Received: January 16, 2021. Revised: February 27, 2021.

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

An Empirical Comparison of Resampling Ensemble Methods of Deep Learning Neural Networks for Cross-Project Software Defect Prediction

Mahmoud O. Elish[1]* **Karim Elish[2]**

[1]*Computer Science Department, Gulf University for Science and Technology, Mishref, Kuwait*
[2]*Department of Computer Science, Florida Polytechnic University, Lakeland, FL, USA*
* Corresponding author's Email: elish.m@gust.edu.kw

Abstract: Software defect prediction is one of the most important quality assurance activities during software development. This paper contributes empirical insights into the effectiveness of three resampling ensemble methods (bagging, boosting, and dagging) of Deep Learning Neural Networks (DLNN) for cross-project software defect prediction, compared to individual DLNN. An empirical study was conducted using five datasets. The results indicate that the bagging ensembles of DLNN offer only 0.24% increase in accuracy, on average, compared to the individual DLNN models, whereas the boosting and dagging ensembles degrade the accuracy. Furthermore, the results show that the three resampling ensembles of DLNN outperform the individual DLNN models in precision; with a maximum improved precision by 25.15% on average using the boosting ensembles. The results however indicate that none of the resampling ensembles improve the recall. Lastly, for a balanced performance in terms of both precision and recall, the results indicate improvements ranging from 0.98% on average by applying the bagging ensembles to 11.67% on average by applying the boosting ensembles.

Keywords: Software defect prediction, Deep learning, Ensemble methods, Neural networks, Empirical studies.

1. Introduction

Software testing and inspection play a vital role in software quality assurance. However, software testing is costly and labour-intensive, and can take up to 50% of the software development costs and more for safety-critical systems [1]. It is therefore important to focus software testing and inspection activities on defect-prone modules of software systems. This requires the development of effective software defect prediction models.

Cross-project software defect prediction refers to the process of predicting defects in a new software system using the historical data of other systems. On one hand, cross-project defect prediction is more practical and industrially viable than within project, as the later suffers from the scarcity of data in the early phases of software development. On the other hand, the accuracy of the cross-project prediction models is usually lower than those of the within project models due to the differences between the source and the target software projects.

In recent years, Deep Learning Neural Networks (DLNN) have been successfully applied in several fields and demonstrated to be effective. These include speech recognition [2], natural language processing [3], image processing [4], and software engineering [5-9]. The performance of DLNN varies from one dataset to another, as with other machine learning models.

Ensemble methods aggregate a group of base learners as a committee of an ensemble that decides on the prediction results by consensus using a combination rule. They aim to manage each of their individual base learners' strengths and weaknesses, and thus leading to the best possible decision being taken overall. There are heterogeneous and homogeneous ensembles [10]. Heterogeneous ensembles combine multiple models built from different machine learning algorithms, whereas homogeneous combine multiple models built from a single machine learning algorithm. Resampling ensemble methods belong to the homogeneous ensembles category.

Most of the existing studies [11-16] have applied ensemble methods for within project software defect prediction, not for the cross-project software defect prediction. Only few studies focused on cross-project software defect prediction and applied ensemble methods with Naïve Bayes (NB) [17], Support Vector Machine (SVM) [18], or Decision Tree (DT) and NB as base learners [19]. None of them have investigated the use of ensemble methods (bagging, boosting, and dagging) with DLNN as their base learner.

The objective of this paper is to empirically evaluate and compare the predictive effectiveness of three resampling ensemble methods (bagging, boosting, and dagging) that use DLNN as their base learner, and benchmark their performance against individual DLNN in the context of cross-project software defect prediction. The research question is whether or not bagging, boosting, and/or dagging ensemble methods of DLNN are more effective for cross-project software defect prediction compared to individual DLNN?

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 describes briefly the resampling ensemble methods. Section 4 reports the empirical study and analyzes its results. Finally, Section 5 provides the concluding remarks and suggests directions for future work.

2. Related work

Several studies have applied ensemble methods for within project software defect prediction. Rathore and Kumar [11, 12] investigated some linear and non-linear heterogeneous ensemble methods for intra-release and inter-release predictions of the number of faults in software systems. Zheng [13] evaluated cost-sensitive boosting of Artificial Neural Networks (ANN) for software defect prediction. Shanthini and Chandrasekaran [14] evaluated bagged ensemble of SVM for software fault prediction. Aljamaan and Elish [15] investigated bagging and boosting ensembles in identifying faulty classes in object-oriented software. As base classifiers, they used MultiLayer Perceptron (MLP), Radial Basis Function (RNF) network, Bayesian Belief Network (BBN), NB, SVM, and DT. Misirli et al. [16] presented an ensemble method that combines NB, ANN, and voting feature intervals for locating software defects.

In the context of cross-project software defect prediction, some studies have applied ensemble methods. Chen et al. [17] introduced double transfer boosting for cross-company software defects prediction, and used NB as a base learner. Ryu et al. [20] proposed a transfer cost-sensitive boosting approach for cross-project defect prediction that uses also NB as a base learner. Ryu et al. [18] applied SVM learner with boosting in the context of cross-project defect prediction. Zhang et al. [19] investigated bagging and boosting ensembles for cross-project defect prediction, and used DT and NB as base learners. Additionally, they applied average voting, majority voting, and random forest. Uchigaki et al. [21] proposed an ensemble of Logistic Regression (LR) model that uses weighted sum of outputs for cross-project fault prediction.

Few studies have applied DLNN models for software defect prediction, but not as ensembles. Qiao et al. [22] applied deep neural network-based model to predict the number of defects in software systems. The results indicated that their proposed method reduces the mean square error by more than 14% and increases the squared correlation coefficient by more than 8%. Majd et al. [23] used deep-learning models for statement-level software defect prediction, and the results showed their effectiveness.

Some systematic literature reviews have been conducted on software fault prediction [24-27]. The most relevant one is by Hosseini et al. [27] who conducted a systematic literature review and meta-analysis on cross-project defect prediction. Among their findings, it was found that NB and LR are the most widely used models for that purpose as individual models or as the base for ensembles. NB seems to have an average performance, whereas Nearest Neighbour (NN), SVM, and DT have the highest median F-measure. Ensemble methods, however, perform below average in terms of F-measure, and best in terms of AUC measure.

Unlike other related works, this paper focuses on cross-project software defect prediction to predict the defects in a new software system using the historical data of other systems and utilizing three resampling ensemble methods of DLNN to achieve this goal.

3. Resampling ensemble methods

The ensemble learning for a classification problem aggregates a group of base classifiers as a committee of ensemble that decides on the prediction results by consensus using a combination rule such as majority voting. In case of a resampling ensemble method, base classifiers are trained on different subsets of the training data using a resampling technique. Three popular resampling ensemble methods (bagging, boosting, and dagging) were empirically evaluated in this research for the aim of

Received: January 16, 2021. Revised: February 27, 2021.

cross-project software prediction using DLNN as base classifiers.

DLNN represents stacked neural networks, i.e., networks composed of several layers. The layers consist of nodes where each layer is trained using distinct set of features based on the outputs from its previous layer. The nodes are computational units that combine their inputs using weights. These input-weight outcomes are then summed and forwarded to an activation function that produces the final outcome.

3.1 Bagging

Bagging stands for Bootstrap Aggregation, and was introduced by Bieman [28]. This ensemble method starts with bootstrap sampling (i.e. random sampling with replacement) of the training dataset. A base classifier is then developed for each sample. Finally, the results of these multiple classifiers are then combined using majority voting. Bagging is thus considered a parallel ensemble since the base classifiers are developed in parallel during the training phase. Figure. 1 provides an overview of the bagging ensemble, whereas Figure. 2 depicts its algorithm.

3.2 Boosting

Boosting, which was introduced by Freund [29], works by training a set of classifiers sequentially by combing them for classification, where each latter classifier focusses on the errors of the earlier classifiers. In the first iteration of this ensemble method, a base classifier takes the training dataset and assigns equal weight to each of its instances. In the next iteration, the incorrectly classified instances are assigned to the next base classifier with a higher weightage. The iterations then continue likewise until the algorithm can correctly classify the output. There are a few types of boosting algorithms, and this research has used Adaptive Boosting (AdaBoost) [30]. Figure. 3 provides an overview of the boosting ensemble, and Figure. 4 depicts its algorithm.

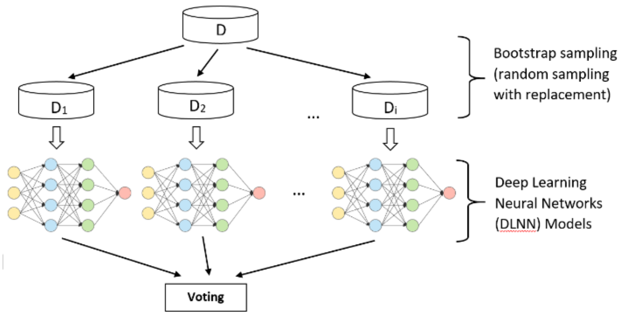
D
D₁
D₂
Dᵢ
Bootstrap sampling (random sampling with replacement)
Deep Learning Neural Networks (DLNN) Models
Voting

Figure. 1 Bagging ensemble

```
model generation
Let $N$ be the number of instances in the training data $D$
For each iteration $i$:
    $D_i$ = random sample of size $N$ with replacement from $D$
    Apply the classifier algorithm to $D_i$
    Store the resulting $\text{model}_i$
classification
For each model $i$:
    Predict the class of instance using $\text{model}_i$
Return class that has been predicted most often (majority voting)
```

Figure. 2 Bagging algorithm [15]

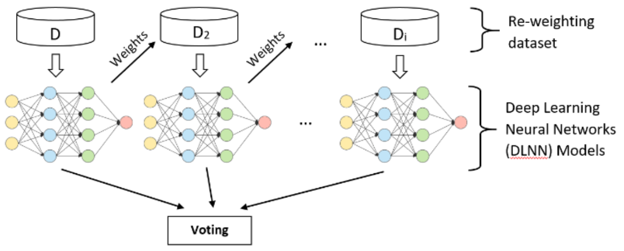

Figure. 3 Boosting ensemble

```
model generation
Assign equal weight to each instance in the training data $D$
For iteration $i$:
    Apply the classifier to the weighted dataset $D_i$
    Store the resulting $\text{model}_i$
    Compute error $e$ of $\text{model}_i$
    If $e = 0$ or $e \geq 0.5$:
        Terminate models generation
    Else generate $D_{i+1}$ as follows:
        For each instance in $D_i$:
            If instance was classified correctly by $\text{model}_i$:
                Multiply the weight of instance by $e$ $(1–e)$
            Copy the instance from $D_i$ to $D_{i+1}$
    Normalize the weights of all instances in $D_{i+1}$
classification
For each model $i$ (or less):
    Add $–\log(e/(1–e))$ to the weight of class predicted by $\text{model}_i$
Return class with highest weight
```

Figure. 4 Boosting algorithm [15]

3.3 Dagging

Dagging [31] is similar to bagging except that disjoint sampling is applied, instead of bootstrap sampling, to the training dataset. Following the resampling of the training dataset, a base classifier is developed for each sample. The results of these multiple classifiers are then combined using majority voting. Dagging is therefore considered a parallel ensemble. Figure. 5 provides an overview of it, and Figure. 6 depicts its algorithm.

4. Empirical study

The goal of this empirical study was to evaluate and compare the accuracy of bagging, boosting, and

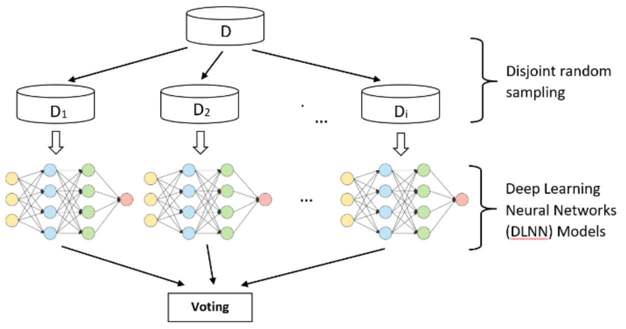

Figure. 5 Dagging ensemble

```
model generation
Let N be the number of instances in the training data D
For each iteration i:
        D_i = disjoint random sample from D
        Apply the classifier algorithm to D_i
        Store the resulting model_i
classification
For each model i:
        Predict the class of instance using model_i
Return class that has been predicted most often (majority voting)
```

Figure. 6 Dagging algorithm

dagging ensemble methods of DLNN against individual DLNN for the purpose of cross-project software defect prediction. It aimed to determine the extent to which these resampling ensemble methods offer reliable and improved classification accuracy over the individual model using different datasets. This section reports the details of the empirical study that was conducted and analyzes its results.

4.1 Datasets

Publicly accessible datasets were chosen that represent five open source software projects implemented in Java: Ant 1.7, Camel 1.6, Synapse 1.2, Velocity 1.6.1, and Xalan 2.6. These datasets are available through the PROMISE repository of empirical software engineering data [32]. The observations in these datasets correspond to the classes in the corresponding object-oriented software. Furthermore, these datasets have one dependent variable and 17 independent variables. The dependent variable is a binary variable that indicates whether or not the corresponding class is defective regardless of the number, type and severity of defects if any. The independent variables, on the other hand, represent several class-level metrics that measure various structural characteristics such as size, complexity, coupling, cohesion, and inheritance. Descriptive statistics of these five datasets in terms of the number of classes (observations) and the percentage of defective classes are provided in Table 1, whereas Table 2 lists the independent variables.

Table 1. Descriptive statistics of the datasets

Dataset	# of classes	% of defective classes
Ant 1.7	745	22.3%
Camel 1.6	965	19.5%
Synapse 1.2	256	33.6%
Velocity 1.6.1	229	34.1%
Xalan 2.6	885	46.4%

Table 2. Independent variables

Metric	Description
WMC	Weighted methods per class
DIT	Depth of inheritance tree
NOC	Number of Children
CBO	Coupling between object classes
RFC	Response for a class
LCOM	Lack of cohesion in methods
CA	Afferent couplings
CE	Efferent couplings
NPM	Number of public methods
LCOM3	Lack of cohesion in methods (another
LOC	Lines of code
DAM	Data access metric
MOA	Measure of aggregation
MFA	Measure of function abstraction
CAM	Cohesion among methods of class
IC	Inheritance couplings
CBM	Coupling between methods
AMC	Average method complexity
MAX_C	Maximum McCabe's cyclomatic
AVG_C	Average McCabe's cyclomatic complexity

4.2 Models development and performance metrics

Given the five datasets representing five different software projects, five experiments were conducted. In each experiment, four datasets were used for training the models and the fifth was used for testing. Four classification prediction models were developed and optimized using Eclipse Deeplearning4j programming library and WEKA machine learning software during each experiment:

1. DLNN – individual DLNN model
2. Bagging – bagging ensemble of DLNN
3. Boosting – boosting ensemble of DLNN
4. Dagging – dagging ensemble of DLNN

In order to assess the performance of the classification prediction models, several metrics were used: accuracy, precision, recall, and F-measure. The accuracy is a measure of correct classification rate, which is the ratio of classes that were correctly classified to the total number of classes in the software. The precision is the ratio of classes that were correctly classified as defective to the total number of classes that were classified as defective. The recall is the ratio of classes that were correctly classified as defective to the total number of actually defective classes. The F-measure is the harmonic mean of both precision and recall metrics, since there is a trade-off between both of them. Mathematically, these metrics are calculated as follows:

$$Accurcay = \frac{TP+TN}{N} \tag{1}$$

$$Precision = \frac{TP}{TP+FP} \tag{2}$$

$$Recall = \frac{TP}{TP+FN} \tag{3}$$

$$F-measure = \frac{2\ x\ Precision\ x\ Recall}{Precision+Recall} \tag{4}$$

Where TP is the number of true positive cases, i.e., the cases that are actually defective and correctly predicted as defective. TN is the number of true negative cases, i.e., the cases that are actually not defective and correctly predicted as not defective. FP is the number of false positive cases, i.e., the cases that are actually not defective but predicted as defective. FN is the number of false negative cases, i.e., the cases that are actually defective but predicted as not defective. N in the total number of cases.

In addition to the above metrics, Area Under Curve (AUC) was measured, which is calculated based on the Receiver Operating Characteristic (ROC) curve that plots the true positive rate versus the false positive rate at various threshold settings. It is calculated as follows [33]:

$$AUC = \sum_i \{(1-\beta_i \cdot \Delta\alpha) + \frac{1}{2}[\Delta(1-\beta)\cdot\Delta\alpha]\} \tag{5}$$

$$1-\beta = TruePositiveRate = \frac{TP}{TP+FN} \tag{6}$$

$$\alpha = FalsePositiveRate = \frac{FP}{FP+TN} \tag{7}$$

4.3 Results and analysis

Table 3 reports the measures of classification performance of the ensemble models and the individual DLNN models when applied for cross-project software defect prediction over the five datasets. Additionally, gain/loss analysis of the performance of the ensemble methods in each measure over the individual DLNN models are reported in Table 4, Table 5, Table 6, Table 7, and Table 8; where gain values are in green font.

Table 3. Classification prediction results

Dataset	Model	Accuracy	Precision	Recall	F-measure	AUC
Ant	DLNN	**0.809**	0.398	**0.611**	0.482	0.657
	Bagging	0.804	0.398	0.589	0.475	**0.667**
	Boosting	0.596	**0.867**	0.340	0.489	0.380
	Dagging	0.780	0.524	0.506	**0.515**	0.536
Camel	DLNN	**0.789**	0.138	0.382	0.203	0.586
	Bagging	0.781	0.101	0.311	0.153	**0.602**
	Boosting	0.762	**0.218**	0.331	**0.263**	0.569
	Dagging	0.784	0.181	**0.386**	0.246	0.589
Synapse	DLNN	0.723	0.302	0.703	0.423	**0.676**
	Bagging	**0.734**	0.349	**0.714**	0.469	0.672
	Boosting	0.711	**0.512**	0.579	**0.543**	0.451
	Dagging	0.715	0.372	0.627	0.467	0.655
Velocity	DLNN	**0.664**	0.090	**0.538**	0.154	0.639
	Bagging	0.655	0.103	0.471	0.168	**0.675**
	Boosting	0.651	**0.295**	0.479	**0.365**	0.622
	Dagging	0.651	0.141	0.458	0.216	0.606
Xalan	DLNN	0.655	0.307	0.863	0.452	0.592
	Bagging	0.677	0.345	**0.893**	0.498	**0.664**
	Boosting	**0.684**	**0.599**	0.681	**0.637**	0.494
	Dagging	0.654	0.392	0.742	0.513	0.494

By analyzing the obtained accuracy results (Table 3 and Table 4) of the ensemble methods and comparing it to the individual DLNN models, the following can be observed. The bagging ensemble models offered accuracy improvements of 1.17% and 2.15%, in the Synapse and Xalan datasets respectively, over the individual DLNN models. They achieved an average improvement of only 0.24% across the five datasets. In case of the boosting ensemble models, an accuracy improvement of 2.82% was only observed in the Xalan dataset. However, these boosting models degraded the accuracy of the individual DLNN models by 4.74% on average across the datasets. Lastly, the dagging ensemble models consistently degraded the accuracy of the individual DLNN models with an average of 1.11% across the datasets.

Looking at the precision results (Table 3 and Table 5), the bagging ensemble models offered improvements in precision over the individual DLNN models in all datasets, except the Camel dataset. On average, 1.22% improvement was observed across

Table 4. Gain/loss analysis of ensemble methods' accuracy

Dataset	Bagged_DLN	Boosted_DLN	Dagged_DLN
Ant	-0.54%	-21.34%	-2.95%
Camel	-0.73%	-2.69%	-0.41%
Synaps	1.17%	-1.17%	-0.78%
Velocit	-0.87%	-1.31%	-1.31%
Xalan	2.15%	2.82%	-0.11%
Avg.	*0.24%*	*-4.74%*	*-1.11%*

Table 5. Gain/loss analysis of ensemble methods' precision

Dataset	Bagged_DLN	Boosted_DLN	Dagged_DLN
Ant	0.00%	46.99%	12.65%
Camel	-3.72%	7.98%	4.26%
Synaps	4.65%	20.93%	6.98%
Velocit	1.28%	20.51%	5.13%
Xalan	3.89%	29.20%	8.52%
Avg.	*1.22%*	*25.12%*	*7.51%*

the datasets. On the other hand, both the boosting and dagging ensemble models consistently outperformed the individual DLNN models across the datasets. In case of boosting, there was a significant improvement of 25.12% in precision on average, and an average improvement of 7.51% in precision was obtained in case of dagging.

By considering the recall results (Table 3 and Table 6), we can observe the following. The bagging ensemble models offered recall improvements of 1.16% and 3.01%, in the Synapse and Xalan datasets respectively, over the individual DLNN models. However, across the datasets, an average degradation of 2.38% was observed. The boosting ensemble models consistently degraded the recall of the individual DLNN models with an average of 13.74% across the datasets. Similarly, the dagging ensemble models degraded the recall in all datasets except the Camel dataset; with an average of 7.55% across the datasets.

When considering the F-measure results (Table 3 and Table 7) to overcome the trade-off between precision and recall, the analysis yields the following

Table 6. Gain/loss analysis of ensemble methods' recall

Dataset	Bagged_DLN	Boosted_DLN	Dagged_DLN
Ant	-2.18%	-27.07%	-10.53%
Camel	-7.09%	-5.17%	0.40%
Synaps	1.16%	-12.38%	-7.53%
Velocit	-6.79%	-5.93%	-8.01%
Xalan	3.01%	-18.16%	-12.11%
Avg.	*-2.38%*	*-13.74%*	*-7.55%*

Table 7. Gain/loss analysis of ensemble methods' F-measure

Dataset	Bagged_DLN	Boosted_DLN	Dagged_DLN
Ant	-0.69%	0.72%	3.30%
Camel	-5.05%	5.97%	4.33%
Synaps	4.60%	12.04%	4.44%
Velocit	1.46%	21.12%	6.18%
Xalan	4.58%	18.49%	6.03%
Avg.	*0.98%*	*11.67%*	*4.86%*

Table 8. Gain/loss analysis of ensemble methods' AUC

Dataset	Bagged_DLN	Boosted_DLN	Dagged_DLN
Ant	1.00%	-27.70%	-12.10%
Camel	1.60%	-1.70%	0.30%
Synaps	-0.40%	-22.50%	-2.10%
Velocit	3.60%	-1.70%	-3.30%
Xalan	7.20%	-9.80%	-9.80%
Avg.	*2.60%*	*-12.68%*	*-5.40%*

observations. The bagging ensemble models slightly outperformed the individual DLNN models in three out of the five datasets, with an average improvement of just 0.98%. Both boosting and dagging ensemble models consistently outperformed the individual DLNN models across the datasets. In case of boosting, there was a significant improvement of 11.67% on average, and an average improvement of 4.86% was observed in case of dagging.

Finally, the AUC results (Table 3 and Table 8) indicate that the bagging ensemble models have better AUC values compared to the individual DLNN models in all datasets except the Synapse dataset. An average gain of 2.6% was achieved by bagging across

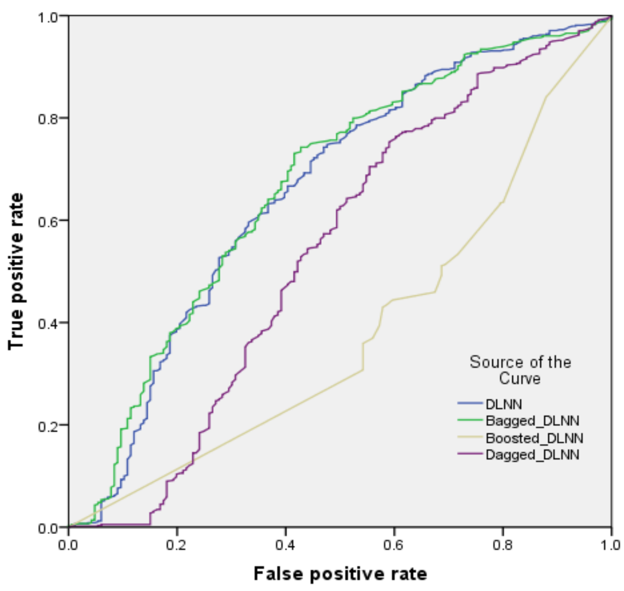

Figure 7. ROC curves - ant

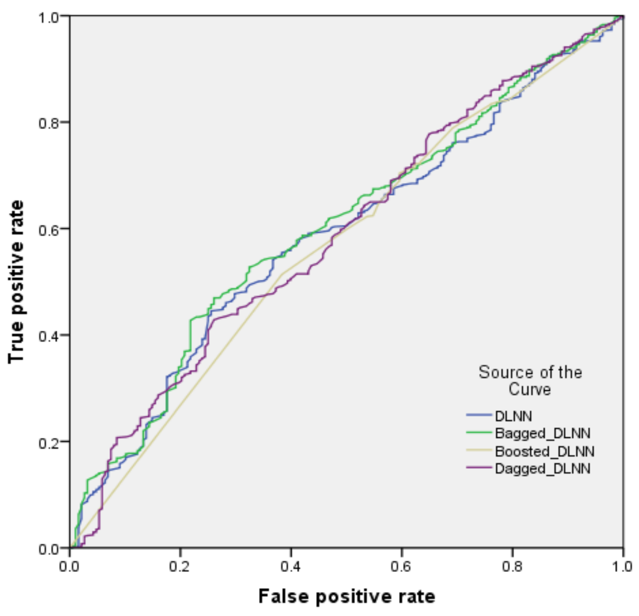

Figure 8. ROC curves – camel

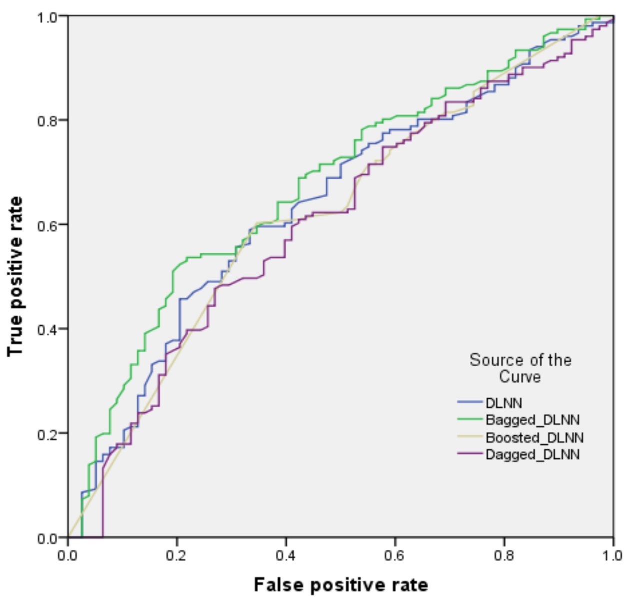

Figure 10. ROC curves - velocity

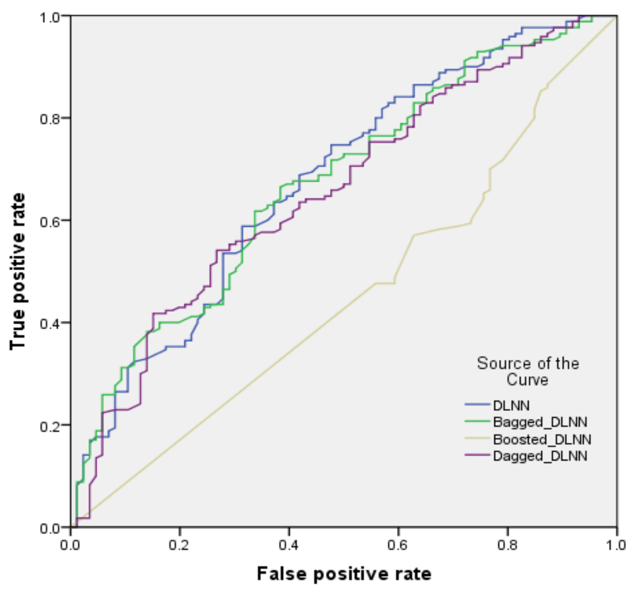

Figure 9. ROC curves - synapse

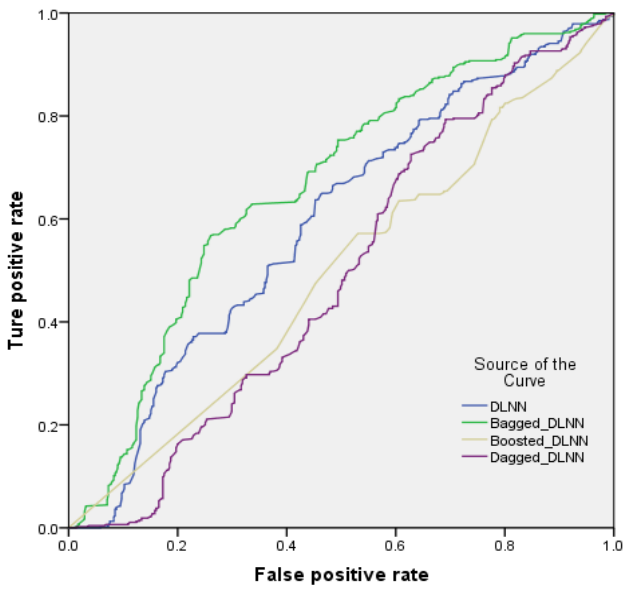

Figure 11. ROC curves - xalan

the datasets. On the contrary, an average loss of 12.68% was observed in case of the boosting ensemble models, and an average loss of 5.4% in case of the dagging ensemble models. These observations are further supported by the plots of the ROC curves that are provided in Figure 7, Figure 8, Figure 9, Figure 10, and Figure 11. The ROC curves of the bagging ensemble models are the top-most curves in these plots.

5. Concluding remarks

This paper has empirically evaluated and compared the predictive effectiveness of three resampling ensemble methods (bagging, boosting, and dagging) of Deep Learning Neural Networks (DLNN) against individual DLNN for the purpose of cross-project software defect prediction. An empirical study was conducted using five datasets. Prediction classification models were assessed using accuracy, precision, recall, F-measure, and AUC. The results suggest that the answers to the questions of (i) whether or not bagging, boosting, or dagging ensemble methods of DLNN should be applied instead of individual DLNN for cross-project software defect prediction? and if so, (ii) which one of them should be applied? depend on the classification performance metric that has the highest priority. In other words, the results indicate that the bagging ensembles of DLNN offer only 0.24%

increase in accuracy, on average, compared to the individual DLNN models, whereas the boosting and dagging ensembles degrade the accuracy. Furthermore, the results show that the three resampling ensembles of DLNN outperform the individual DLNN models in precision; with a maximum improved precision by 25.15% on average using the boosting ensembles. The results however indicate that none of the resampling ensembles improve the recall. Lastly, for a balanced performance in terms of both precision and recall, the results indicate improvements ranging from 0.98% on average by applying the bagging ensembles to 11.67% on average by applying the boosting ensembles.

This paper has contributed novel and interesting empirical insights into the applications of three resampling ensemble methods (bagging, boosting, and dagging) of DLNN to the problem of cross-project software defect prediction. The effectiveness of these ensemble methods, as suggested by the results of this study, depends on the classification performance metric that has the highest priority from the perspective of a software developer/manager as discussed previously. Therefore, it is recommended to first decide on the highest priority classification performance metric, and then select the proper ensemble method or just an individual DLNN to apply accordingly.

As a future work, more empirical studies may be conducted to further support the findings of this study and to accumulate knowledge using other datasets. Investigation of other types of ensemble methods, including heterogeneous ensembles, is another direction for future work. It is also worth investigating the effectiveness of ensemble methods on other software quality and security prediction problems.

Conflicts of Interest

The authors declare no conflict of interest.

Author Contributions

Conceptualization, M. Elish; methodology, M. Elish; validation, M. Elish and K. Elish; formal analysis, M. Elish and K. Elish; investigation, M. Elish and K. Elish; resources, M. Elish; data curation, M. Elish and K. Elish; writing—original draft preparation, M. Elish and K. Elish; writing—review and editing, M. Elish and K. Elish; visualization, M. Elish and K. Elish.

References

[1] P. Ammann and J. Offutt, *Introduction to Software Testing*. Cambridge University Press, 2008.

[2] Y. Qian, M. Bi, T. Tan, and K. Yu, "Very Deep Convolutional Neural Networks for Noise Robust Speech Recognition", *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, Vol. 24, No. 12, pp. 2263–2276, 2016.

[3] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing", *IEEE Computational Intelligence Magazine,* Vol. 13, No. 3, pp. 55-75, 2017.

[4] T. Goswami, "Impact of deep learning in image processing and computer vision", In: *Proc. of Microelectronics, Electromagnetics and Telecommunications*, pp. 475–485, 2018.

[5] G. Zhao and J. Huang, "Deepsim: Deep learning code functional similarity", In: *Proc. of 26 ACM Joint Meeting on European Software Engineering Conf. and Symposium on the Foundations of Software Engineering,* pp. 141-151, 2018.

[6] J. Guo, J. Cheng, and J. Cleland-Huang, "Semantically enhanced software traceability using deep learning techniques", In: *Proc. of 39th IEEE/ACM International Conf. on Software Engineering*, pp. 3-14, 2017.

[7] M. White, C. Vendome, M. Linares-Vásquez, and D. Poshyvanyk, "Toward deep learning software repositories", In: *Proc. of 12th Working Conf. on Mining Software Repositories*, pp. 334–345, 2015.

[8] M. Balog, A. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow, "Deepcoder: Learning to write programs", In: *Proc. of 5th International Conf. on Learning Representations*, pp. 1-19, 2017.

[9] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural- network-driven autonomous cars", In: *Proc. of 4th International Conf. on Software Engineering,* pp. 303-314, 2018.

[10] M. Elish, T. Helmy, and M. Hussain, "Empirical Study of Homogeneous and Heterogeneous Ensemble Models for Software Development Effort Estimation", *Mathematical Problems in Engineering*, 2013.

[11] S. Rathore and S. Kumar, "Towards an ensemble based system for predicting the number of software faults", *Expert Systems with Applications*, Vol. 82, pp. 357–382, 2017.

[12] S. Rathore and S. Kumar, "Linear and non-linear heterogeneous ensemble methods to predict the number of faults in software systems", *Knowledge-Based Systems,* Vol. 119, pp. 232-256, 2017.

[13] J. Zheng, "Cost-sensitive boosting neural networks for software defect prediction", *Expert Systems with Applications,* Vol. 37, No. 6, pp. 4537–4543, 2010.

[14] A. Shanthini and R. Chandrasekaran, "Analyzing the effect of bagged ensemble approach for software fault prediction in class level and package level metrics", In: *Proc. of IEEE International Conf. on Information Communication and Embedded Systems*, pp. 1-5, 2014.

[15] H. Aljamaan and M. Elish, "An Empirical Study of Bagging and Boosting Ensembles for Identifying Faulty Classes in Object-Oriented Software", In: *Proc. of IEEE Symposium on Computational Intelligence and Data Mining,* pp. 187-194, 2009.

[16] A. Misirli, A. Bener, and B. Turhan, "An industrial case study of classifier ensembles for locating software defects", *Software Quality Journal,* Vol. 19, No. 3, pp. 515–536, 2011.

[17] L. Chen, B. Fang, Z. Shang, and Y. Tang, "Negative samples reduction in cross-company software defects prediction", *Information and Software Technology,* Vol. 62, pp. 67–77, 2015.

[18] D. Ryu, O. Choi, and J. Baik, "Value-cognitive boosting with a support vector machine for cross-project defect prediction", *Empirical Software Engineering,* Vol. 21, No. 1, pp. 43-71, 2016.

[19] Y. Zhang, D. Lo, X. Xia, and J. Sun, "An empirical study of classifier combination for cross-project defect prediction", In: *Proc. of 39th IEEE Annual Computer Software and Applications Conf.,* pp. 264–269, 2015.

[20] D. Ryu, J. I. Jang, and J. Baik, "A transfer cost-sensitive boosting approach for cross-project defect prediction", *Software Quality Control,* Vol. 25, No. 1, pp. 235–272, 2017.

[21] S. Uchigaki, S. Uchida, K. Toda, and A. Monden, "An ensemble approach of simple regression models to cross-project fault prediction", In: *Proc. of 13th ACIS International Conf. on Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing,* pp. 476–481, 2012.

[22] L. Qiao, X. Li, Q. Umer, and P. Guo, "Deep learning based software defect prediction", *Neurocomputing,* Vol. 385, pp. 100-110, 2020.

[23] A. Majd, M. Vahidi-Asl, A. Khalilian, and P. Poorsarvi-Tehrani, "SLDeep: Statement-level software defect prediction using deep-learning model on static code features", *Expert Systems with Applications,* Vol. 147, 2020.

[24] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "Systematic Literature Review on Fault Prediction Performance in Software Engineering", *IEEE Transactions on Software Engineering,* Vol. 38, No. 6, pp. 1276–1304, 2012.

[25] C. Catal, "Software fault prediction: A literature review and current trends", *Expert Systems with Applications,* Vol. 38, No. 4, pp. 4626–4636, 2011.

[26] R. Wahono, "A Systematic Literature Review of Software Defect Prediction: Research Trends, Datasets, Methods and Frameworks", *Journal of Software Engineering,* Vol. 1, No. 1, 2015.

[27] S. Hosseini, B. Turhan, and D. Gunarathna, "A systematic literature review and meta-analysis on cross project defect prediction", *IEEE Transactions on Software Engineering,* Vol. 45, No. 2, pp. 111-147, 2017.

[28] L. Breiman, "Bagging predictors", *Machine Learning,* Vol. 24, No. 2, pp. 123-140, 1996.

[29] Y. Freund, "Boosting a weak learning algorithm by majority", *Information and Computation,* Vol. 121, No. 2, pp. 256–285, 1995.

[30] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm", In: *Proc. of 13th International Conf. on Machine Learning,* Italy, pp. 148-156, 1996.

[31] K. Ting and I. Witten, "Stacking Bagged and Dagged Models", In: *Proc. of 14th International Conf. on Machine Learning*, pp. 367-375, 1997.

[32] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan, "The promise repository of empirical software engineering data", *West Virginia University, Department of Computer Science, 2012.*

[33] A. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms", *Pattern Recognition,* Vol. 30, No. 7, pp. 1145-1159, 1997.