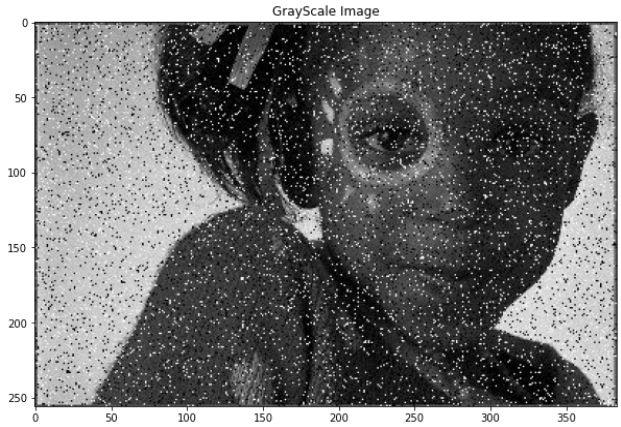
E/19/129
K.H. Gunawardana

CO543 - Image Processing
Lab 03

Input Image:



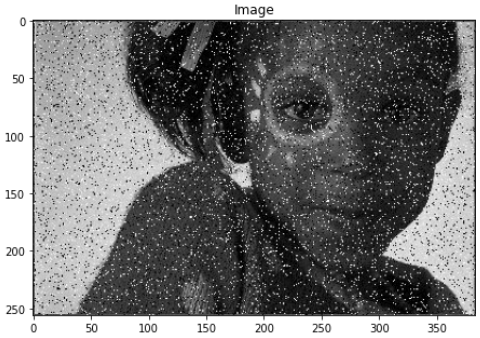
1. Apply Mean filtering with mask sizes 3x3 and 5x5

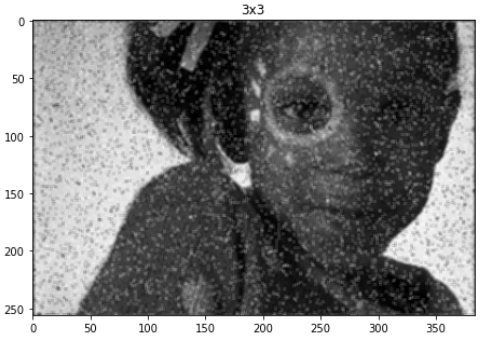
Code:

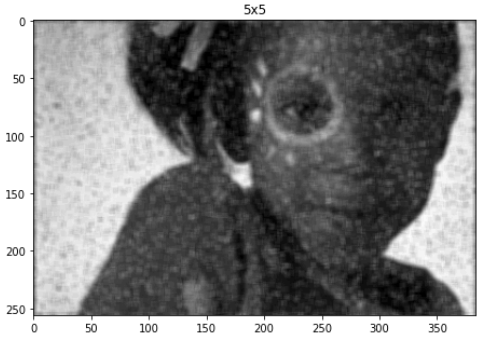
```python
def meanFilter(I, size):

    # Define a custom kernel (example: edge detection kernel)
    kernel = np.ones((size[0], size[1])) / (size[0] * size[1])

    # Apply the custom kernel to the image
    return cv2.filter2D(I, -1, kernel)
```

✓ 0.0s

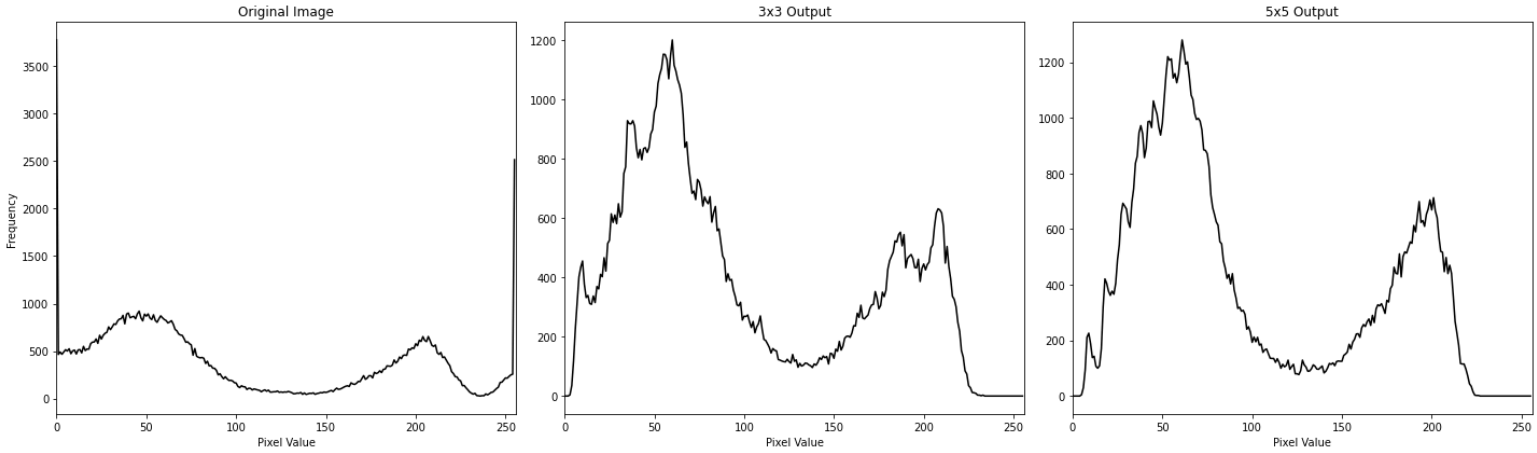
Outputs:







Histograms:



Comparison:

Using a 3x3 mask reduces noise moderately and keeps some details. With a 5x5 mask, it reduces noise but also blurs the edges and fine details more. The histogram of the filtered images shows a tighter grouping of pixel values, indicating less noise, but the images look more blurred compared to the original.

The mean filter replaces each pixel value with the average intensity of its neighbours. It is effective in reducing Gaussian noise but tends to blur edges, especially with larger kernel sizes like 5x5. This blurring effect is a trade-off for smoother noise reduction.

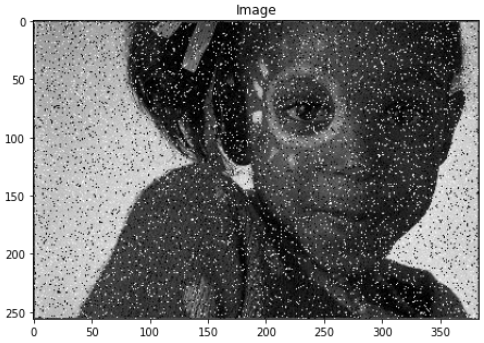
2. Apply Highpass filtering with mask sizes 3x3 and 5x5

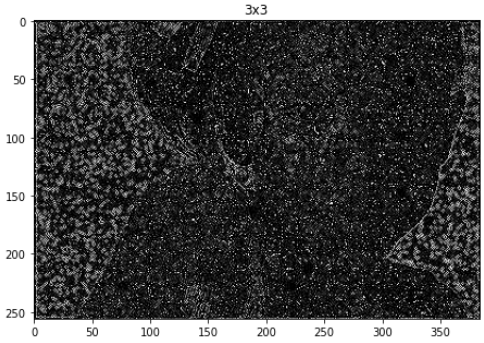
Code:

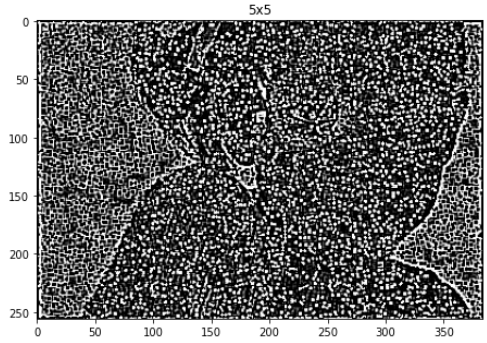
```python
def highPassFilter(I, size):

    # kernal for 3 and 5
    if size == 3:
        kernel = np.array([[0, -1, 0], [-1, 4, -1], [0, -1, 0]])
    elif size == 5:
        kernel = np.array([[-1, -1, -1, -1, -1],
                           [-1,  1,  2,  1, -1],
                           [-1,  2,  4,  2, -1],
                           [-1,  1,  2,  1, -1],
                           [-1, -1, -1, -1, -1]])

    # Apply the custom kernel to the image
    return cv2.filter2D(I, -1, kernel)
```

✓ 0.0s

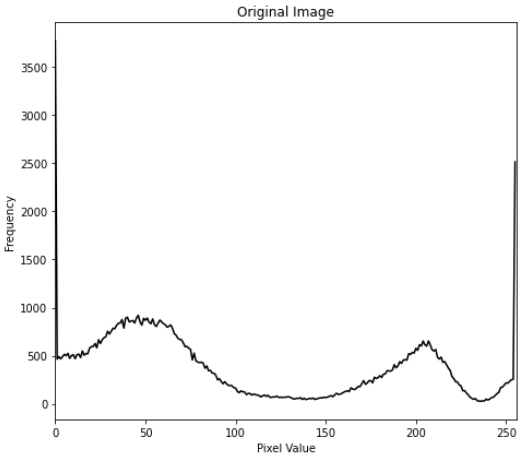
Outputs:

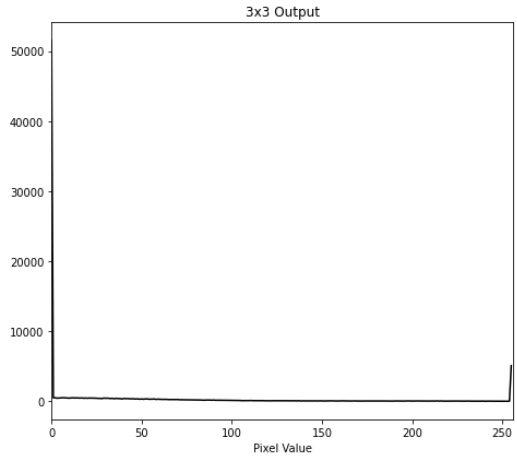


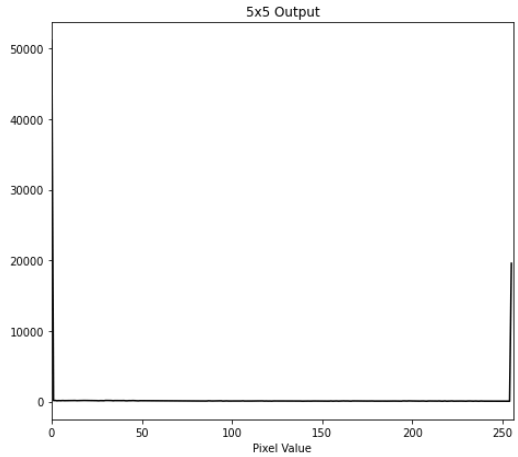




Histograms:







Comparison:

With 3x3 and 5x5 kernels, the image create more and more high frequency. Thus, the output images are more noisy and not very clear. With a 3x3 mask, it sharpens the image and makes the edges clearer. Using a 5x5 mask increases this effect, making edges even sharper but also increasing noise. The histogram shows a wider range of pixel values, reflecting the higher contrast and more pronounced details.

In general, The high pass filter emphasizes high-frequency components, such as edges and fine details. While it can sharpen the image, it also amplifies noise, making it less suitable for primary denoising. The 5x5 kernel amplifies these effects more than the 3x3 kernel.

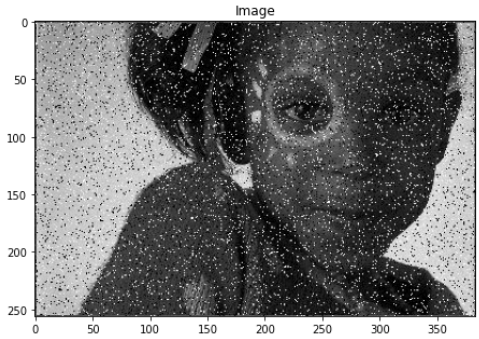
3. Apply lowpass filtering with mask sizes 3x3 and 5x5

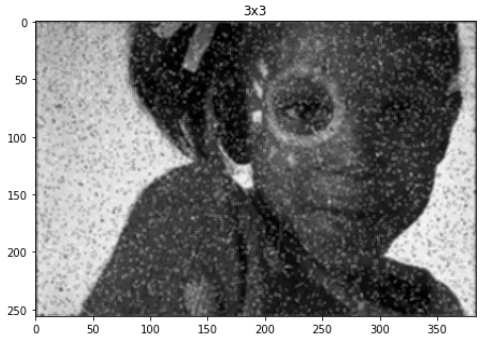
Code:

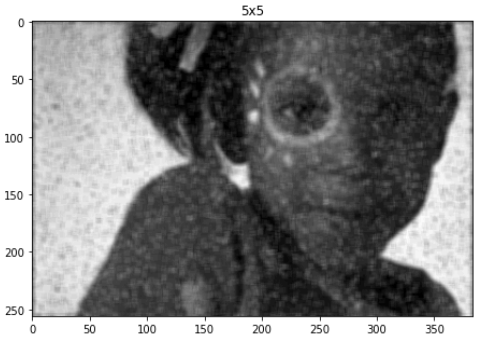
```python
def lowPassFilter(I, size):

    # Define a custom kernel (example: edge detection kernel)
    kernel = np.ones((size[0], size[1])) / (size[0] * size[1])

    # Apply the custom kernel to the image
    return cv2.filter2D(I, -1, kernel)
```

✓ 0.0s

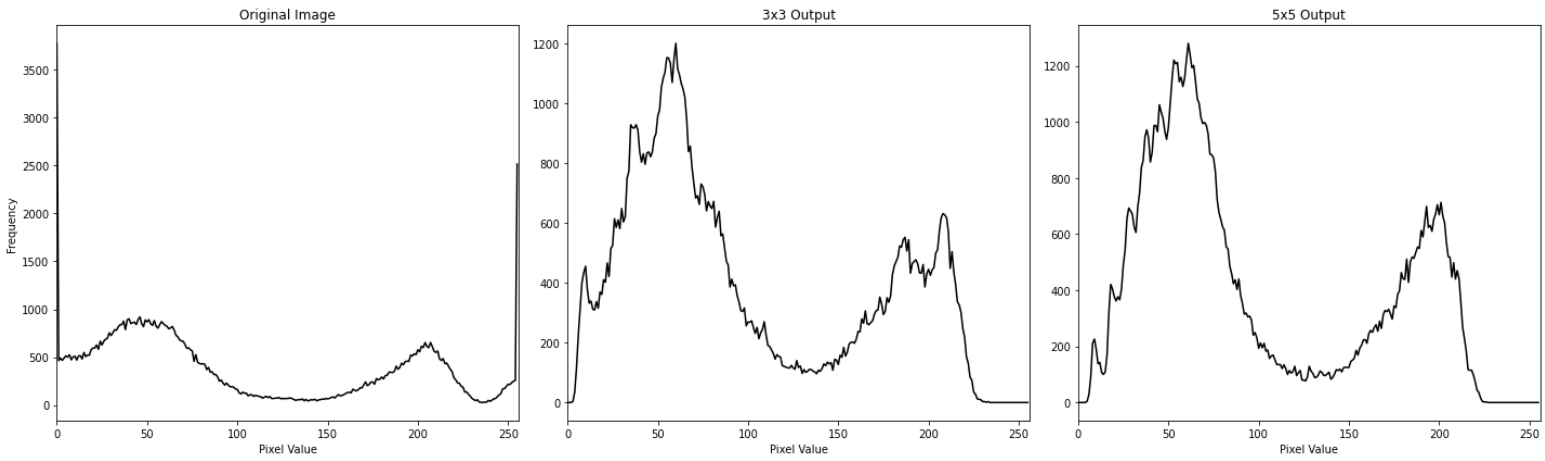
Outputs:







Histograms:



Comparison:

The 3x3 mask reduces noise while slightly blurring edges. The 5x5 mask reduces the noise but blurs the edges more. The histogram shows a reduction in the spread of pixel values, indicating smoother images but with less detail.

Thus, Low-pass filters are used to suppress high-frequency components, which include noise and fine details. They are effective in reducing noise but can also blur important edges in the image. The 5x5 kernel offers more significant noise reduction at the expense of more blurring compared to the 3x3 kernel.

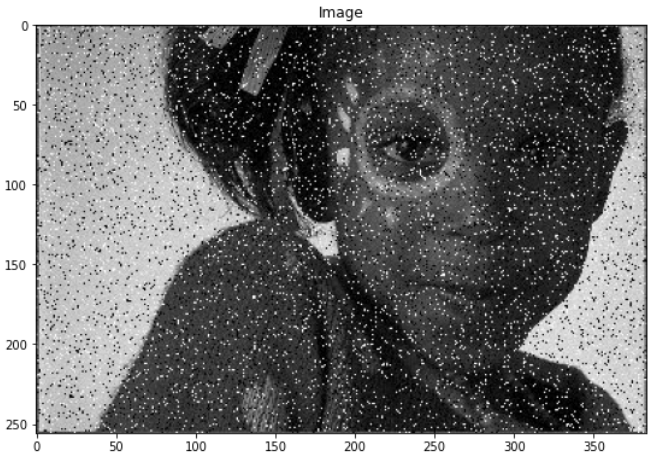
4. A bilateral filter with mask size 5 × 5 with appropriate values of σ and, set 2 d σr 2 through experimentation.

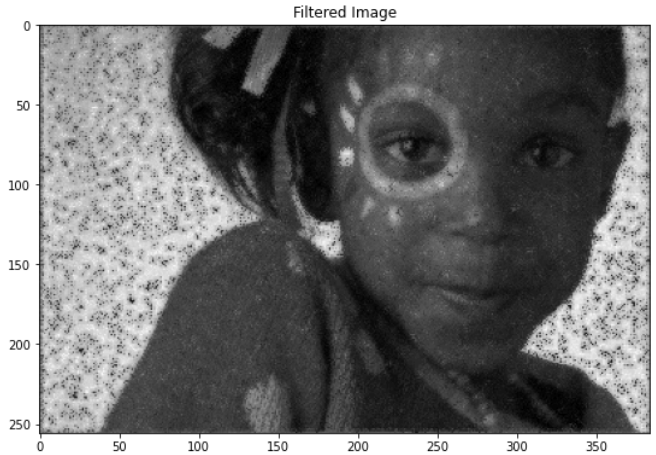
Code:

```python
def gaussian(x, sigma):
    return np.exp(- (x**2) / (2 * sigma**2))

def bilateralFilter(image, d, sigma_color, sigma_space):
    height, width = image.shape
    filtered_image = np.zeros_like(image, dtype=np.float32)

    half_size = d // 2

    for i in range(height):
        for j in range(width):
            wp = 0
            filtered_value = 0
            for k in range(-half_size, half_size + 1):
                for l in range(-half_size, half_size + 1):
                    ni = i + k
                    nj = j + l
                    if 0 <= ni < height and 0 <= nj < width:
                        gi = gaussian(np.sqrt(k**2 + l**2), sigma_space)
                        intensity_diff = image[ni, nj] - image[i, j]
                        gr = gaussian(intensity_diff, sigma_color)
                        w = gi * gr
                        filtered_value += w * image[ni, nj]
                        wp += w
            if wp != 0:
                filtered_image[i, j] = filtered_value / wp
            else:
                filtered_image[i, j] = image[i, j]

    return np.uint8(filtered_image)
```

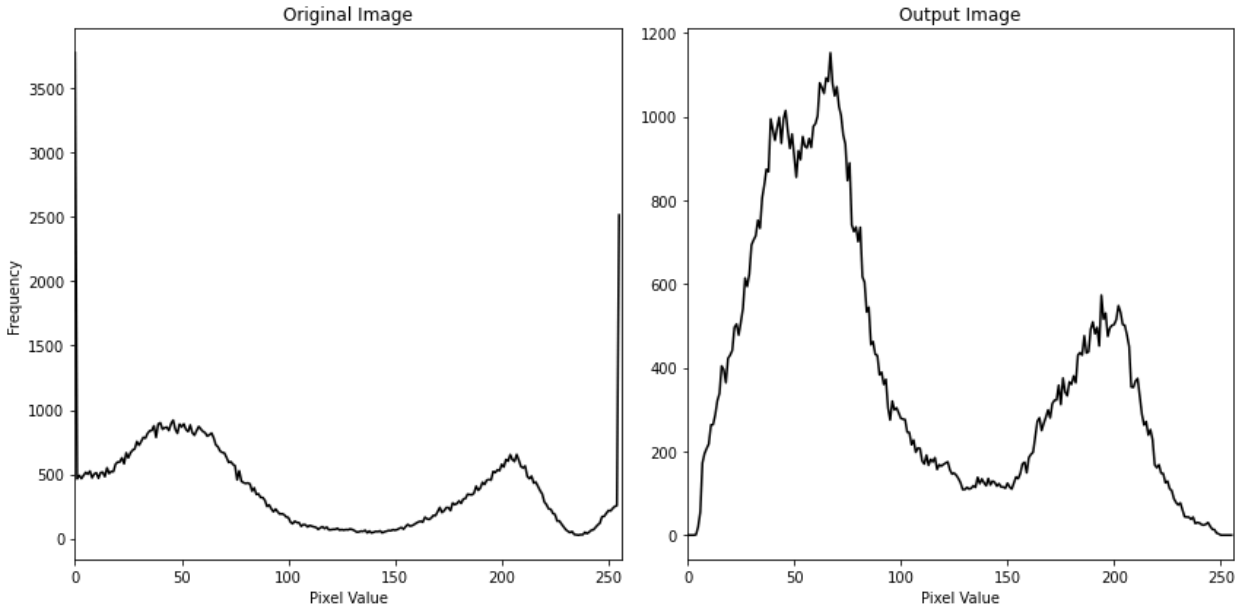
✓ 0.0s

Outputs:





Histograms:



Comparison:

The bilateral filter reduces noise while keeping edges sharp. It considers both the distance between pixels and the difference in their values. Experimenting with different values for spatial and range parameters helps find the best balance. The filtered image keeps more detail compared to mean and Gaussian filters and the edges remain clear. The histogram shows reduced noise without losing much detail. The distribution is smoother and distributed more generally.

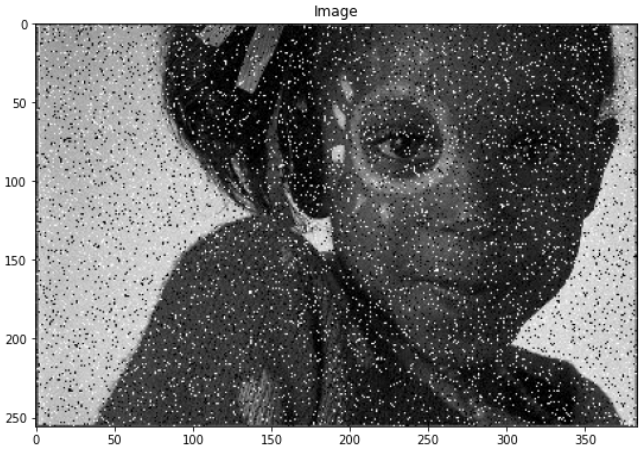
5. A Gaussian filter with mask size 5 × 5 appropriate values of σ.

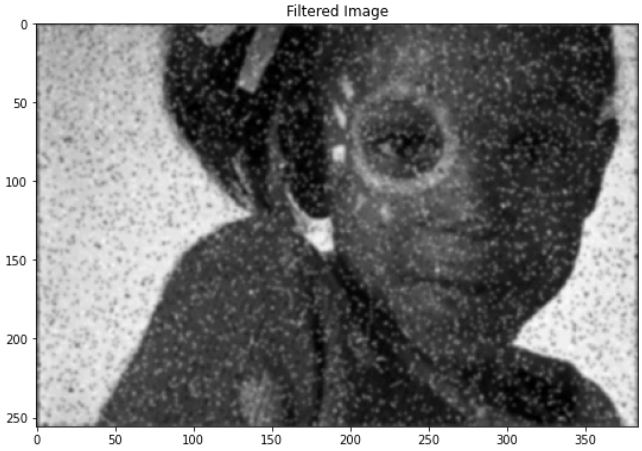
Code:

```python
def guassianFilter(I, size, sigma):

    kernel = np.fromfunction(
        lambda x, y: (1/(2*np.pi*sigma**2)) * np.exp(-((x-(size-1)/2)**2 + (y-(size-1)/2)**2) / (2*sigma**2)),
        (size, size))

    kernel /= np.sum(kernel)

    return cv2.filter2D(I, -1, kernel)
```

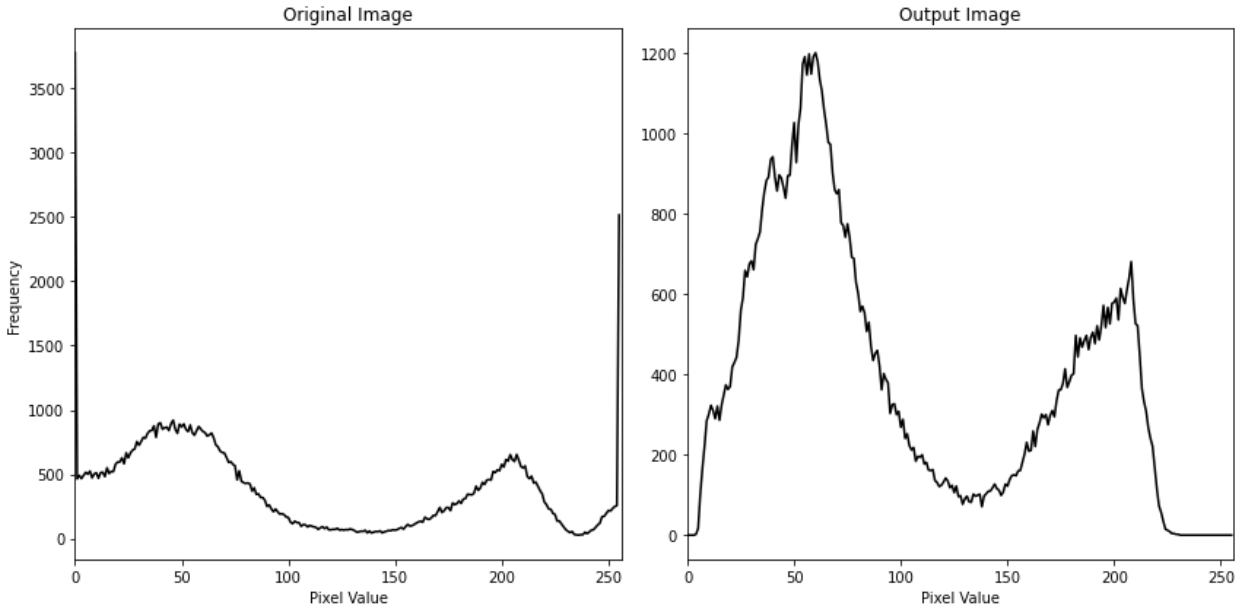
0.0s

Outputs:





Histograms:



Comparison:

The Gaussian filter smooths the image using a Gaussian function, which reduces noise without blurring edges as much as the mean filter. The histogram shows reduced noise variance, indicating smoother images with retained detail. The sigma value (σ) is important for controlling the balance between noise reduction and edge preservation. The distribution is smoother and distributed more generally.

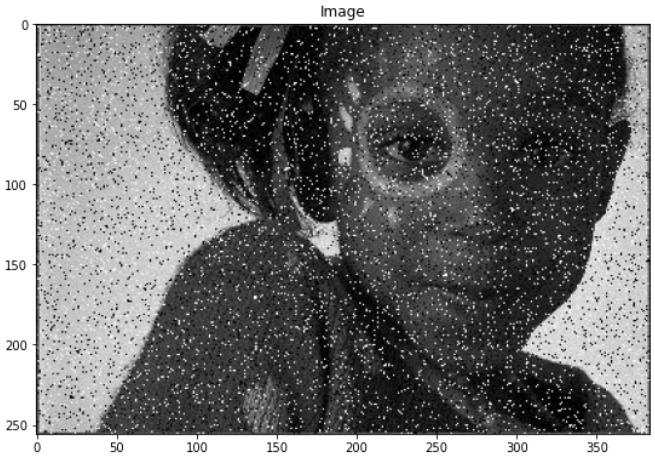
6. A laplacian filter with mask size 5 × 5 appropriate values of σ . 7. A median filter of appropriate window size.

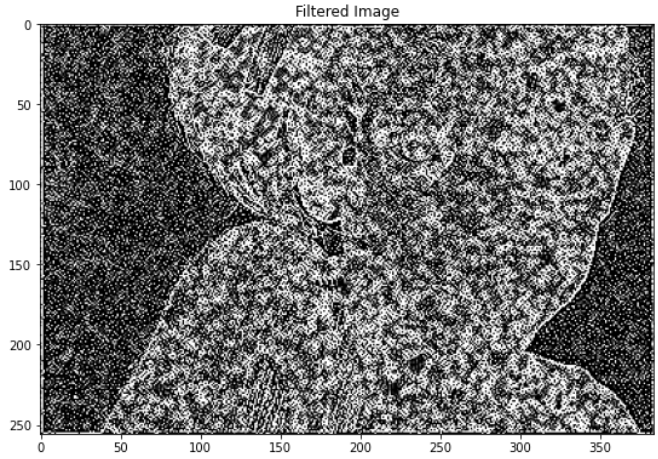
Code:

```python
def laplacianFilter(I, size):

    # Define a custom kernel (example: edge detection kernel)
    if size == 3:
        kernel = np.array([[0, 1, 0], [1, -4, 1], [0, 1, 0]])
    elif size == 5:
        kernel = np.array([[0, 0, 1, 0, 0],
                           [0, 1, 2, 1, 0],
                           [1, 2, -16, 2, 1],
                           [0, 1, 2, 1, 0],
                           [0, 0, 1, 0, 0]])

    # Apply the custom kernel to the image
    return cv2.filter2D(I, -1, kernel)
```

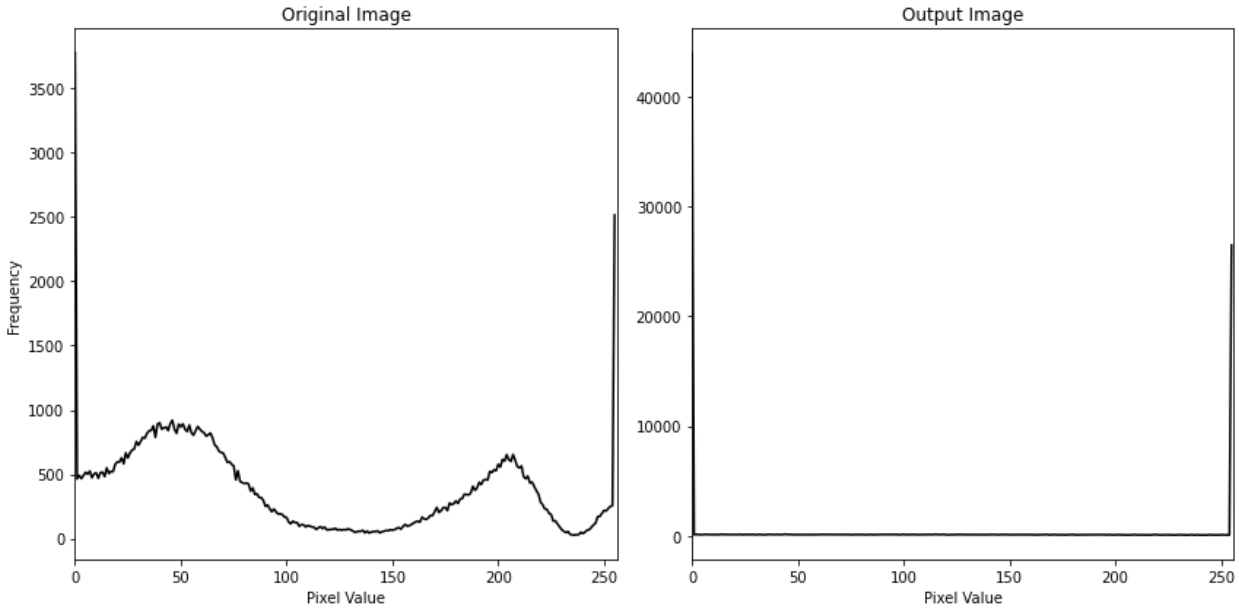
✓ 0.0s

Outputs:





Histograms:



Comparison:
The Laplacian filter highlights edges by detecting changes in pixel values. This makes edges more visible but also increases noise. The filtered image shows sharper edges, which are more noisy. The histogram shows an increase in high-frequency components, reflecting the emphasis on edges.

The Laplacian filter generally highlights edges by calculating the second derivative of the image intensity. While it can sharpen edges, it also amplifies noise edges, which requires careful interpretation. This filter is more suited for edge detection rather than primary denoising.

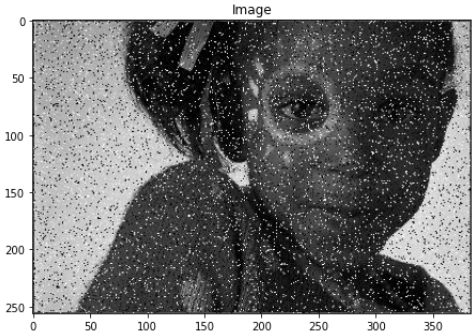
7. A median filter of appropriate window size.

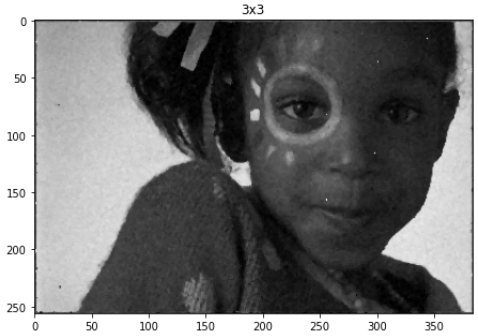
Code:

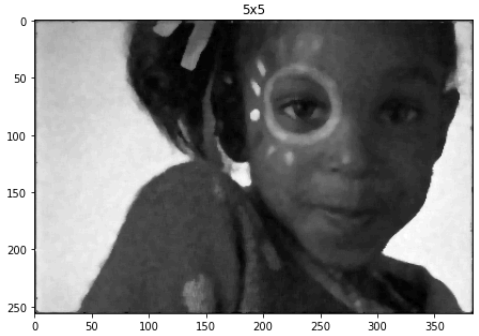
```python
def medianFilter(I, size):

    padded_image = np.pad(I, ((size//2, size//2), (size//2, size//2)), mode='constant')

    filtered_image = np.zeros_like(I)

    for i in range(I.shape[0]):
        for j in range(I.shape[1]):
            neighborhood = padded_image[i:i+size, j:j+size]
            filtered_image[i, j] = np.median(neighborhood)

    return filtered_image
```

✓ 0.0s

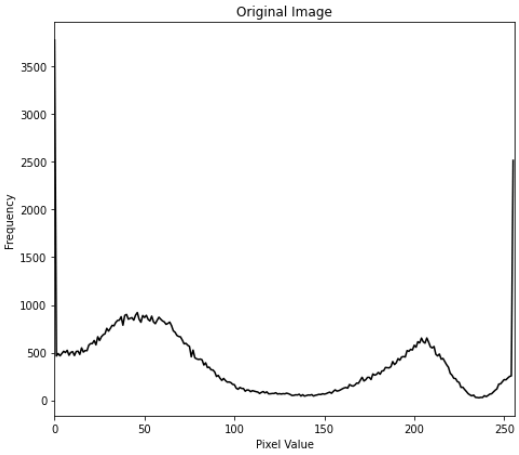
Outputs:

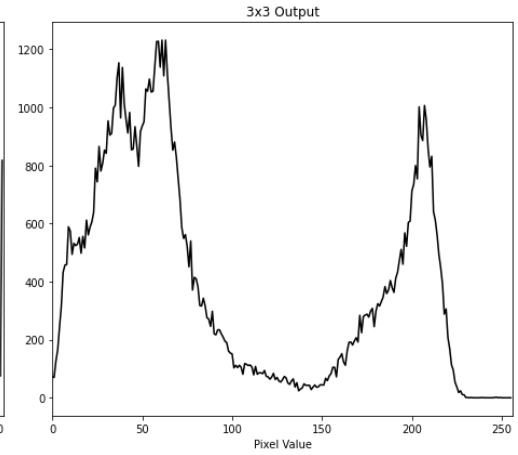


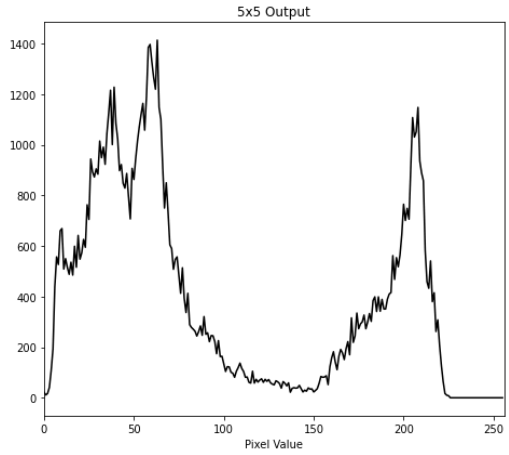




Histograms:







Comparison:

A 3x3 window reduces noise and preserves edges well. A 5x5 window reduces more noise but can blur small details. The histogram shows fewer noise peaks, resulting in smoother images while maintaining the overall structure.

The median filter replaces each pixel with the median intensity of its neighbours, effectively reducing impulsive noise like salt-and-pepper noise while preserving edges. The window size, usually odd (e.g., 3x3, 5x5), should be chosen based on the noise characteristics.