INT3404E - Image Processing : Homeworks 2

Pham Thu Trang - 21020248

1 Image Filtering

1.1 Padding

Code for Replicate padding shows in firgue 1.

1.2 Mean Filtering

Code for mean filter and results shows in firgue 2 and 3, respectively.

1.3 Median Filtering

Code for mean filter and results shows in firgue 4 and 5, respectively.

1.4 Peak Signal-to-Noise Ratio (PSNR) metric

Firgue 6 shows code of loadding image.

$$PNSR = 10.\log_{10}(\frac{MAX^2}{MSE}) \tag{1}$$

where MAX is the maximum possible pixel value (typically 255 for 8-bit images), and MSE is the Mean Square Error between the two images.

After applying the PSNR metric to the filtered image and the ground truth image, the PSNR of the image using the mean filter and the median filter are 26.26318211013139 and 36.97746079407715, respectively. It can be easily seen that for the given original image, the median filter performs better.

2 Fourier Transform

2.1 1D Fourier Transform

The Discrete Fourier Transform:

$$F(s) = \frac{1}{N}\sum_{n=0}^{N-1} f[n]e^{-i2\pi sn/N} \tag{2}$$

and its inverse by:

$$f[n] = \sum_{s=0}^{N-1} F[s]e^{i2\pi sn/N} \tag{3}$$

```python
def padding_img(img, filter_size=3):
    # Need to implement here
    img_padded = np.zeros((np.shape(img)[0] + filter_size // 2 * 2, np.shape(img)[1] + filter_size // 2 * 2), dtype = np.float32)
    img_padded[filter_size//2:-filter_size//2 + 1, filter_size//2:-filter_size//2 + 1] = img
    img_padded[:filter_size//2, filter_size//2:-filter_size//2 + 1] = img[0]
    img_padded[-filter_size//2 + 1:, filter_size//2:-filter_size//2  + 1] = img[-1]
    img_padded[filter_size//2:-filter_size//2 + 1, :filter_size//2] = img[:, 0:1]
    img_padded[filter_size//2:-filter_size//2 + 1, -filter_size//2 + 1:] = img[:, -1:np.shape(img)[1]]

    img_padded[:filter_size//2, :filter_size//2] = img[0, 0]
    img_padded[:filter_size//2, -filter_size//2 + 1:] = img[0, -1]
    img_padded[-filter_size//2 + 1:, :filter_size//2] = img[-1, 0]
    img_padded[-filter_size//2 + 1:, -filter_size//2 + 1:] = img[-1, -1]
    return img_padded
```

Figure 1: Code replicate padding

```python
def mean_filter(img, filter_size=3):
    # Need to implement here
    img_padded = padding_img(img, filter_size)
    img_filtered = np.zeros(np.shape(img), dtype = np.float32)
    for i in range(len(img_filtered)):
        for j in range(len(img_filtered[i])):
            img_filtered[i][j] = np.average(img_padded[i:i+filter_size, j:j+filter_size])
    return img_filtered
```

Figure 2: Code mean filter

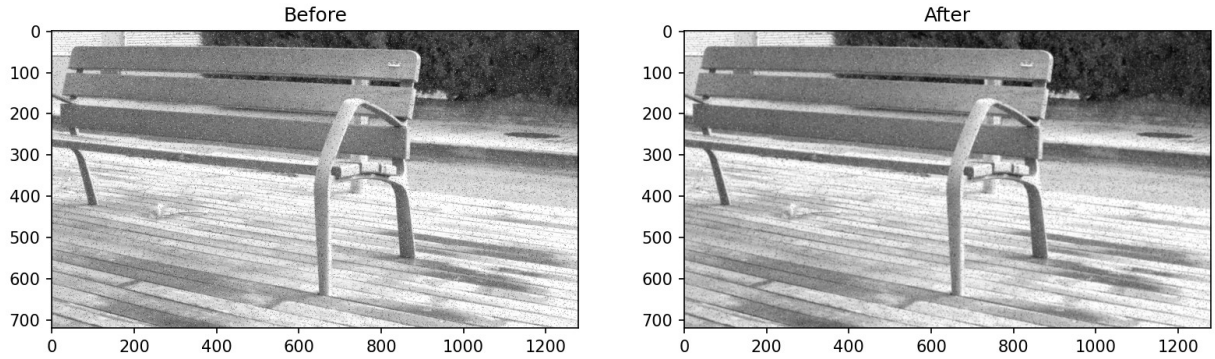

Figure 3: Image after using mean filter

```python
def median_filter(img, filter_size=3):
    # Need to implement here
    img_padded = padding_img(img, filter_size)
    img_filtered = np.zeros(np.shape(img), dtype = np.float32)
    for i in range(len(img_filtered)):
        for j in range(len(img_filtered[i])):
            img_filtered[i, j] = np.median(img_padded[i:i+filter_size, j:j+filter_size])
    return img_filtered
```

Figure 4: Code median filter

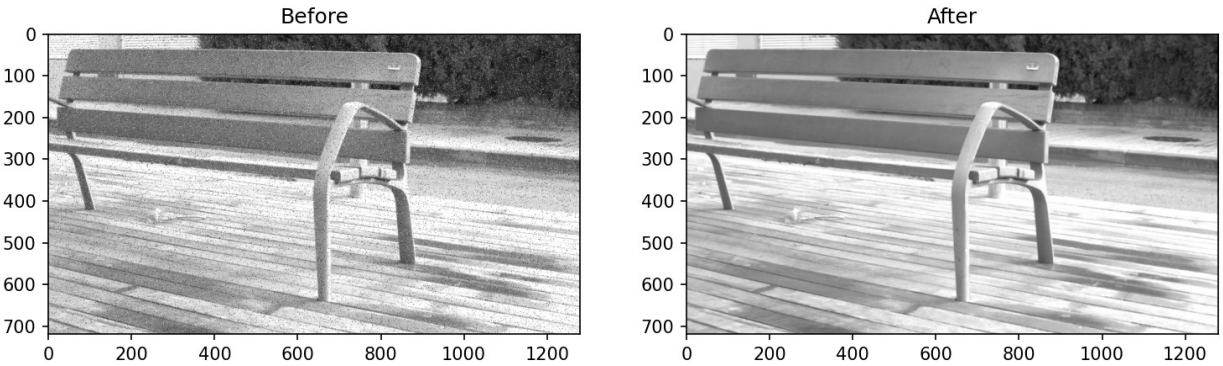

Figure 5: Image after using median filter

```python
def psnr(gt_img, smooth_img):
    # Need to implement here
    gt_img = gt_img.astype(np.float32)
    mse_score = np.mean((gt_img - smooth_img) ** 2)
    return 10 * np.log10(255.0 ** 2 / mse_score)
```

Figure 6: Code peak Signal-to-Noise Ratio (PSNR) metric

```python
def DFT_slow(data):
    # You need to implement the DFT here
    N = np.shape(data)[0]
    matrix = np.zeros((N, N), dtype = np.complex64)
    for n in range(N):
        for s in range(N):
            matrix[n][s] = np.exp(-2j * np.pi * n * s / N)
    return np.dot(matrix, data)
```

Figure 7: Code the Discrete Fourier Transform (DFT) on a one-dimensional signal

where s represents the frequency, and n denotes the sampling order

$$\begin{bmatrix} F_0 \\ F_1 \\ \dots \\ F_n \end{bmatrix} = \begin{bmatrix} e^{i2\pi 0*0/N} & e^{i2\pi 0*1/N} & e^{i2\pi 0*2/N} & \dots & e^{i2\pi 0*N/N} \\ e^{i2\pi 1*0/N} & e^{i2\pi 1*1/N} & e^{i2\pi 1*2/N} & \dots & e^{i2\pi 1*N/N} \\ \dots & \dots & \dots & \dots & \dots \\ e^{i2\pi N*0/N} & e^{i2\pi N*1/N} & e^{i2\pi N*2/N} & \dots & e^{i2\pi N*N/N} \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ \dots \\ f_n \end{bmatrix} \tag{4}$$

Code for the Discrete Fourier Transform (DFT) on a one-dimensional signal shows in firgue 7.

2.2 2D Fourier Transform

The procedure to simulate a 2D Fourier Transform is as follows:

- Conducting a Fourier Transform on each row of the input 2D signal. This step transforms the signal along the horizontal axis.
- Perform a Fourier Transform on each column of the previously obtained result.

The code and the output show in firgue 8 and 9, respectively.

2.3 Frequency Removal Procedure

Follow these steps to manipulate frequencies in the image:

- 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

The code and the output show in firgue 10 and 11, respectively.

```python
def DFT_2D(gray_img):
    # You need to implement the DFT here
    H, W = np.shape(gray_img)
    row_fft = np.zeros((H, W), dtype = np.complex64)
    for h in range(H):
        row_fft[h] = DFT_slow(gray_img[h])
    row_col_fft = row_fft.T.copy()
    for w in range(W):
        row_col_fft[w] = DFT_slow(row_col_fft[w])
    row_col_fft = row_col_fft.T
    return row_fft, row_col_fft
```

Figure 8: Code the Discrete Fourier Transform (DFT) on a two-dimensional signal

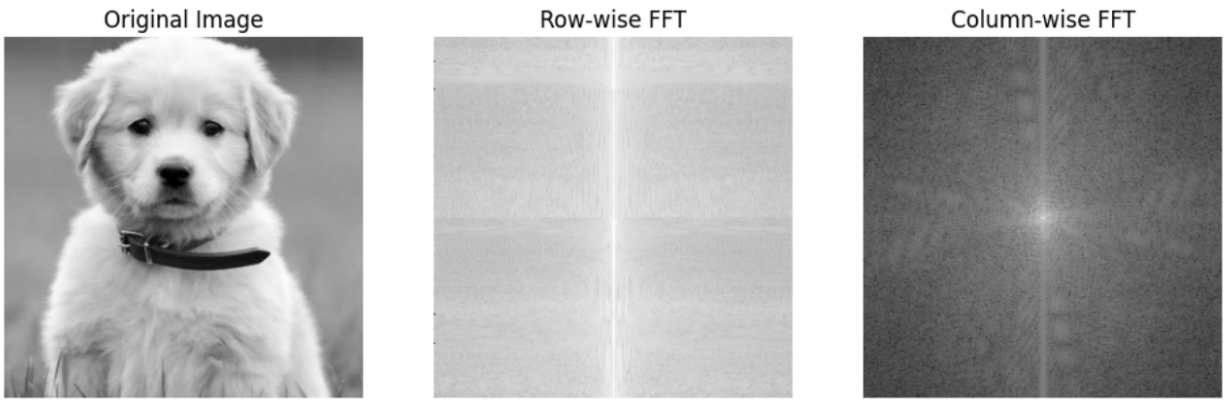

Figure 9: The output of the Discrete Fourier Transform (DFT) on a two-dimensional signal

```python
def filter_frequency(orig_img, mask):
    # You need to implement this function
    fft_img = np.fft.fft2(orig_img)

    shifted_img = np.fft.fftshift(fft_img)

    f_img = shifted_img * mask

    shifted_back_img = np.fft.ifftshift(f_img)

    img = np.fft.ifft2(shifted_back_img)

    return f_img, img
```

Figure 10: Code for 2D Frequency Removal Procedure

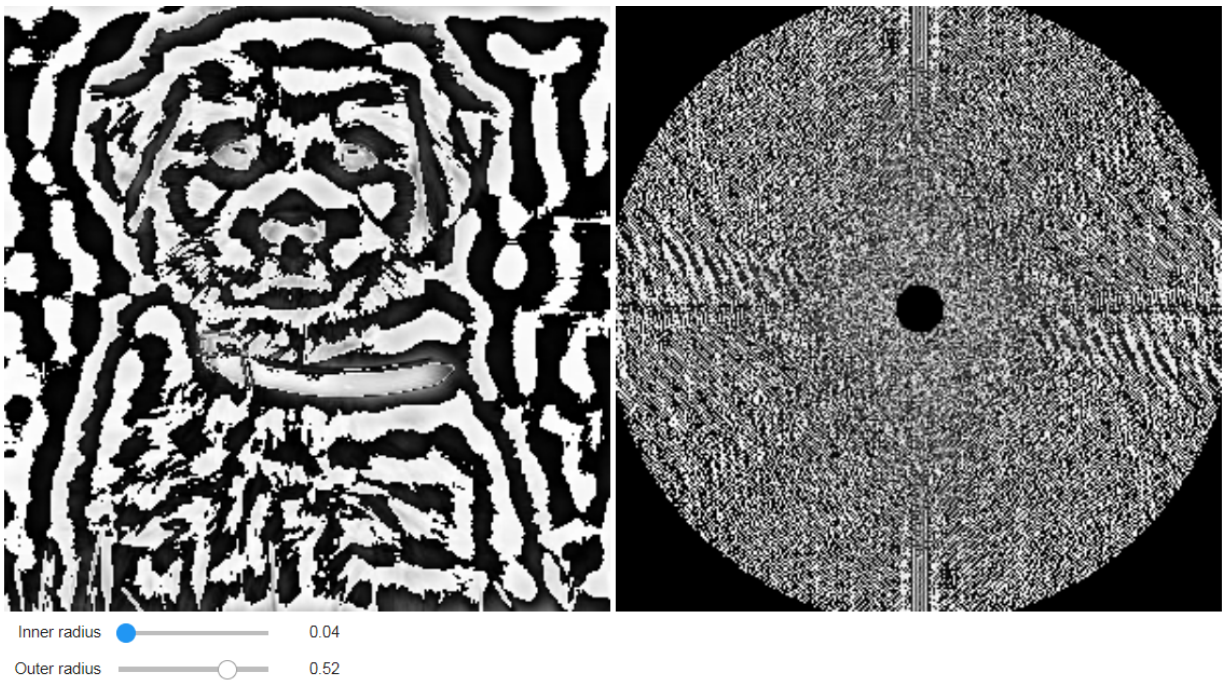

Figure 11: The output of the 2D Frequency Removal Code

2.4 Creating a Hybrid Image

To create a hybrid image, follow these steps:

- Transform using fft2
- Shift frequency coefs to center using fftshift
- Create a mask based on the given radius (r) parameter
- Combine frequency of 2 images using the mask
- Shift frequency coefs back using ifftshift
- Invert transform using ifft2

The code and the output show in firgue 12 and 13, respectively.

3 Link Github

Link github homework 2

```python
def create_hybrid_img(img1, img2, r):
  # You need to implement the function

  fft_img1 = np.fft.fft2(img1)
  fft_img2 = np.fft.fft2(img2)

  shifted_img1 = np.fft.fftshift(fft_img1)
  shifted_img2 = np.fft.fftshift(fft_img2)

  rows, cols = shifted_img1.shape
  mask = np.zeros((rows, cols))
  center_row, center_col = rows // 2, cols // 2
  y, x = np.ogrid[:rows, :cols]
  mask = (x - center_col) ** 2 + (y - center_row) ** 2 <= r ** 2

  f_hybrid = mask * shifted_img1 + (1 - mask) * shifted_img2

  f_hybrid_back = np.fft.ifftshift(f_hybrid)

  hybrid_img = np.real(np.fft.ifft2(f_hybrid_back))

  return hybrid_img
```

Figure 12: Code for Creating a Hybrid Image

Figure 13: The output of Creating a Hybrid Image