INT3404E 20 - Image Processing: Homework 2

Cao Thị Minh Tâm

1 Ex1 - Image Filtering

1.1 Mã nguồn

1.1.1 padding_img() function

Listing 1: Code of padding_img() function

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

- Thuật toán:
  - Sử dụng hàm np.pad từ thư viện NumPy để thêm các hàng và cột vào xung quanh ảnh ban đầu. Tham số pad_size được tính bằng cách chia filter_size cho 2, và sau đó np.pad được sử dụng để thêm các hàng và cột tương ứng.
  - Chế độ (mode) được đặt là 'edge', điều này có nghĩa là các giá trị pixel ở biên của ảnh sẽ được sao chép từ các hàng và cột cuối cùng của ảnh. Điều này giúp giữ nguyên các biên của ảnh và tránh tạo ra hiệu ứng không mong muốn khi đệm.

1.1.2 mean_filter() function

Listing 2: Code of mean_filter() function

```
def mean_filter(img, filter_size=3):
    # Perform padding
    padded_img = padding_img(img, filter_size)

    # Get image shape
    height, width = img.shape

    # Initialize smoothed image
    smoothed_img = np.zeros_like(img)

    # Apply mean filter
    for i in range(height):
        for j in range(width):
            # Extract neighborhood
            neighborhood = padded_img[i:i + filter_size, j:j + filter_size]
            # Apply mean filter
            smoothed_img[i, j] = np.mean(neighborhood)

    return smoothed_img
```

- Thuật toán:
  - Hình ảnh đầu vào được thêm viền bằng cách sử dụng hàm padding_img, đảm bảo các pixel ở mép của hình ảnh cũng có thể được xử lý.

- Một ma trận có kích thước giống với hình ảnh đầu vào được khởi tạo để chứa hình ảnh được làm mịn.
- Hàm duyệt qua từng pixel của hình ảnh, và tại mỗi vị trí, trích xuất vùng lân cận xung quanh pixel đó bằng cách sử dụng kích thước bộ lọc.
- Bộ lọc trung bình được áp dụng bằng cách tính trung bình của các giá trị pixel trong vùng lân cận.
- Giá trị trung bình được gán cho pixel tương ứng trong hình ảnh được làm mịn.

1.1.3 median_filter() function

Listing 3: Code of median_filter() function

```
def median_filter(img, filter_size=3):
    # Perform padding
    padded_img = padding_img(img, filter_size)

    # Get image shape
    height, width = img.shape

    # Initialize smoothed image
    smoothed_img = np.zeros_like(img)

    # Apply median filter
    for i in range(height):
        for j in range(width):
            # Extract neighborhood
            neighborhood = padded_img[i:i + filter_size, j:j + filter_size]
            # Apply median filter
            smoothed_img[i, j] = np.median(neighborhood)

    return smoothed_img
```

- Thuật toán:
  - Hình ảnh đầu vào được thêm viền bằng cách sử dụng hàm padding_img, đảm bảo các pixel ở mép của hình ảnh cũng có thể được xử lý.
  - Một ma trận có kích thước giống với hình ảnh đầu vào được khởi tạo để chứa hình ảnh được làm mịn.
  - Hàm duyệt qua từng pixel của hình ảnh, và tại mỗi vị trí, trích xuất vùng lân cận xung quanh pixel đó bằng cách sử dụng kích thước bộ lọc.
  - Bộ lọc trung vị được áp dụng bằng cách tính giá trị trung vị của các giá trị pixel trong vùng lân cận.
  - Giá trị trung vị được gán cho pixel tương ứng trong hình ảnh được làm mịn.

1.1.4 psnr() function

Listing 4: Code of psnr() function

```
def psnr(gt_img, smooth_img):
    # Calculate Mean Square Error (MSE)
    mse = np.mean((gt_img - smooth_img) ** 2)

    # Maximum possible pixel value
    max_pixel = 255
```

```
    # Calculate PSNR
    psnr_score = 10 * math.log10((max_pixel ** 2) / mse)

    return psnr_score
```

- Thuật toán:
  - Tính giá trị MSE (Mean Square Error) giữa ảnh gốc và ảnh đã lọc, sau đó sử dụng công thức PSNR đã cho để tính toán điểm số PSNR và trả về nó.

1.2 Output

1.2.1 Kết quả sau khi thực hiện bộ lọc mean filter

- Ảnh sau khi thực hiện bộ lọc mean filter:

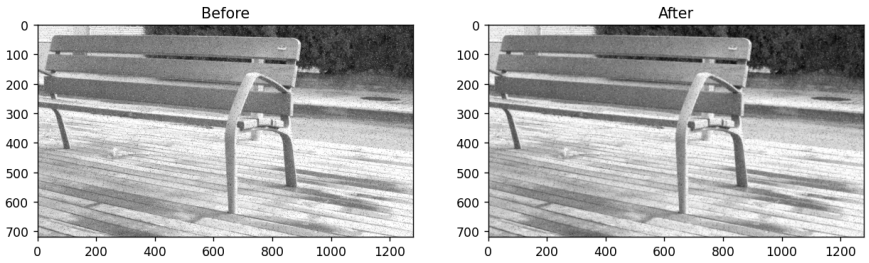

Figure 1: After mean filter

- Giá trị PSNR score: `31.61`

1.2.2 Kết quả sau khi thực hiện bộ lọc median filter

- Ảnh sau khi thực hiện bộ lọc median filter:

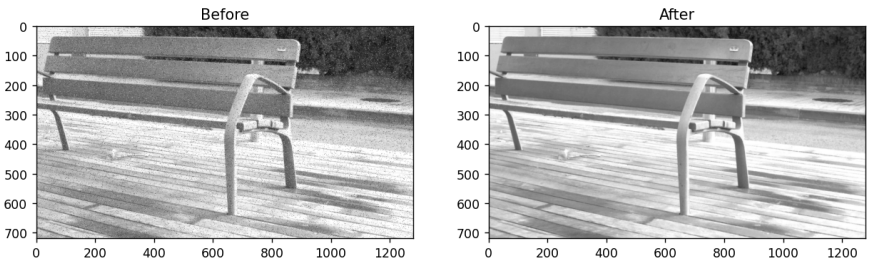

Figure 2: After median filter

- Giá trị PSNR score: `37.12`

1.2.3 Nhận xét về PSNR score

- Dựa trên các chỉ số Peak Signal-to-Noise Ratio (PSNR) đã cung cấp:
  - Điểm số PSNR của mean filter: 31.61
  - Điểm số PSNR của median filter: 37.12
- Điểm số PSNR đo lường chất lượng của ảnh sau khi lọc, trong đó giá trị PSNR cao hơn chỉ ra chất lượng ảnh tốt hơn. Trong trường hợp này, median filter có điểm số PSNR cao hơn đáng kể so với mean filter. Điều này ngụ ý rằng median filter có hiệu suất tốt hơn trong việc bảo tồn chất lượng hình ảnh và giảm nhiễu.
- Do đó, xét theo các chỉ số PSNR, median filter nên được chọn hơn mean filter cho các hình ảnh đã cung cấp.

2 Ex 2.1 và Ex 2.2 - Fourier Transform

2.1 Mã nguồn

2.1.1 DFT_slow() function

Listing 5: Code of DFT_slow() function

```
def DFT_slow(data):
    N = len(data)
    n = np.arange(N)
    k = n.reshape((N, 1))
    e = np.exp(-2j * np.pi * k * n / N)
    return np.dot(e, data)
```

- Thuật toán:
  - Xác định độ dài của dữ liệu đầu vào.
  - Tạo một mảng numpy chứa các chỉ số từ 0 đến độ dài dữ liệu - 1.
  - Tạo một ma trận exponents.
  - Tính toán DFT bằng cách nhân ma trận exponents với dữ liệu đầu vào và trả về kết quả.

2.1.2 DFT_2D() function

Listing 6: Code of DFT_2D() function

```
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
```

- Thuật toán:
  - Thực hiện biến đổi Fourier theo hàng cho mỗi hàng của hình ảnh đầu vào, sử dụng np.fft.fft với trục axis=1.
  - Thực hiện biến đổi Fourier theo cột cho mỗi cột của kết quả từ bước trước, sử dụng np.fft.fft với trục axis=0.

2.2 Output

2.2.1 Kết quả sau khi thực hiện DFT_2D() function

- Ảnh sau khi thực hiện DFT_2D() function:

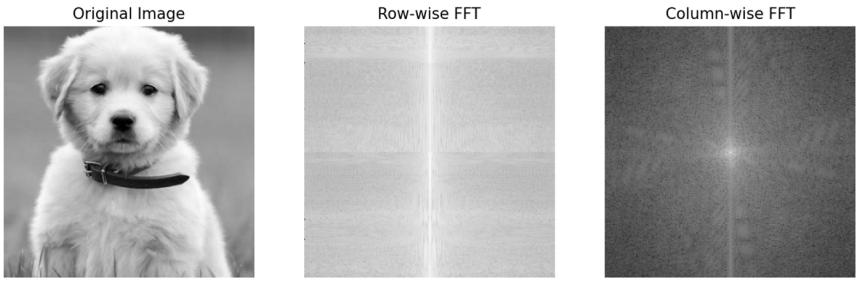

Figure 3: After DFT_2D() function

3 Ex 2.3 và Ex 2.4

3.1 Mã nguồn

3.1.1 filter_frequency() function

Listing 7: Code of filter_frequency() function

```
def filter_frequency(orig_img, mask):
    # Fourier transform of the original image
    f_img = fft2(orig_img)

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

    # Apply mask in frequency domain
    f_img_filtered = f_img_shifted * mask

    # Shift frequency coefficients back
    f_img_filtered_shifted = ifftshift(f_img_filtered)

    # Inverse transform
    img = np.abs(ifft2(f_img_filtered_shifted))

    return f_img_filtered, img
```

- Thuật toán:
  - Biến đổi Fourier của hình ảnh gốc (orig_img): Hình ảnh gốc được chuyển từ miền không gian sang miền tần số bằng cách sử dụng biến đổi Fourier (fft2), tạo ra một hình ảnh tần số (f_img).
  - Dịch các hệ số tần số về trung tâm: Các hệ số tần số của hình ảnh được dịch sang trung tâm bằng cách sử dụng hàm fftshift. Điều này cần thiết để chuẩn bị cho việc áp dụng mask.

- Áp dụng mask trong miền tần số: Mask được áp dụng trực tiếp lên hình ảnh tần số dịch chuyển (f_img_shifted). Mask này được ánh xạ pixel từ pixel tương ứng trên hình ảnh tần số.
- Dịch ngược các hệ số tần số: Sau khi áp dụng mask, các hệ số tần số được dịch trở lại vị trí ban đầu bằng cách sử dụng ifftshift
- Biến đổi nghịch: Cuối cùng, biến đổi Fourier nghịch (ifft2) được áp dụng để chuyển hình ảnh từ miền tần số trở lại miền không gian. Giá trị tuyệt đối của kết quả được lấy để đảm bảo giá trị thực.

3.1.2 create_hybrid_img() function

Listing 8: Code of reate_hybrid_img() function

```
def create_hybrid_img(img1, img2, r):
    # Step 1: Fourier transform for two input images
    img1_fft = fft2(img1)
    img2_fft = fft2(img2)

    # Step 2: Shift the frequency coefficients to center using fftshift
    img1_fft_shifted = fftshift(img1_fft)
    img2_fft_shifted = fftshift(img2_fft)

    # Step 3: Create mask based on radius (r)
    rows, cols = img1.shape
    crow, ccol = rows // 2, cols // 2
    mask = np.zeros((rows, cols), dtype=np.float32)
    for i in range(rows):
        for j in range(cols):
            dist = np.sqrt((i - crow) ** 2 + (j - ccol) ** 2)
            if dist <= r:
                mask[i, j] = 1

    # Step 4: Combine frequencies of two images using mask
    img1_hybrid_fft = img1_fft_shifted * mask
    img2_hybrid_fft = img2_fft_shifted * (1 - mask)
    hybrid_img_fft = img1_hybrid_fft + img2_hybrid_fft

    # Step 5: Shift the frequency coefficients back using ifftshift
    hybrid_img_fft_shifted = ifftshift(hybrid_img_fft)

    # Step 6: Invert transform using ifft2
    hybrid_img = np.abs(ifft2(hybrid_img_fft_shifted))

    return hybrid_img
```

- Thuật toán:
  - Biến đổi Fourier: Thực hiện biến đổi Fourier cho hai hình ảnh đầu vào (img1 và img2) bằng cách sử dụng hàm fft2, biến đổi các hình ảnh từ miền không gian sang miền tần số.
  - Dịch các hệ số tần số: Các hệ số tần số của cả hai hình ảnh được dịch đến trung tâm bằng cách sử dụng fftshift
  - Tạo Mask: Một mask được tạo ra dựa trên bán kính được cung cấp r. Mask này là một mảng nhị phân trong đó các pixel trong bán kính được đặt thành 1, và các pixel ngoài bán kính được đặt thành 0. Nó xác định các tần số nào từ img1 sẽ chiếm ưu thế trong hình ảnh hybrid.
  - Kết hợp các tần số: Các thành phần tần số của cả hai hình ảnh được kết hợp bằng cách sử dụng mask được tạo ra ở trên, đảm bảo rằng hình ảnh hybrid sẽ có các tần số ưu thế từ img1 trong bán kính đã chỉ định, và các tần số từ img2 ở bên ngoài bán kính đó.

– Dịch lại các hệ số tần số: Sau khi kết hợp các tần số, các hệ số tần số của hình ảnh hybrid kết quả được dịch trở lại vị trí ban đầu bằng cách sử dụng ifftshift

– Biến đổi nghịch: Cuối cùng, biến đổi Fourier nghịch (ifft2) được áp dụng để thu được hình ảnh hybrid trong miền không gian. Giá trị tuyệt đối của biến đổi nghịch được lấy để đảm bảo giá trị thực của đầu ra.

3.2 Output

3.2.1 Kết quả sau khi thực hiện filter_frequency() function

- Ảnh thu được:

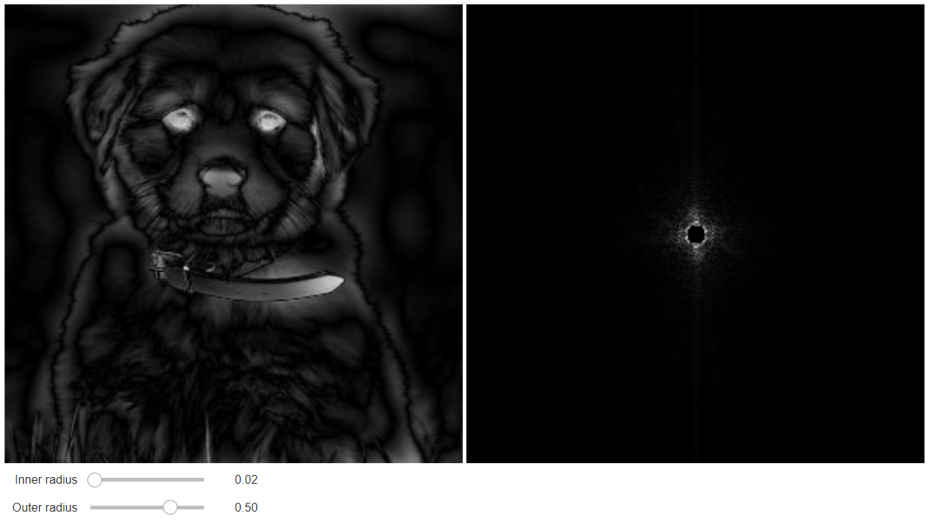

Figure 4: After filter_frequency() function

3.2.2 Kết quả sau khi thực hiện create_hybrid_img() function

- Ảnh thu được:

Figure 5: After create_hybrid_img() function