Software Defect Prediction using Deep Learning

Meetesh Nevendra* and Pradeep Singh

Department of Computer Science and Engineering
National Institute of Technology, Raipur, India
e-mail: mnevendra.phd2018.cs@nitrr.ac.in and psingh.cs@nitrr.ac.in

Abstract: An increasing number of defects in software, damages the quality and reliability of that software. The detection of defective instances is becoming increasingly important, and current detection techniques require a great deal of improvement. However, Machine Learning (ML) techniques are effectively used, to detect defects in software. The primary purpose of ML techniques in Software Defect Prediction (SDP) is to predict defects, according to historical data. Establishing a critical SDP model on high-dimensional and limited data is still a challenging task. Thus, in this paper, we proposed an approach to detect defective modules in software using enhanced Convolutional Neural Networks (CNNs). The paper aims to identify the defective instance using the enhanced deep learning method. Our experiments are based on Within Project Defect Prediction (WPDP), where K-fold cross-validation is performed. The proposed approach has been evaluated on nineteen open-source software defect datasets, with respect to different evaluation metrics. Empirical results show that our proposed approach is significantly better than Li's CNN and standard ML model. In addition, we performed the Scott-Knot ESD test, which shows the effectiveness of our proposed approach.

Keywords: Software defect; CNN; Deep learning

1 Introduction

In the modern area, software quality is increasing rapidly, which directly affects the cost and reliability of the software product. However, the presence of defects in the software linearly decreases the quality and increases the software product's cost. SDP [1-3] is a technique that builds a classifier and predicts the code areas that potentially contain defects. The classifier's outcomes (i.e., defective programming regions) can locate indications for code reviewers to allocate their efforts. SDP is an essential part of software quality analysis [4] and is analyzed through software reliability engineering. In software engineering, early detection of defective parts of a software system can help developers and engineers in finding the correct way to use the limited resources in the testing and maintenance phases of software development.

Several research studies have been done for SDP to predict defective instances from historical data [1] [5]. SDP is more feasible because it can predict the defective instances and ensure developers from where defects arise. In recent work, Dam et al. proposed experimental research on SDP using a deep tree-based prediction model [6].

Nowadays, deep learning (DL) has played an essential role in the ML literature [7], and it has been used by different research areas and proven to be very useful, especially in speech recognition [8] and image processing [9]. Still, the use of DL for SDP is not thoroughly investigated. This study develops an enhanced deep learning model to investigate how the deep learning model will work with defect prediction datasets. We proposed an approach that utilizes enhanced CNN to predict the defective instances from a historical dataset. The proposed approach comprises two stages: model construction and prediction. In the model-construction phase, we first select the appropriate features using the feature selection (FS) technique, then these features are used to build the model using an enhanced CNN approach. Once the model is built, we predict the software defectiveness in the prediction phase.

To evaluate our approach, we used accuracy, precision, recall, and f1-score, one of the most widely used metrics. We conducted experiments on 19 open source software defect datasets. The experimental results suggest that the suggested approach performs better than Li's CNN and standard ML models. We also performed the Scott-Knot ESD (SK-ESD) test to indicate that our proposed approach has utility.

The remaining of our paper is summarized as follows. In Section 2, we discuss related works. Section 3 provides an overview and description of the CNN architecture. Section 4 the datasets and performance measures are given. Section 5 shows the overall outline of our approach. Section 6 describes our experiments and results. Finally, Section 7 offers Conclusions and future work.

2 Related Work

In order to more understand the SDP, we look back at the previous work done by the researchers in past years. Before going into SDP for details, we do an in-depth review of valuable methods that are indifferent to software metrics bases and well-known in this area of SDP. Chidamber et al. [10] explain that software metrics are suited for SDP. Basili et al. [11] examined and validated the object-oriented design metrics (OODM) and determines that either this OODM is valid or not for SDP.

The reliability of the software varies according to different coding styles and various other parameters. Reliability is also one of the critical issues in SDP. To solve the problem, additional knowledge needs to be gathered in the form of historical source code. However, this problem is addressed by Gyimothy et al. [12] and by a new

method based on object-oriented metrics analysis and determined that these metrics are highly suitable for improving the model's prediction performance. Singh and Verma [13] also performed the defect prediction, in the design phase. They explained that design metrics are beneficial for the software development life cycle (SDLC), and the prediction of defects should happen at the initial phase of SDLC for early warning and cost-effective software development.

Several prior approaches have been investigated for classifying the SDP. Byoung et al. [14] established a novel polynomial function-based neural network (pf-NN) model for SDP. The approach aggregates fuzzy C-means and genetic clustering techniques, facilitating the acquisition of nonlinear parameters for a procedure. However, in terms of classification, clustering techniques are not suitable for achieving enhanced prediction performance. Different ML algorithms have been used for SDP to overcome this problem, such as Naïve Bayes (NB) classifier, support vector machine (SVM), decision tree, neural network, and DL techniques. Elish et al. [15] established the SDP scheme using SVM. They evaluated the effectiveness of SVM against eight statistical and ML models in the context of four open-source NASA datasets. The outcomes demonstrate that SVM is more effective in finding defects compare to other models. Researchers also found that NB is suitable for classifying SDP. Shivaji et al. [16] employed the NB classification techniques for SDP using FS methods. They found that NB using FS improves the significant performance of defective f-measure by 21%. However, they also show that NB achieves a 12% improvement over SVM. Correspondingly, Dejaeger et al. [17] performs the SDP using 15 different Bayesian networks and other popular ML methods and found that augmented NB classifiers perform better than other classifiers in terms of the ROC curve.

Santosh et al. [18] also performed the SDP using decision tree classification and developed a recommendation system for SDP. They found that the tree classification technique is more suitable for finding the defects. Singh and Verma [19] utilized 16 open-source datasets to find the defects using a multi classifier approach, a combination of SVM, NB, and Random forest (RF). They found that this technique is more effective for performing the SDP. Singh et al. [20] utilized the fuzzy rule-based approach for finding the defect in software metrics. They find that a fuzzy rule-based classification technique can produce competitive or improved performance than C4.5, RF, and NB classifiers. They also determine that the proposed technique is a more comprehensive option than the other existing techniques for understanding several aspects that determine software defects.

Recently, Yang et al. [21] utilized a DL-based technique for SDP. They utilized a deep belief network to predict defects on six large open-source projects. The experimental results show that the proposed approach can achieve significant results than other approaches. Manjula and Florence [22] also developed a deep based hybrid approach for SDP using software metrics. They combine the genetic algorithm and deep neural network for classification. The outcomes showed that the proposed approach performs significantly better than other techniques. Li et al. [23]

and Phan et al. [24] utilized CNN to predict the software defect. Also, Zhao et al. [25] predict the software defect via cost-sensitive siamese parallel fully connected neural networks. The outcomes of these studies show that the DL technique plays an essential role in ML for SDP and can be utilized in several classification areas.

However, these techniques still go through many problems such as accuracy, computational time, and complexity with respect to defect prediction. Nevertheless, these problems can be further improved. Here we present an enhanced CNN approach that helps to reduce the overfitting problem and provide a significantly better classification rate to overcome these issues. The entire process of our proposed model is shown in Section 5.

3 Convolutional Neural Network Architecture

Our proposed approach enhances the CNN model to predict software defect instances in software defect datasets. Our CNN model has four convolution layers, two pooling layers, a flattening layer, and two dense layers. Li's CNN model and our enhanced CNN model are compared in Table 1.

Table 1
Li's CNN model compared to our enhanced CNN model

	Li’s CNN	Enhanced CNN
Convolutional Layers	One	Four
Embedding Layer	✓	×
Dense layer	One	Two
Pooling Layers	One	Two
Dropout	×	✓
Activation function	ReLU and sigmoid	ReLU and sigmoid
Training and optimizer	Mini-batch SGD and Adam	Adam and binary cross-entropy
Parameter initialization	×	✓

Our enhanced CNN model and Li's model have several changes, like convolutional layers, pooling layers, dense layers and dropout layers, activation function, optimizer and parameter initializer. We utilized one dropout between convolutional layers and one dropout between dense layers in our enhanced CNN architecture. These changes in the model will help to increase the performance and reduce the problem of overfitting.

3.1 Convolution Layer

CNN relies heavily on convolutional layers as a building block. In a convolutional layer, the goal is to extract features from the input data. Each layer has a set of

learnable filters as its parameter $w = w_1, w_2, \dots, w_n$ and biases = $b = b_1, b_2, \dots, b_n$. Layers apply convolution operations to generate a feature map X_n and pass the result on to subsequent layers. A nonlinear element-wise transform $\sigma(\cdot)$ is applied to these features, and the same process is repeated for each convolutional layer k.

$$X_n^k = \sigma(w_n^{k-1} * X^{k-1} + b_n^{k-1}) \quad (1)$$

3.2 Pooling Layer

Using a pooling layer, which reduces the resolution of the feature maps to achieve shift-invariance, is usually placed between two convolutional layers. Each feature map in a pooling layer is connected to its corresponding feature map in the convolutional layer preceding it in the pipeline. For each feature map $a^l_{:,:,k}$ we have $pool(\cdot)$ as the pooling function.

$$y^l_{i,j,k} = pool\left(a^l_{m,n,k}\right), \forall (m,n) \in \mathcal{R}_{i,j} \quad (2)$$

where $\mathcal{R}_{i,j}$ is a local neighbourhood around location (i,j). Average pooling and maximum pooling are the two most common pooling operations [26].

3.3 Flatten Layer

It converts the data into a 1-dimensional array so that it can be input into the next layer. We flatten the output of the convolutional layers to create a single long feature vector. A fully connected layer links it to the final classification model.

3.4 Dropout

Dropout was first introduced by Hinton et al. [27], and it has been shown to be very effective in reducing overfitting. As part of our enhanced CNN architecture, we apply dropout after the max-pooling and the fully connected layers, respectively. Dropout has the following output:

$$y = r * a(W^T x) \quad (3)$$

where $x = [x_1, x_2, \dots, x_n]^T$ is the input to a layer that is fully connected $W^T \in \mathcal{R}^{n*d}$ is a weight matrix and r is a binary vector of size d.

3.5 Dense Layer

The CNN has dense layers after convolution and pooling. It's important to note that each node in the dense layer is fully connected to every other node in previous layers. Dense layers integrate local information with category differentiation in the convolutional or pooling layer, which is the function of the layer. This equation can be represented as:

$$dl_1 = f(\sum_{p=1}^{N} \varpi_{1,p} * o_p + b) \quad (4)$$

A neuron's activation function is denoted by the f, where ϖ is a weight vector, o is the input vector, and b is the bias value.

For this paper, the dense layer's activation function was referred to as Rectified Linear Unit (ReLU). The ReLU is treated as a standard activation function in CNN, however. They have shown that they are faster to train than standard sigmoid units in the hidden layer and can sometimes help with discriminative performance [28]. In this case, the derivative of the ReLU activation function is given as:

$$f'(x) = \frac{\delta f(x)}{\delta x} = \begin{cases} 0 \; if \; x < 0 \\ 1 \; if \; x > 0 \end{cases} \quad (5)$$

Our CNN architecture utilized the sigmoid activation function (SAF) at the last layer (output layer). The SAF is used to find a probability between 0 to 1. Mainly SAF is utilized where the probability needs to be found as an output. However, the probability has occurred only between 0 to 1; thus, the SAF is the correct choice. Mathematically the sigmoid activation function is defined as:

$$S(x) = \frac{1}{1+e^{-x}} \quad (6)$$

4 Datasets and Performance Measure

4.1 Datasets

To estimate the prediction capabilities of our CNN model, we experimented on 19 open-source software defect datasets. These defect datasets are collected from the tera-PROMISE data repository [29]. The instance of every dataset corresponds to two parts: metrics and labels. Table 2 show the statistics of utilized datasets. Columns one and four represent the dataset ID, columns two and six represent the dataset name, columns three and six represent the line of code, and columns four and eight represent the number of instances and defects. However, Table 3 shown the features present in the dataset. Columns 1 and 3 represent the feature ID, and columns 2 and 4 represent the features name of all the datasets.

Table 2
Statistics of Datasets

D.ID	Dataset	LOC	Instance / Defects	D.ID	Dataset	LOC	Instance / Defects
D01	log4j-1.0	21,549	135 / 34	D11	synapse-1.1	42,302	222 / 60
D02	log4j-1.2	38,191	205 / 189	D12	synapse-1.2	53,500	256 / 86
D03	lucene-2.0	50,596	195 / 91	D13	velocity-1.4	51,713	196 / 147
D04	lucene-2.2	63,571	247 / 144	D14	velocity-1.6	57,012	229 / 78

D05	lucene-2.4	102,859	340 / 203	D15	xalan-2.4	225,088	723 / 110
D06	poi-1.5	55,428	237 / 141	D16	xalan-2.5	304,860	803 / 387
D07	poi-2.0	93,171	314 / 37	D17	xalan-2.6	411,737	885 / 411
D08	poi-2.5	119,731	385 / 248	D18	xerces-1.2	159,254	440 / 71
D09	poi-3.0	129,327	442 / 281	D19	xerces-1.3	167,095	453 / 69
D10	synapse-1.0	159,254	440 / 71	-	Total	2,306,238	7,147 / 2,858

Table 3
Features in the datasets

F.ID	Features name	F.ID	Features name
1	wmc	11	moa
2	dit	12	mfa
3	noc	13	cam
4	cbo	14	ic
5	rfc	15	cbm
6	lcom	16	amc
7	lcom3	17	ca
8	npm	18	ce
9	loc	19	max_cc
10	dam	20	avg_cc

4.2 Performance Measure

In order to compare the results, we evaluate measures such as:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{7}$$

The accuracy is the percentage of correct classifications in the total number of classifications. Where T and F stand for true and false, P and N stand for positive and negative samples, respectively.

$$Precision = \frac{TP}{TP+FP} \tag{8}$$

The precision is calculated by dividing the number of correct classifications by the number of incorrect.

$$Recall = \frac{TP}{TP+FN} \tag{9}$$

The recall measures the number of correct classifications minus the number of missed entries.

$$F1 - score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{10}$$

On the other hand, an F1-score is a derived effectiveness measurement that measures the harmonic mean of precision and recall.

5 Proposed Approach

The overall workflow of our proposed approach is shown in Figure 1 below. The proposed enhanced CNN model was applied for predicting the defects in software projects. There are two phases to this approach: model construction and prediction. For model construction, the dataset is divided into k-folds, where k-1 folds are used to train the CNN model, and one fold is used to test the model.

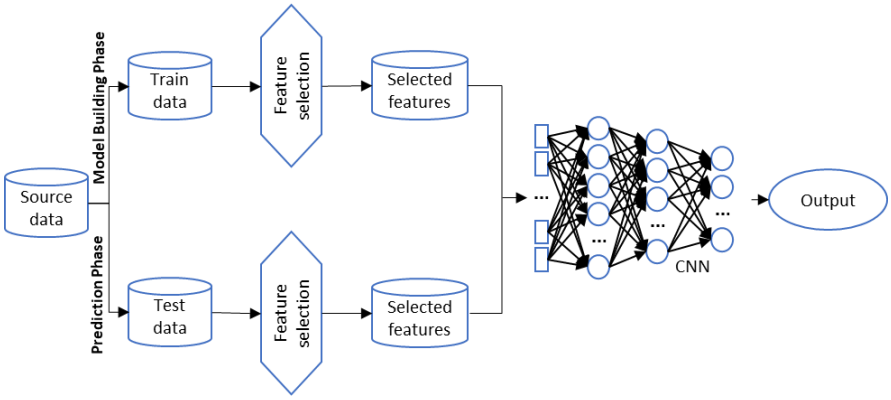


Figure 1
The overall workflow of our proposed approach

In order to train the CNN model, the FS technique is applied to the selected k-1 parts of data because the CNN model's input size should be expressed in metric units of $n*n$. The $n*n$ metrics cannot accommodate m number of features in our dataset. So, either increase the number of features in the dataset or remove those that aren't needed. This is why we removed unnecessary features to convert our dataset into $n*n$ format in order to avoid performance degradation. We used Chi-square (Chi2) filter-based FS to select N number of features in our experiment. It has been demonstrated by Nam et al. [30] that Chi-square is the most effective FS of all approaches. When features and classes are linked together, Chi2 performs well. When selecting relevant features, it helps to identify the frequency between class and feature. In our execution, we selected only those features that had the highest Chi2 scores. Calculation of the Chi2 score is denoted by:

$$Chi - square\ (\chi^2) = \frac{(observed\ frequency - Expected\ frequency)^2}{Expected\ frequency} \quad (11)$$

The number of examined classes is defined as the observed frequency. However, if there were no association between feature and target, the predicted class number would represent the expected frequency of that feature. Chi2 FS ranked the features based on their relationship to the class as long as they are ranked in order. To create n*n features metrics, we removed the lower-ranked feature metric from the features metrics. $n * n$ 2D metrics are created after N features are chosen from the source data.

Algorithm 1 Proposed Approach

Input: $Dataset\ \mathcal{D} = \{x_t, y_t\}_{t=1}^{n}$

$Candidate\ CNN\ algorithm$

Output: $Predict\ the\ performance\ score$

Initialization: $Select\ k = 10\ for\ k - fold\ cross - validation$

$N = 10\ for\ number\ of\ runs$

Procedure: $for\ i = 1\ to\ N$

$for\ each\ fold\ k\ do$

$Train, Test = \mathcal{D}(test\ size = 0.9, train\ size = 0.1)$

$Select\ m\ top\ features\ from\ Train\ data$

$train_{new} = \chi^2 = \sum_{r=1}^{p}\sum_{s=1}^{q}\frac{(D_{rs}-E_{rs})^2}{E_{rs}}$

$Model = CNN(train_{new})$

$test_{new} = Select\ same\ features\ from\ Test\ as\ train_{new}$

$Predict_{test} = Model.predict(test_{new})$

$Performance\ score\ [k] = Performance(Predict_{test})$

$end\ for$

$Performance\ score\ [i] = Performance[k].average$,

$end\ for$

$Final_{Performance} = Performance[i].average$

Further, the two-dimensional metric is transformed into 3D feature maps. Then the 3D metrics are then used to train a CNN model based on these metrics. In the prediction phase, the generated model is used to predict the defects (defective or non-defective) from the remaining one part of the data; before the prediction phase, the remaining one part of the data is selected as the same as features like the k-1 part of the data. The experiments are conducted ten times, and the average of all runs is selected as an outcome. The final outcomes are compared with Li's CNN and benchmarking ML approaches, SVM, AdaBoost, and KNN. Wu et al. [31] also added these three algorithms as the top 10 algorithms in data mining. Algorithm 1 shows the implementation of our proposed approach.

The SVM [32] is one of the effective and accurate supervised ML approaches. However, SVM has to find the best classification function to distinguish between members of two classes (0 and 1) in the training dataset.

The AdaBoost algorithm [33] was recommended by Freund and Schapire in 1977 and found one of the essential ensembles approaches since it has a substantial theoretical establishment, extremely accurate prediction, incredible simplicity, and extensive applications. The AdaBoost primary function is to generate a classifier by using the base learning algorithms repeatedly. The AdaBoost algorithm is used for both classification and regression problems.

K-nearest neighbor (KNN) [33] classifier has used a cluster of k substances in a training dataset neighboring the test entity. It bases the allocation of a label on the majority of a specific class in this neighborhood. The algorithm computes the

distance of the z object from the test object with training objects to establish its nearest neighbor. Once the nearest-neighbor list is achieved, the test object is classified based on its nearest neighbors' majority class and classifies their respective classes.

6 Results

For the experiments, we utilized a 10-fold cross-validation technique [34] to evaluate the execution of the proposed model in within-version scenarios. The proposed approach divides the datasets into ten equal parts, each with equal features and classes. We need to create the $n * n$ metrics from the given feature metrics, and it can't be possible to convert the $n * n$ metrics from the given 20 feature metrics. So to transform our data into $n * n$ metrics, we choose the top 16 features out of 20 features. In order to transform metrics into $4 * 4$ 2D feature metrics, we select the 16 most important features from each metric. As shown in Table 4, the selected feature ID can be found. A total of 16 features are selected for each dataset that is specified. According to their class, the features are chosen. First, the feature with the highest relative importance is chosen, followed by the second, and so on. The features that have been selected are displayed in decreasing order. It's crucial to select the maximum relevance features first and then minor relevance features at the end.

Table 4
Selected features ID for proposed approach

D01	D02	D03	D04	D05	D06	D07	D08	D09	D10	D11	D12	D13	D14	D15	D16	D17	D18	D19
11	6	11	6	11	6	11	11	11	11	11	11	11	11	11	6	11	11	11
6	18	6	11	6	11	6	6	6	5	6	5	18	6	6	11	6	6	6
5	11	5	5	5	5	18	5	5	19	5	6	6	5	5	5	18	5	5
4	7	18	1	4	1	5	1	1	18	8	4	5	18	18	1	5	18	18
7	4	1	9	1	9	1	9	9	6	19	18	19	8	1	9	1	1	8
1	5	19	4	9	18	8	17	18	8	1	8	1	1	9	19	9	13	1
9	9	8	8	7	8	4	8	4	1	18	19	17	9	19	18	4	8	4
19	1	4	7	8	3	19	7	8	7	4	1	16	4	4	17	19	17	13
8	17	9	17	18	4	9	19	19	4	9	7	7	19	7	4	7	4	19
18	3	3	18	17	17	13	16	17	9	13	9	3	3	8	7	8	9	17
20	8	13	13	3	13	17	3	13	3	7	17	4	13	17	3	13	3	7
13	12	7	10	19	19	7	13	7	20	12	13	2	17	13	13	3	7	16
17	16	17	3	13	16	3	2	16	17	14	16	8	15	20	8	17	19	9
16	19	12	19	16	2	20	14	20	16	20	12	20	10	3	20	10	2	12
12	14	16	12	12	14	16	10	12	13	2	20	14	16	16	16	20	14	3
3	20	8	15	10	7	14	4	10	15	10	10	12	2	15	2	14	20	10

After converting 4*4 2D metrics to 3D metrics, feed this 3D metrics data into our enhanced CNN model. Before the prediction phase, the test data feature metrics are converted into 3D metrics so that they are the same as the source data.

Every fold of the data is used ten times in the execution of the program. In every iteration, the model's training is performed with nine parts of the data; however, the

remaining part is used for testing. For the statistical reliability of the result, we run our experiment 10 times and record the average performance. However, we used accuracy, precision, recall and f1-score as a performance measure.

Table 5 shows the obtained result of our proposed approach vs different ML techniques, the best outcome of the average result in bold. We found that the proposed approach performs significantly better than different "state-of-the-art" ML approaches. The proposed model shows improvement with respect to all the performance evaluations. Proposed model overcome the KNN, SVM and AdaBoost with respect to accuracy by 3.68%, 5.98%, 2.48%, with respect to precision by 3.51%, 5.79%, 2.07%, with respect to recall by 2.95%, 4.8%, 3.46% and with respect to f1-score by 3.23%, 5.35% and 2.45% respectively. However, to identify the significance of our proposed model, we applied the SK-ESD test, which is implemented by Tantithamthavorn et al. [35]. It is also available on CRAN[1].

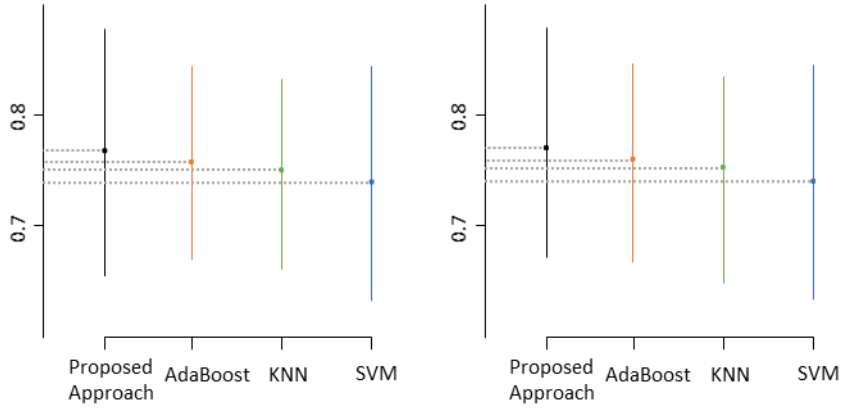
Table 5
Comparison of Proposed Approach

D06	D07	D08	D09	D10	D11	D12	D13	D14	D15	D16	D17	D18	D19	Average
0.696	0.875	0.761	0.751	0.872	0.716	0.73	0.729	0.654	0.842	0.671	0.714	0.806	0.823	0.747
0.726	0.853	0.75	0.762	0.815	0.723	0.735	0.751	0.674	0.842	0.643	0.697	0.847	0.836	0.755
0.749	0.825	0.782	0.78	0.815	0.719	0.734	0.792	0.687	0.875	0.663	0.7	0.853	0.875	0.767
0.737	0.839	0.766	0.771	0.815	0.721	0.734	0.771	0.68	0.858	0.653	0.698	0.85	0.855	0.761
0.674	0.882	0.706	0.742	0.897	0.734	0.672	0.759	0.676	0.846	0.596	0.618	0.834	0.852	0.73
0.705	0.854	0.732	0.759	0.892	0.732	0.641	0.753	0.685	0.856	0.589	0.658	0.824	0.846	0.738
0.721	0.861	0.744	0.791	0.871	0.724	0.674	0.753	0.679	0.84	0.611	0.695	0.854	0.883	0.753
0.713	0.857	0.738	0.775	0.881	0.728	0.657	0.753	0.682	0.848	0.6	0.676	0.839	0.864	0.745
0.717	0.841	0.802	0.778	0.853	0.765	0.718	0.841	0.703	0.817	0.651	0.733	0.779	0.834	0.756
0.771	0.824	0.871	0.832	0.831	0.739	0.602	0.858	0.775	0.799	0.628	0.674	0.753	0.822	0.766
0.754	0.864	0.837	0.833	0.833	0.711	0.609	0.87	0.738	0.829	0.643	0.734	0.755	0.852	0.769
0.762	0.844	0.854	0.832	0.832	0.725	0.605	0.864	0.756	0.814	0.635	0.703	0.754	0.837	0.767
0.696	0.883	0.744	0.898	0.897	0.729	0.749	0.694	0.847	0.848	0.586	0.839	0.847	0.842	**0.775**
0.729	0.871	0.753	0.902	0.883	0.734	0.765	0.704	0.834	0.821	0.561	0.862	0.876	0.834	**0.782**
0.732	0.886	0.76	0.91	0.886	0.732	0.754	0.71	0.859	0.829	0.584	0.856	0.888	0.856	**0.79**
0.73	0.878	0.756	0.906	0.884	0.733	0.759	0.707	0.846	0.825	0.572	0.859	0.882	0.845	**0.786**

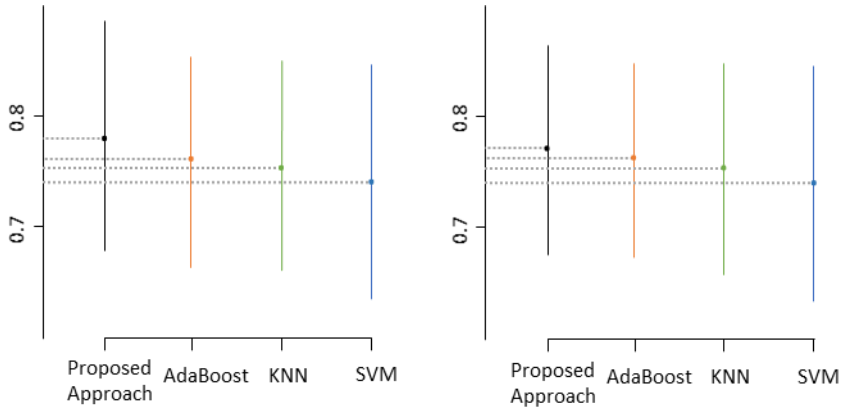
[1] https://cran.r-project.org/web/packages/ScottKnottESD/index.html

	DATA	D01	D02	D03	D04	D05
KNN	Acc	0.731	0.917	0.641	0.63	0.629
	pre	0.752	0.92	0.54	0.714	0.761
	rec	0.783	0.926	0.645	0.667	0.707
	F1	0.767	0.923	0.588	0.69	0.733
SVM	Acc	0.739	0.921	0.471	0.607	0.644
	pre	0.724	0.912	0.545	0.653	0.654
	rec	0.753	0.922	0.566	0.659	0.697
	F1	0.738	0.917	0.555	0.656	0.675
AdaBoost	Acc	0.717	0.907	0.625	0.619	0.664
	Pre	0.775	0.958	0.631	0.667	0.741
	rec	0.752	0.943	0.656	0.645	0.747
	F1	0.763	0.95	0.643	0.656	0.744
Proposed Approach	Acc	0.748	0.942	0.628	0.618	0.697
	Pre	0.756	0.952	0.638	0.658	0.721
	rec	0.778	0.962	0.646	0.648	0.732
	F1	0.767	0.957	0.642	0.653	0.726

The SK-ESD test is a mean comparison approach. It is an alternative approach of the Scott-Knott test [36] that finds the magnitude difference of each means within a group and between groups. The SK-ESD test finds the mean ranking. We apply the SK-ESD test in order to find the magnitude difference between the proposed approach and other presented ML approaches.



(a) Accuracy (b) Precision



(c) Recall (d) F1-score

Figure 2
Scott knot test result

Figure 2 shows the SK-ESD mean ranking comparison of the proposed approach. The result of the SK-ESD test lies between the proposed approach and present machine learning approaches. Figures 2(a), 2(b), 2(c) and 2(d) show the SK-ESD mean ranking comparison of accuracy, precision, recall and f1-score, respectively. In these figures, the horizontal grey dashed line indicates the mean value of all the presented methods, helping to visualize the mean differences between each method. We found that the proposed approach obtained the highest mean rank in all four scenarios from the figure. This shows that the proposed approach is performing better than the compared models for SDP. Table 6 also shows the obtained result of our proposed approach vs Li's CNN architecture. The best-performed result is in boldface. From Table 6, we can see that our proposed model is performed better than Li's CNN architecture. The proposed approach improves the performance over 15 datasets concerning all the applied performance evaluations. The proposed model overcomes Li's CNN model concerning the accuracy, precision, recall and f1-score by 15.12%, 16.32%, 16.3% and 16.52%, respectively.

Table 6
Performance comparison of enhancing CNN model with Li's CNN architecture

Data	Li's CNN				Enhanced CNN			
	Acc	Pre	Rec	F1	Acc	Pre	Rec	F1
D01	0.682	0.623	0.533	0.574	**0.748**	**0.756**	**0.778**	**0.767**
D02	0.922	0.902	0.910	0.906	**0.942**	**0.952**	**0.962**	**0.957**
D03	0.468	0.528	0.570	0.548	**0.628**	**0.638**	**0.646**	**0.642**
D04	0.583	0.573	0.568	0.570	**0.618**	**0.658**	**0.648**	**0.653**
D05	0.597	0.586	0.592	0.589	**0.697**	**0.721**	**0.732**	**0.726**
D06	0.596	0.573	0.563	0.568	**0.696**	**0.729**	**0.732**	**0.730**
D07	**0.883**	**0.872**	**0.888**	**0.880**	**0.883**	0.871	0.886	0.878

D08	0.644	0.653	0.672	0.662	**0.744**	**0.753**	**0.76**	**0.756**
D09	0.636	0.643	0.657	0.650	**0.898**	**0.902**	**0.91**	**0.906**
D10	**0.898**	**0.900**	**0.905**	**0.902**	0.897	0.883	0.886	0.884
D11	0.720	0.718	**0.738**	0.728	**0.729**	**0.734**	0.732	**0.733**
D12	0.462	0.462	0.467	0.464	**0.749**	**0.765**	**0.754**	**0.759**
D13	**0.728**	**0.792**	**0.870**	**0.829**	0.694	0.704	0.71	0.707
D14	0.371	0.345	0.381	0.362	**0.847**	**0.834**	**0.859**	**0.846**
D15	**0.849**	**0.856**	**0.832**	**0.844**	0.848	0.821	0.829	0.825
D16	0.491	0.486	0.489	0.487	**0.586**	**0.561**	**0.584**	**0.572**
D17	0.470	0.467	0.476	0.471	**0.839**	**0.862**	**0.856**	**0.859**
D18	0.839	0.829	0.812	0.820	**0.847**	**0.876**	**0.888**	**0.882**
D19	0.822	0.810	0.801	0.805	**0.842**	**0.834**	**0.856**	**0.845**
Average	0.666	0.664	0.670	0.666	**0.775**	**0.782**	**0.790**	**0.786**

In conclusion, we can say that the performance of enhanced CNN depends on the input data and model architecture. The data which are less valuable are eliminated to train the model. However, carefully increasing the architecture and parameters of the CNN model will lead to enhance the model. As a result, it enhances the training process and become a good prediction model. Compared to all other methods, the proposed CNN model performed significantly better.

Conclusions

The early detection and prediction of software defects plays an important role in modern software development, in terms of effective resource allocation. To address this issue of SDP, in this paper, we developed an enhanced CNN approach. First, in this approach, FS is applied to select $n * n$ 2D metric in the training dataset, and further, this metric is mapped into the 3D metrics. The CNN model is then trained using the generated metrics; it can predict the defective instances once the model is trained. The proposed enhanced CNN model significantly improves compared to existing benchmark classification schemes such as KNN, SVM, AdaBoost and Li's CNN model. We performed 10-fold cross-validation and calculated the average of all runs, resulting in the final result. Scott Knott ESD's mean ranking comparison of the proposed approach and other presented ML approaches shows that the proposed approach achieves the highest ranking.

Future research will focus on time reduction and accelerated network training. Also, exploration of other software metrics, aimed at the development of more efficient DL models, will be considered.

References

[1] Arar, Ö. F., Ayan, K.: Software defect prediction using cost-sensitive neural network. *Appl. Soft Comput.*, 33, 2015, pp. 263-277

[2] Chen, X. et al.: Software defect number prediction: Unsupervised vs supervised methods. *Inf. Softw. Technol.*, 106, 2019, pp. 161-181

[3] Nevendra, M., Singh, P.: *Multistage Preprocessing Approach for Software Defect Data Prediction*. In: Communications in Computer and Information Science. 2018, pp. 505-515

[4] Miholca, D. L. et al.: A novel approach for software defect prediction through hybridizing gradual relational association rules with artificial neural networks. *Inf. Sci. (Ny).*, 441, 2018, pp. 152-170

[5] Akmel, F. et al.: A Literature Review Study of Software Defect Prediction using Machine Learning Techniques. *Int. J. Emerg. Res. Manag. Technol.*, 6 (6), 2018, p. 300

[6] Dam, H. K. et al.: A deep tree-based model for software defect prediction. *arXiv Prepr. arXiv1802.00921*, 2018

[7] Jordan, M. I., Mitchell, T. M.: Machine learning: Trends, perspectives, and prospects. *Science (80-.).*, 349 (6245), 2015, pp. 255-260

[8] Graves, A. et al.: Speech recognition with deep recurrent neural networks. *ICASSP, IEEE Int. Conf. Acoust. Speech Signal Process. - Proc.*, (6), 2013, pp. 6645-6649

[9] Affonso, C. et al.: Deep learning for biological image classification. *Expert Syst. Appl.*, 85, 2017, pp. 114-122

[10] Chidamber, S. R., Kemerer, C. F.: A Metrics Suite for Object Oriented Design. *IEEE Trans. Softw. Eng.*, 20 (6), 1994, pp. 476-493

[11] Basili, V. R. et al.: A validation of object-oriented design metrics as qualityindicators. *IEEE Trans. Softw. Eng.*, 22 (10), 1996, pp. 751-761

[12] Gyimothy, T. et al.: Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction. *IEEE Trans. Softw. Eng.*, 31 (10), 2005, pp. 897-910

[13] Singh, P., Verma, S.: Cross Project Software Fault Prediction at Design Phase. *Int. J. Comput. Inf. Eng.*, 9 (3), 2015, pp. 800-805

[14] Park, B. J. et al.: The design of polynomial function-based neural network predictors for detection of software defects. *Inf. Sci. (Ny).*, 229, 2013, pp. 40-57

[15] Elish, K. O., Elish, M. O.: Predicting defect-prone software modules using support vector machines. *J. Syst. Softw.*, 81 (5), 2008, pp. 649-660

[16] Shivaji, S. et al.: Reducing features to improve code change-based bug prediction. *IEEE Trans. Softw. Eng.*, 39 (4), 2013, pp. 552-569

[17] Dejaeger, K. et al.: Toward comprehensible software fault prediction models using bayesian network classifiers. *IEEE Trans. Softw. Eng.*, 39 (2), 2013, pp. 237-257

[18] Rathore, S. S., Kumar, S.: A decision tree logic based recommendation

system to select software fault prediction techniques. *Computing*, 99 (3), 2016, pp. 1-31

[19] Singh, P., Verma, S.: Multi-classifier model for software fault prediction. *Int. Arab J. Inf. Technol.*, 15 (5), 2018, pp. 912-919

[20] Singh, P. et al.: Fuzzy Rule-Based Approach for Software Fault Prediction. *IEEE Trans. Syst. Man, Cybern. Syst.*, 47 (5), 2017, pp. 826-837

[21] Yang, X. et al.: *Deep Learning for Just-in-Time Defect Prediction*. In: 2015 IEEE International Conference on Software Quality, Reliability and Security. IEEE, 2015, pp. 17-26

[22] Manjula, C., Florence, L.: Deep neural network based hybrid approach for software defect prediction using software metrics. *Cluster Comput.*, 2018, pp. 1-17

[23] Li, J. et al.: *Software Defect Prediction via Convolutional Neural Network*. In: 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS) IEEE, 2017, pp. 318-328

[24] Viet Phan, A. et al.: *Convolutional Neural Networks over Control Flow Graphs for Software Defect Prediction*. In: 2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI) IEEE, 2017, pp. 45-52

[25] Zhao, L. et al.: Software defect prediction via cost-sensitive Siamese parallel fully-connected neural networks. *Neurocomputing*, 352, 2019, pp. 64-74

[26] Nagi, J. et al.: *Max-pooling convolutional neural networks for vision-based hand gesture recognition*. In: 2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA) IEEE, 2011, pp. 342-347

[27] Hinton, G. E. et al.: Improving neural networks by preventing co-adaptation of feature detectors. *arXiv Prepr. arXiv1207.0580*, 2012

[28] Dahl, G. E. et al.: *Improving deep neural networks for LVCSR using rectified linear units and dropout*. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE, 2013, pp. 8609-8613

[29] *tera-PROMISE: Welcome to one of the largest repositories of SE research data*. no date

[30] Nam, J., Kim, S.: *Heterogeneous defect prediction*. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015. New York, New York, USA: ACM Press, 2015, pp. 508-519

[31] Wu, X. et al.: Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, 14 (1), 2008, pp. 1-37

[32] Vapnik, V.: *The nature of statistical learning theory*. Springer science & business media, 2013

[33] Fix, E.: *Discriminatory analysis: nonparametric discrimination, consistency properties*. USAF School of Aviation Medicine, 1951

[34] Kohavi, R.: A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. *Int. Jt. Conf. Artif. Intell.*, 14 (2), 1995, pp. 1137-1145

[35] Tantithamthavorn, C. et al.: The Impact of Automated Parameter Optimization on Defect Prediction Models. *IEEE Trans. Softw. Eng.*, 45 (7), 2019, pp. 683-711

[36] Jelihovschi, E. G. et al.: The ScottKnott clustering algorithm. *Univ. Estadual St. Cruz-UESC, Ilheus, Bahia, Bras.*, 2014