Saliency Detection in Remote Sensing Images

Rick Ghosh (208170783), Shashwat Amit Parikh (241040080), and Damini Sarma (232010001)

Abstract

A significant problem in remote sensing is the accurate identification of salient regions from the large images containing heterogeneous objects in complex backgrounds. Some of the pre-existing methods like SLIC, commonly used to solve image segmentation problems have limitations in terms of irregular cluster sizes and higher computational requirements. In this project, we propose a novel algorithm that uses an adaptive threshold-based clustering method that iteratively filters noise, refines clusters and utilizes color similarity and spatial distance to merge the image segments. This algorithm employs KD-Trees to achieve efficient neighbor identification, reduces redundant computations and enhances the connectivity of clusters. The robustness of the algorithm is ensured by imposing size constraints relative to the dimensions of the image. The experimental results from satellite images indicate improved segmentation quality, a reduction in the processing time and maintenance of precise object boundaries in complicated remote sensing scenes. The proposed method is expected to provide an efficient and scalable solution for saliency detection in remote sensing images with potential applications in object detection.

1. Introduction

The advancement of remote sensing techniques like acquiring satellite and aerial imagery from various spaceborne and airborne platforms has generated a large number of high-resolution remote sensing images. The level of spatial detail in these images showcases tremendous applications in areas like urban planning, defense (Hu et al., 2018), environmental monitoring, etc. A major challenge with the processing of these high-resolution remote sensing images is feature extraction and object detection. Further, another fundamental challenge in remote sensing image processing is the detection of salient regions – i.e. areas of interest from among complicated backgrounds. In remote sensing, effective saliency detection techniques involve the reduction of processing time, and computational costs, enhancing the quality of generated outputs, interpretation of complicated landscapes, etc. (Wang et al., 2013). But different constraints are encountered in the process of saliency detection in remotely sensed images like heterogeneity of objects, minute structural details, extensive background areas, etc (Ding et al., 2022).

The conventional saliency detection methods based on super-pixels like SLIC (Simple Linear Iterative Clustering), Mean Shift, K-Means Clustering, etc, have been effective for segmentation tasks involving standard images. SLIC uses a K-means-based clustering method to generate superpixels from pixels and reduces data complexity (Achanta et al., 2012). The SLIC algorithm is efficient but there are limitations when it is applied to high-resolution remote sensing images. For instance, the superpixel shape and size can be inconsistent in image scenes consisting of heterogeneous objects and textures, leading to errors in segmentation. SLIC is also reliant on a fixed set of parameters like the number of superpixels thereby reducing its adaptability and applicability for diverse remote sensing images. The Mean Shift algorithm groups pixels based on color gradients or intensity. This algorithm provides a good adherence to boundaries but there are constraints in the computational efficiency for remote sensing images. As a result, it may produce non-uniform superpixels, produce inaccurate delineation of the salient regions, become increasingly complex for large-scale datasets and produce large processing times. The other saliency detection methods like graph-based methods – NC05 (Normalized Cuts) provide very good adherence to boundaries but there is a limitation in terms of scalability and computational demands. In this algorithm, the superpixels are created by partitioning pixels into graphs that are based on the similarity between features, As a result, the superpixels created are regular and ideal for well-defined objects. However, in the case of large remote-sensing images, this algorithm can require an excessive amount of memory and processing time (Achanta et al., 2012).

The limitations of these existing methods are addressed by proposing a novel algorithm in this project. This algorithm combines efficient clustering with adaptive thresholding to improve saliency detection in remote-sensing images. This method introduces adaptive clustering that refines clusters iteratively based on spatial distance and color characteristics. This enables the algorithm to handle the diversity

and complexity that is characteristic of remote-sensing images in most cases. Further, a KD-Tree structure is incorporated to optimize neighbor search operations and ensure there is efficient connectivity across different clusters. This method also ensures compact and meaningful clusters are generated even for heterogeneous scenes in remote sensing images. This algorithm also includes a step for adaptive noise-filtering to remove clusters below a specific pixel threshold. This also reduces the background noise impact and enhances boundary adherence.

In our proposed algorithm, after initial clustering has been done, a union find structure is further employed that merges the clusters formed based on color similarity. This is useful for maintaining spatial coherence within the clusters and addresses over segmentation which is usually a problem in the traditional saliency detection methods using superpixels. Unlike the SLIC or Mean Shift algorithms, the proposed algorithm can dynamically adjust depending on the complexity of image thereby increasing its robustness. The algorithm also includes a steo where bounding boxes are drawn to better identify the salient regions. These bounding boxes are drawn based on the cluster extents and it provides a simple and scalable solution to mark the salient regions. This approach can be suitable for large datasets, real time applications and remote sensing based applications that are sensitive to time (military reconnaissance, detection of wildfires etc.).

2. Some Existing Super-Pixel Based Methods

Super-pixel algorithms, which group pixels into meaningful clusters, are widely used in image processing. In this section, we will discuss some of the existing algorithms, which can generally be categorized into graph-based and gradient-ascent approaches. Each of these methods offers unique advantages depending on the application. However, as we will outline, our proposed algorithm addresses some of the limitations found in these traditional approaches, potentially offering improved adaptability and efficiency for complex remote sensing images (Achanta et al., 2012).

2.1. Graph-Based Algorithms

In graph-based methods, each pixel is represented as a node within a graph, where the edges connecting these nodes are weighted by the similarity between adjacent pixels. Super-pixels are then formed by optimizing a cost function defined across the graph.

Normalized Cuts: This method recursively partitions a graph that represents the entire image. It uses cues like texture and contours to define partition boundaries, creating super-pixels that are visually balanced and consistent. However, this method performs less effectively in adhering to exact image boundaries and is computationally intense, especially with larger images (Achanta et al., 2012).

Felzenszwalb and Huttenlocher's Method: This approach clusters pixels by treating them as nodes within a graph and grouping them through an agglomerative clustering technique that yields super-pixels corresponding to minimum spanning trees of these pixel groups. This method is noted for its high boundary adherence, though it produces super-pixels with irregular shapes and sizes. It is relatively efficient but lacks precise control over the number and compactness of generated super-pixels (Felzenszwalb & Huttenlocher, 2004) .

2.2. Gradient-Ascent Based Algorithms

Gradient-ascent algorithms, starting from an initial clustering, iteratively refine clusters until they converge to a set criterion, thus forming super-pixels.

Mean Shift: This technique applies mean shift, an iterative method for locating local density maxima within the color or intensity space of an image, defining super-pixels as regions that converge to the same density peak. Although effective, the mean shift algorithm often produces super-pixels with irregular shapes and sizes and can be computationally expensive (Comaniciu & Meer, 2002) .

Quick Shift: Using a mode-seeking procedure, Quick Shift initializes clusters with a medoid shift, then assigns pixels based on density estimates in the feature space. While it offers good boundary adherence, the Quick Shift method is slower than some alternatives and lacks flexibility in controlling super-pixel size and count (Vedaldi & Soatto, 2008).

3. Proposed Methodology

3.1. Traverse-based Adaptive Clustering for Saliency Detection

The methodology proposed here for saliency detection and segmentation of remote sensing images starts with preprocessing of the image to simply it for intensity based segmentation. The input image is first converted to a gray scale image (if it is in a different format) as gray scale images can facilitate image segmentation based on the intensity of a pixel. Next, we apply Otsu's thresholding to estimate an initial intensity threshold that can adapt to varying lighting conditions and diverse image landscapes. This step

of thresholding provides a baseline for the segmentation, which can then be fine-tuned to capture object-specific details.

Next, parameter tuning is an essential step of our algorithm to achieve an adaptable and accurate level of segmentation. The threshold that we obtain from Otsu's thresholding is adjusted to a specific range (say 0–20) by scaling in proportion to the image dimensions and characteristics. To manage noise, we set a noise threshold based on the image dimensions. This threshold determines the minimum size of clusters to be retained and is computed as a function of the total pixel count. In order to account for images at varying scales, non-linear scaling techniques like square-root, logarithmic, or power-law relationships are applied here. For instance, the noise threshold can be calculated as:

$$\text{noise\_threshold} = k \times \sqrt{\text{total\_pixels}}$$

for moderate scaling,

$$\text{noise\_threshold} = k \times \log(\text{total\_pixels})$$

for slower growth, and

$$\text{noise\_threshold} = k \times (\text{total\_pixels})^n \quad \text{where} \quad 0.5 \leq n < 1$$

for customized scaling.

In the next step, we will be doing cluster formation by making use of a region-growing algorithm. This process starts by tracking the unvisited pixels to allow for efficient identification of the next starting point for cluster formation. Each unvisited pixel is then treated as a potential new cluster which then grows using BFS (Bread First Search). The pixels that meet a defined intensity similarity criterion (set by adjusting the initial threshold) are added to the cluster. Once they have been visited, the pixels are marked in order to avoid revisiting. This saves the computation time and ensures efficiency.

The post-processing step involves filtering out noise by removing clusters smaller than the specified noise threshold, as these clusters are likely to represent noise rather than meaningful objects of interest. Only those clusters that meet the predefined criteria, including pixel size and intensity variance, are retained. A list of these significant clusters is maintained for further analysis.

Further, a graph-based approach is used to merge neighboring clusters that are spatially and spectrally similar. For each such cluster, the centroid (computed as the mean of x and y coordinates) is obtained, allowing for the construction of a KD-Tree for efficient neighbor search. Clusters within a specified radius of each other are identified, and those having similar average intensity values, within a color difference threshold, are merged. The union-find data structure is used here to manage connected components efficiently, facilitating the process of merging.

Next, bounding boxes are generated around each significant cluster, helping to localize the objects in an image for further analysis. Only clusters that meet certain pixel size criteria defined by the lower and upper pixel thresholds, are retained, and bounding boxes are drawn around them on a new image to provide a visual representation of detected objects. This is the saliency detection step for the remote sensing images achieved using our proposed algorithm.

In the final step, optimization and performance tuning are applied to the algorithm. To accelerate cluster formation, the list of unvisited pixels is periodically updated which helps to reduce the search space for subsequent clusters. In addition, the noise threshold and initial thresholds are adjusted based on experimental results and specific to the image content. This helps to achieve an optimal balance between noise reduction and the retention of significant image features.

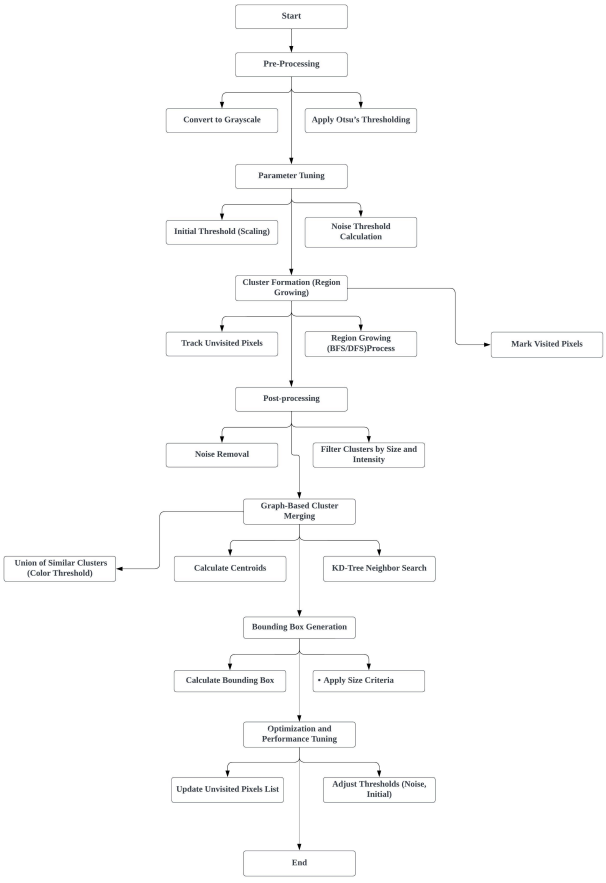


Figure 1. Flowchart of the Proposed Methodology for Saliency Detection

Algorithm 1 Traverse-Based Adaptive Clustering for Saliency Detection

/* Initialization */
Compute the initial threshold using Otsu's method, scaled to a desired range (0 to 20).
Set `noise_threshold` based on image size (e.g., proportional to the square root of total pixels).
Initialize `visited` array of the same size as the image, filled with `False`.
Initialize an empty list `clusters` to hold the resulting clusters.
Initialize the list `unvisited_pixels` with coordinates of all pixels in the image.

repeat
/* Cluster Formation */
Select the next unvisited pixel (i, j) from `unvisited_pixels`.
Initialize a new cluster `cluster = []`.
Initialize a queue `Q = [(i, j)]`.
Set `visited[i, j] = True`.
while `Q` is not empty:
Pop pixel (x, y) from `Q`.
for each neighbor (nx, ny) of (x, y):
if (nx, ny) is within image bounds and `visited[nx, ny] == False`:
if `abs(image[x, y] - image[nx, ny]) <` `threshold`:
Set `visited[nx, ny] = True`.
Add (nx, ny) to the `cluster`.
Append (nx, ny) to queue `Q`.
Add `cluster` to `clusters`.

/* Update Unvisited Pixels */
Recreate the list `unvisited_pixels` by checking all pixels and removing those that are visited.

until `unvisited_pixels` is empty

/* Remove Noisy Clusters */
Remove clusters with size $<$ `noise_threshold`.

/* Cluster Merging */
Compute centroids for each cluster in `clusters`.
Build a KD-tree for fast neighborhood search.
for each centroid i:
Find all centroids within a radius r using the KD-tree.
for each neighboring centroid j:
if the average intensity difference between clusters i and j is below `color_diff_threshold`, merge clusters i and j using a union-find structure.

In the final step, optimization and performance tuning are applied to the algorithm. To accelerate cluster formation, the list of unvisited pixels is periodically updated, which helps to reduce the search space for subsequent clusters. In addition, the noise threshold and initial thresholds are adjusted based on experimental results and specific image content. This helps to achieve an optimal balance between noise reduction and the retention of significant image features.

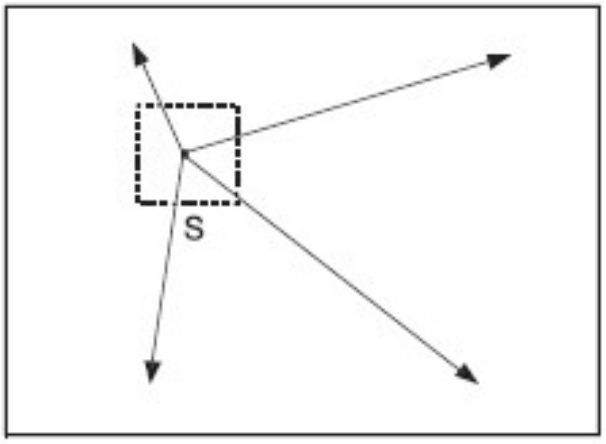


Figure 2. Traverse-based Adaptive Clustering for Saliency Detection

3.2. Algorithm Complexity

The overall complexity of the image processing workflow primarily depends on the pixel-based flood-filling and cluster management operations. Loading the image and calculating basic statistics like mean and standard deviation both operate in O(N) , where N=height×width, representing the total number of pixels. Cluster detection using a flood-fill algorithm also operates in O(N), as each pixel is processed once. Noise filtering, centroid calculations, and coloring processes further iterate over each pixel, maintaining an O(N) complexity. KDTree operations for clustering, including construction and neighbor searches, contribute an additional O(ClogC) complexity, where C is the number of clusters, though typically CN. Union-Find operations, which merge clusters based on proximity, exhibit near-linear behavior with an amortized complexity of O(C(C)), where is the inverse Ackermann function (very slow-growing). Finally, bounding box drawing and colorization iterate across relevant clusters, remaining within O(N). Consequently, the overall complexity is approximately O(N+ClogC), dominated by O(N), making it efficient for practical values of C.

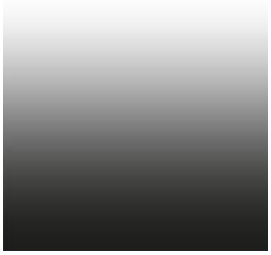
4. Results and Analysis

4.1. Dataset

The dataset used for this project is the Airbus Aircrafts Sample Dataset, which is available on Kaggle. The dataset contains high-resolution satellite imagery with various scenes showing aircraft and related features, representing a complex landscape that is ideal for testing saliency detection methods in remote sensing images. A subset of 100 images was considered from the dataset with a particular focus on images exhibiting greater diversity. The selection of these images is important for a comprehensive assessment of how the proposed method is performing across different background complexities and spatial setups. Each image in the dataset will be useful for verifying the robustness of the proposed clustering-based saliency detection method.

In addition to the Airbus dataset, five unique images were used as an initial testing set for algorithm development:

(a) Bird over Water



(b) Gradient Background

(c) School of Fish

(d) Human Face

(e) Twitter Logo

Figure 3. Initial testing set of images for the traverse-based adaptive clustering algorithm.

1. **Bird over Water**: An image of a bird in flight over a water background, useful for testing the algorithm's ability to isolate objects in motion and identify clear boundaries against a homogeneous background.

2. **Human Face**: A simple image of a human face against a neutral background, that will help to evaluate the algorithm's performance on features and fine details, especially useful for recognizing facial structures.

3. **School of Fish**: An illustration of multiple fish with one distinct, contrasting fish in the group. This image will be useful for examining the algorithm's capability to detect salient features within a repetitive pattern.

4. **Gradient Background**: An image with a gradient from light to dark, which provides a smooth transition in intensity values. This image tests the algorithm's sensitivity to gradual changes and its ability to adapt thresholds based on varying lighting conditions.

5. **Logo (Twitter Icon)**: A simple graphic of a logo with sharp edges and uniform color, which is ideal for testing the algorithm's precision in identifying defined shapes and handling high-contrast, synthetic images.

These initial images will serve as simplified datasets to allow early testing of the algorithm's core functionality in isolating and detecting salient features across a range of visual conditions. Together with the Airbus dataset, this comprehensive collection facilitates the evaluation of the algorithm across both controlled and complex, real-world scenarios

The Traverse-based Adaptive Clustering for Saliency detection was applied to these initial images and the output images obtained can be interpreted as:

1. **Bird over Water (Output Image 1)**: The algorithm successfully identifies the bird as the main salient object, delineating its structure against the background. The red bounding box highlights the bird's shape, demonstrating the algorithm's ability to detect and separate a distinct moving object in a natural environment. This output confirms the method's effectiveness in isolating foreground objects from homogeneous backgrounds like water.

2. **Gradient Background (Output Image 2)**: For the gradient background, the algorithm provides a uniform segmentation, reflecting the even color distribution across the image. This result suggests that the algorithm appropriately treats the gradient as a single segment, respecting the lack of distinctive foreground objects. This output verifies the algorithm's capacity to recognize and adapt to images with smooth transitions and minimal saliency.

3. **School of Fish (Output Image 3)**: The algorithm distinguishes individual fish within a dense pattern by applying bounding boxes to each fish shape. The colorful segmentation demonstrates the algorithm's adaptability in handling repetitive yet distinct objects within close proximity. This result showcases the algorithm's capability for detecting multiple similar objects within a complex scene, useful for analyzing clustered structures in an image.

4. **Human Face (Output Image 4)**: The algorithm effectively segments the face and prominent facial features, placing bounding boxes around distinct areas like the eyes, mouth, and overall face structure. This demonstrates the algorithm's capability in isolating and focusing on facial features, showing potential applicability in human-centered tasks such as facial recognition or feature extraction.

5. **Twitter Logo (Output Image 5)**: The algorithm identifies the logo and segments it into its primary colors with clear boundary definition, encapsulating the logo shape within a bounding box. This output highlights the algorithm's precision in handling high-contrast images with simple, defined shapes, making it suitable for logo and icon segmentation.

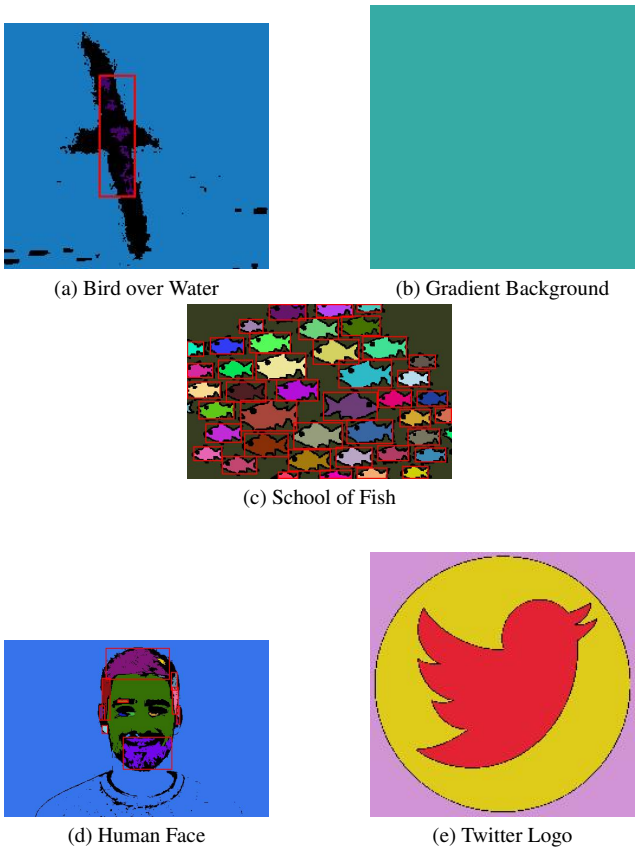
Each output confirms the robustness of the traverse-based adaptive clustering algorithm in handling a variety of visual structures, from natural scenes to synthetic graphics, gradients, and detailed facial elements.

In the next step, we will remote sensing images that can be for testing saliency detection, especially for identifying aircraft, runways, and surrounding infrastructure.

1. **Image 1**: This image shows a large runway with multiple taxiways and aircraft lined up along the edges, ready for takeoff or parked. The layout provides a mix of open space and objects, making it suitable for testing the algorithm's ability to detect and separate aircraft from the runway and grassy areas.

2. **Image 2**: A densely packed airport terminal area with numerous airplanes positioned at the gates. The image includes complex structures with taxiways and terminal buildings. This image challenges the algorithm to identify aircraft closely spaced and recognize detailed structures in a busy airport setting.

3. **Image 3**: A central terminal with a unique, symmetrical design, surrounded by various taxiways and aircraft parking spots. The round and radial structure, with aircraft dispersed around it, tests the algorithm's capability to detect patterns and symmetry within airport layouts.



(a) Bird over Water (b) Gradient Background

(c) School of Fish

(d) Human Face (e) Twitter Logo

Figure 4. Output images generated with the traverse-based adaptive clustering algorithm.

After the Traverse based Adaptive Clustering method for saliency detection is applied, we can observe the same images as:

- **Image 1 Output**: The algorithm has effectively segmented various areas within the airport, such as the runway, taxiways, and terminals, by applying distinct colors and bounding boxes. This segmentation showcases the algorithm's ability to detect and differentiate between the large open spaces (e.g., runway) and more intricate regions (e.g., parked airplanes and smaller buildings), demonstrating its effectiveness in handling an image with both expansive and detailed components.

- **Image 2 Output**: In this output, the algorithm has segmented the dense terminal area, identifying individual aircraft and terminal structures with varying colors and bounding boxes. This result highlights the algorithm's capacity for detecting numerous objects in close proximity, such as aircraft parked at gates, while still maintaining clarity in differentiating each object. This output is particularly useful for understanding crowded ar-

eas within airports.

- **Image 3 Output**: The algorithm has successfully segmented the central terminal complex and surrounding taxiways in this image. The unique radial layout of the terminal is clearly captured with vibrant colors, and distinct aircraft and infrastructure are outlined, showcasing the algorithm's adaptability to more complex and symmetrical designs. This result reflects the algorithm's precision in segmenting intricate architectural layouts within an airport environment.

5. Discussion

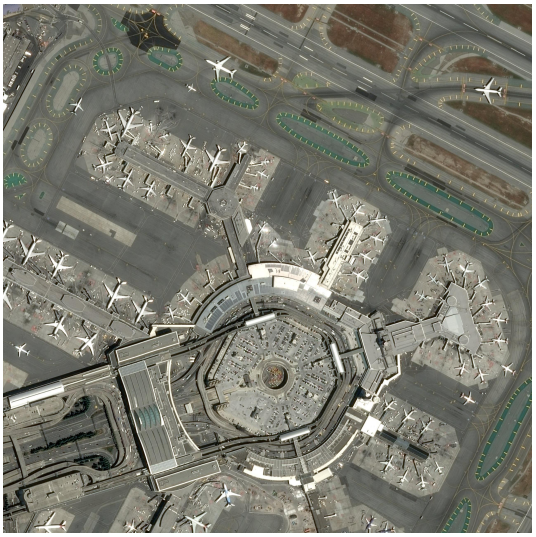
The proposed methodology, Traverse-Based Adaptive Clustering for Saliency Detection, addresses key limitations in traditional segmentation techniques like SLIC, Mean Shift, and NC05, offering a more flexible and efficient approach for saliency detection in remote sensing images. Traditional methods often suffer from constraints like irregular cluster sizes and fixed parameter settings, which reduce adaptability in high-resolution and heterogeneous scenes. The introduction of adaptive thresholding and KD-Trees in our methodology enhances both precision and computational efficiency, allowing it to handle complex backgrounds and maintain object boundaries. This algorithm iteratively refines clusters based on spatial and spectral characteristics, resulting in more accurate segmentations compared to static methods (Achanta et al., 2012; Wang et al., 2013).

The proposed method's scalability is underscored through its efficient processing of diverse scenes, such as the densely populated regions of airport terminals or isolated features like aircraft on runways. Through experiments on both controlled images and complex scenes in the Airbus dataset, the algorithm demonstrates robustness and adaptability across varied spatial and spectral conditions. By merging clusters using a KD-Tree for spatial coherence and filtering noise below a pixel threshold, the algorithm ensures that meaningful clusters are retained while reducing background noise. This approach addresses issues common in remote sensing, such as over-segmentation and inconsistencies in heterogeneous environments (Ding et al., 2022).

Furthermore, the algorithm's adaptability is evident in its ability to perform well across diverse applications, from environmental monitoring to object detection. The bounding box feature, employed post-clustering, enables simple yet effective visualization of detected salient regions, facilitating real-world applications like military reconnaissance or infrastructure monitoring. Our algorithm offers a novel and scalable solution for complex remote sensing imagery, with

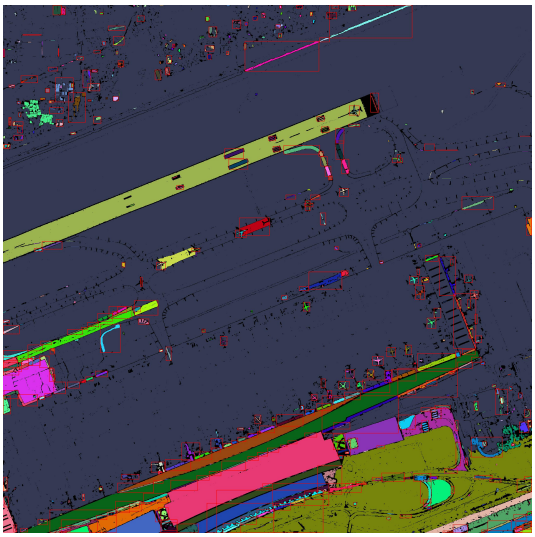
(a) Image of Airport Layout 1

(b) Image of Airport Layout 2

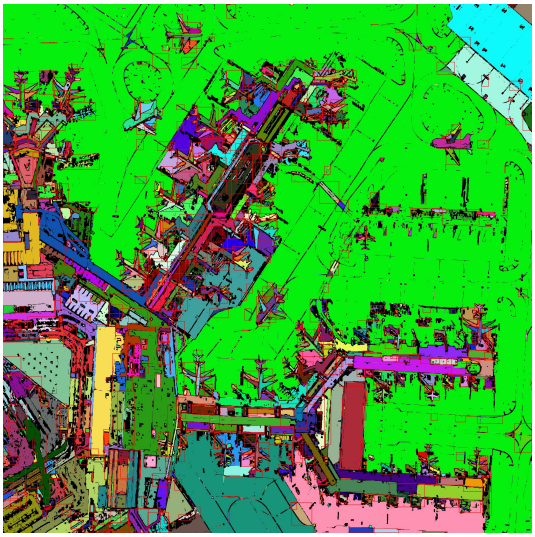


(c) Image of Airport Layout 3

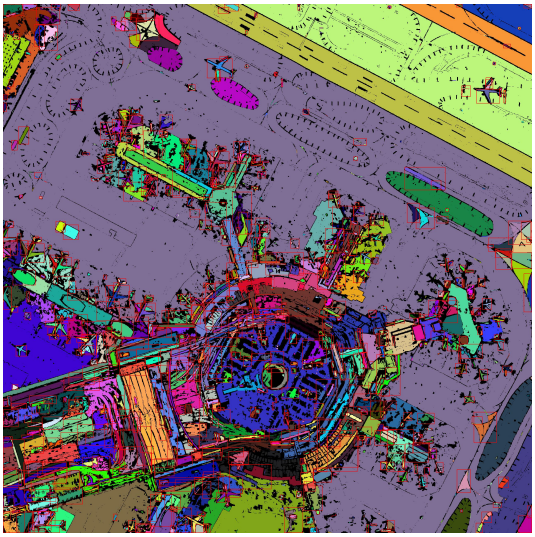
Figure 5. Sample output images generated with the traverse-based adaptive clustering algorithm, showing segmented views of different airport layouts. These images highlight the algorithm's ability to handle complex structures and accurately segment various airport elements.



(a) Segmented Image of Airport Layout 1



(b) Segmented Image of Airport Layout 2



(c) Segmented Image of Airport Layout 3

Figure 6. A sample of 3 output images generated with the traverse-based adaptive clustering algorithm, showing segmented views of different airport layouts. These images highlight the algorithm's ability to handle complex structures and accurately segment various airport elements.

potential applications across a broad range of fields. Future work could focus on optimizing the algorithm for real-time analysis, further improving its utility in time-sensitive applications like wildfire detection and disaster response.

6. Potential Application: Aircraft Detection

A significant potential application of the Traverse-Based Adaptive Clustering method lies in aircraft detection within high-resolution remote sensing images. This approach, if successfully implemented with accurate classification, can play a crucial role in identifying and monitoring aircraft across various terrains, particularly in busy airport settings or areas where aircraft may need to be tracked for security and logistical purposes.

For effective aircraft detection, it is essential that the clustering and segmentation process accurately separates salient regions, such as aircraft bodies, from complex backgrounds. Our algorithm achieves this by iteratively refining clusters based on adaptive thresholds for color and spatial proximity, which is crucial in distinguishing aircraft from similar-looking objects like buildings, vehicles, or infrastructure elements. However, successful detection hinges on the algorithm's ability to classify these segmented regions accurately. Only when classification is successful—meaning the algorithm can accurately differentiate between aircraft and other detected objects—can the method be fully applied for reliable aircraft detection.

Once classification is effectively integrated, this methodology can enable precise identification and localization of aircraft. Bounding boxes around segmented clusters allow for clear and scalable visualization of detected aircraft, facilitating applications such as real-time monitoring in high-traffic airports, military surveillance, and emergency response scenarios. This could be especially beneficial for large-scale airport management or in scenarios where rapid response to unauthorized aircraft is critical.

In summary, while the Traverse-Based Adaptive Clustering method shows promising segmentation results for salient region detection, its potential for aircraft detection will be fully realized with the addition of a robust classification step, ensuring accurate distinction of aircraft from other objects in high-resolution remote sensing images.

References

1. Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., & Süsstrunk, S. (2012). SLIC Superpixels Compared to State-of-the-Art Superpixel Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 34*(11), 2274–2282. https://doi.org/10.1109/TPAMI.2012.120

2. Comaniciu, D., & Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 24*(5), 603–619. https://doi.org/10.1109/34.1000236

3. Ding, L., Wang, X., & Li, D. (2022). Visual Saliency Detection in High-Resolution Remote Sensing Images Using Object-Oriented Random Walk Model. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 15*, 4698–4707. https://doi.org/10.1109/JSTARS.2022.3179461

4. Felzenszwalb, P. F., & Huttenlocher, D. P. (2004). Efficient Graph-Based Image Segmentation. *International Journal of Computer Vision, 59*(2), 167–181. https://doi.org/10.1023/B:VISI.0000022288.19776.77

5. Hu, G., Yang, Z., Han, J., Huang, L., Gong, J., & Xiong, N. (2018). Aircraft detection in remote sensing images based on saliency and convolution neural network. *EURASIP Journal on Wireless Communications and Networking, 2018*(1), 26. https://doi.org/10.1186/s13638-018-1022-8

6. Kumar, J. M. (2020). Image Segmentation using K-means Clustering. *International Journal of Advanced Science and Technology, 29*(6).

7. Li, W., & Pan, C. (2011). Saliency-Based Automatic Target Detection in Remote Sensing Images. In G. Shen & X. Huang (Eds.), *Advanced Research on Computer Science and Information Engineering* (Vol. 153, pp. 327–333). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-21411-0$_5$3.

8. Liu, B., Zhao, L., Li, J., Zhao, H., Liu, W., Li, Y., Wang, Y., Chen, H., & Cao, W. (2021). Saliency-Guided Remote Sensing Image Super-Resolution. *Remote Sensing, 13*(24), 5144. https://doi.org/10.3390/rs13245144.

9. Vedaldi, A., & Soatto, S. (2008). Quick Shift and Kernel Methods for Mode Seeking. In D. Forsyth, P. Torr, & A. Zisserman (Eds.), *Computer Vision – ECCV 2008* (Vol. 5305, pp. 705–718). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-88693-8$_5$2.

10. Wang, X., Lv, Q., Wang, B., & Zhang, L. (2013). Airport detection in remote sensing images: A method based on saliency map. *Cognitive Neurodynamics, 7*(2), 143–154. https://doi.org/10.1007/s11571-012-9223-z.

11. Yang, S., & Zhao, X. (2018). Remote Sensing Image Change Saliency Detection Technology. *Journal of Physics: Conference Series, 1069*, 012110. https://doi.org/10.1088/1742-6596/1069/1/012110.

12. Zhao, D., Wang, J., Shi, J., & Jiang, Z. (2015). Sparsity-guided saliency detection for remote sensing images. *Journal of Applied Remote Sensing, 9*(1), 095055. https://doi.org/10.1117/1.JRS.9.095055.

13. Zhou, H., Wang, X., & Schaefer, G. (2011). Mean Shift and Its Application in Image Segmentation. In H. Kwaśnicka & L. C. Jain (Eds.), *Innovations in Intelligent Image Analysis* (Vol. 339, pp. 291–312). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-17934-1$_1$3.

7. Appendix

7.1. Code

[language=Python, caption=Segmentation and Bounding Box Detection Code] from PIL import Image, ImageDraw import numpy as np import random import cv2 from scipy.spatial import KDTree

Parameters image$_path$ =′ D : $/ImageProcessing/project/bird.jpg′initial_threshold$ = $5noise_threshold$ = $30recluster_threshold$ = $30radius$ = $30color_diff_threshold$ = $40pixel_threshold_lower$ = $100pixel_threshold_upper$ = 5000

Load and preprocess the image image = Image.open(image$_path$).$convert('L')image$ = $np.array(image, dtype$ = $np.int32)height, width$ = $image.shapevisited$ = $np.zeros((height, width), dtype$ = $bool)neighbor_offsets$ = $np.array([[-1, 0], [1, 0], [0, -1], [0, 1]])$

Define function to get neighboring pixels def get$_neighbors(x, y)$: $neighbors$ = $[]foroffsetinneighbor_offsets$: nx, ny = $x + offset[0], y + offset[1]if0 <= nx < heightand0 <= ny < width$: $neighbors.append((nx, ny))returnneighbors$

Define function to process the image into clusters def process$_image(image, threshold)$: $clusters$ = $[]foriinrange(height)$: $forjinrange(width)$: $ifnotvisited[i, j]$: $cluster$ = $[]queue$ = $[(i, j)]whilequeue$: x, y = $queue.pop()ifvisited[x, y]$: $continuevisited[x, y]$ = $Truecluster.append((x, y))fornx, nyinget_neighbors(x, y)$: $ifnotvisited[nx, ny]andabs(image[x, y]$ − $image[nx, ny])$ <= $threshold$: $queue.append((nx, ny))clusters.append(cluster)returnclusters$

Define function to remove noisy clusters def remove$_noisy_clusters(clusters, noise_threshold)$: $return[clusterforclusterinclustersiflen(cluster)$ >= $noise_threshold]$

Process clusters and filter noisy ones clusters = process$_image(image, initial_threshold)filtered_clusters$ = $remove_noisy_clusters(clusters, noise_threshold)$

Calculate centroids of clusters centroids = [] for cluster in filtered$_clusters$: x_coords = $np.array([xforx, yincluster], dtype$ = $np.int64)y_coords = np.array([yforx, yincluster], dtype = np.int64)centroid_x$ = $np.mean(x_coords)centroid_y$ = $np.mean(y_coords)centroids.append((centroid_x, centroid_y))$

Build KDTree for efficient neighborhood search kdtree = KDTree(centroids) edges = [] for i, centroid in enumerate(centroids): neighbors = kdtree.query$_ball_point(centroid, r$ = $radius)forneighbor_indexinneighbors$: $ifneighbor_index!$ = i : $edges.append((i, neighbor_index))$

Define Union-Find class class UnionFind: def $_init_(self, n):self.parent = list(range(n))self.rank = [0] * n$

def find(self, u): if self.parent[u] != u: self.parent[u] = self.find(self.parent[u]) return self.parent[u]

def union(self, u, v): root$_u$ = $self.find(u)root_v$ = $self.find(v)ifroot_u! = root_v$: $ifself.rank[root_u] > self.rank[root_v]$: $self.parent[root_v] = root_uelifself.rank[root_u] < self.rank[root_v]$: $self.parent[root_u] = root_velse$: $self.parent[root_v] = root_uself.rank[root_u] += 1$

Perform merging based on color differences avg$_colors$ = $[sum(image[x, y]forx, yincluster)/len(cluster)forclusterinfiltered_c$ $UnionFind(len(filtered_clusters))foredgeinedges$: i, j = $edgeifabs(avg_colors[i]$ − $avg_colors[j])$ < $color_diff_threshold$: $uf.union(i, j)$

Generate final segmented image with bounding boxes merged$_color_image$ = $np.zeros((height, width, 3), dtype$ = $np.uint8)forcluster_id, cluster_pixelsinmerged_clusters.items()$: $color = [random.randint(0, 255)for_inrange(3)]for(x, y)incluster_pixe$ $merged_color_image[x, y] = color$

Display the final segmented image with bounding boxes merged$_color_image_pil$ = $Image.fromarray(merged_color_image)draw$ = $ImageDraw.Draw(merged_color_image_pil)forclusterinmerged_cluster$ $ifpixel_threshold_lower$ < $len(cluster)$ < $pixel_threshold_upper$: x_coords, y_coords = $zip(*cluster)min_x, max_x$ = $min(x_coords), max(x_coords)min_y, max_y$ = $min(y_coords), max(y_coords)draw.rectangle([min_y, min_x, max_y, max_$ $"red", width = 2)$ merged$_color_image_pil.show()$

7.2. Dataset

The dataset used in this code is from the Airbus Aircrafts Sample Dataset, available at: https://www.kaggle.com/datasets/airbusgeo/airbus-aircrafts-sample-dataset.