INT3404E 20 - Image Processing: Homeworks 2

Duong Quang Minh - 21020219

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 Set up

It is assumed that you have set up a virtual environment. You can install the necessary packages using the command:

```
pip install -r requirements.txt
```

Additionally, you may choose to upload the notebook to Google Colaboratory for execution if you prefer not to run it on your local machine.

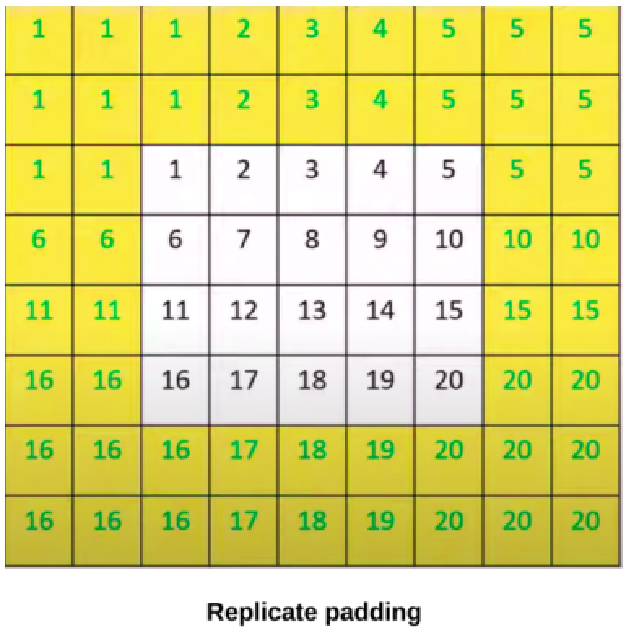
3 Details

3.1 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 in the supplied code file: padding_img, mean_filter, median_filter. Figure 1 illustrates replicate image padding used in the exercise



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

Replicate padding

Figure 1: Illustration of replicate padding

Sau đây là đoạn code của hàm mean filter. Trong đoạn code dưới đây, đầu vào của hàm là 1 hình ảnh, và kích thước filter để áp dụng trong ảnh; đầu ra là hình ảnh đã được áp dụng bộ lọc trung bình. Ta sẽ sử dụng 2 vòng lặp for để áp dụng bộ lọc median filter cho hình ảnh đã được áp dụng padding:

```python
def mean_filter(img, filter_size=3):
"""
Smoothing image with mean square filter with the size of filter_size.
Inputs:
    img: cv2 image: original image
    filter_size: int: size of square filter,
Return:
    smoothed_img: cv2 image: the smoothed image with mean filter.
"""
# Need to implement here
    height, width = img.shape[:2]
    smoothed_img = np.zeros((height, width), dtype=np.float32)

    padded_image = padding_img(img, filter_size=filter_size)

    pad_width = filter_size // 2
    for i in range(pad_width, height + pad_width):
        for j in range(pad_width, width + pad_width):
            matrix = padded_image[i - pad_width:i + pad_width + 1,
                                  j - pad_width:j + pad_width + 1]
            mean_value = np.mean(matrix)
            smoothed_img[i - pad_width, j - pad_width] = mean_value

    return np.array(smoothed_img, np.uint8)
```

Sau đây là đoạn code của hàm median filter: Trong đoạn code dưới đây, đầu vào của hàm là 1 hình ảnh, và kích thước filter để áp dụng trong ảnh; đầu ra là hình ảnh đã được áp dụng bộ lọc trung vị median. Ta sẽ sử dụng 2 vòng lặp for để áp dụng bộ lọc median cho hình ảnh đã được áp dụng replicate padding.

```
    def median_filter(img, filter_size=3):
    """
        Smoothing image with median square filter with the size of filter_size.
        Inputs:
            img: cv2 image: original image
            filter_size: int: size of square filter
        Return:
            smoothed_img: cv2 image: the smoothed image with median filter.
    """
    # Need to implement here
        height, width = img.shape[:2]
        smoothed_img = np.zeros((height, width), dtype=np.float32)

        padded_image = padding_img(img, filter_size=filter_size)
        pad_width = filter_size // 2

        for i in range(pad_width, height + pad_width):
            for j in range(pad_width, width + pad_width):
                 matrix = padded_image[i - pad_width:i + pad_width + 1,
                                       j - pad_width:j + pad_width + 1].flatten()
            median_value = np.median(sorted(matrix))
            smoothed_img[i - pad_width, j - pad_width] = median_value

        return np.array(smoothed_img, np.uint8)
```

Sau đây là đoạn code cho hàm replicate padding image:

```
    def padding_img(img, filter_size=3):
    """
    The surrogate function for the filter functions.
    Inputs:
        img: cv2 image: original image
        filter_size: int: size of square filter
    Return:
        padded_img: cv2 image: the padding image
    """
  # Need to implement here
    pad_width = filter_size // 2

    padded_image = np.pad(img, (pad_width, pad_width), mode='edge')

    return padded_image
```

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

$$PSNR = 10\log_{10} * \left(\frac{MAX^2}{MSE}\right) \tag{1}$$

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.

Sau đây là đoạn code cho hàm tính toán PSNR metric:

```
    def psnr(gt_img, smooth_img):
    """
        Calculate the PSNR metric
        Inputs:
            gt_img: cv2 image: groundtruth image
            smooth_img: cv2 image: smoothed image
        Outputs:
            psnr_score: PSNR score
    """
        # Need to implement here
        gt_img = np.array(gt_img, dtype=np.float32)
        smooth_img = np.array(smooth_img, dtype=np.float32)
        square_error = np.square(gt_img - smooth_img)
        mse_score = np.mean(square_error)
        max_value = 255.0
        psnr_score = 10 * math.log10((max_value ** 2) / mse_score)

        return psnr_score
```

(c) Considering the PSNR metrics, which filter should we choose between the mean and median filters for provided images?
Dựa trên PSNR metrics để đánh giá kết quả của các bộ lọc trung bình và trung vị sau khi áp dụng lên ảnh gốc. Kết quả thu được cho thấy điểm PSNR cho bộ lọc trung bình là 18.294 và kết quả PSNR cho bộ lọc trung vị là 17.835. Ta thấy được bộ lọc trung bình sẽ cho kết quả PSNR cao hơn một chút so với bộ lọc trung vị. Tuy nhiên hình ảnh được áp dụng median filter thu được ta có cảm giác nó mượt hơn so với mean filter. Bởi vậy, nếu dựa trên kết quả PSNR ta nên chọn bộ lọc trung bình cho ảnh được cung cấp.

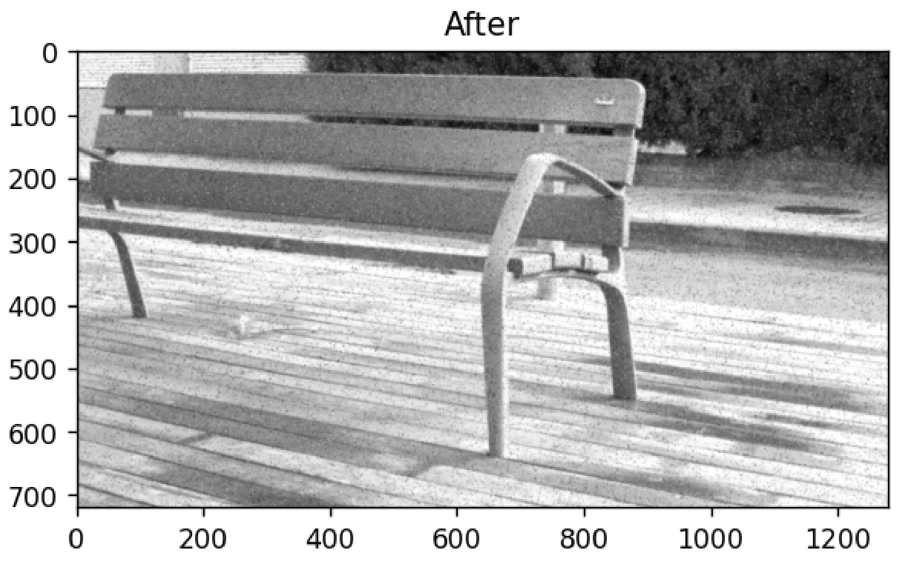


Figure 2: Hình ảnh sau khi áp dụng bộ lọc trung bình

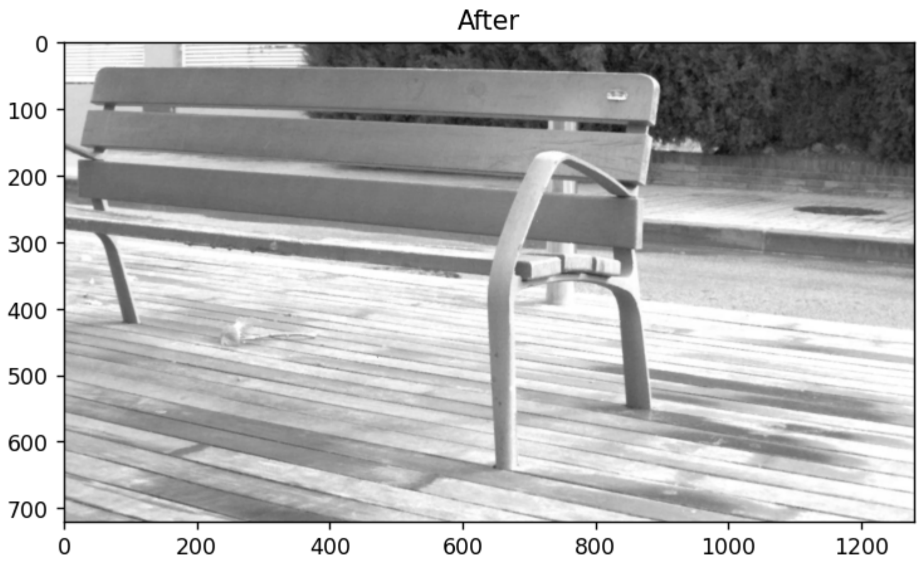


Figure 3: Hình ảnh sau khi áp dụng bộ lọc trung vị

3.2 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).

3.2.1 1D Fourier Transform

Implement a function named DFT_slow to perform the Discrete Fourier Transform (DFT) on a one-dimensional signal. Sau đây là đoạn code cho hàm Dicrete Fourier Transform (DFT):

```python
def DFT_slow(data):
  """
  Implement the discrete Fourier Transform for a 1D signal
  params:
    data: Nx1: (N, ): 1D numpy array
  returns:
    DFT: Nx1: 1D numpy array
  """
  # You need to implement the DFT here
  len_data = len(data)
  array = np.zeros(len_data, dtype=np.complex128)

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

  return array
```

3.2.2 2D Fourier Transform

Along with the scipy.fft, Numpy.fft provides functions related to the Fourier transform.

1. fft: Compute the one-dimensional discrete Fourier Transform.
2. ifft: Compute the one-dimensional inverse discrete Fourier Transform.

3. fft2: Compute the 2-dimensional discrete Fourier Transform.
4. ifft2: Compute the 2-dimensional inverse discrete Fourier Transform.
5. fftn: Compute the N-dimensional discrete Fourier Transform.
6. fftn: Compute the N-dimensional inverse discrete Fourier Transform.

In this exercise, we aim to manually simulate the operation of the np.fft.fft2 function, which performs a twodimensional Fourier Transform on a 2D signal. To achieve this, we will solely utilize the np.fft.fft function, designed for one-dimensional Fourier Transforms. Your task is to implement the function DFT_2D within the provided code. 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.

Sau đây là đoạn code cho hàm DFT_2D được mô tả ở trên:

```
def DFT_2D(gray_img):
"""
Implement the 2D Discrete Fourier Transform
Note that: dtype of the output should be complex_
params:
    gray_img: (H, W): 2D numpy array

returns:
    row_fft: (H, W): 2D numpy array that contains the row-wise FFT of the input image
    row_col_fft: (H, W): 2D numpy array that contains the column-wise FFT of the input image
"""
    # You need to implement the DFT here
    height, width = gray_img.shape

    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
```

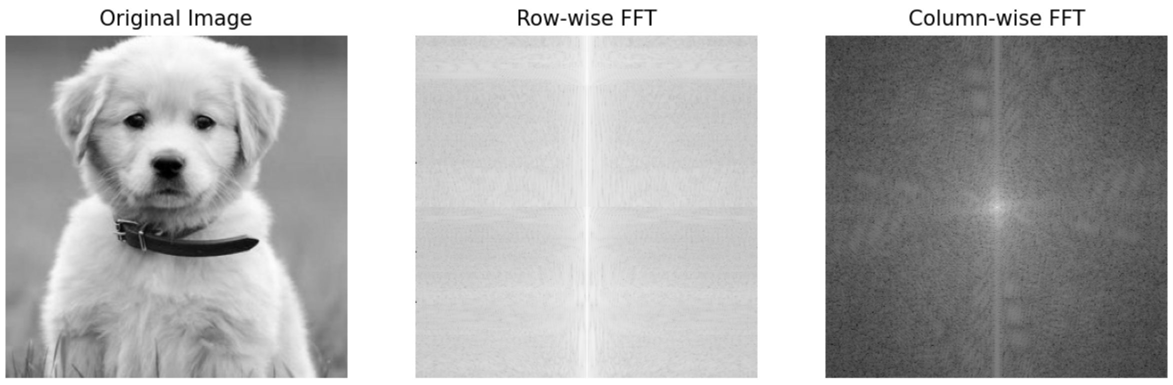


Figure 4: Kết quả cho hàm 2D Fourier Transform

3.2.3 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

Sau đây là hàm filter frequency được mô tả ở trên:

```python
def filter_frequency(orig_img, mask):
    """
    You need to remove frequency based on the given mask.
    Params:
      orig_img: numpy image
      mask: same shape with orig_img indicating which frequency hold or remove
    Output:
      f_img: frequency image after applying mask
      img: image after applying mask
    """
    # You need to implement this function
    f_img = np.fft.fft2(orig_img)

    f_img_shifted = np.fft.fftshift(f_img)

    f_img_filtered_shifted = f_img_shifted * mask

    f_img_filtered = np.fft.ifftshift(f_img_filtered_shifted)

    img = np.abs(np.fft.ifft2(f_img_filtered))

    return np.abs(f_img_filtered_shifted), img
```

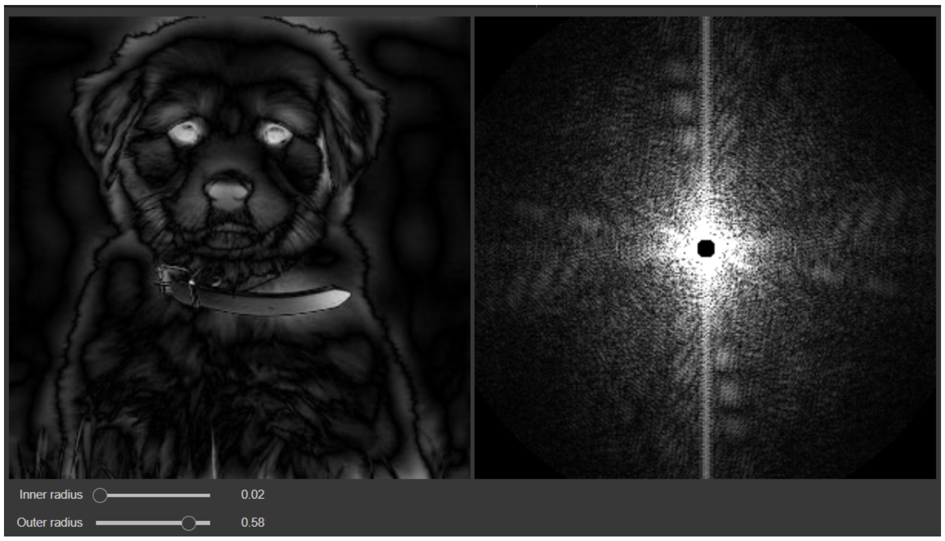


Figure 5: Kết quả cho hàm 2D Frequency Removal

3.2.4 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

Sau đây là hàm create_hybrid_img như được mô tả ở trên:

```python
def create_hybrid_img(img1, img2, r):
  """
  Create hydrid image
  Params:
    img1: numpy image 1
    img2: numpy image 2
    r: radius that defines the filled circle of frequency of image 1.
  """
  f_img1 = np.fft.fftshift(np.fft.fft2(img1))
  f_img2 = np.fft.fftshift(np.fft.fft2(img2))

  rows, cols = img1.shape[:2]
  crow, ccol = rows // 2, cols // 2
  mask = np.zeros((rows, cols), dtype=np.uint8)
  mask[crow-r:crow+r, ccol-r:ccol+r] = 1

  f_img1_filtered = f_img1 * mask
  f_img2_filtered = f_img2 * (1 - mask)

  f_hybrid = f_img1_filtered + f_img2_filtered

  hybrid_img = np.abs(np.fft.ifft2(np.fft.ifftshift(f_hybrid)))

  hybrid_img = np.clip(hybrid_img, 0, 255).astype(np.uint8)

  return hybrid_img
```

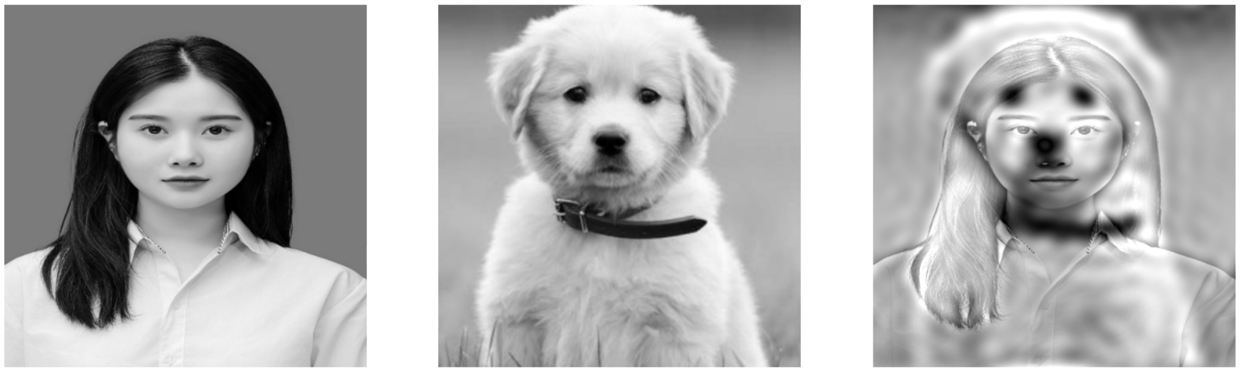


Figure 6: Kết quả của hàm Hybrid image

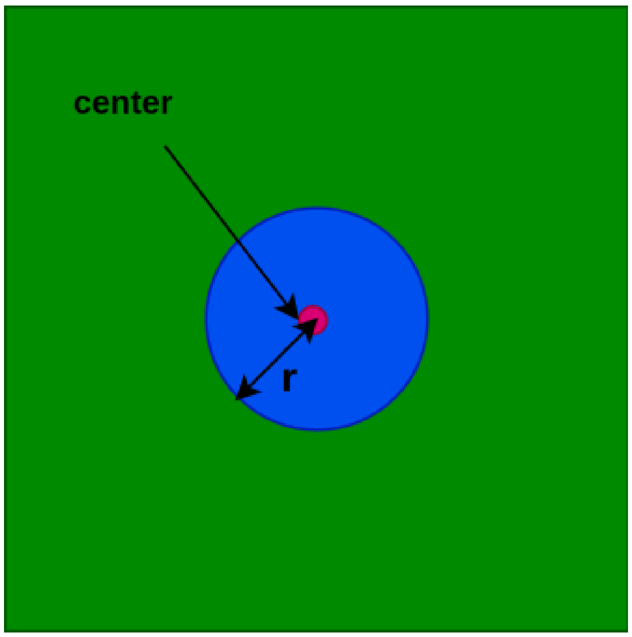


Figure 7: Mask Creation: The blue area represents the frequencies from the first image, while the remaining area pertains to the frequencies of the second image.