INT3404E 20 - Image Processing: Homework 2

Doan Duc Kien - 21020207

This report presents the code implementation, explanation and results of two exercises in HW2. The first one deals with image filtering, particularly mean and median filter for noise reduction. The second one involves manipulating images in the frequency domain, a notable application of which being creating a hybrid image.

1 Exercise 1: Image Filter

1.1 Filters

```python
def padding_img(img, filter_size=3):
    return np.pad(img, pad_width=filter_size//2, mode='edge')
```

Listing 1: An implementation of padding_img function

Listing 1 shows the source code for padding_img function. The implementation only uses `np.pad` function, which takes in an input image, a padding width and padding mode. The padding width for convolution is $\lfloor filter\_size/2 \rfloor$ since the center of the kernel is placed at every pixel. The padding mode here is 'edge', which means replicate padding.

```python
def mean_filter(img, filter_size=3):
    result = np.zeros(img.shape)
    padded_img = padding_img(img, filter_size)

    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            region = padded_img[i:i+filter_size, j:j+filter_size]
            result[i, j] = np.mean(region)

    return result.astype(img.dtype)
```

Listing 2: An implementation of mean_filter function

```python
def median_filter(img, filter_size=3):
    result = np.zeros(img.shape)
    padded_img = padding_img(img, filter_size)

    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            region = padded_img[i:i+filter_size, j:j+filter_size]
            result[i, j] = np.median(region)

    return result.astype(img.dtype)
```

Listing 3: An implementation of median_filter function

As for the mean filter implementation, there are 3 steps:

1. Initialize the result as an empty array.

2. Pad the image using the padding_img function
3. Calculate the value of the resulting image at point (i, j). The value is computed by taking the mean of a region centered at (i, j) in the original unpadded image and having size $(filter\_size, filter\_size)$. However, after padding, (i, j) is mapped to $(i + pad\_width, j + pad\_width)$, which means the top coordinate of the region is (i, j). Therefore, we need to extract a region beginning at (i, j) in the padded image (`padded_img[i:i+filter_size, j:j+filter_size]` in python code)

Similarly, the implementation of median filter is presented in listing 3. The code only differs in the use of `np.median` instead of `np.mean`. The results for mean and median filter are shown in figure 1 and 2.

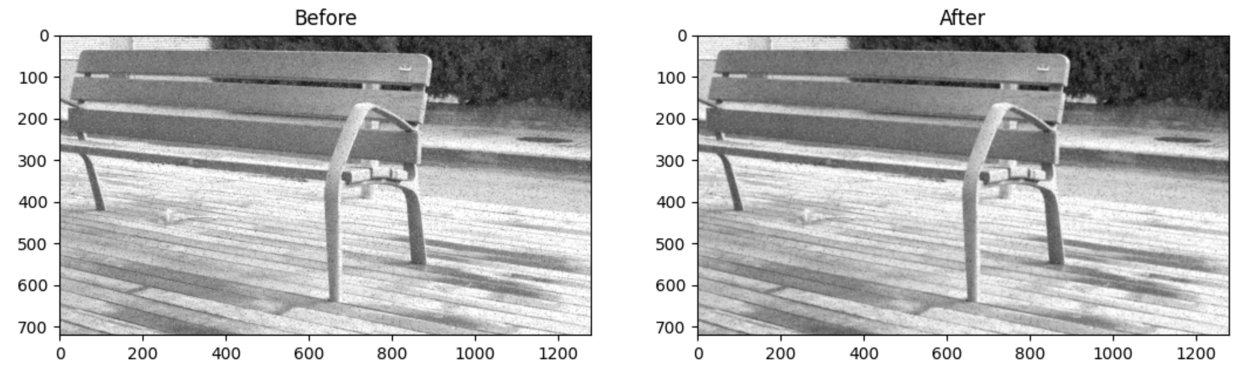

Figure 1: Result of mean_filter

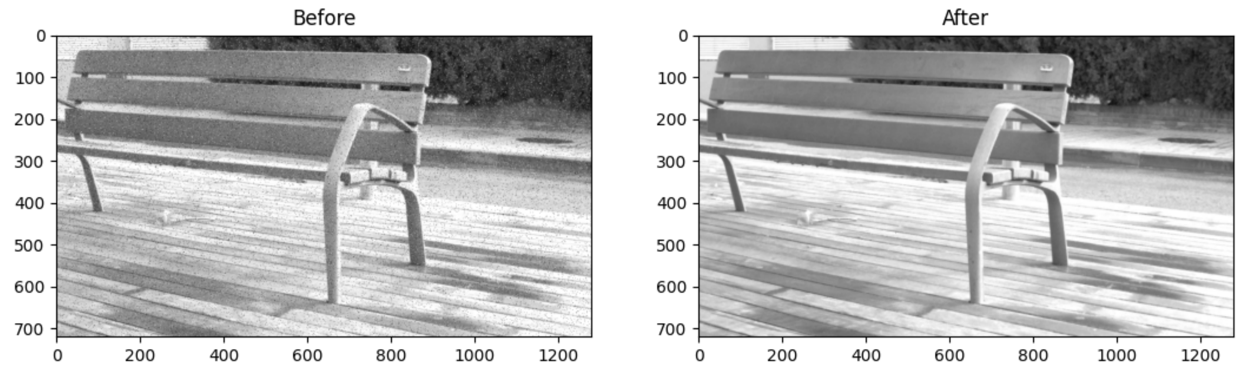

Figure 2: Result of median_filter

1.2 PSNR

```python
def psnr(gt_img, smooth_img):
    gt_img = gt_img.astype(np.float32)
    smooth_img = smooth_img.astype(np.float32)

    mse = np.mean((gt_img - smooth_img) ** 2)
    return 10 * np.log10(255**2/mse)
```

Listing 4: PSNR calculation

The Peak Signal-to-Noise Ration (PSNR) metric is computed according to the formula

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX^2}{MSE}\right)$$

The implementation is straight forward, the only precaution being casting image arrays to float32 to prevent overflow in subtraction. Table 1 shows the PSNR result for the two filters applied on the sample image. Based on this result, we should choose median filter for noise reduction in the provided image as it has a higher PSNR.

Filter	**PSNR**
Mean	26.202394060335106
Median	36.97746079407715

Table 1: The PSNR after applying the two filters

2 Exercise 2: Fourier Transform

2.1 1D and 2D Fourier Transform

```python
def DFT_slow(data):
  N = data.shape[0]
  F = np.fromfunction(lambda n, s: np.exp(-2j*np.pi*n*s/N), (N, N))
  return F @ data
```

Listing 5: 1D Fourier Transform

```python
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
```

Listing 6: 2D Fourier Transform

Listing 5 and 6 shows the source code of the 1D and 2D fourier transform. The 1D transform is performed by constructing a matrix F where $F_{ns} = \exp(\frac{1}{N}(-2j) \cdot \pi n \cdot s)$. This implementation uses `np.fromfunction`, which creates an array from values of a function of coordinates. The 2D transform is done by first taking Fourier transform of the rows, which is then transformed again but in the column direction. This corresponds to running **np.fft.fft** 2 times but in different axes. The result is shown in figure 3.

2.2 Frequency removal procedure

```python
def filter_frequency(orig_img, mask):
    f_img = np.fft.fft2(orig_img)
    f_img = np.fft.fftshift(f_img)
    f_img = f_img * mask
    f_img_shifted = np.fft.ifftshift(f_img)
    return np.abs(f_img), np.abs(np.fft.ifft2(f_img_shifted))
```

Listing 7: Frequency filter

The procedure for applying a frequency mask is shown in listing 7. The steps are:

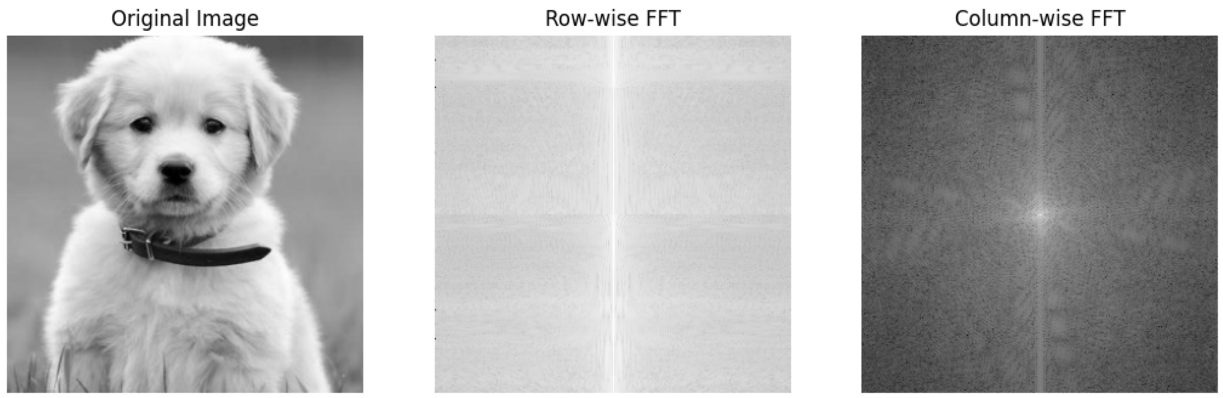

Figure 3: Result of DFT_2D

1. Transform using fft2
2. Shift frequency coefs to center using fftshift
3. Filter in frequency domain by multiplying the transformed array with the given mask
4. Shift frequency coefs back using ifftshift
5. Invert transform using ifft2

The result is illustrated in figure 4, in which the low frequencies are removed.

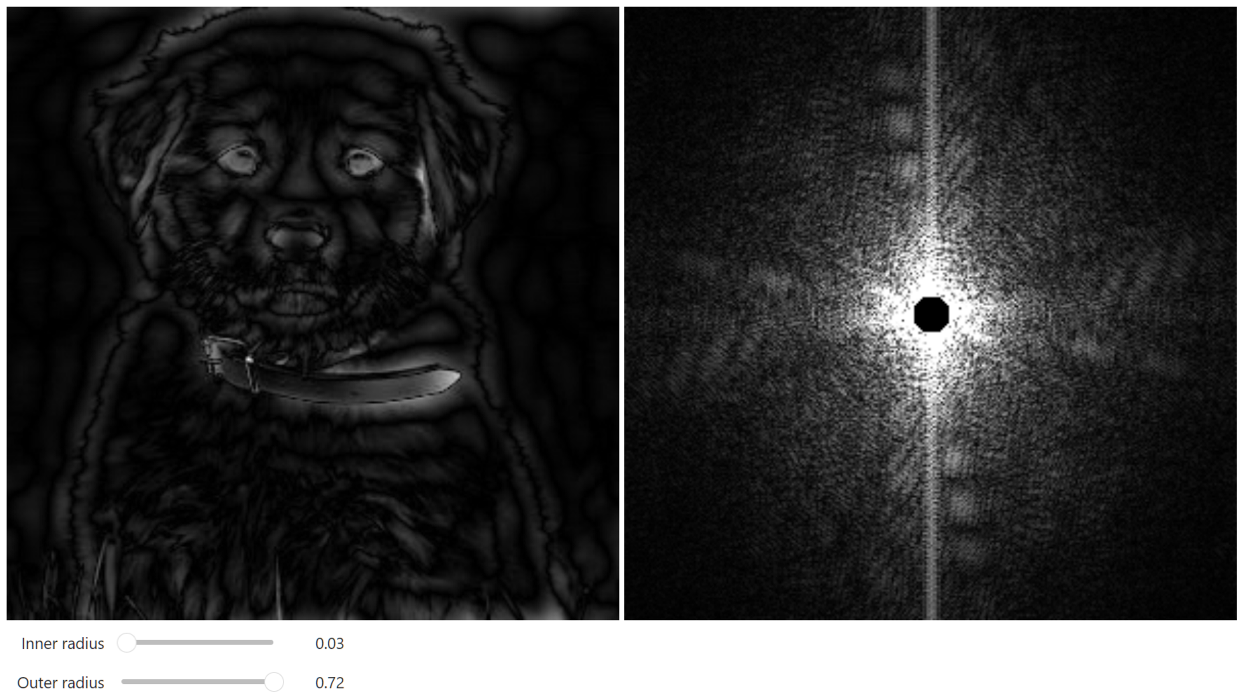
Inner radius 0.03
Outer radius 0.72

Figure 4: Result of filter_frequency

```python
def create_hybrid_img(img1, img2, r):
    fft1 = np.fft.fft2(img1)
    fft2 = np.fft.fft2(img2)
    fft1 = np.fft.fftshift(fft1)
    fft2 = np.fft.fftshift(fft2)
    cy, cx = img1.shape[0] // 2, img1.shape[1] // 2
    mask = np.fromfunction(lambda i, j: ((i-cy)**2 + (j-cx)**2) ** 0.5, img1.shape) <= r
    masked_img = (fft1 * mask + fft2 * (1 - mask))
    shifted_masked_img = np.fft.ifftshift(masked_img)
    hybrid_img = np.abs(np.fft.ifft2(shifted_masked_img))
    hybrid_img = 255*hybrid_img/(hybrid_img.max()+0.0001)
    return hybrid_img.astype(np.uint8)
```

Listing 8: Hybrid image creation procedure

2.3 Creating a hybrid image

As shown in listing 8, the steps to create a hybrid image are

1. Apply a Fourier Transform to the 2 images
2. Shift the 0 frequency to the center
3. Create a mask that has value 1 inside a circle with radius r and value 0 outside the circle.
4. Apply the mask to the first image and the inverse mask to the second image.
5. Inverse fourier transform
6. Normalize max pixel to 255

The result is shown in figure 5.

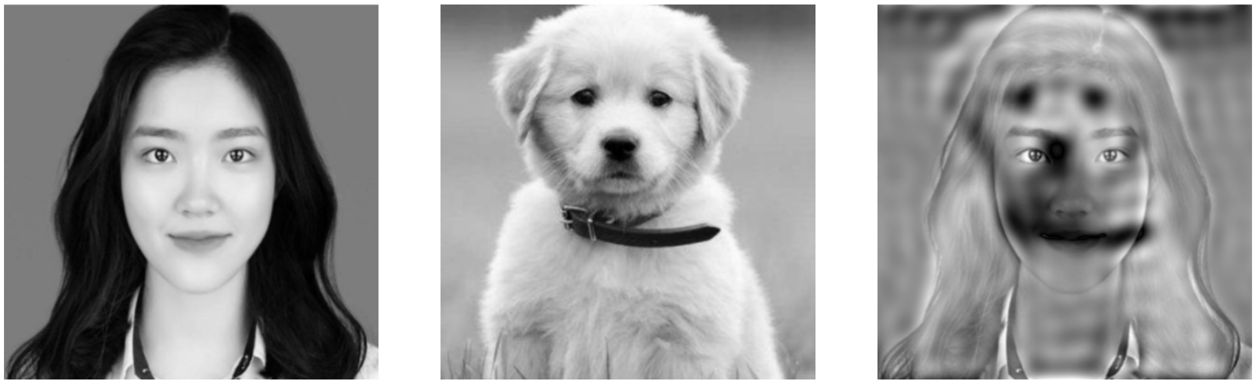
(a) 1st image (b) 2nd image (c) Hybrid image

Figure 5: The hybrid image is created by blending the high frequencies of the first image with the low frequencies of the second image