International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Hybrid Neural Network with Weighted Modified Cuckoo Search Optimization for Software Defect Prediction: A Soft Computing Approach

Devi Priya Gottumukkala[1], Prasad Reddy P V G D [2], S. Krishna Rao[3]

Submitted:13/03/2024 **Revised**: 28/04/2024 **Accepted**: 05/05/2024

Abstract: Defects in software significantly impact quality, reliability, and maintenance. Early detection and prediction using data mining and classification techniques offers an effective means of identifying potential defects before they manifest in production environments, but accurate prediction requires handling complex datasets. This paper proposes a soft computing model called Hybrid Neural Network with Weighted Modified Cuckoo Search Optimization (WMCSO) to detect the defect in the software. The proposed model first performs the clustering process with the Modified Fuzzy C–means algorithm (MFCM) to retrieve the important new attributes from the dataset. The software defect prediction and classification are performed using the HNN, and the WMCSO model is used to fine-tune the weights of the HNN. The HNN-WMCSO method is evaluated based on the evaluation of prediction rate and execution time. The experimental analysis stated that the proposed model exhibits improved performance relative to the current method in regard to an efficient prediction rate.

Keywords: *Software Defect Prediction (SDP), Fuzzy C-means (FCM), Cuckoo Search (CS), Machine Learning, Hybrid Neural Network (HNN)*

1. Introduction

Software Defect Prediction (SDP) is considered effective for reducing costs in software development and maintenance to achieve high-quality software [1]. Prolonged software failures can cause automated defects, impacting developers and stakeholders. Defect prediction techniques improve quality, reduce costs, and enhance product development time by proactively addressing issues [2]. A defect predictor with high performance consists of static attributes and an effective learning process. SDP employs various metrics to evaluate features and attributes in software based on lines of code (LOC) and information change to predict defect proneness. This predictive approach supports software activities by detecting likely defects during the initial stages of development, enabling teams to focus on critical areas and improve software quality [3]. Software defects are considered one of the

central factors precipitating the failure of massive engineering projects, which causes massive financial burdens. The software quality maintenance is evaluated with different prediction techniques [4]. Unsupervised learning can be used in conjunction with supervised techniques. Clustering can be used to identify homogeneous groups of data that can be further attached to the original data for supervised learning, enhancing the efficacy of defect prediction models [5].

Conventional SDP comprises an optimized prediction model for accurately estimating the fault instances within a software module. The goal is to achieve precise defect prediction, despite the presence of noisy data. By employing advanced techniques and methodologies, the model aims to provide an accurate prediction of defects, contributing to improved software quality and reliability[6].Machine learning algorithms have been proposed to anticipate potential defects in software systems[7-9].Ensemble-based SDP is a composite strategy integrating several prediction models to enhance the accuracy and reliability of defect prediction [10-11].Using unsupervised algorithms before classification in SDP brings several benefits and helps improve the overall predictive performance[12].

Using deep learning instead of traditional machine learning approaches in SDP offers several potential advantages. Neural networks in deep learning (DL) are adept at capturing complex patterns and representations from raw data, making it suitable for complex and high-dimensional software datasets [13]. DL Models have the capability to autonomously extract pertinent features from raw software metrics, dispensing with the manual creation of features. This ability can be especially beneficial when dealing with unstructured data, such as source code [14].

[1] *Research Scholar, Department of CS&SE, TDR-HUB, Andhra University, Visakhapatnam, India. Email: mantena2377@gmail.com ORCID ID: 0009-0002-2447-4449*
[2] *Senior Professor, Department of CS&SE, Andhra University, Visakhapatnam, India. Email: prasadreddy.vizag@gmail.com*
[3] *Professor, Dept. of CSE, Sir C.R.R.College of Engineering, ELURU, A.P, India. Email: skrao71@gmail.com*

Weight optimization is crucial in deep learning models, as it optimizes parameters like weights and biases to minimize loss or error function. The goal is to find weights that enable accurate predictions and generalization to unseen data [15]. Genetic algorithms (GA) optimize neural network weights due to their flexibility and Capacity to discern near-optimal solutions amidst intricate search spaces, unlike gradient-based algorithms that struggle with local minima especially in high-dimensional spaces with numerous parameters (weights) to optimize [16]. This research aimed to formulate an efficient SDP strategy. The specific contribution of the research is presented as follows:

1. To perform the SDP, this research proposed HNN-WMCSO. The model uses SDP through the HNN.
2. The collected data is computed based on the clustering process performed with the FCM-based approach to generate a more informative feature space. The Modified Fuzzy C Means Clustering (MFCM) technique is implemented for execution, followed by the classification process.
3. The classification model uses the HNN for the learning and testing process. To increases the efficiency of the neural network optimization algorithm is integrated with Weighted modified CS model.
4. The simulation analysis stated that the proposed model achieves the reduced processing time with the

   increased prediction rate.

The structure of this document is as follows: The relevant works for software fault prediction are included in Section 2. Section 3 presents the software defect prediction methodology and findings, while Section 4 presents the comparative analysis. Finally, Section 5 presents the general conclusion model.

2. Related Works

In [17] developed the back-propagation neural network (BPNN) algorithm for improving the training of Forward-propagation neural networks. The classic BP method has certain shortcomings, including a sluggish convergence rate and an easy tendency to become caught in local minima. In order to develop BP in attaining quick meeting point rates and avoiding local minima problems, their work suggested a search method dubbed CS. The effectiveness of the model is evaluated in comparison to existing hybrid variations and an artificial bee colony utilizing the BP method. The simulation results exhibits that the advised hybrid technique greatly enhances the computational efficiency of the BP training process.

Artificial Bee Colony and published a new Artificial Neural Network (ANN) In [18]. The ABC technique, which is used to verify the optimal weights for which a neural network should result, is employed in training the neural network. The false positive rate(FPR) and false negative rate(FNR), multiplied by the cost coefficients, are used to maximize the artificial bee colony. Five prominent sets from the NASA Metrics Data Program repository were studied using our methodology. Accuracy, probability of detection, probability of false alarm, profit, area under curve, and normalized expected cost of misclassification were the main performance characteristics of the classification strategy used in this procedure for the offered findings. To avoid any features of randomness, the ensemble was quite randomly combined, and then their approach was performed ten times. This was done by using n-fold cross-validation each of these times.

A novel hybrid version of the hitherto Salp Swarm Algorithm and Simulated Annealing, SSA-SA, based BPNN is proposed in [19]. The objective is to optimize parameters using BPNN estimator in SDE.

They evaluate the hybrid SSA-SA performance on a variety of SDE benchmark datasets. They tested the SSA-SA outcomes with respect to its competency to the SSA-BPNN and conventional BPNN. The hybrid algorithm is accomplished in parameter optimization in SDE and assessment criteria.

A unique SDP model built upon the GA-BP algorithm in light of the drawbacks of traditional BP (back propagation, or BP, for short) neural networks, which have the issue of easily falling into local optimization when building SDP models, which ultimately impacts the performance was reviewed in [20]. First, the Back Propagation neural network's weights and thresholds are optimized using the GA optimization capability. Then the GA-BP is applied to build the model. The program WEKA is used to convert the public dataset NASA MDP and clean it after that. The experimental results show that the proposed software fault prediction approach is efficient.

In [21] suggested the use of an SVM-based mixed CS under-sampled SDP model. First, the non-defective sample is chosen synchronously, and the SVM parameters are optimized with dynamic regional search (HMOCS). The non-defective modules are then chosen using three under-sampled approaches for decision region range. The three indicators— FPR, probability of detection (pd), and G-mean—are used in the simulation to assess how well the suggested algorithm is working. The Promise database are also chosen in order to validate the suggested SDP model. The suggested strategy is effective in resolving the problem of SDP when compared to the output of eight prediction models.

An innovative SDP framework across different stages was developed in [22]. First, a pre-processing stage was applied to the incoming data. The prepared data are used to extract

statistical characteristics, along with any relationships. Furthermore, enhanced PCA (Principal Component Analysis) is used to choose the required properties. Next, an enhanced CNN is used to predict flaws using the selected characteristics. The SALO method is used to correctly tune the CNN weights.

The five distinct datasets from the NASA Promise repository and the characteristics are chosen using a GA. The model is then trained applying various methods, including: Feed forward Neural Network (FNN) and Recurrent Neural Network (RNN), before the group of the chosen attributes is created using PSO in [23]. Finally, different classification metrics are calculated. According to their research, deep neural networks produce the greatest accuracy results. Findings from the experiments demonstrate that the proposed solution is a good approach for predicting software faults.

In [24] combined two algorithms wolf swarm algorithm and the particle swarm method to realize their complimentary benefits in accordance with the advantages. The model's fitness function is derived from the loss function, the hybrid technique is used to search for model hyper parameter optimization, and the swarm intelligence population's cooperative search ability is utilized to identify the globally ideal solution in a number of adjacent solution areas. The model utilizing the hybrid algorithm has higher and better indicators. In this research, the assessment of performance regarding to confusion matrix are used to evaluate the model. The model's performance has increased much more after auto encoder processing.

3. Proposed HNN-WMCSO for the defect prediction

The software defect detection comprises of the three modules. Initially, the data related to defects are collected and retained in the database for further processing. In the clustering stage the collected data is clustered to be organized into meaningful clusters, which may help identify underlying patterns or separate different classes more effectively and provides a more informative feature space. Additional features are incorporated into the original data for the subsequent classification model. The MFCM approach is utilized to achieve effective data prediction. Hybrid neural networks are used by the classification model for testing and learning. Modified Cuckoo Search model with weight factors is merged with the neural network optimization technique to boost its efficiency. The process involved in the proposed model is illustrated in the figure 1.

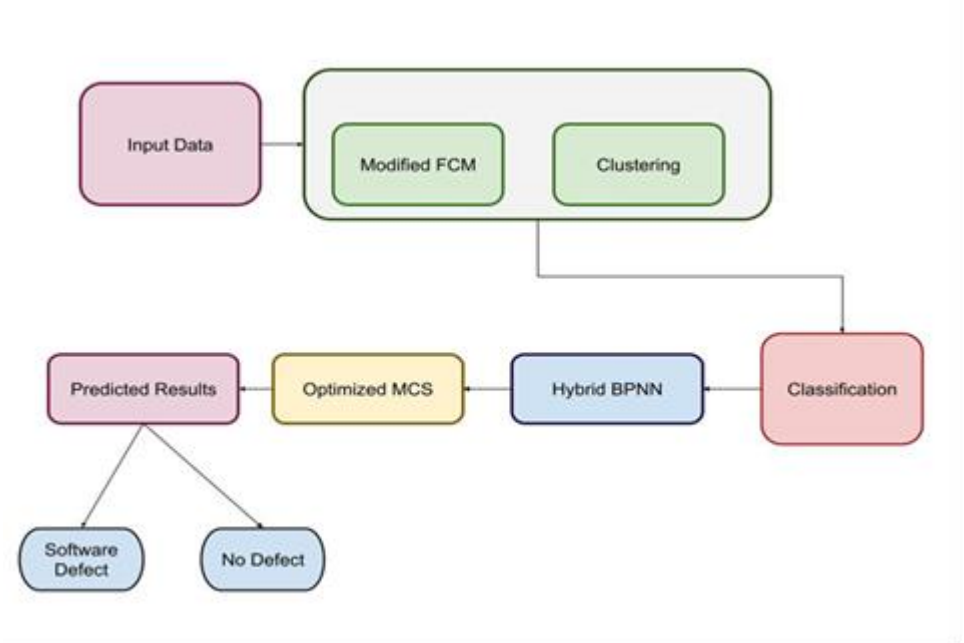


Fig. 1. Process in HNN - WMCSO model

3.1 Modified Fuzzy C-means Clustering

MFCM determines clusters in terms of the possibility that a data point belongs to that cluster with the membership function itself. where h is a probability that a data point is a member of a cluster. It means that the clustering depends on outcomes of membership function and tolerance measures for the desired accuracy. Membership functions with extremely low values are not computed during the needed iterations in order to gain the wished clustering outcome.

One of the major application limitations of FCM is its computational cost. The cluster centroid and membership variables are changed periodically until convergence in FCM because of its iterative technique and sensitivity to initialization. The iterative procedure can be exceedingly time-consuming and computationally expensive when the

dataset is huge if there are many clusters. To address this issue and achieve more accurate weight measurements, a modified approach called MFCM is adopted. The MFCM algorithm validates clusters using partitioned coefficients and partition entropy values to assess the quality of the resulting clusters. Partition coefficients and partition entropy are two such measures commonly used for this purpose. In the MFCM model, the initial cluster is selected based on the dependent value of the membership function. The MFCM then establishes initial cluster centers through arbitrary membership functions. The MFCM clustered assigned with each category based on membership fuzzy function aims to minimize the overall fuzziness of the clustering while also minimizing the distances between data points and cluster centroids as stated in equation (1)

$$M = \sum_{i=1}^{I}\sum_{j=1}^{J}(\mu_{ij})^{v}\frac{\|t_i - c_j\|^2}{\rho_i} \quad (1)$$

In above equation (1), t_i is the i^{th} data point, centroid of cluster j is represented as c_j, constant value is represented as v (often set to 2 in fuzzy clustering algorithms) and weight factor associated with data point i is stated as ρ_i for cluster i stated in equation (2), where ρ_i is a coefficient associated with the distance between data point i and cluster centroid j

$$\rho_i = \left\{\frac{\sum_{j=1}^{J}\delta_{ij}^{v}\|t_i - c_j\|^2}{\sum_{j=1}^{J}\delta_{ij}^{v}}\right\}^{1/2} \quad (2)$$

Each data point for each cluster is given a degree of membership by the membership function, which indicates how much the data point is related to that cluster. The membership function is presented in equation (3)

$$\mu_{ij} = \frac{1}{\sum_{k=1}^{J}\left\|\frac{t_i - \frac{c_j}{\rho_i}}{t_i - \frac{c_k}{\rho_i}}\right\|^{\frac{2}{v-1}}} \quad (3)$$

The centroid cluster values are computed as in equation (4)

$$c_j = \frac{\sum_{i=1}^{N}{\mu_{ij}}^{v}\cdot t_i}{\sum_{i=1}^{N}{\mu_{ij}}^{v}} \quad (4)$$

Based on the two iterations count the changes in coefficient values are repeated based on the sensitivity threshold value in equation (5)

$$max_{ij}\left\|\delta_{ij}^{(k)} - \delta_{ij}^{(k+1)}\right\| < \varphi \quad (5)$$

Above equation (5) verifies whether the maximum change in coefficient values between successive iterations is less than the sensitivity threshold ϕ. The clustering solution may not be considerably improved by more iterations if this criterion is met, which indicates that the algorithm is converging. The algorithm can therefore come to an end.

than the sensitivity threshold ϕ. The clustering solution may not be considerably improved by more iterations if this criterion is met, which indicates that the algorithm is converging. The algorithm can therefore come to an end.

3.2 Hybrid Neural Networks

Hybrid Neural Network (HNN) provides the resemblance of the biological counterparts for the estimation of effective tasks.

The process involved in HNN comprises of neurons with biological components to perform intended tasks in effective manner. The process of HNN is shown in figure 2.

Input Layer: This level includes the same quantity of neurons as that of inputs. However, here, the neurons are passive elements. It means that the input layer neurons do not transform the signal for the next layers .

Hidden Layer: This level also includes several neurons of any number. Although the hidden layer's neurons are active and they modulate the signal. Thus, it is easy to identify that the hidden layers make processing possible.

Output Layer: As the name suggests, this level includes the same number of neurons as the output. Unlike the other two layers, the output layer neurons symbolize the final output of the neural network due to the activation.

3.2.1 Training Phase

The input layer of HNN consists of M neurons, where M represents the count of inputs. In the hidden layers, there are NH neurons, and in the output layer, there are N neurons, each corresponding to one class. The Hybrid neural network model for the training with back propagation algorithm are presented in figure 3.

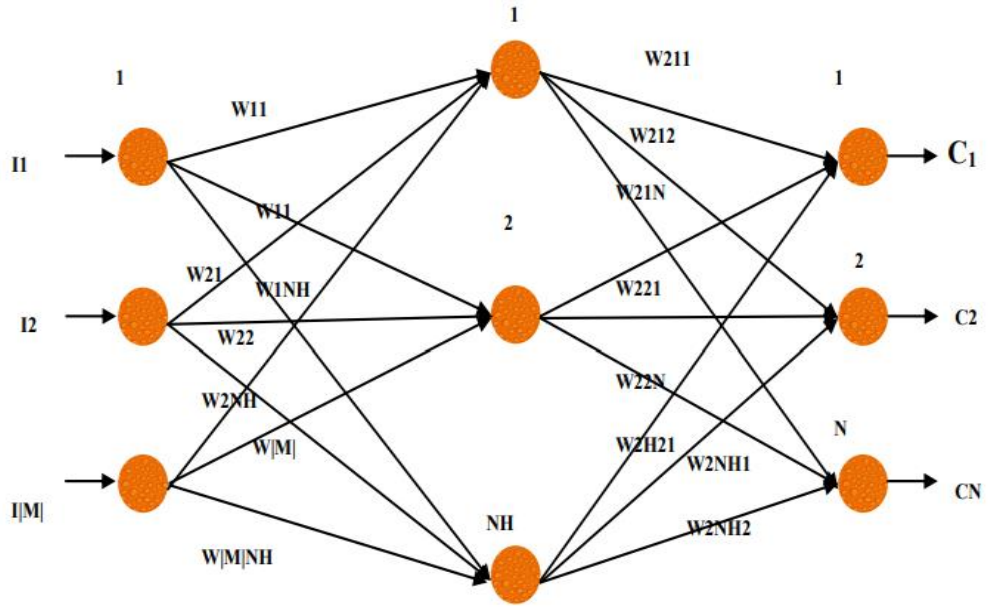


Fig 2: Structure of HNN

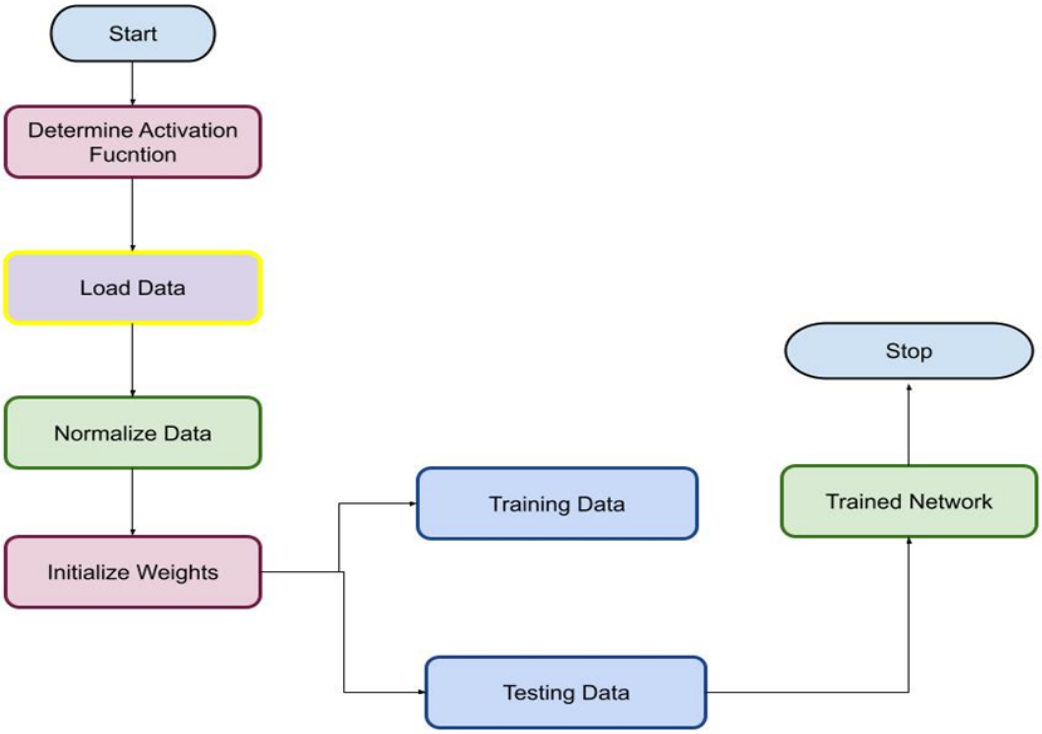


Fig 3: Process in HNN

The components in the Hybrid Neural Network are presented as shown:

Step 1: Generate hidden and output layer neurons through arbitrary weights between the interval [0,1]. The weight unity values are computed for the input layer of neurons.

Step 2: With the training dataset the BT determines the classification with the consideration of equation (6)

$$BP_{err} = C_{tar} - C_{out} \quad (6)$$

In the above equation (6), the targeted output is represented as C_{tar} for the network output stated as C_{out} those can be stated as

$$C_{out} = \left[Y_2^{(1)} Y_2^{(2)} \dots \dots . Y_2^{(N)}\right], Y_2^{(1)} Y_2^{(2)} \dots \dots . Y_2^{(N)}$$

for the output network. The output of network is stated as follows in equation (7)

$$Y_2^{(I)} = \sum_{r=1}^{N_H} w_{2rl} Y_1(r) \quad (7)$$

In equation (7)

$$Y_1(r) = \frac{1}{1+exp(-w_{l\,1r}.C_{in})}$$

In the above equation (7) the activation function is evaluated dependent on the hidden and output layer of the network.

Step 3: The neurons weights are evaluated, $w = 1 + \Delta w$, the changes in the weights are stated as Δw as in equation (8)

$$\Delta w = \gamma . Y_2 . BP_{err} \quad (8)$$

In above equation (8) the learning rate of network is stated as γ.

Step 4: The process gets repeated until BP achieves the minimal least value, then continues Step 2 for the criteria. $BP_{err} < 0.1$.

3.2.2 Training Phase with enhanced Weight Optimization Through Modified Cuckoo Search Algorithm (MCSA)

In the training phase, the NN process is incorporated through the use of optimization approach to achieve the optimization weights in the training process. The proposed model uses the MCSA for the superior classification performance and effective recognition, as shown in Figure 4. Once training is done next in the trial stage, the trained NN with processed weights is used, and the output is calculated for the classification for the test dataset.

The process in the MCSA comprises an effective meta-heuristics-based CS algorithm for breeding process implementation. The process comprises the multitude of nests for the egg solution, with the superior replacement in the nest. To find the most appropriate set of weights that minimizes the objective function, the neural network's weights are iteratively modified throughout the training phase utilizing both backpropagation and the MCSA. The objective of this procedure is to enhance the neural network's ability to solve the specified problem.

Step 1: Initialization Step

The population (mi, where i=1, 2, n) of host nest is started randomly.

Step 2: Generation of Cuckoo

By integrating CS with levy flight, the novel value is generated those are engendered for the examination of objective function to ascertaining the solution.

Step 3: Fitness Evaluation Step

The fitness operation is evaluated by consideration of equation (8) and equation (9) with the present value in equation (9)

$$P_{max} = \frac{P_s}{P_T} \quad (9)$$

In above equation (9) P_{max} represents fitness highest popularity value.

At which, P_s denotes the chosen population and P_T presents the sum of population

Step 4: Update

In the update phase the levy flights are employed with the cosine transform for the selected arbitrarily nest quality. The selected quality nest is evaluated for the superior function through replacement of novel Cuckoo solution. The previous solution is evaluated with the employed CS based levy flights as stated in equation (10)

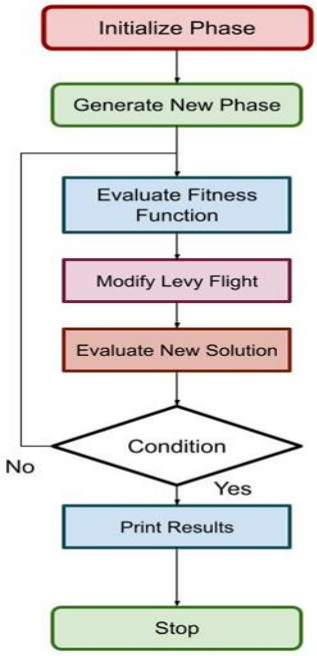


Fig 4. Steps in Modified Cuckoo Search

$$m_i^* = m_i^{(t+1)} = m_i^{(t)} + \alpha \oplus Levy\,(n) \quad (10)$$

Through consideration of above equation (10) the levy equation is evaluated for the gaussian distribution as stated in equation (11)

$$m_i^* = m_i^{(t+1)} = m_i^{(t)} + \alpha \oplus \sigma_s \quad (11)$$

where, $\sigma_s = \sigma_0 \exp(-\mu k)$, the constants are stated as σ_0, μ and present generation symbolizes is represented as K.

Step 5: Reject Worst Nest

The worst nests are eliminated using the potential values and unique values are developed with computation of fitness function to achieve the best solution with ranking process. Through optimal solution estimation best solution is detected and marked.

Step 6: Stopping Criterion

The maximal process of loop is computed based on the optimization function.

3.3 Assessment criteria:

The effectiveness of proposed SDP approach is assessed using some of the metrics, such as execution time and prediction rate. An assessment criterion is utilized to measure the effectiveness of SDP technique and to validate the theoretical and practical progressions of these systems. The prediction is then compared with that of the existing algorithm.

3.3.1 Execution Time

Execution time refers indicates the amount of time a responsible system spends executing the given task, and it includes the time a system spends in executing run-time or system services. Therefore, the programs or algorithm's execution time is established by measuring the elapsed time from inception and termination of execution. The following is an expression of the formula for determining execution time:

$$execution\_time = end\_time - start\_time$$

3.3.2 Prediction rate/ Accuracy

Prediction rate or accuracy, is a statistic measure to assess how well categorization systems work. It displays the proportion of all labels in the dataset that are correctly classified. The following formula can be used to determine the prediction rate or accuracy:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} * 100$$

Table 1: Train accuracy

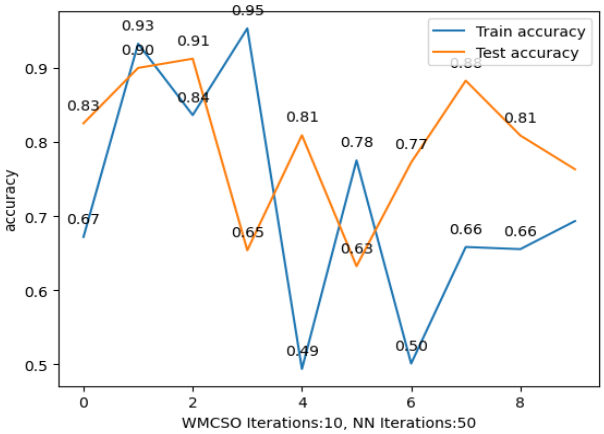
WMCSO Iterations	Max Accuracy		
	NN Iterations = 5	NN Iterations = 20	NN Iterations = 50
10	0.77	0.78	0.95
20	0.76	0.91	0.97
30	0.76	0.93	0.94

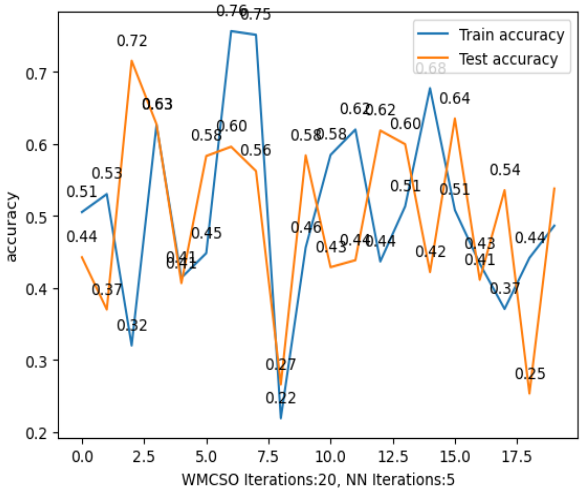
Table 2: Test accuracy

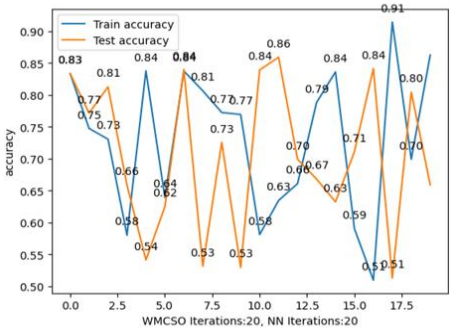
WMCSO Iterations	Max Accuracy		
	NN Iterations = 5	NN Iterations = 20	NN Iterations = 50
10	0.73	0.92	0.91
20	0.72	0.86	0.97
30	0.81	0.86	0.97

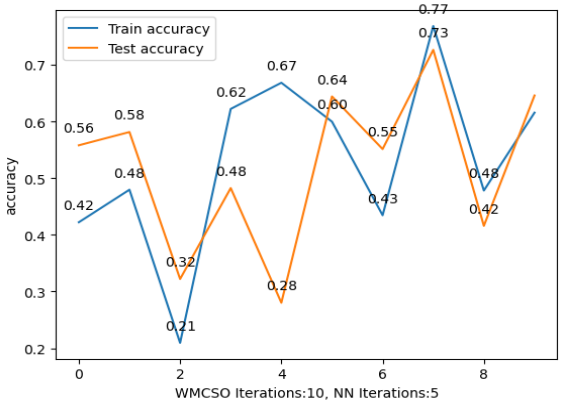
Table 1 presents the training accuracy of the suggested method. The method was tested with varying numbers of WMCSO iterations (10, 20, and 30) and NN iterations (5,20,50). For instance, with 10 iterations, the maximum accuracy ranged from 0.77 to 0.95, increasing with more neural network (NN) iterations. As the number of iterations increased to 20 and 30, the maximum accuracies improved notably, reaching up to 0.97 for 50 NN iterations. This indicates that increasing both the number of WMCSO iterations and NN iterations enhances the performance of the suggested method, with the maximum accuracy gained when utilizing 50 NN iterations under WMCSO 30 iterations.

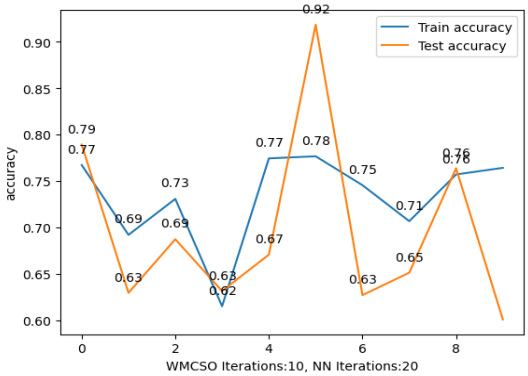
Table 2 illustrates the testing accuracy achieved by the recommended method. The best accuracy of 0.97 was obtained for WMCSO with 20 and 30 iterations when using 50 NN iterations, although the maximum accuracies varied as the number of iterations climbed to 20 and 30. Findings indicate that the planned method works best with WMCSO with 20 and 30 iterations, especially when using 50 NN iterations, which yields maximum accuracies of 0.97.

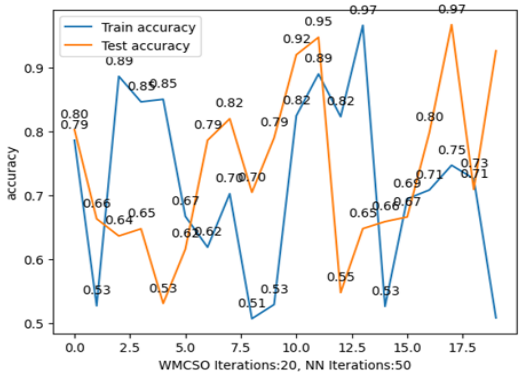












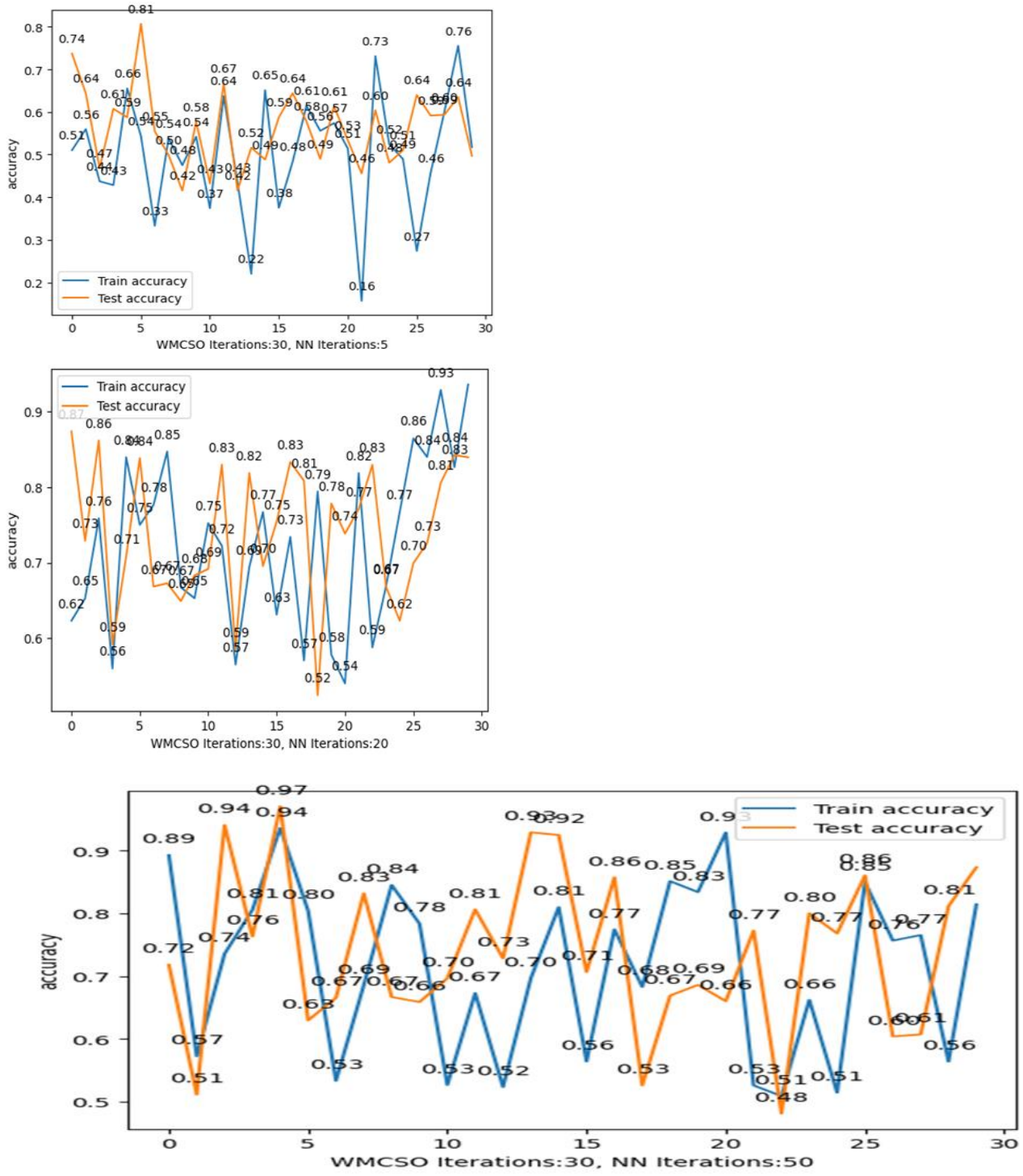


Fig 5: Assessing differences in train and test accuracy levels

Figure 5 shows analyzing train and test accuracy disparities of the model with no of NN hidden layers=1.

Table 3: Train Accuracy

WMCSO Iterations	Train accuracy (NN Hidden Layers)		
	1	2	4
10	0.95	0.95	0.96
20	0.97	0.96	0.97
30	0.94	0.97	0.97

Table 4: Test Accuracy

WMCSO Iterations	Test accuracy (NN Hidden Layers)		
	1	2	4
10	0.91	0.90	0.92
20	0.97	0.97	0.97
30	0.97	0.98	0.98

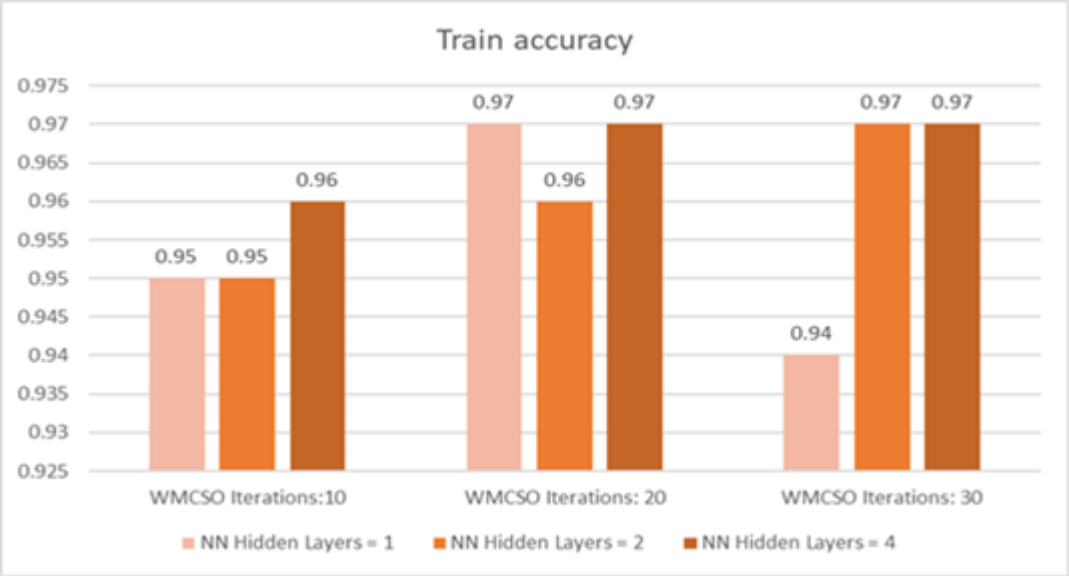


Fig 6: Train accuracy

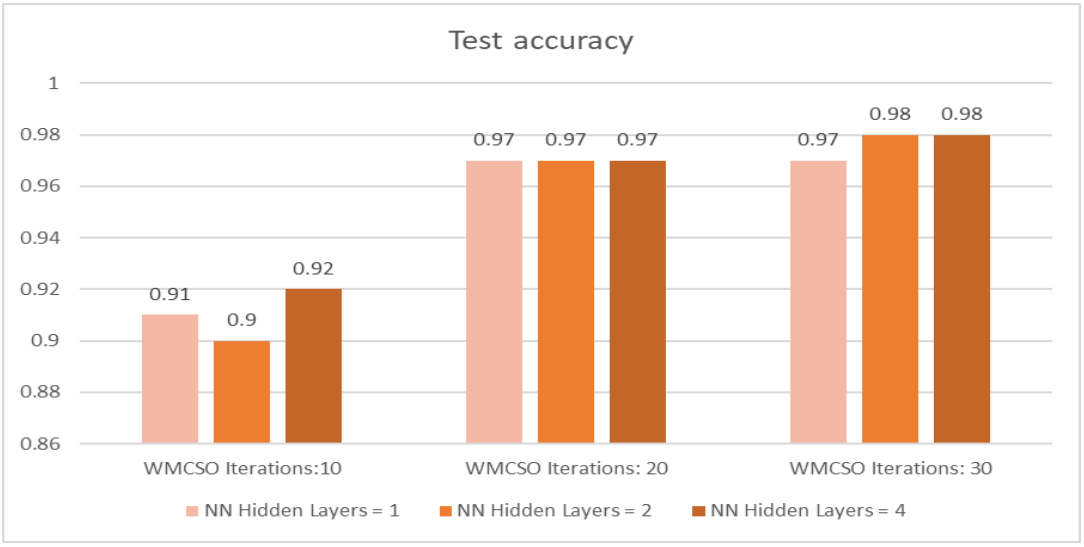


Fig 7: Test accuracy

The train and test accuracies for the model with different no of NN hidden layers are given in Table 3, 4, Figure 6, and 7. The model has been executed for 10, 20, and 30 iterations with the WMCSO algorithm and neural network hidden layers configurations like 1, 2, and 4. For all the iteration count of the WMCSO algorithm, the model achieved the highest train accuracy for 2 or 4 NN hidden layers. Train accuracy slightly varied for the number of iterations, and the highest one is 0.97 for WMCSO 20 iterations no with 1 or 4 NN hidden layers. Test accuracy increased with the Iteration number, and the greatest one is 0.98 when using 2 or 4 NN hidden layers for no of WMCSO iterations 30.

Table 5: Measure of Computation Time

WMCSO Iteration	Time (s)		
	NN Iteration = 5	NN Iteration = 20	NN Iteration = 50
10	30.64	72.70	124.41
20	57.66	108.40	256.66
30	69.84	154.55	397.70

Table 5 and 6 shows the computational time in seconds associated with different configurations of the model. Table 5 focuses on the impact of varying numbers of NN

iterations with one hidden layer, while Table 2 explores the effect of different numbers of NN hidden layers. Both tables show that increasing the complexity of the model (either by adding more iterations or hidden layers) leads to higher execution times.

Table 6: Measurement of Time with Hidden Layer

WMCSO Iteration	Time (s) (NN Hidden Layers)		
	1	2	4
10	124.41	241.56	435.67
20	256.66	456.76	678.70
30	397.70	567.80	754.50

4. Comparative Analysis

Table 7 and 8 compares the time measures and accuracy of the suggested model HNN-WMCSO and existing model HNN. As shown in the tables, the proposed HNN-WMCSO performs better than the existing HNN in all the iterations. For both models, the accuracy is seen to improve as the number of iterations increases. However, the rate of improvement was higher for the planned HNN-WMCSO. On the other hand, the proposed HNN-WMCSO requires more computational time than the existing HNN in all iterations. As the intensity increases, the gap in computational time requirements for the two models becomes significantly larger. Thus, the results indicate that integrating the WMCSO method with the hierarchical neural network model achieves a higher level of accuracy. However, the computational time needed for this model was also high.

Table 7: Comparison of accuracy

WMCSO Iterations	Existing HNN	Proposed HNN-WMCSO
10	0.87	0.92
20	0.90	0.97
30	0.91	0.98

Table 8: Comparison of Time Measures

WMCSO Iterations	Existing HNN	Proposed HNN-WMCSO
10	234.6	435.67
20	447.87	678.70
30	544.34	754.50

5. Conclusion

This study presents a novel approach, termed Hybrid Neural Network with Weighted Modified Cuckoo Search Optimization (HNN-WMCSO), for SDP utilizing a soft computing framework. The research model recombines several computational approaches, such as neural networks and optimization algorithms. This, in turn, may provide a closer-to-ideal solution to the issue of properly predicting defects, ensuring both accuracy and reliability in software systems. Specifically, using MFCM for data pre-processing, HNN for defect prediction, and WMCSO for optimizing neural network weights, the HNN-WMCSO research model illustrates a better form of performance, especially at prediction accuracy, when contrasted with other models. The empirical analysis demonstrates that the suggested approach is effective as it helps improve software quality and reliability by capturing defects early before their manifestation in production setups. In general, the current research emphasizes the role of using a hybrid SC-based approach to enhance efforts to address the complexities apparent in SDP. This enhances the use of more reliable software systems in the future.

References

[1] Matloob, F., Ghazal, T. M., Taleb, N., Aftab, S., Ahmad, M., Khan, M. A., ... & Soomro, T. R. (2021). Software defect prediction using ensemble learning: A systematic literature review. IEEE Access, 9, 98754-98771.

[2] Thota, M. K., Shajin, F. H., & Rajesh, P. (2020). Survey on software defect prediction techniques. International Journal of Applied Science and Engineering, 17(4), 331-344.

[3] Pachouly, J., Ahirrao, S., Kotecha, K., Selvachandran, G., & Abraham, A. (2022). A systematic literature review on software defect prediction using artificial intelligence: Datasets, Data Validation Methods, Approaches, and Tools. Engineering Applications of Artificial Intelligence, 111, 104773.

[4] Alsawalqah, H., Hijazi, N., Eshtay, M., Faris, H., Radaideh, A. A., Aljarah, I., & Alshamaileh, Y. (2020). Software defect prediction using heterogeneous ensemble classification based on segmented patterns. Applied Sciences, 10(5), 1745.

[5] Li, N., Shepperd, M., & Guo, Y. (2020). A systematic review of unsupervised learning techniques for software defect prediction. Information and Software Technology, 122, 106287.

[6] Zheng, W., Shen, T., Chen, X., & Deng, P. (2022). Interpretability application of the Just-in-Time

software defect prediction model. Journal of Systems and Software, 188, 111245.

[7] Stradowski, S., & Madeyski, L. (2022). Machine learning in software defect prediction: A business-driven systematic mapping study. Information and Software Technology, 107128.

[8] Aljamaan, H., & Alazba, A. (2020, November). Software defect prediction using tree-based ensembles. In Proceedings of the 16th ACM international conference on predictive models and data analytics in software engineering (pp. 1-10).

[9] Azzeh, M., Elsheikh, Y., Nassif, A. B., & Angelis, L. (2023). Examining the performance of kernel methods for software defect prediction based on support vector machine. Science of Computer Programming, 226, 102916.

[10] Ali, U., Aftab, S., Iqbal, A., Nawaz, Z., Bashir, M. S., & Saeed, M. A. (2020). Software defect prediction using variant based ensemble learning and feature selection techniques. International Journal of Modern Education & Computer Science, 12(5).

[11] Sharma, T., Jatain, A., Bhaskar, S., & Pabreja, K. (2023). Ensemble Machine Learning Paradigms in Software Defect Prediction. Procedia Computer Science, 218, 199-209.

[12] Ning Li , Martin Shepperd , Yuchen Guo (2020). A systematic review of unsupervised learning techniques for software defect prediction. Information and Software Technology. Volume 122, 106287.

[13] Qiao, L., Li, X., Umer, Q., & Guo, P. (2020). Deep learning based software defect prediction. Neurocomputing, 385, 100-110.

[14] Gorkem Giray, Kwabena Bennin, Omer Koksal,

[15] Onder Babur, Bedir Tekinerdogan(2023). On the use of deep learning in software defect prediction. Journal of systems and software. Volume 195, 111537.

[16] Tong Yu, Hong Zhu (2020). Hyper-Parameter Optimization: A Review of Algorithms and Applications.cs-arXiv:2003.05689.

[17] Shifei Ding, Li Xu, Chunyang Su, Hong Zhu (2010). Using Genetic Algorithms to Optimize Artificial Neural Networks. Journal of Convergence Information Technology 5(8):54-62.

[18] Nazri Mohd. Nawi, Abdullah Khan, and Mohammad Zubair Rehman (2013). A New Back-Propagation Neural Network Optimized with Cuckoo Search Algorithm. B. Murgante et al. (Eds.): ICCSA 2013, Part I, LNCS 7971, pp. 413–426, 2013.

[19] Omer Faruk Arar, Kurşat Ayan (2015). Software defect prediction using cost-sensitive neural network.Applied Soft Computing.Volume 33, Pages 263-277

[20] Sofian Kassaymeh, Mohamad Al-Laham , Mohammed Azmi Al-Betar, Mohammed Alweshah, Salwani Abdullah, Sharif Naser Mahkadmeh (2022). Backpropagation Neural Network optimization and software defect estimation modelling using a hybrid Salp Swarm optimizer-based Simulated Annealing Algorithm. Knowledge-Based Systems. Volume 244, 23 May 2022, 108511.

[21] Mengtian Cui, Yameng Huang, Jing Luo (2019). Software Defect Prediction Model Based on GA-BP Algorithm. Cyberspace Safety and Security: 11th International Symposium, CSS 2019.

[22] Xingjuan Cai, Yun Niu, Shaojin Geng, Jiangjiang Zhang, Zhihua Cui, Jianwei Li, Jinjun Chen (2019). An under-sampled software defect prediction method based on hybrid multi-objective cuckoo search. Concurrency and computation practice and experience. Volume 32, issue 5 e5478.

[23] Dr. S Balasubramaniam, Dr. Shantappa G Gollagi (2022). Software defect prediction via optimal trained convolutional neural network. Advances in Engineering Software.Volume 169, 103138.

[24] Safial Islam Ayon (2019). Neural Network based Software Defect Prediction using Genetic Algorithm and Particle Swarm Optimization. 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT).

[25] Zhen Li, Tong Li, YuMei Wu, Liu Yang, Hong Miaoand DongSheng Wang (2021). Software Defect Prediction Based on Hybrid Swarm Intelligence and Deep Learning. Computational Intelligence and Neuroscience. 2021; 2021: 4997459.