Reproducibility Report of DEEPCRIME: Mutation Testing of Deep Learning Models Based on Real Faults

Abdul Wahab, Lameya Islam, **Saad Abdullah**,
Åbo Akademi University

Reproducibility Summary

Scope of Reproducibility — This paper claims that a mutation testing framework for deep learning (DL) models, called DEEPCRIME, effectively generates realistic faults (mutants) that can help evaluate and improve the robustness of DL test sets. Specifically, the paper asserts that DEEPCRIME's mutation operators produce "killable" and non-trivial mutants, meaning these mutants can expose weaknesses in test sets by simulating real-world DL faults. The goal of our reproduction is to validate these claims by replicating DEEPCRIME's experiments and assessing the effectiveness and killability of selected mutation operators on a DL model.

Methodology — To reproduce the core experiments, we utilized a subset of DEEPCRIME's mutation operators and applied them to the MNIST dataset. Mutations included changes to hyperparameters such as learning rate, batch size, number of epochs, and layer configurations. Each mutation configuration involved retraining the model to introduce faults deliberately, followed by testing to evaluate if these faults ("mutants") were killed, meaning whether the test set could detect the presence of faults. Statistical measures, including mutation scores, were used to confirm the effectiveness of each mutation and to differentiate meaningful changes from random variations. All experiments were conducted on an M1 MacBook Pro 2020 with 8GB RAM.

Results — We reproduced the result of a subset of the total work, particularly on the MNIST dataset tuning some hyperparameters with some selective mutation operators. Where most of the mutations were "killed".

What was easy — The authors provided a Google Colab notebook for an initial code overview, though it was outdated. The original paper's clear explanations of each mutation operator facilitated our reproduction of DEEPCRIME's methodology, particularly for the MNIST CNN model. The external repository, though unofficial, provided a useful starting point aligned with the paper's approach. Additionally, Keras and TensorFlow's well-documented frameworks simplified the setup and execution of mutation testing.

What was difficult — Narrowing down the problem to a manageable reproduction set was challenging due to the high resource demands. The author's code wasn't optimized for modern hardware like M1 Macs, leading to compatibility issues and requiring manual adjustments. Some mutations, like change_Weights, were extremely resource-intensive, taking over 7 hours to run, making it unfeasible to apply all mutations across multiple models. Ultimately, we limited our tests to the MNIST dataset. Additionally, while the authors provided a script to execute all mutations on a model, it frequently crashed, requiring us to run mutations individually.

Communication with original authors — We did not contact the original authors as both the code and dataset were publicly available, allowing us to proceed without additional clarification.

Code is available at https://github.com/wahabmangat/DEEPCRIME

1 Introduction

Deep learning models are increasingly being applied in domains like autonomous systems, speech recognition, and computer vision. Incorporating these models into safety-critical systems demands strict quality assurance, yet conventional software testing methods, such as code coverage, are insufficient. Unlike traditional systems, where performance is closely tied to code structure, DL models' performance depends more on factors like model architecture, hyperparameters, and data quality. As a result, test adequacy criteria specifically suited for DL systems are becoming essential.

Mutation testing, a useful technique for assessing software robustness, involves purposefully introducing "mutations"—small, controlled changes—to create modified versions of the software, known as "mutants." The effectiveness of a test suite is then evaluated based on its ability to detect these changes. The underlying idea is that a test suite capable of identifying mutants—by prompting them to produce different outputs or behaviors from the original program—is likely to be more comprehensive and effective at uncovering real-world faults.

However, existing DL mutation techniques, such as DeepMutation++, have limitations. They rely on arbitrary, post-training alterations that lack grounding in real-world fault patterns observed in DL systems. Addressing this gap, DEEPCRIME introduces mutation operators that simulate actual DL faults documented in empirical studies. These operators inject mutations before training, allowing faults to influence the model's behavior throughout the training process. Additionally, DEEPCRIME employs a statistical mutation killing criterion specifically designed for the stochastic nature of DL models, enhancing its ability to detect meaningful faults.

This report reviews DEEPCRIME's methodology, implementation, experimental results, challenges encountered, and recommendations for future improvements, highlighting its unique approach to mutation testing for DL systems.

2 Scope of reproducibility

The paper claims that DEEPCRIME effectively generates meaningful, real-fault-based mutations for deep learning models, producing mutants that are both killable by competent test suites and non-trivial, thus providing valuable insights into model robustness. By introducing mutations pre-training, DEEPCRIME outperforms post-training mutation tools like DeepMutation++, as its faults closely mirror realistic DL issues, enhancing test adequacy evaluation. Additionally, DEEPCRIME shows high sensitivity in distinguishing between weak and strong test sets, making it a powerful tool for assessing test suite quality. Using a statistical framework to determine mutation killing, DEEPCRIME accounts for the stochastic nature of DL models, ensuring that observed differences stem from genuine faults rather than random training variations.
For reproducibility, we are going to test:

- Reproduction of Mutation Operators
- Evaluation of Killability

3 Methodology

The methodology involves using the MNIST dataset to evaluate the effectiveness of selecting mutation operators from DEEPCRIME. Specific hyperparameters, such as learning rate, batch size, and number of training epochs, are altered to simulate realistic faults in the model's training process. Key mutation operators applied include those that modify training data and hyperparameters. Each mutation configuration is tested by retraining the model on the modified parameters, followed by evaluating the model's performance and the ability of the test set to detect these faults. Statistical analysis, including mutation scores and effect sizes, is used to confirm whether the introduced mutations are effectively "killed," ensuring observed changes are due to the faults rather than random variations.

3.1 Model descriptions

The model used for reproducibility experiments is a convolutional neural network (CNN) implemented in Keras with TensorFlow backend, designed to classify 28x28 grayscale images of handwritten digits into 10 classes (0–9). The model consists of two convolutional layers (32 and 64 filters with ReLU activation), followed by max pooling, dropout for regularization, a fully connected layer with 128 units, and an output layer with softmax activation for class probabilities. With 1,199,882 parameters, the model is trained from scratch using the Adadelta optimizer, categorical crossentropy loss, a batch size of 128, and 12 epochs on the MNIST dataset. Pixel values are normalized to [0, 1] for training stability. This model achieves about 99% accuracy on the test set, providing a strong baseline for mutation testing with DEEPCRIME to evaluate robustness under various fault conditions.

3.2 Datasets

The MNIST Handwritten Digits dataset consists of 70,000 grayscale images of handwritten digits, with 60,000 images in the training set and 10,000 in the test set, covering digits 0 through 9 in a balanced distribution. Each image is 28x28 pixels and assigned a label corresponding to the digit it represents. For preprocessing, pixel values are normalized to the [0, 1] range to standardize input scales, and images are reshaped according to the backend format (either (28, 28, 1) or (1, 28, 28)). The dataset is publicly available and can be accessed directly through the Keras API here.

3.3 Hyperparameters

In these experiments, a select set of hyperparameters are systematically modified to simulate realistic faults in model training and assess the robustness of the test suite. Key hyperparameters include:

- **Learning Rate:** Adjusted to test the model's sensitivity to different step sizes, ranging from small to large learning rates.
- **Batch Size:** Varied across experiments to examine its effect on model stability and generalization.
- **Epochs:** Increased or decreased to evaluate the impact of training duration on model performance and mutation results.
- **Optimizer:** Different optimizers (such as Adadelta, SGD, etc.) are applied as mutation operators to assess how changes in optimization affect the model's convergence behavior.

These hyperparameters are explored manually in targeted trials, rather than through an extensive hyperparameter search. Each trial simulates specific faults to observe resulting behaviors, and a total of six mutation operators were used for the selected configurations, focusing on parameters known to influence model behavior.

3.4 Experimental setup and code

The reproduction of this work involved replicating the experimental setup described in the source paper to validate the claims regarding the effectiveness of DEEPCRIME. The reproduction process involved several key steps:

- **Selcting Subject DL Systems:** One subject system has been selected and that was MNIST dataset
- **Selecting Mutation Operators:** Six mutation operators have been applied to the model:
  1. change_loss_function
  2. change_epochs
  3. add-bias
  4. remove_bias
  5. change_optimize_function
  6. remove_activation_function

The total replication of the code has been implemented through several key steps:

1. **Injecting Mutations into Python Code:** DEEPCRIME uses Python's Abstract Syntax Tree (AST) module to parse the source code of the DL model under test and inject mutations.
    - It identifies specific locations in the code (target nodes) where each mutation operator should be applied. For example, for operators like TRD (Remove Portion of Training Data), it targets calls to the fit or compile APIs.
    - By analyzing the arguments and keywords of these calls, DEEPCRIME can extract information about the model, training data, and hyperparameters.
    - It then modifies these parameters or inserts new code snippets to implement the desired mutations. These modifications could involve:
2. **Generating Mutants:** After modifying the AST, DEEPCRIME unparsed the modified tree back into Python code. This process generates a mutated version of the original DL model, known as a mutant.
3. **Retraining and Evaluation:** The original model and each mutant are retrained multiple times to account for the stochastic nature of DL training. The performance of the original model and the mutant are then evaluated on a given test set.
4. **Statistical Mutation Killing:** DEEPCRIME determines whether a mutant is killed by comparing the performance of the original model and the mutant using statistical analysis. A mutant is considered killed if the difference in performance is statistically significant with a non-negligible effect size.
5. **Mutation Score Calculation:** The mutation score is calculated as the proportion of killed mutants to the total number of mutants generated. A higher mutation score indicates a more effective test suite.
6. **Reporting:** DEEPCRIME generates CSV reports that detail the results of the mutation analysis. These reports include information about:
    - Whether each mutant was killed
    - p-value
    - Effect size

3.5 Computational requirements

For these experiments on an M1 MacBook Pro (2020) with 8GB RAM, the following hardware and performance observations were made:

Hardware Specifications:
Device: Apple M1 MacBook Pro (2020)
CPU: 8-core Apple M1 chip
RAM: 8GB (though the processes often required 18-20GB, utilizing swap memory)
GPU: Integrated Apple M1 GPU

Average Runtimes for Mutations:
Given the hardware limitations, runtimes varied significantly across different mutations. Below is an approximate summary of the time required for each mutation type based on experiment observations:

- **change_loss_function**: ~4-5 hours (high resource demand, longest runtime due to iterative optimization)
- **change_epochs_binarysearch**: ~3-4 hours (binary search increases the complexity of the training loop)
- **add_bias**: ~2-3 hours (adds complexity in model training, especially with multiple runs)
- **remove_bias_mutated**: ~2-3 hours (similar to add_bias but slightly faster)
- **remove_activation**: ~1-2 hours (relatively less intensive, modifies the activation without significant structural changes)
- **change_activation**: ~10 minutes to 1 hour (quickest, minor adjustments in activation without heavy retraining)
- **change_weights_initialisation:** ~8 hours plus (still we were not able to extract full details from it, thats why it's not added in replication report separately)

Total GPU Hours: Not applicable, as this system relied on the integrated M1 GPU.
Total Time Spent (for all experiments): Approximately **18-20** hours in total, accounting for restarts and failures.

Also, we have some additional considerations for Execution:

- **Dataset Complexity**: More complex datasets increase computational demand. For example, MNIST is relatively simple, but using a dataset like UDACITY would require significantly more memory and processing power.
- **Model Architecture**: Models with more layers and parameters need greater resources.
- **Mutation Operator Selection**: Some mutations are more resource intensive. For instance, change_weights_initialization requires retraining from scratch, making it far more demanding than simpler mutations like remove_activation.

It is clear that running DEEPCRIME requires significant computational resources, especially when dealing with complex DL models and large datasets. For better performance recommendations include using a dedicated GPU and 16GB RAM, prioritizing efficient mutation operators, managing dependencies, and using virtual environments to isolate the DEEPCRIME environment and prevent conflicts.

4 Results

The results of this report support the main claims of the original DeepCrime paper, affirming that the mutation operators generated realistic and significant faults in a deep learning model, allowing for effective evaluation of test suite robustness. By systematically applying pre-training mutations, the reproduction experiments demonstrated DeepCrime's capacity to create killable and non-trivial mutants across various configurations. Although constrained by computational resources, the experiments validated the framework's robustness and relevance, particularly on the MNIST dataset.

4.1 Results reproducing original paper

The experiments below support the claims that DEEPCRIMES mutation operators produce realistic, significant faults in the DL model, allowing for effective evaluation of test suite robustness through mutation scores. Each experiment's results are grouped by mutation operator, highlighting the mutation's impact on model performance and whether the test suite successfully detected (killed) the mutant.

Result 1 — Mutation Operator: Change Loss Function

This experiment tested multiple loss functions to observe how changes impacted the model's performance and whether the test suite could detect the mutation.

Loss Function	p-value	Effect Size	Mutation Killed
mean_squared_error	0.0	6.171491020054803	True
mean_absolute_error	0.0	13.480828169150435	True
mean_absolute_percentage_error	0.0	3.643699702233569	True
mean_squared_logarithmic_error	0.0	14.745202198947892	True
squared_hinge	0.0	6.060443393347542	True
hinge	0.0	18.55853565062475	True
categorical_hinge	0.0	4.67404825447717	True
logcosh	0.0	10.510518782430774	True
huber_loss	0.0	14.354362846378017	True
categorical_crossentropy	0.048	0.8855227227001907	True
binary_crossentropy	0.0	3.2584297012929837	True
kullback_leibler_divergence	0.0	2.695556940962461	True
poisson	0.0	4.754570036091073	True

Each loss function variation resulted in a statistically significant effect size, and all mutations were killed, supporting the claim that loss function changes create noticeable, killable mutations.

Result 2 — Mutation Operator: Change Epochs (Binary Search)

This experiment used binary search to alter the number of epochs and observe the model's performance sensitivity to training duration.

Epoch Configuration (Start-Stop)	p-value	Effect Size	Mutation Killed
12–1	0.0	3.8356167471328404	True
12–1–6	0.003	1.3091030496610165	True
12–6–9	0.001	1.5498335870285938	True
12–9–10	0.015	1.088872738380244	True
12–10–11	0.03	0.9679548444643971	True

All tested epoch configurations resulted in statistically significant killability, validating that altering training duration impacts model behavior meaningfully.

Result 3 — Mutation Operator: Remove Bias (No Search)

This mutation removed bias from specific layers and evaluated the impact on model robustness.

Bias Removal Setting	p-value	Effect Size	Mutation Killed
Setting 1	0.002	1.4137726562552755	True
Setting 2	0.004	1.2898540760441835	True
Setting 3	0.0	-6.196900116744196	False

While most settings killed the mutation, one configuration resulted in a non-killable mutation, showcasing variability in how bias impacts model performance.

Result 4 — Mutation Operator: Remove Activation Function (No Search)

Removing activation functions in certain layers was tested to evaluate its influence on model output and robustness.

Activation Removal Setting	p-value	Effect Size	Mutation Killed
Setting 1	0.02	1.036380296705971	True
Setting 2	0.0	2.203495103984147	True
Setting 3	0.0	-5.520972464356253	False

Similar to the bias operator, removing activation functions generally resulted in killable mutations, with some non-killable instances indicating configuration sensitivity.

Result 5 — Mutation Operator: Change Weights Initialization (Exhaustive Search)

This mutation tested different weight initialization strategies to observe its effect on model stability.

Initialization Method	p-value	Effect Size	Mutation Killed
zeros	0.0	720.9834377196854	True
ones	0.0	2.458623979578215	True
constant	0.0	720.9834377196854	True
random_normal	0.0	2.187761013110443	True
random_uniform	0.0	2.0231286308769807	True
truncated_normal	0.003	1.3277887211042372	True
orthogonal	0.479	0.3168204855828206	False
lecun_uniform	0.868	-0.07405330178906498	False
glorot_normal	0.066	0.8218851523073014	False
glorot_uniform	0.024	1.0120456884636726	True
he_normal	0.459	-0.3311378420049132	False
lecun_normal	0.056	-0.8533000293591885	False
he_uniform	0.2	-0.5726985056149041	False

Initialization methods like zeros, constant, and random consistently produced killable mutations, while methods like orthogonal and lecun initialization were less killable, showing that initialization choice significantly affects killability.

Overall, the results from these experiments align with the main claims of the original DEEPCRIME paper. Each mutation operator produced statistically significant faults in the DL model, supporting DEEPCRIME's methodology of using meaningful, pre-training mutations to assess model robustness.

4.2 Results beyond original paper

To complement the original experiments, additional tests were conducted with minor adjustments to certain hyperparameters that were unspecified in the paper. This included varying the layers targeted by specific mutation operators and experimenting with different batch sizes. These adjustments aimed to assess whether slight changes in configuration would influence the mutation outcomes.

Additional Result 1 — Hyperparameter Tuning

The results showed minimal differences in mutation scores and killability, indicating that the overall robustness and behavior of the mutations remained stable across these variations. These findings suggest that the DEEPCRIME framework's effectiveness in generating meaningful faults is not highly sensitive to these particular hyperparameter tweaks.

5 Discussion

The experimental results largely support the primary claims of the DEEPCRIME paper, demonstrating that the framework effectively generates meaningful, killable mutations based on realistic deep learning faults. The results indicate that DEEPCRIME's mutation operators create variations in model behavior that can be reliably detected by a competent test suite, highlighting its utility for robustness evaluation in deep learning models.

Strengths: The approach adhered closely to the original methodology, replicating key mutation operators on the MNIST model and observing consistent mutation outcomes across various configurations. Additional experiments, such as testing layer-specific mutations and varying batch sizes, showed minimal impact on the results, further reinforcing the framework's claims regarding mutation robustness. These findings suggest that DEEPCRIME's approach is versatile, effectively evaluating model resilience under various fault conditions, even with slight configuration differences.

Weaknesses: Due to resource constraints, the scope of replication was limited to a single model (MNIST) and a select number of mutation operators. While this subset provided valuable insights, replicating additional models and operators would provide a more comprehensive validation of DEEPCRIME's generalizability across different architectures and datasets. Furthermore, codebase issues occasionally required troubleshooting, which limited the time available for running extended experiments. Future work could focus on broader replication with varied datasets and models to further validate the framework's scalability and effectiveness across different domains.

In summary, the replicated experiments provide strong support for the claims in the paper, affirming DEEPCRIME's approach to generating meaningful, statistically significant mutations for deep learning models.

5.1 What was easy

The reproducibility of DEEPCRIME's methodology was facilitated by the clear and well-structured explanations provided in the original paper, especially regarding the purpose and functionality of each mutation operator. This detailed explanation made it straightforward to implement mutation operators and apply them to the selected model (MNIST CNN). Additionally, the repository based on DEEPCRIME provided a useful starting point, containing code that generally followed the paper's methodology. Although not created by the paper's authors, the repository still outlined the key mutation operations, making it easier to set up and execute basic experiments.

Another factor that contributed to the ease of reproduction was the use of Keras and TensorFlow, which provided familiar and accessible frameworks for deep learning experiments. The API documentation for these libraries is extensive and detailed, and this familiarity allowed for a smoother setup of the core CNN model and training process, ensuring that the primary aspects of the mutation testing workflow were straightforward to reproduce.

5.2 What was difficult

Several aspects of reproducing DEEPCRIME proved challenging, mainly due to compatibility and resource constraints. First, the repository was not authored by the original paper's creators, which introduced dependencies on external code that contained bugs and issues. Debugging these issues and adjusting the code to work on Apple M1 Macs required additional effort, particularly when compatibility issues arose with Keras imports. The standalone Keras imports needed to be replaced with TensorFlow's Keras module for compatibility, as the original imports did not run on M1 hardware.

Another significant challenge was the computational resource requirement. Many mutation operators demanded substantial memory (often 18-20GB of RAM), which exceeded the capacity of the M1 MacBooks. This limitation led to frequent reliance on swap memory and a marked slowdown in processing. Certain mutations, like

change_loss_function, required extensive runtime (4-5 hours) and occasionally failed due to missing files at the end of execution, further increasing the time needed for experiments. Finally, some subjectivity in interpreting the operator designs from the paper introduced complexity, as it was occasionally unclear how specific faults should be translated into code mutations, requiring trial and error to match the paper's intended approach.

5.3 Communication with original authors

We did not directly communicate with the original authors, as both the code and dataset were readily available. For code-related issues, particularly package compatibility, we chose to create GitHub issues on their repository. This allowed us to document challenges and potentially assist other users facing similar problems.

References

1. PreCrime Project Website. 2021. PreCrime: Prediction, Prevention and Reduction of Crime through Artificial Intelligence and Big Data Technologies. [Online]. Available: https://www.pre-crime.eu/
2. Humbatova, N., Jahangirova, G., & Tonella, P. (2021). DEEPCRIME: Mutation Testing of Deep Learning Systems Based on Real Faults. Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA). Association for Computing Machinery. Retrieved from https://huang.isis.vanderbilt.edu/cs8395/readings/testing-dl.pdf
3. Gunel Jahangirova and Paolo Tonella. An empirical evaluation of mutation operators for deep learning systems. In IEEE International Conference on Software Testing, Verification and Validation, ICST'20, page 12 pages. IEEE, 2020.