An Enhanced Convolutional Neural Network Schema for Structured Class-based Software Fault Prediction

Faisal Nabi, School of computing, Muhammad Ali Jinnah University, Karachi, Pakistan

Abstract

Malicious software detection is the most prominent process required by various industries to avoid server failure. It is required to detect malicious software accurately to avoid time and cost wastage. Various research works have been introduced earlier for the detection of malicious software. In the existing work Support Vector Machine (SVM) is introduced for malicious software detection. However, existing works cannot perform well where there are error modules in the software. It is addressed in this suggested study by developing Coupling and Cohesion Metrics based Fault Detection (CCMFD). In this research work, structural measures are mainly examined which come under the cohesion measures and comprise deficient cohesion in approaches (LCOM), and Conceptual Coupling between Object Classes (CCBO). Failure situ- ations and measures relating to information flow are used in other techniques. A high-quality service has a low coupling and a high cohesiveness. These extracted features will be given as input to the enhanced Convolutional Neural Network (CNN) for software mistake forecasting. A complete study analysis is done in a Java simulator, indicating that the suggested approach tends to have superior fault prediction outcomes than the current method.

Keywords: SVM, CNN, Software fault prediction, Coupling, Cohesion, Object classes, Failure cases, Information flow.

1. Introduction

Software quality is a vital factor for any software company. Software fault forecasting is a data mining process that helps to improve quality [1]. Software fault forecasting is a vital approach in software engineering that helps to enhance the quality of the software and verification in a short amount of time at a low cost [2]. It is deployed before the software development life cycle's testing stage. Errors or the number of failures is given by software fault forecasting methods. A lot of experts have been driven to suggest various methods with a program or cross-program to increase many qualities and tracking guarantee of software. Building a software fault forecasting system may be done in two ways: supervised and unsupervised learning [3]. The issue with supervised learning is that it requires past information or established findings to build the software fault forecasting system. The system training inside the project is done successfully, however, it creates a difficult challenge in comprehending other fresh projects. Several public databases, such as PROMISE,

Eclipse, and Apache, are freely available for investigators to help them fix the complicated issue of training for future projects. Various researchers were interested in developing a cross-project fault forecasting system using a variety of measures such as class level measures, process measures, and static code measures, yet they were unable to develop better realistic and accurate systems [4].

Nave Bias, SVM, Random Tree, J48, and Logistic Regression are some of the classification models or learning algorithms that could be used to choose a large range of software measures. Several valuable findings have been reached by such algorithms. Most of today's software forecasting approaches are based on complicated criteria that allow the system to attain a high level of accuracy. The contribution of this study is related to the present status of the investigation. It also proposed a forecasting system for feature selection based on a simpler set of indicators. It also showed that a software forecasting system built with a smaller number of measures can get a satisfactory outcome.

The machine learning technique is used in most software fault forecasting processes [5, 6, 7]. The initial stage in the prediction procedure is to identify cases in software, which can be codes, functions, classes, methods, and so on. Such cases could be generated from a variety of sources, including problem detection systems, revision control systems, and e-mail archiving. various measures are discovered by the software for each case. Such situations can be classified as buggy B (count of bugs) or clean C (amount of clean).

The initial phase of machine learning pre-processing methods used on cases is to build innovative cases of a similar kind after identifying them with the category and measurements. To extort the attributes, scale the information, and remove noise, pre-processing is used [8, 9, 10]. This is not required to use on complete fault forecasting systems [11]. Following pre-processing of the data, the fault forecasting system was trained with new data. The outcome is given to buggy and clean cases by the forecasting system. Regression refers to the number of problems in a given case. This is also called binary classification since it provides exactly two outcomes such as buggy or clean.

The main contribution of the research is to examine the software to predict faults accurately. This is done with the help of coupling and cohesion metrics which will be extracted from the software. These features will be learned using a modified convolutional neural network for the final forecasting of software problems. The following section depicts the general structure of the study method: The second section discusses the different related research projects that have been completed. Section 3 delves into the suggested study technique in depth, including relevant examples and explanations. The empirical analysis of the suggested research study is presented in section 4. Lastly, the entire study is concluded in section 5 with appropriate performance criteria.

2. Related review

This section highlighted the recent works which are related to the proposed technique and stated the limitations of existing as follows,

Bibi et al. [12] used Regression through Classification (RvC) to determine the number of software defects with a confidence interval in a software fault forecasting task. RvC outperformed typical regression algorithms in terms of regression error. The investigation was conducted using the Pekka database from a large Finland commercial bank and the ISBSG (International Software Benchmarking Standards Group) database. The Mean Absolute Error (MAE) was used to evaluate performance.

Using two databases, Bingbing, Qian, Shengyong, and Ping [13] employed the Affinity Propagation clustering technique and compared the results to that of the K-means clustering approach. Brendan J. Frey and Delbert Dueck created Affinity Propagation. As per the Type-II error, the approach outperformed K- means clustering on two databases. Type-I error, Type-II error, and overall correct classification rate (CCR) were used to evaluate performance.

Catal and Diri [14] concentrated on high-performance defect forecasters depending on machine learn- ing, like Random Forests, and techniques relying on Artificial Immune Systems, a novel computational intelligence technique. NASA databases were utilized for this study. Because of the Area under Receiver Operating Characteristics Curve (AUC) assessment measure, they found that Random Forests has the highest recognition rate for huge databases while Naive Bayes has the greatest recognition rate for small databases.

Turhan and Bener [15] demonstrated that the Naive Bayes system's interpretation is not adverse to PCA pre-processing, and they were using PD, PF, and balancing factors in the research. To uncover fault patterns, Chang, Chu, and Yeh [16] suggested a fault forecasting method assumed from association rules. It stated that the forecasting findings were outstanding. The advantage of the system is that the fault patterns observed can be employed in a causal investigation to understand the sources of defects.

Tosun, Turhan, and Bener [17] used publicly available resources to evaluate Zimmermann and Nagappan's article from ICSE'08. The investigation was conducted using three embedded software projects. For complex projects, network metrics are key markers of fault-prone components, according to the researchers. PD, PF, and accuracy were the quality assessment measures. Turhan, Kocak, and Bener [18] looked into 25 telecommunications projects and used the NASA MDP dataset to build classifiers. To create fault classifiers, they employed static call graph-based ranking (CBGR) and nearest neighbor samples. Researchers foundthat by evaluating just 6% of code using the Naive Bayes method and 3% of code with the CBGR method, a minimum of 70% of problems are recognized.

Okutan A & Yıldız [19] utilized Bayesian systems to decide the probabilistic persuasive connections among coding measurements and inaccurate prone. Notwithstanding the measurements utilized in the Promise data store, the study characterizes two additional measurements, for example, Gesture for the number of designers and LOCQ for the source code quality. The study removes these measurements by assessing the source code archives of the chosen Promise data storehouse informational indexes. Toward thefinish of the modeling, the study gained proficiency with the minor inaccurate prone likelihood of the entire coding framework, the classification of best measurements, and the compelling connections among measure-ments and inadequacy. The investigations on nine open-source Promise data store informational indexes demonstrate that reaction for class, lines of code, and absence of coding quality are the best measurements though coupling between items, weighted strategy per class, and absence of attachment of techniques are less compelling measurements on defect prone. Moreover, many youngsters and the profundity of a legacy tree have exceptionally constrained impact and are conniving. Then again, because of the tests on Poi, Tomcat, and Xalan informational indexes, the study sees that there is a positive connection between the number of engineers and the dimension of fault.

Software defect prediction majorly focuses on the components which are prone to defects just before the commencement of testing [20]. Generally, the fault prediction models are merged with the prediction values to increase the performance of the prediction [21]. In [22], the authors proposed a new model which improves the efficiency and the way we can apply the methods of fault prediction. In [23], the authors have identified 54 different metrics of inheritance, 78 public datasets along with various other combinations. The incorporation of various ensembling techniques in predicting the software faults and defects [24]. The application of Weka in both software and machine learning was explained in [25].

The authors in [26] have proposed a model which was implemented on various datasets taken from dif- ferent repositories like Promise and Eclipse. The evaluation of various fault prediction models was discussed in [27] to evaluate and identify the fault-prone modules effectively. The ideology of systematic usage of combinatorial coverage with respect to characterizing different training sets and test sets for various ma- chine learning models was demonstrated in [28]. In [29], the authors compared various oversampling methods with several class imbalance techniques and different baseline models. Finally, in [30], The authors explained coupling and cohesion metrics-based fault prediction (CCMFD) which was developed by taking structural measures by taking Convolutional Neural Networks (CNN) into consideration.

2.1. Research gaps identified in the Literature

The following points are gathered by conducting various experiments and scientific workflow proof with a literature survey, as a research gap for the software fault prediction.

- If the software project team is not tested and handed over to the customer, then the customer will receive the faulty product in the end. An error may start to creep into the system from the requirementphases and this falls till the coding phase. Risk is the likelihood that a program fault will result in a negative impact on the software business.

- The debugging process can occur the test cases are started to execute, test execution results are assessed and the differences between the actual and the expected behavior are identified and analyzed. Once the root cause of the problem is diagnosed then the developer will correct and remove the faults from the software.

- Care should be taken while performing the debugging process to make sure the error is properly removed and does not introduce any new kinds of errors in the system.

The paper aims to automate the failure prediction process, structural measures are mainly examined which come under the group of cohesion measures and comprise deficient cohesion in approaches LCOM, and CCBO. Failure situations and measures relating to information flow are used in other techniques. A high- quality service has a low coupling and a high cohesiveness. These extracted features will be given as input to the enhanced CNN for software mistake forecasting.

3. Proposed System (Software Fault Prediction System)

In object-oriented (OO) software systems, coupling, and cohesion metrics capture the extent of commu- nication and interactions among source code components such as methods and parameters. In an object- oriented system, classes have a higher level of cohesiveness and are less tightly coupled. These characteristics make it simple to comprehend test coverage, repeatability, and serviceability. It starts with Conceptual Cou- pling between Object Classes (CCBO), which is founded on the well-known CBO coupling measure, and then moves on to Conceptual Lack of Cohesion in Methods (CLCOM5), which is focused on the LCOM5 coupling measure. The examined unstructured data inherent in the source code, such as remarks and identifiers, are used to create a new metric for class cohesion and coupling in OO software applications. The textual coher- ence is estimated using the Conceptual Cohesion of Classes (C3) metric. C3 is based on the examinations of textual information of source code, represented in remarks and identifiers. To extort, interpret, and evaluate textual features from input code, Latent Semantic Indexing (LSI) is used. The measurement of cohesiveness could be interpreted as a metric of a class's textual coherence within the context of the entire system; the designer generates remarks for future analysis. The processing flow of the proposed work is shown in Figure 1.

3.1. Coupling

Chidamber and Kemerer describe RFC (Response for a Class) as a combination of protocols that aclass provides to the customers in addition to protocols it desires from further classes. Gauging the entire transmission probable, the extent is associated with coupling and is not free of coupling amid the class.

- **Strength 1:** Get into the boundary of whichever server class SC, specified SC is a consistent class/attributes however a steady interface, inoffensive kind of Class coupling happens since no change reliance is presented.

- **Strength 2:** Varying the boundary of an SC technique named through an object local to one among the CC's techniques, simply the final technique must be modified respectively. identical dispute usesto the instance where SC is a kind of factor of CC technique.

- **Strength 3:** Varying the boundary of an SC technique induced through a note forwarded to one among the CC's case variables of class SC, because of the class range of case variables, possibly the entire approaches of CC are disturbed. Likewise, modifying the border of a technique of super class SC of CC has an emotional impact on the entire technique of CC called the superclass technique. Therefore, once more the whole approach of CC might be disturbed. Since a global variable is easy to
  get to from the entire techniques of a class, a similar dispute uses for global variables, also.

- **Strengths 4 and 5:** After the similar disputes as of strengths 2 and 3 and observing that modifying dependencies are more solid while getting through the info hiding standard, these task outcomes.

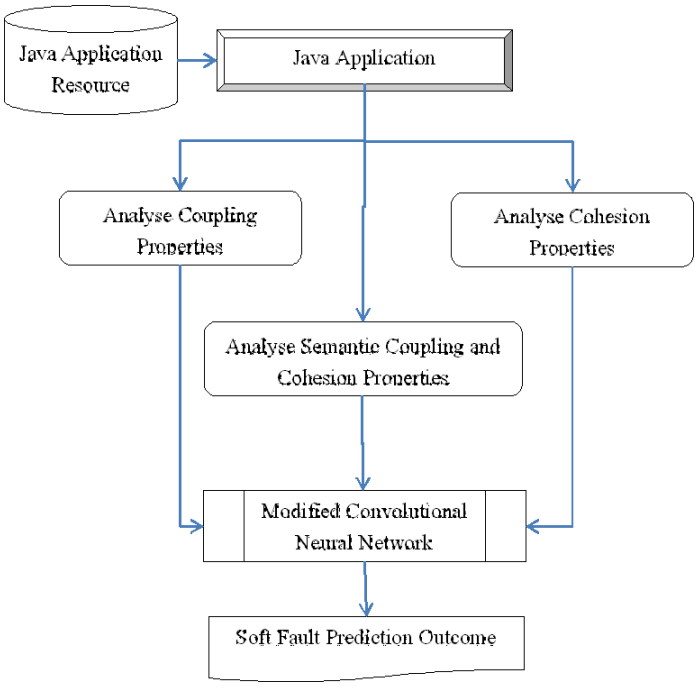

Figure 1: Processing flow of the proposed methodology

3.2. *Cohesion*

Cohesion is a significant feature relating to abstraction quality grasped by class beneath deliberation. superior abstractions naturally reveal higher cohesion. The real object-oriented cohesion measure as specified by Chidamber and Kemerer (illuminated by similar authors) signifies a converse metric for cohesion. it describes Lack of Cohesion in Methods (LCOM) as the number of pairs of approaches functioning on a disjoint group of case variables, condensed by the number of approach groups performing on a minimum of one shared instance variable. The significance specified is replicated here:

Let $C1$ class with n methods $M1, M2, ., ., Mn$.

Consider $\{Ij\}$ = group of instance variables utilized by Method $Mj.n$ similar sets $\{I1\}... \{In\}$

Consider $P = \{(Ii, Ij) \mid Ii \cap Ij = \emptyset\}$

$Q = \{(I_i, I_j)$ $I_j = .\emptyset\}$ When entire n sets $\{I_1\}... \{I_n\}$ are $\emptyset$, consider $P = \emptyset$ $|-|Q.|$

I $.LCOM = |P$

i

When $|P| > |Q| = 0$ or else. hence, LCOM is 2 - 1 = 1.

Even if the source notion behindhand here looks extremely reasonable, the ensuing cohesion measure reveals numerous irregularities relating to the natural consideration of feature, the high significance of whichwould be elucidated here. Lack of Cohesion in Methods measures computations.

- **LCOM 1:** Consider every group of methods in the class in addition to identifying the group of fieldseach of them gets into. When they are disjointed groups of field admissions, the number R rises by

  1. When they contribute to a minimum of 1 field admission, S rises by 1. Subsequently taking every group of methods as per Eq. 1:

$$Result = (R > S)?(R - S) : 0 \tag{1}$$

Less rate specifies increased coupling amid methods. It specifies the possibly increased consistency as well as a decent class plan.

- **LCOM 2:** It is an enhanced edition of LCOM1. specify subsequent objects in the class:
  - *me*: Total methods in the class.
  - *ac*: Total attributes in the class.
  - *meA*: Total methods which access the attribute *a*.
  - *sum*(*meA*): Total *meA* on the whole the attributes in the class.
  - *mPr*: Total private functions in a class.
  - *mPub*: Total public functions in a class.
  - *mPro*: Total protected functions in a class.
  - *mPr* + *mPro*: Total (*mPr* + *mPro*) on the whole of the attributes in the class as per Eq. 2.

$$LCOM2 = 1 - \frac{sum(meA)}{(me * ac)} \tag{2}$$

  When the number of methods or variables is zero (0) in a class, LCOM2 is indeterminate asshown as zero.

- **LCOM 3:** it is an additional enhancement on LCOM1 and LCOM2, stated like as per Eq. 3:

$$LCOM3 = me - \frac{sum(meA)}{ac} \Big/ (me - 1) \tag{3}$$

here *me, ac, meA, sum*(*meA*) are similar to LCM2. subsequent things must be noticed regarding LCM3: LCOM3 value changes amid [0, 2]. LCOM3>1 specifies a deficiency of cohesion and is taken as a type of alert. When just a single method exists in a class, LCOM3 is indeterminate, as well when no attributes exist in the class LCOM3 is as well indeterminate, stated as zero. Every diverse measure of LCOM contains a distinctive means to compute the LCOM value. A great deficiency in cohesion for instance LCOM3>1 specifies that a specific class must be divided into more than two classes. When complete attribute members in a class simply gain access to the external of the class and do not ever gain access within the class, LCOM3 would display an increased value. marginally increased value of LCOM denotes we could enhance the structure by dividing the classes or reorganizing some functions within the class group.

- **LCOM4:** It is an additional enhancement on LCOM1, LCOM2, and LCOM3. It's computed as perEq. 4

$$LCOM4 = \frac{me - [sum(meA) - sum(mPr + mPro)]}{me - 1} \tag{4}$$

Here *me, ac, meA, sum*(*meA*), *mPr, mPub, mPro* are similar to LCOM2.

- **LCOM5:** It is one more enhancement of LCOM, LCOM2, LCOM3, and LCOM4. indicated as perEq. 5:

$$LCOM5 = \frac{\{[1 - (u(Aj))] - m\}}{(1 - m)} \Big/ a \tag{5}$$

Here, *a* is a $\frac{total\ attributes}{instance\ variables}$, $u(Aj)$ is a total method that has the right to use attribute Aj and a total of complete attributes $j = 1 - n$, and m is a total method in Class.

The novel proposed metric for cohesion and coupling of classes in OO software systems depends upon the inspection of non-structured info implanted in source code, like remarks as well as identifiers. Metric tagged as Conceptual Cohesion of Classes (C3), is employed to guesstimate the coherence of the text. C3 depends on the examination of the text information in the source code, stated in remarks and identifiers. Latent Semantic Indexing (LSI), to excerpt, signifies, examining the text info from the source code. The cohesion metric is transformed as a metric of textual coherence of a class within the structure of the complete model, the remarks are produced by the designer for the forthcoming study.

3.3. Conceptual Cohesion & Coupling Metrics

The novel theoretical cohesion, as well as class coupling, are described as follows: the source code of the software system is parsed as well as transferred into a collection of textual information, each article relating to the development of a procedure. LSI technique uses the corpora as an input and generates a word-by-document matrix that captures word dispersion and co-occurrence in class functions. the LSI subspace is created using SVD. In the LSI subspace, all of the procedures in these matrices are represented as vectors. A similarity of cosine between two vectors is used as a measure of conceptual overlap between the two approaches, and it is assumed to reveal shared conceptual information between the two methods in the entire software application. The Conceptual Coupling of Classes and Conceptual Cohesion of Classes metrics both show this method for detecting conceptual similarity among texts. There are some definitions for the CCBO and CLOM5 models.

3.4. Principal Definitions

- **Definition1 for System, Class, Method:** Here describes an OO system as a class group $C = c1, c2, ..., cn$ with total classes as $n = |C|$. A class contains a group of functions. each class $c \in C, M(c) = m\{1, ..., mt\}$ signifies its group of functions, here $t = M(c)$ known as total functions in class c. collection of functions in a system is represented as $M(C)$.

  OO system C is denoted as a collection of connected graphs $GC = G1, .., Gn$ where Gi signifying class ci. Every $ci \in C$ is denoted by graph $Gi \in GC$ with the intension $Gi = (Vi, Ei)$, here $Vi = M(ci)$ is collection of vertices equivalent to functions in class $ci \& Ei \subset Vi$, vi is collection of weighted edges, which associate pairs of functions from class.

- **Definition2: (Conceptual Similarity among Functions):** Conceptual similarity amid functions/methods (CSM) $mk \in M(C)$, $mj \in M(C)$, $CSM(mk, mj)$, is calculated as cosine amongst two vectors vmk and vmj, signifying mk and mj in LSI semantic space as per Eq. 6:

$$CSM(m_k, m_j) = \frac{vmkT \times vmj}{|vmk|^2 x |vmj|^2} \tag{6}$$

  By the indication, the value of $CSM(mk, mj)$ [-1, 1], as CSM known as a cosine similarity in LSI space. With the purpose of the satisfying non-negativity feature of software measures, enhanced CSM is as per Eq. 7:

$$CSM1(mK, mj) = CSM(mk, mj); if\ CSM(mk, mj) > 0, 0 = else. \tag{7}$$

  CSM1 is utilized as a standard for describing C3 [39], CoCC [40] metrics.

- **Definition 3: (Parameterized Conceptual Similarity):** This proposed work, described con- ceptual cohesion as well as coupling measures using the counting process, starting from previous structural measures that are subtle to source info for instance nodes and edges (such as functions and attribute indication). Therefore, here presented an idea of a parameterized conceptual similarity that discriminates amid important and unimportant conceptual connections amongst functions of a class. Especially, predicted that it is likely to experientially get a threshold value for a specified software scheme to differentiate between sturdy and poor conceptual resemblance. further officially, outlined parameterized CSMP is as per Eq. 8:

$$CSMP(mk, mj, t) = 1; if\ CSM1(mk, mj) > t \quad 0 = else. \tag{8}$$

Specific threshold *t* based upon particular software structure. Based on past knowledge, the com- plete value of the cosine similarity could not be utilized as a consistent pointer of the existence or nonexistence of conceptual associations amongst pairs of functions as further complete examination of similarity distributions is essential. Among the key investigation queries in the experimental as- sessment is on experimentally deriving like threshold and examination of the effect of the selection of threshold on the ensuing measures.

3.5. Conceptual Lack of Cohesion in Classes

CLCOM5 utilizes CSMP as a basis for calculating conceptual resemblance amongst functions of classes, on the other hand, with the counting approach we depend upon the idea from formerly stated structural metrics, known as LCOM5, graph-dependent cohesion measure. key dissimilarity amid measure, conceptual cohesion of classes measures CLCOM5 and C3, is described a parameterized edition of cohesion measure using a diverse including technique:

$CLCOM5(c, x) = NoCC(G)$,

here, NoCC detects total associated elements in the graph $GC = (M(c), E), c \in C, E \in M(c), (mk, mj) \in E$ when $CSMP(mk, mj, t) = 1$.

3.5.1. Conceptual Coupling between Object Coupling

Description of CCBO based upon former description for CoCC measure. Here stated the descriptions elucidate how attuned them in the present research. Consider $ck \in C$ and $cj \in C$ are 2 discrete $(ck \neq cj)$ classes in the model. Every class contains a group of functions $\{mk1, \ldots, mkr\}$, here $r = |M(ck)|$ and $M(cj) = \{mj1, \ldots, mjt\}$, here $t = |M(cj)|$. Amid each pair of functions (mk, mj) a similarity measure $CSMP(mk, mj)$ exists. There could likewise describe the conceptual resemblance between two classes cj and ck, i.e., CSCP, as per Eq. 9:

$$CSCP(ck, cj, t) = 1 \; if \; CSC1ck, \; cj \geq t \; 0 = else. \tag{9}$$

Description guarantees that conceptual resemblance between two classes is balanced, like $CSC(ck, cj) = CSC(cj, ck)$. Here, utilized class granularity for constructing the quantity. It is the key dissimilarity in calculating CLCOM5 and CCBO measures. here enhanced the conceptual resemblance for a class c as perEq. 10:

$$CCBO(c, t) = \sum ck \in C, c \; ckCSCP(c, ck, t) \tag{10}$$

It is the summation of the parameterized conceptual resemblance amid class c and additional classes in the model.

3.6. Software Fault Prediction using Enhanced CNN

Classification is done using a modified convolutional neural network is used for the classification of the dataset. In this work, there are three layers applied to ensure computation overhead reduced accurate prediction. These are the input layer, convolution layer, pooling layer, and finally soft max or fully connected layer. CNN is usually composed of two parts. In part 1, the convolution operation is used to produce deep attributes of unprocessed information. furthermore, in part 2, the attributes are connected to MLP for categorization. Here are some details for each layer:

- **Input layer:** $N \times k$ neurons make up the input layer, where k signifies total input data samples and N specifies the total of each data sample.
- **Convolutional layer:** Convolution filters are used to conduct convolution functions on the data of the previous layer. Filter numbers m, convolution strides s, and filter size $k \times l$, wherein k specifies the varying amount of the data in the previous layers and l specifies the length of the filter, are some of the filter factors that should be set beforehand based on domain expertise or simply by trials. Inside this layer, a nonlinear transformation function f must also be established. For example, if the previous layer has k-variate information and the length of every information is N, following the

$s+1$

convolutional process, obtain m-variate data and the length of every uni-variate is $\frac{N-1}{}$.

- **Pooling Layer:** Pooling layers or down-sampling layers perform dimension reduction, minimizing total attributes in the input sample. The pooling method spreads a filter throughout the entire sample data, with the exception that this filter has no weights. Rather, the kernel uses an aggregation function to populate the output array with the values from the receptive field.

- **SoftMax layer or Fully Connected Layer:** A squashing function is a sort of SoftMax function. The SoftMax function estimates the event's probability distribution over n distinct events. The function, generally, will compute the probability of each target text across all potential target texts. The estimated possibilities are subsequently used to determine the target text for the input data. The most significant benefit of adopting SoftMax is the wide range of output possibilities. This range
would be 0 to 1, and the aggregate of all modifications will be sufficient. When the SoftMax function is employed for a multi-classification system, it delivers the possibilities of each class, as well as a high likelihood for the target text. The equation calculates the exponential (e-power) of a particular initial value, as well as the total exponential values for the entire input data. The result of the SoftMax functions is the proportion of the exponential initial value to the total exponential values. It's employed for multi-categorization tasks as well as within neural network layers. The maximum level has a better chance than the other values. There are certain limitations to SoftMax layers when it comes to evaluating multi-class probability. when total classes increase, SoftMax could become highly expensive. Candidate sampling could be a more efficient solution in certain cases. A SoftMax layer would limit the availability of its computations to a certain number of classes via candidate sampling.
- **Output layer:** The output layer has n neurons that correlate to n different attribute classes. It's completely linked to the feature layer. In a categorization job, the most common strategy is to use the highest output neuron as a class label for the input emotion.

The difficulty of CNN to encode Orientational and relative spatial relationships, as well as view angle, is one of its primary flaws. The position and orientation of data are not encoded by CNN. Inability to be spatially consistent with the input data sample. The CNN is learned through a series of training instances $((x1, y1), (x2, y2), ..., (xN, yN))$ with $xt \in (N \times k)$, $yt \in R^n$ for $1 \leq t \leq N$. The network receives the high-order
attributes xt as input, whereas the vector yt represents the goal output. The network can be trained using the steps outlined below:

- **Step 1** Create the network's foundation. Establish the CNN architecture, which includes the Convo- lutional and SoftMax layers. Set the number of neurons in the input and output layers based on the categorization problem. Reset the CNN parameters to their default values. Use a modest arbitrary to start the weights and bias. Choose a learning rate r and an activation function f, with the sigmoid function as an instance (Eq. 11):

$$f(x) = sigmoid(x) = \frac{1}{(1 + e^{-x})} \tag{11}$$

- **Step 2** selects a training sample from the training set arbitrarily.
- **Step 3** Select optimal values of bias and weight value using the Practical Swam Optimization (PSO) algorithm. In this work, Particle Swarm Optimization Algorithm based bias and weight-optimized CNN is proposed. When tuning the weights and biases of CNN, a total number of tuning parameters should be calculated based on the CNN structure. Each individual in PSO then seizes candidate
results equivalent to total tuning parameters. For every iteration, CNN computes an output to the difficulty based on parameters indicated through PSO. A separate cost function has to be defined that compares the deviations between output and real target data. PSO minimizes the cost function in several iterations until no additional improvements can be done after which optimization is terminated. Optimized values are replaced in the final CNN structure.
Initialize parameter values
produce starting population
While $i <$ MaxIteration & Bestfitness $<$ MaxFitness **do**
Fitness

computation
Update *pBest*
Update *gBest*
End while
Return the best solution

- **Step 4** Calculate the solution of every layer.
  - The solution of the convolutional layer is mentioned as per Eq. 12

$$Cr(t) = f\left(\sum_{i=1}^{l}\sum_{j=1}^{k} x(i + s(t-1), j)wr(i, j) + b(r)\right) \tag{12}$$

here $x \in R^{N\times k}$ denotes the input higher-order features or result of the previous layer, s indicates convolution stride, $Cr(t)$ = is the t^{th} element of r^{th} feature map, $wr \in l \times k$, $b(r)$ indicates weights and bias of the r^{th} convolution filter.

  - The result of the output layer is given as per Eq. 13

$$O(j) = f\left(\sum_{i=1}^{M} z(i)wf(i, j) + bf(j)\right); j = 1, 2, ..., n. \tag{13}$$

here z indicates the last feature map in the feature layer, bf is a bias of the final layer, and $wf \in R^{M\times n}$ refers to the association weights among feature and final layers. As a result, the mean- square error is given as per Eq. 14,

$$E = \frac{1}{2}\sum_{k=1}^{n} e(k)^2 = \frac{1}{2}\sum_{k=1}^{n} (o(k) - y(k))^2 \tag{14}$$

- **Step 5** revises the weights and biases through a gradient descent system as per Eq. 15.

$$p = p - \eta\frac{\partial E}{\partial p} \tag{15}$$

here p is the parameter value, and p indicates wr, wf, b, or bf in CNN.

- **Step 6** Choose another training sample and move to Step 3 till the entire samples in the training set are learned.
- **Step 7** Increase the iteration value. If the iteration value is equivalent to the highest value that is set earlier, end the procedure. or else, move to Step 2. Based on the above steps the software faults are classified.

In this proposed work, the performance of the CNN is improvised by introducing the optimal parameter selection, in which CNN parameter values will be selected more optimally using the PSO algorithm. This optimal parameter selection process would lead to an accurate and efficient classification outcome. The most crucial phase in the CNN classification model is the parameter value estimate, which seems to produce the best classification results. correct parameter value choosing will result in correct decision- making. For the accuracy and optimal choosing of parameter values, a specific database is separated into three subgroups in this research.

4. Results and Discussion

The datasets employed here come from the NASA Metrics Data Program (MDP) database. This work is implemented in Java programming Language and six CK metrics are taken as input, and output is the fault prediction accurateness rate essential for designing the software. The performance of the proposed Coupling and Cohesion Metrics based Fault Detection (CCMFD) model is matched up with the previous techniques of SVM and ANFIS approaches. The whole research is implemented in the Java simulation environment and diverse types of faults are studied and identified to assess the system's performance.

Figure 2 provides the evaluation of the precision and accuracy of SVM, ANFIS, and the suggested CCMFD system. From Figure 1, it is proved that the research method provides greater measurements

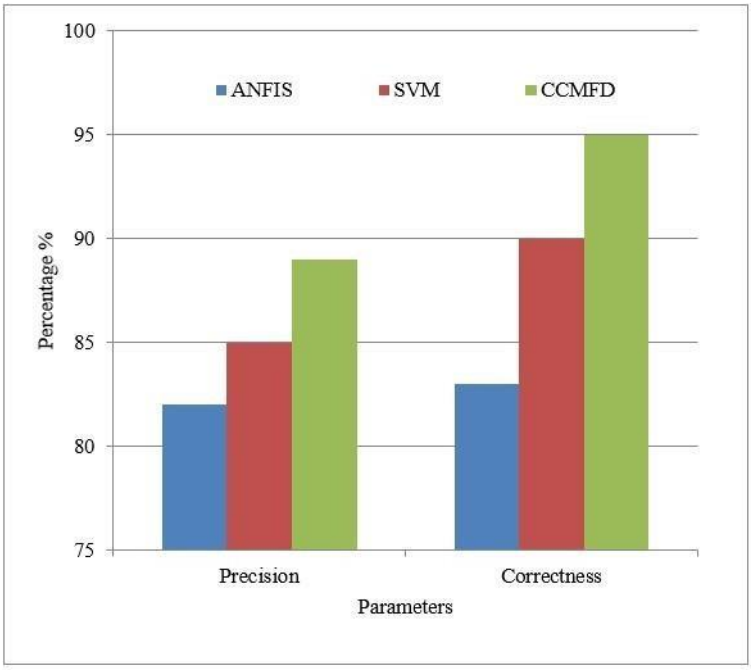

Figure 2: Precision and correction comparison of proposed Vs. traditional

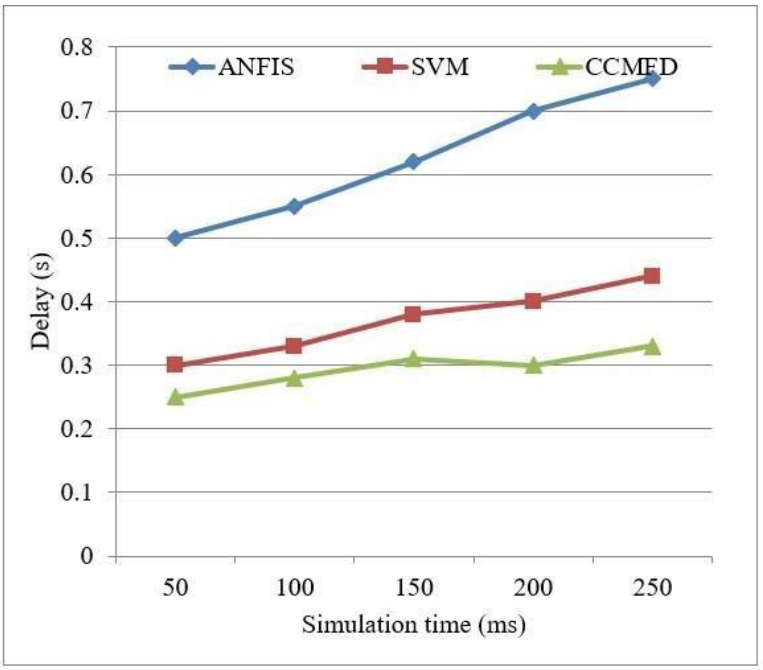

Figure 3: Delay analysis of proposed Vs. traditional

and guesstimates the faults with greater precision as well as superior correctness. So, it is confirmed that the presented technique CCMFD provides 5% improved precision compared to ANFIS and 9%

improved precision compared to the SVM technique. The research technique CCMFD provides a 6% superior outcome

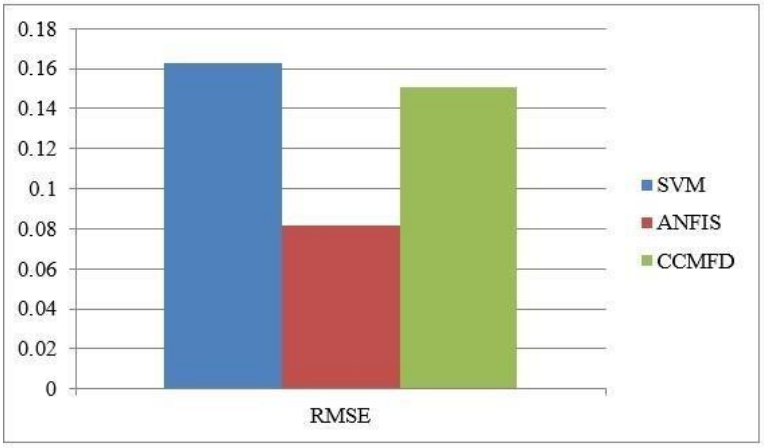

Figure 4: Root Mean Square comparison of proposed Vs. traditional

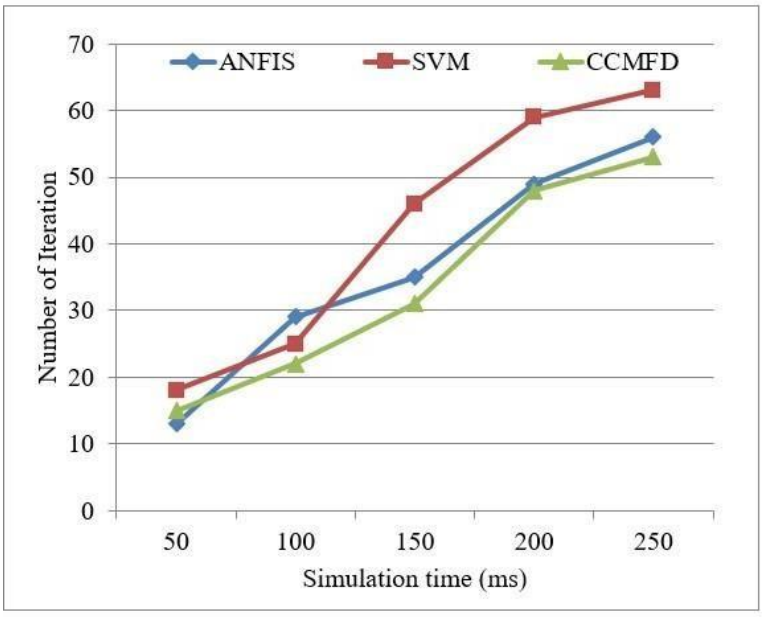

Figure 5: The number of iteration comparisons of proposed Vs. traditional

compared to ANFIS and a 14% improved outcome compared to the SVM technique.

Figure 3 shows the evaluation of the delay of the SVM, ANFIS, and the research method CCMFD. While the simulation time is 200 seconds, the proposed CCMFD model contains a delay of 0.3 seconds which is 25% and 58% below ANFIS as well as the SVM model. According to Figure 3, it is exposed that the proposed method provides improved measurements and guesstimates the faults with minimum delay.

Figure 4 shows the evaluation of the RMSE of the SVM, ANFIS, and the research method CCMFD. According to Figure 4, it is exposed that the proposed method provides improved measurements and guessti-mates the faults with a better RMSE value.

Figure 5 shows the evaluation of the RMSE of the SVM, ANFIS, and the research method CCMFD. According to Figure 5, it is exposed that the proposed method provides improved measurements and guessti-mates the faults with a smaller number of iterations.

5. Conclusion & Future Scope

In this paper, structural measures are commonly examined which come under the type of cohesion mea- sures and comprise the Lack of Cohesion in Methods (LCOM), and Conceptual Coupling between Object Classes (CCBO). further approaches employ fault instances and information flow-dependent measures. re- duced coupling and greater cohesion provide a high-quality result. These extracted features will be given as input to the Modified Convolutional Neural Network for software error forecast. The overall analysis of the work is done in a Java simulator where it is indicated that the suggested approach tends to have improved fault prediction outcomes than the present method. In the future, Java applications can be further divided into micro levels to find the data flow-based connectivity. Instead of a classification algorithm, in the future, authors can utilize an optimization algorithm to choose the best component in the proposed that can cause significant variation in the outcome.

Conflict of Interest

The authors declare no conflict of interest.

References

[1] Steidl, D., Deissenboeck, F., Poehlmann, M., Heinke, R., & Uhink-Mergenthaler, B. (2014, September). Continuous software quality control in practice. In Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on (pp. 561-564). IEEE.

[2] Miguel, J. P., Mauricio, D., & Rodríguez, G. (2014). A review of software quality models for the evaluation of software products. arXiv preprint arXiv:1412.2977.

[3] Laradji, I. H., Alshayeb, M., & Ghouti, L. (2015). Software defect prediction using ensemble learning on selected features. Information and Software Technology, 58, 388-402.

[4] Kumar, R., & Gupta, D. L. (2015). Software Fault Prediction: A Review.

[5] Bacchelli, M. D'Ambros, and M. Lanza. Are popular classes more defect-prone? In Proceedings of the 13th Interna- tional Conference on Fundamental Approaches to Software Engineering, FASE'10, pages 59– 73, Berlin, Heidelberg, 2010. Springer-Verlag

[6] E. Engstrom, P. Runeson, and G. Wikstrand. An empirical evaluation of regression testing based on fix cache ¨ recommen- dations. In Software Testing, Verification and Validation (ICST), 2010 Third International Conference on, pages 75–78, April 2010.

[7] N. Nagappan and T. Ball. Use of relative code churn measures to predict system defect density. In Proceedings of the 27th international conference on Software engineering, ICSE '05, pages 284–292, 2005

[8] S. Kim, H. Zhang, R. Wu, and L. Gong. Dealing with noise in defect prediction. In Proceeding of the 33rd international conference on Software engineering, ICSE '11, pages 481–490, New York, NY, USA, 2011. ACM

[9] T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. IEEE Trans. Softw. Eng., 33:2–13, January 2007.

[10] J. Nam, S. J. Pan, and S. Kim. Transfer defect learning. In Proceedings of the 2013 International Conference on Software Engineering, ICSE '13, pages 382–391, Piscataway, NJ, USA, 2013. IEEE Press.

[11] T. Jiang, L. Tan, and S. Kim. Personalized defect prediction. In Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on, pages 279–289, Nov 2013.

[12] Bibi, S., Tsoumakas, G., Stamelos, I., & Vlahvas, I. (2008). Regression via classification applied on software defect estimation. Expert Systems with Applications, 34(3), 2091–2101
[13] Bingbing, Y., Qian, Y., Shengyong, X., & Ping, G. (2008). Software quality prediction using affinity propagation algorithm. In IJCNN 2008 (pp. 1891–1896).
[14] Catal, C., & Diri, B. (2009a). Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problems. Information Sciences, 179(8), 1040–1058
[15] Turhan, B., & Bener, A. (2009). Analysis of Naive Bayes' assumptions on software fault data: An empirical study. Data Knowledge Engineering, 68(2), 278–290
[16] Chang, C., Chu, C., & Yeh, Y. (2009). Integrating in-process software defect prediction with association mining to discover defect patterns. Information and Software Technology, 51(2), 375–384.

[17] Tosun, A., Turhan, B., & Bener, A. (2009). Validation of network measures as indicators of defective modules in software systems. In Proceedings of the 5th international conference on predictor models in software engineering (Vancouver, British Columbia, Canada, May 18–19, 2009). PROMISE '09 (pp. 1–9). New York, NY: ACM.
[18] Turhan, B., Kocak, G., & Bener, A. (2009). Data mining source code for locating software bugs: A case study in the telecommunication industry. Expert Systems and Application, 36(6), 9986–9990.
[19] Okutan, A. and Yıldız, O.T., 2014. Software defect prediction using Bayesian networks. Empirical Software Engineering, 19(1), pp.154-181.
[20] R. Jothi, "A Comparative Study of Unsupervised Learning Algorithms for Software Fault Prediction," 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS), 2018, pp. 741-745, doi: 10.1109/IC- CONS.2018.8663154.
[21] R. Shatnawi, "Improving Software Fault Prediction With Threshold Values," 2018 26th International Conference on Soft- ware, Telecommunications and Computer Networks (SoftCOM), 2018, pp. 1-6, doi: 10.23919/SOFTCOM.2018.8555818.
[22] J. Song, B. Chen, X. Li, Y. Yang, C. Liu and H. Li, "The software fault prediction model based on the AltaRica language," 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), 2019, pp. 2549-2552, doi: 10.1109/ITNEC.2019.8729235.
[23] S. R. Aziz, T. A. Khan and A. Nadeem, "Efficacy of Inheritance Aspect in Software Fault Prediction—A Survey Paper," in IEEE Access, vol. 8, pp. 170548-170567, 2020, doi: 10.1109/ACCESS.2020.3022087.
[24] Matloob, F., Ghazal, T. M., Taleb, N., Aftab, S., Ahmad, M., Khan, M. A., & Soomro, T. R. (2021). Software defect prediction using ensemble learning: A systematic literature review. IEEE Access.
[25] Hao, Z. (2021, August). Application of Open Source Data Mining Software Weka in Marketing Teaching. In EAI Interna- tional Conference, BigIoT-EDU (pp. 356-365). Springer, Cham
[26] Mangla, M., Sharma, N. & Mohanty, S.N. "A sequential ensemble model for software fault prediction". Innovations Syst Softw Eng 18, 301–308 (2022).
[27] Jorayeva, M., Akbulut, A., Catal, C., & Mishra, A. (2022). Machine learning-based software defect prediction for mobile applications: A systematic literature review. Sensors, 22(7), 2551.
[28] Cody, T., Lanus, E., Doyle, D. D., & Freeman, L. (2022, April). Systematic training and testing for machine learning using combinatorial interaction testing. In 2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW) (pp. 102-109). IEEE.
[29] S. S. Rathore, S. S. Chouhan, D. K. Jain and A. G. Vachhani, "Generative Oversampling Methods for Handling Imbalanced Data in Software Fault Prediction," in IEEE Transactions on Reliability, vol. 71, no. 2, pp. 747-762, June 2022, doi: 10.1109/TR.2022.3158949.
[30] P. Bandarupalli and P. Yalla, "Coupling and Cohesion Metrics-based Fault Predictions using Machine learning Algo- rithm," 2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC), 2022, pp. 412-418, doi: 10.1109/ICAAIC53929.2022.9792917.