Homework 2

AMATH 582/482, Winter 2026

Due on Feb 2, 2026 at midnight.

Directions, Reminders and Policies

Read these instructions carefully:

- **You are required to upload a PDF report to Gradescope along with your code in the form of a Jupyter notebook or a zip file of your main scripts and functions. The report and the code are to be submitted under separate assignments. DO NOT INCLUDE YOUR CODE IN THE REPORT.**
- The report should be a maximum of 10 pages in accordance with the template on Canvas including references. Minimum font size 11pts and margins of at least 1inch on A4 or standard letter size paper. The report should be formatted as follows:
  - Title/author/abstract: Title, author/address lines, and short (100 words or less) abstract. This is not meant to be a separate title page.
  - Sec. 1. Introduction and Overview
  - Sec. 2. Theoretical Background
  - Sec. 3. Algorithm Implementation and Development
  - Sec. 4. Computational Results
  - Sec. 5. Summary and Conclusions
  - Acknowledgments. Mention other students if you were part of a study group.
  - References
- **Recall our late policy: If you plan to use your tokens please send an email to Saba (`heravi@uw.edu`). If you do not use a token you will lose 1/3 of your grade for each day the submission is late. WE WILL NOT AUTOMATICALLY APPLY TOKENS. YOU MUST INFORM US IF YOU PLAN TO USE THEM.**

Problem Description: Edge Detection using Wavelets

Your goal in this homework is to design an image edge detector using wavelet transforms and thresholding. You will use the PyWavelets package in Python to perform wavelet transforms and inverse transforms. Download and run the helper notebook `HW2_Helper.ipynb` on Canvas to get started. This notebook will load a benchmark image from the PyWavelets package, add noise to it, and visualize the results.

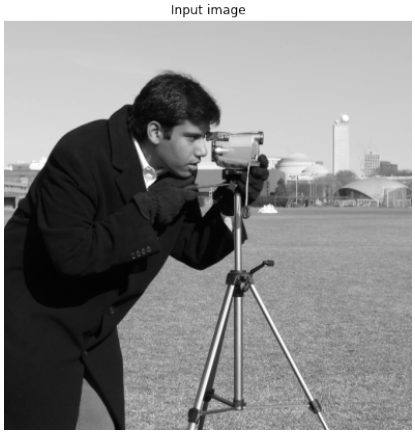

Figure 1 The benchmark image "camera" from the PyWavelets package.

The guiding principle behind your edge detector is the fact that edges correspond to singularities in the image, and wavelet transforms are effective at capturing these singularities. By applying a wavelet transform to the noisy image, you can identify coefficients that correspond to edges as those will have larger magnitudes compared to the coefficients associated with flat regions. The presence of these large coefficients is also robust to noise as edge coefficients will appear across multiple scales in the wavelet decomposition while noise tends to affect the coefficients at a particular scale (typically the finest).

Figure 2 shows the magnitude of horizontal, vertical, and diagonal wavelet coefficients at level 1 of the image as well as the total magnitude of the wavelet coefficients at level 1, i.e.,

$$C_{i,j} = \sqrt{|CH_{i,j}|^2 + |CV_{i,j}|^2 + |CD_{i,j}|^2},$$

where the matrices CH, CV, and CD represent the horizontal, vertical, and diagonal wavelet coefficients, respectively. As you can see in the figure, the wavelet coefficients for indices that are close to edges in the image have significantly larger magnitudes relative to other coefficients in the transform.

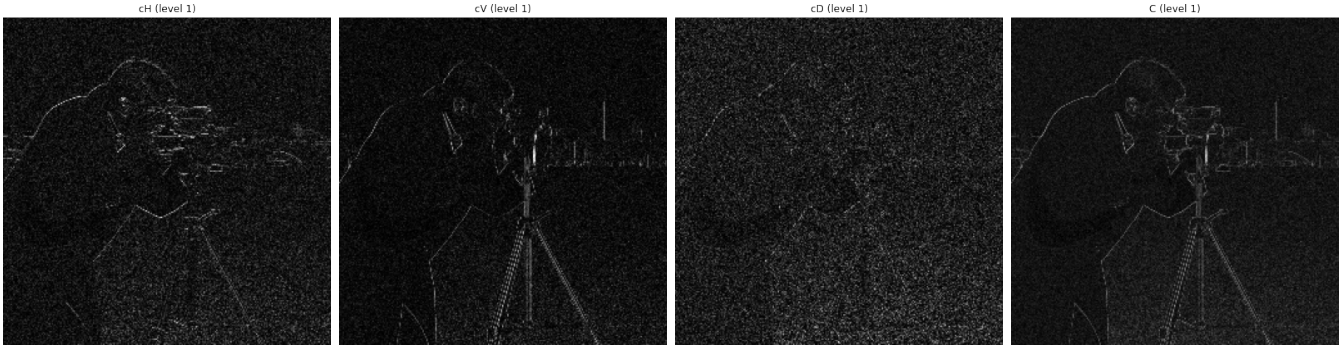

Figure 2 Relative magnitude of horizontal, vertical, diagonal, and total wavelet coefficients at fine scale.

Tasks

Below is a list of tasks to complete in this homework and discuss in your report.

1. Implement a method for the detection of edges in an image using the wavelet transform by thresholding the wavelet coefficients, i.e., by finding the pixels that correspond to the large wavelet coefficients. Use your method to create a binary edge map of the noisy image, i.e., an image with black and white pixels where the white pixels correspond to detected edges while black pixels correspond to non-edge regions; see Figure 3. Clearly explain your method in the report.

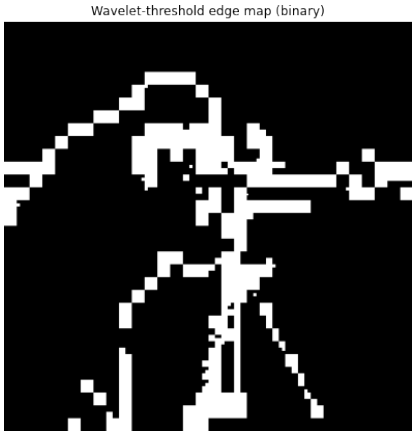

Figure 3 Example binary edge map of an image. White pixels correspond to detected edges while black pixels correspond to non-edge regions.

2. Test the performance of your edge detector on noisy instances of the image. There is code to add noise to the image in the helper notebook. Test your method with noise levels $\sigma = 0.1, 0.25, 0.5$ (standard deviation of the Gaussian noise).
3. Further test the performance of your method with at least three different wavelets, e.g., Haar, Daubechies, Symlets, Coiflets, etc.
4. Take natural images using your cell phone or a camera and apply your edge detector to them. Try different images, low/high light, low/high contrast, etc, and discuss how well your method performs. Be sure to convert the images to grayscale before applying your edge detector to simplify the process and avoid having to deal with color channels. Also downsample the images to a managable size (e.g. 512x512) to improve computational times.

Some comments and hints

Here are some useful comments and facts to guide you along the way.

1. You can apply your edge detector on the finest scale wavelet coefficients (level 1) but you will get better results if you combine information from multiple scales when the noise level is high.
2. There are many ways of creating the mask. The simplest is to modify the wavelt coefficients to get a low resolution binary image and then upsample it to the original image size using built in functions in `skimage` or `OpenCV`. Another way is to directly create a binary mask at the original image size by mapping the wavelet coefficients to the original image pixels by carefully considering the support of the wavelet basis functions.