INT3404E 20 - Image Processing: Homework 2

Pham Duc Trung - 21021548

19th April 2024

1 *padding_img* function

Code:

```
def padding_img(img, filter_size=3):
  pad_amount = filter_size // 2
  padded_shape = (
      img.shape[0] + 2 * pad_amount,
      img.shape[1] + 2 * pad_amount
  )
  padded_img = np.zeros(padded_shape, dtype=img.dtype)
  padded_img[pad_amount:padded_shape[0] - pad_amount,
             pad_amount:padded_shape[1] - pad_amount] = img
  padded_img[:pad_amount, pad_amount:-pad_amount] = img[0]
  padded_img[-pad_amount:, pad_amount:-pad_amount] = img[-1]
  padded_img[pad_amount:-pad_amount, :pad_amount] = img[:, 0:1]
  padded_img[pad_amount:-pad_amount, -pad_amount:] = img[:, -1:]
  padded_img[:pad_amount, :pad_amount] = img[0, 0]
  padded_img[-pad_amount:, :pad_amount] = img[-1, 0]
  padded_img[:pad_amount, -pad_amount:] = img[0, -1]
  padded_img[-pad_amount:, -pad_amount:] = img[-1, -1]
  return padded_img
```

2 *mean_filter* function

Code:

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

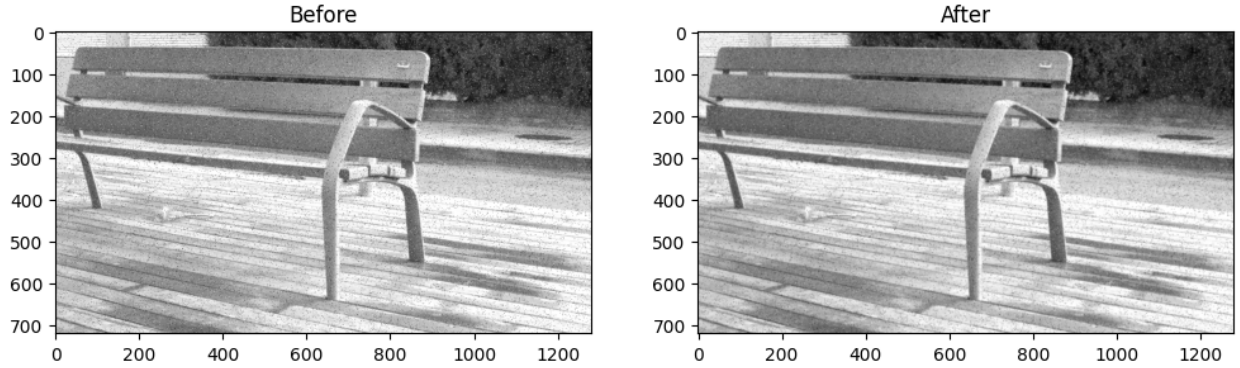

Figure 1: *mean_filter* result

3 *median_filter* function

Code:

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

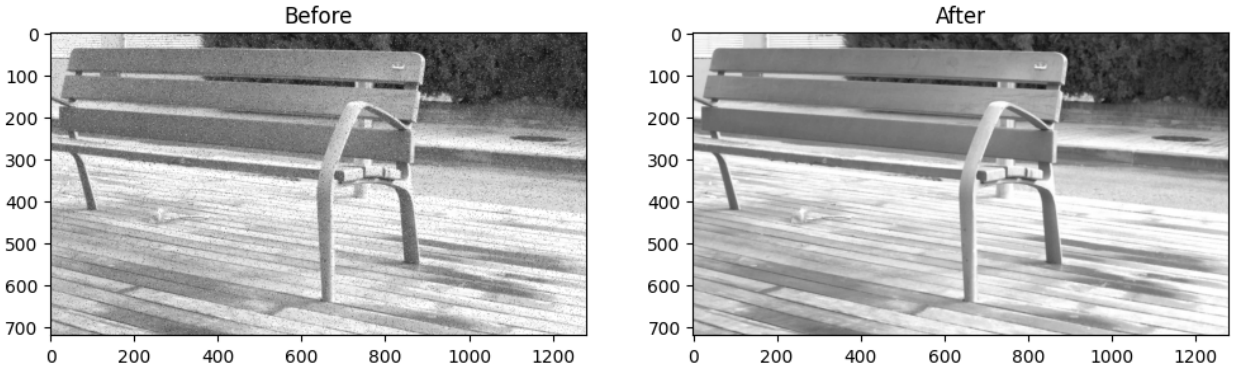

Figure 2: *median_filter* result

4 *psnr* function

Code:

```
def psnr(gt_img, smooth_img):
  gt_img = gt_img.astype(np.float32)
  smooth_img = smooth_img.astype(np.float32)
  mse = np.mean((gt_img - smooth_img) ** 2)
  MAX = np.max(gt_img)
  psnr_score = 10 * np.log10((MAX ** 2) / mse)
  return psnr_score
```

PSNR score of mean filter: 18.29409429382258
PSNR score of median filter: 17.8352122494595

5 *DFT_slow* function

Code:

```
def DFT_slow(data):
  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
```

6 *DFT_2D* function

Code:

```
def DFT_2D(gray_img):
  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
```

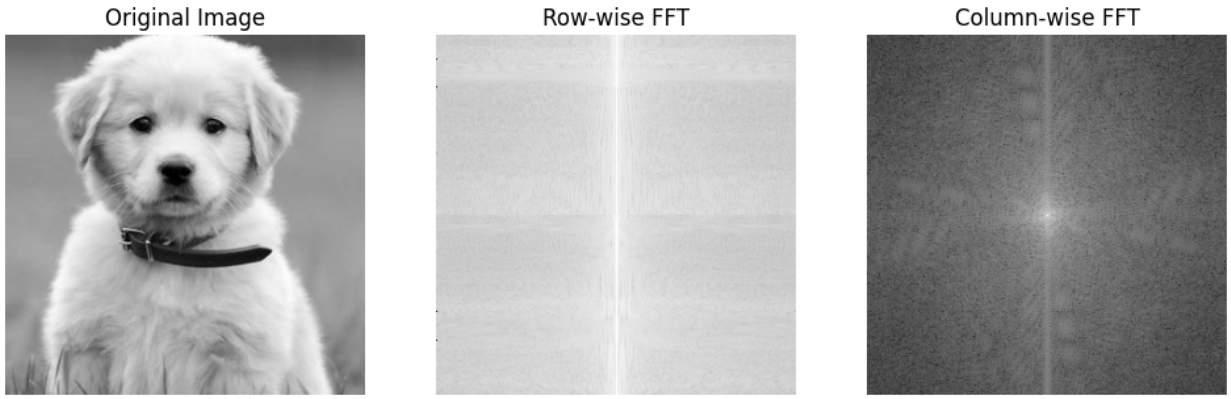

Figure 3: *DFT_2D* result

7 *filter_frequency* function

Code:

```
def filter_frequency(orig_img, 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).real
  return f_img_filtered, img
```

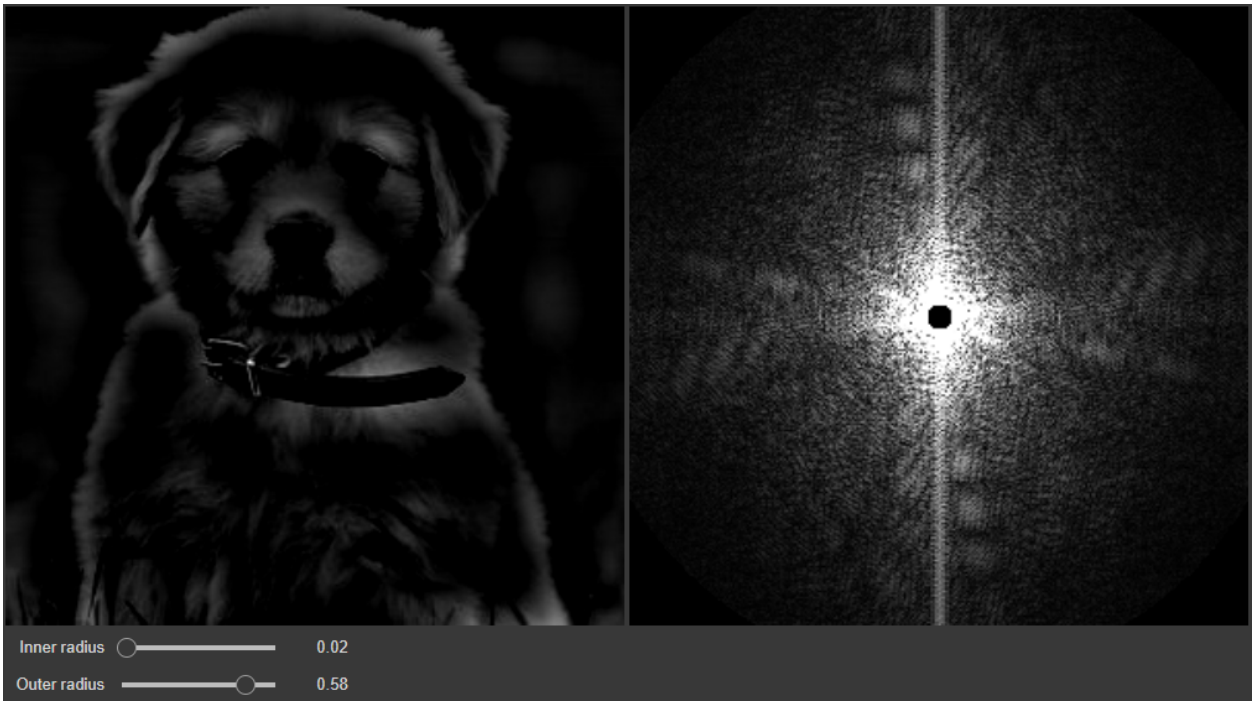

Figure 4: *filter_frequency* result

8 *create_hybrid_img* function

Code:

```
def create_hybrid_img(img1, img2, r):
  img1_fft = np.fft.fft2(img1)
  img2_fft = np.fft.fft2(img2)
  img1_fft_shifted = np.fft.fftshift(img1_fft)
  img2_fft_shifted = np.fft.fftshift(img2_fft)
  rows, cols = img1.shape
  mask = np.zeros((rows, cols), dtype=int)
  center = (rows // 2, cols // 2)
  for i in range(rows):
      for j in range(cols):
          if (i - center[0])**2 + (j - center[1])**2 <= r**2:
              mask[i, j] = 1
  img1_hybrid_fft = img1_fft_shifted * mask
  img2_hybrid_fft = img2_fft_shifted * (1 - mask)
  hybrid_img_fft = img1_hybrid_fft + img2_hybrid_fft
  hybrid_img_fft_shifted = np.fft.ifftshift(hybrid_img_fft)
  hybrid_img = np.fft.ifft2(hybrid_img_fft_shifted).real
  return hybrid_img
```

Figure 5: *create_hybrid_img* result