INT3404E 20 - Image Processing: Homeworks 2

Nguyen Thi Thu Trang - 22028254

1 Các bộ lọc ảnh(Image Filtering)

1.1 Hàm padding_img

Hàm $padding\_img(img, filter\_size = 3)$ sử dụng thư viện numpy và công cụ $np.pad()$ để thêm viền(padding) vào hình ảnh ban đầu. Việc này giúp đảm bảo rằng khi áp dụng bộ lọc(filter) lên hình ảnh, kích thước của hình ảnh kết quả vẫn giữ nguyên. Trong trường hợp này, viền được sao chép từ các pixel ở biên của hình ảnh.

```
def padding_img(img, filter_size=3):
    """
    The surrogate function for the filter functions.
    The goal of the function: replicate padding the image such that when applying the kernel
    with the size of filter_size, the padded image will be the same size as the original image.
    WARNING: Do not use the exterior functions from available libraries such as OpenCV,
    scikit-image, etc. Just do from scratch using function from the numpy library or functions
    in pure Python.
    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_size = filter_size // 2
    padded_img = np.pad(img, pad_size, mode='edge')
    return padded_img
```

1.2 Hàm mean_filter

Hàm $mean\_filter(img, filter\_size = 3)$ sử dụng thư viện numpy thực hiện việc làm mịn (smoothing) hình ảnh bằng bộ lọc trung bình.$numpy.zero\_like()$ được sử dụng để tạo một ma trận trống có cùng kích thước như hình ảnh ban đầu, và $numpy.mean()$ được sử dụng để tính giá trị trung bình của các pixel trong cửa sổ lọc.Tức là hàm $mean\_filter$ trượt một cửa sổ hình vuông qua hình ảnh và thay thế mỗi pixel bằng giá trị trung bình của các pixel láng giềng trong cửa sổ. Trong trường hợp này, việc padding được sử dụng để đảm bảo rằng kích thước của hình ảnh sau khi làm mịn vẫn giống như kích thước của hình ảnh ban đầu.

```
def mean_filter(img, filter_size=3):
    """
    Smoothing image with mean square filter with the size of filter_size. Use replicate padding
    for the image.
    WARNING: Do not use the exterior functions from available libraries such as OpenCV,
    scikit-image, etc. Just do from scratch using function from the numpy library or functions in
    pure Python.
    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
    padded_img = padding_img(img, filter_size)
```

```
    smoothed_img = np.zeros_like(img)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            neighborhood = padded_img[i:i+filter_size, j:j+filter_size]
            smoothed_img[i, j] = np.mean(neighborhood)
    return smoothed_img
```

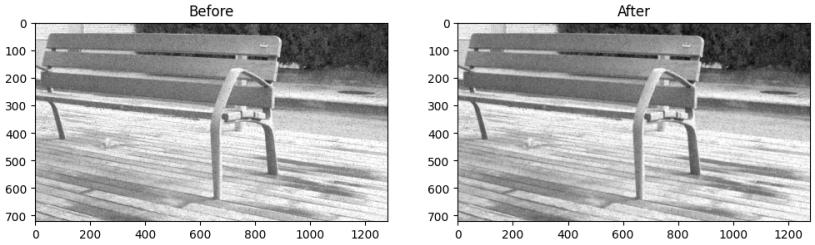

Figure 1: Ảnh lọc bởi bộ lọc trung bình(mean filter)

1.3 Hàm median_filter

Hàm $median\_filter(img, filter\_size = 3)$ sử dụng thư viện numpy thực hiện việc làm mịn (smoothing) hình ảnh bằng bộ lọc trung vị.$numpy.median()$ được sử dụng để tính giá trị trung vị của các pixel trong cửa sổ lọc. Tức là hàm $median\_filter$ thay thế mỗi pixel bằng giá trị trung vị của các pixel láng giềng trong cửa sổ hình vuông trượt qua hình ảnh. Tương tự như $mean\_filter()$, việc padding được sử dụng để đảm bảo kích thước của hình ảnh sau khi làm mịn không thay đổi.

```
def median_filter(img, filter_size=3):
    """
        Smoothing image with median square filter with the size of filter_size. Use replicate
        padding for the image.
        WARNING: Do not use the exterior functions from available libraries such as OpenCV,
        scikit-image, etc. Just do from scratch using function from the numpy library or functions
        in pure Python.
        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
    padded_img = padding_img(img, filter_size)
    smoothed_img = np.zeros_like(img)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            neighborhood = padded_img[i:i+filter_size, j:j+filter_size]
            smoothed_img[i, j] = np.median(neighborhood)
```

```
    return smoothed_img
```

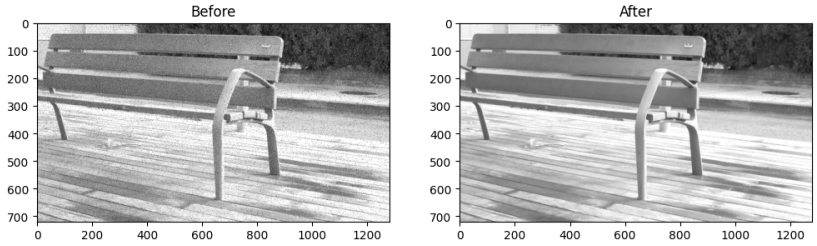

Figure 2: Ảnh lọc bởi bộ lọc trung vị(median filter)

1.4 Hàm tính PSRN

Công thức tính tỷ số tín hiệu cực đại trên nhiễu ($PeakSignal - to - NoiseRatio$ hay $PSNR$):

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

Hàm $psnr(gt\_img, smooth\_img)$ dựa vào công thức bên trên tính toán chỉ số PSNR giữa hai hình ảnh: hình ảnh gốc và hình ảnh sau khi làm mịn. PSNR là một chỉ số đánh giá chất lượng của hình ảnh sau khi áp dụng các phép làm mịn. Để tính toán PSNR, hàm sử dụng thư viện numpy để tính độ lỗi trung bình bình phương (MSE) giữa hai hình ảnh, thông qua hàm $numpy.mean()$. Sau đó, nó sử dụng $math.log10()$ để tính toán hàm logarit cơ số 10. Trong trường hợp này, giá trị MAX thường là giá trị tối đa có thể của một pixel (thông thường là 255 cho hình ảnh 8-bit), và MSE là độ lỗi trung bình bình phương giữa hai hình ảnh.

```
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
    mse = np.mean((gt_img - smooth_img) ** 2)
    max_pixel = 255.0
    psnr_score = 10 * math.log10((max_pixel ** 2) / mse)
    return psnr_score
```

Kết quả thu được sau khi tính PSNR của bộ lọc trung bình(mean_filter) và bộ lọc trung vị(median_filter) như sau:

- PSNR score of mean filter: 31.60889963499979

- PSNR score of median filter: 37.119578300855245

Dựa vào kết quả thu được, chúng ta có thể thấy bộ lọc trung vị có điểm PSNR cao hơn đáng kể so với bộ lọc trung bình.
Giá trị PSNR càng cao thì chất lượng hình ảnh được lọc càng tốt so với hình ảnh bị nhiễu ban đầu.
=> Giá trị PSNR của bộ lọc trung vị cao hơn cho thấy ít nhiễu hơn và bảo toàn chi tiết hình ảnh tốt hơn bộ lọc trung bình.
Do đó, trong trường hợp này, chúng ta nên chọn bộ lọc trung vị thay vì bộ lọc trung bình cho hình ảnh được cung cấp vì nó tạo ra kết quả mượt mà hơn và ít nhiễu hơn.

2 Fourier Transform

2.1 1D Fourier Transform

Hàm $DFT\_slow(data)$ thực hiện biến đổi Fourier rời rạc (DFT) cho một tín hiệu một chiều (1D). Trong phạm vi của nó, nó sử dụng một phương pháp chậm và không hiệu quả tính toán, được gọi là phương pháp "*slow*" để tính toán DFT của một tín hiệu 1D. Hàm này sử dụng một vòng lặp lồng nhau để tính toán từng phần tử của DFT dựa trên công thức DFT.
Cụ thể hoạt động của hàm DFT_slow như sau:

- Hàm nhận đầu vào là một mảng 1D data chứa tín hiệu cần biến đổi Fourier.
- Tính toán chiều dài tín hiệu: Lấy chiều dài của mảng data để biết số lượng mẫu trong tín hiệu.
- Tạo mảng DFT trống: Tạo một mảng DFT cùng kích thước với data để lưu trữ kết quả của DFT, với kiểu dữ liệu là số phức.
- Tính toán DFT: Sử dụng hai vòng lặp để tính toán từng phần tử của DFT theo công thức của biến đổi Fourier rời rạc. Vòng lặp bên ngoài (vòng lặp for s) lặp qua tất cả các tần số (s) trong miền tần số, và vòng lặp bên trong (vòng lặp for n) tính tổng các mẫu trong tín hiệu nhân với phần tử của DFT.
- Trả về mảng DFT chứa kết quả của phép biến đổi Fourier cho tín hiệu đầu vào.

```
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
    N = len(data)
    DFT = np.zeros(N, dtype=np.complex_)
    for s in range(N):
            for n in range(N):
                DFT[s] += data[n] * np.exp(-1j * 2 * np.pi * s * n / N)
    return DFT
```

2.2 2D Fourier Transform

Hàm $DFT\_2D(gray\_img)$ thực hiện biến đổi Fourier rời rạc 2 chiều (2D) cho một hình ảnh xám. Đầu vào của hàm này là một mảng numpy 2D biểu diễn hình ảnh xám. Trong phạm vi của nó, hàm này gọi DFT_slow để tính toán biến đổi Fourier của từng hàng và từng cột của hình ảnh. Kết quả cuối cùng là hai mảng 2D: một mảng chứa DFT của từng hàng (row-wise FFT) và một mảng chứa DFT của từng cột (column-wise FFT) của hình ảnh.

Cụ thể hoạt động của hàm DFT_2D như sau:

- Hàm này nhận đầu vào là một hình ảnh xám 2D (gray_img).
- Tạo mảng FFT trống: Tạo hai mảng trống row_fft và row_col_fft, cùng kích thước với hình ảnh đầu vào gray_img, để lưu trữ kết quả của biến đổi Fourier. Cả hai mảng đều có kiểu dữ liệu là số phức.
- Row-wise FFT: Dùng một vòng lặp để duyệt qua từng hàng của hình ảnh. Đối với mỗi hàng, gọi hàm DFT_slow để tính toán DFT của hàng đó và lưu vào row_fft.
- Column-wise FFT: Dùng một vòng lặp để duyệt qua từng cột của hình ảnh. Đối với mỗi cột, gọi hàm DFT_slow để tính toán DFT của cột đó từ row_fft và lưu vào row_col_fft.
- Trả về hai mảng row_fft và row_col_fft chứa kết quả của phép biến đổi Fourier theo hàng và cột của hình ảnh đầu vào.

```python
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
    H, W = gray_img.shape
    row_fft = np.zeros((H, W), dtype=np.complex_)
    row_col_fft = np.zeros((H, W), dtype=np.complex_)

    # Row-wise FFT
    for i in range(H):
        row_fft[i, :] = DFT_slow(gray_img[i, :])

    # Column-wise FFT
    for j in range(W):
        row_col_fft[:, j] = DFT_slow(row_fft[:, j])

    return row_fft, row_col_fft
```

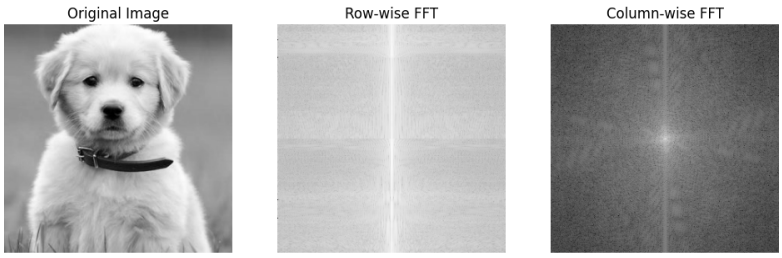

Figure 3: Ảnh kết quả triển khai 2D-Fourier Transform với ảnh

2.3 Hàm lọc tần số (Frequency Removal Procedure)

Hàm $filter\_frequency(orig\_img, mask)$ dùng để loại bỏ các tần số dựa trên mặt nạ được cung cấp.

- Đầu tiên, hình ảnh gốc được chuyển đổi sang miền tần số bằng phép biến đổi Fourier.
- Sau đó, các hệ số tần số được dịch chuyển để đặt tần số ở trung tâm và mặt nạ được áp dụng. Mặt nạ xác định các tần số nào sẽ được giữ lại hoặc loại bỏ khỏi hình ảnh.
- Tiếp theo, các hệ số tần số được dịch chuyển lại và biến đổi ngược để nhận được hình ảnh đã lọc.
- Kết quả là hình ảnh được lọc và mảng chứa các hệ số tần số sau khi áp dụng mặt nạ.

  Các bước này sử dụng các hàm và công cụ trong thư viện NumPy như np.fft.fft2, np.fft.fftshift, np.fft.ifftshift, và np.abs.

```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
    """
    # Apply FFT to the original image
    f_img = np.fft.fft2(orig_img)

    # Shift frequency coefficients to center
    f_img_shifted = np.fft.fftshift(f_img)

    # Apply mask in the frequency domain
    f_img_filtered = f_img_shifted * mask

    # Shift frequency coefficients back
    f_img_filtered_shifted = np.fft.ifftshift(f_img_filtered)
```

```
    # Invert transform to get the filtered image
    img = np.abs(np.fft.ifft2(f_img_filtered_shifted))

    f_img_filtered = np.abs(f_img_filtered)
    return f_img_filtered, img
```

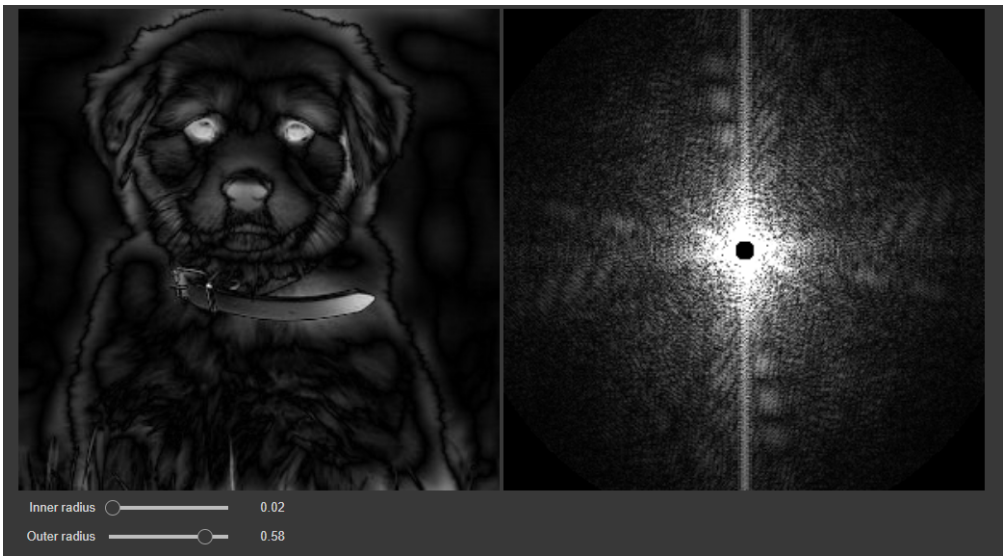

Figure 4: Ảnh loại bỏ tần số 2D

2.4 Tạo Hybrid Image

Hàm $create\_hybrid\_img(img1, img2, r)$ được sử dụng để tạo ra một hình ảnh kết hợp từ hai hình ảnh đầu vào.

- Đầu tiên, hai hình ảnh đó được chuyển đổi sang miền tần số bằng phép biến đổi Fourier.
- Sau đó, một mặt nạ được tạo ra dựa trên bán kính (r) đã được chỉ định, với mục đích chọn lọc các tần số từ cả hai hình ảnh.
- Cuối cùng, các hệ số tần số của hai hình ảnh được kết hợp lại bằng cách sử dụng mặt nạ, và sau đó chuyển đổi ngược trở lại miền không gian để tạo ra hình ảnh kết hợp.
- Kết quả là một hình ảnh kết hợp chứa thông tin từ cả hai hình ảnh đầu vào, với sự kết hợp được điều chỉnh dựa trên mặt nạ và bán kính đã được chỉ định.

  Các công cụ và hàm chính trong thuật toán bao gồm biến đổi Fourier (np.fft.fft2, np.fft.fftshift, np.fft.ifftshift, np.fft.ifft2) và tạo lưới điểm (np.linspace, np.meshgrid).

```
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. Refer to the homework title
        to know more.
    """
    # You need to implement the function
```

```
    # Apply FFT to the images
    f_img1 = np.fft.fft2(img1)
    f_img2 = np.fft.fft2(img2)

    # Shift frequency coefs to center
    f_img1_shifted = np.fft.fftshift(f_img1)
    f_img2_shifted = np.fft.fftshift(f_img2)

    # Create a mask based on the given radius (r)
    H, W = img1.shape
    x = np.linspace(-W//2, W//2, W)
    y = np.linspace(-H//2, H//2, H)
    xv, yv = np.meshgrid(x, y)
    mask = (xv**2 + yv**2) < r**2

    # Combine frequency of 2 images using the mask
    f_hybrid = f_img1_shifted * mask + f_img2_shifted * (1 - mask)

    # Shift frequency coefs back
    f_hybrid_shifted = np.fft.ifftshift(f_hybrid)

    # Invert transform to get the hybrid image
    hybrid_img = np.fft.ifft2(f_hybrid_shifted).real

    return hybrid_img
```

Figure 5: Ảnh Hybrid