A Semantic Convolutional Auto-Encoder Model for Software Defect Prediction

Zhihan Wang[1] and Lu Lu[1,2*]

[1]School of Computer Science and Engineering South China University of Technology Guangzhou, China
[2]Modern Industrial Technology Research Institute, South China University of Technology, Meizhou, China
[*]Corresponding author email: lul@scut.edu.cn

***Abstract*—In traditional software defect prediction, previous researches mainly focused on manually designing complex features and building classifiers based on features. However, such traditional features often fail in capturing rich syntactic and semantic information in programs. Thus, an effecient prediction model is unable to be constructed in some cases. In this study, a framework called semantic convolutional auto-encoder (SCAE) is proposed to effectivelly extract semantic features from source code. Token vectors are extracted from Abstract Syntax Trees (ASTs) of programs and then encoded as numerical vectors. Convolutional auto-encoder (CAE) can learn semantic features from the numerical vectors by decreasing the reconstruction error between input and output. After that, the CAE-based features are utilized to train a classifier. To enhance the transferability of CAE-based features for different projects, we perform domain adaptation by matching kernel embedding of layer representations across domains in reproducing kernel Hilbert spaces. Extensive experimental results verify that the SCAE yields referential methods on ten open-source projects.**

***Keywords*—Software defect prediction, Transfer learning, Semantic feature learning, Convolutional auto-encoder**

I. Introduction

As one of the research hotspots in software engineering, software defect prediction (SDP) technology can detect potential bugs in an application, then help developers assign limited testing resources and effectively enhance software reliability. An ideal defect prediction model can accurately determine whether there are defects in program, and thus plays an important role in improving software quality, shortening the development cycle, and decreasing maintaining costs.

Traditional software defect prediction (SDP) technologies utilize features representing software complexity and software scale [1] to find the potential defects in program such as Halstead metric [2], which was based on the number of operators and operands in program, and McCabe metric [3], which was calculated by loop complexity. Besides, CK metric [4] and other object-oriented metrics [5] were also adopted for defect prediction. Previous researchers mainly focused on elaborately designing and choosing handcrafted features that reflect the software characteristic for better performance of defect prediction. Asad et al. [6] evaluated the quality of software design and presented several coupling metrics for defect prediction. Conducting an empirical study for 32 feature selection methods, Xu et al. [7] showed that the selection of features has great effects on classification performance.

A key issue in SDP is the extraction of the syntax and semantic information from program. Deep learning technique is also used in the latest studies. Wang et al. [8] and Li et al. [9] leverage deep belief network (DBN) and convolutional neural network (CNN) for semantic feature generation respectively. However, compression of higher features is often accompanied by the information loss, which may cause a bad effect on making classifications. As an unsupervised method in deep learning, auto-encoder (AE) can automatically learn features from a large amount of unlabeled data and is widely used in various tasks [10], [11]. The convolutional auto-encoder (CAE) is an effective variant of the basic auto-encoder, and can reserve as much program semantics as possible by reconstructing input. The convolution and pooling operations in CAE can capture local patterns more effectively during the period of feature generation. When transfering knowledge from a labeled source domain to an unlabeled target domain, different domains may exhibit distributional discrepancy in transfer learning, which caused cross-domain knowledge adaptation problems [12]. In cross project defect prediction, distribution discrepancy between different projects also hinders the transerability of semantic feature, and degrades the performance of classifier.

In this work, a framework named semantic convolutional auto-encoder (SCAE) is developed, which uses the convolutional auto-encoder to effectively capture semantic information from ASTs of program. Specifically, we first parse source code into ASTs, then extract AST node to form token vectors and map token into number according to the mapping table. The embedded numerical vectors are fed into convolutional auto-encoder. By decreasing the reconstruction error between input and output, the CAE model can automatically learn high-level representation of input. To bridge the substantial distributional discrepancy

DOI reference number: 10.18293/SEKE2020-036

between different projects, a domain confusion loss based maximum mean discreapancy (MMD) is introduced in feature extraction. Enhancing the semantic feature transferibility generalizes the model of SCAE to the domain adaptation scenario. Evaluating the proposed approach on ten open-source java projects, experimental results indicate that the SCAE can improve the performance on both within project defect prediction (WPDP) and cross project defect prediction (CPDP).

This work makes the following contributions:

- To capture semantic information hidden in ASTs, convolutional auto-encoder is utilized for a better feature generation in defect prediction by reconstructing the input and output.
- Considering the domain discrepancy, we optimized the model of CAE by minimizing the distributional discrepancy between source and target projects to obtain transferable semantic features.
- For WDDP and CPDP, extensive experiments are conducted to verify the effectiveness of our method, and the results shows that our approach can enhance the classification performance of SDP.

The rest of this paper is organized as follows. The related work of SDP is described in Section II. Section III illustrates the proposed approach and the experiment setup is given in Section IV. The proposed approach is evaluated and the experimental results are analyzed in Section V. Section VI presents the threats to validity. In Section VII, we present the conclusion of this work and possible directions in the future.

II. Related work

Over the past few decades, software defect prediction becomes a hotspot research area in software engineering and there are many researches in the literature [13], [14]. Traditionally, most researchers mainly leverage traditional features for defect prediction, such as Halstead features [2], McCabe features [3], CK features [4], etc. The selection of features plays a vital role in classification performance [7] and Jacob et al. [15] also proposed a method to identify and remove redundant features for SDP. Ni et al. [16] investigated multi-objective features selection for SDP and proposed a method taking feature selection and construction of prediction models into account. Moreover, different algorithms of classification in machine learning are also utilized to build classifiers for defect prediction. Support Vector Machine [17], Logistic Regression [9], Naive Bayes [18] were leveraged to build classification models in SDP respectively. Researchers also build prediction models on CPDP, in which the training set and the test set are from different projects. Turhan et al. [19] took the distance of different data into account, and proposed nearest neighbor filter to remove irrelevant instances in source project. Considering the difference of data distribution, Nam et al. [20] presented an approach, i.e. TCA+, using Transfer Component Analysis (TCA) to make the source and target project distributions similar in feature space. To improve CPDP, Qiu et al. [21] constructed an ensemble classifier for the target project, which is trained on multiple components of source project data.

Recently, several deep learning techniques are also applied to build models for defect prediction. Wang et al. [8] utilized a deep learning technique, i.e. Deep Belief Network, to learn the semantic representation in source code of program. Convolutional Neural Network were leveraged [9] for defect prediction, and achieved significant results on seven open-source projects in terms of F1-score. Liang et al. [22] also employed Long Short Term Memory (LSTM) network in defect prediction. Although previous studies [8], [22] utilized neural network to learn nonlinear high-dimensional features for CPDP, but they overlooked the impact of domain discrepancy existing in different projects. Besides, convolution and pooling operations were also taken advantage of [9], but the proposed method in this work mainly has the following two differences. Firstly, the CAE does not require the information of label in the period of semantic feature generation. Secondly, we perform extensive experiments for CPDP in this work.

III. The proposed approach

As shown in Figure 1, the proposed approach consists of the following major three steps: 1) Parsing source code into ASTs; 2) Mapping tokens and embedding vectors; 3) Using convolutional auto-encoder to generate high-level semantic features and making classifications.

A. Parsing source code

The proposed approach takes source code files as input, and we need to transform the input to learn semantic information. It is proved that ASTs can be successfully applied to various tasks. In this paper, we firstly transform source code into ASTs so as to perform the following steps. Source files are parsed into ASTs and appropriate token nodes are selected from ASTs to generate token sequences. Following this work [9], four types of AST nodes are mainly chosen for both WPDP and CPDP.

B. Encoding Tokens and word embedding

In part III-A, each source file is parsed into a token sequence. Since the proposed model requires numerical vectors as input, each token sequence needs to encoded as numerical vectors based on the mapping table. Specifically, a mapping relation between tokens and integers is constructed, which means that for every token in token sequences, it has a unique integer identifier. According to the mapping table, the token sequence can be converted into an integer vector. In this way, different integer vectors may differ from each other in length, so zero is appended at the end of integer vectors to keep the same length with the longest vector. Following the work [8], infrequent tokens that occurs less than three times are also filtered out during this process. Word embedding is also performed

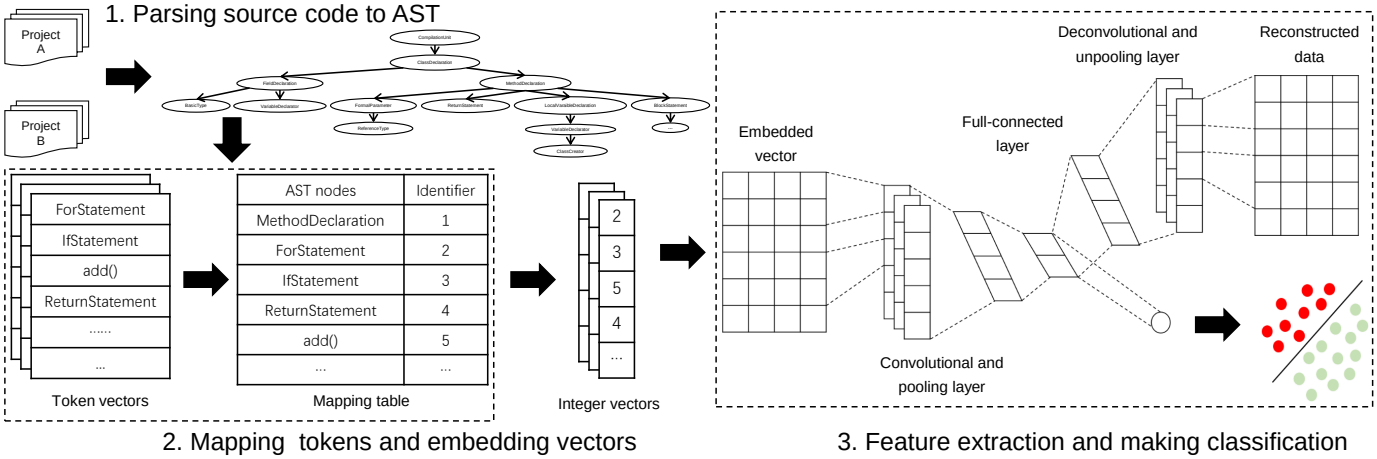

Fig. 1. The framework of SCAE

before training the model. It is difficult to draw the distance of different AST nodes only using an integer index. Word embedding technique maps a token to a fixed-length vector, and tokens in similar context tend to have similar representations. Here, the embeded size is chosen as 30, and each token is mapped to a vector whose length is 30 on the real-number field.

Similar to other classification tasks in machine learning, there also exists the issue of class imbalance in SDP. Specifically, the number of defective instances is less than the non-defective instances in a project, which makes the classifier tend to predict non-defective. Data sampling technique is also used in this work by randomly duplicating the minority class instances, i.e. Random Over-Sampling, to obtain more balanced training samples.

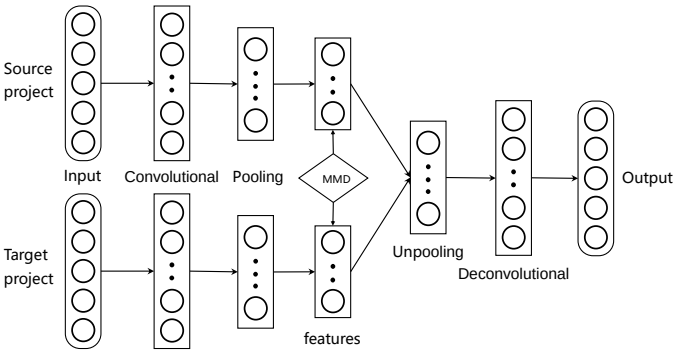

Fig. 2. Overview of CAE-based feature generation for CPDP

C. Building convolutional auto-encoder and making prediction

1) Training CAE for WPDP: Comparing with basic auto-encoder, CAE integrates the convolution and pooling operations for the advantage of CNN in feature extraction to better capture the syntactic and semantic information in program. In particular, CAE includes an encoder and a decoder. The encoder consists of a convolutional layer, a pooling layer, a fully-connected layer, and the decoder also includes an unpooling layer, a deconvolutional layer.

The embedded vector $x \in \mathbb{R}^{l\times e}$, where l and e are the length of input and the embedding size respectively, passes through the convolutional layer and the pooling layer and is encoded as the feature $y = f_e(x) \in \mathbb{R}^h$, where h denotes the feature size. During the step of decoding, the hidden representation y also moves through unpooling and deconvolutional layer, and is decoded as $\tilde{x} = f_d(f_e(x))$. The reconstruction error, $J_{ae} = \sum_{i=1}^{n} \|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|_2^2$, can be calculated by the input x and output $\tilde{x}$. In addition, a regularization item J_{wd} is added to avoid overfitting during model training. The objective function to be minimized is shown in Equation 1, where W, b denote the parameters in CAE and factors λ, γ are the tradeoff parameters for the J_{wd} penalty and J_{mmd} penalty respectively. In WPDP, we do not perform the domain adaptation, which means that the regularization hyperparameter γ is set as 0.

$$\begin{aligned}\min_{W,b} \mathcal{J}(W,b) &= J_{ae} + \lambda J_{wd} + \gamma J_{mmd} \\ &= \frac{1}{2n_s}\sum_{i=1}^{n_s} \|x - \tilde{x}\|_2^2 + \lambda \sum \|\omega\|_2^2 + \gamma MMD(X_s, X_t)\end{aligned} \tag{1}$$

CAE is an unsupervised learning method, and cannot directly be utilized for classification tasks. Therefore, an additional classifier needs to be constructed. After feature extraction, we use the CAE-based features of training set with labels to learn a classifier, and test its performance on the test set. In this work, the algorithm of Logistic Regression is utilized to established the classifier for both WPDP and CPDP.

2) Training CAE for CPDP: To promote transferability of learned features, we need to reduce the distribution difference between source and target projects. By mapping distribution X_s and X_t to a reproducing kernel Hibert space $\mathcal{H}_k$, maximum mean discreapancy (MMD) measures the mean value of these two distributions in the $\mathcal{H}_k$. Figure 2 shows the CAE-based feature generation for CPDP, and

the MMD loss, i.e. J_{mmd}, is to be minimized during model training. The MMD distance can be resolved as follows,

$$\begin{aligned} MMD(X_s, X_t) &= \left\| \frac{1}{n_s} \sum_{i=1}^{n_s} \phi(\boldsymbol{x_i}) - \frac{1}{n_t} \sum_{i=1}^{n_t} \phi(\boldsymbol{x_i}) \right\|_{\mathcal{H}_k}^2 \\ &= \frac{1}{n_s^2} \sum_{i=1}^{n_s} \sum_{j=1}^{n_s} k(\boldsymbol{x_i^s}, \boldsymbol{x_j^s}) + \frac{1}{n_t^2} \sum_{i=1}^{n_t} \sum_{j=1}^{n_t} k(\boldsymbol{x_i^t}, \boldsymbol{x_j^t}) \\ &- \frac{2}{n_s n_t} \sum_{i=1}^{n_s} \sum_{j=1}^{n_t} k(\boldsymbol{x_i^s}, \boldsymbol{x_j^t}) \end{aligned} \tag{2}$$

where $\phi(\cdot)$ is a nonlinear feature mapping function on the reproducing kernel hilbert spaces $\mathcal{H}_k$. The most important property is that $MMD(X_s, X_t) = 0$ when $X_s = X_t$. The characteristic kernel function $k(\boldsymbol{x_i}, \boldsymbol{x_j}) = \langle \phi(\boldsymbol{x_i}), \phi(\boldsymbol{x_j}) \rangle$ calculates the dot product of $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ in $\mathcal{H}_k$. In this study, we adopted Gaussian kernel: $k(\boldsymbol{x_i}, \boldsymbol{x_j}) = exp(-\left\|\boldsymbol{x_i} - \boldsymbol{x_j}\right\|^2 / 2\sigma)$, and σ is the kernel width.

IV. Experiments

This section describes the experiment setup in detail. Extensive experiments are carried out on ten open source project (listed in Table I) to evaluate effectiveness of the proposed approach. Here are two research questions:

- *RQ1: Does the semantic features extracted by the proposed approach outperforms traditional features in WPDP?*
- *RQ2: Does the distribution adaptation method can improve the performance of CPDP?*

TABLE I
The description of projects from PROMISE repository

Project Name	Versions	Avg. files	Avg. Bug Rate(%)
ant	1.6, 1.7	521	21.4
camel	1.4, 1.6	888	18.5
ivy	1.4, 2.0	284	7.2
jedit	4.0, 4.1	269	21.2
lucene	2.2, 2.4	272	60.7
poi	2.5, 3.0	391	63.9
synapse	1.0, 1.1	190	20.0
velocity	1.5, 1.6	218	49.1
xalan	2.4, 2.7	759	61.5
xerces	1.3, 1.4	370	35.0

A. Dataset

To evaluate the performance of the approach, experiments are performed on ten different open-source projects from the PROMISE repository which has been used in prior studies [8], [22]. Table I shows the necessary information about projects in experiments, including project name, project versions (including the training set and the testing set), the number of files and the bug rate. Traditional features for these projects are also widely used in conventional defect prediction [13], [20], [21]. Besides, the projects in the dataset have different data sizes, i.e. different numbers of files and defect rates, and we assure that it can verify the generalization of the model.

B. Evaluation metric

In this study, F1-score (also F-measure) is adopted to assess the performance of prediction. As a composite measure, F1-score is the harmonic average of precision and recall, which has been widely used in binary classification.

C. Baseline of methods

The proposed approach is compared with the following methods, LR based on 20 traditional features, and other two deep learning methods, including DBN and CNN. Besides, NNFilter, TCA and TCA+ are also performed for CPDP to verify the validity of domain adaptation method.

- **LR:** A logistic regression classifier is built based on 20 traditional features.
- **DBN [8]:** This method utilizes Deep Belief Network to capture semantic information in source code.
- **CNN [9]:** Supervised Convolutional Neural Network is utilized to extract semantic features from ASTs of programs.
- **NNFilter [19]:** This method uses KNN algorithm to select instance from multi source projects .
- **TCA [23]:** Transfer Component analysis, the state-of-the-art method in transfer learning.
- **TCA+ [20]:** This method optimizes the normalization process of TCA for enhancing CPDP.

When implementing these two deep learning algorithms, their optimal parameters are selected according to their papers and we ensure that the experiments are conducted with the same data processing for a fair comparison, including parsing source code into ASTs, mapping tokens, as well as data imbalance preprocessing. The proposed model is built from a training set using Adam as the optimizer. Same network architectures and parameters are adopted when implementing CAE. We set the epoch size as 30, the batch size as 16, the filter size as 3, the number of filter as 10, the number of nodes as 20.

V. Results and analysis

This section presents the results of experiments conducted on PROMISE dataset. According to analysis of the results, we answer the two questions raised in Section III.

RQ1: Does the semantic features extracted by the proposed approach outperforms traditional features in WPDP?

With 20 traditional features, a classifier is constructed using Logistic Regression algorithm for WPDP. We train the model with the instances from an old version of project, and make classifications on a new version of the same project. Additional two deep learning methods, i.e. DBN and CNN are also performed to extract semantic features. Conducting experiments on ten projects as listed

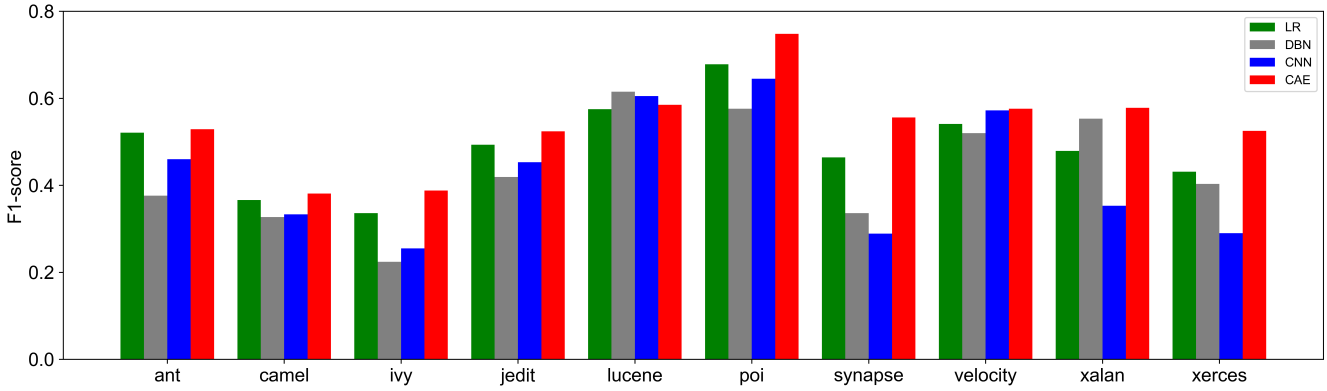

Fig. 3. F1-scores of traditional method using LR and deep learning methods using DBN, CNN, CAE respectively.

in Table I, these three methods are compared with our method, built on CAE-based features using LR algorithm.

As we can see from Figure 3, our approach can obtain higher F1-scores in most cases. Taking project synapse as an example, we use 1.0 version for model training, 1.1 version for testing. The F1-score are respectively 0.464, 0.336, 0.289, 0.556 for LR, DBN, CNN and CAE, and the CAE can obtain higher F1-scores than other methods on project synapse. The average of F1-scores over ten projects are 0.488, 0.435, 0.426 and 0.539 for these four methods, and CAE outperforms 10.36%, 23.94%, 26.67% than LR, DBN and CNN on F1-score respectively.

RQ2: Does the proposed distribution adaptation approach for CAE can improve the performance of CPDP?

In CPDP, the domain adaptation is performed on CAE, and we call it CAE+. CAE and CAE+ are compared with other three cross-project defect prediction techniques, including NNFilter, TCA and TCA+. Different from WPDP in RQ1, we only use the new version of projects for CPDP. For ten projects from PROMISE dataset, each CPDP task takes a project from these projects as a target project, and another project as source project. Therefore, 90 sets of CPDA task are constructed for each method.

Giving a target project, 9 CPDP tasks can be established respectively using the remaining 9 projects, and we calculate the average F1-score for each target project. Table II illustrates the average results of our methods and other compared approaches on PROMISE dataset. Similarly, each row in the table represents the average of F1-scores of a project, and the best result are also maked in bold. The next-to-last row reports the w/t/l, which means that our CAE+ wins w projects, ties t projects, loses l datasets, versus other methods at coressponding column. Also taking a project, e.g. xerces as an example, the highest F1-score is 0.608 achieved by CAE+. Compared with NNFilter method, CAE+ wins 9 projects, ties 0 projects, loses 1 projects, and compared with TCA method, CAE+ wins 8 projects, ties 0 projects, loses 1 projects respectively.

As Table II reports, the F1-score of CAE and CAE+ is 0.503 and 0.521, which indicates that our proposed models obtains better performances for CPDP. The CAE+ outperforms LR, NNFilter, TCA, TCA+, DBN, CNN, and CAE by 8.8%, 11.3%, 4.4%, 13.8%, 17.9%, 26.2%, and 3.6% respectively. The results also proves that, in general, domain adaptation method can enhance the performance of CPDP.

VI. Threats to validity

A. Implementation of compared methods

The proposed model is compared with other deep learning methods, DBN [8] and CNN [9]. These two methods are reproduced according to their papers. However, we can not guarantee that all the implementation details have been taken into account. Considering the randomness involved in batch shuffle, we repeated the experiment ten times, recording the average of F1-score.

B. Pamameter selection

During the training of the proposed model, we adjust the hyperparameters of CAE to get promising performance. Considering the large space of parameters, experiments cannot be done on all combinations of parameters, which may make a difference in experimental results.

VII. Conclusion

In this work, a framework of SCAE is proposed to extract semantic features from source code for defect prediction. In particular, token nodes are deliberately selected from ASTs of program. Each token in token sequence is mapped into an integer, and then word embedding is performed on numerical vector. The data then are fed into the convolutional auto-encoder to capture the intermediate representation of syntactic and semantic information of source code. Considering the distributional discrepancy of semantic representations between source and target project, an additional domain loss item is introduced during feature generation in CPDP. Conducting experiments on ten open-source PROMISE projects, the results prove

TABLE II
AVERAGE CROSS-PROJECT DEFECT PREDICTION RESULTS ON PROMISE DATASET

Target	LR	NNFilter	TCA	TCA+	DBN	CNN	CAE	CAE+
ant	0.49	0.487	0.464	0.387	0.336	0.38	0.483	**0.495**
camel	0.348	0.348	0.343	0.341	0.312	0.278	0.32	**0.349**
ivy	0.27	0.275	0.265	0.21	0.176	0.227	**0.291**	0.263
jedit	0.452	0.445	0.448	0.211	0.337	0.378	0.445	**0.471**
lucene	0.603	0.592	**0.664**	0.646	0.582	0.528	0.644	0.646
poi	0.611	0.605	0.627	0.602	0.625	0.573	**0.699**	0.675
synapse	0.49	0.478	0.456	0.468	0.396	0.384	**0.501**	0.494
velocity	0.53	0.493	0.56	0.514	0.435	0.429	0.542	**0.564**
xalan	0.49	0.49	0.64	0.629	**0.663**	0.545	0.643	0.647
xerces	0.509	0.47	0.523	0.576	0.562	0.405	0.466	**0.608**
CPCAE:(w/t/l)	9/0/1	9/0/1	8/0/2	9/1/0	9/0/1	10/0/0	7/0/3	
Average	0.479	0.468	0.499	0.458	0.442	0.413	0.503	**0.521**

that the proposed SCAE can improve performance on both WPDP and CPDP in terms of the evaluation metric, i.e. F1-score. In future work, we plan to explore other domain adaptation methods for CPDP, and our future investigation involves applying the proposed approach to more projects.

VIII. Acknowledgements

This work is supported in part by the National Nature Science Foundation of China (No. 61370103), Guangzhou Produce & Research Fund (201902020004) and Meizhou Produce & Research Fund (2019A0101019).

References

[1] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: current results, limitations, new approaches," *Automated Software Engineering*, vol. 17, no. 4, pp. 375–407, 2010.

[2] M. Halsted, "Elements of software science (operating and programming systems series)," 1977.

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

[4] R. Subramanyam and M. S. Krishnan, "Empirical analysis of ck metrics for object-oriented design complexity: implications for software defects," *IEEE Transactions on Software Engineering*, vol. 29, no. 4, pp. 297–310, 2003.

[5] 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.

[6] A. A. Asad and I. Alsmadi, "Evaluating the impact of software metrics on defects prediction. part 2." *Computer Science Journal of Moldova*, vol. 22, no. 1, 2014.

[7] Z. Xu, J. Liu, Z. Yang, G. An, and X. Jia, "The impact of feature selection on defect prediction performance: An empirical comparison," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2016, pp. 309–320.

[8] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*, 2016, pp. 297–308.

[9] 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 2017, Prague, Czech Republic, July 25-29, 2017*, 2017, pp. 318–328.

[10] T. Bao, C. Ding, S. Karmoshi, and M. Zhu, "Video anomaly detection based on adaptive multiple auto-encoders," in *Advances in Visual Computing - 12th International Symposium, ISVC 2016, Las Vegas, NV, USA, December 12-14, 2016, Proceedings, Part II*, 2016, pp. 83–91.

[11] W. Xu, H. Sun, C. Deng, and Y. Tan, "Variational autoencoder for semi-supervised text classification," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[12] M. Long, Y. Cao, Z. Cao, J. Wang, and M. I. Jordan, "Transferable representation learning with deep adaptation networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 12, pp. 3071–3085, 2018.

[13] X.-Y. Jing, S. Ying, Z.-W. Zhang, S.-S. Wu, and J. Liu, "Dictionary learning based software defect prediction," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 414–423.

[14] Z. Li, X. Y. Jing, and X. Zhu, "Progress on approaches to software defect prediction," *IET Software*, vol. 12, no. 3, pp. 161–175, 2018.

[15] S. Jacob and G. Raju, "Software defect prediction in large space systems through hybrid feature selection and classification," *Int. Arab J. Inf. Technol.*, vol. 14, pp. 208–214, 2017.

[16] C. Ni, X. Chen, F. Wu, Y. Shen, and Q. Gu, "An empirical study on pareto based multi-objective feature selection for software defect prediction," *Journal of Systems and Software*, vol. 152, pp. 215–238, 2019. [Online]. Available: https://doi.org/10.1016/j.jss.2019.03.012

[17] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "Using the support vector machine as a classification method for software defect prediction with static code metrics," in *Engineering Applications of Neural Networks*, D. Palmer-Brown, C. Draganova, E. Pimenidis, and H. Mouratidis, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 223–234.

[18] H. JI, S. HUANG, X. LV, Y. WU, and Y. FENG, "Empirical studies of a kernel density estimation based naive bayes method for software defect prediction," *IEICE Transactions on Information and Systems*, vol. E102.D, pp. 75–84, 01 2019.

[19] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 14, no. 5, pp. 540–578, Oct 2009.

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

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

[22] H. Liang, Y. Yu, L. Jiang, and Z. Xie, "Seml: A semantic LSTM model for software defect prediction," *IEEE Access*, vol. 7, pp. 83 812–83 824, 2019.

[23] 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, 2010.