CSE166_FA23_assignment_5

November 15, 2023

1 CSE 166 Image Processing, Fall 2023 - Assignment 5

Instructor: Ben Ochoa

Assignment due: Wed, Nov 22, 11:59 PM

Name:

PID:

1.1 Instructions

1. All solutions to the problems below must be written in this notebook. Programming aspects of the assignment must be completed using Python (preferably 3.10).
2. This assignment must be completed **individually**. For more details, please review the Academic Integrity Policy and Collaboration Policy on Canvas.
3. The packages you need to complete the assignment will be indicated in the problem descriptions.You are not allowed to use the packages that directly solve the problems. Feel free to ask the instructor and TAs if you are unsure about the packages to use.
4. It is highly recommended that you begin working on this assignment early.
5. After finishing the assignment in the notebook, please export the notebook as a PDF and a python source file. You need to **submit 3 files: the Notebook, the PDF and the python file** (i.e. the `.ipynb`, the `.pdf` and the `.py` files) on Gradescope.

- To convert the notebook to PDF, you can choose one way below:
  - You may first export the notebook as HTML, and then print the web page as PDF
    - e.g., in Chrome: File → Save and Export Notebook as → "HTML"; or in VS-code: Open the Command Palette by pressing Ctrl+Shift+P (Windows/Linux) or Cmd+Shift+P (macOS), search for Jupyter: Export to HTML
    - Open the saved web page and right click → Print... → Choose "Destination: Save as PDF" and click "Save")
  - If you have XeTex installed on your machine, you may directly export the notebook as PDF: e.g., in Chrome, File → Save and Export Notebook as → "PDF"
  - You may use nbconvert to convert the ipynb file to pdf using the following command `jupyter nbconvert --allow-chromium-download --to webpdf filename.ipynb`

- To convert the notebook to python file, you can choose one way below:
  - You may directly export the notebook as py: e.g., in Chrome, File → Save and Export Notebook as → "Executable script"; or in VScode: Open the Command Palette and search for Jupyter: Export to Python Script
  - You may use nbconvert to convert the ipynb file to python file using the following command `jupyter nbconvert --to script filename.ipynb`

6. 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.
7. While submitting on gradescope, please make sure to assign the relevant pages in your PDF submission for each problem.

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 (13 points)

Download the following textbook problems from the Resources tab on Piazza. **Answers to different versions of these problems will not be accepted.**

These textbook problems are conceptual and/or math problems, not programming problems. You are provided with a Markdown cell to answer each problem. Do not change this cell to a Code cell or create a new Code cell. **Answers in the form of code will not be accepted.**

2.1 a) Problem 6.1 (5 points)

Your answer here:

2.2 b) Problem 6.3 (3 points)

Your answer here:

2.3 c) Problem 6.17 (1 point)

Your answer here:

2.4 d) Problem 6.30 (2 points)

Your answer here:

2.5 e) Problem 8.5(a) (1 point)

Your answer here:

2.6 f) Problem 8.9(a) (1 point)

Your answer here:

3 Problem 2: Programming: DCT, DWT, wavelet-based image processing and image compression (55 points)

```
[ ]: import numpy as np
     import matplotlib.pyplot as plt
     from skimage import data
     import pywt
     from skimage.color import rgb2gray
```

3.1 Part 1: The Discrete Cosine Transform (10 points)

In this problem, we will implement the discrete cosine transform (DCT) with matrix based transform, as well as the inverse DCT with matrix based transform and basis images.

1) Create an 8×8 test image named F

$$F = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \\ 25 & 26 & 27 & 28 & 29 & 30 & 31 & 32 \\ 33 & 34 & 35 & 36 & 37 & 38 & 39 & 40 \\ 41 & 42 & 43 & 44 & 45 & 46 & 47 & 48 \\ 49 & 50 & 51 & 52 & 53 & 54 & 55 & 56 \\ 57 & 58 & 59 & 60 & 61 & 62 & 63 & 64 \end{bmatrix}$$

Display F. Set the value range to [min, max] (default).

```
[ ]: # your code here
```

2) Complete the function `get_DCT_transformation_matrix` that computes the DCT transform matrix A which consists of DCT basis vectors.

```
[ ]: # function to compute the DCT transform matrix
     def get_DCT_transformation_matrix(M):
       """
       M: The nuber of basis vectors (integer)

       return:
       A:  DCT transform matrix (M,M)
       """
       # your code here


       return A
```

3) Call the function `get_DCT_transformation_matrix` for $M = 8$ to generate a DCT transform matrix A. Compute the 2D forward DCT of the test image F using matrix-based transforms (i.e., $T = AFA^T$) and the 2D inverse DCT of T (i.e., $F' = A^TTA$).

Display T and F'. Set the value range to [min, max] on all display calls (default).

```
[ ]: # your code here
```

4) The inverse 2D DCT transform can also be performed using basis images.

Complete the function `get_basis_images` that computes the basis images for the Discrete Cosine Transform. The function takes a (M,M) size as input and returns M^2 basis images, each of size M^2.

```
[ ]: # function to compute the basis images for the Discrete Cosine Transform
def get_basis_images(size):
  """
  size: size of the basis image (M,M)

  returns:
  basis_images: basis images of DCT (M,M,M,M)
  """
  # your code here

  return basis_images
```

Call the function `get_basis_images` with size (8,8). Display all the 64 basis images using the `matplotlib.subplots` function. Set the value range to [min, max] on all display calls (default).

```
[ ]: """
Call the function get_basis_images with size (8,8).
Display all the 64 basis images using the matplotlib.subplots function.
"""
# your code here
```

5) Compute the 2D inverse DCT of T (computed above) using basis images.

Display the result. Set the value range to [min, max] (default).

```
[ ]: # your code here
```

3.2 Part 2: The wavelet transform (10 points)

3.3 PyWavelets package

You need to use the PyWavelets package for this assignment. Check out the following links for more details:

https://pywavelets.readthedocs.io/en/latest/

https://pywavelets.readthedocs.io/en/latest/ref/index.html

1) Create a 128×128 array named "img" where every pixel value is 1. Then, use the function `pywt.dwt2` (**not** `pywt.wavedec2`) to obtain the approximation coefficients for decomposition levels 1, 2, 3, 4, and 5 for Haar wavelet decomposition of the image. Print the approximation coefficients for level = 0, 1, 2, 3, 4, and 5.

```
[ ]: # your code here
```

2) Determine the equation that scales pixel values in the original image (i.e., the image at decomposition level $n = 0$) to pixel values in the image at decomposition level n. Ensure this equation holds regardless of the pixel values in the original image.

Your answer here:

3) Complete the function `wavelet_decomposition` that performs the following.

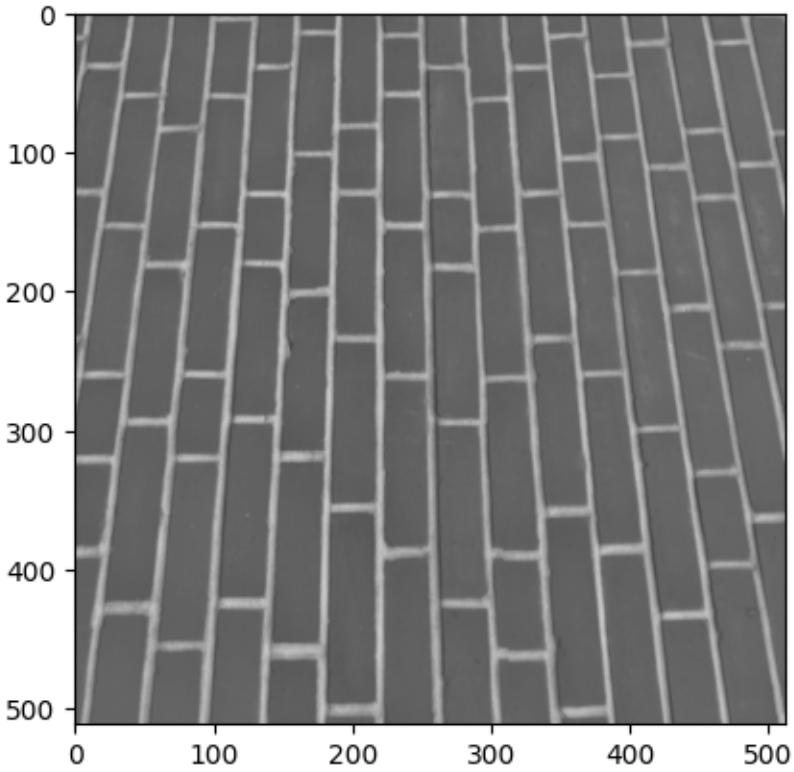
- Uses the function `pywt.dwt2` (**not** `pywt.wavedec2`) to compute a 3-scale Haar discrete wavelet decomposition (hint: 3 uses of `pywt.dwt2`) of the image.
- Calculates the approximate and detail coefficients to create a figure showing 3-level wavelet decomposition, i.e., a figure similar to the one shown in the lower right of lecture 11, slide 43. Ensure the approximation coefficients are scaled correctly using the scale determined in the previous subproblem. (hint: for better visuallization, scale the value of each detail coefficients "image" by 1/2 the maximum magnitude (i.e., absolute value) of the coefficients, then add 1/2 to the result).
- Additionally, returns the approximation coefficients for decomposition levels 1 and 2. Ensure the approximation coefficients are scaled correctly using the scale determined in the previous subproblem.

[ ]:
```
# Read description above for full function description
def wavelet_decomposition(img):
  """
  img: input image (N,N)

  returns:
  res: figure similar to the one shown in the lower right of lecture 11, slide␣
↪43 (N,N)
  A1: decomposition level 1 approximate coefficient (N/2, N/2)
  A2: decomposition level 2 approximate coefficient (N/4, N/4)
  """
  # your code here


  return res, A1, A2
```

4) Call the function `wavelet_decomposition` with the brick image. The image intensity value is alreay scaled such that the intensity range is [0,1].

Display the 3-level wavelet decomposition figure similar to the one shown in the lower right of lecture 11, slide 43. Display the approximation coefficients images for decomposition levels 1 and 2. Set the value range to [0, 1] on all display calls.

[ ]:
```
# Read and display image
img = data.brick()/255.

plt.imshow(img, cmap='gray',vmin=0, vmax=1)
plt.show()
```



[ ]:
```
# your code here
```

3.4 Part 3: Wavelet-based "edge" detection (10 points)

1) Complete the function `edge_detection` that takes an image as input and computes a 1-, 2-, 3-, and 4-scale Symlets 4 wavelet decomposition of the input image. For each resulting decomposition, set the approximation coefficients to zero and perform wavelet reconstruction (back up to decomposition level 0), again using Symlets 4 wavelet filters.

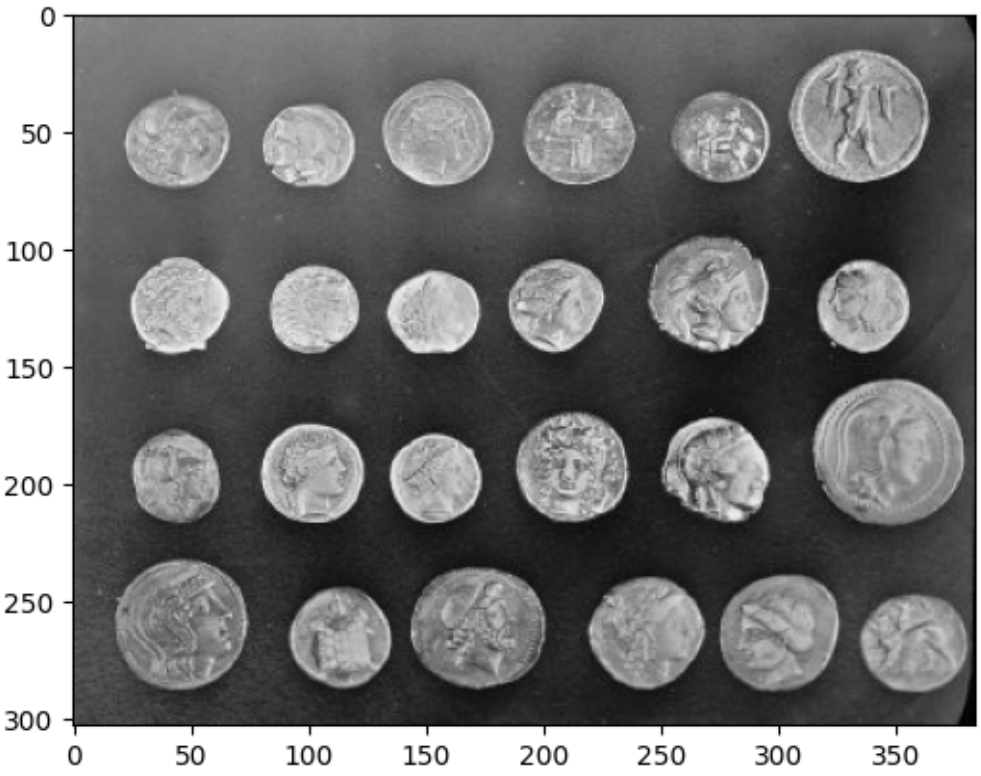
Notes: - Use the functions `pywt.wavedec2` and `pywt.waverec2` (highly recommended), or `pywt.dwt2` and `pywt.idwt2` - 'sym4' corresponds to the Symlets 4 wavelet decomposition in the pywt functions - When calling these functions, ensure you set the mode parameter to 'periodization' (otherwise, you will encounter issues with different sized images)

[ ]:
```
# function that detects edges using Symlets 4 wavelet decomposition
def edge_detection(img):
  """
  img: input image (N,N)

  returns:
  E1: Reconstructed 'edge' image at scale 1 (N,N)
  E2: Reconstructed 'edge' image at scale 2 (N,N)
```

```
    E3: Reconstructed 'edge' image at scale 3 (N,N)
    E4: Reconstructed 'edge' image at scale 4 (N,N)
    """
    # your code here

    return E1, E2, E3, E4
```

2) Call the `edge_detection` function with the coin image. Display the "edge" images for all four scales. Set the value range to [min, max] on all display calls (default).

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

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



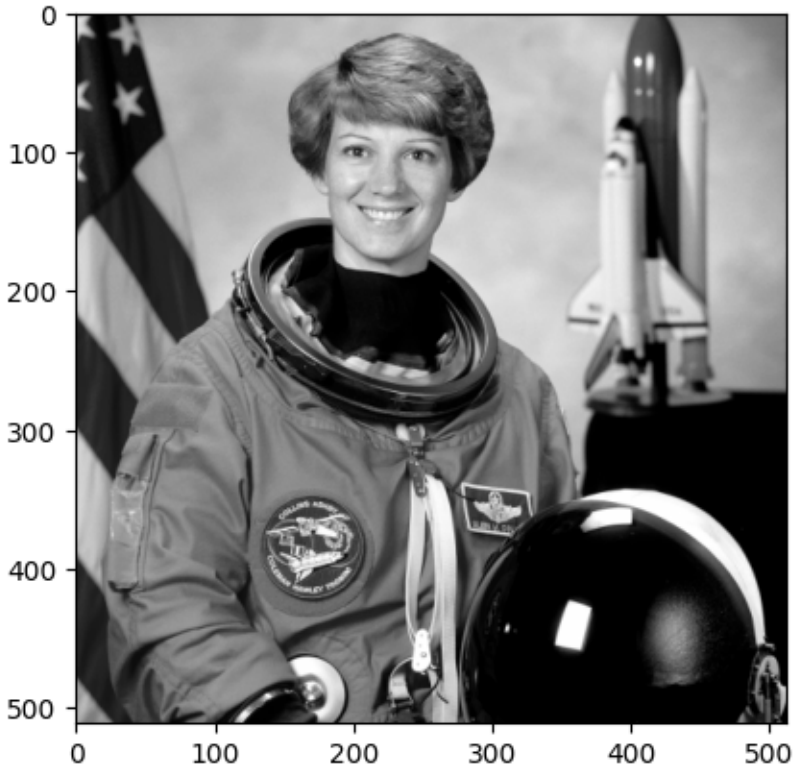
[ ]:
```
# your code here
```

3) Briefly discuss the resulting images, including any differences between them.

Your answer here:

3.5 Part 4: The Discrete Cosine Transform and Lossy Block Processing (10 points)

```
[ ]: # Read and display image
img = 255 * rgb2gray(data.astronaut())

plt.imshow(img, cmap='gray',vmin=0,vmax=255)
plt.show()
```



1) Complete the function `lossy_dct_transform` that takes an input image, a block size M and a percent p which is the percent of the smallest magnitude DCT coefficients that will be set to zero during compression.

The function

- Divides the input image into non-overlapping blocks (i.e., subimages) of size $M \times M$.

- Computes the DCT with matrix-based transform for each block, sets the p percent smallest magnitude (i.e., absolute value) DCT coefficients to zero.
- Computes the reconstructed blocks by using the inverse DCT with matrix-based transform and sythesizes the blocks into one image.
- You **must** use the function `get_DCT_transformation_matrix` you completed in Part 1 (hint: and you should only call it once) and use the transformation matrix for the forward and inverse

transformations.

- Clamps the compressed image between [0,255] and convert it to uint8.

[ ]:
```
# see description above for complete function description
def lossy_dct_transform(img, M, p):
  """
  img: input image (N,N)
  M: Block size (integer)
  p: percent of smallest DCT coefficients that need to be set to zero (floating␣
 ↪point between 0 and 1)

  returns:
  out: output image after processing (N,N)
  """
  # your code here

  return out.astype(np.uint8)
```

2) Call the function `lossy_dct_transform` with the astronaut image, a block size $M = 8$ and each percent $p = 0, 0.5, 0.75, 0.875, 0.94, 0.97$. Display the input image and the resulting output image for each value of p. Set the value range to [0, 255] on all display calls.

Note: you should get a total of 6 processed images.

[ ]:
```
"""
Call the function lossy_dct_transform with the astronaut image, a block size M=8
 and for each p = {0, 0.5, 0.75, 0.875, 0.94, 0.97}.

Display the input image and the resulting output image for each value of p.
"""
# your code here
```

3) Display the error image and print the root-mean-square error associated with each output image.

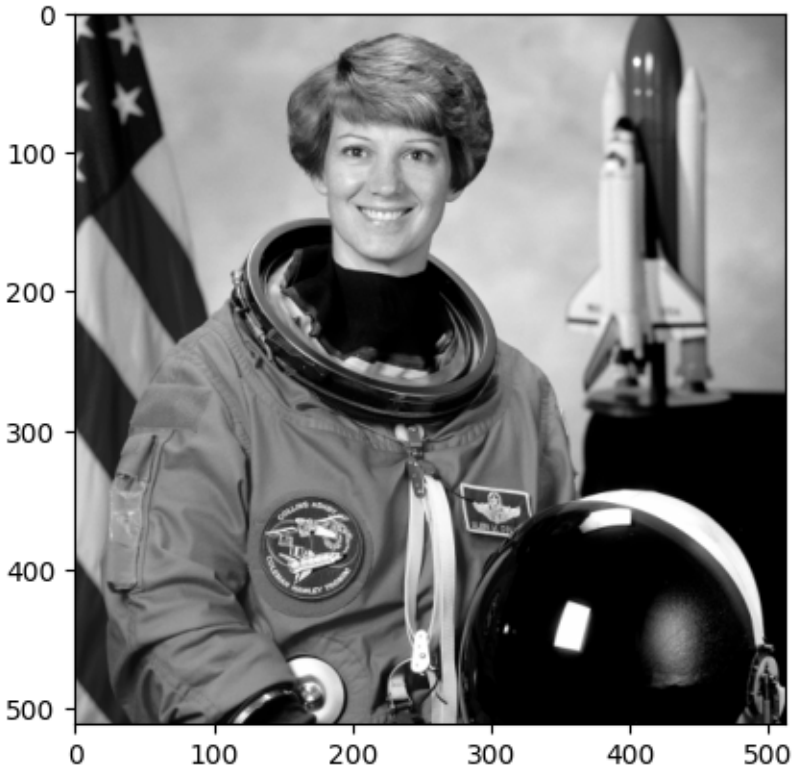
[ ]:
```
"""
Display the error images and the root-mean-square error associated with each␣
 ↪output image.
"""
# your code here
```

4) Briefly discuss the results.

Your answer here:

3.6 Part 5: The Discrete Wavelet Transform and Lossy Processing (15 points)

[ ]:
```
# Read and display image
img = 255 * rgb2gray(data.astronaut())

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



1) Complete the function `lossy_wavelet_transform` that computes the 2-level Discrete Wavelet Transform (DWT), sets p percent of the smallest magnitude detail coefficients to zero, and computes the inverse DWT. The function takes an image, the wavelet type and an floating-point number p as input, and returns the output image after processing. Clamps the values of reconstructed image between [0,255] and convert the image to uint8.

Note:

- Use the functions `pywt.wavedec2` and `pywt.waverec2` (highly recommended), or `pywt.dwt2` and `pywt.idwt2`

[ ]:
```
# see description above for full function description
def lossy_wavelet_transform(img, wavelet, p):
  """
  img: input image (N,N)
```

```
    wavelet: wavelet algorithm to use ('haar' or 'db4' or 'bior4.4')
    p: percent of smallest magnitude detail coefficients that need to be set to␣
    ↪zero (floating point between 0 and 1)

    returns:
    out: output image after processing

    """
    # your code here

    return out
```

2) Call the function `lossy_wavelet_transform` with the astronaut image, for each wavelet type {Haar, Daubechies, and biorthogonal} and for each $p = 50\%, 75\%, 87.5\%, 97\%$. Display the output images. Set the value range to [0, 255] on all display calls.

Notes: - You can specify the wavelet types as 'haar', 'db4', and 'bior4.4' to the pywt functions for the Haar, Daubechies, and biorthogonal wavelets, respectively. - you should get a total of 12 output images

[ ]:
```
"""
Call the function lossy_wavelet_transform with the astronaut image, for each
wavelet type {Haar, Daubechies, and biorthogonal} and for each p = {50%, 75%,␣
 ↪87.5%, 97%}

Display the output images (12 images).
"""
# your code here
```

3) Display the error images and print the root-mean-square error associated with each output image.

[ ]:
```
"""
Display the error images (12 images) and print the root-mean-square error␣
 ↪associated with each output image.
"""
# your code here
```

4) Briefly discuss the resulting images, including how the choice of wavelet and p effects compression quality. Also, discuss any differences you observe with compression based on the DCT.

Your answer here: