Cross-Project Defect Prediction via Transferable Deep Learning-Generated and Handcrafted Features

Shaojian Qiu[1], Lu Lu[1*], Ziyi Cai[1] and Siyu Jiang[2]
[1]School of Computer Science and Engineering, South China University of Technology, Guangzhou, China
[2]School of Software Engineering, South China University of Technology, Guangzhou, China
[*]Corresponding author email: lul@scut.edu.cn

***Abstract*—Although the machine learning-based software defect prediction (SDP) method has shown promising value in software engineering, yet challenges remain. To improve the performance of SDP, some researchers have used deep learning algorithms to extract the semantic and structural features of the program. However, in more practical cross-project defect prediction (CPDP) tasks, whether deep learning-generated features can be directly used should be explored due to the data distribution shift that usually exists in different projects. In this paper, we propose a Transferable Hybrid Features Learning with Convolutional Neural Network (CNN-THFL) framework to conduct CPDP. Specially, CNN-THFL mines deep learning-generated features from token vectors extracted from programs' abstract syntax trees via convolutional neural network. Furthermore, CNN-THFL learns the transferable joint features simultaneously considering deep learning-generated and handcrafted features by applying a transfer component analysis algorithm. Finally, the features generated by CNN-THFL are fed to the classifier to train a defect prediction model. Extensive experiments verify that CNN-THFL can outperform referential methods on 72 pairs of CPDP tasks formed by 9 open-source projects.**

***Keywords*—Software defect prediction, Cross-project defect prediction, Transfer learning, Semantic feature learning**

I. Introduction

Under the trend of increasing software scale and complexity, how to effectively ensure the reliability of software has become a popular research topic. Software defect prediction (SDP), as a novel proposition of quality assurance technology, has received a great deal of attention from researchers [1]–[3]. It is designed to build a predictive model with the machine-learning model and historical software defect data before detecting the defect-prone modules or files in the software. Effective defect prediction can help software quality assurance teams or code reviewers allocate testing resources more reasonably.

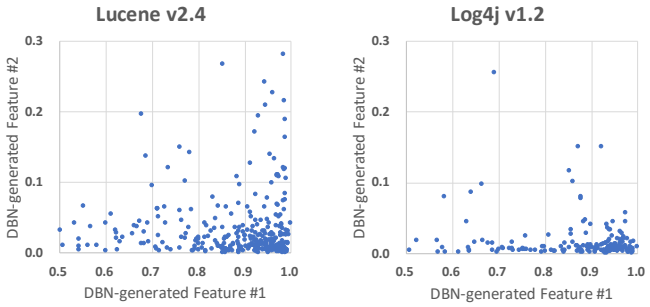
SDP usually consists of two phases: extracting features from program files and training the predictor via the machine-learning [4] method. In previous studies, the discriminant features adopted to construct predictive models were elaborately extracted. These handcrafted features mainly include Halstead features [5] based on operators and operands, McCabe features [6] based on dependencies, and CK features [7] based on the object-oriented concept. In recent years, some researchers [4], [8] have suggested that it is not enough to only consider handcrafted features. To expand the features available in SDP for improving predictive performance, they tried to use the deep-learning methods (e.g., Deep Belief Network, DBN and Convolutional Neural Network, CNN) to mine the semantic and structural features hidden in the software program. The main idea of these methods is to extract the deep learning-generated features from the token vectors generated by programs' Abstract Syntax Trees (ASTs) and feed to the data with these generated features to the machine-learning classifier to obtain the SDP model. Their experiments show that DBN and CNN methods are superior to the traditional SDP methods that use only handcrafted features in the cross-version defect prediction tasks [9].

In practice, it is often difficult to establish an accurate defect predictor for new projects due to the scarcity of defect labels. To overcome this problem, cross-project defect prediction (CPDP) [7] was proposed as an alternative solution to defect predictors that learn new projects (called target projects) by using labeled data from mature projects (called source projects). Then, in the tasks of CPDP, there is a research question of whether deep learning-generated features extracted from the source project can be directly used for the SDP task of the target project. In [10], Wang et al. explore the effect of DBN-generated features on CPDP tasks; however, their experiments were based only on the assumption that the semantic features generated by DBN can capture the common features of defects, which means the features obtained from a project can be used in other projects. In this paper, we posit that this assumption may not always hold. We suggest that, because the data of the source and target projects usually have different distributions, the deep learning-generated features should not be used directly in CPDP tasks.

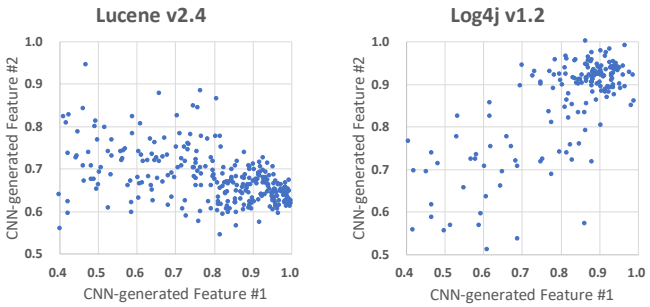
Let us use an example to show this distribution discrepancy. As shown in Figure 1, we present the data distribution of two real projects, Apache Lucene v2.4 and Apache Ant v1.7. Using these two projects, we can form a sample based on DBN-generated features (Figure 1a) and a sample based on CNN-generated features (Figure 1b). For a comparable demonstration, we selected two-dimensional features to be drawn on the X and Y axes, respectively. We normalized the data to be displayed with min-max scaling and charted the main distributions in the coordinates. In Figure 1, the discrepancy of the data distribution is clearly shown to exist in the two projects ($Pr(X_s) \neq Pr(X_t)$). As we know, if the distribution of training and test data do not match, we are facing sample selection bias or covariate shift problems that will greatly

DOI reference number: 10.18293/SEKE2019-070

affect the performance of the predictive model [11].



(a) Distribution discrepancy based on DBN-generated features



(b) Distribution discrepancy based on CNN-generated features

Fig. 1: The distribution discrepancy existed between projects.

To solve this problem, we proposed the Transferable Hybrid Features Learning framework with Convolutional Neural Network (CNN-THFL) to perform CPDP. Specifically, we parsed the source files into ASTs and selected representative nodes on the AST to form token vectors. By converting the token vectors into integer vectors, we adopted CNN to extract the deep learning-generated features. In the process, we concatenated deep learning-generated features with the traditional handcrafted features to construct the hybrid features [4]. To enhance the transferability of hybrid features in CPDP tasks, we attempted to learn transfer components across projects in a reproducing kernel Hilbert space (RKHS) using transfer component analysis (TCA), which could match the data distribution between projects. Finally, the transferable hybrid features generated by CNN-THFL could be fed to the base classifier to train the CPDP model.

The main contributions of this paper are:

- We propose a features-learning framework called CNN-THFL, which jointly considers the transferability of deep learning-generated and handcrafted features in cross-project programs, to handle the distributions discrepancy between projects in CPDP tasks.

- To validate the effectiveness of CNN-THFL, we evaluate it on 72 pairs of CPDP tasks formed by 9 open-source projects. The experimental results show that in terms of average MCC, CNN-THFL improves the TCA by 30.1% and the DP-CNN by 62.8%.

- By experimental comparison, we found that when using CNN or DBN to mine deep learning-generated features, better results can be attained if this is simultaneously combined with handcrafted features and TCA algorithm, indicating the necessity of using hybrid features and considering their transferability.

We have organized the rest of this paper as follows. In Section II, we review some related works. In Section III, we show the high-level framework of CNN-THFL and elaborate its steps. In Section IV, we provide the experimental setup. In Section V, we show the experimental results to validate the effectiveness of CNN-THFL framework. In Section VI, we discuss the threats to validity. We concludes our work and point out the potential future works in Section VII.

II. Related Work

In this section, we briefly review the related CPDP works.

To effectively apply the SDP technique early in the software lifecycle, CPDP is proposed as a more feasible solution. Its central concept is learning a defect predictor for a new project (target project) by using labeled data from a mature project (source project). In a previous study of CPDP, Zimmermann et al. [12] conducted a large-scale experimental study on its feasibility. They selected 12 real-world projects and analyzed a total of 622 pairs of CPDP tasks. The results show that only 3.4% of tasks were able to achieve acceptable performance. To enhance the performance of the CPDP, Ma et al. [13] developed a transfer Naive Bayes model (TNB) that uses the weighted source data based on target set information to train a weighted Naive Bayes classifier. Nam et al. [14] applied a transfer-learning approach TCA+, which extended TCA [15] with customized normalizing rules, to make data distributions in source and target projects similar. The experimental results show that TCA+ can improve the performance of CPDP by transferable feature learning.

However, the aforementioned methods only use handcrafted features (e.g., Halstead [5], McCabe [6], and CK features [7]). In fact, if we can reasonably mine and use the semantic and structural information of the programs, it will be possible to further improve the performance of SDP. Recently, Wang et al. [10] tried to leverage DBN to automatically learn semantic features using token vectors extracted from the programs' ASTs. Their evaluation of 10 open-source projects shows that the DBN-learned features improve the performance of SDP. Based on the framework of DBN method, Li et al. [4] proposed that CNN is more advanced than DBN in capturing local patterns. They proposed a framework called Defect Prediction via Convolutional Neural Network (DP-CNN) to extract semantic and structural features from token vectors. The experimental results show that on average, DP-CNN improves the performance of SDP. It is worth mentioning that DP-CNN concatenates the CNN-learned feature vectors with traditional handcrafted feature vectors to avoid losing potential

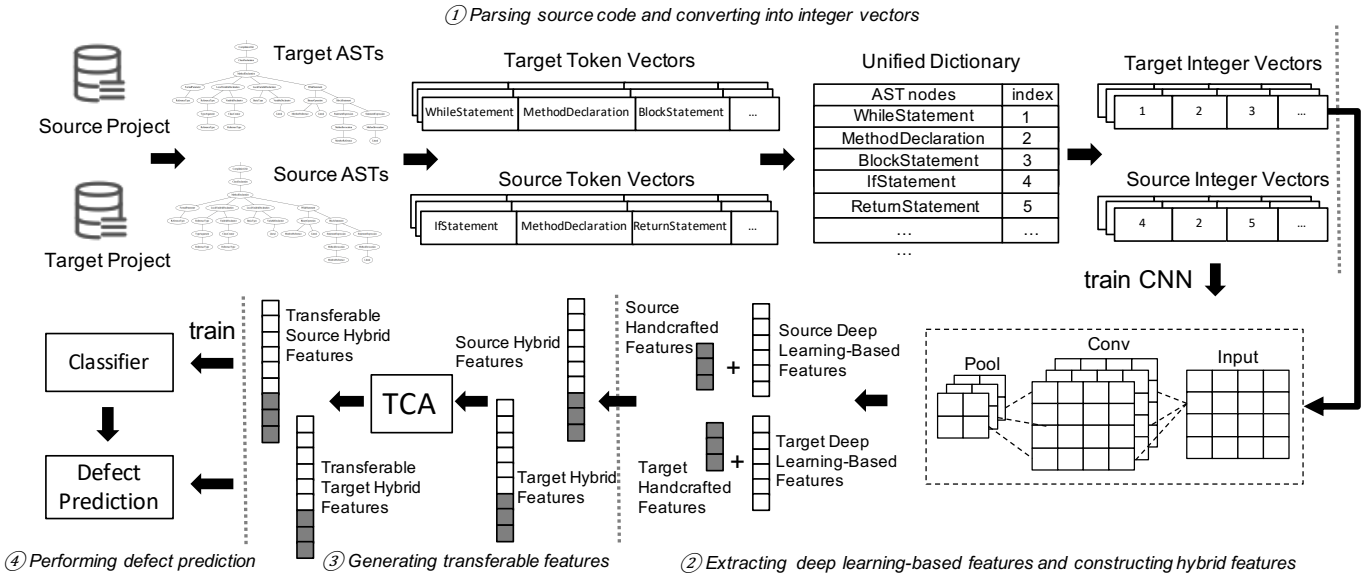


Fig. 2: Overview of our CNN-THFL framework to perform CPDP.

information in the latter. However, neither the DBN nor the CNN has considered the situation that there is distribution discrepancy across projects in CPDP tasks (Wang et al. [10] only assume that semantic features can capture the common features of defects and directly use DBN-generated features in CPDP tasks). Furthermore, this situation is exactly what we want to study and handle in this paper.

III. Methodology

A. Overall Framework

As presented in Figure 2, our CNN-THFL framework consists of four major steps: 1) parsing source code into tokens and converting them into integer vectors; 2) leveraging the CNN to automatically extract deep learning-generated features and constructing hybrid features; 3) generating transferable hybrid features via TCA; and 4) building classifier and predicting defects.

B. Parsing Source Code and Converting into Integer Vectors

AST is a tree representation of source code, and each structure in the source code can be represented as a node in the tree. Related work [10] has proven that ASTs can be used to detect the integrity and defects of source code. In this paper, we used the *Javalang*[1], an open-source Python package, to parse the Java files and generate corresponding token vectors. Table I lists the node categories and types used in this report. We use the name of the node type as the identifier for each node in the token vector. Considering that the names of methods, classes, and types are project-specific [8], we also used the type name (e.g., MethodDeclarations and ClassInvocation) to label nodes instead of the specific name used in [4].

The token vector extracted by Javalang cannot be directly used as input for CNN model training. To solve this problem, referring to [4], we established a mapping dictionary between tokens and integers so that the same token would be represented as the same integer. In this way, we could convert the token vectors into integer vectors. Furthermore, because CNN requires input vectors to have the same length, all input vectors would be filled with zero to the length of the longest vector.

TABLE I: The selected AST nodes.

Node Category	Node Type
Nodes of method invocations and instance creations	MethodInvocation, SuperMethodInvocation, ClassCreator
Declaration-related nodes	PackageDeclaration, InterfaceDeclaration, ClassDeclaration, ConstructorDeclaration, MethodDeclaration, VariableDeclarator, FormalParameter
Control-flow-related nodes	IfStatement, ForStatement, WhileStatement, DoStatement, AssertStatement, BreakStatement, ContinueStatement, ReturnStatement, ThrowStatement, TryStatement, SynchronizedStatement, SwitchStatement, BlockStatement, CatchClauseParameter, TryResource, CatchClause, SwitchStatementCase, ForControl, EnhancedForControl
Other nodes	BasicType, MemberReference, ReferenceType, SuperMemberReference, StatementExpression,

C. Extracting Deep Learning-Generated Features and Constructing Hybrid Features

In this study, we applied CNN's feature-generation capability to capture the semantics and local structure of source code [4]. Our CNN included an embedded layer, a convolutional layer, a maximum pool layer, a full-connection layer, and the last output layer as input to the base classifier. All other layers adopted the ReLU activation function except the

[1]https://pypi.org/project/javalang/0.9.2/

output layer, which used the sigmoid as the activation function. We implemented CNN by *Pytorch*[2], which has efficient tools for neural networks construction and enables fast, flexible experimentation.

To apply the knowledge carried in the handcrafted features at the same time, we stitched the these features of each project with the deep learning-generated features. The handcrafted features used in this paper were drawn from the metrics used by Jureczko and Madeyski in their defect prediction work [16]. Notice that we used the same handcrafted features from the source and target projects. We spliced deep learning-generated feature vectors with handcrafted feature vectors using the *Concatenate* method of Python to obtain hybrid feature vectors as the input for the next step.

D. Generating Transferable Features

In this study, we hoped to find transferable feature representation to handle the distribution discrepancy between source and target projects. TCA [15] is a transfer-learning method that allows knowledge of defects from a source project to be transferred to a target project. TCA attempts to learn some of the transferable components in the RKHS using maximum mean discrepancy [17]. In the subspace spanned by these transferable components, the properties of source and target data are preserved, and the data distributions in different projects are similar to each other. Therefore, through the new mapping data in this RKHS, we could train the base classifier in the source project, which was also available for the target project.

E. Performing Defect Prediction

For this paper, we chose logistic regression (LR) as a base classifier. We followed the aforementioned steps to process the files in the source and target projects and obtain the transferable hybrid feature for each file. After we fed the TCA-handled data of source project and corresponding label to the LR model, the weights and deviations in our LR would be obtained. Then we used the trained model to predict whether the instances of the target project were defective.

IV. Experimental Setup

A. Evaluated Projects

To assess the CNN-THFL framework, we collected 9 Java open-source projects from the PROMISE repository, which has been commonly used in recent CPDP researches [18]–[20]. Table II presents the basic information for the 9 projects. Each project consists of a collection of Java files, their corresponding 20 static code attributes (the detailed descriptions of which can be found in [18]), and a label (defective or clean). To verify the generality of our approach, the data sets were composed of several projects with different sizes (ranging from 205 to 815) and defective rates (a minimum value of 11.4% and a maximum value of 98.8%). In our CPDP task, given one of the projects as the training data, another eight projects could be used as the test data, respectively (e.g., using the data of Ant v1.7 as a training set and the data of Camel v1.6 as the test set). Thus, 72 pairs of CPDP tasks could be performed in this study.

[2]https://pytorch.org

TABLE II: The 9 projects selected from the PROMISE repository.

Project Name	Project Version	Instance Count	Defect Rate
Ant	1.7	745	22.3%
Camel	1.6	965	19.5%
Ivy	2.0	352	11.4%
Log4j	1.2	205	92.2 %
Lucene	2.4	340	59.7%
Synapse	1.2	256	33.6%
Velocity	1.6.1	229	34.1%
Xalan	2.7	909	98.8%
Xerces	1.4.4	588	74.3%

Software defect data have the typical characteristic of imbalanced distribution [21], [22], with the number of minority instances being less than the number of majority instances (e.g., in the Ivy-v2.0 project, the number of defective instances is far less than that of clean instances). In this study, we used the method of random oversampling to avoid imbalanced data that would degrade the performance of our model.

B. Evaluation Metrics

The metric of predictive performance is very important. Although the F1-socre has been widely used in recent years [18], [19], we believe that it has problems in SDP tasks [3], especially when there are imbalanced data sets. For example, F1-score excludes true negatives (TN) from calculations, which may be problematic. The test manager will be happy to know whether the component is truly defect-free. Thus, we adopted the Matthews Correlation Coefficient [23] (MCC), as a metric of predictive performance.

As shown in Table III, there may be four outputs when using the dichotomous classifier in the SDP task.

TABLE III: Confusion Matrix

	Truly Defective	Truly Clean
Predictively Defective	TP	FP
Predictively Clean	FN	TN

MCC is the geometric mean of the regression coefficients of the problem and its dual. Based on the confusion matrix, MCC is calculated by the following formula:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

As a correlation coefficient, MCC measures the relationship between predictive class and true class by taking into account all components of confusion matrix. Its return value is on a scale [-1,1] where 1 means a perfect positive correlation and

-1 means a perfect negative correlation. Specifically, MCC can take into account TN, and it is less sensitive to the imbalanced data set.

C. Comparative Methods

We compared CNN-THFL with 9 methods including:

- **LR.** Traditional method, which builds a LR classifier only using handcrafted features.
- **TCA.** A classic transferable features learning method [15].
- **DBN.** A standard DBN model to extract semantic features for SDP [10].
- **DBN-TCA.** A variant of DBN, which applies TCA to obtain transferable DBN-learned features.
- **DBN-DP.** An improved version of DBN proposed by [4], which concatenates the DBN-learned features with the handcrafted features.
- **DBN-THFL.** A variant of CNN-THFL framework which adopted DBN to generate deep learning features.
- **CNN.** A SDP method that extracts deep learning-generated features via standard CNN.
- **CNN-TCA.** A variant of CNN that applies TCA to obtain transferable CNN-learned features.
- **CNN-DP.** A state-of-the-art SDP method that is an improved version of CNN proposed by [4].

Regarding the implementation of DBN, we adopted the same network architectures and parameters as in [10], i.e., 10 hidden layers and 100 nodes in each hidden layer. When implementing CNN, referring to [4], we set the batch size as 32, the epoch number as 15, the embedding dimension as 30, the number of hidden nodes as 100, the number of filters as 10, and filter length as 5. To make fair comparisons, we followed the same code-parsing process to generate integer vectors for neural networks. For TCA, we used the source code provided by its author [15]. For LR, we used the same implementation of *LogisticRegression* in *sklearn.linear_model*, and we adopted default parameters settings by *sklearn*. Considering the process of random oversampling and batch shuffle involve randomness, we conducted each method 20 times, recording their average result of MCC.

V. Results

Because some data sets tend to produce over- or under-performing models, we adopted the Scott-Knott ESD [24], [25] test to compare the performance of the methods we examined (see Figure 3). The Scott-Knott ESD test used here is a mean comparison method that leverages hierarchical clustering to divide a set of MCCs into statistically distinct groups with non-negligible differences. The approach of this test consists of two steps: (1) finding the partitions that maximize the MCC means between groups, and (2) splitting into two groups or merging them together in one group. A detailed description of the Scott-Knott ESD test can be seen in [24]. We can use the *sk_esd* function of the *ScottKnottESD*[3] R package to implement the test easily and quickly.

[3]https://github.com/klainfo/ScottKnottESD

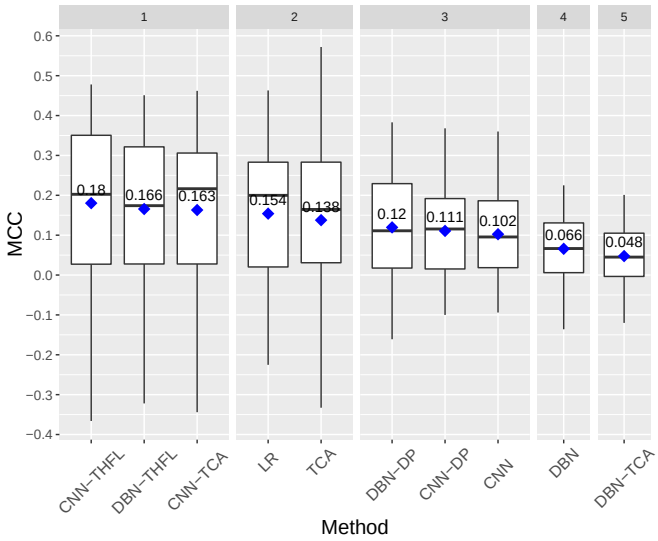


Fig. 3: The Scott-Knott ESD ranking of 10 different methods across the 72 CPDP tasks. The blue diamond indicates the average MCC of our studied methods.

Figure 3 presents the Scott-Knott ESD test of the MCC results (including for 72 CPDP tasks). By comparing 9 referential methods, the average MCC of CNN-THFL was 0.18, which respectively outperformed DBN-THFL, CNN-TCA, LR, TCA, DBN-DP, CNN-DP, and CNN by 8.9%, 10.5%, 17.1%, 30.1%, 50.8%, 62.8%, and 76.1%.

The following two points can be observed from Figure 3:

1) DBN-DP and CNN-DP, which consider concatenating the deep learning-generated features with the handcrafted features, will perform better than pure DBN and CNN.

2) Better prediction performance (in terms of MCC) can be obtained by combining the THFL framework with deep-learning methods. Among them, CNN-THFL can perform better.

In summary, our CNN-THFL framework improves the performance of CPDP tasks with the consideration of distribution discrepancy between projects. It is worth mentioning that not only CNN-THFL but also DBN-THFL will achieve better performance than the methods without THFL, so we recommend using hybrid features and adapting the distribution discrepancy between projects when performing CPDP. We believe that this improvement of SDP would provide more effective help to test teams for detecting software defects and reasonably allocating test resources.

VI. Threats to Validity

A. *Implementation of DBN and DP-CNN*

In this study, we compared the DBN method [10] and DP-CNN method [4], which are the state-of-the-art deep learning-based SDP methods. Because their implementations have not been publicly released, we tried to reimplement the corresponding methods by *Pytorch* with the same network

structures and parameters. However, we still cannot guarantee that the effects of DBN and DP-CNN that we re-implemented are exactly the same as those in [10] and [4]. However, in our experiments, we used the unified processes (e.g., code-parsing and oversampling steps) and tools (e.g., *Pytorch* and LR classifier) to implement the CPDP framework. As such, our comparative experiment should be fair.

B. *Experimental Results Might Not Be Generalizable*

In our experiments, 9 open-source projects were selected from the PROMISE repository to access the CPDP methods. The experimental results of these 9 projects (72 CPDP tasks) do not represent all cases. For some other programming languages and commercial software, our proposed method might obtain better or worse results.

C. *MCC Might Not Be the Only Appropriate Measure*

In our work, we used MCC as the metric of predictive performance. In fact, other metrics (e.g., F-score and G-measure) can also be used in the SDP task. In this paper, we recommend using MCC because it can take TN into account, and it is less sensitive to the problem of imbalanced data sets.

VII. Conclusion and Future Work

In this paper, we proposed a features learning framework called CNN-THFL to improve the performance of CPDP. CNN-THFL aims to mine both transferable deep learning-generated and handcrafted features between source and target projects. An important advantage of CNN-THFL is that it explores the necessity for distribution adaptation of hybrid features that concatenate deep learning-generated features with the traditional handcrafted features in a CPDP task. CNN-THFL is robust to differences in distribution between projects. A large number of experiments on 9 projects with 72 CPDP tasks have been carried out to verify that the proposed framework can achieve better performance in terms of MCC than the advanced referential methods. In future work, we plan to investigate the up-to-date distribution adaptation method to reduce the distribution discrepancy between projects. In addition, we will try to solve the defect prediction across multiple projects with CNN-THFL.

Acknowledgment

This research was supported by the National Nature Science Foundation of China (No. 61370103), Guangzhou Produce & Research Fund (201802020006) and Zhongshan Produce & Research Fund(2017A1014).

References

[1] I. H. Laradji, M. Alshayeb, and L. Ghouti, "Software defect prediction using ensemble learning on selected features," *Information and Software Technology*, vol. 58, pp. 388–402, 2015.

[2] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, "An empirical study on software defect prediction with a simplified metric set," *Information and Software Technology*, vol. 59, pp. 170–190, 2015.

[3] Q. Song, Y. Guo, and M. Shepperd, "A comprehensive investigation of the role of imbalanced learning for software defect prediction," *IEEE Transactions on Software Engineering*, 2018.

[4] J. Li, P. He, J. Zhu, and M. R. Lyu, "Software defect prediction via convolutional neural network," in *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pp. 318–328, IEEE, 2017.

[5] H. H. Maurice, "Elements of software science (operating and programming systems series)," 1977.

[6] T. J. McCabe, "A complexity measure," *IEEE Transactions on software Engineering*, no. 4, pp. 308–320, 1976.

[7] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on software engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[8] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pp. 297–308, IEEE, 2016.

[9] Z. Xu, J. Liu, X. Luo, and T. Zhang, "Cross-version defect prediction via hybrid active learning with kernel principal component analysis," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 209–220, IEEE, 2018.

[10] S. Wang, T. Liu, J. Nam, and L. Tan, "Deep semantic feature learning for software defect prediction," *IEEE Transactions on Software Engineering*, 2018.

[11] J. Huang, A. Gretton, K. Borgwardt, B. Schölkopf, and A. J. Smola, "Correcting sample selection bias by unlabeled data," in *Advances in neural information processing systems*, pp. 601–608, 2007.

[12] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in *The 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pp. 91–100, ACM, 2009.

[13] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248–256, 2012.

[14] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *2013 35th International Conference on Software Engineering (ICSE)*, pp. 382–391, IEEE, 2013.

[15] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang, "Domain adaptation via transfer component analysis," *IEEE Transactions on Neural Networks*, vol. 22, no. 2, pp. 199–210, 2011.

[16] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, p. 9, ACM, 2010.

[17] K. M. Borgwardt, A. Gretton, M. J. Rasch, H. P. Kriegel, B. Schölkopf, and A. J. Smola, "Integrating structured biological data by kernel maximum mean discrepancy," *Bioinformatics*, vol. 22, no. 14, pp. e49–e57, 2006.

[18] X. Xia, D. Lo, S. J. Pan, N. Nagappan, and X. Wang, "Hydra: Massively compositional model for cross-project defect prediction," *IEEE Transactions on software Engineering*, vol. 42, no. 10, pp. 977–998, 2016.

[19] X. Y. Jing, F. Wu, X. Dong, and B. Xu, "An improved sda based defect prediction framework for both within-project and cross-project class-imbalance problems," *IEEE Transactions on Software Engineering*, vol. 43, no. 4, pp. 321–339, 2017.

[20] S. Qiu, L. Lu, and S. Jiang, "Multiple-components weights model for cross-project software defect prediction," *IET Software*, vol. 12, no. 4, pp. 345–355, 2018.

[21] 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.

[22] S. Qiu, L. Lu, S. Jiang, and Y. Guo, "An investigation of imbalanced ensemble learning methods for cross-project defect prediction," *International Journal of Pattern Recognition and Artificial Intelligence*, 2019.

[23] P. Baldi, S. Brunak, Y. Chauvin, C. A. Andersen, and H. Nielsen, "Assessing the accuracy of prediction algorithms for classification: an overview," *Bioinformatics*, vol. 16, no. 5, pp. 412–424, 2000.

[24] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "An empirical comparison of model validation techniques for defect prediction models," *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 1–18, 2017.

[25] S. Herbold, "Comments on scottknottesd in response to" an empirical comparison of model validation techniques for defect prediction models"," *IEEE Transactions on Software Engineering*, vol. 43, no. 11, pp. 1091–1094, 2017.