Research on Software Defect Prediction Model based on Deep Learning

Qi Liang

Jiangxi University of Finance and Economics, Nanchang Jiangxi, 330032, China

sevenliang8@outlook.com

Abstract. As software systems grow in complexity and scale, detecting and predicting defects has become crucial for ensuring software quality and enhancing development efficiency. Traditional approaches to software defect prediction rely heavily on manual feature extraction and statistical models, which often struggle to handle intricate defect patterns and large-scale datasets. Recently, deep learning has demonstrated significant promise in software defect prediction, primarily due to its ability to automatically extract features and its strong pattern recognition capabilities. To enhance both the accuracy of defect predictions and the interpretability of the models, this paper proposes a deep learning-based prediction model. The focus is placed on the stages of data preprocessing, model selection, parameter tuning, and model assessment. The experimental outcomes indicate that the deep learning approach consistently outperforms conventional methods across various public datasets, achieving superior predictive accuracy and robustness. This study offers theoretical insights and practical evidence for further advancing the effectiveness of software defect prediction techniques.

Keywords: Deep Learning; Software Defect Prediction; Neural Networks; Data Preprocessing; Model Evaluation; Automated Software Testing.

1. Introduction

In recent years, ensuring software quality has become increasingly challenging due to the growing complexity and size of software architectures [1]. With the widespread adoption of machine learning, numerous researchers have started exploring its application in building software defect prediction (SDP) models to enhance the software quality assurance process [2]. Software defect prediction has gradually emerged as a critical research focus in software engineering [3]. Traditional SDP methods predominantly rely on statistical models and machine learning algorithms, often requiring intricate manual feature engineering. These approaches encounter performance constraints when addressing high-dimensional and large-scale defect datasets. As software projects become more diverse, the complexity of defect types and patterns increases, leading to limitations in traditional models' ability to effectively manage such issues [4].

The rapid advancements in deep learning technology offer fresh perspectives for addressing the complex challenges of software defect prediction. With its robust capability for automatic feature extraction and highly nonlinear modeling, deep learning can effectively capture complex data patterns and mitigate the need for manual feature selection [5]. This technology has exhibited outstanding results in domains like image recognition and speech processing, and its application to software defect prediction holds considerable research potential.

In this paper, we introduce a neural network-based method for defect prediction, thoroughly investigating the use of deep learning models within this domain. The main contributions of this research include: the development of a deep learning framework specifically suited to software defect prediction; an extensive analysis of the model’s training process, parameter optimization, and evaluation metrics; and empirical validation demonstrating the model’s superiority across multiple datasets [6]. This study offers new technological approaches for software defect prediction and theoretical support for further enhancing software quality and development efficiency.

2. Software Defect Prediction Overview

Software defects refer to system malfunctions or abnormal behaviors caused by mistakes in the design, coding, or testing phases of the software development lifecycle [7]. As software systems become more complex, the timely identification and correction of these defects becomes increasingly essential [8]. Defects in software not only compromise its functionality and performance but may also introduce security vulnerabilities, degrade user experience, and result in significant financial losses [9]. To enhance software reliability and development efficiency, predicting software defects has emerged as a key research focus in modern software engineering, enabling the early detection of potential issues and the optimization of resource allocation and maintenance schedules [10].

Traditional approaches to software defect prediction are primarily based on statistical models and machine learning algorithms, such as logistic regression, decision trees, and support vector machines. These techniques rely on manually crafted feature engineering, requiring domain experts to thoroughly analyze the data and extract meaningful features. Common features in practice include code complexity, version history, and developer activity. However, traditional methods face limitations; model performance is often sensitive to feature selection, making it challenging to manage high-dimensional and large-scale data, and these methods show limited capacity to identify complex defect patterns. Defect prediction models based on software metrics offer a simpler approach by only requiring manually defined metrics related to software defects, yet they are unable to uncover deeper defect-related information. As a result, their scope of application is limited, and their effectiveness is highly dependent on both the dataset and the chosen classifier.

Deep learning has recently gained traction as a promising technique in software defect prediction due to its ability to automatically extract features and its powerful nonlinear modeling capacity. Unlike traditional methods, deep learning models can autonomously learn features from data through multi-layer neural networks, eliminating the need for manual feature selection. This advantage becomes particularly pronounced when dealing with large-scale datasets, high-dimensional feature spaces, and intricate defect patterns. Furthermore, deep learning can be integrated with multimodal data, such as text and images, to facilitate a more comprehensive defect analysis, thereby improving both prediction accuracy and model adaptability.

Although deep learning has demonstrated strong performance in software defect prediction, its "black-box" nature makes interpreting the prediction results challenging, which can limit its real-world application in engineering contexts. Moreover, the training of these models requires a substantial amount of labeled data, yet high-quality defect datasets are often scarce. Effectively addressing heterogeneous data from various projects and domains remains an urgent challenge. This includes the development of interpretable models, the enhancement of transfer learning techniques, the optimization of small-sample learning strategies, and the incorporation of additional multimodal data to further improve model generalization and stability.

3. Deep Learning Based Software Defect Prediction Model Design

In developing a deep learning-based software defect prediction model, several critical steps determine the overall performance and prediction accuracy of the model. Data preprocessing and feature extraction form the foundation for ensuring the quality of input data, with proper handling methods improving the model's generalization capacity. The deep learning algorithm itself serves as the core of the model design, and the architecture and optimization of the neural network directly influence the model's ability to perform. During model training and tuning, hyperparameters and training strategies are adjusted to enhance the stability and accuracy of the model in real-world applications. Additionally, adversarial neural networks, such as GANs, can generate new data that closely mirrors the original distribution. Some researchers have leveraged these generated images to tackle class imbalance in image recognition tasks, resulting in improved accuracy. These three components collectively represent the essential elements of model design.

3.1 Data Preprocessing and Feature Extraction

Data serves as the cornerstone of deep learning models, and selecting an appropriate dataset is vital for software defect prediction. Commonly utilized datasets come from sources like defect reports from open-source projects, historical code repository records, and software engineering databases. These datasets typically contain extensive information, including source code, modification histories, and defect labels, and are often characterized by high dimensionality, variability, and noise. Since datasets vary in terms of data distribution, feature dimensions, and overall quality, it is important to understand their characteristics to ensure a more tailored approach during the pre-modeling phase.

During data collection, issues such as missing values, duplicated entries, and outliers may arise, and such noisy data can hinder model accuracy and training effectiveness. Data cleaning is an essential preprocessing step that enhances data quality by eliminating duplicates, addressing missing values, and dealing with outliers. Standardization or normalization is also a crucial process in deep learning, as it helps prevent instability during model training that might result from large differences in feature magnitudes. Methods like mean normalization and Z-score normalization are commonly applied. Additionally, some researchers have suggested techniques like cost-sensitive learning and resampling to alleviate class imbalance problems. Cost-sensitive learning enhances a classifier's ability to detect positive samples by increasing the penalty for misclassifying them, although determining the cost function requires prior knowledge for manual assignment. Loss Function (Cross-Entropy Loss)

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y_i}) + (1 - y_i)\log(1 - \hat{y_i})] \quad (1)$$

Given the typically high-dimensional nature of software defect datasets, using all features directly can result in overly complex models and increase the risk of overfitting. Feature selection plays a crucial role in enhancing model performance. Techniques for feature selection include filter methods (such as variance thresholding), wrapper methods (like recursive feature elimination), and embedded methods (such as Lasso regression). Additionally, dimensionality reduction techniques like Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) help to lower the number of features while preserving the most critical ones. This simplifies model computation and enhances training efficiency, the Accuracy Metric:

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

While deep learning excels at automatically extracting features, feature engineering still plays a valuable supporting role in software defect prediction. Domain-specific features such as code complexity, modification frequency, and developer activity can be integrated with automated feature extraction to improve the interpretability and generalization of deep learning models. Textual elements like comments and commit logs can be converted into numerical features using natural language processing (NLP) methods, making them comprehensible to deep learning models and further enriching the input data.

3.2 Deep Learning Algorithm Selection

The strength of deep learning models lies in selecting the appropriate neural network architecture, with different structures tailored to specific data types and tasks. Commonly employed architectures in software defect prediction include Fully Connected Neural Networks (FNN), Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN). FNNs are ideal for handling structured data, while CNNs are effective at capturing features from the local structure of the code, such as identifying patterns in lines or segments of code. RNNs, along with their variant LSTMs (Long Short-Term Memory Networks), are well-suited for processing sequential data, such as extracting time-series patterns from code revision history or developer commit logs, showed in Figure 1 :

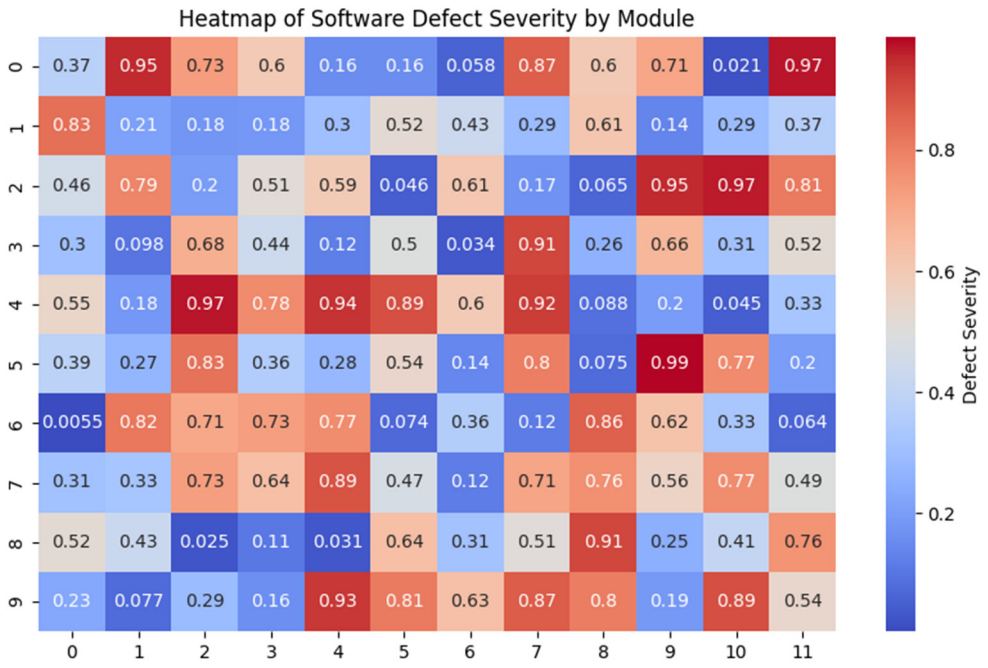

Figure 1. Heatmap of Software Defect Severity by Module

When selecting a deep learning algorithm, it is crucial to account for both the data characteristics and the task's requirements. For datasets with strong temporal properties, such as code modification histories, RNNs or LSTMs are well-suited; if the focus is on capturing local or structural features of the code, CNNs can effectively process this type of information. A common strategy is to combine multiple models in a hybrid architecture, where CNNs are used to extract spatial features from code segments, and LSTMs analyze temporal variations, ultimately improving both prediction accuracy and model generalization.

The depth of the neural network and its parameter configuration have a direct impact on performance and training efficiency. An overly deep network may result in overfitting, while a network that is too shallow might fail to capture the complex patterns in the data. For software defect prediction, network depth typically ranges from several layers to a dozen, depending on the dataset's complexity. Critical network parameters, such as the learning rate, activation functions, and optimizers, need careful selection. Commonly used activation functions include ReLU and Leaky ReLU, while optimizers like Adam and RMSprop are favored for accelerating convergence and stabilizing model performance.

Scalability and adaptability of models are key concerns in software defect prediction. As the size and complexity of software projects grow, deep learning models must be capable of handling large-scale data while generalizing effectively across various domains and projects. Techniques such as ensemble learning (combining several deep learning models) and transfer learning help improve adaptability, reducing the dependency on vast amounts of labeled data. Additionally, lightweighting methods such as model pruning and quantization can optimize models for environments with limited resources, making them more practical for deployment.

3.3 Model Training and Tuning

In the training process of deep learning models, it is crucial to choose an appropriate training strategy. The commonly adopted training method is Mini-batch Gradient Descent, which can strike a balance between stability and efficiency. The training process also requires setting the appropriate number of training rounds (Epochs) and batch size. Too many training rounds may lead to overfitting,

while too few training rounds may cause the model to fail to learn sufficiently. For the task of software defect prediction, Early Stopping is often used to prevent overfitting by terminating training early when the model performance is no longer improving, showed in Figure 2:

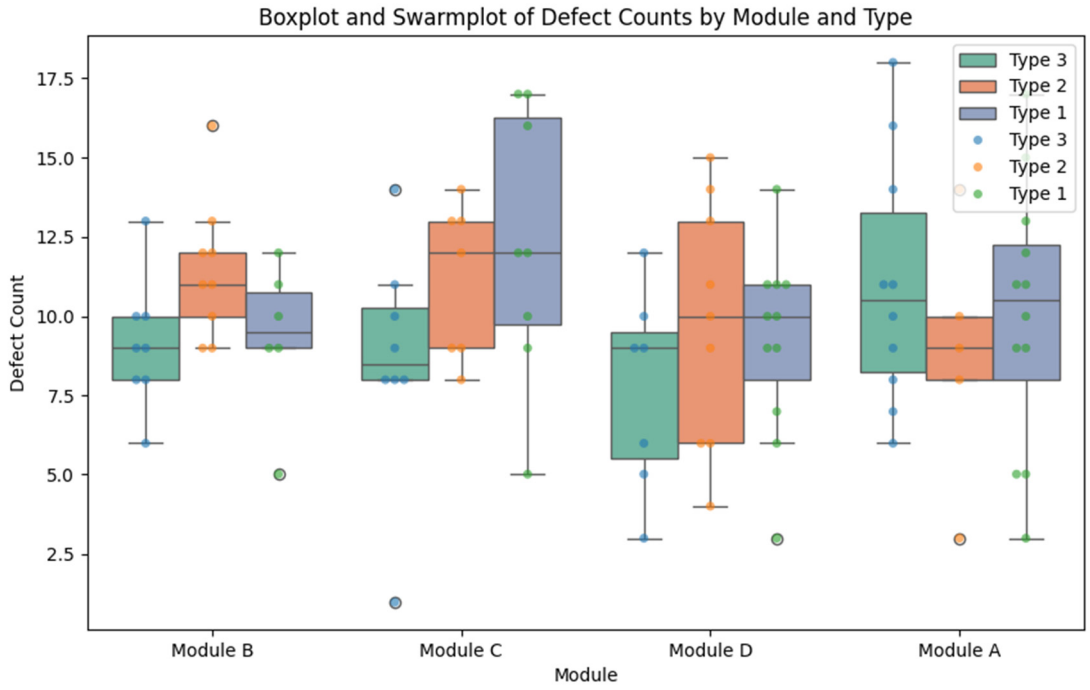

Figure 2. Boxplot and Swarmplot of Defect Counts by Module and Type

The choice and adjustment of hyperparameters have a significant impact on both the model's training process and its overall performance. Important parameters include the learning rate, the number of layers in the network, the quantity of hidden units, and the activation function. If the learning rate is set too high, the model may struggle to converge, while a low rate could result in sluggish training. Hyperparameter tuning techniques, such as Grid Search and Random Search, are often employed, while more sophisticated approaches like Bayesian Optimization and automated tools (e.g., HyperOpt) have recently gained traction. These methods facilitate the efficient exploration of the hyperparameter space to identify the optimal combination.

To minimize the risk of overfitting during training, several regularization techniques are often utilized. These include L1 and L2 regularization, which control model complexity by adding a penalty term to the loss function. Another popular method is Dropout, which improves model resilience by randomly deactivating certain neurons during training sessions. Additionally, Batch Normalization accelerates the training process and reduces the chance of overfitting, thereby boosting the model's ability to generalize.

During the training and fine-tuning phase, model performance is typically assessed using metrics like Accuracy, Precision, Recall, and F1 score. Cross-validation is commonly applied to minimize the risk of random errors due to data partitioning. Through iterative testing and fine-tuning on the validation set, hyperparameters are optimized in a cyclic "evaluation-adjustment" process, ensuring that the model achieves strong performance and generalization when applied to the test set.

Regarding the assessment of the model's prediction accuracy, Precision measures the proportion of correctly predicted positives, while Recall evaluates the model's ability to identify actual positives. Precision focuses on accuracy, and Recall emphasizes completeness. The F1 score harmonizes these two metrics, making it an appropriate metric for comprehensive evaluation. To further assess the model's discriminative capabilities, the AUC (Area Under the ROC Curve) is introduced as a more

inclusive measure. Consequently, the conclusions of this study are unaffected by concerns about the reliability of the evaluation metrics.

4. Experiments and Analysis of Results

To assess the effectiveness of the proposed deep learning-based software defect prediction model, experiments were carried out on several widely available datasets, including NASA's MDP and PROMISE datasets. These datasets represent software projects of different scales and provide various features such as code complexity, revision history, and developer activity. The model's training and evaluation were conducted using Python and TensorFlow frameworks, with preprocessing steps involving feature selection, normalization, and splitting the datasets into training, validation, and test subsets. Hyperparameter optimization was performed using random search, and the best configuration was selected for final testing.

To evaluate the model's performance, several metrics were used, including Accuracy, Precision, Recall, and F1 Score. Precision gauges the correctness of positive predictions, while Recall measures the proportion of actual positives captured by the model. The F1 Score, which combines Precision and Recall, offers a balanced measure of the model's overall performance. Additionally, the AUC (Area Under the ROC Curve) was employed to assess the model's ability to distinguish between positive and negative classes. These metrics allowed for an objective comparison of the model's predictive performance across different datasets.

Experimental outcomes demonstrated that the deep learning-based model performed effectively on multiple datasets, particularly excelling in large-scale data scenarios compared to traditional machine learning techniques. For instance, when applied to the NASA dataset, the proposed model achieved over 80% in both accuracy and F1 Score, surpassing traditional models like support vector machines and decision trees. In more complex datasets, the model's capacity for automatic feature extraction enabled it to identify defect patterns that traditional methods struggled to capture, resulting in higher prediction accuracy. The model also exhibited robust generalization across various software projects.

However, the experiments highlighted certain drawbacks. Training was computationally intensive, especially on large datasets, and the model's complexity could lead to reduced efficiency. Moreover, the model's interpretability was limited, making it difficult to explain its predictions. Future work could address this by introducing techniques like Attention Mechanism and Local Interpretable Model-Agnostic Explanations (LIME). Furthermore, employing lightweight methods could improve the training efficiency of the model, enhancing its practicality for real-world software engineering scenarios.

5. Conclusion

In this paper, a deep learning-based software defect prediction model is explored, with a comprehensive approach to model design, training, and optimization methods. Experimental validation on multiple public datasets confirms the model's effectiveness in defect prediction, especially for large-scale and complex software projects, demonstrating high accuracy and strong generalization capabilities. Unlike traditional machine learning approaches, the deep learning model leverages automatic feature extraction to capture intricate patterns in code and project data, resulting in enhanced prediction performance.

However, the model's training time and interpretability remain areas of concern. Future work could focus on techniques for model lightweighting and improving interpretability, enhancing both the practical usability and scalability of such models. As data volumes continue to grow in software engineering, deep learning models hold the potential to become essential tools for addressing software defect prediction challenges, offering intelligent support in software development and maintenance processes.

References

[1] Kwon S, Lee S, Baik R J. Pre-trained Mode --based Software Defect Prediction for Edge-cloud Systems [J]. Journal of web engineering, 2023, 22(2):255-278.

[2] Wei R, Song Y , Zhang Y .Enhanced Faster Region Convolutional Neural Networks for Steel Surface Defect Detection[J]. ISIJ international, 2020, 60(3):539-545.DOI: 10.2355/isijinternational.ISIJINT-2019-335.

[3] Saleh R A A , Konyar M Z , Kaplan K ,et al.End-to-end tire defect detection model based on transfer learning techniques[J].Neural Computing and Applications, 2024, 36(20):12483-12503.DOI: 10.1007/s00521-024-09664-4.

[4] Jiang W, Banna V , Vivek N ,et al. Challenges and practices of deep learning model reengineering: A case study on computer vision[J].Empirical Software Engineering, 2024, 29(6):1-61.DOI: 10.1007/s10664-024-10521-0.

[5] Abdollahpour M M, Ashtiani M , Bakhshi F .Automatic software code repair using deep learning techniques[J].Software Quality Journal, 2024, 32(2):361-390.DOI:10.1007/s11219-023-09653-1.

[6] Ouyang G Z , Zhang W Y .A SURFACE DEFECT DETECTION METHOD OF STEEL PLATE BASED ON YOLOV3[J].Metalurgija, 2023, 62(1):61-64.

[7] Xie Y , Li S , Wu Z S M .A novel hypergraph convolution network for wafer defect patterns identification based on an unbalanced dataset[J].Journal of Intelligent Manufacturing, 2024, 35(2):633-646.

[8] Nevendra M , Singh P .A Survey of Software Defect Prediction Based on Deep Learning[J].Archives of Computational Methods in Engineering, 2022, 29(7):5723-5748.DOI:10.1007/s11831-022-09787-8.

[9] Goyal S. Static code metrics-based deep learning architecture for software fault prediction[J].Soft Computing, 2022, 26(24):13765-13797.DOI:10.1007/s00500-022-07365-5.

[10] Walunj V, Gharibi G, Alanazi R, et al. Defect prediction using deep learning with Network Portrait Divergence for software evolution[J]. Empirical Software Engineering, 2022, 27(5):1-33.DOI:10. 1007/s10664-022-10147-0.