Received March 5, 2020, accepted March 30, 2020, date of publication April 2, 2020, date of current version April 16, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2985290

The Influence of Deep Learning Algorithms Factors in Software Fault Prediction

OSAMA AL QASEM[1], **MOHAMMED AKOUR**[1,2], **AND MAMDOUH ALENEZI**[3]
[1]Information Systems Department, Yarmouk University, Irbid 21163, Jordan
[2]Computer Engineering Department, Al Yamamah University, Riyadh 13541, Saudi Arabia
[3]Computer Science Department, Prince Sultan University, Riyadh 11586, Saudi Arabia

Corresponding author: Mohammed Akour (mohammed.akour@yu.edu.jo)

ABSTRACT The discovery of software faults at early stages plays an important role in improving software quality; reduce the costs, time, and effort that should be spent on software development. Machine learning (ML) have been widely used in the software faults prediction (SFP), ML algorithms provide varying results in terms of predicting software fault. Deep learning achieves remarkable performance in various areas such as computer vision, natural language processing, speech recognition, and other fields. In this study, two deep learning algorithms are studied, Multi-layer perceptron's (MLPs) and Convolutional Neural Network (CNN) to address the factors that might have an influence on the performance of both algorithms. The experiment results show how modifying parameters is directly affecting the resulting improvement, these parameters are manipulated until the optimal number for each of them is reached. Moreover, the experiments show that the effect of modifying parameters had an important role in prediction performance, which reached a high rate in comparison with the traditional ML algorithm. To validate our assumptions, the experiments are conducted on four common NASA datasets. The result shows how the addressed factors might increase or decrease the fault detection rate measurement. The improvement rate was as follows up to 43.5% for PC1, 8% for KC1, 18% for KC2 and 76.5% for CM1.

INDEX TERMS Deep learning algorithms, software fault prediction, classification, hyper parameters.

I. INTRODUCTION

Developing high-quality software is one of the most challenges for software engineers. For that, software development should pass through a sequence of activities under certain constraints to come up with reliable and high quality software. A major drawback to having good quality and reliable software is the occurrences of faults, where faults degraded the software quality and become unreliable end products also not acquire customer satisfaction. Reference [1] In order to achieve high-quality software, suitable planning, and control of software development cycle measures must be followed. The existence of faults is inevitable and it might occur in various phases of software development. One of the quality models that help to reduce software failure is Software fault prediction that also helps to avoid learning may provide valuable improvement in software fault prediction.

Developing software free from faults is very difficult.

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

Most of the time, unforeseen deficiencies and unknown bugs might be exposed even when a team has carefully applied development methodologies. It is very important to predict the potential faults of any software and to apply better planning and management for testing and maintenance of a project. Fault prediction will give the development team more chances to execute testing more than once on modules or files with high faults probability. This will lead to more focus on the faulty modules. As a result of this, the probability of fixing the remaining faults will increase and any software products released to end-users will become more qualified.

This approach also decreases the maintenance and support efforts for the project. Low quality of software can obviously be caused by software faults, these faults required extensive effort to rectify the errors, and SFP has been used to reduce the impact of these faults.

The SFP also reduce the costs, time and effort should be spent on software products [3]. Reference [38] reported that the most expensive software development activities are the cost finding and correcting faults. a large number of

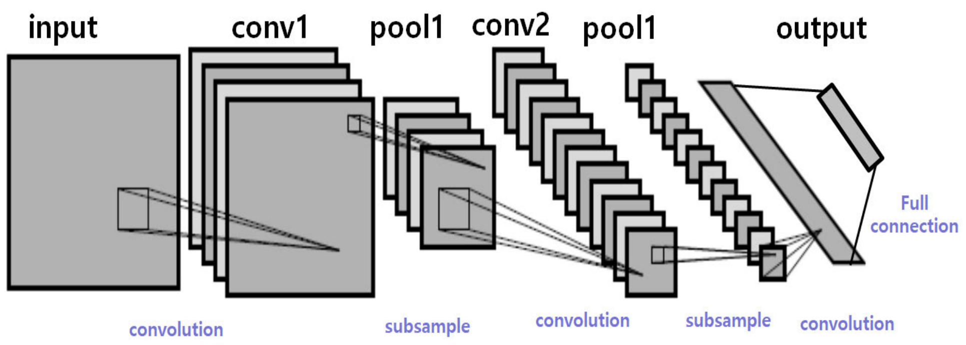


FIGURE 1. CNN architecture [38].

researches have been conducted using machine learning techniques for software fault prediction such as support vector machine, genetic algorithm [42] Artificial Neural Network (ANN) [43], decision tree [44], etc. More efforts need to be explored.

Mu *et al.* [45], addressed different deep learning techniques and answer the following questions, what are the common models of deep learning and their optimization methods, are they commonly using open source frameworks, what are the major existing problems and what are the future suggested solution. Their systematic review saved our effort and time by find the most appropriate resources and findings.

In this research deep learning algorithms had been used with keeping in mind the shortcomings of the previous ones.

Deep Learning is a set of techniques used for learning from many layers in Neural Networks; it is a subfield of machine learning that uses supervised and/or unsupervised strategies.

This has magnificent success in various domains [34]. Deep learning lets computational models that are made from multi layers to learn representations of data with multiple levels of abstraction [9]. It automatically extracts essential features from raw data and makes it robust, with respect to variations in input [33].

Moreover, Deep learning has the capability to handle large amounts of data, provides a lot of models that permit exploiting unlabeled data to learn useful patterns, representations learned by deep neural networks can be shared across different tasks [13].

Convolutional Neural Networks (CNNs) employs a mathematical operation called convolution [5]. It is a type of feedforward neural network [10]. CNNs consist of combinations of the stacked convolutional layers that are divided into smaller sized convolutional layers to reduce the computational complexity. Max pooling (subsampling) layers that conform one or more pairs (often each pooling layer is placed after a convolutional layer). Then fully-connected layers and the final layer are a classifier as shown in Figure 1. Neurons in the convolutional layer are connected to the neurons in the next layer, depending on their relative location. In CNN the training process contains forward propagation used to compute the actual classification of the input data with current parameters, and back propagation used to bring up to date the trainable parameters for minimizing the differences between the actual and desired classification output. CNNs are trained with the back propagation algorithm. It begins by giving initial weights randomly for the entire network, then the weight will update.

Weight sharing reduces the need for computation and is one of the advantages of CNN. Also, max pooling is used to reduce the input data size at each step of nonlinear computation, and subsampling the result minimizes the effect of distortion. While minimizing the number of parameters it is accompanied by the decreasing number of connections, shared weights, and downsampling [11]. CNNs give better performance, savings in memory and computation complexity requirements.

The essential example of a deep learning model is the feedforward deep network or multilayer perceptrons (MLPs). The capability of Deep Learning to solve complex applications and its accuracy continually increases [5]. Multilayer Perceptron (MLP) is the essential deep learning models. Extending Artificial NNs with many layers, Artificial Neural Networks with multi-hidden layers called deep neural networks have become popular because leading to amazing performance gains on difficult learning tasks and success in various machine learning projects. Therefore, deep neural networks are becoming favored over shallow networks. The deep neural network has hundreds of hyper-parameters and complex topologies. Moreover, the choice of design very important; most of the time success counts on finding the right architecture for the problem. Recently the developer focused on designing distinct architectures on new problems. When the numbers of layers and units are increased it will represent functions with higher complexity.

Deep Learning Algorithms are employed in several type of research to investigate their capability in SFP.

Software failures have economic impact, time loss and other effects. The reliability of the software become the core concern for the industry and community, therefore, the software should be free of fault. Software fault prediction is studied by many researchers using various approaches, each approach provide different degree of accuracy. Deep learning proves to be very effective in various domain, like image processing, speech recognition, natural language processing. Using Deep learning in the area of Software fault prediction might show new contribution in enhancing the accuracy of fault prediction in comparison with existing approaches.

The following research questions are addressed in this study:

1. Whether the deep learning can be leverages the software faults prediction performance?
2. Does the modifying architecture of the model make enhancement to algorithms accuracy?
3. Which deep learning algorithms provide the best SFP performance?

The significance of the proposed model can be summarized as follows:

Measuring the effectiveness of deep learning algorithms for SFP and determined the best algorithm quality, addressing the effectiveness of the modifying architecture of the deep learning algorithms. The technique will be helpful to developers and testers to focus on code, which will eventually restrain the testing and maintenance cost along with contributing towards improvement of software and the reliability for the whole product.

The main contribution of this paper is to investigate the factors that might have an influence on the performance of deep learning algorithms in the area of SFP. In this paper, two main algorithms are used to conduct the experiments and monitor the factors influence i.e. Multi-layer perceptron's (MLPs) and Convolutional Neural Network (CNN). These factors are: number of layers, numbers of epoch, the batch size, dropout rate and the optimizer. Several comparisons are performed and discussed in detail in the result section.

II. LITERATURE REVIEW

In this section, we review the most important relevant studies that focused on SFP and discussed the previous results for the-stat-of-art, which used ML, NN, and Deep Learning. There are many studies improved software quality, and making better use of resources, and reducing or eliminating fault. Therefore, it is necessary to explore these researches to ensure understanding of the aspects of SFP.

A. MACHINE LEARNING

ML successfully applied in SFP and still there is a lot of space to improve the accuracy of prediction for that [12] propose collaborative representation classification (CRC) based software defect prediction (CSDP) approach, prepared to classify whether the query software modules are faulty or non-faulty.

Used ten datasets from NASA MDP in the experiment, and compare the proposed approach with various approaches that are weighted Naive Bayes (NB), cost-sensitive boosting neural network (CBNN), Compressed C4.5 decision tree (CC4.5), and coding based ensemble learning (CEL). The result appears the best performance was for the proposed approach.

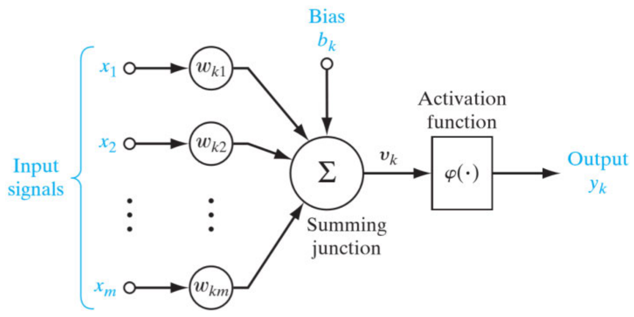


FIGURE 2. Neuron [6].

Borandag *et al.* [39] developed a method called Majority Vote based Feature Selection algorithm (MVFS) that had been used to identify the most influential software metrics.

They used PC1, CM1, KC1, and JM1 to exam the efficiency of MVFS, used of combinations of filter (Information Gain, Symmetrical Uncertainty ReliefF feature, Correlation-based approach) and machine algorithms. The results show that the MVFS method able to enhances defect prediction performance by finding out the most influential software metrics. Reference [40] proposed model includes a modified under-sampling method and a correlation feature selection with ensemble learning to increase the accuracy of prediction and decrease processing time, the model had been applied on ten open source datasets. The experiments showed that the proposed model gave a perfect performance in the prediction process, the improvements reached up to 96% in terms of F1 measure.

The following techniques use a clustering model to predict the fault for the unsupervised data. They suggest and evaluate new algorithms, such as K-Sorensen-means clustering, which is a new SFP clustering algorithm for K-means, using Sorensen distance to calculate cluster distance. JM1, PC1, and CM1 are three datasets subject to the proposed approach.

The results show the K-Sorensen clustering advantage compared to K-Canberra means [14]. Clustering is applied in few studies for SFP because it has some problems like the difficulties to determine the number of clusters, to solve it the expectation-maximization (EM) and Xmeans model suggested by [15] to predict the fault when training data are not present, for experiment used AR3, AR4, and AR5 PROMISE repository data. The analysis data show that the Xmeans is superior over EM and another model from previous research. Furthermore, using data without feature selection gives better accuracy reaching 90.48%. Wahono and Herman [16] proposed a model that combined the genetic algorithm to deal with the feature selection and bagging algorithm to deal with the class imbalance problem. This model is applied over nine NASA datasets and compares the result among SVM, DT, NN and Statistical Classifiers that show a remarkable improvement in prediction performance for most classifiers and the best was for SVM 89.9%. Wang *et al.* [17] use ensembles of feature ranking technique with comparison to filter - based feature ranking techniques like information gain, gain

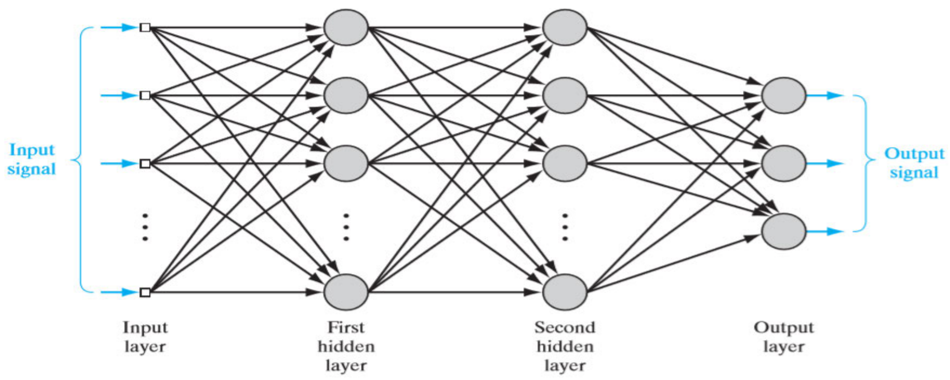


FIGURE 3. Feed forward network [6].

ratio, the models are built with NB, multilayer perceptron, KNN, SVM, and logistic regression. Three datasets from the PROMISE data repository are used. Results reveal that the ensemble technique has the best performance over any other individual ranker.

B. NEURAL NETWORK APPROACH

Usually, neural networks consist of three components. The first is neurons. It is simple computing cells. Each neuron can receive input signals, process the signals and finally produce an output signal. Neural networks employ a massive interconnection between them to achieve good performance.

Figure 2 shows the model of a neuron that has a set connecting links, each of which is characterized by weight, an adder for summing the input and an activation function.

The second component is network architecture. The feed-forward network is the most common type of neural network architecture that is composed of an input layer, a hidden layer, and an output layer. In a Feed-forward network the information moves forward in the neural network from left to right, from the input nodes, through the hidden layers to the output nodes as shown in Figure 3. The third is a learning algorithm that is used during the learning processes to describe a process that adjusts the weights of the network to reduce the errors of the outputs.

The weights are iteratively trained with the errors propagated back from the output layer by back propagation algorithm [6], [7]. A neural network derives its computing power through its massively parallel distributed structure and its ability to learn.

These capabilities make it able to find good solutions to complex problems. Reference [7] Neural networks can perform complex functions and present their superiority over the human brain to solve the problems. That was obvious in many fields like classification, pattern recognition, and speech recognition. Reference [4] NN is a dominating supervised learning technique [8]. Reference [4] designed a neural network model to measure the performance using the mean squared error function that provides the result in terms of accuracy. The model was designed using Gradient descent with adaptive learning (GDA), Baysian Regulation (BR) and Levenberg Marquardt (LM) techniques to train the neural network. This configuration was done on MATLAB 2013. The result reveals prediction in neural networks by using the GD. Its best-predicted accuracy was 98.97% on dataset ANT 1.7 dataset. Additionally, it is considered a faster technique. Jin *et al.* [12] used ANN to reduce the dimensionality of the metrics space that is combined with the support vector machine for SFP. They propose a new software fault-prone prediction model called SFPM based on the concept of the filter model and sequential search strategy. They used the open-source WEKA, and four mission-critical NASA software projects datasets (PC1 and CM1(C)), (KC1 and KC3 (C++, Java)). SVM prediction performance was compared with the four models DT, KNN, logistic regression (LR) and NB. SFPM has been proven to be effective for establishing a relationship between software metrics and fault-proneness. Reference [3] used ANN with other models to compose a hybrid model, employed the ANN and Adaptive Neuro-Fuzzy Inference System. The model suggests the iterative prediction starts with a Fuzzy Inference Systems (FIS) when no data is available for the software project. When the data becomes available the data-driven method is used.

The dataset in this research PROMISE (consist of 18 object oriented metrics) was used for comparison with the proposed model. ANN, Adaptive Neuro-Fuzzy Inference System (ANFIS), FIS are three different classification methods employed to predict software faults. The experiments are performed in MATLAB to validate the performance. The proposed iterative fault prediction model is implemented as a plugin for the Eclipse environment. Therefore, the results indicate that the iterative SFP is successful and can be transformed into a tool that can automatically locate fault-prone modules and the performance of the ANN exceeds the performance of the ANFIS.

C. DEEP LEARNING

Li *et al.* [18] propose Defect Prediction via Convolutional Neural Network (DP-CNN). CNN is used to automatically learn the semantic and structural features of programs.

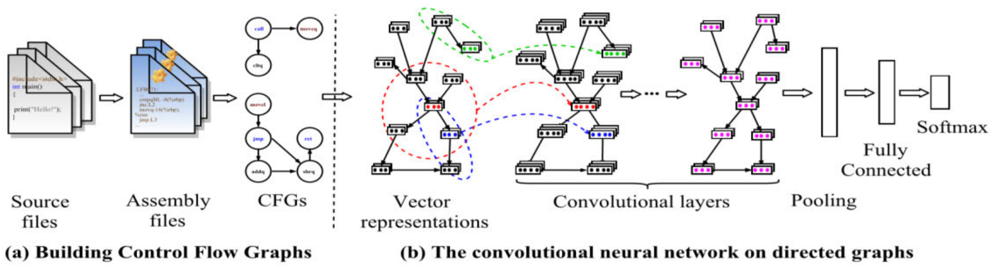


FIGURE 4. DGCNNs steps [2].

The approach contains four phases, starting with Abstract Syntax Trees (ASTs) which is used to extract tokens that are encoded into numerical vectors. Then it employs CNN and combines it with traditional defect prediction features. Finally, it uses the Logistic Regression to decide if the code files are having buggies or not. The experiments were made over seven open source projects and show that the DP-CNN improves the state-of-the-art method on average 12%. Dam *et al.* [19] develop a prediction model able to learn features automatically for representing source code that has been used for defect prediction. They use a tree-structured Long Short- Term Memory network (LSTM) that matches directly with the Abstract Syntax Tree representation of source code (ASTs). The model is built as a tree-structured network of LSTM units to reflect better syntactic and many levels of semantics in source code. The function results from training the model are used to automatically decide the defectiveness of new files whether within the project or in a different project. Also, it is capable of locating the parts in a source file that probably causes a defect. This aids in understanding and identifying exactly what the model is considering and to what degree it has specific defects. Their approaches have four main phases that are: (i) Parse a source code file into an Abstract Syntax Tree; (ii) embed AST nodes that are used to map the label name of each AST node into a fixed-length continuous-valued vector; (iii) input the AST embedding's to a tree-based network of LSTMs to obtain a vector representation for the whole AST; and (iv) use classifier such as Logistic Regression to predict defect by using the vector from previous phase as an input. They performed an evaluation of the Samsung dataset that contains open-source projects and PROMISE repository datasets. The result has been obtained from this approach demonstrates that can be applied to practice.

Previous studies focus on extracting features using tree representations of programs, the performance of the models is affected due to existing features and tree structures that sometimes fail to capture the semantics of programs. To investigate the semantics of programs deeply, [2] proposed a model to automatically learn defect features. They applied directed graph-based CNN (DGCNNs) to learn semantic features. The key step for the model is generating Control Flow Graphs (CFGs) by assembly code using g++ on Linux to obtain the graph representation of a program. CFG is constructed to describe the execution flows of the assembly instructions which disclose the behavior of the program. The other step is applying a graphical model on CFG datasets that contain the multi-view multi-layer CNN for directed labeled graphs to give models based on CFG data.

These steps start with vector representations that represent a vertex as a set of real-valued vectors corresponding with the number of views. Next, gather extracted features from all the parts of the graphs into a vector after applying two convolution layers followed by a dynamic pooling layer. At the final step, a feature vector is forward to a fully-connected layer and an output layer to compute the categorical distributions for possible outcomes as presented in Figure 4.

The experiments were conducted over four datasets (SUMTRIAN, FLOW016, MNMX, SUBINC) obtained from the CodeChef site. They employed several machine learning algorithms to build predictive models that are NN, SVM, and KNN. This approach presents the best performance in comparison with the feature-based approach (the improvement was from 4.08% to 15.49%) and tree-based approaches (the improvement was from 1.20% to 12.39%).

Reference [41] They combined the word embedding and Long Short- Term Memory (LSTM) algorithm for defect prediction. The proposed model contains three steps; the first step extracts a token from its abstract syntax tree. Second, the token is transformed into a vector. Third, use the vector and its labels to build a Long Short-Term Memory. The LSTM algorithms perform defect prediction by automatically learn the semantic information of programs. The experiments applied to eight open-source projects show that the proposed model outperforms state-of-the-art defect prediction approaches. Deep learning algorithms utilization in the prediction areas reveals promising results; some of the most recent published articles used CNN and MLP without concentrate on addressing the effectiveness of manipulating the main factors that might have direct influence of the performance of prediction [46], [47].

III. RESEARCH METHODOLOGY

In this section, we started to review the basic design steps, which required using MLPs and CNN for software faults prediction using the NASA datasets. The methodology steps,

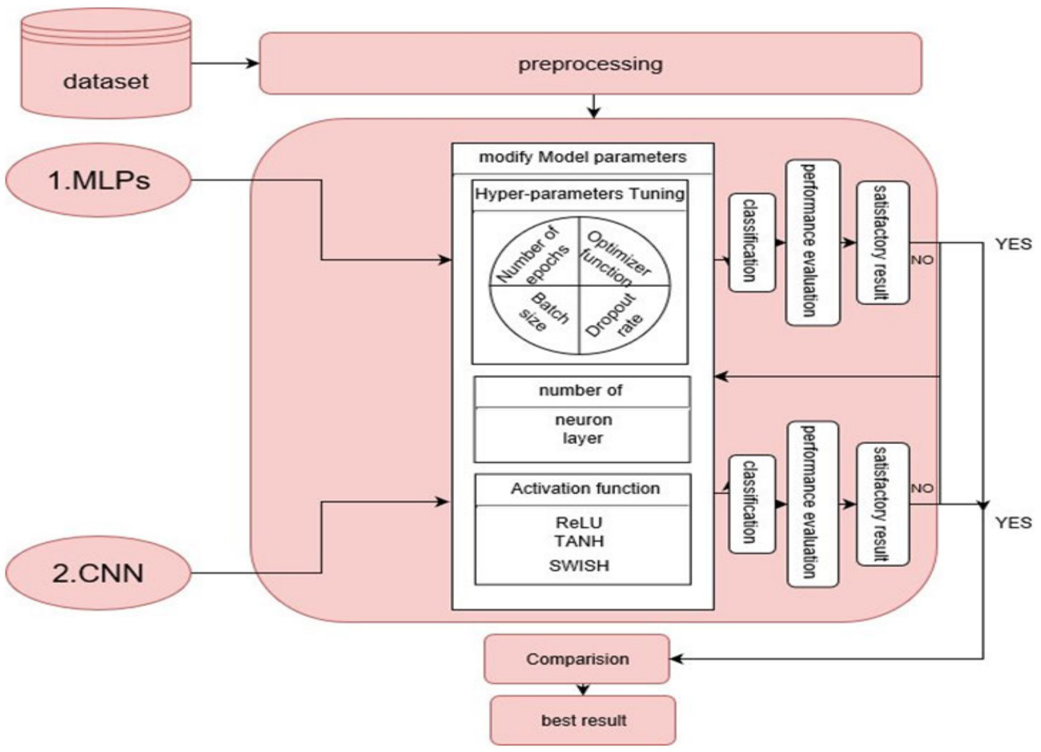


FIGURE 5. Research framework.

we started with selected four datasets with various fault percentage, then normalized for these datasets as preprocessing steps. The MLP is applied initially, and modify its parameters to measure the performance for it.

```
Algorithm    pseudo-code

Choose an initial (Number of layer, hyper parameters) randomly
Perform validation
Repeat
    Select the number of layer (1-N: N depends on experiments)
        Select Hyper parameter
        Perform training
        Perform validation ( if Detection Rate ≠ best Detection Rate)
        Replace hyper parameters
    Replace the number of layer
Until stopping criteria
```

FIGURE 6. Algorithm pseudo code.

Finally, we compared the results to find the best results achieved and applied the same steps for CNN. The MLP and CNN algorithms are implemented on python 3.6 language based on many libraries (such as Keras, Numpy, Panda and Sklearn, Matplotlib) to perform the experiments. Thereupon, each step is discussed in greater details in the subsequent subsections. An overview of our proposed methodology and the pseudo code are presented in in Figure 5 and 6 respectively.

A. SELECT DATASET

The datasets are selected from the NASA Metrics Data Program (MDP), include software measurement data and associated error data collected. NASA MDP dataset is made publicly available in order to encourage repeatable, verifiable, refutable, and/or improvable predictive models of software engineering.

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. NORMALIZATION

Normalization is used with numerical attributes that can find new ranges from an existing range based on an equation.

It is performed during the preprocessing step, useful for classification algorithms as NN, and distance measurements as (KNN, clustering). This study applied the standardization method such that the attributes preserve the normal distribution. Standardization is a useful technique to transform attributes with a Gaussian Distribution and differing means and distribution. In Python, we use a Standard Scaler from scikit-learn library. The following equation (1) represents the Standardization formula.

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

C. APPLYING DEEP LEARNING ALGORITHMS

In this phase, we applied two algorithms (MLPs and CNN) to address their abilities in enhancing the accuracy of SFP and determined the factor affecting the performance. In this step, we will select the various numbers of layers, different functions and different hyperparameters on each experiment.

Then we repeat the process until we get a satisfactory result.

D. MODIFY MODEL PARAMETERS

The settings which have to be defined for the network include hyper-parameter activation function, number of layers in each layer and hyper-parameter.

1) HYPER PARAMETERS

Different hyper-parameters will be altered during each test case, to examine the impact of it on accuracy. Tuning the hyper-parameter is important to choose the correct settings to obtain desired results. There are no precise rules to choose exactly the right parameters, in general, the choice relies on the type and size of the training dataset. Choosing correct settings is essential but consider as a complex part of network training. However, the tuning for hyper-parameter is frequently dependent upon experience rather than theoretical knowledge. Trade-offs are intrinsic in the parameter selection due to restrictions such as memory limitation [22].

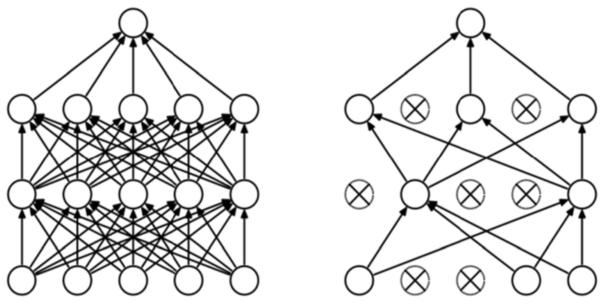
In our research, we conducted many experiments before recording the results, for each algorithm the values of hyper-parameters have been tested and evaluated its results. When reaching to best number for one of these hyperparameters then start to manipulate other hyperparameters to reach the best result. After that, compare the values that improve the performance of algorithms.

Also, drop out the values that did not have a positive effect on the results of the algorithms. After that, compare the values that improve the performance of algorithms.

Also, drop out the values that did not have a positive effect on the results of the algorithms.

The hyper-parameters settings are listed below: Number of epochs: an epoch is the number of passes through the data set [22]. In our experiment, the number of epochs will change, start from 10 up to 20000 epochs, to determine the best number of epochs for the network to converge properly. Batch size: is the number of training samples. Batch size is used to control many predictions that must be made at a time, and fit the model. In general, the larger batch size required more memory space. Reference [23] On the other hand, small-batch sizes are preferable since they head to produce convergence in a small number of epochs [24]. Different batch sizes will test, start from 1 up to 20. Dropout rate: Dropout is a technique used to prevent over-fitting and provide a way of approximately combining exponentially many different neural network architectures efficiently. The process of dropping units is done randomly [31]. In a neural network, dropping out units, it's temporarily removing it with all its incoming and outgoing connections as shown in Figure 7.

The dropout is used to feed forward neural networks, also it can use it applied to graphical models used such as Boltzmann Machines [32]. Different dropout rates will test from 0.2 to 0.6, using increments of 0.1. Optimizer function: is used to update weights in back propagation stage to reduce the error. We will use two Optimization functions, Adam (adaptive moments) and Adagrad (adaptive gradient) [5].



(a) Standard Neural Net (b) After applying dropout.

FIGURE 7. **Dropout [30].**

2) THE NUMBER OF LAYERS

There are many factors might influence greatly the performance of the networks such as a number of hidden layers, neurons in the hidden layer. The number of layers becomes the most important criterion in the architecture of the networks [25].

3) ACTIVATION FUNCTION

The activation function of an artificial neuron defines the output of that neuron based on the input or set of inputs. Each activation function takes one x input and executes a specific mathematical calculation on it. The common activation functions that are used in Deep Neural Network are Sigmoid, Rectified Linear Unit and hyperbolic tangent. Sigmoid: frequently used in back propagation NN. The range is only over (0, 1). It is an appropriate extension of nonlinearity that limits previous uses in NN also presents an adequate degree of smoothness [26]. The Sigmoid function is defined by the formula (2):

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

Hyperbolic Tangent (Tanh): is defined as the ratio between the hyperbolic sine and the cosine functions [27]. The Tanh function defined by the formula (3):

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

Rectified linear units (ReLUs): used as an activation function for the hidden layers in a Deep Neural Network, it becomes widely used to train a much deeper network than Sigmoid or Tanh activation functions. ReLU provides faster and more efficient learning for Deep Neural Network(DNN) over complex, high-dimensional data. The most important advantage of ReLU is that it does not require an expensive computation, just a comparison, and multiplication. ReLU has an efficient backpropagation without exploding or vanishing gradient which makes it a particularly appropriate choice for DNN [28].

Swish: is a smooth, non-monotonous function, unbounded above, bounded below, it is results are similar or significantly outperforms ReLU on the deep neural network across a variety of challenging datasets. Swish defined as f (x) = x · Sigmoid(x) [29].

E. COMPARISON

A comparison will be conducted in order to determine the effectiveness of manipulating the studied parameters (hyperparameter, number of layers and neurons, activation function).

F. DATA ANALYSIS & INTERPRETATION

This study performs a comparison of deep learning algorithms in terms of classification accuracy. In addition to use, Detection rate and TNR, that is computed by considering the positive and negative prediction of objects. The performance measures are computed through the following equations.

Sensitivity/Detection rate: It is the proportion of positive cases that were correctly identified. Describe if a class has a fault, how often the fault will be positive. (True positive rate) Detection rate = TP/(TP+FN). Accuracy: is the closeness of a measured value to the true value, is the proportion of the total number of predictions that were correct. 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

While working on the implementation, two machines were used in accordance to the time needed for testing. The first one, a personal laptop, was used primarily for conducting small tests that take short time. The second machine, a virtual machine loaned from the Computer Centre, was used primarily for long testing purposes.

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 virtual machine was running under the Linux Operating System, Ubuntu 18.04.1 LTS version. It was primarily used

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	.923	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	.408	.931		8000	.819	.504	.899		17000	.905	191	.979
	10000	.935	.363	.977		25000	.849	.538	.894		15000	.814	.438	.908		20000	.872	.040	.964

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	.996	10000 and 5layer	5	.842	.489	.906	20000 and 3layer	5	.825	.476	.913	15000 and 5layer	5	.891	.148	.968
	10	.928	.142	.987		10	.846	.429	.922		10	.831	.447	.928		7	.899	.170	.975
	30	.928	.051	.994		15	.843	.388	.924		15	.819	.451	.911		10	.887	.212	.957

for testing without rendering any additional services in order to provide as independent testing as possible. The laptop Operating System was Ubuntu 18.04.1 LTS.

IV. RESULT

The implementation of the MLPs and CNN is done in Python 3.6.5 using Keras Frameworks. In addition, Numpy, Panda and Sklearn libraries were used. Visualization is done using the Matplotlib library and Spyder served as the development environment. This work has developed without separation between the modification of the network and the implementation of the network. We had tried more than two hundred experiments with different (hyper-parameter, activation function and a number of layers). Experiments cover the modification of the model parameters used to improve the results. The following sub-sections present the results obtained.

A. MLPS RESULT

To address the effect of (number of epoch, batch size, dropout rate, Optimizer, number of layers, activation function) the proposed approach achieved the following results by MLPs algorithms as described in tables 3 - 8.

NUMBER OF EPOCH In order to examine the effect of a number of the epoch, we performed the following experiment in MLPs as described in table 3.

BATCH SIZE In order to examine the effect of batch size, we performed the following experiments as described in table 4.

DROPOUT RATE In order to examine the effect of the dropout rate, we performed the following experiments as described in table 5.

OPTIMIZER In order to examine the effect of Optimizer, we performed the following experiments as described in table 6. adgrad gave better accuracy and Detection rate values.

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	.346	.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

NUMBER OF LAYERS In order to examine the effect of a number of layers, we performed the following experiments as described in table 7.

ACTIVATION FUNCTION In order to examine the effect of activation function, we performed the following experiments as described in table 8.

B. CNN RESULT

To address the effect of (number of epoch, batch size, number of layers, activation function) the proposed approach achieved the following results by MLPs algorithms as described in table 9, 10, 11.

NUMBER OF EPOCH In order to examine the effect of the number of the epoch, we performed the following experiment on CNN as described in table 9.

BATCH SIZE In order to examine the effect of batch size, we performed the following experiment on CNN as described in table 10.

NUMBER OF LAYERS In order to examine the effect of a number of layers, we performed the following experiment on CNN as described in table 11.

C. COMPARISON

COMPARISON IN TERMS OF PERFORMANCE Metrics Table 12 shows the best results we obtained from both algorithms. The results show a clear advantage for CNN.

COMPARISON IN TERM OF TIME AND NUMBER OF EXPERIMENTS Table 13 shows the difference in effort in terms of the number and time of experiments to obtain the satisfactory results of both algorithms.

As a summary of MLPs and CNN experiments, Tables 14 and 15 present improvements on results as parameters changed based on the detection rate.

V. DISCUSSION

To discuss findings and interpret the results, we evaluate the appropriate parameters of the CNN and MLPs algorithms which give us useful predictions.

A. MLPS

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

THE EFFECT OF THE HYPERPARAMETER The number of epoch had a significant effect, especially in increasing the Detection rate. When we increase epoch number then all of the following are increased: the Detection rate, the accuracy 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 7. 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	0.93	.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	.127	.982
	4	.925	.350	.968		4	.853	.407	.935		4	.822	.476	.909		4	.895	.170	.971
	5	.935	.363	.977		5	.846	.371	.933		5	.816	.495	.896		5	.905	.191	.979

TABLE 8. Activation function effect.

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	.905	.191	.979
	Tanh	.926	.194	.981		Tanh	.855	.199	.973		Tanh	.821	.439	.920		Tanh	.893	.0212	.984
	Swish	.920	.083	.988		Swish	.852	.165	.977		Swish	.808	.420	.918		Swish	.903	0	.997

the ratio of the TNR decreases and, there is a consequent decrease in the accuracy. For example, when applying to the KC1 dataset and increasing the number of the epoch from the optimal number (15000) to 25000, the percentage of TNR decreases from .931 to.894 and the accuracy of .854 to .849. The effect of the batch size is similar to that of the epoch. As the batch size increases, the accuracy increases, until it reaches the optimal size. After that, any increase in the batch size results in a decrease in accuracy. For example, when the batch is changed from 5 to 7 and then to 10, the accuracy and TNR change as follows: accuracy (.891- .899-.891), TNR (.968-.975-.957). The results showed that the best dropout rate was when using the value (.5) except PC1 where it was (.2). This may be due to a small percentage of faults that PC1 has. Also, the Agrad optimizer is better than Adam.

THE EFFECT OF THE NUMBER OF LAYERS The use of five layers for the PC1 and KC1 dataset achieved the best result, in terms of accuracy, Detection rate, TNR. On the other hand, KC1 and KC2 gave the best results when using three layers. In both cases with an increased number of layers, the Detection rate is increased while reducing the TNR of the data that reduces the accuracy. Through the results mentioned above, the number of layers is different according to the database itself. There may be a relationship between ratios of faults and the number of instances with the ideal number of the layers. This is what we will try to study and investigate in future research.

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	.983		3000	.987	1.00	.991		3000	.987	.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	.058	1.00
	10	.954	.347	1.00		10	.999	.992	1.00		10	.955	.987	.951		10	.973	.823	.992
	15	.969	.782	.983		15	1.00	1.00	1.00		15	.987	.968	.991		15	.932	.705	.962

THE EFFECT OF THE ACTIVATION FUNCTION by comparing the results in which different activation functions were used, the ReLU showed superior results over other activation functions.

B. CNN ALGORITHM

Based on our experiments and the results we obtained from these experiments, there was an effect created by modifying the network architecture. This effect was as follows:

THE EFFECT OF THE HYPERPARAMETER The increase in the number of epoch had a positive effect on all the datasets on which CNN was applied, whether it was accuracy or other means of measurements. 4000 epoch was the optimal number of epoch used for all experiments.

The effect of the batch size is similar to that of the epoch. As the batch size increases, the percentage of means of measurements increases, until the optimal size is reached. For CM1 the optimal batch size was 10 and was 15 for the rest of datasets.

THE EFFECT OF THE NUMBER OF LAYERS The use of five layers for the PC1 and KC1 dataset achieved the best results across the four statistical measures: accuracy, Detection rate, TNR. On the other hand, KC1 and KC2 gave the best results when using three layers. In both cases with an increased number of layers the Detection rate increases, while reducing the specificity of the data reduces the accuracy. Through the results mentioned above, the number of layers is different according to the database itself. There may be a relationship between ratios of faults has it and the number of instances with the ideal number of the layers. This is what we will try to study and investigate in future research.

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 15 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	.996		3	1.00	1.00	1.00		3	.993	.968	1.00		3	.973	.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	.831	.447	.928	.905	.191	.979
CNN	.978	.739	.996	1.00	1.00	1.00	.993	.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	12 00	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

C. THREAT TO VALIDITY

In this work, two deep learning algorithms are studied which are Multi-layer perceptron's (MLPs) and Convolutional Neural Network (CNN) to address the factors that might have an influence on the performance of the algorithms in SFP.

Although the experiment results show how manipulating the picked parameters are directly affecting the prediction performance, still there are threats to construct validity refer to the generalizability of our results.

The major Internal Validity Threats in our study were

The addressed parameter settings, as we selected set parameters to be manipulated in the proposed technique and then the comparisons have been conducted base on these parameter sittings.

TABLE 14. Effects of modification (MLPS).

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%	3.9%	5.6%	6.5%	4.8%	15.1%	6.4%	5.4%	19.1%	6.4%

TABLE 15. Effects of modification (MLPS).

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

The addressed code instrumentation: the source code that used in our experiment was built to favor the two algorithms that used in this study.

Finally, data set collection: we collect very common dataset from real-world experiments.

As external Validity threats, we might not use the most complex and big enough data set for evaluation purposes. Moreover, the addressed and implemented algorithms might be not enough to generalize the result as there are too many others algorithms in the literature.

Authors try to mitigate the internal and external threats in this work as follows:

1- There are others deep learning algorithms in the literature, therefore as future work, we plan to mitigate this threat further by addressing even more Deep learning algorithms and conduct a comprehensive comparisons. In spite of this threat, in this work we studied the most commonly used deep learning algorithms by past software engineering studies to evaluate the factors effectiveness in fault prediction [2], [18], [32], [35], [36], etc. Thus, we believe there is little threat to construct validity from this part side. 2- In this work, four factors' influences are studied. In the future, we plan to reduce this threat by addressing others factors that might enhance or even reduce the algorithms performance. 3- This study used a very common public dataset (the PROMISE dataset), these datasets provide a real software faults and other related features. However, we have carefully picked the information that is provided within the dataset and perform some preprocessing steps to retrieve the relevant data of their applications. We tried to minimize threats to validity by using standard performance measures for fault prediction (TNR, Accuracy and the error ratio). However, in the future we intend to investigate other public or even commercial data set that represents diverse software applications.

VI. CONCLUSION AND FUTURE WORK

Machine learning is widely used in the area of prediction, one of the most promising subset is deep learning, the researchers prove that how deep learning achieves tangible performance in terms of prediction in various fields as computer vision, natural language processing, bioinformatics and software engineering etc. In this article, authors aimed to concentrate on answering two main research questions, Does the manipulating algorithms parameter could lead to introduce any performance enhancement in terms of accuracy?, Which of the studied deep learning algorithms provide the best SFP performance? The main essence of this study is to investigate the factors that have a tangible effect on the performance of the studied deep learning algorithms in the field of the SFP.

Several experiments have been conducted and followed by analysis and comparisons where very common used data set is used in these experiments. Results from the experiments were evaluated using accuracy, Detection rate, TNR. The results obtained from the CNN algorithm presented outstanding results of up to 100% for the KC1. Moving to the modifying parameters affecting, each of the parameters has a positive effect as the number increases to the optimal number which gives the best results. The results showed that the increase in the number of layers has a positive effect on the optimal number of layers for each dataset. As for activation functions, the ReLU activation function showed the best performance.

As the summary for the experiments, the impact of enhancing parameters had exceptional effects, which reached good results, especially for detection rate measurement. In the future work, we intend to perform further experiments and utilize other data set to address if the data set plays an important role (domain) or it really depends on the algorithms parameters solely. One of the main limitations in this work is not investigating the effectiveness of all hyper parameters; therefore, we intend to address some other factors in the future work. This study aims to determine the best deep learning algorithms for SFP and to reach the best possible results. For future work, it would be worthwhile to investigate the relationship between the dataset and its fault ratio with the appropriate algorithm and its parameters. After determining the potential relationship, it is necessary to develop a tool that uses deep learning algorithms for SFP and, possibly, for other fields.

REFERENCES

[1] S. Parnerkar, A. V. Jain, and C. Birchha, "An approach to efficient 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 flow 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 prediction using artificial neural network and simplified swarm optimization," *IJARCCE*, vol. 6, no. 3, pp. 601–605, Mar. 2017.

[9] Y. LeCun and Y. 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 classification 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, "Artificial 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.

[17] H. Wang, T. M. Khoshgoftaar, J. Van Hulse, and K. Gao, "Metric selection for software defect prediction," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 21, no. 02, pp. 237–257, Mar. 2011.

[18] J. Li, P. He, J. Zhu, and M. R. Lyu, "Software defect prediction via convolutional neural network," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. (QRS)*, Jul. 2017, pp. 318–328.

[19] H. Khanh Dam, T. Pham, S. Wee Ng, T. Tran, J. Grundy, A. Ghose, T. Kim, and C.-J. Kim, "A deep tree-based model for software defect prediction," 2018, *arXiv:1802.00921*. [Online]. Available: http://arxiv.org/abs/1802.00921

[20] S. D. Chandra, "Software defect prediction based on classification rule mining," Dept. Comput. Sci. Eng., Nat. Inst. Technol. Rourkela, Rourkela, India, Tech. Rep., 2013.

[21] J. S. Shirabad and T. J. Menzies, "The PROMISE repository of software engineering databases," School Inf. Technol. Eng., Univ. Ottawa, Ottawa, ON, Canada, 2005. Accessed: 2018. [Online]. Available: http://promise.site.uottawa.ca/SERepository

[22] I. A. F. Snuverink, "Deep learning for pixelwise classification of hyperspectral images," M.S. thesis, Delft Univ. Technol., Delft, The Netherlands, 2017.

[23] P. M. Radiuk, "Impact of training set batch size on the performance of convolutional neural networks for diverse datasets," *Inf. Technol. Manage. Sci.*, vol. 20, no. 1, Jan. 2017.

[24] A. Devarakonda, M. Naumov, and M. Garland, "AdaBatch: Adaptive batch sizes for training deep neural networks," 2017, *arXiv:1712.02029*. [Online]. Available: https://arxiv.org/abs/1712.02029

[25] I. Shafi, J. Ahmad, S. I. Shah, and F. M. Kashif, "Impact of varying neurons and hidden layers in neural network architecture for a time frequency application," in *Proc. IEEE Int. Multitopic Conf.*, Dec. 2006, pp. 188–193.

[26] J. Han and C. Moraga, *The Influence of the Sigmoid Function Parameters on the Speed of Backpropagation Learning*. London, U.K.: Springer-Verlag, 1995.

[27] B. Karlik and A. Vehbi, "Performance analysis of various activation functions in generalized MLP architectures of neural networks," *Int. J. Artif. Intell. Expert Syst.*, vol. 1, pp. 111–122, Feb. 2011.

[28] F. Farhadi, "Learning activation functions in deep neural networks," M.S. thesis, University of Montreal, Montreal, QC, Canada, Canada, 2017.

[29] P. Ramachandran, B. Zoph, and Q. V. Le, "Swish: A self-gated activation function," vol. 7, Oct. 2017, *arXiv:1710.05941*. [Online]. Available: https://arxiv.org/abs/1710.05941

[30] N. Srivastava, "Improving neural networks with dropout," Univ. Toronto, Toronto, ON, Canada, Tech. Rep. 566, 2013, vol. 7.

[31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[32] D. Gupta. (2018). *Architecture of Convolutional Neural Networks (CNNs) Demystified*. Analytics Vidhya. [Online]. Available: https://www.analyticsvidhya.com/blog/2017/06/architecture-ofconvolutional-neural-networks-simplified-demystified/

[33] P. S. Soniya and L. Singh, "A review on advances in deep learning," in *Proc. IEEE Workshop Comput. Intell., Appl. Future Directions (WCI)*, Kanpur, India, Dec. 2015, pp. 14–17.

[34] S. Peng, H. Jiang, H. Wang, H. Alwageed, and Y.-D. Yao, "Modulation classification using convolutional neural network based deep learning model," in *Proc. 26th Wireless Opt. Commun. Conf. (WOCC)*, Newark, NJ, USA, Apr. 2017, pp. 7–8.

[35] D. Ghosh and J. Singh, "A novel approach of software fault prediction using deep learning technique," in *Automated Software Engineering: A Deep Learning-Based Approach*. Cham, Switzerland: Springer, 2020, pp. 73–91.

[36] X. Lige, S. Z. Hua, and S. Z. Feng, "Road machinery fault prediction based on big data and machine learning," in *Proc. 5th Int. Conf. Control, Autom. Robot. (ICCAR)*, Apr. 2019, pp. 536–540.

[37] K. Goseva-Popstojanova, M. Ahmad, and Y. Alshehri, "Software fault proneness prediction with group lasso regression: On factors that affect classification performance," in *Proc. IEEE 43rd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, vol. 2, Jul. 2019, pp. 336–343.

[38] C. Jones and O. Bonsignour, *The Economics of Software Quality*. London, U.K.: Pearson, 2012.
[39] E. Borandag, A. Ozcift, D. Kilinc, and F. Yucalar, "Majority vote feature selection algorithm in software fault prediction," *Comput. Sci. Inf. Syst.*, vol. 16, no. 2, pp. 515–539, 2019.
[40] P. Lingden, A. Alsadoon, P. W. C. Prasad, O. H. Alsadoon, R. S. Ali, and V. T. Q. Nguyen, "A novel modified undersampling (MUS) technique for software defect prediction," *Comput. Intell.*, vol. 35, no. 4, pp. 1003–1020, Nov. 2019.
[41] H. Liang, Y. Yu, L. Jiang, and Z. Xie, "Seml: A semantic LSTM model for software defect prediction," *IEEE Access*, vol. 7, pp. 83812–83824, 2019.
[42] C. Jin, "Software reliability prediction based on support vector regression using a hybrid genetic algorithm and simulated annealing algorithm," *IET Softw.*, vol. 5, no. 4, p. 398, 2011.
[43] S. Kanmani, V. R. Uthariaraj, V. Sankaranarayanan, and P. Thambidurai, "Object-oriented software fault prediction using neural networks," *Inf. Softw. Technol.*, vol. 49, no. 5, pp. 483–492, May 2007.
[44] S. Rathore and S. A. Kumar "A decision tree regression based approach for the number of software faults prediction," *ACM SIGSOFT Softw. Eng.*, vol. 41, no. 1, pp. 1–6, 2016.
[45] R. Mu and X. Zeng, "A review of deep learning research," *TIIS* vol. 13, no. 4, pp. 1738–1764, 2019.
[46] M. Akour, S. H. Al, and O. Al Qasem, "The effectiveness of using deep learning algorithms in predicting students achievements," *Indonesian J. Elect. Eng. Comput. Sci.*, vol. 19, no. 1, pp. 387–393, 2020.
[47] M. Akour, O. Al Qasem, and H. Al Sghaier, "The effectiveness of using deep learning algorithms in predicting daily activities," *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 8, no. 5, pp. 2231–2235, 2019.

OSAMA AL QASEM received the master's degree in computer information systems from the Faculty of Information Technology and Computer Sciences, Yarmouk University. He is currently working on building Strong Research in the area of big data. He has publications in the fields of software engineering, big data analytics, and software fault prediction.

MOHAMMED AKOUR received the bachelor's and master's degrees (Hons.) in computer information systems from Yarmouk University, in 2006 and 2008, respectively, and the Ph.D. degree (Hons.) in software engineering from North Dakota State University (NDSU).

He joined Yarmouk University as a Lecturer, in August 2008. In August 2009, he left Yarmouk University to pursue the Ph.D. degree. He joined Yarmouk University, in April 2013, where he was the Head of Accreditation and Quality Assurance, the Director of Computer and Information Center, and the Vice Dean of Student Affairs. He is currently an Associate Professor of software engineering with Al Yamamah University (YU), Riyadh, Saudi Arabia. He is a member of the International Association of Engineers (IAENG). He serves as a keynote speaker, an organizer, the co-chair, and the publicity chair for the several IEEE conferences and ERB for more than ten ISI indexed prestigious journals.

MAMDOUH ALENEZI received the M.S. degree from DePaul University, in 2011, and the Ph.D. degree from North Dakota State University, in 2014. He is currently the Dean of Educational Services with Prince Sultan University. He has extensive experience in data mining and machine learning where he applied several data mining techniques to solve several software engineering problems. He has conducted several research areas and development of predictive models using machine learning to predict fault-prone classes, comprehend source code, and predict the appropriate developer to be assigned to a new bug.