Subwindow Neighborhood aggregation for edge detecting cellular automata

Bachelor's Thesis

Name:	Timon Christiansen
Email:	timon.christiansen@code.berlin
Address:	Weinbergstr. 14, 12555 Berlin

Study Program:	Bachelor of Science (B.Sc.) - Software Engineering
Semester:	Fall Semester 2021
Enrollment Number:	17.03.008

Supervisor One:	Ulrich von Zadow
Supervisor Two:	Fabio Fracassi

Date:	August 12, 2021

Abstract

Cellular Automata (CA) based approaches for Edge Detection have retained a particular interest in the research community due to their simplicity, adjustability, and efficiency. Countless edge detection literature also evaluates noise robustness since image noise is a common problem in digital imagery to this day. This thesis adapts P. L. Rosin's 2006 proposed technique of constructing subwindows within an extended neighborhood to the field of edge detection and evaluates its potential to make CA-based methods more noise-robust. I adapted and modified two CA-based edge detecting algorithms from previous literature using the subwindow neighborhood technique and then tested their performance on a public binary shape image dataset with manually created ground truth versions. I compared the proposed method with classical edge detectors and the two based upon and unmodified CA edge detectors using the five metrics: run time, Pratt's Figure of Merit, Baddeley's Delta Metric, Root-Mean-Squared error, and Peak-Signal-to-Noise ratio. Testing with different noise types and levels proved that subwindow neighborhood aggregation brings noise robustness to edge detecting Cellular Automata without loss in edge quality or run time. Through future development, the proposed method also offers the potential to replace common pairs of denoising functions and edge detectors, for instance, Gaussian blur and the Canny operator.

Key Words: Cellular Automata, Edge detection, Subwindow Neighborhood, Totalistic Cellular Automata

Contents

List of Abbreviations

BDM	Baddeley's Delta Metric
CA	Cellular Automata
CLA	Cellular Learning Automata
OTCA	Outer Totalistic Cellular Automata
PFoM	Pratt's Figure of Merit
PSNR	Peak Signal-to-Noise Ratio
RMSE	Root-Mean-Squared Error
TCA	Totalistic Cellular Automata

1 Introduction

1.1 Motivation

In the field of Image Processing, Cellular Automata (CA) are composed of cells that can change their state at discrete timesteps based upon a set of rules and their neighboring cells. In 2006, professor P. L. Rosin studied CA training for various image processing tasks such as noise filtering, thinning, or convex hulls [Ros06]. He uses a feedback function and a feature selection algorithm to select and combine the best rules for the given task. His set of available rules contains all 51 unique binary patterns in the eight-pixel neighborhood without reflections and symmetries.

Rosin concludes that the results are promising even though the number of possible patterns forms a bottleneck for the training of the CA. In the last section, entitled VI. Conclusions and Discussions, Rosin proposes a new technique for future work that enables a bigger neighborhood without a bigger set of possible neighborhood patterns. To illustrate, he showed a figure of a 24-pixel neighborhood with eight subwindows, seen in Figure 1.1. His idea was to take the majority pixel value of each subwindow and put the result into a smaller eight-pixel neighborhood.

This technique would widen the neighborhood and coarsen the effect of individual pixels inside the neighborhood without resulting in a larger rule search space. Rosin concludes this idea as follows: "However, experiments on both the noise removal and thinning tasks did not demonstrate any improvements in results over the basic 3×3 neighborhood approach." [Ros06].

One and a half decades later, computers have become so powerful that selecting the best combination of 51 patterns should be less of a problem. However, a larger neighborhood with 24 neighboring pixels and $2^{24} = 16\,777\,216$ possible neighborhood patterns might still take a while. In addition to that, I believe the proposed technique offers other benefits, especially for edge detection. First, through pixel aggregation in subwindows, image noise should have less impact or be mitigated entirely until a certain noise intensity. Second, the subwindow aggregation should not result in slow run time through the time-efficient nature of the underlying Cellular Automata. Therefore, the proposed noise-robust and efficient CA modification has even the potential to perform better and faster than the standard combination of edge detector and image denoising method.

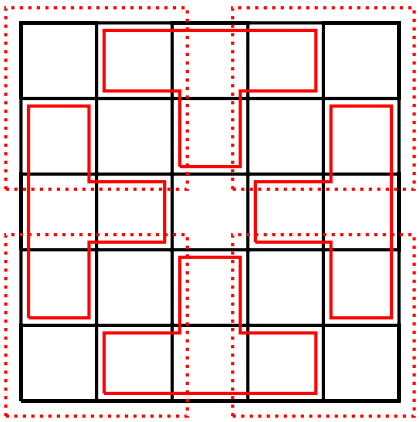

Fig. 1.1: Rosin's subwindow configuration with dashed and solid lines to visualize overlap

In summary, this thesis will test and evaluate the proposed subwindow modification on CA edge detectors from previous literature, proving the three parts of the hypothesis that the proposed subwindow aggregation:

- enhances the noise robustness of edge detection CAs
- does not result in a marginally longer run time
- has the future potential to replace common combinations of standard edge detectors and image denoising methods

First, Section 1.2 will give a small introduction to Cellular Automata. Section 2 presents an overview of the current state of edge detection and Cellular Automata. Followed by Section 3, which outlines the hypothesis and elaborates on the consequences of validating or invalidating it. Section 4 will detail the proposed method's algorithmic implementation and the test metrics. Last, Sections 5 and 6 will report on the achieved test results and discuss their meaning in the current state of research.

1.2 Cellular Automata

1.2.1 Short Introduction

Von Neumann first introduced Cellular Automata (CA) in 1966 to simulate finite state machines [Neu66]. A CA consists of a state, a transition function, a neighborhood, and a boundary condition. For image processing CA, the state is usually a two-dimensional array of cells, or in this case, pixels, where each cell's state represents a pixel value. The set of available cell states in binary images is only zero or one, while for grayscale images, the group of possible states

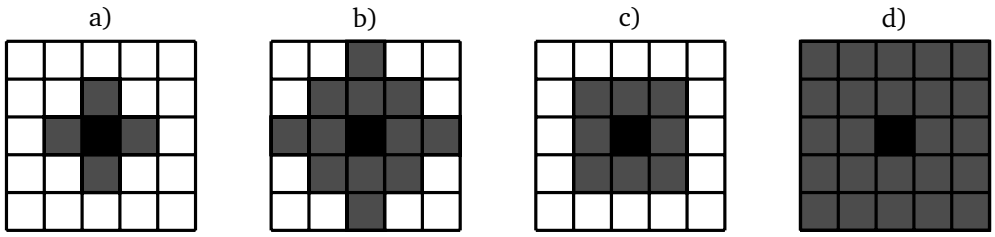

Fig. 1.2: Standard neighborhoods: a) von Neumann, b) extended von Neumann, c) Moore, d) extended Moore

would range from zero to 255. A CA works with discrete time intervals, and for every time interval, the transition function defines the next state of a cell based on the cell's neighborhood. The neighborhood of a cell is often a set of cells in close vicinity (see Figure 1.2 for common neighborhoods). The boundary condition provides fallback values for neighboring cells that do not exist because the inspected cell is at the edge of the CA.

1.2.2 Different Modifications and Types

Over the years, researchers have proposed many different CA modifications or labels for different types of CA. This paragraph gives an overview of the most relevant and prominent variations.

Iterations Convey's Game of Life [ZCZ20], famous in Computer Science, is a CA implementation with no limitations on the number of iterations that it simulates. However, we have to define stopping criteria for CA used in image processing to know which iteration we define as the output of our algorithm. Here we can find two different strategies for CA: fixed number of iterations and dynamic number of iterations. As the name suggests, the former uses a fixed and predefined number of iterations, while the ladder has a stopping criterion that leads to varying iterations. One of the stopping criteria for varying iterations is the converging criteria where the process is complete as soon as the last two states of the CA match. After two equal states, the CA has settled into a static state.

Rule Finding Finding the perfect set of rules for a specific image processing task can be quite hard. Many researchers approach this phase manually or via brute force. However, with the increasing interest in genetic algorithms, evolutionary CA introduced the automated rule finding process with a training dataset, a set of rules to choose from, and a feedback function.

Rule Application Uniform and Hybrid CA define two types where the former applies the same transition function to every cell, and the ladder allows different rules for different cells.

Rule Framework I like to call the basis on which the transition function defines the next state the rule framework. One central rule framework, which I will refer to as count-based, uses the sum inside neighborhoods for the transition function. For instance, Conway's Game of Life [ZCZ20] is likely the most famous CA utilizing a count-based ruleset. The other central framework, which could be called pattern-based, is especially interesting for edge detection. CA using this rule framework decide based on the pattern of alive cells in the neighborhood, giving them, for instance, the ability to differentiate between alive neighbors in the top left or top right corner. There are a few papers covering pattern-based CA for edge detection [AZZ16; Ros06; Ros10]. Besides pattern-based and count-based CA, countless other variations exist and do not fall into any of these groups.

TCA and OTCA Totalistic Cellular Automata (TCA) and Outer Totalistic Cellular Automata (OTCA) are two types of count-based CA that I will continually mention in this research paper. TCA decide the cell's next state based on the count of alive cells in the neighborhood, including the central cell. OTCA, on the other hand, separate the count of the neighborhood and the central cell's state. This gives the central cell more weight in determining the next state. Given a standard Moore neighborhood, TCA have a set of ten possible rules (zero to nine alive neighbors) while OTCA have 18 (zero to eight alive neighbors for each central cell state).

Neighborhoods The research community has proposed many different neighborhoods for various applications. The most commonly used are the Moore neighborhood, the von Neumann neighborhood [Neu66], and their extended variants which can be seen in Figure 1.2. Other neighborhood types also include pixels from previous iterations [Pri+17] or use different neighborhoods at different iterations [ZCZ20; Mof+15].

2 Related Work

This chapter will first review the standard and well-known edge detectors Canny, Sobel, Prewitt, and Roberts, followed by a literature review of Cellular Automata's state-of-the-art in edge detection.

2.1 Standard Edge Detectors

Roberts

The Roberts cross operator was one of the first edge detectors and was proposed in 1963 [Rob63]. It convolves the image using the two 2×2 kernels shown in Equation 2.1 where G_x results from convoluting with the first matrix and G_y results from the second. Combining these two using the following Equation 2.2 results in the gradient image G visualizing the edges.

$$G_x = \begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix} \quad \text{and} \quad G_y = \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix} \tag{2.1}$$

$$G = \sqrt{G_x^2 + G_y^2} \tag{2.2}$$

Sobel

Sobel & Feldman proposed two 3×3 matrices for edge detection [SF68], which meant a slightly lower performance due to the bigger size but a very well-defined matrix center because of the odd dimensions.

The Sobel matrices are defined in Equation 2.3 and combining them using the Equation 2.2 from Roberts Cross yields the gradient image G.

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \quad \text{and} \quad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.3}$$

Prewitt

The Prewitt Operator, proposed by Judith Prewitt in 1970 [Pre70], is similar to the Sobel operator but does not place a stronger weight on the pixels in the current row/column. The matrices used for convolving the image are hence defined as follows in Equation 2.4. The rest is equal to the method of Sobel.

$$G_y = \begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix} \quad \text{and} \quad G_y = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{2.4}$$

Canny

The Canny edge detector was developed 16 years later in a 1986 paper by John Canny [Can86]. In his groundbreaking paper, he first expressed three criteria for good edge detection, which are:

- low error rate: there should be as few false positive and false negative detected edges as possible
- good localization: the distance between the detected edges and the actual edges should be as minimal as possible
- unicity: one edge should only generate a single response

Following his edge detection criteria, he proposed his five-step method for edge detection.

1. Apply the Gaussian filter to smoothen noise and improve performance.
2. Calculate the intensity gradient of the image.
3. Apply non-maximum suppression to thin out edges.
4. Use the double threshold technique to identify strong, weak, and non-relevant pixels.
5. Edge Tracking by Hysteris: finalize edge detection by removing all weak edge pixels not connected to at least one strong pixel.

The Canny operator is one the most used edge detectors to this date.

2.2 Edge Detecting Cellular Automata

The study by Wongthanavasu & Sadananda employs most likely the most straightforward CA transition function for edge detection [WS03]. It states that all pixels keep their values except for white pixels where all eight Moore neighbors are white. It is also understandable why this simple rule works. A foreground pixel, represented as a white pixel, can be considered an edge as long as the pixel has one or more black pixels as neighbors. Being located between foreground and background pixels, is the definition of an edge. However, a white pixel surrounded by white pixels can not be part of the edge and should therefore transition to black in the next iteration.

Uguz *et al.* demonstrated the efficiency of their proposed CA-based edge detector over the classical algorithms Canny, Sobel, and Prewitt [USS13]. Through optimizing their algorithm implementation only to contain matrix multiplication, their algorithm was the fastest on all four test images.

In their paper, Mofrad *et al.* proposed a combination of Cellular Automata and Cellular Learning Automata [Mof+15]. A cellular learning operator learns, adapts, and improves their following actions based on a positive or negative reinforcement signal. As a result, the CLA gradually learned to choose one out of two neighborhoods correctly, wherein the first was the Moore neighborhood and the second was the von Neumann neighborhood. The CLA chose for each pixel individually, and the CA then employed the desired neighborhood per pixel. The authors tested the proposed CA with low Gaussian and Salt-and-pepper noise levels and reported promising results for binary and grayscale images.

While the last decade has witnessed increased neural network development, Cellular Neural Networks have also found their way into edge detection. Huaqing *et al.* demonstrated edge detection of noisy images using two cellular neural networks [Li+11]. The authors trained the first Cellular Neural Network to denoise the image and the second Cellular Neural Network to find the edges in the restored image.

A different approach was explored by a study by Patel & More in 2013, utilizing fuzzy processing [PM13]. In the study, the authors proposed a combination of fuzzy image processing and a Cellular Automata used as a second step to improve the edge map. The authors claim that their method is competitive, but the fragmented edge maps and lack of quantitative error measurements fail to support that claim.

The study by Amrogowicz *et al.* proposed Outer Totalistic Cellular Automata (OTCA) for edge detection [AZZ16]. The authors did a thorough literature review and compared various CA types from previous literature with different rulesets to their own proposed rulesets. Amrogowicz *et al.* conclude that TCA and OTCA performed the best among all CA regardless of the level

or type of noise. In addition, they note that OTCA perform better with salt-and-pepper noise under five percent, and TCA take the lead above five percent. However, "for all of proposed OTCA rules, resulting edges are thin and continuos" as concluded by the authors.

Several studies have explored the space of evolutionary edge-detecting CA and have proposed different rule-finding techniques. Rosin utilized the sequential floating forward search method for efficient rule selection [Ros06] while Uguz *et al.* used the particle swarm optimization (PSO) method for rule finding on grayscale images [USS15]. The latter study was extended in 2019 by Dumitru *et al.* [Dum+19] to enable RBG color images as input, also using PSO. All three evolutionary CA studies used the Moore neighborhood and pattern-based rules. Here, subwindow aggregation would enable a bigger neighborhood without the set of possible patterns to increase exponentially.

3 Research Hypothesis

Rosin's initial argument that subwindow aggregation is attractive because it widens the neighborhood without widening the possible set of patterns is the first clear advantage of the method [Ros06]. While a Moore neighborhood has $2^8 = 256$ possible patterns without the central cell, the extended van Neumann and extended Moore neighborhood have $2^{12} = 4\,096$ and $2^{24} = 16\,777\,216$ respectively. Thus, reducing the set of possible patterns by making the neighborhood smaller is already a substantial advantage of the method for Cellular Learning Automata.

Given Rosin's 2006 proposed subwindow configuration with eight subwindows shown in Figure 1.1 Page 2 and the majority aggregation rule, I expect three additional benefits from this method. First, the proposed modification should increase the noise robustness in the CA edge detection since single noisy pixels lose their effect through the majority aggregation rule. This noise robustness should hold for any image noise type. Second, the subwindow aggregation CA should still have a competitive run time compared to other algorithms. Finally, given future efforts, the previously mentioned hypotheses lead to believe that this modification could form the basis for a subwindow noise-robust CA to replace a whole combination of a standard edge detector and an image denoising method.

If these claims hold true in the tests, subwindow aggregation could be an easy method to improve the noise robustness of edge detecting CA.

4 Test Procedure

The tests aim to show that subwindow neighborhood aggregation improves their noise robustness while preserving their efficiency. This thesis does not propose a complete new CA to prove the viability of subwindows. However, it modified two existing edge detecting CA from previous literature to do so. I chose the work of Amrogowicz *et al.* [AZZ16] as a basis for my subwindow modifications due to several reasons. First of all, Amrogowicz *et al.*'s work has made a significant contribution to the field by comparing the performance of over 30 different edge detectors, including four different CA types with 18 different rule sets. This very descriptively comparison resulted in an excellent reproducibility of the tests. Furthermore, Amrogowicz *et al.* evaluated the influence of different types and different levels of noise on the results while using Pratt's Figure of Merit (PFoM) and the run time as metrics. Hence, this paper, as a basis for the subwindow modifications, enables me:

- to implement and apply the subwindow modifications to several already proposed CA with ease
- to improve the noise robustness and compare the results with their performance under different image noise configurations
- to compare the modified, subwindow version also in terms of execution time

Besides comparing my modified versions to Amrogowicz *et al.* base CA, I want to prove that my algorithms are competitive in speed and quality to standard edge detection algorithms like Canny, Sobel, Prewitt, and Roberts. The comparison with standard algorithms will happen with and without an extra pre-denoising step for the mentioned algorithms. This comparison will prove my hypothesis that the proposed subwindow CA come close to the combined performance of edge detector and denoising method.

In the next sections, I am going to outline the test setup and the planned test execution. First, I will describe the algorithm implementation and image dataset setup. Then, I will elaborate on how I planned to test the algorithms before concluding with the used performance metrics.

4.1 Setup

4.1.1 The Dataset

To compare my modified version to the base version of Amrogowicz *et al.* [AZZ16], this thesis will use the same testing conditions they utilized in their paper. Consequently, I will perform all tests on binary images. I have chosen the binary shape image dataset MPEG7 CE Shape-1 Part B [TGJ07] for several reasons. First of all, the simplicity of the binary shapes enables me to generate the ground truth edge maps with the simple transition rule of [WS03]. The transition rule defines that only white pixels are changed to black if they have precisely eight white pixels in its Moore neighborhood. All white pixels surrounded by white pixels are rendered black. Second, the simple shapes and edge maps let me inspect the performance in detail rather than the big picture. Detecting the edges roughly in the right place is easier than getting every pixel in the edge correct. With this dataset, I can inspect the latter for the different edge detectors. Furthermore, the small and simple edge maps enable simple visual inspection. With 60 shapes and 20 instances of each shape, the dataset also presents a good variety and size.

I randomly divided the binary shape image dataset into two groups of 600 images for the test execution. The first group of images is the training dataset on which I fine-tuned the proposed method and experimented. The second dataset is the test dataset used for the final test results, shown later in this thesis. This strategy prevents overfitting and ensures good results on unseen data. While overfitting originally comes from machine learning, it is still relevant in many experiments and test setups like this one. Without this split into training and testing datasets, the tweaking of the algorithm using the training dataset might lead to almost perfect results. However, the algorithm could fail to perform on other datasets if the training set was not heterogeneous enough. This dataset split attempts to mitigate overfitting.

4.1.2 Algorithm Implementation

For the following algorithm implementation, I used GNU Octave version 6.1.0 and uploaded the complete source code to GitHub[1]. The developed program can perform edge detection using a count-based Uniform CA. The algorithm can simulate various proposed algorithms from Amrogowicz *et al.*, meaning it can perform edge detection using TCA and OTCA. Furthermore, the subwindow neighborhood aggregation can be enabled or disabled by choice to generate metrics comparing the two versions.

[1] https://github.com/8BitJonny/sub-window-cellular-automata

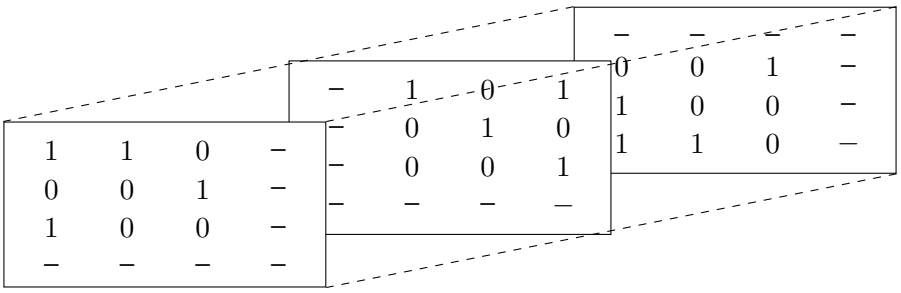

Fig. 4.1: 3D sparse matrix of partially overlapping neighborhoods

Octave, as well as Matlab, are optimized for working with scalars and vectors. Hence, the goal is to write code that does not involve any loop structures but instead uses vectors or matrices to execute statements on multiple values. If the reader is new to the topic of Code Vectorization, they can refer to the specific chapters in the Matlab[2] or Octave[3] documentation. This thesis will also shortly elaborate on that topic while explaining my algorithm.

Figure 4.2 shows the high-level pseudo code for my implementation and the four significant steps required for the edge detection process. First of all, we get a binary image with $m * n$ pixel as input. The algorithm loads the rule lookup table in the initialization phase, enabling us to look up the cell's next state based on its current neighborhood count. It also instantiates an empty four-dimensional $m \times n \times m \times n$ sized sparse matrix to use for subwindow computations. This matrix must be sparse, meaning it only stores non-zero elements in memory, since a full matrix for a 500* 500-pixel image would result in 62 500 000 000 numbers in memory, and we would only need 0,003% of that. The four dimensions are required because we compute $m * n$ partially overlapping neighborhoods for all subwindows in parallel. Thus, to use the matrix, you use the x and y coordinates of a pixel as the first and second dimension index. The result is a two-dimensional $m \times n$ sparse matrix containing the neighborhood of that pixel at the exact position the neighborhood would have in the actual image. Hence, you can see how the four-dimensional sparse matrix is just a two-dimensional array of two-dimensional arrays, each containing one neighborhood that would otherwise overlap. Fig. 4.1 shows this approach broken from four to three dimensions by not showing a two-dimensional but a one-dimensional array of overlapping neighborhoods.

In step two, the algorithm computes all $m * n$ neighborhoods using the earlier majority aggregation technique. First, through the $sw\_indexes$ indexing matrix, it reorganizes the image pixels into the individual subwindows. The majority aggregation then happens by dividing the number of alive cells within each subwindow by the total cells of the subwindow and rounding the result. The results are partially overlapping aggregated neighborhoods saved

[2]https://de.mathworks.com/help/matlab/matlab_prog/vectorization.html
[3]https://octave.org/doc/v4.2.2/Basic-Vectorization.html

Input: $img_{m \times n}$ Binary; m = number of rows; n = number of columns
Output: $edge\_img_{m \times n}$ where m = number of rows; n = number of columns
begin
    Step 1: initialization
    $LUT \leftarrow$ Rule Look Up Table
    $sw\_state_{m \times n \times m \times n} \leftarrow$ Empty Sparse Matrix$_{m \times n \times m \times n}$

    Step 2: compute Subwindow States
    $sw\_state_{m \times n \times m \times n} \leftarrow round($
        $sum($
            $img[sw\_indexes]$
        $)/total\_cells\_in\_sw$
)
    Step 3: count Alive Neighbors for each Subwindow
    $alive\_neighbor\_count_{m \times n} \leftarrow sum(subwindow\_state_{m \times n \times m \times n})$
    Step 4: translate Alive Neighbors into next State
    $edge\_img_{m \times n} \leftarrow LUT[alive\_neighbor\_count_{m \times n}]$
end

Fig. 4.2: Pseudo code subwindow edge detection CA

into the $subwindow\_state_{m \times n \times m \times n}$ sparse matrix. The maximum number of non-zero elements within the sparse matrix is defined by $m * n * number\_of\_subwindows$.

In step three, the implementation counts the non-zero elements for each previously computed neighborhood. The result is a $m * n$ sized full matrix where each index stores the number of alive neighborhood pixels for the pixel at that index.

In the final step, the algorithm looks up the next state of each pixel by the number of alive pixels and returns the resulting $m * n$ edge image as the final result.

4.1.3 Algorithm Time and Space Complexity

Given the limited time, most of the code is optimized but not to perfection. As seen in the pseudo-code in Figure 4.2, the implementation does not utilize any loop. All defined functions accept and return matrices of pixels as inputs and output. Through this Code Vectorization, Octave is able to optimize the code execution in its Fortran, C, or C++ implementation or its lower-level dependencies [@Eat+20].

However, this Code Vectorization makes the calculation of the theoretical time complexity of the proposed method a bit more challenging. As previously stated, I optimized my program not to use a single loop. Thus, the number of operations done at my program level does

not vary depending on the size of the input image. Consequently, one could argue that the theoretical time complexity of the proposed method is $\mathcal{O}(1)$ meaning constant time. However, the problem is that we do not know the implementation details of some of Octave's functions or where Octave optimizes the users' vector instructions. We get a much higher time complexity, considering the lower level of implementation and the worst-case scenario where Octave is not optimizing any vector instructions. In this case, the most expensive operation in the algorithm is to operate on the four-dimensional sparse matrix. In Matlab, we know that the computational complexity of sparse matrix operations is proportional to the number of nonzero elements [@Inc21]. However, we can implement two for loops to iterate over each pixel and a third loop to aggregate each subwindow in Octave yielding the same complexity. Here the theoretical upper bound is $\mathcal{O}(m * n * number\_of\_subwindows)$ where m is the number of rows and n the number of columns inside the image. We can remove the constant factor $number\_of\_subwindows$ for $m * n \to \infty$ and simplify the theoretical upper bound to $\mathcal{O}(m * n)$ or $\mathcal{O}(p)$, where p equals the number of pixels. Hence, the algorithm would follow linear time complexity.

The space complexity of the proposed method is interesting since my current implementation saves time by using more memory. As already shown in the pseudo-code Figure 4.2, the algorithm computes all neighborhoods from all subwindows simultaneously. Since the resulting neighborhoods are partially overlapping, my solution involves saving them in a four-dimensional sparse matrix. This sparse matrix has the dimensions $m \times n \times m \times n$ where m is the number of pixel rows, and n is the number of columns in the input image. However, the maximum count of non-negative numbers stored within this matrix is $m * n * neighborhood\_size$. Apart from this matrix, the algorithm implementation stores the input image (size $m \times n$), the result (size $m \times n$), and two matrices (size $m \times n \times neighborhood\_size$) which store values to index the input image and the sparse matrix. I can calculate the resulting Space Complexity using the following formula:

$$SpaceComplexity = m * n * (2 * neighborhood\_size + 2)$$

We can conclude that the current implementation of the proposed subwindow aggregation follows Linear Space Complexity.

4.2 Result Evaluation

4.2.1 Testing Methods

I will use several methods to ensure proper comparison and evaluation of the proposed method. The result evaluation will include:

- (a) tests executed with and without noise
- (b) different types and different intensities of simulated noise
- (c) comparison of the base version of Amrogowicz *et al.* CA with the subwindow versions
- (d) comparison with standard edge detectors Canny, Sobel, Prewitt, Roberts
- (e) comparison to the mentioned standard edge detectors with and without different standard denoising filters as a pre-denoising step only for these algorithms

Following these five methods, I will show and ensure:

- through (a) and (b), the viability of the proposed method with and without noise
- through (c), the superiority of the proposed method over Amrogowicz *et al.* base versions
- through (d), the superiority of the proposed method over the standard edge detectors
- through (e), the potential of the proposed approach to replace combinations of standard edge detectors and separate denoising functions in the future

To prove these claims, I will interpret the edge detection results visually and with the following objective error measurements.

4.2.2 Error Measurements

It is an ongoing problem to find the best method to compare and evaluate the performance of different edge detectors. Researchers have proposed many different approaches over the years, but they have not reached a common agreement on the best method so far. While Arbelaez *et al.* claim a wide acceptance to use the BSDS500 Precision-Recall-Based evaluation method [Arb+11], Lopez-Molina *et al.* conclude their research paper two years later by saying that there currently is no convincing method for performance comparison [LDB13]. Furthermore, Lopez-Molina *et al.* speculate that the current solution may be to combine different error measurements to give a well-rounded performance review. This conclusion is well reasoned. Consequently, I chose the combination of Pratt's Figure of Merit, Baddeley's Delta Metric,

the Root-Mean-Squared error, the Peak signal-to-noise ratio, and the mean run time as my performance metrics. The goal to stay close to the original paper by Amrogowicz *et al.* [AZZ16] supports this decision since they also use Pratt's Figure of Merit for evaluation [AZZ16].

Pratt's Figure of Merit (PFoM)

PFoM [AP79] is defined as

$$PFoM(E_{gt}, E_c) = \frac{1}{max(|E_{gt}|, |E_c|)} \sum_{p \in E_c} \frac{1}{1 + K * d^2(p, E_{gt})}$$

where $K \in \mathbb{R}^+$ is a constant, usually chosen as 1/9 [Bad92b], E_{gt} and E_c are the ground truth and actual edge images, and d denotes the distance, in this instance between the detected pixel and the actual edge. Pratt's Figure of Merit results in a floating-point number between zero and one where one represents a perfect match and zero represents a complete miss-match of the two edge maps.

Baddeley's Delta Metric (BDM)

BDM was proposed in 1992 by Adrian Baddeley as a response to Pratt's Figure of Merit and other statistical missclassification rates [Bad92a]. It is defined as follows for $1<k<\infty$

$$\triangle_w^k(E_{gt}, E_c) = [\frac{1}{|P|} \sum_{p \in P} |w(d(p, E_{gt})) - w(d(p, E_c))|^k]^{1/k}$$

where P is the set of all pixels, $|P|$ is the number of pixels and w is a concave function. The function w is used to inflect the distance and can therefore alter the penalization of wrong pixels. The tests in this thesis use the parameters $k = 1$ and $w(x) = x$. The resulting number, calculated by BDM, is an indicator of the amount miss-match. The lower the BDM result, the better the match between the two images.

Root-Mean-Squared Error (RMSE)

RMSE is defined by taking the square root of the average of squared errors.

$$RMSE = \sqrt{Mean((A - B)^2)}$$

Based on the equation, we can follow that lower RMSE values represent a better match between two edge maps.

The RMSE is simple to compute and easy to interpret. Lower RMSE values mean lower errors between the two images and thus a more accurate edge map. However, it is also too simple to accurately quantify an edge map and falls short, for instance, when the detected edges are off by a bit.

Peak Signal-to-Noise Ratio (PSNR)

PSNR is expressed in dB and puts the noise, represented by MSE, in relation to the peak signal, which is the maximum pixel value inside the image.

$$PSNR = 10 * \log_{10}(\frac{MAX^2}{MSE})$$

The same shortcomings of the RMSE also apply to the PSNR since both calculations rely on the measured Mean Squared Error. However, in contrast to RMSE, higher PSNR values represent better edge maps and lower PSNR values worse edge maps.

Speed

Since the hypothesis states that the subwindow modification will not lead to significantly higher run time complexity, I will also measure the mean run time. If the results show that the proposed method is within the reach of the other methods, then the claim is proven right. I will not further define the meaning of "within reach" since the run time optimization of the proposed method is not yet optimal. The algorithm currently has a solid implementation through Octave's Code Vectorization techniques, but there is still room to improve. I could not finish the run-time optimization due to time constraints.

5 Test Results

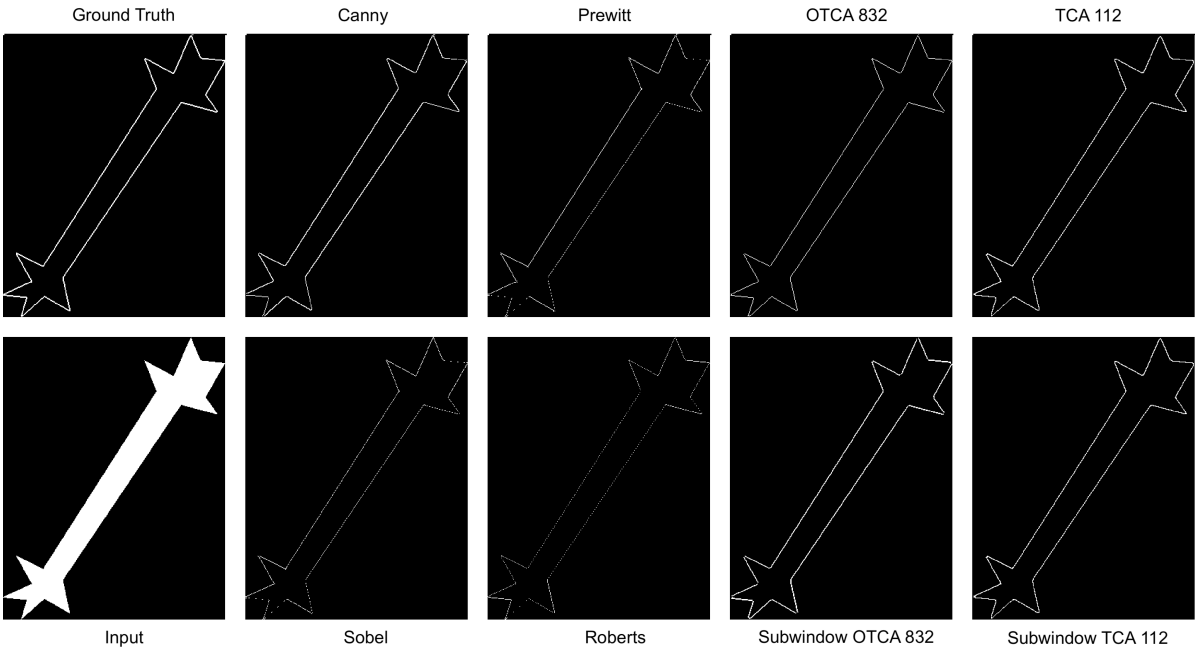

Fig. 5.1: Noise-free edge detection result images

Figure 5.1 shows the first scenario where I ran all edge detectors on noise-free images. The bottom left image shows the image passed to the edge detectors, and the top left is the ground truth version, generated by the simple transition rule from Wonghtanavasu *et al.* [WS03]. OTCA 832 and TCA 112 are the edge detectors from Amrogowicz *et al.* [AZZ16] and below them, the figure shows their modified version with the proposed subwindow neighborhood aggregation. All metrics are the mean values of all measured results of the 600 test images, while the best values for each metric are in bold. The tests ran on a 13 inch MacBook Pro 2017 with 8 GB of RAM and a 3,1 GHz Dual-Core Intel Core i5.

From the combination of the measured performance metrics in Table 5.1 and the edge maps in Figure 5.1 we can note a couple of points. The run time of the proposed modified version is 0.07 - 0.09ms longer but is still very much within the reach of all compared algorithms. The BDM, PFoM, PSNR, and RMSE values do not show a clear favorite between the base edge detectors from Amrogowicz *et al.* and the subwindow modified versions. BDM and PSNR are slightly in favor of the CA-based edge detectors over the standard edge detectors. The edge maps also support this tendency where the edge maps from Roberts, Sobel, and Prewitt are a bit porous.

Tab. 5.1: Noise-free edge detection

	BDM	PFoM	PSNR (dB)	RMSE	Time (ms)
Canny	0.63	0.94	19.65	0.11	0.02
Sobel	1.42	0.77	18.21	0.13	**0.01**
Prewitt	1.42	0.77	18.34	0.13	**0.01**
Roberts	0.73	0.90	18.81	0.12	**0.01**
OTCA832	0.61	0.93	19.80	0.11	0.08
Subwindow OTCA832	0.81	0.94	18.73	0.12	0.15
TCA112	0.60	0.95	20.09	**0.10**	0.09
Subwindow TCA112	**0.59**	**0.96**	**20.92**	**0.10**	0.14

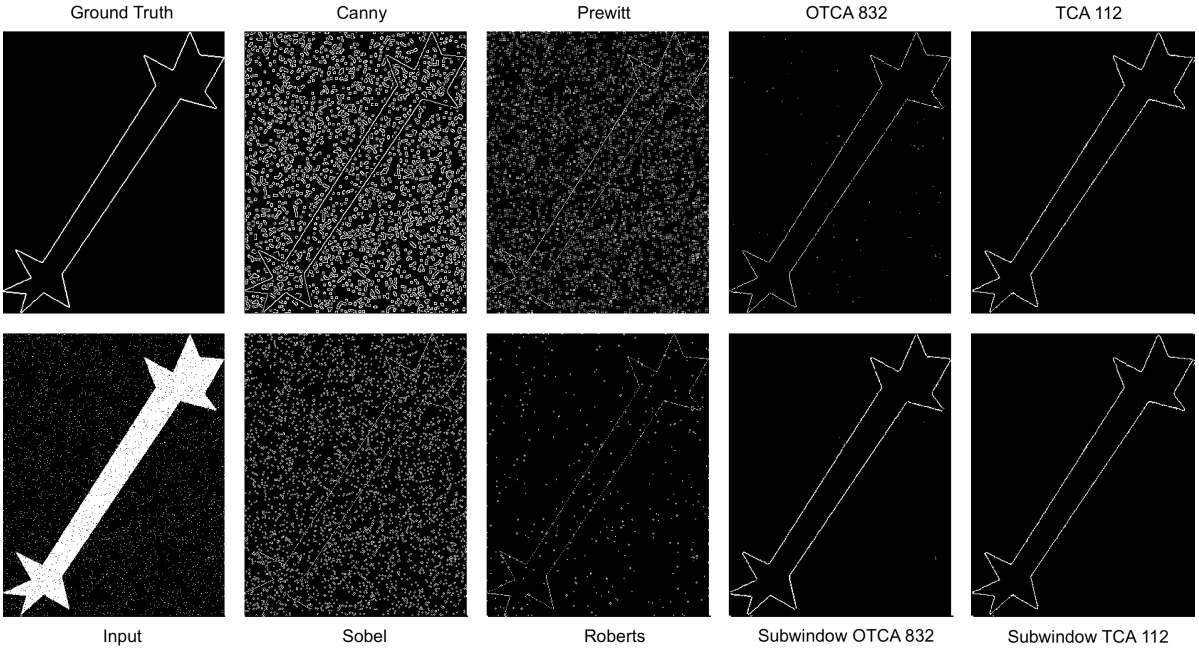

Fig. 5.2: Five percent Salt-and-pepper noise + Gaussian blur

Since this thesis is about the noise robustness of the proposed methods, Figure 5.2 shows the test results with five percent Salt-and-pepper noise. For a fair comparison, I added a pre-noising step for the four standard algorithms. The following scenario uses the Gaussian blur for denoising, one of the most popular and widely known image denoising methods. One can see that the standard edge detectors are still heavily impacted by the noise. However, the edge maps and metrics in Table 5.2 also reveal that both subwindow versions perform better and contain less noise in the edge map than the base CA edge detectors.

I reran the test, this time with a more powerful denoising method for the standard algorithms. Figure 5.3 shows the achieved results with Perona & Malik's Anisotropic diffusion method [PM90] for denoising. The CA-based edge detectors' results are not in Figure 5.3 because

Tab. 5.2: Five percent Salt-and-pepper noise + Gaussian blur

	BDM	PFoM	PSNR (dB)	RMSE	Time (ms)
Canny	45.13	0.17	7.82	0.41	0.04
Sobel	44.11	0.29	10.68	0.30	**0.02**
Prewitt	43.99	0.32	11.05	0.29	**0.02**
Roberts	38.80	0.88	16.72	0.15	**0.02**
OTCA832	27.64	0.94	19.49	0.11	0.09
Subwindow OTCA832	9.56	0.93	18.60	0.12	0.16
TCA112	12.11	0.95	20.12	0.10	0.10
Subwindow TCA112	**1.43**	**0.96**	**20.96**	**0.09**	0.15

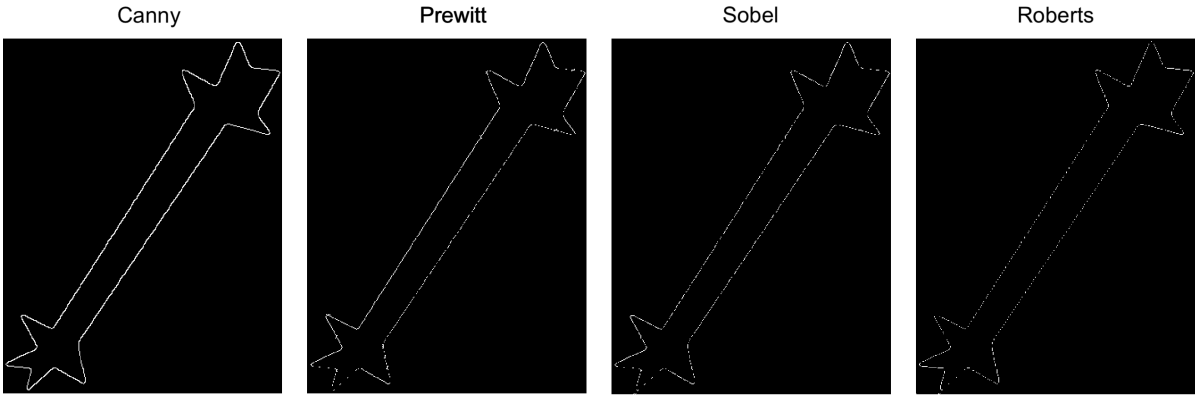

Fig. 5.3: Five percent Salt-and-pepper noise + Perona & Malik

they do not differ from what Figure 5.2 shows. With the performance metrics shown in Table 5.3 and the edge maps, it is clear that the standard edge detectors have no problem once the Perona & Malik method has almost entirely restored the image. However, all four CA methods also produce good results. While the run time is now almost equal through the additional computations from the P&M denoising, the Subwindow TCA 112 has the best PFoM, PSNR, and RMSE values and only is slightly behind Canny in the BDM results. The high BDM values for the other three edge detection CA originate because Baddeley's Delta Metric is the metric that punishes false positive edges the most. Hence, the small edges caused by noise visible in Figure 5.2 for OTCA832, Subwindow OTCA832, and TCA 112 significantly impact the BDM values.

This trend continues if we use higher noise values like 15% Salt-and-pepper noise. In Figure 5.4, you can see that the combination of standard edge detector plus Perona & Malik performs well under 15% noise. The CA-generated edge maps do not look as clean and have noise. However, we can again witness in Table 5.4 that the subwindow modified CA outperform Amrogowicz *et al.* versions in every metric. Furthermore, the Subwindow TCA112 ranked the best among all algorithms for the PFoM, PSNR, and RMSE metrics. Only the BDM shows the penalization of the wrong detected edges caused by the noise the subwindows could not filter out. The

Tab. 5.3: Five percent Salt-and-pepper noise + Perona & Malik

	BDM	PFoM	PSNR (dB)	RMSE	Time (ms)
Canny	**1.03**	0.91	19.22	0.12	0.14
Sobel	1.57	0.76	18.10	0.13	0.12
Prewitt	1.43	0.78	18.24	0.13	0.12
Roberts	1.29	0.84	18.34	0.13	0.12
OTCA832	27.40	0.94	19.50	0.11	**0.09**
Subwindow OTCA832	8.12	0.93	18.61	0.12	0.16
TCA112	12.97	0.95	20.11	0.10	0.10
Subwindow TCA112	1.10	**0.96**	**20.97**	**0.09**	0.15

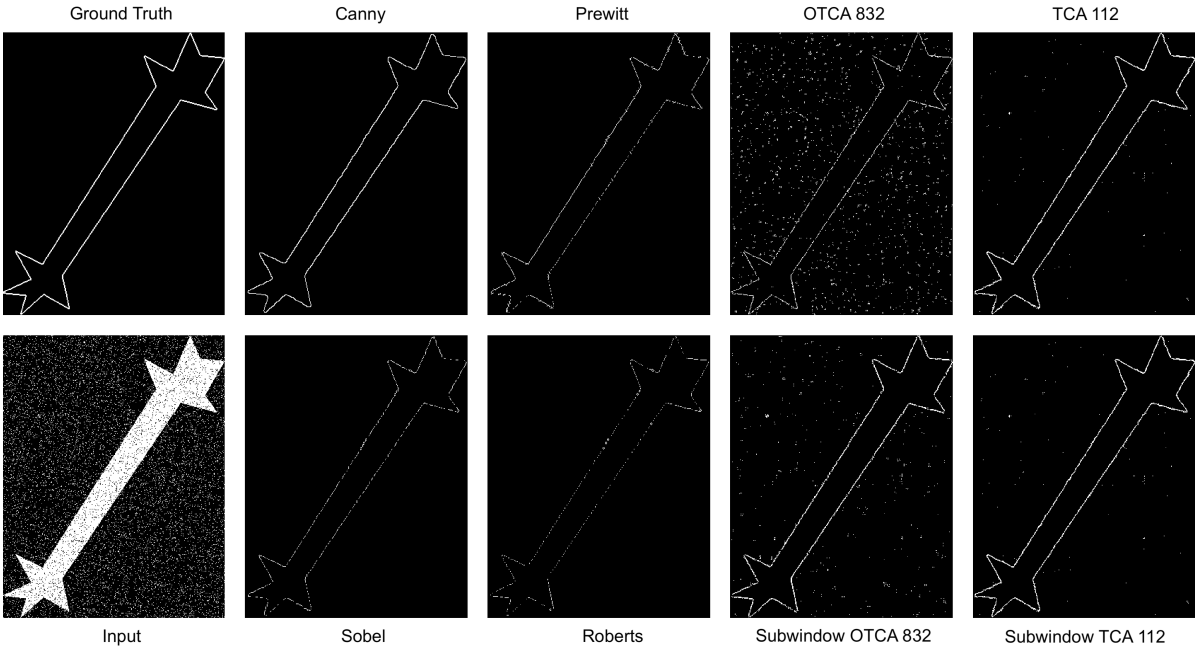

Fig. 5.4: 15% Salt-and-pepper noise + Perona & Malik

same tendencies show if we consider the test results from the zero-mean Gaussian noise with 0.01 variance. Other than Salt-and-pepper noise, Gaussian noise is additive and does not replace whole pixel values. However, after applying Gaussian noise, the effect is confined by re thresholding the grayscale Gaussian noise image back to binary. Nevertheless, we can see a first indication of the effect on the resulting performance metrics in table 5.5. Again, we see the subwindow TCA 112 performing best in the PFoM, PSNR, and RMSE metrics while Canny leads in the BDM metric. All subwindow modified CA outperform or are equal to their Amrogowicz *et al.* counterparts. A surprising finding is that the subwindow CA seem to have a higher noise resistance within the shape boundaries than they do outside of them. This would mean the tested subwindow CAs have fewer problems filtering the pepper salt than the salt noise.

Tab. 5.4: 15% Salt-and-pepper noise + Perona & Malik

	BDM	PFoM	PSNR (dB)	RMSE	Time (ms)
Canny	**1.12**	0.91	19.05	0.12	0.13
Sobel	1.27	0.81	18.27	0.13	0.12
Prewitt	1.14	0.83	18.22	0.13	0.12
Roberts	1.37	0.81	18.04	0.13	0.12
Plain OTCA832	40.49	0.79	16.38	0.15	**0.09**
Subwindow OTCA832	32.77	0.89	17.37	0.14	0.16
Plain TCA112	38.44	0.84	17.30	0.14	0.10
Subwindow TCA112	27.03	**0.96**	**20.40**	**0.10**	0.16

Tab. 5.5: Zero mean, 0.01 variance Gaussian noise + Perona & Malik

	BDM	PFoM	PSNR (dB)	RMSE	Time (ms)
Canny	**1.06**	0.91	19.11	0.12	0.15
Sobel	1.33	0.80	18.21	0.13	0.14
Prewitt	1.18	0.81	18.23	0.13	0.14
Roberts	1.33	0.82	18.14	0.13	0.14
Plain OTCA832	38.00	0.89	17.73	0.13	**0.10**
Subwindow OTCA832	29.25	0.93	18.06	0.13	0.19
Plain TCA112	34.06	0.95	18.71	0.12	0.12
Subwindow TCA112	18.71	**0.96**	**20.80**	**0.10**	0.18

Figure 5.5 highlights this using the Subwindow OTCA832 with 20, 30, 40, and 50% Salt-and-pepper noise. Only at 50% noise do we start to see the first noisy pixels within the shape. Wishful thinking might lead us to believe that a subwindow configuration or ruleset can copy this inner noise robustness to the outer areas, but we must exercise caution. Considering the limited data on inner- & outer- noise robustness, further research is needed to evaluate the potential behind these first data points.

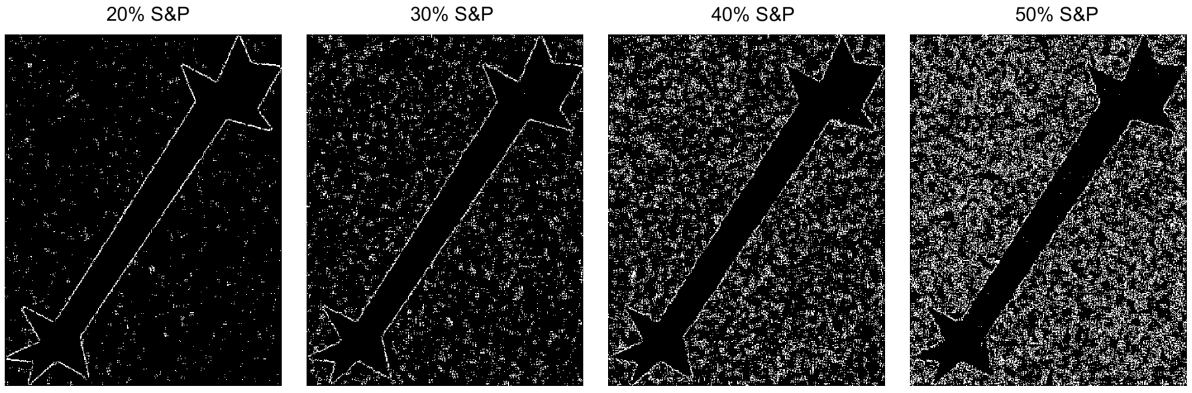

Fig. 5.5: Inner- & Outer- Noise robustness

6 Conclusion and Future Work

This thesis proposed and evaluated subwindow neighborhood aggregation for edge detecting Cellular Automata. Initially mentioned by Rosin for other image processing tasks, the idea showed promising results. This thesis modified two CA edge detectors from previous literature to utilize the proposed method. The results showed an improvement in noise robustness over the based upon CA edge detectors and a competitive run time without being fully optimized. Through the noise robustness and efficiency, subwindow neighborhood CA demonstrated the potential to replace commonly used pairs of denoising methods and edge detectors in future work efforts. Despite Perona & Malik plus Canny remaining the favorite in images with high noise intensities, the proposed method already outperforms Gaussian Blur plus standard edge detectors.

The study was limited in several ways. First, this thesis only tested the proposed method on a binary shape dataset with 60 different shapes and 20 instances per shape, but no diversification beyond that. Future work efforts might therefore include modifications to enable subwindow neighborhood aggregation on grayscale images. This would also allow the proper use of the BSDS500 image dataset [Arb+11] with their proposed performance metrics to compare the results with countless other state-of-the-art algorithms.

Second, this thesis showed the benefits of the proposed method on count-based CA. Exploring the effect on other CA types like pattern-based CA could be particularly interesting since Rosin's proposed subwindow configuration [Ros06] retains the image features in the respective direction inside of the neighborhood.

Third, this thesis used Rosin's initially proposed subwindow configuration for edge detection. Exploring and evaluating different sizes, shapes, and arrangements of subwindows might lead to better results.

Finally, the current interest in Neural Networks and Cellular Neural Networks might inspire the application of subwindow neighborhood aggregation to the latter. Combining subwindow aggregation and Cellular Neural Networks has the potential to utilize the best of both methods. The Cellular Neural Network can optimize itself for the given task and conditions while the subwindows enhance the neighborhood information through aggregation and rearrangement.

Bibliography

[AP79] Ikram E Abdou and William K Pratt. “Quantitative design and evaluation of enhancement/thresholding edge detectors”. In: *Proceedings of the IEEE* 67.5 (1979), pp. 753–763 (cit. on p. 16).

[AZZ16] Sebastian Amrogowicz, Yitian Zhao, and Yifan Zhao. “An edge detection method using outer Totalistic Cellular Automata”. In: *Neurocomputing* 214 (2016), pp. 643–653 (cit. on pp. 4, 7, 10, 11, 16, 18).

[Arb+11] Pablo Arbeláez, Michael Maire, Charless Fowlkes, and Jitendra Malik. “Contour detection and hierarchical image segmentation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.5 (2011), pp. 898–916 (cit. on pp. 15, 23).

[Bad92a] Adrian J Baddeley. “An error metric for binary images”. In: *Robust Computer Vision: Quality of Vision Algorithms* (1992). Ed. by W. Forstner and S. Ruwiedel, pp. 59–78 (cit. on p. 16).

[Bad92b] Adrian J Baddeley. “Errors in binary images and an Lp version of the Hausdorff metric”. In: *Nieuw Archief voor Wiskunde* 10.4 (1992), pp. 157–183 (cit. on p. 16).

[Can86] John F. Canny. “A computational approach to edge detection”. In: *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986), pp. 679–698 (cit. on p. 6).

[Dum+19] Delia Dumitru, Anca Andreica, Laura Diosan, and Zoltán Bálint. “Particle swarm optimization of cellular automata rules for edge detection”. In: *2019 21st International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE. 2019, pp. 320–325 (cit. on p. 8).

[Li+11] Huaqing Li, Xiaofeng Liao, Chuandong Li, Hongyu Huang, and Chaojie Li. “Edge detection of noisy images based on cellular neural networks”. In: *Communications in Nonlinear Science and Numerical Simulation* 16.9 (2011), pp. 3746–3759 (cit. on p. 7).

[LDB13] Carlos Lopez-Molina, Bernard De Baets, and Humberto Bustince. “Quantitative error measures for edge detection”. In: *Pattern Recognition* 46.4 (2013), pp. 1125–1139 (cit. on p. 15).

[Mof+15] Mohammad Hasanzadeh Mofrad, Sana Sadeghi, Alireza Rezvanian, and Mohammad Reza Meybodi. “Cellular edge detection: Combining cellular automata and cellular learning automata”. In: *AEU-International Journal of Electronics and Communications* 69.9 (2015), pp. 1282–1290 (cit. on pp. 4, 7).

[Neu66] John Von Neumann. *Theory of Self-Reproducing Automata*. Ed. by Arthur W. Burks. University of Illinois Press, Urbana, 1966 (cit. on pp. 2, 4).

[PM13] Dhiraj Kumar Patel and Sagar A More. “Edge detection technique by fuzzy logic and Cellular Learning Automata using fuzzy image processing”. In: *2013 International Conference on Computer Communication and Informatics*. IEEE. 2013, pp. 1–6 (cit. on p. 7).

[PM90] Pietro Perona and Jitendra Malik. “Scale-space and edge detection using anisotropic diffusion”. In: *IEEE Transactions on pattern analysis and machine intelligence* 12.7 (1990), pp. 629–639 (cit. on p. 19).

[Pre70] Judith MS Prewitt. “Object enhancement and extraction”. In: *Picture processing and Psychopictorics* 10.1 (1970), pp. 15–19 (cit. on p. 6).

[Pri+17] Blanca Priego, Abraham Prieto, Richard J Duro, and Jocelyn Chanussot. “Spatio-temporal cellular automata-based filtering for image sequence denoising”. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2017, pp. 2362–2369 (cit. on p. 4).

[Rob63] Lawrence G Roberts. “Machine perception of three-dimensional solids”. PhD thesis. Massachusetts Institute of Technology, 1963. 82 pp. (cit. on p. 5).

[Ros06] Paul L Rosin. “Training cellular automata for image processing”. In: *IEEE Transactions on Image Processing* 15.7 (2006), pp. 2076–2087 (cit. on pp. 1, 4, 8, 9, 23).

[Ros10] Paul L Rosin. “Image processing using 3-state cellular automata”. In: *Computer vision and image understanding* 114.7 (2010), pp. 790–802 (cit. on p. 4).

[SF68] Irwin Sobel and Gary Feldman. “A 3x3 isotropic gradient operator for image processing”. In: *a talk at the Stanford Artificial Project in* (1968), pp. 271–272 (cit. on p. 5).

[TGJ07] Ninad Thakoor, Jean Gao, and Sungyong Jung. “Hidden Markov model-based weighted likelihood discriminant for 2-D shape classification”. In: *IEEE Transactions on Image Processing* 16.11 (2007), pp. 2707–2719 (cit. on p. 11).

[USS13] Selman Uguz, Ugur Sahin, and Ferat Sahin. “Uniform cellular automata linear rules for edge detection”. In: *2013 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE. 2013, pp. 2945–2950 (cit. on p. 7).

[USS15] Selman Uguz, Ugur Sahin, and Ferat Sahin. “Edge detection with fuzzy cellular automata transition function optimized by PSO”. In: *Computers & Electrical Engineering* 43 (2015), pp. 180–192 (cit. on p. 8).

[WS03] Sartra Wongthanavasu and R Sadananda. “A CA-based edge operator and its performance evaluation”. In: *Journal of Visual Communication and Image Representation* 14.2 (2003), pp. 83–96 (cit. on pp. 7, 11, 18).

[ZCZ20] Fan Zhang, Xiaopan Chen, and Xinhong Zhang. “Parallel thinning and skeletonization algorithm based on cellular automaton”. In: *Multimedia Tools and Applications* 79.43 (2020), pp. 33215–33232 (cit. on pp. 3, 4).

Webpages

[@Eat+20] John W. Eaton, David Bateman, Søren Hauberg, and Rik Wehbring. *GNU Octave version 6.1.0 manual: a high-level interactive language for numerical computations*. 2020. URL: `https://www.gnu.org/software/octave/doc/v6.1.0/` (visited on Aug. 1, 2021) (cit. on p. 13).

[@Inc21] The Mathworks Inc. *Sparse Matrix Operations*. 2021. URL: https://www.gnu.org/software/octave/doc/v6.1.0/ (visited on Aug. 7, 2021) (cit. on p. 14).

List of Figures

List of Tables

Colophon

This thesis was typeset with LaTeX 2_ε. It uses the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at `http://cleanthesis.der-ric.de/`.

Declaration

I hereby declare that the work titled "***Subwindow Neighborhood aggregation for edge detecting cellular automata***" was written independently, as I am the sole author. I did not use any other assistance or sources, other than those mentioned. All references taken literally or correspondingly from published and unpublished writings are marked as such. I am aware that any content used from the internet must have been acknowledged and added as an electronically stored resource.

This work was prepared to be presented and evaluated as a Bachelor's Thesis. It was not submitted either in its entirety or parts thereof to any other examination authority.

Date: Berlin, 12.08.2021

Signature: