INT3404E 20 - Image Processing: Homework 2

Bui Minh Thanh

1 Padding Image:

```
def padding_img(img, filter_size=3):
    padded_img = np.pad(np.array(img), pad_width = filter_size // 2, mode = 'edge')
    return padded_img
```

The above function uses np.pad method to add padding to the original image. The width of the padding is set to filter_size // 2 to fit to the filter.

2 Mean Filter:

```
def mean_filter(img, filter_size=3):
    padded_img = padding_img(img, filter_size)
    h, w = img.shape
    smoothed_img = np.zeros(shape = (h, w))
    pad = filter_size // 2
    for i in range(h):
        for j in range(w):
            window = padded_img[i : i + 2 * pad + 1, j : j + 2 * pad + 1]
            smoothed_img[i, j] = np.mean(window)
    smoothed_img = smoothed_img.astype(np.uint8)
    return smoothed_img
```

The above function first add padding to the image, then calculate the convolution of the image with the mean filter by using window sliding method. Then the smoothed image is cast back to np.uint8 integer type.

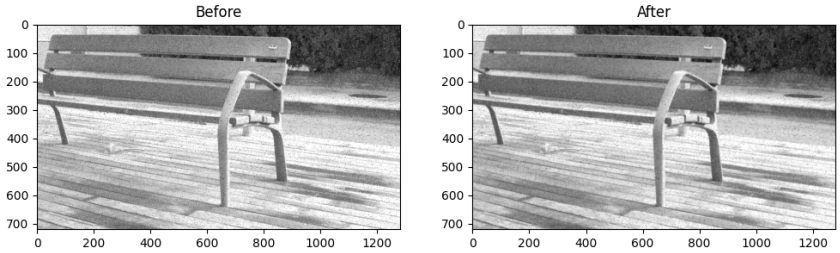

Figure 1: The right image was filtered by mean filter

3 Median Filter:

```
def median_filter(img, filter_size=3):
    padded_img = padding_img(img, filter_size)
    h, w = img.shape
    smoothed_img = np.zeros(shape = (h, w))
    pad = filter_size // 2
    for i in range(h):
        for j in range(w):
            window = padded_img[i : i + 2 * pad + 1, j : j + 2 * pad + 1]
            smoothed_img[i, j] = np.median(window)
    smoothed_img = smoothed_img.astype(np.uint8)
    return smoothed_img
```

The above function first add padding to the image, then calculate the convolution of the image with the median filter by using window sliding method. The smoothed image then is cast back to np.uint8 integer type.

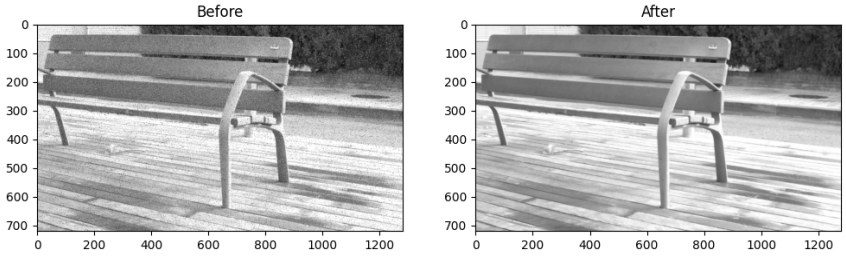

Figure 2: The right image was filtered by median filter

4 PSNR Score:

```
def psnr(gt_img, smooth_img):
    gt_img = np.array(gt_img, dtype = 'double')
    smooth_img = np.array(smooth_img, dtype = 'double')
    mse = np.mean((gt_img - smooth_img) ** 2)
    if mse == 0:
        return 100
    max_sq = 255 * 255
    return 10 * math.log10(max_sq / mse)
```

The above function calculate the psnr score by the given formula. In case mse equals to 0, we set the return value to 100.

5 1-D Discrete Fourier Transform:

```
def DFT_slow(data):
    N = len(data)
    res = []
    for i in range(N):
        tmp = 0
        for k in range(N):
            tmp += data[k] / np.exp(2j * np.pi * i * k / N)
        res.append(tmp)
    return np.array(res)
```

The above function calculate 1-D DFT with quadratic time complexity.

6 2-D Discrete Fourier Transform:

```
def DFT_2D(gray_img):
    img = np.array(gray_img)
    h, w = img.shape
    row_fft = []
    row_col_fft = []
    for i in range(h):
        tmp = np.fft.fft(img[i, 0 : w])
        row_fft.append(tmp)
    row_fft = np.array(row_fft)
    src2 = row_fft.transpose()
    for i in range(w):
        tmp = np.fft.fft(src2[i, 0 : h])
```

```
        row_col_fft.append(tmp)
    row_col_fft = np.array(row_col_fft)
    row_col_fft = row_col_fft.transpose()
    return row_fft, row_col_fft
```

The above function calculate 2-D DFT. First we perform DFT on each row of the image to get the row-wise DFT of the original signal. Then we transpose that matrix and again perform DFT to each row of the resulting image. By perform another transposition of the new resulting image, we get the 2-D DFT.

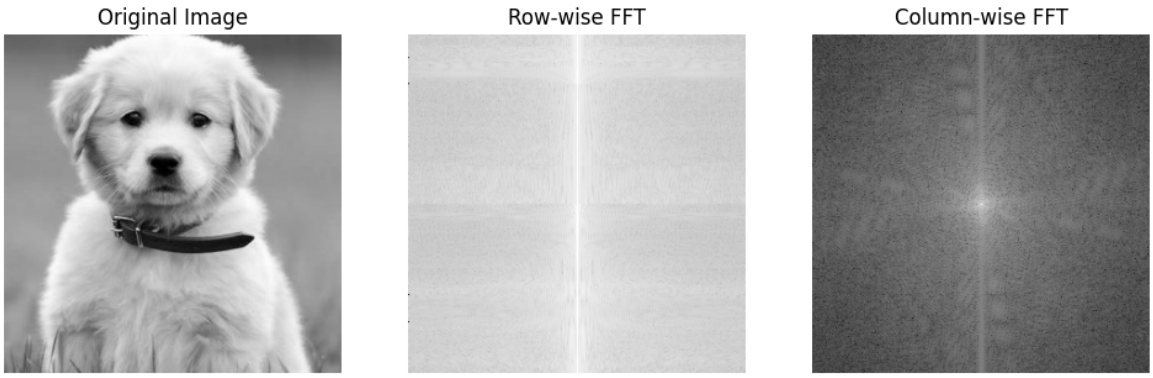

Figure 3: 2-D DFT result

7 Filter Frequency:

```
def filter_frequency(orig_img, mask):
    fft = np.fft.fft2(orig_img)
    shifted_fft = np.fft.fftshift(fft)
    filtered_fft = shifted_fft * mask
    shifted_filtered_fft = np.fft.ifftshift(filtered_fft)
    filtered_img = np.abs(np.fft.ifft2(shifted_filtered_fft))
    return np.abs(filtered_fft), filtered_img
```

The above function first computes the 2D Fourier Transform of the original image by np.fft.fft2 method, then it shifts all zero-frequency component to the center by np.fft.fftshift method. Then, the filter is applied by multiplying the shifted Fourier Transform by mask. After that, it uses np.fft.ifftshift method to shift back the zero-frequency to its original location. Finally, it gets the filtered image by inverse the Fourier Transform by np.fft.ifft2. Because the output we get is a complex-value image, we apply np.abs to transfer to a real-value image.

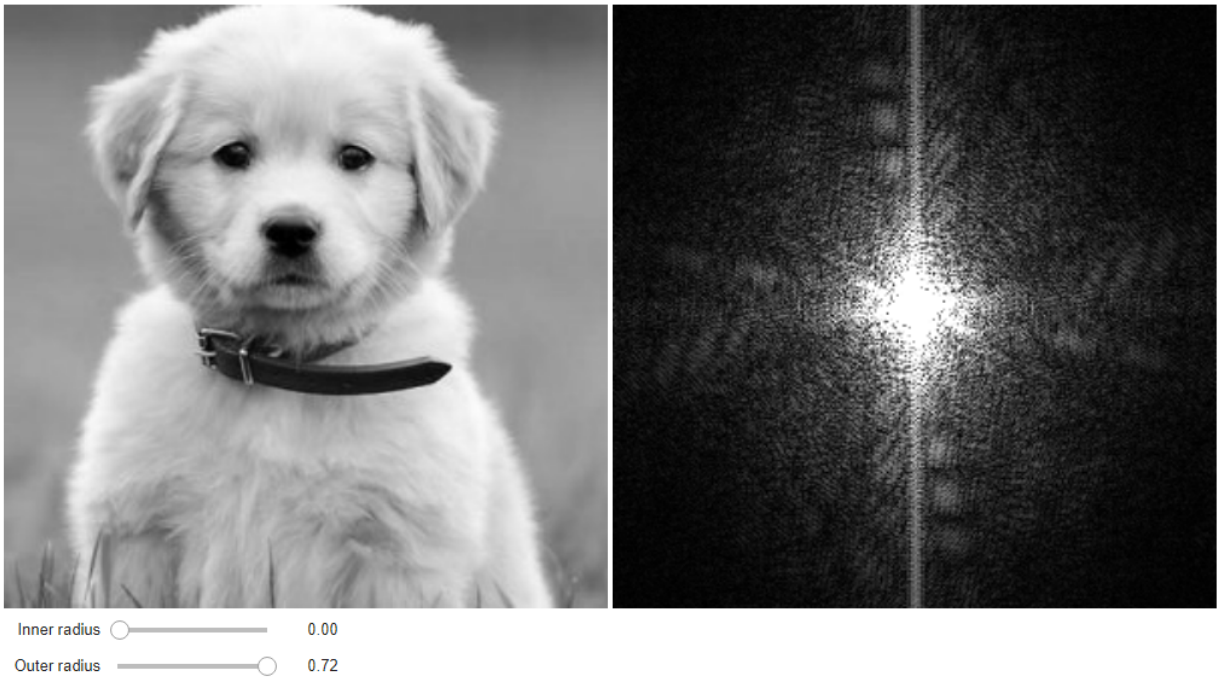

Figure 4: Frequency filter result

8 Hybrid Image:

```python
def create_hybrid_img(img1, img2, r):
    fft1, fft2 = np.fft.fft2(img1), np.fft.fft2(img2)
    shifted1, shifted2 = np.fft.fftshift(fft1), np.fft.fftshift(fft2)
    y, x = np.indices(img1.shape)
    center = np.array(img1.shape) // 2
    distance = ((x - center[0]) * (x - center[0]) + (y - center[1]) * (y - center[1])) ** 0.5
    mask = (distance <= r)
    masked = shifted1 * mask + shifted2 * (~mask)
    shifted_masked = np.fft.ifftshift(masked)
    hybrid = np.abs(np.fft.ifft2(shifted_masked))
    return hybrid
```

The above function creates the hybrid image of two given images. Firstly, it computes 2D Fourier Transform of the images, then it shifts zero-frequency components to the center. After that it uses a circular mask with radius is r and center is the center of the image. This mask have true value inside the circle and false value outside the circle. Then it applies this mask for the first image's shifted Fourier Transform and apply the inverse of this mask for the second image's shifted Fourier Transform. The result is a spectrum that contains low frequencies of the first image and high frequencies of the second image. Finally it uses np.fft.ifftshift and np.fft.ifft2 relatively to invert the transform and we get the resulting image.

Figure 5: Frequency filter result