INT3404E 20 - Image Processing: Homeworks 2

Nguyen Ngoc Tuan - 21020237

1 Ex1.py - Image Filtering

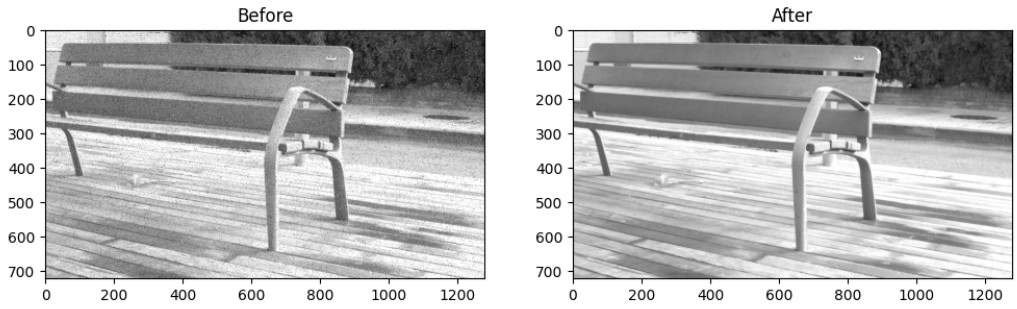

Figure 1: The Mean and Median filters for provided images

```python
import cv2
import matplotlib.pyplot as plt
import numpy as np
import math
import os


def read_img(img_path):
    """
        Read grayscale image
        Inputs:
        img_path: str: image path
```

```
        Returns:
        img: cv2 image
    """
    return cv2.imread(img_path, 0)


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 kerne
    WARNING: Do not use the exterior functions from available libraries such as OpenCV, sci
    Inputs:
        img: cv2 image: original image
        filter_size: int: size of square filter
    Return:
        padded_img: cv2 image: the padding image
    """
  # Need to implement here
    height, width = img.shape[:2]
    pad_width = filter_size // 2
    padded_img = np.pad(img, pad_width, mode='edge')
    return padded_img

def mean_filter(img, filter_size=3):
    """
    Smoothing image with mean square filter with the size of filter_size. Use replicate pad
    WARNING: Do not use the exterior functions from available libraries such as OpenCV, sci
    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
    padded_img = padding_img(img, filter_size)
    smoothed_img = np.zeros_like(img, dtype=np.float32)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            neighborhood = padded_img[i:i+filter_size, j:j+filter_size]
            mean_value = np.mean(neighborhood)
            smoothed_img[i, j] = mean_value

    return smoothed_img.astype(np.uint8)


def median_filter(img, filter_size=3):
    """
    Smoothing image with median square filter with the size of filter_size. Use replicate p
    WARNING: Do not use the exterior functions from available libraries such as OpenCV, sci
```

```
    Inputs:
        img: numpy array: original image
        filter_size: int: size of square filter
    Return:
        smoothed_img: numpy array: the smoothed image with median filter.
    """
    # Need to implement here
    padded_img = padding_img(img, filter_size)
    smoothed_img = np.zeros_like(img, dtype=np.uint8)

    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            neighborhood = padded_img[i:i+filter_size, j:j+filter_size]
            median_value = np.median(neighborhood)
            smoothed_img[i, j] = median_value

    return smoothed_img


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
    assert gt_img.shape == smooth_img.shape, "Input images must have the same dimensions"
    assert gt_img.dtype == smooth_img.dtype, "Input images must have the same data type"
    mse = np.mean((gt_img - smooth_img) ** 2)
    max_pixel_value = np.iinfo(gt_img.dtype).max
    psnr_score = 20 * np.log10(max_pixel_value) - 10 * np.log10(mse)

    return psnr_score


def show_res(before_img, after_img):
    """
        Show the original image and the corresponding smooth image
        Inputs:
            before_img: cv2: image before smoothing
            after_img: cv2: corresponding smoothed image
        Return:
            None
    """
    plt.figure(figsize=(12, 9))
    plt.subplot(1, 2, 1)
```

```
    plt.imshow(before_img, cmap='gray')
    plt.title('Before')

    plt.subplot(1, 2, 2)
    plt.imshow(after_img, cmap='gray')
    plt.title('After')
    plt.show()


if __name__ == '__main__':
    img_noise = "./ex1_images/noise.png" # <- need to specify the path to the noise image
    img_gt = "./ex1_images/ori_img.png"  # <- need to specify the path to the gt image
    img = read_img(img_noise)
    filter_size = 3

    # Mean filter
    mean_smoothed_img = mean_filter(img, filter_size)
    show_res(img, mean_smoothed_img)
    print('PSNR score of mean filter: ', psnr(img, mean_smoothed_img))

    # Median filter
    median_smoothed_img = median_filter(img, filter_size)
    show_res(img, median_smoothed_img)
    print('PSNR score of median filter: ', psnr(img, median_smoothed_img))
```

2 Ex212.py - 2D Fourier Transform

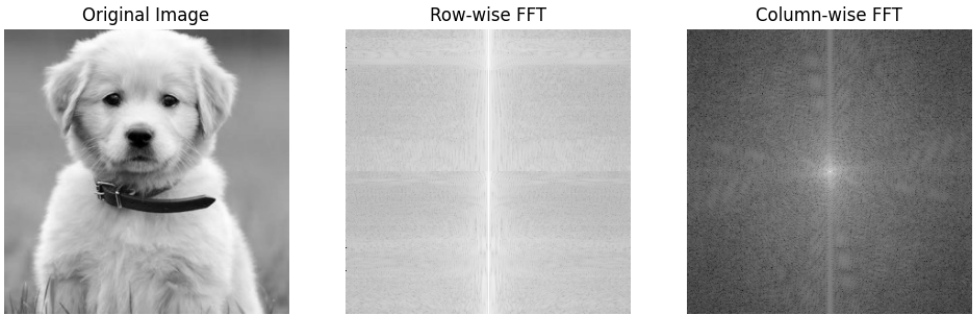

Figure 2: Output for 2D Fourier Transform Exercise

```python
import numpy as np
from skimage import io as io_url
import matplotlib.pyplot as plt


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)
    n = np.arange(N)
    k = n.reshape((N, 1))
    exp_term = np.exp(-2j * np.pi * k * n / N)
    return np.dot(exp_term, data)


```

```python
def show_img(origin, row_fft, row_col_fft):
    """
    Show the original image, row-wise FFT, and column-wise FFT

    params:
        origin: (H, W): 2D numpy array
        row_fft: (H, W): 2D numpy array
        row_col_fft: (H, W): 2D numpy array
    """
    fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(12, 8))
    axs[0].imshow(origin, cmap='gray')
    axs[0].set_title('Original Image')
    axs[0].axis('off')
    axs[1].imshow(np.log(np.abs(np.fft.fftshift(row_fft))), cmap='gray')
    axs[1].set_title('Row-wise FFT')
    axs[1].axis('off')
    axs[2].imshow((np.log(np.abs(np.fft.fftshift(row_col_fft)))), cmap='gray')
    axs[2].set_title('Column-wise FFT')
    axs[2].axis('off')
    plt.show()


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
    """
    H, W = 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(H):
        row_fft[i, :] = DFT_slow(gray_img[i, :])

    for j in range(W):
        row_col_fft[:, j] = DFT_slow(row_fft[:, j])

    return row_fft, row_col_fft


if __name__ == '__main__':
    # Load the image
    img = io_url.imread('https://img2.zergnet.com/2309662_300.jpg')
```

```
    gray_img = np.mean(img, -1)  # Convert to grayscale

    # Compute the 2D DFT
    row_fft, row_col_fft = DFT_2D(gray_img)

    # Display the results
    show_img(gray_img, row_fft, row_col_fft)
import numpy as np
from skimage import io as io_url
import matplotlib.pyplot as plt


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)
    n = np.arange(N)
    k = n.reshape((N, 1))
    exp_term = np.exp(-2j * np.pi * k * n / N)
    return np.dot(exp_term, data)


def show_img(origin, row_fft, row_col_fft):
    """
    Show the original image, row-wise FFT, and column-wise FFT

    params:
        origin: (H, W): 2D numpy array
        row_fft: (H, W): 2D numpy array
        row_col_fft: (H, W): 2D numpy array
    """
    fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(12, 8))
    axs[0].imshow(origin, cmap='gray')
    axs[0].set_title('Original Image')
    axs[0].axis('off')
    axs[1].imshow(np.log(np.abs(np.fft.fftshift(row_fft))), cmap='gray')
    axs[1].set_title('Row-wise FFT')
    axs[1].axis('off')
    axs[2].imshow((np.log(np.abs(np.fft.fftshift(row_col_fft)))), cmap='gray')
    axs[2].set_title('Column-wise FFT')
    axs[2].axis('off')
    plt.show()
```

```python
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
    """
    H, W = 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(H):
        row_fft[i, :] = DFT_slow(gray_img[i, :])

    for j in range(W):
        row_col_fft[:, j] = DFT_slow(row_fft[:, j])

    return row_fft, row_col_fft


if __name__ == '__main__':
    # Load the image
    img = io_url.imread('https://img2.zergnet.com/2309662_300.jpg')
    gray_img = np.mean(img, -1)  # Convert to grayscale

    # Compute the 2D DFT
    row_fft, row_col_fft = DFT_2D(gray_img)

    # Display the results
    show_img(gray_img, row_fft, row_col_fft)
```

3 Ex234.ipynb - Frequency Removal Procedure and Hybrid Image

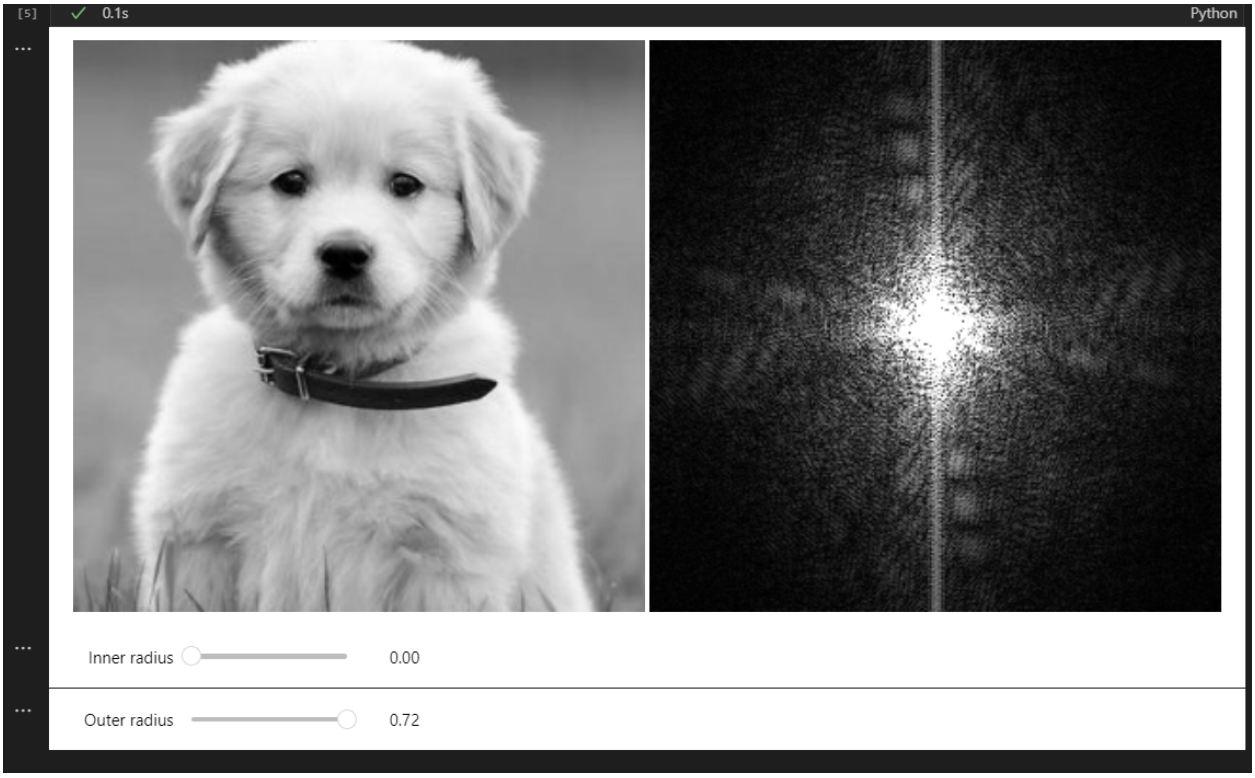

Figure 3: Expected Output for 2D Frequency Removal Exercise (Inner Rad = 0.00)

```
import ipywidgets as widgets
import matplotlib.pyplot as plt
import PIL.Image
import numpy as np
import urllib
from skimage.transform import resize
from matplotlib.image import imread
import os
from IPython.display import display
from skimage import io as io_url
import cv2
import numpy as np
from PIL import Image

# Create image widgets
image3_spatial = widgets.Image(format='png', width=500, height=500, description='Spatial')
image3_freq    = widgets.Image(format='png', width=500, height=500, description='Frequency
sidebyside     = widgets.HBox([image3_spatial, image3_freq])

# Create slider/select widgets
slider_inner   = widgets.FloatSlider(value=0,      min=0, max=1,      step=0.01, descriptio
slider_outer   = widgets.FloatSlider(value=1.44/2, min=0, max=1.44/2, step=0.01, descriptio
```

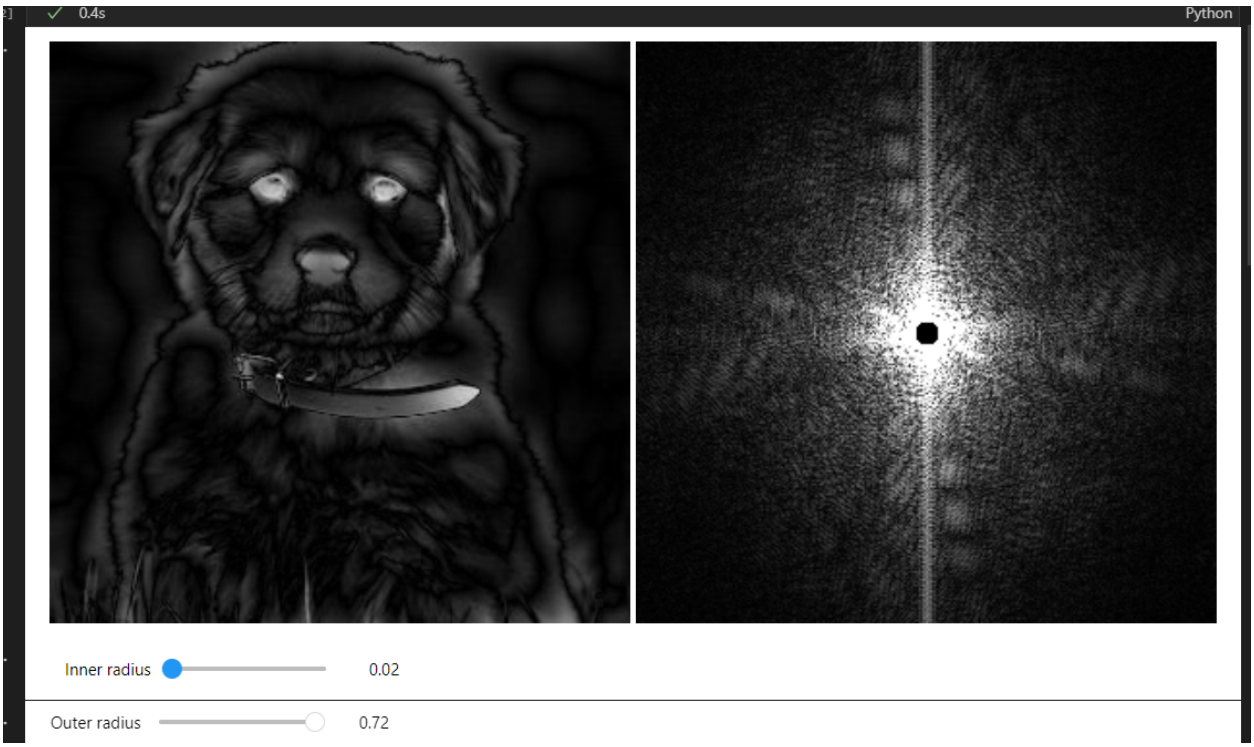

Figure 4: Expected Output for 2D Frequency Removal Exercise (Inner Rad = 0.02

```
23
24  buf             = io.BytesIO()
25
26  orig_img = io_url.imread('https://img2.zergnet.com/2309662_300.jpg')
27  orig_img = np.mean(orig_img, -1)
28
29  x = np.fft.fftfreq(orig_img.shape[0]);
30  y = np.fft.fftfreq(orig_img.shape[1]);
31
32  xv, yv = np.meshgrid(x, y)
33  xv = np.fft.fftshift(xv)
34  yv = np.fft.fftshift(yv)
35
36  def filter_frequency(orig_img, mask):
37      """
38      You need to remove frequency based on the given mask.
39      Params:
40        orig_img: numpy image
41        mask: same shape with orig_img indicating which frequency hold or remove
42      Output:
43        f_img: frequency image after applying mask
44        img: image after applying mask
45      """
46      # You need to implement this function
47      f_img = np.fft.fft2(orig_img)
```

```
48
49      f_img_shifted = np.fft.fftshift(f_img)
50
51      f_img_filtered_shifted = f_img_shifted * mask
52
53      f_img_filtered = np.fft.ifftshift(f_img_filtered_shifted)
54
55      img = np.abs(np.fft.ifft2(f_img_filtered))
56
57      return np.abs(f_img_filtered_shifted), img
58
59  def on_value_change3(change):
60      mask = (np.sqrt(xv**2 + yv**2) < slider_outer.value) & \
61              (np.sqrt(xv**2 + yv**2) >= slider_inner.value)
62      mask = np.float32(mask)
63
64      fimg, img = filter_frequency(orig_img, mask)
65      buf.seek(0)
66      tmp = PIL.Image.fromarray(255*img/(img.max()+0.0001))
67      tmp = tmp.convert('L')
68      tmp.save(buf, 'png')
69      image3_spatial.value = buf.getvalue()
70
71      buf.seek(0)
72      tmp = PIL.Image.fromarray(255*np.log(0.0001*fimg + 1))
73      tmp = tmp.convert('L')
74      tmp.save(buf, 'png')
75      image3_freq.value = buf.getvalue()
76
77
78  slider_inner.observe(on_value_change3, names='value')
79  slider_outer.observe(on_value_change3, names='value')
80
81  on_value_change3(0)
82
83  display(sidebyside)
84  display(slider_inner)
85  display(slider_outer)
```

```
1   def read_img(img_path, img_size=(512, 512)):
2     """
3       +    c      nh
4       + C h u y n   t h  nh grayscale
5       + Thay    i     k ch   t h    c    nh    th nh img_size
6     """
7     img = cv2.imread(img_path, 0)
8     img = cv2.resize(img, img_size)
9     return img
10
```

Figure 5: Hybrid Image

```
11
12  def create_hybrid_img(img1, img2, r):
13      """
14      Create hydrid image
15      Params:
16        img1: numpy image 1
17        img2: numpy image 2
18        r: radius that defines the filled circle of frequency of image 1. Refer to the homeworl
19      """
20      f_img1 = np.fft.fftshift(np.fft.fft2(img1))
21      f_img2 = np.fft.fftshift(np.fft.fft2(img2))
22
23      rows, cols = img1.shape[:2]
24      crow, ccol = rows // 2, cols // 2
25      mask = np.zeros((rows, cols), dtype=np.uint8)
26      mask[crow-r:crow+r, ccol-r:ccol+r] = 1
27
28      f_img1_filtered = f_img1 * mask
29      f_img2_filtered = f_img2 * (1 - mask)
30
31      f_hybrid = f_img1_filtered + f_img2_filtered
32
33      hybrid_img = np.abs(np.fft.ifft2(np.fft.ifftshift(f_hybrid)))
34
35      hybrid_img = np.clip(hybrid_img, 0, 255).astype(np.uint8)
36
37      return hybrid_img
38
39      image_1_path = "./ex1_images/girl.png" # <-- need to change
40      image_2_path = "./ex1_images/dog.png" # <-- need to change
41      img_1 = read_img(image_1_path)
42      img_2 = read_img(image_2_path)
43      hybrid_img = create_hybrid_img(img_2, img_1, 14)
44      fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(18, 15))
45      axes[0].imshow(img_1, cmap="gray")
```

```
46  axes[0].axis("off")
47  axes[1].imshow(img_2, cmap="gray")
48  axes[1].axis("off")
49  axes[2].imshow(hybrid_img, cmap="gray")
50  axes[2].axis("off")
```