Received: 23 February 2020 | Revised: 30 July 2020 | Accepted: 4 August 2020

DOI: 10.1002/smr.2312

RESEARCH ARTICLE - METHODOLOGY

Journal of Software: Evolution and Process | WILEY

Improving deep-learning-based fault localization with resampling

Zhuo Zhang[1,4] | Yan Lei[2,3] | Xiaoguang Mao[4] | Meng Yan[2,3] | Ling Xu[2,3] | Junhao Wen[2,3]

[1]Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin, China

[2]Key Laboratory of Dependable Service Computing in Cyber Physical Society, Chongqing University, Ministry of Education, Chongqing, China

[3]School of Big Data & Software Engineering, Chongqing University, Chongqing, China

[4]College of Computer, National University of Defense Technology, Hunan, China

Correspondence
Yan Lei, School of Big Data & Software Engineering, Chongqing University, Chongqing, China.
Email: yanlei@cqu.edu.cn

Funding information
Guangxi Key Laboratory of Trusted Software, Grant/Award Number: kx202008; Fundamental Research Funds for the Central Universities, Grant/Award Number: 2019CDXYRJ0011; National Natural Science Foundation of China, Grant/Award Numbers: 61602504, 61379054, and 61672529; Scientific Research Fund of Hunan Provincial Education Department, Grant/Award Number: 15A007

Abstract

Many fault localization approaches recently utilize deep learning to learn an effective localization model showing a fresh perspective with promising results. However, localization models are generally learned from class imbalance datasets; that is, the number of failing test cases is much fewer than passing test cases. It may be highly susceptible to affect the accuracy of learned localization models. Thus, in this paper, we explore using data resampling to reduce the negative effect of the imbalanced class problem and improve the accuracy of learned models of deep-learning-based fault localization. Specifically, for deep-learning-based fault localization, its learning feature may require duplicate essential data to enhance the weak but beneficial experience incurred by the class imbalance datasets. We leverage the property of test cases (i.e., passing or failing) to identify failing test cases as the duplicate essential data and propose an iterative oversampling approach to resample failing test cases for producing a class balanced test suite. We apply the test case resampling to representative localization models using deep learning. Our empirical results on eight large-sized programs with real faults and four large-sized programs with seeded faults show that the test case resampling significantly improves fault localization effectiveness.

KEYWORDS

fault localization, debugging, neural networks, deep learning, resampling

1 | INTRODUCTION

In the process of software development and maintenance, debugging is to find and fix the bugs within the program. It usually needs much manual involvement and has been proved to be one of the most expensive and time-consuming activities for software developers. In order to reduce the cost, researchers have developed many fault localization techniques to provide the assistance in seeking the positions of the faults in the program (e.g., previous studies[1-7]). The recent progress on deep learning shows its promising ability of learning useful models in various applications (e.g., image classification, object detection, and segmentation) and providing tremendous improvement in robustness and accuracy.8Some researchers have exploited the use of this learning ability to discuss and evaluate the potential of deep learning in fault localization.[9,10,12] Their research has shown that deep learning provides a new perspective for fault localization and can significantly improve localization effectiveness.

Given a faulty program, they require a collection of specific data to construct a test suite; then execute the test suite and abstract the runtime information of the test suite as an information model for fault localization algorithms; finally based on the information model, evaluate the suspiciousness of each statement (or other program elements) of being faulty. Test cases are indispensable for the information model, and also used as the input to initiate the learning process for fault localization. There are two classes of test cases with distinct features: passing test cases and

failing test cases. Based on the two classes of test cases, deep-learning-based fault localization[10] utilizes their execution information (i.e., code coverage) as training samples, and their test results (i.e., passing or failing) as the labels, to learn an effective localization model. However, it is difficult to construct failing test cases in reality; that is, the number of passing test cases is much more than that of failing test cases. It leads to the class imbalance phenomenon and a bias to the feature learned from passing test cases. In deep learning, the class imbalance phenomenon of the training set causes negative impact on the classifier.[13,14] It will affect the accuracy of the learned localization model. Furthermore, the research[15] has found that the effect of including more passing test cases is unpredictable and adding passing test cases could both improve or degrade diagnostic accuracy. Thus, the bias to passing test cases may worsen the learning process of an effective localization model.

Since fault localization based on deep learning has the learning feature distinct from traditional fault localization, we explore using the learning feature to address the issues discussed above. Failing test cases are always beneficial (or safe) for fault localization effectiveness[15,16] whereas they are difficult to be constructed. In reality, it is almost infeasible to generate more failing test cases for improving fault localization effectiveness. However, due to the learning feature, we may not need to generate more failing test cases, *i.e.* we may use the existing failing test cases by oversampling those data with beneficial impact on the localization model. Oversampling is to randomly sample with replacement (i.e., cloning) the minority class (e.g., failing test cases) to the same size as the majority class (e.g., passing test cases).[17] In this way, we can enhance the learning of the weak but beneficial experience for fault localization. The weak but beneficial experience means the weak but beneficial pattern learned from the minority class. It means that we leverage the cloning of failing test cases to augment the experience learned from failing test cases and reduce the bias to passing test cases. Consequently, it may improve fault localization effectiveness.

Following this idea, it means that we can augment useful data using oversampling for improving deep-learning-based fault localization. We face two problems with using oversampling. One is what data is useful for oversampling. Since failing test cases are beneficial for fault localization, we solve this problem by using the learning feature; that is, we clone failing test cases into existing ones to augment the weak experience learned from failing test cases. The other one is how much data should be augmented. In fault localization, Zhang et al.[18] have found that a class-balanced test suite is useful for localization. In deep learning, many studies have shown that algorithms trained with balanced data surpass those trained with imbalanced data in performance.[19,20] Therefore, we use oversampling to iteratively argument test cases until we obtain a balanced test suite, in which passing test cases and failing test cases are with the same number.

Some may argue that although oversampling is simple yet effective and incurs a little cost, there are many data resampling techniques more effective than oversampling technique, meaning that it should be better to use other more effective techniques rather than oversampling. The resampling techniques can be roughly categorized into three commonly used types: oversampling, undersampling, and sampling with the creation of artificial data. Undersampling technique randomly removes the data from the majority class (e.g., passing test cases) in order to be the same number as the minority class (e.g., failing test cases). Since the number of minority class (e.g., failing test cases) is usually very small, the size of the test suite will substantially reduced after undersampling. It is harmful for deep-learning-based fault localization, because it requires many test cases and a small test suite will cause much information loss. Sampling with the creation of artificial data (e.g., SMOTE[21] and ADASYN[22]) will generate new artificial data similar to the minority class and fix the same label of the minority class to the new artificial data. For example, we generate a new image by switching the eyes of a dog of an image. We can still identify the label of the new image as a dog. However, in software testing and debugging, we cannot simply conclude that a test case similar to a failing test case should be the failing test case. Consequently, we cannot simply fix the labels of the similar new data as the same as the minority class. Since a new artificial datum is denoted as a vector in deep-learning-based fault localization, showing an execution path of a program. It is necessary to generate a test case executing such path and run the test case to obtain the exact label (i.e., failing or passing). It is very costly to generate a test case according to an execution path in practice, and even worse, a specific input may not theoretically exist to execute such path, meaning that we cannot generate a test case to execute such path. As a reminder, such verification process is also very costly. Furthermore, the research[23] has shown that a test case similar to a failing test case is usually not the failing test case. It means that we spend much effort to generate a set of test cases similar to failing test cases, most of them are still not the failing test cases. Consequently, the dataset still suffers from the imbalanced class problem.

Thus, we propose a test case resampling approach using failing test cases to enhance the effectiveness of deep-learning-based fault localization techniques. Our approach first identifies failing test cases as duplicate essential data; then iteratively resamples failing test cases into original test cases; finally stops the iterative resampling process until obtaining a balanced test suite, where the number of failing test case is the same as that of passing test cases. The balanced test suite augments the weak but beneficial experience related to failing test cases, thus obtaining a more well-learned localization model. To evaluate our test case resampling approach, we design and conduct a large-scale empirical study on 12 real-life programs, among which eight programs are with real faults and four ones are with seeded faults. The results show that oversampling failing test cases to construct a class-balanced test suite can improve the performance of four representative deep-learning-based fault localization techniques. Specifically, our approach can save the number of the statements to be examined up to 63.30% on CNN-FL,10 84.44% on MLP-FL,[12] 61.20% on BPNN-FL,[2,24] and 56.35% on BiLSTM-FL,[10,25]respectively.

The main contributions of this paper can be summarized as follows:

- We propose an over-sampling approach by resampling failing test cases to augment the weak but beneficial experience for deep-learning-based fault localization.

- We evaluate the test case resampling approach across various large real-life programs, showing the potential of using resampling to alleviate the class imbalance problem for improving fault localization.

The structure of the rest paper is organized as follows. Section 2 introduces deep-learning-based fault localization. Section 3 describes our test case augmentation approach. Section 4 presents the results of our empirical study including experiment subjects, experiment design, data analysis, and threats to validity. Section 5 summarizes related work, and Section 6 concludes.

2 | DEEP-LEARNING-BASED FAULT LOCALIZATION

Deep-learning-based fault localization utilizes the promising learning ability[26-30] of a neural network to learn a fault localization model to evaluate the suspiciousness of each statement being faulty. We will introduce four state-of-the-art deep-learning-based fault localization approaches used for the experiments, that is, CNN-FL, MLP-FL, BPNN-FL, and BiLSTM-FL. They share the similar structure and just use different types of neural networks. Specifically, CNN-FL10 uses Convolutional Neural Networks (CNN), MLP-FL12 utilizes Multi-Layer Perceptron (MLP), BPNN-FL uses a Back-Propagation Neural Network[2,24](BPNN), and BiLSTM-FL[10,25] utilizes a recurrent neural network named Bi-directional Long Short-Term Memory (BiLSTM).

2.1 | Information model

Deep-learning-based fault localization first defines an information model to represent the runtime information of a test suite. Figure 1 shows the definition of the information model. Specifically, given a program P with N statements, it is executed by a test suite T with M test cases, which contain at least one failing test case (see Figure 1). The element $x_{ij} = 1$ means that the statement j is executed by the test case i, and $x_{ij} = 0$ otherwise. The $M \times N$ matrix records the execution information of each statement in the test suite T. The error vector e represents the test results. The element e_i equals to 1 if the test case i failed, and 0 otherwise. The error vector shows the test results of each test case (i.e., failures or non-failures). Since they can associate the execution information of a statement (i.e., the matrix) with failures or non-failures (i.e., the error vector), deep-learning-based fault localization techniques use the matrix and the error vector as the training samples and their corresponding labels, respectively, where each training sample is an N-dimensional vector.

2.2 | Training process

Figure 2 shows the architecture of deep-learning-based fault localization: one input layer, deep leaning components, several hidden layers, and one output layer. In the input layer, deep-learning-based fault localization takes the information model defined in the Figure 1 as input.

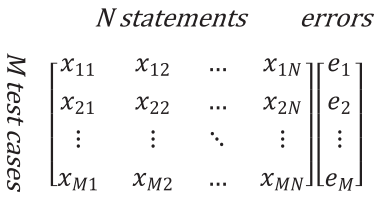


FIGURE 1 The information mode of deep-learning-based fault localization

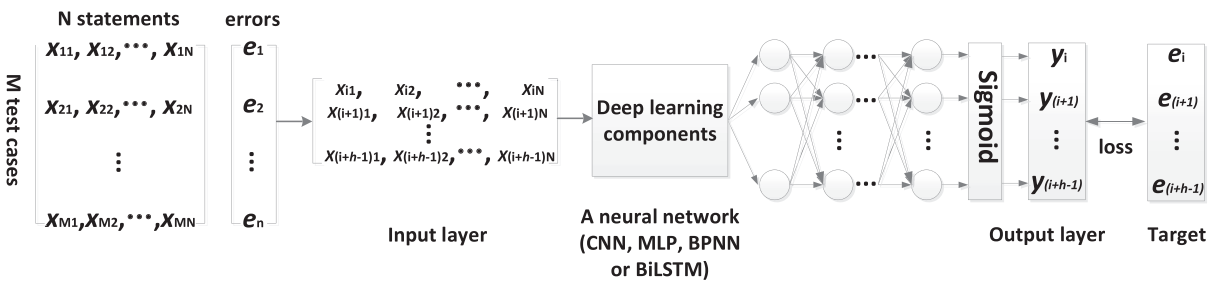


FIGURE 2 The architecture of deep-learning-based fault localization

Specifically, h rows of the matrix $M \times N$, and its corresponding error vector is used as an input, which are the coverage information of h tests and their corresponding test results starting from the ith row, where $i \in \{1,1+h,1+2h,\ldots,1+(\lfloor M/h \rfloor+1)\times h\}$. In deep learning components, there may be convolution layers, pooling layers, or fully connected layers. After that, there may be some hidden layers. CNN-FL, MLP-FL, BPNN-FL, and BiLSTM-FL use convolutional neural network, multi-layer perceptron, back-propagation neural network, and bi-directional long short-term memory for deep learning components, respectively. In the output layer, the models use *sigmoid* function10 because values sent into a *sigmoid* function will be 0 to 1. Each element in the result vector of the sigmoid function has a difference with the corresponding element of the target vector. The back propagation algorithm is used to fine-tune the parameters of the model, and the goal is to minimize the difference between training result y and error vector e. The network is trained iteratively.

The training process will learn a trained model, reflecting the complex nonlinear relationship between the statement coverage and test results. Finally, the models construct a set of virtual test cases (see Figure 3) as the testing input to measure the association of each statement with test results. Concretely, each time the models choose one virtual test case and input it to the network, the output is the estimation of the probability of causing a failure by executing the virtual test case. Furthermore, suppose that if the virtual test only covers one statement, the output is also the estimation of the probability of causing a failure by executing the statement. The estimation can show the suspiciousness of a statement of being faulty.

Thus, as shown in Figure 3, the models construct N virtual tests, equaling to the number of the statements, where each virtual test only covers one statement. Specifically, the element $x_i = 1$ means that the statement i is only covered by the virtual test t_i, and $x_i = 0$ in the other virtual tests. When the coverage vector of a virtual test is inputted to the trained neural network, the output of the network is the estimation of the virtual test's execution result of being a failure by covering only one statement. The value of the result is between 0 and 1. The larger the value is, the more likely it is that the statement only covered by the coverage vector is the buggy statement. For example, the statement i is likely to be the buggy statement. Then debugging engineers input t_i to the trained neural network (e.g., the CNN of CNN-FL, the MLP of MLP-FL, the BPNN of BPNN-FL, and the BiLSTM of BiLSTM-FL), and the output of virtual test t_i represents the probability of test execution result of being a failure by only covering the statement i. The value of the result is the suspiciousness of the statement i.

2.3 | Example

We take CNN-FL as an example to illustrate how deep-learning-based fault localization approaches work. In Figure 4, there is a faulty program P with 16 statements, among which s_3 is the faulty statement. And there are five test cases, t_1 to t_5. In Figure 4, the table is the information model of deep-learning fault localization (see Figure 1), and its detailed description can refer to Section 3.2.

The first step of CNN-FL is to construct a CNN model with one input layer with the number of nodes being 16 (i.e., the number of the statements), two convolution layers, two pooling layers with rectified linear units, two fully connected layers with the number of nodes in each layer simply set to be 8, and one output layer with the number of nodes being 1, outputting the result with *sigmoid* function.

The second step is to train the neural networks with coverage data, and input the error vector into the target vector. The batch size h is 3 in this example. Therefore, CNN-FL chooses the first three vectors in the information model (see Figure 4) as its first input matrix ((1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0), (1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0), (1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1)) and its target output vector (1, 0, 0). CNN-FL will repeat training the network by choosing three vectors in the information model each time until the loss is small enough and reach the condition of convergence. After training, CNN-FL will output a localization model, revealing the complex nonlinear relationship between statement coverage information and test results.

The third step is to construct a virtual test set (see Figure 3) that contains 16 test cases, where each test case covers only one statement out of 16 statements. We will input the virtual test set into the trained model, outputting a suspiciousness vector, where the elements of the vector denote the suspiciousness of each statement of being faulty. This step finally organizes the localization result as a ranking list of all statements in descending order of suspiciousness. As shown in Figure 4, the ranking list of CNN-FL is $\{s_{16}, s_5, s_7, s_4, s_8, s_{14}, s_{15}, s_6, s_3, s_9, s_1, s_{10}, s_{11}, s_2, s_{12}, s_{13}\}$, where CNN-FL ranks the faulty statement s_3 as the ninth place.

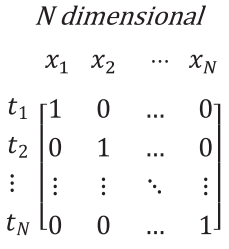


FIGURE 3 Virtual test cases

Program P

S_1:Read(a,b,c)
S_2:d1=0,d2=0,d3=0;
S_3:if(b < 0){
S_4:d1 = b;
S_5:d2 = c;
S_6:d3 = a;}
S_7:else {d1 = b+1;
S_8: d2 = c+1;
S_9:if(a < 0){
S_{10}:a = a+c;}
S_{11}: else a = a+b;
S_{12}: d3 = a+1;}
S_{13}:if(c>0){
S_{14}:output(d1);}
S_{15}:else {output(d2);
S_{16}:output(d3);}

The test case t1 will be cloned 3 times. Then there are 4 passing test cases and 4 failing test cases.

Bug information

S_3 is faulty.
Correct form:
If(b<6){

test	a,b,c	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}	S_{16}	result
t1	-1,5,3	1	1	1	0	0	0	1	1	1	1	0	1	1	1	0	0	1
t2	-2,-7,5	1	1	1	1	1	1	0	0	0	0	0	0	1	1	0	0	0
t3	5,-6,-8	1	1	1	1	1	1	0	0	0	0	0	0	1	0	1	1	0
t4	-5,8,-8	1	1	1	0	0	0	1	1	1	1	0	1	1	0	1	1	0
t5	4,7,11	1	1	1	0	0	0	1	1	1	1	0	1	1	1	0	0	0
CNN-FL	suspiciousness	0.358	0.350	0.366	0.427	0.471	0.385	0.446	0.421	0.365	0.355	0.354	0.345	0.264	0.410	0.392	0.473	
	rank	11	14	**9**	4	2	8	3	5	10	12	13	15	16	6	7	1	
CNN-FL	suspiciousness	0.702	0.512	0.643	0.480	0.614	0.596	0.541	0.664	0.582	0.604	0.593	0.599	0.581	0.645	0.512	0.617	**resampled**
	rank	1	15	**4**	16	6	9	13	2	11	7	10	8	12	3	14	5	
MLP-FL	suspiciousness	0.487	0.493	0.492	0.50	0.51	0.495	0.487	0.492	0.492	0.490	0.498	0.494	0.494	0.491	0.495	0.499	
	rank	15	9	**12**	2	1	6	16	10	11	14	4	8	7	13	5	3	
MLP-FL	suspiciousness	0.812	0.788	0.785	0.768	0.814	0.739	0.751	0.773	0.794	0.756	0.804	0.759	0.763	0.808	0.785	0.769	**resampled**
	rank	2	6	**7**	11	1	16	15	9	5	14	4	13	12	3	8	10	

FIGURE 4 An example illustrating the test case resampling

3 | TEST CASE RESAMPLING

3.1 | Methodology

In a typical software testing process, the test cases in a test suite after execution can be categorized into two classes, that is, passing test cases and failing test cases.[31] Deep-learning-based fault localization utilizes the two classes of test cases to fix the labels. However, it is difficult to construct failing test cases in reality; that is, the number of passing test cases is much higher than that of failing test cases. It leads an apparent bias to the experience learned from passing test cases. The biased experience will weaken the feature learned from failing test cases.[19,20]

It is necessary to reduce the effect of biased experience on the learning process. Data resampling[17] is usually the solution to address the biased problem. For example, suppose that there is a test suite with 98 passing test cases and two failing test cases. We apply undersampling[17] and delete 96 passing test cases. The test suite has the same number of passing test cases and failing ones. A small sample with four test cases means much useful information may be removed and are inadequate to conduct deep learning to learn an effective localization model. The second solution is sampling with the creation of artificial data,[17] which is similar to the two failing test cases. In deep-learning-based fault localization, the minority class corresponds to failing test cases and the label of minority class is the failing result. However, a test case similar to the minority class (that is, failing test cases) is not necessarily a failing test case.[23] The third one is oversampling, [17,21,22] which clones the data from the minority class to generate a balanced dataset. Since many resampling approaches are more effective than oversampling, oversampling is not the first choice. However, the aforementioned discussion shows that oversampling may be the one possible candidate solution to alleviate the biased problem in deep-learning-based fault localization.

Thus, we try to propose a test case resampling approach based on oversampling. In deep learning, oversampling (i.e., cloning) specific data will improve the loss function of their corresponding class and thus augment the experience learned from those data.[32] Therefore, the basic idea is that if we know that a weak but beneficial experience learned from some data, we can leverage the learning feature by cloning those data to argument this weak but beneficial experience.

To initiate the realization of the basic idea, we should first identify those data related to the weak but beneficial experience. The research[15,16] has found failing test cases are always beneficial (or at least safe) for fault localization effectiveness.16 However, failing test cases are much less than passing test cases. It means that the experience learned from failing test cases is weak. The learning feature enables the use of cloning failing test cases for enhancing its corresponding experience.

Next, we should identify how many failing test cases should be cloned as the augmentation. In both fault localization and deep learning, many studies[18-20] have found that a class-balanced test suite is useful for localization, and also the algorithms trained with balanced data surpass those trained with imbalanced data in performance. It means that we should clone failing test cases into original test suite until the passing test cases and the failing test cases have the same number. We define a ratio $\theta = Pnum/Fnum$, where $Pnum$ and $Fnum$ denote the number of passing test cases and the number of failing test cases, respectively. Our test case resampling approach will clone failing test cases until the ratio $\theta = 1$.

Algorithm 1 shows how our resampling approach uses iterative over-sampling to generate a balanced test suite. Line 1 to line 2 get the number of passing test cases (i.e., $Pnum_{orig}$) and the number of failing test cases (i.e., $Fnum_{orig}$) in T_{orig}, respectively. Line 3 to line 4 assign the original test suite and its failing test cases to T_{new} and T_{origF}. We iteratively clone the entire set of original failing test cases to let the number of failing test cases be close to the number of the original passing test cases. Specifically, line 5 to line 8 will iteratively oversample failing test cases until the number of failing test cases is close to that of original passing tests. In other words, we will clone the entire original failing test cases T_{origF} with $(k-1)$ times and add all cloned failing test cases to the test suite T_{new}, where $k = \lfloor P_{orig}/F_{orig} \rfloor$ is the integer part of P_{orig}/F_{orig}, $addTests(T_{origF}, T_{new})$ is a function to add all test cases of T_{origF} into the test suite T_{new}, that is, $T_{new} = T_{origF} \cup T_{new}$. Now, the number of failing test cases in T_{new} is $k * Fnum_{orig}$; we still require to add m failing test cases to obtain a balanced test suite, where $m = Pnum_{orig} - k * Fnum_{orig}$. Line 9 to line 12 randomly clone m failing test cases from the original failing ones T_{origF} and add the m failing test cases to T_{new}, $getRandomTest(T_{origF})$ is a function randomly returning a test case from T_{origF}. Finally, in the new test suite T_{New}, the number of failing test cases equals to that of passing test cases, meaning the ratio $\theta = 1$.

Algorithm 1 The iterative resampling algorithm

Input: The original test suite, T_{orig};
Output: The new test suite, T_{new}.

```
1:  Pnum_oirg = getNumberofPassingTests(T_orig);
2:  Fnum_oirg = getNumberofFailingTests(T_orig);
3:  T_new = T_orig;
4:  T_origF = getFailingTests(T_orig);
5:  k = ⌊Pnum_orig / Fnum_orig⌋;
6:  for i = 1; i < k; i++ do
7:      addTests(T_origF, T_new);
8:  end for
9:  m = Pnum_orig − k ∗ Fnum_orig;
10: for i = 1; i <= m; i + + do
11:     addTests(getRandomTest(T_origF), T_new);
12: end for
13: return T_New;
```

Deep learning-based fault localization uses the execution information (i.e., coverage information) and test results (i.e., failure or non-failure) of the new test suite T_{New} to construct and train a neural network, thus learning a localization model. Based on the localization model, it evaluates the suspiciousness of each statement of being faulty, and outputs a ranking list of all statements in terms of suspiciousness [1].

3.2 | An illustrative example

This section uses an example to illustrate how the test case resampling approach is applied to deep-learning-based fault localization (e.g., CNN-FL and MLP-FL). Figure 4 illustrates a faulty program P with 16 statements (i.e., the first and second rows), among which s_3 is the faulty statement. The third row shows the names of the test cases, the input variables of the program, the statements, and the test results. The next five rows are the execution information of the five test cases, where the first column on the left are five test cases (i.e., t_1 to t_5), the second column represents the values of the input variables in each test case, the next 16 columns show the execution information of 16 statements in each test case, and the rightmost column shows the test results of each test case. Specifically, the cells below each statement indicate whether the statement is covered by the execution of a test case or not (1 for executed and 0 for not executed), and the rightmost cells represent whether the execution of a test case is failed or not (0 for pass and 1 for fail). We can observe that there are five test cases, in which the test case t_1 is a failing test case. The next 8 rows are localization results (the suspiciousness and the ranks of each statement) by using CNN-FL and MLP-FL. The rightmost cells with *resampled* show the localization results corresponds to the ones of CNN-FL and MLP-FL after applying our resampling approach.

Since we have only one failing test case, our test case resampling approach iteratively clones the test case t_1 three times. Then, it adds the three test cases into original test suite to form a new test suite with four failing test cases and four passing test cases. We input the new test suite into CNN-FL and MLP-FL, and they will conduct the process as described above. They also output a suspiciousness vector with the suspiciousness of each statement of being faulty.

[1]Code available at https://github.com/zhuozhangNUDT/test-case-resampling.

TABLE 1 Summary of subject programs

Program	Description	Versions	KLOC	Test	Type
chart	JFreeChart	26	96	2205	Real
math	Apache Commons Math	106	85	3602	Real
mockito	Framework for unit tests	38	6	1075	Real
time	Joda-Time	27	53	4130	Real
python	General-purpose language	8	407	355	Real
gzip	Data compression	5	491	12	Real
libtiff	Image processing	12	77	78	Real
space	ADL interpreter	35	6.1	13585	Real
nanoxml:v1	XML parser	7	5.4	206	Seeded
nanoxml:v2	XML parser	7	5.7	206	Seeded
nanoxml:v3	XML parser	10	8.4	206	Seeded
nanoxml:v5	XML parser	7	8.8	206	Seeded

As shown in Figure 4, there are four ranking lists of all statements in descending order of suspiciousness. With the original test suite, CNN-FL is $\{s_{16}, s_5, s_7, s_4, s_8, s_{14}, s_{15}, s_6, s_3, s_9, s_1, s_{10}, s_{11}, s_2, s_{12}, s_{13}\}$, and MLP-FL is $\{s_5, s_4, s_{16}, s_{11}, s_{15}, s_6, s_{13}, s_{12}, s_2, s_8, s_9, s_3, s_{14}, s_{10}, s_1, s_7\}$. With the test case resampling, CNN-FL is $\{s_1, s_8, s_{14}, s_3, s_{16}, s_5, s_{10}, s_{12}, s_6, s_{11}, s_9, s_{13}, s_7, s_{15}, s_2, s_4\}$, and MLP-FL is$\{s_5, s_1, s_{14}, s_{11}, s_9, s_2, s_3, s_{15}, s_8, s_{16}, s_4, s_{13}, s_{12}, s_{10}, s_7, s_6\}$. We can observe that CNN-FL and MLP-FL rank the faulty statement s_3 as the ninth and the 12th place, respectively. After applying our test case resampling, CNN-FL and MLP-FL rank the faulty statement s_3 as the fourth and seventh place, respectively. It means that our test case resampling approach has promoted both five places of the faulty statement in CNN-FL (from ninth to fourth) and MLP-FL (from 12th to seventh), showing better localization results.

4 | AN EXPERIMENTAL STUDY

4.1 | Experimental setup

To evaluate the effectiveness of our approach, we use the four state-of-the-art deep learning-based fault localization approaches: CNN-FL,[10] MLP-FL,12 BPNN-FL,[2,24] and BiLSTM-FL,[10,25] for the experiments. We will compare the effectiveness of the two approaches in two scenarios: using the test case resampling approach and without using it. Furthermore, we choose those widely used large-sized programs in the field of software debugging (e.g., previous studies[1,2,4-7]) for the experiments, varying from 5 KLOC to 491 KLOC.

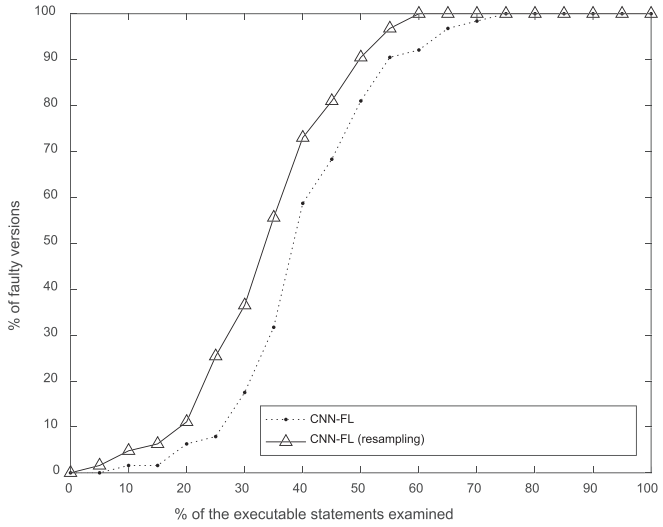
Table 1 summarizes the characteristics of the subject programs used. For each program, it depicts a brief functional description (column "Description"), the number of faulty versions (column "Versions"), the number of thousand lines of statements (column "KLOC"), the number of test cases (column "Test"), and the type of the faults (column "Type"). The first four programs (i.e., *chart*, *math*, *mockito*, and *time*) are from Defects4J[2], which is a database and extensible framework providing real faults to enable reproducible studies in software testing research.[33] *Python*, *gzip*, and *libtiff* are collected from ManyBugs[3], *space* is acquired from the SIR[4].The next four programs are seeded faults of the four sperate releases of *nanoxml* acquired from the SIR.

The physical environment on which we conducted the experiments was on a computer containing a CPU of Intel I5-2640 with 128G physical memory and two 12G GPUs of NVIDIA TITAN X Pascal. The operating systems were Ubuntu 16.04.3. We conducted the experiments on the MATLAB R2016b.

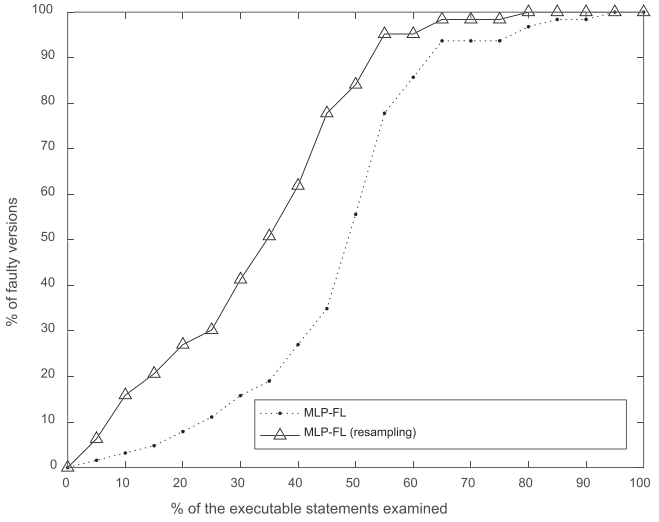
4.2 | Evaluation metrics

To evaluate fault localization effectiveness, we adopt two widely used metrics: fault localization accuracy (referred as *EXAM*)34 and relative improvement (referred as *RImp*).35-37 *EXAM* is defined as the percentage of statements to be examined before finding the actual faulty statement. A lower value of *EXAM* indicates better performance. *RImp* is to compare the total number of statements that need to be examined to find

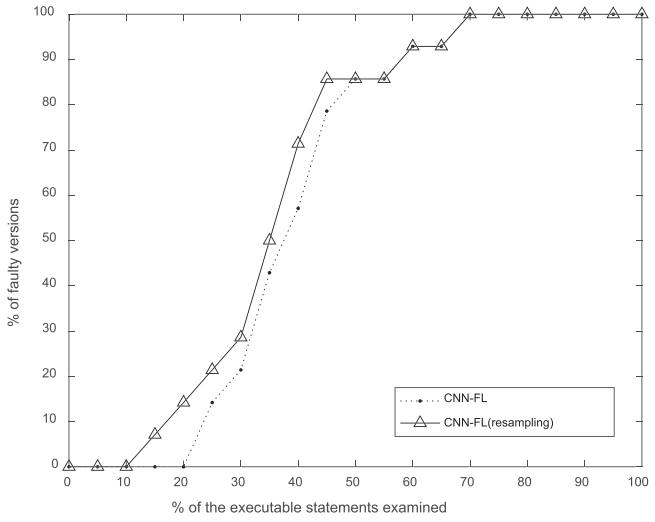
[2]Defects4J, http://defects4j.org
[3]ManyBugs, http://repairbenchmarks.cs.umass.edu/ManyBugs/
[4]SIR, http://sir.unl.edu/portal/index.php



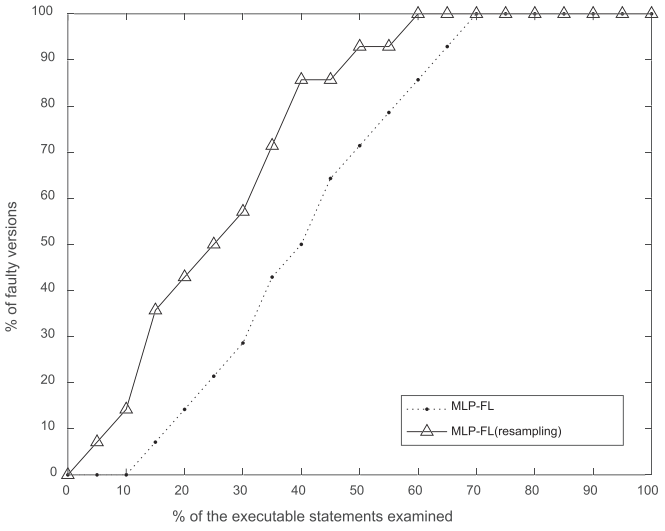
(A) CNN-FL versus CNN-FL using resampling on real faults.



(B) MLP-FL versus MLP-FL using resampling on real faults.



(C) CNN-FL versus CNN-FL using resampling on seeded faults.



(D) MLP-FL versus MLP-FL using resampling on seeded faults.

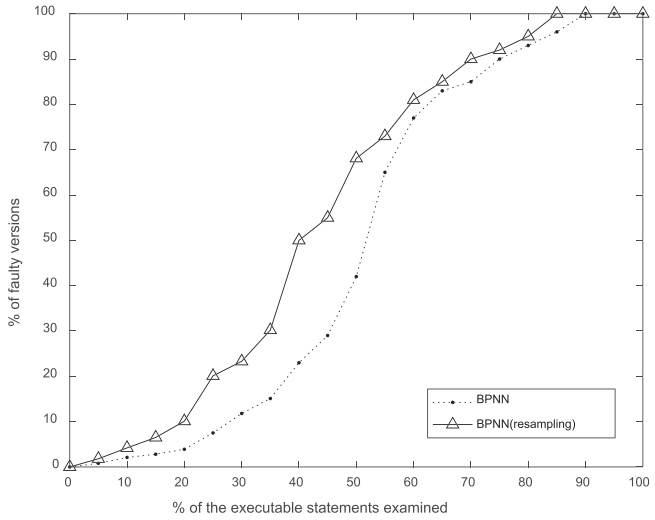
FIGURE 5 *EXAM* comparison between using test case resampling approach and without using it

all faults using a deep learning-based fault localization approach with the test case resampling versus the number that need to be examined by using the deep learning-based fault localization approach without the test case resampling. A lower value of *RImp* shows better improvement of our approach over the one without test case resampling.

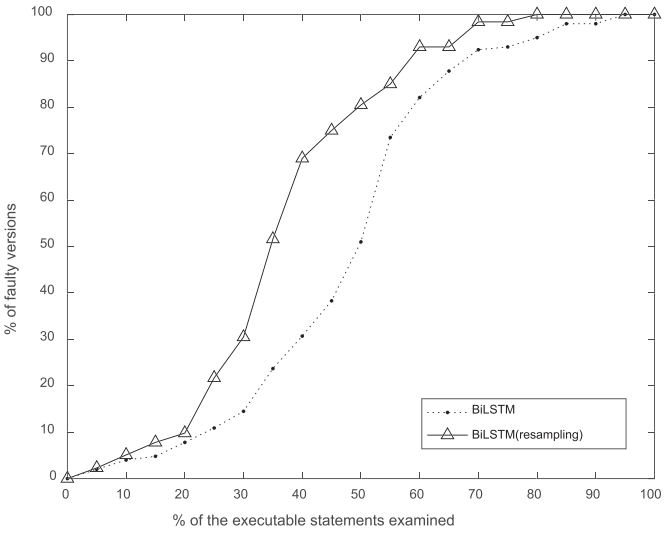
4.3 | Data analysis

4.3.1 | EXAM distribution

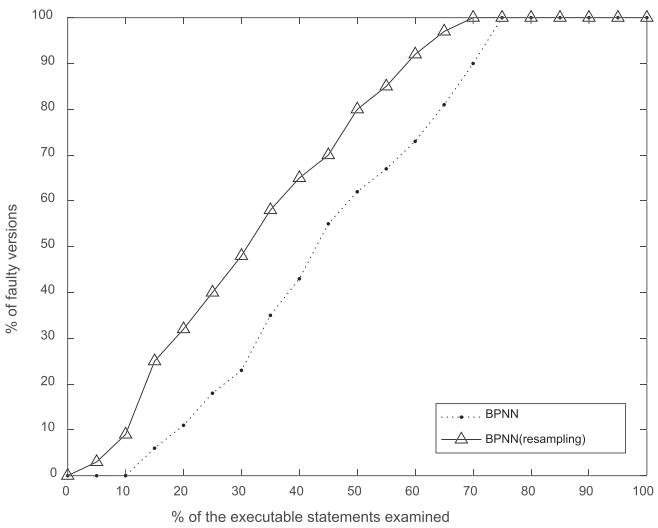
Figures 5 and 6 illustrate the *EXAM* comparison between our test case resampling approach and other localization approaches in all faulty versions. Specifically, Figure 5A,C shows the *EXAM* comparison between CNN-FL and CNN-FL using our test case resampling approach in the context of real faults and seeded faults, respectively; Figure 5B,D shows the comparison between MLP-FL and MLP-FL using resampling in the context of real faults and seeded faults, respectively. Figure 6A,C shows the *EXAM* comparison between BPNN-FL and BPNN-FL using our test case resampling approach in the context of real faults and seeded faults, respectively; Figure 6B,D shows the comparison between BiLSTM-FL and BiLSTM-FL using resampling in the context of real faults and seeded faults, respectively. In Figures 5and 6, the horizontal axis represents the percentage of statements examined in all versions of subjects (i.e., *EXAM*), while the vertical axis is the percentage of fault located in all faulty



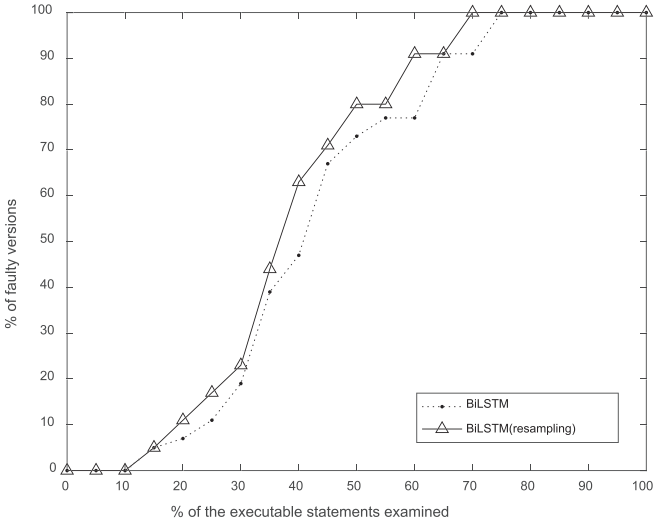
(A) BPNN-FL versus BPNN-FL using resampling on real faults.



(B) BiLSTM-FL versus BiLSTM-FL using resampling on real faults.



(C) BPNN-FL versus BPNN-FL using resampling on seeded faults.



(D) BiLSTM-FL versus BiLSTM-FL using resampling on seeded faults.

FIGURE 6 *EXAM* comparison between using test case resampling approach and without using it

versions. A point in Figures 5 and 6 denotes when a percentage of statements is examined in each faulty version, the percentage of faulty versions has located their faults. From the eight subfigures, we can observe that the curves of CNN-FL (resampling), MLP-FL (resampling), BPNN-FL (resampling), and BiLSTM-FL (resampling) are always above the ones of CNN-FL, MLP-FL, BPNN-FL, and BiLSTM-FL in both real faults and seeded faults. The results show that our approach improves the localization effectiveness of CNN-FL, MLP-FL, BPNN-FL,24 and BiLSTM-FL. Before applying our resampling approach, the localization effectiveness on real faults are much higher than that of artificial faults. Thus, the resampling approach performs differently in the two scenarios; that is, the improvement on seed faults is much higher than real faults.

Furthermore, Table 2 illustrates the three types of *EXAM* scores of our test case resampling approach in comparison with CNN-FL, MLP-FL, BPNN-FL, and BiLSTM-FL on real faults while Table 3 illustrates the three types of *EXAM* scores on seeded faults. For each approach, it provides three types of *EXAM* scores: *best*, *average*, and *variance*. The *best* score means the minimum *EXAM* score of all the versions of a program, showing the best effectiveness of an approach can reach. The *average* score means the average *Exam* score of all the versions of a program, showing the average effectiveness of an approach can achieve. The *variance* score means the variance of *EXAM* scores of all the versions of a program, showing whether the effectiveness of an approach is stable or not.

As shown in Table 2, we observe that among the eight real-faults programs, CNN-FL using test case resampling obtains five firsts in *best EXAM* scores, seven firsts in *average Exam* scores, and five firsts in *variance EXAM* scores compared with CNN-FL without test case resampling. MLP-FL using test case resampling obtains four firsts in *best EXAM* scores, six firsts in *average EXAM* scores, and seven firsts in *variance EXAM* scores compared with MLP-FL without test case resampling. BPNN-FL using test case resampling obtains five firsts in *best EXAM* scores, six firsts

TABLE 2 The best, average, and variance *EXAM* scores on real faults

		python	gzip	libtiff	space	chart	math	mockito	time
	best	0.033764	**0.006329**	**0.110802**	0.100686	**0.041451**	0.031021	0.167024	0.054494
CNN-FL	average	0.276977	**0.104460**	0.302835	0.339422	0.220833	0.214605	0.232968	0.072166
	variance	0.208231	0.088525	**0.107424**	0.139387	**0.155409**	0.259628	**0.053328**	0.249930
	best	**0.030410**	0.064984	0.191492	**0.087243**	0.088083	**0.020305**	**0.136181**	**0.050280**
CNN-FL (resampling)	average	**0.226208**	0.277906	**0.278616**	**0.324122**	**0.178832**	**0.128861**	**0.195547**	**0.059650**
	variance	**0.194044**	**0.066240**	0.113832	**0.121806**	0.197779	**0.153522**	0.067376	**0.132512**
	best	0.133016	**0.006329**	0.195170	**0.111660**	**0.145078**	0.119571	**0.008208**	0.344848
MLP-FL	average	0.322419	**0.244269**	0.412189	0.402564	0.356506	0.349378	0.185645	**0.341644**
	variance	0.159671	**0.174213**	0.214892	0.167041	0.186936	0.324996	0.147868	0.451910
	best	**0.107999**	0.106013	**0.177695**	0.183539	0.165803	**0.053801**	0.014632	**0.297301**
MLP-FL (resampling)	average	**0.243261**	0.249593	**0.331293**	**0.394992**	**0.258620**	**0.072022**	**0.093986**	0.344287
	variance	**0.100335**	0.178988	**0.140644**	**0.138368**	**0.080551**	**0.025768**	**0.082215**	**0.066448**
	best	0.271173	0.229535	0.231676	0.279287	**0.210502**	0.237418	**0.136181**	**0.211274**
BPNN-FL	average	0.407999	**0.321778**	0.543658	0.450206	0.424556	0.579186	0.290354	0.344847
	variance	**0.159670**	**0.102828**	0.358621	0.137370	0.218310	0.179720	**0.179765**	**0.041784**
	best	**0.165299**	**0.128664**	**0.142857**	**0.209054**	0.237911	**0.217418**	0.157218	0.220280
BPNN-FL (resampling)	average	**0.328829**	0.543658	**0.449286**	**0.404938**	**0.310345**	**0.316742**	**0.249408**	**0.338449**
	variance	0.208231	0.188520	**0.107411**	**0.073834**	**0.128450**	**0.151738**	0.181736	0.116387
	best	0.518816	0.118297	0.046811	0.073876	0.236184	**0.168077**	0.110992	0.159963
BiLSTM-FL	average	0.357230	0.267565	0.356844	0.412287	0.317215	0.242409	0.261782	**0.307159**
	variance	0.145292	**0.137333**	0.258995	0.199216	**0.087361**	**0.105122**	0.136977	0.208166
	best	**0.143540**	**0.106013**	**0.046811**	**0.073880**	**0.199807**	0.174187	**0.114056**	**0.187301**
BiLSTM-FL (resampling)	average	**0.260260**	**0.258730**	**0.310721**	**0.377503**	**0.266184**	**0.199571**	**0.191514**	0.322165
	variance	**0.133467**	0.192020	**0.107428**	**0.117970**	0.154077	0.116386	**0.060189**	**0.066925**

in *average Exam* scores, and four firsts in *variance EXAM* scores compared with BPNN-FL without test case resampling. BiLSTM-FL using test case resampling obtains six firsts in *best EXAM* scores, seven firsts in *average EXAM* scores, and five firsts in *variance EXAM* scores compared with BiLSTM-FL without test case resampling.

And in Table 3, among the four seeded-faults programs, CNN-FL using test case resampling obtains three firsts in *best EXAM* scores, three firsts in *average Exam* scores, and one first in *variance EXAM* scores compared with CNN-FL without test case resampling. MLP-FL using test case resampling obtains three firsts in *best EXAM* scores, three firsts in *average EXAM* scores, and three firsts in *variance EXAM* scores compared with MLP-FL without test case resampling. BPNN-FL using test case resampling obtains three firsts in *best EXAM* scores, four firsts in *average Exam* scores, and three first in *variance EXAM* scores compared with BPNN-FL without test case resampling. BiLSTM-FL using test case resampling obtains three firsts in *best EXAM* scores, three firsts in *average EXAM* scores, and three firsts in *variance EXAM* scores compared with BiLSTM-FL without test case resampling.

The results show that after applying our test case resampling approach, CNN-FL, MLP-FL, BPNN-FL, and BiLSTM-FL become more effective and more stable on both real faults and seeded faults.

4.3.2 | RImp distribution

For a detailed improvement, we adopt *RImp* to evaluate our approach. Figure 7 shows *RImp* score of using test case resampling over the four fault localization approaches (i.e., CNN-FL, MLP-FL, BPNN-FL, and BiLSTM-FL) with original test suite.

As shown in Figure 7A, the *RImp* score is less than 100% in all programs except for *time* and *nanoxml:v1*, meaning that CNN-FL using resampling improves fault localization effectiveness in almost all programs. Except for *time* and *nanoxml:v1*, the statements that need to be examined decrease ranging from 36.97% in *gzip* to 99.31% in *python*. This means that CNN-FL using test case resampling needs to examine from 36.97% to 99.31% of statements that CNN-FL with original test suite needs to examine of. It also means that the test case resampling approach, in comparison with CNN-FL, obtains a maximum saving of 63.03% (100% − 36.97% = 63.03%) in *gzip* and a minimum saving of 0.69% (100% − 99.31% = 0.69%) in *python*. The average saving is 16.61%, which means that the test case resampling approach can save an average of 16.61% of the number of statements examined by using CNN-FL to locate all faults.

TABLE 3 The best, average, and variance *EXAM* scores on seeded faults

		nanoxml:v1	nanoxml:v2	nanoxm_v3	nanoxml:v5
	best	**0.298781**	0.169492	0.282927	0.258621
CNN-FL	average	**0.301829**	0.193974	0.391880	0.367228
	variance	**0.004311**	**0.108019**	**0.097831**	0.097317
	best	0.304878	**0.084746**	**0.191667**	**0.224138**
CNN-FL (resampling)	average	0.329268	**0.167608**	**0.356243**	**0.318096**
	variance	0.034493	0.127572	0.157328	**0.084009**
	best	0.115854	0.067797	0.353846	**0.241379**
MLP-FL	average	0.185976	0.171375	0.476591	**0.344978**
	variance	0.099167	0.083306	**0.116304**	0.130271
	best	**0.079268**	**0.028249**	**0.014925**	0.260465
MLP-FL (resampling)	average	**0.094512**	**0.062147**	**0.240357**	0.365608
	variance	**0.021558**	**0.033131**	0.137631	**0.128446**
	best	0.115854	0.0711860	0.222222	**0.255172**
BPNN-FL	average	0.109756	0.141243	0.365546	0.491228
	variance	0.041780	0.039548	**0.18173**	0.254450
	best	**0.079268**	**0.079096**	**0.191667**	0.264138
BPNN-FL (resampling)	average	**0.085366**	**0.084746**	**0.320773**	**0.460465**
	variance	**0.031680**	**0.031116**	0.210672	**0.226850**
	best	0.109756	**0.028249**	0.120833	0.060185
BiLSTM-FL	average	**0.073171**	0.090395	0.304167	0.302326
	variance	0.031223	0.066928	**0.116380**	0.218311
	best	**0.109756**	0.084746	**0.092683**	**0.060185**
BiLSTM-FL (resampling)	average	0.079268	**0.073446**	**0.202308**	**0.224138**
	variance	**0.030258**	**0.041781**	0.179720	**0.203110**

As shown in Figure 7B, the *RImp* score is less than 100% in all programs, meaning that MLP-FL using resampling improves fault localization effectiveness in all programs. In comparison with MLP-FL, the test case resampling approach gets a maximum saving of 84.44% (100% − 15.56% = 84.44%) in *math*, a minimum saving of 3.80% (100% − 97.20% = 3.80%) in *python*, and an average saving of 38.35%.

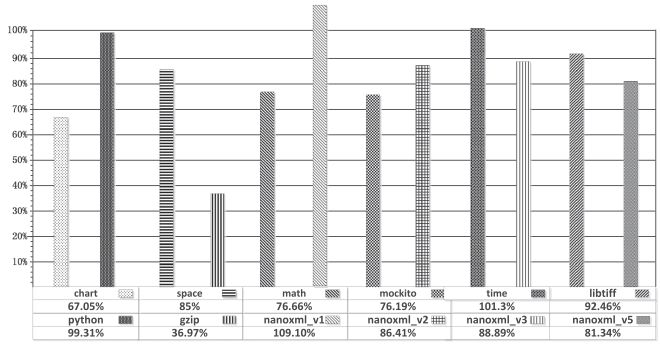
As shown in Figure 7C, the *RImp* score is less than 100% in all programs, meaning that BPNN-FL using resampling improves fault localization effectiveness in all programs. In comparison with BPNN-FL, the test case resampling approach gets a maximum saving of 84.44% (100% − 38.80% = 61.20%) in *nanoxml:v2*, a minimum saving of 0.82% (100% − 91.27% = 8.73%) in *space*, and an average saving of 26.38%.

As shown in Figure 7D, the *RImp* score is less than 100% in all programs, meaning that BiLSTM-FL using resampling improves fault localization effectiveness in all programs. In comparison with BiLSTM-FL, the test case resampling approach gets a maximum saving of 56.35% (100% − 43.65% = 56.35%) in *math*, a minimum saving of 0.82% (100% − 99.18% = 0.82%) in *nanoxml:v1*, and an average saving of 21.02%.

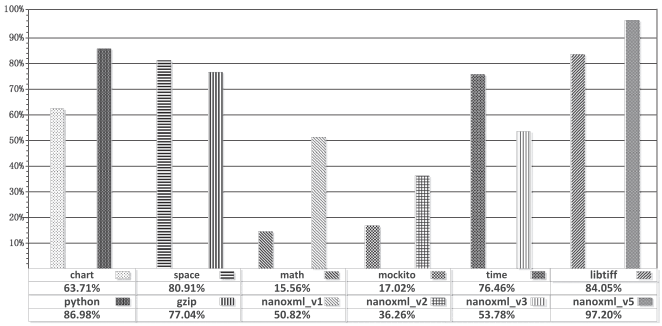
Based on the *RImp* scores, we can observe that there is a significant saving after using our test case resampling approach. Hence, oversampling failing test cases is an effective way in improving deep learning-based fault localization.

4.3.3 | Statistical comparison

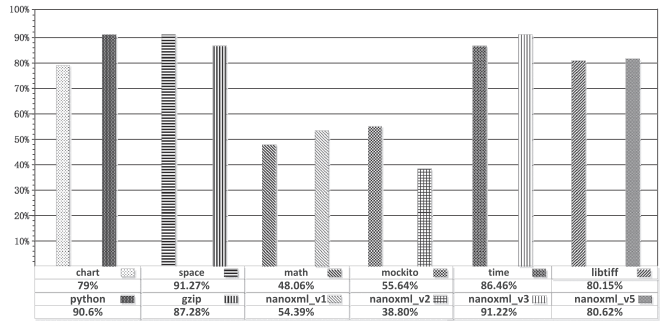
Although *RImp* can show more detailed improvement, the analysis using *RImp* evaluates effectiveness from the overview of the results and may miss other detailed views of the results. For example, the *RImp* scores of *libtiff* and *python* in Figure 7A are very close to 100%, which means that CNN-FL using test case resampling performs closely to CNN-FL with the original test suites in *libtiff* and *python*. However, a case may happen. Suppose that CNN-FL using test case resampling has higher but not quite higher effectiveness than CNN-FL with the original test suite in each faulty version of a program, the *RImp* score will show that they perform closely. Nevertheless, in this case, it is difficult to conclude that CNN-FL using test case resampling performs closely to CNN-FL without resampling because CNN-FL using test case augmentation performs better in each faulty version of the program. For another example, suppose that CNN-FL using test case resampling just has very higher effectiveness than CNN-FL with the original test suite in several faulty versions of a program, and however, CNN-FL with the original test suite has moderately



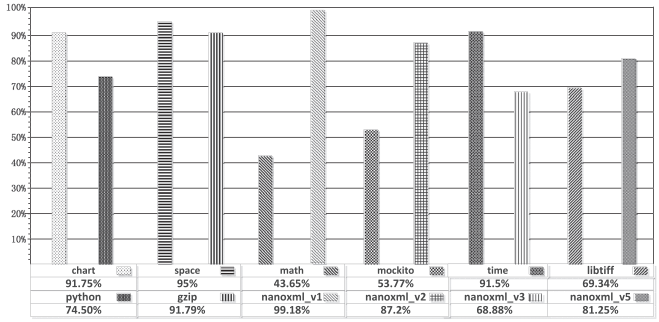
(A) The *RImp* of CNN-FL using resampling over CNN-FL.



(B) The *RImp* of MLP-FL using resampling over MLP-FL.



(C) The *RImp* of BPNN-FL using resampling over BPNN-FL.



(D) The *RImp* of BiLSTM-FL using resampling over BiLSTM-FL.

FIGURE 7 The *RImp* of test case resampling over deep-learning fault localization

TABLE 4 Statistical results on CNN-FL (resampling) versus CNN-FL

	Wilcoxon-Signed-Rank test				
Program	**Two-tailed**	**One-tailed (right)**	**One-tailed (left)**	**Conclusion**	**A-Test**
chart	6.98E−03	9.89E−01	5.45E−03	BETTER	0.88
gzip	4.31E−03	9.85E−01	2.95E−02	BETTER	0.92
libtiff	5.45E−03	9.87E−01	6.87E−03	BETTER	0.80
math	1.09E−02	9.69E−01	9.07E−03	BETTER	0.67
mokito	3.35E−02	9.43E−01	2.09E−02	BETTER	0.84
python	1.00E+00	5.72E−01	5.72E−01	SIMILAR	0.44
space	3.57E−02	8.26E−01	1.83E−02	BETTER	0.55
time	6.55E−01	5.00E−01	8.14E−01	SIMILAR	0.50
nanoxml:v1	3.17E−02	4.89E−02	9.77E−01	WORSE	0.13
nanoxml:v2	4.93E−02	7.98E−01	3.95E−02	BETTER	0.67
nanoxml:v3	4.63E−02	7.99E−01	2.65E−02	BETTER	0.61
nanoxml:v5	1.09E−02	9.69E−01	9.07E−03	BETTER	0.78
Total	4.02E−05	9.99E−01	2.04E−05	BETTER	0.72

higher effectiveness in most faulty versions of the programs. The sheer high effectiveness of CNN-FL using test case resampling in the several faulty versions may make its *RImp* score over CNN-FL with original test suite lower than 100%, showing that CNN using test case resampling performs better than CNN-FL with the original test suite. However, in such case, we cannot conclude that CNN using test case resampling performs better than CNN-FL with the original test suite.

Thus, we need a rigorous method to obtain a detailed result and adopt Wilcoxon-Signed-Rank Test38 to achieve this goal. Wilcoxon-Signed-Rank Test is a non-parametric statistical hypothesis test for testing the differences between pairs of measurements F(x) and G(y), which do not follow a normal distribution. At a given significant level σ, we can use both two-tailed and one-tailed p-value to obtain a conclusion. For the two-

TABLE 5 Statistical results on MLP-FL (resampling) versus MLP-FL

	Wilcoxon-Signed-Rank test				
Program	Two-tailed	One-tailed (right)	One-tailed (left)	Conclusion	A-Test
chart	6.79E−03	9.78E−01	5.02E−03	BETTER	0.81
gzip	3.45E−02	8.60E−01	2.09E−02	BETTER	0.72
libtiff	7.96E−03	9.70E−01	5.28E−03	BETTER	0.88
Math	1.09E−02	9.69E−01	9.07E−03	BETTER	0.89
mokito	4.51E−02	9.09E−01	2.45E−02	BETTER	0.88
python	4.65E−02	8.19E−01	2.92E−02	BETTER	0.63
space	7.96E−03	9.99E−01	4.24E−04	BETTER	0.74
time	1.80E−02	9.63E−01	1.86E−02	BETTER	0.93
nanoxml:v1	1.80E−02	9.63E−01	1.86E−02	BETTER	0.96
nanoxml:v2	1.09E−02	9.69E−01	9.07E−03	BETTER	0.78
nanoxml:v3	2.77E−02	9.89E−01	1.80E−02	BETTER	0.94
nanoxml:v5	5.93E−02	3.95E−01	7.89E−01	SIMILAR	0.33
Total	1.79E−02	9.91E−01	9.03E−03	BETTER	0.75

TABLE 6 Statistical results on BPNN-FL (resampling) versus BPNN-FL

	Wilcoxon-Signed-Rank test				
Program	Two-tailed	One-tailed (right)	One-tailed (left)	Conclusion	A-Test
chart	6.79E−03	9.78E−01	5.02E−03	BETTER	0.81
gzip	2.25E−02	9.11E−01	1.40E−02	BETTER	0.68
libtiff	4.31E−02	9.85E−01	2.95E−02	BETTER	0.88
Math	1.09E−02	9.69E−01	9.07E−03	BETTER	0.95
mokito	4.34E−02	9.76E−01	2.74E−02	BETTER	0.88
python	4.79E−02	9.23E−01	4.02E−02	BETTER	0.75
space	7.96E−04	9.99E−01	4.24E−04	BETTER	0.72
time	1.80E−02	9.63E−01	1.86E−02	BETTER	0.95
nanoxml:v1	1.80E−02	9.63E−01	1.86E−02	BETTER	0.95
nanoxml:v2	1.08E−02	9.69E−01	9.07E−03	BETTER	0.67
nanoxml:v3	2.77E−02	9.89E−01	1.80E−02	BETTER	0.95
nanoxml:v5	2.85E−02	9.09E−01	2.11E−02	BETTER	0.67
Total	1.92E−09	9.99E−01	9.83E−10	BETTER	0.78

tailed p-value, if $p \geq \sigma$, the null hypothesis H_0 that F(x) and G(y) are not significantly different is accepted; otherwise, the alternative hypothesis H_1 that F(x) and G(y) are significantly different is accepted. For one-tailed p-value, there are two cases: the right-tailed case and the left-tailed case. In the right-tailed case, if $p \geq \sigma$, H_0 that F(x) does not significantly tend to be greater than the G(y) is accepted; otherwise, H_1 that F(x) significantly tends to be greater than the G(y) is accepted. And in the left-tailed case, if $p \geq \sigma$, H_0 that F(x) does not significantly tend to be less than the G(y) is accepted; otherwise, H_1 that F(x) significantly tends to be less than the G(y) is accepted.

The experiments performed four paired Wilcoxon-Signed-Rank tests (i.e., CNN-FL [resampling] versus CNN-FL, MLP-FL [resampling] versus MLP-FL, BPNN-FL [resampling] versus BPNN-FL, and BiLSTM-FL [resampling] versus BiLSTM-FL) by using *EXAM* as the pairs of measurements F(x) and G(y). Each test uses both the two-tailed and one-tailed checking at the σ level of 0.05. Specifically, given a localization technique FL1, we use the list of *EXAM* of FL1 using test case resampling in all faulty versions of all programs as the list of measurements of F(x), while the list of measurements of G(y) is the list of *Exam* of FL1 without resampling in all faulty versions of all programs. Hence, in the two-tailed test, FL1 using resampling has SIMILAR effectiveness as FL1 when H_0 is accepted at the significant level of 0.05. And in the one-tailed test (right), FL1 using resampling has WORSE effectiveness than FL1 when H_1 is accepted at the significant level of 0.05. Finally, in the one-tailed test (left), FL1 using resampling has BETTER effectiveness than FL1 when H_1 is accepted at the significant level of 0.05.

Tables 4-6, and 7 show the statistical results on CNN-FL MLP-FL, BPNN-FL, and BiLSTM-FL using test case resampling versus CNN-FL, MLP-FL, BPNN-FL, and BiLSTM-FL with the original test suite. Take *chart* in Table 4 as an example. The p values of two-tailed, one-tailed (right),

TABLE 7 Statistical results on BiLSTM-FL (resampling) versus BiLSTM-FL

	Wilcoxon-Signed-Rank test				
Program	Two-tailed	One-tailed (right)	One-tailed (left)	Conclusion	A-Test
chart	4.65E−02	8.19E−01	2.92E−02	BETTER	0.92
gzip	4.86E−02	7.05E−01	3.94E−02	BETTER	0.80
libtiff	2.25E−02	9.11E−01	1.40E−02	BETTER	0.80
Math	4.93E−02	7.89E−01	3.95E−02	BETTER	0.87
mokito	3.96E−01	7.12E−01	4.94E−02	BETTER	0.66
python	6.79E−03	9.78E−01	5.02E−03	BETTER	0.83
space	8.69E−03	9.96E−01	4.57E−03	BETTER	0.81
time	6.55E−01	5.00E−01	8.14E−01	SIMILAR	0.50
nanoxml:v1	4.55E−02	8.14E−01	4.91E−02	BETTER	0.50
nanoxml:v2	1.00E+00	6.05E−01	6.05E−01	SIMILAR	0.56
nanoxml:v3	2.77E−02	9.89E−01	1.80E−02	BETTER	0.77
nanoxml:v5	1.80E−02	9.63E−01	1.86E−02	BETTER	0.73
Total	3.47E−05	9.99E−01	1.76E−05	BETTER	0.74

and one-tailed (left) are 6.98E−03, 9.82E−01, and 5.45E−03, respectively. It means that the *EXAM* of CNN-FL using resampling is significantly less than that of CNN-FL with the original test suite. Therefore, we obtain a BETTER conclusion; that is, our test case resampling approach significantly improves the localization effectiveness of CNN-FL in *chart*. We can observe that CNN-FL using resampling approach obtains BETTER results in all programs except for two SIMILAR results in *python* and *time* and one WORSE result in *nanoxml:v1*; MLP-FL using resampling approach obtains the BETTER results in all programs except for one SIMILAR result in *nanoxml:v5*; BPNN-FL using resampling approach obtains the BETTER results in all programs; BiLSTM-FL using resampling approach obtains BETTER results in all programs except for two SIMILAR results in *time* and *nanoxml:v2*.

To further assess the difference quantitatively, we use the nonparametric Vargha-Delaney A-test, which is recommended in previous study,39 to evaluate the magnitude of the difference by measuring effect size (scientific significance). For A-test, the bigger deviation of A-statistic from the value of 0.5, the greater difference of the two studied groups. Vargha and Delaney40 suggest that A-test of greater than 0.64 (or less than 0.36) is indicative of "medium" effect size and of greater than 0.71 (or less than 0.29) can be indicative of a promising "large" effect size. Based on the A-Test results in Tables 4-6, and 7, our approach has six "large" effect sizes and two "medium" effect sizes over CNN-FL, 11 "large" effect sizes, and one "medium" effect size over MLP-FL, nine "large" effect sizes and three "medium" effect sizes over BPNN-FL, and eight "large" effect sizes and one "medium" effect size over BiLSTM-FL. Thus, our approach significantly improves the localization effectiveness of the four deep-learning-based fault localization approaches.

Many studies have shown that algorithms trained with balanced data surpass those trained with imbalanced data in performance.[18-20] Thus, our resampling approach generates a balanced test suite. To further verify whether a balanced test suite is better than unbalanced test suite, we use our resampling approach to clone different ratios of failing test cases. Specifically, we use the ratio $\theta = Pnum/Fnum$, where *Pnum* and *Fnum* denote the number of passing test cases and the number of failing test cases. We resample failing tests to generate different test suites with θ = 0.25, θ = 0.5, θ = 1 (i.e., our approach), θ = 1.25, and θ = 1.5. Table 8 shows the statistical results of our approach with θ = 1 versus different ratios in 12 programs. Take (θ = 1) versus (θ = 1.25) as an example. In comparison with (θ = 1.25), our approach (θ = 1) obtains 11 BETTER results (91.67%) on MLP-FL, CNN-FL, and BPNN-FL, respectively, and 12 BETTER results (100%) on BiLSTM-FL. Based on the results of Table 8, we can conclude that the balanced test suites are better than unbalanced ones, being consistent with the previous research. [18-20]

Thus, based on all the results and analysis, we can safely conclude that our test case resampling approach significantly improves CNN-FL, MLP-FL, BPNN-FL, and BiLSTM-FL, showing that leveraging the learning feature to use the oversampling of critical data into deep learning-based fault localization is potential to improve fault localization effectiveness.

4.4 | Threats to validity

There are some threats to the validity of our experiments. We adopted deep learning methods, whose outputs are not stable, meaning that the fault localization results are not the same through different training times. That drawback is caused by a characteristic of neural network technology. To make the results more reliable, we followed the convention by repeating the fault localization process; that is, we computed 10 times and used the average score as the results for the experimental study.

TABLE 8 Statistical results on our approach versus different resampling ratios

Comparison on fault localization approaches					
(θ = 1) vs. (θ = 1.5)		Result	(θ = 1) vs. (θ = 0.5)		Result
MLP-FL	BETTER	10 (83.33%)	MLP-FL	BETTER	11 (91.67%)
	SIMILAR	2 (16.67%)		SIMILAR	1 (8.33%)
	WORSE	0 (0%)		WORSE	0 (0%)
CNN-FL	BETTER	10 (83.33%)	CNN-FL	BETTER	9 (75%)
	SIMILAR	1 (8.33%)		SIMILAR	3 (25%)
	WORSE	1 (8.33%)		WORSE	0 (0%)
BPNN-FL	BETTER	11 (91.67%)	BPNN-FL	BETTER	11 (91.67%)
	SIMILAR	1 (8.33%)		SIMILAR	1 (8.33%)
	WORSE	0 (0%)		WORSE	0 (0%)
BiLSTM-FL	BETTER	11 (91.67%)	BiLSTM-FL	BETTER	10 (83.3%)
	SIMILAR	0 (0%)		SIMILAR	0 (0%)
	WORSE	1 (8.3%)		WORSE	2 (16.67%)
(θ = 1) vs. (θ = 1.25)		Result	(θ = 1) vs. (θ = 0.25)		Result
MLP-FL	BETTER	11 (91.67%)	MLP-FL	BETTER	12 (100%)
	SIMILAR	1 (8.33%)		SIMILAR	0 (0%)
	WORSE	0 (0%)		WORSE	0 (0%)
CNN-FL	BETTER	11 (91.67%)	CNN-FL	BETTER	11 (91.67%)
	SIMILAR	0 (0%)		SIMILAR	1 (8.33%)
	WORSE	1 (8.33%)		WORSE	0 (0%)
BPNN-FL	BETTER	11 (91.67%)	BPNN-FL	BETTER	12 (100%)
	SIMILAR	1 (8.33%)		SIMILAR	0 (0%)
	WORSE	0 (0%)		WORSE	0 (0%)
BiLSTM-FL	BETTER	12 (100%)	BiLSTM-FL	BETTER	10 (83.3%)
	SIMILAR	0 (0%)		SIMILAR	1 (8.33%)
	WORSE	0 (0%)		WORSE	1 (8.33%)

In our experiment, cloning failing test cases to seek a balanced distribution may cause the model to overfit the minority class in some cases. We use dropout to alleviate this problem, and the results show that our approach is effective for improving fault localization. However, it is still difficult to solve the overfitting problem fundamentally. Thus, it is worthwhile to conduct the research on alleviating overfitting problem (e.g., the generation of noisy failing test cases) for more improvement in terms of fault localization effectiveness.

Another threat to external validity is the subject programs used for our experiments. Our subject programs are commonly used in fault localization and program repair, which are all from the real-life development. Thus, the results should be reliable. However, the experimental results may not apply to all programs because there are still many unknown and complicated factors in realistic debugging that could affect the experiment results. For example, since our approach is derived from the existing state-of-the-art localization approaches, it may not hold in some cases where these localization approaches suffer from, for example, the multiple-fault case. Thus, it is worthwhile to conduct the experiments on more large-sized programs with all real faults to further strengthen the experimental results.

We adopt the widely used metrics, *EXAM* and *RImp*, to evaluate localization effectiveness. According to the extensive use of the measurement, the threat is acceptably mitigated.

5 | RELATED WORK

In this section, we briefly survey the class proportion of data sets and fault localization studies, especially coverage-based fault localization and fault localization using learning algorithms. More work on fault localization can refer to the survey2 by Wong et al.

In the field of machine learning and fault localization, the class proportion of data sets has been widely studied. Wong et al.[41] show that the class imbalance phenomenon of data sets has an influence on the efficiency of classification. To some extent, deep learning-based fault localization can be viewed as a pattern of classification problem, which can be influenced by the characteristics of test suites. Japkowizc et al.[13]

empirically draw a conclusion that the class imbalance phenomenon of the training set causes a negative impact to classifier problems. Baudry et al.[42] conduct the experiments and show that fewer test cases could achieve the same fault localization effectiveness. Hao et al. [43] improve the localization effectiveness by using test suite reduction techniques. However, Yu et al.[44] suggest that test suite reduction techniques may reduce the effectiveness of fault localization. Gong et al.[31] conduct an experimental study showing that the identical number of passing test cases and the failing test cases is beneficial for fault localization. In contrast, our approach uses data resampling to alleviate the biased problem in deep-learning-based fault localization.

Machine learning techniques are used in the context of fault localization based on statement coverage and test results of test cases. Wong et al.24 propose a fault localization approach based on back-propagation (BP) neural network, which introduces and implements a simple structure. Due to the drawbacks of BP networks (e.g., paralysis), Wong et al.[45] afterwards propose another approach based on radial basis function (RBF) networks to improve fault localization using BP networks. Recently, deep learning method has witnessed a rapid development to tackle the limitations of traditional machine learning techniques and is successfully applied in many disciplines, for example, computer vision and natural language processing. Based on the methods proposed by Wong et al. and advantages of deep learning methods, Zheng et al.[12] construct a fault localization model using Multi-Layer Perceptrons (MLP). Zhang et al.[9] use dynamic slices to enhance fault localization effectiveness in the context of deep neural networks. Briand et al.[36] propose a fault localization method using decision tree algorithm and construct those rules that classify test cases into various partitions. Zhang et al.[10] explore more on deep learning and propose a fault localization approach based on convolutional neural networks. Differently, our approach aims at using test case resampling to alleviate the biased problem in those localization approaches based on learning algorithms.

Coverage-based fault localization techniques convert program spectrum data from test executions to suspiciousness score of program entities and rank them in descending order.[46] Among existing coverage-based localization methods, spectrum-based fault localization is the most popular one. Chen et al.[47] propose Jaccard technique, Jones et al.[48] propose Tarantula technique, and Abreu et al.[49] apply Ochiai technique. The three techniques are widely used and compared techniques in the subsequent studies. Wong et al.50 use data and control flow to present many suspiciousness evaluation formulas such as Wong1–3 and Wong3′. Wong et al.[51,52] also propose DStar (D*) based on crosstab. Xie et al.[53,54] investigate more than 30 suspiciousness evaluation formulas and theoretically prove the maximal formulas for single-fault scenarios, and further, they[55] discussed the limits of purely coverage-based fault localization approaches. Pearson et al.[56] afterward conduct a systematical study on evaluating and summarizing the existing coverage-based fault localization. Papadakis et al.[57] propose mutation-based fault localization, in which mutants that are killed mostly by failing tests provide a good indication about the location of a fault. Coverage-based fault localization does not use learning algorithms and thus has no learning feature from learning algorithms such as deep learning. In contrast, our approach studies deep learning-based fault localization and utilizes its learning feature to propose a test case augmentation approach for improving deep-learning-based fault localization's effectiveness.

6 | CONCLUSION

The recent rapid progress on deep learning shows the promising potential of many neural network architectures in making sense of data. The fault localization community notices the potential of deep learning and has proposed deep-learning-based fault localization showing promising results. This paper explores more on deep-learning-based fault localization; that is, we notice its learning feature distinct from traditional fault localization approach. Due to the learning feature, if we can identify a weak but beneficial experience learned from some specific data, we can leverage the learning feature by oversampling the data to argument this weak but beneficial experience.

Following this idea, we propose a test case resampling approach for deep-learning-based fault localization. This approach first identifies failing test cases as the critical data corresponding to the weak but beneficial experience, then use iterative oversampling to clone those failing test cases into the original test cases, and finally feeds the new test suite into deep-learning-based fault localization. To evaluate the effectiveness of our approach, we conduct a large-scale experimental study on eight programs with real faults and four programs with seeded faults. The results show that our test case resampling approach significantly improves fault localization effectiveness, revealing that oversampling critical data is an effective way of using the learning feature.

In future, we plan to optimize our test case resampling approach by introducing the context into the learning process. Furthermore, we will explore more on the learning feature of deep-learning-based fault localization and obtain more insights of this feature beneficial for improving fault localization effectiveness. Besides, how to reduce the effect as far as possible and whether there are differences in various kinds of faults are also of great interest to our research.

ACKNOWLEDGEMENTS

This work is partially supported by the Guangxi Key Laboratory of Trusted Software (no. kx202008), the National Natural Science Foundation of China (nos. 61602504, 61379054, and 61672529), the Fundamental Research Funds for the Central Universities (no. 2019CDXYRJ0011), and the Scientific Research Fund of Hunan Provincial Education Department (no. 15A007).

ORCID

Yan Lei https://orcid.org/0000-0003-4504-6806

REFERENCES

1. Debroy V, Wong WE. A consensus-based strategy to improve the quality of fault localization. *Software: Pract Exper*. 2013;43(8):989-1011.
2. Wong WE, Gao R, Li Y, Rui A, Wotawa F. A survey on software fault localization. *IEEE Trans Softw Eng (TSE)*. 2016;42(8):707-740.
3. Le TDB, Oentaryo RJ, Lo D. Information retrieval and spectrum based bug localization: better together. In: Joint Meeting on Foundations of Software Engineering (FSE 2015); 2015; Bergamo, Italy:579-590.
4. Sun C, Khoo SC. Mining succinct predicated bug signatures. In: Joint Meeting on Foundations of Software Engineering (FSE 2013); 2013; Saint Petersburg, Russia:576-586.
5. Chan WK, Cai Y. In quest of the science in statistical fault localization. *Softw: Pract Exper*. 2013;43(8):971-987.
6. Neelofar N, Naish L, Ramamohanarao K. Spectral-based fault localization using hyperbolic function. *Softw: Pract Exper*. 2018;48(3):641-664.
7. Tu J, Xie X, Chen TY, Xu B. On the analysis of spectrum based fault localization using hitting sets. *J Syst Softw*. 2019;147:106-123.
8. Lecun Y, Bengio Y, Hinton G. Deep learning. *Nature*. 2015;521(7553):436.
9. Zhang Z, Lei Y, Tan Q, Mao X, Zeng P, Chang X. Deep learning-based fault localization with contextual information. *IEICE Trans Info Syst*. 2017;100(12):3027-3031.
10. Zhang Z, Lei Y, Mao X, Li P. CNN-FL: An effective approach for localizing faults using convolutional neural networks. In: The 26th International Conference on Software Analysis, Evolution and Reengineering (SANER 2019). IEEE; 2019; Hangzhou, China, China:445-455.
11. Zheng W, Hu D, Wang J. Fault localization analysis based on deep neural network. *Math Probl Eng*. 2016;2016:1-11.
12. Japkowicz N, Stephen S. The class imbalance problem: a systematic study. *Intell Data Anal*. 2002;6(5):429-449.
13. Guo H, Viktor HL. Learning from imbalanced data sets with boosting and data generation: the DataBoost-IM approach. *Sigkdd Explor*. 2004;6(1):30-39.
14. Rui A, Zoeteweij P, Golsteijn R, Gemund AJCV. A practical evaluation of spectrum-based fault localization. *J Syst Softw*. 2009;82(11):1780-1792.
15. Lei Y, Sun C, Mao X, Su Z. How test suites impact fault localization starting from the size. *IET Softw*. 2018;12(3):190-205.
16. Tantithamthavorn C, Hassan AE, Matsumoto K. The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. *IEEE Trans Softw Eng*. 2018:1-22.
17. Zhang L, Yan L, Zhang Z, Zhang J, Chan WK, Zheng Z. A theoretical analysis on cloning the failed test cases to improve spectrum-based fault localization. *J Syst Softw*. 2017;129:35-57.
18. He H, Garcia EA. Learning from imbalanced data. *IEEE Trans Knowl Data Eng (TKDE)*. 2008;21(9):1263-1284.
19. Krawczyk B. Learning from imbalanced data: open challenges and future directions. *Progr Artif Intell*. 2016;5(4):221-232.
20. Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE: synthetic minority over-sampling technique. *J Artif Intell Res*. 2002;16(1):321-357.
21. He H, Bai Y, Garcia EA, Li S. ADASYN: adaptive synthetic sampling approach for imbalanced learning. In: 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence); 2008; Hong Kong, China:1322-1328.
22. Artzi S, Dolby J, Tip F, Pistoia M. Directed test generation for effective fault localization. In: The 19th International Symposium on Software Testing and Analysis (ISSTA 2010). ACM; 2010; Trento, Italy:49-60.
23. Wong W, Qi Y. BP neural network-based effective fault localization. *Int J Softw Eng Knowl Eng*. 2009;19(04):573-597.
24. Graves A, Mohamed Ar, Hinton G. Speech recognition with deep recurrent neural networks. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE; 2013:6645-6649.
25. Jarrett K, Kavukcuoglu K, Ranzato M, Lecun Y. What is the best multi-stage architecture for object recognition? In: IEEE International Conference on Computer Vision (ICCV 2010); 2010; Kyoto, Japan:2146-2153.
26. Lecun Y, Huang FJ, Bottou L. Learning methods for generic object recognition with invariance to pose and lighting. In: The 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2014), Vol. 2; 2004; Columbus, OH, USA:II-104.
27. Lee H, Grosse R, Ranganath R, Ng AY. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In: The 26th Annual International Conference on Machine Learning (ICML 2009); 2009; Montreal, Canada:609-616.
28. Pinto N, Doukhan D, DiCarlo JJ, Cox DD. A high-throughput screening approach to discovering good forms of biologically inspired visual representation. *PLoS Comput Biol*. 2009;5(11):e1000579.
29. Turaga SC, Murray JF, Jain V, et al. Convolutional networks can learn to generate affinity graphs for image segmentation. *Neural Comput*. 2010;22(2):511-538.
30. Gong C, Zheng Z, Li W, Hao P. Effects of class imbalance in test suites: an empirical study of spectrum-based fault localization. In: Computer Software and Applications Conference (COMPSAC 2012); 2012; Izmir, Turkey:470-475.
31. Huang S, Gao M, Yang D, Huang X, Elgammal A, Zhang X. Unbalanced graph-based transduction on superpixels for automatic cervigram image segmentation. In: The 12th International Symposium on Biomedical Imaging (ISBI 2015). IEEE; 2015; New York, NY, USA:1556-1559.
32. Jalali D, Ernst MD. Defects4J: a database of existing faults to enable controlled testing studies for Java programs. In: International Symposium on Software Testing and Analysis (ISSTA 2014); 2014; Hilton San Jose, Bay Area, California, USA:437-440.
33. Mao X, Lei Y, Dai Z, Qi Y, Wang C. Slice-based statistical fault localization. *J Syst Softw*. 2014;89(1):51-62.
34. Debroy V, Wong WE, Xu X, Choi B. A grouping-based strategy to improve the effectiveness of fault localization techniques. In: International Conference on Quality Software (QSIC 2010); 2010; Zhangjiajie, China:13-22.
35. Briand LC, Labiche Y, Liu X. Using machine learning to support debugging with tarantula. In: The IEEE International Symposium on Software Reliability (ISSRE 2007); 2007; Trollhattan, Sweden:137-146.
36. Lei Y, Mao X, Dai Z, Wang C. Effective statistical fault localization using program slices. In: Computer Software and Applications Conference (COMPSAC 2012); 2012; Izmir, Turkey:1-10.
37. Corder GW, Foreman DI. *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*. Wiley; 2013.
38. Arcuri A, Briand L. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: 2011 33rd International Conference on Software Engineering (ICSE). IEEE; 2011; Honolulu, HI, USA:1-10.

39. Vargha A, Delaney HD. A critique and improvement of the CL common language effect size statistics of McGraw and Wong. *J Edu Behav Stat*. 2000; 25(2):101-132.
40. Wong E, Wei T, Qi Y, Zhao L. A crosstab-based statistical method for effective fault localization. In: International Conference on Software Testing, Verification, and Validation (ICST 2008); 2008; Lillehammer, Norway:42-51.
41. Baudry B. Improving test suites for efficient fault localization. In: International Conference on Software Engineering (ICSE 2006); 2006; Shanghai, China:82-91.
42. Hao D, Pan Y, Zhang L, Zhao W, Mei H, Sun J. A similarity-aware approach to testing based fault localization. In: IEEE International Conference on Automated Software Engineering (ASE 2005); 2005; Long Beach, CA, USA:291-294.
43. Yu Y. An empirical study of the effects of test-suite reduction on fault localization. In: ACM/IEEE International Conference on Software Engineering (ICSE 2009); 2009; Leipzig, Germany:201-210.
44. Wong WE, Debroy V, Golden R, Xu X, Thuraisingham B. Effective software fault localization using an RBF neural network. *IEEE Trans Reliab*. 2012; 61(1):149-169.
45. Naish L, Lee HJ, Ramamohanarao K. A model for spectra-based software diagnosis. *ACM Trans Softw Eng Methodol*. 2011;20(3):1-32.
46. Chen MY, Kiciman E, Fratkin E, Fox A, Brewer E. Pinpoint: problem determination in large, dynamic internet services. In: International Conference on Dependable Systems and Networks (DSN 2002); 2002; Washington, DC, USA, USA:595-604.
47. Jones JA. Fault localization using visualization of test information. In: International Conference on Software Engineering (ICSE 2004); 2004; Edinburgh, UK, UK:54-56.
48. Abreu R, Zoeteweij P, Van Gemund AJ. An evaluation of similarity coeffcients for software fault localization. In: Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing (PRDC 2006); 2006; Riverside, USA:39-46.
49. Wong WE, Qi Y, Zhao L, Cai KY. Effective fault localization using code coverage. In: Computer Software and Applications Conference (COMPSAC 2007); 2007; Beijing, China:449-456.
50. Wong WE, Debroy V, Choi B. A family of code coverage-based heuristics for effective fault localization. *J. Syst. Softw*. 2010;83(2):188–208.
51. Wong WE, Debroy V, Li Y, Gao R. Software fault localization using DStar (D*). In: IEEE Sixth International Conference on Software Security and Reliability (SERE 2012); 2012; Gaithersburg, Maryland, USA:21-30.
52. Xie X, Chen TY, Kuo FC, Xu B. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. *ACM Trans Softw Eng Methodol (TOSEM)*. 2013;22(4):31.
53. Xie X, Kuo FC, Chen TY, Yoo S, Harman M. *Provably Optimal and Human-Competitive Results in SBSE for Spectrum Based Fault Localisation*. Germany: Springer, Berlin, Heidelberg; 2013:224-238.
54. Yoo S, Xie X, Kuo FC, Chen TY, Harman M. Human competitiveness of genetic programming in SBFL: Theoretical and Empirical Analysis. *ACM Trans Softw Eng Methodol*. 2017;26(1):4:1-4:30.
55. Pearson S, Campos J, Just R, et al. Evaluating and improving fault localization. In: The International Conference on Software Engineering (ICSE 2017); 2017; Buenos Aires, Argentina:609-620.
56. Papadakis M, Traon YL. Metallaxis-FL: mutation-based fault localization. *Softw Test Verif Rel*. 2015;25(5-7): 605-628.

How to cite this article: Zhang Z, Lei Y, Mao X, Yan M, Xu L, Wen J. Improving deep-learning-based fault localization with resampling. *J Softw Evol Proc*. 2020;e2312. https://doi.org/10.1002/smr.2312