INT3404E 20 - Image Processing: Homeworks 2

Le The Quang

April 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

```
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
    """
    pad_size = filter_size // 2
    return np.pad(img, pad_size, mode='edge')
```

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.
    Inputs:
        img: cv2 image: original image
        filter_size: int: size of square filter,
    Return:
        smoothed_img: cv2 image: the smoothed image with mean filter.
    """
    # Padding on the image
    padded_img = padding_img(img, filter_size)
    # Initialize smoothed image with zeros
    smoothed_img = np.zeros_like(img)

```

```python
    rows, cols = img.shape

    # Apply mean filter to each pixel
    for i in range(rows):
        for j in range(cols):
            # Extract neighborhood pixels
            neighbor = padded_img[i:i+filter_size,j:j+filter_size]
            # Assign computed mean value to corresponding pixel
            smoothed_img[i,j] = np.mean(neighbor)
    return smoothed_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.
    """
    # Padding on the image
    padded_img = padding_img(img, filter_size)
    # Initialize smoothed image with zeros
    smoothed_img = np.zeros_like(img)

    rows, cols = img.shape
    # Apply median filter to each pixel
    for i in range(rows):
        for j in range(cols):
            # Extract neighborhood pixels
            neighbor = padded_img[i:i+filter_size,j:j+filter_size]
            # Assign computed median value to corresponding pixel
            smoothed_img[i,j] = np.median(neighbor)
    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

```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
    """
    # Convert images to float32
    gt_img = gt_img.astype(np.float32)
```

```
    smooth_img = smooth_img.astype(np.float32)

    # Calculate the Mean Square Error (MSE)
    mse = np.mean((gt_img - smooth_img) ** 2)

    # Return infinity if MSE is zero (the two images are exactly the same)
    if mse == 0:
        return float('inf')

    max_pixel = 255

    # Calculate the PSNR score
    psnr_score = 20 * math.log10(max_pixel / np.sqrt(mse))

    return psnr_score
```

(c) PSNR is a measure to evaluate the quality of an image after applying image processing techniques such as compression, filtering,etc. It compares an original image to the one that's been processed. Higher PSNR scores (measured in decibels, dB) mean the processed image is closer to the original, indicating better quality.
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 26.202 for the mean filter and 36.977 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

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:
```

Figure 3: Noise image with Mean filter

Figure 4: Noise image with Mean filter

```
     DFT: Nx1: 1D numpy array
    """
    # You need to implement the DFT here
    N = len(data)
    DFT = np.zeros(N, dtype=complex)

    for s in range(N):
        for n in range(N):
            DFT[s] += data[n] * np.exp(-2j * np.pi * s * n / N)

    return DFT
```

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
    """
    H, W = gray_img.shape
    # Conducting a Fourier Transform on each row of the input 2D signal
    row_fft = np.zeros_like(gray_img, dtype=complex)
    for i in range(H):
        row_fft[i, :] = np.fft.fft(gray_img[i, :])
    # Perform a Fourier Transform on each column of the previously obtained result
    row_col_fft = np.zeros_like(row_fft, dtype=np.complex_)
    for j in range(W):
```

```
        row_col_fft[:, j] = np.fft.fft(row_fft[:, j])

    return row_fft, row_col_fft
```

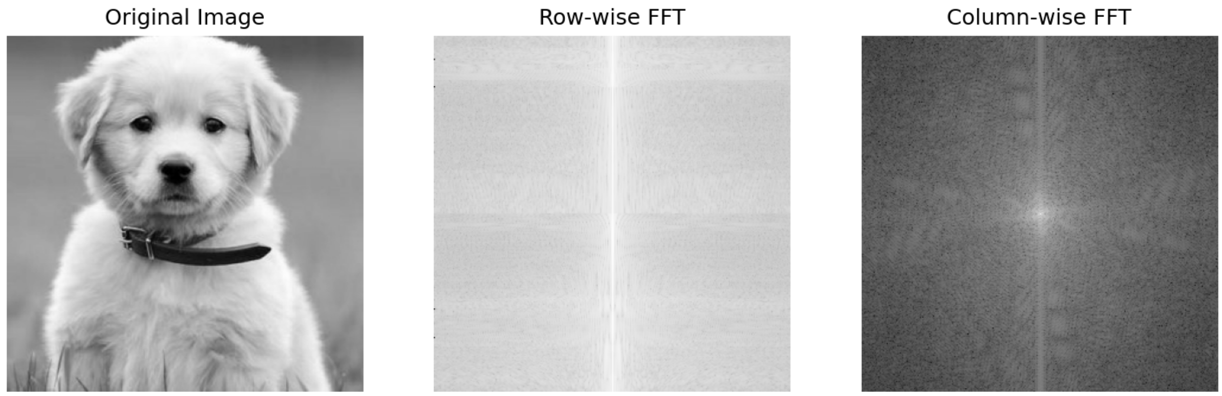

Figure 5: Output for 2D Fourier Transform Exercise

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
  """
  # Step 1: Transform the image to the frequency domain using fft2
  f_img = np.fft.fft2(orig_img)

  # Step 2: Shift the frequency coefficients to the center using fftshift
  f_img_shifted = np.fft.fftshift(f_img)

  # Step 3: Filter the frequency domain representation using the given mask
  f_img_filtered = f_img_shifted * mask

  # Step 4: Shift the frequency coefficients back to their original positions using ifftshift
  f_img_filtered_shifted = np.fft.ifftshift(f_img_filtered)

  # Step 5: Invert the transform using ifft2 to get the filtered image in the spatial domain
  img = np.abs(np.fft.ifft2(f_img_filtered_shifted))
  f_img_filtered = np.abs(f_img_filtered)

  return f_img_filtered, img
```

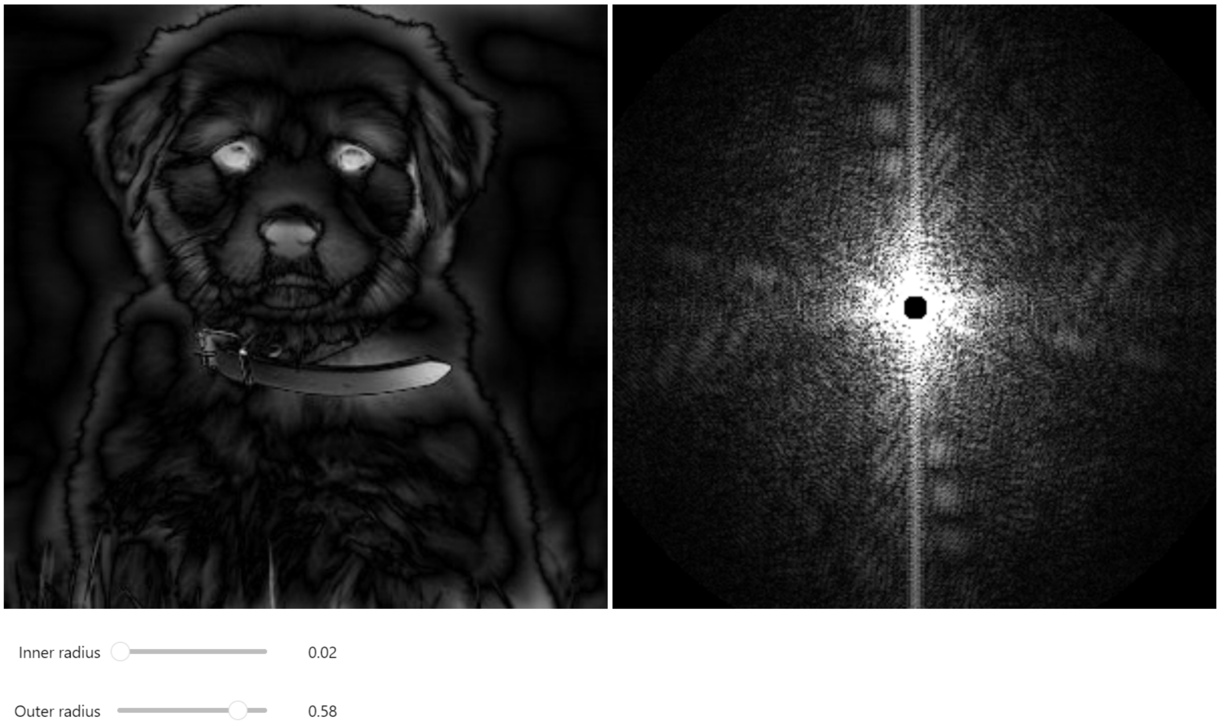

Figure 6: Output for 2D Frequency Removal Exercise

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.
  """

  # 1. Transform the images to the frequency domain using fft2
  img1_fft = np.fft.fft2(img1)
  img2_fft = np.fft.fft2(img2)

  # 2. Shift frequency coefficients to center using fftshift
  img1_fft_shifted = np.fft.fftshift(img1_fft)
  img2_fft_shifted = np.fft.fftshift(img2_fft)

  # 3. Create a mask based on the given radius (r) parameter
  mask = np.zeros_like(img1_fft_shifted, dtype=float)
  rows, cols = img1.shape
  center_x, center_y = rows // 2, cols // 2
  y, x = np.ogrid[:rows, :cols]
  dist_from_center = np.sqrt((x - center_x)**2 + (y - center_y)**2)
  mask = dist_from_center <= r

  # 4. Combine frequency of 2 images using the mask
```

```
    hybrid_fft_shifted = img1_fft_shifted * mask + img2_fft_shifted * (1 - mask)

    # 5. Shift frequency coefficients back using ifftshift
    hybrid_fft = np.fft.ifftshift(hybrid_fft_shifted)

    # 6. Invert transform using ifft2
    hybrid_img = np.abs(ifft2(hybrid_fft))

    return hybrid_img
```

Figure 7: Hybrid Image