FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Advancements in Image Processing: Bridging Variational Approaches and Mathematical Morphology for Efficient Optimization Solutions

Miguel Faria Amorim



Mestrado em Engenharia Informática e Computação

Supervisor: António Coelho

Co-Supervisor: Jean-Christophe Pesquet

July 25, 2024

© Miguel Faria Amorim, 2024

Advancements in Image Processing: Bridging Variational Approaches and Mathematical Morphology for Efficient Optimization Solutions

Miguel Faria Amorim

Mestrado em Engenharia Informática e Computação

Approved by:

President: Gilberto Bernandes
Referee: Carlos Santiago
Referee: António Coelho

July 25, 2024

Abstract

In recent years, we have witnessed the success of variational methods in image processing, particularly in optimization-based problem formulations. Simultaneously, mathematical morphology (MM) is a well-established methodology for tackling various image-processing tasks. This research bridges the gap between variational methods and MM by studying efficient ways to solve the variational formulation, introduced in the work of Jean-Christophe Pesquet, Hugues Talbot, and Laurent Najman[35]. This work includes variational dilation, erosion, opening, and closing. The primary objective of this thesis is to propose an efficient algorithm for solving the variational formulation of erosion and dilation operations for use on graph data.

Building upon the work of Jean-Christophe Pesquet, Hugues Talbot, and Laurent Najman[35], this thesis continues studying the world of continuous calculus and convex analysis. This study contributes with algorithms and their implementations, to solve the variational problem present in [35]. This project enables, from one side, to efficiently solve, in different ways, the variational formulation previously mentioned, and consequently the study of possible applications to these new solutions.

This work offers a concise overview of the variational formulation of MM operations and proposes efficient algorithms and their implementations for solving the variational formulation and applying it to computer vision tasks such as denoising. Additionally, this delves into the significance of splitting methods in convex optimization, since every solution is based in splitting methods and logic, leveraging PyTorch[47] and OpenCV[13] capabilities. It explores one proximal splitting method, and various root-finding and zeroth order methods to address the variational MM operations, advancing the field of morphological image processing with promising applications in computer vision.

Keywords: Algorithm, Convex Optimization, PyTorch, Variational Morphology, Computer vision, Mathematics, Image enhancement, Neural Networks, Optimization.

ACM Classification: Computing methodologies → Artificial intelligence → Computer vision → Computer vision problems, Mathematics of computing → Mathematical analysis → Mathematical optimization → Continuous optimization

Acknowledgments

I sincerely thank my esteemed supervisors, Professor António Coelho, Jean-Christophe Pesquet, Hugues Talbot and Antonio Silveti Falls for their unwavering support, invaluable guidance, and continuous encouragement throughout the research journey. Their expertise in variational methods, mathematical morphology, and convex optimization has been instrumental in shaping this thesis.

I want to acknowledge the contributions of Jean-Christophe Pesquet, Hugues Talbot, and Laurent Najman, whose foundational work laid the groundwork for this research. Their pioneering efforts in discrete calculus and convex analysis have been a source of inspiration.

Furthermore, I thank not only the Faculty of Engineering of the University of Porto for providing the necessary resources and a conducive research environment but also the CentraleSupélec University for helping me and allowing me to develop this thesis.

Lastly, I thank my family, girlfriend, and friends for their unwavering support, understanding, and encouragement throughout this academic journey. Their belief in my abilities has been a constant source of motivation.

This research was only possible with the collaborative efforts of all those mentioned above. Thank you for being an integral part of this academic endeavor.

Miguel Amorim

"Concurrently, mathematical morphology (MM) is widely recognized a fundamental methodology for solving image processing problems. It thus appears useful to build bridges between these two fields."

Jean-Christophe Pesquet

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Mathematical Morphology and Image Processing . . . 2
  - 1.2.1 Morphological Operations for Image Processing . . . 2
  - 1.2.2 Applications of morphological operations . . . 8
- 1.3 Motivation . . . 9
- 1.4 Objectives . . . 9
- 1.5 Structure of Thesis . . . 10

2 Variational Mathematical Morphology and Convex Optimization **11**
- 2.1 Introduction . . . 11
- 2.2 Evolution of Image Processing through Convex Optimization . . . 12
  - 2.2.1 Projection to proximity operators . . . 12
  - 2.2.2 Minimization with Proximal Algorithms . . . 14
- 2.3 Variational Mathematical Morphology . . . 14
  - 2.3.1 Fundamental Propositions . . . 15
- 2.4 Conclusion . . . 19

3 Algorithms to compute variational mathematical morphological operations **20**
- 3.1 Introduction . . . 20
- 3.2 Methods Developed . . . 25
  - 3.2.1 Root-Finding algorithms for solving 3.3 . . . 25
  - 3.2.2 Zeroth order Algorithms . . . 35
  - 3.2.3 Parallel Proximal Point Algorithm . . . 39
- 3.3 Comparison of algorithms . . . 42
  - 3.3.1 Performance Testing . . . 42
- 3.4 Pytorch Implementation . . . 43
  - 3.4.1 PyTorch Performance Testing . . . 44
- 3.5 Performance of GPU implementation . . . 44
- 3.6 Practical Application and Image Processing Illustrations . . . 45
- 3.7 Conclusions . . . 46

4 Applications of variational MM operations to graph denoising and neural networks **47**
- 4.1 Graphs and mesh denoising . . . 47
  - 4.1.1 Introduction . . . 47
  - 4.1.2 NetworkX Library: A Comprehensive Tool for Graph and Network Analysis 49
  - 4.1.3 Mesh 3D and Trimesh Library: An Overview . . . 52
  - 4.1.4 Rank Filter . . . 54

4.1.5 Conclusion . . . 55
4.2 Neural Networks and Variational MM layers . . . 56
4.2.1 Introduction . . . 56
4.2.2 Learn parameters from minimization problem . . . 57
4.2.3 Introduction to Variational Morphological Layers in Neural Networks . . . 58
4.2.4 Neural Network Study Case - 8 Square Filters . . . 59
4.2.5 Neural Network Study Case - 8 Filters with different U_i . . . 76
4.2.6 Conclusion . . . 89

5 Conclusion **90**
5.1 Future Work . . . 91

References **92**

A Demonstrations of different methods developed **97**
A.1 Demonstration of Bisection Approach . . . 98
A.2 Demonstration of ITP Approach . . . 100
A.3 Demonstration of Golden-Section Search Approach . . . 103
A.4 Demonstration of Ternary Search Approach . . . 106
A.5 Demonstration of PPXA Approach . . . 109

B Convergence Algorithms Testing **112**
B.1 Bisection Method . . . 112
B.2 ITP Method . . . 113
B.3 Golden-Section Search Method . . . 114
B.4 Ternary Search Method . . . 114
B.5 PPXA Method . . . 115

C Confusion Matrix - Neural Network Study Case **117**

List of Figures

1.1 Representation of structuring elements, being 1's the neighborhood and the rest 0's. 3
1.2 A toy example image with concentric diamonds. 4
1.3 Result of applying a dilation to the image in Fig. 1.2. with a square 3x3 ST. . . . 4
1.4 A natural image (cat) 4
1.5 Result of applying a dilation to the image in Fig. 1.4. with a square 3x3 ST. . . . 4
1.6 A toy example image with concentric diamonds. 5
1.7 Result of applying a erosion to the image in Fig. 1.6. with a square 3x3 ST. . . . 5
1.8 A natural image (cat). 5
1.9 Result of applying a erosion to the image in Fig. 1.8. with a square 3x3 ST. . . . 5
1.10 A toy example image with concentric diamonds. 6
1.11 Result of applying a opening to the image in Fig. 1.10. with a square 3x3 ST. . . 6
1.12 A natural image (cat) 6
1.13 Result of applying a opening to the image in Fig. 1.12. with a square 3x3 ST. . . 6
1.14 A toy example image with concentric diamonds. 7
1.15 Result of applying a closing to the image in Fig. 1.14. with a square 3x3 ST. . . . 7
1.16 A natural image (cat). 7
1.17 Result of applying a closing to the image in Fig. 1.16. with a square 3x3 ST. . . . 7
1.18 Calculating inner and outer boundaries, using erosion or dilation. The radius of the structuring element can be used to tune the boundary thickness. 8

2.1 A toy example image with concentric diamonds. 16
2.2 Result of (i) in Fig. 2.1. with $\alpha_i^+ = +\infty$. 16
2.3 Result of (ii) in Fig. 2.1. with $\alpha_i^- = -\infty$. 16
2.4 A natural image (cat). 16
2.5 Result of (i) in Fig. 2.4. with $\alpha_i^+ = +\infty$. 16
2.6 Result of (ii) in Fig. 2.4. with $\alpha_i^- = -\infty$. 16
2.7 A toy example image with concentric diamonds. 17
2.8 A natural image (cat). 17
2.9 Demonstration of Proposition 2 implementation with different values of ρ. 18
2.10 Demonstration of Proposition 3 implementation with different values of ρ. 19

3.1 A toy example image with concentric diamonds. 20
3.2 A natural image (cat). 20
3.3 Results of solving 3.1 with different U_i on image 3.1 and 3.2. 21
3.4 Demonstration of the closed form solution given in Proposition 1 with $\alpha^+ = \infty$ and different U_i's on image 4.1 and 4.2 22
3.5 Demonstration of the closed form solution given in Proposition 1 implementation with $\alpha^- = -\infty$ and different U_i's on images 3.1 and 3.2. 22

3.6 Demonstration of the closed form solution given in Proposition 2 with different values of ρ and different U_i on images 3.1 and 3.2. . . . 23
3.7 Demonstration of the closed form solution given in Proposition 3 with different values of ρ and different U_i on images 3.1 and 3.2. . . . 24
3.8 First derivative plot (equation 3.5) of random pixel of image 3.1 showing the solution to the optimality condition of 3.5. . . . 26
3.9 Inverse of second derivative plot of random pixel of Fig. 3.8 . . . 26
3.10 Bisection Algorithm's Example . . . 27
3.11 Demonstration of Bisection root finding implementation while $p = 1.2$ and U_i of Square 7x7 on images 3.1 and 3.2. . . . 29
3.12 Graphic Examples of ITP Algorithm Steps. . . . 31
3.13 Newton's Method Visual Example. . . . 34
3.14 Diagram of a Golden-section Search. The initial triplet of x values is x_1, x_2, x_3. If f(x_4) = f_{4a}, the triplet x_1, x_2, x_4 is chosen for the next iteration. If f(x_4) = f_{4b}, the triplet x_2, x_4, x_3 is chosen. . . . 36
3.15 Graphic Example of Ternary Search Algorithm . . . 38
3.16 Parallel proximal algorithm (PPXA). . . . 41
3.17 Performance results with PPXA method using U_i Square 3x3. . . . 42
3.18 Performance results without PPXA method using U_i Square 3x3. . . . 42
3.19 Performance results with PPXA method using U_i Square 3x3 on a image 200x200. 43
3.20 Performance results without PPXA method using U_i Square 3x3 on a image 200x200. 43
3.21 PyTorch vs Non-Pytorch Performance changes while increasing image size - PPXA Approach. . . . 44
3.22 PyTorch vs Non-Pytorch Performance changes while increasing image size - Bisection Approach. . . . 45

4.1 Mesh/Graph Implementation Teapot example . . . 48
4.2 Mesh/Graph Implementation Virus example . . . 49
4.3 2D Graph generated by NetworkX . . . 50
4.4 Demonstration of the application of Bisection approach on graph represented in Fig. [**?**] with $p = 1.4$ . . . 50
4.5 3D Graph generated by NetworkX . . . 51
4.6 Demonstration of the application of Bisection approach on graph represented in Fig. with $p = 1.4$ . . . 51
4.7 3D Teapot Mesh . . . 53
4.8 Results of the application of Bisection on mesh represented in Fig. with $p = 1.5$. 53
4.9 Generated 3D Teapot mesh . . . 54
4.10 Noisy 3D Teapot mesh . . . 54
4.11 Results of the application of Rank Filter on graph represented in Fig. with $p = 1.4$ 55
4.12 Example of all digits presented in MNIST dataset . . . 59
4.13 Loss and Validation Accuracy of Convolution Layer . . . 63
4.14 Learned Filters - Convolution layer . . . 63
4.15 Output Filters - Convolution Layer . . . 63
4.16 Loss and Validation Accuracy of variational Dilation Layer . . . 64
4.17 Learned Filters - variational Dilation layer . . . 64
4.18 Output Filters - variational Dilation Layer . . . 64
4.19 Loss and Validation Accuracy of variational Erosion Layer . . . 65
4.20 Learned Filters - variational Erosion layer . . . 65
4.21 Output Filters - variational Erosion Layer . . . 65

4.22 Loss and Validation Accuracy of variational Opening Layer 66
4.23 Learned Filters - variational Opening layer 66
4.24 Output Filters - variational Opening Layer 66
4.25 Loss and Validation Accuracy of variational Closing Layer 67
4.26 Learned Filters - variational Closing layer 67
4.27 Output Filters - variational Closing Layer 67
4.28 Loss and Validation Accuracy of variational Gradient Layer 68
4.29 Learned Filters - variational Gradient layer 68
4.30 Output Filters - variational Gradient Layer 69
4.31 Loss and Validation Accuracy of variational Internal Gradient Layer 69
4.32 Learned Filters - variational Internal Gradient layer 69
4.33 Output Filters - variational Internal Gradient Layer 70
4.34 Loss and Validation Accuracy of variational External Gradient Layer 70
4.35 Learned Filters - variational External Gradient layer 70
4.36 Output Filters - variational External Gradient Layer 71
4.37 Loss and Validation Accuracy of variational TopHat Opening Layer 71
4.38 Learned Filters - variational TopHat Opening layer 72
4.39 Output Filters - variational TopHat Opening Layer 72
4.40 Loss and Validation Accuracy of variational TopHat Opening Layer 73
4.41 Learned Filters - variational TopHat Closing layer 73
4.42 Output Filters - variational TopHat Closing Layer 73
4.43 Accuraccy Plot - All methods 74
4.44 Different values of U_i - neighborhood - ordered for each filter of variational MM layer 78
4.45 Loss and Validation Accuracy of Convolution Layer 78
4.46 Output Filters - Convolution Layer 78
4.47 Loss and Validation Accuracy of variational Dilation Layer 79
4.48 Output Filters - variational Dilation Layer 79
4.49 Loss and Validation Accuracy of variational Erosion Layer 80
4.50 Output Filters - variational Erosion Layer 80
4.51 Loss and Validation Accuracy of variational Opening Layer 81
4.52 Output Filters - variational Opening Layer 81
4.53 Loss and Validation Accuracy of variational Closing Layer 82
4.54 Output Filters - variational Closing Layer 82
4.55 Loss and Validation Accuracy of variational Gradient Layer 83
4.56 Output Filters - variational Gradient Layer 83
4.57 Loss and Validation Accuracy of variational Internal Gradient Layer 84
4.58 Output Filters - variational Internal Gradient Layer 84
4.59 Loss and Validation Accuracy of variational External Gradient Layer 85
4.60 Output Filters - variational External Gradient Layer 85
4.61 Loss and Validation Accuracy of variational TopHat Opening Layer 86
4.62 Output Filters - variational TopHat Opening Layer 86
4.63 Loss and Validation Accuracy of variational TopHat Opening Layer 87
4.64 Output Filters - variational TopHat Closing Layer 87
4.65 Accuraccy Plot - All methods 88

A.1 Demonstration of Bisection root finding implementation while $p = 1.5$ and U_i of square 7x7 on images 3.1 and 3.2 98

A.2 Demonstration of Bisection root finding implementation while $p = 1.8$ and U_i of square 7x7 on images 3.1 and 3.2 . . . 99
A.3 Demonstration of ITP root finding implementation while $p = 1.2$ and U_i of square 7x7 on images 3.1 and 3.2 . . . 100
A.4 Demonstration of ITP root finding implementation while $p = 1.5$ and U_i of square 7x7 on images3.1 and 3.2 . . . 101
A.5 Demonstration of ITP root finding implementation while $p = 1.8$ and U_i of square 7x7 on images 3.1 and 3.2 . . . 102
A.6 Demonstrations of Golden-Section Search zeroth order implementation while $p =$ 1.2 and U_i of square 7x7 on images 3.1 and 3.2 . . . 103
A.7 Demonstrations of Golden-Section Search zeroth order implementation while $p =$ 1.5 and U_i of square 7x7 on images 3.1 and 3.2. . . . 104
A.8 Demonstrations of Golden-Section Search zeroth order implementation while $p =$ 1.8 and U_i of square 7x7 on images 3.1 and 3.2. . . . 105
A.9 Demonstration of Ternary Search zeroth order implementation while $p = 1.2$ and U_i of square 7x7 on images 3.1 and 3.2. . . . 106
A.10 Demonstration of Ternary Search zeroth order implementation while $p = 1.5$ and U_i of square 7x7 on images 3.1 and 3.2. . . . 107
A.11 Demonstration of Ternary Search zeroth order implementation while $p = 1.8$ and U_i of square 7x7 on images 3.1 and 3.2. . . . 108
A.12 Demonstrations of PPXA implementation while $p = 1.2$ and U_i of square 3x3 on images 3.1 and 3.2. . . . 109
A.13 Results of implementation of PPXA implementation while $p = 1.5$ and U_i of square 3x3 on images 3.1 and 3.2. . . . 110
A.14 Results of implementation of PPXA implementation while $p = 1.8$ and U_i of square 3x3 on images 3.1 and 3.2. . . . 111

C.1 Confusion Matrix of Convolutional layer. . . . 117
C.2 Confusion Matrix of Variational Dilation layer. . . . 118
C.3 Confusion Matrix of Variational Erosion layer. . . . 118
C.4 Confusion Matrix of Variational Opening layer. . . . 119
C.5 Confusion Matrix of Variational Closing layer. . . . 119
C.6 Confusion Matrix of Variational Gradient layer. . . . 120
C.7 Confusion Matrix of Variational Internal Gradient layer. . . . 120
C.8 Confusion Matrix of Variational External Gradient layer. . . . 121
C.9 Confusion Matrix of Variational TopHat Opening layer. . . . 121
C.10 Confusion Matrix of Variational TopHat Closing layer. . . . 122
C.11 Confusion Matrix of Convolutional layer. . . . 122
C.12 Confusion Matrix of Variational Dilation layer. . . . 123
C.13 Confusion Matrix of Variational Erosion layer. . . . 123
C.14 Confusion Matrix of Variational Opening layer. . . . 124
C.15 Confusion Matrix of Variational Closing layer. . . . 124
C.16 Confusion Matrix of Variational Gradient layer. . . . 125
C.17 Confusion Matrix of Variational Internal Gradient layer. . . . 125
C.18 Confusion Matrix of Variational External Gradient layer. . . . 126
C.19 Confusion Matrix of Variational TopHat Opening layer. . . . 126
C.20 Confusion Matrix of Variational TopHat Closing layer. . . . 127

Abbreviations and symbols

MM	Mathematical Morphology
ST	Structuring Element
ITP	Interpolation Truncation Projection
PPXA	Parallel Proximal Point Algorithm
GPU	Graphics Processing Unit
CPU	Central Processing Unit
$\oplus$	Dilation operation
$\ominus$	Erosion operation

Chapter 1

Introduction

1.1 Context

A paradigm shift towards integrating advanced mathematical techniques and optimization strategies marks the contemporary landscape of image processing. As the demand for sophisticated image analysis grows, traditional methods face challenges in accommodating the complexities posed by diverse data structures and real-world noise. In this context, our exploration centers on two key pillars: mathematical morphology (MM) and convex optimization.

Mathematical morphology, deeply based in theory, provides a unique lens for characterizing and manipulating image shapes. Its operations, such as dilation, erosion, opening, and closing, offer powerful tools for feature extraction and structural analysis[39]. Variational extensions of these operations introduce a layer of adaptability, allowing work beyond choice of structuring element. This proves to be particularly significant in scenarios where brightness contrasts vary, necessitating asymmetry in the processing approach.

> "Concurrently, mathematical morphology (MM) is widely recognized a fundamental methodology for solving image processing problems. It thus appears useful to build bridges between these two fields."
>
> *Jean-Christophe Pesquet*[35]

Moreover, convex optimization serves as a foundational framework tailored to tackle challenges arising from both differentiable and non-differentiable objectives. This is especially pertinent in domains such as sparse approximation and compressed sensing, where traditional optimization approaches face limitations. By integrating variational MM operations basis with the optimization capabilities of convex methods, this thesis ambitions to navigate the inherent complexities in contemporary image processing.

This research bridges the gap between mathematical morphology and convex optimization, aiming to form innovative solutions in modern image processing. The goal is to enhance the understanding of how these two fields interact, tapping into their combined potential to address

modern challenges in image analysis and manipulation. The project seeks to contribute to the ongoing development of image processing methods, aligning with the demands of today's complex visual data domain.

1.2 Mathematical Morphology and Image Processing

Image processing has experienced an evolution marked by a heightened focus on variational formulations of problems[35] and iterative methods to solve them. The versatility and efficacy of these approaches have elevated them to a prominent position in tackling a broad spectrum of signal and image processing tasks. These tasks involve reducing noise, inpainting, segmentation, and additional tasks.

Mathematical morphology (MM)[35] is a fundamental solution for many image processing problems. In this area of work, morphology refers to depicting the shape and structure of objects within an image. MM operations work on the principles of set theory, emphasizing the relative ordering of pixels, such as in a graph, rather than their numerical values. This distinctive feature enhances their utility in image processing compared to other methodologies. While these operations are explicitly defined for binary images, they prove equally beneficial when extended to grayscale images.

To understand and work with this concept, it is necessary to start studying and analyzing the basics of MM operations and image processing operators[39]: dilation, erosion, opening, and closing.

1.2.1 Morphological Operations for Image Processing

Morphological image processing encompasses the use of a set of non-linear operations associated with the shape or morphology of features within an image[58]. These operations are exclusively dependent on the relative ordering of pixel values, disregarding their numerical values. As a result, morphological operations are well-suited for manipulating binary images. In addition to their utility for binary images, they are also useful for grayscale images and can be extended to color images, showing the flexibility and overall utility and applicability of these operations.

While morphological operations are rooted in set theory, many fundamentally rely on logical operations, making them straightforward to apply. The core of morphological operations, namely erosion, and dilation, are the foundation for the additional operations discussed in this chapter, opening and closing operations, which are based on compositions of the previous operations. The subsequent explanation elaborates on the concept of the structuring element and several other morphological operations[33].

1.2.1.1 Structuring Element

Structuring elements[46] or STs, as it is referred to in this thesis, are fundamental in morphological image processing, representing small matrix sets or sub-images that interact dynamically with the

image under examination. Their crucial role is defining neighborhood structures and facilitating the execution of morphological operations[58].

Typically represented by rectangular matrices of odd dimensions, structuring elements conventionally position the origin at the matrix center. However, variations exist where the origin is strategically placed outside the matrix, showcasing the adaptability of these elements. Binary structuring elements, comprising zeros and ones, play a pivotal role. The matrix arrangement of 0s and 1s reflects active and inactive areas in the image, with some representations focusing solely on the active elements.

The application of an MM operation involves translating the structuring element's origin to all possible pixel locations in the image and comparing image pixels and the structuring element. This dynamic process ensures adaptability to various image regions[39]. Two-dimensional structuring elements broadly fall into flat and non-flat categories, usually called textures. The former maintains a constant value throughout its domain, while the latter exhibits varying values. Analogous to convolution kernels in linear image filtering, structuring elements in morphological image processing form the basis for shaping outcomes in morphological operations. The following image (sourced from [65]) shows a simple representation of different structuring elements.

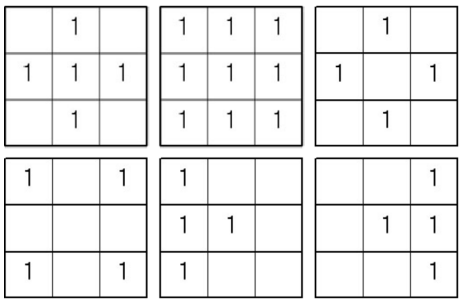


Figure 1.1: Representation of structuring elements, being 1's the neighborhood and the rest 0's.

1.2.1.2 Flat structuring functions

It is common to use flat structuring elements in morphological applications. Flat structuring functions are functions $b(x)$ in the form

$$b(x) = \begin{cases} 0, & \text{if } x \in B, \\ -\infty, \text{otherwise} \end{cases} \tag{1.1}$$

where B is a subset of the image E, denoted as $B \subseteq E$[39].

1.2.1.3 Dilation

Dilation (usually represented by $\oplus$) is one of the basic operations in mathematical morphology. The dilation operation usually uses a structuring element for probing and expanding the shapes

contained in the input image. Therefore, it is dependent on the shape and nature of the structuring element used. The operation of dilating a grayscale image f by a structuring element b is represented as $f \oplus b$, and the resulting image is denoted by the following formula[39]:

$$g(x) = (f \oplus b)(x) = \sup_{y \in E}[f(y) + b(x-y)] \tag{1.2}$$

When applied to a binary image, the transformed image g, pixels are assigned a value of 1 at positions where the origin of the structuring element aligns with the corresponding region in the input image f. More formally, $g(x,y)$ equals 1 if b intersects with f at the location (x,y), and 0 otherwise. This operation is applied to all pixel coordinates (x,y).

The following images show examples of this transformation.

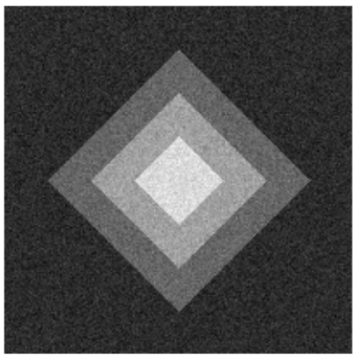


Figure 1.2: A toy example image with concentric diamonds.

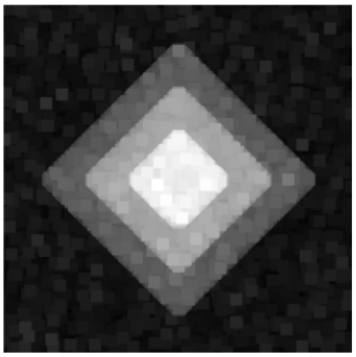


Figure 1.3: Result of applying a dilation to the image in Fig. 1.2. with a square 3x3 ST.



Figure 1.4: A natural image (cat)



Figure 1.5: Result of applying a dilation to the image in Fig. 1.4. with a square 3x3 ST.

1.2.1.4 Erosion

Erosion (usually represented by $\ominus$) is one of two fundamental operations in morphological image processing from which all other morphological operations are based. As dilation, it was originally defined for binary images, later being extended to grayscale images, and subsequently to complete lattices. The erosion operation usually uses a structuring element for probing and reducing the shapes contained in the input image. That's why this operation works, in a practical way, as the opposite of dilation. To erode a grayscale image f with a structuring element b, denoted as $f \ominus b$[39], the resulting image is obtained with

$$g(x) = (f \ominus b)(x) = \inf_{y \in B}[f(x+y) - b(y)] \tag{1.3}$$

When applied to a binary image, the transformed image g, pixels are assigned a value of 1 at positions (x,y) where the structuring element's origin aligns with the corresponding region in the input image f. Formally, $g(x,y) = 1$ if b fits within f at the location (x,y), and $g(x,y) = 0$ otherwise. This operation is repeatedly applied to all the pixel coordinates (x, y).

The following images show examples of this transformation.

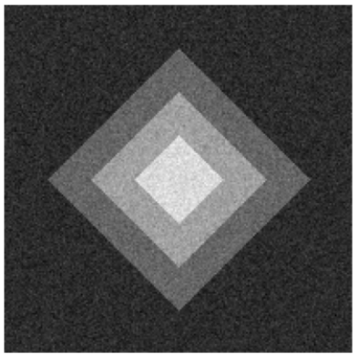


Figure 1.6: A toy example image with concentric diamonds.

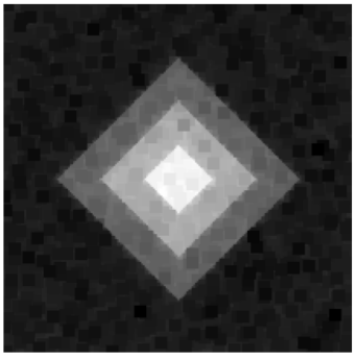


Figure 1.7: Result of applying a erosion to the image in Fig. 1.6. with a square 3x3 ST.



Figure 1.8: A natural image (cat).

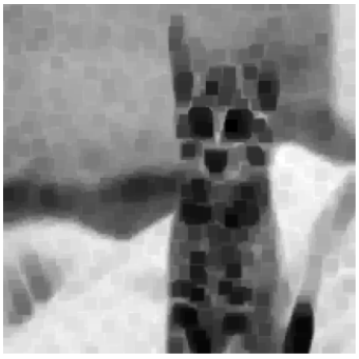


Figure 1.9: Result of applying a erosion to the image in Fig. 1.8. with a square 3x3 ST.

1.2.1.5 Opening

In mathematical morphology, opening is the dilation of the erosion of an image g by a structuring element b[39], creating what it is called an opened image:

$$g \circ b = (g \ominus b) \oplus b \tag{1.4}$$

The opening operation serves in computer vision and image processing as a basic procedure of morphological noise removal. Opening removes small objects from the foreground (usually taken as the bright pixels) of an image, placing them in the background, while closing, which will be discussed in the following subsection, removes small holes in the foreground, changing small islands of background into foreground[55]. These techniques can also be used to find specific shapes in an image. Opening can be used to find things into which a specific structuring element can fit (edges, corners, ...). The opening operation transforms the image by smoothing the outline of an object, effectively removing narrow bridges and minor extensions in the original image f. This operation helps refine the overall structure and appearance of the image, resulting in an improved representation g, useful in detailed situations as blood-vessel images[28], where the representation of thin vessels is improved. The following images show examples of this transformation.

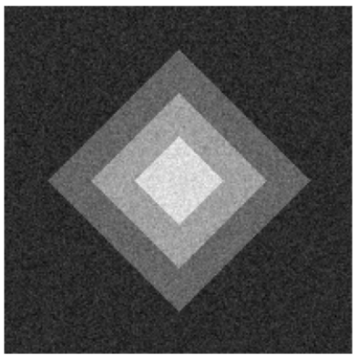


Figure 1.10: A toy example image with concentric diamonds.

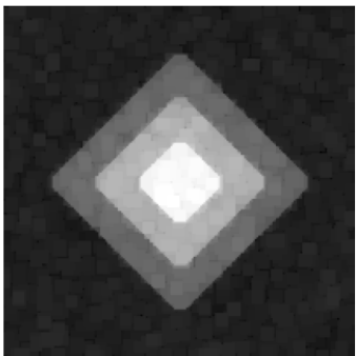


Figure 1.11: Result of applying a opening to the image in Fig. 1.10. with a square 3x3 ST.



Figure 1.12: A natural image (cat)



Figure 1.13: Result of applying a opening to the image in Fig. 1.12. with a square 3x3 ST.

1.2.1.6 Closing

In mathematical morphology, the closing of an image g by a structuring element b is the erosion of the dilation of that set, creating what it is called a closed image:

$$g \bullet b = (g \oplus b) \ominus b \tag{1.5}$$

This operation is, together with opening, the basic workhorse of morphological noise removal[55]. Opening removes small objects, while closing removes small holes. The closing operation can help smooth some sections of contours but its main effect is to blend narrow breaks and thin gaps. This feature proves to be effective in eliminating minuscule gaps and filling in the boundaries of objects. The following images show examples of this transformation.

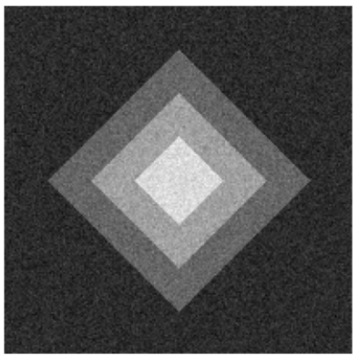


Figure 1.14: A toy example image with concentric diamonds.

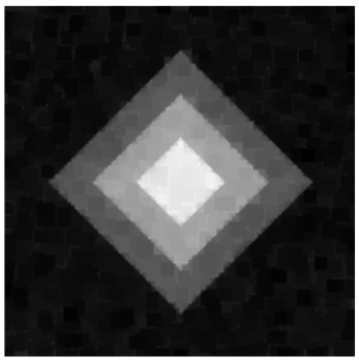


Figure 1.15: Result of applying a closing to the image in Fig. 1.14. with a square 3x3 ST.



Figure 1.16: A natural image (cat).



Figure 1.17: Result of applying a closing to the image in Fig. 1.16. with a square 3x3 ST.

1.2.1.7 Boundaries and outlines

It's possible to make use of the operations mentioned to identify outlines in a binary image. To do this, it's mandatory to have a clear definition of what we mean by 'outline'.

The inner boundary may be defined as the foreground pixels that are adjacent to background pixels. It's possible to determine the inner boundary by the following procedure:

- duplicating the binary image;
- eroding with a 3×3 structuring element;
- subtracting the eroded image from the original.

The outer boundary may be defined as the background pixels that are adjacent to foreground pixels. It's possible to determine the outer boundary by the following procedure:

- duplicating the binary image;
- dilating with a 3×3 structuring element;
- subtracting the original image from the dilated image.

The following image (sourced from [9]) is an example of result boundaries got by MM operations.

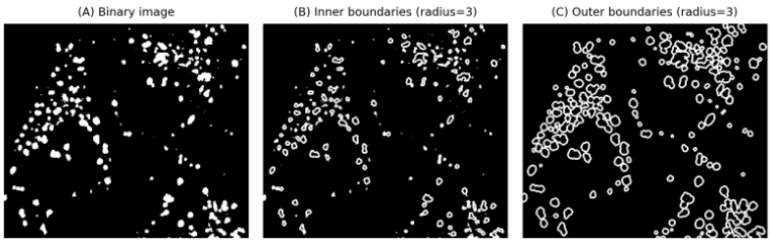


Figure 1.18: Calculating inner and outer boundaries, using erosion or dilation. The radius of the structuring element can be used to tune the boundary thickness.

1.2.2 Applications of morphological operations

The applications of these morphological operations are various. They provide a robust framework for preprocessing and analyzing images, which is essential for improving the accuracy and efficiency of subsequent image processing tasks[39]. For example, in automated medical diagnostics, morphological operations can enhance the visibility of critical features, aiding in the detection and classification of diseases[53]. In robotics, these techniques help in path planning and obstacle detection by simplifying the environment representation[6]. Additionally, morphological operations are pivotal in enhancing the performance of machine learning algorithms by providing clean and

simplified input data. The ability to effectively clean up noise, fill in gaps, and extract meaningful structural information from binary/grayscale images makes these operations indispensable in a wide range of fields, including remote sensing, where they are used to analyze satellite images, and in forensics, where they assist in fingerprint and pattern analysis.

1.3 Motivation

In the area of image processing, the escalating intricacies of visual data pose a challenge to conventional methodologies. Linear approaches, practical in more straightforward contexts, confront limitations in addressing the nuances of complex images, including intricate structures, varied contrasts, and noise influences. The motivation for this research is grounded in the importance to bridge these differences and explore innovative solutions that can robustly handle the evolving demands of modern image-processing applications.

This journey unfolds with a deep dive into variational MM operations[35], a group of versatile operations which are rooted on pixel order relations. Variational MM's unique ability to capture complex image shapes provides a promising avenue for overcoming obstacles encountered in traditional methods. Within this context, variational approaches emerge as pivotal tools, specifically the most basic operations, like dilation, erosion, opening, and closing. Their asymmetric regularization capabilities hold the potential to enhance image processing outcomes, favoring adaptability in scenarios where brightness contrasts vary.

This project explores convex optimization, recognizing its contribution in addressing differentiable tasks ubiquitous in sparse approximation, compressed sensing, and hard-constrained problems. The motivation behind this thesis is the quest for novel image-processing implementations capable of seamlessly navigate through the core of contemporary visual data.

By combining the strengths of MM operations and approaches with the efficiency of convex optimization, this research aims to design and analyze new adaptable and robust image processing solutions that can target challenges of modern applications.

1.4 Objectives

The main goal of this thesis is to enhance image processing by introducing highly efficient numerical algorithms to solve the optimization problem related to the variational formulation of MM operations. Furthermore, this projects aims to leverage the power of the PyTorch framework[47] to develop an efficient algorithm that can run on GPUs for faster processing. This increase in efficiency will enable the application of the algorithm to high-dimensional data beyond typical images, such as those with graph structure.

By progressing through this objective, this thesis aims to contribute to the theoretical understanding of variational morphological operations and efficient convex optimization methods that compute them. Besides the theoretical contribution, there will also be a practical contribution in

the form of efficient software implementations that will be available in an open-source fashion and will be applicable to contemporary computer vision and image processing problems.

Furthermore, this project seeks to demonstrate the versatile nature of the proposed methods and their potential impact on various domains within computer vision and signal processing. One possible application of these operations is in neural networks for classification, potentially enhancing the network's performance by incorporating advanced image processing techniques. Additionally, the algorithm could be applied to graph signal denoising. To explore and validate these potential applications, experiments and study cases were conducted, as discussed later in this thesis. These experiments aim to find practical applications for the solutions developed in this project, studying and concluding their effectiveness in real-world scenarios.

1.5 Structure of Thesis

This thesis comprises five chapters: an introductory chapter, three main chapters, and a conclusion.

The opening chapter introduces the context, motivation, and organizational structure. The second chapter gives a short background on convex optimization as it relates to the problem considered in the thesis and introduces variational MM operations. The third chapter deeply investigates several iterative methods for solving the variational MM operations problem mentioned in this project and evaluation of results. The fourth chapter is based on experiments/study cases of all the work developed on this project.

The thesis concludes summarizing the contributions made in these directions and proposing some possible future work.

Chapter 2

Variational Mathematical Morphology and Convex Optimization

2.1 Introduction

The intersection of Convex Analysis and Mathematical Morphology within graph structures represents a novel and promising frontier in image processing and computational geometry. While various formulations exist in the literature for expressing morphological operations, integrating convex optimization principles with graph-based structures provides a unique and conceptually more straightforward approach. This synthesis addresses continuous and discrete aspects, offering a comprehensive framework for implementing variational morphological operations on graphs and images.

Traditionally, dilation, erosion, opening, and closing operations have been explored through partial differential equations (PDEs) in continuous settings. Noteworthy contributions by Alvarez et al.[2, 3], Brockett and Maragos[14] and Boomgaard and Smeulders[60], and others have laid the foundation for the formulations presented in [35] which form the foundation for the research in this thesis. In fact, the extension of these ideas to graph-based variational frameworks has been largely unexplored until now, leveling the work in this thesis.

In contrast to continuous settings, where challenges exist in defining combined operations like openings and closings in a separable way, the discrete nature of graph structures allows for a complete and well-defined treatment of these operators. This paper explores a graph-based variational framework for mathematical morphology, showcasing its potential to overcome difficulties encountered in the continuous domain. Notably, the framework facilitates the definition of morphological operators in a conceptually clear and computationally efficient manner.

This chapter dives into the main topic of this thesis, variational MM operations and the fundamental explanation of their formulations and propositions. This section works as an introduction to the main problem, followed by possible solution paths. On the other hand, this chapter explains the importance of convex optimization and its relation with variational MM operations.

2.2 Evolution of Image Processing through Convex Optimization

The landscape of image processing has undergone a significant transformation, moving away from its earlier linear methodologies grounded in classical functional analysis and linear algebra. This evolution can be attributed to the advancements in convex optimization[35], a field that emerged as a powerful tool in mathematical modeling and computation[12]. Convex optimization techniques, rooted in convex analysis, have revolutionized how problems in image processing are formulated and solved.

This transformation was instigated by the development of nonlinear analysis in mathematics during the late 1950s and early 1960s[48]. The advent of faster computers coincided with this mathematical progression, enabling the gradual rise of nonlinear techniques. Convex optimization played a pivotal role in this paradigm shift by introducing a robust framework that has demonstrated remarkable efficiency in computing reliable solutions across an expanding spectrum of applications.

The core of numerous challenges in image processing lies in formulating problems in a way that is amenable to convex optimization. These problems involve minimizing sums of convex functions, expressed as $f_1(x) + \cdots + f_m(x)$, where $f_1, \ldots, f_m$ are convex functions mapping from R^N to R.

2.2.1 Projection to proximity operators

In the realm of signal processing, the Projection Onto Convex Sets (POCS) algorithm[48, 15] stands as one of the pioneering convex optimization splitting methods. This algorithm plays a crucial role in recovering or synthesizing signals that adhere to multiple convex constraints simultaneously. The formalization of such a problem aligns within the framework of a convex optimization problem:

$$\min_{x \in R^N} (f_1(x) + \ldots + f_m(x)) \tag{2.1}$$

where $f_1, \ldots, f_m$ are convex functions from R^N to $]-\infty, +\infty]$.

Let C be a nonempty subset of R^N, its indicator function is:

$$\mathbf{1}_C(x) = \begin{cases} 1 & \text{if } x \in C \\ +\infty & \text{if } x \notin C \end{cases} \tag{2.2}$$

This formulation transforms 2.1 into the classical convex feasibility problem:

$$\text{find } x \in \bigcap_{i=1}^{m} C_i \tag{2.3}$$

The POCS algorithm engages each set C_i individually through its projection operator PC_i. The iterative updating rule driving the algorithm is expressed as:

$$x_{n+1} = PC_1 \dots PC_m x_n \tag{2.4}$$

When the intersection of the sets $\bigcap_{i=1}^{m} C_i$ is nonempty ($\bigcap_{i=1}^{m} C_i \neq \emptyset$), the sequence $(x_n)_{n \in N}$ produced by the algorithm converges to a solution of the convex feasibility problem. The problem can be formulated as finding the point x that minimizes the sum of the indicator function $\mathbf{1}_C(x)$ and the squared distance from an initial guess x_0:

$$\min_{x \in R^N} \left(\mathbf{1}_C(x) + \frac{1}{2} \|x_0 - x\|^2 \right) \tag{2.5}$$

Projection algorithms, including the POCS algorithm, have witnessed enrichments and extensions to address a broader spectrum of convex optimization problems. Variants have been proposed to tackle more complex problems, such as solving inverse problems in image reconstruction or signal processing, where the objective is to find the best approximation of an original signal or image from noisy or incomplete observations. However, in scenarios extending beyond the scope of projection methods, where more general operators are required to handle convex optimization problems, proximity operators emerge as a well-suited tool.

Under certain hypotheses, where the indicator function f belongs to the class $\Gamma_0(R^N)$, the proximity operator, denoted as $prox_f$, generalizes the concept of a projection operator. Specifically, for a function $f \in \Gamma_0(R^N)$, the proximity operator is defined by the unique solution to the minimization problem.

The prox computed for each i corresponds and follows the basic operations with proximal algorithms and converges to this formula[49]:

$$\min_{y \in R^N} \left(f(y) + \frac{1}{2} \|x - y\|^2 \right) \tag{2.6}$$

The proximity operator $prox_f : R^N \to R^N$ exhibits attractive properties, making it useful for iterative minimization algorithms. Notably, $prox_f$ is firmly nonexpansive, as expressed by the equality:

$$\mathrm{prox}_{\mathrm{Sf}}(x_i) = (\mathrm{Id} + S\nabla f)^{-1}(x_i) \tag{2.7}$$

where Id corresponds to the identity matrix and S to a scale value.

Proximity operators thus offer a promising avenue for developing algorithms to solve convex optimization problems, akin to the role played by projection operators in convex feasibility algorithms. The attractive properties of proximity operators position them as valuable tools in the context of signal processing, providing a natural interpretation in denoising problems and presenting a versatile framework for addressing a diverse array of convex optimization challenges[38].

2.2.2 Minimization with Proximal Algorithms

Navigating the landscape of convex optimization becomes notably intricate when faced with problems involving more than two functions. The core challenge lies in the formulation of a general minimization problem represented as $f_1(x) + \cdots + f_m(x)$, where $f_1, \ldots, f_m$ are convex functions mapping from R^N to R. Convex optimization algorithms for differentiable functions typically require that the gradient of the objective function is Lipschitz-continuous[5], which is not always satisfied even if the function is continuously differentiable.

However, the existing methodologies are tailored for scenarios involving only two functions. To overcome this limitation, a reformation is proposed, treating the problem as a 2-function challenge in the m-fold product space $H = R^N \times \cdots \times R^N$. This leads to an equivalent problem:

$$\min_{(x_1,\ldots,x_m)\in H} f_1(x_1) + \cdots + f_m(x_m) \tag{2.8}$$

which can be expressed as

$$\min_{x\in H} \iota_D(x) + f(x) \tag{2.9}$$

where $D = \{(x_1, \ldots, x_N) \in H \mid x_i \in R, i = 1, \ldots, N\}$ and $f : x \mapsto f_1(x_1) + \cdots + f_m(x_m)$.

Addressing this complex problem involves specialized algorithms. For example, the Parallel Proximal Point Algorithm (PPXA)[21, 48], derived from the Douglas–Rachford algorithm [10, 48], stands out. It employs a parallelized strategy, converging to solutions for the extended problem, an approach that will be discussed and applied in more detail in a later chapter.

The challenges posed by problems with more than two convex functions find applications in various domains. In scenarios like image recovery, where the goal is to minimize multiple convex functions, the PPXA algorithm proves its significance. This approach is particularly effective in solving numerical problems, providing solutions to complex situations in imaging and signal processing.

Furthermore, a product space strategy is introduced to handle extensions of the problem and composite challenges, showcasing the adaptability of convex optimization methods to intricate scenarios.

In conclusion, delving into convex optimization problems with multiple convex functions ($m \geq 2$) reveals many challenges and algorithmic solutions crucial for addressing complex scenarios in signal processing and related fields.

2.3 Variational Mathematical Morphology

Within mathematical morphology, there are many operations, as seen in the previous chapter. However, a few fundamental operations stand out, namely erosion and dilation, since the more sophisticated MM operations are compositions of these two. Traditionally approached as separate entities with distinct mathematical representations, recent advances in convex optimization have led to a unified variational formulation, providing a cohesive framework for these operations.

Building on the variational formulations of MM operations given in [35], it is possible to use the following formula to define the most basic variational MM operations: erosion and dilation. For every $i \in \{1,\dots,n\}$, let U_i be a nonempty subset of V (although it may be any subset, we will be mainly interested in the case when it corresponds to a neighborhood or reciprocal neighborhood of node i). We will first consider the following convex minimization problem:

$$\min_{x=(x_i)_{1\le i\le n}\in R^n} \sum_{i=1}^{n} \sum_{j\in U_i} \phi_i(\sigma_{\Omega_i}(x_i - y_j - w_{i,j})) \tag{2.10}$$

Here, $(w_{i,j})_{j\in U_i} \in R^{|U_i|}$ is a vector of shift parameters and $\phi_i : [0,+\infty[\to R$ is a (strictly) increasing lower-semicontinuous convex function[17], which means it is one that is continuous from above, meaning that for every $\varepsilon > 0$, there exists $\delta > 0$ such that $\phi_i(x) > \phi_i(y) - \varepsilon$ whenever $x > y$ and $|x-y| < \delta$. Ω_i is a closed real interval, and σ_{Ω_i} denotes the support function of Ω_i, which is defined as:

$$(\forall v \in R) \quad \sigma_{\Omega_i}(v) = \sup_{\xi\in\Omega_i} v\xi = \begin{cases} \alpha_i^- v & \text{if } v < 0 \\ 0 & \text{if } v = 0 \\ \alpha_i^+ v & \text{if } v > 0 \end{cases} \tag{2.11}$$

where $\alpha_i^- = \inf \Omega_i$ and $\alpha_i^+ = \sup \Omega_i$.

The solution to this optimization problem provides a vector of node weights, interpreted as the outcome of nonlinear processing applied to the original node weights y. The dependence of the solution on the choice of intervals Ω_i is a significant aspect requiring further investigation. Throughout the thesis, we will make the following assumption:

Assumption 1: For every $i \in \{1,\dots,n\}$, $-\infty \le \alpha_i^- < 0 < \alpha_i^+ \le +\infty$ with $(\alpha_i^-, \alpha_i^+) \neq (-\infty, +\infty)$

Note: The values α_i^- and α_i^+ may be equal to $-\infty$ and $+\infty$, respectively, which is advantageous for subsequent discussions.

This assumption defines the possible values for the variables α_i^- and α_i^+, ensuring that they are within valid ranges for the problem to be convex and well-defined.

2.3.1 Fundamental Propositions

The following propositions are formulated in [35] and also explained with more detail. Their numerical demonstrations come from the Python implementation developed within this project and explained in the upcoming chapter 3.

2.3.1.1 Proposition 1

Proposition 1 addresses the existence of a solution to the minimization problem under certain assumptions. Let's delve into the proposition and its corresponding formulas:

This formulation assures us that a solution exists for the minimization problem under the specified conditions.

(i) If $\alpha_i^+ = +\infty$:

$$x_i = \min_{j \in U_i}(y_j + w_{i,j}) \tag{2.12}$$

(ii) If $\alpha_i^- = -\infty$:

$$x_i = \max_{j \in U_i}(y_j + w_{i,j}) \tag{2.13}$$

(iii) If ϕ_i is strictly convex, then x_i is uniquely defined.

These properties characterize the behavior of the solution depending on the range limits of α_i^+ and α_i^-, and the convexity of the function ϕ_i.

In summary, Proposition 1 provides valuable insights, showing that the variational formulation agrees with the vanilla MM operations in the limit, i.e., when α^+ or α^- is equal to $+\infty$ or $-\infty$ respectively, laying the groundwork for further analysis and understanding of the optimization process.

All the following numerical results are based on operations envolving a U_i of a square with shape 3x3.

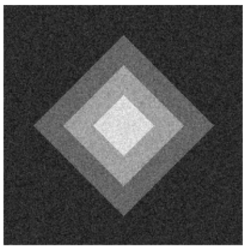


Figure 2.1: A toy example image with concentric diamonds.

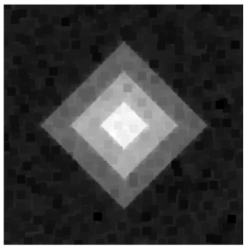


Figure 2.2: Result of (i) in Fig. 2.1. with $\alpha_i^+ = +\infty$.

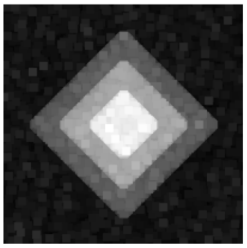


Figure 2.3: Result of (ii) in Fig. 2.1. with $\alpha_i^- = -\infty$.



Figure 2.4: A natural image (cat).

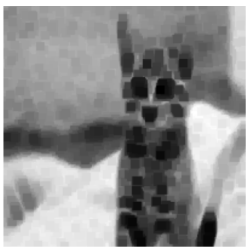


Figure 2.5: Result of (i) in Fig. 2.4. with $\alpha_i^+ = +\infty$.



Figure 2.6: Result of (ii) in Fig. 2.4. with $\alpha_i^- = -\infty$.

2.3.1.2 Proposition 2

Proposition 2 presents a closed-form solution to the variational formulation 2.10 for a particular choice of ϕ_i. It computes the optimal values of x_i, the i-th component of the optimal solution to the

optimization problem. Let $i \in \{1,\dots,n\}$ and let x_i be the i-th component of the optimal solution to 2.10 where $\varphi_i : [0,+\infty[\to R$ is defined as $\xi \mapsto \xi^2$. Denote by $(z_{i,j})_{1\leq j\leq |U_i|}$ the coefficients $(y_j + w_{i,j})_{j\in U_i}$ which have been re-indexed in increasing order, and set $z_{i,0} = -\infty$ and $z_{i,|U_i|+1} = +\infty$. If $(\alpha_i^-, \alpha_i^+) \in R^2$, then x_i is the unique value in $[z_{i,j_i-1}, z_{i,j_i}]$ with $j_i \in \{1,\dots,|U_i|+1\}$ such that:

$$x_i = \frac{(\alpha_i^+)^2 \sum_{j=1}^{j_i-1} z_{i,j} + (\alpha_i^-)^2 \sum_{j=j_i}^{|U_i|} z_{i,j}}{(j_i-1)(\alpha_i^+)^2 + (|U_i| - j_i + 1)(\alpha_i^-)^2} \tag{2.14}$$

where:

- α_i^- and α_i^+ represent the lower and upper bounds of the interval Ω_i respectively;
- $|U_i|$ denotes the cardinality of the set U_i, representing the number of elements in the subset U_i;
- $z_{i,j}$ are coefficients calculated as $y_j + w_{i,j}$, where $w_{i,j}$ are the shift parameters;
- j_i represents the index within the set U_i such that x_i lies in the interval $[z_{i,j_i-1}, z_{i,j_i}]$.

This formula provides a useful expression for computing the optimal value of x_i, without being necessary to solve the optimization problem using an iterative algorithm taking into account the bounds of α^+ and α^-. It offers a practical tool for implementing algorithms to solve the optimization problem 2.10 and enables a deeper understanding of the behavior of the optimal solution in relation to the problem parameters.

All the upcoming implementations are based on operations envolving a U_i of a square with shape 3x3 on images 2.7 and 2.8.

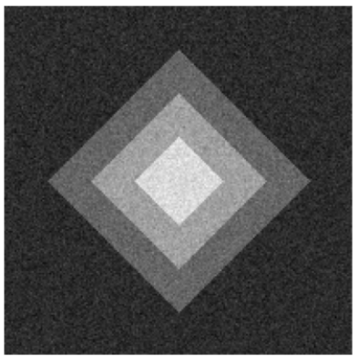
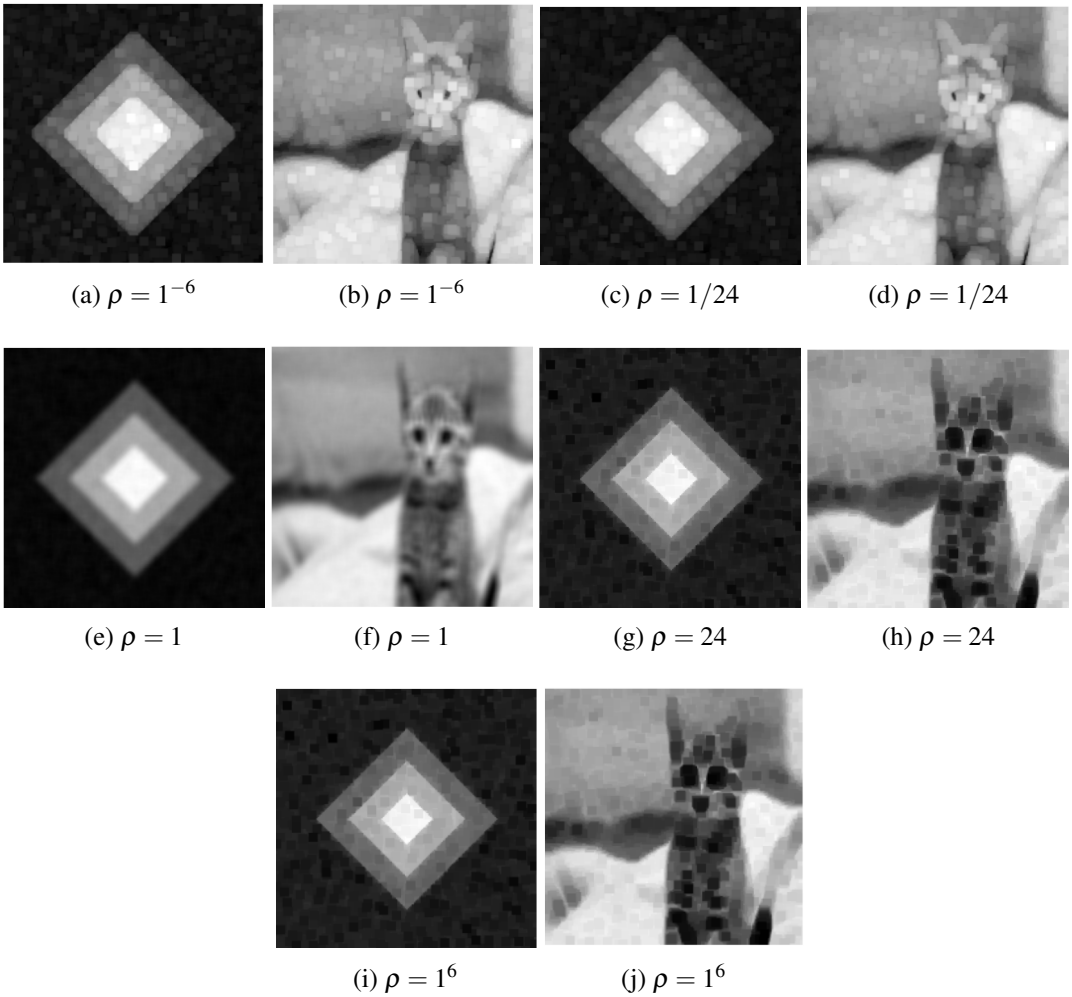


Figure 2.7: A toy example image with concentric diamonds.



Figure 2.8: A natural image (cat).

Let $\rho = -\alpha^+/\alpha^-$, the following images (image 2.9) show some examples of implementations of proposition 2 with different values of ρ:



(a) $\rho = 1^{-6}$ (b) $\rho = 1^{-6}$ (c) $\rho = 1/24$ (d) $\rho = 1/24$

(e) $\rho = 1$ (f) $\rho = 1$ (g) $\rho = 24$ (h) $\rho = 24$

(i) $\rho = 1^{6}$ (j) $\rho = 1^{6}$

Figure 2.9: Demonstration of Proposition 2 implementation with different values of ρ.

2.3.1.3 Proposition 3

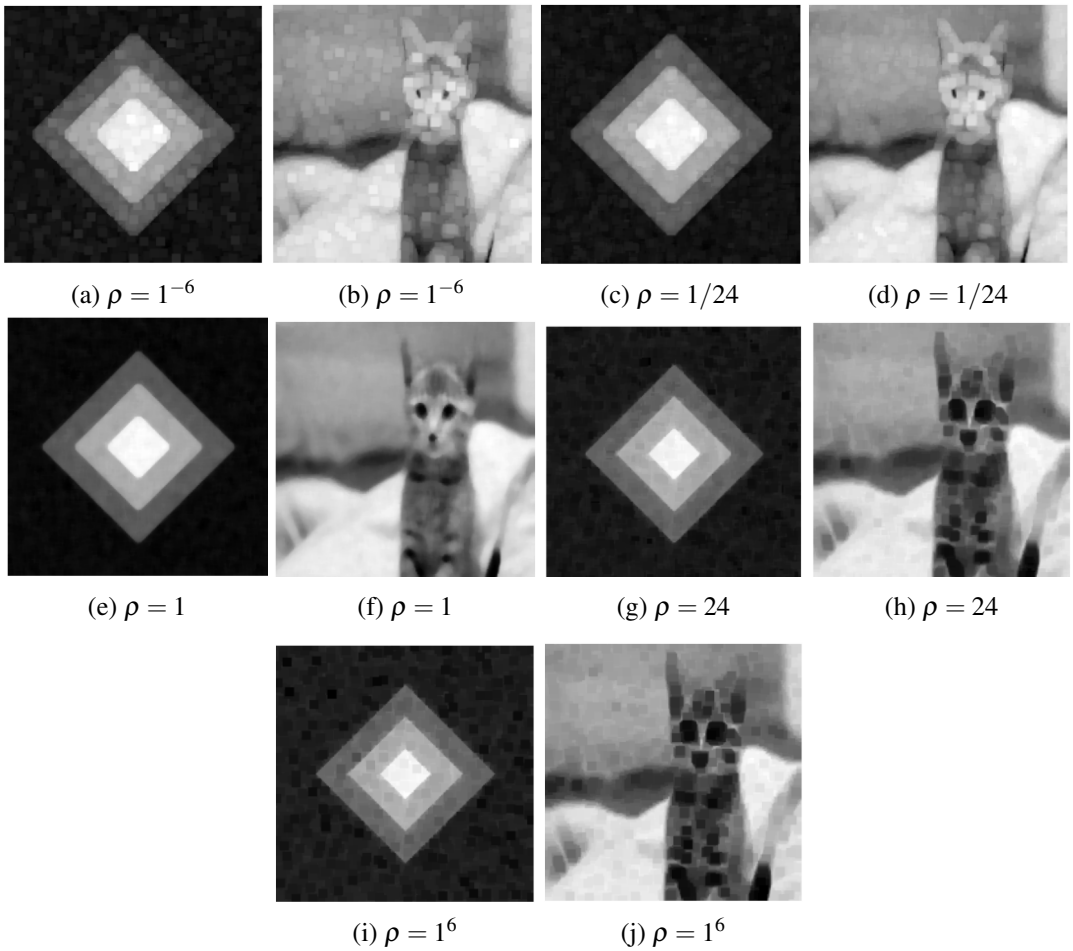
In this proposition, i belongs to the set $\{1,\ldots,n\}$, and x_i represents the i-th component of an optimal solution to problem 2.10. Here, $\varphi_i : [0,+\infty[\to R : \xi \mapsto \xi$ is the function for each pixel i. The coefficients $(z_{i,j})_{j=1}^{|U_i|}$ represent the re-indexed sums $(y_j + w_{i,j})$ over the neighborhood U_i, sorted in increasing order. The algorithm calculates the value j_i as per the given formula:

$$j_i = \left\lfloor \frac{|U_i|}{1-((\alpha_i^+)/(\alpha_i^-))} \right\rfloor + 1 \tag{2.15}$$

Then, if $\frac{|U_i|}{1-((\alpha_i^+)/(\alpha_i^-))} \notin N$, $x_i = z_{i,j_i}$.

If $\frac{|U_i|}{1-((\alpha_i^+)/(\alpha_i^-))} \in N$, then x_i can be chosen equal to any value in $[z_{i,j_i-1}, z_{i,j_i}]$.

Using the same images as in Proposition 2 (images 2.7 and 2.8), all the following implementations (image 2.10) are based on operations envolving a U_i of a square with shape 3x3.



(a) $\rho = 1^{-6}$ (b) $\rho = 1^{-6}$ (c) $\rho = 1/24$ (d) $\rho = 1/24$

(e) $\rho = 1$ (f) $\rho = 1$ (g) $\rho = 24$ (h) $\rho = 24$

(i) $\rho = 1^{6}$ (j) $\rho = 1^{6}$

Figure 2.10: Demonstration of Proposition 3 implementation with different values of ρ.

2.4 Conclusion

This chapter explored the intersection of convex optimization and variational mathematical morphology within graph structures, offering the basics formulations of variational MM, their relation with the vanilla MM operations and examples of their respective implementation. On the other hand, it was explored the differences between traditional MM operations and the variational ones presented on this chapter and examined their formulations through convex optimization.

Furthermore, we presented key propositions that underpin the variational MM operations and also this all project, demonstrating the practical implementation of these concepts through detailed examples and visualizations. These propositions lay the groundwork for a deeper understanding of the optimization processes and the behavior of solutions under various conditions.

In the next chapter, we will delve into the methods developed for implementing the proposed framework, providing a thorough explanation of the algorithms and their practical applications. This will include a detailed discussion on the implementation of variational MM operations using the principles outlined in this chapter, supported by empirical results and performance analysis.

Chapter 3

Algorithms to compute variational mathematical morphological operations

3.1 Introduction

Using 2.10 presented in chapter 2, the first and main goal of this project was to explore practical and efficient ways to solve the following minimization problem:

$$\min_{x=(x_i)_{1\leq i\leq n}\in R^n} \sum_{i=1}^{n} \sum_{j\in U_i} \phi_i(\sigma_{\Omega_i}(x_i - y_j - w_{i,j}))\} \tag{3.1}$$

The main goal of this chapter is to show the algorithms developed to solve this minimization problem while having a really strong implementation in terms of efficiency and performance. The operations we use to solve 3.1 are vectorized over all pixels in an image or all nodes in a graph, so that we do not have to loop over the pixels or nodes respectively. With this, a really good performance could be achieved and, consequently, have implementations that could be used everyday in an open-source way.

Furthermore, another important goal achieved was the versatility of the process/algorithm developed, being able to process and work with different values of U_i (also known as neighborhood of each pixel). Using different neighborhood shapes U_i, the variational mathematical morphological operation can be designed to extract specific features from an image, for instance geometric shapes like lines or crosses can be emphasized by choosing a neighborhood U_i with this shape. Examples of this implementation on images 3.1 and 3.2 are illustrated in the following image 3.3.

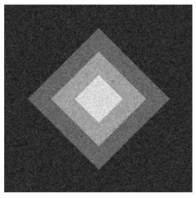
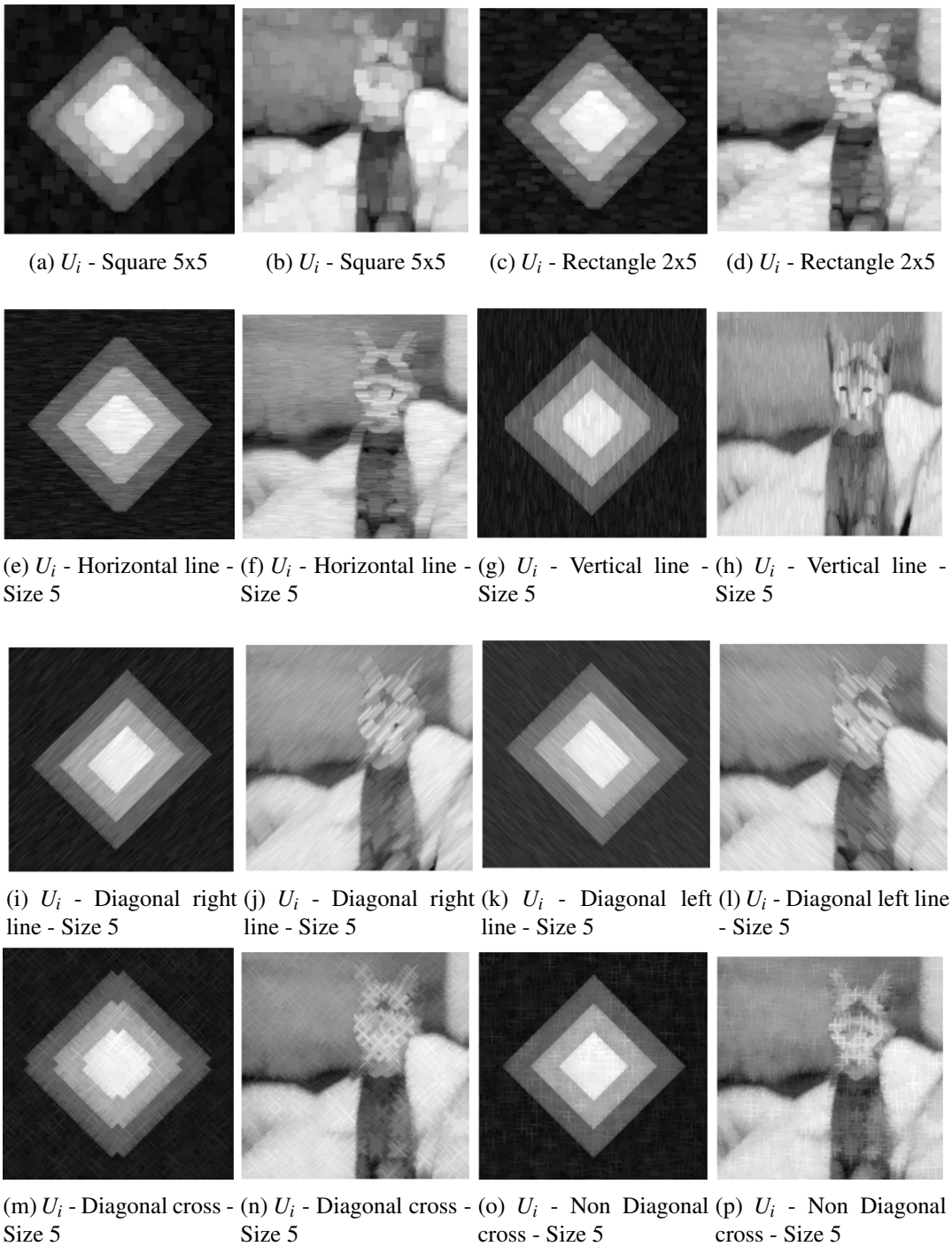


Figure 3.1: A toy example image with concentric diamonds.



Figure 3.2: A natural image (cat).

Let $\rho = 1e-6$ ($\rho = -\alpha^+/\alpha^-$):



(a) U_i - Square 5x5 (b) U_i - Square 5x5 (c) U_i - Rectangle 2x5 (d) U_i - Rectangle 2x5

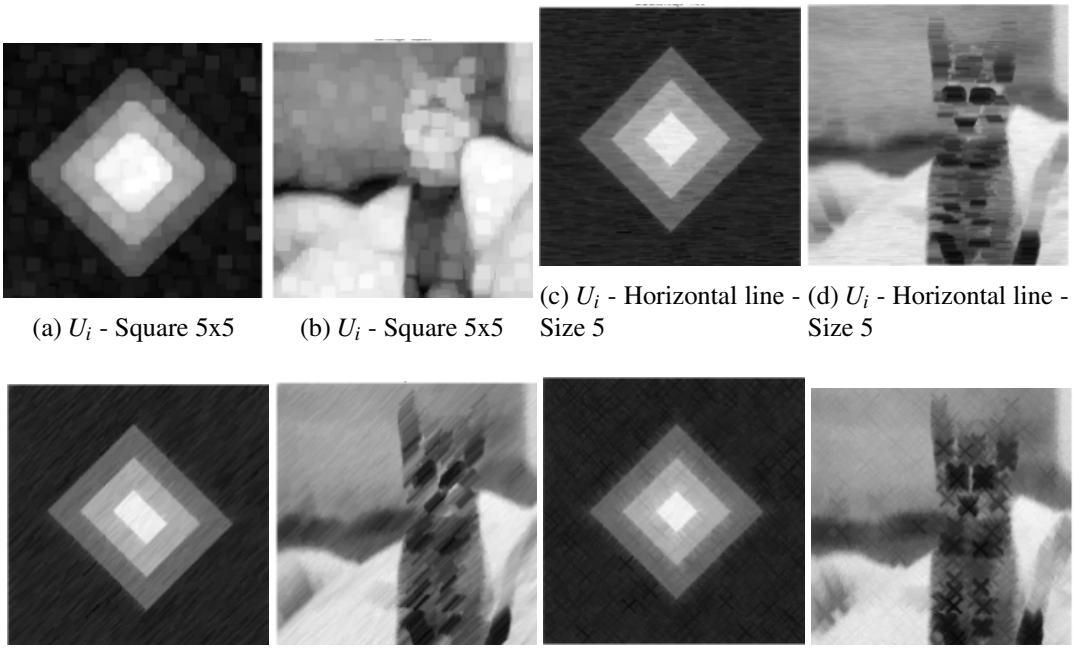
(e) U_i - Horizontal line - Size 5 (f) U_i - Horizontal line - Size 5 (g) U_i - Vertical line - Size 5 (h) U_i - Vertical line - Size 5

(i) U_i - Diagonal right line - Size 5 (j) U_i - Diagonal right line - Size 5 (k) U_i - Diagonal left line - Size 5 (l) U_i - Diagonal left line - Size 5

(m) U_i - Diagonal cross - Size 5 (n) U_i - Diagonal cross - Size 5 (o) U_i - Non Diagonal cross - Size 5 (p) U_i - Non Diagonal cross - Size 5

Figure 3.3: Results of solving 3.1 with different U_i on image 3.1 and 3.2.

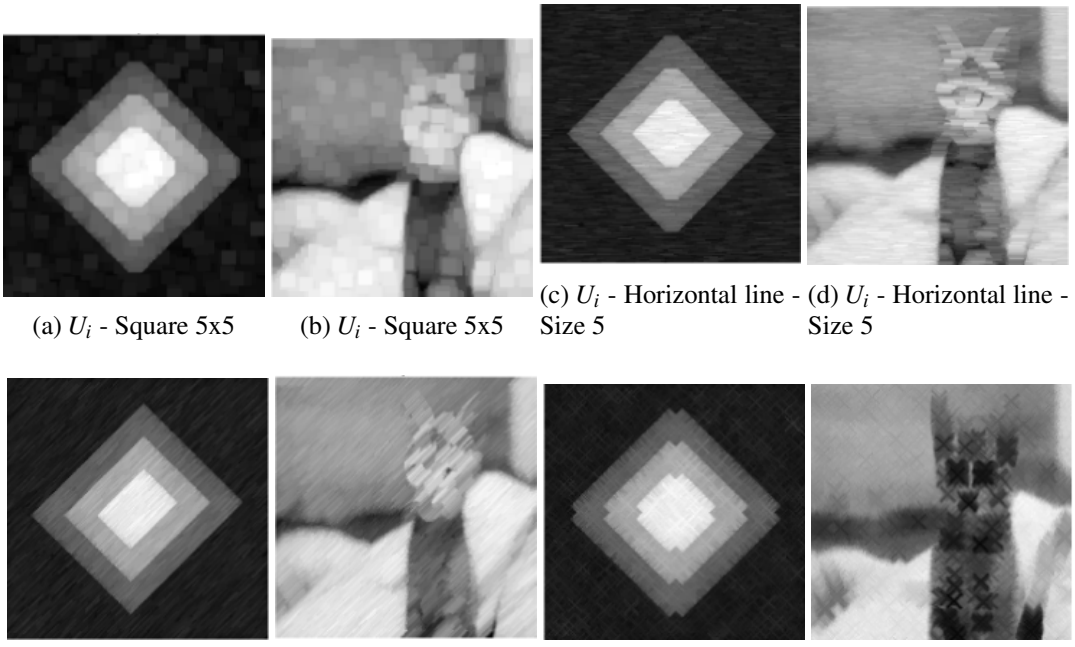
With all of this, a fast Python Implementation of the three propositions formulated in [35] and introduced on previous chapter 3 was possible and examples of the results are illustrated in 3.4 to 3.7.



(a) U_i - Square 5x5 (b) U_i - Square 5x5 (c) U_i - Horizontal line - Size 5 (d) U_i - Horizontal line - Size 5

(e) U_i - Diagonal right line - Size 5 (f) U_i - Diagonal right line - Size 5 (g) U_i - Diagonal cross - Size 5 (h) U_i - Diagonal cross - Size 5

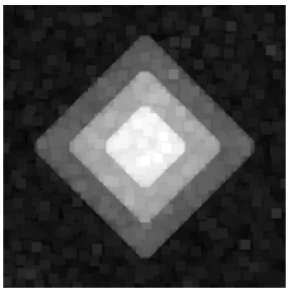
Figure 3.4: Demonstration of the closed form solution given in Proposition 1 with $\alpha^+ = \infty$ and different U_i's on image 4.1 and 4.2



(a) U_i - Square 5x5 (b) U_i - Square 5x5 (c) U_i - Horizontal line - Size 5 (d) U_i - Horizontal line - Size 5

(e) U_i - Diagonal right line - Size 5 (f) U_i - Diagonal right line - Size 5 (g) U_i - Diagonal cross - Size 5 (h) U_i - Diagonal cross - Size 5

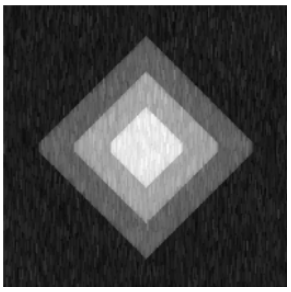
Figure 3.5: Demonstration of the closed form solution given in Proposition 1 implementation with $\alpha^- = -\infty$ and different U_i's on images 3.1 and 3.2.



(a) $\rho = 1^{-6}$ U_i - Square 7x7



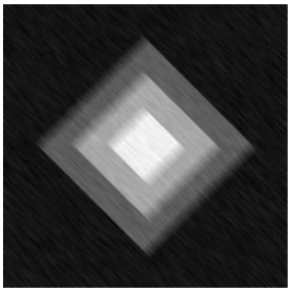
(b) $\rho = 1^{-6}$ U_i - Square 7x7



(c) $\rho = 1/24$ U_i - Vertical Line - Size 7



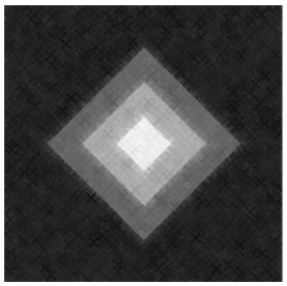
(d) $\rho = 1/24$ U_i - Vertical Line - Size 7



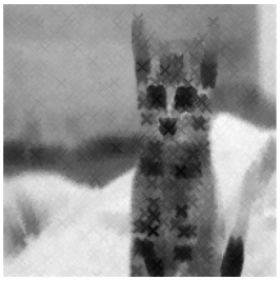
(e) $\rho = 1$ U_i - Diagonal Left Line - Size 7



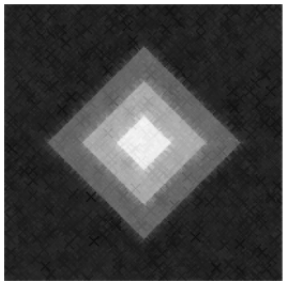
(f) $\rho = 1$ U_i - Diagonal Left Line - Size 7



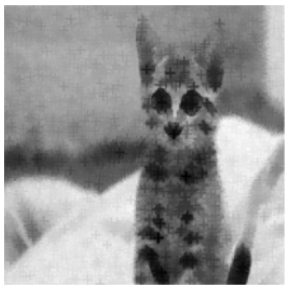
(g) $\rho = 24$ U_i - Diagonal Cross - Size 7



(h) $\rho = 24$ U_i - Diagonal Cross - Size 7



(i) $\rho = 1^6$ U_i - Non Diagonal Cross - Size 7



(j) $\rho = 1^6$ U_i - Non Diagonal Cross - Size 7

Figure 3.6: Demonstration of the closed form solution given in Proposition 2 with different values of ρ and different U_i on images 3.1 and 3.2.



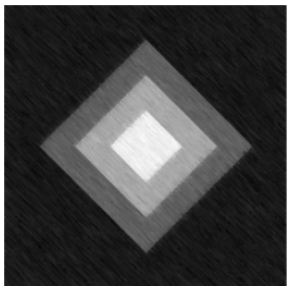
(a) $\rho = 1^{-6}$ U_i - Square 7x7

(b) $\rho = 1^{-6}$ U_i - Square 7x7

(c) $\rho = 1/24$ U_i - Vertical Line - Size 7



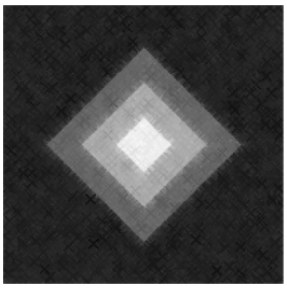
(d) $\rho = 1/24$ U_i - Vertical Line - Size 7



(e) $\rho = 1$ U_i - Diagonal Left Line - Size 7



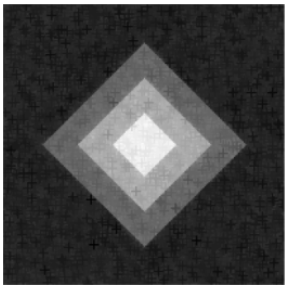
(f) $\rho = 1$ U_i - Diagonal Left Line - Size 7



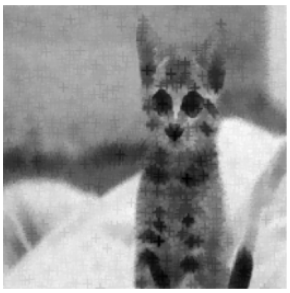
(g) $\rho = 24$ U_i - Diagonal Cross - Size 7



(h) $\rho = 24$ U_i - Diagonal Cross - Size 7



(i) $\rho = 1^6$ U_i - Non Diagonal Cross - Size 7



(j) $\rho = 1^6$ U_i - Non Diagonal Cross - Size 7

Figure 3.7: Demonstration of the closed form solution given in Proposition 3 with different values of ρ and different U_i on images 3.1 and 3.2.

3.2 Methods Developed

In this section, it is presented all the algorithms that we tested for solving 3.1 and their respective explanations and results. Not only that, but also it is presented their validations tests.

This project has as its main goal, the development of efficient solutions for the minimization problem in 3.1. Throughout this thesis, we will only consider 3.1 under the assumption that ϕ_i has the following form, uniformly in i,

$$\phi_i(z_i, p) = (z_i)^p, \quad \text{with} \quad z_i = (\sigma_{\Omega_i}(x_i - y_j - w_{i,j})) \tag{3.2}$$

and p belonging to the interval $]1, \infty[$. With this assumption, the minimization problem 3.1 can be written in a more detailed way as the following:

$$\min_{x=(x_i)_{1\leq i\leq n}\in R^n} \sum_{i=1}^{n} \sum_{j\in U_i} (\sigma_{\Omega_i}(x_i - y_j - w_{i,j}))^p \tag{3.3}$$

With a more detailed analysis of the minimization problem, 3.3 is parametrized by the indices i, j, where i corresponds to the pixel of the image and j to the neighbors of that specific pixel. Based on this, it's possible to conclude that 3.3 is separable on each pixel of the image, so that each output pixel is the solution of the following subproblem:

$$\min_{x_i\in R} \sum_{j\in U_i} F_{ij}(x_i), \quad \text{being} \quad F_{ij}(x_i) = (\sigma_{\Omega_i}(x_i - y_j - w_{i,j}))^p \tag{3.4}$$

Because the whole problem 3.3 can be separated into subproblems for each pixel, we vectorize a solver for the 1D subproblem and apply this in a parallel way.

The algorithms that we will consider to solve 3.3 can be divided in the following groups: root-finding algorithms[26] applied to finding a zero of the first derivative, zeroth-order search[41, 8] applied to the objective function of 3.3 or based on a parallel proximal point algorithm[49].

3.2.1 Root-Finding algorithms for solving 3.3

To determine the solution to our problem, we apply Fermat's rule[11], which states that the minimum of a differentiable function occurs at points where the derivative is zero. Therefore, we need to find the zeros of the derivative of our function $F_{i,j}$, to find the optimal value for the problem 3.3. So, the methods implemented in this section, try to root-finding the following equation:

$$\frac{\partial F_{i,j}}{\partial x_i}(x_i) = \begin{cases} p\alpha_i^-(\alpha_i^-(x_i - y_j - w_{i,j}))^{p-1}, & \text{if} \quad x_i < y_j + w_{ij} \\ 0, & \text{if} \quad x_i = y_j + w_{ij} \\ p\alpha_i^+(\alpha_i^+(x_i - y_j - w_{i,j}))^{p-1}, & \text{if} \quad x_i > y_j + w_{ij} \end{cases} \tag{3.5}$$

corresponding to the first derivative of $F_{ij}(x_i)$ presented in 3.4.

As it is illustrated in image 3.8, the plot of the equation 3.5 for a random pixel of image 3.1 can be characterized as continuous. Following this conclusion, the methods implemented in this section are Bisection Algorithm[56], ITP Algorithm[4, 44] and Newton's Method[63].

Despite the possibility of derivation of 3.4 this same equation can't have a second derivative:

$$\frac{\partial^2 F_{i,j}}{\partial x_i^2} \tag{3.6}$$

at every point, since the first derivative has some vertical sections. In image 3.9, it's possible to visualize the inverse of the second derivative of 3.3, where each spike to zero corresponds to a spike to $+\infty$ on the second derivative.

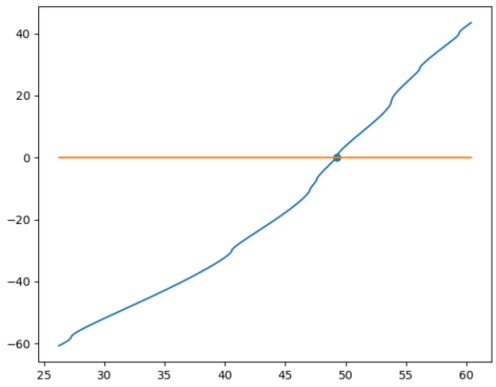


Figure 3.8: First derivative plot (equation 3.5) of random pixel of image 3.1 showing the solution to the optimality condition of 3.5.

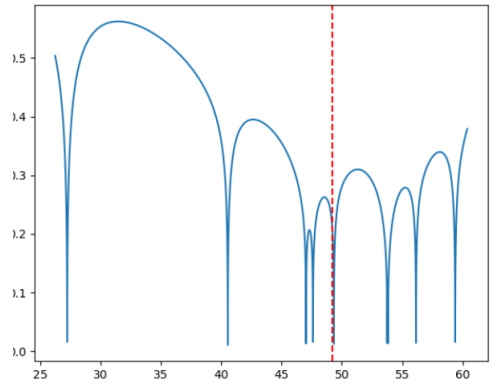


Figure 3.9: Inverse of second derivative plot of random pixel of Fig. 3.8 .

3.2.1.1 Bisection Algorithm

The bisection algorithm is a common method for finding the roots of one-dimensional functions. Its simplicity and robustness make it a popular choice, especially when dealing with continuous functions where the root lies within a known interval.

This algorithm guarantees the existence of a root within an interval where the function takes on values of opposite signs at the endpoints[56]. The algorithm starts with such an interval, iteratively bisects it, and narrows down to a smaller interval containing the root.

We summarize the bisection algorithm below.

1. **Initialization**: Start with an interval $[a,b]$ where $f(a)$ and $f(b)$ have opposite signs, indicating that the root lies within this interval.
2. **Iteration**: Compute the midpoint $c = \frac{a+b}{2}$ of the interval.
3. **Update Interval**: If $f(c) = 0$, the algorithm terminates as c is the root. Otherwise, update the interval:
   - If $f(a)$ and $f(c)$ have opposite signs, set $b = c$.
   - If $f(b)$ and $f(c)$ have opposite signs, set $a = c$.
4. **Convergence**: Repeat the iteration until the interval becomes sufficiently small.

A visual example of this process is illustrated in image 3.10, which is sourced from [18], followed by a pseudocode example.

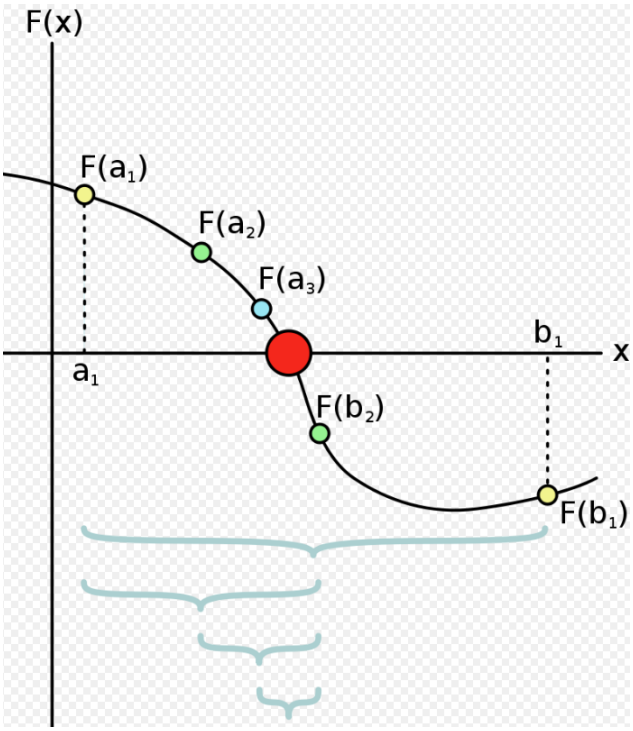


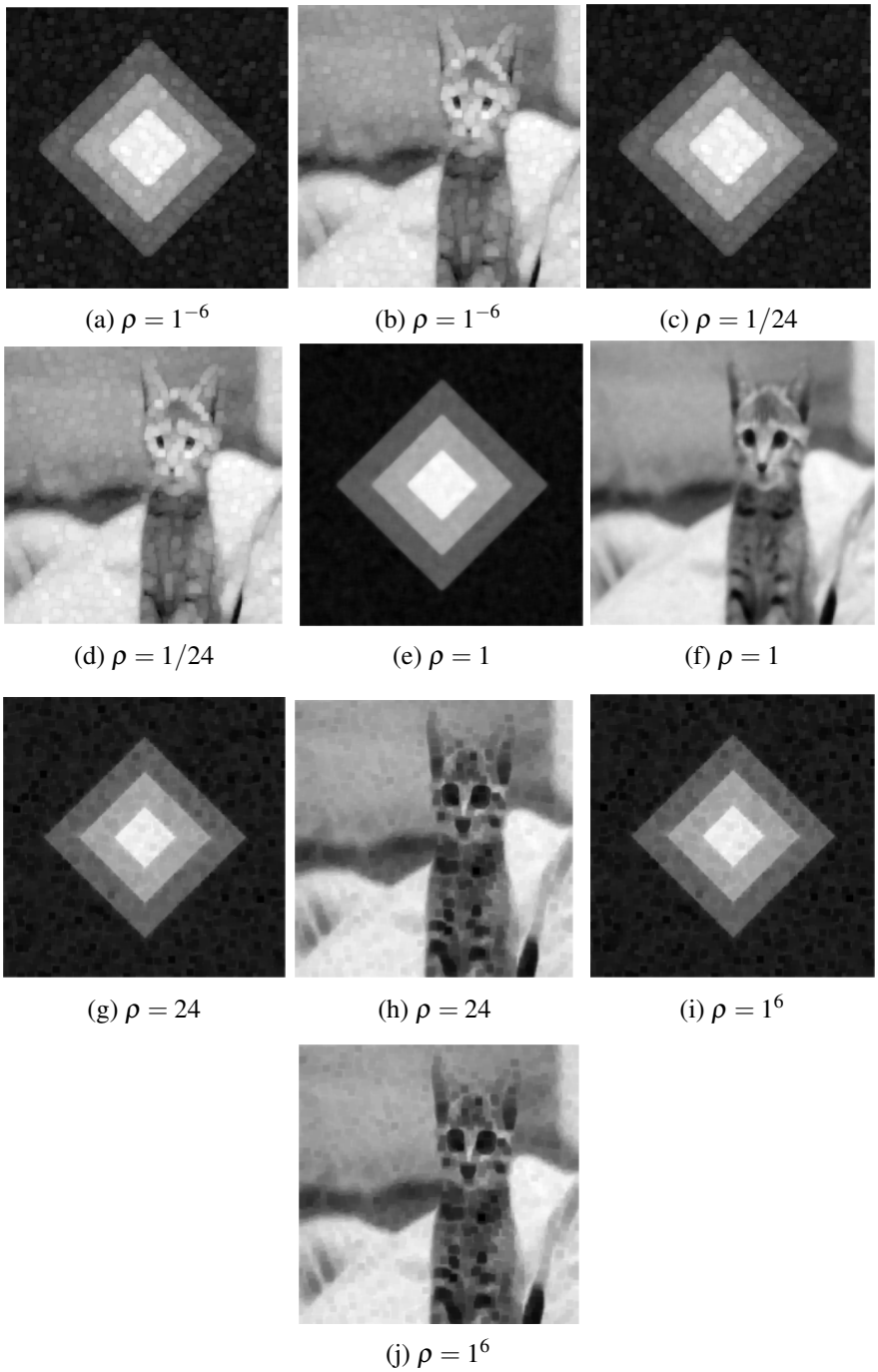
Figure 3.10: Bisection Algorithm's Example

Algorithm 1 Bisection Algorithm

Input:
    f: a continuous function
    a, b: endpoints of the interval $[a, b]$
    tol: tolerance for stopping criterion
    max_iter: maximum number of iterations
Output:
    c: approximate root of f
procedure BISECTION(f, a, b, tol, max_iter)
    if $f(a) \cdot f(b) \geq 0$ **then**
        print("The function must have different signs at endpoints a and b.")
        return None
    end if
    $iter \leftarrow 0$
    while $(b-a)/2 > tol$ **and** $iter < max_iter$ **do**
        $c \leftarrow (a+b)/2$ ▷ Midpoint
        if $f(c) == 0$ **then**
            return c ▷ c is the root
        else if $f(a) \cdot f(c) < 0$ **then**
            $b \leftarrow c$ ▷ The root is in the left half $[a, c]$
        else
            $a \leftarrow c$ ▷ The root is in the right half $[c, b]$
        end if
        $iter \leftarrow iter + 1$
    end while
    return $(a+b)/2$ ▷ Return the midpoint as the best approximation
end procedure

As we discussed in the introduction of this chapter, it was developed a vectorized Bisection based algorithm with the objective of finding the root of the first derivative of the formula 3.3. In this approach, the interval used for every pixel is determined by the overall minimum and maximum values of all the pixels in the image. Specifically, we find the global minimum and maximum pixel values across the entire image. The interval for each pixel is then defined as the range between these two values. This means that the same interval, corresponding to the global range of pixel values, is applied uniformly to all pixels in the image. This algorithm converges to the optimal value of the minimization and evaluation tests are shown next in section 3.3.

Using the bisection algorithm, one can solve 3.3 and compute various variational MM operations and some examples of results are illustrated on Figure 3.11. More examples can be seen on the appendix section.



(a) $\rho = 1^{-6}$ (b) $\rho = 1^{-6}$ (c) $\rho = 1/24$

(d) $\rho = 1/24$ (e) $\rho = 1$ (f) $\rho = 1$

(g) $\rho = 24$ (h) $\rho = 24$ (i) $\rho = 1^{6}$

(j) $\rho = 1^{6}$

Figure 3.11: Demonstration of Bisection root finding implementation while $p = 1.2$ and U_i of Square 7x7 on images 3.1 and 3.2.

3.2.1.2 ITP Algorithm

The ITP (Interpolate Truncate and Project) algorithm is another root-finding algorithm similar to the bisection method[4, 44].

The ITP method builds upon the foundations of bisection method, incorporating elements of each to achieve a superior convergence rate while maintaining optimal worst-case performance. We summarize the ITP algorithm below:

- **Interpolation Step:** The algorithm starts by calculating two points within the current interval: the midpoint, determined via bisection as:

$$x_{1/2} \equiv \frac{a+b}{2} \tag{3.7}$$

  and another point using the *regula falsi*[20] method as:

$$x_f \equiv \frac{bf(a) - af(b)}{f(a) - f(b)} \tag{3.8}$$

  These points serve as initial estimates for the location of the root.

- **Truncation Step:** Perturb the estimator towards the center:

$$x_t \equiv x_f + \sigma\delta \tag{3.9}$$

  where,

$$\sigma \equiv \text{sign}(x_{1/2} - x_f) \tag{3.10}$$

  and

$$\delta \equiv \min\{\kappa_1 |b-a|^{\kappa_2}, |x_{1/2} - x_f|\} \tag{3.11}$$

- **Projection Step:** Project the estimator to minmax[40] interval:

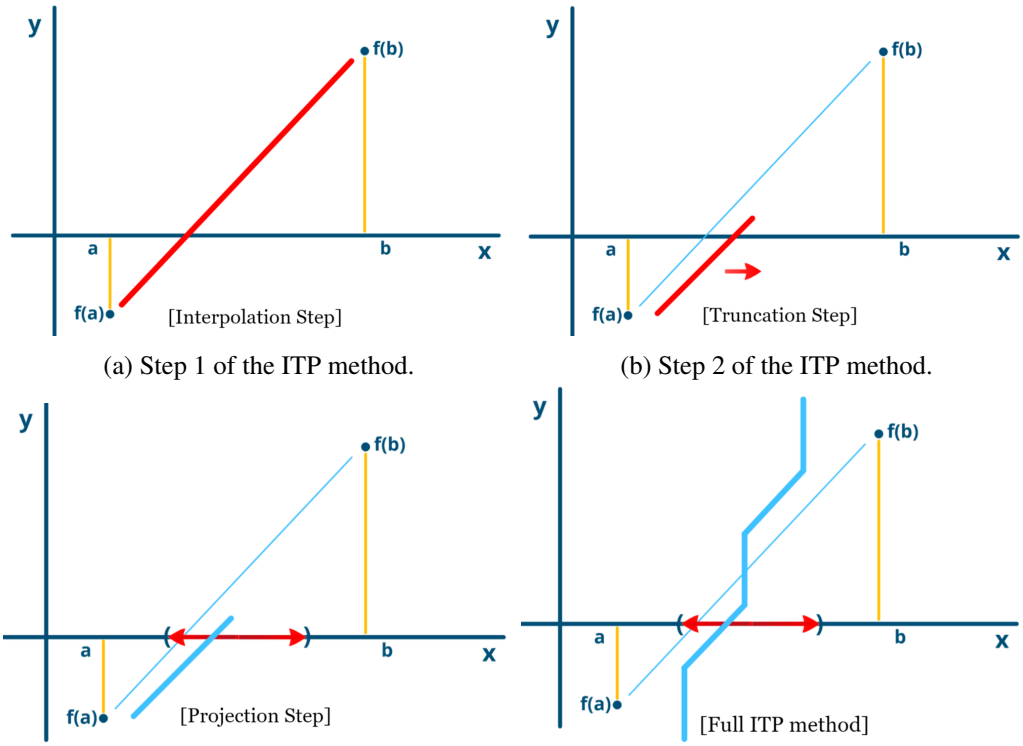
$$x_{\text{ITP}} \equiv x_{1/2} - \sigma\rho_k \tag{3.12}$$

  where,

$$\rho_k \equiv \min\{\varepsilon 2^{n_{1/2}+n_0-j} - \frac{b-a}{2}, |x_t - x_{1/2}|\} \tag{3.13}$$

By iteratively applying these steps, the ITP method progressively narrows down the interval containing the root, ultimately converging to a solution with high accuracy and efficiency. A

visual example of this process is illustrated in image 3.12, which is sourced from [59], followed by a pseudocode example.



(a) Step 1 of the ITP method.

(b) Step 2 of the ITP method.

(c) Step 3 of the ITP method.

(d) Result of the three steps of ITP method.

Figure 3.12: Graphic Examples of ITP Algorithm Steps.

Algorithm 2 ITP Algorithm

Input: $a, b, \varepsilon, \kappa_1, \kappa_2, n_0, f$
Preprocessing:
  $n_{1/2} = \lceil \log_2 \frac{b-a}{2\varepsilon} \rceil$
  $n_{\max} = n_{1/2} + n_0$
  $j = 0$
while $b - a > 2\varepsilon$ **do**
    Calculating Parameters:
      $x_{1/2} = \frac{a+b}{2}$
      $r = \varepsilon 2^{n_{\max} - j} - \frac{b-a}{2}$
      $\delta = \kappa_1 (b-a)^{\kappa_2}$
    Interpolation:
      $x_f = \frac{y_b a - y_a b}{y_b - y_a}$
    Truncation:
      $\sigma = \text{sign}(x_{1/2} - x_f)$
    if $\delta \leq |x_{1/2} - x_f|$ **then**
      $x_t = x_f + \sigma\delta$
    else
      $x_t = x_{1/2}$
    end if
    Projection:
    if $|x_t - x_{1/2}| \leq r$ **then**
      $x_{\text{ITP}} = x_t$
    else
      $x_{\text{ITP}} = x_{1/2} - \sigma r$
    end if
    Updating Interval:
    $y_{\text{ITP}} = f(x_{\text{ITP}})$
    if $y_{\text{ITP}} > 0$ **then**
      $b = x_{\text{ITP}}$
      $y_b = y_{\text{ITP}}$
    else if $y_{\text{ITP}} < 0$ **then**
      $a = x_{\text{ITP}}$
      $y_a = y_{\text{ITP}}$
    else
      $a = x_{\text{ITP}}$
      $b = x_{\text{ITP}}$
    end if
    $j = j + 1$
end while
Output: $\hat{x} = \frac{a+b}{2}$

As discussed in the introduction to this chapter, it was developed a vectorized ITP based algorithm with the objective of finding the root of the first derivative of the formula 3.3. As discussed in the subsection on the bisection method, we have a root bracket given by [a,b] with a = global minimum value of a pixel, b = global maximum value of a pixel. This algorithm converges to the optimal value of the minimization and evaluation tests are shown next section 3.3.

Because the problem 3.3 is strictly convex with a unique minimizer, any algorithm we use to solve it will produce the same output up to numerical precision. However, demonstrations of each method developed are illustrated on appendix.

3.2.1.3 Newton's Algorithm

Newton's method, also known as the Newton–Raphson method, is a powerful root-finding algorithm named after Isaac Newton and Joseph Raphson[63]. This iterative technique is used to produce successively better approximations to the roots (or zeroes) of a real-valued function.

The basic version of Newton's method begins with a real-valued function f, its derivative f', and an initial guess x_0 for a root of f. Given that f satisfies certain conditions and that the initial guess x_0 is sufficiently close to the actual root r, the method iteratively provides improved approximations using the formula:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{3.14}$$

Geometrically, this implies that the point $(x_1, 0)$ is the x-intercept of the tangent line to the graph of f at the point $(x_0, f(x_0))$. In other words, the improved guess x_1 is the unique root of the linear approximation of f at the initial guess x_0.

The process is iteratively applied, updating the approximation with each step as follows:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{3.15}$$

This iteration continues until the sequence $\{x_n\}$ converges to a sufficiently precise value. Formally, if the initial guess x_0 is chosen such that $|x_0 - r| < \delta$ for some small $\delta > 0$, the sequence generated by Newton's method will converge to the root r. This ensures that the error decreases as the iterations progress, given that the initial guess x_0 is within the convergence radius δ. To summarize the iterative process of Newton's method:

1. **Initial Guess**: Start with an initial guess x_0. The closer this guess is to the actual root, the better the method performs. In the absence of a good guess, the intermediate value theorem might help narrow down a reasonable interval.

2. **Compute Tangent**: Approximate the function f by its tangent line at x_n.

3. **Update Approximation**: Compute the x-intercept of this tangent line to get a new approximation with equation 3.3

4. **Iterate**: Repeat the process with the new approximation until the result is accurate enough.

A visual example of this process is illustrated in image 3.13, which is sourced from [42], followed by a pseudocode example.

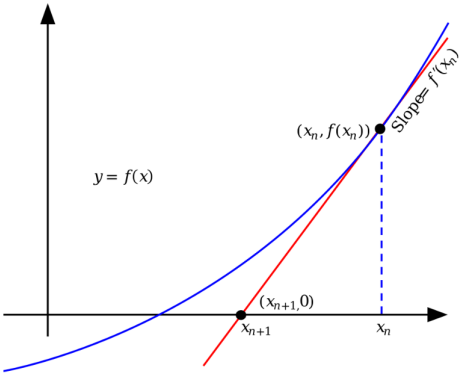


Figure 3.13: Newton's Method Visual Example.

Algorithm 3 Newton's Method

Input: Initial guess x_0, tolerance ε, maximum iterations N, function f, derivative f'
Output: Approximate root x
$x \leftarrow x_0$
for $i = 0$ to $N-1$ **do**
    $f_x \leftarrow f(x)$
    $f'_x \leftarrow f'(x)$
    if $|f_x| < \varepsilon$ **then**
        return x
    end if
    if $f'_x = 0$ **then**
        Error: Derivative is zero. No convergence.
        return x
    end if
    $x \leftarrow x - \frac{f_x}{f'_x}$
end for
Error: Maximum iterations reached. No convergence.
return x

In contrast to the bisection and ITP algorithms, it was possible to develop a vectorized Newton's Method based algorithm with the objective of finding the root of the first derivative of the formula 3.3, but this implementation does not have theoretical convergence guarantees because our function does not satisfy the hypotheses of the theorems for newton's method[63]. This statement implies that the function that has been used possesses at least one derivative, which is a prerequisite for applying mathematical methods like Newton's Method[63], where derivatives are essential for determining convergence and making iterative improvements. The objective of this implementation is to locate the root of the first derivative pertaining to the minimization problem described in equation 3.3. This means that for Newton's Method to work, it would be necessary to be possible to derivate twice the equation 3.3, what is not possible at every point as explained previously on image 3.9.

3.2.2 Zeroth order Algorithms

Zeroth-order algorithms[54] are optimization methods that do not require any derivative information about the objective function. They rely solely on function evaluations to guide the search for the optimum. These methods are sometimes called "black-box" optimization methods because they treat the objective function as a black box that can be queried for function values at various points.

Zeroth-order algorithms are particularly useful when the objective function is complex, non-differentiable, or when derivative information is difficult or expensive to obtain. By only using function evaluations, these methods provide a versatile and robust approach to optimization.

This section explores the implementation of two zeroth-order algorithms: Golden-section Search [41] and Ternary Search [8]. Both methods are designed to find the minimum of the function on 3.3 and, consequently, find the optimal value for the minimization problem.

3.2.2.1 Golden-section Search Algorithm

The Golden-section Search[41] is a technique for finding a minimum of a function within a specified interval. For a convex function with a minimum in an interval, this method will always converge to the minimizer. In cases where the interval contains multiple extrema, including possible extrema at the interval boundaries, the method will converge to one of these points. If the only minimum within the interval is at a boundary, the algorithm will converge to that boundary point[51].

Mathematically, this can be expressed as:

Let $f(x)$ be a function defined on the interval $[a,b]$. If $f(x)$ has a local minimum at $x = a$ or $x = b$, and there are no other local minima of $f(x)$ within the interval (a,b), then any optimization algorithm aiming to find the minimum of $f(x)$ within $[a,b]$ will converge to $x = a$ or $x = b$, depending on which boundary contains the minimum[51].

The Golden-section Search operates by successively narrowing the range of values within the specified interval. The technique derives its name from the fact that the algorithm maintains the function values for four points such that the three interval widths are in the ratio $\varphi : 1 : \varphi$, where φ is the golden ratio. These ratios are maintained throughout each iteration, ensuring maximal efficiency.

To illustrate the technique, it's necessary to evaluate the function $f(x)$ at three points: x_1, x_2, and x_3, where $f(x_1) > f(x_2) < f(x_3)$. This configuration ensures that a minimum lies between x_1 and x_3. The next step is to evaluate the function at a new point, x_4, chosen inside the largest interval, typically between x_2 and x_3.

If $f(x_4) > f(x_2)$, the minimum lies between x_1 and x_4, updating our triplet to x_1, x_2, and x_4. Conversely, if $f(x_4) < f(x_2)$, the minimum lies between x_2 and x_3, updating our triplet to x_2, x_4, and x_3. In either case, the new search interval is guaranteed to contain the minimum.

To ensure the intervals are proportionally reduced, the Golden-section Search algorithm chooses the point x_4 such that the resulting intervals maintain the golden ratio $\varphi \approx 1.618$. Mathematically,

this means setting the intervals to be equal, ensuring that $b = a + c$, where b is the larger interval and a and c are the smaller intervals.

The golden ratio φ ensures that the spacing between evaluation points remains proportional[41], avoiding situations where points are too close together and ensuring the interval width shrinks consistently by the same proportion in each step. The golden ratio is defined as:

$$\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618 \tag{3.16}$$

The algorithm terminates when the interval $\Delta x = x_4 - x_1$ is sufficiently small, indicating an accurate estimation of the minimum. A visual example of this process is illustrated in image 3.14, which is sourced from [45], followed by a pseudocode example.

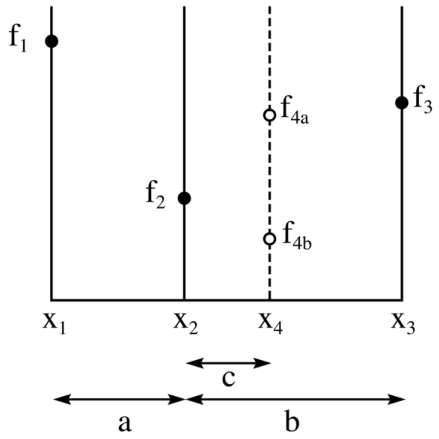


Figure 3.14: Diagram of a Golden-section Search. The initial triplet of x values is x_1, x_2, x_3. If f(x_4) = f_{4a}, the triplet x_1, x_2, x_4 is chosen for the next iteration. If f(x_4) = f_{4b}, the triplet x_2, x_4, x_3 is chosen.

Algorithm 4 Golden-section Search

Input: Function f, interval $[a,b]$, tolerance τ
Output: Approximate minimum x
$\varphi \leftarrow \frac{1+\sqrt{5}}{2}$ ▷ Golden ratio
$c \leftarrow b-(b-a)/\varphi$
$d \leftarrow a+(b-a)/\varphi$
$f_c \leftarrow f(c)$
$f_d \leftarrow f(d)$
while $|b-a| > \tau(|c|+|d|)$ **do**
    if $f_c < f_d$ **then**
        $b \leftarrow d$
        $d \leftarrow c$
        $c \leftarrow b-(b-a)/\varphi$
        $f_d \leftarrow f_c$
        $f_c \leftarrow f(c)$
    else
        $a \leftarrow c$
        $c \leftarrow d$
        $d \leftarrow a+(b-a)/\varphi$
        $f_c \leftarrow f_d$
        $f_d \leftarrow f(d)$
    end if
end while
return $(b+a)/2$

Following the requirements of this zeroth order algorithm, it was possible to develop a vectorized Golden-section Search based algorithm with the objective of finding the minimum of the equation shown in 3.3. In this approach, the first interval follows the same strategy as explained in Bisection approach section. This algorithm converges to the optimal value of the minimization and evaluation tests are shown next section 3.3.

Like in the ITP approach section about the ITP method, although this algorithm computes the same result as the others, in appendix it is possible to see some demonstrations of the Golden-section Search approach.

3.2.2.2 Ternary Search Algorithm

The Ternary Search algorithm[8] is another technique in computer science for finding the minimum of a function.

Assume we are looking for a minimum of $f(x)$ and we know the minimum lies somewhere between A and B. For the algorithm to be applicable, there must exist some value x such that:

- For all a,b with $A \leq a < b \leq x$, we have $f(a) > f(b)$, and
- For all a,b with $x \leq a < b \leq B$, we have $f(a) < f(b)$.

Let $f(x)$ be a function on some interval $[l,r]$. Take any two points m_1 and m_2 in this segment: $l < m_1 < m_2 < r$. Then there are three possibilities:

- If $f(m_1) > f(m_2)$, then the required minimum cannot be located on the left side $[l,m_1]$. Therefore, it makes sense to look only in the interval $[m_1,r]$.
- If $f(m_1) < f(m_2)$, then the situation is similar to the previous case, but symmetrically reversed. Now, the required minimum cannot be on the right side $[m_2,r]$, so the search continues in the segment $[l,m_2]$.
- If $f(m_1) = f(m_2)$, then the search should be conducted in $[m_1,m_2]$. This case can be handled similarly to either of the previous two scenarios for simplicity in implementation. Eventually, as the segment length becomes sufficiently small, the process can be terminated.

The points m_1 and m_2 are chosen as follows:

$$m_1 = l + \frac{r-l}{3} \tag{3.17}$$

$$m_2 = r - \frac{r-l}{3} \tag{3.18}$$

These points m_1 and m_2 divide the interval $[l,r]$ into three equal parts, allowing for the systematic narrowing down towards the location of the minimum.

A visual example of this process is illustrated in image 3.15 (image got from[64]) followed by a pseudocode example.

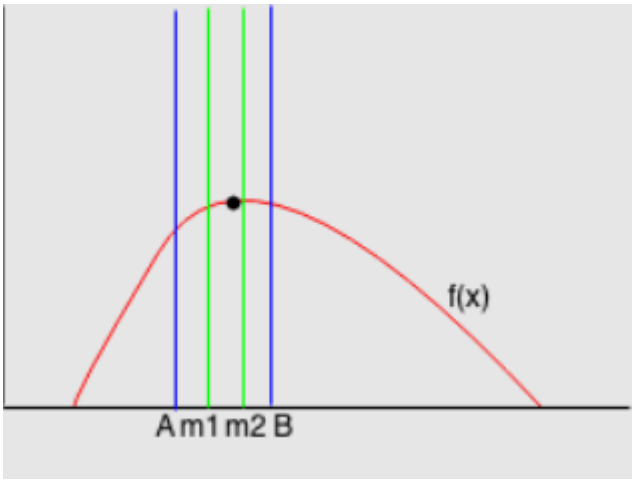


Figure 3.15: Graphic Example of Ternary Search Algorithm

Following the requirements of the method explained in this section, it was possible to develop a vectorized Ternary Search based algorithm with the objective of finding the minimum of the equation shown in 3.3. In this approach, the first interval follows the same strategy as other explained earlier. This algorithm converges to the optimal value of the minimization and evaluation tests are shown next section 3.3.

Algorithm 5 Ternary Search

Input: Function f, interval $[l, r]$, tolerance τ
Output: Approximate maximum x
while $|r - l| > \tau$ **do**
    $m1 \leftarrow l + (r - l)/3$
    $m2 \leftarrow r - (r - l)/3$
    if $f(m1) < f(m2)$ **then**
        $l \leftarrow m1$
    else
        $r \leftarrow m2$
    end if
end while
return $(l + r)/2$

Like in the Golden-section Search approach section about the Golden-section Search method, although this algorithm computes the same result as the others, in appendix it is possible to see some demonstrations of the Ternary Search approach.

3.2.3 Parallel Proximal Point Algorithm

Although proximal methods are not new work as have been contributions written on various aspects of this topic over the years, as Lemaire[37] in 1989 or Iusem[30] in 1994, their integration into the signal-processing domain is relatively recent. Nonetheless, their adoption has gained rapid traction due to their efficacy and versatility.

The primary objective of this section is to introduce proximity operators, elucidating their fundamental properties and exploring in more detail the solution implemented based on parallel proximal.

3.2.3.1 Concept of Proximal Algorithms

A proximal algorithm is a versatile optimization tool offering a practical approach to solving convex optimization problems[16]. At its essence, a proximal algorithm relies on the proximal operators of the objective terms to efficiently navigate the optimization landscape. For instance, consider the proximal minimization algorithm, which iteratively minimizes a convex function f by repeatedly applying its proximal operator to an initial point x_0.

3.2.3.2 Implementation

This algorithm is an implementation of the following Parallel Proximal Point Algorithm (PPXA) with the goal of minimizing the problem in 3.3.

The differentiability of the objective function in 3.3 is ensured since the function ϕ_i and the support function σ_{Ω_i} are both differentiable themselves. As introduced in 3.1, the objective function is defined as:

$$\min \sum_{i=1}^{n} \sum_{j \in U_i} \phi_i(\sigma_{\Omega_i}(x_i - y_j - w_{i,j})) \tag{3.19}$$

where U_i represents a nonempty subset of V, $w_{i,j}$ is a vector of shift parameters, and $\phi_i : [0, +\infty[\to R$ is a strictly increasing lower-semicontinuous convex function. Additionally, σ_{Ω_i} denotes the support function of the closed interval Ω_i, given by:

$$\sigma_{\Omega_i}(v) = \begin{cases} \alpha_i^- v & \text{if } v < 0 \\ 0 & \text{if } v = 0 \\ \alpha_i^+ v & \text{if } v > 0 \end{cases} \tag{3.20}$$

where $\alpha_i^- = \inf \Omega_i$ and $\alpha_i^+ = \sup \Omega_i$.

As mentioned before, the minimization problem in 3.3 can be vectorized over all pixels in a image or all nodes in a graph, so taking this procedure in consideration the equation to minimize using the algorithm PPXA ends up being the following one:

$$\min_{x_i \in R} \sum_{j \in U_i} F_{ij}(x_i), \quad \text{where} \quad F_{ij}(x_i) = (\sigma_{\Omega_i}(x_i - y_j - w_{i,j}))^p \tag{3.21}$$

The *prox* computed for each j corresponds and follows the basic operations with proximal algorithms and converges to this formula:

$$prox_{SF_{i,j}}(x_i) = (\mathrm{Id} + S\nabla F_{ij})^{-1}(x_i) \tag{3.22}$$

Let S be a scale, Id the identity matrix and ∇F_{ij} the gradient of function F_{ij}. The gradient of F is computed using the following closed formula:

$$\frac{\partial f}{\partial v} = \begin{cases} pS\alpha_i^-(\alpha_i^- v^-)^{p-1} & \text{if } v < 0 \\ 0 & \text{if } v = 0 \\ pS\alpha_i^+(\alpha_i^+ v^+)^{p-1} & \text{if } v > 0 \end{cases} \tag{3.23}$$

Based on this and in the formula of 3.22 is possible to get the formula of *prox* for each i:

$$prox_{SF_i}(x_i) = \begin{cases} v_i^-, \text{ such that } v_i^- + pS\alpha_i^-(\alpha_i^- v_i^-)^{p-1} - x_i = 0 & \text{if } x_i < 0 \\ 0 & \text{if } x_i = 0 \\ v_i^+, \text{ such that } v_i^+ + pS\alpha_i^+(\alpha_i^+ v_i^+)^{p-1} - x_i = 0 & \text{if } x_i > 0 \end{cases} \tag{3.24}$$

Notice that to compute the function $prox_{SF_i}(x_i)$ it is necessary to solve a root-finding problem, which must be done iteratively.

However, this is not yet complete, since the main goal is not getting the value of the *prox* for a specific i but for a specific pair of i, j, corresponding two a pair of pixels, a specific pixel and its neighbor. So, to achieve that goal and using the following property of proximal operations:

$$prox_{SF_{i,j}}(x_i) = prox_{SF_{ij}}(x_i - y_i - w_{ij}) + y_i + w_{ij} \tag{3.25}$$

the final formula to calculate the prox for a specific pair i, j is the following:

$$prox_{SF_{i,j}}(x_i) = \begin{cases} v_i^- + y_i + w_{ij}, v_i^- + pS\alpha_i^-(\alpha_i^- v_i^-)^{p-1} - x_i + y_i + w_{ij} = 0 & \text{if } x_i < y_i + w_{ij} \\ y_i + w_{ij} & \text{if } x_i = y_i + w_i j \\ v_i^+ + y_i + w_{ij}, v_i^+ + pS\alpha_i^+(\alpha_i^+ v_i^+)^{p-1} - x_i + y_i + w_{ij} = 0 & \text{if } x_i > y_i + w_{ij} \end{cases} \tag{3.26}$$

Based on this formula, it is possible to minimize 3.21, computing the *prox* for each pair of neighbors inside the neighborhood decided[48].

$$prox_{SF_{i,j}}(x_i) = \begin{cases} v_i^- + y_j + w_{ij}, & \text{if } \quad v_i^- + pS\alpha_i^-(\alpha_i^- v_i^-)^{p-1} - x_i + y_j + w_{ij} = 0, \quad x_i < y_j + w_{ij} \\ y_j + w_{ij}, & \text{if } \quad x_i = y_j + w_{ij} \\ v_i^+ + y_j + w_{ij}, & \text{if } \quad v_i^+ + pS\alpha_i^+(\alpha_i^+ v_i^+)^{p-1} - x_i + y_j + w_{ij} = 0, \quad x_i > y_j + w_{ij} \end{cases} \tag{3.27}$$

A pseudocode example of this algorithm is illustrated on image 3.16, which is sourced from [48].

Fix $\varepsilon \in]0,1[$, $\gamma > 0$, $(\omega_i)_{1\le i\le m} \in]0,1]^m$ such that
$\sum_{i=1}^m \omega_i = 1$, $y_{1,0} \in \mathbb{R}^N, \ldots, y_{m,0} \in \mathbb{R}^N$
Set $x_0 = \sum_{i=1}^m \omega_i y_{i,0}$
For $n = 0, 1, \ldots$
  For $i = 1, \ldots, m$
    $p_{i,n} = \mathrm{prox}_{\gamma f_i/\omega_i} y_{i,n}$
  $p_n = \sum_{i=1}^m \omega_i p_{i,n}$
  $\varepsilon \le \lambda_n \le 2 - \varepsilon$
  For $i = 1, \ldots, m$
    $y_{i,n+1} = y_{i,n} + \lambda_n (2p_n - x_n - p_{i,n})$
  $x_{n+1} = x_n + \lambda_n (p_n - x_n)$.

Figure 3.16: Parallel proximal algorithm (PPXA).

Following the parallel proximal algorithm explanation, it was developed a vectorized PPXA based algorithm with the objective of finding the minimum of the equation shown in 3.21. This algorithm converges to the optimal value of the minimization and evaluation tests are shown next section 3.3.

With this implementation was possible to generate various variational MM operations and demonstrations of results are illustrated on appendix.

3.3 Comparison of algorithms

Validation testing is a crucial aspect of any project to ensure that the methods and algorithms used produce accurate and reliable results. We are primarily interested in validating the performance of our methods in terms of execution time. The results of these operations are highly specific and unique to the problem at hand, making it difficult to compare them directly to a ground truth. All the tests here in this section are made with image 3.1.

3.3.1 Performance Testing

Before we begin to describe the performance testing of the various algorithms we have implemented to solve 3.3, we note that we have verified that all the algorithms applied to a given image produce the same output pixels, i.e., each algorithm is correctly solving 3.3 (see appendix).

Now we will present the performance of the various algorithms to solve 3.3 on some example images. We will consider the execution time as a function of the size of the image and as a function of the radius of the neighborhood U_i, respectively.

3.3.1.1 Increasing Image Size

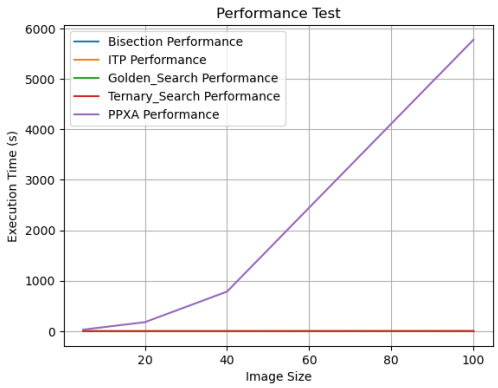


Figure 3.17: Performance results with PPXA method using U_i Square 3x3.

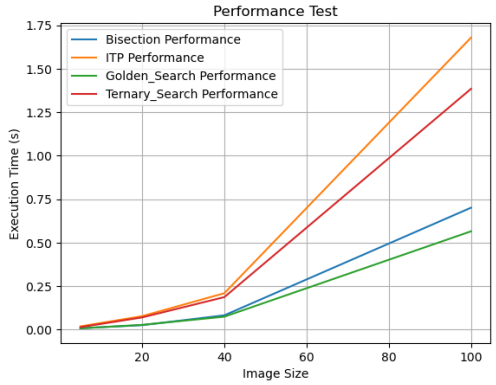


Figure 3.18: Performance results without PPXA method using U_i Square 3x3.

From these results it's clear that PPXA is too slow to converge compared to the other algorithms, taking more than 100 times as long in terms of execution time.

3.3.1.2 Increasing ST Size

The tests in this section were developed always using a U_i square.

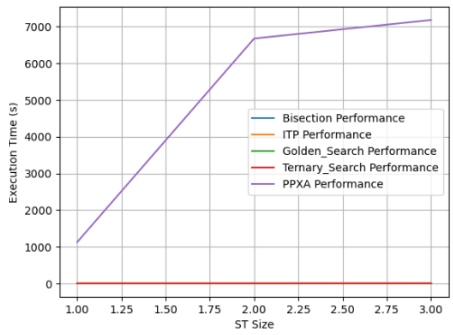


Figure 3.19: Performance results with PPXA method using U_i Square 3x3 on a image 200x200.

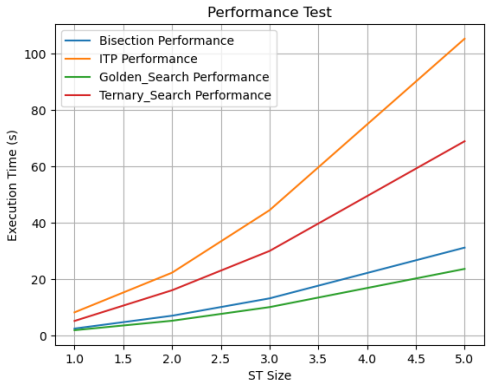


Figure 3.20: Performance results without PPXA method using U_i Square 3x3 on a image 200x200.

From these results, it's possible to conclude the same as in the previous performance testing: it's clear that PPXA is too slow to converge compared to the other algorithms, taking more than 100 times as long in terms of execution time.

3.4 Pytorch Implementation

PyTorch[47] is a versatile machine learning library that originated from the Torch library and is widely used in applications such as computer vision and natural language processing. Originally developed by Meta AI and now under the Linux Foundation umbrella, PyTorch[47] has become one of the leading machine learning frameworks alongside TensorFlow[1]. It offers free and open-source software, released under the modified BSD license, making it accessible to developers worldwide.

One of PyTorch's key features is its tensor computing capabilities, which are similar to NumPy[43, 27] but with strong GPU acceleration. This enables efficient computation on large-scale datasets and complex neural network architectures. Additionally, PyTorch provides a tape-based automatic differentiation system, facilitating the implementation of deep neural networks and enabling seamless training and optimization.

Overall, PyTorch's rich set of features, active community, and ongoing development efforts make it a powerful tool for researchers, developers, and practitioners in the field of machine learning and artificial intelligence.

3.4.1 PyTorch Performance Testing

In order to be able to apply variational MM operations to large-scale datasets, like graphs representing point-clouds or large images, we extended our implementation of algorithms to solve 3.3 to include GPU implementations using the PyTorch framework[47].

PyTorch enables us to parallelize algorithms for solving equation 3.3 more efficiently by leveraging GPU acceleration. This capability significantly speeds up the computation process, making it ideal for handling large-scale optimization tasks and neural network training where computation speed is crucial.

3.5 Performance of GPU implementation

Tests were realized to compare approaches and achieve a conclusion on the upgrade that the usage of Pytorch and GPU could have in terms of performance.

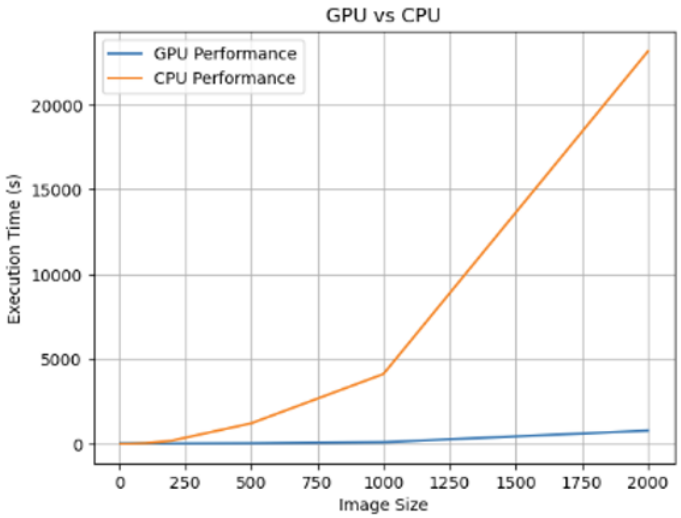


Figure 3.21: PyTorch vs Non-Pytorch Performance changes while increasing image size - PPXA Approach.

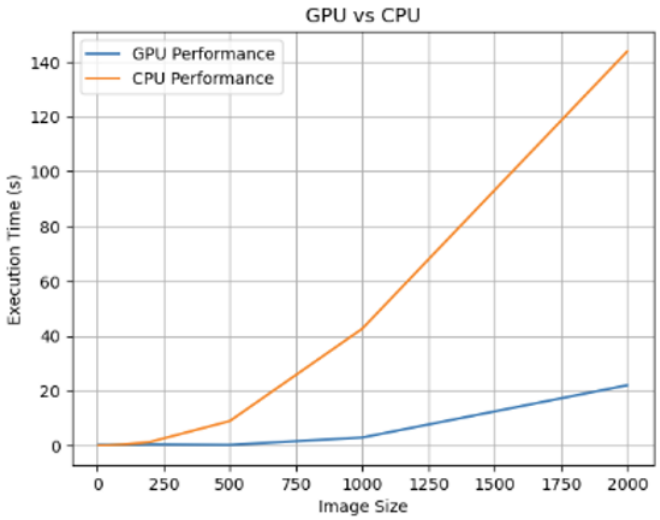


Figure 3.22: PyTorch vs Non-Pytorch Performance changes while increasing image size - Bisection Approach.

In this comparative study, it was evaluated the performance of the Proximal Proximal Point Algorithm (PPXA) approach and Bisection approach (the worst method developed in terms of performance and one of the bests, respectively), implemented in both PyTorch and non-PyTorch environments across a dataset of eight images with varying sizes. The aim was to assess the efficiency and effectiveness of the PyTorch-based approach compared to the traditional implementation.

Across all image sizes, ranging from small to large, the PyTorch implementation consistently demonstrated superior performance metrics compared to its non-PyTorch counterpart. Specifically, we observed significant reductions in computation time and resource utilization when executing the algorithm using PyTorch.

Furthermore, the PyTorch implementation showcased enhanced scalability, effectively handling larger image sizes without compromising performance. This scalability is attributed to PyTorch's efficient tensor computations and optimization techniques, which leverage GPU acceleration where available.

Overall, these findings suggest that adopting PyTorch for implementing PPXA algorithm offers clear benefits in terms of performance, scalability, and development efficiency. These advantages make PyTorch a compelling choice for researchers and practitioners seeking to optimize their image processing algorithms for real-world applications.

3.6 Practical Application and Image Processing Illustrations

The methods explored and developed in this chapter are capable of solving 3.3, with many of them being parallelizable in a vectorized way. This results in very efficient solvers, except for the PPXA algorithm, that allow one to apply the variational MM operations to large images in a short amount of time. This was further emphasized by porting the algorithms to PyTorch[47] and using a GPU to maximize the vectorization of the algorithms discussed, such that one can

apply a variational MM operation to a 1750x1750 image in under 20 seconds. By manipulating the parameters and characteristics of the convex function involved, the formulation allows for the creation of customized operations that can mimic and go beyond the vanilla MM operations presented on the previous chapter. The adaptability of this framework is demonstrated through image examples, showcasing the versatility of the variational MM operations in handling diverse spatial neighborhoods and different parameters.

3.7 Conclusions

In this chapter, several iterative algorithms to solve 3.1 for computing variational Mathematical Morphology (MM) operations were presented. From algorithmic implementations to practical demonstrations, this chapter provides a comprehensive overview of the methodologies employed, their theoretical underpinnings, and their practical outcomes. These variational MM operations are defined through the support functions of closed real intervals and increasing convex functions, making them applicable in diverse domains such as graph processing. The inherent flexibility of these operators, achieved by adjusting parameters, allows for the creation of a diverse class of operations, offering significant versatility in their applications.

Importantly, the study has established that the introduced variational MM operations encompass standard morphological dilation and erosion as special cases, thereby providing a unified perspective on these traditional operations. This unification not only enhances our theoretical understanding but also opens new avenues for practical applications within the scope of this chapter.

Furthermore, the approaches developed in this chapter hold potential due to their open-source nature. By making these algorithms accessible to the community, we facilitate collaborative development and adoption across various fields. This democratization of tools enables researchers and practitioners to tailor variational MM operations to specific needs, fostering innovation and advancing the state of the art in image processing and beyond.

Performance testing conducted in this chapter underscores the practical viability of our approaches. By rigorously benchmarking against existing methods, we demonstrate the efficiency and scalability of our open-source implementations. These results underscore our commitment to delivering high-performance solutions that meet the demands of real-world applications, ensuring that variational MM operations can be computed accurately and efficiently.

The potential applications of these variational MM operations are explored next chapter. They extend to strong graph, image and mesh denoising, as well their applicability inside a segmentation neural network as a variational morphological layer. This functionality makes them valuable tools for addressing image processing needs and tailoring solutions to different scenarios.

Chapter 4

Applications of variational MM operations to graph denoising and neural networks

4.1 Graphs and mesh denoising

4.1.1 Introduction

The significance of high-quality 3D models has grown a lot in various fields, such as computer graphics, medical imaging, virtual reality, and computer-aided design. Graphs and meshes are typically used to represent these models, capturing their geometric and topological information.

However, a common challenge in processing these representations is the presence of noise, which can greatly reduce the quality and accuracy of the models. Denoising plays a crucial role in eliminating noise while preserving important features like edges and corners. Effective denoising improves the visual fidelity and structural integrity of 3D models, making them more suitable for subsequent analysis and applications. Traditional denoising techniques often struggle to find a balance between noise removal and feature preservation, particularly in complex models with intricate details.

Variational morphological operations provide a possible solution to address this challenge. By leveraging the principles of mathematical morphology, these operations can effectively distinguish between noise and significant features within graph and mesh structures. This capability is invaluable for applications that require high precision and accuracy in 3D modeling.

This section explores the application of variational morphological operations to address the issue of 3D denoising in graphs and meshes. We delve into the methodology for applying these operations to 3D structures and present experimental results showcasing their effectiveness. The use of variational morphological operations in this context not only enhances the quality of 3D models but also highlights the versatility and power of these operations in solving practical problems. Subsequent sections provide detailed explanations of the techniques and algorithms used,

the experimental setup, and the results of applying variational morphological operations to graph and mesh denoising. Through this exploration, we aim to demonstrate the potential of these operations to significantly enhance the quality of 3D models, leading to more accurate and reliable applications in various fields.

4.1.1.1 Conversion Implementation

The objective in this section is use the methods from chapter 3 and implement them on graphs and mesh3D. To be possible to do that, this project needed an effective way of converting graphs and 3D meshes. Consequently, that was the first step in this implementation, be possible to transform from graph to the neighborhood 2D array/tensor. In case of 3D mesh, the conversion steps are from mesh to graph and from graph to array/tensor. All the reverse processes of turning back to graph or mesh were achieved too. This way, the implementation developed is capable of working with any graph or mesh as it is illustrated next image.

In terms of computational efficiency and performance, this implementation aims to optimize execution times and resource utilization. The following images 4.1 and 4.2 illustrate examples of the implemented approach in action (the models used have as source[23] and[62] respectively). Notably, the method achieves impressive results, computing operations on a 4719-node graph (teapot) in less than 1 second and handling a 17858-node graph (virus) in approximately 2 seconds. These benchmarks underscore the effectiveness and scalability of the algorithm in processing large-scale data sets efficiently.

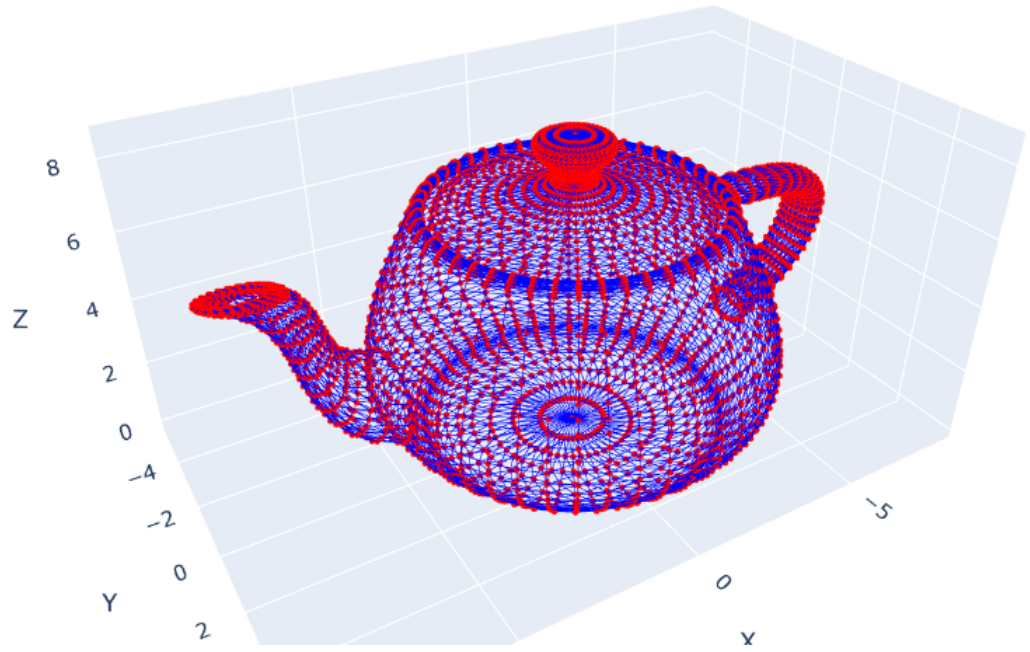


Figure 4.1: Mesh/Graph Implementation Teapot example

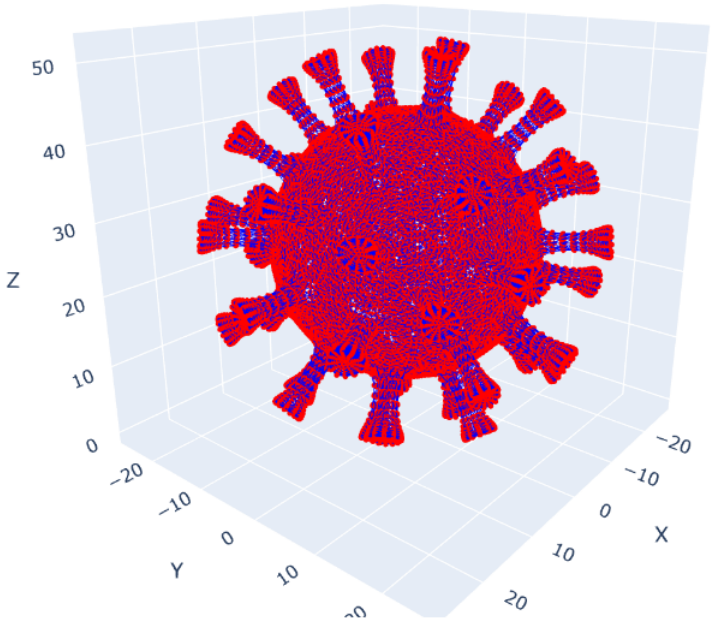


Figure 4.2: Mesh/Graph Implementation Virus example

4.1.2 NetworkX Library: A Comprehensive Tool for Graph and Network Analysis

NetworkX[24] is a versatile Python package designed for creating, manipulating, and analyzing complex networks, making it essential for researchers and developers across various fields.

4.1.2.1 Graph signal denoising

Value graph denoising involves applying techniques in graph-based data structures to smooth or denoise the values associated with each node. This process is essential in scenarios where node values are susceptible to noise or fluctuations due to measurement errors, data transmission issues, or other external factors. The denoising process aims to enhance the accuracy and reliability of node values, thereby improving the quality and robustness of subsequent analyses or operations performed on the graph. Here are some critical characteristics of value denoising:

- **Node Value Smoothing:** This process entails the application of algorithms that adjust the nodes' values based on the values of their neighbors. The objective is to minimize abrupt changes or inconsistencies in node values, resulting in a smoother and more coherent distribution across the graph.
- **Application Areas:** Value graph denoising finds applications in various fields, including signal processing, social network analysis, and biological networks. For example, denoising the measured values at each sensor node in sensor networks can eliminate random noise, leading to more accurate environmental monitoring.

In brief, value graph denoising improves the quality of node-associated values in a graph by smoothing and reducing noise, thus enhancing the reliability of data-driven decisions and analyses across diverse applications.

4.1.2.2 Graphs 2D

The following images show the variational MM operation applied on a 2D example graph, created by NetworkX[24].

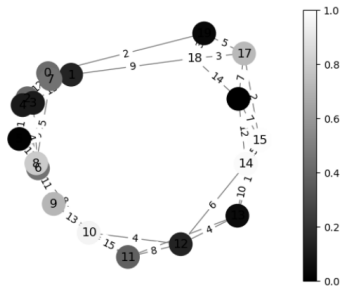
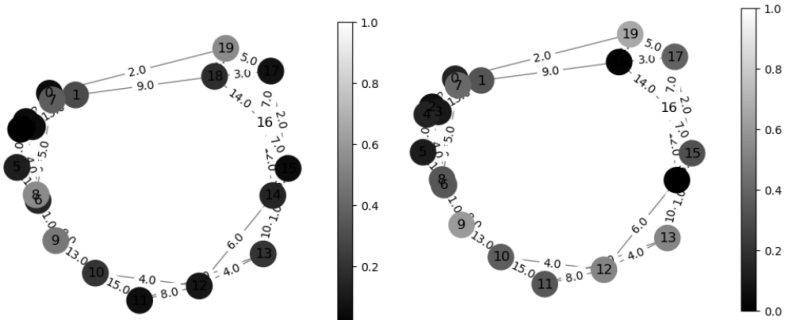
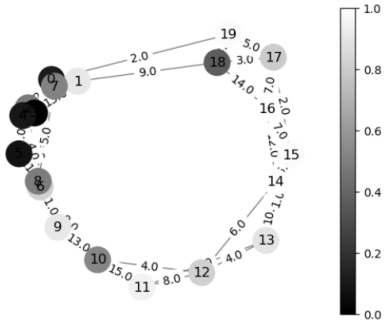


Figure 4.3: 2D Graph generated by NetworkX



(a) $\rho = 1e6$ (b) $\rho = 1$



(c) $\rho = 1e-6$

Figure 4.4: Demonstration of the application of Bisection approach on graph represented in Fig. [**?**] with $p = 1.4$

4.1.2.3 Graphs 3D

The following images show the variational MM operation applied on a 3D example graph, created by NetworkX[24].

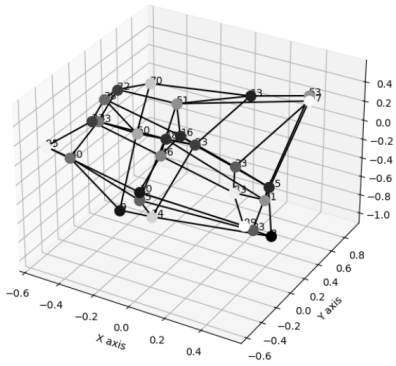
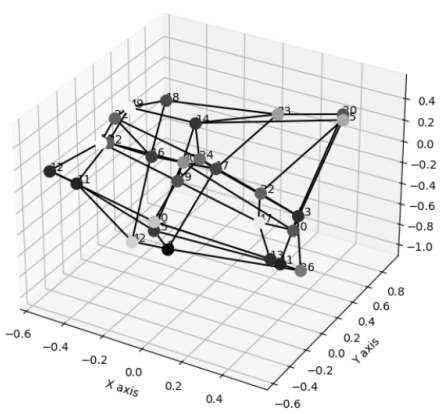
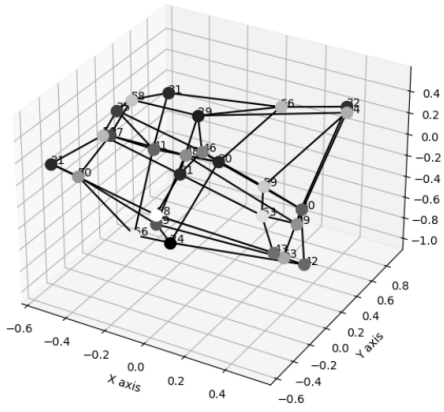


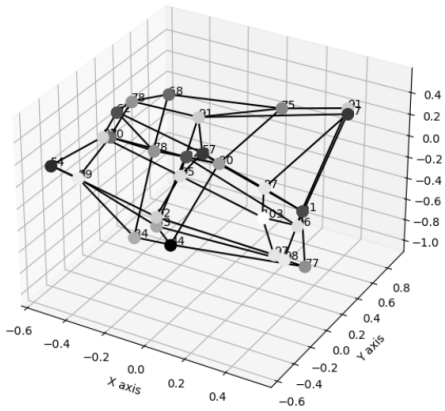
Figure 4.5: 3D Graph generated by NetworkX



(a) $\rho = 1e6$



(b) $\rho = 1$



(c) $\rho = 1e-6$

Figure 4.6: Demonstration of the application of Bisection approach on graph represented in Fig. with $p = 1.4$

4.1.3 Mesh 3D and Trimesh Library: An Overview

Trimesh[19] is a Python library designed for efficient loading, manipulation, and analysis of triangular meshes, especially those with watertight surfaces. Compatible with Python 3.7[61] and above, it provides a comprehensive Trimesh object for streamlined 3D model handling.

- **Mesh Visualization and Previewing:** Allows easy previewing using Pyglet or Jupyter notebooks via three.js, facilitating quick inspection of 3D models.
- **Advanced Mesh Analysis and Manipulation:** Provides tools for calculating face adjacencies, angles, vertex defects, cross-sectional slicing, and repairing mesh issues like triangle winding errors and filling holes.
- **Geometric and Topological Operations:** Supports convex hulls, boolean operations, meshes, and generating volume meshes, and can split meshes based on face connectivity.
- **Caching and Performance Optimization:** Utilizes internal caching of computed values to enhance performance, avoiding redundant calculations.
- **Utility Functions and Change Tracking:** Provides various utility functions for efficient mesh data handling and manipulation.

In summary, Trimesh[19] is a powerful library for working with 3D triangular meshes in Python. Its support for various file formats, advanced analysis tools, and robust visualization capabilities make it indispensable for developers, researchers, and engineers working with 3D models.

4.1.3.1 Graph signal denoising

As previously elaborated, value graph denoising focuses on smoothing or eliminating the noise associated with each node's values within a graph. Extending these principles to 3D meshes involves several critical operations:

- **Node Value Smoothing:** Within 3D meshes, this procedure entails applying the algorithms formulated previous chapter to adjust the values tied to vertices, considering the values of neighboring vertices. This strategy minimizes abrupt changes or inconsistencies, ensuring a smoother and more consistent value distribution throughout the mesh.
- **Application Areas:** The application of value graph denoising in 3D meshes proves especially beneficial in sectors such as computer graphics, medical imaging, and 3D modeling. Specifically, denoising vertex values in 3D models can significantly enhance texture mapping and surface evenness, ultimately elevating visual fidelity.

By instituting value graph denoising within 3D meshes, the quality and dependability of vertex-associated values receive substantial enhancement, a pivotal factor for conducting high-fidelity 3D analysis and applications.

This method developed can be applied to model files, making it easy to use. The resulting transformed graph after applying the variational MM operation is outputted into an interactive HTML file. This is illustrated in the following images using the same teapot model[23] as previously:

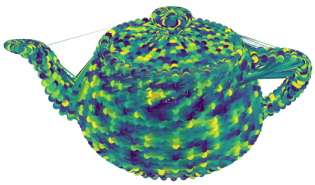
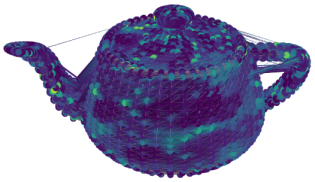
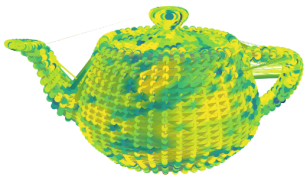


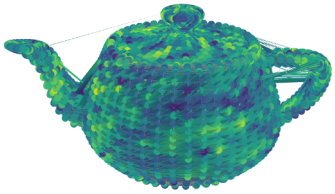
Figure 4.7: 3D Teapot Mesh



(a) $\rho = 1e6$



(b) $\rho = 1e-6$



(c) $\rho = 1$

Figure 4.8: Results of the application of Bisection on mesh represented in Fig. with $p = 1.5$

4.1.4 Rank Filter

Rank filters are a powerful tool in signal processing, utilized to manage and clean data by sorting a set of n input values and selecting the kth value as the output[57]. This selection is guided by a single parameter, ranging between 0 and 1, which determines the value of k for any given n. For instance, a rank of 1 corresponds to taking the maximum value from the set, while a rank of 0.5 represents a median filter.

These filters are particularly effective in eliminating impulsive noise, which is common in various signal processing applications such as 2-photon laser scanning microscopy (2P-LSM)[50]. By limiting the influence of individual input values on the overall signal estimation, rank filters help in reducing noise without significantly affecting the integrity of the original data.

Unlike other noise reduction techniques that may bluntly remove high spatial frequencies, rank filters are adept at preserving sharp edges and other crucial details. This quality is vital for maintaining the fidelity of human visual perception and enhancing the accuracy of automated object detection systems. Consequently, rank filters serve as a versatile and robust method for improving signal quality while retaining important structural features in the data.

Other application for the Variational MM operations explained and implemented in this project would be Rank filters on a 3D mesh. With the variational MM operations it's possible to achieve a variational Rank Filter implementation, corresponding to a variational erosion followed the rank filter. In the following images 4.9 to 4.11 are illustrated results of applying the variational Rank Filter to a adaption of the teapot 3D mesh used previously, where the goal is denoising a 20% noisy mesh in order to recover the yellow line, which correspond to nodes that were given a specific value to have a big contrast with the rest of the mesh.

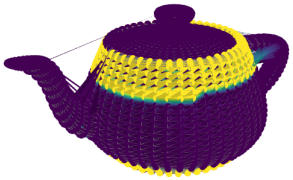


Figure 4.9: Generated 3D Teapot mesh

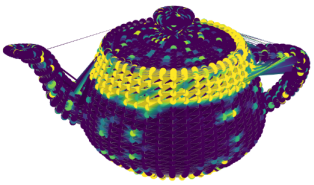
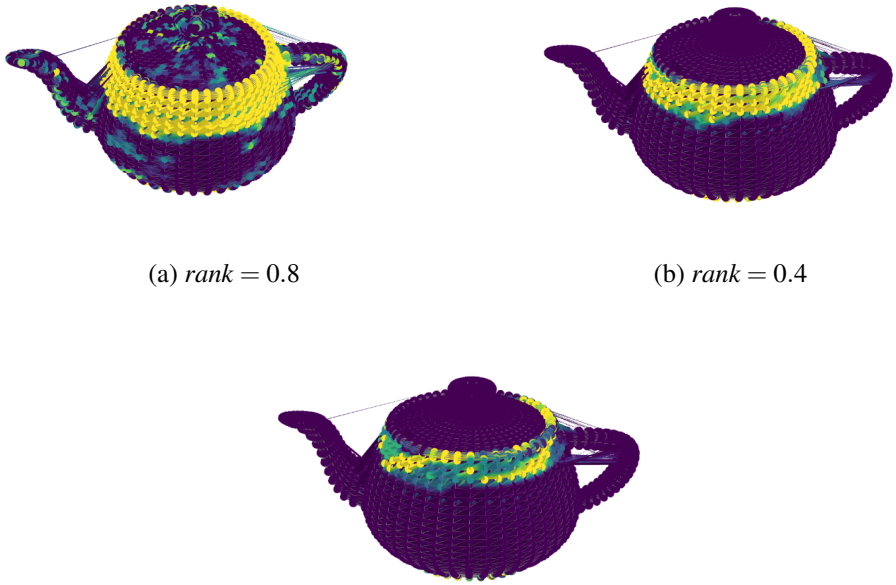


Figure 4.10: Noisy 3D Teapot mesh



(a) $rank = 0.8$

(b) $rank = 0.4$

(c) $rank = 0.1$

Figure 4.11: Results of the application of Rank Filter on graph represented in Fig. with $p = 1.4$

With this study case, it's possible to see different outcomes depending on the value of *rank*. The results are better with lower values of rank. With $rank = 0.8$, it's possible to see the denoising of the yellow line, however it kept most noise for the rest of the mesh object. The best result is with $rank = 0.4$, having almost full recovery of the target and all noise from the object removed. Meanwhile, for $rank = 0.1$ it's possible to see a more poor result on recovering the target if compared with the previous case, despite a similar denoising result on the rest of the mesh as the best case.

Variational rank filters hold significant potential for image processing in medical environments. Their ability to effectively eliminate noise while preserving crucial details makes them potentially useful in medical imaging applications, where clarity and precision are paramount. By integrating variational rank filters, medical professionals might enhance image quality, facilitating better diagnosis and treatment planning.

4.1.5 Conclusion

In the experiments conducted with the application of variational denoising and rank filters on graphs 2D/3D, and 3D meshes, enhancements were observed in their utility across various domains, as it's seen in the last experiment, graph signal denoising.

In medical imaging, for example, these techniques could improve the accuracy of anatomical models derived from MRI or CT scans[22], aiding in surgical planning and treatment simulations.

In engineering and architecture, variational denoising in meshes ensures that structural designs are robust and reliable, minimizing errors in fabrication and construction phases.

Moreover, these advancements facilitate more accurate data-driven decision-making in fields such as environmental science, finance, and social network analysis. By enhancing the fidelity of data representation and analysis, variational techniques contribute to better insights, predictions, and innovations.

In summary, the implementations of variational denoising techniques in graphs and meshes, alongside variational rank filters, represent advancements in computational and analytical methodologies. These techniques not only improve data quality but also expand the capabilities of applications across diverse fields, ultimately driving innovation and progress in scientific and industrial domains.

4.2 Neural Networks and Variational MM layers

4.2.1 Introduction

Neural networks represent an influential class of machine learning models that emulate the decision-making processes of the human brain, replicating the interconnected nature of biological neurons to identify patterns, evaluate options, and draw conclusions. These models consist of layers of nodes or artificial neurons, including an input layer, one or more hidden layers, and an output layer. Each node in a neural network is linked to others through associated weights and thresholds. Activation of a node occurs when its output surpasses a specified threshold, leading to the transmission of data to the subsequent layer in the network.

Training data plays a pivotal role in enabling neural networks to learn and enhance their accuracy over time. Once optimized, these networks become potent tools in the fields of computer science and artificial intelligence, facilitating swift data classification and clustering. Tasks such as speech recognition and image recognition, which would require hours if performed manually by human experts, can be executed in mere minutes. Google's search algorithm is one of the most renowned demonstrations of a neural network.

Neural networks, referred to as artificial neural networks (ANNs) or simulated neural networks (SNNs), represent a subset of machine learning and serve as the cornerstone of deep learning models. Leveraging an array of mathematical principles and algorithms, these networks are adept at processing and analyzing data.

4.2.1.1 How Do Neural Networks Work?

Within a neural network, each node functions akin to an independent linear regression model, comprising input data x_i, weights w_i, a bias (or threshold) b, and an output. The governing formula for a single node is as follows:

$$\sum w_i x_i + b = w_1 x_1 + w_2 x_2 + w_3 x_3 + b \tag{4.1}$$

The output of the node is determined by the ReLU (Rectified Linear Unit) activation function:

$$\text{output} = \max(0, \sum w_i x_i + b) \tag{4.2}$$

In the context of ReLU[7], the activation function outputs the result of the linear combination $\sum w_i x_i + b$ if it is positive, and outputs 0 otherwise. This function is crucial in neural networks as it introduces non-linearity, allowing the network to learn complex patterns and relationships in the data.

4.2.2 Learn parameters from minimization problem

4.2.2.1 Introduction to Implicit and Explicit Layers

In neural networks, layers can generally be classified into two categories: implicit layers[31, 29] and explicit layers[32]. Understanding these concepts is fundamental to comprehending the structure and functionality of complex neural network architectures.

4.2.2.2 Implicit Layers

Implicit layers, often referred to as implicit function layers[31, 29], are defined by functions that determine the output by solving an equation involving both the input and the output. Instead of providing a direct formula, implicit layers determine the output through a relationship or constraint that must be satisfied. For instance, given an implicit function $f(x,y) = 0$, where y is the output and x is the input, the output y is computed such that this equation holds true. These layers are particularly useful for problems where the relationships between inputs and outputs are best described by constraints or equilibrium conditions.

Implicit layers allow neural networks to handle scenarios where the relationship between inputs and outputs is implicitly defined by complex equations or constraints, such as optimization problems or physics-based simulations. They extend the network's capability beyond standard layers by incorporating constraints that shape the output based on the input, ensuring solutions satisfy intricate conditions without explicit formulation.

4.2.2.3 Explicit Layers

Explicit layers provide a direct mapping from inputs to outputs through well-defined functions. Examples of explicit layers include convolutional layers, fully connected layers, and activation functions in neural networks. These layers apply straightforward, often linear or nonlinear, transformations directly to the input data. The output of an explicit layer is calculated by applying a specific function to the input, such as:

$$y = f(x) \tag{4.3}$$

where y is the output, x is the input, and f is the explicit function defining the layer.

Explicit layers are typically easier to implement and understand due to their straightforward nature[32]. They are widely used in various neural network architectures for tasks such as feature extraction, classification, and regression.

Understanding the distinction between implicit and explicit layers allows for better design and optimization of neural network models, particularly when dealing with complex data dependencies and constraints.

In the context of neural networks, while a concept as implicit layers[31, 29] offer a potent methodology for addressing complex issues by applying the implicit function theorem[34], there are instances where adopting an explicit layer strategy yields greater efficiency and simplicity. This is especially true when parameter $p = 2$ permits exploiting a closed-form solution for the minimization problem.

With $p = 2$, the output of the variational MM reduces to the closed form solution given in 2.3.1.2 as it can be seen in section 2.14, facilitating the derivation of an explicit solution. It becomes possible to differentiate the solution directly, bypassing the need for iterative solvers or reliance on the implicit function theorem. This explicit layer methodology capitalizes on the closed-form solution, enhancing computational efficiency and simplifying implementation within a neural network framework.

Adopting this explicit solution enables the direct computation of gradients, facilitating straightforward backpropagation during neural network training. Capitalizing on this closed-form solution ensures that the variational MM operation is piecewise linear in this case and so there is no problem to differentiate it in PyTorch.

In conclusion, when $p = 2$, utilizing an explicit layer approach with a closed-form solution for the minimization problem simplifies the differentiation process, boosts computational efficiency, and facilitates seamless integration into neural network architectures. This methodology proves particularly beneficial in scenarios dominated by quadratic forms, offering a viable alternative to the implicit layer approach.

So, using this explicit layer concept in the next section we will show how to use variational MM operations as a layer in a neural network. We will show not only that is possible to learn a set of parameters from the minimization problem given a dataset of images as used to test the impact of variational MM layers on neural networks.

4.2.3 Introduction to Variational Morphological Layers in Neural Networks

In the evolving field of neural networks, particularly for tasks involving image classification, traditional layers such as convolutional have long constituted the foundation of model architectures. These layers excel in capturing spatial hierarchies and reducing dimensionality. However, the introduction of variational morphological layers represents a significant advancement, offering unique advantages that could substantially enhance the performance of neural networks in classification tasks.

4.2.3.1 Variational Morphological Layers

Inspired by the principles of mathematical morphology, a theory dedicated to analyzing and processing geometrical structures, variational morphological layers are tailored to focus on the morphological attributes of input data, such as shapes and contours. This focus renders them optimally suited for classification tasks, where the precise delineation of object boundaries is paramount.

4.2.3.2 Comparison to convolution layers

A comparison with convolutional layers is imperative to evaluate the efficacy of variational morphological layers, especially in the context of neural network performance on tasks such as image classification.

4.2.3.3 Convolutional Layers

These layers, applying linear filters to the input data, are pivotal in capturing local patterns like edges and textures. While instrumental for feature extraction, they may need more capacity to capture global shape properties as efficiently as morphological layers.

4.2.3.4 Evaluating Performance on MNIST classification

Considering the application of variational morphological layers in a neural network designed for the MNIST classification task highlights their impact. The MNIST dataset[36], with its simplicity and apparent shape-based features, serves as an optimal benchmark for evaluating classification performance. The dataset contains 60,000 training images and 10,000 testing images of handwritten digits. The images are grayscale and each has a size of 28x28 pixels. These images are normalized to fit into a 28x28 pixel bounding box and are anti-aliased, introducing grayscale levels. MNIST[36] is widely used for training and testing in the field of machine learning, especially for image classification tasks. The structure and content of the MNIST dataset make it efficient for tasks involving image recognition and classification, offering a standard and widely recognized benchmark for performance assessment. The following image 4.12 show an example for all digits presented on MNIST dataset and work as input in the neural networks for every experiment.

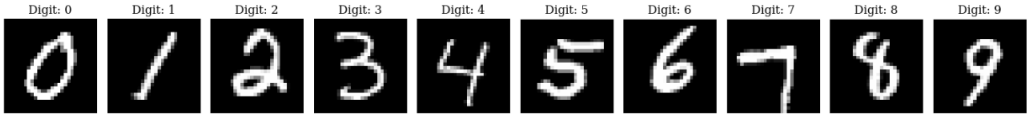


Figure 4.12: Example of all digits presented in MNIST dataset

4.2.4 Neural Network Study Case - 8 Square Filters

The experiment involves training a Convolutional Neural Network (CNN) and other 9 different version where the first convolutional layer is replaced for a variational MM layer, which is building

upon 8 Square kernels all with the same size (3x3) on the MNIST dataset to classify handwritten digits. This section provides an overview of the model architecture, data preparation steps, training process, evaluation metrics, and visualizations of the learned features.

4.2.4.1 Model Architecture

Both CNN architecture and all variational MM neural networks used for this experiment are designed to process grayscale images of size 28×28. These archictures consist in:

4.2.4.2 CNN Architecture

- Two convolutional layers (‘Conv1‘ and ‘Conv2‘) with ReLU activation followed by max pooling layers.
- Dropout layer with probability $p = 0.5$ to prevent overfitting.
- Two fully connected layers (‘Fully Connected Layer 1‘ and ‘Fully Connected Layer 2‘) with ReLU activation.
- Output layer (‘Output‘) with logits for classification into 10 classes (digits 0-9).

4.2.4.3 Variational MM neural networks Architecture

- One variational MM layer with ReLU activation followed by a max pooling layer.
- One convolutional layer (‘Conv2‘) with ReLU activation followed by a max pooling layers.
- Dropout layer with probability $p = 0.5$ to prevent overfitting.
- Two fully connected layers (‘Fully Connected Layer 1‘ and ‘Fully Connected Layer 2‘) with ReLU activation.
- Output layer (‘Output‘) with logits for classification into 10 classes (digits 0-9).

4.2.4.4 Data Preparation

The MNIST dataset is used, which consists of $60,000$ training images and $10,000$ test images of handwritten digits. The following transformations are applied to each image:

- Conversion to grayscale.
- Conversion to PyTorch tensors.
- Normalization based on MNIST dataset statistics.

A subset of 1024 samples is used for both training and testing to expedite experimentation. Using a consistent subset size for training and testing significantly reduces the computation power needed.

4.2.4.5 Training

The model is trained using:

- Cross-entropy loss function for multi-class classification.
- Adam optimizer with a learning rate of 0.01.
- Learning rate scheduler to adjust learning rate based on validation loss.
- Training for 35 epochs with batch size 512.

During training, the model's performance metrics (loss and accuracy) are monitored on both training and validation sets.

4.2.4.6 Evaluation

After training, the model is evaluated on the test set to assess its generalization ability. Evaluation metrics include:

- Confusion matrix to visualize predicted vs actual class distributions.
- Classification report showing precision, recall, and F1-score for each class.

These metrics provide insights into the model's performance across different digits.

4.2.4.7 Visualizations

Several visualizations are generated to analyze the model's behavior and learned features:

- Training history plots showing loss and accuracy over epochs.
- Visualization of learned filters in the first convolutional layer.
- Output of learned filters applied to example images.

These visualizations aid in understanding how the model processes and distinguishes between different digits.

The structure of the CNN neural network is illustrated in the following pseudocode:

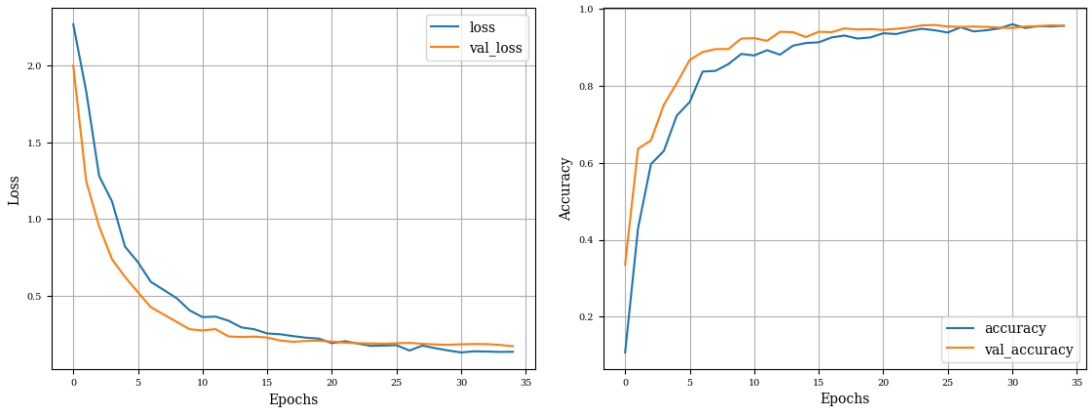
Algorithm 6 CNNModel class definition

Input: Number of classes *num_classes*
Output: CNN model with specified architecture
class CNNModel(**nn.Module**) {
function __init__($num_classes = 10$) {
super(CNNModel, self).__init__()
$self.conv1 = nn.Conv2d(1, 8, kernel_size = (3,3), padding = 1)$ *[r]Changed input channels to 1
$self.pool1 = nn.MaxPool2d(kernel_size = (2,2), stride = (2,2))$
$self.conv2 = nn.Conv2d(8, 8, kernel_size = (3,3), padding = 1)$ *[r]Changed input channels to 8
$self.pool2 = nn.MaxPool2d(kernel_size = (2,2), stride = (2,2))$
$self.flatten = nn.Flatten()$
$self.dropout = nn.Dropout(p = 0.5)$
$self.fc1 = nn.Linear(8 * 7 * 7, 128)$
$self.fc2 = nn.Linear(128, num_classes)$
}
function forward(x) {
$x = \text{torch.relu}(self.conv1(x))$
$x = self.pool1(x)$
$x = \text{torch.relu}(self.conv2(x))$
$x = self.pool2(x)$
$x = self.flatten(x)$
$x = self.dropout(x)$
$x = \text{torch.relu}(self.fc1(x))$
$x = self.fc2(x)$
return x
}
}

To study the effectiveness of variational MM operations as neural network layers, we will replace the first convolution layer ('conv1') with one of the following 9 variational MM operations, one for each experiment and test whether the accuracy obtained in the classification problem improves. These variational operations include the less complex ones that were already mentioned and explained previously: **variational Dilation**, **variational Erosion**, **Opening**[39] and **Closing**[39]. Not only those but also a **Gradient**[52, 66], corresponding to the difference between the dilation and the erosion on a target image, a **Internal Gradient**[52], corresponding to the difference between the erosion and the original image, a **External Gradient**[52], corresponding to the difference between the dilation and the original image. Furthermore, this experiment also includes **Top Hat Closing**[25] and **Top Hat Opening**[25], corresponding, respectively, to the difference between the original image and the closing/opening. All these operations are considered variational since they are all compositions of variational MM operations. Each of following results are illustrated with a explanation about the initialization of the layer's parameters, the result Confusion Matrix, the filters learned, the output filters and the graphics for loss and accuracy. All confusion

matrix plots can be seen on appendix section.

4.2.4.8 Convolution Layer



(a) Loss Convolution Plot (b) Accuracy Convolution Plot

Figure 4.13: Loss and Validation Accuracy of Convolution Layer



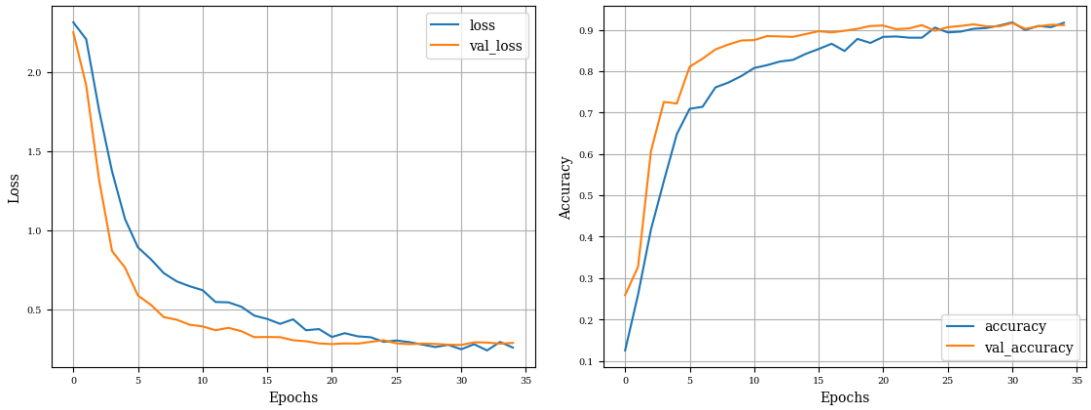
Figure 4.14: Learned Filters - Convolution layer



Figure 4.15: Output Filters - Convolution Layer

4.2.4.9 Variational Dilation Layer

In this layer, the initialization of parameters in the variational MM layer is crucial for stable training and effective pattern recognition. Positive influence parameters (each filter's α^+) are initialized between 0.5 and 1.5. Negative influence parameters (each filter's α^-) are set between -1e8 and -1e7. Weight parameters (each filter's $w_{i,j}$) range from 0.0 to 100.0, allowing for varying importance across different patterns. A constant parameter p is initialized to 2.0, as it was explained before. These initialization ranges collectively make the layer be initialized as a variational dilation layer.



(a) Loss variational Dilation Plot (b) Accuracy variational Dilation Plot

Figure 4.16: Loss and Validation Accuracy of variational Dilation Layer

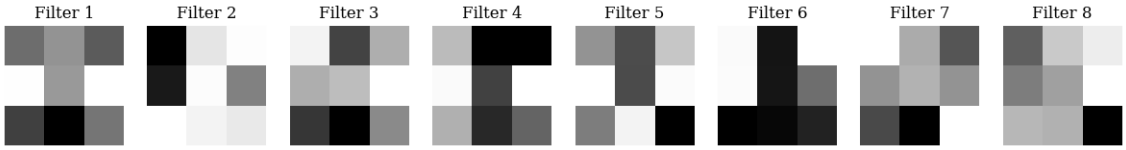


Figure 4.17: Learned Filters - variational Dilation layer

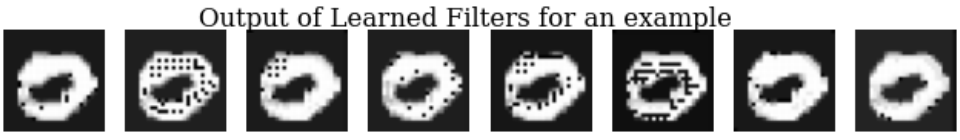
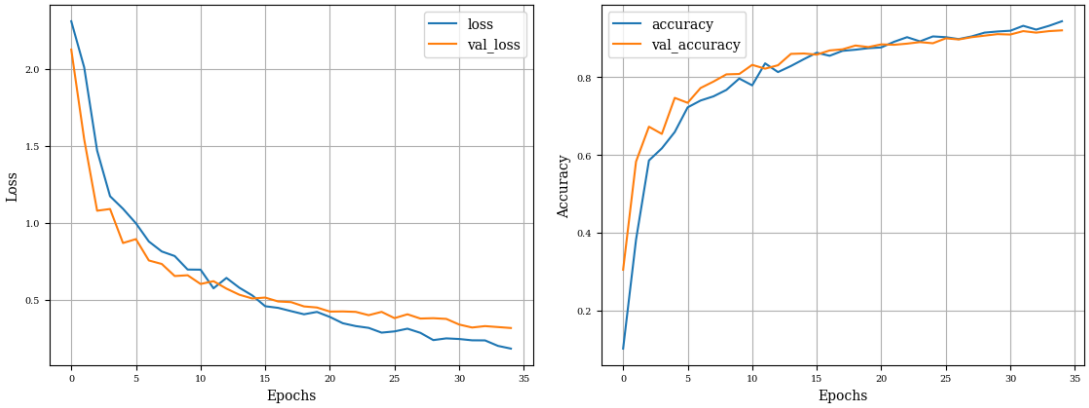


Figure 4.18: Output Filters - variational Dilation Layer

4.2.4.10 Variational Erosion Layer

In this layer, the initialization of parameters in the variational MM layer is crucial for stable training and effective pattern recognition. Positive influence parameters (each filter's α^+) are initialized between 1e7 and 1e8. Negative influence parameters (each filter's α^-) are set between -1.5 and -0.5. Weight parameters (each filter's $w_{i,j}$) range from 0.0 to 100.0, allowing for varying importance across different patterns. A constant parameter p is initialized to 2.0, as it was explained before. These initialization ranges collectively make the layer be initialized as a variational erosion layer.



(a) Loss variational Erosion Plot (b) Accuracy variational Erosion Plot

Figure 4.19: Loss and Validation Accuracy of variational Erosion Layer

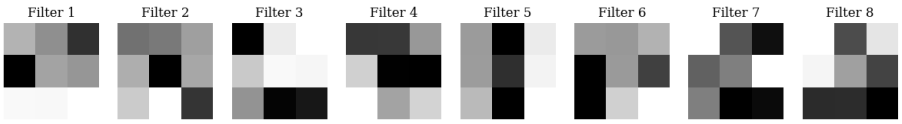


Figure 4.20: Learned Filters - variational Erosion layer

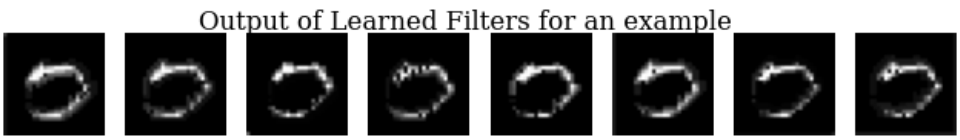


Figure 4.21: Output Filters - variational Erosion Layer

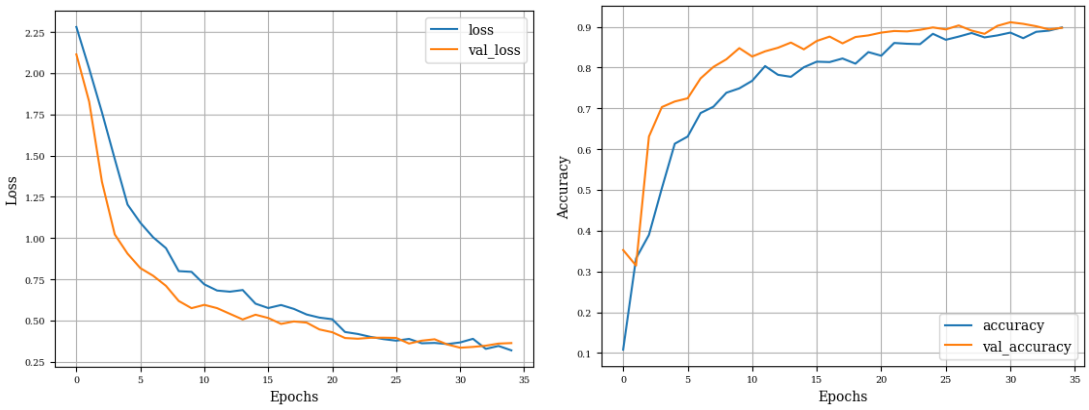
4.2.4.11 Variational Opening Layer

The **morphological opening operation**[39] given a set A and a structuring element B, is defined as:

$$\text{Opening}(A) = (A \ominus B) \oplus B \tag{4.4}$$

where $\ominus$ denotes the erosion operation and $\oplus$ denotes the dilation operation.

In this layer, the initialization of parameters in the variational MM layer is designed to facilitate effective learning and pattern extraction. Positive erosion influence parameters (each filter's α^+ for erosion operation) are initialized between 1e7 and 1e8. Negative erosion influence parameters (each filter's α^- for erosion operation) are set between -1.5 and -0.5. Positive dilation influence parameters (each filter's α^+ for dilation operation) are initialized between 0.5 and 1.5. Negative dilation influence parameters(each filter's α^- for dilation operation) range from -1e8 to -1e7. A constant parameter p is set to 2.0, as explained earlier. Weight parameters (each filter's $w_{i,j}$) range from 0.0 to 100.0, allowing for varying importance across different patterns. These initialization ranges collectively make the layer be initialized as a variational opening layer.



(a) Loss variational Opening Plot (b) Accuracy variational Opening Plot

Figure 4.22: Loss and Validation Accuracy of variational Opening Layer

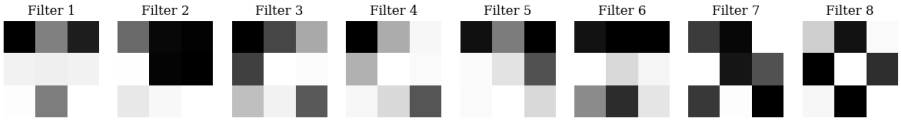


Figure 4.23: Learned Filters - variational Opening layer

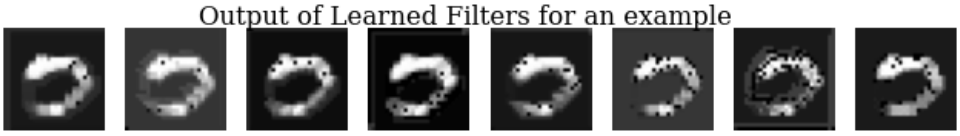


Figure 4.24: Output Filters - variational Opening Layer

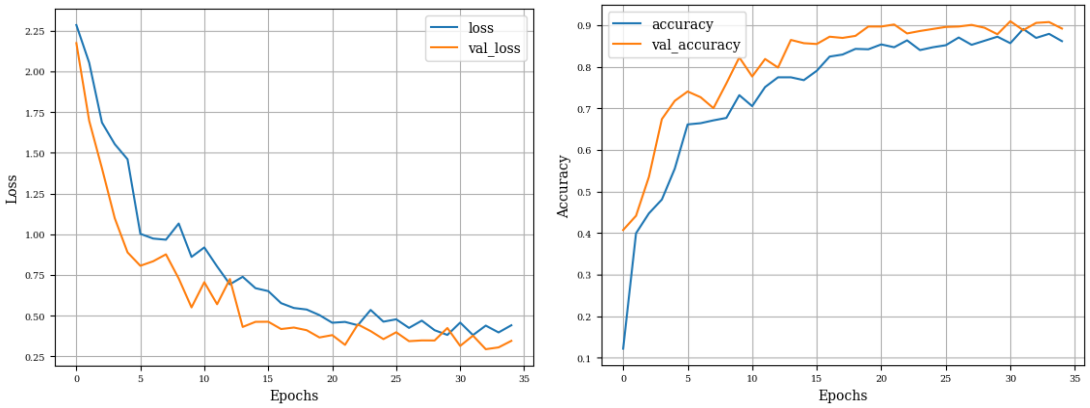
4.2.4.12 Variational Closing Layer

The **morphological closing operation**[39], given a set A and a structuring element B, is defined as:

$$\text{Closing}(A) = (A \oplus B) \ominus B \tag{4.5}$$

where $\oplus$ denotes the dilation operation and $\ominus$ denotes the erosion operation.

In this layer, the initialization of parameters in the variational MM layer is designed to facilitate effective learning and pattern extraction. Positive erosion influence parameters (each filter's α^+ for erosion operation) are initialized between 1e7 and 1e8. Negative erosion influence parameters (each filter's α^- for erosion operation) are set between -1.5 and -0.5. Positive dilation influence parameters (each filter's α^+ for dilation operation) are initialized between 0.5 and 1.5. Negative dilation influence parameters(each filter's α^- for dilation operation) range from -1e8 to -1e7. A constant parameter p is set to 2.0, as explained earlier. Weight parameters (each filter's $w_{i,j}$) range from 0.0 to 100.0, allowing for varying importance across different patterns. These initialization ranges collectively make the layer be initialized as a variational closing layer.



(a) Loss variational Closing Plot (b) Accuracy variational Closing Plot

Figure 4.25: Loss and Validation Accuracy of variational Closing Layer

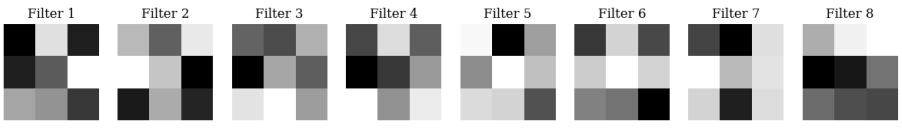


Figure 4.26: Learned Filters - variational Closing layer



Figure 4.27: Output Filters - variational Closing Layer

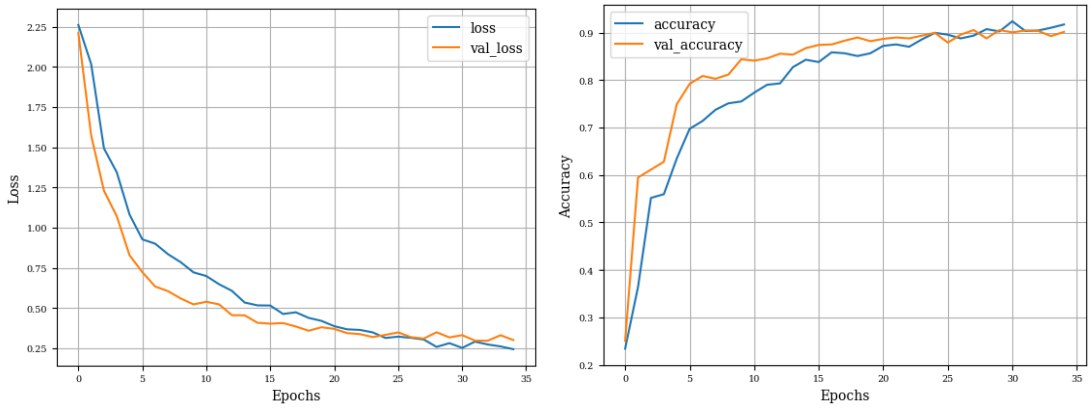
4.2.4.13 Variational Gradient Layer

The **morphological gradient**[52, 66] of a set A with respect to a structuring element B is defined as the difference between the dilation and the erosion of A:

$$\text{Gradient}(A) = (A \oplus B) - (A \ominus B) \tag{4.6}$$

where $\oplus$ denotes the dilation operation and $\ominus$ denotes the erosion operation.

In this layer, the initialization of parameters in the variational MM layer is designed to facilitate effective learning and pattern extraction. Positive erosion influence parameters (each filter's α^+ for erosion operation) are initialized between 1e7 and 1e8. Negative erosion influence parameters (each filter's α^- for erosion operation) are set between -1.5 and -0.5. Positive dilation influence parameters (each filter's α^+ for dilation operation) are initialized between 0.5 and 1.5. Negative dilation influence parameters(each filter's α^- for dilation operation) range from -1e8 to -1e7. A constant parameter p is set to 2.0, as explained earlier. Weight parameters (each filter's $w_{i,j}$) range from 0.0 to 100.0, allowing for varying importance across different patterns. These initialization ranges collectively make the layer be initialized as a variational gradient layer.



(a) Loss variational Gradient Plot

(b) Accuracy variational Gradient Plot

Figure 4.28: Loss and Validation Accuracy of variational Gradient Layer

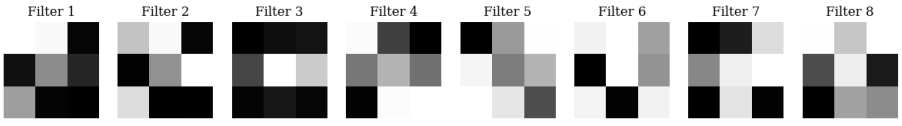


Figure 4.29: Learned Filters - variational Gradient layer



Figure 4.30: Output Filters - variational Gradient Layer

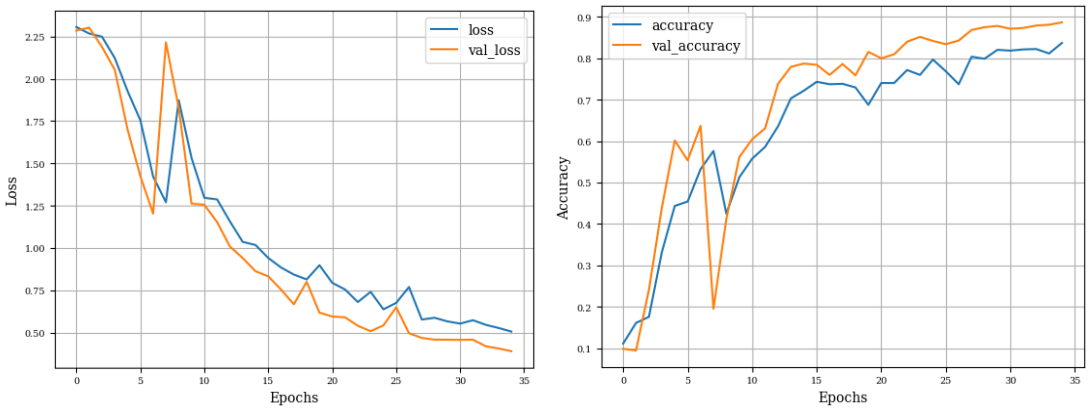
4.2.4.14 Variational Internal Gradient Layer

The **internal gradient**[52] of a set A with respect to a structuring element B is defined as the difference between the set A and its erosion:

$$\text{Internal Gradient}(A) = A - (A \ominus B) \tag{4.7}$$

where $\ominus$ denotes the erosion operation.

In this layer, the initialization of parameters in the variational MM layer is designed to facilitate effective learning and pattern extraction. Positive erosion influence parameters (each filter's α^+ for erosion operation) are initialized between 1e7 and 1e8. Negative erosion influence parameters (each filter's α^- for erosion operation) are set between -1.5 and -0.5. A constant parameter p is set to 2.0, as explained earlier. Weight parameters (each filter's $w_{i,j}$) range from 0.0 to 100.0, allowing for varying importance across different patterns. These initialization ranges collectively make the layer be initialized as a variational internal gradient layer.



(a) Loss variational Internal Gradient Plot (b) Accuracy variational Internal Gradient Plot

Figure 4.31: Loss and Validation Accuracy of variational Internal Gradient Layer

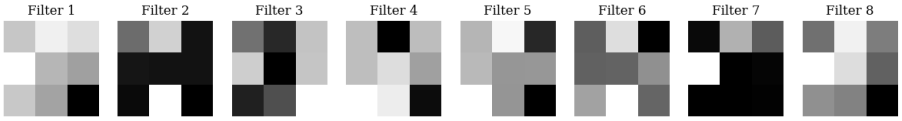


Figure 4.32: Learned Filters - variational Internal Gradient layer



Figure 4.33: Output Filters - variational Internal Gradient Layer

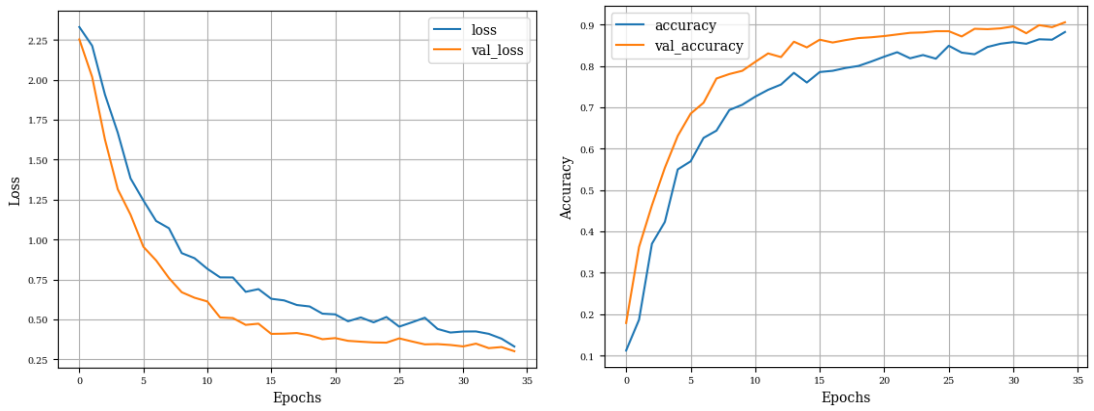
4.2.4.15 Variational External Gradient Layer

The **external gradient**[52] of a set A with respect to a structuring element B is defined as the difference between the dilation of A and the set A:

$$\text{External Gradient}(A) = (A \oplus B) - A \tag{4.8}$$

where $\oplus$ denotes the dilation operation.

In this layer, the initialization of parameters in the variational MM layer is designed to facilitate effective learning and pattern extraction. Positive dilation influence parameters (each filter's α^+ for dilation operation) are initialized between 0.5 and 1.5. Negative dilation influence parameters(each filter's α^- for dilation operation) range from -1e8 to -1e7. A constant parameter p is set to 2.0, as explained earlier. Weight parameters (each filter's $w_{i,j}$) range from 0.0 to 100.0, allowing for varying importance across different patterns. These initialization ranges collectively make the layer be initialized as a variational external gradient layer.



(a) Loss variational External Gradient Plot (b) Accuracy variational External Gradient Plot

Figure 4.34: Loss and Validation Accuracy of variational External Gradient Layer

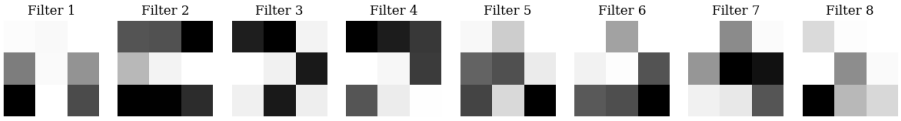


Figure 4.35: Learned Filters - variational External Gradient layer



Figure 4.36: Output Filters - variational External Gradient Layer

4.2.4.16 Variational TopHat Opening Layer

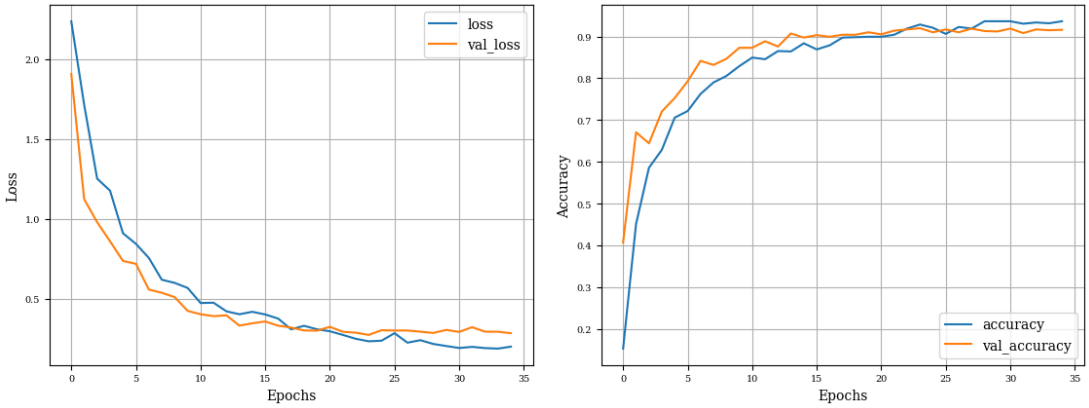
The **top-hat (opening) transform**[25] of a set A with respect to a structuring element B is defined as the difference between the set A and its opening:

$$\text{Top-Hat}(A) = A - (A \circ B) \tag{4.9}$$

where $\circ$ denotes the opening operation, which is the erosion followed by dilation:

$$A \circ B = (A \ominus B) \oplus B \tag{4.10}$$

In this layer, the initialization of parameters in the variational MM layer is designed to facilitate effective learning and pattern extraction. Positive erosion influence parameters (each filter's α^+ for erosion operation) are initialized between 1e7 and 1e8. Negative erosion influence parameters (each filter's α^- for erosion operation) are set between -1.5 and -0.5. Positive dilation influence parameters (each filter's α^+ for dilation operation) are initialized between 0.5 and 1.5. Negative dilation influence parameters(each filter's α^- for dilation operation) range from -1e8 to -1e7. A constant parameter p is set to 2.0, as explained earlier. Weight parameters (each filter's $w_{i,j}$) range from 0.0 to 100.0, allowing for varying importance across different patterns. These initialization ranges collectively make the layer be initialized as a variational tophat opening layer.



(a) Loss variational TopHat Opening Plot (b) Accuracy variational TopHat Opening Plot

Figure 4.37: Loss and Validation Accuracy of variational TopHat Opening Layer

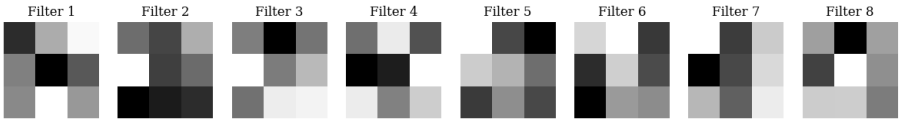


Figure 4.38: Learned Filters - variational TopHat Opening layer



Figure 4.39: Output Filters - variational TopHat Opening Layer

4.2.4.17 Variational TopHat Closing Layer

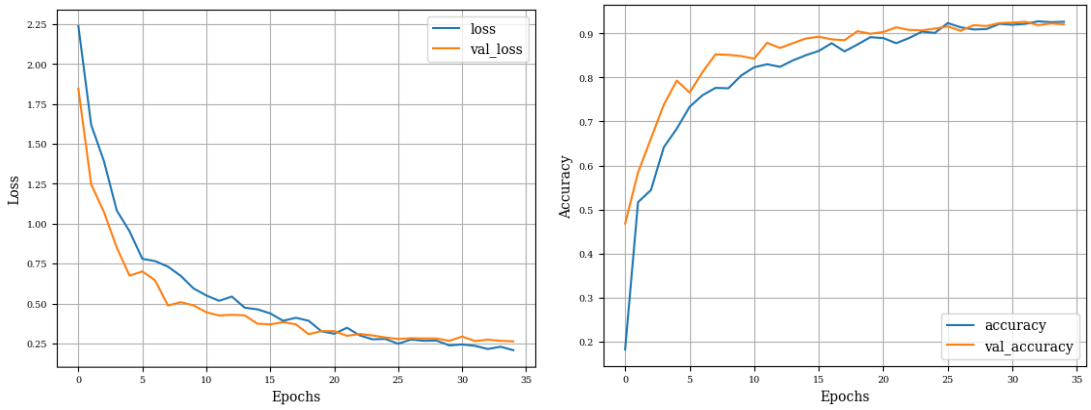
The **top-hat (closing) transform**[25] of a set A with respect to a structuring element B is defined as the difference between the closing of A and the set A:

$$\text{Top-Hat}(A) = (A \bullet B) - A \tag{4.11}$$

where $\bullet$ denotes the closing operation, which is the dilation followed by erosion:

$$A \bullet B = (A \oplus B) \ominus B \tag{4.12}$$

In this layer, the initialization of parameters in the variational MM layer is designed to facilitate effective learning and pattern extraction. Positive erosion influence parameters (each filter's α^+ for erosion operation) are initialized between 1e7 and 1e8. Negative erosion influence parameters (each filter's α^- for erosion operation) are set between -1.5 and -0.5. Positive dilation influence parameters (each filter's α^+ for dilation operation) are initialized between 0.5 and 1.5. Negative dilation influence parameters(each filter's α^- for dilation operation) range from -1e8 to -1e7. A constant parameter p is set to 2.0, as explained earlier. Weight parameters (each filter's $w_{i,j}$) range from 0.0 to 100.0, allowing for varying importance across different patterns. These initialization ranges collectively make the layer be initialized as a variational tophat closing layer.



(a) Loss variational TopHat Closing Plot (b) Accuracy variational TopHat Closing Plot

Figure 4.40: Loss and Validation Accuracy of variational TopHat Opening Layer

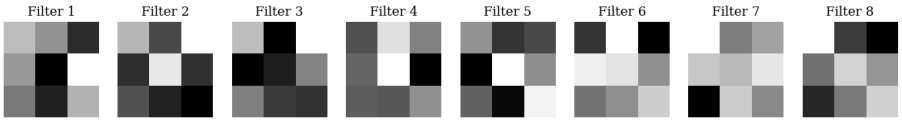


Figure 4.41: Learned Filters - variational TopHat Closing layer



Figure 4.42: Output Filters - variational TopHat Closing Layer

4.2.4.18 Comparing Results

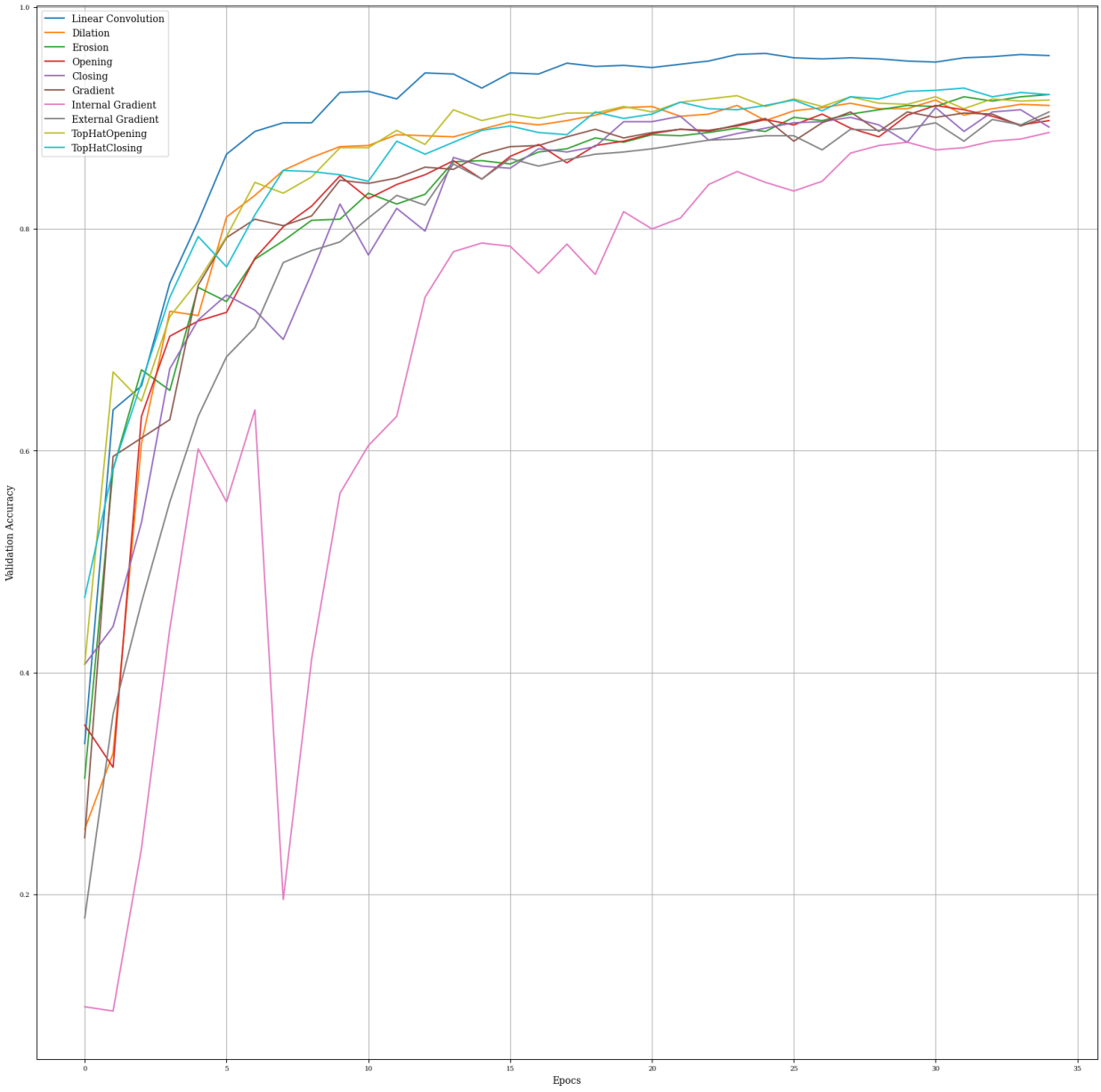


Figure 4.43: Accuraccy Plot - All methods

With the analysis of Figure 4.43, it's possible to see the validation accuracy of all methods and compare them between each other and compare them with the Convolution Layer. It's possible to understand that, unfortunately none of the variational MM layers managed to have better results than the target on, the Convolution layer. Nonetheless, this results show that is possible to use variational MM operations to have decent results, as with the variational Erosion layer.

With this training, it was possible to have the best values of accuracy for each layer, which it helps the conclusion made from Fig. 4.43.

- Linear Convolution: 0.9599609375
- Dilation: 0.91796875
- Erosion: 0.9443359375
- Opening: 0.8984375
- Closing: 0.890625
- Gradient: 0.923828125
- Internal Gradient: 0.8369140625
- External Gradient: 0.8818359375
- TopHat Opening: 0.9365234375
- TopHat Closing: 0.927734375

4.2.5 Neural Network Study Case - 8 Filters with different U_i

The experiment involves training a Convolutional Neural Network (CNN) and other 9 different version where the first convolutional layer is replaced for a variational MM layer, which is building upon 8 different filters, which will be explained later in this section, on the MNIST dataset to classify handwritten digits. This section provides an overview of the model architecture, data preparation steps, training process, evaluation metrics, and visualizations of the learned features.

4.2.5.1 Model Architecture

Both CNN architecture and all variational MM neural networks used for this experiment are designed to process grayscale images of size 28×28. These archictures consist in:

4.2.5.2 CNN Architecture

- Two convolutional layers (‘Conv1‘ and ‘Conv2‘) with ReLU activation followed by max pooling layers.
- Dropout layer with probability $p = 0.5$ to prevent overfitting.
- Two fully connected layers (‘Fully Connected Layer 1‘ and ‘Fully Connected Layer 2‘) with ReLU activation.
- Output layer (‘Output‘) with logits for classification into 10 classes (digits 0-9).

4.2.5.3 Variational MM neural networks Architecture

- One variational MM layer with ReLU activation followed by a max pooling layer.
- One convolutional layer (‘Conv2‘) with ReLU activation followed by a max pooling layers.
- Dropout layer with probability $p = 0.5$ to prevent overfitting.
- Two fully connected layers (‘Fully Connected Layer 1‘ and ‘Fully Connected Layer 2‘) with ReLU activation.
- Output layer (‘Output‘) with logits for classification into 10 classes (digits 0-9).

4.2.5.4 Data Preparation

The MNIST dataset is used, which consists of $60,000$ training images and $10,000$ test images of handwritten digits. The following transformations are applied to each image:

- Conversion to grayscale.
- Conversion to PyTorch tensors.
- Normalization based on MNIST dataset statistics.

A subset of 1024 samples is used for both training and testing to expedite experimentation. Using a consistent subset size for training and testing significantly reduces the computation power needed.

4.2.5.5 Training

The model is trained using:

- Cross-entropy loss function for multi-class classification.
- Adam optimizer with a learning rate of 0.01.
- Learning rate scheduler to adjust learning rate based on validation loss.
- Training for 35 epochs with batch size 512.

During training, the model's performance metrics (loss and accuracy) are monitored on both training and validation sets.

4.2.5.6 Evaluation

After training, the model is evaluated on the test set to assess its generalization ability. Evaluation metrics include:

- Confusion matrix to visualize predicted vs actual class distributions.
- Classification report showing precision, recall, and F1-score for each class.

These metrics provide insights into the model's performance across different digits.

4.2.5.7 Visualizations

Several visualizations are generated to analyze the model's behavior and learned features:

- Training history plots showing loss and accuracy over epochs.
- Visualization of learned filters in the first convolutional layer.
- Output of learned filters applied to example images.

These visualizations aid in understanding how the model processes and distinguishes between different digits.

As in the experiment of equal 8 square filters, the structure of the default convolutional neural network is the same as in 6.

In this experiment, the objective is the same in last one but now each variational MM layer has 8 different filters, using different values for U_i (neighborhood). Instead of using 8 square 3x3 filters, now the first two filters work with U_i being a square 3x3, the third and fourth use U_i being an horizontal line size 3 and last four use, respectively, diagonal right line, diagonal left line, diagonal

cross and non-diagonal cross (examples of usage of these different structures for U_i is shown in earlier chapter 1 and 2). These different values used for U_i can be seen in image 4.44.

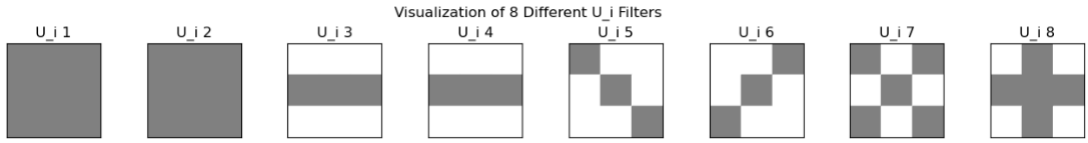
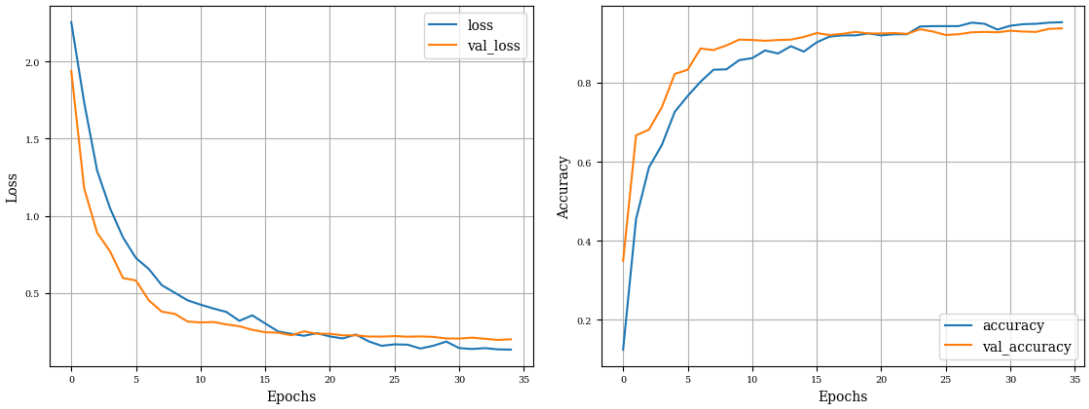


Figure 4.44: Different values of U_i - neighborhood - ordered for each filter of variational MM layer

Each following layer results are illustrated with the Confusion Matrix, the output filters and the graphics for loss and accuracy. All confusion matrix plots can be seen on appendix section.

4.2.5.8 Convolution Layer



(a) Loss Convolution Plot

(b) Accuracy Convolution Plot

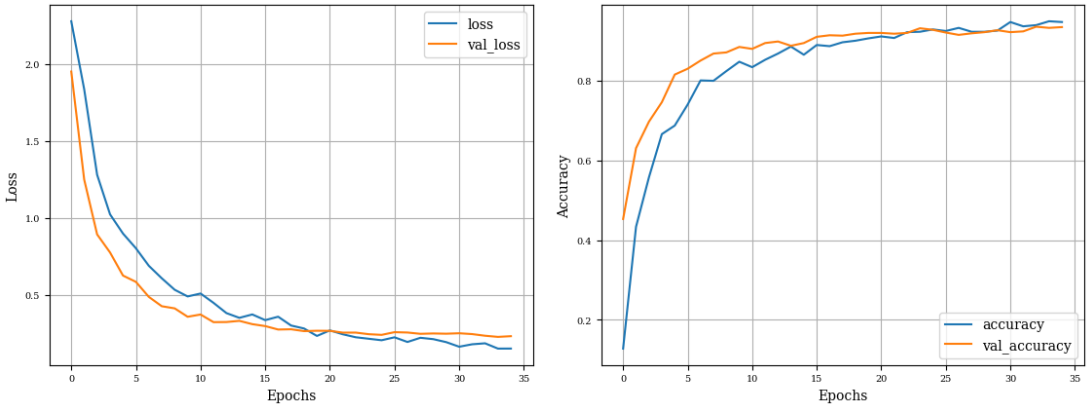
Figure 4.45: Loss and Validation Accuracy of Convolution Layer



Figure 4.46: Output Filters - Convolution Layer

4.2.5.9 Variational Dilation Layer

In this layer, the initialization of parameters in the variational MM layer is crucial for stable training and effective pattern recognition. Positive influence parameters (each filter's α^{+}) are initialized between 0.5 and 1.5. Negative influence parameters (each filter's α^{-}) are set between -1e8 and -1e7. Weight parameters (each filter's $w_{i,j}$) range from 0.0 to 100.0, varying the structure of U_i as explained before. A constant parameter p is initialized to 2.0, as it was explained before. These initialization ranges collectively make the layer be initialized as a variational dilation layer.



(a) Loss variational Dilation Plot (b) Accuracy variational Dilation Plot

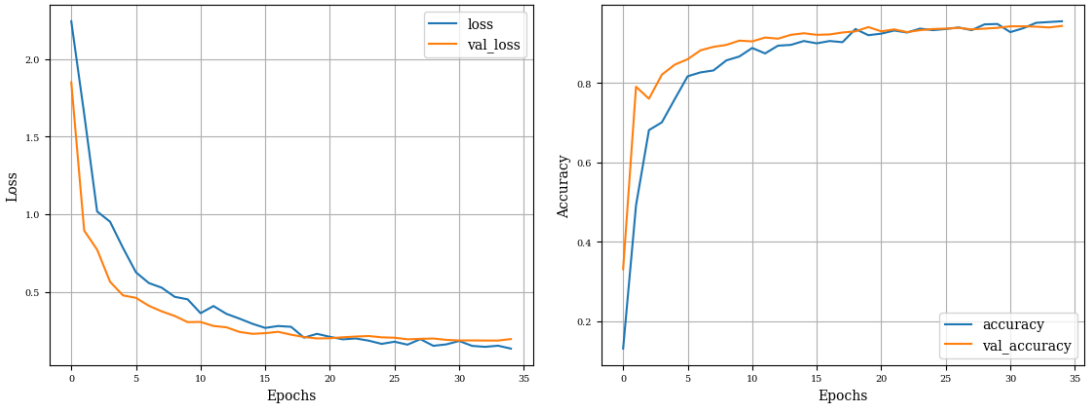
Figure 4.47: Loss and Validation Accuracy of variational Dilation Layer



Figure 4.48: Output Filters - variational Dilation Layer

4.2.5.10 Variational Erosion Layer

In this layer, the initialization of parameters in the variational MM layer is crucial for stable training and effective pattern recognition. Positive influence parameters (each filter's α^+) are initialized between 1e7 and 1e8. Negative influence parameters (each filter's α^-) are set between -1.5 and -0.5. Weight parameters (each filter's $w_{i,j}$) range from 0.0 to 100.0, varying the structure of U_i as explained before. A constant parameter p is initialized to 2.0, as it was explained before. These initialization ranges collectively make the layer be initialized as a variational erosion layer.



(a) Loss variational Erosion Plot (b) Accuracy variational Erosion Plot

Figure 4.49: Loss and Validation Accuracy of variational Erosion Layer



Figure 4.50: Output Filters - variational Erosion Layer

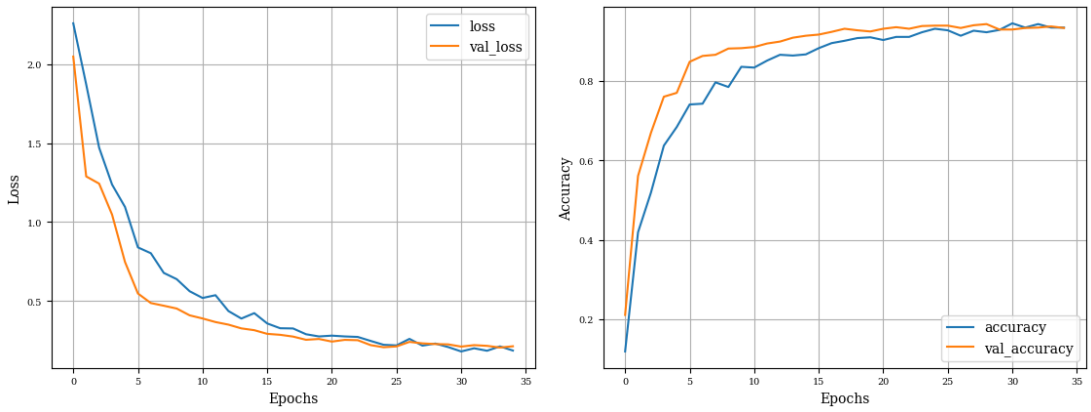
4.2.5.11 Variational Opening Layer

The **morphological opening operation**[39] given a set A and a structuring element B, is defined as:

$$\text{Opening}(A) = (A \ominus B) \oplus B \tag{4.13}$$

where $\ominus$ denotes the erosion operation and $\oplus$ denotes the dilation operation.

In this layer, the initialization of parameters in the variational MM layer is designed to facilitate effective learning and pattern extraction. Positive erosion influence parameters (each filter's α^+ for erosion operation) are initialized between 1e7 and 1e8. Negative erosion influence parameters (each filter's α^- for erosion operation) are set between -1.5 and -0.5. Positive dilation influence parameters (each filter's α^+ for dilation operation) are initialized between 0.5 and 1.5. Negative dilation influence parameters(each filter's α^- for dilation operation) range from -1e8 to -1e7. A constant parameter p is set to 2.0, as explained earlier. Weight parameters (each filter's $w_{i,j}$) range from 0.0 to 100.0, varying the structure of U_i as explained before. These initialization ranges collectively make the layer be initialized as a variational opening layer.



(a) Loss variational Opening Plot (b) Accuracy variational Opening Plot

Figure 4.51: Loss and Validation Accuracy of variational Opening Layer



Figure 4.52: Output Filters - variational Opening Layer

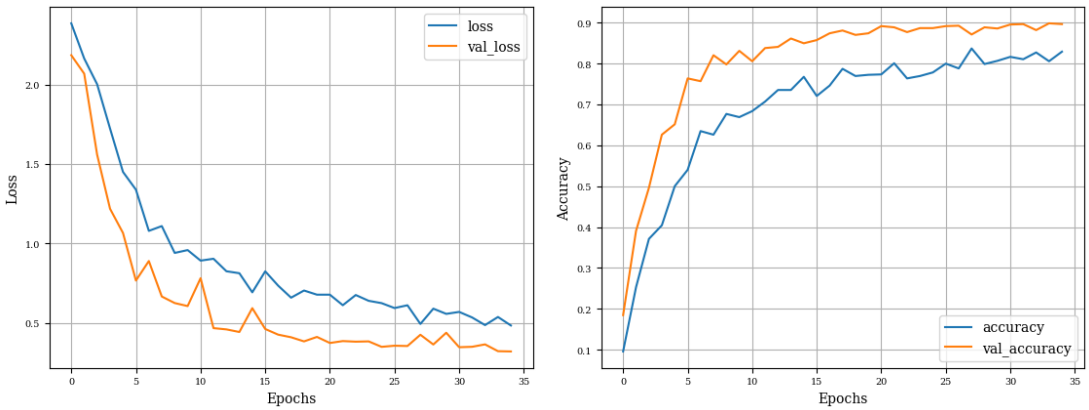
4.2.5.12 Variational Closing Layer

The **morphological closing operation**[39], given a set A and a structuring element B, is defined as:

$$\text{Closing}(A) = (A \oplus B) \ominus B \tag{4.14}$$

where $\oplus$ denotes the dilation operation and $\ominus$ denotes the erosion operation.

In this layer, the initialization of parameters in the variational MM layer is designed to facilitate effective learning and pattern extraction. Positive erosion influence parameters (each filter's α^+ for erosion operation) are initialized between 1e7 and 1e8. Negative erosion influence parameters (each filter's α^- for erosion operation) are set between -1.5 and -0.5. Positive dilation influence parameters (each filter's α^+ for dilation operation) are initialized between 0.5 and 1.5. Negative dilation influence parameters(each filter's α^- for dilation operation) range from -1e8 to -1e7. A constant parameter p is set to 2.0, as explained earlier. Weight parameters (each filter's $w_{i,j}$) range from 0.0 to 100.0, varying the structure of U_i as explained before. These initialization ranges collectively make the layer be initialized as a variational closing layer.



(a) Loss variational Closing Plot (b) Accuracy variational Closing Plot

Figure 4.53: Loss and Validation Accuracy of variational Closing Layer



Figure 4.54: Output Filters - variational Closing Layer

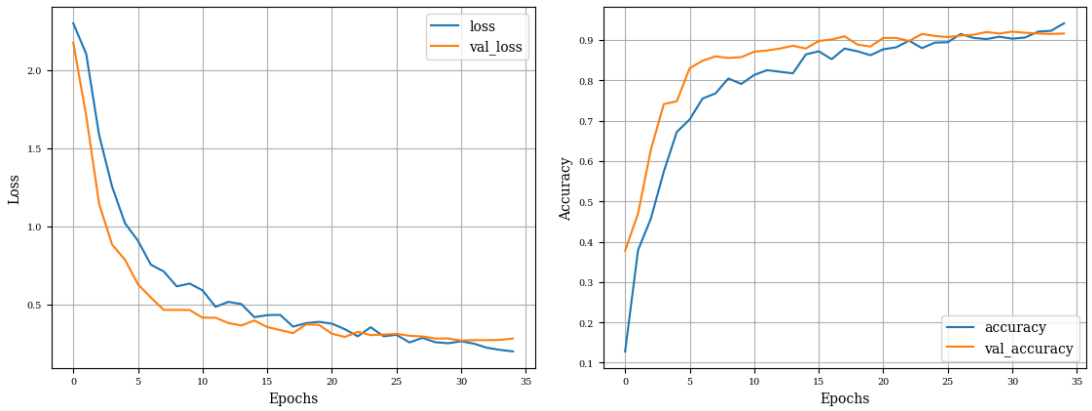
4.2.5.13 Variational Gradient Layer

The **morphological gradient**[52, 66] of a set A with respect to a structuring element B is defined as the difference between the dilation and the erosion of A:

$$\text{Gradient}(A) = (A \oplus B) - (A \ominus B) \tag{4.15}$$

where $\oplus$ denotes the dilation operation and $\ominus$ denotes the erosion operation.

In this layer, the initialization of parameters in the variational MM layer is designed to facilitate effective learning and pattern extraction. Positive erosion influence parameters (each filter's α^+ for erosion operation) are initialized between 1e7 and 1e8. Negative erosion influence parameters (each filter's α^- for erosion operation) are set between -1.5 and -0.5. Positive dilation influence parameters (each filter's α^+ for dilation operation) are initialized between 0.5 and 1.5. Negative dilation influence parameters(each filter's α^- for dilation operation) range from -1e8 to -1e7. A constant parameter p is set to 2.0, as explained earlier. Weight parameters (each filter's $w_{i,j}$) range from 0.0 to 100.0, varying the structure of U_i as explained before. These initialization ranges collectively make the layer be initialized as a variational gradient layer.



(a) Loss variational Gradient Plot (b) Accuracy variational Gradient Plot

Figure 4.55: Loss and Validation Accuracy of variational Gradient Layer



Figure 4.56: Output Filters - variational Gradient Layer

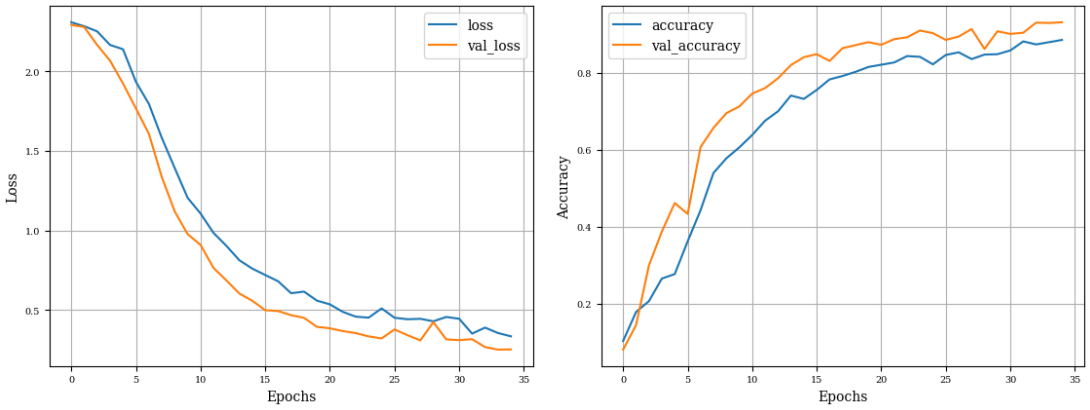
4.2.5.14 Variational Internal Gradient Layer

The **internal gradient**[52] of a set A with respect to a structuring element B is defined as the difference between the set A and its erosion:

$$\text{Internal Gradient}(A) = A - (A \ominus B) \tag{4.16}$$

where $\ominus$ denotes the erosion operation.

In this layer, the initialization of parameters in the variational MM layer is designed to facilitate effective learning and pattern extraction. Positive erosion influence parameters (each filter's α^+ for erosion operation) are initialized between 1e7 and 1e8. Negative erosion influence parameters (each filter's α^- for erosion operation) are set between -1.5 and -0.5. A constant parameter p is set to 2.0, as explained earlier. Weight parameters (each filter's $w_{i,j}$) range from 0.0 to 100.0, varying the structure of U_i as explained before. These initialization ranges collectively make the layer be initialized as a variational internal gradient layer.



(a) Loss variational Internal Gradient Plot (b) Accuracy variational Internal Gradient Plot

Figure 4.57: Loss and Validation Accuracy of variational Internal Gradient Layer



Figure 4.58: Output Filters - variational Internal Gradient Layer

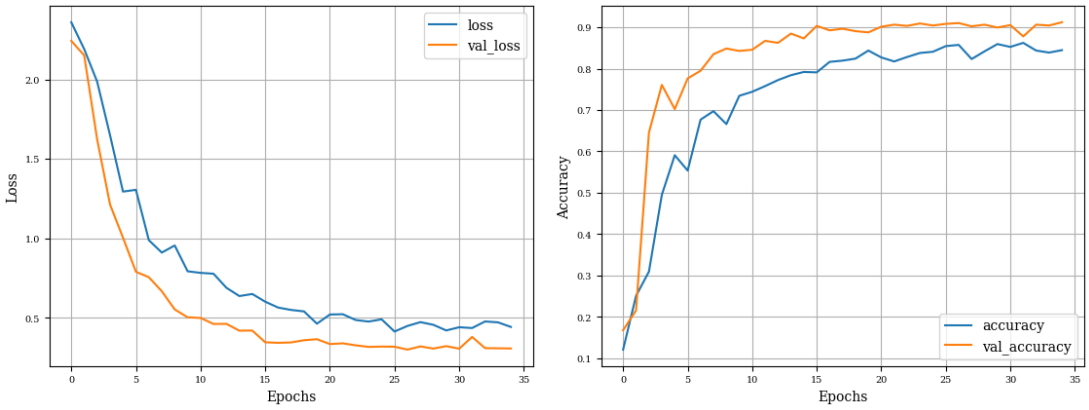
4.2.5.15 Variational External Gradient Layer

The **external gradient**[52] of a set A with respect to a structuring element B is defined as the difference between the dilation of A and the set A:

$$\text{External Gradient}(A) = (A \oplus B) - A \tag{4.17}$$

where $\oplus$ denotes the dilation operation.

In this layer, the initialization of parameters in the variational MM layer is designed to facilitate effective learning and pattern extraction. Positive dilation influence parameters (each filter's α^+ for dilation operation) are initialized between 0.5 and 1.5. Negative dilation influence parameters(each filter's α^- for dilation operation) range from -1e8 to -1e7. A constant parameter p is set to 2.0, as explained earlier. Weight parameters (each filter's $w_{i,j}$) range from 0.0 to 100.0, varying the structure of U_i as explained before. These initialization ranges collectively make the layer be initialized as a variational external gradient layer.



(a) Loss variational External Gradient Plot (b) Accuracy variational External Gradient Plot

Figure 4.59: Loss and Validation Accuracy of variational External Gradient Layer



Figure 4.60: Output Filters - variational External Gradient Layer

4.2.5.16 Variational TopHat Opening Layer

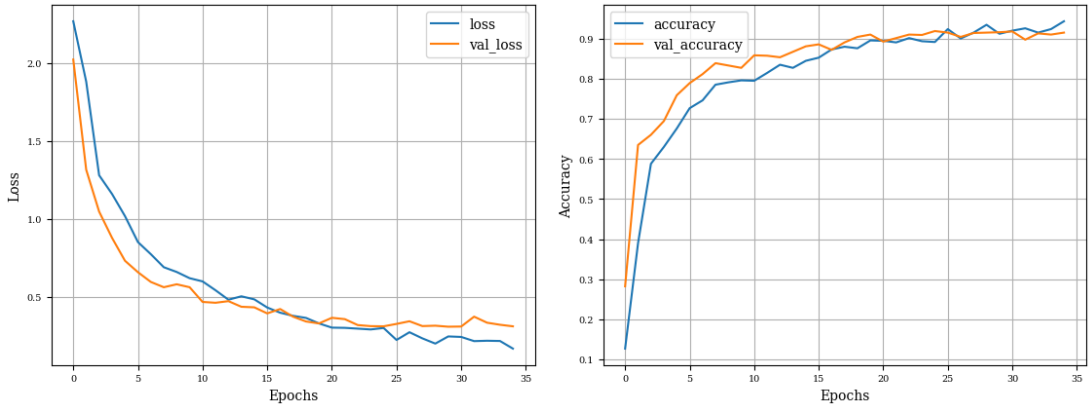
The **top-hat (opening) transform**[25] of a set A with respect to a structuring element B is defined as the difference between the set A and its opening:

$$\text{Top-Hat}(A) = A - (A \circ B) \tag{4.18}$$

where $\circ$ denotes the opening operation, which is the erosion followed by dilation:

$$A \circ B = (A \ominus B) \oplus B \tag{4.19}$$

In this layer, the initialization of parameters in the variational MM layer is designed to facilitate effective learning and pattern extraction. Positive erosion influence parameters (each filter's α^+ for erosion operation) are initialized between 1e7 and 1e8. Negative erosion influence parameters (each filter's α^- for erosion operation) are set between -1.5 and -0.5. Positive dilation influence parameters (each filter's α^+ for dilation operation) are initialized between 0.5 and 1.5. Negative dilation influence parameters(each filter's α^- for dilation operation) range from -1e8 to -1e7. A constant parameter p is set to 2.0, as explained earlier. Weight parameters (each filter's $w_{i,j}$) range from 0.0 to 100.0, but with the structure explained earlier. These initialization ranges collectively make the layer be initialized as a variational tophat opening layer.



(a) Loss variational TopHat Opening Plot (b) Accuracy variational TopHat Opening Plot

Figure 4.61: Loss and Validation Accuracy of variational TopHat Opening Layer

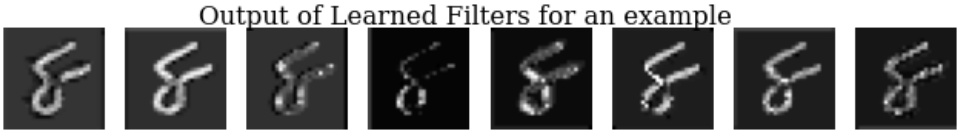


Figure 4.62: Output Filters - variational TopHat Opening Layer

4.2.5.17 Variational TopHat Closing Layer

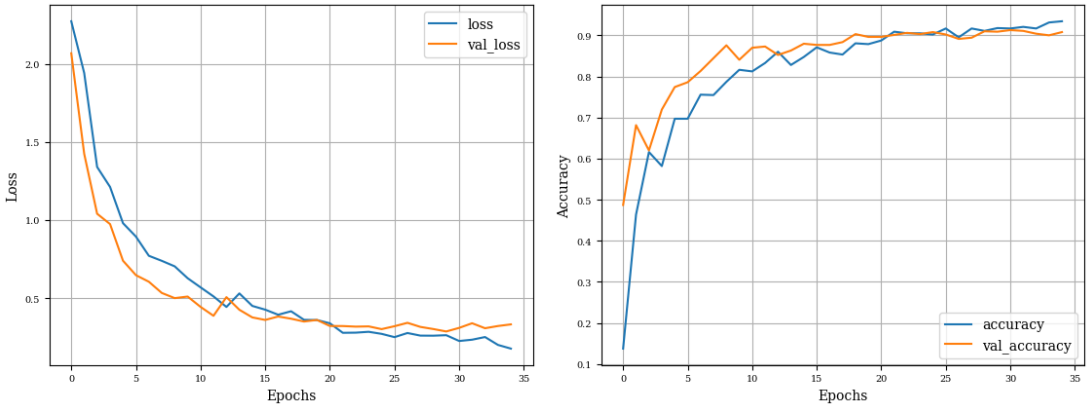
The **top-hat (closing) transform**[25] of a set A with respect to a structuring element B is defined as the difference between the closing of A and the set A:

$$\text{Top-Hat}(A) = (A \bullet B) - A \tag{4.20}$$

where $\bullet$ denotes the closing operation, which is the dilation followed by erosion:

$$A \bullet B = (A \oplus B) \ominus B \tag{4.21}$$

In this layer, the initialization of parameters in the variational MM layer is designed to facilitate effective learning and pattern extraction. Positive erosion influence parameters (each filter's α^+ for erosion operation) are initialized between 1e7 and 1e8. Negative erosion influence parameters (each filter's α^- for erosion operation) are set between -1.5 and -0.5. Positive dilation influence parameters (each filter's α^+ for dilation operation) are initialized between 0.5 and 1.5. Negative dilation influence parameters(each filter's α^- for dilation operation) range from -1e8 to -1e7. A constant parameter p is set to 2.0, as explained earlier. Weight parameters (each filter's $w_{i,j}$) range from 0.0 to 100.0, varying the structure of U_i as explained before. These initialization ranges collectively make the layer be initialized as a variational tophat closing layer.



(a) Loss variational TopHat Closing Plot (b) Accuracy variational TopHat Closing Plot

Figure 4.63: Loss and Validation Accuracy of variational TopHat Opening Layer



Figure 4.64: Output Filters - variational TopHat Closing Layer

4.2.5.18 Comparing Results

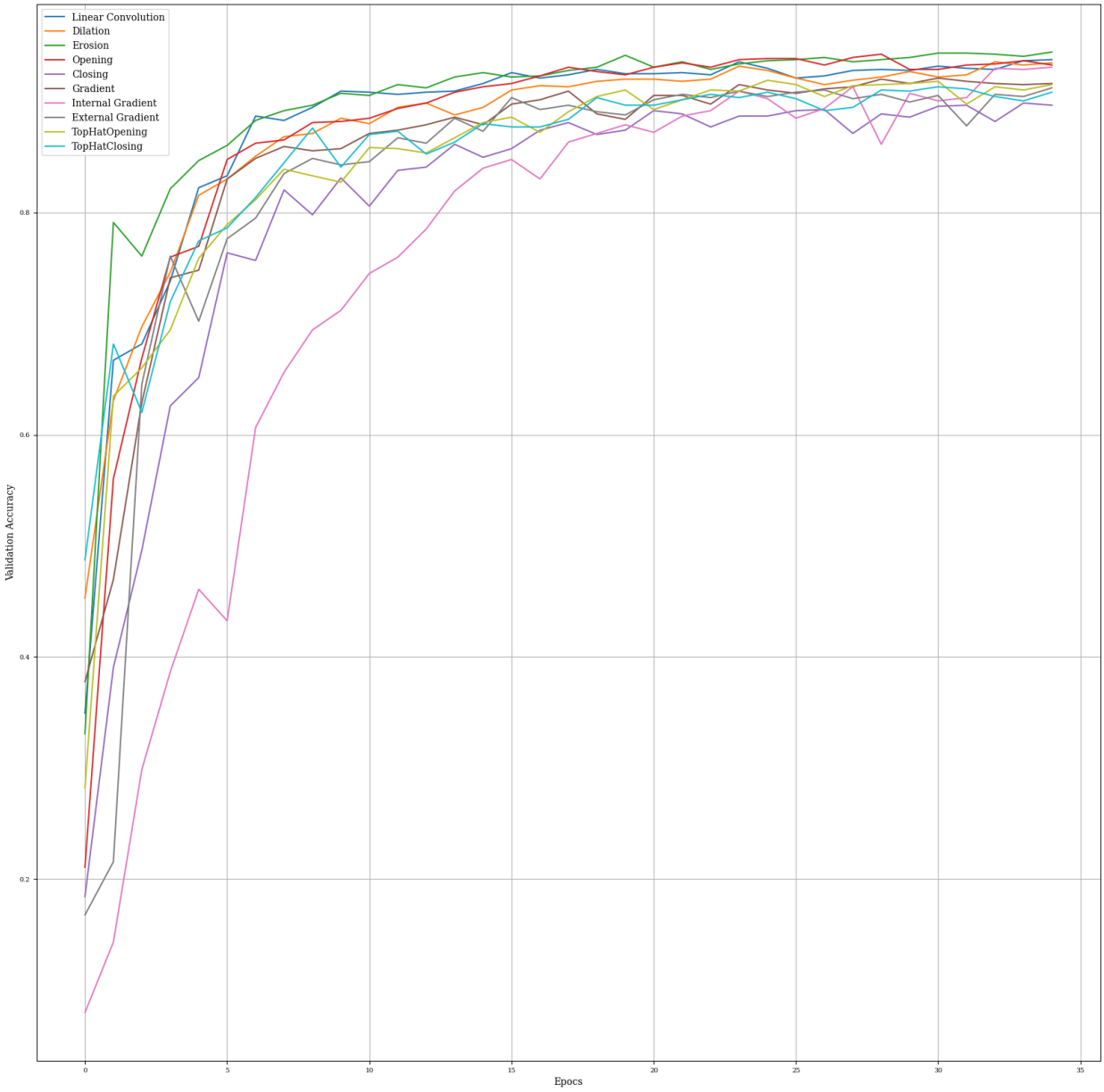


Figure 4.65: Accuraccy Plot - All methods

With the analysis of Figure 4.65, it's possible to see the validation accuracy of all methods and compare them between each other and compare them with the Convolution Layer. It's possible to understand that, unlike the experiment with all filters being squares, this experiment show improvement on many layers and, especially, on variational Erosion layer which has better results then the Convolution layer and variational Opening layer which has better values than Convolution layer for almost all training and finishes with a really similar value.

With this training, it was possible to have the best values of accuracy for each layer, which it helps the conclusion made from Figure 4.65 and shows good improvement from the best accuracy

values got from last experiment.

- Linear Convolution: 0.953125
- Dilation: 0.94921875
- Erosion: 0.9560546875
- Opening: 0.9443359375
- Closing: 0.8369140625
- Gradient: 0.94140625
- Internal Gradient: 0.884765625
- External Gradient: 0.8623046875
- TopHat Opening: 0.943359375
- TopHat Closing: 0.9345703125

Based on these results, it can be concluded that employing variational mathematical morphology operations enhances the accuracy of image classification tasks within neural networks. Furthermore, it's the flexibility of processing the same image with different filters, that makes those results possible and achievable.

4.2.6 Conclusion

Variational morphological layers, emerging as a novel development in neural network architecture, present a promising complementary approach to traditional convolutional and max-pooling layers, especially in image classification tasks like MNIST. By prioritizing shape and structural properties, these layers can potentially improve loss reduction and accuracy in classification models. Future investigations will delve into their integration and potential advantages across more complex and varied datasets, aiming to pave the path toward more robust and efficient neural network architectures.

Chapter 5

Conclusion

This thesis has presented significant advancements in addressing the primary minimization problem within variational mathematical morphology (MM). By leveraging the principles of convex optimization, we have developed robust solutions that effectively solve the core variational MM problem. These solutions were not only theoretically validated but also empirically tested to ensure their convergence and performance.

One of the key goals and achievements of this work was the implementation of effective and optimized solutions, as the Pytorch version of the solution developed in this thesis, a widely-used deep learning framework. This implementation facilitated efficient computation and allowed for extensive testing on various datasets. The empirical tests demonstrated that the solutions converge reliably and perform efficiently, underscoring their practical applicability in real-world scenarios. This performance goal was possible to achieve thanks to the all vectorized logic implemented throughout the implementations.

Furthermore, it was successfully developed a variational MM layer that outperformed traditional convolutional layers. This new layer demonstrated superior results in preserving crucial details and enhancing image quality, making it a valuable addition to neural network architectures.

Beyond solving the major variational MM problem presented here, this thesis explored the integration of the solutions accomplished into advanced computational techniques such as 3D meshes, graphs, neural networks, and rank filters. These methodologies offer substantial benefits, particularly in enhancing the efficiency of data representation and analysis.

The application on 3D meshes, exemplified by the Trimesh library, has proven invaluable in the manipulation and analysis of 3D models. The capacity to import, export, and visualize complex geometries enables accurate modeling of real-world structures, important in fields like 3D printing and medical imaging. The advanced tools for mesh repair, transformation, and analysis ensure the maintenance of high-quality data, preserving essential details and facilitating robust analysis.

Graph-based approaches, as demonstrated by the NetworkX library, provide powerful tools for handling complex network structures. The ability to analyze, model, and visualize intricate relationships within data sets enhances the depth of analysis, particularly in fields requiring detailed

structural insights. The versatility of graph algorithms in solving problems related to connectivity, centrality, and clustering significantly contributes to informed decision-making processes.

Neural networks, integrated within the PyTorch framework, further extend the capabilities of image processing by enabling sophisticated pattern recognition and data classification. The combination of neural networks with variational methods enhances accuracy and precision, which is particularly beneficial in medical imaging, where these characteristics are critical.

Additionally, the implementation of variational rank filters has been highlighted as a powerful technique for eliminating impulsive noise without compromising high spatial frequencies. These filters, by leveraging rank-based selection, maintain sharp edges and crucial details, which are vital for both human perception and automated object detection. This makes them exceptionally suitable for medical applications, where the integrity of image features is essential for accurate diagnosis and treatment planning.

5.1 Future Work

Looking ahead, there are several avenues for future research to build upon the work presented in this thesis. One promising direction is to continue exploring beyond the current neural network applications. Specifically, it would be valuable to investigate the effects of changing the size of the neighborhood (U_i) in variational MM layers. Understanding how different sizes impact performance could lead to optimized configurations for various tasks.

Additionally, an interesting extension would be to evaluate the effects of instead of replacing the convolutional layer, it would be the max-pooling layer with a variational MM layer, for example a variational dilation layer within neural network architectures. This approach could potentially enhance feature extraction and retention, providing further improvements in image processing and analysis tasks. These investigations could open new possibilities for refining neural network designs and improving their applicability in diverse domains.

In conclusion, the solutions developed and presented in this thesis effectively address the primary minimization variational MM problem, demonstrating both theoretical soundness and practical applicability. The integration of 3D meshes, graphs, neural networks, and rank filters into computational workflows provides a robust framework for achieving high-quality data representation and analysis. These advancements promise to drive further innovations, particularly in medical imaging, where the demands for accuracy and detail are ever-increasing. The work presented in this thesis lays a solid foundation for future research and development, paving the way for more sophisticated and effective computational tools in various domains.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Luis Alvarez, Frédéric Guichard, P-L Lions, and J-M Morel. Axiomatisation et nouveaux opérateurs de la morphologie mathématique. *Comptes rendus de l'Académie des sciences. Série 1, Mathématique*, 315(3):265–268, 1992.

[3] Luis Álvarez, Frédéric Guichard, P-L Lions, and J-M Morel. Axiomes et equations fondamentales du traitement d'images.(analyse multiechelle et edp). *Comptes rendus de l'Académie des sciences. Série 1, Mathématique*, 315(2):135–138, 1992.

[4] IK Argyros, MA Hernández-Verón, and MJ Rubio. On the convergence of secant-like methods. *Current Trends in Mathematical Analysis and Its Interdisciplinary Applications*, pages 141–183, 2019.

[5] Larry Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of mathematics*, 16(1):1–3, 1966.

[6] Ben Beklisi Kwame Ayawli, Albert Yaw Appiah, Isaac Kofi Nti, Frimpong Kyeremeh, and Esinam Irene Ayawli. Path planning for mobile robots using morphological dilation voronoi diagram roadmap algorithm. *Scientific African*, 12:e00745, 2021.

[7] Yuhan Bai. Relu-function and derived function review. In *SHS Web of Conferences*, volume 144, page 02006. EDP Sciences, 2022.

[8] Manpreet Singh Bajwa, Arun Prakash Agarwal, and Sumati Manchanda. Ternary search algorithm: Improvement of binary search. In *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 1723–1725. IEEE, 2015.

[9] Peter Bankhead. Analyzing fluorescence microscopy images with imagej. *ImageJ*, 1(195):10–1109, 2014.

[10] Heinz Bauschke and Walaa Moursi. On the douglas–rachford algorithm for solving possibly inconsistent optimization problems. *Mathematics of Operations Research*, 03 2023.

[11] Heinz H Bauschke, Patrick L Combettes, Heinz H Bauschke, and Patrick L Combettes. Fermat's rule in convex optimization. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*, pages 497–514, 2017.

[12] Stephen P Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[13] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[14] Roger W Brockett and Petros Maragos. Evolution equations for continuous-scale morphological filtering. *IEEE transactions on Signal Processing*, 42(12):3377–3386, 1994.

[15] A Enis Cetin, Alican Bozkurt, Osman Gunay, Yusuf Hakan Habiboglu, Kivanc Kose, Ibrahim Onaran, Mohammad Tofighi, and Rasim Akin Sevimli. Projections onto convex sets (pocs) based optimization by lifting. In *2013 IEEE Global Conference on Signal and Information Processing*, pages 623–623. IEEE, 2013.

[16] Laurent Condat. A generic proximal algorithm for convex optimization—application to total variation minimization. *IEEE Signal Processing Letters*, 21(8):985–989, 2014.

[17] Rafael Correa, Alejandro Jofre, and Lionel Thibault. Characterization of lower semicontinuous convex functions. *Proceedings of the American Mathematical Society*, 116(1):67–72, 1992.

[18] Dake. Bisection method to find zeroes of a function. https://fr.wikipedia.org/wiki/Utilisateur:Dake, 2005. Drawn with Inkscape 0.42. GFDL.

[19] Dawson-Haggerty et al. trimesh.

[20] Mark Dowell and Peter Jarratt. A modified regula falsi method for computing the root of an equation. 1971.

[21] Raffaele Gaetano, G. Chierchia, and Beatrice Pesquet. Parallel implementations of a disparity estimation algorithm based on a proximal splitting method. pages 1–6, 11 2012.

[22] Jens Goronzy, Sophia Blum, Albrecht Hartmann, Verena Plodeck, Lea Franken, Klaus-Peter Günther, and Falk Thielemann. Is mri an adequate replacement for ct scans in the three-dimensional assessment of acetabular morphology? *Acta radiologica*, 60(6):726–734, 2019.

[23] 3D graphics 101. The utah teapot, 2014. CC Attribution-NonCommercial License.

[24] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[25] Allan Hanbury. The morphological top-hat operator generalised to multi-channel images. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 1, pages 672–675. IEEE, 2004.

[26] Eldon Hansen and Merrell Patrick. A family of root finding methods. *Numerische mathematik*, 27(3):257–269, 1976.

[27] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[28] Gehad Hassan, Nashwa El-Bendary, Aboul Ella Hassanien, Ali Fahmy, Vaclav Snasel, et al. Retinal blood vessel segmentation approach based on mathematical morphology. *Procedia Computer Science*, 65:612–622, 2015.

[29] Zhichun Huang, Shaojie Bai, and J Zico Kolter. Implicit layers for implicit representations. *Advances in Neural Information Processing Systems*, 34:9639–9650, 2021.

[30] Alfredo N Iusem, Benar Fux Svaiter, and Marc Teboulle. Entropy-like proximal methods in convex programming. *Mathematics of Operations Research*, 19(4):790–814, 1994.

[31] Kenji Kawaguchi. On the theory of implicit deep learning: Global convergence with implicit layers. *arXiv preprint arXiv:2102.07346*, 2021.

[32] Minkyu Kim, Hyun-Soo Choi, and Jinho Kim. Explicit feature interaction-aware graph neural network. *IEEE Access*, 2024.

[33] Alexandre Kirszenberg, Guillaume Tochon, Elodie Puybareau, and Jesus Angulo. Going beyond p-convolutions to learn grayscale morphological operators. 02 2021.

[34] Steven George Krantz and Harold R Parks. *The implicit function theorem: history, theory, and applications*. Springer Science & Business Media, 2002.

[35] Hugues Talbot Laurent Najman, Jean-Christophe Pesquet. When Convex Analysis Meets Mathematical Morphology on Graphs. *HAL Science Ouverte*, 2015.

[36] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[37] Bernard Lemaire. The proximal algorithm. *International series of numerical mathematics*, 87:73–87, 1989.

[38] Charles A Micchelli, Lixin Shen, and Yuesheng Xu. Proximity algorithms for image models: denoising. *Inverse Problems*, 27(4):045009, 2011.

[39] Laurent Najman and Hugues Talbot. *Mathematical morphology: from theory to applications*. John Wiley & Sons, 2013.

[40] Feiping Nie, Shiming Xiang, and Changshui Zhang. Neighborhood minmax projections. In *IJCAI*, pages 993–998. Citeseer, 2007.

[41] Mohammad Noroozi, Hamed Mohammadi, Emad Efatinasab, Ali Lashgari, Mahdiyeh Eslami, and Baseem Khan. Golden search optimization algorithm. *IEEE Access*, 10:37515–37532, 2022.

[42] Olegalexandrov and Pbroks13. Vector, Public domain. Public domain, via Wikimedia Commons.

[43] Travis E Oliphant et al. *Guide to numpy*, volume 1. Trelgol Publishing USA, 2006.

[44] Ivo FD Oliveira and Ricardo HC Takahashi. An enhancement of the bisection method average performance preserving minmax optimality. *ACM Transactions on Mathematical Software (TOMS)*, 47(1):1–24, 2020.

[45] PAR. Goldensectionsearch, 2005. PAR, CC BY-SA 3.0, via Wikimedia Commons.

[46] Hochong Park and Roland T. Chin. Decomposition of arbitrarily shaped morphological structuring elements. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(1):2–15, 1995.

[47] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[48] Jean-Christophe Pesquet Patrick L. Combettes. Proximal Splitting Methods in Signal Processing. *Cornell University*, 2010.

[49] Jean-Christophe Pesquet and Nelly Pustelnik. A parallel inertial proximal optimization method. *Pacific Journal of Optimization*, 8(2):273–305, 2012.

[50] Henry Pinkard, Kaitlin Corbin, and Matthew F Krummel. Spatiotemporal rank filtering improves image quality compared to frame averaging in 2-photon laser scanning microscopy. *PloS one*, 11(3):e0150430, 2016.

[51] William H Press, Saul A Teukolsky, and William T Vetterling. *Numerical recipes with source code CD-ROM 3rd edition: the art of scientific computing*. Cambridge university press, 2007.

[52] Jean-Francois Rivest, Pierre Soille, and Serge Beucher. Morphological gradients. *Journal of Electronic Imaging*, 2(4):326–336, 1993.

[53] Nicholas E Ross, Charles J Pritchard, David M Rubin, and Adriano G Duse. Automated image processing method for the diagnosis and classification of malaria on thin blood smears. *Medical and Biological Engineering and Computing*, 44:427–436, 2006.

[54] Abdurakhmon Sadiev, Aleksandr Beznosikov, Pavel Dvurechensky, and Alexander Gasnikov. Zeroth-order algorithms for smooth saddle-point problems. In *International Conference on Mathematical Optimization Theory and Operations Research*, pages 71–85. Springer, 2021.

[55] Lloyd J Sartor and Arthur R Weeks. Morphological operations on color images. *Journal of Electronic Imaging*, 10(2):548–559, 2001.

[56] Krzysztof Sikorski. Bisection is optimal. *Numerische Mathematik*, 40:111–117, 1982.

[57] Pierre Soille. On morphological operators based on rank filters. *Pattern recognition*, 35(2):527–535, 2002.

[58] Ravi Srisha and Am Khan. Morphological operations for image processing : Understanding and its applications. 12 2013.

[59] Trojan.chess. Mnist dataset comparison. https://creativecommons.org/licenses/by-sa/4.0, 2024. CC BY-SA 4.0, via Wikimedia Commons.

[60] Rein Van Den Boomgaard and Arnold Smeulders. The morphological structure of images: The differential equations of morphological scale-space. *IEEE transactions on pattern analysis and machine intelligence*, 16(11):1101–1113, 1994.

[61] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[62] VladimKosobokov. Virus, 2020. CC Attribution License.

[63] Eric W Weisstein. Newton's method. *https://mathworld. wolfram. com/*, 2002.

[64] Michael Young. Ternary search. The Computer Science Handbook.

[65] Hyun Yu and Rodney Roberts. Morphological image segmentation for co-aligned multiple images using watersheds transformation [electronic resource] /.

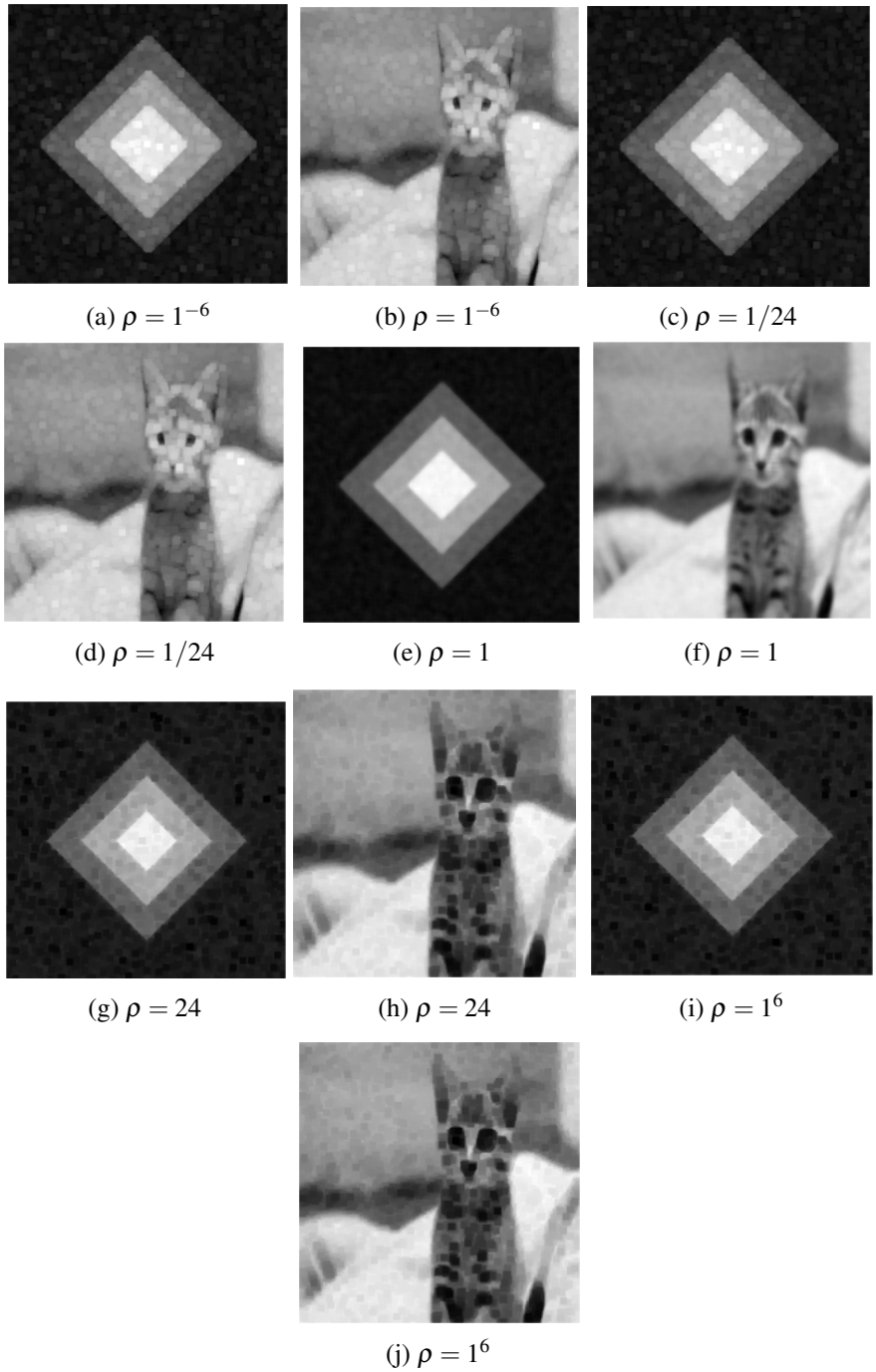
[66] Huimin Zhao, Rui Yao, Ling Xu, Yu Yuan, Guangyu Li, and Wu Deng. Study on a novel fault damage degree identification method using high-order differential mathematical morphology gradient spectrum entropy. *Entropy*, 20(9):682, 2018.

Appendix A

Demonstrations of different methods developed

This appendix exclusively contains visual demonstrations of the methods developed and discussed in Chapter 3. Each section showcases practical implementations through images, providing clear visual representations of the outcomes achieved. These demonstrations serve to illustrate the application and effectiveness of the methods discussed in the main text. Each method is presented with annotated images that highlight key features and results, offering readers a comprehensive overview of the practical aspects of the methodologies introduced in Chapter 3.

A.1 Demonstration of Bisection Approach



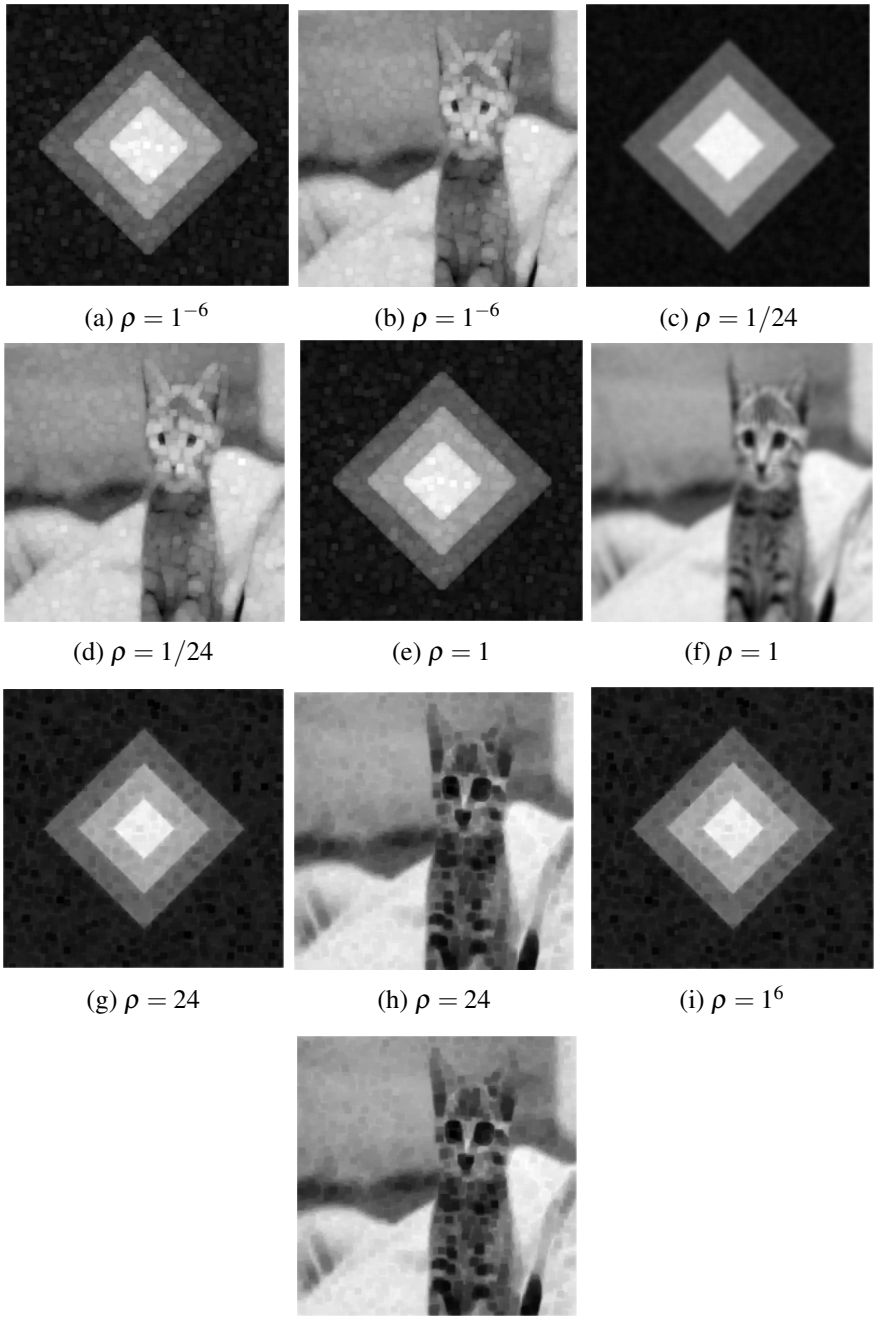
(a) $\rho = 1^{-6}$ (b) $\rho = 1^{-6}$ (c) $\rho = 1/24$

(d) $\rho = 1/24$ (e) $\rho = 1$ (f) $\rho = 1$

(g) $\rho = 24$ (h) $\rho = 24$ (i) $\rho = 1^{6}$

(j) $\rho = 1^{6}$

Figure A.1: Demonstration of Bisection root finding implementation while $p = 1.5$ and U_i of square 7x7 on images 3.1 and 3.2



(a) $\rho = 1^{-6}$ (b) $\rho = 1^{-6}$ (c) $\rho = 1/24$

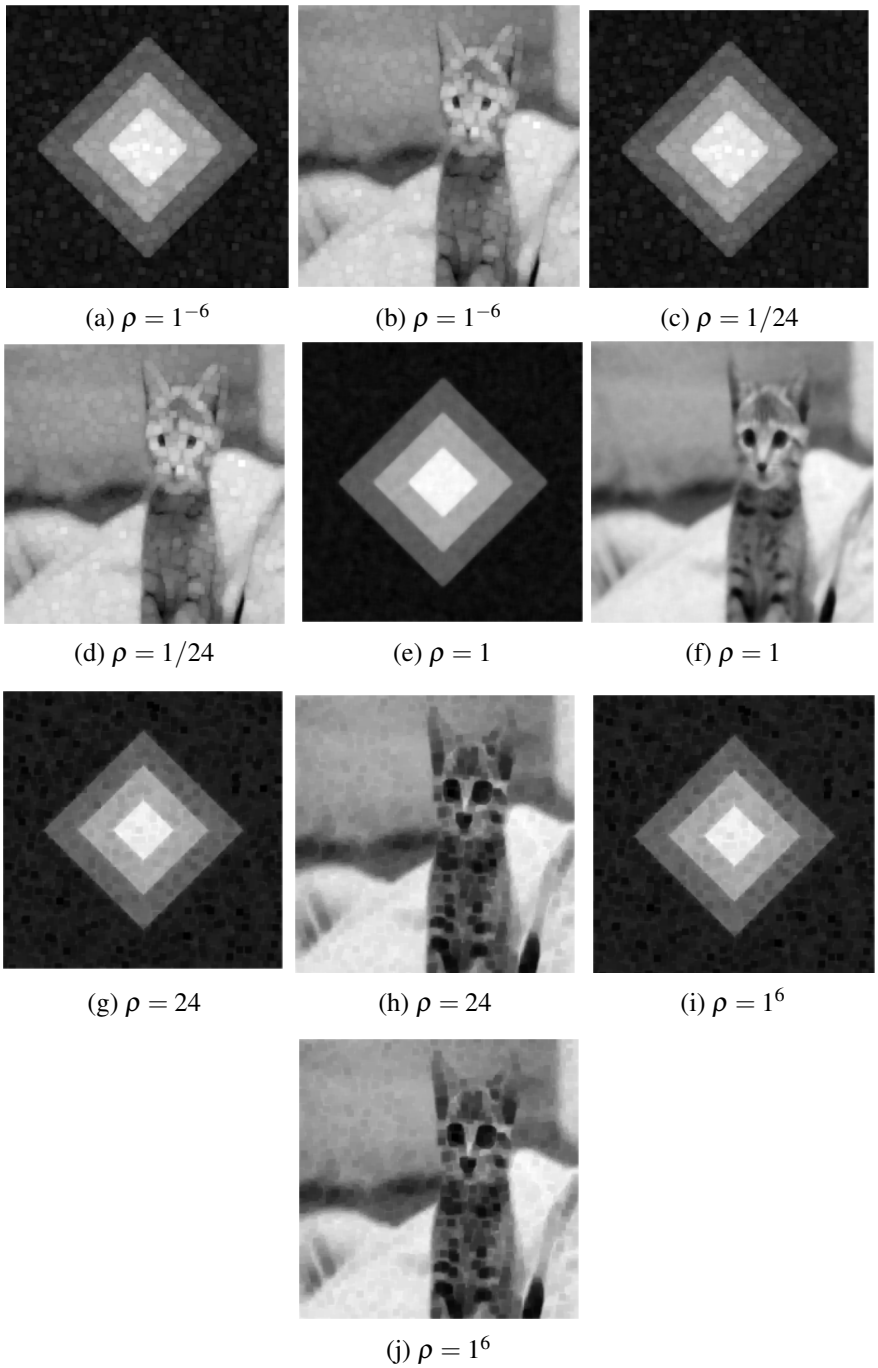
(d) $\rho = 1/24$ (e) $\rho = 1$ (f) $\rho = 1$

(g) $\rho = 24$ (h) $\rho = 24$ (i) $\rho = 1^{6}$

(j) $\rho = 1^{6}$

Figure A.2: Demonstration of Bisection root finding implementation while $p = 1.8$ and U_i of square 7x7 on images 3.1 and 3.2

A.2 Demonstration of ITP Approach



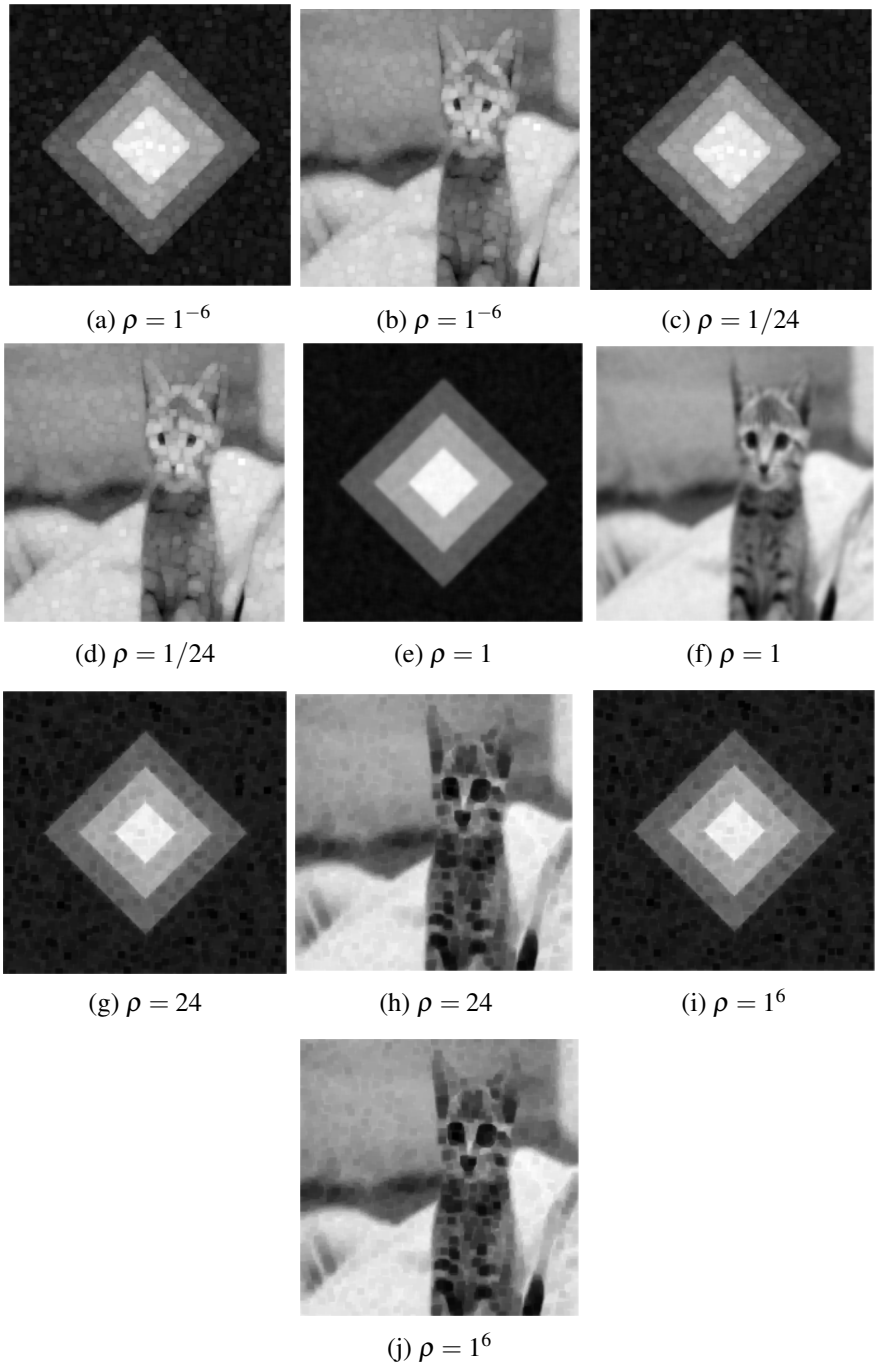
(a) $\rho = 1^{-6}$ (b) $\rho = 1^{-6}$ (c) $\rho = 1/24$

(d) $\rho = 1/24$ (e) $\rho = 1$ (f) $\rho = 1$

(g) $\rho = 24$ (h) $\rho = 24$ (i) $\rho = 1^{6}$

(j) $\rho = 1^{6}$

Figure A.3: Demonstration of ITP root finding implementation while $p = 1.2$ and U_i of square 7x7 on images 3.1 and 3.2



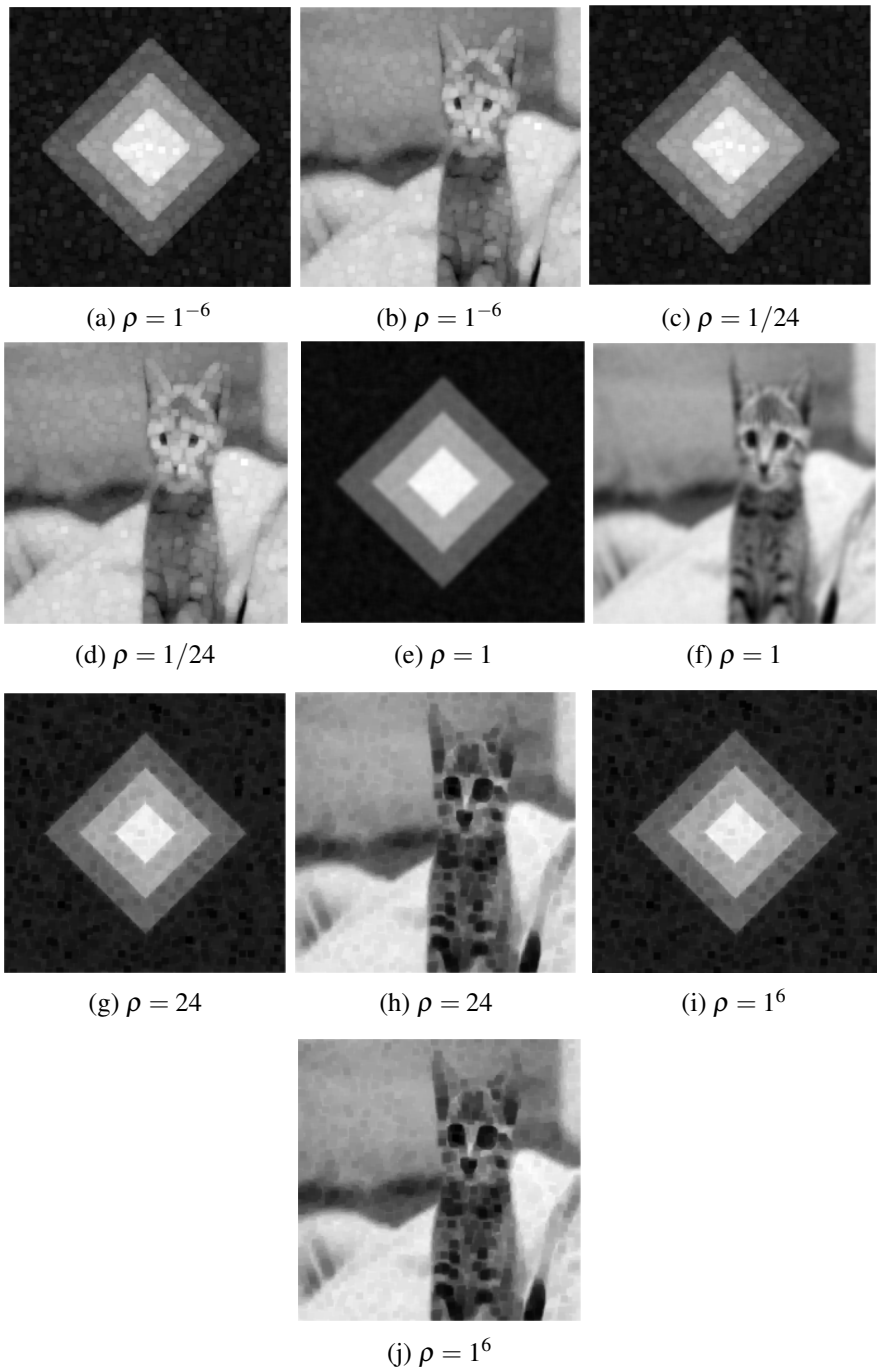
(a) $\rho = 1^{-6}$ (b) $\rho = 1^{-6}$ (c) $\rho = 1/24$

(d) $\rho = 1/24$ (e) $\rho = 1$ (f) $\rho = 1$

(g) $\rho = 24$ (h) $\rho = 24$ (i) $\rho = 1^{6}$

(j) $\rho = 1^{6}$

Figure A.4: Demonstration of ITP root finding implementation while $p = 1.5$ and U_i of square 7x7 on images3.1 and 3.2



(a) $\rho = 1^{-6}$ (b) $\rho = 1^{-6}$ (c) $\rho = 1/24$

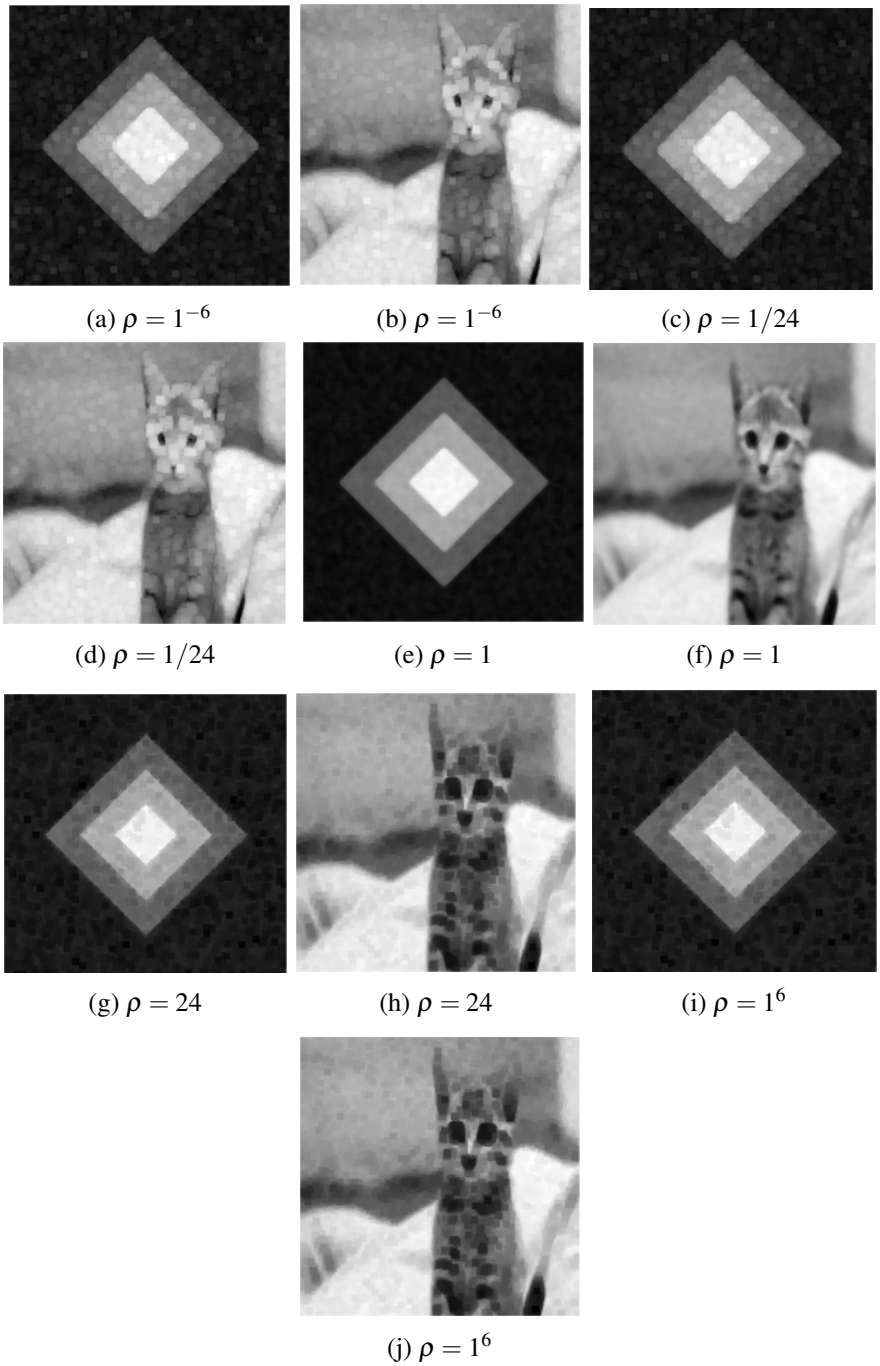
(d) $\rho = 1/24$ (e) $\rho = 1$ (f) $\rho = 1$

(g) $\rho = 24$ (h) $\rho = 24$ (i) $\rho = 1^{6}$

(j) $\rho = 1^{6}$

Figure A.5: Demonstration of ITP root finding implementation while $p = 1.8$ and U_i of square 7x7 on images 3.1 and 3.2

A.3 Demonstration of Golden-Section Search Approach



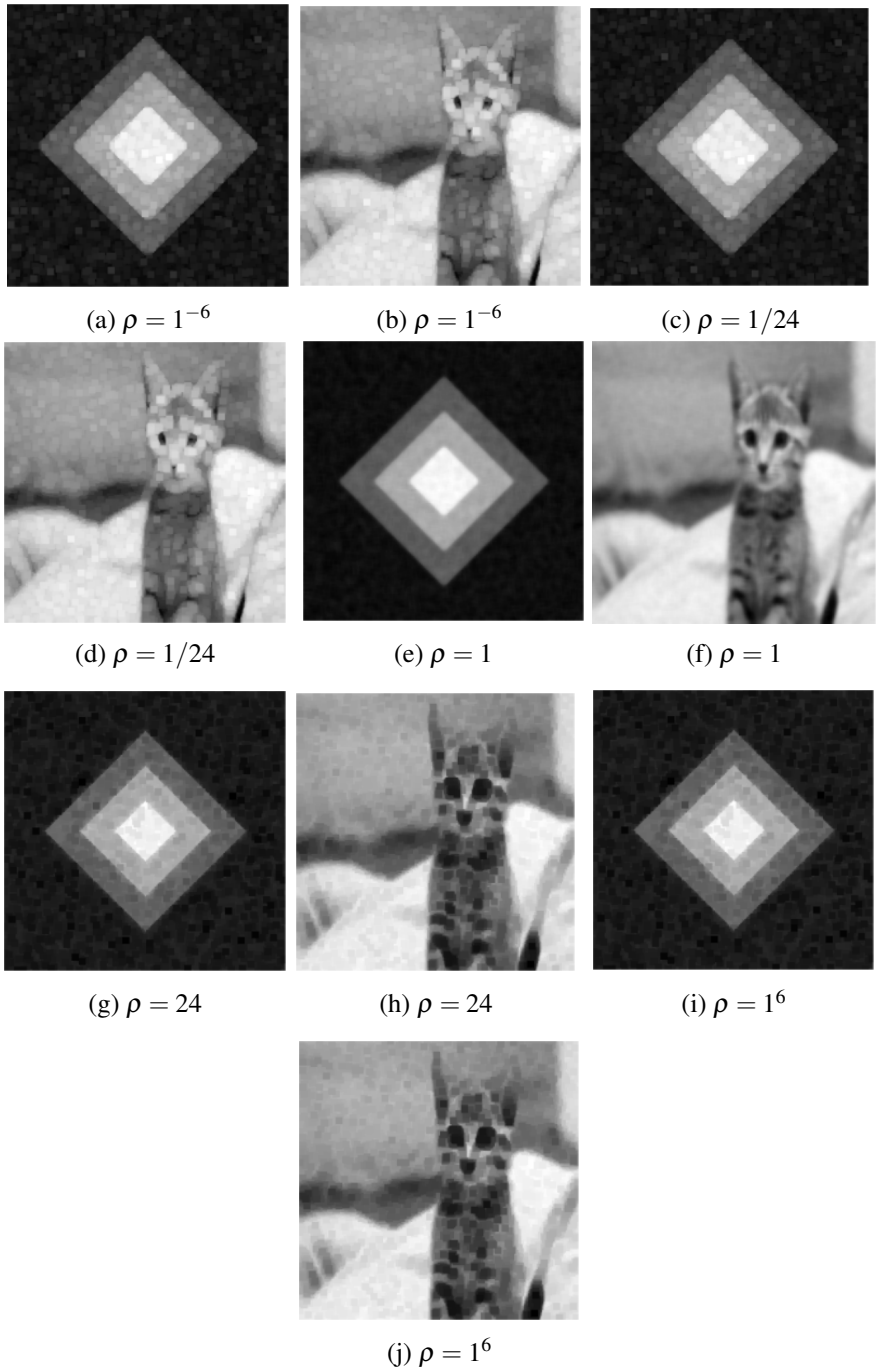
(a) $\rho = 1^{-6}$ (b) $\rho = 1^{-6}$ (c) $\rho = 1/24$

(d) $\rho = 1/24$ (e) $\rho = 1$ (f) $\rho = 1$

(g) $\rho = 24$ (h) $\rho = 24$ (i) $\rho = 1^{6}$

(j) $\rho = 1^{6}$

Figure A.6: Demonstrations of Golden-Section Search zeroth order implementation while $p = 1.2$ and U_i of square 7x7 on images 3.1 and 3.2



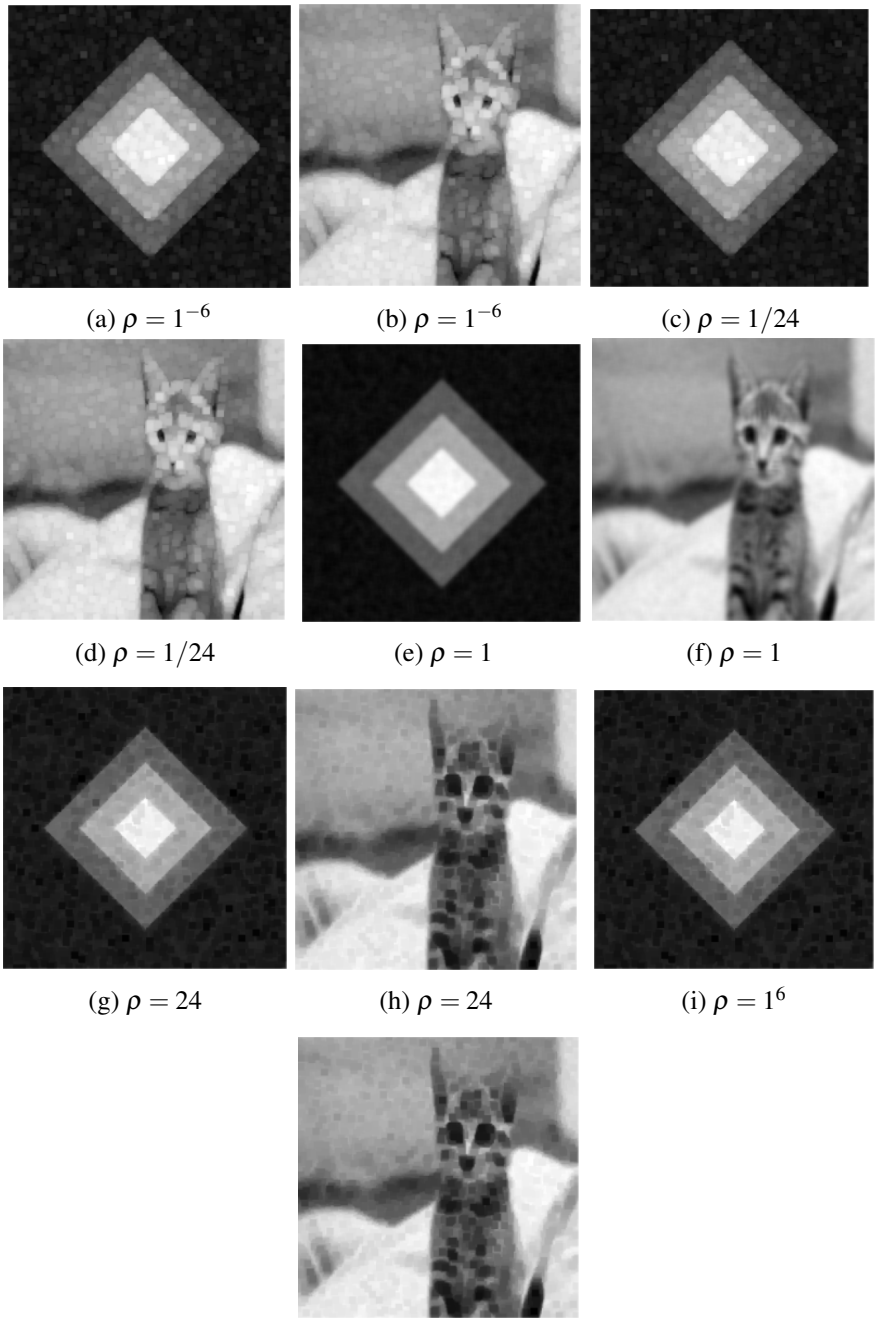
(a) $\rho = 1^{-6}$ (b) $\rho = 1^{-6}$ (c) $\rho = 1/24$

(d) $\rho = 1/24$ (e) $\rho = 1$ (f) $\rho = 1$

(g) $\rho = 24$ (h) $\rho = 24$ (i) $\rho = 1^{6}$

(j) $\rho = 1^{6}$

Figure A.7: Demonstrations of Golden-Section Search zeroth order implementation while $p = 1.5$ and U_i of square 7x7 on images 3.1 and 3.2.



(a) $\rho = 1^{-6}$ (b) $\rho = 1^{-6}$ (c) $\rho = 1/24$

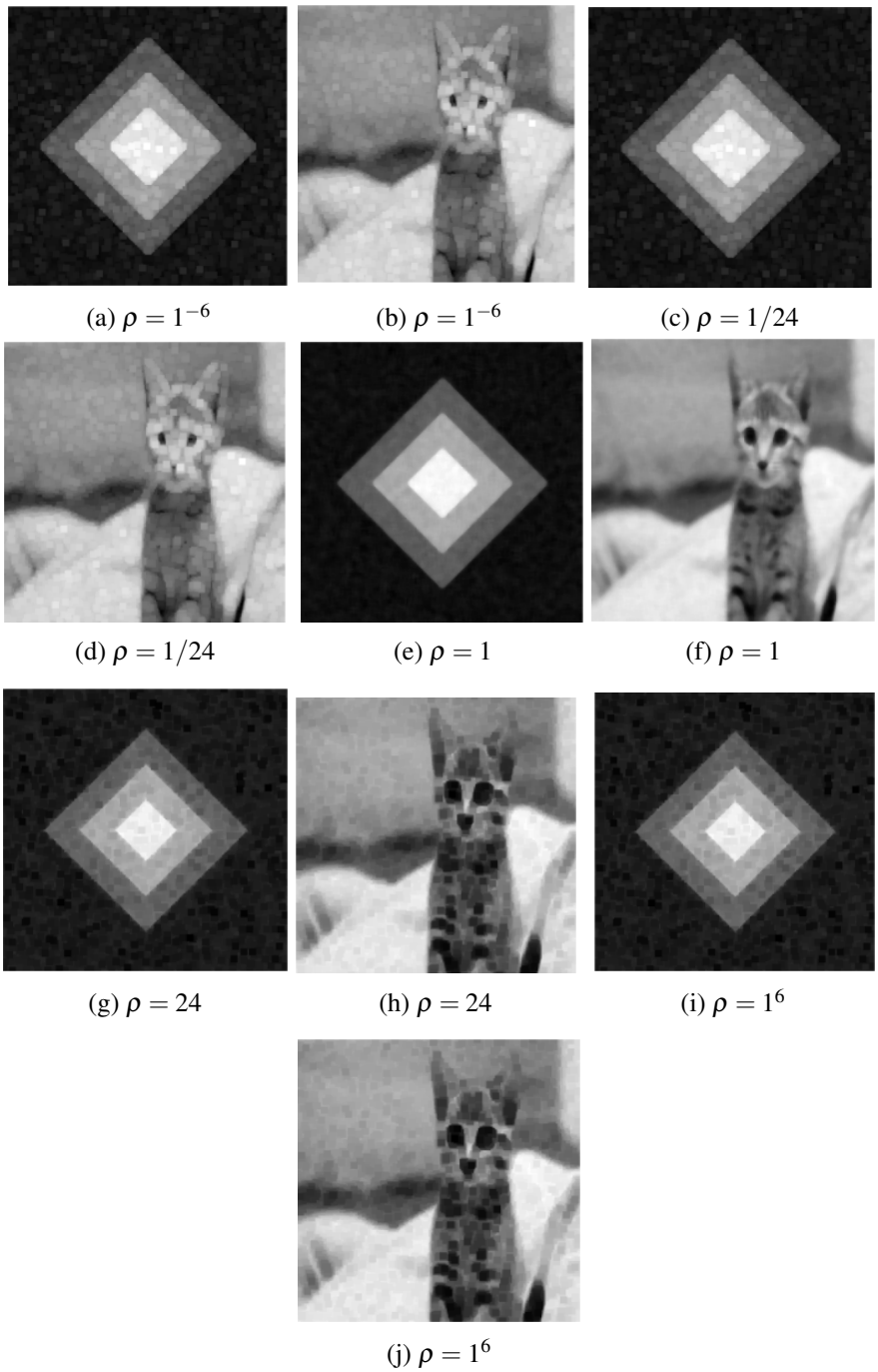
(d) $\rho = 1/24$ (e) $\rho = 1$ (f) $\rho = 1$

(g) $\rho = 24$ (h) $\rho = 24$ (i) $\rho = 1^{6}$

(j) $\rho = 1^{6}$

Figure A.8: Demonstrations of Golden-Section Search zeroth order implementation while $p = 1.8$ and U_i of square 7x7 on images 3.1 and 3.2.

A.4 Demonstration of Ternary Search Approach



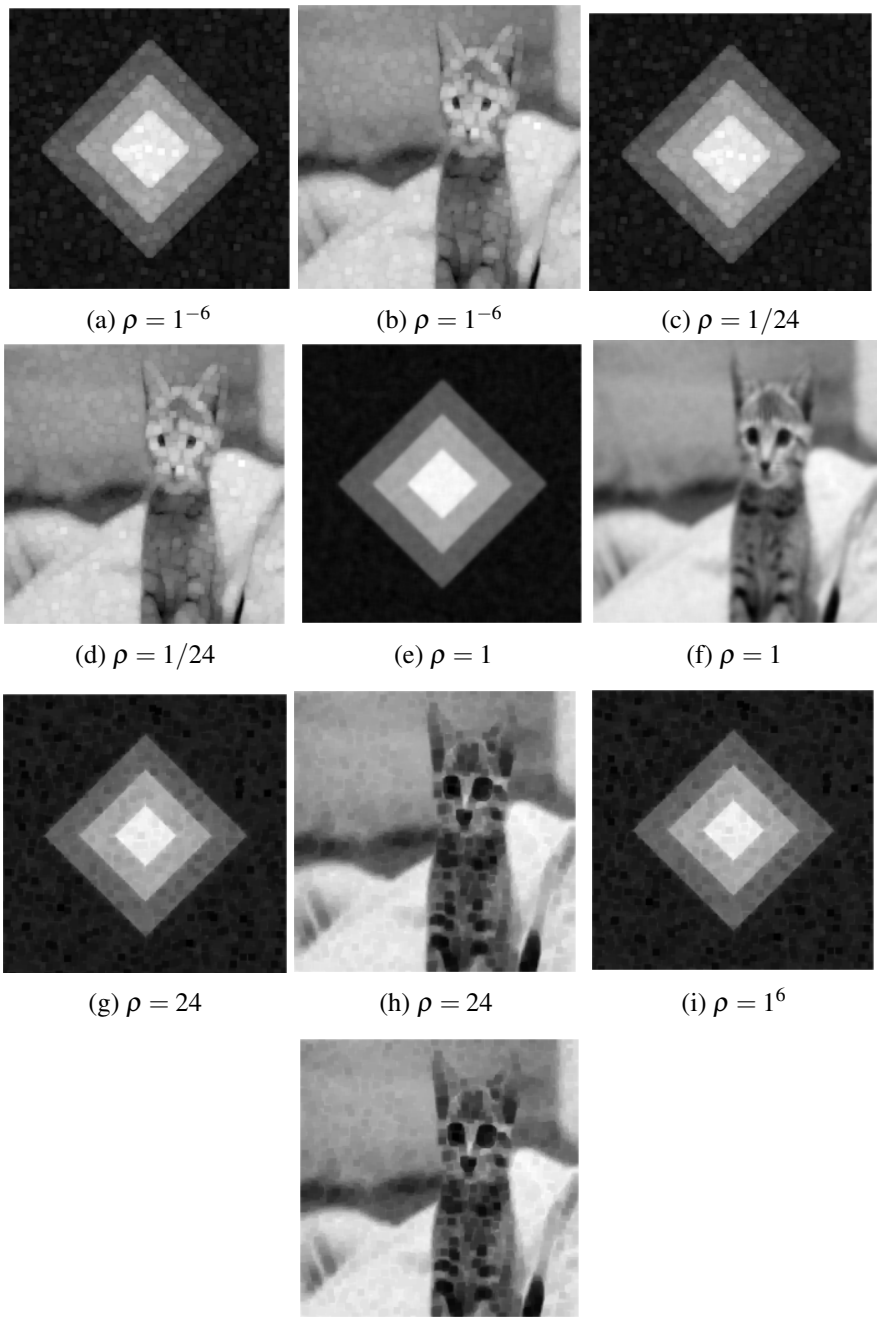
(a) $\rho = 1^{-6}$ (b) $\rho = 1^{-6}$ (c) $\rho = 1/24$

(d) $\rho = 1/24$ (e) $\rho = 1$ (f) $\rho = 1$

(g) $\rho = 24$ (h) $\rho = 24$ (i) $\rho = 1^{6}$

(j) $\rho = 1^{6}$

Figure A.9: Demonstration of Ternary Search zeroth order implementation while $p = 1.2$ and U_i of square 7x7 on images 3.1 and 3.2.



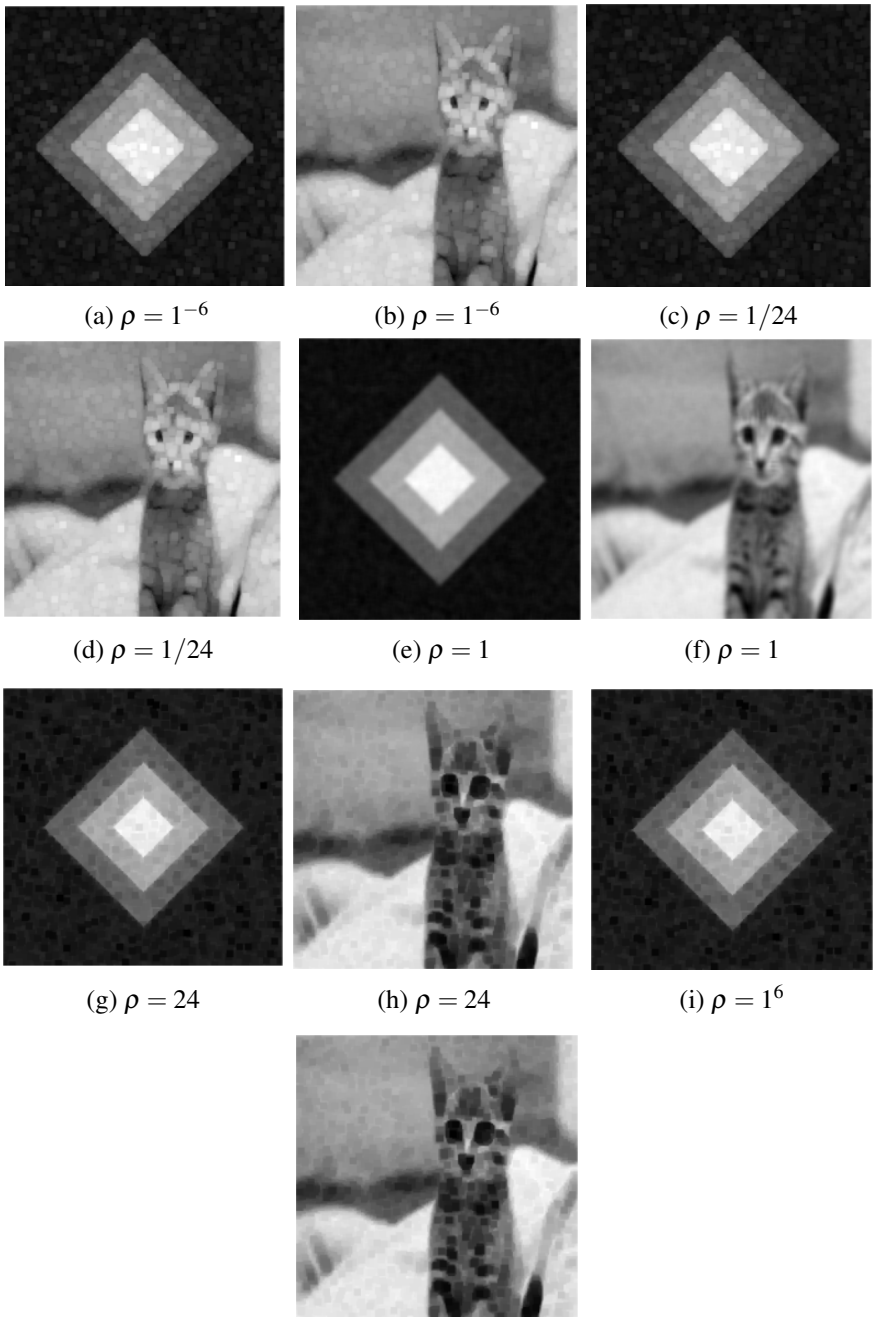
(a) $\rho = 1^{-6}$ (b) $\rho = 1^{-6}$ (c) $\rho = 1/24$

(d) $\rho = 1/24$ (e) $\rho = 1$ (f) $\rho = 1$

(g) $\rho = 24$ (h) $\rho = 24$ (i) $\rho = 1^{6}$

(j) $\rho = 1^{6}$

Figure A.10: Demonstration of Ternary Search zeroth order implementation while $p = 1.5$ and U_i of square 7x7 on images 3.1 and 3.2.



(a) $\rho = 1^{-6}$ (b) $\rho = 1^{-6}$ (c) $\rho = 1/24$

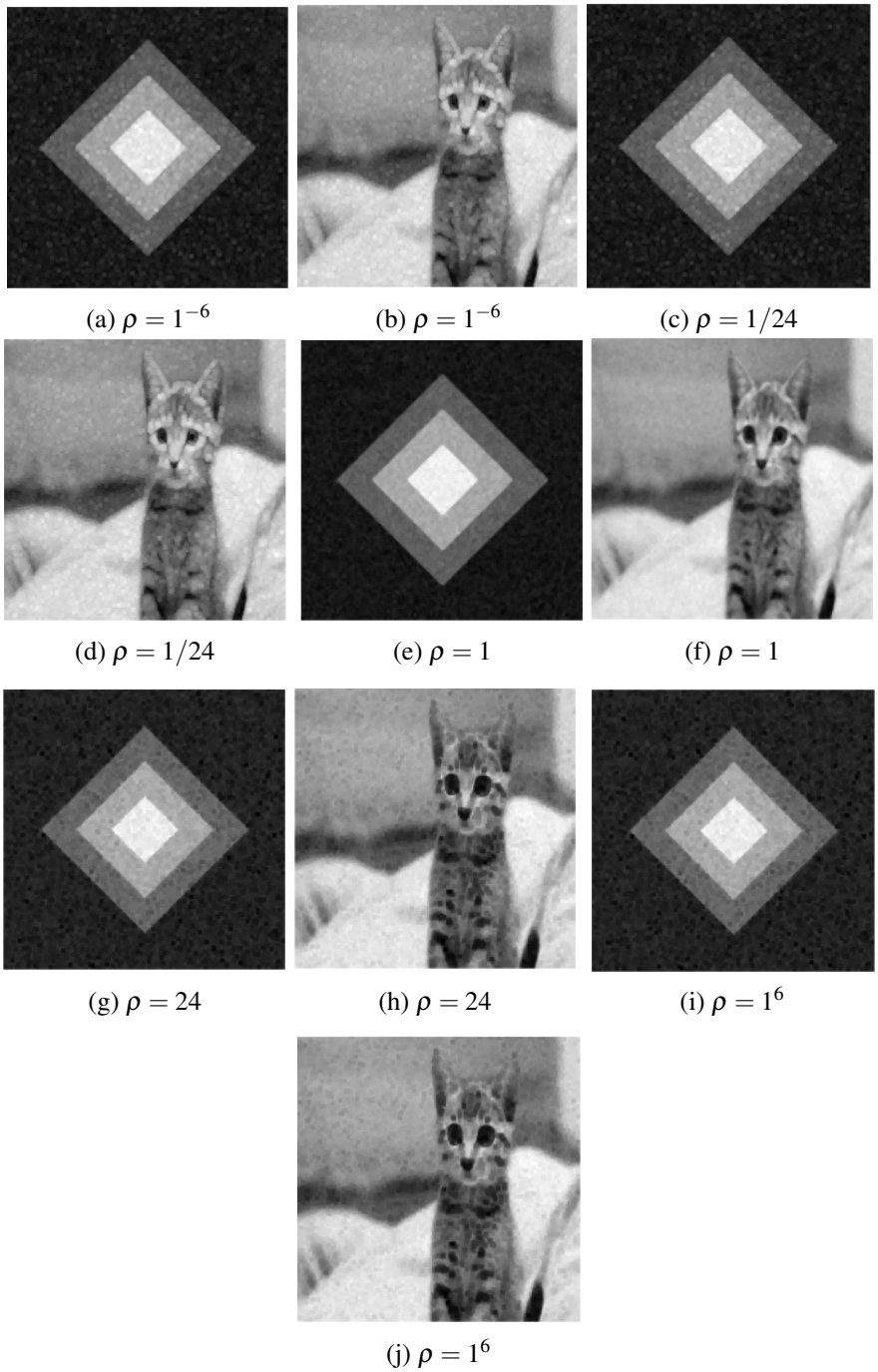
(d) $\rho = 1/24$ (e) $\rho = 1$ (f) $\rho = 1$

(g) $\rho = 24$ (h) $\rho = 24$ (i) $\rho = 1^{6}$

(j) $\rho = 1^{6}$

Figure A.11: Demonstration of Ternary Search zeroth order implementation while $p = 1.8$ and U_i of square 7x7 on images 3.1 and 3.2.

A.5 Demonstration of PPXA Approach



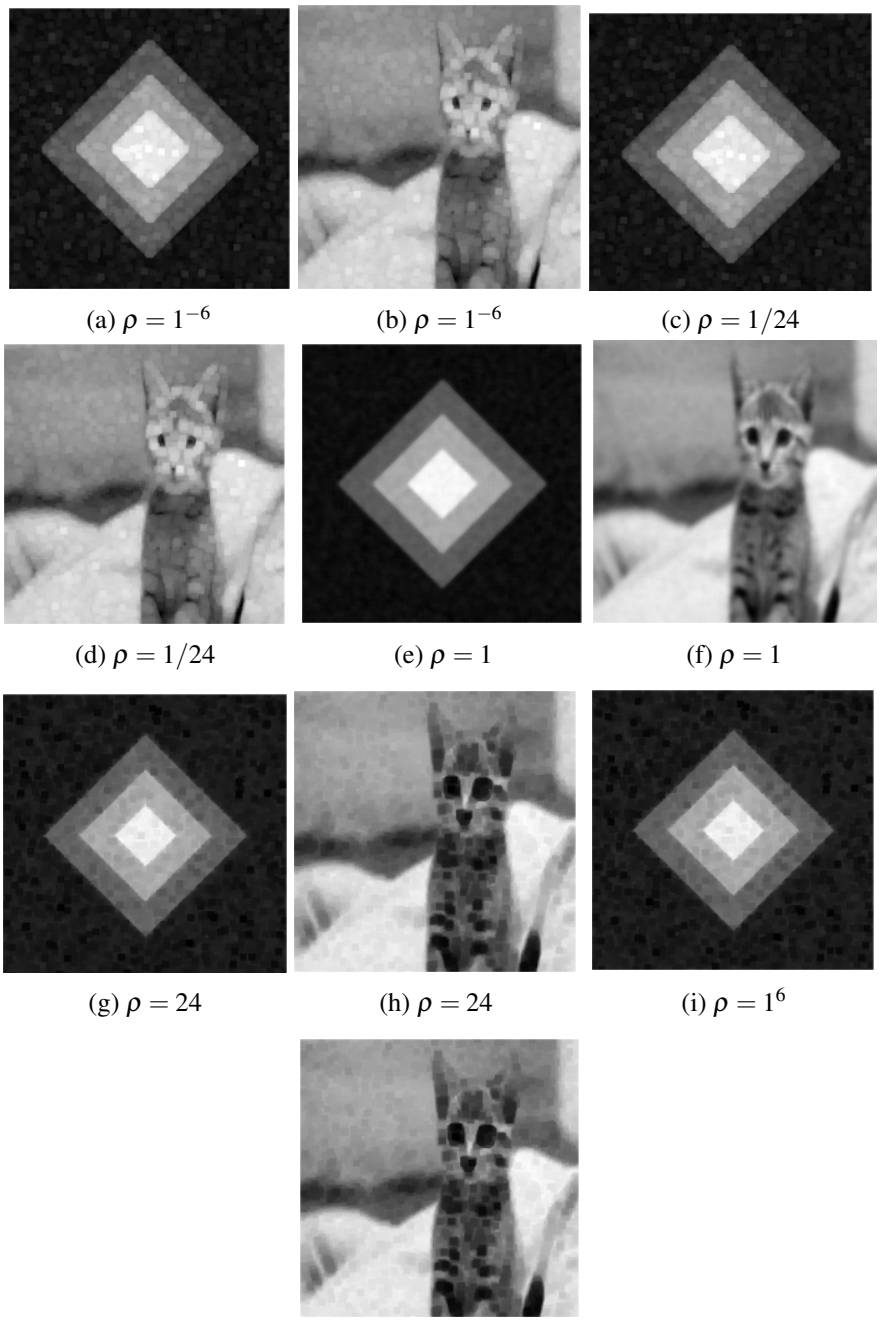
(a) $\rho = 1^{-6}$ (b) $\rho = 1^{-6}$ (c) $\rho = 1/24$

(d) $\rho = 1/24$ (e) $\rho = 1$ (f) $\rho = 1$

(g) $\rho = 24$ (h) $\rho = 24$ (i) $\rho = 1^{6}$

(j) $\rho = 1^{6}$

Figure A.12: Demonstrations of PPXA implementation while $p = 1.2$ and U_i of square 3x3 on images 3.1 and 3.2.



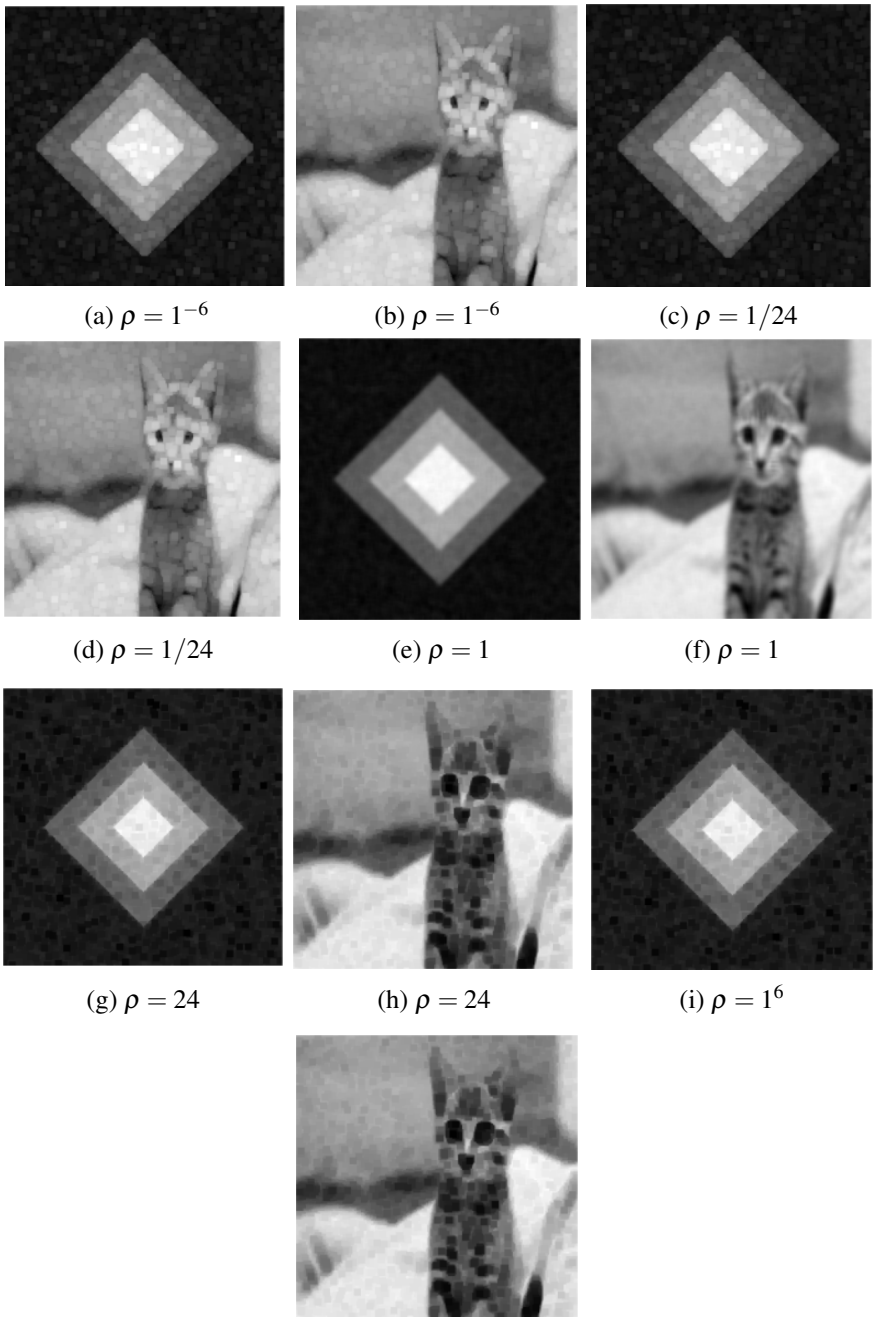
(a) $\rho = 1^{-6}$ (b) $\rho = 1^{-6}$ (c) $\rho = 1/24$

(d) $\rho = 1/24$ (e) $\rho = 1$ (f) $\rho = 1$

(g) $\rho = 24$ (h) $\rho = 24$ (i) $\rho = 1^{6}$

(j) $\rho = 1^{6}$

Figure A.13: Results of implementation of PPXA implementation while $p = 1.5$ and U_i of square 3x3 on images 3.1 and 3.2.



(a) $\rho = 1^{-6}$ (b) $\rho = 1^{-6}$ (c) $\rho = 1/24$

(d) $\rho = 1/24$ (e) $\rho = 1$ (f) $\rho = 1$

(g) $\rho = 24$ (h) $\rho = 24$ (i) $\rho = 1^{6}$

(j) $\rho = 1^{6}$

Figure A.14: Results of implementation of PPXA implementation while $p = 1.8$ and U_i of square 3x3 on images 3.1 and 3.2.

Appendix B

Convergence Algorithms Testing

To address the challenge of comparing the different methods developed with this project, tests were made to check convergence. Convergence testing involves comparing the results of different methods to ensure they all converge to similar optimal values. Even though we cannot compare the outcomes to a known ground truth, we expect that all robust and correctly implemented methods will find similar optima when applied to the same problem. By comparing the convergence behavior and final results of various methods, we can validate that each method is functioning correctly and reliably converging towards an optimal solution. The following images illustrate the test of convergence made: for each method, is tested by how close the output of every other method is based on l1, l2 and infinity norm difference.

By comparing these norms, we have shown that despite the specific nature of variational MM operations, all methods converge to similar optimal values. This convergence is crucial for validating the accuracy and reliability of our methods. Additionally, the performance testing ensured that the methods are not only accurate but also efficient, making them suitable for practical applications. In these tests, all operations use a U_i of a square size 3x3 and $\rho = 1/4$.

B.1 Bisection Method

- TEST WITH ITP - $p = 1.5$
  - **Inf Norm**: $1.4210854715202004 \times 10^{-14}$
  - **L2 Norm**: $6.888967415955462 \times 10^{-14}$
  - **L1 Norm**: $5.115907697472721 \times 10^{-13}$
- TEST WITH GOLDEN_SEARCH - $p = 1.5$
  - **Inf Norm**: $1.1892013560554915 \times 10^{-06}$
  - **L2 Norm**: $4.36583906644577 \times 10^{-06}$
  - **L1 Norm**: $6.403350819539355 \times 10^{-05}$
- TEST WITH TERNARY_SEARCH - $p = 1.5$

  - **Inf Norm**: $8.20619561636704 \times 10^{-07}$
  - **L2 Norm**: $3.1614767571437927 \times 10^{-06}$
  - **L1 Norm**: $4.658561879011813 \times 10^{-05}$

- TEST WITH PPXA - $p = 1.5$
  - **Inf Norm**: $1.4779288903810084 \times 10^{-12}$
  - **L2 Norm**: $1.0697090505847408 \times 10^{-11}$
  - **L1 Norm**: $1.8072654484058148 \times 10^{-10}$

These results highlight the remarkable precision of the PPXA and ITP methods, which show differences close to the order of 10^{-14} and 10^{-12} respectively. The Golden-section Search and Ternary Search methods also demonstrate solid performance with differences in the range of 10^{-6}.

B.2 ITP Method

- TEST WITH BISECTION - $p = 1.5$
  - **Inf Norm**: $2.842170943040401 \times 10^{-14}$
  - **L2 Norm**: $7.418283949015283 \times 10^{-14}$
  - **L1 Norm**: $5.186961971048731 \times 10^{-13}$
- TEST WITH GOLDEN_SEARCH - $p = 1.5$
  - **Inf Norm**: $9.181002695868301 \times 10^{-07}$
  - **L2 Norm**: $4.347653155335114 \times 10^{-06}$
  - **L1 Norm**: $6.761779040687088 \times 10^{-05}$
- TEST WITH TERNARY - $p = 1.5$
  - **Inf Norm**: $7.383183913134417 \times 10^{-07}$
  - **L2 Norm**: $3.3133939188398476 \times 10^{-06}$
  - **L1 Norm**: $4.68744265660348 \times 10^{-05}$
- TEST WITH PPXA - $p = 1.5$
  - **Inf Norm**: $1.6200374375330284 \times 10^{-12}$
  - **L2 Norm**: $1.1213145579864289 \times 10^{-11}$
  - **L1 Norm**: $1.8753354424916324 \times 10^{-10}$

These results highlight the remarkable precision of the PPXA and Bisection methods, which show differences close to the order of 10^{-14} and 10^{-12} respectively. The Golden-section Search and Ternary Search methods also demonstrate solid performance with differences in the range of 10^{-6}.

B.3 Golden-Section Search Method

- TEST WITH BISECTION - $p = 1.5$
  - **Inf Norm**: $8.421572488259699 \times 10^{-07}$
  - **L2 Norm**: $4.272749980794265 \times 10^{-06}$
  - **L1 Norm**: $6.691820806992155 \times 10^{-05}$
- TEST WITH ITP - $p = 1.5$
  - **Inf Norm**: $8.421572346151152 \times 10^{-07}$
  - **L2 Norm**: $4.2727499801537045 \times 10^{-06}$
  - **L1 Norm**: $6.691820806281612 \times 10^{-05}$
- TEST WITH TERNARY_SEARCH - $p = 1.5$
  - **Inf Norm**: $8.835344829094538 \times 10^{-07}$
  - **L2 Norm**: $4.735316549235579 \times 10^{-06}$
  - **L1 Norm**: $7.41732214919466 \times 10^{-05}$
- TEST WITH PPXA - $p = 1.5$
  - **Inf Norm**: $8.421578598927226 \times 10^{-07}$
  - **L2 Norm**: $4.272754377118525 \times 10^{-06}$
  - **L1 Norm**: $6.691826601468165 \times 10^{-05}$

These results highlight the remarkable precision of the PPXA and ITP methods, which show differences close to the order of 10^{-7} and 10^{-7} respectively. The Golden-section Search and Ternary Search methods also demonstrate solid performance with differences in the range of 10^{-6}.

B.4 Ternary Search Method

- TEST WITH BISECTION - $p = 1.5$
  - **Inf Norm**: $8.079809390437731 \times 10^{-07}$
  - **L2 Norm**: $3.482975423304626 \times 10^{-06}$
  - **L1 Norm**: $4.876768271344645 \times 10^{-05}$
- TEST WITH ITP - $p = 1.5$
  - **Inf Norm**: $8.079809674654825 \times 10^{-07}$
  - **L2 Norm**: $3.4829754317516425 \times 10^{-06}$
  - **L1 Norm**: $4.8767682692130165 \times 10^{-05}$

- TEST WITH GOLDEN_SEARCH - $p = 1.5$
  - **Inf Norm**: $1.178870945750532 \times 10^{-06}$
  - **L2 Norm**: $4.370176332201267 \times 10^{-06}$
  - **L1 Norm**: $6.714487795989044 \times 10^{-05}$
- TEST WITH PPXA - $p = 1.5$
  - **Inf Norm**: $8.079820759121503 \times 10^{-07}$
  - **L2 Norm**: $3.482980685057956 \times 10^{-06}$
  - **L1 Norm**: $4.876775729911742 \times 10^{-05}$

These results highlight the precision of the methods, with differences close to the order of 10^{-6} for most methods. The Bisection, ITP, and PPXA methods show differences in the range of 10^{-6}, indicating consistent performance across these approaches. The Golden-section Search method demonstrates slightly higher differences, still within 10^{-6}, showcasing robust performance across all tested methods.

B.5 PPXA Method

- TEST WITH BISECTION - $p = 1.5$
  - **Inf Norm**: $1.3926637620897964 \times 10^{-12}$
  - **L2 Norm**: $1.038996202372254 \times 10^{-11}$
  - **L1 Norm**: $1.765272372722393 \times 10^{-10}$
- TEST WITH ITP - $p = 1.5$
  - **Inf Norm**: $1.3926637620897964 \times 10^{-12}$
  - **L2 Norm**: $1.0379234619720916 \times 10^{-11}$
  - **L1 Norm**: $1.7627854731472326 \times 10^{-10}$
- TEST WITH GOLDEN_SEARCH - $p = 1.5$
  - **Inf Norm**: $9.335566915069649 \times 10^{-07}$
  - **L2 Norm**: $4.003240076083606 \times 10^{-06}$
  - **L1 Norm**: $6.0261079326551226 \times 10^{-05}$
- TEST WITH TERNARY_SEARCH - $p = 1.5$
  - **Inf Norm**: $7.678451368064998 \times 10^{-07}$
  - **L2 Norm**: $3.33403540903373 \times 10^{-06}$

– **L1 Norm**: $4.9412557103778454 \times 10^{-05}$

These results highlight the precision of the methods, with differences in the range of 10^{-12} to 10^{-6}. The BISECTION and ITP methods exhibit exceptionally low differences close to 10^{-12}, showcasing their high precision. The Golden-section Search and Ternary Search methods also demonstrate solid performance with differences in the range of 10^{-6}, indicating consistent accuracy across these methods.

Appendix C

Confusion Matrix - Neural Network Study Case

In this section, all confusion matrix are shown for both neural network study cases presented in chapter 4:

Demonstration of Confusion Matrices for all different layers tested in experiment on section 4.2.4:

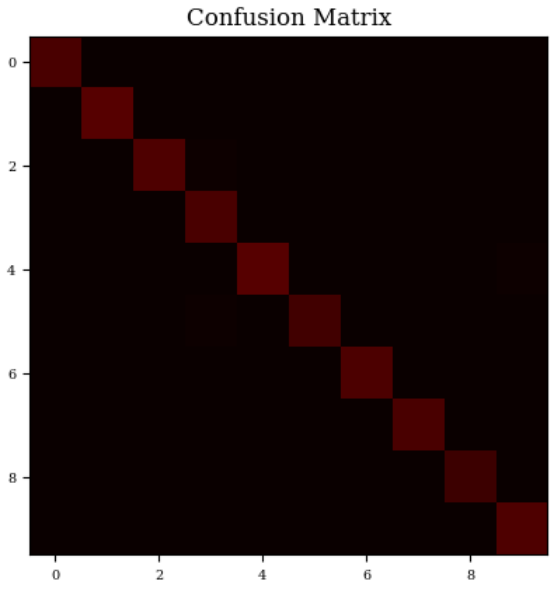


Figure C.1: Confusion Matrix of Convolutional layer.

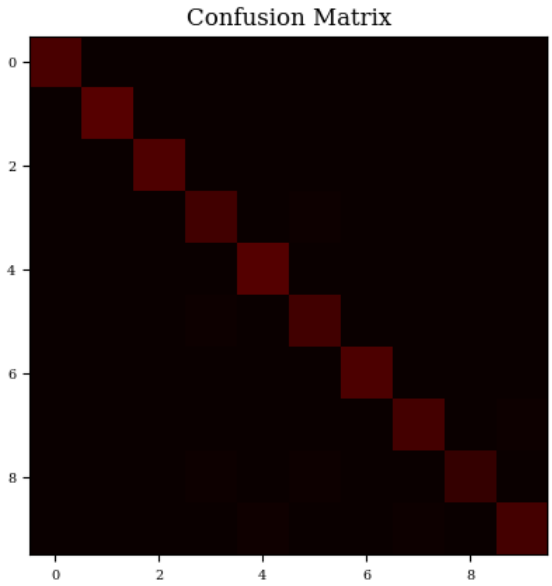


Figure C.2: Confusion Matrix of Variational Dilation layer.

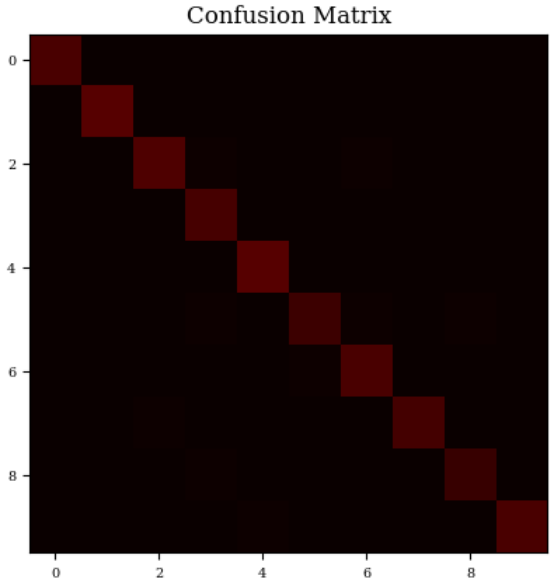


Figure C.3: Confusion Matrix of Variational Erosion layer.

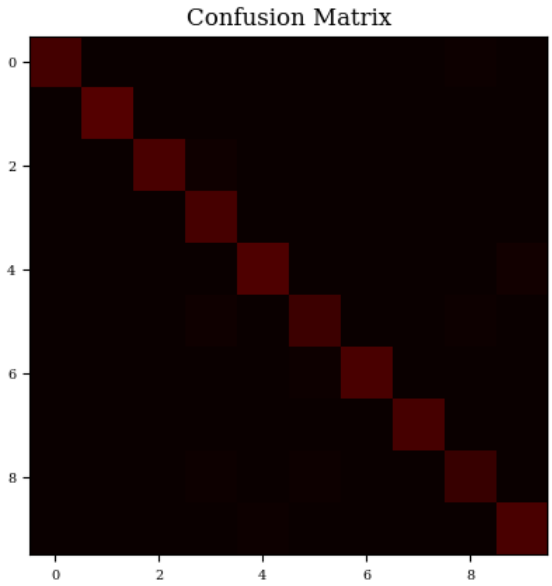


Figure C.4: Confusion Matrix of Variational Opening layer.

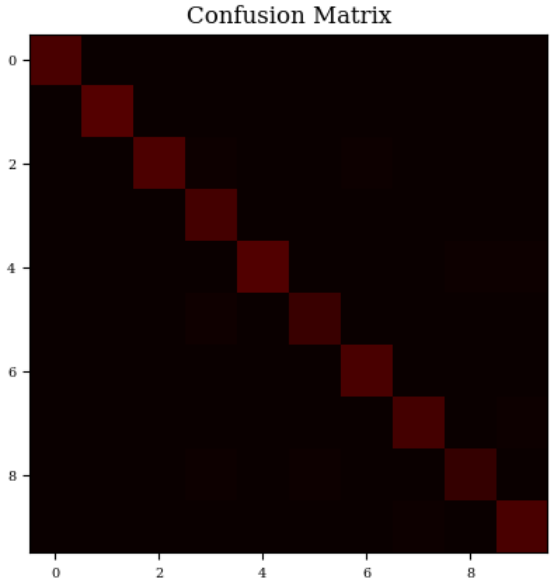


Figure C.5: Confusion Matrix of Variational Closing layer.

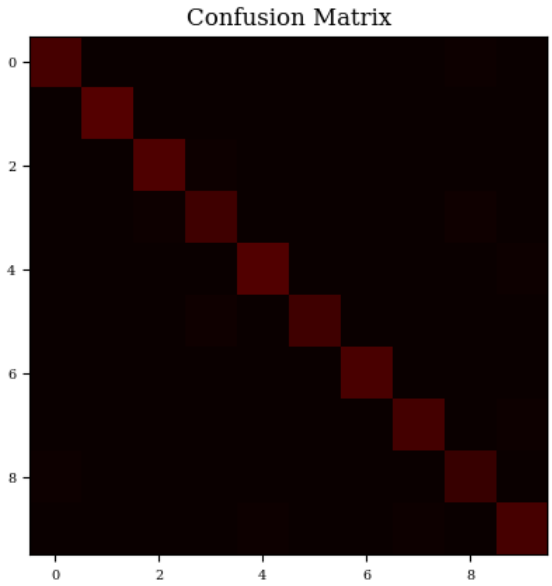


Figure C.6: Confusion Matrix of Variational Gradient layer.

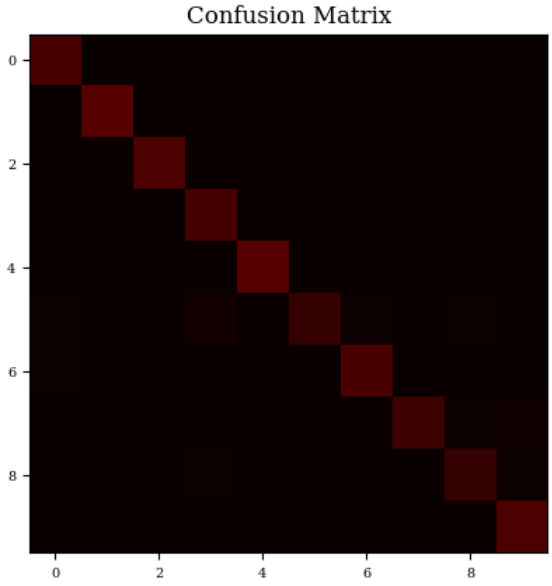


Figure C.7: Confusion Matrix of Variational Internal Gradient layer.

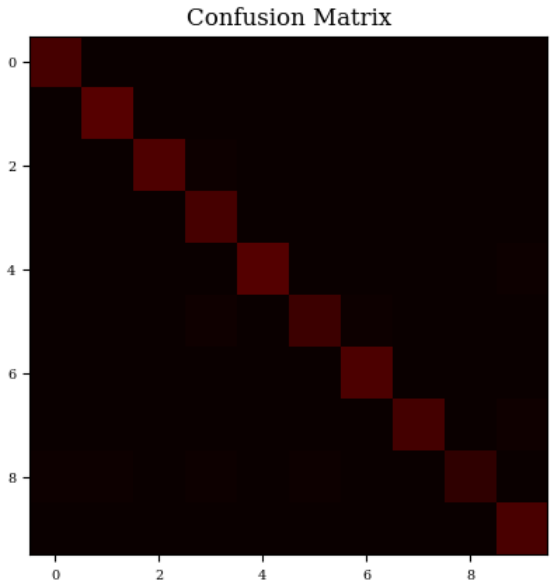


Figure C.8: Confusion Matrix of Variational External Gradient layer.

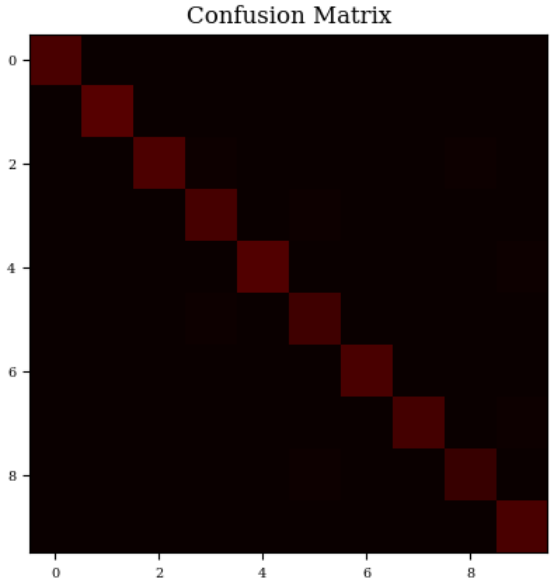


Figure C.9: Confusion Matrix of Variational TopHat Opening layer.

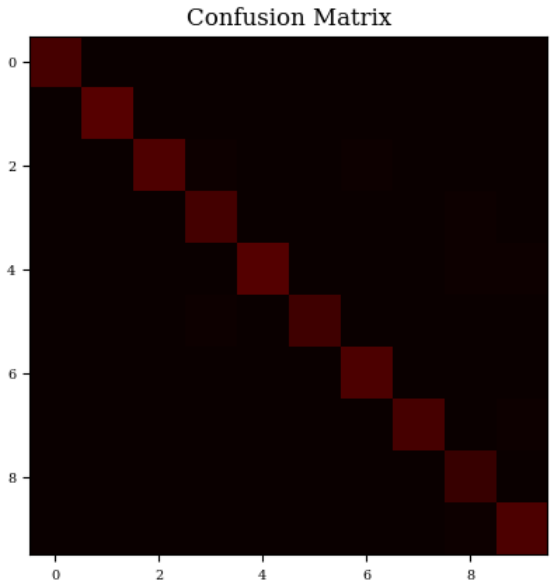


Figure C.10: Confusion Matrix of Variational TopHat Closing layer.

Demonstration of Confusion Matrices for all different layers tested in experiment on section 4.2.5:

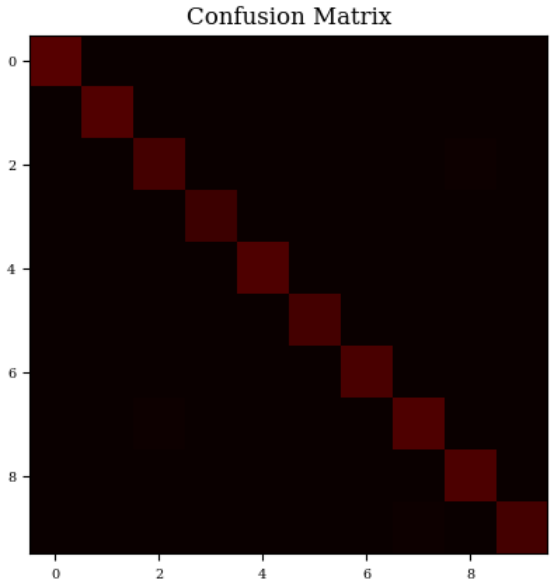


Figure C.11: Confusion Matrix of Convolutional layer.

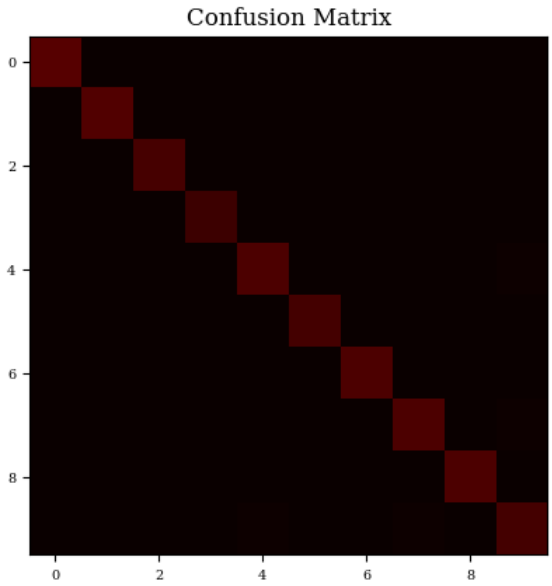


Figure C.12: Confusion Matrix of Variational Dilation layer.

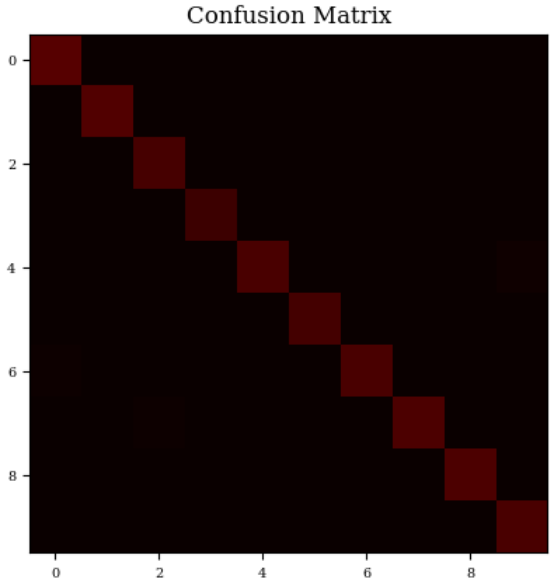


Figure C.13: Confusion Matrix of Variational Erosion layer.

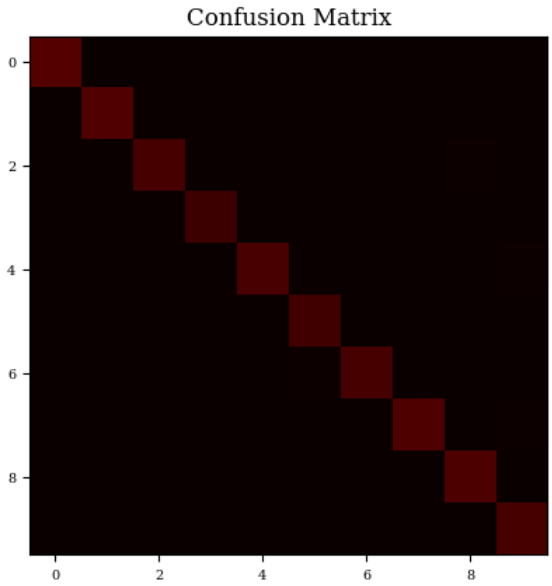


Figure C.14: Confusion Matrix of Variational Opening layer.

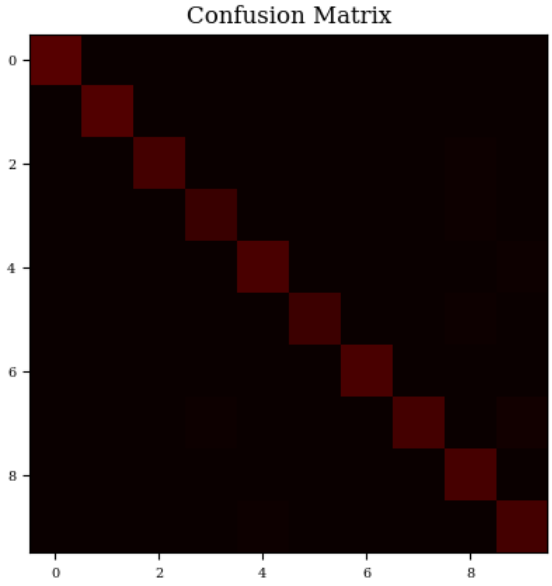


Figure C.15: Confusion Matrix of Variational Closing layer.

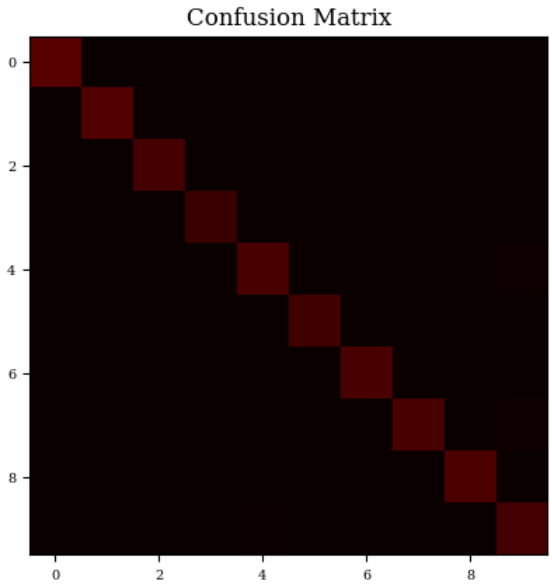


Figure C.16: Confusion Matrix of Variational Gradient layer.

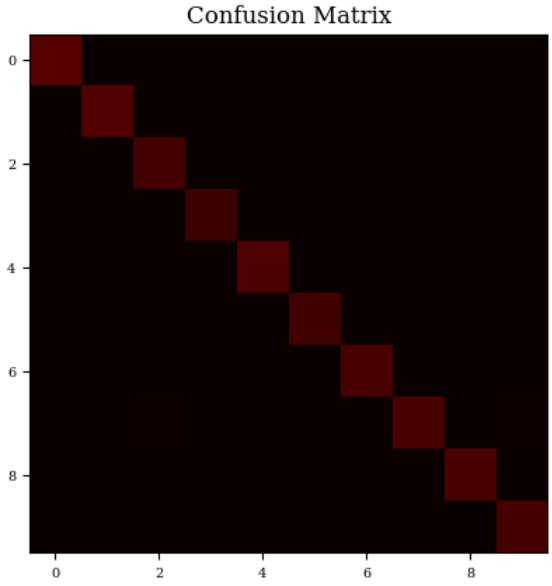


Figure C.17: Confusion Matrix of Variational Internal Gradient layer.

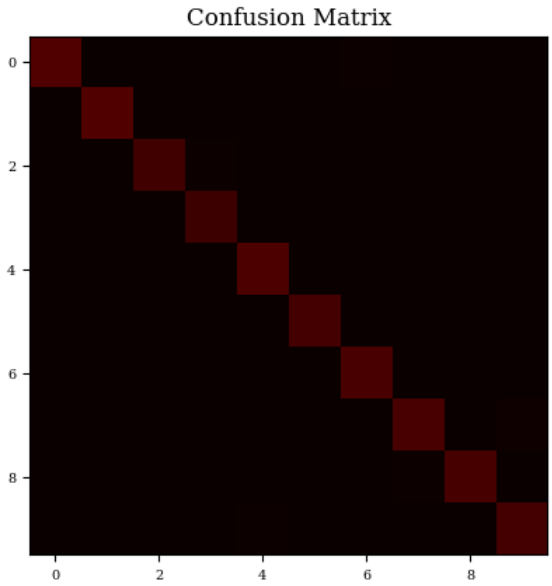


Figure C.18: Confusion Matrix of Variational External Gradient layer.

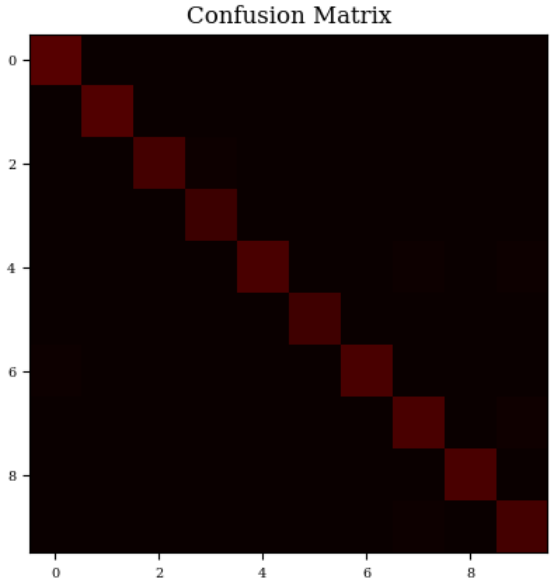


Figure C.19: Confusion Matrix of Variational TopHat Opening layer.

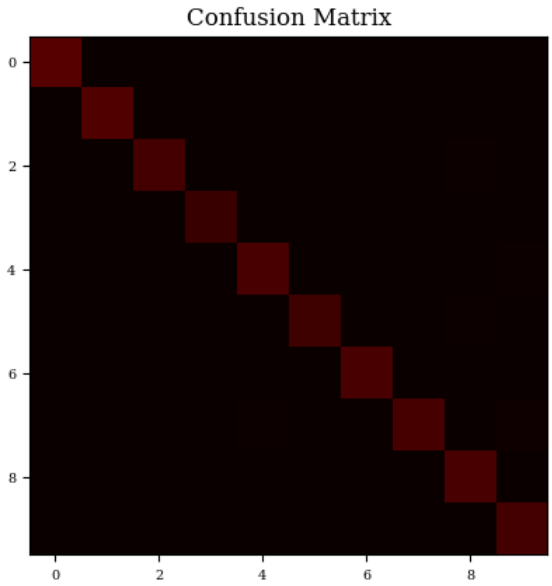


Figure C.20: Confusion Matrix of Variational TopHat Closing layer.