INT3404E 20 - Image Processing: Homeworks 2

Le Hong Duc

March 2024

1 Homework Objectives

Here are the detailed objectives of this homework:

1. To achieve a comprehensive understanding of how basic image filters operate
2. To gain a solid understanding of the Fourier Transform (FT) algorithm.

2 Details

2.1 Image Filtering

The exercise will help you understand basic image filters by manipulating box/mean and median filters. You will implement the first two problems in the provided Python script file (ex1.py), Specifically, you are required to:

- Implement one Replicate padding
- Implement the box/mean and median filters for removing noise
- Implement the evaluation metric.

Function implementation:

```python
def padding_img(img, filter_size=3):
    height, width = img.shape
    padding_size = filter_size // 2
    # Create a padded image with zeros
    padded_img = np.zeros(
        (height + 2 * padding_size, width + 2 * padding_size), dtype=img.dtype
    )

    # Copy the original image into the center of the padded image
    padded_img[
        padding_size : padding_size + height, padding_size : padding_size + width
    ] = img

    # Replicate padding for borders
    padded_img[:padding_size, padding_size : padding_size + width] = img[0, :]  # Top
    padded_img[padding_size + height :, padding_size : padding_size + width] = img[
        height - 1, :
    ]  # Bottom
    padded_img[:, :padding_size] = padded_img[
        :, padding_size : padding_size + 1
    ]  # Left
    padded_img[:, padding_size + width :] = padded_img[
        :, padding_size + width - 1 : padding_size + width
    ]  # Right
```

```python
def mean_filter(img, filter_size=3):
    padded_img = padding_img(img, filter_size)

    smoothed_img = np.zeros_like(img)
    height, width = padded_img.shape
    pad_size = filter_size // 2

    for x in range(pad_size, height - pad_size):
        for y in range(pad_size, width - pad_size):
            neighborhood = padded_img[x : x + filter_size, y : y + filter_size]
            median_value = np.mean(neighborhood)
            smoothed_img[x - pad_size, y - pad_size] = median_value

    return smoothed_img
```

```python
def median_filter(img, filter_size=3):
    padded_img = padding_img(img, filter_size)
    smoothed_img = np.zeros_like(img)
    height, width = padded_img.shape
    pad_size = filter_size // 2

    for x in range(pad_size, height - pad_size):
        for y in range(pad_size, width - pad_size):
            neighborhood = padded_img[x : x + filter_size, y : y + filter_size]
            median_value = np.median(neighborhood)
            smoothed_img[x - pad_size, y - pad_size] = median_value

    return smoothed_img
```

```python
def psnr(gt_img, smooth_img):
    gt_img = gt_img.astype(np.float64)
    smooth_img = smooth_img.astype(np.float64)

    mse = np.mean(np.square(gt_img - smooth_img))

    max_pixel = np.max(gt_img)

    psnr_score = 20 * np.log10(max_pixel) - 10 * np.log10(mse)

    return psnr_score
```

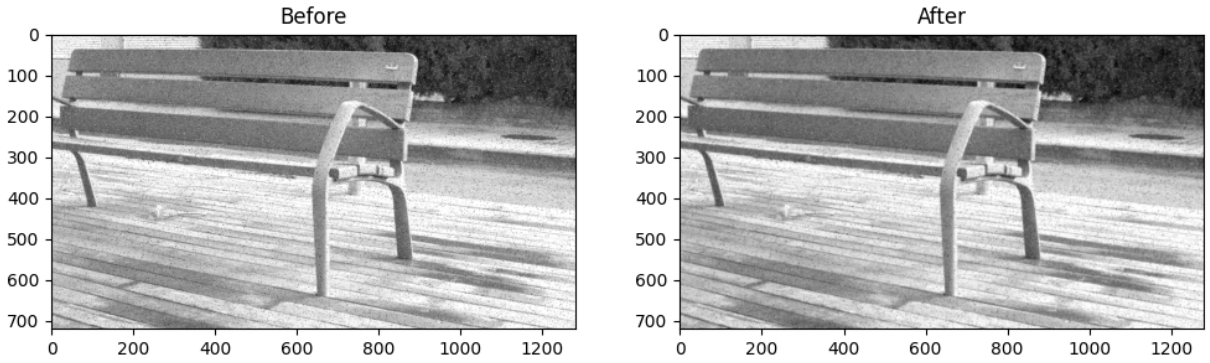

Figure 1: Mean filter result.

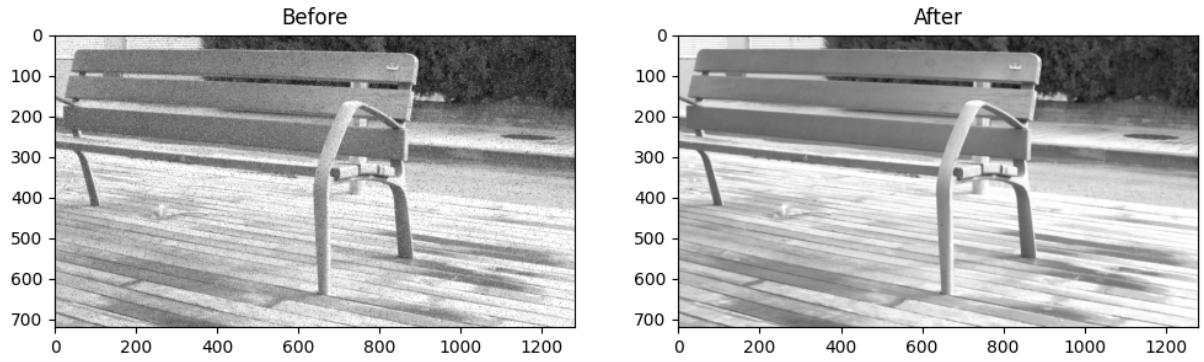

Figure 2: Median filter result.

Consider the PSNR metrics, which is approximately 24.54 for mean filter and 27.74 for median filter, the higher score filter means the better images after applying the filter. Therefore, the median filter should be chosen.

2.2 Fourier Transform

In this exercise, you will implement the Discrete Fourier Transform (DFT) algorithm from scratch. The goal is to familiarize yourself with the fundamental concepts and procedural steps involved in applying this algorithm. You will implement the first two problems in the provided Python script file (ex212.py), and the remaining two problems will be completed in the provided Jupyter notebook (ex223.ipynb)

2.2.1 1D Fourier Transform

Referencing the representation provided, implement a function named DFT slow to perform the Discrete Fourier Transform (DFT) on a one-dimensional signal.

```
def DFT_slow(data):
    N = len(data)

    DFT = np.zeros(N, dtype=np.complex_)

    for k in range(N):
        for n in range(N):
            DFT[k] += 1 / N * data[n] * np.exp(-2j * np.pi * k * n / N)

    return DFT
```

2.2.2 2D Fourier Transform

```
def DFT_2D(gray_img):
    row_fft = np.zeros_like(gray_img, dtype=np.complex_)
    for i in range(gray_img.shape[0]):
        row_fft[i, :] = DFT_slow(gray_img[i, :])

    row_col_fft = np.zeros_like(gray_img, dtype=np.complex_)
    for j in range(gray_img.shape[1]):
        row_col_fft[:, j] = DFT_slow(row_fft[:, j])

    return row_fft, row_col_fft
```

Result:

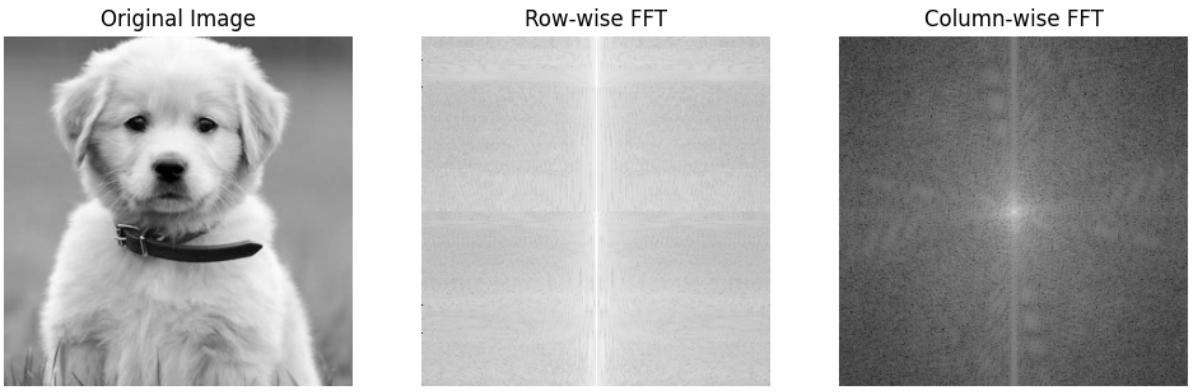

Figure 3: Expected output for 2D Fourier Transform exercise.

2.2.3 Frequency Removal Procedure

Implement the filter frequency function in the notebook as described.

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

Result:

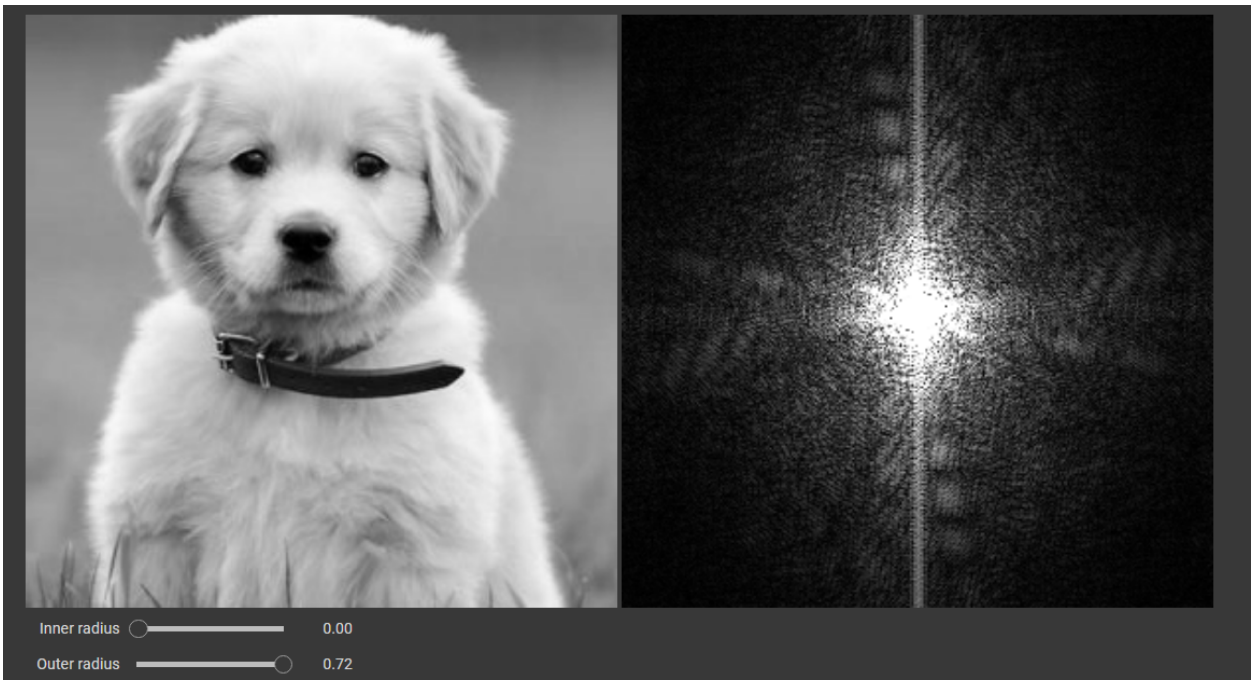

Figure 4: Expected Output for 2D Frequency Removal exercise.

2.2.4 Creating a Hybrid Image

Implement the function create hybrid img in the notebook as instructed.

```
def create_hybrid_img(img1, img2, r):
    f_img1 = fft2(img1)
    f_img2 = fft2(img2)

    f_img1_shifted = fftshift(f_img1)
    f_img2_shifted = fftshift(f_img2)

    H, W = img1.shape
    mask = np.zeros((H, W), dtype=np.float64)
    center = (H // 2, W // 2)
    for i in range(H):
        for j in range(W):
            if (i - center[0]) ** 2 + (j - center[1]) ** 2 <= r**2:
                mask[i, j] = 1

    f_hybrid_img = f_img1_shifted * mask + f_img2_shifted * (1 - mask)

    f_hybrid_img_shifted = ifftshift(f_hybrid_img)

    hybrid_img = np.abs(ifft2(f_hybrid_img_shifted))

    return hybrid_img
```

Result:

Figure 5: Expected output of exercise creating a Hybrid Image.