On Accelerating Deep Neural Network Mutation Analysis

by

Lauren Lyons

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 10, 2025

Keywords: Deep Neural Network, Neuron, Mutation Analysis, Mutant, Clustering, Test Data Selection

Approved by

Ali Ghanbari, Chair, Assistant Professor of Computer Science and Software Engineering
Akond Rahman, Member, Assistant Professor of Computer Science and Software Engineering
Sanchuan Chen, Member, Assistant Professor of Computer Science and Software Engineering

Abstract

The usage of Deep Neural Networks (DNNs) is increasing rapidly across various domains, necessitating rigorous testing to ensure validity, usability, and effectiveness. Mutation analysis of DNNs, a technique that applies mutations to models, creating mutants used to evaluate effectiveness, has emerged as a powerful approach for assessing model robustness. However, existing mutation analysis techniques for DNNs face prohibitive computational costs, especially for large real-world models. This creates a critical need to accelerate the analysis of DNN mutations while maintaining the effectiveness of the testing.

The primary contribution of this thesis is DEEPMAACC, a novel tool that was designed to mitigate these computational expenses. DEEPMAACC implements two distinct acceleration methods: neuron clustering and mutant clustering. Both methods utilize hierarchical agglomerative clustering to group neurons or mutants with similar weights, with the aim of improving efficiency while maintaining the accuracy of the mutation score. To evaluate DEEPMAACC, this research conducts an empirical study using eight DNN models on four popular classification datasets and two DNN architectures.

The secondary contribution of this thesis is two more approaches that are used to accelerate the mutation analysis of DNNs. The approaches are Random Mutant Selection and Boundary Sample Size Selection. Random Mutant Selection is inspired by Ghanbari *et al.* [1] and acts as a baseline to show that randomly choosing a certain percentage of mutants will not result in the same outcomes as DEEPMAACC. Boundary Sample Size Selection is a unique approach inspired by Shen *et al.* [2] which only tests on certain sensitivities of decision boundary samples.

Acknowledgments

I would like to acknowledge my advisor and committee chair Dr. Ali Ghanbari for his support during my accelerated degree at Auburn. His patience, kindness, and urgency was key to any of my success. I appreciated his depth of knowledge and guidance throughout the development of this thesis, along with the special opportunities he granted me. I also would like to express my gratitude to my committee members: Dr. Akond Rahman and Dr. Sanchuan Chen for their unique perspectives and feedback. I would like to extend my sincere thanks to my family, friends, and roommates for their continuous support through my bachelor's and master's degrees. Finally, I would like to honor and thank my late father, Edward Eugene Lyons III, for inspiring my attendance and success at Auburn University. I wish he could have been here to join me for this journey.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep learning [3] that is based on *deep neural networks* (DNNs), has found applications in numerous fields. These include computer vision [4], speech recognition [5], natural language processing [6], software analysis [7], malware detection [8], and more. These applications have deep neural networks as important components, meaning the behavior of these models must be robust and efficient. To ensure that these deep network systems work as intended, we can treat them similarly to traditional software systems. Among the various methods for ensuring the quality of DNNs [9, 10], testing is a commonly used approach [11]. However, we must approach the testing of Traditional Software and Deep Neural Networks separately, as their testing behaves differently. The different programming paradigms and behavior expectations results in it being difficult to directly apply traditional software testing practices to these Deep Neural Networks.

Mutation analysis [12, 13], as a promising method for assessing test data quality, has recently been introduced in the context of DNNs [14, 15, 16, 17]. DNN mutation analysis can either be applied to the source code constructing and training the model, aka *source-level mutation analysis*, or directly to the trained model itself, aka *graph-/model-level mutation analysis*. Since their introduction, both source-level and model-level mutation analyses, have had plenty of applications [18, 19, 20, 21, 22, 23, 24, 25, 1, 26, 27, 28, 29, 30, 31]. Despite these opportunities, both forms of DNN mutation analysis involve generating and testing a large number of mutants, making them extremely costly processes. For traditional programs, *i.e.*, programs that are not exclusively based on data-driven pipelines, there is a large body of work concerning the topic of mutation analysis acceleration and their performance in various applications [32, 33].

There are a few methods that have been applied in the literature for accelerating DNN mutation analysis [34, 35, 1, 31, 36, 2]. However, to the best of our knowledge, we still do not have any insights on the performance of techniques based on clustering in terms of mutation testing cost reduction and their cost on the mutation score.

DEEPMAACC has shown the ability to reduce the overall mutation analysis time by several minutes. An exhaustive, vanilla approach of creating and testing all mutants for the allotted 60 runs can take around 250 minutes for FCNN models while LeNet-5 models take around 384 minutes. All acceleration approaches are run 6 different times across 10 different parameters, resulting in 60 different runs. The Neuron Clustering Approach can reduce FCNN model runtime to 75 minutes and LeNet-5 model runtime to 116 minutes. The Mutant Clustering Approach can reduce FCNN model runtime to 154 minutes and LeNet-5 model runtime to 255 minutes. The Random Mutant Selection Approach can reduce FCNN model runtime to 115 minutes and LeNet-5 model runtime to 175 minutes. The Boundary Sample Size Selection Approach can reduce FCNN model runtime to 25 minutes and LeNet-5 model runtime to 32 minutes.

1.1 DEEPMAACC Outline

To address the computational challenge of mutation analysis of DNNs, we introduce *DeepMAACC*, a technique and a tool that accelerates DNN mutation analysis. *DeepMAACC* employs two separate approaches: (1) neuron clustering to reduce the number of mutants generated, and (2) mutant clustering to select representative mutants for mutation testing. Both approaches use hierarchical agglomerative clustering methods to group neurons and mutants within a single layer with similar weights, which allows for more efficient mutation analysis while maintaining high coverage of the network's functionality.

Neuron clustering reduces the number of generated mutants by clustering neurons within a single layer that provide similar contributions to the networks and creating mutants by mutating the clusters rather than individual neurons. Creating mutants from each of the clustered neurons individually would be redundant, assuming clustered neurons behave similarly. In order to create mutants, we utilize three different model-level mutation operators that directly inject

faults into DNN models without a training process. Afterward, all mutants created from the neuron clusters are tested.

Mutant clustering does not reduce the number of mutants, as they are needed to create the mutant clusters, but instead reduces the number of mutants to be tested by selecting representative mutants for testing. Selecting a representative mutant means only that mutant will undergo testing for a cluster, while the others are not used. So, while it maintains the time and space it takes to create all the clusters, it speeds up the mutation testing phase by choosing representatives from each clustered mutant group and allowing that representative's mutation testing performance, *i.e.* mutation score, to be reused for all other mutants in that cluster that it is representing, therefore eliminating the need to test all mutants. Clustered mutants have similar weights, alluding that their applied mutations have had similar effects and will produce the same killed labels. However, the impact of the representative mutant is still proportional to its cluster size.

DEEPMAACC was evaluated on eight DNN models across four popular classification datasets and two DNN architectures. Then it is compared with an exhaustive, or vanilla, approach. To understand the performance of each approach, we focus on certain performance metrics and compare the difference from a vanilla approach.

1.2 New Acceleration Methods

The two acceleration methods, Random Mutant Selection and Boundary Sample Size Selection (BSS), were built on top of the DEEPMAACC framework.

Random Mutant Selection maintains all of the mutants from the original mutant set, but reduces mutant testing time, similar to the Mutant Clustering approach. It reduces the amount of mutants to test by selecting a user given percentage of mutants to test at random. The selected mutants are the only ones to be tested, meaning they contribute to the mutation score.

Boundary Sample Selection also maintains the original set of mutants, but reduces the mutant testing time by using a subset of the test dataset. This subset is chosen based on a user given ratio parameter that measures how far a test data point is from a decision boundary. The smaller the ratio, the closer the data point has to be to the decision boundary. When completing

the mutation score testing, the test set size can be reduced down to a minimal 3%. This smaller test set size reduces the mutant testing time by a reasonable amount since the mutant testing must perform an inference pass of the entire test data set for each mutant.

These two acceleration approaches are also evaluated on the same eight DNN models across the same four popular classification datasets and two DNN architectures. Then, they are compared with the vanilla approach as well in order to understand their performance.

1.3 Research Objectives

This thesis will focus on investigating three aspects of DEEPMAACC's performance for the two clustering approaches and the two selection approaches: speedup, mutation score, and their trade-offs. These goals can be realized in these research questions:

- **RQ1:** How much *speedup* is gained when using Neuron Clustering or Mutant Clustering versus Vanilla mutation testing?
- **RQ2:** How much *mutation score* is lost when using Neuron Clustering or Mutant Clustering versus Vanilla mutation testing?
- **RQ3:** What is the impact of non-determinism in the training process on the mutation testing *speedup* and *mutation score* of DEEPMAACC when using Neuron Clustering or Mutant Clustering versus Vanilla mutation testing?
- **RQ4:** How much *speedup* is gained when using Random Mutant Selection or Boundary Sample Size Selection versus Vanilla mutation testing?
- **RQ5:** How much *mutation score* is lost when using Random Mutant Selection or Boundary Sample Size Selection versus Vanilla mutation testing?
- **RQ6:** By what percentages is the testing data set reduced when generated by Boundary Sample Size Selection? Are they representative to evaluate the results of mutation testing under the whole test data? Are they more efficient subsets?

These questions are designed to comprehensively evaluate the effectiveness and trade-offs of the proposed acceleration techniques. These metrics are tested during neuron clustering, mutant clustering, random mutant selection, boundary sample size selection, and an exhaustive approach.

Mutation Analysis for DNN models is a very active area of research with two main mutation testing frameworks proposed by DeepMutation and MuNN. In this thesis, we focus on accelerating mutation analysis for DNNs. Many areas of research overlap in this statement. The work *DeepMutation* by Ma *et al.* [15] introduced a mutation testing framework specialized for DNNs, which gave this paper great inspiration. Some of the mutation operators, metrics, and methods have been more or less directly ported from *DeepMutation* in the attempt to have a commendable baseline. The research directions for testing and verification of DNNs range from directly from general mutation analysis methods to new DNN specific methods that highlight specific DNN characteristics.

DeepMAACC hopes to add further speedup to these various methods, exploring the application and performance of different neuron clustering approaches that reduce the number of tested mutants based on layer neuron similarities. So, much like the works by Feng *et al.* [34] and Wang *et al.* [35], a movement towards avoiding redundant mutants would improve speed while performing nearly the same in all metrics. The combination of several of these methods could be explored in a future work, and could prove to multiply the performance effects. The work of Dhulipala *et al.* [37] gives one method of neuron clustering while the work of *Incite* [31] follows a separate method of neuron clustering. Klabunde *et al.* [38] surveys a list of methods for measuring functional and representational similarities between DNN models. These methods could also be used for quantifying mutant similarities and clustering. However, the performance enhancement would need to be checked.

1.4 Contributions

This thesis improves the state of the art by (1) studying the effectiveness of four techniques for reducing the costs of DNN mutation analysis; (2) providing an implementation of the techniques, in a tool named DEEPMAACC, that can be integrated with existing mutation analysis

framekworks [15, 16] with a bit of engineering work. Our results demonstrate a trade-off between mutation testing speed and mutation score error.

In summary, accelerating mutation testing, a costly step in mutation analysis, could benefit many other applications that rely on DNN mutation analysis. This thesis makes the following contributions in this direction.

- **Approach:** This research revisits four methods for accelerating mutation testing. Two are based on clustering of neurons (to generate fewer mutants) and clustering of mutants (to test fewer mutants). The other two are based on randomly selecting mutant subsets (to test fewer mutants) and applying a filter to the test data set using the model's decision boundary (to reduce test data).

- **Implementation:** We have implemented the two mutation testing acceleration approaches using clustering in a publicly available, and easy to use, framework [39], named DEEPMAACC. The acceleration methods named Random Mutant Selection and Boundary Sample Selection are built on top of DEEPMAACC, but not released publicly. They are elaborated on within this thesis.

- **Results:** Our results shed light on the relative performance of four mutation analysis acceleration methods by comparing them to vanilla mutation testing. For the two clustering methods, our results suggest that one of the approaches yields a higher gain in mutation testing speed, while the other incurs less error in mutation score. For the two selection methods, they perform well in both aspects, but Random Mutant Selection has a lower Speedup with lower MSE. Boundary Sample Size Selection has a higher average error, but also a larger speedup. A replication package is available [39].

1.5 Bibliographical Notes

Parts of this thesis were previously published in an international conference. Specifically, DEEPMAACC work is published in the Proceedings of the 18th IEEE International Conference on Software Testing, Verification and Validation (ICST 2025). Dr. Ali Ghanbari from Auburn University was a co-author of this paper as well as my advisor for this thesis.

Chapter 2

Background

2.1 Mutation Analysis of Traditional Programs

Software testing has always been an integral part of software development. Mutation analysis [12, 13] is a program analysis method for assessing the quality of a test suite. The concept of mutation analysis dates back much farther to 1971 in a work by Richard Lipton [40]. Mutation operators were introduced as a key component of mutation analysis by Budd *et al.* [41, 42], specifically for Fortran. Extensive research and development has extended the application of mutation operators to different languages, including C, Java, C#, SQL, AspectJ, Ada, and IDL [43, 44, 45, 46, 47, 48, 49, 50, 51]. Great efforts have also resulted in performance enhancements for mutation analysis [52, 53, 54, 55, 56]. Research continues on this topic. *Mutation Testing for the New Century* brings together cutting-edge research on mutation testing from a wide range of researchers. The book addresses key challenges, provides innovative solutions, and highlights open research questions, making it a valuable resource for advancing the study and application of mutation testing [57].

The methodology of mutation analysis used by traditional programs involves generating a set of program variants, called *mutants*, by mutating program elements, *e.g.*, negating a numeric literal, using *mutation operators*, aka *mutators*, and running the test suite on the mutants to check if any differences between the outputs of the mutants and that of the original program is observed. If the check is positive, the mutant is marked as *killed*, otherwise it will be marked as *survived*. A mutant might survive because it is semantically equivalent to the original program, hence the name *equivalent mutant*. Test suite quality is measured by *mutation score*, which

is traditionally defined to be the ratio of killed mutants over non-equivalent survived mutants. Mutation score of a test suite is assumed to be proportional to the capability of the test suite in detecting real bugs [58].

2.2 Mutation Analysis of Deep Neural Networks

DNN mutation analysis has been applied in many areas [18, 19, 20, 21, 22, 23, 24, 25, 1, 26, 27, 28, 29, 30, 31], and, like its traditional counterpart [58, 59], holds potential for many other applications. And like its traditional counterpart, DNN mutation analysis is known to be an extremely costly process [15, 20, 60]. Motivated by the potentials of DNN mutation analysis, in recent years, several techniques for accelerating DNN mutation analysis have been proposed.

For example, Feng *et al.* [34] and Wang *et al.* [35] took steps to reduce redundant mutants by identifying sufficient subsets of existing mutation operators from the literature [15, 14]. Ghanbari *et al.* [1] adopts random mutant selection, and the technique presented by Li *et al.* [36] accelerates mutation testing through higher-order mutants akin to higher-order mutation in traditional programs [61]. The technique presented by Shen *et al.* [2], reduces mutation analysis costs through test data selection. It relies on the assumption that mutants of a DNN model are more likely to produce different results for test data points around the decision boundary of the model, so it samples the test data points that lie at the decision boundary of the model during test data inference. Lastly, *Incite* [31] uses neuron clustering to curtail the cost of mutation analysis for the purpose of modular decomposition of DNNs. In this research, we studied the effectiveness of this technique and concluded that, while neuron clustering results in greater speedup, clustering at the level of mutants better preserves mutation score.

Additionally, although there have been attempts to reuse the existing mutation analysis tool PIT [62] on Java-based DNN systems [63, 64], the efforts of the research community has shifted entirely toward DNN mutation analysis systems with specialized mutators. Shen *et al.* [14] and Ma *et al.* [15] developed the first dedicated techniques for mutation analysis of DNNs. DNN mutation can be conducted at the source level, *i.e.*, the training program and/or the data, hence the name *source-level mutation analysis*, or at the model level, *i.e.*, the graph

corresponding to the trained model, hence the name *graph-level* or *model-level mutation analysis*. Both forms of mutation analysis are costly [20, 60], so there is a recent research trend in accelerating this process [34, 35, 1, 36, 2, 31]. The two forms differ from each other primarily in the way the mutants are generated, the former involves mutating the source and/or the training data and training the resulting mutants, while the latter directly mutates an already trained model. The process of testing the mutants in both forms of DNN mutation analysis are identical to each other, and one of the cost reduction methods presented in this thesis is readily applicable in both contexts. Both approaches to mutation analysis of DNNs, especially model-level mutation, has been applied in many areas, including adversarial sample detection [18] and generation [19], robustness analysis [20, 21], aiding manual labeling of test data via prioritization of test data [22], accuracy estimation to alleviate the need for manual labeling of test data [23], fault localization [1], automated repair of DNNs [24, 25], modular decomposition of DNN models [31], and improving the quality of test dataset by generating new data points guided by mutation testing [26, 27, 28, 29]. However, good performance on the test dataset does not necessarily imply the robustness and generality of a DNN model, and a systematic way for assessing the quality of the test dataset itself is needed. This need is amplified with the growing applications of DNNs in safety and mission critical systems, such as autonomous driving [65], aviation [66], healthcare [67], financial fraud detection [68], and energy [69]. The need for accuracy and robustness of such systems, justifies research on quality assurance of DNNs. Additionally, researchers have developed mutation analysis frameworks specialized for certain types of DNN-based systems, *e.g.*, autonomous driving [30], as well as certain DNN flavors, such as reinforcement learning [17, 70]. The cost reduction techniques implemented in DEEPMAACC could benefit all these application areas.

Mutation analysis of deep neural networks (DNNs) is a promising method for effective evaluation of test data quality and model robustness, but it can be computationally expensive, especially for large models. The application transferring from traditional programs to DNNs carries many similarities, but the system behaviors require changes to ensure total applicability. This tweaked method still involves mutant generation through mutation operators, while the quality of a test suite is measured by the mutation score. Again, a test suite with a higher

mutation score has a greater capability of catching real defects in the program. This leads to improved confidence in the reliability and robustness of the model.

The main difference in the mutant survival methods appears in the actions that result in a killed or survived mutant. The traditional program's method of calculating a killed mutant would yield a much higher kill rate, as any probability output by the mutant model being different would kill the mutant. This would cause the deep learning system to lose the precision to evaluate the quality of test data [15]. To correct this for DNN systems, a test data point can kill a mutant if two conditions are satisfied: (1) the correct label is assigned by the original model, and (2) an incorrect label is assigned by the mutant model. This is further elaborated in Section 3.1 and Section 4.3.1.

The mutation score metric mentioned above is directly suited for classification problems, as this research focuses mainly on classification problems. However, the idea behind it can be extended to handle numerical prediction problems by using user-defined thresholds as the error allowance margin [71].

2.3 Deep Neural Networks

Deep neural networks (DNNs) are neural networks that have at least two hidden layers between the input and output layers. Two examples of DNN architectures are fully-connected neural networks (FCNNs) and convolutional neural networks (CNN). A FCNN consists of a series of fully connected layers, where each layer is a function $\mathbb{R}^m \Rightarrow \mathbb{R}^n$ where $m, n > 0$ are real integers. A common CNN architecture is the LeNet-5 architecture [72].

For this research, we use these two types of DNN architectures, namely, LeNet-5 and fully-connected neural network (FCNN). LeNet-5 architecture is a representative of the convolutional neural networks (CNNs) with no residual blocks, such as AlexNet [73] and VGGNet [74], while FCNN architecture is a representative of simple neural networks. These model architectures are trained on four popular classification datasets, *i.e.*, MNIST [75], Fashion MNIST [76], Extended MNIST [77], and Kuzushiji MNIST [78], resulting in eight DNN models in total.

2.4 Clustering

Within the mutation analysis of DNNs, we use two types of clustering to attempt acceleration while maintaining a stable mutation score. Agglomerative hierarchical clustering is used to partition neurons within a layer for neuron clustering. Parallel hierarchical agglomerative graph clustering (ParHAC) [37] is used to cluster DNN mutants for mutant clustering.

Agglomerative Clustering [79] is a type of hierarchical clustering that groups similar objects together by iteratively merging the most similar clusters until all objects are in clusters. Different linkage methods can be used to determine the similarity between clusters, such as single or complete linkage. Within this research, the measure of similarity in the clustering algorithm is Euclidean distance [80], inspired by Ghanbari [31]. Pre-defined cluster size N given by the user is passed to this clustering algorithm to result in all clusters having a max of N neurons within them.

ParHAC performs hierarchical agglomerative clustering on large similarity graphs [37]. The algorithm begins by taking a similarity graph as input, where vertices represent data points and weighted edges represent the similarities between these points. In particular, for the use case of this research, the data points are neuron weights, and the edges are the Euclidean distance between different neurons of the same layer. To parallelize, ParHAC employs a technique called geometric layering, where edges are grouped into layers based on their weights. These weight-based layers are then processed sequentially, starting with the layer containing the highest-weight edges. Throughout, ParHAC utilizes a compressed clustered graph representation to efficiently update the similarity graph after cluster merges, focusing on the neighborhoods of merged vertices rather than updating the entire graph.

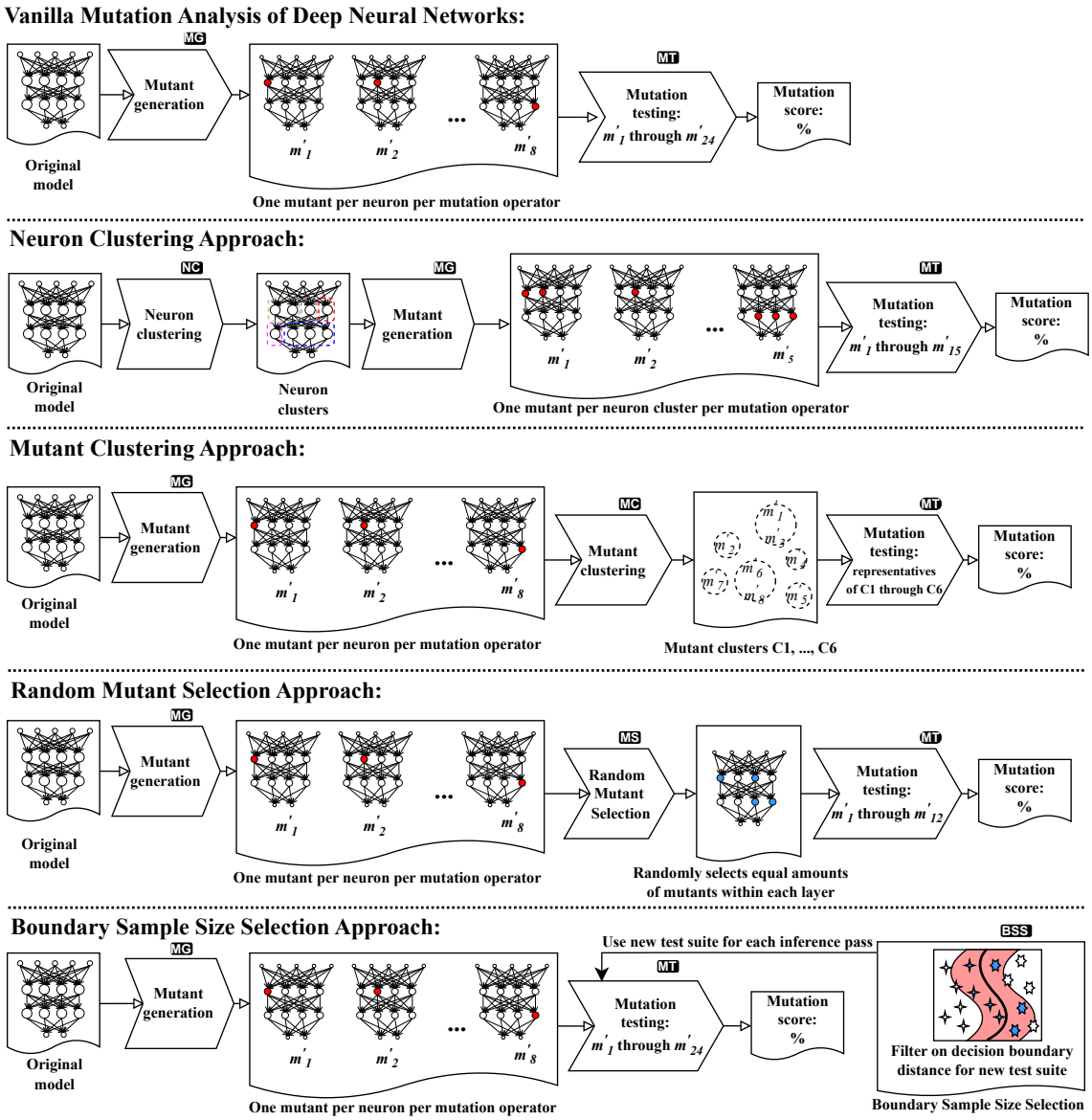

Figure 2.1: An overview of approaches implemented in DEEPMAACC: vanilla clustering approach that is the exhaustive baseline for this research (top row); neuron clustering approach that creates fewer mutants by clustering the neurons and mutating neuron clusters instead of individual neurons (second row); mutant clustering approach that clusters the mutants and tests a representative from each cluster (third row); random mutant selection approach that selects a random subset from each model layer to test (fourth row); boundary sample size selection approach that selects a smaller test suite based on test points near the decision boundary (bottom row). Chevrons (V-shaped blocks) represent processes, control flow is denoted by arrows, and document symbol denotes artifacts generated by the processes. Processes are marked with abbreviations for ease of reference in the text.

Chapter 3

Methodology

This research studies four methods for accelerating DNN mutation analysis that fall into the traditional categories of *doing few* and *doing faster* mutation analysis acceleration paradigms [53]. Specifically, we design, implement, and evaluate a system, named DEEPMAACC, that implements two approaches for mutation testing cost reduction through clustering for the purpose of reducing the number of generated mutants and also the number of tested mutants. Then, as an addition to DEEPMAACC, we implement two more approaches on top of DEEPMAACC, one that reduces the number of tested mutants and one that reduces the test data set size.

The first approach (doing few) clusters neurons within a layer, where all neurons in each of those clusters are simultaneously mutated to create a single mutant. An entire cluster is mutated at once due to the assumption that clusters contain neurons that behave similarly, reducing redundant mutants. This approach was recently implemented and applied for mutation-based DNN modular decomposition [31]; however, we lack insight into its applicability for accelerating mutation analysis to assess the quality of the test dataset.

The second approach (doing faster) involves generating mutants by mutating individual neurons, as one would do in an approach like DeepMutation [15] or MuNN [14], but the generated mutants are clustered based on the similarities of the mutated location. DEEPMAACC selects a representative of each cluster and reuses the mutation testing results, *i.e.*, whether the mutant's classes have been killed or survived, to all other mutants in the cluster. The intuition behind this approach is that mutations that result in similar impact are likely to behave similarly. Although this is a well-known mutation analysis acceleration technique for traditional

programs [81, 82, 83], to the best of our knowledge, acceleration through mutant clustering has not been applied in the context of DNN mutation analysis yet.

The third approach (doing few) selects at random a user-given percentage of the mutant set to test. This method has been implemented and tested before, [1, 84], with varying levels of success. Instead of exhaustively evaluating all mutants, this cost-reduction technique will probabilistically sample subsets of our already generated mutant set. This thesis employs the specific method named *two-round random selection*, named by Zhang *et al.* [84], but with a different implementation. Instead of selecting a mutant operator first and then randomly selecting a mutant generated by that operator, this method iterates through the Deep Neural Networks' layers and selects an equal amount of randomly selected mutants from each layer. Empirical studies demonstrate that selecting even half of the mutants via the *one-round* or *two-round* random selection can achieve high correlation while significantly reducing computational effort [84, 1]. Although we generally use this approach as a baseline for the effectiveness of mutation analysis, the intuition behind it is the assumption that the randomly chosen subset can adequately represent the overall pool of mutants by evenly sampling across all mutants. Being known as a baseline method, this approach can be used as such later in the paper.

The fourth approach (doing faster) selects a subset of the original test data set by strategically selecting test inputs near the model's decision boundary to test mutants' mutation scores. To choose the test data subset, we leverage the ratio of each point's highest and second highest output probabilities within a certain user-given threshold in order to identify the boundary-approximate examples. Empirical studies demonstrate that subsets less than a fifth of the size of the original test set can achieve comparable mutation scores while conserving computational resources [2]. By focusing computational resources on critical regions of the latent space, boundary sampling reduces labeling effort and execution costs without sacrificing test effectiveness. The intuition behind this approach is that the decision boundary data points are the most uncertain and sensitive to perturbations caused by the mutations. So, these samples are more likely to expose behavioral divergences in mutants since mutations disproportionately alter decision boundaries instead of samples farther from the decision boundary.

This thesis studies the four above-mentioned mutation analysis acceleration methods in terms of mutation testing cost reduction as well as their cost on mutation score. In our experiments, we use two types of DNN architectures, namely, LeNet-5 [72] and fully-connected neural network (FCNN). LeNet-5 architecture is a representative of the convolutional neural networks (CNNs) with no residual blocks, such as AlexNet [73] and VGGNet [74], while FCNN architecture is a representative of simple neural networks. These model architectures are trained on four popular classification datasets, *i.e.*, MNIST [75], Fashion MNIST [76], Extended MNIST [77], and Kuzushiji MNIST [78], resulting in eight DNN models in total.

We compare the performance of the four mutation analysis acceleration techniques to that of a *vanilla method*, which simply tests all the generated mutants sequentially and exhaustively, in terms of mutation testing cost reduction as well as their cost on mutation score. The results in our dataset of models show that, compared to the vanilla method, the first approach, *i.e.*, neuron clustering, results in an average of 69.77% speedup in mutation analysis at the cost of -26.84% average error in mutation score. Mutant clustering, the second approach, accelerates mutation analysis by 35.31%, an average, with an average mutation score error of only 1.96%. The third approach, random mutation selection, accelerates mutation analysis by an average of 54.07% with an average mutation score error of 1.40%. Meanwhile, boundary sample selection, the fourth and final approach, results in an average of 86.23% speedup in mutation analysis at the cost of 14.10% average error in mutation score.

These results provide empirical evidence that all approaches yield significant gain in mutation testing speed with the second and third approach incurring less error in mutation score than the first and fourth. And in any potential application, it will be a matter of making a trade-off between choosing one method over another.

3.1 Mutation Score Metric Introduction

Applications for mutation analysis have expanded, including to Deep Neural Networks(DNNs) [58]. For DNNs specifically, mutation analysis may be effective for quality assurance [15]. Where the *mutation score* for traditional software resembles $\frac{\text{\# of killed mutants}}{\text{\# of all tested mutants}}$, the formula must change to be applicable and useful for DNNs. The mutation operators that create the mutants

themselves must also change. So, *DeepMAACC* utilizes the definition of *Mutation Score* and some of the model-level mutation operators given by Ma *et al.* [15]. A model-level mutation operator directly changes the weights of an already trained model to create mutants.

For a set of DNN mutants M', if we were to kill a mutant if even one output was different from the test data, there would be a high likelihood that a mutant would be killed for every one of the test data points t from our test suite $t \in T$. The precision of our *mutation score* would suffer if we did not change it to suit the DNNs. Instead, given a set of mutants M' generated from a model M, Ma *et al.* introduce a concept of *killed classes*. For the original model M, we keep the test data points $t \in T$ only if it is correctly handled by M. Correctly handled is defined as the ground truth of a data point being equal to argmax($M(t)$), *i.e.*, the classification output of the DNN model. For a given k-class classifier, we let $C = \{c_1, ..., c_k\}$ be all the k classes given by argmax(t). A killed class can now be defined as when a mutant $m \in M'$ wrongly classifies a test data point $t \in T$, so argmax($m(t)$)$\neq$argmax($M(t)$) where M is the original model. Then, the mutation score can be calculated for a test suite T and a set of DNN mutants M' where for a test data point $t' \in T'$, we say that t' kills $c_i \in C$ of mutant $m' \in M'$ if the following conditions are satisfied: (1) t' is correctly classified as c_i by the original DNN model M, and (2) t' is not classified as c_i by m'. Where KilledClasses(T', m') is the set of classes of m' killed by test data in T': *Definition 3.1 (Mutation Score):*

$$\text{MutationScore(T',M')} = \frac{\Sigma_{m' \in M'} \mid \text{KilledClasses}(T', \text{m'}) \mid}{\mid M' \mid \times \mid C \mid}$$

3.2 DeepMAACC Design

DeepMAACC originally implements two approaches to mutation analysis: (1) neuron clustering approach; (2) mutant clustering approach. Separate from DeepMAACC, but built on top of its system is two more approaches: (3) random mutant selection; (4) boundary sample selection. A fifth approach of vanilla mutation analysis, which involves sequential testing of mutants, is used as a baseline (see 3.2.2 for more details about the vanilla approach). Fig. 2.1 outlines the architecture of DeepMAACC and the steps involved in each of the implemented

approaches. Common to all approaches, including the vanilla baseline, is the set of mutators used to generate the mutants. So, in what follows, we explain the set of DEEPMAACC mutators before explaining the mutation analysis acceleration approaches.

3.2.1 DEEPMAACC Mutators

DEEPMAACC reuses the model-level mutators from DeepMutation [15] with small modifications to eliminate the sources of randomness in them. Removing randomness is essential for enabling us to compare the three approaches in a fair, reproducible manner in section 4.4. We would like to emphasize that DEEPMAACC does not make any assumption as to how the mutants are generated, and any other set of mutators can be used in practice. So, this modification, which is done solely for the purpose of fairness and reproducibility of our experiments, does not have any impact on the applicability of our techniques.

DEEPMAACC creates mutants by applying a set of 3 model-level mutators that directly inject faults into DNN models without a training process. These mutation operators target a single neuron or a neuron cluster. This creates 3 mutants per mutatable neuron/neuron cluster in the model, *i.e.*, a neuron or a neuron cluster that belong to a dense or a convolutional layer. The specific model-level mutators are as follows.

- **Change Weights:** This mutator changes the weights and biases of a neuron/neuron cluster by a user-defined fraction parameter p, which defaults to 0.1. This way, DEEPMAACC mutates the weights and biases that allows proportional changes to each value in the weight vector. If W is the weight and bias vector, the operation is $(1+p)W = W'$ where W' is the new weights or biases.

- **Neuron Effect Block:** This mutator blocks the effect of a neuron/neuron cluster by setting its weights and biases to zero. Therefore, this neuron/neuron cluster will not influence the DNN's classification output. It is similar to deleting the neuron/neuron cluster, as its weights no longer propagate through the rest of the model layers.

- **Neuron Activation Inverse:** This mutator inverses the weights and biases of a neuron/neuron cluster by multiplying the entire weight and bias vector by -1. It changes the

sign out the output value of a neuron/neuron cluster, creating non-linear behaviors that will effect the DNN's classification output.

We have not included DeepMutation's layer-level mutators, and opted to not to study them, as those operators are known to be less effective in practice, and later works [20, 18] did not find them useful. However, having such mutators would not impact the operation of the cost reduction methods in DEEPMAACC, as they are agnostic to the mutator operator.

3.2.2 Baseline Vanilla Approach

The vanilla approach follows the classic methodology of DNN mutation analysis: it is the same approach of mutant generation and testing employed by systems like DeepMutation [15], where the generated mutants are tested sequentially. In this approach, in mutation generation phase, the 3 mutation operators, discussed in section 3.2.1, are applied to each neuron within each mutable layer of the model. This creates 3 different mutants per neuron in each mutable layer. During mutation testing phase, the provided test dataset is used to test the mutants sequentially so as to calculate a mutation score using the Formula 4.1.

3.2.3 Neuron Clustering Approach

The *neuron clustering approach*, outlined in the second row of Fig. 2.1, begins by clustering neurons within each mutable layer based on similarities of the weights and biases between each neuron (the step marked with NC in Fig. 2.1). Agglomerative hierarchical clustering [79] has been used to cluster the neurons. In our implementation, this algorithm uses Euclidean distance [80] of the vectors of the weights and biases of the neurons to calculate the similarities of the neurons. Each cluster is created with a user-specified parameter s that is the average number of neurons to be put into each cluster. For each of the L mutable layers of the model M, we produce $\lceil \frac{n}{s} \rceil$ clusters, where n is the number of neurons within layer L_i. In practice, the number of neurons will vary per cluster, ranging from $[1, n - \lceil \frac{n}{s} \rceil + 1]$, but there will always be $\lceil \frac{n}{s} \rceil$ clusters for each layer. More precisely, DEEPMAACC partitions the neurons per layer into clusters that have similar weights and biases before mutation generation. The neurons per cluster parameter *s* creates clusters with at most *s* neurons inside. Capping the amount of

neurons within the clusters allow the parameter to equally partition clusters over every layer. The amount of clusters will be proportional to the amount of neurons within a single mutable layer. This parameter can generalize throughout the different mutable layers better than the alternative parameter number of clusters. This would create a fixed set of clusters for every layer, dismissing the varying amount of neurons per layer in most models.

To generate mutants, the mutation operators are applied to each of the neuron clusters, thereby mutating all the neurons in each neuron cluster. The motivation behind this approach relies on the assumption that the neurons that have similar weights and biases perform similar tasks in the DNN models [85, 86, 31]. This clustering allows for less mutants to be created and tested, saving on time and storage, because instead of having 3 mutants per neuron in each mutable layer, we will have 3 mutants per neuron cluster in each mutable layer. During mutation testing phase, the given test dataset is test the mutants sequentially so as to calculate the size of *killed classes* for each mutant. These numbers are ultimately used to calculate mutation score.

3.2.4 Mutant Clustering Approach

The *mutant clustering approach*, outlined in the third row of Fig. 2.1, generates the mutants by mutating individual neurons. After creating a set of mutants, this approach of DEEPMAACC clusters the mutants based on the similarity of the mutated locations (the step is marked with MC in Fig. 2.1). Using Euclidean distance as the measure of similarity, DEEPMAACC utilizes ParHAC clustering[37] to cluster the mutants. ParHAC clustering requires a complete undirected weighted graph to function, so DEEPMAACC constructs a graph in the following manner. Each node in this graph denotes a mutant and the weight of each edge is the reciprocal of the Euclidean distance of tuples of the form (l, n, w, b), where l is the layer number, n is the index of the mutated neuron in layer l, w is the mutated weighs inlined in the tuple, and b is the mutated bias. After passing this graph to ParHAC clusterer, together with a user configurable threshold value, p, called linkage threshold [37], it returns the clusters. Linkage threshold represents the minimum similarity required for two clusters to be merged, ranging from 0 (no similarity) to 1 (identical).

During mutation testing, instead of testing all of the mutants sequentially, DEEPMAACC retrieves a random mutant from each cluster that will act as a representative of its respective cluster. It then uses the given dataset to compute the size of the *killed classes* for the representative, which will be reused for all the mutants in the cluster represented by the selected mutant. These numbers are then used to calculate mutation score.

The intuition behind this approach is that mutating locations close to each other in a similar manner is likely to result in the same mutation outcome, so one can reduce the number of mutants by clustering them based on the proximity of the mutation location and the similarities of the mutated weights and biases. This is a well-known method to curtail the costs of mutation analysis in traditional programs [81, 82, 83].

3.2.5 Random Mutant Selection Approach

The *random mutant selection approach*, outlined in the fourth row of Fig. 2.1, starts to generate the mutant set M' by applying the mutation operators to the individual neurons of the original model M. For M, we keep test data points $t \in T$ only if it is correctly handled by M. Then, using the user-given and configurable percentage P, we can create a mutant subset M''. This is completed through *two-round random selection* [84]. This first round happens for every mutable layer l_i in M, the percentage P of mutants within that layer l_i are randomly selected to append to M'' (the step marked with MS in Fig. 2.1). From there, this subset of mutants M'' will be tested with the test suite T' to get the ending mutation score.

Random mutant selection reduces the cost of mutation analysis by sub-sampling mutants while maintaining diagnostic power. So, only testing on a percentage of the mutants will result in speedup. It resulted in an average of 54.07% speedup. The intuition behind this approach comes from the effort to balance representativeness and efficiency. Random mutant sampling prioritizes broad coverage of the model M's structural and functional space. For DNNs, this approach leverages the observation that faults often arise from subtle, distributed interactions between components rather than isolated mutant operator-specific defects [84]. The average mutation score error results were an average of 1.40%.

3.2.6 Boundary Sample Selection Approach

The *boundary sample selection approach*, outlined in the fifth row of Fig. 2.1, begins with the original k-class classifier model M. For M, we keep test data points $t \in T$ only if it is correctly handled by M. The subset test suite T' is then further filtered on the original test suite T to get test suite T''. For every test data point $t \in T$, we perform inference by passing t through the model M to receive its k class label probabilities. The threshold B, a user configurable threshold value, is then used to identify samples close to the decision boundary (the step marked with BSS in Fig. 2.1). For each t, the ratio of the highest class probability y_{max} to the second-highest y_{secmax} is computed to compare to this threshold B. Samples satisfying $y_{max}/y_{secmax} < B$ are selected for the subset test suite T''. This criteria captures instances where the model's confidence is ambiguous, as high ratios indicate clear classifications resistant to mutation-induced perturbations. Moving forward, the boundary sample approach generates mutants by mutating individual neurons from the model M to get the mutant set M'. Then, test suite T'' is used to test all of the mutants in M', resulting in its ending mutation score.

This minimal test suite T' allows an average speedup of 86.23% across all of its tested models with an average mutation score error of 14.10%. This approach and its metrics are essentially ported over from Shen *et al.* [2]. The intuition behind this approach is rooted in the geometric and probabilistic properties of decision boundaries, which define regions where model predictions transition between classes. Samples near the decision boundary represent classification edge cases where the model exhibits lower confidence, meaning the top two class probabilities are nearly equal or reasonably similar. So, smaller thresholds, *e.g.* 10, will result in a test suite containing samples closer to the decision boundary. By prioritizing these samples, boundary sampling selection amplifies the likelihood of detecting subtle behavioral divergences between the original mutated models.

3.3 Overall Performance

DEEPMAACC has run on all four approaches, as well as the vanilla baseline. Then, DEEPMAACC has been evaluated on eight DNN models across four popular classification datasets

and two DNN architectures. When compared to exhaustive, or vanilla, mutation analysis, the results provide empirical evidence that neuron clustering approach, on average, accelerates mutation analysis by 69.77%, with an average -26.84% error in mutation score. Meanwhile, mutant clustering approach, on average, accelerates mutation analysis by 35.31%, with an average 1.96% error in mutation score. The third approach, random mutation selection, accelerates mutation analysis by an average of 54.07% with an average mutation score error of 1.40%. Meanwhile, boundary sample selection, the fourth and final approach, results in an average of 86.23% speed-up in mutation analysis at the cost of 14.10% average error in mutation score. Our results demonstrate that a trade-off can be made between mutation testing speed and mutation score error.

Chapter 4

Evaluation

In this section, we apply DEEPMAACC on a set of DNN models to answer the following research questions (RQs) that were introduced in section 1.3.

- **RQ1:** How much *speedup* is gained when using Neuron Clustering or Mutant Clustering versus Vanilla mutation testing?
- **RQ2:** How much *mutation score* is lost when using Neuron Clustering or Mutant Clustering versus Vanilla mutation testing?
- **RQ3:** What is the impact of non-determinism in the training process on the mutation testing *speedup* and *mutation score* of DEEPMAACC when using Neuron Clustering or Mutant Clustering versus Vanilla mutation testing?
- **RQ4:** How much *speedup* is gained when using Random Mutant Selection or Boundary Sample Size Selection versus Vanilla mutation testing?
- **RQ5:** How much *mutation score* is lost when using Random Mutant Selection or Boundary Sample Size Selection versus Vanilla mutation testing?
- **RQ6:** By what percentages is the testing data set reduced when generated by Boundary Sample Size Selection? Are they representative to evaluate the results of mutation testing under the whole test data? Are they more efficient subsets?

To study the two clustering approaches of DEEPMAACC, the values of their two respective parameters, namely neurons per cluster s and ParHAC linkage threshold p, are varied. For the

Table 4.1: Dataset of DNN models used in our experiments*

Arch.	Dataset	Model Size	Train #	Test #	Classes	Test Acc.
FCNN	EMNIST	480,383	124,800	20,800	26	0.8698
FCNN	FMNIST	477,782	60,000	10,000	10	0.8744
FCNN	KMNIST	477,782	60,000	10,000	10	0.8678
FCNN	MNIST	477,782	60,000	10,000	10	0.9661
LeNet-5	EMNIST	190,355	124,800	20,800	26	0.9195
LeNet-5	FMNIST	186,020	60,000	10,000	10	0.9026
LeNet-5	KMNIST	186,020	60,000	10,000	10	0.9495
LeNet-5	MNIST	186,020	60,000	10,000	10	0.9871

* Train # and Test # represent the number of data points in the train and test datasets, respectively, and Classes denotes the number of classes/labels in the test dataset.

neuron clustering approach, s is ranged over the sequence of numbers from 1 to 10, creating mutants of the neuron clusters of size 1 to 10. For the mutant clustering approach, p is ranged over 0.1 to 0.99 to account for different linkage thresholds.

To study the two approaches added onto DEEPMAACC, their respective parameters, selection fraction f and BSS threshold b, are varied over 10 different values. For the random mutant selection approach, f is ranged from 0.05 to 0.99, which essentially means testing 5% of the mutants and 99% of the mutants. For the boundary sample selection approach, b is ranged from 1.5 to 1000, testing varying distances from the decision boundary.

DEEPMAACC is ran on a set of DNN models for each parameter. The reader is referred to section 4.3.4 for more details about our methodology.

4.1 Experiment Execution Environment

DEEPMAACC experiments are conducted on two identical Dell Precision workstations with AMD Ryzen Threadripper @ 2.7 GHz CPU, 1 TB of RAM, and two NVIDIA RTX A6000 GPUs. These two machines run Ubuntu 22.04.4 LTS. DEEPMAACC is written in Python 3.9 and is based on Keras 3.4.1, with TensorFlow 2.16.2 backend.

4.2 Datasets of DNN Models

Two types of DNN architectures are used to evaluate DEEPMAACC: fully-connected neural networks (FCNN) and LeNet-5 [72]. There are a total of 4 FCNN models and 4 LeNet-5 models that are trained on the well-known classification datasets MNIST [72], Extended MNIST [77] (EMNIST), Fashion MNIST [76] (FMNIST), and Kuzushiji MNIST [78] (KMNIST). Datasets MNIST, FMNIST, and KMNIST represent 10-class classification problems, while EMNIST is for a 26-class classification problem.

The MNIST dataset is designed for handwritten image recognition, containing 60,000 training data points and 10,000 test data points of handwritten digits. The 70,000 total data points are all within 10 classes, which are the digits 0 to 9, grayscale, of size 28 pixels by 28 pixels [72]. The FMNIST (Fashion MNIST) dataset is the same size, consisting of 60,000 training data points and 10,000 test data points. However, these images provide a more complex classification task of labeling 10 categories of fashion items, also grayscale and of size 28 pixels by 28 pixels [76]. The KMNIST (Kuzushiji MNIST) dataset consists of 60,000 training data points and 10,000 test data points, where Kuzushiji is a Japanese cursive writing style. The 10 labels for this dataset are 10 different characters from this writing style, still grayscale 28 pixels by 28 pixels [78]. The EMNIST (Extended MNIST) dataset is an extension of the MNIST dataset, containing handwritten characters including digits and letters [77]. Contrastingly from the other MNIST datasets, this dataset is much larger and has 26 different classes. It is made up of 124,800 training data points and 20,800 test data points. Each of the 26 classes represents a letter of the alphabet which are grayscale and of size 28 pixels by 28 pixels, as well.

The architecture of the FCNN model consists of an input layer, a flatten layer, 3 hidden layers each with 50 ReLU-based neurons, and an output layer with softmax activation. The LeNet-5 architecture consists of an input layer, 2 convolution layers and 3 fully-connected layers with tanh activation, and a softmax output layer.

Table 4.1 presents more details about our dataset. For the varying architectures paired with these four datasets, the respective test accuracy is presented in the right-most column. The testing accuracy is the percentage of correctly labeled test data points from the original dataset.

Since the model only correctly labels these percentage amounts, this is the percentage of the test data points that we use to test our mutants, as we do not use incorrectly labeled points as specified in section 4.3.1.

4.3 Measures

4.3.1 Mutation Score

To study the impact of the mutation testing acceleration techniques on the mutation score, we use the mutation score calculation formula for DNN classifiers introduced by Ma *et al.* [15]. Assume that m is a k-class classifier and let $C = \{c_1, \ldots, c_k\}$ be all the k classes of the test dataset T, *i.e.*, $C = \{\text{argmax}(t) | t \in T\}$. Given a test data point $t \in T$, t is said to kill the class $c \in C$ of a mutant m', if t is correctly classified as c by the original model m, yet it is not classified as c by m'. Now for a set of mutants M', the mutation score is defined to be:

$$\frac{\Sigma_{m' \in M'} |\text{KilledClasses}(T, m')|}{|M'| \times |C|}, \tag{4.1}$$

where KilledClasses denotes the set of classes of m' killed by test data in T.

4.3.2 Mutation Testing Speedup

Speedup, *i.e.* the amount of acceleration, is calculated by dividing the difference between the average vanilla mutation testing time $t_{\text{v-avg}}$ and the average mutation testing time for neuron/mutant clustering approach, *i.e.*, one of the two approaches implemented by DEEPMAACC, t by the average vanilla mutation testing time $t_{\text{v-avg}}$. It represents the percentage of reduction in mutation testing time due to the use of any of DEEPMAACC's mutation testing cost reduction approaches *vs.* the vanilla approach.

$$S = \frac{t_{\text{v-avg}} - t}{t_{\text{v-avg}}} \tag{4.2}$$

All mutation testing times, *i.e.* the time needed for testing a set of generated mutants, are measured in seconds.

4.3.3 Mutation Score Error (Loss)

Mutation score error, or loss, that is calculated based on the Formula 4.1. This is the percentage of deviation of DEEPMAACC mutation score from vanilla mutation score, and is calculated by dividing the difference between the average vanilla mutation testing score $s_{\text{v-avg}}$ and the average neuron/mutant clustering-based mutation testing score s by the average vanilla mutation testing score $s_{\text{v-avg}}$.

$$L = \frac{s_{\text{v-avg}} - s}{s_{\text{v-avg}}} \tag{4.3}$$

4.3.4 Method

All of the five approaches studied in this paper, namely neuron clustering, mutant clustering, random mutant selection, boundary sample selection, and vanilla, have been executed multiple time per model in order to account for randomness and to align with the many executions of the four approaches due to their various parameter values.

Our neuron clustering approach is executed 6 times per a parameter value s (explained in 3.2.3), which ranges over the set {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, while the mutant clustering approach executes 6 times per a ParHAC threshold value p (explained in 3.2.4), which ranges over the set {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99}. The number of neurons per cluster ranges from $(0, \infty)$, but seeing that there are reasonable results with the values ranging from 1 to 10, and the mutation score error continually, but slowly increasing while the speedup plateaus, we see that there is no need to extend testing with that parameter. So, these two sets of 10 parameters align nicely. ParHAC threshold can only range from [0, 1.0], but we go from 0.1 to 0.99 because we do not need to test where the clusters need no similarity or where they have full similarity, *i.e.*, they are identical.

The random mutant selection approach is executed 6 times per a fraction parameter f (explained in 3.2.5), which ranges over the set {0.05, 0.1, 0.15, 0.2, 0.25, 0.5, 0.75, 0.8, 0.9, 0.99}. while the boundary sample selection approach executes 6 times per a BSS threshold value b (explained in 3.2.6), which ranges over the set {1.5, 2, 5, 8, 10, 12, 15, 20, 100,

1000}). The tested fraction parameters can range from $[0, 1]$ as they must represent a part of the test suite. While the fraction f may be 0 or 1.0, there is no reason to test these values since they give no new insight since 0 allows for nothing to be evaluated and 1.0 runs BSS the same as vanilla would. The Boundary Sample Selection thresholds range from $(0, \infty)$, but the BSS threshold of 10 provides a practical trade-off between sensitivity and efficiency [2]. Therefore, we only test surrounding values as these all have the potential to balance the subset size and representativeness of the original test data set.

These ten parameter values used to test the mutant clustering approach, neuron clustering approach, random mutant selection approach, and boundary sample selection all align with the ten parameters of the other approaches, allowing for a meaningful side-by-side comparison. While these parameters are used in the x-axes of the plots, the y-axes consist of *Speedup* and *Mutation Score Error*.

Overall, there are 2,160 executions of DEEPMAACC that we collected data from, resulting from 270 executions for each model: 30 runs from vanilla approach, 6 executions for each of the 10 parameters for neuron clustering approach, 6 executions for each of the 10 parameters for mutant clustering approach, 6 executions for each of the 10 parameters for random mutant selection approach, and 6 executions for each of the 10 parameters for boundary sample selection approach.

4.3.5 Results

This section provides insight into the results of all experiments. We use Fig. 4.1, Fig. 5.1, Fig. 5.2, Fig. 5.3, and Fig. 5.4 to help answer the multiple research questions. This section will explain how to interpret each figure. Section 5 will provide further insight by elaborating on the results.

4.4 Experiments

Fig. 4.1 plots the average number of mutants tested for each parameter for both the FCNN and LeNet-5 models using Neuron Clustering or Mutant Clustering. This plot shows the difference

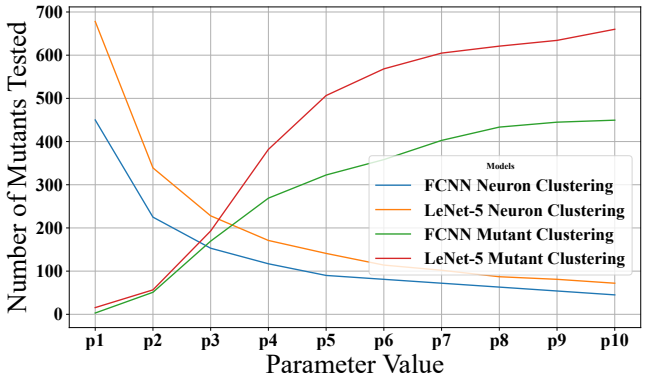

Figure 4.1: *Number of Mutants Tested vs. Parameter Value* for the two different approaches used by DEEPMAACC. Each of the four lines represent the average *Number of Mutants Tested* for a different DNN architecture and clustering approach. The parameter for Neuron Clustering is *Neurons per Cluster*, having the values of {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}. The parameter used for Mutant Clustering is *ParHAC Threshold* having the values of {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99}.

in number of tested mutants, which can help explain the behavior and use of the different parameters. For Neuron Clustering, the *Number of Mutants Tested* decreases as the parameter increases. This is expected, as the more Neurons per Cluster there are, the less clusters are required to hold all neurons. Larger clusters results in less clusters and therefore fewer mutants as there are 3 mutants created for each cluster. For Mutant Clustering, the *Number of Mutants Tested* increases as the parameter increases. This is expected, as the similarity threshold increases, the ParHAC clusterer requires neurons to hold more similar weights to cluster them. A small similarity threshold will create large clusters, resulting in less mutants to test. A large similarity threshold requires the neurons to be nearly identical, which results in smaller clusters, increasing the number of mutants to be tested. These patterns also show up in later figures, so this explanation seems to remain consistent. Random Mutant Selection and Boundary Sample Selection are not compared along with the other approaches in this plot because of their uninsightful behaviors. Random Mutant Selection reduces the *Number of Mutants Tested* directly by the parameter amount, a fraction, so it would show a linear trajectory. However, the parameter values do not have equal distances between them all - {0.05, 0.1, 0.15, 0.2, 0.25, 0.5, 0.75, 0.8, 0.9, 0.99} - unlike the two clustering approaches. So, the plotted line would be more jagged and would not be representative of the approach's true behavior. The Boundary Sample

Selection approach is not compared simply because BSS does not reduce mutants tested, only the size of the testing suite. So, the lines for both FCNN and LeNet-5 would be straight lines with slopes of 0 at the amount of mutants in the original mutant set.

Fig. 5.1 shows the *Speedup* (Column 1) and *Mutation Score Error* (Column 2) of our different models and clustering approaches over multiple executions as described in section 4.3.4. We use the four FCNN models and four LeNet-5 models described earlier in this section. The plots in this figure are separated by DNN architecture type, FCNN (Row 1 and 3) and LeNet-5 (Row 2 and 4). Each of the colored lines in each plot correspond to different models. Blue lines correspond to a model (FCNN or LeNet-5) trained on the EMNIST dataset, yellow corresponds to a model trained on FMNIST, green corresponds to a model trained on KMNIST, and red corresponds to a model trained on MNIST.

Fig. 5.2 shows the *Speedup* for both clustering approaches across the 8 models represented by the 8 plots. Each parameter has a pair of box-and-whisker plots. The blue box-and-whisker plot on the left represents neuron clustering *Speedup* while the red box-and-whisker plot on the right represents mutant clustering *Speedup*. So, the parameters on the y-axes for neuron clustering is *Neurons per Cluster* and *ParHAC Threshold* for mutant clustering. The annotation above each pair is the p-value obtained via Mann-Whitney U-Test. In column one, from top to bottom, the models represented are FCNN-EMNIST, FCNN-FMNIST, LeNet-5-EMNIST, and LeNet-5-FMNIST. In column two, from top to bottom, the models represented are FCNN-KMNIST, FCNN-MNIST, LeNet-5-KMNIST, LeNet-5-MNIST.

Fig. 5.3 shows the average *Speedup* (Column 1) and average *Mutation Score Error* (Column 2) of our different models for the Random Mutant Selection approach over multiple executions as described in section 4.3.4. We use the four FCNN and four LeNet-5 models described in section 4.2. The four plots in this plot are separated by DNN architecture type, FCNN (Row 1) and LeNet-5 (Row 2). Each of the colored lines in each plot correspond to different models. Blue lines correspond to a model (FCNN or LeNet-5) trained on the EMNIST dataset, yellow corresponds to a model trained on FMNIST, green corresponds to a model trained on KMNIST, and red corresponds to a model trained on MNIST.

Fig. 5.4 shows the average *BSS Size Percentage* (Row 1), the average *Mutation Score Error* (Row 2), and the average *Speedup* (Row 3) of our different models for the Boundary Sample Selection approach over multiple executions as described in section 4.3.4. *BSS Size Percentage* on the y-axis is the percentage of the original test data set left after the BSS test suite filter is used. This metric shows the percentage perspective of how many test data points are decision boundary points according to the *BSS Threshold* parameter for each data set. However, the test data sets are first filtered on whether the original model correctly labels the test data point, which is why the two plots presenting *BSS Size Percentage* data for the FCNN and LeNet-5 models show different behavior. It is a two step-filter. We use the four FCNN and four LeNet-5 models described in section 4.2. The four plots in this plot are separated by DNN architecture type, FCNN (Column 1) and LeNet-5 (Column 2). Each of the colored lines in each plot correspond to different models. Blue lines correspond to a model (FCNN or LeNet-5) trained on the EMNIST dataset, yellow corresponds to a model trained on FMNIST, green corresponds to a model trained on KMNIST, and red corresponds to a model trained on MNIST.

Various configurations of these plots are available in Section A of the Appendix to facilitate different comparisons between the different approaches and their metrics.

Chapter 5

Discussion

5.1 Answering Research Question 1

This research question investigates the effectiveness in *Speedup* of DEEPMAACC through varying parameter values and models for each clustering approach of the mutation testing. We will answer this for our neuron clustering approach and our mutant clustering approach separately.

In the first column of Fig. 5.1, we will observe the first two rows of plots to answer this question for neuron clustering. In these two plots, it is shown that the *Speedup* for all the models is nearly identical at the separate parameter values. The *Speedup* does continually increase in a non-linear fashion, where an increase in *Speedup* indicates a decrease in mutation testing time. A value of 1 neuron per cluster causes the DEEPMAACC mutation analysis to operate similarly to the vanilla approach because each neuron will be in a cluster on its own. So, the same number of mutants will be created. This leads to the initial *Speedup* value of 0.0 since the vanilla and neuron clustering approaches are the two being compared. The graph suggests that the *Speedup* will plateau at a value of 1.0. We can explain the similarity in the graphs for each model across both DNN architectures using Fig. 4.1 because the reduction in tested mutants should be the same for each parameter. We can see the plotted lines for FCNN Neuron Clustering and LeNet-5 Neuron Clustering have similar slopes, but differ only in height, which is explained by the larger number of mutable neurons in a LeNet-5 model. Overall, we can observe a trend that neuron clustering increases the *Speedup* for the mutation testing of a model

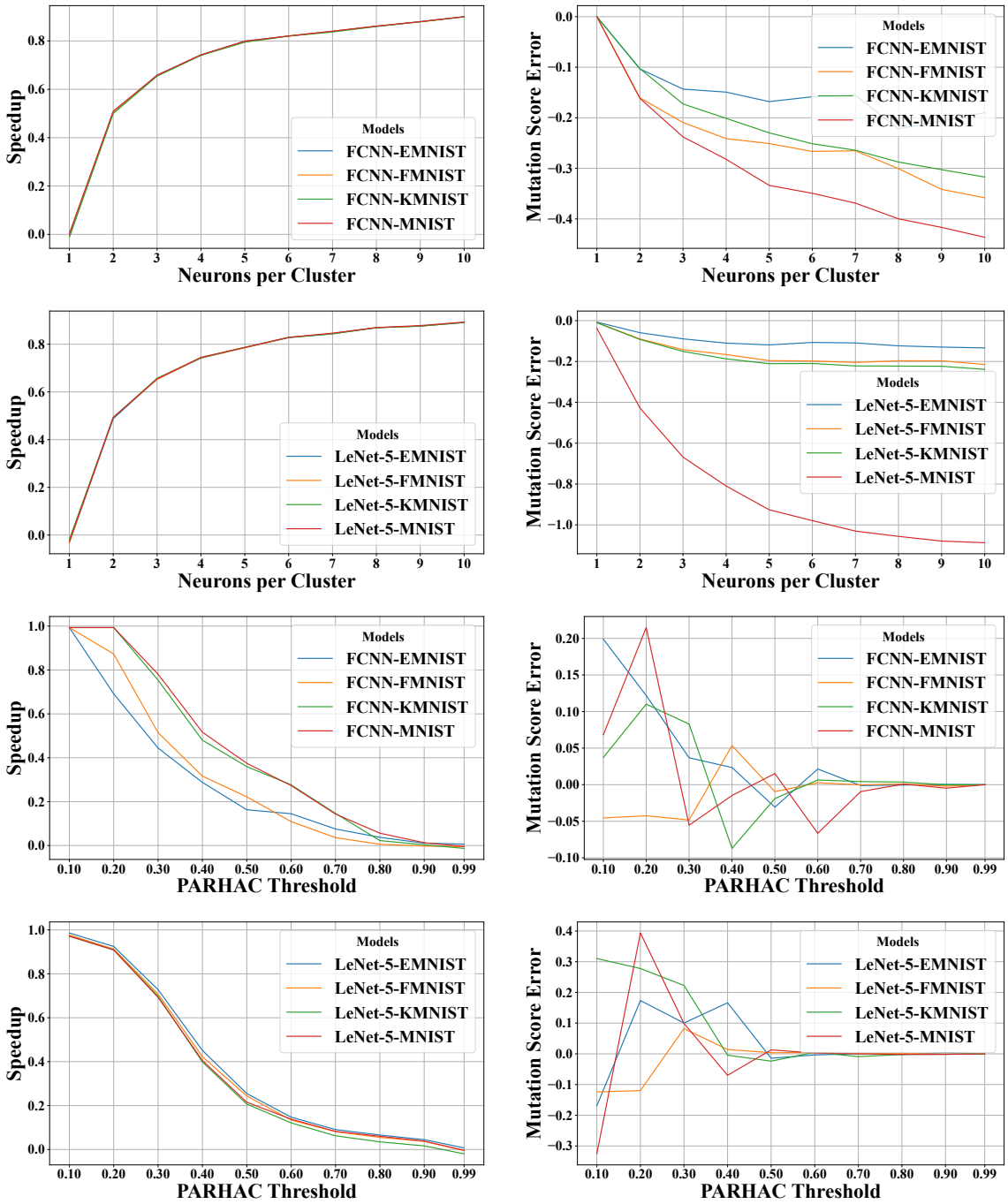

Figure 5.1: From the left to right: Column 1 plots the Speedup, while Column 2 shows the Mutation Score Error metric. From the top to bottom: Rows 1 and 2 reports the results of the Neuron Clustering Approach for FCNN and LeNet-5 models, respectively, and Rows 3 and 4 present the results of the Mutant Clustering Approach for FCNN and LeNet-5 models, respectively.

as the parameter increases. FCNN models obtain an average 69.95% *Speedup* while LeNet-5 models obtain an average 69.59% *Speedup*.

In the first column of Fig. 5.1, we will use the last two rows of plots to answer this question for mutant clustering. In the top plot, there is some deviation in the exact plotted lines, but the overall trend is similar. We observe an opposite trend than that of the neuron clustering approach, which can be explained by observing Fig. 4.1. In that figure, there is a positive slope that is similar to the inverse of the plot we are looking at in Fig. 5.1. Testing more mutants takes a longer time, so as the number of mutants tested increases, the *Speedup* decreases. However, the line in Fig. 4.1 is the average of all FCNN models, so the deviation in *Speedup* for Fig. 5.1 is likely explained by a deviation in size of mutant clusters, which would be due to the inherent behavior of ParHAC clustering. In the second plot, the plotted lines are almost identical, suggesting that there is less deviation in the number of clusters for this model type. The four different model's plotted lines have a negative trend, decreasing as the parameter increases. When the *ParHAC Threshold* reaches 0.99, DEEPMAACC's mutant clustering approach should cause the mutation testing to operate similarly to the vanilla approach. The *Speedup* approaches a value of 0.0 since 0.99 is approaching 1.0. FCNN models obtain an average 35.19% *Speedup* while LeNet-5 models obtain an average 35.43% *Speedup.*

5.2 Answering Research Question 2

This research question investigates the effectiveness in Mutation Score Error of DEEPMAACC through varying parameter values and models for each approach. We will answer this for our neuron clustering approach and our mutant clustering approach separately.

For neuron clustering, we test 10 different *Neurons per Cluster* values for the number of neurons in each cluster n to see the effects on mutation testing time. In Fig. 5.1, from the second column we will use the first two rows of plots to answer this question. The first plot shows only FCNN models while the second plot shows LeNet-5 models. In both plots, we can observe the general negative trend of *Mutation Score Error* as the *Neurons per Cluster* increase. A negative *Mutation Score Error* means that the mutation scores resulting from the DEEPMAACC neuron clustering approach are higher than that of the vanilla approach. As a model's mutation score approaches 1, more classes are killed by each mutant. Ideally, the *Mutation Score Error* will

near 0, as we do not want the mutation analysis results to change at all, as a larger change in mutation score indicates a larger change in model behavior.

DEEPMAACC's goal is to accelerate mutation analysis without altering the performance. There is an obvious outlier for LeNet-5-MNIST in the second plot. The reason for this large jump in error will need to be explored in future works, as the behavior does not reflect any other part of the analysis. However, it seems that the LeNet-5-MNIST and FCNN-MNIST both have worse trends than these models trained on the datasets FMNIST, KMNIST, and EMNIST. Having that worse trend may align with the MNIST dataset being simpler than the other tested datasets. So, this trend shows up for both the FCNN-MNIST and LeNet-5-MNIST models for neuron clustering, which could suggest that the neuron clustering method works better the more complex the model it. In all other models across both plots, the declining trend is similar, with deviations around 0.1 as the parameter values increase. This trend is expected, as the clusters grow larger, more neurons are mutated during the production of one mutant, understandably producing a larger *Mutation Score Error*. As mentioned before, the *Mutation Score Error* would near 0.0 because these neurons have been clustered with the idea that they control the same behaviors of the model. So, ideally, no matter the amount of neurons we mutate, we are altering the same behavior and the same classes would be killed for each cluster as an individual neuron mutant would kill. Overall, we can observe a trend that neuron clustering increases the *Mutation Score Error* in the negative direction for the mutation testing of a model as the *Neurons per Cluster* increase.

For mutant clustering, we test 10 different parameter values for the ParHAC linkage threshold used to cluster mutants to see the effects on mutation testing time. In the second column, we will use the last two rows of plots to answer this question. In both plots, we witness some deviation in the beginning *ParHAC Thresholds*. This is due to the randomness of the mutant cluster representative choice. In Fig. 4.1 we can see that there is a small number of mutants tested in the beginning *ParHAC Thresholds*, which means there were large clusters. When a random representative is chosen from a large cluster, it is reasonable that the mutation score error has high deviations. DEEPMAACC strives to group behaviorally similar mutants

in the same cluster, but small similarity values will cause groups of differently behaving clusters. As the *ParHAC Threshold* increases, we can see that the *Mutation Score Error* stabilizes. For LeNet-5 models, there is less deviation for the different models, and the *Mutation Score Error* seems to stabilize earlier. The FCNN models have more deviation and stabilize at a later *Mutation Score Error*.

With a goal of mutation testing acceleration, we can observe that by applying neuron clustering to FCNN models, we obtain an average of 69.95% *Speedup* with an average -22.52% *Mutation Score Error*. LeNet-5 models have an average of 69.59% *Speedup* with an average -31.17% *Mutation Score Error*. Applying mutant clustering to mutation analysis for FCNN models can produce an average *Speedup* of 35.19% with an average 1.41% *Mutation Score Error* and an average of 35.43% *Speedup* with an average 2.50% *Mutation Score Error* when applied to LeNet-5 models. For the mutation clustering approach, the deviations cancel out, causing the average *Mutation Score Error* to be minimal.

5.3 Answering Research Question 3

This research question investigates the effects of non-determinism in the performance of DEEPMAACC for *Speedup* and *Mutation Score Error*. Non-determinism is present in the form of model training randomness, killed classes randomness, and the random choice of a cluster representative in mutation clustering in DEEPMAACC's executions. Multiple executions were run for each model to mitigate some of these non-deterministic effects.

We can observe that the *Speedup* is not random by assesing Fig. 5.2. These box-and-whisker plots show DEEPMAACC's *Speedup* performance and deviations through varying parameter values and models for each approach. The box-and-whisker plot pairs show the mutation testing *Speedup* for the neuron clustering and mutation clustering approaches. The *Speedup* is shown to have opposite slopes for the two different clustering approaches, suggesting that they are nearly independent of a possible random chance to obtain these values. To confirm this visual assessment, a Mann-Whitney U-Test [87] has been conducted over all the pairs of series of values for the different box plots, with the null hypothesis being that there is no statistically significant difference between the speedup values of neuron clustering vs

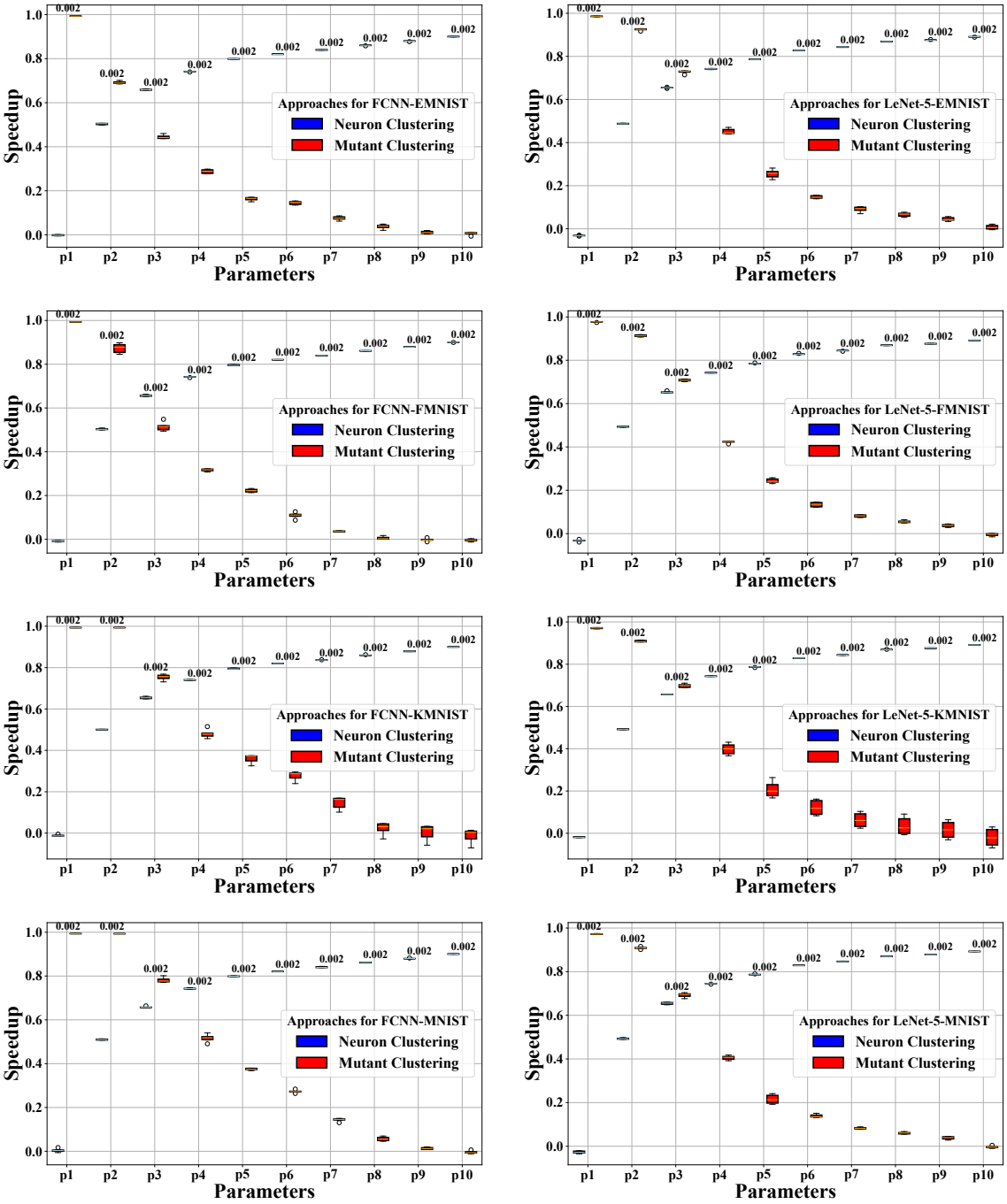

Figure 5.2: Box plot pairs for mutation testing Speedup for the 8 different models. In each pair, blue box-and-whisker, on the left, represents speedup for Neuron Clustering Approach, and the red box-and-whisker, on the right, represents that of Mutant Clustering Approach. Each pair of box-and-whiskers in each plot is annotated with the p-value obtained *via* Mann-Whitney U-Test. The parameter used for Neuron Clustering is *Neurons per Cluster*, ranging over {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, while the parameter used for Mutant Clustering is *linkage threshold*, ranging over {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99}. In column one, from top to bottom, the model's *SpeedUp* that is shown are FCNN-EMNIST, FCNN-FMNIST, FCNN-KMNIST, FCNN-MNIST, respectively. In column two, from top to bottom, the model's *SpeedUp* that is reported is LeNet-5-EMNIST, LeNet-5-FMNIST, LeNet-5-KMNIST, LeNet-5-MNIST, respectively.

mutant clustering. The alternate hypothesis is that the difference in the values are statistically significant. The p-values obtained from the test are shown as annotations above the pairs of box-and-whiskers in Fig. 5.2. As we can see, almost all of the values are lower than 0.05, rejecting the null-hypothesis with a 95% confidence. From this observation, we can conclude that DEEPMAACC tends to produce a similar or identical amount of mutants through every execution for a single parameter.

To observe the randomness in *Mutation Score Error*, we can view column 2 Fig. 5.1. For neuron clustering in the top two plots of the right column, we can observe non-determinism's presence in the slight deviations of the graph. The jagged output could also be an inherent characteristic of the models, but we can assume it is a result of randomness present in the training of the model and the model's classification of the different classes. However, in mutant clustering, seen in the bottom two plots of the right column, we can observe a more harsh result of randomness for the smaller parameter values. This large random effect results from the mutant clustering's random approach to selecting a cluster representative. When a large cluster is created from a small similarity threshold, the grouped mutants will not all behave as similar as wanted. DEEPMAACC strives to group behaviorally similar mutants in the same cluster, so that when a representative of the mutant is killed (or survived), all the mutants within that cluster can be safely marked as killed (or survived) without explicitly testing them. As the *ParHAC Threshold* increases, the *Mutation Score Error* lines stabilize, meaning that DEEPMAACC is no longer choosing random representatives that do not successfully represent the entire cluster. This suggests that these *ParHAC Threshold* values produce quality clusters that behave similarly. Utilizing a small similarity threshold produces unwanted outcomes based on these plots and explanations. As the model is suggestedly random, the randomness cancels itself out when calculating the mean *Mutation Score Error* for the DNN architectures' plots. For mutant clustering on FCNN models, the average *Mutation Score Error* is 1.41% while the average is 2.50% for LeNet-5 models.

Overall, it is seen that neuron clustering obtains larger *Speedup* values at the cost of a larger *Mutation Score Error*. Mutant clustering obtains lesser *Speedup* values, but nearly mitigates

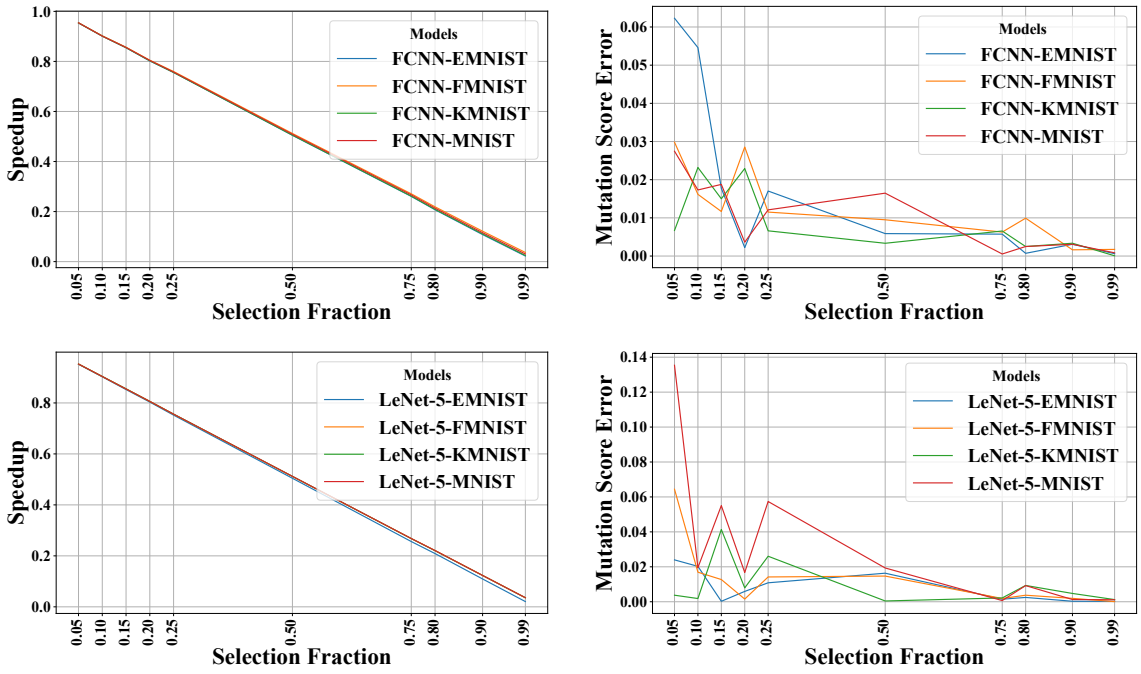

Figure 5.3: Random Mutant Selection Approach's Speedup and Mutation Score Error. From the left to right: Column 1 plots the Speedup, while Column 2 shows the Mutation Score Error metric. From the top to bottom: Row 1 reports the results for FCNN models. Row 2 reports the results for LeNet-5 models.

the *Mutation Score Error*, suggesting that this method chooses quality clusters that behave similarly.

5.4 Answering Research Question 4

This research question investigates the effectiveness in *Speedup* of DEEPMAACC through varying parameter values and models for the two approaches: Random Mutant Selection and Boundary Sample Size Selection. This question will specifically be answered for Random Mutant Selection first and then Boundary Sample Size Selection afterwards.

Figure 5.3 holds the *Speedup* and *Mutation Score Error* for Random Mutant Selection, so the first column of plots will be observed only to answer this question. In these two plots, it is shown that the *Speedup* for all FCNN and LeNet-5 models are nearly identical throughout the parameter values, decreasing linearly. Random Selection Fraction selects a random subset of mutants from each layer based on a user-given *Selection Fraction*. The respective parameter values and *Speedup* values should be inversely related, as the amount of *Speedup* should come

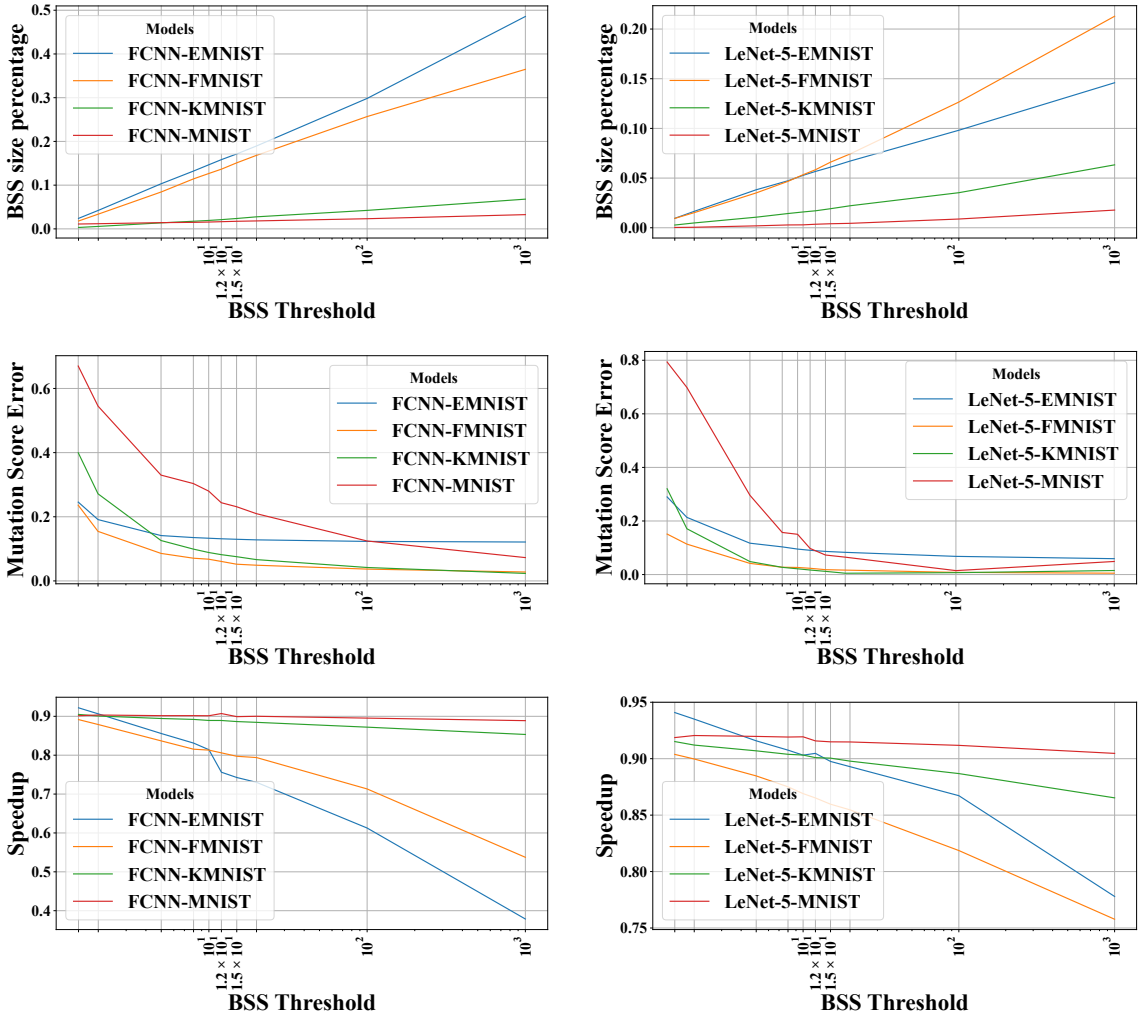

Figure 5.4: Boundary Sample Size Selection Approach's BSS Size Percentage, Speedup, and Mutation Score Error. From the left to right: Column 1 plots the FCNN models, while Column 2 shows the LeNet-5 models. From the top to bottom: Row 1 reports the BSS Size Percentages for both models. Row 2 reports the Mutation Score Error for both models. Row 3 shows the Speedup results for both models.

directly from the reduction of mutants tested. For the *Selection Fraction* value 0.20, it can be seen within the plotted line on either the FCNN model lines or LeNet-5 model lines, that the *Speedup* values is 0.8. 0.8 would come from 1.0 subtracting 0.20, our *Selection Fraction*. This behavior is seen throughout the entirety of both plots. So, the *Speedup* value for the Random Mutant Selection is evidently related directly to the *Selection Fraction* value since that percentage of mutants is removed from the tested mutants. The FCNN models obtain an average 54.00% *Speedup* while LeNet-5 models obtain an average 54.15% *Speedup*.

Figure 5.4 presents six plots in a different format from the usual plots. These six plots show the *Speedup*, *Mutation Score Error*, and *BSS Size Percentage* in rows instead of columns. So, to consider the *Speedup* performance, we will observe the bottom row, where the left plot holds the FCNN model data and the right plot holds the LeNet-5 model data. This strategy of acceleration is also different because it does not reduce mutants tested, like the rest of the approaches, but reduces the size of the test suite. A smaller test suite results in *Speedup* since inference must be run for all test data points in order to calculate mutation score. All datasets present a negative trajectory of *Speedup* within both plots. The models lose *Speedup* as the *BSS Threshold* increases. A smaller *BSS Threshold* will require a smaller ratio of the test datapoint's largest class probability and second largest class probability, which means the data points are near the decision boundary. FMNIST and EMNIST are both more complex classification datasets because FMNIST's classes are fashion items and EMNIST's classes are the 26 alphabet characters while MNIST and KMNIST are just 10 numbers/characters. The decision boundaries of MNIST and KMNIST appear to be closer to each other, explained by the unchanging *Speedup* values. The closeness may present because of the similarity between the different number and character classes, as there are only 10 different ones and both dataset's characters are made up of a combination of simple lines that cross over the same spaces. The lines of the FCNN-MNIST and FCNN-KMNIST have a slope of nearly 0, starting at a speedup around 90% while the lines of LeNet-5-MNIST and LeNet-5-KMNIST start at around 92% *Speedup* ending at approximately 91% and 87%, respectively. These models begin with such a large *Speedup* and end with a similarly large value because almost all the test suite points were near the decision boundary. The other two datasets, KMNIST and EMNIST have their more

distinct classes with seemingly mostly more distinct decision boundaries observed through the similar negative trajectories. The lines of FCNN-FMNNIST, FCNN-EMNIST, LeNet-5-EMNIST, and LeNet-5-FMNIST all start at a *Speedup* of 90+%, but then the FCNN models end around 40-55% while the LeNet-5 models end around 76-78%. These models starting at 90+% but decreasing with later parameters means their test suites are not all close to the decision boundaries. They are all varying distances away, meaning the models are more confident about those labels. Complex datasets would most likely have similar trajectories in *Speedup* to MNIST and KMNIST. Even at the largest parameter, allowing for test data points farther from decision boundaries and potentially being less effective within the mutation score, these models can achieve around 40-90% or 76-90% *Speedups* for FCNN and LeNet-5. However, on average, FCNN models obtain 83.25% *Speedup* while LeNet-5 models obtain 89.21% *Speedup*.

5.5 Answering Research Question 5

This research question investigates the effectiveness in *Mutation Score Error* of DEEPMAACC through varying parameter values and models for each approach. We will answer this for our Random Mutant Selection approach and Boundary Sample Size approach separately within this section.

For the Random Mutant Selection approach, we test 10 different *Selection Fraction*s to see the effects on mutation testing time. We will use the right column in Fig 5.3 to answer this question. The top plot shows the FCNN models while the bottom shows the LeNet-5 models. Both plots show similar slightly negative trends, but consistently surrounding the 0% *Mutation Score Error*. Ideally, the *Mutation Score Error* will be 0.00, since this represents a possibly equivalent mutant set. Across the entire plot for FCNN models, the *Mutation Score Error* stays around a minimal percentage value, which is a very good performance. As the *Selection Fraction* values approach 0.99, the *Mutation Score Error* also approaches 0.00 as 99% of the mutants is essentially equivalent to the original mutant set. There is some randomness presenting itself for the beginning values of *Selection Fraction* through the extremely jagged lines, which is unpredictable behavior. This approach randomly selects a portion of mutants from each layer. Since it selects randomly an equal amount across each layer, I believe this is

why we see a *Mutation Score Error* that dances around 0.01 and 0.03 for many of the values, ultimately ending at a value barely above 0.00. Ensuring the mutants are selected from all layers equally instead of having a large bank of mutants to choose from allows for layers or sections of neurons that have a higher chance of showing. Throughout the research we assume that many neurons work together or perform similar tasks within the models. So, a random selection across each layer allows for a chance to select fewer mutants that have similar behavior. Since there are a minimal number of neurons within each layer of both the FCNN and LeNet-5 models, there is reason to believe this may not work as efficiently across much larger models as there may be a more probable chance to randomly select groups of mutants that are not as representative as these mutant groups. Overall, FCNN models obtain, on average, 1.22% *Mutation Score Error* while LeNet-5 models obtain a *Mutation Score Error* of 1.57%, on average.

For the Boundary Sample Size approach, we observe the middle row of Fig 5.4 to answer our question. We test 10 different parameter values for the *BSS Threshold* used to filter on how close the test suite samples are to the decision boundaries. The FCNN models are shown in the left plot while the LeNet-5 models can be observed in the right plot. Both plots show a gradual negative trend across the first half of the parameter values. The beginning *BSS Threshold* values differ from the later values so much because of these particular values' sensitivities to perturbations. Test data points chosen near the decision boundary by definition should present higher *Mutation Score Error* as these are the most sensitive test data points. But, as the *BSS Threshold* increases and the test suite size increases, it achieves a higher representativeness of the original test suite since it seems a differing value of test data points sit near decision boundaries across all models. Shen *et al.* mention that the new filtered test suite reaches an acceptable substitution rate at which these boundary samples can *almost completely* replace the whole test samples in terms of normal mutation scores [2]. The test suites' performance will plateau at some point in the plot. The model's *Mutation Score Error* seems to stabilize around 0.05-0.14 for the FCNN models while the LeNet-5's values stabilize around 0.00-0.08. So, the LeNet-5 models perform better, *Mutation Score Error* wise. The FCNN models do not perform poorly, but do have nearly double the stabilization values across all datasets. The *Mutation Score Error* values stay nearly the same, starting at different points for the different models,

because the used test suite contains the exact or nearly the exact same data points. Overall, the FCNN models obtain, on average, 16.51% *Mutation Score Error* and the LeNet-5 models obtain a *Mutation Score Error* of 11.69%, on average.

5.6 Answering Research Question 6

This research question investigates the effectiveness in testing data reduction by the Boundary Sample Size Selection approach. This question will be specifically answered by the first row of plots in Fig. 5.4.

The left plot shows the FCNN models while the right plot observes the LeNet-5 models. These plots have *BSS Size Percentage* as the y-axis. *BSS Size Percentage* represents the percent of size remaining of the original test suite's size. Both plots observe a general linearly positive trend in this *BSS Size Percentage*. So, as the *BSS Threshold* increases, more samples are added to the test suite, since values can come from farther away from the decision boundaries. So, values that the models are more confident of will start to appear in the test set based on a larger *BSS Size Percentage*. However, these threshold values follow an exponential growth pattern in order to be applicable and of use. Even with the largest *BSS Threshold* value, the largest *BSS Size Percentage* is near 48% for the FCNN models and near 21% for the LeNet-5 models. This shows that the FCNN models have less test data points near the decision boundaries for the datasets MNIST and KMNIST while there is a bigger spread across the decision space for the datasets EMNIST and FMNIST. EMNIST and FMNIST hold much more complex decision requirements, which may be why there are so many test samples varying distances away from the decision boundaries. The classes may have similar features that explain the model's partitioning of probabilities. The MNIST and KMNIST datasets have a miniscule amount of samples near the boundary decisions, meaning that most samples, around 90-95%, have very confident labels from both FCNN and LeNet-5 models. For datasets that have less data samples near the decision boundaries, this means that they may be less prone to perturbations from our mutation operators. So, they will only require a smaller test suite for the values that would cause a higher mutation score. However, I believe we fail to account for the mass of test samples that are far away from the decision boundaries, which results in the slightly higher *Mutation Score Error*s

in the second row. For FCNN models, the *BSS Size Percentage* has an average of 0.09, or 9% while the LeNet-5 models have an average of 0.04, or 4%.

These new *BSS* test suites seem to be representative enough to use in place of the original test suite for a few of the *BSS Thresholds*. The specific thresholds, 15, 20, 100, and 1000, have both a significant *Speedup* and reasonably small *MSE*. Having a comparable mutation score means that it can replace the original test suite. Having a decent *Speedup* and the comparable mutation score means that this is a desirable substitution suite. These test suites are usually more efficient, as they have the same behavior as the original test sets, but compute at a much faster rate. Instead of running inference on all of the test samples that are almost guaranteed to classify as the same label, using the *BSS* filter allows for us to only test the samples that have the potential to be shifted over those decision boundaries. When a sample is shifted over a decision boundary, its label is shifted from the label that the original model classified it as, resulting in a *killed class*, which produces a higher mutation score.

Chapter 6

Conclusion

Speeding up DNN mutation analysis could benefit plenty of application areas that depend on it. We presented two new clustering approaches and two seen selection approaches with promising results in reducing the costs of DNN mutation testing. All approaches generally decrease the mutation testing time, so optimal parameters and metrics that can balance the trade-offs between *Speedup* and *Mutation Score Error* would benefit their further research immensely. DEEPMAACC will be open-sourced, so the research community can apply and improve its clustering approaches.

Mutation Analysis of DNNs is still expanding and is a current active research topic. There are many challenges to confront, like non-determinism in training and mutation operators, computational expenses related to scalability, and the need to translate mutation operators and metrics from traditional software engineering. However, in this research, the goal is not to study the reliability of mutation score as a measure for test quality, rather the goal is to speed up a single run of mutation testing, so our clustering and selection approaches were both applied as separate methods.

6.1 Threats to Validity and Future Work

Like any other research work with empirical results, there are potential threats to validity to our work.

The generalizability of our results is limited since we do not have a complete representative sample of all DNN models and datasets to test our clustering approaches with. We evaluated

DEEPMAACC on only two DNN architectures (LeNet-5 and FCNN) and four classification datasets. While such a collection of model architectures might include modern models like AlexNet and VGGNet, they are not fully representative of the wide variety of DNN models and tasks in practice. Further evaluation on additional architectures (e.g. ResNet, Transformers) and datasets from diverse domains would strengthen the external validity of our findings.

Furthermore, the main clustering algorithm in this paper is hierarchical agglomerative clustering. Extending the research's reach to other clustering types may result in higher performance rates and speedup. Other hierarchical clustering, non-hierarchical clustering, or other applicable clustering algorithms should be investigated to check their influence on the trade-off between mutation score error and mutation testing speedup. Along with the need for clustering algorithms comes a need for clustering algorithm parameters. Clustering could also be done on different parts of the model. Instead of the neurons' weights and biases, we could potentially cluster on their activation of the test data suite.

For the current models and mutation operators, the *Neurons per Cluster* and *PARHAC Threshold* parameter values heavily impact the effectiveness of DEEPMAACC. Studying these parameters to find the optimal measures could allow for more in-depth research into the behavior of all components of DEEPMAACC. The selection approaches also have their own set of parameters, *Selection Fraction* and *BSS Threshold*. These parameters heavily impact their effectiveness as well, so studying the optimal parameter sets and possible combinations of approaches would be helpful research directions.

There is also a need for expansion of the set of tested mutant operators. However, there are many fault types and new ones are found periodically that may not be compatible with DEEPMAACC's operation. Having few mutant operators could lead to an incomplete assessment of the mutation testing approaches. Incorporating a more diverse set of mutation operators could help correct this, allowing DEEPMAACC to cover as many fault types as possible. With only three mutation operators, the study may not be capturing the full range of potential faults and behaviors that can occur in DNNs. These mutation operators are also only model-level mutation operators when there are also source-level mutation operators. There is a need to check if

these mutation operators produce realistic changes to the models. Could these approaches be applied to real applications?

Another threat could be the evaluation metrics, speedup and mutation score error, as they may not fully capture all relevant aspects of mutation testing acceleration. There could be other important factors, such as memory usage or scalability to very large models, that we did not measure.

The non-deterministic nature of DNN training and some aspects of our approach (e.g. random selection of cluster representatives or random mutant selection) could introduce variability in results. While we attempted to mitigate this through multiple runs, there may still be some effects of randomness. There is a need for a different method of choosing representatives of mutant clusters rather than one that produces these non-deterministic outputs.

DEEPMAACC may contain undetected bugs or implementation errors. These potential issues could affect the accuracy of the mutation analysis process, the clustering algorithms, or the performance measurements.

Our statistical analysis and conclusions are based on a limited number of models, datasets, mutation operators, and runs. A larger-scale study with more models and statistical rigor would increase confidence in the generalizability of our findings on the trade-offs between speedup and mutation score error. Future extensions of this paper will prove beneficial to mitigating these threats.

6.2 Thesis Conclusion

This paper presents DEEPMAACC, a framework for speeding up DNN mutation analysis through neuron and mutant clustering. DEEPMAACC implements two methods: (1) neuron clustering to reduce the number of generated mutants and (2) mutant clustering to reduce the number of mutants to be tested by selecting representative mutants for testing.

This paper further presents an empirical study of two proposed methods using 8 DNN models across 4 popular classification datasets and 2 model architectures. When compared to vanilla mutation analysis, the results provide empirical evidence that the neuron clustering approach on average speeds up mutation analysis by 69.77% with an average -26.84% error in

mutation score, while the mutant clustering approach speeds up mutation analysis by 35.31% with an average 1.96% error in mutation score. Random mutant selection accelerates mutation analysis by an average of 54.07% with an average mutation score error of 1.40%. Meanwhile, boundary sample selection, the final approach, results in an average of 86.23% speedup in mutation analysis at the cost of 14.10% average error in mutation score

6.3 Data Availability

DEEPMAACC source code and a replication package are publicly available at [39].

References

[1] A. Ghanbari, D.-G. Thomas, M. A. Arshad, and H. Rajan, "Mutation-based Fault Localization of Deep Neural Networks," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Computer Society, 2023, pp. 1301–1313. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ASE56229.2023.00171

[2] W. Shen, Y. Li, Y. Han, L. Chen, D. Wu, Y. Zhou, and B. Xu, "Boundary sampling to boost mutation testing for deep learning models," *Information and Software Technology*, vol. 130, p. 106413, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584920301737

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015. [Online]. Available: https://doi.org/10.1038/nature14539

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[5] A. Y. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng, "Deep speech: Scaling up end-to-end speech recognition," *CoRR*, vol. abs/1412.5567, 2014. [Online]. Available: http://arxiv.org/abs/1412.5567

[6] Y. Goldberg and G. Hirst, *Neural Network Methods in Natural Language Processing*. Morgan & Claypool Publishers, 2017.

[7] M. Pradel and S. Chandra, "Neural software analysis," *Commun. ACM*, vol. 65, no. 1, p. 86–96, dec 2021. [Online]. Available: https://doi.org/10.1145/3460348

[8] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 3422–3426.

[9] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, and M. J. Kochenderfer, "Algorithms for verifying deep neural networks," *Foundations and Trends® in Optimization*, vol. 4, no. 3-4, pp. 244–404, 2021. [Online]. Available: http://dx.doi.org/10.1561/2400000035

[10] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, and X. Yi, "A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability," *Computer Science Review*, vol. 37, p. 100270, 2020.

[11] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 1–36, 2022.

[12] R. DeMillo, R. Lipton, and F. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, vol. 11, no. 4, pp. 34–41, 1978.

[13] R. Hamlet, "Testing programs with the aid of a compiler," *IEEE Transactions on Software Engineering*, vol. SE-3, no. 4, pp. 279–290, 1977.

[14] W. Shen, J. Wan, and Z. Chen, "MuNN: Mutation Analysis of Neural Networks," in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2018, pp. 108–115.

[15] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao, and Y. Wang, "DeepMutation: Mutation Testing of Deep Learning Systems," in *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, 2018, pp. 100–111.

[16] N. Humbatova, G. Jahangirova, and P. Tonella, "Deepcrime: mutation testing of deep learning systems based on real faults," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 67–78. [Online]. Available: https://doi.org/10.1145/3460319.3464825

[17] Y. Lu, W. Sun, and M. Sun, "Towards mutation testing of reinforcement learning systems," *Journal of Systems Architecture*, vol. 131, p. 102701, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762122001977

[18] J. Wang, G. Dong, J. Sun, X. Wang, and P. Zhang, "Adversarial sample detection for deep neural network through model mutation testing," in *Proceedings of the 41st International Conference on Software Engineering*, ser. ICSE '19. IEEE Press, 2019, p. 1245–1256. [Online]. Available: https://doi.org/10.1109/ICSE.2019.00126

[19] Q. Hu, Y. Guo, M. Cordy, M. Papadakis, and Y. L. Traon, "MUTEN: Mutant-Based Ensembles for Boosting Gradient-Based Adversarial Attack," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2023, pp. 1708–1712.

[20] Q. Hu, L. Ma, X. Xie, B. Yu, Y. Liu, and J. Zhao, "DeepMutation++: A Mutation Testing Framework for Deep Learning Systems," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019, pp. 1158–1161.

[21] R. Lin, Q. Zhou, B. Wu, and X. Nan, "Robustness evaluation for deep neural networks via mutation decision boundaries analysis," *Information Sciences*, vol. 601, pp. 147–161, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025522003541

[22] Z. Wang, H. You, J. Chen, Y. Zhang, X. Dong, and W. Zhang, "Prioritizing test inputs for deep neural networks via mutation analysis," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 397–409.

[23] Q. Hu, Y. Guo, X. Xie, M. Cordy, M. Papadakis, L. Ma, and Y. L. Traon, "Aries: Efficient testing of deep neural networks via labeling-free accuracy estimation," in *Proceedings of the 45th International Conference on Software Engineering*, ser. ICSE '23. IEEE Press, 2023, p. 1776–1787. [Online]. Available: https://doi.org/10.1109/ICSE48619.2023.00152

[24] J. Sohn, S. Kang, and S. Yoo, "Arachne: Search-based repair of deep neural networks," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 4, May 2023. [Online]. Available: https://doi.org/10.1145/3563210

[25] H. Wu, Z. Li, Z. Cui, and J. Liu, "GenMuNN: A mutation-based approach to repair deep neural network models," *International Journal of Modeling, Simulation, and Scientific Computing*, vol. 13, no. 02, p. 2341008, 2022. [Online]. Available: https://doi.org/10.1142/S1793962323410088

[26] V. Riccio, N. Humbatova, G. Jahangirova, and P. Tonella, "DeepMetis: Augmenting a Deep Learning Test Set to Increase its Mutation Score," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2021, pp. 355–367.

[27] H. Deokuliar, R. S. Sangwan, Y. Badr, and S. M. Srinivasan, "Improving testing of deep-learning systems: A combination of differential and mutation testing results in better test data." *Queue*, vol. 21, no. 5, p. 54–65, Nov. 2023. [Online]. Available: https://doi.org/10.1145/3631340

[28] T. Zohdinasab, V. Riccio, and P. Tonella, "Focused test generation for autonomous driving systems," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 6, Jun. 2024. [Online]. Available: https://doi.org/10.1145/3664605

[29] M. V. Pour, Z. Li, L. Ma, and H. Hemmati, "A Search-Based Testing Framework for Deep Neural Networks of Source Code Embedding," in *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE Computer Society, 2021, pp. 36–46. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICST49551.2021.00016

[30] G. Jahangirova, A. Stocco, and P. Tonella, "Quality metrics and oracles for autonomous vehicles testing," in *2021 14th IEEE conference on software testing, verification and validation (ICST)*. IEEE, 2021, pp. 194–204.

[31] A. Ghanbari, "Decomposition of deep neural networks into modules via mutation analysis," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2024. New York, NY, USA: Association for Computing Machinery, 2024, p. 1669–1681. [Online]. Available: https://doi.org/10.1145/3650212.3680390

[32] A. V. Pizzoleto, F. C. Ferrari, J. Offutt, L. Fernandes, and M. Ribeiro, "A systematic literature review of techniques and metrics to reduce the cost of mutation testing," *Journal of Systems and Software*, vol. 157, p. 110388, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121219301554

[33] M. P. Usaola and P. R. Mateo, "Mutation testing cost reduction techniques: A survey," *IEEE Software*, vol. 27, no. 3, pp. 80–86, 2010.

[34] L.-C. Feng, X.-Y. Wang, S.-Y. Zhang, R.-Z. Gao, and Z.-H. Zhao, "Mutation operator reduction for cost-effective deep learning software testing via decision boundary change measurement," *Journal of Internet Technology*, vol. 23, no. 3, pp. 601–610, 2022. [Online]. Available: https://jit.ndhu.edu.tw/article/view/2705

[35] Y. Wang, Z. Zhang, Y. Yao, and Z. Huang, "A fine-grained evaluation of mutation operators for deep learning systems: A selective mutation approach," in *Proceedings of the 14th Asia-Pacific Symposium on Internetware*, ser. Internetware '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 123–133.

[36] Y. Li, W. Shen, T. Wu, L. Chen, D. Wu, Y. Zhou, and B. Xu, "How higher order mutant testing performs for deep learning models: A fine-grained evaluation of test effectiveness and efficiency improved from second-order mutant-classification tuples," *Information and Software Technology*, vol. 150, p. 106954, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584922000994

[37] L. Dhulipala, D. Eisenstat, J. Łącki, V. Mirronki, and J. Shi, "Hierarchical agglomerative graph clustering in poly-logarithmic depth," 2022. [Online]. Available: https://arxiv.org/abs/2206.11654

[38] M. Klabunde, T. Schumacher, M. Strohmaier, and F. Lemmerich, "Similarity of neural network models: A survey of functional and representational measures," 2024. [Online]. Available: https://arxiv.org/abs/2305.06329

[39] L. Lyons and A. Ghanbari, "On Accelerating Deep Neural Network Mutation Analysis by Neuron and Mutant Clustering," 2025, accessed: 01/25. [Online]. Available: https://github.com/snoylnerual/ICST_DeepMAACC

[40] R. J. Lipton, "Fault diagnosis of computer programs," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, 1971.

[41] T. A. Budd and F. G. Sayward, "Users guide to the pilot mutation system," Yale University, New Haven, Connecticut, Tech. Rep. Techreport 114, 1977.

[42] T. A. Budd, R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "The design of a prototype mutation system for program testing," in *Proceedings of the AFIPS National Computer Conference*, Anaheim, New Jersey, June 1978, pp. 623–627.

[43] H. Agrawal, R. A. DeMillo, B. Hathaway, W. Hsu, E. W. Krauser, R. J. Martin, A. P. Mathur, and E. Spafford, "Design of mutant operators for the c programming language," Purdue University, West Lafayette, Indiana, Tech. Rep. SERC-TR-41-P, March 1989.

[44] S. Kim, J. A. Clark, and J. A. McDermid, "Investigating the effectiveness of object-oriented testing strategies using the mutation method," in *Proceedings of the 1st Workshop on Mutation Analysis (MUTATION'00)*, San Jose, California, October 2001, pp. 207–225.

[45] Y.-S. Ma, A. J. Offutt, and Y.-R. Kwon, "Mujava: An automated class mutation system," *Software Testing, Verification Reliability*, vol. 15, no. 2, pp. 97–133, June 2005.

[46] A. Derezińska, "Advanced mutation operators applicable in c# programs," Warsaw University of Technology, Warszawa, Poland, Tech. Rep., 2005.

[47] ——, "Quality assessment of mutation operators dedicated for c# programs," in *Proceedings of the 6th International Conference on Quality Software (QSIC'06)*, Beijing, China, October 2006.

[48] W. K. Chan, S. C. Cheung, and T. H. Tse, "Fault-based testing of database application programs with conceptual data model," in *Proceedings of the 5th International Conference on Quality Software (QSIC'05)*, Melbourne, Australia, September 2005, pp. 187–196.

[49] F. C. Ferrari, J. C. Maldonado, and A. Rashid, "Mutation testing for aspect-oriented programs," in *Proceedings of the 1st International Conference on Software Testing, Verification, and Validation (ICST'08)*, Lillehammer, Norway, April 2008, pp. 52–61.

[50] J. Offutt, J. Payne, and J. M. Voas, "Mutation operators for ada," Department of Information and Software Systems Engineering, George Mason University, Tech. Rep. ISSE-TR-96-09, March 1996.

[51] J. Offutt and W. Xu, "Specification-level mutation operators for testing idl-based software," in *Proceedings of the 12th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE Computer Society, 2003, pp. 200–209.

[52] R. A. DeMillo and A. J. Offutt, "Constraint-based automatic test data generation," *IEEE Transactions on Software Engineering*, vol. 17, no. 9, pp. 900–910, September 1991.

[53] A. J. Offutt and R. H. Untch, *Mutation 2000: Uniting the Orthogonal*. Boston, MA: Springer US, 2001, pp. 34–44. [Online]. Available: https://doi.org/10.1007/978-1-4757-5939-6_7

[54] T. A. Budd, "Mutation analysis of program test data," Ph.D. dissertation, Yale University, New Haven, Connecticut, 1980.

[55] B. J. M. Grün, D. Schuler, and A. Zeller, "The impact of equivalent mutants," in *Proceedings of the 4th International Workshop on Mutation Analysis (MUTATION'09)*, Denver, Colorado, April 2009, pp. 192–199.

[56] Y. Jia and M. Harman, "Constructing subtle faults using higher order mutation testing," King's College London, Strand Campus, London WC2R 2LS, UK, Tech. Rep. Technical Report TR-08-03, 2008.

[57] W. E. Wong, Ed., *Mutation Testing for the New Century*. Boston, MA: Springer US, 2001.

[58] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. L. Traon, and M. Harman, "Chapter six - mutation testing advances: An analysis and survey," in *Advances in Computers*, ser. Advances in Computers, A. M. Memon, Ed. Elsevier, 2019, vol. 112, pp. 275–378. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065245818300305

[59] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 649–678, 2011.

[60] G. Jahangirova and P. Tonella, "An empirical evaluation of mutation operators for deep learning systems," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 2020, pp. 74–84.

[61] Y. Jia and M. Harman, "Higher order mutation testing," *Information and Software Technology*, vol. 51, no. 10, pp. 1379–1393, 2009, source Code Analysis and Manipulation, SCAM 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584909000688

[62] H. Coles, T. Laurent, C. Henard, M. Papadakis, and A. Ventresque, "Pit: a practical mutation testing tool for java," in *Proceedings of the 25th international symposium on software testing and analysis*, 2016, pp. 449–452.

[63] N. Chetouane, L. Klampfl, and F. Wotawa, "Investigating the effectiveness of mutation testing tools in the context of deep neural networks," in *Advances in Computational Intelligence - 15th International Work-Conference on Artificial Neural Networks, IWANN 2019, Gran Canaria, Spain, June 12-14, 2019, Proceedings, Part I*, ser. Lecture Notes in

Computer Science, I. Rojas, G. Joya, and A. Català, Eds., vol. 11506. Springer, 2019, pp. 766–777. [Online]. Available: https://doi.org/10.1007/978-3-030-20521-8_63

[64] L. Klampfl, N. Chetouane, and F. Wotawa, "Mutation testing for artificial neural networks: An empirical evaluation," in *20th IEEE International Conference on Software Quality, Reliability and Security, QRS 2020, Macau, China, December 11-14, 2020*. IEEE, 2020, pp. 356–365. [Online]. Available: https://doi.org/10.1109/QRS51102.2020.00054

[65] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *CoRR*, vol. abs/1604.07316, 2016. [Online]. Available: http://arxiv.org/abs/1604.07316

[66] H. Naessens, T. Philip, M. Piatek, K. Schippers, and R. Parys, "Predicting flight routes with a deep neural network in the operational air traffic flow and capacity management system," *EUROCONTROL Maastricht Upper Area Control Centre, Maastricht Airport, The Netherlands, Tech. Rep*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:202697629

[67] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, "Deep learning for healthcare: review, opportunities and challenges," *Briefings in Bioinformatics*, vol. 19, no. 6, pp. 1236–1246, 05 2018. [Online]. Available: https://doi.org/10.1093/bib/bbx044

[68] A. Roy, J. Sun, R. Mahoney, L. Alonzi, S. Adams, and P. Beling, "Deep learning detecting fraud in credit card transactions," in *2018 Systems and Information Engineering Design Symposium (SIEDS)*, 2018, pp. 129–134.

[69] L. Zhang, J. Lin, B. Liu, Z. Zhang, X. Yan, and M. Wei, "A review on deep learning applications in prognostics and health management," *IEEE Access*, vol. 7, pp. 162 415–162 438, 2019.

[70] D.-G. Thomas, M. Biagiola, N. Humbatova, M. Wardat, G. Jahangirova, H. Rajan, and P. Tonella, "µPRL: A Mutation Testing Pipeline for Deep Reinforcement Learning

based on Real Faults," in *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2025, pp. 444–456. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICSE55347.2025.00036

[71] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," 2018. [Online]. Available: https://arxiv.org/abs/1708.08559

[72] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[73] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, p. 84–90, May 2017. [Online]. Available: https://doi.org/10.1145/3065386

[74] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1409.1556

[75] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[76] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *CoRR*, vol. abs/1708.07747, 2017. [Online]. Available: http://arxiv.org/abs/1708.07747

[77] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: Extending MNIST to handwritten letters," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 2921–2926.

[78] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, "Deep learning for classical japanese literature," *CoRR*, vol. abs/1812.01718, 2018. [Online]. Available: http://arxiv.org/abs/1812.01718

[79] L. Kaufman and P. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, ser. Wiley Series in Probability and Statistics. Wiley, 1990. [Online]. Available: https://books.google.com/books?id=Q5wQAQAAIAAJ

[80] Wikipedia contributors, "Euclidean distance — Wikipedia, the free encyclopedia," 2024, accessed: 10/24. [Online]. Available: https://aub.ie/SCI10v

[81] S. Hussain, "Mutation clustering," Master's thesis, King's College London, 2008.

[82] C. Ji, Z. Chen, B. Xu, and Z. Zhao, "A novel method of mutation clustering based on domain analysis," in *Proceedings of the 21st International Conference on Software Engineering & Knowledge Engineering (SEKE'2009), Boston, Massachusetts, USA, July 1-3, 2009*. Knowledge Systems Institute Graduate School, 2009, pp. 422–425.

[83] M. Yu and Y.-S. Ma, "Possibility of cost reduction by mutant clustering according to the clustering scope," *Software Testing, Verification and Reliability*, vol. 29, no. 1-2, p. e1692, 2019, e1692 stvr.1692. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.1692

[84] L. Zhang, S.-S. Hou, J.-J. Hu, T. Xie, and H. Mei, "Is operator-based mutant selection superior to random mutant selection?" in *Proceedings of the 32nd International Conference on Software Engineering (ICSE)*. ACM, 2010, pp. 435–444.

[85] B. Qi, H. Sun, X. Gao, and H. Zhang, "Patching weak convolutional neural network models through modularization and composition," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3551349.3561153

[86] B. Qi, H. Sun, H. Zhang, and X. Gao, "Reusing convolutional neural network models through modularization and composition," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 3, mar 2024. [Online]. Available: https://doi.org/10.1145/3632744

[87] Wikipedia contributors, "Mann–whitney u test — Wikipedia, the free encyclopedia," https://aub.ie/kKkOyh, 2024, accessed: 10/24.

Appendices

Appendix A

Reconfigured Figures For Comparisons

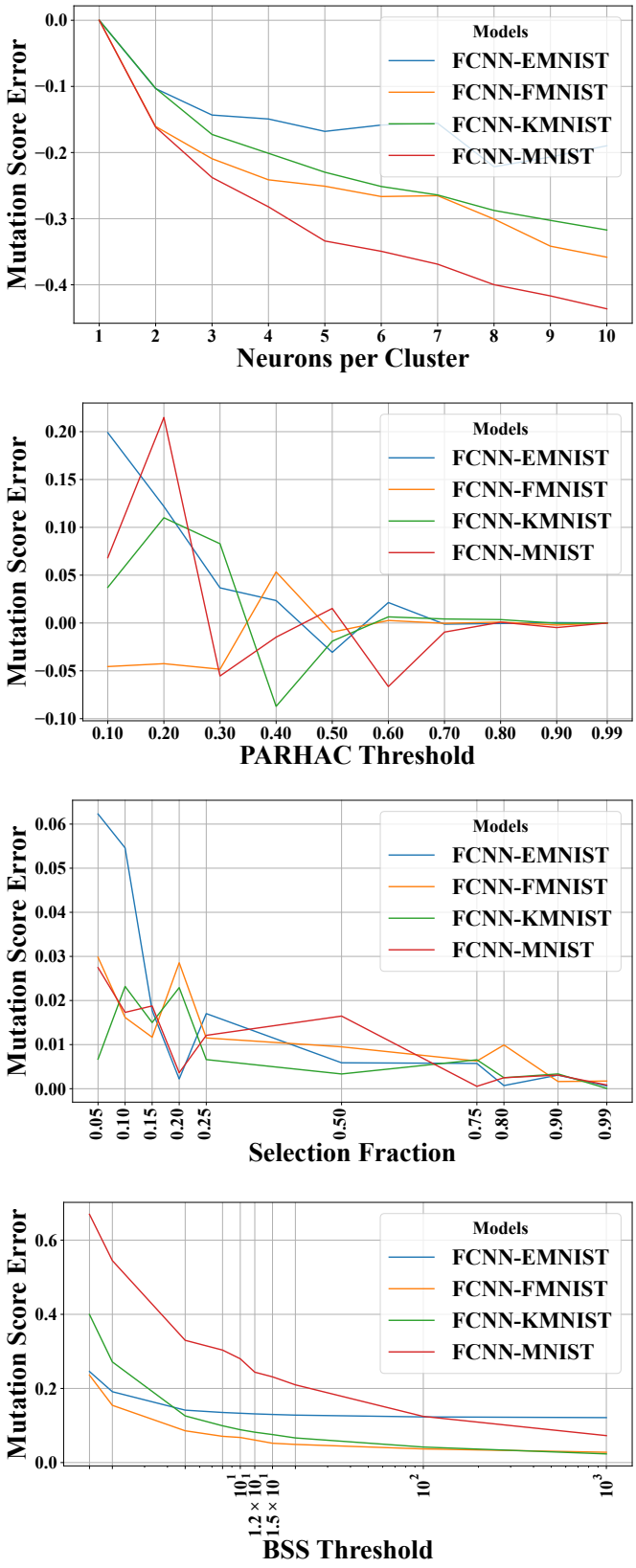

Figure A.1: This figure shows the Mutation Score Error of the four approaches with their respective parameters for FCNN models. The top plot is Neuron Clustering, the second plot is Mutant Clustering, the third plot is Random Mutation Selection, and the bottom plot is Boundary Sample Size Selection.

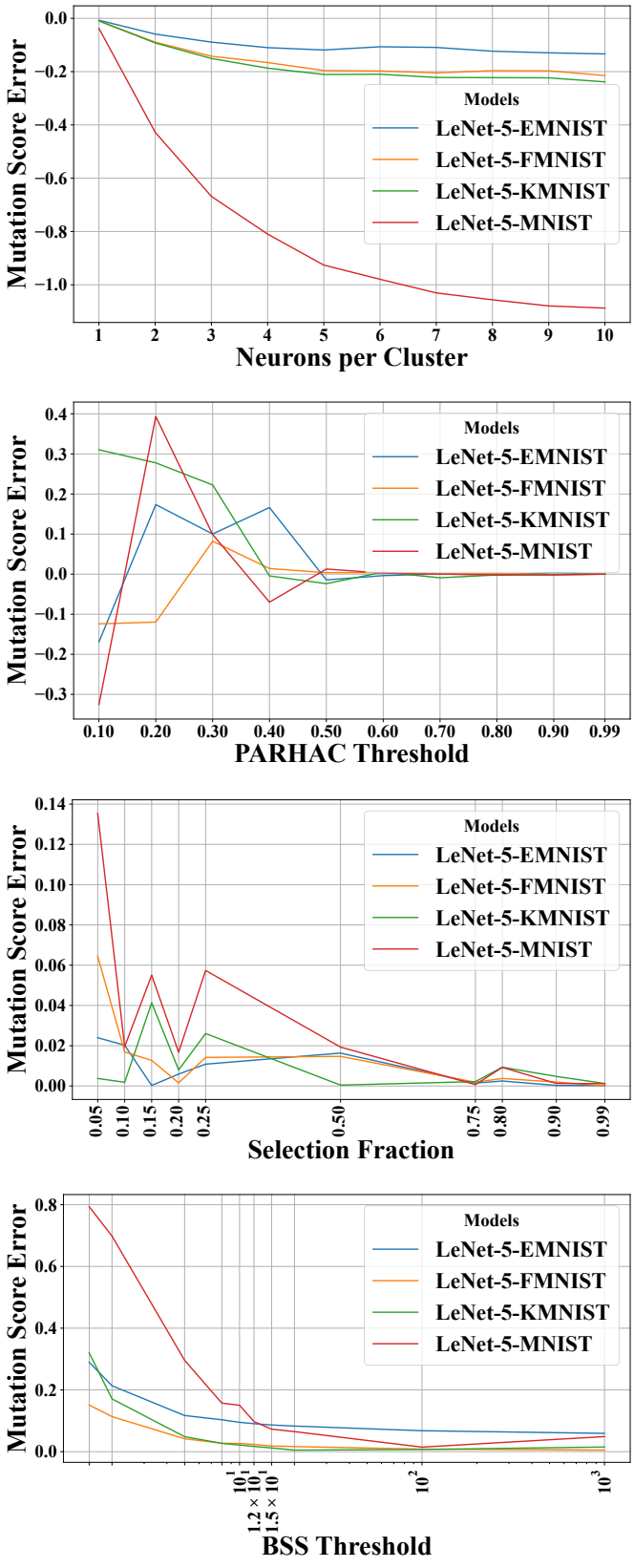

Figure A.2: This figure shows the Mutation Score Error of the four approaches with their respective parameters for LeNet-5 models. The top plot is Neuron Clustering, the second plot is Mutant Clustering, the third plot is Random Mutation Selection, and the bottom plot is Boundary Sample Size Selection.

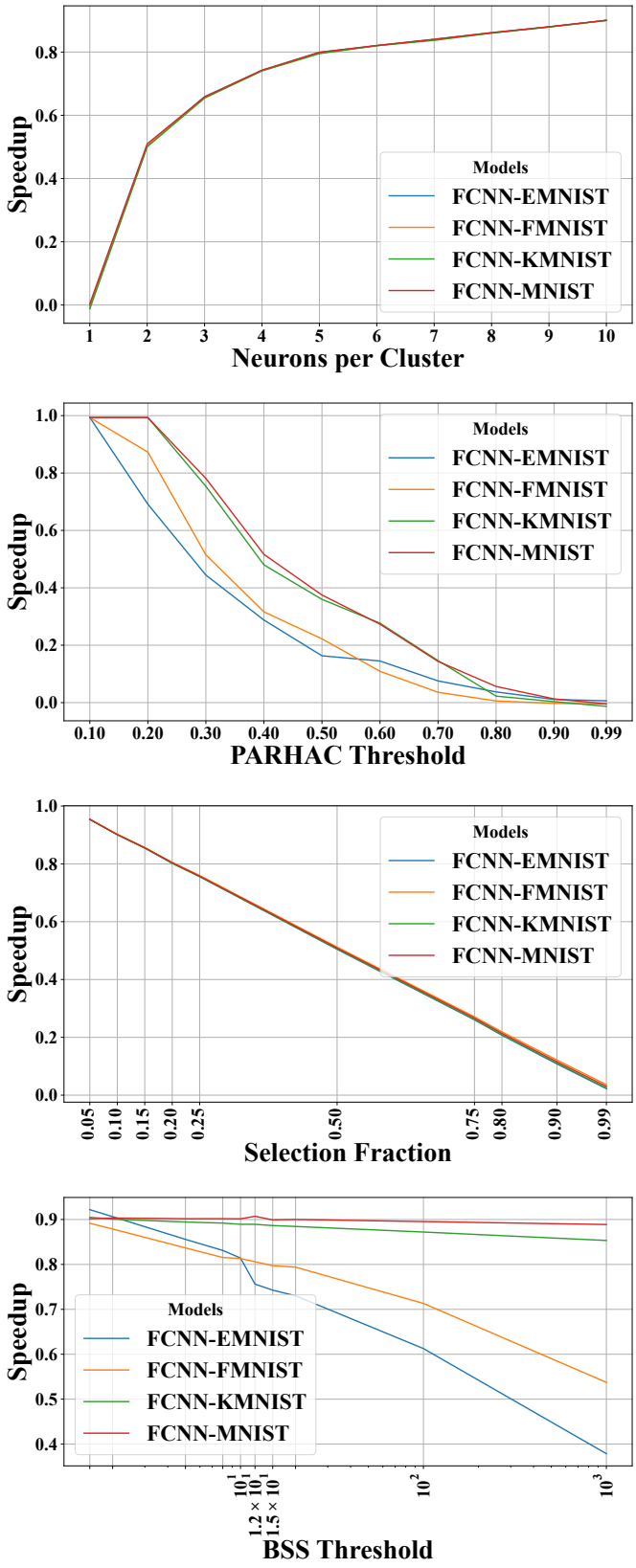

Figure A.3: This figure shows the Speedup of the four approaches with their respective parameters for FCNN models. The top plot is Neuron Clustering, the second plot is Mutant Clustering, the third plot is Random Mutation Selection, and the bottom plot is Boundary Sample Size Selection.

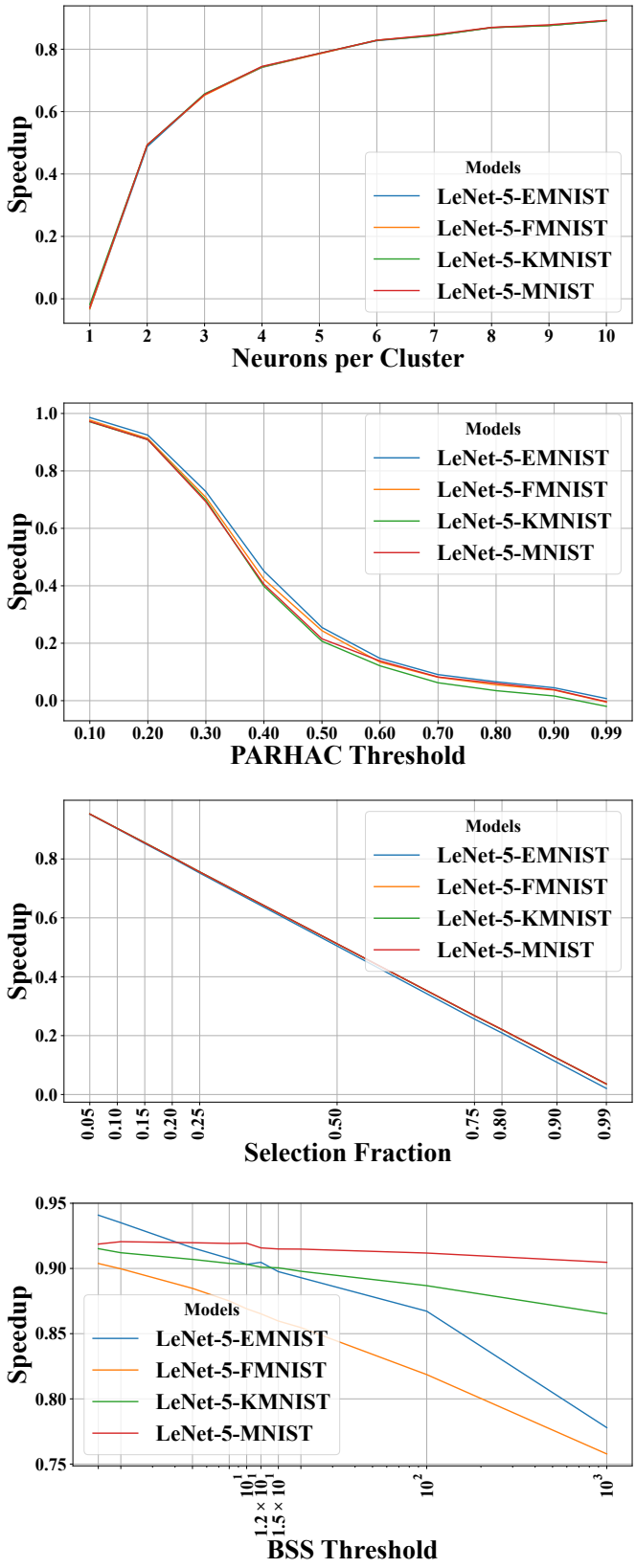

Figure A.4: This figure shows the Speedup of the four approaches with their respective parameters for LeNet-5 models. The top plot is Neuron Clustering, the second plot is Mutant Clustering, the third plot is Random Mutation Selection, and the bottom plot is Boundary Sample Size Selection.

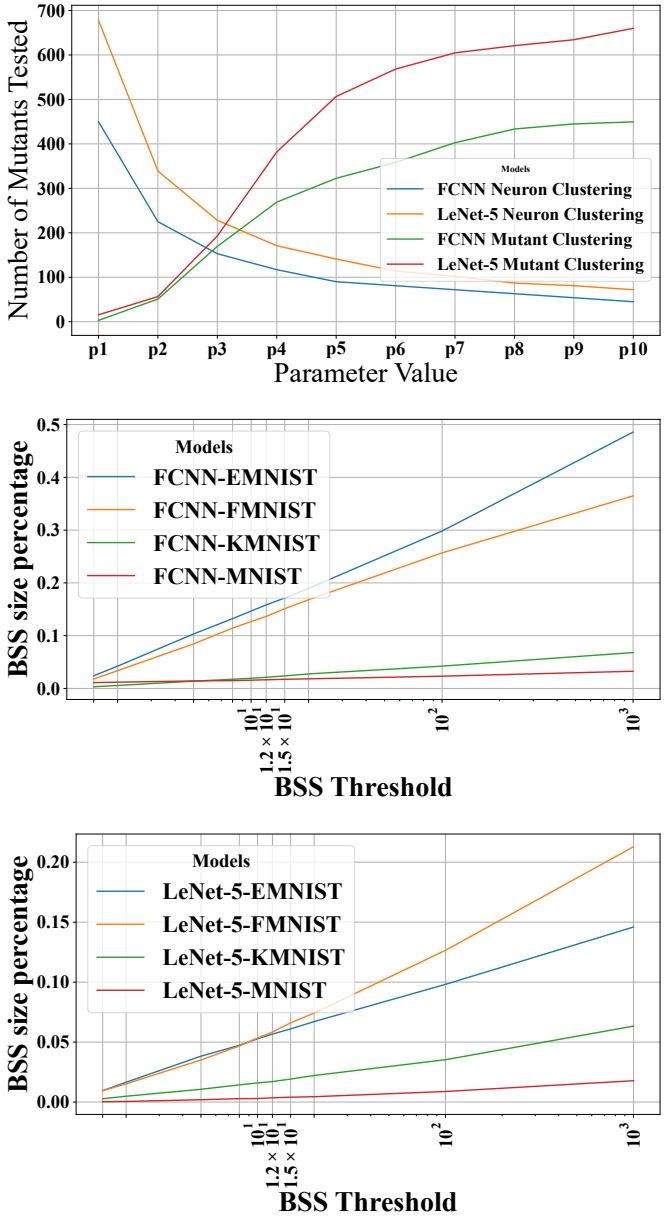

Figure A.5: This figure shows the relative size comparisons of their approaches. While not directly comparable since the top plot shows *Number of Mutants Tested* and the next two plots show *BSS Size Percentage*, you are able to observe their relative efficiencies for reduction. The top plot shows how many mutants are tested for the Neuron Clustering and Mutant Clustering approaches, having both FCNN and LeNet-5 lines within the same plot. The second and third plot show the *BSS Size Percentage*, which is what percent of the test suite is left after the BSS filter. The second plot holds the FCNN models and the third plot holds the LeNet-5 models.