AI APPROACH FOR RELIABILITY ESTIMATION OF SAFETY-CRITICAL SYSTEMS

Project Submitted to the
SRM University AP, Andhra Pradesh
for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology

in

Computer Science & Engineering

School of Engineering & Sciences

submitted by

Aman Abdul Rasheed(AP20110010230)

Romith Bondada(AP20110010248)

Sai Sushrooth Chand(AP20110010263)

Sukanya Karasala(AP20110010264)

Under the Guidance of
Prof. Raj Sekhar Krovi

Department of Computer Science & Engineering
SRM University-AP
Neerukonda, Mangalgiri, Guntur
Andhra Pradesh - 522 240
May 2024

DECLARATION

I undersigned hereby declare that the project report **AI APPROACH FOR RELIABILITY ESTIMATION OF SAFETY-CRITICAL SYSTEMS** submitted for partial fulfillment of the requirements for the award of degree of Bachelor of Technology in the Computer Science & Engineering, SRM University-AP, is a bonafide work done by me under supervision of Prof. Raj Sekhar Krovi. This submission represents my ideas in my own words and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources. I also declare that I have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree of any other University.

Place	:	Date	: May 11, 2024
Name of student	: Aman Abdul Rasheed	Signature	: ....
Name of student	: Romith Bondada	Signature	: ....
Name of student	: Sai Sushrooth Chand	Signature	: ....
Name of student	: Sukanya Karasala	Signature	: ....

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SRM University-AP
Neerukonda, Mangalgiri, Guntur
Andhra Pradesh - 522 240

CERTIFICATE

This is to certify that the report entitled **AI APPROACH FOR RELIABILITY ESTIMATION OF SAFETY-CRITICAL SYSTEMS** submitted by **Aman Abdul Rasheed, Romith Bondada, Sai Sushrooth Chand, Sukanya Karasala** to the SRM University-AP in partial fulfillment of the requirements for the award of the Degree of Master of Technology in in is a bonafide record of the project work carried out under my/our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

Project Guide	Head of Department
Name : Prof. Raj Sekhar Krovi	Name : Prof. Niraj Upadhyaya
Signature:	Signature: ....

ACKNOWLEDGMENT

I wish to record my indebtedness and thankfulness to my group members who helped me prepare this Project Report titled **AI APPROACH FOR RELIABILITY ESTIMATION OF SAFETY-CRITICAL SYSTEMS** and present it satisfactorily.

I am especially thankful for my guide and supervisor Prof. Raj Sekhar Krovi in the Department of Computer Science & Engineering for giving me valuable suggestions and critical inputs in the preparation of this report. I am also thankful to Prof. Niraj Upadhyaya, Head of Department of Computer Science & Engineering for encouragement.

My friends in my class have always been helpful and I am grateful to them for patiently listening to my presentations on my work related to the Project.

Aman Abdul Rasheed, Romith Bondada, Sai Sushrooth Chand, Sukanya Karasala
(Reg. No. AP20110010230, AP20110010248, AP20110010263, AP20110010264)
B. Tech.
Department of Computer Science & Engineering
SRM University-AP

ABSTRACT

The reliability of a software module has always been a conundrum for software developers. Software Defect Prediction is an early step in achieving reliability as it helps reduce time and workload. Detecting software faults beforehand is crucial for enhancing software quality and reducing costs, time, and effort in the development process. Machine Learning (ML) algorithms have worked across various scientific domains, including the prediction of software faults, and has played a vital role in Software Fault Prediction (SFP). Deep Learning Models have proven exceptional performance in software reliability apart from other key domain areas of computer vision, natural language processing and others. Machine Learning algorithms like Decision Trees, Logistic Regression, etc, and deep learning models like Multi-layer Perceptron (MLP) and Convolution Neural Network (CNN) have been used extensively in software reliability. The novel application of a new Deep Learning Model and Recurrent Neural Network (RNN) is implemented in this study and the experimental modification of previously mentioned deep learning models resulted in an efficient performance. The experimental study was carried on four datasets taken from NASA open data portal where tens of thousands of datasets are available to the public. The results show an overall improvement rate in performance of 59% for PC1 and 143% for CM1 using MLP. In the case of CNN, the improvement rate is 9% and 2% for PC1 and CM1.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1
INTRODUCTION TO THE PROJECT

Until the 1940s, the term "reliability" was mostly regarded as "repeatability". One can say a system is reliable if it can achieve the same test results repeatedly. Thus far, this definition has been improved and rephrased as follows: Reliability is the "probability of a system or component, performing its intended functions under specified operating conditions for a specified period"[5]. Developing reliable software is one of the challenging parts of software development. One of the crucial side to attaining high-quality and dependable software is the occurrence of errors. These errors deprecate quality of software followed by undependable outcome that fail to meet customer satisfaction. The occurrence of these faults is unavoidable and turn up in various phases of software development. Certain measures like suitable planning, and control of software development cycle are to be followed to achieve high-quality software. One of the quality models that helps to reduce software failure is Software fault prediction which also helps to avoid learning and may provide valuable improvement in software fault prediction[6]. Predicting faults allows the development team to conduct multiple rounds of testing on modules or files with a high probability of faults. This increased focus on problematic modules enhances the likelihood of resolving remaining faults, thereby improving the overall quality of software products released to end-users. This approach also reduces the maintenance and support efforts required for the project. Software faults are a significant contributor to low software quality, often demanding extensive efforts to rectify errors. The use of SFP aims to mitigate the impact of these faults. The SFP also reduces the costs, time, and effort that should be spent on software products[7].
[8] reported that the most expensive software development activities are the cost of finding and correcting faults. A large number of researches have been conducted using machine learning techniques for software fault prediction such as support vector machine, genetic algorithm [9] Artificial Neural Network (ANN) [10], decision tree [11], etc. [12] addressed different deep learning techniques and answer the following questions, what are the common models of deep learning and their optimization methods, are they commonly using open source frameworks, what are the major existing problems and what are the future suggested solutions. Their coherent and efficient evaluation was fuel-saving and time-effective to obtain the necessary resources and findings appropriate for the research. Following this research utilised deep learning algorithms, by putting mind of the previous approaches. Deep Learning is a set of techniques used for learning from

many layers in Neural Networks; it is a sub field of machine learning that uses supervised and/or unsupervised strategies. This has had magnificent success in various domains [13]. Deep learning lets computational models that are made from multiple layers learn representations of data with multiple levels of abstraction [14]. It automatically extracts essential features from raw data and makes it robust, concerning variations in input [15]. Moreover, Deep learning has the capability to handle large amounts of data, provides a lot of models that permit exploiting unlabeled data to learn useful patterns, and representations learned by deep neural networks can be shared across different tasks [16].
Recurrent Neural Networks (RNNs) have emerged as powerful tools in various domains, including natural language processing, time series analysis, and sequential data modeling. Unlike traditional feed-forward neural networks, RNNs possess a unique ability to capture temporal dependencies within sequential data by maintaining internal memory. This inherent capability makes RNNs particularly well-suited for tasks involving sequential data, such as software defect prediction. By leveraging the sequential nature of software development processes and historical defect data, RNNs hold the potential to uncover complex patterns and correlations that may be indicative of future software defects. In this report, we explore the implementation of RNNs in the context of software defect prediction, aiming to enhance our understanding of how these advanced neural network architectures can contribute to the improvement of software quality and reliability.

1.1 MOTIVATION

Several researches which were done on the software fault prediction domain have provided a curious thought of implementing a new deep learning model. The previous performance of the models can be improved with some experimental analysis such as fine-tuning or developing a new model. The earlier phase of the research was at a halt due to the poor performance of the models which needed more focus on the data. After careful observations, there was a huge class imbalance in all four datasets. This obstacle has been overcome by using the data augmentation method **"SMOTE"** which balances the dataset with an equal number of both faulty and non-faulty data samples.

1.2 SCOPE AND LIMITATIONS

This research focuses on investigating and developing a model for predicting faulty files, encompassing the following components:

- A pre-processing method for organizing the metrics into a desired format.

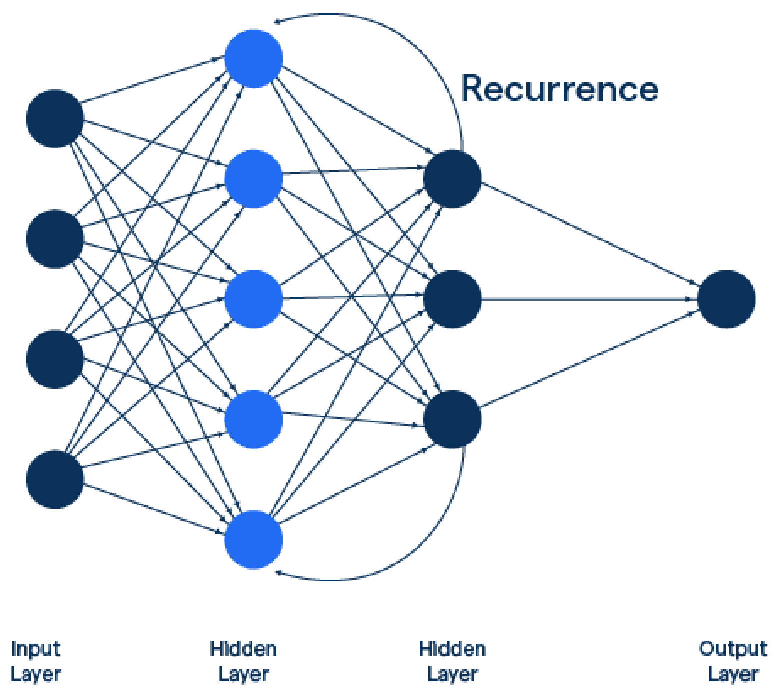

Figure 1.1: Recurrent Neural Network.

- An analysis accompanied by explanations of the generated results.
- Conclusions regarding the reasons behind the outcomes.

However, the research does not cover the following aspects:

- Developing a method for data acquiring as the data is taken from open source datasets.
- This data does not consider the most crucial part of the SFP which is the semantic understanding of code.

The biggest limitation of this research is the viability of the data as the improvements in large language models which extracts the semantic information of the code itself where this static code metrics fail to provide.

1.3 RESEARCH QUESTIONS

Following is the list of research questions this master thesis attempts to answer.

RQ1 - What Deep Learning models have been used for SFP? Studies were done on deep learning models such MLP and CNN . [The purpose of this research project is to fine tune the previously used model so that the prediction performance of metric sets are increased and study a novel model's performance]. **RQ2 - What types of metrics have been used in prior SFP studies?** The four NASA datasets (CM1, MW1, PC1, and MC1) consist of static code metrics such as LOC, Halfstead, and McCabe's. **RQ3 - Which model(s) provide the best SFP performance?** As this research project investigates a novel model combined with enhanced previous models, the answer to this question will be answered in chapter 8.

Chapter 2
BACKGROUND AND RELATED WORK

In this section, we have concentrated on Software Fault Prediction(SFP) by looking at its significant relevant studies as well as state-of-the-art techniques by previous findings. Numerous studies have contributed to enhancing software quality, optimizing resource utilization, and minimizing or eradicating faults. Hence, it is essential to delve into these research works to gain a comprehensive understanding of various aspects of SFP.

2.1 NEURAL NETWORKS

ML was successfully applied in SFP. yet, there is a lot of space to improve the accuracy of prediction for that [1] has done a study on how manipulation of hyper parameters can affect the performance of a model. Some results show a significant improvement in the detection rate. Four datasets were taken and were trained by changing the hyper-parameters such as epochs, activation function, dropout rate, and optimizer studying the effect and determining which model among MLP and CNN has a better performance. Due to the complex network obviously, CNN had an advantage.

2.2 SOFTWARE METRICS

Software metrics can be described as a quantitative measurement that assigns numbers or symbols to attributes of the measured entity [17]. An entity could represent the source code of an application or an activity within the software development process. Various families of software metrics exist to measure these attributes. This research focuses on Static code metrics.

2.2.1 Static Code Metrics

Static code metrics offer invaluable insights into the structural and functional characteristics of source code. Among these metrics are 'LOC_ EXECUTABLE', which counts lines containing executable statements, and 'CYCLOMATIC_ COMPLEXITY', which measures the complexity of decision structures. Additionally, metrics like 'ESSENTIAL_COMPLEXITY' and 'DESIGN_COMPLEXITY' evaluate inherent intricacies and architectural complexities, respectively. 'HALSTEAD_LENGTH' assesses code length, while 'HALSTEAD_DIFFICULTY' gauges code comprehension difficulty.

’LOC_COMMENTS’ and ’LOC_BLANK’ quantify comments and blank lines, aiding in documentation and readability. Metrics such as ’NUM_ UNIQUE_ OPERATORS’ and ’NUM_UNIQUE_OPERANDS’ capture syntactic and semantic complexities, while ’BRANCH_COUNT’ evaluates control flow complexity. Collectively, these metrics provide developers with crucial insights for understanding, maintaining, and improving software quality.

Chapter 3
MATERIALS AND METHODOLOGY

The data which were used in this research are available on the internet. NASA's Metrics Data Program (MDP) Repository [[18], [19], [20]]is a database that stores problem, product, and metrics data. The main objective of this data repository is to offer various NASA space project data to the software community. To achieve this goal, the MDP gathers artifacts from an extensive NASA dataset, computes metrics on these artifacts, and subsequently produces reports accessible to the public free of charge. The key features of the data are outlined in Table 3.1.

Data Set	Attributes	Instances	Language	Description
CM1	38	344	C	NASA spacecraft instrument
PC1	38	759	C	Flight software for earth orbiting satellite
MW1	38	264	C	Zero-gravity experiment related to combustion
MC1	39	9466	C,C++	Storage management fr ground data

Table 3.1: Description of the NASA aerospace system datasets.

The four NASA datasets (CM1, MW1, PC1, and MC1) consist of static code measures [[18], [20], [25]], including metrics such as LOC, Halfstead, and McCabe's, with their corresponding defect labels presented numerically. These metrics are derived from the product's size, complexity, and vocabulary.

3.1 SFP METHODOLOGY

The software defect prediction methodology proposed in this paper consists of two main phases: the Training phase and the Validation phase. The Training phase encompasses tasks such as data pre-processing, and classification of the training data. In contrast, the Validation phase consists evaluation of the performance of classifiers studied in this research. Furthermore, the classifiers' performance is ranked based on metrics such as classification accuracy, detection rate, True Negative Rate (TNR), Precision, and F1 score. Figure 1 illustrates the experimental architecture of this research.

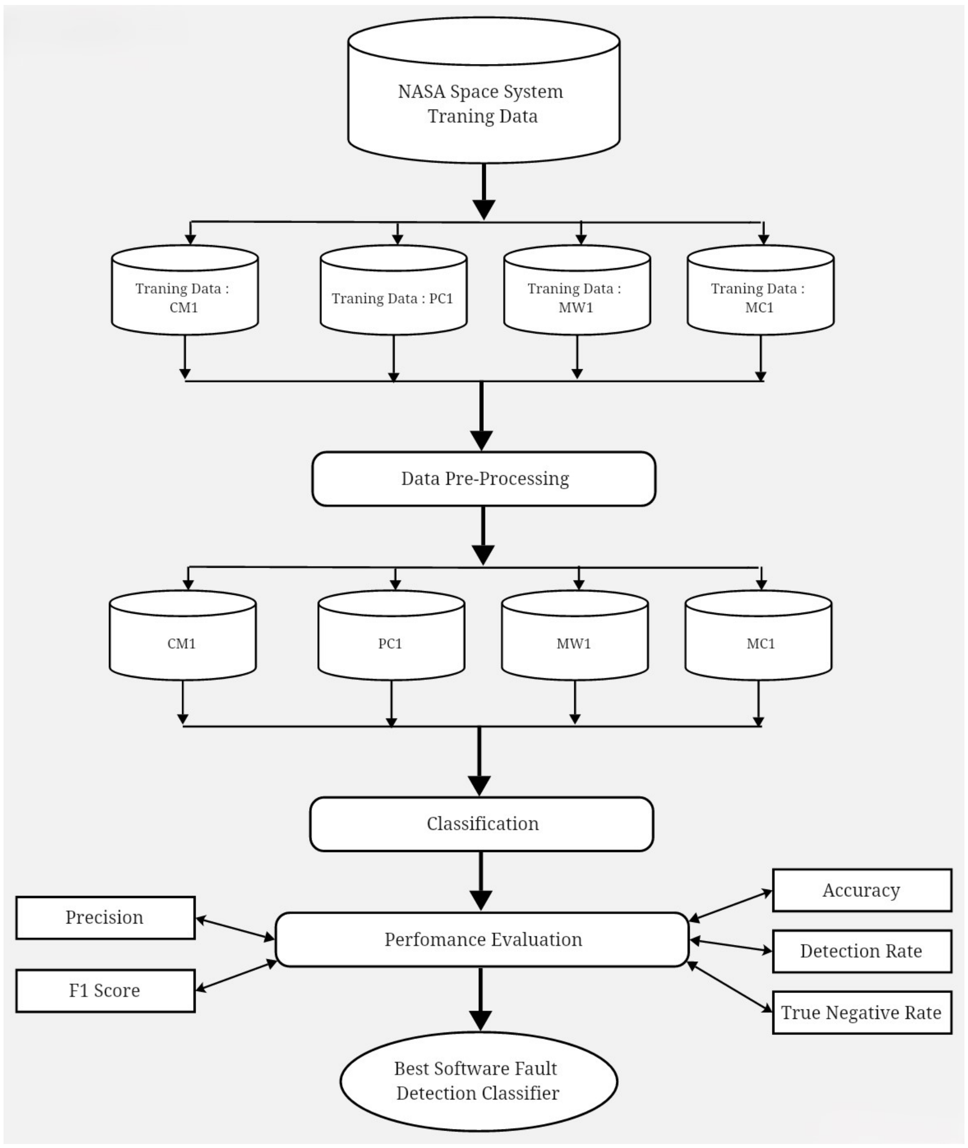

Figure 3.1: Proposed software fault prediction framework.

After conducting a thorough comparison of results, we identified the most optimal outcomes and proceeded to apply the same methodology to Recurrent Neural Network (RNN) models. The implementation of both Multi layer Perceptron (MLP) and Convolutional Neural Network (CNN) algorithms was carried out using Python 3.6, leveraging various libraries such as Keras, Numpy, Pandas, Scikit-learn, and Matplotlib for conducting experiments. Each step of the process is elaborated upon in subsequent sections. The datasets have been extensively utilized in experiments related to software defect prediction. A detailed overview of the datasets' characteristics is provided in Table 3.1. Each dataset has 20 features or attributes of each software module. The attributes are listed in table 3.2.

Attribute ID	Attribute	Definition
1	LOC	McCabe's line count of code
2	V(g)	McCabe "cyclomatic complexity"
3	V(g)/LOC	McCabe "cyclomatic density"
4	EV(G)	McCabe "essential complexity"
5	IV(G)	McCabe "design complexity"
6	L	Halstead "program length"
7	D	Halstead "difficulty"
8	I	Halstead "intelligence"
9	E	Halstead "effort"
10	EE	Halstead error est
11	C	Halstead content
12	PT	Halstead prog time
13	LOC comt	loc comments
14	LOC blnk	loc blank
15	LOC CC	loc code and comment
16	uniq Op	unique operators
17	uniq Opnd	unique operands
18	total Op	total operators
19	total Opnd	total operands
20	branchcount	of the flow graph

Table 3.2: Attributes of datasets.

Chapter 4
IMPLEMENTATION

4.1 HYPER PARAMETERS

During the training phase, various hyper-parameters will be adjusted to analyze their effect on performance. Tuning these hyper-parameters is crucial to selecting the optimal settings for achieving desired results. Optimal parameter selection is essential, yet it is often considered a complex aspect of network training. Moreover, hyper parameter tuning often relies more on practical experience rather than purely theoretical knowledge. In our research, we conducted numerous experiments to fine-tune the hyper-parameters before recording the results. For each model, we systematically tested and evaluated various values of hyper-parameters. Upon identifying an optimal value for one hyper-parameter, we proceeded to adjust the remaining hyper-parameters to further enhance the results. We then compared the performance improvements achieved by different parameter values. Additionally, we eliminated values that did not positively impact the algorithm's results. This iterative process allowed us to identify and retain only the values that contributed to the algorithm's performance enhancements.

4.1.1 Number of Epochs

The Universal rule dictates that the more you train a model, the more it gets adjusted to different data instances which helps it detect various types of data provided while testing. The over-fitting problem is not to be ignored as there is a possibility that this leads to a decrease in the efficiency of the model. This can be observed in Results[Chapter 5].

4.1.2 Activation Function

The output of a neuron is determined by the activation function on the input given. Each activation function performs a distinct mathematical operation on its input. Sigmoid, Rectified Linear Unit (ReLU), and hyperbolic tangent are commonly used as activation functions.
Sigmoid: Frequently used in back-propagation. The range is from 0 to 1. It is an appropriate extension of non-linearity that limits previous uses in NN and also presents an adequate degree of smoothness [22]. The Sigmoid

function is defined by the formula:

$$S(x) = \frac{1}{1+e^{-1}}$$

Hyperbolic Tangent (tanh): is defined as the ratio between the hyperbolic sine and the cosine functions [23]. The tanh function is defined by the formula:

$$Tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Rectified linear units (ReLUs): used as an activation function for the hidden layers in a Deep Neural Network, it becomes widely used to train a much deeper network than Sigmoid or Tanh activation functions. ReLU provides faster and more efficient learning for Deep Neural Network(DNN) over complex, high-dimensional data. The most important advantage of ReLU is that it does not require an expensive computation, just a comparison, and multiplication. ReLU has an efficient back propagation without exploding or vanishing gradient which makes it a particularly appropriate choice for DNN [24].

$$f(x) = max(0,x)$$

Swish: Its results are similar to or significantly outperforms ReLU on the deep neural network across a variety of challenging datasets. Swish defined as

$$f(x) = x * Sigmoid(x)$$

.

4.1.3 Optimizer

Adam (Adaptive Moment Estimation): An adaptive learning rate optimization algorithm that has the benefits of both momentum and RMSProp. It computes adaptive learning rates for each parameter, based on estimates of the first and second moments of the gradients. Adam dynamically adjusts the learning rate during training, making it suitable for a wide range of machine-learning tasks.
Adagrad (Adaptive Gradient Algorithm): An adaptive learning rate optimization algorithm that adjusts the learning rates of individual parameters based on historical gradients. It allocates larger learning rates to infrequently updated parameters and smaller rates to frequently updated parameters. Adagrad is particularly effective in training sparse datasets but may suffer from diminishing learning rates over time.
SGD (Stochastic Gradient Descent): A simple yet effective optimization algorithm which is commonly used in training neural networks. It updates the model parameters based on the gradient of the loss function concerning the training data. SGD operates by randomly selecting a subset of training

samples (mini-batch) to compute the gradient, leading to faster convergence compared to traditional gradient descent. However, SGD may exhibit high variance in parameter updates due to the randomness of the mini-batch selection.

4.2 DEEP LEARNING MODELS

The experiment aimed to assess the discriminatory capabilities of various metric sets. For this purpose, the machine learning algorithms were employed with their default configurations. Below is the list of algorithms utilized to construct the prediction models in this project.

Multi Layer Perceptron - A neural network that learns through back-propagation to set the appropriate weights of the connections[21]. MLP is a type of feed-forward neural network composed of multiple layers of nodes, including an input layer, one or more hidden layers, and an output layer. Each node in the network, except for those in the input layer, is a neuron that computes a weighted sum of its inputs, passes it through an activation function, and forwards the result to the subsequent layer. MLPs are well-suited for tasks such as classification and regression and have been widely used in various domains due to their flexibility and scalability. In the context of software fault prediction, MLPs can learn complex patterns from software metrics and predict the likelihood of faults in software systems.

Convolutional Neural Network - a deep learning network whose architecture is designed accordingly for processing grid-like data, like images, by leveraging layers in CNN by employing convolutional layers to detect patterns and hierarchically extract features. CNN is a type of neural network which is designed specially for processing grid-like data, such as images, by leveraging convolutional layers. CNNs consist of alternating convolutional layers, pooling layers, and fully connected layers. Convolutional layers apply convolution operations which extract spatial hierarchies of features from input data whereas pooling layers down sample the feature maps to reduce dimensionality and computational complexity simultaneously. CNNs are particularly effective in capturing spatial and temporal dependencies in data, making them suitable for tasks such as image classification and object detection. In the context of software fault prediction, CNNs can be adapted to process software metrics in a grid-like format, such as time series data, to identify patterns indicative of faults.

Recurrent Neural Network - A type of neural network suitable for sequential data processing, where connections between nodes form directed cycles, enabling the network to retain memory of past inputs and efficiently process sequences of variable lengths. RNN is a type of neural network which is designed to process sequential data by maintaining an internal state or memory. RNNs have connections that form directed cycles, which allows them to retain information about past inputs and effectively capture temporal dependencies in sequences. RNNs consist of recurrent layers that apply the same set of weights across different time steps, enabling them to process sequences of variable lengths. RNNs are very much useful for certain tasks like sequence prediction, time series analysis, and natural language processing. In accordance with software fault prediction, we can use RNNs to model the temporal dynamics of software metrics and predict the occurrence of faults based on historical data.

4.3 MODEL EVALUATION

Every model is evaluated based on five metrics which are derived from the confusion matrix on each test sample. The calculation of accuracy, precision, and recall makes use of the confusion matrix and is done by;

$$Accuracy = \frac{true\ positives + true\ negatives}{true\ positives + true\ negatives + false\ positives + false\ negatives}$$

$$Precision = \frac{true\ positives}{false\ positives + true\ positives}$$

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

$$DetectionRate = \frac{true\ positives}{true\ positives + false\ negatives}$$

$$TrueNegativeRate = \frac{true\ negatives}{true\ negatives + false\ positives}$$

$$F1Score = 2X\frac{PrecisionxRecall}{Precision + Recall}$$

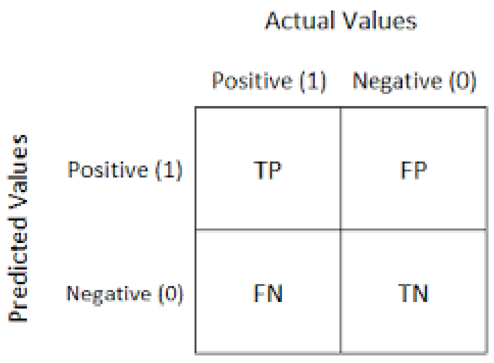

Figure 4.1: Confusion Matrix

A confusion matrix is an essential tool in evaluating the performance of a classification model. It is a structured representation of the model's predictions compared to the actual class labels in the dataset.
Actual Class: The rows in the matrix represent the actual classes or labels of the data.
Predicted Class: The columns in the matrix represent the predicted classes by the model.
True Positive (TP): This is the count of instances where the model correctly predicts the positive class.
False Positive (FP): This is the count of instances where the model incorrectly predicts the positive class when the actual class is negative.
True Negative (TN): This is the count of instances where the model correctly predicts the negative class.
False Negative (FN): This is the count of instances where the model incorrectly predicts the negative class when the actual class is positive.

Chapter 5
RESULTS

This section reports the results from the experiment described in Chapter 4. The model building and evaluation however took about 25 days. The hardware specifications are a basic laptop with 8GB RAM and a 2.40 GHz Intel i5 processor. The prediction performance is evaluated using five distinct measures outlined in Section 4.3. Initially, the accuracy of various prediction models across four datasets is assessed. Subsequently, the results of precision, detection rate, true negative, and f1-score rate are presented in sequential order. The findings from the Deep learning model evaluations inform the subsequent presentation of prediction performance across the metric sets. Tables and graphs illustrating the performance of prediction models are provided, which are used for discussions in the later chapter.

5.1 ACCURACY

Across various datasets and training epochs, accuracy serves as a fundamental metric to gauge the overall performance of the predictive models. It is a metric that measures how many times the algorithm predicts the outcome correctly. Comparing the accuracy values obtained under different parameters provides insights into the effectiveness of the models in correctly classifying software instances.

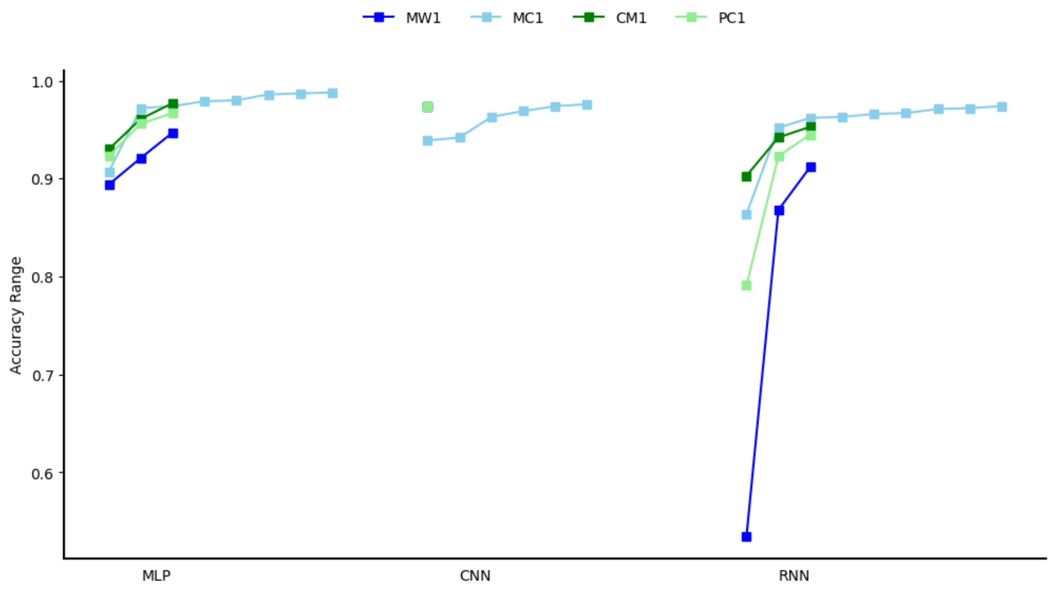

Figure 5.1: Accuracy

	CM1	MC1	PC1	MW1
Model	CNN	MLP	CNN	CNN
Mean accuracy	0.974	0.977	0.974	0.974
std. dev.	0	0.019	0	0

Table 5.1: Top-Performing Model Based on Accuracy

5.2 PRECISION

Precision reflects the models' ability to accurately classify positive predictions among all instances predicted as positive. Analyzing precision values across different settings provides insights into the models' ability to avoid false positives, which is critical in software fault prediction to prevent unnecessary alarm or intervention.

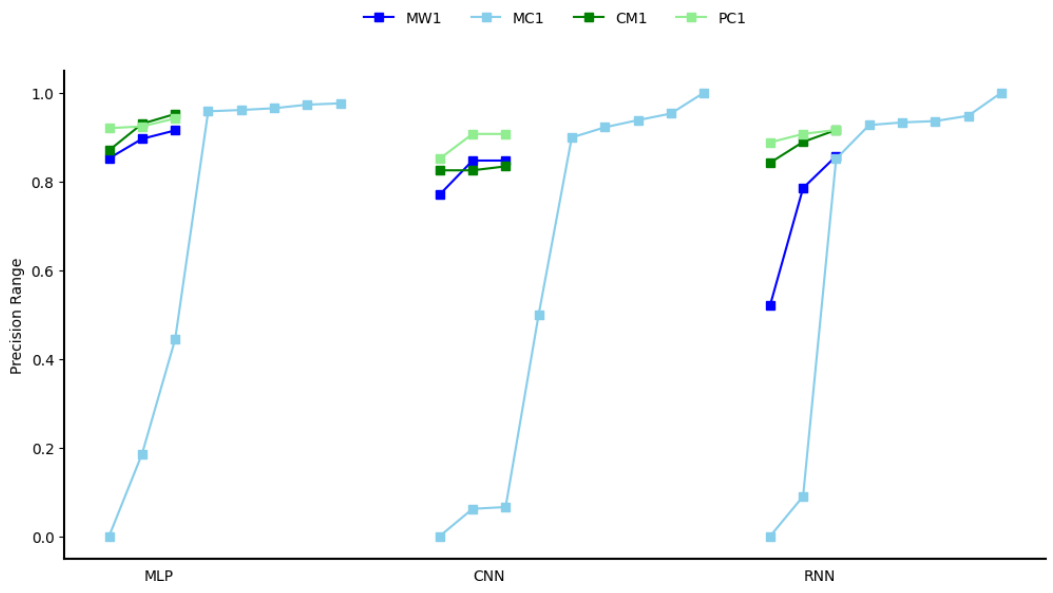

Figure 5.2: Precision

	CM1	MC1	PC1	MW1
Model	MLP	MLP	MLP	MLP
Mean precision	0.888	0.773	0.917	0.929
std. dev.	0.026	0.334	0.035	0.009

Table 5.2: Top-Performing Model Based on precision

5.3 DETECTION RATE

Detection rate, also referred to as sensitivity or recall, measures the models' ability to correctly identify positive instances, i.e., software faults. It calculates the number of correct positive predictions made as a proportion to all of them. Evaluating the detection rate across various configurations sheds light on the models' effectiveness in capturing true positive instances and minimizing false negatives, which is essential for reliable fault detection.

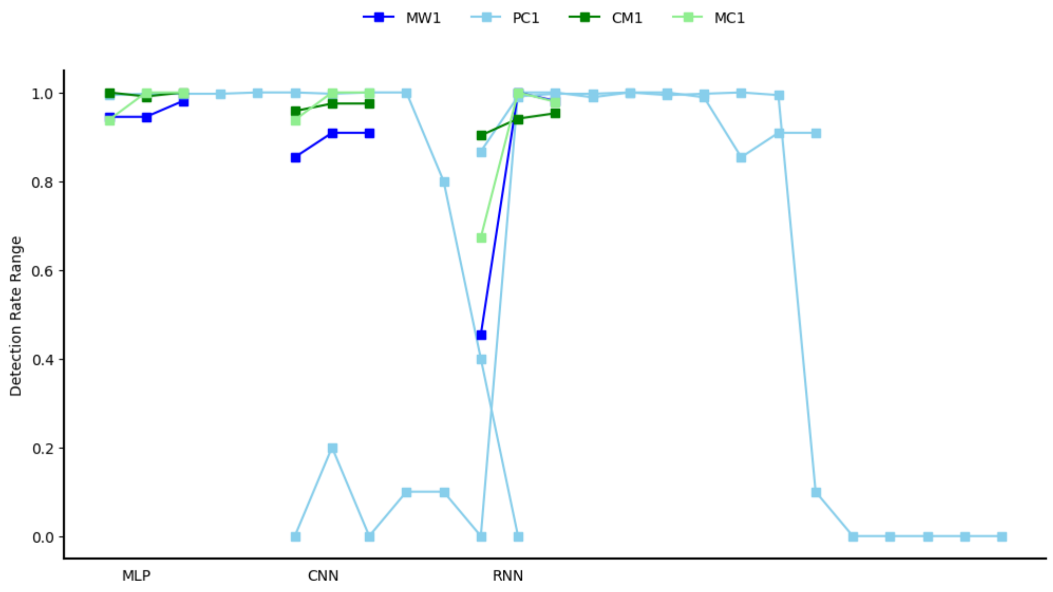

Figure 5.3: Detection rate

	CM1	MC1	PC1	MW1
Model	MLP	MLP	MLP	CNN& MLP
Mean detection rate	0.957	0.846	0.997	0.978
std. dev.	0.017	0.305	0.004	0.023

Table 5.3: Top-Performing Model Based on detection rate

5.4 TN RATE

TNR, also known as specificity, represents the ability of the models to correctly identify negative instances. How many instances the model predicted negative which originally turn out to be negative. Analyzing TNR across different datasets and epochs offers valuable insights into the models' capability to avoid false positives, which is crucial in software fault prediction to minimize misclassification of non-faulty instances.

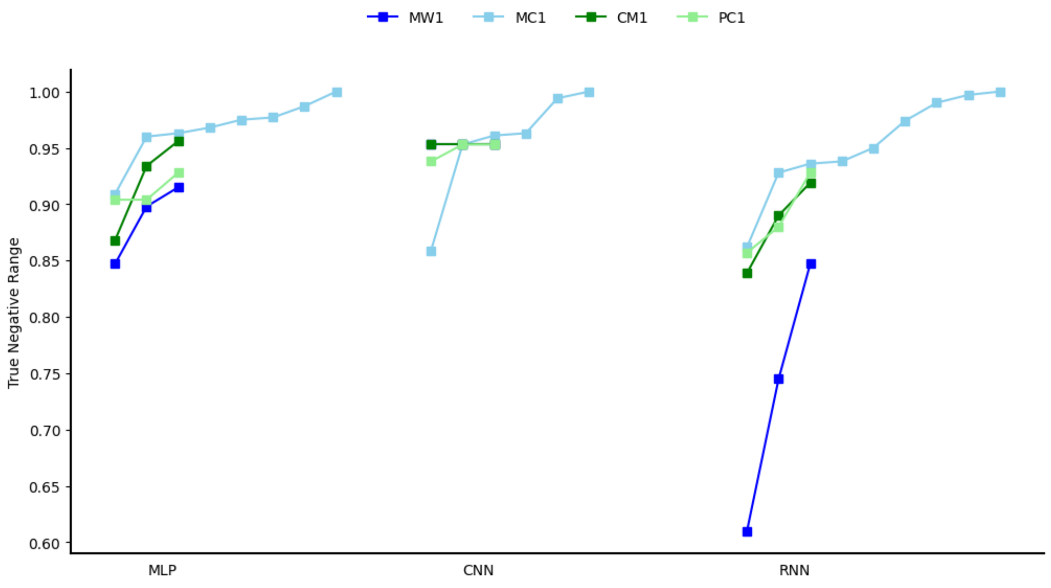

Figure 5.4: True negative rate

	CM1	MC1	PC1	MW1
Model	CNN	MLP	CNN	CNN
Mean TNR	0.953	0.963	0.953	0.953
std. dev.	0	0.022	0	0

Table 5.4: Top-Performing Model Based on tnr

5.5 F1-SCORE

F1-score, the harmonic mean of precision and recall, offers a balanced assessment of the models' performance, considering both false positives and false negatives. The relative contribution of both these metrics consists of the f1-score. Examining F1-scores across various configurations helps in understanding the overall effectiveness of the models in software fault prediction tasks.

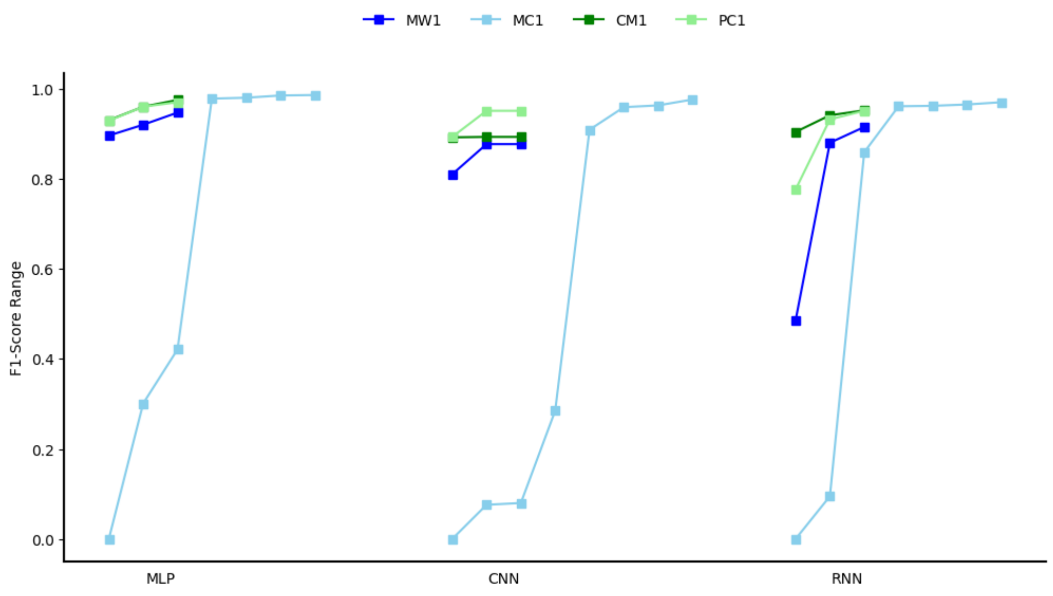

Figure 5.5: F1-score

	CM1	MC1	PC1	MW1
Model	MLP	MLP	MLP	MLP
Mean F1-score	0.921	0.792	0.955	0.953
std. dev.	0.0208	0.331	0.019	0.017

Table 5.5: Top-Performing Model Based on f1-score

Chapter 6
ANALYSIS AND DISCUSSION

6.1 MLP

6.1.1 Number of Epochs

Across all datasets and epochs, there is a consistent trend of accuracy improvement as epochs increase. For instance, in the CM1 dataset, accuracy increases from 89.5% at 100 epochs to 94.7% at 10000 epochs. The performance metrics vary across datasets, indicating that models may perform differently depending on the dataset characteristics. For example, the detection rate for MC1 consistently remains high compared to other datasets, even though the accuracy varies.
Consistent High Performance: Despite variability across datasets, certain metrics maintain high values consistently across epochs. For instance, the precision values for the MW1 dataset consistently exceed 95% across all epochs. Some metrics, such as accuracy and F1-score, seem to converge to a stable value as the number of epochs increases. This suggests that the models may have reached a point of stability or saturation in performance. There are instances where the performance metrics for training data (e.g., detection rate) are exceptionally high, indicating potential overfitting. This is particularly evident in the MC1 dataset, where detection rates reach 100% for certain epochs.

Epochs	CM1					MC1				
	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score
100	0.895	0.945	0.847	0.852	0.897	0.981	0.995	0.968	0.966	0.980
1000	0.921	0.945	0.898	0.897	0.920	0.986	0.997	0.975	0.974	0.985
10000	0.947	0.982	0.915	0.915	0.947	0.987	0.997	0.978	0.976	0.987

Table 6.1: NUMBER OF EPOCHS COMPARISON

Epochs	MW1					PC1				
	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score
100	0.923	0.939	0.905	0.92	0.929	0.930	1.0	0.869	0.871	0.931
1000	0.967	1.0	0.929	0.942	0.970	0.961	0.992	0.934	0.930	0.96
10000	0.956	1.0	0.905	0.925	0.961	0.977	1.0	0.956	0.953	0.976

Table 6.2: NUMBER OF EPOCHS COMPARISON

6.1.2 Activation Function

Firstly, focusing on accuracy, ReLU achieves the highest accuracy score of 0.987, indicating its effectiveness in correctly classifying instances. Tanh follows closely behind with an accuracy of 0.981, while Swish achieves an accuracy of 0.979. Next, examining the detection rate, tanh and Swish both demonstrate perfect detection rates of 1.0, implying that they accurately identify all positive instances in the dataset. ReLU also achieves a high detection rate of 0.997, indicating its strong performance in identifying positive cases. Moving on to the true negative rate (TNR), ReLU and swish both achieve TNR values of 0.978 and 0.961, respectively. Tanh, however, exhibits a slightly lower TNR of 0.963, suggesting a higher rate of false positives compared to the other activation functions.
Precision measures the proportion of true positive classifications among all positive predictions made by the model. ReLU achieves the highest precision of 0.976, followed by tanh with a precision of 0.961 and swish with a precision of 0.959. Lastly, the F1 score, which considers both precision and recall, also highlights ReLU as the top performer with an F1 score of 0.987. Tanh and Swish follow closely behind with F1 scores of 0.980 and 0.979, respectively. In summary, ReLU emerges as the most effective activation function for the MLP model trained on the MC1 dataset, demonstrating superior performance across multiple key metrics including accuracy, precision, and F1 score. However, tanh and swish also exhibit strong performance, particularly in terms of detection rate, indicating their potential suitability for certain applications or datasets.

MC1						
		Accuracy	Detection rate	TNR	precision	F1 score
MLP	ReLU	0.987	0.997	0.978	0.976	0.987
	tanh	0.981	1.0	0.963	0.961	0.980
	swish	0.979	1.0	0.961	0.959	0.979

Table 6.3: ACTIVATION COMPARISON

6.1.3 Optimizer

Starting with the performance under the Adam optimizer, ReLU activation achieves the highest accuracy score of 0.987, indicating its effectiveness in correctly classifying instances. Tanh closely follows with an accuracy of 0.981, while logistic achieves an accuracy of 0.980. In terms of detection rate, ReLU and tanh both demonstrate perfect detection rates of 1.0, indicating accurate identification of all positive instances. Logistic also achieves a perfect detection rate under Adam. Moving on to the true negative rate (TNR), ReLU and tanh both achieve TNR values of 0.978 and 0.963, respectively, under the Adam optimizer. Logistic achieves a TNR of 0.961. Precision measures the proportion of true positive classifications among all positive predictions made by the model.
ReLU achieves the highest precision of 0.976, followed by tanh with a precision of 0.961, and logistic with a precision of 0.959. Lastly, the F1 score, which balances precision and recall, highlights ReLU as the top performer under Adam with an F1 score of 0.987, followed by tanh with an F1 score of 0.980, and logistic with an F1 score of 0.979. Comparing the performance under the SGD optimizer, ReLU still achieves the highest accuracy among the activation functions, although it decreases to 0.8. Tanh and logistic exhibit accuracies of 0.4 and 0.0, respectively, under SGD. Interestingly, while ReLU and tanh maintain perfect detection rates of 1.0 under SGD, the logistic's detection rate drops to 0.0. In summary, under both Adam and SGD optimizers, ReLU and tanh activations consistently outperform logistic activation across various metrics. ReLU emerges as the top performer, demonstrating strong accuracy, detection rate, TNR, precision, and F1-score under both optimizers, highlighting its effectiveness in the MLP model trained on the MC1 dataset.

MC1										
Activations	adam					sgd				
	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score
ReLU	0.987	0.997	0.978	0.976	0.987	0.8	0.910	0.907	0.186	0.302
tanh	0.981	1.0	0.963	0.961	0.980	0.4	0.987	0.972	0.444	0.421
logistic	0.980	1.0	0.961	0.959	0.979	0.0	1.0	0.975	0.0	0.0

Table 6.4: OPTIMISER COMPARISON

6.2 CNN

6.2.1 Number of Epochs

For the dataset CM1, with 100 epochs, the model achieves an accuracy of 0.974, indicating a high level of correct classifications. The detection rate, representing the model's ability to identify positive instances, stands at 0.855. The true negative rate, indicating the model's ability to correctly identify negative instances, is measured at 0.953. Precision, which quantifies the model's ability to make correct positive predictions, is 0.771. Similar performance trends are observed for the 1000 and 10000 epochs. For the dataset MC1, the model maintains an accuracy of 0.974 across all epochs, indicating consistent performance. The detection rate remains high at 0.919 for 100 epochs and 1.0 for 1000 and 10000 epochs.
The true negative rate is consistent at 0.953 for all epochs, and precision ranges from 0.899 to 0.923. Moving to the dataset MW1, the model again demonstrates an accuracy of 0.974 across all epochs, suggesting consistent performance. The detection rate remains high at 1.0 for all epochs. The true negative rate is consistent at 0.953, and the precision ranges from 0.877 to 0.923. For the dataset PC1, the model achieves an accuracy of 0.974 across all epochs, reflecting consistent performance. The detection rate ranges from 0.939 to 1.0, indicating effective identification of positive instances. The true negative rate remains consistent at 0.953, and precision ranges from 0.907 to 0.923. Overall, the model demonstrates consistent and high-performance metrics across different datasets and epochs, indicating its robustness and reliability in classification tasks. The F1-score, which balances precision and recall, is not provided in the table but would offer further insight into the model's overall performance.

Epochs	CM1					MC1				
	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score
100	0.974	0.855	0.953	0.771	0.810	0.974	0.919	0.953	0.899	0.909
1000	0.974	0.909	0.953	0.847	0.877	0.974	1.0	0.953	0.923	0.960
10000	0.974	0.909	0.953	0.847	0.877	0.974	1.0	0.953	0.923	0.960

Table 6.5: NUMBER OF EPOCHS COMPARISON

Epochs	MW1					PC1				
	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score
100	0.974	0.939	0.953	0.852	0.893	0.974	0.959	0.953	0.835	0.892
1000	0.974	1.0	0.953	0.907	0.951	0.974	0.975	0.953	0.825	0.894
10000	0.974	1.0	0.953	0.907	0.951	0.974	0.975	0.953	0.825	0.894

Table 6.6: NUMBER OF EPOCHS COMPARISON

6.2.2 Activation Function

For the ReLU activation function, the CNN model achieves an accuracy of 0.974, indicating robust classification performance. The detection rate, representing the model's ability to identify positive instances, is 1.0, indicating that all positive instances are correctly identified. The true negative rate, indicating the model's ability to correctly identify negative instances, is 0.953. Precision, which quantifies the model's ability to make correct positive predictions, is measured at 0.923. The F1-score, which balances precision and recall, is 0.960, indicating effective overall performance. Using the tanh activation function, the CNN model achieves a slightly higher accuracy of 0.977 compared to ReLU.
The detection rate remains high at 1.0, indicating effective identification of positive instances. The true negative rate is 0.860, slightly lower than with ReLU. Precision is notably higher at 0.954, indicating fewer false positives. The F1-score is 0.976, indicating a balance between precision and recall and overall strong performance. For the swish activation function, the CNN model achieves an accuracy of 0.964, slightly lower than with ReLU and Tanh. The detection rate remains high at 0.990, indicating effective identification of positive instances. The true negative rate is consistent with tanh at 0.860. Precision is slightly lower at 0.939 compared to tanh. The F1-score is 0.963, indicating good overall performance but slightly lower than with Tanh. Overall, the tanh activation function exhibits the highest precision and F1-score among the three activation functions evaluated, indicating its effectiveness in this CNN model for the MC1 dataset.

MC1						
		Accuracy	Detection rate	TNR	precision	F1 score
CNN	ReLU	0.974	1.0	0.953	0.923	0.960
	tanh	0.977	1.0	0.860	0.954	0.976
	swish	0.964	0.990	0.860	0.939	0.963

Table 6.7: ACTIVATION COMPARISON

6.2.3 Optimizer

For the ReLU activation function with the Adam optimizer, the model achieves an accuracy of 0.974. The detection rate is 1.0, indicating effective identification of positive instances, while the true negative rate is 0.953, showing strong performance in identifying negative instances. Precision is measured at 0.923, indicating a relatively low false positive rate. The F1-score is 0.960, suggesting a balance between precision and recall. With the SGD optimizer, the performance significantly drops, with an accuracy of 0.942. This suggests that SGD may not be suitable for this dataset with the ReLU activation function. Additionally, the F1-score is notably lower at 0.08 compared to Adam. Using the tanh activation function, the Adam optimizer achieves a slightly higher accuracy of 0.977 compared to ReLU. The detection rate remains high at 1.0, indicating effective identification of positive instances.
However, the true negative rate decreases to 0.860, indicating a slight decrease in the model's ability to correctly identify negative instances. Precision is notably higher at 0.954 compared to ReLU with Adam. The F1-score is 0.976, indicating strong overall performance. With the SGD optimizer, the accuracy decreases to 0.940, but the F1-score improves to 0.770 compared to ReLU with SGD, suggesting that tanh may be more suitable with SGD. For the swish activation function, both Adam and Adgrad optimizers achieve high accuracies of 0.964 and 0.975, respectively. The detection rate remains high at around 0.989-0.990, indicating effective identification of positive instances. However, the true negative rate decreases with adgrad to 0.860, suggesting a potential trade-off in correctly identifying negative instances. Precision varies between 0.938 and 1.0, depending on the optimizer, while the F1-score remains high, indicating overall strong performance. Overall, the choice of activation function and optimizer significantly influences model performance, with Tanh generally performing well across all optimizers, while ReLU may require careful selection of the optimizer for optimal results.

MC1															
Activations	adam					sgd					adgrad				
	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score
ReLU	0.974	1.0	0.953	0.923	0.960	0.942	0.1	0.964	0.666	0.08	0.970	0.0	0.995	0.0	0.0
tanh	0.977	1.0	0.860	0.954	0.976	0.940	0.1	0.961	0.0625	0.770	0.975	0.2	0.995	0.5	0.286
swish	0.964	0.989	0.860	0.938	0.963	0.975	0.0	1.0	0.0	0.0	0.975	0.0	1.0	1.0	0.0

Table 6.8: OPTIMISER COMPARISON

6.3 RNN

6.3.1 Number of Epochs

For the dataset CM1, with 100 epochs, the model achieves an accuracy of 0.535, indicating relatively lower performance compared to other datasets. The detection rate, representing the model's ability to identify positive instances, is 0.454. The true negative rate, indicating the model's ability to correctly identify negative instances, is measured at 0.610. Precision, which quantifies the model's ability to make correct positive predictions, is 0.521. The F1-score, which balances precision and recall, is 0.485. As the number of epochs increases to 1000 and 10000, there's a significant improvement in performance metrics across the board. For the dataset MC1, the model achieves a high accuracy of 0.864 with 100 epochs, indicating effective classification performance. The detection rate remains consistently high, ranging from 0.866 to 1.0 across different numbers of epochs. The true negative rate ranges from 0.862 to 0.997, indicating the model's ability to correctly identify negative instances.
Precision ranges from 0.851 to 0.939, reflecting the model's ability to make correct positive predictions. The F1-score ranges from 0.859 to 0.962, indicating a balance between precision and recall. Moving to the dataset MW1, the model demonstrates an accuracy of 0.791 with 100 epochs, reflecting relatively lower performance compared to other datasets. The detection rate ranges from 0.673 to 0.929 across different numbers of epochs. The true negative rate ranges from 0.776 to 0.923, indicating varying levels of performance in correctly identifying negative instances. Precision ranges from 0.903 to 0.953, reflecting the model's ability to make correct positive predictions. The F1-score ranges from 0.839 to 0.916. For the dataset PC1, the model achieves an accuracy of 0.904 with 100 epochs, indicating relatively high performance compared to other datasets. The detection rate ranges from 0.975 to 0.992 across different numbers of epochs. The true negative rate ranges from 0.839 to 0.920, indicating varying levels of performance in correctly identifying negative instances. Precision ranges from 0.843 to 0.916, reflecting the model's ability to make correct positive predictions. The F1-score ranges from 0.904 to 0.952, indicating a balance between precision and recall.

epochs	CM1					MC1				
	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score
100	0.535	0.454	0.610	0.521	0.485	0.864	0.866	0.862	0.851	0.859
1000	0.868	1.0	0.746	0.786	0.88	0.963	0.992	0.936	0.934	0.962
10000	0.912	0.982	0.847	0.857	0.915	0.967	0.997	0.939	0.937	0.966

Table 6.9: NUMBER OF EPOCHS COMPARISON

epochs	MW1					PC1				
	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score
100	0.791	0.673	0.929	0.917	0.776	0.903	0.975	0.839	0.843	0.904
1000	0.945	1.0	0.881	0.907	0.951	0.942	1.0	0.891	0.890	0.942
10000	0.923	0.980	0.857	0.889	0.932	0.953	0.992	0.920	0.916	0.952

Table 6.10: NUMBER OF EPOCHS COMPARISON

6.3.2 Activation Function

Using the ReLU activation function, the RNN model achieves an accuracy of 0.967. The detection rate is high at 0.997, indicating effective identification of positive instances. The true negative rate stands at 0.939, suggesting a good ability to correctly identify negative instances. Precision is measured at 0.937, indicating a relatively low false positive rate. The F1-score is 0.966, reflecting a balance between precision and recall. When employing the tanh activation function, the model achieves a slightly lower accuracy of 0.963 compared to ReLU. However, the detection rate remains high at 1.0, indicating effective identification of positive instances. The true negative rate decreases slightly to 0.929, suggesting a slight decrease in the model's ability to correctly identify negative instances. Precision is relatively high at 0.927, and the F1-score is 0.962, indicating strong overall performance. Using the swish activation function, the RNN model achieves the highest accuracy of 0.972 among all activation functions. The detection rate remains high at 0.995, indicating effective identification of positive instances. The true negative rate increases to 0.951, suggesting improved performance in correctly identifying negative instances compared to ReLU and Tanh. Precision is relatively high at 0.948, and the F1-score is 0.971, indicating excellent overall performance. In summary, the choice of activation function significantly influences the RNN model's performance on the MC1 dataset, with swish yielding the highest accuracy and overall strong performance across all metrics.

MC1						
		Accuracy	Detection rate	TNR	precision	F1 score
RNN	ReLU	0.967	0.997	0.939	0.937	0.966
	tanh	0.963	1.0	0.929	0.927	0.962
	swish	0.972	0.995	0.951	0.948	0.971

Table 6.11: ACTIVATION COMPARISON

6.3.3 Optimizer

Using the Adam optimizer with the ReLU activation function, the model achieves an accuracy of 0.967. The detection rate is high at 0.997, indicating effective identification of positive instances. The true negative rate stands at 0.939, suggesting a good ability to correctly identify negative instances. Precision is measured at 0.937, indicating a relatively low false positive rate. The F1-score is 0.966, reflecting a balance between precision and recall. When employing the SGD optimizer with the same activation function (ReLU), the accuracy slightly increases to 0.952. However, the detection rate decreases to 0.1, indicating a significant decrease in the ability to identify positive instances effectively. The true negative rate increases to 0.974, suggesting an improvement in correctly identifying negative instances compared to the Adam optimizer. Precision decreases to 0.909, and the F1-score decreases to 0.095, indicating a decline in overall performance compared to Adam. Using the adgrad optimizer with the ReLU activation function, the accuracy remains at 0.975, indicating consistent performance across optimizers. The detection rate remains high at 1.0, indicating effective identification of positive instances. The true negative rate remains at 0.939, similar to Adam's, suggesting consistent performance in correctly identifying negative instances. Precision remains at 0.937, and the F1-score remains at 0.966, indicating consistent performance compared to Adam.
When employing the tanh activation function with different optimizers, the model's performance varies. With the Adam optimizer, the accuracy decreases slightly to 0.963 compared to ReLU. However, the detection rate remains high at 1.0, indicating effective identification of positive instances. The true negative rate decreases slightly to 0.929, suggesting a slight decrease in the ability to correctly identify negative instances compared to ReLU. Precision remains at 0.927, and the F1-score remains at 0.962, indicating consistent performance compared to ReLU. Using the SGD optimizer with tanh, the accuracy decreases to 0.972 compared to Adam. The detection rate remains high at 0.997, indicating effective identification of positive instances. The true negative rate remains at 0.0, indicating a decrease in the ability to correctly identify negative instances compared to Adam. Precision decreases to 0.0, and the F1-score decreases to 0.972, indicating a decline in overall performance compared to Adam. With the adgrad optimizer and tanh, the accuracy remains at 0.975, consistent with ReLU. The detection rate remains high at 1.0, indicating effective identification of positive instances. The true negative rate remains at 0.0, similar to SGD, suggesting a decrease in the ability to correctly identify negative instances compared to Adam. Precision remains at 0.937, and the F1-score remains at 0.966, indicating consistent performance compared to ReLU. Overall, the choice of optimizer and activation function significantly influences the model's performance on the MC1 dataset, with Adam and ReLU yielding the highest accuracy and overall strong performance across all metrics.

MC1															
Activations	adam					sgd					adgrad				
	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score
ReLU	0.967	0.997	0.939	0.937	0.966	0.952	0.1	0.974	0.909	0.095	0.975	0.0	1.0	1.0	0.0
tanh	0.963	1.0	0.929	0.927	0.962	0.972	0.0	0.997	0.0	0.0	0.972	0.0	0.997	0.0	0.0
swish	0.972	0.995	0.951	0.948	0.971	0.975	0.0	1.0	0.0	0.0	0.972	0.0	0.997	0.0	0.0

Table 6.12: OPTIMIZER COMPARISON

Explaining the rationale behind selecting dataset MC1 for subsequent analysis based on its superior accuracy provides context for the subsequent comparisons. Dataset MC1 exhibited the highest accuracy among all datasets, making it a suitable choice for further analysis. The decision to focus on MC1 for subsequent comparisons is based on its superior performance in terms of accuracy, indicating its relevance and effectiveness in software fault prediction tasks. Analyzing the performance of different activation functions—ReLU, tanh, and swish—on dataset MC1 with 10000 epochs reveals notable differences. ReLU and tanh consistently outperform Swish across all metrics. Except in RNN the dataset MC1 had a bit better accuracy on swish (97%). But for comparison purposes, we carried on with three of the activation functions onto the next subsequent parameters. A detailed comparison of activation functions (ReLU, tanh, swish) on dataset MC1 with 10000 epochs explores their impact on accuracy, TNR, detection rate, precision, and F1-score. This analysis helps in identifying the activation function that yields the best performance for software fault prediction tasks. Comparing the performance of different optimizers—Adam, SGD, and adgrad—on dataset MC1 with 10000 epochs highlights significant variations. Adam consistently yields the highest performance across all metrics, followed by SGD and Adgrad. The superior performance of Adam can be attributed to its adaptive learning rate mechanism, which enables faster convergence and better optimization of model parameters. Hence, Adam is chosen as the preferred optimizer for subsequent analysis.
An overarching comparison of the experimental results synthesizes the findings from accuracy, TNR, detection rate, precision, and F1-score analysis. This section concludes by summarizing the optimal configurations for software fault prediction models based on the comprehensive evaluation of various metrics. Training models on dataset MC1 with 10000 epochs using the ReLU activation function and the Adam optimizer consistently results in the highest accuracy, TNR, detection rate, precision, and F1 score. This combination represents the most effective configuration for software fault prediction tasks, providing reliable and accurate predictions. These findings provide valuable insights on how tuning of hyper parameters affects the performance of neural network models. Selecting the optimal combination of

epochs, activation function, and optimizer can enhance the predictive accuracy and reliability of software fault prediction models, thereby improving software quality and reliability.

6.4 MODEL RESULTS

6.4.1 MLP Model

CM1 Dataset: Achieved an accuracy of 0.947, with a detection rate of 0.982 and a true negative rate of 0.915. Precision and F1-score were 0.915 and 0.947, respectively.

MC1 Dataset: Demonstrated high performance with an accuracy of 0.987, a detection rate of 0.997, and a true negative rate of 0.978. Precision and F1-score were 0.976 and 0.987, respectively.

MW1 Dataset: Maintained a high accuracy of 0.987, with a detection rate of 1.0 and a true negative rate of 0.976. Precision and F1-score were 0.956 and 0.976, respectively.

PC1 Dataset: Showcased strong performance with an accuracy of 0.947, a detection rate of 0.975, and a true negative rate of 0.956. Precision and F1-score were 0.953 and 0.976, respectively.

6.4.2 CNN Model

CM1 Dataset: Achieved an accuracy of 0.974, with a detection rate of 0.909 and a true negative rate of 0.953. Precision and F1-score were 0.847 and 0.877, respectively.

MC1 Dataset: Demonstrated excellent performance with an accuracy of 0.974, a detection rate of 1.0, and a true negative rate of 0.953. Precision and F1-score were 0.923 and 0.960, respectively.

MW1 Dataset: Maintained a high accuracy of 0.974, with a detection rate of 1.0 and a true negative rate of 0.953. Precision and F1-score were 0.923 and 0.960, respectively.

PC1 Dataset: Showcased strong performance with an accuracy of 0.974, a detection rate of 0.975, and a true negative rate of 0.953. Precision and F1-score were 0.825 and 0.894, respectively.

6.4.3 RNN Model

CM1 Dataset: Achieved an accuracy of 0.912, with a detection rate of 0.982 and a true negative rate of 0.847. Precision and F1-score were 0.857 and 0.915, respectively.

MC1 Dataset: Demonstrated excellent performance with an accuracy of 0.966, a detection rate of 0.997, and a true negative rate of 0.938. Precision and F1-score were 0.937 and 0.966, respectively.

MW1 Dataset: Maintained a high accuracy of 0.997, with a detection rate of 1.0 and a true negative rate of 0.939. Precision and F1-score were 0.937 and 0.966, respectively.

PC1 Dataset: Showcased strong performance with an accuracy of 0.945, a detection rate of 1.0, and a true negative rate of 0.937. Precision and F1-score were 0.923 and 0.952, respectively.

Models	CM1					MC1				
	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score
MLP	0.947	0.982	0.915	0.915	0.947	0.987	0.997	0.978	0.976	0.987
CNN	0.974	0.909	0.953	0.847	0.877	0.974	1.0	0.953	0.923	0.960
RNN	0.912	0.982	0.847	0.857	0.915	0.966	0.997	0.938	0.937	0.966

Table 6.13: MODEL RESULTS

Models	MW1					PC1				
	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score	Accuracy	Detection Rate	True Negative Rate	Precision	F1-score
MLP	0.967	1.0	0.929	0.942	0.970	0.977	1.0	0.956	0.953	0.976
CNN	0.974	1.0	0.953	0.907	0.951	0.974	0.975	0.953	0.825	0.894
RNN	0.945	1.0	0.881	0.907	0.951	0.953	0.992	0.920	0.916	0.952

Table 6.14: MODEL RESULTS

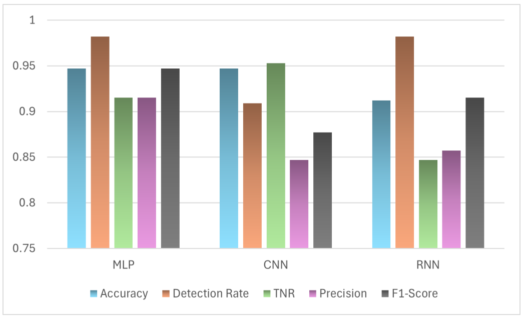

Figure 6.1: CM1

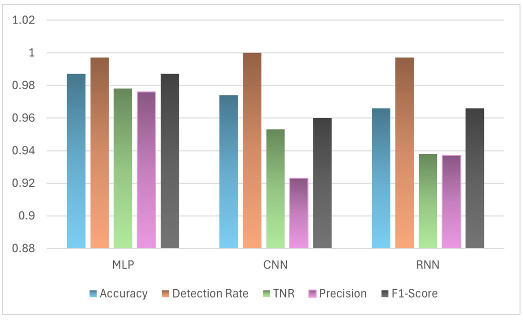

Figure 6.2: MC1

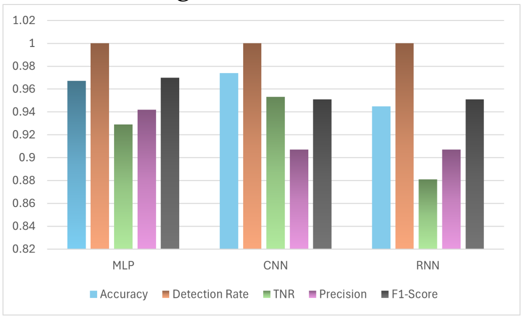

Figure 6.3: MW1

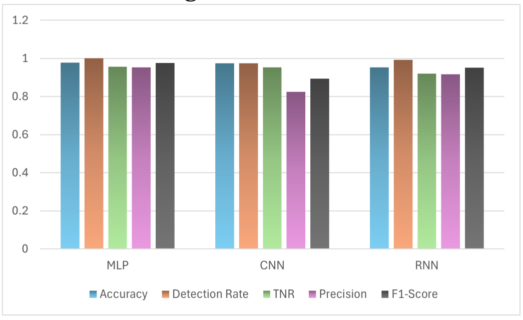

Figure 6.4: PC1

6.5 COMPARISON TO OTHER STUDIES

The comparison between the results from the previous study (Table 6.15) and the present study (Table 6.16) reveals notable advancements in model performance, particularly for the MLP model. In the previous study, MLP achieved accuracies of 0.935 on PC1 and 0.905 on CM1, with detection rates of 0.363 and 0.191, respectively. However, in the present study, MLP showed remarkable improvement, achieving accuracies of 0.977 on PC1 and 0.947 on CM1, with detection rates of 1.0 and 0.982, respectively. This substantial enhancement in detection rates indicates the model's increased capability to correctly identify positive instances in the datasets. Furthermore, CNN's performance remained consistently high across both studies, with accuracies of 0.978 and 0.974 on PC1 and 0.973 and 0.974 on CM1 in the previous and present studies, respectively. These findings suggest that while both MLP and CNN models demonstrate strong performance, the MLP model exhibits significant improvement in detection rates and overall accuracy, making it the better-performing model in the present study compared to the previous one.

Models	PC1			CM1		
	Accuracy	Detection Rate	True Negative Rate	Accuracy	Detection Rate	True Negative Rate
MLP	0.935	0.363	0.977	0.905	0.191	0.979
CNN	0.978	0.739	0.996	0.973	0.823	0.992

Table 6.15: BASE PAPER RESULTS

Models	PC1			CM1		
	Accuracy	Detection Rate	True Negative Rate	Accuracy	Detection Rate	True Negative Rate
MLP	0.977	1.0	0.956	0.947	0.982	0.915
CNN	0.974	0.975	0.953	0.974	0.909	0.953

Table 6.16: PRESENT PAPER RESULTS

6.5.1 Improvement Rate

The improvement rate describes how much better the new enhanced model is working compared to the models used in previous studies. Due to limited performance measures of previous study, only three performance metrics have been considered. The improvement rate is calculated using the formula below.

$$Improvement\_rate_{(i)} = \frac{i(new) - i(old)}{i(old)} * 100$$

$$Overall\_Improvementrate = \frac{Improvement\_rate_{(1)} + ... + Improvement\_rate_{(i)}}{i}$$

Dataset	PC1		CM1	
Model	MLP	CNN	MLP	CNN
Overall Improvement rate	59.24	9.07	143.24	2.20

Table 6.17: Improvement Rate

The values in table 6.16 are considered as new and table 6.15 as old and 'i' representing each model performance measure. The table 6.17 presents a comparison of machine learning models MLP and CNN across two datasets, PC1 and CM1, highlighting their respective improvement rates over previous models. For the PC1 dataset, MLP exhibits a substantial improvement rate of 59.24%, signifying significant progress over the prior model, while CNN demonstrates a more modest improvement rate of 9.07%. Conversely, on the CM1 dataset, MLP showcases a remarkable improvement rate of 143.24%, indicating a substantial enhancement compared to the previous model, whereas CNN shows a comparatively minimal improvement rate of 2.20%. These findings underscore the effectiveness of the MLP model in achieving notable advancements across both datasets, positioning it as a favorable choice for software fault prediction tasks.

Chapter 7
CONCLUSION AND FUTURE WORK

According to the data outlined in Chapters 5 and 6, MLP demonstrates a notable enhancement in performance, whereas CNN maintains a consistently high accuracy rate with marginal improvements. All the three of combination epochs, activation functions, and optimizers are considered in this enhancement. As mentioned in Chapter 5 with respect to individual performance metrics, all the metrics excluding the parameter accuracy, indicate that MLP is the better model for SFP as accuracy provides an overall view of the model's correctness, it may not capture the complete picture, especially in imbalanced datasets. In the case of RNN, the performance was not up to the mark due to its nature of handling sequential data where the metric data is non-sequential. As the hidden memory state is not used, resulted in additional noise in the data and subsequently leading to the depleting performance. This research concludes an important note that despite the higher accuracy of CNN among all the DL models, MLP has a suitable performance considering all the performance measures.
The results also suggest that the peak of software fault prediction using software metric data with both viability and availability of metrics have been done towards superlative and exhaustive results since the software development progresses into a new era of AI and its influence. Further work can be done on the semantic understanding of the code rather than the static code analysis and static code metrics.

Chapter 8
LIST OF ABBREVATIONS

• SFP	-	Software Fault Prediction
• RNN	-	Recurrent Neural Networks
• CNN	-	Convolutional Neural Networks
• MLP	-	Multi Layer Perceptron
• SCM	-	Source Code Management
• LOC	-	Lines of Code
• MDP	-	Metrics Data Program
• TNR	-	True Negative Rate
• DNN	-	Deep Neural Network
• SGD	-	Stochastic Gradient Descent
• ReLU	-	Rectified Linear Unit

REFERENCES

[1] **Qasem, Osama Al and Akour, Mohammed and Alenezi, Mamdouh**,"The Influence of Deep Learning Algorithms Factors in Software Fault Prediction",*IEEE Access*, VOLUME 8, pp. 63945-63960, 2020.

[2] **M. Liljeson and A. Mohlin,** , "Software defect prediction using machine learning on test and source code metrics", Dissertation, 2014.

[3] **Jacob, S. and Raju**, "Software defect prediction in large space systems through hybrid feature selection and classification", Vol. 14,208-214, Mar 1959.

[4] **Saraygord Afshari and Fatemeh Enayatollahi and Xiangyang Xu and Xihui Liang**, "Machine learning-based methods in structural reliability analysis: A review",Reliability Engineering & System Safety,Volume 219,ISSN:0951-8320, Mar 2022.

[5] **Barlow, Richard E. and Proschan, Frank**, (1996), Mathematical Theory of Reliability [Online]. Available: `https://epubs.siam.org/doi/abs/10.1137/1.9781611971194`.

[6] **D. Himabindu,K. Pranitha Kumari**, (2022, Sep.), Software Fault Prediction Using Machine Learning Algorithms,International Journal of Computer Engineering In Research Trends (IJCERT),ISSN:2349-7084 ,Vol.9, Issue 9 (Online), 170-174. Available: `https://ijcert.org/ems/ijcert_papers/V9I901.pdf`

[7] **Ezgi Erturk and Ebru Akcapinar Sezer**, (2016). Iterative software fault prediction with a hybrid approach. ScienceDirect Journals & Books,Applied Soft Computing,Volume 49.(Online),1020-1033. Available: `https://www.sciencedirect.com/science/article/pii/S1568494616304197`

[8] **C. Jones and O. Bonsignour**,"The Economics of Software Quality"Addison-Wesley Professional, Reading, 2011

[9] **C. Jin**, (2011, Aug.). Software reliability prediction based on support vector regression using a hybrid genetic algorithm and simulated annealing algorith.IET Software. (Online).5(4), 398-405. Available:`https://digitallibrary.theiet.org/content/journals/10.1049/iet- sen.2010.0073`

[10] **Selvadurai, Kanmani and Uthariaraj, V. and Sankaranarayanan, V. and Thambidurai, Perumal**, (2007,May.). Object-oriented software prediction using neural networks. IEEE Trans. Information and Software Technology.49, 483-492.

[11] **Rathore, Santosh and Kumar, Sandeep**, (2016, Feb.). A Decision Tree Regression based Approach for the Number of Software Faults Prediction ACM SIGSOFT Software Engineering Notes.41, 1-6.

[12] **Mu, R. and Zeng, X.**, (2019, Apr). A review of deep learning research. KSII Transactions on Internet and Information Systems.13, 1738-1764.

[13] **S. Peng, H. Jiang, H. Wang, H. Alwageed, and Y.-D. Yao** , (2017) Modulation classification using convolutional neural network based deep learning model, *26th Wireless Opt. Commun. Conf. (WOCC),* , Newark, NJ, USA, 7-8.

[14] **Y. LeCun and Y. H. Bengio And Hinton**,(2015"Deep learning," Nature.vol.521,no.7553,436-444.

[15] **P. S. Soniya and L. Singh**, 'A review on advances in deep learning, *. IEEE Workshop Comput. Intell., Appl. Future Directions (WCI)*,Kanpur, India 14-17, Dec 2015

[16] **C. Zhang, P. Patras, and H. Haddadi**, "Deep learning in mobile and wireless networking: A survey, *. IEEE Commun. Surveys Tuts.*vol. 21, no. 3, 2224–2287, 3rd Quart, 2019.

[17] **Fenton, Norman and Pfleeger, Shari Lawrence**, Software metrics (2nd ed.): a rigorous and practical approach, PWS Publishing Co., USA, 1997

[18] **McCabe T**, (1976). A Complexity Measure IEEE Transactions Software Engineering, vol. 2, no. 4, 308-320.

[19] **Menzies T., Greenwald J., and Frank A**, (2007). "Data Mining Static Code Attributes to Learn Defect Predictors. IEEE Transactions on Software Engineering, vol. 33, no. 1, pp. 2-13.

[20] **Metric Data Program MDP**, http://mdp.ivv.nasa.gov, Last Visited 2014.

[21] **Ian H. Witten, Eibe Frank, and Mark A. Hall**, Data Mining: Practical Machine Learning Tools and Techniques, Third Edition (The Morgan Kaufmann Series in Data Management Systems). Morgan Kaufmann, 2011.

[22] **Jun Han and Claudio Moraga** (1995), The Influence of the Sigmoid Function Parameters on the Speed of Backpropagation Learning [Online]. Available: `https://api.semanticscholar.org/CorpusID:2828079`

[23] **Olgac, A and Karlik, Bekir**, (2011, Feb.). Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks,*International Journal of Artificial Intelligence And Expert Systems*.Vol 1, 112-122.

[24] **F. Farhadi**, Learning activation functions in deep neural networks, M.S. thesis, University of Montreal, Montreal, QC, Canada, Canada, 2017.

[25] **Twala, Bhekisipho**, (2011, Jul.). Predicting Software Faults in Large Space Systems using Machine Learning Techniques. Defence Science Journal (Online).61(4), 306-316. Available: `https://publications.drdo.gov.in/ojs/index.php/dsj/article/view/1088`

LIST OF PUBLICATIONS