Mutation Testing of Deep Learning Systems

Abdulhayyu Lawal

October 29, 2019

Abstract

Deep learning (DL) models have become increasingly popular and an important concept in many useful application domains such as self-driving systems, facial identity systems, automatic speech recognition and more others. Presence of defective deep learning model in these applications may lead to catastrophic accidents and consequences. Research efforts have been going on to test these DL systems to examine their quality and performance, one way to measure the performance can be through evaluating the quality of the test dataset by applying mutation testing (MT) technique. A mutation testing framework that is specialized for DL systems proposed by Lei et al. in their DeepMutation paper that measures the quality of test data and examines the DL model's performance on the test dataset is studied and replicated. In this master thesis, I aim at studying the usefulness of mutation testing techniques for DL models by replicating the approach of the DeepMutation paper and at analyzing and evaluating the testing technique usefulness and contribution of the paper. Furthermore, my thesis work will also cover a comprehensive study and investigation of the mutation operators impact.

1 Introduction

Deep Learning (DL) has in the past decades become a fundamental concept in many applications and in automation systems including safety applications, such as self-driving cars, image recognition, visual art processing, robotics, games and many others [12]. However, the quality of these deep learning models remain a major concern and present a big challenge more especially in safety-critical systems. After witnessing some catastrophic accidents related to DL like the recent Uber autonomous car incident in March 2018 that led to the death of a pedestrian, because of the inability of the autonomous car to accurately identify the object (Person) in time to adopt necessary action before the collision. The need for safety, high robust and accurate DL systems has raised more than ever before [7],[13].

Deep learning systems performance can be evaluated through examining their performance on a test dataset which can be by mainly measuring their accuracy on the test dataset. With an evaluated and high quality test data, the robustness and performance of the DL systems can be examined. To measure the quality of the test data and to gain confidence of the trained DL models, mutation testing (MT) technique is one of the software testing method to do so. MT is a well established fault-based testing technique for quality evaluation of test suites, which involves designing and executing of the test suites against mutated program to evaluate the ability of the test suite in detecting the injected mutants/faults in the program. Due to the architectural structure of DL systems, traditional software testing cannot be applied directly to DL systems, a mutation testing framework specialized for DL systems to measure the quality of the test data proposed by [7] is repeated and adopted to test deep learning-based software [7],[13],[12].

In this master thesis, I will replicate the work of Lei et al.[7] on DeepMutation. In their work, they proposed a mutation testing framework specialized for the DL systems using the same spirit of mutation testing for traditional software. The paper applied MT to DL models to measure the quality of test data in

order to gain performance knowledge of the DL systems. The paper proposed and designed mutation operators that introduce the potential faults into the software under test (SUT) to create a modified version of the SUT consisting of mutants. The quality of test is examine to check to what extent a test suite could detect the behavior differences of mutants and the corresponding original SUT.

In applying MT to DL systems, two major sources to inject mutants/faults can be through the training data set and the training program model. The authors designed two different groups of mutation operators; source-level mutation operators to inject potential faults into the training data and model-level mutation operators to inject faults directly into the training neural network. Original training data D and original training DL program model program P are modified through introducing mutants to create mutated version of both as D' and P'. In this way, a number of mutated DL program models $(M'_1, M'_2, ...M'_n)$ are obtained and each of the created program mutant models M'_i is executed and analyzed against the test set T in correspondence to original DL program model M. The $(M'_1, M'_2, ...M'_n)$ mutant are executed on each of the corresponding originals to filter out failed cases and those that did not pass the filter are sent back to the original D to be fixed and re-executed again. Given a test input $t \ \epsilon \ T$ that runs on $(M'_1, M'_2, ...M'_n)$ is further executed against the both passed M and M' and t detects the behavior difference of M and M'_i if the outputs of M and M' are consistent on t. If there are differences, the M' are detected and killed otherwise the mutant M' are still alive and more test needs to be carried out.

I will study and replicate the approach procedure of Lei et al.[7] paper by implementing mutation testing technique to evaluate test suites through which the quality performance deep learning can be examined. My contribution will be to replicate, analyze and evaluate their work findings and that of the mutation testing framework in general, involving the demonstration of the usefulness of the proposed testing technique and its evaluation. The test will be carried out on Deep Neural Networks (DNNs) with different structures, complexity and on

two widely used data sets, the MNIST and CIFAR-10. The analysis and evaluation will involve quantitative and qualitative analysis study. The thesis will also investigate and cover a study on the mutation operators impact i.e which mutant operators perform better or has least impact as a mutant.

2 State-of-the-Art and Related work

Since the birth of mutation testing technique, much research work on the various kinds of approach seeking to turn mutation testing into a practical method has been carried out but there has not been many attempts on how the technique can be applied to specifically test Deep learning system's quality performance [7],[9]. Mutation testing and analysis can be used for software testing at all levels, including unit, integration and specification to prove the quality performance and effectiveness of deep learning systems through assessing and improving of test suite [13] [7].

In this section, I have gathered some of the literature/research that are either relevant to mutation testing or its application to deep learning systems. Besides the papers mentioned, there are more other research papers that are read and are considered to be supportive in the implementation of the thesis but not mentioned here.

Lei et al.[7] This thesis lies solely on this paper as its work will be replicated and some additional study also added. The paper proposes a mutation testing framework to assess test cases and to measure the quality of deep learning systems. The paper is the first research to proposed and designed two sets of mutation testing operators, a set of source-level mutation operators to inject faults to the source data and another set of model-level mutation operators to introduce faults to the model program. The paper evaluates the quality of the test case by analyzing the extent to which the injected and introduced faults can be detected. The quality of the DL system is then measured based on the behavior performance of the model to the introduced faults data and the original data.

Muhammed et al.[1] This paper conducted a review of different convolution neural network architectures, tested and evaluated their performance especially in autonomous driving systems. The paper also explained in details how convolutional neural network are used and applied in autonomous vehicles application, helping to make better critical decisions.

Rene et al.[6]. Provides some suggestions on how to improve mutation testing analysis. They examined how mutants faults are differentiated from real faults in software testing.

Vincent et al.[2] Focused on mutation analysis as a way to systematically qualify and improve a set of test data for detecting faults in a program under test. They created faulty versions of programs from original programs by systematically injecting one single fault per version of the program. They measured the ability to highlight the fault injected in each mutated version (killing these mutants).

Goran et al.[8] Showed how mutation testing can be applied to test industrial applications. Their work will help in understanding the importance of mutation testing, quantifying the costs of unproductive mutants and suggesting ways to effectively handle the current challenges of efficiently and effectively applying Mutation Testing in an industrial-scale software development process.

Xiaowei et al.[10] Talked about verification and testing of deep learning systems. They verify the DL systems by global optimization approach and by layer-by-layer approach with exhaustive search.

Jia et al.[5] Talked about several Mutation Testing developments. The authors provides almost a complete survey of all publications related to mutation testing since 1970s and analyzed the testing technique's development trends.

Jingyi et al.[11] Proposed an approach to detect adversarial samples for Deep Neural Networks at runtime. The authors propsed mutation testing to teast adversarial sample which are more sensitive than normal samples by measuring their sensitivity.

Dawei et al.[3] Used mutation to test simulated program bugs and observe different types of mutants and make conclusions based on the results they found

about the impacts of mutants machine learning codes.

Lorena et al.[4] Talked about mutation operator to be introduced to program, how to define and classify accordingly to be good enough for the test case.

Weijun et al.[9] Proposed mutation operators to introduce potential faults and define test case to be execute the faults and calculated the mutation score as test adequacy.

3 Research Goal

The key contribution of this thesis is to apply, analysis and evaluate mutation testing framework on deep learning systems. The goal is to repeat the testing technique of Lei et al.[7] paper on deep learning system to evaluate test suites and assess the quality performance of the DL models. The two mutation operator groups proposed by the authors in the paper to generate mutants to be injected into both the training data and the DL model will be re-defined and similar approach to applying traditional software testing method to DL systems will be replicated and adopted. In addition to the base paper's work replication and evaluation, mutants proposed and introduced are studied and investigated to find out the impact of each on the performance of the DL models. The thesis aims specifically at:

- Replicating the work of Lei et al.[7] DeepMutation paper.
- Analyzing and evaluating mutation testing usefulness.
- Understanding and evaluating the paper's contribution i.e is their results/contributions in accordance with findings?
- Investigating more on mutants contribution. E.g, which mutant performs better or has least impact?

4 Method

4.1 Proposed Methodology

This thesis will applied mutation testing to deep learning systems based on the concept used by Lei et al.[7] DeepMutation paper. To facilitate their approach, same traditional software testing approach specialized to test the DL systems is replicated in which two different groups of mutation operators are designed; source-level mutation operators for mutating the source data and model-level mutation operators for mutating the DL model program. Test suites to find and catch the faults/mutants from each mutation operators will be generated. The proposed testing framework will be carried out on the two well known datasets; MNIST and CIFAR-10 and will be evaluated based on the mutation score and average error rate of the each dataset. There will be quantitative and qualitative evaluations; quantitative analysis involving statistical inference to evaluate MT, similar to [7] method and some additional metrics and the qualitative analysis will elaborate more on the technique's usefulness and procedure applied. Figure 2 depicted the proposed process of this thesis.

4.2 Test Subject

Dataset and DL Models

The first step in mutation testing is to prepare the test data and the training data for mutation and evaluation. As mentioned already; MNIST and CIFAR-10 dataset will be used for training the models. Deep neural networks (DNNs) model with different architectural designs will be used on each dataset and each architecture will be studied to investigate and evaluate the quality of the test data. After preparing the data for actual mutation, only relevant tests t' that passed the test set after executing the complete T test sets are used further to execute P'. Quality of test data will be ensured through a controlled experiment to be carried out.

Fault/Mutant Type	Description
Data Repetition (DR)	Duplicate training data globally and some specific type of data loacally
Label Error (LE)	Falsify result (e.g labels) of data globally and falsify specific results of data loaclly
Data Missing (DM)	Remove selected data in a global way and locally remove specific types of data
Data Shuffle (DF)	Shuffle selected training data globally and some specific types of data locally
Noise Pertubation (NP)	Add global noise to training data and as well add loacally to specific type of data
Layer Removal (LR)	Global removal of a layer
Layer Addition (LA)	Global addition of a layer
Activation Func Remov	Globally remove activation functions

Table 1: Source-level mutation operators[7]

4.3 Proposed Test Mutation Operators

The general goal of Mutation Testing is to evaluate the effectiveness of a test case that are executed on the mutants/faults deliberately introduced to slightly modify original data/program to produce mutated version. In this subsection, the two groups of mutation operators that will be used as source of introducing mutants to the data/program with each operator comprising sets of mutants are highlighted. Each mutation operator that performs a modification to the original program by introducing potential faults will be discussed and experimental evaluation will be conducted. Mutants for the training data will be introduced either globally to all data or locally to specific types of data, same pattern used by [7]. The list of mutation operators for the input training data is shown in table 1.

Due to the architectural structure of deep learning systems, traditional mutation testing cannot be applied directly to them, The same mutation testing framework procedure specialized for DL systems proposed by authors will be adopted here. The procedure includes using the same spirit of mutation testing shown in figure 1 for traditional software but with different set of mutation operators to introduce mutants to the neural network. The replicated proposed mutation operators for mutating the DL model is in table 2.

Mutation Operator	Description
Gaussian Fuzzing (GF)	Fuzz weight by Gaussian Distribution
Weight Shuffling (WS)	Shuffle selected weights of neurons
Neuron Effect Block. (NEB)	Block a neuron effect on following layers
Neuron Activation Inverse (NAI)	Invert the activation status of a neuron
Neuron Switch (NS)	Switch two neurons of the same layer
Layer Deactivation (LD)	Deactivate the effects of a layer
Layer Addition (LAm)	Add a layer in neuron network
Activation Func Remov (AFRm)	Remove activation functions functions

Table 2: Deep learning model-level mutation operators[7]

4.4 Test Implementation

Mutation is achieved by designing mutation operators that will introduce potential faults into the system under test (SUT) to create mutated versions of the originals. Mutants/faults will be introduced through a mutation operator into the original data D and model program P accordingly to slightly modify and obtain faulty mutated D' and P' respectively. The general MT implementation approach as demonstrated in figure 1 and same procedure of applying MT by [7] DeepMutation paper is closely followed in this thesis. Given an original program D or P, a set of faulty data D' or programs P' consisting of mutants are created based on predefined rules of the mutation operators, each of which slightly modifies the D or P. For example, a mutation operator can syntactically change '+' operator in the program to '−' operator. A complete test set T is executed against P and only the passed tests T' (a subset of T) are used for mutation testing. Each mutant of P' is executed on T'. If the test result for a mutant $p' \ \epsilon \ P'$ is different from that of P, then P' is killed otherwise, p' is survived. When all the mutants in P have been tested against T, mutation score is calculated as the ratio of killed mutants to all the generated mutants [7].

The testing process involves only mutating a very small amount of the original

training data while each of the mutation operator for model-program is applied and is then trained on the original training data D to obtain both D' and P'. After the training data and training program are mutated by the mutation operators, a set of mutant DL models M' is obtained. Each test data point $t' \ \epsilon \ T'$ that is correctly handled by the original DL model M is evaluated on the set of mutant models M'. Mutation operators for model-program will be designed to randomly mutate the weights and neurons of the to be used DNN architectures using the a tool.

The implementation will be based on a Keras compatible version, running on top of Tensorflow backend. Mutants injection to the source training data will be carried out through automated mutant generation and python mutant generation tool will be used to inject mutants to the models-program. Introducing mutants to the model-program will involve analyzing the DNNs structure and automatically introducing of the mutant to either the weight or neurons in a randomnized way depending on the analyzation and purpose of the mutant. The produced mutated models versions will then be serialized and stored on a python binary data file format.

5 Evaluation

The goal of this thesis is to replicate the approach used by Let et.al [7] DeepMutation paper to apply mutation testing to deep learning systems. I will propose two methods of analysis to evaluate the testing framework after replicating the paper's approach and the work of the authors; a quantitative analysis and a qualitative analysis. In quantitative evaluation, the mutation testing that will be re-implemented on the DL model will be evaluated based on the mutation score, following the same pattern used by the authors and as well a qualitative study to compare, analyze and evaluate the result contribution of the authors. Quantitative analysis method will statistically describe and interpret the general mutation testing and the author's research work using some metrics involving obtaining the mutation score. The mutation score metric for traditional soft-

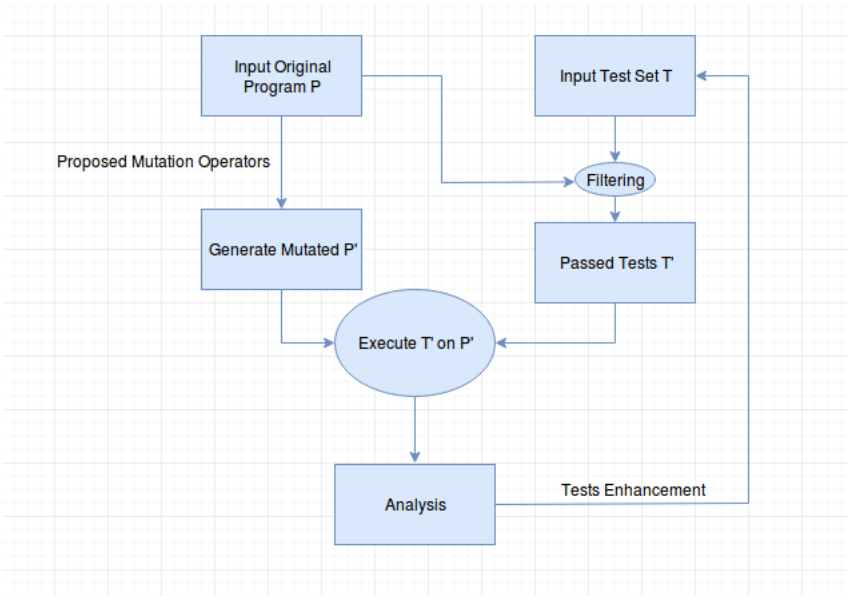

Figure 1: A General Process of Mutation Testing [7].

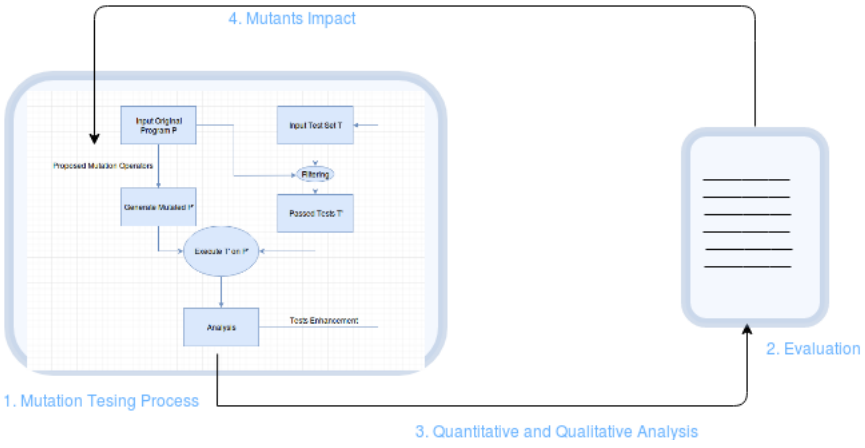

Figure 2: Proposed Method.

ware can easily be obtained as the ratio of number of faults/mutants killed in the mutated version of the dataset or model program detected by the test case in comparison against the total number of introduced mutants by the mutation operator. $MutationScore = \frac{NumOfMutantskilled}{NumOfAllGeneratedMutants}$

Using this simple metric to calculate the mutation score for DL models, precision in evaluating the quality of the test data for DL systems may not be achieved because if the size of T' is large, it is very easy for T' to kill all mutant in DL models. The authors therefore defined a metric for calculating the mutation score that focuses on classification problem. By correctly classifying the input data $c_1, ..., c_k$ and a test data point $t' \ \epsilon \ T'$ that runs on the data, if t' kills $c_i \ \epsilon \ C$ of mutant $m' \ \epsilon \ M'$, then t' is correctly classified as c_i by the original DL Model M or else t' is not classified as c_i by m' a more precise mutation score can then be calculated where KilledClasses(T', M') is the set of classes of m' killed by test data in T'

$$MutationScore\,(T', M') = \frac{\sum m'\epsilon M' |KilledClasses(T', m')|}{|M'| \times |C|}$$

. Simmilarly, average error rate (AER) will be used to measure the error rate of each mutant to avoid too many behavioral differences of the mutated model from original and to whether considered such mutant for further analysis or not using same metric as the proposed metric that takled the classification problem.

$$AER\,(T', M') = \frac{\sum m'\epsilon M' |ErrorRate(T', m')|}{|M'|}$$

[7] The measuring metric for AER of a mutant will also be used in investigating the impact of mutant and other evaluation metrics (i.e F1 Score) may also be considered to support evaluation.

Qualitative evaluation will aim to increase the overall understanding and usefulness of mutation testing technique and the contribution of the work of the authors. In the qualitative study, I will further elaborate mutation testing framework understanding, analyze the approach and report findings of the research.

Phase	Weeks
Initiation, proposal and presentation	-
Prepare data and DL models	2
Design and introduce mutation operators	4
Execute test (Train)	4
Evaluation (Using Metrics)	4
Investigate mutants impact	4
Analyse, report and documentation	6

Table 3: Time Schedule

6 Schedule

Table 3 shows how this thesis is planned to be carried out, highlighting time allocation for each tasks involved. The thesis is expected to be completed within 6 months from the date of official registration and time duration for each step to be involved is assigned as shown in the table 3.

7 Success criteria

The success criteria requirement of this thesis is organized into three categories as shown in Table 4. Since the main objective of this thesis is to repeat the work of Lei et.al DeepMutation paper, therefore; a must have requirement will be to re-implement their work. Analyzing and evaluating their work contribution will as well be a must have criteria while other tasks intended to be carried out in this thesis such as analyzing mutants impact will be a may have criteria.

8 Conclusion

By the end of this thesis. Mutation Testing will be implemented and applied to deep neural networks to test the quality of test case and to gain confidence of the trained model. The implementation procedure will be similarly to the approach

Task	**Must Have**	**May Have**	**May Not Have**
Implement mutation testing DL (Replication)	x	-	-
-Design mutation operators	x	-	-
-Metrics to measure and evaluate mutation score	x	-	-
Analyze and evaluate contribution	x	-	-
Investigate mutants impact	-	x	-
Mutation testing challenges			
-Equivalent mutants problem	-	-	x

Table 4: Success Criteria List

applied by Let et.al DeepMutation paper. The objective will be to analyse and evaluate their contribution and mutation testing technique usefulness in software testing. The Two groups of mutation operators are defined to be used as source of mutants for injection of faults and test cases will be designed to tackle and kill the mutants. The impact of the mutation operators will also be investigated to know the extent to how a mutant impacts the program.

References

[1] M. Al-Qizwini, I. Barjasteh, H. Al-Qassab, and H. Radha. Deep learning algorithm for autonomous driving using googlenet. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 89–96, June 2017.

[2] Vincent Aranega, Jean-Marie Mottu, Anne Etien, Thomas Degueule, Benoit Baudry, and Jean-Luc Dekeyser. Towards an automation of the mutation analysis dedicated to model transformation. *Software Testing, Verification and Reliability*, 25(5-7):653–683, 2015.

[3] Dawei Cheng, Chun Cao, Chang Xu, and Xiaoxing Ma. Manifesting bugs in machine learning code: An explorative study with mutation testing. In *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 313–324. IEEE, 2018.

[4] Lorena Gutiérrez-Madronal, Jose J Domınguez-Jimenez, and Inmaculada Medina-Bulo. Mutation testing: Guideline and mutation operator classification. In *The Ninth International Multi-Conference on Computing in the Global Information Technology*, pages 171–179, 2014.

[5] Y. Jia and M. Harman. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, Sep. 2011.

[6] Rene Just, Darioush Jalali, Laura Inozemtseva, Michael D Ernst, Reid Holmes, and Gordon Fraser. Are mutants a valid substitute for real faults in software testing? 11 2014.

[7] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, et al. Deepmutation: Mutation testing of deep learning systems. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, pages 100–111. IEEE, 2018.

[8] G. Petrovic, M. Ivankovic, B. Kurtz, P. Ammann, and R. Just. An industrial application of mutation testing: Lessons, challenges, and research directions. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 47–53, April 2018.

[9] Weijun Shen, Jun Wan, and Zhenyu Chen. Munn: Mutation analysis of neural networks. In *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 108–115. IEEE, 2018.

[10] Youcheng Sun, Xiaowei Huang, and Daniel Kroening. Testing deep neural networks. *CoRR*, abs/1803.04792, 2018.

[11] Jingyi Wang, Guoliang Dong, Jun Sun, Xinyu Wang, and Peixin Zhang. Adversarial sample detection for deep neural network through model mutation testing. In *Proceedings of the 41st International Conference on Software Engineering*, pages 1245–1256. IEEE Press, 2019.

[12] Xiaoyuan Xie, Joshua WK Ho, Christian Murphy, Gail Kaiser, Baowen Xu, and Tsong Yueh Chen. Testing and validating machine learning classifiers by metamorphic testing. *Journal of Systems and Software*, 84(4):544–558, 2011.

[13] Yuhao Zhang, Yifan Chen, Shing-Chi Cheung, Yingfei Xiong, and Lu Zhang. An empirical study on tensorflow program bugs. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2018, pages 129–140, New York, NY, USA, 2018. ACM.