Indonesian Journal of Electrical Engineering and Computer Science
Vol. 35, No. 1, July 2024, pp. 620~630
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v35.i1.pp620-630

Multi-layer perceptron hyperparameter optimization using Jaya algorithm for disease classification

Andien Dwi Novika, Abba Suganda Girsang
Department of Computer Science, Master of Computer Science, Bina Nusantara University Jakarta, Jakarta, Indonesia

Article Info

Article history:

Received Feb 2, 2024
Revised Feb 27, 2024
Accepted Mar 26, 2024

Keywords:

Classification
Hyperparameter optimization
Jaya algorithm
Metaheuristic algorithm
Multilayer perceptrons

ABSTRACT

This study introduces an innovative hyperparameter optimization approach for enhancing multilayer perceptrons (MLP) using the Jaya algorithm. Addressing the crucial role of hyperparameter tuning in MLP's performance, the Jaya algorithm, inspired by social behavior, emerges as a promising optimization technique without algorithm-specific parameters. Systematic application of Jaya dynamically adjusts hyperparameter values, leading to notable improvements in convergence speeds and model generalization. Quantitatively, the Jaya algorithm consistently achieves convergences at first iteration, faster convergence compared to conventional methods, resulting in 7% higher accuracy levels on several datasets. This research contributes to hyperparameter optimization, offering a practical and effective solution for optimizing MLP in diverse applications, with implications for improved computational efficiency and model performance.

Corresponding Author:

Andien Dwi Novika
Department of Computer Science, Master of Computer Science, Bina Nusantara University Jakarta
Jakarta 11480, Indonesia
Email: andien.novika@binus.ac.id

1. INTRODUCTION

Artificial intelligence (AI) has become part of our lives. Some examples of AI that have been used daily are AI-personal assistant (i.e. Siri and Alexa), smart home, and smart car. AI has brought many conveniences to our lives. Machine learning, as a part of AI, has had massive growth since the first time it was found. Machine learning has been involved in several fields, such as health, engineering, and art. Data mining is the most machine learning application [1]. One of the main data mining techniques that is applied in many different fields is classification. Classification is used to predict group membership for data [2]. In its application, classification faced several problems and one of that is choosing the right value of hyperparameter. Hyperparameter refers to parameters that cannot be changed during machine learning training. Almost all machine learning algorithm has hyperparameter [3]. Hyperparameters significantly impact the performance of machine learning models and directly affect the processes of training algorithms [4]. Due to their impact on model performance and the uncertainty surrounding the ideal combination of hyperparameter, the configuration has become a crucial and challenging aspect in the application of machine learning algorithms [5].

Multilayer perceptron (MLP), a highly utilized machine learning method [6], falls under the category of artificial neural networks (ANNs). It encompasses a range of hyperparameters, including hidden layer quantity, neuron count, learning rate, activation function, optimizer, and batch size. Several studies have proved that MLP with hyperparameter optimization has better performance than MLP without hyperparameter optimization [5]-[9]. Hyperparameters could be tuned manually, but it has several difficulties, such as a lot of hyperparameters to be tuned, model complexity, and time-consuming, also it needs a deep understanding of machine learning usage and its hyperparameter value settings [10]. A metaheuristic algorithm is one of a way to resolve manual tuning hyperparameter problems. The term metaheuristic refers to a more complex heuristic that is suggested to solve a variety of optimization issues [11]. Some examples of metaheuristic algorithms are genetic algorithm (GA), artificial bee colony (ABC) algorithm, bat algorithm (BA), differential evolution (DE), whale optimization algorithm (WOA), firefly algorithm (FA), genetic pattern research (GPS), biogeography-based optimization (BBO), and Jaya algorithm.

Several studies have been conducted on hyperparameter optimization for MLP using metaheuristic algorithm, one of which was conducted by [6]. The study optimized MLP hyperparameters using GA and demonstrated significant improvements after hyperparameter optimization for MLP. Another study was conducted by [7]. The study optimized MLP hyperparameters using DE and showed better results compared to MLP without hyperparameter optimization. Dokeroglu *et al.* [12] also conducted research on hyperparameter optimization for MLP using particle swarm optimization (PSO) for regression. The study showed that hyperparameter optimization for MLP using PSO did not yield significant results compared to MLP without hyperparameter optimization.

Metaheuristics are a useful tool for effectively searching the search space for ideal or nearly ideal answers. Because of their tremendous efficiency, they are especially well-suited for hyperparameter optimization issues that include huge configuration spaces [13]. All aforesaid optimization algorithms have two parameters, and that is population size and number of generations. Apart from those two parameters, almost all metaheuristic algorithms have algorithm-specific parameters. For example, crossover probability, mutation probability, and selection operator in GA and initial weights in PSO. That algorithm-specific parameter needs to be adjusted correctly to achieve the best outcome. On the other hand, incorrect adjustment of these algorithm-specific parameters could lead to a rise in computing cost and local optima entrapment [14].

Jaya algorithm is a metaheuristic algorithm and applies convenience and simplicity. Unlike other metaheuristic algorithms, the Jaya algorithm does not have algorithm-specific parameters to obtain optimal results [15]. Jaya algorithm is believed to be able to reduce computing time since it is a parameter-less algorithm [16]. The Jaya algorithm was proposed by Yu *et al.* [17] and aroused interest from many parties. This is due to the characteristics of the Jaya algorithm which is easy to use [18]. Several studies have proved that Jaya algorithm has better performance than other metaheuristic method [14], [19]-[23].

In 2022, Alshutbi *et al.* [23] conducted a study on breast cancer classification using support vector machine (SVM) with the Jaya algorithm employed for feature selection optimization. The SVM-Jaya combination was compared with several other metaheuristic algorithms, including GA, CS, PSO, and DE. The results indicated that SVM-Jaya outperformed other metaheuristic algorithms in optimizing feature selection. Another study utilizing the Jaya algorithm was conducted by Wang *et al.* [21] in 2018. Their research demonstrated that Jaya, when used as a training algorithm, yielded higher accuracy compared to GA, PSO, GPS, BBO, ABC, and FA. These studies collectively suggest that Jaya produces superior results compared to other metaheuristic algorithms. However, there has been no research utilizing Jaya as an algorithm for hyperparameter optimization, particularly in MLP. Therefore, this study employs Jaya as the optimization algorithm for searching the optimal hyperparameters.

The Jaya algorithm can also be applied in fields other than computer science, such as engineering. One example is a study on power flow optimization conducted by [24], and research on surface grinding optimization by [25]. in the same year, which demonstrates that the Jaya algorithm can outperform other optimization methods. Goel *et al.* [26] also used Jaya algorithm for tuning of speed PI controller of DTC induction motor drive.

The objective of this paper is to design a model for MLP hyperparameter optimization using the Jaya algorithm (further referred to as MLP-Jaya) that focuses on low computational cost and without complex algorithm-specific adjustment. To assess the performance of the MLP-Jaya, disease classification testing will be conducted. This paper also provides a comprehensive analysis of several metaheuristic approaches, such as MLP-GA and MLP-PSO.

2. METHOD

The research begins with data collection followed by dataset preprocessing. Preprocessing involves label encoding and splitting the data into training, validation, and testing sets. The training and validation data will undergo hyperparameter optimization using the Jaya Algorithm. The outcome of this process is optimal hyperparameters, which will then be utilized in the testing phase. Subsequently, the testing results will be evaluated. The research stages are illustrated in Figure 1.

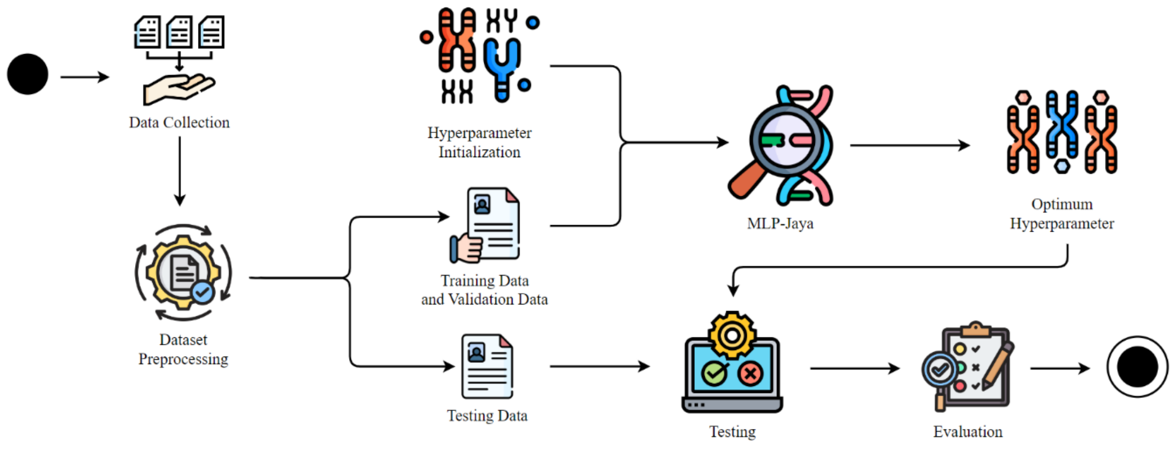

Figure 1. Research stages

2.1. Data collection

This research used 6 disease datasets. The datasets used in this research were all gathered for free from a variety of resources, such as Kaggle and UCI machine learning. The six datasets used in this study are provided in Table 1.

Table 1. Datasets

Code	Dataset	Number of attributes	Number of rows
D1	Breast cancer wisconsin (diagnostic)	30	569
D2	Parkinson	24	195
D3	Chronic kidney disease	26	400
D4	Diabetes	9	768
D5	Heart disease	14	1025
D6	Glioma (Tasci et al., 2022)	24	839

2.2. Data preprocessing

Collected datasets will go through three stages of pre-processing, namely imputation, label encoding and data splitting. Imputation is the process of filling in empty data in a dataset. Numerical data columns that have empty values will have the average value calculated which will then be used to fill in the empty data rows. Empty values in the categorical data column will be filled with the mode value in that column. Label encoding is the stage of changing a categorized data column into a numeric data column. The purpose of label encoding is so that the model can receive data because model can only accept numeric data. The last step is data splitting. The data is divided into two parts, namely training and testing. Training data will be used to train the model and will be used for hyperparameter optimization. Testing data will be used to assess the performance of the trained model. 70% of the dataset becomes training data, 20% of the dataset becomes testing data, and 10% of the dataset becomes validation data.

2.3. MLP-Jaya

MLP is a type of artificial neural network that consists of several layers and each layer consists of several neurons. MLP is a model that can be used for regression, classification, and pattern recognition [27]. MLP was developed by Rumelhalt *et al.* in 1986, MLP is a type of supervised learning and has a feedforward architecture [28]. MLP consists of three main layers, namely the input layer, hidden layer, and output layer. Each layer is connected to the layers before and after it through weight. The input layer consists of features from the dataset which will then be processed in the hidden layer. The results of hidden layer processing will be displayed in the output layer.

The Jaya algorithm, introduced by Rao [25] is a metaheuristic algorithm that blends the "survivability for the fittest" concept from evolutionary algorithms with the "follow the leader for searching the optimal solution" idea from swarm intelligence. Derived from Sanskrit, the term "Jaya" translates to victory, reflecting the algorithm's approach of converging toward optimal solutions and steering clear of inferior solutions in each iteration [15]. The Jaya algorithm, categorized as a metaheuristic algorithm, emphasizes convenience and simplicity. Unlike many other metaheuristic algorithms, Jaya does not rely on algorithm-specific parameters to achieve optimal results. It is believed that Jaya can reduce computing time due to its parameter-less nature. With two inputs, namely population and iteration, the Jaya algorithm establishes the population randomly, and iteration represents the number of repetitions the algorithm must undergo. MLP-Jaya is divided into two main processes, namely the hyperparameter optimization process and the testing process. The detailed flow of MLP-Jaya is illustrated in Figure 2 and Table 2 describes the hyperparameter that will be optimized.

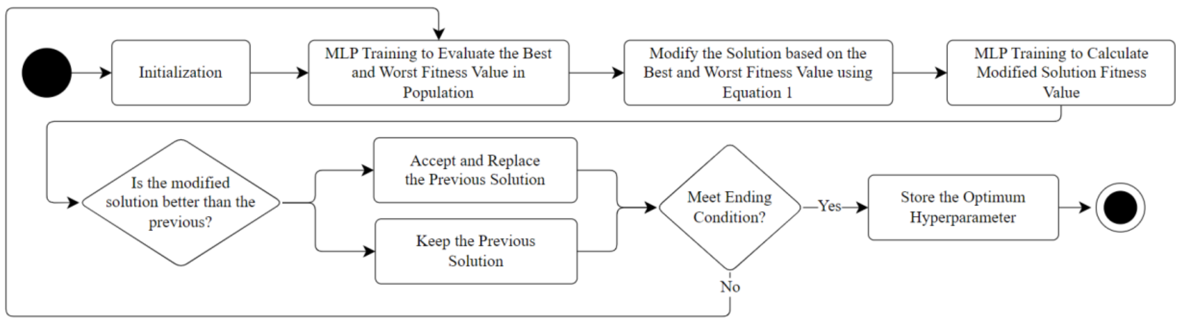
Initialization
MLP Training to Evaluate the Best and Worst Fitness Value in Population
Modify the Solution based on the Best and Worst Fitness Value using Equation 1
MLP Training to Calculate Modified Solution Fitness Value
Is the modified solution better than the previous?
Accept and Replace the Previous Solution
Keep the Previous Solution
Meet Ending Condition?
Yes
No
Store the Optimum Hyperparameter

Figure 2. MLP-Jaya flow

Table 2. MLP hyperparameter that will be optimized

Hyperparameter	Range	Interval	Total
Learning rate	[0.001 – 0.1]	0.001	100
Batch size	[16 – 128]	1	113
Number of layers	[1 – 5]	1	3
Number of neurons	[10 – 100]	1	91
Optimizer	[‘adam’:1, ‘sgd’:2]	-	2
Activation function	[‘logistic’:1, ‘tanh’:2, ‘relu’:3]	-	3

The MLP hyperparameter optimization process using Jaya is as:

i. Step 1 Initialization. The initialization process is the process of determining the initial parameter values for the Jaya algorithm. Jaya algorithm parameter used in this research are 20 population size and 50 iterations. After the initialization process, a random population will be generated equal to the population size. Population consists of 20 hyperparameter set. Table 3 provides an illustration of the population.

Table 3. MLP-Jaya population illustration

Population	X_1	X_2	X_3	X_4	X_5	X_6
1	0.038	83	2	99	1	1
2	0.023	114	2	43	2	3
…	….	…	…	…	…	…
20	0.055	16	3	58	3	1

Each population consist of X_1 through X_6, representing the hyperparameters to be optimized. Specifically, X_1 stands for learning rate, X_2 stands for batch size, X_3 stands for number of layers, X_4 stands for number of neurons, X_5 stands for optimizer, and X_6 stands for activation function. The optimizer and activation function will be represented by integers and then converted to their respective real values.

ii. Step 2 Determine the best and worst fitness value. Each candidate in the population undergoes the MLP training process to obtain an accuracy value from the validation set, which serves as the objective function value. The candidate with the highest accuracy is deemed the best candidate, while the candidate with the lowest accuracy is considered the worst candidate.

iii. Step 3 Population modification. Each candidate solution will have its value modified using (1).

$$X'_{j,k,i} = X_{j,k,i} + r_{1,j,i}\left(X_{j,best,i} - |X_{j,k,i}|\right) - r_{2,j,i}\left(X_{j,worst,i} - |X_{j,k,i}|\right) \quad (1)$$

$X_{j,k,i}$ stands for j^{th} X value of the k^{th} candidate in i^{th} iteration. $X'_{j,k,i}$ stands for updated value of $X_{j,k,i}$. $r_{1,j,i}$ and $r_{2,j,i}$ stands for the first and second random value for j^{th} variable in i^{th} iteration. $X_{j,best,i}$

stands for j^{th} X value of the best candidate in i^{th} iteration and $X_{j,worst,i}$ stands for j^{th} X value of the worst candidate in i^{th} iteration. The best candidate is the candidate that has the highest accuracy and worst candidate is the candidate that has the lowest accuracy.

iv. Step 4 MLP training to calculate modified solution fitness value. Step 3 produces a population with modified candidates. MLP training is then performed again to obtain the fitness value of the latest population.

v. Step 5 Update the population. If the modified population has a better fitness value, the initial candidates will be replaced with the modified candidates using (1). Otherwise, the initial candidates will be retained.

vi. Step 6 Iterations. Continue iterating through Steps 2 to 5 until the number of iterations is reached.

vii. Step 7 Store the optimized hyperparameters. The candidate with the highest accuracy in the final iteration's population will become the best solution.

The first phase yields an optimal solution, comprising a combination of hyperparameters with the highest accuracy value, deemed as optimized hyperparameters. These optimized hyperparameter values will be employed for the testing phase. After concluding the testing phase, the evaluation stage begins. The MLP-Jaya model's evaluation encompasses precision, recall, F1-score, and accuracy metrics. The MLP-Jaya process will be iterated 10 times with varying populations and tested across 6 disease datasets to attain more precise results. Fitness value and execution time from each iteration will be averaged.

2.4. Evaluation

The final stage involves assessing performance, employing metrics such as accuracy, precision, recall, and F1-score for model evaluation. All these evaluation measures are derived from the confusion matrix. A higher value in these metrics indicates a more accurate classification by the model. Confusion matrix could be seen in Table 4. Precision, recall, F1-score, and accuracy will be calculated using (2) to (5).

Table 4. Confusion matrix

	Predicted positive	Predicted negative
Actual positive	True positive (TP)	False negative (FN)
Actual negative	False positive (FP)	True negative (TN)

$$Precision = \frac{TP}{TP+FP} \tag{2}$$

$$Recall = \frac{TP}{TP+FN} \tag{3}$$

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision+Recall} \tag{4}$$

$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN} \tag{5}$$

3. RESULTS AND DISCUSSION

This study examined the influence of the Jaya algorithm on MLP hyperparameter optimization, whereas previous studies have focused on the Jaya algorithm's effects on feature selection or its effects on weight and bias optimization. The results demonstrate that the Jaya algorithm yields high accuracy when utilized for optimizing MLP hyperparameters and Jaya algorithm could improve MLP accuracy (Table 5).

Table 5. Improvement in MLP-Jaya performance

Dataset	MLP	MLP-Jaya	Performance improvement
D1	0.9342	0.9608	2.77%
D2	0.8103	0.8211	1.33%
D3	0.655	0.864	31.91%
D4	0.6351	0.8487	33.63%
D5	0.7556	0.8231	8.93%
D6	0.8262	0.7897	-3.33%

This section implements the proposed method's classification performance. Furthermore, MLP-Jaya's performance compared with MLP-GA and MLP-PSO. Ten different random populations had been used in ten experiments. Output from the experiments is optimized hyperparameter. Optimized hyperparameter is a set of hyperparameters obtained from model with highest accuracy and lowest execution time. Optimized hyperparameter from each model and each dataset shown in Table 6.

Table 6. Optimized hyperparameter

Dataset	Model	Optimized hyperparameter						
		Learning rate	Batch size	Number of hidden layers	Number of neurons	Optim-izer	Activation function	Accuracy
D1	**MLP-Jaya**	**0.093**	**68**	**1**	**71**	**Adam**	**Relu**	**98.48%**
	MLP-GA	0.017	56	23	19	Adam	Relu	96.49%
	MLP-PSO	0.049	49	2	56	Adam	Relu	98.25%
D2	**MLP-Jaya**	**0.022**	**80**	**1**	**5**	**Adam**	**Relu**	**87.18%**
	MLP-GA	0.019	67	18	141	Sgd	Relu	84.61%
	MLP-PSO	0.026	46	3	48	Adam	Relu	82.05%
D3	**MLP-Jaya**	**0.011**	**38**	**2**	**129**	**Adam**	**Relu**	**95%**
	MLP-GA	0.003	51	13	24	Adam	Relu	91.25%
	MLP-PSO	0.059	33	1	53	Adam	Relu	92.5%
D4	MLP-Jaya	0.031	43	3	34	Adam	Relu	76.62%
	MLP-GA	**0.015**	**34**	**8**	**14**	**Adam**	**Relu**	**77.27%**
	MLP-PSO	0.046	39	2	59	Adam	Relu	71.43%
D5	**MLP-Jaya**	**0.001**	**98**	**10**	**7**	**Adam**	**Relu**	**81.46%**
	MLP-GA	0.013	20	9	10	Adam	Relu	78.05%
	MLP-PSO	0.018	82	6	31	Adam	Relu	80.97%
D6	MLP-Jaya	0.118	48	2	14	Adam	Logistic	87.5%
	MLP-GA	0.007	60	11	22	Adam	Relu	86.90%
	MLP-PSO	**0.047**	**27**	**1**	**79**	**Adam**	**Tanh**	**88.09%**

Table 5 illustrates the improvement in MLP performance after hyperparameter optimization. These results represent the average of 10 experiments conducted with different populations in the Jaya algorithm and different random states in MLP without optimization. The results in Table 5 indicate an improvement in MLP with hyperparameter optimization using Jaya across all datasets except D6. Significant improvement after hyperparameter optimization is observed in D3 and D4. All features in dataset D6 are categorical data that have been label-encoded, resulting in data containing only 0 or 1. Since the data is not continuous, MLP with default hyperparameters is more appropriate.

Table 6 shows that from 4 from 6 datasets MLP-Jaya has the highest accuracy compared to MLP-GA and MLP-PSO. Different datasets have different levels of accuracy, suggesting that the type of data has an impact on how well the models work. D4 has lesser accuracy across all optimization methods, for instance, whereas D1 and D3 typically have great accuracy, that is surpass 90%. The MLP-GA results show some variation in hyperparameter values, and the achieved accuracy varies across datasets. It tends to use moderate learning rates and batch sizes. The number of hidden layers and neurons is diverse, reflecting the nature of the genetic algorithm's exploration.

Relu is consistently selected as the activation function and Adam is consistently selected as the optimizer across all datasets and optimization techniques. This could imply that these decisions hold up well when applied to various datasets and optimization techniques. Relu is widely employed in neural networks for several reasons. It has been demonstrated to exhibit superior training performance across various domains. Additionally, Relu contributes to accelerating the learning convergence in deep learning, a crucial factor when training extensive and intricate neural networks [29]. The widespread utilization of the Adam optimizer is attributed to its capacity for adaptive gain and swifter convergence in optimization algorithms, especially within the realm of deep learning. In contrast to conventional optimization methods, Adam dynamically adjusts the learning rate for each parameter independently, relying on the average of the first and second moments of the gradients. This adaptability in learning rate and momentum enables Adam to expedite convergence and diminish the likelihood of getting stuck in local optima, particularly in intricate optimization landscapes characterized by a substantial number of parameters, as is often encountered in deep learning models [30], [31]. The fact that different datasets have different optimized hyperparameters emphasises how crucial hyperparameter tuning is to get the best results on tasks [32]. Table 7 shows average accuracy and execution time comparison from 10 experiments. The experiments repeated 10 times with different random populations to get more accurate result. MLP-Jaya outperforms the other algorithms in four out of six experiments. In D5 case, MLP-Jaya had the maximum accuracy, but lost to MLP-PSO when the accuracy is averaged from 10 experiments. The performance varies significantly across datasets for each algorithm. For example, D1 and D3 show relatively high accuracy, while D4 consistently has lower accuracy across all algorithms. MLP-Jaya tends to achieve high accuracy on D1, D3, and D5. MLP-GA generally exhibits a broader range of accuracy, with higher standard deviation values. It achieves high accuracy on some datasets (e.g., D1, D2, D3) but lower accuracy on others (e.g., D4, D5). In some cases, high standard deviation values indicate that the algorithm's performance is sensitive to the choice of initial conditions or randomness in the optimization process. Lower standard deviation values suggest more stable and predictable performance.

Table 7. Accuracy and execution time comparison from 10 experiments

Dataset	Model	Accuracy				Avg. execution time (minute)
		Max	Min	Average	Std. Deviation	
D1	MLP-Jaya	**0.9848**	**0.9474**	**0.9608**	0.0117	19.72
	MLP-GA	0.9649	0.6228	0.8211	0.1708	23.1
	MLP-PSO	0.9825	0.6228	0.864	0.1668	**6.2**
D2	MLP-Jaya	**0.8718**	**0.8205**	**0.8487**	0.0255	**1.65**
	MLP-GA	0.8462	0.8205	0.8231	0.0081	11.12
	MLP-PSO	0.8462	0.641	0.7897	0.0671	2.3
D3	MLP-Jaya	**0.95**	**0.65**	**0.7838**	0.1422	2.94
	MLP-GA	0.9125	0.35	0.6425	0.1332	26.04
	MLP-PSO	0.9375	0.65	0.7638	0.1163	**2.93**
D4	MLP-Jaya	0.7662	**0.5519**	**0.683**	0.0673	12.5
	MLP-GA	**0.7727**	0.3571	0.6671	0.1128	17.53
	MLP-PSO	0.7143	0.3571	0.6246	0.1039	**7.04**
D5	MLP-Jaya	**0.8146**	**0.7512**	**0.78**	0.0167	12.23
	MLP-GA	0.7805	0.4976	0.5729	0.1144	34.58
	MLP-PSO	0.8098	0.4976	0.7166	0.1167	**9.46**
D6	MLP-Jaya	0.875	0.5298	0.7652	0.1553	**7.49**
	MLP-GA	0.869	0.5298	0.6759	0.1692	40.28
	MLP-PSO	**0.881**	**0.4702**	**0.789**	0.1464	9.97

MLP-GA consistently has the highest execution times, indicating that it requires more computational resources to find solutions. The computational expense and time in genetic algorithms are further influenced by the iterative execution of selection, crossover, and mutation processes on population until a termination condition is satisfied, demanding substantial computational resources [33]. MLP-Jaya and MLP-PSO demonstrate relatively lower execution times, with MLP-PSO being the most efficient in terms of execution time. This finding contradicts [23], which states that the Jaya algorithm has a faster execution time compared to PSO. The execution time in this study is heavily influenced by the MLP training process with a sufficiently large number of layers, thereby causing the MLP to operate for a longer duration. Figure 3 shown precision, recall, and f1-score for all models across all datasets. Figure 3(a) shown evaluation metrics for all models on D1, Figure 3(b) on D2, Figure 3(c) on D3, Figure 3(d) on D4, Figure 3(e) on D5, and Figure 3(f) on D6.

The results presented in Figure 3 are consistent with the results shown in Table 5. Across all datasets, MLP-Jaya generally demonstrates higher precision, recall, and F1-score compared to MLP-GA and MLP-PSO. MLP-GA tends to have lower precision and recall values compared to the other algorithms. This suggests that there might be a trade-off between precision and recall in the solutions found by MLP-GA, resulting in a compromise in f1-score and MLP-GA may struggle to find optimal solutions that balance precision and recall effectively. Speed of convergence comparison for all models across all datasets can be shown in Figure 4. Figure 4(a) shown speed of convergence for all models on D1, Figure 4(b) on D2, Figure 4(c) on D3, Figure 4(d) on D4, Figure 4(e) on D5, and Figure 4(f) on D6.

The convergence criterion in metaheuristics refers to the conditions under which a metaheuristic algorithm is considered to have converged to a solution. MLP-Jaya consistently achieves convergence in fewer iterations across most datasets. It reaches the convergence criterion (e.g., validation accuracy above 0.95 or 1.0) in either the first iteration or a very low number of iterations, makes it more suitable to real-time application [34]. MLP-Jaya can achieve rapid convergence due to Jaya's ability to direct the population towards the best solution and away from the worst solution [35].

MLP-GA tends to require more iterations to achieve convergence compared to MLP-Jaya. In the cases of D1 and D3, MLP-GA achieves convergence in a relatively low number of iterations with competitive validation accuracy. In D2, D4, D5, and D6, MLP-GA has the same validation accuracy from the first iteration, but the accuracy obtained cannot approach the validation accuracy of MLP-Jaya and MLP-PSO. MLP-PSO demonstrates a competitively fast convergence rate. MLP-PSO generally converges faster than MLP-GA, but in some cases, it requires more iterations to converge compared to MLP-Jaya. In D1 and

D3, MLP-PSO converges relatively quickly, while in D5 and D6, it requires more iterations compared to other datasets.

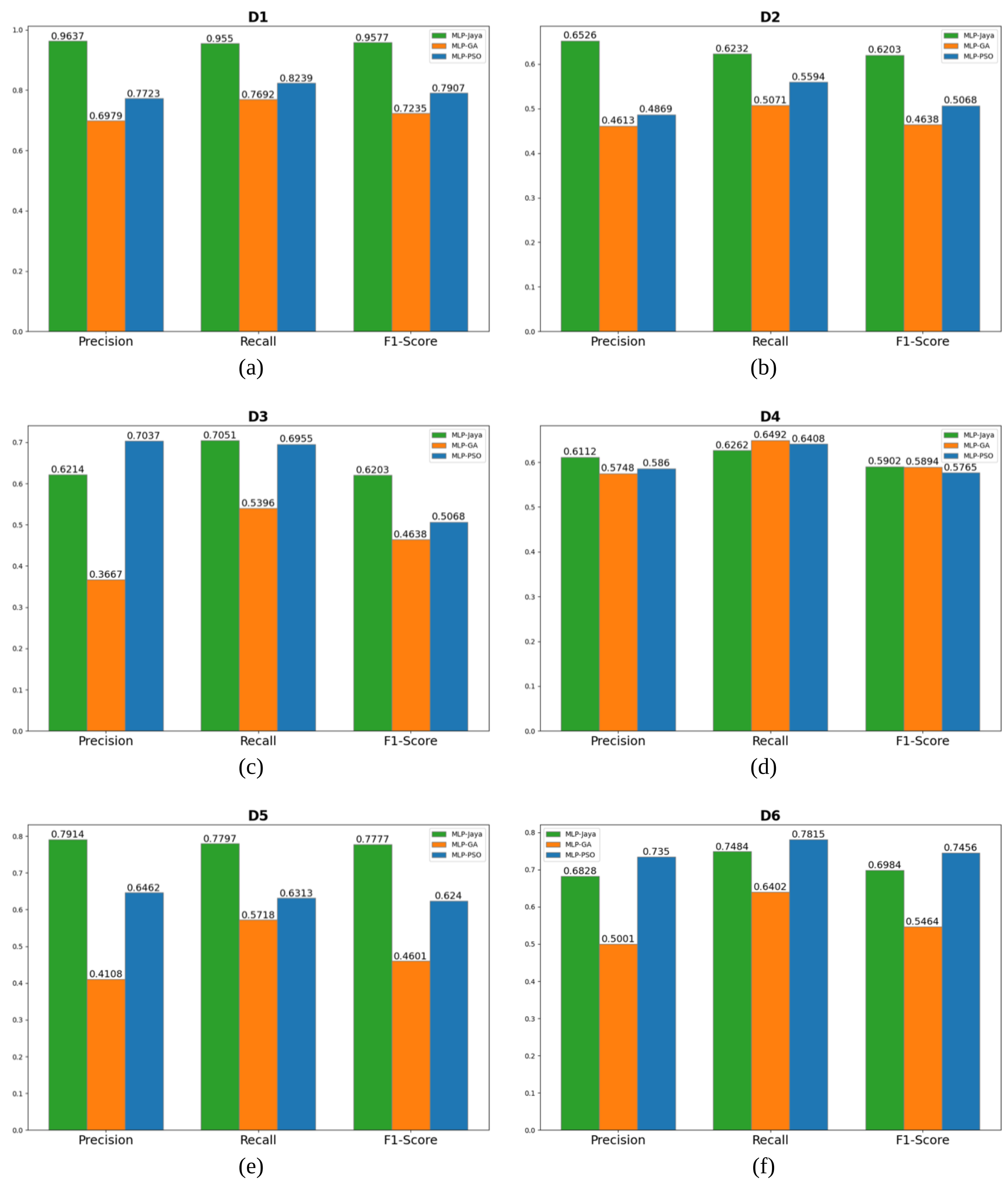

Figure 3. Evaluation metrics on (a) D1, (b) D2, (c) D3, (d) D4, (e) D5, and (f) D6

While MLP-Jaya often converges quickly, the corresponding validation accuracy varies across datasets. In some cases, MLP-PSO achieves higher accuracy at convergence, as seen in D3 and D6. MLP-GA, on the other hand, may not reach as high accuracy levels as MLP-Jaya or MLP-PSO on certain datasets. The convergence speed and accuracy at convergence are influenced by the characteristics of each dataset. For example, D3 and D6 show relatively faster convergence across all algorithms, while D2 and D4 may pose challenges for convergence, especially for MLP-GA.

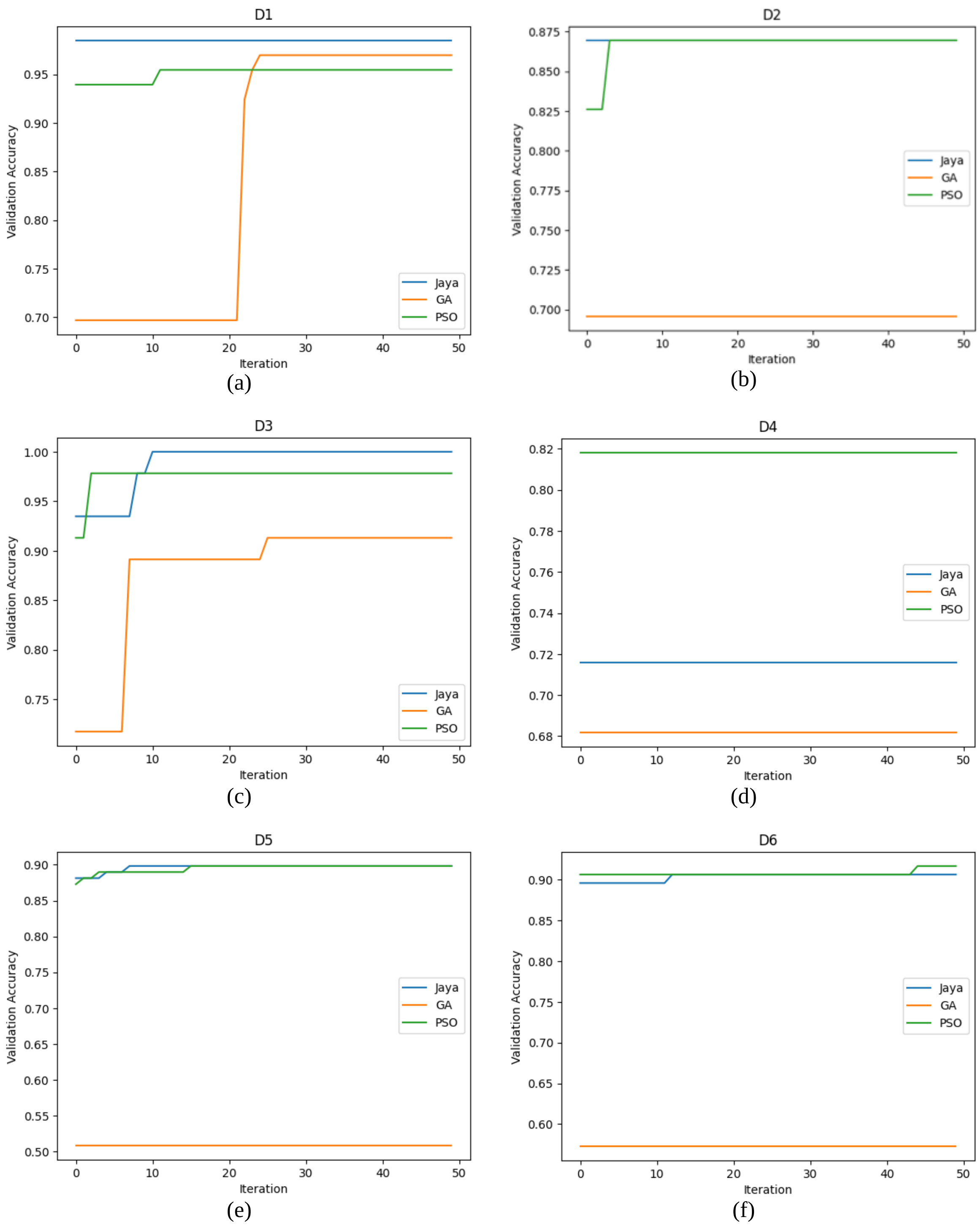

Figure 4. Speed of convergence comparison on (a) D1, (b) D2, (c) D3, (d) D4, (e) D5, and (f) D6

Despite its competitive performance, MLP-Jaya is susceptible to overfitting, much like other metaheuristic algorithms. This is due to the heavy dependence of metaheuristic algorithms on the initialization of the population. Random population initialization is utilized in this study because it adheres to the implementation of the Jaya algorithm as outlined by [17]. Random initialization suffers from high discrepancy and inefficient coverage of the search space. The discrepancy of the random numbers significantly impacts the authenticity of the randomly generated solutions within the search spaces [36]. Additional and thorough investigations may be necessary to validate the impact of population initialization on the performance of the Jaya algorithm.

This study illustrates that the Jaya algorithm is more dependable compared to PSO and GA. Despite its quick execution time, PSO frequently gets stuck in local optima solutions, whereas GA demands more computational resources. This positions Jaya as a competitive option for optimization tasks, particularly for real-time applications. This is because Jaya can achieve competitive results while quickly converging. Future research studies could investigate the performance of the Jaya algorithm on real-time applications using alternative methods of population initialization.

4. CONCLUSION

In summary, this paper investigates the utility of the Jaya algorithm for hyperparameter optimization in the widely used MLP model. The proposed MLP-Jaya aims to create an efficient and straightforward model, eliminating the need for intricate adjustments. Results indicate the superiority of MLP-Jaya over other metaheuristic approaches (MLP-GA and MLP-PSO) in terms of accuracy (highest accuracy is 96.08% for MLP-Jaya, 82.11% for MLP-GA, 86.4% for MLP-PSO in breast cancer Wisconsin dataset) across diverse datasets. The algorithm demonstrates robustness, achieving competitive hyperparameter settings and faster convergence (MLP-Jaya reaching convergence below 15 iterations), particularly beneficial for real-time applications. Evaluation metrics such as precision, recall, and F1-score consistently favor MLP-Jaya, highlighting its efficiency in execution time compared to MLP-GA and competitive performance against MLP-PSO. In conclusion, MLP-Jaya emerges as a potent and streamlined hyperparameter tuning solution for MLP, offering simplicity, parameter-less design, and reliable performance across various scenarios.

REFERENCES

[1] S. B. Kotsiantis, "Supervised machine learning: a review of classification techniques," *Informatica*, vol. 31, pp. 249–268, 2007.

[2] G. Kesavaraj and S. Sukumaran, "A study on classification techniques in data mining," in *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, Jul. 2013, pp. 1–7, doi: 10.1109/ICCCNT.2013.6726842.

[3] R. Andonie, "Hyperparameter optimization in learning systems," *Journal of Membrane Computing*, vol. 1, no. 4, pp. 279–291, Dec. 2019, doi: 10.1007/s41965-019-00023-0.

[4] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei, and S.-H. Deng, "Hyperparameter optimization for machine learning models based on Bayesian optimization," *Journal of Electronic Science and Technology*, vol. 17, no. 1, pp. 26–40, 2019, doi: 10.11989/JEST.1674-862X.80904120.

[5] R. O. Ogundokun, S. Misra, M. Douglas, R. Damaševičius, and R. Maskeliūnas, "Medical internet-of-things based breast cancer diagnosis using hyperparameter-optimized neural networks," *Future Internet*, vol. 14, no. 5, p. 153, May 2022, doi: 10.3390/fi14050153.

[6] S. Sanders and C. Giraud-Carrier, "Informing the use of hyperparameter optimization through metalearning," in *2017 IEEE International Conference on Data Mining (ICDM)*, Nov. 2017, pp. 1051–1056, doi: 10.1109/ICDM.2017.137.

[7] J. Han, C. Gondro, K. Reid, and J. P. Steibel, "Heuristic hyperparameter optimization of deep learning models for genomic prediction," *G3 Genes|Genomes|Genetics*, vol. 11, no. 7, p. jkab032, Jul. 2021, doi: 10.1093/g3journal/jkab032.

[8] N. Lin, Y. Chen, H. Liu, and H. Liu, "A comparative study of machine learning models with hyperparameter optimization algorithm for mapping mineral prospectivity," *Minerals*, vol. 11, no. 2, p. 159, Feb. 2021, doi: 10.3390/min11020159.

[9] K. Shankar, Y. Zhang, Y. Liu, L. Wu, and C.-H. Chen, "Hyperparameter tuning deep learning for diabetic retinopathy fundus image classification," *IEEE Access*, vol. 8, pp. 118164–118173, 2020, doi: 10.1109/ACCESS.2020.3005152.

[10] Z. Shen, Y. Zhang, L. Wei, H. Zhao, and Q. Yao, "Automated machine learning: from principles to practices," Oct. 2018, doi: arXiv:1810.13306v1.

[11] E. Uzlu, "Application of Jaya algorithm-trained artificial neural networks for prediction of energy use in the nation of Turkey," *Energy Sources, Part B: Economics, Planning, and Policy*, vol. 14, no. 5, pp. 183–200, May 2019, doi: 10.1080/15567249.2019.1653405.

[12] T. Dokeroglu, E. Sevinc, T. Kucukyilmaz, and A. Cosar, "A survey on new generation metaheuristic algorithms," *Computers & Industrial Engineering*, vol. 137, p. 106040, Nov. 2019, doi: 10.1016/j.cie.2019.106040.

[13] A. Pranolo, Y. Mao, A. P. Wibawa, A. B. P. Utama, and F. A. Dwiyanto, "Optimized three deep learning models based-PSO hyperparameters for Beijing PM2.5 prediction," *Knowledge Engineering and Data Science*, vol. 5, no. 1, p. 53, Jun. 2022, doi: 10.17977/um018v5i12022p53-66.

[14] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, pp. 295–316, Nov. 2020, doi: 10.1016/j.neucom.2020.07.061.

[15] H. Das, B. Naik, and H. S. Behera, "A Jaya algorithm based wrapper method for optimal feature selection in supervised classification," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 6, pp. 3851–3863, Jun. 2022, doi: 10.1016/j.jksuci.2020.05.002.

[16] E. H. Houssein, A. G. Gad, and Y. M. Wazery, "Jaya algorithm and applications: a comprehensive review," in *Lecture Notes in Electrical Engineering*, Springer International Publishing, 2021, pp. 3–24.

[17] K. Yu, B. Qu, C. Yue, S. Ge, X. Chen, and J. Liang, "A performance-guided JAYA algorithm for parameters identification of photovoltaic cell and module," *Applied Energy*, vol. 237, pp. 241–257, Mar. 2019, doi: 10.1016/j.apenergy.2019.01.008.

[18] R. Venkata Rao, "Jaya: a simple and new optimization algorithm for solving constrained and unconstrained optimization problems," *International Journal of Industrial Engineering Computations*, pp. 19–34, 2016, doi: 10.5267/j.ijiec.2015.8.004.

[19] R. A. Zitar, M. A. Al-Betar, M. A. Awadallah, I. A. Doush, and K. Assaleh, "An intensive and comprehensive overview of JAYA algorithm, its versions and applications," *Archives of Computational Methods in Engineering*, vol. 29, no. 2, pp. 763–792, Mar. 2022, doi: 10.1007/s11831-021-09585-8.

[20] Y.-D. Zhang *et al.*, "Smart pathological brain detection by synthetic minority oversampling technique, extreme learning machine, and Jaya algorithm," *Multimedia Tools and Applications*, vol. 77, no. 17, pp. 22629–22648, Sep. 2018, doi: 10.1007/s11042-017-5023-0.

[21] S.-H. Wang, P. Phillips, Z.-C. Dong, and Y.-D. Zhang, "Intelligent facial emotion recognition based on stationary wavelet entropy and Jaya algorithm," *Neurocomputing*, vol. 272, pp. 668–676, Jan. 2018, doi: 10.1016/j.neucom.2017.08.015.

[22] P. Singh, K. K. Mishra, and P. Dwivedi, "Enhanced hybrid model for electricity load forecast through artificial neural network and Jaya algorithm," in *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*, Jun. 2017, pp. 115–120, doi: 10.1109/ICCONS.2017.8250660.

[23] M. Alshutbi, Z. Li, M. Alrifaey, M. Ahmadipour, and M. M. Othman, "A hybrid classifier based on support vector machine and Jaya algorithm for breast cancer classification," *Neural Computing and Applications*, vol. 34, no. 19, pp. 16669–16681, Oct. 2022, doi: 10.1007/s00521-022-07290-6.

[24] W. Warid, H. Hizam, N. Mariun, and N. Abdul-Wahab, "Optimal power flow using the Jaya algorithm," *Energies*, vol. 9, no. 9, p. 678, Aug. 2016, doi: 10.3390/en9090678.

[25] R. V. Rao, D. P. Rai, and J. Balic, "Surface grinding process optimization using Jaya algorithm," in *Advances in Intelligent Systems and Computing*, Springer India, 2016, pp. 487–495.

[26] N. Goel, S. Chacko, and R. N. Patel, "A parameter less stochastic optimization technique for tuning of speed PI controller of DTC induction motor drive," *IAES International Journal of Robotics and Automation (IJRA)*, vol. 8, no. 2, p. 105, Jun. 2019, doi: 10.11591/ijra.v8i2.pp105-112.

[27] H. Ramchoun, M. Amine, J. Idrissi, Y. Ghanou, and M. Ettaouil, "Multilayer perceptron: architecture optimization and training," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 4, no. 1, p. 26, 2016, doi: 10.9781/ijimai.2016.415.

[28] F. Ecer, "Comparing the bank failure prediction performance of neural networks and support vector machines: the Turkish case," *Economic Research-Ekonomska Istraživanja*, vol. 26, no. 3, pp. 81–98, Jan. 2013, doi: 10.1080/1331677X.2013.11517623.

[29] K. Hara, D. Saito, and H. Shouno, "Analysis of function of rectified linear unit used in deep learning," in *2015 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2015, pp. 1–8, doi: 10.1109/IJCNN.2015.7280578.

[30] E. C. Seyrek and M. Uysal, "Performance analysis of MADGRAD, MirrorMADGRAD and ADAM optimizers on a CNN model in the hyperspectral image classification," in *2023 10th International Conference on Recent Advances in Air and Space Technologies (RAST)*, Jun. 2023, pp. 1–6, doi: 10.1109/RAST57548.2023.10197855.

[31] Z. Fang, X. Xu, X. Li, H. Yang, and C. Gong, "SPGD algorithm optimization based on Adam optimizer," in *AOPC 2020: Optical Sensing and Imaging Technology*, Nov. 2020, p. 174, doi: 10.1117/12.2579991.

[32] H.-T. Vo, H. T. Ngoc, and L.-D. Quach, "An approach to hyperparameter tuning in transfer learning for driver drowsiness detection based on Bayesian optimization and random search," *International Journal of Advanced Computer Science and Applications*, vol. 14, no. 4, 2023, doi: 10.14569/IJACSA.2023.0140492.

[33] P. S. Oliveto and C. Witt, "Improved time complexity analysis of the simple genetic algorithm," *Theoretical Computer Science*, vol. 605, pp. 21–41, Nov. 2015, doi: 10.1016/j.tcs.2015.01.002.

[34] M. Psarakis, A. Dounis, A. Almabrok, S. Stavrinidis, and G. Gkekas, "An FPGA-based accelerated optimization algorithm for real-time applications," *Journal of Signal Processing Systems*, vol. 92, no. 10, pp. 1155–1176, Oct. 2020, doi: 10.1007/s11265-020-01522-5.

[35] J. Yu, C.-H. Kim, A. Wadood, T. Khurshaid, and S.-B. Rhee, "Jaya algorithm with self-adaptive multi-population and Lévy flights for solving economic load dispatch problems," *IEEE Access*, vol. 7, pp. 21372–21384, 2019, doi: 10.1109/ACCESS.2019.2899043.

[36] J. O. Agushaka and A. E. Ezugwu, "Initialisation approaches for population-based metaheuristic algorithms: a comprehensive review," *Applied Sciences*, vol. 12, no. 2, p. 896, Jan. 2022, doi: 10.3390/app12020896.

BIOGRAPHIES OF AUTHORS

Andien Dwi Novika is presently a master's degree student in computer science at Bina Nusantara University. Having earned her bachelor's degree in computer science from Diponegoro University, she brings a wealth of academic knowledge to her current pursuit. She has gained practical experience through various roles, including serving as an integration staff at NicePay, a full stack developer intern at Maven Digital Asia, and a back-end developer intern at Gagas Cipta Persada. She can be reached at: andien.novika@binus.ac.id.

Abba Suganda Girsang currently a lecturer in the Master of Information Technology program at Bina Nusantara University in Jakarta. He earned his Ph.D. from the Institute of Computer and Communication Engineering, Department of Electrical Engineering, at National Cheng Kung University in Tainan, Taiwan, in 2014. His undergraduate degree is from the Department of Electrical Engineering at Gadjah Mada University (UGM) in Yogyakarta, Indonesia, in 2000. Subsequently, he pursued his master's degree in the Department of Computer Science at the same university from 2006 to 2008. Throughout his career, he gained experience as a staff consultant programmer at Bethesda Hospital in Yogyakarta in 2001 and worked as a web developer from 2002 to 2003. He later joined the faculty of the Department of Informatics Engineering at Janabadra University, serving as a lecturer from 2003 to 2015. Additionally, he taught various subjects at different universities from 2006 to 2008. Abba Suganda Girsang's research interests encompass swarm intelligence, combinatorial optimization, and decision support systems. For further communication, he can be reached at email: agirsang@binus.edu.