Improving Test Cases Using Mutation Testing on Convolutional Neural Networks with Diverse States

Abdulhayyu H Lawal

August 23, 2019

1 Motivation

In the past few years, the use of Deep Learning (DL) has become an important concept in many applications and automation systems for making better analytical decisions. Learning from large amount of data, deep learning models automatically learn complex relationships in the training data. The quality of these deep learning architectures remain a major concern and present a big challenge especially in mission critical systems like Autonomous Driving. For example, the 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 person in time. Hence increasing the need for models with high accuracy. Convolutional Neural Network (CNN), one of the Deep Learning architectures has proven to be very effective in the area of visual recognition, classification, object analysis and in the autonomous driving field. This thesis will apply Mutation Testing (MT) to a Convolutional Neural Network deployed in a deep driving application system to assess the effectiveness and quality of a test suite in order to gain confidence on the deep learning model architecture. Mutation Testing is a fault-based testing methodology type where test case is designed and executed against mutated program to evaluate the ability of the test case in detecting the introduced mutants in the program [10], [19].

In this thesis, I propose a Mutation Testing for Convolutional Neural Network to evaluate the quality of the prepared test data and the effectiveness of the designed test cases. The deep driving framework implemented by paper [4] will be used to carry out the Mutation Testing, experiments will be carried out on the data and the deep learning model of the application that is built upon the state-of-the-art of deep CNN framework. The framework automatically learns image features for estimating driving affordance and makes driving decisions. The implementation will include designing two different groups of instructions for generating and introducing of mutants, the mutation operators for mutating the dataset (model-level mutation operator) and for mutating the CNN program (source-level mutation operators). Each of the mutation operators will be evaluated through a detailed experimentation using additional test cases designed to test and detect the mutants introduced by the mutation operator. The test cases will be executed against the mutated data/program to detect as many mutants as possible for every mutation operator. The testing process involves designing and injection of mutants to the original dataset and the original model to obtain mutated dataset and mutated model, and execution of the test case against the mutants in the mutated version to detect the differences.

As a byproduct, the challenges involved in applying Mutation Testing in general and to Convolutional Neural Networks particularly will be investigated. The two most common challenges to be investigated are the computational costs and equivalent mutant problems. These two challenges will be studied and ways to improve the testing technique by making it more widely applicable will be discussed. [19],[4].

2 Problem Statement

Majority of the research efforts conducted in the deep learning domain focuses on building more accurate models that can accurately achieve their intended goals [18]. However, little work has been done on assuring the quality and correct performance of these deep learning applications. The quality of these models can be evaluated through examining their performance on a test dataset by a test suite. Assessing the quality of the dataset and effectiveness of a test case to provide confidence on the models using MT. Mutation Testing is faced with some challenges and limitations and moreover the robust nature of machine learning models makes it even more harder contributing to the more challenges, holding back Mutation Testing from being a practical testing technique. [10],[18],[8].

This thesis proposes applying a mutation testing technique on a Convolutional Neural Network to assess the effectiveness of test cases which helps to gain insights and provide knowledge of the deep learning model's performance and accuracy. Gaining insights and confidence of the DL models will help address the accuracy concern and make the model more precisely functional in achieving their intended goals of implementation [12]. Applying Mutation Testing to assess and improve the test cases and make DL models more accurate comes with challenges that most programmers encounter and find difficult to address and my work of applying Mutation Testing to a deep learning model will be faced with similar challenges and problems. Although the test technique effectively assesses the quality of the test cases, problems such as the high computational cost of implementing the enormous number of mutants and executions and the equivalent mutant problem remain major challenges.

Mutation Testing has not been widely adopted in practice mainly due to its computational expense and manual effort required by developers in investigating unkilled mutants(i.e. mutants which are not identified up by the test suite) [3]. A program or data can have a fault in many possible places and with only one inserted semantic fault, only one mutant will be tested. Identifying these faults is a time consuming because a test case is executed for each of the corresponding mutant which may require several retraining and testing in order to catch and kill the mutant. To reduce these costs and make Mutation Testing affordable, my thesis propose some techniques such as using incremental mutation testing to limit the scope of mutant generation to sections of code which have changed since the last test suite execution or by reducing the number of mutant operators to limit the number of mutants and make the test more cost affordable. First Order Mutant (Exactly one mutation) and Higher Order Mutant (Mutant of mutants) are the techniques that can be used to decrease the number of mutants or operators to be executed [12].

The second challenge in applying mutation test is the equivalent mutant problem, which occurs when mutant operators that are syntactically different but exhibit the same functional behaviour. These mutants are hard to be determined in terms of contribution and difficult to catch by a test case. Increase in the number of equivalent mutants decreases the adequacy of the mutation score which decreases the test suite sufficiency to detect all mutants. To solve this challenge, techniques like compiler optimization, using constraints test data generation, program slicing, selective mutation and categorizing mutants based on the impact can be adopted. [8], [12].

This thesis do not only apply a Mutation Testing to a Convolutional Neural Network implemented in a deep driving framework to assess and improve the effectiveness of test cases which helps to gain insights and provide knowledge of the model's performance and make them more precise and accurate but also looks into the challenges involved in applying Mutation Testing to provide technical solutions to solving the challenges.

3 State-of-the-Art and Related work

There has not been many research attempts on how Mutation Testing technique can be applied to specifically test Deep learning system's quality performance [10]. Although, there are many mutation testing research work designed to test software systems or applications, but proving the quality performance and effectiveness of deep learning systems through assessing and improving of test suite using Mutation Testing have not been given the attention it deserves [19]. The most recent adequate research work that applied mutation testing on deep learning system was conducted by [10]Lei Ma et al. August 2018.

In this section, I have gathered some of the literature that are relevant to Mutation Testing, Convolutional Neural Network or related to both and also papers that discuss challenges and problems of Mutation Testing. 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. Aug 2018.[10] The paper proposes a Mutation Testing framework to assess test cases and to measure the quality of deep learning systems. The paper 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. 2017.[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.

Xie et al. 2011[18]. This is a research paper that focuses on applying metamorphic testing to test and validate Machine learning classifiers. Machine learning classification applications have become more crucial due to their prevalence in society and the paper help to address the quality of such applications or programs by applying the test technique and validating the results. The paper conducted cross-validation and mutation analysis to prove the effectiveness and correctness of the classifiers.

Rene et al. 2014[9]. 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. 2014.[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).

Junhuo et al. 2017.[6] Proposed an approach for validating the classification accuracy of a deep learning framework that includes a convolutional neural network, a deep learning executing environment, and a massive image data set. They proposed and explained approaches to validate deep learning classifiers using metamorphic validation approach.

Goran et al. 2017.[13] 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.

Chen et al. 2015.[4] Developed a model that is built upon the state-of-the-art deep CNN framework that automatically learn image features for estimating driving affordance and driving decision behavior. The framework and model implemented by them will be used in my thesis work.

Xiaowei Huang 2017.[16] 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.

Murphy et al. 2016.[11] Discussed some of the major network parameters and learning techniques related to Convolutional Neural Network. They provided an overview of some CNN architectures and their recent developments.

Jia et al. 2011.[8] 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.

Mark et al. [3] Addressed computational expense as one of the practical challenges of Mutation Testing and proposed a method to address the challenge.

Quanget et al.[12] Talked about problems of Mutation Testing. They considered the main limitations of Mutation Testing and discussed higher order mutation to solve the problem of computational expenses of the testing.

Jingyi et al. [17] 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. [5] 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. [7] 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. [14] Proposed mutation operators to introduce potential faults and define test case to be execute the faults and calculated the mutation score as test adequacy.

4 Research Goal

The main goal of this thesis is to apply Mutation Testing to Convolutional Neural Network, to evaluate and assess the quality performance of the test suite and to investigate the problems and challenges of using mutation testing technique. Two mutation operator groups to generate mutants to be injected into both the training data and the architecture of the CNN model will be defined and a test suite will be designed to be executed on the mutants in order to catch and kill the injected mutants. This is with the aim of evaluating the test suite effectiveness and examining the DL model's performance on a dataset to obtain confidence of the trained CNN model. The thesis will address the two most common challenges of Mutation Testing technique and propose solutions and techniques to improve the testing process.

The test will be carried out on an already developed driving decision controller framework by paper [4]. The driving decision controller consist of a CNN model that automatically learns image features for estimating driving affordance and driving decision behavior. The mutation operators are introduced to the image training dataset captured from TORCS game engine and to the CNN model program to assess the test suite for the mutated data/program. Technical recommendations/solutions to improve the testing technique will be discussed in the thesis. In the process of applying the testing technique, to evaluate the test cases and proposing of solutions to improve the testing technique; the following research questions will be addressed.

- How can Mutation Testing be applied to Convolutional Neural Network that is already deployed in a driving decision controller application?
- What are the challenges/problems of applying Mutation Testing specifically to deep learning systems and how these challenges can be approach to make the testing technique more viable.

  Computational cost of Mutation Testing and

  Equivalent mutants problem.
- What makes more sense to test and which of the test mutation operator performs better or has the least impact as mutant? A detailed discussion and explanation of each test mutant operator and its type of impact i.e alive, stubborn or equivalent mutants.
- What is the overall performance and effectiveness of a Convolutional Neural Network (Application Under Test) after applying Mutation Testing and what effect does the Mutation Testing have on the driving decision, if any?

5 Methodology

In this thesis, Mutation Testing will be applied to a deep Convolutional Neural Network that is trained using recordings from a driving scenario in TORCS game engine. A vision-based paradigm proposed by paper [4] will be mutated using proposed mutation operators. Their framework [4] uses direct perception approach to estimate the affordance for driving which is then passed to a driving controller and trained the CNN on the recorded images from the TORCS engine. Both the recorded training data and the model program will be mutated and tested to assess the test suite effectiveness and to obtain the model's performance accuracy on the recorded test dataset and to provide confidence of the DL model itself. Training dataset for this thesis will be collected from a driving scenario in the TORCS game engine or optionally from BeamNG game engine simulator. The already collected recordings from paper [4] can be reused as the training data as well or together with the new recordings. This is so, different sets of data can be used to asses the test case effectiveness. To adequately evaluate my implementation work, the test may also be applied to the KITTI dataset consisting of over 40,000 stereo image pairs taken by a car driving through European urban areas or to any other external dataset, this is in order to analyze and validate the test effectiveness of the test case on real dataset also.

The thesis will first involve full implementation and testing of the framework from paper [4] and then Mutation Testing will be applied to the data and the CNN model in the framework. The trained model takes a driving scene image from the game and estimates the affordance indicators for driving. The driving controller processes the indicators and computes the steering and acceleration or brake commands and then the driving commands are sent back to the game to drive the host car [4]. Upon successful implementation and running of the framework, the training dataset and the model will be mutated using the proposed mutation operators, each proposed mutation operator will be analyzed individually to identify its impact as mutant to the AUT and to assess the effectiveness of the test case for each mutant. Mutation operators in the implementation section shows the list of proposed groups of mutation operators defined to mutate the training dataset and the CNN model one at a time and both at the same time to analyze the different mutation implementation. The reason for the different mutation implementations is to find out if mutating one at a time or both at the same time results differently or not or has a different behavior.

Mutating the training data is about modifying the recorded images by introducing the proposed mutants into the images. For example, an image is modified by adding or deleting one of its property like color, texture or adding noise to some pixels of the image. Injecting of these mutants will create a modified version of the original image and then a test case will be executed on the mutated and unmutated image. A good test case catches the differences between the two image versions by killing the injected mutants. In the CNN architecture, a mutation operator will be introduced to create a mutated version of the model's architecture, i.e an input layer can be added or removed in the model's architecture to create a different mutated version of the original and a test case will then be executed to test and catch such mutants.

The general procedure of applying Mutation Testing to both the training data and the training model architecture is illustrated in figure 1. Using the proposed mutation operators, the original training data D and original training CNN model architecture P are modified to create mutated version of both as D' and P'. The Input test set T depending on the purpose it will be designed for either D or P is executed against either the D and P respectively. The data and program that did not pass the test are adjusted. Passed test set T' from the T that runs on D and on P is further executed against each D' and P'. If a test result for a mutant D' or P' of all mutants is different from main original D and P, the mutants in D' or P' are caught and killed, otherwise mutants in D' and P' are still alive and more tests need to be carried out . Mutants after testing can be and are classified as either killed, living or stubborn and when the mutants are still alive after several tests, the mutant is referred as a stubborn mutant [15], [10].

After all D' and P' are executed and tested against T', a mutation score can then be obtained which can be calculated as number of all mutants killed divided by the number of all mutants introduced. The mutation score obtained decides so much on the quality and effectiveness of the test case, the higher the score of the mutation, the more likely to capture the injected defects as mutants, proving the test case's effectiveness. In the evaluation section, a detailed way of calculating the mutation score will be discussed (Equivalent mutants are also considered) and evaluation metrics such as classification metrics and F1 score will be used to evaluate the test effectiveness and accuracy.

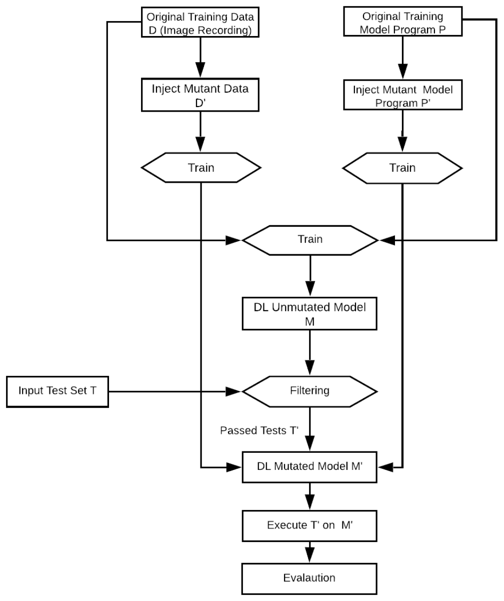

Figure 1: A General Mutation Testing Architecture Flow [10].

6 Implementation

6.1 Test Subject

The first step in Mutation Testing is to prepare the data for mutation, evaluation of the test case and for the training and testing the model for this thesis will be collected from TORCS or optionally BeamNG or even other openly available image datasets. Both game engines are open source racing car simulator environment and are widely used in artificial intelligence research field. Data from either of the two frameworks could be used as the source of training and testing data and will be collected by recording from a driving scenario or alternatively the already recorded dataset or the KITTI dataset by [4] can be reused and mutated.

Either the training, testing or both fractions of the data or the CNN neural network can be mutated and compared with the corresponding unmutated data/program results. From the perspective of the Convolutional Neural Network, a mutation is a change in the CNN's neural architecture that will then be checked to see, if after the change/mutation, the test case is able to catch the change/mutant. The mutating of the test data or the model architecture can be done either by manually introducing mutant operators or in an automatic way using openly available mutation tools like MutPy.

6.2 Proposed Test Mutant Operators

The general goal of Mutation Testing is to evaluate the effectiveness of a test case that are executed on the mutants/faults that are deliberately introduced to slightly modify original data/program to produce mutated version. In this subsection, I propose 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. Each mutation operator that performs a modification to the original program by introducing potential faults will be discussed and experimental evaluation will be conducted on both mutant operator groups. The Mutation Testing operators for the input training data include;

Data Duplication (DD): This operator duplicates a portion of the same data in the training data. Since the training data at the moment is planned to be collected from the TORCS simulation environment, some small portion of the collected or recorded image data will be duplicated to serve as a modification to it.

Data Property Manipulation (DPM): This operator manipulates the properties of some data. This can be done by editing the properties of an image, e.g editing image's pixel dimension or the number of layers in the image.

Data Outliers Introduction (DOI): This operator will involve introducing data outliers to the dataset group that do not fit well with rest of the dataset.

This is to indicate that the dataset now contains some additional unfit and unusual data in the test sets.

Noise Perturbation (NP): Noise is added to the training data randomly, some images of the training data will be added or similar disturbance that can cause noise to the data.

Data Shuffle (DS): Training data is shuffled before the training process, this is for the training data to appear to the model in unexpected order. Shuffling the order of the test data can allow the model to be observed if the shuffling impacts its behavior or not.

Due to the architectural structure of deep learning systems, traditional Mutation Testing cannot be applied directly to DL systems, I adopt the same mutation testing framework procedure specialized for DL systems proposed by [10]. The procedure includes using the same spirit of mutation testing for traditional software but with different set of mutation operators to introduce mutants to the neural network. The proposed mutation operators for mutating the CNN model are:

Activation Function Change (AFC): This modifies the neural network, the activation function of a whole layer is changed. The operator changes one or more activation functions in the deep neural network.

Weight Shuffling (WS): Some selected weights are changed through shuffling. The operator shuffles the weight of a neuron's connection, previous neuron determines the weight of a layer below it and with this operator the weight between the two layers is shuffled.

Layer Deactivation (ND): This operator removes only the transformation effects of a layers, because removing the whole layer would bring inconsistency to the DL model, only the effects are removed and it will be as if the whole layer is deleted.

Neuron Activation Inverse (NAI): With this mutation operator, the activation status of a neuron is changed. Changing the status also creates non-linear behaviors of the model.

Neuron Effect Block (NEB): This operator blocks the neuron effect of all of the connected neurons on the layer following it.

6.3 Test Implementation

The implementation involves designing and applying of Mutation Testing to a Convolutional Neural Network that is already deployed in a deep driving decision controller developed by [4] in a deep autonomous driving framework. The Mutation is achieved by designing mutation operators that will introduce potential faults into the application under test to create mutated versions of the originals. Due to the domain characteristics of the neural network and in order to add influence or manipulate the neural network model to produce mutated model/program, two groups of mutation operators are designed in this thesis and each operator consist of number of mutants in it that modifies the data or program and are introduced either globally to all or locally to specific types of data or program. The mutation operators that are designed and proposed are derived from some of the mutation operators from the start-of-the-art. The proposed mutation operators will be defined using some of the available mutant generation tools like MutPy and manual creation and introduction of these mutants will also be carried out especially in the mutating the data/images.

The implementation will be in Python programming language running on Linux and will be built based on Caffe communicating with TORCS environment. The implementation of the testing technique is to be carried out in three test rounds; in the first round of test, the test is only applied to the training dataset without mutating the model. The second test will only be on the CNN model while the dataset remains unmutated and in the third test, both the dataset and the model program will be mutated at the same time. In all the three rounds, the aim remains the same which is to assess the test suites effectiveness and also watch closely if mutating a part of the AUT alone has a different effect. With the different rounds of test, I look to investigate into the testing technique's challenges and how they can be improved.

After installing and running of the paper [4] framework to understand how the application to be mutated works, the next step is to collect and record the training dataset or use the already collected dataset and design the two mutation operator group and begin the testing process. The general implementation process of Mutation Testing to the AUT and for this work is shown in Figure 1. It involves re-training of the models any time faults are injected to obtained mutated versions and test case is executed. The testing process starts by injecting and creating of mutated version of the data and program. The mutation operators are then injected into the original data/program D and P to slightly modify and obtain faulty mutated D' and P' respectively.

The mutants injection for example in the CNN model program involves changing of the activation functions. Activation functions that determine the output of the neural network in the model is mutated by changing the effect of the function. Each neuron has parameters which include; its weight matrix (W), bias matrix (B), the activation function (Af) and the network structure (Ns). When inputs (x) are passed into the neurons, the inputs to the neuron is multiplied by the neuron's weight and the output is taken to the next neuron layer. Any modification to any of the parameter changes the output and affects the next layer. A test case T is executed to catch this modification, before the modification, an output f(y) is expected from f(x) and modifying $f(x)'$ to become mutated, a $f(y)'$ mutated is obtained. The test case T is executed to catch op_{AFCw} and when $f(x)'$ is not same as $f(y)'$, the mutant is said to be feasible and is killed.

When a mutant is not killed by the test case, the mutant is said to be either living mutant that is not feasible or a stubborn mutant that require more test is required. After all $f(y)'$ mutants are tested against $f(x)'$, a mutation score is then calculated to find the ratio of the killed mutants ($f(y)'$) to all generated mutants. The quality of test suites can then be determined and the higher the mutation score the more likely to capture real defects in the program [10], [17].

7 Experiment

I plan to evaluate the effectiveness of the test suite by carrying out experiment on each of the proposed mutation operator. Same dataset that will be collected will be used to test all the mutant operators and evaluate their test cases. The CNN will be trained on the already collected data or on the new recordings and on the mutated versions as well, operators contributions and effects, statistical facts about the layers and neurons of the model will be fully documented after implementation and practical training of the model. The mutant operators shown in the proposed mutant operators section are to generate the mutant for both the training and testing dataset and the model. The mutation accuracy of the trained CNN model will be known and result of the its performance is obtained and evaluated. For example; in carrying out the practical experiment, the data images in my case are mutated using the proposed operators. The DPM operator for example can be used to edit the properties of the collected images, modifying the number of layers and pixel dimension of the image data. The test case is then executed to kill these mutants and a score of the mutation is obtained that determines the test case's ability and effectiveness. The research question to how I can apply mutation testing to CNN is discussed in the methodology and implementation sections and by the end of the thesis practical implementation and documentation to how it is done will be provided. Question to the challenges of applying MT to DL systems and how the test can be improved is in the evaluation section but the question about the mutant operator types and their performance effects can only be answered when the practical experiment and implementation is carried out. By the end of this thesis and the experiments, results of each mutant operator and the performance accuracy of the trained model will be known and real statistical results are obtained and presented. Experiment and answers to the research question can be fully addressed at the end of the whole work with real figures and supporting results.

8 Evaluation

The evaluation will be based on the mutation score that will be obtained from mutation process execution of the test suite, the 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. Since the implementation process will involve three different testing execution rounds, mutation score for each test suite will be obtained for each

test execution; mutating of the training data only, mutating of CNN model only and mutating of both training data and the model at the same time.
After mutating and training of the original data and program, the test case that behaves well and correctly handled by the original data or program is considered passed T' and is again run on each of the corresponding generated mutant versions. While testing the mutated model, a test is considered a passed test if the test case is able to distinguish between the original and mutated program by killing all the mutant introduced.

8.1 Evaluation Metrics

Classification Accuracy
Using a classification accuracy metric, mutation score for passed tests that may include equivalent mutants can be obtained as shown.

$$MutationScore = \frac{KilledMutants}{TotalMutants - EquivalentMutants}$$

The total killed mutants ratio is compared against total mutants introduced minus the equivalent mutants that may appear. The equivalent mutants ration is included in order to obtain a 100 percent mutation score. The higher the mutation score, the more accurate and effective the test cases in catching and killing the mutants.

F1 Measure
F1 score will as well be considered in this thesis to measure the test's accuracy. To measure the test case accuracy through the mutation score using this method, a Precision and a Recall will both be calculated to computing the score. Precision is number of correct positive results which in my case will be the number of mutants killed by the test case, divided by the total predicted positive observations which will be the total number of mutants introduced. Recall, on the other hand, is the number of correct positive results (killed mutants) divided by the total actual positives (total introduced mutants). A mutation score obtained with a high precision but lower recall will give us a high accurate confidence and at the end the greater the F1 Score, the better test case's effectiveness.

$$F1 = 2 * \frac{1}{\frac{1}{Precision} + \frac{1}{Recall}}$$

8.2 Proposed Solutions to the Chanllenges

The two most challenging problems of Mutation Testing as stated in the previous sections are further discussed and some recommendations or solutions to the challenges are proposed to make the testing technique more widely used by testers. In Mutation Testing, programs are mutated. The re-execution of test case against the program and re-training anytime a mutant is introduced or when ever a mutant is searched to be identified make MT a costly testing technique as it requires time to re-train and re-execute. Reducing the cost of re-executing or re-training can be through applying incremental mutation testing. With incremental mutation testing, the scope of the mutant generation is limited to some section of testing data or program and to part of the data/program that has not been changed since the last mutation run. This method puts the mutant to a specific part of the data/program and reduce the number of mutant operators making the test cost affordable due to the decrease in the number of mutants or operators to execute, this technique can be through First Order Mutant (Exactly one mutation) and Higher Order Mutant (Mutant of mutants)[12].

The other worrying challenge of Mutation Testing is the equivalent mutants problem. This problem occurs because some mutant operators may be syntactically different but functionally equivalent and can produce equivalent results that may have the same behavior with other syntactically mutants or with the original program. These mutants are hard to catch and killed and increase in the number of equivalent mutants decreases the adequacy of the mutation score which decreases the test set sufficiency to detect all mutants. To solve this challenges, the mutation score formula is first reformulated to include equivalent mutants in order to make the score for adequate and sufficient. Some of the techniques to help deal with the equivalent mutant problem includes compiler optimization to minimize some attributes of the executable program, automatic generation of test data using constraints test data generation, program slicing to easily locate source of error, selective mutation to reduce number of mutants that must be executed and lastly categorizing of mutants through examining the impact of each mutant [8], [12].

Detail evaluation and proposed recommendations will be fully discussed and documented, and the final documentation will contain real figures, facts and numbers from the experiment that will be performed on each proposed mutation operator while carrying out the testing process.

9 Requirement List and Schedule

9.1 Requirement List

The thesis requirement is organized into three categories as shown in Table 1:

- MUST HAVE requirements are the most important part of the thesis that I must implement and accomplish.
- MIGHT HAVE requirements are to support and improve the quality of this thesis.
- MUST NOT requirements define the scope of this thesis that are not planned to be implemented.

9.2 Schedule

The schedule in Table 2 shows how the thesis is planned to be carried out, highlighting time allocation for phases, duration and tasks to be involved in the thesis.

Requirement List	
Requirements Options	Remark
Must Have	• Apply mutation testing to deep learning system (convolutional neural network). This invloves designing of mutation operators, assessing and evaluating test suite that are executed on the mutated data/program. • Investigate the challenges of applying mutation testing and provide recommendations/solution to how the challenges can be solved or minimized.
Might	• Perform the test also on some widely used datasets apart from the training dataset (Recordings from the TORCS environment). Mutate different convolutional nueral network architecture type to find out which CNN is more resilient to mutation testing.
May Not	• No own driving decision controller program will be developed, the driving controller developed by [4] paper will be modified and used. • The test maybe applied to CNN as it is only, Not a framework with CNN e.g Paper ([4])

Table 1: Thesis requirement list.

Schedule		
Phase	Duration in Months	Tasks
Initiation, Planning, Proposal and presentation	1-6 (Before official registration)	topic research, -related materials gathering, -state-of-the-art research, proposal documentation, -discussion/feedback -submission and presentation of proposal.
Implementation and Documentation	1-5	reading, -coding, -design, -documentation -discussion/feedback.
Testing and Submission	1	final testing, -presentation, -submission.

Table 2: Time allocation

10 Conclusion

By the end of this thesis. Mutation Testing will be implemented and applied to Convolutional neural network to test the effectiveness of test suite and to gain confidence of the trained model. It will become clear to whether and how mutation testing can be applied to CNN deployed in a deep driving application framework of an autonomous driving system. Two groups of mutation operators will be defined to be used as source of mutants for injection of faults and test suites will be designed to tackle and kill the mutants. Solutions to the problem of applying mutation testing will be provided to make the test more affordable and evaluation of the test's accuracy will carried out and one or two evaluation metrics for measuring deep learning algorithms performance will be used to calculate mutation score and evaluate the test adequacy from which a conclusion will be made about the effectiveness of the test.

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] Mark Anthony Cachia, Mark Micallef, and Christian Colombo. Addressing practical challenges of mutation testing.

[4] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[5] 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.

[6] J. Ding, X. Kang, and X. Hu. Validating a deep learning framework by metamorphic testing. In *2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET)*, pages 28–34, May 2017.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] John Murphy. An overview of convolutional neural network architectures for deep learning. 2016.

[12] Quang Vu Nguyen and Lech Madeyski. Problems of mutation testing and higher order mutation testing. In Tien van Do, Hoai An Le Thi, and Ngoc Thanh Nguyen, editors, *Advanced Computational Methods for Knowledge Engineering*, pages 157–172, Cham, 2014. Springer International Publishing.

[13] 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.

[14] 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.

[15] Ben H Smith and Laurie Williams. On guiding the augmentation of an automated test suite via mutation analysis. *Empirical Software Engineering*, 14(3):341–369, 2009.

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

[17] 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.

[18] 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.

[19] 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.