Journal of Theoretical and Applied Information Technology
30th June 2024. Vol.102. No. 12

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

DETECTIING REDUNDANT TEST CASES USING DEEP LEARNING

RAGHAD J, DARABSEH[1], AHMAD A, SAIFAN[2]

[1]Information System Department, Yarmouk University, Irbid, Jordan

[2]Information System Department, Yarmouk University, Irbid, Jordan

E-mail: [1]raghaddarabseh@gmail.com, [2]ahmads@yu.edu.jo

ABSTRACT

Software testing encompasses the examination of various data scenarios to assess output and observe software behaviour. However, comprehensive testing of all software cases poses challenges due to its intricate and complex nature. This paper is dedicated to the identification of redundant test cases through the application of deep learning techniques.

Four distinct deep learning algorithms—Convolutional Neural Network (CNN), Deep Belief Network (DBN), Deep Neural Network (DNN), and Long-Term Memory (LSTM)—were employed in this study. These algorithms were applied to three datasets: Common Utils for Rapied, JSOUP, and Junit. The outcomes affirm the effectiveness of deep learning algorithms in pinpointing redundant test cases. The results demonstrated that the deep neural network (DNN) is able to detect repeated test cases, which ultimately leads to fewer test cases. Compared with other algorithms of deep learning algorithms, it was found that the deep neural network (DNN) is able to cover the test cases, and it has reached a relatively high accuracy, with a result of 82.66%

Keywords: *Test Case Reduction, Redundant Test Cases, Deep Learning, Deep Neural Network, Deep Belief Network, Convolutional Neural Network, Long-Term Memory.*

1. INTRODUCTION

Software testing is considered one of the priorities for developers and programmers in companies; it aims to detect faults and defects in order to treat them. It is difficult to determine whether the software works in all circumstances and cases, as it is necessary to identify the cases for testing to show the conditions in which the software works and the conditions in which it does not work to treat them later [1].

Software testing includes defining specific test cases and applying them in different environments based on certain rules, as they are selected so that they can cover the largest number of test cases (Test Coverage), ensuring that they cover all system requirements, and ensuring the test cases are not repeated [2]. A system must be tested for errors during system testing by testing all cases in the system to ensure its correctness. However, testing all cases is time-consuming, costly, and complicated [3], so it is better to build a system with a small and effective number of test cases and not rely on random tests [4].

Redundant test cases negatively affect the testing process of the software, as when testing the test cases, they are checked again as if they were being tested for the first time, thus leading to serious consequences in terms of time and cost in the testing [5].

Regression testing is an important topic in test case reduction. It is a type of software testing that aims to ensure that modifications to the program have not negatively affected its functionality; such modifications can be adding a new feature, fixing bugs, or even fixing performance problems. After this kind of modification, we should test the modified system in the test cases fully or partially to ensure that the new modifications work well [6].

In the past few years, there has been an interest in deep learning research and its applications. [7] Noted that deep learning is

considered one of the fields of machine learning, as the computer is learning to do business and solve problems, learning from experience and without prior training. [8] Proved in their study that using different types of deep learning algorithms are very efficient in reducing the number of test cases up to 4.6%.

There are several machine learning algorithms were applied in reducing test cases in the literature including: including K-nearest Neighbors (KNN) [9], Naïve based [10], K-means [11, 38], and dynamic domain reduction (DDR) [12]. However, in this paper, we used deep learning algorithms to detect whether a test case is redundant or not. This choice was made due to the widespread use of deep learning in various fields and the lack of studies on how deep learning algorithms can detect and then reduce the number of test cases. The main contribution of this paper is to detect redundant test cases using deep learning algorithms. In addition, to evaluate the results that were reached based on accuracy, performance time, and test coverage.

Based on the research purpose, and after finishing this research, we will be able to answer the following two questions:

- How much it is effective of using deep learning algorithms to detect and reduce the redundant test cases?
- What is the appropriate deep learning algorithm that can be used in test case reduction?

2. BACKGROUND

In this section, the background of the research will be clarified, as it is divided into two main sections: deep learning and test case reduction.

2.1 Deep Learning

Deep learning is divided into three main categories: unsupervised deep learning, supervised

In unsupervised deep learning networks (generative learning), it is done by giving the training data, but the outputs are not specific and unknown. This is done by applying a deep learning algorithm to present the outputs, where only the data are given and it calculates to provide the outputs. Hybrid deep learning resembles a mixture of unsupervised and supervised learning; it deals with an interactive environment and learns from its mistakes; it is like Facebook. In Google and YouTube, for example, if you like to read many articles from a certain site, it will show you ads from the site itself; if it finds the opposite, it will not show you anything about the site, meaning that it learns from your behavior [13].

In this paper, we discuss Convolutional Neural Network (CNN), Deep Belief Network (DBN), Deep Neural Network (DNN), and Long-Term Memory (LSTM). These algorithms were chosen due to the availability of more references and studies for them than other algorithms.

Table 1: Differences Between Deep Learning Algorithms

Criteria/ Algorithms	CNN	LSTM	DNN	DBN	Ref
Aims	Ability to detect features automatically thus reduces the number of trainable network parameters	Predicting, classifying, and processing based on time series data.	Data processing in complex and difficult ways, using mathematical modeling.	It aims at how to represent features, so that it solves many tasks simultaneously based on the representation of common features.	[16] [17] [18]
Number of Layer	Three layers (convolutional layers, pooling layers, and fully connected layers)	Three layer (gate) Input gate, output gate, and forget gate.	Input layer, several hidden layers and output layers).	Input layer, several hidden-layer neurons, and output layer.	[19] [20] [21]
Examples of applications	1. image recognition 2. Extended for NLP, speech processing	Anomaly detection in network traffic	Real-Time Semantic Segmentation	speech popularity problems	[22] [23] [24] [25] [26]

There are many differences between the algorithms. Table 1 shows some differences between deep learning algorithms based on different criteria:

2.2 Test case reduction

During the testing of the system, it is difficult to conduct testing of all test cases since adding a new requirement to the system requires a new system test, and to maintain the efficiency and quality of the system, it is necessary to check the system. The system is usually tested using regression testing.

The regression test is a type of testing for the system, so that the system is tested after the

modification, as it is confirmed that changing the code on the program does not affect the functionality of the system but rather works well after making the required adjustments and modifications, to correct existing faults and errors, or to change one of the features in the current system, as it is considered a criterion of confidence for developers because it makes sure that there are no unexpected side effects after making the modification [10, 14].

The authors in [15] Pointed out that there are many techniques of regression test case reduction that are classified based on coverage, requirement, genetic algorithms, slicing, hybrid algorithms, fuzzy logic, greedy algorithms, and clustering. In general, the testing process helps in examining the appropriate software, but it is difficult to test all cases of the system due to its large size. It needs a lot of money and time. It is necessary to reduce the number of test cases to the minimum so that it covers all defects in the least amount of time.

3. LITERATURE REVIEWS

The following section provides a summary of previous studies, which is divided into sections, the deep learning and test case reduction.

3.1 Deep Learning

Deep learning has been used in many smart technologies and applications, such as self-driving cars, facial recognition systems, image recognition, NLP processing, and speech recognition. Here we focused on applications processed using deep learning algorithms.

It has been noted by Li [27] that one of the applications uses deep learning for natural language processing. The study shows that the performance is significantly superior to traditional techniques, and deep learning relies on a strong mechanism for classifying the natural language, which reflects positively on the results. [28] Noted that another application includes image processing in the medical field and biometrics. The study turns out that the use of deep learning in medical image processing has improved more than before, as it has achieved high accuracy in image processing results and reduced the error rate from 10% to 20%.

The authors in [29] Conducted a study aimed at predicting defects in projects based on static metrics using the Deep Belief Network (DBN) algorithm and transfer learning. The methodology of the study relies on three steps: first, data collection and processing; second, building a model to improve the prediction of errors in projects based on DBN and transfer learning; and third, building a defect prediction model. They collected 13 datasets from 10 open-source projects written in the Java.

Programming language. The data set contains 20 measures. The methodology of the study uses the matrix as an input feature, such as WMC, DIT, NOC, CBO, etc. After conducting the experiment, it relied on F-measure measurement in the confusion matrix. After that, the proposed model was evaluated based on three settings, which are DBN_Only, T_DBN, and T_DBN [SMOTE]. Based on F-measures, DBN_Only is 4.9%, T_DBN is 3.6%, and T_DBN [SMOTE] is 5.1%. Therefore, the proposed model with three settings is better than the model of TCA/TCA+ techniques.

Another study conducted in [30] indicated the detection of the incidence of skin cancer using deep learning algorithms. Detecting skin cancer depends on the image that is taken. To extract the features in the image and distinguish whether there is skin cancer or not, this study used artificial neural networks (ANN), convolutional neural networks (CNN), Kohonen self-organizing neural networks (KNN), and the Radial Basis Function Network (RBFN) for image classification in the detection of melanoma. The study showed that each of the algorithms has advantages and disadvantages for detecting skin cancer, but the CNN algorithm got the best results in classifying the image.

The study in [31] aims to use deep learning algorithms to detect malware in order to determine the most effective algorithms for malware detection, the study used three deep learning algorithms: Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), and Deep Neural Network (DNN). The results of the study conclude the superiority of deep learning algorithms for malware detection, and CNN algorithm was the best for detection, followed by LTSM and finally DNN; the accuracy rate reached to 96% for all algorithm, the Recall rate reached to 97%, and Precision rate reached to 97%.

3.2 Test Case Reduction

There are several techniques that help detect repeat test cases; this section reviews some studies on redundant test case detection.

The authors in [32] conducted a study to use static techniques in reducing test cases. ABB was used to detect redundancy test cases as it provides information on whether a test case can be deleted or combined with other test cases. An

experiment was conducted on a group of test cases that reached 6246 procedures, where 4218 test cases and 1637 test units were found. Three measures were used to evaluate the similarity in the test cases of the proposed ABB. It was concluded that 7%–23% of the test cases are considered redundant, either to be deleted or combined with other test cases.

Later, a comparison was done between the Tester-Assisted method and the traditional method for detecting the redundancy of test cases; it found that the traditional method was less efficient and effective than the Tester-Assisted method.

Another think about conducted by [33] used the hybrid Particle Swarm Optimization (PSO) to select which test cases require execution, Java was used to implement the suggested algorithm, it gives an input, and this method returns the number of predicted test cases, and the number of particles needed for the test cases. The results of the study indicated that the PSO algorithm provides the best and most effective result to reduce the number of test cases, it reached a rate equivalent to 50%.

According to [1] the purpose of this study was to identify and refactor the source code to reduce the redundancy in the development of software test cases by using code on Android applications, this method helps to make changes in the source code by improving the quality and reducing the redundancy in the building of the test cases by making it easier to read the source code. In this study, the application was Alogcat (Android) and DART (Eclipse) was implemented as an add-on tool that helps reduce the number of test cases. According to the study, DART reduces the number of test cases by 28% as it can cover 5 of the test cases.

Measurement of the redundancy Score in test cases, since the test cases are implemented using the same instructions with the same line of code and require high time and cost to implement, a test case matching approach was proposed by [34], the study was conducted on a test set of duplicate test cases, it was performed on two Java programs (student grades and quadratic function). The test case matching approach measured the redundancy score on both test groups, and the detection of the redundant test cases was 37% and 67% respectively.

The authors in [35] Indicated in their study that aimed to mutation-based fault localization (MBFL) using Contribution-based Test Case Reduction Strategy (CBTCR), due to the high cost when implementing a large mutation, (CBTCR) value of each test case is measured and it selected based on the value. Depending on Top-N and MAP metrics, CBTCR was is best than MBFL such as FTMES and IETCR. In addition CBTCR reduce cost by 85.43%.

4. RESEARCH METHODOLOGY

This section provides an overflow of the phases used to implement our approach. It is used to detect whether the test case is a redundant or not and then reduce test cases using deep learning algorithms. Below, each phase will be explained in details. Figure 1 represents the approach followed to detect the redundancy of test cases using deep learning algorithms.

The details of this process will be discussed in the next section.

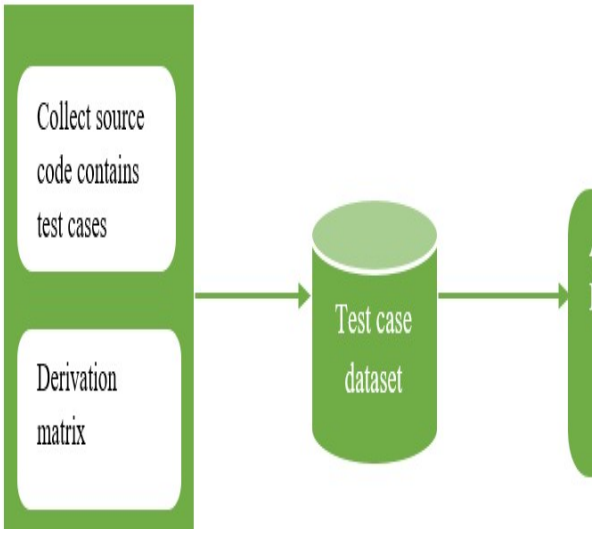

Figure 1: Research Methodology.

4.1 Research Plan

Phase1: Collect the source code with test case and derivation metrics

This phase consists of several parts: collecting the Java open-source codes that have test cases, identify the metrics of test cases that represent the features, and the extract the values of the metrics from the test cases. The approach starts with collecting a set of under-tested open-source code written in Java programming language. After that, the codes were run on the Eclipse tool, which is an integrated development environment (IDE), to

make sure they were working correctly. We selected applications that already have test cases. Moreover, Junit is used to automatically generate a set of test cases for any given input class.

This process used to increase the number of test cases and to make sure that we have redundant test cases. After that, we identified the metrics that represents the feature for test cases.

Table 2 contains a description of the metrics used to build the dataset [36].

Table 2: Dataset Metrics

Matrix Name	Matrix Description
Test Case	Number of classes and method names
Class Label	Retrieve the status of test cases whether it is redundancy or not
Coverage	The Percentage of test cases covered
Covered Instruction	Covered The number of code Instruction executed and covered
Missed Instructions	The number of code Instruction that not executed and covered
Total Instructions	The number of codes executed and non- executed
BRANCH	The number of Branch statement in each method that Coverage
CALL	The number of total call expression for each method
CALLED	The number of methods that called Whether it is override or overload.
CALLEDt	The number of methods that called Whether it is override or overload by time.
Executable statements (EXEC)	The number of Executable statements for each method
Line Of Code (LOC)	The number of Lines of Code for each method
CAUGHT	The total number of exception class in each method are caught it.
Halsted Difficulty (D)	The level of difficulty for understanding a method
Halsted Effort (E)	The level of effort for understanding a method
Halsted Length (N)	The number of operator and operands in each method
Number of Type parameter(NTP)	The total number of types of parameters for every method
STAT	The total statement for every method
THROWS	The number of exception class it declares in "throws" clause
Halsted Volume (V)	Size of the method, it is defined as N*log(n).
v(g)	Cyclomatic complexity, it is the number of operations the method will perform, and the time spent

The features have been optimized in the three datasets, and they deal with common features in the dataset as their number was optimized to 20 features, including: coverage, covered instruction, missed instructions, total instructions, batch, call, called, calledt, executable statements (EXEC), line of code (LOC), caution, halsted difficulty (D), halsted effort (E), halsted length (N), number of parameters (NP), number of type parameters (NTP), stats, throws, halsted volume (V), and v(g).

Phase2: Build the dataset

Data preparation is one of the basic operations of preparing data and preparing it for work, as it includes data processing, cleansing, validation, transformation and making it suitable for use and analysis of results. The data were collected from three datasets (Common Utils For Rapied dataset, JSOUP dataset, and junit dataset), each dataset extracted from different Open-Sources.

Therefore, it has different characteristics and domain from the other, as each dataset has a set of different metrics from the other so we selected 20 features because it the common in three datasets. After selecting the metrics, the dataset had missing values so it has been removed, and the necessary values that have the effect of reducing test cases were calculated using the Understand tool edition 6.1 to calculate the value of metrics to use it.

To build the datasets, we used the understand tool Edition 6.1 to measure the metrics for each test case in the dataset. This tool is an integrated development environment (IDE). It is able to analyze and understand the source code by calculating the selected metrics (Understand, 2024). In this paper deals with three datasets that have different features and instances.

The first dataset is called Common Utils the Rapied dataset which contains a low number of test cases; the number of test cases is 437. The scorned dataset is JSOUP dataset which contains 665 test cases. A third dataset has been added, which is the junit dataset. It contains a high number of test cases, 1269 test cases. Thus, the test cases in the datasets were applied using deep learning algorithms to detect the redundancy of test cases.
The three collected datasets are different in terms of the number of lines in code for each test case, the number of coverages, the number of covered instructions, the number of total instructions, and the number of calls. Table 3 contains information about these datasets.

Table 3: Dataset Description

Matrix Name	Common Utils For Rapied	JSOUP	Junit
Number of Test Case	437	665	1269
Number of coverage	73	44	124
Number of covered Instruction	115	10	29
Number of missed instruction	48	31	48
Number of total instruction	129	50	153
Number of call	61	4	359

Phase 3: Apply deep learning algorithms

Through the use of deep learning algorithms, the original test cases were reduced to the smallest set of test cases that covered the source code. In this phase, four deep learning algorithms were applied, including the convolutional neural network (CNN), deep belief network (DBN), deep neural network (DNN), and long short-term memory (LSTM).

While previewing test cases datasets, there are several redundant test cases in it. For the purpose of reducing time spent, cost, and effort, the number of test cases should be reduced.

The CNN, DBN, DNN, and LSTM algorithms are applied to the three datasets. For example, apply the CNN algorithm to the Common Utils for Rapied dataset (D1), the JSOUP dataset (D2), the junit dataset (D3), and so on for the rest of the algorithms. These algorithms are used to detect and identify which test case in the datasets is redundant with another test case. In this paper, these algorithms will identify redundant test cases based on a set of features (metrics) of test cases in the source code.

Each algorithm has different settings, but in general, what distinguishes deep learning is the hidden layer, as it performs mathematical calculations and transactions within each algorithm, so it is necessary to choose the number of hidden layers carefully. The number of hidden layers is associated with the percentage of accuracy, as many experiments were conducted to carefully choose the number of hidden layers to increase the percentage of accuracy. The experiment was carried out on a

number of hidden layers at 1, 3, 5, 8, based on the highest accuracy we obtained when the number of hidden layers equals three.

When applying deep learning algorithms, it is necessary to define epochs, which represent the hyper parameter that describes how many times the algorithm must repeat a training dataset (Papadomanolaki et al., 2016). After one epoch, the internal model parameters have been updated for each sample in the training dataset. Each epoch is made up of one or more batches (Papadomanolaki et al., 2016). In this paper, we applied the deep learning algorithm with 3000 epochs, which means the model will be updated 3000 times in one epoch, and the batch size is 300 batches at a time. The primary reason we use the sigmoid function is that it exists between 0 and 1. Therefore, it's specifically used for models in which we should predict the opportunity as an output. Since the opportunity for something exists simplest between the varieties of 0 and 1, sigmoid is the proper choice.

The activation function is differentiable, that means the slope will be discovered of the sigmoid curve at any point; it is monotonic, but the function's by-product is not. The logistic sigmoid- for the activation function- can motivate a neural network to get caught up in the training time. The softmax function is a more generalized logistic activation function that is used for multiclass classification. Finally, the sigmoid function curve seems like an S-shape.

This phase is very important because the results obtained will be used in the next phase. It shows the ability of deep learning algorithms to detect redundancy in test cases. Thus, after applying the deep learning algorithms to the three datasets, the number of reduced test cases per dataset is shown. When applying DL algorithms, we must divide the datasets into training and testing data. The common Utils for Rapied dataset uses 68% for training and 32% for testing (297 instances for training and 140 instances for testing). The JSOUP dataset uses 45% for testing and 55% for training (300 instances for testing and 365 instances for training). junit dataset, using 23% for testing and 77% for training (292 instances for testing and 977 instances for training).

Phase 4: Reduce the test case dataset

Building the proposed approach model in the Knime tool is the most important step in reducing the number of tests because all subsequent operations on the test case depend on the model. Once the model is completed the efficiency of the test cases generated by the model increases.

The proposed approach model is generated to create reduced test cases. The difficulty of creating test cases depends on the size of the program, as it is difficult to create test cases for complex systems such as neural networks without automation. If the dataset is very large, this usually means that the number of test cases is very large, or even infinite. However, as the quality of the system improves, it needs to identify good test cases to help the evaluators find as many defects as possible in the system at a suitable cost [37].

This phase shows the number of test cases after applying deep learning algorithms. We show the datasets after removing the redundant test cases, as a dataset will be obtained for each algorithm. For example, after applying the CNN algorithm to the three datasets, new datasets will be obtained, which are the reduced D1 dataset using CNN, the reduced D2 dataset using CNN, and the reduced D3 dataset using CNN. Thus, these algorithms are ready to be compared to determine which algorithm is more effective in detecting redundant test cases.

Phase 5: Evaluate the test cases

In the evaluation phase, the results obtained are collected to ensure that they have achieved the objectives that were identified or not, and evaluation is considered essential to ensuring the effectiveness and efficiency of the proposed approach.

In this phase, the reduced datasets using deep learning algorithms are evaluated to determine which algorithm is best suited for detecting redundant test cases. There are several techniques that are used to evaluate the performance of results, such as accuracy, error rate, complexity, time, and test coverage.

Performance measures techniques:

Accuracy: This is the ratio of the number of accurate predictions out of all the predictions; it is given as a percentage. In general, the number of samples to be matched is divided by the number of all samples; the higher the number, the better the classifier. In this paper, the accuracy values are posted from 0 to 1, and then they are converted into a percentage value from 1 to 100.

$$Accuracy = \frac{total\ number\ of\ correct\ Predection}{total\ number\ of\ all\ Predection} * 100 \quad (1)$$

Complexity time: It expresses the amount of time a system, code, or algorithm takes to run. That is the time required to process and get results within a given number of inputs. There are different types of complexity time used, such as linear time, constant time, and polynomial time.

Test coverage: this is a specific type of systematic and sequential test in which each line of code is evaluated individually. As a type of

software testing, test coverage is done in the category of technical testing methods rather than as part of an overall strategy of code.

Phase 6: Analysis the result

In the last stage, the accuracy, complexity, time, and coverage for each deep learning algorithm in detecting the redundancy of test cases will be calculated.

First of all, the deep learning algorithms that were used will be compared, including CNN, DBN, DNN, and LSTM, to determine which algorithms are best in terms of accuracy, so the higher the percentage, the better the results.

Based on terms of complexity and time, depend on the low percentage. That is, the lower percentage for executing the code is the better algorithm than other algorithms. Based on terms of test coverage, depend on the high percentage. That is, with the highest percentage of covered test cases, it is the better algorithm than other algorithms.

4.2 Discussion the result

We applied a deep learning algorithm to the KNIME tool to detect the redundancy of test cases. We evaluated the algorithms based on three techniques: accuracy, test coverage, and performance time, which will be summarized in detail below.

4.2.1: Accuracy

Accuracy is expressed as the number of times the classifier's answer was correct. The accuracy scale was used in our paper because it is one of the most widespread criteria, and most studies depend on the accuracy criterion to measure the results that have been reached [38].

Table 4 summarizes the accuracy rate of each algorithm in the three datasets for redundancy detection of test cases.

As noted in Table 4, shown is the accuracy statistic for CNN, DBN, DNN, and LSTM on three data sets (Common Utils for Rapied, JSOUP, and Junit).

Based on the Common Utils for Rapid dataset, the most accurate is 83.57% in the DNN algorithm, with 117 correct classifications out of 140 instances, while the lowest percentage of accuracy is 78.57% based on the LSTM algorithms.

Table 4: Dataset description

Data set	Algorithm	Accuracy	# of classify	Correct	Wrong
Common Utils For Rapied	CNN	79.286%	140	111	29
	DBN	81.42%	140	114	26
	DNN	**83.57%**	**140**	**117**	**23**
	LSTM	78.57%	140	110	30
JSOUP	**CNN**	**83.33%**	**300**	**250**	**50**
	DBN	82.66%	300	248	52
	DNN	**83.33%**	**300**	**250**	**50**
	LSTM	76%	300	228	72
Junit	**CNN**	**80.47%**	**292**	**235**	**57**
	DBN	78.08%	292	228	64
	DNN	76.71%	292	224	68
	LSTM	69.86%	292	204	88

As for the JSOUP dataset, the CNN and DNN algorithms each had the same value of accuracy, which equaled 83.33%; while the lowest percentage of accuracy is 76% based on LSTM algorithms.

In the Junit dataset, the accuracy rate reached 80.47% in the CNN algorithm, where the number of correct classifieds reached 192 out of 292, while the lowest percentage of accuracy was 69.86% based on the LSTM algorithms.

4.2.2: Test Coverage

Test coverage is one of the methods used to test the source code of software. We used the test coverage to find out whether removing redundant test cases affects the coverage rate of the source code or not. By using Eclipse for Java code coverage, we depended on the EclEmma feature.

Table 5: Summary of test Coverage

Dataset / Algorithm	Test coverage before reduction						Test coverage after reduction					
	Number of instances				Percentage	Time in second	Number of instances				Percentage	Time in Second
	D1	D2	D3	Total			D1	D2	D3	Total		
CNN	437	665	1269	2371	37.5%	3.63s	73	87	262	442	33.7%	3.26s
DBN	437	665	1269	2371	1.5%	3.851s	73	87	262	442	0.9%	3.601s
DNN	437	665	1269	2371	22.6%	2.0s	73	87	262	442	22.6%	2.0s
LSTM	437	665	1269	2371	66.3%	2.3s	73	87	262	442	66.3%	2.3s

As shown in Table 5, the percentage of the coverage for CNN was 37.5% before test case reduction, but after the test case reduction, it was 33.7%. It took 3.26 seconds less than before coverage, so it covered 442 test cases out of the 2371 test cases. As for the DBN algorithm, the percentage test coverage before reduction is 1.5%, but after covering 442 test cases out of 2371, it got a 0.9% percentage.

After test coverage, it takes 3.601 seconds to perform. We also note that the DNN and LSTM algorithms kept the percentage of test coverage before and after reduction unchanged since the DNN is 22.6% and the LSTM is 66.3%, as the time for DNN before and after reduction was 2.0 seconds and for LSTM before and after reduction was 2.3 seconds.

1.2. Performance time

The performance time it expresses for the time taken to execute the deep learning algorithms shows the execution time for each block in all datasets (Common Utils for Rapied, JSOUP, and Junit). Tables 6 summarize the results for performance time for each algorithm in the Knime tool.

Table 6: Performance time

Algorithm	Time
CNN	242 milliseconds
DBN	636 milliseconds
DNN	169 milliseconds
LSTM	194 milliseconds

Table 6 shows the performance time for deep learning algorithms; the results are distinct amongst different algorithms. The lowest percentage for performance time was DNN, as it reached a value equivalent to 169 milliseconds. It was followed by the LSTM algorithms, which took 194 milliseconds. After that, the CNN algorithm took 242 milliseconds, but the most time-consuming algorithm is DBN, which is equivalent to 636 milliseconds.

2. Threat To Validity

The paper presents a case study involving three deep learning algorithms applied on three test cases dataset. The purpose of the case study is to affirm the effectiveness of deep learning algorithms in pinpointing redundant test cases. Both the deep learning algorithms and the selected test cases dataset bear some properties which may affect the validity of the results and the usability of the approach. We only choose three deep learning algorithms which are not enough for drawing general conclusion about the ability of deep

learning algorithms to reduce the number of test cases. The small size of datasets utilized in this study poses a significant threat to validity, potentially compromising the generalizability and reliability of the findings. With limited data points, the risk of sampling bias increases, leading to skewed results that may not accurately represent the broader population or phenomena under investigation.

3. CONCLUSION

The software testing process plays a crucial role in providing transparent insights into the quality of software and the potential risks of failure for future users. This is significant because the early identification and resolution of software errors contribute to saving time and money during the testing phase.

Conducting exhaustive software testing is often challenging, leading to the selection of a subset of test cases to ensure comprehensive coverage without redundancy. Detecting redundant test cases poses its own challenges, and various techniques, tools, and algorithms, including those based on deep learning are employed for this purpose.

This paper focuses on leveraging deep learning algorithms to identify redundancy in test cases. Four deep learning algorithms—Convolutional Neural Network (CNN), Deep Neural Network (DNN), Deep Belief Network (DBN), and Long Short-Term Memory (LSTM)—have been chosen due to the abundance of references and studies supporting their effectiveness.

The application of these deep learning algorithms involves three datasets: Common Utils for Rapied, JSOUP, and Junit. The datasets are meticulously prepared, with class labels classified manually to facilitate integration with the KNIME tool.

The data is divided into testing and training sets, and each algorithm is applied to the three datasets using the KNIME tool. The comparison of results is based on three criteria: accuracy, test coverage, and performance time.

The paper yielded notable results, with accuracy being a key focus. The DNN algorithm proved most accurate for the Common Utils for Rapied dataset, achieving 83.57%. For the JSOUP dataset, both CNN and DNN algorithms demonstrated an identical accuracy of 83.33%. In the case of the Junit dataset, the CNN algorithm achieved an accuracy rate of 80.47%.

Examining test coverage revealed interesting findings. The CNN algorithm exhibited test coverage of 37.5% before test reduction, which decreased to 33.7% after reduction. The DBN algorithm, on the other hand, started with 1.5% test coverage before reduction and reached 0.9% after reduction, taking 3.601 seconds to complete. Meanwhile, the DNN and LSTM algorithms maintained constant test coverage percentages before and after reduction—22.6% for DNN and 66.3% for LSTM.

Considering performance time, the DNN algorithm emerged as the most efficient, boasting the lowest percentage at 169 milliseconds.

4. REFERENCES

[1] Ibrahim, R., Ahmed, M., Nayak, R., & Jamel, S. "Reducing redundancy of test cases generation using code smell detection and refactoring". *Journal of King Saud University-Computer and Information Sciences*, 32(3), 2020. 367-374.

[2] Bierig, R., Brown, S., Galván, E., & Timoney, J. "Essentials of Software Testing". *Cambridge University Press*. 2021.

[3] Normann, F. "Test Case Selection Based on Code Changes". *Teknisk- naturvetenskaplig fakultet UTH-enheten*. 2019.

[4] Zhang, C., Groce, A., & Alipour, M. A. "Using test case reduction and prioritization to improve symbolic execution". *In Proceedings of the 2014 International Symposium on Software Testing and Analysis.* 2014, pp. 160-170. ACM.

[5] Alsukhni, A. A. Saifan, and H. Alawneh. "A new data mining-based framework to test case prioritization using software defect prediction". *International Journal of Open Source Software and Processes (IJOSSP)*, no. 1, 2017. 21-41.

[6] Rosero, R. H., Gómez, O. S., & Rodríguez, G. "Regression testing of database applications under an incremental software development setting". *IEEE,* 2017, pp. 18419-18428.

[7] Yuan, X., He, P., Zhu, Q., & Li, X. "Adversarial examples: Attacks and defenses for deep learning". *IEEE transactions on neural networks and learning systems*. 2019.

[8] Zhou, J., Li, F., Dong, J., Zhang, H., & Hao, D. "Cost-Effective Testing of a Deep Learning Model through Input Reduction". *In 2020 IEEE 31st International Symposium on Software Reliability* Engineering (ISSRE), 2020, pp. 289-300.

[9] Meçe, E. K., Paci, H., & Binjaku, K. "The Application Of Machine Learning In Test Case Prioritization-A Review". *European Journal of*

Electrical Engineering and Computer Science, 4(1). 2020.

[10] Saifan, A. A. "Test Case Reduction Using Data Mining Classifier Techniques". *JSW*, 11(7), 2016, pp.656-663.

[11] Pandey, A., & Malviya, A. K. "Enhancing test case reduction by k-means algorithm and elbow method". *International Journal of Computer Sciences and Engineering*, 6(6), pp. 299-303, 2018.

[12] Chaurasia, V., Chauhan, Y., & Thirunavukkarasu, K. "A survey on test case reduction techniques". *International Journal of Science and Research (IJSR*), 2014.

[13] Deng, L. A. "Tutorial survey of architectures, algorithms, and applications for deep learning". *APSIPA Transactions on Signal and Information Processing*, 2014.

[14] Kapfhammer, G. M. "Regression testing". *In Encyclopedia of Software Engineering Three-Volume Set (Print), Auerbach Publications,* 2010, pp. 893-915.

[15] Alian, M., Suleiman, D., & Shaout, A. "Test case reduction techniques-survey". *International Journal of Advanced Computer Science & Applications*, 1(7), 2016, pp. 264-275.

[16] Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O. & Farhan, L. "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions". *Journal of big Data*, 8(1), 2021, pp. 1-74.

[17] Staudemeyer, R. C., & Morris, E. R. "Understanding LSTM--a tutorial into Long Short-Term Memory Recurrent Neural Networks". *arXiv preprint arXiv:1909.09586,* 2019.

[18] Hailesilassie, T. "Rule extraction algorithm for deep neural networks: A review". *arXiv preprint arXiv:1610.05267*, 2016.

[19] Johansson, A., & Sandberg, O. "A Comparative Study of Deep-Learning Approaches for Activity Recognition Using Sensor Data in Smart Office Environments", 2018.

[20] Jiang, J., Zhang, J., Zhang, L., Ran, X., Jiang, J., & Wu, Y. "DBN Structure Design Algorithm for Different Datasets Based on Information Entropy and Reconstruction Error". *Entropy*, 20(12), 2018, pp. 927.

[21] Salman, A. G., Heryadi, Y., Abdurahman, E., & Suparta, W. "Single layer & multi-layer long short-term memory (LSTM) model with intermediate variables for weather forecasting". *Procedia Computer Science*, 135, 2018, pp. 89-98.

[22] Lee, S., Kim, H., Lieu, Q. X., & Lee, J. "CNN-based image recognition for topology optimization". *Knowledge-Based Systems*, 198, 2020, pp. 105887.

[23] Krizhevsky, A., Sutskever, I., & Hinton, G. E. "Imagenet classification with deep convolutional neural networks". *Advances in neural information processing systems*, 25, 2012, pp.1097-1105.

[24] Malhotra, P., Vig, L., Shroff, G., & Agarwal, P."Long short-term memory networks for anomaly detection in time series". *In Proceedings*, Vol. 89, 2015, pp. 89-94.

[25] Paszke, A., Chaurasia, A., Kim, S., & Culurciello, E. "Enet: A deep neural network architecture for real-time semantic segmentation". *arXiv preprint arXiv:1606.02147*. 2016

[26] Prasetio, M. D., Hayashida, T., Nishizaki, I., & Sekizaki, S. "Deep belief network optimization in speech recognition". *In 2017 International Conference on Sustainable Information Engineering and Technology (SIET).* IEEE, 2017, pp. 138-143.

[27] Li, H. "Deep learning for natural language processing: advantages and challenges". *National Science Review*. 2017.

[28] Vargas, R., Mosavi, A., & Ruiz, R. "Deep learning: a review". *Advances in Intelligent Systems and Computing*. 2017

[29] Saifan, A. A., & Al Smadi, N. "Source code-based defect prediction using deep learning and transfer learning". *Intelligent Data Analysis*. 23(6), 2019, pp. 1243-1269.

[30] Dildar, M., Akram, S., Irfan, M., Khan, H. U., Ramzan, M., Mahmood, A. R., & Mahnashi, M. H. "Skin Cancer Detection: A Review Using Deep Learning Techniques". *International journal of environmental research and public health*, 18(10), 2021, pp.5479.

[31] Altaiy, M., YILDIZ, İ., & Bahadır, U. Ç. A. N. "Malware detection using deep learning algorithms". *AURUM Journal of Engineering Systems and Architecture*, 7(1), 2023, pp.11-26.

[32] Li, N., Francis, P., & Robinson, B. "Static Detection of Redundant Test Cases: An Initial Study". *In 19th International Symposium on Software Reliability Engineering*. IEEE, 071-9458/08, 2008.

[33] Nagar, R., Kumar, A., Kumar, S., & Baghel, A. S. "Implementing test case selection and

reduction techniques using meta-heuristics". *In 2014 5th international conference-confluence the next generation information technology summit* (*Confluence*). IEEE, 2014. pp. 837-842.

[34] Saputra, M. C., Katayama, T., Kita, Y., Yamaba, H., Aburada, K., & Okazaki, N. "Measuring Redundancy Score for Test Suite Evaluation by Using Test Cases Matching Approach". *Journal of Robotics, Networking and Artificial Life*. 2021.

[35] Wang, H., Yang, K., Zhao, X., Cui, Y., & Wang, W. "Contribution-based Test Case Reduction Strategy for Mutation-based Fault Localization". DOI reference number: 10.18293/SEKE2023-180. 2023.

[36] Vuori, M. "About metrics and reporting in model-based robot assisted functional testing. Project report". 2014, pp. 13.

[37] Saifan, A., & Dingel, J. "Model-based testing of distributed systems". *Technichal report*, 2008, pp.548.

[38] Saifan A. A., E. Alsukhni, H. Alawneh and A. Al-Sbaih, "Test case reduction using data mining technique," International Journal of Software Innovation (IJSI), vol. 4, no. 4, pp. 56–70, 2016.