Software Reliability Prediction by using Deep Learning Technique

Shivani Yadav, Balkishan
Department of Computer Science & Applications
Maharshi Dayanand University, Rohtak-124001, Haryana, India

***Abstract*—The importance of software systems and their impact on all sectors of society is undeniable. Furthermore, it is increasing every day as more services get digitized. This necessitates the need for evolution of development and quality processes to deliver reliable software. For reliable software, one of the important criteria is that it should be fault-free. Reliability models are designed to evaluate software reliability and predict faults. Software reliability prediction is always an area of interest in the field of software engineering. Prediction of software reliability can be done using numerous available models but with the inception of computational intelligence techniques, researchers are exploring new techniques such as machine learning, genetic algorithm, deep learning, etc. to develop better prediction models. In the current study, a software reliability prediction model is developed using a deep learning technique over twelve real datasets from different repositories. The results of the proposed model are analyzed and found quite encouraging. The results are also compared with previous studies based on various performance metrics.**

Keywords*—*Software reliability; deep learning; performance metrics; prediction; dense neural network; fault prediction

I. Introduction

Reliability is an essential and one of the most critical aspects of a software product and it is also one of the major attributes to determine software quality. Software reliability can be described as its ability to perform its intended functions accurately and successfully. Regular checks during software development ensure the prevention of faults which can further lead to failure and might incur huge efforts to correct or recover if detected later. Therefore, reliability prediction is an important aspect of any software development approach. For reliable software, it is important that it should be fault-free. Computational intelligence techniques like machine learning, genetic algorithm, deep learning, etc. are gaining the attention of researchers for reliability prediction. The current study uses a deep learning-based technique for software reliability prediction due to its potential to predict high accuracy on the huge amount of unstructured or unlabeled data [1]. Early fault prediction using deep learning models helps to improve the reliability of the software.

Deep learning is a subset of machine learning algorithms that are built on Artificial Neural Network (ANN). Neural networks are computational systems that respond to external inputs with dynamic state changes and try to determine underlying relationships within a dataset. ANN with two or three layers is a basic neural network and the neural network with more than three layers is considered as a deep learning concept [38]. The label deep was inspired by the number of processing layers that data must pass through. Deep learning advances have resulted in the development of neural networks with more complexity to generate more powerful learning abilities. The deep learning model takes an input and performs a step-by-step nonlinear transformation and then uses the learnings to generate a statistical model as output. The model continues these iterations until the output is accurate enough. Due to the data-hungry nature of deep learning algorithms and increased dataset size, complex problems can be easily solved more accurately and efficiently.

Deep learning integration into Software Engineering (SE) tasks has become increasingly popular among software developers and researchers these days. Deep learning assists SE experts in extracting requirements from natural language text, generating source code, and predicting software faults for typical SE tasks. Deep learning in SE has increased the interest of both the SE and Artificial Intelligence (AI) experts.

This paper aims to develop a novel neural network-based deep learning reliability prediction model. The choice of the deep learning model has been determined because of its ability to automatically capture and learn the discriminative features from data, which results in an improved reliability prediction model. This research will open the road for other deep learning approaches to be used in fault prediction. So, that software engineers will be able to better predict the likelihood of faults which results in greater resource use, risk management and better quality control.

The remaining paper is organized into five sections. Section 2 conducts a literature review of related studies to explore the various models already used for predicting the software reliability and its accuracy so, that the scope of further improvement can be identified. Section 3 discusses the proposed model design for improving the accuracy of software reliability prediction. Step by step process is also discussed in this section. Section 4 implements the model and presents the results. Results are presented in tabular as well as graphical form and also discussed in detail. Section 5 compares the result of the current study with previous studies. In the final section of the paper, the work has been summarized with possible directions for future research.

II. Literature Review

The use of Computational Intelligence (CI) in the field of software engineering is expanding nowadays. It can be witnessed by the huge research work undergoing and still being

carried out by various researchers. Some important research work related to software reliability prediction is filtered and studied to conduct the current work.

The term CI can be traced back to 1983 when Nick Cercone and Gordon McCalla started the International Journal of Computational Intelligence (IJCI). Cercone and McCalla sought to differentiate their work from existing studies in the broad Artificial Intelligence domain [2]. Bezdek[2] was the first to propose a technical definition of CI and its relation to neural networks like computational networks. Marks [3] summarized that fuzzy systems, genetic algorithms, neural networks, and evolutionary programming is building blocks of CI. In the same year,Karunanithi et al. [4] explored the application of connectionist models based on a neural network for software reliability growth prediction and claim better results as compared to traditional parametric models. In the same field, Ho et al. [5] extended the research, the work compared traditional and connectionist models while extensively studying software reliability prediction using connectionist models. Therefore, neural networks give good results in predicting errors but do not provide appropriate results under different circumstances. In 2005, Tian and Noore[6] mentioned that neural networks are difficult to interpret physically the neurons in layer and proposed an alternative approach based on genetic algorithm predict software reliability and Costa et al. [7] proposed a hybrid approach which used both genetic algorithms and evolutionary neural networks for improving the reliability prediction. In this approach, a genetic algorithm is used to analyze the number of neurons in each layer of ANN. The use of hybridization became prominent since 2005 in the field of predicting software reliability. Another study by Pai and Hong [8] experimented combination of Simulated Annealing (SA) and Support Vector Machines (SVMs) for predicting software reliability. In this study, SA is used to choose the SVM parameters. However, the authors suggest exploring other searching techniques for improving the results. Hu et al. [9] used recurrent neural networks (RNNs) and genetic algorithms for designing generic software reliability models and showed better results with the larger datasets. In 2011, Lo [10] introduced techniques Support Vector Machine (SVM) and Autoregressive Integrated Moving Average (ARIMA), both the proposed models predict better results as compared to the results of the traditional model. Li et al. [11] used the Adaboost technique based on machine learning which combines weak predictors into a single predictor to improve prediction accuracy and the results are verified using two case studies. Similarly, Roy et al.[12]a proposed neuro-genetic algorithm in which ANN is trained using backpropagation and further the weights of the network are optimized using Genetic Algorithm (GA). Further the results are compared with traditional methods and good results are obtained by the model. Then, researchers focused more on machine learning and deep learning methods. Jin et al. [13] proposed a combination of Quantum Particle Swarm Optimization (QPSO) and hybrid Artificial Neural Network (ANN) for predicting fault-proneness of software modules. QPSO was used for dimensionality reduction whereas ANN classified modules into non-faulty and faulty categories. The approach is simple to implement, and results showed the correlation between a module's software metrics and fault-proneness, which makes it possible to minimize cost and effort for software maintenance. Malhotra [14] reviewed various machine learning techniques for software fault prediction, performance is assessed and compared with statistical techniques. The study proved that machine learning technique models predict software fault better than traditional models, but these techniques are still limited. Wahono[15] proposed three influential frameworks i.e., Lessmann et al., Menzies et al., and Song et al. by combining Machine Learning (ML) classifiers for predicting software defects and improving the accuracy but these frameworks are not able to handle noisy data. Jaiswal and Malhotra [16] studied the application of various ML techniques including Instance-Based Learning (IBL), Cascading Forward Backpropagation Neural Network (CFBPNN), Multilayer Perceptron (MLP), General Regression Neural Network (GRNN), Feed Forward Backpropagation Neural Network (FFBPNN), Bagging, and Adaptive Neuro-Fuzzy Inference System (ANFIS) on industrial software. The results showed that ANFIS provides better reliability prediction compared to other methods. Several recent studies indicate the strength of the deep learning approach in software reliability prediction such as Clemente et al.[17] developed a predictive model using a deep learning technique that predicts security bugs with more accuracy (73.50%) as compared to machine learning techniques.[18][19][20][21][22] identified all the challenges, metrics required for finding faults and testing using different computational techniques.[23][24][25][26][27] fire reviewed, and assessed qualityparameters for component-based software using different computational intelligence techniques. Qasem et. al. [28] predicted software faults using two deep learning algorithms i.e., the Multi-layer Perceptrons (MLP) and Convolutional Neural Network (CNN) using four NASA datasets and concluded CNN is a better model but implemented on limited datasets.

The literature review shows that there are a lot of techniques being used by various researchers in predicting software reliability, but more work needs to be done for predicting reliability for complex or large datasets. However, the neural network-based deep learning approach is gaining the attention of researchers due to its capability of providing better results. However, still, there is a scope on improving the accuracy of the reliability prediction by detecting faults in the software. To further improve prediction accuracy, a deep learning model is designed which is presented in subsequent sections.

III. Design of Model

Deep learning algorithms are based on ANN where hidden layers try to uncover relationships between data. An artificial neural network works by processing inputs through several dynamic state responses. The interconnected processing elements between different layers are called neurons and are responsible for facilitating the computational system. Artificial neural networks have evolved to provide increasingly complex structures with powerful learning abilities.

The framework used for building this model is shown in Fig. 1.

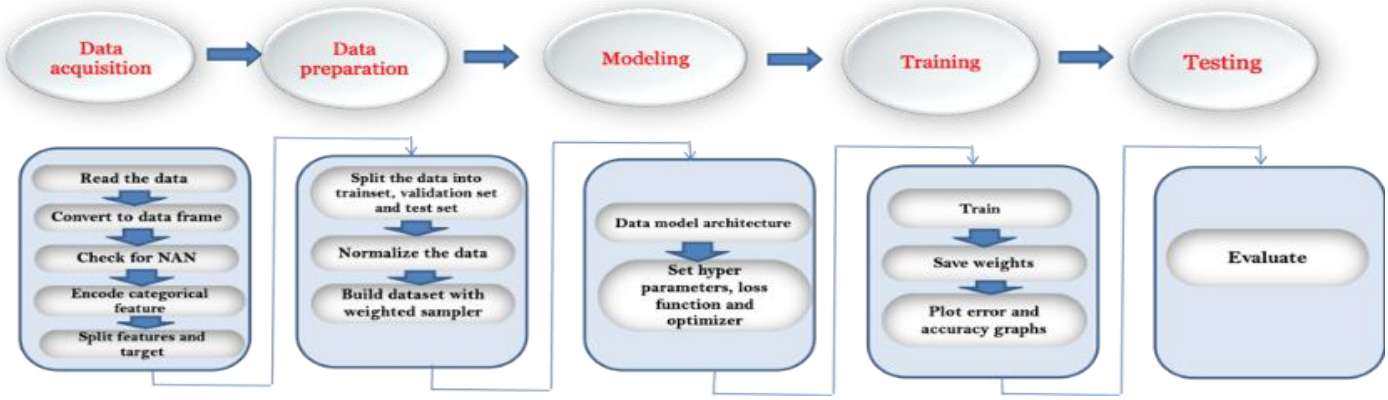

Fig. 1. Model Design Framework.

A. Data Acquisition

Data acquisition obtains meaningful data and transforms it into a digital form that can be processed by the model. The data used in the current study is obtained through various online sources and loaded into panda's data frames for further processing. A total of twelve datasets are used which consists of various features. Table VI presents the detail of all the datasets along with their sources. NAN (invalid or not a number) values check, and categorical features encoding are performed on the datasets. If the dataset contained NAN, it is ignored as they may create noise in further processing and lower the accuracy of the prediction. The attributes of the datasets are also divided into features and target attributes.

B. Data Preparation / Preprocessing

Collected data contains some impurities, therefore, not suitable for modeling in its raw form. It needs to be cleaned and pre-processed. For preprocessing the data transformation and normalization are carried out. This is accomplished by the application of natural logarithmic transformation and min-max normalization. Natural logarithmic transformation is used to reduce the skewness of the dataset distribution [38] and the min-max normalization technique provides high accuracy, learning speed and transforms the large value ranges into small range values. After normalization, a dataset is built using a weighted random sampler technique. The dataset is divided into sub-datasets for training, validation, and testing. This distribution is done randomly with 70% training data, 10%validation data, and 20%testing data. The purpose of training is to make the dataset applicable to train or fit for the model. Validation is used for unbiased evaluation at the time of hyperparameter tuning and the test set does unbiased evaluation of the final model.

The dataset contains many outliers which can affect the sample mean/variance and skew the results. To eliminate the noise due to outliers, considering the median and the interquartile range can yield better results. Therefore, Robust Scaling is applied to relevant features in the data set.

C. Modelling

The model is implemented using a dense neural network which consists of three types of layers: input, hidden, and output and shown in Fig. 2. In this type of network, all the neurons at one layer are connected with all the neurons of the previous layer. Various configurations of the model are designed for each dataset and later the configuration with the best results is finalized. Activation functions along with the layers are decided to design a network. Also, the initial values of hyperparameters are decided.

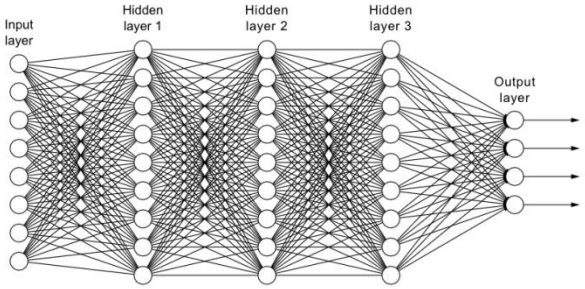

Fig. 2. Dense Neural Network Architecture.

For different configurations on each dataset different activation functions are used within the hidden layers in this study like ReLU (Rectified Linear Unit), GELU (Gaussian Error Linear Unit), Tanh (Hyperbolic Tangent), Softmax, and Sigmoid [29][30] and Table I represents all the activation functions with range.

- ReLU is a non-linear, differentiable, and computationally fast converge training phase of the network.
- A sigmoid activation function is non-linear, differentiable, and output ranges from 0 to 1 so that the output layer produces the result in probability for binary classification.
- Tanh is non-linear, differentiable, monotonic, and used for classification. The negative inputs are mapped strongly negative, and the zero inputs are mapped near zero.
- GELU is formed by combining properties of dropout, zoneout, and ReLu. It is a neuron activation function based on the Gaussian function.
- The softmax activation function normalizes the probability distribution of predicted target classes.

TABLE I. DIFFERENT ACTIVATION FUNCTION

Activation Function	Function f(x)	Range
ReLU	$\begin{cases} 0 \text{ if } x \le 0 \\ x \text{ if } x > 0 \end{cases}$	[0,∞)
Sigmoid, σ(y)	$\frac{1}{(1+e^{-y})}$	(0,1)
Tanh, tanh(y)	$\frac{e^{y} - e^{-y}}{e^{y} + e^{-y}}$	(-1,1)
GELU	$x \cdot \frac{1}{2}\left[1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right]$ where erf(z) is the error function	(-0.17,∞)
Softmax	$\frac{e^{z_i}}{\sum_j e^{z_j}}$ for i= 1,2,3,…,j	(0,1)

D. Training

In the training section, cross-entropy is used as a loss function. Cross-entropy calculates the difference between two probability distributions. SGD (Stochastic Gradient Descent) and Adam (Adaptive Moment Estimation) are used for optimization. They are used to update the weights after each iteration. The updated weights are saved so that further, loss and accuracy can be calculated. SGD is used as the optimization technique because of its ability to learn faster by randomly selecting a subset of data, generally called batch, and performing gradient descent iteratively on that subset. Adam optimization is an enhancement over SGD. It brings the best of AdaGrad and RMSProp, which are extensions of SGD to provide an adaptive learning rate with little memory requirements and computational efficiency.

Cross entropy (L_{CE}) is a loss function that is used during training to adjust model weights and find optimal weights. The aim is to minimize the loss, where a perfect model has zero loss. While zero loss is often difficult to achieve practically, models are optimized to minimize the loss to the extent possible.

$$L_{CE} = -\sum_{j=1}^{n} t_j \log p_j \quad (1)$$

for n classes, where t_j is the truth label and p_j is the Softmax probability for the j^{th}class [31].

Further, the cost-sensitive learning method is used to tackle the class imbalance problem by assigning different weights to both classes (faulty and non-faulty). The difference in weights influences the classification of the classes during the training phase. The whole purpose is to penalize the misclassification.

E. Testing

In this phase, the evaluation of the model is done statistically using four standard performance metrics accuracy, precision, recall, and F1-score. The percentage of correct predictions for test data is referred to as accuracy. The confusion matrix along with all these four-performance metrics calculates support value. The support is the actual number of occurrences of a response class in a dataset. Further, the accuracy of the model is evaluated using formulas:

$$\text{Accuracy} = (TP + TN) / (TP + FP + FN + TN) \quad (2)$$

where, TP= True positive, TN= True negative, FP= False positive, FN= False negative.

Precision is the number of positive class predictions that are actually positive class predictions. It is calculated as number of correctly predicted positive observations divided by total predicted positive observations [32].

$$\text{Precision} = TP/ (TP + FP) \quad (3)$$

A recall is defined as the number of correct positive predictions divided by all correct positive samples [32].

$$\text{Recall} = TP/ (TP + FN) \quad (4)$$

F1-score measures the accuracy of a model on a dataset and is calculated as the harmonic mean of the model's precision and recall [32],

$$F1 = 2*(\text{precision} * \text{recall})/ (\text{precision} + \text{recall}) \quad (5)$$

IV. IMPLEMENTATION AND RESULTS

The deep learning model is implemented on various datasets as shown in 0and determines its software reliability prediction ability. The objective of the model is to classify modules as faulty or non-faulty, based on different features of the dataset and all the datasets are shown in Table II.

TABLE II. DATASET

Dataset	Data with defects	Data with no defects	Target Feature
MJ	14299	79849	Bugs
PC5	5176	16670	Defective
JM1	2106	8779	Defects
MC1	68	9398	C
PC2	23	5566	C
KC1	326	1783	Problem
PC4	178	1280	C
PC1	77	1032	Defects
PC3	77	1032	Defects
KC2	107	415	Problems
Datatrieve	11	119	Faulty
COCOMO NASA	26	34	Rely

The MadeyskiJureczko (MJ) dataset presents metrics that are used to build software defect prediction models for component-based software. Different metrics included are 6Chidamber and Kemerer (CK) metrics, 1 Henderson-Sellers (HS) metric, 5 Bansiy and Davis (BD) metrics, 3 Tang and 2 Martin metrics. Other metrics are based on McCabe's complexity. The target attribute is named 'bugs'.

Datasets MC1, PC1, PC2, PC3, PC4, and PC5 are used for software defect prediction with 40, 21, 36, 22, 37, 39attributes respectively. Each dataset has 1 target attribute for predicting faults. The target attribute for MC1, PC1, PC2, PC3, PC4, PC5 is named as 'C', 'defects', 'C', 'defects', 'C' and 'defective' respectively.

JM1, KC1, and KC2 datasets are used to encourage repeatable, verifiable, refutable, and improve predictive models of software engineering. All datasets have 22 attributes

consisting of 5 different lines of code measure, 3 McCabe metrics, 4 base Halstead measures, 8 derived Halstead measures, a branch count, and 1 target field. The JM1 target field is named as'defects', KC1 target attribute is named as 'problem'and KC2 target attribute is classified as 'problems' which tells whether the module contains/does not contain reported defects in terms of 1 and 0.

Datatrieve dataset consists of a total of nine attributes including eight condition attributes and one target attribute. The target attribute is named 'faulty6_1' which has values of either 1 or 0. 0 indicates no faults are found and 1 indicates that faults are present. The purpose of the dataset is to study the correlation of code quality with the characteristics of the modules and the transition process between two versions of the software. The characteristics of the modules are recorded using attributes "LOC6_0", "LOC6_1", "AddedLOC", "DeletedLOC", "DifferentBlocks", "ModificationRate", "ModuleKnowledge", "ReusedLOC", "Faulty6_1".

The COCOMO NASA dataset attributes are used to find the required software reliability. The target feature is named 'Attribute Rely'. Values of various attributes are represented in the form of nominal, very high, high, low which are further converted into 0 and 1 during preprocessing. The seventeen attributes and its characteristics used are RELY (Required software reliability), DATA (Database size), CPLX (Process complexity), TIME (Time constraint), STOR (Main memory), VIRT, TURN (Turnaround time), ACAP (analysts), AEXP(Application), PCAP(Programmers), VEXP (Virtual machine), LEXP(Language), MODP (Modern Programming), TOOL (use of software), SCED (Schedule information), LOC (Line of code), ACT_EFFORT (Actual effort).

On all the twelve datasets, the same modeling approach is used with different configurations. In modeling, different layouts of neurons, and the values of hyperparameters (epoch, batch size, learning rate) have experimented and all the values are hyper tuned to achieve better results, the optimal combination of hyperparameters minimizes the loss function. Different dataset results in different values of parameters and different configurations for optimal results are shown in 0.

The loss and accuracy graph over the number of epochs for every dataset is shown in TABLE VFig. 4 to 27. A good prediction model should have low loss and high accuracy. As observed from the loss and accuracy graphs from all the datasets, the accuracy of the model over the iterations is higher than the loss respectively.

An accuracy metric is used to measure how accurate the developed model's prediction is as compared to actual data. The loss values are calculated on training data and verified using validation data. Loss values are observed after each iteration of optimization to find the optimal model parameters. The model's loss and accuracy data for each epoch are saved in the history which is used by the model's developer to make more informed decisions about the architectural choices that must be made. Optimal Configuration for the datasets is represented in Table III.

TABLE III. OPTIMAL CONFIGURATION

Dataset	Layers in Model	Learning Rate	Activation Function
MJ	[24,1024,112,1]	0.04	ReLu
PC5	[39,1024,812,512,2]	0.0094	Tanh
JM1	[21,1024,512,256,1]	0.0004	Softmax, GELU, ReLu
MC1	[40,1024,2]	0.009	Softmax
PC2	[35,1024,256,2]	0.001	ReLu
KC1	[21,1024,512,256,128,64,2]	0.0099	Tanh
PC4	[37, 1024,812,512,2]	0.0094	Tanh
PC1	[21,1024,2048,2]	0.0099	Tanh
PC3	[37,1024,512,2]	0.01	GELU
KC2	[21, 1024,256,1]	0.005	Sigmoid, Tanh
Datatrieve	[9, 256,512,64,1]	0.00001	Tanh, ReLu
COCOMO NASA	[17,512,128,1]	0.2	Tanh

The design model is tested using various performance metrics i.e., accuracy, precision, recall, and F1- score. These are the most commonly used reliable metrics for assessing the performance of a prediction model. The performance evaluation is done using a confusion matrix. The confusion matrix provides a summary of the individual class predictions for class-specific evaluations and provides information in terms of TP, TN, FP, and FN.

The results of the prediction model are shown in Table IV TABLE IVand Fig. 3.

TABLE IV. PERFORMANCE METRICS

Dataset	Accuracy	Precision	Recall	F1-score	Support
MJ	89%	0.90	0.96	0.93	55894
PC5	91%	0.99	0.90	0.95	11669
JM1	89%	0.92	0.95	0.93	5266
MC1	95%	0.99	0.95	0.97	7518
PC2	86%	0.99	0.86	0.93	3896
KC1	84%	0.90	0.91	0.91	1248
PC4	89%	0.99	0.87	0.93	895
PC1	85%	0.99	0.84	0.91	722
PC3	83%	0.99	0.81	0.89	1052
KC2	86%	0.89	0.94	0.92	311
Datatrieve	86%	0.97	0.87	0.92	83
COCOMO NASA	96%	0.99	0.91	0.95	23

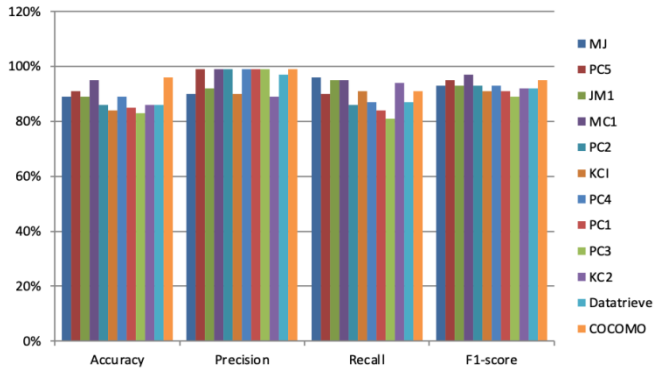

Fig. 3. Performance Metrics Graph.

From the results following are the observations that are made:

- Among all the datasets, the model showed the highest accuracy for the COCOMO NASA dataset i.e., 96% accuracy with precision 99%, recall 91% and f1-score 95% but it has the least instances among all the datasets. Datatrieve dataset also contains fewer instances and the model's accuracy on Datatrieve is 86% with good precision 97%, recall 87%, and f1-score 92%.
- MJ dataset has the highest number of instances among all the datasets and prediction accuracy on this dataset is 89% which is validating the model as a good model. The model showed a precision of 90%, recall of 96%, and f1-score of 93%. This shows that this model is working well on a large dataset.
- The prediction accuracy on MC1 and JM1 datasets is 95%, 89% respectively, though its instances are less than MJ. Results of precision 99%; 92%, recall 95%; 95% and f1-score 97%; 93% respectively are also very promising.
- The prediction accuracy on PC1, PC2, PC3, PC4, and PC5 datasets is more than 80%. The results are average as compared to previous work on these datasets. It concludes that this model is giving optimum results on these datasets.

V. Comparison with Existing Models

Our proposed deep learning-based reliability prediction model shows better results in terms of accuracy, precision, recall, and f1-measure as compared to other techniques like decision tree, linear regression, backpropagation neural network, SVM, random tree, random forest, naïve bayes, hybrid machine learning techniques, etc. For the dataset KC1 accuracy is second highest after VOTE [34] proposed by the author Wang et.al achieved the highest precision but better as compared to other models like Under Sampling Strategy (USS), Random Forest (RF), and Naïve Bayes (NB), whereas KC2 dataset achieved the highest accuracy, precision, recall and f1-measure when compared with other machine learning techniques. Datatrieve dataset achieved the highest accuracy and precision when compared with the previous model (USS) result given by author Rao et al. [35]. COCOMO NASA dataset is evaluated with the highest score among all the datasets, but the result cannot be reliable because the dataset is small. Except for accuracy, where it is second after Random Forest by Wang et.al[34], the MJ dataset, which is the largest component-based dataset, outperforms all other approaches like Linear Regression (LR), Decision Trees (DT), Naïve Bayes (NB), SVM, Stochastic Gradient Boosting, KNN in all performance metrics[33]. While JM1 dataset results top in all the metrics as compared to models USS [35], VOTE, RT, NB [33]. Dataset MC1, PC1, PC2, PC3, PC4, and PC5 achieve the highest results in precision, recall, and f1-score.Performance metrics of various models for different datasets are listed in Table VII.

TABLE V. Loss and Accuracy Graph for Various Datasets

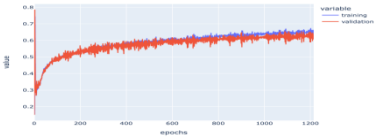

Fig. 4. Accuracy Graph for MJ.

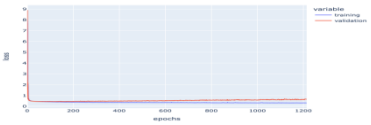

Fig. 5. Loss Graph for MJ.

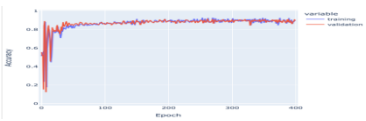

Fig. 6. Accuracy Graph for PC5.

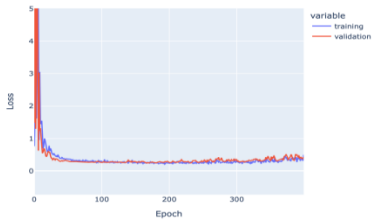

Fig. 7. Loss Graph for PC5.

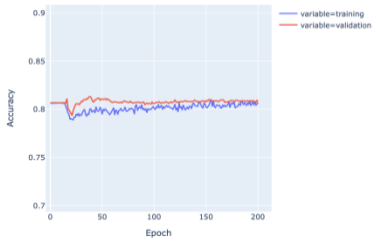

Fig. 8. Accuracy Graph for JM1.

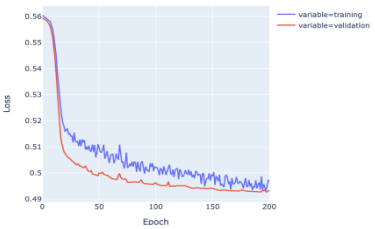

Fig. 9. Loss Graph for JM1.

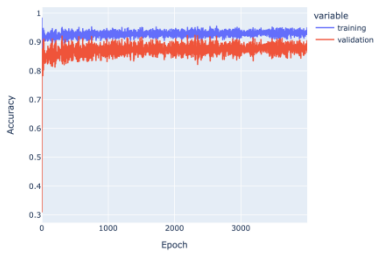

Fig. 10. Accuracy Graph for MC1.

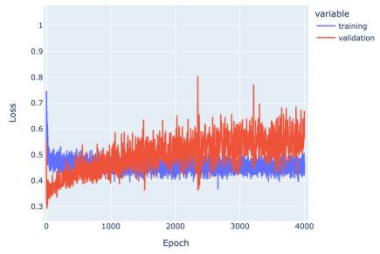

Fig. 11. Loss Graph for MC1.

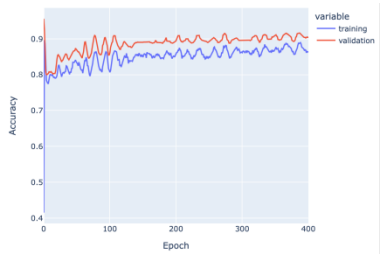

Fig. 12. Accuracy Graph for PC2.

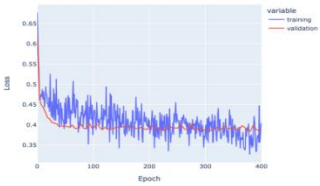

Fig. 13. Loss Graph for PC2.

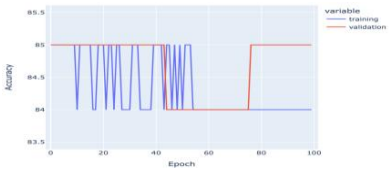

Fig. 14. Accuracy Graph for KC1.

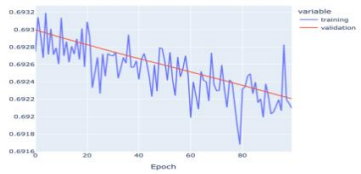

Fig. 15. Loss Graph for KC1.

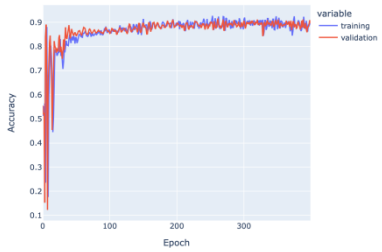

Fig. 16. Accuracy Graph for PC3.

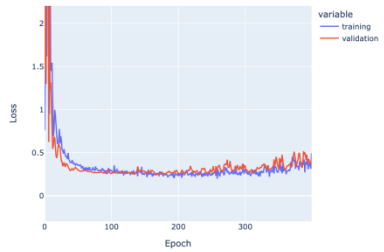

Fig. 17. Loss Graph for PC4.

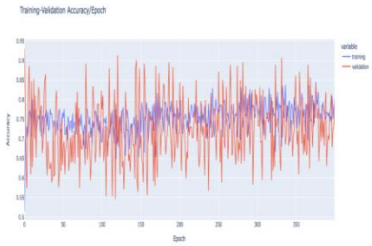

Fig. 18. Accuracy Graph for PC1.

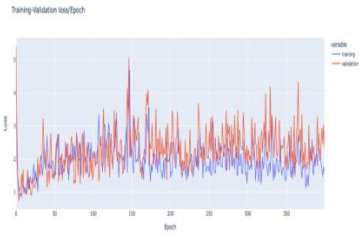

Fig. 19. Loss Graph for PC1.

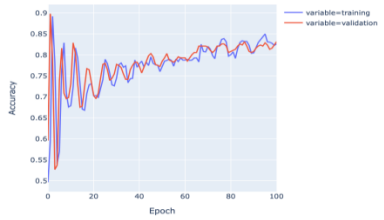

Fig. 20. Accuracy Graph for PC3.

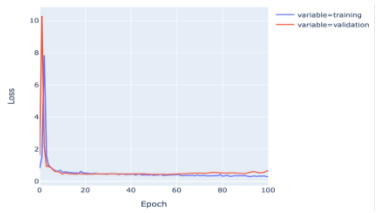

Fig. 21. Loss Graph for PC3.

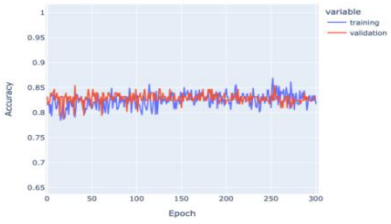

Fig. 22. Accuracy Graph for KC2.

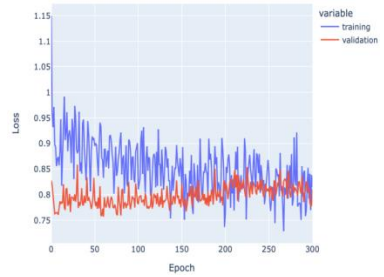

Fig. 23. Loss Graph for KC2.

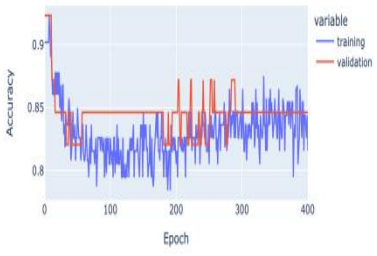

Fig. 24. Accuracy Graph for Transaction.

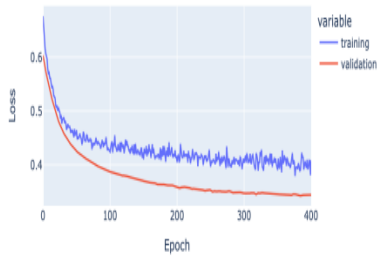

Fig. 25. Loss Graph for Transaction.

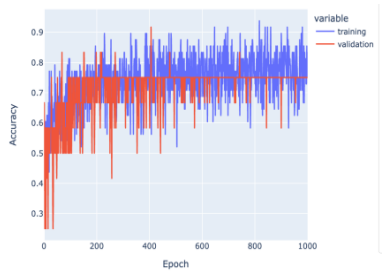

Fig. 26. Accuracy Graph for COCOMO NASA.

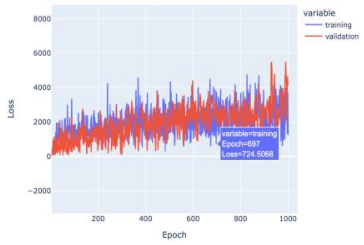

Fig. 27. Loss Graph for COCOMO NASA.

TABLE VI. DATASET DESCRIPTION

Dataset	Criterion	No. of Attributes	No. of instances	Source of Dataset
MJ	Software defect prediction	24	94148	https://madeyski.e-informatyka.pl/tools/software-defect-prediction/
PC5	Software defect prediction	39	17186	https://github.com/klainfo/NASADefectDataset/raw/master/OriginalData/MDP/PC5.arff
JM1	Software defect prediction	22	10885	http://promise.site.uottawa.ca/SERepository/datasets/jm1.arff
MC1	Software defect prediction	40	9466	https://www.openml.org/data/download/53939/mc1.arff
PC2	Software defect prediction	36	5589	https://www.openml.org/data/download/53952/pc2.arff
KC1	Software defect prediction	21	2109	http://promise.site.uottawa.ca/SERepository/datasets/kc1.arff
PC4	Software defect prediction	37	1458	https://www.openml.org/data/download/53932/pc4.arff
PC1	Software defect prediction	22	1109	http://promise.site.uottawa.ca/SERepository/datasets/pc1.arff
PC3	Software defect prediction	22	1109	https://www.openml.org/data/download/53933/pc3.arff
KC2	Software defect prediction	22	522	http://promise.site.uottawa.ca/SERepository/datasets/kc2.arff
Datatrieve	Success/ failure in the transaction	9	130	http://promise.site.uottawa.ca/SERepository/datasets/datatrieve.arff
COCOMO NASA	Required software reliability	17	60	http://promise.site.uottawa.ca/SERepository/datasets/cocomonasa_v1.arff

TABLE VII. COMPARISON OF ACCURACY WITH THE EXISTING MODELS IN THE LITERATURE

Dataset	Model	Accuracy	Precision	Recall	F - measure
MJ	**Proposed**	89%	90.00%	96%	93%
	Linear Regression (LR) [33]	74.99%		18.22%	
	Decision Tree (DT) [33]	74.45%		10.79%	
	Naive Bayes (NB) [33]	73.76%		22.28%	
	Support Vector Machine (SVM) [33]	78.19%		26.58%	
	Stochastic Gradient Boosting (GBM) [33]	76.16%		22.03%	
	K-Nearest Neighbor (KNN) [33]	84.24%		56.83%	
PC5	**Proposed**	91%	99%	90.00%	95%
	VOTE [34]	97.46%			
	Random Tree [34]	97.08%			
	Naive Bayes [34]	96.44%			

Dataset	Model	Accuracy	Precision	Recall	F - measure
JM1	**Proposed**	89%	92%	95%	93%
	USS[35]	66.40%	82.50%	96.90%	89.10%
	VOTE [34]	81.44%			
	Random Tree [34]	75.30%			
	Naive Bayes [34]	80.45%			
MC1	**Proposed**	95%	99%	95%	97%
	USS[35]	85.50%	67%	43.30%	49.70%
	VOTE [34]	99.42%			
	Random Tree [34]	99.43%			
	Naive Bayes [34]	93.80%			
PC2	**Proposed**	86%	99%	86%	93%
	VOTE [34]	99.53%			
	Random Tree [34]	99.29%			
	Naive Bayes [34]	97.11%			
KC1	**Proposed**	84%	90.00%	91%	91%
	USS[35]	78.50%	87.80%	95.30%	91.40%
	VOTE [34]	85.62%			
	Random Tree [34]	82.85%			
	Naive Bayes [34]	82.50%			
PC4	**Proposed**	89%	99%	87%	93%
	ILLE-SVM (Improved Locally Linear Embedding and Support)[37]	90.00%	62.50%	83.33%	71.43%
	VOTE [34]	90.28%			
	Random Tree [34]	87.74%			
	Naive Bayes [34]	87.11%			
PC1	**Proposed**	85%	99%	84%	91%
	USS[35]	84.10%	52.60%	36.30%	40.90%
	ILLE-SVM (Improved Locally Linear Embedding and Support)[37]	84.78%	78.26%	90.00%	83.68%
	LASSO-SVM[36]	78.26%	79.40%	75.46%	79.85%
	SVM[36]	71.32%	69.29%	69.25%	70.64%
	Linear regression (LR)[36]	84.20%	61.50%	69.60%	65.30%
	Back propagation neural network(BPNN)[36]	79.30%	60.60%	72.40%	66.90%
	Cluster Analysis (CA)[36]	71.60%	63.50%	71.20%	67.10%
	VOTE [34]	93.73%			
	Random Tree [34]	91.64%			
	Naive Bayes [34]	89.12%			
PC3	**Proposed**	83%	99%	81%	89%
	USS[35]	76.60%	37.60%	26.10%	30.10%
	ILLE-SVM (Improved Locally Linear Embedding and Support)[37]	89.66%	73.08%	86.36%	79.05%
	VOTE [34]	89.12%			
	Random Tree [34]	86.01%			
	Naive Bayes [34]	48.30%			
KC2	**Proposed**	86%	89%	94%	92%
	VOTE [34]	82.91%			
	Random Tree [34]	79.86%			
	Naive Bayes [34]	83.62%			
Datatrieve	**Proposed**	86%			
	USS[35]	50.00%	91.20%	99%	95.40%
COCOMO NASA	**Proposed**	96%	99%	91%	95%

VI. Conclusion

Predicting software reliability has become an essential activity in software development to develop better quality software. Recently, the researcher community has identified that computational intelligence techniques can outperform traditional prediction methods. This study predicts the software reliability using a dense neural network which is implemented using deep learning. The classification is performed on twelve datasets KC1, KC2, Datatrieve, COCOMO NASA, MJ, JM1, MC1, PC1, PC2, PC3, PC4, and PC5. The optimal model is designed with different configurations for each dataset for classification. Results are evaluated using four standard performance metrics, i.e., accuracy, precision, recall, and f1-score. The results obtained by our model show better results as compared to previous models in terms of accuracy, especially dataset MJ, JM1, KC2, and COCOMO NASA.

Hybridization of deep learning techniques with other computational intelligence techniques can be explored for better results. The same study can be extended with large industrial datasets to achieve better results and can also be experimented with other algorithms.

References

[1] C. Chen etal., "Reliability analysis using deep learning," International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, vol. 51739, pp. 1–10, Aug. 2018.

[2] J. C. Bezdek, "On the relationship between neural networks, pattern recognition and intelligence", International journal of approximate reasoning, vol. 6, pp. 85–107, 1992.

[3] Robert J. Marks II, "Intelligence: Computational versus artificial," IEEE Trans. Neural Networks, vol. 4, pp. 737–739, 1993.

[4] N. Karunanithi, D. Whitley and Y. K. Malaiya, "Prediction of software reliability using connectionist models," IEEE Transactions on software engineering, vol. 18, p. 563, 1992.

[5] S. L. Ho, M. Xie and T. N. Goh, "A study of the connectionist models for software reliability prediction," Computers & Mathematics with Applications, vol. 46, pp. 1037–1045, 2003.

[6] L. Tian and A. Noore, "Evolutionary neural network modeling for software cumulative failure time prediction," Reliability Engineering & system safety, vol. 87, pp. 45–51, 2005.

[7] E. O. Costa, G. A. de Souza, A. T. R. Pozo and S. R. Vergilio, "Exploring genetic programming and boosting techniques to model software reliability," IEEE Transactions on Reliability, vol. 56, pp. 422–434, 2007.

[8] P.F. Pai and W.C. Hong,, "Software reliability forecasting by support vector machines with simulated annealing algorithms," Journal of Systems and Software, vol. 79, pp. 747–755, 2006.

[9] Q. P. Hu, M. Xie, S. H. Ng and G. Levitin, "Robust recurrent neural network modeling for software fault detection and correction prediction," Reliability Engineering & System Safety, vol. 92, pp. 332–340, 2007.

[10] J. H. Lo, "A study of applying ARIMA and SVM model to software reliability prediction," International Conference on Uncertainty Reasoning and Knowledge Engineering, vol. 1, pp. 141–144, 2011.

[11] H. Li, M. Zeng, M. Lu, X. Hu and Z. Li, "Adaboosting-based dynamic weighted combination of software reliability growth models," Quality and Reliability Engineering International, vol. 28, pp. 67–84, 2012.

[12] P. Roy, G. S. Mahapatra and K. N. Dey, "Neuro-genetic approach on logistic model based software reliability prediction," Expert systems with Applications, vol. 42, pp. 4709–4718, 2015.

[13] C. Jin and S. W. Jin, "Prediction approach of software fault-proneness based on hybrid artificial neural network and quantum particle swarm optimization," Applied Soft Computing, vol. 35, pp. 717–725, 2015.

[14] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," Applied Soft Computing, vol. 27, pp. 504–518, 2015.

[15] R. S. Wahono, "A Systematic Literature Review of Software Defect Prediction," Journal of Software Engineering, vol. 1, pp. 1–16, 2015.

[16] A. Jaiswal and R. Malhotra, "Software reliability prediction using machine learning techniques," International Journal of System Assurance Engineering and Management, vol. 9, pp. 230–244, 2018.

[17] C. J. Clemente, F. Jaafar and Y. Malik, "Is predicting software security bugs using deep learning better than the traditional machine learning algorithms?," IEEE International Conference on Software Quality, Reliability and Security (QRS), pp. 95–102, 2018.

[18] O. Dahiya and K. Solanki,"An Efficient APHT Technique for Requirement-Based Test Case Prioritization," International Journal of Engineering Trends and Technology, vol. 69, pp. 215–227, 2021.

[19] O. Dahiya and K. Solanki,"Prevailing Standards in Requirement-Based Test Case Prioritization: An Overview," ICT Analysis and Applications, pp. 467–474, 2021.

[20] O. Dahiya and K. Solanki,"A Study on Identification of Issues and Challenges Encountered in Software Testing," In Proceedings of International Conference on Communication and Artificial Intelligence , pp. 549–556, 2021.

[21] O. Dahiya, K. Solanki, and A. Dhankhar, "Risk-based testing: identifying, assessing, mitigating & managing risks efficiently in software testing," International Journal of advanced research in engineering and technology, vol. 11, pp. 192–203, 2020.

[22] O.Dahiya and K. Solanki,"A systematic literature study of regression test case prioritization approaches," International Journal of Engineering & Technology, vol. 7, pp. 2184–2191, 2018.

[23] S. Yadav and B. Kishan, "Reliability of Component-Based Systems- A Review", International Journal of Advanced Trends in Computer Science and Engineering, vol.8, pp. 293–299, 2019.

[24] S. Yadav and B. Kishan, "Assessment of software quality models to measure the effectiveness of software quality parameters for Component Based Software (CBS)," Journal of Applied Science andComputations, vol. 6, pp. 2751–2756, 2019.

[25] S. Yadav and B. Kishan, "Analysis and Assessment of Existing Software Quality Models to Predict the Reliability of Component-Based Software," International Journal of Emerging Trends in Engineering Research, vol. 8, pp. 2824–2840,2020.

[26] S. Yadav and B. Kishan, "Component-Based Software System using Computational Intelligence Technique for Reliability Prediction," International Journal of Advanced Trends in Computer Science and Engineering, vol. 9, pp. 3708–3721, 2020.

[27] S. Yadav and B. Kishan, "Assessments of Computational Intelligence Techniques for Predicting Reliability of Component Based Software Parameter and Design Issues," International Journal of Advanced Research in Engineering and Technology, vol. 11, pp. 565–584, 2020.

[28] O. Al Qasem and M. Akour, "Software fault prediction using deep learning algorithms," International Journal of Open Source Software and Processes (IJOSSP), vol. 10, pp. 1–19, 2019.

[29] Wikipedia contributors, "Activation function," [Online]. Available: https://en.wikipedia.org/w/index.php?title=Activation_function&oldid=1076034609. [Accessed 2022].

[30] Dishashree26, "Activation Functions | Fundamentals Of Deep Learning," January 2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/. [Accessed December 2021].

[31] Wikipedia contributors, "Cross entropy," 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Cross_entropy&oldid=1071450106. [Accessed 2022].

[32] M. Sunasra, "Performance Metrics for Classification problems in Machine Learning," March 2019. [Online]. Available: https://medium.com/@MohammedS/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b. [Accessed January 2022].

[33] B. S. Deshpande, B. Kumar and A. Kumar, "Assessment of Software Reliability by Object Oriented Metrics using Machine Learning Techniques," International Journal of Grid and Distributed Computing, vol. 14, pp. 01–10, 2021.

[34] T. Wang, W. Li, H. Shi and Z. Liu, "Software defect prediction based on classifiers ensemble," Journal of Information & Computational Science, vol. 8, pp. 4241–4254, 2011.

[35] K. N. Rao and C. S. Reddy, "A novel under sampling strategy for efficient software defect analysis of skewed distributed data," Evolving Systems, vol. 11, pp. 119–131, 2020.

[36] K. Wang, L. Liu, C. Yuan and Z. Wang, "Software defect prediction model based on LASSO--SVM," Neural Computing and Applications, pp. 8249–8259, 2021.

[37] C. Shan, H. Zhu, C. Hu, J. Cui and J. Xue, "Software defect prediction model based on improved LLE-SVM," in 2015 4th International Conference on Computer Science and Network Technology (ICCSNT), vol. 1, 2015, pp. 530–535.

[38] L. Qiao, X. Li, Q. Umer and P. Guo, "Deep learning based software defect prediction," Neurocomputing, vol. 385, pp. 100–110, 2020.