

Image Segmentation Optimization Using Swarm Intelligence Algorithms

Master's Thesis

Weerathunga Arachchige Haritha Weerathunga

Supervised By

Professor Xiao-Zhi Gao



UNIVERSITY OF
EASTERN FINLAND

School of Computing

Computer Science

February 2024

Abstract

Image segmentation is one of the most vital areas in the computer science field. This takes a huge importance in computer vision and image processing. Basically, this involves segmenting an image into meaningful partitions or objects based on their pixel intensity, visual color, texture, or any other relevant characteristic. In today's world Image Segmentation has numerous applications. This includes medical imaging, satellite image-based segmentations in forestry, ocean mapping and many more.

Particle Swarm Optimization (PSO) is an optimization technique that draws inspiration from the collective behavior which is observed in bird flocks or schools of fish. In this thesis it is explored by each particle being corresponding to a possible segmentation scenario, navigating through various pixel or region combinations to find the optimal segmentation. Another Swarm Intelligence algorithm which is explored in this thesis is Grey Wolf Optimization Algorithm. This is based on the hunting behavior of wolves. This mimics the way wolves lead their pack in search of a prey. Each solution is a wolf, and the population is categorized as alpha, beta etc. They explore solution space to arrive at the best space to converge in this context it is the best segmentation outcome. The last Swarm Intelligence algorithm we explore in this thesis is the Firefly algorithm. This was inspired by the flashing behavior of fireflies. This mainly uses bioluminescent communication among fireflies to form solutions to optimization-based problems. When relating to the image segmentation each firefly signifies a possible segmentation solution while the entire swarm is moving in the image pixel search space. The movement is influenced by the brightness in image pixels and will iteratively find the optimal segmentation of the image.

This thesis represents an initial effort in how to use these swarm intelligence algorithms in image segmentation optimization, what are the most effective less time and resource consuming algorithms when it comes to image segmenting and what gives the best outcome when comparing the output of these three algorithms. All the implementations are done in Python and using computer vision libraries.

Keywords: Swarm Intelligence Algorithm, Grey Wolf Optimization Algorithm, Firefly Algorithm, Particle Swarm Optimization Algorithm, Image Segmentation

Acknowledgement

This was completed as a partial fulfillment of the Masters in Computer Science Degree which is offered by the University of Eastern Finland's School of Computing. I would like to express my special thanks of gratitude to my supervisor Professor Xiao-Zhi Gao who gave me the opportunity to do this thesis with him and always guided me in the correct direction to make this work a success. Doing this thesis with him gave me a lot of in depth knowledge in to the area of Swarm Intelligence and how image segmentation can be benefited from those kinds of algorithms. Also thanking University of Eastern Finland for nurturing me with the latest computer science knowledge.

List of abbreviations

SI – Swarm Intelligence

CV – Computer Vision

PSO – Particle Swarm Optimization

GWO – Grey Wolf Optimization

FFO – Firefly Optimization

List of Tables

Table 1: Comparison of Image Segmentation Techniques	14
Table 2: Visual Difference Between Image Segmentation Using Different Swarm Intelligence Algorithms. (Q values are also given).....	46
Table 3: Segmentation Time Efficiency	50

List of figures

Figure 1 : PSO Image Segmentation Implementation	16
Figure 2: Image Loading Code Snippet	17
Figure 3: Flatten Image Code Snippet	17
Figure 4: Objective Function Code Snippet.....	18
Figure 5: PSO Optimization Code Snippet.....	18
Figure 6: PSO Optimization Loop	19
Figure 7: Firefly Algorithm Image Segmentation Implementation	21
Figure 8: Firefly Algorithm Initialization Code Snippet.....	22
Figure 9: Perform Image Segmentation Code Snippet	23
Figure 10: Grey Wolf Optimization Pseudo Code	24
Figure 11: Grey Wolf Optimization Algorithm Code Snippet	26
Figure 12: Alpha, Beta, Delta position handling code snippet	27
Figure 13: Grey Wolf Optimization Algorithm Image Segmentation Implementation	29
Figure 14: Image Load and Flatten Code Snippet	30
Figure 15: Initializing the GWO	31
Figure 16: GWO Optimization Loop Code Snippet	32
Figure 17: Q Function Formula	34
Figure 18: Code Snippet for Calculating the Color Error.....	35
Figure 19: Q Function Code Snippet	36
Figure 20: PSO Segmentation 1	38
Figure 21: PSO Segmentation 2.....	38
Figure 22: PSO Segmentation 3.....	39
Figure 23: PSO Segmentation 4.....	39
Figure 24: PSO Segmentation 5.....	40
Figure 25: PSO Segmentation 6.....	40
Figure 26: Firefly Segmentation 1	41
Figure 27: Firefly Segmentation 2	41
Figure 28: Firefly Segmentation 3	42
Figure 29: Firefly Segmentation 4	42
Figure 30: Firefly Segmentation 5	43
Figure 31: Firefly Segmentation 6	43
Figure 32: Grey Wolf Optimization Segmentation 1	44
Figure 33: Grey Wolf Optimization Segmentation 2	44
Figure 34: Grey Wolf Optimization Segmentation 3	45
Figure 35: Grey Wolf Optimization Segmentation 4	45
Figure 36: Grey Wolf Optimization Segmentation 5	45
Figure 37: Grey Wolf Optimization 6	46
Figure 38: PSO Q Value Experiment 1	47
Figure 39: Firefly Q Value Experiment 2	47
Figure 40: Q Value Comparison	48
Figure 41: Q Value Comparison 2	49

Figure 42: Algorithm Image Segmentation Time Comparison..... 51

Table of Contents

1.	Introduction.....	8
1.1	Background.....	8
1.2	Problem Statement.....	8
1.3	Research Questions	9
2.	Literature Review.....	10
3.	Methodology.....	16
3.1	PSO Implementation	16
3.2	Firefly	20
3.3	Grey Wolf Optimization.....	24
3.4	Q Function.....	33
4.	Experiments	38
4.1	Experiments with Particle Swarm Optimization	38
4.2	Experiments with Firefly Algorithm	41
4.3	Experiments with Grey Wolf Optimization Algorithm.....	44
4.4	Experiment Based on Segmentation Evaluation Algorithm.....	47
4.5	Experiment based on the time it takes to segment the image.....	50
5.	Conclusion and Future Work	53

1. Introduction

This portion of the thesis provides the background, research problem statements and research questions regarding this research topic. In computer vision image segmentation is a key problem. Getting to the most optimized version of segmentation helps to identify certain important regions in the image. This is mainly utilized in a lot of key areas starting from medical imaging to autonomous driving.

The main ideology behind this thesis is to explore the possible Swarm Intelligence Algorithms which can be used to enhance image segmentation and conduct experiments to showcase their accuracy and efficiency. The algorithms selected for this work are Particle Swarm Optimization Algorithm, Firefly Algorithm, Gray Wolf Optimization Algorithm. This research work intends to use these algorithms and use them in image segmentation tasks and figure out the most efficient algorithm that can be used to do image segmentation. Also, as an additional effort this research has used a formula that can give the image segmentation a quantitative value and then compare what kind of algorithm is most suitable for image segmentation when using Swarm Intelligence algorithms.

1.1 Background

Image segmentation is one of the most prominent applications in the world of computer vision and artificial intelligence. Segmenting images gives a lot of insights into the image. This is also one of the main ways the computing field contributes to the Medical Computing sector. By segmenting an image, we can identify and isolate different types of components in the picture. This leads to more in-depth and thorough analysis and accurate disease diagnosis which helps the medical personnel to plan treatments and navigation during surgeries.

1.2 Problem Statement

Image segmenting is a vital task in computer vision which enables to partition a given image into meaningful segments or area of interest based on their visual characteristics. There are lot of adverse effects in traditional methods of image segmentation, for example noise sensitivity. Image segmentation does require a lot of computation power. In this research we are also looking into the accuracy and efficiency of the swarm intelligence algorithms on image segmentation.

1.3 Research Questions

- How can we use Swarm Intelligence optimizations to do image segmentation?
 - This thesis will discuss how SI algorithms including PSO, Firefly and Grey Wolf can be adapted for the task of image segmentation.
 - Inspect the mechanisms by which these algorithms improve segmentation outcomes by simulating the natural behaviors observed in swarms, fireflies, and wolves.
 - Provide different kinds of output segmented images by using these three algorithms.
 - Coming up with an implementation that can use these algorithms and get a segmented output.
- How well the Swarm Intelligence algorithms segment images.
 - This research question seeks to evaluate the performance of SI algorithms in the task of image segmentation. This thesis will provide a comprehensive set of experiments which will give an initial idea on “how well” these algorithms perform on different images. And what is the accuracy and the computational power associated with this.
- What are the best swarm intelligence algorithms when it comes to image segmentation in the following areas?
 - Time constraints
 - Most accurate outputs

This thesis consists of five parts including this introduction. A thorough background research was done prior to the research, and it will be presented in the next chapter. The swarm intelligence algorithms used in image segmentation which are “Particle Swarm Optimization Algorithm”, “Firefly Algorithm”, “Grey Wolf Optimization Algorithm” are explained in detail in the third chapter. In this third chapter the Q function which was used to evaluate the accuracy of the segmentation is also discussed in detail. The experiments done and the segmented outputs are given in the fourth section of the thesis with some segmented outputs and graphs. Future improvements suggestions and directions are given with the conclusion in the last chapter. In the end cited papers and resources are given.

2. Literature Review

There are many traditional image segmentations which are mentioned in literature. One of the most prominent papers in literature “A Comprehensive Review of Image Segmentation Techniques” [1] which was done by Salwa Khalid and Mohanand Dawood shows the drawbacks of traditional image segmentations. The paper has used main techniques like edge based (boundary based), region-based segmentation and hybrid-based image segmentation. These kinds of techniques have their own set of disadvantages.

Edge based segmentation focuses on identifying sudden changes in intensity in an image. Techniques like Sobel, Canny and Prewitt operators are commonly used. These methods detect edges by looking for points in an image at which the brightness changes sharply. They are useful when you have well defined object boundaries. In boundary-based segmentations it does not function well with images that have a lot of edges or if the images are not clear, and it is not suitable for images which are very noisy. This technique can also fail in low contrast situations and might have difficulties in blurred edges.

When it comes to Region Based Image segmentation techniques, it involves partitioning an image into regions that are similar according to a set of predefined criteria. This involves regions growing, splitting, and merging. These methods are more suitable for images where the intensity variation within objects is low and the contrast between objects and background is high. They are computationally intensive and may not perform well in high complexity images. Region based techniques can also result in over segmentation.

Application of Swarm Intelligence Optimization Algorithms in Image Processing: A Comprehensive Review of Analysis, Synthesis, and Optimization suggests the ant colony algorithm improves the problems of traditional image segmentation algorithms such as low accuracy and long segmentation times [2]. The paper has introduced the best way to do image segmentation is introducing ant colony algorithm and a clustering algorithm together.

Edge Detection using Guided Image Filtering and Enhanced Ant Colony Optimization paper suggests that including a clustering algorithm, guided filtering methods does not increase the

computational complexity of the algorithm thus it increases the accuracy of segmenting images using edge detection [3].

Ant colony optimization with horizontal and vertical crossover search demonstrates the replacement of the search method in two-dimensional maximum entropy segmentation with ACO. This approach significantly reduces segmentation time while ensuring real time performance. Addressing the computational intensity and poor real time performance issues of traditional image segmentation methods.

Ant Colony Stream Clustering: A Fast Density Clustering Algorithm for Dynamic Data Streams this paper suggests the introduction of Ant Colony Optimization into the density peak clustering algorithm for optimal cluster center in medical image segmentation shows the potential for ACO in enhancing the segmentation precision in the specialized fields [4].

Chaotic random spare ant colony optimization for multi-threshold image segmentation explores the combination of threshold segmentation methods with ACO, focusing on updating ant pheromone concentration and improving initial cluster centers. This strategy leads to reduced segmentation time and higher accuracy, illustrating the optimization capabilities of ACO in image segmentation [5].

Theoretical and experimental evaluation of Hybrid ACO-k-means Image segmentation algorithm for MRI Images using Drift Analysis this presents the combination of Swarm Intelligence algorithm Ant Colony and K means together can improve the segmentation results [6].

Some papers suggest that using K-Means as a clustering mechanism is beneficial when extracting larger regions of the images [7].

In this research the application of Particle Swarm Optimization, Firefly Algorithm and Grey Wolf Optimization algorithm is discussed in image segmentation. Particle Swarm Optimization is one of the most known Swarm Intelligence techniques which was introduced by James Kennedy and Russell Eberhart in 1995. In this a potential solution for the optimization problem is being represented by each particle. And it navigates the search space under the influence of both personal best and global best solutions. This mechanism has been successfully applied to

image segmentation by adjusting particles to optimize the fitness function related to segmentation quality. This can be any quality where the pixel qualities can be uniquely evaluated. The result of each particle position can represent a segmentation threshold or parameters of a segmentation model.

The other algorithm which is discussed in this thesis is the firefly algorithm. This was proposed by X.S. Yang in 2009. This is inspired by the bioluminescence and mating behavior of fireflies. In this context each firefly can represent a possible solution or candidate solution with its brightness correlating to the fitness of the solution. This can be used for this research by taking the brightness of a firefly can correspond to the effectiveness of a segmentation threshold or parameter set. The dynamic adaptability of attractiveness among fireflies provides the discovery of optimized segmentation strategies by exploiting the search space dynamically.

The last algorithm discussed in this thesis is the Grey Wolf Optimization algorithm [8,9]. This had demonstrated its capability to fine tune image segmentation algorithms for more accurate and effective outcomes. By optimizing the parameters of segmentation techniques such as region growing, clustering or the watershed method, GWO can identify optimal threshold values that precisely differentiate between the foreground and background of images. This optimization process is particularly valuable in medical imaging. Where GWO has been leveraged to refine the segmentation of organs or tumors, thereby providing critical support in the diagnosis and treatment planning processes. Ramezani, F., and Zolfaghari, S. (2019) highlight GWO's application in medical imagery to not only improve the clarity of organ and tumor segmentation but also to facilitate more informed decision-making in both diagnosis and therapeutic strategies.

From the background literature survey conducted Swarm Intelligence Algorithm optimizations offer a significant improvement and advantage over traditional image segmentation methods. The unique abilities like adaptivity, decentralized and self-organizing of swarm intelligence algorithms pave the way for more efficient exploration and exploitation of the solution space leading to faster and more accurate image segmentation.

Based on the above comprehensive review of various image segmentation techniques and the exploration of Swarm Intelligence Algorithm optimizations it is evident that the field of image

segmentation has evolved tremendously. When using noisy data most classical methodologies like edge-based segmentation and region-based segmentation does not perform well in terms of accuracy, efficiency and adaptability. The introduction of algorithms like Particle Swarm Optimization, Firefly Algorithm and Grey Wolf Optimization plays a vital role in the paradigm shift towards leveraging the collective behavior and adaptability of swarm intelligence for improving image segmentation outcomes.

To effectively summarize and classify the discussed image segmentation techniques we can create a detailed table that contrasts traditional methods with Swarm Intelligence algorithms. The following table will highlight the vital attributes such as the underlying principles, computational complexity, ideal use cases and most notable strengths and weaknesses of the particular method.

Table 1: Comparison of Image Segmentation Techniques

Aspect	Traditional Image Segmentation	Swarm Intelligence Based Image Segmentation
Technique/Algorithm	Edge based segmentation Region based segmentation	Ant Colony Optimization, Particle Swarm Optimization, Firefly Algorithm, Grey Wolf Optimization.
Underlying Principle	Edge based: Detects edges through gradient changes. Region based: Partitions image into similar regions	Ant Colony: Foraging behavior of ants. Particle Swarm: Social behavior of flocking species. Firefly: Bioluminescence and mating behavior. Grey Wolf: Mimic grey wolves' leadership
Computational Complexity	Edge based: Low to medium. Region based: Medium to high	Generally medium, Grey Wolf Optimization: Medium to high
Ideal Use Cases	Edge based: Objects with well- defined boundaries. Region based: Uniform intensity objects	Ant Colony: Complex or noisy images, medical images Particle Swarm: Evolving images Firefly: Images requiring dynamic adaptability Grey Wolf: Precise image segmentation, medical imaging
Strengths	Edge based: Good for clear distinct edges, simple implementation [11] Region based: Very effective with clear boundaries/pixel contrast between objects	Ant Colony: Good accuracy and time reduction. Particle Swarm: Adaptability to change, complex fitness functions [10,11] Firefly: Multi modal optimization Grey Wolf: High accuracy

Weaknesses	<p>Edge based: Poor functionality with noisy and blurry images, cannot cope with low contrast</p> <p>Region based: Computationally very expensive</p>	<p>Ant Colony: Parameter tuning required</p> <p>Particle Swarm: Local optima convergence</p> <p>Firefly: Computationally expensive</p> <p>Grey Wolf: Require high domain knowledge to tune parameters</p>
------------	---	---

3. Methodology

3.1 PSO Implementation

In the context of image segmentation PSO is used to partition an image into segments, which are represented by clusters of pixels with similar attributes. The PSO algorithm simulates a swarm of particles moving through the solution space, where each particle represents a potential solution to the clustering problem.

In the flow chart the process begins by loading the input image and flattening it into a one-dimensional array of pixels, which is essential for subsequent optimization using PSO[12-14]. The flattened pixel array is then subjected to PSO. After that the centroids obtained from the algorithm is inserted into the K Means Clustering algorithm. And segmented regions are colored accordingly. Detailed step by step information is given below the figure.

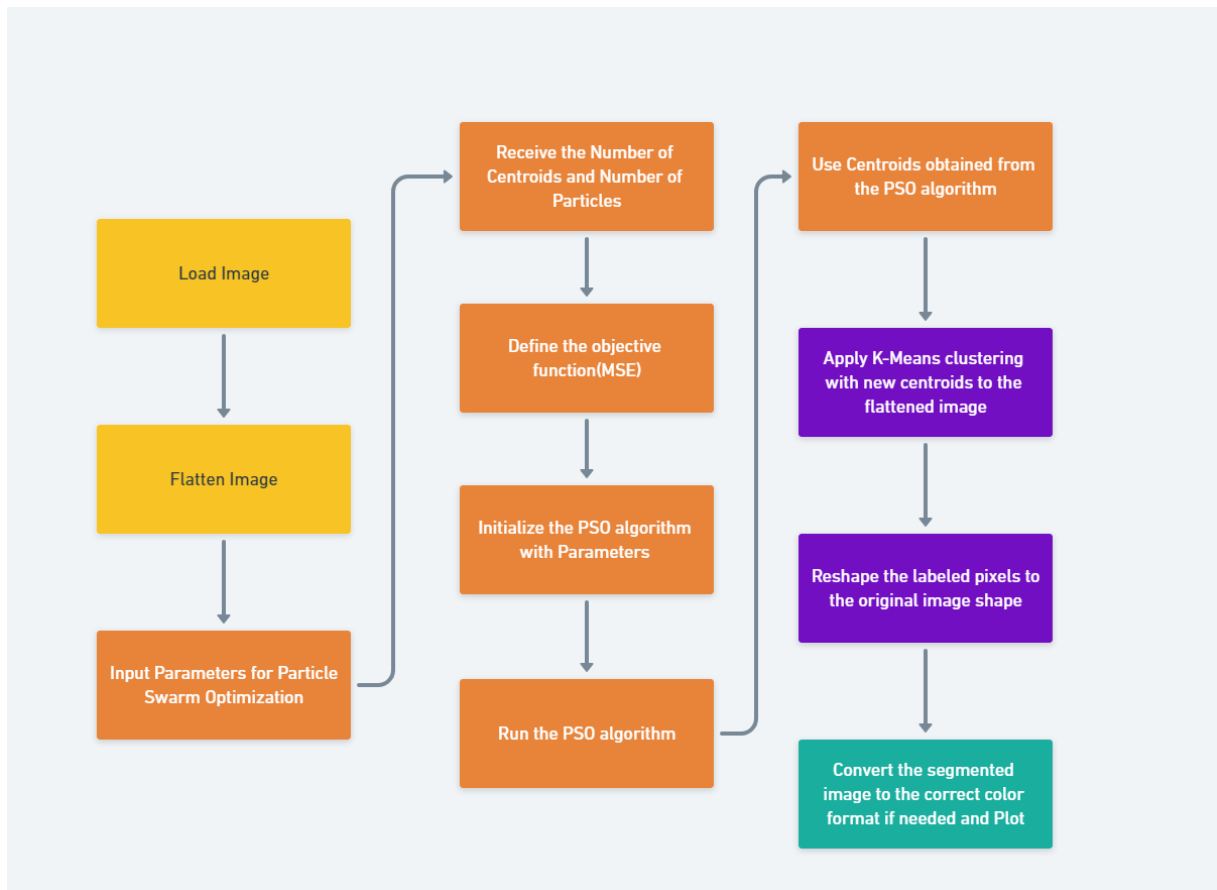


Figure 1 : PSO Image Segmentation Implementation

- **Image Loading and Preprocessing**

```
def load_image(image_path, color=True):  
    ... if color:  
    ...     image = cv2.imread(image_path)  
    ...     image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  
    ... else:  
    ...     image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)  
    ... return image
```

○

Figure 2: Image Loading Code Snippet

- The input image is loaded using the load_image function. If the color flag is set to true, the image is read in color mode and then converted from BGR(Blue, Green, Red) to RGB(Red, Green, Blue) to align with the color representation used in most image processing libraries

```
def flatten_image(image):  
    ... if len(image.shape) == 3:  
    ...     pixels = image.reshape(-1, 3)  
    ... else:  
    ...     pixels = image.reshape(-1, 1)  
    ... return pixels
```

○

Figure 3: Flatten Image Code Snippet

- The flatten_image function then flattens the image into a one-dimensional array of pixel values. For color images each pixel is represented by three values (RGB) while for grey scale images each pixel is represented by a single intensity value.

- Initialization of PSO

```
def objective_function(positions, pixels):  
    # Reshape the positions to cluster centroids  
    if len(pixels[0]) == 3:  
        centroids = positions.reshape(-1, 3)  
    else:  
        centroids = positions.reshape(-1, 1)  
    # Assign each pixel to the closest centroid  
    labels = KMeans(n_clusters=len(centroids), init=centroids,  
                   n_init=1).fit_predict(pixels)  
    # Calculate the mean squared error between the original pixels and the centroids  
    mse = np.mean((pixels - centroids[labels])**2)  
    return mse
```

○

Figure 4: Objective Function Code Snippet

- The flattened array of pixels serves as the input to the PSO algorithm. An objective function, **objective_function**, is defined to evaluate the mean squared error (MSE) between the original pixels and their assigned cluster centroids. This function is critical for guiding the PSO towards optimal segmentation.

```
def perform_pso_optimization(pixels, num_centroids, num_particles):  
    lower_bound = np.zeros(pixels.shape[1] * num_centroids)  
    upper_bound = np.ones(pixels.shape[1] * num_centroids) * 255  
    result, _ = pso(objective_function, lower_bound,  
                   upper_bound, args=(pixels, ), swarmsize=num_particles, maxiter=200, debug=True)  
    # Reshape the optimized result to get the final centroids  
    if pixels.shape[1] == 3:  
        centroids = result.reshape(-1, 3)  
    else:  
        centroids = result.reshape(-1, 1)  
    return centroids
```

○

Figure 5: PSO Optimization Code Snippet

- The PSO algorithm is initialized with a population of particles, each representing a set of potential cluster centroids for the image segments.

- **PSO Optimization Loop**

```
def perform_image_segmentation(image_path, num_centroids, num_particles, color=True):
    image = load_image(image_path, color)
    pixels = flatten_image(image)
    centroids = perform_pso_optimization(pixels, num_centroids, num_particles)
    labels = KMeans(n_clusters=len(centroids), init=centroids,
                   n_init=1).fit_predict(pixels)
    if len(image.shape) == 3:
        segmented_image = centroids[labels].reshape(image.shape)
        segmented_image = cv2.cvtColor(
            segmented_image.astype(np.uint8), cv2.COLOR_RGB2BGR)
    else:
        segmented_image = centroids[labels].reshape(image.shape)
        segmented_image = segmented_image.astype(np.uint8)

    # Display the original and segmented images
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(image)
    plt.title('Original Image')

    plt.subplot(1, 2, 2)
    plt.imshow(segmented_image)
    plt.title('Segmented Image')

    plt.show()
```

Figure 6: PSO Optimization Loop

- Within the **perform_pso_optimization** function, the PSO algorithm iteratively updates the positions of the particles (i.e., the candidate centroids) to minimize the objective function. Each particle's position is adjusted based on its own experience and the experience of neighboring particles, emulating a social sharing of information.
- The algorithm uses the positions that result in the lowest objective function value (i.e., MSE) as guides. The search space is bounded to ensure that the centroids correspond to valid pixel values (ranging from 0 to 255 for each color channel)
- **Segmentation and Reconstruction**
 - Once the PSO algorithm converges to a solution, the final positions of the particles represent the centroids of the clusters that define the image segments.

- The **KMeans** clustering algorithm is then applied with the optimized centroids to partition the pixels into segments, where each pixel is assigned to the nearest centroid.
- The segmented image is reconstructed by replacing each pixel with the value of its corresponding centroid. For color images, the RGB values are reassigned accordingly, and for grayscale images, the intensity values are reassigned.
- **Visualization and Comparison**
 - To visualize the effectiveness of the PSO-based segmentation, the original and segmented images are displayed side-by-side using Matplotlib. This allows for a qualitative comparison between the unsegmented and segmented images.

In this implementation, PSO is used to determine the optimal cluster centroids that minimize intra-cluster variance, which corresponds to the color differences within each segment. This optimization problem is non-trivial due to the high dimensionality of the solution space and the complexity of natural images. PSO offers a heuristic approach that is capable of navigating the solution space more effectively than traditional gradient-based optimization methods, which makes it particularly suitable for the segmentation task where the global optimum is difficult to find analytically.

3.2 Firefly

The Firefly Algorithm is a nature-inspired, metaheuristic optimization algorithm that mimics the behavior of fireflies, where the primary purpose of their luminescence is to act as a signal system to attract other fireflies[15]. In the domain of image segmentation, the Firefly Algorithm is utilized to optimize the clustering of pixels such that the resulting segments exhibit homogeneity in color or intensity.

The following flowchart describes the methodology taken in this thesis to use Firefly Algorithm in image segmentation.

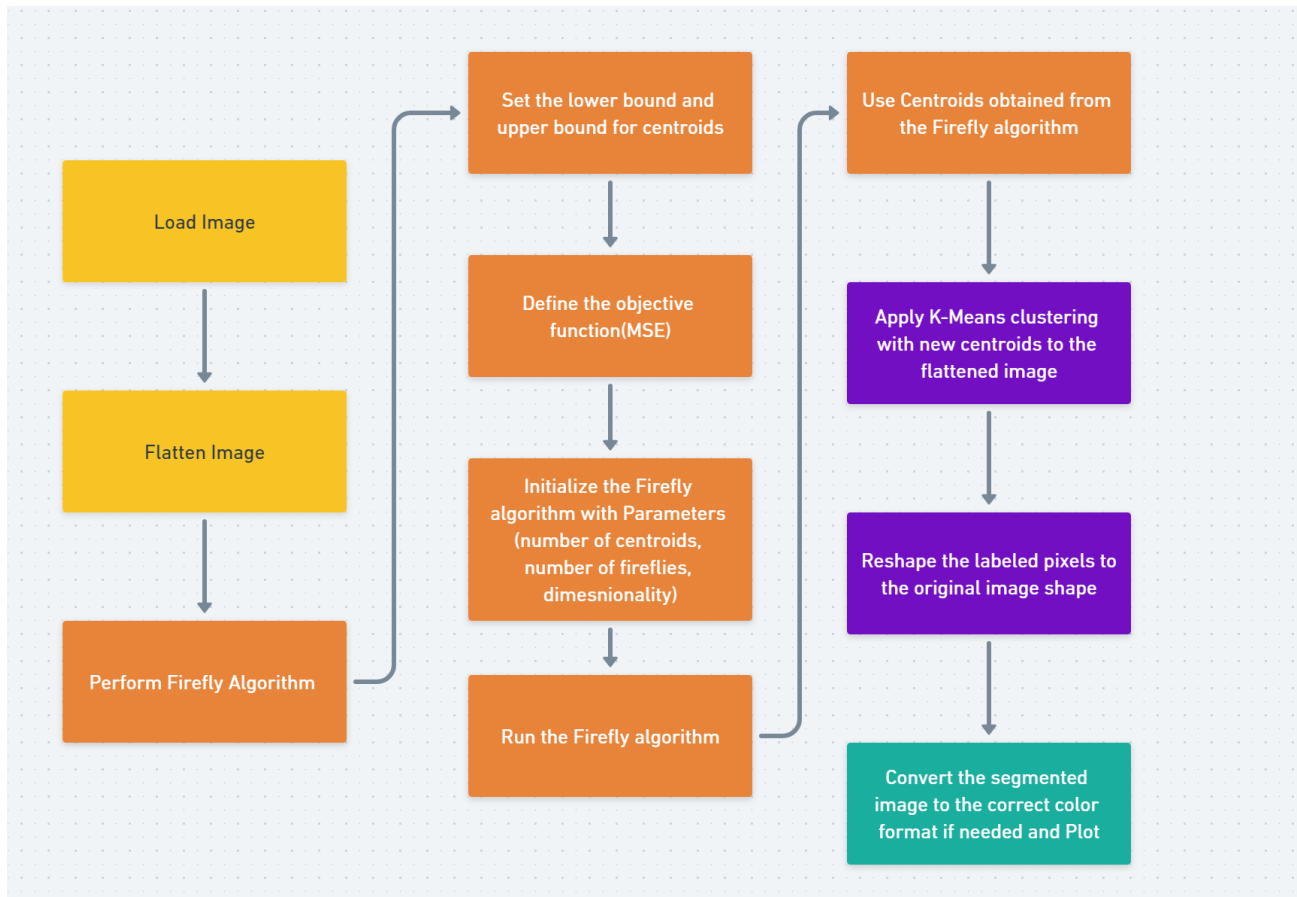


Figure 7: Firefly Algorithm Image Segmentation Implementation

- **Image Loading and Preprocessing:**

- The input image is loaded with the **load_image** function. If the image is to be processed in color, it is read in color mode and the color space is converted from BGR to RGB. For grayscale images, the color flag is set to false.
- The **flatten_image** function flattens the multidimensional image array into a one-dimensional array of pixel values, preparing it for the optimization process. This is essential as the algorithm processes the pixels as points in a multidimensional space.

- **Initialization of Firefly Algorithm:**

```
def objective_function(centroids, pixels):
    labels = KMeans(n_clusters=len(centroids), init=centroids,
                   n_init=1).fit_predict(pixels)
    mse = np.mean((pixels - centroids[labels])**2)
    return mse

def perform_firefly_algorithm_optimization(pixels, num_centroids, num_fireflies):
    lower_bound = np.zeros(pixels.shape[1] * num_centroids)
    upper_bound = np.ones(pixels.shape[1] * num_centroids) * 255
    ....
    def objective(centroids):
        return objective_function(centroids.reshape(-1, pixels.shape[1]), pixels)
    ....
    firefly_algorithm = FireflyAlgorithm()
    best_solution = firefly_algorithm.run(objective, dim=num_centroids * pixels.shape[1],
                                         lb=lower_bound, ub=upper_bound, max_evals=200)
    ....
    if best_solution.size == 1:
        # If the optimization did not converge, provide a default solution
        centroids = np.random.uniform(0, 255, size=(num_centroids, pixels.shape[1]))
    else:
        centroids = best_solution.reshape(-1, pixels.shape[1])
    ....
    return centroids
```

○

Figure 8: Firefly Algorithm Initialization Code Snippet

- The flattened pixel array serves as the domain over which the Firefly Algorithm will operate. The **objective_function** is defined to measure the quality of any given solution (set of cluster centroids) by computing the mean squared error (MSE) between the pixel values and their closest centroid.
- A population of fireflies (solutions) is initialized, each representing a possible set of centroids for the image segments.
- **Firefly Algorithm Optimization Loop:**
 - The **perform_firefly_algorithm_optimization** function employs the Firefly Algorithm to move the fireflies through the solution space. Each firefly's brightness is proportional to the objective function's value (lower MSE means higher brightness).
 - Fireflies are attracted to brighter ones, and thus, they move towards better solutions, updating their positions based on the relative brightness. The search space is constrained to valid pixel values, ensuring centroids correspond to actual colors or intensities in the image.

- **Segmentation and Reconstruction:**

```
def perform_image_segmentation(image_path, num_centroids, num_fireflies, color=True):  
    ... image = load_image(image_path, color)  
    ... pixels = flatten_image(image)  
    ... centroids = perform_firefly_algorithm_optimization(pixels, num_centroids, num_fireflies)  
    ... labels = KMeans(n_clusters=len(centroids), init=centroids,  
    ... n_init=1).fit_predict(pixels)  
    ...  
    ... if len(image.shape) == 3:  
    ...     segmented_image = centroids[labels].reshape(image.shape)  
    ...     segmented_image = cv2.cvtColor(  
    ...         segmented_image.astype(np.uint8), cv2.COLOR_RGB2BGR)  
    ... else:  
    ...     segmented_image = centroids[labels].reshape(image.shape)  
    ...     segmented_image = segmented_image.astype(np.uint8)  
  
    ... # Display the original and segmented images  
    ... plt.figure(figsize=(10, 5))  
  
    ... plt.subplot(1, 2, 1)  
    ... plt.imshow(image)  
    ... plt.title('Original Image')  
  
    ... plt.subplot(1, 2, 2)  
    ... plt.imshow(segmented_image)  
    ... plt.title('Segmented Image')  
  
    ... plt.show()
```

○

Figure 9: Perform Image Segmentation Code Snippet

- Upon convergence, the algorithm provides the optimized centroids, which are used by the **KMeans** algorithm to assign each pixel to the nearest centroid, forming the image segments. Since KMeans algorithm gets trapped in local minima we can use this method to optimize it using the firefly algorithm[16].
- The segmented image is reconstructed by mapping each pixel to its cluster centroid. This results in segments of uniform color or intensity, representing different parts of the image.

- **Visualization and Comparison:**

- To evaluate the segmentation, the original and the segmented images are plotted side by side. This visual comparison allows for a qualitative assessment of the segmentation performance.

In this implementation, the Firefly Algorithm is specifically tailored for image segmentation by optimizing the positions of the cluster centroids to minimize the intra-cluster variance, represented by the MSE. The algorithm's bio-inspired mechanisms allow it to efficiently explore the solution space and avoid local minima, making it particularly effective for the high-dimensional optimization characteristic of image segmentation.

3.3 Grey Wolf Optimization

Another Swarm Intelligence Optimization algorithm we used is the Grey Wolf Optimization function. The following pseudo code is taken from the “Grey Wolf Optimizer” paper [8].

```
Initialize the grey wolf population  $X_i$  ( $i = 1, 2, \dots, n$ )
Initialize  $a$ ,  $A$ , and  $C$ 
Calculate the fitness of each search agent
 $X_\alpha$ =the best search agent
 $X_\beta$ =the second best search agent
 $X_\delta$ =the third best search agent
while ( $t < \text{Max number of iterations}$ )
    for each search agent
        Update the position of the current search agent by equation (3.7)
    end for
    Update  $a$ ,  $A$ , and  $C$ 
    Calculate the fitness of all search agents
    Update  $X_\alpha$ ,  $X_\beta$ , and  $X_\delta$ 
     $t=t+1$ 
end while
return  $X_\alpha$ 
```

Figure 10: Grey Wolf Optimization Pseudo Code

Grey Wolf Optimization (GWO) is an optimization algorithm that simulates the leadership hierarchy and hunting mechanism of grey wolves in nature. Applied to image segmentation, GWO seeks to identify clusters of pixels that minimize intra-cluster variance, effectively partitioning the image into meaningful segments based on color or intensity similarities. This can be considered as an efficient meta heuristic technique of optimization [17]. The above code has been created in Python and the main structure of the implementation is given below[18].

Implementation Details of Grey Wolf Optimization Algorithm

```
def GWO(objf,lb,ub,dim,SearchAgents_no,Max_iter):  
  
    # initialize alpha, beta, and delta_pos  
    Alpha_pos=np.zeros(dim)  
    Alpha_score=float("inf")  
  
    Beta_pos=np.zeros(dim)  
    Beta_score=float("inf")  
  
    Delta_pos=np.zeros(dim)  
    Delta_score=float("inf")  
  
    if not isinstance(lb, list):  
        lb = [lb] * dim  
    if not isinstance(ub, list):  
        ub = [ub] * dim  
  
    #Initialize the positions of search agents  
    Positions = np.zeros((SearchAgents_no, dim))  
    for i in range(dim):  
        Positions[:, i] = np.random.uniform(0,1, SearchAgents_no) * (ub[i] - lb[i]) + lb[i]  
  
    Convergence_curve=np.zeros(Max_iter)  
  
    # Loop counter  
    print("GWO is optimizing \"+objf.__name__+"\")  
  
    # Main Loop  
    for l in range(0,Max_iter):  
        for i in range(0,SearchAgents_no):
```

```

for l in range(0,Max_iter):
    for i in range(0,SearchAgents_no):
        # Return back the search agents that go beyond the boundaries of the search space
        for j in range(dim):
            Positions[i,j]=np.clip(Positions[i,j], lb[j], ub[j])

        # Calculate objective function for each search agent
        fitness=objf(Positions[i,:])

        # Update Alpha, Beta, and Delta
        if fitness<Alpha_score :
            Alpha_score=fitness; # Update alpha
            Alpha_pos=Positions[i,:].copy()

        if (fitness>Alpha_score and fitness<Beta_score ):
            Beta_score=fitness # Update beta
            Beta_pos=Positions[i,:].copy()

        if (fitness>Alpha_score and fitness>Beta_score and fitness<Delta_score):
            Delta_score=fitness # Update delta
            Delta_pos=Positions[i,:].copy()

a=2-1*((2)/Max_iter); # a decreases linearly from 2 to 0

# Update the Position of search agents including omegas
for i in range(0,SearchAgents_no):
    for j in range (0,dim):
        r1=random.random() # r1 is a random number in [0,1]
        r2=random.random() # r2 is a random number in [0,1]

```

Figure 11: Grey Wolf Optimization Algorithm Code Snippet

- **Initialization**

- The algorithm starts by defining key parameters such as the lower ('lb') and upper ('ub') bounds of the search space. Meaning lower bound and upper bound. The dimensionality of the problem ('dim'), the number of search agents ('SearchAgents_no') and the maximum number of iterations is defined as "Max_iter".
- This initializes the positions of the search agents (greywolves) within the search space, ensuring they are spread across the space by generating random positions within the defined bounds.

- **Objective Function**

- The fitness of each grey wolf is evaluated using the objective function "objf" passed to the GWO function. This function quantifies how well a given solution addresses the problem, with lower values typically indicating better solutions.

- **Alpha, Beta and Delta positions**

```

# Update the Position of search agents including omegas
for i in range(0,SearchAgents_no):
    for j in range (0,dim):

        r1=random.random() # r1 is a random number in [0,1]
        r2=random.random() # r2 is a random number in [0,1]

        A1=2*a*r1-a; # Equation (3.3)
        C1=2*r2; # Equation (3.4)

        D_alpha=abs(C1*Alpha_pos[j]-Positions[i,j]); # Equation (3.5)-part 1
        X1=Alpha_pos[j]-A1*D_alpha; # Equation (3.6)-part 1

        r1=random.random()
        r2=random.random()

        A2=2*a*r1-a; # Equation (3.3)
        C2=2*r2; # Equation (3.4)

        D_beta=abs(C2*Beta_pos[j]-Positions[i,j]); # Equation (3.5)-part 2
        X2=Beta_pos[j]-A2*D_beta; # Equation (3.6)-part 2

        r1=random.random()
        r2=random.random()

        A3=2*a*r1-a; # Equation (3.3)
        C3=2*r2; # Equation (3.4)

        D_delta=abs(C3*Delta_pos[j]-Positions[i,j]); # Equation (3.5)-part 3
        X3=Delta_pos[j]-A3*D_delta; # Equation (3.5)-part 3

        Positions[i,j]=(X1+X2+X3)/3 # Equation (3.7)
        Convergence_curve[l]=Alpha_score;

    #if (L%1==0):
        #print(['At iteration '+ str(L)+ ' the best fitness is '+ str(Alpha_score)]);

print(Positions.shape)
print("Alpha position=",Alpha_pos);
print("Beta position=" Beta_pos);

```

Figure 12: Alpha, Beta, Delta position handling code snippet

- The algorithm maintains a hierarchy among the grey wolves based on their fitness scores. The best three solutions are identified as the alpha (best), beta (second best) and delta (third best) positions. These roles are essential for guiding the search process, with the rest of the pack following these leaders[19].
- **Main Loop**
 - The main loop of the algorithm iterates over a set number of generations (Max_iter). During each iteration, the algorithm performs several main operations.
 - One is boundary check, this check ensures each wolf's position is adjusted to remains within the defined search space boundaries.
 - The other one is fitness evaluation. This is done to figure out If there has been any improvements regarding the each wolf's position.

- Next one is updating the hierarchy. This is done based on the current fitness scores and then alpha, beta and delta wolves are updated. This ensures that the current hierarchy represents the best current solutions[20].
- **Update Positions**
 - This includes calculating vectors toward the alpha, beta and delta wolves, influenced by randomly generated coefficients that control exploration and exploitation. These positions are influenced by the distances to the alpha, beta, and delta positions with the aim of simulating the process of surrounding prey. This mimics the social hierarchy and hunting behavior of grey wolves in nature.
- **Boundary Constraints**
 - After updating the positions the algorithm checks and enforces boundary constraints to ensure that wolves do not move outside the search space. This is crucial for maintaining the validity of the solutions.
- **End of Main Loop**
 - This loop continues until the stopping criteria is met. This can be either by reaching the maximum number of iterations or achieving a desired level of fitness.
- **Return Best Solution**
 - Once the stopping criterion is met the algorithm returns the positions of the alpha and beta wolves as the best solutions found during this optimization process. The final positions and fitness scores of the alpha, beta and delta wolves are also calculated thus providing insight into the optimization outcome.

Implementation of Image Segmentation Optimization using Grey Wolf Optimization Algorithm

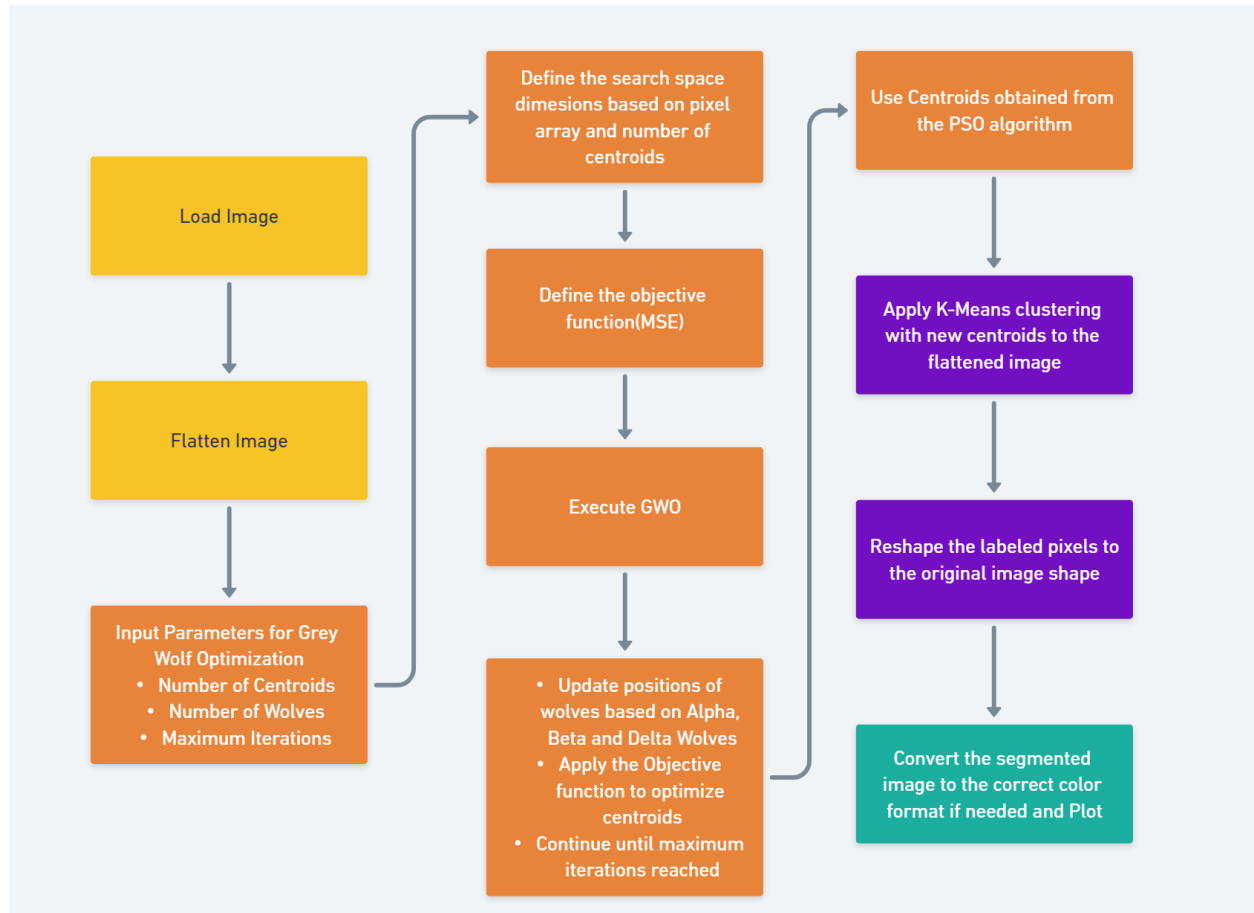


Figure 13: Grey Wolf Optimization Algorithm Image Segmentation Implementation

- **Image Loading and Preprocessing:**

```
def load_image(image_path, color=True):
    if color:
        image = cv2.imread(image_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    else:
        image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    return image

def flatten_image(image):
    if len(image.shape) == 3:
        pixels = image.reshape(-1, 3)
    else:
        pixels = image.reshape(-1, 1)
    return pixels
```

○

Figure 14: Image Load and Flatten Code Snippet

- Initially, the **load_image** function is employed to load the image. If the image is to be processed in color, it is read in color mode and converted from BGR (which is the OpenCV's default color space) to RGB format (which is more standard and suitable for matplotlib operations). Conversely, for a grayscale image, the color flag is set to false, and the image is read as such.
- The **flatten_image** function is then used to transform the 2D image array into a 1D array of pixel values, which is a necessary step for the optimization process to treat the pixels as points in a multi-dimensional search space. Each row represents a pixel, and each column represents a color channel. In this case it is RGB for color images and intensity levels for greyscale images. This flattening is later useful in k means algorithm since it requires a 2D array of pixels as input.

- **Initialization of GWO:**

```
def objective_function(centroids, pixels):
    labels = KMeans(n_clusters=len(centroids), init=centroids,
                    n_init=1).fit_predict(pixels)
    mse = np.mean((pixels - centroids[labels]) ** 2)
    return mse

def perform_gwo_optimization(pixels, num_centroids, num_wolves, max_iter):
    dim = pixels.shape[1] * num_centroids
    lb = 0
    ub = 255

    def objective(centroids):
        return objective_function(centroids.reshape(-1, pixels.shape[1]), pixels)

    alpha_pos, _ = GWO(objective, lb, ub, dim, num_wolves, max_iter)
    centroids = alpha_pos.reshape(-1, pixels.shape[1])
    return centroids
```

-

Figure 15: Initializing the GWO

- The algorithm begins by initializing a pack of grey wolves, where each wolf (search agent) represents a potential solution, i.e., a set of cluster centroids for the image segments.
 - The search space is defined by the lower and upper bounds of pixel values (0 to 255 for each color channel). Positions of the grey wolves are initialized within this space.
- **Objective Function:**
 - This takes the set of centroids and flattened pixel data as inputs and then perform the K Means clustering. Then it calculates the Mean Squared Error (MSE) between the pixels and their assigned centroid pixels. This value of the MSE serves as the objective to minimize. Lower the better, which indicates that the centroids are accurately representing the clusters / segments of the given image.

- **GWO Optimization Loop:**

```
def perform_image_segmentation(image_path, num_centroids, num_wolves, max_iter, color=True):
    image = load_image(image_path, color)
    pixels = flatten_image(image)
    centroids = perform_gwo_optimization(pixels, num_centroids, num_wolves, max_iter)
    labels = KMeans(n_clusters=num_centroids, init=centroids, n_init=1).fit_predict(pixels)

    if len(image.shape) == 3:
        segmented_image = centroids[labels].reshape(image.shape)
        segmented_image = cv2.cvtColor(
            segmented_image.astype(np.uint8), cv2.COLOR_RGB2BGR)
    else:
        segmented_image = centroids[labels].reshape(image.shape)
        segmented_image = segmented_image.astype(np.uint8)

    # Display the original and segmented images
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(image)
    plt.title('Original Image')

    plt.subplot(1, 2, 2)
    plt.imshow(segmented_image)
    plt.title('Segmented Image')

    plt.show()
```

-

Figure 16: GWO Optimization Loop Code Snippet

- The **perform_gwo_optimization** function incorporates the main loop of the GWO, where wolves (search agents) update their positions in the search space under the guidance of the alpha (best solution), beta, and delta wolves, which have the most promising solutions at any given iteration.
- The algorithm employs equations inspired by the social hierarchy and hunting behavior of wolves to update the positions of the pack members, aiming to converge towards optimal solutions that minimize the objective function – in this case, the mean squared error (MSE) between pixel values and their corresponding cluster centroids.
- **Segmentation and Reconstruction:**
 - After convergence, the alpha wolf's position represents the optimized cluster centroids. These centroids are then used by the **KMeans** clustering algorithm to

assign each pixel to the nearest centroid, thus forming distinct segments within the image.

- The segmented image is reconstructed by assigning the centroid values to each pixel, effectively recoloring the image based on the segmentation results.
- **Visualization and Comparison:**
 - For qualitative assessment, the original and segmented images are displayed side by side. This allows for a visual evaluation of the segmentation effectiveness.

The GWO is particularly well-suited for image segmentation because of its ability to efficiently explore and exploit search space, which in this case is the possible configurations of pixel clusters. Its social hierarchy-based movement strategy enables it to adaptively adjust the exploration-exploitation balance, thus effectively avoiding local minima and ensuring convergence towards a global optimum.

In this implementation, GWO is employed to optimize the clustering of pixels such that the resulting image segmentation maximizes the homogeneity within segments and the heterogeneity between them. This optimization problem is challenging due to the high dimensionality and the desire to find a globally optimal segmentation, which GWO addresses through its collective social behavior-inspired search mechanisms.

3.4 Q Function

In order to measure the performance of image segmentation methods without human interactions we need an evaluation criteria. There are many image segmentation evaluation functions that have been presented in literature[10, 21-22]. Those functions have been divided into numerous types including quantitative evaluation measures Borostti et al and also in “A new Optimization Based Image Segmentation method by Particle Swarm Optimization”.

$$Q(I) = \frac{1}{1000 \times Na} \sqrt{n} \sum_{j=1}^n \left[\frac{e_j^2}{1 + \log L(R_j)} + \left(\frac{M(L(R_j))}{L(R_j)} \right)^2 \right] \quad (5)$$

Figure 17: Q Function Formula

Where I is a given image, Na is the number of pixels in I, n is the number of pixels in jth region and e_j is the color error of jth region. The first term of this equation is a normalization factor, and the second term penalizes results with too many regions. Based on this we have developed a python function which can evaluate the quality of image segmentation by calculating a value that reflects the effectiveness of the segmentation in partitioning the image into regions with uniform color or intensity. The q function incorporates both the color error within regions and the size of the regions to penalize large variations within a region to normalize the impact of a region size on overall score. Specifically, the formula calculates a term for each region that combines the sum of squared color errors within the region, normalized by the region's size and the mean pixel value, with a logarithmic penalty for smaller regions.

- Na is the total number of pixels in the image,
- e_j is the sum of squared color errors within the jth region,
- LR_j is the number of pixels in the jth region,
- MLR_j is the mean pixel value in the jth region.

So here the lower Q value would indicate a better segmentation. This is because a lower Q value suggests that the regions have been partitioned in such a way that the within region color variance (error) is minimized relative to the size of the region and its mean intensity. The function penalize high color errors and smaller regions (which are less likely to represent meaningful segments of the image) and rewards segmentations that achieve uniformity within regions while considering the size of those regions.

The evaluation is performed through a Python script that first defines the $Q(I)$ function. This script operates by calculating the color error for each segmented region and then computing the overall $Q(I)$ value for the image. The color error is the sum of the squared differences between the mean gray level of a region and the gray levels of each pixel within that region. This metric reflects the homogeneity of the color within the region, with a lower error indicating a more consistent region in terms of color.

- **calculate_color_error(image, region_mask):**

```
def calculate_color_error(image, region_mask):  
    """  
    Calculate the color error for a given region in an image.  
    """  
    region_image = image[region_mask]  
    mean_gray_level = np.mean(region_image)  
    color_error = mean_gray_level - region_image  
    return np.sum(color_error ** 2) # Sum of squared errors in the region
```

○

Figure 18: Code Snippet for Calculating the Color Error

- This function computes the color error for a given region of an image. It uses the region mask to extract the relevant part of the image, calculates the mean gray level, and then determines the sum of squared differences from this mean.

- **Q(I, regions):**

```
def Q(I, regions):
    """
    Calculate Q for a given image I and a list of regions.
    """
    Na = I.size # number of pixels in I
    n = len(regions) # number of regions

    sum_term = 0
    for region_mask in regions:
        Rj = I[region_mask] # Pixels in jth region
        L_Rj = Rj.size # number of pixels in jth region
        ej = calculate_color_error(I, region_mask)
        M_L_Rj = np.mean(Rj) # mean pixel value in jth region

        sum_term += (ej / (1 + np.log(L_Rj))) + (M_L_Rj / L_Rj) ** 2

    Q_value = (1 / (1000 * Na)) * np.sqrt(n) * np.sqrt(sum_term)
    return Q_value
```

-

Figure 19: Q Function Code Snippet

- It calculates the Q value for a segmented image I and a list of region masks. The function takes into account the size of the image, the number of regions, and includes the normalization factor and penalties as described by the original $Q(I)$ function formula.

- **compare_segmentations(image1, image2, num_regions=5):**

- The core function that loads two images, potentially resizes them for comparison, plots them for visual inspection, and then uses the previously mentioned functions to compute and return the Q values for each image.

To compare the segmentation quality, the script simulates segmentation by generating regions within two images and then applying the $Q(I)$ function. The images are first converted to grayscale to simplify the color error calculation. This conversion is typical in segmentation tasks where color information may not be as crucial as the structure and outlines of the segmented regions.

The script is designed to be flexible, allowing for an adjustable number of regions in the segmentation comparison. This feature is particularly useful when testing the robustness of the $Q(I)$ evaluation across different segmentation granularities.

4. Experiments

Following are some segmented outputs using Particle Swarm Optimization algorithm. Notice the segmentation getting better with more centroids and more particles.

4.1 Experiments with Particle Swarm Optimization

- 6 centroids and 36 particles

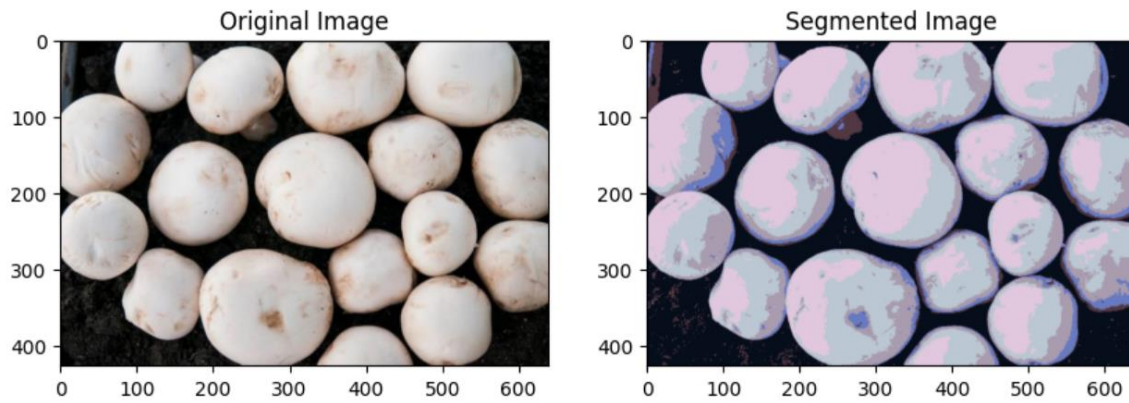


Figure 20: PSO Segmentation 1

- 7 centroids and 49 particles

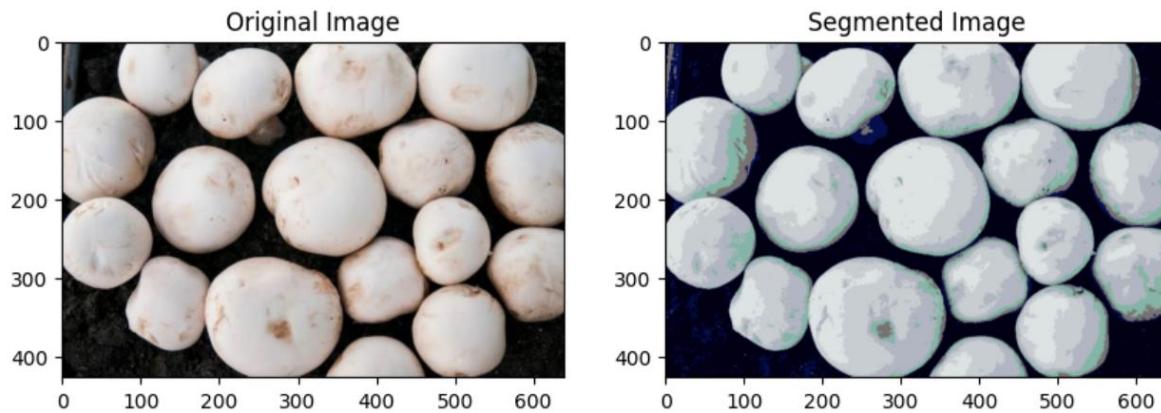


Figure 21: PSO Segmentation 2

- 8 centroids and 64 particles

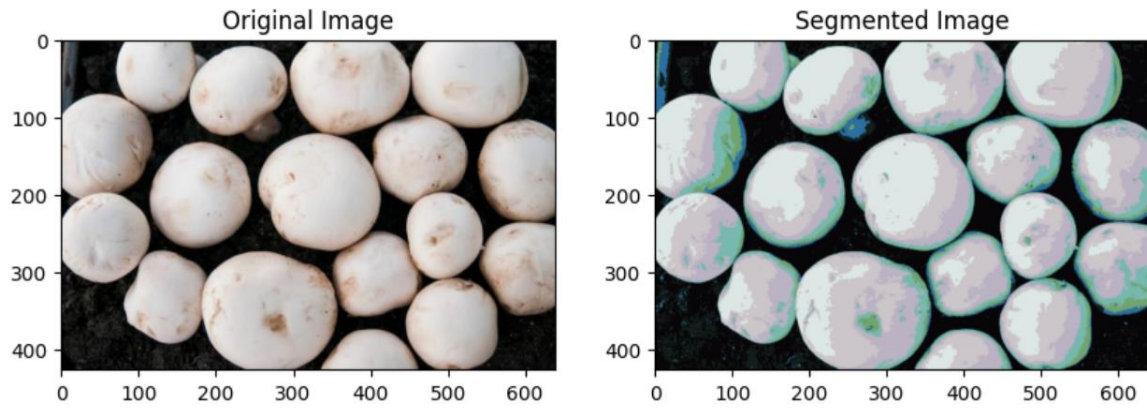


Figure 22: PSO Segmentation 3

- 9 centroids and 81 particles

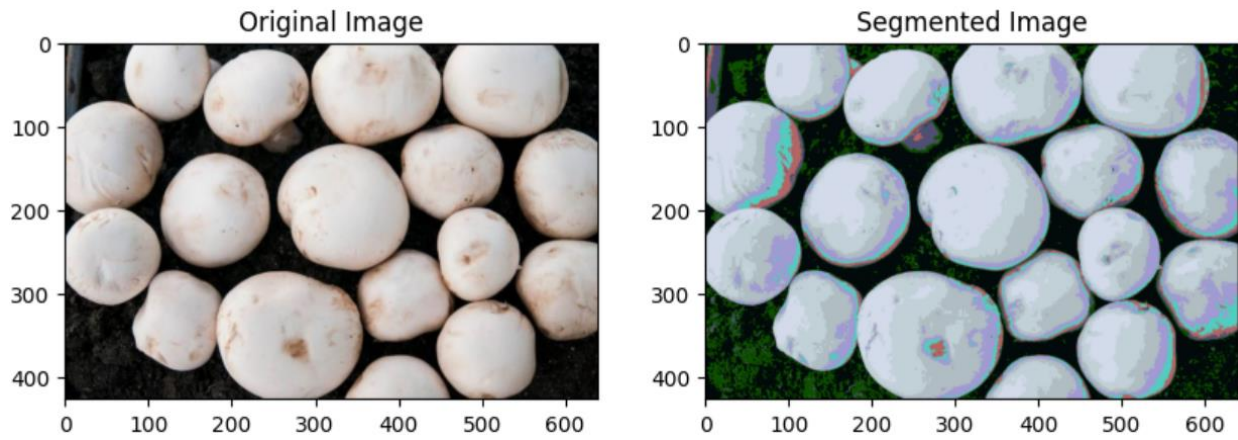


Figure 23: PSO Segmentation 4

- 10 centroids and 100 particles

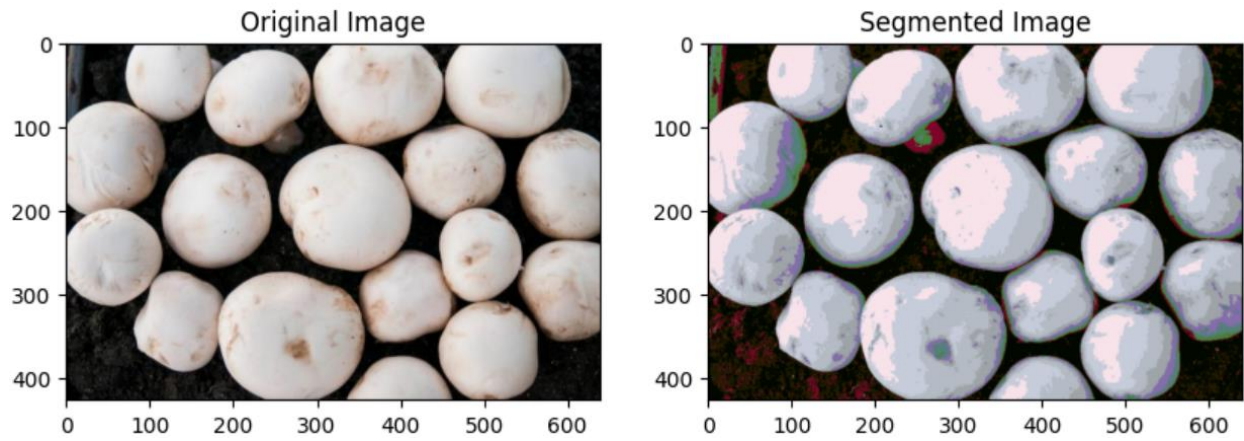


Figure 24: PSO Segmentation 5

- 11 centroids and 121 particles

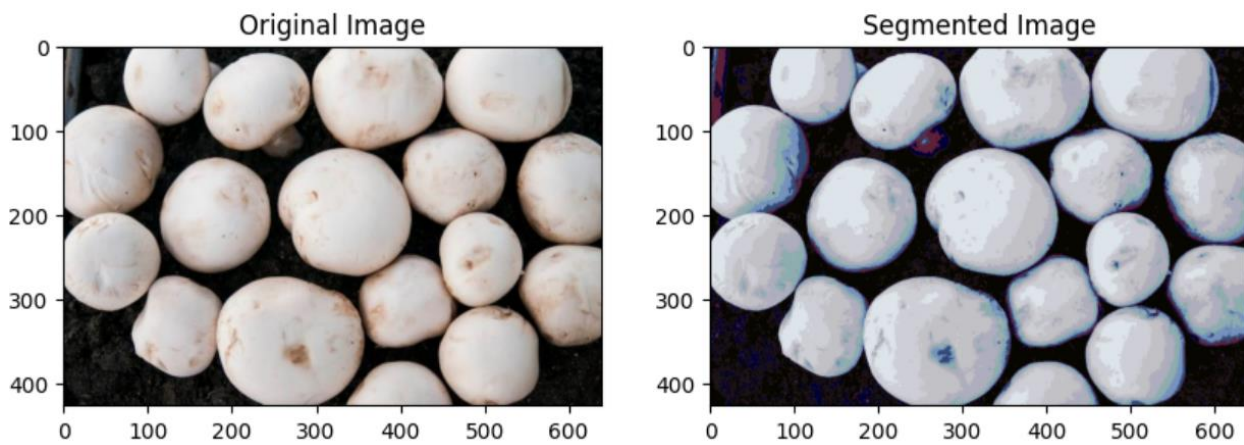


Figure 25: PSO Segmentation 6

4.2 Experiments with Firefly Algorithm

In the following experiments Firefly is being used as the optimization algorithm when doing the segmentation.

- 6 centroids and 36 Fireflies

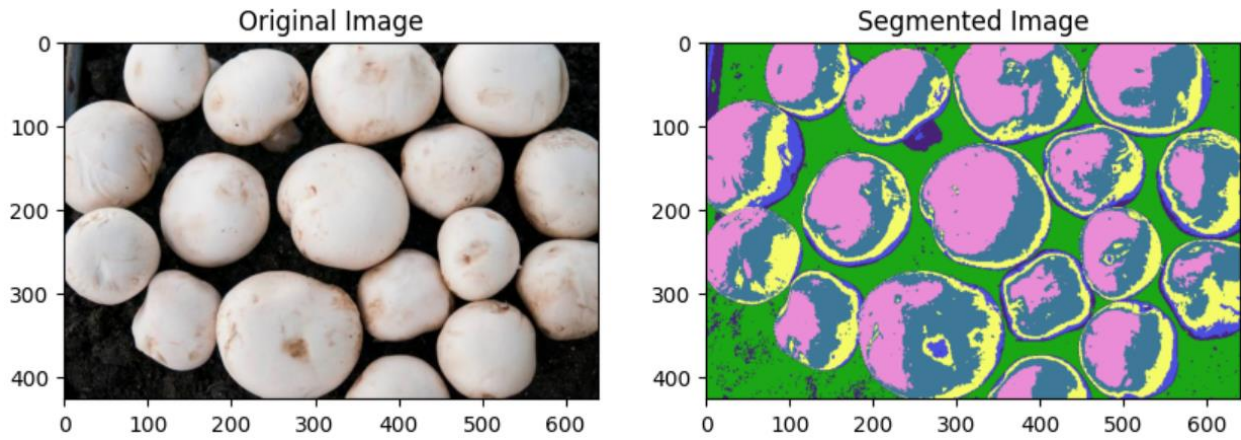


Figure 26: Firefly Segmentation 1

- 7 centroids and 49 Fireflies

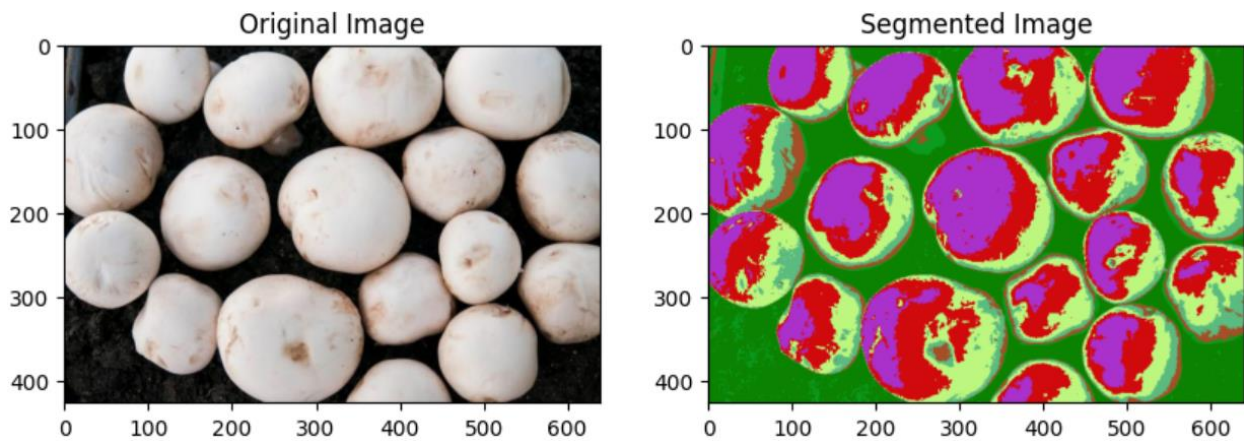


Figure 27: Firefly Segmentation 2

- 8 centroids and 64 Fireflies

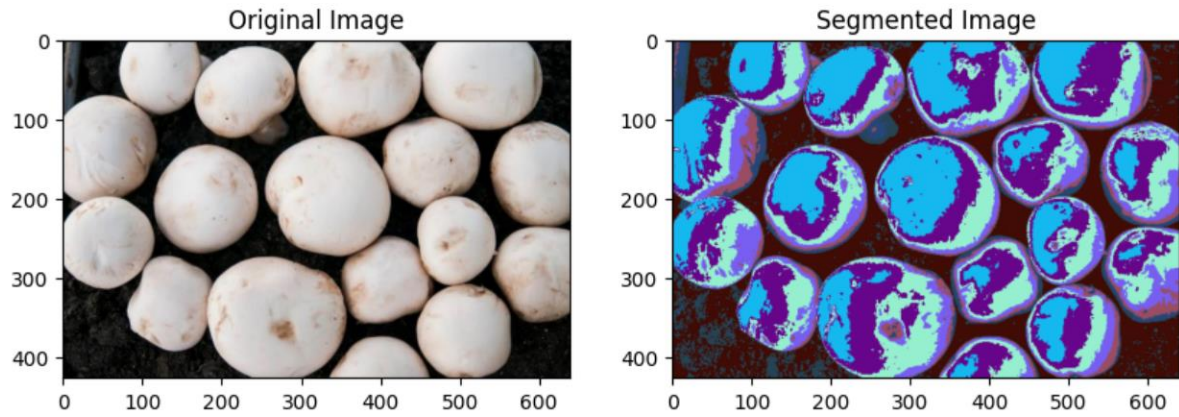


Figure 28: Firefly Segmentation 3

- 9 Centroids and 81 fireflies

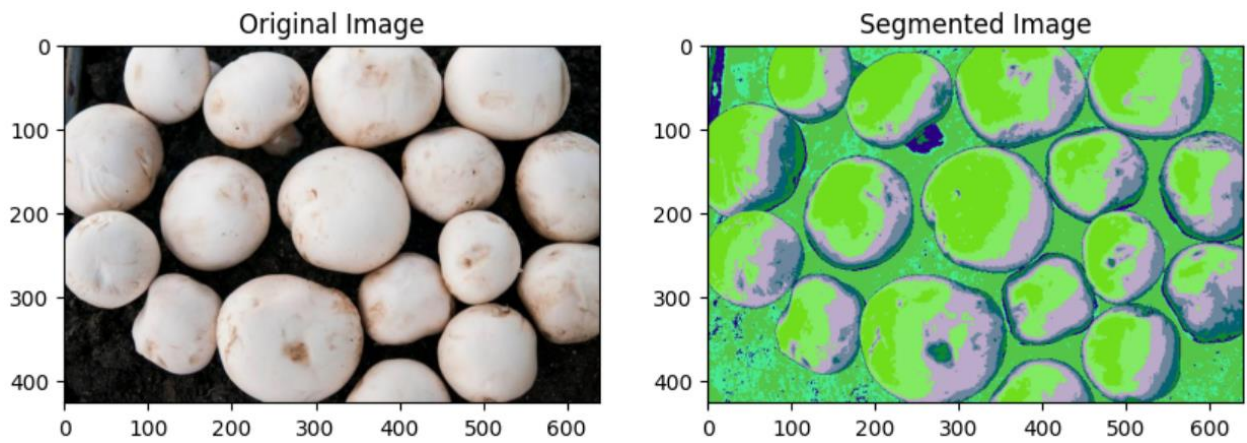


Figure 29: Firefly Segmentation 4

- 10 centroids and 100 Fireflies

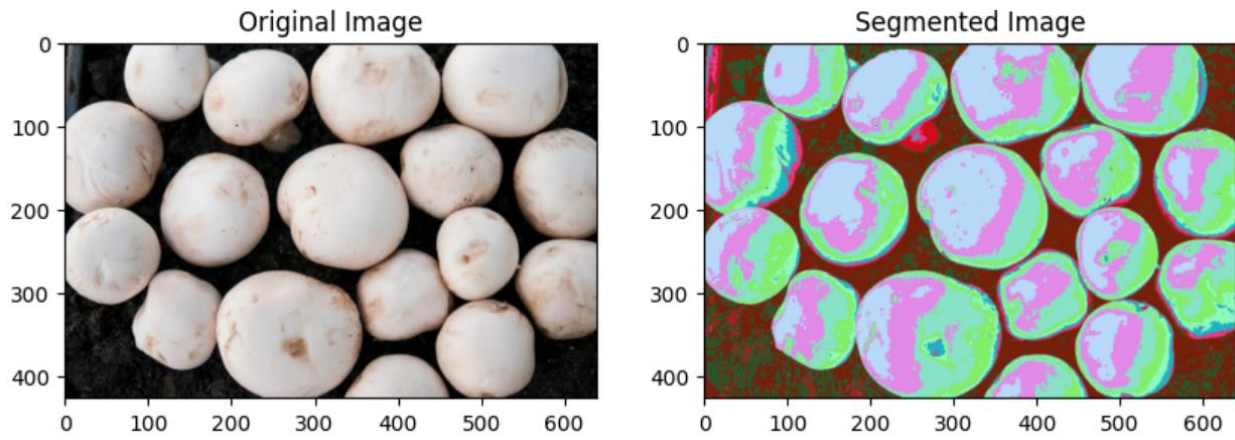


Figure 30: Firefly Segmentation 5

- 11 centroids and 121 fireflies

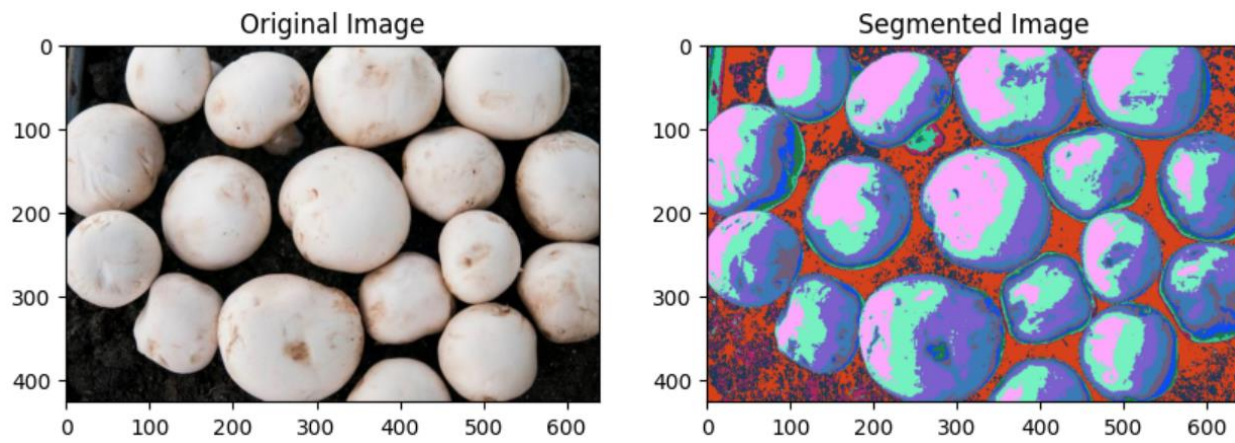


Figure 31: Firefly Segmentation 6

4.3 Experiments with Grey Wolf Optimization Algorithm

The following results are taken from segmentation of images using the Grey Wolf Optimization Function

- 6 centroids with 36 wolves

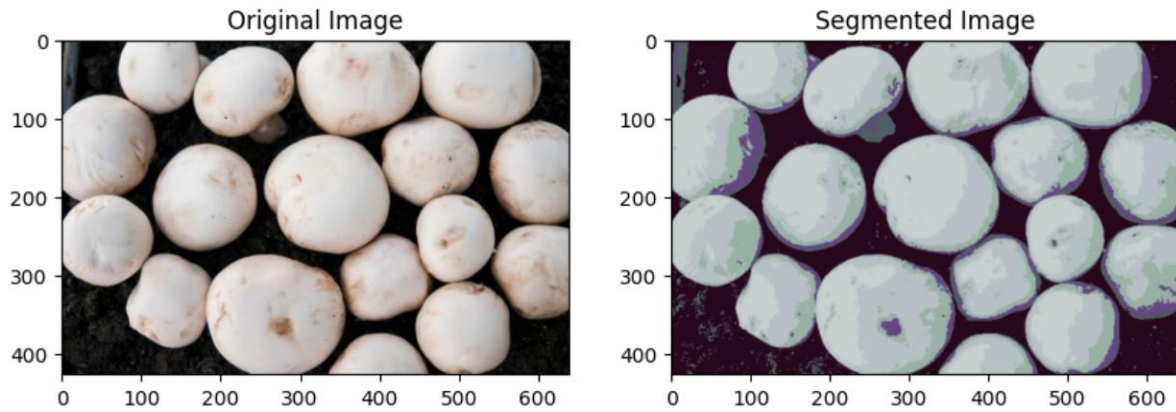


Figure 32: Grey Wolf Optimization Segmentation 1

- 7 centroids with 49 wolves

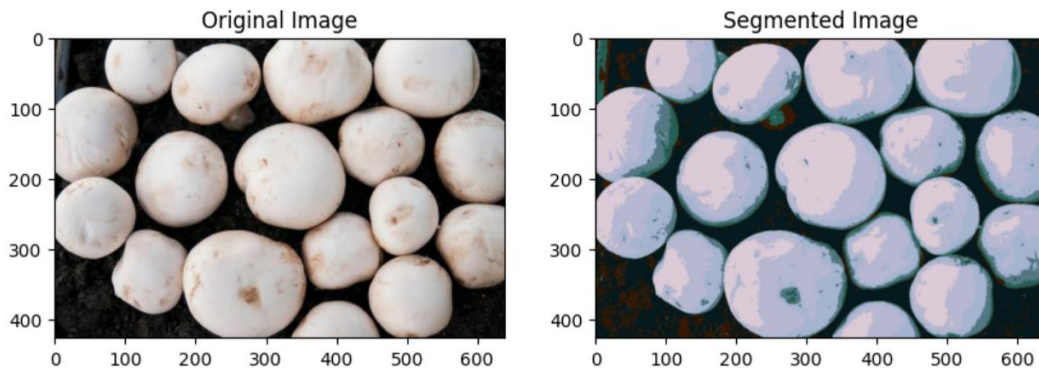


Figure 33: Grey Wolf Optimization Segmentation 2

- 8 centroids with 64 wolves

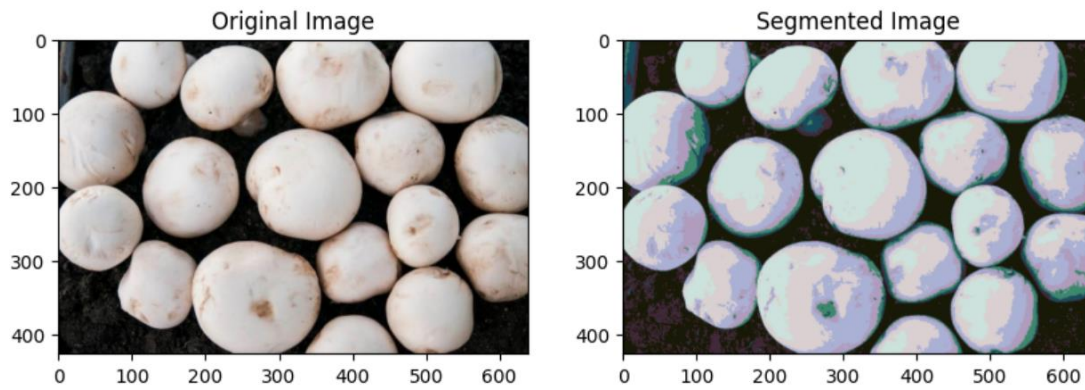


Figure 34: Grey Wolf Optimization Segmentation 3

- 9 centroids with 81 wolves

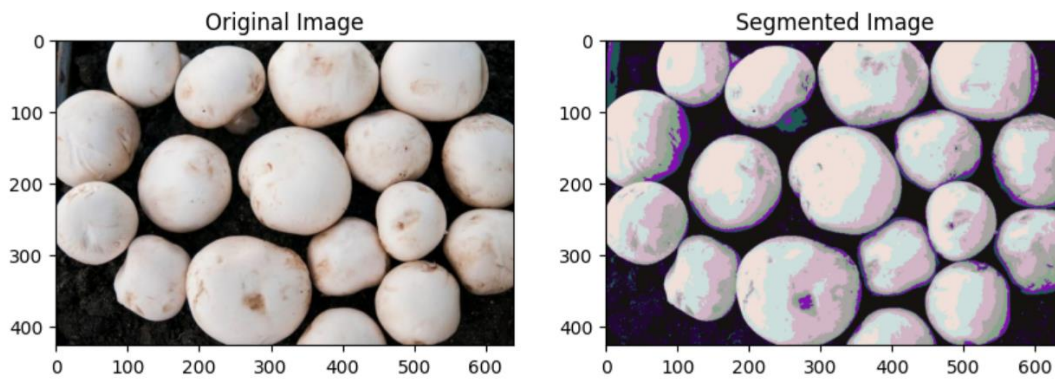


Figure 35: Grey Wolf Optimization Segmentation 4

- 10 centroids with 100 wolves

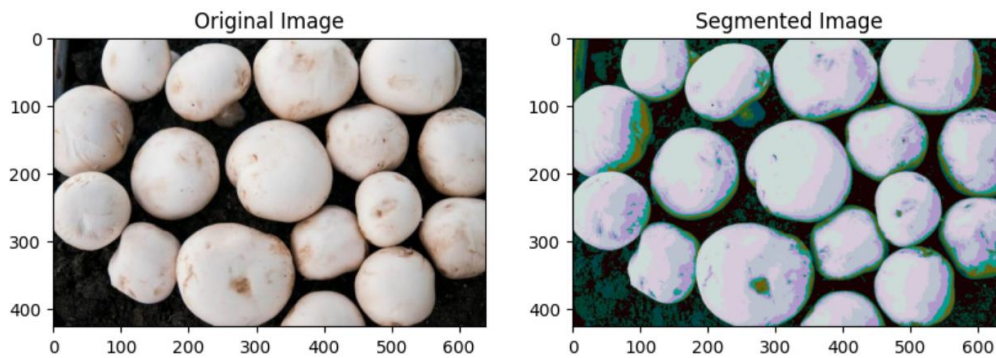


Figure 36: Grey Wolf Optimization Segmentation 5

- 11 centroids with 121 Wolves

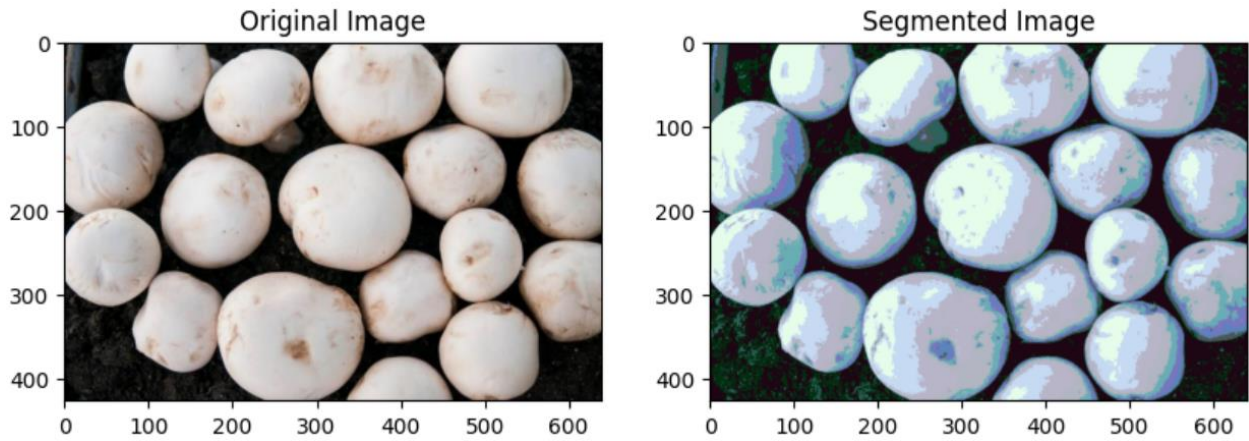
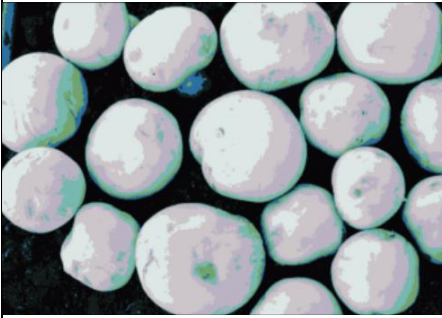
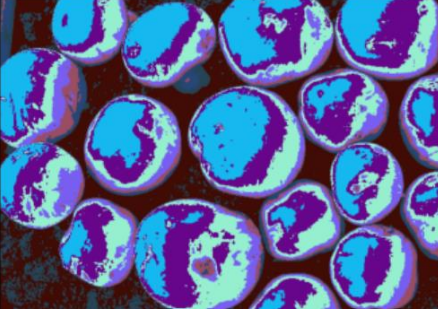
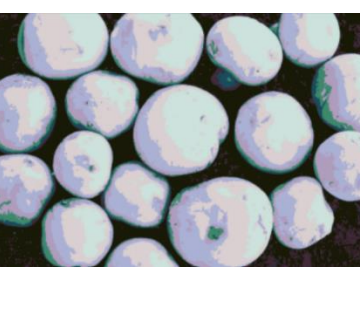
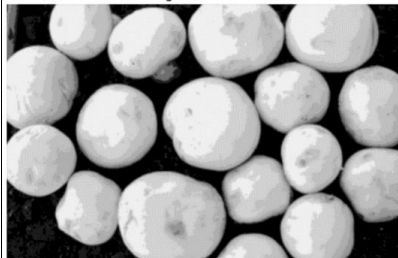
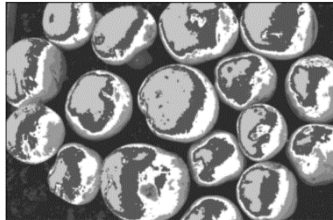



Figure 37: Grey Wolf Optimization 6

The following table shows a visual comparison of the three Swarm Intelligence algorithms when using the same number of particles. (Lower the Q value better the segmentation)

Table 2: Visual Difference Between Image Segmentation Using Different Swarm Intelligence Algorithms. (Q values are also given)

Particle Swarm Optimization	Firefly Optimization	Grey Wolf Optimization
		
Q value for the image: 0.00012690431162244316 	Q value for the image: 8.98730973204653e-05 	Q value for the image: 0.0001158353440669702 

4.4 Experiment Based on Segmentation Evaluation Algorithm

The following is also an example showcasing the Q function. Here image 1 is an image segmented using the Particle Swarm Optimization Algorithm. For this image 6 centroids and 36 particles have been used. Image 2 is an image segmented using Firefly Optimization Algorithm. Which uses 6 centroids with 6 fireflies.

Image 1

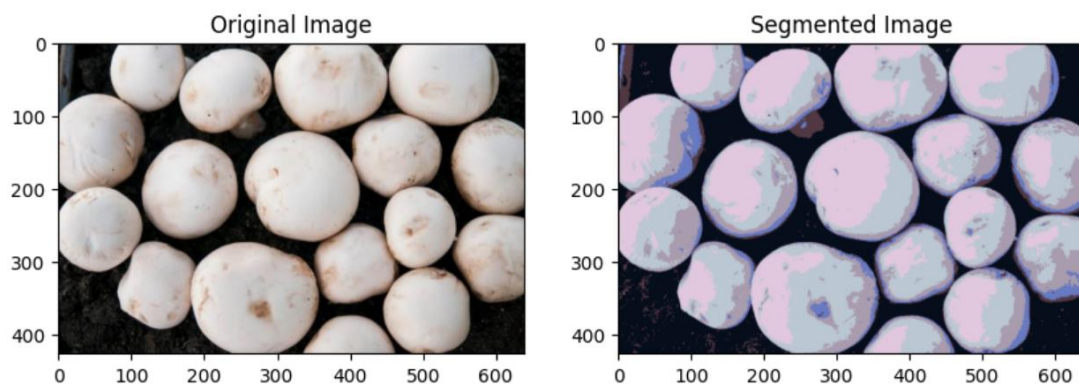


Figure 38: PSO Q Value Experiment 1

Image 2

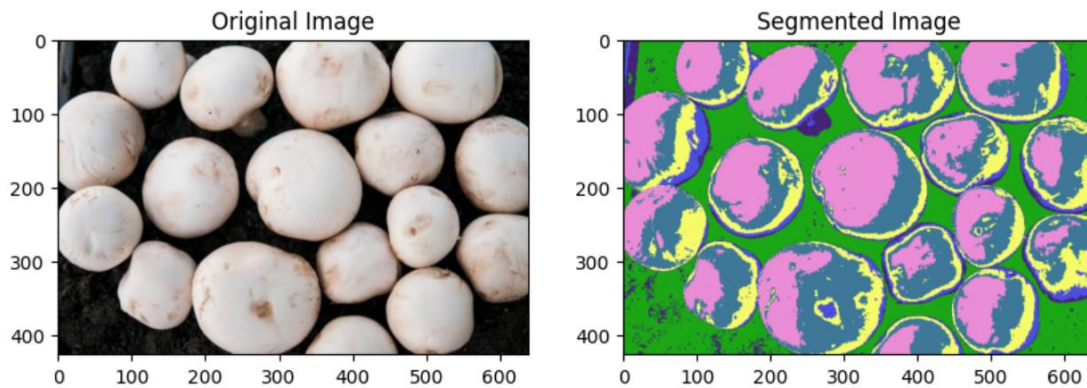


Figure 39: Firefly Q Value Experiment 2

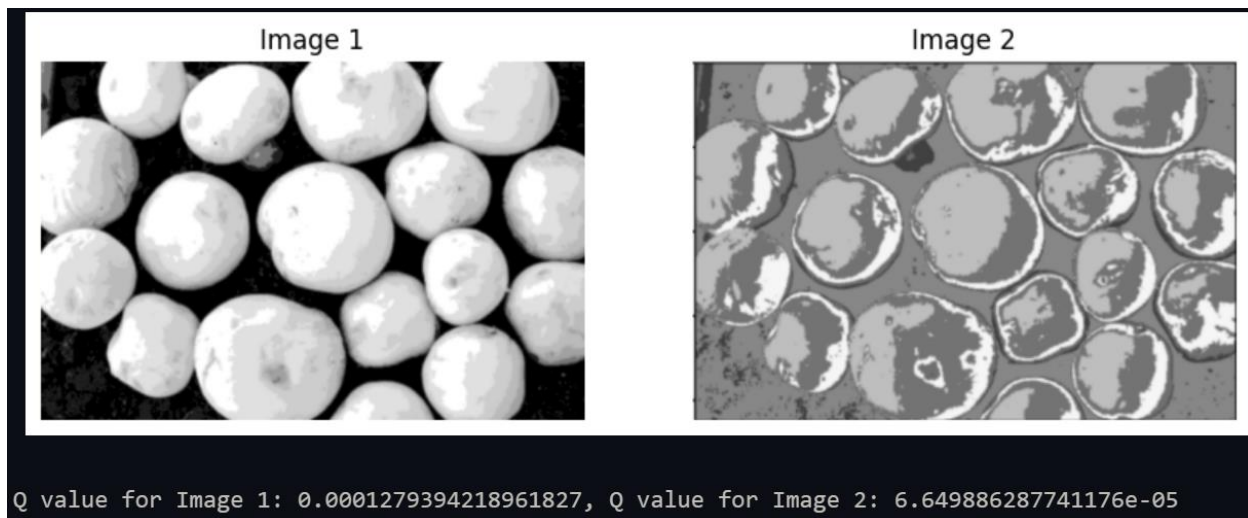


Figure 40: Q Value Comparison

Q Value for Image 1 : 0.0001279394218961827

Q Value for Image 2 : 6.649886287741176e-05

The Q function as described aims to evaluate the quality of image segmentation. This is specifically designed to quantitatively assess the performance of image segmentation methods without the need for human intervention, by evaluating the uniformity and accuracy of the segmentation in dividing the image into meaningful regions. The analysis of the Q values obtained for two segmented images highlights the effectiveness of this evaluation method in distinguishing between different segmentation qualities.

For Image 1, with a Q value of 0.0001279394218961827, the segmentation quality is lower compared to Image 2. This higher Q value suggests that Image 1's segmentation has either larger within-region color variances or smaller region sizes, both of which are penalized by the Q function. The presence of larger within-region color variances indicates a lack of homogeneity in color distribution within the segments, meaning that the algorithm might have struggled to accurately identify uniform regions based on color or intensity. On the other hand, smaller regions could imply an over-segmentation where the image is divided into too many small segments, potentially missing the larger, more meaningful structures within the image.

Image 2, which achieved a Q value of 6.649886287741176e-05, demonstrates a better segmentation quality. This lower Q value indicates a successful partitioning of the image into

regions with more uniform color or intensity, in line with the desired outcome of the segmentation process. It suggests that the segmentation method applied to Image 2 was more effective in minimizing within-region color error and creating segments of appropriate sizes that likely represent meaningful parts of the image. This improved performance could be due to a better algorithmic approach in handling the nuances of the image's features or a more suitable parameter setting that aligns with the specific characteristics of the image. This implies that the segmentation achieved by the Firefly Optimization algorithm image 2 has resulted in regions with more homogeneous color values and better adherence to the mean gray levels across the segments.

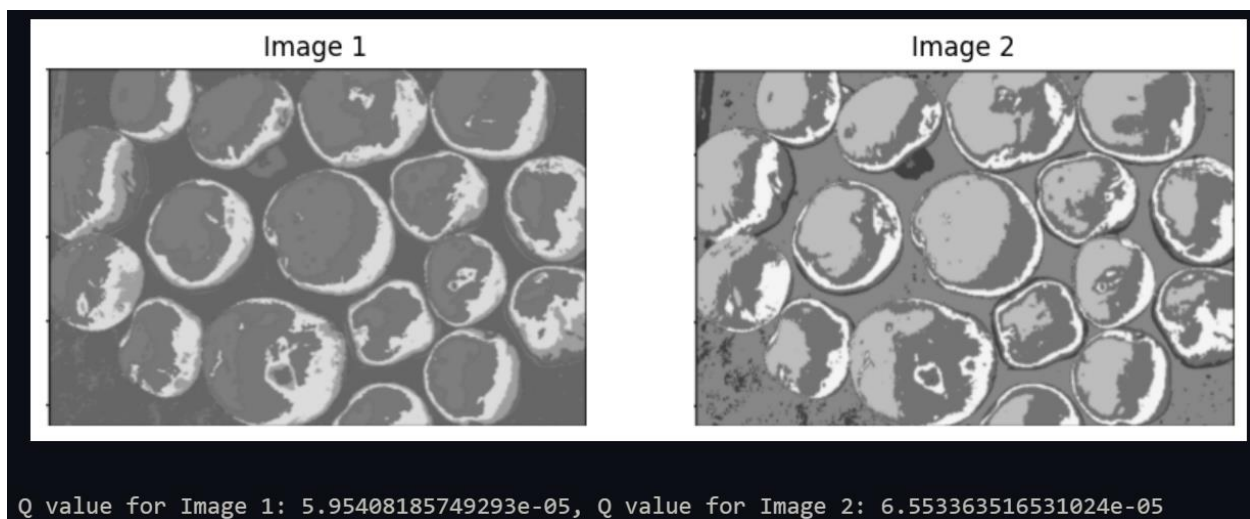


Figure 41: Q Value Comparison 2

In summary, the Q function serves as a robust tool for the quantitative evaluation of image segmentation, allowing for an objective comparison between different segmentation outputs. The lower Q value obtained for Image 2 underscores its superior segmentation quality, showcasing the algorithm's proficiency in creating homogenous and meaningful segments. This evaluation method not only aids in the assessment of segmentation techniques but also in the refinement of algorithms to achieve more accurate and visually coherent segmentation results.

4.5 Experiment based on the time it takes to segment the image.

Table 3: Segmentation Time Efficiency

Centroids	Fireflies / Particles / Wolves	Firefly Algorithm Time(seconds)	Particle Swarm Optimization Time(seconds)	Grey Wolf Optimization Time (seconds)
2	4	20.57	23.47	3.42
4	16	38.80	197.57	21.79
5	25	56.98	301.22	45.07
6	36	65.11	458.12	66.87
7	49	72.71	887.82	108.63
8	64	89.67	1102.42	141.25
9	81	150.74	2138.76	207.41
10	100	96.78	6088.10	289.03
11	121	83.07	1800.14	347.68
12	144	103.81	4194.43	533.31
13	169	99.09	3652.30	577.31
14	256	139.60	5636.07	982.07

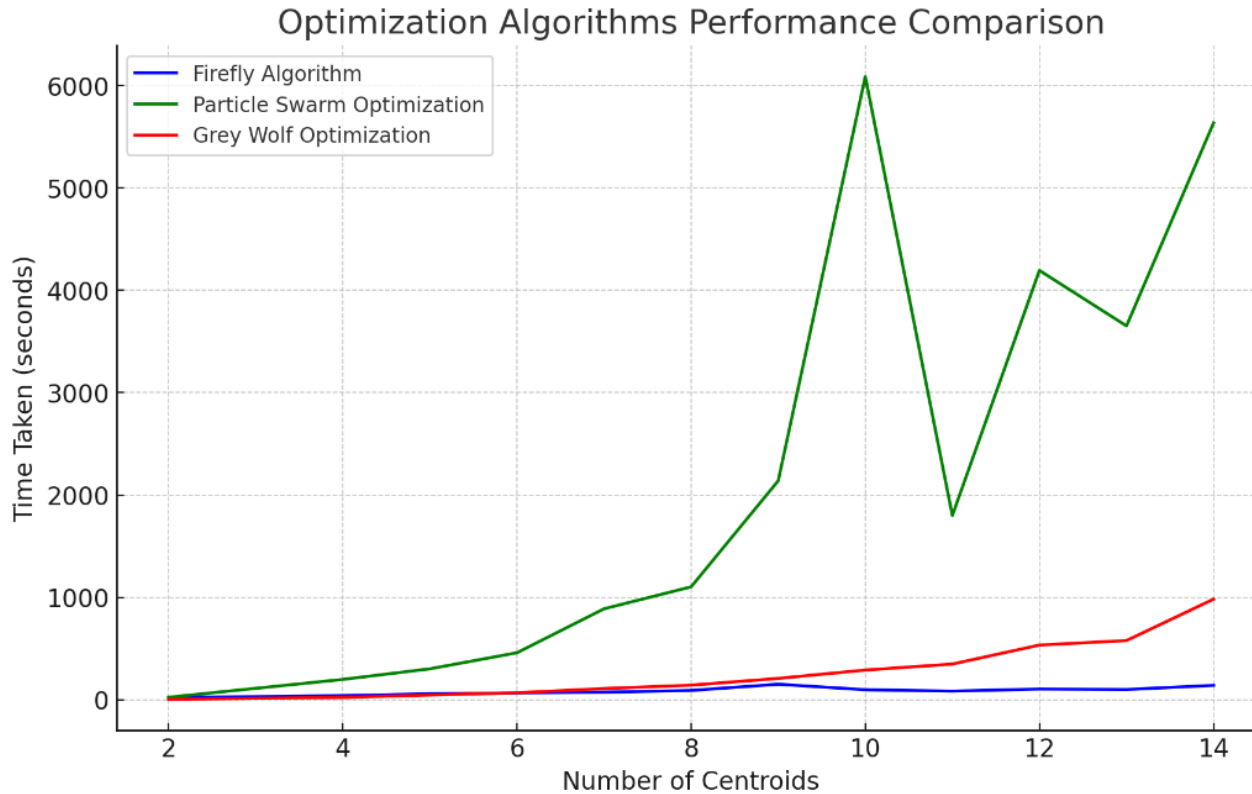


Figure 42: Algorithm Image Segmentation Time Comparison

According to the above table the efficiency of these algorithms varies significantly depending on the number of centroids and agents (fireflies, particles, wolves) used. This indicates that the complexity and computational cost of each algorithm are influenced by the parameters with higher numbers showing an increased processing time. During image segmentation process the Grey Wolf Optimization algorithm consistently shows the best performance in terms of time efficiency across all the tested settings. It is evident that with the lowest segmentation times in each scenario it appears to be the most efficient algorithm for this specific task of image segmentation especially in more complex scenarios with higher number of centroids and agents. The performance of the particle swarm optimization algorithm is notably affected as the complexity increases (from 2 to 5 centroids and 4 to 25 agents). It shows a substantial increase in processing time which suggests that PSO may not scale as efficiently as the other algorithms for this purpose. The Firefly algorithm demonstrates a more consistent increase in processing time relative to the increase in centroids and agents. This indicates a predictable scaling of computational cost which can be advantageous for planning and resource allocation in practical applications.

When choosing the right algorithm, the specific requirements of the image segmentation task should be taken into account. For tasks with quicker execution with less concern for computational resources GWO seems preferable. But for tasks where a balance between time efficiency and resource usage is needed Firefly is more suitable. The Q function experiments shows that segmentation examples when done in the same setting the segmentations done with Firefly algorithm tends to have a better score in terms of accuracy.

5. Conclusion and Future Work

This thesis has comprehensively evaluated the usage of Grey Wolf Optimization, Particle Swarm Optimization and Firefly algorithms in the context of image segmentation, thus revealing significant variations in performance based on the number of centroids and agents involved. The Q values obtained for segmented images using PSO, Firefly, and GWO algorithms under similar settings (i.e., with the same number of particles and centroids) were 0.00012690431162244316, 8.98730973204653e-05, and 0.0001158353440669702, respectively. These values serve as objective indicators of segmentation quality, with lower Q values denoting superior segmentation outcomes.

The GWO algorithm consistently outperforms others in time efficiency across various settings, making it an ideal candidate for tasks requiring rapid execution with less emphasis on computational resource management. It emerges as the top performer in terms of time efficiency, consistently outpacing both PSO and the Firefly algorithm across various configurations. This characteristic positions GWO as a preferable choice for applications that prioritize rapid execution over computational resource considerations. However, the Q value for GWO, while competitive, does not secure the top spot in segmentation quality, suggesting a trade-off between speed and the precision of segmentation outcomes.

Conversely PSO's performance deteriorates with increased complexity including its potential scalability issues for image segmentation tasks. On the other hand, it shows a decline in performance with increased complexity. This observation hints at potential scalability issues, marking PSO as less suitable for complex image segmentation tasks where both the number of centroids and agents are high. Its Q value, the highest among the three, further underscores this limitation, indicating that while PSO can achieve decent segmentation results, it may not be the most efficient or effective option when faced with intricate segmentation challenges.

The firefly algorithm demonstrates a predictable increase in processing time with more centroids and agents, suggesting a scalable and resource efficient option for applications especially when accuracy is important. This is also noted for its predictable increase in processing time with the

addition of more centroids and agents, demonstrates a balance between scalability and resource efficiency. Its Q value, the lowest among the three algorithms, signifies the highest segmentation quality, revealing its strength in accurately partitioning images into homogeneous regions. This outcome suggests that the Firefly algorithm excels in scenarios where accuracy and quality of segmentation are paramount, benefiting applications that can afford the trade-off for slightly increased computational costs. The Q function's evaluating segmentation performance without human intervention is crucial. And when doing the experiments, the lowest Q value was taken by the Firefly algorithm. Which suggests that the Firefly algorithm is the best option among the selected algorithms to use for image segmentation when using Swarm Intelligence.

In conclusion, the evaluation showcases the Firefly algorithm as the superior choice for image segmentation tasks within the realm of Swarm Intelligence, especially when the focus is on achieving high-quality segmentation. Its performance, as evidenced by the lowest Q value, suggests that it adeptly balances computational demands with the precision of segmentation outcomes. Future work could explore further optimization of the Firefly algorithm to enhance its time efficiency, potentially making it the most robust option for a broader range of image segmentation applications. Additionally, investigating hybrid approaches that combine the strengths of these algorithms could open new avenues for achieving both high-speed and high-quality image segmentation, offering a compelling direction for subsequent research in this field. Another important thing to note is that there are also some future improvements that can be made in the Q function. We can make use of machine learning to segment the images and then use a modified Q function to evaluate those images. Especially when using Firefly algorithm there is a growing concern to use deep neural networks with ensemble methods to use in semantic segmentation[23]. Another field to look forward is the medical computing where research suggests that using K Means with Swarm Intelligence in medical image segmentation gives better results when coupled with Convolutional Neural Networks[24].

In conclusion of this thesis, the field of image segmentation is on the cusp of significant technological advancements. The exploration of Swarm Intelligence algorithms in this work lays a solid foundation for future research and development.

References

- [1] A. Kumara and S. Raheja, "Edge Detection using Guided Image Filtering and Enhanced Ant Colony Optimization," in *Proceedings of the International Conference on Smart Sustainable Intelligent Computing and Applications under ICITETM2020*, Procedia Computer Science, vol. 173, pp. 8–17, 2020.
- [2] C. Fahy, S. Yang, and M. Gongora, "Ant Colony Stream Clustering: A Fast Density Clustering Algorithm for Dynamic Data Streams," *IEEE Transactions on Cybernetics*, vol. 49, no. 6, pp. 2215, June 2019.
- [3] Muhammad, and H. Chen, "Chaotic random spare ant colony optimization for multi-threshold image segmentation of 2D Kapur entropy," *Knowledge-Based Systems*, pp. 106510, Nov. 2020.
- [4] M. Xu, L. Cao, D. Lu, Z. Hu, and Y. Yue, "Application of Swarm Intelligence Optimization Algorithms in Image Processing: A Comprehensive Review of Analysis, Synthesis, and Optimization," *Biomimetics*, vol. 8, no. 235, 2023.
- [5] S. El-Khatib, Y. Skobtsov, and S. Rodzin, "Theoretical and Experimental Evaluation of Hybrid ACO-k-means Image Segmentation Algorithm for MRI Images Using Drift-analysis," *Procedia Computer Science*, vol. 150, pp. 324-332, 2019.
- [6] S. Abdulateef and M. Salman, "A Comprehensive Review of Image Segmentation Techniques," *Iraqi Journal for Electrical and Electronic Engineering*, vol. 17, pp. 166-175, 2021.
- [7] D. Libouga Li Gwet, M. Otesteanu, I. O. Libouga, L. Bitjoka, and G. D. Popa, "A Review on Image Segmentation Techniques and Performance Measures," *International Journal of Computer and Information Engineering*, vol. 12, no. 12, 2018.
- [8] S. Mirjalili, S. Mirjalili, and A. Lewis, "Grey Wolf Optimizer," *Advances in Engineering Software*, vol. 69, pp. 46–61, 2014.
- [9] S. A. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey Wolf Optimizer," in *Advances in Engineering Software*, vol. not specified, pp. 46-61, 2014. DOI: not provided.

- [10] F. M. A. Mohsen, M. M. Hadhoud, and K. Amin, "A new Optimization-Based Image Segmentation Method By Particle Swarm Optimization," *International Journal of Advanced Computer Science and Applications (IJACSA), Special Issue on Image Processing and Analysis*, pp. 10-18
- [11] H. Zhang, J. E. Fritts, and S. A. Goldman, "Image segmentation evaluation: A survey of unsupervised methods," *Computer Vision and Image Understanding*, vol. 110, no. 2008, pp. 260–280, 2008.
- [12] V. Vijaya, A. R. Kavitha, and S. R. Roselene Rebecca, "Automated Brain Tumor Segmentation and Detection in MRI using Enhanced Darwinian Particle Swarm Optimization (EDPSO)," in *Proceedings of the 2nd International Conference on Intelligent Computing, Communication & Convergence (ICCC-2016)*, S. Patnaik, Ed., Bhubaneswar, Odisha, India, 2016, *Procedia Computer Science*, vol. 92, pp. 475-480.
- [13] Y. Shi, "A Modified Particle Swarm Optimizer," in *Proc. of the IEEE International Conference on Evolutionary Computation*, June 1998
- [14] J. Kennedy and R. C. Eberhart, *Swarm Intelligence*. [Book]. pp. 287-320.
- [15] A. Sharma and S. Sehgal, "Image segmentation using firefly algorithm," in *2016 International Conference on Information Technology (InCITe) - The Next Generation IT Summit on the Theme - Internet of Things: Connect your Worlds*, presented on 16 February 2017.
- [16] R. Capor-Hrosik, E. Tuba, E. DOLICANIN, R. Jovanovic, and M. Tuba, "Brain Image Segmentation Based on Firefly Algorithm Combined with K-means Clustering," *Studies in Informatics and Control*, vol. 28, 2019, doi: 10.24846/v28i2y201905.
- [17] S. Kapoor, I. Zeya, C. Singhal, and S. J. Nanda, "A Grey Wolf Optimizer Based Automatic Clustering Algorithm for Satellite Image Segmentation," in *Proceedings of the 7th International Conference on Advances in Computing & Communications (ICACC-2017)*, Cochin, India, Aug. 22-24, 2017.
- [18] H. Zhang, Z. Cai, L. Xiao, A. A. Heidari, H. Chen, D. Zhao, S. Wang, and Y. Zhang, "Face Image Segmentation Using Boosted Grey Wolf Optimizer," *Biomimetics*, vol. 8, no. 6, p. 484, 2023.

- [19] X. Yu and X. Wu, "Ensemble grey wolf optimizer and its application for image segmentation," *Expert Systems with Applications*, vol. 209, 2022, Art. no. 118267. ISSN 0957-4174.
- [20] X. Yao, Z. Li, L. Liu, and X. Cheng, "Multi-Threshold Image Segmentation Based on Improved Grey Wolf Optimization Algorithm," in *IOP Conference Series: Earth and Environmental Science*, vol. 252, no. 042105, 2019
- [21] Y. J. Zhang, "A survey on evaluation methods for image segmentation," *Pattern Recognition*, vol. 29, no. 8, pp. 1335-1346, 1996
- [22] H. Zhang, J. E. Fritts, and S. A. Goldman, "Image segmentation evaluation: A survey of unsupervised methods," *Computer Vision and Image Understanding*, vol. 110, no. 2, pp. 260-280, 2008.
- [23] L. Zhang, S. Slade, C. P. Lim, H. Asadi, S. Nahavandi, H. Huang, and H. Ruan, "Semantic segmentation using Firefly Algorithm-based evolving ensemble deep neural networks," *Knowledge-Based Systems*, vol. 277, no. 110828, Oct. 9, 2023.
- [24] C. Kaushal, M. K. Islam, S. A. Althubiti, F. Alenezi, and R. F. Mansour, "A Framework for Interactive Medical Image Segmentation Using Optimized Swarm Intelligence with Convolutional Neural Networks," *Computational Intelligence and Neuroscience*, vol. 2022, Article ID 7935346, 2022. Published online Aug. 24, 2022. [Online]. Available: <https://doi.org/10.1155/2022/7935346>