INT3404E 20 - Image Processing: Homeworks 2

Ngô Lê Hoàng - 22028042

1 Image Filtering

1.1 Replicate padding

```
def padding_img(img, filter_size=3):
    """
    Inputs:
        img: cv2 image: original image
        filter_size: int: size of square filter
    Return:
        padded_img: cv2 image: the padding image
    """
    height, width = img.shape
    # new_height = height + filter_size - 1
    # new_width = width + filter_size - 1

    # must pad each axis (filter_size - 1) padder, and each side in each direction will be (
                                                    filter_size - 1) // 2

    padded_add = (filter_size - 1) // 2
    new_height, new_width = height + padded_add * 2, width + padded_add * 2
    padded_img = np.zeros((new_height, new_width), dtype=np.uint8)
    for x in range(padded_add, new_height - padded_add):
        for y in range(padded_add, new_width - padded_add):
            padded_img[x][y] = img[x - padded_add][y - padded_add]

    # for corner pixel
    for x in range(padded_add + 1):
        for y in range(padded_add + 1):
            padded_img[x][y] = img[0][0]
            padded_img[new_height - x - 1][y] = img[-1][0]
            padded_img[x][new_width - y - 1] = img[0][-1]
            padded_img[new_height - x - 1][new_width - y - 1] = img[-1][-1]

    # for other border pixel
    for x in range(padded_add + 1, new_height - padded_add - 1):
        for y in range(padded_add):
            padded_img[x][y] = img[x - padded_add][0]
            padded_img[x][new_width - y - 1] = img[x - padded_add][-1]

    for y in range(padded_add + 1, new_width - padded_add - 1):
        for x in range(padded_add):
            padded_img[x][y] = img[0][y - padded_add]
            padded_img[new_height - x - 1][y] = img[-1][y - padded_add]

    return padded_img
```

1.2 Mean filters

```
def mean_filter(img, filter_size=3):
    """
    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)

    height, width = img.shape
    smoothed_img = np.zeros((height, width), dtype=float)

    for x in range(height):
        for y in range(width):
            smoothed_img[x][y] += sum(
                padded_img[xx][yy]
                for xx in range(x, x + filter_size)
                for yy in range(y, y + filter_size)
            )
            smoothed_img[x][y] /= filter_size**2

    return smoothed_img
```

Figure 1: Image after using mean filter

1.3 Median filters

```
def median_filter(img, filter_size=3):
    """
    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)


    height, width = img.shape
    smoothed_img = np.zeros((height, width), dtype=float)

    median_index = (filter_size**2) // 2
    for x in range(height):
        for y in range(width):
            pixels = list(
                padded_img[xx][yy]
                for xx in range(x, x + filter_size)
                for yy in range(y, y + filter_size)
            )
            smoothed_img[x][y] = sorted(pixels)[median_index]


    return smoothed_img
```

Figure 2: Image after using median filter

1.4 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
    """
```

```
    # Formula can be found here: https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio
    # or just using formula in hw2_title.file
    height, width = gt_img.shape
    maxi = 255
    mse = sum(
        (gt_img[x][y] - smooth_img[x][y]) ** 2
        for x in range(height)
        for y in range(width)
    ) / (height * width)

    psnr = 20 * math.log10(maxi) - 10 * math.log10(mse)

    return psnr
```

Filter Type	PSNR Score
Mean Filter	18.295335205529753
Median Filter	17.835212311092132

Table 1: PSNR scores of the mean and median filters.

Based on the PSNR score, we can conclude that we should choose the median 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)
    DFT = np.zeros((N,), 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
```

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
    """
    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
```

2.3 Frequency Removal Procedure

```
def filter_frequency(orig_img, 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
    """
    transform = np.fft.fft2(orig_img)
    shift_fft = np.fft.fftshift(transform)

    f_img = shift_fft * mask
    shift_f_img = np.fft.ifftshift(f_img)

    f_img = np.abs(f_img)

    img = np.fft.ifft2(shift_f_img)
    img = np.abs(img)

    return f_img, img
```

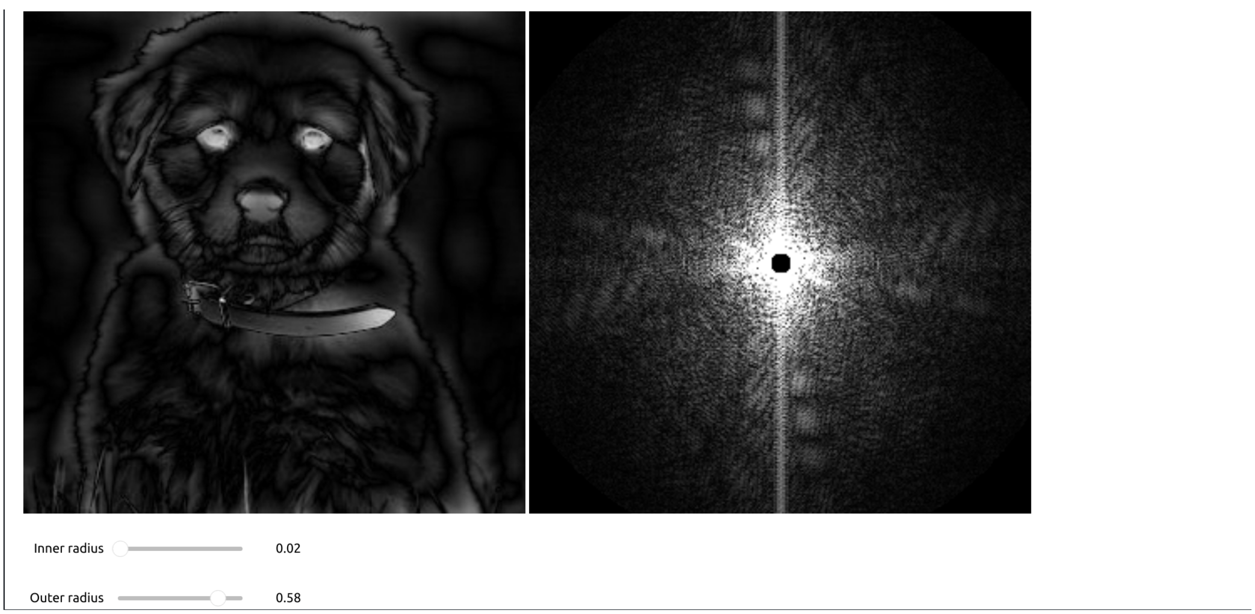

Figure 3: 2D Frequency Removal

2.4 Creating a Hybrid Image

```
def create_hybrid_img(img1, img2, r):
    f1 = fft2(img1)
    f2 = fft2(img2)

    f1s = fftshift(f1)
```

```
    f2s = fftshift(f2)

    rows, cols = img1.shape
    center_row, center_col = rows // 2, cols // 2
    mask = np.ones((rows, cols))
    r2 = r * r
    for i in range(rows):
        for j in range(cols):
            if (i - center_row) ** 2 + (j - center_col) ** 2 < r2:
                mask[i, j] = 0

    f1s = f1s * mask
    f2s = f2s * (1 - mask)
    hybrid = f1s + f2s

    hybrid = ifftshift(hybrid)
    img_hybrid = ifft2(hybrid)
    img_hybrid = np.abs(img_hybrid)

    return img_hybrid
```

Figure 4: Creating hybrid image