COMPUTER VISION

Project-2

Program: Graph based Image Segmentation

Name: **NIKHIL CHEZIAN**
Matriculation Number: **11027706**
Professor: **Dr. MILAN GNJATOVIC**
Date: **15th June 2023**

ABSTRACT

Image segmentation plays a critical role in computer vision applications, enabling the extraction of meaningful information from images by dividing them into distinct regions. This project utilizes the Felzenszwalb segmentation algorithm to segment an input image. The algorithm combines boundary and region-based cues to identify regions with similar attributes. The parameters for segmentation, such as scale and size threshold, are specified to control the segmentation process. The results of segmentation are visualized using a jet colormap, color overlay, and boundary overlay on the original image. The segmented image provides clear differentiation of regions, while the color overlay and boundary overlay offer contextual understanding of the segmentation. The project demonstrates the effectiveness of the Felzenszwalb algorithm in accurately segmenting images and highlights the importance of image segmentation in computer vision tasks.

INTRODUCTION

Image segmentation is the division of an image into distinct, significant parts or regions. It seeks to separate pixels or regions with similar properties from the surrounding environment while grouping them together. The objective is to gather useful data in order to foster higher-level image analysis and comprehension. There are several approaches and techniques for performing image segmentation. The choice of method depends on the specific requirements of the application and the characteristics of the image data.

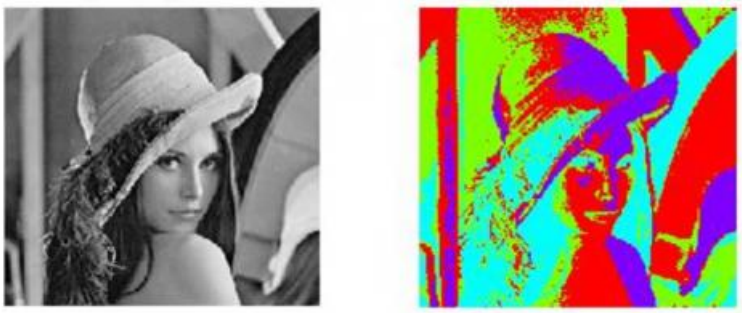

Fig.1: Example of image segmentation.

Graph-based segmentation is a popular approach to image segmentation that leverages the concept of graphs to partition an image into regions. It formulates the segmentation problem as an optimization task on a graph structure, where nodes represent pixels or image regions, and edges represent the relationships between them. The main idea behind graph-based segmentation is to construct a graph in

which the weights of the edges reflect the dissimilarity or similarity between neighboring pixels or regions. The segmentation process then involves finding an optimal cut in the graph that separates regions based on the dissimilarities. This cut corresponds to the segmentation result.

The Felzenszwalb algorithm is a graph-based image segmentation algorithm proposed by Pedro Felzenszwalb and Daniel Huttenlocher. The algorithm groups pixels together based on their similarity in color and proximity, aiming to capture meaningful boundaries and structures in the image. It falls under the category of bottom-up or unsupervised image segmentation methods, as it does not rely on prior knowledge or training data.

A brief of the working principle is elucidated as follows-

- The algorithm starts by creating a graph of the image, treating each pixel as a node.
- It computes the similarity between neighboring pixels based on their colors and proximity.
- The algorithm merges regions by connecting pixels with similar colors and close proximity.
- It updates the region boundaries to define the outer contour of each merged region.
- The merging process continues until all pixels have been processed.
- The goal is to group pixels with similar colors and close spatial relationships together to form meaningful regions.
- Felzenszwalb's algorithm provides an efficient way to segment images into coherent regions by considering both color and spatial information.

CODE IMPLEMENTATION

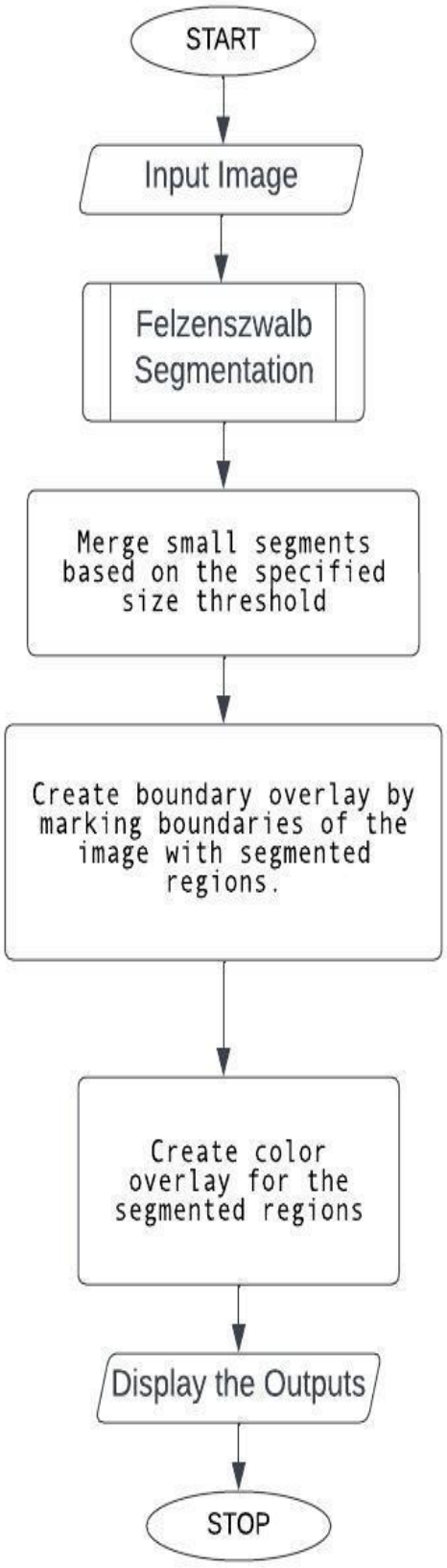

Fig.2: Pictorial representation of the code flow.

1. Image and Parameters:
   - The input image is loaded from the specified file path.
   - The algorithm's parameters, such as scale and size threshold, are set to control the segmentation process.

2. Felzenszwalb Segmentation:
   - The Felzenszwalb algorithm is applied to the image using the skimage.segmentation.felzenszwalb() function.
   - The scale parameter determines the size of the segments.
   - The algorithm constructs a graph representation of the image and merges pixels into segments based on color similarity and edge weights.

3. Segment Merging:
   - Small segments are merged to enhance the segmentation results.
   - The label() function from skimage.measure is used to assign unique labels to each segment.
   - The regionprops() function is employed to calculate properties of each segment.
   - Segments with an area below the specified size threshold are merged by assigning them label 0.

4. Visualization:
   - The boundaries of the segmented regions are visualized using the skimage.segmentation.mark_boundaries() function.
   - The original image, segmented image with a colormap visualization, segmented image with a color overlay, and boundary overlay are displayed using matplotlib.pyplot.

RESULT & ANALYSIS

The code was run multiple times to compare and analyze the impact of variation in parameters.

Constant Parameter: Sigma =1.5, Parameter controlling the size of segments: scale = 100 .

1. Threshold to merge small segments : size_threshold = 50. Number of segments: 621

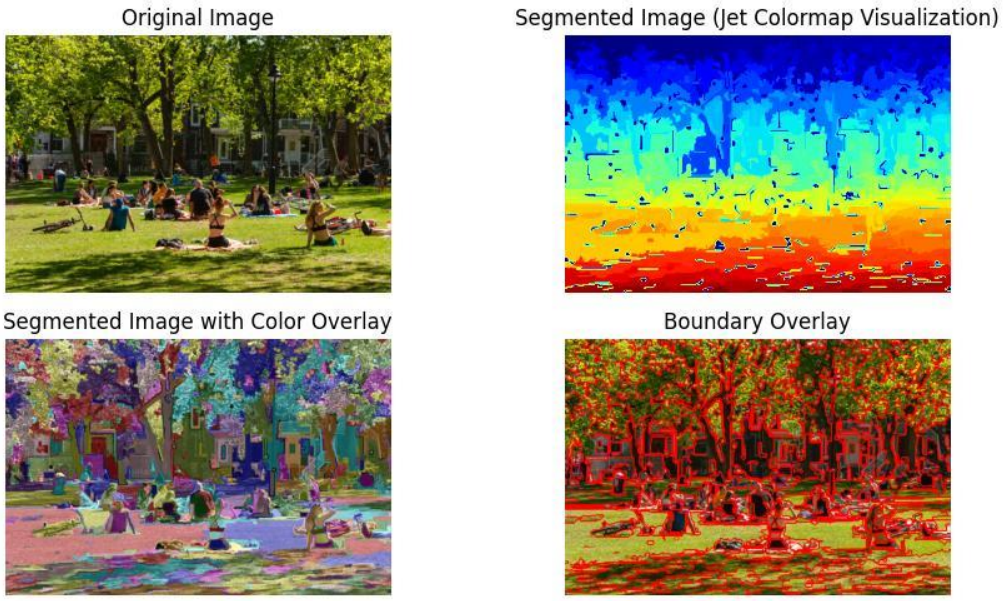

Fig3.1: Output obtained with scale=100 & size_threshold = 50

2. Threshold to merge small segments : size_threshold = 200. Number of segments: 252

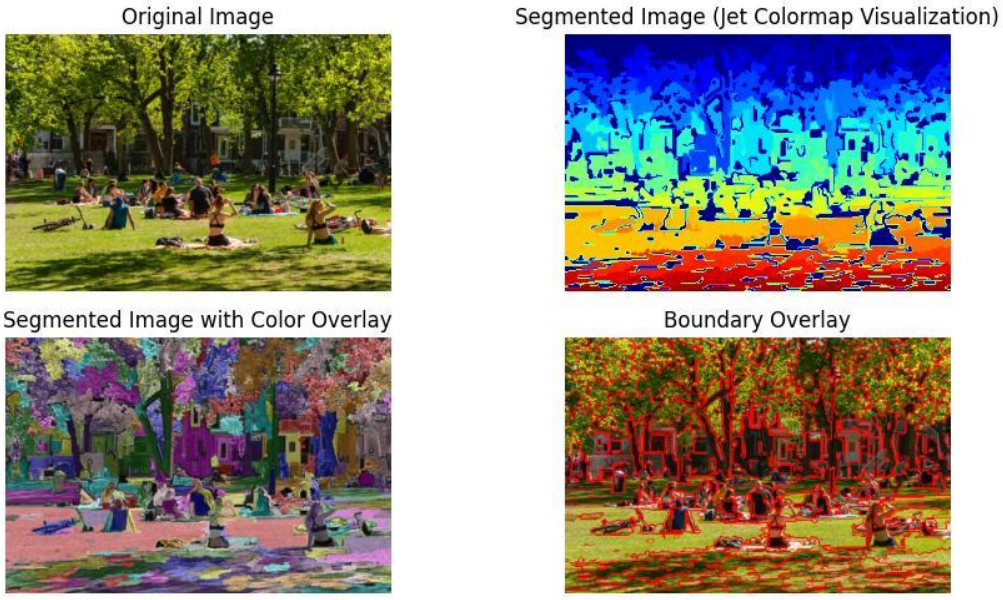

Fig3.2: Output obtained with scale=100 & size_threshold = 200

The scale parameter was set at a constant value of 100. The size_threshold was varied, which basically

determines the minimum size of a segment. Furthermore, with a size_threshold of 50, smaller segments are merged together. This configuration is ideal for capturing broader regions or objects in the image.
In contrast, the second case with a size_threshold of 200 allows for the preservation of larger segments. Although the scale value remains the same at 100, the increased size_threshold prevents smaller segments from being merged, resulting in a larger number of smaller segments. This setting is advantageous for preserving finer details and smaller objects in the image, enhancing the level of granularity in the segmentation.

Constant Parameter: Sigma =1.5, Threshold to merge small segments : size_threshold = 200.

3. Parameter controlling the size of segments: scale = 100. Number of segments: 252

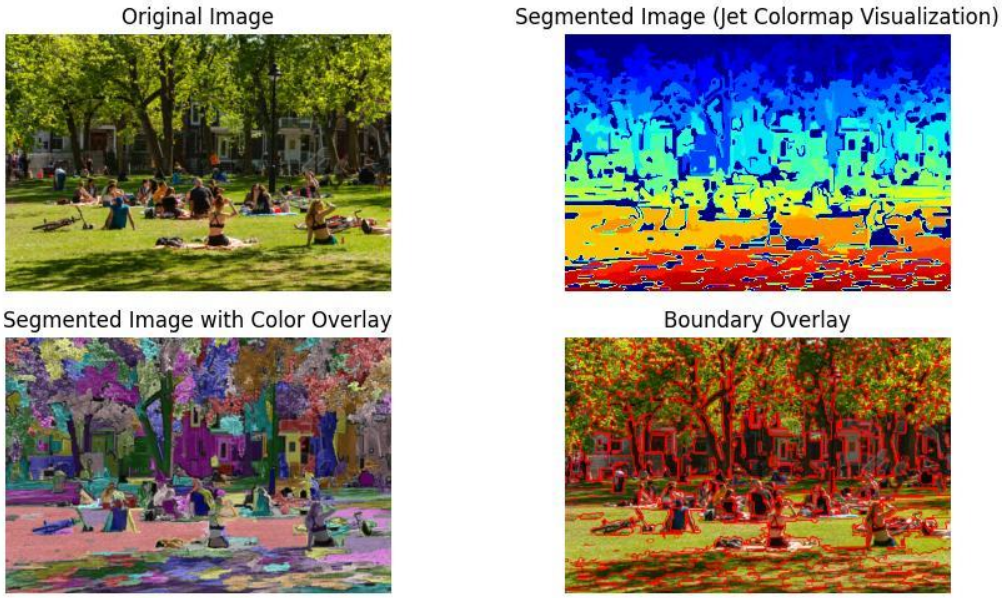

Fig3.3: Output obtained with scale=100 & size_threshold = 200

4. Parameter controlling the size of segments: scale = 400. Number of segments:

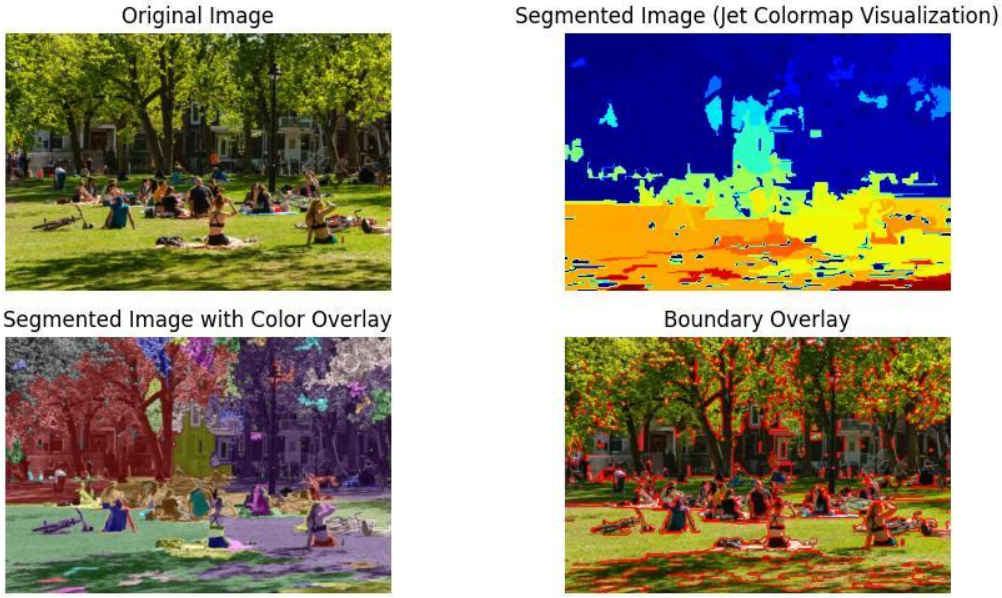

Fig3.4: Output obtained with scale=400 & size_threshold = 200

In the first case, scale is set to 100, and size_threshold is set to 200. In the second case, scale is increased to 400, while size_threshold remains the same at 200. As the scale is intially set to 100, relatively smaller segments are generated, capturing finer details and smaller objects within the image. The size_threshold of 200 ensures that only segments larger than this threshold are retained, effectively merging smaller segments. This configuration is well-suited for applications where a balance between capturing details and merging smaller segments is desired.

On the other hand, the second case utilizes a higher scale value of 400. This leads to the generation of larger segments, encompassing more substantial regions and objects in the image. However, since the size_threshold remains at 200, smaller segments are still merged if their area is below the threshold. Consequently, the resulting segmentation preserves larger structures while also incorporating some level of merging to avoid excessive fragmentation.

Comparing the two cases, the first case with a lower scale value of 100 produces finer-grained segmentation, capturing intricate details and smaller objects. In contrast, the second case with a higher scale of 400 generates larger segments, offering a more holistic view of the image's content while maintaining a certain level of consolidation through the size_threshold parameter.

Finally we vary the sigma parameter keeping the other 2 parameters constant.

Constant Parameter: Parameter controlling the size of segments: scale = 100, Threshold to merge small segments : size_threshold = 200.

5. Sigma = 1.5, No of segments generated: 252

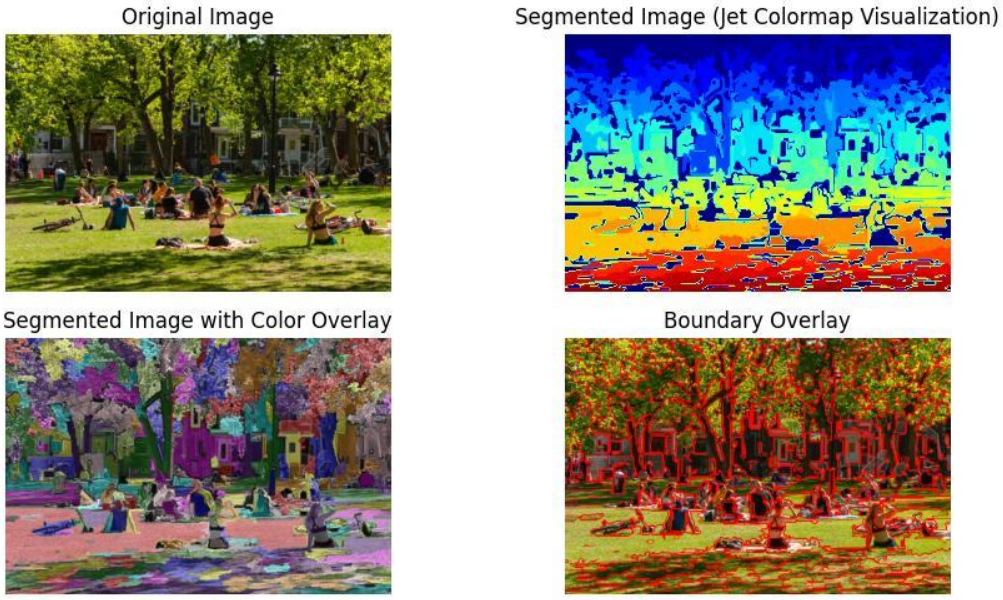

Fig3.5: Output obtained with Sigma=1.5 and other parameters constant.

6. Sigma = 1.0, No of segments generated: 289

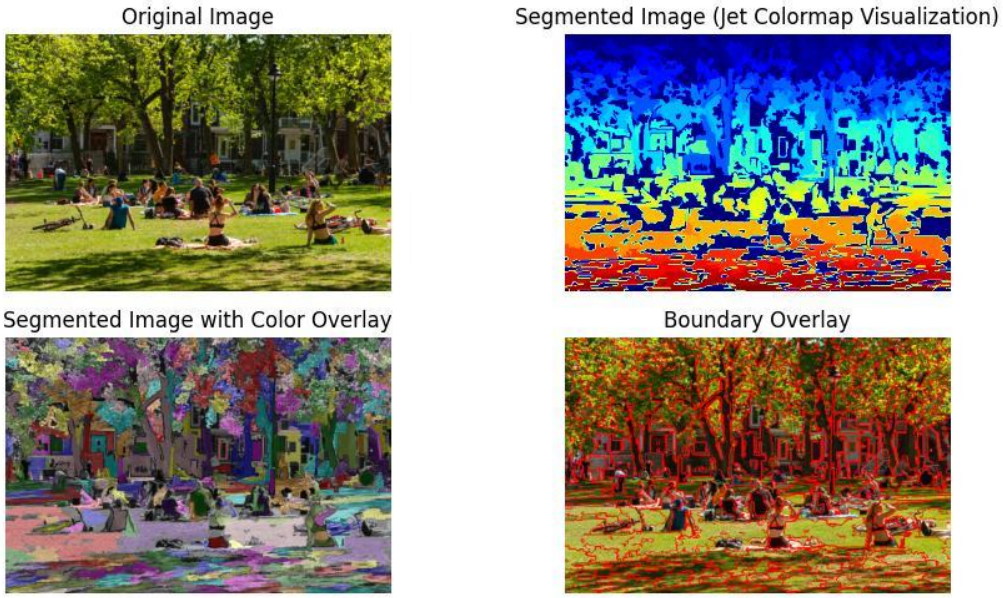

Fig3.6: Output obtained with Sigma=1.0 and other parameters constant.

7. Sigma = 0.5, No of segments generated: 283

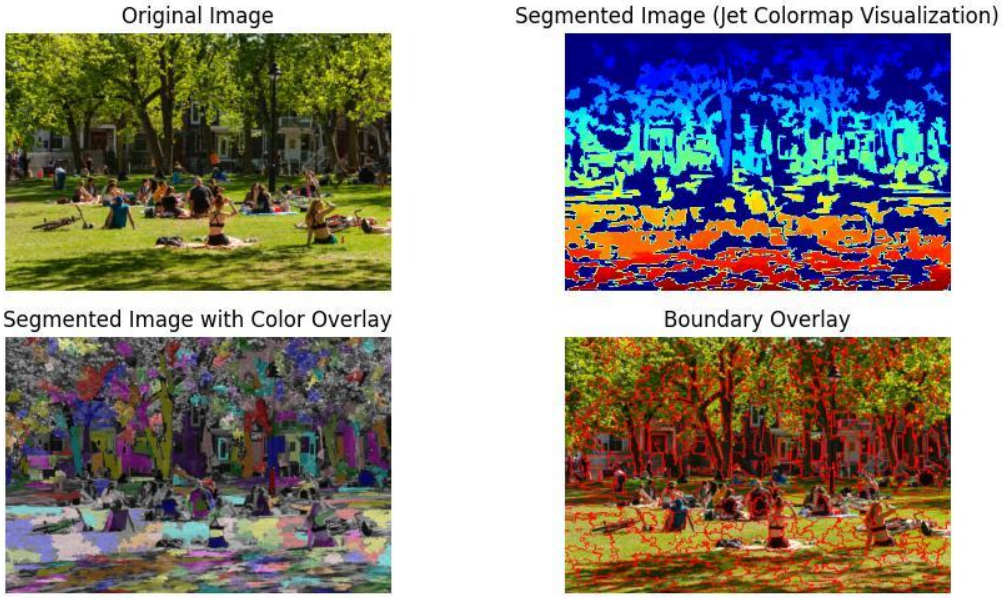

Fig3.7: Output obtained with Sigma=0.5 and other parameters constant.

From the results, it can be observed that adjusting the sigma parameter in the Felzenszwalb algorithm influences the level of detail in the segmentation. By decreasing the sigma value from 1.5 to 1.0, the number of segments increased from 252 to 289, indicating a more fine-grained segmentation with smaller regions being detected. However, further decreasing the sigma to 0.5 resulted in a slight decrease in the number of segments to 283, suggesting that excessively reducing sigma may cause some smaller segments to merge. This analysis suggests that adjusting the sigma parameter allows control over the level of detail in the segmentation output, affecting the number and size of the segments obtained.

CONCLUSION

The Felzenszwalb segmentation, which is a graph based segmentation algorithm effectively divided the input image into distinct regions based on the specified parameters. By merging small segments using a size threshold, the algorithm ensured that only meaningful and significant regions were retained.

The segmented image with the jet colormap visualization provided a clear differentiation of the various regions in the image. This visualization can be particularly useful for tasks such as object detection or region-based analysis.

The color overlay on the segmented image allowed for a comprehensive understanding of the segmented regions in the context of the original image. It provided a visual representation of the segmentation results, facilitating interpretation and analysis.

The boundary overlay highlighted the boundaries between different segments, aiding in visualizing the separation between regions.

REFERENCES

1. http://gnjatovic.info/imageprocessing/
2. Input Image was obtained from the following URL: https://media.istockphoto.com/id/1248713094/photo/people-gathering-during-coronavirus-pandemic-in-laurier-park.jpg?s=612x612&w=0&k=20&c=tPToQsM_vwQIh1-DB5XHNtDezCGLx8KQKv3f-WLdyew=
3. Flowchart was constructed using lucid chart tool from the internet. https://lucid.app/lucidchart/a85a902c-8d5f-4077-a820-18d83e405dc9/edit?invitationId=inv_3f59da29-0ab9-4534-8c19-adf7b3694583&page=0_0#
4. Scikit documentation:https://scikit-image.org/docs/stable/auto_examples/segmentation/plot_segmentations.html
5. Abdulateef, Salwa & Salman, Mohanad. (2021). A Comprehensive Review of Image Segmentation Techniques. Iraqi Journal for Electrical and Electronic Engineering. 17. 166-175. 10.37917/ijeee.17.2.18.
6. Felzenszwalb, Pedro & Huttenlocher, Daniel. (2004). Efficient Graph-Based Image Segmentation. International Journal of Computer Vision. 59. 167-181. 10.1023/B%3AVISI.0000022288.19776.77.
7. https://chat.openai.com/c/0e9ad100-b851-4144-ae77-b511736bdcb1
8. https://en.wikipedia.org/wiki/Image_segmentation#Classes_of_segmentation_techniques
9. https://chat.openai.com/
    - Deriving syntax for implementing 'Felzenszwalb' algorithm-Line:26,29-33.
    - Concept of Boundary overlay for visualization- Line:36.
    - Concept of Color overlay for visualization- Line:39.
    - Using len() to display segment count-Lines:42-43.
    - Apart from the above-mentioned lines, ChatGPT was mainly used to debug errors like installing missing libraries and understanding importing functions.

    **** The above lines of code were extracted from ChatGPT to adhere to syntax, however the methods used were determined after due diligence.**

¶¶¶ Thank You for your Patience ¶¶¶