Bug Localization with Semantic and Structural Features using Convolutional Neural Network and Cascade Forest

Yan Xiao, Jacky Keung, Qing Mi, Kwabena E. Bennin
Department of Computer Science
City University of Hong Kong, Kowloon, Hong Kong
yanxiao6-c@my.cityu.edu.hk,Jacky.Keung@cityu.edu.hk,{Qing.Mi,kebennin2-c}@my.cityu.edu.hk

ABSTRACT

Background: Correctly localizing buggy files for bug reports together with their semantic and structural information is a crucial task, which would essentially improve the accuracy of bug localization techniques. ***Aims:*** To empirically evaluate and demonstrate the effects of both semantic and structural information in bug reports and source files on improving the performance of bug localization, we propose CNN_Forest involving convolutional neural network and ensemble of random forests that have excellent performance in the tasks of semantic parsing and structural information extraction. ***Method:*** We first employ convolutional neural network with multiple filters and an ensemble of random forests with multi-grained scanning to extract semantic and structural features from the word vectors derived from bug reports and source files. And a subsequent cascade forest (a cascade of ensembles of random forests) is used to further extract deeper features and observe the correlated relationships between bug reports and source files. CNN_Forest is then empirically evaluated over 10,754 bug reports extracted from AspectJ, Eclipse UI, JDT, SWT, and Tomcat projects. ***Results:*** The experiments empirically demonstrate the significance of including semantic and structural information in bug localization, and further show that the proposed CNN_Forest achieves higher Mean Average Precision and Mean Reciprocal Rank measures than the best results of the four current state-of-the-art approaches (NP-CNN, LR+WE, DNNLOC, and BugLocator). ***Conclusion:*** CNN_Forest is capable of defining the correlated relationships between bug reports and source files, and we empirically show that semantic and structural information in bug reports and source files are crucial in improving bug localization.

CCS CONCEPTS

• **Software and its engineering** → **Software reliability**; **Maintaining software**; • **Computing methodologies** → *Neural networks*; Supervised learning by classification;

KEYWORDS

bug localization, convolutional neural network, cascade forest, word embedding, semantic information, structural information

ACM Reference Format:
Yan Xiao, Jacky Keung, Qing Mi, Kwabena E. Bennin. 2018. Bug Localization with Semantic and Structural Features using Convolutional Neural Network and Cascade Forest. In *EASE'18: 22nd International Conference on Evaluation and Assessment in Software Engineering 2018, June 28–29, 2018, Christchurch, New Zealand*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3210459.3210469

EASE'18, June 28–29, 2018, Christchurch, New Zealand

ACM ISBN 978-1-4503-6403-4/18/06…$15.00
https://doi.org/10.1145/3210459.3210469

1 INTRODUCTION

Software bugs are coding mistakes that tend to induce unexpected or abnormal behaviors in a software project [4]. Once discovered, these bugs are described in bug reports or issue reports submitted by developers, testers, or users of the project system. A bug report provides information about the scenarios in which the software does not behave as expected and how to reproduce this abnormal behavior [23]. Bug reports are the main source of information for developers to understand and analyze bugs. When a bug report is received and assigned to a developer for fixing, the developer needs to investigate the contents in the bug report and then search through the source files to locate the potential buggy files that are relevant to the bug. This process is called bug localization [32]. However, manually localizing buggy files is painstaking for developers if there are tens of thousands of source files. In recent years, many automatic bug localization techniques have been proposed to alleviate the burden of software maintenance teams.

Most recent bug localization techniques transform the terms in bug reports and source files into textual representations and adopt some models to measure the textual similarity between them. For example, Latent Dirichlet Allocation (LDA)-based techniques [2, 27] were used to represent bug reports and source files, and BugLocator proposed by Zhou et al. [52] applied revised Vector Space Model (VSM) [12] to calculate the textual similarity. These techniques focused more on textual similarity than the semantics of the terms in bug reports and source files.

Current state-of-the-art techniques have acknowledged the lexical mismatch problem between the texts in bug reports and code tokens in source files caused by the ignorance of the semantic information in them. Several studies [21, 22, 48, 49]

try to bridge the lexical gap by including semantic information of the bug reports and source files. Ye et al. [49] applied word embedding techniques to obtain vectors of words in bug reports and source files and measured the similarity between them. Lam et al. [21, 22] employed deep neural networks to correlate the frequently occurred terms in bug reports and source files. These techniques considered the semantics but inadvertently ignored the structural information of both bug reports and source files. NP-CNN proposed by Huo et al. [16] employed convolutional neural networks (CNNs) to extract structural features from both bug reports and source files and used a fully-connected network to fuse features. However, the model did not consider the semantic information in the bug reports and source files.

The existing challenge is not only how to make effective use of the semantic information but also how to extract the structural information from bug reports and source files. Additionally, there are obvious differences between natural languages and programming languages [34] and these differences can stifle the localization process. Compared to natural languages, programs contain rich statistical properties and more stringent structural information [14, 29]. Therefore, we propose to use the techniques based on tree-structure that have shown excellent performance on extracting the structural and syntactic information in dealing with source code [25, 29, 31, 44].

To address the aforementioned issues, this paper proposes CNN_Forest based on CNN and random forest for bug localization. The proposed model first converts the bug reports and source files into word vectors by the word embedding technique to preserve the semantics of them. Due to the differences between natural languages and programming languages, we then employ different techniques to extract features from bug reports and source files. CNN with multiple filters is applied to parse the semantics and extract features from bug reports by adopting the procedure leveraged by the natural language processing techniques [13, 19]. Since source files are composed of code tokens, an ensemble of random forests involving various decision trees is used to extract the structural information from them. After the feature extraction, cascade forest is adopted to extract further features, and the correlated relationships between bug reports and source files are learned benefiting from the alternate cascade structure that is similar to the layer-structure in deep learning.

The main contributions of this paper are listed as follows:

- We leverage two different techniques to extract features respectively from bug reports and source files according to the differences between natural languages and programming languages.
- Both semantic and structural information of bug reports and source files are extracted. The ensemble of random forests is applied to detect the structural information from the source code. The alternate cascade forest works as the layer-structure in deep learning to learn the correlated relationships between bug reports and source files.
- A set of experiments is conducted to validate the feasibility and effectiveness of CNN_Forest. The experimental results confirm the significance of applying different techniques to extract features from bug reports and source code.

The remainder of the paper is structured as follows. Section 2 reviews the background knowledge related to CNN_Forest. This is followed in Section 3 by a detailed description of the proposed model. Section 4 presents the experimental preparations and results, followed by the discussion in Section 5. The related works are reviewed in Section 6. We conclude this paper and discuss the future works in Section 7.

2 BACKGROUND

This section provides the backgrounds of convolutional neural network and random forest.

2.1 Convolutional Neural Network (CNN)

CNN is inspired by biological processes [11] and is variations of multi-layer perceptron [15]. Figure 1 illustrates the architecture of a typical CNN with two convolutional layers and pooling layers for image processing. The input image has a dimension of 28×28. The first convolutional layer uses 32 filters for each filter size (e.g., 5×5) to calculate convolutions on the input image. The obtained 32 feature maps with the dimension of 28×28 are then added by a bias to input into a non-linear function (e.g., Rectified Linear Units (ReLU) [20]). The first pooling layer is used to reduce the dimension of the feature maps into 14×14. The mean-over-time pooling or max-over-time pooling operation [9] is frequently used in the pooling layer. The second convolutional and pooling layer operate as the first ones but use more filters. Lastly, a fully-connected layer is allocated after a couple of convolutional and pooling layers. A non-linear function is also used in the fully-connected layer to increase the nonlinear properties of the network. The final *softmax* function [30] operates the classification.

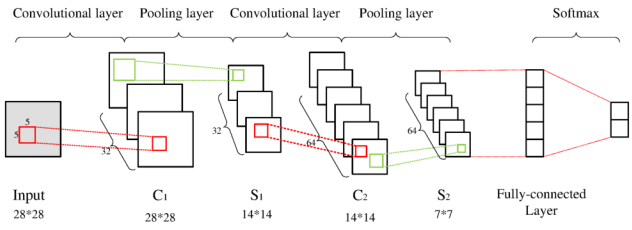

Figure 1: The Architecture of a Typical CNN.

Different from traditional image processing algorithms that extract features manually from the original images, CNN can omit the pre-treatment process, which implies that original images can be input into the CNN directly. Moreover, a full connection is used between different layers in traditional neural networks, which results in a huge number of parameters. Therefore, a lot of time is required to train

the network and it is not always successful. To eliminate this drawback, CNN adopts parameter sharing scheme. CNNs have wide applications in image recognition [6], video analysis [17], natural language processing [13] and so on [7, 42].

2.2 Random Forest

Random Forest (RF) is an ensemble learning method based on multiple decision trees [3]. As shown in Figure 2, RF first extends the original training set $\{D_1, D_2, ..., D_n\}$ into k sub-training sets $\{Sub_1, Sub_2, ..., Sub_k\}$ using random sampling with replacement. The dimension of the extended training set is the same as that of the original training set. Then a decision tree is trained on each sub-training set and a total of k decision trees {Decision Tree 1, Decision Tree 2, ..., Decision Tree k} is obtained. The final output is determined by combining the results of the k decision trees, e.g., averaging or voting.

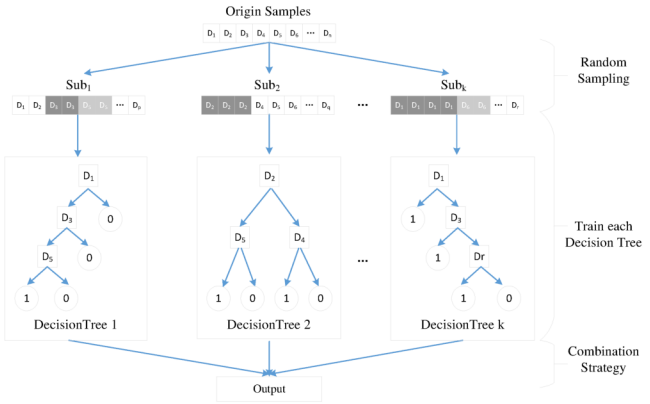

Figure 2: The Illustration of Random Forest.

RF introduces randomness into decision trees, so as to obtain the integrated learners with more powerful ability of generalization. The property of randomness is shown in two aspects. Firstly, based on simple random sampling with replacement, the original training set is divided into multiple sub-training sets, which usually contain different duplicate samples. Secondly, during the process of feature selection in decision trees, a subset of all the current features is randomly selected. Then the optimal feature is selected from the sub-feature set as the root node. By adding these two random properties, the generalization of RF is significantly improved. However, RF usually converges slowly. Fortunately, as the number of decision tree increases, RF can converge to lower generalization error. Moreover, RF is simple with low cost and easy to implement, and it has achieved good performance in many practical tasks.

3 THE PROPOSED APPROACH

In this section, we will describe the proposed model, CNN_Forest, a convolutional neural network (CNN) and random forest-based approach for bug localization. Figure 3 presents the overall workflow of the proposed approach. As shown in Figure 3, the words in bug reports and source files are first transformed into vectors by the word embedding technique to preserve the semantic information. Due to the different characteristics of bug reports and source files, their vectors are then processed respectively by CNN with multiple filters and the ensemble of random forests with multi-grained scanning to extract the features and reduce dimensions. After that, these feature vectors are fed into the cascade forest to learn the correlated relationships between bug reports and source files, and to further extract features to obtain the class vectors, whose average values are the final class vectors (output).

3.1 Data Preprocessing

Bug reports usually consist of summaries and descriptions but one of them may be vacant. We thus first combine them into one document as the new bug reports. Each text in bug reports and source files is then filtered for punctuation (e.g., transform gtk_widget_get_window into gtk widget get window) using the Natural Language Toolkit (NLTK) [39] package[1]. In addition, some texts are combinations of multiple words, which are also needed to be split into the single word based on the CamelCase Naming Convention [1]. For example, "WorkbenchWindowConfigurer" can be split into "Workbench", "Window" and "Configurer". Finally, all texts are cast to lower case.

3.2 Word Embedding

After pre-processing, all the words in bug reports and source files are converted into word vectors by a word embedding technique (word2vec [28]). As reported in the literature [49], the word vectors obtained by word2vec from project-specific and Wikipedia corpus have similar performance for bug localization. Therefore, we use the Skip-gram model pre-trained on the Wikipedia corpus to obtain the word vectors. For words not present in the corpus, we randomly initialize them [19].

3.3 Feature Extraction

Given bug reports $B = \{b_1, ..., b_i, ...\}$ and source code files $C = \{c_1, ..., c_j, ...\}$, each bug report $b_i \in \mathbb{R}^{n_{bs} \times n_{bw} \times k}$ and each source file $c_j \in \mathbb{R}^{n_{cl} \times n_{cw} \times k}$ are 3-dimensional vectors after word embedding, where k is the dimension of the word vectors, n_{bs} denotes the maximum number of sentences in bug reports, and n_{bw} denotes the maximum number of words in each sentence, while n_{cl} denotes the maximum number of lines in source files, and n_{cw} denotes the maximum number of words in each line. Next, we will describe how to respectively extract features based on these 3-dimensional vectors from bug reports and source files and simultaneously reduce the dimensions.

CNN with Multiple Filters for Bug Reports

[1] http://www.nltk.org/

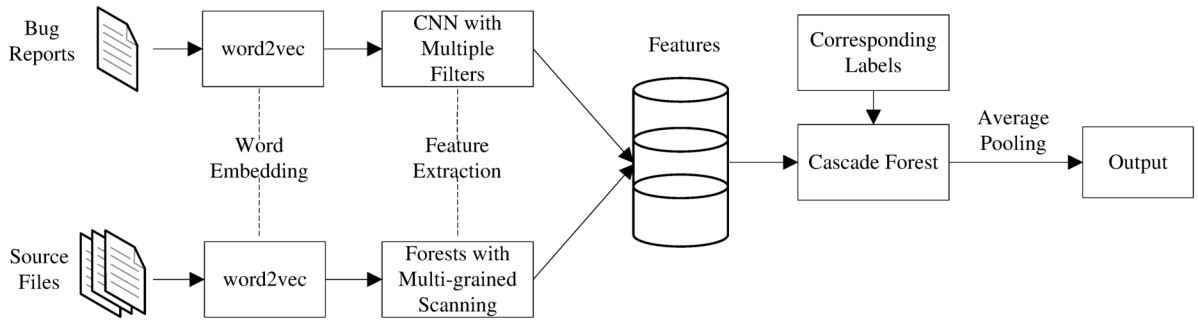

Figure 3: The Overall Workflow of CNN_Forest.

Bug reports are written in natural languages and transforming them into word vectors will still contain the semantic information of bug reports. However, this kind of 3-dimensional vectors for each bug report will cause high memory overhead if directly fed into the following cascade forest. In addition, CNN performs excellently in the semantic parsing field of natural language processing because of the convolving filters that can extract local features. Therefore, we adopt the CNN with multiple filters to extract the features of bug reports and reduce the dimensions.

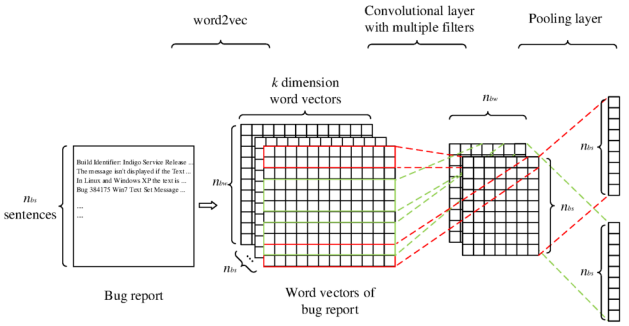

Figure 4: The Feature Extraction of Bug Reports using CNN.

Figure 4 illustrates how to extract features from bug reports using CNN with two kinds of filter sizes, which are the sliding windows with the size of $2 \times k$ and $3 \times k$. The convolutional layer employs a filter $w \in \mathbb{R}^{d \times k \times n_{bs}}$ ($d = 2, 3$ in Figure 4) to convolve the vectors of each bug report. The Rectified Linear Units (ReLU) [20] function is then used to extract more nonlinear features from the vectors. After the convolutional layer, the dimension of each bug report is reduced from $n_{bs} \times n_{bw} \times k$ to $n_{bs} \times n_{bw}$, followed by the pooling layer to further reduce the dimension of the features into $n_{bs} \times 1$. In Figure 4, the CNN has two kinds of filter sizes in each of which there is only one filter. To preserve the integrity of the features in bug reports, we implement the CNN with not only more kinds of filter sizes but also more kinds of filters for each filter size. For example, if we adopt two kinds of filter sizes and each filter size contains 100 kinds of filters, then the dimension of the features is $200n_{bs} \times 1$ after the pooling layer. The size and number of filters can be fine-tuned during training.

Ensemble of Random Forests with Multi-grained Scanning for Source Files

Source files are composed of code tokens that are similar to the texts in natural languages, but the programming languages contain more structural information than natural languages. Since the source code can be parsed into the abstract syntax tree and also be reverted from the abstract syntax tree, we will adopt the ensemble of random forests involving multiple decision trees to extract the structural features from the source files.

Figure 5 illustrates how to extract features from source files using the ensemble of random forests with multi-grained scanning. Firstly, different sliding windows with the size of $d \times k$ ($d = 2, 3$ in Figure 5) are employed to scan the word vectors of the source files. The new instance vectors with the dimension of $n_{cl}(n_{cw} - d + 1) \times k$ are then generated from the original features (the dimension is $n_{cl} \times n_{cw} \times k$) by the multi-grained scanning process. These new instances obtained from the same sliding window are further fed to train the completely-random tree forest (CRF) [26] and random forest (RF) [3]. Two types of forests are used to encourage diversity that is crucial for ensemble models [53]. For each kind of sliding window, the new feature vectors generated from each forest have the dimension of $2 \times n_{cl}(n_{cw} - d + 1)$, suppose there are 2 classes, and then these feature vectors are concatenated as the transformed features. As shown in Figure 5, the final $4n_{cl}(n_{cw} - d + 1) \times 1$ dimensional feature vectors are generated from the original $n_{cl} \times n_{cw} \times k$ dimensional input for each kind of sliding window with the size of $d \times k$.

3.4 Cascade Forest

The *deep* in deep learning is beneficial from the layer-by-layer processing of the input to extract deep features from them. Similar to this, the cascade structure in cascade forest performs as the layer structure in the deep learning model. In order to extract deeper features from the bug reports and source files, and learn the correlated relationships between them, cascade forest is employed to process their feature vectors obtained respectively from the CNN with multiple

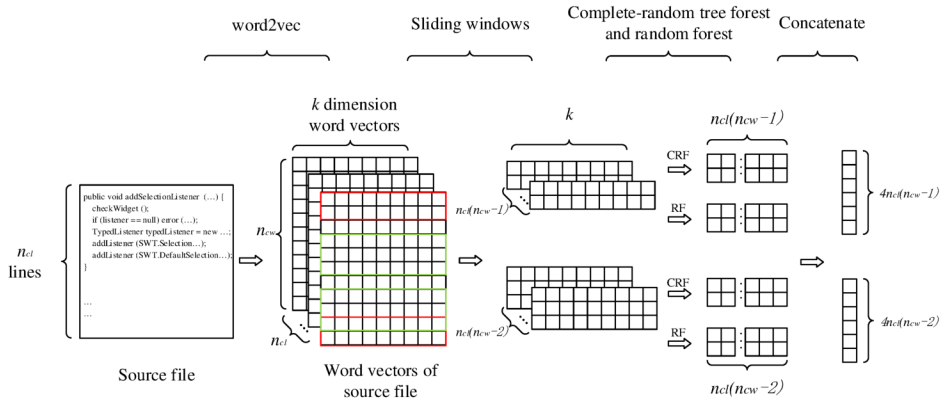

Figure 5: The Feature Extraction of Source Files using Ensemble of Random Forests.

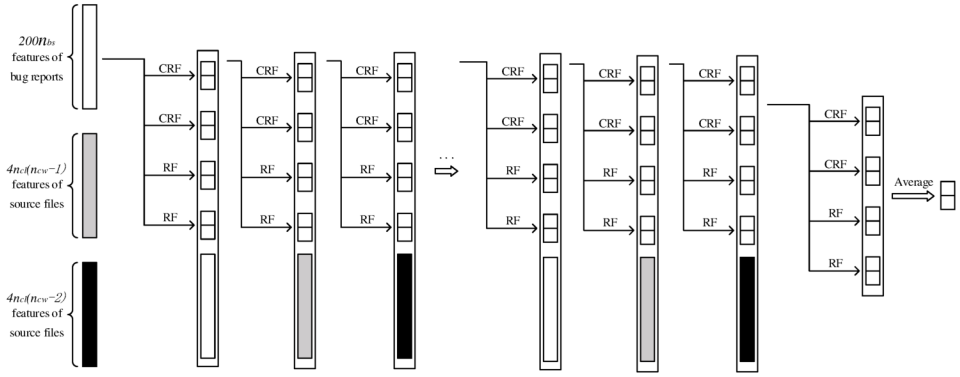

Figure 6: Cascade Forest to Further Extract Features and Learn the Correlated Relationships between Bug Reports and Source Files.

filters and the ensemble of random forests with multi-grained scanning. Similar to the method in the literature [54], two completely-random tree forests (CRF) and two random forests (RF) are leveraged to constitute one ensemble in the cascade forest, where there are several such ensembles. This kind of ensemble structure of ensembles is able to extract deeper features from the bug reports and source files.

Figure 6 depicts how the cascade forest deals with the feature vectors ($200n_{bs} \times 1$) extracted from the bug reports and those ($4n_{cl}(n_{cw} - d + 1) \times 1$) from source files. The $200n_{bs} \times 1$ dimensional feature vectors of bug reports are first fed into the first level of cascade forest with two CRFs and two RFs to generate the first four class vectors. These class vectors are then concatenated with the original $200n_{bs} \times 1$ dimensional feature vectors to be fed into the second level. Likewise, the second class vectors are concatenated with the original $4n_{cl}(n_{cw} - 1) \times 1$ dimensional feature vectors extracted from the source files using the first type of sliding window to be the input of the third level. Similarly, the third class vectors are concatenated with the $4n_{cl}(n_{cw} - 2) \times 1$ dimensional vectors extracted from the second type of sliding window. These cascade procedures will continue until there are no significant improvements during training. This alternate cascade structure is used to learn the correlated relationships between bug reports and source files. The final class vectors are the average values of the last four class vectors.

4 EXPERIMENTS

Several experiments are conducted to evaluate the performance of our model CNN_Forest. We will first describe the experimental preparations in Section 4.1. The experimental results and the corresponding analyses will be presented in Section 4.2.

4.1 Experimental Preparations

This section will present the experimental preparations including datasets, experimental settings, evaluation metrics, and competitors.

Datasets

Table 1: Subject Projects.

Project	Time Range	# of Bug Reports for Evaluation	# of Bug Reports for Tuning	# of Bug Reports for Training	# of Bug Reports for Testing
AspectJ	03/2002-01/2014	593	100	400	93
Eclipse UI	10/2001-01/2014	3,656	1,500	500	1,656
JDT	10/2001-01/2014	2,632	1,500	500	632
SWT	02/2002-01/2014	2,817	1,500	500	817
Tomcat	07/2002-01/2014	1,056	400	500	156

In this paper, we use five open-source Java projects (AspectJ[2], Eclipse UI[3], JDT[4], SWT[5] and Tomcat[6]) to evaluate the performance of CNN_Forest. The before-fixed version of the source code [48] in each project is obtained to perform the experiments. We make the before-fixed version [48] of the source code in each project publicly available[7] and provide the tool used to construct the dataset. To make the comparison with existing techniques easier, a splitting strategy similar to that in [49] is adopted. The oldest bug reports of each project are used for tuning the model (validation set) while the older bug reports are for training the model (training set) and the newest bug reports are for testing the model (testing set). Table 1 provides more details about the number of bug reports in each set.

Experimental settings

The hyper-parameters of CNN_Forest include the dimension of word vectors (k), the size and the number of filters used in CNN, the size of sliding windows in the ensemble of random forests for scanning, and the number of trees in each forest. According to the work [46], the dimension of word vectors $k = 100$ can achieve comparable performance and consume relatively less time. The filter size ranging from 4 to 6 and the number of filters equal to 100 performs well in bug localization, which implies that the size of sliding windows in CNN can be chosen from $4 \times k$, $5 \times k$, and $6 \times k$. The size of sliding windows in the ensemble of random forests to scan the source files is set to 2 and 3 as suggested in the literature [16] considering the characteristics of the statements in source code. In order to evaluate the generalized effects of the forest ensemble and cascade forest on bug localization, the default number (500) of the trees in each forest as recommended by [54] is used in this paper.

Evaluation metrics

The mean average precision (MAP) and mean reciprocal rank (MRR) are used to evaluate the performance of CNN_Forest, which are frequently used in existing studies of bug localization [22, 48, 49]. The higher the values of MRR and MAP, the better the performance of the bug localization technique.

- *MRR* is the mean of the accumulations of the inverse of the ranks of the first correctly-located buggy file for each bug, which is computed in this paper by:

$$MRR = \frac{1}{Q}\sum_{i=1}^{Q}\frac{1}{first_i} \tag{1}$$

where Q is the number of bug reports, $first_i$ represents the ranks of the first correctly-located buggy file for the i_{th} bug report.

- *MAP* is the mean of average precision values for all Q bug reports. The following is the formulation of MAP used in the field of bug localization:

$$MAP = \frac{1}{Q}\sum_{i=1}^{Q}\sum_{r=1}^{R}\frac{(N_T(r)/r) \times ind(r)}{N_B(i)} \tag{2}$$

where R is the maximum ranks of correct buggy files for i_{th} bug report located by the bug localization technique, $ind(r)$ indicates whether the file located in rank r is correct buggy file ($ind(r) = 1$) or not ($ind(r) = 0$), $N_B(i)$ denotes the number of buggy files for the i_{th} bug report and $N_T(r)$ stands for the number of buggy files correctly located in Top r.

Competitors

In order to exhibit the importance of applying different strategies to extract features from bug reports and source files, we will first compare CNN_Forest with the following two models on the validation set:

- *CNN_CNN* uses the same strategy (CNN) to extract features from bug reports and source files.
- *Forest_Forest* applies the same ensemble of random forests to extract features of bug reports and source files.

All CNNs and random forests use same parameters, as well as the subsequent cascade forest.

CNN_Forest is then compared to the following four state-of-the-art techniques on the testing set:

- *NP-CNN* proposed by Huo et al. [16] applied CNN to learn unified features from natural and programming languages.
- *LR+WE* [49] enhanced their previously-proposed learning-to-rank model (LR) by adding new features obtained by word embedding (WE) techniques.
- *DNNLOC* proposed by Lam et al. [22] combined deep learning techniques with the information retrieval techniques to localize the buggy files for bug reports.

[2] http://eclipse.org/aspectj/
[3] http://projects.eclipse.org/projects/eclipse.platform.ui
[4] http://www.eclipse.org/jdt/
[5] http://www.eclipse.org/swt/
[6] http://tomcat.apache.org
[7] https://github.com/yanxiao6/BugLocalization-dataset

- *BugLocator* [52] measured the textural similarity between the texts in bug reports and source files using the revised Vector Space Model (rVSM).

Since the original implementations of the four models are not released, we tried our best to implement our own versions of these four models. We strictly followed the steps given by [16, 22, 49, 52], but all the detailed implementations might not be all the same. We used the same dataset to test their performance and got similar results. Thus, we are confident to argue that the comparisons are fair.

4.2 Results and Analyses

In this section, we report the results of CNN_Forest and competitors on the intrinsic and extrinsic evaluation, and analyze the results.

Intrinsic Evaluation

The intrinsic evaluation is conducted on the validation set to show the improvements achieved by CNN_Forest compared to CNN_CNN and Forest_Forest. The MAP and MRR values of these models are presented in Table 2 and the best MAP and MRR values for each project has been highlighted in bold. It can be observed from the results that CNN_Forest is able to achieve the best MAP and MRR except that Forest_Forest obtains 0.005 higher MAP than CNN_Forest on the Project Eclipse UI, which is not a big loss. The highest improvement on MAP achieved by CNN_Forest compared to the best result of the other two models is 0.023, that is about 5% relative improvement.

Table 2: Performance Comparison for Intrinsic Evaluation.

Project	Model	MAP	MRR
AspectJ	CNN_Forest	**0.458**	**0.563**
	CNN_CNN	0.449	0.560
	Forest_Forest	0.430	0.549
Eclipse UI	CNN_Forest	0.460	**0.590**
	CNN_CNN	0.441	0.561
	Forest_Forest	**0.465**	0.588
JDT	CNN_Forest	**0.448**	**0.517**
	CNN_CNN	**0.448**	0.502
	Forest_Forest	0.435	0.492
SWT	CNN_Forest	**0.462**	**0.530**
	CNN_CNN	0.415	0.507
	Forest_Forest	0.439	0.512
Tomcat	CNN_Forest	**0.627**	**0.681**
	CNN_CNN	0.623	0.669
	Forest_Forest	0.604	0.647

Thus, we can conclude that it is important to adopt different strategies to extract features from bug reports and source files. The different results also validate that the characteristics of the texts and code tokens in bug reports and source files are different even though they are both written by humans. Bug reports are written in natural languages and CNN performs well in semantic parsing in the field of natural language processing because of the convolving filters. Therefore, CNN_Forest performs better than Forest_Forest. On the other hand, source files are written in programming languages that contain rich and explicit structural information. The source code could be parsed into abstract syntax tree and also reverted. Moreover, tree-structure is good at extracting the structural information from source code [25, 29, 31, 44]. CNN_Forest thus has better performance than CNN_CNN.

Extrinsic Evaluation

The CNN_Forest is then compared to other four current state-of-the-art techniques (NP-CNN [16], LR+WE [49], DNNLOC [22], and BugLocator [52]) in the field of bug localization on the testing set. Their performances on five open-source projects are shown in Table 3. The best MAP and MRR results are also highlighted in bold. As we can observe from the table, CNN_Forest always performs better than the four competitors except for the MRR in Project JDT, but the loss is negligible. The improvements on MAP of CNN_Forest compared to the best results of the four competitors range from 0.003 to 0.034. Moreover, the improvements are more obvious when evaluating on the small dataset, i.e. Project AspectJ and Tomcat.

Table 3: Performance Comparison with the State-of-the-art Techniques.

Project	Model	MAP	MRR
AspectJ	CNN_Forest	**0.436**	**0.519**
	NP-CNN	0.402	0.487
	LR+WE	0.302	0.454
	DNNLOC	0.323	0.475
	BugLocator	0.278	0.364
Eclipse UI	CNN_Forest	**0.432**	**0.534**
	NP-CNN	0.429	0.529
	LR+WE	0.398	0.461
	DNNLOC	0.413	0.514
	BugLocator	0.332	0.383
JDT	CNN_Forest	**0.423**	0.514
	NP-CNN	0.405	0.463
	LR+WE	0.417	**0.516**
	DNNLOC	0.342	0.452
	BugLocator	0.290	0.367
SWT	CNN_Forest	**0.394**	**0.482**
	NP-CNN	0.371	0.466
	LR+WE	0.381	0.446
	DNNLOC	0.369	0.445
	BugLocator	0.269	0.312
Tomcat	CNN_Forest	**0.550**	**0.614**
	NP-CNN	0.529	0.585
	LR+WE	0.503	0.556
	DNNLOC	0.523	0.604
	BugLocator	0.425	0.481

NP-CNN applied CNN to extract features from both bug reports and source files. The performance is limited because the characteristics of the texts in bug reports and the code

tokens in source files are different just as we observed in the intrinsic evaluation. The structural information of the source files extracted by NP-CNN is subject to each code line, however, the programming syntactic information inside the program blocks is not extracted. Our proposed model CNN_Forest considers both the structural information and programming syntactic information benefiting from the ensemble of random forests with multi-grained scanning. Moreover, the texts and code tokens are processed by word embedding techniques before fed into the feature extraction part. The semantic information is therefore preserved, which is lost in NP-CNN without word embedding techniques. That's why CNN_Forest has better performance than NP-CNN. On the other hand, compared to CNN_Forest and NP-CNN, LR+WE and DNNLOC tried to add the semantic information into their models but did not take the program structure related information into consideration. Thus, their performances are worse than CNN_Forest. BugLocator was based on textural similarity and did not involve the semantic and structural features of bug reports and source files. Therefore, BugLocator's performance is much worse than the other techniques.

5 DISCUSSION

5.1 Why does CNN_Forest Work?

CNN_Forest tries to extract both semantic information and programming structural information from source files, which is different from the ones in bug reports written in natural languages. This kind of information is seldom considered by existing techniques.

The word embedding technique is first applied to convert the texts in bug reports and source files into word vectors. Similar words are close to each other in the vector space, which preserves the semantic information. Different strategies are then used to extract the features from bug reports and source files. Bug reports are written in natural languages where CNN has performed excellently because of convolving filters. On the other hand, source files are composed of code tokens that involve more stringent structural information. They can be parsed into abstract syntax tree and also be reverted from the abstract syntax tree. Thus, we leverage the ensemble of random forests involving many trees with multiple sliding windows to extract the programming structural information. Therefore, CNN_Forest not only preserves the semantics but also leverages the structural information of lexical terms in bug reports and the programming code tokens in source files. Finally, unlike the linear combination of the extracted features used in the literature [16, 48, 49], the cascade forest is used to further extract features from both bug reports and source files, and the correlated relationships between them are learned by the alternate cascade structure that is similar to the layer-structure in deep learning.

5.2 Threats to Validity

The experimental results demonstrate the feasibility and effectiveness of CNN_Forest, however, we do acknowledge that there are still some potential threats to validity of our study. Following the recommendations of Wohlin et al. [45], we discuss the threats to internal validity, construct validity and external validity of this study.

Internal Validity

The performance of our proposed model may be somewhat dependent on the performance of the word embedding techniques. We have checked the property of the adopted word embedding techniques before adopting them in our model. It is important to do so before implementing our model. Secondly, the hyper-parameters configuration set could introduce some bias in the experimental results. However, we set the parameters according to the suggestions of the existing studies, which enabled our choices to be reasonable. For instance, the size and number of the filters used in CNN and the size of the sliding windows used in the ensemble of random forests were set as the suggestions given in the literature [16, 46]. And in order to examine the intrinsic property of the forest structures, we used the default number of trees in each forest as recommended by Zhou et al. [54]. More fine-tunes might be needed for our model. We leave this for future studies.

Construct Validity

When conducting experiments, we split the dataset in each project into validation set (oldest bug reports), training set(older bug reports) and testing set (newest bug reports) similar to the strategy used in the literature [49]. However, the splitting percentage is different from that used in most machine learning techniques [5, 29, 35]. It would be interesting to examine the effects of different splitting percentages on our proposed model, which is worth to be explored in further studies. In this paper, we used two measures (MAP and MRR) that are widely used in most bug localization studies [21, 49, 52]. In future work, we will consider other evaluation metrics, such as Accuracy@k [48] and Area Under the receiver operator characteristic Curve (AUC) [24]. Besides, other validations (e.g., statistical significance and effect size of the results) will be considered in future when there are more project dataset used to evaluate the proposed model.

External Validity

We evaluated our model on five dataset from Java projects as many bug localization studies [21, 48, 49] did and tried to report the general results. However, the selection of only five projects may have potentially limited representativeness. On the other hand, the performance of our model on other projects written in other programming languages is still unknown. We intend to examine our model on more projects, especially the ones written in, for example, C++, in a future study.

6 RELATED WORK

6.1 Bug Localization Techniques

In the literature, several approaches have been proposed to localize buggy files automatically. Poshyvanyk et al. [36] applied Latent Semantic Indexing (LSI) to represent code and queries as vectors, and estimated the similarity between their vector representations using the cosine similarity. Lukins et al. [27] proposed a Latent Dirichlet Allocation (LDA)-based

approach that measured the similarity between the descriptions of bug reports and the topics of source files estimated by LDA. Gay et al. [12] transformed bug reports and source files into feature vectors based on Vector Space Model (VSM) and then measured the similarity between them. Zhou et al. [52] proposed BugLocator based on a revised VSM (rVSM) to measure the textural similarity between bug reports and source files. The document length and similar bugs that have been resolved before were added as new features into the VSM. These VSM-based techniques have been experimentally validated better than LSI and LDA-based techniques [2, 27]. Scanniello et al. [37, 38] and Zhang et al. [50, 51] used both textual information from source code and bug reports and syntactic information from source code for concept location. However, the aforementioned approaches were based on the textual representation of the bug reports and source files but inadvertently ignored the semantic information in them.

Topic-based models were used to extract the semantic information for many tasks in the field of software engineering [33, 47]. BugSout proposed by Nguyen et al. [32] was a topic-based model for bug localization. However, these approaches were not automatic because of the tuning process of the right number of topics. Ye et al. [49] applied word embedding techniques to convert the texts in bug reports and source files to word vectors, and then added the similarities between the vectors as new features into their previously-proposed learning-to-rank (LR) model [48] that was an adaptive ranking model. Lam et al. [21, 22] combined deep neural networks (DNNs) based on auto encoder with the information retrieval technique (rVSM) to improve the performance of localizing buggy files. They applied DNNs to combine the following three features: relevancy between bug reports and source files learned from another two DNNs, textual similarity collected by rVSM, and projects' metadata such as bug fixing history, code changes and so on. However, these approaches did not consider the structural information of the bug reports and source files.

Huo et al. [16] proposed NP-CNN, a convolutional neural network (CNN)-based model, to learn the unified features from bug reports and source files written in natural and programming languages. In their work, CNN was employed to extract features from both bug reports and source files. Different from their model, we adopted CNN with multiple filters and the ensemble of random forests with multi-grained scanning to respectively extract structural information from bug reports and source files. The experimental results validated that this different feature extraction strategies for bug reports and source files indeed make sense. Furthermore, we employed word embedding techniques to preserve the semantics of feature vectors before feeding them into the CNN and random forests, which was not included in NP-CNN.

6.2 Convolutional Neural Network

Convolutional neural network (CNN) has excellent performance in the field of image processing [20, 40] and natural language processing [8, 9]. Kim [19] conducted a series of experiments for sentence classification tasks using CNN with the pre-trained word vectors obtained by the word embedding technique and achieved excellent results on several benchmarks. Johnson et al. [18] studied CNN on text categorization (sentiment classification and topic classification) to make use of the internal structure of text data. Besides NP-CNN discussed previously for bug localization, Mou et al. [29] proposed a CNN over tree structures for programming language processing in the field of software engineering. This model contained vector representation layer, coding layer, tree-based convolutional layer whose convolution kernel was designed over programs' abstract syntax tree and dynamic pooling layer, as well as a fully-connected hidden layer to detect structural information from programs. It was superior in terms of program classification and code pattern detection.

6.3 Cascade Forest

The cascade forest is a cascade of ensembles of random forests that are the ensemble of decision trees. The ensemble methods combine multiple learners and are powerful on many tasks [53]. The cascade procedure is able to automatically determine the number of cascade levels and thus have a control on its model complexity, which is related to boosting procedure [10, 41, 43]. Zhou et al. [54] proposed deep forest based on the cascade structure for many tasks including image categorization, face recognition, sentence classification, and achieved comparable or even better results. They analyzed that this cascade structure behaved like the layer structure in deep learning. Inspired by their work, we adopted the cascade forest to further extract the structural information from bug reports and source files, and learn the correlated relationships between them.

7 CONCLUSIONS AND FUTURE WORKS

The existing techniques for bug localization did not highlight the differences between the bug reports written in natural languages and source files written in programming languages. And few of them considered both the semantic and structural information in bug reports and source files.

This paper proposed CNN_Forest, a new CNN and random forest-based model for bug localization. After pre-processing the texts in bug reports and source files, CNN_Forest first employs the word embedding technique to convert the words into vectors with the preservation of the semantic information in them. The CNN with multiple filters and the ensemble of random forests with multi-grained scanning are then used to extract both semantic and structural features from the word vectors of bug reports and source files. Finally, the cascade forest is used to further conclude features from the feature vectors extracted by CNN and the ensemble of random forests, and learn the correlated relationships between bug reports and source files benefiting from the alternate cascade structure. Our evaluation on five open-source projects showed that the separately learned features do improve the performance of the model. Moreover, CNN_Forest achieved

higher MAP (at most 0.034) and MRR (at most 0.032) than the best results of the four current state-of-the-art techniques (NP-CNN, LR+WE, DNNLOC, and BugLocator).

In the future, we would like to fine-tune our model to further improve its performance, such as more different kinds of random forests (e.g., extra-trees) to enhance diversity. Moreover, few existing studies focused on the projects written in other programming languages, e.g., C/C++. We intend to extend our model to these projects for bug localization and use more measures (Accuracy@k, AUC, and statistical significance) to evaluate their performances. In addition, it would be interesting to examine the performance of our model in other applications of software engineering, such as defect prediction.

ACKNOWLEDGMENTS

This work is supported in part by the GRF of the Research Grants Council of Hong Kong [No. 11208017], and the research funds of City University of Hong Kong [No. 9678149, 9440180, 7004683, 7004474].

REFERENCES

[1] Dave Binkley, Marcia Davis, Dawn Lawrie, and Christopher Morrell. 2009. To camelcase or under_score. In *Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on*. IEEE, 158–167.

[2] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of Machine Learning Research* 3, Jan (2003), 993–1022.

[3] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[4] Bernd Bruegge and Allen H Dutoit. 2004. *Object-oriented software engineering using UML, patterns and Java-(Required)*. Prentice Hall.

[5] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, Trang Pham, Aditya Ghose, and Tim Menzies. 2016. A deep learning model for estimating story points. *arXiv preprint arXiv:1609.00489* (2016).

[6] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. 2012. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 3642–3649.

[7] Christopher Clark and Amos Storkey. 2015. Training deep convolutional neural networks to play go. In *International Conference on Machine Learning*. 1766–1774.

[8] Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*. ACM, 160–167.

[9] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research* 12, Aug (2011), 2493–2537.

[10] Yoav Freund and Robert E Schapire. 1995. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*. Springer, 23–37.

[11] Kunihiko Fukushima and Sei Miyake. 1982. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and Cooperation in Neural Nets*. Springer, 267–285.

[12] Gregory Gay, Sonia Haiduc, Andrian Marcus, and Tim Menzies. 2009. On the use of relevance feedback in IR-based concept location. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*. IEEE, 351–360.

[13] Edward Grefenstette, Phil Blunsom, Nando de Freitas, and Karl Moritz Hermann. 2014. A deep architecture for semantic parsing. *arXiv preprint arXiv:1404.7296* (2014).

[14] Abram Hindle, Earl T Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. 2012. On the naturalness of software. In *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE, 837–847.

[15] David H Hubel and Torsten N Wiesel. 1962. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology* 160, 1 (1962), 106–154.

[16] Xuan Huo, Ming Li, and Zhi-Hua Zhou. 2016. Learning Unified Features from Natural and Programming Languages for Locating Buggy Source Code.. In *International Joint Conference on Artificial Intelligence (IJCAI)*. 1606–1612.

[17] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 2013. 3D convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence* 35, 1 (2013), 221–231.

[18] Rie Johnson and Tong Zhang. 2014. Effective use of word order for text categorization with convolutional neural networks. *arXiv preprint arXiv:1412.1058* (2014).

[19] Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* (2014).

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 1097–1105.

[21] An Ngoc Lam, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N Nguyen. 2015. Combining deep learning with information retrieval to localize buggy files for bug reports (n). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, 476–481.

[22] An Ngoc Lam, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N Nguyen. 2017. Bug localization with combination of deep learning and information retrieval. In *Proceedings of the 25th International Conference on Program Comprehension*. IEEE Press, 218–229.

[23] Thomas D LaToza and Brad A Myers. 2010. Hard-to-answer questions about code. In *Evaluation and Usability of Programming Languages and Tools*. ACM, 8.

[24] Stefan Lessmann, Bart Baesens, Christophe Mues, and Swantje Pietsch. 2008. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering* 34, 4 (2008), 485–496.

[25] Ziyi Lin, Hao Zhong, Yuting Chen, and Jianjun Zhao. 2016. LockPeeker: detecting latent locks in Java APIs. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ACM, 368–378.

[26] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE, 413–422.

[27] Stacy K Lukins, Nicholas A Kraft, and Letha H Etzkorn. 2010. Bug localization using latent dirichlet allocation. *Information and Software Technology* 52, 9 (2010), 972–990.

[28] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[29] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. 2016. Convolutional Neural Networks over Tree Structures for Programming Language Processing.. In *AAAI*. 1287–1293.

[30] Nasser M Nasrabadi. 2007. Pattern recognition and machine learning. *Journal of electronic imaging* 16, 4 (2007), 049901.

[31] Anh Tuan Nguyen and Tien N Nguyen. 2015. Graph-based statistical language model for code. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, Vol. 1. IEEE, 858–868.

[32] Anh Tuan Nguyen, Tung Thanh Nguyen, Jafar Al-Kofahi, Hung Viet Nguyen, and Tien N Nguyen. 2011. A topic-based approach for narrowing the search space of buggy files from a bug report. In *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*. IEEE, 263–272.

[33] Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N Nguyen, David Lo, and Chengnian Sun. 2012. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 70–79.

[34] John F Pane, Brad A Myers, et al. 2001. Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human-Computer Studies* 54, 2 (2001), 237–264.

[35] Hao Peng, Lili Mou, Ge Li, Yuxuan Liu, Lu Zhang, and Zhi Jin. 2015. Building program vector representations for deep learning.

In *International Conference on Knowledge Science, Engineering and Management*. Springer, 547–553.

[36] Denys Poshyvanyk, Yann-Gael Gueheneuc, Andrian Marcus, Giuliano Antoniol, and Vaclav Rajlich. 2007. Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. *IEEE Transactions on Software Engineering* 33, 6 (2007).

[37] Giuseppe Scanniello and Andrian Marcus. 2011. Clustering support for static concept location in source code. In *Program Comprehension (ICPC), 2011 IEEE 19th International Conference on*. Ieee, 1–10.

[38] Giuseppe Scanniello, Andrian Marcus, and Daniele Pascale. 2015. Link analysis algorithms for static concept location: an empirical assessment. *Empirical Software Engineering* 20, 6 (2015), 1666–1720.

[39] Bird Steven, Ewan Klein, and Edward Loper. 2009. Natural language processing with python. *OReilly Media Inc* (2009).

[40] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.

[41] Paul Viola and Michael Jones. 2001. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, Vol. 1. IEEE, I–I.

[42] Izhar Wallach, Michael Dzamba, and Abraham Heifets. 2015. AtomNet: a deep convolutional neural network for bioactivity prediction in structure-based drug discovery. *arXiv preprint arXiv:1510.02855* (2015).

[43] Geoffrey I Webb. 2000. Multiboosting: A technique for combining boosting and wagging. *Machine learning* 40, 2 (2000), 159–196.

[44] Michael W Whalen, Suzette Person, Neha Rungta, Matt Staats, and Daniela Grijincu. 2015. A flexible and non-intrusive approach for computing complex structural coverage metrics. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE Press, 506–516.

[45] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.

[46] Yan Xiao, Jacky Keung, Qing Mi, and Kwabena E Bennin. 2017. Improving Bug Localization with an Enhanced Convolutional Neural Network. In *Asia-Pacific Software Engineering Conference (APSEC), 2017 24th*. IEEE, 338–347.

[47] Xihao Xie, Wen Zhang, Ye Yang, and Qing Wang. 2012. Dretom: Developer recommendation based on topic models for bug resolution. In *Proceedings of the 8th international conference on predictive models in software engineering*. ACM, 19–28.

[48] Xin Ye, Razvan Bunescu, and Chang Liu. 2014. Learning to rank relevant files for bug reports using domain knowledge. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 689–699.

[49] Xin Ye, Hui Shen, Xiao Ma, Razvan Bunescu, and Chang Liu. 2016. From word embeddings to document similarities for improved information retrieval in software engineering. In *Proceedings of the 38th International Conference on Software Engineering*. ACM, 404–415.

[50] Yun Zhang, David Lo, Xin Xia, Tien-Duy B Le, Giuseppe Scanniello, and Jianling Sun. 2016. Inferring links between concerns and methods with multi-abstraction vector space model. In *Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on*. IEEE, 110–121.

[51] Yun Zhang, David Lo, Xin Xia, Giuseppe Scanniello, Tien-Duy B Le, and Jianling Sun. 2017. Fusing multi-abstraction vector space models for concern localization. *Empirical Software Engineering* (2017), 1–44.

[52] Jian Zhou, Hongyu Zhang, and David Lo. 2012. Where should the bugs be fixed?-more accurate information retrieval-based bug localization based on bug reports. In *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 14–24.

[53] Zhi-Hua Zhou. 2012. *Ensemble methods: foundations and algorithms*. CRC press.

[54] Zhi-Hua Zhou and Ji Feng. 2017. Deep forest: Towards an alternative to deep neural networks. *arXiv preprint arXiv:1702.08835* (2017).