INT3404E 20 - Image Processing: Homework 2

Trần Phương Linh

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 pad_size parameter 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', which means that the 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 avoid 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 pixels at the edge 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 goes through each pixel of the image, and at each location, extracts the neighborhood around that pixel using the filter size.
- The median filter is applied by calculating the average of the pixel values in the neighborhood.
- The mean 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, ensuring pixels at the edge 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 goes through each pixel of the image, and at each location, extracts the neighborhood around that pixel using the filter size.
  - The median filter is applied by calculating the median of the pixel values in the neighborhood.
  - 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 MSE (Mean Square Error) value between the original image and the filtered image, then use the given PSNR formula to calculate the PSNR score and return it.

1.2 Output

1.2.1 Results after performing the mean filter

- Image after performing the mean filter:

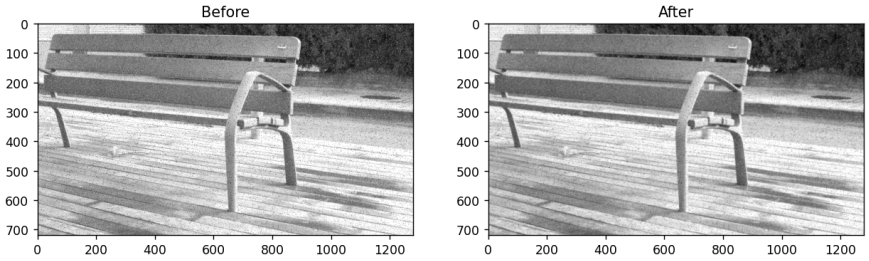

Figure 1: After mean filter

- PSNR score value: `31.61`

1.2.2 Results after performing the median filter

- Image after implementing the median filter:

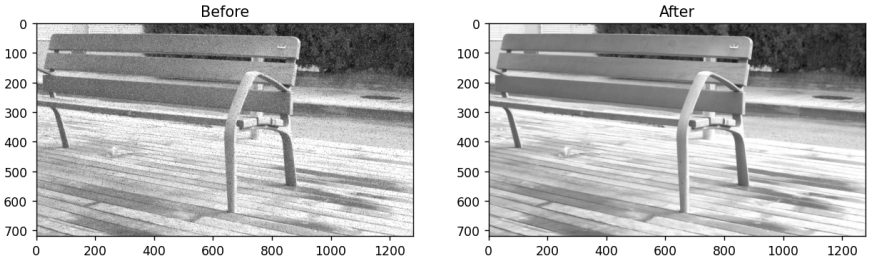

Figure 2: After median filter

- PSNR score value: `37.12`

1.2.3 Comments on PSNR score

- Based on the provided Peak Signal-to-Noise Ratio (PSNR) metrics:
  - Mean filter PSNR score: 31.61
  - Median filter PSNR score: 37.12
- The PSNR score measures the quality of an image after filtering, where a higher PSNR value indicates better image quality. In this case, the median filter has a significantly higher PSNR score than the mean filter. This implies that the median filter has better performance in preserving image quality and reducing noise.

- Therefore, considering the PSNR metrics, median filter should be chosen over mean filter for the provided images.

2 Ex 2.1 và 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)
```

- Source code:
  - Determines the length of the input data.
  - Create a numpy array containing indices from 0 to data length - 1.
  - Create a matrix of exponents.
  - Compute the DFT by multiplying the exponents matrix with the input data and returning 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 a row-wise Fourier transform for each row of the input image, using np.fft.fft with axis=1.
  - Perform a columnwise 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 Result after executing DFT_2D() function

- Image after executing DFT_2D() function:

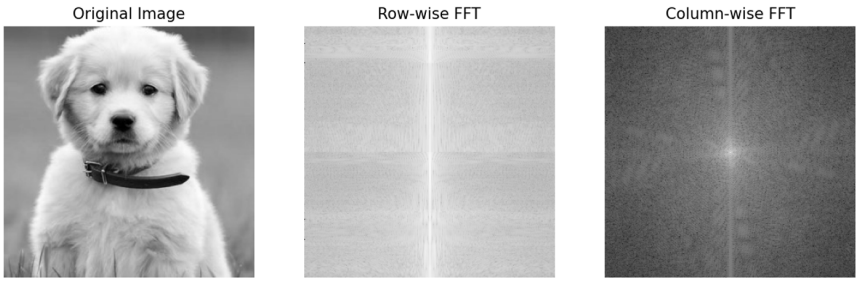

Figure 3: After DFT_2D() function

3 Ex 2.3 và 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 converted from the spatial domain to the frequency domain using the Fourier transform (fft2), creating a frequency image (f_img).
  - Shift frequency coefficients to center: The frequency coefficients of the image are shifted to center using the fftshift function. This is necessary to prepare for the application of the mask.
  - Apply mask in frequency domain: Mask is applied directly to the frequency shifted image (f_img_shifted). This mask is pixel-mapped from the corresponding pixel on the frequency image.
  - Backshift the frequency coefficients: After applying the mask, the frequency coefficients are shifted back to their original positions using ifftshift.

- Inverse transform: Finally, inverse Fourier transform (ifft2) is applied to convert the image from the frequency domain back to the spatial domain. The absolute value of the result is taken to ensure the true value.

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: Performs a Fourier transform for two input images (img1 and img2) using the fft2 function, transforming the images from the spatial domain to the frequency domain.
  - Shift frequency coefficients: The frequency coefficients of both images are shifted to the center using fftshift.
  - Create Mask: A mask is created 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 specified, and the frequencies from img2 are outside that radius.
  - Re-shift frequency coefficients: After combining the frequencies, the frequency coefficients of the resulting hybrid image are shifted back to their original positions using ifftshift.
  - Inverse transform: Finally, inverse Fourier transform (ifft2) is applied to obtain hybrid images in the spatial domain. The absolute value of the inverse transform is taken to ensure the true value of the output.

3.2 Output

3.2.1 Result after executing filter_frequency() function

- Photos obtained:

Figure 4: After filter_frequency() function

3.2.2 Result after executing create_hybrid_img() function

- Photos obtained:

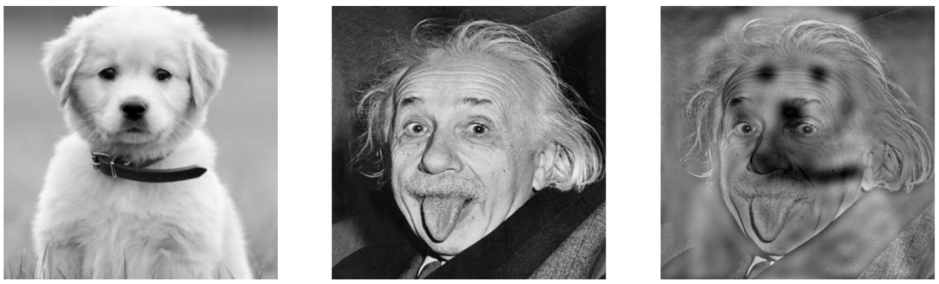

Figure 5: After create_hybrid_img() function