ISSN: 2633-4828 Vol. 5 No.4, December, 2023

International Journal of Applied Engineering & Technology

DEEP LEARNING AND ARTIFICIAL IMMUNE SYSTEM BASED SOFTWARE DEFECT DETECTION

Kavita Chourasia[1] and Dr. Harsh Mathur[2]
[1]Phd Scholar and [2]Associate Professor, Department of Computer Science, Rabindra Nath Tagore University, Bhopal, MP, India
[1]kavichourasia@gmail.com and [2]harsh.mathur@aisectuniversity.ac.in

ABSTRACT
Quality in software is a crucial element, given the rising demand. As software designs grow in complexity, the likelihood of encountering defects increases. Testers play a key role in enhancing software quality by addressing these defects. Consequently, defect analysis becomes instrumental in the overall improvement of software quality. The intricate nature of software contributes to a greater occurrence of defects, making manual detection a time-intensive endeavor. This challenge has motivated researchers to devise methods for automated software defect detection. In this paper a novel deep learning model SDDAIDL (Software Defect Detection by Artificial Immune & Deep Learning) was developed by optimizing the input software features. Two stage feature optimization was done first is artificial immune system and other was convolution, maxpooling operations. Filter and transformed features were used for the learning of fully connected neural network. Experiment was done on different software feature sets and result shows that proposed model has increase the evaluation parameter values.

Index Terms: Deep Learning, Software Defect Detection Genetic Algorithm, Feature optimization.

I. INTRODUCTION
The software business is dynamic because it responds to consumer demand and advances in technology. Since people are responsible for the bulk of software development, errors are to be expected [1]. Planning, analyzing, designing, implementing, testing, integrating, and maintaining a software system nowadays is no easy task. A software engineer's job entails meeting strict deadlines and spending constraints while creating complex software systems. Software engineering (SE) professionals and academics have worked on improving the quality and decreasing the cost of the software development life-cycle for decades. Numerous academic works have been produced in pursuit of better, more efficient methods of pinpointing potential problem spots [2]. However, the difficulty rises as the size of the code-base continues to expand.

When creating software, it's fairly uncommon for errors to arise due to things like sloppy logic or inadequate data handling, both of which force developers to start again and drive up the price of maintenance. The decline in satisfied customers may be traced back to all of these factors. In this paradigm, problems are prioritized by severity, and remedial and preventative measures are implemented accordingly [3].

Defects in software have the potential to significantly lower the quality of the product, which is an issue for both consumers and programmers. Manual software detection becomes a difficult and time-consuming operation due to the increasing complexity of program designs and technologies [4]. Because of the high stakes involved, expanding the knowledge base and developing novel defect prediction and modeling approaches is of utmost importance in the software engineering industry, which is particularly vulnerable to financial losses caused by defects. To understand how to use machine learning software defect prediction to enhance quality and save costs during software development. This has made research into automated software identification a priority for many businesses in recent years. Many research have examined various ML methods, such as Bayesian learning, neural networks, Support Vector Machine (SVM), and so on, with the goal of creating defect prediction models for certain classes. But what you need to know is that all of these methods have varying degrees of success with different types of data sets. In order to address this issue, the authors of this research presented a deep learning model [5].

Here's how the rest of the paper is laid out: In Section 2, we take a quick look at the many different defect models for software that have been presented. In addition, the SDDAIDL model is fully described with block diagram and explanation in Section 3. Section 4 describes the experiment, whereas Section 5 discusses the findings and draws conclusions.

II. RELATED WORK

In [6], S. M. Rifat et al. present an enhanced YOLOv3 network model and generate a large-scale bearing-cover defect dataset. The proposed model comprises four submodels: a Bottleneck Attention Network, serving as an attention prediction subnet model, a large-size output feature branch and a defect localization subnet model. This work devise a comparison experiment under unusual lighting circumstances to assess the new model's generalizability, robustness, and applicability. To test the reliability of the suggested modules, we devise an ablation experiment.

Combinations of machine learning techniques have been proven to be useful for software fault prediction by Z. Zheng et al. in [7]. Intriguingly, the mean accuracy of a classifier built from an Artificial Neural Network and a Random Forest is 91%. The results of this study prove that Machine Learning may be an effective tool in the field of computer programming.

To ensure that projects of varying sizes are handled with the same degree of detail, A. S. Kurdija et al. in [8] developed a model based on a convolutional graph for defect detection. The model analyzes the nodes and edges in the abstract syntax tree of a software module's source code to determine whether or not the module is flawed.

In order to gain a more comprehensive understanding of how software attributes impact the effectiveness of deep learning-based software defect predictors, D. -L. Miholca et al. conducted a thorough investigation, as detailed in [9]. Going beyond the extensive feature sets commonly found in literature for identifying defect-proneness, we enhance our approach by incorporating conceptual software characteristics that encapsulate the semantics of the source code. To automatically engineer these conceptual properties.

A hybrid Deep Neural Network model for improved software bug prediction is presented by Kajal Tameswar et al. in [10]. To better optimize the Deep Neural Network design, a number of different Nature-Inspired Algorithms have been used to the problem of exploring the hyperparameter solution space. Using a NASA dataset, researchers have undertaken experiments to try and anticipate software faults.

In [11], Jingyu Liu et al. integrate the codes from several computer languages with information derived from natural language literature in order to improve the semantic characteristics. an innovative approach for software defect prediction that integrates the Transformer architecture with a multi-channel CNN. The model leverages a pretrained language model and a CNN-based classifier to generate context-aware representations and capture the local correlation of sequences.

Using a feature set consisting of classes, objects, codes, etc., Tanujit et al. [12] suggested a new hybrid approach for locating software flaws. In order to improve the efficacy of feed-forward neural network models, the authors of this research created a hellinger tree. The Hellinger net model employs a very asymmetric distance for dealing with class problems. The adoption of a tree topology for neural network learning is effective in general. This model's functioning accuracy might be improved by additional feature reduction.

III. PROPOSED METHODOLOGY

In this study, we present a methodology for detecting defects in software at a late stage of development. Dataset processing and feature clustering are the initial steps in the proposed SDDAIDL (Software Defect Detection by Artificial Immune & Deep Learning) methodology. In Fig. 1 we saw a block diagram depicting the interaction between the various modules. Table 1 contains a selection of the notation used to describe the suggested model.

Table 1: Notation used in SDDAIDL.

Notation	Meaning
SDRD	Software Dataset Raw Dataset
PD	Processed Data
ND	Normalized Data
J	J^{th} Column in Data
F	Number of Features
F_s	Selected Features
F_r	Rejected Features
A	Antibody
AF	Fitness
Ff	Filter feature
B	Block of software session
C	Convolution
s	Stride
p	Padding

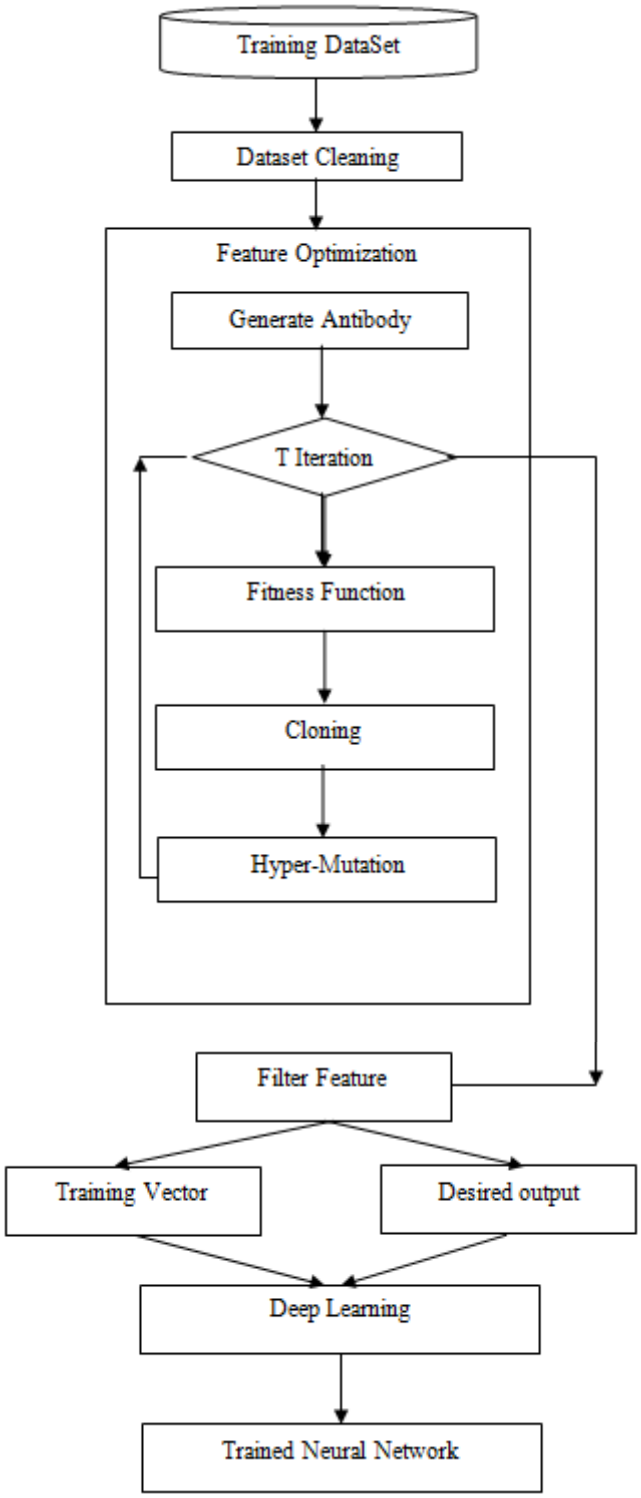

Fig. 1 Proposed SDDAIDL block diagram.

Raw Dataset Cleaning Due to the original data's unstructured nature, the cleaning process involved converting the data to meet the specific environment's requirements for rows and columns. The dataset was cleaned up by

removing things like software versions, class names, function names, etc. Column average or zero count was used to fill in missing data or cells with null values [13]. Some numbers in the paper's examples ranged from zero to one ('Data Access Metrics,' 'Cohesion across Methods,' etc.), while others ranged in the hundreds ('Inheritance Depth,' 'Measure of Aggregation,' etc.). For example, the amount of lines of code might be thousands long. Each column's values were sent through Eq. 1 individually to complete the normalization [14].

$$ND_{i,j} = \frac{PC_{i,j} - Min(PC_j)}{Max(PC_j) - Min(PC_j)}$$ ----------Eq. 1

The values in columns PC are normalized from zero to one using Eq. 1. The normalize function mitigates the bias that might arise while training on a limited set of features. In the case of feature selection with uneven values, the leaning is diminished regardless of which column is chosen (high or low range).

Clustering of Features

The suggested approach employs an artificial immune system [15] that has been tweaked to successfully cultivate input processed feature sets.

The defect class is determined by analyzing a matrix of characteristics extracted from the preprocessed dataset. This study employs an artificial immune system technique to compile such a feature ontology. This method updates the population in two ways, both of which change the chromosomal values in a single iteration: by cloning and by mutation. The chances of arriving at a successful feature ontology solution improve with each iteration thanks to a random cloning and mutation stage.

Generate Antibodies

To generate antibodies in the context of this study, a random set of features is formulated using a Gaussian function, resulting in a binary feature set composed of 0s and 1s. This binary feature set is similar to an antibody in the genetic algorithm, where the presence of a feature is denoted by 1, and its absence is denoted by 0. Within this framework, each feature in the population is equipped with two flags: 1 signifies presence, while 0 indicates absence. Additionally, the population is defined by lower and upper bounds, specifying the presence and absence of features, respectively. This closes in the generation of m antibodies (denoted as A), representing the initial population, shown in Eq. 2.

A←Generate_Antibody(m) ----------Eq. 2

Affinity

The concept of affinity is introduced to evaluate the effectiveness of antibodies within the population. Affinity is gauged by constructing a temporary deep learning model, and defect detection is performed based on the trained model to estimate accuracy. The accuracy value estimate as per Eq. 3, specifically in correctly detecting the defect class, serves as the affinity of the antibody.

Af←Affinity(A) ----------Eq. 3

Affinity

Input: A, PD

Output: Af

1. Loop 1:m
2. [Fs Fr]←A[m]
3. TModel←TrainCNN(Fs, PD)
4. Af[m]←TestCNN(PD)
5. End

Cloning

Cloning, a crucial step in the process, involves deriving the best solution (Ab) from the population by considering the affinity value of each antibody. In the cloning phase, the status of a few features in the best antibody, Ab, is randomly altered. This alteration may involve changing the status from present to absent or vice versa, ultimately leading to the cloning of the model.

$A \leftarrow Cloning(A_b, A)$ ----------Eq. 4

Hypermutation

The chromosomes undergo a hypermutation procedure, wherein they are mutated in inverse proportion to their affinity. The clones derived from the best antibody undergo the least mutation, while those from the poorest antibody undergo the most significant mutation. The hypermutation process can exhibit uniform, Gaussian, or exponential characteristics. Following mutation, an analysis is conducted on both the clones and their original antibodies, and the best N antibodies are selected for the subsequent iteration in the evolutionary algorithm.

$A \leftarrow Hypermutation(A)$ ----------Eq. 5

Final Feature Set

After T number of iteration of artificial immune system algorithm final feature set was select. This selected feature set was used for the training of spiking neural network. Normalized dataset selected features were transformed into training vector with desired output class of the either defect represent by 1 or normal class represent by 0.

Deep Learning Model Convolution Neural Network

The main goal of CNN is to utilize filter feature Ff, structural information and reuse weight parameters. To achieve this goal, CNNs propose two new operations (i.e., the convolution operation and the pooling operation) [17]. In order to increase the learning above two operation transform the matrix geometrically for updates. [14, 15]. This section, briefly discuss the convolution operation and the pooling operation in CNNs. Input feature vector os software session is transformed into block B having dimension bxb.

$B \leftarrow Block(FF, b)$--------Eq. 6

Convolution

Hence, CNNs are easier to train and less vulnerable to over-fitting.

$C \leftarrow ConvolutionOperator(B, s, p, F_c)$--------Eq. 7

Stride is a variable employed for controlling movement speed, characterized by integer values. Padding involves the addition of null rows or columns to a block if deemed necessary. The filter, denoted as F, is applied to the block B.

Max-Pooling

Max-pooling is a common technique in Convolutional Neural Networks (CNNs) for downsampling, achieved through either maximum pooling or average pooling. Post-pooling, the filter feature maps are effectively enlarged by a factor of a, making the convolution operation s times more impactful in expanding the receptive field. To progressively encode high-level filter features, pooling operations typically employ a factor of s = 2. Convolution and pooling operations often collaborate in groups to achieve their objectives.

$C \leftarrow Maxpooling\ (C, s, p, F_m)$--------Eq. 8

Fully Convolutional Networks

In Fully Convolutional Networks (FCNs), convolution and pooling operations enable CNNs to extract spatial information from the input space. However, the per-pixel classification formulation in FCNs hinders the utilization of spatial correlation in the output space. The input block B, obtained after the convolutional operation, serves as a training vector in FCN. The desired output is categorized as either a defect or non-defect class. A

trained CNN can directly accept the processed (convolutional operation) blocked filter feature as input and predict the block class (defect or normal).

Testing of Deep Learning model

In this second module of proposed work SDDAIDL input raw dataset was processed and filter by selected artificial immune system or final feature sets. Extracted features were normalize and passed in the trained deep trained model to get class of the software session either defect or normal.

Proposed SDDAIDL Algorithm

Input: SDRD // Software Defect Raw Dataset

Output: SDDM // Software Defect Detection Model

1. PD←Dataset_Cleaning(SDRD)
2. **A←Generate_Antibody(m)**
3. **Loop 1:T**
4. A←Cloning(A_b, A)
5. **Af←Detection_accuracy(A)**
6. A←Hypermutation(A)
7. End Loop
8. Ff←Max_position_Antibody(Affinity(A))
9. B←Block(FF, b)
10. Loop 1:n // Number of software sessions
11. B←ConvolutionOperator(B, s, p, F_c)
12. B←Maxpooling (B, s, p, F_m)
13. EndLoop
14. SDDM←TrainCNN(Fs, B)

Input raw dataset is preprocessed and find best set of features for the software bug detection was done by this model SDDAIDL. Artificial Immune System was used for the feature selection and CNN for the leaning. Trained model will predict the defect in software.

IV. EXPERIMENT AND RESULTS

The implementation of the model involved utilizing a MATLAB program on a computer equipped with a 4 GB RAM and an i6 generation CPU. To assess the performance of the model, metrics including precision, recall, f-measure, and accuracy were employed for comparative analysis.

Dataset

In order to conduct the experiments, the IC-DePress dataset served as the primary dataset for the program. The evaluation of the proposed model spanned across six separate projects, encompassing a total of 13,522 sessions within the dataset. A comparison of the proposed model was conducted using software fault detection methodologies as outlined in [12].

Evaluation Parameters

Precision, Recall, and F-score were evaluated as test factors for predictive ability. The True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN) values [19] determine these values.

Table 1 Confusion matrix for comparison.

Actual	Algorithm	
	True	**False**
Positive	TP	FP
Negative	TN	FN

In above true positive value is obtain by the system when the prediction class and actual class also says that defect present. While in case of false positive value it is obtain by the system when the input session is not defected and actual class also says that session is defected.

RESULTS

Table 2 Accuracy based comparison of software defect detection models.

Software	**Hellinger Net**	SDDAIDL
S_Camel	83.12	88.21
S_IVY	83.55	84.19
S_JEdit	87.45	89.72
S_Licene	73.34	75.66
S_POI	69.04	79.72

Table 2 shows accuracy of software defect detection models. It was found that for all set testing software proposed model SDDAIDL has performed well as compared to existing model Hellinger Net. Further fig. 2 shows average accuracy of software defect detection and it was found that SDDAIDL model has improved values by 4.2% as compared Hellinger Net. Hence use of artificial immune gentic algorithm has increases the work efficiency of learning and detection.

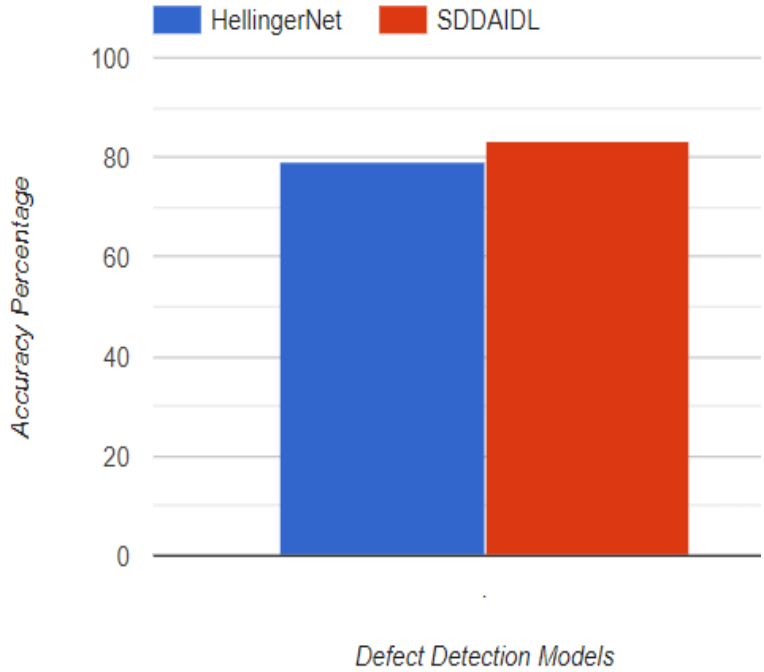

Fig. 2: Average accuracy percentage value based comparison of software defect detection models.

Table 3 Precision based comparison of software defect detection models.

Software	**Hellinger Net**	SDDAIDL
S_Camel	0.9556	1

S_IVY	0.9974	1
S_JEdit	0.9825	0.9994
S_Licene	0.855	0.9059
S_POI	0.8898	0.9983

Precision values of software defect detection models were shown in table 2 and it was found that proposed SDDAIDL has improved the values. Fig. 3 shows that proposed SDDDIDL model has improved the precision value by 4.55% as compared to existing model.

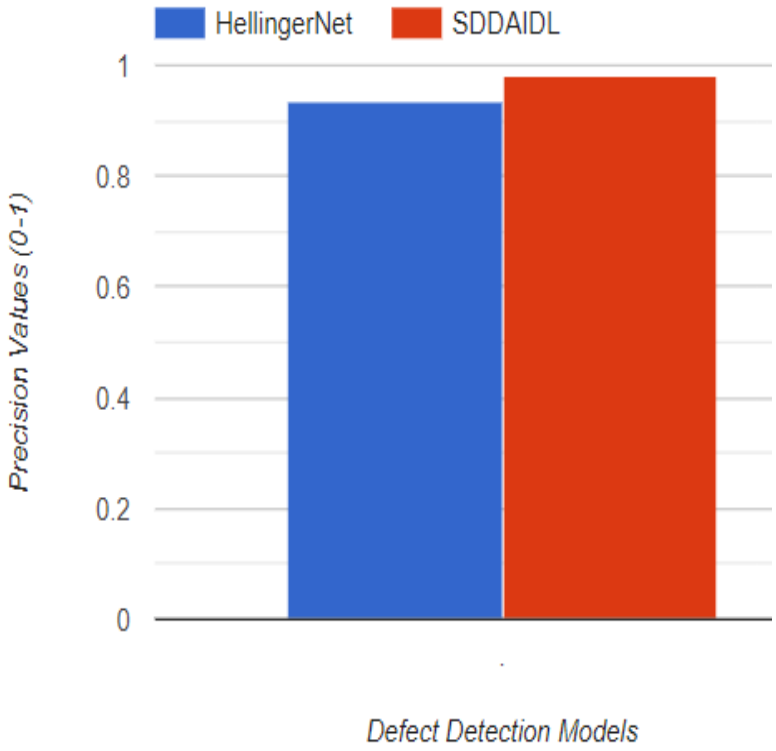

Fig. 3: Average precision value based comparison of software defect detection models.

Table 4: Recall based comparison of software defect detection models.

Software	**Hellinger Net**	SDDAIDL
S_Camel	0.8311	0.9073
S_IVY	0.8351	0.842
S_JEdit	0.8736	0.9065
S_Licene	0.7417	0.7887
S_POI	0.6905	0.8285

Table 4 shows recall values of comparing models of software detect detection and it was found that use artificial immune genetic algorithm for feature reduction has improved the work efficiency. SDDAIDL has increases the recall value by 7.04% as compared to existing model comparing model proposed in [12].

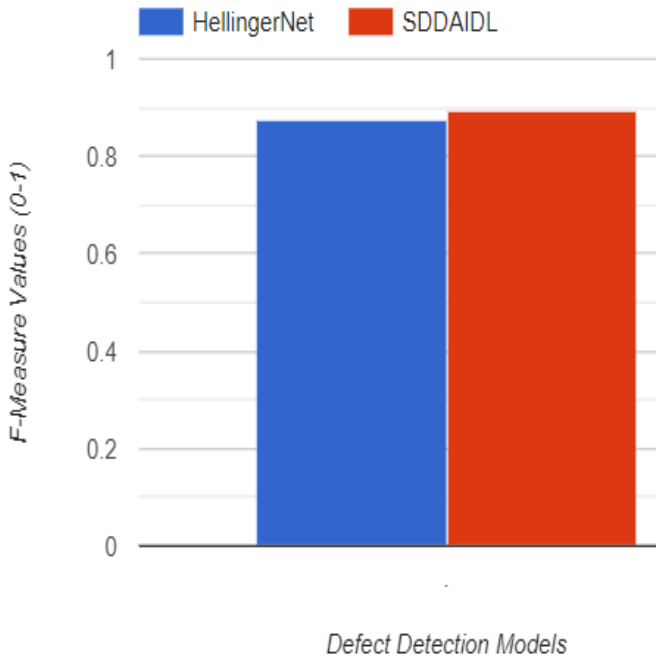

Fig. 4: Average F-measure value based comparison of software defect detection models.

Table 5: F-measure based comparison of software defect detection models.

Software	Hellinger Net	SDDAIDL
S_Camel	0.9078	0.9308
S_IVY	0.9102	0.9131
S_JEdit	0.9323	0.9429
S_Licene	0.8156	0.8205
S_POI	0.8163	0.8581

F-measure values shown in table 5 for different testing software and it was obtained that CNN based detection model has high accurate. This enhancement of CNN was obtained by use of filtered feature set of AIS algorithm.

Table 6: Reliability based comparison of software defect detection models.

Software	Hellinger Net	SDDAIDL
S_Camel	0.9469	0.9516
S_IVY	0.3951	0.3952
S_JEdit	0.9271	0.9276
S_Licene	0.598	0.6783
S_POI	0.607	0.6264

Reliability values of the proposed SDDAIDL was high as compared to Hellinger Net model in each testing software. Reliability values of SDDAIDL was improved by 2.93% as compared to Hellinger Net.

V. CONCLUSIONS

This paper presents the development of a software defect detection model achieved through the optimization of the input dataset. The optimization process involved carefully selecting suitable attributes using an artificial immune system. Additionally, the incorporation of convolutional and max-pooling operators contributed to enhancing the model's overall performance. Experiments were conducted using authentic software testing datasets, and evaluations were carried out across various parameters. The results indicate a noteworthy enhancement in the accuracy of defect detection by 5.029%, alongside a considerable improvement in recall

values by 7.04%. These findings underscore the effectiveness of feature optimization in the model's learning process. Moving forward, researchers are encouraged to explore alternative learning models for the detection of software defects, building upon the insights gained from this study.

REFERENCES

1. M. M. T. Thwin, & T. S. Quah, Application of Neural Network for Software Quality Prediction Using Object-Oriented Design Metrics, Journal of Systems and Software, 76(2), 2005, 147-156.
2. Manjula, C.; Florence, L. Deep neural network based hybrid approach for software defect prediction using software metrics. Clust. Comput. 2019, 22, 9847–9863.
3. Lino Ferreira da Silva Barros, M.H.; Oliveira Alves, G.; Morais Florêncio Souza, L.; da Silva Rocha, E.; Lorenzato de Oliveira, J.F.; Lynn, T.; Sampaio, V.; Endo, P.T. Benchmarking Machine Learning Models to Assist in the Prognosis of Tuberculosis. Informatics 2021.
4. Fan, G.; Diao, X.; Yu, H.; Yang, K.; Chen, L. Software defect prediction via attention-based recurrent neural network. ScientificProgramming 2019, 2019, 6230953.
5. Cetiner M, Sahingoz OK. A comparative analysis for machine learning based software defect prediction systems. Proceedings of the 2020 11th International Conference on Computing Communication and Networking Technologies (ICCCNT), 2020 1–3 July.; Kharagpur, India; 2020. p. 1–7.
6. S. M. Rifat, A. U. Bhuyain, M. S. Hossain, M. S. Mia and M. Rahman, "A Systematic Approach for Enhancing Software Defect Prediction Using Machine Learning," 2023 International Conference on Next-Generation Computing, IoT and Machine Learning (NCIM), Gazipur, Bangladesh, 2023, pp. 1-6.
7. Z. Zheng, J. Zhao and Y. Li, "Research on Detecting Bearing-Cover Defects Based on Improved YOLOv3," in IEEE Access, vol. 9, pp. 10304-10315, 2021.
8. L. Šikić, A. S. Kurdija, K. Vladimir and M. Šilić, "Graph Neural Network for Source Code Defect Prediction," in IEEE Access, vol. 10, pp. 10402-10415, 2022.
9. D. -L. Miholca, V. -I. Tomescu and G. Czibula, "An in-Depth Analysis of the Software Features' Impact on the Performance of Deep Learning-Based Software Defect Predictors," in IEEE Access, vol. 10, pp. 64801-64818, 2022.
10. Kajal Tameswar, Geerish Suddul, Kumar Dookhitram. "A hybrid deep learning approach with genetic and coral reefs metaheuristics for enhanced defect detection in software". International Journal of Information Management Data Insights, Volume 2, Issue 2, 2022.
11. Jingyu Liu, Jun Ai, Minyan Lu, Jie Wang, Haoxiang Shi. "Semantic feature learning for software defect prediction from source code and external knowledge". Journal of Systems and Software Volume 204, 2023.
12. Tanujit Chakraborty and Ashis Kumar Chakraborty. "Hellinger Net: A Hybrid Imbalance Learning Model to Improve Software Defect Prediction". IEEE Transactions On Reliability, 2020.
13. Karpagalingam Thirumoorthy, Jerold John Britto J. "A feature selection model for software defect prediction using binary Rao optimization algorithm" Applied Soft Computing, Volume 131, 2022.
14. Balogun, A.O.; Basri, S.; Capretz, L.F.; Mahamad, S.; Imam, A.A.; Almomani, M.A.; Adeyemo, V.E.; Alazzawi, A.K.; Bajeh, A.O.; Kumar, G. Software Defect Prediction Using Wrapper Feature Selection Based on Dynamic Re-Ranking Strategy. Symmetry 2021, 13, 2166.
15. Ying Tan, "Artificial Immune System," in Artificial Immune System: Applications in Computer Security , IEEE, 2016, pp.1-25.

16. I F Astachova et al. "The application of artificial immune system to solve recognition problems". 2019 J. Phys.: Conf. Ser. 1203 012036.
17. Jyoti Ahirwar , Dr. Mukesh Yadav. "Intrusion Detection System using Machine Learning Approach". IJSRET Journal Volume 7, issue 6, nov-dec 2021.
18. Khleel, N.A.A., Nehéz, K. A novel approach for software defect prediction using CNN and GRU based on SMOTE Tomek method. J Intell Inf Syst 60, 673–707 (2023).

First Author

Mrs. Kavita Chourasia

Phd.Scholar, Dept of computer Science, Rabindra Nath Tagore University,Area of Interest:Machine Learning,Data Science. Presently working on Deep Learning and Artificial Immune System Based Software Defect Detection.

Brief Profile: Kavita Chourasia is an accomplished educator and researcher with a Master's in Technology from RGPV University. Her thesis focused on enhancing centroid text classification algorithms. With nine years of teaching experience, she's known for her innovative teaching methods and dedication to student success. Kavita's passion for learning and research continues to drive advancements in computer science and technology.

Second Author:

Dr. Harsh Mathur

has more than 13 years of Experience in teaching. He has done his PhD in Computer Science and Engineering from Rabindra Nath Tagore University, Raisen in 2019. His Area of interest are Image processing, Vehicular Adhoc Network, Artificial Intelligence, Machine Learning etc. He has guided 7 PhD students and 8 students are currently pursuing and more than 35 Mtech students till date in their research work. He has published patents in Artificial intelligence and Machine learning Field. He is also a life time member of CSTA, IJCSE, IAENG and SSRG.He has Published more than 50 International research papers in Reputed journals , Springer Chapters ,IEEE Conferences and Scopus Journals.