ISSN: 2249-7196

IJMRR/Jan 2021/ Volume 11/Issue 1/ 1-22

Dr. V Venkata Ramana[1]/ International Journal of Management Research & Review

Software Defect Prediction Under the Impact of Deep Learning Algorithms

Dr. V Venkata Ramana[1,] Dr. V Lokeswara Reddy[2] Dr K Sreenivasa Rao[3], M Ramanjeneya Reddy[4]

[1,2,3, 4]Professor, Department of CSE, K.S.R.M College of Engineering(A), Kadapa

ABSTRACT

Early software flaw detection is crucial for increasing software quality and lowering the costs, time, and effort required for software development. Software faults prediction (SFP) has made extensive use of machine learning (ML), and ML techniques provide a variety of benefits.

results for fault prediction in software. Deep learning excels in a number of fields, including voice recognition, computer vision, and natural language processing. In this research, the performance of two deep learning algorithms—Multi-layer perceptrons (MLPs) and Convolutional Neural Networks (CNN)—is examined in order to address potential influencing variables.

The results of the experiment demonstrate how changing certain factors directly affects the improvement that occurs; these parameters are changed until the right value is found for each one. Additionally, the results demonstrate that the impact of changing the parameters had a significant impact on prediction accuracy, which increased significantly in comparison to the conventional ML algorithm. The tests are run on four widely used NASA datasets to verify our presumptions. The outcome demonstrates how the parameters addressed may boost or lower the measurement of the fault detection rate. Up to 43.5% for PC1, 8% for KC1, 18% for KC2, and 76.5% for CM1 showed improvement.

INDEX TERMS

Deep learning methods, classication, hyperparameters, and software fault prediction.

INTRODUCTION

The most difficult task for software developers is creating high-quality software. For such, software development needs go through a series of tasks under certain conditions.

limitations to develop dependable and high-quality software. The presence of errors, which degrade software quality and render final products unreliable and unsatisfactory, is a significant disadvantage of having excellent quality and trustworthy software. [1] Reference The software development cycle must be well planned and controlled in order to produce high-quality software.

Faults are inescapable and may appear at any stage of the software development process. Software fault prediction, which prevents learning and helps to decrease software failure, is one of the high-quality models that may be used to enhance software fault prediction.

It's very difficult to create software that is error-free.Most of the time, even when a team meticulously follows development procedures, unanticipated deciencies and undiscovered bugs may still surface.

To effectively plan and manage testing and project maintenance, it is necessary to anticipate probable software flaws. The development team will have additional opportunities to run testing many times on modules or lessons with a high fault probability thanks to fault prediction. This will make the defective modules more prominent. The likelihood of eliminating the remaining flaws will thus rise, and any software products made available to end users will be of higher quality.

Additionally, this strategy reduces the project's maintenance and support requirements. Evidently, software flaws may lead to low-quality software. Since fixing these flaws needed a lot of work, SFP has been used to lessen their effects.

The SFP also decreases the expenses, amount of time, and effort needed to produce software products [3]. The expense of finding and fixing errors is said to be the most costly software development activity in reference [38]. an abundance of

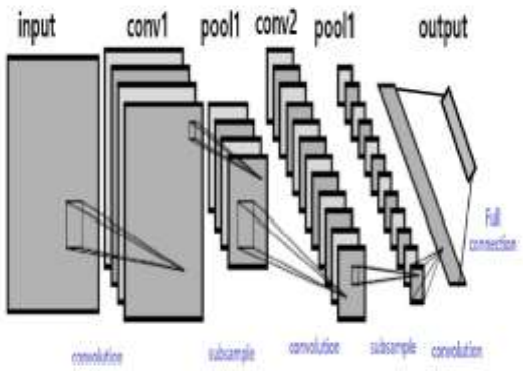

FIGURE 1. CNN architecture [38].

For the purpose of predicting software faults, studies utilizing support vector machines and genetic algorithms have been carried out [42]. Affective Neural Network

Decision tree [44], (ANN) [43], etc. There should be more research done.

The following questions were addressed by Muet al. [45] in their discussion of various deep learning techniques: what are the most popular deep learning models and methods for optimizing them; do they frequently use open source frameworks; what are the most pressing current issues; and what are the suggested future solutions. By identifying the most relevant materials and findings, their systematic review helped us save time and effort.

Deep learning algorithms were utilized in this study while also considering the drawbacks of the earlier ones.

Deep Learning is a field of machine learning that employs supervised and/or unsupervised approaches to learn from several layers of neural networks.

This has had enormous success in a variety of fields [34].

Computational models with numerous layers may learn representations of data at various degrees of abstraction thanks to deep learning [9]. It automatically separates the most important characteristics from the raw data and strengthens it against input fluctuations [33].

Deep learning can also manage enormous volumes of data, provide a variety of models that allow for the use of unlabeled data to discover relevant patterns, and allows for the sharing of representations across various tasks [13].

Convolutional Neural Networks (CNNs) use the convolution mathematical process [5]. It belongs to the feedforward neural network family [10]. CNNs are made up of combinations of stacked convolutional layers that have been split into smaller convolutional layers in order to simplify the computation. Each pooling layer is often positioned after a convolutional layer. Max pooling (subsampling) layers that conform one or more pairs. Then, as seen in Figure 1, fully-connected layers and the nal layer are more elegant.

Depending on their relative positions, neurons in the convolutional layer are linked to neurons in the next layer.

Forward propagation is a technique used in CNN's training process to calculate the true classication of the input data.

The trainable parameters were updated using current parameters and back propagation to reduce discrepancies between the output of the classication process and what was wanted. CNNs

using the back propagation approach for training. It starts by randomly allocating initial weights over the whole network, after which the weight will update.

One benefit of CNN is weight sharing, which lessens the requirement for calculation. Additionally, each stage of the nonlinear computation uses max pooling to decrease the amount of the input data, and subsampling the outcome reduces distortion. The number of connections, shared weights, and downsampling all decrease when the number of parameters is reduced [11]. Better speed, memory savings, and reduced computational complexity are all benefits of CNNs.

The multilayer perceptron or feedforward deep network is the primary illustration of a deep learning model (MLPs).

Deep Learning's accuracy and capacity to handle complicated applications are both constantly improving [5]. The fundamental deep learning model is the multilayer perceptron (MLP).

Deep neural networks, which are extensions of artificial neural networks (ANNs) with many hidden layers, have gained popularity as a result of their astonishing performance improvements on challenging learning tasks and their effectiveness in a variety of machine learning applications. Deep neural networks are thus winning out over shallow networks. The deep neural network includes intricate topologies and hundreds of hyper-parameters. Additionally, the design decision is crucial since finding the ideal architecture for a given challenge is often what separates success from failure. The developer recently concentrated on creating unique structures for brand-new issues.

Higher complexity functions are represented when the number of layers and units is increased.

Deep Learning Algorithms are used in a variety of studies to examine its potential for SFP.

Failures of software may result in lost time, money, and other things. Software should be fault-free since software reliability has become the industry's and community's top issue. Numerous scholars have examined software failure prediction using a variety of methodologies, each of which offers a different level of accuracy. Deep learning

is particularly successful in a variety of fields, including voice recognition, natural language processing, and image processing.

the use of deep learning to the prediction of software faults may provide a fresh contribution to improving fault prediction accuracy as compared to current methods.

In this investigation, the following research questions are addressed:

1. Can the software defects prediction performance be used by deep learning?

2. Does the model's altering architecture

accuracy of algorithms being improved?

3. What deep learning techniques provide the highest SFP performance?

The suggested model's significance may be summed up as follows:

measuring the efficiency of deep learning algorithms for SFP, identifying the optimal algorithm quality, and assessing the efficiency of changing the deep learning algorithms' design. The method will enable developers and testers to concentrate on the code, which will ultimately reduce testing and maintenance costs while also improving the program and boosting overall product dependability.

This paper's major contribution is an investigation into the variables that could affect how well deep learning systems work in the context of SFP. In this study, two basic algorithms—Multi-layer perceptrons (MLPs) and Convolutional Neural Networks—are used to carry out the tests and track the effects on the components (CNN). These elements include the number of layers, epochs, batch size, dropout rate, and optimizer. The result section includes a detailed discussion of a number of comparisons that were made.

LITERATURE REVIEW

The most significant pertinent works that addressed SFP are reviewed in this part, together with the prior findings for the state-of-the-art using ML, NN, and Deep Learning.

There are several studies that show how to increase software quality, make better use of resources, and reduce or eliminate error.

To assure comprehension of the characteristics of SFP, it is vital to investigate these studies.

MACHINE LEARNING

The effective use of ML in SFP has left more room for improvement in prediction accuracy, hence [12] proposes collaborative representation classication (CRC) based soft- ware defect prediction (CSDP) method, designed to categorize if the questioned software modules are defective or not. In the experiment, 10 datasets from NASA MDP were used, and the suggested method was compared against weighted Naive Bayes (NB), cost-sensitive boosting neural network (CBNN), compressed C4.5 decision tree (CC4.5), and coding-based ensemble learning methods (CEL). The outcome suggests

that the suggested technique performed the best.

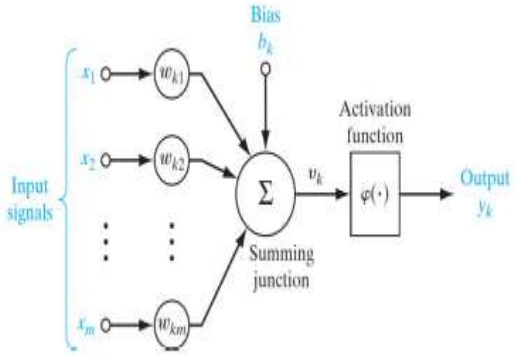

FIGURE 2. Neuron [6].

Majority Vote based Feature Selection algorithm (MVFS), developed by Borandag et al. [39], was used to determine the most important software metrics.

They tested the effectiveness of MVFS using combinations of machine methods and lter (Information Gain, Symmetrical Uncertainty ReliefF feature, Correlation- based approach). They employed PC1, CM1, KC1, and JM1. The findings demonstrate that the MVFS technique may improve defect prediction performance by identifying the most crucial software metrics. In order to improve prediction accuracy and shorten processing time, Reference [40] suggested model incorporates a modified under-sampling approach and a correlation feature selection with ensemble learning. The model has been applied to 10 open source datasets. The results of the studies demonstrated that the suggested model performed flawlessly throughout the prediction process; the improvements in terms of F1 measure went up to 96%.

The following methods forecast the error for the unsupervised data using a clustering model. They propose and assess novel methods, such as K-Sorensen-means clustering, a new SFP clustering technique for K-means that calculates cluster distance using the Sorensen metric. Three datasets—JM1, PC1, and CM1—are included in the suggested methodology.

The findings demonstrate that K-Sorensen clustering outperforms K-Canberra means [14]. Few studies have used clustering for SFP because it is difficult to determine the number of clusters. To solve this problem, expectation-maximization (EM) and the Xmeans model proposed by [15] were used to predict errors when training data were absent. The experiment used data from the AR3, AR4, and AR5 PROMISE repository. The results of the investigation demonstrate that the Xmeans model outperforms the EM and another model from earlier studies.

Furthermore, accuracy increases and reaches 90.48% when data are used without feature selection. Wahono and Herman [16] proposed a strategy that merged the bagging method to address the class imbalance issue with the genetic approach to address feature selection. This model is used on nine NASA datasets, comparing the results from SVM, DT, NN, and statistical classifiers. The best prediction performance was achieved by SVM, which achieved 89.9%. In contrast to filter-based feature ranking strategies like information gain and gain, Wang et al. [17] provide examples of feature ranking methodology.

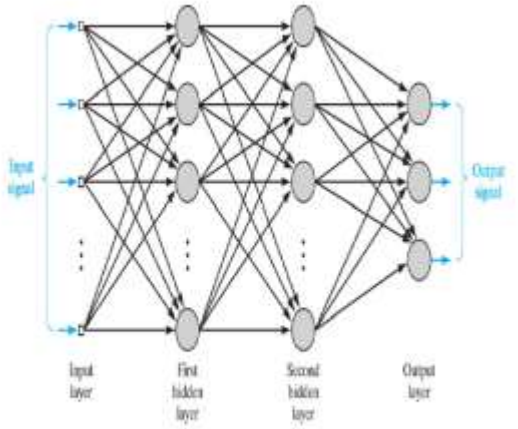

FIGURE 3. Feed forward network [6].

NB, multilayer perceptron, KNN, SVM, and logistic regression are used in the models' construction. The PROMISE data repository provides three datasets. Findings indicate that the

The performance of ensemble technique is superior to that of any other individual ranker.

NEURAL NETWORK APPROACH

Neural networks typically include three parts. First are neurons. Simple computing cells are used. Each neuron has the ability to take in impulses, analyze them, and finally act.

create a signal output. To attain high performance, neural networks need extensive connections between one another.

Figure 2 depicts a model of a neuron with a series of connecting connections, each of which contains an adder for summing the input and an activation function in addition to a weight.

The design of the network is the second element. The feed-forward network, which consists of an input layer, a hidden layer, and an output layer, is the most popular form of neural network design. In a feed-forward network, data flows from input nodes to hidden layers and output nodes, as illustrated in Figure 3, from left to right in the neural network. The third is a learning algorithm, which is a term used to describe a procedure that modifies the weights of the network to lessen output error.

The back propagation algorithm is used to repeatedly train the weights as the mistakes are sent back from the output layer [6], [7]. A neural network's massively parallel distributed topology and capacity for learning provide it with its computational power.

These talents enable it to find effective answers to challenging issues. [7] Reference In comparison to the human brain, neural networks can accomplish difficult tasks and demonstrate their superiority in problem solving. This was clear in a number of fields, including classical analysis, pattern recognition, and voice recognition. NN is a popular supervised learning approach [8] (reference [4]). In reference [4], a neural network model was created to evaluate performance using the accuracy-oriented mean squared error function. Gradient descent was used in the model's development.

Three components commonly make up neural networks. Neurons are the first. The usage of basic computing cells. Every neuron has the capacity to receive impulses, process them, and then act.Output a signal, please. Neural networks need a lot of connections between them in order to operate well.

In Figure 2, a model of a neuron is shown with a network of connections, each of which has a weight, an adder for summing the input, and an activation function.

The second component is how the network is designed. The most common kind of neural network architecture is the feed-forward network, which has three layers: an input layer, a hidden layer, and an output layer. Data travels from input nodes to hidden layers and output nodes in a feed-forward network, as shown in Figure 3, from left to right in the neural network. The third is a learning algorithm, which is the name given to a process that adjusts the network's weights to reduce output error.

As errors are conveyed back from the output layer, the back propagation technique is utilized to continually train the weights [6], [7]. The computing strength of a neural network comes from its massively parallel distributed architecture and learning capability.

It is able to solve complex problems because to these skills. Reference: [7] Neural networks can complete challenging tasks and show why they are better at solving problems than the human brain. This was evident in a variety of domains, including speech recognition, pattern recognition, and classical analysis. NN is a well-liked supervised learning technique [8]. (reference [4]). A neural network model was developed to assess performance using the accuracy-oriented mean squared error function in reference [4]. The model was created using gradient descent.

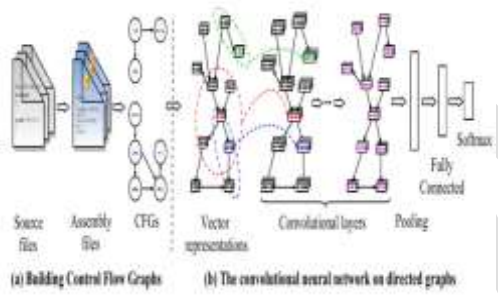

FIGURE 4. DGCNNs steps [3].

The method is divided into four parts, beginning with the extraction of tokens from numerically encoded vectors using Abstract Syntax Trees (ASTs). Then it uses CNN.

and incorporates it with conventional defect prediction tools.

In order to determine if the code les are experiencing bugs or not, it employs the Logistic Regression. The trials, which were conducted across seven open source projects, demonstrate a 12% average improvement of the state-of-the-art approach by the DP-CNN. A prediction model capable of autonomously learning characteristics for describing source code that has been utilized for defect prediction is created by Dam et al. [19]. They use an LSTM that is immediately compatible with the Abstract Syntax Tree representation of source code, which is tree-structured (ASTs). To better reflect the syntactic structure and depth of semantics in source code, the model is constructed as a tree-structured network of LSTM units. The model's function results from training are used to automatically determine if new hires are faulty, whether they are working on the same project or a different one. Additionally, it is able to identify the components in a source code that almost certainly lead to a flaw. This helps in comprehending and recognizing precisely what the model is taking into account, as well as to what extent it has certain flaws.

Their methods consist of four main stages: I parsing a source code file into an Abstract Syntax Tree; (ii) embedding AST nodes, which map the label names of each AST node into a continuous-valued vector of x

length; (iii) feeding the AST embeddings into a tree-based network of LSTMs to obtain a vector representation for the entire AST; and (iv) using a classifier like Logistic Re

They conducted an analysis of the Samsung dataset, which includes datasets from the PROMISE repository and open-source initiatives. The outcome of this methodology proves that it can be used in actual practice.

The performance of the models is impacted by existing features and tree structures that sometimes fall short of capturing the semantics of programs. Previous research have focused on extracting features using tree representations of programs.

[2] suggested a technique to automatically learn fault characteristics in order to thoroughly explore the semantics of programs. They used CNNs based on directed graphs (DGCNNs) to learn semantic characteristics. The crucial process for the concept is creating Control Flow Graphs (CFGs) using Linux's gCC and assembly code.

to get a program's graph representation. CFG is designed to explain how assembly instructions are executed and reveal how the program behaves. The

To create models based on CFG data, another step is to apply a graphical model to CFG datasets that comprise multi-view multi-layer CNNs for directed labeled graphs.

In the first stage, a vertex is represented as a collection of real-valued vectors according to the number of views. Apply two convolution layers, then a dynamic pooling layer, and then compile all of the extracted features from the graphs into a vector. The last stage involves sending a feature vector to an output layer and a fully-connected layer to calculate the category distributions for potential outcomes, as shown in Figure 4.

Four datasets (SUMTRIAN, FLOW016, MNMX, and SUBINC) were collected from the CodeChef website and used in the experiments. They used a number of machine learning methods to create NN, SVM, and KNN prediction models. In comparison to the feature-based strategy (increase from 4.08% to 15.49%) and the tree-based technique (improvement from 1.20% to 12.39%), this approach performs the best.

citation [41] For defect prediction, they coupled the Long Short- Term Memory (LSTM) algorithm with word embedding. The first step in the suggested model's three-step structure pulls a token from its abstract syntax tree.

Second, a vector is created from the token. Third, create a long short-term memory using the vector and its labels.

The LSTM algorithms identify defects by automatically picking up on program semantic data. The studies conducted on eight open-source projects demonstrate that the suggested model works better than cutting-edge fault prediction techniques. Utilizing deep learning algorithms for prediction yields encouraging results; some recently published research employed CNN and MLP without addressing the efficacy of altering the key variables that may directly affect the performance of prediction [46, [47].

PART III: RESEARCH METHODS

We began reviewing the fundamental design processes in this part, which called for employing MLPs and CNN to anticipate software defects using NASA datasets. stages in the approach,

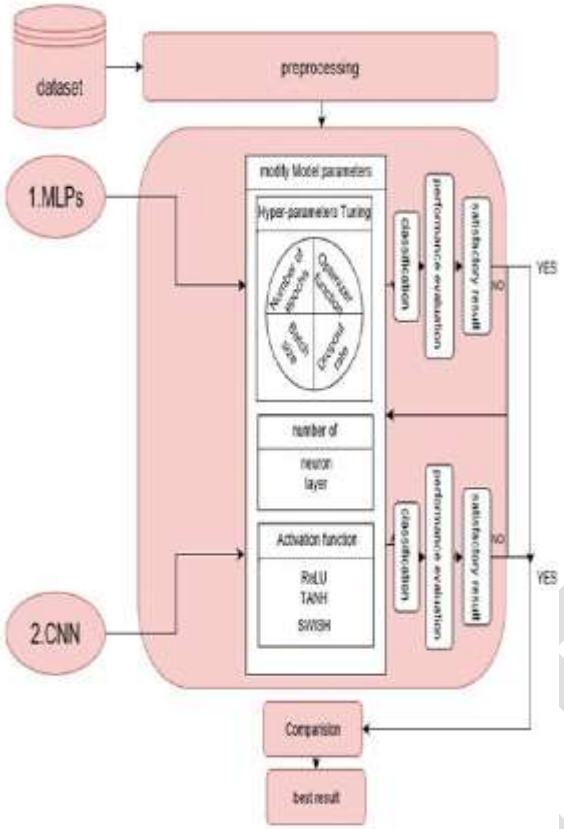

FIGURE 5. Research framework.

As a first step, we chose four datasets with different defect rates, normalized them for these datasets, and then continued. When first applied, the MLP modifies its settings.

Finally, we compared the outcomes to identify the best outcomes obtained and followed the same procedures for CNN. The Python 3.6 language is used to build the MLP and CNN algorithms.

To carry out the studies, a variety of libraries (including Keras, Numpy, Panda, Sklearn, and Matplotlib) were used. The following subsections go into deeper depth about each step after that. Figures 5 and 6, respectively, provide an overview of our suggested technique and the pseudocode.

SELECT DATASET, first.

The datasets are taken from the NASA Metrics Data Program (MDP), and they comprise data from software measurements and related error information gathered. In order to promote repeatable, verifiable, debatable, and/or improved prediction models of software engineering, NASA MDP dataset is made accessible to the general public.

TABLE 1. Dataset characteristics.

Dataset	#attributes	Language	Faulty Percentage	Description
CM1	22	C	9.83%	a NASA spacecraft instrument
KC1	22	C++	15.45%	storage management for receiving and processing ground data
KC2	22	C++	20.50%	Science data processing
PC1	22	C	6.94%	flight software for earth-orbiting satellite

The datasets have been heavily used in software defect prediction experiments. The data sets' characteristics are presented in table 1. The attributes of the datasets are shown in Table 2 [20], [21].

TABLE 2. Attributes of datasets.

Attribute ID	Attribute	Definition
1	LOC	McCabe's line count of code
2	V(g)	McCabe "cyclomatic complexity"
3	EV(g)	McCabe "essential complexity"
4	IV(g)	McCabe "design complexity"
5		Halstead total operators + operands
6	V	Halstead "volume"
7	L	Halstead "program length"
8	D	Halstead "difficulty"
9	I	Halstead "intelligence"
10	E	Halstead "effort"
11		Halstead
12	T	Halstead's time estimator
13	lOCode	Halstead's line count
14	lOComment	Halstead's count of lines of comments
15	lOBlank	Halstead's count of blank lines
16	lOCodeAndCommentc	Numeric
17	uniq_Op	unique operators
18	uniq_Opnd	unique operands
19	total_Op	total operators
20	total_Opnd	total operands
21	branchCount	of the flow graph
22	Problems	{no,yes}

B. REGULATION

When a numerical attribute may find new ranges from an existing range using an equation, normalization is applied.

It is carried out during the preprocessing stage and is helpful for distance measurements and classication algorithms like NN (KNN, clustering). This research used the standardization approach to maintain the natural distribution of the qualities. To change characteristics with a Gaussian Distribution and various means and

distributions, standardization is an effective strategy. We use a Standard Scaler from the scikit-learn module in Python. The Standardization formula is shown in equation (1) below.

$$Z = \frac{x - min(x)}{max(x) - min(x)} \qquad (1)$$

C. USING DEEP LEARNING METHODS

During this stage, we used two algorithms (MLPs and CNN) to examine how well they might improve the accuracy of SFP and identified the performance-influencing factor.

Choosing the various layers, functions, and hyperparameters for each experiment will be done in this stage.

Once we are satisfied with the outcome, we repeat the procedure.

Modifying the model parameters.

The network's parameters must take into account the hyper-parameter activation function and the number of layers in each layer.

1) HIERARCHIVE PARAMETERS

For each test scenario, a different hyper-parameter will be changed to see how it affects accuracy. To choose the proper parameters and get the intended outcomes, tuning the hyper-parameter is essential. Although there are no specific guidelines for selecting the ideal parameters, the decision often depends on the nature and scope of the training dataset. Selecting the appropriate parameters is crucial, but it is a challenging aspect of network training. Hyper-parameter tuning, however, typically relies more on practical expertise than on theoretical understanding. Due to limitations like memory limits, trade-offs are an inherent part of parameter selection [22].

Before documenting the findings of our research's many tests, we examined and assessed the hyper-parameter values for each method. When one of these hyperparameters has been optimized, work on additional hyperparameters to get the optimum outcome. Compare the settings that help algorithms perform better after that.

Remove any values that have a negative impact on the algorithms' outcomes as well. Compare the settings that help algorithms perform better after that.

Remove any values that have a negative impact on the algorithms' outcomes as well.

Below is a list of the hyper-parameters settings: Epoch count: An epoch is the total number of times the data set has been traversed [22]. To find the optimal number of epochs for the network to properly converge, we will vary the number of epochs in our experiment from 10 to 20000. The amount of training samples makes up the batch size. The number of predictions that may be produced at once is controlled by batch size, along with the model. The increased batch size often requires more RAM. citation [23] Contrarily, small-batch sizes are preferred since they lead to convergence in a condensed number of epochs [24].

Test batches will range in size from 1 to 20. Dropout rate: Dropout is a strategy for mixing exponentially many distinct neural network topologies effectively while also preventing over-tetting. The falling of troops happens at random [31]. As seen in Figure 7, dropping out units in a neural network means temporarily eliminating each unit together with all of its incoming and outgoing connections.

In addition to being used to graphical models like Boltzmann Machines, the dropout is utilized to feed forward neural networks [32]. Increments of 0.1 will be used to test various dropout rates between 0.2 and 0.6. Updated weights are applied at the back propagation step to decrease error using the optimizer function.

Adam (adaptive moments) and Adagrad (adaptive gradient) are the two optimization functions we'll be using [5].

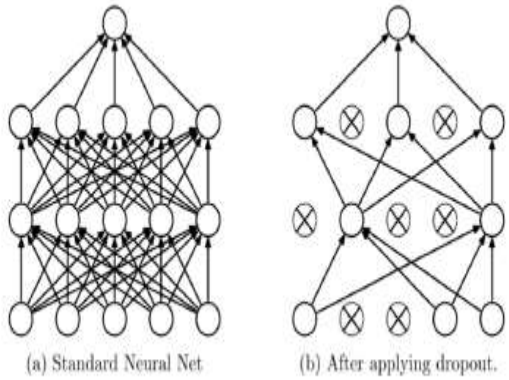

FIGURE 7. Dropout [30].

2) THE NUMBER OF LAYERS

The performance of the networks may be strongly influenced by a variety of factors, including the presence of hidden layers and neurons in those levels. The quantity of layers increases.

3) ACTIVATION FUNCTION

An artificial neuron's activation function determines its output based on an input or combination of inputs.

Each activation function executes an after receiving one x input unique mathematical analysis on it. Deep neural networks frequently use the activation functions sine, rectilinear unit, and hyperbolic tangent. Sigmoid: a popular back propagation NN metric. The range only extends over (0, 1). It is a suitable extension of nonlinearity that restricts earlier uses in NN and also exhibits a suitable level of smoothness [26]. The following formula (2) disallows the Sigmoid function:

$$S(x) = \frac{1}{1+e^{-1}} \qquad (2)$$

The ratio of the hyperbolic sine and cosine functions is known as the hyperbolic tangent (Tanh) [27]. The formula (3) denies the Tanh function as follows:

$$Tanh(x) = \frac{e^{x} - e^{-x}}{e^{x}+e^{-x}} \qquad (3)$$

Rectified linear units (ReLUs) are extensively utilized to train a deeper network than Sigmoid by acting as an activation function for the hidden layers in a Deep Neural Network.

Tanh's activation processes. ReLU enables Deep Neural Networks (DNN) to learn from complex, high-dimensional data more quickly and effectively. The main benefit of ReLU is that it only requires comparison and multiplication, not an expensive computation. ReLU is a particularly good option for DNN because it provides an efficient backpropagation without exploding or disappearing gradient [28].

Swish: is a smooth, non-monotonic function with no upper or lower bounds and similarly or very similar results.

performs better than ReLU on the deep neural network across a range of difficult datasets. As f (x) D x Sigmoid(x), Swish dened [29].

E. COMPARISON

To evaluate the impact of changing the investigated parameters (hyperparameter, number of layers and neurons, activation function), a comparison will be performed.

function).

F. DATA ANALYSIS & INTERPRETATION

This research evaluates the performance of several deep learning algorithms for classification. Use, Detection Rate, and TNR, which are calculated by taking into making both optimistic and pessimistic predictions about things. The following formulae may be used to calculate the performance indicators:

The sensitivity or detection rate measures how well positive instances were recognized. Specify the frequency with which the fault will be positive if the class has a fault. (The percentage of genuine positive results)

D TP/ D = Detection Percentage (TPCFN). The accuracy of a forecast is the degree to which the prediction corresponds to reality, expressed as a percentage. Accuracy

= (TP+TN)/(TP+TN+FP+TN). Specificity/ True Negative Rate (TNR): The ability of the test to identify correctly those do not have the faults. TNR = TN/(TN+FP).

G. HARDWARE SPECIFICATION

It was determined that two computers would be sufficient for testing the implementation, and those machines were utilized throughout the development process. The first, a laptop computer, was mostly used for preparing for and presenting quick and painless diagnostics The second was a virtual computer borrowed from the Computer Center, and it was utilized mostly for long-term testing.

The hardware specifications as in the table 4:

Specifications		Personal Laptop	Virtual Machine
CPU	Type	Intel(R) Core(TM) i5-5200U	Intel(R) Xeon(R) Platinum 8168 CPU
	Speed	2.20GHz	2.70GHz
	Processor	4	32
RAM	Size	12GiB	178GiB
HDD	Model Number	1- PNY CS900 SSD 2- TOSHIBA MQ01ABD1 sdb.	1- Virtual Disk 2- Virtual Disk
	Size	1-240GB 2-931.5G	1- 64G sda 2- 256G sdb

H. SOFTWARE SPECIFICATIONS

The Ubuntu 18.04.1 LTS version of Linux was installed on the virtual computer. The majority of its applications

TABLE 3. Number of EPOCH effect.

PC1					KC1					KC2					CM1				
architecture	No. of epoch	accuracy	Detection rate	TNR	architecture	No. of epoch	accuracy	Detection rate	TNR	architecture	No. of epoch	accuracy	Detection rate	TNR	architecture	No. of epoch	accuracy	Detection rate	TNR
5 batch size and 5layer	1000	.929	.012	.998	10 batch size and 5layer	10000	.840	.386	.925	5 batch size and 5 layer	5000	.821	.519	.897	7 batch size and 5layer	5000	.899	.170	.975
	2000	.928	.054	.994		15000	.854	.406	.951		8000	.819	.504	.899		17800	.905	.191	.979
	10000	.935	.363	.977		25000	.849	.536	.894		15000	.814	.438	.908		20000	.872	.840	.964

TABLE 4. Batch size effect.

TABLE 4. Batch size effect.

PC1					KC1					KC2					CM1				
architecture	batch size	accuracy	Detection rate	TNR	architecture	batch size	accuracy	Detection rate	TNR	architecture	batch size	accuracy	Detection rate	TNR	architecture	batch size	accuracy	Detection rate	TNR
2000 and 5layer	5	.930	.051	.986	10000 and 5layer	5	.842	.488	.906	20000 and 3layer	5	.825	.476	.913	15000 and 5layer	5	.881	.148	.968
	10	.928	.142	.987		10	.846	.429	.922		10	.851	.447	.928		7	.899	.170	.975
	30	.928	.051	.994		15	.843	.388	.924		15	.819	.451	.911		10	.887	.212	.957

in order to give the most objective testing possible, testing is performed without providing any extra services to the client. Ubuntu 18.04.1 LTS was used as the OS on the laptop.

IV. RESULT

Both the MLPs and the CNN are written in Python 3.6.5, with Keras Frameworks used for implementation. Moreover, the Numpy, Panda, and Sklearn libraries were employed. An array of visual tools is used for the development environment was based around the Matplotlib library and Spyder. Nothing in this work has been developed in isolation from the other. We conducted over two hundred tests varying the (hyper-parameter), (activation function), and (layer count) of the system. Modifying the parameters of the model to achieve better results is the subject of the experiments. The obtained results are presented in the following subsections.

MLPS RESULT

The MLPS Outcome A.

The impact of (epochs, batches, dropout rate, Optimizer, layers, and activation function) is addressed by the

tables 3–8 detail the outcomes that the suggested method, using MLPs algorithms, was able to attain.

As shown in table 3, we conducted an experiment using MLPs to investigate the impact of the epoch number. According to table 4, we carried out the following tests to investigate the impact of batch size.

PREVALENCE OF ATTRIBUTION The experiments listed in table 5 were conducted to analyze the impact of the attrition rate.

OPTIMIZER The following tests were conducted to evaluate the impact of Optimizer and are detailed in table 6. Adgrad provided more reliable detection and accuracy rates.

TABLE 5. DROPOUT rate effect.

PC1					KC1					KC2					CM1				
architecture	Dropout rate	accuracy	Detection rate	TNR	architecture	Dropout rate	accuracy	Detection rate	TNR	architecture	Dropout rate	accuracy	Detection rate	TNR	architecture	Dropout rate	accuracy	Detection rate	TNR
10000 , 5batch size and 5layer	.2	.935	.363	.977	15000, 10and 3layer	.2	.853	.371	.941	25000, 15 and 3layer	.2	.810	.429	.908	15000, 7 and 5layer	.2	.891	.224	.964
	.4	.924	.389	.964		.4	.849	.246	.941		.4	.814	.439	.910		.4	.895	.108	.975
	.5	.922	.389	.962		.5	.857	.315	.956		.5	.825	.485	.911		.5	.899	.170	.975
	.6	.930	0	1		.6	.855	.352	.944		.6	.812	.476	.904		.6	.890	.085	.975

TABLE 6. Optimizer Effect.

PC1				
architecture	optimizer	accuracy	Detection rate	TNR
10000 and 5 batch size	adam	.930	0	1.00
	adgrad	.935	.363	.977

COUNT OF SHEETS Table 7 details the trials we ran to determine the impact of varying layer counts.

PURPOSE OF ACTIVATION As shown in table 8, we conducted the experiments below to investigate the impact of activation function.

CNN RESULT

The suggested method used MLPs techniques to produce the following outcomes by adjusting the parameters (number of epochs, batch size, number of layers, activation function). Tables 9, 10, and 11 provide details.

EPISODE COUNT We ran the experiment on CNN shown in table 9 to analyze the impact of epoch number.

SIZE OF EACH BATCH See table 10 for details of an experiment we ran on CNN to measure the impact of different batch sizes.

COUNT OF SHEETS The experiment we ran on CNN is detailed in table 11, and it was designed to assess the impact of a variety of layer configurations.

C. COMPARE AND CONTRAST PERFORMANCE COMPARISONS Metrics

The top outcomes from both algorithms are shown in Table 12. In this case, the findings favor CNN significantly.

Time and experiment count comparisons Differences in the time and number of experiments required to obtain the success with both algorithms that is satisfactory.

Summary results for experiments with MLPs and CNNs showing improvements as parameters were adjusted based on detection rate are shown in Tables 14 and 15.

V. DISCUSSION

We analyze the proper settings of the CNN and MLPs algorithms that provide us relevant predictions in order to discuss and comprehend the findings.

MLPS

Based on experimental results on MLPs algorithm, the pro- posed approach obtained the effective, which achieved by modifying the network parameters as follows:

THE EFFECT OF THE HYPERPARAMETER The num-

ber of epoch had a signi_cant effect, especially in increasing the Detection rate. When we increase epoch number then all of the following are increased: the Detection rate, the accu- racy and the model ability to predict faults. For example, when applying to the PC1 dataset and increasing the number of the epoch from 1000 to 10000, the Detection rate ratio increases from .012 to .363 and accuracy from 92 to 93.5.

However, after reaching the optimal number of the epoch,

TABLE T. Number of layers effect.

PC1					KC1					KC2					CM1				
architecture	No. of layer	accuracy	Detection rate	TNR	architecture	No. of layer	accuracy	Detection rate	TNR	architecture	No. of layer	accuracy	Detection rate	TNR	architecture	No. of layer	accuracy	Detection rate	TNR
10000 and 5 batch size	3	0.95	.350	.968	15000 and 10 batch size	3	.857	.315	.956	20000 and 10 batch size	3	.831	.447	.928	17000 and 7 batch size	3	.901	.327	.982
	4	.925	.350	.968		4	.853	.407	.955		4	.822	.476	.989		4	.895	.370	.971
	5	.935	.363	.977		5	.846	.371	.953		5	.816	.495	.896		5	.905	.391	.979

PC1					KC1					KC2					CM1				
architecture	Activation function	accuracy	Detection rate	TNR	architecture	Activation function	accuracy	Detection rate	TNR	architecture	Activation function	accuracy	Detection rate	TNR	architecture	Activation function	accuracy	Detection rate	TNR
10000 , 5 batch size and 5layer	ReLU	.935	.363	.977	15000 , 10 batch size and 3layer	ReLU	.857	.315	.956	20000 , 10 batch size and 3layer	ReLU	.825	.485	.911	17000 , 7 batch size and 5layer	ReLU	.965	.191	.979
	Tanh	.926	.194	.981		Tanh	.855	.199	.973		Tanh	.821	.439	.920		Tanh	.895	.0212	.964
	Swish	.920	.083	.988		Swish	.852	.165	.977		Swish	.808	.420	.910		Swish	.903	0	.997

the ratio of the TNR drops and, there is a consequent \sdecrease in the accuracy. For example, when applying to\sthe KC1 dataset and increasing the number of the epoch from the ideal number (15000) to 25000, the percent- \sage of TNR declines from .931 to.894 and the accuracy \sof .854 to .849. The influence of the batch size is comparable to \sthat of the epoch. As the batch size grows, the accuracy \simproves, until it reaches the ideal size. After then, any \sincrease in the batch size leads in a drop in accuracy.

For example, when the batch is adjusted from 5 to 7 and then \sto 10, the accuracy and TNR vary as follows: accuracy \s(.891- .899-.891), TNR (.968-.975-.957). (.968-.975-.957). The findings showed \sthat the best dropout rate was when using the value (.5) except \sPC1 where it was (.2). (.2). This may be due to a tiny percentage \sof errors that PC1 possesses. Also, the Agrad optimizer is superior than \sAdam.

THE EFFECT OF THE NUMBER OF LAYERS The use \sof ve layers for the PC1 and KC1 dataset produced the best \sresult, in terms of accuracy, Detection rate, TNR. On the other \shand, KC1 and KC2 delivered the greatest outcomes when employing three \slayers. In both situations with a larger number of layers, \sthe Detection rate is enhanced while diminishing the TNR of the \sdata that affects the accuracy. Through the findings mentioned \sabove, the number of layers is variable according to the \sdatabase itself. There may be a link between ratios \sof defects and the number of cases with the optimal number \sof the layers. This is what we will aim to explore and investigate \sin future studies.

TABLE 9. Number of EPOCH effect.

PC1					KC1					KC2					CM1				
architecture	No. of epoch	accuracy	Detection rate	TNR	architecture	No. of epoch	accuracy	Detection rate	TNR	architecture	No. of epoch	accuracy	Detection rate	TNR	architecture	No. of epoch	accuracy	Detection rate	TNR
15 batch size and 3layer	2000	.966	.608	.993	15 batch size and 3layer	2000	.989	.987	1.00	15 batch size and 3layer	2000	.816	.812	.816	10 batch size and 3layer	2000	.936	.387	.997
	3000	.969	.782	.883		3000	.987	1.00	.991		3000	.887	.968	.991		3000	.925	.285	.995
	4000	.978	.739	.996		4000	1.00	1.00	1.00		4000	.993	.992	1.00		4000	.973	.823	.992

TABLE 10. Batch size effect.

PC1					KC1				KC2				CM1			
architecture	batch size	accuracy	Detection rate	TNR	architecture	batch size	accuracy	Detection rate	TNR	architecture	batch size	accuracy	Detection rate	TNR	architecture	batch size	accuracy	Detection rate	TNR
3000 and 3layer	5	.969	.565	1.00	10000 and 5layer	5	1.00	1.00	1.00	20000 and 3layer	5	.923	.843	.943	15000 and 5layer	5	.892	.858	1.00
	10	.954	.347	1.00		10	.999	.992	1.00		10	.955	.987	.951		10	.973	.823	.992
	15	.969	.702	.893		15	1.00	1.00	1.00		15	.897	.968	.991		15	.932	.705	.962

THE EFFECT OF THE ACTIVATION FUNCTION by \scomparing the outcomes in which several activation functions\swere utilized, the ReLU exhibited better results over other activation functions.

B. CNN ALGORITHM

Based on our research and the data we acquired from \sthese trials, there was an influence caused by modifying \sthe network design. This impact was as follows:

INFLUENCE OF THE HYPERPARAMETER The \sincrease in the number of epoch had a favorable influence on all the \sdatasets on which CNN was used, whether it was accuracy \sor other ways of measures. 4000 epoch was the optimal \snumber of epoch utilized for all tests.

The influence of the batch size is comparable to that of the \sepoch. As the batch size grows, the proportion of means \sof measurements increases, until the ideal size is attained.

For CM1 the ideal batch size was 10 and was 15 for the \srest of datasets.

THE EFFECT OF THE NUMBER OF LAYERS The use \sof ve layers for the PC1 and KC1 dataset produced the \sbest results across the four statistical measures: accuracy, \sDetection rate, TNR. On the other hand, KC1 and KC2 gave \sthe greatest outcomes when employing three layers. In both situations with \san increasing number of layers the Detection rate increases, \swhile limiting the speci city of the data decreases the accu- \sracy. Through the findings provided above, the number of \slayers is variable according to the database itself. There \smay be a link between ratios of defects has it and the \snumber of occurrences with the optimal number of the layers.

This is what we will aim to explore and investigate in future \sresearch.

TABLE 11. Number of a layer effect.

PC1					KC1				KC2				CM1			
architecture	No. of layers	accuracy	Detection rate	TNR	architecture	No. of layers	accuracy	Detection rate	TNR	architecture	No. of layers	accuracy	Detection rate	TNR	architecture	No. of layers	accuracy	Detection rate	TNR
4000 and 10 batch size	1	.954	.347	1.00	1000 and 5 batch size	1	.993	.951	1.00	4000 and 15 batch size	1	.935	.812	.967	4000 and 10 batch size	1	.892	.0581	1.00
	2	.963	.478	1.00		2	.996	.987	.998		2	.967	.968	.967		2	.946	.588	.992
	3	.978	.739	.986		3	1.00	1.00	1.00		3	.993	.968	1.00		3	.975	.823	.992

TABLE 12. Performance metrics comparisons.

Algorithms	PC1			KC1			KC2			CM1		
	accuracy	Detection rate	TNR	accuracy	Detection rate	TNR	accuracy	Detection rate	TNR	accuracy	Detection rate	TNR
MLPs	.935	.363	.977	.857	.315	.956	.851	.447	.928	.905	.191	.979
CNN	.978	.739	.996	1.00	1.00	1.00	.995	.968	1.00	.973	.823	.992

TABLE 13. Time and number of experiments comparisons.

Algorithms	PC1			KC1			KC2			CM1			total		
	No. of experiment	Average time(hours)	total	No. of experiment	Average time(hours)	total	No. of experiment	Average time(hours)	total	No. of experiment	Average time(hours)	total	No. of experiment	hour	day
MLPs	60	20	1200	30	28	840	25	12	252	35	12	420	150	2712	113
CNN	24	4	96	25	5	125	20	3	60	30	3	90	60	371	16

THREAT TO VALIDITY

Multi-layer perceptrons (MLPs) and Convolutional Neural Networks (CNNs) are two deep learning algorithms that are examined in this study to address the elements that might have an impact on the accuracy of these models.

In spite of the fact that the results of the experiments reveal how adjusting the selected parameters has a noticeable impact on the prediction performance, there are still dangers to the construct validity with respect to the generalizability of our findings.

The addressed parameter settings presented the greatest threat to our study's internal validity since they determined which parameters would be influenced by our suggested method and on which settings our comparisons would be based

TABLE 14. Effects of modification (MLP5).

PC1			KC1			KC2			CM1		
Number of epoch	Batch size	Dropout rate	Activation function	Number of layers	Number of epoch	Batch size	Dropout rate	Activation function	Number of layers	Number of epoch	Batch size	Dropout rate	Activation function	Number of layers	Number of epoch	Batch size	Dropout rate	Activation function	Number of layers
35.1%	9.1%	2.6%	28%	1.3%	15.2%	10.1%	5.6%	15%	9.2%	8.1%	8.9%	5.6%	6.5%	4.8%	16.1%	6.4%	5.4%	19.1%	6.4%

TABLE 15. Effects of modification (MLP5).

PC1			KC1			KC2			CM1		
Number of epoch	Batch size	Number of layers	Number of epoch	Batch size	Number of layers	Number of epoch	Batch size	Number of layers	Number of epoch	Batch size	Number of layers
18%	43.5%	39.2%	1.3%	8%	4.9%	18%	14.4%	15.6%	53.8%	76.5%	76.5

Instrumentation of the code to which it is addressed: the source code we used in our experiment was written to give an advantage to the two algorithms tested here.

We conclude by collecting a large, standard dataset derived from actual experiments.

We may not use the most advanced or extensive data set for testing because of concerns about external Validity.

Further, there are likely too many other algorithms in the literature for the addressed and implemented algorithms to be sufficient for generalizing the result.

The authors of this piece make the following attempts to lessen the impact of both internal and external dangers:

1 - Other deep learning algorithms exist in the literature; as future work, we intend to address additional Deep Learning algorithms and conduct thorough comparisons in order to further reduce this threat. In spite of this danger, we investigated the most popular deep learning algorithms used in previous software engineering studies to assess the effectiveness of the factors in fault prediction [2, [18], [32], [35], [36], etc. As a result, we think this side of the argument poses little danger to the construct's validity. The effects of four variables are investigated in this work. We plan to mitigate this risk in the future by attending to other elements that can improve or hinder the efficacy of the algorithms. Three, the PROMISE dataset, which was used in this research, is a widely available public dataset that contains information about actual cases of software failure.

similar characteristics. However, we have selectively used the dataset's offered information and implemented preprocessing procedures to extract the data that is most important to them.

Applications.

Standard performance metrics for fault prediction (TNR, Accuracy, and the error ratio) were used to reduce validity risks as much as possible. In the future, though, we want to look into additional publicly available or even commercial data sets that also reflect a wide range of software products.

VI. CONCLUSION AND FUTURE WORK

Deep learning is a promising subset of machine learning, and this study demonstrates how it may be applied to provide concrete results in the field of prediction.

in terms of forecasting in fields as diverse as computer science, NLP, bioinformatics, software design, etc. This article's writers set out to find the answers to two primary research questions: can adjusting an algorithm's parameters improve its performance in terms of accuracy, and, among the deep learning algorithms that have been researched, which one yields the best SFP results? The primary objective of this research is to discover what aspects of deep learning systems' SFP performance may be improved upon.

A widely-used data collection has been utilized in a number of experiments, which have been followed by analysis and comparisons. According to the experiment results, measured by means of TNR, TNR percentage, and detection rate. The CNN algorithm produced stellar results, reaching 100% for the KC1. Changing gears to the editing parameters influencing, as the total number of parameters rises, each individual parameter contributes positively to achieving the optimum outcomes. The findings indicated that the ideal number of layers for each dataset increases with the number of layers. To top it all off, when compared to other activation functions, ReLU performed very well.

In conclusion, the trials showed that improving parameters had outstanding benefits, yielding excellent findings, in particular when measuring the detection rate. Our long-term goal is to determine whether or not the data set itself plays a significant role (domain), or if the results instead simply depend on the algorithms' parameters, and this will need more tests and the use of other data sets in the near future. Since the usefulness of all hyper parameters was not explored in this study, we want to tackle additional issues in future research. To that end, this research endeavors to identify the most effective deep learning algorithms for SFP. The correlation between the dataset's failure ratio and the algorithm's settings is an interesting research question for the future. The next step, after discovering the link, is to create a platform that use deep learning algorithms for SFP and, maybe, other fields.

REFERENCES

[1] S. Parnerkar, A. V. Jain, and C. Birchha, ``An approach to ef_cient software bug prediction using regression analysis and neural networks," *Int. J. Innov. Res. Comput. Commun. Eng.*, vol. 3, no. 10, Oct. 2015.

[2] A. V. Phan, M. L. Nguyen, and L. T. Bui, ``Convolutional neural networks over control _ow graphs for software defect prediction," in *Proc. IEEE 29th Int. Conf. Tools Artif. Intell. (ICTAI)*, Nov. 2017, pp. 45_52.

[3] E. Erturk and E. A. Sezer, ``Iterative software fault prediction with a hybrid approach," *Appl. Soft Comput.*, vol. 49, pp. 1020_1033, Dec. 2016.

[4] R. Kumar and D. Gupta, ``Software Bug Prediction System Using Neural Network," *Eur. J. Adv. Eng. Technol.*, vol. 3, no. 7, pp. 78_84, 2016.

[5] I. B. Y. Goodfellow and A. Courville, *Deep Learning*, 1st ed. Cambridge, U.K.: MIT Press, 2016.

[6] S. Haykin, *Networks and Learning Machines*. London, U.K.: Pearson, 2009.

[7] Y.-S. Su and C.-Y. Huang, ``Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models," *J. Syst. Softw.*, vol. 80, no. 4, pp. 606_615, Apr. 2007.

[8] A. Pahal and R. S. Chillar, ``A hybrid approach for software fault predic- tion using arti_cial neural network and simpli_ed swarm optimization," *IJARCCE*, vol. 6, no. 3, pp. 601_605, Mar. 2017.

[9] Y. LeCun andY. H. Bengio And Hinton, ``Deep learning," *Nature*, vol. 521, no. 7553, pp. 436-444, 2015.

[10] S. Yang, L. Chen, T. Yan, Y. Zhao, and Y. Fan, ``An ensemble classi_cation algorithm for convolutional neural network based on AdaBoost," in *Proc. IEEE/ACIS 16th Int. Conf. Comput. Inf. Sci.*, May 2017, pp. 401_406.

[11] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello, ``Hardware accelerated convolutional neural networks for synthetic vision systems," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2010, pp. pp. 257_260.

[12] C. W. S. Jin Jin and M. J. Ye, ``Arti_cial neural network-based metric selection for software fault-prone prediction model," *IET Software*, vol. 6, no. 6, pp. 479_487, Dec. 2012.

[13] C. Zhang, P. Patras, and H. Haddadi, ``Deep learning in mobile and wireless networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2224_2287, 3rd Quart., 2019.

[14] D. Kaur, A. Kaur, S. Gulati, and M. Aggarwal, ``A clustering algorithm for software fault prediction," in *Proc. Int. Conf. Comput. Commun. Technol. (ICCCT)*, Sep. 2010, pp. 603_607.

[15] M. Park and H. Hong, ``Software fault prediction model using clustering algorithms determining the number of clusters automatically," *Int. J. Softw. Eng. Appl.*, vol. 8, no. 7, pp. 199_204, 2014. [16] R. S. Wahono and N. S. Herman, ``Genetic feature selection for software defect prediction," *Adv. Sci. Lett.*, vol. 20, no. 1, pp. 239_244, Jan. 2014.