Software Defect Prediction Using Neural Network Based SMOTE

Rizal Broer Bahaweres[1], Fajar Agustian[2], Irman Hermadi [3], Arif Imam Suroso[4], Yandra Arkeman[5]
[1,3]Computer Science Dept. IPB University
[1,2]Department of Informatics, Syarif Hidayatullah State Islamic University, Jakarta, Indonesia
[4]School of Business, IPB University, Bogor Indonesia
[5]Department of Agro-Industrial Tech., IPB University, Bogor, Indonesia
rizalbroer@computer.org, fajar.agustian17@mhs.uinjkt.ac.id, irmanhermadi@apps.ipb.ac.id, Arifimamsuroso@apps.ipb.ac.id, yandra.arkeman@gmail.com

***Abstract*—Software defect prediction is a practical approach to improve the quality and efficiency of time and costs for software testing by focusing on defect modules. The dataset of software defect prediction naturally has a class imbalance problem with very few defective modules compared to non-defective modules. This situation has a negative impact on the Neural Network, which can lead to overfitting and poor accuracy. Synthetic Minority Over-sampling Technique (SMOTE) is one of the popular techniques that can solve the problem of class imbalance. However, Neural Network and SMOTE both have hyperparameters which must be determined by the user before the modelling process. In this study, we applied the Neural Networks Based SMOTE, a combination of Neural Network and SMOTE with each hyperparameter of SMOTE and Neural Network that are optimized using random search to solve the class imbalance problem in the six NASA datasets. The results use a 5*5 cross-validation show that increases Bal by 25.48% and Recall by 45.99% compared to the original Neural Network. We also compare the performance of Neural Network-based SMOTE with "Traditional" Machine Learning-based SMOTE. The Neural Network-based SMOTE takes first place in the average rank.**

Index Terms*—*Software Defect Prediction*, *Class Imbalance*, *Synthetic Minority Over-sampling Technique*, *Neural Network

I. Introduction

As times evolve, the need for information technology is increasing. As software systems grow quickly they also become more complicated. Consequently, this increases costs due to the need for thorough testing processes. A survey indicated that 23% of this cost was spent on quality assurance and testing [1]. Software defect prediction provides a method to reduce the costs of testing. With software prediction, it can predict defect modules quickly with machine learning methods [2]. Once a defective module is predicted, it becomes easier to allocate limited resources by prioritizing specific defective modules for testing [3]. The purpose of predicting software defects is to determine which modules will be prioritized before testing, the next testing process can be done manually or automatically [4].

Previous studies reported, using classification algorithms such as Neural Network [1] for software defect prediction. The benefits of applying the Neural Network algorithm are its capacity to overcome nonlinear data, learn and capture features from data automatically so that it can produce more accurate predictions without the feature selection process [3]. However, the problem that is encountered in software defect prediction is the existence of a dataset that contains a class imbalance between the defective module and the non-defective module [5]. Neural Network that is trained using a dataset with class imbalance can lead to overfitting and poor performance [6]. Algorithms will tend to be biassed towards non-defective module, and in the extreme case, this will ignore defective module [2]. Another problem with neural networks is the determination of hyperparameters, such as the number of hidden layer, Neurons, and Learning Rate before the training process. Neural Network performance depends on proper hyperparameter settings [7].

Several studies introduce techniques to overcome class imbalances. There are algorithmic approaches and data approaches. In [1] proposed an algorithmic approach method for class imbalance problems using a cost-sensitive neural network. This method can improve the classification performance, but the results are not significant. They recommend using a data approach to solve the class imbalance problem. Research by [8] undersampled non-defect classes to balance training data. The use of undersampled results in reduced information needed by the model for the training process. The alternative is to use one popular oversampling method, SMOTE (Synthetic Minority Over Sampling) [9]. SMOTE works by adding a minority class with randomly created instances [9].

The use of SMOTE for class imbalance problems in software defect prediction has been widely used and has been proven to improve classification performance in several studies such as [2] [10]. SMOTE has parameter control which can be optimized, and they did not consider the optimization parameter of SMOTE. In [11] shows the advantages of SMOTE parameter optimization; the method is called SMOTUNED. SMOTUNED performance outperforms SMOTE significantly. However, in that study, they did not consider optimizing the hyperparameter of the classifier.

Based on the explanation above, we propose Neural Network based SMOTE, a combination of Neural Network and SMOTE with each hyperparameters that are optimized using random search . We optimized the number of oversampling ratios and the number of SMOTE neighbors, then we optimized the number of hidden layers, neurons, Learning Rate, and Drop-outs of Neural Network. We use bal to optimize, and recall as our evaluation metrics to compare the proposed model. We also compare the performance of Neural Network based SMOTE with ”traditional” machine learning based SMOTE. We use six Nasa Software project as our dataset.

The research questions in this study are as follows:

1) Can Neural Network based SMOTE improve performance of original Neural Network on software defect prediction?
2) How does the performance of the Neural Network based SMOTE compare to ”traditional” machine learning based SMOTE on software defect prediction?

We identify the scope of the problem in this study as:

1) The dataset used is 6 NASA PROMISE and TERA-PROMISE datasets that are public.
2) The tools used are python-3.7, sci-kit Learn [12], Tensorflow.Keras [13], and imblearn [14].

II. LITERATURE REVIEW

A. Related Search

TABLE I
RELATED SEARCH

Ref	Neural Network	SMOTE	NASA Dataset	Hyperparameter Optimization	Method
[3]	✓				DNN Regression
[15]			✓		Ensemble Oversampling with Many ML
[5]			✓		KMFOS with Many ML
[10]		✓	✓		Ensemble SMOTE with Many ML
[2]		✓	✓		SMOTE with Ensemble ML
[16]	✓		✓	✓	DNN
[1]	✓		✓		Cost-Sensitive NN
Author	✓	✓	✓	✓	SMOTE with NN

Table I describes the research related to the study. It is evident the studies reviewed apply different approaches to overcome the problem of class imbalance in software defect prediction. In some cases, the authors have applied several types of oversampling technique. The data approach is carried out by [15] by combining three techniques oversampling, ROS (Random Oversampling), MWMOTE (Majority Weighted Minority Oversampling Technique), and Fidos (Fuzzy-Based on Feature and Instance Recovery using Information decomposition). in [5], the authors applied Cluster-Based Oversampling noise filtering (CLNI). in [10], the authors applied SMOTE with Bagging and Naive Bayes techniques.

In [1], the author has used a cost-sensitive neural network with ABC algorithm, which can overcome class imbalances even though the results obtained are not significant. Deep learning techniques were applied in [3] to get the number of defect modules with DNN for regression. in [16], the authors implemented a deep neural network to classify the file level using static metrics.

From the above research, hyperparameter optimization was not explored in the imbalance problem of software defect prediction. We combine SMOTE and Neural Network with hyperparameter random search optimization.

B. Synthetic Minority Oversampling Technique (SMOTE)

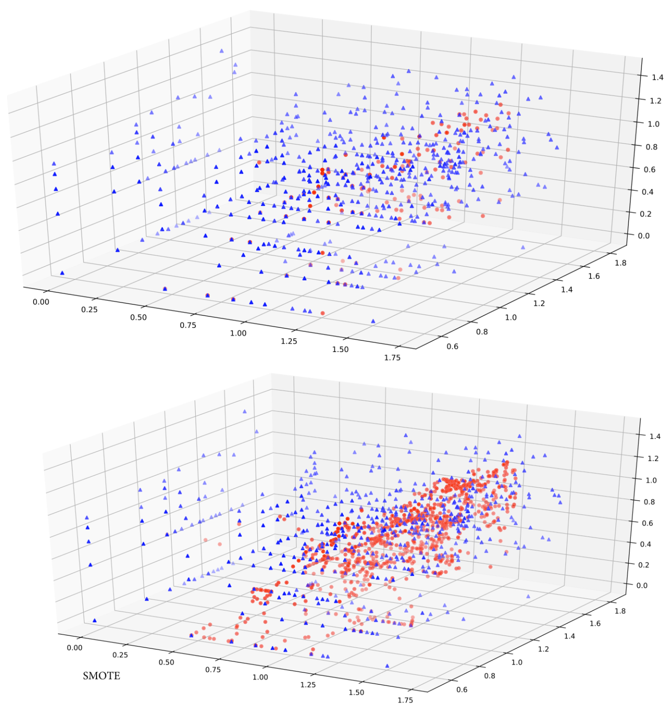

Fig. 1. PC4 SMOTE Distribution

SMOTE is an oversampling method that is often used to overcome the problem of class imbalance and was first introduced by Chawla, et al [9]. The SMOTE method used by Chawla [9] is a different version of SMOTE from the existing oversampling method. To handle the class imbalance, SMOTE method adds the amount of data to the free class with synthesis data (synthetic data). Synthesis data were obtained from k-NN (k-nearest neighbor).

SMOTE has 2 parameters that can be adjusted - ratio, and number of neighbors. The ratio parameter is defined as the number of major and minor classes and The number of neighbors is the number of closest neighbours to create syntactic data [14].

There are differences in the processes for producing synthetic data on a numerical and category scale. When using numerical data calculation, it is based on euclidian distance;

whereas for categorical data, the calculation uses mode values. [9]. In this study, the data used are numerical. Fig 1 is a sample distribution made by SMOTE. Synthetic instances are shown in red.

C. Neural Network (NN)

Neural Network is a type of computational approach or processor / computer model based on the human brain [17]. Neural Networks consist of many simple, connected processors called neurons [18]. Neural Networks consist of three layers: 1) the input layer (at least one), 2) hidden, and 3) an output [1]. The problem with Neural Networks in predicting software defects is the number of hyperparameters must be determined before training begins so that it is known that the performance of Neural Networks generalization depends on good hyperparameter settings [7].

The hyperparameter of a neural network can be optimized according to three factors: 1) hidden layer, 2) neurons, 3) learning rate. There are two techniques for Hyperparameter, Grid Search and Random Search. When using grid search we try each possible hyperparameter to optimize. The drawbacks with this method are the expensive running costs - as well, the process is time-consuming. An alternative method is a random search, we attempt random hyperparameter to optimize, that the result can reach near optimum [19].

D. Recall and Balance (bal)

A good method for overcoming class imbalance in predicting software defects is to classify defective and non- defective samples accurately. Recall and balance are used as a measure of performance in evaluating the methods used in [5].

Recall is a comparison of the number of samples that are predicted to be defective and true positive [5]. The performance of the method used is considered to be better when the recall value is greater.

$$Recall = \frac{TP}{TP + FN} \tag{1}$$

bal is the balance between recall and pf values. Pf is the ratio of the number of non-defect samples that are incorrectly predicted by the number of non-defect samples. The greater the value of bal, the better the results of the method.

$$bal = 1 - \frac{\sqrt{(1 - Recall)^2 + pf^2}}{\sqrt{2}} \tag{2}$$

III. RESEARCH METHODOLOGY

Fig 2 shows the methodology that will be used in this study. The first stage is collecting NASA datasets. After that, we applied the normalization process to each dataset. We divide the data into 5*5 cross-validation training, validation, and testing data. Training and validation data were used to search for SMOTE and Neural Network hyperparameters. The hyperparameter search process uses random search by bal optimization on the performance results of the neural network and SMOTE classifications, and hyperparameter optimization use data validation. Best hyperparameter will be tested using testing data. The evaluation performance that we will use is Bal and Recall. More explanation will be discussed in the next subsection.

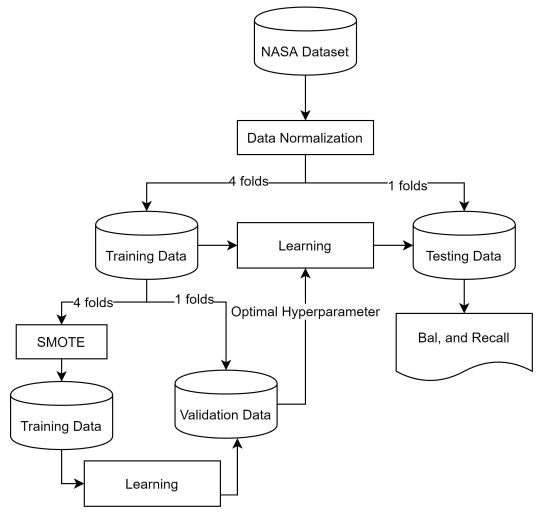

Fig. 2. Methodology

A. Dataset

TABLE II
DATASET

Dataset	# Minority	# Majority	Features	# Defect Rate (%)
JM1	2103	8777	22	19.33
KC1	326	1783	22	15.46
KC2	107	415	22	20.5
PC1	77	1032	22	6.94
PC3	134	943	38	12.44
PC4	178	1280	38	12.21

In this study, we used the NASA Promise and Tera-Promise dataset, which is a public dataset. The dataset selected has a class imbalance between the defective module and the non-defective module. TABLE II is a list of datasets that will be used for research. All features in the dataset will be used in the modelling process.

B. Min-Max Normalization

Data Normalization is a technique commonly used in pre-processing datasets. In this study, the normalization method used is min-max normalization [20]. This technique is used to improve performance and speed up the training process by changing the distribution scale to 0 and 1. We use min-max normalization because the distribution of the defect prediction software dataset is very variant.

C. Cross Validation

In the distribution of datasets, the technique we employ is 5*5 folds cross-validation. We divide one fold for testing data and the remaining folds for training data. Training data is used to find the optimal hyperparameter by dividing it into four

TABLE III
LIST OF SMOTE PARAMETER

Parameter	optimization Range	Desc
Neighbors	[1,10]	Number of neighbors
Ratio	[0.8,1]	Number of Ratio between major and minor instance after oversampling

TABLE IV
LIST OF NEURAL NETWORK HYPERPARAMETER

Parameter	optimization Range	Description
Learning Rate	[0.001, 0.01]	The initial learning rate used
Drop Out	[0.1,0.5]	The percentage dropping out units (both hidden and visible)
Neurons	[25,100]	The number of neurons each hidden layer
Hidden Layer	[1,5]	The number of hidden layer
Batch Size	[64]	Total number of training examples present in a single batch
Epoch	[15]	The total of entire dataset is passed forward and backward through the neural network

folds training data for hyperparameter optimization and one fold for data validation. The best hyperparameter will be tested by testing data.

We realize that the nature of cross-validation is that all folds can be training, validation, and testing data. However, for hyperparameter optimization, we use data that is different from the testing data, so that there will be no leak, and the testing data will remain unseen. Each fold will have a different hyperparameter.

D. SMOTE

After the data is divided into training data and validation data, train set will be applied SMOTE. Classes containing defective modules will be made synthetic instances so that it offsets the majority class, namely classes that contain non-defective modules. The tools we use are SMOTE imblearn. We need to emphasize only the training data that will be applied by SMOTE.

E. Neural Network Architecture

The Neural Network topology that we use is rectangular, fully connected. The activation function that will be used is ReLU (rectified linear unit) at the input and hidden layer. At the output layer, the activation function used is sigmoid with the aim of binary classification. Realizing that the optimal hyperparameter depends on the dataset used in the training data, we use different hyperparameters in each dataset.

F. Hyperparameter Optimization

As explained in the previous sub-section, the optimal neural network hyperparameter can differ depending on the dataset as well as the number of ratios and neighbours in SMOTE. We use the Random Search CV (Cross Validation) [19] to obtain the optimal hyperparameter. We use Bal as our measurement on hyperparameter optimization process, where the higher the true positive rate (recall) and the lower false alarm are, the higher the performance of the bal. The hyperparameter that we use is the best in the first fifty iterations.

Fig 3 shows the diagram activity of Hyperparameter Optimization of Neural Network and SMOTE. The Neural Network Hyperparameter that will be optimized can be seen in Table IV. We use fixed bath sizes and Epoch, which are 64 and 15, respectively. We apply Dropout on each layer to avoid overfitting in the training process.

The SMOTE parameter can be seen in Table III. The parameters we will optimize are the number of neighbors and the number of ratio.

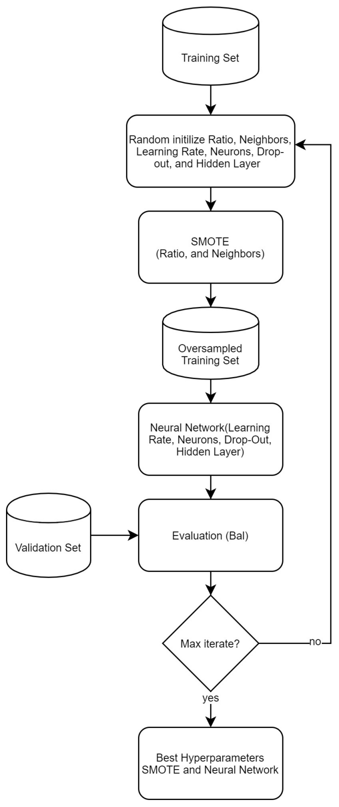
Training Set
Random initilize Ratio, Neighbors, Learning Rate, Neurons, Drop-out, and Hidden Layer
SMOTE (Ratio, and Neighbors)
Oversampled Training Set
Neural Network(Learning Rate, Neurons, Drop-Out, Hidden Layer)
Validation Set
Evaluation (Bal)
Max iterate?
no
yes
Best Hyperparameters SMOTE and Neural Network

Fig. 3. Hyperparameter Optimization Neural Network Based SMOTE

G. Predict and Classification Performance

After the model is tested, performance will be calculated using the balance and recall that have been mentioned in the literature study. The measurement is used to measure the prediction of software defects in unbalanced datasets. The higher the balance and recall values, the better the performance of the model. Recall is used because in predicting software defects, it is more important to predict modules that are defective and true than predicting modules that are not defective.

IV. RESULT AND DISCUSSION

We now answer our research questions:

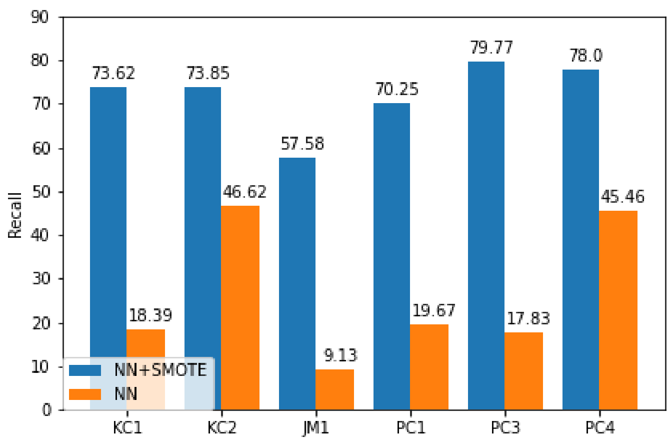

Fig. 4. Comparison of Recall Neural Network based SMOTE and Neural Network

A. *Can Neural Network based SMOTE improve performance of original Neural Network on software defect prediction?*

Our test is done by combining the neural network algorithm with the oversampling technique, namely SMOTE. Each hyperparameter of SMOTE and Neural Network is optimized using random search. The results are derived from the average bal and recall on the test set of each fold. each fold is tested with a different hyperparameter depending on the best hyperparameter selected during the hyperparameter optimization process. Hyperparameter is chosen based on the highest bal. in addition, we count recall.

These tests can be seen in Fig 4 shows the results of recall from various datasets. The recall results vary depending on the dataset. The NN algorithm with SMOTE significantly increases recall by 45.99%. Based on Fig 4, all datasets that implemented Neural Network based SMOTE have a higher recall compared to original Neural Network. Based on the graph, SMOTE has an influence on the Recall calculation on the Neural Network.

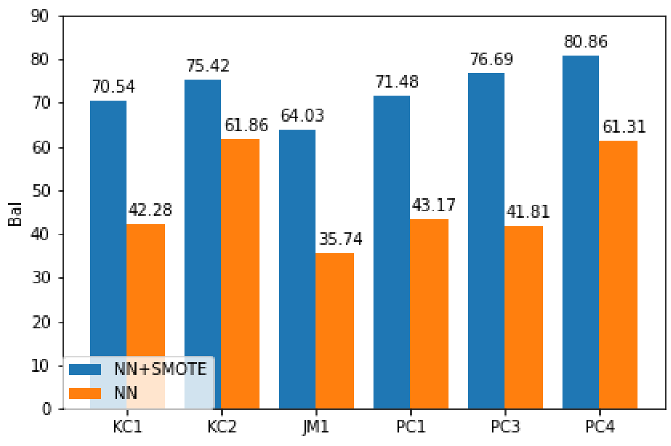

Fig. 5. Comparison of Bal Neural Network based SMOTE and Neural Network

TABLE V
LIST OF HYPERPARAMETER EACH ALGORITHM

Algo	Hyperparameter	optimization Range	Desc
RF	Estimators	[100,300]	The Number of tress in the forest.
	Max Depth	[5,10]	The maximum depth of the tree
KNN	Leaf Size	[1,100]	Leaf size passed to BallTree.
	Neighbors	[1,10]	
DT	Max Depth	[5,10]	The maximum depth of the tree
LR	C	[1,10]	Inverse of regularization strength
NB	-	-	-

TABLE VI
COMPARISON OF BAL ON NEURAL NETWORK BASED SMOTE AND TRADITIONAL MACHINE LEARNING BASED SMOTE

	KC1	KC2	JM1	PC1	PC3	PC4	Avg.Rank
NN	70.54	75.42	64.03	71.48	**76.69**	80.86	2.17
RF	**71.51**	72.27	64.79	69.5	72.33	**83.15**	2.33
KNN	69.89	71.8	63.44	**77.98**	72.07	78.63	3.83
DT	66.52	70.58	**65.65**	69.12	70.07	80.07	4
LR	69.91	**76.06**	64.87	67.71	73.29	80.58	2.67
NB	57.68	62.47	44.25	51.92	43.47	75.14	6

Fig 5 shows the results of bal. Just like recall, Neural Network based SMOTE can improve the performance of the model, Bal has increased by 25.48%. All datasets are tested outperform the original neural network without SMOTE. The higher the balance value, the higher the performance of a model. Based on the graph, the dataset that has good performance on the Neural Network is the PC4 dataset which has a bal of 80.86%. In contrast, JM1 has a lower balance and recall value below the average compared to other datasets.

In summary, Neural Network based SMOTE can improve the bal and recall of each dataset significantly.

B. *How does the performance of the Neural Network based SMOTE compare to "traditional" machine learning based SMOTE on software defect prediction?*

To answer this question, we compared the Neural Network based SMOTE model with five classification algorithms (Random Forest (RF), KNN, Logistic Regression (LR), Decision Tree (DT), Naive Bayes (NB) which are often used in research on software defect prediction. We apply the same method as the neural network process, apply SMOTE to all classification algorithms, and do hyperparameter optimization on each algorithm with bal optimization. Several hyperparameter options were adopted from [21]. each hyperparameter can be seen in Table V

Table VI is the result of bal from each dataset. All algorithm has applied by SMOTE. The results obtained are very variant; each algorithm has the best results on different datasets. Nothing is dominant, only the Random Forest algorithm that outperforms the two datasets, the other algorithm only surpasses one dataset each. To overcome these variant results, we calculate the average algorithm ranking for each dataset. The smaller the average ranking, the better the performance of the algorithm compared to other algorithms.

Based on the average ranking in each dataset, neural networks occupy the first position, then Random Forest, which has an average ranking that is not much different. The algorithm with the worst results is Naive Bayes, with the last position. Decision Tree and Logistic Regression show standard results.

We also count recall as an additional performance evaluation that can be seen in table VII. The results are very variant depending on each dataset, and no algorithm that dominates. We can see Naive Bayes recall on PC3 where the recall value is higher than other algorithm. However, we can see the performance of Naive Bayes on bal PC3 is very low, in the sense of a high false-positive rate. That is one of the reasons we optimize bal for hyperparameter optimization rather than recall.

TABLE VII
COMPARISON OF RECALL ON NEURAL NETWORK AND TRADITIONAL MACHINE LEARNING

	KC1	KC2	JM1	PC1	PC3	PC4
NN	**73.62**	73.85	57.58	70.25	79.77	78.00
RF	68.39	66.23	56.49	59.75	67.86	**81.38**
KNN	64.09	67.14	60.15	**75.33**	70.17	79.21
DT	57.66	63.46	**63.91**	61.00	64.81	76.29
LR	66.56	**74.68**	58.87	58.50	72.99	79.73
NB	41.08	47.49	21.35	32.58	**87.21**	70.21

In summary, we can see Neural Network based SMOTE on Avg. Rank bal occupies the first position. this shows that neural network based SMOTE can outperform the performance (bal) of other algorithms.

V. CONCLUSION AND FUTURE WORK

A. Conclusion

This paper proposed a Neural Network based SMOTE to improve neural network performance on class imbalance problem of software defect prediction. Each Hyperparameter of SMOTE and Neural Network are optimized using a random search to find the best combination of hyperparameters SMOTE and Neural Network. Best hyperparameter will be used for evaluation. The dataset used is six NASA datasets. Based on the results of the study using 5*5 cross validation, Neural Network-based SMOTE can increase significantly (bal by 25.48%, and recall 45.99%) compared to the original Neural Network without SMOTE. We also implement SMOTE to other traditional classification algorithms and compare to our proposed model, Neural Networks based SMOTE is first average rank on software defect prediction using NASA Dataset. However, each dataset has a different winning algorithm, and the differences are not significant.

There are things to consider, the hyperparameter optimization used in this study is random search, which allows an algorithm to not reach an optimal value in the optimization process.

B. Future Work

For further research, optimization algorithms such as genetic algorithms and Bayesian optimization can be used to avoid bias in the results of hyperparameter optimization on software defect prediction.

REFERENCES

[1] Ö. F. Arar and K. Ayan, “Software defect prediction using cost-sensitive neural network,” *Applied Soft Computing Journal*, 2015.

[2] H. Alsawalqah, H. Faris, I. A. B, and L. Alnemer, “Hybrid SMOTE-Ensemble Approach,” *Springer International Publishing*, vol. 1, no. April, 2017.

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

[4] R. B. Bahaweres, K. Zawawi, D. Khairani, and N. Hakiem, “Analysis of Statement Branch and Loop Coverage in Software Testing With Genetic Algorithm,” no. September, pp. 19–21, 2017.

[5] L. Gong, S. Jiang, and L. Jiang, “Tackling Class Imbalance Problem in Software Defect Prediction Through Cluster-Based Over-Sampling With Filtering,” *IEEE Access*, vol. 7, pp. 145725–145737, 2019.

[6] Q. Fan, Z. Wang, and D. Gao, “One-sided Dynamic Undersampling No-Propagation Neural Networks for imbalance problem,” *Engineering Applications of Artificial Intelligence*, 2016.

[7] R. S. Wahono, N. S. Herman, and S. Ahmad, “Neural network parameter optimization based on genetic algorithm for software defect prediction,” *Advanced Science Letters*, vol. 20, no. 10-12, pp. 1951–1955, 2014.

[8] T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, and Y. Jiang, “Implications of ceiling effects in defect predictors,” in *Proceedings - International Conference on Software Engineering*, 2008.

[9] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic Minority Over-sampling Technique,” *Journal Of Artificial Intelligence Research*, 2002.

[10] A. Saifudin, S. W. Hendric, B. Soewito, F. L. Gaol, E. Abdurachman, and Y. Heryadi, “Tackling Imbalanced Class on Cross-Project Defect Prediction Using Ensemble SMOTE,” *IOP Conference Series: Materials Science and Engineering*, vol. 662, no. 6, 2019.

[11] A. Agrawal and T. Menzies, “Is ”better data” better than ”better data miners”?: On the benefits of tuning SMOTE for defect prediction,” *Proceedings - International Conference on Software Engineering*, pp. 1050–1061, 2018.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[13] F. Chollet, “Keras.” https://github.com/fchollet/keras, 2015.

[14] G. Lemaître, F. Nogueira, and C. K. Aridas, “Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning,” *Journal of Machine Learning Research*, vol. 18, no. 17, pp. 1–5, 2017.

[15] S. Huda, K. Liu, M. Abdelrazek, A. Ibrahim, S. Alyahya, H. Al-Dossari, and S. Ahmad, “An Ensemble Oversampling Model for Class Imbalance Problem in Software Defect Prediction,” *IEEE Access*, vol. 6, pp. 24184–24195, 2018.

[16] R. Ferenc, D. Bán, T. Grósz, and T. Gyimóthy, “Deep learning in static, metric-based bug prediction,” *Array*, vol. 6, no. February, p. 100021, 2020.

[17] V. Vashisht, M. Lal, and G. S. Sureshchandar, “A Framework for Software Defect Prediction Using Neural Networks,” *Journal of Software Engineering and Applications*, vol. 8, pp. 384–394, 2015.

[18] J. Schmidhuber, “Deep Learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, 2015.

[19] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *Journal of Machine Learning Research*, 2012.

[20] S. K. Patro and K. K. Sahu, “Normalization: A Preprocessing Stage,” *IARJSET*, 2015.

[21] R. Shu, T. Xia, L. Williams, and T. Menzies, “Better Security Bug Report Classification via Hyperparameter Optimization,” pp. 1–12, 2019.