CO 543 : IMAGE PROCESSING

LAB 03: REPORT

E/19/167

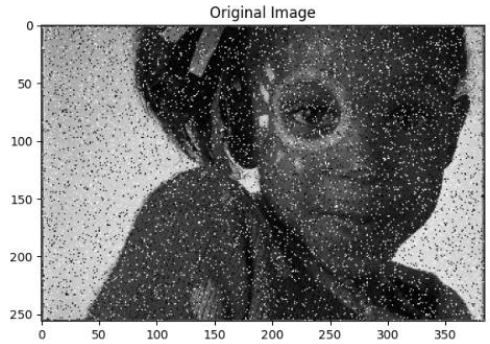
Jayawardena H.D.N.S.

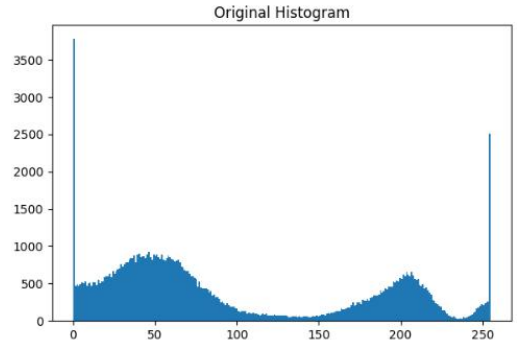
1. Mean Filter

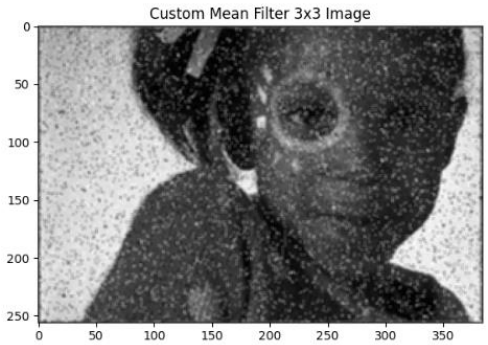
Code

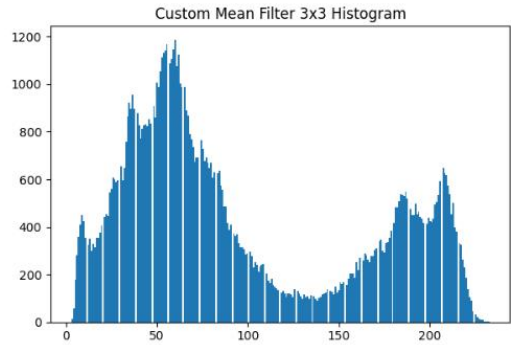
```
def mean_filter_custom(image, mask_size):
    kernel = np.ones((mask_size, mask_size)) / (mask_size * mask_size)
    return convolve(image, kernel)

def mean_filter_opencv(image, mask_size):
    return cv2.blur(image, (mask_size, mask_size))


custom_mean = mean_filter_custom(image, 3)
opencv_mean = mean_filter_opencv(image, 3)
display_images(image, custom_mean, opencv_mean, 'Mean Filter 3x3')

custom_mean = mean_filter_custom(image, 5)
opencv_mean = mean_filter_opencv(image, 5)
display_images(image, custom_mean, opencv_mean, 'Mean Filter 5x5')
```

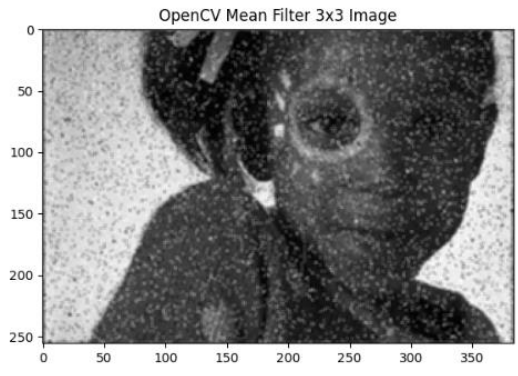
Output

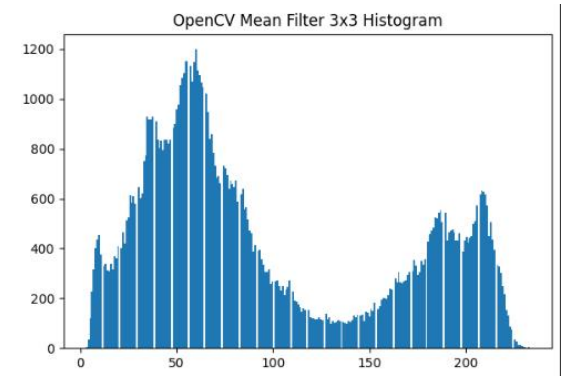


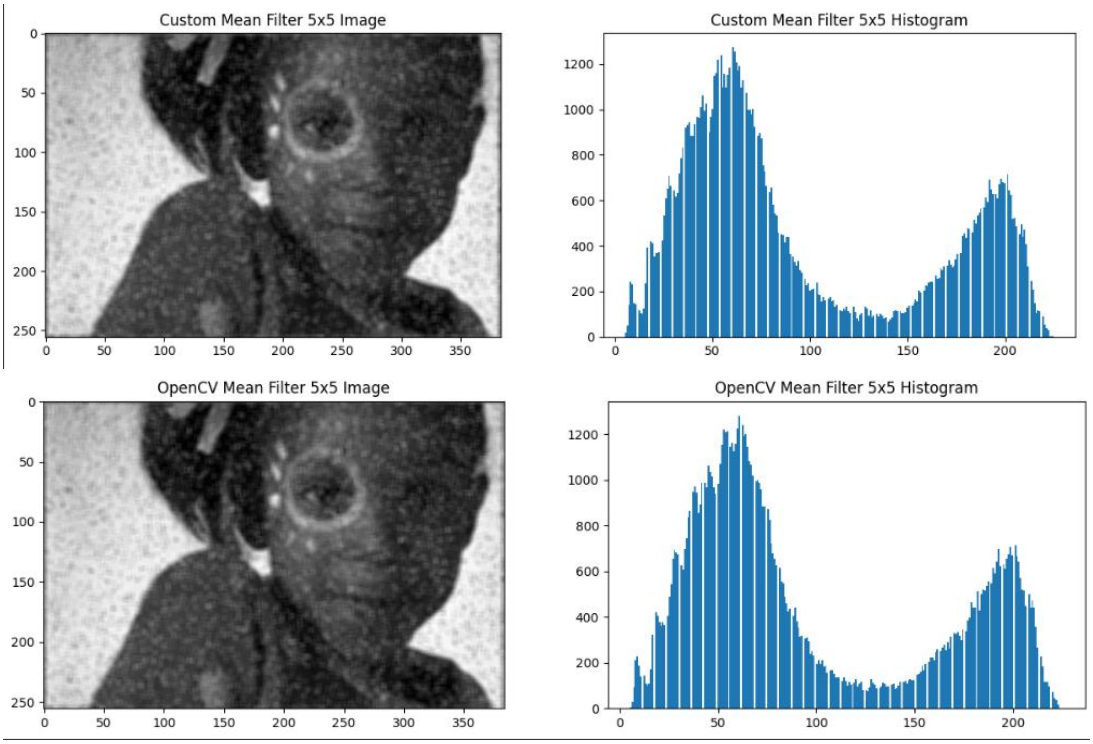










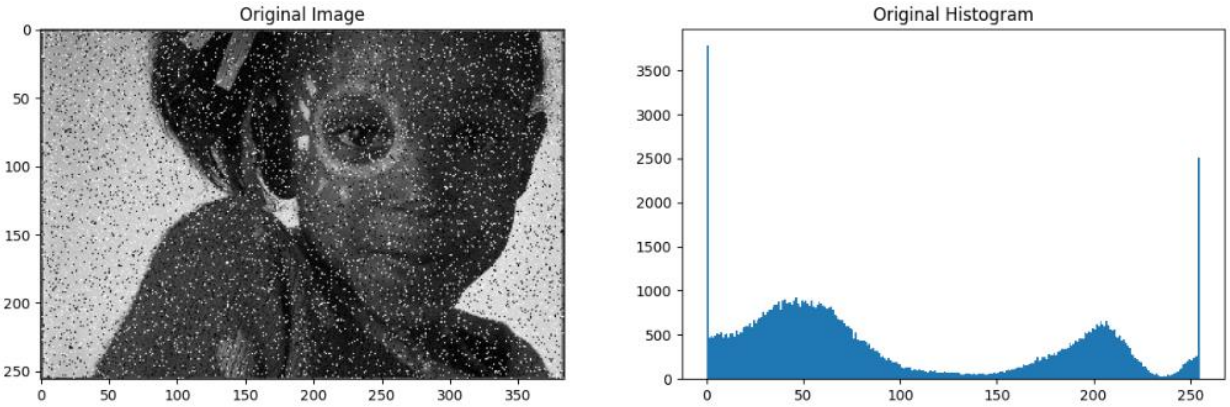


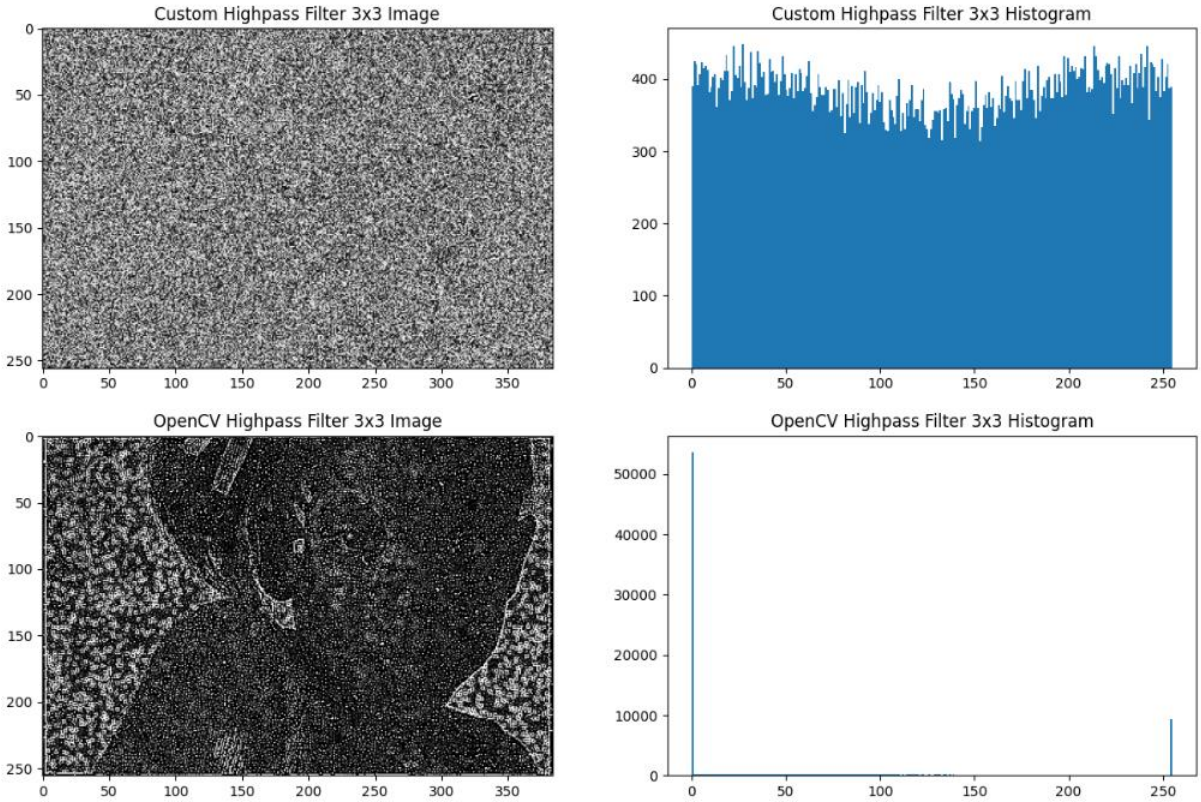
2. Highpass filter

Code

```python
def highpass_filter_custom(image, mask_size):
    kernel = -np.ones((mask_size, mask_size))
    kernel[mask_size//2, mask_size//2] = mask_size*mask_size - 1
    return convolve(image, kernel)

def highpass_filter_opencv(image):
    kernel = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]])
    return cv2.filter2D(image, -1, kernel)
```

Output



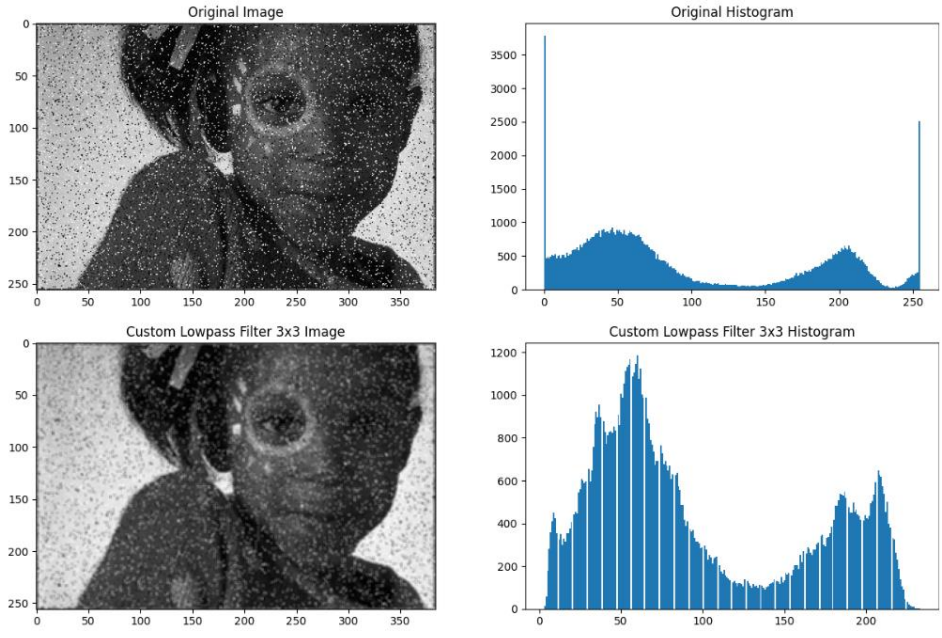


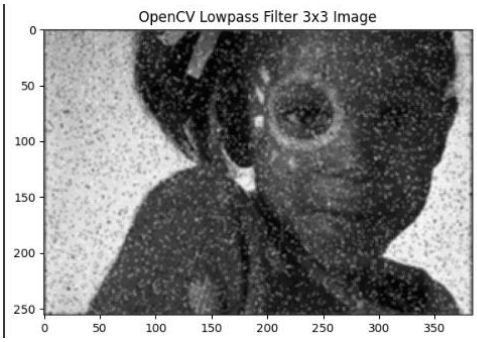
3. Lowpass filter

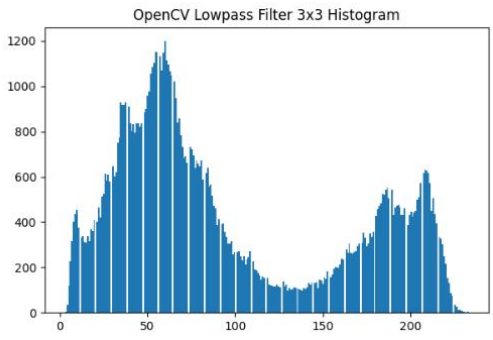
Code

```python
def lowpass_filter_custom(image, mask_size):
    kernel = np.ones((mask_size, mask_size)) / (mask_size * mask_size)
    return convolve(image, kernel)

def lowpass_filter_opencv(image, mask_size):
    return cv2.blur(image, (mask_size, mask_size))
```

Output





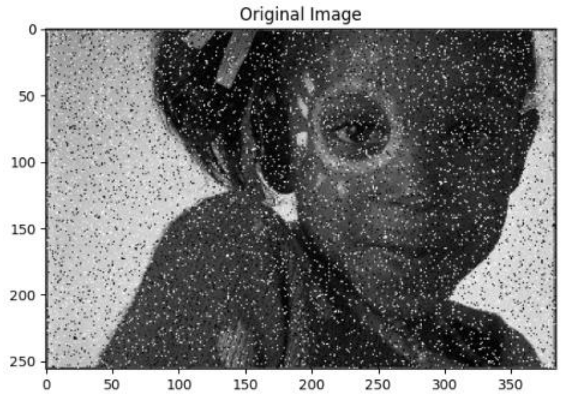


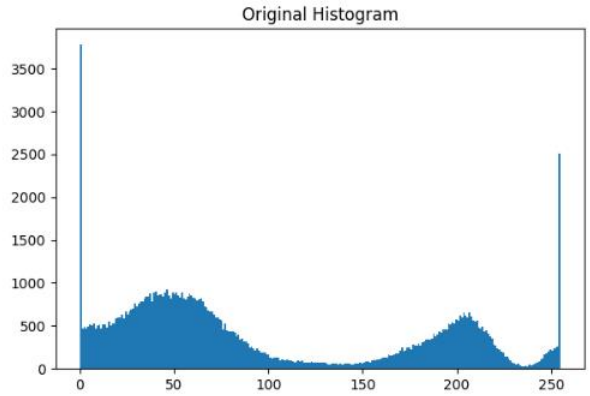
4. Bilateral filter

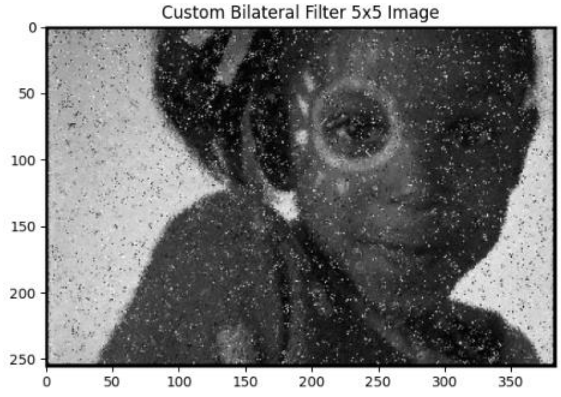
Code

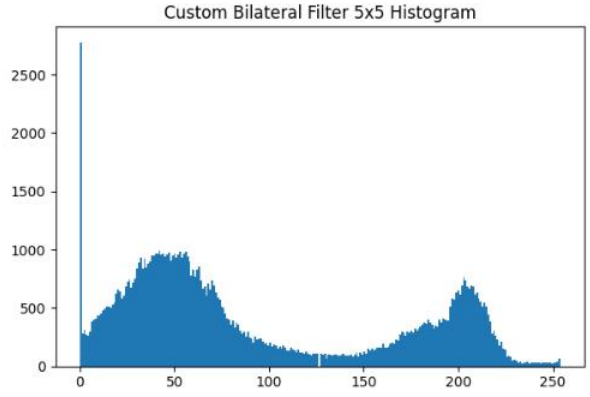
```python
def bilateral_filter_custom(image, mask_size, sigma_d, sigma_r):
    def gaussian(x, sigma):
        return np.exp(-(x**2) / (2 * sigma**2))

    filtered_image = np.zeros_like(image)
    half_size = mask_size // 2
    for i in range(half_size, image.shape[0] - half_size):
        for j in range(half_size, image.shape[1] - half_size):
            local_region = image[i-half_size:i+half_size+1, j-half_size:j+half_size+1]
            center_pixel = image[i, j]
            distance_weights = gaussian(np.arange(-half_size, half_size+1), sigma_d)
            intensity_weights = gaussian(local_region - center_pixel, sigma_r)
            weights = distance_weights[:, None] * distance_weights[None, :] * intensity_weights
            filtered_image[i,•j] = np.sum(weights * local_region) / np.sum(weights)
    return filtered_image

def bilateral_filter_opencv(image, d, sigma_color, sigma_space):
    return cv2.bilateralFilter(image, d, sigma_color, sigma_space)
```

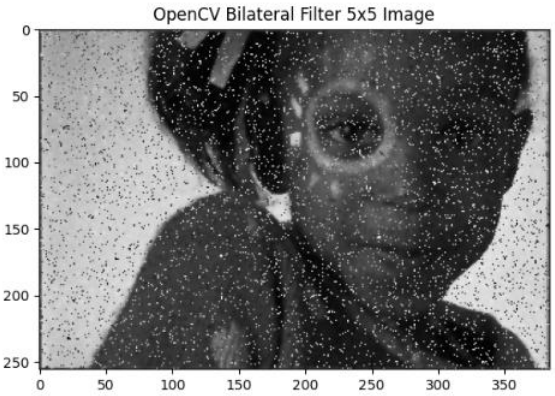
Output

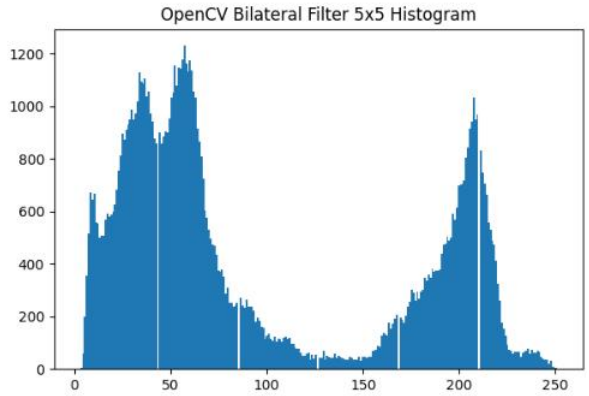










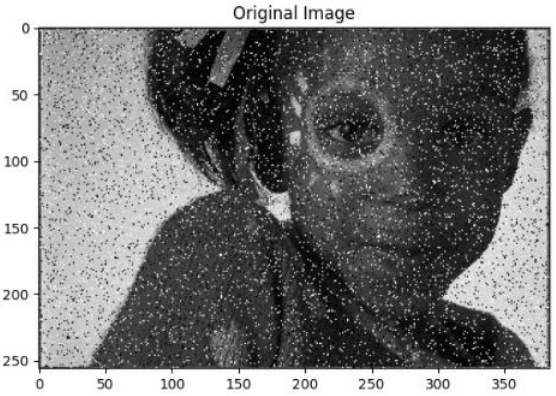


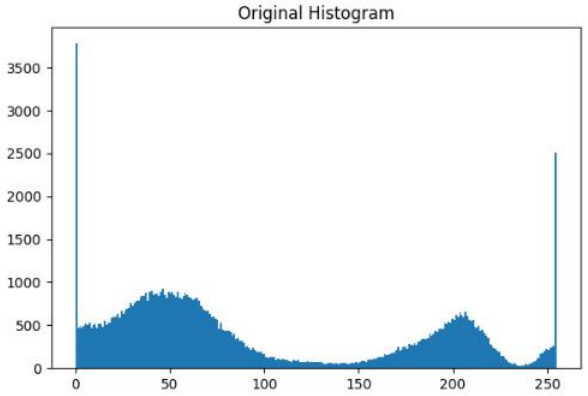
5. Gaussian filter

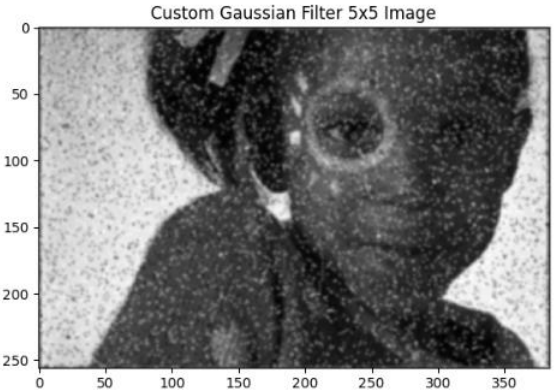
Code

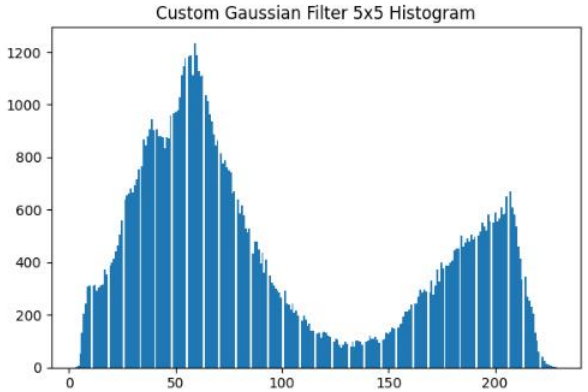
```python
def gaussian_filter_custom(image, mask_size, sigma):
    def gaussian_kernel(size, sigma):
        ax = np.linspace(-(size // 2), size // 2, size)
        xx, yy = np.meshgrid(ax, ax)
        kernel = np.exp(-0.5 * (np.square(xx) + np.square(yy)) / np.square(sigma))
        return kernel / np.sum(kernel)

    kernel = gaussian_kernel(mask_size, sigma)
    return convolve(image, kernel)

def gaussian_filter_opencv(image, mask_size, sigma):
    return cv2.GaussianBlur(image, (mask_size, mask_size), sigma)
```

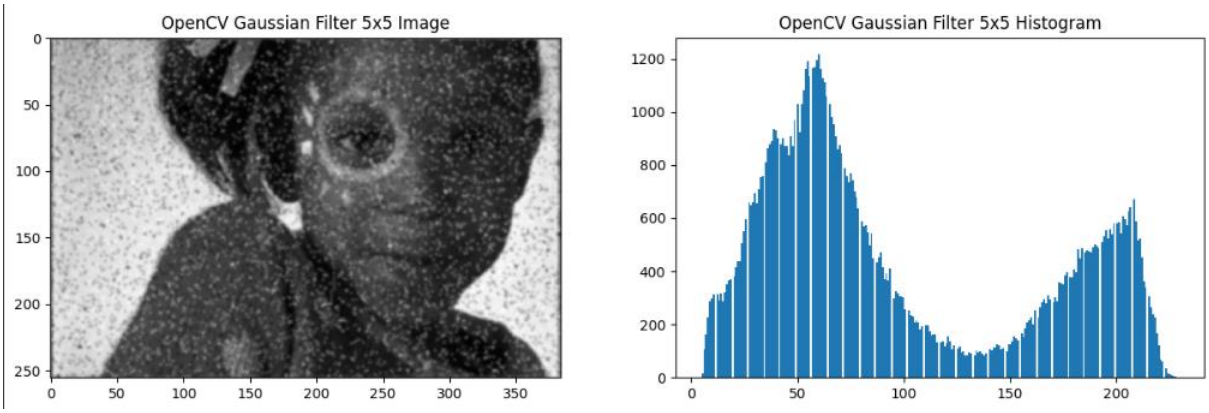
Output









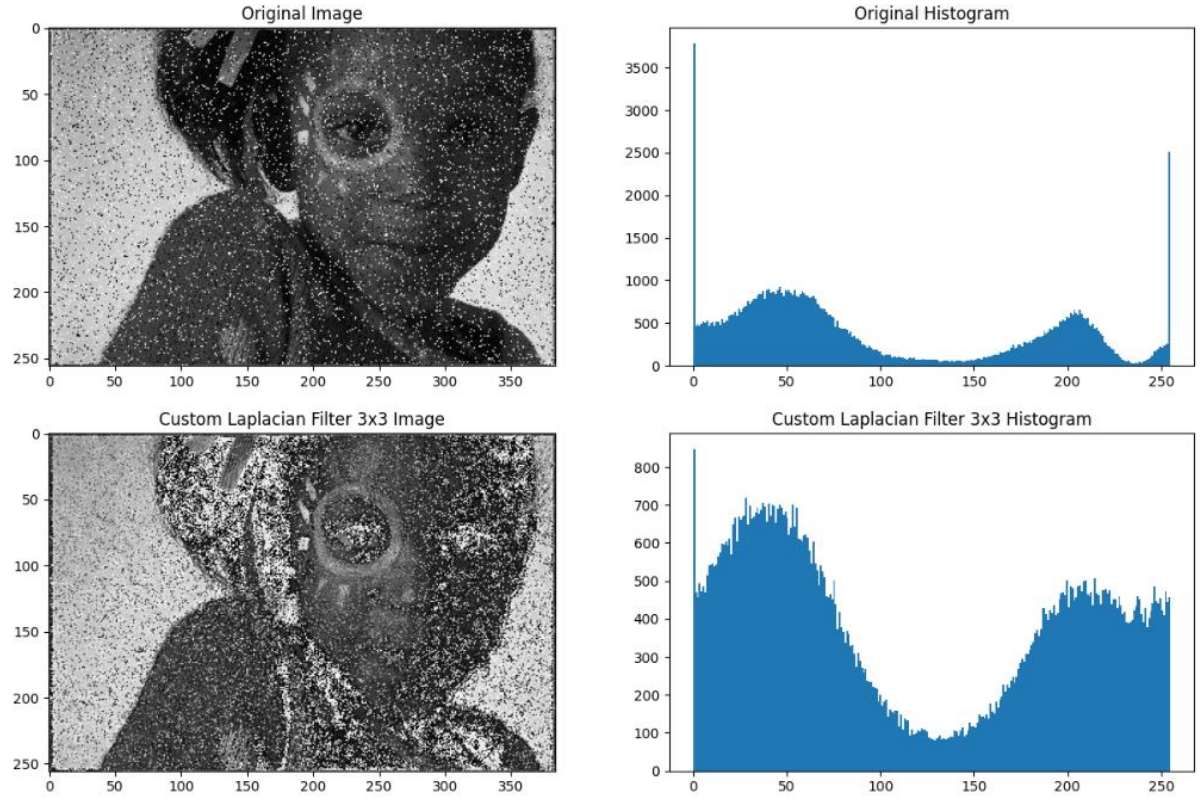


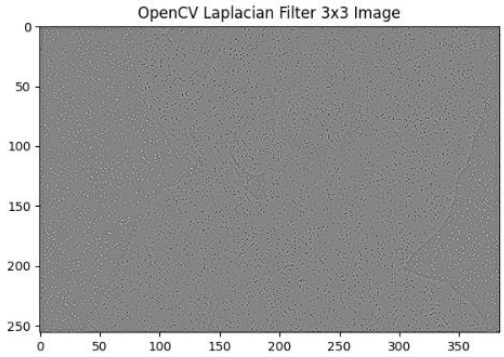
6. Laplacian filter

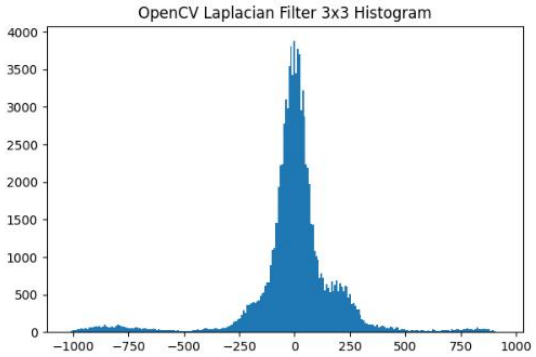
Code

```
def laplacian_filter_custom(image, mask_size):
    kernel = np.zeros((mask_size, mask_size))
    kernel[mask_size//2, mask_size//2] = 2
    kernel = kernel - np.ones((mask_size, mask_size)) / (mask_size * mask_size)
    return convolve(image, kernel)

def laplacian_filter_opencv(image):
    return cv2.Laplacian(image, cv2.CV_64F)
```

Output







7. Median filter

Code

```
def median_filter_custom(image, mask_size):
    return median_filter(image, size=mask_size)

def median_filter_opencv(image, mask_size):
    return cv2.medianBlur(image, mask_size)
```

Output