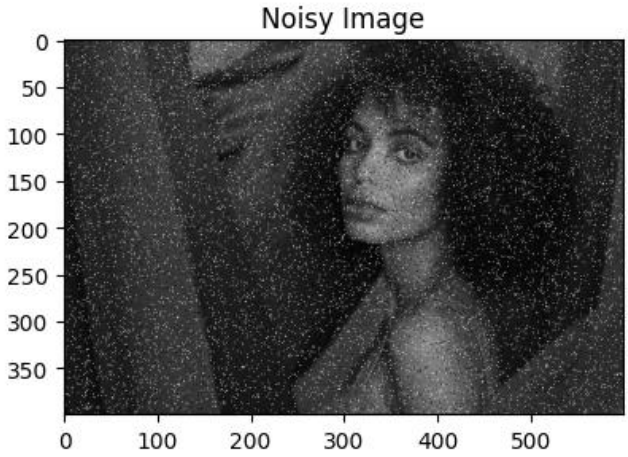
CO543: Image Processing
Lab 3

Ranage R.D.P.R. - E/19/310

The following are the original images that was used to implement the functions in the lab tasks.



The following functions are used throughout the lab to plot each figure to show the results.

```
# Importing required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import convolve2d
from skimage.util import random_noise

def plot_histogram(image, title, main_title):
    # Convert image to grayscale if it is not already
    if len(image.shape) == 3 and image.shape[2] == 3:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Convert image to uint8 if it is not already
    if image.dtype != np.uint8:
        image = cv2.normalize(image, None, 0, 255, cv2.NORM_MINMAX)
        image = image.astype(np.uint8)

    # Calculate histogram
    hist = cv2.calcHist([image], [0], None, [256], [0, 256])

    # Plot histogram
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.suptitle(main_title)
```

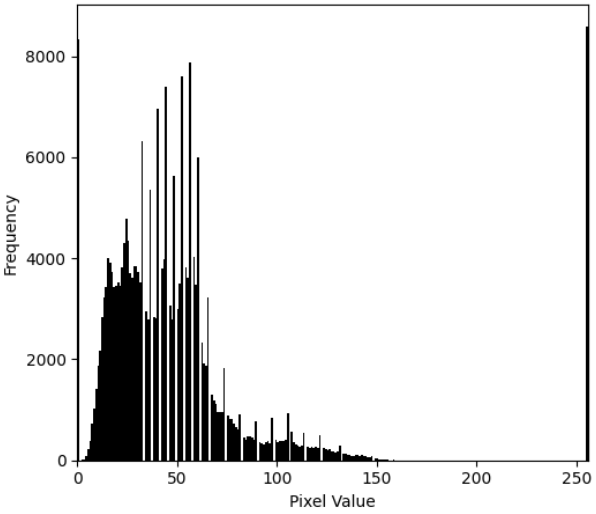
```
    plt.imshow(image, cmap='gray')
    plt.title("Image")
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.hist(image.ravel(), 256, [0, 256], color='black')
    plt.title(title)
    plt.xlabel('Pixel Value')
    plt.ylabel('Frequency')
    plt.xlim([0, 256])
    plt.tight_layout()
    plt.show()
```

Apply Denoising Filters

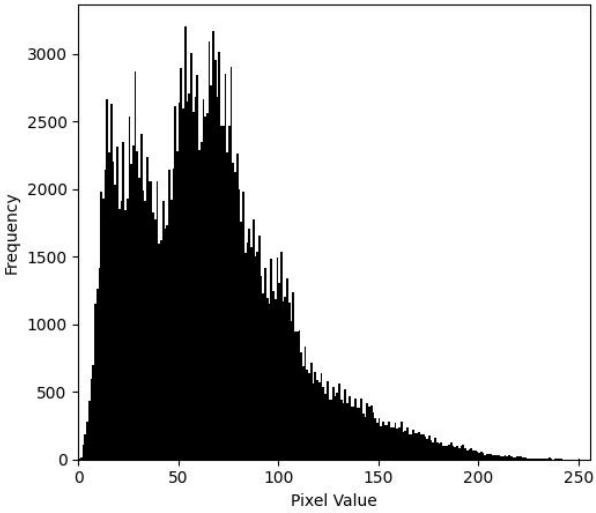
1. Apply Mean filtering with mask size 3x3 and 5x5

```
def mean_filter(image, mask_size):
    pad_size = mask_size // 2
    padded_image = np.pad(image, pad_size, mode='constant', constant_values=0)

    # Initialize the output image
    filtered_image = np.zeros_like(image)

    # Get the dimensions of the image
    rows, cols = image.shape

    # Apply the mean filter
    for i in range(rows):
        for j in range(cols):
            # Extract the region of interest
            roi = padded_image[i:i+mask_size, j:j+mask_size]
            # Compute the mean value of the region
            mean_value = np.mean(roi)
            # Set the mean value to the corresponding pixel in the output image
            filtered_image[i, j] = mean_value

    return filtered_image

mean_3x3 = mean_filter(gray_img,3)
mean_5x5 = mean_filter(gray_img,5)
plot_histogram(gray_img,"","Original Image")
plot_histogram(mean_3x3,"","Filtered 3x3 Image")
plot_histogram(mean_5x5,"","Filtered 5x5 Image")
```

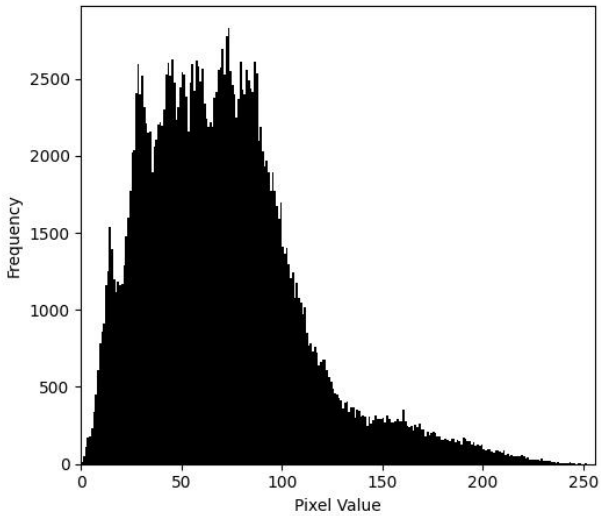
Original Image



Filtered 3x3 Image



Filtered 5x5 Image

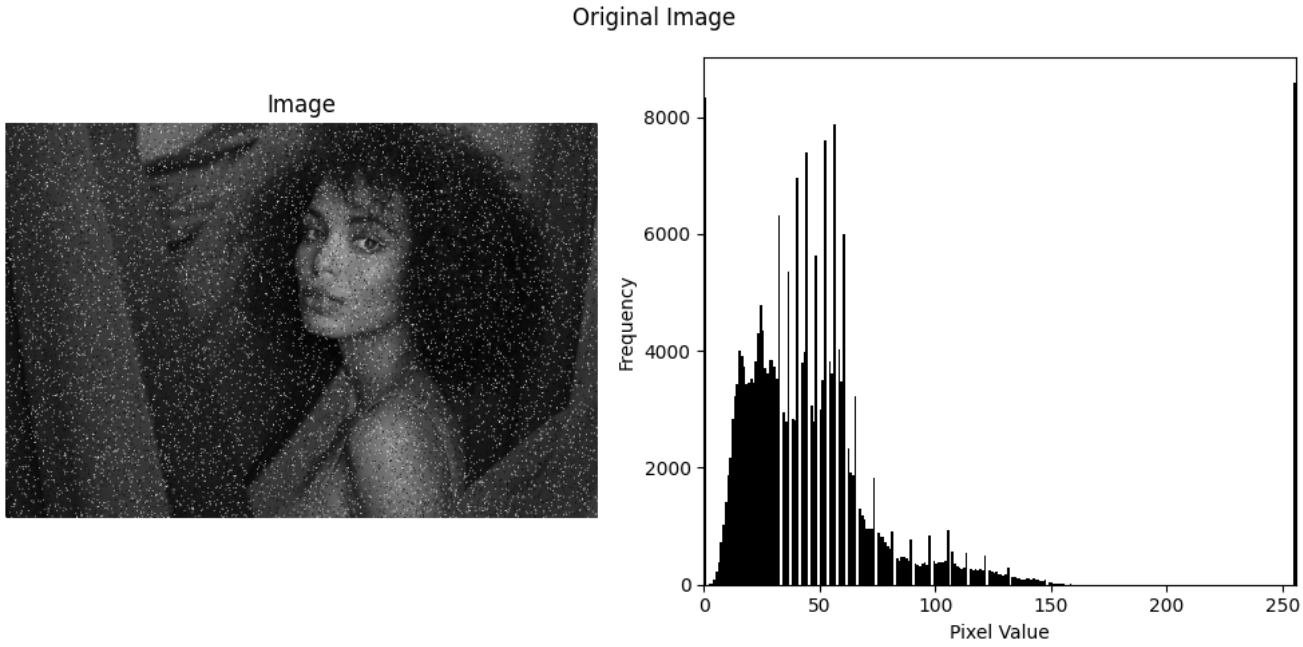


When applied with mask sizes of 3x3 and 5x5, the filtering process produces two distinct sets of results. With a 3x3 mask size, the filtering is relatively mild, resulting in a reduction of noise while preserving finer details and edges in the image. However, with a 5x5 mask size, the filtering becomes more aggressive, leading to a more pronounced smoothing effect and potentially causing a loss of finer details. Comparing the original image with the filtered images, it's evident that the latter exhibit smoother textures and reduced noise.

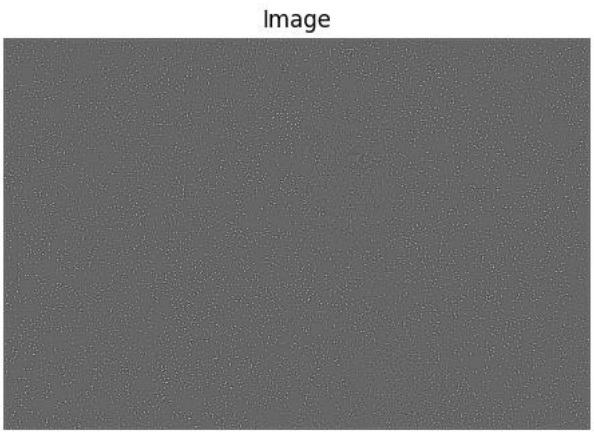
Analyzing the histograms of the original and filtered images further underscores the impact of mean filtering. In the original image's histogram, distinct peaks and valleys represent various intensity levels present in the image. However, after mean filtering, the histogram of the filtered image becomes more concentrated around the mean intensity value. Peaks are often smoothed out, and valleys may be filled in, resulting in a histogram that is more uniform or bell-shaped.

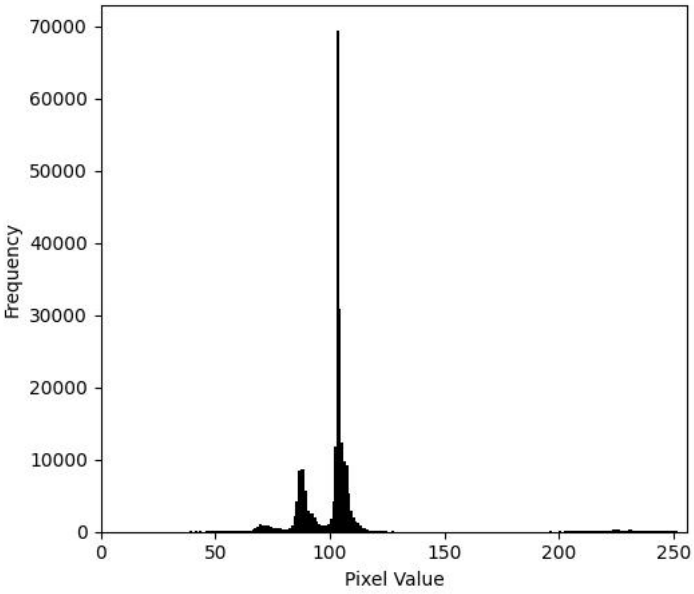
2. Apply Highpass filtering with mask size 3x3 and 5x5

```
def high_pass_filter(image, kernel_size):
    kernel = -np.ones((kernel_size, kernel_size), dtype=float)
    kernel[kernel_size // 2, kernel_size // 2] = kernel_size * kernel_size - 1

    return convolve2d(image, kernel)

hp_3x3 = high_pass_filter(gray_img,3)
hp_5x5 = high_pass_filter(gray_img,5)
plot_histogram(gray_img,"","Original Image")
plot_histogram(hp_3x3,"","Filtered 3x3 Image")
plot_histogram(hp_5x5,"","Filtered 5x5 Image")
```

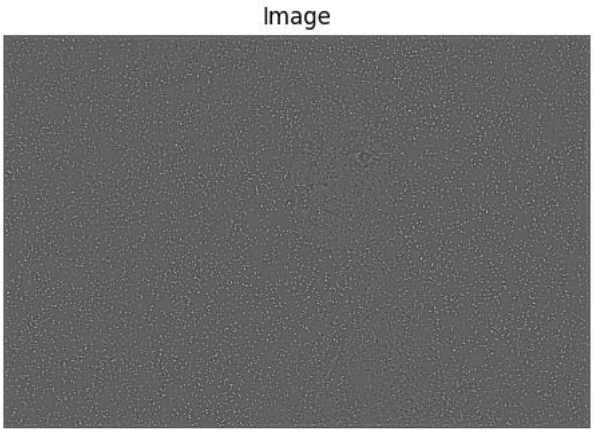


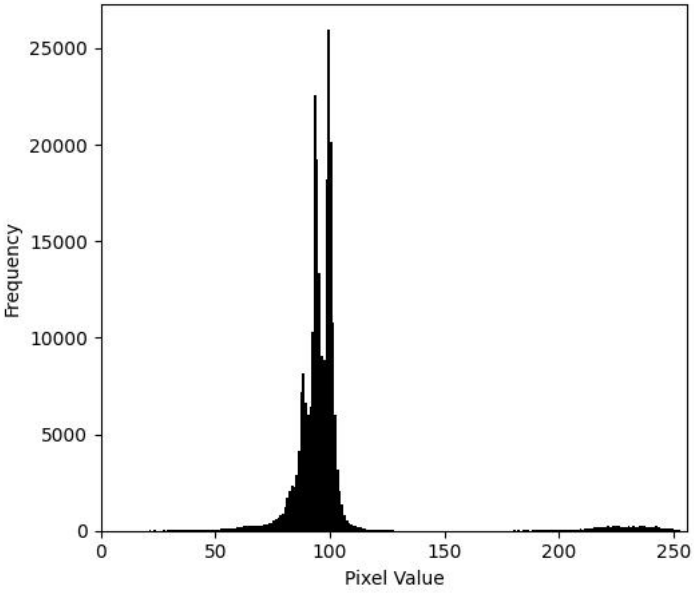
Filtered 3x3 Image





Filtered 5x5 Image



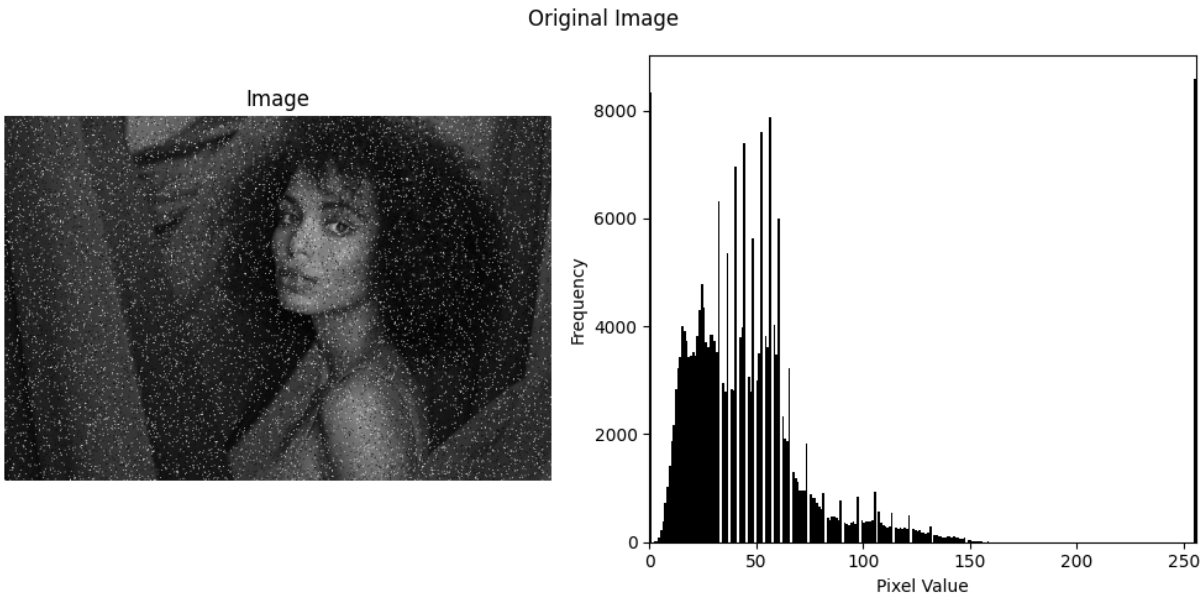


Applying high pass filtering with mask sizes of 3x3 and 5x5 to an image produces distinct sets of results. With a 3x3 mask size, the filtering emphasizes edges and high-frequency details, resulting in sharper edges and enhanced fine features while maintaining a relatively balanced representation of low-frequency components. Conversely, a 5x5 mask size amplifies this effect, further emphasizing edges and fine details while potentially suppressing low-frequency components to a greater extent. Comparing the original image with the filtered images reveals these differences vividly, with the filtered images exhibiting sharper edges and more pronounced high-frequency details.

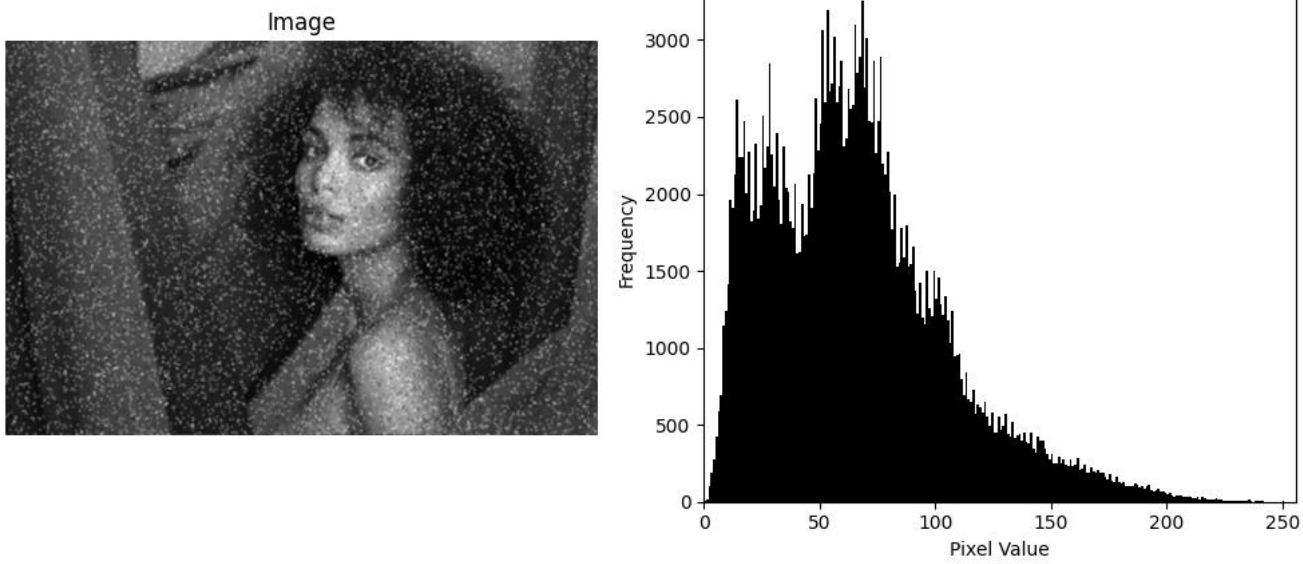
Analyzing the histograms of the original and filtered images further illustrates the impact of highpass filtering. In the original image's histogram, a distribution of pixel intensities reflects the various features present, while after highpass filtering, the histogram of the filtered image may display an increase in intensity values corresponding to edges and high-frequency components. Peaks representing these features become more prominent, indicating their enhancement through the filtering process. Meanwhile, intensity values corresponding to low-frequency components may be subdued, leading to a reduction in their representation in the histogram.

3. Apply lowpass filtering with mask size 3x3 and 5x5

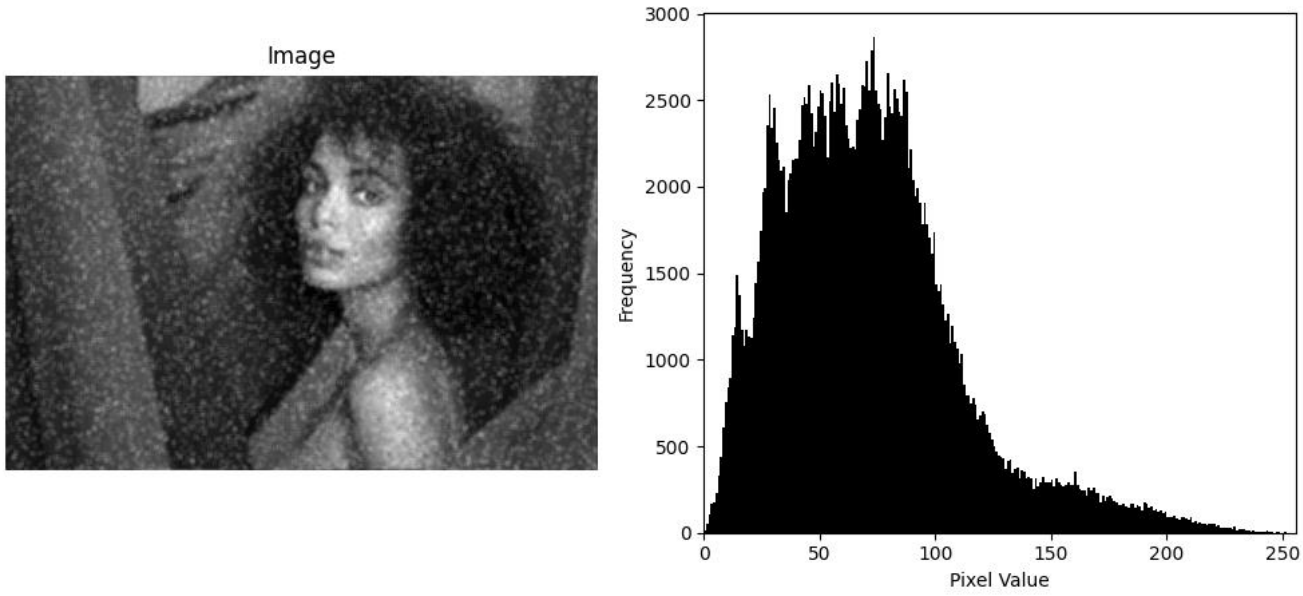
```
def low_pass_filter(image, kernel_size):
    # Create an averaging filter kernel
    kernel = np.ones((kernel_size, kernel_size), dtype=float) / (kernel_size *
kernel_size)

    # Apply the convolution
    filtered_image = convolve2d(image, kernel, mode='same', boundary='wrap')

    # Normalize the filtered image
    filtered_image = (filtered_image - np.min(filtered_image)) /
(np.max(filtered_image) - np.min(filtered_image)) * 255

    return filtered_image.astype(np.uint8)

lp_3x3 = low_pass_filter(gray_img,3)
lp_5x5 = low_pass_filter(gray_img,5)
plot_histogram(gray_img,"","Original Image")
plot_histogram(lp_3x3,"","Filtered 3x3 Image")
plot_histogram(lp_5x5,"","Filtered 5x5 Image")
```



Filtered 3x3 Image



Filtered 5x5 Image



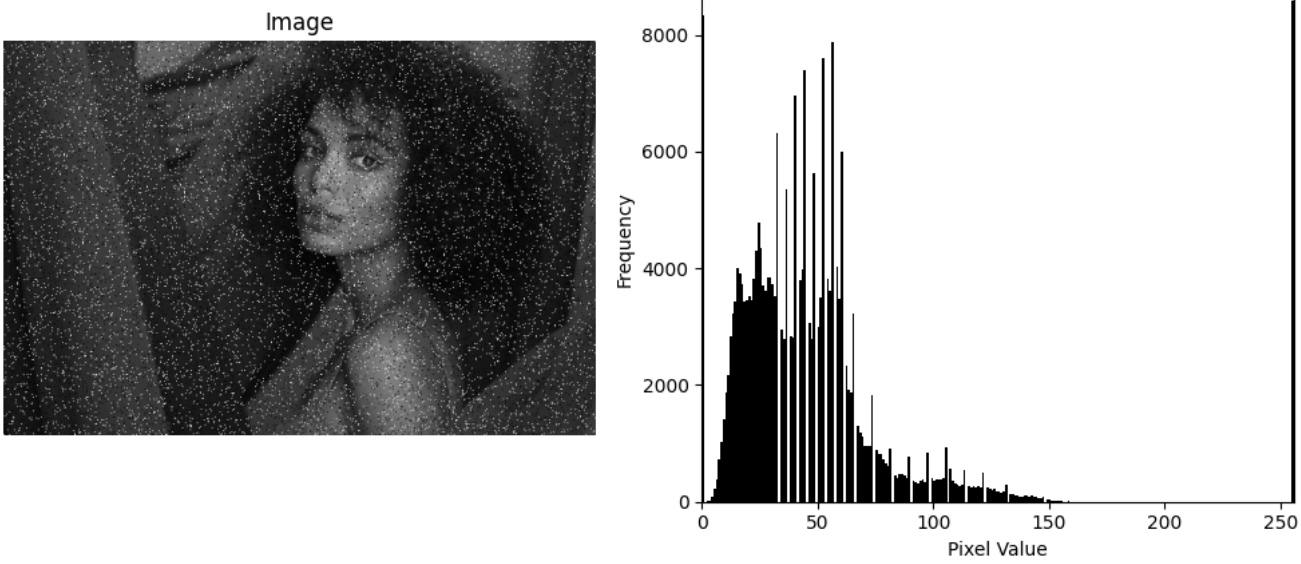
When lowpass filtering is applied with mask sizes of 3x3 and 5x5, distinct outcomes are produced. The 3x3 mask provides a moderate level of smoothing, subtly blurring the image to reduce noise and fine details without significantly affecting the overall structure of the image. The 5x5 mask, being larger, offers a more intense smoothing effect, leading to a more pronounced blurring of details and a greater reduction in noise. This can sometimes result in a loss of important image details.

The histograms of the original and filtered images also reveal insightful changes. The histogram of the original image typically displays a broad range of intensity values with sharp peaks, indicating the presence of detailed features and varying levels of brightness. In contrast, the histograms of the filtered images tend to flatten and broaden, indicating a reduction in the contrast of pixel intensities. This shift is more marked with the 5x5 mask, where the smoothing effect is stronger, leading to a histogram that shows fewer peaks and less distinction between different intensities.

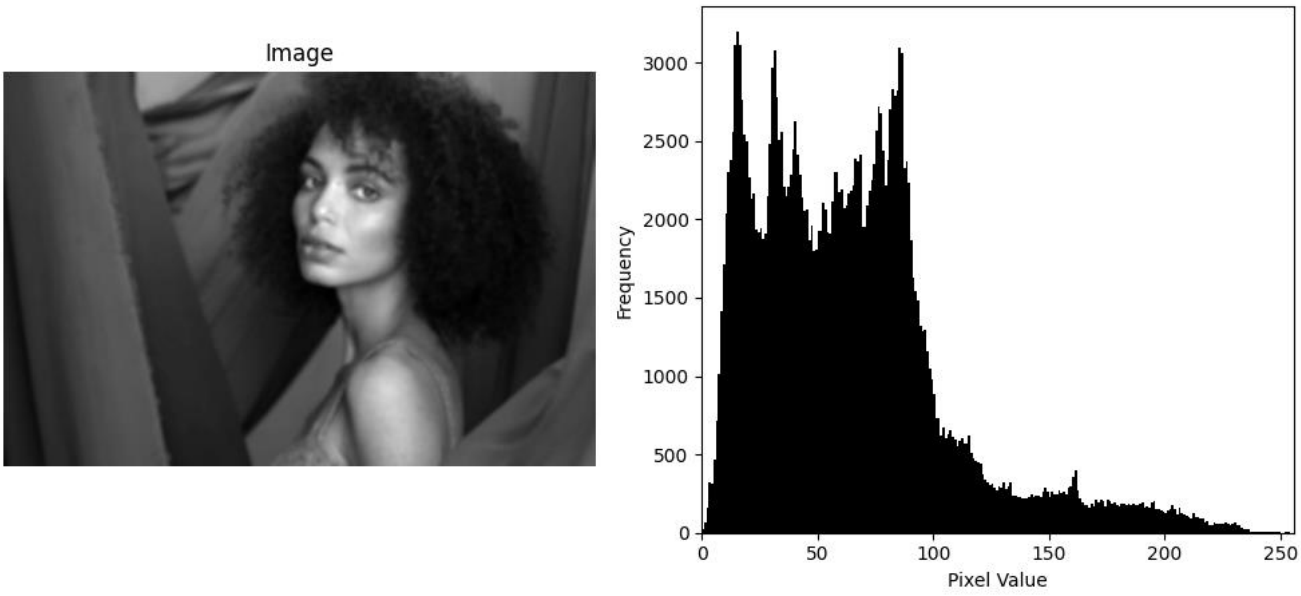
4. A bilateral filter with mask size 5 × 5 with appropriate values of σ and , set 2 d σr 2 through experimentation.

```
from skimage import data, color, img_as_float
from scipy.ndimage import gaussian_filter

def bilateral_filter(image, d, sigma_color, sigma_space):
    if len(image.shape) == 3:
        image = color.rgb2gray(image)

    # Padding the image to handle edges
    padded_image = np.pad(image, d // 2, mode='reflect')
    filtered_image = np.zeros_like(image)

    # Create Gaussian distance kernel
    ax = np.arange(-d // 2 + 1., d // 2 + 1.)
    xx, yy = np.meshgrid(ax, ax)
    gaussian_spatial = np.exp(-(xx**2 + yy**2) / (2. * sigma_space**2))

    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            region = padded_image[i:i + d, j:j + d]
            gaussian_intensity = np.exp(-(region - image[i, j])**2 / (2. *
sigma_color**2))
            weights = gaussian_spatial * gaussian_intensity
            filtered_image[i, j] = np.sum(weights * region) / np.sum(weights)

    return np.clip(filtered_image, 0, 1)

filtered_image = bilateral_filter(img, 5, 75, 75)
plot_histogram(gray_img,"","Original Image")
plot_histogram(filtered_image,"","Bilateral Filtered Image")
```

Original Image



Bilateral Filtered Image

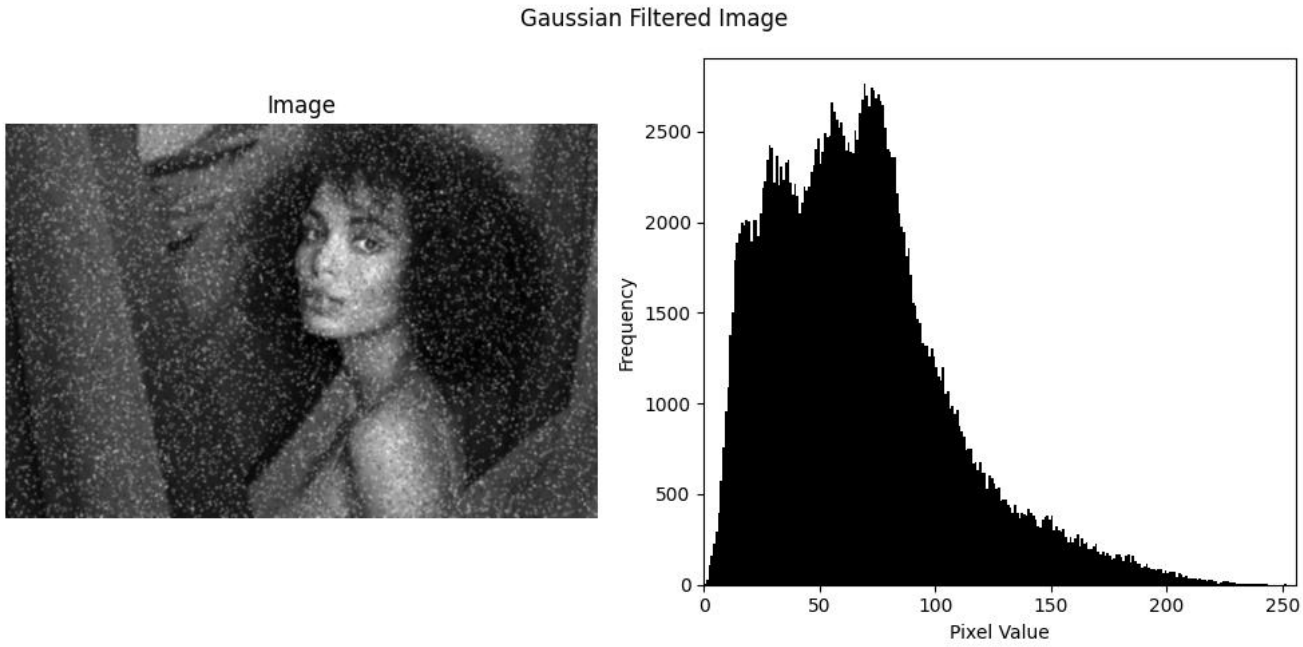


Applying a bilateral filter with a mask size of 5x5 and optimally tuned parameters, σ_d for spatial distance and σ_r for intensity differences, allows for a nuanced approach to smoothing. This filter is unique in that it considers both the geometric closeness of pixels and their intensity similarity, which helps in maintaining edge sharpness while smoothing areas with similar intensities. Visually, the filtered image maintains the edge integrity better than typical lowpass filters while smoothing noise in less detailed areas.

The histograms of the original and filtered images also exhibit notable differences. The original image's histogram typically shows a wider range of intensity values with distinct peaks and troughs corresponding to the various textures and edges within the image.

5. A Gaussian filter with mask size 5 × 5 appropriate values of σ.

```
def gaussian_filter(image, kernel_size, sigma):
    def create_gaussian_kernel(size, sigma):
        """Creates a Gaussian kernel."""
        ax = np.arange(-size // 2 + 1., size // 2 + 1.)
        xx, yy = np.meshgrid(ax, ax)
        kernel = np.exp(-0.5 * (np.square(xx) + np.square(yy)) / np.square(sigma))
        return kernel / np.sum(kernel)

    image = img_as_float(image)

    kernel = create_gaussian_kernel(kernel_size, sigma)

    pad_width = kernel_size // 2
    padded_image = np.pad(image, pad_width, mode='reflect')

    filtered_image = np.zeros_like(image)
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            region = padded_image[i:i+kernel_size, j:j+kernel_size]
            filtered_image[i, j] = np.sum(region * kernel)

    return filtered_image

filtered_image = gaussian_filter(gray_img, 5, 1.0)
plot_histogram(gray_img,"","Original Image")
plot_histogram(filtered_image,"","Gaussian Filtered Image")
```



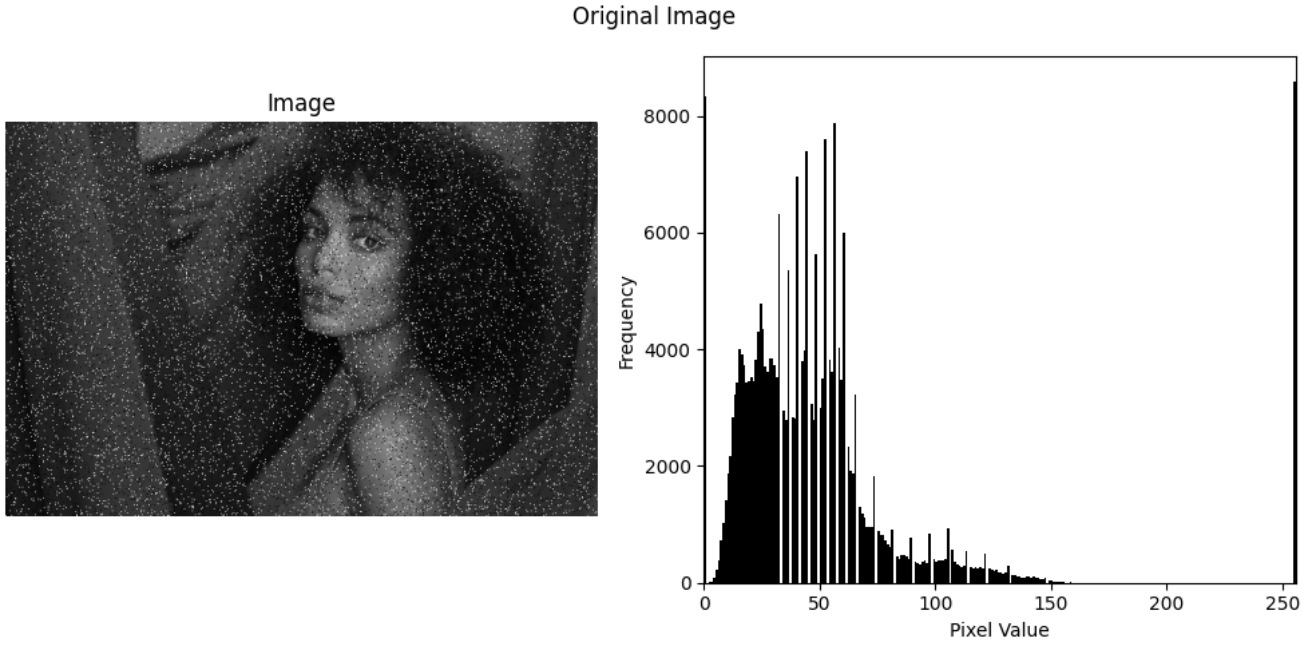
Employing a Gaussian filter with a 5x5 mask and an appropriately chosen value for the standard deviation (σ) allows for a controlled smoothing effect that spreads pixel values in a manner weighted by their spatial closeness, prioritizing closer pixels more significantly. The filtered image visually appears smoother, with a reduction in noise and fine details. This effect is beneficial in scenarios where reducing image noise is crucial, albeit at the expense of some detail loss and slight edge blurring.

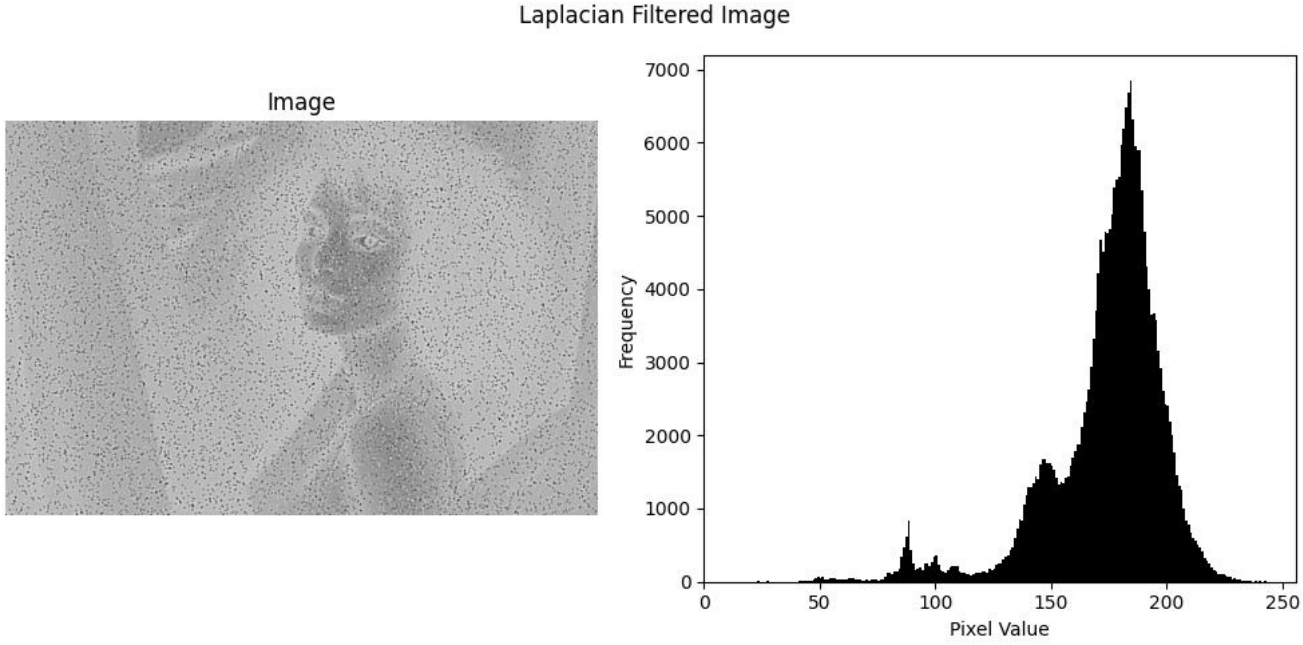
The original image's histogram typically features sharp peaks and varied intensity distributions reflecting the presence of both noise and detail. After the application of the Gaussian filter, the histogram of the filtered image generally displays a more homogenized distribution of intensities. Peaks in the histogram become less pronounced and broader, indicating a reduction in contrast and a blending of pixel values. This shift results in fewer abrupt changes in pixel intensity, which corresponds to the visual perception of a smoother and cleaner image.

6. A laplacian filter with mask size 5 × 5 appropriate values of σ.

```
def laplacian_filter(image, sigma):
    # Create a 5x5 Laplacian kernel with appropriate values of sigma
    size = 5
    ax = np.arange(-size // 2 + 1., size // 2 + 1.)
    xx, yy = np.meshgrid(ax, ax)
    kernel = ((xx**2 + yy**2 - 2 * sigma**2) / sigma**4) * np.exp(-(xx**2 + yy**2) /
(2 * sigma**2))

    # Convolve the image with the Laplacian kernel
    filtered_image = convolve2d(image, kernel, mode='same', boundary='symm')

    return filtered_image
```

```
filtered_image = laplacian_filter(gray_img, 1.1)
plot_histogram(gray_img,"","Original Image")
plot_histogram(filtered_image,"","Laplacian Filtered Image")
```





When applied with a 5x5 mask size and an appropriate value for σσ (if used in conjunction with Gaussian smoothing), the Laplacian filter sharply outlines edges and transitions while suppressing flat regions in the image. This filter is particularly useful for emphasizing boundaries and other structural details in an image. The visual contrast between the original and Laplacian-filtered images is quite striking. The filtered image appears significantly different, with edges and transitions

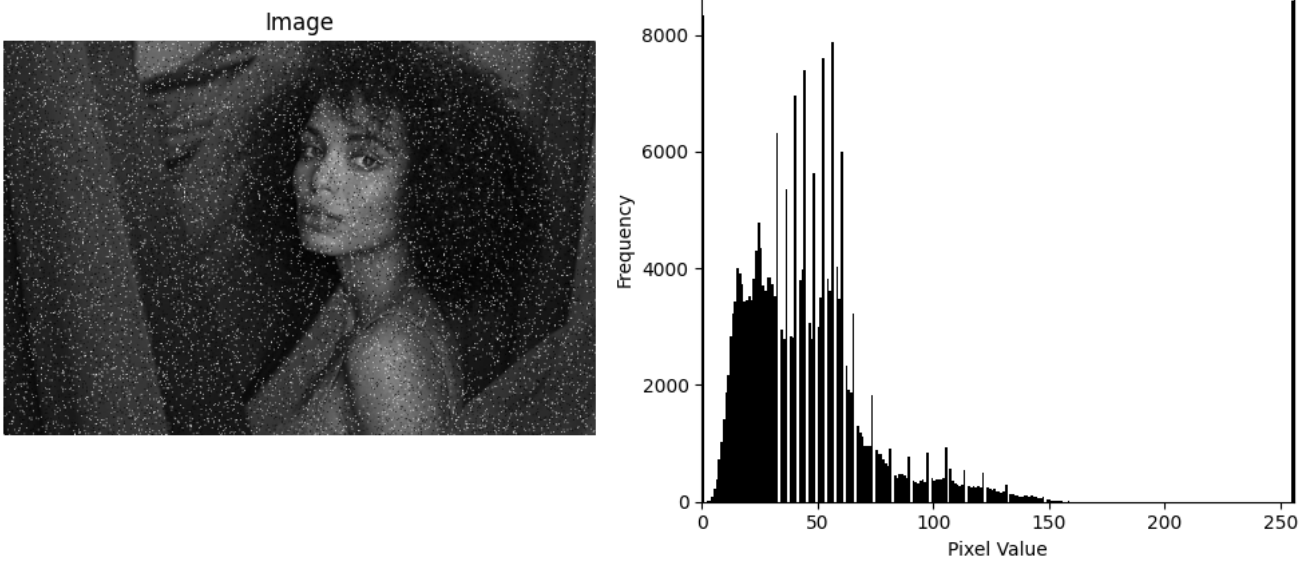
becoming very pronounced while the smoother areas become less noticeable. This enhancement of edges makes the filtered image appear crisper, but with potential introduction of artifacts or increased noise around the edges due to the nature of the filter emphasizing rapid changes in intensity.

The histograms of the original and filtered images also reflect these changes. The histogram of the original image typically shows a continuous distribution of pixel intensities with a range of peaks and valleys representing different features of the image. In contrast, the histogram of the Laplacian-filtered image often exhibits more extreme values with higher peaks at the spectrum ends, indicating an increase in contrast.

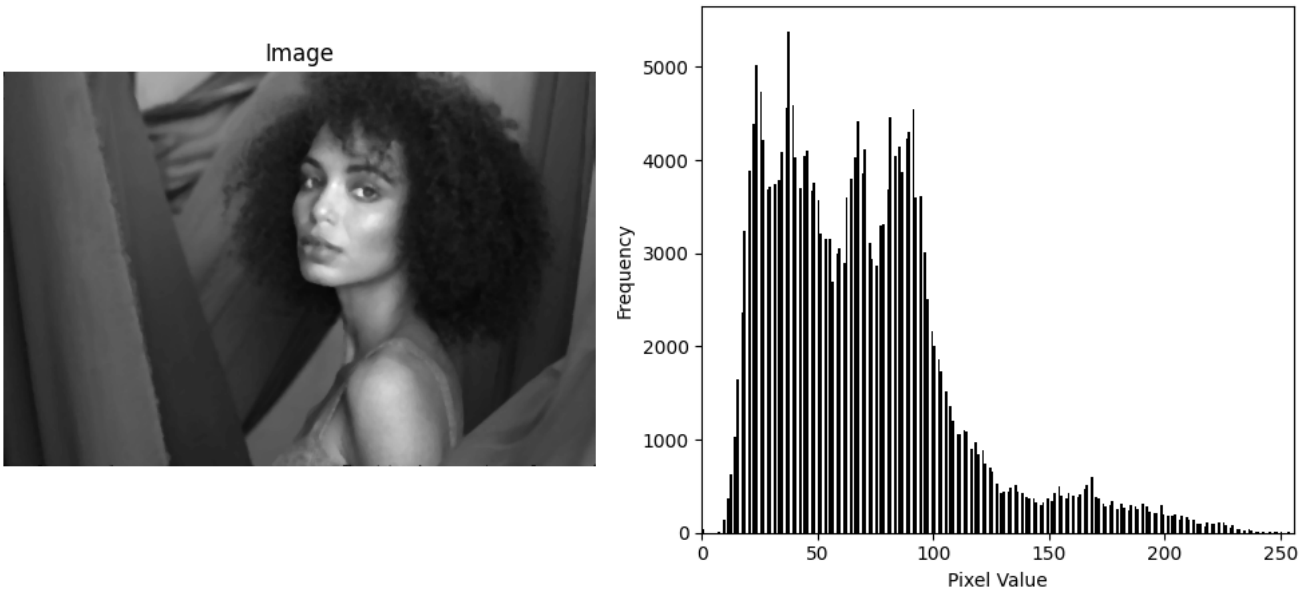
7. A median filter of appropriate window size

```
def median_filter(image, kernel_size):
    # Ensure kernel size is odd
    if kernel_size % 2 == 0:
        raise ValueError("Kernel size must be odd")

    # Calculate the padding size
    pad_size = kernel_size // 2

    # Pad the image
    padded_image = np.pad(image, pad_size, mode='constant', constant_values=0)

    # Create an empty array to store the filtered image
    filtered_image = np.zeros_like(image)

    # Iterate over the image
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            # Extract the neighborhood
            neighborhood = padded_image[i:i+kernel_size, j:j+kernel_size]

            # Compute the median of the neighborhood
            median_value = np.median(neighborhood)

            filtered_image[i, j] = median_value

    return filtered_image

filtered_image = median_filter(gray_img, 5)
plot_histogram(gray_img,"","Original Image")
plot_histogram(filtered_image,"","Median Filtered Image")
```

Original Image



Median Filtered Image



The median filter is a non-linear digital filtering technique often used in image processing to reduce "salt and pepper" noise without significantly blurring the images. This filter works by replacing each pixel value with the median value of the pixels within a specified window around it. For effective noise reduction, a typical window size might be 3x3 or 5x5, depending on the level of noise and the required preservation of details. Upon applying a median filter of an appropriate window size, notable changes can be observed between the original and filtered images. The filtered image generally appears smoother, especially in areas affected by noise, with the median filter effectively removing the outliers in the pixel values.

The histogram of the original image might display a broader range of intensity values with several spikes indicating the presence of noise or outliers. In contrast, the histogram of the filtered image typically appears smoother and more continuous without sharp spikes, reflecting the reduction in variance and outlier pixel values. The peaks in the filtered image's histogram tend to be less pronounced and more rounded, which signifies that extreme values have been moderated.