Noise filter implementation

200698X
Weerasekara W.M.T.B.

Introduction

In the realm of image processing, mitigating noise is pivotal for extracting meaningful information and enhancing visual quality. This report presents the optimized implementation of four fundamental noise filters-Mean, Median, k-closest averaging, and threshold averaging in Python. These filters address noise challenges in diverse scenarios, offering a versatile toolkit for real-time image processing. The code focuses on speed, ensuring fast execution for real-time processing. Through concise explanations and sample outputs, this report demonstrates the effectiveness of each filter in enhancing image quality by reducing noise, providing a valuable resource for Python-based image processing applications.

Imported libraries in Python code

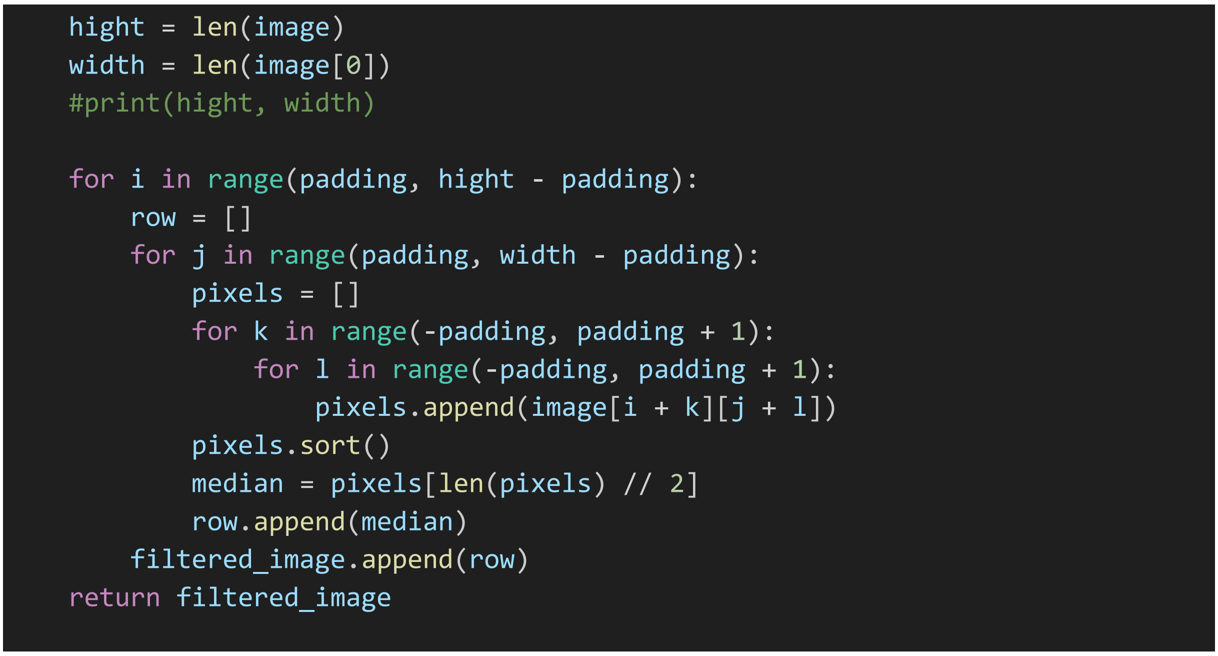
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
```

Mean filter

```

def mean_filter(image, filter_size):
    filtered_image = []
    padding = filter_size // 2

    # width and height of the image
    hight = len(image)
    width = len(image[0])
    print('Image hight =', hight, 'Width =', width)

    for i in range(padding, hight - padding):
        row = []
        for j in range(padding, width - padding):
            pixels = []
            for k in range(-padding, padding + 1):
                for l in range(-padding, padding + 1):
                    pixels.append(image[i + k][j + l])
            average = sum(pixels) // (filter_size ** 2)
            row.append(average)
        filtered_image.append(row)
    return filtered_image

```

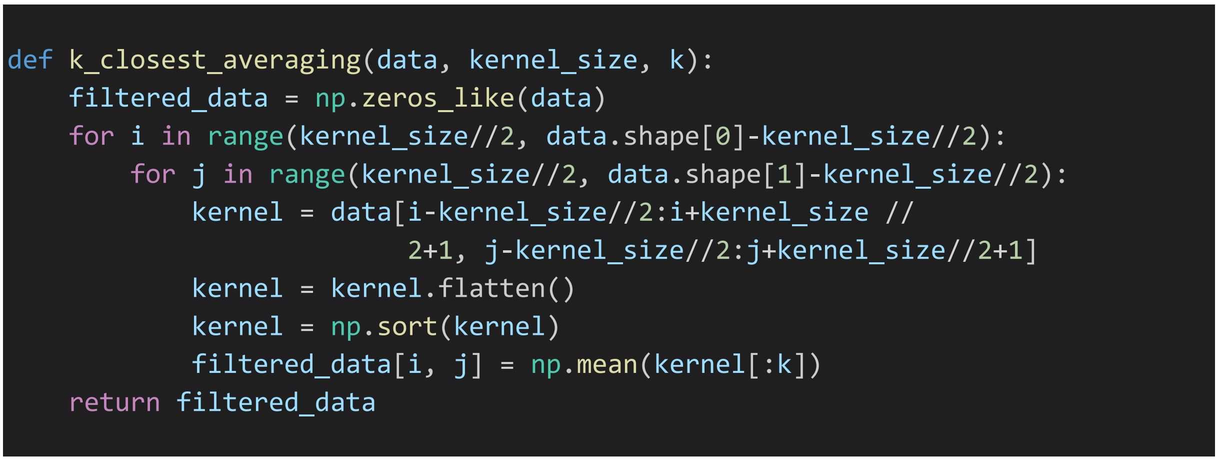
Median filter

```

def median_filter(image, filter_size):
    filtered_image = []
    padding = filter_size // 2

    # width and height of the image
```



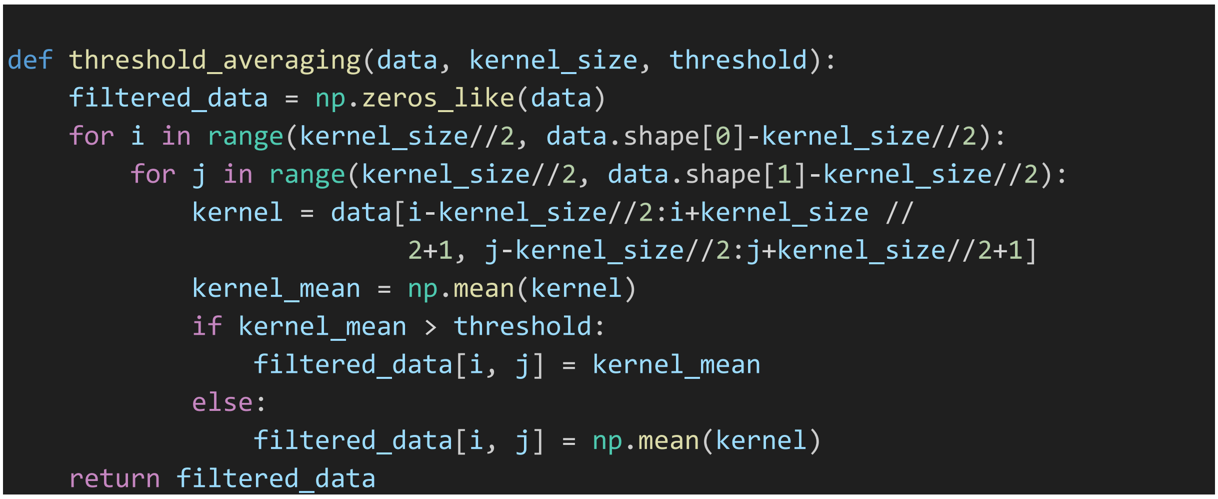
```
    hight = len(image)
    width = len(image[0])
    #print(hight, width)

    for i in range(padding, hight - padding):
        row = []
        for j in range(padding, width - padding):
            pixels = []
            for k in range(-padding, padding + 1):
                for l in range(-padding, padding + 1):
                    pixels.append(image[i + k][j + l])
            pixels.sort()
            median = pixels[len(pixels) // 2]
            row.append(median)
        filtered_image.append(row)
    return filtered_image
```

K-closest averaging filter



```
def k_closest_averaging(data, kernel_size, k):
    filtered_data = np.zeros_like(data)
    for i in range(kernel_size//2, data.shape[0]-kernel_size//2):
        for j in range(kernel_size//2, data.shape[1]-kernel_size//2):
            kernel = data[i-kernel_size//2:i+kernel_size //
                          2+1, j-kernel_size//2:j+kernel_size//2+1]
            kernel = kernel.flatten()
            kernel = np.sort(kernel)
            filtered_data[i, j] = np.mean(kernel[:k])
    return filtered_data
```

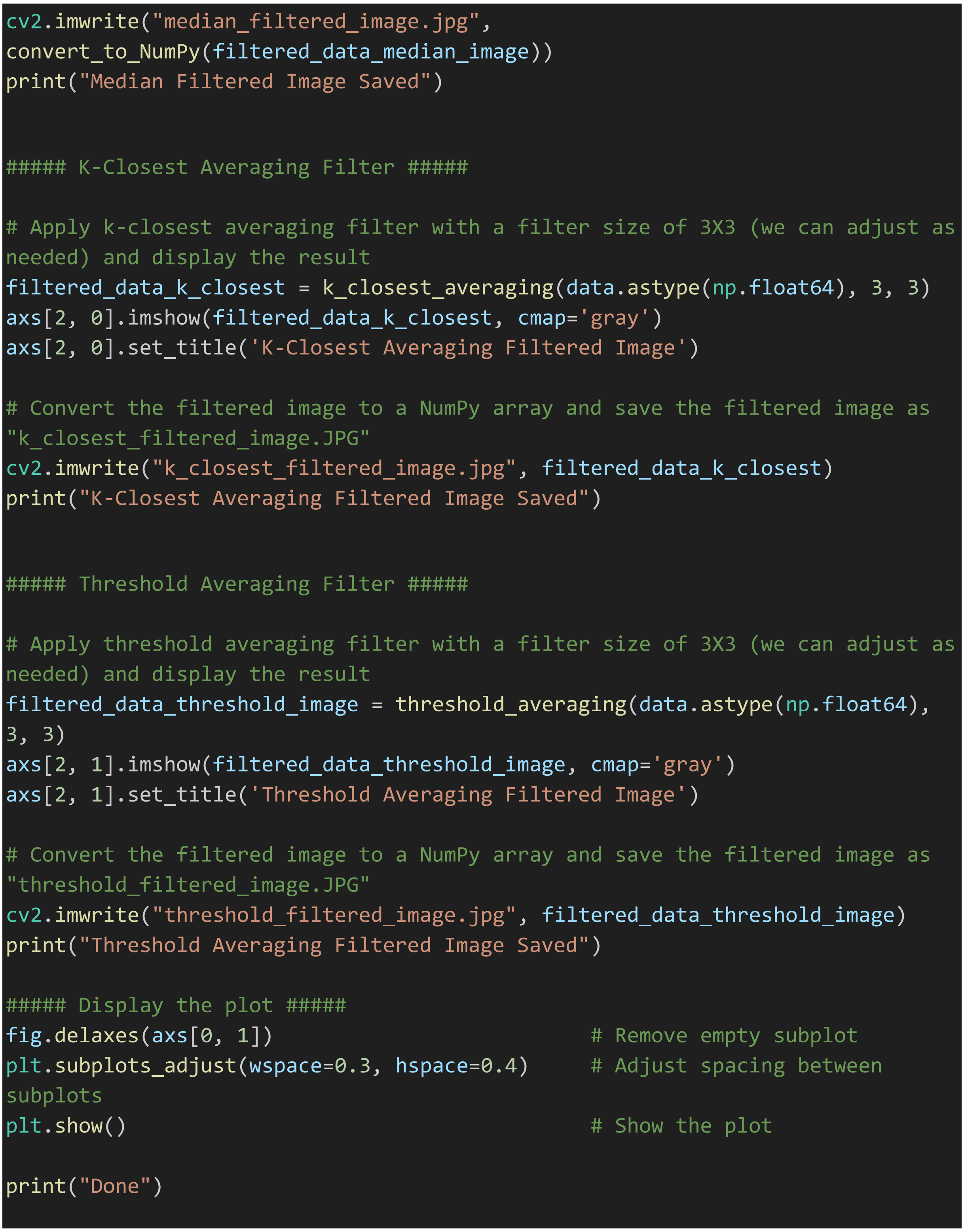
Threshold averaging filter



```
def threshold_averaging(data, kernel_size, threshold):
    filtered_data = np.zeros_like(data)
    for i in range(kernel_size//2, data.shape[0]-kernel_size//2):
        for j in range(kernel_size//2, data.shape[1]-kernel_size//2):
            kernel = data[i-kernel_size//2:i+kernel_size //
                          2+1, j-kernel_size//2:j+kernel_size//2+1]
            kernel_mean = np.mean(kernel)
            if kernel_mean > threshold:
                filtered_data[i, j] = kernel_mean
            else:
                filtered_data[i, j] = np.mean(kernel)
    return filtered_data
```

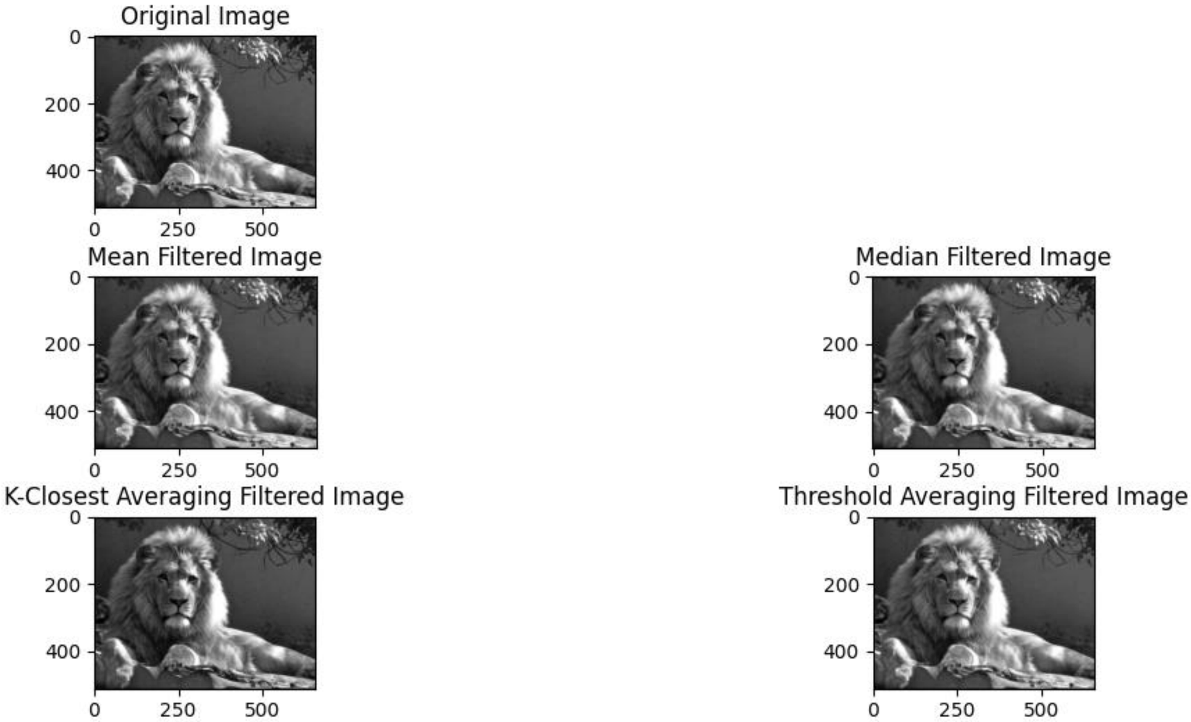
Testing with an Image

```
def convert_to_NumPy(filtered_image):
    """ Converts a list (filtered image) to a NumPy array with a data type of
unsigned 8-bit integer. """
    return np.array(filtered_image, dtype=np.uint8)


file_path = 'noiseImage.jpeg'
data = cv2.imread(file_path)
data = cv2.cvtColor(data, cv2.COLOR_BGR2GRAY)  # Convert to grayscale

# Create subplots for the original image and the filtered images
fig, axs = plt.subplots(3, 2, figsize=(10, 10))

# Show the original image
axs[0, 0].imshow(data, cmap='gray')
axs[0, 0].set_title('Original Image')


##### Mean Filter #####

# Apply mean filter with a filter size of 3 (we can adjust as needed) and
display the result
filtered_data_mean_image = mean_filter(data.astype(np.float64), 3)
axs[1, 0].imshow(filtered_data_mean_image, cmap='gray')
axs[1, 0].set_title('Mean Filtered Image')

# Convert the filtered image to a NumPy array and save the filtered image as
"mean_filtered_image.JPG"
cv2.imwrite("mean_filtered_image.jpg",
convert_to_NumPy(filtered_data_mean_image))
print("Mean Filtered Image Saved")


##### Median Filter #####

# Apply median filter with a filter size of 5 (we can adjust as needed) and
display the result
filtered_data_median_image = median_filter(data.astype(np.float64), 5)
axs[1, 1].imshow(filtered_data_median_image, cmap='gray')
axs[1, 1].set_title('Median Filtered Image')

# Convert the filtered image to a NumPy array and save the filtered image as
"median_filtered_image.JPG"
```



```
cv2.imwrite("median_filtered_image.jpg",
convert_to_NumPy(filtered_data_median_image))
print("Median Filtered Image Saved")


##### K-Closest Averaging Filter #####

# Apply k-closest averaging filter with a filter size of 3X3 (we can adjust as
needed) and display the result
filtered_data_k_closest = k_closest_averaging(data.astype(np.float64), 3, 3)
axs[2, 0].imshow(filtered_data_k_closest, cmap='gray')
axs[2, 0].set_title('K-Closest Averaging Filtered Image')

# Convert the filtered image to a NumPy array and save the filtered image as
"k_closest_filtered_image.JPG"
cv2.imwrite("k_closest_filtered_image.jpg", filtered_data_k_closest)
print("K-Closest Averaging Filtered Image Saved")


##### Threshold Averaging Filter #####

# Apply threshold averaging filter with a filter size of 3X3 (we can adjust as
needed) and display the result
filtered_data_threshold_image = threshold_averaging(data.astype(np.float64),
3, 3)
axs[2, 1].imshow(filtered_data_threshold_image, cmap='gray')
axs[2, 1].set_title('Threshold Averaging Filtered Image')

# Convert the filtered image to a NumPy array and save the filtered image as
"threshold_filtered_image.JPG"
cv2.imwrite("threshold_filtered_image.jpg", filtered_data_threshold_image)
print("Threshold Averaging Filtered Image Saved")

##### Display the plot #####
fig.delaxes(axs[0, 1])                           # Remove empty subplot
plt.subplots_adjust(wspace=0.3, hspace=0.4)      # Adjust spacing between
subplots
plt.show()                                       # Show the plot

print("Done")
```

Results


Original Image
Mean Filtered Image
Median Filtered Image
K-Closest Averaging Filtered Image
Threshold Averaging Filtered Image
0
200
400
0
250
500

GitHub Link

https://github.com/ThejanB/ImagePrecessing