INT3404E 20 - Image Processing: Homework 03

Phạm Vũ Duy

1 Ex1 - Image Filtering

1.1 Source Code

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
```

- Algorithm:
  - Use the np.pad function from the NumPy library to add rows and columns around the original image. The parameter pad_size is calculated by dividing filter_size by 2, and then np.pad is used to add the corresponding rows and columns.
  - The mode is set to 'edge', meaning that pixel values at the edges of the image will be copied from the last rows and columns of the image. This helps preserve the edges of the image and avoids creating unwanted effects when padding.

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
```

- Algorithm:
  - The input image is padded using the padding_img function, ensuring that pixels at the edges of the image can also be processed.
  - A matrix with the same size as the input image is initialized to contain the smoothed image.

- The function iterates through each pixel of the image, and at each position, it extracts the neighborhood region around that pixel using the size of the filter.
- The average filter is applied by calculating the average of pixel values in the neighborhood region.
- The average value is assigned to the corresponding pixel in the smoothed image.

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
```

- Algorithm:
  - The input image is padded using the padding_img function to ensure that pixels at the edges of the image can also be processed.
  - A matrix with the same size as the input image is initialized to contain the smoothed image.
  - The function iterates through each pixel of the image, and at each position, extracts the neighborhood region around that pixel using the size of the filter.
  - The median filter is applied by calculating the median value of the pixel values in the neighborhood region.
  - The median value is assigned to the corresponding pixel in the smoothed image.

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
```

- Algorithm:
  - Calculate the Mean Square Error (MSE) between the original image and the filtered image, then use the provided PSNR formula to compute the PSNR score and return it.

1.2 Output

1.2.1 The result after applying the mean filter

- Image after applying the mean filter:

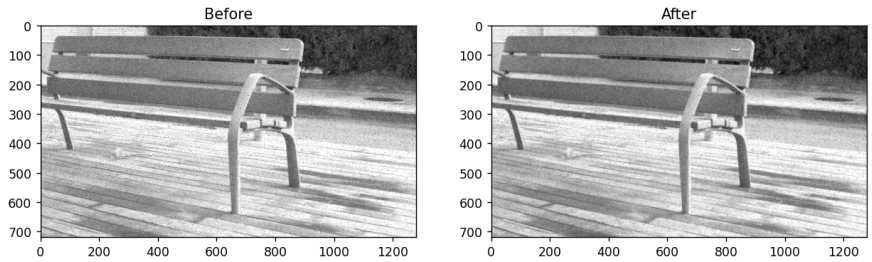

Figure 1: After mean filter

- The value of PSNR score: `31.61`

1.2.2 The result after applying the median filter

- Image after applying the median filter:

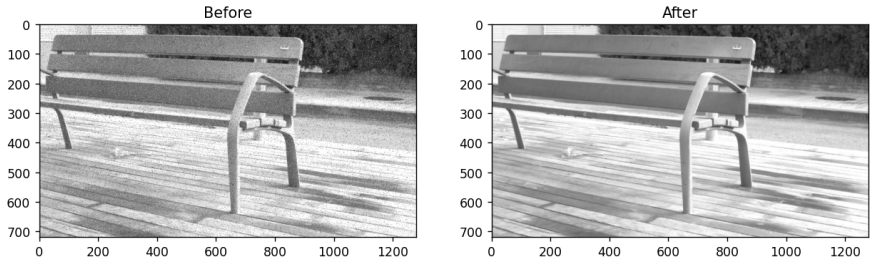

Figure 2: After median filter

- The value of PSNR score: `37.12`

1.2.3 Comments on PSNR score

- Based on the provided Peak Signal-to-Noise Ratio (PSNR) metrics:
  - The PSNR score of mean filter: 31.61

  - The PSNR score of median filter: 37.12

- The PSNR score measures the quality of the image after filtering, where a higher PSNR value indicates better image quality. In this case, the median filter has a significantly higher PSNR score compared to the mean filter. This implies that the median filter performs better in preserving image quality and reducing noise.
- Therefore, based on the PSNR metrics, the median filter should be chosen over the mean filter for the provided images.

2 Ex 2.1 and Ex 2.2 - Fourier Transform

2.1 Source Code

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)
```

- Algorithm:
  - Determine the length of the input data.
  - Create a NumPy array containing indices from 0 to the length of the data minus 1.
  - Create a matrix of exponents.
  - Compute the Discrete Fourier Transform (DFT) by multiplying the exponent matrix with the input data and return the result.

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
```

- Algorithm:
  - Perform row-wise Fourier transform for each row of the input image using np.fft.fft with axis=1.
  - Perform column-wise Fourier transform for each column of the result from the previous step using np.fft.fft with axis=0.

2.2 Output

2.2.1 The result after applying the DFT_2D() function

- Image after applying the DFT_2D() function:

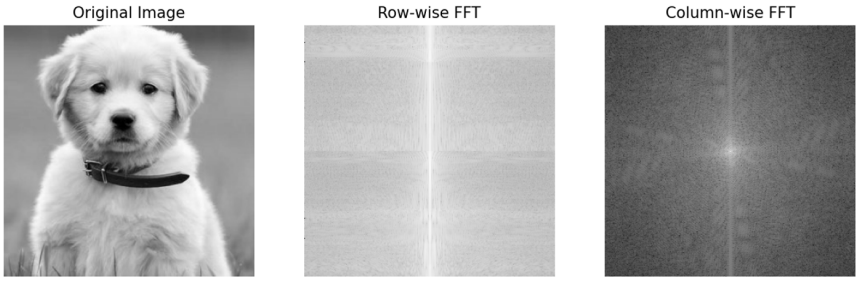

Figure 3: After DFT_2D() function

3 Ex 2.3 and Ex 2.4

3.1 Source Code

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
```

- Algorithm:
  - Fourier transform of the original image (orig_img): The original image is transformed from the spatial domain to the frequency domain using the Fourier transform (fft2), producing a frequency image (f_img).
  - Shift the frequency coefficients to the center: The frequency coefficients of the image are shifted to the center using the fftshift function. This is necessary to prepare for applying the mask.

- Apply mask in the frequency domain: The mask is directly applied to the shifted frequency image (f_img_shifted). This mask is pixel-wise mapped onto the corresponding pixels in the frequency image.
- Reverse shift the frequency coefficients: After applying the mask, the frequency coefficients are reversed back to their original positions using ifftshift.
- Inverse transform: Finally, the inverse Fourier transform (ifft2) is applied to transform the image from the frequency domain back to the spatial domain. The absolute value of the result is taken to ensure real values.

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
```

- Algorithm:
  - Fourier Transform: Perform Fourier transforms for two input images (img1 and img2) using the fft2 function, transforming the images from the spatial domain to the frequency domain.
  - Shift the frequency coefficients: The frequency coefficients of both images are shifted to the center using fftshift.
  - Create Mask: A mask is generated based on the provided radius, r. This mask is a binary array where pixels within the radius are set to 1, and pixels outside the radius are set to 0. It determines which frequencies from img1 will dominate in the hybrid image.
  - Combine Frequencies: The frequency components of both images are combined using the mask created above, ensuring that the hybrid image will have dominant frequencies from img1 within the specified radius, and frequencies from img2 outside that radius.

- Reverse shift the frequency coefficients: After combining the frequencies, the frequency coefficients of the resulting hybrid image are reversed back to their original positions using ifftshift.
- Inverse transform: Finally, the inverse Fourier transform (ifft2) is applied to obtain the hybrid image in the spatial domain. The absolute value of the inverse transform is taken to ensure the real values of the output.

3.2 Output

3.2.1 Result after applying the filter_frequency() function

- Image obtained:

Figure 4: After filter_frequency() function

3.2.2 The result after applying the create_hybrid_img() function

- Image obtained:

Figure 5: After create_hybrid_img() function