SSRG International Journal of Electronics and Communication Engineering
ISSN: 2348-8549/ https://doi.org/10.14445/23488549/IJECE-V13I2P102

Volume 13 Issue 2, 11-29, February 2026
© 2026 Seventh Sense Research Group®

Original Article

Comparative Analysis of Parallel Particle Swarm Optimization in Lightweight Deep Learning: A MobileNet Case Study for Skin Cancer Detection

Salvia Elmassry[1], Ahmed Abouelfarag[2], Marwa Ali Elshenawy[1*], Rania Birry[1]

[1]*Department of Computer Engineering, College of Engineering, Arab Academy for Science, Technology and Maritime Transport, Alexandria, Egypt.*
[2]*College of Artificial Intelligence, Arab Academy for Science, Technology and Maritime Transport, Alexandria, Egypt.*

[*]*Corresponding Author : marwa_elshenawy@aast.edu*

Received: 15 November 2025 Revised: 16 December 2025 Accepted: 17 January 2026 Published: 20 February 2026

Abstract *- Skin Cancer is one of the most common cancers around the world. Detecting it in early stages is essential to improve the patient's chance of survival. This study introduces a framework designed to improve the classification performance of binary skin cancer classification by tuning the hyperparameters of the MobileNet model using Particle Swarm Optimization (PSO). The optimizer is implemented using two approaches: a traditional sequential PSO and a parallel multi-threaded PSO approach. The ISIC 2018 dataset is used, and the hyperparameters such as the batch size, optimizer selection, and dense layer size were considered for tuning. The findings show that both approaches have achieved a strong classification accuracy between 95% and 97%. The parallel approach, however, has shown 20x up to 46x reduction in the training time, yet this was accompanied by an increase in memory usage, which rose by approximately 20% to 150% compared to the sequential approach, depending on swarm size and iteration count. These findings show the resource trade-off between training time vs memory consumption for applying an optimized Deep Learning Model in clinical settings, especially where computing resources are limited.*

Keywords *- Deep Learning, Hyperparameter optimization, MobileNet, Particle Swarm Optimization, Skin cancer classification.*

1. Introduction

Among malignant diseases, skin cancer accounts for a substantial and steadily increasing global disease burden. It is essential to detect it as early as possible with a high level of accuracy, as early detection of lesions increases survival rates. Accurate diagnosis's main dependency is on expert dermatologists, which creates bottlenecks, especially in places and regions with limited healthcare resources. As a result of these limitations, increasing attention has shifted toward automated computer-aided diagnostic systems within both clinical practice and research. Deep Learning based approaches are being explored for their ability to support dermatological decision-making in a manner that is rapid, scalable, and cost-effective.

Automated skin lesion classification is now predominantly addressed using Deep Neural Networks (DNNs), mainly due to the availability of large-scale dermoscopic datasets such as ISIC and HAM10000. Within this setting, strong results have been reported on public benchmark datasets using a range of Convolutional Neural Network (CNN) architectures, including ResNet, DenseNet, MobileNet, EfficientNet, as well as YOLO-based pipelines [1]. For example, "DSCC_Net" achieved classification accuracies of up to 94.8% in multiclass skin lesion classification experiments using combined dermoscopic datasets and a deep CNN-based pipeline. However, real-time deployment, especially in mobile or edge computing environments, may be restricted because of the dependence on deep and computationally intensive CNN architectures. Comparative studies evaluating YOLOv3 through YOLOv7 reported mean average precision values of approximately 70–75% when applied to skin lesion detection tasks [2]. These results show their real detection capabilities and suitability for tele dermatology applications. At the same time, they indicate ongoing challenges in accurately detecting small or irregular lesions.

The direction towards using lightweight CNN models was because of their computational efficiency, which was the reason behind Turkernet, who used an efficient architecture designed for deployment on resource-

This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)

constrained devices and yet achieved 92.5% for classification accuracy [3]. For this reason, the compromise between maintaining classification accuracy and reducing computational complexity proved challenging. MobileNet variants augmented with fused spatial channel attention have achieved classification accuracy of up to 96.7% on the HAM10000 dataset [4]. This shows that attention-enhanced lightweight models can deliver high performance while still maintaining computational efficiency. However, combining multiple models within a single framework has become a common strategy for skin lesion classification. One reported implementation integrated several CNN architectures, including VGG16, Inception-V3, and ResNet-50, and achieved classification accuracy of up to 96% on the balanced HAM10000 dataset. Comparable performance was also observed on the ISIC benchmark, with the ensemble design mitigating the effects of class imbalance [5]. Hybrid pipelines that integrate Deep Learning with segmentation and feature fusion have been reported to achieve classification accuracies above 98%. However, this level of performance is typically accompanied by increased computational overhead and higher inference latency [6]. Performance gains have also been observed using optimization-driven and attention-enhanced network designs. One reported approach based on SpaSA-optimized fully convolutional encoder–decoder networks, combined with adaptive CNN classifiers, achieved classification accuracies exceeding 95% on dermoscopic datasets [7]. In related work, multi-branch CNN architectures incorporating grouping cascade attention produced accuracy levels in the range of approximately 94–96% across several medical image classification benchmarks [8]. These improvements, however, were accompanied by increased computational demands. Strong results have likewise been reported using transfer learning and stacking ensemble strategies, with accuracies reaching up to 95.1% [9]. A key limitation of these methods is their sensitivity to dataset bias and domain shift, factors that reduce robustness when applied across different data distributions.

Evidence from the literature published between 2023 and 2025 shows that CNN-based methods are capable of achieving strong performance in skin lesion analysis, as reflected across survey articles, review studies, and representative research contributions. Under controlled experimental conditions, classification accuracies on benchmark dermoscopic datasets are frequently reported to exceed 94% [10-12]. Achieving high classification accuracy alone has not been sufficient to enable broad clinical use. Instead, real-world deployment continues to be limited by recurring issues such as dataset variability, inconsistencies in annotation quality, restricted model interpretability, and significant computational demands. These factors collectively constrain large-scale clinical deployment. Improving diagnostic accuracy alone does not adequately address the requirements of real-world clinical deployment. Practical use instead depends on achieving an effective balance between computational efficiency, robustness, and scalability.

Model performance in Deep Learning is not determined by network architecture alone. A substantial influence comes from the selection of training hyperparameters, including learning rate, batch size, optimizer choice, dropout rate, and dense layer configuration. In practice, identifying appropriate values for these parameters through manual tuning or grid search becomes computationally expensive, particularly as the dimensionality of the search space increases. For this reason, Metaheuristic Optimization Methods such as Particle Swarm Optimization (PSO) have been increasingly adopted. Sequential PSO has been shown to automate CNN hyperparameter selection effectively, improving classification performance while reducing reliance on manual tuning [13, 14]. Reported results indicate that PSO-tuned CNN models outperform baseline configurations and random search strategies, and further comparative studies have demonstrated that PSO frequently outperforms other evolutionary algorithms for CNN hyperparameter optimization [15].

Variants of PSO that incorporate multiobjective and hierarchical strategies have been used to improve optimization effectiveness and scalability in complex search spaces, primarily by enhancing convergence behavior and reducing communication overhead [16]. Within medical imaging applications, these PSO-based Deep Learning Frameworks have produced strong classification results across multiple tasks. For breast cancer diagnosis, multiobjective PSO-optimized models have reported classification accuracies of approximately 95%-96% [16]. Similarly, PSO-driven CNN hyperparameter optimization applied to mammography has achieved accuracies exceeding 97% under controlled experimental conditions [17]. Comparable performance has also been reported in other medical imaging applications, including breast histopathology analysis, where classification accuracy typically falls within the 95%–97% range. Taken together, these results support the suitability of optimized learning frameworks for high-accuracy diagnostic tasks [18].

Sequential PSO is inherently limited by its execution time, since particle evaluations are performed one after another and frequently involve training deep CNN models. As swarm size and iteration count increase, this computational burden grows, which directly constrains scalability. To address this limitation, parallel PSO frameworks have been developed that leverage CPUs, GPUs, and distributed computing environments. Reported implementations based on CUDA have achieved speedups of up to 15× relative to sequential PSO, while parallel and multi-swarm PSO strategies have produced substantial improvements in runtime scalability for large optimization workloads.

PSO-driven optimization frameworks have been applied to Deep Learning and medical imaging tasks without compromising classification accuracy, while offering measurable gains in computational efficiency. Several parallel variants have focused on improving convergence behavior. Parallel elitist and multi-swarm PSO approaches, for example, improve the balance between exploration and exploitation and reduce premature stagnation, resulting in faster convergence and fewer required iterations compared with standard PSO [19, 20].

Large-scale distributed optimization has also been explored using Spark-based PSO frameworks, which have reported runtime speedups on the order of 10% for data-intensive workloads, although communication and synchronization overhead across computing nodes remains a limiting factor [21]. Asynchronous and distributed parallel PSO implementations address part of this limitation by removing global update barriers, thereby reducing synchronization delays and idle time while preserving solution quality comparable to sequential implementations [24, 25]. Collectively, these developments illustrate the practical potential of parallel PSO for scalable optimization in computationally intensive applications and support its integration within deep learning workflows, where performance is strongly influenced by hardware configuration and parallel execution efficiency.

Parallel PSO has delivered measurable computational benefits across a range of application domains. However, its use for hyperparameter optimization in medical imaging has received comparatively limited attention. Existing work rarely provides a systematic comparison between sequential and parallel PSO implementations for lightweight CNNs, particularly with respect to execution time, memory consumption, and classification accuracy.

As a result, current studies often emphasize either accuracy gains or reductions in computational time in isolation. The absence of analyses that jointly consider execution time, memory usage, and scalability limits informed decision-making when deploying optimized Deep Learning Models in resource-constrained clinical and edge environments.

A direct comparison is conducted between sequential and parallel PSO implementations for hyperparameter optimization of a MobileNet-based binary skin cancer classifier. Performance is evaluated in terms of execution time, memory consumption, and classification accuracy across different swarm sizes and iteration counts. This comparison makes the trade-offs between computational efficiency and resource utilization explicit, providing practical insight into deploying Deep Learning–based skin cancer detection systems in real-world healthcare environments.

The proposed model and the rationale behind the design of each phase are presented in Section 2, where the role of every component within the overall pipeline is explained. Section 3 focuses on the experimental setup, starting with a description of the dataset and followed by the presentation and discussion of the obtained results. The paper concludes in Section 4 with a summary of the main findings, an outline of the key contributions, and a brief discussion of possible directions for future research.

2. Materials and Methods

This research aims to investigate the utilization of the hyperparameters of the DL model to enhance the performance of the binary skin cancer classification. Figure 1 shows the system architecture and flow used in the research. Particle Swarm Optimization (PSO) was applied to the hyperparameters of the MobileNet model. Parallel and sequential PSO implementations were introduced, focusing on their memory consumption and training time as one of the performance measures to monitor. The system consists of several stages:

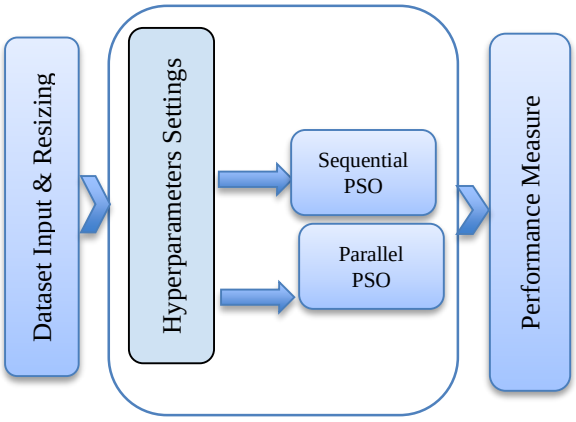


Fig. 1 The system architecture and flow

2.1. Dataset Description and Resizing

The ISIC 2018 dataset, obtained from Kaggle for skin cancer binary classification, containing 3297 RGB images, was used. It included 1800 benign and 1497 malignant images. The dataset was divided into training and testing subsets. The training set contained 1440 benign and 1197 malignant images, while 360 benign and 300 malignant images were kept for testing. All images were normalized and resized to 224x224 pixels to fulfill the Deep Learning Model's requirements.

2.2. Transfer Model (MobileNet)

Researchers commonly employ pre-trained models rather than training models from scratch. When evaluating the transfer learning models listed in Table 1, the number of parameters was one of the significant factors for the

selection process. Due to its low parameter count, MobileNet was selected as the preferred model for this study.

Table 1. Transfer models' parameters

Transfer Model Architecture	Approximate Number of Parameters
VGG16	~138 million
VGG19	~144 million
ResNet50	~23.5 million
ResNet101	~44.5 million
ResNet152	~60 million
InceptionV3	~23.8 million
Xception	~22.9 million
MobileNetV1	~4.2 million
MobileNetV2	~3.5 million
EfficientNetB0	~5.3 million
EfficientNetB7	~66 million
DenseNet121	~7 million
DenseNet201	~20 million

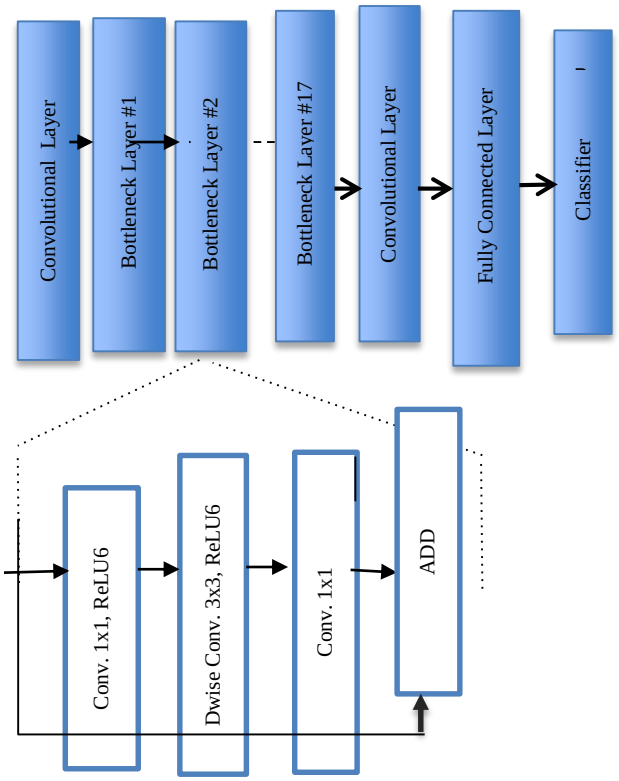


Fig. 2 MobileNetV2 architecture

The MobileNetV2 architecture, as depicted in Figure 2, was introduced by Google researchers. It is a lightweight and efficient convolutional neural network designed for on-device and embedded vision applications. MobileNetV2 employs depth-wise separable convolutions and inverted residual blocks to reduce computational complexity while maintaining competitive accuracy significantly. It balances between efficiency and performance, which is why it has been increasingly used for skin lesion and skin cancer classification. Cheng et al. [4] demonstrated the effectiveness of attention-enhanced MobileNet variants for skin lesion classification, while Tuncer et al. [3] highlighted the suitability of lightweight CNN architectures for resource-constrained medical imaging applications. In addition, combining ensemble and hybrid DL approaches with efficient backbone networks has achieved strong classification outcomes while addressing deployment limitations [5, 22].

MobileNet has its hyperparameters that need to be tuned and declared to enhance the performance of the model. These parameters are either for the architecture of the model itself or for the general training process. The training parameters are stated as follows:

- Epoch: The number of times the dataset is passed through the training model. Few epochs can lead to underfitting, while too many epochs can lead to overfitting.
- Learning rate: determines the step size the model updates its parameters during the training iteration. Too high can cause the training process to become unstable and diverge, whereas too low can result in excessively slow convergence.
- Dropout rate: applied to prevent overfitting by deactivating neurons during training.
- Batch size: defines the number of training samples processed before updating the model. Small batch sizes can improve generalization, while larger sizes speed up the training but might cause overfitting.
- Number of neurons: refers to the number of neurons in the fully connected layers attached to the MobileNet base as a classification head.
- Optimizer: The algorithm used to adjust the network's weights during training (e.g., Adam, SGD, RMSprop).

The next step in the research was to investigate the optimization strategies. The optimization aims to maximize or minimize a defined objective function. The optimization problem consists of three factors:

- Decision Variables (x): This component represents the tunable parameters to be adjusted to find the optimal solution. The decision variables are the hyperparameters of the DL model used.
- Objective Function (f(x)): This is the Function to be maximized or minimized to enhance the performance measures required. The objective Function in the research is the DL model. It is required to maximize its classification accuracy.

- Constraints: This is the possible range of each selected hyperparameter. For example, the range of the batch size hyperparameter was defined to be between 32 and 128.

Various intelligent optimization algorithms inspired by natural and biological processes have been a subject of study. These algorithms included Genetic Algorithms (GAs), Ant Colony Optimization (ACO), Simulated Annealing (SA), and Particle Swarm Optimization (PSO). In this study, PSO was selected due to its efficiency in hyperparameter optimization for convolutional neural networks [13, 14]. Not to mention its flexibility in parallel optimization settings [15]. PSO also has shown strong performance in medical imaging applications, including breast cancer diagnosis and mammography image classification [17, 18]. This was the primary motivation for its selection for skin lesion analysis in this work.

The MobileNet hyperparameters are depicted in Table 2. Some hyperparameters, such as the epochs, learning rate, and dropout rate, were set to fixed values. In contrast, other hyperparameters were designated for optimization and assigned both a lower and an upper bound.

Table 2. MobileNet hyperparameter settings

Hyperparameter	Fixed value	Optimized by PSO
Epochs	10	-
Learning Rate	0.001	-
Dropout	0.5	-
Batch size	-	[32 64 128]
Neurons	-	[64 128 256 512 1024 2048 4096]
Optimizer	-	[Adam, SGD, RMSprop, Adadelta, Adagrad]

2.2.1. Particle Swarm Optimization (Sequential)

Particle Swarm Optimization (PSO) is a powerful metaheuristic technique widely used to solve complex optimization problems. It works by mimicking the social behavior of a swarm, like a flock of birds or a school of fish. The main idea behind PSO is that a group of "particles" iteratively explores a search space. As these particles explore the search space with a certain velocity, they adjust their positions. This adjustment is influenced by their own best values (personal bests) and the overall best solution discovered by other swarms. As illustrated in Algorithm 1, the particles (swarms) with Size N and the number of iterations M must be declared before running the algorithm. The process starts by initializing these N particles. Each particle then gets to have its initial random Velocity (Vi) and its initial random Position (Pi). After initializing the swarm size, random velocities, and positions, these randomly generated positions serve as inputs to the fitness (objective) function. Now each particle has its Personal Best Value (Pbest). The best value achieved by all particles is now considered to be the Global Best Value (Gbest). The final step is updating the velocity and position based on Equations 1 and 2. This loop is executed M times.

$$V_{new} = \omega V_{old} + c_1 r_1 (Pbest - P) + c_2 r_2 (Gbest - P) \quad (1)$$

$$P_{new} = P + v_{new} \quad (2)$$

Where ω is the inertia weight, c1, c2, r1, and r2 are random values between 0 and 1, and P is the current position of particle i. Table 3 delivers the meaning and exact values used in this research.

Table 3. Values used for the velocity update

Parameter	Meaning	Value
ω	Inertia weight	0.6
$\mathbf{c_1}$	Cognitive parameter for PSO	2.0
$\mathbf{c_2}$	Social learning parameter for PSO	2.0
$\mathbf{r_1}$	Random number	[0, 1]
$\mathbf{r_2}$	Random number	[0, 1]

Algorithm 1: Particle Swarm Optimization (PSO) Algorithm

```
Input: N: particle size
       M: number of iterations
Initialize N particles
for i ← 1 to N do
    Generate random velocity V_i.
    Generate a random position P_i.
for i ← 1 to M do
    for i ← 1 to N do
        Evaluate the fitness function of particle i
        Get the local best (PBest) value of particle i
    Update the Global best (Gbest) of the particles
    for i ← 1 to N do
        Update the velocity of particle i using Equation (1)
        Update the position of particle i using Equation (2)
```

To clarify, each particle, such as particle i, consists of a set of values representing the selected hyperparameters. These hyperparameters, as mentioned earlier, are the batch size, number of neurons, and optimizer type. The allowable discrete values for these parameters are provided in Table 2 as follows: batch sizes of 32, 64, or 128; neuron sizes ranging from 64 to 4096; and optimizer options include Adam, SGD, RMSprop, Adadelta, and Adagrad. These hyperparameters are then used to train the MobileNet model, which serves as the fitness (objective) function. The aim is to maximize the model's classification accuracy. The remaining N particles are also assigned different values of these three hyperparameters, resulting in achieving different function values. It is obvious that the initialization phase is designed for sequential execution. Specifically, when initializing N particles. The velocity and position for each particle (from particle 1 to particle N) are generated randomly in an iterative way. Furthermore, the computation of the fitness function for each particle is also carried out

sequentially. This sequential behavior extends to nearly all phases of the algorithm, except for the global best value update. The update of the global best is performed at the end of each iteration, as it requires comparing the fitness values of particles.

2.2.2. Parallel Particle Swarm Optimization

After studying the phases of the sequential PSO algorithm, depicted in Figure 3, it was clear that specific components of the algorithm were executed sequentially relative to N, the particle size. These were the particle initialization, fitness function evaluation, and the updating of velocities and positions. Finding the structure of the iterations relative to the particle size gave the motivation to parallelize the algorithm, as supported by previous studies. It has been reported significantly from previous research that it caused speedups using GPU/CUDA-based implementations, parallel elitist and multi-swarm strategies [19, 20], and distributed frameworks such as Spark PSO [21]. However, developments involving asynchronous and multiobjective PSO have also shown the suitability of PSO for parallel execution [15, 16, 24]. Many of these phases can be processed concurrently by multi-threading. Thereby enabling parallel execution. Accordingly, the algorithm phases were revised and restructured to operate in a parallel manner. As illustrated in Figure 4, the phases were parallelized due to their inherent nature, hence leaving some to be executed sequentially.

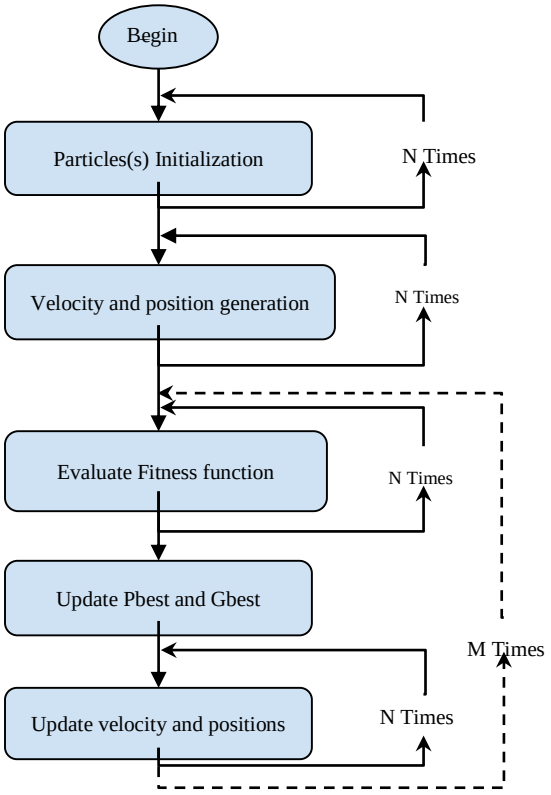


Fig. 3 Original sequential particle swarm organization

The algorithm starts by initializing the particles and creating multiple threads corresponding to the particle size (N). The N threads then work in a parallel manner to generate random velocities and positions for all particles.

The next stage involves the computation of the fitness function of the particles, which is also done at the same time. After that, the parallel evaluation of their respective Best Positions (Pbest) occurs. However, the determination of the global best value then breaking the parallel sequence, as updating the global best requires the knowledge of all best values of all particles, making it impossible to be parallelized.

Returning to parallelization, the updating of velocities and positions is performed for all particles. The phases that start by evaluating the fitness function and end with updating the velocities and positions are to be repeated M times.

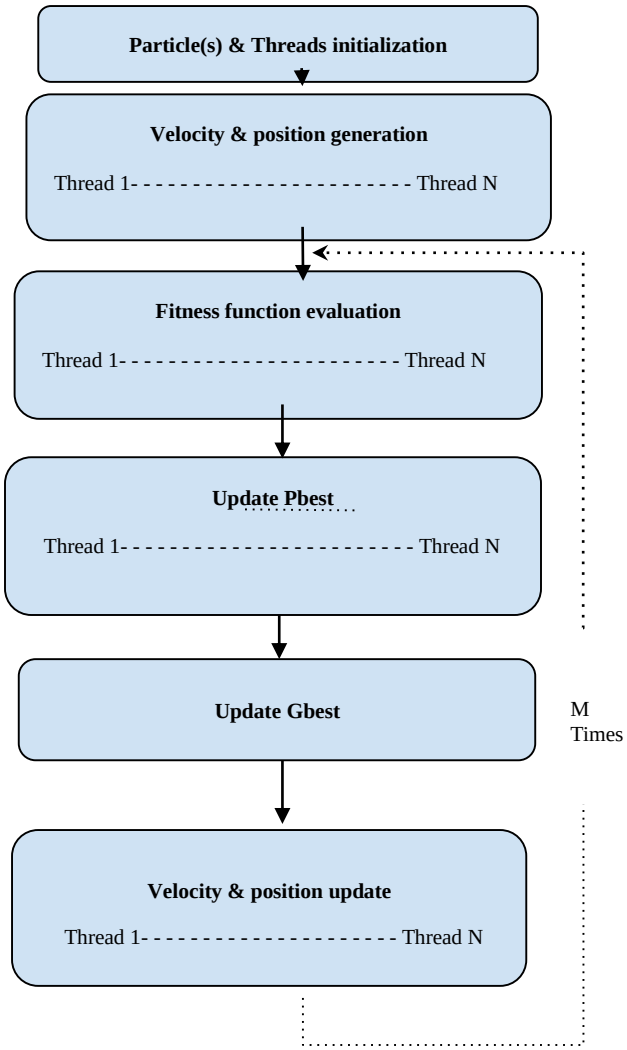


Fig. 4 Parallel particle swarm optimization

2.3. Performance Evaluation

The objective of this research is to evaluate and analyze the impact of implementing PSO in a sequential and parallel manner on skin cancer classification. The evaluation is done by measuring the execution time and the memory consumption during the training process during the training process of the DL model. An extra investigation was done to note how variations in swarm size and the number of iterations might change these measures. It was also essential to maintain the accuracy of the model, despite the application of these techniques. The performance of the model was evaluated using the following criteria:

- Accuracy
- Execution Time
- Memory Consumption

Accuracy: It is the measure of how well the model detects true positives and true negatives after being trained. A confusion matrix shown in Figure 5 and Equation 3 shows the proportion of total correct predictions (both true positives and true negatives) out of all predictions made.

$$Accuracy = \frac{TP + TN}{TP+TN+FP+FN} \quad (3)$$

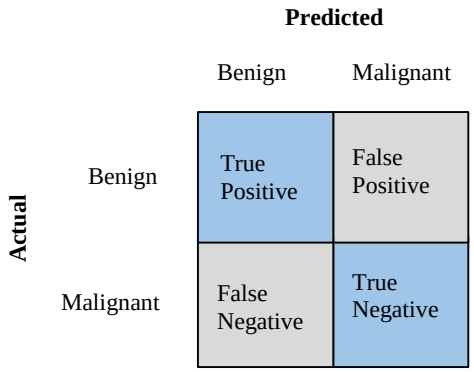


Fig. 5 Confusion matrix

Execution Time: It is the time taken by any of the two techniques to execute the process of optimization.

Memory consumption: it's the memory storage consumed while the training process is running.

3. Results and Discussion

3.1. Experimental Setup

All the experiments were implemented on Google Colab, Pro environment. The hardware configuration used was an Intel Xeon CPU operating at 2.20 GHz with 25 GB of RAM. The code was implemented using Python 3.10 with extra libraries as TensorFlow 2.12, Karas 2.12, NumPy 1.24, Pandas 1.5, and scikit-learn 1.2. As for the parallel implementation of PSO, Python's multiprocessing library was used, specifically the Pool (processes = n) function, where the number of processes was set equal to the swarm size.

The proposed model used the ISIC-2018 dataset with benign and malignant images. It has been used as a benchmark to measure the performance of sequential and parallel PSO implementations. The experiments were conducted on a stratified subset consisting of 300 images (benign and malignant) for training and 100 images for testing.

The MobileNet model used during the training process had specific hyperparameters kept constant across all experimental iterations. In contrast, other hyperparameters were identified for optimization by the Particle Swarm Optimization (PSO) algorithm, as previously specified in Table 2.

The first experimental setup, the sequential PSO, started with a swarm size set to 2 and increased to 6. During each phase, the number of iterations ranged from 2 to 10 with a step increase of 2. For fair comparison, the identical setup was delivered to execute the parallel PSO technique.

3.2. Results

3.2.1. Accuracy Measure

The techniques implemented both have maintained classification accuracy between 95% and 97%. Although the main aim of the research was to investigate the effect of using the sequential over parallel PSO technique on performance measure, maintaining a reasonable accuracy value was also essential. This was considered alongside studies that employed PSO-based hyperparameter optimization for convolutional neural networks in medical imaging applications.

Upon comparisons with previous studies, CNN optimization driven by PSO has reported classification accuracies in the range of approximately 93–96% across various image classification tasks [13, 14]. When applying multiobjective PSO to breast cancer diagnosis, an accuracy of around 95%-96% was achieved [16]. However, hyperparameter optimization using PSO for mammography classification reported accuracies exceeding 97% under controlled experimental conditions [17]. These results confirm that maintaining classification performance within the 95% to 97% is consistent with existing literature and further supports the robustness of the proposed approach.

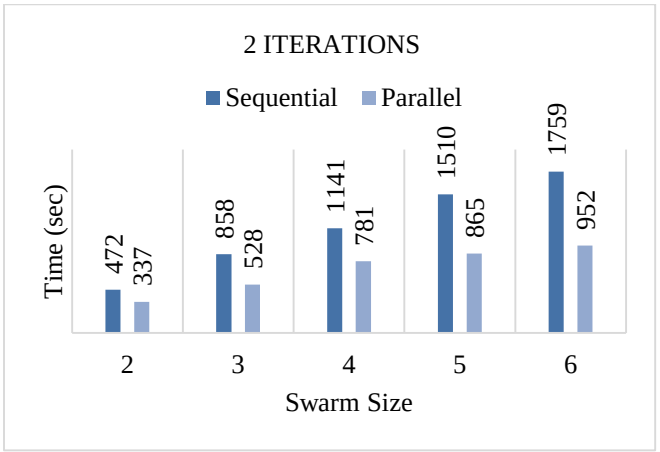
3.2.2. Execution Time Measure

To measure the performance of the sequential and parallel techniques, execution time was measured on the PSO phase rather than the whole code. The phases, such as image loading and library imports, were common to both approaches and hence were not taken into consideration in the measurement process.

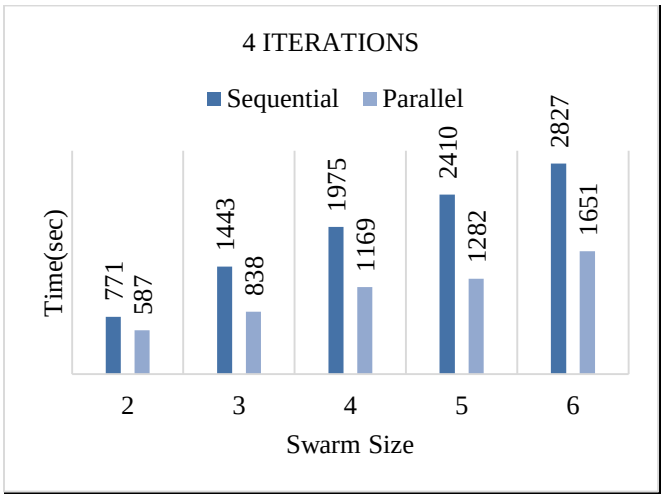
The execution time for each technique was measured and presented twice. The first experimental setup was made by fixing the swarm size values between 2 and 6 while trying different Numbers Of Iterations (M). This is shown in Figure 6. Figure 7 shows the second experimental setup, where the number of iterations is set to be from 2 to 10 while changing the swarm size. These setups were designed to measure the impact of each factor on execution time.

Figure 6 illustrates the effect of increasing swarm size on the execution time per iteration. Across all subfigures in Figure 6 (a)-(e), a speedup was observed when comparing the sequential and parallel techniques. It was also observed that the magnitude of the speedup varied.

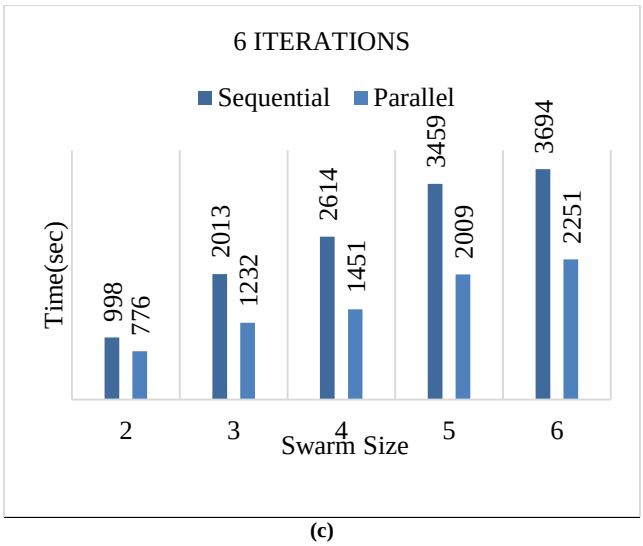
Upon comparing Figure 6 (a) (2 iterations) up to Figure 6 (c) (6 iterations), it is evident that, for a given number of iterations, the speedup increased with larger swarm sizes. However, Figure 6 (d) showed a relatively stable speedup rate regardless of swarm size. This trend was inverted at 10 iterations, Figure 6 (e), where the speedup rate decreased as the swarm size increased.



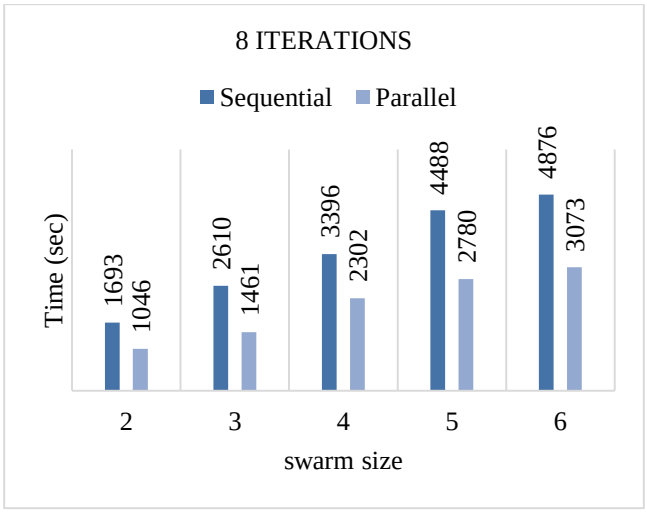
(a)



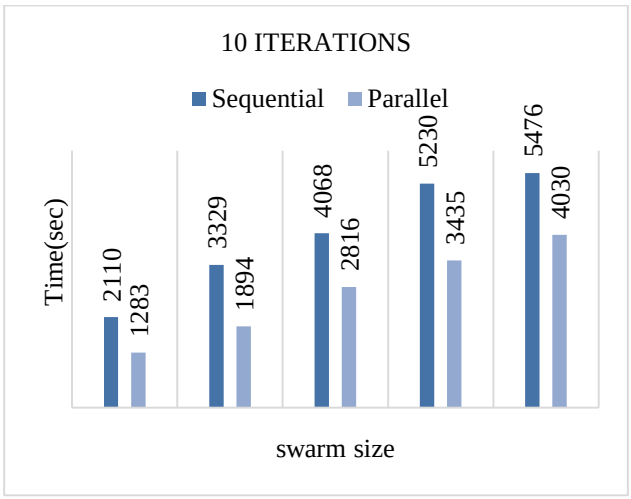
(b)



(c)



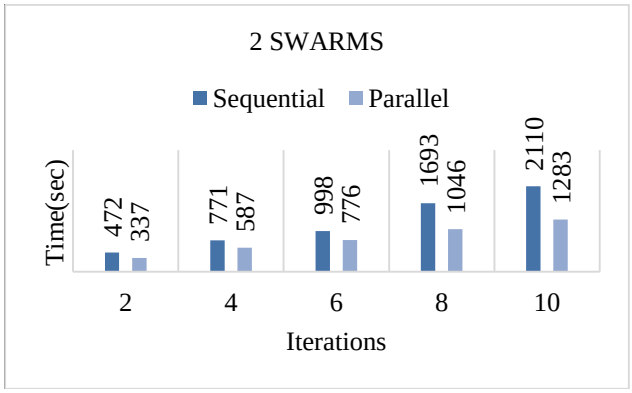
(d)



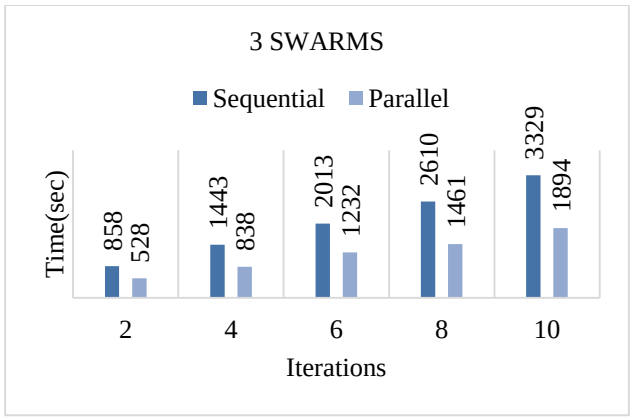
(e)

Fig. 6 Effect of the change in iterations over time with varying the swarm size from 2 till 8: (a) 2 Iterations, (b) 4 Iterations, (c) 6 Iterations, (d) 8 Iterations, and (e) 10 Iterations.

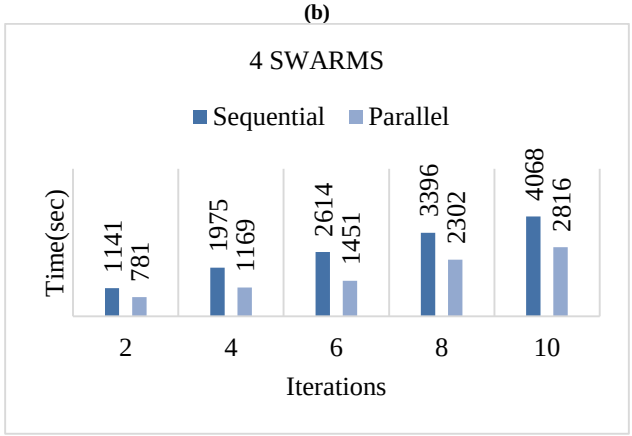
As depicted in Figure 7, an analysis of swarm size changes over iterations revealed consistent speedup improvements with increasing iterations for swarm sizes 2, 3, and 4 (Figure 7 (a-c)). In contrast, a swarm size of 5 showed a decreasing speed increase rate after the seventh iteration, while a swarm size of 6 in Figure 7 (e) exhibited a decrease in speed up from the beginning. An analysis of swarm size changes over iterations revealed consistent speedup improvements with increasing iterations for swarm sizes 2, 3, and 4. In contrast, a swarm size of 5 showed a decreasing speed increase rate after the seventh iteration, while a swarm size of 6 exhibited a decrease in speedup from the very beginning.



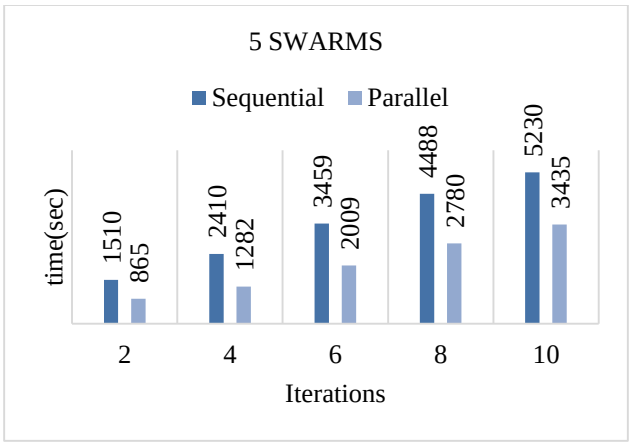
(a)



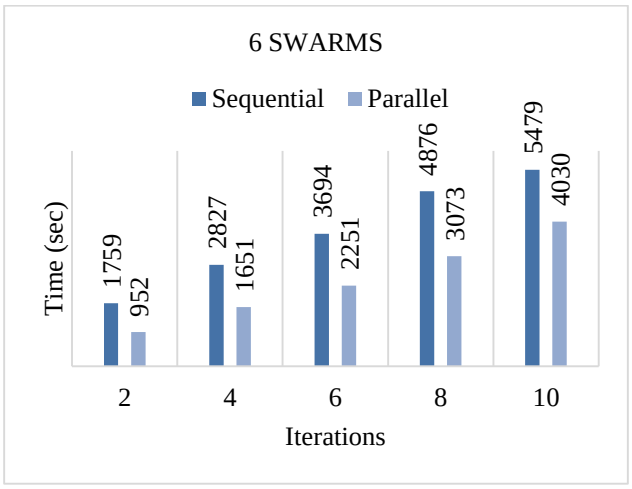
(b)



(c)



(d)



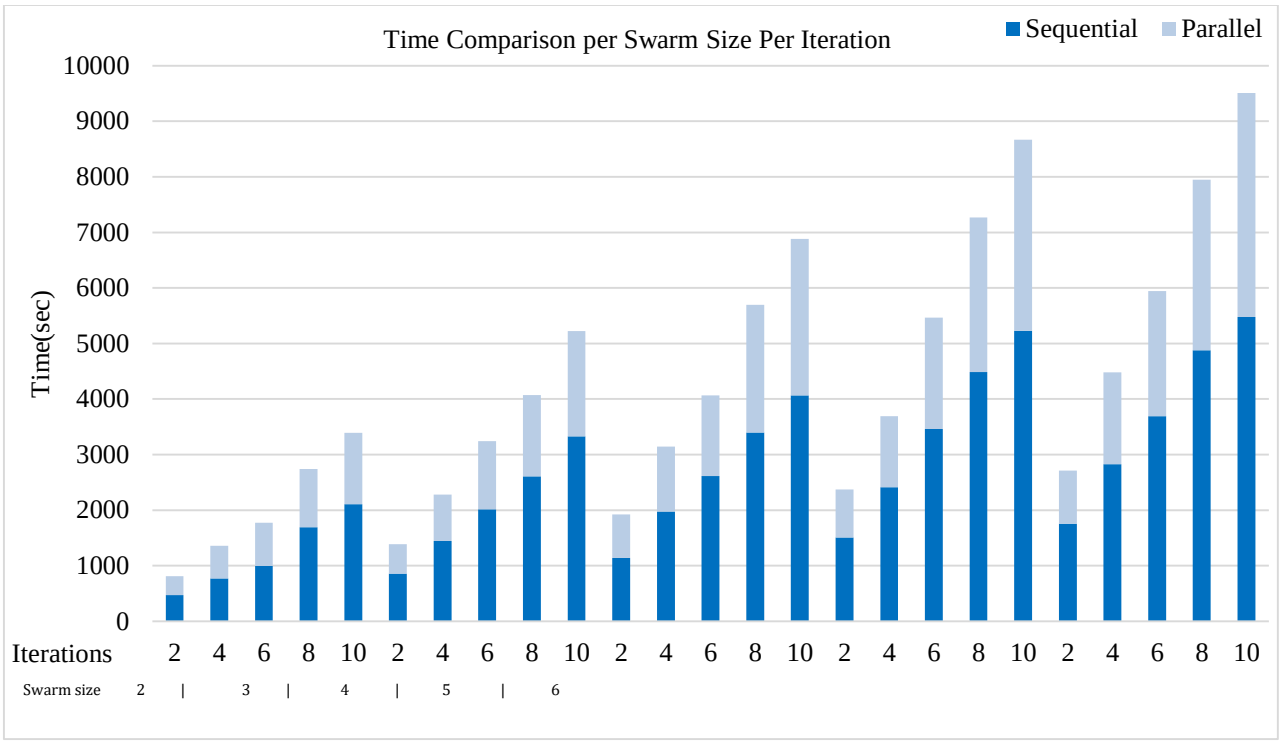
(e)

Fig. 7 Effect of the change in swarm size over time with varying the number of iterations from 2 till 10: (a) 2 Swarms, (b) 3 Swarms, (c) 4 Swarms, (d) 5 Swarms, and (e) 6 Swarms.

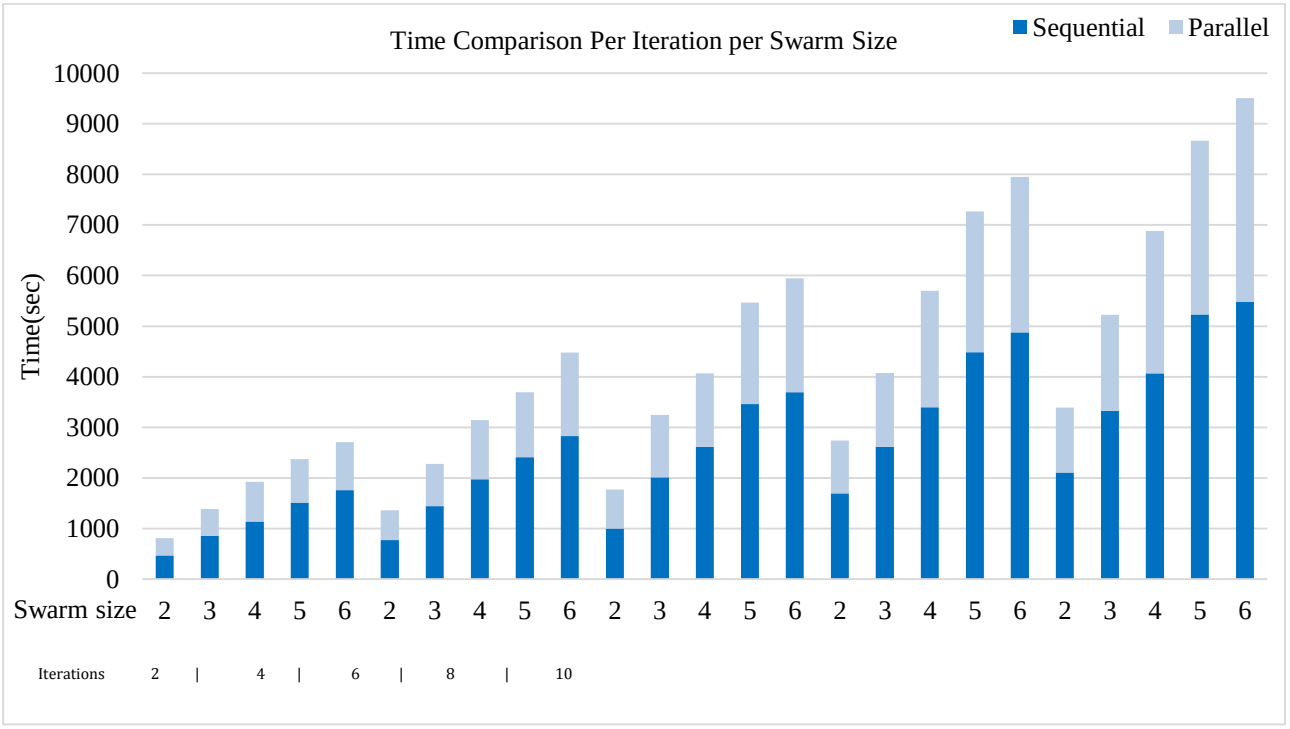
Upon analyzing and summarizing the effect on the execution time of both techniques in varying the Size of the swarm and the number of iterations, the results are presented concisely in Figure 8.

Figure 8 (a) shows the effect of increasing the swarm size on the execution time taken for both techniques. Whereas Figure 8 (b) reflects the same measure upon the changes in iterations. Upon analyzing the values depicted in Figures 6 and 7, the analysis revealed a time speedup of 26% to 45%.

Figure 8 (a) shows that the parallel technique had shorter execution times than the sequential technique, despite the swarm size. The sequential technique's execution time increased at a faster rate as the swarm size grew compared to the parallel technique. The same observation regarding the execution time was made when studying the effect of increasing the number of iterations, as depicted in Figure 8 (b).



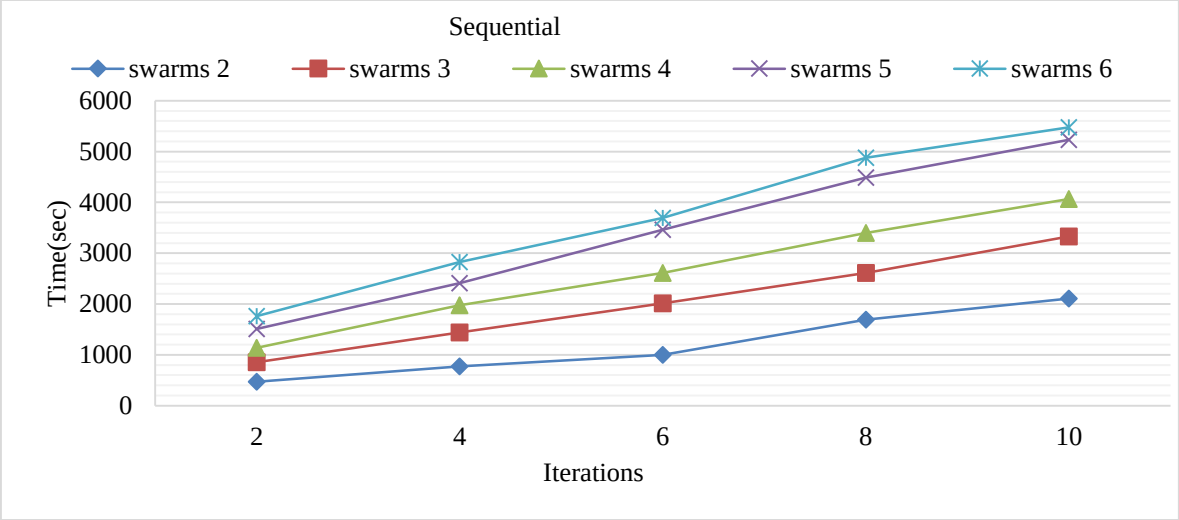
(a)



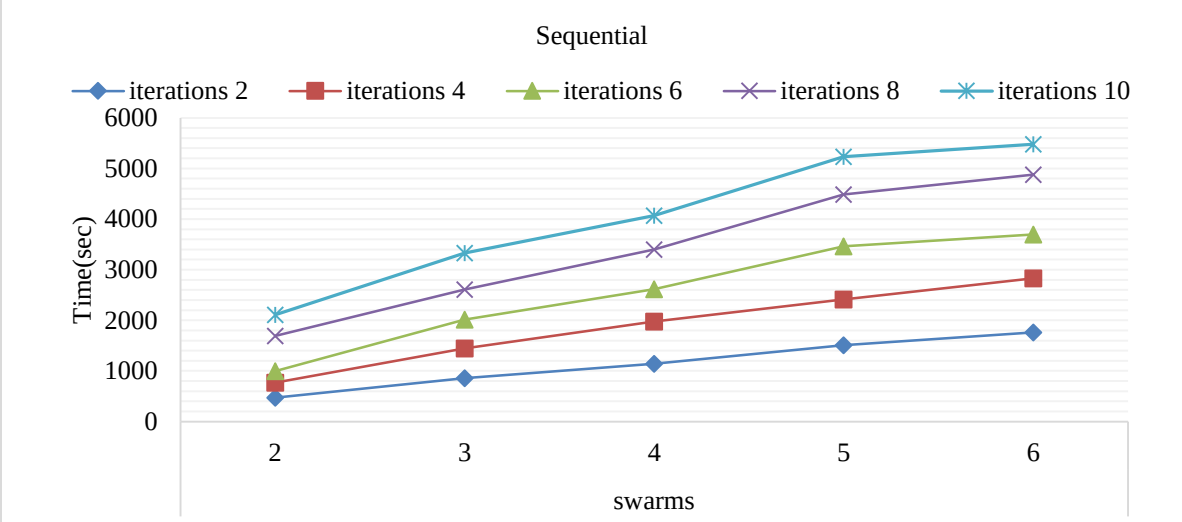
(b)

Fig. 8 Summary of time consumption (a) Relevant to the change in iterations per swarm size, and (b) Relevant to the change in swarm size per iteration.

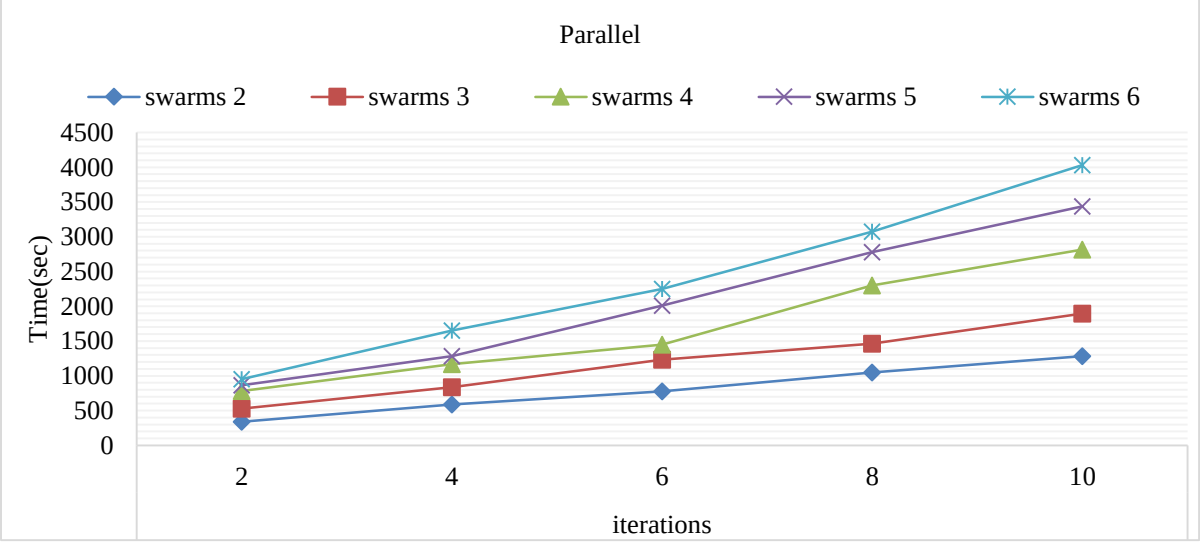
To provide a more comprehensive analysis, the rate of increase in time, derived from the previous values, was also examined and delivered in Figure 9. The figures highlighted that the parallel method's rate of increase in execution time was lower than that of the sequential method.


Sequential
swarms 2
swarms 3
swarms 4
swarms 5
swarms 6
Time(sec)
6000
5000
4000
3000
2000
1000
0
2
4
6
8
10
Iterations

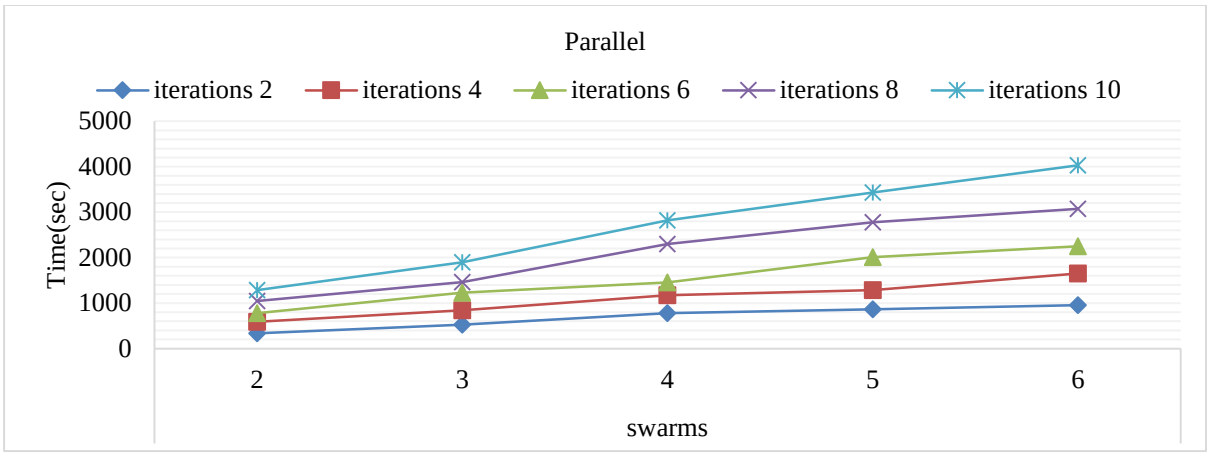
(a)


Sequential
iterations 2
iterations 4
iterations 6
iterations 8
iterations 10
Time(sec)
6000
5000
4000
3000
2000
1000
0
2
3
4
5
6
swarms

(b)


Parallel
swarms 2
swarms 3
swarms 4
swarms 5
swarms 6
Time(sec)
4500
4000
3500
3000
2500
2000
1500
1000
500
0
2
4
6
8
10
iterations

(c)

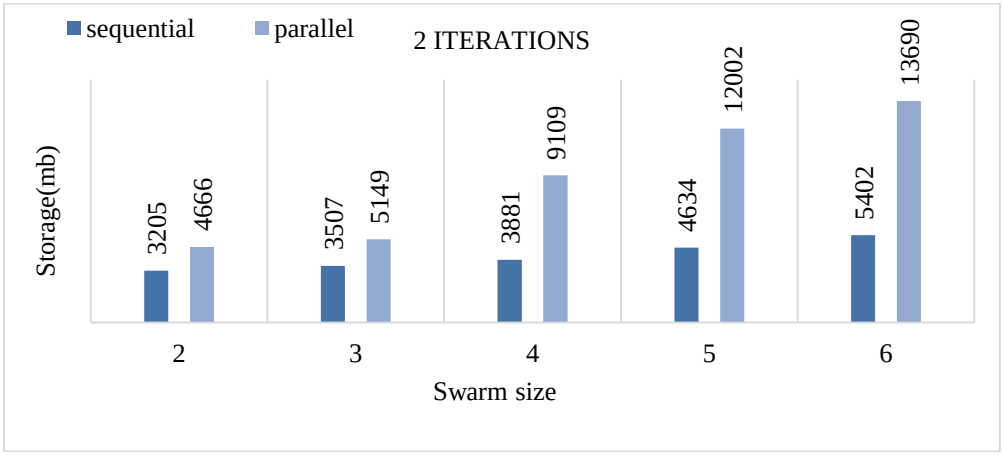


(d)

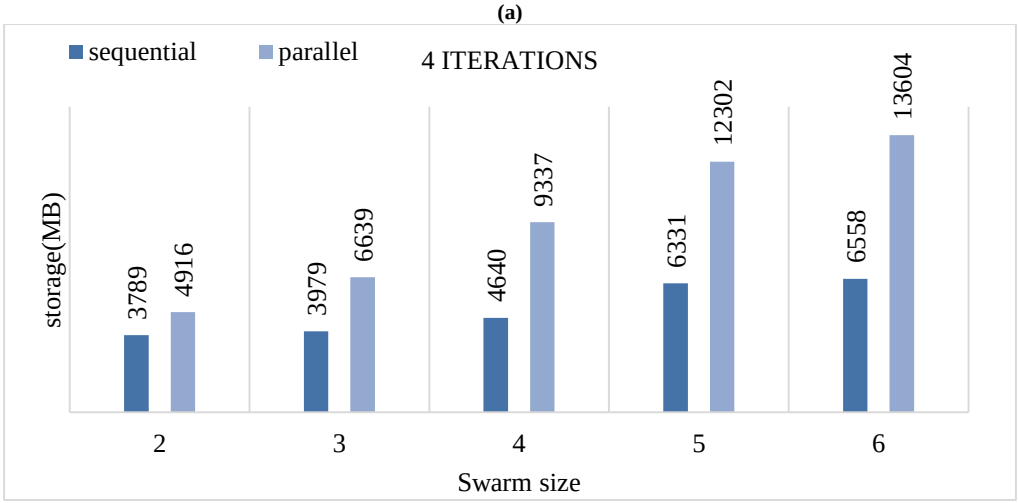
Fig. 9 Rate of time increase, (a) Relevant to the change in iterations per swarm size for sequential PSO, (b) Relevant to the change in swarm size per iteration for sequential PSO, (c) Relevant to the change in iterations per swarm size for parallel PSO, and (d) Relevant to the change in swarm size per iteration for parallel PSO.

3.2.3. Memory Consumption Measure

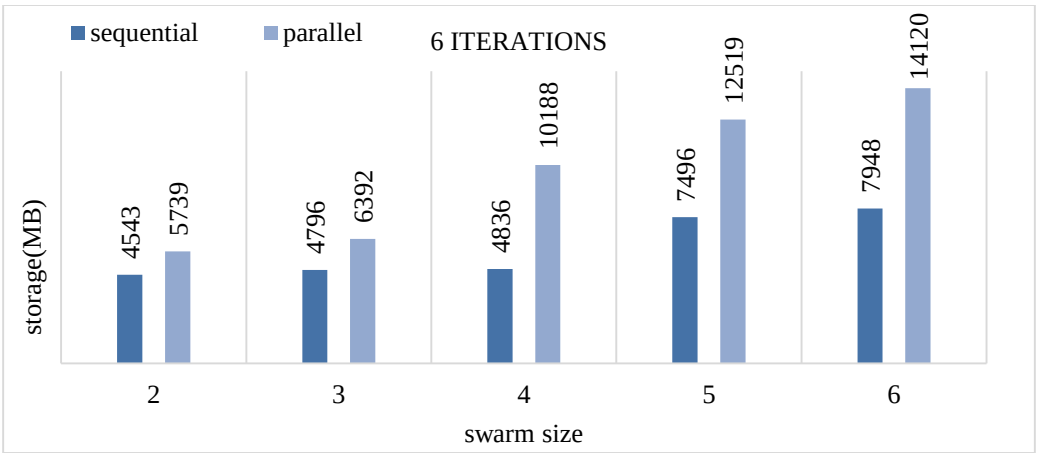
Another performance measure was the memory consumption of both sequential and parallel PSO techniques. Similarly, the standard processes, such as image loading and library imports, were excluded from this measure too. Hence, the comparison of memory consumption was measured while executing the PSO algorithm only. This was common while running the sequential or the parallel technique.



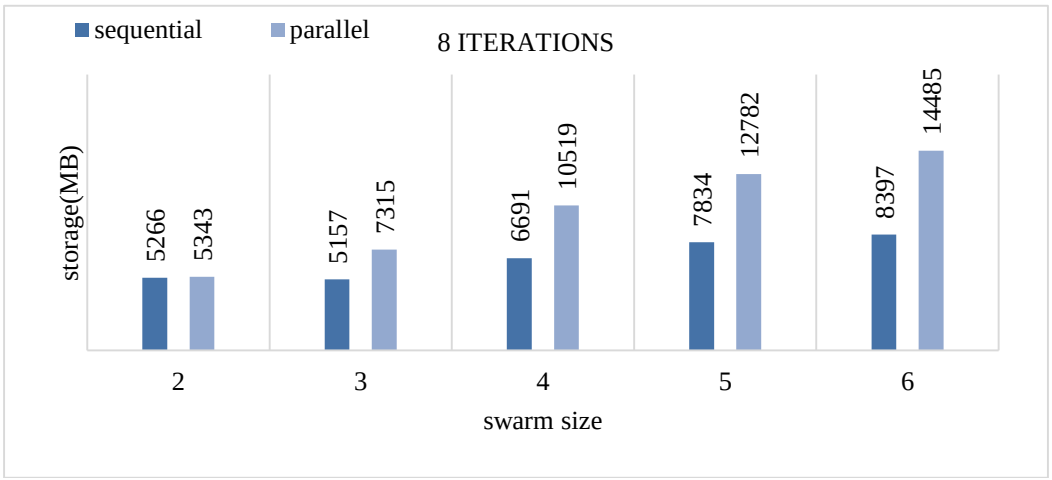
(a)



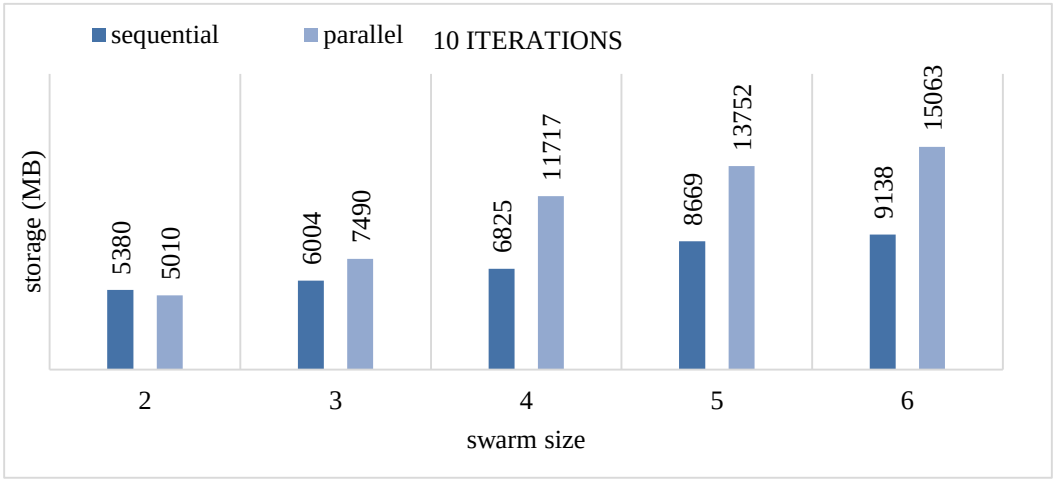
(b)



(c)



(d)



(e)

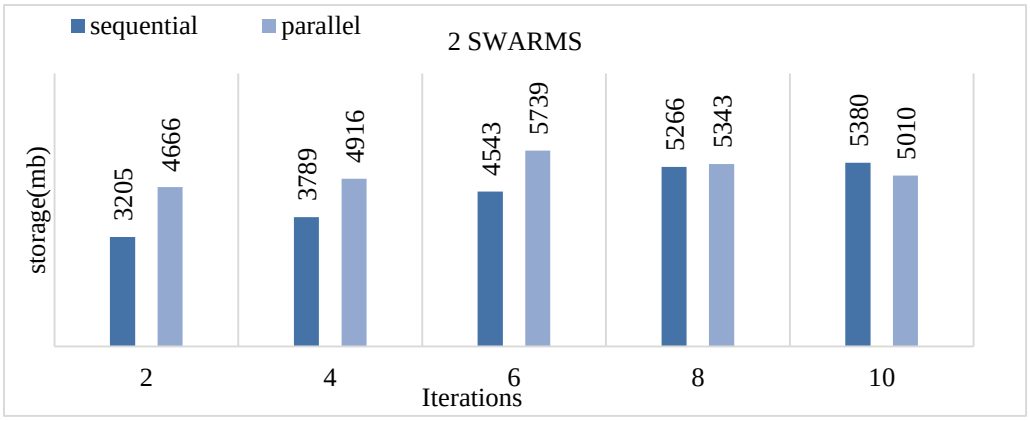
Fig. 10 Effect of the change in iterations over storage with varying the swarm size from 2 till 8: (a) Iterations = 2, (b) Iterations = 4, (c) Iterations = 6, (d) Iterations = 8, and (e) Iterations = 10.

Using the same setup mentioned to measure the execution time, the swarm size, and the number of iterations, the memory consumption was measured and analyzed. The effect of varying the number of iterations on sequential and parallel techniques is presented in Figure 10, while Figure 11 shows the impact of altering the Size of the swarm.

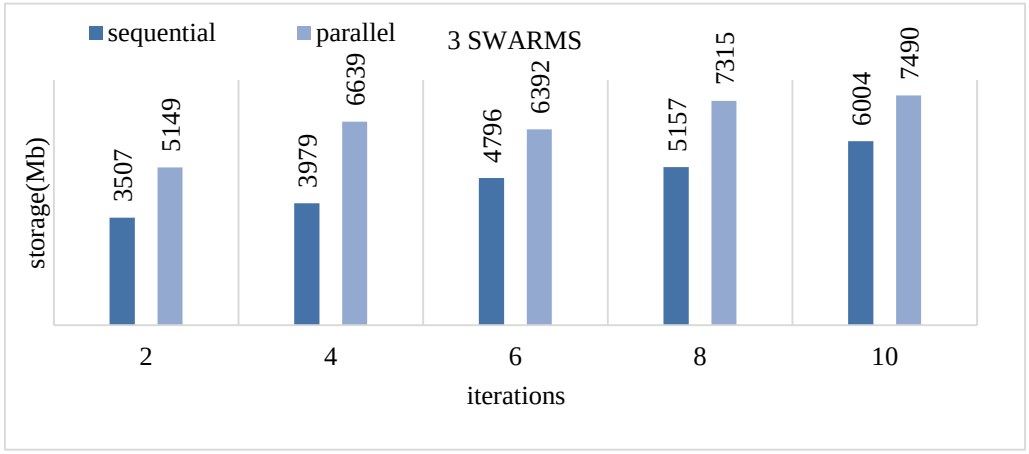
As illustrated in Figure 10 (a)-(e), the parallel technique exhibited substantially greater memory consumption (in MB) than its sequential counterpart, regardless of the iteration count.

Upon closer examination of Figure 10 (a)-(e) at iteration 2 through iteration 10, it is evident that the parallel technique's memory consumption grew significantly faster than the sequential techniques.

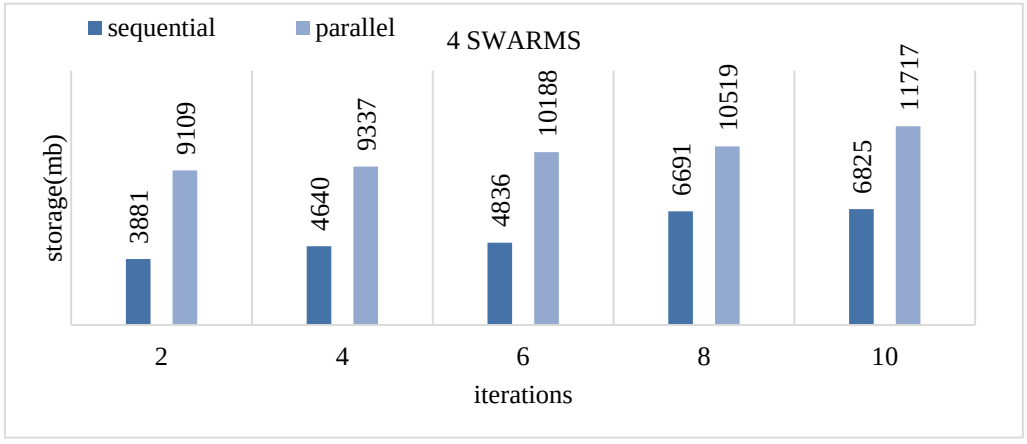
Conversely, another way to interpret the results is to study the memory consumption increase with more iterations within the same swarm size. This is clearly shown in Figure 11 (a) through Figure 11 (e). Despite an increase in memory consumption per swarm size, the gap between sequential and parallel techniques is narrow for small swarms (2 or 3). However, this gap becomes wider as the swarm size increases.



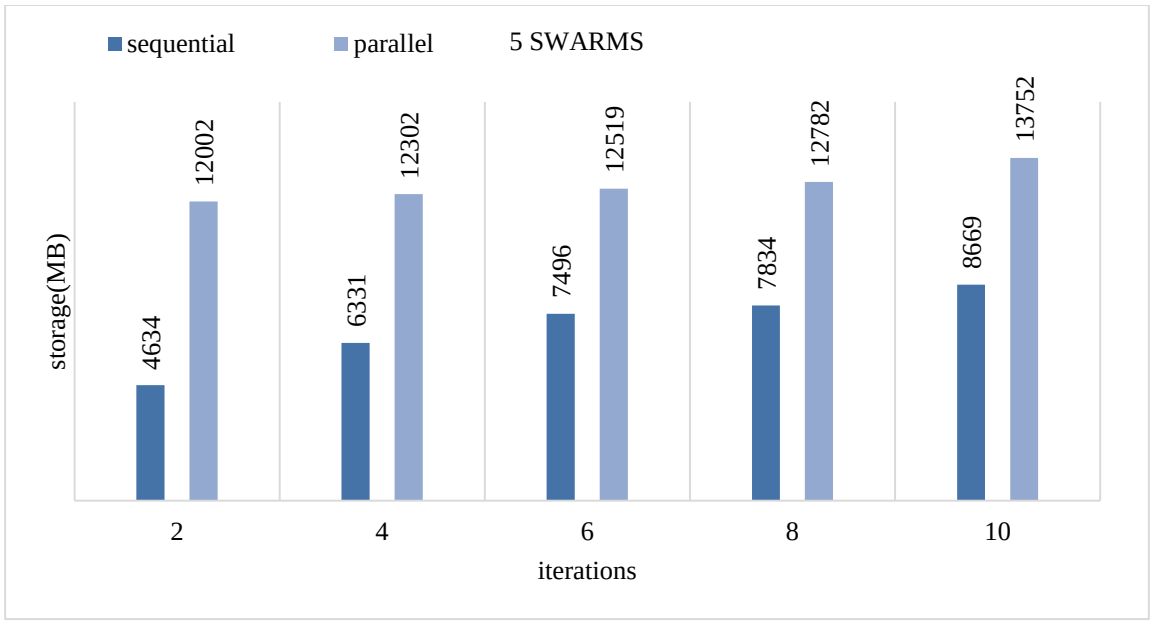
(a)



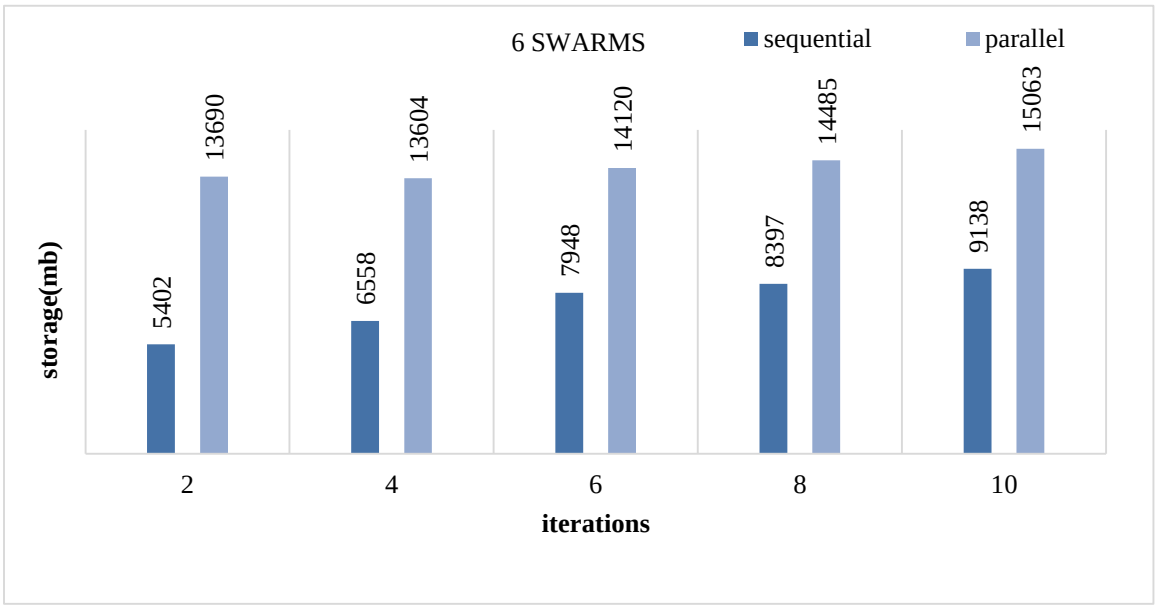
(b)



(c)



(d)

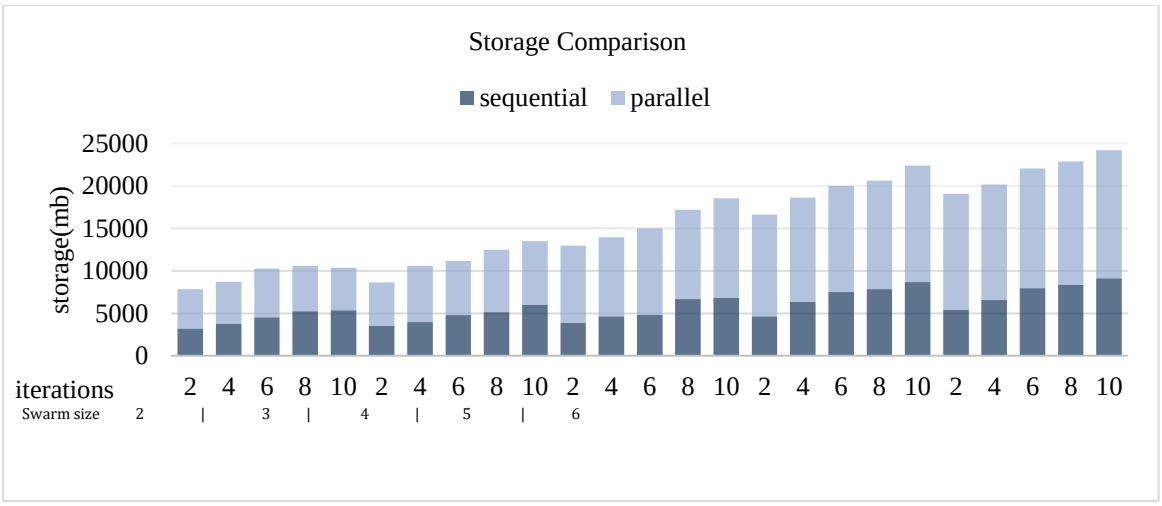


(e)

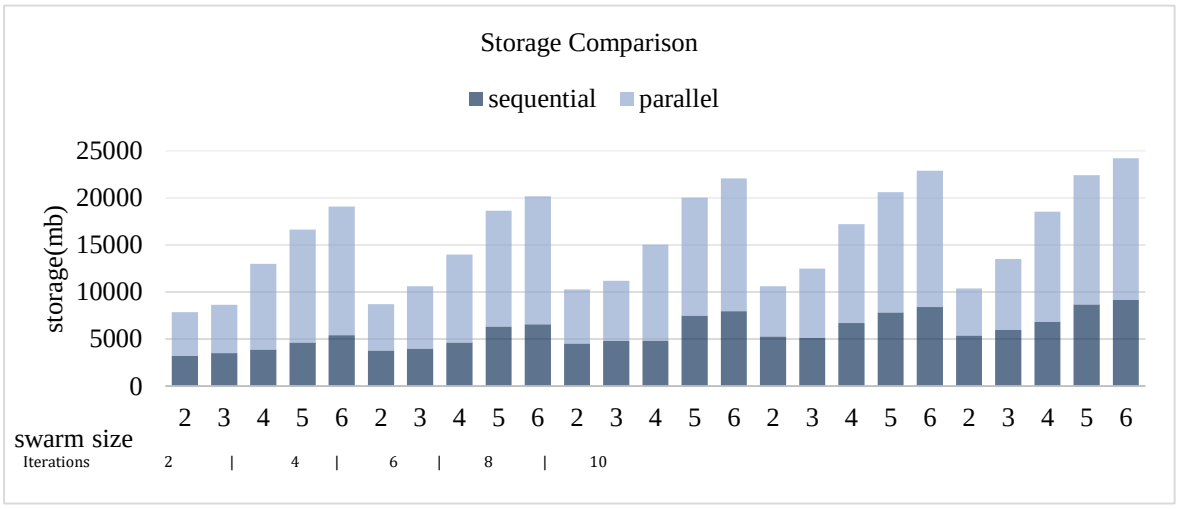
Fig. 11 Effect of the change in swarm size over memory consumption with varying the number of iterations from 2 till 10, (a) Swarm Size = 2, (b) Swarm Size = 3, (c) Swarm Size = 4, (d) Swarm Size = 5, and (e) Swarm Size = 6.

For a clearer view and easier comparisons, Figure 12 presents a summary of memory consumption for both sequential and parallel techniques, with variations in swarm size and the number of iterations. This consolidated figure allows for a direct visual assessment of how each factor influences the memory usage of the two approaches. These figures show that the parallel technique consumes more storage than the sequential technique (between 20 % to 150 % more). Figure 12 (a) and Figure 12 (b) also illustrate that memory consumption for the parallel technique increases with swarm size, with a greater rate of growth than the sequential technique.

To summarize the results, the rate at which memory consumption increased was also examined and depicted in Figure 13 in the form of a graph. Figure 13(a) and (b) serve as proof of concept. These figures show that the rate of increase in the sequential technique was relatively low despite the Change in swarm size and number of iterations. The parallel technique, unlike the sequential one, showed an increase in memory consumption as the swarm size grew, with this behavior being consistent regardless of the number of iterations (Figure 13 (c), Figure 13 (d)).

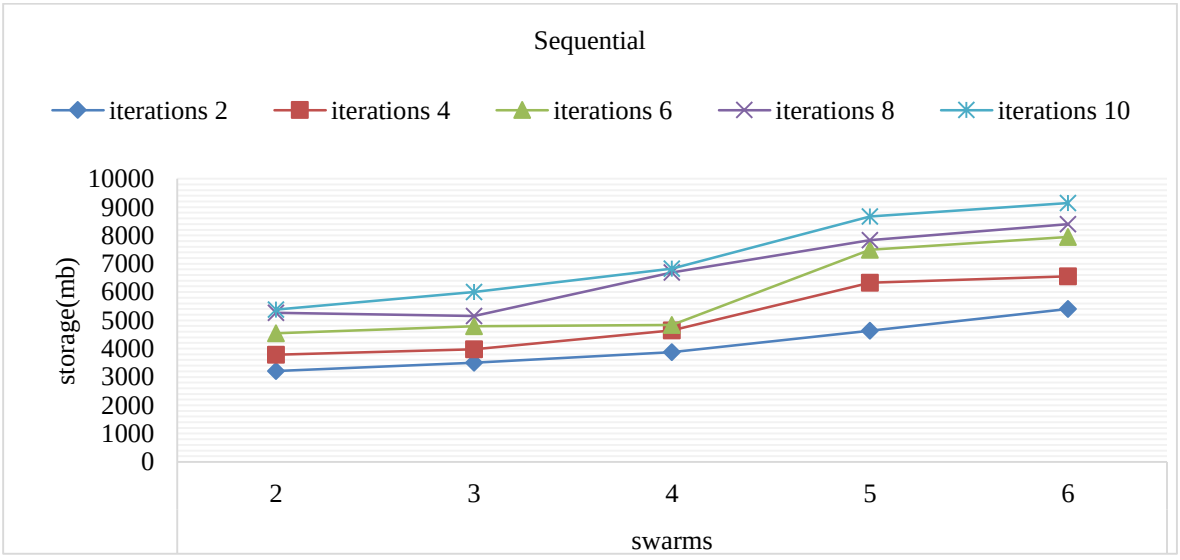


(a)

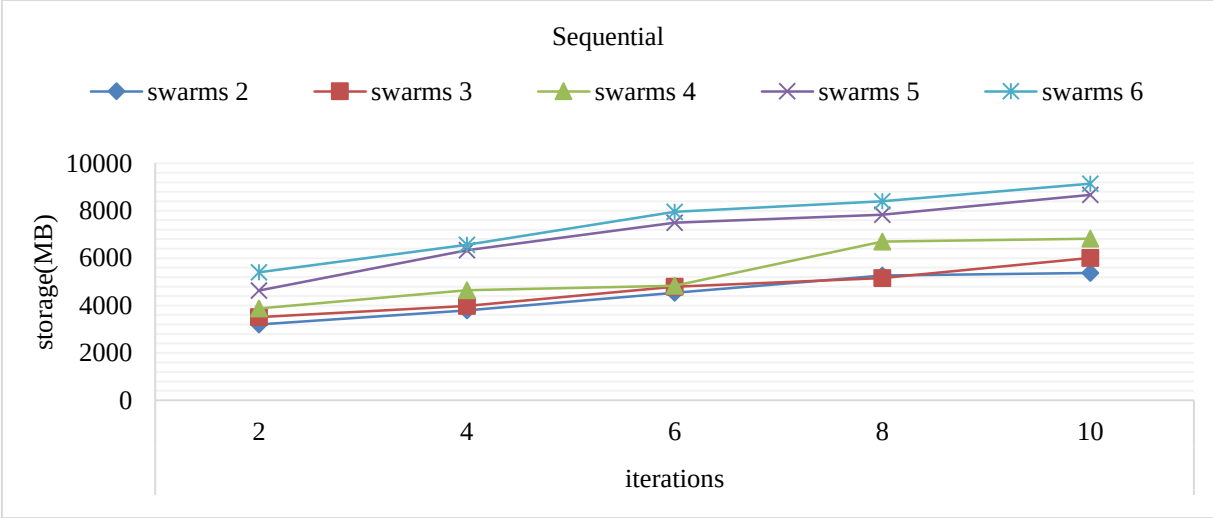


(b)

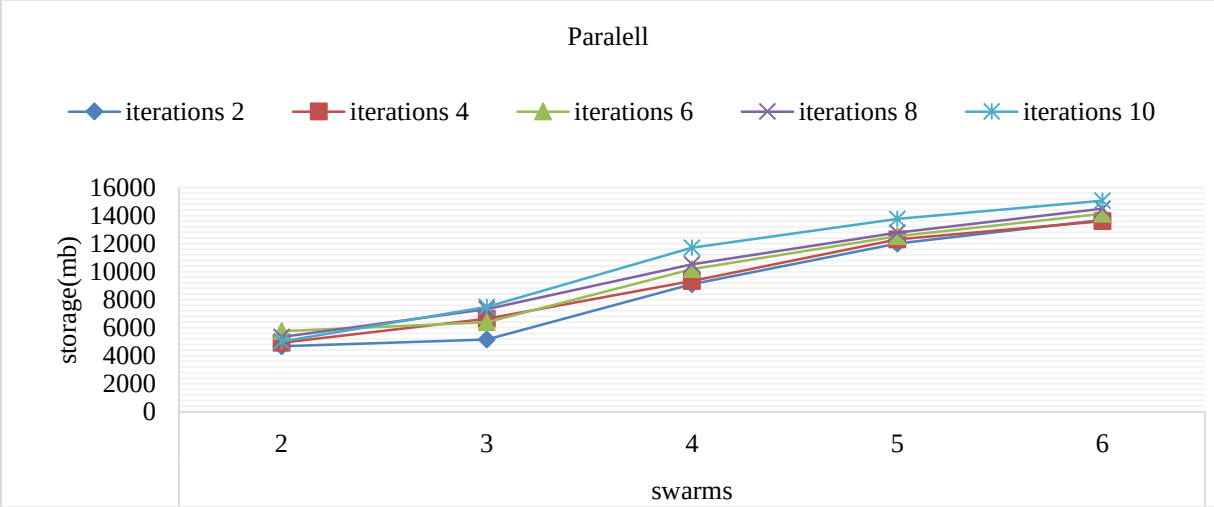
Fig. 12 Summary of memory consumption, (a) Relevant to the change in iterations per swarm size, (b) Relevant to the change in swarm size per iteration.



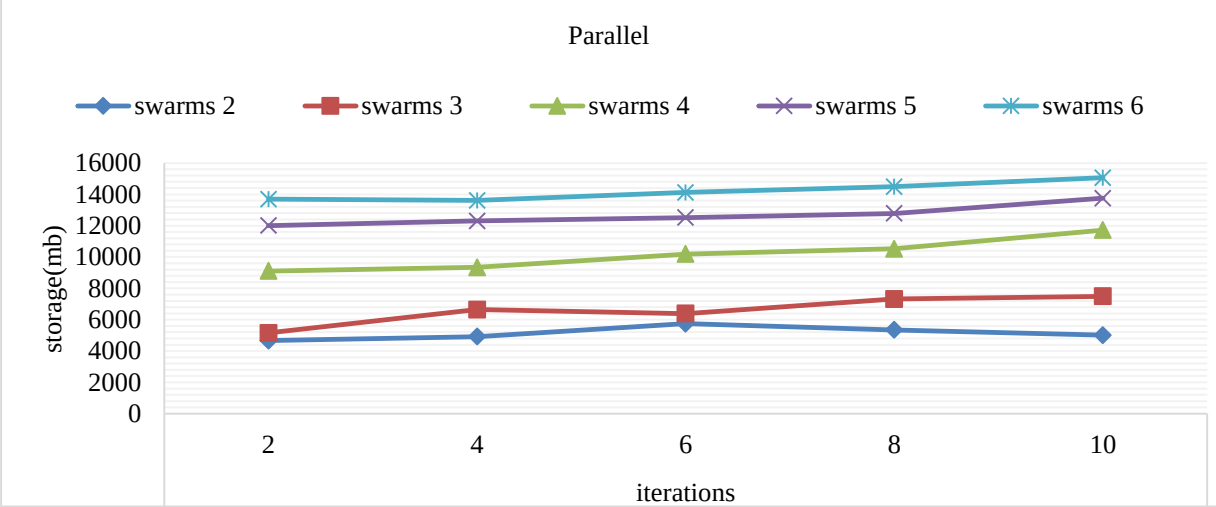
(a)



(b)



(c)



(d)

Fig. 13 Rate of memory consumption increase, (a) Relevant to the change in iterations per swarm size for sequential PSO, (b) Relevant to the change in swarm size per iteration for sequential PSO, (c) Relevant to the change in iterations per swarm size for parallel PSO, and (d) Relevant to the change in swarm size per iteration for parallel PSO.

4. Conclusion

This research compared the sequential and parallel implementations of Particle Swarm Optimization (PSO) for hyperparameter tuning of a MobileNet model for skin cancer classification. Regarding accuracy, both techniques achieved values between 95% and 97%. The parallel PSO technique reduced execution time by up to 45% but with increased memory consumption. These findings highlight the trade-off between execution speed and resource usage. This is essential for selecting the appropriate optimization strategies in resource-constrained environments. The two techniques were evaluated by varying the swarm size and the number of iterations to examine their effects on accuracy, execution time, and memory consumption.

The results showed that when using 2 through 6 iterations, the achieved speedup increased with larger swarm sizes. However, when 8 iterations were used, the speedup rate remained stable despite the swarm size. This was not the case when using 10 iterations, where the speedup rate decreased as the swarm size increased.

This concludes the trade-off between parallelization efficiency and associated overhead. Up to 8 iterations, parallel execution effectively distributes particle evaluations, resulting in reduced total runtime compared to sequential PSO. However, at 10 iterations, the additional overhead related to process management, synchronization, and memory allocation in the parallel setting begins to accumulate. As the number of iterations increases, this overhead grows unevenly, canceling part of the parallelization gains and leading to a reduction in overall speedup. In other words, although parallel PSO accelerates computation, beyond a certain threshold, the cost of managing multiple processes outweighs the benefits, which explains the reduced performance improvement observed at 10 iterations.

In contrast, the rate of memory consumption increase was examined as a function of the number of iterations within a fixed swarm size. As clearly shown, the parallel technique required significantly more memory with each additional iteration as the swarm size increased. Although the sequential technique's memory usage rate also rose with increasing swarm size, it consistently demanded less memory than the parallel technique.

This concluded that both the Size of the swarm and the number of iterations directly affected the execution time and the memory consumed. Nevertheless, since the primary objective was to study the execution time, a trade-off emerged in the form of increased memory requirements to accommodate the execution of the parallel technique. However, to pinpoint the exact breaking point where the rate of speedup in time begins to trade off against memory consumption, future work must include larger swarm sizes and more iterations.

Future work will extend this study in two primary directions. First, the proposed framework will be deployed in GPU-enabled environments to evaluate performance improvements relative to CPU-based implementations. Since GPU acceleration has been shown to reduce computation time for metaheuristic optimization in Deep Learning significantly, such an evaluation will provide a more comprehensive understanding of the scalability and practicality of parallel PSO in real-world medical AI applications. Second, the optimization process will be expanded to include additional hyperparameters beyond batch size, optimizer type, and dense layer size, such as learning rate schedules, dropout rates, and width multipliers. This broader search space is expected to yield further improvements in model robustness and generalization, thereby enhancing the clinical applicability of MobileNet-based skin cancer detection systems.

References

[1] Maryam Tahir et al., "DSCC_Net: Multi-Classification Deep Learning Models for Diagnosing of Skin Cancer Using Dermoscopic Images," *Cancers*, vol. 15, no. 7, pp. 1-28, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[2] Nasser A. AlSadhan et al., "Skin Cancer Recognition Using Unified Deep Convolutional Neural Networks," *Cancers*, vol. 16, no. 7, pp. 1-16, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[3] Turker Tuncer et al., "A Lightweight Deep Convolutional Neural Network Model for Skin Cancer Image Classification," *Applied Soft Computing*, vol. 162, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[4] Hebin Cheng, Jian Lian, and Wanzhen Jiao, "Enhanced MobileNet for Skin Cancer Image Classification with Fused Spatial Channel Attention Mechanism," *Scientific Reports*, vol. 14, pp. 1-13, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[5] Su Myat Thwin, and Hyun-Seok Park, "Skin Lesion Classification using a Deep Ensemble Model," *Applied Sciences*, vol. 14, no. 13, pp. 1-17, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[6] Alaa S. Al-Waisy et al., "A Deep Learning Framework for Automated Early Diagnosis and Classification of Skin Cancer Lesions in Dermoscopy Images," *Scientific Reports*, vol. 15, pp. 1-20, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[7] Rizwan Ali et al., "A Novel SpaSA based Hyper-Parameter Optimized FCEDN with Adaptive CNN Classification for Skin Cancer Detection," *Scientific Reports*, vol. 14, pp. 1-17, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[8] Shiwei Liu et al., "Multi-branch CNN and Grouping Cascade Attention for Medical Image Classification," *Scientific Reports*, vol. 14, pp. 1-15, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[9] Naveed Ahmad et al., "A Novel Framework of Multiclass Skin Lesion Recognition from Dermoscopic Images using Deep Learning and Explainable AI," *Frontiers in Oncology*, vol. 13, pp. 1-17, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[10] Taye Girma Debelee, "Skin Lesion Classification and Detection Using Machine Learning Techniques: A Systematic Review," *Diagnostics*, vol. 13, no. 19, pp. 1-40, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[11] Abdelkader Alrabai, Amira Echtioui, and Fathi Kallel, "Explainable Deep Learning Approaches for Skin Cancer Diagnosis," *Network Modeling Analysis in Health Informatics and Bioinformatics*, vol. 14, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[12] Sevda Gül et al., "YOLOSAMIC: A Hybrid Approach to Skin Cancer Segmentation with the Segment Anything Model and YOLOv8," *Diagnostics*, vol. 15, no. 4, pp. 1-26, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[13] Muhammad Munsarif, Muhammad Sam'an, and Andrian Fahrezi, "Convolution Neural Network Hyperparameter Optimization using Modified Particle Swarm Optimization," *Bulletin of Electrical Engineering and Informatics*, vol. 13, no. 2, pp. 1268-1275, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[14] Özkan Inik, "SwarmCNN: An Efficient Method for CNN Hyperparameter Optimization using PSO and ABC Metaheuristic Algorithms," *The Journal of Supercomputing*, vol. 81, pp. 1-42, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[15] Zhengping Liang et al., "A Multi-Objective Multi-Task Particle Swarm Optimization based on Objective Space Division and Adaptive Transfer," *Expert Systems with Applications*, vol. 255, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[16] S. Manimurugan et al., "Breast Cancer Diagnosis Model using Stacked Autoencoder with Particle Swarm Optimization," *Ain Shams Engineering Journal*, vol. 15, no. 6, pp. 1-12, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[17] Khadija Aguerchi et al., "A CNN Hyperparameters Optimization based on Particle Swarm Optimization for Mammography Breast Cancer Classification," *Journal of Imaging*, vol. 10, no. 2, pp. 1-17, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[18] Merve Korkmaz, and Kaplan Kaplan, "Effectiveness Analysis of Deep Learning Methods for Breast Cancer Diagnosis based on Histopathology Images," *Applied Sciences*, vol. 15, no. 3, pp. 1-25, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[19] Tamir Shaqarin, and Bernd R. Noack, "A Fast-Converging Particle Swarm Optimization through Targeted, Position-Mutated, Elitism (PSO-TPME)," *International Journal of Computational Intelligence Systems*, vol. 16, pp. 1-17, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[20] Yiying Zhang, "Elite Archives-Driven Particle Swarm Optimization for Large Scale Numerical Optimization and its Engineering Applications," *Swarm and Evolutionary Computation*, vol. 76, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[21] Huidong Ling et al, "A Parallel Multiobjective PSO Weighted Average Clustering Algorithm Based on Apache Spark," *Entropy*, vol. 25, no. 2, pp. 1-14, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[22] Sarah A. Alzakari et al., "LesionNet: An Automated Approach for Skin Lesion Classification using SIFT Features with Customized Convolutional Neural Network," *Frontiers in Medicine*, vol. 11, pp. 1-16, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[23] Shafiqul Islam et al., "Leveraging AI and Patient Metadata to Develop a Novel Risk Score for Skin Cancer Detection," *Scientific Reports*, vol. 14, pp. 1-12, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[24] Mario García-Valdez et al., "Distributed and Asynchronous Population-Based Optimization Applied to the Optimal Design of Fuzzy Controllers," *Symmetry*, vol. 15, no. 2, pp. 1-21, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[25] Ming Li et al., "A Parallel Particle Swarm Optimization Framework based on a Fork-Join Thread Pool using a Work-Stealing Mechanism," *Applied Soft Computing*, vol. 145, 2023. [CrossRef] [Google Scholar] [Publisher Link]