

INT3404E 20 - Image Processing: Homeworks 2

Trần Minh Quân

1 Homework Objectives

Here are the detailed objectives of this homework:

1. To achieve a comprehensive understanding of how basic image filters operate.
2. To gain a solid understanding of the Fourier Transform (FT) algorithm.

2 Image Filtering

- (a) Implement functions in the supplied code file: `padding_img`, `mean_filter`, `median_filter`. The result of `mean_filter` and `median_filter` are shown in Figure 3 and Figure 4.

Listing 1: Padding Image function

```
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.  
    5    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:  
    10    padded_img: cv2 image: the padding image  
    """  
    # Need to implement here  
    height, width = img.shape[:2]  
  
    15    pad_size = filter_size // 2  
  
    padded_img = np.zeros((height + 2 * pad_size, width + 2 * pad_size), dtype=img.dtype)  
  
    padded_img[pad_size:pad_size + height, pad_size:pad_size + width] = img  
    20  
    for i in range(pad_size):  
        padded_img[i, pad_size:pad_size + width] = img[0, :]  
        padded_img[-i - 1, pad_size:pad_size + width] = img[-1, :]  
  
    25    for i in range(pad_size):  
        padded_img[pad_size:pad_size + height, i] = padded_img[pad_size:pad_size + height,  
        pad_size]  
        padded_img[pad_size:pad_size + height, -i - 1] = padded_img[pad_size:pad_size +  
        height, -pad_size - 1]  
  
    padded_img[:pad_size, :pad_size] = img[0, 0]  
    30    padded_img[:pad_size, -pad_size:] = img[0, -1]  
    padded_img[-pad_size:, :pad_size] = img[-1, 0]  
    padded_img[-pad_size:, -pad_size:] = img[-1, -1]  
  
    return padded_img
```

Listing 2: Mean filter function

```

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.
    5
    Inputs:
        img: cv2 image: original image
        filter_size: int: size of square filter,
    Return:
        smoothed_img: cv2 image: the smoothed image with mean filter.
    10
    """
    # Need to implement here
    padded_img = padding_img(img, filter_size)
    smoothed_img = np.zeros_like(img)
    for i in range(img.shape[0]):
    15
        for j in range(img.shape[1]):
            smoothed_img[i, j] = np.mean(padded_img[i:i + filter_size, j:j + filter_size])
    return smoothed_img

```

Listing 3: Median filter function

```

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.
    5
    Inputs:
        img: cv2 image: original image
        filter_size: int: size of square filter
    Return:
        smoothed_img: cv2 image: the smoothed image with median filter.
    10
    """
    # Need to implement here
    padded_img = padding_img(img=img, filter_size=filter_size)
    height, width = img.shape[:2]
    15
    smoothed_img = np.zeros_like(img)

    for i in range(height):
        for j in range(width):
            neighborhood = padded_img[i:i + filter_size, j:j + filter_size]
            median_value = np.median(neighborhood)
            20
            smoothed_img[i, j] = median_value

    return smoothed_img

```

- (b) Implement the Peak Signal-to-Noise Ratio (PSNR) metric, where MAX is the maximum possible pixel value (typically 255 for 8-bit images), and MSE is the Mean Square Error between the two images.

$$PSNR = 10 \cdot \log_{10} \left(\frac{MAX^2}{MSE} \right)$$

Listing 4: PSNR function

```

def psnr(gt_img, smooth_img):
    """
    Calculate the PSNR metric
    Inputs:
    5      gt_img: cv2 image: groundtruth image
          smooth_img: cv2 image: smoothed image
    Outputs:
          psnr_score: PSNR score
    """
    10    mse = (np.square(np.subtract(gt_img, smooth_img)).mean())
    max_pixel = 255.0
    if mse == 0:
        return float('inf')
    else:
    15    psnr = 10 * math.log10((max_pixel ** 2) / mse)
        return psnr

```

- (c) When comparing between mean filter and median filter based on PSNR values, the one with a higher PSNR is more effective in enhancing image quality. In this case, with PSNR scores of 31.608 for the mean filter and 37.119 for the median filter, the median filter significantly outperforms the mean filter in terms of PSNR. Therefore, Thus, considering the PSNR metrics, the median filter should be the chosen one.



Figure 1: Original image



Figure 2: Noise image



Figure 3: Noise image with Mean filter



Figure 4: Noise image with Median filter

3 Fourier Transform

3.1 1D Fourier Transform

Implement a function named `DFT_slow` to perform the Discrete Fourier Transform (DFT) on a one-dimensional signal.

Listing 5: `DFT_slow` function

```
def DFT_slow(data):  
    """  
    Implement the discrete Fourier Transform for a 1D signal  
    params:  
5     data: Nx1: (N, ): 1D numpy array  
    returns:  
     DFT: Nx1: 1D numpy array  
    """  
    # You need to implement the DFT here  
10    N = len(data)  
    n = np.arange(N)  
    k = n.reshape((N, 1))  
    e_t = np.exp(-2j * np.pi * k * n / N)  
    D = np.dot(e_t, data)  
15    return D
```

3.2 2D Fourier Transform

The procedure to simulate a 2D Fourier Transform is as follows:

1. Conducting a Fourier Transform on each row of the input 2D signal. This step transforms the signal along the horizontal axis.
2. Perform a Fourier Transform on each column of the previously obtained result.

The result is shown in Figure 5.

Listing 6: 2D Fourier Transform function

```
def DFT_2D(gray_img):  
    """  
    Implement the 2D Discrete Fourier Transform  
    Note that: dtype of the output should be complex_  
5     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  
10     row_col_fft: (H, W): 2D numpy array that contains the column-wise FFT of the input image  
    """  
    # You need to implement the DFT here  
    H, W = gray_img.shape  
    row_fft = np.zeros((H, W), dtype=np.complex_)  
15    row_col_fft = np.zeros((H, W), dtype=np.complex_)  
  
    # Row-wise FFT  
    for i in range(H):  
        row_fft[i, :] = np.fft.fft(gray_img[i, :])  
20  
    # Column-wise FFT  
    for j in range(W):  
        row_col_fft[:, j] = np.fft.fft(row_fft[:, j])
```

```
25 | return row_fft, row_col_fft
```

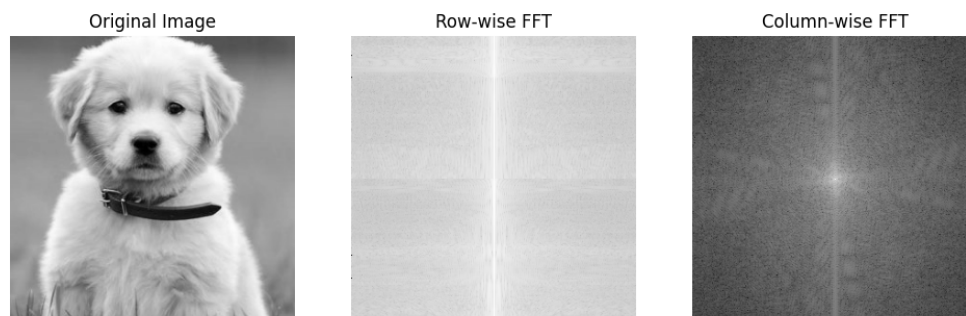


Figure 5: Output for 2D Fourier Transform Exercise

3.3 Frequency Removal Procedure

Implement the `filter_frequency` function in the notebook. The result is shown in Figure 6.

Listing 7: Frequency filter function

```
def filter_frequency(orig_img, mask):  
    """  
    You need to remove frequency based on the given mask.  
    Params:  
5     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    """  
    filter_img = np.fft.fft2(orig_img)  
  
    filter_img_shifted = np.fft.fftshift(filter_img)  
  
15    filter_img_filtered = filter_img_shifted * mask  
  
    filter_img_filtered_shifted = np.fft.ifftshift(filter_img_filtered)  
  
    img = np.fft.ifft2(filter_img_filtered_shifted)  
20    img = np.real(img)
```

```
f_img = np.abs(filter_img_filtered)
return f_img, img
```

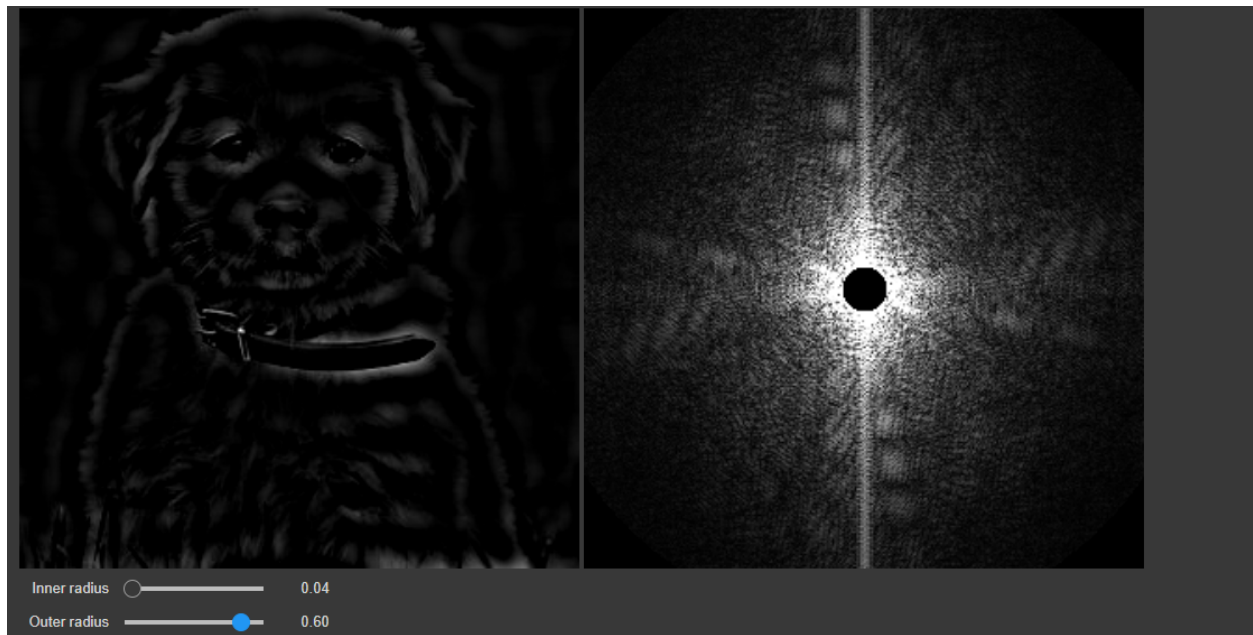


Figure 6: Output for 2D Frequency Removal Exercise

3.4 Creating a Hybrid Image

Implement the function `create_hybrid_img` in the notebook. The result is shown in Figure 7.

Listing 8: Creating a Hybrid Image function

```
def create_hybrid_img(img1, img2, r):
    """
    Create hybrid image
    Params:
    5     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.
    """
    # You need to implement the function
    10 img1_transformed = np.fft.fft2(img1)
    img2_transformed = np.fft.fft2(img2)

    dft1_shifted = np.fft.fftshift(img1_transformed)
    dft2_shifted = np.fft.fftshift(img2_transformed)
    15 mask = create_mask(img1=img1, r=r)

    hybrid_dft = dft1_shifted * mask + dft2_shifted * (1 - mask)

    20 inv_hybrid_dft = np.fft.ifftshift(hybrid_dft)

    hybrid_img = np.fft.ifft2(inv_hybrid_dft)
    hybrid_img = np.real(hybrid_img)
```

```
25 | return hybrid_img
```

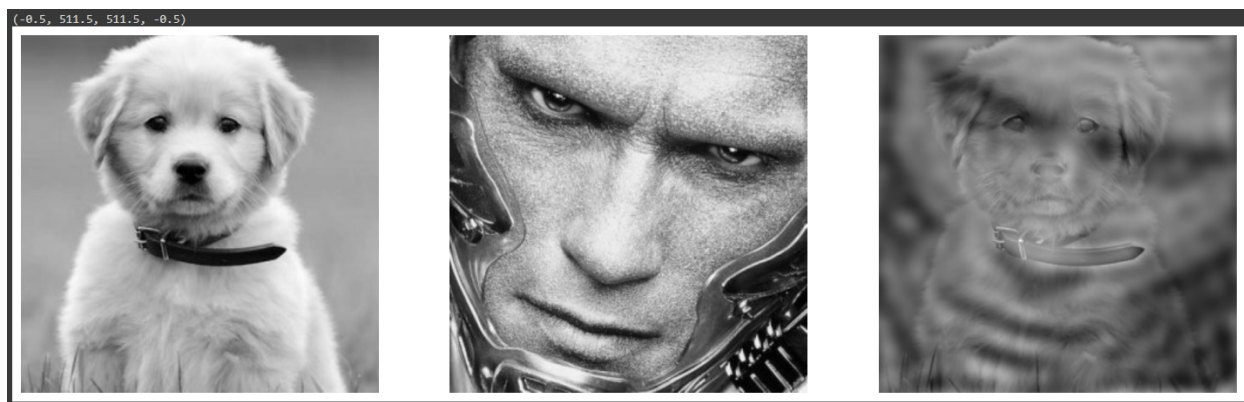


Figure 7: Hybrid Image