CSE166_WI23_assignment_3

January 30, 2023

1 CSE 166: Image Processing, Winter 2023 – Assignment 3

- Instructor: Ben Ochoa
- Due: Monday, February 6, 2023

1.1 Instructions

Please answer the questions below using Python in the attached Jupyter notebook and follow the guidelines below:

- This assignment must be completed **individually**. For more details, please review the Academic Integrity Policy and Collaboration Policy on Canvas.
- All the solutions must be written in the Jupyter notebook only.
- After finishing the assignment in the notebook, please export the notebook as a PDF and **submit both the Notebook and the PDF** (i.e. the `.ipynb` and the `.pdf` files) on Gradescope. While submitting the PDF on gradescope -
  - Make sure that the outputs generated are clearly visible and are not cut-off or partially cropped in the final PDF.
  - Make sure to assign the relevant pages in your PDF submission for each problem.
  - Do not export the jupyter notebook as a single page. Please see the detailed submission instructions at the end of this file.
- You may use basic algebra packages (e.g. `NumPy`, `SciPy`, etc) but you are not allowed to use the packages that directly solve the problems. Feel free to ask the instructor and the teaching assistants if you are unsure about the packages to use.
- It is highly recommended that you begin working on this assignment early.

Late Policy: Assignments submitted late will receive a 15% grade reduction for each 12 hours late (i.e., 30% per day). Assignments will not be accepted 72 hours after the due date. If you require an extension (for personal reasons only) to a due date, you must request one as far in advance as possible. Extensions requested close to or after the due date will only be granted for clear emergencies or clearly unforeseeable circumstances.

2 Problem 1: Textbook problems (11 points)

2.1 a) Problem 4.3 (1 point)

Your answer here

2.2 b) Problem 4.4 (1 point)

Your answer here

2.3 c) Problem 4.9 (2 points)

Your answer here

2.4 d) Problem 4.27 (1 point)

Your answer here

2.5 e) Problem 4.33 (2 points)

Your answer here

2.6 f) Problem 4.40 (3 points)

Your answer here

2.7 g) Problem 4.45 (1 point)

Your answer here

3 Problem 2: Programming: The Fourier transform and filtering in the frequency domain (35 points)

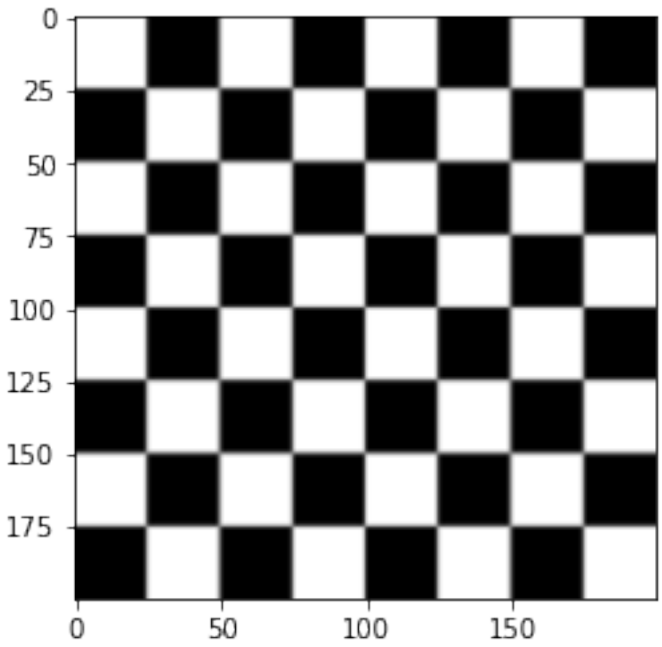
3.1 Part 1: Fourier Transform Pair (10 points)

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data
import math
import cv2
import time
```

[ ]:
```
# Read and display image
img = data.checkerboard()

plt.imshow(img, cmap='gray')
```

[ ]:
```
<matplotlib.image.AxesImage at 0x7f57856e9490>
```



1) Complete the function `dft2d` that computes the 2D discrete Fourier transform of an array. The function takes as input a 2D array and returns as output a 2D array of complex numbers corresponding to the discrete Fourier transform.

[ ]:
```
# function that computes the 2D discrete Fourier transform of an array.
def dft2d(arr):
  """
  arr: input array (H,W)

  returns:
  F: 2D discrete Fourier transform of the input array (H,W)
  """
  # your code here

  return F
```

2) Generate a random numpy array of size 100×100 using `numpy.random.rand` and print the execution time of your `dft2d` function on this array. Compare the execution time of your `dft2d` function with the `numpy.fft.fft2` function.

[ ]:
```
"""
Call the function dft2d with a 100*100 array and print the execution time
Compare the execution time of your dft2d function with the numpy.fft.fft2␣
 ↪function
"""
# your code here
```

3) Why do you think the execution times are different?

Your answer here

4) Complete the function `dft_ift` which, in order,

- Calls the `dft2d` function with the input image to compute the discrete Fourier transform (DFT) $F(u, v)$ (shifted such that the zero-frequency component $F(0, 0)$ is centered) of the input image $f(x, y)$
- Calculates the magnitude $\|F(u, v)\|$ and phase $\phi(u, v)$ of $F(u, v)$
- Calculates the DFT $G(u, v) = \|F(u, v)\|e^{j\phi(u,v)}$
- Computes the inverse discrete fourier transform (IDFT) $g(x, y)$ of $G(u, v)$ (after inverting the centering shift)
- Returns the real part of $g(x, y)$, magnitude $\|F(u, v)\|$ and phase $\phi(u, v)$ of $F(u, v)$

Note: You may use Python built-in functions to perform the intermediate steps.

[ ]:
```
def dft_ift(img):
  """
  img: input image (H,W)

  returns:
  img_g_real: real part of g(x,y) (H,W)
  f_mag: magnitude ||F(u,v)|| (H,W)
  f_phase: phase of F(u,v) (H,W)
  """
  # your code here

  return img_g_real, f_mag, f_phase
```

5) Call the `dft_ift` function with the checkerboard image. Display the input and the output image. Additionally, display figures of the log magnitude $\log\|F(u, v)\|$ and phase $\phi(u, v)$ of the discrete fourier transform $F(u, v)$ of the checkerboard image.

[ ]:
```
"""
Call the dft_ift function with the checkerboard image. Display the input and␣
 ↪the output image.
Additionally, display figures of log(||F(u,v)||) and phase(u,v).
"""
# your code here
```

6) What are the row and column indices of the $F(0, 0)$ component before and after the centering shift?

Your answer here

3.2 Part 2: The Convolution Theorem (15 points)

The objective of this problem is to show that the output image

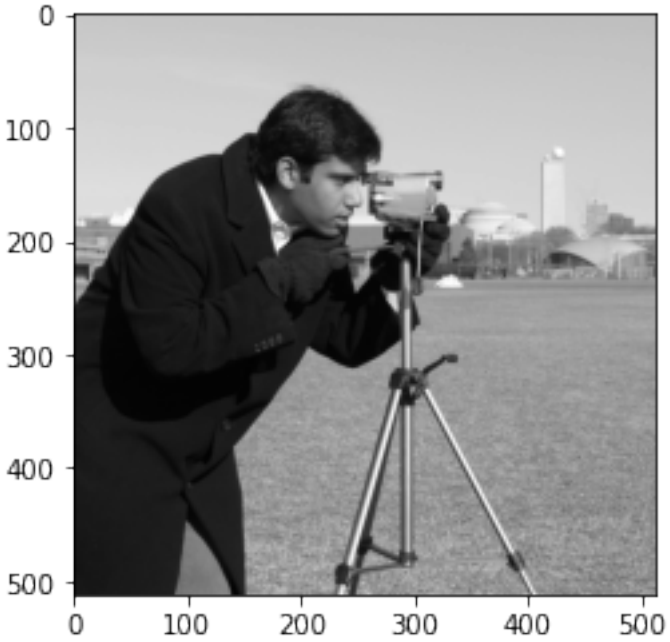
$$g(x, y) = f(x, y)\ h(x, y) = \mathfrak{F}^{-1}\{F(u, v)H(u, v)\}$$

where $F(u, v)$ and $H(u, v)$ are the DFTs of the input image $f(x, y)$ and kernel $h(x, y)$, respectively.

[ ]:
```
# Read and display the image
img = data.camera()

plt.imshow(img, cmap='gray')
```

[ ]:
```
<matplotlib.image.AxesImage at 0x7f9d8a417df0>
```



1) Complete the function `filter_in_frequency_domain` that applies a filter to an image in the frequency domain. The function inputs are a grayscale image and a kernel, and the function output is the filtered (floating point) image corresponding to the input image. The inputs and output are in the spatial domain. Zero padding must be used to mitigate wraparound error. The calculated output image must be the same size as the input image. You may use the Python functions `numpy.fft.fft2`, `numpy.fft.ifft2`, `numpy.fft.fftshift`, `numpy.fft.ifftshift`.

[ ]:
```
# function that applies a filter to an image in the frequency domain
def filter_in_frequency_domain(img, kernel):
  """
  img: input image (H, W)
  kernel: filter (kH, kW)

  returns:
  f_img: image filtered in frequency domain (H, W)
  """
```

```
  # your code here

  return f_img
```

2) Complete the function `conv_theorem` that applies a filter to the image $f(x, y)$ in the frequency and spatial domain. The function takes as input an image and a kernel, and returns the output images filtered in the frequency and spatial domain.

- You must call the function `filter_in_frequency_domain` to apply the filter in the frequency domain.
- You may use the function `cv2.filter2D` to apply the filter in the spatial domain.

[ ]:
```
# function that applies the filter to the image in the frequency and spatial␣
↪domain.
def conv_theorem(img, kernel):
  """
  img: input image (H,W)
  kernel: kernel (kH, kW)

  returns:
  f_img: image filtered in frequency domain (H, W)
  sp_img: image filtered in spatial domain (H, W)
  """
  # your code here

  return f_img, sp_img
```

3) Call the function `conv_theorem` with the Laplacian kernel $h(x, y)$ and the camera image $f(x, y)$, where

$$h(x, y) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Display the input image along with both resulting filtered images.

[ ]:
```
"""
Call the function conv_theorem with the Laplacian kernel and the camera image. ␣
↪
Display the input image along with both resulting filtered images.
"""
# your code here
kernel = np.array([[1, 1, 1], [1, -8, 1], [1, 1, 1]])
sp_img, f_img = conv_theorem(img, kernel)

fig, ax = plt.subplots(1,3, figsize=(16,10))
ax[0].imshow(img, cmap='gray')
ax[1].imshow(sp_img, cmap='gray', vmin=np.min(sp_img), vmax=np.max(sp_img))
ax[1].set_title('Spatial domain filtering')
ax[2].imshow(f_img, cmap='gray', vmin=np.min(f_img), vmax=np.max(f_img))
```

```
ax[2].set_title('Frequency domain filtering')
```

4) Briefly discuss your results. Why would it be beneficial to implement filtering in the frequency domain?

Your answer here

5) **Optional:** Subtract the filtered image from the input image to yield a sharpened image. Display the result.

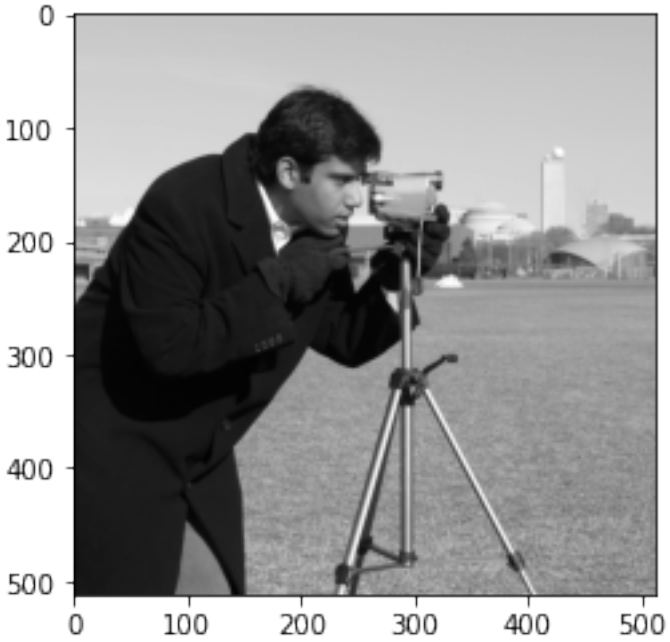
[ ]:
```
"""
Subtract a filtered image from the input image to yield a sharpened image
and display the result.
"""
# your code here
```

3.3 Part 3: Lowpass Filtering (10 points)

[ ]:
```
# Read and display the image
img = data.camera()

plt.imshow(img, cmap='gray')
```

[ ]:
```
<matplotlib.image.AxesImage at 0x7f30803cef40>
```



1) Complete the function `ideal_lpf` that applies an ideal lowpass filter in the frequency domain. The function inputs are an image and a radius for the ideal lowpass filter, and the function

output is the image with ideal low-pass filter applied in the frequency domain.

[ ]:
```
# function that applies ideal low-pass filter in the frequency domain
def ideal_lpf(img, radius):
  """
  img: input image (H,W)
  radius: Radius for the ideal lowpass filter

  returns:
  out: output image after low-pass filter has been applied in the frequency␣
 ↪domain
  """
  # your code here

  return out
```

2) Complete the function `gaussian_lpf` that applies a Gaussian lowpass filter in the frequency domain. The function inputs are an image and a variance for the gaussian lowpass filter, and the function output is the image with Gaussian lowpass filter applied in the frequency domain.

[ ]:
```
# function that applies gaussian low-pass filter in the frequency domain
def gaussian_lpf(img, var):
  """
  img: input image (H,W)
  var: variance for the gaussian lowpass filter

  returns:
  out: output image after low-pass filter has been applied in the frequency␣
 ↪domain
  """
  # your code here

  return out
```

3) Call the function `ideal_lpf` with the camera image and for three different radii values. One of the function call must have radius $D_0 = 50$. Display the original image and all three ideal lowpass filtered images.

[ ]:
```
"""
Call the function `ideal_lpf` with the camera image and for three different␣
 ↪radii.
One of the function call must have radius = 50
Display the original image and all three ideal lowpass filtered images.
"""
# your code here
```

4) Briefly discuss your results

Your answer here

5) Call the function `gaussian_lpf` with the camera image and for three different variance (σ^2) values. Display the original image and all three Gaussian lowpass filtered images.

[ ]:
```
"""
Call the function `gaussian_lpf` with the camera image and for three different␣
↪variance.
Display the original image and all three Gaussian lowpass filtered images.
"""
# your code here
```

6) Briefly discuss the difference between your Gaussian lowpass filtered results and your ideal lowpass filtered results.

Your answer here

4 Submission Instructions

1. Remember to submit **both** the Jupyter notebook file (`.ipynb`) and the PDF version (`.pdf`) of this notebook to Gradescope.
2. Please make sure the content in each cell (e.g. code, output images, printed results, etc.) are clearly visible and are not cut-out or partially cropped in your final PDF file.
3. While submitting on gradescope, please make sure to assign the relevant pages in your PDF submission for each problem.
4. Do not export the jupyter notebook as a single page.

To convert the notebook to PDF, you can choose one way below:

1. You can print the web page and save as PDF (e.g. Chrome: Right click the web page → Print… → Choose "Destination: Save as PDF" and click "Save").
2. You can find the export option in the header: File → Download as → "PDF via LaTeX"
3. You can use nbconvert (https://nbconvert.readthedocs.io/en/latest/install.html) to convert the ipynb file to pdf using the following command

```
jupyter nbconvert --allow-chromium-download --to webpdf <ipynb filename>
```