MANİSA
CELAL BAYAR
UNIVERSITY

CSE 3113 / CSE 3214

INTRODUCTION TO DIGITAL IMAGE PROCESSING

SPRING 2023

Homework 3 Report

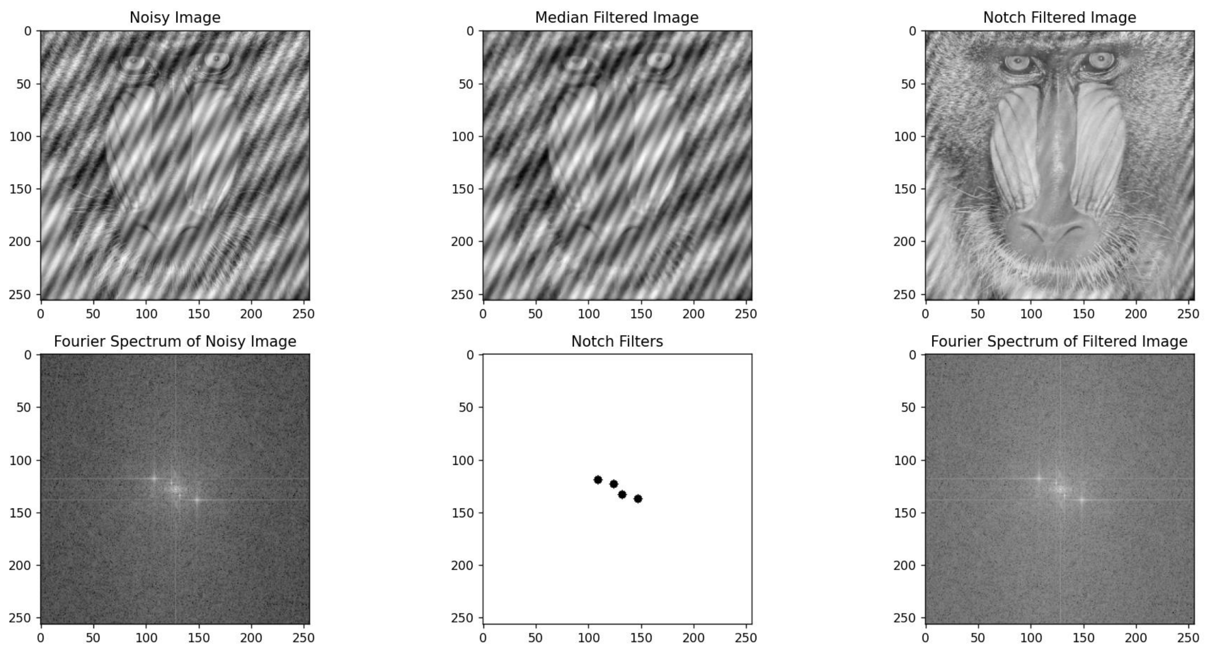
Yaren Mamuk – 190315021

Submission Date: 30 May 2023

Programming Language	*[✓ Python] ☐ Matlab ☐ Octave*
Programming Environment	*PyCharm – Python 3.10*
Reflections	*I had a hard time correcting some parts because the codes were linked. I learned to use filters while fixing this.*

Results & Discussion

1. *Paste the output figures that you generated.*



2. *Make a discussion about your results around the following questions.*
    a. *Is the median filter effective at removing this type of noise? Why/why not?*
    The median filter is effective at removing impulse or salt-and-pepper noise. It works by replacing each pixel with the median value within its local neighborhood, making it robust to outliers and preserving fine details. However, its effectiveness depends on the specific characteristics of the noise and the desired outcome.

b. *What is the effect of changing the kernel size of the median filter?*
   Changing the kernel size of a median filter has the following effects:
   1. Smoothing effect: *Larger kernel sizes result in stronger smoothing, reducing noise but blurring the image. Smaller kernel sizes preserve more details and introduce less smoothing.*
   2. Edge preservation: *Smaller kernel sizes better preserve edges and fine details. Larger kernel sizes can blur or soften edges.*
   3. Computational cost: *Larger kernel sizes require more computations, leading to slower processing times.*
   4. Filtering artifacts: *Different kernel sizes can introduce artifacts such as isolated noisy pixels or oversmoothing.*

c. *Can you remove this type of noise effectively using some other spatial domain filters?*
   The choice of the most effective filter depends on the specific type and characteristics of the noise present in the image. Here are a few common spatial domain filters used to address specific types of noise:
   1. Gaussian Filter: *Gaussian noise, characterized by random variations in pixel intensities, can be effectively reduced using a Gaussian filter. This filter applies a weighted average to each pixel's neighborhood, with the weights determined by a Gaussian distribution. It effectively blurs the image, reducing the impact of Gaussian noise.*
   2. Mean Filter: *Mean filters compute the average intensity of a pixel's neighborhood. While it can reduce Gaussian noise to some extent, it is not as effective as a Gaussian filter. Mean filters work well for noise that follows a uniform distribution, such as uniform noise.*
   3. Wiener Filter: *The Wiener filter is a statistical filter that can effectively reduce noise while preserving image details. It operates by estimating the power spectrum of both the noise and the original image, and then applying a filter that minimizes the mean square error between the original image and the filtered image. The Wiener filter is commonly used for noise reduction in images corrupted by additive white Gaussian noise.*

4. Bilateral Filter: *The bilateral filter is useful for preserving edges and textures while reducing noise. It considers both the spatial proximity and intensity similarity of neighboring pixels. By applying different weights based on these factors, the bilateral filter effectively smooths noise while preserving important image features.*
5. Adaptive Filter: *Adaptive filters adapt their filter parameters according to the local characteristics of the image. These filters can effectively reduce noise while maintaining image details. Examples include the adaptive mean filter and the adaptive median filter, which adjust their kernel sizes based on the noise level and local image content.*

Source Code

```python
import cv2
import numpy as np
from matplotlib import pyplot as plt

def notch_reject_filter(shape, d0, u_k=0, v_k=0):
    P, Q = shape
    H = np.zeros((P, Q))

    for u in range(0, P):
        for v in range(0, Q):
            D_uv = np.sqrt((u - P / 2 + u_k) ** 2 + (v - Q / 2 + v_k) ** 2)
            D_muv = np.sqrt((u - P / 2 - u_k) ** 2 + (v - Q / 2 - v_k) **
2)

            if D_uv <= d0 or D_muv <= d0:
                H[u, v] = 0.0
            else:
                H[u, v] = 1.0

    return H

image_path = "image12.tif"
im1 = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

im2 = cv2.medianBlur(im1, ksize=3)

im3 = np.fft.fftshift(np.fft.fft2(im2))

plt.subplot(2, 3, 1)
plt.imshow(im1, cmap="gray")
plt.title("Noisy Image")

plt.subplot(2, 3, 2)
plt.imshow(im2, cmap="gray")
plt.title("Median Filtered Image")

f = np.fft.fft2(im1)
```

```
fshift = np.fft.fftshift(f)
phase_spectrumR = np.angle(fshift)
magnitude_spectrum = 1*np.log(np.abs(fshift))
img_shape = im1.shape

H1 = notch_reject_filter(img_shape, 4,5, 4)
H2 = notch_reject_filter(img_shape, 4, 9, 19)
H3 = notch_reject_filter(img_shape, 4, 245, 235)
H4 = notch_reject_filter(img_shape, 4, 250, 252)

NotchFilter = H1*H2*H3*H4
NotchRejectCenter = fshift * NotchFilter
NotchReject = np.fft.ifftshift(NotchRejectCenter)
inverse_NotchReject = np.fft.ifft2(NotchReject)

Result = np.abs(inverse_NotchReject)
Result_final = Result
plt.subplot(2, 3, 3)
plt.imshow(Result, cmap="gray")
plt.title("Notch Filtered Image")


im4 = np.fft.fftshift(np.fft.fft2(im1))

plt.subplot(2, 3, 4)
plt.imshow(np.log(1 + np.abs(im4)), cmap="gray")
plt.title("Fourier Spectrum of Noisy Image")


im5 = np.ones_like(im4)
im5[5, 4] = 0
im5[9, 19] = 0
im5[245, 235] = 0
im5[250, 252] = 0

im6 = im4 * im5

plt.subplot(2, 3, 5)
plt.imshow(NotchFilter, cmap="gray")
plt.title("Notch Filters")

plt.subplot(2, 3, 6)
plt.imshow(np.log(1 + np.abs(im6)), cmap="gray")
plt.title("Fourier Spectrum of Filtered Image")


plt.tight_layout()
plt.show()
```