Name: Bidyut Kr. Das

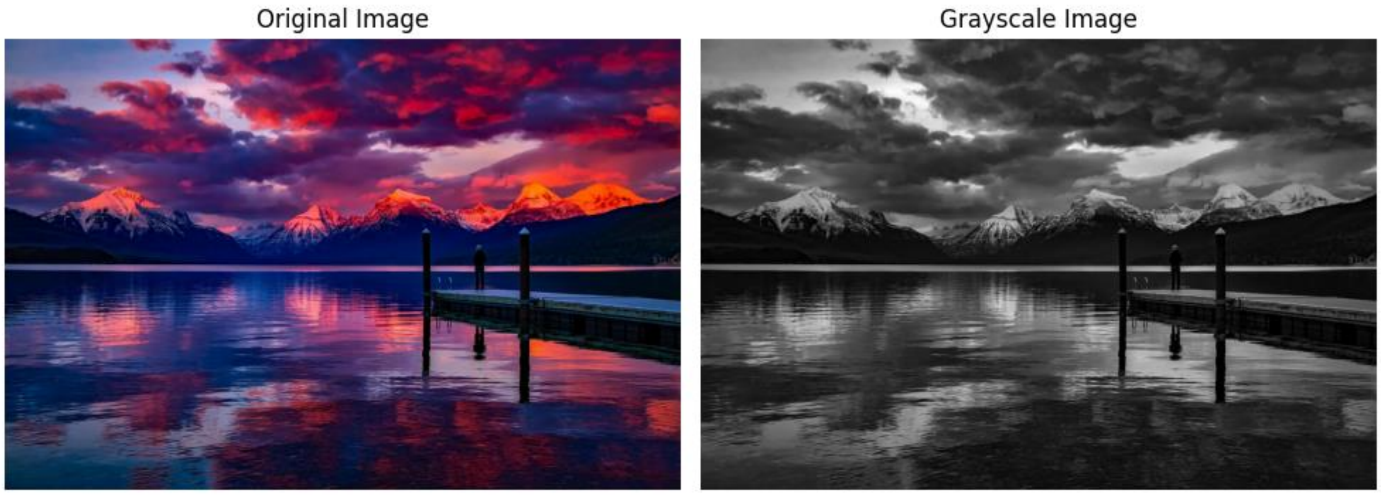
Id: 221001011060 Batch: BCS-4D

Question 1: Write a program in python to convert colored image to gray image.

Use rations (0.28:0.59:0.10)

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
def rgb_to_gray(image):
   image = Image.open(image)
   image_array = np.array(image)
   r, g, b = image_array[:, :, 0], image_array[:, :, 1], image_array[:, :, 2]
   grayscale = 0.28 * r + 0.59 * g + 0.10 * b
   grayscale_image = grayscale.astype(np.uint8)
   plt.figure(figsize=(10, 5))
   plt.subplot(1, 2, 1)
   plt.title("Original Image")
   plt.imshow(image_array)
   plt.axis("off")
   plt.subplot(1,2, 2)
   plt.title("Grayscale Image")
   plt.imshow(grayscale_image, cmap="gray")
   plt.axis("off")
   plt.tight_layout()
   plt.show()
   return grayscale_image
input_image = './image.jpg'
gray_image = rgb_to_gray(input_image)
```

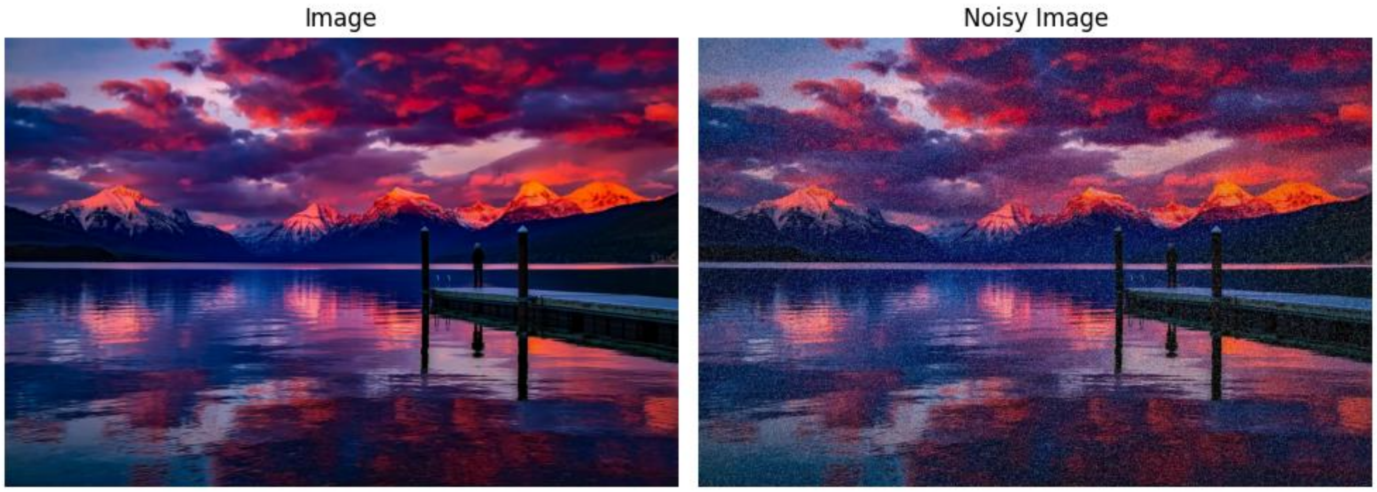


Question 2: Given an image perform the following operations accordingly

i. Add salt pepper noise to it
ii. Perform mean filter on the noise image
iii. Perform gaussian filter on the noise image

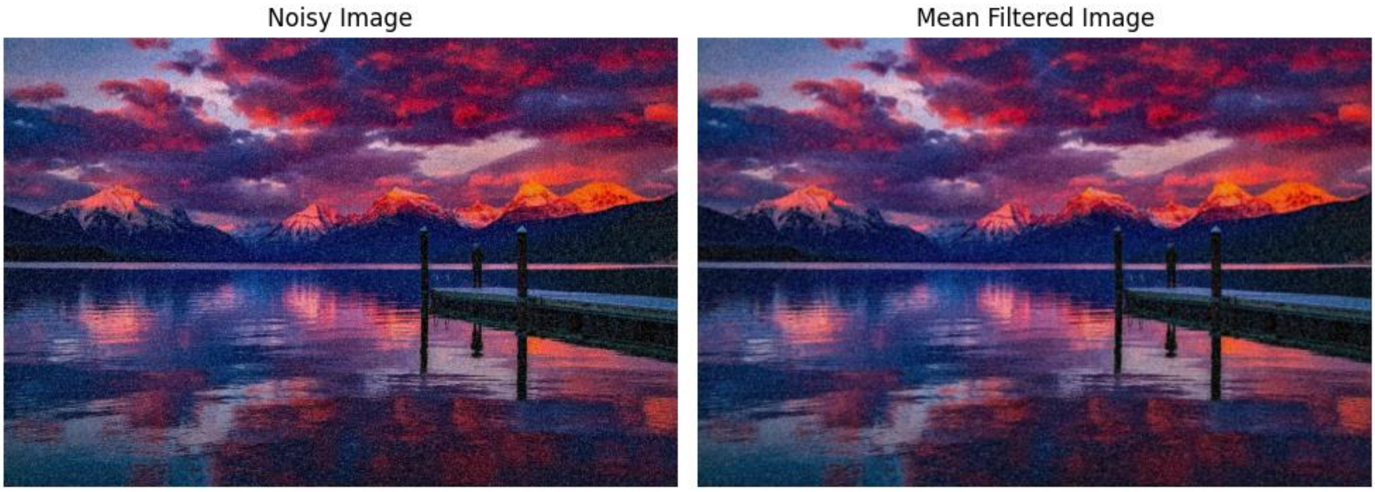
Code:

```
def add_salt_pepper_noise(image, salt_prob, pepper_prob):

    image = Image.open(image)

    image = np.array(image)

    noisy_image = np.copy(image)

    total_pixels = image.size


    num_salt = int(total_pixels * salt_prob)

    num_pepper = int(total_pixels * pepper_prob)


    coords_salt = [np.random.randint(0, i, num_salt) for i in image.shape]

    noisy_image[coords_salt[0], coords_salt[1]] = 255  # Assuming 8-bit grayscale


    coords_pepper = [np.random.randint(0, i, num_pepper) for i in image.shape]

    noisy_image[coords_pepper[0], coords_pepper[1]] = 0


    return noisy_image


salt_prob = 0.02

pepper_prob = 0.02


salt_image = add_salt_pepper_noise(input_image, salt_prob, pepper_prob)


image = Image.open(input_image)

image_arr = np.array(image)


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

plt.subplot(1,2,1)

plt.title("Image")

plt.imshow(image_arr)

plt.axis("off")


plt.subplot(1,2,2)
plt.title("Noisy Image")
plt.imshow(salt_image)
plt.axis("off")

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



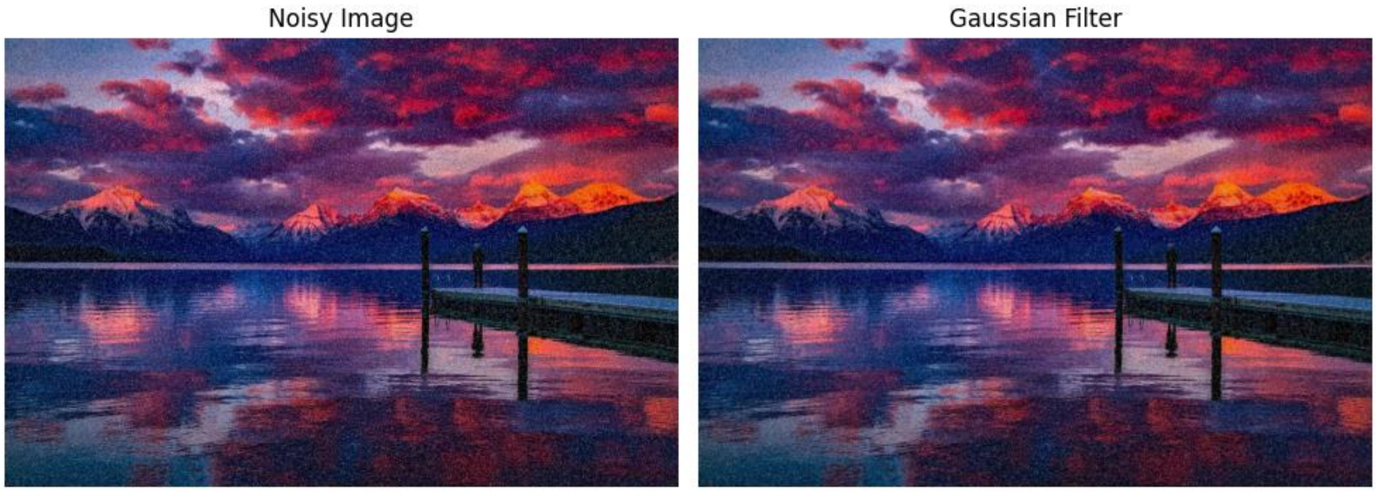
II Code:

```
def mean_filter(image, kernel_size = 3):
    height, width, channels = image.shape
    pad = kernel_size // 2
    blurred_image = np.zeros_like(image, dtype=np.uint8)
       for c in range(channels):
        padded_channel = np.zeros((height + 2 * pad, width + 2 * pad))
        padded_channel[pad:pad + height, pad:pad + width] = image[:, :, c]
        for i in range(height):
            for j in range(width):
                kernel_region = padded_channel[i:i + kernel_size, j:j + kernel_size]
                kernel_mean = np.sum(kernel_region) / (kernel_size * kernel_size)
                blurred_image[i, j, c] = kernel_mean
    return blurred_image
blurred_image = mean_filter(salt_image, kernel_size=5)
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.title("Noisy Image")
plt.imshow(salt_image)
plt.axis("off")
plt.subplot(1,2,2)

plt.title("Mean Filtered Image")

plt.imshow(blurred_image)

plt.axis("off")

plt.tight_layout()

plt.show()
```



III Code:

```
#creation of gaussian_kernel

def gaussian_kernel(size, sigma=1):

    kernel_1d = np.linspace(-size//2, size//2, size)

    kernel_1d = np.exp(-0.5 * (kernel_1d / sigma) ** 2

    # Normalize the kernel so that the sum of all values equals 1

    kernel_1d /= np.sum(kernel_1d)

    # Create a 2D kernel by taking the outer product of the 1D kernel with itself

    kernel_2d = np.outer(kernel_1d, kernel_1d)

    # Normalize the 2D kernel (this is just a safeguard, should already sum to 1)

    kernel_2d /= np.sum(kernel_2d)

    return kernel_2d

def gaussian_blur_color(image, kernel_size=5, sigma=1):

    kernel = gaussian_kernel(kernel_size, sigma)

    height, width, channels = image.shape

    pad = kernel_size // 2

    blurred_image = np.zeros_like(image, dtype=np.uint8)


    for c in range(channels):

        channel = image[:, :, c]
        padded_channel = np.pad(channel, ((pad, pad), (pad, pad)), mode='constant', constant_values=0)

        for i in range(height):

            for j in range(width):

                region = padded_channel[i:i + kernel_size, j:j + kernel_size]

                blurred_image[i, j, c] = np.sum(region * kernel)

    return blurred_image

kernel_size = 5

sigma = 1

blurred_image = gaussian_blur_color(salt_image, kernel_size, sigma)

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

plt.subplot(1,2,1)

plt.title("Noisy Image")

plt.imshow(salt_image)

plt.axis("off")

plt.subplot(1,2,2)

plt.title("Gaussian Filter")

plt.imshow(blurred_image)

plt.axis("off")

plt.tight_layout()

plt.show()
```



Noisy Image

Gaussian Filter