CS3713 - Image Processing

Noise Filter Implementation

Contents

Introduction

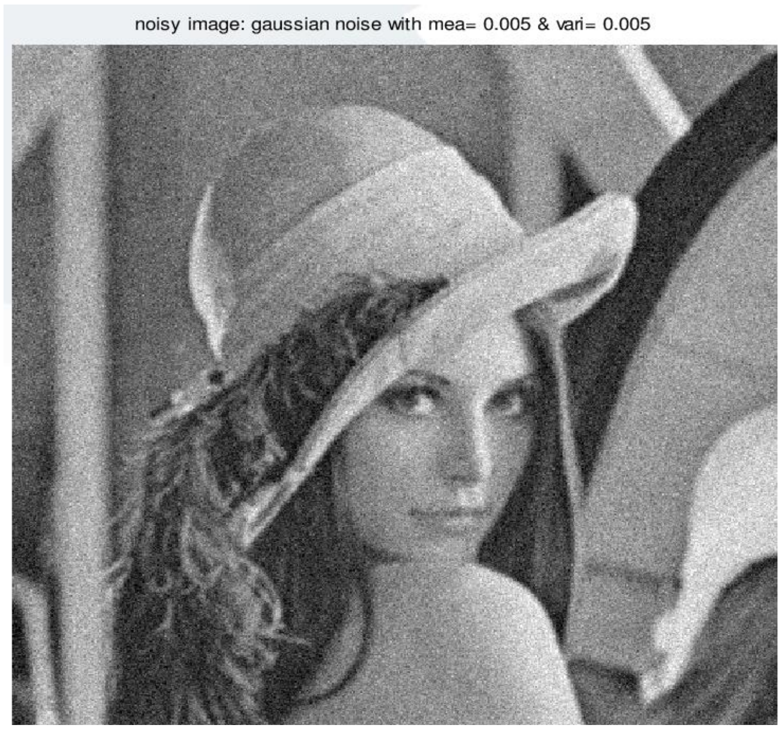
Image filtering is a fundamental process in image processing that involves modifying the pixel values of an image to enhance or suppress certain features. In this report, we explore four different image filtering methods implemented in Python using the OpenCV and NumPy libraries. The methods covered include mean filtering, median filtering, k-closest averaging, and threshold averaging. Each method aims to achieve a specific effect on the input image, and we will discuss their implementations and parameters.

Mean Filtering

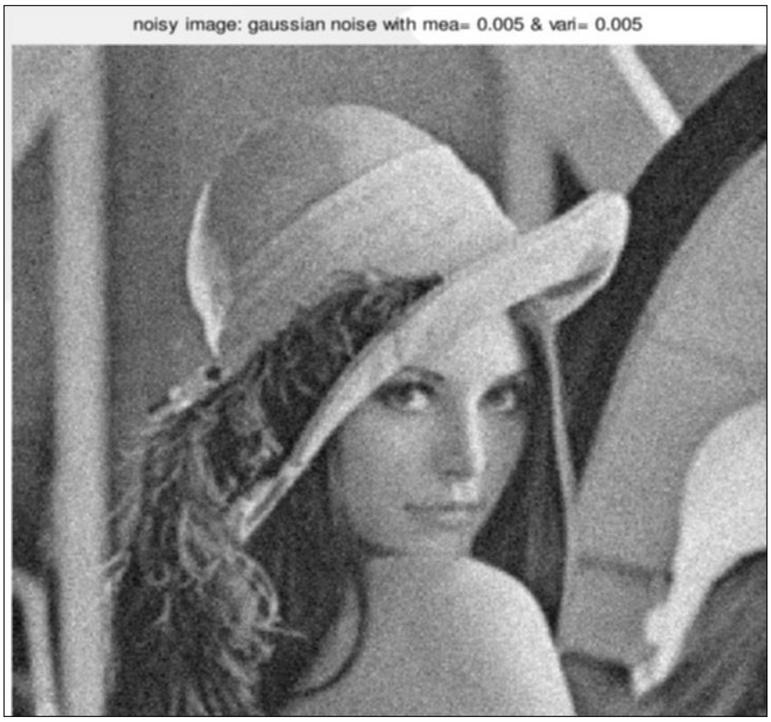
Mean filtering is a basic linear smoothing technique that replaces each pixel value with the average of its neighboring pixels. The mean filter implemented here takes a kernel size as a parameter. The larger the kernel size, the stronger the smoothing effect. The parameter kernel_size determines the size of the square-shaped filter.

```python
def mean_filter(image, kernel_size):
    # Create a mean filter kernel
    kernel = np.ones((kernel_size, kernel_size), dtype=np.float32) / 
(kernel_size**2)
    result = np.zeros_like(image, dtype=np.float32)
    pad = kernel_size // 2

    # Apply mean filter to the image
    for i in range(pad, image.shape[0] - pad):
        for j in range(pad, image.shape[1] - pad):
            result[i, j] = np.sum(image[i-pad:i+pad+1, j-pad:j+pad+1] * 
kernel)

    return result.astype(np.uint8)
```

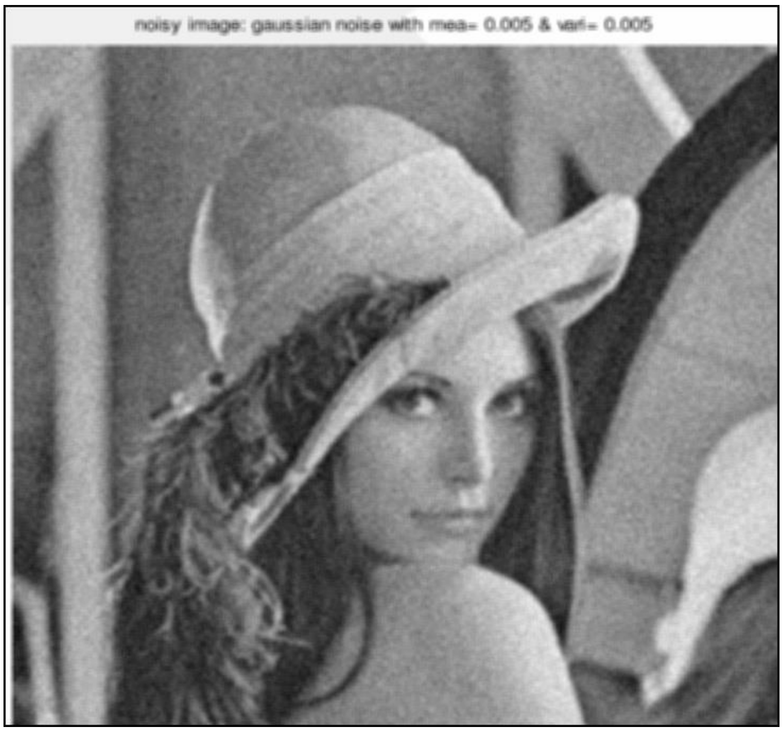
Original Image



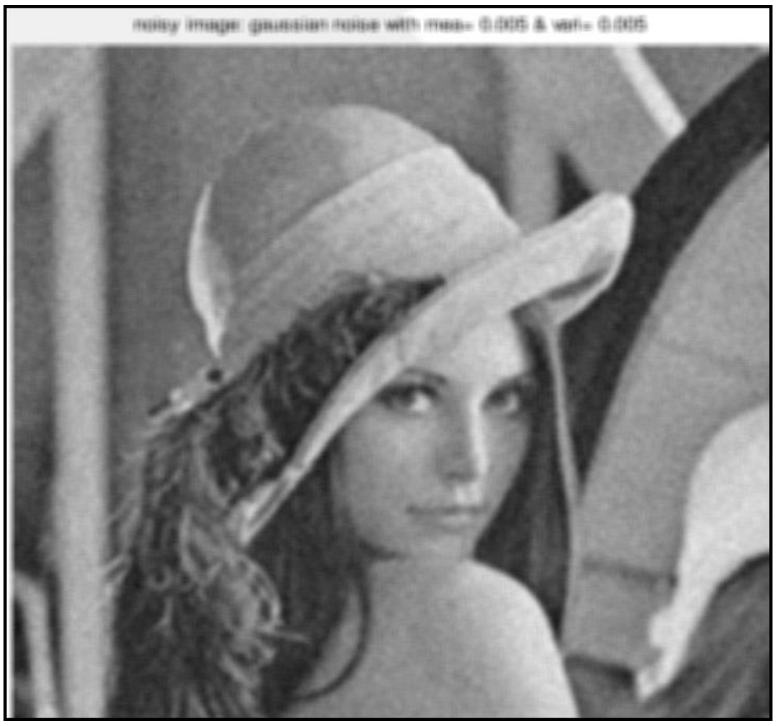
Kernel Size = 3



Kernel Size = 5



Kernel Size = 7

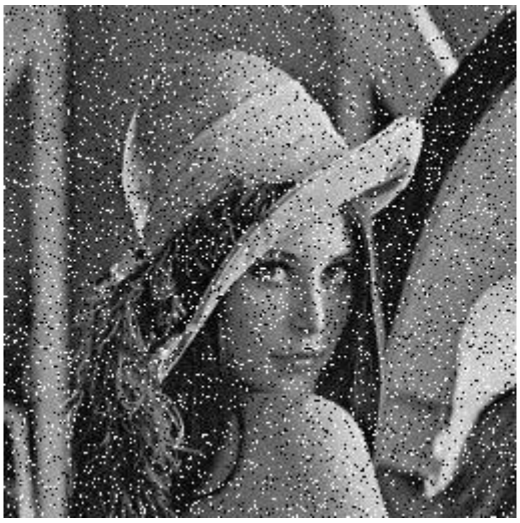


Median Filtering

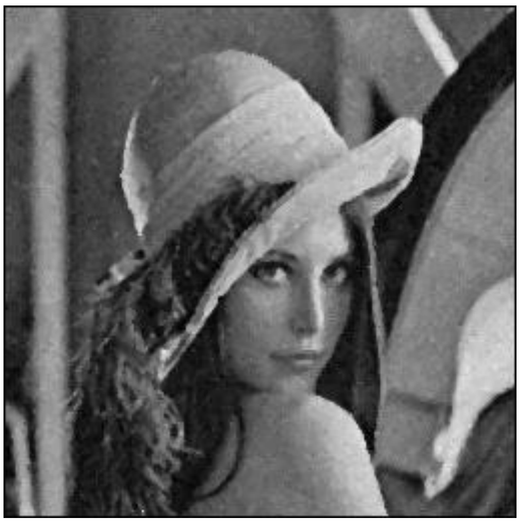
Median filtering is a non-linear technique that replaces each pixel value with the median of its neighboring pixels. This method is effective in removing salt-and-pepper noise. The kernel_size parameter is again utilized, determining the size of the square-shaped filter.

```
def median_filter(image, kernel_size):
    result = np.zeros_like(image, dtype=np.uint8)
    pad = kernel_size // 2

    # Apply median filter to the image
    for i in range(pad, image.shape[0] - pad):
        for j in range(pad, image.shape[1] - pad):
            result[i, j] = np.median(image[i-pad:i+pad+1, j-
pad:j+pad+1])

    return result
```

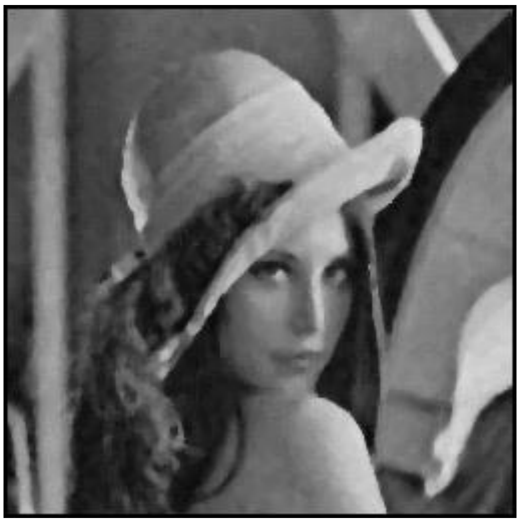
Original Image



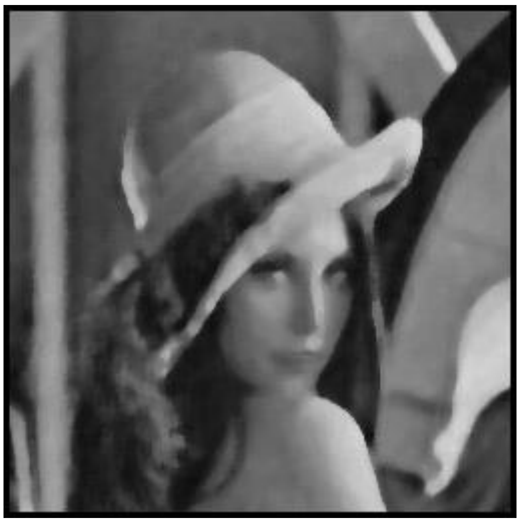
Kernel = 3



Kernel = 5



Kernel = 7

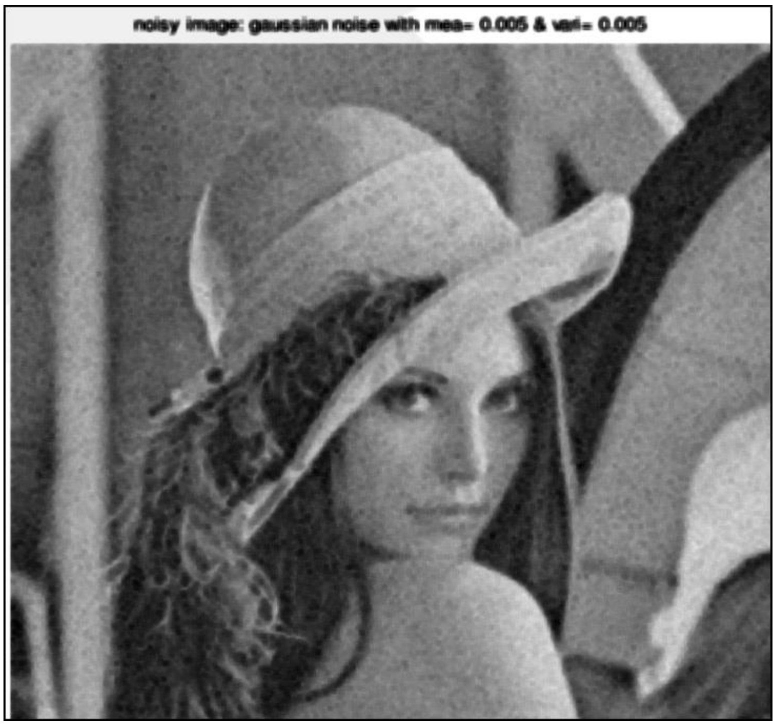


K-Closet Averaging

The k-closest averaging filter computes the average of the k smallest pixel values in the neighborhood. This method is useful for preserving edges and fine details while reducing noise. The parameters include kernel_size for the size of the filter and k for the number of closest values to consider.

```
def k_closest_averaging(image, kernel_size, k):
    result = np.zeros_like(image, dtype=np.uint8)
    pad = kernel_size // 2

    # Apply k-closest averaging filter to the image
    for i in range(pad, image.shape[0] - pad):
        for j in range(pad, image.shape[1] - pad):
            values = image[i-pad:i+pad+1, j-pad:j+pad+1].ravel()
            values.sort()
            result[i, j] = np.mean(values[:k])
    return result
```
Original Image

Kernel Size = 5 , K = 10



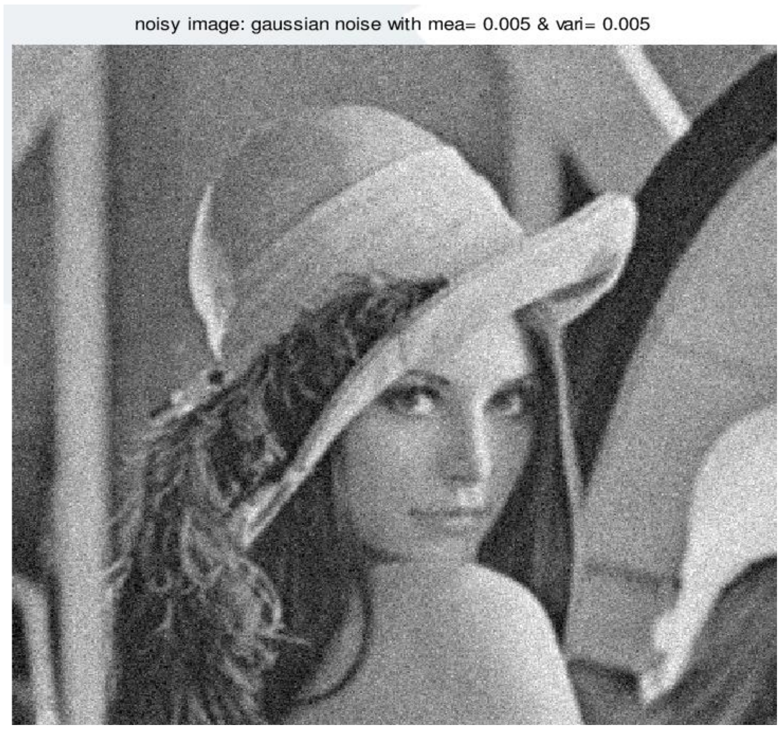
Kernel Size = 5 , K = 20

Threshold Averaging

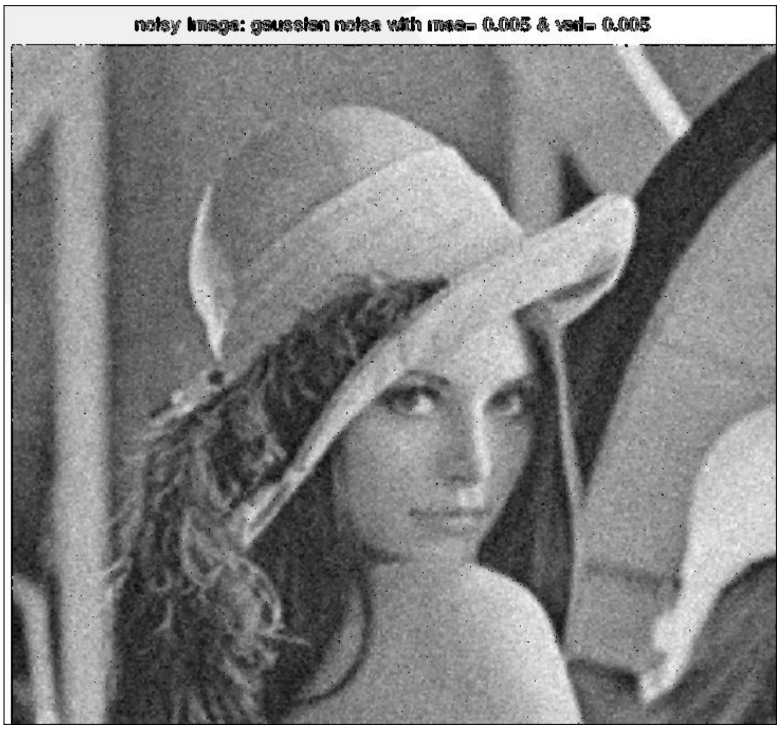
Threshold averaging calculates the mean of pixel values in the neighborhood, excluding those that deviate significantly from the mean based on a specified threshold. This can be beneficial in cases where noise has a noticeable impact on the image. The parameters include kernel_size for the filter size and threshold to control the exclusion of values deviating from the mean.

```
def threshold_averaging(image, kernel_size, threshold):
    result = np.zeros_like(image, dtype=np.uint8)
    pad = kernel_size // 2

    # Apply threshold averaging filter to the image
    for i in range(pad, image.shape[0] - pad):
        for j in range(pad, image.shape[1] - pad):
            values = image[i-pad:i+pad+1, j-pad:j+pad+1].ravel()
            mean_value = np.mean(values)
            result[i, j] = np.mean([v for v in values if abs(v - 
mean_value) < threshold])

    return result
```

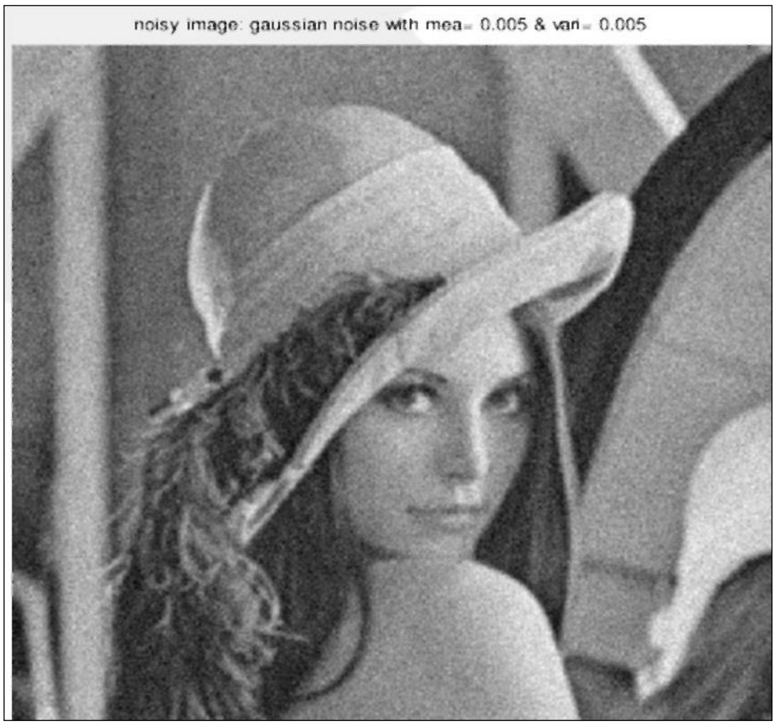
Original Image



Kernel Size = 3, Threshold = 20



Kernel Size = 3, Threshold = 100



Conclusion

In conclusion, the implemented image filtering methods provide a range of options for processing images based on specific requirements. Mean filtering is effective for general smoothing, median filtering is suitable for noise removal, k-closest averaging preserves fine details, and threshold averaging selectively reduces the impact of noisy pixels. The choice of method and parameters depends on the characteristics of the input image and the desired outcome. The provided sample images demonstrate the effects of these filters on a common test image.