INT3404E 20 - Image Processing: Homeworks 2

Chu Ngoc Vuong

1 Image Filtering

The exercise will help to understand basic image filters by manipulating box/mean and median filters by implement the first two problems in the provided Python script file (ex1.py)

1.1 Replicate padding

```python
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

    filter_size //= 2

    padded_img = np.zeros((height + 2 * filter_size, width + 2 * filter_size), dtype=img.dtype)

    padded_img[filter_size:filter_size + height, filter_size:filter_size + width] = img

    # Padding top
    padded_img[:filter_size, filter_size:filter_size + width] = img[0]
    #Padding bottom
    padded_img[-filter_size:, filter_size:filter_size + width] = img[-1]

    for k in range (filter_size):
    # Padding left
      padded_img[:, k] = padded_img[:, filter_size]
    # Padding right
      padded_img[:, - k -1] = padded_img[:, - filter_size - 1]

    return padded_img
```

Result:

```
1 sample = np.random.randint(10, size=(5, 5))
2 sample_pad = padding_img(sample, 3)
3 sample_pad
```

```
array([[5, 5, 4, 2, 1, 7, 7],
       [5, 5, 4, 2, 1, 7, 7],
       [6, 6, 3, 5, 5, 4, 4],
       [8, 8, 3, 3, 6, 5, 5],
       [6, 6, 9, 3, 4, 7, 7],
       [3, 3, 0, 9, 5, 3, 3],
       [3, 3, 0, 9, 5, 3, 3]])
```

Figure 1: Replicate padding function result

1.2 Box/mean filter

```
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.
    """
  # Need to implement here

    height, width = img.shape

    padded_img = padding_img(img, filter_size)

    # Assume stride = 1

    kernel = np.ones((filter_size, filter_size), dtype=np.float32) / (filter_size ** 2)

    filtered_img = np.zeros(img.shape, dtype = np.float32)

    for i in range (height):
      for j in range (width):
        neighbor = padded_img[i:i+filter_size, j:j+filter_size]
        filtered_img[i, j] = np.sum(neighbor * kernel)

    filtered_img = filtered_img.astype(img.dtype)
    return filtered_img
```

Result:

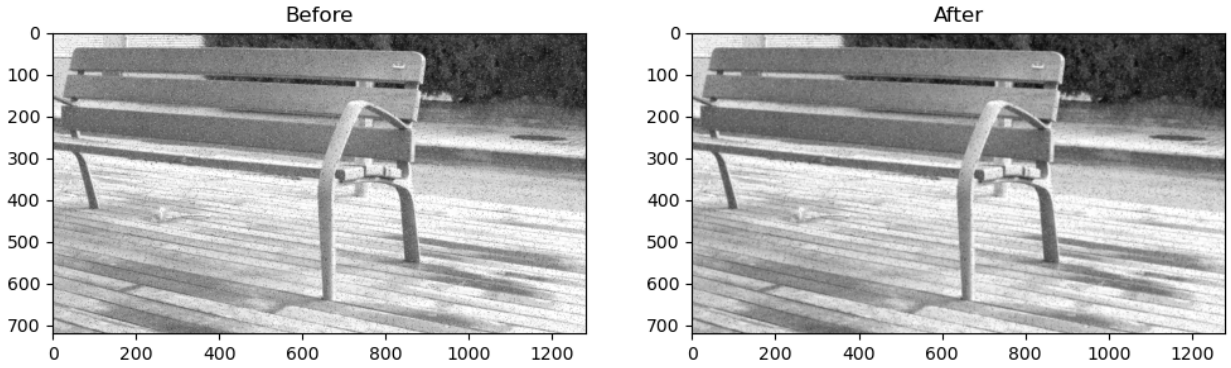

Figure 2: Mean filter function result

1.3 Median filter

```
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.
    """
  # Need to implement here
    height, width = img.shape
```

```
    padded_img = padding_img(img, filter_size)

    # Assume stride = 1

    kernel = np.ones((filter_size, filter_size), dtype=np.float32) / (filter_size ** 2)

    filtered_img = np.zeros(img.shape, dtype = np.float32)

    for i in range (height):
      for j in range (width):
        neighbor = padded_img[i:i+filter_size, j:j+filter_size]

        filtered_img[i, j] = np.median(neighbor)

    filtered_img = filtered_img.astype(img.dtype)

    return filtered_img
```

Result:

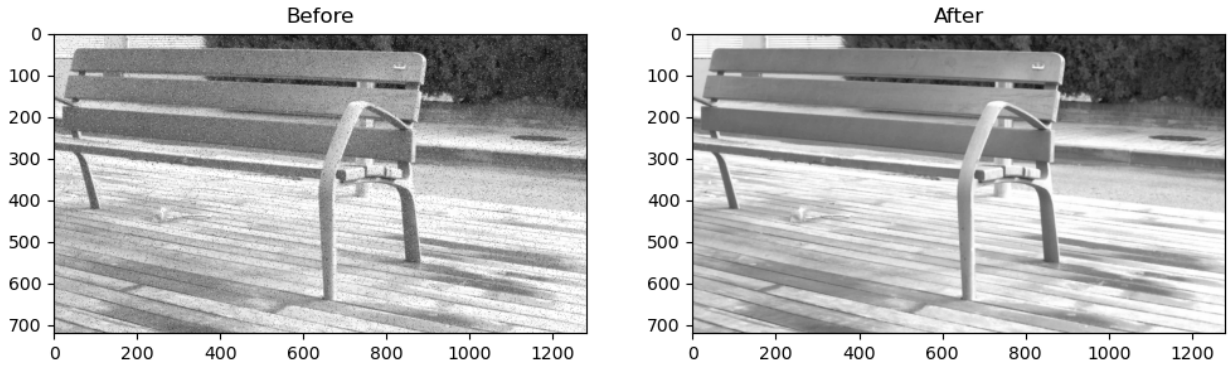

Figure 3: Median filter function result

2 Fourier Transform

In this exercise, we will implement the Discrete Fourier Transform (DFT) algorithm from scratch. The goal is to familiarize the fundamental concepts and procedural steps involved in applying this algorithm. first two problems wil be done in the provided Python script file (ex212.py), and the remaining two problems will be completed in the provided Jupyter notebook (ex234.ipynb).

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, dtype=np.complex128)

  for s in range (N):
    for n in range (N):
       DFT[s] += data[n] * np.exp(-1j * 2 * np.pi * s * n / N)
      #  print(DFT[s])
    DFT[s] /= N

    """
    Note:
    Some Discrete Fourier Transform formula doesn't include the 1/N part
    Therefore, removing the DFT[s] /= N part will match this fucntion result to the np.fft.fft()
    """

  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
    """

    img_row_fft = np.fft.fft(gray_img, axis = 1)
    img_col_fft = np.fft.fft(img_row_fft, axis = 0)

    return img_row_fft, img_col_fft
```

Result:

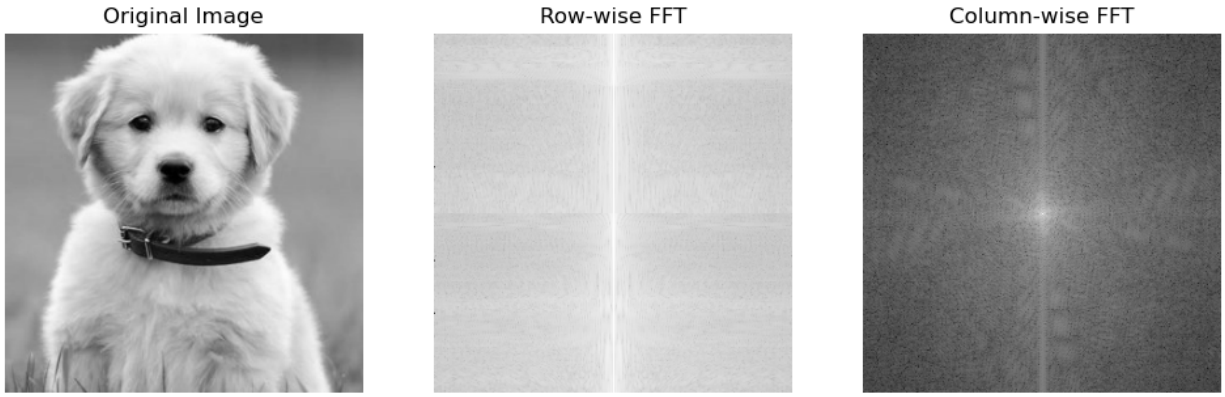

Figure 4: 2D Fourier Transform result

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
  """

  f_img = np.fft.fft2(orig_img)
  f_img = np.fft.fftshift(f_img)
  f_img = f_img * mask
  img = np.fft.ifftshift(f_img)
  img = np.fft.ifft2(f_img)

  return np.abs(f_img), np.abs(img)
```

Result:

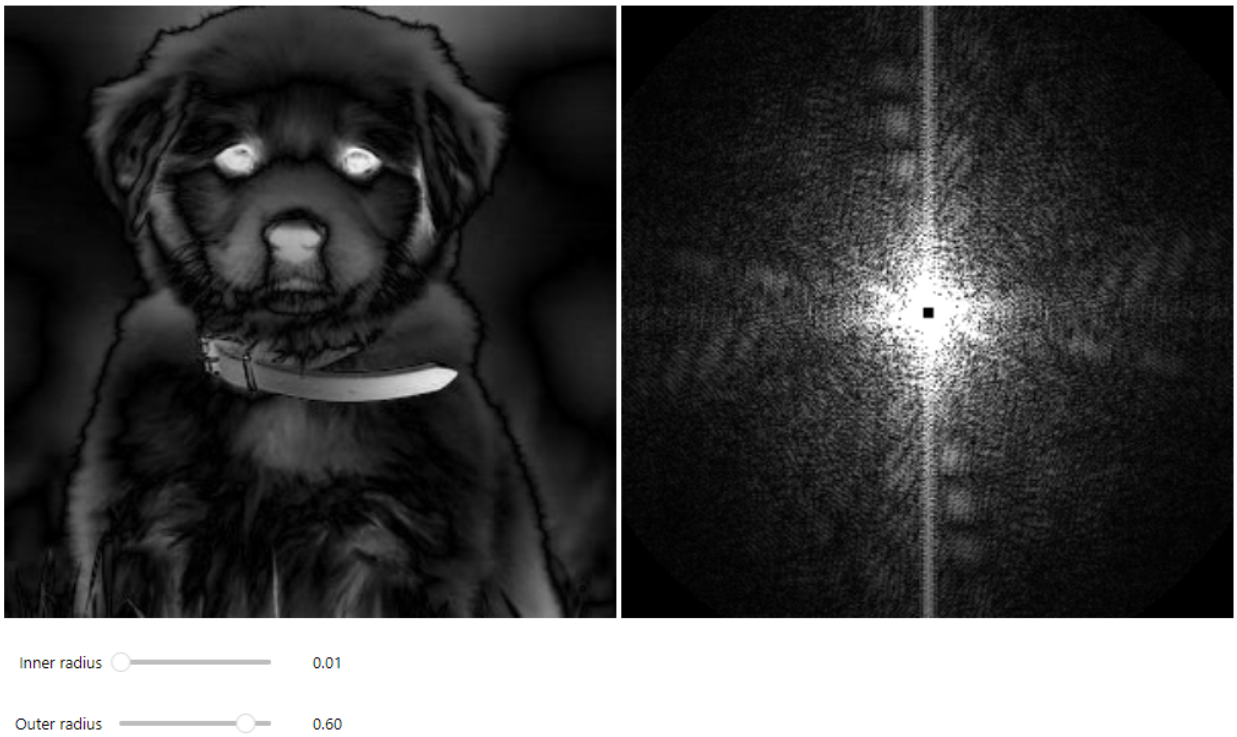

Figure 5: Frequency removal result

2.4 Creating a Hybird Image

```
def create_hybrid_img(img1, img2, r):
  """
  Create hydrid image
  Params:
    img1: numpy image 1
    img2: numpy image 2
    r: radius that defines the filled circle of frequency of image 1
  """
  # You need to implement the function
  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)

    height, width = img1.shape
    mask = np.zeros(img1.shape, dtype=np.float32)
    center = (height // 2, width // 2)
    for i in range(height):
        for j in range(width):
            if (i - center[0])**2 + (j - center[1])**2 <= r**2:
              mask[i, j] = 1

    hybrid_fft_shifted = img1_fft_shifted * (mask) + img2_fft_shifted * (mask - 1)

    hybrid_fft = np.fft.ifftshift(hybrid_fft_shifted)

    hybrid_img = np.fft.ifft2(hybrid_fft)

    resulst = np.real(hybrid_img)

    return resulst
```

Result:

Figure 6: Hybird image result