Analysis of Algorithm for Image Detetction

by

Pirunrat Kaewphanoaw <st124003@ait.asia>

Rakshya Lama Moktan <st124088@ait.asia>

A Project Progress Proposal Submitted in Partial Fulfillment of the Requirements for the
Degree of Master of Engineering in
Data Science and Artificial Intelligence

Examination Committee: Prof. Chantri Polprasert (Chairperson)

Nationality: Thai, Nepali

Asian Institute of Technology
School of Engineering and Technology
Thailand

March 2024

AUTHOR'S DECLARATION

We, Pirunrat and Rakshya, declare that the research work carried out for this project was in accordance with the regulations of the Asian Institute of Technology. The work presented in it are our own and has been generated by us as the result of our own original research, and if external sources were used, such sources have been cited. It is original and has not been submitted to any other institution to obtain another degree or qualification. This is a true copy of the thesis including final revisions.

Date: March 13, 2024

Signature: 

ACKNOWLEDGEMENTS

We would like to extend our heartfelt gratitude to Dr. Chantri for providing us with the invaluable opportunity to work on this project. Your guidance, support, and encouragement throughout the duration of this endeavor have been instrumental in shaping our understanding and skills in the field.

We are also deeply thankful to all the co-authors of the papers associated with this project for their continuous guidance and insightful feedback. Your expertise and dedication have been crucial in navigating through the challenges and complexities of our research work.

Furthermore, we would like to express our sincere appreciation to TA Sandhya for her unwavering support and guidance. Her assistance has been invaluable in ensuring the smooth progress of our project and addressing any issues that arose along the way.

This project has been a truly enriching experience, and we are grateful to each and every individual who has contributed to its success.

CONTENTS

CHAPTER 1
INTRODUCTION

1.1 Background of the Study

Image segmentation is the process of partitioning an image into multiple meaningful regions or objects. It plays a crucial role in many computer vision tasks, including object detection, image recognition, and image annotation. Image annotation involves labeling or marking specific regions or objects within an image for various purposes, such as training machine learning models or understanding image content. Image segmentation is quite popular mostly in the extraction of rural residential buildings from laser scanning data and astrophotographs in construction), video or X-ray image segmentation for precise diagnosis through insightful and detailed analysis, and so on. Among many popular approaches for image segmentation, some notable names are Quadtress Segmentation, Brute Force, and Mean-Shift Clustering. Brute force is a simple approach to image segmentation that is based on exhaustive search and pixel-by-pixel analysis. It involves examining each pixel in an image and making segmentation decisions based on local or global properties of the pixels, such as color, intensity, texture, or gradient information. Brute force segmentation does not rely on complex algorithms or sophisticated techniques but instead explores all possible combinations to determine regions or objects in an image. Mean shift is a popular non-parametric clustering algorithm that has been adapted for image segmentation. It is a data-driven approach that identifies regions or objects in an image based on the local density of pixels. Mean shift segmentation operates by iteratively shifting the center of a window or kernel towards the mode of the pixel distribution within that window. The mode represents the highest density of pixels, and the shifting process aims to converge to the modes, which correspond to the distinct regions or objects in the image. Mean shift segmentation has gained attention for its ability to handle various types of images and its capability to capture fine details and boundaries. A quadtree is a hierarchical data structure that recursively divides an image into smaller rectangular regions until a stopping criterion is met. Each region in the quadtree represents a square or rectangular area of the image, and the division process continues until the regions satisfy certain criteria, such as reaching a minimum size or having uniform color properties. The basic idea behind quadtree segmentation is to represent an image as a tree structure in which each node represents a region or sub-

region of the image. The tree starts with a root node representing the entire image, and at each level of the tree, the image is divided into four equal-sized quadrants or children nodes.

1.2 Statement of the Problem

The problem addressed in this study is the comparison of different image segmentation algorithms. The objective is to evaluate and analyze the performance and characteristics of various segmentation techniques to understand their strengths, weaknesses, and suitability for different types of images and applications.

1.3 Objectives

The main objectives of the study and comparison of different image segmentation algorithms are as follows:

Algorithm Selection: Identify a diverse set of image segmentation algorithms for the study. These algorithms can represent different categories, such as region-based methods, edge-based methods, clustering-based methods, or deep learning-based methods. The selection should ensure a comprehensive representation of the state-of-the-art segmentation techniques.

Evaluation Metrics: Define appropriate evaluation metrics to assess the performance of the segmentation algorithms. These metrics may include accuracy, precision, recall, F1-score, boundary accuracy, and computational efficiency. The evaluation metrics should capture different aspects of segmentation quality and computational cost.

Implementation: Implement or obtain existing implementations of the selected segmentation algorithms for experimentation. Ensure that the implementations are correctly configured and optimized for the chosen datasets and evaluation metrics.

Experimental Setup: Design a comprehensive experimental setup to evaluate the segmentation algorithms. This setup should include appropriate processing steps, parameter tuning, and any necessary post-processing techniques. Factors such as algorithm complexity, computational resources, and runtime efficiency will be considered.

Performance Analysis: Execute the experiments using the selected datasets and segmentation algorithms. Analyze and compare the performance of each algorithm based on

the defined evaluation metrics. Identify the strengths and weaknesses of each algorithm about different image characteristics and application scenarios. Visual Results: Generate visual results to illustrate the segmentation outputs of each algorithm. Visualize the segmented regions or objects overlaid on the original images, highlighting the similarities, differences, and potential limitations of the algorithms. Discussion and Conclusion: Discuss the findings and implications of the comparative study. Summarize the strengths and limitations of each algorithm, their suitability for different image types and applications, and potential areas for further improvement. Provide recommendations for selecting the most appropriate segmentation algorithm based on specific requirements and constraints.

By achieving these objectives, the proposed approach aims to provide an effective solution for image extraction from provided input. This solution can contribute to harnessing image segmentation and provide detailed analysis.

CHAPTER 2
LITERATURE REVIEW

All the research and publications we researched:

2.1 Simplified Quadtree Image Segmentation for Image Annotation

This paper presents a novel Quadtree image segmentation technique for image annotation tasks. The method efficiently divides images into homogeneous segments by merging adjacent regions based on border and color information. It offers flexibility in controlling the level of segmentation detail, making it suitable for time-restricted scenarios. Experimental results demonstrate the effectiveness of the proposed method, particularly for images with large objects and simple textures. However, parameter tuning and further enhancement of feature selection are identified as areas for future research. Additionally, the paper highlights the potential for multilevel annotation leveraging the hierarchical nature of Quadtree and suggests the possibility of parallel implementation to improve speed. Overall, the paper contributes a valuable approach to image segmentation, particularly beneficial for image annotation purposes, and identifies avenues for future research and improvement. Márquez, Escalante, and Sucar (2011).

2.2 Color Image Coding Based on Block Truncation Coding Using Quadtree Segmentation

This paper presents a novel color image compression technique based on block truncation coding, offering adjustable bit rates to suit different multimedia applications. The method utilizes quadtree segmentation to partition the color image into variable-sized blocks based on their activities. Encoding rules vary depending on block sizes, with smooth and complex blocks treated differently using block mean vectors and block truncation coding with bitmap omission, respectively. Experimental results demonstrate that the proposed technique achieves good image quality in reconstructed color images at low bit rates. The study highlights the adaptability of the bit rate selection according to application requirements and suggests that optimal image quality can be achieved with appropriately chosen threshold values, albeit at the cost of higher bit rates. Overall, the paper contributes an adaptive compression method tailored for color images, balancing image quality and bit rate efficiency. Hu, Liu, and Chang (2018).

2.3 Image segmentation based on Blob analysis and quad-tree algorithm

This paper explores image segmentation methods, focusing on implementing and comparing two improved existing techniques: Split and Merge and Blob Coloring algorithms, in both 2D and 3D contexts. To assess their performance, the authors establish an evaluation function (FMI) to quantify segmentation quality. The split and merge algorithm utilizes quad-tree segmentation based on homogeneity, while the blob algorithm merges pixels with similar homogeneity, considering L shapes. Experimental results indicate that the split and merge algorithm generally outperforms the blob algorithm, though the latter occasionally achieves high FMI scores. Real-image application scenarios include interior design and animal imagery, with potential applications in biomedical and public management fields. However, challenges arise when using region-growing techniques for boundary detection, especially when objects have distinct internal differences not recognized by the algorithm. Further analysis reveals that the blob algorithm struggles with images containing blurry regions near boundaries or intricate details like fur and skin. Its iterative nature makes it susceptible to errors propagating from previous points. Meanwhile, the split and merge algorithm faces limitations regarding image size requirements (exponents of 2) and the trade-off between homogeneity criteria and segmentation quality. Lower criteria result in clearer images but poorer segmentation. Overall, the paper provides insights into the strengths and weaknesses of these segmentation methods and identifies areas for improvement and consideration in practical applications. Fan and Xiao (2018).

2.4 Image segmentation using a hue-guided quadtree

This paper introduces a splitting-and-region-merging method for image segmentation, focusing on finding object boundaries accurately. The method employs quadtree segmentation to establish relationships between blocks, treating each block as a seed and merging those with similar hue angles between adjacent regions. Utilizing the Lab2000HL color space, which closely aligns with human color perception, the proposed system effectively reduces unnecessary blocks while accurately identifying object boundaries. Experimental results validate the efficacy of the method in achieving precise boundary detection. The approach is based on the hue angle difference, with the quadtree initially dividing the image to ensure block uniformity. Then, by merging homogeneous regions step by step, larger regions are formed, leading to the identification of the main object boundary and the reduction of unnecessary blocks. Currently, the experimentation

focuses on hue angles, but future work will explore additional techniques for quadtree-image segmentation, potentially incorporating features such as brightness, texture, etc. This suggests a commitment to further enhancing the segmentation method by integrating additional relevant features for improved accuracy and versatility. Ho and Tsai (2019)

2.5 A Novel Approach to Image Steganography Using Quadtree Partition

This paper introduces a novel approach to image steganography utilizing quadtree partitioning. The quadtree partitioning method divides the image based on pixel value variations, distinguishing between fine and coarse-grained areas within the image. The rationale is akin to hiding something in dense roadside scrub rather than in plain sight on a road. The authors identify areas with frequent and random pixel changes as ideal for concealing secret messages. The proposed algorithm searches for complex textured areas by employing quadtree partitioning with a threshold of 0.1. It then operates on 2×2 blocks derived from this partitioning, concealing a message's bitstream within these blocks using the least significant bit method. The paper discusses two approaches for retrieving the hidden message from the stego-image. Additionally, it suggests extending the concept to color images and embedding one image within another. In complex textured areas, multiple least significant bits may be used to hide the message or image bits. Kumar (2016)

2.6 Quadtree-based KNN Search on Road Networks

In this paper, we proposed a quadtree-based KNN search algorithm for moving objects on road networks. The algorithm uses quadtree cells to extend the query scope, effectively prunes the entire search space, provides a suitable query scope, and reduces the query response time. Experimental results show that the proposed algorithm has a better performance on large and unevenly distributed moving object datasets. Wen and Xiong (2017)

2.7 Improved quadtree image segmentation approach to region information

This paper discusses the importance of image segmentation in various image-processing applications, such as pattern recognition and image analysis. It categorizes segmentation algorithms based on the methods they employ, including single or multiple thresholding, edge detection, region similarity, clustering, artificial neural networks (ANN), and fuzzy logic techniques. The authors implement six segmentation algorithms, namely

Otsu's algorithm, K-means, quadtree, Delta E, region growth, and the algorithms, in MATLAB. They evaluate the performance of these algorithms on six simple and complex images from the literature. The results indicate that the efficacy of the segmentation algorithms varies depending on the complexity of the input image. For simple images with a single object, Delta-E performs better compared to other algorithms. However, Otsu's and K-means algorithms may incorrectly identify the background as an object. As the complexity of the input image increases, the performance of the segmentation algorithms degrades. While all algorithms can detect a tiger in a moderately complex image, none can accurately segment objects in a complex image like "man woman." The authors suggest that running the algorithms on images with multiple objects might improve performance and leave this for future work. Muhsen, Rehman, Altameem, Saba, and Uddin (2014)

2.8 Use of Quadtrees for Image Segmentation

The usefulness of image segmentation of a quadtree approximation of a 2nx2 n gray level image is examined. Estimates of the thresholds i defining object regions Ri in an image are obtained by analyzing the histogram of an image reconstructed from the quadtree. This information is used with the quadtree data structure to find the approximate location of the Ri within the image. A separate local threshold j to extract each Ri is then calculated from a histogram of those areas in the original image that are known to be in the vicinity of R The threshold j is then applied to the original image locally in the vicinity of R to yield a region Ri similar to R'. The procedure is then repeated for other thresholds i+a, where a is a small interval, to yield regions q (k = i-c,i,i+) similar to R. The best q is chosen by correlating each one with an edge map of the original image. Ranade, Rosenfeld, and Prewitt (1980)

2.9 Quad-tree segmentation for texture-based image query

The paper focuses on using quad-tree segmentation for texture-based image queries. The goal is to develop an efficient and effective method for retrieving images based on their texture content. The quad-tree structure is employed to segment the images into regions with similar texture properties. The quad-tree is constructed by recursively dividing the image into quadrants until termination conditions are met. The termination conditions are typically based on the homogeneity of texture within a region. Once the quad-tree segmentation is obtained, texture descriptors are computed for each region. These de-

scriptors capture the texture characteristics of the segmented regions. The descriptors can be based on statistical measures, such as texture histograms or co-occurrence matrices, or they can utilize more advanced techniques, such as wavelet transforms. To perform a texture-based image query, the query image is first segmented using the same quad-tree approach. The texture descriptors for the regions in the query image are computed. Then, a similarity measure, such as Euclidean distance or correlation, is used to compare the descriptors of the query image regions with those of the database images. The paper discusses various aspects of quad-tree segmentation for texture-based image queries, including the choice of termination conditions, the selection of texture descriptors, and the similarity measures used for matching. It also presents experimental results that demonstrate the effectiveness of the proposed method in retrieving images based on their texture content. Overall, the paper provides insights into the use of quad-tree segmentation as a technique for efficient and accurate texture-based image queries. It highlights the importance of considering both the segmentation process and the choice of texture descriptors to achieve optimal retrieval performance. Smith and Chang (1994)

2.10 Image Segmentation Using Quadtree-Based Similarity Graph and Normalized Cut

The paper focuses on image segmentation using a quadtree-based similarity graph and the normalized cut algorithm. The aim is to efficiently divide an image into meaningful regions based on the similarity of pixel values. The quadtree structure is used to partition the image into regions of varying sizes. Each node in the quadtree represents a rectangular region, and the tree is constructed by recursively subdividing the image until termination conditions are met. The termination conditions can be based on factors such as a maximum level of detail or a region becoming homogeneous. Once the quadtree is constructed, a similarity graph is built, where each node in the quadtree corresponds to a vertex in the graph. The similarity between regions is determined by comparing pixel values within and across regions. Various similarity measures can be used, such as color similarity or texture similarity. The normalized cut algorithm is then applied to the similarity graph to perform image segmentation. This algorithm aims to find a partition of the graph that minimizes the normalized cut, which is a measure of dissimilarity between different regions. By minimizing the normalized cut, the algorithm identifies meaningful boundaries between regions. The paper discusses the details of constructing the quadtree-based similarity graph and applying the normalized cut algorithm for im-

age segmentation. It also presents experimental results to demonstrate the effectiveness of the proposed method in segmenting images. Overall, the paper proposes a method that combines the quadtree structure, similarity graph construction, and normalized cut algorithm to achieve image segmentation. This approach provides an efficient and effective way to divide an image into regions based on pixel similarity, leading to meaningful segmentation results. Fu et al. (2013)

2.11 Segmentation for High-Resolution Optical Remote Sensing Imagery Using Improved Quadtree and Region Adjacency Graph Technique

The paper addresses the problem of segmentation for high-resolution optical remote sensing imagery using an improved quadtree and region adjacency graph technique. The objective is to accurately and efficiently partition the imagery into meaningful regions for further analysis and interpretation. The proposed method begins with the construction of an improved quadtree structure. The quadtree is built by recursively subdividing the image until termination conditions are met. The improvement lies in the use of adaptive thresholding techniques to determine the termination conditions, taking into account the local characteristics of the image. This ensures more accurate and adaptive segmentation results. Once the quadtree is constructed, a region adjacency graph (RAG) is created to represent the relationships between adjacent regions. The RAG provides a compact and efficient representation of image segmentation. It allows for the incorporation of spatial information and facilitates subsequent analysis tasks. The paper also introduces a set of features to characterize the regions obtained from the quadtree segmentation. These features capture various properties such as color, texture, and shape. They are computed for each region and used to further refine the segmentation results. To evaluate the proposed method, the authors conduct experiments on high-resolution optical remote sensing imagery. They compare the performance of their approach with other state-of-the-art segmentation methods. The evaluation includes metrics such as segmentation accuracy, boundary adherence, and computational efficiency. The results demonstrate that the improved quadtree and region adjacency graph technique achieves accurate and efficient segmentation for high-resolution remote sensing imagery. The method effectively handles complex scenes, preserves object boundaries, and provides a compact representation for subsequent analysis tasks. In summary, the paper presents an improved quadtree and region adjacency graph technique for the segmentation of high-resolution optical remote sensing imagery. The approach offers accurate segmen-

tation results by adaptively determining termination conditions, incorporates spatial relationships through the region adjacency graph, and employs a set of features for further refinement. de Carvalho, Ferreira, and Costa (2010)

2.12 Improved Quadtree Method for Split Merge Image Segmentation

The paper introduces an improved quadtree method for split-merge image segmentation. The goal is to accurately partition an image into meaningful regions by iteratively splitting and merging the quadtree structure. The proposed method starts with an initial quadtree representation of the image, where each node represents a rectangular region. The splitting process is then applied to refine the initial quadtree. This is done by evaluating the homogeneity of each node based on certain criteria, such as color similarity or texture consistency. If a node is deemed heterogeneous, it is split into four child nodes, resulting in a more detailed representation of the image. After the splitting phase, the merging process is performed to reduce the number of regions and achieve a more coherent segmentation. The merging is done by evaluating the similarity between adjacent nodes and determining if they should be merged based on predefined similarity thresholds. This step aims to merge neighboring regions that exhibit similar characteristics. To improve the overall segmentation quality, the paper introduces two additional techniques. First, a region boundary refinement method is applied to enhance the accuracy of region boundaries by adjusting the merging decisions near boundaries. Second, an adaptive thresholding method is employed to dynamically adjust the similarity thresholds during the merging process, considering the local characteristics of the image. The proposed method is evaluated using various benchmark datasets, and its performance is compared to other state-of-the-art segmentation approaches. The evaluation includes metrics such as segmentation accuracy, boundary adherence, and computational efficiency. The experimental results demonstrate that the improved quadtree method achieves competitive segmentation accuracy while effectively reducing computational complexity compared to other methods. The adaptive thresholding and boundary refinement techniques contribute to further improving the segmentation quality and boundary accuracy. In summary, the paper presents an improved quadtree method for split-merge image segmentation. The method combines splitting and merging operations, along with adaptive thresholding and boundary refinement techniques, to achieve accurate and efficient image segmentation. Kelkar and Gupta (2008)

2.13 Efficient region segmentation on compressed gray images using quadtree and shading representation

The paper focuses on efficient region segmentation of compressed gray images using a combination of quadtree and shading representation techniques. The objective is to achieve accurate segmentation results while minimizing computational complexity, particularly for compressed grayscale images. The proposed method starts by decompressing the grayscale image and converting it into a quadtree structure. The quadtree represents the image as a hierarchy of rectangular regions, with each node representing a region at a certain level of detail. The quadtree construction is based on the image's intensity variations and texture properties. To efficiently represent the shading information within each region, a shading representation technique is introduced. This technique captures the local shading characteristics of the image, which can aid in distinguishing between different regions. The shading representation is computed for each quadtree node or region. Next, a region segmentation algorithm is applied to the quadtree structure and the shading representations. This algorithm uses the quadtree structure to guide the segmentation process and incorporates the shading information to refine the segmentation boundaries. The goal is to accurately identify the regions in the image based on both intensity variations and shading cues. To evaluate the proposed method, experiments are conducted on compressed grayscale images, and the results are compared against other segmentation approaches. Performance metrics such as segmentation accuracy, boundary adherence, and computational efficiency are considered. The experimental results demonstrate that the combination of quadtree and shading representation techniques leads to efficient and accurate region segmentation of compressed grayscale images. The method effectively captures the local shading characteristics, which improves the segmentation quality and boundary delineation. Additionally, the proposed method shows computational efficiency, making it suitable for real-time or resource-constrained applications. In summary, the paper presents an efficient region segmentation method for compressed gray images using quadtree and shading representation. The combination of these techniques enables accurate segmentation results while minimizing computational complexity. The method has potential applications in various fields, including image compression, computer vision, and pattern recognition. Chung, Huang, and Lu (2004)

2.14 Image segmentation on cell-center sampled quadtree and octree grids

The paper discusses image segmentation using cell-center sampled quadtree and octree grids. The objective is to efficiently partition an image into meaningful regions using hierarchical grid structures. The method utilizes quadtree grids for 2D image segmentation and octree grids for 3D image segmentation. These hierarchical grid structures divide the image or volume into cells, with each cell representing a region of interest. The cells are sampled at their center points, providing a compact representation of the image. In the case of 2D image segmentation, a quadtree grid is constructed by recursively subdividing the image into quadrants until termination conditions are met. The termination conditions can be based on factors like a maximum level of detail or the homogeneity of regions. For 3D image segmentation, an octree grid is constructed by recursively dividing the volume into octants. Again, termination conditions are employed to determine when to stop the subdivision process. After constructing the grid structure, a segmentation algorithm is applied to assign labels to the cells based on their properties. Various techniques can be employed, such as thresholding, clustering, or region growing, depending on the specific application. The advantages of using cell-center sampled grids for image segmentation include their hierarchical nature, which allows for adaptive and multi-scale segmentation, and their compact representation, which reduces memory requirements. The paper presents experimental results and evaluates the proposed method using different image datasets. The evaluation includes metrics such as segmentation accuracy, boundary adherence, and computational efficiency. Overall, the paper demonstrates that image segmentation using cell-center sampled quadtree and octree grids provides an efficient and effective approach for partitioning images or volumes into meaningful regions. The hierarchical grid structures and compact representation offer flexibility, adaptability, and computational advantages for various segmentation tasks.

Kim and Tsiotras (2009)

CHAPTER 3
METHODOLOGY

Image segmentation, the process of partitioning an image into meaningful regions, is a crucial task in computer vision and image processing. Traditional segmentation methods often struggle with complex images due to their limited ability to adaptively capture local variations and structures. To address this, quadtree-based segmentation has emerged as a promising approach. Quadtree segmentation leverages hierarchical decomposition, recursively dividing an image into quadrants based on local characteristics. This methodology section presents a systematic framework for quadtree image segmentation, aiming to provide a clear understanding of its implementation and evaluation. One potential benefit of quadtree segmentation is its ability to handle complex images more effectively than traditional methods. By locally adapting to image characteristics, it can capture finer details and structures, leading to more accurate segmentation results. Additionally, the hierarchical nature of quadtree segmentation allows for efficient processing, particularly for large-scale images. However, there are also potential drawbacks to consider. Quadtree segmentation may introduce computational overhead due to the recursive nature of the algorithm, particularly for images with high levels of detail. Additionally, determining the optimal parameters for quadtree construction and segmentation can be challenging and may require extensive experimentation. Furthermore, the effectiveness of quadtree segmentation may vary depending on the specific characteristics of the images being segmented. In this section, we detail the key stages of quadtree segmentation, including image preprocessing, quadtree construction, segmentation, and evaluation. We outline the algorithms, parameters, and experimental setup used to assess the performance of the quadtree segmentation algorithm.

By following this methodology, researchers can gain insights into the effectiveness and limitations of quadtree segmentation for various image analysis tasks. This structured approach facilitates experimental validation and comparison with existing methods, paving the way for advancements in image processing and computer vision applications.

3.1 Image Preprocessing:

Image preprocessing plays a crucial role in preparing the input data for segmentation. This section outlines the preprocessing steps applied to the input images, including

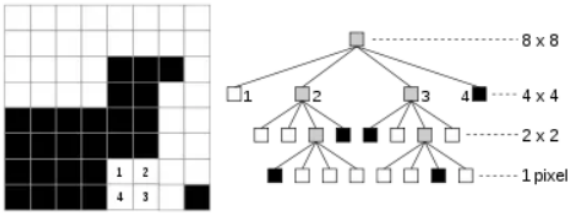
grayscale conversion, normalization, and any noise reduction techniques.

- The first step is to load the image
- The second step is to convert the input image to grayscale since it's common and easy to segment an image
- The third step is to filter the input image by using Gaussian blur to reduce noise.

3.2 Quadtree Construction:

The quadtree data structure serves as the foundation for hierarchical image representation in quadtree segmentation. This section describes the process of quadtree construction, starting from the root node and recursively subdividing regions based on local characteristics.

Figure 3.1



Source:https://medium.com/analytics-vidhya/transform-an-image-into-a-quadtree-39b3aa6e019a

A quadtree is a tree data structure that is particularly useful for spatial indexing of two-dimensional data. It is commonly used in computer graphics, geographic information systems (GIS), and collision detection algorithms. The main idea behind a quadtree is to recursively divide a space into four quadrants, or "children" until a certain condition is met or a predetermined level of depth is reached. Each node in the quadtree represents a region of the space, and leaf nodes typically represent individual points or objects.

Here's a basic explanation of how a quadtree works:

- Dividing the Space: Initially, the entire space is represented by a single node, known as the root node. This root node covers the entire space.
- Partitioning: As the quadtree is built, each node is recursively subdivided into four equal quadrants (children). These quadrants are usually referred to as northeast, northwest, southeast, and southwest.
- Insertion: When inserting a point or object into the quadtree, the algorithm traverses down the tree from the root node to the appropriate leaf node based on the

location of the point. At each level, it determines which quadrant the point belongs to and descends into that quadrant.

- Node Splitting: Nodes can have a maximum capacity, meaning they can only hold a certain number of points or objects. When this capacity is reached, the node is split into its four children, and the points or objects are redistributed among them. This process continues recursively as needed.
- Querying: When querying the quadtree, such as for finding points within a given region or for collision detection, the algorithm traverses the tree, examining nodes and their contents to determine which points or objects are relevant to the query.
- Pruning: In some implementations, quadtrees may be pruned to remove empty or unnecessary branches, optimizing memory usage and query performance.

3.3 Segmentation Algorithm:

The segmentation algorithm extracts meaningful regions from the quadtree representation of the image.

- Traverse Quadtree: Traverse the quadtree to perform segmentation. Start from the root node and recursively visit each node.
- Leaf Node Analysis: At each leaf node, analyze the image region represented by the node.
- Thresholding Decision: Determine whether to further split the node or apply thresholding based on a chosen statistic (e.g., variance, mean intensity). For example, if the variance of a node's region is below a threshold, apply thresholding directly.
- Thresholding: Apply thresholding techniques (e.g., global thresholding, adaptive thresholding) to segment the region into binary foreground and background regions.
- Recursive Splitting: If further splitting is required based on the chosen statistic, recursively split the node into quadrants and repeat the segmentation process.

3.4 Evaluation Methodology:

Evaluation is essential for assessing the performance of the quadtree segmentation algorithm. This section defines quantitative metrics for evaluating segmentation accuracy, computational efficiency, and robustness. It also outlines the experimental setup, including datasets used, parameter configurations, and comparison with ground truth

annotations or other segmentation methods.

- Segmentation Accuracy: Segmentation Accuracy is implemented by using Intersection over Union, which is calculated by

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}} \tag{3.1}$$

  It measures the overlap between the segmented regions and ground truth annotations.
- Complexity Analysis: In this analysis, we compare the complexities between the proposed algorithm, Brute Force, and algorithms given by opencv (Mean-Shift Clustering, etc.)

3.5 Optimization and Fine-tuning:

Optimization techniques are employed to improve the performance of the quadtree segmentation algorithm. Explore the impact of different parameters on the segmentation results, including:

- Threshold values: Investigate how varying the threshold for splitting quadrants affects the level of detail and accuracy of the segmentation.
- Maximum splitting level: Determine the optimal maximum level of recursion to balance detail and computational cost.
- Standard deviation threshold: Evaluate the effect of adjusting the threshold for splitting based on standard deviation.

CHAPTER 4

Pseudocode

4.1 Pseudocode

Node Class:

- **Attributes:**
  - `image`: Image data associated with the node
  - `is_leaf`: Indicates whether the node is a leaf or not
  - `level`: Current level of the node in the tree
  - `quad_tree`: Reference to the quadtree instance
- **Constructor:**
  - Initialize node with provided image and quadtree reference
- **split Method:**
  - Increase node level by 1
  - Calculate height and width of the image
  - Get the standard deviation from the root node
  - If std_deviation(image) $\geq$ std_threshold or (height $>$ base_element and width $>$ base_element):
    - Split image into quadrants (top_left, top_right, bottom_left, bottom_right)
    - Create child nodes for each quadrant
    - Recursively split child nodes
  - Else:
    - Apply thresholding to the image in which the quadrant having the total cost less than the standard deviation from the root node will be labeled as 0, otherwise will be 255.

QuadTree Class:

- **Attributes:**
  - `threshold`: Threshold value for splitting nodes
  - `root`: Root node of the quadtree
- **Constructor:**
  - Initialize quadtree with provided image and threshold
- **split Method:**
  - Start recursive splitting from the root node

4.2 Explanation:

The main idea behind this pseudocode is to create a hierarchical data structure (quadtree) to partition the input image into smaller regions based on certain criteria (e.g., standard deviation threshold). This approach allows for adaptively capturing local variations and structures within the image, ultimately facilitating image segmentation. The quadtree is recursively split until the specified conditions for splitting are no longer met, at which point thresholding is applied to the image to create meaningful segmentation results.

Node Class:

- **Attributes:**
  - `image`: Represents the image data associated with the node.
  - `is_leaf`: Indicates whether the node is a leaf node or not. Initially, it's set to True.
  - `level`: Denotes the current level of the node within the quadtree.
  - `quad_tree`: References the quadtree instance to which the node belongs.
- **Constructor:**
  - Initializes a node with the provided image data and a reference to the quadtree instance.
- **split Method:**
  - This method recursively splits the node and its descendants into quadrants until certain conditions are met. Here's what it does:
    - Increases the node's level by 1.
    - Calculates the height and width of the image associated with the node.
    - Calculates the standard deviation threshold based on the quadtree's threshold.
    - If the standard deviation of the image is greater than or equal to the threshold, or if the dimensions of the image exceed a certain base element size:
      - Splits the image into quadrants (top_left, top_right, bottom_left, bottom_right).
      - Creates child nodes for each quadrant and initializes them with the respective image data.
      - Recursively calls the split method on each child node.
    - Otherwise, if the conditions are not met, the thresholding is applied to

the image.

CHAPTER 5

Actual Code

5.1 Node Class

```
class Node:
    def __init__(self, image, quad_tree, is_leaf=True):
        self.image = image
        self.is_leaf = is_leaf
        self.level = 0
        self.quad_tree = quad_tree

    def split(self):
        self.level += 1
        height, width = self.image.shape
        half_height, half_width = height // 2, width // 2
        # Access threshold from QuadTree
        std_threshold = self.quad_tree.threshold


        if np.std(self.image) >= std_threshold
        or (height > 2 and width > 2):
            # Split the image into quadrants
            top_left = self.image[:half_height, :half_width]
            top_right = self.image[:half_height, half_width:]
            bottom_left = self.image[half_height:, :half_width]
            bottom_right = self.image[half_height:, half_width:]

            # Create child nodes
            self.top_left_node = Node(top_left,
            self.quad_tree, is_leaf=False)
            self.top_right_node = Node(top_right,
            self.quad_tree, is_leaf=False)
            self.bottom_left_node = Node(bottom_left,
```

```
            self.quad_tree, is_leaf=False)
            self.bottom_right_node = Node(bottom_right,
            self.quad_tree, is_leaf=False)

            # Recursively split child nodes
            self.top_left_node.split()
            self.top_right_node.split()
            self.bottom_left_node.split()
            self.bottom_right_node.split()
        else:
            # Apply thresholding
            quadrants = [self.image[:half_height, :half_width],
                         self.image[:half_height, half_width:],
                         self.image[half_height:, :half_width],
                         self.image[half_height:, half_width:]]
            for i, quadrant in enumerate(quadrants):
                theta = 0.8
                mean = np.mean(quadrant)
                std = np.std(quadrant)
                totalCost = mean * theta + (1 - theta) * std

                if totalCost < std_threshold:
                    quadrants[i][:] = 0
                else:
                    quadrants[i][:] = 255
```

5.2 QuadTree Class

```
class QuadTree:
    def __init__(self, image, threshold):
        self.threshold = threshold
        self.root = Node(image, self)

    def split(self):
        self.root.split()
```

CHAPTER 6
Result

6.1 Complexity Analysis

Complexity analysis is a fundamental aspect of algorithmic design and evaluation, providing insights into how the performance of an algorithm scales with respect to the size of the input. In this section, we delve into the complexity analysis of the QuadTree algorithm, specifically focusing on the recurrence relation that governs the splitting process.

Understanding the complexity of the QuadTree algorithm is crucial for assessing its efficiency and scalability in handling image data. By analyzing the recurrence relation derived from the splitting process, we can ascertain the algorithm's time complexity, which indicates how the computational time increases as the size of the input image grows.

Complexity analysis involves assessing both time complexity, which quantifies the number of computational steps an algorithm requires, and space complexity, which measures the amount of memory or storage space needed.

6.1.1 Time Complexity Analysis

To analyze the recurrence of the splitting process in the QuadTree algorithm, we need to examine how many times the `split()` function is called and how the size of the image changes with each split.

Let's denote:

- n as the number of pixels
- $T(n)$ as the number of times the `split()` function is called.

When splitting the image, it divides into four quadrants, each with half the height and half the width of the original image. Therefore, each time we split, the size of the image reduces to one-fourth of the original size.

Now, let's analyze the recurrence relation:

In the `split()` function, we recursively call `split()` on four child nodes if the condition is met; otherwise, we terminate.

- Each time we split, we create four child nodes.
- The size of the image reduces to one-fourth with each split.

Based on this analysis, we can express the recurrence relation as follows:

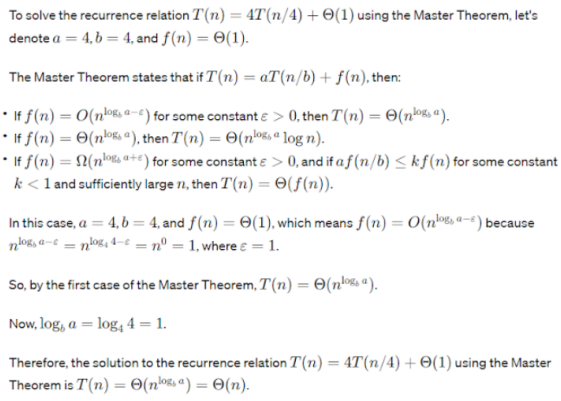
$$T(n) = 4T(n/4) + \Theta(1)$$

Where:

- $T(n)$ is the number of times the `split()` function is called for an image.
- $4T(n/4)$ represents the four recursive calls on child nodes.
- $\Theta(1)$ represents the constant time operations within each function call.

We can solve the recurrence relation $T(n)$ using the master theorem.

Figure 6.1



To solve the recurrence relation $T(n) = 4T(n/4) + \Theta(1)$ using the Master Theorem, let's denote $a = 4, b = 4$, and $f(n) = \Theta(1)$.

The Master Theorem states that if $T(n) = aT(n/b) + f(n)$, then:

- If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
- If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$.
- If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if $af(n/b) \leq kf(n)$ for some constant $k < 1$ and sufficiently large n, then $T(n) = \Theta(f(n))$.

In this case, $a = 4, b = 4$, and $f(n) = \Theta(1)$, which means $f(n) = O(n^{\log_b a - \varepsilon})$ because $n^{\log_b a - \varepsilon} = n^{\log_4 4 - \varepsilon} = n^0 = 1$, where $\varepsilon = 1$.

So, by the first case of the Master Theorem, $T(n) = \Theta(n^{\log_b a})$.

Now, $\log_b a = \log_4 4 = 1$.

Therefore, the solution to the recurrence relation $T(n) = 4T(n/4) + \Theta(1)$ using the Master Theorem is $T(n) = \Theta(n^{\log_b a}) = \Theta(n)$.

Master theorem to get the complexity

6.2 Space Complexity Analysis

To analyze the space complexity of the provided code, we need to consider the memory consumption at each step of the QuadTree splitting process. The space complexity is primarily influenced by the memory used to store the image data and the additional memory required for the recursive function calls and the creation of Node objects.

Image Data: The original image data is stored in memory. This requires space proportional to the size of the image, which is $O(n^2)$, where n is the dimension of the image.

Node Objects: Each node in the QuadTree is represented by a Node object. The memory required for these objects depends on the number of nodes created during the splitting process. Since each node stores references to child nodes, its own image data,

and other attributes, the space complexity related to node objects can be expressed as $O(\text{number of nodes})$.

Recursive Calls: During the splitting process, the `split()` function is called recursively on child nodes. This incurs additional space for function call stack frames. The maximum depth of the recursion tree corresponds to the maximum level of splitting in the QuadTree, which depends on the size of the input image. Therefore, the space complexity due to recursive calls can be expressed as $O(\text{maximum depth of recursion})$.

Considering these factors, the space complexity of the provided code can be summarized as follows:

$$\text{Total space complexity} = O(n^2) + O(\text{number of nodes}) + O(\text{maximum depth of recursion})$$

Determining the exact number of nodes created and the maximum depth of recursion requires a detailed analysis of the splitting process and the specific characteristics of the input image and threshold values. Additionally, the space complexity may vary depending on implementation details such as memory management overhead and data structure choices.

In practical terms, the space complexity of the QuadTree algorithm is typically dominated by the storage of image data, while the memory overhead for node objects and recursive function calls is relatively small compared to the

CHAPTER 7
Evaluation

Evaluation is a pivotal aspect in assessing the efficacy of the quadtree segmentation algorithm. It serves as a crucial step in quantifying segmentation accuracy, computational efficiency, and overall robustness. Establishing comprehensive evaluation metrics is fundamental to providing a nuanced understanding of algorithm performance.

In delineating our evaluation methodology, we adopt a multifaceted approach that encompasses various facets such as experimental setup, utilized datasets, parameter configurations, and comparative analyses against ground truth annotations or alternative segmentation methods. By incorporating these elements, we aim to provide a rigorous evaluation framework that offers insights into both the quantitative and qualitative aspects of segmentation results.

To gauge the accuracy of segmentation, we employ Intersection over Union (IoU) as the primary metric. IoU quantifies the overlap between segmented regions produced by different algorithms. Specifically, we compare the segmented images generated by the proposed quadtree algorithm against those obtained through the utilization of OpenCV, a widely recognized library for image processing and computer vision tasks. By leveraging IoU, we can ascertain the degree of similarity between the outputs of the two algorithms, thereby facilitating a comprehensive assessment of segmentation accuracy.

In addition to IoU, we incorporate Peak Signal-to-Noise Ratio (PSNR) as another key metric for assessment. PSNR quantifies the fidelity of segmented regions compared to the original image, offering insights into image preservation during the segmentation process. Integrating PSNR alongside IoU allows us to capture both spatial accuracy and image fidelity aspects, thereby providing a more comprehensive evaluation of segmentation quality.

Beyond accuracy assessment, we delve into the analysis of time complexity to elucidate the computational efficiency of the quadtree segmentation algorithm. Time complexity analysis provides valuable insights into the computational resources required by the algorithm as a function of input size or other relevant parameters. By evaluating the time complexity, we gain a deeper understanding of the algorithm's scalability and perfor-

mance characteristics under varying conditions.

In addition to time complexity, we also conduct an analysis of space complexity to further elucidate the resource utilization patterns of the quadtree segmentation algorithm. Space complexity analysis focuses on quantifying the amount of memory or storage space required by the algorithm to execute successfully. By examining space complexity, we can identify potential memory usage bottlenecks and assess the algorithm's suitability for deployment in resource-constrained environments.

REFERENCES

Chung, K.-L., Huang, H.-L., & Lu, H.-I. (2004). Efficient region segmentation on compressed gray images using quadtree and shading representation. *Pattern Recognition*, *37*(8), 1591–1605. doi: 10.1016/j.patcog.2004.02.009

de Carvalho, M. A., Ferreira, A. C., & Costa, A. L. (2010). Image segmentation using quadtree-based similarity graph and normalized cut. *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, 329–337. doi: 10.1007/978-3-642-16687-7_45

Fan, W.-Q., & Xiao, W.-S. (2018). Image segmentation based on blob analysis and quadtree algorithm. In *2018 ieee 3rd advanced information technology, electronic and automation control conference (iaeac).* (Preprint) doi: 10.1109/iaeac.2018.8577729

Fu, G., et al. (2013). Segmentation for high-resolution optical remote sensing imagery using improved quadtree and region adjacency graph technique. *Remote Sensing*, *5*(7), 3259–3279. doi: 10.3390/rs5073259

Ho, K.-C., & Tsai, C.-S. (2019). Image segmentation using a hue-guided quadtree. In *2019 ieee eurasia conference on iot, communication and engineering (ecice).* (Preprint) doi: 10.1109/ecice47484.2019.8942787

Hu, Y.-C., Liu, Y.-H., & Chang, I.-C. (2018). Color image coding based on block truncation coding using quadtree segmentation. In *2018 3rd international conference on computer and communication systems (icccs).* (Preprint) doi: 10.1109/ccoms.2018.8463236

Kelkar, D., & Gupta, S. (2008). Improved quadtree method for split merge image segmentation. In *2008 first international conference on emerging trends in engineering and technology.* (Preprint) doi: 10.1109/icetet.2008.145

Kim, B., & Tsiotras, P. (2009). Image segmentation on cell-center sampled quadtree and octree grids. In *Spie proceedings.* (Preprint) doi: 10.1117/12.810965

Kumar, J. (2016). A novel approach to image steganography using quadtree partition. In *2016 2nd international conference on next generation computing technologies (ngct).* (Preprint) doi: 10.1109/ngct.2016.7877396

Muhsen, Z. F., Rehman, A., Altameem, A., Saba, T., & Uddin, M. (2014). Improved quadtree image segmentation approach to region information. *Imaging Science Journal*, *62*(1), 56–62. doi: 10.1179/1743131X12Y.0000000063

Márquez, G. R., Escalante, H. J., & Sucar, L. (2011). Simplified quadtree image segmentation for image annotation. *CEUR Workshop Proceedings*, *719*, 24–34.

Ranade, S., Rosenfeld, A., & Prewitt, J. M. (1980). *Use of quadtrees for image segmentation* (Tech. Rep.). Preprint. doi: 10.21236/ada090243

Smith, J., & Chang, S.-F. (1994). Quad-tree segmentation for texture-based image query. In *Proceedings of the second acm international conference on multimedia.* (Preprint) doi: 10.1145/192593.192676

Wen, Y., & Xiong, H. (2017). Quadtree-based knn search on road networks. In *2017 international conference on computer technology, electronics and communication (icctec).* (Preprint) doi: 10.1109/icctec.2017.00135