Journal of Theoretical and Applied Information Technology
15th September 2022. Vol.100. No 17

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

JAVA SOURCE CODE SIMILARITY DETECTION USING SIAMESE NETWORKS

DIVYA KUMARI TANKALA[1], Dr. T. VENUGOPAL[2], VIKAS B[3]

[1]Assistant Professor, G. Narayanamma Institute of Technology and Science, Department of CSE, Hyderabad, Telangana, India

[2]Professor, JNTUH UCE, Department of CSE, Jagitya, Telangana, India

[3]Assistant Professor, GITAM School of Technology, Vishakhapatnam, Andhra Pradesh, India

E-mail: [1]0000-0002-2842-4898, [2]0000-0001-7782-7311, [3]vboddu2@gitam.edu

ABSTRACT

Software plagiarism checkers can be important during coding competitions, to review, evaluate, and rank the participants. For a problem statement, if the number of submissions is relatively small then inspecting each code submission and being able to determine whether they are similar to the existing code segment or not is easy, but in the case of huge code submissions, it is very difficult to determine the presence of code clones in the submitted code snippet. Therefore, there is a need for plagiarism checkers to detect similar clones. In existing studies, we could use various approaches to detect code clones, but code clone detection using an abstract syntax tree is one of the popular approaches. Our proposed approach based on AST is experimented on a BigCloneBench dataset consisting of Java code fragments and implemented using recursive neural networks. The siamese networks were used to detect similarities between two code fragments and presented the influence of contrastive learning in source code clone detection with high accuracy. This paper showcases the improvement in precision, recall and F1-score at least with 5% compared to existing approaches.

Keywords: *Code similarity*, *Plagiarism detection*, *Siamese networks, Recursive Neural Networks, LSTM, Contrastive learning*

1. INTRODUCTION

Sometimes, Source Code Plagiarism checkers can help academia to evaluate students' programming assignments and judge their work. The participant's work can easily be inspected against plagiarism checkers even in coding competitions. Code clone detection is the base work to design such as plagiarism checkers. But in the industrial field, programmers look for particular source code, by which their task is highly satisfied. They will not bother about plagiarism or author details. There are too many online platforms to host projects' source code, even researchers sharing their work publicly. This makes it easy for developers to easily search for code and copy the code for their software implementations. Of course, copy-paste approach in software development has advantages and disadvantages. However, to check for the similarity in code segments, we followed an Abstract Syntax Tree (AST) based approach, employed deep learning models, and evaluated on an open-source large dataset available publicly.

Most of the research studies addressed the problem with clone detection. An existing research work [1] proposed AST-based neural networks to extract abstract syntax trees from code fragments and transform them into code vectors. The vector representation of ASTs then processed by a bidirectional Gated Rectifier Unit (GRU) to get source code embedding. Such embeddings help train recurrent neural networks and detect syntactically accurate and similar representations of source codes. Comparatively, this ASTNN approach showed better results than the previous traditional approaches. However, this model was evaluated on two publicly available datasets namely OJClone and BigCloneBench. Another model ASTNN-c [2] proposed Siamese-Networks [9] to identify code clones of the C language. In this approach, a contrastive study was provided between cross entropic loss function and contrastive loss function.

Our main contribution to work is to detect java source code similarity by supervised contrastive learning [3] by modifying the ASTNN model. Contrastive learning was mostly used in image

classification work by many researchers. The Siamese neural network architecture consists of two or more identical networks [9] that can help in the prediction of code similarities.

The structure of our paper is as follows. Section II covers related work toward the detection of similar code fragments, Section III covers methodology of proposed approach, section IV covers conclusion and limitations of work, section V gives the overall discussion of the model and its results, and finally conclusion and future enhancement of our work.

2. RELATED WORK

Code clone can be similar or identical fragments of code which refers to the duplication of source code. Basically, code clones are of 4 types [8], such as exact clone, which have no changes in code fragments except blank spaces, comments, remarks. Renamed clones, are syntactically similar clones may vary in variable names, identifier names. Near miss clones, can have changes in statements of codes such as modifications, adding or removal of statements in source codes. Semantic clone, which are syntactically different but semantically similar clones. Among them identifying semantic clones is still a challenging problem to solve. Of course, Code clone detection is not a novel task in the software engineering field. Many existing research contributions made it easy with traditional approaches. Still, there are challenges in the detection of code clones. The technique we use to transform a source code into computable form and similarity findings between code fragments.

2.1 Source Code Transformation

The major approaches followed by existing techniques were Tree-based and graph-based approaches to represent source codes. Abstract Syntax Trees (ASTs) provide lexical and syntactical information on source code [4]. Control flow graphs (CFGs) give structural or semantic information about the source code [5]. However, most of the approaches use either Syntax trees (ASTs) or Graphs (CFGs) of source codes, and few word embeddings used to represent them. But, handling a large syntax tree or huge graph to train a model may lose some sensitive semantic information. Therefore, some other optimized approaches must be followed to achieve better results.

2.2 Source Code Similarity

The existing techniques use various similarity measures to detect similarity between code vectors. The Euclidean distance metric is used to cluster the vectors. The vectors in one clone cluster can be treated as similar. Levenshtein distance is a metric for measuring the distance between two sequences, such distance is defined as the minimum edits required to transform one to the other. Some approaches use cosine similarity to identify code clones. Most of the techniques develop a model to compare and detect similar code snippets. K means clustering algorithm used to cluster the source code representations and calculates the similarities. In image classification techniques Siamese network is used to identify similar faces, which takes two inputs (images) and forms two sub-networks to process the inputs to predict similarity scores based on similarity function as:

$$S_{sim} = F_{sim}(I_1, I_2)$$ Where I_1, I_2 are inputs

2.3 Recurrent Neural Networks

Recurrent Neural networks RNN) are one of the artificial neural networks. RNNs are mainly used for Natural Language Processing tasks. To handle sequence-related problems, the issues we identify are the variable size of input/output neurons, too much computation, and no parameter sharing. RNNs address such issues by adapting to work with time-series data or data that involves sequences. In our research work code sequence data is also to be handled similarly without losing the syntactic and structural flow of code fragments Therefore, LSTM is a type of RNN used in our work, which is suitable to handle tree structures.

2.4 Siamese Neural Networks

Sometimes, learning similarities between java code clones can be useful. Siamese neural network is an artificial neural network composed of two subnetworks that give two output embeddings.

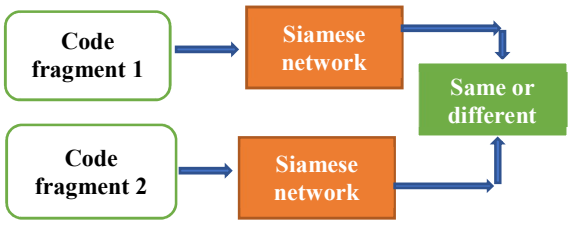


Figure 1: Example of Siamese architecture

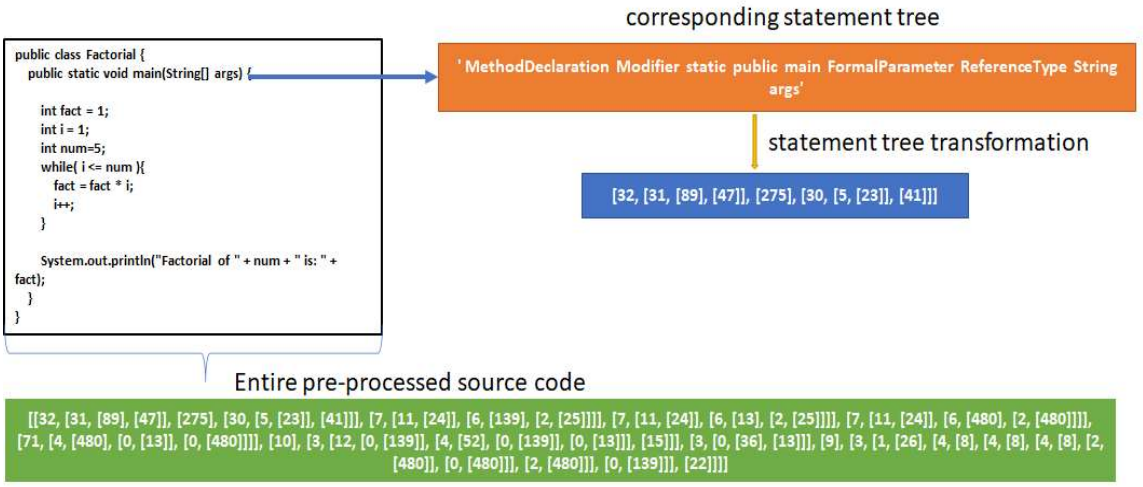


Figure 2: Preprocessing framework for the input code snippets

These embeddings will be used as inputs to a loss function. This twin encoding network shares weights but of course, the same network is used twice. Such loss function is implemented to minimize the distance between similar inputs (code fragments).

2.5 Contrastive Loss Function

Cross entropic loss or log Loss function is commonly used for classification problems to minimize the loss. While this Cross entropy works well for both binary classification and multi-classification problems, also helpful for similarity detection problems. Mostly, this approach was used for image classification to correctly separate image embeddings from various classes. Though this learning method performs well, the contrastive learning method shows better performance and predicts the distance between code fragments is shown to outperform the cross entropic method.

3. METHODOLOGY

In this chapter, the detailed approach for java code clone detection is explained in each of the sub-sections. Our proposed work is based on the ASTNN model [1]. Firstly, the dataset description is elaborated in detail, then in the second sub-section, preprocessing steps used in the ASTNN model, and in the third sub-section, the proposed approach implemented with Siamese neural networks is described with little modification of the ASTNN model.

3.1 Dataset Description

We have chosen a large dataset, the BigcloneBench (BCB) [6] dataset for Java Language. It is a public dataset benchmark for code classification and clone detection. This dataset consists of known true and false positive clones from a big data inter-project Java repository, which has been majorly used by researchers concerning clone detection. BCB dataset consists of 6 million true clone pairs and 260 thousand false clone pairs.

3.2 Data Preprocessing

We cannot feed the model directly with the dataset, so preprocessing is required. We have to transform each source code into an intermediate form, such that each code fragment is converted into a statement tree with the help of python library javalang. Instead of processing large syntax trees of code fragments, split them into small statement trees as shown in Figure 2. In this model the steps followed for preprocessing the dataset are:

Word2vec dictionary

Token	Index
'MethodDeclaration'	48
'IfStatement'	15
'ForStatement'	74
'WhileStatement'	89

Figure 3: Representation of token indexes

Step 1: Parse the source codes to extract lexical information from code fragments.

Step 2: Read pairs of code fragments (ASTs) and split them into statement sequences.

Step 3: split the data into training, validation, and testing sets, and train word embeddings using the word2vec model.

Step 4: Generate block sequences for the dataset to process the model.

We extract the tokens from each statement block and generate its corresponding vectors as shown in Figure 4, such as ForStatement, whilestatement, MethodDeclaration, etc. Word2Vec model is used to extract token indexes as shown in Figure 3. As the deep learning model cannot use actual tokens to process, token indexes of them used to work with deep learning model. In the end, to form a sequence of preprocessed statement trees as shown in figure 2, each statement block of code fragment is joined together.

3.3 Clone Detection Model

In this section, detailed description of the model given, that helps detect java code clones. Siamese networks are mostly used to predict the similarity distance between inputs such as images, texts, code fragments. Our model also implemented with Siamese networks; it takes two source codes through loss function. When training a Siamese network, two or more inputs are encoded and the output features are compared through the loss functions such as cross-entropy loss and contrastive loss.

Corresponding vector for the following tokens

```
token: ForStatement
vector: [0.8975477 0.30159312 2.0667598
0.09544923]
token: WhileStatement
vector: [ 0.2084294 -0.29155913 1.2945276 -
0.6455277 ]
token: MethodDeclaration
vector: [ 0.97751176 -2.2895794 1.3722514
1.5088714 ]
token: IfStatement
vector: [-0.677365 0.23554918 1.4510689 -
0.4594793 ]
```

Figure4: Representation of vectors for tokens

To encode the sequence of statement trees into vector representations, the statement tree encoder is used as discussed above in the 3.2 section.

Table 1: Sample representation of blocks for source code.

Sample code in Java	Blocks obtained for Source code

public class Factorial {
public static void main (String[] args) { int fact = 1;
int i = 1;
Scanner sc = new Scanner (System.in);
 	
System.out.println("Enter a number whose factorial is to be found: ");
int num = sc.nextInt();
while(i <= num){
fact = fact * i;
i++;
}
 	
System.out.println("Factorial of " + num + " is: " + fact);
}
}
	block 1: *MethodDeclaration*
	block 2: *LocalVariableDeclaration*
	block 3: *LocalVariableDeclaration*
	block 4: *LocalVariableDeclaration*
	block 5: *StatementExpression*
	block 6: *LocalVariableDeclaration*
	block 7: *WhileStatement*
	block 8: *BlockStatement*
	block 9: *StatementExpression*
	block 10: *StatementExpression*
	block 11: *End*
	block 12: *StatementExpression*

With the batch tree encoder, particular block of source code is given with a sequence of tree nodes, by performing recursive calls on the root of each statement tree, it is encoded into vector representation [1], which is represented as per the below algorithm Figure 6. Bidirectional GRU used in the model, which takes the output of the batch tree encoder as input and is used to learn the dependencies of the statement trees of actual code in blocks as per table 1, in which it transforms the syntax tree into a structure of nested arrays representing blocks of the source code, then it returns vector representation of the total source code as shown in Figure 2.

The process of retrieving the AST of a source code and splitting them into subtrees to extract the more syntactic and semantic information about the source code is mentioned in the model [1]. Few modifications were made to this model to predict similar java source codes. As stated earlier Siamese networks are used for prediction and contrastive learning is used to minimize the loss for comparison of input code fragments.

Journal of Theoretical and Applied Information Technology
15th September 2022. Vol.100. No 17

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

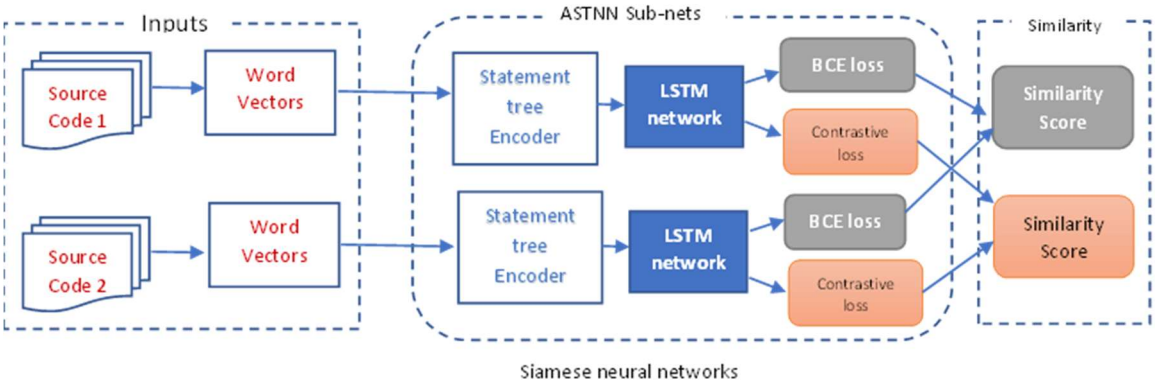


Figure 5: Working process of Siamese networks in detection of similar source codes

Contrastive loss takes the output from the twin network for a code fragment and calculates its distance from the same class (True Positive) of code fragments and contrasts with the distance to another class (False positive or False-negative or True negative) of code fragments. Hence, instead of the binary cross entropic loss function, we used the contrastive loss to predict the relative distance between two code fragments (inputs).

The contrastive loss function is defined as L:

$$Mean(T + D^2 \times (1 - T) \times (\max(0, m - D))^2) \quad \text{-----(1)}$$

Where T is True values, D is the labels we used for training, and m is the margin value provided for contrastive function.

Algorithm: Batch Tree Encoder

Input: Array of Statement trees, A
Output: Vectors for batched Statement Trees, BV
Function BatchProcess(ns, ids) //ns=node sequence, ids=their indexes
Initialize two array lists C, CI ← φ to record child nodes and their batch indexes;
for each node ∈ ns do
for each children node, the child node does group the child by its position, and records the child to C and its batch index to CI;
end for

Figure 6: Algorithm for batch tree encoder [1].

3.4 Evaluation Metrics

The evaluation metrics used in our approach are precision, recall, and F1-score are given below:

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + Fa \quad Positve} \quad \text{-------(2)}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + Fals \quad Negative} \quad \text{-------(3)}$$

$$\text{F1-Score} = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad \text{-------(4)}$$

While evaluating the model true positives, true negatives, false positives, and false negative values can be made.

Siamese network takes input source codes and generates statement trees for them as per Figure 2. Then, these statement trees are passed to the statement tree encoder which outputs vector for each input source code. The Figure 6 shows the overall working process of Siamese networks to detect java code clones. For each code vector, ASTNN based sub network formed, which is tree-based LSTM network to process the code vector. Both binary cross entropic loss and contrastive loss scores compared to get best evaluation.

3.5 Evaluation of Model

In this section, extensive evaluation of the approach used in this work is explained. Basically, the model is evaluated by splitting the dataset into 3 sets, 60% for a training set, 20% for a validation set, and 20% into a test set. The model performance evaluated with the specific experiments on java code fragments and also help to show the performance between binary cross-entropy and contrastive learning.

3.5.1 Performance with Java Source codes

We evaluated our model using two learning approaches called binary-cross-entropy and contrastive learning on the test dataset with respect to precision, recall, and F1- Score. In table 2, the performances with these learning approaches are shown. We used the top performance (F1-score) of the model and threshold distance used to predict similar input source codes. The figures 7,8 show confusion matrices for the model evaluation.

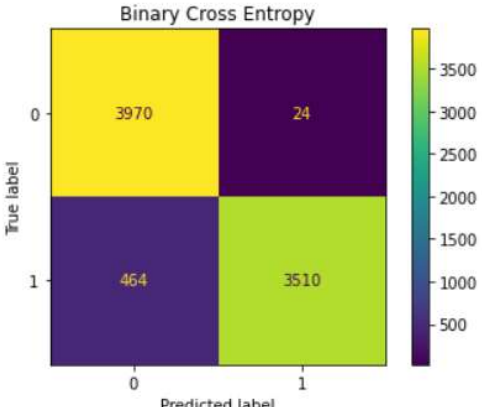


Figure 7: Confusion matrix of the model using Binary Cross Entropy learning approach

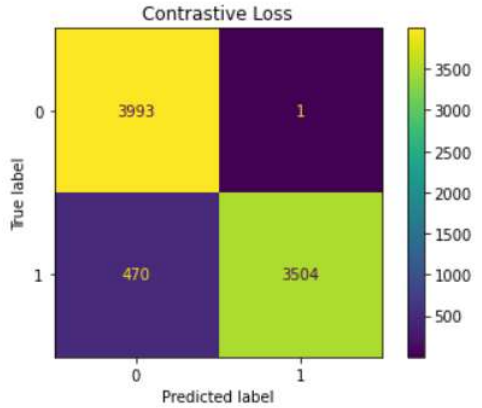


Figure 8: Confusion matrix of the model using Contrastive learning approach

Table 2: Performances with different learning approaches

Approach	Precision	Recall	F1-Score
BCE	0.993	0.884	0.936
Contrastive	1.000	0.993	0.938

We calculated the precision, recall, and F1-score of the models mentioned in Table 3, on the test dataset and compared them.

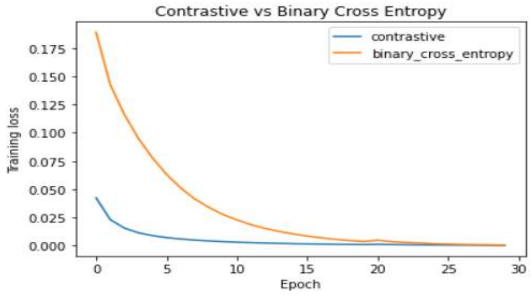
Table 3: Performances of different ASTNN models

Model	Precision	Recall	F1-Score
ASTNN	0.972	0.899	0.935
ASTNN-C	0.975	0.992	0.983
ASTNN-Java	1.000	0.993	0.938

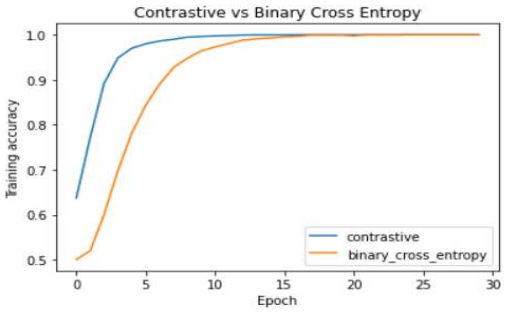
As per the results mentioned above, Contrastive learning performs better than the cross-entropic learning approach.

3.5.2 Results and discussion

We also recorded training and validation loss. The following graphs show that contrastive learning converges faster than binary cross-entropic training. However, contrastive learning loss value may seem lower but this is because loss is calculated differently for each type of learning. As a result of converging faster than binary cross entropic learning, contrastive learning reaches higher training accuracy values faster than binary cross entropic training.



(a)



(b)

Figure 9: (a) Graph represents Loss Vs Epochs (b) Graph represents Accuracy Vs Epochs

3.5.3 Interface Design

The user interface can be implemented with HTML and CSS. Flask is a web framework, which is a python module used to develop web applications. Our application runs using the ngrok website. Create an app instance to handle web requests and send responses to the user. ngrok is a cross-platform application that enables developers to expose a local development server to the Internet with minimal effort. The following figures are shown for predicting java code clones with the threshold distance of 0.5. We can upload two code files to predict the similarity between them using Siamese networks as shown in Figure 9. We can

change the margin and threshold values to predict accurate results.

4. CONCLUSION AND LIMITATIONS

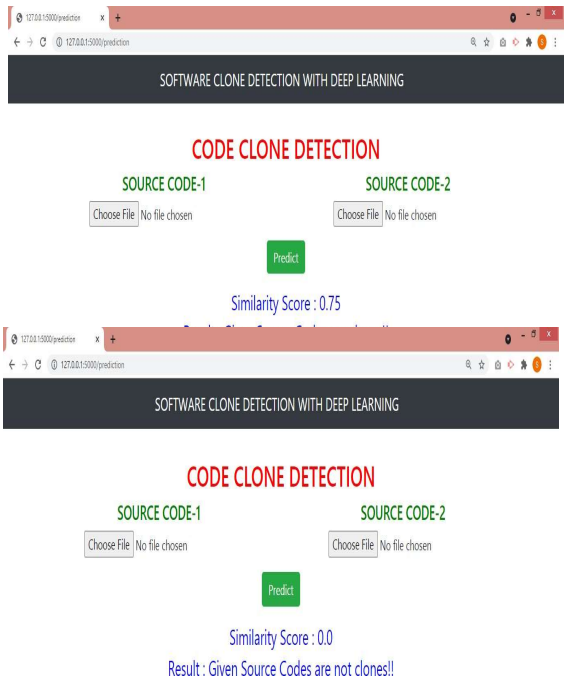


Figure 10: top figure- UI to predict dissimilar code clones and bottom figure- UI to detect similar code clones

In this paper, we give a modified version of the ASTNN model proposed for java source code similarity detection [1]. There are three major phases in our model, firstly, the embedding network to convert tokens of source code into word embeddings. Secondly, transforming the source code into a tree structure. Our approach outperformed the proposed model. Abstract Syntax Tree representation is used for code and deep neural networks for determining similar code fragments. At last, The LSTM-based Siamese Neural Networks predict code clones. Of course, Code clone detection is still a challenging problem to be solved. The proposed model can be extended to different programming languages available. The major limitation is OJClone dataset, which is not collected from real environment. There are the chances to identify false clone pairs as there are the programs under the same problem can belong to same clone pair. BigCloneBench java codes collected from large repositories of SourceForge [6], which can be helpful in identifying true clone pairs.

5. FUTURE SCOPE

In the future, we can further evaluate our model on big datasets of different or various programming languages. The work can be extended for different software engineering tasks like similar code clone detection for maintenance of software, and bug detection in software [7] so that able to reduce error propagation in software. However, recursive neural networks are more computationally expensive. Therefore, we will explore other neural network models along with intermediate code representations such as dependency graphs, etc. to extract more semantics from source codes.

REFERENCES:

[1] 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 Proceedings of the 41st International Conference on Software Engineering. IEEE Press, 2019, pp. 783–794.

[2] M. A. Fokam and R. Ajoodha, "Influence of Contrastive Learning on Source Code Plagiarism Detection through Recursive Neural Networks," 2021 3rd International Multidisciplinary Information Technology and Engineering Conference (IMITEC), 2021, pp. 1-6, doi: 10.1109/IMITEC52926.2021.9714688.

[3] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, Dilip Krishnan “Supervised Contrastive Learning” https://doi.org/10.48550/arXiv.2004.11362

[4] Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant’Anna, Lorraine Bier, Clone Detection Using Abstract Syntax Trees, IEEE/ International Conference on Software Maintenance 1998.

[5] Gang Zhao and Jeff Huang. DeepSim: Deep Learning Code Functional Similarity. In Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE ’18), November 4–9, 2018.

[6] J. Svajlenko, J. F. Islam, I. Keivanloo, C. K. Roy, and M. Mia, “Towards a big data curated benchmark of inter-project code clones,” in ICSME, 2014, p. 5

[7] Chen, J., Alalfi, M.H., Dean, T.R. et al. Detecting Android Malware Using Clone

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Detection. J. Comput. Sci. Technol. 30, 942–956 (2015).

[8] Dr. T. Venu Gopal, Divya Kumari Tankala "A Systematic Review Of Code Search And Clone Detection Techniques" Journal of Critical Reviews (JCR). 2020. Pp 7573-7588.

[9] I. Melekhov, J. Kannala and E. Rahtu, "Siamese network features for image matching," 2016 23rd International Conference on Pattern Recognition (ICPR), 2016, pp. 378-383, doi: 10.1109/ICPR.2016.7899663.