

# INT3404E 20 - Image Processing: Homeworks 2

Nguyen Ngoc Tuan - 21020237

## 1 Ex1.py - Image Filtering

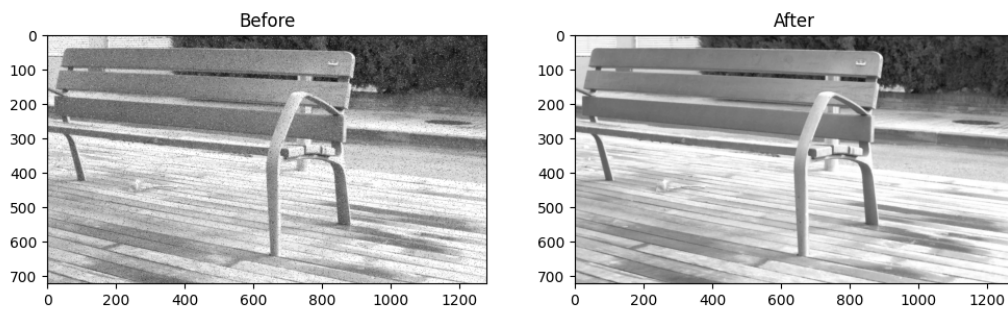


Figure 1: The Mean and Median filters for provided images

```
1 import cv2
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import math
5 import os
6
7
8 def read_img(img_path):
9     """
10         Read grayscale image
11         Inputs:
12             img_path: str: image path
```

```

13     Returns:
14         img: cv2 image
15     """
16     return cv2.imread(img_path, 0)
17
18
19 def padding_img(img, filter_size=3):
20     """
21     The surrogate function for the filter functions.
22     The goal of the function: replicate padding the image such that when applying the kernel
23     WARNING: Do not use the exterior functions from available libraries such as OpenCV, scipy
24     Inputs:
25         img: cv2 image: original image
26         filter_size: int: size of square filter
27     Return:
28         padded_img: cv2 image: the padding image
29     """
30     # Need to implement here
31     height, width = img.shape[:2]
32     pad_width = filter_size // 2
33     padded_img = np.pad(img, pad_width, mode='edge')
34     return padded_img
35
36 def mean_filter(img, filter_size=3):
37     """
38     Smoothing image with mean square filter with the size of filter_size. Use replicate padding
39     WARNING: Do not use the exterior functions from available libraries such as OpenCV, scipy
40     Inputs:
41         img: cv2 image: original image
42         filter_size: int: size of square filter,
43     Return:
44         smoothed_img: cv2 image: the smoothed image with mean filter.
45     """
46     # Need to implement here
47     padded_img = padding_img(img, filter_size)
48     smoothed_img = np.zeros_like(img, dtype=np.float32)
49     for i in range(img.shape[0]):
50         for j in range(img.shape[1]):
51             neighborhood = padded_img[i:i+filter_size, j:j+filter_size]
52             mean_value = np.mean(neighborhood)
53             smoothed_img[i, j] = mean_value
54
55     return smoothed_img.astype(np.uint8)
56
57
58 def median_filter(img, filter_size=3):
59     """
60     Smoothing image with median square filter with the size of filter_size. Use replicate padding
61     WARNING: Do not use the exterior functions from available libraries such as OpenCV, scipy

```

```
62     Inputs:
63         img: numpy array: original image
64         filter_size: int: size of square filter
65     Return:
66         smoothed_img: numpy array: the smoothed image with median filter.
67     """
68     # Need to implement here
69     padded_img = padding_img(img, filter_size)
70     smoothed_img = np.zeros_like(img, dtype=np.uint8)
71
72     for i in range(img.shape[0]):
73         for j in range(img.shape[1]):
74             neighborhood = padded_img[i:i+filter_size, j:j+filter_size]
75             median_value = np.median(neighborhood)
76             smoothed_img[i, j] = median_value
77
78     return smoothed_img
79
80
81 def psnr(gt_img, smooth_img):
82     """
83     Calculate the PSNR metric
84     Inputs:
85         gt_img: cv2 image: groundtruth image
86         smooth_img: cv2 image: smoothed image
87     Outputs:
88         psnr_score: PSNR score
89     """
90     # Need to implement here
91     assert gt_img.shape == smooth_img.shape, "Input images must have the same dimensions"
92     assert gt_img.dtype == smooth_img.dtype, "Input images must have the same data type"
93     mse = np.mean((gt_img - smooth_img) ** 2)
94     max_pixel_value = np.iinfo(gt_img.dtype).max
95     psnr_score = 20 * np.log10(max_pixel_value) - 10 * np.log10(mse)
96
97     return psnr_score
98
99
100 def show_res(before_img, after_img):
101     """
102     Show the original image and the corresponding smooth image
103     Inputs:
104         before_img: cv2: image before smoothing
105         after_img: cv2: corresponding smoothed image
106     Return:
107         None
108     """
109     plt.figure(figsize=(12, 9))
110     plt.subplot(1, 2, 1)
```

```
111     plt.imshow(before_img, cmap='gray')
112     plt.title('Before')
113
114     plt.subplot(1, 2, 2)
115     plt.imshow(after_img, cmap='gray')
116     plt.title('After')
117     plt.show()
118
119
120 if __name__ == '__main__':
121     img_noise = "./ex1_images/noise.png" # <- need to specify the path to the noise image
122     img_gt = "./ex1_images/ori_img.png" # <- need to specify the path to the gt image
123     img = read_img(img_noise)
124     filter_size = 3
125
126     # Mean filter
127     mean_smoothed_img = mean_filter(img, filter_size)
128     show_res(img, mean_smoothed_img)
129     print('PSNR score of mean filter: ', psnr(img, mean_smoothed_img))
130
131     # Median filter
132     median_smoothed_img = median_filter(img, filter_size)
133     show_res(img, median_smoothed_img)
134     print('PSNR score of median filter: ', psnr(img, median_smoothed_img))
```

## 2 Ex212.py - 2D Fourier Transform

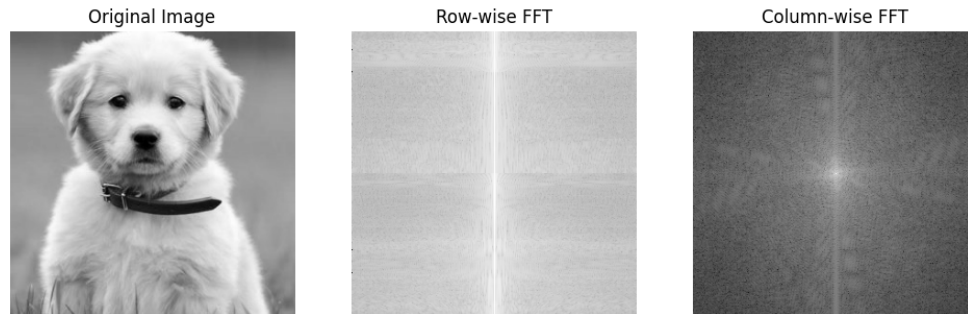


Figure 2: Output for 2D Fourier Transform Exercise

```
1 import numpy as np
2 from skimage import io as io_url
3 import matplotlib.pyplot as plt
4
5
6 def DFT_slow(data):
7     """
8     Implement the discrete Fourier Transform for a 1D signal
9     params:
10         data: Nx1: (N, ): 1D numpy array
11     returns:
12         DFT: Nx1: 1D numpy array
13     """
14     N = len(data)
15     n = np.arange(N)
16     k = n.reshape((N, 1))
17     exp_term = np.exp(-2j * np.pi * k * n / N)
18     return np.dot(exp_term, data)
19
20
```

```
21 def show_img(origin, row_fft, row_col_fft):
22     """
23     Show the original image, row-wise FFT, and column-wise FFT
24
25     params:
26         origin: (H, W): 2D numpy array
27         row_fft: (H, W): 2D numpy array
28         row_col_fft: (H, W): 2D numpy array
29     """
30     fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(12, 8))
31     axs[0].imshow(origin, cmap='gray')
32     axs[0].set_title('Original Image')
33     axs[0].axis('off')
34     axs[1].imshow(np.log(np.abs(np.fft.fftshift(row_fft))), cmap='gray')
35     axs[1].set_title('Row-wise FFT')
36     axs[1].axis('off')
37     axs[2].imshow(np.log(np.abs(np.fft.fftshift(row_col_fft))), cmap='gray')
38     axs[2].set_title('Column-wise FFT')
39     axs[2].axis('off')
40     plt.show()
41
42
43 def DFT_2D(gray_img):
44     """
45     Implement the 2D Discrete Fourier Transform
46     Note that: dtype of the output should be complex_
47     params:
48         gray_img: (H, W): 2D numpy array
49
50     returns:
51         row_fft: (H, W): 2D numpy array that contains the row-wise FFT of the input image
52         row_col_fft: (H, W): 2D numpy array that contains the column-wise FFT of the input
53     """
54     H, W = gray_img.shape
55     row_fft = np.zeros_like(gray_img, dtype=np.complex_)
56     row_col_fft = np.zeros_like(gray_img, dtype=np.complex_)
57
58     for i in range(H):
59         row_fft[i, :] = DFT_slow(gray_img[i, :])
60
61     for j in range(W):
62         row_col_fft[:, j] = DFT_slow(row_fft[:, j])
63
64     return row_fft, row_col_fft
65
66
67 if __name__ == '__main__':
68     # Load the image
69     img = io_url.imread('https://img2.zergnet.com/2309662_300.jpg')
```

```

70     gray_img = np.mean(img, -1) # Convert to grayscale
71
72     # Compute the 2D DFT
73     row_fft, row_col_fft = DFT_2D(gray_img)
74
75     # Display the results
76     show_img(gray_img, row_fft, row_col_fft)
77 import numpy as np
78 from skimage import io as io_url
79 import matplotlib.pyplot as plt
80
81
82 def DFT_slow(data):
83     """
84     Implement the discrete Fourier Transform for a 1D signal
85     params:
86         data: Nx1: (N, ): 1D numpy array
87     returns:
88         DFT: Nx1: 1D numpy array
89     """
90     N = len(data)
91     n = np.arange(N)
92     k = n.reshape((N, 1))
93     exp_term = np.exp(-2j * np.pi * k * n / N)
94     return np.dot(exp_term, data)
95
96
97 def show_img(origin, row_fft, row_col_fft):
98     """
99     Show the original image, row-wise FFT, and column-wise FFT
100
101     params:
102         origin: (H, W): 2D numpy array
103         row_fft: (H, W): 2D numpy array
104         row_col_fft: (H, W): 2D numpy array
105     """
106     fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(12, 8))
107     axs[0].imshow(origin, cmap='gray')
108     axs[0].set_title('Original Image')
109     axs[0].axis('off')
110     axs[1].imshow(np.log(np.abs(np.fft.fftshift(row_fft))), cmap='gray')
111     axs[1].set_title('Row-wise FFT')
112     axs[1].axis('off')
113     axs[2].imshow(np.log(np.abs(np.fft.fftshift(row_col_fft))), cmap='gray')
114     axs[2].set_title('Column-wise FFT')
115     axs[2].axis('off')
116     plt.show()
117
118

```

```
119 def DFT_2D(gray_img):
120     """
121     Implement the 2D Discrete Fourier Transform
122     Note that: dtype of the output should be complex_
123     params:
124         gray_img: (H, W): 2D numpy array
125
126     returns:
127         row_fft: (H, W): 2D numpy array that contains the row-wise FFT of the input image
128         row_col_fft: (H, W): 2D numpy array that contains the column-wise FFT of the input
129     """
130     H, W = gray_img.shape
131     row_fft = np.zeros_like(gray_img, dtype=np.complex_)
132     row_col_fft = np.zeros_like(gray_img, dtype=np.complex_)
133
134     for i in range(H):
135         row_fft[i, :] = DFT_slow(gray_img[i, :])
136
137     for j in range(W):
138         row_col_fft[:, j] = DFT_slow(row_fft[:, j])
139
140     return row_fft, row_col_fft
141
142
143 if __name__ == '__main__':
144     # Load the image
145     img = io_url.imread('https://img2.zergnet.com/2309662_300.jpg')
146     gray_img = np.mean(img, -1) # Convert to grayscale
147
148     # Compute the 2D DFT
149     row_fft, row_col_fft = DFT_2D(gray_img)
150
151     # Display the results
152     show_img(gray_img, row_fft, row_col_fft)
```



### 3 Ex234.ipynb - Frequency Removal Procedure and Hybrid Image

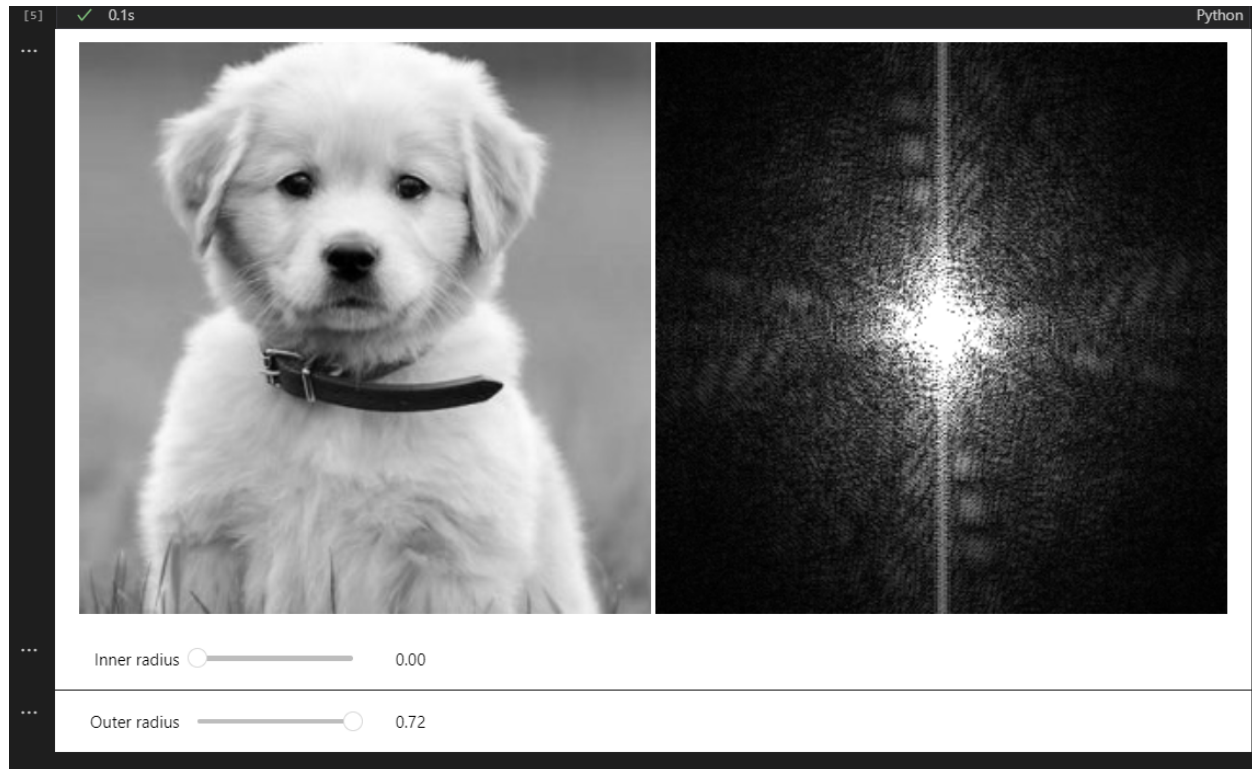


Figure 3: Expected Output for 2D Frequency Removal Exercise

```

1 import ipywidgets as widgets
2 import matplotlib.pyplot as plt
3 import PIL.Image
4 import numpy as np
5 import urllib
6 from skimage.transform import resize
7 from matplotlib.image import imread
8 import os
9 from IPython.display import display
10 from skimage import io as io_url
11 import cv2
12 import numpy as np
13 from PIL import Image
14
15 # Create image widgets
16 image3_spatial = widgets.Image(format='png', width=500, height=500, description='Spatial')
17 image3_freq    = widgets.Image(format='png', width=500, height=500, description='Frequency')
18 sidebyside    = widgets.HBox([image3_spatial, image3_freq])
19
20 # Create slider/select widgets
21 slider_inner   = widgets.FloatSlider(value=0, min=0, max=1, step=0.01, description='Inner radius')
22 slider_outer   = widgets.FloatSlider(value=1.44/2, min=0, max=1.44/2, step=0.01, description='Outer radius')

```

```

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)

```



Figure 4: Hybrid Image

```

1 def read_img(img_path, img_size=(512, 512)):
2     """
3     + c      nh
4     + Chuyen th nh grayscale
5     + Thay i k ch th 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
11
12 def create_hybrid_img(img1, img2, r):
13     """
14     Create hybrid 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 homework
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")
```