INT3404E 20 - Image Processing: Homeworks 2

Ngô Tùng Lâm

1 Image Filtering

1.1 Replicate padding function: `padding_img`

- **Purpose:** Create a padded image using replicate padding. Replicate padding adds the closest values outside the boundary, ensuring that values outside the boundary are set equal to the nearest image border value. The thickness of the padded part depends on the filter size.

- **Implementation:**

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

1.2 Removing noise filter

1.2.1 Box/mean filtering function: `mean_filter`

- **Purpose:** Filtering the image using mean filter. The idea of mean filtering is replacing each pixel value in an image with the mean ('average') value of its neighbors, including itself.

- **Implementation:**

```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)
    smoothed_img = np.zeros_like(img)
```

```
    for i in range(smoothed_img.shape[0]):
        for j in range(smoothed_img.shape[1]):
            smoothed_img[i,j] = np.sum(padded_img[i:i+filter_size,j:j+filter_size]) // (
                filter_size ** 2)
    return smoothed_img
```

- **Result:**

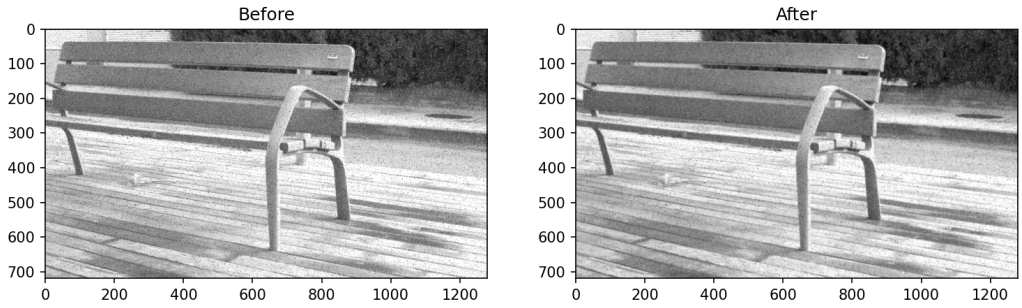

Figure 1: Image after applying mean filter

1.2.2 Median filtering function: `median_filter`

- **Purpose:** Filtering the image using mean filter. The idea of median filtering is replacing each pixel value in an image with the median of its neighbors, including itself. The median is calculated by first sorting all the pixel values from the surrounding neighborhood into numerical order and then replacing the pixel being considered with the middle pixel value.

- **Implementation:**

```
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)
    smoothed_img = np.zeros_like(img)
    for i in range(smoothed_img.shape[0]):
        for j in range(smoothed_img.shape[1]):
            smoothed_img[i,j] = np.median(padded_img[i:i+filter_size,j:j+filter_size])
    return smoothed_img
```

- **Result:**

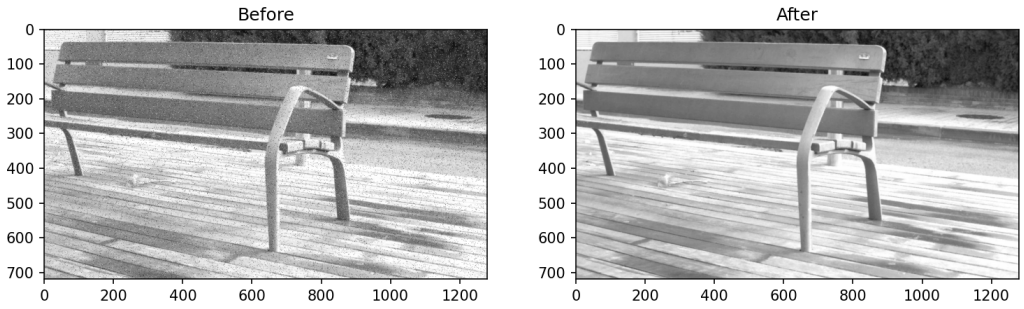

Figure 2: Image after applying median filter

1.3 Peak Signal-to-Noise Ratio (PSNR) metric

The mathematical representation of the PSNR is as follows:

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

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

- **Implementation:**

```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
    """
    # Need to implement here
    mse = np.mean((gt_img - smooth_img) ** 2)
    if mse == 0:
        return 100
    MAX_PIXEL = 255
    return 20 * math.log10(MAX_PIXEL / math.sqrt(mse))
```

- **Result:**
  - PSNR score of mean filtered image: 31.60889963499979
  - PNSR score of median filtered iamge: 37.11957830085524

  For the provided image, the PSNR score of the median filtered image is higher than the score of the mean filtered. So in this case, median filtering is a better choice.

  Median filtering is better at removing noise and retaining detail than median filtering in most cases.

2 Fourier Transform

2.1 1D Fourier Transform

- **Purpose:** Perform the Discrete Fourier Transform (DFT) on a one-dimensional signal, convert the signal to the frequency domain

- **Implementation:**

```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
    """
    # You need to implement the DFT here
    N = len(data)
    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
```

2.2 2D Fourier Transform

- **Purpose:** Perform the Discrete Fourier Transform (DFT) on a two-dimensional signal, convert the signal to the frequency domain
- Idea:
    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.
- **Implementation:**

```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
    """
    # You need to implement the DFT here
    row_fft = np.fft.fft(gray_img, axis=1)
    row_col_fft = np.fft.fft(row_fft, axis=0)

    return row_fft, row_col_fft
```

- **Result:**

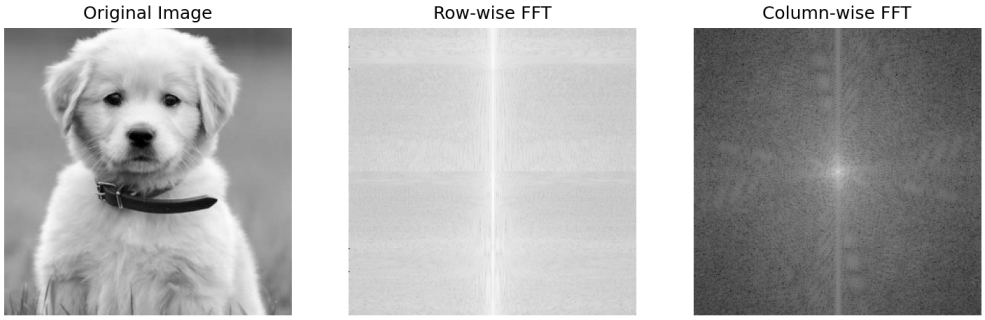

Figure 3: 2D Fourier Transform result

2.3 Frequency Removal Procedure

- **Purpose:** Remove frequency based on the given mask.
- Idea:
  1. Transform using fft2
  2. Shift frequency coefs to center using fftshift
  3. Filter in frequency domain using the given mask
  4. Shift frequency coefs back using ifftshift
  5. Invert transform using ifft2
- **Implementation:**

```
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
  """
  # You need to implement this function
  f_img = np.fft.fft2(orig_img)

  f_shift_img = np.fft.fftshift(f_img)

  f_filtered_img = f_shift_img * mask

  ifftshift_filtered_img = np.fft.ifftshift(f_filtered_img)

  img = np.abs(np.fft.ifft2(ifftshift_filtered_img))
  f_filtered_img = np.abs(f_filtered_img)
  return f_filtered_img, img
```

- **Result:**

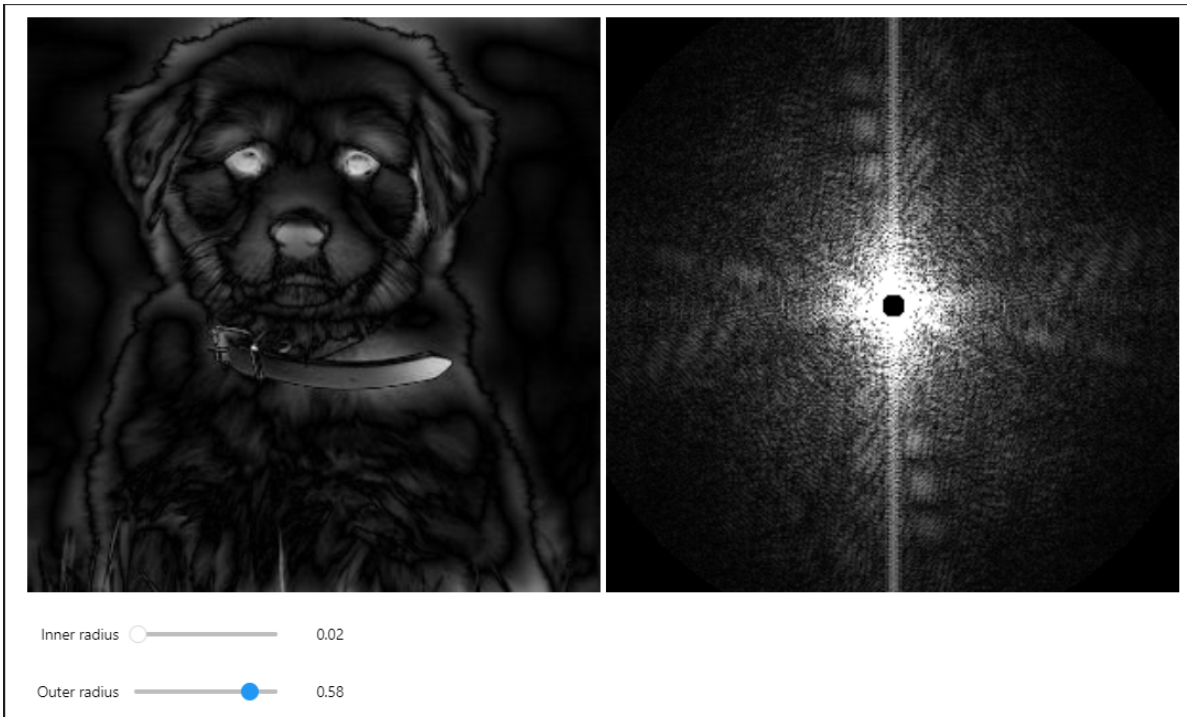

Figure 4: Output for 2D Frequency Removal Exercise

2.4 Creating a Hybrid Image

- **Purpose:** Create a Hybrid Image from 2 given images
- Idea:
  1. Transform using fft2
  2. Shift frequency coefs to center using fftshift
  3. Create a mask based on the given radius (r) parameter
  4. Combine frequency of 2 images using the mask
  5. Shift frequency coefs back using ifftshift
  6. Invert transform using ifft2
- **Implementation:**

```
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
  f_img1 = np.fft.fft2(img1)
  f_img2 = np.fft.fft2(img2)

  f_shift_img1 = np.fft.fftshift(f_img1)
  f_shift_img2 = np.fft.fftshift(f_img2)
```

```
15
    rows, cols = f_shift_img1.shape
    crow, ccol = rows//2 , cols//2
    mask = np.zeros((rows,cols),np.uint8)
    center = [crow, ccol]
20  x, y = np.ogrid[:rows, :cols]
    mask_area = (x - center[0])**2 + (y - center[1])**2 <= r*r
    mask[mask_area] = 1

    f_shift_img1 = f_shift_img1 * mask
25  f_shift_img2 = f_shift_img2 * (1-mask)

    f_shift_hybrid = f_shift_img1 + f_shift_img2
    f_hybrid = np.fft.ifftshift(f_shift_hybrid)
    hybrid_img = np.real(np.fft.ifft2(f_hybrid))
30  return hybrid_img
```

- **Result:**

Figure 5: A Hybrid Image from 2 given image