INT3404E 20 - Image Processing: Homeworks 2

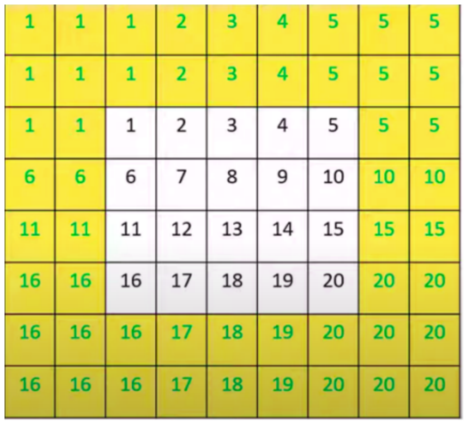
Tong Minh Tri - 21020249

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

(a) Implement the following functions: *padding_ img, mean_filter, median_filter.*



1	1	1	2	3	4	5	5	5
1	1	1	2	3	4	5	5	5
1	1	1	2	3	4	5	5	5
6	6	6	7	8	9	10	10	10
11	11	11	12	13	14	15	15	15
16	16	16	17	18	19	20	20	20
16	16	16	17	18	19	20	20	20
16	16	16	17	18	19	20	20	20

Figure 1: Replicate padding

Listing 1: Replicate padding

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

    height, width = img.shape
    half_filter_size = filter_size // 2
    padded_height = height + 2 * half_filter_size
    padded_width = width + 2 * half_filter_size

    padded_img = np.zeros((padded_height, padded_width), dtype=img.dtype)

    for i in range(height):
        for j in range(width):
            padded_img[i + half_filter_size, j + half_filter_size] = img[i, j]
    for i in range(half_filter_size):
        padded_img[i, half_filter_size:-half_filter_size] = img[0]
        padded_img[-i - 1, half_filter_size:-half_filter_size] = img[-1]
        padded_img[half_filter_size:-half_filter_size, i] = img[:, 0]
        padded_img[half_filter_size:-half_filter_size, -i - 1] = img[:, -1]

    return padded_img
```

Listing 2: Mean filter

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

    padded_img = padding_img(img, filter_size)
    height, width = padded_img.shape

    smoothed_img = np.zeros_like(img)

    for i in range(height - filter_size + 1):
        for j in range(width - filter_size + 1):
            window = padded_img[i:i + filter_size, j:j + filter_size]
            smoothed_img[i, j] = np.mean(window, axis=(0, 1))

    return smoothed_img
```

Listing 3: Median filter

```python
def median_filter(img, filter_size=3):

    padded_img = padding_img(img, filter_size)
    height, width = padded_img.shape

    smoothed_img = np.zeros_like(img)

    for i in range(height - filter_size + 1):
        for j in range(width - filter_size + 1):
            window = padded_img[i:i + filter_size, j:j + filter_size]
            smoothed_img[i, j] = np.median(window, axis=(0, 1))

    return smoothed_img
```

(b) Implement the Peak Signal-to-Noise Ratio (PSNR) metric

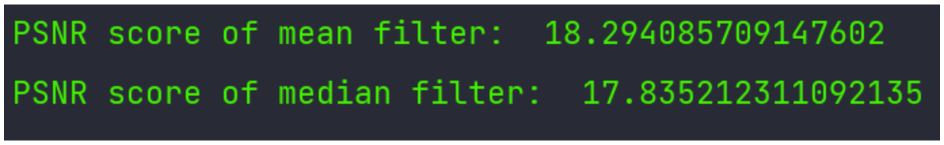
$$\text{PSNR} = 10 \cdot \log_{10}\left(\frac{\text{MAX}^2}{\text{MSE}}\right)$$

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.

Listing 4: PSNR metric

```python
def psnr(gt_img, smooth_img):

    gt_img = gt_img.astype(np.float64)
    smooth_img = smooth_img.astype(np.float64)

    mse = np.mean((gt_img - smooth_img) ** 2)

    max_pixel_value = 255

    psnr_score = 10 * np.log10((max_pixel_value ** 2) / mse)

    return psnr_score
```

(c) Because the mean filter gives higher PSNR score so we will choose mean filter for provided images



```
PSNR score of mean filter:  18.294085709147602
PSNR score of median filter:  17.835212311092135
```

Figure 2: PSNR score

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).

2.1 1D Fourier Transform

Implement a function named DFT_slow to perform the Discrete Fourier Transform (DFT) on a one-dimensional signal

Listing 5: DFT_slow

```python
def DFT_slow(data):

    N = len(data)
    n = np.arange(N)
    k = n.reshape((N, 1))

    # Exponential term in the DFT formula
    exp_term = np.exp(-2j * np.pi * k * n / N)

    # Calculate DFT using the formula
    DFT = np.dot(exp_term, data)
    return DFT
```

2.2 2D Fourier Transform

Implement the function DFT_2D. The procedure to simulate a 2D Fourier Transform is as follows:

1. Conducting a Fourier Transform on each row of the input 2D signal. This step transforms the signal along the horizontal axis.

2. Perform a Fourier Transform on each column of the previously obtained result

Listing 6: DFT_2D

```python
def DFT_2D(gray_img):

    H, W = gray_img.shape
    # Initialize arrays to store row-wise and column-wise FFTs
    row_fft = np.zeros((H, W), dtype=np.complex_)
    row_col_fft = np.zeros((H, W), dtype=np.complex_)

    # Compute row-wise FFT
    for i in range(H):
        row_fft[i, :] = np.fft.fft(gray_img[i, :])

    # Compute column-wise FFT
    for j in range(W):
        row_col_fft[:, j] = np.fft.fft(row_fft[:, j])

    return row_fft, row_col_fft
```

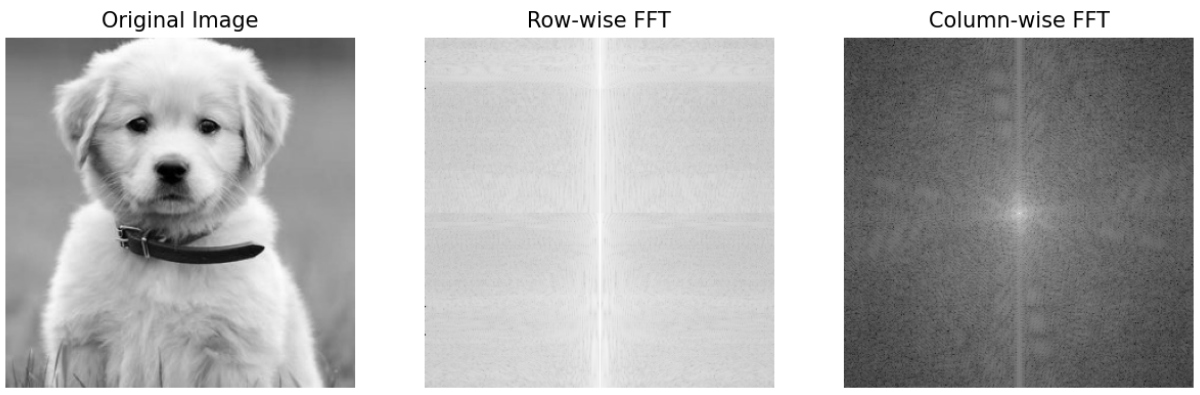


Figure 3: DFT_2D result

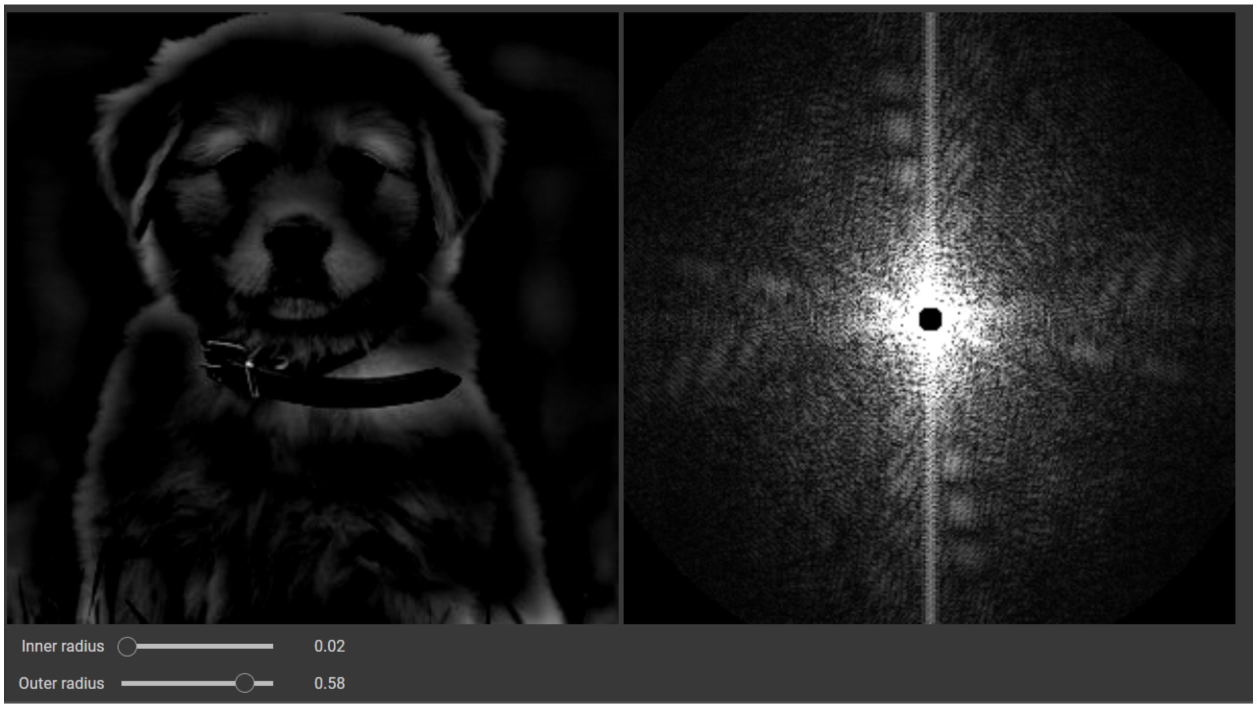
2.3 Frequency Removal Procedure

Follow these steps to manipulate frequencies in the image:

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

Listing 7: Filter frequency

```python
def filter_frequency(orig_img, mask):

    # Transform using fft2
    f_img = fft2(orig_img)

    # Shift frequency coefficients to center using fftshift
    f_img_shifted = fftshift(f_img)

    # Filter in frequency domain using the given mask
    f_img_filtered = f_img_shifted * mask

    # Shift frequency coefficients back using ifftshift
    f_img_filtered_shifted = ifftshift(f_img_filtered)

    # Invert transform using ifft2
    img = ifft2(f_img_filtered_shifted).real

    return f_img_filtered, img
```



Inner radius 0.02

Outer radius 0.58

Figure 4: Filter frequency result

2.4 Creating a Hybrid Image

To create a hybrid image, follow these steps:

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

Listing 8: Create hybrid image

```python
def create_hybrid_img(img1, img2, r):

    # Combine frequency of 2 images using the mask
    hybrid_img_fft = img1_fft_shifted * mask + img2_fft_shifted * (1 - mask)

    # Shift frequency coefficients back
    hybrid_img_fft_shifted = ifftshift(hybrid_img_fft)

    # Invert transform using inverse Fourier Transform
    hybrid_img = np.abs(ifft2(hybrid_img_fft_shifted))

    # Normalize the resulting image
    hybrid_img = cv2.normalize(hybrid_img, None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)

    return hybrid_img
```

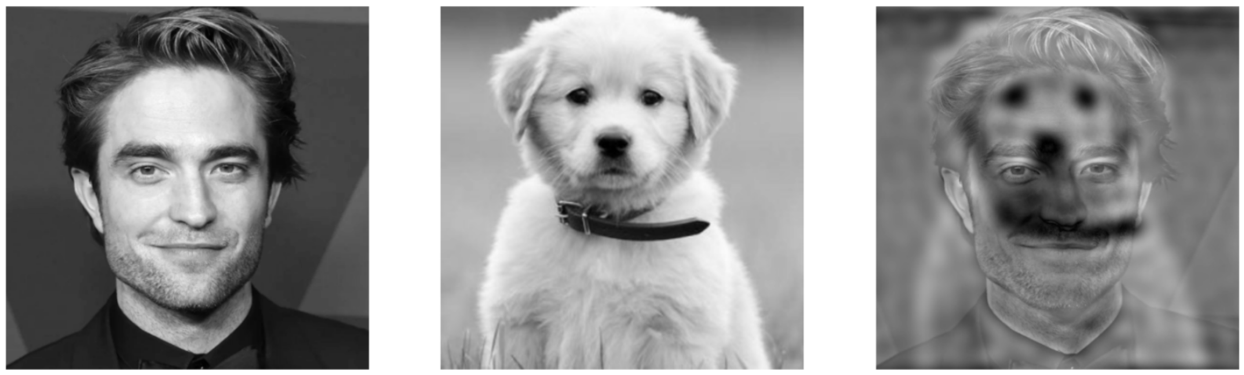


Figure 5: Hybrid image