# INT3404E 20 - Image Processing: Homeworks 2

Nguyen Tien Dat 20021327

## 1 Introduction

All the implementations are done in Python Notebook files namely: ex1.ipynb, ex212.ipynb, ex234.ipynb

## 2 Image Filtering

This exercise is about image filtering using kernel on padded images. For replicate padding method, i implemented from scratch method.

```
def padding_img(img, filter_size=3):
    """
    The surrogate function for the filter functions.
    The goal of the function: padding the image such that when applying the kernel with the size of filter_size,
    WARNING: Do not use the exterior functions from available libraries such as OpenCV, scikit-image, etc. Just d
    Inputs:
        img: cv2 image: original image
        filter_size: int: size of square filter
        padding_mode: str: 'zero'| 'mirror' | 'replicate'
    Return:
        padded_img: cv2 image: the padding image
    """
    img_height, img_width = img.shape
    pad_size = filter_size // 2
    padded_img = np.zeros((img_height + 2 * pad_size, img_width + 2 * pad_size), dtype=img.dtype)
    padded_img[pad_size:pad_size+img_height, pad_size:pad_size+img_width] = img
    return padded_img
```

For mean filtering, a convolution sum of the padded image and the kernel is applied using 2 loops to get the convolved sum wise,then return the restore image from padding. The result give us a slight little change of noise removal in image.

```
def mean_filter(img, filter_size=3):
    #padding image
    img = padding_img(img, filter_size)
    # print(img.shape)
    img_height, img_width = img.shape
    kernel_wind = np.full((filter_size, filter_size), 1 / filter_size ** 2)
    # Create output image
    output_img = np.zeros_like(img)
    # Perform convolution
    for i in range(filter_size-1, img_height - filter_size+1):
        for j in range(filter_size-1, img_width - filter_size+1):
            output_img[i, j] = (kernel_wind * img[i - int(filter_size/2) :
            i + 1+int(filter_size/2), j - int(filter_size/2) : j + 1+int(filter_size/2)]).sum()
    return output_img[filter_size-1:img_height - (filter_size-1), (filter_size-1):img_width - (filter_size-1)]
```

For median filter, the same 2 loops are perform. This time we get the median of the tracked image crop. This give a better noise removal with PSNR score of median filter: 37.119578300855245 .

```
def median_filter(img, filter_size=3):
  img = padding_img(img, filter_size)
  img_height, img_width = img.shape
  # Create output image
  output_img = np.zeros_like(img)
  # Perform convolution
```

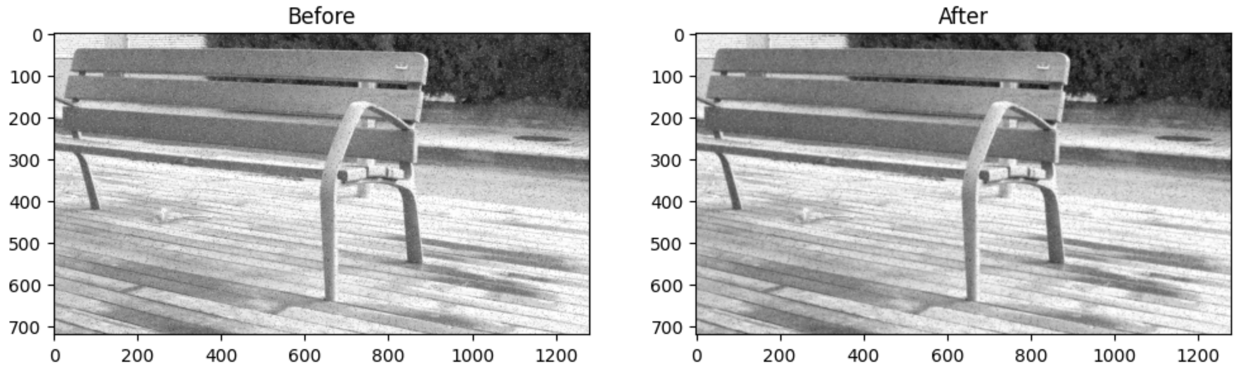


Figure 1: Mean filtering. The PSNR score of mean filter: 31.605849430056452

```
    for i in range(filter_size-1, img_height - filter_size+1):
        for j in range(filter_size-1, img_width - filter_size+1):
            output_img[i, j] = np.median(img[i - int(filter_size/2) : i + 1+int(filter_size/2), j - int(filter_size
    return output_img[filter_size-1:img_height - (filter_size-1), (filter_size-1):img_width - (filter_size-1)]
    # Need to implement here
```

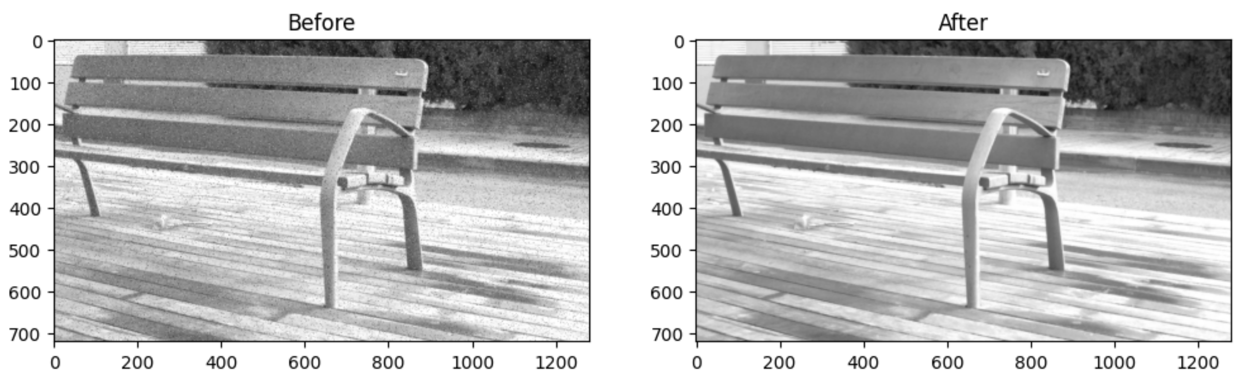


Figure 2: Median Filtering

The implemented criteria is PSNR score.

```
def psnr(gt_img, smooth_img):
    mse_score = mse(gt_img, smooth_img)
    max_pixel = 255.0
    psnr_score = 10 * math.log10(max_pixel*max_pixel / mse_score)
    return psnr_score
def mse(gt_img, smooth_img):
    mse_score = np.mean((gt_img - smooth_img) ** 2)
    return mse_score
```

# 3 Fourier Transform

## 3.1 1D Fourier Transform

This is implementation with one loop after the FT formula.

```
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.complex_)
    for k in range(N):
        for n in range(N):
            DFT[k] += data[n] * np.exp(-1j * 2 * np.pi * n * k / N)
    return DFT
```

Result: True

### 3.2 2D FT

Performed FT on 2 axes horizontal and vertical.

```
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
    row_fft = np.zeros((H, W), dtype=np.complex_)
    row_col_fft = np.zeros((H, W), dtype=np.complex_)

    #Horizontal FFTi____
    for i in range(H):
        row_fft[i, :] = np.fft.fft(gray_img[i, :])

    #Vertical FFT j_____
    for j in range(W):
        row_col_fft[:, j] = np.fft.fft(row_fft[:, j])

    return row_fft, row_col_fft
```

Result:

### 3.3 Frequency Removal Procedure

This exercise is performed after the instruction given.

```
def filter_frequency(orig_img, mask):
  """
  Remove frequency based on the given mask.
  Params:
    orig_img: numpy image
    mask: same shape with orig_img indicating which frequency to 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_shifted = np.fft.fftshift(f_img)
  f_img_filtered = f_img_shifted * mask
```

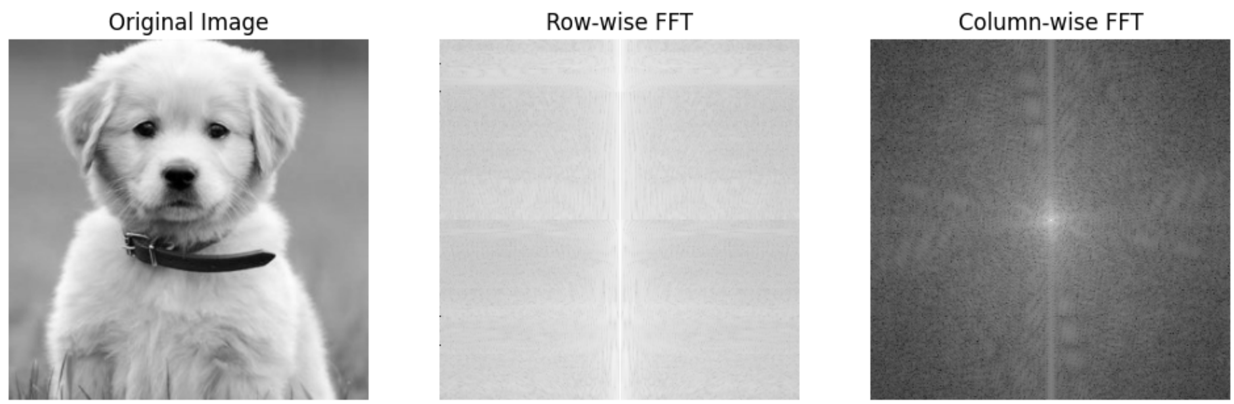


Figure 3: 2D FT

```
  #shift back
  f_img_filtered_shifted = np.fft.ifftshift(f_img_filtered)
  #invert
  img = np.fft.ifft2(f_img_filtered_shifted)
  return np.abs(f_img_filtered), np.abs(img)
```

Return: Filtered image is print out in the application along with the frequency domain.

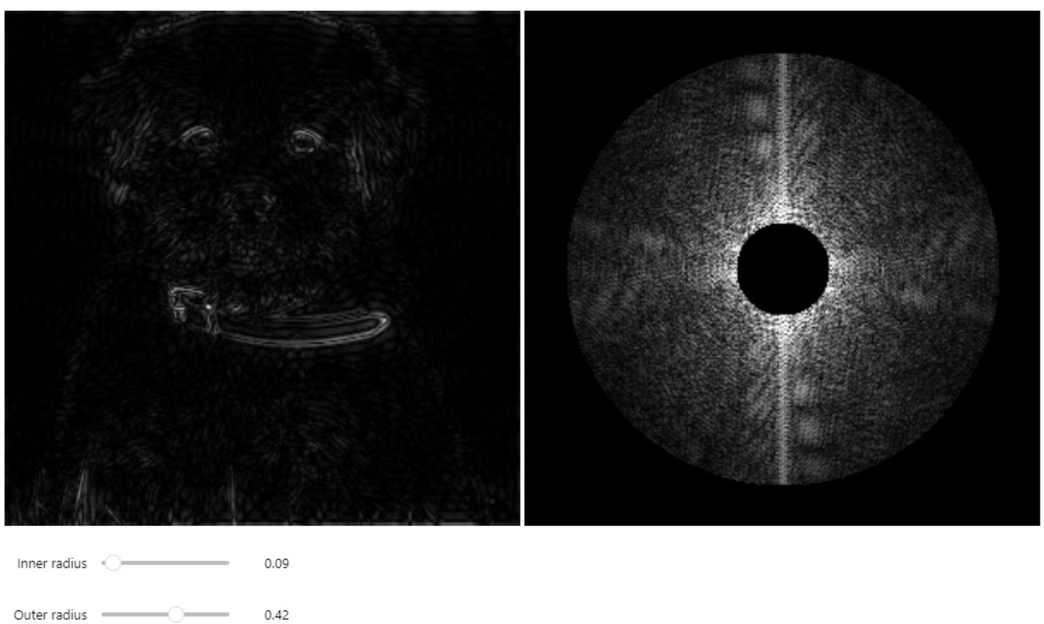


Figure 4: Enter Caption

## 3.4 Creating a Hybrid Image

This exercise is performed after the instruction given.

```
def create_hybrid_img(img1, img2, r):
  """
  Create hybrid image
  Params:
    img1: numpy image 1
```

```python
    img2: numpy image 2
    r: radius that defines the filled circle of frequency of image 1. Refer to the homework title to know more.
  """
  # Apply Fourier Transform to both images
  f_img1 = np.fft.fftshift(np.fft.fft2(img1))
  f_img2 = np.fft.fftshift(np.fft.fft2(img2))

  # Create a mask based on the radius
  mask = np.zeros_like(img1)
  center_x, center_y = img1.shape[0] // 2, img1.shape[1] // 2
  for i in range(img1.shape[0]):
    for j in range(img1.shape[1]):
      if np.sqrt((i - center_x) ** 2 + (j - center_y) ** 2) < r:
        mask[i, j] = 1

  # Apply the mask to the frequency domain of image 1
  f_img1_filtered = f_img1 * mask

  # Combine the filtered frequency domain of image 1 and the frequency domain of image 2
  f_hybrid = f_img1_filtered + (1 - mask) * f_img2

  # Apply inverse Fourier Transform to obtain the hybrid image
  hybrid_img = np.abs(np.fft.ifft2(np.fft.ifftshift(f_hybrid)))

  return hybrid_img
```

Result: 2 images is merged

Figure 5: Enter Caption