INT3404E 20 - Image Processing: Homeworks 1

Tran Thi Van Anh

1 Image Filtering

1.1 Implement the following functions in the supplied code file: padding_img, mean_filter, median_filter.

Implement padding_img 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
    """
    pad_width = filter_size // 2
    padded_shape = (
        img.shape[0] + 2 * pad_width,  # height
        img.shape[1] + 2 * pad_width,  # width
        *img.shape[2:]
        )
    padded_img = np.zeros(padded_shape, dtype=img.dtype)
    padded_img[pad_width:pad_width + img.shape[0], pad_width:pad_width + img.shape[1]] = img

    # Replicate the border values to fill in the padded regions
    padded_img[:pad_width, pad_width:-pad_width] = img[0]
    padded_img[-pad_width:, pad_width:-pad_width] = img[-1]
    # Left and right padding
    padded_img[pad_width:-pad_width, :pad_width] = img[:, 0:1]
    padded_img[pad_width:-pad_width, -pad_width:] = img[:, -1:]
    # Corners
    padded_img[:pad_width, :pad_width] = img[0, 0]
    padded_img[:pad_width, -pad_width:] = img[0, -1]
    padded_img[-pad_width:, :pad_width] = img[-1, 0]
    padded_img[-pad_width:, -pad_width:] = img[-1, -1]

    return padded_img
```

Resut:

- Shape of original image: (720, 1280)
- Shape of padded image: (722, 1282)

Implement 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.
    """
    padded_img = padding_img(img, filter_size)
    smoothed_img = np.zeros_like(img)
    for i in range(img.shape[0]):
        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
```

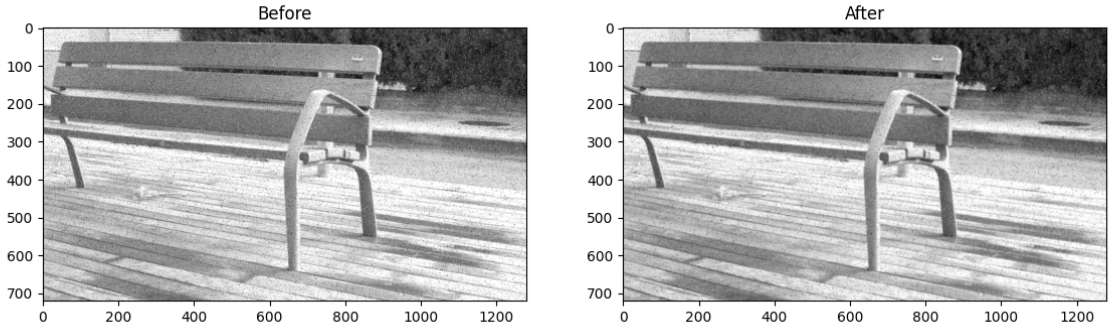

Figure 1: Result of mean_filter function

Implement 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.
        Inputs:
            img: cv2 image: original image
            filter_size: int: size of square filter
        Return:
            smoothed_img: cv2 image: the smoothed image with median filter.
    """
    padded_img = padding_img(img, filter_size)
    smoothed_img = np.zeros_like(img)
    for i in range(img.shape[0]):
```

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

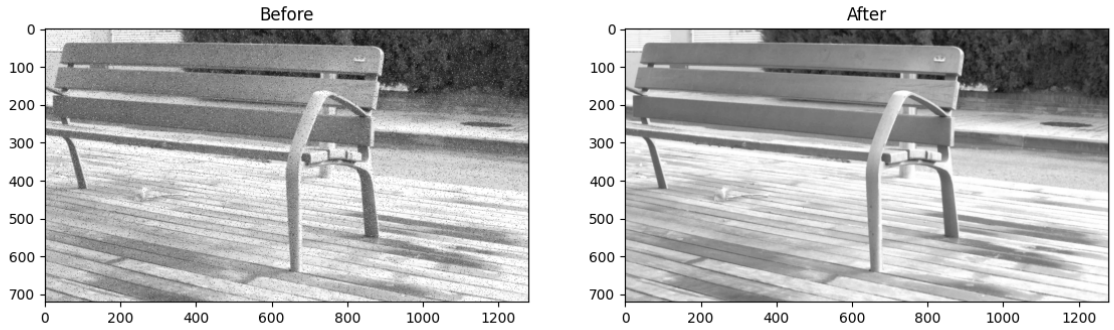

Figure 2: Result of median_filter function

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

```
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
    gt_img = gt_img.astype(np.float64)
    smooth_img = smooth_img.astype(np.float64)

    mse = (np.square(np.subtract(gt_img,smooth_img)).mean())
    max_pixel_value = 255.0
    if mse == 0:
        return float('inf')
    else:
        psnr_score = 10 * np.log10((max_pixel_value ** 2) / mse)
        return psnr_score
```

Result:

- PSNR score of mean filter: 18.2940945653814
- PSNR score of median filter: 17.835212311092135

1.3 Considering the PSNR metrics, which filter should we choose between the mean and median filters for provided images?

Based on the PSNR scores obtained:

- PSNR score of mean filter: 18.29
- PSNR score of median filter: 17.84

The mean filter has a higher PSNR score compared to the median filter. This suggests that, for the provided images, the mean filter better preserves image quality in terms of the PSNR metric. Therefore, we should choose the mean filter.

2 Fourier Transform

2.1 1D Fourier Transform

```
  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 = len(data)
n = np.arange(N)
k = n.reshape((N, 1))
exp_term = np.exp(-2j * np.pi * k * n / N)
DFT = np.dot(exp_term, data)
return DFT
```

2.2 2D Fourier Transform

```
  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
  row_fft = np.zeros((H, W), dtype=np.complex_)
  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, :])

  # Column-wise FFT
  for j in range(W):
      row_col_fft[:, j] = np.fft.fft(row_fft[:, j])

  return row_fft, row_col_fft
```

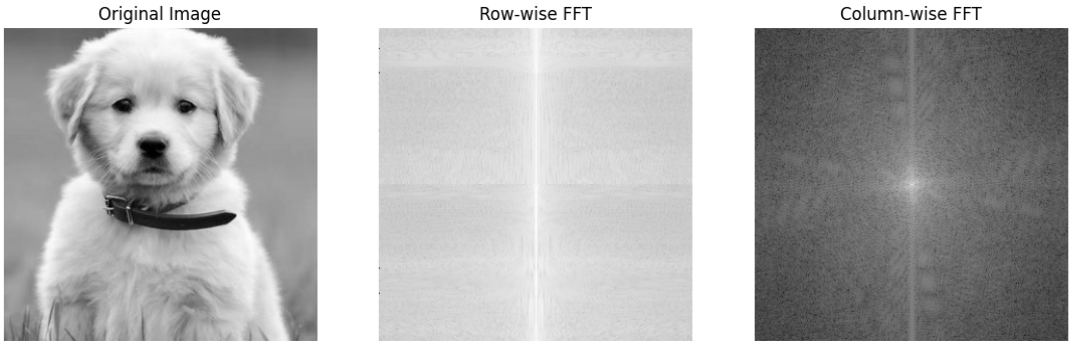

Figure 3: Output of 2D Fourier Transform Exercise

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_shifted = np.fft.fftshift(f_img)

    f_img_filtered = f_img_shifted * mask

    f_img_filtered_shifted = np.fft.ifftshift(f_img_filtered)
    img = np.fft.ifft2(f_img_filtered_shifted)

    return np.abs(f_img_filtered), np.abs(img)
```

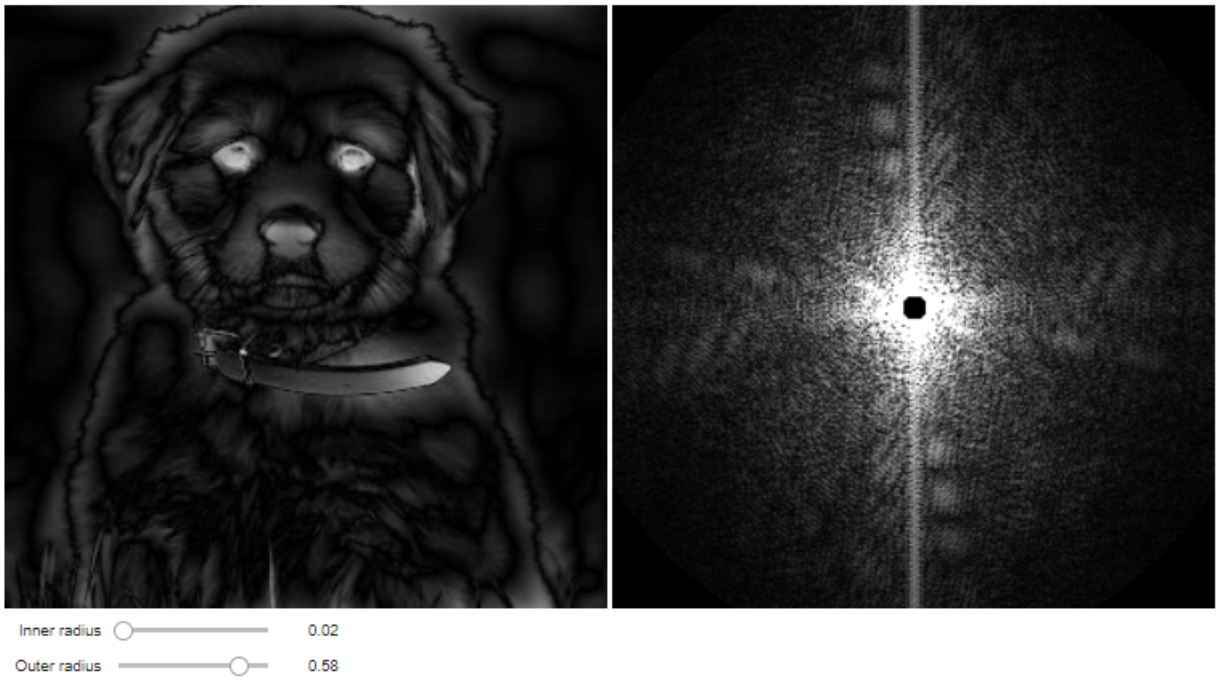

Figure 4: Output of Frequency Removal Procedure

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.
"""
img1_fft = fft2(img1)
img2_fft = fft2(img2)

img1_fft_shifted = fftshift(img1_fft)
img2_fft_shifted = fftshift(img2_fft)

rows, cols = img1.shape
crow, ccol = rows // 2, cols // 2
mask = np.zeros((rows, cols), np.uint8)
mask[crow - r:crow + r, ccol - r:ccol + r] = 1

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

hybrid_fft = ifftshift(hybrid_fft_shifted)

hybrid_img = np.abs(ifft2(hybrid_fft))

hybrid_img = np.clip(hybrid_img, 0, 255)

return hybrid_img.astype(np.uint8)
```

Figure 5: Output of create Hybrid Image function