INT3404E 20 - Image Processing: Homeworks 2

Nguyen Minh Quan

21/04/2024

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) Implemen 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

```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
    """
    # Need to implement here
    padding_size = filter_size // 2
    h, w = img.shape[:2]
    padded_h, padded_w = h + 2 * padding_size, w + 2 * padding_size
    padded_img = np.zeros((padded_h, padded_w), dtype=img.dtype)
    padded_img[padding_size:padded_h - padding_size, padding_size:padded_w - padding_size] = img

    # Replicate padding for the top and bottom borders
    padded_img[0:padding_size, padding_size:padded_w - padding_size] = img[0]
    padded_img[padded_h - padding_size:padded_h, padding_size:padded_w - padding_size] = img[h - 1]

    # Replicate padding for the left and right borders
    padded_img[:, 0:padding_size] = padded_img[:, padding_size:padding_size + 1]
    padded_img[:, padded_w - padding_size:padded_w] = padded_img[:, padded_w - padding_size - 1:padded_w - padding_size]

    return padded_img
```

Listing 2: Mean filter function

```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.
    """
    # Need to implement here
    padded_img = padding_img(img, filter_size)
    h, w = padded_img.shape
    padding_size = filter_size // 2
    for i in range(padding_size, h - padding_size):
        for j in range(padding_size, w - padding_size):
            window = padded_img[i - padding_size: i + padding_size + 1, j - padding_size: j
                + padding_size + 1]
            padded_img[i][j] = np.mean(window)

    filtered_img = padded_img[padding_size:h - padding_size, padding_size:w - padding_size]

    return filtered_img
```

Listing 3: Median filter function

```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.
    """
    # Need to implement here
    padded_img = padding_img(img, filter_size)
    h, w = padded_img.shape
    padding_size = filter_size // 2
    for i in range(padding_size, h - padding_size):
        for j in range(padding_size, w - padding_size):
            window = padded_img[i - padding_size: i + padding_size + 1, j - padding_size: j
                + padding_size + 1]
            padded_img[i][j] = np.median(window)

    filtered_img = padded_img[padding_size:h - padding_size, padding_size:w - padding_size]

    return filtered_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:
            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)

    # MSE = 0 means no noise
    if mse == 0:
        return 100
    max_pixel = 255.0
    psnr = 20 * np.log10(max_pixel / np.sqrt(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 30.524 for the mean filter and 33.637 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 Mean 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:
    data: Nx1: (N, ): 1D numpy array
  returns:
    DFT: Nx1: 1D numpy array
  """
    # You need to implement the DFT here
    N = len(data)
    exp_matrix = np.zeros((N, N), dtype=np.complex_)
    for m in range(N):
        for n in range(N):
            exp_matrix[m][n] = np.exp(-2j * np.pi * m * n / N)

    return np.dot(exp_matrix, data)
```

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_
    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
    """
    # You need to implement the DFT here
    H, W = gray_img.shape
    row_fft = np.zeros((H, W), dtype=np.complex_)
    row_col_fft = np.zeros((H, W), dtype=np.complex_)

    for h in range(H):
        row_fft[h] = np.transpose(DFT_slow(np.transpose(gray_img[h])))

    for w in range(W):
        row_col_fft[:, w] = DFT_slow(row_fft[:, w])

    return row_fft, row_col_fft
```

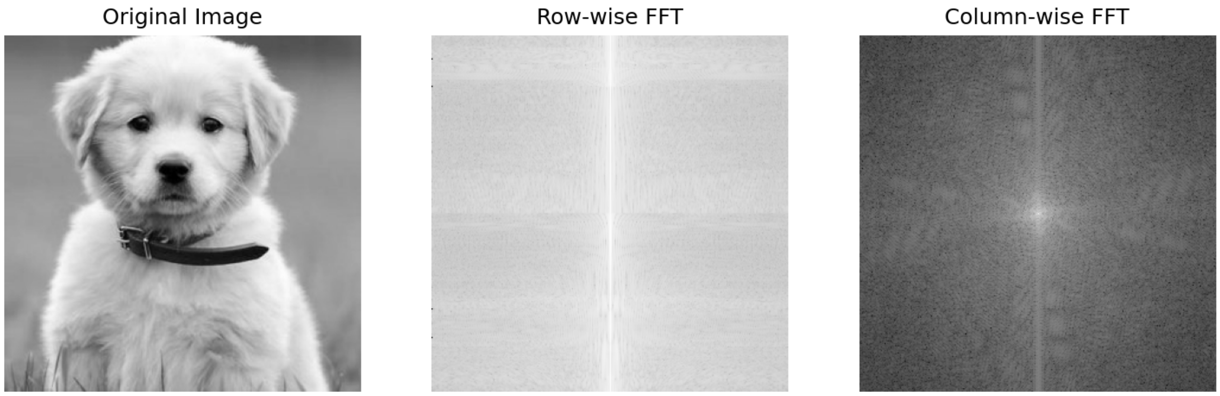

Figure 5: Output for 2D Fourier Transform Exercise

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

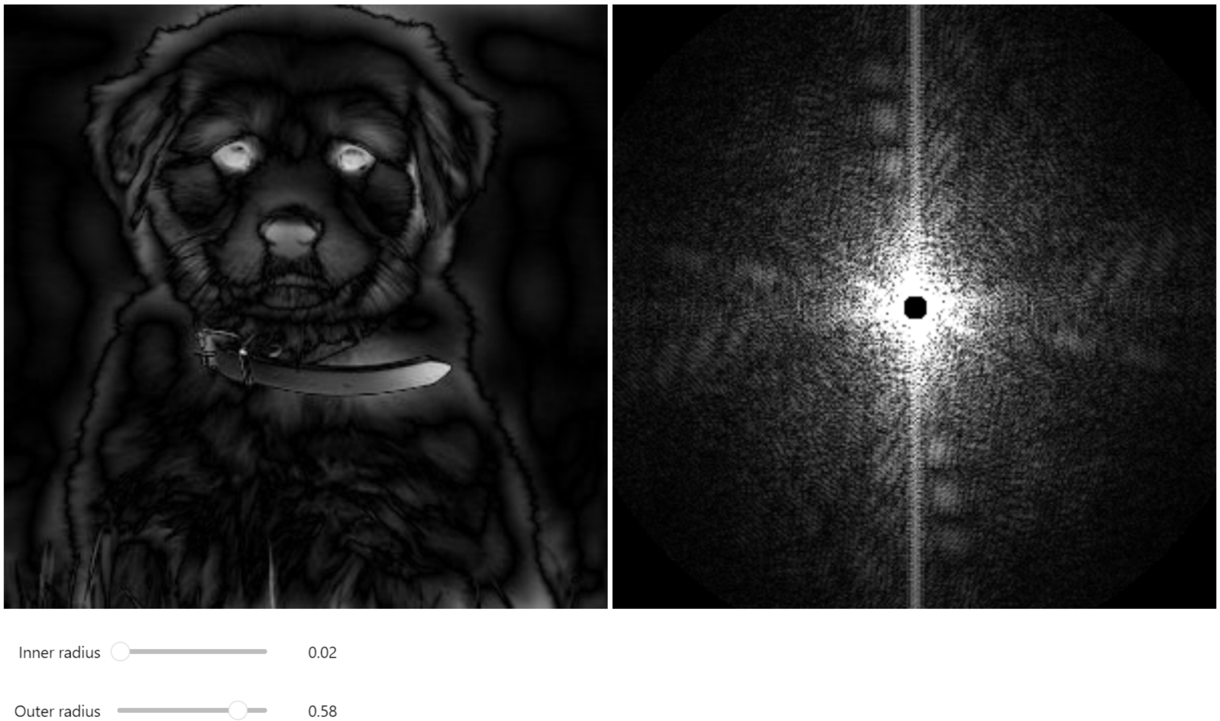

Figure 6: Output for 2D Frequency Removal Exercise

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 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.
  """
  # You need to implement the function
  x1 = np.fft.fftfreq(img1.shape[0])
  y1 = np.fft.fftfreq(img1.shape[1])

  xv1, yv1 = np.meshgrid(x1, y1)
  xv1 = np.fft.fftshift(xv1)
  yv1 = np.fft.fftshift(yv1)

  mask1 = (np.sqrt(xv1**2 + yv1**2) < r)
  mask1 = np.float32(mask1)
  mask2 = np.float32(1 - mask1)

  f_img1, img1_after = filter_frequency(img1, mask1)
  f_img2, img2_after = filter_frequency(img2, mask2)

  f_img = f_img1 + f_img2
  f_img = np.fft.ifftshift(f_img)
```

```
    hybrid = np.fft.ifft2(f_img)

    return np.abs(hybrid)
```

Figure 7: Hybrid Image