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 segmentation methods. For each image, a specific segmentation objective was set, and 3 different 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 [11] [6, 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) [6, p.743]. Setting the threshold value is done by selecting the value which maximises the between-class variance of the resulting binary image [6, 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 [6, 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 [6, 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 considered 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 [13]. 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 were then again used to get smooth and continuous segmentations. This entire process is shown in Figure 1.1.

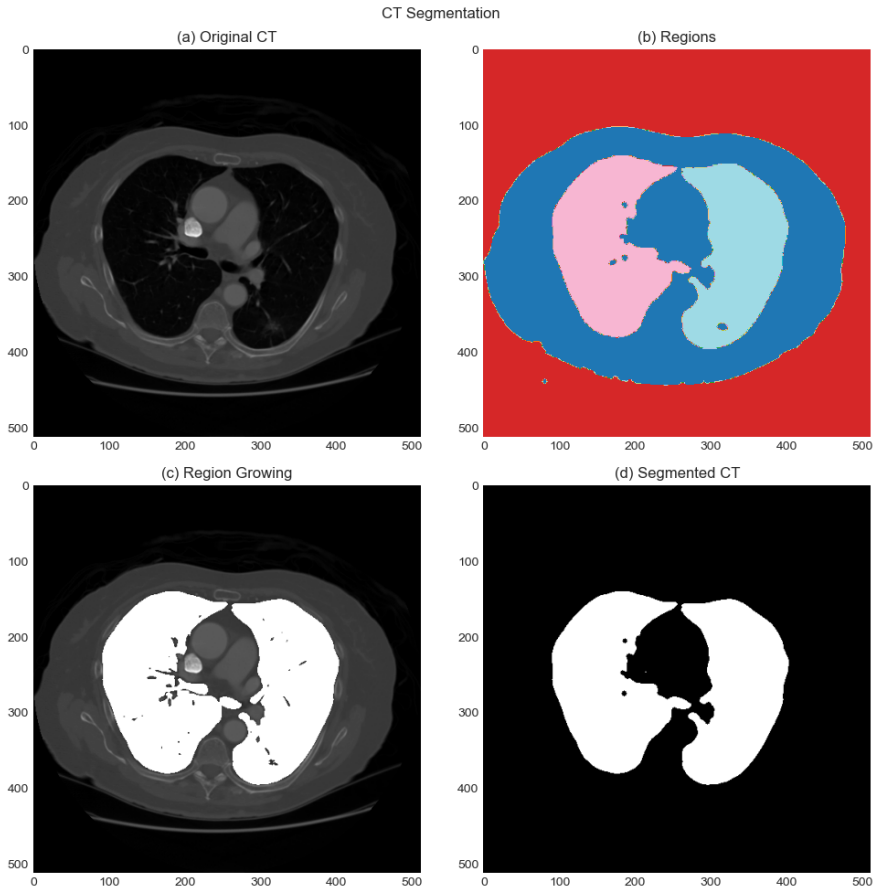


Figure 1.1: The segmentation of the `CT.png` image.

Using a custom implementation

For the final part of this exercise, a custom implementation of the same method is used to segment the CT scanned lungs. Custom code was only written for the Otsu thresholding and the region-growing algorithm, and the results are shown in Figure 1.2.

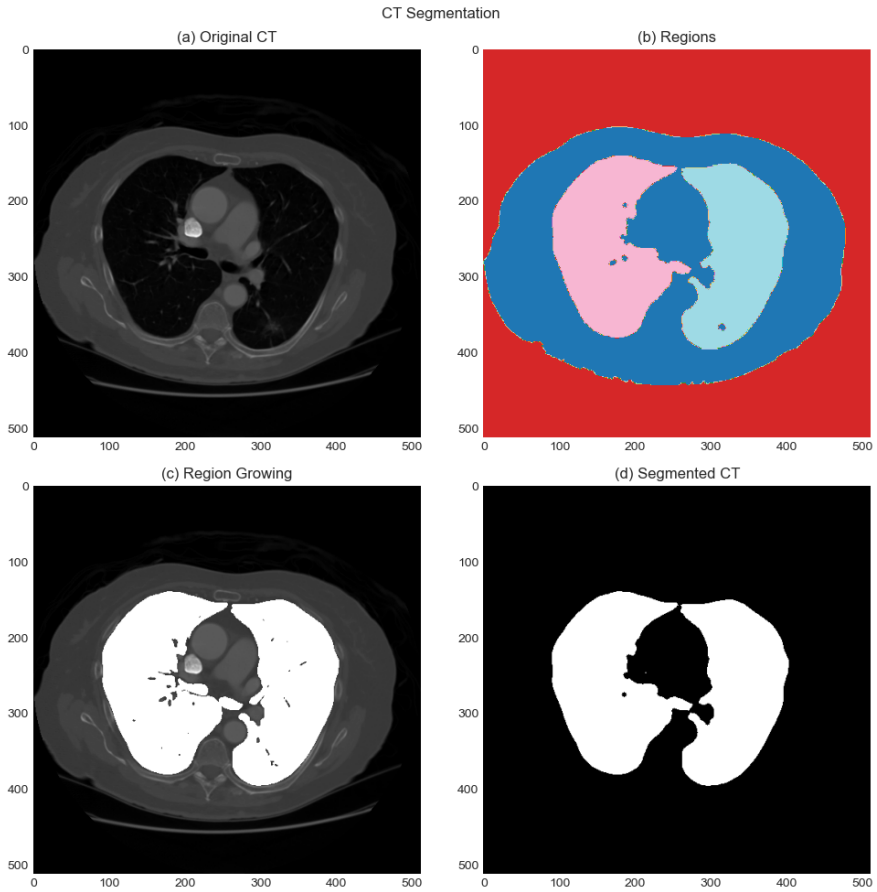


Figure 1.2: The segmentation of the `CT.png` image using a custom implementation.

As can be seen in Figure 1.2, the custom implementation gives very similar results. The Otsu thresholding algorithm was coded following the description of how to find the maximum between-class variance [1]. The region-growing algorithm was also coded following the description of the algorithm [19], however the method was modified to apply a threshold on the normalised intensity difference between the current pixel and the seed pixel, rather than taking the region mean. This was done to keep the algorithm lightweight as the region grows.

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 [20]. Plotting grayscale images of each of those channels (see Figure 1.3), it can be seen that the hue channel is most useful for segmentation.

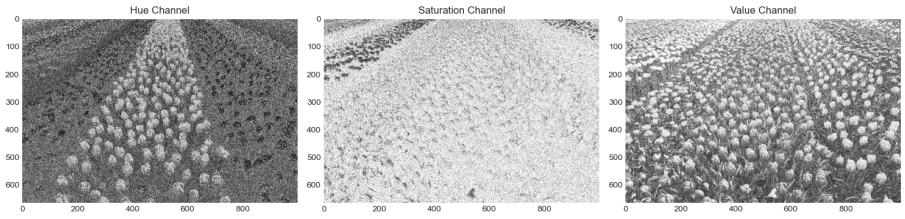


Figure 1.3: The HSV channels of the `noisy_flower.jpg` image.

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 Figure 1.4). Applying 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 [6, pp. 166-170], smoothing out the binary image's noise, and giving back a grayscale image.

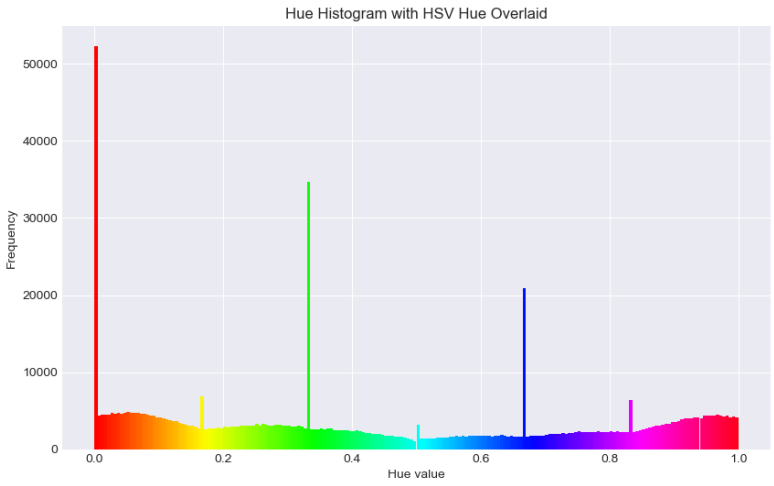


Figure 1.4: The histogram of the hue channel of the `noisy_flower.jpg` image.

Finally, the main algorithm used for this image is the Chan-Vese segmentation algorithm [3]. 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 [16]. 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 1.5.

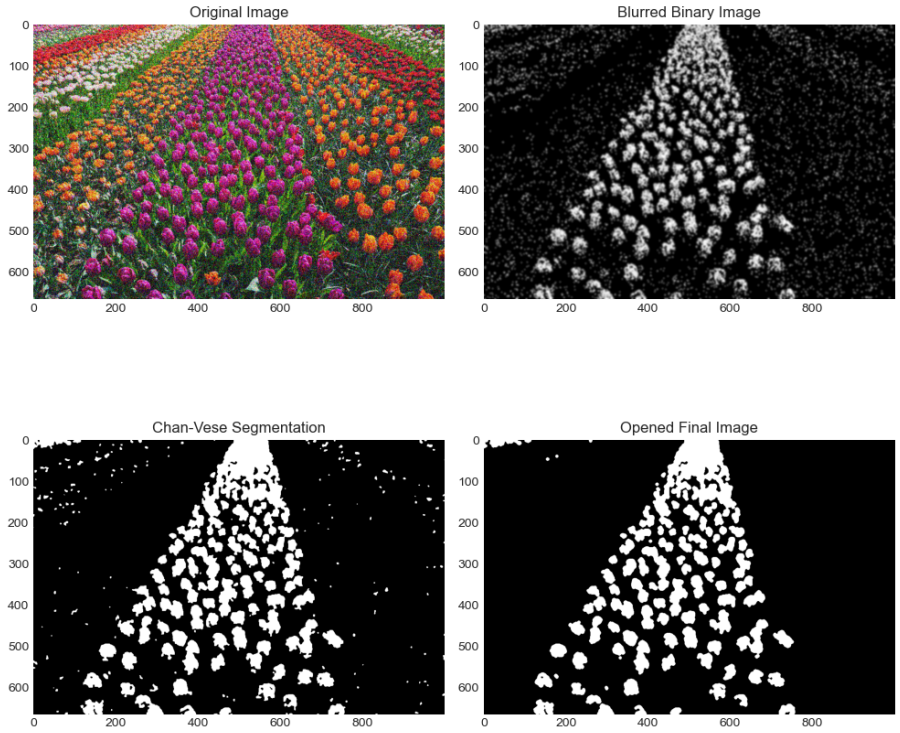


Figure 1.5: The segmentation process of the `noisy_flower.jpg` image.

1.1.3 Image 3: `coins.png`

For this problem, the task was to segment specific coins from an image of antique coins, which has been corrupted with vertical black lines across the image. Specifically, the 1st coin of the 1st row, 2nd coin of the 2nd row, and so on until the 4th row. Here, the main challenge was dealing with the lines and then the lighter background at the top left of the image, which almost matched the lighter shade of the coin.

The segmentation for this problem first made use of an inpainting algorithm [12]. This uses biharmonic functions, constructed using Laplacian and normal derivatives of the boundary data [4]. Once the image is inpainted, the contrast of the image is increased using the scikit-image `exposure.rescale_intensity` function. This function linearly scales the intensity of the image to a certain range, which can be set as the minimum and maximum values of the image [14]. This is done to make the coins more distinguishable from the background.

The second part of the segmentation was to try and remove the background of the image, using a rolling-ball algorithm. This is akin to a filter as it uses kernels to define the shape of the object that is "rolled". For a sphere kernel, one can imagine creating a discretized surface where each square column represents a pixel, and having a ball roll on the underside of that surface. With sufficient radius, that sphere will not "fall up" in the peaks of the image, and the background will be estimated as the height of the top of the ball [15]. With the background estimates, it can be subtracted from the image, making the segmentation of the coins easier.

Third, K-means clustering is used to segment all the coins in the image. By setting the number of clusters to 2, the algorithm initializes two values between 0 and 255. It then assigns each pixel to the cluster with the closest mean value. This process is repeated iteratively until the cluster assignments no longer change, resulting in the image being divided into two regions [17]. The coins are then labelled, and the chosen coins are isolated from those labels. The result of this process is shown for each coin in Figures 1.6, 1.7, 1.8, and 1.9.

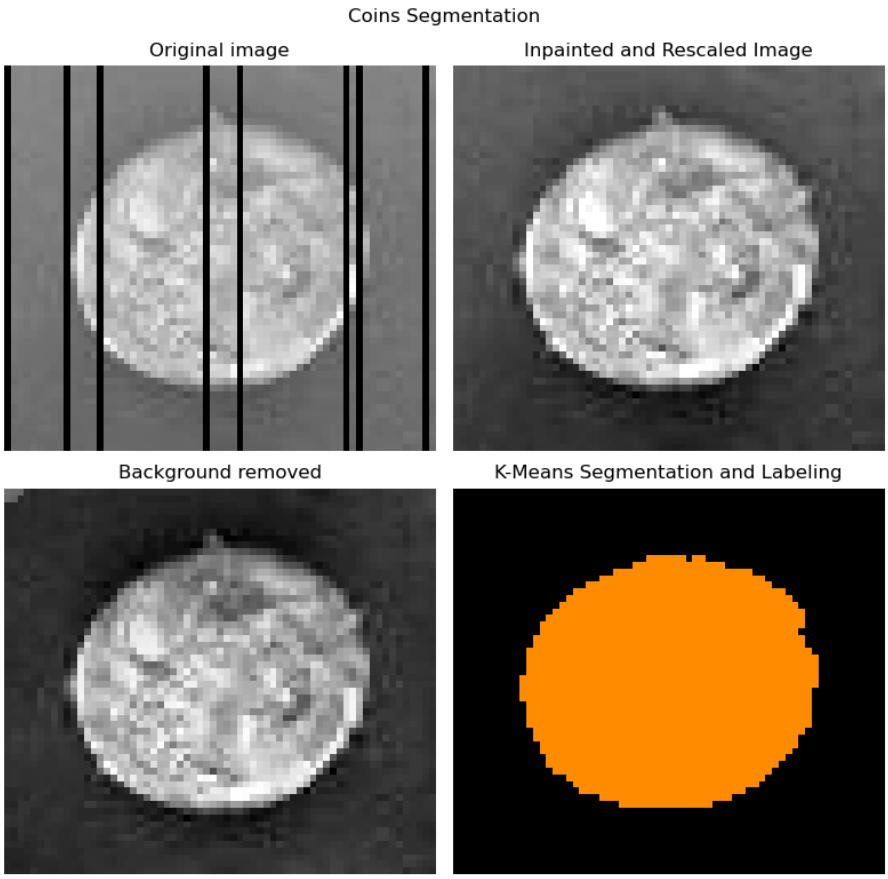


Figure 1.6: The segmentation of the 1st coin in the 1st row.

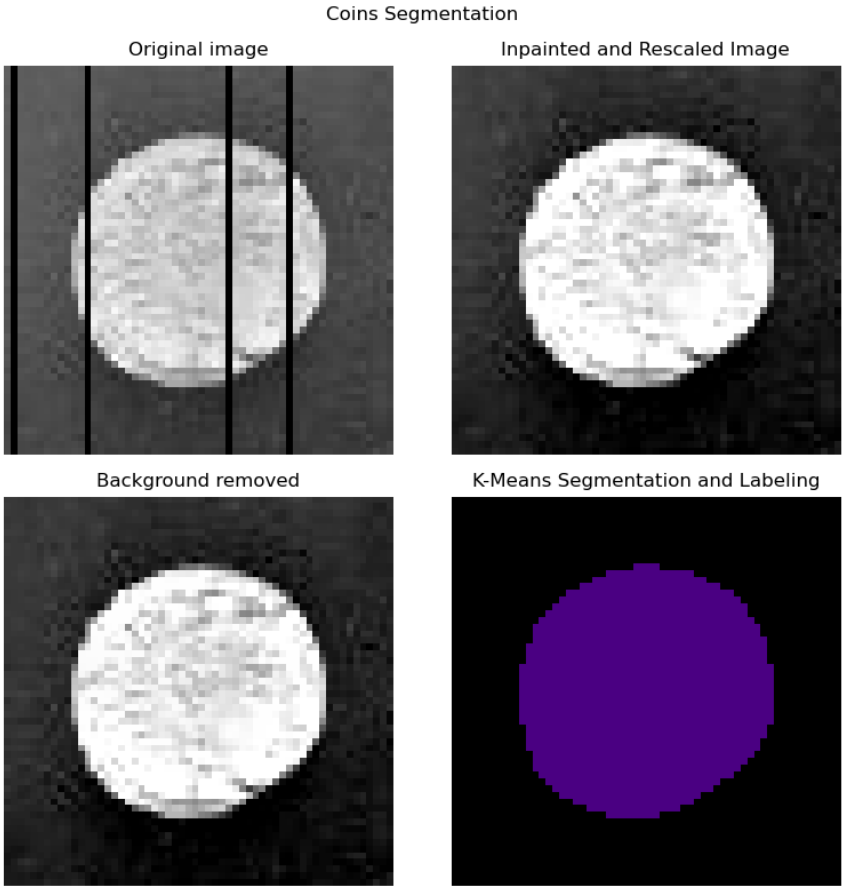


Figure 1.7: The segmentation of the 2nd coin in the 2nd row.

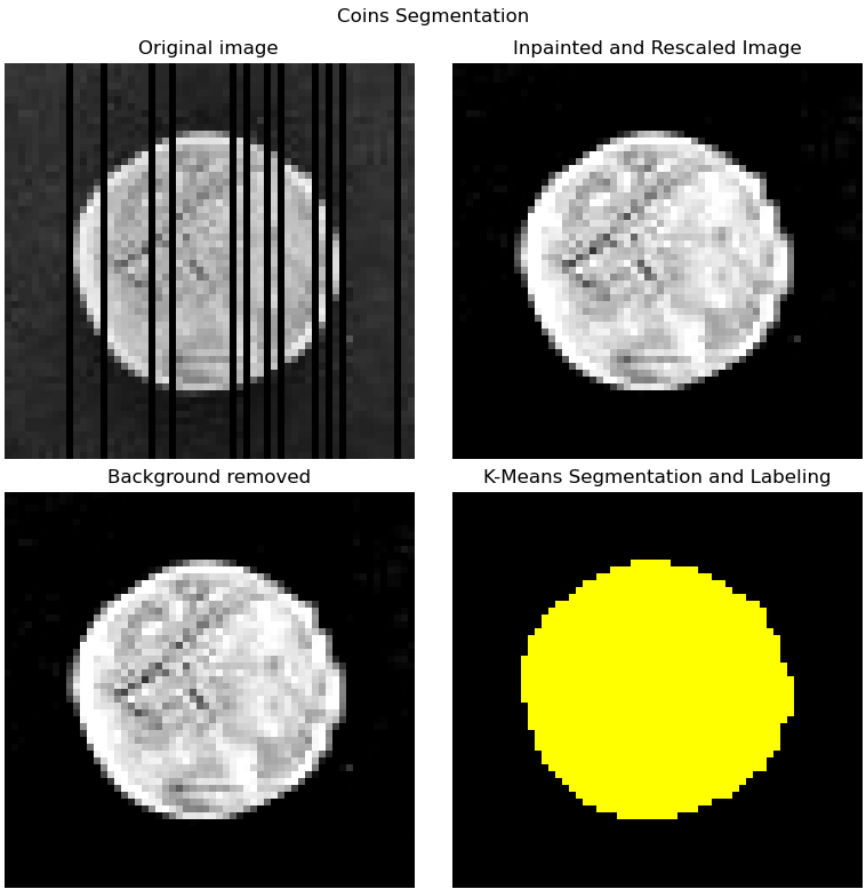


Figure 1.8: The segmentation of the 3rd coin in the 3rd row.

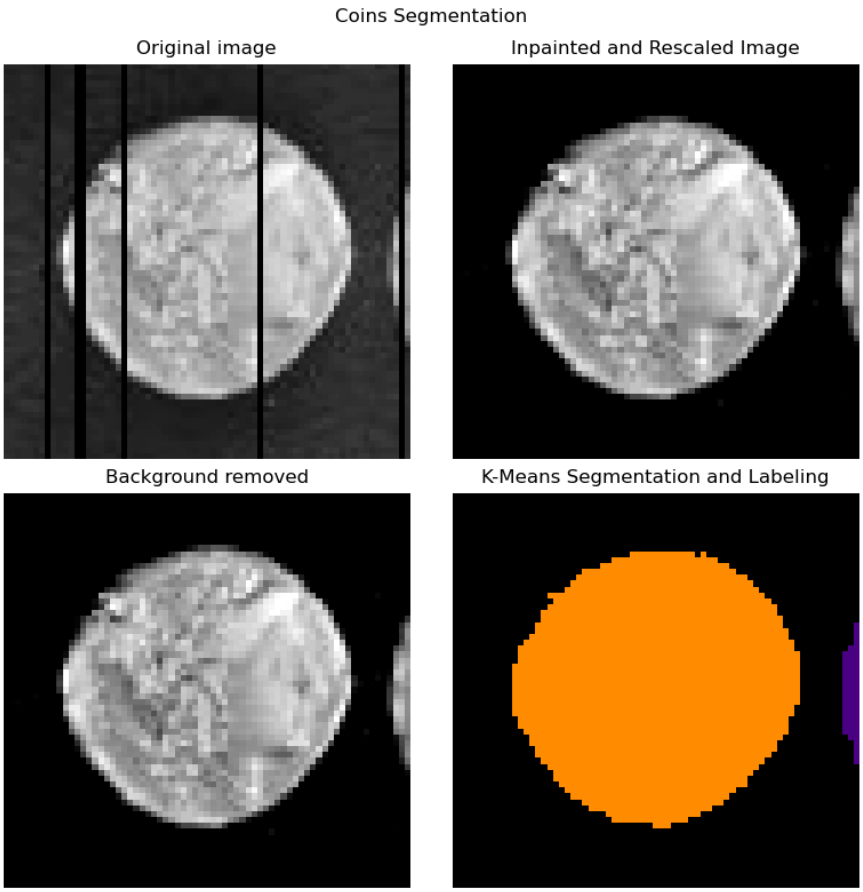


Figure 1.9: The segmentation of the 4th coin in the 4th row.

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.

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 [21], while l_2 minimisation is the minimisation of the sum of the squares of the residuals [8]. Switching to a more explicit notation of this problem, this gives:

$$||(ax+b)-f||_{l_1} = \sum_{i=1}^{n} |(ax_i+b)-f_i| \tag{2.1}$$

$$||(ax+b)-f||_{l_2} = \sqrt{\sum_{i=1}^{n} ((ax_i+b)-f_i)^2} \tag{2.2}$$

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.minimize` 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

	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.

	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.

of quadratic problems that are easier to solve [5]. Using this method, the fitted parameters were obtained and are shown in Table 2.1, and the plots of the fitted lines are shown in Figure 2.1.

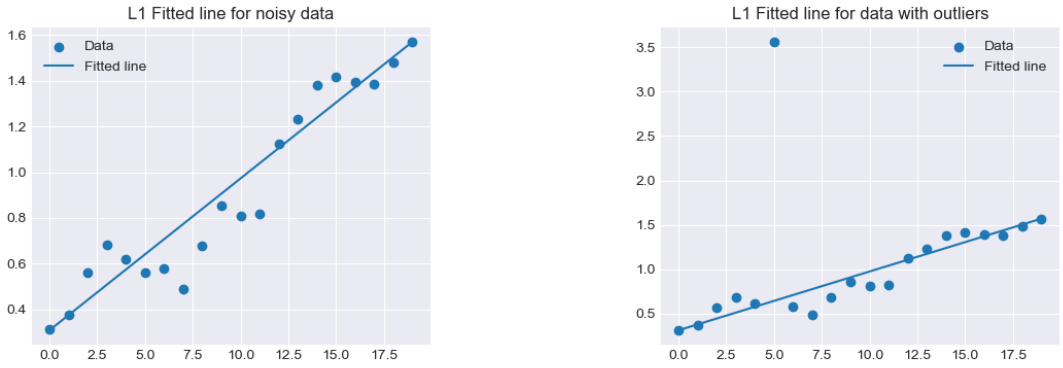


Figure 2.1: Plots of the fitted straight line to the noisy (left), and outlier (right) data using l_1 minimisation, assuming the x-coordinate of the data points is the index of the data point.

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 [18]. 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.

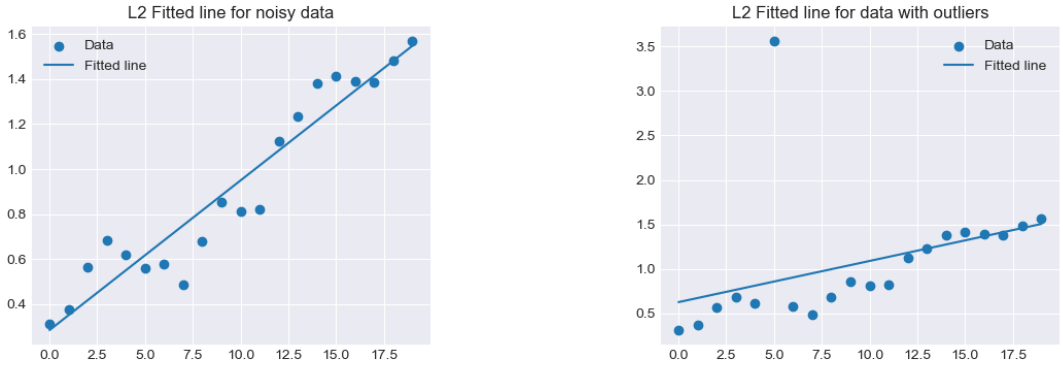


Figure 2.2: Plots of the fitted straight line to the noisy (left), and outlier (right) data using l_2 minimisation, assuming the x-coordinate of the data points is the index of the data point.

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

$$\arg\min\left(\frac{1}{2}||\mathcal{F}_\Omega\hat{x}-y||_2^2+\lambda|\hat{x}|_1\right) \tag{2.3}$$

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

$$\text{SoftThresh}(x,\lambda)=\begin{cases}x-\lambda & \text{if } x>\lambda\\ x+\lambda & \text{if } x<-\lambda\\ 0 & \text{otherwise}\end{cases} \tag{2.4}$$

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

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.

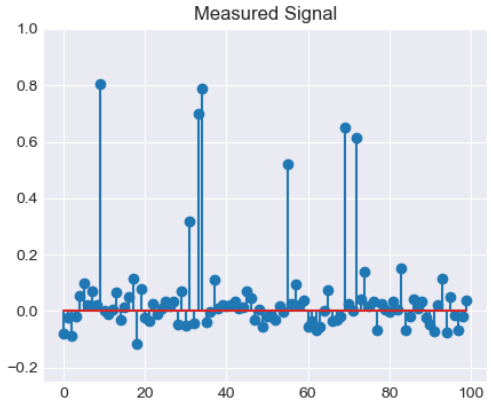


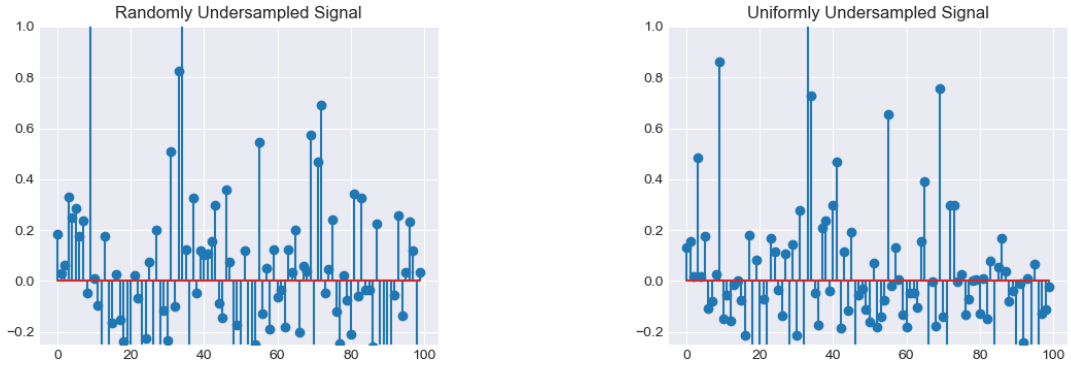
Figure 2.3: Plot of the noisy signal.

To switch from the time to the frequency domain, the fourier transform of the noisy vector is taken. And it 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 using the Iterative Soft Thresholding Algorithm (ISTA).

2.2.2 The Iterative Soft Thresholding Algorithm

In order to reconstruct the original signal, the ISTA is used. This algorithm is an iterative method that solves the minimisation problem described above by iteratively applying a soft thresholding function to the signal, with the addition of a data consistency constraint. The algorithm is shown in Algorithm 1.



(a) Random Undersampling (b) Uniform Undersampling

Figure 2.4: Plots of the undersampled signals.

Algorithm 1: Iterative Soft Thresholding Algorithm

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

Using this method the reconstructions are obtained and shown in Figure 2.5. As can be seen, the random undersampling method gives a better reconstruction of the original signal than the uniform undersampling method. This is due to the random undersampling method being more efficient at capturing the signal's features, as it is more likely to sample the signal's peaks and troughs. This is in contrast to the uniform undersampling method, which may miss these features if the samples are

not taken at the right points.

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 2.6, the image studied is shown.

2.3.1 Daubechies Wavelet Transform

Discrete wavelet transforms make the use of a scaling function adn wavelets to express images as a linear combination of them. The scaling function generates a series of approximations of the image, and the wavelets encode the differences between adjacent approximations. And the decomposition can be made at different resolution levels, with each level encoding more detail of the image [7].

Once computed, the wavelet coefficients can be visualised as an image, as shown in Figures 2.7 and 2.8. As can be seen, each describes details of the image at different scales. Namely, the 2nd level coefficients seem to capture the features of the riverside image more intensely than the 1st level coefficients.

One can then reconstruct it using the inverse wavelet transform, and the image is shown in Figure 2.9.

And comparing this with the original image (2.10), it can be seen that the reconstruction is quite good, with the main features of the image being preserved. 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.

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 figures 2.11 and 2.12.

Figure 2.5: The reconstructions of the original signal using random and uniform undersampling.

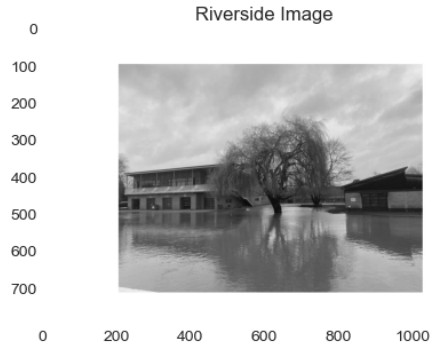


Figure 2.6: The riverside image studied in this exercise.

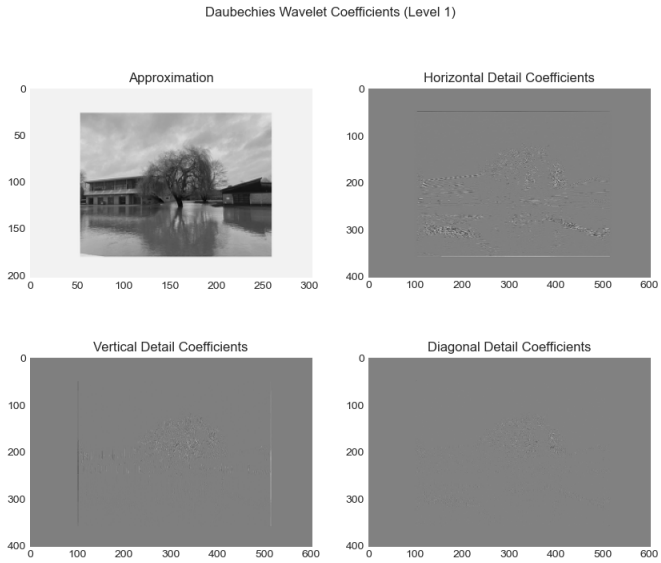


Figure 2.7: The 1st scaling level Daubechies wavelet transform coefficients of the riverside image.

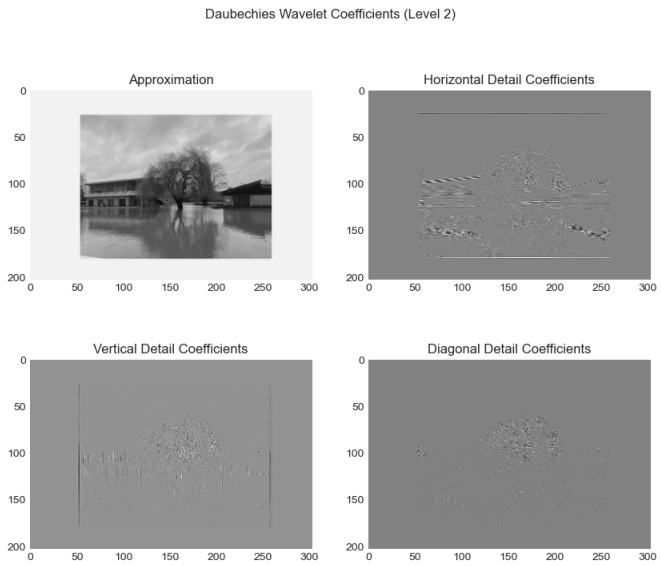


Figure 2.8: The 2nd scaling level Daubechies wavelet transform coefficients of the riverside image.

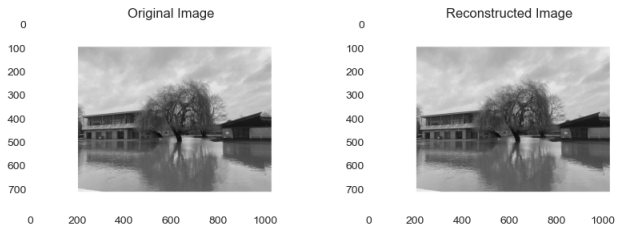


Figure 2.9: The reconstructed riverside image using the Daubechies wavelet transform.

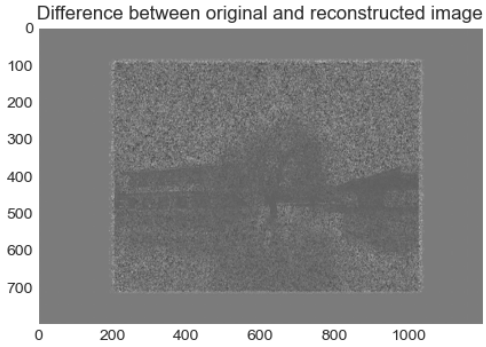


Figure 2.10: The difference between the original and reconstructed riverside images.

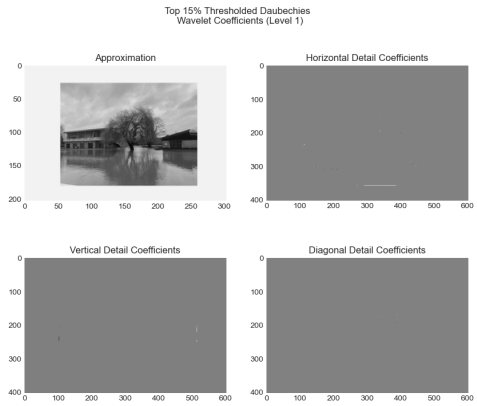


Figure 2.11: The thresholded riverside image 1st level Daubechies wavelet transform coefficients, for a top 15% threshold.

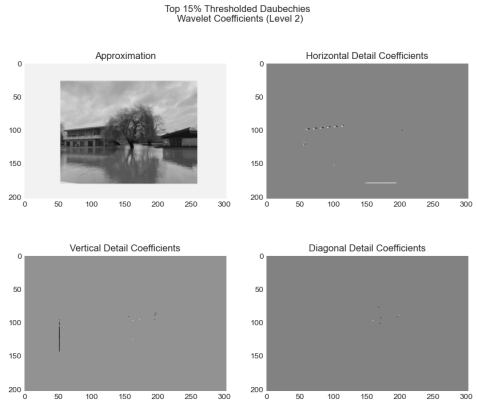


Figure 2.12: The thresholded riverside image 2nd level Daubechies wavelet transform coefficients, for a top 15% threshold.

For the other thresholds, the results are shown in Figures 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, and 4.8, 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 2.13, 2.14.

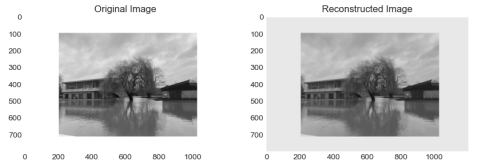


Figure 2.13: The reconstructed riverside image using the thresholded Daubechies wavelet transform coefficients, for a top 15% threshold.

For the other thresholds, the results are shown in Figure 4.9, 4.11, 4.13, and 4.15, in the Appendix. And the differences between the original and reconstructed images are shown in Figure 4.10, 4.12, 4.14, and 4.16.

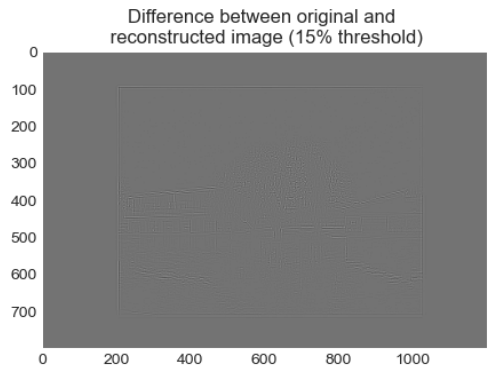


Figure 2.14: The difference between the original and reconstructed riverside images, for a top 15% threshold.

2.3.3 Discussion

As expected, the higher the threshold the less coefficient's information is kept, and for the 1st level decomposition, this reduces to a nearly blank image for thresholds lower than 15%. However, reconstructions stays qualitatively good, and the difference plots show less noise in the reconstructed images the lower the threshold value. This is because the noise can be captured and will be seen in the lower value coefficients, and by thresholding these out, the "noise information" is lost. 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.

Chapter 3

Solving Inverse Problems

In this chapter, the reconstruction of images from noisy data is discussed. And multiple methods are compared, mainly a classical model-driven regularization (Total Variation regularized reconstruction), and a data-driven regularization (Learned gradient descent with a neural network). The image studied is the Shepp-Logan phantom.

3.1 Exercise 3.1

As an entry problem into inverse problems, the task of minimizing the function $f : \mathbb{R}^2 \to \mathbb{R}$ defined as

$$f(x) = \frac{1}{2}[x]_1^2 + [x]_2^2, \quad \text{where} \quad x = \begin{bmatrix}[x]_1 \\ [x]_2\end{bmatrix}, \tag{3.1}$$

is considered. As gradient descent is the chosen method here, the gradient of the function must be computed and is given by: $\nabla f(x) = \begin{bmatrix}[x]_1 \\ 2[x]_2\end{bmatrix}$. In order to use the L-smooth function property given in the coursework paper, it must be shown that the gradient of $f(x)$ is L-Lipschitz continuous. This is done by showing that for all $x, y \in \mathbb{R}^2$, there exists a constant L such that [10]:

$$||\nabla f(x) - \nabla f(y)|| \leq L||x - y|| \tag{3.2}$$

This can be done by computing both norms, of the gradient differences and the vector differences. The norm of the gradient difference is given by:

$$||\nabla f(x) - \nabla f(y)|| = \left|\left|\begin{bmatrix}[x]_1 - [y]_1 \\ 2[x]_2 - 2[y]_2\end{bmatrix}\right|\right| = \sqrt{([x]_1 - [y]_1)^2 + 4([x]_2 - [y]_2)^2} \tag{3.3}$$

Then, for the vector difference norm:

$$||x-y|| = \left|\left|\begin{bmatrix}[x]_1-[y]_1\\ [x]_2-[y]_2\end{bmatrix}\right|\right| = \sqrt{([x]_1-[y]_1)^2+([x]_2-[y]_2)^2} \tag{3.4}$$

And it is clear that the gradient of $f(x)$ is 2-Lipschitz continuous, as one can easily show that:

$$\sqrt{([x]_1-[y]_1)^2+4([x]_2-[y]_2)^2} \leq 2\times\sqrt{([x]_1-[y]_1)^2+([x]_2-[y]_2)^2} \tag{3.5}$$

since the squared terms are positive so multiplying both of them by 4 will result in a larger value than just multiplying one. More formally, the smallest L that satisfies this condition is given by the maximum eigenvalue of the Hessian matrix of $f(x)$ [9, claim 5]:

$$\nabla^2 f(x) = \begin{bmatrix}1 & 0\\ 0 & 2\end{bmatrix} \tag{3.6}$$

And the eigenvalues of this matrix satisfy: $\det(\lambda I - \nabla^2 f(x)) = 0$, which gives $(\lambda-1)(\lambda-2)=0$, giving a maximum eigenvalue of 2. Thus, the gradient of $f(x)$ is L-Smooth with $L=2$.

Thus, the following inequality holds:

$$f(x_K) - f(x^*) \leq \frac{2\|x_0-x^*\|^2}{2K} \tag{3.7}$$

where x^* is the vector X for which $f(x)$ is minimised, x_0 is the initial guess, and x_K is the K-th iteration of the gradient descent algorithm. This inequality can be used to determine the number of iterations needed to reach a certain accuracy, given the initial guess and optimal vector. Now the function $f(x)$ has a clear minimum at $x^* = \begin{bmatrix}0\\0\end{bmatrix}$, and the gradient descent algorithm is run with an initial guess of $x_0 = \begin{bmatrix}1\\1\end{bmatrix}$.

For that case, the value of K needed for the RHS of the inequality to be less than $\epsilon = 0.01$ is given by:

$$0.01 \geq \frac{2\|x_0-x^*\|^2}{2K} \iff K \geq \frac{2\|x_0-x^*\|^2}{2\times 0.02} = \frac{2\times 2}{0.02} = 200 \tag{3.8}$$

However, when applying the gradient descent algorithm, setting the learning rate to 1/L, the algorithm converges to an $\epsilon \leq 0.01$ in 3 iterations, and the evolution of the vector x's components is shown in Figure 3.1. And the final value of ϵ is 0.0078.

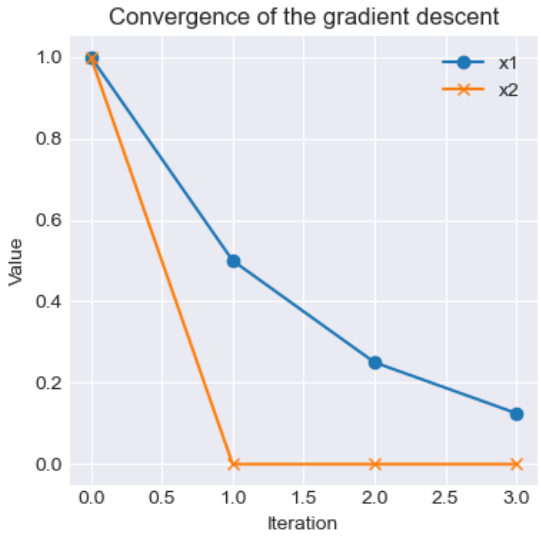


Figure 3.1: The convergence of the gradient descent algorithm for the function $f(x)$.

3.2 Exercise 3.2

In this exercise the 2 methods mentioned at the start of this chapter are compared. The image considered is the Shepp-Logan phantom, and is shown in Figure 3.2, along with the image's sinogram and its filtered back-projection (FBP) reconstruction. In both methods, the FBP reconstruction will be used as initialization.

Figure 3.2: The Shepp-Logan phantom image, its sinogram, and its FBP reconstruction.

The first method implements the ADMM algorithm to solve a Total Variation (TV) regularized reconstruction problem for the Shepp-Logan phantom image. The ADMM algorithm relies on the use of a variable splitting method, decomposing the problem into smaller subproblems, and then alternatingly updating them [2]. The TV-regularized reconstruction problem is given by:

$$\min_u ||y - Ax||_2^2 + \lambda||\nabla x||_1 \tag{3.9}$$

where y is the noisy sinogram of the image to be reconstructed, A is the measurement matrix, x is the image to be reconstructed, λ is the regularization parameter, and ∇x is the gradient of the image. The ADMM algorithm is used to solve this problem and find u that minimises the TV, and the results are shown in Figure 3.3.

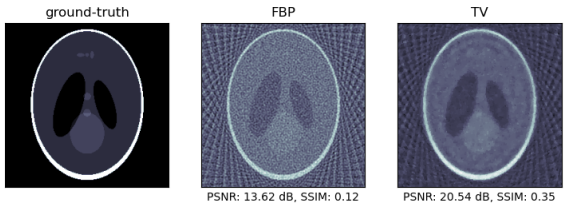


Figure 3.3: The TV-regularized reconstruction of the Shepp-Logan phantom image.

The second method uses a neural network to solve the same problem, with a learned gradient descent (LGD) algorithm. The neural network is based on a shallow 3-layer convolutionnal neural network (CNN) which takes in the current prediction and the gradient of the fidelity term. The predicted reconstruction is then obtained by iteratively (5 iterations) updating the prediction with gradient descent. The parameters that the network learns are related to the learning rate of that gradient descent, hence learned gradient descent. The network is trained on the Shepp-Logan phantom image, and the results are shown in Figure 3.4.

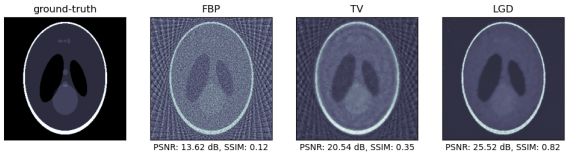


Figure 3.4: The learned gradient descent reconstruction of the Shepp-Logan phantom image.

3.2.1 Discussion

As can be clearly seen, the LGD does a much better job at reducing the FBP's noise.

Chapter 4

Appendix

4.1 README

4.1.1 Image Analysis Coursework

Description

This repository contains the code and written report for the Image Analysis Coursework. The aim was to study 3 aspects of image analysis: Segmentation, Inverse problems and multiresolution analysis, and solving inverse problems using model vs data driven reconstructions

Contents

Inside this `tmb76/` directory, there are a few sub-directories one can explore. There's the code directory (`src/`), which contains all the code used for the project. An important note is that the code will give outputs in the command line but also store the plots in a `Plots/` directory which will be created as the first code file is run. So if there is no `Plots/` directory in the repository yet, running the code once should lead to creating one. Then there is the `Report/` directory, which contains the LaTeX file for the report, as well as the pdf version of it, along with the references `.bib` file. Finally, there is a `data/` directory which holds images and data files for the code. More importantly, there are an `environment.yml` and `Dockerfile` files, which one is advised to use.

How to run the code

For permissions reasons, the `Dockerfile` is not set up to pull the repository directly as it builds the image. Therefore, one must first download this repository to their local machine and then are free to build the Docker image from the `Dockerfile`.

To run the solver on a Docker container, one first has to build the image and run the container. This can be done as follows:

```
$ docker build -t ia_coursework .
$ docker run --rm -ti ia_coursework
```

The `ia_coursework` is not a strict instruction, it can be set to any other name the user may prefer.

If there is a need to get the plots back on the local machine, the second line above can be ran without the `--rm` and can also set the container name using `--name=container_name` (any valid name is fine). From there, run all the code as instructed below. Once all desired outputs and plots have been obtained. One can exit the container and then run:

```
$ docker cp docker cp container_name:/ia_coursework/Plots ./Plots
```

The `Plots/` directory will get copied into the local folder the container was ran from.

As you run this, the Docker image will get built and the container ran, providing the user with a bash terminal-like interface where the solver code can be run as follows:

- For Module 1 on segmentation:

```
$ python src/mod_1_*.py
```

where `*` can be either `coins`, `CT_custom`, `CT`, or `tulips`.

- For Module 2 on Inverse Problems and multiresolution analysis

```
$ python src/mod_2_q_*.py
```

where `*` can be either `1`, `2`, or `3`.

- For Module 3, on solving inverse problems with different methods

```
$ python src/mod_3_*.py
```

where `*` can be either `q_1` or `LGD`. If the `LGD` file is run, a `skip_training` argument must be specified in the command line: a boolean (True or False). If set to True, the code will load the trained model state and predict the reconstruction immediately, rather than training the neural network from scratch. Each contain the code to get the results those specific tasks/parts of the coursework.

Note on time: Running the file should take close to 2 minutes. Running the file should take longer, closer to 5-6 minutes. This is based on running all of these on a MacBook Air M2 (2022, Ventura 13.2.1), with 8 GB of Memory, so this may be slower on a container.

Further development

If one wishes to further develop this code, such as adding more algorithms to try, when the image is built, git is installed and initialized and the pre-commit hooks are

installed.

Use of Generative AI

GitHub Copilot's autocompletion feature was used in coding the project, when writing docstrings for the functions, though sometimes adding elements ourselves, and for repetitive parts of the code. ChatGPT was also used to help in debugging the code, by providing the traceback as a prompt when an error was difficult to understand, asking to explain what the error refers to. Additionally, it was used to give suggestions on what package/function could be used for l1 minimization for which it recommended the `scipy.optimize` library. It was used for clarification on the `np.random.permutation` library. It was used when debugging issues with loading dependencies, by looking at the coursework given notebook and the commands used to install dependencies on the virtual environment to see differences and what could be the reason: got suggestion to look into what channels were used to load the dependencies.

4.2 Extra Plots

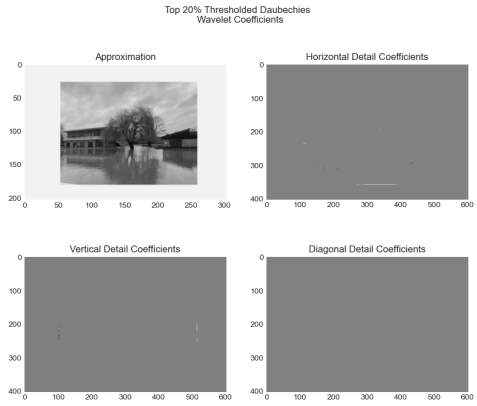


Figure 4.1: The thresholded riverside image 1st level Daubechies wavelet transform coefficients, for a top 20% threshold.

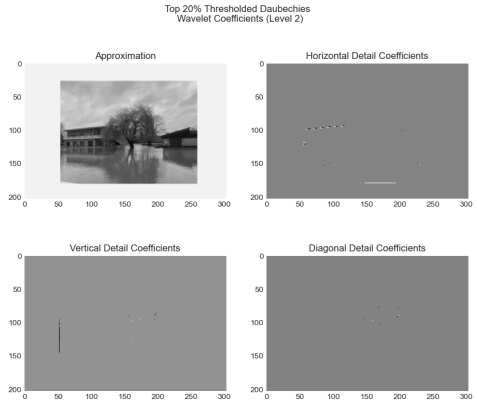


Figure 4.2: The thresholded riverside image 2nd level Daubechies wavelet transform coefficients, for a top 20% threshold.

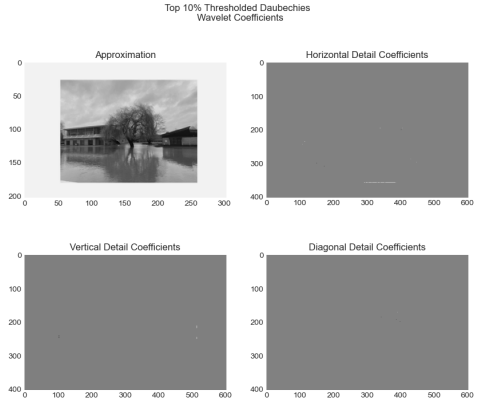


Figure 4.3: The thresholded riverside image 1st Daubechies wavelet transform coefficients, for a top 10% threshold.

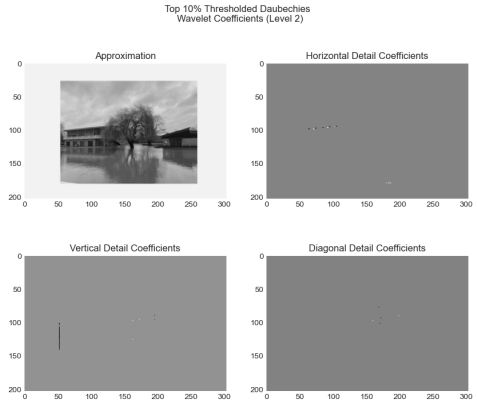


Figure 4.4: The thresholded riverside image 2nd Daubechies wavelet transform coefficients, for a top 10% threshold.

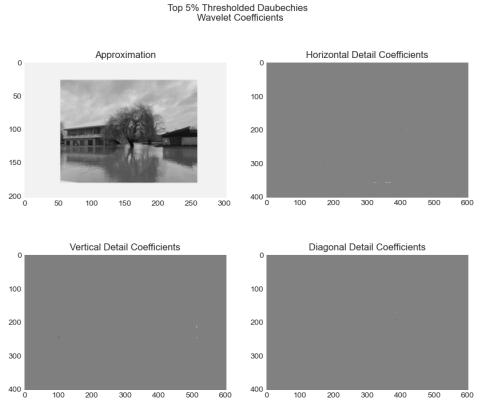


Figure 4.5: The thresholded riverside image 1st level Daubechies wavelet transform coefficients, for a top 5% threshold.

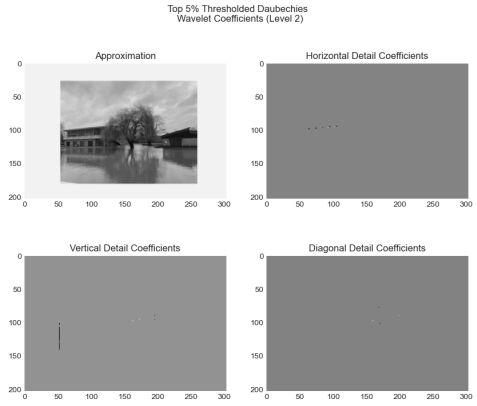


Figure 4.6: The thresholded riverside image 2nd level Daubechies wavelet transform coefficients, for a top 5% threshold.

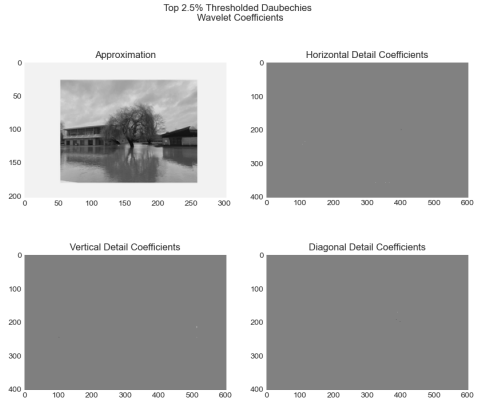


Figure 4.7: The thresholded riverside image 1st level Daubechies wavelet transform coefficients, for a top 2.5% threshold.

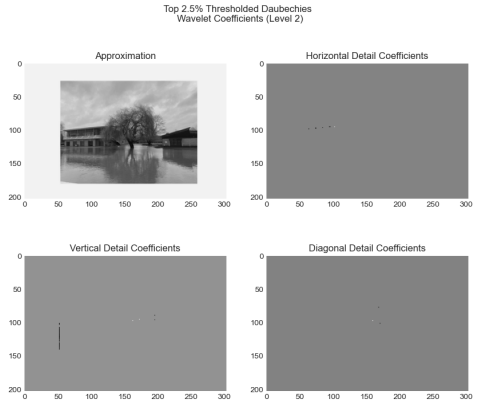


Figure 4.8: The thresholded riverside image 2nd level Daubechies wavelet transform coefficients, for a top 2.5% threshold.

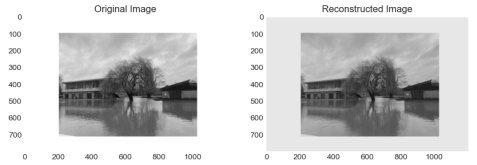


Figure 4.9: The reconstructed riverside image using the thresholded Daubechies wavelet transform coefficients, for a top 20% threshold.

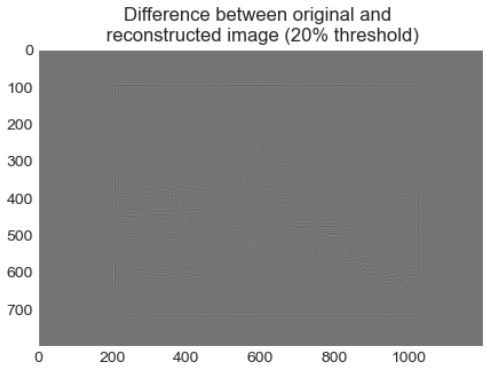


Figure 4.10: The difference between the original and reconstructed riverside images, for a top 20% threshold.

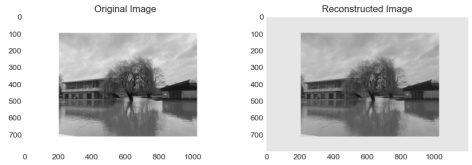


Figure 4.11: The reconstructed riverside image using the thresholded Daubechies wavelet transform coefficients, for a top 10% threshold.

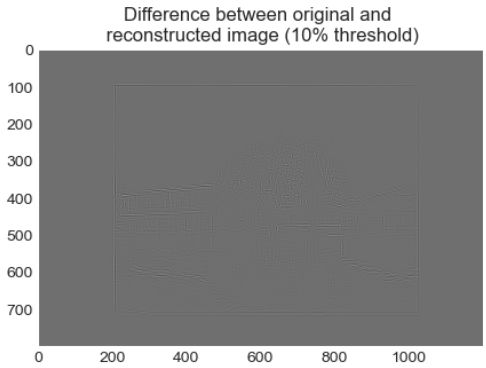


Figure 4.12: The difference between the original and reconstructed riverside images, for a top 10% threshold.

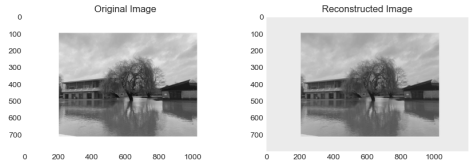


Figure 4.13: The reconstructed riverside image using the thresholded Daubechies wavelet transform coefficients, for a top 5% threshold.

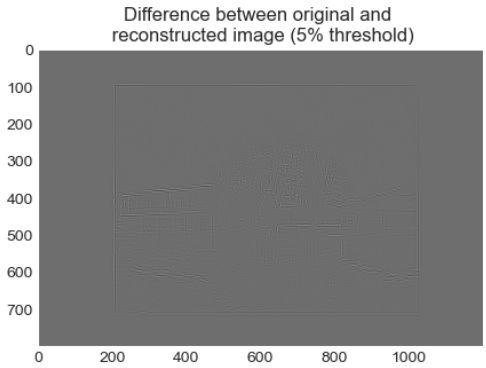


Figure 4.14: The difference between the original and reconstructed riverside images, for a top 5% threshold.

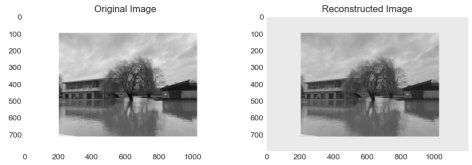


Figure 4.15: The reconstructed riverside image using the thresholded Daubechies wavelet transform coefficients, for a top 2.5% threshold.

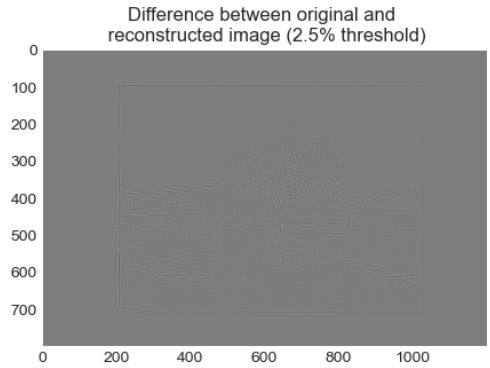


Figure 4.16: The difference between the original and reconstructed riverside images, for a top 2.5% threshold.

Bibliography

[1] Baeldung. Otsu's thresholding for image segmentation. *Baeldung*, 2021.

[2] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

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

[4] S. B. Damelin and N. S. Hoang. On surface completion and image inpainting by biharmonic functions: Numerical aspects. *International Journal of Mathematics and Mathematical Sciences*, 2018:3950312, 2018.

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

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

[7] S.G. Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, 1989.

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

[9] Lecture Notes. Lecture 12 - nonlinear optimization. *Linköping University*, 2021.

[10] Lecture Notes. Optimization methods. *University of Cambridge*, 2021.

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

[12] Scikit-image. Inpaint biharmonic. *Scikit-image Documentation*, 2021.

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

[14] Scikit-image. Rescale intensity. *Scikit-image Documentation*, 2021.

[15] Scikit-image. Rolling ball background subtraction. *Scikit-image Documentation*, 2021.

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

[17] Scikit-learn. K-means clustering. *Scikit-learn Documentation*, 2021.

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

[19] SBME Tutorials. Computer vision - week 6. *SBME Tutorials*, 2019.

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

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