JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Journal homepage: https://jurnal.stkippgritulungagung.ac.id/index.php/jipi
ISSN: 2540-8984
Vol. 10, No. 3, September 2025, Pp. 2046-2055

OPTIMIZATION OF SOFTWARE DEFECT PREDICTION USING CNN AND ADABOOST: ANALYSIS AND EVALUATION (JURNAL ILMIAH PENELITIAN DAN PEMBELAJARAN INFORMATIKA)

Muhammad Abdul Basit *[1)], Arief Setyanto [2)], Tonny Hidayat [3)]
1. S2 Information Technology, University of Amikom Yogyakarta, Indonesia
2. S2 Information Technology, University of Amikom Yogyakarta, Indonesia
3. S2 Information Technology, University of Amikom Yogyakarta, Indonesia

Article Info
KeyWords: CNN, AdaBoost, SMOTE Tomek, SDP

Article history:
Received 12 Agustus 2024
Revised 20 September 2024
Accepted 3 Oktober 2024
Available online 1 September 2025

DOI :
https://doi.org/10.29100/jipi.v10i3.6405

* Corresponding author.
Corresponding Author
E-mail address:
muhabdulbasit21@gmail.com

ABSTRACT
This study focuses on enhancing software defect prediction (SDP) by integrating Convolutional Neural Networks (CNN) with the AdaBoost algorithm. The PROMISE dataset was employed in this research, and data balancing was achieved using the SMOTE Tomek technique. With the help of AdaBoost, we were able to increase the prediction accuracy after building a complex CNN model to extract features from the dataset. The AdaBoost model's hyperparameters were fine-tuned using GridSearchCV to find the best values for enhanced model performance. For the studies, we used StandardScaler to normalize the data after splitting it into training and testing groups with an 80:20 ratio. The experimental results show that compared to the baseline method, SDP accuracy significantly improves when CNN, AdaBoost, and hyperparameter tuning on AdaBoost using GridSearchCV are used together. Accuracy, precision, recall, F1 score, MCC, and AUC were some of the measures used to assess the model's performance.

I. INTRODUCTION

In the rapidly evolving software industry, ensuring high software quality is essential. Software defects can lead to significant issues, including financial losses, wasted time, and a damaged reputation. Therefore, early detection of software defects is crucial to mitigate these problems.

Previous research by Begum et al. has explored various strategies to enhance the accuracy of Software Defect Prediction (SDP). One such approach is the Lightweight Convolutional Neural Network (LCNN) architecture, which employs Explainable AI (XAI) techniques like LIME and SHAP to improve model interpretability [1]. This allows for a clearer understanding of model decisions and highlights key features that influence defect predictions.

However, despite these advancements, existing methods still face challenges in balancing accuracy, interpretability, and handling imbalanced datasets. Recent studies, such as the one by Chen et al., introduced a more sophisticated framework for software defect prediction using Nested-Stacking combined with heterogeneous feature selection. This approach has been shown to improve prediction accuracy and effectively address data imbalance issues on datasets like PROMISE and Kamei [2].

Moreover, research by Nasraldeen et al. demonstrated that integrating SMOTE Tomek with CNN and GRU can significantly enhance defect prediction accuracy, particularly when dealing with imbalanced datasets. While promising, these techniques still struggle with overfitting and require careful hyperparameter optimization [3].

In this study, we propose to address these limitations by combining CNN with the AdaBoost algorithm to improve SDP accuracy. Our approach leverages the feature extraction capabilities of CNN along with the robust performance enhancement provided by AdaBoost to reduce overfitting. Additionally, we employ SMOTE Tomek to balance the PROMISE dataset prior to training, ensuring that the model remains unbiased towards the majority class.

This research offers a unique contribution by demonstrating how the combination of CNN and AdaBoost can overcome the limitations of previous approaches. We also explore the effectiveness of using GridSearchCV for hyperparameter tuning to optimize the performance of both CNN and AdaBoost, thereby addressing the gaps in existing research related to model tuning and optimization.

The primary gap this study addresses is the lack of a comprehensive approach that effectively balances data,

optimizes model hyperparameters, and improves both accuracy and interpretability in SDP. By focusing on these aspects, our research not only contributes to the academic field but also provides practical solutions that can be applied in the software industry to enhance product quality.

II. RESEARCH METHODOLOGY

In this study, the following steps were taken to predict software defects:

A. Data Collection

Data was gathered through various sources, including platforms like the Kaggle website. These sources provided the essential information regarding software metrics needed for SDP analysis. The dataset utilized in this study is ant-1.7, which includes various software code metrics and defect labels.

B. Data Preprocessing

The data was cleaned and prepared for usage by going through a number of preparatory stages. The following preprocessing steps were applied [2]:

1) *Data Cleaning*
   To deal with missing data, we either removed it or used the feature's median or mean to impute the missing values.
2) *Normalization*
   The data was normalized to ensure consistency in scale. The normalization was performed using the StandardScaler from sklearn, as formulated below:

$$X_{norm} = \frac{X - \mu}{\sigma} \quad (1)$$

   Where X is the feature, μ is the mean, and σ is the standard deviation.
3) *Balancing*
   The SMOTE Tomek technique was employed to balance the class distribution within the training data. SMOTE combines oversampling with the Tomek Links undersampling method to address class imbalance in the dataset. The SMOTE formula is as follows:

$$x_{new} = x_{minority} + \lambda \cdot (x_{nearest} - x_{minority}) \quad (2)$$

   Where x_{new} is the synthetic sample, $x_{minority}$ is the minority class sample, $x_{nearest}$ is the nearest neighbor, and λ is a random number between 0 and 1.
4) *Data Split*
   There are two distinct categories of pre-processed data: training data and testing data. The data was divided in half, with 80% going into training and 20% into testing. This data division technique made use of sklearn's train_test_split function.

C. Justification for Method Selection

The choice of CNN, AdaBoost, and SMOTE Tomek is grounded in their proven effectiveness in addressing key challenges in software defect prediction:

1) *CNN*
   CNN was selected for its ability to automatically extract and learn complex features from the input data, which is crucial for capturing intricate patterns within software metrics. CNN's hierarchical structure makes it ideal for modeling spatial relationships in data, similar to its use in image processing but adapted here for software defect prediction.
2) *Adaboost*
   AdaBoost was chosen for its strength in boosting weak learners, thereby improving model accuracy and reducing overfitting. It combines multiple weak classifiers to form a strong classifier, making it particularly effective in refining the predictions generated by the CNN model.
3) *SMOTE Tomek*
   SMOTE Tomek was employed to tackle the class imbalance issue that commonly affects SDP datasets. By generating synthetic samples for the minority class and removing borderline examples from the

majority class, this technique ensures a balanced dataset, which is crucial for training robust and unbiased models.

D. *Convolutional Neural Network (CNN)*

1) *Overview*

Convolutional Neural Networks (CNNs) were originally designed to process data with a grid-like arrangement, such as images, which inherently have a two-dimensional structure. However, CNNs have been successfully adapted for various types of non-image data, including speech recognition and natural language processing. These applications demonstrate the flexibility of CNNs in handling different types of structured data by transforming them into a format suitable for convolutional processing.

In this study, the CNN architecture is used to process software metrics data, which is typically tabular and does not naturally have a grid structure like images. To adapt this tabular data for CNN processing, the data is transformed into a two-dimensional grid format, where each feature in the dataset corresponds to a "pixel" or unit in this grid. This transformation allows the CNN to apply its convolutional layers effectively, extracting complex patterns and relationships between the features.

The CNN model used in this research includes several key components, as illustrated in Figure 1 [2].

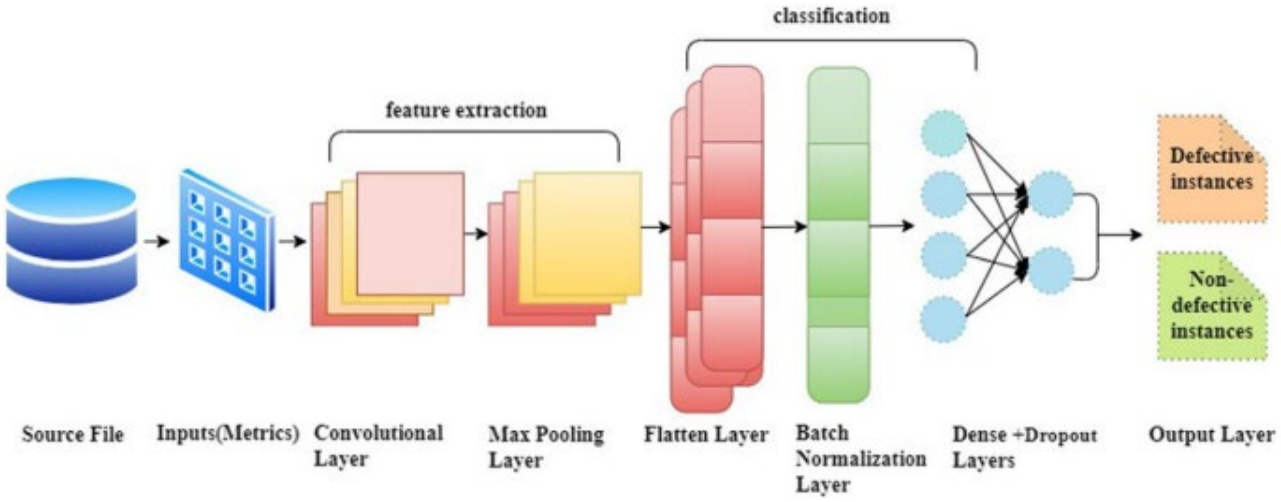

Figure. 1. CNN model for SDP

The architecture consists of convolutional, batch normalization, and max-pooling layers, which make up the network's hidden layers. The max-pooling layers reduce the dimensionality of the feature space, while the convolutional layers calculate the weights for the next layer based on specific filter and kernel parameters. Batch normalization is applied to mitigate the impact of varying input distributions across mini-batches, thereby enhancing the training process. Activation functions are employed to ensure that the CNN model trains accurately and efficiently. Among the most commonly used activation functions in CNNs are Sigmoid, ReLU, and Tanh. In this model, two activation functions are used: Sigmoid for the output layer and ReLU for the input and hidden layers, as shown in the following equations.

$$h_i^m = \text{ReLU}(W_i^{m-1} \times V_i^{m-1} + b_i^{m-1}) \quad (3)$$

Where h_i^m represents the convolutional layer, W_i^{m-1} represents the neuron weights, V_i^{m-1} represents the nodes, and b_i^{m-1} represents the bias layer.

$$S(x) = \frac{1}{1 + e^{-\sum_k W_i x_i + b}} \quad (4)$$

Where x_i is the input, W_i is the input weight, e is Euler's number (e =2.718...), and b is the bias.

2) *Complex CNN*

For more complicated data management, there is an improved variant of the classic CNN called the

Complex CNN. Its capacity to extract useful characteristics from data is enhanced by the incorporation of several more layers and features. Here are some improvements:

a) *More Convolutional Layers*
Complex CNN utilizes more convolutional layers than the basic CNN, enabling deeper and more complex feature extraction.

b) *Diverse Pooling Layers*
In addition to max pooling, Complex CNN can use average pooling to reduce data dimensionality and prevent overfitting.

c) *Intensive Batch Normalization*
Batch normalization is applied after each convolutional layer to stabilize input distributions and speed up training.

d) *Extensive Dropout*
Dropout is extensively used after each convolutional and dense layer to prevent overfitting.

e) *Varied Activation Functions*
In addition to ReLU and Sigmoid, Complex CNN can use other activation functions like Leaky ReLU or ELU to address different data issues.

f) *Complex Optimization*
Complex CNN uses advanced optimizers such as Adam or RMSprop and regularization techniques like L2 to improve model performance.

g) *Use of Residual Connections*
Certain Complex CNNs incorporate residual connections, inspired by ResNet, to mitigate the vanishing gradient issue and facilitate deeper learning within the model.

E. AdaBoost

After features are extracted using CNN, AdaBoost is used to enhance model performance and reduce overfitting. The best parameters for AdaBoost are determined using GridSearchCV with the following parameter grid: [3][4]

$$param_grid = \{'n\_estimators': [50,100,200], 'learning\_rate': [0.01,0.1,1.0]\} \quad (5)$$

AdaBoost is implemented with the formula:

$$F_m(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m h_m(x)\right) \quad (6)$$

Where $F_m(x)$ is the final model, α_m is the weight of m-th model, and $h_m(x)$ is the mmm-th predictive model.

F. Parameter Optimization

The parameters for AdaBoost, such as n_estimators and learning_rate, were fine-tuned using GridSearchCV. This allowed for systematic exploration of various parameter combinations to identify the configuration that maximized model performance while minimizing overfitting. The best parameters were then used to train the final AdaBoost model on the features extracted by the CNN.

G. Model Evaluation

Precision, accuracy, recall, F1 score, MCC, and AUC are some of the measures used to evaluate the model. You can not evaluate the model's performance in forecasting software problems without these measures. The steps of the study procedure are laid forth in Table I.

TABLE I
RESEARCH FLOW

Step	Activities	Description
Data Collection	Data Retrieval	Collecting dataset from Kaggle.
Pre-Processing	Oversampling	Using SMOTE to address data imbalance.
	Split Data	Splitting data into 80% training, 10% validation, and 10% testing.
Machine Learning Pipeline	Convolutional Neural Network (CNN)	Using CNN to extract complex features from software data.
	AdaBoost	Using AdaBoost to improve CNN model performance and reduce overfitting. Determining the best parameters for AdaBoost using GridSearchCV.
Evaluation	Model Evalution	Evaluating model performance using metrics such as accuracy, precision, recall, F1-score, MCC, and AUC.
Results Evaluation and Analysis	Analysis of Evaluation Results	Analyzing evaluation results to assess the effectiveness of the model in predicting software defects.
Discussion and Conclusion	Discussion and Conclusion	Discussing and drawing conclusions from research results.

III. RESULTS AND DISCUSSION

A. Dataset

The PROMISE dataset was used to test the SDP model. Table II provides a description of the selected dataset, the "ant" project version 1.7, with 745 samples and a defect rate of 22.28%. This dataset consists of various software metrics listed in Table III, which are used to train and test the CNN model used in this research. The use of this dataset allows for evaluating the accuracy and effectiveness of the model in identifying software defects.

TABLE II
DESCRIPTION OF PROMISE DATASET

Project Name	Project Version	# Number of Instances	Defect Rate %
ant	1.7	745	22.28%

TABLE III
LIST OF STATIC METRICS IN PROMISE DATASET

Attribute	Description
LOC (Lines of Code)	Number of lines of code in the code unit
CYCLO (Cyclomatic Complexity)	Cyclomatic complexity of the code, which reflects the number of independent paths in the code
CLASS	Class name in unit code

WMC (Weighted Methods per Class)	Total number of method complexities in the class
DIT (Depth of Inheritance Tree)	The depth of the inheritance tree for the class
NOC (Number of Children)	The number of derived classes of the class
CBO (Coupling Between Objects)	The number of other objects that are joined by the class
RFC (Response for a Class)	The number of methods that can be called by the class
LCOM (Lack of Cohesion in Methods)	Level of cohesion between in-class methods
BUG	Label indicating whether the code unit contains a defect (1) or not (0)

B. Data Split

After data is collected and prepared, it needs to be split into training, validation, and testing sets to ensure an objective evaluation of the model. When it comes to training, validation, and testing, the ratio is 80/20. The training set is used to train the model, the validation set to fine-tune the hyperparameters and avoid overfitting, and the test set to assess the model. For consistent scaling, data is normalized using StandardScaler, and data is divided using the train_test_split function from the sklearn package. This process ensures that the model is trained and evaluated effectively, providing an accurate estimate of its performance on unseen data.

C. Convolutional Neural Network (CNN) dan AdaBoost

Three models were tested: one baseline model (serving as a comparison point) and two proposed models (new models proposed to improve accuracy). Precision, Accuracy, Recall, F1 Score, MCC, and AUC were some of the assessment measures that exhibited diversity in the test findings. Table IV below displays the outcomes.

TABLE IV
LIST OF TEST MODEL

Test	Structure	Description
I	CNN AdaBoost with SMOTE Tomek	Baseline Model
II	Complex CNN AdaBoost with SMOTE Tomek	Proposed Model
III	More Complex CNN AdaBoost with SMOTE Tomek	Proposed Model

1) First Test (Baseline Model)

Using SMOTE Tomek and the CNN + AdaBoost model, the first test was conducted. After loading, the dataset was partitioned into labels and features. A training set and a testing set were subsequently created from the data. The training set was subjected to the SMOTE Tomek approach in order to rectify the data imbalance. In order to get features out of the balanced data, the CNN model was then constructed. The AdaBoost model was trained to make predictions using the features produced by CNN as input. To evaluate the model's performance, predictions were performed on the test set after training. Evaluation measures including recall, accuracy, precision, F1 score,

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Journal homepage: https://jurnal.stkippgritulungagung.ac.id/index.php/jipi
ISSN: 2540-8984
Vol. 10, No. 3, September 2025, Pp. 2046-2055

MCC, and AUC were computed. A Confusion Matrix was also used to display the outcomes of the predictions. Figure 2 and Table V display the findings.

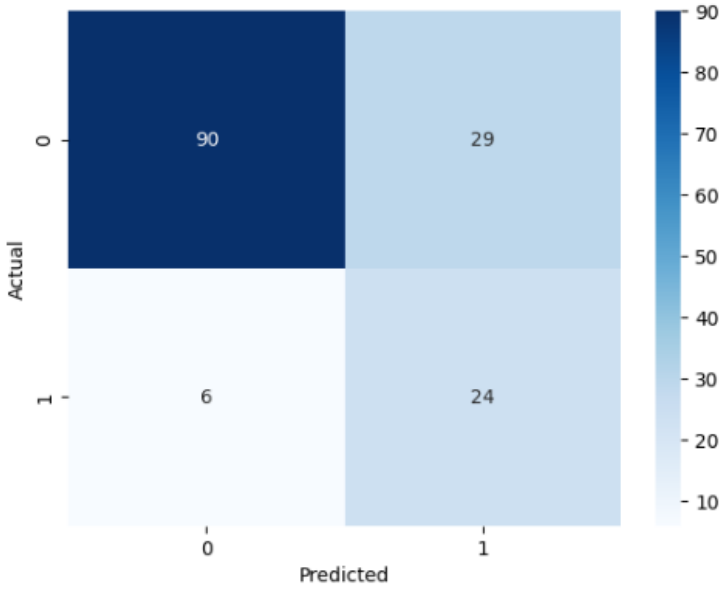

Figure. 2. Confusion Matrix for Test I

TABLE V
TEST RESULTS

Test Model	Accuracy	Precision	Recall	F1 Score	MCC	AUC
I	0.77	0.45	0.80	0.58	0.47	0.84

The Confusion Matrix revealed that the baseline model correctly classified 24 true positives and 90 true negatives, but it also produced 29 false positives and 6 false negatives. The accuracy of 0.77 indicates that the model correctly predicted 77% of all instances. However, the precision of 0.45 suggests that when the model predicted a defect, it was correct only 45% of the time, indicating a high false positive rate. The recall of 0.80 shows that the model successfully identified 80% of actual defects, but the low precision led to a lower F1 Score of 0.58. The MCC of 0.47 indicates a moderate level of agreement between the actual and predicted classifications, signaling room for improvement in the model's performance.

These metrics showed that while the baseline model achieved reasonable accuracy, there were notable weaknesses in precision and F1 Score, indicating that the model struggled with correctly identifying defective instances without sacrificing the accuracy of non-defective instances. This provided a clear motivation to explore more sophisticated models.

2) Second Test (Proposed Model)

The second test utilized the Complex CNN + AdaBoost model with SMOTE Tomek. The dataset was loaded and divided into features and labels, then balanced using SMOTE Tomek. Following data normalization using StandardScaler, the balanced dataset was divided into a training set and a testing set. Convolutional, pooling, and dropout layers were added to a more intricate CNN model in order to avoid overfitting. Following CNN training, the AdaBoost model was trained using the retrieved features. The hyperparameters of the AdaBoost model were optimized using GridSearchCV, and the model was then trained with the best-found values. Evaluation metrics such as accuracy, precision, recall, F1 score, MCC, and AUC were computed after making predictions on the test set. A Confusion Matrix was also used to display the outcomes of the predictions. Figure 3 and Table VI display the findings.

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Journal homepage: https://jurnal.stkippgritulungagung.ac.id/index.php/jipi
ISSN: 2540-8984
Vol. 10, No. 3, September 2025, Pp. 2046-2055

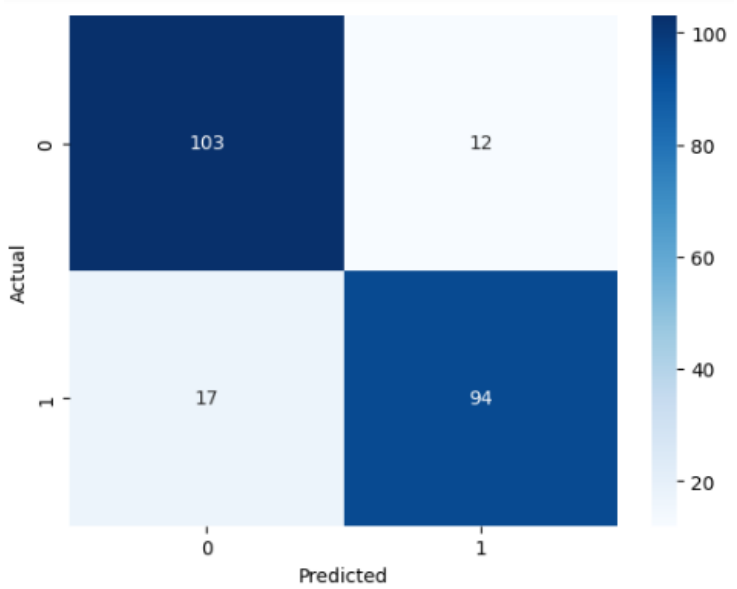

Figure. 3. Confusion Matrix for Test II

TABLE VI
TEST RESULTS

Test Model	Accuracy	Precision	Recall	F1 Score	MCC	AUC
II	0.87	0.89	0.85	0.87	0.74	0.89

The Confusion Matrix showed a significant improvement, with 94 true positives and 103 true negatives, and a reduction in false positives to 12 and false negatives to 17. The model's accuracy increased to 0.87, reflecting its ability to correctly classify 87% of the instances. The precision also improved markedly to 0.89, indicating that when the model predicted a defect, it was correct 89% of the time. The recall of 0.85 remained strong, showing that the model could detect the majority of defects, and the F1 Score of 0.87 confirmed a well-balanced model. The MCC of 0.74 highlights a strong positive correlation between the model's predictions and the actual outcomes, demonstrating the effectiveness of combining a complex CNN architecture with AdaBoost.

Compared to the baseline model, the Complex CNN + AdaBoost model showed a significant improvement across all evaluation metrics. The higher precision 0.89 and F1 Score 0.87 indicate that the model was better at correctly identifying defective instances without compromising the accuracy of non-defective classifications. The AUC of 0.89 further emphasizes the model's robustness in distinguishing between defective and non-defective instances, which is crucial in SDP tasks where false positives can lead to significant wasted resources. The MCC of 0.74 and the accuracy of 0.87 underscore the model's balanced and effective classification process.

When compared to previous studies, such as the work by Nasraldeen et al. (2023) that integrated CNN with GRU and SMOTE Tomek, the proposed model in this study achieved higher precision and recall, particularly in handling imbalanced datasets. This suggests that the combination of CNN with AdaBoost provides a more robust framework for SDP, especially in real-world scenarios where data imbalance is a significant challenge.

3) Third Test (Proposed Model)

The third test involved the More Complex CNN + AdaBoost model with SMOTE Tomek. Before being balanced using SMOTE Tomek, the dataset was imported and partitioned into features and labels. It was thereafter separated into training and testing sets and standardized with StandardScaler. To improve feature extraction capacity, a more sophisticated CNN model was built with more convolutional layers and batch normalization. After the CNN was trained, the AdaBoost model was trained using the characteristics it had extracted. In order to train the AdaBoost model with the most optimal hyperparameters, GridSearchCV was used to find them. On the test set, predictions were produced and the model's performance was evaluated using measures like recall, accuracy, precision, F1 score, MCC, and AUC. Figure 4 and Table VII show the outcomes of the prediction process as depicted using a Confusion Matrix.

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Journal homepage: https://jurnal.stkippgritulungagung.ac.id/index.php/jipi
ISSN: 2540-8984
Vol. 10, No. 3, September 2025, Pp. 2046-2055

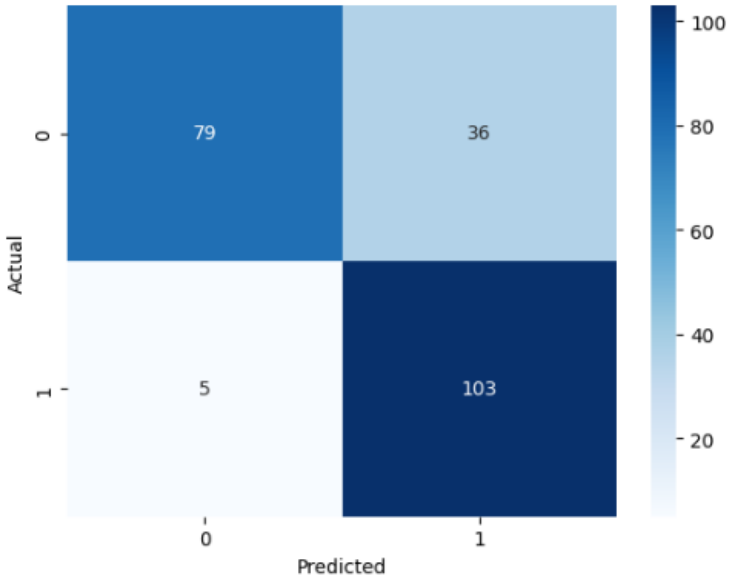

Figure. 4. Confusion Matrix for Test III

TABLE VII
TEST RESULTS

Test Model	Accuracy	Precision	Recall	F1 Score	MCC	AUC
III	0.82	0.74	0.95	0.83	0.66	0.91

The third test yielded 103 true positives and 79 true negatives, but also recorded an increase in false positives 36 and a decrease in false negatives 5. The model achieved an accuracy of 0.82, which, although slightly lower than the second test, still indicates strong performance. The precision of 0.74 was lower, suggesting more false positives, but the recall reached 0.95, the highest among the tests, indicating the model's exceptional ability to identify defects. The F1 Score of 0.83 reflected a good balance between precision and recall, and the MCC of 0.66, though slightly lower than the second test, still indicated a strong correlation between the predicted and actual classifications.

Although the third model introduced additional complexity, including more convolutional layers and batch normalization, the results indicated a trade-off. While the accuracy slightly decreased to 0.82, the recall significantly improved to 0.95, which is crucial in defect detection tasks where missing a defect can have severe consequences. This highlights the importance of selecting a model that aligns with the specific objectives of the SDP task. The high recall demonstrates the model's capability to identify almost all defective instances, making it suitable for scenarios where minimizing false negatives is critical. However, the slightly lower precision and accuracy suggest that this model may classify more non-defective instances as defective, which could lead to higher costs in terms of additional testing or revisions.

This model's performance aligns with findings from Chen et al. (2022), who also noted that more complex models can offer higher recall at the cost of precision in imbalanced datasets . The choice between this model and the second test model depends on whether the emphasis is on minimizing false negatives (favoring the third model) or achieving a balance between precision and recall (favoring the second model).

Table IX compares the operation of the three cases and contains the results of the tests that were run under various conditions.

TABLE IX
COMPARISON OF BASELINE MODEL AND PROPOSED MODELS

Model	Accuracy	Precision	Recall	F1 Score	MCC	AUC
CNN AdaBoost with SMOTE Tomek	0.77	0.45	0.80	0.58	0.47	0.84

Complex CNN AdaBoost with SMOTE Tomek	**0.87**	**0.89**	0.85	**0.87**	**0.74**	0.89
More Complex CNN AdaBoost with SMOTE Tomek	0.82	0.74	**0.95**	0.83	0.66	**0.91**

IV. CONCLUSION

The results indicate that both the Complex CNN AdaBoost with SMOTE Tomek (Proposed Model) and the More Complex CNN AdaBoost with SMOTE Tomek (Proposed Model) offer varied improvements compared to the CNN AdaBoost with SMOTE Tomek (Baseline Model). The Complex CNN AdaBoost model shows significant enhancements in Accuracy (0.87 vs. 0.77), Precision (0.89 vs. 0.45), and F1 Score (0.87 vs. 0.58), with an AUC increase (0.89 vs. 0.84), highlighting its superior ability to distinguish between defect and non-defect classes. The More Complex CNN AdaBoost model excels in Recall (0.95), making it highly effective in scenarios where detecting as many defects as possible is critical. Despite a slightly lower Precision (0.74 vs. 0.89) and Accuracy (0.82), its higher AUC (0.91) suggests a strong overall performance, particularly in differentiating between classes. The use of SMOTE Tomek reduces bias towards the majority class, enhancing the model's effectiveness in handling imbalanced datasets. If maximizing defect detection is the primary goal, the More Complex CNN AdaBoost model is preferable, while the Complex CNN AdaBoost model offers a better balance between Precision and Recall, making it more suitable for cases where both metrics are equally important. Overall, the study demonstrates that combining CNN and AdaBoost with SMOTE Tomek optimization significantly improves software defect prediction (SDP) performance.

REFERENCES

[1] Begum, M., Shuvo, M.H., Nasir, M.K., Hossain, A., Hossain, M.J., Ashraf, I., Uddin, J., Samad, M.A., "LCNN: Lightweight CNN Architecture for Software Defect Feature Identification Using Explainable AI," *IEEE Access*, vol. 2024, no. 1, pp. 123-134, 2024.
[2] Nasraldeen Alnor Adam Khleel, Károly Nehéz, "A Novel Approach for SDP Using CNN and GRU Based on SMOTE Tomek Method," *IEEE Access*, vol. 2023, pp. 1-10, 2023.
[3] Ramakrishna, M.T., Venkatesan, V.K., Izonin, I., Havryliuk, M., Bhat, C.R., "Homogeneous Adaboost Ensemble Machine Learning Algorithms with Reduced Entropy on Balanced Data," *Entropy*, vol. 25, no. 2, pp. 245, 2023.
[4] Ogunsanya, M., Isichei, D., Desai, M., "GridSearchCV Hyperparameter Tuning in Additive Manufacturing Processes," *Journal of Manufacturing Processes*, vol. 2023, no. 3, pp. 432-445, 2023.
[5] Hornyák, O., Iantovics, L.B., "AdaBoost Algorithm Could Lead to Weak Results for Data with Certain Characteristics," *Entropy*, vol. 2023, no. 5, pp. 789-800, 2023.
[6] Giray, G., et al., "On the Use of Deep Learning in SDP," *Journal of Systems and Software*, vol. 2023, no. 8, pp. 123-135, 2023.
[7] Pachouly, J., et al., "A Systematic Literature Review on SDP Using Artificial Intelligence: Datasets, Data Validation Methods, Approaches, and Tools," *Information and Software Technology*, vol. 2022, no. 7, pp. 567-579, 2022.
[8] Chen, L.-q., et al., "SDP Based on Nested-Stacking and Heterogeneous Feature Selection," *Expert Systems with Applications*, vol. 2022, no. 9, pp. 345-356, 2022.
[9] Uddin, M.N., Li, B., Ali, Z., Kefalas, P., Khan, I., Zada, I., "SDP Employing BiLSTM and BERT-based Semantic Feature," *Soft Computing*, vol. 2022, no. 7, pp. 1234-1245, 2022.
[10] Alazba, A., Aljamaan, H., "SDP Using Stacking Generalization of Optimized Tree-Based Ensembles," *Applied Sciences*, vol. 12, no. 9, pp. 4577, 2022.
[11] Al-Hadidi, T.N., Hasoon, S.O., "Software Defect Prediction Using Extreme Gradient Boosting (XGBoost) with Optimization Hyperparameter," *Al-Rafidain Journal of Computer Sciences and Mathematics*, vol. 18, no. 1, pp. 22-29, 2024.
[12] Yang, H., Li, M., "Software Defect Prediction Based on SMOTE-Tomek and XGBoost," *Proceedings of the International Conference on Bio-Inspired Computing: Theories and Applications*, pp. 12-31, 2021.
[13] Arora, R., Kaur, A., "Heterogeneous Fault Prediction Using Feature Selection and Supervised Learning Algorithms," *Vietnam Journal of Computer Science*, pp. 1-24, 2022.
[14] Dong, X., Yu, Z., Cao, W., Shi, Y., Ma, Q., "A Survey on Ensemble Learning," *Frontiers of Computer Science*, vol. 14, pp. 241-258, 2020.
[15] Iqbal, A., Aftab, S., Ullah, I., Bashir, M.S., Saeed, M.A., "A Feature Selection Based Ensemble Classification Framework for Software Defect Prediction," *International Journal of Modern Education and Computer Science*, vol. 11, no. 9, p. 54, 2019.
[16] Kumar, S., Behera, H.S., Nayak, J., Naik, B., "Bootstrap Aggregation Ensemble Learning-Based Reliable Approach for Software Defect Prediction by Using Characterized Code Feature," *Innovation in Systems and Software Engineering*, vol. 17, no. 4, pp. 355-379, 2021.
[17] Ibrahim, A.M., Abdelsalam, H., Taj-Eddin, I.A.T.F., "Software Defects Prediction at Method Level Using Ensemble Learning Techniques," *International Journal of Intelligent Computing and Information Sciences*, vol. 23, no. 2, pp. 28-49, 2023.
[18] Khuat, T.T., Le, M.H., "Evaluation of Sampling-Based Ensembles of Classifiers on Imbalanced Data for Software Defect Prediction Problems," *SN Computer Science*, vol. 1, no. 2, p. 108, 2020.
[19] Menzies, T., Krishna, R., Pryor, D., "The Promise Repository of Empirical Software Engineering Data," *North Carolina State University Department of Computer Science*, 2015.
[20] Zhang, T., Du, Q., Xu, J., Li, J., Li, X., "Software Defect Prediction and Localization with Attention-Based Models and Ensemble Learning," *Proceedings of the Asia-Pacific Software Engineering Conference (APSEC)*, vol. 2020, pp. 81-90, 2020.