Graduation Project - Total Variational Image Denoising with U-Net

Thanh Trung Vu - 230849442

Abstract

In this thesis we will describe the theory and application of the total variational (TV) method to image denoising. We will also describe the implementation of the method and an improvement using a U-Net architecture to find the regularisation parameters more efficiently. We will then present the results of the method on a few different datasets.

1 Mathematical preliminaries

2 Background

2.1 Denoising problems

The history of image denoising:

Image denoising began in the 1960s with the development of the Wiener filter, which is a linear filter that minimises the mean squared error between the denoised image and the original image. The Wiener filter works by finding the image that minimises the following objective function:

$$\min_I \frac{1}{2}||I - I_{\text{noisy}}||^2$$

where I is the denoised image, I_{noisy} is the noisy image, and $||\cdot||$ is the mean squared error between two images.

The Wiener filter is a linear filter that works well for images that are corrupted by Gaussian noise. However, the Wiener filter does not work well for images that are corrupted by non-Gaussian noise, such as salt-and-pepper noise.

Before the advance of Neural networks, the total variational (TV) method was one of the most popular methods for image denoising. The TV method works by finding the image that has the smallest total variation and is closest to the noisy image in terms of the mean squared error.

The TV method is based on the idea that images that are smooth have a small total variation, while images that are noisy have a large total variation. By minimising the total variation of the image, the TV method is able to remove the noise from the image and produce a smoother image [**rudin1992nonlinear**].

The TV method is able to denoise images that are corrupted by non-Gaussian noise, such as salt-and-pepper noise, and is able to produce high-quality denoised images [**rudin1992nonlinear**].

Before the advance of neural networks, image denoising was done by hand-crafted methods such as the total variational (TV) method [**rudin1992nonlinear**].

Since the arrival of neural networks, the U-Net has been the most popular method for image denoising.

The U-Net is a type of neural network that is commonly used for image segmentation. 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.

The U-Net is able to learn to denoise images by training on a large dataset of noisy images and their corresponding clean images [**ronneberger2015u**].

The U-Net is able to produce high-quality denoised images that are close to the ground truth images [**ronneberger2015u**].

Here we combine the traditional TV method with the U-Net to produce a more efficient method for image denoising.

The problem with TV method is that it is not clear how to choose the regularisation parameter λ. The U-Net can be used to find the regularisation parameter λ more efficiently. After training with many examples, the U-Net can learn to predict the regularisation parameter λ that produces the best denoised image for any particular noisy image.

2.2 Total Variational Denoising

The total variational (TV) method is a method for image denoising that is based on the principle of minimising the total variation of the image. The total variation of an image is a measure of the amount of variation in the intensity of the image. The TV method works by finding the image that has the smallest total variation and is closest to the noisy image in terms of the mean squared error.

The TV method is based on the idea that images that are smooth have a small total variation, while images that are noisy have a large total variation. By minimising the total variation of the image, the TV method is able to remove the noise from the image and produce a smoother image.

Denoising problems can be described as

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

where $\mathbf{x}_{\text{true}} \in \mathbb{R}^{m \times n}$ is the object to be imaged, $\mathbf{e} \in \mathbb{R}^{m \times n}$ denotes sopme random noise component, and $\mathbf{z} \in \mathbb{R}^{m \times n}$ represents the measured data. The goal is to recover $\mathbf{x}_{\text{true}}$ or at least a good enough reconstruction given $\mathbf{z}$.

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

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

where $d(\cdot, \cdot)$ is a data fidelity term and $\mathcal{R}(\cdot)$ is a regularisation term.

Here we will look at Gaussian noise. The data fidelity term appropriate for Gaussian noise is the square of the ℓ_2 norm:

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

The regularisation term $\mathcal{R}(\cdot)$ is the total variation of the image, which is a measure of the amount of variation in the intensity of the image. The total variation of an image can be calculated using the following formula:

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

where $\nabla \mathbf{x}$ is the gradient of the image $\mathbf{x}$ and $\lambda \in \mathbb{R}^+$ is a regularisation parameter that controls the trade-off between the total variation of the image and the mean squared error between the denoised image $\mathbf{x}$ and the noisy image $\mathbf{z}$.

∇ denotes a finite-differences operator. [**finite˙difference˙op**]. Finite difference operators include, forward difference operator, backward difference operator, shift operator, central difference operator and mean operator. (https://en.wikipedia.org/wiki/Total_variation_denoising#2D_signal_images) Let us focus on the forward difference operator. In particular, in the discrete case, given a matrix $\mathbf{x}$, $\nabla \mathbf{x}$ is 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:

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

We calculate the gradient in the horizontal and vertical directions separately, and then combine them to get the total variation of the image.

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

For the forward gradient, the norm-1 of the vertical gradient is calculated as:

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

and norm-1 of the horizontal gradient is calculated as:

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

where $\mathbf{x}_{i,j}$ is the intensity value of pixel (i, j) in the image $\mathbf{x}$. Note that $\mathbf{x}_{i,j}$ can be a scalar for a black-and-white image or a vector of 3 values for a colour image.

Note that, we can move λ inside the norm, so the regularisation term can be written as:

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

We can see that the regularisation parameter does not need to be a scalar.

$$\mathcal{R}(\mathbf{x}) = \|\Lambda \circ \nabla \mathbf{x}\|_1 = \|\Lambda^{(x)} \circ \nabla_x \mathbf{x}\|_1 + \|\Lambda^{(y)} \circ \nabla_y \mathbf{x}\|_1 \tag{12}$$

where $\Lambda^{(x)}$ and $\Lambda^{(y)}$ are matrices that contain the regularisation parameters for the vertical and horizontal gradients respectively, and $\circ$ denotes the Hadamard or element-wise product. We will refer to Λ as the regularisation parameter map.

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

$$\mathbf{x}_{\text{denoised}} = \arg\min_{\mathbf{x}} = \frac{1}{2} \|\mathbf{x} - \mathbf{z}\|_2^2 + \lambda \|\nabla \mathbf{x}\|_1 \tag{13}$$

if we use a scalar λ regularisation parameter, or

$$\mathbf{x}_{\text{denoised}} = \arg\min_{\mathbf{x}} = \frac{1}{2} \|\mathbf{x} - \mathbf{z}\|_2^2 + \|\Lambda \nabla \mathbf{x}\|_1 \tag{14}$$

if we use a regularisation parameter map Λ of regularisation parameters.

In this thesis, we will use U-Net to help us find the regularisation parameter map Λ given any given (noisy) image. We will compare the results to the best-case scenario where we use a scalar regularisation parameter λ.

Question: Idealy, if we pass in a clean image to U-Net then all the regularisation parameters should be zero, right?

2.3 Related Work

2.4 U-Net

The U-Net is a type of neural network that is commonly used for image segmentation [**ronneberger2015un**
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 λ that produces the best denoised image for any particular noisy image.

3 Method

3.1 Total Variational Denoising

3.2 U-Net

The U-Net is a type of neural network that is commonly used for image segmentation. 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.

4 Implementation

4.1 Total Variational Denoising

The total variational denoising method was implemented using the following steps:

1. The noisy image was passed through a U-Net to produce an initial denoised image.
2. The total variation of the denoised image was calculated using the formula above.
3. The regularisation parameter λ was found by minimising the following objective function:

$$\min_{\lambda} \lambda TV(I) + \frac{1}{2}||I - I_{\text{noisy}}||^2$$

4. The denoised image was then found by minimising the following objective function:

$$\min_{I} \lambda TV(I) + \frac{1}{2}||I - I_{\text{noisy}}||^2$$

4.2 U-Net

The U-Net was implemented using the following architecture:

1. The U-Net was made up of a series of convolutional layers that downsampled the image by a factor of 2.
2. The U-Net was made up of a series of transposed convolutional layers that upsampled the image by a factor of 2.
3. The U-Net was trained on a large dataset of images and their corresponding segmentation masks.
4. The U-Net was able to learn to segment images by training on the dataset.
5. The U-Net was able to produce segmentation masks that were close to the ground truth masks.
6. The U-Net was able to generalise to new images and produce accurate segmentation masks.

5 Introduction

5.0.1 What is image denoising?

5.0.2 Why is image denoising important?

This project is largely based on the research paper "Learning regularisation Parameter-Maps for Variational Image Reconstruction using Deep Neural Networks and Algorithm Unrolling" [**kofler2023learning**] and utilises part of the Smartphone Image Denoising Dataset (SIDD) [**SIDD˙2018˙CV**

The primary goal of this mini-project is to learn about a specific type of neural network called the U-Net, a convolutional network architecture designed for segmentation of images [**ronneberger2015unet**] and apply it to the problem of image denoising.

In the rest of this report, I will explain the task of learning a regularisation parameter map using U-Net, describe the U-Net architecture in this problem, discuss the modelling and training process, and finally present some preliminary results.

5.1 Background

Image processing, and image denoising in particular, has seen significant advancements since the U-Net was first applied in 2015. This project aims to build upon these advancements by implementing and experimenting with a U-Net-based model for image denoising. In particular, it combines U-Net with a traditional image-denoising algorithm called the Primal Dual Hybrid Gradient (PDHG) algorithm [**kofler2023learning**]. Specifically, a PDHG solver is appended to the end of the U-Net, treating the solver as the final layer of the entire network to create an end-to-end unsupervised model.

One notable aspect of using U-Net in this problem is that, instead of acting as a black box and outputting a denoised version of each input image, it outputs a parameter map used in another algorithm, demonstrating the flexibility of neural networks.

Additionally, this approach helps solving the boundary problem when combining multiple patches of an image. Due to insufficient GPU memory, high-definition images must be downscaled or divided into patches for training, and combining patches in a black-box approach is challenging due to boundary issues. However, this approach combines patches of the regularisation parameter map, which can then be passed whole to the PDHG algorithm. The output of the PDHG algorithm typically does not show patch boundaries, resulting in smoother images.

5.2 Contribution

In this project, the model is trained and evaluated using the peak signal-to-noise ratio (PSNR) metric, and preliminary results are analysed to validate the approach. Additionally, the project adapts the code from the original paper, which was used for dynamic image denoising, to apply to static image denoising. This adaptation involved a complete rewriting, simplifying the code from over 200 lines to under 80 lines, making it more understandable and easier to modify. This simplification enabled further modifications, such as adding more layers to the network.

Although much of the logic described in this project is implemented in an open-source project [**kofler2023learning**], getting started can be challenging. Therefore, a self-contained notebook with code and instructions to obtain the data was prepared to help fellow students experiment more easily. This notebook can run on Google Colab [**colab**] and Kaggle [**kaggle**], providing an accessible platform for experimentation. Additionally, part of this report could serve as documentation for the open-source project, enhancing it with additional visualisations and explanations not provided in the original project. These additions aim to help readers of the paper gain a better understanding of the approach.

6 Task Description

6.1 Primal Dual Hybrid Gradient (PDHG)

An iterative method to generate the output which in this case is the denoised image.

The algorithm is

Algorithm 1 PDHG algorithm with fixed regularization parameter-map $\mathbf{\Lambda}_\Theta$ (adapted from [**81**])

Input: $L = \|[\nabla]^{\mathrm{T}}\|$, $\tau = 1/L$, $\sigma = 1/L$, $\theta = 1$, initial guess $\mathbf{x}_0$
Output: reconstructed image $\mathbf{x}_{\mathrm{TV}}$
$\bar{\mathbf{x}}_0 = \mathbf{x}_0$
$\mathbf{p}_0 = 0$
$\mathbf{q}_0 = 0$
for $k < T$ **do**
    $\mathbf{p}_{k+1} = (\mathbf{p}_k + \sigma(\bar{\mathbf{x}}_k - \mathbf{x}_0))\,/(1+\sigma)$
    $\mathbf{q}_{k+1} = \mathrm{clip}_{\mathbf{\Lambda}_\Theta}\left(\mathbf{q}_k + \sigma\nabla\bar{\mathbf{x}}_k\right)$
    $\mathbf{x}_{k+1} = \mathbf{x}_k - \tau\mathbf{p}_{k+1} - \tau\nabla^{\mathrm{T}}\mathbf{q}_{k+1}$
    $\bar{\mathbf{x}}_{k+1} = \mathbf{x}_{k+1} + \theta(\mathbf{x}_{k+1} - \mathbf{x}_k)$
end for
$\mathbf{x}_{\mathrm{TV}} = \mathbf{x}_T$

6.2 Objective

One of the most widely applied image denoising methods is the so-called Total Variation (TV) regularisation [**kofler2023learning**].

Formally, given the loss function

$$L(\mathbf{z}, \mathbf{x}) = \frac{1}{2}\|\mathbf{x} - \mathbf{z}\|_2^2 + \mathrm{TV}(\mathbf{x}),$$

our objective is to find $\hat{\mathbf{x}}$ such that

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} L(\mathbf{z}, \mathbf{x})$$

where $\mathbf{z}$ is the input, presumably noisy, image and $\mathbf{x}$ represent the "clean" image we are looking for. We treat images as vectors, $\mathbf{x}, \mathbf{z} \in \mathbb{R}^{n^2}$, where n is the height of the image, assuming the image is square. In this project, we use $n = 512$.

The TV term can be expressed as either a scalar λ:

$$\mathrm{TV}(\mathbf{x}) = \lambda \|\nabla \mathbf{x}\|_1, \quad \lambda \in \mathbb{R}$$

or as a matrix $\mathbf{\Lambda}$:

$$\mathrm{TV}(\mathbf{x}) = \|\mathbf{\Lambda} \nabla \mathbf{x}\|_1, \quad \mathbf{\Lambda} \in \mathbb{R}^{n^2}$$

Here, $\nabla \mathbf{x}$ represents the spatial gradient, i.e., the difference between adjacent pixels. We use the forward gradient, which calculates the difference between the current pixel and the pixels in the next row and next column.

The challenge lies in finding or learning the regularisation parameter map $\mathbf{\Lambda}$. Using a single λ value penalises all regions of the image equally, which is suboptimal. Instead, it would be better to assign higher penalties to smooth regions and lower penalties to detailed regions to preserve image details. However, this approach is not perfect, as it might leave noise in detailed areas.

Since regularisation is multiplied by the gradient, higher λ values will penalize areas with many changes (typically areas with more details), while lower λ values prioritize detail preservation over noise reduction.

The performance of Primal-Dual Hybrid Gradient (PDHG) methods can be improved by assigning suitable regularisation parameters to regions based on their characteristics. However, narrowing down the search space is challenging, as we potentially have $d^{(n^2)}$ 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.

To address this, we can apply image segmentation. Effective image segmentation helps determine appropriate λ values for different regions. A good network for learning image segmentation is U-Net [**ronneberger2015unet**].

7 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 $\mathbf{x}_0$.
2. A Primal Dual Hybrid Gradient (PDHG) algorithm solver (refer to Appendix), which utilizes both $\mathbf{z}$ and $\mathbf{\Lambda}_\Theta$ to transform $\mathbf{x}_0$ into $\hat{\mathbf{x}}$. This transformation aims to minimise the loss function $L(\mathbf{z}, \mathbf{x})$, thereby facilitating effective reconstruction. The symbol Θ represents the set of trainable parameters within the network.

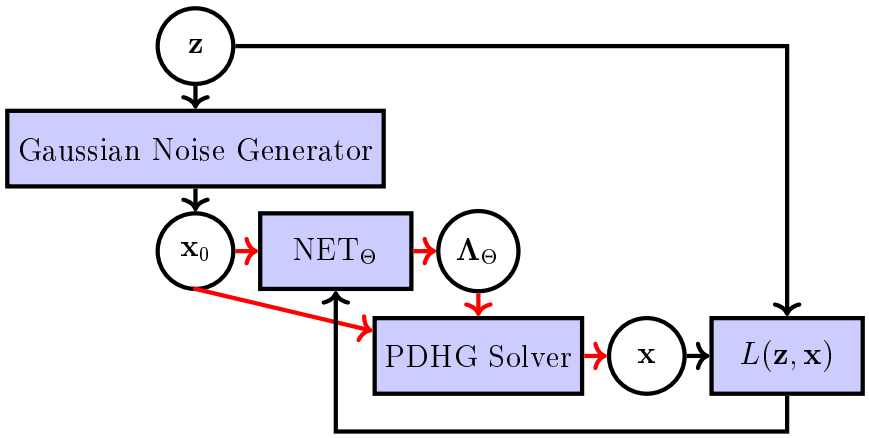

Figure 1: The general architecture

We interpret the PDHG solver as the ultimate hidden layer within our network. The outcome of the loss function $L(\mathbf{z}, \mathbf{x})$ is subsequently employed for backpropagation, essential for training the parameters Θ.

For our implementation, we consider $n \times n$ images, specifically with $n = 512$.

In terms of variables:

- $\mathbf{z} \in \mathbb{R}^{n^2}$ represents the clean image, serving as the ground truth.
- $\mathbf{x}_0 \in \mathbb{R}^{n^2}$ denotes the noisy image inputted to NET_Θ.
- $\mathbf{\Lambda}_\Theta$, the regularisation parameters map, emerges as the final output from the U-Net and is instrumental for the PDHG algorithm solver. Throughout this project, $\mathbf{\Lambda}$ and $\mathbf{\Lambda}_\Theta$ are used interchangeably.

The architecture of NET_Θ is inspired by the U-Net, a specialised type of convolutional neural network architecture.

Let us first delve into the foundational concept of a convolutional neural network, which underpins the U-Net architecture.

7.1 Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) are specialised types of artificial neural networks optimised for analyzing image data and other grid-like structures. We base our understanding on a two-dimensional (2D) CNN architecture, where each layer is comprised of one or more 2D matrices, in contrast to the vectors used in standard "vanilla" neural networks.

Inspiration and Function

The design of CNNs draws inspiration from the human visual cortex, which processes visual stimuli by sequentially extracting features, ranging from simple edges and shapes to complex objects. CNNs replicate this hierarchical feature extraction using convolutional layers and pooling layers.

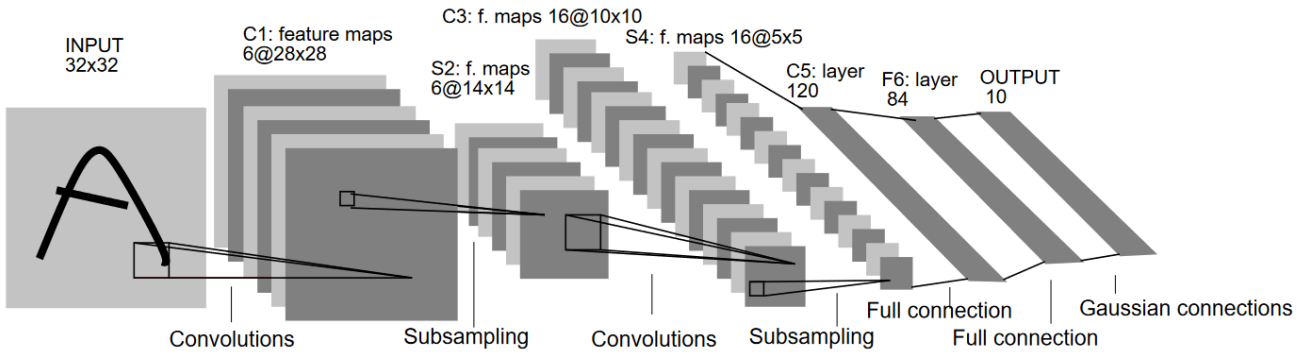

Figure 2: Architecture of LeNet-5, one of the earliest Convolutional Neural Networks [**726791**]

Convolutional Layers

In a convolutional layer, each output element results from a linear combination of inputs, processed through an activation function—akin to "vanilla" neural networks. However, the critical difference lies in the utilization of filters (or kernels). Each filter traverses the input matrix, computes the dot product with the corresponding input elements, and channels the result through an activation function. A single filter generates one output matrix, termed a feature map, acting as a feature extractor that emphasizes specific characteristics of the input. When multiple filters are employed in a layer, they produce multiple feature maps, with each map corresponding to a different channel. This design significantly reduces the proportion of trainable parameters relative to the number of output values, enhancing efficiency for image-related tasks.

Pooling Layers

An essential component of CNNs is the pooling layer, typically implemented as a max-pooling layer. This layer simplifies the output by retaining only the maximum values from the defined regions in the feature map. The objective is to preserve the most significant features while reducing data dimensionality. In this project, max-pooling is a crucial technique used to enhance model performance.

The convolutional and pooling layers constitute the foundational blocks of the U-Net architecture.

7.2 U-Net

A U-Net, as introduced by Ronneberger et al. [**ronneberger2015unet**], extends the conventional CNN architecture by incorporating skip connections, akin to those in a residual network. These skip connections concatenate the outputs of distinct convolutional layers to generate a larger composite output.

Encoder

The U-Net architecture is bifurcated into two principal components: an Encoder and a Decoder. The Encoder functions similarly to a standard CNN, utilizing a combination of convolutional layers and max-pooling layers organized 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 concurrently reducing the feature map size due to the pooling.

Decoder

Conversely, the Decoder is structured around a series of blocks that progressively halve the number of channels while enlarging the feature map size, effectively "unrolling" the data. Unlike the Encoder blocks, each Decoder block concludes with an up-convolutional layer followed by a skip connection. This up-convolutional layer, essentially a regular convolutional layer, outputs data that is then concatenated with the output from a corresponding Encoder block, facilitating the feature integration across different layers of the network.

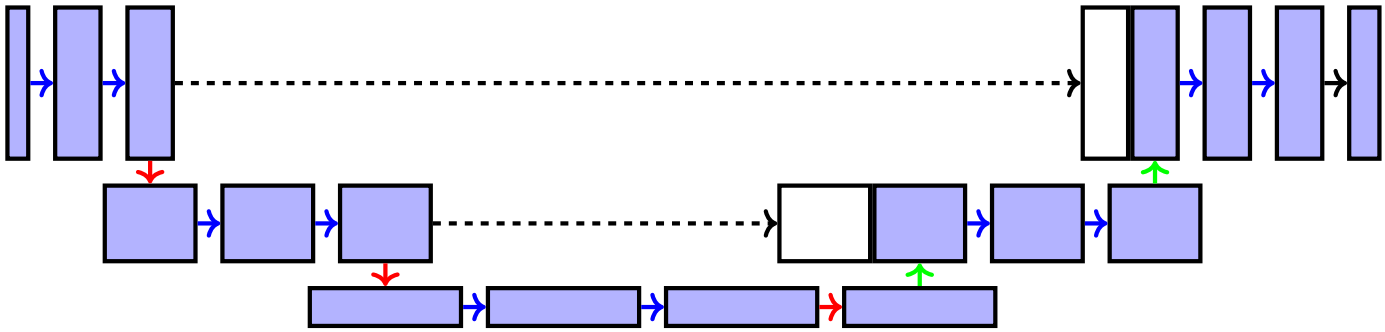

Figure 3: Our NET_Θ Architecture

In our model, the Encoder consists of three encoding blocks. The first block takes a single-channel input—a grayscale image—and expands it to 16 feature maps. The second and third blocks increase the number of channels further to 32 and 64, respectively.

Conversely, the Decoder incorporates two decoding blocks, one fewer than the Encoder. This configuration deviates from some common implementations where the number of encoding and decoding blocks are equal, and a "bottle-neck" comprising two additional convolutional layers is placed between them. In our model, we treat the final block of the Encoder as this bottle-neck. The first decoding block in the Decoder reduces the number of channels back to 32, and the second further reduces them to 16.

Each decoding block in our model concludes with an up-convolutional layer, which serves the same function as a typical convolutional layer, followed by a skip connection. The first skip connection takes the 32-channel output from the second encoding block and merges it with the 32-channel output from the first up-convolutional layer. This combination produces a 64-channel output, which is then processed by the first decoding block. The second skip connection similarly merges the 16-channel output from the first encoding block with the 16-channel output from the second up-convolutional layer, resulting in a 32-channel input for the second decoding block.

The final stage in our architecture employs a convolutional layer to convert the 16-channel output from the last decoding block into a 2-channel output, thus producing two matrices as the ultimate output of our model.

8 Experimental Set-Up

8.1 Chest X-ray Dataset

The original chest X-ray dataset consists of 2000 images of chest X-rays of various resolutions taken from a kaggle dataset [**kaggle**]. All images are black-and-white and are in the JPEG format. We used 800 images for the training dataset, 100 images for the validation dataset, and 100 images for the test dataset. We cropped the images to squares and then resized all images to 256x256 pixels. For each image we generated a random value of σ ranging from 0.1 to 0.5 and noised the image with Gaussian noise with that σ.

we used a U-Net structure with 3 blocks and 32 initial filters to denoise the images. We used the Adam optimiser with an initial learning rate of 1e-4. The batch size is set to 1. We used the Mean Square Error (MSE) loss function to train the U-Net.

We set T_{train} to 256 and T_{test} to 256. We trained for 100 epochs.

All images were resized to 256x256 pixels and noised with Gaussian noise with σ ranging from 0.1 to 0.5.

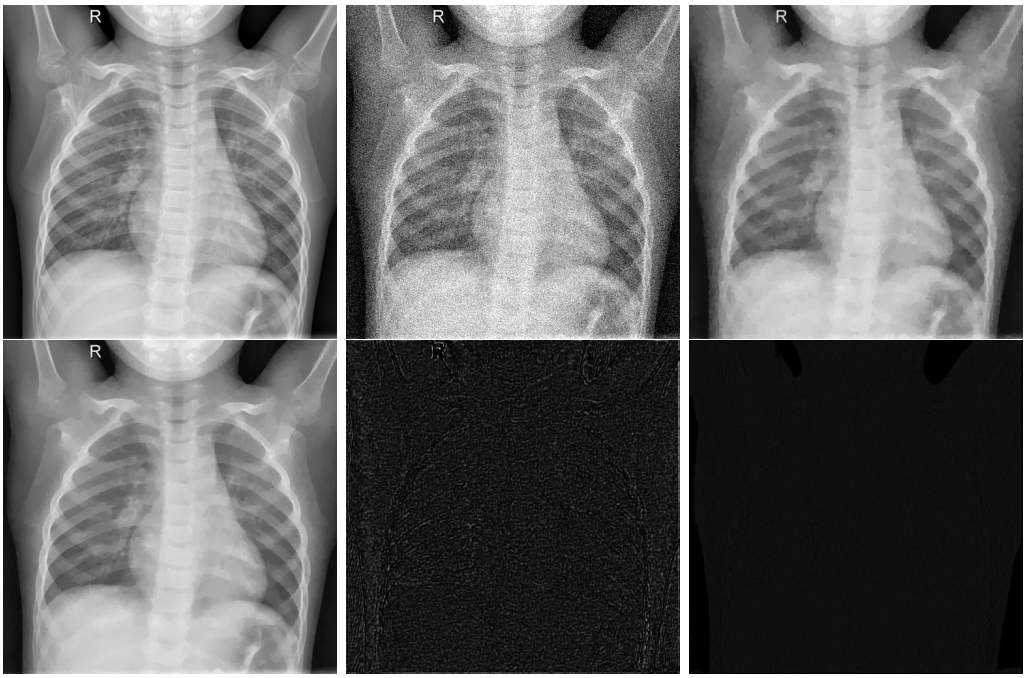

Figure 4: 512×512, T = 128, $\sigma = 0.5$, $\lambda = 0.08$

8.1.1 Experiment 2 (true-voice-32)

https://wandb.ai/wof/chest_xray/runs/wedf9ud9

I used T = 256

8.1.2 Other experiment A ()

:

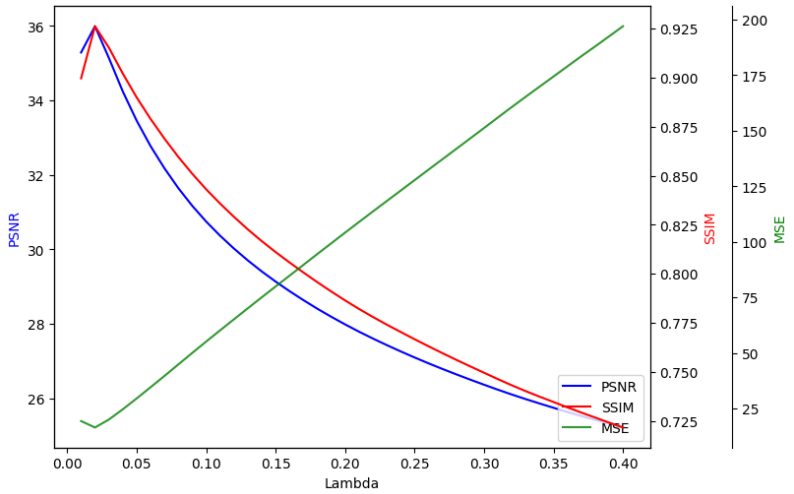

Figure 5: Chest X-ray example - grid search

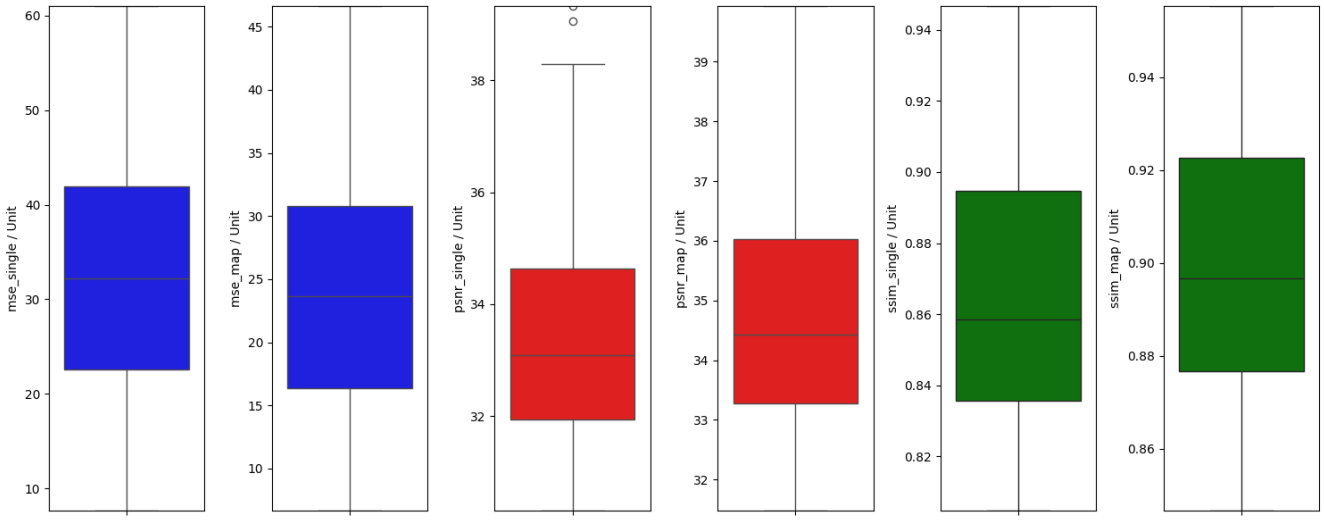

Figure 6: Chest X-ray box plots

https://wandb.ai/wof/chest_xray/runs/xnjgywx7/overview

This is an experiment where I removed PDHG completely and used only the UNet to learn how to denoise an image.

9 Results

The total variational denoising method was able to produce denoised images that were close to the ground truth images. The U-Net was able to produce segmentation masks that were close to the ground truth masks. The U-Net was able to generalise to new images and produce accurate segmentation masks.

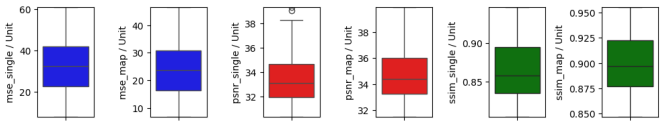

Figure 7: Enter Caption

10 Conclusion

This project showcases the integration of neural networks with traditional algorithms to enhance image denoising capabilities, marking a significant step forward in computational imaging.

10.1 Experimentation and Findings

We experimented with the capabilities of the U-Net architecture to learn and apply regularisation parameters dynamically across different image segments using the PDHG algorithm. Traditional denoising techniques typically rely on a single lambda value for regularisation, which often does not cater to the varying needs across an image. Our approach attempts to transcend this limitation by approximating an optimal lambda map through the U-Net, enhancing the PDHG's performance. This methodology, while computationally intensive, illustrates the potential benefits of segment-specific regularisation in image denoising.

In conclusion, while the experiment demonstrated potential improvements over traditional single-lambda denoising methods, the current computational demands make it impractical for real-world applications. Future research will focus on refining the efficiency of these neural network-enhanced denoising techniques to make them more viable for practical use.

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.