IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 29, 2020, accepted November 15, 2020, date of publication November 24, 2020,
date of current version December 9, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3040065

Applying Convolutional Neural Networks With Different Word Representation Techniques to Recommend Bug Fixers

SYED FARHAN ALAM ZAIDI[1], FARAZ MALIK AWAN[2], MINSOO LEE[1], HONGUK WOO[3], AND CHAN-GUN LEE[2]

[1]CAU Institute of Innovative Talent of Big Data, Department of Computer Science and Engineering, Chung-Ang University, Seoul 06974, South Korea
[2]Department of Computer Science and Engineering, Chung-Ang University, Seoul 06974, South Korea
[3]Department of Software, Sungkyunkwan University, Suwon 16419, South Korea

Corresponding author: Chan-Gun Lee (cglee@cau.ac.kr)

This work was supported by the National Research Foundation of Korea(NRF) grant (NRF-2017R1E1A1A01075803), a grant from the Korea Atomic Energy Research Institute, "Development and Operation of Information and Communications Technology (ICT) based Nuclear Energy Safety Validation System (524320-18)" funded by Ministry of Science and ICT, and the ICT Creative Consilience program under Grant IITP-2020-0-01821 supervised by the IITP (Institute for Information & communications Technology Planning & Evaluation).

ABSTRACT Bug triage processes are intended to assign bug reports to appropriate developers effectively, but they typically become bottlenecks in the development process—especially for large-scale software projects. Recently, several machine learning approaches, including deep learning-based approaches, have been proposed to recommend an appropriate developer automatically by learning past assignment patterns. In this paper, we propose a deep learning-based bug triage technique using a convolutional neural network (CNN) with three different word representation techniques: Word to Vector (Word2Vec), Global Vector (GloVe), and Embeddings from Language Models (ELMo). Experiments were performed on datasets from well-known large-scale open-source projects, such as Eclipse and Mozilla, and top-k accuracy was measured as an evaluation metric. The experimental results suggest that the ELMo-based CNN approach performs best for the bug triage problem. GloVe-based CNN slightly outperforms Word2Vec-based CNN in many cases. Word2Vec-based CNN outperforms GloVe-based CNN when the number of samples per class in the dataset is high enough.

INDEX TERMS Bug triage, CNN, GloVe, Word2Vec, ELMo, word representation, word embedding, bug report, bug fixing, recommending bug fixer, deep learning.

I. INTRODUCTION

The process of finding and assigning an appropriate developer for a given bug is referred to as "bug triage." When a bug is found, it is typically documented in a bug report containing information about the bug. The bug report is then assigned to a developer who investigates and fixes the related bug. A triager typically references histories of fixed bug reports and their fixers (developers) to choose an appropriate developer who has fixed similar bugs.

Bug triage is challenging in today's large-scale software projects where many bug reports are issued daily. Choosing the appropriate developer is complicated because there are many developers with diverse skills. For example, more than 333,000 bugs were reported in the Eclipse project, with approximately 99 bugs daily from October 2001 to December 2010 [1]. Identifying an appropriate developer can be a time-consuming and challenging task for human triagers. In many cases, manual bug-triaging can be error-prone due to a lack of knowledge among developers [2]. Many software engineering studies have investigated the bug triage problem. Automated techniques for bug triaging that exploit the knowledge from large sets of fixed bugs stored in public repositories have gained attention in both industry and academia. Many large-scale open-source projects, such as Mozilla, Eclipse, and Google, maintain a history of fixed bugs and can be used to develop automated bug triage systems. Maintaining this history can become a bottleneck of the development process, especially for large-scale software projects such as Eclipse and Mozilla, because it requires manual labor by triagers and a large number of daily bug reports.

The associate editor coordinating the review of this manuscript and approving it for publication was Mouloud Denai.

A bug report contains an identifier, status (fixed or not), fixer (the developer who fixed the bug), type, title or summary, detailed description, reporter name, and report time [3]. The fixer information is not entered until the bug is fixed. Among the information in the bug report, previous studies commonly use the summary and description of the bug report. In contrast, other studies also include more sophisticated information, such as developer community network and expertise scores, by analyzing various data recorded in bug tracking systems.

Many previous studies have been conducted to propose automated bug triage systems that recommend a list of developers appropriate for a given bug report. We classify these studies into two categories. In the first category, we summarize the studies that did not use machine learning methods along with minimal or no Natural Language Processing (NLP) techniques. In the second category, we briefly describe the trends of the studies that rely on various machine learning techniques. The literature in this category is discussed in Section II.

The studies in the first category are not based on machine learning and use minimal or no NLP; however, they demonstrated comparable performance. Yadav *et al.* proposed an approach that recommends the fixers for bug reports by identifying similarity among the bug reports and calculating the developer expertise score [4]. Another technique extracts the keywords from tags and triage based on the social expertise of the developers [5]. Peng *et al.* proposed to use a search technique exploiting inverted indexed terms from different topics for the bug triage problem [6]. Recently, Kumari *et al.* proposed to use a bug dependency-based mathematical model by interpreting a bug's summary and description in terms of entropy to develop software reliability growth models [7].

For the second category, various machine learning techniques, including deep learning, have been adopted. They often use NLP techniques spanning from simple tf-idf to word embedding methods to process metadata fields, categorical attributes, and text fields (e.g., summary, description, and comments) in the bug reports. Section II presents the latest literature. Classical machine learning techniques, such as Naïve Bayes, Support Vector Machine (SVM), and k-nearest neighbor, require hand-crafted features to train the classifier. However, deep learning techniques can learn more diversified features automatically.

In this paper, we restrict our discussion to the deep learning-based approaches and assess new opportunities in this direction. Recent approaches based on neural networks often exploit word-embedding techniques to convert words and sentences into vector forms. Word to Vector (Word2Vec), Global Vector (GloVe), Embeddings from Language Models (ELMo), and Bidirectional Encoder Representations from Transformers (BERT) are word-embedding techniques frequently adopted in recent NLP-based approaches. As suggested by a recent study by Stein *et al.*, a selection among different word embedding techniques such as Word2Vec, GloVe, and fastText may significantly impact the performance of a text classification task [8]. Unfortunately, there has been little effort to measure such impacts on the bug triage problem.

This paper studies the performance of a convolutional neural network (CNN)- based bug triage system on different word-embedding techniques and presents an analysis of the experimental results. The adopted CNN model is derived from our previous work [9]. We consider three embedding techniques: two context-insensitive (Word2Vec and GloVe) and one context-sensitive (ELMo).

The intent of our paper is not to argue that deep learning is superior to traditional machine learning in an automated system for the bug triage task. Instead, we investigate the performance impact based on the chosen word-embedding technique when the system is built on a CNN model.

To the best of our knowledge, our study is the first to address the effects of different word-embedding techniques for the bug triage problem. Furthermore, we review recent efforts in automated bug triage using deep learning, such as CNNs and recurrent neural networks (RNNs), and report the comparison results for their performance on the same dataset.

The main contributions of our paper are as follows:

- We compare three word-embedding techniques in the context of the bug triage problem: Word2Vec, GloVe, and ELMo.
- We compare our implementation with others presented in recent deep learning-based studies, such as Deep Triage [3], CNN-based approach [9], [10] and ELM based bug assignment method [11].

The rest of the paper is organized as follows. Section II presents the literature review and our motivation for improving the task of bug triage. Section III explains the preprocessing technique and the proposed methodology for bug triage. Section IV presents the data collection sources, evaluation metrics, experimental results, comparisons, and evaluation of the proposed methods. Threats to validity are discussed in Section V. Some limitation are discussed in Section VI Finally, we conclude our findings and briefly state potential future directions in Section VII.

II. RELATED STUDIES AND MOTIVATION

This section provides a brief review of recent studies and the motivation of our study, assessing the impacts on bug triage of different embedding techniques.

Anvik *et al.*'s seminal research on automated bug triage adopted supervised-learning algorithms and proposed a semi-automated system to recommend developers for a given bug report [12]. Ahsan *et al.* presented a comparison between information retrieval and machine-learning methods, such as SVM, to acquire effective compositional recommendation methods for the bug triage problem [13]. Wu *et al.* proposed a k-nearest neighbor based triage system with an expertise ranking-based approach [14]. Zhou *et al.* introduced an information retrieval-based approach and proposed BugLocator, which ranks bug reports and source files based on text similarity [15]. Shokripour *et al.* proposed an information

TABLE 1. Literature review of recent studies related to bug triage.

Author	Year	Bug Information	Word representation and feature extractor	Technique	Dataset	top-1 Accuracy %	top-5 Accuracy %	top-10 Accuracy %
Yang et al. [20]	2014	Summary, description, product, component, priority, Status	Core-NLP and TMT	Core NLP used for tokenizing words and TMT used for finding similar topic words. The k-nearest neighbors is used for the classification task	Eclipse Mozilla NetBeans	63 58 59	- - -	76 81 81
Xuan et al. [21]	2015	Summary and description	Tf-idf	Naïve Bayes used for the classification task	Mozilla Eclipse	- -	46.46 60.40	- -
Badashian et al. [5]	2015	Summary, description, keywords, project language, stackoverflow and github's tags	-	Extracted keywords from tags and Triage by social expertise with matching keywords	20 Git Hub Projects	-	89.43	-
Dedik et al. [22]	2016	Summary and description	Tf-idf	Tf-idf used for feature extraction and Support Vector Machine (SVM) used for classification task	Firefox OSS Industrial Data	59 54	- -	97 90
Jonsson et al. [23]	2016	Summary and description	Tf-idf	Used tf-idf for feature extraction with stacked generalization of a classifier ensemble	Industrial	89	-	-
Xuan et al. [24]	2017	Summary, description	Tf-idf	Tokenized as list of words and used Naïve Bayes Classifier and expectation-maximization	Eclipse	21.04	48.07	-
Peng et al. [6]	2017	Summary and description	-	Used inverted indexing to sort list of terms from bug reports and relevant search technique for developer recommendation	Mozilla Eclipse	22.3 30.7	51.8 61.3	- -
Lee et al. [9]	2017	Summary and description	Word2Vec	Used Word2Vec for Word representation and CNN based classifier for assigning developers	Eclipse (JDT) Eclipse (Platform) Firefox	46.6 36.1 27.1	74.3 56.7 50.5	- - -
Yin et al. [11]	2018	Summary, description and comments of developer	Tf-idf	Used tf-idf weighting for constructing feature space and Genetic Algorithm based optimal Extreme Learning Machine (ELM) for classification tasks with diversified features	Bugzilla Eclipse NetBeans GCC	62.4 65.8 71.2 63.3	- - - -	- - - -
Mani et al. [3]	2019	Summary and description	Word2Vec	Proposed Bi-directional RNN with Attention mechanism and Word2Vec for conversion in vector	Mozilla Firefox Mozilla Core Google Chrome	- - -	- - -	46.6 ± 6.4 38.8 ± 3.2 42.7 ± 3.5
Guo et al. [10]	2020	Summary and description	Word2Vec	Developer Activity based CNN	Eclipse Mozilla NetBeans	25.20 12.44 35.54	46.03 26.93 62.66	53.37 34.46 74.89

retrieval-based approach that uses commit messages to recommend appropriate developers [16]. Later, a follow-up study by Shokripour *et al.* used the title, description, and source code information from the bug report for the bug triage task, adopted weighted unigram noun terms as features, and used the bug location and developer expertise for triage [17]. Alenezi *et al.* performed bug triage processes using text mining [18] and used the title of the bug reports to train the Naïve Bayes classifier. They also used four term-selection methods to reduce the high dimensionality of the term space: log odds ratio, chi-square, term frequency relevance frequency, distinguishing feature selector, and mutual information. Zimmermann *et al.* used open-source repositories such as GitHub and Bugzilla [19] to study the impact of switching bug-detection tools for medium-sized projects. In Table 1,literature related to bug triage published in the last five years is reviewed and presented in chronological order.

Yang *et al.* [20] used the Stanford Topic Modeling Toolbox (TMT) to analyze datasets and exploit topic similarity for bug triage. Xuan *et al.* [21] used tf-idf to extract features and the Naïve Bayes classifier for classification. Badashian *et al.* [5] used the title, description, keywords, project language, and Stack Overflow as features and matching keywords for triage. Dedik *et al.* [22] used tf-idf for feature extraction and SVM for the classification task. Jonssons *et al.* [23] used tf-idf for feature extraction and stacked generalization for classifier ensembles. Xuan *et al.* [24] used tf-idf to tokenize and extract features and the Naïve Bayes algorithm with

expectation-maximization for classification. Peng *et al.* [16] used inverted indexing to sort the terms extracted from the summary and descriptions and applied search techniques to recommend developers.

Following the deep learning trend, several studies have also used deep learning-based techniques for the bug triage problem. Lee *et al.* [9] proposed a CNN-based bug triage approach using Word2Vec, which provides pre-trained word vectors for NLP. They observed the performance differences for both industrial and open-source projects. They attributed higher accuracy to the controlled quality of the bug reports and characteristics such as stable and smaller developer pools for industrial projects. Yin *et al.* [11] used tf-idf to extract features and apply a genetic algorithm-based optimized Extreme Learning Machine (ELM) for bug triage. The ELM is a type of feed-forward neural network and does not use the back-propagation algorithm [25]. Mani *et al.* [3] proposed a bi-directional recurrent neural network-based technique for automatic bug triage. They used an attention mechanism that learns the syntactic and semantic features from long word sequences. They used Word2Vec for word representation. Their experimental results demonstrated that deep learning-based approaches are superior in performance compared to classical machine-learning-based approaches.

Recently, Guo *et al.* [10] proposed a developer activity-based convolutional neural network (CNN-DA) method for bug triage that recommends a list of developers. They used Word2Vec with 200 embedding dimensions for word representation and applied word segmentation, stop word removal and stemming technique in the preprocessing step. Their method was validated on three large datasets; Mozilla, Eclipse, and Netbeans. They compared CNN-DA with One-hot CNN [26]. They used summary and description from bug reports for training the network.

In what follows, we summarize several recent studies on text classification because their approaches are often similar to those of the studies on bug triage systems. Kapočiūtė-Dzikienė *et al.* applied traditional machine learning and deep learning methods to the sentiment analysis of Lithuanian texts. Their analysis results demonstrated the superior performance of traditional techniques over deep learning techniques, although the performance gap was not significant. Furthermore, deep learning methods were useful for small datasets [27]. Jang *et al.* implemented a Word2Vec-based CNN to classify news articles and tweets. Their experiments compared the performance of two implementation variants of Word2Vec, continuous bag-of-words (CBOW), and skip-gram. The CBOW model exhibited higher accuracy for news articles, but the skip-gram model outperformed the CBOW model for tweets [28]. Stein *et al.* studied hierarchical text classification tasks and assessed the effectiveness of different strategies—flat or hierarchical—for modeling the category information. Furthermore, they presented the impact analysis of various word-embedding techniques such as Word2Vec, GloVe, and fastText on the hierarchical text classification [8].

Arguably, one of the most important achievements made in the deep learning-based NLP area is a mechanism for representing textual words into dense vector space models, referred to as a word-embedding technique. Young *et al.* [29] states that the deep neural networks based on various word-embedding techniques have demonstrated superior results on various NLP tasks because they enable multi-level automatic feature-representation learning. However, traditional machine learning-based NLP approaches depend heavily on hand-crafted features, which are time-consuming and often incomplete.

Typical word-embedding techniques frequently adopted in recent NLP-based approaches include Word2Vec, GloVe, and ELMo. The former two approaches are context-insensitive, and the latter is context-sensitive. Each word is always mapped to a specific single vector. However, the word can take on different meanings in different situations with context-insensitive embedding techniques (which do not consider the context). In contrast, the context-sensitive word-embedding techniques may generate different vectors for the same word in different contexts. One of the evident limitations in context-insensitive embedding techniques is that a polysemous word is forced to share the same representation, which could pose various disadvantages for applications using such embedding techniques.

ELMo [30] is one of the most popular context-sensitive word-embedding approaches. It generates contextualized representations of a word by concatenating the internal states of a two-layer bidirectional long short-term memory (BiLSTM) language model. Due to the advantage of context-sensitive representations, ELMo has been applied successfully to many NLP problems and demonstrated performance improvements. For example, ELMo has demonstrated superb performance in concept extraction [31], discourse relation recognition [32], and named entity recognition [33]. Qi *et al.* [34] used ELMo for bi-directional semantic matching of Chinese sentences.

In recent years, deep learning-based approaches have become popular and often adopt word-embedding techniques. However, to the best of our knowledge, the impact of different word-embedding techniques has not been studied in the context of the bug triage problem despite their importance. In the next section, we present the experimental setup designed to measure the performance of a CNN-based bug triage system on three word-embedding techniques: Word2Vec, GloVe, and ELMo.

III. METHODOLOGY

This section introduces the framework and training process of the proposed bug triage system. The general structure of the bug triage system is depicted in Figure 1. The data is passed through the preprocessing phase. Then, the processed data is converted into word vectors using the word-embedding technique. These word vectors are the input for the CNN network. After training on the given word vectors,

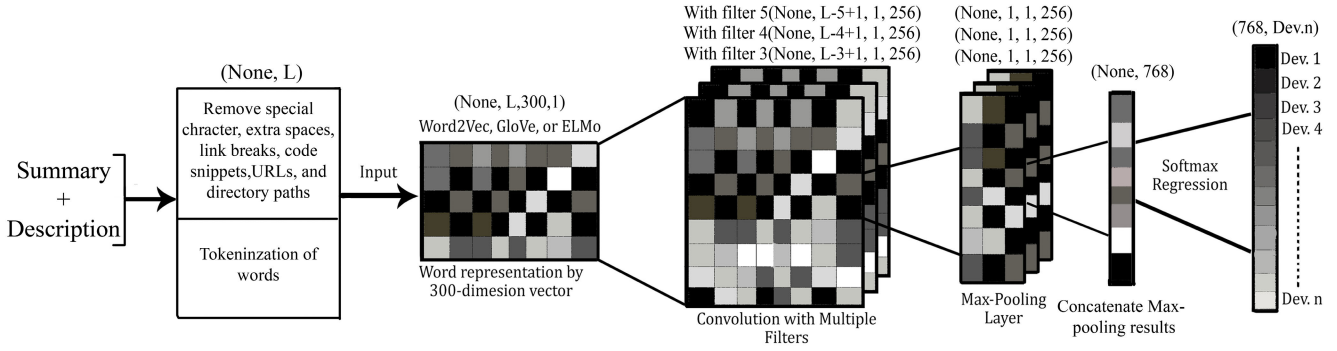


FIGURE 1. CNN Architecture with word representation for the automated bug triage system. The tensor shapes of each layer are explained on the top of the layer, where L denotes the sentence's sequence length/maximum length. The tensor shape on convolution and pooling layers shows the structure for filters 3, 4, and 5. The Dev.n indicates the number of total developers or class. Each convolution and pooling layer has 256 filters. Also, we use 512 number of filters to perform some experiments in this work.

the CNN model makes predictions to recommend a ranked list of appropriate developers. The details are discussed in the following subsections.

A. PREPROCESSING

Preprocessing is required to train NLP applications effectively. A bug triage system using a large set of bug reports from different open-source projects is proposed. The title and description are used for input data, with other attributes filtered out. Then, special characters, extra spaces, line breaks, code snippets, URLs, and directory paths are removed in the preprocessing phase. Furthermore, the preprocessed summary and description are converted into tokenized words to create the input vectors.

B. CNN MODEL WITH WORD REPRESENTATION

The proposed CNN technique consists of the word representation by vector, convolutional layer, pooling layer, and softmax regression layer.

1) WORD REPRESENTATION BY VECTOR

The word representation layer is the first layer that takes the preprocessed summary and description as inputs and converts them into vector forms. The layer adopts word-embedding techniques for the conversion step, and the converted vectors are supplied to a convolution layer designed to learn the features effectively. The shape of the input vector is set to the maximum length of the sentence. In Figure 1, L denotes the sequence length (or maximum length) of the sentence. The adopted embedding technique converts the input vector to a 300-dimension word representation form. The resultant vector of the layer has the shape $(L, 300)$. Three types of word-embedding techniques— Word2Vec, GloVe, and ELMo—are used in this research and introduced in the next section.

a: Word2Vec WORD EMBEDDING

Word2Vec converts pre-processed data into vector representations. Each word of a bug report is converted using pre-trained Word2Vec[1] generated from Google News datasets with approximately 100 billion words. The pre-trained model has 300-dimensional vectors for 3 million words and phrases. Each row in the input matrix of the dataset corresponds to a single word. Thus, training data are organized into rows of dimensions and columns of words in a bug report. The length of the rows is 300, which is consistent with our settings for the Word2Vec-based vectors. The length of the columns matches the number of words in the bug reports.

b: GloVe WORD EMBEDDING

In contrast to Word2Vec, GloVe focuses on the ratio of co-occurrence probabilities instead of the co-occurrence probabilities themselves. A harmonic function is used while weighting contexts in GloVe's implementation: if a context word is three tokens away, this context word will be counted as one-third of an occurrence. In contrast, the weighting of a contextual word is calculated by dividing the distance from the focus word by the window size in Word2Vec. In GloVe, the ratio of probabilities offers the information. This information is then encoded as vector differences. A weighted least-squares objective J (cost function) that attempts to minimize the difference between the dot product of two vectors has been proposed.

$$J = \sum_{i,j=1}^{v} f(X_{ij})(w_i^t w_j' + b_i + b_j' - \log X_{ij})^2 \qquad (1)$$

In the above equation, X is the word-word co-occurrence count matrix. where X_{ij} illustrates the number of times the word j occurs in the context of word i. w_i and b_i are the word vector and bias of word i. Similarly, w_j' and b_j' are the context word vector and bias of word j. A weighting function f assigns relatively lower weights to rare and frequent co-occurrences. GloVe takes the word-context co-occurrence matrix instead of the whole corpus because

[1]The pre-trained Word2Vec vectors are publicly available on: https://code.google.com/archive/p/Word2Vec/GoogleNews-vectors-negative300.bin.gz

the co-occurrence counts can be encoded in the word-context co-occurrence matrix.

In this word representation layer, each word of a bug report is converted using pre-trained GloVe [2] vectors. These pre-trained vectors are a word-word co-occurrence file that contains 840 billion tokens, 2.2 million vocabulary words, and 300-dimensional vectors. Similar to Word2Vec, each row in the input matrix of the data sets corresponds to a single word.

ELMo word embedding: ELMo is a context-sensitive technique that solves two challenging tasks of learning representations. The first challenge is representing the complex characteristics of a word: syntax and semantics. The second challenge is model polysemy. The vectors are derived from biLSTM, and the biLSTM is trained with a coupled language model objective on a large corpus. ELMo word representation is more in-depth than other contextualized techniques because it is the function of all internal bidirectional language models (biLMs). The biLM layer has two language models: a forward language model and a backward language model. The biLM is implemented using LSTM memory cells and calculates the probability of the sequence by modeling the probability of a token with the context in both directions (forward and backward) [30]. The pre-trained model of ELMo[3] model is used in this study. The embedding has trainable parameters in which module exposes four trainable scalar weights for layer aggregation.

2) CONVOLUTION LAYER

Let $z \in \mathbb{R}^M$ be a vector with M dimensions corresponding to a bug report. Each element in the vector is also a vector generated by Word2Vec, GloVe, or ELMo embedding. The convolution layer performs a convolution of input matrix z with convolutional filters c^k, with a different output computed for each filter. Three different filters with kernel sizes $k = 3, k = 4, k = 5$ are used to extract features of different lengths from the input vector.

For each feature window size, the N filters or neurons are used to learn complementary features. A convolution operation with a filter c^k is applied to the z word vector to generate a feature map F with stride (S) 1. Zero padding (P) is used where necessary because the word vector dimensions are fixed at 300. The feature map $F = \{f_1, f_2, f_3, \ldots f_l \ldots, f_{(n-k+1)}\}$ is extracted by applying the convolution operation on n-length data. In F, f_l represents the lth feature.

The convolution layer uses the embedded word vectors with tensor shape $(None, L, 30, 1)$ as the input. L is the maximum length of the sentence or sequence. The shape of each of the three convolution filters is (height of the filter, width of the filter, in channels, out channels). The three types of convolution filters are used with heights of 3, 4, and 5 and widths of 300. There are 1 in-channel and 256 out-channels because of the 256 filters/neurons in each layer. For some experiments, we used 512 inputs. After sliding the convolution filter with stride 1, the tensor of the shape $(None, L - Filter_{height}, 1, 256)$ was obtained. The height is the height of the convolution filter, which can be 3, 4, or 5. These details are illustrated in Figure 1.

Back-propagation is applied to calculate the gradient, which is needed to determine the weights used for training the neural network. The Rectified Linear Unit (ReLU) is used as a nonlinear activation function to compute the feature map from the convolution layer. The ReLU is defined as: $f(x) = \max(0, x)$. This function is zero for all negative values and grows linearly for positive values [35], [36]. There are many other functions used, such as binary step, linear, and sigmoid functions. Linear functions and binary step functions are both linear functions. Although the sigmoid function is nonlinear, the ReLU function (Which is also nonlinear) is generally preferred because the sigmoid function suffers from the problem of a vanishing gradient. We also adopted dropout to avoid overfitting.

3) POOLING LAYER

The pooling layer is used to sub-sample the features from the feature map F. Three types of pooling techniques are available: min-pooling, max-pooling, and average pooling. This study uses the max-pooling function to select the maximum value from F because it is widely adopted in the literature and illustrates high performance. Kernel k is also applied on the feature map with a Stride (S) of 1. Multiple filters with varied sizes provide results with a diverse F. As described in [9], when the number of filters is h, the F_{pool} is calculated using the max-pooling function with the following equation:

$$F_{pool} = \max_{1 \le j \le h} F_j^k \tag{2}$$

Here, k is the kernel size, and j is the index for the number of filters.

Similar to the convolution layer, the max-pooling layer uses three different filters or kernels to sub-sample the feature map. The size of the kernel is $(1, L - Filter_{height} + 1, 1, 1)$, and the filter height can be 3, 4, or 5. These three kernels apply to the outputs of the convolution layer with a stride of 1. The resultant tensor with shape $(None, 1, 1, 256)$ is obtained, which includes the 256 out-channels.

4) SOFTMAX-REGRESSION LAYER

The softmax regression layer concatenates all of the sub-sampled features F_{pool} for each kernel size. Softmax regression is used as an activation function and calculates the assignment probability of all developers for a given bug report. Softmax can be defined by the Bayes' Theorem [37], and its equation is as follows:

$$P(C_k|x) = \frac{P(x|C_k)P(C_k)}{\sum_{j=1}^{n} P(x|C_j)(P(C_j)} \tag{3}$$

[2] http://nlp.stanford.edu/data/wordvecs/GloVe.840B.300d.zip

[3] The trained model is available at https://tfhub.dev/google/ELMo/2. These are the embedding from the language model trained on 1 Billion words benchmark.

C_k is the selected developer class, and C_j is the jth developer class. P is the probability.

All outputs of the max-pooling layers are concatenated and produce the tensor of shape (None,768) for the 256 filters for each filter size. The Softmax classifier completes the training and produces the output of shape (768, number of classes). The number of classes depends on the number of fixers in the datasets. The classifier's results are the probability score of each class. The developer that has the maximum probability value is selected as the first-ranked developer.

Overfitting is a significant challenge while working on a neural network. Overfitting customizes the weights of neural networks on training data very tightly [38]. If an overfitted model is exposed to unseen data, its accuracy can be significantly worsened compared to the training accuracy. Therefore, the following techniques are used to avoid overfitting and improve performance:

Dropout: Dropout randomly drops some units (neurons) during the training process to prevent the many co-adaptions [39]. Therefore, neural networks forget specific learned weights during training and prevent the model from becoming over-trained or over-fitted. The value of dropout can be a real number between 0 and 1.

l2_regularization: A penalty is applied to the outliers to prevent the model from being distorted using l2_normalization. Outliers increase the mean error. Therefore, l2 loss with l2 regularization is used to foist the outliers. The l2 loss with regularization lambda λ is applied to all model parameters. These parameters are then combined with the softmax cross-entropy with logits function to calculate the cost of the model.

Xavier Initializer: Initialization is essential to achieving convergence. Xavier initialization keeps the scale of gradient the same for all layers in the network using uniform a distribution, which maintains activation variance and back-propagated gradients at controlled levels [40], [41].

$$Weights \sim U\left[-\frac{\sqrt{6}}{\sqrt{w_t + w_{t+1}}}, \frac{\sqrt{6}}{\sqrt{w_t + w_{t+1}}}\right] \quad (4)$$

In the above equation, U is the normal distribution, w_t is the tensor weight of an input layer, and w_{t+1} is the tensor weight of the output layer.

C. TRAINING THE CNN

The Adam optimizer is used to train the CNN. The Adam optimizer controls the learning rate using the Kingma and Ba's Adam Algorithm [42]. The Adam optimizer uses momentum (the average of the parameters), which enables a larger step size during training and converges to the step size without fine-tuning. It also removes noise and oscillation using momentum [43].

The dynamic learning rate is computed during training, starting with a high learning rate. The minimum learning rate is set to 0.0001, and the maximum learning rate is set to 0.0050. The learning rate is computed for the training of each batch using the following equation:

$$\rho = \rho_{min} + (\rho_{max} - \rho_{min}) \times e^{\frac{-s}{d}} \quad (5)$$

In the above equation, ρ is the learning rate, where ρ_{min} and ρ_{max} represent the minimum and maximum learning rate, and s is the total step count. The decay d is calculated by the decay coefficient and the ratio between the total number of training examples and batch size. Therefore, $d = decay \times ((\#trainingdata)/(batch - size))$. The model is trained on 20 epochs with 0.5 dropout probability and dynamic learning rate ρ. Different batch sizes and numbers of filters are used in training to identify superior results. The hyper-parameters are depicted in Table 2. The results are compared with other studies in the next section.

TABLE 2. Hyper-parameters for training CNN models.

Parameter	Value	Parameter	Value
Epochs	20	Batch size	32 or 64
Min. learning rate ρ_{min}	0.0001	Decay coefficient	2.5
Max. learning rate ρ_{max}	0.005	Dropout	0.5
l2 regularization λ	0.1,0.2	Embedding Dimension	300
Filter size	3,4,5	Number of Filters	256 or 512

The proposed technique is implemented using the Python scripting language with the Keras library and TensorFlow.

IV. EVALUATION AND RESULTS

This section evaluates the performance of the proposed method and addresses the following research questions:

- Which is more effective for word representation in CNN-based bug triage, and does it make any difference if a different word embedding is used?
- Which method is best to achieve superior top-1 accuracy?
- Does the number of filters in CNN and batch size affect the performance of Word2Vec-CNN and GloVe-CNN?
- Does data imbalance degrade the performance of learning, and do increases in sample per class have any effect?

A. DATA COLLECTION

Eclipse and Mozilla are examples of large-scale open-source projects with a vast corpus of bug reports. Eclipse datasets (Platform and JDT) and two variants of Mozilla Firefox datasets were used in this study. One Firefox dataset was used by Lee *et al.* [9], and the other Firefox dataset was used by Mani *et al.* [3]. These datasets are used to verify the performance and trend of the proposed method by varying the number of samples (reports) per class (developer). Eclipse's Platform dataset has 4,825 bug reports recorded between 2013–08–01 and 2016–07–31. The number of total developers (or owners) was 225. Eclipse's JDT dataset and Firefox dataset were recorded between 2013–10–20 and 2016–10–20 and between 2013–08–01 and 2016–07–31, respectively. The number of developers for JDT and Firefox were 70 and 848, respectively. The dataset information is summarized in Table 3. The datasets and model are available on GitHub.[4]

[4]https://github.com/farhan-93/bugtriage

TABLE 3. Dataset used for experiment & evaluation.

Dataset	Bug reports	Developers
JDT	1465	70
Platform	4825	225
Firefox by [9]	13667	848

The second Firefox dataset [3] has 162,307 bug reports. Mani *et al.* [3] used 138,093 for training and 24,214 for classification tasks. The dataset is available in four formats, separated by thresholds: 0, 5, 10, and 20. These thresholds are used to create datasets with different numbers of samples per class. Therefore, all bug reports are included when the threshold is zero. The second dataset variant is derived using a threshold of 5 that includes all developers who have fixed at least five bug reports; other bug reports and developers are excluded. Similarly, developers who have fixed less than 10 and 20 bug reports for thresholds 10 and 20 were also excluded. These four dataset variants are derived from the Firefox dataset. The processed Firefox dataset is publicly available on the web page[5] in JSON format.

Four more datasets are used for comparing our models with recent studies. One is the GNU compiler collection (GCC) [6] dataset, which is a small dataset that has 2102 bug reports with an average of 32 bug reports for each developer which was used in [11]. The 64% of bug reports are selected for training, and the rest is used for testing.

The other three datasets have been used in CNN-DA [10], which have a massive number of bug reports. The Eclipse[7] dataset is the largest dataset with 39669 bug reports and 771 developers, collected between 2001-10-10 and 2014-12-29. The NetBeans[8] dataset has 19149 bug reports with 265 developers, recorded between 2000-10-21 and 2014-12-31. The last one is the Mozilla[9] dataset having 15501 bug report with 1022 developers, collected between 1999-03-21 and 2014-12-31. Only those bug reports are selected that have status "FIXED" and marked as "CLOSED," "VERIFIED," and "RESOLVED." For training we used 80% of the bug reports and their fixers, and the remaining 20% were used as a test dataset. The cleaned datasets and models are available on GitHub.[10]

B. EVALUATION MEASURE

The proposed method was evaluated by top-k accuracy: top-1 to top-10 accuracies were calculated to make a valid comparison with studies [3] and [9]. The top-k accuracy was calculated using the following equation:

$$Top - k\ accuracy = \frac{\sum_{i=1}^{N} I(rec_i@k, dev_i)}{|N|} \quad (6)$$

In the equation, N is the total number of bug reports, and k is the number of developers in a recommendation list. $rec_i@k$ and dev_i indicate recommended k developers and the fixer for bug report b_i, respectively. The function I returns 1 if the first parameter (a recommendation list) includes the second parameter (a fixer) and 0 otherwise.

A 10-fold cross-validation is used to evaluate the proposed model in our paper. The model is validated using the 10 percent of the total data. Recent studies including [5], [6], [18], [21]–[24], [44], [44], [45] also adopted cross-validation technique.

C. EXPERIMENTAL RESULTS AND EVALUATION

This section presents the experimental results of the proposed CNN-based bug triager with three-word representation techniques. First, the comparison of Word2Vec-CNN and GloVe-CNN is conducted by performing three different experiments with different batch sizes and numbers of filters (neurons). Second, a comparison of Word2Vec-CNN, GloVe-CNN, and ELMo-CNN with batch sizes of 32 and 256 filters is conducted. Third, we discuss the significance of the reported results. Finally, the described research questions are addressed by analyzing the experimental results and conducting a qualitative analysis.

1) COMPARISON OF Word2Vec-CNN AND GloVe-CNN

Several experiments with different hyper-parameter values were conducted to make valid comparisons between two different embedding-based models: Word2Vec-CNN and GloVe-CNN. The experimental results illustrate how different batch sizes and numbers of filters affect the accuracy of CNN in the bug triage problem. The following three experiments were performed, where G and W represent GloVe-CNN and Word2Vec-CNN based approaches, respectively:

G1/W1: GloVe-CNN and Word2Vec-CNN were trained on batch size 32 and 256 filters (neurons).

G2/W2: GloVe-CNN and Word2Vec-CNN were trained on batch size 64 and 256 filters (neurons).

G3/W3: GloVe-CNN and Word2Vec-CNN were trained on batch size 32 and 512 filters (neurons).

Although further experiments with different parameters other than the above could be conducted, we only report the results for the above experiments because the performance pattern can be observed. Each experiment was repeated five times, and the average top-1 to top-10 accuracy is depicted in Figures 2, 3, and 4. The top-k values are depicted on the x-axis, and percent accuracy values are depicted on the y-axis.

Figure 2 illustrates the comparison of the three experiments on the JDT dataset. GloVe has the highest accuracy in all experiments. G3 has a significant difference in accuracy until top-7 and a negligible difference from top-8 to top-10. Nevertheless, Word2Vec has superior results, and W2 and W3 have similar accuracy. Figure 3 illustrates the results of GloVe-CNN and Word2Vec-CNN for the Platform dataset. GloVe-CNN emphatically outperforms Word2Vec. Similarly,

[5]http://bugtriage.mybluemix.net/
[6]https://gcc.gnu.org/bugzilla/query.cgi
[7]https://bugs.eclipse.org/bugs/
[8]https://bz.apache.org/netbeans/query.cgi
[9]https://bugzilla.mozilla.org/query.cgi?
[10]https://github.com/farhan-93/bugtriage

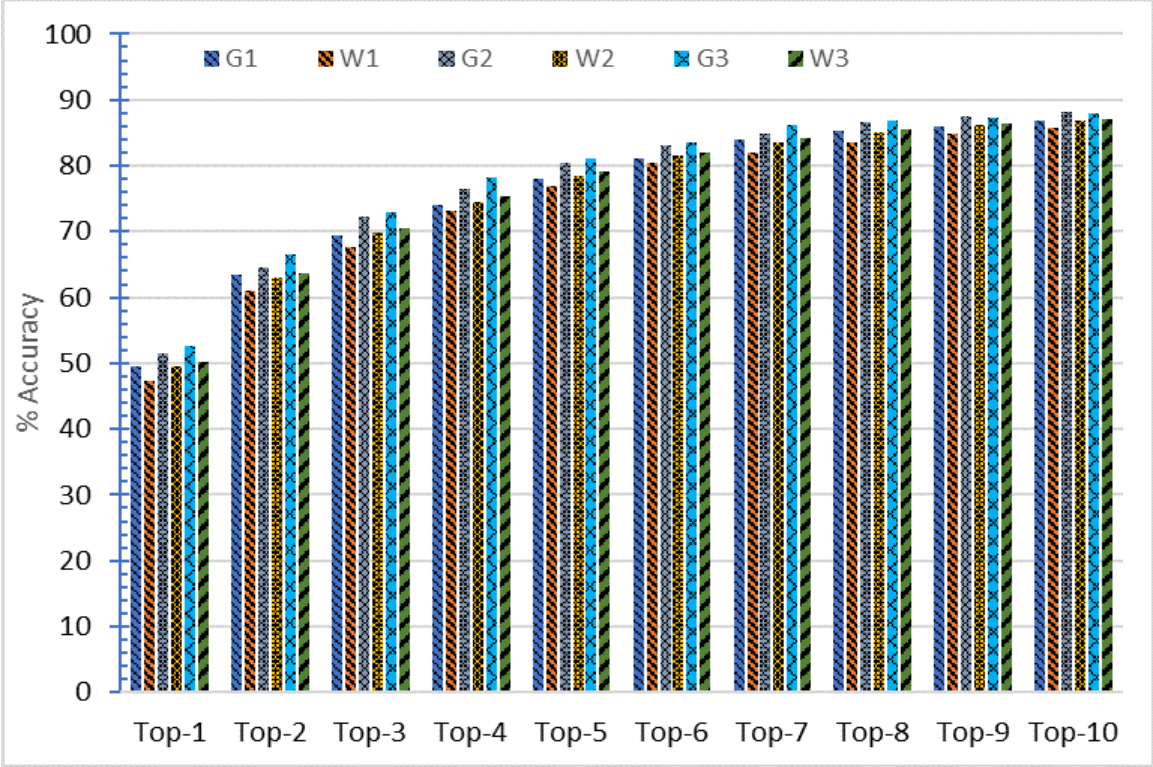


FIGURE 2. Comparison of all three experiments on JDT dataset.

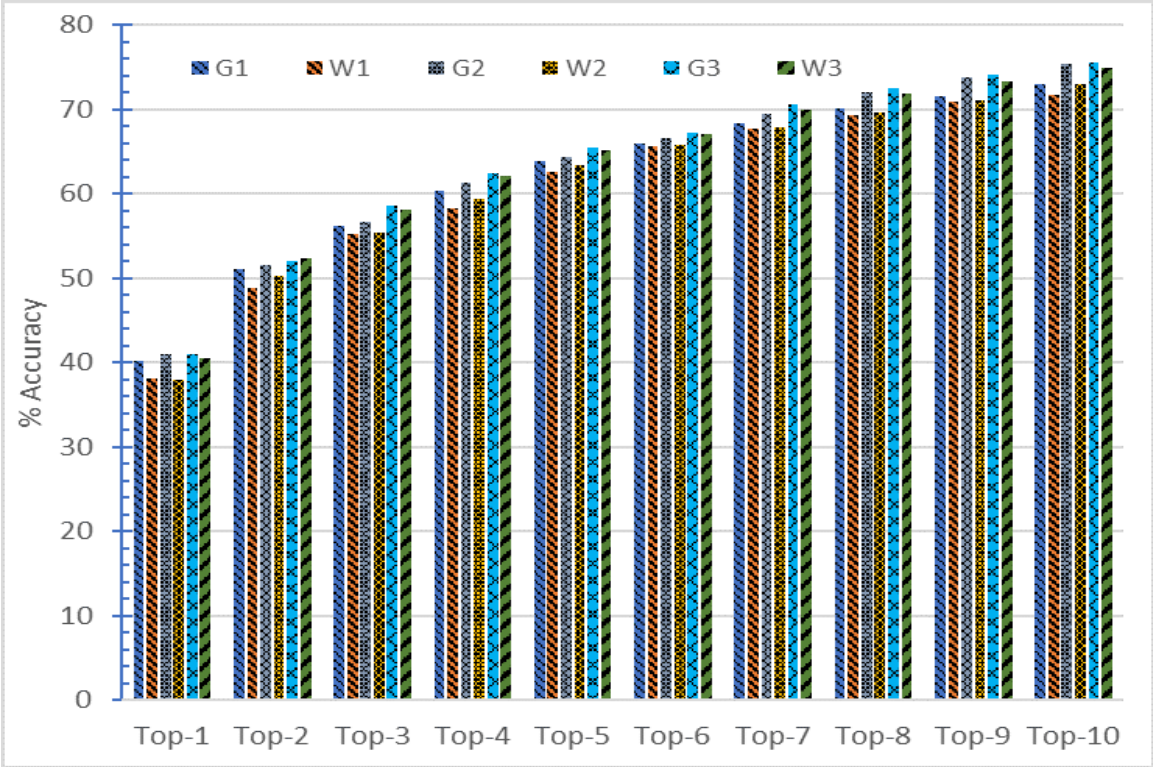


FIGURE 3. Comparison of all three experiments on Platform dataset.

top-8 to top-10 accuracy of G2 and G3 are negligibly different, with G3 superior for less than top-8 accuracy. W1 and W2 illustrate similar top-k accuracy, while W3 has higher accuracy than both W1 and W2.

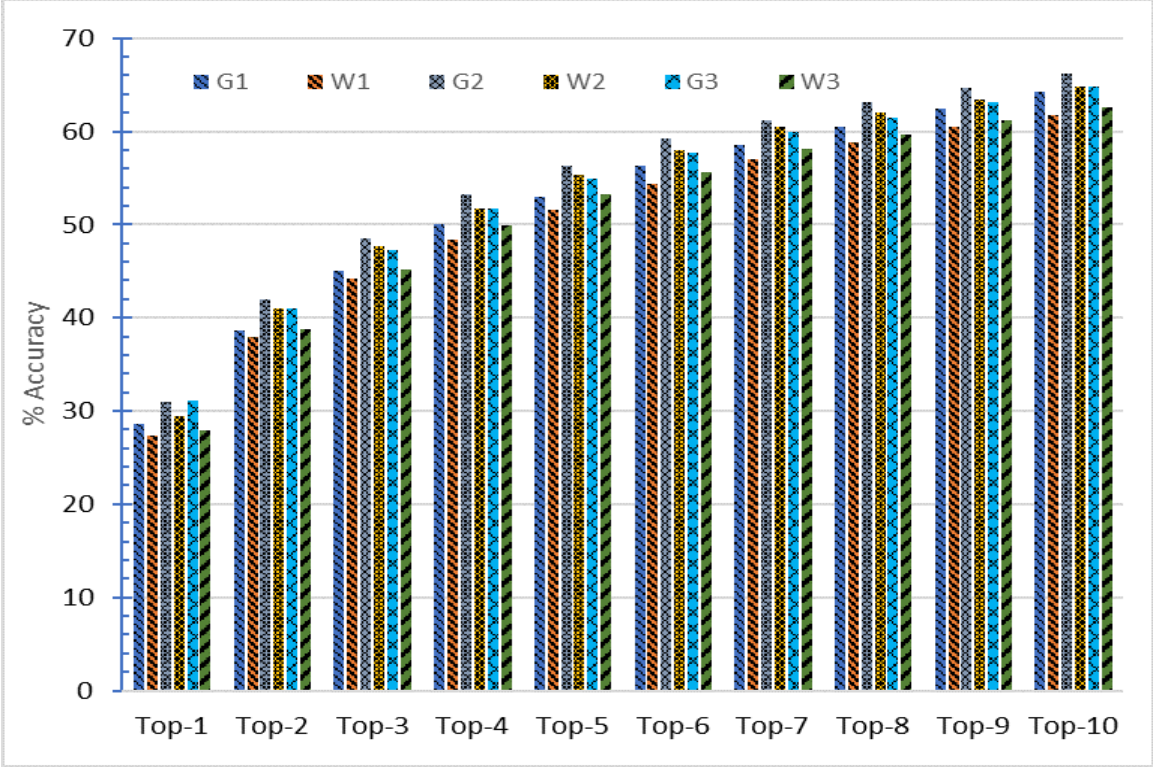


FIGURE 4. Comparison of all three experiments on Firefox dataset [9].

Figure 4 Figure 4 presents the comparison of experiments on the Mozilla Firefox dataset. GloVe-CNN is superior to Word2Vec-CNN in all cases. Comparing the GloVe-CNN experiments, G2 is significantly superior to G3 and G1. This observation differs significantly from the results of other datasets (JDT and Platform) where G3 is superior to G2. As previously described, Firefox [9]

2) COMPARISON OF ELMo-CNN WITH Word2Vec-CNN AND GloVe-CNN

This section presents a comparison between the suggested models based on the three different embedding techniques: ELMo-CNN, GloVe-CNN and Word2Vec-CNN. The parameters for ELMo-CNN experiments are the same as W1 and G1. The comparison results are depicted in Tables 4 and 5. As previously described, we used two Firefox datasets. The dataset used in the experiment depicted in Table 4 is from [9], and the dataset used in Table 5 is from [3]. The latter dataset is larger than the former and contains thresholds indicating the minimum number of samples per developer. The experimental results illustrate that ELMo-CNN outperforms Word2Vec-CNN and GloVe-CNN. The top-1 to top-10 accuracy is not significantly different for the JDT dataset. However, ELMo-CNN significantly outperforms Word2Vec-CNN and GloVe-CNN. Nevertheless, ELMo-CNN presents a much higher top-1 accuracy for the Platform and Firefox datasets.

The experimental results of the larger Firefox dataset with thresholds present that ELMo-CNN demonstrates the highest performance. Table 5 presents the top-1 to top-10 accuracy for each approach; however, the DeepTriage results are depicted only for top-10 accuracy because Mani *et al.* [3] reported the top-10 case only. The results present that ELMo-CNN outperforms GloVe-CNN, and GloVe-CNN outperforms Word2Vec-CNN. The accuracy significantly increases with the number of bug reports per developer. A detailed discussion of the results is included to answer the research questions.

3) SIGNIFICANCE OF RESULTS

We performed several tests to determine whether the results are statistically significant. A well-known non-parametric Friedman test is adopted to verify the overall significance of the results. The iterated results of Word2Vec-CNN, GloVe-CNN, and ELMo-CNN are considered as separate groups. Furthermore, a post hoc test is conducted for more precise statistical significance of the results. For the Nemenyi post hoc test, we count the repeated top-k accuracy's results of CNN variants with significant comparisons when the overall Friedman's test was significant. We use a significance level of $\alpha = 0.05$ for all tests in this study. We calculate the mean-rank for each CNN variant to determine the significant differences between them. These significance tests serve as evidence for other research questions.

JDT Dataset: The experimental results demonstrate minor differences in the mean top-k accuracy. For top-1 accuracy, ELMo has the highest performance, and GloVe outperforms Word2Vec. For top-1 accuracy, the Friedman test has a p-value of less than 0.05, indicating the presence of a

TABLE 4. The average top-1 to top-10 Accuracy obtained on JDT, Platform and Firefox [9] projects across the 10-fold cross validation. The best performing values are shown in bold.

Dataset	Techniques	top-1 Accuracy	top-2 Accuracy	top-3 Accuracy	top-4 Accuracy	top-5 Accuracy	top-6 Accuracy	top-7 Accuracy	top-8 Accuracy	top-9 Accuracy	top-10 Accuracy
JDT	Lee et al. [9]	46.6	61.1	68.1	71.7	74.3	-	-	-	-	-
	Word2Vec-CNN	47.364	60.912	67.586	73.232	76.93	80.502	82.062	83.434	84.8	85.674
	GloVe-CNN	49.55	63.372	69.480	74.006	78.040	81.152	84.012	85.222	86.014	86.912
	ELMo-CNN	**49.750**	**63.470**	**71.826**	**74.801**	**78.261**	**81.346**	**84.160**	**85.230**	**86.218**	**87.234**
Platform	Lee et al. [9]	36.1	45.8	50.5	53.7	56.7	-	-	-	-	-
	Word2Vec-CNN	38.036	48.870	55.160	58.220	62.492	65.604	67.632	69.324	70.832	71.714
	GloVe-CNN	40.132	51.004	56.234	60.318	63.770	65.964	68.252	70.058	71.554	72.930
	ELMo-CNN	**43.622**	**51.830**	**56.366**	**60.704**	**63.822**	**66.528**	**69.068**	**70.964**	**72.794**	**74.264**
Firefox [9]	Lee et al. [9]	27.1	36.7	42.8	47.1	50.5	-	-	-	-	-
	Word2Vec-CNN	27.396	37.894	44.256	48.346	51.604	54.422	57.006	58.858	60.430	61.750
	GloVe-CNN	28.614	38.660	45.062	50.094	53.024	56.374	58.496	60.522	62.410	64.224
	ELMo-CNN	**30.418**	**41.104**	**47.81**	**52.508**	**55.738**	**58.362**	**60.404**	**62.314**	**63.598**	**64.696**

TABLE 5. The average top-1 to top-10 Accuracy obtained on Firefox project used by Mani *et al.* [3] across the 10-fold cross validation. The best performing values are shown in bold.

Threshold	Techniques	top-1 Accuracy	top-2 Accuracy	top-3 Accuracy	top-4 Accuracy	top-5 Accuracy	top-6 Accuracy	top-7 Accuracy	top-8 Accuracy	top-9 Accuracy	top-10 Accuracy
Minimum 0 samples per class	Deep Triage [3]	-	-	-	-	-	-	-	-	-	38.1
	Word2Vec-CNN	15.2	23.52	28.77	32.07	35.15	37.34	39.56	41.37	42.89	43.63
	GloVe-CNN	16.84	25.22	31.78	33.78	35.76	39.19	42.19	43.96	43.96	45.32
	ELMo-CNN	**20.86**	**29.04**	**34.332**	**38.03**	**41.18**	**43.57**	**45.72**	**47.68**	**49.28**	**50.73**
Minimum 5 samples per class	Deep Triage [3]	-	-	-	-	-	-	-	-	-	44.5
	Word2Vec-CNN	17.43	25.01	29.58	33.28	37.50	40.87	42.98	44.32	45.76	45.91
	GloVe-CNN	18.47	26.8	32.62	36.94	38.29	41.34	43.41	45.31	46.01	47.64
	ELMo-CNN	**26.51**	**37.02**	**43.29**	**48.17**	**51.77**	**54.20**	**56.05**	**58.12**	**59.78**	**61.41**
Minimum 10 samples per class	Deep Triage [3]	-	-	-	-	-	-	-	-	-	51.4
	Word2Vec-CNN	19.19	27.55	33.58	39.05	41.41	43.85	45.13	46.02	47.90	51.06
	GloVe-CNN	19.25	29.69	34.65	40.23	42.64	44.68	46.68	48.35	49.44	51.67
	ELMo-CNN	**31.01**	**42.85**	**49.83**	**54.50**	**57.96**	**60.77**	**63.78**	**64.92**	**66.62**	**67.90**
Minimum 20 samples per class	Deep Triage [3]	-	-	-	-	-	-	-	-	-	55.8
	Word2Vec-CNN	24.34	34.40	39.73	44.62	48.19	58.68	52.93	54.78	56.74	58.94
	GloVe-CNN	23.16	33.43	39.63	44.72	48.58	52.10	54.34	56.74	58.50	59.92
	ELMo-CNN	**35.89**	**47.03**	**53.93**	**58.94**	**62.14**	**65.04**	**67.49**	**69.39**	**70.89**	**72.65**

significant difference in results. The Nemenyi test demonstrates that ELMo has significantly different values than Word2Vec because the p-value is less than 0.05. However, no significant difference is found between ELMo and GloVe. The values of the five repetitions are illustrated in Figure 5 (a) using a boxplot. The same test is performed for top-5 accuracy. The Friedman test demonstrates an insignificant difference in results. Nevertheless, ELMo-CNN outperforms the others. Furthermore, none of the p-values is less than 0.05 for the pairwise comparison of all three techniques. Figure 5 (b) illustrates a boxplot for top-5 accuracy. The significant test results for top-10 accuracy are equivalent to those of top-5 accuracy. Figure 5 (c) illustrates the boxplot for top-10 accuracy.

Platform Dataset: The Friedman test demonstrates a p-value of less than 0.05 for top-1 accuracy. The result of the Nemenyi test demonstrates the significance of ELMo-CNN over Word2Vec-CNN. However, GloVe-CNN demonstrates an in significant difference in both Word2Vec-CNN and ELMoCNN. For top-5 and top-10, the Friedman's p-values

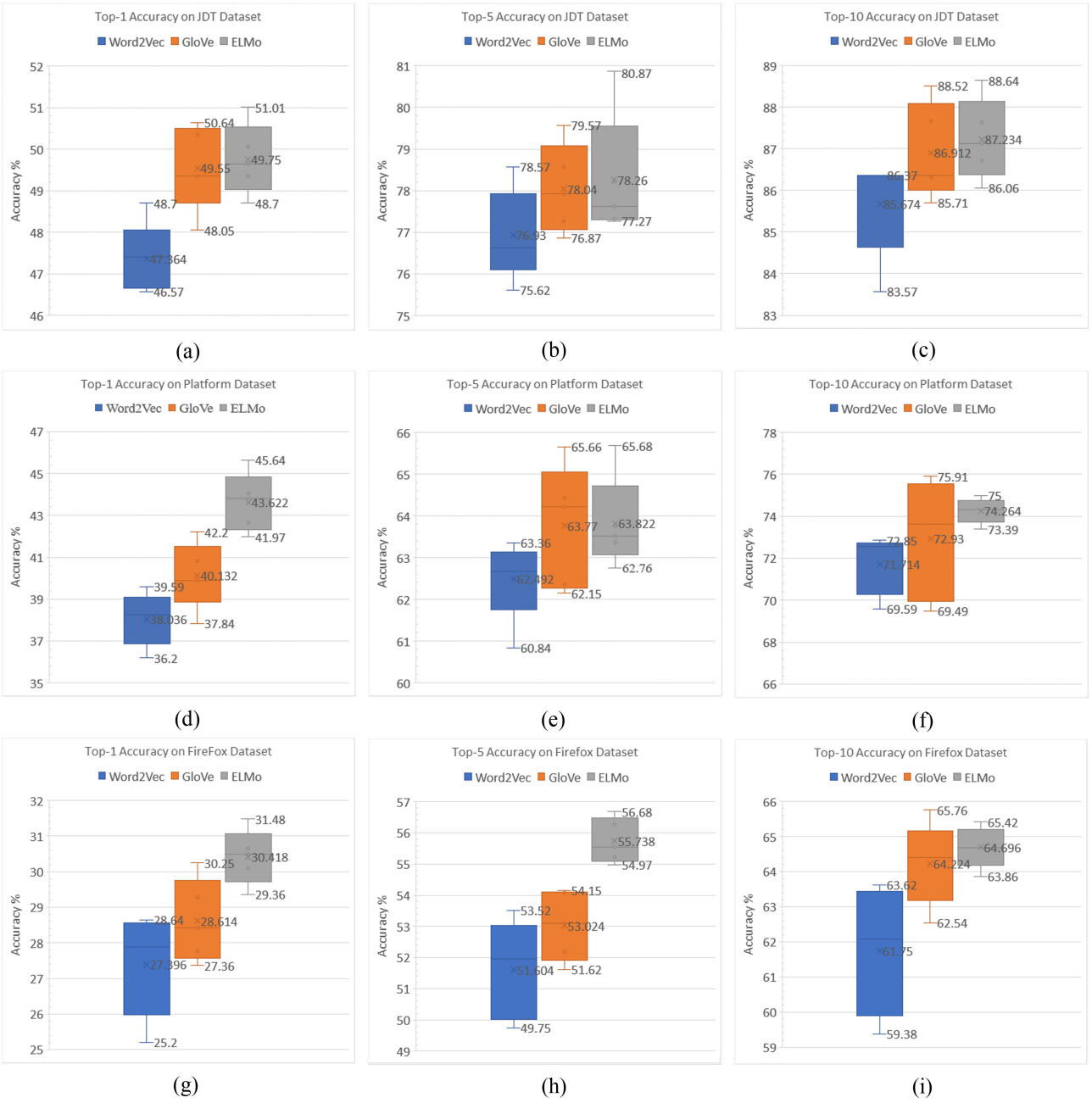


FIGURE 5. Box and Whisker plot for comparing the performance of Word2Vec-CNN, GloVe-CNN, and ELMo-CNN on JDT, Platform, and Firefox [9] datasets. Columns 1 to 3 show the top-1, top-5, top-10 accuracies, respectively. Rows 1 to 3 show plots for JDT, Platform, and Firefox datasets, respectively.

are greater than 0.05, which suggests an insignificant difference in the results; accordingly, the post hoc test is not performed. Figure 5 (d), (e), and (f) illustrate the boxplots for top-1, top-5, and top-10 accuracies, respectively.

Firefox Dataset: The Friedman test suggests that top-1 accuracy is significantly different because the p-value is less than 0.05, which is also true for top-5 accuracy. The Nemenyi test demonstrates that ELMo has significantly different accuracy values than Word2Vec. Nevertheless, GloVe-CNN is not significantly different from Word2Vec and ELMo-CNN. The Friedman test produces a p-value greater than 0.05 and illustrates insignificantly different results for top-10 accuracy. Figures 5 (g), (h), (i) illustrate the boxplots for top-1, top-5, and top-10 accuracies, respectively.

4) ADDRESSING THE RESEARCH QUESTIONS

RQ 1: Which is more effective for word representation in CNN-based bug triage, and does it make any difference if a different word embedding is used?

The experimental results illustrate that the CNN with ELMo word representation achieves the highest accuracy among the three approaches. As previously described,

ELMo is a context-sensitive word representation, whereas Word2Vec and GloVe are context-insensitive techniques. Context-sensitive word presentation assists the CNN model in learning the feature map more effectively than the context-insensitive technique, resulting in higher triage accuracy.

The experimental results suggest that GloVe-CNN outperforms Word2Vec-CNN. GloVe is more effective at exploiting parallelism than Word2Vec; thus, GloVe can be beneficial when handling a large set of training data [46]. Furthermore, GloVe can handle negative examples more effectively than Word2Vec. Keywords may be overlapped in many bug reports [44]. In contrast to the method in [3], it is difficult for CNN to remember long sentence semantics. However, CNN performs well with GloVe properties compared to Word2Vec. GloVe learns its vectors through dimension-reduction on the co-occurrence counts matrix; however, Word2Vec learns its vectors by improving the loss of predicting the words from context words [47]. As depicted in Figures 2, 3, and 4, the GloVe-based CNN technique achieves remarkable top-k accuracy and supports these arguments. The boxplots also illustrate that the ELMo-CNN achieves high mean-accuracy than GloVe-CNN and Word2Vec CNN.

RQ 2: Which method is the best to achieve superior top-1 accuracy?

ELMo-CNN has the highest top-1 accuracy, with significant differences for all datasets. ELMo-CNN also has higher accuracy than Word2Vec-CNN and GloVe-CNN for the JDT dataset but with a negligible difference. The experimental results on the Firefox dataset with thresholds demonstrate significant differences in top-1 accuracy. Figures 2, 3, and 4 illustrate that GloVe-CNN performs well for top-1 accuracy with a noticeable difference for JDT, Platform, and Firefox datasets in comparison to Word2Vec-CNN. G2 and G3 demonstrate modest performance for all datasets. G3 outperforms G2 and G1 for top-1 accuracy. The Demšar diagrams are depicted in Figure 6; the critical distance is calculated using a 95% confidence level (p-value ≤ 0.05). The Demšar diagram illustrates the average method ranks with critical distance above the rank line. The variants of CNN are connected in the Demšar diagram, which demonstrates an insignificant difference in top-1 accuracy. The Demšar diagram supports this research question partially and reveals a significant difference for ELMo-CNN over Word2Vec-CNN, whereas GloVe-CNN does not demonstrate any significance. ELMo-CNN outperforms GloVe-CNN, but it does not demonstrate any significant difference with a 95% confidence interval. These experimental results suggest that ELMo-CNN has the highest performance, followed by GloVe-CNN and then Word2Vec-CNN. ELMo-CNN demonstrates a significant difference in accuracy compared to Word2Vec-CNN. The ELMo-CNN outperforms GloVe-CNN but does not illustrate any significant difference for top-1 accuracy.

RQ 3:Does the number of filters in CNN and batch size affect the performance of Word2Vec-CNN and GloVe-CNN?

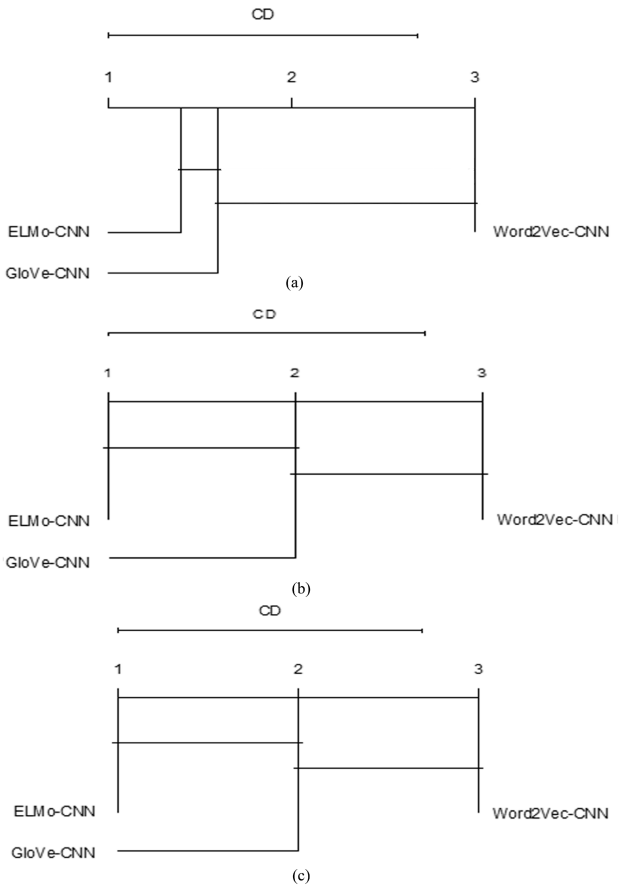


FIGURE 6. The Demšar diagram compares all three variants of CNN on all datasets for top-1 accuracy. The CD is the critical distance calculated by the Nemenyi test. The calculated CD value is 1.6873 with $\alpha = 0.05$. Sub-figure (a) shows the comparison for JDT, sub-figure (b) shows on comparison on Platform dataset, and sub-figure (c) shows the comparison on Firefox [9] dataset.

The experimental results of the JDT dataset demonstrate that G3 performs better for top-1 to top-7 accuracies. G2 (GloVe- CNN with 256 filters and 64 batches) exhibits similar accuracy for top-8 to top-10. Observations of the Platform dataset demonstrate comparable results. The Firefox dataset exhibits different results for G2 and G3. G2 and W2 outperform G3 and W3, respectively. Recall that our CNN is a shallow network. The batch size and number of filters are key parameters among the various hyper-parameters for CNN training.

If a complex and big dataset is given to a small CNN, then the batch size should be substantial. However, for small datasets, a strategy using the small batch size with many filters is a superior choice. Recall that the Firefox dataset is more substantial than either the JDT or Platform datasets. Experimental results support the previous statements. It is observed that G2 and W2 outperform G3 and W3, respectively, on the Firefox dataset. However, for the JDT and Platform datasets, G3 and W3 outperform G2 and W2, respectively. Word2Vec-CNN does not outperform GloVe-CNN.

From these observations, we can argue that the number of filters and batch size affect the performance of Word2VecCNN and GloVe-CNN. Large numbers of filters are recommended for small datasets such as JDT and Platform, where the number of bug reports per developer is large. When dealing with a large dataset and the number of bug reports per developer is small, we observed a greater increase in its batch size.

RQ 4: Does data imbalance degrade the performance of learning and do increases in sample per class have any effect?

Data imbalance in training can degrade the performance of machine learning. The JDT dataset contains 1465 bug reports and 70 developers or approximately 20 bug reports per developer on average. However, some developers fixed as few as five bug reports, which severely negatively affects the performance of CNN-based models.

All three techniques are tested on the Firefox dataset [3] with different thresholds to confirm the validity of the previous assertion. ELMo-CNN, Word2Vec-CNN, and GloVe-CNN perform well; however, ELMo-CNN demonstrates a significant accuracy difference. The Friedman test and Nemenyi test demonstrate the significance of the stated results. The top-1 accuracy results passed the Friedman and Nemenyi tests, which supports the assertion that ELMo-CNN is significantly different from Word2Vec-CNN and GloVe-CNN. The Friedman test has a p-value of 0.015, which is less than 0.05. The pairwise comparison using the Nemenyi test has p-values of less than 0.05 with ELMo-CNN for top-1 for all thresholds. Furthermore, ELMo-CNN demonstrates significant top-5 accuracy for all thresholds with a p-value of less than 0.05. No significant difference exists between Word2Vec-CNN and GloVe-CNN for top-5 and top-10 accuracy.

Similar results are observed for top-10 accuracy on all thresholds. The results demonstrate that ELMo-CNN achieves significantly high accuracy than Word2Vec-CNN and GloVe-CNN. The accuracy results demonstrate a small difference in top-k accuracy for a threshold minimum of 0 samples per class. Top-k accuracy increases as the number of samples per class increases. The top-10 accuracy of GloVe-CNN is 45.32%, 47.64%, 51.67%, and 59.92% for minimum samples per class of 0, 5, 10, and 20. The top-10 accuracy of Word2Vec-CNN is 43.63%, 45.91%, 51.06%, and 58.94% for thresholds of 0, 5, 10 and 20. The findings are similar for top-1 accuracy. ELMo-CNN has the highest top-10 accuracy results at 50.73%, 61.41%, 67.90%, and 72.65% for thresholds of 0, 5, 10, and 20. ELMo-CNN has the highest top-1 accuracy, with significant differences compared to Word2VecCNN and GloVe-CNN. GloVe-CNN outperforms Word2VecCNN. Word2Vec-CNN outperforms GolVe-CNN only for a threshold of 20. GloVe-CNN has higher accuracy for top-5 and top-10 accuracy with negligible differences.

The above results suggest that the data imbalance may degrade the performance of learning. Furthermore, the increase in training samples per class yields superior performance.

D. COMPARISONS WITH OTHER RESEARCH

All three models are compared with few previous studies. Table 4 presents the comparison of ELMo-CNN, GloVe-CNN, and Word2Vec-CNN with the results of Lee *et al.* [9]. The results are compared with batch 32 and 256 filters.

ELMo-CNN significantly outperforms that of Lee *et al.*, which is our previous approach [9]. In Table 4, the ELMo-CNN results reveal insignificant differences compared to GloVe-CNN; however, they are significantly superior to Lee *et al.* [9] and Word2Vec-CNN for the JDT dataset. ELMo seems to be the best choice among all three models, and GloVe-CNN outperforms Word2VecCNN. Word2Vec-CNN and Lee *et al.*'s approach [9] are almost identical, except the former uses the pretrained embedding vectors, and the latter is trained on the entire bug dataset used in the experiments. In the Platform and Firefox datasets, the approach of using pre-trained vectors outperforms the approach trained over the dataset. However, there is no clear winner between Word2Vec and Lee *et al.* [9] with the JDT dataset.

The results in Table 5 present a small difference in percent accuracy between GloVe and Word2Vec. Firefox [3] is a large dataset compared to other datasets. ELMo-CNN presents a significant difference for all thresholds. GloVe-CNN performs well for thresholds of 0, 5, and 10. Word2Vec-CNN performs more effectively on a threshold of at least 20 samples per class when compared to GloVe-CNN. All three approaches outperform the approach proposed by Mani *et al.* [3] with noticeable differences. Table 5 presents detailed experimental results, and the best values are depicted in bold.

Table 6 shows the comparative results of the GCC dataset, which was used in [11]. The dataset was split in 66% and 34% for the training set and testing set. The ELM approach performs better than our techniques. Word2Vec-CNN, GloVe-CNN, and ELMo-CNN perform better than SVM, Naive Bayes, C4.5 (decision tree), and KNN. However, ELMo-CNN shows a small difference in accuracy results with ELM. The reported results are average of 5 experiments. The top-5 and top-10 accuracy are also shown in Table 6.

Table 7 shows the comparison results of our models with CNN-DA, bag of words + Naïve Bayes (BOW+NB), and one-hot CNN methods that were reported in [10]. We use the first 80% bug reports for training, and the last 20% is used for testing based on a chronological order of the submission time. The batch size is set to 50 to meet the parameter requirements. Overall, ELMo-CNN and GloVe-CNN perform better than the CNN-DA for all top-k accuracy except on Eclipse Dataset. The Word2Vec-CNN results show a negligible difference in top-1 to top-10 accuracy. The CNN-DA performs well on the Eclipse dataset for top-1 accuracy with a small difference. However, the ELMo-CNN

TABLE 6. The average top-1 to top-10 Accuracy obtained on Yin *et al.*'s dataset [11]. The dataset is split and 34% of data is used for testing. The best performing values are shown in bold.

Dataset	Techniques	Accuracy	top-5	top-10
GCC	ELM [11]	**63.3**	-	-
	SVM [11]	55.6	-	-
	Naive Bayes [11]	39.4	-	-
	C4.5 [11]	55.6	-	-
	KNN [11]	46.2	-	-
	Word2Vec-CNN	55.92	71.04	78.3
	GloVe-CNN	59.65	71.25	79.85
	ELMo-CNN	62.45	74.02	83.20

shows better results from top-2 to top-10 accuracy. The top-10 accuracy results show a notable difference between the performance of CNN-DA and ELMo-CNN. For NetBeans and Mozilla datasets, ELMo-CNN and GloVe-CNN perform better than the CNN-DA method. A small difference is found between CNN-DA and ELMo-CNN results from top-8 to top-10 accuracy for NetBeans dataset, and CNN-DA shows better top-10 accuracy results than ELMo-CNN. The Word2Vec-CNN shows similar performance with small differences in accuracy results because CNN-DA also used Word2Vec representation with 200-dimensions while Word2Vec-CNN used pre-trained vector for word-embedding with 300-dimensions. The GloVe-CNN and ELMo-CNN show better results most of the time throughout the experiment compared to Word2Vec. The superiority of the GloVe and ELMo is already explained in RQ 1. Therefore, ELMo-CNN seems to be an appropriate candidate to achieve good top-1 accuracy.

E. COMPLEXITY AND SCALABILITY

Deep learning methods are costly due to their onerous memory storage requirements, long learning time, and computational complexity. For our study, we used pre-trained vectors, which do not require significant time to embed the word vectors. The working CNN network model is shallow and not very complicated in terms of storage. We executed the experiments on an Intel Core i7 machine with a GeForce GTX 1080Ti GPU and 64 GB of RAM. Two to three hours were required to embed and train the network on Firefox datasets. For the JDT dataset, the model required less than 10 and 20 minutes for embedding and training the model on the JDT and Platform datasets, respectively. The same model required approximately 1 hour and 15 minutes to train the model on the Firefox [9] dataset. We can conclude that the model is not computationally complex.

Furthermore, our proposed technique is scalable. We tested on a system with less main memory (16 GB) and a lower-performing GPU (GeForce GTX 1050). Several hours were required for embedding and training, which demonstrates the scalability of the method, which can be readily used for different datasets. Most of the bug reports have title/summary, description, and fixer information. All open-source projects and industrial projects have datasets in Extensible Markup Language (XML) format, which can be easily converted to Comma Separated Values (CSV) format. Therefore, this method can be adapted to any dataset or industrial project.

V. THREATS TO VALIDITY

The following are possible threats to validity.

- Only the summaries and descriptions from bug reports were used. The performance of the proposed work is validated on open source projects. The summaries and descriptions are used as input, while the owner information is used as a class attribute. Many recent studies used only these attributes to solve the bug triage problem. Bug triage is a software engineering problem that is being solved by NLP techniques. In contrast, additional information might help the machine to learn more effectively and improve the results, but storage and time complexities would increase.
- This study uses the pre-trained vectors for GloVe and Word2Vec embedding and compares the results with [3]. A large corpus of the Firefox dataset for training the Word2Vec model was used, which is publicly available. Mani *et al.* [3] separate the unassigned, unresolved, and unfixed bug reports from a large corpus to train the Word2Vec model and fixed and resolved bug reports to train the classifier. The same dataset is used to train our CNN models. The dataset for the Word2Vec model's training is not used in this study because the proposed method uses pre-trained vectors. Therefore, the validity of the comparison with the proposed work can be questioned. Nonetheless, the comparison is valid because both studies use the same dataset to train the classifier.
- The proposed method was not tested on industrial projects. We do not argue that our method is best for industrial projects because industrial projects may follow different patterns and have different characteristics than open-source projects. As previously described, this study is an extension of the technique presented by Lee *et al.* [9], which tested the proposed method on industrial projects. Therefore, we hope that our method is tested on industrial projects and yields similar improvements observed in this study.
- Another question can be raised as to why the proposed work has been compared with only few studies. Comparing our results with many other studies is not possible because most of the other studies used different datasets. Even in the cases where the same project is considered, the data collection duration or periods are not matched. In [3], a benchmark dataset was created that contains a large-scale dataset of three open-source projects. Therefore, the Firefox dataset is used to validate this work.
- Yet another risk is related to the difficulties with reproducing the training and testing datasets utilized in the previous research. The cleaned datasets are unavailable

TABLE 7. The average top-1 to top-10 Accuracy obtained on Guo *et al.*'s dataset [10]. The dataset is split and 20% of data is used for testing. The best performing values are shown in bold.

Dataset	Techniques	top-1 Accuracy	top-2 Accuracy	top-3 Accuracy	top-4 Accuracy	top-5 Accuracy	top-6 Accuracy	top-7 Accuracy	top-8 Accuracy	top-9 Accuracy	top-10 Accuracy
Mozilla	CNN-DA [10]	0.1244	0.1909	0.2254	0.2491	0.2693	0.2826	0.3017	0.3171	0.3300	0.3446
	One-Hot+CNN [10]	0.0816	0.1569	0.1967	0.20768	0.2357	0.2519	0.2718	0.2835	0.3011	0.3286
	BOW+NB [10]	0.0815	0.1143	0.1369	0.1578	0.1749	0.1868	0.1976	0.2129	0.2265	0.2369
	Word2Vec-CNN	0.1274	0.1783	0.2196	0.2570	0.2741	0.2931	0.3124	0.3292	0.3453	0.3576
	GloVe-CNN	0.1609	0.2412	0.2902	0.3182	0.3411	0.3631	0.3847	0.4008	0.4127	0.4263
	ELMo-CNN	**0.1673**	**0.2518**	**0.3015**	**0.3398**	**0.3647**	**0.3895**	**0.4089**	**0.4237**	**0.4405**	**0.4540**
Eclipse	CNN-DA [10]	**0.2520**	0.3508	0.3970	0.4328	0.4603	0.4834	0.4988	0.5124	0.5236	0.5337
	One-Hot+CNN [10]	0.1973	0.2949	0.3128	0.3394	0.3536	0.3619	0.3867	0.3918	0.4193	0.4294
	BOW+NB [10]	0.0374	0.0374	0.0374	0.03748	0.0374	0.0374	0.0374	0.0374	0.0374	0.0374
	Word2Vec-CNN	0.2249	0.3415	0.3870	0.4139	0.4311	0.4614	0.4772	0.4981	0.5086	0.5174
	GloVe-CNN	0.2470	0.3600	0.3912	0.4218	0.4720	0.4911	0.5184	0.5271	0.5299	0.5435
	ELMo-CNN	0.2500	**0.3641**	**0.4196**	**0.4562**	**0.4917**	**0.5026**	**0.5322**	**0.5471**	**0.5601**	**0.5721**
NetBeans	CNN-DA [10]	0.3554	0.4802	0.5436	0.5893	0.6266	0.6539	0.6811	0.7111	0.7273	**0.7489**
	One-Hot+CNN [10]	0.3049	0.4462	0.4876	0.5391	0.5618	0.6029	0.6379	0.6661	0.6793	0.6915
	BOW+NB [10]	0.2157	0.3041	0.3649	0.4039	0.4359	0.4572	0.4625	0.5018	0.5195	0.5400
	Word2Vec-CNN	0.3573	0.4840	0.5399	0.5910	0.6134	0.6271	0.6458	0.6618	0.6780	0.6892
	GloVe-CNN	0.3851	0.4927	0.5517	0.5911	0.6201	0.6481	0.6799	0.7087	0.7126	0.7284
	ELMo-CNN	**0.4017**	**0.5313**	**0.5905**	**0.6302**	**0.6607**	**0.6825**	**0.7006**	**0.7154**	**0.7292**	0.7423

in many cases. Most researchers reported limited information only such as Bugzilla main page/source URL, time interval, resolution, and status, etc. Especially, the cleaning recipes are not explained in detail in most previous research, hence it is hard to make sure that we are performing the experiments on the same dataset. We tried our best to reproduce the dataset by following the information provided in the previous research; however, we should admit that there still can be a chance that different bug reports might be selected during cleansing of data.

- ELMo-CNN is only compared to the other techniques using one parameter set (32 batch size and 256 filters). ELMo is a context-sensitive technique; however, Word2Vec and GloVe are context-insensitive techniques. The word vectors of Word2Vec and GloVe are available with 300 dimensions. The ELMo generates the 1024-dimensional embedding vector, but the dimensions of ELMo are reduced to 300 dimensions. For a valid comparison, only Word2Vec and GloVe embedding techniques were used, and they were sufficient to observe the trend between two parameters (the batch size and number of filters).
- We did not use batch sizes smaller than 32 or larger than 64 for the final results. We performed experiments with smaller batch sizes of 10 and 16, which exhibited the lowest performance and thus did not include those results. Larger batch sizes demonstrated a similar trend, so we did not include those results either. By increasing batch size, deep learning performance is increased, but significant memory is required for computation. Comparative studies also used a batch size of 32, so we used batch sizes of 32 and 64; these are sufficient to identify trends. Performance is increased by increasing the batch size for more massive datasets. The use of a small batch size is a superior choice if the dataset is not too large.

VI. LIMITATIONS

This section describes the limitations of our work. Most automated approaches for the bug triage which are utilizing machine learning techniques like our study typically exploit the information of resolved bugs in the past. Such approaches suggest a set of developers based on the participation records of the developers in bug fixing activities, hence their recommendations do not include new developers who do not appear in the history of the resolved bugs. For example, a new developer hired by an organization is not recorded as a fixer for any resolved bugs, hence the developer will not be considered as a candidate by the automated triager. Our approach also suffers from this limitation.

Another point is that typical bug triage dataset is highly imbalanced and skewed. There exist many developers who has fixed very few bug reports. These developers can be treated as outliers by the machine learner during the training, hence these outliers may negatively affect the performance of the models.

The above problems can be addressed by a sub-field of machine learning known as one-class classification, which is widely used for detecting outliers or anomalies. One-class classification is an unsupervised learning algorithm that can model normal examples to classify a new input as either normal or abnormal.

The one-class classification technique can be useful for imbalanced multi-class datasets where few instances are available for minority classes, or no coherent structure exists to separate the class that could be learned by a supervised technique. We are intended to use the one-class classification technique in the future to address the imbalance problem in the context of the bug triage application.

The one-class classification technique can also be used for addressing the issue of new developers. We can fine-tune the existing model with a new developer's feature using one-class classification, hence the model can be manipulated to recommend a new developer. Perera *et al.* [48] have proposed the idea of learning deep features using the one-class classification for anomaly detection and novelty detection, which showed good performance. The proposed methods operated on top of CNN and produced descriptive feature space while maintaining a low intra-class variance in the feature space for the target class. Hempstalk *et al.* [49] used one-class classification for the multi-class classification task. They collected positive examples of each class for training and testing. They evaluated their five techniques; multi-class classification (biased), two-class classification (biased), multi-class classification (unbiased), two-class classification (unbiased), and one-class classification. The one-class classification performed better than the unbiased multi-class classifier because the one-class classification is intended to deal with new classes and learns only the target class during the training. So, we can use combinations of multiple one-class classifiers for multi-class classification problems.

In summary, we are planning to adopt one-class classifier to improve the performance of the automated bug-triage approach in our future research.

VII. CONCLUSION

Bug triage is a crucial software engineering problem. In this paper, we proposed a CNN-based technique that uses two context-insensitive and one context-sensitive word representation techniques. The pre-trained vectors Word2Vec and GloVe are used for word embedding, whereas the trainable ELMo model is used for context-sensitive word embedding. The proposed technique learns the summaries and descriptions from bug reports and recommends a ranked list of ten appropriate developers. We use top-k accuracy as an evaluation metric. For the experimental analysis, the bug reports are collected from Eclipse's Platform, Eclipse's JDT, and Mozilla's Firefox datasets, GCC and NetBeans. The experimental results demonstrate that ELMo-CNN outperforms GloVe-CNN and Word2Vec-CNN models. The Friedman and Nemenyi tests were conducted to confirm the significance of the results. The experimental results demonstrate significant differences in top-1 accuracy. The ELMo-CNN demonstrated significant top-1 and top-5 accuracy compared to the other two techniques for large Firefox datasets except for the minimum 0 class threshold. The Nemenyi test demonstrated that ELMo-CNN has significant top-1 accuracy compared to Word2Vec-CNN. However, there was no significant difference in top-1 accuracy for GloVe-CNN. The mean-accuracy results demonstrate that ELMo-CNN is superior for all top-1 to top-10 accuracies. Word2Vec-CNN achieves higher top-1 accuracy compared to GloVe-CNN, where the number of samples is at least 20 for each class; otherwise, GloVe-CNN outperforms Word2Vec-CNN. In all cases of large Firefox datasets except 0 thresholds, the Nemenyi test demonstrates higher performance for ELMo-CNN compared to the other two techniques for top-1 and top-5 accuracy.

Furthermore, three types of experiments were conducted with different parameters for GloVe-CNN and Word2Vec-CNN to study the trend between batch size and number of filters in the CNN. The comparison of experimental results finds that if a large dataset with a considerable number of classes is available, increasing the batch size is a suitable option. If a small dataset is available, then increasing the number of filters is a suitable option. We also conclude that context-sensitive word-embedding techniques yield superior results to context-insensitive techniques. The ELMo model is trainable because it uses the softmax classifier. ELMo model has four trainable scalar weights for layer aggregation so that the ELMo model can be fine-tuned on the given training data during the embedding task. A shallow network was used for the training of the classifier. Finally, the technique is scalable and can be easily adapted to any dataset.

In the future, we plan to experiment with other forms of neural networks for bug triage problems, such as a bio-inspired spiking CNN (SCNN). Such a network lies in the third generation of neural networks and is considered higher performing than the traditional, non-spiking neural networks because of its bio-realism. Moreover, we intend to use an extensive corpus of bug data in the future. Also, we are intended to improve the triage system to assign the new developers to the bug reports.

REFERENCES

[1] H. Hu, H. Zhang, J. Xuan, and W. Sun, "Effective bug triage based on historical bug-fix information," in *Proc. IEEE 25th Int. Symp. Softw. Rel. Eng.*, Nov. 2014, pp. 122–132. [Online]. Available: http://ieeexplore.ieee.org/document/6982620/

[2] K. Sahu, U. Lilhore, and N. Agarwal, "Survey of various data reduction methods for effective bug report analysis," *Ijsrcseit*, vol. 3, no. 1, pp. 661–665, 2018. [Online]. Available: http://ijsrcseit.com/paper/CSEIT11831139.pdf

[3] S. Mani, A. Sankaran, and R. Aralikatte, "DeepTriage: Exploring the effectiveness of deep learning for bug triaging," in *Proc. ACM India Joint Int. Conf. Data Sci. Manage. Data*, 2019, pp. 171–179. [Online]. Available: https://dl.acm.org/citation.cfm?id=3297023

[4] A. Yadav, S. K. Singh, and J. S. Suri, "Ranking of software developers based on expertise score for bug triaging," *Inf. Softw. Technol.*, vol. 112, pp. 1–17, Aug. 2019.

[5] A. Sajedi Badashian, A. Hindle, and E. Stroulia, "Crowdsourced bug triaging," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2015, pp. 506–510. [Online]. Available: http://ieeexplore.ieee.org/document/7332503/

[6] X. Peng, P. Zhou, J. Liu, and X. Chen, "Improving bug triage with relevant search," in *Proc. 29th Int. Conf. Softw. Eng. Knowl. Eng.*, Jul. 2017, pp. 123–128. https://ksiresearchorg.ipage.com/seke/seke17paper/seke17paper_41.pdf

[7] M. Kumari, A. Misra, S. Misra, L. Fernandez Sanz, R. Damasevicius, and V. B. Singh, "Quantitative quality evaluation of software products by considering summary and comments entropy of a reported bug," *Entropy*, vol. 21, no. 1, p. 91, Jan. 2019.

[8] R. A. Stein, P. A. Jaques, and J. F. Valiati, "An analysis of hierarchical text classification using word embeddings," *Inf. Sci.*, vol. 471, pp. 216–232, Dec. 2019.

[9] S.-R. Lee, M.-J. Heo, C.-G. Lee, M. Kim, and G. Jeong, "Applying deep learning based automatic bug triager to industrial projects," in *Proc. 11th Joint Meeting Found. Softw. Eng.*, 2017, pp. 926–931. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3106237.3117776

[10] S. Guo, X. Zhang, X. Yang, R. Chen, C. Guo, H. Li, and T. Li, "Developer activity motivated bug triaging: Via convolutional neural network," in *Proc. Neural Process. Lett.*, pp. 1–18, Dec. 2020.

[11] Y. Yin, X. Dong, and T. Xu, "Rapid and efficient bug assignment using ELM for IOT software," *IEEE Access*, vol. 6, pp. 52713–52724, 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8458103/

[12] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proc. 28th Int. Conf. Softw. Eng.*, New York, NY, USA, 2006, p. 361. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1134285.1134336

[13] S. N. Ahsan, J. Ferzund, and F. Wotawa, "Automatic software bug triage system (BTS) based on latent semantic indexing and support vector machine," in *Proc. 4th Int. Conf. Softw. Eng. Adv.*, Sep. 2009, pp. 216–221. [Online]. Available: http://ieeexplore.ieee.org/document/5298419/

[14] W. Wu, W. Zhang, Y. Yang, and Q. Wang, "DREX: Developer recommendation with K-Nearest-Neighbor search and expertise ranking," in *Proc. 18th Asia–Pacific Softw. Eng. Conf.*, Dec. 2011, pp. 389–396. http://ieeexplore.ieee.org/document/6130646/

[15] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports," in *Proc. 34th Int. Conf. Softw. Eng. (ICSE)*, Jun. 2012, pp. 14–24. [Online]. Available: http://ieeexplore.ieee.org/document/6227210/

[16] R. Shokripour, Z. M. Kasirun, S. Zamani, and J. Anvik, "Automatic bug assignment using information extraction methods," in *Proc. Int. Conf. Adv. Comput. Sci. Appl. Technol. (ACSAT)*, Nov. 2012, pp. 144–149. [Online]. Available: http://ieeexplore.ieee.org/document/6516342/

[17] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "Why so complicated? Simple term filtering and weighting for location-based bug report assignment recommendation," in *Proc. 10th Work. Conf. Mining Softw. Repositories (MSR)*, May 2013, pp. 2–11. [Online]. Available: http://ieeexplore.ieee.org/document/6623997/

[18] M. Alenezi, K. Magel, and S. Banitaan, "Efficient bug triaging using text mining," *J. Softw.*, vol. 8, no. 9, pp. 2185–2191, Sep. 2013. [Online]. Available: http://www.jsoftware.us/vol8/jsw0809-12.pdf

[19] T. Zimmermann and A. C. Artís. (2019). *Impact of Switching Bug Trackers: A Case Study on a Medium-Sized Open Source Project*. [Online]. Available: https://hal.inria.fr/hal-01951176v2/document

[20] G. Yang, T. Zhang, and B. Lee, "Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports," in *Proc. IEEE 38th Annu. Comput. Softw. Appl. Conf.*, Jul. 2014, pp. 97–106. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6899206

[21] J. Xuan, H. Jiang, Y. Hu, Z. Ren, W. Zou, Z. Luo, and X. Wu, "Towards effective bug triage with software data reduction techniques," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 1, pp. 264–280, Jan. 2015. [Online]. Available: http://ieeexplore.ieee.org/document/6815966/

[22] V. Dedik and B. Rossi, "Automated bug triaging in an industrial context," in *Proc. 42th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, Aug. 2016, pp. 363–367. [Online]. Available: http://ieeexplore.ieee.org/document/7592819/

[23] L. Jonsson, M. Borg, D. Broman, K. Sandahl, S. Eldh, and P. Runeson, "Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts," *Empirical Softw. Eng.*, vol. 21, no. 4, pp. 1533–1578, Aug. 2016. [Online]. Available: http://link.springer.com/10.1007/s10664-015-9401-9

[24] J. Xuan, H. Jiang, Z. Ren, J. Yan, and Z. Luo, "Automatic bug triage using semi-supervised text classification," 2017, *arXiv:1704.04769*. [Online]. Available: http://arxiv.org/abs/1704.04769

[25] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, nos. 1–3, pp. 489–501, Dec. 2006.

[26] R. Johnson and T. Zhang, "Supervised and semi-supervised text categorization using LSTM for region embeddings," 2016, *arXiv:1602.02373*. [Online]. Available: http://arxiv.org/abs/1602.02373

[27] R. Damaăevicius and M. Wožniak, "Sentiment analysis of lithuanian texts using traditional and deep learning approaches," *Computers*, vol. 8, no. 1, p. 4, Jan. 2019.

[28] B. Jang, I. Kim, and J. W. Kim, "Word2vec convolutional neural networks for classification of news articles and tweets," *PLoS ONE*, vol. 14, no. 8, Aug. 2019, Art. no. e0220976.

[29] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *IEEE Comput. Intell. Mag.*, vol. 13, no. 3, pp. 55–75, Aug. 2018.

[30] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," 2018, *arXiv:1802.05365*. [Online]. Available: http://arxiv.org/abs/1802.05365

[31] Y. Si, J. Wang, H. Xu, and K. Roberts, "Enhancing clinical concept extraction with contextual embeddings," 2019, *arXiv:1902.08691*. [Online]. Available: http://arxiv.org/abs/1902.08691

[32] H. Bai and H. Zhao, "Deep enhanced representation for implicit discourse relation recognition," 2018, *arXiv:1807.05154*. [Online]. Available: http://arxiv.org/abs/1807.05154

[33] C. Dogan, A. Dutra, A. Gara, A. Gemma, L. Shi, M. Sigamani, and E. Walters, "Fine-grained named entity recognition using ELMo and wikidata," 2019, *arXiv:1904.10503*. [Online]. Available: http://arxiv.org/abs/1904.10503

[34] K. Qi, J. Du, Q. Ou, L. Jin, and J. Zhong, "Bidirectional semantic matching with deep contextualized word embedding for chinese sentence matching," in *Proc. CCKS Tasks*, 2018, pp. 69–76.

[35] *ReLu Activations—Keras Documentation*. Accessed: Nov. 4, 2018. [Online]. Available: https://keras.io/activations/#relu

[36] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for LVCSR using rectified linear units and dropout," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2013, pp. 8609–8613. [Online]. Available: http://ieeexplore.ieee.org/document/6639346/

[37] C. M. Bishop, *Pattern Recognition and Machine Learning* (Information Science Statistics). New York, NY, USA: Springer, 2006.

[38] I. V. Tetko, D. J. Livingstone, and A. I. Luik, "Neural network studies. 1. comparison of overfitting and overtraining," *J. Chem. Inf. Model.*, vol. 35, no. 5, pp. 826–833, Sep. 1995. [Online]. Available: http://pubs.acs.org/cgi-bin/doilookup/?10.1021/ci00027a006

[39] N. Srivastava and G. Hinton, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[40] *Initializers—Keras Documentation*. Accessed: Nov. 4, 2018. [Online]. Available: https://keras.io/initializers/

[41] X. Glorot, Y. B. P. of the thirteenth international Conference, and U. 2010, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. 13th Int. Conf. Artif. Intell. Statist.*, 2010, pp. 249–256.

[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980

[43] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural Networks: Tricks Trade*. Berlin, Germany: Springer, 2012, pp. 437–478. [Online]. Available: http://link.springer.com/10.1007/978-3-642-35289-8_26

[44] M. Wei, S. Guo, R. Chen, and J. Gao, *Enhancing Bug Report Assignment with an Optimized Reduction of Training Set*. Cham, Switzerland: Springer, Aug. 2018, pp. 36–47. [Online]. Available: http://link.springer.com/10.1007/978-3-319-99247-1_4

[45] H. Zhao, S. Sun, and B. Jin, "Sequential fault diagnosis based on LSTM neural network," *IEEE Access*, vol. 6, pp. 12929–12939, 2018. [Online]. Available: http://ieeexplore.ieee.org/document/8272354/

[46] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: https://www.semanticscholar.org/paper/Glove%3A-Global-Vectors-for-Word-Representation-Pennington-Socher/0825788b9b5a18e3dfea5b0af123b5e939a4f564

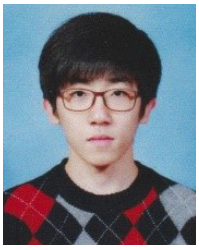
[47] H.-Y. Kim, J. Lee, N. Y. Yeo, M. Astrid, S.-I. Lee, and Y.-K. Kim, "CNN based sentence classification with semantic features using word clustering," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2018, pp. 484–488. [Online]. Available: https://ieeexplore.ieee.org/document/8539546/

[48] P. Perera and V. M. Patel, "Learning deep features for one-class classification," *IEEE Trans. Image Process.*, vol. 28, no. 11, pp. 5450–5463, Nov. 2019.

[49] K. Hempstalk and E. Frank, "Discriminating against new classes: One-class versus multi-class classification," in *Proc. Australas. Joint Conf. Artif. Intell.* Cham, Switzerland: Springer, 2008, pp. 325–336.

SYED FARHAN ALAM ZAIDI was born in Lahore, Pakistan, in 1993. He received the B.S. degree in information technology from the University of Education, Lahore, Pakistan, in 2014, and the M.S. degree in computer science from COMSATS University Islamabad, Pakistan, in 2017. He is currently pursuing the Ph.D. degree in computer science and engineering with Chung-Ang University, Seoul, South Korea. After completing B.S. degree, he worked as a Web Developer with a software development company in Pakistan. After completing M.S. degree, he involved with teaching as a Visiting Lecturer in various universities. Along with Ph.D. degree, he is also working as a Research Assistant with the Real-Time Software Engineering Laboratory. His research interests include software engineering based problems, natural language processing, deep/machine learning, data mining, image processing, and medical imaging.

FARAZ MALIK AWAN was born in Islamabad, Pakistan, in 1993. He received the B.S. degree in computer science from the COMSATS Institute of Information Technology, Pakistan, (now known as COMSATS University Islamabad, Pakistan), in 2015, and the M.S. degree in computer science and engineering with a major in system software from Chung-Ang University, Seoul, South Korea, in 2018. He is currently pursuing the Ph.D. degree with Telecom SudParis. After completing B.S. degree, he joined a German Development Agency, called Deutsche Gesellschaft für Internationale Zusammenarbeit (GIZ) in 2015 and started working as a Software Engineer. Along with M.S. degree, he worked as a Research Assistant with the Real-Time Software Engineering Laboratory. After completing M.S. degree, he moved to Paris, France, where he is also working as a Research Engineer with Telecom SudParis. His research interests include deep/machine learning focusing on smart cities, natural language processing, data mining, and big data.



MINSOO LEE was born in Seoul, South Korea, in 1995. He received the B.S. degree in computer science and engineering from Chung-Ang University, Seoul, in 2019, where he is currently pursuing the M.S. degree in computer science and engineering. His research interests include natural language processing, deep learning, and verification and validation of artificial intelligence software.

HONGUK WOO was born in Seoul, South Korea. He received the B.S. degree in computer science from Korea University, Seoul, in 1995, and the M.S. and Ph.D. degrees in computer sciences from The University of Texas at Austin, Austin, TX, USA, in 2002 and 2008, respectively. From 2008 to 2018, he worked with Samsung Research of Samsung Electronics as a Principal Engineer and the Vice President. Since 2018, he has been an Assistant Professor with the Department of Software, Sungkyunkwan University, Suwon, South Korea. He is the coauthor of more than 30 research articles and ten patents. His research interests include data-centric application, analytic monitoring and intelligence, and networked cyber-physical systems.

CHAN-GUN LEE was born in Seoul, South Korea, in 1972. He received the B.S. degree in computer engineering from Chung-Ang University, Seoul, in 1996, the M.S. degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, in 1998, and the Ph.D. degree in computer science from The University of Texas at Austin, Austin, TX, USA, in 2005. From 2005 to 2007, he was a Senior Software Engineer with Intel, Hillsboro, Oregon. Since 2007, he has been a Professor with the Department of Computer Science and Engineering, Chung-Ang University. He is the author of more than 30 articles and conference papers. His research interests include software engineering and real-time systems. He was a recipient of the Korea Foundation of Advanced Studies (KFAS) Fellowship from 1999 to 2005.