DeepMutation: Mutation Testing of Deep Learning Systems

Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, Yadong Wang

Arxiv 2018

Introduction

Without a systematic way to evaluate and understand the quality of the test data, it is difficult to conclude that good performance on the test data indicates the robustness and generality of a DL system.

New evaluation criteria on the quality of DL systems are highly desirable and the quality evaluation of test data is of special importance.

Mutation testing measures the quality of tests by examining whether a test set can reveal certain types of defects that are injected into the original software under test (SUT).

The training data set and the structure of DNNs are two major sources of defects of DL systems.

Introduction

After the faults are injected, the training process could be re-executed, using the mutated training data or training program, to generate the corresponding mutated DL model.

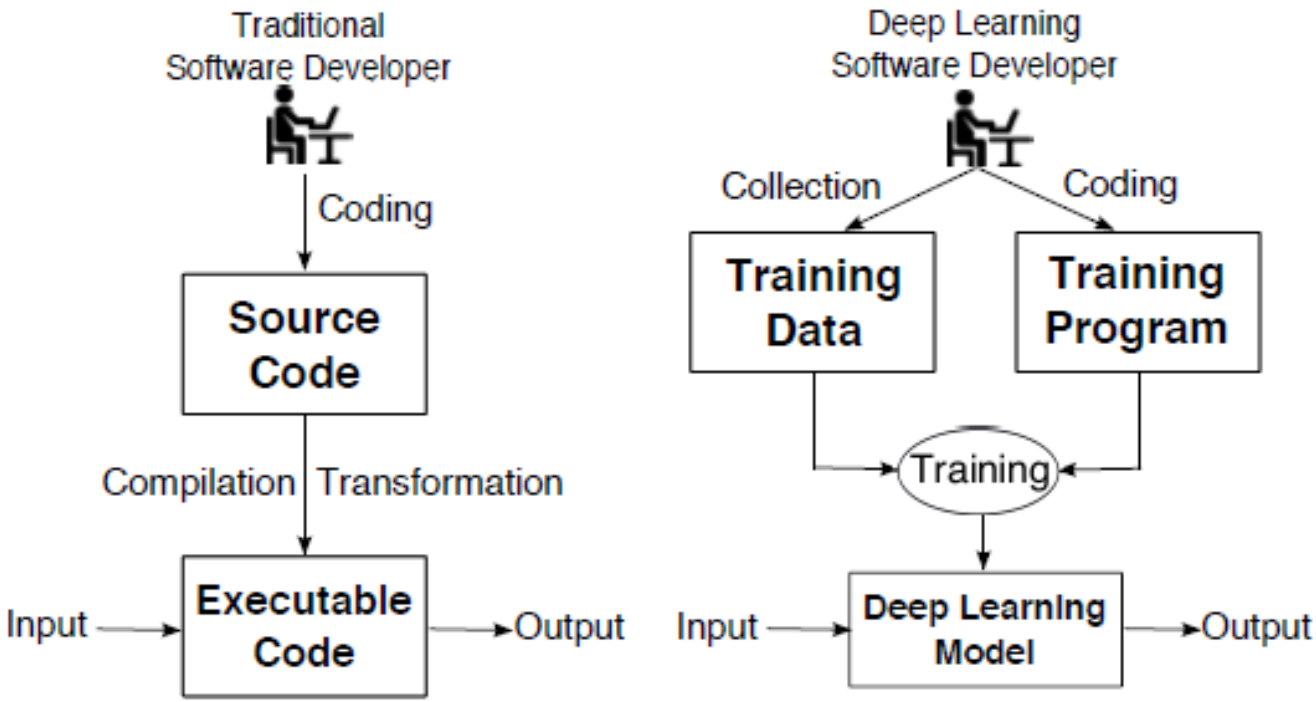
They propose a framework for mutation testing for DL systems:

- First they design eight source level mutation testing operators that directly manipulate the training data and training programs.
- They further design eight mutation operators to directly mutate DL models.

Background

Programming Paradigms

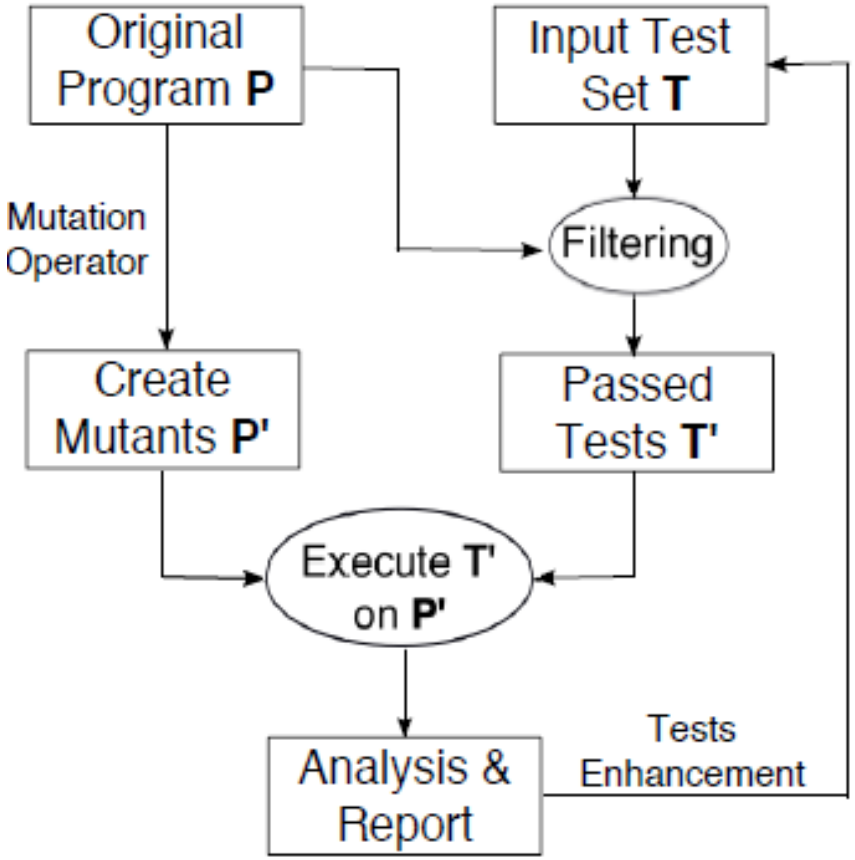
Building deep learning based systems is fundamentally different from that of traditional software systems.



In contrast to traditional software, DL models are often difficult to be decomposed or interpret, making them unamenable to most existing software testing techniques.

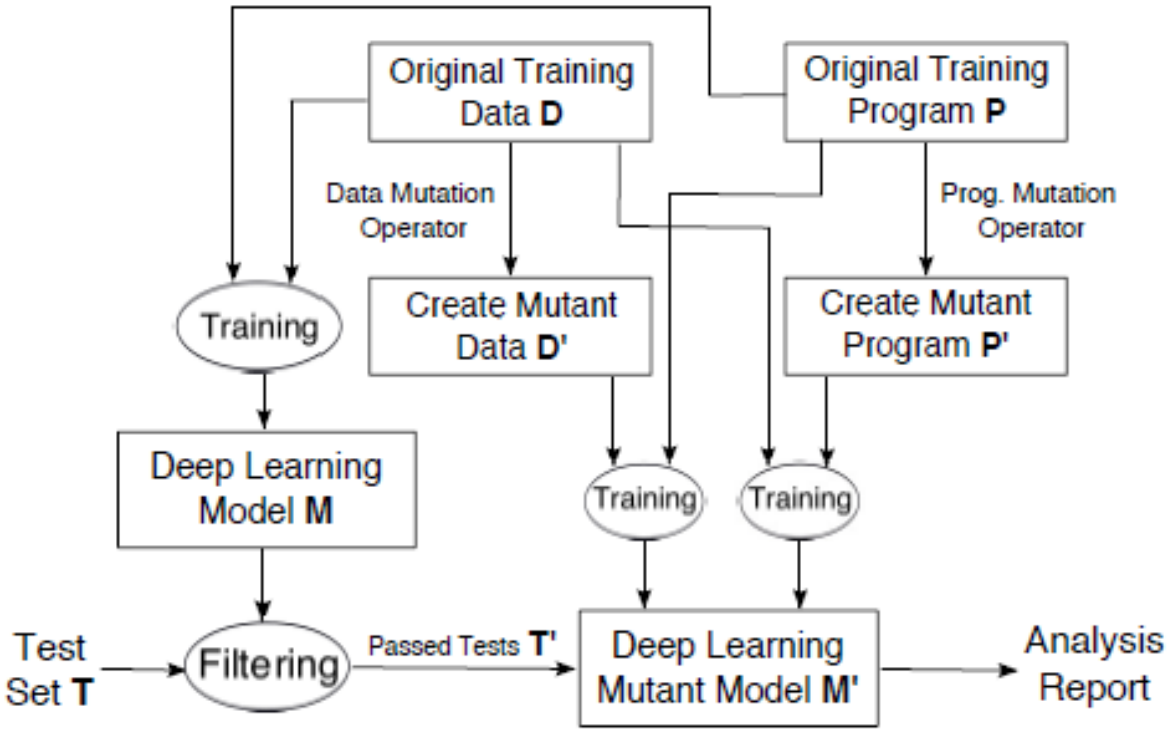
Background

Mutation Testing



Source-Level Mutation Testig of Programs

With the same spirit of mutation testing for traditional software, directly introducing defects into the programming sources of a DL system is a reasonable approach to create its mutated DL models, which simulates the bugs introduced by human developers.



Source-Level Mutation Testig of Programs

They propose two groups of mutation operators, namely *data mutation operators* and *program mutation operators*.

SOURCE-LEVEL MUTATION TESTING OPERATORS FOR DL SYSTEMS.

Fault Type	Level	Target	Operation Description
Data Repetition (DR)	Global	Data	Duplicates training data
	Local		Duplicates specific type of data
Label Error (LE)	Global	Data	Falsify results (e.g., labels) of data
	Local		Falsify specific results of data
Data Missing (DM)	Global	Data	Remove selected data
	Local		Remove specific types of data
Data Shuffle (DF)	Global	Data	Shuffle selected training data
	Local		Shuffle specific types of data
Noise Perturb. (NP)	Global	Data	Add noise to training data
	Local		Add noise to specific type of data
Layer Removal (LR)	Global	Prog.	Remove a layer
Layer Addition (LA_s)	Global	Prog.	Add a layer
Act. Fun. Remov. (AFM_s)	Global	Prog.	Remove activation functions

Preparing training data is usually laborious and sometimes error-prone. Our data mutation operators try to imitate the various kinds of problems that are introduced during the data collection process.

- **Data Repetition (DR):** The DR operator duplicates a small portion of training data.
- **Label Error (LE):** It is not uncommon that some data points can be mislabeled. The LE operator mimics such kind of mistakes by changing the label for a data.
- **Data Missing (DM):** The DM operator removes some of the training data
- **Data Shuffle (DF):** The DF operator shuffles the training data into different orders before the training process
- **Noise Perturbation (NP):** The NP operator randomly adds noise to training data.

Similar to traditional programs, a training program is commonly coded using high-level programming languages (e.g., Python) under specific DL framework.

- **Layer Removal (LR):** The LR operator randomly deletes a layer of the DNNs on the condition that input and output structures of the deleted layer are the same.
- **Layer Addition (LAs):** In contrast to the LR operator, the LAs operator adds a layer to the DNNs structure.
- **Activation Function Missing (AFM):** The AFMs operator randomly removes all the activation functions of a layer.

The mutation score of traditional mutation testing is calculated as the ratio of killed mutants to all mutants.

In the mutation testing of DL systems, it is relatively easy for T′ to kill a mutant m′ when the size of T′ is large.
They focus on DL systems for classification problems.

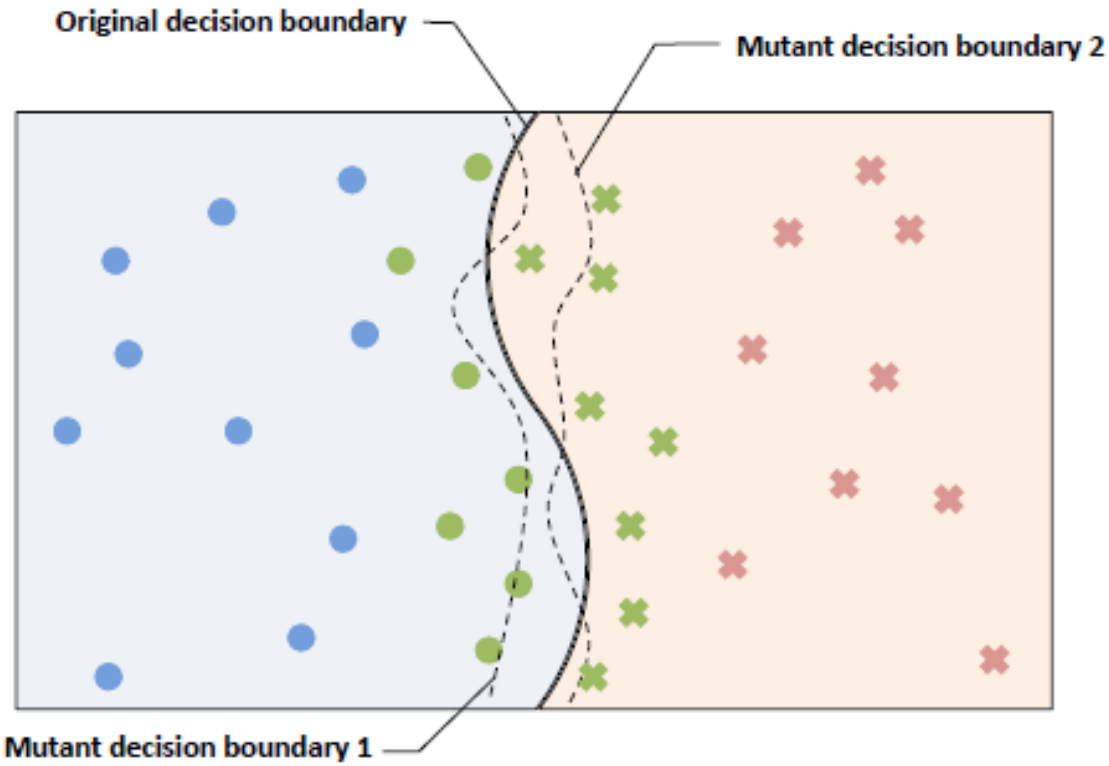
$$\text{MutationScore}(T', M') = \frac{\sum_{m' \in M'} |\text{KilledClasses}(T', m')|}{|M'| \times |C|}$$

To avoid introducing too many behavioural differences for a DL mutant model from its original counterpart, they measure error rate of each mutant m′ on T′. If the error rate of m′ is too high they excluded such mutant models from M′ for further analysis.

They define average error rate (AER) of T′ on the each mutant model $m' \in M'$ to measure the overall behavior differential effects introduced by all mutation operators.

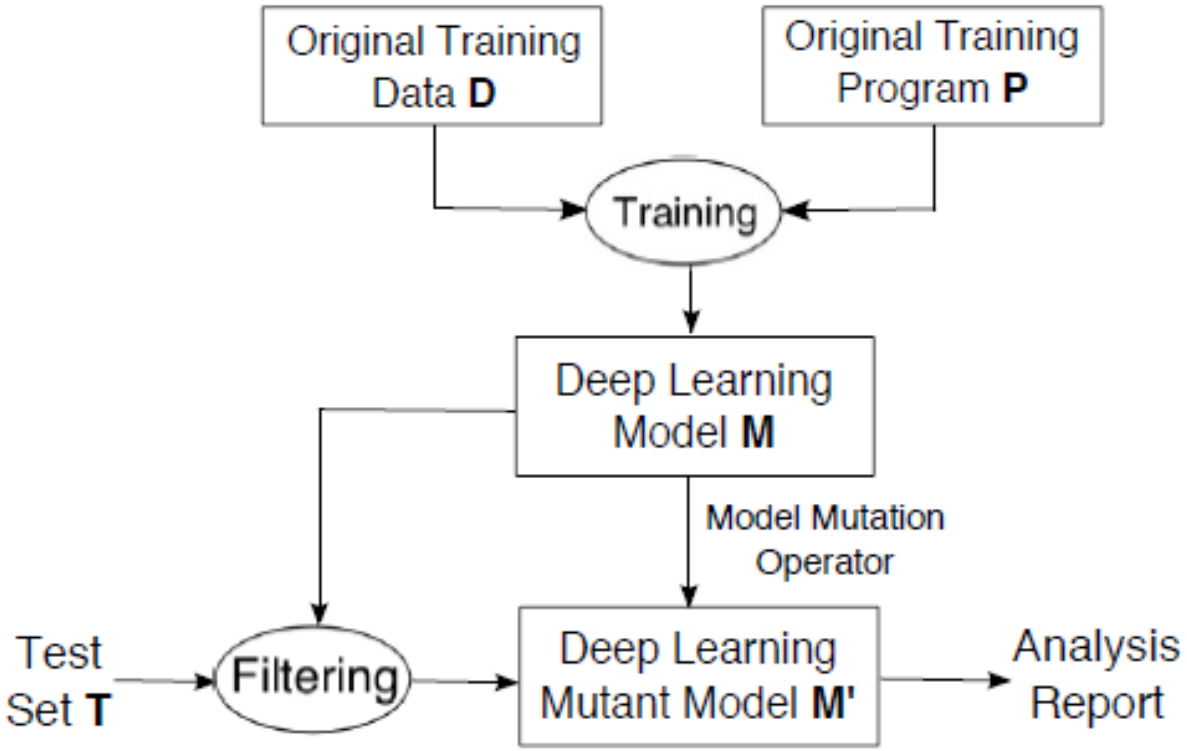
$$\text{AveErrorRate}(T', M') = \frac{\sum_{m' \in M'} \text{ErrorRate}(T', m')}{|M'|}$$

Their mutation testing metrics favor to identify the test data that locate in the sensitive region near the decision boundary.



Model-Level Mutation Testing of Programs

Traditional mutation testing techniques are designed to work on a low-level software representation (e.g., Bytecode, Binary Code) instead of the source code.



Model-Level Mutation Testing of Programs

Model-level Mutation Operators for DL Systems

To simulate possible problems from DL model perspectives, they propose model-level mutation operators, which directly mutate the structure and parameters of DL models.

MODEL-LEVEL MUTATION TESTING OPERATORS FOR DL SYSTEMS.

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

- **Gaussian Fuzzing (GF):** A natural way to mutate the weight is to fuzz its value and change the connection importance it represents. The GF operator follows the Gaussian distribution.
- **Weight Shuffling (WS):** The WS operator randomly selects a neuron and shuffles the weights of its connections with previous layer.
- **Neuron Effect Blocking (NEB):** The NEB operator blocks a neuron effects from all of the connected neurons in next layers, which can be achieved by resetting its connection weights of the next layers to zero.
- **Neuron Activation Inverse (NAI):** The NAI operator tries to invert the activation status of a neuron, which can be achieved by changing the sign of the output value of a neuron before applying its activation function.

- **Neuron Switch (NS):** The NS operator switches two neurons within a layer to exchange their roles and influences for next layers.
- **Layer Deactivation (LD):** The LD operator is a layer level mutation operator that removes a whole layer's transformation effects as if it is deleted from the DNNs.
- **Layer Addition (LA):** The LA operator tries to make the opposite effects of the LD operator, by adding a layer to the DNNs.
- **Activation Function Removal (AFR):** AFR operator removes the effects of activation function of a whole layer.

Evaluation

They have implemented DeepMutation based on Keras (ver.2.1.3) and Tensorflow (ver.1.5.0).
The weight-level and neuron-level mutation operators are implemented through mutating the randomly selected portion of the DNN's weight matrix elements.
They evaluated the implemented mutation testing framework on two practical datasets and three DL model architectures,

Evaluation

Subject Dataset and DL Models

They selected two popular publicly available datasets MNIST and CIFAR-10 as the evaluation subjects. For each dataset, they study popular DL models that are widely used in previous work.

MNIST		CIFAR-10
A (LeNet5)[20]	B[34]	C[35]
Conv(6,5,5)+Relu	Conv(32,3,3)+Relu	Conv(64,3,3)+Relu
MaxPooling (2,2)	Conv(32,3,3)+Relu	Conv(64,3,3)+Relu
Conv(16,5,5)+Relu	MaxPooling(2,2)	MaxPooling(2,2)
MaxPooling(2,2)	Conv(64,3,3)+Relu	Conv(128,3,3)+Relu
Flatten()	Conv(64,3,3)+Relu	Conv(128,3,3)+Relu
FC(120)+Relu	MaxPooling(2,2)	MaxPooling(2,2)
FC(84)+Relu	Flatten()	Flatten()
FC(10)+Softmax	FC(200)+Relu	FC(256)+Relu
	FC(10)+Softmax	FC(256)+Relu
		FC(10)
#Train. Para. 107,786	694,402	1,147,978
Train. Acc. 97.4%	99.3%	97.1%
Test. Acc. 97.0%	98.7%	78.3%

They design experiments to investigate whether their mutation testing technique is helpful to evaluate the quality of the test data.

Evaluation

Controlled Dataset and DL Mutant Model Generation

A good test dataset should be comprehensive and covers diverse functional aspects of DL software use-case.
To demonstrate the usefulness of their mutation testing for the measurement of test data quality, they performed a controlled experiment on two data settings.

Controlled Data Set	MNIST/CIFAR-10			
	Setting 1		Setting 2	
	Group 1	Group 2	Group 1	Group 2
Source	Train. data	Train. data	Test data	Test data
Sampling	Uniform	Non-uniform	Uniform	Non-uniform
#Size	5000	5000	1000	1000

Evaluation

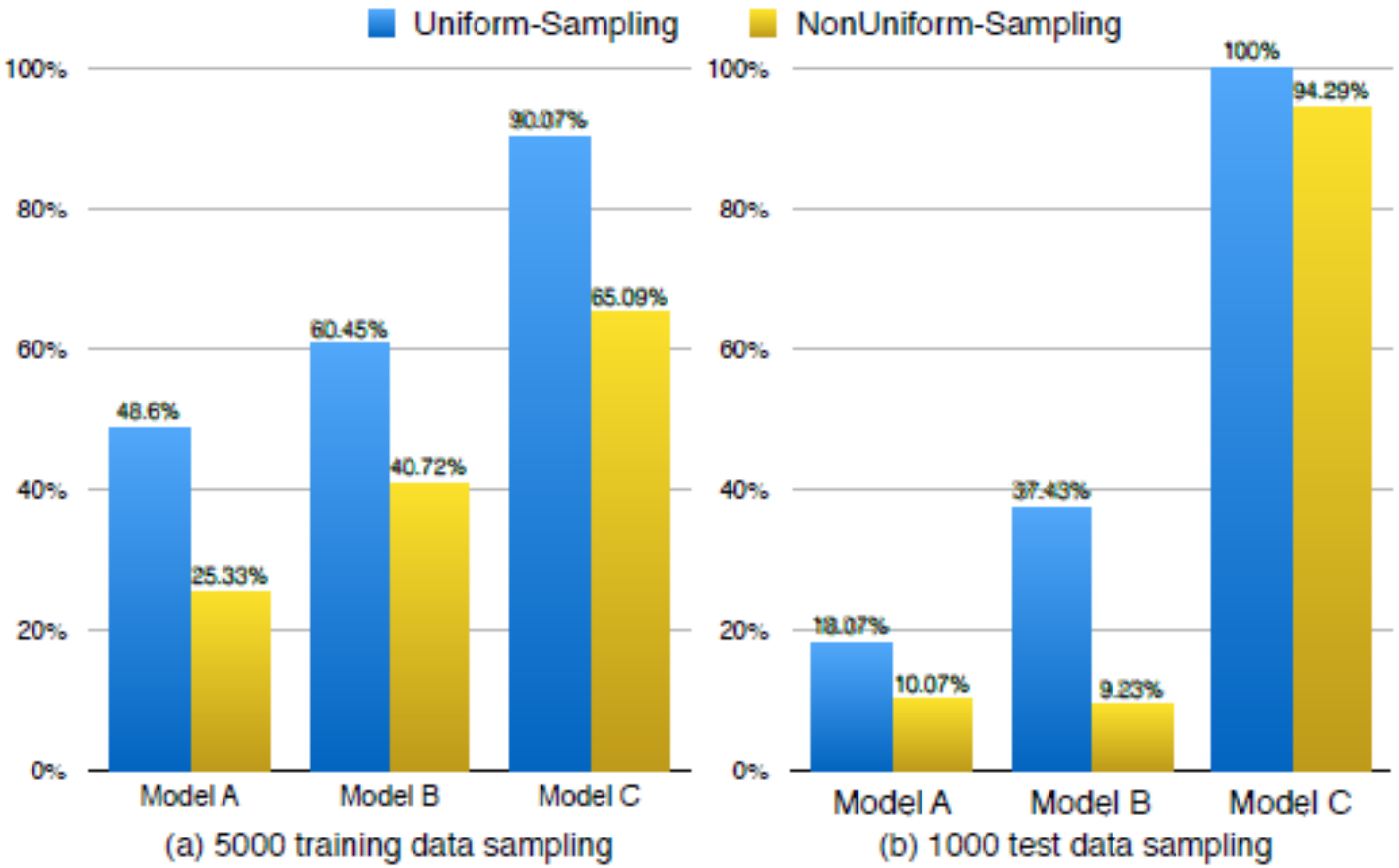
Results

They configure to generate 20 DL mutants for each *data-level* mutation operator.
For program-level mutators, they try to perform mutation whenever the conditions are satisfied with a maximal 20 mutant models for each *program-level* operator.
For each weight and neuron-level mutation operator they generate 50 mutant models.
They generate *layer-level* mutant models when DNN's structure conditions are satisfied with a maximal 50 mutant models for each operator

Model	Source Level (%)				Model Level (%)			
	5000 train.		1000 test.		5000 train.		1000 test.	
Samp.	Uni.	Non.	Uni.	Non.	Uni.	Non.	Uni.	Non.
A	2.43	0.13	0.23	0.17	4.55	4.30	4.38	4.06
B	0.49	0.28	0.66	0.21	1.67	1.56	1.55	1.47
C	3.84	2.99	17.20	13.44	9.11	7.34	11.48	9.00

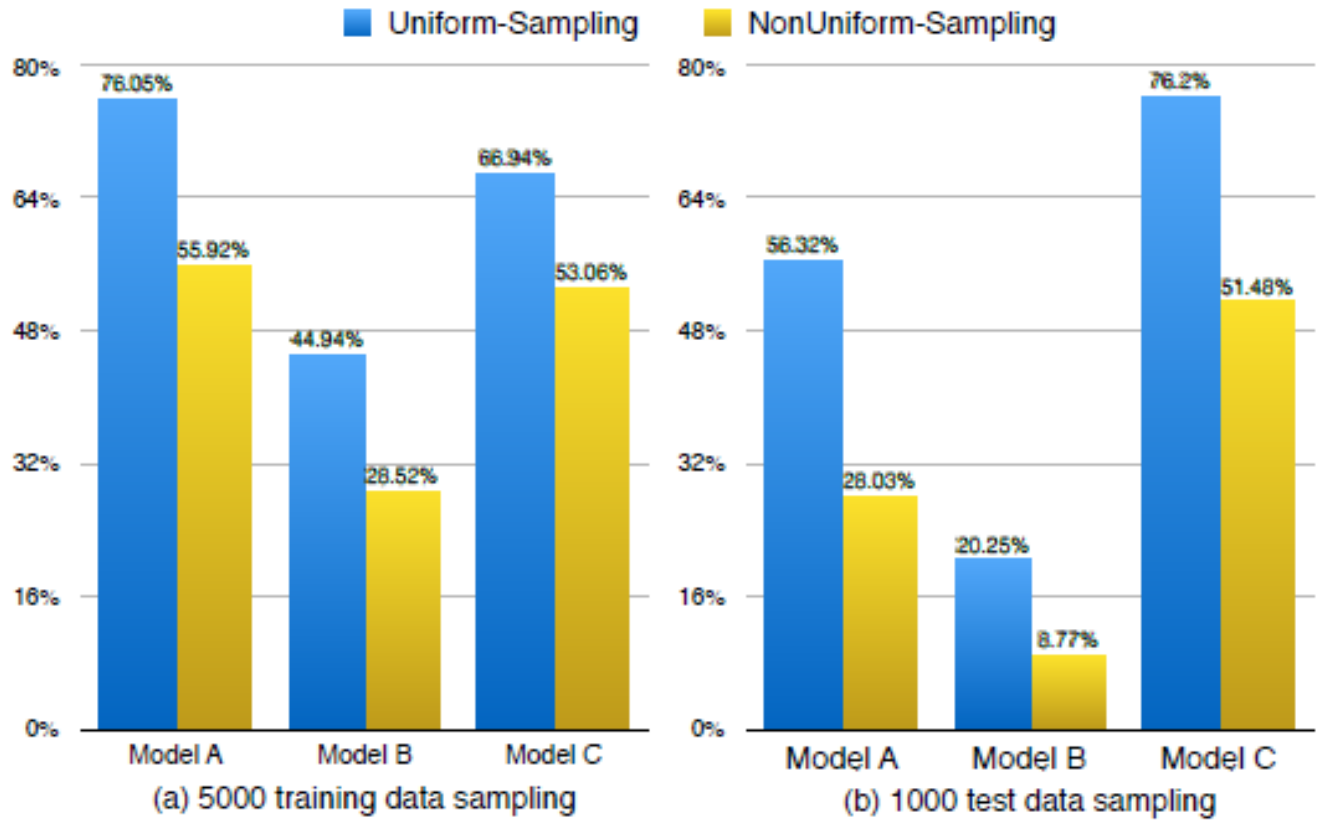
Evaluation

Results



Evaluation

Results



Evaluation

Results

M.	Eval.	Classification Class (%)									
		0	1	2	3	4	5	6	7	8	9
A	mu.	7.22	8.75	9.03	6.25	8.75	8.19	8.75	9.17	9.72	9.03
	avg.err	3.41	3.50	1.81	1.48	4.82	2.52	5.50	4.25	10.45	3.11
B	mu.	1.59	3.29	8.29	7.44	5.49	4.02	8.17	3.66	5.85	8.41
	avg.err.	0.41	1.42	1.12	1.55	1.07	2.92	2.95	1.21	1.24	2.11
C	mu.	8.33	7.95	8.97	9.74	9.74	9.62	9.62	8.97	9.74	7.56
	avg.err.	3.67	6.22	14.80	8.84	9.11	11.53	6.83	11.48	8.87	8.55

In summary, our mutation testing enables the quantitative analysis on test data quality of each class.