COMPUTER VISION AND IMAGE ANALYSIS

Tasks

Task 1: Perform the following steps:

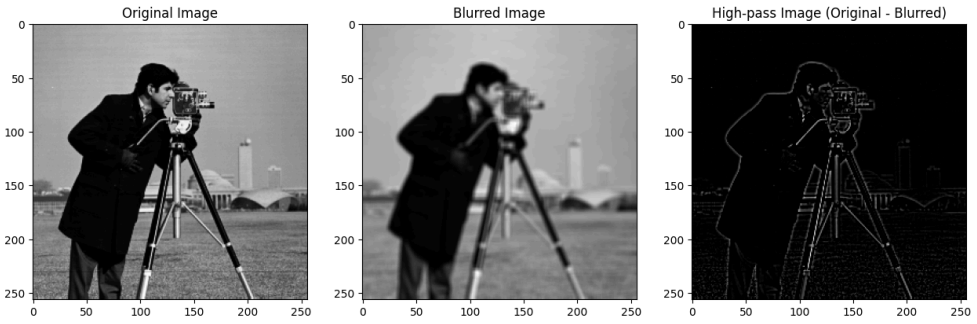
Step 1: Read the input image (preferably 256 x 256 gray scale cameraman image)

Step 2: Blur the image by passing it through low pass filter. (Chose the mask in such a way that blurring effect is visible)

Step 3: Subtract the blurred version of the image from the input image and comment on the observed result.

```
import numpy as np

import cv2

import matplotlib.pyplot as plt


# Step 1: Read the input image

image = cv2.imread('cameraman.png', cv2.IMREAD_GRAYSCALE)


# Step 2: Blur the image using a low-pass filter

kernel = np.ones((5, 5), np.float32) / 25

blurred_image = cv2.filter2D(image, -1, kernel)


# Step 3: Subtract the blurred version of the image from the input image

high_pass_image = cv2.subtract(image, blurred_image)


# Display the images

plt.figure(figsize=(10, 10))


plt.subplot(1, 3, 1)

plt.title('Original Image')

plt.imshow(image, cmap='gray')
```

```
plt.subplot(1, 3, 2)

plt.title('Blurred Image')

plt.imshow(blurred_image, cmap='gray')



plt.subplot(1, 3, 3)

plt.title('High-pass Image (Original - Blurred)')

plt.imshow(high_pass_image, cmap='gray')


plt.show()
```



Task 2: Perform the following steps

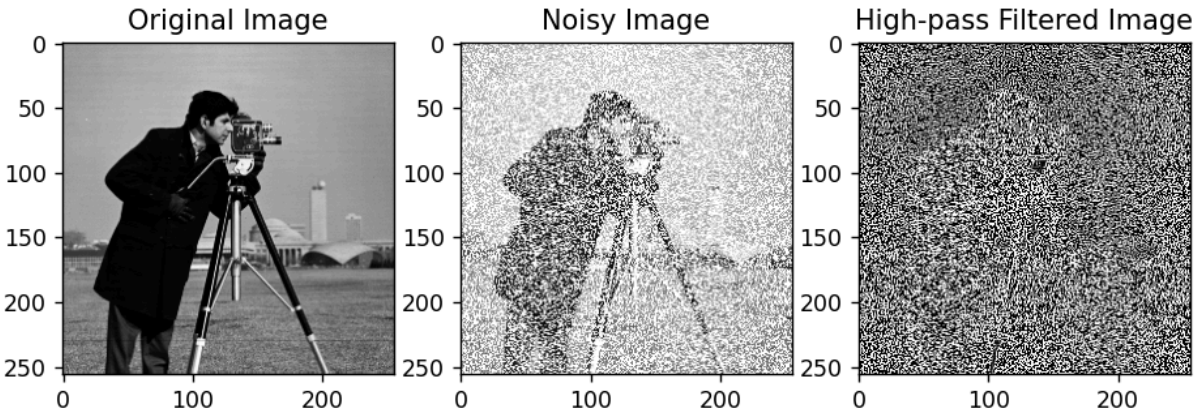
Step 1: Read the input image (gray scale image)

Step 2: Add random noise to the input image.

Step 3: Pass the noisy image through high pass filter and comment on the observed result.

```
# Step 1: Read the input image

image = cv2.imread('cameraman.png', cv2.IMREAD_GRAYSCALE)


# Step 2: Add random noise to the input image
```

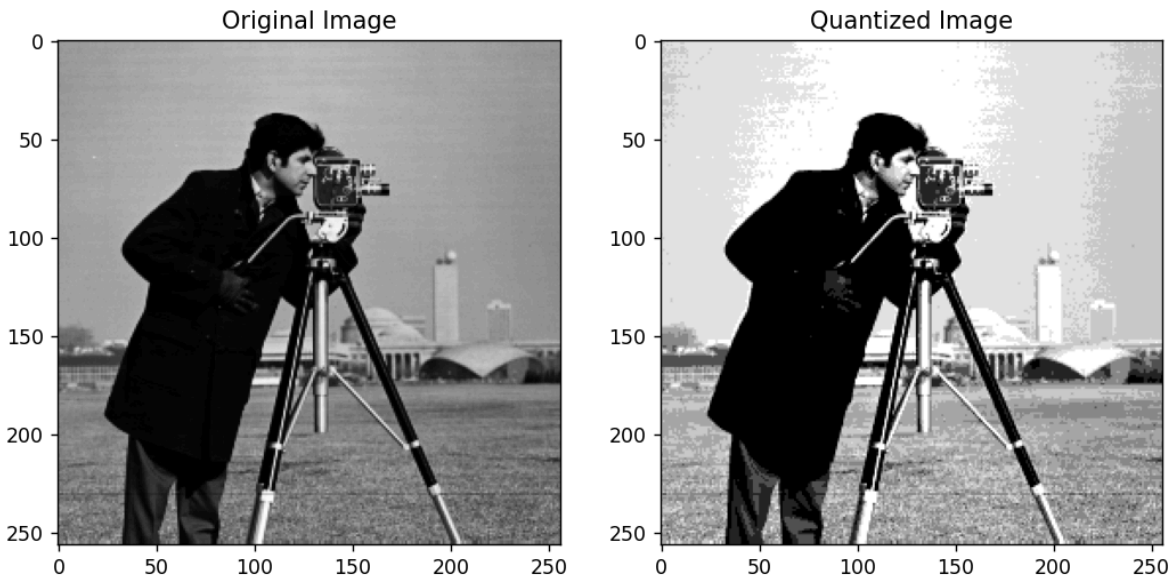
```
noise = np.random.normal(0, 25, image.shape).astype(np.uint8)
noisy_image = cv2.add(image, noise)

# Step 3: Pass the noisy image through a high-pass filter
# Using a simple Laplacian filter for high-pass filtering
high_pass_filter = np.array([[-1, -1, -1],
                             [-1,  8, -1],
                             [-1, -1, -1]])
high_pass_image = cv2.filter2D(noisy_image, -1, high_pass_filter)

# Display the images
plt.figure(figsize=(10, 10))

plt.subplot(1, 3, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')

plt.subplot(1, 3, 2)
plt.title('Noisy Image')
plt.imshow(noisy_image, cmap='gray')

plt.subplot(1, 3, 3)
plt.title('High-pass Filtered Image')
plt.imshow(high_pass_image, cmap='gray')

plt.show()
```



Task 3: Write a python code to perform Vector Quantization of the input matrix as discussed in the class.

```
from sklearn.cluster import KMeans


def vector_quantization(image, n_clusters):
    # Reshape the image into a 2D array of pixels
    pixels = image.reshape(-1, 1)

    # Perform KMeans clustering
    kmeans = KMeans(n_clusters=n_clusters, random_state=0).fit(pixels)

    # Replace each pixel value with its corresponding cluster center
    quantized_image =
kmeans.cluster_centers_[kmeans.labels_].reshape(image.shape)

    return quantized_image


# Example usage
image = cv2.imread('cameraman.png', cv2.IMREAD_GRAYSCALE)
n_clusters = 8  # Number of gray levels
```

```
quantized_image = vector_quantization(image, n_clusters)


plt.figure(figsize=(10, 5))


plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')


plt.subplot(1, 2, 2)
plt.title('Quantized Image')
plt.imshow(quantized_image, cmap='gray')


plt.show()
```

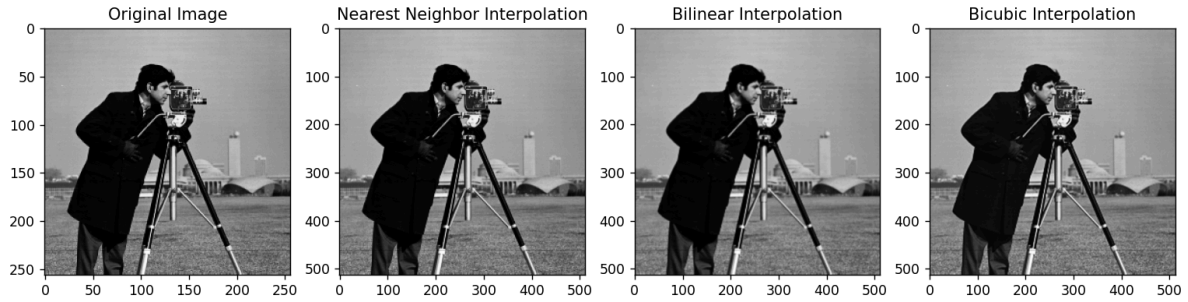


Task 4: Write a python code to perform (a) nearest neighbor (b) Bilinear and (c) Bicubic interpolation of the input image and comment on the observed result.

```
# Interpolation functions
```

```
def nearest_neighbor_interpolation(image, new_size):
    return cv2.resize(image, new_size, interpolation=cv2.INTER_NEAREST)

def bilinear_interpolation(image, new_size):
    return cv2.resize(image, new_size, interpolation=cv2.INTER_LINEAR)

def bicubic_interpolation(image, new_size):
    return cv2.resize(image, new_size, interpolation=cv2.INTER_CUBIC)

# Example usage
image = cv2.imread('cameraman.png', cv2.IMREAD_GRAYSCALE)
new_size = (512, 512)

nn_image = nearest_neighbor_interpolation(image, new_size)
bilinear_image = bilinear_interpolation(image, new_size)
bicubic_image = bicubic_interpolation(image, new_size)

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

plt.subplot(1, 4, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')

plt.subplot(1, 4, 2)
plt.title('Nearest Neighbor Interpolation')
plt.imshow(nn_image, cmap='gray')

plt.subplot(1, 4, 3)
plt.title('Bilinear Interpolation')
```

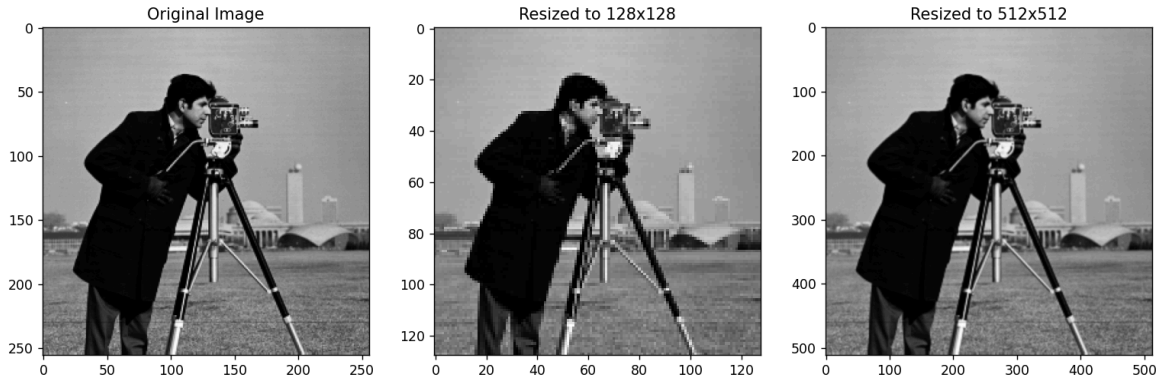
```
plt.imshow(bilinear_image, cmap='gray')


plt.subplot(1, 4, 4)
plt.title('Bicubic Interpolation')
plt.imshow(bicubic_image, cmap='gray')


plt.show()
```



Task 5: Write a python code to perform resizing of the input image. If the input image size is 256 x 256, first resize it to 128 x 128, then resize the input image to 512 x 512 and comment on the observed result.

```
# Example usage
image = cv2.imread('cameraman.png', cv2.IMREAD_GRAYSCALE)


# Resize to 128x128
resized_down = cv2.resize(image, (128, 128),
interpolation=cv2.INTER_LINEAR)


# Resize to 512x512
resized_up = cv2.resize(image, (512, 512),
interpolation=cv2.INTER_LINEAR)


plt.figure(figsize=(15, 5))
```

```
plt.subplot(1, 3, 1)

plt.title('Original Image')

plt.imshow(image, cmap='gray')



plt.subplot(1, 3, 2)

plt.title('Resized to 128x128')

plt.imshow(resized_down, cmap='gray')



plt.subplot(1, 3, 3)

plt.title('Resized to 512x512')

plt.imshow(resized_up, cmap='gray')



plt.show()
```

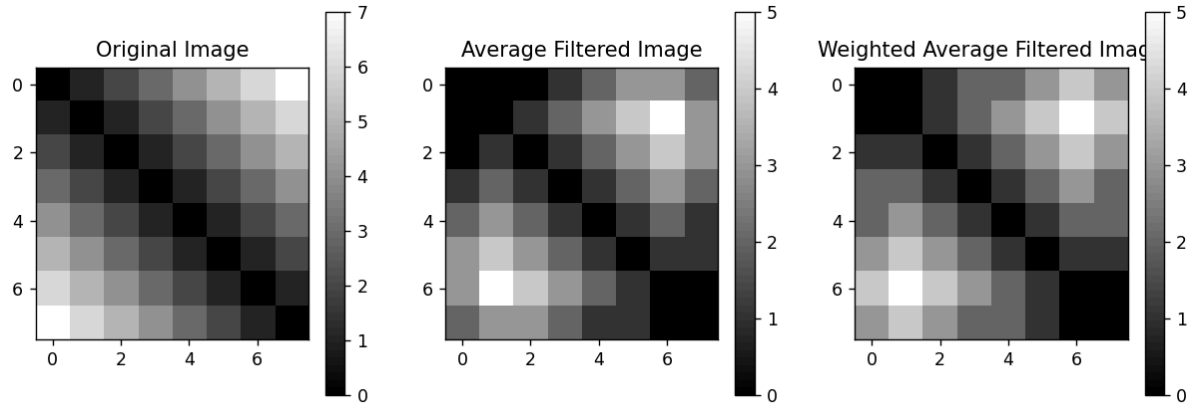


Task 6: Generate an 8 x 8 image which is given by f[m,n]=|m-n| for m,n=0,1,2,3,4,5,6,7. Pass this image through 3x 3 average filter, 3 x 3 weighted average filter. Use subplot to plot the generated image, average filtered image and weighted average filtered image and comment on the observed result.

```
import numpy as np

import matplotlib.pyplot as plt

from scipy.ndimage import convolve



```

```
# Define the original 8x8 image
def generate_image(size):
                return    np.abs(np.arange(size).reshape(size,    1)    -
np.arange(size).reshape(1, size))


# Define a 3x3 average filter
def average_filter(image):
    kernel = np.ones((3, 3)) / 9.0
    return convolve(image, kernel, mode='constant', cval=0.0)


# Define a 3x3 weighted average filter
def weighted_average_filter(image):
    kernel = np.array([[1, 2, 1],
                       [2, 4, 2],
                       [1, 2, 1]]) / 16.0
    return convolve(image, kernel, mode='constant', cval=0.0)


# Generate the 8x8 image
image = generate_image(8)


# Apply the filters
average_filtered_image = average_filter(image)
weighted_average_filtered_image = weighted_average_filter(image)


# Plot the images
plt.figure(figsize=(12, 4))


plt.subplot(1, 3, 1)
plt.title("Original Image")
```

```
plt.imshow(image, cmap='gray')

plt.colorbar()



plt.subplot(1, 3, 2)

plt.title("Average Filtered Image")

plt.imshow(average_filtered_image, cmap='gray')

plt.colorbar()



plt.subplot(1, 3, 3)

plt.title("Weighted Average Filtered Image")

plt.imshow(weighted_average_filtered_image, cmap='gray')

plt.colorbar()



plt.show()
```



Task 7: Generate the 16 x 16 image which is given below

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & 0 \\
0 & 1 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 2 & 1 & 0 \\
0 & 1 & 2 & 3 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 3 & 2 & 1 & 0 \\
0 & 1 & 2 & 3 & 4 & 5 & 5 & 5 & 5 & 5 & 5 & 4 & 3 & 2 & 1 & 0 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 6 & 6 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 6 & 6 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
0 & 1 & 2 & 3 & 4 & 5 & 5 & 5 & 5 & 5 & 5 & 4 & 3 & 2 & 1 & 0 \\
0 & 1 & 2 & 3 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 3 & 2 & 1 & 0 \\
0 & 1 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 2 & 1 & 0 \\
0 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

Use some logic to generate this image. **Hint:** The image has outer ring made of gray level value ‘0’, inner ring of gray level value ‘1’ and so on.

```python
import numpy as np

import matplotlib.pyplot as plt

from scipy.ndimage import convolve


def generate_ring_image(size):

    image = np.zeros((size, size), dtype=int)

    for i in range(size):

        for j in range(size):

            image[i, j] = min(i, j, size - i - 1, size - j - 1)

    return image


image = generate_ring_image(16)


plt.figure(figsize=(8, 8))

plt.title("16x16 Ring Image")
```

```
plt.imshow(image, cmap='gray')

plt.show()
```