INT3404E 20 - Image Processing: Homeworks 2

21021491 - Ngo Thuong Hieu

April 19, 2024

1 Image Filtering

1.1 Padding Function

```python
def padding_img(img, filter_size=3):
"""
The surrogate function for the filter functions.
The goal of the function: replicate padding the image such that when
   applying the kernel with the size of filter_size, the padded image
    will be the same size as the original image.
WARNING: Do not use the exterior functions from available libraries
   such as OpenCV, scikit-image, etc. Just do from scratch using
   function from the numpy library or functions in pure Python.
Inputs:
    img: cv2 image: original image
    filter_size: int: size of square filter
Return:
    padded_img: cv2 image: the padding image
"""
def pad(row, padding):
    target = np.concatenate(([row[0]]*padding, row, [row[-1]]*padding
       ), axis=None)
    return target

padding = filter_size // 2
padding_img = np.zeros((padding * 2 + img.shape[0], padding * 2 + img
   .shape[1]))

for row in range(img.shape[0]):
    if row == 0:
        padding_img[:padding] = pad(img[row], padding) * padding
    elif row == img.shape[0] - 1:
        padding_img[-padding:] = pad(img[row], padding) * padding
    else:
        padding_img[row] = pad(img[row], padding)
return padding_img
```

Result: Figure 1 shows the result of padding function after applying it to original image. The black bold border covering around the original image is the padding of it.

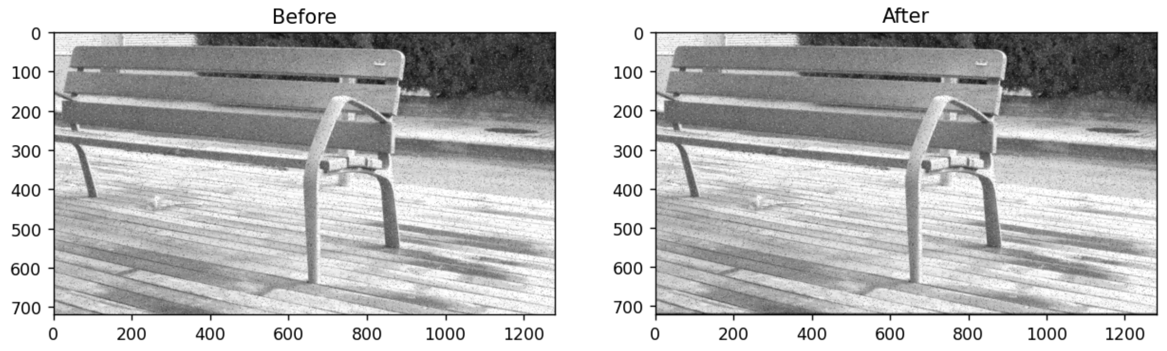


Figure 1: Result of padding function.

1.2 Mean Filtering

```
    def mean_filter(img, filter_size=3):
    """
    Smoothing image with mean square filter with the size of filter_size.
        Use replicate padding for the image.
    WARNING: Do not use the exterior functions from available libraries
       such as OpenCV, scikit-image, etc. Just do from scratch using
       function from the numpy library or functions in pure Python.
    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
    smoothed_img = np.copy(img)

    for i in range(0, img.shape[0] - filter_size + 1):
        for j in range(0, img.shape[1] - filter_size + 1):
            smoothed_img[i:i+filter_size, j:j+filter_size] = np.mean(img[
                i:i+filter_size, j:j+filter_size])

    return smoothed_img
```

Result: Figure 2 illustrates the filtered image after going through a mean filter layer. The result image does not seem to have significant changes compared with the original one.

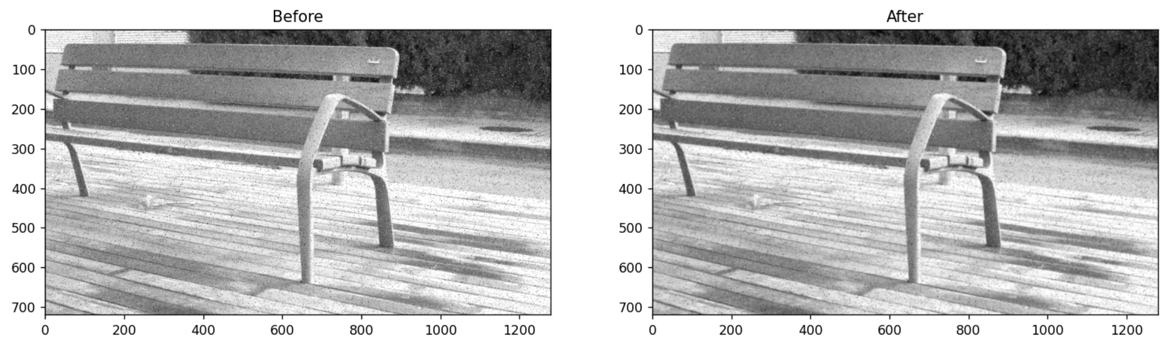


Figure 2: Result of mean filtering function.

1.3 Median Filtering

```
    def median_filter(img, filter_size=3):
    """
        Smoothing image with median square filter with the size of
           filter_size. Use replicate padding for the image.
        WARNING: Do not use the exterior functions from available
           libraries such as OpenCV, scikit-image, etc. Just do from
           scratch using function from the numpy library or functions in
           pure Python.
        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
    smoothed_img = np.copy(img)
    for i in range(0, img.shape[0] - filter_size + 1):
        for j in range(0, img.shape[1] - filter_size + 1):
            smoothed_img[i:i+filter_size, j:j+filter_size] = np.median(
               img[i:i+filter_size, j:j+filter_size])

    return smoothed_img
```

Result: Figure 3 shows the result of median filtering function that applied to original image. The result is quite good compared with the original one, the noises are removed and the target image seems to be clearer and smoother.

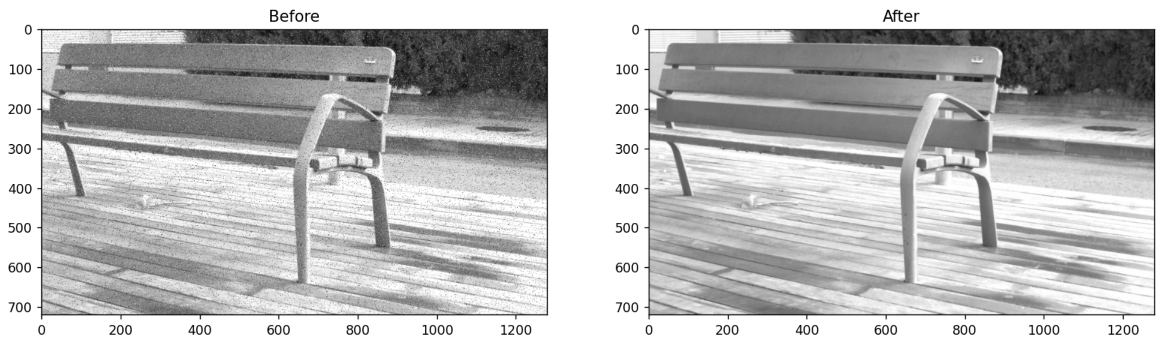


Figure 3: Result of median filtering function.

1.4 PSNR Score

```
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
"""
# Need to implement here
MAX = 255
MSE = np.sum(np.square(gt_img - smooth_img)) / (gt_img.shape[0] *
    gt_img.shape[1])

return 10 * np.log(MAX ** 2 / MSE)
```

Definition: The PSNR block computes the peak signal-to-noise ratio, in decibels, between two images. This ratio is used as a quality measurement between the original and a compressed image. The higher the PSNR, the better the quality of the compressed, or reconstructed image.

Result on mean filtering: PSNR score of mean filter: 70.14201295428222.

Result on median filtering: PSNR score of median filter: 74.60816146296379.

So according to PSNR scores of the two filters, the median filter provides better score and we can choose that filter.

2 Fourier Transform

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
    """
    # You need to implement the DFT here
    N = data.shape[0]
    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
```

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
    """
    # You need to implement the DFT here
    row_fft = np.array([DFT_slow(row) for row in gray_img])
    row_col_fft = np.array([DFT_slow(row) for row in row_fft.T]).T

    return row_fft, row_col_fft
```

Result: Figure 4 shows the 2D Fourier Transformation that has been applied to the row-based and col-based of the original image.

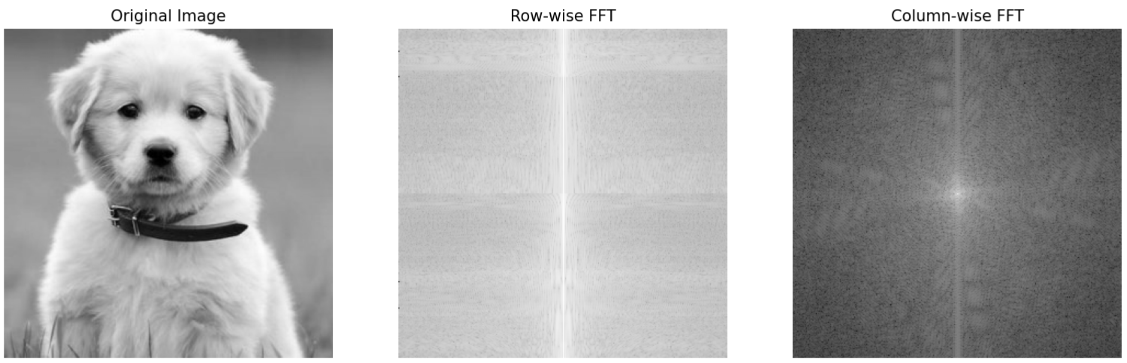


Figure 4: Result of 2D Fourier Transform.

2.3 Frequency Removal Procedure

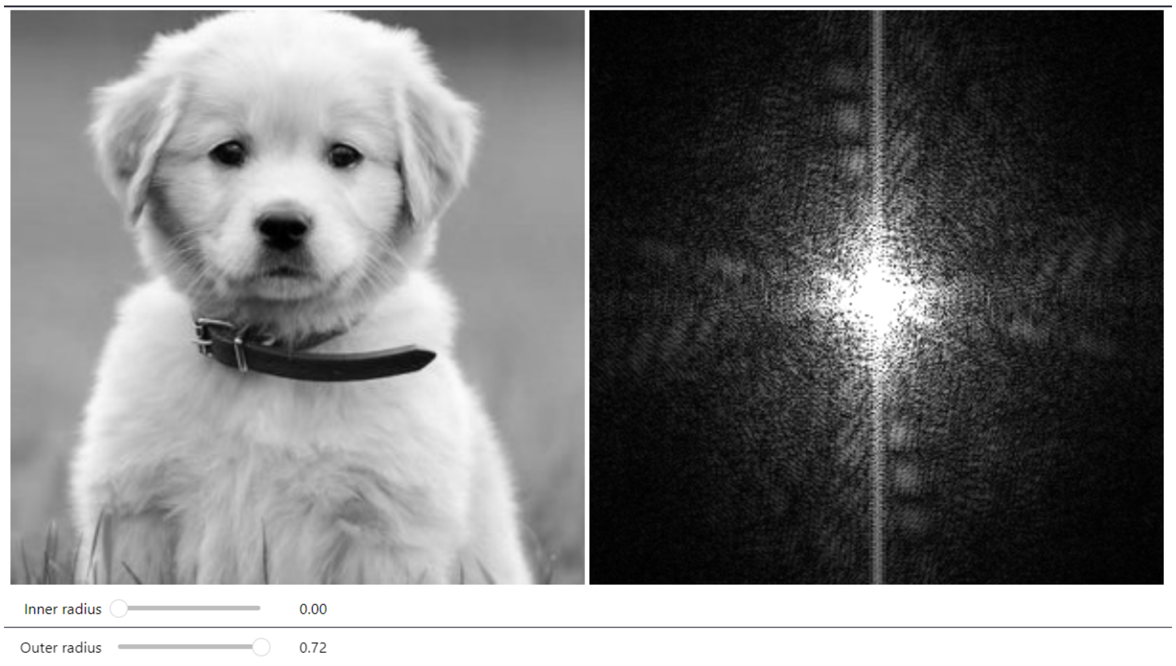
```python
def filter_frequency(orig_img, mask):
  """
  You need to remove frequency based on the given mask.
  Params:
    orig_img: numpy image
    mask: same shape with orig_img indicating which frequency hold or
        remove
  Output:
    f_img: frequency image after applying mask
    img: image after applying mask
  """
  # You need to implement this function
  fft2 = np.fft.fft2(orig_img)
  coefs = np.fft.fftshift(fft2)
  masked_coefs = coefs * mask
  coefs_inversed = np.fft.ifftshift(masked_coefs)
  img_inversed = np.fft.ifft2(coefs_inversed)

  return np.abs(masked_coefs), np.abs(img_inversed)
```

Result: Figure 5a and Figure 5b describe the effects of Frequency removal applied to the image.

2.4 Creating a Hybrid Image

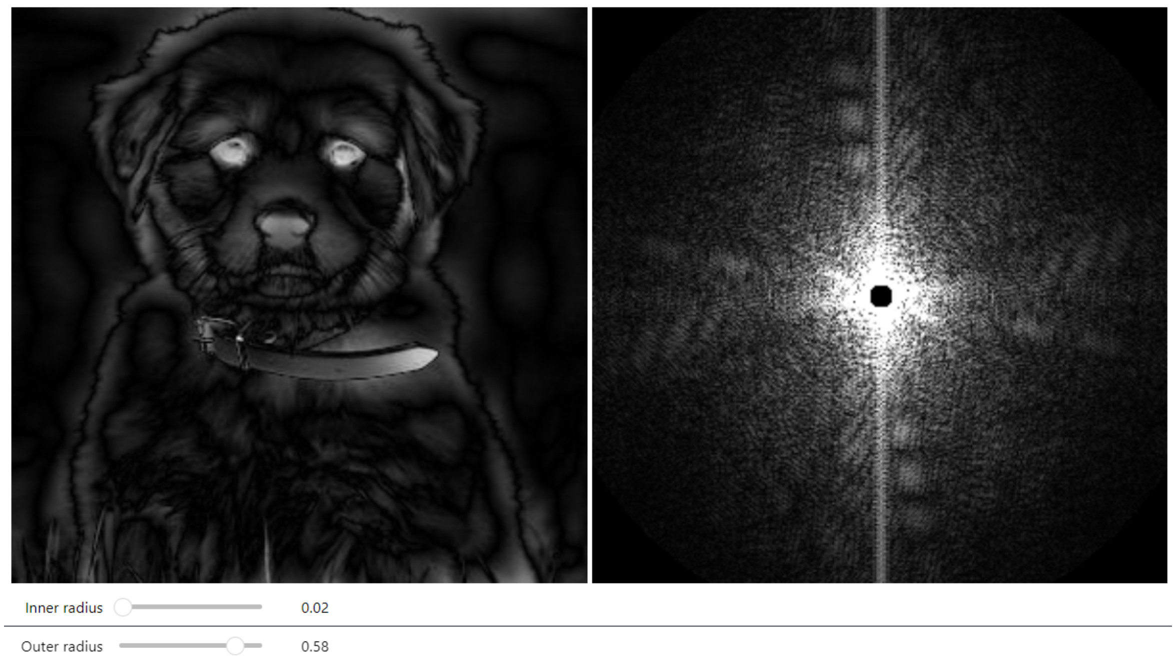
```
def create_hybrid_img(img1, img2, r):
  """
  Create hydrid image
  Params:
    img1: numpy image 1
    img2: numpy image 2
    r: radius that defines the filled circle of frequency of image 1.
        Refer to the homework title to know more.
  """
  # You need to implement the function
  def create_circle_matrix(shape, radius):
    rows, cols = shape
    row_center, col_center = rows//2, cols//2
    Y, X = np.ogrid[:rows, :cols]
    distance_squared = (X - col_center)**2 + (Y - row_center)**2
    circle_mask = distance_squared <= radius**2

    return circle_mask

  img1_fft = np.fft.fft2(img1)
  img2_fft = np.fft.fft2(img2)
  coefs1 = np.fft.fftshift(img1_fft)
  coefs2 = np.fft.fftshift(img2_fft)
  mask = create_circle_matrix(img1.shape, r)
  masked_coefs = coefs1 * mask + coefs2 * (1 - mask)
  coefs_inversed = np.fft.ifftshift(masked_coefs)
  img_inversed = np.fft.ifft2(coefs_inversed)

  return np.abs(img_inversed)
```

Result: Figure 6 shows the hybrid image that is the combination of two original images.


Inner radius 0.00

Outer radius 0.72

(a) Frequency removed 1.


Inner radius 0.02

Outer radius 0.58

(b) Frequency removed 2.

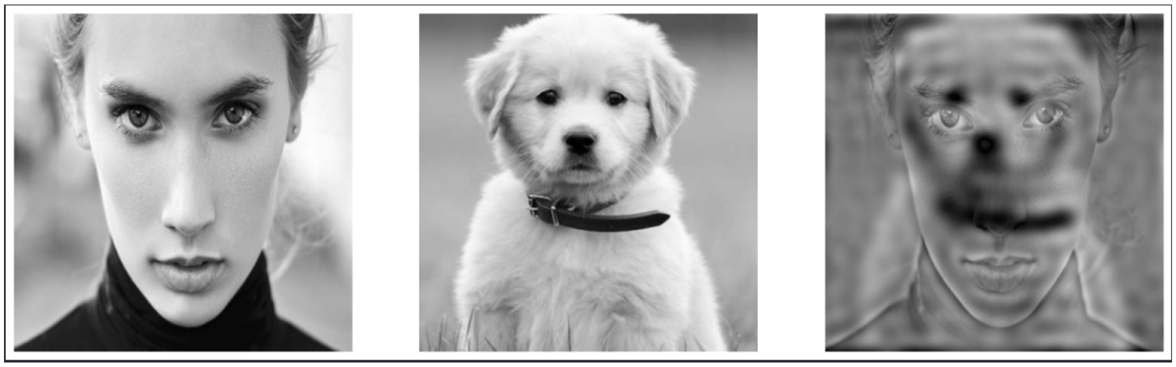


Figure 6: Result of 2D Fourier Transform.