INT3404E 20 - Image Processing: Homework 2

Nguyễn Văn Sơn - 22028020

1 Image Filtering Functions

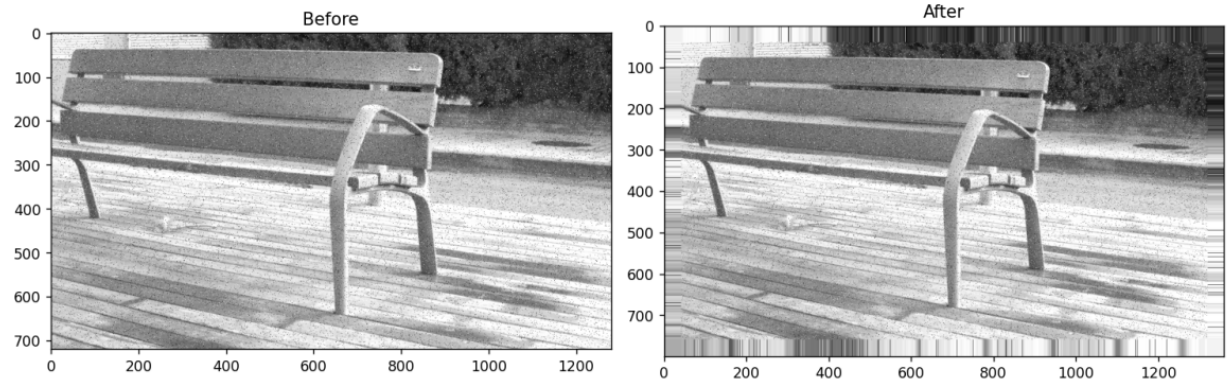
1.1 Implementation

Replicate padding

Code:

```
def padding_img(img, filter_size=3):
  # Need to implement here
    pad_size = filter_size // 2
    return np.pad(img, pad_size, mode='edge')
```

Result:

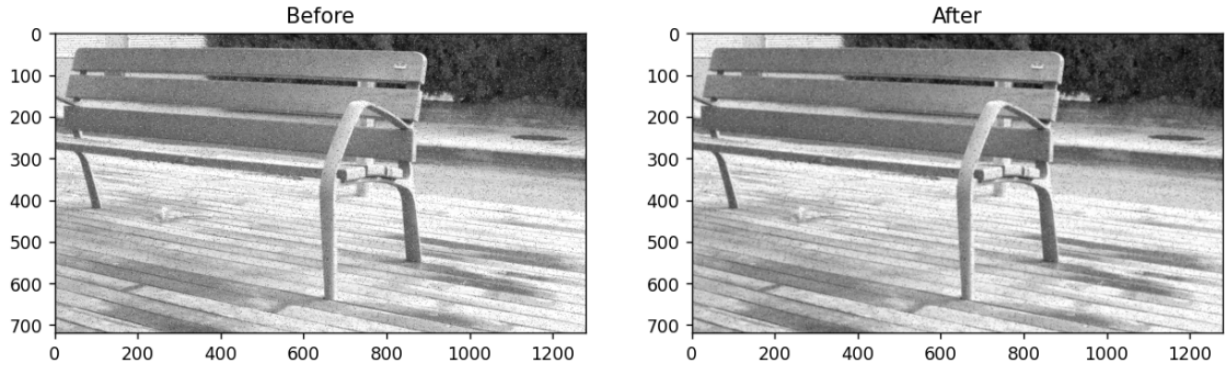


Box/mean filter

Code:

```
def mean_filter(img, filter_size=3):
  # Need to implement here
    padded_img = padding_img(img, filter_size)
    smoothed_img = np.zeros_like(img)
    rows, cols = img.shape

    for i in range(rows):
        for j in range(cols):
            patch = padded_img[i:i+filter_size,j:j+filter_size]
            smoothed_img[i,j] = np.mean(patch)
    return smoothed_img
```

Result:

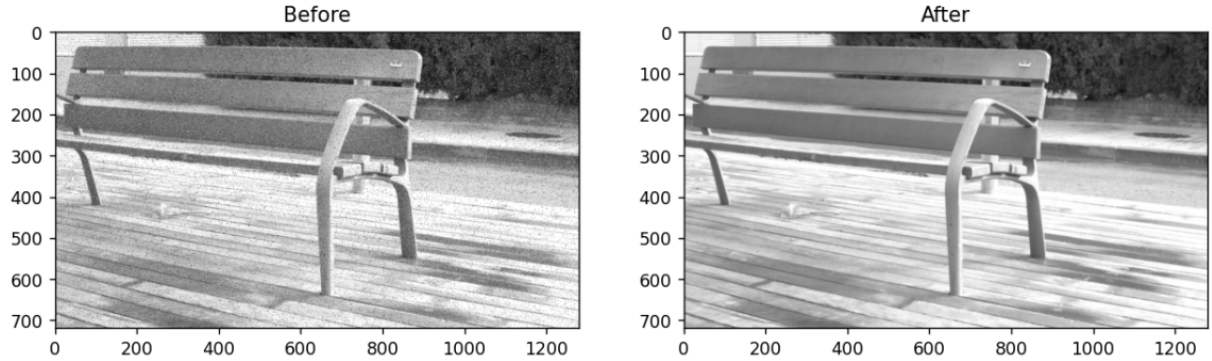


Median filter

Code:

```
def median_filter(img, filter_size=3):
    padded_img = padding_img(img, filter_size)
    smoothed_img = np.zeros_like(img)
    rows, cols = img.shape

    for i in range(rows):
        for j in range(cols):
            patch = padded_img[i:i+filter_size,j:j+filter_size]
            smoothed_img[i,j] = np.median(patch)
    return smoothed_img
```

Result:



1.2 Evaluation

Code:

```
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
    """

    # Calculate the Mean Square Error (MSE)
    mse = np.mean((gt_img - smooth_img) ** 2)
```

```
    # If MSE is zero, the two images are exactly the same, so return infinity
    if mse == 0:
        return float('inf')

    # Check the image depth and decide the maximum pixel value
    max_pixel = 0
    if gt_img.dtype == np.uint8:
        max_pixel = 255.0
    elif gt_img.dtype == np.uint16:
        max_pixel = 65535.0
    else:
        max_pixel = 1.0

    # Calculate the PSNR score
    psnr_score = 20 * math.log10(max_pixel / math.sqrt(mse))

    return psnr_score
```

PNSR scores for the filters are:
1. Mean filter: 31.60889963499979
2. Median filter: 37.11957830085524
→ Median filter is a better choice for provided image.

2 Fourier Transform

2.1 1D Fourier Transform

Code:

```
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
  """
  # You need to implement the DFT here
  N = len(data)
  DFT = np.zeros(N, dtype=complex)
  for s in range(N):
    for n in range(N):
      WN = np.exp(-2j * np.pi * s * n / N)
      DFT[s] += data[n] * WN
  return DFT
```

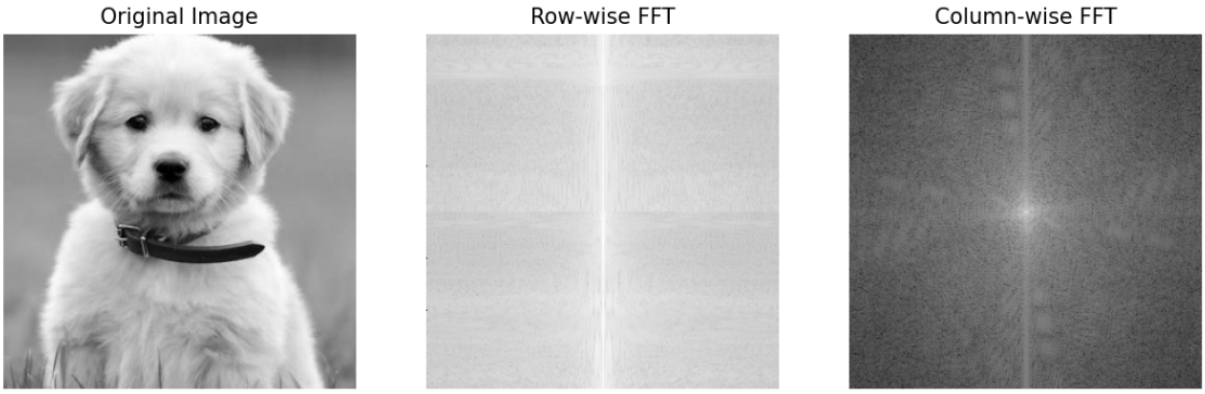
2.2 2D Fourier Transform

Code:

```
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
    """
```

```
    H, W = gray_img.shape
    # Conducting a Fourier Transform on each row of the input 2D signal
    row_fft = np.zeros_like(gray_img, dtype=np.complex_)
    for i in range(H):
        row_fft[i, :] = np.fft.fft(gray_img[i, :])

    # Perform a Fourier Transform on each column of the previously obtained result
    row_col_fft = np.zeros_like(row_fft, dtype=np.complex_)
    for j in range(W):
        row_col_fft[:, j] = np.fft.fft(row_fft[:, j])

    return row_fft, row_col_fft
```

Result:



2.3 Fourier Transform Applications

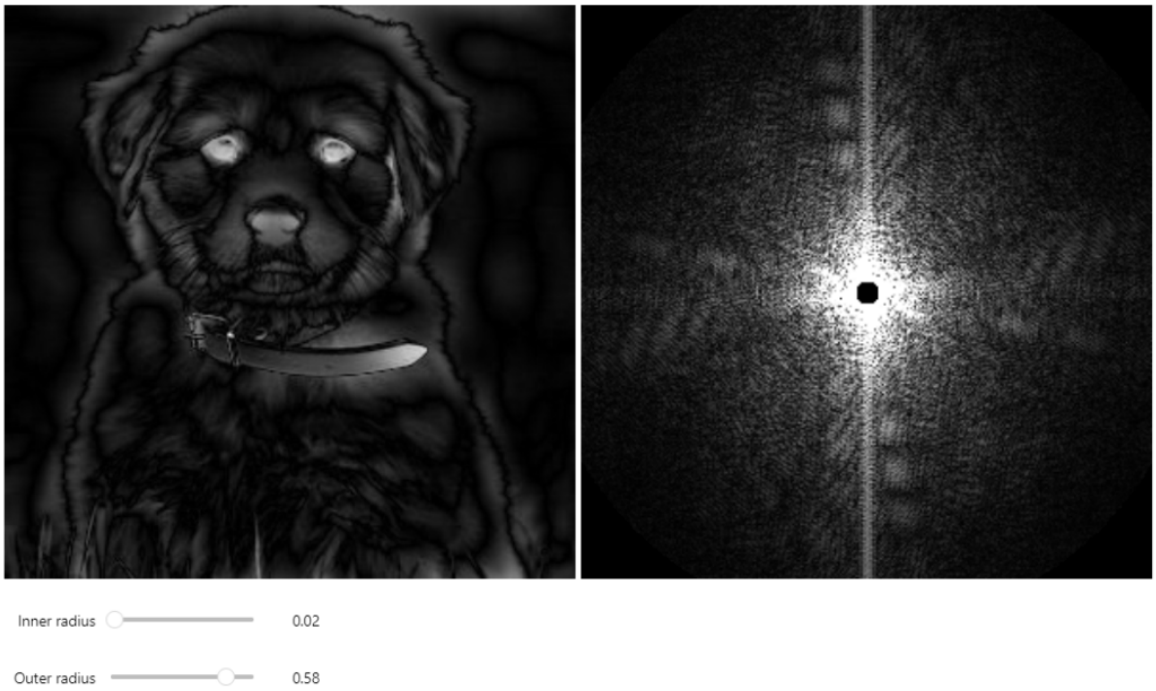
2.3.1 Frequency Removal Function

Code:

```
def filter_frequency(orig_img, mask):
  """
  You need to remove frequency based on the given 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
  """
  # 1. Transform using fft2
  img_fft = np.fft.fft2(orig_img)

  # 2. Shift frequency coefs to center using fftshift
  img_ffshift = np.fft.fftshift(img_fft)

  # 3. Filter in frequency domain using the given mask
  f_img = img_ffshift * mask

  # 4. Shift frequency coefs back using ifftshift
  f_img_shift = np.fft.ifftshift(f_img)

  # 5. Invert transform using ifft2
  img = np.fft.ifft2(f_img_shift)

  # 6. Extract absolute value for visualization
```

```
    img = np.abs(img)
    f_img = np.abs(f_img)

    return f_img, img
```

Result:



2.4 Creating a Hybrid Image

Code:

```
def create_hybrid_img(img1, img2, r):
    # Transform using fft2
    img1_fft = np.fft.fft2(img1)
    img2_fft = np.fft.fft2(img2)

    # Shift frequency coefs to center using fftshift
    img1_fftshift = np.fft.fftshift(img1_fft)
    img2_fftshift = np.fft.fftshift(img2_fft)

    # Create a mask based on the given radius (r) parameter
    mask = np.zeros(img1_fftshift.shape)
    for i in range(img1_fftshift.shape[0]):
        for j in range(img1_fftshift.shape[1]):
            dist = np.sqrt((i - img1_fftshift.shape[0] // 2) ** 2 + (j - img1_fftshift.shape[1] // 2) ** 2)
            if dist <= r:
                mask[i, j] = 1

    # Combine frequency of 2 images using the mask
    hybrid_fft = img1_fftshift * mask + img2_fftshift * (1 - mask)

    # Shift frequency coefs back using ifftshift
    hybrid_fftshift = np.fft.ifftshift(hybrid_fft)

    # Invert transform using ifft2
    hybrid_image = np.fft.ifft2(hybrid_fftshift)
```

```
    return np.abs(hybrid_image)
```

Result:



3 Source Code

Source code and images are placed in GitHub: link