

INT3404E 20 - Image Processing: Homeworks 2

Lã Thị Thanh Thúy - MSV: 21020478

1 Image Filtering

1.1 Hàm `padding_img`

```
def padding_img(img, filter_size=3):  
    """  
    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  
    return np.pad(img, pad_size, mode='edge')  
  
if __name__ == '__main__':  
    ...  
    # Padding image  
    padding_image = padding_img(img, filter_size=100)  
    show_res(img, padding_image)
```

Hàm `padding_img` thực hiện việc tạo padding cho ảnh đầu vào. Hàm nhận vào hai tham số:

- `noise.png`: Ảnh gốc dưới dạng ảnh cv2.



Figure 1: noise.png.

- `filter_size`: Kích thước của bộ lọc vuông sẽ được áp dụng lên ảnh.

Trước tiên, hàm tính toán kích thước padding cần thiết bằng cách chia kích thước bộ lọc cho 2 và lấy phần nguyên. Sau đó, hàm `np.pad` được gọi để thêm padding vào ảnh, sử dụng chế độ 'edge' để tạo padding

bằng giá trị cạnh của ảnh. Kết quả trả về (padded_img) là ảnh đã được padding, với kích thước lớn hơn ảnh gốc tùy thuộc vào kích thước của bộ lọc.

Figure 1

- □ ×

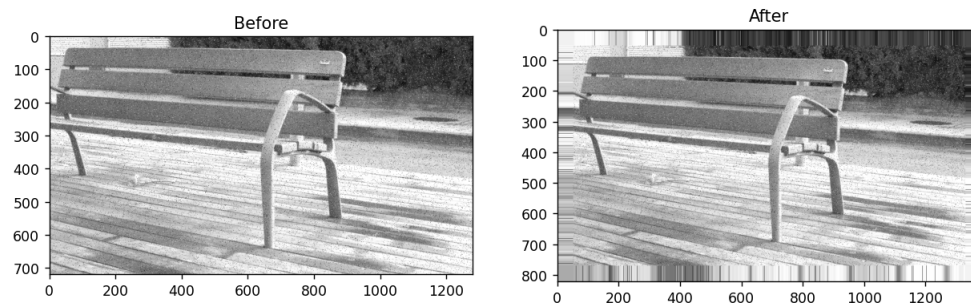


Figure 2: Kết quả trả về của hàm padding_img().

1.2 Hàm mean_filter

Hàm mean_filter dùng để làm mịn ảnh bằng bộ lọc trung bình có kích thước filter_size.

Hàm nhận vào hai tham số:

- Figure 1: noise.png: Ảnh gốc dưới dạng ảnh cv2.
- filter_size: Kích thước của bộ lọc vuông sẽ được áp dụng lên ảnh.

Hàm được triển khai như sau:

```
def mean_filter(img, filter_size=3):
    # Need to implement here
    img_pad = padding_img(img, filter_size)
    h, w = img.shape
    img_smooth = np.zeros_like(img)
5   for i in range(h):
        for j in range(w):
            img_smooth[i, j] = np.mean(img_pad[i:i+filter_size, j:j+filter_size])
    return img_smooth
10
if __name__ == '__main__':
    ...
    # Mean filter
    mean_smoothed_img = mean_filter(img, filter_size)
15   show_res(img, mean_smoothed_img)
```

Trước tiên, hàm tạo padding cho ảnh gốc sử dụng hàm `padding_img` đã được đề cập ở trên. Sau đó, hàm tạo một ảnh mới với kích thước giống như ảnh gốc. Cuối cùng, hàm đi qua từng pixel của ảnh và tính giá trị trung bình của các pixel xung quanh nó trong cửa sổ có kích thước `filter_size`. Hàm trả về `smoothed_img`, là ảnh đã được làm mịn bằng bộ lọc trung bình. Kích thước của ảnh này giống với ảnh gốc.

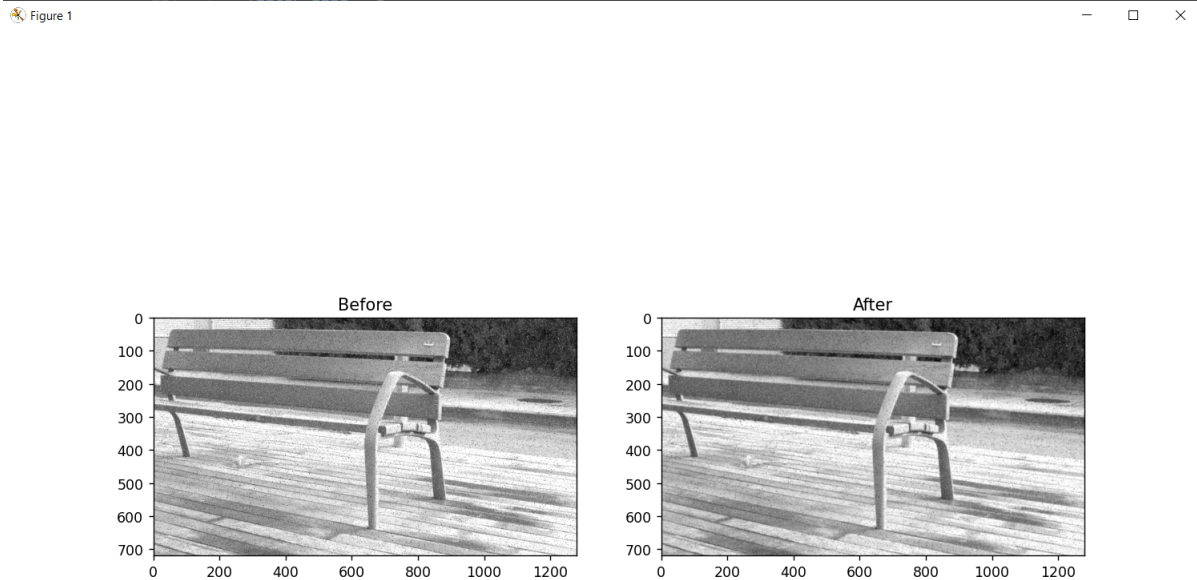


Figure 3: Kết quả trả về của hàm `mean_filter()`.

1.3 Hàm `median_filter`

Hàm `median_filter` dùng để làm mịn ảnh bằng bộ lọc trung vị có kích thước `filter_size`. Hàm nhận vào hai tham số:

- `Figure 1: noise.png`: Ảnh gốc dưới dạng ảnh `cv2`.
- `filter_size`: Kích thước của bộ lọc vuông sẽ được áp dụng lên ảnh.

Hàm được triển khai như sau:

```
def median_filter(img, filter_size=3):
    # Need to implement here
    img_pad = padding_img(img, filter_size)
    h, w = img.shape
    img_smooth = np.zeros_like(img)
    for i in range(h):
        for j in range(w):
            img_smooth[i, j] = np.median(img_pad[i:i+filter_size, j:j+filter_size])
    return img_smooth
if __name__ == '__main__':
    ...
    # Median filter
    median_smoothed_img = median_filter(img, filter_size)
    show_res(img, median_smoothed_img)
    print('PSNR score of median filter: ', psnr(img, median_smoothed_img))
```

Trước tiên, hàm tạo padding cho ảnh gốc sử dụng hàm `padding_img`. Sau đó, hàm tạo một ảnh mới với kích thước giống như ảnh gốc. Cuối cùng, hàm đi qua từng pixel của ảnh và tính giá trị trung vị của các pixel xung quanh nó trong cửa sổ có kích thước `filter_size`. Kết quả trả về (`smoothed_img`) là ảnh đã được làm mịn bằng bộ lọc trung vị.

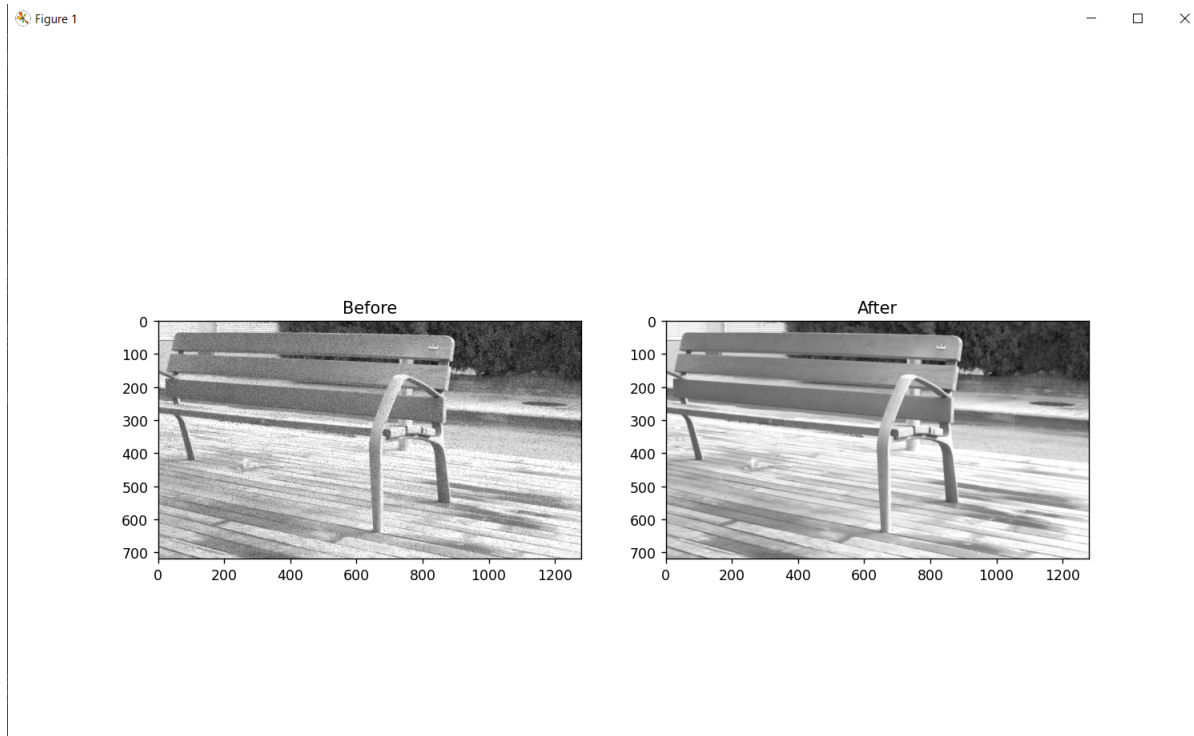


Figure 4: Kết quả trả về của hàm `median_filter()`.

1.4 Hàm `psnr`

```
def psnr(gt_img, smooth_img):
    """
    Inputs:
        gt_img: cv2 image: groundtruth image
        smooth_img: cv2 image: smoothed image
    Outputs:
        psnr_score: PSNR score
    """
    mse = np.mean((gt_img - smooth_img) ** 2)
    max_pixel = 255.0
    psnr_score = 20 * math.log10(max_pixel / math.sqrt(mse))
    return psnr_score
```

Hàm `psnr` được sử dụng để tính toán chỉ số PSNR (Peak Signal-to-Noise Ratio), một chỉ số đo lường chất lượng của hình ảnh sau khi được xử lý so với hình ảnh gốc. Hàm nhận vào hai tham số, `gt_img` là hình ảnh gốc và `smooth_img` là hình ảnh đã được làm mịn.

Hàm đầu tiên tính MSE (Mean Squared Error), đo lường sự khác biệt giữa hai hình ảnh. Sau đó, nó tính giá trị PSNR dựa trên MSE và giá trị pixel tối đa (trong trường hợp này là 255.0 cho hình ảnh grayscale). Kết quả trả về là `psnr_score`, biểu thị chất lượng của hình ảnh sau khi được xử lý so với hình ảnh gốc.

Kết quả PSNR của 2 hàm `mean_filter()` và `median_filter()` lần lượt là:

- PSNR score of mean filter: 31.60889963499979

- PSNR score of median filter: 37.11957830085524

Bởi vậy, `median_filter()` cho kết quả ảnh chất lượng hơn.

2 Fourier Transform

2.1 DFT_slow()

```
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
    """
    N = data.shape[0]
    n = np.arange(N)
    k = n.reshape((N, 1))
    e = np.exp(-2j * np.pi * k * n / N)
    DFT = np.dot(e, data)
    return DFT

if __name__ == '__main__':
    # ex1
    x = np.random.random(1024)
    print(DFT_slow(x))
    print(np.allclose(DFT_slow(x), np.fft.fft(x)))
```

Hàm `DFT_slow` được sử dụng để thực hiện Biến đổi Fourier rời rạc (DFT) cho một tín hiệu 1 chiều. Hàm này nhận đầu vào là một mảng numpy 1 chiều `data`.

Trước tiên, hàm tính số lượng các phần tử trong mảng `data` (`N`). Sau đó, nó tạo một mảng `n` gồm các số từ 0 đến `N-1` và một mảng `k` là mảng `n` được chuyển vị.

Tiếp theo, hàm tạo một mảng `e` bằng cách tính toán hàm mũ phức của $-2j * \pi * k * n / N$.

Cuối cùng, hàm tính DFT bằng cách nhân ma trận `e` với mảng `data` và trả về kết quả.

```
True
PS D:\XLA\hw2\HW2> & d:/XLA/hw2/HW2/.venv/Scripts/python.exe d:/XLA/hw2/HW2/ex212.py
[5.18099868e+02+0.j          9.89871903e-02-3.2909907j
 8.63094234e+00-5.4819813j ... 1.20101635e+01+1.2660281j
 8.63094234e+00+5.4819813j 9.89871903e-02+3.2909907j]
True
```

Figure 5: Kết quả trả về của hàm `DFT_slow()`.

2.2 Hàm DFT_2D

```
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
```

```

"""
H, W = gray_img.shape
row_fft = np.zeros((H, W), dtype=complex)
row_col_fft = np.zeros((H, W), dtype=complex)
15 for i in range(H):
    row_fft[i, :] = DFT_slow(gray_img[i, :])
    for i in range(W):
        row_col_fft[:, i] = DFT_slow(row_fft[:, i])
20 return row_fft, row_col_fft

if __name__ == '__main__':

# ex2
img = io_url.imread('https://img2.zerqnet.com/2309662_300.jpg')
25 gray_img = np.mean(img, -1)
row_fft, row_col_fft = DFT_2D(gray_img)
show_img(gray_img, row_fft, row_col_fft)

```

Hàm DFT_2D thực hiện Biến đổi Fourier rời rạc 2 chiều (2D DFT) cho một hình ảnh xám đầu vào.

Hàm này trước tiên tạo hai mảng row_fft và row_col_fft với kích thước tương tự như hình ảnh đầu vào và kiểu dữ liệu là số phức.

Sau đó, nó thực hiện DFT theo hàng cho hình ảnh đầu vào bằng cách gọi hàm DFT_slow cho mỗi hàng và lưu kết quả vào mảng row_fft.

Tiếp theo, nó thực hiện DFT theo cột cho row_fft bằng cách gọi hàm DFT_slow cho mỗi cột và lưu kết quả vào mảng row_col_fft.

Hàm trả về cả hai mảng row_fft và row_col_fft, biểu diễn DFT 2D của hình ảnh đầu vào.

Figure 1

— □ ×

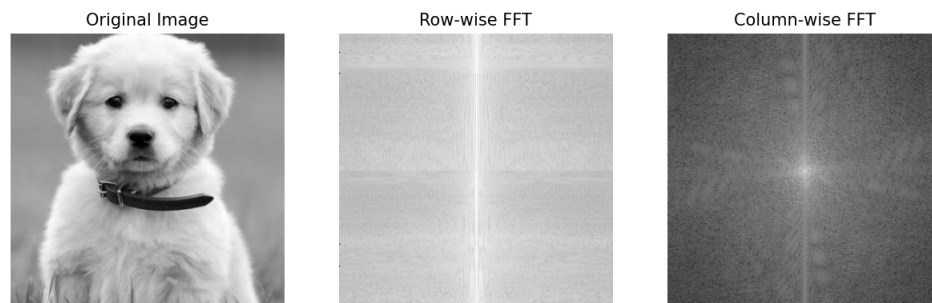


Figure 6: Kết quả trả về của hàm DFT_2D.

2.3 Hàm filter_frequency

```
def filter_frequency(orig_img, 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
"""
10 f_img = np.fft.fft2(orig_img)
    f_img_shift = np.fft.fftshift(f_img)
    f_img_shift_masked = f_img_shift * mask
    f_img_masked = np.fft.ifftshift(f_img_shift_masked)
    img_filtered = np.fft.ifft2(f_img_masked)
15 img_filtered = np.real(img_filtered)

return f_img_shift_masked, img_filtered

```

Hàm `filter_frequency` được sử dụng để loại bỏ tần số dựa trên mặt nạ đã cho. Hàm này nhận vào hai tham số, `orig_img` là hình ảnh gốc và `mask` là một mảng có cùng kích thước với `orig_img` chỉ ra tần số nào được giữ lại hoặc loại bỏ.

Hàm trả về hình ảnh tần số sau khi áp dụng mặt nạ và hình ảnh sau khi áp dụng mặt nạ.



Figure 7: Kết quả trả về của hàm `filter_frequency`.

2.4 Hàm `create_hybrid_img`

```

def create_hybrid_img(img1, img2, r):
    # You need to implement the function
    # Compute 2D FFTs
    fft_img1 = fftshift(fft2(img1))
    5 fft_img2 = fftshift(fft2(img2))

    # Create a mask
    cy, cx = np.array(img1.shape) // 2
    y, x = np.ogrid[-cy:cy, -cx:cx]
    10 mask = x*x + y*y <= r*r

    # Combine frequency of 2 images using the mask
    hybrid_fft = fft_img2 * mask + fft_img1 * ~mask


```

```
15 # Shift frequency coefs back and invert transform
    hybrid_img = np.abs(fft2(fftshift(hybrid_fft)))

    return hybrid_img
```

Hàm `create_hybrid_img` tạo ra một hình ảnh lai từ hai hình ảnh đầu vào `img1` và `img2` bằng cách sử dụng một mặt nạ tần số.

Hàm trả về hình ảnh lai.

 (-0.5, 511.5, 511.5, -0.5)

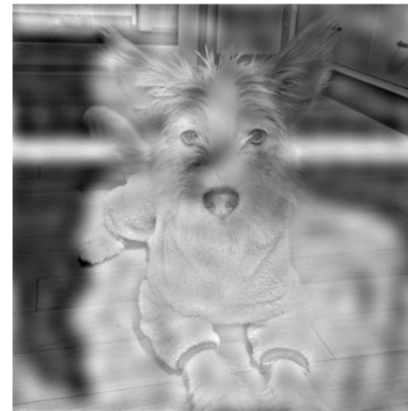


Figure 8: Kết quả trả về của hàm `create_hybrid_image()`.