow(im_detail)

alpha = [1, 5, 10]
for i in range(3):
    im_sharp = np.clip(im + alpha[i] * im_detail, 0, 1)
    axes[3 + i].imshow(im_sharp), axes[3 + i].set_title(
        "Sharpened Image, alpha=" + str(alpha[i]), size=15
    )

for ax in axes:
    ax.axis("off")

fig.tight_layout()
pylab.show()
```




```python
# Image Negative
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image, ImageOps


def compute_negative(image_path):
    original_image = Image.open(image_path)
    original_array = np.array(original_image)
    negative_array = 255 - original_array
    negative_image = Image.fromarray(negative_array)
    return original_image, negative_image


def display_images(original_image, negative_image):
    # Display the original and negative images side by side
    fig, axes = plt.subplots(1, 2, figsize=(10, 5))

    axes[0].imshow(original_image)
    axes[0].set_title("Original Image")

    axes[1].imshow(negative_image)
    axes[1].set_title("Negative Image")

    for ax in axes:
        ax.axis("off")
    plt.show()


if __name__ == "__main__":
    image_path = "image.jpeg"
    original, negative = compute_negative(image_path)
```

```
    display_images(original, negative)
```

Original Image

Negative Image

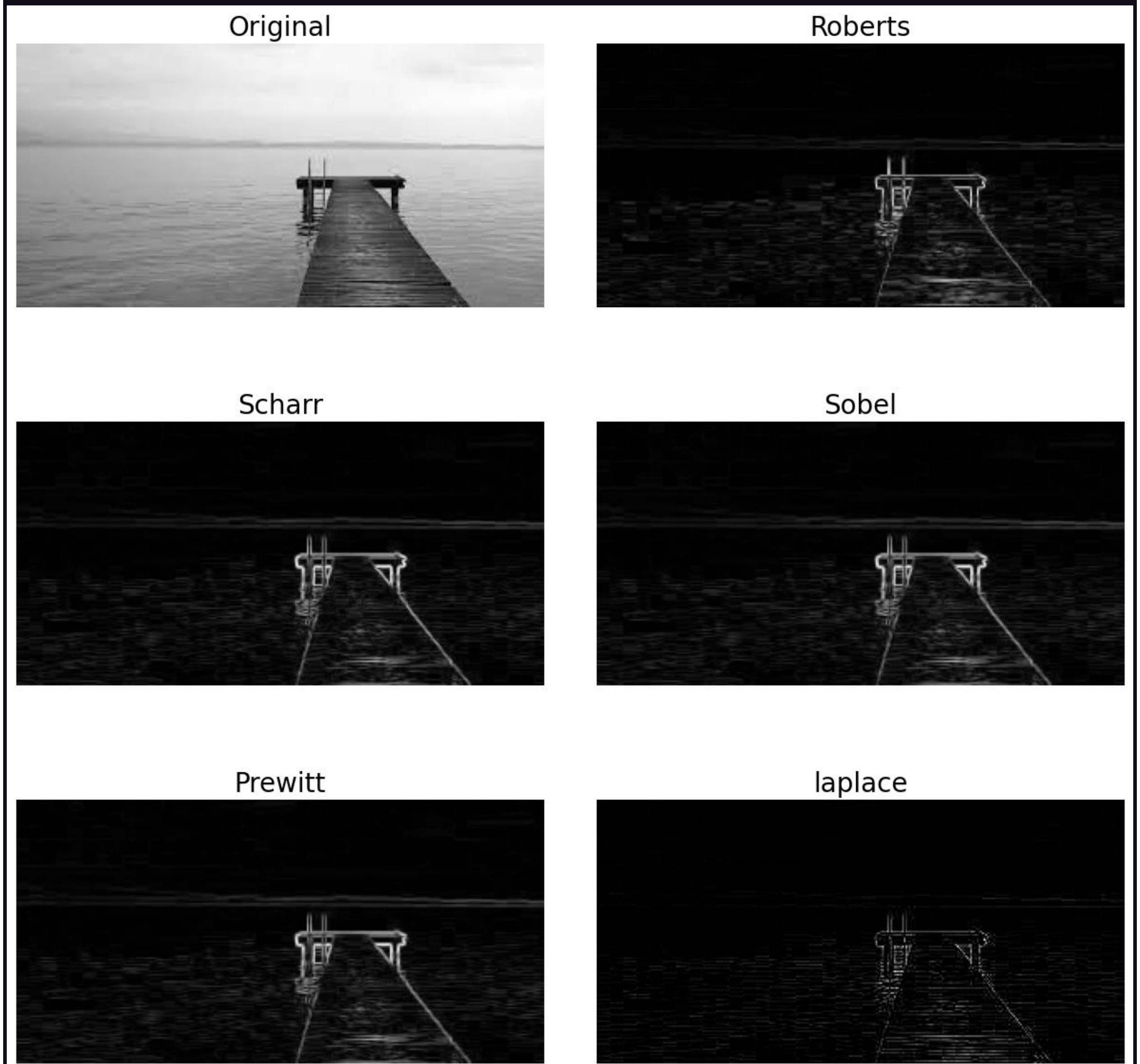
**Practical 4:**

Implement Sobel Image Detector & Canny Edge Detector using Scikit-Image

Code:

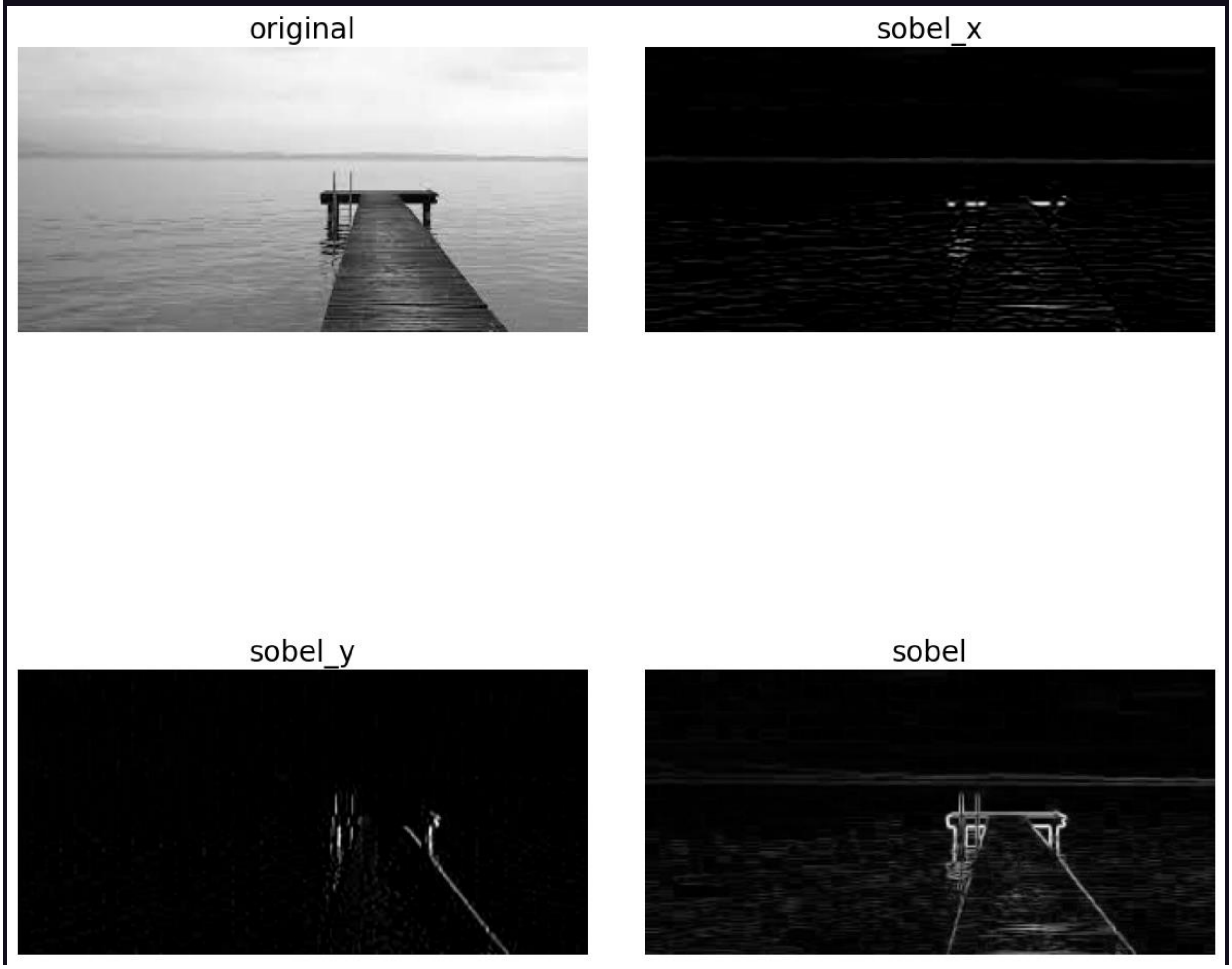
```
import matplotlib.pylab as pylab
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image, ImageFilter
from scipy import misc
from scipy import ndimage as ndi
from scipy import signal
from skimage import feature, filters, img_as_float
from skimage.color import rgb2gray
from skimage.io import imread
from skimage.util import random_noise


def plot_image(image, title=""):
    pylab.title(title, size=20), pylab.imshow(image)
    pylab.axis("off")

pylab.gray()

im = imread("image.jpeg")
im = rgb2gray(im)
pylab.figure(figsize=(15, 15))
pylab.subplot(3, 2, 1), plot_image(im, "Original")
edges = filters.roberts(im)
pylab.subplot(3, 2, 2), plot_image(edges, "Roberts")
edges = filters.scharr(im)
pylab.subplot(3, 2, 3), plot_image(edges, "Scharr")
edges = filters.sobel(im)
pylab.subplot(3, 2, 4), plot_image(edges, "Sobel")
edges = filters.prewitt(im)
pylab.subplot(3, 2, 5), plot_image(edges, "Prewitt")
```

```
edges = np.clip(filters.laplace(im), 0, 1)
pylab.subplot(3, 2, 6), plot_image(edges, "laplace")
pylab.subplots_adjust(wspace=0.1, hspace=0.1)
pylab.show()
```

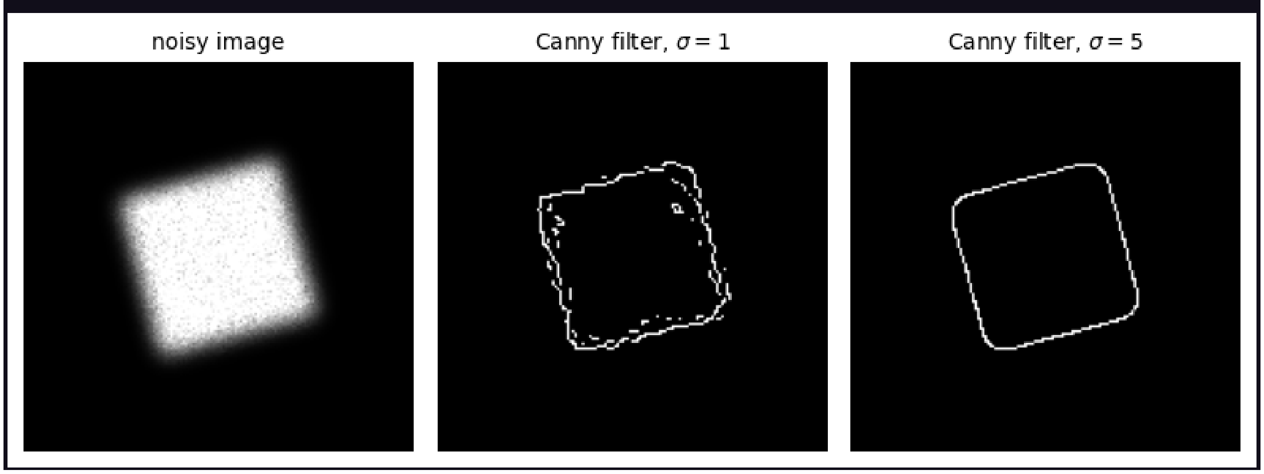



```
# Sobel Image Detector with scikit-image

im = imread("image.jpeg")
im = rgb2gray(im)
pylab.figure(figsize=(15, 15))
pylab.subplot(2, 2, 1)
```

```
plot_image(im, "original")
pylab.subplot(2, 2, 2)
edges_x = filters.sobel_h(im)
plot_image(np.clip(edges_x, 0, 1), "sobel_x")
pylab.subplot(2, 2, 3)
edges_y = filters.sobel_v(im)
plot_image(np.clip(edges_y, 0, 1), "sobel_y")
pylab.subplot(2, 2, 4)
edges = filters.sobel(im)
plot_image(np.clip(edges, 0, 1), "sobel")
pylab.subplots_adjust(wspace=0.1, hspace=0.1)
pylab.show()
```



```python
# Canny Edge Detector with scikit-image

image = np.zeros((128, 128), dtype=float)
image[32:-32, 32:-32] = 1
image = ndi.rotate(image, 15, mode="constant")
image = ndi.gaussian_filter(image, 4)
image = random_noise(image, mode="speckle", mean=0.05)

edges1 = feature.canny(image)
edges2 = feature.canny(image, sigma=3)
fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(8, 3))

ax[0].imshow(image, cmap="gray")
ax[0].set_title("noisy image", fontsize=10)

ax[1].imshow(edges1, cmap="gray")
ax[1].set_title(r"Canny filter, $\sigma=1$", fontsize=10)

ax[2].imshow(edges2, cmap="gray")
ax[2].set_title(r"Canny filter, $\sigma=5$", fontsize=10)

for a in ax:
    a.axis("off")

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



**Practical 5:**

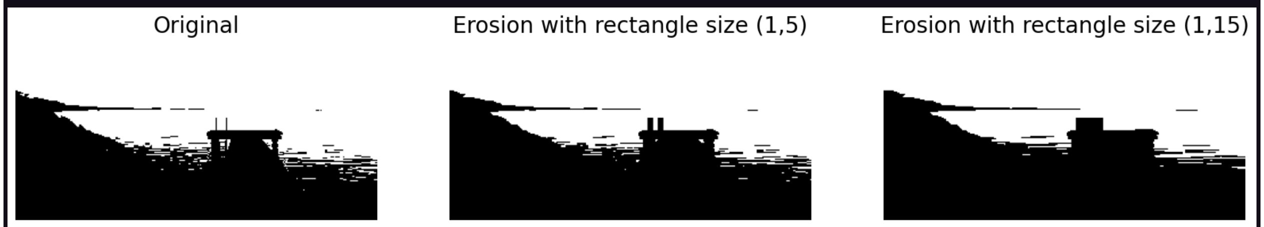
Perform the following:

1. Erosion
2. Dilation
3. Opening and Closing
4. Skeletonization
5. Covex Hull
6. White and Black Top-Hats
7. Boundary Extraction

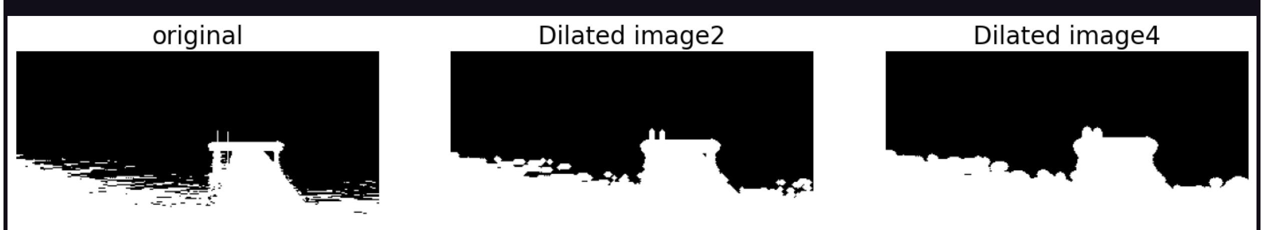
Code:

```
import matplotlib.pylab as pylab
import numpy as np
from skimage import img_as_float
from skimage.color import rgb2gray
from skimage.io import imread
from skimage.morphology import (binary_closing, binary_dilation,
                                binary_erosion, binary_opening,
black_tophat,
                                convex_hull_image, disk, rectangle,
                                skeletonize, square, white_tophat)

def plot_image(image, title=""):
    pylab.title(title, size=20), pylab.imshow(image)
    pylab.axis("off")


# Erosion
im = rgb2gray(imread("image.jpeg"))
im[im <= 0.5] = 0 # Create binary image with fixed threshold 0.5
im[im > 0.5] = 1
pylab.figure(figsize=(20, 20))
pylab.subplot(1,3,1), plot_image(im, 'Original')
im1 = binary_erosion(im, rectangle(1,5))
```

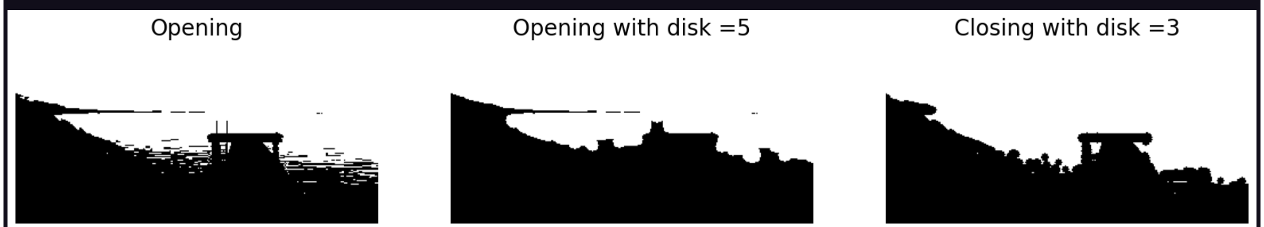
```
pylab.subplot (1,3,2), plot_image(im1, 'Erosion with rectangle size
(1,5)')
im1 = binary_erosion(im, rectangle(1,15))
pylab.subplot(1,3,3), plot_image(im1, 'Erosion with rectangle size
(1,15)')
pylab.show()
```




```
# Dilation
im = img_as_float(imread("image.jpeg"))
im = 1 - im[..., 2]
im[im <= 0.5] = 0
im[im > 0.5] = 1
pylab.gray()
pylab.figure(figsize=(18, 9))
pylab.subplot(131)
pylab.imshow(im)
pylab.title("original", size=20)
pylab.axis("off")
for d in range(1, 3):
    pylab.subplot(1, 3, d + 1)
    im1 = binary_dilation(im, disk(2 * d))
    pylab.imshow(im1)
    pylab.title("Dilated image" + str(2 * d), size=20)
    pylab.axis("off")
pylab.show()
```

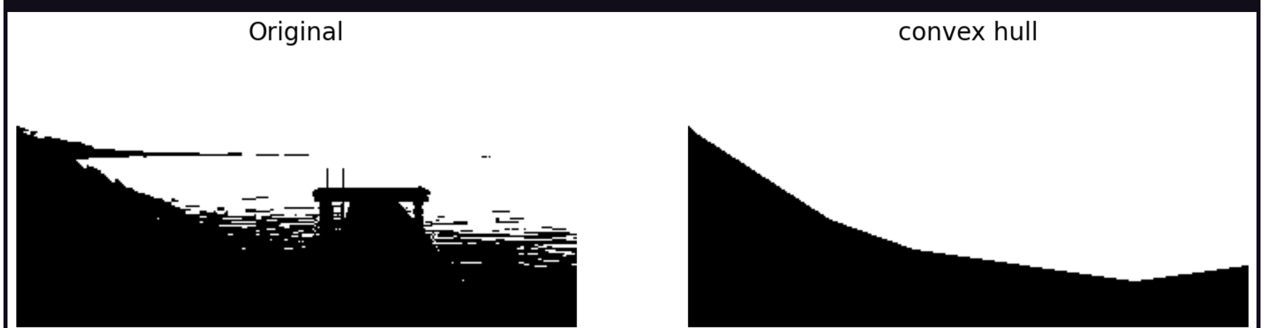


```
# Opening and Closing
im=rgb2gray(imread("image.jpeg"))
im[im<=0.5]=0
im[im>0.5]=1
pylab.gray()
pylab.figure(figsize=(20,14))
pylab.subplot(1,3,1), plot_image(im, 'Opening')
im1 = binary_opening(im, disk(5))
pylab.subplot(1,3,2), plot_image(im1, 'Opening with disk ='+str(5))
im1 = binary_closing(im, disk(3))
pylab.subplot(1,3,3), plot_image(im1, 'Closing with disk ='+str(3))
```




```
# Skeletonization
def plot_horizontally(original, filtered, filter_name, sz=(18,7)):
    pylab.gray()
    pylab.figure(figsize =sz)
    pylab.subplot(1,2,1), plot_image(original, 'Original')
    pylab.subplot(1,2,2), plot_image(filtered, filter_name)
    pylab.show()

im = img_as_float(imread('image.jpeg')[...,2])
threshold =0.5
im[im<=threshold]=0
im[im>threshold]=1
skeleton=skeletonize(im)
plot_images_horizontally(im, skeleton, 'skeleton', sz=(18,9))
```

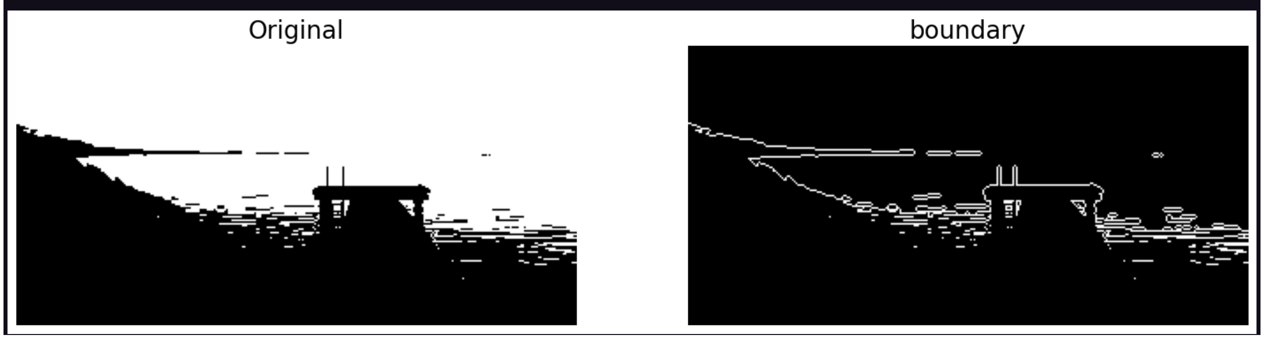

```
# Convex Hull
from skimage.morphology import convex_hull_image

im=rgb2gray(imread('image.jpeg'))
threshold=0.5
im[im<=threshold]=0
im[im>threshold]=1
con_hull=convex_hull_image(im)
plot_images_horizontally(im ,con_hull,'convex hull', sz=(18,9))
```




```
# White-Hat, Black-Hat
from skimage.morphology import white_tophat, black_tophat,square

im = imread('image.jpeg')[...,2]
threshold=0.5
im[im<=threshold]=0
im[im>threshold]=1
im1=white_tophat(im,square(10))
im2=black_tophat(im,square(10))
pylab.figure(figsize=(20,15))
pylab.subplot(1,2,1), plot_image(im1, 'White Top-hat')
```

```
pylab.subplot(1,2,2), plot_image(im2, 'Black Top-hat')
pylab.show()
```


```
# Boundary Extraction
from skimage.morphology import binary_erosion

im=rgb2gray(imread('image.jpeg'))
threshold=0.5
im[im<=threshold]=0
im[im>threshold]=1
boundary=im-binary_erosion(im)
plot_images_horizontally(im,boundary,'boundary',sz=(18,9))
```

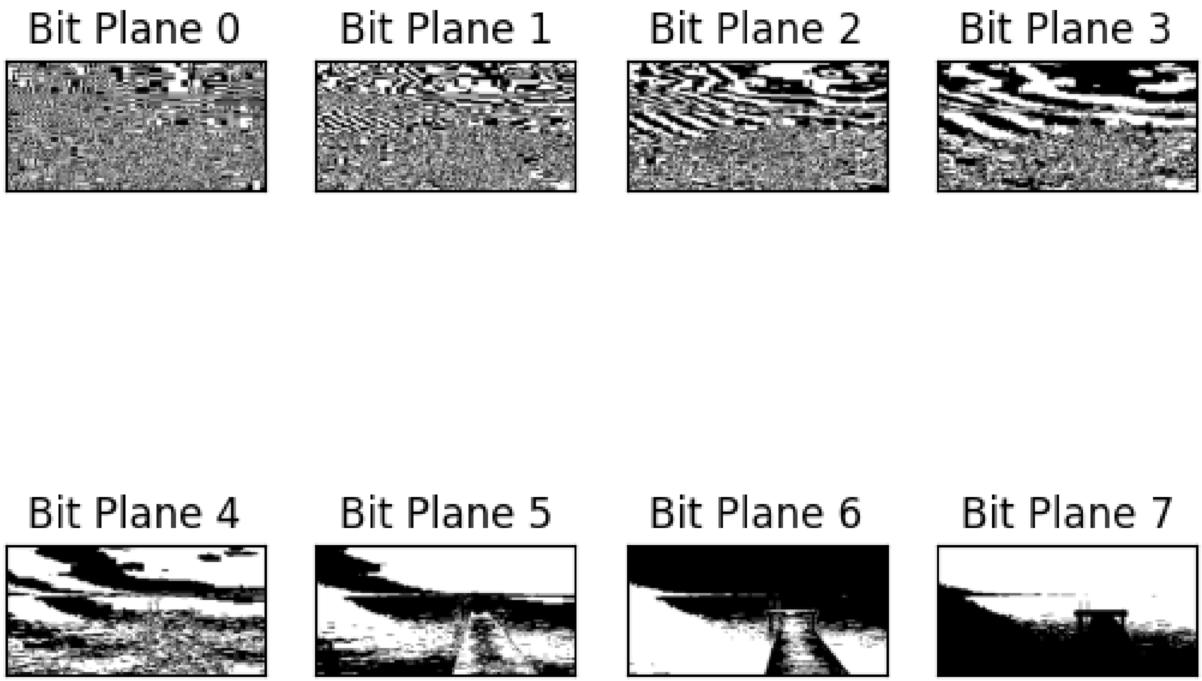


**Practical 6:**

Implement Bit-Plane Slicing

Code:

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

input_file = "image.jpeg"
input_img = Image.open(input_file)

# converts image into numpy array
input_img = input_img.convert("L")
input_arr = np.array(input_img)

# perform bit-plane slicng
bit_planes = []
for i in range(8):
    bit_plane = np.bitwise_and(input_arr, 2**i)
    bit_plane = bit_plane.astype(np.uint8)
    bit_planes.append(bit_plane)

# display the resulting images
for i in range(8):
    plt.subplot(2, 4, i + 1)
    plt.imshow(bit_planes[i], cmap="gray")
    plt.title("Bit Plane {}".format(i))
    plt.xticks([])
    plt.yticks([])

plt.show()
```


Bit Plane 0
Bit Plane 1
Bit Plane 2
Bit Plane 3
Bit Plane 4
Bit Plane 5
Bit Plane 6
Bit Plane 7

**Practical 7:**

Implement Basic Compression

Code:

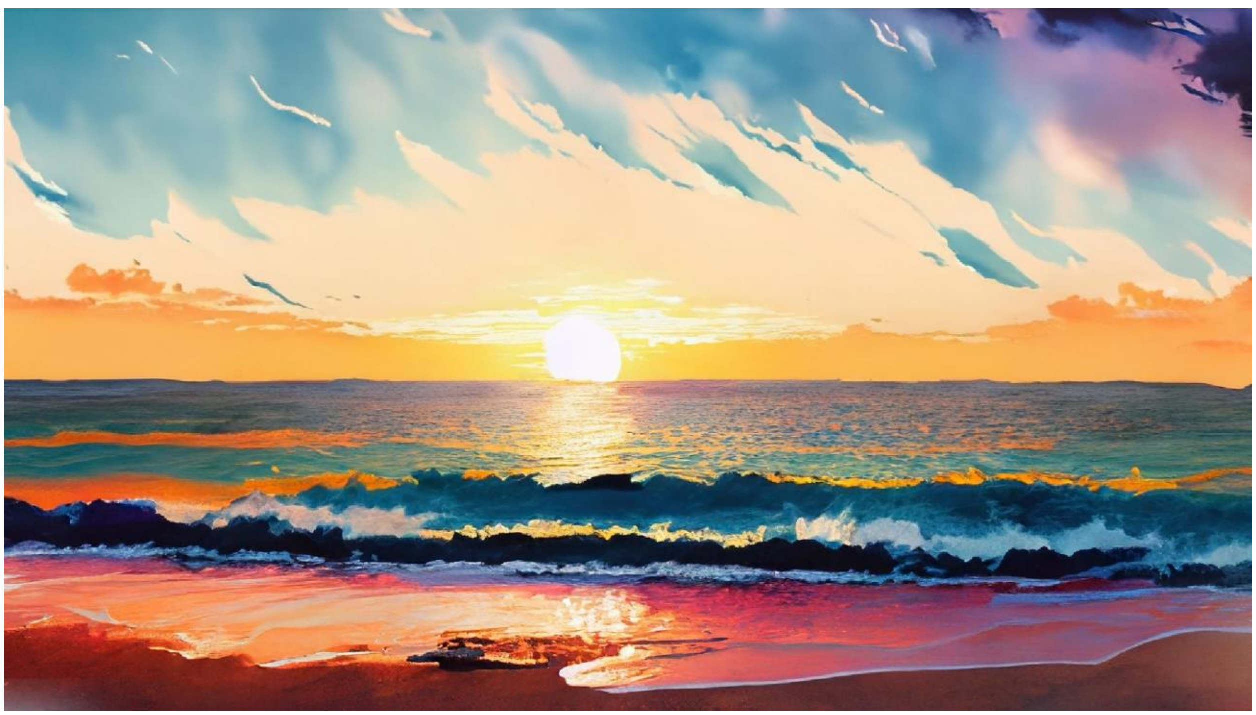
```
from PIL import Image
import os

bmp_file = "image.bmp"
bmp_img = Image.open(bmp_file)

# compress the bmp image into jpeg format
jpeg_img =bmp_img.convert("RGB")
jpeg_file = "output.jpeg"
jpeg_img.save(jpeg_file,"JPEG",quality=50)

# print the size of both files
bmp_size = os.path.getsize(bmp_file)
jpeg_size = os.path.getsize(jpeg_file)
print("Original BMP file size: ",bmp_size,"bytes")
print("Compressed JPEG file size: ",jpeg_size,"bytes")

jpeg_img.save(jpeg_file)
```

Output:

```
Original BMP file size:  6220856 bytes
Compressed JPEG file size:  151634 bytes
```

Original Image:

Compressed Image:



**Practical 8:**

Implement LZW Compression

Code:

```
def lzw_compress(data):
    dict = {chr(i): i for i in range(256)}
    result = []
    w = ""
    for c in data:
        wc = w + c
        if wc in dict:
            w = wc
        else:
            result.append(dict[w])
            dict[wc] = len(dict)
            w = c

    if w:
        result.append(dict[w])
    return result


input = "ABABABABABABABABA"

# Compress using LZW Compression
compressed_data = lzw_compress(input)

# Compute
original_size = len(input)
compressed_size = len(compressed_data)
compression_ratio = compressed_size / original_size

redundant_data = original_size - compressed_size

print("Input Data: ", input)
print("Compressed Data: ", compressed_data)
```

```python
print("Original Size: ", original_size, "bytes")
print("Compressed Size: ", compressed_size, "bytes")
print("Compression Ratio: ", compression_ratio)
print("Redundancy Ratio: ", 1 - compression_ratio)
print("Redundant Data: ", redundant_data, "bytes")
```

Output:

```
Input Data:  ABABABABABABABABA

Compressed Data:  [65, 66, 256, 258, 257, 260, 259, 65]

Original Size:  17 bytes

Compressed Size:  8 bytes

Compression Ratio:  0.47058823529411764

Redundancy Ratio:  0.5294117647058824

Redundant Data:  9 bytes
```

```python
import numpy as np
from PIL import Image


def lzw_compress(input_sequence):
    dict_size = 256
    dict = {chr(i): i for i in range(dict_size)}
    compressed_data = []
    w = ""

    for symbol in input_sequence:
        ws = w + symbol
        if ws in dict:
            w = ws
        else:
            compressed_data.append(dict[w])
            dict[ws] = dict_size
            dict_size += 1
            w = symbol

    if w:
        compressed_data.append(dict[w])

    return compressed_data


def imagetolzw(image):
    img = Image.open(image).convert("L")

    pixels = list(img.getdata())
    pixel_sequence = "".join([chr(pixel) for pixel in pixels])

    compressed_data = lzw_compress(pixel_sequence)

    return compressed_data


# Example
image = "image.bmp"
compressed_data = imagetolzw(image)
```

```python
print(f"Compressed Data Size: {len(compressed_data)} elements")
```

Output:

```
Compressed Data Size: 481471 elements
```

**Practical 9:**

Program for Upsampling and Downsampling of an image.

Code:

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import misc, ndimage


def display_image(image, title):
    plt.imshow(image, cmap="gray")
    plt.title(title)
    plt.axis("off")
    plt.show()


def upsampling_downsampling_demo(image, scale_factor):
    upsampled_image = ndimage.zoom(image, zoom=scale_factor, order=3)
    downsampled_image = ndimage.zoom(image, zoom=1 / scale_factor,
order=3)

    display_image(image, "Original Image")
    display_image(upsampled_image, f"Upsampled Image(Scale
Factor:{scale_factor})")
    display_image(downsampled_image, f"Downsampled Image(Scale
Factor:{scale_factor})")

    print(f"Original Image Shape: {image.shape}")
    print(f"Upsampled Image Shape: {upsampled_image.shape}")
    print(f"Downsampled Image Shape: {downsampled_image.shape}")


def main():
    image = misc.ascent()
    scale_factor = 2
    upsampling_downsampling_demo(image, scale_factor)
```

```python
if __name__ == "__main__":
    main()
```


```
Original Image Shape: (512, 512)
Upsampled Image Shape: (1024, 1024)
Downsampled Image Shape: (256, 256)
```

**Practical 10:**

Perform the following:

1. Image Steganography
2. Visible Watermarking

Code:

```
# Image Steganography

import cv2
import numpy as np
from PIL import Image


def encode_text_in_image(image_path, text, output_path):
    img = Image.open("/content/cat.jpeg")
    img = img.convert("RGBA")
    data = np.array(img)

    binary_text = "".join(format(ord(i), "08b") for i in text)
    binary_text += "1111111111111110"
    if len(binary_text) > data.size:
        raise ValueError("Text is too long to be encoded in the
image")

    # Encode text into image
    data_flat = data.flatten()
    for i in range(len(binary_text)):
        if binary_text[i] == "0":
            if data_flat[i] % 2 != 0:
                data_flat[i] -= 1  # Make it even
        else:
            if data_flat[i] % 2 == 0:
                data_flat[i] += 1  # Make it odd

    # Reshape and save the new image
```

```
    data = data_flat.reshape(data.shape)
    encoded_img = Image.fromarray(data)
    encoded_img.save(output_path)

    return output_path


def decode_text_from_image(image_path):
    # Load the image
    img = Image.open(image_path)
    data = np.array(img)

    # Flatten the image array and extract the LSB of each element
    data_flat = data.flatten()
    binary_text = "".join(["1" if i % 2 else "0" for i in data_flat])

    # Split the binary text into 8-bit chunks and convert to
characters
    chars = [binary_text[i : i + 8] for i in range(0,
len(binary_text), 8)]
    text = "".join([chr(int(c, 2)) for c in chars])

    # Detect the delimiter to signify end of text
    delimiter = text.find("\xff\xff")
    if delimiter != -1:
        text = text[:delimiter]

    return text


# Set the paths and text
original_image_path = "image.jpeg"
encoded_image_path = "encoded_image.png"
text_to_hide = "Hello, world. We are in Class!"

# Encode the text into the image
encode_text_in_image(original_image_path, text_to_hide,
encoded_image_path)
```

```
# Decode the text from the image
hidden_text = decode_text_from_image(encoded_image_path)
print("Decoded text:", hidden_text)
```

## Output:

```
Decoded text: Hello, world. We are in
Class!ÿþ333Ý333ÝÝ»µ»[±»333333333333333333»±ÿ
```

```python
# Visible Watermarking

def watermark_image(image_path, watermark_path, output_path):
    """Embeds a visible watermark onto an image."""

    # Load images
    image = cv2.imread(image_path)
    watermark = cv2.imread(watermark_path, cv2.IMREAD_UNCHANGED)

    # Resize watermark (adjust as needed)
    watermark_resized = cv2.resize(
        watermark, (int(image.shape[1] * 0.2), int(image.shape[0] *
0.2))
    )

    # Define placement (adjust coordinates)
    x_offset, y_offset = 10, 10

    # Overlay with transparency
    alpha = watermark_resized[:, :, 3] / 255.0  # Extract alpha
channel
    for c in range(0, 3):
        image[
            y_offset : y_offset + watermark_resized.shape[0],
            x_offset : x_offset + watermark_resized.shape[1],
            c,
        ] = (
            alpha * watermark_resized[:, :, c]
            + (1 - alpha)
            * image[
                y_offset : y_offset + watermark_resized.shape[0],
                x_offset : x_offset + watermark_resized.shape[1],
                c,
            ]
        )

    # Save
    cv2.imwrite(output_path, image)
```

```python
# Example usage
image_path = "cat.jpeg"
watermark_path = "watermark.png"
output_path = "watermarked_image.jpg"

watermark_image(image_path, watermark_path, output_path)

print("Watermarked image saved to:", output_path)
```

**Practical 11:**

Program for 2D Convolution in frequency domain in an input image.

Code:

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.fft import fft2, fftshift, ifft2
from scipy.signal import convolve2d
from skimage import color, data, util

image = color.rgb2gray(data.coffee())
image = util.img_as_float(image)

kernel = np.array([[1, 2, 1], [2, 4, 2], [1, 2, 1]])
kernel = kernel / np.sum(kernel)

fft_image = fft2(image)
fft_kernel = fft2(kernel, s=image.shape)

fft_result = fft_image * fft_kernel
convolved_image = ifft2(fft_result).real
direct_convolution = convolve2d(image, kernel, mode="same")
fig, ax = plt.subplots(1, 3, figsize=(15, 5))

ax[0].imshow(image, cmap="gray")
ax[0].set_title("Original Image")
ax[0].axis("off")
ax[1].imshow(convolved_image, cmap="gray")
ax[1].set_title("Convolved Image(Frequency Domain)")
ax[1].axis("off")
ax[2].imshow(direct_convolution, cmap="gray")
ax[2].set_title("Convolved Image(Spatial Domain)")
ax[2].axis("off")
plt.show()
```


Original Image
Convolved Image(Frequency Domain)
Convolved Image(Spatial Domain)

**Practical 12:**

Implement Lowpass Filters in Frequency Domain.

Code:

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.fft import fft2, fftshift, ifft2, ifftshift
from skimage import color, data, util


# Helper functions to create filters
def ideal_lowpass_filter(shape, cutoff):
    rows, cols = shape
    center_row, center_col = rows // 2, cols // 2
    filter = np.zeros((rows, cols))
    for x in range(cols):
        for y in range(rows):
            if (x - center_col) ** 2 + (y - center_row) ** 2 <
cutoff**2:
                filter[y, x] = 1
    return filter


def butterworth_lowpass_filter(shape, cutoff, order):
    rows, cols = shape
    center_row, center_col = rows // 2, cols // 2
    filter = np.zeros((rows, cols))
    for x in range(cols):
        for y in range(rows):
            distance = np.sqrt((x - center_col) ** 2 + (y -
center_row) ** 2)
            filter[y, x] = 1 / (1 + (distance / cutoff) ** (2 *
order))
    return filter


def gaussian_lowpass_filter(shape, cutoff):
```

```
    rows, cols = shape
    center_row, center_col = rows // 2, cols // 2
    filter = np.zeros((rows, cols))
    for x in range(cols):
        for y in range(rows):
            distance = np.sqrt((x - center_col) ** 2 + (y -
center_row) ** 2)
            filter[y, x] = np.exp(-(distance**2) / (2 * (cutoff**2)))
    return filter


# Load an example image and convert it to grayscale
image = color.rgb2gray(data.coffee())
image = util.img_as_float(image)

# FFT of the image
fft_image = fftshift(fft2(image))

# Filter parameters
cutoff = 50  # Cutoff frequency
order = 2  # Order for Butterworth filter

# Create filters
ideal_filter = ideal_lowpass_filter(image.shape, cutoff)
butterworth_filter = butterworth_lowpass_filter(image.shape, cutoff,
order)
gaussian_filter = gaussian_lowpass_filter(image.shape, cutoff)

# Apply filters
ideal_result = ifft2(ifftshift(fft_image * ideal_filter)).real
butterworth_result = ifft2(ifftshift(fft_image *
butterworth_filter)).real
gaussian_result = ifft2(ifftshift(fft_image * gaussian_filter)).real

# Plotting
fig, ax = plt.subplots(2, 4, figsize=(20, 10))
ax[0, 0].imshow(image, cmap="gray")
ax[0, 0].set_title("Original Image")
ax[0, 0].axis("off")
```

```python
ax[0, 1].imshow(ideal_filter, cmap="gray")
ax[0, 1].set_title("Ideal Filter")
ax[0, 1].axis("off")

ax[0, 2].imshow(butterworth_filter, cmap="gray")
ax[0, 2].set_title("Butterworth Filter")
ax[0, 2].axis("off")
ax[0, 3].imshow(gaussian_filter, cmap="gray")
ax[0, 3].set_title("Gaussian Filter")
ax[0, 3].axis("off")

ax[1, 1].imshow(ideal_result, cmap="gray")
ax[1, 1].set_title("Ideal Filter Result")
ax[1, 1].axis("off")

ax[1, 2].imshow(butterworth_result, cmap="gray")
ax[1, 2].set_title("Butterworth Result")
ax[1, 2].axis("off")
ax[1, 3].imshow(gaussian_result, cmap="gray")
ax[1, 3].set_title("Gaussian Result")
ax[1, 3].axis("off")

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