IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 15, 2020, accepted October 7, 2020, date of publication October 22, 2020, date of current version November 16, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3033045

Duplicate Bug Report Detection and Classification System Based on Deep Learning Technique

**ASHIMA KUKKAR[1], RAJNI MOHANA[1], YUGAL KUMAR[1],
ANAND NAYYAR[2,3], (Senior Member, IEEE),
MUHAMMAD BILAL[4], (Senior Member, IEEE),
AND KYUNG-SUP KWAK[5], (Life Senior Member, IEEE)**

[1]Department of Computer Science, Jaypee University of Information Technology, Wakanghat 173234, India
[2]Graduate School, Duy Tan University, Da Nang 550000, Vietnam
[3]Faculty of Information Technology, Duy Tan University, Da Nang 550000, Vietnam
[4]Computer and Electronics Systems Engineering Department, Hankuk University of Foreign Studies, Seoul 17035, South Korea
[5]Information and Communication Engineering, Inha University, Incheon 22212, South Korea

Corresponding authors: Muhammad Bilal (m.bilal@ieee.org) and Kyung-Sup Kwak (kskwak@inha.ac.kr)

This work was supported by the National Research Foundation of Korea funded by the Korean Government, Ministry of Science and ICT, under Grant NRF-2020R1A2B5B02002478.

ABSTRACT Duplicate bug report detection is a process of finding a duplicate bug report in the bug tracking system. This process is essential to avoid unnecessary work and rediscovery. In typical bug tracking systems, more than thousands of duplicate bug reports are reported every day. In turn, human cost, effort and time are increased. This makes it an important problem in the software management process. The solution is to automate the duplicate bug report detection system for reducing the manual effort, thus the productivity of triager's and developer's is increased. It also speeds up the process of software management as a result software maintenance cost is also reduced. However, existing systems are not quite accurate yet, in spite of these systems used various machine learning approaches. In this work, an automatic bug report detection and classification model is proposed using deep learning technique. The proposed system has three modules i.e. Preprocessing, Deep Learning Model and Duplicate Bug report Detection and Classification. Further, the proposed model used Convolutional Neural Network based deep learning model to extract relevant feature. These relevant features are used to determine the similar features of bug reports. Hence, the bug reports similarity is computers through these similar features. The performance of the proposed system is evaluated on six publicly available datasets using six performance metrics. It is noticed that the proposed system outperforms the existing systems by achieving an accuracy rate in the range of 85% to 99 % and recall@k rate in between 79%-94%. Moreover, the effectiveness of the proposed system is also measured on the cross training datasets of same and different domain. The proposed system achieves a good high accuracy rate for same domain data sets and low accuracy rate for different domain datasets.

INDEX TERMS Duplicate bug report detection, Siamese networks, natural language processing, deep learning, bug tracking system, software maintenance, software development, convolutional neural network, software engineering.

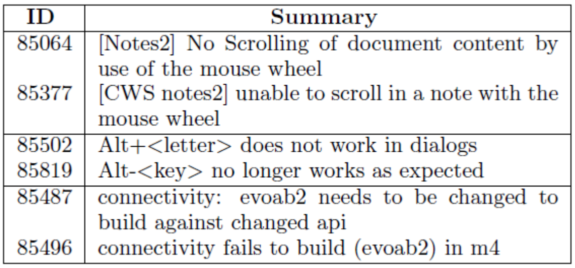
I. INTRODUCTION

The bug reporting is the important part of software maintenance, testing and development process. In today's agile, the bug tracking system (BST) is used by the tester or user to keep the track record on a bugging which is encountered during the usage of particular software [1]. The input to the BTS is a bug report. It also upholds the master list of bug reports. Usually, natural language is used to write the bug reports. The same bug report can be written in various ways by tester, developer and user who encounter the bug. It is because the vocabulary varies among tester, user and developer on their level of technical knowledge. The content of the bug report is later analyzed by the expert who has complete knowledge of the software called triager. The triager has two main tasks. Firstly, triager translates the wording of bug report into more technical language for a better understanding of the developer. Secondly, triager carries out the search process in the bug repository for potential duplicated bugs having the similar signature. Further, if the new bug

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

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

report is not duplicate then it is added to the master bug report list else, it is considered as a duplicate bug report. However, the filtering process of duplicate bug report requires extensive amount of manual effort, time and complete knowledge of bugs [2]. Previously study showed that near about 220 bug reports are submitted in BST per day in which 25-30% reports are duplicates [3]. The duplicate bug reports are assigned to different developers for resolving the bug, that results wasting of effort and time of developer. Therefore, to automate the duplicate bug report detection process is very productive. It reduces the human cost, effort and time. The reduction in the manual effort also increases the triager's and developer's productivity. It also speeds up the process of software management as a result the cost of software maintenance is also reduced. Further, triager need not to entertain the multiple bug reports on the same issues and have more information about every bug report, hence each bug can be fixed faster [4]. An example of duplicate bug report is illustrated in Figure 1. A bug report has two types of information i.e. structured and unstructured. The structured information contains the information about the component, severity, product, reported to date etc. of the bug. The unstructured information contains a natural language description of the bug, conservation between many developer and user to know the root cause and resolve the bug [5]. The description or conservation may be long or short. Therefore, the main challenges of automatic duplicate bug report detection are described below [6]:



ID	Summary
85064	[Notes2] No Scrolling of document content by use of the mouse wheel
85377	[CWS notes2] unable to scroll in a note with the mouse wheel
85502	Alt+<letter> does not work in dialogs
85819	Alt-<key> no longer works as expected
85487	connectivity: evoab2 needs to be changed to build against changed api
85496	connectivity fails to build (evoab2) in m4

FIGURE 1. An example of duplicate bug reports.

- The content of the bug report is in natural language text. These bug reports may describe the same bug with different words. So, inherent variability of natural language to write the same bug report due to the vast and ambiguous nature of language, that results the difficulty in finding the similar bug report 7].
- The quantity of a bug report is huge and writing quality is very poor, mostly presence of noise text, incorrect information, inadequate and missing of important information that produce further technical challenges in resolving the bug [8], [9].

A. MOTIVATION

The basic aim of duplicate bug report detection and classification systems is to detect the similar bug report. The detection of duplicate bug report can be considered as problem of classification. The dataset of duplicate bug report is created utilizing the bug report content. The semantic relationship between the content of the bug report is identified and this relationship is used to determine the relevant (important) features for detecting the duplicate bug report. In literature, the various machine learning and feature extraction techniques are used to extract the relevant features for developing an automatic duplicate bug report detection and classification system [10]–[12]. In spite of the substantial volume of work, the accuracy of existing duplicate detection system is comparatively low because of the following reasons: -

- Ignore the semantic relationship between the bug report content.
- All latent features are not captured due to linear equation and fixed length of polynomial terms.
- Not understand the interpretation of slightly variant sentences.
- Sentences of bug reports are examined as word stacks with unordered word sequence.

For example, in Figure 1, the sentence in Bug ID 85502 reads "does not work in dialogs" and in Bug ID 85819 "no longer works as expected". Both sentences have same meaning but the way of writing is different. The previous machine learning and feature extraction techniques do not capture the latent features, semantic relationship of text, ignore the word sequence and consider these sentences as different sentences. Therefore, there is still much room for improvement in the automatic duplicate bug report detection system. To overcome the above mentioned issues, a deep learning [13] based feature learning structure is used to identify the important features for predicting the duplicate bug. Deep Learning (DL) techniques have gained too much attention from the last few years. Many researchers have found, deep learning has been established successfully in many other domains like text and image classification [14], [15], speech recognition [16], extracting raw signals [17] and natural language processing (NLP) [18]. In the NLP domain, the deep learning techniques work better than traditional machine learning techniques [19]. Further, DL based on Convolutional Neural Network (CCN) technique is widely utilized in the NLP field [22]. CNN technique has the ability of capturing the nonlinear mapping of features between the input and output. It can also learn the sematic relationship between the text and extract the relevant features automatically via the multi-layer structure of network. It is better than predefined sets of features and manual feature extraction.

B. CONTRIBUTION

The basic aim of this paper is to leverage the deep learning technique to examine the bug repositories for extracting the relevant features that are useful in classifying the duplicate bug reports. The Siamese variant of CNN is employed to extract relevant features from structured and unstructured information (content of bug report) [20]. The generality of the proposed system is also improved by using different kernel with various filters. The proposed system is also self- capable of learning the representation of features to compute the bug reports similarity and categorize the duplicate bug report from non-duplicate bug report without the interference of manual feature engineering. This paper addresses the following significant contributions:

1. The CNN based deep learning model is proposed for recognizing the duplicate or similar bug reports by means of textual content available in the bug repository.
2. The proposed system employed Siamese structure, trained on cross-entropy loss objective function to compute the bug reports similarity.
3. The detailed experimental results on six large published datasets are presented [21], [32]. The average accuracy of the proposed system is in the range of 85% to 99 %, and recall @20 is in between 79% to 94%, Hence, the performance of the proposed system outperforms the previous reported systems.
4. The proposed system is also tested on cross training datasets and achieves a good high accuracy rate for same domain datasets whereas, low accuracy rate for different domain datasets.

This research paper is organized as follows. The related work is illustrated in Section 2. The proposed duplicate bug report detection and classification system is described in Section 3. Experimental results and discussions are represented in Section 4. The conclusion and future work are discussed in Section 5.

II. RELATED WORK

The duplicate bug report detection is very time consuming and boring job because of different writing style of the large number of reported bug reports. Thus, there is a need to automate the process of duplicate bug report detection to avoid the manual effort. The challenging issue is to train the discriminative model for verifying the new reported bug report is duplicate or not. From the past few years a variety of approaches have been proposed on automatic identification of duplicate bug report [5], [11], [22]–[40]. The detail of these approaches is presented below:

To find the duplicity, in 2007, Kim *et al.* [22] designed a duplicate bug report approach using the vector space model for feature vectors and cosine similarity parameter to measure the similarity between the features of new and existing bug reports. The results showed that about 40% duplicate bug reports are found. The another duplicate bug detection approach is implemented by Jalbert *et al.* in 2008 [23], by utilizing the knowledge of surface feature like severity, summaries and daily comments. These features are fed into a proposed graph clustering based classifier to find the textual bug reports similarity. The proposed approach minimizes the development cost by detecting the 8% duplicate bug reports. Jalbert *et al.* [24] measured the similarity between natural language content and execution traces of bug reports using the vector space model (VSM) and cosine similarity function. It is noticed from the experimental results that the hybrid information detects more duplicate bug rather than only using natural language content by achieving the improvement of 25% on Firefox project. Further, in 2009, Wang *et al.* [5] proposed a duplicate bug detection model using information techniques and information of natural language to extract and calculate the textual similarity in bug repositories. The experimental results demonstrated that proposed model improves the result of existing by 58% on Open office dataset.

To address the effort of triager or developer for manually identifying the similar bug reports, Wang *et al.* [25] deeply studied various software projects. These projects are Eclipse, Bugzilla and SeaMonkey. The studied showed that a single triager can detect the 50% duplicate bug reports in 24 hours. A duplicate bug classification model is developed by focusing this study. The proposed model achieves 68% accuracy rate and 60% recall rate on all datasets. To study the performance of various information retrieval and topic based model for detecting the duplicate bugs, Bettenburg *et al.* [26] conducted an experiment by applying Log Entropy, LDA, LSI and Random Projections approaches. These approaches are tested on Firefox and Eclipse project using recall performance parameter. Authors claimed that information retrieval models achieve the higher recall rate i.e. 60% for Eclipse and 58% for Firefox rather than topic based models. Developers also suggested that characteristics and domain of project show a significant role in improving the information retrieval models.

In 2010, Kaushik *et al.* [11] implemented a discriminative model based on supervised learning. The working of the proposed model is categorized into two modules. In the first module, the idfsum, idfdesc and idf weights are calculated to find the text similarity of bug reports. In the second module, computes a similarity score fed into the support vector machine to classify the bug report into duplicate or non-duplicate classes. The proposed approach is tested over eclipse and open office datasets. These datasets are divided into three parts. The first part contained the summaries; the second contained the description and third one contained the collection of summaries and description of bug reports. It is noticed that performance of the proposed model outperforms the existing model by achieving 65% recall rate for all datasets. In the continuation of their work, Sun *et al.* [27] improves recall rate by extending a REP method with the help of BM25F similarity measure approach. The proposed method calculated the similarity between textual and non-textual information like comments, component, version and product details. The performance of proposed method is evaluated on the OpenOffice, Mozilla and Eclipse bug repositories. It is observed that proposed method detected more duplicate bug reports rather than SVM based detection model. To detect the duplicate bugs, Sun *et al.* [28] applied the extension of BM25F in REP method on the hybrid information dataset. Author also represented a new documentation of relative similarity to find the significance of similarity between bug reports. The accuracy rate of proposed method improves by 160% over Sun *et al.* model. A new text representation is presented by Tian *et al.* [29] in 2010 called character level representation. They designed an n- gram character model to capture the discriminating features of bug report for detecting the duplicate bugs. The proposed approach is applied on 213000 reports of Eclipse project and achieves the recall rate for randomly selected bug reports is up to 66.00%. In 2012 Sureka *et al.* [30] extended the work of Sun *et al.* by combining Latent Dirichlet Allocation (LDA) with BM25F. The objective of this work is to calculate the similarity between the text and topic of bug reports. It is noticed that proposed model increased the recall rate by 20% from Sun *et al.* model on Eclipse and Open office dataset. To identify the stack traces in bug reports, Nguyen *et al.* [31] proposed a duplicate bug detection approach. The proposed approach is divided into three parts. In first part, stack traces of bug report is transformed into set of methods. In second part, the term frequency is used to measure the similarity between these sets. In third part, sets are ranked according to their frequency scores. The effectiveness of proposed approach is validated using Eclipse dataset. It is observed that proposed approach can provide better results with fewer requirement. To identify the duplicate bugs in the bug repository, Lerch *et al.* [32] designed an approach based on crash graph. In proposed approach, the graph of same bug reports stacks traces is created to combine the multiple stack traces. Further, these graphs are compared with new bug report graphs for detecting the duplicates using graph similarity functions. The effectiveness of proposed approach is evaluated on Firefox and Gnome projects using precision and recall parameters. It is observed that the recall rate of proposed approach is 67.65% and 71% for Firefox and Gnome project respectively. To compute the similarity between textual and categorical bug report text, Alipour *et al.* [33] created six-word list from bug reports. The word lists contained the words based on usability, portability, functionality, maintainability, reliability and efficiency. Author also developed a duplicate bug report detection model. This model has two parts. In first part, BM25F is applied to measure the textual similarity score. In second part, various machine learning algorithms such as Logic Regression, Naive Bayes, K-nearest neighbor and C4.5 are used to classify the duplicate bugs. perform the detection task. It is observed that the C4.5 algorithm outperforms the other machine learning algorithms by achieving 92% recall rate for Android ecosystem project. Author claimed that domain knowledge plays an important role in detecting the duplicate bugs. In the continuation of their work Hindle *et al.* and Hindle *et al.* [34] extended the work of Sun *et al.* by adding the comparison of software contexts such as topics of software (extracted by LDA approach), non-functional requirements and software architecture. The proposed approach is tested over four software projects such as Open office, Mozilla, Eclipse and Android. It is noticed that the proposed approach outperforms the Sun *et al.* approach by improving the accuracy rate between 1.5%-11.5%. Authors also claimed that comparison of software contexts improves the performance of machine-learning classifiers for detecting the duplicate bugs. In 2015, Aggarwal *et al.* [35] proposed a new method by measuring similarity between the words of software engineering (SE) textbooks and protect documentation called software literature content approach using BM25F. This proposed approach showed the same results as but consume less time and effort. In the continuation of their work, Aggarwal *et al.* extended the previous work by extracting the domain-specific features with project-specific features [36]. Author also introduced context hierarchy for capturing the SE knowledge to increase the performance gains. The proposed approach is tested over Eclipse, Open office and Mozilla datasets using kappa and accuracy parameters. It is observed from the experimental results that context hierarchy improves the performance of proposed approach by achieving 3.8% accuracy rate and 10.8% kappa score for each dataset. To detect the duplicate bug reports, Deshmukh *et al.* [37] developed a retrieval and classification model using deep learning approach. In this model, Long Short Term Memory (LSTM) is applied on short description and CNN is applied on long description of bug report to detect the similarity between the text. The purposed model is tested on Open office, Eclipse and NetBeans datasets. The proposed model achieves 90% accuracy rate and 80% recall rate for all datasets. To identify the duplicate bug reports, Koopaei *et al.* [38] designed a model using n-gram and Markov approach. In the proposed model, the stack traces of duplicate bug reports groups are collected with n-gram and Markov approach. Further, the automaton of stack traces of duplicate group is compared with the stack traces of new bug report. The proposed approach achieved better results than state of art approaches. To detect the duplicate bugs, Sabor *et al.* [39] utilized name of package in stack traces in Eclipse dataset. The objective of this paper is to minimize the computation time for large stack traces. The working of proposed approach is divided into three parts. In first part, n–grams features are extracted from orders of name packages. In second part, extracted features are used to measure the similarity between stack traces of new and historical bug reports. Author claimed that proposed approach minimizes the computation time for large stack traces. In 2019, Ebrahimi *et al.* [40] improved the automata using Hidden Markov Model. The effectiveness of proposed approach is tested over Firefox and Gnome projects using recall parameter. it is observed that the recall rate of proposed approach is 76.43% for Firefox and 71.57% for Gnome dataset.

III. PROPOSED DUPLICATE BUG REPORT DETECTION AND CLASSIFICATION SYSTEM

The duplicate bug report detection involves variability and ambiguity of linguistic expression. Bug Report contains sentences, which are further composed of words. The combination of words, forms, clauses and phrases. The examination of these sentences helps to understand its meaning. The deep neural network model helps in capturing the word relationship from the multiple points of view. A new duplicate bug report detection model is designed by keeping these phenomena in mind. Figure 2 illustrates the framework of

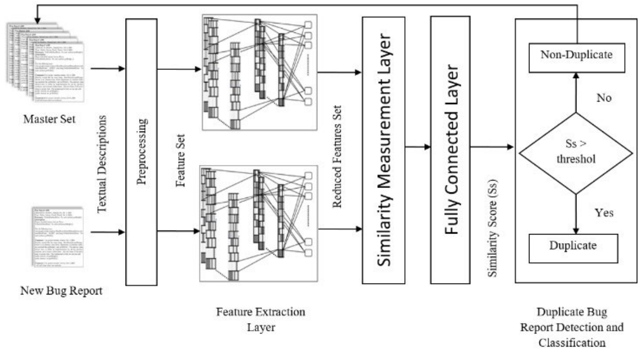


FIGURE 2. Framework of proposed duplicate bug report detection and classification system.

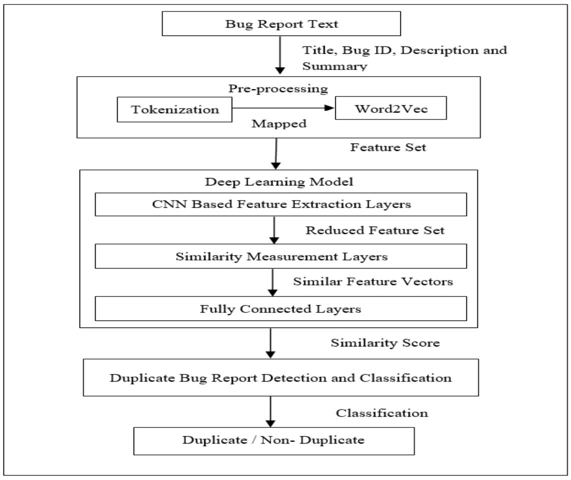


FIGURE 3. Working of proposed duplicate bug report detection and classification system based on deep learning technique.

the proposed system. The working of the proposed system is represented in Figure 3. The proposed system comprises of three modules:

- Preprocessing: The basic aim of this module is to convert each term of the bug report into more manageable representation, remove the unwanted terms from the bug reports.
- Deep Learning Model: In this module deep learning based model is proposed for extracting the semantic and syntactic relationship of words for the textual similarity measurement between bug reports.
- CNN based Feature Extraction Layer: CNN technique is used to extract the relevant features. It has various convolution filters that capture the local features and examines all words of the bug reports from multiple prospective.
- Similarity Measurement Layer: This layer contains similarity measurement metric, which compare the sentence representation of the bug reports.
- Fully Connected Layer: This layer computes the similarity score of the sentences of bug reports.
- Duplicate Bug report Detection and Classification: This module classifies the duplicate bug report from non-duplicate bug report based on the final decision based on the similarity scores.

A. PREPROCESSING

The input of this module is the bug reports. It has variety of fields like status, component, title, bug ID description, severity, version, priority, Summary etc. In this research paper, the bug report title, bug ID, summary and description are used for duplicate bug report detection. The unwanted words like non alpha and numeric characters are removed from the bug report text. These unwanted words can degrade the learning performance of the detection system. The objective of preprocessing module is to reduce the bug report features space. As, result the effort cost of triager is minimized [15]. Next, the sentences are broken down into the words called token. After that, each word is mapped to its corresponding embedding by Word2Vec in the dictionary [41]. The output of this step sets of feature vectors that represent each word in the bug report. Further, the proposed CNN model uses pre-trained word embedding for extracting the relevant features.

B. DEEP LEARNING MODEL (DLM)

This subsection illustrates the proposed DLM for duplicate bug report detection. DLM is applied to determine the set of reduced features from given set of bug report features. The aim of DLM is to identify more relevant (important) features from the bug report feature set and measure similarity between text of bug reports. The deep learning feature extraction process is consisted of five layers such as Input, Convolutional, Activation, Max pooling. The textual similarity measurement process has two layers i.e. Similarity Measurement and Fully connected layer. The graphic representation of deep learning technique based feature extraction and similarity measurement is illustrated in Figure 4.

1) CNN BASED FEATURE EXTRACTION LAYER

In this layer, CNN model used Siamese structure to generate the semantic and syntactic relationship of each word of the bug report content from its preceding and following words. The structure has two networks, one extract the relevant feature of one bug report and another extract the features of other, but the feature weight tied down with each feature. Further the CNN model uses pre-trained word embedding for this word representation.

The input layer gets the stream of feature vectors (words). These words interpret as the temporal sequence where close words are correlated. Let assume, bug report contains n number of sentences $BR = (sbr_1, sbr_2, \ldots .sbr_k \ldots sbr_n)$. Each sentence of bug report $sbr \in S^{l \times d}$ is the sequence with l word input represented by d - dimensional word embedding's and $sbr_k \in S^d$ represents the embedding of k^{th} word in the sequence. Further, $sbr_{k:h}$ is the concatenation of word

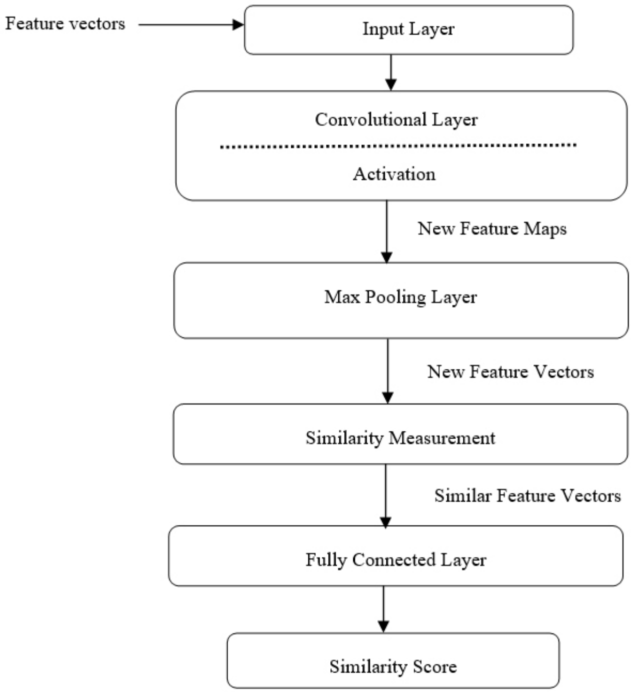


FIGURE 4. **Proposed deep learning technique based feature extraction and similarity measurement.**

embedding's from k^{th} to h^{th} words, $sbr_k^{[i]}$ represents the k^{th} word vector with i^{th} dimension and the $sbr_{k:h}^{[i]}$ is the vector of the k^{th} to h^{th} words having the i^{th} dimension

The convolution layer contains the filter as $\langle w, h, b, a\rangle$, where w is the sliding window, $h \in S^{w\times d}$ is the vector weight, $b \in S$ is the bias, a is the tanh activation function. When filter f is enforced to sentence of bug report sbr then the inner product is figure out between h and each word embedding window of length w in sbr, after that the bias value is added and sigmoid function is applied. This provides an output vector $f_{out} \in S^{1+l-w}$ where ith entry is expressed as

$$f_{out}\,[i] = a\,(h.sbr_{i:i+w-1} + b)\,, \qquad \text{where } i \in [1, 1+l-w] \tag{1}$$

This filter matches the word sequence regions and considers the entirety of each word embedding at each position as shown in Figure 3. These are called holistic filters. The information is also target at finer granularity by the pre-defined filters which are included separately for every dimension of input word embedding's as described in Figure 3. Suppose f^k for every dimension k of word embedding's, where $h \in S^w$ and $f^k = \langle w, h_{fk}, b_{fk}, a_{fk}, \rangle$ gives the output vector $f_{out}^k \in S^{1+l-w}$ for every k dimension where ith entry (for i $\in [1, 1+l-w]$) is expressed as

$$f_{out}^k\,[i] = a_{fk}\left(h_{fk}.sbr^k_{i:i+w-1} + b_{fk}\right) \tag{2}$$

The utilization of word embedding's in both ways helps in extracting richer sentence modeling. Furthermore, by updating the word embedding in training phase capture more distinct information. In this research, the convolution layer is considered as set of filters that share same filter type (holistic and predefined), activation function and width of window size that is preferred by modeler and filter weights (h and b) are learned.

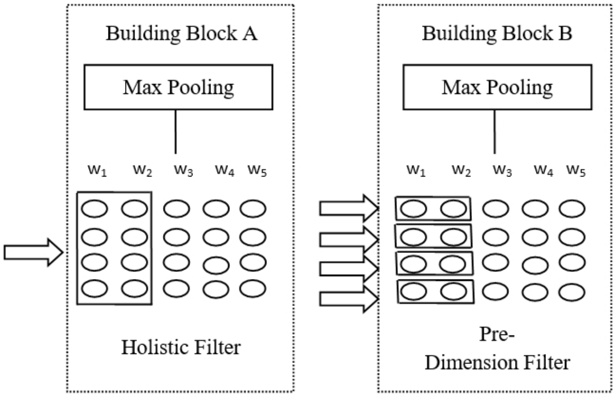


FIGURE 5. **Block Representation with Holistic and Pre-Dimension Filters (In Block A, holistic filters match the entire feature vector on the hand in Block B, pre-dimension filter independently matches each dimension of word embedding).**

Further, the output vector f_{out} is transformed into the scalar for consequent use by the pooling method. In this study, the max pooling is operated across the entries of f_{out} and gives the max value. As shown in Figure 5, the two building blocks are defined to set the groups, $\mathrm{B_A}$ and $\mathrm{B_B}$. The instance of $\mathrm{B_A}$ contains the three holistic convolution layers with same filter, pooling and window size. For a $\mathrm{B_A}$ $(w_a, pooling, sbr)$ with the number of filters nf_A and convolution layer, the output p_{out_A} is defined as the vector length nf_A where h is the entry:

$$p_{out_A}\,[h] = \max\_pooling(outf_h) \tag{3}$$

where f_h are the filters and output of each filter vector after applying the pooling function in block the $\mathrm{B_A}$ $(*)$.

The block $\mathrm{B_B}$ contains the two groups of convolution layers with the pre- dimensional filters Dim, max pooling and window size. It operates on the particular dimensions of the word embedding. The output p_{out_B} of block $\mathrm{B_B}$ ($w_b, pooling, sbr$) is defined as $Dim\times nf_B$ where $[k]\,[h]$ is the entry:

$$p_{out_B}\,[k]\,[h] = \max\_pooling(outf_h^{[k]}). \tag{4}$$

where, $f_h^{[k]}$ is defined as h filter with k dimensions. There is no pooling across the multiple filters in the block or layers. Each pooling layers perform independently on the matches of single filter.

2) SIMILARITY MEASUREMENT LAYER

This layer computes the similarity between the sentences of two bug reports by extracting and comparing all the feature maps of bug reports. The structural comparison is carried on the representation of sentences of two bug report in parallel over particular regions as shown in Figure 6. The following aspect is considered for comparing the local regions: whether

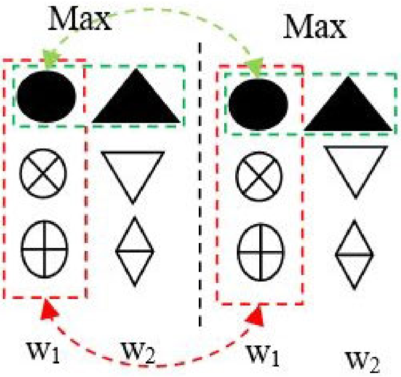


FIGURE 6. Example of comparison of local region of sentences representation of two bug reports that uses Block A only. The algorithm 1 (horizontal comparison) is shown with green dotted line and algorithm 2 (vertical comparison) is presented with red dotted lines. Every sentence representation in the bug report utilizes size of window w1and w2.with max pooling and $nf_B = 1$ filters.

they belong to same building block and same filter of the convolution layer. The sentence representation of two local regions is computes by the cosine similarity metric [42]. The expression is described below:

$$Cos\left(BR_{new}, BR_{src}\right) = \frac{\overrightarrow{BR_{new}}.\overrightarrow{BR_{src}}}{\left\|\overrightarrow{BR_{new}}\right\| \left\|\overrightarrow{BR_{src}}\right\|} \tag{5}$$

Let $\overrightarrow{BR_{new}}$ and $\overrightarrow{BR_{src}}$ be the representations of vector space of the new and source sentences of bug reports respectively. The denominator represents the dot product of Euclidean norms of these bug reports vectors and numerator represents the product of these vectors. The two algorithms are designed to compare the sentences of two bug reports. The Algorithm 1 works on the $\mathrm{B_A}$ output and Algorithm 2 works on both $\mathrm{B_A}$ and $\mathrm{B_B}$ outputs, focusing on the regions which share same blocks with different filters. Given two bug reports having n sentences sbr_n^1 and sbr_n^2. The window size of $\mathrm{B_A}$ and $\mathrm{B_B}$ is set to 1. The number of filters nf_A by n+1 matrix is represented as $regMatrix*$. Further assume that each group of $block*$ provides the output p_{out_*}. At last the output features are accumulated as final output $feat$.

Algorithm 1 Horizontal Comparison

```
Step 1:  for pooling p = max do
Step 2:      for with w = 1 do
Step 3:          $regMatrix_1[*][w] = BA\,(w, p, sbr_n^1)$
Step 4:          $regMatrix_2[*][w] = BA\,(w, p, sbr_n^2)$
Step 5:      end for
Step 6:      for $i = 1\ldots..\,nf_A$ do
Step 7:          $feat_h = \mathrm{Cos}\,(regMatrix_1[i], regMatrix_2[i])$
Step 8:          accumulate $feat_h$
Step 9:      end for
Step 10: end for
```

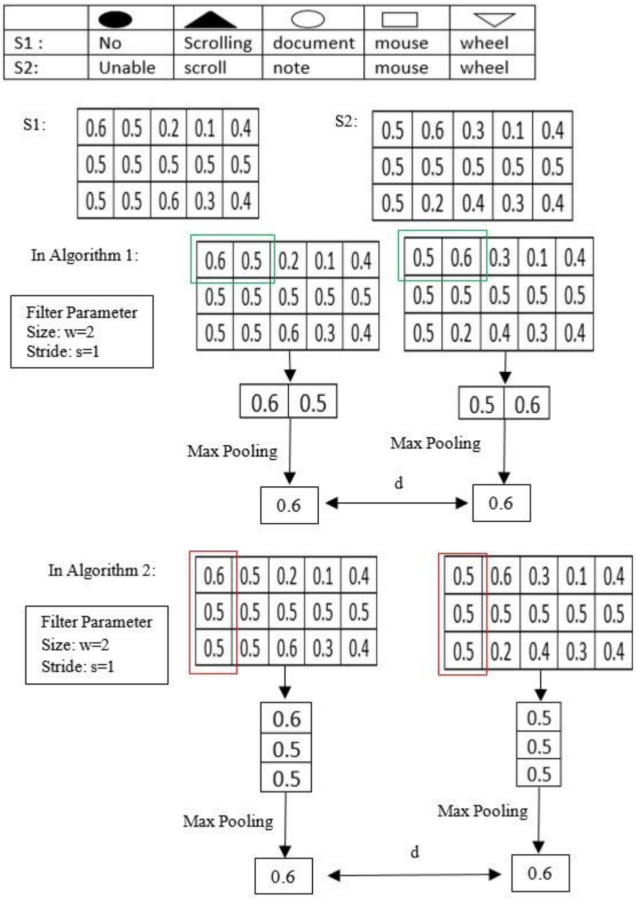
Algorithm 2 Vertical Comparison

```
Step 1:  for pooling p = max do
Step 2:      for with $w_1 = 1$ do
Step 3:          $p_{out_{1A}} = \mathrm{B_A}\,(w_1, p, sbr_n^1)$
Step 4:          for with $w_2 = 1$ do
Step 5:              $p_{out_{2A}} = \mathrm{B_A}\,(w_2, p, sbr_n^2)$
Step 6:              $feat_a = \mathrm{Cos}\,(p_{out_{1A}}, p_{out_{2A}})$
Step 7:              accumulate $feat_a$
Step 8:          end for
Step 9:      end for
Step 10:     for with $w_1 = 1$ do
Step 11:         $p_{out_{1B}} = \mathrm{B_B}\,(w_1, p, sbr_n^1)$
Step 12:         $p_{out_{2B}} = \mathrm{B_B}\,(w_2, p, sbr_n^2)$
Step 13:         for $i = 1\ldots..\,nf_B$ do
Step 14:             $feat_B = \mathrm{Cos}\,(p_{out_{1B}}, p_{out_{2B}})$
Step 15:             accumulate $feat_B$
Step 16:         end for
Step 17:     end for
Step 18: end for
```

3) FULLY CONNECTED LAYER

The output similarity feature vectors ($feat_B$) of the similarity measurement layer fed into fully connected layer. In this layer similarity of the sentences of bug reports is computed. This layer has stack of two liner layers with the tanh activation function in between and followed by log softmax function as final output as similarity score of the sentences of bug reports.

Example: Two input sentences from the two bug reports are processed in parallel by identical CNN based feature extraction layer. The output of the CNN based feature extraction layer is the sentence representations of the bug reports in the form of matrix. The sentence representations are compared by the similarity measurement layer using cosine similarity metric. The similarity features are then fed into the fully-connected layer for computing the similarity score. Further, the graphical representation of working example is illustrated in Figure 7 to show how algorithms 1 and 2 compare outputs of blockA only. The sentence representations of the bug reports are arranged into the form of sentence matrices as in Figures 6 and 7. In Algorithms 1 and 2, local regions of the two matrices are compared by rows and columns. Each equal-sized max group is extracted as a vector and is compared to the corresponding one for the other sentence by using cosine distance d as illustrated in equation 5. This process is repeated for all rows and comparisons are shown in green solid lines, as performed by Algorithm 1. In Algorithm 2. each column of the sentence matrix is compared with the column of the other sentence after the max pooling operation. This is shown in red dotted lines in the Figure 7 and listed in lines 2 to 9 in Algorithm 2.



	●	▲	○	▭	▽
S1 :	No	Scrolling	document	mouse	wheel
S2:	Unable	scroll	note	mouse	wheel

FIGURE 7. Example of comparison of local region of sentences representation of two bug reports that uses Block A only.

C. DUPLICATE BUG REPORT DETECTION AND CLASSIFICATION

In this module the final textual similarity score of new and existing bug report is compared with the specific threshold. Firstly, the similarity scores of all sentences are combined to get the bug report final similarity score. If the textual similarity between the new bug report and master bug report is larger than a specific threshold, then the new bug otherwise non- duplicate and added to the master set.

IV. EXPERIMENT RESULTS AND DISCUSSIONS

The experiment results of proposed duplicate bug report detection and classification system is demonstrated in this section. The proposed system is applied on six public available datasets such as Mozilla, Eclipse, NetBeans, Gnome, Open office and Firefox. F-measure, Precision, Accuracy, Recall@k and Recall metrics are used to evaluate the performance of proposed system. A system having window 10, NVIDIA GEFORCE GPU, Intel Core i5 processor, CPU @ 2.70GHz and 8GB RAM is utilized to implement the proposed system.

Although, Keras API in TensorFlow is used to implement deep learning based similarity measure and feature extraction technique [41].

Algorithm 3 Duplicate Bug Report Detection and Classification

```
Input: Master Bug Report Set MBr, New Bug Report NBr, Threshold thres
Output A list of Duplicate Bug Reports DRr
Step 1:  for each bug report Br in MBr and NBr do
Step 2:      Extract the word embedding vectors using Word2Vecand fed into layer CNN for extracting the feature vectors of the textual description of bug reports;
Step 3:  end for
Step 4:  for each bug report Br in MBr do
Step 5:      for local region comparison lrc [horizontal, vertical] do
Step 6:          Compute the cosine similarity between the local regions of MBr and NBr and denote as Ss
Step 7:      end for
Step 8:      if Ss > thres
Step 9:          than Br classified as duplicate and add in DRr
Step 10:     else
Step 11:         Br classified as Non-duplicate and add in MBr
Step 12:     end if
Step 13: end for
```

TABLE 1. Characteristics of duplicate bug report datasets utilized in the experiments.

Datasets	Total Number of Bugs	Duplicate Bug Reports
Mozilla	700000	140000
Eclipse	110181	29037
NetBeans	148343	30769
Gnome	4600	789
Open office	111121	19785
Firefox	2883	545
Combined	1003197	220925

A. DATASET

In this work, the Mozilla, Eclipse, NetBeans, Gnome, Open office and Firefox open source datasets are adopted, which are created and published by Lazar *et al.* [21] and Lerch *et al.* [32] (https://bugs.eclipse.org/bugs/, https://github.com/ logpai/bugrepo). The combined bug report dataset is also created by combining all the datasets and named as 'Combined'. The characteristics of these datasets are illustrated in Table 1.

B. TRAINING

The objective of proposed system is to figure out the similarity between the two bug reports. The procedure ought to be learned in such manner that it gives high similarity score for duplicate bug reports and low similarity score for non-duplicate bug reports. This concept is formulated as cross-entropy training loss [15] objective expressed as below:

$$E = -\frac{1}{n}\sum\nolimits_{i=1}^{n}\left[L^{i}logL_{o}^{i} + \left(1-L^{i}\right) log \left(1-L_{o}^{i}\right)\right] \quad (6)$$

where L_o^i, represents the output result of i-th neuron in the output layer. L^i represents the results of corresponding target. The back propagation and stochastic gradient decent [42] is used to minimizing the loss averaged across the training samples. This supervised training helps in finding the latent and discriminative features. Further the bug report dataset is divided into train and test pairs as shown in Table 2.

TABLE 2. Train and test bug report pairs of datasets.

Datasets	Test	Train
Mozilla	700000	264586
Eclipse	88144	22037
NetBeans	118674	29669
Gnome	4600	1345
Open office	88896	22225
Firefox	2883	1455
Combined	1003197	793216

C. PARAMETER SETTINGS

The Word2Vec word embedding is used to represent part of speech tags and words. It is trained on the Google news dataset. The number of holistic filters are equal to the input word embedding. Therefore $nf_A = 300$. The number of per- dimension filters are $nf_B = 20$, so total filters equal to $20*Dim$. The learning rate and dropout of proposed algorithm is equal to 0.01 and 0.5. Tanh and softmax are used as activation function. Further, 4 convolution layers are utilized with 150 hidden units in the fully connected layer.

D. PERFORMANCE METRICS

In this subsection five performance metrics are used to compute the performance of proposed duplicate bug report detection and classification system as listed below [43].

- Accuracy: It specifies the correctly classified duplicate and non-duplicate bug reports. the accuracy of proposed system is stated using equation 7.

$$\text{Accuracy} = \frac{\text{P}+\text{N}}{\text{P}+\text{N}+\text{F}+\text{G}} \quad (7)$$

In equation 7, P denotes duplicate bug reports correctly classified, N represents non-duplicate bug reports correctly classified, F denotes non-duplicate bug reports incorrectly classified as duplicate bug reports and G denotes duplicate bug reports incorrectly classified as non-duplicate bug reports.

- Precision: It specifies duplicate or non-duplicate bug reports that can be classified as actually duplicate or non-duplicate. The another name of precision is Type-1 error. Precision is computed using equation 8.

$$\text{Precision} = \frac{\text{P}}{\text{P}+\text{F}} \quad (8)$$

- Recall: This metrics indicates actual duplicate or non-duplicate bug reports among all bug reports. Recall metric is also known as Type-2 error and stated as equation 9.

$$\text{Recall} = \frac{\text{P}}{\text{P}+\text{G}} \quad (9)$$

- F-measure: It is demonstrated as expression of precision and recall. It only takes the positive bug reports either classified as negative or positive. Equation 10 is used to compute the F-measure.

$$\text{F}-\text{measure} = \frac{2\text{P}}{2\text{P}+\text{P}+\text{G}} \quad (10)$$

- Recall@k: It is presented as the percentage of duplicate bug reports found in top list size k bug. This metric is calculated using equation 11.

$$\text{Recall@k} = \frac{U_{classified@k}}{U_{Total}} \quad (11)$$

In the equation 11, $U_{classified@k}$ denotes the number of correctly classified duplicate and no duplicate bug reports and U_{Total} is the total number of bug reports in the dataset.

E. RESULTS AND DISCUSSION

This subsection represents the experimental results of proposed system. The results of proposed system are compared with eight existing duplicate bug report detection systems. At last research questions are answered to know the effectiveness of proposed system over existing automated duplicated bug report detection system.

1) RESULT OF PROPOSED SYSTEM

The experimental results of proposed system are demonstrated in this subsection. To predict the bug is duplicate or not, the proposed system based on deep learning model is used. The six datasets and five performance metrics are adopted to evaluate a performance of duplicate bug report detection and classification system as mentioned above. The proposed system adopted the binary classification schema. The experimental result of proposed system using seven datasets is illustrated in Table 3. It can be seen that proposed system effectively computes the accuracy, precision, recall@k, recall and f-measure rates for each duplicate and non-duplicate bug report of each dataset. It can be seen from table, precision of proposed system for NetBeans, Eclipse, Open office, Gnome, Mozilla, Combined and Firefox datasets is 82.90%, 97.23%, 97.44%, 86.16%, 98.70%, 94.35% and 8024% respectively. The recall rate of NetBeans, Mozilla, Eclipse, Open office, Gnome, Combined and Firefox datasets obtained by proposed system is 80.23%, 98.42%, 97.04%, 95.34%, 83.35% 96.34% and 81.23% respectively. The F-measure of NetBeans, Eclipse, Open office, Mozilla, Gnome, Combined and Firefox datasets is 81.54%, 97.24%, 96.45%, 98.89%, 84.73%, 97.44% and

TABLE 3. Experimental results of proposed duplicate bug report detection and classification system using seven datasets.

Datasets	Performance Metrics			
	Precision (%)	Recall (%)	F-measure (%)	Accuracy (%)
Mozilla	98.70	98.42	98.89	99.38
Eclipse	97.23	97.04	97.24	98.33
NetBeans	82.90	80.23	81.54	85.45
Open office	97.44	95.34	96.45	95.24
Firefox	80.24	81.23	83.45	83.45
Gnome	86.16	83.35	84.73	83.00
Combined	94.35	96.34	97.44	96.34

TABLE 4. Experimental results of proposed system based on recall@20 metric of seven datasets.

Datasets	Performance Metrics		
	Recall@5	Recall@10	Recall@20
Mozilla	88.04%	91.48%	93.43%
Eclipse	88.93%	90.37%	93.93%
NetBeans	76.43%	80.54%	83.86%
Open office	83.43%	85.37%	88.21%
Firefox	77.67%	81.95%	83.18%
Gnome	73.86%	77.01%	79.95%
Combined	86.04%	88.71%	92.56%

83.45% respectively. The accuracy rate of proposed system is 99.38%, 85.45%, 98.33%, 95.24%, 83%, 96.34% and 83.45% for Mozilla, NetBeans, Eclipse, Open office, Gnome, Combined and Firefox datasets respectively. It can be seen that Mozilla open source dataset attains greater f-measure, recall, accuracy and precision rate as compared to other datasets. Therefore, it can say that proposed detection and classification system is efficient and capable for duplicate bug report detection and classification. Further, recall@1, recall@5, recall@10 and recall@20 metrics are utilized to determine the performance of proposed system as shown in Table 4. The recall@20 rate of proposed system is higher than recall@5 and recall@10. Further, the eclipse dataset achieves higher recall@20 i.e. 93.93, whereas, Gnome dataset achieves low recall@20 i.e. 79.75. The Mozilla dataset achieves higher recall@10 rate i.e. 91.48 and proposed system gets low recall@10 rate i.e. 80.54 for NetBeans dataset. Eclipse and Gnome datasets exhibits worst recall@5 rate among all datasets.

2) COMPARISON WITH TRADITIONAL DUPLICATE BUG REPORT DETECTION SYSTEMS

In this section, experimental results of proposed duplicate bug report detection system are compared with traditional systems based on Mozilla, NetBeans, Eclipse, Open office, Gnome, Combined and Firefox datasets. These systems are proposed by Sun *et al.*, Nguyen *et al.*, Aggarwal *et al.*, Hindle *et al.*, Deshmubk *et al.*, Ebrahimi *et al.* and Kim *et al.* [11], [27], [30], [32], [34], [36], [37], [40]. Sun *et al.* proposed two duplicate bug report detection system using discriminative approach and BM25F function with retrieval function [11], [27]. Nguyen *et al.* developed detection system using the combination of topic modeling and information retrieval [30]. Kim *et al.* designed a duplicate bug report detection system using crash graph approach [32] and Ebrahimi *et al.* developed system using HMM based approach [40]. Deshmukh *et al.* proposed a retrieval system for detecting duplicate bugs using deep learning techniques [37]. Aggarwal *et al.* implemented a detection system utilizing domain knowledge with software engineering [36]. Hindle *et al.* used contextual approach for ranking and detecting the duplicate bug reports [34]. The performance of these systems are evaluated using different performance metrics. The accuracy results of proposed system are compared with Aggarwal *et al.*, Hindle *et al.*, Deshmubk *et al.* and Nguyen *et al.* as shown in Table 5 and Figure 8. It is stated that proposed detection and classification system achieves the better accuracy rate than other traditional duplicate bug report systems on NetBeans, Eclipse, Mozilla and Open office datasets. The accuracy rate of proposed system is in between 85-99% for four datasets. Whereas, the accuracy rate of Aggarwal *et al.*, Nguyen *et al.*, and Hindle *et al.*

TABLE 5. Comparison of proposed duplicate bug report detection and classification system with existing systems using accuracy metric.

Datasets	Nguyen et al. (2012)	Aggarwal et al. (2016)	Hindle et al. (2016)	Deshmubk et al. (2017)	Proposed System
Eclipse	82%	93.18%	96.75%	72.68%	98.33%
Open office	73%	91.57%	93.67%	82.75%	95.24%
Mozilla	74%	93.07%	94.77%	-	99.38%
NetBeans	-	-	-	77.45%	85.45%

- The dataset is not used in the paper

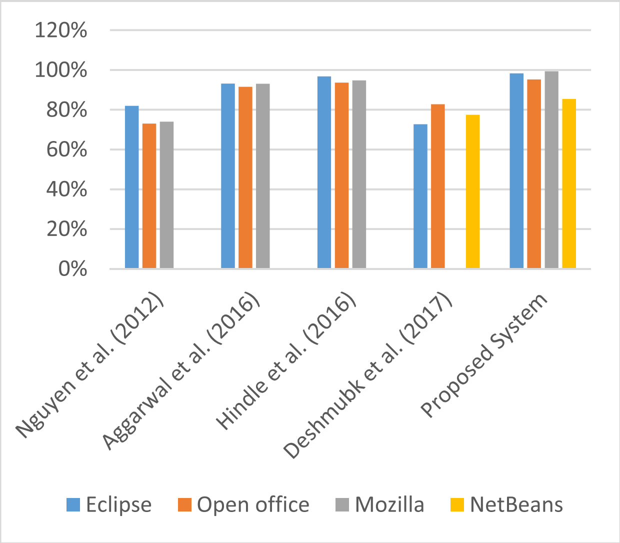


FIGURE 8. The graphical representation of the comparison of proposed duplicate bug report detection and classification system with existing systems using accuracy metric.

systems are in the range of 91-94%, 73-82% and 93-97% respectively for Eclipse, Open office and Mozilla datasets. The accuracy rate of Deshmubk *et al.* is between 72-73% for Eclipse, Open office and NetBeans datasets. In the conclusion, the classification and detection rate of proposed system is improved significantly than Aggarwal *et al.*, Hindle *et al.*, Deshmubk *et al.* and Nguyen *et al.* systems. It is additionally seen that proposed deep learning based system determines the relevant features for similarity measurement and enhances the result of classification of bugs as duplicate and non-duplicate.

TABLE 6. Comparison of proposed duplicate bug report detection and classification system with existing systems using recall@20 metric.

Datasets	Sun et al. (2010)	Sun et al. (2011)	Kim et al. (2011)	Ebrahimi et al. (2019)	Proposed System
Eclipse	62%	67%	-	-	93.93%
Open office	66%	69%	-	-	88.21%
Gnome	-	-	71%	71.57%	79.95%
Firefox	-	-	67.65%	76.43%	83.18%

- The dataset is not used in the paper

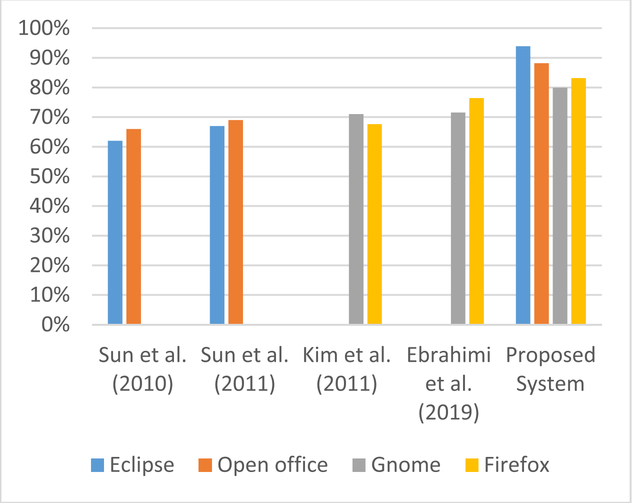


FIGURE 9. The graphical representation of the comparison of proposed duplicate bug report detection and classification system with existing systems using recall@20 metric.

Further, Table 6 and Figure 9, illustrate the performance of proposed system, Sun *et al.*, Kim *et al.*, and Ebrahimi *et al.* using recall@20 performance metric on Open office, Eclipse, Firefox and Gnome datasets. The recall@20 rate of proposed system is higher than Sun *et al.*, Kim et al *et al.* and Ebrahimi et al systems. The recall@20 rate of proposed system for Eclipse and Open office datasets are 93.93% and 88.21%. While recall@20 of both Sun *et al.* systems for Eclipse and Open office datasets are 62% and 67% respectively. The recall@20 rate of proposed system is 79.95% and 83.18% for Gnome and Firefox datasets. Whereas, Kim et al *et al.* and Ebrahimi *et al.* systems achieve the recall@20 rate i.e. 71% and 71.57% for Gnome dataset and 67.65% and 76.43% for Firefox datasets. Furthermore, it is found that proposed systems obtains better recall@20 rate utilizing all datasets i.e. 83-94% and Eclipse dataset archives higher recall@20 rate rather than others datasets. Hence, it is summarized as proposed deep learning approach improves the duplicate bug report detection and classification rate significantly rather than other existing approaches.

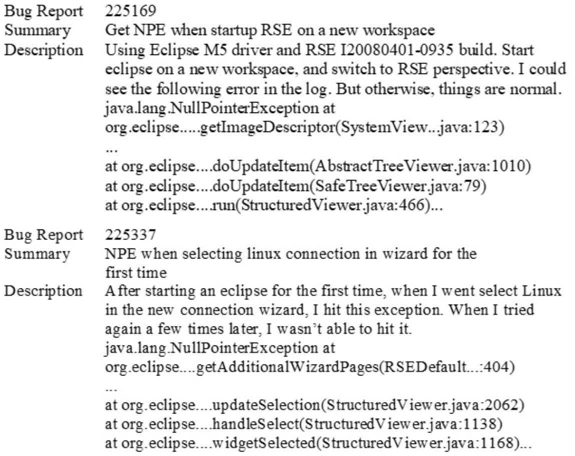
3) RESEARCH QUESTIONS

The two research questions are answered to know the effective of proposed system using deep learning technique over the existing automated duplicate bug report detection systems using traditional techniques are follows:

RQ1: Is deep learning technique extract the relevant (important) features for automated duplicate bug report detection and classification system?

Ans. RQ1): In the proposed system, deep learning approach i.e. CNN is employed with similarity measurement layer and classification model. Initially bug reports text is pre-processed and each word is mapped to its corresponding embedding by Word2Vec in the dictionary. The output of this step is set of feature vectors that represent each word in the bug report. These all feature are not equally significant and a few of them are irrelevant. These irrelevant features can be degrading the performance of duplicate bug report detection and classification system. To enhance the results of system, previous researchers have utilized various feature extraction techniques to determine the important set of features. In this paper, CNN based feature extraction technique is adopted to extract the important features form feature sets. Further, above mentioned feature set are processed parallel by the identical neural network i.e. CNN and the output is relevant feature of the bug report text. The relevant feature of bug reports is compared by the similarity measurement layer; output is the similar features. These features are fed into fully connected layer for computing the similarity score. This score is fed into duplicate bug report detection and classification module to classify the bug report as duplicate and non-duplicate. It is stated that CNN based feature extraction technique significantly extracts the relevant features for all datasets. An example of input bug report with bug ID 225169 and result of detection system result with bug ID 225337 is shown in Figure 10. It can be seen that the detected bug report is nearly similar to the input bug report. Tables 4 and 5 showed the experimental results of proposed system. The proposed system obtains state of art results as compared to Sun *et al.*, Nguyen *et al.*, Aggarwal *et al.*, Hindle *et al.*, Deshmubk *et al.*, Kim et al *et al.* and Ebrahimi *et al.* systems. The experimental results of proposed system show the significant improvement due to the relevant features extraction using CNN based feature extraction technique. Therefore, it is concluded that deep learning technique extracts relevant features for automated duplicate bug report detection and classification system and also improve the classification results.

RQ2. Is deep learning based feature extraction and similarity measurement technique enhance the performance of



```
Bug Report    225169
Summary       Get NPE when startup RSE on a new workspace
Description   Using Eclipse M5 driver and RSE I20080401-0935 build. Start
              eclipse on a new workspace, and switch to RSE perspective. I could
              see the following error in the log. But otherwise, things are normal.
              java.lang.NullPointerException at
              org.eclipse.....getImageDescriptor(SystemView...java:123)
              ...
              at org.eclipse....doUpdateItem(AbstractTreeViewer.java:1010)
              at org.eclipse....doUpdateItem(SafeTreeViewer.java:79)
              at org.eclipse....run(StructuredViewer.java:466)...
Bug Report    225337
Summary       NPE when selecting linux connection in wizard for the
              first time
Description   After starting an eclipse for the first time, when I went select Linux
              in the new connection wizard, I hit this exception. When I tried
              again a few times later, I wasn't able to hit it.
              java.lang.NullPointerException at
              org.eclipse....getAdditionalWizardPages(RSEDefault...:404)
              ...
              at org.eclipse....updateSelection(StructuredViewer.java:2062)
              at org.eclipse....handleSelect(StructuredViewer.java:1138)
              at org.eclipse....widgetSelected(StructuredViewer.java:1168)...
```

FIGURE 10. **An Example of input bug report and detected bug report by the proposed system of Eclipse dataset.**

duplicate bug report detection and classification system than traditional feature extraction and similarity measurement techniques?

Ans. RQ2): In proposed system, deep learning model is proposed for feature extraction and similarity measurement task for detection of duplicate bug reports. The relevant features are extracted through CNN based feature extraction model. CNN model has five layers such as Input, Convolutional, Activation, Dropout, Max pooling. These layers have Siamese structure to generate the semantic and syntactic relationship of each word of the bug report content from its preceding and following words. The structure has two networks, one extract the relevant feature of one bug report and another extract the features of other, the feature weight is tied down with each feature. Further, CNN model has various convolutions filters that capture the local features, semantic relationship between words and examines all words of the bug reports from multiple prospective. Further, deep learning model has two layers i.e. Similarity Measurement layer and Fully connected layer to calculate the similarity of bug reports. Similarity Measurement Layer contains similarity measurement metric, which compare the sentence representation of the bug reports. Fully Connected Layer calculates the similarity score of the sentences of bug reports. The experimental results of proposed system are compared with Sun *et al.*, Nguyen *et al.*, Aggarwal *et al.*, Hindle *et al.*, Deshmubk *et al.*, Kim et al *et al.* and Ebrahimi *et al.* systems. These systems consist of tradition feature extraction techniques for duplicate bug report detection as mentioned in the section D and subsection 2. The experimental results of all existing systems are illustrated in Tables 5 and 6. It can be seen that proposed system provides more significant results than existing systems.

It is observed that average accuracy rate of proposed system ranges in between 85% to 99 %, whereas of Nguyen *et al.*, Aggarwal *et al.*, Hindle *et al.*, and Deshmubk *et al.* systems are in between 73%-82%, 91%-94%, 93%-97% and 72%-83% respectively. It is also noticed that average recall@20 rate of proposed system ranges in between 79%-94%, whereas of both Sun *et al.*, Kim et al *et al.* and Ebrahimi *et al.* systems are in between 62%-66%, 67%-69%, 67%-71% and 71%-77% respectively. Hence, it can be said that proposed duplicate bug report detection and classification system outperforms than existing systems. In proposed system, deep learning based feature extraction and similarity measurement technique is implemented for detecting the duplicate bug reports. So, it can be stated that deep learning based feature extraction and similarity measurement technique improves the performance of proposed system and also provides better results than traditional feature extraction and similarity measurement techniques.

4) CROSS TRAINING RESULTS OF PROPOSED SYSTEM

In reality, many researchers do not have supervised training bug report data in mostly industrial software projects. This problem restricts the researchers from using the supervised approaches in practice for various software projects. To address this problem, the proposed system is trained by bug report samples from other software project. The results of this experiment is illustrated in Table 7. In this experiment the proposed system is trained with Eclipse dataset (project) and tested with NetBeans dataset vice-versa. It can be seen from the results that the proposed system also provides good accuracy prediction because both datasets have same domain and genre i.e. Integrated Development Environment. Further, the proposed system is trained with Eclipse dataset and tested with Open office dataset. The accuracy of this cross training experiment is low due to the different domain of datasets. it is demonstrated that the proposed system learns the generic syntactic and semantic relationship of features as well as latent features of the domain, thus provides the better results for cross training project with in the same domain.

TABLE 7. **Accuracy of proposed system using Cross training datasets.**

Tested Dataset	Trained Dataset	Accuracy
Net Beans	Eclipse	80.09
Eclipse	Net Beans	84.32
Open office	Eclipse	70.22

F. THREATS TO VALIDITY

External Validity: Six datasets are used to evaluate the proposed system. These datasets belong to the different platforms due to this generalization of the proposed system is not guaranteed. Further, external validity is hampered the data source as accuracy of duplicate bug report labels or the lack of labels that depends on developer marking the bug reports that way.

Internal Validity: In this work, domain specific features (extracted with CNN based Feature extraction layer) are combined into a duplicate bug report detection. We only have used one combination style, but there is other different styles or ways of combining the domain specific features. However, the integration of word embedding feature into the CNN model has been produced more effective results than baseline methods. The other integration style, pooling functions is discussed in the future.

V. CONCLUSION

In this paper, a novel automatic duplicate bug report detection and classification system using deep learning technique is proposed. The proposed system utilized CNN based deep learning model for relevant feature extraction. Further, the CNN model has various convolution filters that capture the local features and examines all words of the bug reports from multiple prospective. Further, deep learning model has two layers i.e. Similarity Measurement layer and Fully connected layer to calculate the similarity of bug reports. Similarity Measurement Layer contains similarity measurement metric, which compare the sentence representation of the bug reports. Fully Connected Layer computes the similarity score of two bug reports. After that, bug reports are classified as duplicate or non-duplicate on the basis of similarity scores. The performance of the proposed system is tested over publicly available datasets. These datasets are Mozilla, Eclipse, NetBeans, Gnome, Open office and Firefox. The experimental results are evaluated using F-measure, precision, accuracy, recall@k and recall metrics. The performance of proposed system is compared with existing duplicate bug report systems such as Sun *et al.*, Nguyen *et al.*, Aggarwal *et al.*, Hindle *et al.*, Deshmubk *et al.*, Kim *et al.* and Ebrahimi *et al.* It is observed that the proposed system achieves higher accuracy rate in the range of 85% to 99 % and recall@k rate in between 79%-94% rather than existing systems. The proposed system improves the recal@20 by 19%-31% over both Sun *et al.* systems, 8%-16% over Kim *et al.* and 7%-8% over Ebrahimi *et al.* system. The accuracy rate of proposed system improves by 2%-5%, 4%-6%, 17%-25% and 8%-26% over Nguyen *et al.*, Aggarwal *et al.*, Hindle *et al.*, Deshmubk *et al.* systems respectively. The proposed system does not utilize hand crafted features like mostly other existing systems for duplicate bug report detection. Moreover, the effectiveness of proposed system is also measured on the cross training datasets of same and different domain. The proposed system provides good high accuracy rate for same domain datasets and low accuracy rate for different domain datasets. In future, other deep learning based feature extraction techniques will be explored to extract the relevant features for predicting the similarity score of bug reports. The efficiency of proposed system is also measured in the absence of structural information in a bug report.

REFERENCES

[1] Y. Sharma, A. Dagur, and R. Chaturvedi, "Automated bug reporting system with keyword-driven framework," in *Soft Computing and Signal Processing*. Singapore: Springer, 2019, pp. 271–277.

[2] T. S. S. Angel, G. S. Kumar, V. M. Sehgal, and G. Nayak, "Effective bug processing and tracking system," *J. Comput. Theor. Nanoscience*, vol. 15, no. 8, pp. 2604–2606, Aug. 2018.

[3] G. Canfora and L. Cerulo, "How software repositories can help in resolving a new change request," in *Proc. STEP*, 2005, p. 99.

[4] J. Anvik, L. Hiew, and G. C. Murphy, "Coping with an open bug repository," in *Proc. OOPSLA Workshop Eclipse Technol. Exchange*, 2005, pp. 35–39.

[5] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *Proc. 13th Int. Conf. Softw. Eng.*, 2008, pp. 461–470.

[6] J. L. Davidson, N. Mohan, and C. Jensen, "Coping with duplicate bug reports in free/open source software projects," in *Proc. IEEE Symp. Vis. Lang. Hum.-Centric Comput. (VL/HCC)*, Sep. 2011, pp. 101–108.

[7] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report," in *Proc. 16th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2008, pp. 308–318.

[8] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neu- ral probabilistic language model," *J. Mach. Learn. Res.*, vol. 3, pp. 1137–1155, Apr. 2003.

[9] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proc. 28th Int. Conf. Softw. Eng.*, 2006, pp. 361–370.

[10] E. Kn, I. Ms, A. Hamou-Lhadj, and M. Hamdaqa, "An effective method for detecting duplicate crash reports using crash traces and hidden Markov models," in *Proc. 26th Annu. Int. Conf. Comput. Sci. Softw. Eng.*, 2016, pp. 75–84.

[11] N. Kaushik and L. Tahvildari, "A comparative study of the performance of IR models on duplicate bug detection," in *Proc. 16th Eur. Conf. Softw. Maintenance Reeng.*, Mar. 2012, pp. 159–168.

[12] A. Tsuruda, Y. Manabe, and M. Aritsugi, "Can we detect bug report duplication with unfinished bug reports?" in *Proc. Asia–Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2015, pp. 151–158.

[13] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 444–469, 2015.

[14] X. Wang, J. Jiao, J. Yin, W. Zhao, X. Han, and B. Sun, "Underwater sonar image classification using adaptive weights convolutional neural network," *Appl. Acoust.*, vol. 146, pp. 145–154, Mar. 2019.

[15] A. Kukkar, R. Mohana, A. Nayyar, J. Kim, B.-G. Kang, and N. Chilamkurti, "A novel Deep-Learning-Based bug severity classification technique using convolutional neural networks and random forest with boosting," *Sensors*, vol. 19, no. 13, p. 2964, Jul. 2019.

[16] B. Am, J. Ahmad, N. Rahim, and B. Sw, "Speech emotion recognition from spectrograms with deep con- volutional neural network," in *Proc. Int. Conf. Platform Technol. Service (PlatCon)*, 2017, pp. 1–5.

[17] H. Yang, C. Meng, and C. Wang, "Data-driven feature extraction for analog circuit fault diagnosis using 1-D convolutional neural network," *IEEE Access*, vol. 8, pp. 18305–18315, 2020.

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

[19] H. Li, "Deep learning for natural language processing: Advantages and challenges," *Nat. Sci. Rev.*, vol. 5, no. 1, pp. 24–26, Jan. 2018.

[20] J. Mueller and A. Thyagarajan, "Siamese recurrent architectures for learning sentence similarity," in *Proc. 13th AAAI Conf. Artif. Intell.*, 2016, pp. 1–5.

[21] A. Lazar, S. Ritchey, and B. Sharif, "Generating duplicate bug datasets," in *Proc. 11th Work. Conf. Mining Softw. Repositories*, 2014, pp. 392–395.

[22] S. Kim, T. Zimmermann, and N. Nagappan, "Crash graphs: An aggregated view of multiple crashes to improve crash triage," in *Proc. IEEE/IFIP 41st Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2011, pp. 1–8.

[23] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *Proc. 29th Int. Conf. Softw. Eng. (ICSE)*, May 2007, pp. 499–510.

[24] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in *Proc. IEEE Int. Conf. Dependable Syst. Netw. With FTCS DCC (DSN)*, 2008, pp. 52–61.

[25] X. Wang, D. Lo, J. Jiang, L. Zhang, and H. Mei, "Extracting paraphrases of technical terms from noisy parallel software corpora," in *Proc. ACL-IJCNLP Conf. Short Papers*, 2009, pp. 197–200.

[26] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate bug reports considered harmful 2026; Really?" in *Proc. IEEE Int. Conf. Softw. Maintenance*, 2008, pp. 337–345.

[27] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng.*, 2010, pp. 45–54.

[28] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in *Proc. 26th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2011, pp. 253–262.

[29] Y. Tian, C. Sun, and D. Lo, "Improved duplicate bug report identification," in *Proc. 16th Eur. Conf. Softw. Maintenance Reeng.*, Mar. 2012, pp. 385–390.
[30] A. Sureka and P. Jalote, "Detecting duplicate bug report using character N-Gram-Based features," in *Proc. Asia Pacific Softw. Eng. Conf.*, Nov. 2010, pp. 366–374.
[31] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in *Proc. 27th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2012, pp. 70–79.
[32] J. Lerch and M. Mezini, "Finding duplicates of your yet unwritten bug report," in *Proc. 17th Eur. Conf. Softw. Maintenance Reeng.*, Mar. 2013, pp. 69–78.
[33] A. Alipour, A. Hindle, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection," in *Proc. 10th Work. Conf. Mining Softw. Repositories (MSR)*, May 2013, pp. 183–192.
[34] A. Hindle, A. Alipour, and E. Stroulia, "A contextual ap- proach towards more accurate duplicate bug report detection and ranking," *Empirical Softw. Eng.*, vol. 21, no. 2, pp. 368–410, 2016.
[35] K. Aggarwal, T. Rutgers, F. Timbers, A. Hindle, R. Greiner, and E. Stroulia, "Detecting duplicate bug reports with software engineering domain knowledge," in *Proc. IEEE 22nd Int. Conf. Softw. Anal., Evol., Reeng. (SANER)*, Mar. 2015, pp. 1–5.
[36] K. Aggarwal, F. Timbers, T. Rutgers, A. Hindle, E. Stroulia, and R. Greiner, "Detecting duplicate bug reports with software engineering domain knowledge," *J. Softw., Evol. Process*, vol. 29, no. 3, p. e1821, Mar. 2017.
[37] J. Deshmukh, K. M. Annervaz, S. Podder, S. Sengupta, and N. Dubash, "Towards accurate duplicate bug retrieval using deep learning techniques," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2017, pp. 115–124.
[38] K. Ne and A. Hamou-Lhadj, "CrashAutomata: An ap- proach for the detection of duplicate crash reports based on generalizable automata," in *Proc. CASCON*, 2015, pp. 201–210.
[39] K. K. Sabor, A. Hamou-Lhadj, and A. Larsson, "DURFEX: A feature extraction technique for efficient detection of duplicate bug reports," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. (QRS)*, Jul. 2017, pp. 240–250.
[40] N. Ebrahimi, A. Trabelsi, M. S. Islam, A. Hamou-Lhadj, and K. Khanmohammadi, "An HMM-based approach for automatic detection and classification of duplicate bug reports," *Inf. Softw. Technol.*, vol. 113, pp. 98–109, Sep. 2019.
[41] K. W. Church, "Word2 Vec," *Natural Lang. Eng.*, vol. 23, no. 1, pp. 155–162, Jan. 2017.
[42] F. S. Al-Anzi and D. AbuZeina, "Toward an enhanced arabic text classification using cosine similarity and latent semantic indexing," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 29, no. 2, pp. 189–195, Apr. 2017.
[43] S. Kumar and M. Kumar, "A study on the image detection using convolution neural networks and TenserFlow," in *Proc. Int. Conf. Inventive Res. Comput. Appl. (ICIRCA)*, Jul. 2018.
[44] A. Kukkar and R. Mohana, "A supervised bug report classification with incorporate and textual field knowledge," *Procedia Comput. Sci.*, vol. 132, pp. 352–361, 2018.
[45] P. Kingma and J. Ba, "Adam: A method for stochastic op- timization," in *Proc. 3rd Int. Conf. Learn. Represent.*, San Diego, CA, USA, 2015, pp. 1–15.

ASHIMA KUKKAR received the B.Tech. degree in information and technology from Rayat Bhara University, Mohali, India, in 2014, the M.Tech. degree in computer science and engineering from the Jaypee University of Information Technology, India, in 2016, and the Ph.D. degree in computer science (bug summarization, severity classification, and assignment for automated bug resolution process) from the Jaypee University of Information Technology, in 2019. She likes applying machine learning and deep learning in text. She has a passion for software engineering with the goal of improving the software development life cycle. Her research interests include problems relating to software engineering and natural language processing.

RAJNI MOHANA received the B.Tech. degree in computer science and engineering from BPUT, Rourkela, India, in 2003, the M.Tech. degree in computer science and engineering from Punjab Technical University, Jalandhar, India, in 2009, and the Ph.D. degree in computer science and engineering from the Jaypee University of Information Technology (JUIT), Waknaghat, India, in 2013.

She has over 17 years of experience in Academia. She is currently an Associate Professor with the Department of CSE, JUIT. She also works on interoperability problem in service oriented architecture computing. She has published her research in various impact factor journals, such as *Senors* (MDPI) and *Proceedings of National Academy of Sciences: Physical Sciences*. She has published around 20 articles in various ESCI and Scopus journals. She has guided four master's student and two Ph.D. students. She also guiding two more students. Her research interest includes sentiment analysis cloud computing. She served as the TPC Chair for the 2015 International Conference on Image Information Processing, JUIT, in December 2015, and the TPC Co-Chair for the 2017 International Conference on Image Information Processing, in December 2017. She serves as a reviewer for various renowned international journals and international conferences.

YUGAL KUMAR received the B.Tech. and M.Tech. degrees from MDU, in 2006 and 2009, respectively, and the Ph.D. degree from the Birla Institute of Technology, in 2016. His research interests include meta heuristic algorithms, swarm intelligence, pattern recognition, medical data mining, and machine learning.

ANAND NAYYAR (Senior Member, IEEE) received the Ph.D. degree in computer science (wireless sensor networks) from Desh Bhagat University, in 2017. He is currently with the Graduate School, Duy Tan University, Da Nang, Vietnam. He has a certified professional with more than 75 professional certificates from CISCO, Microsoft, Oracle, Google, EXIN, Beingcert, GAQM, Cyberoam, and so on. He has published more than 300 research papers in various national and international conferences and international journals (Scopus/SCI/SCIE/SSCI Indexed). He has authored/coauthored cum Edited 25 books of computer science. He was with more than 400 international conferences as a programme committee/advisory board/review board member. He is also an ACM Distinguished Speaker. He holds two patents in the Internet of Things and speech processing. His research interests include wireless sensor networks, MANETS, swarm intelligence, cloud computing, the Internet of Things, blockchain, machine learning, deep learning, cyber security, network simulation, and wireless communications. He was a member of more than 50 associations as a senior and a life member. He received over 20 awards for the Teaching and Research—Young Scientist, the Best Scientist, the Young Researcher Award, the Outstanding Researcher Award, and Publons-Top 1% Reviewer Award (Computer Science, Engineering, and Cross-Fields). He serves as the Editor-in-Chief for the *International Journal of Smart Vehicles and Smart Transportation (IJSVST)* (IGI-Global, USA).

MUHAMMAD BILAL (Senior Member, IEEE) received the B.Sc. degree in computer systems engineering from the University of Engineering and Technology Peshawar, Pakistan, in 2008, the M.S. degree in computer engineering from Chosun University, Gwangju, South Korea, in 2012, and the Ph.D. degree in information and communication network engineering from the School of Electronics and Telecommunications Research Institute (ETRI), Korea University of Science and Technology, in 2017. From 2017 to 2018, he was with Korea University, where he was also a Postdoctoral Research Fellow with the Smart Quantum Communication Center. Since 2018, he has been an Assistant Professor with the Division of Computer and Electronic Systems Engineering, Hankuk University of Foreign Studies, Yongin, South Korea. His research interests include design and analysis of network protocols, network architecture, network security, the IoT, named data networking, blockchain, cryptology, and future internet. He was a Technical Program Committee Member on many international conferences, including the IEEE VTC, the IEEE ICMC AI-IoT, and the IEEE CCNC. He serves as an Editor for the IEEE Future Directions Ethics and Policy in Technology Newsletter and the IEEE Internet Policy Newsletter. He served as a Reviewer for various international journals, including *IEEE Communications Magazine*, the IEEE INTERNET OF THINGS JOURNAL, the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, the IEEE SYSTEMS JOURNAL, IEEE ACCESS, the IEEE COMMUNICATIONS LETTERS, the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, *Journal of Networks*, the IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING, *Journal of Computer Applications*, *Personal and Ubiquitous Computing*, and the *International Journal of Communication Systems*.

KYUNG-SUP KWAK (Life Senior Member, IEEE) received the Ph.D. degree from the University of California.

He was with Hughes Network Systems and IBM Network Analysis Center, USA. He was with Inha University, South Korea, as a Professor. He was also the Dean of the Graduate School of Information Technology and Telecommunications and the Director of the UWB Wireless Communications Research Center. In 2008, he was an Inha Fellow Professor (IFP). He is currently an Inha Hanlim Fellow Professor. His research interests include UWB radio systems, wireless body area network/u-health networks, and nano and molecular communications. In 2006, he was the President of the Korean Institute of Communication Sciences (KICS) and the Korea Institute of Intelligent Transport Systems (KITS) in 2009. He received the official commendations for achievements of UWB radio technology R/D from the Korean President in 2009.