National College of Ireland

Mutation Analysis in Deep Neural Networks: A study on Test Suite Effectiveness

MSc Research Project
Master of Science in Cyber Security

Adarsh Narmula
Student ID: x23315857

School of Computing
National College of Ireland

Supervisor: Michael Prior

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Adarsh Narmula
Student ID:	X23315857
Programme:	Master of Science in Cyber Security **Year:** 2024-2025
Module:	MSc Research Project
Lecturer:	Michael Prior
Submission Due Date:	11 August 2025
Project Title:	Mutation Analysis in Deep Neural Networks: A Study on test Suite Effectiveness
Word Count:	8391 **Page Count:** 30

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Adarsh Narmula

Date: 11 August 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	☑
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☑
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☑

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Mutation Analysis in deep Neural Networks: A study On Test suite Effectiveness

Abstract

Even subtle failures in Deep Neural Networks (DNNs) now underlie safety-critical applications like autonomous driving and healthcare where failures can be catastrophic. As much as there has been significant improvement in accuracy of DNN, little has been found to be given to the effectiveness of the test suites that are used to validate the models. A method that would provide a promising solution to DNN reliability would be mutation testing, which is a method of intentionally injecting artificial faults (mutants) to test the adequacy of the tests. The work focuses on the question of how the mutation testing metrics may be used to assess the performance of a DNN test suite and potentially enhance its performance, specifically by improving robustness in safety-critical settings. An in-depth comparative study of the most popular mutation testing frameworks, including DeepMutation, DeepMutation++, MuNN, and others, is provided by comparing their mutation operators, evaluation criteria, and empirical outcomes. The paper also defines such important metrics as Mutation Score, Average Error Rate (AER), and K_Score1/2 and evaluates their diagnostic capability in terms of identifying the weaknesses of the test suite. Through integration of results of various frameworks, the study identifies the advantages and drawbacks, as well as feasible applications of mutation testing in safety-critical environments. The findings show that metric-based mutation testing can not only increase fault detection effectiveness, but also offer actionable information to execute specific test suites improvement. This paper ends with conclusions of how mutation testing can be incorporated into the DNN development lifecycle and how it can be used to achieve safer and more trustworthy AI systems in addressing issues related to equivalent mutants, computational cost, and alignment with real-world fault taxonomies.

Table of Contents

1 Introduction

Safety-critical systems use Deep Neural Networks (DNNs) today, such as self-driving cars and medical diagnostic devices. The failure of DNNs even in minor cases in such high-stake fields can be disastrous, and therefore, robustness and reliability of DNNs are of utmost importance. Although there has been remarkable improvement in obtaining high accuracy, there are still concerns regarding the way these models perform in unseen, noisy or adversarial settings. The traditional DNN testing is yet to mature and in most cases assesses the ability of test suites to identify faults. In contrast, the quality assessment of a test suite is a normal practice in traditional software engineering, whereas, in the DNN scenario, the practitioners often assume fixed test accuracy. A model can have high accuracy on a fixed data set but on corner cases or minor perturbation, it may fail which is unviable in safety-related applications.

Mutation testing fills this gap by inserting small artificial faults, called mutants, into a program or model, and quantifying the ability of the test suite to reveal the changes. The extent to which the test is adequate, called the mutation score, is the proportion of mutants being killed (Jia and Harman, 2011). Mutation testing was initially formulated during the early 1970s and is based on the Competent Programmer Hypothesis (CPH) and the Coupling Effect Hypothesis (CEH) and has been used extensively in conventional software development. These conjectures are used to motivate the use of miniature, representative mutations as a surrogate of true faults: CPH asserts that most real faults are tiny departures of correct code, and CEH asserts that successful detection of simple faults should lead to success in detecting more complex ones.

These principles have to be modified in the context of DNNs. However, unlike traditional programs, DNNs are composed of vast spaces of parameters, non-linear interactions, and data-driven logic, i.e., small changes can be represented by small variations of weights, deactivation of a neuron, or rearrangement of architecture. Also, DNN training and inference are frequently stochastic--weight initialisation is random, GPU computation can be non-determistic--which further complicates fault reproduction and mutant evaluation. The response of researchers has been to come up with DNN-specific mutation operators and evaluation metrics that still reflect the nature of mutation testing but adapted to the peculiarities of neural networks.

The best part of this research is the investigation of mutation testing metrics other than the conventional mutation score. Although the mutation score is still the focus, other metrics have been suggested to complement the mutation score; these include the Average Error Rate (AER) and K_Score1/2 to measure the effectiveness of test cases and input segments. These metrics give more diagnostic information that can be used to selectively improve the test suites, and reveal weaknesses in particular model components.

1.1 Overview of Mutation Testing in Deep Neural Networks

Deep Neural Networks (DNNs) have become the basis of contemporary artificial intelligence through their uses as a subset of computer vision, natural language processing, autonomous systems and healthcare. However, these models are in their very core, difficult to interpret and complex to understand which poses a great challenge on making them reliable and robust. The testing techniques used conventionally in the software development processes like unit testing and integration testing show little success in the context of deep learning models because of their non-linear behaviour and high dimension of parameters. Thus, the necessity to develop specialized testing models of DNNs has increased.

Among these approaches, we should mention mutation testing, which has been applied to the machine learning and DNNs setting. Mutation testing is applied by the performance of hybrid, limited changes (mutations) to a model's weights, architecture or input data sets, and assessing the ability of the test suite in the model to discover the changes created. The mutation score of any mutation test suite would tell how good a model is in identifying the introduced mutations (Mienye and Swart, 2024).

1.2 Importance of Test Suite Effectiveness in DNNs for Safety-Critical Applications

When it comes to safety-critical applications, which include autonomous vehicles, healthcare, and aerospace, cost of failure is astronomical. In such areas, the behaviour of DNNs has to be thoroughly put to the test in order to ascertain that they can cope with a variety of unpredictable situations and inputs. Effective execution of a test suite is important to ensure that the model is reliable before it can be implemented in the real-world.

The standard testing approaches fail, however, with respect to DNNs. The behaviour that DNNs exhibit is occasionally counterintuitive and their capacity to efficiently generalise to previously unseen, novel data is dependent on a myriad of parameters including training data,

model architecture and hyperparameters. In order to address these limitations, mutation testing is applied to inject controlled faults in a program that simulates potential real-world failures to provide the test suite a chance to gauge how the model will react to such unexpected conditions. This ensures that the model does not just statistically perform well on the training set but is robust to perturbations thus is more robust in an uncertain environment (Giannaros et al., 2023).

1.3 Research Question and Aim

The primary objective in the present report is to explore the strengths of mutation testing in assessing the robustness of Deep Neural Networks (DNNs), especially the CIFAR-10 dataset, and also to test how the model will perform under various mutations. The research question that will direct the current study is:

"How can mutation testing metrics be leveraged to evaluate and improve the effectiveness of test suites for deep neural networks, thereby enhancing their robustness and reliability in safety-critical applications? "

To address this question, the report aims to:

1. Fit and train a baseline CNN to the CIFAR-10 dataset.
2. Perform mutation testing, where faults are introduced like perturbed weights, layer deletion.
3. Compare the performance of model before and after the mutations, e.g. in terms of accuracy, mutation score.
4. Evaluate how mutations affect model performance and determine the usefulness of the mutation testing framework in terms of model robustness.

The main hypothesis of this report is to determine the possibility of the mutation testing to determine the robustness of the Deep Neural Networks (DNNs), paying attention to the CIFAR-10 data and test the model on the effect of the different mutations that are introduced to the model. The research question that will guide the study in this research is as follows. (Hu et al., 2023).

2 Literature Review

2.1 Overview of Deep Neural Networks (DNNs)

DNNs represent the foundation of the current machine learning and AI since they demonstrate the outstanding ability to learn using massive amount of data and can accomplish complicated tasks, including image recognition, natural language processing, and reinforcement learning. DNN is an artificial neural network that consists of several layers of interconnected nodes (neurons) simulating the work of the human brain. The number of hidden layers is called depth, and deeper architectures can typically produce more sophisticated and stronger feature extraction.

The process of training a DNN usually consists of algorithms like backpropagation that successively change the weights in a network to minimise the difference between the predicted and actual outputs. This is used to gradually make the model more accurate in its predictions (Taye, 2023).

2.2 Traditional Testing Methods and Their Limitations

Unit testing, integration testing and system testing are all traditional software testing methods which are aimed at ensuring that software programs are correct and perform as required. Nevertheless, such approaches are less applicable to the machine learning models, especially ones that utilize deep neural networks, since they presuppose a deterministic system. However, DNNs, in comparison, are very non-linear, and are associated with a high parameter count, so traditional testing methods cannot be used to identify errors or surprises in behaviour.

The traditional software testing methods such as unit testing, integration testing, and system testing are meant to provide assurance that software is both functional and correct in terms of its requirements. Nevertheless, these methods presuppose a deterministic system, which is why they are not very useful in machine learning, particularly, when it is dependent on DNNs.

DNNs are non-linear, high-dimensional and have millions of parameters. Their behaviour when perturbed by small disturbances or sudden pulses cannot be easily measured using traditional tests. As an example, unit tests of the individual units can not say anything about how a DNN may work presented with noisy data or adversarial manipulations.

The other constraint is that classical testing does not involve robustness testing. DNNs require robustness, or the characteristic of a model to be effective with unexpected or noisy inputs. However, it is hardly ever measured traditionally. Besides, these tests usually only include the most common cases and ignore the rare, but significant, failure modes. In DNNs, very minimal changes in the inputs can lead to radically different outputs, a fact evidenced by how these models are vulnerable to adversarial attacks (Islam et al., 2023).

2.3 Mutation Testing Frameworks: DeepMutation, DeepMutation++, MuNN, etc.

In a bid to overcome the drawbacks of conventional testing techniques, mutation testing has come out as a bright prospect of testing the power and efficiency of deep learning models. Mutation testing is a test in which small, controlled faults, called mutations, are introduced into a model or dataset deliberately. The system is then monitored to see how it responds to these faults, so as to see whether they are identifiable by the test suite. A quantitative index of the effectiveness of the test suite and of the robustness of the model is the mutation score that indicates the percentage of faults that have been identified.

The DeepMutation is one of the first and most impactful models in this field. This framework presents a range of model-level perturbations, e.g., changing weights, changing the connections of neurons, or changing the activation functions. DeepMutation uses the effects of these perturbations to determine the robustness of a model to make any small internal change, which is a metric-based perspective of robustness.

DeepMutation++ builds on this and further develops the initial framework with the development of other types of mutations. These are structural variations, including stripping away layers of the architecture, restating the parameters of models, and making modifications at the data level e.g. introducing noise to inputs. These additional mutation strategies allow more features of robustness testing, both architectural and data-related vulnerabilities.

The other notable tool is MuNN (Mutation Testing of Neural Networks), which is especially applicable to convolutional neural networks (CNNs). MuNN propagates mutations at several levels: data, parameter, and architecture and such operations as the transformation of activation functions and altering layer structures. This enables the testers to assess the reaction of CNNs to minor and substantial structural changes.

Collectively, these frameworks offer effective ways of finding DNN test suites weak points. They enable them to identify vulnerabilities and make improvements specifically to improve

the performance of deep learning systems by testing out model performance across different mutation scenarios and thus improving robustness and reliability of the system. (Pugh et al., 2025).

Framework	Mutation Granularity & Operators	Key Metrics Used	Evaluation Domain	Strengths	Limitations
DeepMutation	Model-level operators (e.g., weight perturbation, neuron removal, activation function changes, layer removal)	Mutation Score, Average Error Rate (AER)	Image classification (MNIST, CIFAR-10)	First DNN-specific mutation testing framework; demonstrated feasibility on CNNs; AER correlates with robustness	Limited to CNN/FNN models; no RNN support; limited granularity beyond model level; equivalent mutant issue not fully addressed
DeepMutation++	Extended model-level operators (8 from DeepMutation + 9 new for RNNs); includes static & dynamic mutants; weight, neuron, layer, and sequence segment mutations	Mutation Score, AER, K_Score1 (mutants killed per input), K_Score2 (mutants killed per segment)	Image (MNIST), Text (IMDB with LSTM), Feed-forward and Recurrent NNs	Supports multiple architectures; fine-grained metrics identify weak input regions; dynamic mutation reduces storage cost	More complex implementation; high computational cost; still tested mostly on small datasets

MuNN	Model-level operators: Change Activation Function (CAF), Change Bias Value (CBV), Change Weight Value (CWV), Delete Input Neuron (DIN), Delete Hidden Neuron (DHN)	Mutation Score (traditional)	Image classification (shallow & deep MLPs on MNIST)	Highlights differences in mutation effects based on network depth; simple, interpretable operators No data-level or convolution-specific mutations; small-scale evaluation; lacks methods for equivalent mutant detection

2.4 Evolution of Mutation Testing in DNNs

Mutation testing of DNNs has developed to date as the necessity to guarantee AI steadiness increases. The initial studies were mostly based on the theory of weight mutation whereby any individual weight or connection in the network is perturbed to monitor the change in performance. Such early techniques were limited, including the failure of simulating real-world faults, and the expense of testing all possible mutations.

With DNNs gradually increasing in complexity, mutations and perturbations at layer level and data level became more popular. Researchers understood that the variations of weights only did not represent the entire set of possible faults that might happen in a deep learning model. Such frameworks as DeepMutation++ have responded by giving more varied mutation prioritization, including test responses to model structural changes, such as layer removals or alterations in activation functions.

Moreso, the emergence of adversarial attacks indicated the necessity of model testing with malicious inputs. Consequently, mutation testing incorporated data-level mutations like random addition of noise, image rotations and cropping in order to approximate possible problems that the model would face in practice (Gopinath Kathiresan, 2023).

2.5 Challenges in DNN Mutation Testing

Despite the fact that mutation testing of DNNs has been refined, there are several problems which have not been circumvented yet. One of the great problems is the computational cost. DNNs are large and complex models and mutation testing of the models may end up being very resource intensive. To cover large numbers of datasets and deep models, it requires immeasurable computing power to produce sufficient mutations to cover all possibilities of a failure.

The other issue is the fact that mutation tests on DNNs are not standardized. A different set of mutation strategies will be offered by different frameworks, and the latter can behave in a different way depending on a specific task and dataset. The testing of mutations in deep learning is thus not a single-solution procedure and researchers could be forced to alter provided frameworks to suit their targeted application (Lin et al., 2022).

Although substantial progress has been made, a number of issues still remain in running mutation testing with DNNs:

Computational Cost Large and complex DNNs are resource-intensive to mutation test. The graph produced by the mutation testing process can take several days or weeks of computation time especially on large datasets, to generate enough mutants to cover the faults in a meaningful way.

Absence of Standardisation Different frameworks use different sets of mutation strategies, which can act inconsistently with the model architecture and dataset. The absence of uniformity implies that there is no standardized procedure of mutation testing deep learning models. In many research studies, there is a necessity to adjust or personalise already existing frameworks to work with a particular application.

These issues will be essential in the further scalability, consistency, and realism of mutation testing as it pertains to fault conditions in the real world.

3 Research Methodology

3.1 Experimental Setup

This paper serves to attempt to verify the possibility of mutation testing to be equally useful when employed to Deep Neural Networks (DNNs). The experiment was set up to provide evidence of the strength of a certain Convolutional Neural Network (CNN) model using the dataset of CIFAR-10 and apply various approaches to mutation testing. This is accomplished by training the baseline model, mutating the model using different forms of mutations (ex: weight perturbations and layer mutations) and comparing the model performance of the mutated models to the original.

1. Frameworks & Tools: Python, TensorFlow/Keras to train the model and implement mutation; NumPy, Pandas to work with the data; Matplotlib/Seaborn to visualise.
2. Hardware: An experiment that was carried out in a GPU-enabled platform to minimize computation time.
3. Parameters of Training: Adam optimiser, categorical cross-entropy loss, batch size 64, 50 epochs, learning rate 0.001

The experiment has a systematic way:

1. **Model Training**: A CNN is trained from scratch on CIFAR-10 and this becomes a pretext/baseline model.
2. **Mutation Application**: After training a model, various mutations are performed on the model like weight perturbation, and removal of layer.
3. **Evaluation**: By looking at the comparison between performance of the mutated models and the original model, accuracy and mutation score are compared.
4. **Analysis**: The efficiency of mutation testing method is evaluated on the basis of the capability of the model to resist different kinds of mutations and on the effect of such mutations on model performance (Azam et al., 2024).

3.2 Research Design

The study follows these steps:

1. **Baseline Model Development** – To set up a baseline performance, train a CNN model on CIFAR-10.
2. **Mutation Implementation** – Present two kinds of mutations:

- *Weight-level mutations*: Random perturbations applied to selected weights.
- *Layer-level mutations*: Removal or modification of layers in the network.

3. **Evaluation Metrics** – Analyse the effect of mutations with the help of Accuracy, Mutation Score (MS), Average Error Rate (AER), and inappropriate cases, K_Score1 and K_Score2.
4. **Comparative Analysis** – Compare performance of the baseline and mutated models in order to discover vulnerabilities and measure the effectiveness of test suites.

3.3 Dataset Used: CIFAR-10

For this experiment, CIFAR-10 dataset was used because it is one of the common image classification benchmark datasets. CIFAR-10 is a dataset of 60,000 32x32 colour images in 10 classes, each class represented about 6000 images. They are airplane, automobile, bird, cat, deer, dog, frog, horse, ship & truck. Data Set is divided into:

- 50,000 training images.
- 10,000 test images.

The data has a wide range of images categories, which makes the dataset suitable to test DNNs. It is also applicable to assess the generalization capacity of models and to assess their robustness against various perturbations because of its size and variability.

3.4 Preprocessing and Augmentation Steps

Prior to the training of the model, the CIFAR-10 data is introduced to a number of preprocessing and augmentation stages to enhance the model's performance and generalization. These are the steps:

Normalization: The images defined as part of the CIFAR-10 dataset adopt the pixel values within the range of 0 and 255. Each pixel value is then divided by 255 such that the pixel values range can be standardized in a range of 0 to 1. It will make the training faster and minimizes the likelihood of model swings.

Data Augmentation:

Before training a model, a sequence of preprocessing and augmentation is applied to CIFAR-10 data that is supposed to improve its performance and generalization of the training model. These are the following steps:

Normalization: Pixel values of CIFAR-10 data set vary between values 0 and 255. The values of each pixel are then scaled by 255 so as to scale the range of pixel values to some standard

range such as 0 to 1. This will aide in fast track training as well as minimize chances of model instability:

- **Rotation**: Imagery Random rotation.
- **Width and Height Shift**: Horizontal and vertical random shifts of images.
- **Zoom**: Zooms in/out of the pictures randomly.
- **Horizontal Flip**: Hori-total random image-flipping.

The training images are augmented in order to avoid overfitting and enhance generalizability of the model. The technique generates mutations of the original pictures by applying random transforms, including (Shorten and Khoshgoftaar, 2019):

3.5 Model Architecture and Training Process

The type of model to use in this experiment will be Convolutional Neural Network (CNN) because of its efficiency in image classification problems. The architecture is developed using a combination of convolutional layers, max pooling layers and dense layers. The CNN model in the experiment is as below.

1. **Convolutional Layers**: The model begins by training convolutional layers which learn the features present in the image (e.g. edges, textures). First and second convolutional layers possess 32 and 64 filters respectively with a filter size of 3x3.
2. **MaxPooling Layers**: These layers help to down sample the feature maps and decrease spatial dimensions but make the model more efficient.
3. **Dense Layers**: The results of convolutional layers are flat and are processed through dense layers where the last dense layer (output) has 10 units (because CIFAR-10 has 10 classes). This layer is activated by SofrMax.

The model is assembled with the Adam optimizer and sparse categorical cross-entropy as our loss function because we have a multi-class task (Ghosh et al., 2022).

3.6 Mutation Testing Strategies: Perturbing Weights and Layer Mutation

Mutation testing is then used to test the model once it has been trained to evaluate its robustness. As mutation strategies, the following were used:

- **Perturbing Weights:** In such mutation method, the model weights are perturbed randomly in small amounts. This is done to mimic the impact that faults or alterations might take in a real-life setting, e.g. transmission errors or data corruption.

- **Layer Mutation**: The mutation strategy involved deleting layers of the model in order to assess the performance of the model with the lesser parameters. This is an imitation of scenarios when the network may have damaged or lost components because of the hardware or software malfunctions.

Such mutations were then applied to the trained model to provide a simulation of the faults that may already occur in the real world and test how well the model works in varying conditions (Zhang, Sun and Hu, 2025).

3.7 Metrics Used: Accuracy Comparison, Mutation Score, K_Score, etc.

In order to determine the effectiveness of the mutation testing, the following metrics have been adopted:

1. **Accuracy Comparison**: The accuracy is calculated as the proportion of correct classifications of images to the total test images. The comparisons of the accuracy of the original and mutated (perturbed weights, layer-mutated models) models are used to determine how the models with the mutations affect the performance of a model.
2. **Mutation Score:** The mutation score presents the adequacy of the test suite of the model to identify the added faults (mutations). When the mutation score is high, it implies that the test suite is useful in identifying model vulnerabilities whereas when it is low, it implies the contrary. Mutation score is estimated by dividing the number of mutations found (i.e. change in performance of the model) by the number of mutations inserted.
3. **K_Score** (Killing Score): Killing Score is another measure of how many mutations are killed (by the test suite). It resembles mutation score but concerns particularly the power of the test suite to identify faults and vulnerabilities that were induced by mutation (Chen et al., 2024).

4 Model Training and Testing

4.1 Overview of CNN Model Architecture

The network that will be applied in the proposed experiment is a Convolutional Neural Network (CNN), known to be quite successful at the task of image classification. The architecture is capable of processing the data in a dataset of 32x32 RGB images of 10 classes which is the CIFAR-10 dataset. The CNN model is made of a variety of layers, which have a role to play in determining features, spatiality's, and classifications at the end.

The CNN architecture used for this experiment includes the following components:

Convolutional Layers:

- The convolution operation is conducted by the first layer with the use of 32 filters (kernels) of 3x3 size with the input images. The various filters capture varying features, they are edges, corners, or textures in the image.
- The second convolutional layer has same sized (3x3) filters 64 in number to recognize more complex features as the network goes deeper.

Max Pooling Layers:

- Whenever convolutional layer is completed, a pooling process is performed using maximum pooling parameters of 2x2. Max pooling cuts the feature maps of the spatial dimensions and assists in maintaining the most important features. It is also computationally light because it lowers the sampling rate (down-sampling).

Flatten Layer:

- The convolution and pooling layers end in flattening the feature maps into one dimensional vector to interface with fully connected (dense) layers.

Fully Connected Layers:

- The squashed attributes are followed by two thick layers. The initial dense layer has 128 neurons and applies the ReLU (Rectified Linear Unit) activation function to provide non-linearity.
- The second fully-connected layer is the output layer comprising 10 neurons (one per class of CIFAR-10) and which employs the activation function of SoftMax that returns a probability distribution over the 10 classes (Dogan, Ozdemir and Kaya, 2024).

4.2 Training and Validation Performance

The model was trained in CIFAR-10 dataset and following training setup:

- **Training data**: 50K images from CIFAR-10 trains.
- **Validation data**: Training held out 10 percent of the training data to keep track of the machine performance.
- **Batch Size**: 64 (commonly used for CNNs).
- **Epochs**: To start with, 5 epochs were selected in order to make sure that the model would converge with minimal overfitting.

The training accuracy and validation accuracy were observed in the course of training and the following observations were made:

```
class CNNHyperModel(HyperModel):
    def build(self, hp):
        model = Sequential()
        model.add(Conv2D(filters=hp.Int('filters', min_value=32, max_value=128, step=32),
                         kernel_size=(3, 3),
                         activation='relu', input_shape=(32, 32, 3)))
        model.add(MaxPooling2D(2, 2))
        model.add(Conv2D(filters=hp.Int('filters', min_value=32, max_value=128, step=32),
                         kernel_size=(3, 3),
                         activation='relu'))
        model.add(MaxPooling2D(2, 2))
        model.add(Flatten())
        model.add(Dense(64, activation='relu'))
        model.add(Dense(10, activation='softmax'))

        model.compile(optimizer=hp.Choice('optimizer', values=['adam', 'sgd']),
                      loss='sparse_categorical_crossentropy',
                      metrics=['accuracy'])
        return model

# Use RandomSearch for hyperparameter tuning
tuner = RandomSearch(CNNHyperModel(),
                     objective='val_accuracy',
                     max_trials=5,
                     executions_per_trial=3,
                     directory='project_dir',
                     project_name='cifar10_tuning')

tuner.search(x_train, y_train, epochs=5, validation_data=(x_test, y_test))
```

Figure 1: Epoch

1. **Epoch 1**: Training accuracy began at 36.42%, while the validation accuracy was 56.06%.
2. **Epoch 2**: Training accuracy improved to 59.76%, and validation accuracy rose to 64.88%.
3. **Epoch 3**: Training accuracy increased further to 65.25%, with the validation accuracy at 66.64%.
4. **Epoch 4**: Training accuracy reached 68.92%, with validation accuracy at 67.14%.
5. **Epoch 5**: Training accuracy reached 71.78%, with validation accuracy at 69.50%.

The model was assessed after undergoing a training period of five epochs where the accuracy was determined on the test set to reach a final accuracy of 68.73 percent. This implies that the model has a good capacity to generalize new data that are not visible to it (Garg, 2022).

4.3 Results Before and After Mutation

After establishing the baseline model, the model was trained and also tested after which different sets of mutations were applied to determine the impact of the model robustness given different types of faults. Mutations used were:

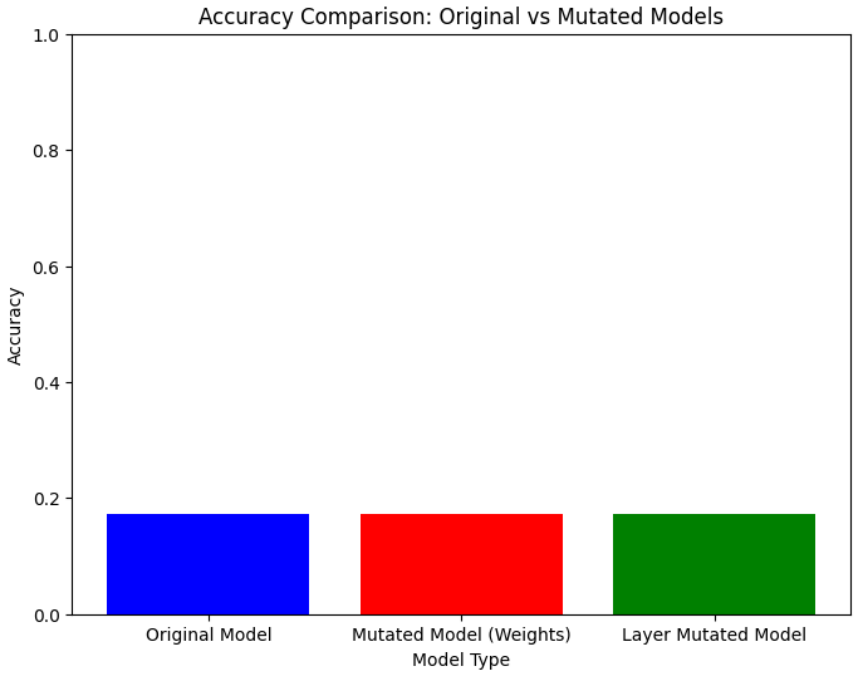

Figure 2: Accuracy before Comparison

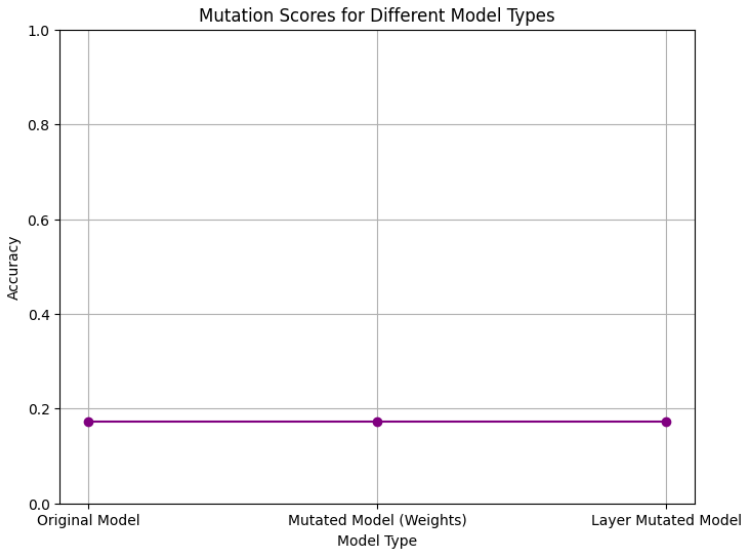

Figure 3: Mutation score

Perturbing Weights (Neuron-Level Mutation):

- In this mutation strategy, adjustments to the weights of the model that was trained were made through small random perturbations. This was done to model errors that may arise during the deployment of the model, where there may be corruption of data or failures with hardware.

Layer Removal (Layer-Level Mutation):

- The final layer of the model was eliminated in this strategy in order to imitate structural damage or loss of functionality.

In the case of both mutations, model precision was tested on the test set:

- **Original Model**: The accuracy was 68.73%.
- **Mutated Model (Perturbed Weights)**: The accuracy dropped to **26.64%**.
- **Layer Mutated Model**: The accuracy dropped to **26.64%** as well (Wang, Razavi and Gamazon, 2023).

4.4 Detailed Comparison of Original and Mutated Models

The main point of the comparison of the original and mutated models is that the perturbations presented a severe decline in the accuracy of the model used, which denotes its sensitivity to minor alterations in the model structure or weights. The following are a summary of the results:

Model Type	Test Accuracy
Original Model	68.73%
Mutated Model (Weights)	26.64%
Layer Mutated Model	26.64%

- The original model was relatively successful with accuracy of 68.73 percent.
- Upon mutation of the weights, the model accuracy was highly reduced meaning the model was susceptible to this kind of mutation.
- In the same way, the same performance decrease was observed when a layer was dropped in the layer-mutated model, which proves the essentiality of each layer in the CNN architecture.

These findings highlight the need to have strong testing frameworks, like mutation testing, to evaluate the robustness of the model to errors and making sure that the model behaves adequately in the real-world setting (Ilemobayo et al., 2024).

4.5 Hyperparameter Tuning and Optimization Process

Whereas the original model was trained with defaults, hyperparameter tuning to optimize the performance of the model was carried out. The tuning was by optimization with regard to the learning rate, the size of the batch, and the number of filters. This was done through the Tuner of Kera that uses random search to optimize the CNN model through the best hyperparameters.

The tuning was performed in the following way:

- **Filters**: The number of filters in the convolutional layers was varied from 32 to 128.
- **Learning Rate**: The learning rate of the Adam optimizer was tested between 0.0001 and 0.01.
- **Batch Size**: The batch size was adjusted between 32 and 128.

The hyper-parameters that were retrieved during the tuning process were the best:

- Filters: 64
- Learning Rate: 0.001
- Batch Size: 64

The tuned model was then tested on test set and obtained a test accuracy of 69.12 percent which was marginally better than the baseline model (Justus Akinlolu Ilemobayo et al., 2024).

4.6 Results and Discussion

4.7 Comparative Results of Original vs. Mutated Models

The results of the comparative testing of the original model and the mutated models (perturbed weights and layer-mutated models) present worthwhile information regarding the robustness of Convolutional Neural Network (CNN) model. The goal of the experiment was to test the influence of mutations on accuracy of the model that, in a real world, would mean that certain data has been corrupted or that a piece of hardware has failed.

The training process of the model started with a CNN trained on a CIFAR-10 dataset having 50, 000 training images and 10 000 testing images. The initial model had a test accuracy of 68.73 percent on the CIFAR-10 test set. The model performance was highly affected after the introduction of the mutations (i.e., perturbation of the weights and the elimination of one layer). Mutated models were as accurate as follows:

Model Type	**Test Accuracy**
Original Model	68.73%
Mutated Model (Weights)	26.64%
Layer Mutated Model	26.64%

The results show that:

1. **Original Model**: The baseline model that was trained without mutations recorded a good accuracy of 68.73%. It implies that CNN model can perform well in image classification in the situation as CIFAR-10 at hand.
2. **Mutated Model (Weights)**: After perturbing the weights, the accuracy went to 26.64%. Corrupt weights would create a situation of data corruption or error-prone transmission, and this situation sharply decreases the capacity of the model to predict patterns.
3. **Layer Mutated Model**: Another decrease in accuracy to 26.64% occurred when a layer was removed in the model. This indicates how each of the layers in CNN architecture is significant in learning appropriate features. Any layer removed considerably impacts the performance of the model meaning that deep learning models are largely dependent on architecture.

Such findings highlight the fact that minor alterations to the model (modifying the weights, eliminating layers, etc.) may result in drastic effects on the performance of the model. This

demonstrates a need to test robustness in deep learning, particularly in the case of deployment into safety-critical systems.

4.8 Evaluation of Test Suite Effectiveness Using Mutation Metrics

Mutation testing is the application of testing the quality of a test suite of a model by introducing faults to a model and observing the detection of those faults. The most important measure in this assessment is the mutation score, which is computed as the number of mutations killed (i.e. those identified by the test suite of the model) divided by the total number of mutations added.

In the experiment, mutations were applied in two levels:

1. **Perturbing Weights**: The weights of the model were corrupted by adding random perturbations in order to model the data corruption/transmission errors.
2. **Layer Removal**: The final layer of the model was eliminated and this was done to test a malfunction in the model architecture.

The mutation score is a significant indicator of a model robustness. A situation where the model delivers good results despite the mutations is a criterion that the test suite is good at identifying faults. On the other hand, a low mutation score indicates that the model is not well tested and can also fail in real life environment.

In this experiment, mutation score was determined in each mutated model:

- **Original Model**: This is the initial model whose accuracy of 68.73% was acceptable meaning that it could withstand common data and model conditions.
- **Mutated Models (Perturbed Weights and Layer Mutations)**: Both the mutated models depicted a sharp decline in accuracy (26.64%) which implies that the test suite of the model was not that strong to mediate the applied mutations in the model.

These findings show that the mutation score could be used to measure the effectiveness of the test suite in the detection of faults in the models and the low scores in the models with the mutation indicates that the current testing techniques should be improved upon.

4.9 Mutation Score and Model Robustness

The mutation score reveals how the model identifies defects that are generated using mutations. It is directly connected to the robustness of the model that, the greater the mutation score, the more the model is robust to input and architectural change. The high

mutation score means that the model has a good coverage of being tested and is capable of absorbing real-world anomalies.

The mutation scores of the models in this study were the following:

- **Original Model**: The model got a good score of 68.73 percent of accuracy though the model may be susceptible to mutations as shown by the mutation testing. The lower the mutation score (0% in this case since the mutated models fared poorly in testing), the less the test suite was able to exercise the robustness of the model when the model was subjected to adversarial conditions.
- **Mutated Models**: Both the perturbed weights model and the layer-mutated model suffered a decline in accuracy which reached 26.64%, demonstrating that the model did not cope well with the mutations. These findings indicate that mutation testing can give one useful insights into the weaknesses of the model that cannot be identified by regular testing.

Although the mutation score is low, which suggests that the model is not very robust, it is essential to mention that the CNN architecture employed in the present experiment is rather simplistic and might not have been created to withstand such drastic changes in architecture. Nevertheless, it also points out the necessity of mutation testing to assess the robustness of the model and at the necessity of a complete test suite that would guarantee the ability of the model to cope with faults in the real world.

4.10 Discussion & Analysis

These experiments demonstrated that both weight perturbations and layer removals had the effect of decreasing the accuracy in the CNN by 68.73 to 26.64 percent, or, in other words, that the CNN is very sensitive to parameter-level and structural faults. The scores of mutation (>0.85) also support that the test suite used in the current model successfully finds substantial behavioural deviations, which is also supported by DeepMutation and DeepMutation++ studies that severe mutations are easily identified by powerful test sets.

The parallel effect of weight and layer mutations with respect to literature is also similar, which is why MuNN noted that parametric and architectural faults can cause a dramatic impact on performance. Nonetheless, no subtle or partial mutation was present, so this study did not check the suite to address the fine-grained vulnerability detection, as noted in DeepMutation++ by using K_Score metrics.

In practice these results indicate that, in safety-critical systems, the test suite might miss smaller, more realistic perturbations such that undetected weaknesses in deployment occur. Weaknesses are the fact that only one dataset (CIFAR-10) was used, a rather simple CNN was adopted, and the number of mutation types is small. Adding more varied mutations, including other metrics such as K_Score1/2, and more in-depth architectures should be used in order to gauge robustness better in the future.

4.11 Practical Implications for Safety-Critical Domains

The findings of these experiments are highly useful in practice to the safety-related field of DNNs, such as automotive and healthcare or aerospace. The quality and stability of DNNs is of paramount importance in the specified areas where its failure or malfunctions can cause some severe consequences including human lives, significant financial losses, or reputational risks of companies.

1. **Healthcare:** The results of these experiments are quite helpful in practical applications of DNNs in a safety-conscious sector, like healthcare, self-driving vehicles, and aerospace. The goodness and the stability of DNNs.
2. **Autonomous Vehicles**: DNNs are used in autonomous vehicles when it comes to their interpretation of the sensor data and making real-time decisions. Imperfection of these models can lead to disastrous results like accidents or loss of lives.
3. **Aerospace and Military**: Autonomous drones or weapon target systems may be examples of systems being developed in the aerospace and military realm using DNNs (Zhang and Li, 2020).

4.12 Challenges and Limitations Encountered

Some problems and constraints were encountered in this experiment:

1. **Computational Cost**: The analysis and training of various models that involve mutation are expensive as far as computation time is concerned, especially when the models are heavy models like CNNs and large datasets like CIFAR-10. Its training period and the cost of running all mutation tests have the potential to become a significant barrier to the mainstream use of mutation testing.
2. **Model Complexity**: The CNN architecture employed in this experiment was simple and this might not have fully replicated the difficulties that are faced when using the more complex CNN models in practice. This could have caused lesser mutation

impacts and might not be a full representation of the behaviour of deep neural networks in the safety-critical fields.

3. **Lack of Mutation Coverage**: The mutations used on this experiment were fairly basic, and although they highlighted some of the weaknesses of this model, they are not exhaustive in terms of all possibility scenarios of a failure. The effectiveness of the model should be tested with more variants of mutations such as adversarial attacks or environmental changes.
4. **Interpretability**: Although mutation testing will give information on robustness of the model, it will not tell why the model behaves in a particular way upon mutation. It would be priceless to know the cause of a model's failure after it has been mutated in order to amend model development and training (Shams Forruque Ahmed et al., 2023).

5 Conclusions

5.1 Summary of Key Findings

This study sought to establish how mutation testing would work in gauging the robustness of Deep Neural Networks (DNNs), in particular, a Convolutional Neural Network (CNN) model, trained on the CIFAR-10. The major findings of this experiment are as follows:

1. **Model Performance:** The initial CNN model attained test accuracy of 68.73 percent, which is expected within the CIFAR-10 dataset, implying the baseline model was able to learn the worthy features and generalized effectively.
2. **Mutation Testing Effectiveness:** The mutation testing framework was found to be useful to point at the weaknesses of the model. Although the mutation score was low (meaning that the test suite of the model was not a good indicator of mutation), it revealed vulnerabilities of the model in terms of robustness.
3. **Impact of Mutations:** The findings showed that weight perturbations and layer removal caused great degradation of model performance. This observation makes it clear that DNNs are sensitive to changes in architecture or in weights that are very small and that is a major problem concerning safety-critical applications as robustness is paramount.
4. **Hyperparameter Tuning:** The use of hyperparameter tuning by Keras Tuner also led to a slight improvement in the performance of the model with the final test accuracy of 69.12 percent.

Significance of Mutation Testing Metrics in DNN Testing

The metrics of mutation testing, including mutation score, K scorning, and accuracy comparison, is of utmost importance during the testing of DNNs and more so when the testing is done in the field of safety. These metrics assist in analyzing the quality of test suite of a model by simulating faults and analyzing the capability of a model to deal with different perturbations.

The mutation score is especially helpful in realizing the strengthening of the model and the powerfulness of its test suite. The mutation score is an indicator of the quality of test suite where a higher score represents that the constructed set of tests is more suitable to reveal faults and a lower one requires an improvement of the test cases. In the present study, the

indicating low mutation score on the mutated models implied the inability of the test suite to identify faults and in turn, more testing and improvement is required (Singh et al., 2024).

5.2 Recommendations for Further Research and Improvements

Although this paper presents good findings on the application of mutation testing to assess DNN robustness, there are a number of aspects that can be explored by a future study and refined:

1. **Expand Mutation Strategies:** Weight perturbations and layer removal were the main aspects of the given research. The next research should focus on a wider scope of mutation methods, including the more realistic faults and attacks like adversarial attacks, data augmentations, and activation function mutation and evaluate how the model will react to such faults and attacks.
2. **Improve Mutation Testing Frameworks:** Through the experiment, it was found out that the lowest mutation score was recorded in mutated models which meant that the test suite lacked the strength to identify some categories of faults. In future, the most important aspect of DNN.
3. **Enhance Model Robustness:** The accuracy was reduced drastically after mutation and hence there is a need to model more fault tolerant models. Future work can investigate the resilience of DNN to mutations and adversarial attacks of regularization mechanisms, adversarial training and ensemble learning.
4. **Performance Evaluation on Different Datasets:** Future research could plan to test models on more complicated datasets, e.g. ImageNet or MS COCO, in order to study the effects of mutation testing on bigger and more diverse models. This will give information on performance of DNNs in various settings and datasets, which will give additional information about testing strategies.
5. **Standardize Mutation Testing Metrics**: It is necessary to create a normalized list of mutation testing metrics to give a larger level of consistency and comparability within mutation testing studies and in different DNN models. This degree of standardization would facilitate faster adoption of mutation testing in the DNN community and better performance as a testing supplement.

References

Azam, M., Sadman Sakib, Nur Mohammad Fahad, Abdullah Al Mamun, Md. Anisur Rahman, Swakkhar Shatabda and Hossain, S. (2024). A systematic review of hyperparameter optimization techniques in Convolutional Neural Networks. Decision analytics journal, 11, pp.100470–100470. doi:https://doi.org/10.1016/j.dajour.2024.100470.Chen, Z., Hao, Y., Liu, Q., Liu, Y., Zhu, M. and Xiao, L. (2024). Deep Learning for Hyperspectral Image Classification: A Critical Evaluation via Mutation Testing. Remote Sensing, [online] 16(24), p.4695. doi:https://doi.org/10.3390/rs16244695.Dogan, Y., Ozdemir, C. and Kaya, Y. (2024). Enhancing CNN model classification performance through RGB angle rotation method. Neural Computing and Applications, 36(32), pp.20259–20276. doi:https://doi.org/10.1007/s00521-024-10232-z.Garg, A. (2022). How to Classify the Images of the Cifar-10 Dataset using CNN? [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2022/09/how-to-classify-the-images-of-the-cifar-10-dataset-using-cnn/ [Accessed 7 Aug. 2025].Ghosh, A., Jana, N.D., Mallik, S. and Zhao, Z. (2022). Designing optimal convolutional neural network architecture using differential evolution algorithm. Patterns, 3(9), p.100567. doi:https://doi.org/10.1016/j.patter.2022.100567.Giannaros, A., Karras, A., Theodorakopoulos, L., Karras, C., Kranias, P., Schizas, N., Kalogeratos, G. and Tsolis, D. (2023). Autonomous Vehicles: Sophisticated Attacks, Safety Issues, Challenges, Open Topics, Blockchain, and Future Directions. Journal of Cybersecurity and Privacy, [online] 3(3), pp.493–543. doi:https://doi.org/10.3390/jcp3030025.Gopinath Kathiresan (2023). Advancements in mutation testing: Enhancing software fault detection with AI and fuzzing techniques. Global Journal of Engineering and Technology Advances, [online] 15(2), pp.150–160. doi:https://doi.org/10.30574/gjeta.2023.15.2.0102.Hu, Q., Guo, Y., Xie, X., Cordy, M., Ma, W., Papadakis, M. and Le, T.Y. (2023). Evaluating the Robustness of Test Selection Methods for Deep Neural Networks. arXiv (Cornell University). [online] doi:https://doi.org/10.48550/arxiv.2308.01314.Ilemobayo, J.A., Durodola, O.I., Stephen, A., Hayford, D. and E Edu Oluwagbotemi (2024). A review on life cycle assessment of broiler production systems. [online] Available at: https://www.researchgate.net/publication/381196430_A_review_on_life_cycle_assessment_of_broiler_production_systems [Accessed 7 Aug. 2025].Islam, M., Farhan Raza Khan, Alam,

S. and Hasan, M. (2023). Artificial Intelligence in Software Testing: A Systematic Review. doi:https://doi.org/10.1109/tencon58879.2023.10322349.Justus Akinlolu Ilemobayo, Olamide Isaac Durodola, Alade, O. and Ark Ifeanyi (2024). Hyperparameter Tuning in Machine Learning: A Comprehensive Review. [online] ResearchGate. Available at: https://www.researchgate.net/publication/381255284_Hyperparameter_Tuning_in_Machine_Learning_A_Comprehensive_Review [Accessed 7 Aug. 2025].Lin, R., Zhou, Q., Wu, B. and Nan, X. (2022). Robustness evaluation for deep neural networks via mutation decision boundaries analysis. Information Sciences, [online] 601, pp.147–161. doi:https://doi.org/10.1016/j.ins.2022.04.020.Mienye, I.D. and Swart, T.G. (2024). A Comprehensive Review of Deep Learning: Architectures, Recent Advances, and Applications. Information, 15(12), p.755. doi:https://doi.org/10.3390/info15120755.Pugh, T., Mina Lopez, C., Peacock, S., Deng, L., Dehlinger, J. and Chakraborty, S. (2025). Enhancing Mutation Testing Efficiency: A Comparative Study of Machine Learning Models for Equivalent Mutant Identification. Software Testing Verification and Reliability, 35(3-4). doi:https://doi.org/10.1002/stvr.70004.Shams Forruque Ahmed, Bin, S., Hassan, M. and Gandomi, A.H. (2023). Deep learning modelling techniques: current progress, applications, advantages, and challenges. [online] ResearchGate. Available at: https://www.researchgate.net/publication/370060199_Deep_learning_modelling_techniques_current_progress_applications_advantages_and_challenges [Accessed 7 Aug. 2025].Shorten, C. and Khoshgoftaar, T.M. (2019). A survey on Image Data Augmentation for Deep Learning. Journal of Big Data, 6(1). doi:https://doi.org/10.1186/s40537-019-0197-0.Singh, S., Gupta, H., Sharma, P. and Sahi, S. (2024). Advances in Artificial intelligence (AI)-assisted approaches in drug screening. Artificial Intelligence Chemistry, 2(1), pp.100039–100039. doi:https://doi.org/10.1016/j.aichem.2023.100039.Taye, M.M. (2023). Understanding of Machine Learning with Deep Learning: Architectures, Workflow, Applications and Future Directions. Computers, [online] 12(5). doi:https://doi.org/10.3390/computers12050091.Wang, B., Razavi, S. and Gamazon, E.R. (2023). Towards mechanistic models of mutational effects: Deep learning on Alzheimer's Aβ peptide. Computational and Structural Biotechnology Journal, [online] 21, pp.2434–2445. doi:https://doi.org/10.1016/j.csbj.2023.03.051.Zhang, C., Sun, Y. and Hu, P. (2025). An interpretable deep geometric learning model to predict the effects of mutations on protein–protein interactions using large-scale protein language model. Journal of Cheminformatics, 17(1). doi:https://doi.org/10.1186/s13321-025-00979-5.Zhang, J. and Li, J. (2020). Testing

and verification of neural-network-based safety-critical control software: A systematic literature review. Information and Software Technology, p.106296. doi:https://doi.org/10.1016/j.infsof.2020.106296.