Deep-Learning Regularisation Parameter-Map with U-Net for Total Variation Image Denoising on Chest X-ray Images

Thanh Trung Vu - 230849442

Contents

1 Introduction **2**

1.1 Related Work 2

1.2 Contribution 2

2 Mathematical Formulation **2**

2.1 Total Variation Denoising 2

2.2 Primal Dual Method 6

3 Neural Network Architecture **8**

3.1 Convolutional Neural Network (CNN) 8

3.2 U-Net 9

3.3 Full Architecture 10

4 Chest X-ray Dataset **11**

4.1 Experimental Set-Up 11

4.2 Training Data 11

4.3 Results 15

5 Turtle ID Dataset **19**

5.1 Results 21

6 Important Implementation Details **21**

6.1 Metrics 21

7 Conclusion **22**

8 Future Work **22**

1 Introduction

This project is largely based on existing research on the learning of regularisation parameter maps for variational image reconstruction using deep neural networks and algorithm unrolling Kofler et al., 2023. In this dissertation, we focus on the problem of image denoising. We define solving the denoising problem as solving the total variation (TV) problem and give a mathematical formulation of variational image denoising. We then discuss the primal dual algorithm for solving the TV denoising problem, and the importance of choosing the regularisation parameter. This led to the idea of using a deep learning method called U-Net to help us find a regularisation parameter map which can improve the denoising performance of the primal dual algorithm. We combine a U-Net architecture with the primal dual algorithm to create an end-to-end unsupervised model for image denoising that can achieve said improvement while staying completely interpretable. We implement the proposed combined model and evaluate it on the Chest X-ray dataset. The results show that the combined model consistently outperforms the traditional method of using a single regularisation parameter.

1.1 Related Work

1.2 Contribution

- Adapt the existing code of the dynamic image denoising project in Kofler et al., 2023 to static image denoising. Train and evaluate the model using a new dataset - Chest X-ray dataset.

- Present the theory of the total variation denoising method and the experimental set-up in detail so that a fellow student can understand the method and implement it themselves.

2 Mathematical Formulation

2.1 Total Variation Denoising

Given an object to be imaged $\mathbf{x}_{\text{true}} \in \mathbb{R}^{m \times n}$, we can represent a noisy image $\mathbf{z} \in \mathbb{R}^{m \times n}$ as:

$$\mathbf{z} = \mathbf{x}_{\text{true}} + \mathbf{e} \tag{1}$$

where $\mathbf{e} \in \mathbb{R}^{m \times n}$ denotes some random noise component. Our goal is to find $\mathbf{x}_{\text{denoised}} \in \mathbb{R}^{m \times n}$ which represents a good reconstruction of the true image.

A common approach is to formulate the reconstruction as a minimisation problem:

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} d(\mathbf{x}, \mathbf{z}) + \mathcal{R}_\lambda(\mathbf{x}) \tag{2}$$

where $d(\cdot, \cdot)$ is a data discrepancy term which represents the closeness between the given noisy image and the reconstruction, and $\mathcal{R}_\lambda(\cdot)$ is a regularisation term. The idea is that a small discrepancy value indicates that important details of the image are preserved, while a small regularisation value indicates that the impact of noise on smooth regions of the image is removed.

Here we will model the noise as Gaussian noise, for which a good discrepancy measure is the ℓ_2 norm

$$d(\mathbf{x}, \mathbf{z}) = \frac{1}{2}\|\mathbf{x} - \mathbf{z}\|_2^2 \tag{3}$$

For our variational denoising problem, the regularisation term $\mathcal{R}_\lambda(\cdot)$ is the total variation of the image, which can be expressed as a scaled ℓ_1 norm

$$\mathcal{R}_\lambda(\mathbf{x}) = \lambda\|\nabla\mathbf{x}\|_1 \tag{4}$$

where $\nabla\mathbf{x}$ is the gradient of the image $\mathbf{x}$ and $\lambda \in \mathbb{R}^+$ is a regularisation parameter.

The discrepancy term and the regularisation term are often in conflict with each other. Sorely minimising the discrepancy term will keep all the noise intact, producing no denoising effect. On the other hand, minimising the regularisation term alone often results in a blurry image and loss of all important details. The regularisation parameter λ controls the trade-off between the two. Therefore, choosing the right value of the regularisation parameter is crucial for the success of the denoising process.

We will soon look at an alternative regularisation term formulation $\mathcal{R}_{\mathbf{\Lambda}}(\mathbf{x})$ which uses a set of regularisation parameters instead of a single scalar value λ, providing more flexibility in the denoising process. Before that, we first define the image gradient operator ∇, also known as the finite-difference operator. Finite difference operators include forward difference operator, backward difference operator, shift operator, central difference operator and mean operator. Let us focus on the forward difference operator. In particular, in the discrete case, given a matrix $\mathbf{x}$, $\nabla\mathbf{x}$ consists of two matrices, one for the vertical gradient $\nabla_x\mathbf{x}$ and one for the horizontal gradient $\nabla_y\mathbf{x}$.

$$\nabla\mathbf{x} = \begin{bmatrix} \nabla_x\mathbf{x} \\ \nabla_y\mathbf{x} \end{bmatrix} \tag{5}$$

For the forward gradient, the vertical gradient is calculated as

$$\begin{aligned}\nabla_x \mathbf{x} &= \begin{bmatrix} \mathbf{x}_{2,1} - \mathbf{x}_{1,1} & \mathbf{x}_{2,2} - \mathbf{x}_{1,2} & \dots & \mathbf{x}_{2,n-1} - \mathbf{x}_{1,n-1} & \mathbf{x}_{2,n} - \mathbf{x}_{1,n} \\ \mathbf{x}_{3,1} - \mathbf{x}_{2,1} & \mathbf{x}_{3,2} - \mathbf{x}_{2,2} & \dots & \mathbf{x}_{3,n-1} - \mathbf{x}_{2,n-1} & \mathbf{x}_{3,n} - \mathbf{x}_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{x}_{m,1} - \mathbf{x}_{m-1,1} & \mathbf{x}_{m,2} - \mathbf{x}_{m-1,2} & \dots & \mathbf{x}_{m,n-1} - \mathbf{x}_{m-1,n-1} & \mathbf{x}_{m,n} - \mathbf{x}_{m-1,n} \\ \mathbf{x}_{1,1} - \mathbf{x}_{m,1} & \mathbf{x}_{1,2} - \mathbf{x}_{m,2} & \dots & \mathbf{x}_{1,n-1} - \mathbf{x}_{m,n-1} & \mathbf{x}_{1,n} - \mathbf{x}_{m,n} \end{bmatrix} \\ &= \begin{bmatrix} -1 & 1 & \dots & 0 & 0 \\ 0 & -1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & -1 & 1 \\ 1 & 0 & \dots & 0 & -1 \end{bmatrix} \mathbf{x} \end{aligned} \tag{6}$$

and the horizontal gradient is calculated as

$$\begin{aligned}\nabla_y \mathbf{x} &= \begin{bmatrix} \mathbf{x}_{1,2} - \mathbf{x}_{1,1} & \mathbf{x}_{1,3} - \mathbf{x}_{1,2} & \dots & \mathbf{x}_{1,n} - \mathbf{x}_{1,n-1} & \mathbf{x}_{1,1} - \mathbf{x}_{1,n} \\ \mathbf{x}_{2,2} - \mathbf{x}_{2,1} & \mathbf{x}_{2,3} - \mathbf{x}_{2,2} & \dots & \mathbf{x}_{2,n} - \mathbf{x}_{2,n-1} & \mathbf{x}_{2,1} - \mathbf{x}_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{x}_{m-1,2} - \mathbf{x}_{m-1,1} & \mathbf{x}_{m-1,3} - \mathbf{x}_{m-1,2} & \dots & \mathbf{x}_{m-1,n} - \mathbf{x}_{m-1,n-1} & \mathbf{x}_{m-1,1} - \mathbf{x}_{m-1,n} \\ \mathbf{x}_{m,2} - \mathbf{x}_{m,1} & \mathbf{x}_{m,3} - \mathbf{x}_{m,2} & \dots & \mathbf{x}_{m,n} - \mathbf{x}_{m,n-1} & \mathbf{x}_{m,1} - \mathbf{x}_{m,n} \end{bmatrix} \\ &= \mathbf{x} \begin{bmatrix} -1 & 0 & \dots & 0 & 1 \\ 1 & -1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & -1 & 0 \\ 0 & 0 & \dots & 1 & -1 \end{bmatrix} \end{aligned} \tag{7}$$

The ℓ_1 norm of the image gradient $\nabla \mathbf{x}$ is the sum of the ℓ_1 norms of the two gradient components

$$\|\nabla \mathbf{x}\|_1 = \|\nabla_x \mathbf{x}\|_1 + \|\nabla_y \mathbf{x}\|_1 \tag{8}$$

where

$$\|\nabla_x \mathbf{x}\|_1 = \left(\sum_{i=1}^{m-1} \sum_{j=1}^{n} |\mathbf{x}_{i+1,j} - \mathbf{x}_{i,j}| \right) + \sum_{j=1}^{n} |\mathbf{x}_{1,j} - \mathbf{x}_{m,j}| \tag{9}$$

and

$$\|\nabla_y \mathbf{x}\|_1 = \left(\sum_{i=1}^{m} \sum_{j=1}^{n-1} |\mathbf{x}_{i,j+1} - \mathbf{x}_{i,j}| \right) + \sum_{i=1}^{m} |\mathbf{x}_{i,1} - \mathbf{x}_{i,n}| \tag{10}$$

where $\mathbf{x}_{i,j}$ is the intensity value of pixel (i, j) in the image $\mathbf{x}$.

Returning to our current formulation of the regularisation term $\mathcal{R}_\lambda(\mathbf{x})$, we can rearrange the regularisation parameter λ as follows

$$\mathcal{R}_\lambda(\mathbf{x}) = \lambda\|\nabla\mathbf{x}\|_1 = \|\lambda\nabla\mathbf{x}\|_1 = \|\lambda\nabla_x\mathbf{x}\|_1 + \|\lambda\nabla_y\mathbf{x}\|_1 \tag{11}$$

Since the gradient is multiplied by the regularisation parameter, high λ values lead to reconstructions that lower the discrepancy value, which may lead to over-smoothing areas where there are important details. Conversely, low values of λ may lead to reconstructions that preserve important details but do not remove noise effectively in smooth areas. Using a single λ value affects all regions of the image equally. A more flexible approach is to apply stronger regularisation to smooth regions and weak regularisation to detailed regions to preserve image details. We can do this by swapping the scalar λ with a regularisation parameter map $\mathbf{\Lambda}$,

$$\mathbf{\Lambda} = \begin{bmatrix} \mathbf{\Lambda}^{(x)} \\ \mathbf{\Lambda}^{(y)} \end{bmatrix} \tag{12}$$

where $\mathbf{\Lambda}^{(x)}, \mathbf{\Lambda}^{(y)} \in \mathbb{R}^{m\times n}$ are matrices that contain the regularisation parameters for the vertical and horizontal gradients respectively. Here we will assume $\mathbf{\Lambda}^{(x)} = \mathbf{\Lambda}^{(y)}$ and denote them as $\mathbf{\Lambda}^{(xy)}$, i.e.

$$\mathbf{\Lambda} = \begin{bmatrix} \mathbf{\Lambda}^{(xy)} \\ \mathbf{\Lambda}^{(xy)} \end{bmatrix} \tag{13}$$

This gives us a new regularisation term that can be written as

$$\mathcal{R}_{\mathbf{\Lambda}}(\mathbf{x}) = \|\mathbf{\Lambda} \circ \nabla\mathbf{x}\|_1 = \|\mathbf{\Lambda}^{(xy)} \circ \nabla_x\mathbf{x}\|_1 + \|\mathbf{\Lambda}^{(xy)} \circ \nabla_y\mathbf{x}\|_1 \tag{14}$$

where $\circ$ denotes the Hadamard or element-wise product.

Putting it all together, the solution to the total variational denoising problem can be defined using the following formula:

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} \frac{1}{2}\|\mathbf{x} - \mathbf{z}\|_2^2 + \lambda\|\nabla\mathbf{x}\|_1 \tag{15}$$

if we use a scalar λ regularisation parameter, or

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} \frac{1}{2}\|\mathbf{x} - \mathbf{z}\|_2^2 + \|\mathbf{\Lambda} \circ \nabla\mathbf{x}\|_1 \tag{16}$$

if we use a regularisation parameter map $\mathbf{\Lambda}$ of regularisation parameters.

2.2 Primal Dual Method

To solve equation 16, we can use an iterative method called the Primal Dual Hybrid Gradient (PDHG) algorithm:

Algorithm 1 PDHG algorithm for image denoising with fixed regularisation parameter-map $\mathbf{\Lambda}$ (adapted from Kofler et al., 2023 using the implementation Shote, 2024)

Input: $L = \|[\mathbf{I}, \nabla]^{\mathrm{T}}\|_2 = 3$, $\tau = \text{sigmoid}(10)/L$, $\sigma = \text{sigmoid}(10)/L$, $\theta = \text{sigmoid}(10)$, noisy image $\mathbf{x}_0$
Output: reconstructed image $\hat{\mathbf{x}}$
$\bar{\mathbf{x}}_0 = \mathbf{x}_0$
$\mathbf{p}_0 = \mathbf{x}_0$
$\mathbf{q}_0 = \mathbf{0}$
for $k < T$ **do**
    $\mathbf{p}_{k+1} = (\mathbf{p}_k + \sigma(\bar{\mathbf{x}}_k - \mathbf{x}_0))\,/(1+\sigma)$
    $\mathbf{q}_{k+1} = \text{clip}_{\mathbf{\Lambda}}\,(\mathbf{q}_k + \sigma\nabla\bar{\mathbf{x}}_k)$
    $\mathbf{x}_{k+1} = \mathbf{x}_k - \tau\mathbf{p}_{k+1} - \tau\nabla^{\mathrm{T}}\mathbf{q}_{k+1}$
    $\bar{\mathbf{x}}_{k+1} = \mathbf{x}_{k+1} + \theta(\mathbf{x}_{k+1} - \mathbf{x}_k)$
end for
$\hat{\mathbf{x}} = \mathbf{x}_T$

where $\text{sigmoid}(y) = 1/(1+\exp(-y))$ is the sigmoid function, $\mathbf{I}$ is the identity matrix, $\text{clip}_{\mathbf{\Lambda}}$ is a function that clips the values of $\mathbf{q}_{k+1}$ to the range $[\mathbf{0}, \mathbf{\Lambda}]$.

It has been proven that, if $L^2\sigma\tau < 1$ then the PDHG algorithm converges to the solution of the primal dual problem Chambolle and Pock, 2011. However, in practice, the algorithm always converges even if $L^2\sigma\tau = 1$ Sidky et al., 2012. Nevertheless, here $\text{sigmoid}(10) = 0.9999546$ is used to scale the parameters to satisfy the condition $L^2\sigma\tau < 1$ as well as $\theta < 1$, ensuring convergence. Since σ and τ are the step sizes, choosing them to be as large as possible will help speed up the convergence of the algorithm.

We can calculate the value $L = \|[\mathbf{I}, \nabla]^{\mathrm{T}}\|_2 = 3$ using the formula

$$\|\mathbf{M}\|_2 = \sigma_{\max}(\mathbf{M})$$

where $\sigma_{\max}(\mathbf{M})$ is the largest singular value of matrix $\mathbf{M}$.

For a square 2D matrix with each side of length $2n$, the largest singular value is 3.

This algorithm can also be applied to the single scalar λ case by simply setting all values of $\mathbf{\Lambda}$ to λ.

The number of iterations T is a hyperparameter that we can choose for the training process. In our case we fix $T_{\text{train}} = 128$ for training. As shown in Kofler et al., 2023, the

higher the number of iterations, the better the denoising performance of the algorithm. However, beyond a certain point, the performance improvement is marginal.

(TODO: Understand what L is. Do we need to include $\theta = 1$? Can we write $\bar{\mathbf{x}}_{k+1} = 2\mathbf{x}_{k+1} - \mathbf{x}_k$? In the code, initially $\mathbf{p}_0 = \mathbf{x}_0$?)

To derive the corresponding dual problem of the given nonlinear primal problem:

$$\min_{x \in X} F(Kx) + G(x), \tag{17}$$

we can use the convex optimization framework, particularly the Fenchel duality approach. Here's the process to derive the dual problem:

Primal Problem:

$$\min_{x \in X} F(Kx) + G(x) \tag{18}$$

Step-by-Step Dual Derivation

1. **Introduce a new variable** y to replace Kx:

$$\min_{x \in X, y = Kx} F(y) + G(x) \tag{19}$$

2. **Write the Lagrangian**:

$$\mathcal{L}(x, y, \lambda) = F(y) + G(x) + \lambda^T (y - Kx) \tag{20}$$

   where λ is the Lagrange multiplier (dual variable).

3. **Form the Dual Function** by minimizing the Lagrangian over x and y:

$$g(\lambda) = \inf_{x \in X, y} \mathcal{L}(x, y, \lambda) \tag{21}$$

4. **Separate the minimization** over y and x:

$$g(\lambda) = \inf_{y} \left[F(y) + \lambda^T y \right] + \inf_{x \in X} \left[G(x) - \lambda^T K x \right] \tag{22}$$

5. **Identify the Fenchel conjugates** of F and G:

$$F^*(\lambda) = \sup_{y} \left[\lambda^T y - F(y) \right] \tag{23}$$

$$G^*(\lambda) = \sup_{x \in X} \left[\lambda^T K x - G(x) \right] \tag{24}$$

6. **Express the dual function** using these conjugates:

$$g(\lambda) = -F^*(-\lambda) - G^*(K^T\lambda) \tag{25}$$

7. **Form the Dual Problem**:

$$\max_{\lambda} g(\lambda) = \max_{\lambda} \left[-F^*(-\lambda) - G^*(K^T\lambda)\right] \tag{26}$$

Dual Problem:

$$\max_{\lambda} \left[-F^*(-\lambda) - G^*(K^T\lambda)\right] \tag{27}$$

In summary, the corresponding dual problem of the given nonlinear primal problem is:

$$\max_{\lambda} \left[-F^*(-\lambda) - G^*(K^T\lambda)\right] \tag{28}$$

where $F^*(-\lambda)$ and $G^*(K^T\lambda)$ are the Fenchel conjugates of F and G respectively.

The challenge lies in finding the regularisation parameter map $\mathbf{\Lambda}$. Narrowing down the search space is challenging, as we potentially have d^{mn} different maps to consider, where d is the number of λ values (typically around 100, ranging from 0.01 to 1) and n is the image dimension. This is computationally infeasible. The U-Net can be used to find the regularisation parameter map $\mathbf{\Lambda}$ more efficiently. The search can be done more efficiently by employing neural network models that are designed for image processing. In this dissertation we will use the U-Net model, which is a specific design of a convolutional neural network.

3 Neural Network Architecture

3.1 Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) are a specific type of artificial neural network particularly well-suited for analysing image data and other grid-like patterns. Here we assume a 2D CNN architecture, where each layer is represented as one or more 2-dimensional matrices, rather than a vector as in a normal "vanilla" network.

Inspiration and Function

CNNs are inspired by the visual cortex in the human brain. The visual cortex processes visual information, extracting features one-by-one, from simple edges and shapes to complex objects. CNNs try to mimic this through convolutional layers and pooling layers.

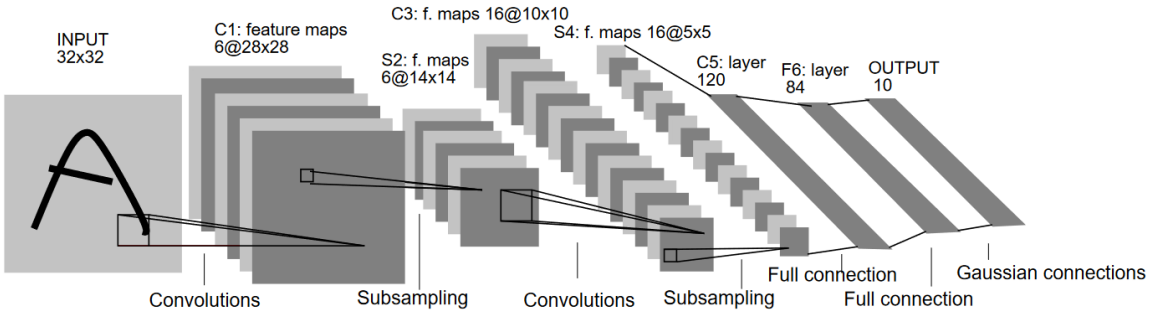


Figure 1: Architecture of LeNet-5, one of the earliest Convolutional Neural Networks Lecun et al., 1998

Convolutional Layers

Similar to a normal "vanilla" neural network layer, each element in a convolutional layer's output comes from a linear combination followed by an activation function. The difference is the use of filters (also called kernels) instead of a single weight matrix. Each filter slides across the input, computing the dot product between its weights and the corresponding elements, then passes the output through an activation function. This is a convolution operation. For each input matrix, one filter produces one output matrix. We can have multiple filters in one layer. Each output matrix is called a feature map, since one filter can be thought of as a feature-extractor that is designed to highlight a specific feature. The number of feature maps is also called the number of channels. Implementation-wise, using filters instead of weight matrices reduces the ratio between the number of weights and the number of output values, hence cutting down on the number of trainable parameters for a image task.

Pooling Layers

An additional type of layer is a pooling layer. Usually we use a max-pooling layer which outputs the maximum instead. This project also utilises max-pooling. The goal is to keep only the most significant values.

These convolutional and pooling layers will form the building block for the U-Net architecture.

3.2 U-Net

The U-Net is a type of neural network that is commonly used for image segmentation Ronneberger et al., 2015. The U-Net is an encoder-decoder network that is designed to take an image as input and produce a segmentation mask as output. The U-Net is made up of a series of convolutional layers that downsample the image and a series of transposed convolutional layers that upsample the image. The U-Net is able to learn to segment images by training on a large dataset of images and their corresponding

segmentation masks.

In our case, we will optimise a U-Net model to find the regularisation parameter map Λ that produces the best denoised image for any particular noisy image.

The U-Net architecture is divided into two principal components: an Encoder and a Decoder.

Encoder

The Encoder functions similarly to a standard CNN, utilising a combination of convolutional layers and max-pooling layers organised into distinct "blocks." In this project, each Encoder block comprises a pair of convolutional layers followed by a max-pooling layer, designed to successively double the number of channels (or feature maps) while reducing the feature map size due to the pooling.

Decoder

The Decoder is also a CNN with a series of blocks. Opposite to an encoding block which doubles the number of channels, each successive decoding block cuts the number of channels (feature maps) in half while increasing the size of the output feature map (hence the "unrolling"). Instead of a max-pooling layer, each decoding block ends with a so-called up-convolutional layer and a skip connection. The up-convolutional layer is just a normal convolutional layer whose output is concatenated with the output of another convolutional layer in an Encoder block.

3.3 Full Architecture

For this project, our model comprises two primary components:

1. A U-Net, denoted as NET_Θ, which is responsible for learning the regularisation parameters $\mathbf{\Lambda}_\Theta$ from an input image $\mathbf{x}_{\text{noisy}}$
2. A Primal Dual Hybrid Gradient (PDHG) algorithm solver

We refer to the set of trainable parameters in the U-Net as Θ, and the output of the U-Net as $\mathbf{\Lambda}_\Theta$.

The general architecture is depicted in shown in Figure 1,

where

- $\mathbf{x}_{\text{true}}$ represents the clean image, serving as the ground truth.
- $\mathbf{x}_{\text{noisy}}$ denotes the noisy image inputted to NET_Θ.
- $\mathbf{\Lambda}_\Theta$ is the regularisation parameters map which is the final output from the U-Net.

We can treat the PDHG solver as the final hidden layer in our network. The result of the loss function $\text{MSE}(\mathbf{x}_{\text{true}}, \mathbf{x}_{\text{denoised}})$ is then used for backpropagation to train the parameters Θ.

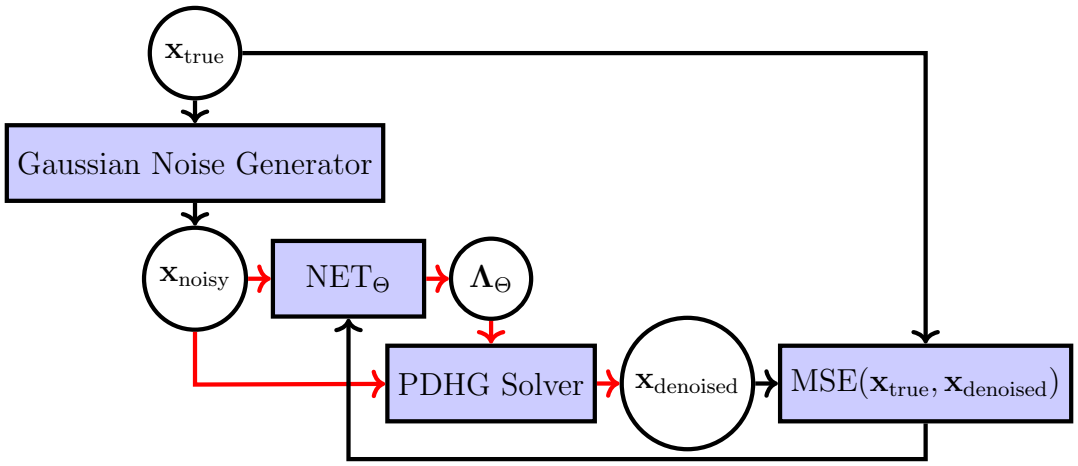


Figure 2: The general architecture

4 Chest X-ray Dataset

4.1 Experimental Set-Up

4.2 Training Data

We used the Chest X-Ray Images (Pneumonia) dataset which was downloaded from Kermany et al., 2018 and

The Chest X-Ray dataset was designed for binary classification of normal and pneumonia chest X-ray images. The dataset consists of 5,863 X-Ray images which are split into two categories (Pneumonia/Normal). All images are black-and-white and are in the JPEG format. The X-rays images have various resolutions.

At the time of download, the original Chest X-ray dataset had already been split into a training set, a validation set, and a test set. We picked 200 images for the training dataset for training and 100 images from the test set for testing. The validation set consists of only 8 images which were all used for validation. We used only images which were classified as normal.

For consistent input, for each dataset, we cropped the images to squares and then resized them to the same resolution. For the Chest X-ray dataset, we set the resolution to 256×256.

All pixel values are originally integers in the range from 0 to 255, so we normalised to the range from 0 to 1 by dividing by 255.

During training, for each image in the Chest X-ray dataset, we generated a random value of σ uniformly in the range from 0.1 to 0.5, and noised the image with Gaussian noise with mean 0 and the corresponding standard deviation σ. The formula used for this dataset is a data-dependent Gaussian noise addition

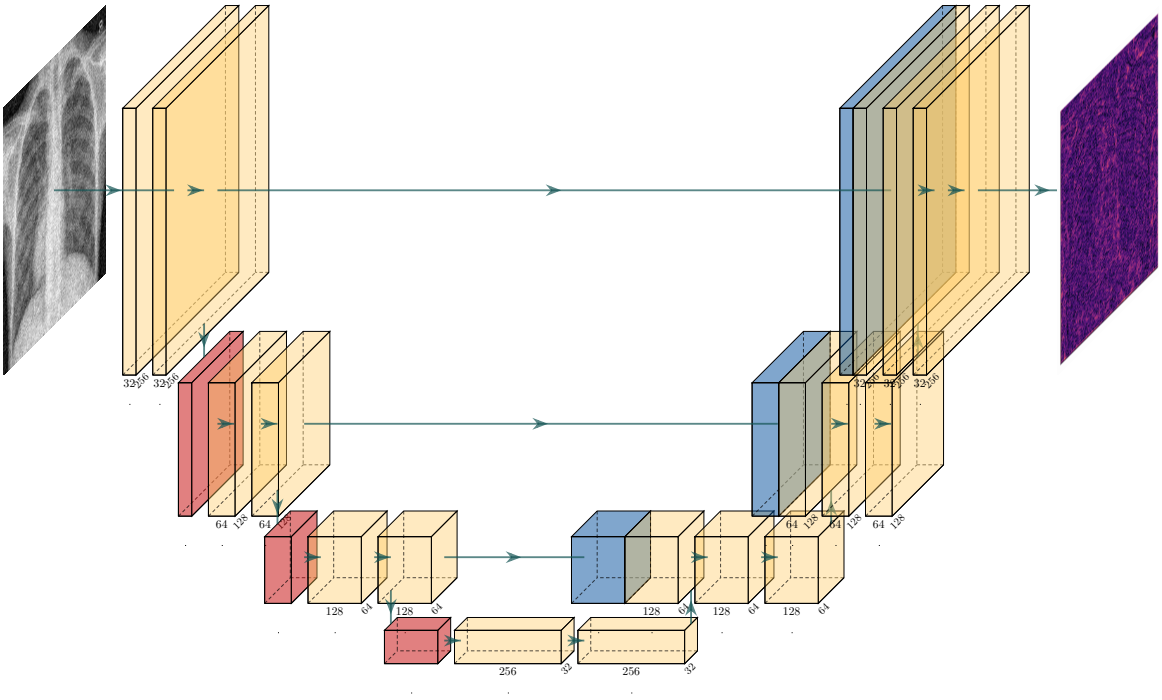


Figure 3: Our NET_Θ Architecture

$$\mathbf{x}_{\text{noisy}} = (\mathbf{x}_{\text{true}} + \sigma\mathbf{n}) \times \sigma_{x_{\text{true}}} \tag{29}$$

where $\sigma_{x_{\text{true}}}$ is the standard deviation of the pixels in the true image $\mathbf{x}_{\text{true}}$. For images with low variation (hence low standard deviation), the added noise is milder.

I am noting this because this might be important to make comparison between different experiments. I have seen that some implementations use custom functions to add noise to the images, which might produce different results. For the same σ value, the noise might be different. The Turtle ID dataset below uses a different method of adding Gaussian noise.

For learning the regularisation parameter map $\mathbf{\Lambda}_\Theta$, we used a U-Net structure with 1 initial double convolutional block, followed by 3 encoding blocks, 3 decoding blocks, and 1 final fully connected layer. The number of initial filters, or output channels of the first convolution layer, is set to 32. As commonly done, each encoding/decoding block contains a(n) downsampling/upsampling step, followed by 2 (fully) convolutional layers, in which the first convolutional layer doubles/halves the number channels which the second maintains the number of channels. In other words, the number of output channels of each subsequent encoding block is doubled, while the number of output channels of each subsequent decoding block is halved. The U-Net has 1 input channel and 2 final output channels. This leads to a 1-32-64-128-256-128-64-32-2 structure.

All convolutional layers have a kernel size of 3×3 and a stride of 1. Each side of

a feature map has zero-padding of size 1 to maintain the size of the feature map after the convolution. Keeping the number of size of the feature map constant after each convolution has the advantage of making the implementation of the skip connections simpler by not having to crop the output of the encoding blocks to match the size of the input of the decoding blocks.

Each encoding block begins with a max pooling layer with 2×2 kernels and a step size of 2. Prior to the max pooling, a zero-padding of size 1 is added in order to exactly halve the length of each side of the output. On the other hand, all upsampling steps are done with linear interpolation with a scale factor of 2 which doubles the length of each side of the feature map. In the end, the total number of trainable parameters of the set Θ is 1,796,034.

For the activation function, we used the Leaky ReLU with a negative slope of 0.01. For the primal dual solver, we set T_{train} to 256. During testing we also set T_{test} to 256.

After the unet produces the regularisation parameter map $\mathbf{\Lambda}_\Theta$, it will be transformed before being used in the PDHG solver. The default is

$$\mathbf{\Lambda}_\Theta = 0.1 \times \times \text{softplus}(\mathbf{\Lambda}_\Theta) \tag{30}$$

which is what I used.

I could also set a custom "up-bound" value so that

$$\mathbf{\Lambda}_\Theta = \text{up-bound} \times \text{sigmoid}(\mathbf{\Lambda}_\Theta) \tag{31}$$

One purpose of these transformations is to ensure that the regularisation parameter map is strictly positive.

Another purpose is to ensure that the regularisation parameters are not too large.

We can control the limit of the regularisation.

A large up-bound value will allow more regularisation power.

However, we might want to limit the regularisation power so that the model does not overfit the training data.

These transformations can reduce the parameters values to 0 after some iterations if the up-bound is set too small. I tried with values up to 0.01 and quickly there was no more improvement. (This is due to the parameter map becoming 0?)

I also tried 0.2 and a similar thing happened.

This is due to the vanishing gradient problem when using sigmoid?

I might experiment changing the code to use only softplus.

Question: What if we simply clip the values to a sensible range like 0 to 0.4?

What if we use only sigmoid or only softplus?

For reproducibility, we manually set a seed value to 42 for the random number generator at the beginning of the training process. We use the Adam optimiser with an initial learning rate of 1e-4, a batch size of 1, and the Mean Square Error (MSE) loss function. We trained for a total of 500 epochs. The total training time was 30 hours, and the GPU memory footprint was around 3 GB.

It is worth noting that, for the implementation of the U-Net, we used the 3D implementations of the convolutional layers as well as the downsampling and upsampling steps from the PyTorch library Paszke et al., 2019. The main reasons we went with 3D instead of the 2D implementations are convenience and generality. The 3D U-Net implementation is adapted from the dynamic image denoising application in Kofler et al., 2023, and can therefore be extended to 3D dynamic imaging applications in the future.

This might, however, increase the GPU memory usage and prevent us from using a larger model or a larger batch size.

Perhaps it is still worth trying to use the 2D implementation.

4.3 Results

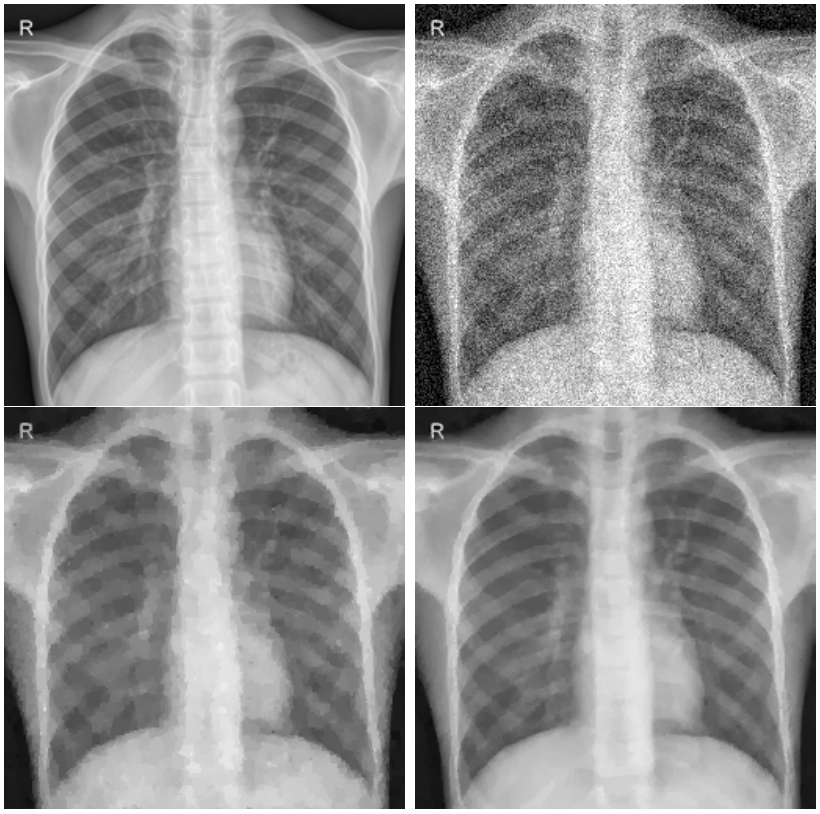
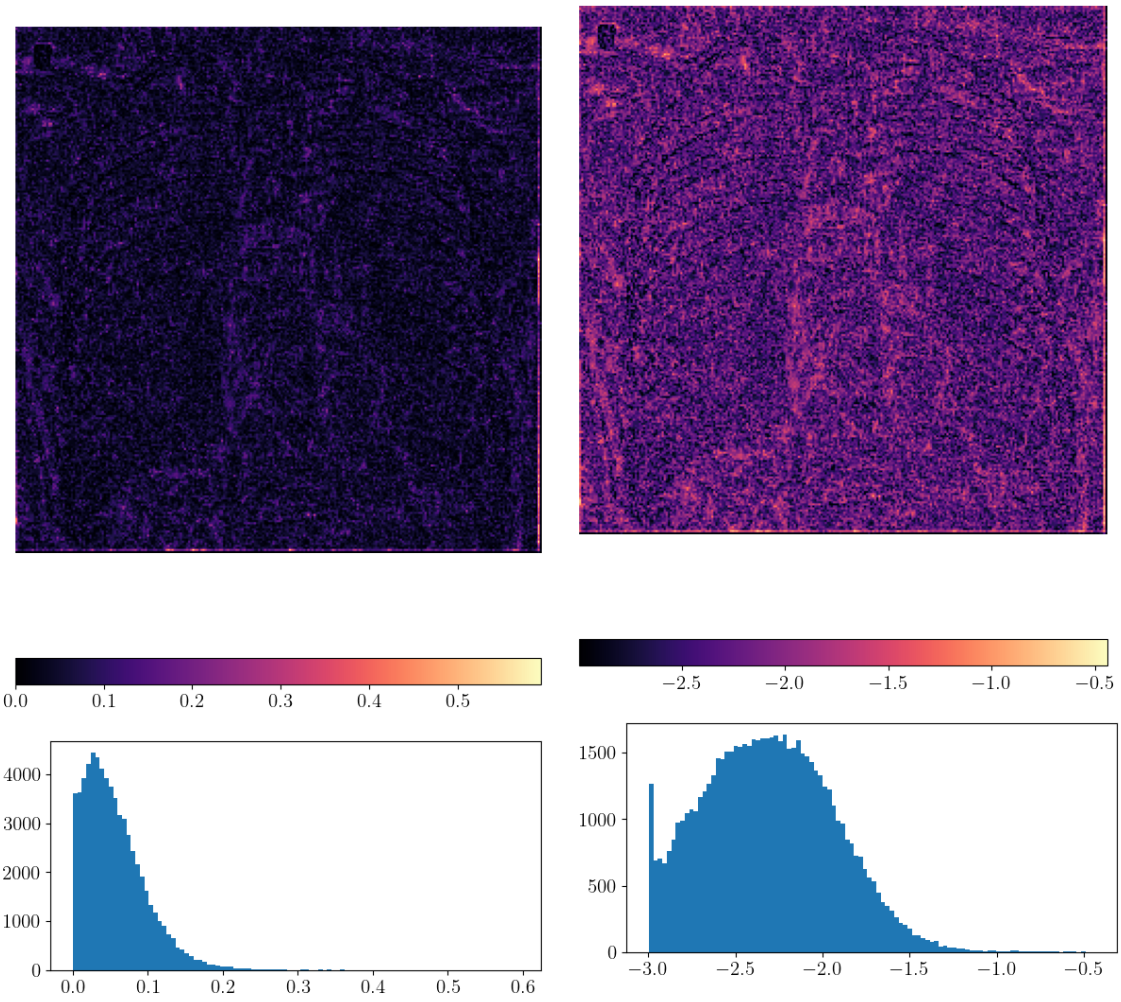


Figure 4: Chest X-ray example. The top row shows the original image on the left and the noisy image, generated with Gaussian noise with $\sigma = 0.5$. The bottom row shows the denoised images using the TV method; the left image with PSNR = 29.945 was produced using the best scalar regularisation parameter $\lambda = 0.07$, while the right image with PSNR = **31.248** was denoised using the regularisation parameter map $\mathbf{\Lambda}_\Theta$ found using NET_Θ.



(a) $\mathbf{\Lambda}_\Theta$ map and histogram of the values

(b) $\log(\mathbf{\Lambda}_\Theta + 0.05)$ and histogram of the log values

Figure 5: The produced $\mathbf{\Lambda}_\Theta$ map for the chest x-ray dataset example, shown as an image on a different colour map, followed by the histogram showing the distribution of the values in the map. The pixel colour represents the magnitude of the regularisation parameter at a particular point. On the left, we show the colours that correspond to the actual values of lambda. As we can see from the histogram, most values stick closely to one another in the lower spectrum, making the image very dark. An additional visualisation is shown on the right, with the colours corresponding to the logarithm of the values with an offset of 0.05 to avoid taking the log of zero. This spread the values out to show more colours for visualisation purposes.

As previously mentioned, we used 100 images with label "normal" from the test set provided in the Chest X-ray dataset for testing. Each test image was noised with a Gaussian noise of mean 0 and standard deviation $\sigma = 0.5$. For each test image, we also

did a grid search with 81 equally spaced values between 0 and 0.8 in order to find the best scalar λ for the total variational denoising method, which we used as a baseline to show the effectiveness of our method.

As an example, we present the results for one of the test images. The results of the grid search is shown in the line plots in figure 6. For this particular test image, the highest value of PSNR of 29.945 dB was achieved by $\lambda = 0.07$. The regularisation parameter map $\mathbf{\Lambda}_\Theta$ found using NET_Θ resulted in a PSNR of 31.248 dB. The clean and original versions as well as the resulting denoised images are shown in figure 4. Qualitatively, we observe a significant reduction in the staircase effect in the denoised image when using the regularisation parameter map $\mathbf{\Lambda}_\Theta$, compared to using the scalar λ.

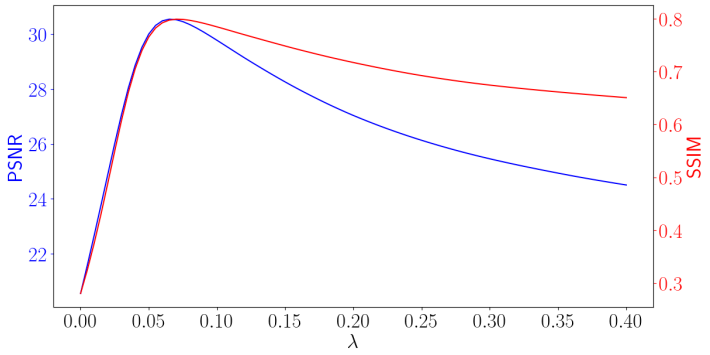


Figure 6: Grid search for the best scalar regularisation parameter λ for the example test image. The highest PSNR of 29.945 dB was achieved by $\lambda = 0.07$.

Figure 5 shows the resulting regularisation parameter map $\mathbf{\Lambda}_\Theta$ found using NET_Θ for this example test image. We can see that the map is made up of a range of values, many of which are less than 0.07, the best scalar regularisation parameter found using the grid search. This shows that at many parts, such as the borders of the body, the details are better preserved and the lines are sharper. On the other hand, there are also many values larger than 0.07, which shows that there are parts that are denoised more heavily. Ideally, these would be parts that were originally smooth. This would help to reduce the staircase effect in the denoised image, which we can see more prevalent in the denoised image using the scalar λ.

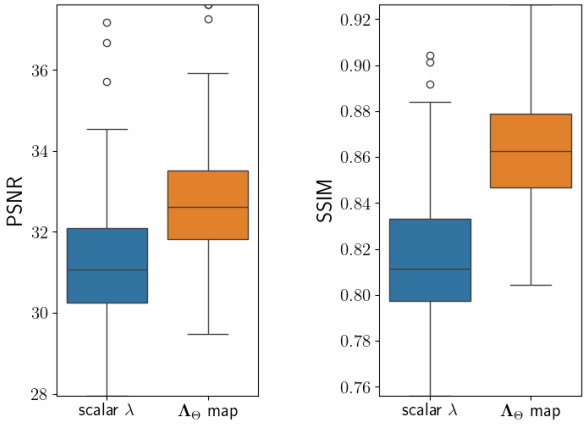


Figure 7: Test results summary

	PDHG - λ	**PDHG - $\mathbf{\Lambda_\Theta}$**
SSIM	0.82 ± 0.03	$\mathbf{0.86} \pm 0.02$
PSNR	31.28 ± 1.39	$\mathbf{32.79} \pm 1.34$

Table 1: Mean and standard deviation of the measures SSIM and PSNR obtained over the test set for the chest x-ray dataset example. The TV-reconstruction using the parameter-map $\mathbf{\Lambda_\Theta}$ improves both measures.

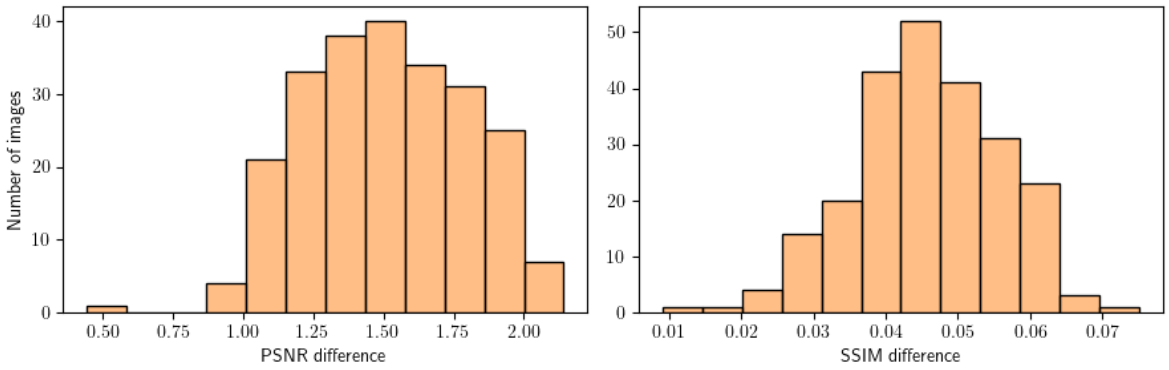


Figure 8: Histograms of the PSNR and SSIM differences between using λ scalar and using $\mathbf{\Lambda_\Theta}$ map for reconstructions of noisy versions of the test images. The $\mathbf{\Lambda_\Theta}$ map method consistently outperforms the scalar λ method.

Plotting the results for all the test images, we can see that the regularisation parameter map consistently outperforms the scalar λ. The box plots in Figure 7 show the distribution of PSNR and SSIM Wang et al., 2004 values for the test images using the two methods. Figure 8 shows the histograms of the differences in PSNR and SSIM

values between the two methods. We can see that the differences are positive in all cases, which shows that the regularisation parameter map consistently produces higher PSNR and SSIM values than the scalar λ.

5 Turtle ID Dataset

We used the Turtle ID data set Adam et al., 2024. The Turtle ID dataset is a dataset of turtle images which are labeled by the IDs of the individual turtles. This dataset is designed for the task of turtle identification. The dataset consists of over 7,000 images, also in JPEG format. All images are color images and have various resolutions.

For the Turtle ID dataset, we split the dataset into training, validation, and test sets with ratio $0.8 : 0.1 : 0.1$. During training, we used 100 training images and 10 validation images.

For the Turtle ID dataset, we set the resolution to 512×512.

The Turtle ID images are converted to greyscale using the common formula $Y = 0.299 \times R + 0.587 \times G + 0.114 \times B$ where R, G, and B are the red, green, and blue channels respectively.

For the turtle ID dataset, We setup using a different method of adding Gaussian noise. We use the common data-independent Gaussian noise addition

$$\mathbf{x}_{\text{noisy}} = \mathbf{x}_{\text{true}} + \sigma\mathbf{n} \tag{32}$$

where the noise $\mathbf{n}$ is drawn from a standard Gaussian distribution with mean 0 and standard deviation 1.

In my implementation, I used the `random_noise` function from `scikit-image` library version `0.24.0`. Examples of noisy images with different σ values generated with my implementation

Before training, for each image, I generated six different noisy images with six different levels of noise: $\sigma = 0.05, 0.1, 0.15, 0.2, 0.25, 0.3$. 100 original images gave me 600 noisy images for training. This helps provide reproducibility for future assessment. For more randomness, we could generate six images with a random level of noise from $0 to 0.3$.

Examples of noisy images with different σ values generated with my implementation are shown in figure 9.

are shown in figure 9.

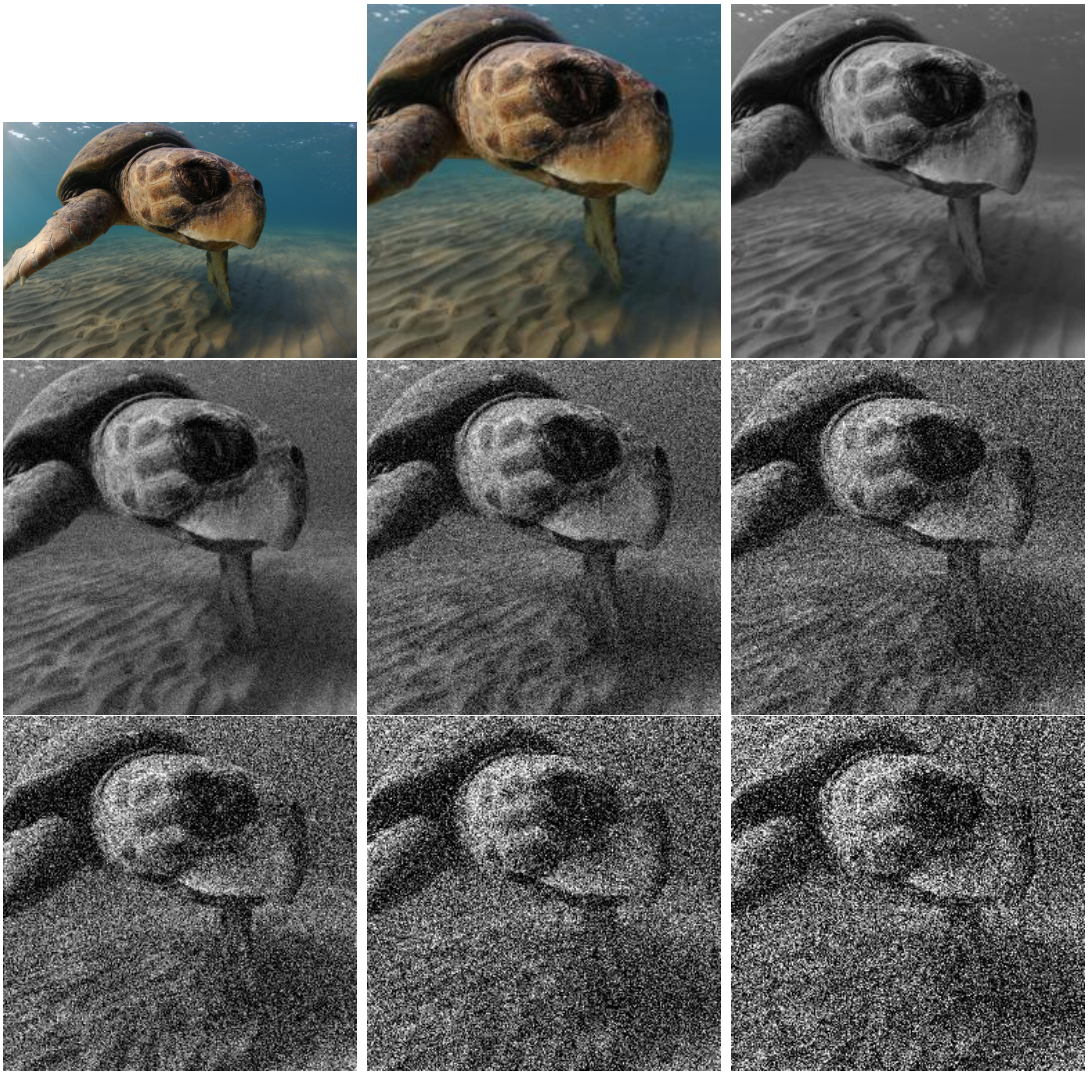


Figure 9: Example of the transformations and noisy versions of one color image. First the image is cropped to a square and rescaled to 256 by 256. Then it is converted to greyscale. Finally, it is noised with Gaussian noise with different σ values. The noise levels are 0.05, 0.1, 0.15, 0.2, 0.25, and 0.3. We can see that at $\sigma = 0.3$, the image is highly degraded which will prove denoising difficult.

5.1 Results

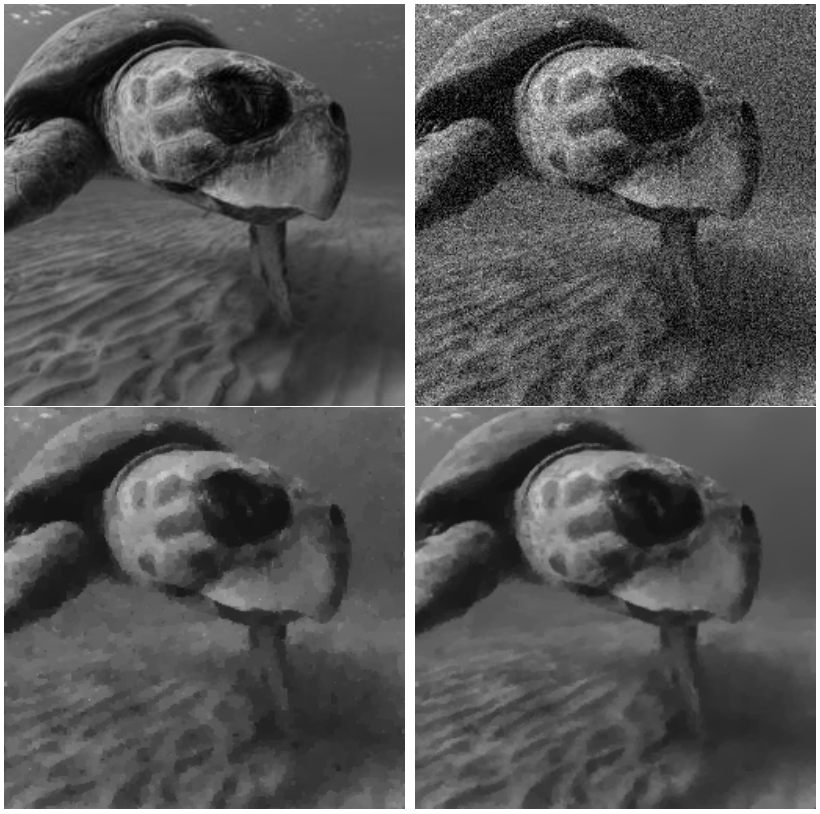


Figure 10: Turtle ID example. The top row shows the original image on the left and the noisy image, generated with Gaussian noise with $\sigma = 0.1$. The bottom row shows the denoised images using the TV method; the left image with PSNR = 29.10 was produced using the best scalar regularisation parameter $\lambda = 0.07$, while the right image with PSNR = **30.12** was denoised using the regularisation parameter map $\mathbf{\Lambda}_\Theta$ found using NET_Θ.

6 Important Implementation Details

6.1 Metrics

The SSIM metrics were calculated using the pytorch Ignite library **pytorch-ignite** for speed. As far as I know, it is the fastest implementation of the SSIM metric. The SSIM metric was calculated using the default settings of the library, which uses a window size of 11 and a Gaussian filter of standard deviation 1.5. This is the common setting for other libraries as well, such as the scikit-image library **scikit-image**. However, the results are a little different from the scikit-image. Furthermore, the results were obtained using the GPU which also differs from the CPU. It might be a good idea to

use scikit-image instead as it is more widely used and tested. The downside is that it uses the CPU which is slower.

My comparison: ...

7 Conclusion

Through experiments with the Chest X-ray dataset, we demonstrated that using U-Net to produce a regularisation parameter map $\mathbf{\Lambda}_\Theta$ enhances the denoising performance of the total variation method compared to the traditional scalar λ.

8 Future Work

- Train models with different T_{train} on the same dataset.
- Train with a different dataset and compare the results.
- Train with a different dataset and test on this dataset, and vice versa.
- Train on colour images.
- Extend to Total Generalised Variation (TGV) method.

References

Adam, L., Čermák, V., Papafitsoros, K., & Picek, L. (2024). Seaturtleid2022: A long-span dataset for reliable sea turtle re-identification. *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 7146–7156. https://www.kaggle.com/datasets/wildlifedatasets/seaturtleid2022

Chambolle, A., & Pock, T. (2011). A first-order primal-dual algorithm for convex problems with applications to imaging. *J. Math. Imaging Vis.*, *40*(1), 120–145. https://doi.org/10.1007/s10851-010-0251-1

Kermany, D., Zhang, K., & Goldbaum, M. (2018). Large dataset of labeled optical coherence tomography (oct) and chest x-ray images. https://doi.org/10.17632/rscbjbr9sj.3

Kofler, A., Altekrüger, F., Ba, F. A., Kolbitsch, C., Papoutsellis, E., Schote, D., Sirotenko, C., Zimmermann, F. F., & Papafitsoros, K. (2023). Learning regularization parameter-maps for variational image reconstruction using deep neural networks and algorithm unrolling. https://github.com/koflera/LearningRegularizationParameterMaps

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://doi.org/10.1109/5.726791

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation.

Shote, D. (2024). Pdhg algorithm implementation for dynamic image denoising. https://github.com/koflera/LearningRegularizationParameterMaps/blob/7537667e81adf3bdebbab8ebdc9networks/dyn_img_primal_dual_nn.py

Sidky, E. Y., Jørgensen, J. H., & Pan, X. (2012). Convex optimization problem prototyping for image reconstruction in computed tomography with the chambolle–pock algorithm. *Physics in Medicine Biology*, *57*(10), 3065. https://doi.org/10.1088/0031-9155/57/10/3065

Wang, Z., Bovik, A., Sheikh, H., & Simoncelli, E. (2004). Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, *13*(4), 600–612. https://doi.org/10.1109/TIP.2003.819861

The following list contain the primary Python libraries used in the implementation of this project.

- `NumPy`: Utilised for numerical representations and operations, including vector and matrix multiplication.

- `PyTorch`: Employed for constructing and training neural network models due to its robust, flexible, and efficient computational dynamics.
- `matplotlib` and `seaborn`: Employed for data visualisation tasks, including the creation of histograms, box plots, and other graphical representations.
- `scikit-learn`: Used for benchmarking our model against traditional machine learning algorithms, providing tools for data preprocessing, cross-validation, and more.
- `random`: Critical for generating random numbers, used extensively in stochastic processes like K-fold cross-validation and bootstrap sampling.
- `wandb`: Integrated for experiment tracking and logging, facilitating the monitoring of model training metrics and performance in real-time.