INT3404E 20 - Image Processing: Homeworks 2

Le Van Tuan

1 Homework Objectives

Here are the detailed objectives of this homework:

1. To achieve a comprehensive understanding of how basic image filters operate.
2. To gain a solid understanding of the Fourier Transform (FT) algorithm.

2 Image Filtering

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

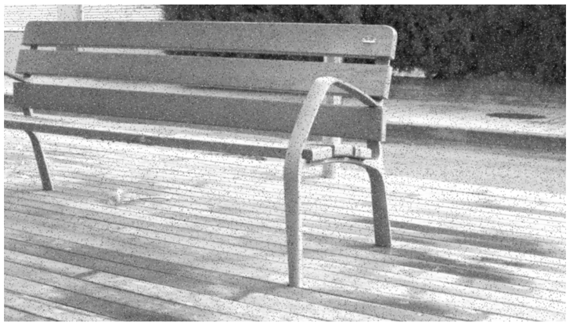


Figure 1: Mean filter

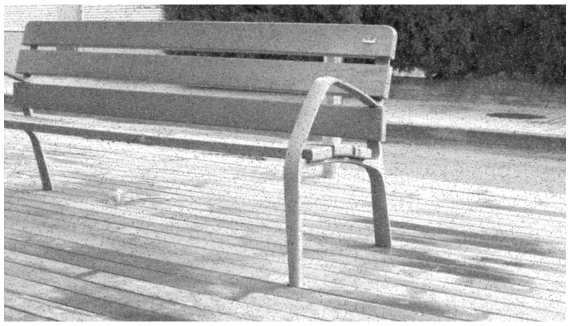


Figure 2: Median filter

Listing 1: Padding Image

```python
def padding_img(img, filter_size=3):
    height, width = img.shape
    pad_size = filter_size
    padded_img = np.zeros((height + 2 * pad_size, width + 2 * pad_size), dtype=img.dtype)
    padded_img[pad_size:pad_size + height, pad_size:pad_size + width] = img
    padded_img[:pad_size, pad_size:pad_size + width] = img[0, :]
    padded_img[pad_size + height:, pad_size:pad_size + width] = img[height - 1,:]
    padded_img[:, :pad_size] = padded_img[:, pad_size:pad_size + 1]
    padded_img[:, pad_size + width:] = padded_img[:, pad_size + width - 1:pad_size + width]
    return padded_img
```

Listing 2: Mean filter

```python
def mean_filter(img, filter_size=3):
    padded_img = padding_img(img, filter_size)
    smoothed_img = np.zeros_like(img)
    pad_size = filter_size
    for i in range(pad_size, padded_img.shape[0] - pad_size):
        for j in range(pad_size, padded_img.shape[1] - pad_size):
            smoothed_img[i - pad_size, j - pad_size] = np.mean(padded_img[
                i - pad_size:i + pad_size + 1,
                j - pad_size:j + pad_size + 1])
    return smoothed_img
```

Listing 3: Median filter

```python
def median_filter(img, filter_size=3):
    padded_img = padding_img(img, filter_size)
    smoothed_img = np.zeros_like(img)
    pad_size = filter_size
    for i in range(pad_size, padded_img.shape[0] - pad_size):
        for j in range(pad_size, padded_img.shape[1] - pad_size):
            smoothed_img[i - pad_size, j - pad_size] = np.median(padded_img[
                i - pad_size:i + pad_size + 1,
                j - pad_size:j + pad_size + 1
            ])
    return smoothed_img
```

2. Put in force the peak signal-to-Noise Ratio (PSNR) metric, in which MAX is the most feasible pixel cost (normally 255 for eight-bit images), and MSE is the suggest square error among the two images. PSNR $= 10 \cdot \log_{10}\left(\frac{MAX^2}{MSE}\right)$

When evaluating filters using PSNR values, the one with a higher PSNR is more powerful in enhancing photograph high-quality.
On this situation, in which the imply clear out achieves a PSNR of 26.202 and the median filter achieves 36.977, the median filter out notably outperforms the imply filter. Hence, prioritizing PSNR scores, the median filter is the most desirable choice for generating higher-fine photographs submit-application.

3 Fourier Transform

3.1 1D Fourier Transform

Implement a function named DFT_slow to perform the Discrete Fourier Transform (DFT) on a one-dimensional signal.

Listing 4: DFT_slow

```python
def DFT_slow(data):
  N = len(data)
  DFT = np.zeros(N, dtype=np.complex128)

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

  return DFT
```

3.2 2D Fourier Transform

The procedure to simulate a 2D Fourier Transform is as follows:

(a) Conducting a Fourier Transform on each row of the input 2D signal. This step transforms the signal along the horizontal axis.

(b) Perform a Fourier Transform on each column of the previously obtained result.

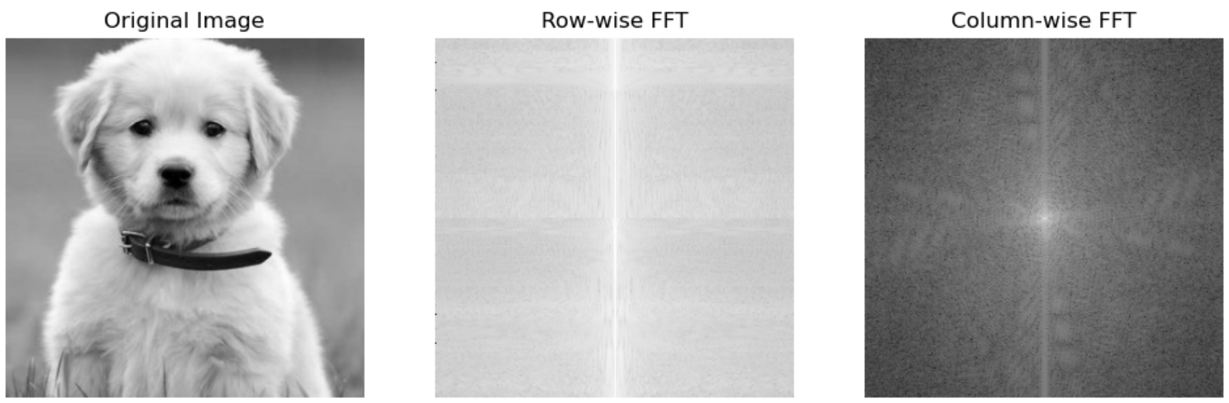


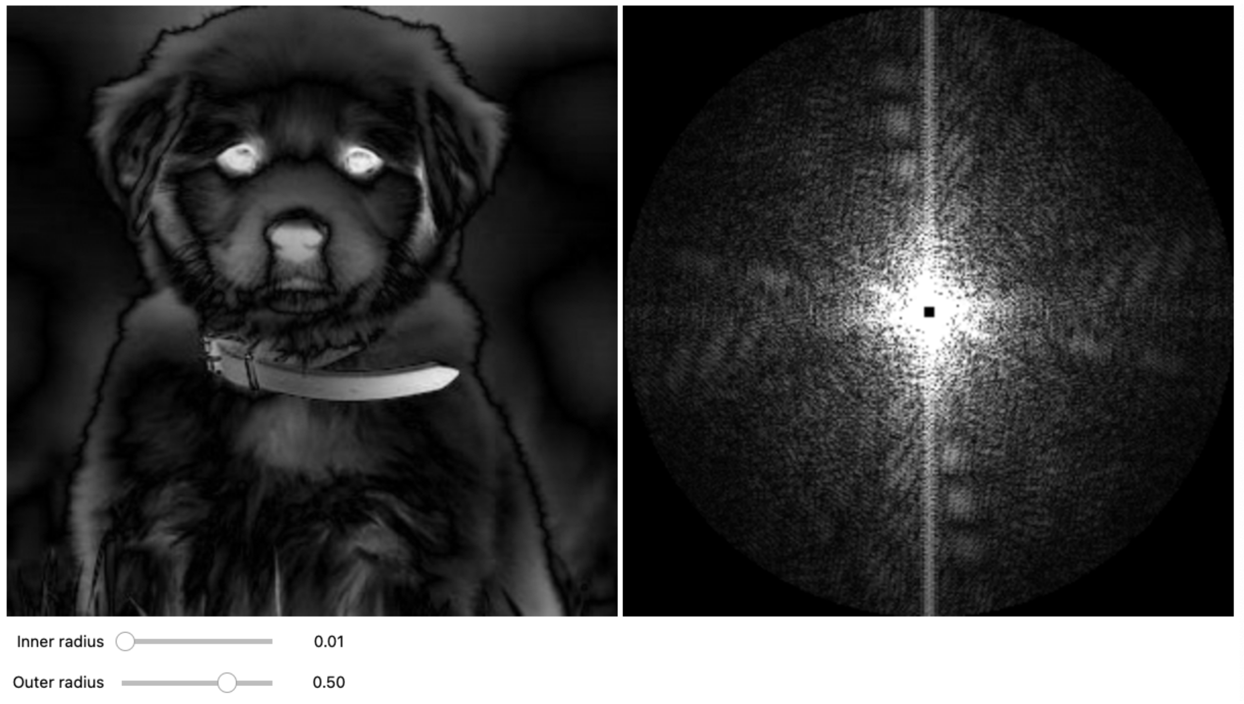
Figure 3: Output for 2D Fourier Transform

Listing 5: Simulate 2D Fourier Transform

```
def DFT_2D(gray_img):
  height, width = gray_img.shape
  row_fft = np.zeros_like(gray_img, dtype=np.complex_)
  row_col_fft = np.zeros_like(gray_img, dtype=np.complex_)
  for i in range(height):
      row_fft[i, :] = np.fft.fft(gray_img[i, :])
  for j in range(width):
      row_col_fft[:, j] = np.fft.fft(row_fft[:, j])
  return row_fft, row_col_fft
```

3.3 Frequency Removal Procedure

`Implement the filter_frequency function in the notebook.`



Inner radius 0.01

Outer radius 0.50

Figure 4: Frequency Removal Tool

Listing 6: DFT_2D

```python
def filter_frequency(orig_img, mask):
  f_img = fft2(orig_img)
  f_img_shifted = fftshift(f_img)
  f_img_filtered = f_img_shifted * mask
  f_img_filtered_array = np.abs(f_img_filtered)
  f_img_filtered_shifted = ifftshift(f_img_filtered)
  img = np.abs(ifft2(f_img_filtered_shifted))

  return f_img_filtered_array, img
```

3.4 Creating a Hybrid Image

Implement the function create_hybrid_img in the notebook.

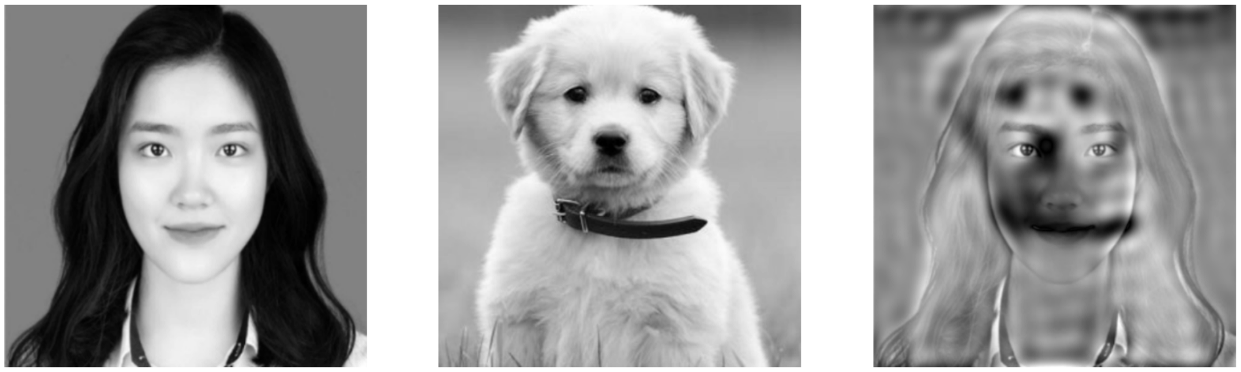


Figure 5: Hybrid image

Listing 7: Creating a Hybrid Image

```python
def create_hybrid_img(img1, img2, r):
  img1_fft = fft2(img1)
  img2_fft = fft2(img2)

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

  mask1 = np.zeros_like(img1_fft_shifted)
  rows, cols = mask1.shape
  center_row, center_col = rows // 2, cols // 2
  y, x = np.ogrid[-center_row:rows - center_row, -center_col:cols - center_col]
  mask1[x**2 + y**2 <= r**2] = 1

  mask2 = (1 - mask1)
  hybrid_fft_shifted = img1_fft_shifted * mask1 + img2_fft_shifted * mask2

  hybrid_fft = ifftshift(hybrid_fft_shifted)

  hybrid_img = np.abs(ifft2(hybrid_fft))

  return hybrid_img
```