Vol 6, Issue 11, 2023 **Impact Factor: 5.355** DOI: https://doi.org/10.31426/ijamsr.2023.6.11.6914

IJAMSR

International Journal of Advanced Multidisciplinary Scientific Research (IJAMSR) ISSN:2581-4281

Enhancing CNNs for Software Defect Prediction: Addressing Imbalance and Noise

Deshmukh Pankaja Rajebhau [1], Dr. Satish Narayan Gurjar [2]

[1] Research Scholar, Dept. Of Computer Science, University of Technology, Jaipur
[2] Dept. Of Computer Science, University of Technology, Jaipur

Email: Pankajadeshmukh@Gmail.Com

ABSTRACT

This study explores the application of Convolutional Neural Networks (CNNs) for software defect prediction, highlighting promising performance metrics while addressing critical challenges such as data imbalance, noise, and model interpretability. Despite achieving high precision for non-faulty instances, the models struggled with accurately identifying faulty components, underscoring the need for balanced F1-scores. Solutions proposed included SMOTE for data imbalance and noise reduction techniques like outlier detection. Optimization efforts focused on hyperparameter tuning and architecture refinement to enhance model robustness. Limitations included computational complexity and the interpretability of CNNs. Practical implications for software engineering involve improving defect detection to enhance software quality and efficiency. The implementation of CNNs demonstrated commendable precision and recall metrics for non-faulty instances, indicating a robust capability to identify correctly functioning software components. However, the models encountered difficulties in accurately detecting faulty instances, as reflected in lower recall rates and a notable number of false negatives. One of the primary challenges identified was the inherent imbalance in the dataset, where non-faulty instances significantly outnumbered faulty instances. This disparity skewed predictions towards the majority class, thereby reducing the model's effectiveness in identifying faults accurately. Proposed solutions included Synthetic Minority Over-sampling Technique (SMOTE) and Adaptive Synthetic Sampling (ADASYN) to rectify this imbalance and improve model performance. Addressing noise, such as outliers and irrelevant data points, emerged as another critical area for enhancing model robustness.

Keywords: ***CNNs, Software Defect Prediction, Data Imbalance, Noise Reduction, Interpretability, Precision, Recall, SMOTE, Outlier Detection, Hyperparameter Tuning.***

1. INTRODUCTION

Software defect prediction is a crucial practice aimed at identifying and addressing potential flaws in software before its release. This proactive approach can significantly reduce development costs, enhance user satisfaction, and improve overall software quality [1]. By predicting where defects are likely to occur, development teams can focus their testing efforts, allocate resources more effectively, and conduct targeted code reviews, leading to more reliable and robust software.

Key Elements of Software Defect Prediction

1. Metrics and Features: To predict defects, various metrics and features from the software code are utilized. These include code complexity, code churn (changes made to the code), code size, and historical defect data. These metrics are essential as they provide insights into the potential risk areas within the software [2-5]. Machine learning models use these metrics as inputs to identify patterns and trends that can forecast where defects are most likely to occur.
2. Machine Learning Algorithms: Predictive models are developed using various machine learning techniques such as neural networks, decision trees, and support vector machines. These algorithms analyze complex datasets to uncover patterns and relationships that might not be immediately apparent [6-10]. By training on historical data, these models can estimate the probability of defects in specific code modules or components, thus guiding where testing should be focused.
3. Data Preprocessing: Before data is fed into machine learning models, it undergoes preprocessing. This involves cleaning the data, handling missing values, and normalizing features to ensure that the data is of high quality [11-14]. Proper preprocessing is critical as it helps maintain the reliability and effectiveness of the predictive models.
4. Training and Testing: The predictive models are trained using historical data to learn patterns associated with defects. Once trained, these models are tested on new, untested code to evaluate their ability to generalize and accurately predict defects in unseen code [15-19]. This step is crucial to ensure that the models are not overfitted to the training data and can perform well in real-world scenarios.
5. Validation and Evaluation: To assess the effectiveness of defect prediction models, several evaluation metrics are used, including precision, recall, F1 score, and the area under the receiver operating characteristic (ROC) curve [20-23]. These metrics help determine how well the model identifies defective code while minimizing false positives and false negatives. High precision and recall rates are desirable as they indicate that the model is effective at predicting defects with minimal errors.
6. Integration into Development Workflow: Successful defect prediction models are integrated into the software development process. They assist development teams during the quality assurance phase by highlighting high-risk areas that need more rigorous testing [24-28]. This integration helps in strategically allocating resources, prioritizing testing efforts, and ensuring that critical areas receive the necessary attention.

7. Continuous Improvement: Defect prediction is an iterative process. As new data becomes available and the software evolves, models need to be regularly retrained and refined to maintain their effectiveness. Continuous improvement ensures that the models adapt to changes in the software and remain relevant in predicting defects.

Role in the Testing Phase of SDLC

During the Testing Phase of the Software Development Life Cycle (SDLC), defect prediction plays a vital role. It helps identify which modules are most likely to contain errors and therefore require more detailed testing [29-33]. By focusing on these high-risk areas, development teams can use their resources more efficiently and adhere to project constraints. While predicting defect-prone modules can be challenging, accurate predictions can significantly reduce the costs associated with defect detection and correction [34-36]. Automated tools that accurately forecast defect locations and guide testing efforts have the potential to save substantial amounts of money annually and improve overall software quality.

2. REVIEW OF LITERATURE

Zain et.al., (2022). Developing successful software with no defects is one of the main goals of software projects. In order to provide a software project with the anticipated software quality, the prediction of software defects plays a vital role. Machine learning, and particularly deep learning, have been advocated for predicting software defects, however both suffer from inadequate accuracy, overfitting, and complicated structure. In this paper, we aim to address such issues in predicting software defects. We propose a novel structure of 1- Dimensional Convolutional Neural Network (1D-CNN), a deep learning architecture to extract useful knowledge, identifying and modelling the knowledge in the data sequence, reduce overfitting, and finally, predict whether the units of code are defects prone. We design large-scale empirical studies to reveal the proposed model's effectiveness by comparing four established traditional machine learning baseline models and four state-of-the-art baselines in software defect prediction based on the NASA datasets. The experimental results demonstrate that in terms of f-measure, an optimal and modest 1DCNN with a dropout layer outperforms baseline and state-of-the-art models by 66.79% and 23.88%, respectively, in ways that minimize overfitting and improving prediction performance for software defects. According to the results, 1D-CNN seems to be successful in predicting software defects and may be applied and adopted for a practical problem in software engineering. This, in turn, could lead to saving software development resources and producing more reliable software.

Rathore et.al., (2022). Imbalanced software fault datasets, having fewer faulty modules than the nonfaulty modules, make accurate fault prediction difficult. It is challenging for software practitioners to handle imbalanced fault data during software fault prediction (SFP). Earlier, several researchers have applied oversampling techniques such as synthetic minority oversampling techniques and others for imbalanced learning in SFP. However, most of these techniques resulted in overfitted prediction models. This article presents generative oversampling methods to handle imbalanced data problems in the SFP. Using the generative adversarial network (GAN) based approach, the presented methods generate synthetic samples of the faulty modules to balance the proportion of faulty and nonfaulty modules in the fault datasets. Further, SFP models are built on the processed fault datasets using different machine learning techniques.

Farid et.al., (2021). In recent years, the software industry has invested substantial effort to improve software quality in organizations. Applying proactive software defect prediction will help developers and white box testers to find the defects earlier, and this will reduce the time and effort. Traditional software defect prediction models concentrate on traditional features of source code including code complexity, lines of code, etc. However, these features fail to extract the semantics of source code. In this research, we propose a hybrid model that is called CBIL. CBIL can predict the defective areas of source code. It extracts Abstract Syntax Tree (AST) tokens as vectors from source code. Mapping and word embedding turn integer vectors into dense vectors. Then, Convolutional Neural Network (CNN) extracts the semantics of AST tokens.

Wang, T., & Li, W. H. (2010). While there was much disagreement over the usefulness of employing static code features to train a defect predictor, software defect predictions were undeniably a powerful tool for increasing testing productivity and software quality. Defect forecasts had previously included a number of data mining techniques. A defect predictor based on Naive Bayes theory was found to exist in several forms, and the difference estimate technique and algorithm complexity of each version were examined. By completing prediction performance assessment, Multivariants Gauss Naive Bayes (MvGNB) was determined to be the best. Next, this model was contrasted with the J48 decision tree learner. It was concluded from experiment findings on MDP benchmarking datasets that MvGNB would be helpful for fault predictions.

Wang et.al., (2010). Many data mining applications relied heavily on feature selection since it was realized that using a single technique for subset selection might result in local optima. As a countermeasure, feature selection method ensembles were investigated, with the goal of integrating many approaches instead of depending on one. A comprehensive empirical investigation was carried out that looked at 17 different ensembles of feature ranking algorithms, including the signal-to-noise filter, 11 threshold-based techniques, and six commonly used methods. With 16 real-world software measurement datasets of different sizes, 13,600 classification models were built by the research.

Zheng, J. (2010). In order to anticipate software defects, the research examined the performance of three cost-sensitive boosting algorithms. Its main objective was to categorize software modules into classes that were prone to defects and those that were not. The high expense of incorrectly categorizing defective modules as opposed to non-defective ones was the goal of these techniques. The first approach improved the accuracy of identifying modules that were prone to defects by adjusting the classification threshold via the use of threshold moving. The threshold moving approach was shown to be the best appropriate for building cost-sensitive software defect prediction models via evaluation utilizing NASA project datasets, especially for projects that were produced using object-oriented language.

Khoshgoftaar et.al., (2010). Choosing the best characteristics for machine learning and dealing with unbalanced data presented difficulties for the data mining and machine learning communities. This research presented a procedure incorporating feature selection and data sample approaches. It was focused on the software engineering sector, specifically software quality prediction. In order to identify and model features for software defect prediction, four scenarios involving the comparison of original and sampled data were investigated. Defect prediction models showed consistent performance regardless of whether training data was original or sampled, according to case study results using nine software measurement datasets. Models based on feature selection from sampled data outperformed those based on original data.

Yan et.al., (2010). By using software metrics to anticipate the number of defects in software modules, regression methods were used to improve the quality of the program. A unique approach to estimating software defect counts was presented in the study: the use of fuzzy support vector regression (FSVR). The regressor's fuzzification input demonstrated proficiency in managing datasets with imbalanced software metrics. Based on experiment findings with the MIS and RSDIMU datasets, a comparative study with the support vector regression technique showed that FSVR offered reduced mean squared error and greater accuracy in forecasting the total number of defects for modules with a considerable defect load.

Table 2.1: Systematic Reviews and Methodology

Author(S) And Year	Research Key	Methodology	Findings
Wang, T., & Li, W. H. (2010)	Software defect prediction using static code attributes	Analyzed various versions of Naive Bayes-based defect predictors and compared Multivariants Gauss Naive Bayes (MvGNB) with the decision tree learner J48 using benchmarking datasets	MvGNB was found to be the most effective for defect prediction, improving software quality and testing efficiency
Wang et.al., (2010)	Feature selection in data mining applications	Conducted an empirical study examining 17 ensembles of feature ranking techniques using 16 software measurement datasets, resulting in 13,600 classification models	Ensembles with very few rankers were more effective than those with many or all rankers, highlighting the effectiveness of ensemble methods
Zheng, J. (2010)	Cost-sensitive boosting algorithms for software defect prediction	Investigated three cost-sensitive boosting algorithms focusing on misclassification costs using NASA project datasets	Threshold-moving algorithm was most suitable for cost-sensitive software defect prediction, especially for object-oriented projects

Khoshgoftaar et.al., (2010)	Feature selection and data sampling in software defect prediction	Explored four scenarios comparing original and sampled data for feature selection and modeling using nine software measurement datasets	Models constructed using feature selection from sampled data consistently outperformed those based solely on original data, demonstrating robust performance in defect prediction.
Yan et.al., (2010)	Enhancing software quality using regression techniques	Proposed Fuzzy Support Vector Regression (FSVR) for predicting software defect numbers, comparing it with support vector regression using MIS and RSDIMU datasets	FSVR yielded lower mean squared error and higher accuracy in predicting defect numbers for modules with substantial defect loads

Rawat and Dubey (2012) Certain faults were inevitable throughout software development, despite careful planning, comprehensive documentation, and efficient process management; this eventually resulted in a drop in quality and possible project failure. Even though it cost a lot of money, deliberate efforts were necessary to regulate and limit flaws in the competitive environment of the time. It also demonstrated how several defect prediction algorithms were put into practice and how successful they were in lowering the total number of faults.

Ma et.al., (2012). Models trained on cross-company data received less attention than within-company data when it came to software fault prediction. There were difficulties in translating within-company models into real-world situations since there were few local data repositories. A new technique for cross-company defect prediction called Transfer Naive Bayes (TNB) was devised when it was realized that transfer learning may help close this gap. In contrast to earlier methods, TNB estimated test data distribution and transferred cross-company data insights into weighted training instances by using information from all relevant characteristics in the training data. The experiments were carried out on a variety of datasets from NASA and Turkish software repositories. The research came to the conclusion that when local training data is scarce, using information from various distribution training data at the feature level may be helpful. This presents a potential way to maximize resource allocation in software testing procedures and lower related expenses.

Li et.al., (2012). Predicting software defects was thought to be a useful tool for improving our comprehension and management of software quality; nevertheless, most approaches relied on project history. However, effective defect prediction was hampered by the lack of such data for new projects and certain businesses. In order to address this, the study suggested sample-based approaches and supported the testing and selection of a tiny proportion of modules inside complex software systems. There was a description of three approaches for selecting samples: The study presented ACoForest, a unique active semi supervised learning technique that selects modules most useful for creating a reliable prediction model. The usefulness of these approaches was shown by experimental findings using PROMISE datasets, indicating that they may find use in industrial practice.

Sun et.al., (2012). In the past, there has been a lot of interest in the use of classification techniques to forecast software error proneness using static code properties. Software defect data's intrinsic classimbalance posed a significant difficulty, which led to the use of a number of techniques including cost-sensitive learning, boosting, bagging, and sampling to address it. But because of changes in the initial data distribution, these traditional methods often ran into issues including overfitting, unanticipated mistakes, and the loss of important information. In response, the study introduced a novel methodology that initially transformed imbalanced binary-class data into balanced multiclass data, followed by the implementation of a specific coding strategy to develop a defect predictor. Experimental results, conducted across 14 NASA datasets using four classification algorithms, three data coding schemes, and six common imbalance data handling techniques, indicated that the new approach generally outperformed others, particularly when employing a one-against-one coding scheme.

Pelayo, L., & Dick, S. (2012). Several studies have employed machine learning techniques to tackle the challenge of predicting software defects based on software metrics. But as a result of skewness in defect prediction datasets, predictions for the minority defective class were often less accurate—a problem referred to as "learning from imbalanced datasets." Using Analysis of Variance on a variety of datasets, this paper examined two important stratification options (under and oversampling) for software fault prediction. Under sampling had a significant main impact and an interaction effect with oversampling, both at $\alpha = 0.05$, according to the research, but oversampling did not have a significant main effect. The results advance our knowledge of efficient stratification strategies for resolving class imbalance in software fault prediction.

3. CONCLUSION AND FUTURE SCOPE

This work is a major step toward the prediction of software defects using deep learning methods, particularly CNNs. Although the models showed encouraging performance metrics and indicated how they may be used in software engineering, issues like noise, unbalanced data, and interpretability of the model still need to be further investigated and improved. For defect prediction models to become more dependable, scalable, and applicable in actual software development settings, these issues must be resolved. We can clear the way for more precise, effective, and significant software defect prediction solutions by advancing methods, working across disciplines, and developing and improving current procedures.

By means of this thorough study, we add to the current conversation on the nexus between software engineering and machine learning, hoping to provide practitioners with knowledge and instruments that improve software development and quality. The development of deep learning in software defect prediction has the potential to completely change the way we approach software maintenance and quality assurance in the future, eventually producing more reliable and robust software systems.

REFERENCES

1. Wahono, R. S. (2015). A systematic literature review of software defect prediction. *Journal of software engineering*, *1*(1), 116.
2. Rawat, M. S., & Dubey, S. K. (2012). Software defect prediction models for quality improvement: a literature study. *International Journal of Computer Science Issues (IJCSI)*, *9*(5), 288.
3. Laradji, I. H., Alshayeb, M., & Ghouti, L. (2015). Software defect prediction using ensemble learning on selected features. *Information and Software Technology*, *58*, 388402.
4. Okutan, A., & Yıldız, O. T. (2014). Software defect prediction using Bayesian networks. *Empirical Software Engineering*, *19*, 154181.
5. Nam, J. (2014). Survey on software defect prediction. *Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Tech. Rep*.
6. He, P., Li, B., Liu, X., Chen, J., & Ma, Y. (2015). An empirical study on software defect prediction with a simplified metric set. *Information and Software Technology*, *59*, 170190.
7. Jing, X. Y., Ying, S., Zhang, Z. W., Wu, S. S., & Liu, J. (2014, May). Dictionary learning based software defect prediction. In *Proceedings of the 36th international conference on software engineering* (pp. 414423).
8. Wang, S., & Yao, X. (2013). Using class imbalance learning for software defect prediction. *IEEE Transactions on Reliability*, *62*(2), 434443.
9. Ma, Y., Luo, G., Zeng, X., & Chen, A. (2012). Transfer learning for cross company software defect prediction. *Information and Software Technology*, *54*(3), 248256.
10. Li, M., Zhang, H., Wu, R., & Zhou, Z. H. (2012). Sample based software defect prediction with active and semi supervised learning. *Automated Software Engineering*, *19*, 201230.
11. Sun, Z., Song, Q., & Zhu, X. (2012). Using coding-based ensemble learning to improve software defect prediction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *42*(6), 18061817.
12. Arora, I., Tetarwal, V., & Saha, A. (2015). Open issues in software defect prediction. *Procedia Computer Science*, *46*, 906912.
13. Wang, T., & Li, W. H. (2010, December). Naive bayes software defect prediction model. In *2010 International conference on computational intelligence and software engineering* (pp. 14). Ieee.
14. Shepperd, M., Bowes, D., & Hall, T. (2014). Researcher bias: The use of machine learning in software defect prediction. *IEEE Transactions on Software Engineering*, *40*(6), 603616.
15. Czibula, G., Marian, Z., & Czibula, I. G. (2014). Software defect prediction using relational association rule mining. *Information Sciences*, *264*, 260278.
16. Rodriguez, D., Herraiz, I., Harrison, R., Dolado, J., & Riquelme, J. C. (2014, May). Preliminary comparison of techniques for dealing with imbalance in software defect prediction. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering* (pp. 110).

17. Ren, J., Qin, K., Ma, Y., & Luo, G. (2014). On software defect prediction using machine learning. *Journal of Applied Mathematics*, *2014*.
18. Arar, Ö. F., & Ayan, K. (2015). Software defect prediction using cost sensitive neural network. *Applied Soft Computing*, *33*, 263277.
19. Yang, X., Tang, K., & Yao, X. (2014). A Learning Works approach to software defect prediction. *IEEE Transactions on Reliability*, *64*(1), 234246.
20. Wang, H., Khoshgoftaar, T. M., & Napolitano, A. (2010, December). A comparative study of ensemble feature selection techniques for software defect prediction. In *2010 Ninth International Conference on Machine Learning and Applications* (pp. 135140). IEEE.
21. Liu, S., Chen, X., Liu, W., Chen, J., Gu, Q., & Chen, D. (2014, July). FECAR: A feature selection framework for software defect prediction. In *2014 IEEE 38th Annual Computer Software and Applications Conference* (pp. 426435). IEEE.
22. Shukla, H. S., & Verma, D. K. (2015). A review on software defect prediction. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, *4*(12), 43874394.
23. Zheng, J. (2010). Cost sensitive boosting neural networks for software defect prediction. *Expert Systems with Applications*, *37*(6), 45374543.
24. Gray, D., Bowes, D., Davey, N., Sun, Y., & Christianson, B. (2011, April). The misuse of the NASA metrics data program data sets for automated software defect prediction. In *15th annual conference on evaluation & assessment in software engineering (EASE 2011)* (pp. 96103). IET.
25. Pelayo, L., & Dick, S. (2012). Evaluating stratification alternatives to improve software defect prediction. *IEEE transactions on reliability*, *61*(2), 516525.
26. Bell, R. M., Ostrand, T. J., & Weyuker, E. J. (2013). The limited impact of individual developer data on software defect prediction. *Empirical Software Engineering*, *18*, 478505.
27. Agarwal, S., & Tomar, D. (2014). A feature selection-based model for software defect prediction. *assessment*, *65*.
28. Khoshgoftaar, T. M., Gao, K., & Seliya, N. (2010, October). Attribute selection and imbalanced data: Problems in software defect prediction. In *2010 22nd IEEE International conference on tools with artificial intelligence* (Vol. 1, pp. 137144). IEEE.
29. Vashisht, V., Lal, M., & Sureshchandar, G. S. (2015). A framework for software defect prediction using neural networks. *Journal of Software Engineering and Applications*, *8*(08), 384.
30. Wang, H., Khoshgoftaar, T. M., Van Hulse, J., & Gao, K. (2011). Metric selection for software defect prediction. *International Journal of Software Engineering and Knowledge Engineering*, *21*(02), 237257.
31. Wahono, R. S., Herman, N. S., & Ahmad, S. (2014). A comparison framework of classification models for software defect prediction. *Advanced Science Letters*, *20*(1011), 19451950.

32. Punitha, K., & Chitra, S. (2013, February). Software defect prediction using software metrics A survey. In *2013 International Conference on Information Communication and Embedded Systems (ICICES)* (pp. 555558). IEEE.
33. Liu, M., Miao, L., & Zhang, D. (2014). Two stage cost sensitive learning for software defect prediction. *IEEE Transactions on Reliability*, *63*(2), 676686.
34. Jindal, R., Malhotra, R., & Jain, A. (2014, October). Software defect prediction using neural networks. In *Proceedings of 3rd International Conference on Reliability, Infocom Technologies and Optimization* (pp. 16). IEEE.
35. Wahono, R. S., & Herman, N. S. (2014). Genetic feature selection for software defect prediction. *Advanced Science Letters*, *20*(1), 239244.
36. Wang, J., Shen, B., & Chen, Y. (2012, August). Compressed C4. 5 models for software defect prediction. In *2012 12th International Conference on quality software* (pp. 1316). IEEE.
37. Zain, Z. M., Sakri, S., Asmak Ismail, N. H., & Parizi, R. M. (2022). Software Defect Prediction Harnessing on Multi 1-Dimensional Convolutional Neural Network Structure. *Computers, Materials & Continua*, *71*(1).
38. Rathore, S. S., Chouhan, S. S., Jain, D. K., & Vachhani, A. G. (2022). Generative oversampling methods for handling imbalanced data in software fault prediction. *IEEE Transactions on Reliability*, *71*(2), 747-762.
39. Farid, A. B., Fathy, E. M., Eldin, A. S., & Abd-Elmegid, L. A. (2021). Software defect prediction using hybrid model (CBIL) of convolutional neural network (CNN) and bidirectional long short-term memory (Bi-LSTM). *PeerJ Computer Science*, *7*, e739.