Received February 6, 2021, accepted March 31, 2021, date of publication April 12, 2021, date of current version April 27, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3072380

Metaheuristic Algorithms in Optimizing Deep Neural Network Model for Software Effort Estimation

MUHAMMAD SUFYAN KHAN[1], FARHANA JABEEN[1], SANAA GHOUZALI[2], ZOBIA REHMAN[1], SHENEELA NAZ[1], AND WADOOD ABDUL[3]

[1]Department of Computer Science, COMSATS University, Islamabad 44000, Pakistan
[2]Department of Information Technology, College of Computer and Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia
[3]Department of Computer Engineering, College of Computer and Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia

Corresponding author: Sanaa Ghouzali (sghouzali@ksu.edu.sa)

This work was supported by a grant from the ''Research Center of the Female Scientific and Medical Colleges'', Deanship of Scientific Research, King Saud University.

ABSTRACT Effort estimation is the most critical activity for the success of overall solution delivery in software engineering projects. In this context, the paper's main contributions to the literature on software effort estimation are twofold. First, this paper examines the application of meta-heuristic algorithms to have a logical and acceptable parametric model for software effort estimation. Secondly, to unravel the benefits of nature-inspired meta-heuristic algorithms usage in optimizing Deep Learning (DL) architectures for software effort estimation, this paper presents a Deep Neural Network (DNN) model for software effort estimation based on meta-heuristic algorithms. In this paper, Grey Wolf Optimizer (GWO) and StrawBerry (SB) meta-heuristic algorithms are applied for having a logical and acceptable parametric model for software effort estimation. To validate the performances of these two algorithms, a set of nine benchmark functions having wide dimensions is applied. Results from GWO and SB algorithms are compared with five other meta-heuristic algorithms used in literature for software effort estimation. Experimental results showed that the GWO has comprehensive superiority in terms of accuracy in estimation. The proposed DNN model (GWDNNSB) using meta-heuristic algorithms for initial weights and learning rate selection, produced better results compared to existing work on using DNN for software effort estimation.

INDEX TERMS Software effort estimation, meta-heuristic algorithms, deep neural networks, deep learning.

I. INTRODUCTION

Software project development is comprised of a different set of activities from requirement gathering to testing and maintenance; that need to be carried out in a specified time and budget [1]. Software engineering is based on logical and analytical work. Compared to other kinds of engineering projects, software development is more complex because of the high rate of change in customer requirements and rapid advancement in technology. Therefore, for effective software project management, it becomes a challenge to achieve specific objectives while satisfying a range of constraints [2]. The rapid trend in hardware advancement created a situation where the hardware is cheap as compared to the programmer. No matter how fast the hardware is, for tangible performance improvement, it is required to optimize the software too, which in return puts the burden on the developers to develop adaptable software in a restricted time and budget, to bridge the gap between hardware and software development. The world of technology is one such place that is advancing at a very fast pace. The current era has already witnessed the emergence of trends in software development ranging from artificial intelligence [9], mobile computing [56], blockchain, virtual reality [61], autonomous vehicles [54], and cyber security. Moreover, with the advancement in the trend of network technologies (e.g., 5G [59], and Information-centric networking [60]), and applications (e.g., intelligent transportation system [54], [56], Internet of Things (IoT), autonomous vehicles [54], mobile computing [56]), there is a rise in the trend towards providing software-defined architectures [58] to simplify management and enable innovation through programmability. Planning and estimation activities are particularly challenging in large and distributed projects.

The associate editor coordinating the review of this manuscript and approving it for publication was Alicia Fornés.

Measuring the sophistication of the software in the initial development stages and making the required estimations precisely is challenging, both for the project managers and developers. Software cost and effort estimation incorporate processes to determine project cost, effort in terms of man-hours, and time duration to complete a project [3], [8]. Project effort underestimation may result in the surpassed budget, underdeveloped or limited functions, mediocre quality, and failure to timely complete the project [5]. Overestimating will result in wastage of resources due to the commitment of too many resources to the project. Therefore, precise time, cost, and effort estimation are not only considered as the critical factors for the success of a project, but also effective in business decision-making. For the success of the software the project, it is necessary to accurately comprehend the requirements such as (i) impacts on the system design, (ii) to determine performance bounds due to hardware advancement, (iii) set forth assumptions on the new environment, (iv) set forth dynamic changes to customer requirements, and (iv) seek for the good trade-off.

Existing research has presented several techniques for cost and effort estimation. Despite the struggle, precise effort estimation is still an open issue. Existing research efforts related to software effort estimation can be broadly divided into two groups: (i) non-algorithmic, and (ii) algorithmic [7]. Techniques based on non-algorithmic models [8], [9] include analogy-based estimation, expert judgment, Fuzzy Logic (FL) [14], learning-based techniques (Artificial Neural Network (ANN)) [9], [40], Machine Learning (ML) [11], [12], Case-Based Reasoning (CBR) [13]. Advancement in the process of software development evolved over the last decades. As a result, most of the available datasets are heterogeneous, as their sources are from a variety of organizational projects and another aspect is a large number of missing values. Multi-objective and high dimensional data classification using Neural Network (NN) is a challenging task. In algorithmic models [17] effort estimation is provided by mathematical/statistical models that utilize the product, project, and process-related attributes. Examples include Putnam's, Function Point Analysis (FPA), COnstructive COst MOdel (COCOMO), Software Evaluation and Estimation of Resources - Software Estimation Model (SEERSEM), Software Lifecycle Model (SLIM) [18]. Effort estimation (related to new software development) is based on the historical data from previous software projects. COCOMO [18]–[20] is one of the most used regression-based algorithmic models for software effort estimation. COCOMO divides projects into three categories based on certain features such as size, staff experience, and software type. COCOMO considers the project category while assigning initial values to the parameters. In reality, the available software project datasets contain information about projects of heterogeneous nature (varying project metrics in terms of size and properties). Therefore, having a single, logical, and acceptable parametric model is difficult. Therefore, for estimation accuracy, it is required to fine-tune the parameters [19].

To overcome the limitations of COCOMO models, existing work on software effort estimation employ different methods to tweak the COCOMO coefficients and improve effort estimation. The rising trend of using meta-heuristic algorithms has been witnessed in the recent literature to tweak COCOMO coefficients as well as to optimize software effort estimation [30]–[34]. These meta-heuristic algorithms performed well in dealing with the optimization problem in different domains-of-interest [36]–[38], [42], because of their unique features including large searching space and random selection technique.

Sehra *et al.* [21], reviews the existing literature related to software effort estimation published in the period 1996-2016. The authors reported that over the last two decades, there is a rise in the research efforts related to ML-based software development effort estimation. Research community is now considering Deep Learning (DL) as a prospective solution for improving cost estimation. Deep Neural Network (DNN) allows to represent complex relationships between effort and cost drivers, thus making it better choice for software cost estimation.

A. PROBLEM FORMULATION

The main drawback of existing literature [21]–[35] is that it is very difficult to figure out which meta-heuristic algorithm provides better accuracy in estimating software effort. The main reasons behind unpredictability in the performances of the meta-heuristic algorithms are as follows. Firstly, the dataset used by each existing work in the model evaluation is different. The available public datasets are heterogeneous in terms of size, and a variety of organizational projects data. The performance of the meta-heuristic algorithm might be excellent based on one dataset, but unfavorable or lower based on other datasets. Secondly, the performance measures used for validation are different. In reality, the performance of the algorithm might be superior on one performance measure such as the Mean Magnitude of Relative Error (MMRE), but it might be unfavorable based on performance measures such as the Effect Size, and Standardized Accuracy (SA). Thirdly, performance evaluation, and experimental setup varies in the existing relevant literature.

The DNN architectures although provides several improvements over existing shallow techniques, but they are also suffered from several shortcomings including large training delays, overfitting, and underfitting. Obtaining accurate DNN model within the moderate time is one of the critical challenges, especially when there are many parameters in model configuration and high feature space dimensionality in the training dataset [39], [42]. Meta-heuristic algorithms support finding better solutions at a reasonable computational cost. Further, such algorithms heuristics allows for finding a solution close to optimum, therefore, can be used with DNN to reduce large training delays [42]. Meta-heuristic algorithms have been implemented in DNN training [40]. Tian *et al.* [43] review existing work on using meta-heuristic algorithms for traditional NNs training and parameter optimization in different domains. Little attention has been paid by the research community related to the diffusion of nature-inspired algorithms in DL for software effort estimation. This provides a window of opportunity to evaluate performance gains that can be achieved by employing meta-heuristic algorithms to DNN for software effort estimation. Considering these limitations, the main contributions of this work are as follows:

B. CONTRIBUTIONS

1) The meta-heuristic algorithms Grey Wolf Optimization (GWO) [36], [47] and StrawBerry SB) algorithm [37], [38], [68]–[70], support solving multi-variable problems, and have been applied for the optimization of various engineering problems. In this paper, GWO and SB meta-heuristic algorithms are applied for having logical, and acceptable parametric models for software effort estimation. To validate the performances of these two algorithms, a set of nine benchmark functions having wide dimensions is applied.
2) The results obtained from GWO and SB algorithms are compared with five other meta-heuristic algorithms used in the literature for software effort estimation. We selected five widely used nature-inspired algorithms (BAT [29], [45], Cuckoo Optimization (CO) [35], [53], [54], Genetic Algorithm (GA) [22], [30], [33] and Ant Colony Optimization (ACO) [24], [32], Particle Swarm Optimization (PSO) [27], [34], [46]) for comparison. In this work, for comparison analysis nature-inspired meta-heuristics algorithms are selected based on inspiration from: (i) Natural biological system (GA, SB), (ii) Theory of evolution (PSO), (iii) Insects activities (ACO), (iv) Group behavior of animals, and birds (GWO, CO, BAT).To validate the performances of these seven algorithms, a set of nine benchmark functions having wide dimensions is applied. GWO and SB algorithms have been applied by the research community for solving complex problems in other fields of science and engineering, but not exploited for software effort estimation.
3) We propose a DNN model (named as GWDNNSB) that exploits meta-heuristic algorithms to optimize the weights and learning rate. GWO algorithm is used for initial weights optimization and SB for the learning rate optimization.

The rest of paper is organized as follows: Section 2 presents related work, section 3 presents the comparative analysis of seven widely used nature-inspired algorithms. The proposed GWDNNSB is discussed in Section 4. Section 5 discusses the conclusion and future work.

II. STATE OF THE ART

Many researchers around the world used different techniques to optimize software estimation methods [17]. There exists work that reviews existing literature related to software cost estimation [19]. For software effort estimation several meta-heuristic algorithms have already been implemented over the last decade [20], [21], [22], [25] to tune the COCOMO parameters. For example, Bee Colony Optimization (BCO) [20], Differential Evolution (DE) [21], Genetic Algorithm (GA) [22], Fire fly [25], Harmony Search (HS) [26], Particle Swarm Optimization (PSO) [27] improved effort estimation by optimizing COCOMO parameters. Venkataiah *et al.* [24] exploited Ant Colony Optimization (ACO) technique to optimize the prediction of software cost estimation.

There exists work that uses hybrid method to optimize software cost and effort estimation. Hybrid method combines two or more methods. Ahmed *et al.* [23] presented Whale-Crow Optimization (WCO) algorithm for software cost estimation. WCO integrates Whale Optimization Algorithm (WOA) and the Crow Search Algorithm (CSA) to find the optimal regression coefficients for a regression model. BATGSA hybrid algorithm based on BAT and Gravitational Search Algorithm (GSA) is presented in [29] to obtain better software estimation. BAT algorithm determines the routing and hunting behavior of bat in the exploration phase which is further improved by using gravitational force effect of the GSA to speed up the searching speed of the BAT. Better estimation is achieved compared to COCOMO model. There exists work, which exploits a different combination of meta-heuristic algorithms to optimize cost and effort estimation. For example, Tabu Search and GA [30]; Partial Swarm Optimization (PSO), Genetic Algorithm (GA) and invasive Weed Optimization Algorithm (WOA) are implied synthetically to improve the COCOMO process model [31]; Ant Colony Optimization (ACO) and Chaos Optimization Algorithm (COA) [32]; GA and Artificial Bee Colony (ABC) [33]; PSO and DE [34]; Cuckoo Search (CS) and Harmony Search (HS) algorithms [35] for optimizing COCOMO-II coefficients.

Currently, ANN is widely used for software effort estimation [39], and is an active research area [12]. Ali *et al.* [39] performed a systematic review of existing work that uses ML techniques to build software effort estimation models. Comparative performance analysis of four NN models is performed in [40]: (i) General Regression Neural Network (GRNN) (ii) Cascade Correlation Neural Network (CCNN) (iii) Multilayer Perceptron (MLP), (iv) Radial Basis Function Neural Network (RBFNN). The work demonstrates that GRNN outperforms compared to other NN models. In [41] random forest technique is investigated for software effort estimation. In this work, the impact on accuracy is evaluated by varying the following parameters values: (i) trees count, and (ii) the attributes count required to grow the tree. In addition, this work optimized the random forest model by selecting the optimal values for these two parameters, considering COCOMO, ISBSG, and Tukutuku datasets. In [11] feed-forward back-propagation multilayer neural network model is used to improve COCOMO Model. Kaushik *et al.* [45] presented hybrid model of Wavelet Neural

Networks (WNN) and meta-heuristic technique for effort estimation. The Firefly and BAT meta-heuristic techniques are considered with WNN. The work demonstrates that integrating WNN with Firefly and BAT techniques produced better estimation.

ANN is associated with several challenges including falling in local minima, and overfitting. Therefore, many researchers recommend using nature-inspired meta-heuristic algorithms for the following: (i) ANN training [46], [53], (ii) feature selection in classification [52], and (iii) weights selection [47], [49]. Wani *et al.*, [46] used functional link ANN model for software effort estimation. PSO algorithm is used for optimizing training by improving the candidate solution iteratively. Emary *et al.* [47] use a modified GWO algorithm that utilizes the reinforcement learning principles, to optimize the selection of weights for neural networks and selecting relevant features for model building. Kodmelwar *et al.* [49] proposed a DL modified NN technique (Deep MNN) using CS Algorithm for initializing the network weights and HPSO for better classification of dataset parameters. CS is used to initialize the weights of the network. The input to this Deep MNN is Effort multipliers i.e. software development, database size, constant value, exponent value, etc. of the COCOMO dataset. This step is followed by the optimization process, which is performed by using HPSO with genetic operators. The proposed technique shows a decline in execution time as compared to traditional NN when 10 instances were considered. But the increase in the number of instances (beyond 50), resulted in an increase in execution time. In [53] ANN model is trained using the CO algorithm to predict software cost estimation.

Usman *et al.* [1] conducted an exploratory longitudinal case study to identify research efforts towards effort estimation related to distributed agile projects. The work demonstrates that team maturity, work distribution, and priorities are the key factors impacting effort estimation accuracy. A neighborhood fuzzy PSO algorithm is proposed to train the ANN-based software reliability model to better predict software reliability [78].

In [55] in order to optimize the fuzzy model for software cost estimation, the PSO is used to optimize the parameter values of model membership functions. Kaushik *et al.* [57] uses ANN and whale optimization algorithm (WOA) to provide an effort estimation method for agile software development. The work compares the performance of Radial Basis Function Neural Network (RBFN) and Functional Link Artificial Neural Network (FLANN).

Now we will discuss the latest research regarding Neural Network and optimization methods. To overcome the problems faced by traditional ANNs, Dendritic Neuron Model (DNM) is presented that exploits the nonlinearity of synapses to solve with high precision complex problems [73]. Dendrites are tree like projections responsible for receiving impulse. In [73] the authors uncover the most appropriate learning algorithm to train DNM among six algorithms, i.e., Evolutionary Strategy (ES), Biogeography-Based Optimization (BBO), PSO, GA, Population-Based Incremental Learning (PBIL), and ACO.

For industrial and big data-related applications efficiently extracting useful knowledge from a High-Dimensional and Sparse (HiDS) matrix is critical. Although, the HiDS matrices are sparse but it do contain meaningful information about involved entities. To extract useful patterns from HiDS, the Nonnegative Matrix Factorization (NMF) model has been widely used. NMF model factorizes HiDS matrices into low dimensional Nonnegative Latent Factor (NLF) matrices. However, one of the limitations of existing NLF models is their dependency on specifically designed learning schemes. In [74] the authors proposed a scheme which supports efficient extraction of NLF from HiDS matrices, and no restriction on training scheme selection.

With the rapid development of cloud, Web, and IoT-based services systems, choosing the most reliable service provider for a particular service is a major challenge. The Quality-of-service (QoS) provided by the service provider may vary over time. In [75], Biased Non-negative Latent Factorization of Tensors (BNLFTs) model is introduced to capture the temporal patterns hidden in data for QoS prediction.

DL techniques, in particular convolutional neural networks, have been proven to be the state-of-the-art technology in the machine vision field. Deep learning model typically requires fine tuning many parameters and therefore efficient learning algorithms should be incorporated to reduce training delays. One of the recently introduced solutions is imitation learning [76], providing efficiency in terms of computing and the knowledge needed for training process. A machine is needed to observe human behaviour in order to learn how to accomplish the specific task. To generate robust and effective models, an imitation approach requires careful design of learning algorithm. In [76] the authors give informative insight into imitation learning methodology for fast response and time-critical applications, such as an autonomous driving

The current NNs implementations based on Von Neumann computing architecture using the Complementary Metal Oxide Semiconductor (CMOS) technology suffer from memory and communication bottleneck problems. These days memristor-based systems draw great attention from the research community and industry in various areas including neural networks, reconfigurable computing, and artificial intelligence. Memristor, deliver energy-efficient neuromorphic computing, exploiting nanosolid state nonvolatile resistive switching [77]. NNs based on memristor crossbar architecture supports efficient neural calculations due to its backing for parallel and high density computing. Memristive Recurrent Neural Networks (MRNNs) is an emerging topic of study. Linear resistors used in classic RNNs circuits are replaced by memristors in MRNNs [77].

III. SOFTWARE EFFORT ESTIMATION USING META-HEURISTIC ALGORITHMS

In this work, for comparison analysis, nature-inspired meta-heuristics algorithms are selected based on inspiration from the following: (i) Natural biological system (GA, SB), (ii) Theory of evolution (PSO), (iii) Insects activities (ACO), (iv) Group behavior of animals, and birds (GWO, CO, BAT). Such, meta-heuristic algorithms are also used in the literature to solve complex problems, especially in the field of science and engineering [42], [43]. Table 1 presents the weaknesses and strengths of these algorithms.

A. BRIEF OVERVIEW OF META-HEURISTIC ALGORITHMS

This section will give a brief overview of the meta-heuristic algorithms that are selected for comparison.

1) GREY WOLF ALGORITHM

Grey wolf optimization is unprecedented nature inspired algorithm, which is based on the natural behavior of grey wolves. It is valuable for searching optimized solutions [36], [47]. Grey wolves live in packs and based on their behavior categorized in four types namely Alpha (α), Beta (β), Delta (δ) and Omega (ω) respectively. Alpha wolf (α) act as the group leader and is held responsible for decision making. The α, β, δ represents the top three fittest solutions, and the remaining wolves are referred to as ω.

Grey wolf optimization algorithm is used to solve the optimization problems [36], [47]. In the initial stage, the search agents are initialized. Values are assigned to α, β, δ and ω by fitness. In the hunting stage the wolves encircle their prey, which is governed by Eq. 1, Eq. 2, Eq. 3, Eq. 4.

$$\vec{D} = |\vec{C}.\overrightarrow{Pos_p}(t) - \overrightarrow{Pos}(t)| \tag{1}$$

$$\overrightarrow{Pos}(t+1) = \overrightarrow{Pos_{(p)}}(t) - \overrightarrow{CV_x}.\vec{D} \tag{2}$$

The grey wolf position is represented by the vector $\overrightarrow{Pos}$, and $\overrightarrow{Pos_{(p)}}$ represents a prey position vector. The constant t represents the current time. The vectors $\overrightarrow{CV_x}$ and $\overrightarrow{CV_y}$ represents coefficients vectors and are calculated as follows:

$$\overrightarrow{CV_x} = 2\vec{a}.\vec{r}_1 - \vec{a} \tag{3}$$

$$\overrightarrow{CV_y} = 2.\vec{r}_2 \tag{4}$$

The values to $\vec{r}_1$ and $\vec{r}_2$ are assigned randomly in the range [0, 1]. Over the progression of iterations the controlling parameter $\vec{a}$ is defined to decrease linearly in the range [2, 0]. The α, β, δ wolves guide for the hunting process. ω wolves follow the leader wolf. The hunting process starts after the encircling process

$$\vec{D}_\alpha = |\overrightarrow{CV_{y_1}}.\overrightarrow{Pos_\alpha} - \overrightarrow{Pos}| \tag{5}$$

$$\vec{D}_\beta = |\overrightarrow{CV_{y_2}}.\overrightarrow{Pos_\beta} - \overrightarrow{Pos}| \tag{6}$$

$$\vec{D}_\delta = |\overrightarrow{CV_{y_3}}.\overrightarrow{Pos_\delta} - \overrightarrow{Pos}| \tag{7}$$

where $\overrightarrow{Pos_1}$, $\overrightarrow{Pos_2}$, $\overrightarrow{Pos_3}$ are defined as follows:

$$\overrightarrow{Pos_1} = \overrightarrow{Pos_\alpha} - \overrightarrow{CV_{x_1}}.(\vec{D}_\alpha) \tag{8}$$

$$\overrightarrow{Pos_2} = \overrightarrow{Pos_\beta} - \overrightarrow{CV_{x_2}}.(\vec{D}_\beta) \tag{9}$$

$$\overrightarrow{Pos_3} = \overrightarrow{Pos_\delta} - \overrightarrow{CV_{x_3}}.(\vec{D}_\delta) \tag{10}$$

TABLE 1. Comparison of meta-heuristic algorithms based on strengths and weaknesses.

Algorithm	Strengths	Limitations
GWO	• Implementation simplicity • Excellent exploitation strategy. • Less control parameters • Strong global optimization	GWO faces problems including poor exploration capability, falls into local optimum, lack of population diversity, and slow convergence.
SB	• It explores several areas of the search space at the same time and thereby can break away from local optimal and achieve the global optimum • The number of computing agents varies from beginning to end. SB algorithm discards half the weak agents and duplicates good agents at each iteration. • Re-initialization strategy in case of trapping in local optimum	Precision depends on initial parameters and maximum iteration number.
CS	• Global search ability of CS is strong. • CS can find a way to converge to the true global optimum.	Local search ability is not strong contributing to slow convergence, and limited convergence precision.
GA	• Finding a global best solution in many problems. • Easy combining with other algorithms.	Genetic Algorithm (GA) faces problems including complex operators usage for selection and cross over, premature convergence rate, taking long run-time, trapping into local optima, and weak local search.
ACO	• Best for solving graph problems and combinatorial optimization.	ACO algorithm faces problems including stagnation, local optimum problem, slow convergence, and lengthy search time.
PSO	• Better global search scheme • Computational efficiency • Easy implementation	PSO faces problems including slow convergence rate, parameter selection problem, easily trapped in a local optimum due to poor exploration, imbalance relationship between exploitation (local search) and exploration (global search), and lose a population diversity quickly.
BAT	• Non-sensitive to initial values • Better convergence speed • Easy to understand and implement	BAT faces problems including high probability of being trapped in local optima as it is local search algorithm, and imbalance relationship between exploration and exploitation.

$$\overrightarrow{Pos}(t+1) = \frac{\overrightarrow{Pos_1} + \overrightarrow{Pos_2} + \overrightarrow{Pos_3}}{3} \tag{11}$$

where $\overrightarrow{Pos_\alpha}$, $\overrightarrow{Pos_\beta}$, $\overrightarrow{Pos_\delta}$ represents the three best solutions. It is considered that leader wolves have superior information about prey position. The controlling parameter a changes $\overrightarrow{CV_x}$, causing the ω wolves to come near or run away from α, β, δ wolves. When $|\overrightarrow{CV_x} > 1|$, the wolves diverge from the prey and start searching for better prey. When $|\overrightarrow{CV_x} < 1|$, the wolves approaches the prey.

Max_{Iter} is used to control searching for better prey. We must take care of computing time as it increases with an increase in Max_{Iter} and can also cause over-fitting. In the optimization process to search globally or locally, the grey wolves are governed by the control parameter. At the start of the search probability of global search is greater; whereas when it is nearing the optimum, local search potential is expected to be greater. When the iterations start executing towards a maximum limit N, a declines linearly from two to zero. However, a maximum admissible error (MAE) is used to end the optimization.

$$a = 2 - t \times \frac{2}{Max_{Iter}} \tag{12}$$

2) STRAWBERRY ALGORITHM

In the case of Strawberry plant, plant movement is governed by Strawberry plant reproduction. The mother plant propagates with the help of a runner; creeping stalk yield in leaf axils and matures out of the plant. Initially, the runner has fewer roots but after acquiring sufficient growth, the daughter plants can grow individually after separating them from the mother plant. Afterward, the daughter plant grows individually; a new runner comes out on it making it a mother plant. After sufficient growth, the runner will become a new daughter plant and so on [37], [38], [68]–[70]. In StrawBerry (SB) algorithm, initially N random points (representing mother plants) are generated in the problem domain. Each mother plant generates two random points: (i) one very close to itself termed as root, (ii) one far away from itself representing the runner. Roots help in searching around the mother plant locations whereas runners help to jump over the local minimums. Therefore, runners' computational agents move with large random steps in the problem domain compared to roots computational agents. The idea exploited here is that plants located in good spots propagate by generating runners close to themselves, whereas those located in poor spots propagate by generating runners in places far from themselves.

The SB considers a set of N, M-dimensional vector $x_k^{mother} = x_{k,1}^{mother}, x_{k,2}^{mother}, \ldots, x_{k,M}^{mother}$, $k = 1, 2, \ldots, N$. At each evaluation i, the daughter plant $x_k^{daughter}(i)$ is generated by the $x_k^{mother}(i)$ randomly. The best mother plant $x_k^{mother-best}(i)$ generates self-equivalent daughter plant, as shown in the Eq. 13 [68], [69].

$$x_k^{daughter}(i) = \begin{cases} x^{mother-best}(i) & \text{k = 1} \\ x_k^{mother}(i) + d_{runners} \times \vec{r_k} \end{cases} \tag{13}$$

$x_k^{mother-best}$ represents the best daughter of the i-1 evaluation chosen using elite selection. r_k values are selected randomly in the range $[-0.5, 0.5]$, and $d_{runners}$ represents maximum distance between daughter and the mother plants.

The best $x_k^{daughter}$ is deemed as a daughter as well as mother plant in the i iteration (Eq. 13). As represented in Eq. 14 the global search is effective, if compared to the best daughter of the $(i-1)^{th}$ iteration at least one of the daughter plant improves the value of cost function.

$$\left| \frac{{}_{k=1,2..N} minf(x_k^{daughter}(i)) - {}_{k=1,2..N}\, minf(x_k^{daughter}(i-1))}{{}_{k=1,2,3..N} minf(x_k^{daughter}(i-1))} \right| \geqslant \theta \tag{14}$$

In case the global search is not successful, then a local search needs to be carried out. To search around the location of mother plant roots are used, whereas to search the farther locations from the mother plant runners are used. Therefore, to simulate the function of roots, local search is carried with random large steps [69], [70]. Whereas to simulate the function of runners, the search is carried out with random small steps [69], [70]. In the scenario, where local search is required, for every next evaluation, the mother plant is selected among the daughters using a combination of elite and roulette wheel selection. Eq. 15 represents the elite selection.

$$x^{mother-best}(i+1) = x^{daughter-best}(i) \tag{15}$$

The fitness of the k^{th} daughter plant is computed using Eq. 16 before the selection of the remaining mother plants. The value of α is a real constant, where $\alpha \in (0, 1)$.

$$fitness(x_k^{daughter}(i)) = \frac{1}{\alpha + f(x_k^{daughter}(i)) - f(x^{daughter-best}(i))} \tag{16}$$

where α is the real constant. Eq. 17 represents the probability of the selecting k^{th} daughter plant of the current evaluation as the mother of the next evaluation.

$$p_k = \frac{fit(x_k^{daughter}(i))}{\sum_{j=1}^{N} fit(x_j^{daughter}(i))} \tag{17}$$

3) CUCKOO OPTIMIZATION

Cuckoo Search (CS) algorithm [35], [53], [54] was proposed by Xin-she Yang and Suash Deb in 2009. This optimization method is based on brood parasitism of cuckoo along with aspects of Levy Flight. The main feature of CS is the Levy Flight random walk which is used to generate new candidate solutions that solve optimization problems. In the Levy Flight mechanism, the step lengths are computed based on a heavy-tailed probability distribution [71].

Cuckoo Search mechanism used following three rules to select the optimal solution [72]: (i) Each cuckoo at a time t lays one egg and randomly select a nest to dump its egg; (ii) best nests with good-quality eggs (i.e. better solutions) will be passed on to the next generation; (iii) host cuckoo will discover an alien egg with a probability of 0-1 when available host nests are fixed.

CS algorithm begins with the cuckoo's initial population. These initial cuckoos lay eggs in other host nests. If the host birds do not identify those eggs and do not remove then they grow. In host nests, each egg provides a solution and a cuckoo creates a new solution. Therefore, the goal is to iteratively replace the worse solution with the better solutions.

Egg Laying Radius (ELR) is the maximum distance from the cuckoos home ground where real cuckoos lay their eggs [73]. ELR is calculated as:

$$ELR_i = \alpha \times \frac{Number\ of\ current\ cuckoo's\ eggs}{Total\ number\ of\ eggs} \times (Var_{max} - Var_{min}) \quad (18)$$

where ELR_i is the maximum distance of i^{th} cuckoo, α is the integer value that handles the maximum value of ELR. var_{max} is the maximum variable limit and var_{min} is minimum variable limits. The objective of ELR is to limit the searching space in every iteration.

4) GENETIC ALGORITHM

Genetic Algorithm (GA) is a technique for global optimization that is proposed by John Holland in 1970s. GA is based on the genetic mechanism used to solve the optimization problems [22], [30], [33]. These are the main key stages in Genetic Algorithm: (i) population initialization, (ii) fitness feature evaluation, and (ii) new population generation. A genetic algorithm with genes called chromosomes carries information which is produced randomly. Chromosomes produce the original population, which has a constant number of individuals. All chromosomes are evaluated using the fitness function to give the closest optimal solution [30]. GA applies many mathematical operations such as mutation, crossover and fittest. Many optimization problems have been solved successfully by using GA.

5) ANT COLONY OPTIMIZATION

Ant Colony Optimization (ACO) is a probabilistic technique [24], [32], proposed by Marco Dorigo in 1991 based on the ant 's ability to find the shortest path route between the food source and the nest. In this method, an ant leaves some pheromone (in various amounts) on the ground as it moves. The route is marked later by the smell of this material. The other members of the colony follow the path for searching of the food source and returning to the nest on the same way. At each node, each ant takes stochastic decisions by employing heuristic information and pheromone trails to select the next hop.

The probability of the Ant is shown in Eq. 19.

$$P_{ij}^{k}(t) = \frac{[c_{ij}(t))]^{\alpha}[I_{ij}(t))]^{\beta}}{\sum_{i=1}^{|I|}[c_{ij}(t))]^{\alpha}[I_{ij}(t))]^{\beta}} \quad (19)$$

Probability of transition from state x to state y depends on the combination of desirability of coupling (amount of pheromone) c_{ij}(t), and trace intensity I_{ij}(t).

The rule for updating global pheromone is as follows:

$$c_{ij} \leftarrow (1-\rho)c_{ij} + \sum_{k}^{m} \mu c_{xy}^{k} \quad (20)$$

where m represents the number of ants, μc_{xy}^{k} is the pheromone amount deposited by kth ant, and ρ is the coefficient representing pheromone evaporation.

6) PARTICLE SWARM OPTIMIZATION

Particle swarm optimization (PSO) [27], [34] is a meta-heuristic algorithm proposed by Kennedy and Eberhart, inspired by the concept of swarm intelligence. PSO optimization algorithm is used to solve complex mathematical problems. PSO algorithm work on the principle of interactions, to share information between the members. This method performs the search for the optimal solution through particles. In the search space, a particle is treated as a feasible solution. The particles flight behavior is considered as the search process using which they traverse a search space for optima.

The PSO algorithm considers a set of N_p particles and each particle moves randomly. A particle i is defined by its velocity and position vectors. During every iteration, the velocity and position of each particle are dynamically updated allowing the historical optimal position of swarm population and optimal position of the particle.

In PSO, each particle updates its velocity VE and positions PO with following equations

$$VE_i(t+1) = \Omega VE_i(t) + c_1 r_1(pbest(i,t) - po_i(t)) + c_2 r_2(gbest(t) - po_i(t)) \quad (21)$$

$$PO_i(t+1) = po_i(t) + VE_i(t+1) \quad (22)$$

where i denote the index of the swarm global best particle, VE is the velocity and Ω is the inertia weighting factor which is dynamically reduced; r_1 and r_2 are random variables generated from the uniform distribution on the interval [0, 1]; c_1 and c_2 parameters denote as acceleration coefficients.

In Eq. 21 ($i.e.\Omega VE_i(t)$) is known as inertia that represents the previous velocity, whereas $(pbest(i,t) - po_i(t))$ is known as cognitive component that encourage the particles to move towards their own best position, and the collaborative effect of the particle is represented by $c_2\ r_2(gbest(t) - po_i(t))$ [34]. $pbest(i,t)$ is the historically best position until iteration t and $gbest$ is the global best particle with best position in the swarm.

$$pbest(i,t) = arg_{k=1,2,\ldots,t}\ min[f(po_i(k))], \quad where\ i \in (1, 2, \ldots N_p)$$

$$gbest(t) = arg_{k=1,2,\ldots,t\ i=1,2,\ldots N_p}\ min[f(po_i(k))] \quad (23)$$

where N_p is the particles count, f is the fitness function, p is the position and t is the current iteration number.

7) BAT ALGORITHM

Xin-She Yang proposes the Bat algorithm (BA) [29], [45], influenced by the echolocation behavior of micro bats. This kind of behavior guides and helps bats in their conduct of flying and hunting. The bats not only move with this amazing orientation mechanism, but also, they distinguish the difference between an obstacle and insect forms even in absolute darkness.

In this algorithm, the position of each bat is defined by x_k^t with velocity v_k^t, frequency f, loudness A_k^t, and the emission pulse rate r_k^t in a search space. The velocity and position of the k^{th} bat are calculated at the time t by using the following equations:

$$f_k = f_{mn} + (f_{max} - f_{mn})\beta \tag{24}$$
$$v_k^t = v_k^{t-1} + (x_k^{t-1} - x_k^t) f_k \tag{25}$$
$$x_k^t = x_k^{t-1} + v_k^t \tag{26}$$

where f_k is the frequency of the sound waves emitted by k^{th} bats; f_{max} and f_{mn} are maximum and minimum frequency of the sound waves respectively; β is a random number generated from uniform distribution [0, 1]. Velocities of k^{th} bat are v_k^t and v_k^{t-1} at time t and time $(t-1)$, respectively, and x_k^t presents the bat's current global optimal location.

For local search, the position of each bat is reformed according to the Eq. 27. Using a local random walk the following equation is used to generate a solution:

$$X_{new} = X_{old} + \delta \acute{L}(t) \tag{27}$$

where δ is a random number generated from the uniform distribution on the interval $[-1, 1]$, X_{old} is randomly selected solution from the current optimal solution, and at the i^{th} iteration, $\acute{L}$ is the average loudness for all bats.

B. PERFORMANCE ANALYSIS AND EVALUATION

1) PERFORMANCE ANALYSIS

COCOMO is the most widely used software estimation model. It predicts the schedule and effort of the software product, which are considered as key parameters in defining the quality of any software product. The effort represents the amount of labor required to complete a task. Effort estimation unit is ''person month'' that represents an individual's months worth of efforts. The schedule represents the amount of time required for job completion, which is proportional to the effort put in. Unit of time such as a week is considered as estimation unit for schedule. Different types of COCOMO models are available for software cost calculation: Basic COCOMO, Intermediate COCOMO, and advanced COCOMO [5], of which the first two are frequently used. The basic COCOMO model considers effort and few other parameters [2].

In Basic COCOMO, the formulas for estimating the effort based on the software product size represented in Kilo Lines of Code (KLOC), for the three categories of software products are as follows:

$$Organic = 2.4 * (KLOC)^{1.05} \tag{28}$$
$$Semidetached = 3.0 * (KLOC)^{1.12} \tag{29}$$
$$Embeddded = 3.6 * (KLOC)^{1.20} \tag{30}$$

2) DATASETS SELECTION

Following publicly available datasets were selected for performance comparison of seven nature-inspired meta-heuristic algorithms. These datasets were downloaded from the software engineering repository [44].

- **NASA:** The data-set having 93 projects is described by 15 attributes. The software projects information is recorded from different NASA centers for many years. It has 15 efforts multipliers and 5 scaling factors [30].
- **COCOMO 81:** The data-set having 63 projects is described by 15 attributes. The software projects data available in this repository is based on the COCOMO software cost model, which measures effort used to develop a software project in calendar months. It also has effort multipliers having standard numeric values [31].
- **Maxwell:** The data-set was downloaded from a promise repository. The total number of projects is 62 having 27 attributes for each project [34].

3) EVALUATION CRITERIA

The research community realizes that MMRE, Pred (x) can be influenced by the presence of outliers [66], therefore, we used the following performance metrics to assess and compare the accuracy of the seven meta-heuristic algorithms [44].

a: MAGNITUDE OF RELATIVE ERROR (MRE)

The error ratio between actual and predicted effort for each project instance in the dataset is computed using MRE (expressed using Eq. 31. To calculate the fitness function, first the MRE is calculated for every project i.

$$MRE_i = \left| \frac{Actualeffort_i - Estimateffort_i}{Actualeffort_i} \right| \tag{31}$$

b: MEAN MAGNITUDE OF RELATIVE ERROR (MMRE)

MMRE measure is used for assessing software estimation technique performance. The values of MRE is calculated for each software project instance. MMRE computes the average over N number of project instances in the data-set. MMRE is defined in Eq. 32.

$$MMRE = \frac{1}{N} \sum_{i=1}^{N} MRE_i \tag{32}$$

c: PERCENTAGE OF PREDICTION (PRED)

PRED(x) represents the percentage of MRE that is less than or equal to the value x/100 among all projects. The PRED(x) is defined in Eq. 33.

$$PRED(25) = \frac{100}{N} \begin{cases} 1 & if\ MRE_i \leq \dfrac{25}{100} \\ 0 & otherwise \end{cases} \tag{33}$$

d: MEAN OF BALANCED RELATIVE ERROR (MBRE)

MBRE better handles outliers as it chastens underestimation and overestimation on the same level [12]. MBRE is formulated in Eq. 34.

$$MBRE = \frac{1}{N}\sum_{i=1}^{N}\frac{|Actualeffort_i - Estimateffort_i|}{min(Actualeffort_i - Estimateffort_i)} \quad (34)$$

e: MEAN INVERTED BALANCED RELATIVE ERROR (MIBRE)

MIBRE is a practical evaluation criterion and is formulated in Eq. 35.

$$MBRE = \frac{1}{N}\sum_{i=1}^{N}\frac{Actualeffort_i - Estimateffort_i}{max(Actualeffort_i - Estimateffort_i)} \quad (35)$$

f: MEAN ABSOLUTE ERROR (MAE)

MAE determines the average of absolute differences between the actual effort and each predicted effort in Eq. 36.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}\frac{|Actualeffort_i - Estimateffort_i|}{N} \quad (36)$$

g: STANDARDIZED ACCURACY (SA)

Sehra *et al.* [21] and Shepperd *et al.* [64] proposed standardized accuracy (SA) based on MAE, as an unbiased error measure. SA formulated in Eq. 37 depicts effectiveness of an estimation method in comparison to random guessing. Higher SA values represent the effectiveness of an estimation method.

$$SA_i = 1 - \frac{MAE_i}{\overline{MAE_{p_0}}} \quad (37)$$

MAE_i is the mean absolute Error while $\overline{MAE_{p_0}}$ is the mean of random guessing approach.

h: EFFECT SIZE

Effect size determines the model likelihood in predicting the correct values rather than occurrence by chance. S_{P_0} represents standard deviation of the random guessing approach.

$$EffectSize = \frac{MAE_{p_i} - \overline{MAE_{p_0}}}{S_{p_0}} \quad (38)$$

i: MEAN OF RESIDUAL (MR)

Existing work [40], [68], critiqued that MMRE measure favors the models that underestimate, whereas the MMER favors the models that overestimate. Therefore, we used the MR measure to find out which model overestimates and which one underestimates. MR is expressed as in Eq. 39.

$$MR = \frac{Actualeffort_i - Estimateffort_i}{Actualeffort_i} \quad (39)$$

Negative MR identifies that the model tends to overestimate, whereas positive MR indicates underestimating behavior.

TABLE 2. Control parameters setting for the meta-heuristic algorithms.

Algorithm	**Parameter**	Value
GWO	Number of pack	5
	Position Vector	[1-3]
	Dimension	17
SB	Length of Runner	400
	Length of root	10
	Dimension	[-3,+3]
	θ	1e-3
CS	Laying eggs	2-4
	Levis Distribution	1.5
	Rate of discovery	0.25
	α	0.01
	Dimension	2
ACO	Evaporation coefficient	0.2
	α	1.0
	β	2.0
PSO	inertia weighting factor	0.9
	Velocity	[0.1, 1]
	Position	[1-8]
	Learning factor (c_1, c_2)	2
	Max Velocity	100
GA	Chromosomes	17
	Probability of crossover	0.8
	Probability of mutation	0.2
BAT	Max frequency	12
	Min frequency	1
	Local walk random	[-1,1]
	Sound Loudness	0.9
	Pulse Rate	0.5

4) EXPERIMENTAL SETUP

The computational tests were performed on a dual-core PC with a 2.2 GHz Core *i*7 and 16GB of RAM. For all the seven algorithms, the two common parameters are set as follows: Number of iterations is set as 100, and the population size is set as 30. Table 2 depicts the setting of control parameters for the seven meta-heuristic algorithms.

5) EXPERIMENTAL RESULTS

The seven meta-heuristic algorithms performances are analyzed using NASA, COCOMO 81, and Maxwell datasets. Table 3 depicts the comparison results of nine benchmark functions using COCOMO 81, NASA, and Maxwell datasets. A comparison has been performed with the basic COCOMO model.

As shown in Table 3, GWO has the lowest MAE, whereas SB has the second smallest MAE. The second column gives the MMRE values, showing that GWO with 1.67

TABLE 3. Comparison results of benchmark performance measures using NASA, COCOMO and Maxwell Datasets.

Optimization Models	MRE	MMRE	MBRE	MIBRE	PRED	MAE	SA	Δ	MR
				NASA					
Basic COCOMO	1.93	4.95	6.39	70.39	19.6	767.2	17.59	0.5	1.90
SB	1.06	2.47	3.79	65.18	58.4	442.2	63.17	0.8	1.06
ACO	1.67	4.28	5.53	67.53	30.4	682.1	27.26	0.7	1.65
CO	1.36	3.72	4.51	65.51	55.8	466.3	53.08	0.7	1.32
GA	1.51	3.87	5.00	67.10	37.0	621	44.51	0.6	1.48
GWO	0.65	1.67	2.15	62.15	72.9	374.3	79.00	0.8	0.65
PSO	1.49	3.81	4.91	66.91	38.1	626	43.58	0.6	1.49
BAT	1.46	3.67	4.68	66.68	42.5	592	49.48	0.7	1.46
				COCOMO 81					
Basic COCOMO	2.76	7.07	9.12	60.33	14.9	1036	22.7	0.4	2.72
SB	1.70	3.61	4.95	56.42	52.5	552	65.7	0.8	1.7
ACO	2.19	5.61	7.24	57.53	28.2	855	29.75	0.7	2.14
CO	1.75	4.07	5.47	55.21	43.2	573	62.04	0.8	1.71
GA	1.87	4.78	6.16	57.13	35.2	741	48.97	0.6	1.84
GWO	0.86	2.21	2.85	52.25	64.1	409	78.86	0.8	0.86
PSO	1.98	5.06	6.53	56.71	27.8	769	39.58	0.6	1.96
BAT	1.90	4.71	5.75	56.62	32.1	594	56.3	0.7	1.90
				Maxwell					
Basic COCOMO	1.49	4.58	6.08	72.38	22.34	1054	36.1	0.3	1.49
SB	1.02	2.72	3.38	66.44	59.23	492	67.2	0.7	1.02
ACO	1.95	4.54	5.78	68.22	31.24	908	37.6	0.3	-1.95
CO	1.08	2.86	4.38	67.36	58.43	522	62.21	0.7	1.08
GA	1.26	3.22	4.36	68.02	42.12	678	53	0.5	1.25
GWO	0.39	1.45	2.02	63.33	68.57	412	82.33	0.8	0.39
PSO	1.34	3.72	4.82	68.88	37.2	598	58.34	0.5	1.35
BAT	1.31	3.64	4.65	68.61	40.11	562	56.44	0.4	1.29

is substantially better than 4.95 for the NASA dataset. SA of GWO depicts about 55% improvement. PRED is a common alternative to MMRE, representing the generalization ability of the technique. Lower MMRE, and higher PRED represents that the derived estimates are more accurate. As shown in Table 3, GWO has the lowest MAE among the three datasets. To inspect the data results accuracy, additional tests using MRE, MBRE, and MIBRE performance measures were also conducted. Table 3, depicts that GWO surpassed other algorithms, across the three datasets. Also, SA determines the results meaningfulness, and △ determines the likelihood of the results occurring by chance. Table 3 shows that the GWO produced better results compared to other algorithms across the three datasets. SA and delta tests depicts that the Basic COCOMO model does not predict well. To assess the propensity of an algorithm to overestimate or underestimate, we used MR. As shown in Table 3, all algorithms except ACO in Maxwell dataset, tended to overestimate. Delta value is high for CO, and GWO across the three datasets, which shows that these two algorithms perform well.

The GWO algorithm reports the best results for three datasets, whereas SB reports the second-best performance for three datasets. Table 3 depicts that among GWO, SB, GA, CO, ACO, PSO, BAT metaheuristic algorithms; GWO has the lowest MAE, MBRE, MIBRE, and MMRE. Moreover, GWO has the largest SA, PRED, and Effect size. Most of the meta-heuristic algorithms underestimate across the three datasets. Whereas only ACO shows different behavior for different datasets in terms of MR. The performance of the GWO, CO, and SB algorithms remains more or less constant across different datasets.

IV. DEEP NEURAL NETWORK MODEL EXPLOITING METAHEURISTIC ALGORITHMS

Currently, DL is faced with many limitations including optimum initial values for the parameters selection technique,

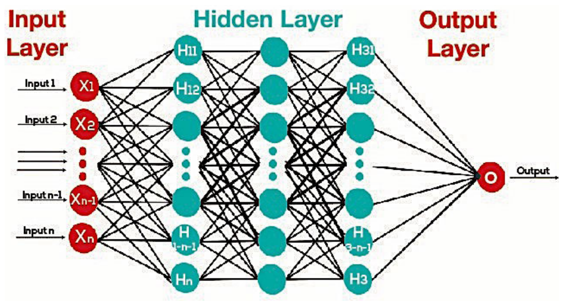


FIGURE 1. Network structure of the proposed GWDNNSB model.

architecture dynamic configuration method, and lack of standard training algorithm. DL architecture performance is heavily dependent on getting optimal initial weights for the architecture. There exists evidence, which emphasizes that for solving the real-world problems using DL architectures; optimum weights can be realized by using nature-inspired algorithms for training [42]. Applying meta-heuristics has its advantages in terms of catering to the weight training to its optimum state at a minimal computational cost [42]. To unravel the benefits of the application of the meta-heuristic algorithms in DL architectures used for software estimation, this section presents the GWDNNSB model and compares its performance with the existing DNN models.

Experimental results (Table 2) showed that the GWO has comprehensive superiority in terms of accuracy in estimation. The GWO algorithm reports the best results for three datasets, whereas SB reports close to second-best performance for three datasets. Table 2 depicts that among GWO, SB, GA, CO, ACO, PSO and BAT meta heuristic algorithms; GWO has the lowest MAE, MBRE, MIBRE, and MMRE. Moreover, GWO has the largest SA, PRED, and Effect Size. The main benefit of the SB algorithm is that it explores several regions of the search space at the same time and thus can break away from local optimum and achieve the global optimum. Exploration (global research) is done at every iteration, and exploitation (local research) is done if exploration does not result in significant improvement in the objective value. The number of computing agents varies from start to finish. SB algorithm discards half of the weak agents and duplicates the good agents at each iteration. Such mechanisms help the SB algorithm to effectively avoid a local trap.

A. MODEL BUILDING

The proposed model is a multi-hidden layer neural network. Fig. 1 depicts the network structure of the proposed GWDNNSB model. There are five layers in our DNN model: one input layer, one output layer, and three hidden layers. The three hidden layers depth will help in enhancing data fitting capability. The nodes in the GWDNNSB model are fully connected; in each layer every node is connected to every node in the following layer.

Number of features of the input data determine the number of input layer nodes. Table 4 and Table 5 depicts the input features considered in the Datasets. On the input layer there are 15 neurons/nodes that are basic attributes of the dataset. Also, each hidden layer node is composed of neurons. The first hidden layer has 50 neurons, the second has 100 neurons, and the third layer has 50 neurons. A node in a layer is connected to the next layer nodes. The network is made deeper by increasing the number of hidden layers [71]. The last layer, which is also known as the output layer, has only one node, which gives the total effort estimated.

Moreover, the neurons are equipped with rectifier activation and aggregation function. Rectified Linear Unit (ReLu) the activation function is used to account for interaction effects. ReLu function [67] is the most used activation function for DNN, which is considered appropriate for the reduction of overfitting influence and improving model training speed of the model [71]. The Rectifier activation function is defined as follows in Eq. 40.

$$0_i = \begin{cases} r_i & if\ r_i \geq 0 \\ 0 & if\ r_i < 0 \end{cases} \tag{40}$$

A given node computes the output after taking the inputs, bias is added to the weighted sum of inputs which is then passed through a non-linear activation function. Previous layer nodes at the output becomes the input to the node in the next layer. The equation for a hidden node f(b+x.d) output lying on layer 1 is given in Eq. 41. This procedure is performed by all the nodes to compute the final output. The input feature in the function is represented by x; d is the weight of the layer, and b is the bias, which means how far off our predictions are from real values.

$$f(\vec{b} + \vec{x}.\vec{d}) = \sum_{i}^{n}(d_i x_i + b) \tag{41}$$

B. MODEL INITIALIZATION AND OPTIMIZATION

Meta-heuristic algorithms are hybridized with DNN to reduce the training time required to train a network and to achieve high accuracy in the required results [42]. A combination of the meta-heuristic algorithms is the lifeblood of the proposed model known as GWDNNSB. In which GWO is used for initial weight optimization of the DNN model. A GWO trainer finds a set of values for weights and biases allowing the highest prediction accuracy. SB algorithm [68], [69] is used to set an optimized learning rate which is an important parameter of a DNN model. The fundamental flowchart of GWDNNSB is presented in Fig. 2.

The amount of change in weights is determined by the learning rate. The learning rate is adjusted by using the SB algorithm [68], [69], N random points are generated initially (representing mother plants) in the problem domain. As part of the global search, each mother plant produces a daughter plant. As represented in Eq. (14), the global search is effective, if upon comparison with the best daughter (computed in the previous iteration) at least one of the daughter plants improves the cost function. In case the global search is not successful, then local search needs to be carried out where

TABLE 4. Selected features of COCOMO 81 and NASA datasets for effort estimation.

Variables	Description	Role
Analyst's Capability	Ability to study and exam the system	Input
Application Experience	Knowledge and skill of application	Input
Process Complexity	Assessment of event and tasks that make process	Input
Database Size	How large and complicated database is	Input
Modern Programming Practice	Updated method used for development	Input
Programmer's Capability	Knowledge and skill of programmer	Input
Required Software Reliability	Probability of failure-free software	Input
Schedule Constraint	limitation placed on a project schedule	Input
Main Memory Constrain	Memory required to efficiently complete various operation	Input
Time Constrain for CPU	Processing time to complete an action	Input
Use of Software Tools	Used of various modern framework	Input
Turnaround Time	Amount of time taken to complete a process	Input
Virtual Machine Experience	Experience required to work on virtual systems	Input
Machine Volatility	Knowledge and experience to work on various machines	Input
Effort	Required effort for development	Output

TABLE 5. Selected features of China dataset for effort estimation.

Variables	Description	Role
Input	Function points of input	Input
Output	Function points of external output	Input
Enquiry	Function points of external enquiry	Input
File	Function points of internal logical files or entity references	Input
Interface	Function points of external interface added	Input
Added	function points of new or added functions	Input
Changed	Function points of changed functions	Input
Deleted	Function points of deleted functions	Input
PDR_AFP	Productivity Delivery Rate (PDR) of Adjusted Function Point(AFP)	Input
PDR_UFP	Productivity delivery rate of unadjusted FP	Input
NPDR_AFP	Normalized Productivity delivery rate AFP	Input
NPDU_UFP	Normalized PDR unadjusted FP	Input
Resource	Team type	Input
Dev.Type	Development type	Input
Effort	Required effort for development	Output

fittest daughter plant variables are adjusted randomly with both small and large steps. In reality, the local search is carried out in two steps: (i) with large random steps in the problem domain, and (ii) with small random steps in problem domain At every iteration, the mother plant generates two random points: (i) one in the vicinity, termed as root; and (ii) one relatively farther location from itself, termed as a runner. The runners move with large random steps in the problem domain [69], [70]. At the runners and roots locations the objective function is evaluated, and the points with high fitness values are selected and envisaged as mother plants for the next iteration, using Eq. 12 to 14.

The process continues until the optimized value is found. SB algorithm choses a learning rate between 0.0 and 1.0, considering the tradeoff between the network convergence to something useful and training time. The learning rate affects the time required for the learning process of DNNs. This learning rate is given to the model which is obtained as output result from the SB algorithm.

The amount of change in weights is determined by the learning rate. The learning rate is adjusted by using the SB algorithm [68], [69], N random points are generated initially (representing mother plants) in the problem domain. As part of global search, each mother plant produces daughter plant. At the runners and roots locations the objective function is evaluated, and the points with high fitness values are selected and envisaged as mother plants for the next iteration, using Eq. 12 to 14. The process continues until the optimized value is found. SB algorithm chose learning rate between 0.0 and 1.0, considering the tradeoff between the network convergence to something useful and training time. The learning rate affects the time required for the learning process of DNNs. This learning rate is given to the model which is obtained as output result from the SB algorithm.

C. TUNING PARAMETERS

Supervised training is used for training GWDNNSB model. The GWO decreases the probability of trapping into the local extrema by allowing the target solution to be systematically assessed by three solutions. The GWO algorithm is used to assign initial weights and bias. The model parameters (weights and biases) are adjusted according to the comparison results. Weights and biases are introduced in a vector form

$$\vec{A} = (\vec{d}, \quad \vec{b}) = (\vec{d}_{1,1}\ , \vec{d}_{1,2}\ \ldots \vec{d}_{n,n}, \quad \vec{b}_{1,1}\ , \vec{b}_{1,2}\ \ \ldots \vec{b}_{n,n}) \tag{42}$$

where n is the number of the input nodes, d_{ij} shows the connection weight from the i^{th} node to the j^{th} node, b_j represents bias of j^{th} hidden node

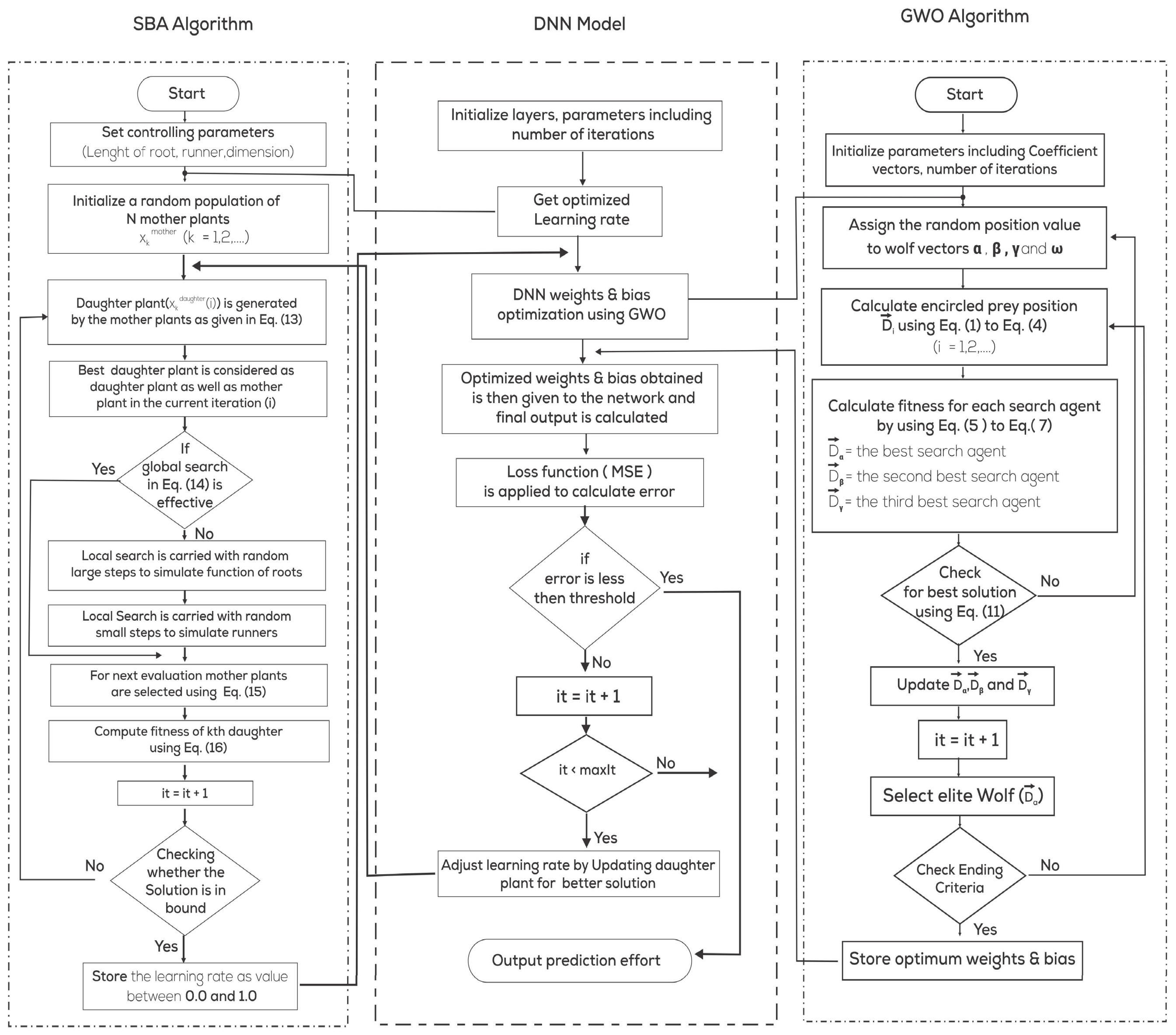

SBA Algorithm
DNN Model
GWO Algorithm
Start
Set controlling parameters (Lenght of root, runner,dimension)
Initialize a random population of N mother plants x_k^{mother} (k = 1,2,....)
Daughter plant($x_k^{daughter}$(i)) is generated by the mother plants as given in Eq. (13)
Best daughter plant is considered as daughter plant as well as mother plant in the current iteration (i)
If global search in Eq. (14) is effective
Yes
No
Local search is carried with random large steps to simulate function of roots
Local Search is carried with random small steps to simulate runners
For next evaluation mother plants are selected using Eq. (15)
Compute fitness of kth daughter using Eq. (16)
it = it + 1
Checking whether the Solution is in bound
Store the learning rate as value between 0.0 and 1.0
Initialize layers, parameters including number of iterations
Get optimized Learning rate
DNN weights & bias optimization using GWO
Optimized weights & bias obtained is then given to the network and final output is calculated
Loss function (MSE) is applied to calculate error
if error is less then threshold
it = it + 1
it < maxIt
Adjust learning rate by Updating daughter plant for better solution
Output prediction effort
Initialize parameters including Coefficient vectors, number of iterations
Assign the random position value to wolf vectors α , β , γ and ω
Calculate encircled prey position $\vec{D}_i$ using Eq. (1) to Eq. (4) (i = 1,2,....)
Calculate fitness for each search agent by using Eq. (5) to Eq.(7)
$\vec{D}_\alpha$ = the best search agent
$\vec{D}_\beta$ = the second best search agent
$\vec{D}_\gamma$ = the third best search agent
Check for best solution using Eq. (11)
Update $\vec{D}_\alpha$, $\vec{D}_\beta$ and $\vec{D}_\gamma$
Select elite Wolf ($\vec{D}_\alpha$)
Check Ending Criteria
Store optimum weights & bias

FIGURE 2. **Flowchart of GWDNNSB model.**

In the wake of instating the factors (weight and biases), we should characterize the fitness function. We used the Mean Square Error (MSE) as the loss function, to figure the output error. It measures the accuracy of each prediction during the training process of our model, as described in Eq. 43.

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(o_i - \hat{o}_i) \tag{43}$$

where m represents the number of training samples, o_i represents the obtained output of the i^{th} instance, and $(\hat{o}_i)$ denotes the ideal output. The training operation will train the DNN by looking at the results of the loss function and using that to adjust the weights in the neural network until they produce the desired output.

The primary objective of training DNN is to limit the MSE values for all training samples. The GWO algorithm adjusts the weights as populaces to improve the error rate and to accomplish the finest execution in training and testing.

D. COMPUTATIONAL COMPLEXITY ANALYSIS

The meta-heuristic algorithm computational complexity is primarily determined by population size (N), Solution Space Dimensions (SSD), and maximal iterations ($MaxIts$).

GWO and SB time complexities are summarized as follows: (i) Population initialization phase: $O(N \times SSD)$; (ii) Control parameters calculation: $O(N \times SSD)$; (iii) Agents position update: $O(N \times SSD \times MaxIts)$; and (iv) Fitness calculation time: $O(N \times SSD \times MaxIts)$.

TABLE 6. Structure of each datasets.

	Training & Testing	Number of input nodes	Number of hidden nodes	Number of output nodes
Dataset1	1,000 labeled samples from NASA dataset are used for training purposes and 400 label samples from COCOMO-81 dataset for testing purpose.	15	• 1st hidden layer = 50 neurons • 2nd hidden layer = 100 neurons • 3rd hidden layer = 50 neurons	1
Dataset2	The dataset consists of 499 labeled samples. In the experiments, 300 labeled samples were used for training purpose, and 199 samples for testing purpose.	15	• 1st hidden layer = 50 neurons • 2nd hidden layer = 100 neurons • 3rd hidden layer = 50 neurons	1

Given *MaxIts*, the total time complexity of the SB and GWO is $O(N \times SSD \times MaxIts)$.

Detail comparison is as follows:

$$\begin{aligned}
O(GWO) &= O(PopulationInitialization)\\
&\quad +O(Cal.ControlParameter)\\
&\quad + O(UpdatePosition)\\
&\quad + O(Fitnesscal.forthePopulation)\\
O(GWO) &= O(N \times SSD)\\
&\quad + O(N \times SSD)\\
&\quad + O(N \times SSD \times MaxIts)\\
&\quad + O(N \times SSD \times MaxIts)\\
O(GWO) &= O((2MaxIts + 1) \times N \times SSD)\\
&= O(MaxIts \times N \times SSD)\\
O(SB) &= O(PopulationInitialization)\\
&\quad + O(Generation\ of\ Daughter\ Plant\\
&\quad + Position\ Update)\\
&\quad + O(Calculation\ of\ Fitness\ of\ new\ Location)\\
&\quad + O(Mother\ Plants\ Selection\ for\\
&\qquad Next\ Generation)\\
O(SB) &= O(N \times SSD) + O(N \times SSD \times MaxIts)\\
&\quad + O(N \times SSD \times MaxIts)\\
&\quad + O(N \times SSD \times MaxIts)\\
O(SB) &= O((3MaxIts + 1) \times N \times SSD)\\
&= O(MaxIts \times N \times SSD)
\end{aligned}$$

GWODNNSB model time complexity is the same as that of GWO and smaller than that of SB.

TABLE 7. Parameters values of meta-heuristic algorithms in GWODNNSB model.

	Parameter	Values
GWO	Number of pack	5
	Position Vector	[1-3]
	Dimension	17
	Population Size	30
	Max. Iterations	100
SB	Population Size (No. of mother plants)	30
	θ	1e-3
	Selection control parameter (α)	0.1
	Length of runner ($d_{runners}$)	400
	Length of root (d_{root})	10
	Dimension	[-3,+3]
	Max. Iterations	100

E. PERFORMANCE ANALYSIS AND EVALUATION

The proposed model GWDNNSB is compared with DeepMNN [49].

1) DATA ACQUISITION AND PROCESSING

To illustrate the generality of the proposed approach, the results are computed using three software estimation datasets: (i) NASA [30] (ii) COCOMO-81 [31], and (iii) China dataset [55]. The aim is to find out whether the accuracy is dataset dependent. From NASA and COCOMO-81 datasets, fifteen common and most important attributes listed in Table 4 are considered as input. The effort attribute is considered as the dependent entity. China [44], [55] dataset consists of 499 projects and the input features considered are

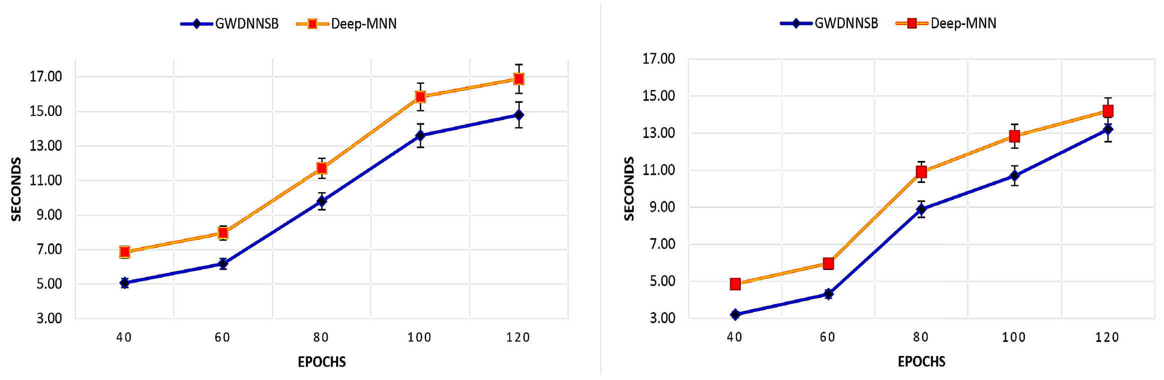


FIGURE 3. (a). Execution time comparison using Dataset1 (b). Execution time comparison using Dataset2.

presented in Table 5. This value at the output layer is the total effort required to build the software.

As depicted in Table 6, two datasets are used. In Dataset1, NASA dataset for training purpose and COCOMO 81 dataset for testing performance. Common attributes from both datasets have been considered. Table 4 presents the selected features of COCOMO 81 and NASA datasets for effort estimation. NASA dataset and COCOMO 81 dataset contains lesser number of instances. Therefore, the augmentation technique SMOTE [59], [72] has been applied to increase the number of instances in the dataset. The second experiment in each kind is conducted using the China dataset. Since China dataset contains sufficient instances, we did not apply any augmentation technique. Before providing the data as input to the DNN, preprocessing of the data is performed.

2) EXPERIMENTAL SETUP

The model was built using TensorFlow [60], which is a software framework for building and deploying machine learning models. Python programming language was used for coding. Different libraries were also used to build the model.

3) EXPERIMENTAL RESULT

In order to verify the performance of our model and prove that our improvement is meaningful, we have designed four sets of experiments. Each experiment is repeated twice for different datasets. The initial parameters of meta-heuristics algorithms are fixed in Table 7.

EXPERIMENT 1:

In this experiment, the proposed model is compared with Deep-MNN [49], in terms of execution time required for training the model. This experiment was repeated to verify the performance on different datasets. Fig. 3(a) shows the results computed by using Dataset1. Fig. 3 (b) shows the results computed by using Dataset2. We run training and testing process using various numbers of epochs from 40 to 120 with interval of 20 epochs each. The results show that our proposed GWDNNSB model takes less execution time as shown in Fig. 3(a) and Fig. 3(b).

EXPERIMENT 2:

In this experiment, the optimized proposed GWDNNSB model is compared with Deep-MNN in terms of effort estimation accuracy. The process was executed up to 100 epochs, and the effort estimation accuracy achieved by the GWDNNSB is almost 94% when Dataset1 is used as shown in Fig. 4 (a), while it goes to 95% when Dataset2 is used as shown in Fig. 4 (b). While Deep-MNN [49] achieved accuracy up to 85% when Dataset2 is used as shown in Fig. 4 (b). The optimized proposed DNN performed better in terms of accuracy as compared to Deep-MNN when used.

EXPERIMENT 3:

To check the effectiveness of the proposed model various other measures including Precision, Recall, and the F-measure are used. Considering the actual effort in each instance of the Dataset, the Dataset instances are divided into three classes (Low, Medium, and High). In each class effort, range values are selected to provide natural splits and also to ensure a balanced distribution of the Dataset. For this purpose, we computed the measures including Quartiles and the Median. We considered the effort estimation problem as a set of binary classification problems, one for each class. For example, when considering the class "Low", a True Positive (TP) occurs when an actual "Low" class effort is correctly predicted.

Per-class Precision and Recall are computed as follows.

$$Recall = \frac{TP}{(TP + FN)} \tag{44}$$

$$Precision = \frac{TP}{(TP + FN)} \tag{45}$$

F-measure is the harmonic mean of recall and precision. Per-class F-measure is calculated as follows:

$$\text{F-measure} = \frac{2 * Recall * Precision}{(Recall + Precision)} \tag{46}$$

Afterwards, we combined the per-class F-measure scores into a single number, using arithmetic mean of the per-class F-measure scores called the averaged F-measure score. In a similar way, we have also computed the averaged Precision and the averaged Recall scores as presented in Fig. 5

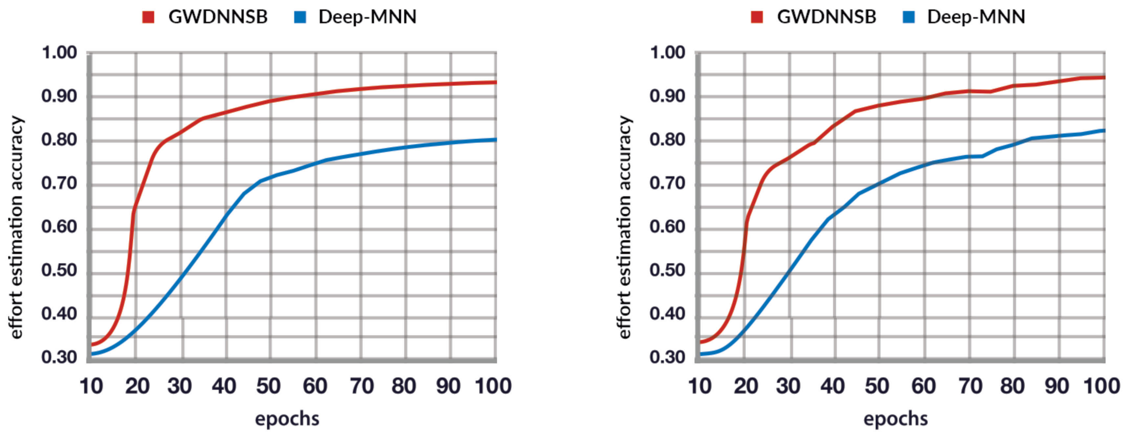


FIGURE 4. (a). Effort estimation accuracy comparison using Dataset1 (b). Effort estimation accuracy comparison using Dataset2.

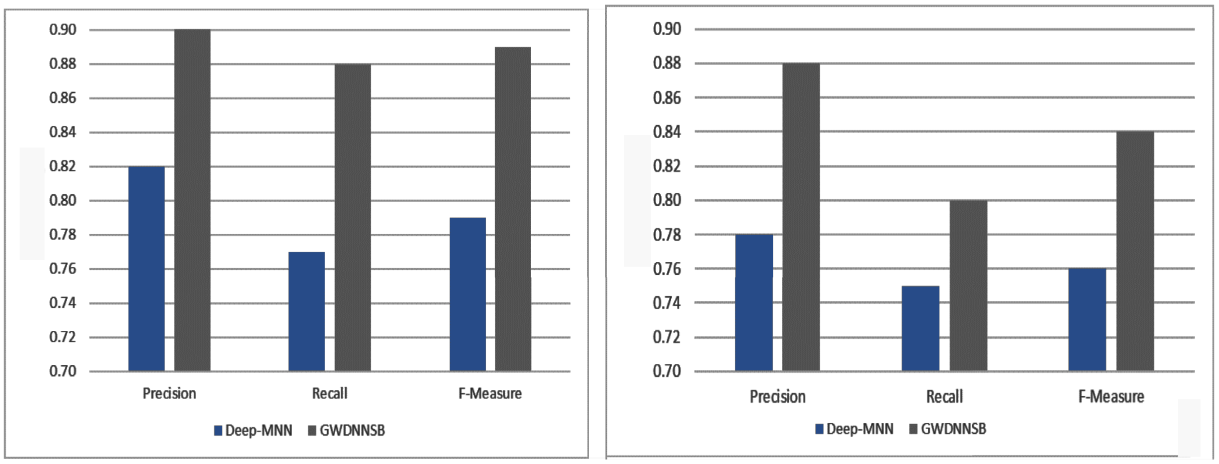


FIGURE 5. (a). Precision, recall and F-measure using Dataset1 (b). precision, recall and F-measure using Dataset2.

TABLE 8. Results of nine benchmark performance measures using Dataset1.

	Dataset1								
Optimization Models	MRE	MMRE	MBRE	MIBRE	PRED	MAE	SA	Δ	MR
Deep-MNN	3.93	6.95	8.39	80.39	10.04	967	13.2	0.5	3.27
GWDNNSB	0.95	2.47	3.45	58.15	42.1	457	63.5	0.9	0.95

(a) and 5 (b). Fig. 5 (a) and 5 (b) depicts, our GWDNNSB have high averaged Precision and averaged Recall as compared to Deep-MNN model. GWDNNSB also results in better F-measure value, using both Dataset1 and Dataset2. From these results, it is clear that the proposed model achieved optimized results.

EXPERIMENT 4:

The Deep-MNN [49] and our proposed GWDNNSB performances are analyzed using Dataset1 and Dataset2. Table 8 depicts the comparison results of nine benchmark functions using Dataset1. Table 9 depicts the comparison results of nine benchmark functions using Dataset2.

Table 8 and Table 9 depicts that the GWDNNSB model is superior to the Deep-MNN model when compared using multiple performance measures: MAE, MRE, MR, MMRE, MBRE, MIBRE, SA, and Δ. For performance comparison, we also used measures that are considered as reliable and unbiased (MAE, MBRE, and MIBRE) by the research community. The research community criticized that the measures that are derived from MRE are biased. Table 8 and Table 9 depicts that the GWDNNSB model has the lowest MAE exploiting the two datasets. Table 8 and Table 9 depicts that MRE, MMRE, MBRE are low for our optimized GWDNNSB model as compared to Deep-MNN. MIBRE is the inverse of MBRE, so its low values in Table 8 and Table 9 for GWDNNSB depict that the model resulted in better accuracy. MAE is high in the case of the Deep-MNN, which shows a higher error rate as compared to GWDNNSB model. SA test

TABLE 9. Results of nine benchmark performance measures using Dataset2.

	Dataset2								
Optimization Models	MRE	MMRE	MBRE	MIBRE	PRED	MAE	SA	Δ	MR
Deep-MNN	5.93	7.95	7.29	74.39	11.04	887	17.6	0.4	-5.91
GWDNNSB	1.25	3.17	4.15	55.15	51.1	397	56.2	0.8	1.24

depicts the reliability and meaningfulness of the prediction model. GWDNNSB model higher SA values for Dataset1 and Dataset2 shows that estimations are reliable. SA alone cannot guarantee the superiority of accuracy. Therefore, we conducted Effect size (Δ) test to figure out the meaningfulness of both prediction models. For the GWDNNSB model, the higher value Δ shows that the estimations are not produced by chance.

V. CONCLUSION AND FUTURE WORK

Constructive COst Model (COCOMO) is the most widely used regression-based algorithmic model for software effort estimation. For estimation accuracy, it is required to fine-tune the model parameters. GWO and SB meta-heuristic algorithms support solving multi-variable problems and have been applied to a variety of practical applications. In this paper, these algorithms are applied to having a logical and acceptable parametric model for software estimation. GWO is the one with the minimum MAE, MBRE, and MIBRE across all datasets. Effort size, SA, MAE, and PRED accuracy measures are considered more trustable by the research community as compared to MBRE, MIBRE. As observed, the GWO surpasses six meta-heuristic algorithms in terms of accuracy using nine accuracy measures across all the three datasets, proving a strong argument to back the reliability of GWO for software effort estimation.

The deep learning architectures although provides several improvements over existing shallow techniques. But they are also suffered from several shortcomings that may reduce the performance of the model including: (i) the fine-tuning of a large number of parameters, and (ii) longer training time. DNNs allows representing complex relationships between software effort and cost drivers, thus making it a better choice for software effort estimation. The performance of the DNN architecture heavily depends on the optimized selection of initial weights and relevant features for model building. Meta-heuristic algorithms allow for finding an optimum solution; therefore, can be used with DNNs to reduce large training delays. To improve the accuracies, such algorithms also, allow learning from the historical dataset structure and involve categorical attributes. In this paper, we investigated the influence of nature-inspired algorithms in optimizing DNN initial weights and learning rate. We harness GWO meta-heuristic algorithm to optimize initial parameter values for DNN and used SB to optimize the learning rate. GWO and SB algorithms have been applied by the research community for solving complex problems in other fields of science and engineering, but not exploited for software effort estimation. GWDNNSB model tuned by metaheuristic algorithms outperforms existing work on using DNN for software effort estimation. The evaluation of comparison using multiple performance measures shows that the GWDNNSB model outperforms the benchmark COCOMO model and the popular metaheuristic algorithms applied for having a logical and acceptable parametric model for software effort estimation.

In this work, we demonstrated that applying nature-inspired meta-heuristic algorithms to DNN in the context of software effort estimation speed up training and improves performance. The nature-inspired algorithms boost our proposed GWDNNSB performance. Given the limitations of GWO and SB algorithms in certain scenarios it can increase convergence time. Therefore, we plan to make a hybrid with improved versions of GWO, SB, and other metaheuristic algorithms that support the balance between exploitation and exploration, have high exploration capability, and do not lose a population diversity quickly. There exists work in other areas where meta-heuristic algorithms have been successfully implemented in DNNs to improve training. Relevant research publications in this direction are still rare. Research efforts should be focused on investigating meta-heuristic algorithms application in DL architectures for software effort estimation.

REFERENCES

[1] M. Usman, R. Britto, L.-O. Damm, and J. Börstler, "Effort estimation in large-scale software development: An industrial case study," *Inf. Softw. Technol.*, vol. 99, pp. 21–40, Jul. 2018.

[2] S. Mensah, J. Keung, M. F. Bosu, and K. E. Bennin, "Duplex output software effort estimation model with self-guided interpretation," *Inf. Softw. Technol.*, vol. 94, pp. 1–13, Feb. 2018.

[3] N. Cerpa, M. Bardeen, C. A. Astudillo, and J. Verner, "Evaluating different families of prediction methods for estimating software project outcomes," *J. Syst. Softw.*, vol. 112, pp. 48–64, Feb. 2016.

[4] V. Garousi, A. Coşkunçay, A. Betin-Can, and O. Demirörs, "A survey of software engineering practices in Turkey," *J. Syst. Softw.*, vol. 108, pp. 148–177, Oct. 2015.

[5] I. Qasim, H. Tufail, A. Fatima, T. Rasool, and F. Azam, "Cost estimation techniques for software development: A systematic literature review," in *Proc. Int. Conf. Eng., Comput. Inf. Technol. (ICECIT)*, 2017, pp. 38–42. [Online]. Available: https://www.researchgate.net/publication/323582454_Cost_Estimation_Techniques_for_Software_ Development_A_Systematic_Literature_Review

[6] G. Gabrani and N. Saini, "Effort estimation models using evolutionary learning algorithms for software development," in *Proc. Symp. Colossal Data Anal. Netw. (CDAN)*, Mar. 2016, pp. 1–6.

[7] N. Padhy, R. P. Singh, and S. C. Satapathy, "Software reusability metrics estimation: Algorithms, models and optimization techniques," *Comput. Electr. Eng.*, vol. 69, pp. 653–668, Jul. 2018.

[8] Y. Keim, M. Bhardwaj, S. Saroop, and A. Tandon, "Software cost estimation models and techniques: A survey," *Int. J. Eng. Res. Technol.*, vol. 3, no. 2, pp. 1763–1768, 2014.

[9] V. S. Dave and K. Dutta, "Neural network based models for software effort estimation: A review," *Artif. Intell. Rev.*, vol. 42, no. 2, pp. 295–307, Aug. 2014.

[10] S. Sarwar and M. Gupta, "Proposing effort estimation of COCOMO II through perceptron learning rule," *Int. J. Comput. Appl.*, vol. 70, no. 1, pp. 29–32, May 2013.

[11] S. Goyal and A. Parashar, "Machine learning application to improve COCOMO model using neural networks," *Int. J. Inf. Technol. Comput. Sci.*, vol. 10, no. 3, pp. 35–51, Mar. 2018.

[12] P. Pospieszny, B. Czarnacka-Chrobot, and A. Kobylinski, "An effective approach for software project effort and duration estimation with machine learning algorithms," *J. Syst. Softw.*, vol. 137, pp. 184–196, Mar. 2018.

[13] K. Zima, "The case-based reasoning model of cost estimation at the preliminary stage of a construction project," *Procedia Eng.*, vol. 122, pp. 57–64, 2015.

[14] X. Huang, D. Ho, J. Ren, and L. F. Capretz, "Improving the COCOMO model using a neuro-fuzzy approach," *Appl. Soft Comput.*, vol. 7, no. 1, pp. 29–40, Jan. 2007.

[15] I. Kaur, G. S. Narula, R. Wason, V. Jain, and A. Baliyan, "Neuro fuzzy—COCOMO II model for software cost estimation," *Int. J. Inf. Technol.*, vol. 10, no. 2, pp. 181–187, Jun. 2018.

[16] R. Malik, K. Solanki, A. Dhankhar, and S. Dalal, "Software reliability estimation using COCOMO II and neuro fuzzy method," *Int. J. Emerg. Technol. Innov. Res.*, vol. 5, no. 9, pp. 385–392, Sep. 2018. [Online]. Available: http://www.jetir.org/papers/JETIR1809705.pdf

[17] S. Shekhar and U. Kumar, "Review of various software cost estimation techniques," *Int. J. Comput. Appl.*, vol. 141, no. 11, pp. 31–34, May 2016.

[18] F. Tahir and M. Adil, "An empirical analysis of cost estimation models on undergraduate projects using COCOMO II," in *Proc. Int. Conf. Smart Comput. Electron. Enterprise (ICSCEE)*, Jul. 2018, pp. 1–5.

[19] R. Marco, N. Suryana, and S. S. Ahmad, "A systematic literature review on methods for software effort estimation," *J. Theor. Appl. Inf. Technol.*, vol. 97, no. 2, pp. 434–464, 2019.

[20] S. Chalotra, S. K. Sehra, Y. S. Brar, and N. Kaur, "Tuning of COCOMO model parameters by using bee colony optimization," *Indian J. Sci. Technol.*, vol. 8, no. 14, p. 1, Jul. 2015.

[21] S. K. Sehra, Y. S. Brar, N. Kaur, and S. S. Sehra, "Research patterns and trends in software effort estimation," *Inf. Softw. Technol.*, vol. 91, pp. 1–21, Nov. 2017.

[22] R. K. Sachan, A. Nigam, A. Singh, S. Singh, M. Choudhary, A. Tiwari, and D. S. Kushwaha, "Optimizing basic COCOMO model using simplified genetic algorithm," *Procedia Comput. Sci.*, vol. 89, pp. 492–498, 2016.

[23] S. W. Ahmad and G. R. Bamnote, "Whale–crow optimization (WCO)-based optimal regression model for software cost estimation," *Sādhanā*, vol. 44, no. 4, pp. 1–15, Apr. 2019.

[24] V. Venkataiah, R. Mohanty, J. S. Pahariya, and M. Nagaratna, "Application of ant colony optimization techniques to predict software cost estimation," in *Computer Communication, Networking and Internet Security*. Singapore: Springer, 2017, pp. 315–325.

[25] N. Ghatasheh, H. Faris, I. Aljarah, and R. M. Al-Sayyed, "Optimizing software effort estimation models using firefly algorithm," *J. Softw. Eng. Appl.*, vol. 8, no. 3, p. 133, 2015.

[26] S. M. S. Jafari and F. Ziaaddini, "Optimization of software cost estimation using harmony search algorithm," in *Proc. 1st Conf. Swarm Intell. Evol. Comput. (CSIEC)*, Mar. 2016, pp. 131–135.

[27] K. Langsari and R. Sarno, "Optimizing effort parameter of COCOMO II using particle swarm optimization method," *Telkomnika*, vol. 16, no. 5, pp. 2208–2216, 2018.

[28] M. Padmaja and D. Haritha, "Software effort estimation using meta heuristic algorithm," *Int. J. Adv. Res. Comput. Sci.*, vol. 8, no. 5, pp. 1–7, 2017.

[29] D. Nandal and O. P. Sangwan, "Software cost estimation by optimizing COCOMO model using hybrid BATGSA algorithm," *Int. J. Intell. Eng. Syst.*, vol. 11, no. 4, pp. 250–263, Aug. 2018.

[30] F. S. Gharehchopogh, R. Rezaii, and B. Arasteh, "A new approach by using tabu search and genetic algorithms in software cost estimation," in *Proc. 9th Int. Conf. Appl. Inf. Commun. Technol. (AICT)*, Oct. 2015, pp. 113–117.

[31] R. Saljoughinejad and V. Khatibi, "A new optimized hybrid model based On COCOMO to increase the accuracy of software cost estimation," *J. Advance Comput. Eng. Technol.*, vol. 4, no. 1, pp. 41–50, 2018.

[32] Z. A. Dizaji and F. S. Gharehchopogh, "A hybrid of ant colony optimization and chaos optimization algorithms approach for software cost estimation," *Indian J. Sci. Technol.*, vol. 8, no. 2, p. 128, Jan. 2015.

[33] F. S. Gharehchopogh, I. Maleki, and A. Talebi, "Using hybrid model of artificial bee colony and genetic algorithms in software cost estimation," in *Proc. 9th Int. Conf. Appl. Inf. Commun. Technol. (AICT)*, Oct. 2015, pp. 102–106.

[34] M. Ahadi and A. Jafarian, "A new hybrid for software cost estimation using particle swarm optimization and differential evolution algorithms," *Inform. Eng., Int. J.*, vol. 4, no. 1, pp. 63–73, 2016.

[35] A. Puspaningrum and R. Sarno, "A hybrid cuckoo optimization and harmony search algorithm for software cost estimation," *Procedia Comput. Sci.*, vol. 124, pp. 461–469, 2017.

[36] C. H. A. U. Hassan, M. S. Khan, A. Ghafar, S. Aimal, S. Asif, and N. Javaid, "Energy optimization in smart grid using grey wolf optimization algorithm and bacterial foraging algorithm," in *Proc. Int. Conf. Intell. Netw. Collaborative Syst.* Cham, Switzerland: Springer, 2017, pp. 166–177.

[37] M. S. Khan, C. A. ul Hassan, M. A. Shah, and A. Shamim, "Software cost and effort estimation using a new optimization algorithm inspired by strawberry plant," in *Proc. 24th Int. Conf. Autom. Comput. (ICAC)*, Sep. 2018, pp. 1–6.

[38] M. S. Khan, C. A. U. Hassan, H. A. Sadiq, I. Ali, A. Rauf, and N. Javaid, "A new meta-heuristic optimization algorithm inspired from strawberry plant for demand side management in smart grid," in *Proc. Int. Conf. Intell. Netw. Collaborative Syst.* Cham, Switzerland: Springer, 2017, pp. 143–154.

[39] A. Ali and C. Gravino, "A systematic literature review of software effort prediction using machine learning methods," *J. Softw., Evol. Process*, vol. 31, no. 10, p. e2211, Oct. 2019.

[40] A. B. Nassif, M. Azzeh, L. F. Capretz, and D. Ho, "Neural network models for software development effort estimation: A comparative study," *Neural Comput. Appl.*, vol. 27, no. 8, pp. 2369–2381, Nov. 2016.

[41] Z. Abdelali, H. Mustapha, and N. Abdelwahed, "Investigating the use of random forest in software effort estimation," *Procedia Comput. Sci.*, vol. 148, pp. 343–352, 2019.

[42] S. Fong, S. Deb, and X.-S. Yang, "How meta-heuristic algorithms contribute to deep learning in the hype of big data analytics," in *Progress in Intelligent Computing Techniques: Theory, Practice, and Applications*. Singapore: Springer, 2018, pp. 3–25.

[43] Z. Tian and S. Fong, "Survey of meta-heuristic algorithms for deep learning training," in *Optimization Algorithms—Methods and Applications*, O. Baskan, Ed. InTech, 2016.

[44] G. Boetticher. (2007). *The PROMISE Repository of Empirical Software Engineering Data*. [Online]. Available: http://promisedata.org/repository

[45] A. Kaushik and N. Singal, "A hybrid model of wavelet neural network and metaheuristic algorithm for software development effort estimation," *Int. J. Inf. Technol.*, pp. 1–10, Aug. 2019, doi: 10.1007/s41870-019-00339-1.

[46] Z. H. Wani and S. M. K. Quadri, "An improved particle swarm optimisation-based functional link artificial neural network model for software cost estimation," *Int. J. Swarm Intell.*, vol. 4, no. 1, pp. 38–54, 2019.

[47] E. Emary, H. M. Zawbaa, and C. Grosan, "Experienced gray wolf optimization through reinforcement learning and neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 3, pp. 681–694, Mar. 2018.

[48] A. Kaushik, D. K. Tayal, K. Yadav, and A. Kaur, "Integrating firefly algorithm in artificial neural network models for accurate software cost predictions," *J. Softw., Evol. Process*, vol. 28, no. 8, pp. 665–688, Aug. 2016.

[49] M. K. Kodmelwar, S. D. Joshi, and V. Khanna, "A deep learning modified neural network used for efficient effort estimation," *J. Comput. Theor. Nanoscience*, vol. 15, no. 11, pp. 3492–3500, Nov. 2018.

[50] S. Gu, R. Cheng, and Y. Jin, "Feature selection for high-dimensional classification using a competitive swarm optimizer," *Soft Comput.*, vol. 22, no. 3, pp. 811–822, Feb. 2018.

[51] V. S. Desai and R. Mohanty, "ANN-Cuckoo optimization technique to predict software cost estimation," in *Proc. Conf. Inf. Commun. Technol. (CICT)*, Oct. 2018, pp. 1–6.

[52] E. E. Miandoab and F. S. Gharehchopogh, "A novel hybrid algorithm for software cost estimation based on cuckoo optimization and K-nearest neighbors algorithms," *Eng., Technol. Appl. Sci. Res.*, vol. 6, no. 3, pp. 1018–1022, Jun. 2016.

[53] S. Chhabra and H. Singh, "Optimizing design of fuzzy model for software cost estimation using particle swarm optimization algorithm," *Int. J. Comput. Intell. Appl.*, vol. 19, no. 1, Mar. 2020, Art. no. 2050005.

[54] D. Zelikman and M. Segal, "Reducing interferences in VANETs," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 3, pp. 1582–1587, Jun. 2015.

[55] A. Kaushik, D. K. Tayal, and K. Yadav, "The role of neural networks and metaheuristics in agile software development effort estimation," *Int. J. Inf. Technol. Project Manage.*, vol. 11, no. 2, pp. 50–71, 2020.

[56] S. Pallavi and S. R. Sarangi, "Internet of Things: Architectures, protocols, and applications," *J. Elect. Comput. Eng.*, vol. 2017, Jan. 2017, Art. no. 9324035.

[57] A. Hussein, L. Chadad, N. Adalian, A. Chehab, I. H. Elhajj, and A. Kayssi, "Software-defined networking (SDN): The security review," *J. Cyber Secur. Technol.*, vol. 4, no. 1, pp. 1–66, Jan. 2020.

[58] Y. B. Zikria, S. W. Kim, M. K. Afzal, H. Wang, and M. H. Rehmani, "5G Mobile services and scenarios: Challenges and solutions," *Sustainability*, vol. 10, p. 3626, 2018. [Online]. Available: https://www.mdpi.com/2071-1050/10/10/3626

[59] A. V. Vasilakos, Z. Li, G. Simon, and W. You, "Information centric network: Research challenges and opportunities," *J. Netw. Comput. Appl.*, vol. 52, pp. 1–10, Jun. 2015.

[60] M. Caria, G. Todde, G. Sara, M. Piras, and A. Pazzona, "Performance and usability of smartglasses for augmented reality in precision livestock farming operations," *Appl. Sci.*, vol. 10, no. 7, p. 2318, Mar. 2020.

[61] M. Shepperd and S. MacDonell, "Evaluating prediction systems in software project estimation," *Inf. Softw. Technol.*, vol. 54, no. 8, pp. 820–827, Aug. 2012.

[62] T. Menzies, E. Kocaguneli, B. Turhan, L. Minku, and F. Peters, *Sharing Data and Models in Software Engineering*. San Mateo, CA, USA: Morgan Kaufmann, 2014.

[63] M. Azzeh, A. B. Nassif, S. Banitaan, and F. Almasalha, "Pareto efficient multi-objective optimization for local tuning of analogy-based estimation," *Neural Comput. Appl.*, vol. 27, no. 8, pp. 2241–2265, Nov. 2016.

[64] A. Fred Agarap, "Deep learning using rectified linear units (ReLU)," 2018, *arXiv:1803.08375*. [Online]. Available: http://arxiv.org/abs/1803.08375

[65] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980

[66] F. Merrikh-Bayat, "The runner-root algorithm: A metaheuristic for solving unimodal and multimodal optimization problems inspired by runners and roots of plants in nature," *Appl. Soft Comput.*, vol. 33, pp. 292–303, Aug. 2015.

[67] S. Akyol and B. Alatas, "Plant intelligence based metaheuristic optimization algorithms," *Artif. Intell. Rev.*, vol. 47, no. 4, pp. 417–462, Apr. 2017.

[68] *How to Fix Vanishing Gradients Using the Rectified Linear Activation Function*. Accessed: Jan. 2021. [Online]. Available: https://machinelearningmastery.com/how-to-fix-vanishing-gradients-using-the-rectified-linear-activation-function/

[69] A. Fernandez, S. Garcia, F. Herrera, and N. V. Chawla, "SMOTE for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary," *J. Artif. Intell. Res.*, vol. 61, pp. 863–905, Apr. 2018.

[70] S. Gao, M. Zhou, Y. Wang, J. Cheng, H. Yachi, and J. Wang, "Dendritic neuron model with effective learning algorithms for classification, approximation, and prediction," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 2, pp. 601–614, Feb. 2019, doi: 10.1109/TNNLS.2018.2846646.

[71] X. Luo, M. Shang, and S. Li, "Efficient extraction of non-negative latent factors from high-dimensional and sparse matrices in industrial applications," in *Proc. IEEE 16th Int. Conf. Data Mining (ICDM)*, Barcelona, Spain, Dec. 2016, pp. 311–319, doi: 10.1109/ICDM.2016.0042.

[72] X. Luo, H. Wu, H. Yuan, and M. Zhou, "Temporal pattern-aware QoS prediction via biased non-negative latent factorization of tensors," *IEEE Trans. Cybern.*, vol. 50, no. 5, pp. 1798–1809, May 2020, doi: 10.1109/TCYB.2019.2903736.

[73] P. M. Kebria, A. Khosravi, S. M. Salaken, and S. Nahavandi, "Deep imitation learning for autonomous vehicles based on convolutional neural networks," *IEEE/CAA J. Automatica Sinica*, vol. 7, no. 1, pp. 82–95, Jan. 2020, doi: 10.1109/JAS.2019.1911825.

[74] G. Bao, Y. Zhang, and Z. Zeng, "Memory analysis for memristors and memristive recurrent neural networks," *IEEE/CAA J. Automatica Sinica*, vol. 7, no. 1, pp. 96–105, Jan. 2020, doi: 10.1109/JAS.2019.1911828.

[75] P. Roy, G. S. Mahapatra, and K. N. Dey, "Forecasting of software reliability using neighborhood fuzzy particle swarm optimization based novel neural network," *IEEE/CAA J. Automatica Sinica*, vol. 6, no. 6, pp. 1365–1383, Nov. 2019, doi: 10.1109/JAS.2019.1911753.

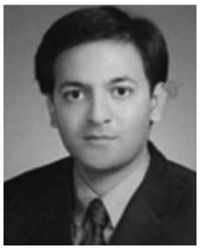
MUHAMMAD SUFYAN KHAN graduated in software engineering from the Department of Software Engineering, Bannu University, Pakistan, in 2015. He is currently an active Young Researcher with the Department of Computer Science, COMSATS University, Islamabad, Pakistan. His main research interests include artificial intelligence, intelligent systems, hybrid metaheuristics, and deep learning.

FARHANA JABEEN received the Ph.D. degree from Manchester University. She is currently working as an Assistant Professor with the School of Computer Science, COMSATS University, Pakistan. Her research interests include different aspects of computer networking, such as VANETs, NDN, SDN, and the IoT, cyber security, and wireless protocol design and analysis. She is also serving as an associate editor, the track chair, and a program committee member for numerous international and national conferences.

SANAA GHOUZALI received the master's and Ph.D. degrees in computer science and telecommunications from University Mohamed V-Agdal, Rabat, Morocco, in 2004 and 2009, respectively. From 2009 to 2011, she was an Assistant Professor with the National School of Applied Sciences (ENSA), Abdelmalek Essaadi University. In 2012, she joined the College of Computer and Information Sciences, King Saud University, as an Assistant Professor. Her research interests include statistical pattern detection and recognition, biometrics, and biometric security and protection. In 2005, she received the Fulbright Grant to undertake dissertation research on a joint-supervision program at the Visual Communication Laboratory, Cornell University, Ithaca, NY, USA.

ZOBIA REHMAN received the Ph.D. degree from Lucian Blaga University, Sibiu, Romania, in 2015. She is currently working as an Assistant Professor with the School of Computer Science, COMSATS University, Pakistan. Her research interests include knowledge management, hybrid metaheuristics, natural language processing, and deep learning.

SHENEELA NAZ received the Ph.D. degree in computer science from the Capital University of Science and Technology, Islamabad, Pakistan, in 2018. She is currently working as an Assistant Professor with the Department of Computer Science, COMSATS University, Islamabad. Her research interests include deep learning, nature-inspired algorithms, content-centric network (ICN), and caching strategies of CCN.



WADOOD ABDUL received the Ph.D. degree in signal and image processing from the University of Poitiers, France, in 2011. He is currently an Associate Professor with the Department of Computer Engineering, CCIS, King Saud University. His research interests include color image watermarking, multimedia security, steganography, fingerprinting, and biometric template protection.