**Automated Bug Triaging**

**A Comparative Analysis**

Sameen Azhar

College of Computing & Information Science - CoCIS

PAF-KIET

Karachi, Pakistan

sameenazhar67@gmail.com

Syed Hamza Haider

College of Computing & Information Science - CoCIS

PAF-KIET

Karachi, Pakistan

Hamzahaider.dsc@gmail.com

***Abstract— Correctly assigning bugs to the right developer or team, i.e. bug triaging, is a costly activity. In this paper, we propose to apply machine learning techniques to assist in bug triage, which maps the words in the bug reports (i.e., the term space) to their corresponding topics (i.e., the topic space) by using text categorization to predict the developer that should work on the bug based on the bug's description. Our main purpose is to reduce the project cost, avoid tossing and wastage of time by assigning bug resolution to the relevant team member. We will be working on the dataset of Google Chromium to provide assistance to Google Chromium developer teams. We will be comparing multiple algorithms and their outcomes according to our dataset. We will then be analyzing and discussing the results we got from our chosen methods related to our dataset and suggest the top-K accurate methods for better bug triaging.***

***Keywords— Bug/Issue triaging system; Comparison of algorithms; Bugs resolution; Classification criteria; Machine learning techniques***

# Introduction

Because of the expanding unpredictability of the framework, numerous product ventures will inevitably release numerous bugs. Bug fixing is an integral part of development and maintenance phase in the lifecycle of a software project. Huge critical software projects must deal with client bugs rapidly. The first step in the process is to triage the bug by assigning it to the team or developer that can fix the bug. The large volume of bug reports submitted daily makes manual bug triaging a time-consuming process. Furthermore, when a bug is assigned to the wrong team or developer, the cost and time to fix the bug is increased [1].

Bug reporting is a standard practice in both open source software projects and commercial projects. When a bug report is submitted to the bug report tracking system, the back-stage manager needs to manually assign the bug report to the appropriate developer based on its description. This process of assigning the appropriate developer to the bug report is called bug triaging. However, bug triaging is a

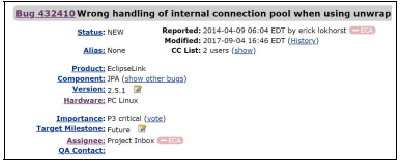


Fig.1. Bug report for bug 432410 in Eclipse. The description is “wrong handling of internal connection pool when using unwrap.” This bug was reported in 2014 and modified in 2017.

time consuming and laborious task with two main challenges. One is that the number of bug reports is very large. For large projects, bug tracking systems usually receive a large number of bug reports every day. For example, around 91 bug reports related to the Eclipse project are summited to bug tracking system every day. The other is that there are so many developers involved in the bug repair, so it is hard for back-stage managers to be familiar with the ability level of all the developers, so the manual bug triaging often fails to assign bug reports to the appropriate workers [2, 3].

Bug prioritization underscores the urgency of fixing a bug. Bug triage is relevant to bug prioritization since it utilizes many bug features to prioritize and assign a bug to the appropriate developers [4]. An example of a bug report with its textual summary is shown in Fig. 1. The process of assigning bugs to the right developers is done by a bug triage who studies the bug report and its features before assigning it to one or more developers. Several features are considered in bug such as its severity level and developer expertise. This article focuses mainly on the priority features of reports for bug triaging [5].

In this report, we will be utilizing six different machines learning algorithms and implementing each algorithm on data sets to evaluate the results. After the outcomes will be achieved, we will compare them and do analysis to know which one is most accurate among all for bug triaging. Bug triaging uses text classification and text summarization techniques by using algorithms and then model training on TensorFlow.

This paper is organized as follows: Section 2 begins with a brief literature survey, previous work system and its limitations. Section 3 presents research methodologies along with details of the software. The algorithms are presented in Section 4. Section 5 outlines the details of the algorithms and Section 6 shows the results. Section 7 includes the comparative analysis of algorithms and discussions on evaluated results. The paper ends with the conclusions and references.

# LITERATURE SURVEY

Throughout the years there have been numerous endeavors to automate bug triaging. The first effort was made by Cubrani ˇ c´ and Murphy [6] by using a text categorization technique with a Naïve Bayes classifier algorithm on Eclipse data. Their dataset comprised of 15,670 bug reports with 162 classes (developers). They accomplished about 30% precision.

## Previous Works

Anvik et al used Support Vector Machines (SVM) on Eclipse, Firefox and GCC data. They achieved precision of 64% and 58% on Firefox and Eclipse data, however, only 6% on GCC data. As for recall, only 2%, 7% and 0.3%: results were achieved on Firefox, Eclipse and GCC data [3].

Ahsan et al. [4] used an SVM classifier on Mozilla data. They reached 44.4% classification accuracy, 30% precision and 28% recall using SVM with LSI.

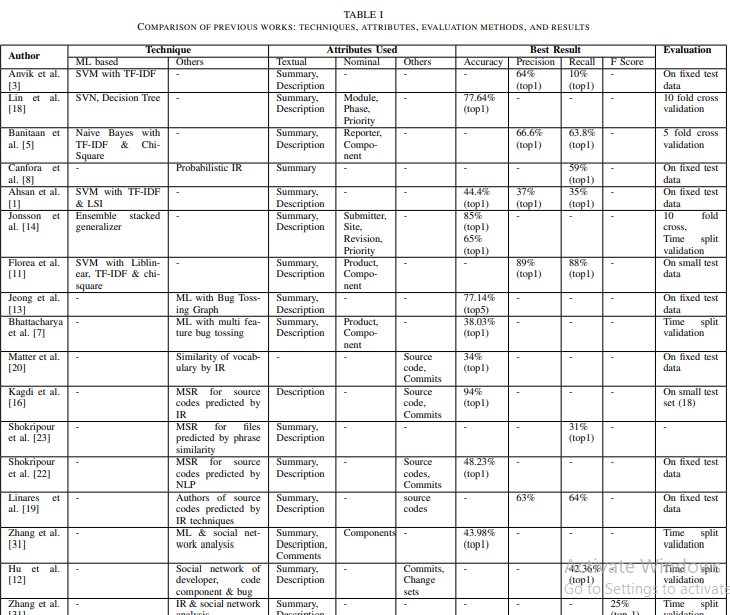
Another method on bug triaging is supplied via way of means of Xuan et al wherein Expectation Maximization and Naïve Bayes Classifiers have been used collectively on categorized bug reports to categories unlabeled ones. The weighted advice listing turned into used to beautify the overall performance of the type method via implementing the load of capabilities approximately builders while education the classifier. Those research works did now no longer don't forget textual capabilities for bug triaging. Xuan et al. brought a data reduction bug triaging approach which makes a specialty of losing the dimensions and improving the first-rate of bug data. The experiments have been performed at the Mozilla and Eclipse projects. While the method addressed the first-rate of textual capabilities, the relationships among the ones capabilities will be discarded while sampling data, resulting in reducing the general effectiveness of bug triage predictive models.

Bug Fixer was developed by Hu et al. [7] as a bug triaging tool for large projects. It implemented a technique to measure the similarity between bug reports through constructing Developer Component Bug Network (DCB) that models the relationship between developers and source code components, and the relationship between the components and their associated bugs. The system calculates the similarity between the new bugs and the existing bugs and then advocates developers according to the structure of DCB. It might not be straightforward to dynamically update the created structure.

Xia et al. proposed the TopicMinerMTM model, adding supervision information to the Latent Dirichlet Allocation (LDA), so that the subject of the bug report is supervised by its characteristics (the metadata mentioned in this paper) to obtain a closer subject distribution. Xuan 123 S. Guo et al. prioritized developers by extending social networking techniques to analyze developer information. Based on the results of the developer ranking, three influencing factors, product characteristics, time variation and noise tolerance were analyzed. Then the researchers dealt with the bug report assignment, severity prediction, and bug report restart prediction based on the priority of the developer. Naguib et al. used the repair information of historical developers to analyze the experience of each developer, and used topic models to compare the similarities of bug reports and developers for bug reports assignment [8].

In the past work, the researchers likewise viewed as other bug reporting data, such as items and segments. Yang et al. first determined the closeness of bug reports by utilizing the topic model LDA strategy. Joined with numerous property data, the bug reports that are conflicting with the current ones were sifted through. In view of these comparable bug reports, the appropriate designers were prescribed to fix the bug reports.

The literature shows that the meta data of the bug report isn't reliable, and just the short description and long depiction text data in the bug report are steady and reliable. All other meta-data is altered again by the back-office manager dependent on the post-repair status of the bug report.



## Existing Methodology

In the existing framework there is no legitimate strategy which can recognize the bugs precisely and give arrangements in exact way. All the current methodologies attempt to get the bugs physically and henceforth OF assigning the bugs likewise done dependent on FCFS way notwithstanding of assigning the bugs dependent on highlight extraction. All the current frameworks don't follow the triage strategy while assigning the bugs to the end clients, thus it is dreary for the end clients who get the bug solution. There are many limitations in the existing system while mapping the bugs and triaging the bugs.

## Limitations of the Existing Methodology

The following are the limitations of the existing methodology. They are as follows:

1) All the existing bug recognition frameworks follow manual way to deal with fathom the bug.

2) There is no strategy like automated bug triage.

3) All the current strategies are restricted uniquely on FCFS method (First Come First Serve).

4) All the existing methodologies attempt to assign the bug naturally for the engineer who is accessible free, but it don't follow the procedure like appointing the bug situated in highlights.

5) Existing techniques require runtime executions.

6) In customary programming improvement, new bugs are physically triaged by a specialist engineer, i.e., a human emergency. Because of the enormous number of every day bugs and the absence of skill of the apparent multitude of bugs, manual bug emergency is costly in time cost and low in exactness.

# Research Methodology

This report is focused on automating bug report triaging, i.e., instructions to diminish the bug information to spare the work cost of engineers and improve the quality to encourage the process of bug triage. Here the information decrease for bug triage expects to construct a little scope and great arrangement of bug information by eliminating bug reports and words, which are excess or non-instructive. Here the automatic method is applied in which the bugs are allocated dependent on the highlights which are extricated from the bug subject and dependent on those keywords the bug is relegated for the suitable developer.

Its main purpose is to reduce the project cost, avoid tossing and wastage of time by assigning bug resolution to the relevant team member. We will be working on the dataset of Google Chromium to provided assistance to Google Chromium developer teams. We will be comparing multiple algorithms and their outcomes according to our dataset. We will then be analyzing and discussing the results we got from our chosen methods related to our dataset and suggest the top-K accurate methods for better bug triaging.

## Functional Features

The functional features that are required in our research methodology for automated bug triage are as follows:

* Natural Language Processing.
* Text summarization (TF-IDF, BOWs)
* Text classification (Naive Bayes algorithm, LDA, K-Nearest Neighbor, Random Forest)
* TensorFlow model training.

All these will be done on 4 different data sets.

* Top-K accuracy
* previous research analysis and comparison
* Report completion

# PROPOSED ALGORITHMS

## Bag of Words Algorithm

Bag of words is a model for numerical representational of words to simplify content of the corpus and remove irrelevant words. This corpus is represented by a set of word vectors[x] ignoring the grammar words and word sequence keeping the total count of word’s occurrence.

Bag of words model is used for Natural Language Processing by reducing the size of data by ignoring the non-relevant and recursive words in text corpus.

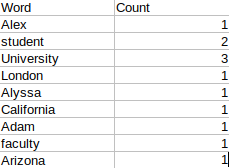
First step for Bag of words is to tokenize sentence which is convert all the sentences of corpus into words only, ignoring the spaces. Then create a list of all the words being used at least one time in the corpus. Give these words a frequency of total number of occurrence in the text and ignoring the stop words [x]. Last step is to create an array which represents each word’s occurrence for each sentence.

For example, we have these three sentences:

Alex is a student in University of London.

Alyssa is a student in University of Calfornia.

Adam is a faculty in univesity of Arizona.



First we tokenize the sentences as follow:

“Alex” “is” “a” “student” “in” “University” “of” “London”

“Alyssa” “is” “a” “student” “in” “University” “of” “California”

“Adam” “is” “a” “faculty” “in” “University” “of” “Arizona”

These sentences are then converted in vectors as shown below:



Now we create a matrix representing occurrence of all words in all the sentences by a vector count. Bag of words model will look like this.

1. *TF-IDF Algorithm*

TF\*IDF is a data recovery strategy that weighs a term's frequency (TF) and its inverse document frequency (IDF). Each word or term has its separate TF and IDF score. The result of the TF and IDF scores of a term is known as the TF\*IDF weight of that term.

It has many uses, most importantly in automated [text analysis](http://www.monkeylearn.com/text-analysis/), and is very useful for scoring words in machine learning algorithms for [Natural Language Processing](https://monkeylearn.com/blog/definitive-guide-natural-language-processing/) (NLP).

TF-IDF (term frequency-inverse document frequency) was invented for document search and information retrieval. It works by increasing proportionally to the number of times a word appears in a document, but is offset by the number of documents that contain the word. So, words that are common in every document, such as this, what, and if, rank low even though they may appear many times, since they don’t mean much to that document in particular.

However, if the word Bug appears many times in a document, while not appearing many times in others, it probably means that it’s very relevant. For example, if what we’re doing is trying to find out which topics some NPS responses belong to, the word Bug would probably end up being tied to the topic Reliability, since most responses containing that word would be about that topic.



1. *The Random Forest Classifier*

Random forest, similar to its name suggests, comprises of an enormous number of decision trees that work as a gathering. Every individual tree in the random forest lets out a class prediction and the class with the most votes turns into our model's prediction

Random forest is an administered learning algorithm which is utilized for both classification as well as regression. Yet, notwithstanding, it is fundamentally utilized for classification problems. As we realize that a forest is comprised of trees and more trees implies more robust forest. Also, similarly, random forest algorithm makes choice trees on information samples and afterward gets the prediction from every one of them lastly chooses the best solution by means of voting. It is an ensemble method which is better than a single decision tree since it diminishes the over-fitting by averaging the outcome.

We can understand the working of Random Forest algorithm with the help of following steps –

**Step 1** − First, start with the selection of random samples from a given dataset.

**Step 2** − Next, this algorithm will construct a decision tree for every sample. Then it will get the prediction result from every decision tree.

**Step 3** − In this step, voting will be performed for every predicted result.

**Step 4** − At last, select the most voted prediction result as the final prediction result.

The following diagram will illustrate its working −



1. *Random Forest with Bag of Words*

First of all, punctuations should be removed in order to scrap our data and clear out unwanted letters which is special characters. Computer reads every input case sensitively. If same word with different capitalizations comes than it will not be considered as same word but every pattern of capitalization will be considered as a new word so all the data must be converted into small characters. Our data has many irrelevant words which do not reflect any meaning by themselves but sentences make sense due to them, these words are known as stop words. These stop words are not much valuable for our data so stop words must be removed to keep data as simple as possible. In a corpus of data, multiple words which are derived from same root can exist like “happened” and “happening” are derived from the word “happen” so we have to stem these words to keep the root words only rather than having multiple leaf words which creates a larger dictionary than we need. These steps make our data ready for the training and testing.

In our dataset, we have two columns; 1 for the bug resolver and the other for description[x]. Our model is trained on the description column. First step we do here is to create a Bag of Words dictionary for each entry in description column and add these words in a new column. When all the entries have their own bag of words, each bag of words will be classified according to the bug resolver name in the first column. This classification will be done using Random Forest.

1. *Random Forest with TF-IDF*

Punctuations are unnecessary so they should be removed in order to scrap our data and clear out unwanted letters which is special characters. Data we enter in computers is case sensitive. If same word with different capitalizations comes than it will not be considered as same word but every pattern of capitalization will be considered as a new word so all the data must be converted into small characters. Our data has many irrelevant words which do not reflect any meaning by themselves but sentences make sense due to them, these words are known as stop words. These stop words are not much valuable for our data so stop words must be removed to keep data as simple as possible. In a corpus of data, multiple words which are derived from same root can exist like “happened” and “happening” are derived from the word “happen” so we have to stem these words to keep the root words only rather than having multiple leaf words which creates a larger dictionary than we need. These steps make our data ready for the training and testing.

In our dataset, we have two columns; 1 for the bug resolver and the other for description[x]. Our model is trained on the description column. First step we do here is to create a group of all the words appearing at least once in a comment. TF-IDF value will then be assigned for each words and array of TF-IDF values will be stored. When all the entries have their own TF-IDF for each word, each array of TF-IDF values will be classified according to the bug resolver name in the first column. This classification will be done using Random Forest.

1. *K-Nearest Neighbors (KNN) Algorithm*

KNN is a model that classifies data points dependent on the points that are generally like it. It utilizes test data to make an "educated guess" on what an unclassified point should be classified as.

KNN algorithm is considered as non-parametric and an example of lazy learning.

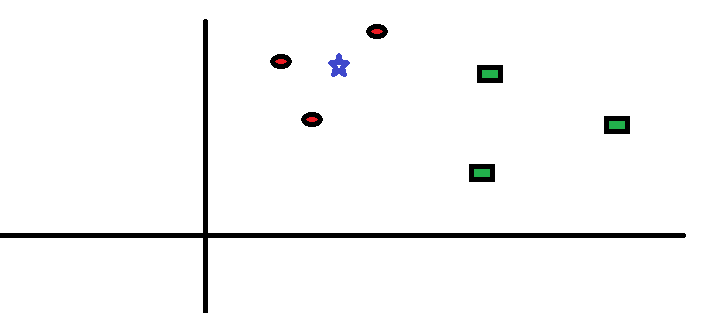
• Non-parametric means that it makes no assumptions. KNN has been used in statistical estimation and pattern recognition in 1970’s as a non-parametric technique.

• Lazy learning means that there is little learning involved while using this method. That’s why, all of the training data is also used in testing when using KNN.

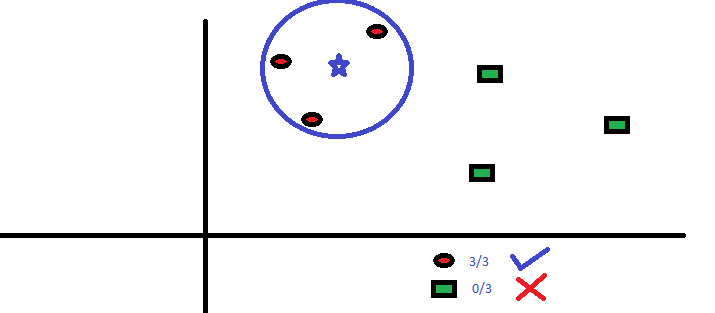
KNN algorithm is used to classify by finding the K nearest matches in training data and using the label of closest matches in training data to predict.

KNN works by finding the distances between an inquiry and all the models in the data, choosing the predefined number models (K) nearest to the query, at that point votes in favor of the most continuous name in the case of classification) or midpoints the marks (in the case of regression).

Let’s take a simple case to understand KNN algorithm. Following is a spread of red circles (RC) and green squares (GS):



You expect to discover the class of the blue star (BS). BS can either be RC or GS and that's it. The "K" is KNN algorithm is the closest neighbor we wish to take the vote from. Suppose K = 3. Thus, we will presently make a circle with BS as the center similarly as large as to encase just three data points on the plane. Refer to the following diagram for additional details:



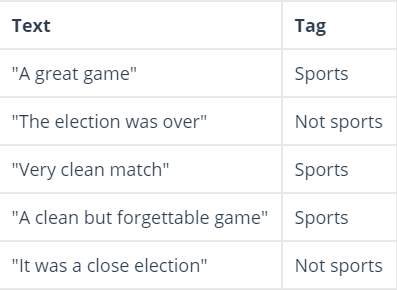
The three nearest focuses to BS is all RC. Thus, with a decent certainty level, we can say that the BS ought to have a place with the class RC. Here, the decision turned out to be extremely evident as every one of the three votes from the nearest neighbor went to RC. The decision of the parameter K is vital in this algorithm.

1. Naive Bayes Algorithm

Naïve Bayes algorithm is an administered learning algorithm, which depends on Bayes hypothesis and utilized for tackling classification problems. It is mostly utilized in text classification that incorporates a high-dimensional training dataset. Naïve Bayes Classifier is one of the basic and best Classification algorithms which help in building the quick AI models that can make fast predictions. It is a probabilistic classifier, which implies it predicts based on the probability of an object. Some mainstream instances of Naïve Bayes Algorithm are spam filtration, Sentimental examination, and grouping articles.

The name naive is utilized on the grounds that it expects the features that go into the model is autonomous of one another. That is changing the value of one element, doesn't directly impact or change the value of any of the other features used in the algorithm.

How about we perceive how this functions by and by with a basic model. Assume we are building a classifier that says whether a text is about sports or not. Our training data has 5 sentences:



Presently, which tag does the sentence a nearby game have a place with?

Since Naive Bayes is a probabilistic classifier, we need to compute the probability that the sentence "A very close game" is Sports and the probability that it's Not Sports. At that point, we take the biggest one. Composed numerically, what we need is P (Sports | a very close game) — the likelihood that the tag of a sentence is Sports given that the sentence is “A very close game”.

1. Latent Dirichlet Allocation (LDA)

In natural language preparing, the latent Dirichlet allocation (LDA) is a generative statistical model that permits sets of perceptions to be clarified by in secret gatherings that clarify why a few pieces of the data are similar.

Linear Discriminant Analysis (LDA) is most commonly used as dimensionality reduction technique in the pre-processing step for pattern-classification and machine learning applications.

In spite of the fact that the name is a significant piece, the idea driving this is basic. To tell quickly, LDA imagines a fixed arrangement of subjects. Every topic represents to a bunch of words. What's more, the objective of LDA is to map all the records to the subjects as it were, with the end goal that the words in each document are generally caught by those imaginary topics.

Let’s take a small example

Sample documents are (each line represents a document):

• I like to eat broccoli and bananas.

• I ate a banana and spinach smoothie for breakfast.

• Chinchillas and kittens are cute.

• My sister adopted a kitten yesterday.

• Look at this cute hamster munching on a piece of broccoli.

Suppose we choose k=2 (number of topics are 2) for our model:

• Topic A: 30% broccoli, 15% bananas, 10% breakfast, 10% munching, …… (we could interpret topic A to be about food)

• Topic B: 20% chinchillas, 20% kittens, 20% cute, 15% hamster, ……… (we could interpret topic B to be about cute animals)

Now some new document can be tagged with the above-given topics using the observations made by the LDA model.

1. Banana and spinach smoothie is a good combination for a healthy breakfast.

2. Kittens look cute as they munch on a bowl of milk, bananas, and chocolates.

Here, we can say that sentence 1 is 100% Topic A and sentence 2 is 40% Topic B with 60% Topic A.

LDA with KNN and TFIDF

As previously explained in this research document, we tested a combination of Latent Dirichlet Allocation (LDA) for topic modeling alongside K-Nearest Neighbors (KNN) for text classification and Term Frequency-Inverse Document Frequency (TF-IDF) for data pre-processing before making it ready for topic modeling and classification. Firstly we preprocessed our data using TF-IDF where we removed stopwords and unwanted characters like commas, fullstops, and semicolons to make it clean and ready to use. Then we converted all of our data in small letters so that we

do not get multiple instances of similar words with different patterns of capitalization. After preprocessing, then comes our topic modeling which we did using Latent Dirichlet Allocation (LDA) Algorithm to pick out one word to represent all of the data written in one bug report. This word is then pronounced to be the topic of that bug report. Last step on this path is Text classification, which we are discussing based on our experiment tested using K-Nearest Neighbors (KNN) Algorithm. In this section, our Machine Learning/Deep Learning model (ML/DL model) learns the usernames of the bug resolvers and the topic they resolved on from the training part of the dataset. This previous learning information is then tested on our testing dataset. Our Machine Learning/Deep Learning (ML/DL model) predicts the name of bug resolver using the previous information it learned. Our Machine Learning/Deep Learning (ML/DL) model picks one one bug report, preprocesses data using Term Frequency- Inverse Document Frequency (TF-IDF), picks out topic using Latent Dirichlet Allocation Algorithm and then finally plots the numeric represention of the bug report on to the graph and gets votes from K nearest bug reports. Number of K is decided by the programmers which we selected 15 so that it does not have equal count of votes on all the neighbors. This is then classified by the class with maximum number of votes. We tested this on 4 different datasets, and we are sharing the results we got after testing this model on all these 4 different datasets.

# REFERENCES

1. A Sarkar, [PC Rigby](https://scholar.google.com/citations?user=lGVxz58AAAAJ&hl=en&oi=sra), B Bartalos - 2019 IEEE International …, 2019 - ieeexplore.ieee.org
2. S Guo, X Zhang, X Yang, [R Chen](https://scholar.google.com/citations?user=IB46_P8AAAAJ&hl=en&oi=sra), C Guo, H Li… - Neural Processing …, 2020 – Springer
3. S Guo, X Zhang, X Yang, [R Chen](https://scholar.google.com/citations?user=IB46_P8AAAAJ&hl=en&oi=sra), C Guo, Li… - Neural Processing …, 2020 – Springer
4. I Alazzam, [A Aleroud](https://scholar.google.com/citations?user=vW6NVIQAAAAJ&hl=en&oi=sra), Z Al Latifah… - IEEE Transactions on …, 2020 - ieeexplore.ieee.org
5. I Alazzam, [A Aleroud](https://scholar.google.com/citations?user=vW6NVIQAAAAJ&hl=en&oi=sra), Z Al Latifah… - IEEE Transactions on …, 2020 - ieeexplore.ieee.org
6. V Dedík, [B Rossi](https://scholar.google.com/citations?user=SivTOTcAAAAJ&hl=en&oi=sra) - 2016 42th Euromicro Conference on …, 2016 - ieeexplore.ieee.org
7. I Alazzam, [A Aleroud](https://scholar.google.com/citations?user=vW6NVIQAAAAJ&hl=en&oi=sra), Z Al Latifah… - IEEE Transactions on …, 2020 - ieeexplore.ieee.org
8. S Guo, X Zhang, X Yang, [R Chen](https://scholar.google.com/citations?user=IB46_P8AAAAJ&hl=en&oi=sra), C Guo, H Li… - Neural Processing …, 2020 - Springer
9. S Guo, X Zhang, X Yang, [R Chen](https://scholar.google.com/citations?user=IB46_P8AAAAJ&hl=en&oi=sra), C Guo, Li… - Neural Processing …, 2020 – Springer