Image Analysis Coursework Report

CRSiD: tmb76

University of Cambridge

Contents

Chapter 1

Fundamentals of Image Analysis

1.1 Exercise 1.1

In this exercise, 3 images were used to test different segmetentation methods. For each image, a specific segmentation objective was set, and 3 differemt main algorithms were used to segment the images, along with some additional methods.

1.1.1 Image 1: `CT.png`

The first image, `CT.png`, is a CT scan of a human torso. The objective of this segmentation is to segment the lungs from the rest of the image, including any tissue or nodule inside it. This segmentation was done using a region-growing algorithm. Region-growing (or the `skimage.segmentation.flood_fill` function) is a conceptually simple algorithm. Starting from a seed-pixel with a certain value, the algorithm grows the region by adding neighbouring pixels that are within a certain threshold of the current region. This is done iteratively until the region stops growing [5] [3, pp.764-766].

The first part of the segmentation was to divide the image into regions, from which it was then possible to set seeds for the region-growing algorithm inside each lung. Getting the regions was done using thresholding and closing. Thresholding is a simple image processing method where by setting a threshold value, one creates a binary image where all pixels above the threshold are set to 1 (0 otherwise) [3, p.743]. Setting the threshold value is done by selecting the value which maximises the between-class variance of the resulting binary image [3, pp. 747-751]. Closing is a morphological operation that is used to fill in small holes (locations with value 0) in the binary image. The first step is to dilate the image then erode it [3, pp.645-648]. Erosion and dilation are two morphological operation that can be considered as filtering operation. The former filters image details smallet than the structuring

element, which is a small template set of pixels used to filter the image [3, pp. 636-638]. The latter filters images in a similar way, but by switching what is considered the background of the image and what is cosnidered an object. Here, this means that erosion will remove small 1-valued regions, while dilation will remove small 0-valued regions.

Once this is done, regions are identified. This is done by taking each continuous area of a single value in the binary image, and assigning it a unique label [6]. From there, the lungs were identified as the 2 smallest regions. The seeds for the region-growing algorithm were then set as the middle point in that region (not centroid, but middle of the array). The region-growing algorithm is then used, obtaining segmentations of the lungs. Thresholding and closing where then again used to get smooth and continuous segmentations. This entire process is shown in Figure 1.1.

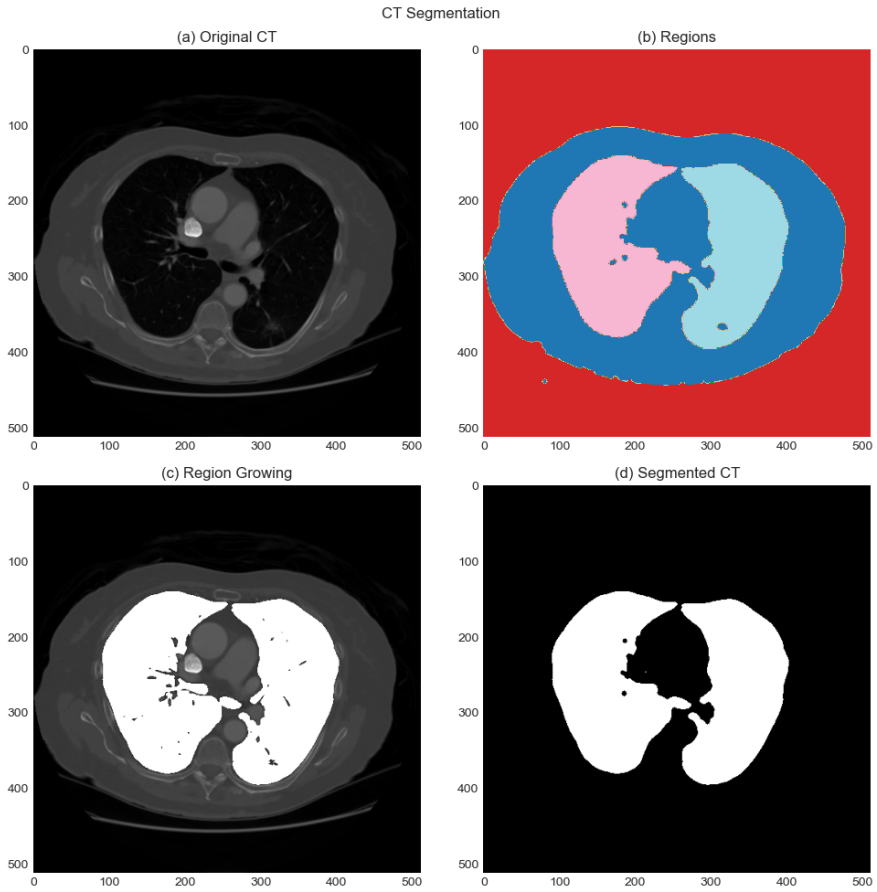


Figure 1.1: The segmentation of the `CT.png` image.

1.1.2 Image 2: `noisy_flower.jpg`

Here, the task was to segment purple tulips from an image of a Dutch tulip field. The main challenge here was the presence of noise in the image, resulting in the purple tulips not being a uniform colour, and other tulips containing some purple. Since the main information here is the color and not so much the brightness of the image, the segmentation was done in the HSV color space. This is a color space that encodes images as hue (color), saturation (intensity of that color), and value (brightness) making it easier to segment based on color [9]. Plotting grayscale images of each of those channels (see Figure 1.2), it can be seen that the hue channel is most useful for segmentation.

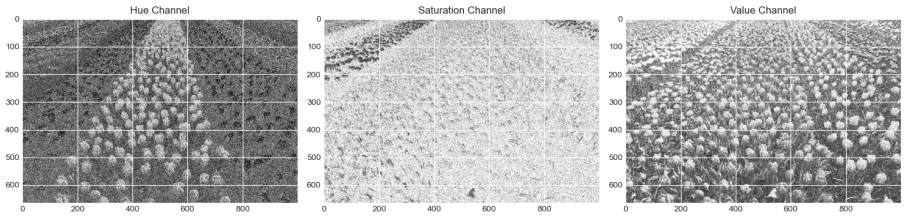


Figure 1.2: The HSV channels of the `noisy_flower.jpg` image.

It can also be seen that, conveniently, the purple hue is close to the largest value on the spectrum, with the exception of red which is at both extremities. And looking at the histogram of the hue channel, with hue overlaid, one can see that using a single threshold may already be very helpful in segmenting the image (see 1.3). Applyin that threshold, the resulting image will be a noisy binary image. But by applying opening (erosion then dilation, see Sec. 1.1.1.) to the image, this removes the darker noise in lighter regions. A gaussian filter is then applied [3, pp. 166-170], smoothing out the binary image's noise, and giving back a grayscale image.

Finally, the main algorithm used for this image is the Chan-Vese segmentation algorithm [1]. This algorithm performs well for objects without clear boundaries. It works by iteratively updating a level set to minimize an energy/loss function, defined as the sum of intensity difference from the average values inside and outside segmented regions [7]. Finally, opening is applied to the segmented image to remove small regions segmented by Chan-Vese from the noise, which do not correspond to the purple tulips. The result of this complete process is shown in Figure 1.4.

1.1.3 Image 3: `coins.png`

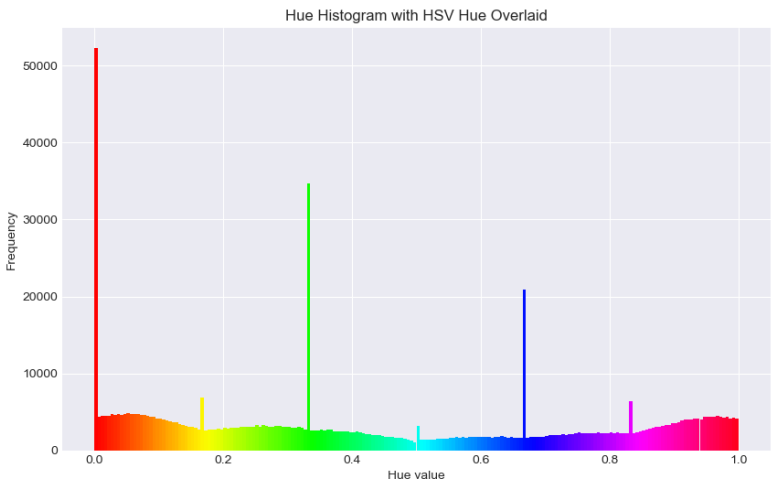


Figure 1.3: The histogram of the hue channel of the `noisy_flower.jpg` image.

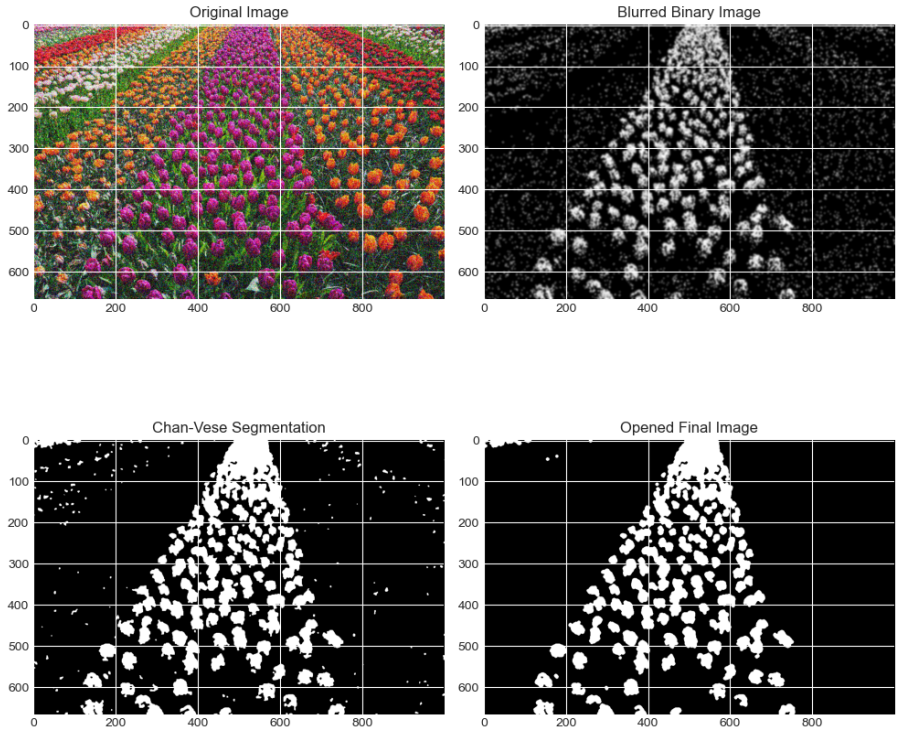


Figure 1.4: The segmentation process of the `noisy_flower.jpg` image.

Chapter 2

Inverse Problems and Multiresolution Analysis

2.1 Exercise 2.1

In this exercise, the solving of ill-posed inverse problems is done for a simple 1D example of fitting a straight line to noisy data. This will be done using a minimisation algorithm of the form:

$$\min_{u} ||Au - f||_{l_n} \tag{2.1}$$

Where A is the parameter matrix, u is the measured input, and l_n describes the type of distance used to measure the error between the model and the data. Here, A is the vector $[a, b]$, where a is the slope and b is the intercept of the fitted line. And the two choices considered are l_1 and l_2 minimisation.

2.1.1 l_1 and l_2 Minimisation

As defined, l_1 minimisation is the minimisation of the sum of the absolute values of the residuals [10], while l_2 minimisation is the minimisation of the sum of the squares of the residuals [4]. Switching to a more explicit notation of this problem, this gives:

$$||(ax + b) - f||_{l_1} = \sum_{i=1}^{n} |(ax_i + b) - f_i| \tag{2.2}$$

$$||(ax + b) - f||_{l_2} = \sqrt{\sum_{i=1}^{n} ((ax_i + b) - f_i)^2} \tag{2.3}$$

	Slope	**Intercept**
Noisy Data	0.0663	0.3092
Data w/ Outliers	0.0662	0.3120

Table 2.1: Table of the fitted parameters for the straight line using l_1 minimisation.

The l_1 minimisation problem is quite uncommon due to it not being differentiable, and resulting in the problem not having an analytical solution.

In this problem, some noisy data was obtained from two files, `y_line.txt` and `y_outlier_line.txt`, where the latter has the same data but with one outlier. The data was then fitted with a straight line using the two minimisation methods. For l_1, the `scipy.optimise.minimise` function was used using the Sequential Least Squares Programming (SLSQP) method. This is a way to solve the minimizing problem using linear programming, by replacing the original problem with a series of quadratic problems that are easier to solve [2]. Using this method, the fitted parameters were otbtained and are shown in Table 2.1, and the plots of the fitted lines are shown in Figure 2.1.

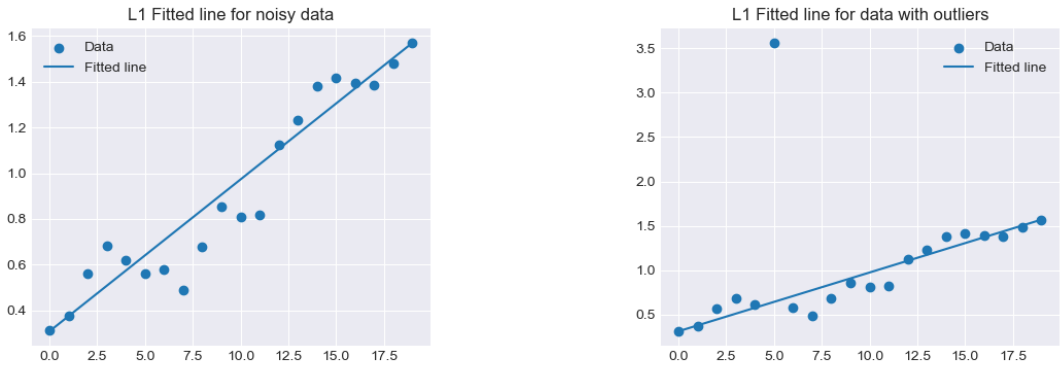


Figure 2.1: Plots of the fitted straight line to the noisy (left), and outlier (right) data using l_1 minimisation, assuming the x-coordinate of the data points is the index of the data point.

For l_2 minimisation, the `scikit-learn` library was used to fit the data using the `LinearRegression` class, which uses the Ordinary Least Squares method to fit the data [8]. The fitted parameters were obtained and are shown in Table 2.2, and the plots of the fitted lines are shown in Figure 2.2.

	Slope	**Intercept**
Noisy Data	0.0665	0.2838
Data w/ Outliers	0.0462	0.6266

Table 2.2: Table of the fitted parameters for the straight line using l_2 minimisation.

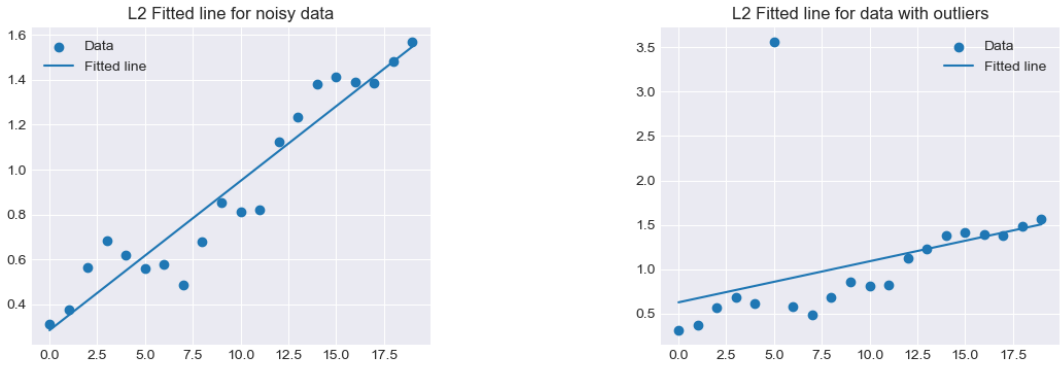


Figure 2.2: Plots of the fitted straight line to the noisy (left), and outlier (right) data using l_2 minimisation, assuming the x-coordinate of the data points is the index of the data point.

2.1.2 Discussion

It is clear that l_1 minimisation is more robust to outliers than l_2 minimisation, as the l_1 fitted line for the data with outliers is almost identical to the line fitted to the noisy data. Whereas the l_2 fitted line for the data with outliers differs significantly from the line fitted to the noisy data. This can be explained from the definition of the two minimisation methods, where due to the squaring of the difference in l_2 minimisation, the effect of outliers on the objective function ($||(ax+b)-f||_{l_2}$) is amplified.

2.2 Exercise 2.2

In this exercise, the importance of random undersampling in compressed sensing theory. In order to do this, random and uniform undersampling will be compared in the reconstruction of a noisy signal, by solving:

$$\arg\min\left(\frac{1}{2}||\mathcal{F}_\Omega\hat{x}-y||_2^2+\lambda|\hat{x}|_1\right) \tag{2.4}$$

Using an iterative soft thresholding algorithm, with a data consistency constraint:

$$\hat{f}_{i+1}[j]=\begin{cases}\hat{f}_i[j] & \text{if } y[j]=0\\ y[j] & \text{otherwise}\end{cases} \tag{2.5}$$

2.2.1 Getting undersampled signals

First, a noisy signal is created. This is done by first creating a zero-filled vector of length 100 and replacing 10 entries by non-zero coefficients between 0 and 1. Gaussian noise, with mean 0 and standard deviation 0.05 is then added to all entries. This results in a noisy vector shown in Figure2.3.

The signal is then undersampled using two methods: random and uniform. For random undersampling, 32 entries from the vector are sampled randomly (without repeats). For uniform undersampling, `numpy.linspace` was used to get 32 regularly spaced samples from the noisy signal. The inverse Fourier Transform of the undersampled signal is computed and then multiplied by 4. The resulting "measured" undersampled signals are shown in Figure 2.4.

With the undersampled signals the goal is now to reconstruct the original signal

2.2.2 The Iterative Soft Thresholding Algorithm

In order to reconstruct the original signal, the iterative soft thresholding algorithm is used. This algorithm is an iterative method that solves the minimisation problem

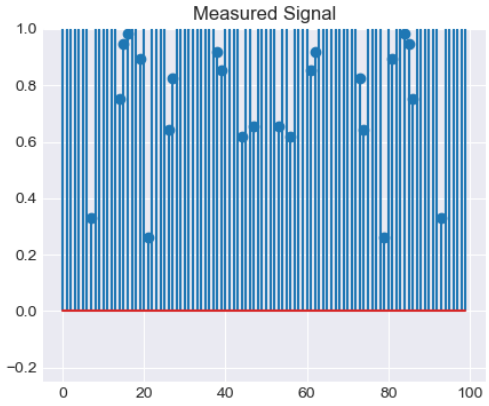
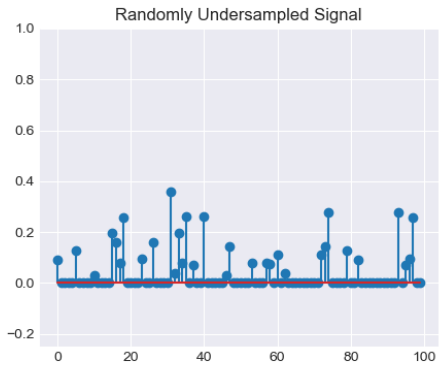
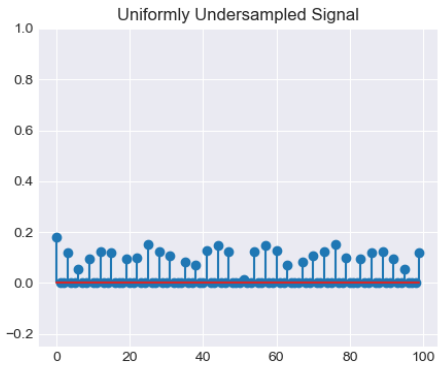


Figure 2.3: Plot of the noisy signal.



(a) Random Undersampling



(b) Uniform Undersampling

Figure 2.4: Plots of the undersampled signals.

described above by iteratively applying a soft thresholding function to the signal. The algorithm is as follows:

Algorithm 1: Iterative Soft Thresholding Algorithm

$\hat{X}_0 \leftarrow y$ ▷ Initial guess
for $i \leq N_{iter}$ **do** ▷ Iterate $i = 0, 1, \ldots, N_{iter}$
  $\hat{x}_i \leftarrow \mathcal{F}^{-1}\hat{X}_i$ ▷ Get an estimate of the signal
  $\hat{x}_i \leftarrow \text{SoftThresh}(\hat{x}_i, \lambda)$ ▷ Apply Soft Thresholding
  $\hat{X}_i \leftarrow \mathcal{F}\hat{x}_i$ ▷ Compute Fourier Transform
  if $y[j] = 0$ **then** ▷ Data Consistency Constraint
    $\hat{X}_{i+1}[j] \leftarrow \hat{X}_i[j]$
  else
    $\hat{X}_{i+1}[j] \leftarrow y[j]$
  end if
  $i \leftarrow i + 1$
end for

This could also be set up as a while loop, where the algorithm stops when the difference between the current and previous estimate is below a certain threshold, i.e. when it converges

2.2.3 Discussion

2.3 Exercise 2.3

In this exercise, compressed reconstruction is discussed, specifically how images can have sparse representations in the wavelet transform domain. In this case, the wavelet transform used is the Daubechies wavelet transform, and the image is then reconstructed. The process of thresholding the wavelet coefficients is also discussed. In Figure 2.5, the image studied is shown.

2.3.1 Daubechies Wavelet Transform

EXPLAIN THE DAUBECHIES WAVELET TRANSFORM

Once computed, the wavelet coefficients can be visualised as an image, as shown in Figure 2.6. As can be seen, each describes details of the image at different scales.

One can then reconstruct it using the inverse wavelet transform, and the image is shown in Figure 2.7.

And comparing this with the original image (2.8), it can be seen that the reconstruction is quite good, with the main features of the image being preserved.

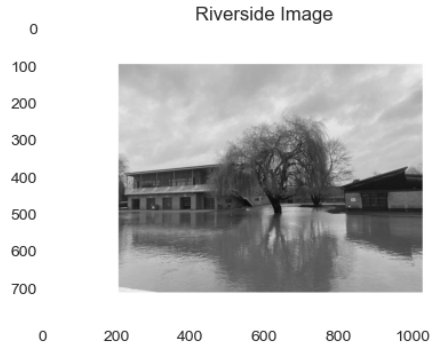


Figure 2.5: The riverside image studied in this exercise.

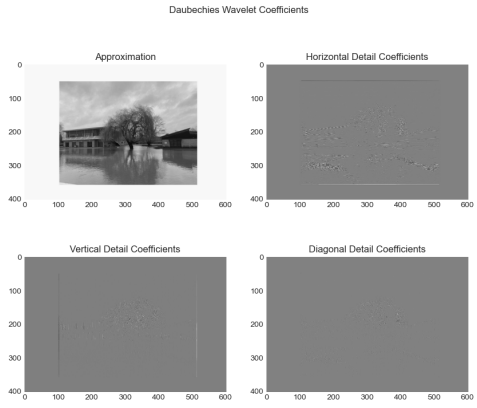


Figure 2.6: The Daubechies wavelet transform coefficients of the riverside image.

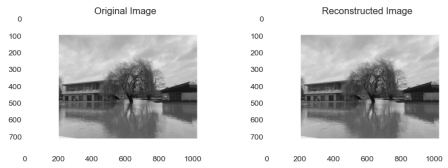


Figure 2.7: The reconstructed riverside image using the Daubechies wavelet transform.

However, one can also see that there is a significant amount of noise in the reconstruction, or indeed in the original image. This is where thresholding comes in.

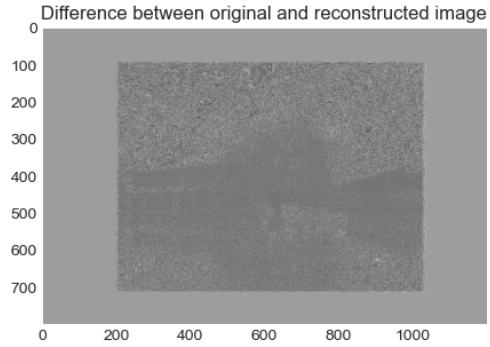


Figure 2.8: The difference between the original and reconstructed riverside images.

2.3.2 Thresholding Coefficients

For the second part of this exercise, the task was to threshold the wavelet coefficients, keeping only the largest $P\%$ of the coefficients. This is done using the `pywt.threshold` function from the `pywavelets` library. The thresholding is done using the hard thresholding method, which is simply defined as:

$$\text{HardThresh}(x, \lambda) = \begin{cases} x & \text{if } |x| > \lambda \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

Thus, this method is used for the top 2.5, 5, 10, 15, and 20% of the coefficients. The result for the top 15% threshold is shown in figure 2.9.

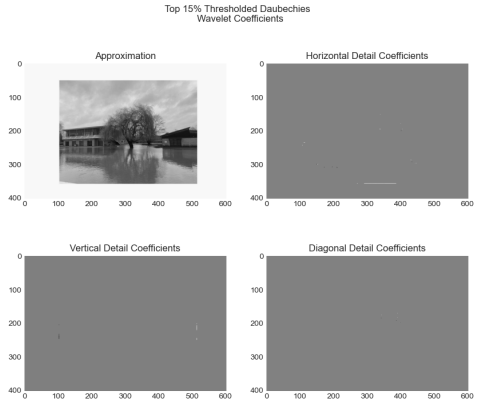


Figure 2.9: The thresholded riverside image Daubechies wavelet transform coefficients, for a top 15% threshold.

For the other thresholds, the results are shown in Figures 4.1, 4.2, 4.3, and 4.4, in the Appendix. For each of these thresholds, the riverside image was reconstructed and compared to the original image. Again, this is first done for the 15% threshold, and the results are shown in Figures 2.10, 2.11.

For the other thresholds, the results are shown in Figure 4.5, 4.7, 4.9, and 4.11, in the Appendix. The difference between the original and reconstructed images is shown in Figure 2.11.

2.3.3 Discussion

As expected, the higher the threshold the less coefficient's information is kept. However, reconstructions stays qualitatively good, and the diff figures show less noise in the reconstructed images. This is because the noise is often in the smaller coefficients, and by thresholding these out, the noise is removed. However, the thresholding also removes some of the detail in the image, as can be seen in the reconstructed images. This is a trade-off that must be considered when thresholding wavelet coefficients.

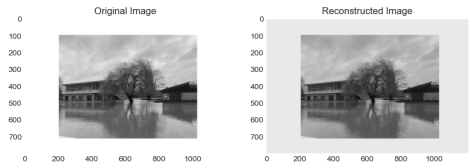


Figure 2.10: The reconstructed riverside image using the thresholded Daubechies wavelet transform coefficients, for a top 15% threshold.

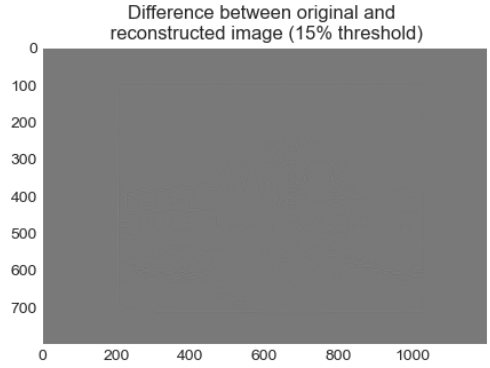


Figure 2.11: The difference between the original and reconstructed riverside images, for a top 15% threshold.

Chapter 3

Solving Inverse Problems

3.1 Exercise 3.1

3.2 Exercise 3.2

3.3 Exercise 3.3

Chapter 4

Appendix

4.1 README

4.1.1 Extra Plots

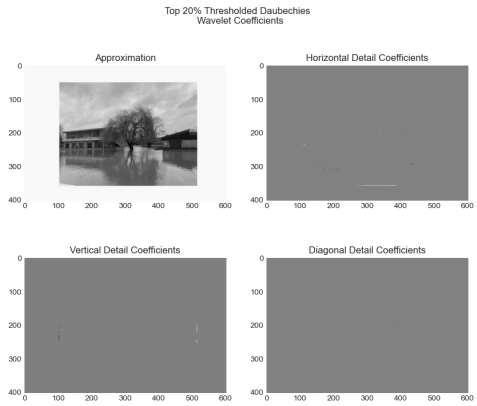


Figure 4.1: The thresholded riverside image Daubechies wavelet transform coefficients, for a top 20% threshold.

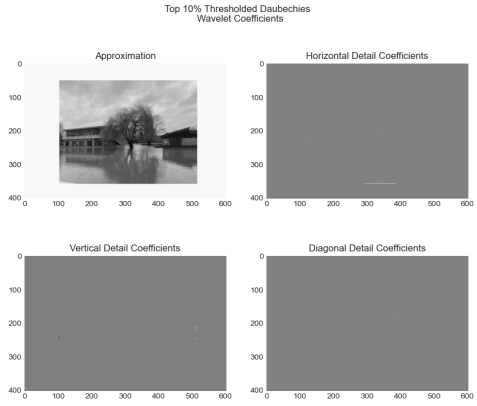


Figure 4.2: The thresholded riverside image Daubechies wavelet transform coefficients, for a top 10% threshold.

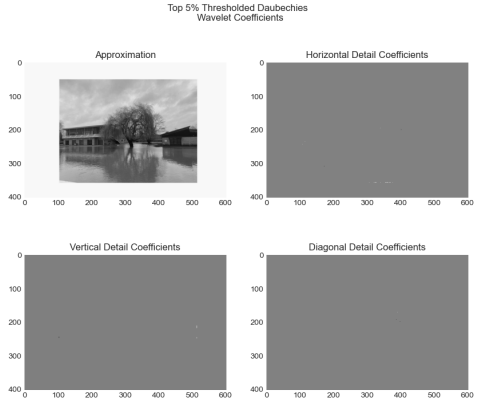


Figure 4.3: The thresholded riverside image Daubechies wavelet transform coefficients, for a top 5% threshold.

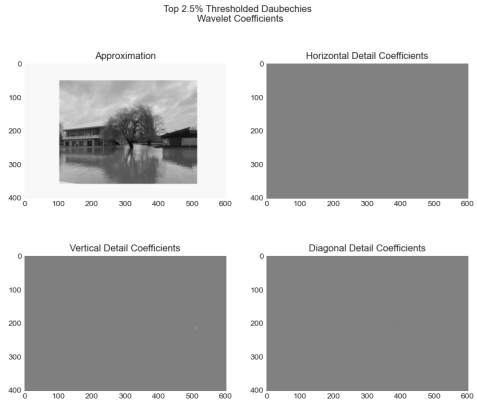


Figure 4.4: The thresholded riverside image Daubechies wavelet transform coefficients, for a top 2.5% threshold.

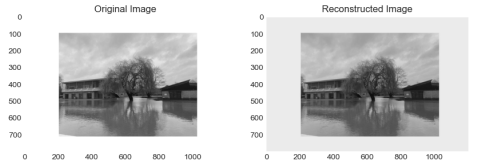


Figure 4.5: The reconstructed riverside image using the thresholded Daubechies wavelet transform coefficients, for a top 20% threshold.

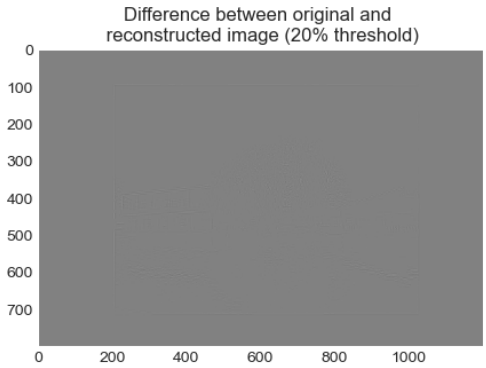


Figure 4.6: The difference between the original and reconstructed riverside images, for a top 20% threshold.

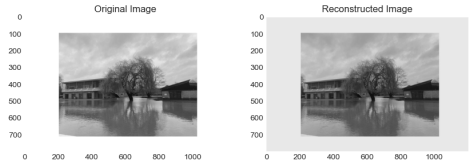


Figure 4.7: The reconstructed riverside image using the thresholded Daubechies wavelet transform coefficients, for a top 10% threshold.

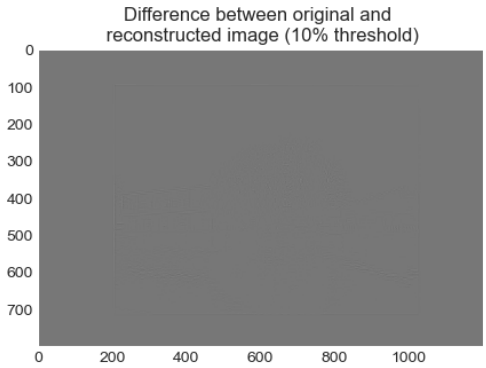


Figure 4.8: The difference between the original and reconstructed riverside images, for a top 10% threshold.

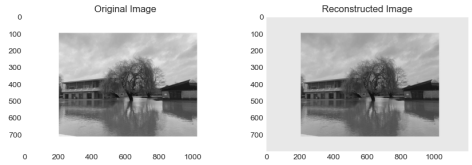


Figure 4.9: The reconstructed riverside image using the thresholded Daubechies wavelet transform coefficients, for a top 5% threshold.

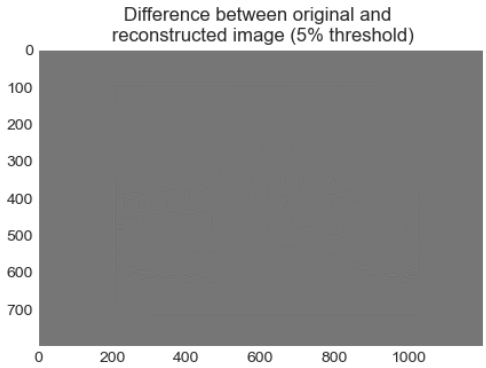


Figure 4.10: The difference between the original and reconstructed riverside images, for a top 5% threshold.

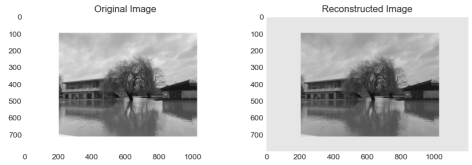


Figure 4.11: The reconstructed riverside image using the thresholded Daubechies wavelet transform coefficients, for a top 2.5% threshold.

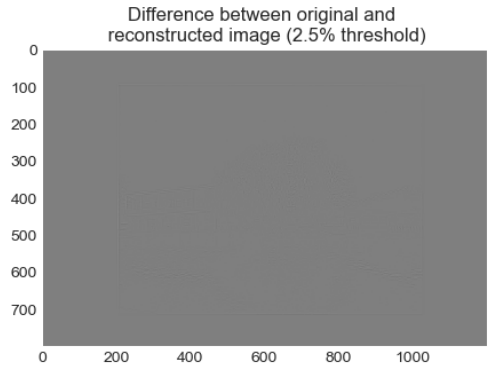


Figure 4.12: The difference between the original and reconstructed riverside images, for a top 2.5% threshold.

Bibliography

[1] Tony Chan and Luminita Vese. An active contour model without edges. In *Scale-Space Theories in Computer Vision*, 1999.

[2] Math Stack Exchange. How does slsqp (sequential least squares programming) algorithm work? *Math Stack Exchange*, 2021.

[3] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Pearson, 4th edition, 2002.

[4] Margalit and Rabinoff. The method of least squares. *Interactive Linear Algebra*, 2021.

[5] Scikit-image. Flood fill. *Scikit-image Documentation*, 2021.

[6] Scikit-image. Label. *Scikit-image Documentation*, 2021.

[7] Scikit-image. Segmentation. *Scikit-image Documentation*, 2021.

[8] Scikit-learn. Linear regression. *Scikit-learn Documentation*, 2021.

[9] Wikipedia. Hsl and hsv. *Wikipedia, The Free Encyclopedia*, 2021.

[10] Wikipedia. Least absolute deviations. *Wikipedia, The Free Encyclopedia*, 2021.