INT3404E 20 - Image Processing: Homeworks 2

Vu Minh Tuan

Contents

List of Figures

1 Introduction

This report indicates the result of Homework 2. The implementation was pushed to Github repository, and due to its length, I will not show all of the code in this report.

2 Image Filtering

2.1 Padding image

I have implemented the *padding_img* function using all *numpy*, the result is represented below. The image *Before* and *After* illustrate the noise image and the padded image respectively.

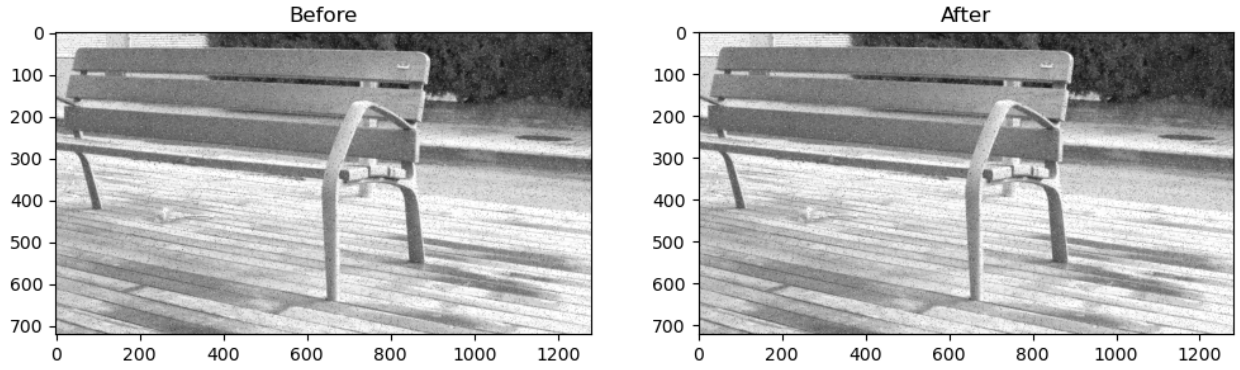

Figure 1: Padding image

2.2 Mean filter

I have implemented the *mean_filter* function, the result is represented below. The image *Before* and *After* illustrate the noise image and the smoothed image respectively.

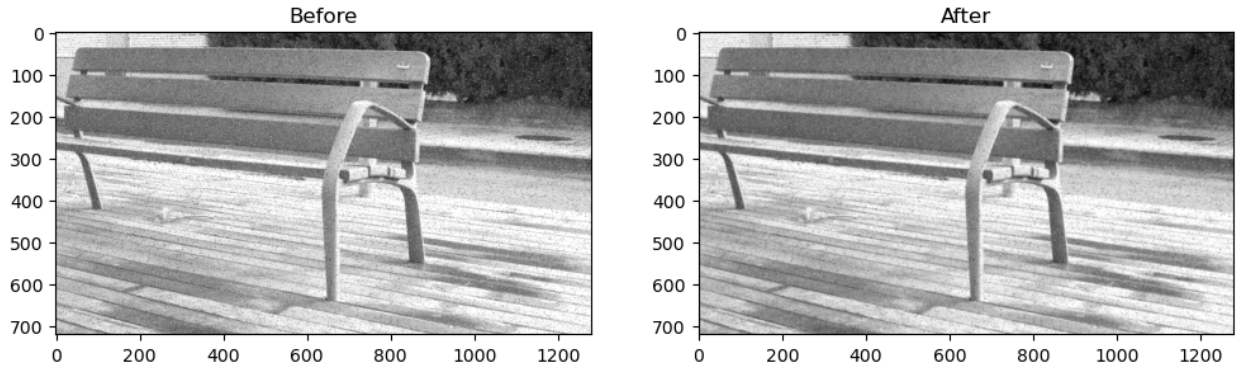

Figure 2: Mean filtering

2.3 Median filter

I have implemented the *median_filter* function, the result is represented below. The image *Before* and *After* illustrate the noise image and the smoothed image respectively.

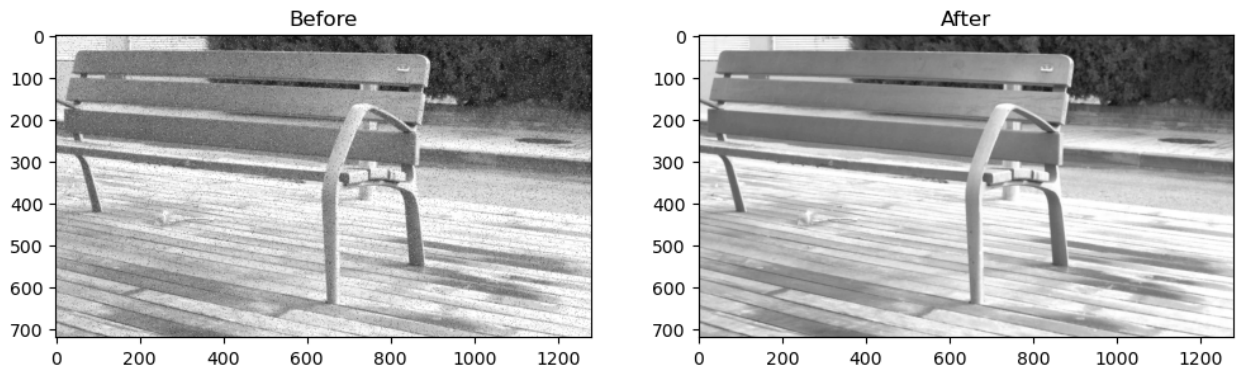

Figure 3: Median filtering

2.4 Peak Signal-to-Noise Ratio

After calculating the PSNR score for *Mean filter* and *Median filter*, the result is:

- PSNR score of mean filter: 31.60889963499979
- PSNR score of median filter: 37.119578300855245

In conclusion, Median filter is a better choice.

3 Fourier Transform

3.1 1D Fourier Transform

I implement the function for 1D Fourier Transform based on the formula Discrete Fourier Transform

Function: DFT_slow

```
def DFT_slow(data):
  N = len(data)
  n = np.arange(N)
  k = n.reshape((N, 1))
  e = np.exp(-2j * np.pi * k * n / N)

  X = np.dot(e, data)

  return X
```

3.2 2D Fourier Transform

Function: DFT_2D

```
def DFT_2D(gray_img):
  row_fft = np.fft.fft(gray_img, axis=1)
  row_col_fft = np.fft.fft(row_fft, axis=0)

  return row_fft, row_col_fft
```

3.3 Frequency Removal Procedure

I implement the *filter_frequency* function based on the guidelines

- Transform using fft2

- Shift frequency coefs to center using fftshift
- Filter in frequency domain using the given mask
- Shift frequency coefs back using ifftshift
- Invert transform using ifft2

Function: filter_frequency

```
def filter_frequency(orig_img, mask):
  dft = np.fft.fft2(orig_img)

  dft_shifted = np.fft.fftshift(dft)

  dft_filtered = dft_shifted * mask

  dft_inv_shifted = np.fft.ifftshift(dft_filtered)

  img = np.fft.ifft2(dft_inv_shifted)
  img = np.real(img)

  f_img = np.abs(dft_filtered)
  return f_img, img
```

3.4 Creating a Hybrid Image

First, I write a function to compute the *mask*. I compute the height and width of the image, then create a zero matrix corresponding to height and width. The center will then be calculated in order to calculate the mask area as what is presented in the code below. The mask area will be marked 1 in the mask matrix.

Function: create_mask

```
def create_mask(img1, r):
  h, w = img1.shape
  mask = np.zeros((h, w), dtype=np.float64)
  center = (h // 2, w // 2)
  y, x = np.ogrid[:h, :w]
  mask_area = (x - center[1]) ** 2 + (y - center[0]) ** 2 <= r**2
  mask[mask_area] = 1
  return mask
```

Similar to what I have done in the previous subsection, I implement a function to create hybrid image from 2 input images. The code is already in Github repository. The output of my implementation is illustrated in Figure 4.

Figure 4: Hybrid image