AN IMPROVED CNN BASED ARCHITECTURE FOR WITHIN-PROJECT SOFTWARE DEFECT PREDICTION

A DESSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE AWARD OF DEGREE

OF

MASTER OF TECHNOLOGY

IN

SOFTWARE ENGINEERING

M. Tech (Software Engineering) HITENDRA SINGH YADAV 2020

Submitted by:

HITENDRA SINGH YADAV

2K18/SWE /008

Under the supervision of

Dr. RUCHIKA MALHOTRA

(Associate Professor)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY(Formerly Delhi College of Engineering) Bawana Road, Delhi-110042

JUNE 2020

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering) Bawana Road,

Delhi - 110042

CANDIDATE’S DECLARATION

I, Hitendra Singh Yadav , Roll No. 2K18/SWE/008 student of M.Tech (Software Engineering), hereby declare that the project dissertation titled **“An Improved CNN Based Architecture For Within-Project Software Defect Prediction ”** which is submitted by me to the Department of Computer Science and Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi HITENDRA SINGH YADAV

Date:

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering) Bawana Road, Delhi - 110042

CERTIFICATE

I hereby certify that the Project dissertation titled **“An Improved CNN Based Architecture For Within-Project Software Defect Prediction”** which is submitted by Hitendra Singh Yadav, Roll No. 2K18/SWE/008, student of Department of Computer Science Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology, is a record of the project work carried out by the student under my supervision. To the best of my knowledge this work has never been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi

Date:

Dr. RUCHIKA MALHOTRA
SUPERVISOR
(ASSOCIATE PROFESSOR)

ACKNOWLEDGMENT

The success of a Project Dissertation requires help and contribution from numerous individuals and the organization. Writing the report of this project work allows me to express my gratitude to everyone who has helped in shaping up the outcome of the project.

I express my heartfelt gratitude to my project guide, **Dr. Ruchika Malhotra,** for allowing me to do my project dissertation work under her guidance. Her constant support and encouragement have made me realize that it is the process of learning. I am highly indebted to the panel faculties during all the progress evaluations for their guidance, constant supervision and for motivating me to complete my work. They helped me throughout by giving new ideas, providing the necessary information and pushing me forward to complete the work.

I also reveal my thanks to all my classmates and my family for constant support.

HITENDRA SINGH YADAV

ABSTRACT

To improve the software quality, the software is generally tested to find out any bugs or a simple reliability test. A reliable software defect checking mechanism is a leading research topic, in the era of dependency on software's for several tasks. Many researchers used different techniques of deep learning algorithm such as CNN i.e convolutional neural networks, and deep belief networks for prediction of software defect. these algorithms failed to provide higher prediction accuracies. To overcome the issues a new algorithm for software defect prediction is required for higher accuracy and other parameters like F- and G-measure and specifically important parameter is Matthews correlation coefficient (MCC) measure. In this paper, a new modified CNN algorithm is proposed, which combines the CNN based models into one and apply concatenate algorithm under SVM i.e. support vector machine classifier. The results clearly indicate that the proposed algorithm improves the parameters and thus is a highly dependable and reliable method for software defect prediction.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

AI	Artificial Network
TN	True Negative
FN	False Negative
TP	True Positive
FP	False Positive
ReLU	Rectified Liner Unit
CNN	Covolution Neural Network
DNN	Deep Neural network
LR	Logistic Regression
ML	Machine Learning
P	Precision
R	Recall
SVM	Support Vector Machine
URL	Uniform Resource Locator
MCC	Matthews correlation coefficient

CHAPTER 1

INTRODUCTION

A product imperfection is a blunder, bug, defect, shortcoming, breakdown or mix-ups in programming that makes an incorrect or unpredicted result. Flaws are basic properties of a framework. They show up from structure or assembling, or outer condition. Programming blemishes are modifying mistakes which cause distinctive execution contrasted and expectation. The dominant parts of the issues are from source code or condescend, some of them are from the off base code producing from compilers.

For programming engineers and customers, programming issues are a peril issue. Programming abandons not just decline programming quality, increment costing yet additionally defer the advancement plan. Programming flaw foreseeing is proposed to illuminate such a difficulty.

In mining software repositories, many different approaches have been developed to predict the number and location of future bugs in the source code. These predictions can help a project manager to quantitatively plan and steer the project according to the expected number of bugs and their bug-fixing effort. But bug prediction can be helpful in a qualitative way whenever the defect location is predicted: testing efforts can then be accomplished with a focus on the predicted bug locations. All of the above-mentioned approaches use history data of a software project to predict defects in the second release. Features (or variables) are taken from raw data. Then these features (learning period) are used together with the goal values (i.e., bug or no bug) to learn a prediction model. To assess such a model, it is fed with data from another period, and the predicted values are examined with the observed ones helping to an accuracy measure.

The downside of these approaches is its temporally coarse evaluations. Generally, a bug prediction algorithm is assessed, in terms of correctness, in only one or different points in time. These selective (insular) analyses make generalizations of the prediction methods difficult: it postulates that the evolution of a project and its data is stable over time. In the proposed work the terminology within project implies that we can find defect within two versions of same project which are mainly java based in your case, and can compare the accuracy of model within project versions.

1.1 Overview

The fault prediction dataset is a group of models and metrics of software systems and their histories. The aim of such a dataset is to permit people to evaluate different fault prediction approaches and to evaluate whether a new technique is an enhancement over existing ones. PROMISE, AEEEM, ReLink, MORPH, NASA, and SOFTLAB are some of the defect datasets which are publically available to the user. In our project we had use the processes data set from the PROMISE repository. Your work is based on the some open source project which are developed using JAVA based programming methodology.

At present the growth of software based system are rising from the previous years due to its advantage. On the other hand, the quality of the system is essential prior it is delivered to end in order to improve the efficiency and quality of software development, software faults can be predicted at early phase of life cycle itself. To predict the software faults a variety of data mining techniques can be used.

Learning techniques are intended to determine whether software module has a higher fault hazards or not. In supervised learning data is extracted using the target class. If machine learning task is trained for each input with consequent target, it is called supervised learning, which will be able to provide target for any new input after adequate training. Targets expressed in some classes are called classification problem.

1.2 Research Objective

The objective is to develop a model which improves the accuracy in term of various measures such as F-measure, G-measure and MCC-measure as compared to the exiting CNN based model [2]. The proposed model using the pre-processed data set PSC [2] , this data set is passed in set of convolutional neural layers, which is a multi headed structure of CNN layers, further max pooling layers used with forward CNN layer along with dropout layer, flattering layer, dense layers and lastly a support vector machine (SVM) for classification of buggy and non-buggy data. This model is evolved by examining various permutation and combination over layers to reach the final architecture. The final architecture is supposed to give better results in term of accuracy of defect prediction in term of F-measure, G-measure and MCC-measure.

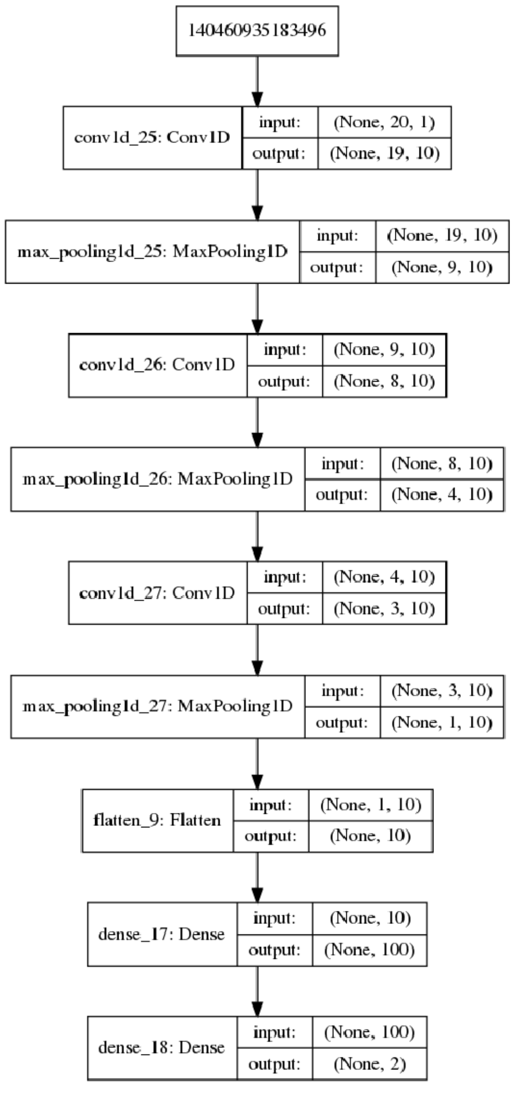
1.3 Organization of Thesis

The project report has been divided into five chapters. Each chapter deals with one component related to this thesis. Chapter 1 being introduction to this thesis, gives us the brief introduction about the project topic, thereafter chapter 2 tells about the project work carried out which further includes literature survey section. Following up is chapter 3 which tells about the proposed work carried out during the development of improved CNN based architecture. Chapter 4 focuses on the implementation work carried out and the result generated during the process. In chapter 4 we also analyzed the results in tabular and in bar chart form. Final chapter, chapter 5, which is concludes the thesis and future scope of work.

CHAPTER 2

RELATED WORK

Various researchers had contributed in making defect prediction more accurate and precise. The model proposed [2] shows the three layer CNN model along with liner regression as classifier. The Three layer model [2] explained in a flow type architecture given below:



140460935183496
conv1d_25: Conv1D | input: (None, 20, 1) | output: (None, 19, 10)
max_pooling1d_25: MaxPooling1D | input: (None, 19, 10) | output: (None, 9, 10)
conv1d_26: Conv1D | input: (None, 9, 10) | output: (None, 8, 10)
max_pooling1d_26: MaxPooling1D | input: (None, 8, 10) | output: (None, 4, 10)
conv1d_27: Conv1D | input: (None, 4, 10) | output: (None, 3, 10)
max_pooling1d_27: MaxPooling1D | input: (None, 3, 10) | output: (None, 1, 10)
flatten_9: Flatten | input: (None, 1, 10) | output: (None, 10)
dense_17: Dense | input: (None, 10) | output: (None, 100)
dense_18: Dense | input: (None, 100) | output: (None, 2)

Fig. 1: Three layer CNN model [2]

As we see in fig. 1 that after every convolution layer they have used a max

pooling layer along with other building blocks to get the desired output.
The data set used is PSC data set [2] which is a processed data set from the original PROMISE repository.
We can understand the approach adopted in model[2] via pictorial representation given below:

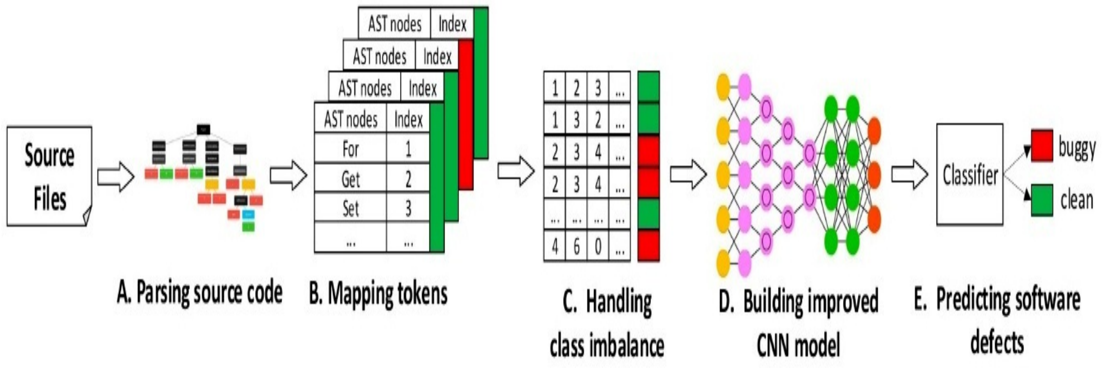


Fig. 2: Architecture used in Cong Pan CNN [2]

2.1 Literature Survey

In the literature survey, several techniques are found which are related to deep learning and machine learning for software defect prediction The software development life cycle for the most part incorporates investigation, plan, execution, test and production stages. [1] The testing stage ought to be worked viably so as to produce the without bug software to end clients. [2] Over the most recent two decades, academicians have taken an expanding enthusiasm for the software defect prediction issue, a few AI methods have been applied for progressively powerful prediction. [3] Software defect prediction is significant in software building. It utilizes the defects found in chronicled software modules to anticipate defects in new software modules, and gives choice help to arranging and procedure the board in software venture. AI (Artificial intelligence) is one of the center exploration bearings in the field of man-made brainpower and spreads numerous controls. [4] In the basic examination of the software based defect prediction algorithms, different AI techniques have been broadly contemplated and applied in various reference papers, and have been confirmed to acquire great execution. During this period of AI and ML i.e machine learning, there are numerous of basic ongoing headways in Software Engineering domain.[5] Different software designing measurements are broke down due to bug detection in the software; negative effect is achieved and the necessary predictions can be made. The software Defect prediction is basically one such movement which is of incredible importance in improving the software quality which helps software designers and analyzers to concentrate on the modules which are bound to defect inclined.

[6] Software defect prediction gives significant yields to software groups while adding to mechanical achievement. Experimental examinations in many research works have been directed on software defect prediction for both cross-venture and AI based undertaking defect prediction. [7] Be that as it may, existing examinations still can't seem to exhibit a strategy for anticipating the quantity of defects in a developed software. Software Quality is the most significant part of a software. Software Defect Prediction can legitimately influence quality and has accomplished noteworthy prevalence in most recent couple of years. [8] Defective software modules have a bad effect over software's quality prompting cost overwhelms, postponed courses of events and a lot of higher upkeep costs.

[9] Mechanized software defect prediction is an importantly significant and basic action in the area of software development. In any case, present day software frameworks are naturally huge and complex with various corresponded measurements that catch various parts of the software segments. [10] This enormous number of corresponded measurements makes constructing a software defect prediction model extremely perplexing. Along these lines, recognizing and choosing a subset of measurements that upgrade the software defect prediction strategy's exhibition are a significant yet testing issue that has gotten little consideration in the writing. [11] The fundamental goal of this paper is to distinguish critical software measurements, to fabricate and assess an autonomous software defect prediction model.

A comparison has been done between the the existing model [2] and your proposed multi headed CNN model, comparison in tabular form is shown below:

	Cong Pan CNN Model	**Multi headed proposed CNN Model**
Input layers	1	2
Convolutional Layers	3	2
Pooling layers	3	2
Activation function	ReLU + sigmoid (last dense layer)	ReLU + sigmoid (last dense layer)
Classification Function	Liner regression Function is used	SVM classifier used
Dropout Layers	1	3
Concatenation Layers	not used	Yes, 1 Layer used for concatenation

Table 1: Comparison of between Cong Pan CNN model and your Multi Headed CNN model

2.2. Background Concepts

In fig 3. A process is shown which is generally employed by researchers in previous work. The steps involve loading of software files, extracting features, training features as per bug or no bug using a suitable classifier. [14] The test sequences are then loaded on classifier which will further give output as buggy or clean software.

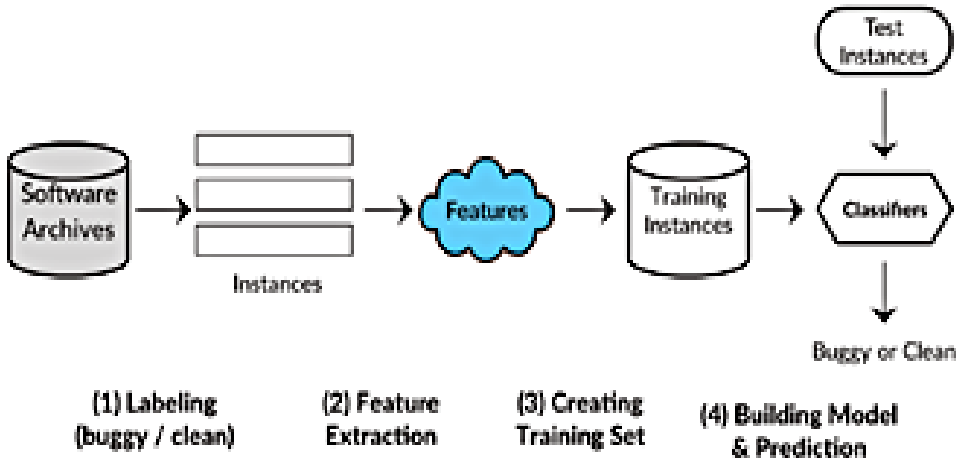


Fig. 3: . Process to detect defect in software [1]

In fig. 1, CNN based architecture is shown. It consists of layers in which convolutional process is applied. There are several layers associated with CNN like pooling, softmax, maxpool layers. Depending on the level of classification one can activate or deactivate the layers are make them fully connected or partially connected. The pooling size can be defined and number of layers can be modified. There are several activation functions used. In this work, ReLu activation function is used. The pooling size controls the sparse connectivity in CNN. The accuracy of the model can be varied by the use of the parameters such as size, connectivity pooling layers, etc[2].

2.2.1 Convolutional Neural Network

As summarized by Yoav Goldberg , “The CNN layer’s responsibility is to extract meaningful sub-structures that are useful for the overall prediction task at hand. A convolutional neural network is designed to identify indicative local predictors in a large structure, and to combine them to produce a fixed size vector representation of the structure, capturing the local aspects that are most informative for the prediction task at hand. In the NLP case the convolutional architecture will identify n-grams that are predictive for the task at hand, without the need to pre-specify an embedding vector for each possible n-gram.” The concepts used in Convolution Neural Network consist of various terminologies which are briefly defined as:

- Convolution: Applying filter to a fixed size window is the task of convolution operation.
- Convolution Filter: It is also known as convolution kernel. It is basically a matrix that is utilized for performing convolution operation.
- Pooling: It is the process of combining the vectors obtained as a result of various convolution windows into a vector single one dimension.
- Feature maps : The significance of number of feature maps is that it directly controls capacity and is dependent on count of available examples and complexity of task.

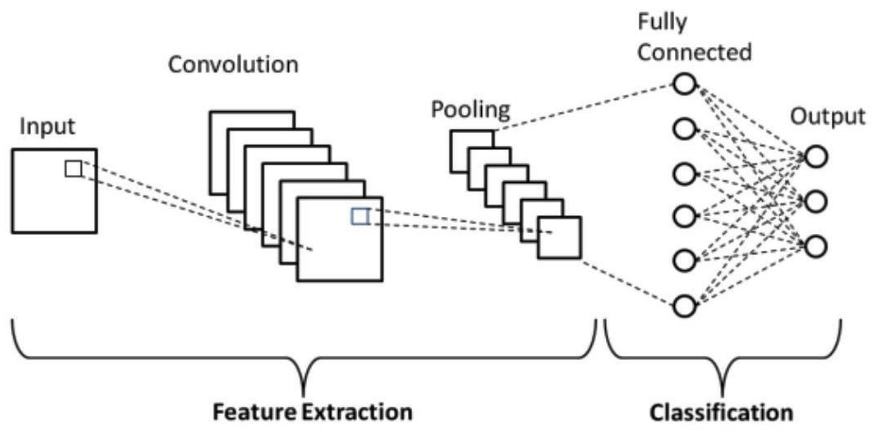


Fig. 4: Sample Convolutional Neural Network

2.2.2 **Support Vector Machine (SVM)**

It can be said that support Vectors are those data points that the margin pushes up against. In this the classifier is a separating hyper plane. The most "important" training points are the support vectors; they define the hyper plane.

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyper plane. In other words, given labeled training data (*supervised learning*), the algorithm outputs an optimal hyper plane which categorizes new examples. In two dimensional space this hyper plane is a line dividing a plane in two parts where in each class lay in either side.

Let us assume that if all data is at least distance 1 from the hyper plane, then the following two constraints follow for a training set $\{(x_i, y_i)\}$ then:

$$\mathbf{w}^T\mathbf{x}_i + b \geq 1 \quad \text{if } y_i = 1$$
$$\mathbf{w}^T\mathbf{x}_i + b \leq -1 \quad \text{if } y_i = -1$$

For support vectors, the inequality becomes an equality.

This concept is used in locating the classifier the bug or no-bug in the defect prediction in software project over the time due to certain features, which are obtained as out data sets of CNN model.

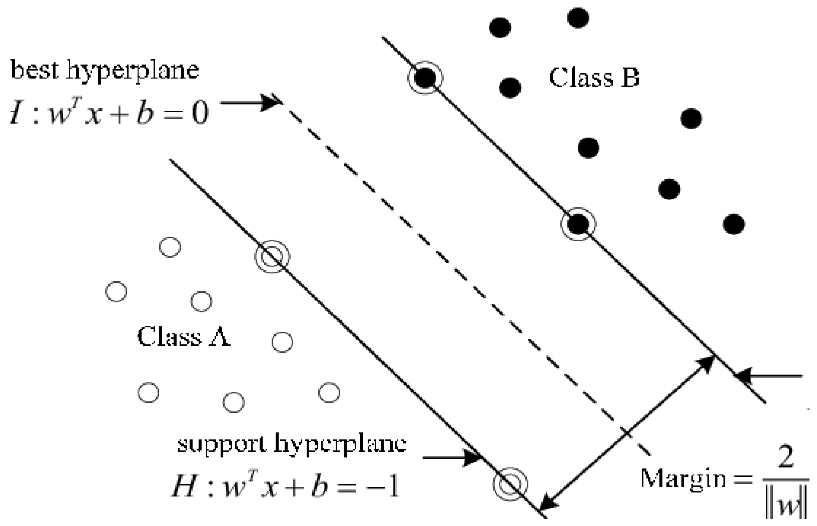


Fig 5: Diagram showing liner SVM

For the instance of multidimensional space, SVM finds the hyperplane that amplifies the edge between two distinct classes. A couple of tests control the choice limit. There are just a couple of preparing tests that touch the choice limit. These are the ones that really control the choice limit and are known as help vectors. Here the help vectors are those specks that have been circumnavigated. One of the significant points of interest of utilizing SVM arrangement is that it performs very well on datasets having numerous traits, in any event, when there are only a couple of cases that are accessible for preparing process.

2.2.3 Confusion Matrix

It maps the relation between what the model has predicted and what the actual result should be as shown in fig 6. If the predicted class is positive and actual class is positive as well, then we get the true positive section. If the predicted class is positive but actual class is negative then we get false positive section, on similar bases if the actual class is positive but the predicted class in negative then it is false negative and if the actual class is negative and predicted class is also negative we get true negative section.

It's essential to comprehend the centrality of these measurements. Precision is a general proportion of right expectation, paying little mind to the class (positive or negative). The supplement of exactness is mistake rate or misclassification rate.

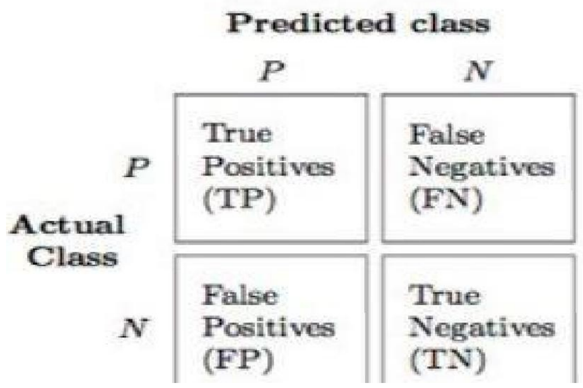


Fig. 6: Confusion Matrix

High review infers that not many positives are misclassified as negatives. High exactness suggests not many negatives are misclassified as positives. There's an exchange off here. In the event that model is fractional towards positives, we'll end up with high review however low exactness. It model favors negatives, we'll end up with low review and high exactness.

CHAPTER 3

PROPOSED WORK

To reach the proposed multi headed CNN based architecture, various experiment were done to get higher accuracy model which can predict the defect in the java based project. Defect is predicted within project means we are comparing two or more versions of same project on basis of defect. It can be understood by observing the result tables.

3.1 Problem Statement

The problem statement is " To develop a CNN based architecture improve the accuracy of defect finding by minimizing the original data loss during the process, this accuracy to be measured in term of F-measure, G-measure and MCC-measure for a set of JAVA based open source projects".

In the wake of parsing source code, a symbolic vector is required for each source record. Be that as it may, these symbolic vectors couldn't fill in as the immediate contribution for a CNN model, and along these lines initially it is expected to delineate from strings to whole numbers. At that point, a transformation that mapped each string token to a whole number extending from one to the absolute number of token sorts with the goal that each extraordinary string was spoken to by a special number constantly to be completed. What's more, the CNN model requires input vectors to have equivalent length. Be that as it may, the length of the information vectors fluctuated by the quantity of separated AST hubs for each source record after change occurs. To take care of the issue, a zero to the whole number vectors to make their lengths equivalent to the longest vector. The digit zero would not influence mapping spaces on the grounds that the mapping began from one.

After processing of data as shown in above paragraph the processed data is passed to the CNN architecture for training and testing purposes. Problem here is to develop a efficient CNN based model which can give desired results.

3.2 Proposed Solution

The Algorithm/ Working Flow of Classification Model for approach or implementation is explained below:

m	No. of samples
n	No. of features
1	Label for OK data
2	Label for Defective data
mT	No. of training samples
mt	No. of testing samples
$X_T^{mT \times n}$:	Training samples
$Y_T^{mT \times 1}$	Training labels
$X_t^{mt \times n}$:	Testing samples
$Y_t^{mt \times 1}$	Testing labels
$Y_{pred}^{mt \times 1}$	Predicted labels

```
1. Prepared dataset $X_{m \times n}$ and label $Y_{m \times 1}$ where $Y \in \{0,1\}$
2.Split data $X_{m \times n}$ and label $Y_{m \times 1}$ into 70% training and 30% testing sets
3. $(X_T^{mT \times n}, Y_T^{mT \times 1})$ is the training set and $(X_t^{mt \times n}, Y_t^{mt \times 1})$ is the testing set
4. Set $batch\ size = 64$ $and$ $epochs = 30$
5. for $epoch = 1,2,3, \ldots \ldots \ldots, epochs$
        5.1 Set $count = 0$
        5.2 Repeat until $count + batch\ size < mT$
                5.2.1 Train classifier on $X_T^{[illegible]}, Y_T^{[illegible]}$
                5.2.2 Calculate accuracy
                5.2.3 Set $count = batch\ size$
6. end for
7. Perform testing on $X_t^{mt \times n}$ and find $Y_{pred}^{mt \times 1}$
8. Compare $Y_{pred}^{mt \times 1}$ and $Y_t^{mt \times 1}$ and calculate
        1. F-Measure
        2. G-Measure
        3. MCC
```

A multi headed architecture is developed and output of this model given to the SVM classifier. SVM classifier is used instead of leaner regression classifier for classification of results as buggy or non-buggy. The outcomes were as a decimal number somewhere in the range of zero and one, in view of which we anticipated a source document as carriage or clean. On the off chance that the outcome was above 0.5, the forecast was viewed as cart; else it was viewed as perfect.
The Proposed multi headed CNN architecture developed for processing of PSC data set is as follows:

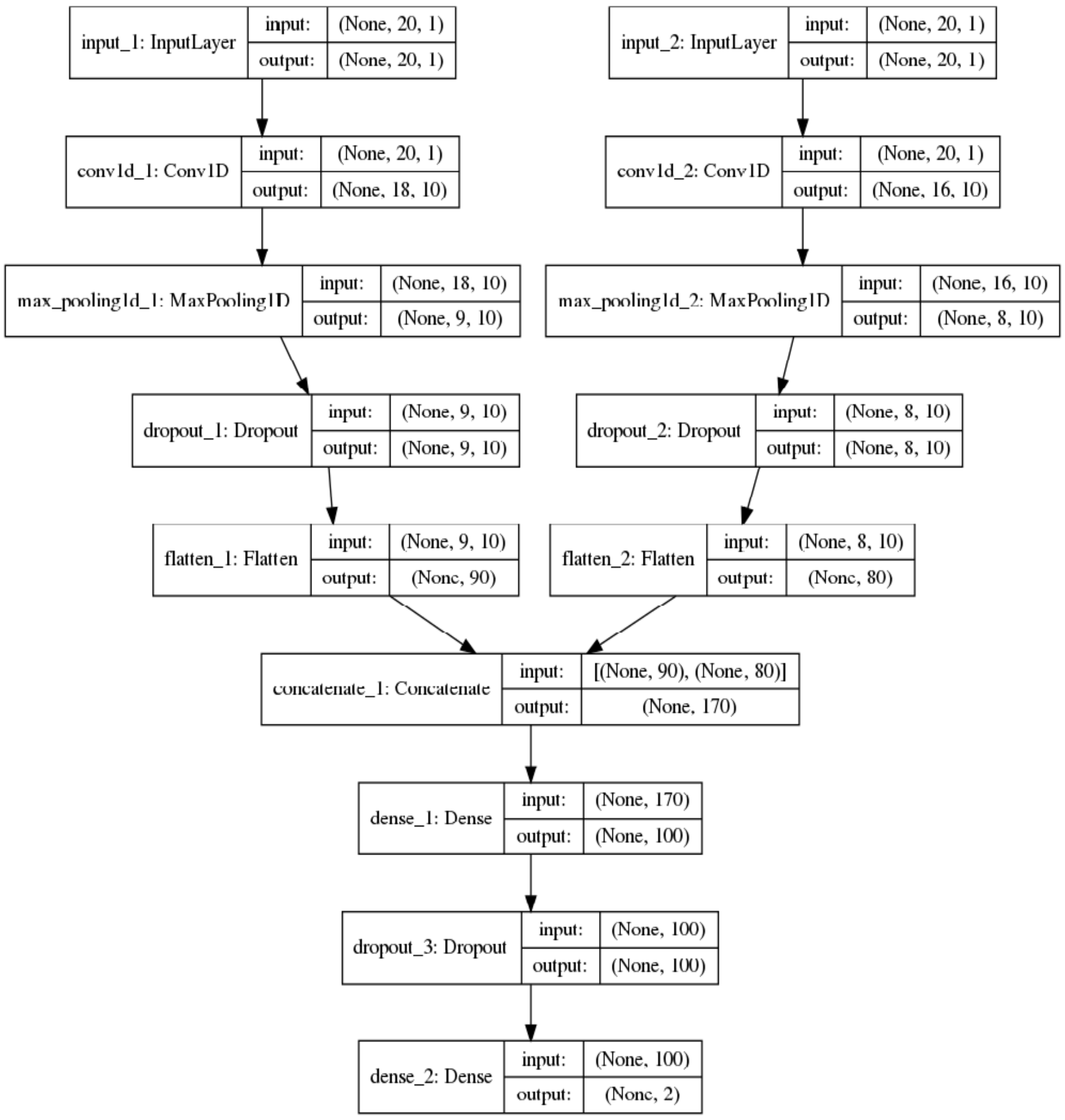


Fig. 7: Proposed Multi headed CNN based model

In starting we have the input layer, where the initial input of 20 parameters from the .csv file is given, we had the separate .csv files for all 14 projects mentioned in table1 and output input layer then passed to next layer i.e convolution layer. Max pooling layer is reducing the nodes to half the number as compared to input. The most widely recognized type of pooling is max pooling. Max pooling decreases the computational expense by diminishing the quantity of boundaries to learn and gives fundamental interpretation invariance to the interior portrayal. Flatten layer straighten the yield of the convolutional layers to make a solitary long element vector.

As we increases the running instances of model over every project the loss of model is reduced and the accuracy increases and it become flat after certain number of repetitive runs carried out, in our case we can say 30 runs. It can be understood by figure shown below:

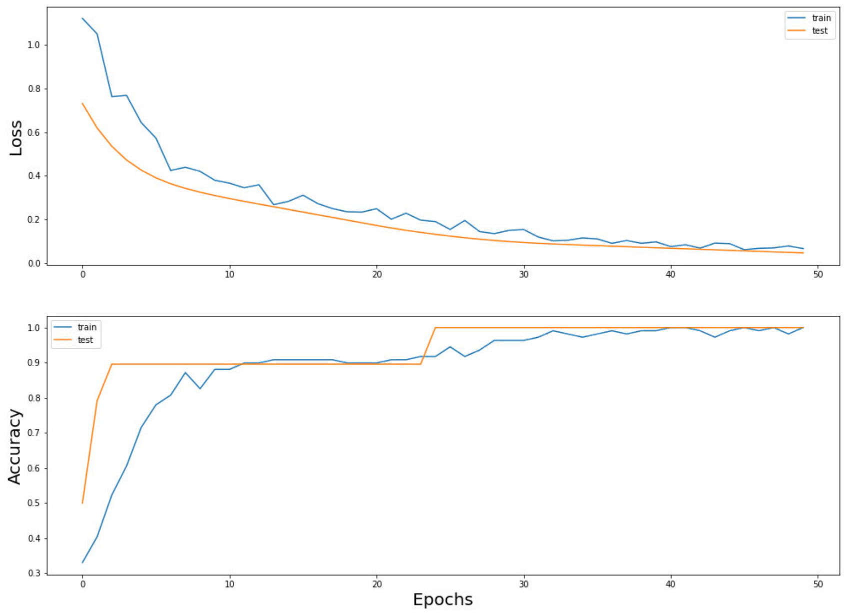
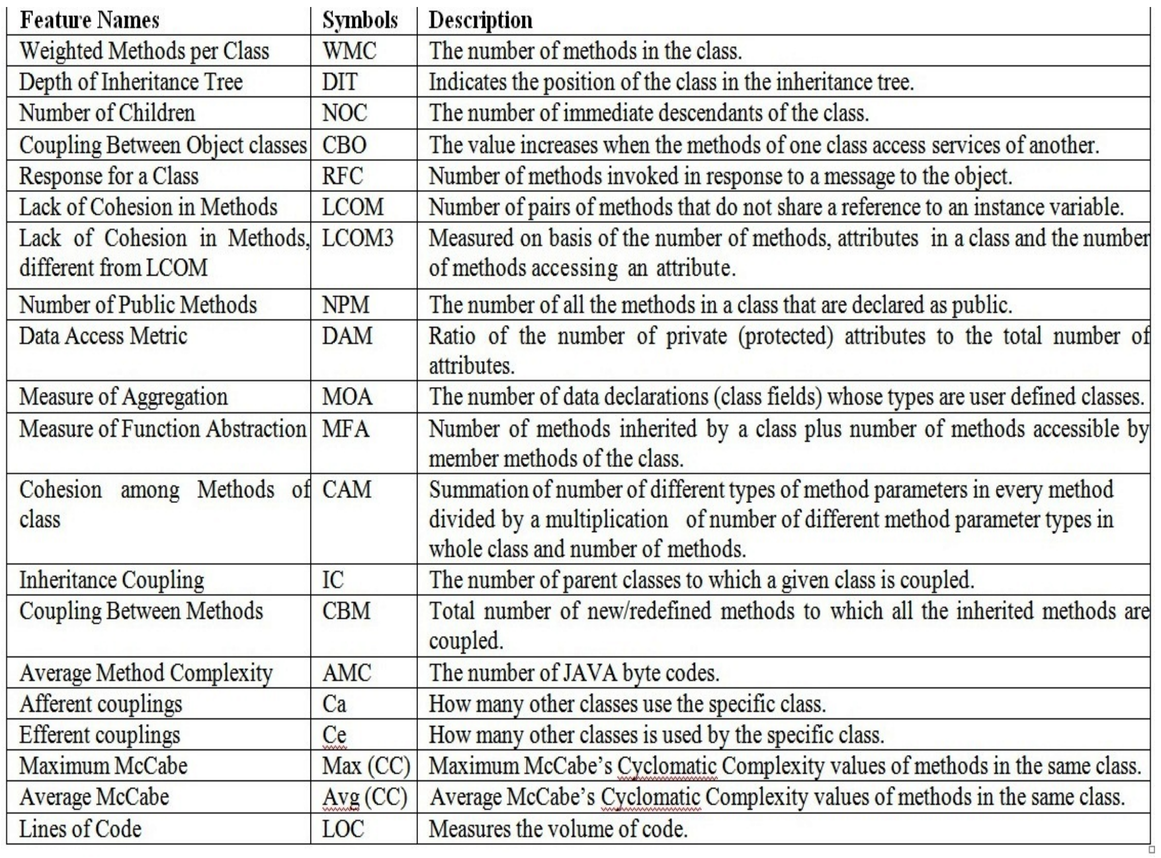


Fig. 8: Improved Accuracy and Decrease in loss with increase in no. of Epochs

The most relevant feature used in our source code are mentioned below in the tabular form:



Feature Names	Symbols	Description
Weighted Methods per Class	WMC	The number of methods in the class.
Depth of Inheritance Tree	DIT	Indicates the position of the class in the inheritance tree.
Number of Children	NOC	The number of immediate descendants of the class.
Coupling Between Object classes	CBO	The value increases when the methods of one class access services of another.
Response for a Class	RFC	Number of methods invoked in response to a message to the object.
Lack of Cohesion in Methods	LCOM	Number of pairs of methods that do not share a reference to an instance variable.
Lack of Cohesion in Methods, different from LCOM	LCOM3	Measured on basis of the number of methods, attributes in a class and the number of methods accessing an attribute.
Number of Public Methods	NPM	The number of all the methods in a class that are declared as public.
Data Access Metric	DAM	Ratio of the number of private (protected) attributes to the total number of attributes.
Measure of Aggregation	MOA	The number of data declarations (class fields) whose types are user defined classes.
Measure of Function Abstraction	MFA	Number of methods inherited by a class plus number of methods accessible by member methods of the class.
Cohesion among Methods of class	CAM	Summation of number of different types of method parameters in every method divided by a multiplication of number of different method parameter types in whole class and number of methods.
Inheritance Coupling	IC	The number of parent classes to which a given class is coupled.
Coupling Between Methods	CBM	Total number of new/redefined methods to which all the inherited methods are coupled.
Average Method Complexity	AMC	The number of JAVA byte codes.
Afferent couplings	Ca	How many other classes use the specific class.
Efferent couplings	Ce	How many other classes is used by the specific class.
Maximum McCabe	Max (CC)	Maximum McCabe's Cyclomatic Complexity values of methods in the same class.
Average McCabe	Avg (CC)	Average McCabe's Cyclomatic Complexity values of methods in the same class.
Lines of Code	LOC	Measures the volume of code.

Table 2: 20 Major Features used in proposed Model [13]

-

CHAPTER 4

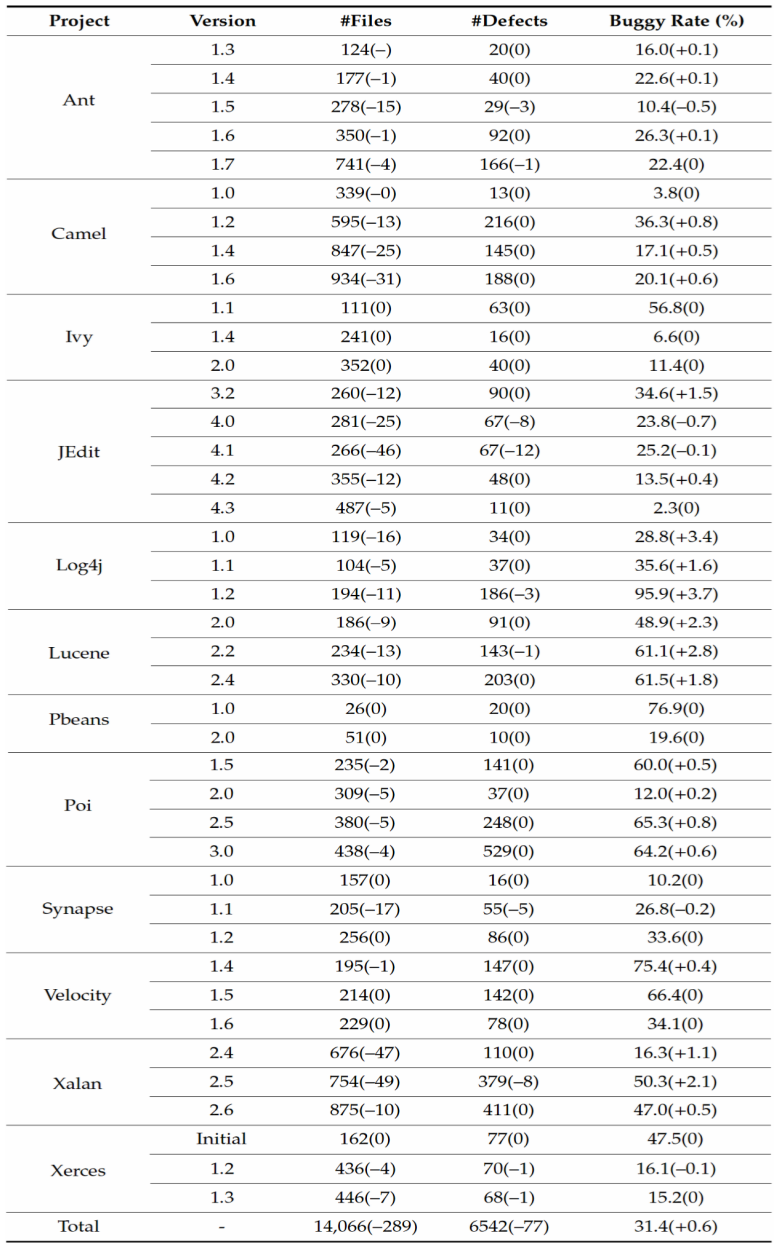
IMPLEMENTATION AND RESULTS

Some parameters used in the coding are mentioned in the table below:

Parameter	**Value**
Number of Filters	10
Kernal Sizes	3 to 5
Drop out	0.5
Batch Size	256
Non-linearity function	ReLU
Pool Size	2

Table 3.: Parameters used in model

The dataset used is PROMISE Source code (PSC) dataset [2]. Negative number in brackets in *#files* column represents the number of files which are removed due to having buggy rate > 95% or data in file not relevant i.e not related to source code. from the data in PROMISE repository. Dataset comprises of nearly 14,006 files from 12 open source Java based projects having 41 versions. PSC data set is available at URL https://github.com/penguincwarrior/CNN-WPDPAPPSCI2019. We had used the processed data set [2] in our model as input. Details of data set given in tabular form given below:



Project	Version	#Files	#Defects	Buggy Rate (%)
Ant	1.3	124(–)	20(0)	16.0(+0.1)
	1.4	177(–1)	40(0)	22.6(+0.1)
	1.5	278(–15)	29(–3)	10.4(–0.5)
	1.6	350(–1)	92(0)	26.3(+0.1)
	1.7	741(–4)	166(–1)	22.4(0)
Camel	1.0	339(–0)	13(0)	3.8(0)
	1.2	595(–13)	216(0)	36.3(+0.8)
	1.4	847(–25)	145(0)	17.1(+0.5)
	1.6	934(–31)	188(0)	20.1(+0.6)
Ivy	1.1	111(0)	63(0)	56.8(0)
	1.4	241(0)	16(0)	6.6(0)
	2.0	352(0)	40(0)	11.4(0)
JEdit	3.2	260(–12)	90(0)	34.6(+1.5)
	4.0	281(–25)	67(–8)	23.8(–0.7)
	4.1	266(–46)	67(–12)	25.2(–0.1)
	4.2	355(–12)	48(0)	13.5(+0.4)
	4.3	487(–5)	11(0)	2.3(0)
Log4j	1.0	119(–16)	34(0)	28.8(+3.4)
	1.1	104(–5)	37(0)	35.6(+1.6)
	1.2	194(–11)	186(–3)	95.9(+3.7)
Lucene	2.0	186(–9)	91(0)	48.9(+2.3)
	2.2	234(–13)	143(–1)	61.1(+2.8)
	2.4	330(–10)	203(0)	61.5(+1.8)
Pbeans	1.0	26(0)	20(0)	76.9(0)
	2.0	51(0)	10(0)	19.6(0)
Poi	1.5	235(–2)	141(0)	60.0(+0.5)
	2.0	309(–5)	37(0)	12.0(+0.2)
	2.5	380(–5)	248(0)	65.3(+0.8)
	3.0	438(–4)	529(0)	64.2(+0.6)
Synapse	1.0	157(0)	16(0)	10.2(0)
	1.1	205(–17)	55(–5)	26.8(–0.2)
	1.2	256(0)	86(0)	33.6(0)
Velocity	1.4	195(–1)	147(0)	75.4(+0.4)
	1.5	214(0)	142(0)	66.4(0)
	1.6	229(0)	78(0)	34.1(0)
Xalan	2.4	676(–47)	110(0)	16.3(+1.1)
	2.5	754(–49)	379(–8)	50.3(+2.1)
	2.6	875(–10)	411(0)	47.0(+0.5)
Xerces	Initial	162(0)	77(0)	47.5(0)
	1.2	436(–4)	70(–1)	16.1(–0.1)
	1.3	446(–7)	68(–1)	15.2(0)
Total	-	14,066(–289)	6542(–77)	31.4(+0.6)

Table 4: Dataset PSC [2]

Details of implementation is given below:

- Implementation framework used: Python 3.1 setup with Keras 2.2.4 and tenserflow 1.14 for bulding model is used.
- All of our experiments were run on windows 8.1 pro with intel(R) core(TM) i5-7200u cpu @ 2.30ghz and having RAM of GB.
- Multi headed CNN based model is used for processing the data set to get the desire results. Model fuctioning is explained in fig. 3.

- ReLU is used as the activation fuction except the last layer which uses SVM for classification.
- The model epoch value is kept at 30 rounds and resut is average of output reacvied by every itration. Number of repetition rounds kept limited due to resorce limitaions.
- Dropouts: Dropout is a system model planning to manage model overfitting issues. Its key thought is to arbitrarily drop neural units just as their associations during preparing, which would forestall complex co-adaptions of units and lower model speculation mistake (contrasts between model execution on preparing set and test set). In the regressive spread procedure of a neural system, the loads of a unit are refreshed given what different units are doing, so the weight may be refreshed to make up for the mix-up of different units, which is called co-adjustment. While including dropout layers, a unit is problematic in light of the fact that it might be arbitrarily dropped. Thusly, every unit would learn better highlights as opposed to fixing the errors of different units. Weight rescaling is performed on test sets to make up for the dropped units. Fig. 6 can be understood by a guiding example to understand the mechanics of dropouts. At the point when we include a dropout layer for the concealed layer and set the dropout likelihood to 0.5, shrouded units are haphazardly picked to be dropped at a likelihood of 0.5, as H2 and H4, for instance. The associations between the two hubs and the information/yield layers are likewise dropped. For this situation, the weight update of H1 and H3 would be autonomous of H2 and H4, which would forestall co-adjustment of units and lower model speculation mistake.

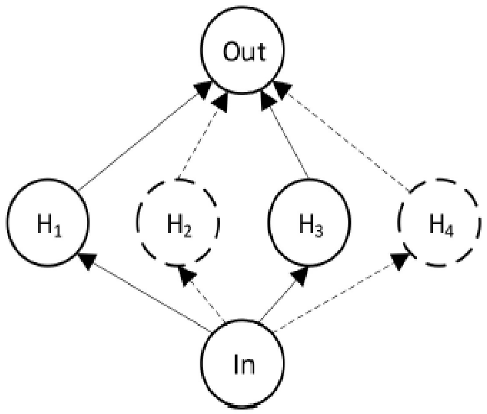


Fig. 9: Drop out example

We had kept the drpoout probability to 0.50.

- We had used the 70 percentage of data for training purpose and 30 percent data for testing of the model and this is done separately for very project independently.
- SVM classifier is used instead of Liner Regression for classification into buggy or non-buggy.
- F-measure, G-measure and MCC-measure values are calculated for compression with existing results.

4.1 Evaluation Matrices

Now, after implementation, the evaluation parameters are calculated. In this section, the evaluation parameters taken into consideration are defined.
The five parameters taken into consideration are Precision , Recall, F Measure, MCC Measure and G Measure.

Precision: It is the ratio of data elements that are correctly classified (for both the minority and majority class) to total number of classified instances.

$$P = TP / (TP + FP)$$

Recall: The ratio of the minority class instances that are correctly classified to the total number of actual minority class instances.

$$R = TP / (TP + FN)$$

F Measure: this parameter depends on precision parameter and recall parameter, which are measured by true positive, true negative, false positive and false negative numbers.
F measure is defined as:

$$\text{F-measure} = \frac{2 * precision * recall}{precision + recall}$$

Thus to calculate the F-measure we drived the values of P(precision) and R(recall) given in table 2 in next page. The values in table represents the average values after runing the proposed model 30 times.

Project Name	Version	Avg. Precision	Avg. Recall
lucene	2	1	1
lucene	2.2	0.9986666667	1
lucene	2.4	1	1
synapse-1.0.csv	1	1	1
synapse-1.1.csv	1.1	1	1
synapse-1.2.csv	1.2	0.9986666667	1
poi-1.5.csv	1.5	1	1
poi-2.0.csv	2	0.9973333333	1
poi-2.5.csv	2.5	1	1
poi-3.0.csv	3	0.9986666667	1
jedit-3.2.csv	3.2	0.9976666667	1
jedit-4.0.csv	4	0.9986666667	1
jedit-4.1.csv	4.1	0.998	1
jedit-4.2.csv	4.2	0.993	1
jedit-4.3.csv	4.3	0.4333333333	0.4333333333
camel-1.0.csv	1	0.3666666667	0.3666666667
camel-1.2.csv	1.2	0.9993333333	1
camel-1.4.csv	1.4	0.9986666667	1
camel-1.6.csv	1.6	0.9993333333	1
xerces-Initial.csv	Initial	1	1
xerces-1.2.csv	1.2	1	1
xerces-1.3.csv	1.3	1	1
log4j-1.0.csv	1	1	1
log4j-1.1.csv	1.1	1	1
log4j-1.2.csv	1.2	0.9993333333	1
ivy-1.1.csv	1.1	0.9966666667	1
ivy-1.4.csv	1.4	0.9333333333	0.9333333333
ivy-2.0.csv	2	0.995	1
xalan-2.4.csv	2.4	0.9946666667	1
xalan-2.5.csv	2.5	1	1
xalan-2.6.csv	2.6	0.9993333333	1
ant-1.3.csv	1.3	1	1
ant-1.4.csv	1.4	0.9953333333	1
ant-1.5.csv	1.5	1	1
ant-1.6.csv	1.6	0.995	1
ant-1.7.csv	1.7	0.9903333333	1
pbeans-1.0.csv	1	0.989	1
pbeans-2.0.csv	2	1	1
velocity-1.4.csv	1.4	1	1
velocity-1.5.csv	1.5	1	1
velocity-1.6.csv	1.6	1	1

Table 5 : Values of Precision and Recall, which are avg. values after 30 runs

In factual investigation of twofold order, the F1 score (likewise F-score or F-measure) is a proportion of a test's precision. It considers both the exactness p and the review r of the test to figure the score: p is the quantity of right positive outcomes partitioned by the quantity of every positive outcome returned by the classifier, and r is the quantity of right positive outcomes separated by the quantity of every single applicable example (all examples that ought to have been recognized as positive). The F1 score is the consonant mean of the exactness and review, where a F1 score arrives at its best an incentive at 1 (immaculate accuracy and review).

G measure: it is defined as the harmonic mean made by true positive rate. It is given as:

$$\text{G-measure} = \frac{2 * TPR * TNR}{TPR + TNR}.$$

Where TPR and TNR are true positive and true negative rate respectively.

MCC measure: the correlation between predicted and true values is MCC. It is defined as:

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}.$$

4.2 Experimental Results

In this section results are shown in tabular and graphical format. The result justifies significant improvement in the proposed method for software defect prediction using modified CNN with combined SVM classifier. In the result table the values of F-measure, G-measure and MCC-measure is average of values obtained after running the proposed multi headed CNN based model 30 times.

In table 6, F Measure results for different projects are shown under different version. It is seen that Camel and Lucene has found the highest F Measure.

The decimal values states F-measure and best value is indicated in bold.

Project Name	**Version**	**Three Layer CNN (Cong Pan)**	**Proposed CNN with SVM**
Lucene	2	0.74	**1**
	2.2	0.63	**0.999333333**
	2.4	0.77	**1**
Synapse	1	0.29	**1**
	1.1	0.67	**1**
	1.2	0.69	**0.989333333**
Poi	1.5	0.61	**1**
	2	0.13	**0.998666667**
	2.5	**0.9**	0.98
	3	0.76	**0.998666667**
JEdit	3.2	0.69	**0.998666667**
	4	0.48	**0.959333333**
	4.1	0.41	**0.999**
	4.2	0.58	**0.996666667**
	4.3	0	**0.433333333**
Camel	1	**0.4**	0.366666667
	1.2	0.69	**0.979666667**
	1.4	0.46	**0.999333333**
	1.6	0.52	**0.999666667**
Xerces	Initial	0.65	**1**
	1.2	0.41	**1**
	1.3	0.56	**1**
Log4j	1	0.77	**1**
	1.1	0.4	**1**
	1.2	**0.97**	0.959666667
Ivy	1.1	0.8	**0.998333333**
	1.4	0.22	**0.933333333**
	2	0.31	**0.997333333**
Xalan	2.4	0.25	**0.997**
	2.5	0.7	**1**
	2.6	0.76	**1**
Ant	1.3	0.67	**1**
	1.4	0.38	**0.897666667**
	1.5	0.25	**1**
	1.6	0.41	**0.997333333**
	1.7	0.39	**0.995**
Pbeans	1	0.89	**0.963333333**
	2	0.67	**1**
Velocity	1.4	**0.9**	0.96666
	1.5	0.78	**1**
	1.6	0.83	**1**
Average		0.570487805	0.961073008

Table 6: Comparison of F-measure of three Layer CNN (Cong Pan) and proposed multi headed CNN with SVM model on PSC data.

In table 7, performance for G measure is shown. Shows the best improvement in proposed model. The decimal values states G-measure and best value is indicated in bold.

Project Name	Version	Three Layer CNN (Cong Pan)	Proposed CNN with SVM
Lucene	2	0.75	**1**
	2.2	0.62	**0.999**
	2.4	0	**1**
Synapse	1	0.49	**1**
	1.1	0.7	**1**
	1.2	0.72	**0.999666667**
Poi	1.5	0.61	**1**
	2	0.25	**0.969666667**
	2.5	**0.82**	0.79999
	3	0.72	**0.998666667**
JEdit	3.2	0.73	**0.999333333**
	4	0.65	**0.999666667**
	4.1	0	**0.999666667**
	4.2	0.78	**0.999**
	4.3	0	**0.433333333**
Camel	1	**0.5**	0.366666667
	1.2	0.76	**1**
	1.4	0.54	**1**
	1.6	0.6	**0.9899**
Xerces	Initial	0.58	**1**
	1.2	0.66	**1**
	1.3	0.65	**1**
Log4j	1	**0.81**	0.79999
	1.1	0.44	**1**
	1.2	0	**0.905333333**
Ivy	1.1	0.65	**0.997666667**
	1.4	0.49	**0.933333333**
	2	0.4	**0.999333333**
Xalan	2.4	0.31	**1**
	2.5	0.69	**1**
	2.6	0.78	**1**
Ant	1.3	0.67	**1**
	1.4	0.51	**0.989333333**
	1.5	0.28	**1**
	1.6	0.53	**0.999**
	1.7	0.46	**0.999333333**
Pbeans	1	**0.89**	0.855666667
	2	0.67	**1**
Velocity	1.4	0.66	**1**
	1.5	0.72	**1**
	1.6	0.84	**0.99**
Average		0.559268293	0.951793821

Table 7: Comparison of G-measure of three Layer CNN (Cong Pan) and proposed multi headed CNN with SVM model on PSC data.

In table 8, MCC comparison is shown which is best for proposed work. The decimal values states MCC-measure and best value is indicated in bold.

Project Name	Version	Three Layer CNN (Cong Pan)	Proposed CNN with SVM
Lucene	2	0.53	**1**
	2.2	0.29	**0.998**
	2.4	0	**1**
Synapse	1	0.2	**1**
	1.1	0.6	**1**
	1.2	0.59	**0.999**
Poi	1.5	0.29	**1**
	2	0.01	**0.998333333**
	2.5	**0.7**	0.6778
	3	0.42	**0.997333333**
JEdit	3.2	0.56	**0.998**
	4	0.29	**0.999**
	4.1	0	**0.998666667**
	4.2	0.51	**0.996333333**
	4.3	0	**0.433333333**
Camel	1	**0.39**	0.366666667
	1.2	0.5	**0.999666667**
	1.4	0.39	**0.999333333**
	1.6	0.42	**0.989666667**
Xerces	Initial	0.23	**1**
	1.2	0.28	**1**
	1.3	0.5	**1**
Log4j	1	0.69	**1**
	1.1	0.29	**1**
	1.2	0	**0.995333333**
Ivy	1.1	0.48	**0.996**
	1.4	0.16	**0.933333333**
	2	0.25	**0.997333333**
Xalan	2.4	0.18	**0.996666667**
	2.5	0.38	**1**
	2.6	0.58	**0.999333333**
Ant	1.3	0.68	**1**
	1.4	0.2	**0.997**
	1.5	0.24	**1**
	1.6	0.22	**0.996666667**
	1.7	0.3	**0.994666667**
Pbeans	1	0.63	**0.952666667**
	2	0.67	**1**
Velocity	1.4	0.56	**1**
	1.5	0.42	**1**
	1.6	**0.76**	0.72229
Average		0.375365854	0.952010325

Table 8: Comparison of MCC-measure of three Layer CNN (Cong Pan) and proposed multi headed CNN with SVM model on PSC data.

Now, all the average values are taken and plot in fig. 10, fig. 11 and fig. 12.

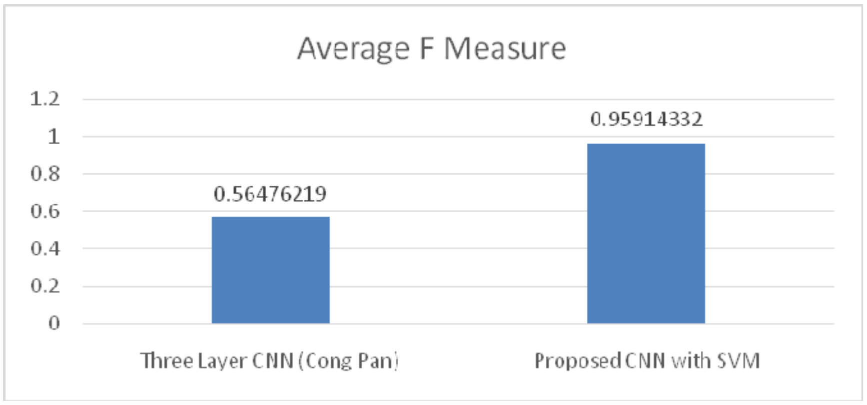


Fig. 10: Average F Measure Comparison Result

In fig. 10, clearly Average F measure is higher in the proposed CNN with SVM new model. Similarly, for G-Measure in Fig 11. below.

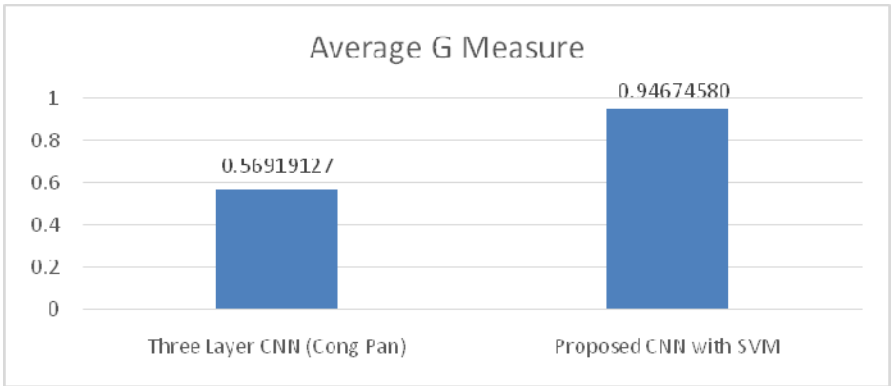


Fig. 11: Average G Measure Comparison Result

In fig. 12, final parameter MCC is also improved as per the shown average values.

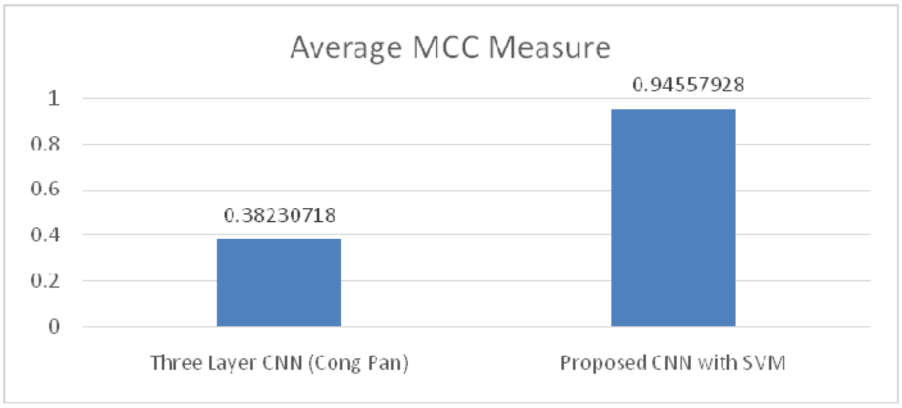


Fig. 12: Average MCC Measure Comparison Result

CHAPTER 5

CONCLUSION AND FUTURE WORK

Conclusion:

Software defect prediction is a procedure of foreseeing code territories that conceivably contain defects, which can assist designers with apportioning their testing endeavors by first checking possibly buggy code. The proposed work offers high reliability in terms of software defect prediction and the parameter analyses is very much improved. After carrying out various experiments with various combination of input layers , convolutional layers, pooling layers, dropouts, dense layers and classifier combination we reached to your proposed multi headed CNN based architecture, which gives much better performance in terms of evolution matrices. result we got are very encouraging and in most of the cases your proposed model outperformed the existing mode. In the process we also restricted the number of layer in order to cater the hardware limitation and running times of the model.

Future Scope:

As it can be seen that model only for the java based projects, hence there is scope that model can be extended for the other projects which are based on python/c/c++ and other programming languages. Also 20 features are used in implementation work and there is scope of increasing features extraction and so the increase in relevant feature, so that efficiency can be further optimized.

REFERENCES

[1] J. Li, P. He, J. Zhu and M. R. Lyu, "Software Defect Prediction via Convolutional Neural Network," *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, Prague, 2017, pp. 318-328, doi: 10.1109/QRS.2017.42.

[2] Pan, C.; Lu, M.; Xu, B.; Gao, H. An Improved CNN Model for Within-Project Software Defect Prediction. *Appl. Sci.***2019**, *9*, 2138

[3] M. Samir, M. El-Ramly and A. Kamel, "Investigating the Use of Deep Neural Networks for Software Defect Prediction," *2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA)*, Abu Dhabi, United Arab Emirates, 2019, pp. 1-6, doi: 10.1109/AICCSA47632.2019.9035240.

[4] E. A. Felix and S. P. Lee, "Integrated Approach to Software Defect Prediction," in *IEEE Access*, vol. 5, pp. 21524-21547, 2017, doi: 10.1109/ACCESS.2017.2759180

[5] K. Yang, H. Yu, G. Fan, X. Yang, S. Zheng and C. Leng, "Software Defect Prediction Based on Fourier Learning," *2018 IEEE International Conference on Progress in Informatics and Computing (PIC)*, Suzhou, China, 2018, pp. 388-392, doi: 10.1109/PIC.2018.8706304.

[6] S. Huda *et al.*, "A Framework for Software Defect Prediction and Metric Selection," in *IEEE Access*, vol. 6, pp. 2844-2858, 2018, doi: 10.1109/ACCESS.2017.2785445.

[7] P. Deep Singh and A. Chug, "Software defect prediction analysis using machine learning algorithms," *2017 7th International Conference on Cloud Computing, Data Science & Engineering - Confluence*, Noida, 2017, pp. 775-781, doi: 10.1109/CONFLUENCE.2017.7943255.

[8] Z. Li, X. Jing and X. Zhu, "Progress on approaches to software defect prediction," in IET Software, vol. 12, no. 3, pp. 161-175, 6 2018, doi: 10.1049/iet-sen.2017.0148.

[9] Z. Tian, J. Xiang, S. Zhenxiao, Z. Yi and Y. Yunqiang, "Software Defect Prediction based on Machine Learning Algorithms," 2019 IEEE 5th International

Conference on Computer and Communications (ICCC), Chengdu, China, 2019, pp. 520-525, doi: 10.1109/ICCC47050.2019.9064412.

[10] S. Sutar, R. Kumar, S. Pai and B. R. Shwetha, "Defect Prediction based on Machine Learning using System Test Parameters," *2019 Amity International Conference on Artificial Intelligence (AICAI)*, Dubai, United Arab Emirates, 2019, pp. 134-139, doi: 10.1109/AICAI.2019.8701345.

[11] P. K. Singh, D. Agarwal and A. Gupta, "A systematic review on software defect prediction," *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, New Delhi, 2015, pp. 1793-1797.

[12] Fan, Guisheng & Diao, Xuyang & Yu, Huiqun & Yang, Kang & Chen, Liqiong. (2019). Software Defect Prediction via Attention-Based Recurrent Neural Network. Scientific Programming. 2019. 1-14. 10.1155/2019/6230953..

[13] Z. He, F. Peters, T. Menzies, and Y. Yang, "Learning from open-source projects: An empirical study on defect prediction," in ESEM'13: Proc. of the International Symposium on Empirical Software Engineering and Measurement, 2013

[14] X.-Y. Jing, S. Ying, Z.-W. Zhang, S.-S. Wu, and J. Liu, "Dictionarylearning based software defect prediction," in ICSE'14: Proc. of theInternational Conference on Software Engineering, 2014.

[15] K. O. Elish and M. O. Elish, "Predicting defect-prone software modulesusing support vector machines," Journal of Systems and Software,vol. 81, no. 5, pp. 649–660, 2008.