Code Clone Detection via Software Visualization Representation Learning

Shaojian Qiu, Shaosheng Wang, Yujun Liang
College of Mathematics and Informatics
South China Agricultural University
Guangzhou, China
qiushaojian@scau.edu.cn

Wenchao Jiang, Fanlong Zhang*
School of Computer Science and Technology
Guangdong University of Technology
Guangzhou, China
zhangfanlong@gdut.edu.cn

***Abstract*—Code clone detection technology aims to automatically detect code similarity and help developers identify and reduce code duplication. While code syntax analysis-based methods are commonly used for clone detection, they may not capture semantic information due to bypassing the analysis of code text. To address this issue, this paper proposes a new method called visualization representation learning for code clone detection (VRL4CCD). This method converts source code fragments into grayscale images to preserve textual information and then utilizes VGG16 and a self-attention mechanism to extract features related to code semantic similarity. A siamese neural network is used to learn the similarity pattern between code features. Experimental results on the Big Clone Bench and Google Code Jam datasets demonstrate that VRL4CCD outperforms current clone detection methods regarding precision, recall, and F1-score, indicating the effectiveness of code visualization technology in clone detection tasks.**

***Index Terms*—Code clone detection, software visualization, siamese neural network, attention mechanism**

I. Introduction

During software development, programmers frequently reuse existing code fragments by copying and pasting them, known as code cloning. However, excessive code cloning can result in code redundancy, which increases the maintenance cost of the software system. Moreover, code defects in the software system can spread through code clones. In response to the above problems, researchers try to detect code clone pairs during software development and encapsulate these clone codes to simplify subsequent software maintenance.

Recently, there has been considerable interest in code clone detection methods that exploit deep learning techniques to extract structural-semantic features from code. These methods typically convert code into intermediate forms that contain both syntax and semantic information, such as Abstract Syntax Trees (ASTs), Control Flow Graphs (CFGs), and Program Dependency Graphs (PDGs) [1], [2]. However, these intermediate representations are indirect, and the effectiveness of feature extraction is heavily reliant on the completeness of the information contained in the syntax tree and semantic graph. Moreover, after conversion to these intermediate representations, the textual information in the source code is no longer utilized, which can result in the loss of valuable information needed to evaluate semantic similarity. Additionally, these methods rely on isomorphism techniques to match code subgraphs and global graphs, which can be computationally expensive and time-consuming.

To address the aforementioned issues, we aim to leverage software code visualization technology to enhance the accuracy of code clone detection. Figure 1 serves as a motivating example. It displays two actual code fragments from the Big Clone Bench (BCB) dataset. We wondered if visualizing the code would aid in illustrating the differences between the two programs. To accomplish this, we converted the ASCII decimal value of each character in the source code into rows and columns and interpreted it as a rectangular image. By visually comparing the code images, significant differences between the two programs can be observed. As a result, we conducted experiments to determine whether these characteristics contribute to clone detection.

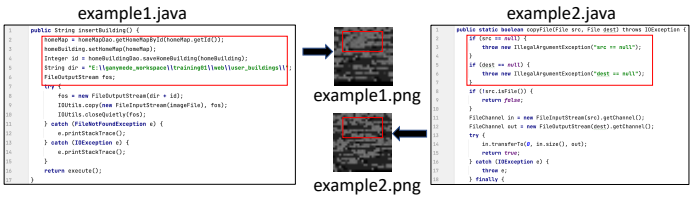


Fig. 1. Motivation Case.

In this paper, we propose a method called Visualization Representation Learning for Code Clone Detection (VRL4CCD). VRL4CCD first utilizes software visualization technology to convert each source code fragment into a grayscale image at the pixel level, preventing the loss of code information. Secondly, we uniformly reshape the generated code images to a standard size. For images that are smaller than the standard size, we pad them with 0 values. For images that are larger than the standard size, we reshape the code image directly. We then feed the code images of the cloned pairs into a siamese neural network fused with Visual Geometry Group 16 (VGG16) and attention modules to extract semantic similarity features. Finally, we perform a clone detection task using the features generated by VRL4CCD.

* is the corresponding author.
DOI reference number: 10.18293/SEKE23-226

The main contributions of our work are as follows:

- We explore the feasibility of code clone detection based on software visualization technology.
- We propose the VRL4CCD method, a code feature extraction method based on code images and siamese neural networks with attention mechanisms.
- We conduct experiments based on two open code clone detection datasets, BCB and Google Code Jam (GCJ). The experimental results show that the precision, recall, and F1-score of VRL4CCD are superior to many current code clone detection methods.

II. Related work

A. Code Clone Detection

According to the degree of similarity, code clone pairs can be broadly classified into four types [3]. Type-1 (T1) refers to code fragments with the same syntax except for comments and white spaces [4]. Type-2 (T2) corresponds to code fragments with the same grammatical structure but different identifiers, constants, and types [5]. Type-3 (T3) represents code fragments that have undergone modifications after copying, such as changing, adding, or deleting a few statements [6]. Type-4 (T4) denotes code fragments that perform the same function but are implemented using different syntactic constructs, such as bubble sort and quick sort [7]. Since the distinction between T3 and T4 is often ambiguous, researchers have further classified them into three types: strongly type-3 (ST3), moderately type-3 (MT3), and weakly type-3/type-4 (WT3/T4) [8].

With the rise of machine learning and deep learning, many researchers have turned to these techniques for code clone detection. Fang et al. [9] proposed a joint code representation that combines fusion embedding to learn the hidden syntactic and semantic features of source code. Tai et al. [10] developed a tool called CDLH, which utilizes binary Tree-LSTM [11] to encode ASTs and hash functions to optimize the distance between AST vector pairs using hamming distance. Wang et al. [12] extended original ASTs by adding direct control and data flow edges and built a graph representation of programs called Flow-Augmented Abstract Syntax Tree (FA-AST).

B. Software Visualization

Visualization technology has a wide range of applications in software engineering. Tian et al. [13] studied the relationship between software architecture and source code, and they found that nearly 30% of practitioners used software architecture visualization and modeling tools. They believe using these tools can help improve system quality attributes, maintainability, and reliability. Lima et al. [14] designed a software visualization method that graphically shows how code comments are distributed and organized in a software system and interact with the user. Chen et al. proposed software visualization and deep transfer learning for effective software defect prediction (DTL-DP) [15]. DTL-DP visualizes programs as images, applies the self-attention mechanism to extract image features, and feeds the image files into a pre-trained, deep-learning model for defect prediction.

Currently, some researchers applied visualization technology in code cloning. Linsbauer et al. [16] propose a visual software reuse method that automatically extracts and combines code to achieve cloning and reuse. Kuar et al. [17] studied the application of machine learning in code clone detection and visual management. Keller et al. [18] proposed the visualization method for code clone detection, where visualization representations of source code are fed into pre-trained image classification neural networks from the field of computer vision. Inspired by these works, we aim to explore further the potential of applying code visualization technology to code clone detection.

III. Method

A. Overall Framework

The VRL4CCD method consists of three main steps: (i) generating code images, (ii) constructing a siamese neural network for code feature extraction, and (iii) detecting code clones. Specifically, we start by converting all code fragments into grayscale images. Then we build a features extraction model based on a siamese neural network that incorporates the VGG16 [19] network and efficient channel attention module (ECA) [20]. Finally, we utilize the features extracted by the network to perform clone detection.

B. Code Image Generation

The process of generating code images is illustrated in Figure 2. In this step, we cluster code fragments into different clone groups and record their corresponding indices. For code fragments of types T1, T2, ST3, and MT3, we group them based on the transitivity of the clones. Each clone pair forms a clone group for code fragments of types WT3/T4.

Next, we extract the decimal values of the ASCII encoding for each letter and symbol in the code snippet. For instance, the letter 'a' and the symbol '(' correspond to decimal ASCII values of 97 and 40, respectively. This process is repeated for all code fragments, generating a collection of ASCII sequences, each corresponding to a code fragment (as depicted in Figure 2). We then arrange these decimal values into a square matrix and convert it into a grayscale image. Each pixel in the resulting image represents a decimal value in the original sequence, with grayscale values ranging from 0 to 255.

To enable clone detection, we transform code snippets into grayscale images with a size of 105x105x1, which can accommodate up to 11025 pixels. This size is sufficient to represent most code snippets in the BCB and GCJ datasets. Although we could have used the standard input size of VGG16, 224x224, it would have included redundant information and slowed down network training. Therefore, we chose a smaller size of 105x105 to capture more effective semantic and structural features within an appropriate sequence length. We opted for a square image because DTL-DP [15] demonstrated that the network's average prediction result improves when the image is closer to a square shape.

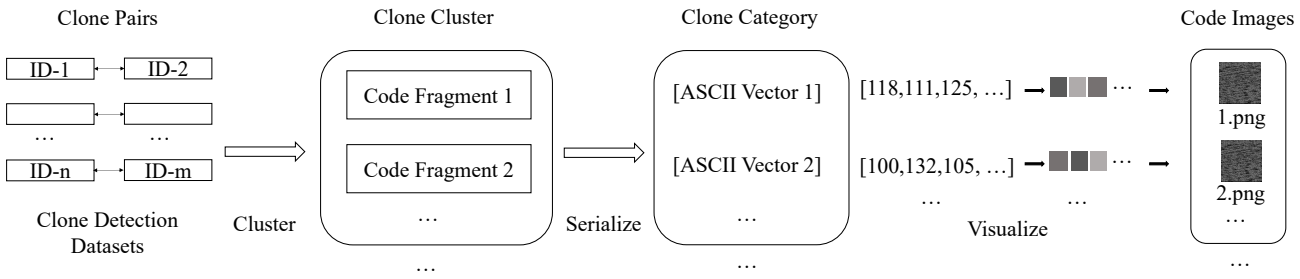


Fig. 2. Code Visualization Process.

C. Network Construction

As depicted in Figure 3, we construct a siamese neural network with two VGG16 branches to extract features and use the code images generated in the previous step as inputs. After the two images pass through several convolution layers, maximum pooling layers, and RELU activation layers, they become two one-dimensional vectors, δ_1 and δ_2, each with a length of 4096. We then subtract the two one-dimensional vectors and calculate the L1-norm of interpolating the two eigenvectors, which is equivalent to finding the distance between the two vectors. Next, we perform two fully connected layers on this distance, with the second layer connected to a neuron whose result is passed through a sigmoid function to restrict the value between 0 and 1. This value represents the similarity between the two input code images.

In the VGG16 structure, we incorporate two attention mechanisms: self-attention and efficient channel attention (ECA). These mechanisms extract structural-semantic features of code images, assign weights, and highlight the differences between the two vectors. We will provide detailed descriptions of these modules later.

1) Network Backbone: We adopt VGG16 as our network's backbone, consisting of five convolutional layers. Each convolutional layer comprises two 3x3 convolution operations and one 2x2 maximum pooling layer, and each feature layer produces 64, 128, 256, 512, and 512 channels, respectively. The input image size is 105x105x1, and the output is 4096x1.

2) Self-Attention Module: To improve the clone detection performance of VRL4CCD by allowing it to focus on key features in different images of code clone fragments, we introduce a self-attention mechanism into our model inspired by the outstanding performance of self-attention in GANs [21]. The self-attention mechanism can effectively capture long-range dependencies by computing the relationship between two locations of code images without stacking many convolutional layers to establish connections between each pixel in the image and other pixels.

As illustrated in the branch of the siamese neural network in Figure 3, we incorporate the attention mechanism into the last three convolutional layers of VGG16. In the attention layer, given a feature map x, we use a 1x1 convolutional layer to linearly map the input features x, resulting in $f(x)$, $g(x)$, and $h(x)$, where $f(x) = W_f x$, $g(x) = W_g x$, and $h(x) = W_h x$, and W_f, W_g, and W_h are learned weight matrices. For example, if the width, height, and number of channels of x are W, H, and C, respectively, the size of x is [C, N], where $N = W \times H$, and the size of $f(x)$ and $g(x)$ is [C/8, N]. The transposed $f(x)$ and $g(x)$ are matrix-multiplied to obtain the auto-correlation in the features, i.e., the relationship of each pixel to all other pixels, where $S_{ij} = f(x_i)^T g(x_j)$. We then apply the softmax activation function to obtain an attention map ($\beta_{j,i}$), which indicates the degree of attention the model pays to the i-th position when generating or obtaining the feature of the j-th pixel. Next, we multiply $\beta_{j,i}$ pixel by pixel with $h(x)$ to obtain adaptive attention feature maps o. The output of this layer is o.

$$\beta_{j,i} = \frac{\exp(s_{ij})}{\sum_{i=1}^{N} \exp(s_{ij})}, o_j = \sum_{i=1}^{N} \beta_{j,i} h(x_i) \tag{1}$$

where exp is the logarithmic function, and $o = (o_1, o_2, \ldots, o_j, \ldots, o_N) \varepsilon R^{C \times N}$.

3) ECA Module: This module enhances the performance of cross-channel interaction. It allows the aggregation of convolutional features from multiple channels of code images to improve the code structure and semantic feature extraction. The module utilizes a one-dimensional convolution with a kernel size k that is adaptively determined and learns channel attention through the sigmoid function. As illustrated in Figure 3, the ECA module captures local cross-channel interactions by considering each channel and its k neighbors, which is:

$$k = \psi(C) = \left| \frac{\log_2 C}{\gamma} + \frac{b}{\gamma} \right|_{odd} \tag{2}$$

where C is the channel size, γ and b are the parameters of mapping function, and odd means to take an odd number.

In the ECA module, W_k represents the learned local cross-channel information interaction attention, which is:

$$\begin{bmatrix} w^{1,1} & \cdots & w^{1,k} & 0 & 0 & \cdots & \cdots & 0 \\ 0 & w^{2,2} & \cdots & w^{2,k+1} & 0 & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & w^{C,C-k+1} & \cdots & w^{C,C} \end{bmatrix} \tag{3}$$

where W_k involves $C * k$ parameters.

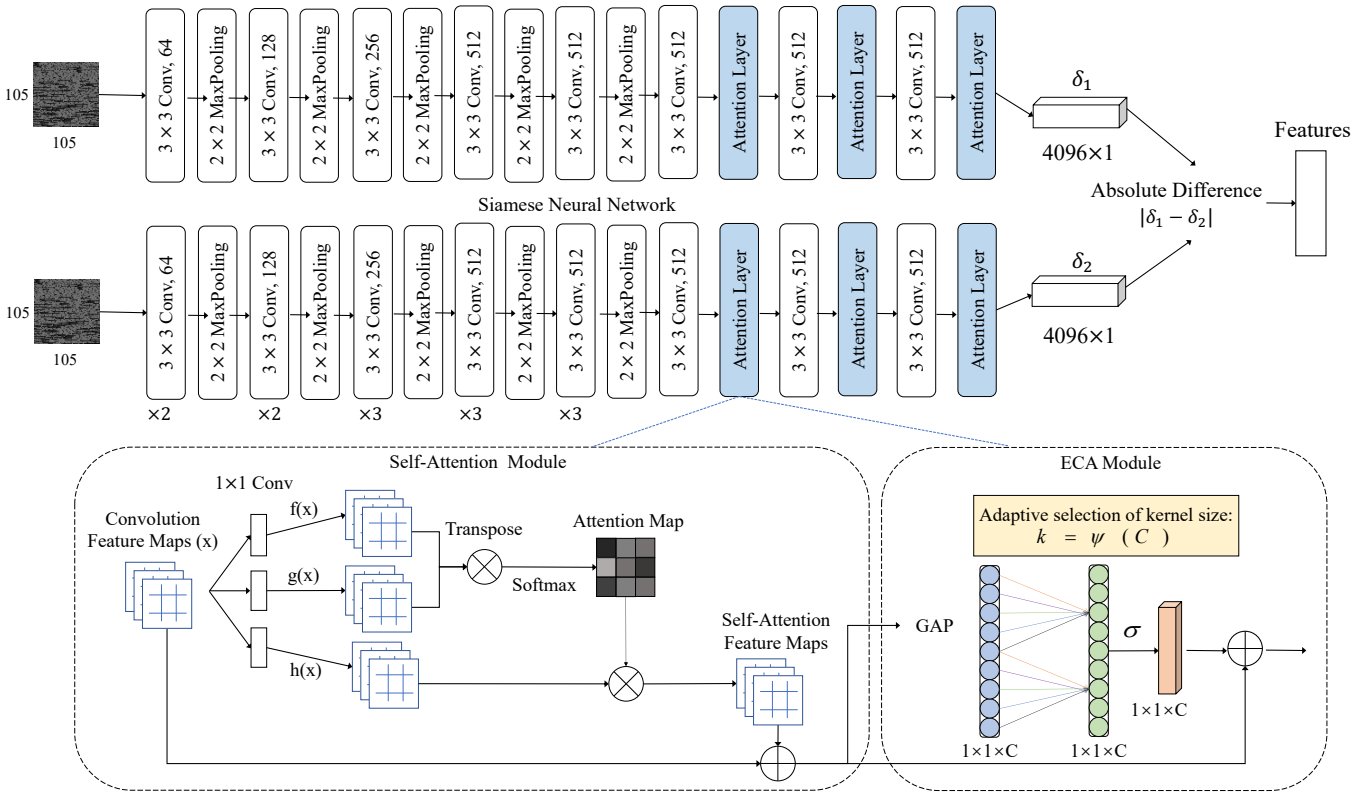


Fig. 3. The network structure of VRL4CCD.

For the weight o_j, this paper only considers the information interaction between y_j and its k neighbors. After the output of the two branches of the siamese neural network passes through two fully connected layers, we can obtain two high-dimensional vectors: δ_1 and δ_2. We subtract these two vectors and take the absolute value to obtain the final output prediction value y. Therefore, for the i-th input feature map x_i, the w_j and final output is given by:

$$w_j = \sigma(\sum_{i=1}^{k} w_j^i o_j^i), y_i = \alpha w_i + x_i \tag{4}$$

where Ω_j^k indicates the set of k adjacent channels of o_j. Where $y_j^i \in \Omega_j^k$, And the α variable is a learnable variable that is initialized to 0. Introducing α allows us to rely on nearby regions to provide informative cues and gradually assign more weight to non-local regions.

D. Clone Detection

During the training of VRL4CCD, code clone fragments with and without the clone relationship are used as inputs to the network. The convolutional and attention layers share the same parameters and extract features from the input fragments. The obtained feature maps are then fed to the fully connected layer of VGG16 for classification prediction.

During testing, we utilize the softmax function as the base classifier. Following the code visualization steps, we extract mixed features related to clone detection from each code file and input two images into the network to obtain a similarity score. A score closer to one indicates the network's inclination toward predicting the presence of clonal relationships between code fragments.

IV. Experiment Design

A. Datasets

The BCB dataset is a well-known benchmark for code clone detection tasks, comprising over 6 million true clone pairs and 260,000 false clone pairs [8] from 10 functions, where each code instance represents a Java method. The GCJ dataset [22] is a collection of Java files from Google's annual online programming competition, and we use the version curated by [12], which includes 1,669 Java files.

B. Compaired Methods

We compare our approach VRL4CCD with the following code clone detection methods:

GGNN [23] is a variation of graph neural networks that updates the node's representation by incorporating information from neighboring nodes.

ASTNN [24] utilizes recursive neural networks (RNN) to encode AST subtrees for statements, then feeds the encodings of all statement trees into an RNN to compute the vector representation for a program.

FA-AST [12] enhances original ASTs by adding control and data flow edges, building a graph representation of programs called Flow-Augmented Abstract Syntax Tree (FA-AST), and applying two different types of graph neural networks on FA-AST to measure the similarity of code pairs.

TBCCD [25] is a state-of-the-art code clone detector that uses ASTs and tree-based convolutions to measure code similarity.

C. Evaluation Indicators

To evaluate the detection performance, we utilize precision, recall, and F1-score as the evaluation metrics of our code clone detection research [26]. Specifically, if clones exist between two code fragments and the prediction result is a clone, it is referred to as a true positive

(TP). Otherwise, it is considered a false positive (FP). Similarly, if two code fragments are not clones, and the prediction result is not a clone, it is called a true negative (TN). Otherwise, it is a false negative (FN). Then, precision (P), recall (R), and F1-score can be defined as:

$$P = \frac{TP}{TP + FP} \tag{5}$$

$$R = \frac{TP}{TP + FN} \tag{6}$$

$$F1 - score = \frac{2 \times P \times R}{P + R} \tag{7}$$

TABLE I
RESULTS ON THE BCB AND GCJ DATASET

Model	BCB			GCJ		
	P	R	F1	P	R	F1
GGNN	0.72	0.89	0.79	0.72	0.87	0.79
ASTNN	0.92	0.94	0.93	0.98	0.93	0.95
FA-AST	0.96	0.94	0.95	0.99	0.97	0.98
TBCCD	0.96	0.96	0.96	0.79	0.85	0.82
VRL4CCD	0.98	0.98	0.98	0.99	0.99	0.99

TABLE II
F1-SCORE COMPARISION WITH VARIOUS CLONES TYPES IN BIGCLONEBENCH DATASET

method	T1	T2	ST3	MT3	WT3/T4
GGNN	1.0	1.0	0.79	0.70	0.60
ASTNN	1.0	1.0	0.99	0.99	0.93
FA-AST	1.0	1.0	0.99	0.98	0.95
TBCCD	1.0	1.0	0.98	0.96	0.96
VRL4CCD	1.0	1.0	0.99	0.99	0.98

V. EXPERIMENT RESULT

Table I presents our code clone detection experiments' precision, recall, and F1-score values. Since deep learning-based methods have inherent randomness, we executed each method 20 times and recorded their averages.

The table shows that our VRL4CCD outperforms the other four methods in all metrics. Specifically, on the BCB dataset, VRL4CCD achieved the highest precision, recall, and F1-score values, all exceeding 0.98, surpassing all the comparison methods. Furthermore, the results of VRL4CCD are highly stable, as the precision, recall, and F1-score values are very close.

Detecting clones in the GCJ dataset is more challenging than in BCB. However, unexpectedly, VRL4CCD showed better in detecting clones in the GCJ dataset than in BCB. This reflects the superior ability of visualization methods to detect clones at the semantic level.

Furthermore, since the results in Table I are obtained by training the model on a mixture of all clone types, it indicates that VRL4CCD can detect a specific type of clone alone and function as a unified model to detect all types of clones. This demonstrates the strong generalization ability of VRL4CCD.

Table II presents the F1-score values of the compared methods on different types of clones. It is evident that our method exhibits a strong detection effect on T1, T2, and ST3 types of clones. Furthermore, the detection performance of VRL4CCD on MT3 and WT3/T4 types is comparable to the state-of-the-art clone detection methods in recent years. The results in Table II also highlight the superior performance of VRL4CCD on ST3, MT3, and WT3/T4 types of clones. Among all the methods, better detection results are observed for T1 and T2 types of clones. For the challenging MT3 and WT3/T4 types, VRL4CCD shows a 2 to 5 percent improvement compared to other baseline methods.

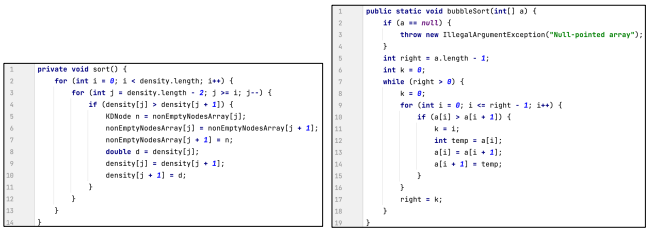
VI. DISCUSSION

A. Are VRL4CCD Suitable For Code Clone Detection?

Regarding the model structure, siamese neural networks are an excellent approach for performing image similarity discrimination tasks. Furthermore, we combined the siamese neural network with the VGG16 network and attention mechanism to enhance the model's effectiveness. The experimental results demonstrate that the principle of image processing is suitable for processing code images, and VRL4CCD already demonstrates remarkable performance even when using default parameters.

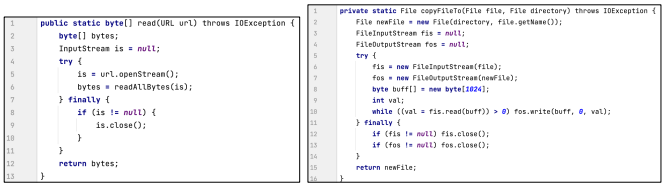
In terms of experimental results, as shown in Table I, our method achieved high precision, recall rate, and F1-score values, with VRL4CCD even reaching 0.99 on the GCJ dataset. These results reflect the effectiveness of VRL4CCD.

In practice, we demonstrate a pair of real clones (Fig.4(a) and Fig.4(b)), where both code snippets achieve file copying. Our method, VRL4CCD, can accurately identify this clonal pair. Additionally, Fig.5 displays a pseudo-cloning pair in BCB, where Fig.5(a) performs the function of URL content crawling and Fig. 5(b) completes the function of file copying. Although these two code fragments are similar at the token and statement levels, our method can still recognize that they are not clones. These two examples demonstrate that our method, VRL4CCD, can effectively learn features from the training data to discriminate between code clones.



a b

Fig. 4. An example of a true clone pair in BCB.



a b

Fig. 5. An example of a false clone pair in BCB.

B. What Is The Improvement Made by The Code Features Learned From VRL4CCD?

Although code intermediate representations, such as ASTs and CFGs, can be used for code clone detection, this approach is indirect and requires additional tools to build the intermediate representations. Once the intermediate representation is generated, the original code text is usually discarded. In contrast, VRL4CCD uses the code as the input, avoiding any information loss from intermediate representation conversion. This method extracts features related to code similarity

by mining the relationship between each character and pixel in the code.

Moreover, we improve the feature extraction process by incorporating a self-attention mechanism into the network. This allows the network to focus on the pixel-to-pixel associations, i.e., character-to-character associations in code snippets. We also introduce the ECA module to enhance the performance of cross-channel network interaction. We add the above two attention mechanisms to the bottleneck region of the network, which has the largest number of separated channels, further to improve the feature extraction capability of the network.

C. Threats to Validity

1) Implementation of Compared Methods: In our experiments, we implemented some baseline methods, such as TBCCD and ASTNN, using their open-source code available online. For the baseline methods without open-source code, we followed the details mentioned in the original paper as closely as possible to ensure their implementation.

2) Precision, Recall, and F1-Score Might Not be the Only Appropriate Measures: Although we used the most widely used metrics, i.e., precision, recall, and F1-score, to evaluate the effectiveness of code clone detection, other performance indicators could also be used, such as AUC, MCC, and G-mean.

3) Generalization of Experimental Results Might be Limited: We conducted experiments on the BCB and GCJ datasets, which have different data scales, to enhance the generalization of our method. However, we cannot guarantee that VRL4CCD will achieve similar improvements on other datasets.

VII. Conclusion

This paper explores the application of software visualization techniques to code clone detection. Our proposed clone detection method, VRL4CCD, utilizes code visualization and a siamese neural network with a self-attention mechanism to extract code similarity-related features. Our experimental results demonstrate that VRL4CCD outperforms current state-of-the-art code clone detection methods. Moving forward, we plan to conduct more code detection tasks in real-world scenarios and explore further applications of software visualization technology in the area of code representation learning.

Acknowledgement

This work is supported in part by the Guangdong Basic and Applied Basic Research Foundation (No. 2022A1515110564), in part by the Science and Technology Program of Guangzhou (No. 202201010312), in part by the Youth Innovative Talents Project of Ordinary Universities of Guangdong (No. 2020KQNCX008).

References

[1] X. Guo, R. Zhang, L. Zhou, and X. Lu, "Precise code clone detection with architecture of abstract syntax trees," in *International Conference on Wireless Algorithms, Systems, and Applications*. Springer, 2022, pp. 117–126.

[2] C. Huang, H. Zhou, C. Ye, and B. Li, "Code clone detection based on event embedding and event dependency," in *Proceedings of the 13th Asia-Pacific Symposium on Internetware*, 2022, pp. 65–74.

[3] D. Yu, Q. Yang, X. Chen, J. Chen, and Y. Xu, "Graph-based code semantics learning for efficient semantic code clone detection," *Information and Software Technology*, vol. 156, p. 107130, 2023.

[4] G. Shobha, A. Rana, V. Kansal, and S. Tanwar, "Code clone detection—a systematic review," *Emerging Technologies in Data Mining and Information Security*, pp. 645–655, 2021.

[5] A. Gupta and R. Goyal, "A study on the metrics-based duplicated code type smell detection techniques relating the metrics to its quality," in *Inventive Communication and Computational Technologies*. Springer, 2023, pp. 515–532.

[6] M. A. Yahya and D.-K. Kim, "Clcd-i: Cross-language clone detection by using deep learning with infercode," *Computers*, vol. 12, no. 1, p. 12, 2023.

[7] A. Zhang, L. Fang, C. Ge, P. Li, and Z. Liu, "Efficient transformer with code token learner for code clone detection," *Journal of Systems and Software*, vol. 197, p. 111557, 2023.

[8] J. Svajlenko, J. F. Islam, I. Keivanloo, C. K. Roy, and M. M. Mia, "Towards a big data curated benchmark of inter-project code clones," in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014, pp. 476–480.

[9] C. Fang, Z. Liu, Y. Shi, J. Huang, and Q. Shi, "Functional code clone detection with syntax and semantics fusion learning," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2020, pp. 516–527.

[10] H. Wei and M. Li, "Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code." in *IJCAI*, 2017, pp. 3034–3040.

[11] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," *arXiv preprint arXiv:1503.00075*, 2015.

[12] W. Wang, G. Li, B. Ma, X. Xia, and Z. Jin, "Detecting code clones with graph neural network and flow-augmented abstract syntax tree," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2020, pp. 261–271.

[13] F. Tian, P. Liang, and M. A. Babar, "Relationships between software architecture and source code in practice: An exploratory survey and interview," *Information and Software Technology*, vol. 141, p. 106705, 2022.

[14] P. Lima, J. Melegati, E. Gomes, N. S. Pereira, E. Guerra, and P. Meirelles, "Cadv: A software visualization approach for code annotations distribution," *Information and Software Technology*, vol. 154, p. 107089, 2023.

[15] J. Chen, K. Hu, Y. Yu, Z. Chen, Q. Xuan, Y. Liu, and V. Filkov, "Software visualization and deep transfer learning for effective software defect prediction," in *Proceedings of the ACM/IEEE 42nd international conference on software engineering*, 2020, pp. 578–589.

[16] L. Linsbauer, S. Fischer, G. K. Michelon, W. K. Assunção, P. Grünbacher, R. E. Lopez-Herrejon, and A. Egyed, "Systematic software reuse with automated extraction and composition for clone-and-own," in *Handbook of Re-Engineering Software Intensive Systems into Software Product Lines*. Springer, 2023, pp. 379–404.

[17] M. Kaur and D. Rattan, "A systematic literature review on the use of machine learning in code clone research," *Computer Science Review*, vol. 47, p. 100528, 2023.

[18] P. Keller, A. K. Kaboré, L. Plein, J. Klein, Y. Le Traon, and T. F. Bissyandé, "What you see is what it means! semantic representation learning of code based on visualization and transfer learning," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 2, pp. 1–34, 2021.

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[20] Q. Wang, B. Wu, P. Zhu, P. Li, W. Zuo, and Q. Hu, "Supplementary material for 'eca-net: Efficient channel attention for deep convolutional neural networks," in *Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, IEEE, Seattle, WA, USA*, 2020, pp. 13–19.

[21] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," in *International conference on machine learning*. PMLR, 2019, pp. 7354–7363.

[22] "Google code jam," https://code.google.com/codejam/contests.html Accessed: 2016-10-8.

[23] D. Beck, G. Haffari, and T. Cohn, "Graph-to-sequence learning using gated graph neural networks," *arXiv preprint arXiv:1806.09835*, 2018.

[24] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu, "A novel neural source code representation based on abstract syntax tree," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 783–794.

[25] H. Yu, W. Lam, L. Chen, G. Li, T. Xie, and Q. Wang, "Neural detection of semantic code clones via tree-based convolution," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 2019, pp. 70–80.

[26] Q. U. Ain, W. H. Butt, M. W. Anwar, F. Azam, and B. Maqbool, "A systematic review on code clone detection," *IEEE access*, vol. 7, pp. 86 121–86 144, 2019.