INT3404E 20 - Image Processing: Homeworks 2

Lưu Văn Đức Thiệu

1 Image Filtering

(a) Implement one Replicate padding, box/mean and median filters for removing noise

```python
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
    """
    assert filter_size % 2 == 1, "Filter size must be odd number"
    assert filter_size <= min(img.shape), "Filter size must not be too large"
    s = filter_size // 2
    pad_top = np.tile(img[0, :], (s, 1))
    pad_bot = np.tile(img[-1, :], (s, 1))
    img = np.concatenate([pad_top, img, pad_bot], axis=0)
    pad_left = np.tile(img[:, 0], (s, 1)).T
    pad_right = np.tile(img[:, -1], (s, 1)).T
    img = np.concatenate([pad_left, img, pad_right], axis=1)
    return img
```

Listing 1: padding_img

```python
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.
    """
    convolve = np.zeros((img.shape))
    img = padding_img(img, filter_size)
    filter = np.ones((filter_size, filter_size)) / (filter_size**2)
    s = filter_size // 2
    x, y = img.shape
    for v in range(s, x - s):
        for h in range(s, y - s):  #
            area = img[(v - s) : (v + s + 1), (h - s) : (h + s + 1)]
            convolve[v - s, h - s] = np.sum(np.multiply(filter, area))
    return convolve
```

Listing 2: mean_filter

```python
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.
    """
    median = np.zeros((img.shape))
    img = padding_img(img, filter_size)
    s = filter_size // 2
    x, y = img.shape
    for v in range(s, x - s):
        for h in range(s, y - s):
            area = img[(v - s) : (v + s + 1), (h - s) : (h + s + 1)]
            median[v - s, h - s] = np.median(area)
    return median
```

Listing 3: median_filter

(b) Implement the Peak Signal-to-Noise Ratio (PSNR) metric

```python
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
    """
    try:
        gt_img = np.array(gt_img)
        smooth_img = np.array(smooth_img)
    except Exception:
        raise ValueError("Input must be 2D array like format")
    max_possible_value = 255
    mse = np.mean((gt_img - smooth_img) ** 2)
    return 10 * np.log10(max_possible_value**2 / mse)
```

Listing 4: psnr

(c) Considering the PSNR metrics, PSNR score of mean filter is: 18.295335205532066 and PSNR score of median filter is: 17.835212311092135 so mean filter is better for provided image.

(d) Output of mean filter: Figure 1, output of median filter: Figure 2

2 Fourier Transform

2.1 1D Fourier Transform

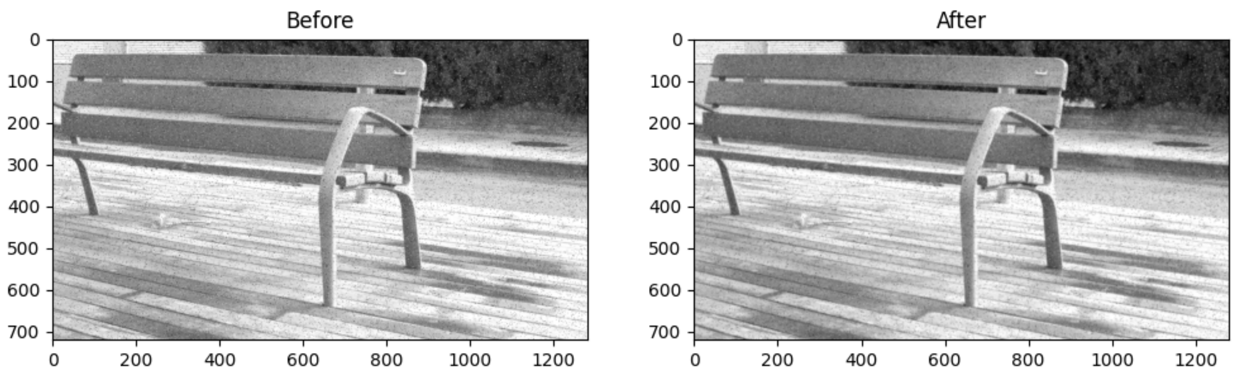

Figure 1: Mean filter result

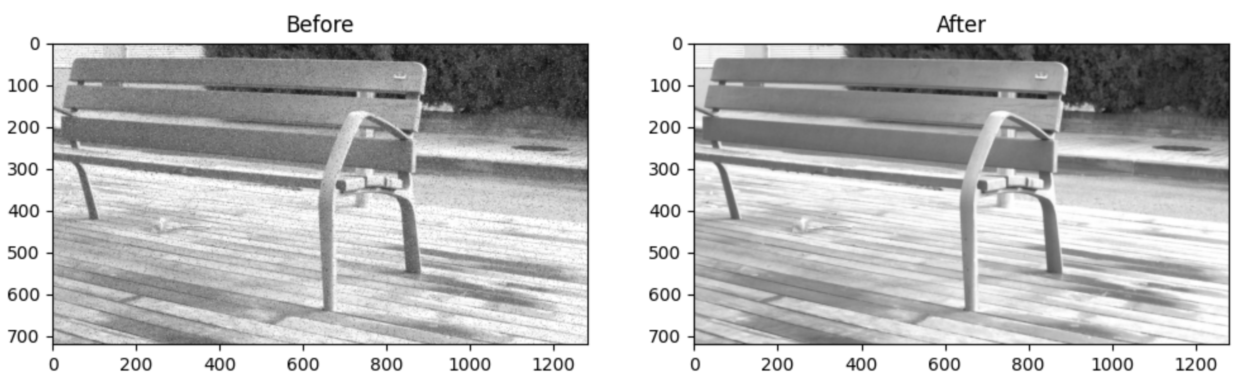

Figure 2: Median filter result

```python
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]
    DFT = np.zeros((N,), dtype=np.complex_)
    for k in range(N):
        real = 0
        imag = 0
        for n in range(N):
            real += data[n] * np.cos(2 * np.pi * n * k / N)
            imag -= data[n] * np.sin(2 * np.pi * n * k / N)
        DFT[k] = complex(real, imag)
    return DFT
```

Listing 5: DFT_slow

2.2 2D Fourier Transform

```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
    """
    row_fft = np.zeros(gray_img.shape, dtype=np.complex_)
    row_col_fft = np.zeros(gray_img.shape, dtype=np.complex_)
    for i in range(gray_img.shape[0]):
        row_fft[i] = np.fft.fft(gray_img[i])
    for j in range(gray_img.shape[1]):
        col = row_fft[:, j]
        row_col_fft[:, j] = np.fft.fft(col)
    return row_fft, row_col_fft
```

Listing 6: DFT_2D

Output of 2D Fourier Transform: Figure 3

2.3 Frequency Removal Procedure

```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
  """
  f_img = np.fft.fft2(orig_img)
  f_img = np.fft.fftshift(f_img)
```

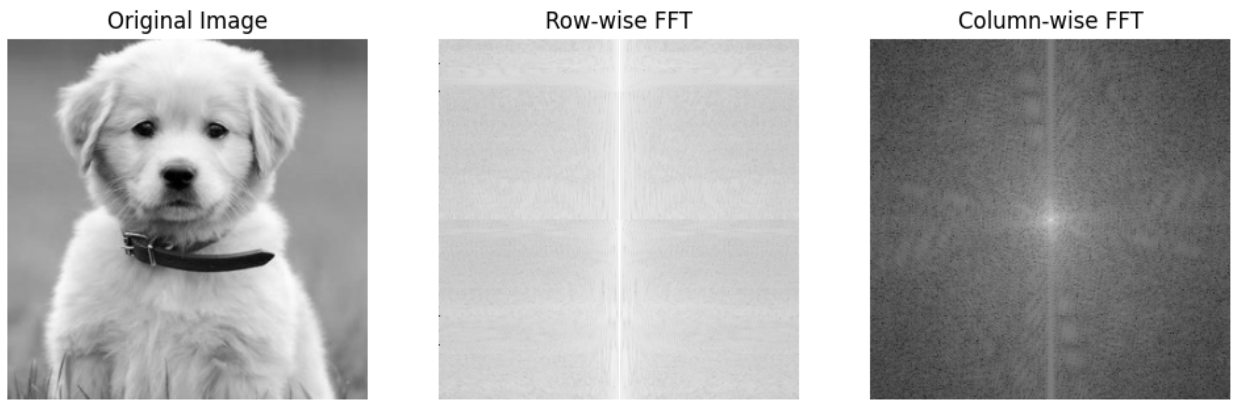

Figure 3: 2D Fourier Transform result

```
    f_img_masked = f_img * mask
    f_img = np.fft.ifftshift(f_img_masked)
    img = np.abs(np.fft.ifft2(f_img))
    f_img_masked = np.abs(f_img_masked)
    return f_img_masked, img
```

Listing 7: filter_frequency

Output of frequency removal: Figure 4

2.4 Creating a Hybrid Image

```
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.
    """
    f_img1 = np.fft.fft2(img1)
    f_img2 = np.fft.fft2(img2)
    f_img1 = np.fft.fftshift(f_img1)
    f_img2 = np.fft.fftshift(f_img2)
    mask = np.zeros_like(f_img1)
    rows, cols = img1.shape
    center = (rows//2, cols//2)

    for i in range(rows):
        for j in range(cols):
            if np.sqrt((i - center[0])**2 + (j - center[1])**2) < r:
                mask[i, j] = 1

    # Apply the mask to the Fourier transform of one image
    img1_fft_filtered = f_img1 * mask

    # Combine the modified Fourier transforms of the two images
    hybrid_fft = img1_fft_filtered + (1 - mask) * f_img2

    # Apply inverse Fourier transform to get the hybrid image
    hybrid_img = np.abs(np.fft.ifft2(hybrid_fft))

```

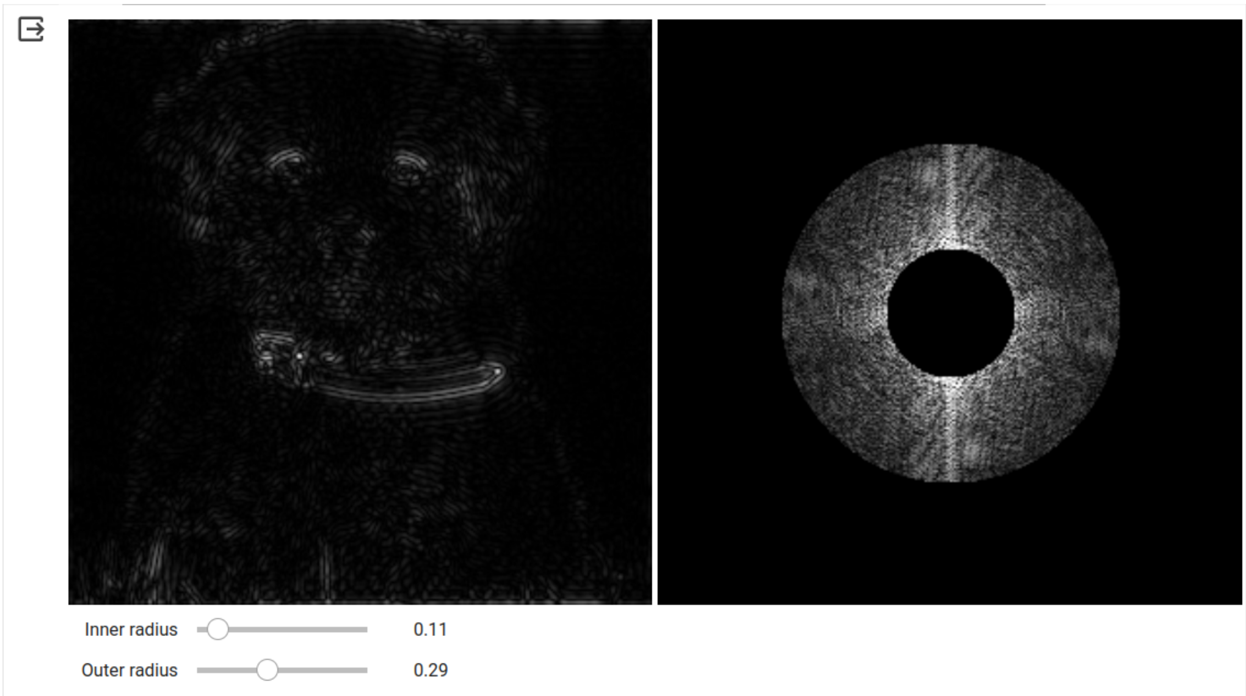

Figure 4: Frequency Removal result

```
    return hybrid_img
```

Listing 8: create_hybrid_img

Output an example of hybrid image: Figure 5

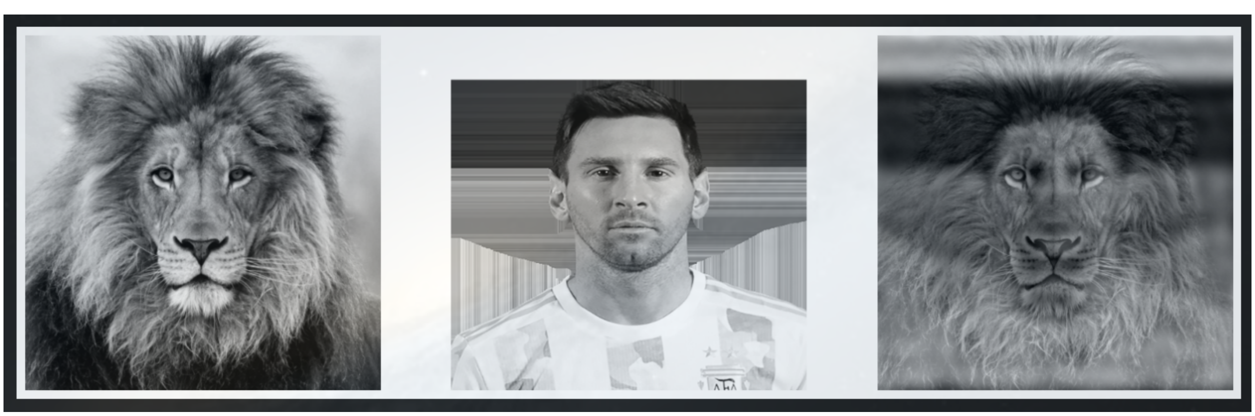

Figure 5: An example of hybrid image