INT3404E 20 - Image Processing: Homeworks 2

Nguyễn Văn Thuyên

1 Ex1

Code:

```python
def padding_img(img, filter_size=3):
    height, width = img.shape[:2]
    pad_width = filter_size // 2
    padded_img = np.pad(img, pad_width, mode='edge')
    return padded_img

def mean_filter(img, filter_size=3):
    padded_img = padding_img(img, filter_size)
    smoothed_img = np.zeros_like(img, dtype=np.float32)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            neighborhood = padded_img[i:i+filter_size, j:j+filter_size]
            mean_value = np.mean(neighborhood)
            smoothed_img[i, j] = mean_value

    return smoothed_img.astype(np.uint8)


def median_filter(img, filter_size=3):
    padded_img = padding_img(img, filter_size)
    smoothed_img = np.zeros_like(img, dtype=np.uint8)

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

    return smoothed_img


def psnr(gt_img, smooth_img):
    assert gt_img.shape == smooth_img.shape,
    assert gt_img.dtype == smooth_img.dtype,
    mse = np.mean((gt_img - smooth_img) ** 2)
    max_pixel_value = np.iinfo(gt_img.dtype).max
    psnr_score = 20 * np.log10(max_pixel_value) - 10 * np.log10(mse)
    return psnr_score
```

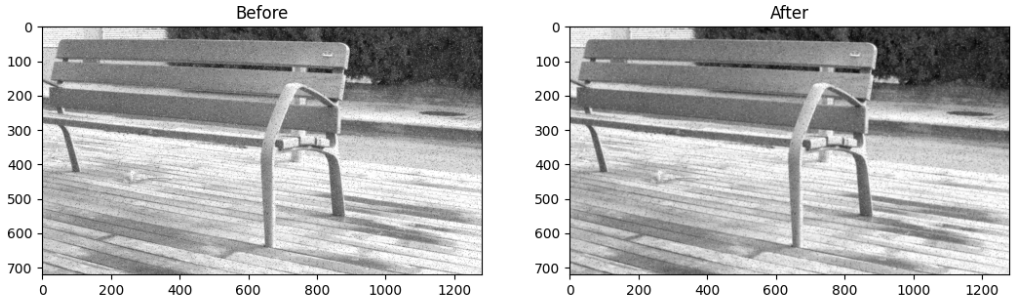

Figure 1: mean$_s moothed - img$

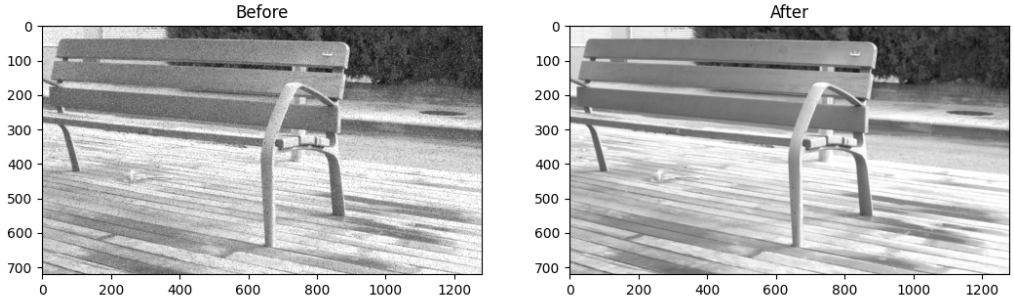

Figure 2: median$_s moothed - img$

PSNR score of mean filter: 31.60889963499979
PSNR score of median filter: 37.11957830085524

2 Ex2

Code:

```
def DFT_slow(data):
    N = len(data)
    DFT = np.zeros(N, dtype=np.complex64)
```

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

    return DFT

def DFT_2D(gray_img):
    row_fft = np.fft.fft2(gray_img)
    row_col_fft = np.fft.fftshift(row_fft)

    return row_fft, row_col_fft
```

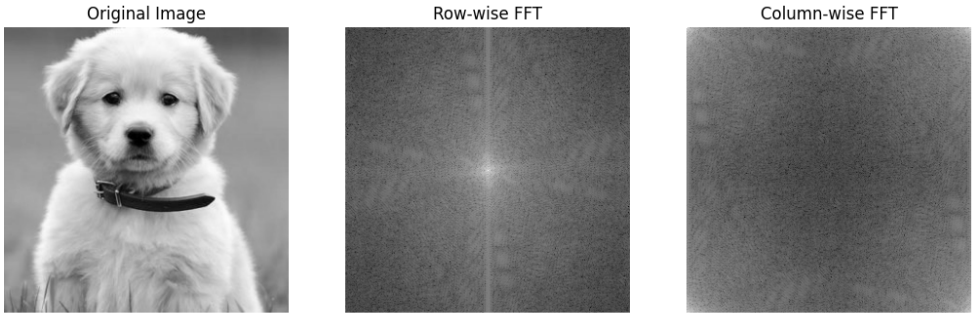

Figure 3: ex212-img

3 Ex3

Code:

```
def filter_frequency(orig_img, mask):
    f_orig_img = np.fft.fft2(orig_img)
    f_shift = np.fft.fftshift(f_orig_img)
    f_shift_masked = f_shift * mask
    f_ishift = np.fft.ifftshift(f_shift_masked)
    img_back = np.fft.ifft2(f_ishift)
    img_back = np.abs(img_back)

    return f_shift, img_back

def create_hybrid_img(img1, img2, r):
  f1 = np.fft.fft2(img1)
  f2 = np.fft.fft2(img2)

```

```
  f1_shift = np.fft.fftshift(f1)
  f2_shift = np.fft.fftshift(f2)

  rows, cols = img1.shape
  crow, ccol = int(rows/2), int(cols/2)
  mask = np.zeros((rows, cols), np.uint8)
  cv2.circle(mask, (crow, ccol), r, 1, thickness=-1)

  f1_shift = f1_shift * mask
  f2_shift = f2_shift * (1-mask)
  f_shift = f1_shift + f2_shift

  f_ishift = np.fft.ifftshift(f_shift)

  img_back = np.fft.ifft2(f_ishift)
  img_back = np.abs(img_back)

  return img_back
```

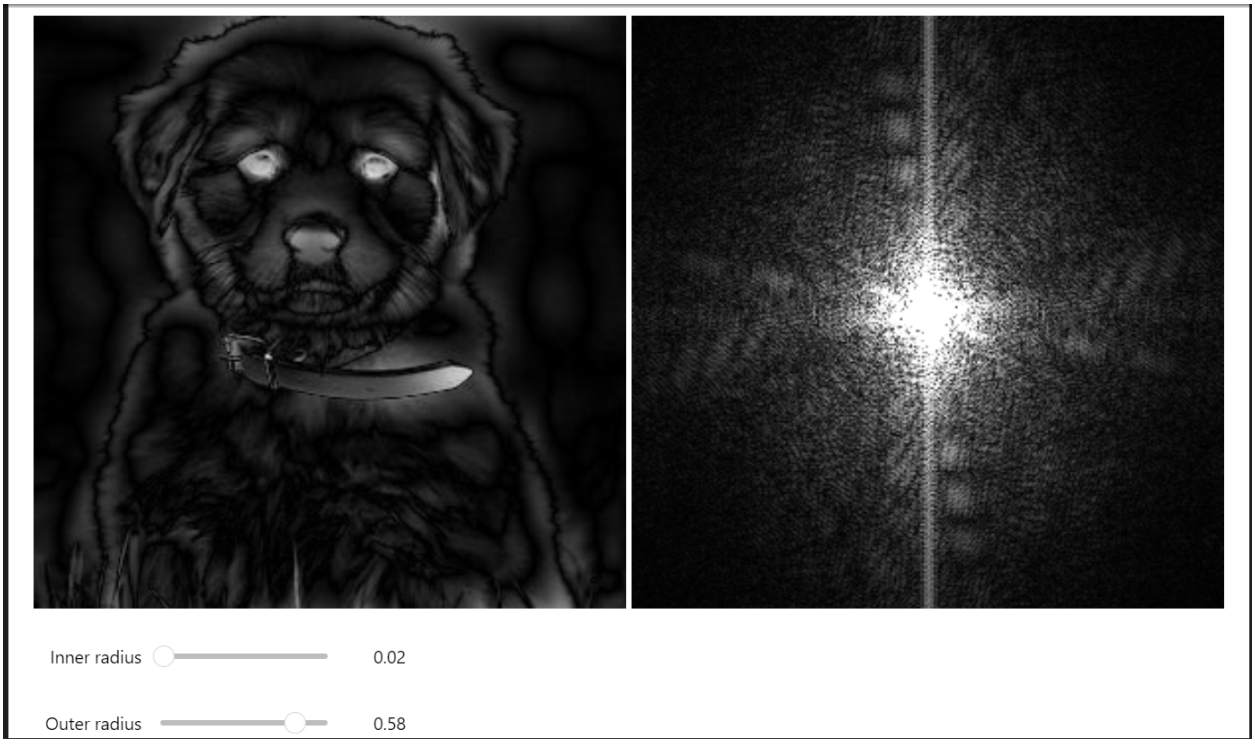

Figure 4: filter-frequency

Figure 5: hybrid-img