2018 IEEE International Conference on Software Quality, Reliability and Security Companion

MuNN: Mutation Analysis of Neural Networks

Weijun Shen, Jun Wan, Zhenyu Chen*
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
* Corresponding author: zychen@nju.edu.cn

***Abstract*—Deep neural networks have made amazing progress in many areas over the past few years. After training deep neural networks, a common way is to build a set of test samples to evaluate the networks. However, test adequacy of deep neural networks is overlooked in both research and practice. This paper proposes MuNN, a mutation analysis method for (deep) neural networks, inspired by the success of mutation analysis in conventional software testing. Five mutation operators are designed on the characteristics of neural networks. The well known dataset MNIST has been used in our experiment. We study two research questions: (1) How does mutation affect neural networks; (2) How does neural depth affect mutation analysis. The experimental results show that mutation analysis of neural networks has strong domain characteristics. This indicates that domain mutation operators are needed to enhance mutation analysis. The experimental results also show that the mutation effects are gradually weakened with the deepening of neurons. We need to design new mutation mechanism for deep neural networks. We believe that mutation analysis can not only be used to measure test adequacy, but also be used to understand neural network operation.**

I. Introduction

Over the past few years, Deep Neural Networks (DNNs), a method of machine learning, have achieved or even surpassed human-level performance for many tasks, including image classification [1], speech recognition [2], malware detection [3], and so on. This has led to widespread adoption and deployment of DNNs in safety-critical systems like self-driving cars [4], and medical Image analyze [5].

The "deep" in DNNs, is relative to "shallow" of earlier machine learning methods, which means a neural network has many hidden layers. DNN-based software is different from the conventional software because its output is not obtained through define-use of variables and rigorous logic control, but through the complex numerical calculation with model parameters and function transformation. Even for developers, the decision in DNNs is an inexplicable black box.

This new type of software has aroused widespread concerns in the community of software testing. Traditional practices in testing DNNs primarily measure their accuracy on test samples (as test inputs) with labels (as test oracles) in a given dataset, so that the results largely depend on test samples in the dataset. Hence, it is important to evaluate test adequacy to draw a conclusion or guide new test generation.

It is important to evaluate test adequacy for DNNs. Traditional deep learning developers evaluate the data adequacy by category-balance [6]. Recently, in DeepXplore [7], the concept of neuron activation coverage was proposed for the first time to evaluate test adequacy. However, evaluation in the perspective of category or neuron coverage is not sufficient. There must be other perspectives driven by bug detection.

Mutation analysis is a popular method to measure test adequacy in software testing. Mutation injects some bugs in software (so-called mutants) and calculates how many bugs are detected by test set as a mutation score. It is in nature that a high mutation score indicates a strong test set. In this paper, we firstly propose a method of mutation analysis on neural networks, namely MuNN. MuNN has five kinds of mutation operators, in which two of them are deleting neuron in input and hidden layers and three of them are changing bias, weight and activation functions, respectively.

The **key contributions** of this paper are:

- To our best knowledge, it is the first time to apply mutation analysis on (deep) neural networks. The preliminary results show the great difference of mutation analysis between conventional software and DNN-based software.
- Two interesting directions, domain-dependent mutation and depth-dependent mutation, are proposed to enhance mutation analysis of DNNs.
- Although MuNN is proposed to measure test adequacy of neural networks, it also shows a potential direction for understanding neural network operation.

II. Related Work

Mutation analysis is a method based on bug detection which provides a test adequacy measurement called mutation score [8]. Mutation score is divided by the total number of mutants according to the number of killed mutants. A mutant is a revised program or model by applied a mutation operator. A mutant is killed by a test if its output is different from the original program (original DNNs model here).

The history of mutation analysis can be traced back to 1971 in a student paper by Lipton [9]. The birth of the field can also be identified in papers published in the late 1970s by DeMillo et al. [10] and Hamlet [11]. Mutation analysis can be used for software testing at all levels, including unit, integration, and specification. There has been much research work on the various kinds of techniques seeking to turn mutation analysis into a practical method [12][13]. Since mutation analysis was proposed in the 1970s, it has been applied to source code [14], specification [15], and so on. Mutation analysis has also been applied to Java, C, python and almost all popular program languages [8].

Traditional mutation analysis [16] targets only a subset of faults which are close to the correct version of the program, with the hope that these will be sufficient to simulate all faults.

978-1-5386-7839-8/18/$31.00 ©2018 IEEE
DOI 10.1109/QRS-C.2018.00032

This is based on two well know hypotheses: the Competent Programmer Hypothesis and the Coupling Effect Hypothesis [10]. In addition to evaluate test adequacy, mutation analysis has also been used to support other testing activities, for example, test generation, test prioritization and test set minimization [17].

Some methods similar to mutation analysis have been already used in the model training and optimizing. Dropout [18] is a technique for addressing the overfitting problem. The key idea is to randomly drop (delete) neuron units (along with their connections) from the neural networks during training. Model compression [19][20] is a technique for reducing the storage and energy required to run inference on large networks so they can be deployed on low-end devices such as mobile devices. The main steps of deep compression are pruning, trained quantization and Huffman coding. Both dropout and model compression are not used to measure test adequacy.

In the field of DNN, a popular method of new test generation is adversarial sample. Adversarial samples are gained by adding subtle perturbations to inputs, which can lead DNN to output incorrect results. There exist many methods to generate adversarial samples, including FGSM, JSMA, DeepFool, One Pixel Attack and so on [21]. On the other hand, in DeepTest [22], metamorphic relations which can alleviate the Oracle problem were applied in DNN to generate new test samples. However, these methods of adversarial sample and metamorphic relation are designed for test generation, not for test adequacy measurement.

Mutation can simulate some real bugs of software and measure test adequacy to identify which test set is better. There are many successful stories of mutation analysis in both research and practice. This inspires us to introduce mutation analysis, namely MuNN, in testing of neural networks. The huge gap between conventional software and DNN-based software bring challenges and opportunities in this area. The experimental results are preliminary but interesting. We believe that mutation analysis is a promising direction for DNNs testing and understanding.

III. Background

A. Deep Neural Networks

DNN-based software is defined to be any software that contains at least one Deep Neural Network (DNN) component. A DNN consists of three kinds of layers, each containing many neurons as shown in Fig. 1. A neuron is an individual computing unit inside a DNN that applies a linear function on its inputs and a nonlinear function (known as activation function) on above output and passes the result to other connected neurons 2. The common activation functions include sigmoid, hyperbolic tangent, or ReLU (Rectified Linear Unit) [23]. The most commonly used activation function currently is ReLU.

Overall, a DNN can be defined mathematically as a multi-input, multi-output parametric function. Each connection between two neurons is specified as a weight parameter representing the strength of the connection between two neurons. After the input is multiplied by the corresponding

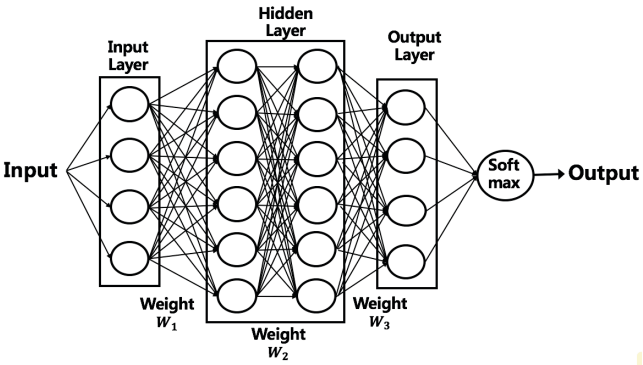


Fig. 1. Fully Connect Feedforward Network

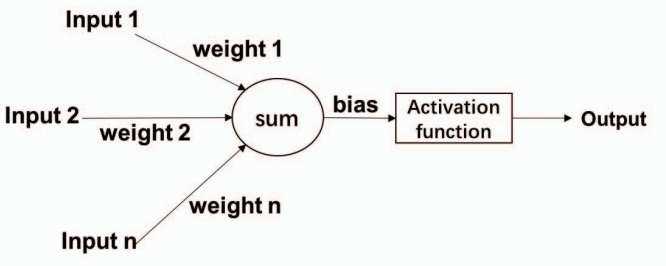


Fig. 2. Single Neuron Model

weight, a scalar called bias will be added to the sum of products. The weights and biases of the connections are learned during training by minimizing a cost function over the training data.

B. Mutation Analysis

The traditional process of mutation analysis is illustrated in Fig. 3. In mutation analysis, from a program p, a set of faulty programs p', called mutants, are generated by a few single syntactic changes to the original program p. As an illustration, changing the and operator && of the original program p(if($x > 0 \&\& y < x + 5$) return 0;), into the operator ||, thereby producing the mutant p'(if($x > 0 || y < x + 5$) return 0;).

After a comprehensive analysis of above introduction, we know that the deep neural network functions mainly on parameters: weights and biases, not the traditional code logic (eg: if, while, for). So it is ineffective if we only mutate the code of the deep neural network. We put forward a new technique to mutate the neural network, which can effectively take the characteristics of the depth learning software into account. This technique can partly simulate the defects of the model to a certain extent and evaluate the ability of a test suite to find defects. Also it can provide testers of such software a completely new thought and method for measuring the testing adequacy.

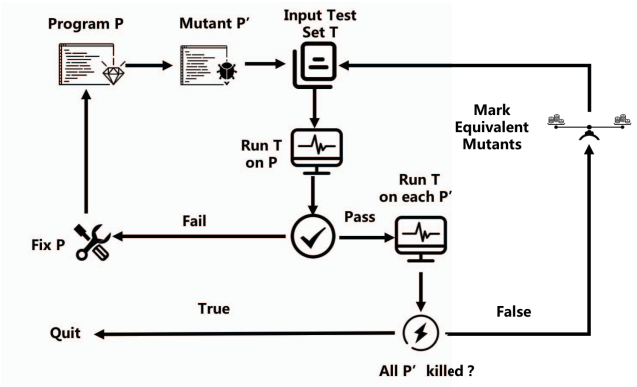


Fig. 3. Generic process of mutation analysis [8]

IV. Methodology

In this paper our mutation analysis does not focus on the syntax rules, but on the characteristics of neural network. The intuition is that mutation can affect the function of neural network and lead to qualitative change in output. Take a neural network of classification function as example. The neural network obtains the final results based on the operation combining with parameters and inputs. The initial output is a continuous vector and a softmax function is applied before the final discrete output. Thus the continuous value is discretized and the final classification is obtained. So as long as the model is mutated even slightly, there will be a slight change in the continuous output, but it is not necessarily a qualitative change in final discrete outputs. Only the changes in final discrete outputs can be called qualitative change.

In a trained neural network, these characteristics will affect the final results: the input dimension, the connection weight, the bias, the activation function, the structure of the neural network and so on. The characteristics of neural networks are similar to the syntax rules in the code. After mutating these features, it will affect the output of the neural network model. A good test set must be sensitive to the mutation and can kill the neural network mutants as more as possible. A low mutation score will drive testers to generate more test samples to achieve sufficient testing.

We assume that the input is x, an n-dimensional vector. The neural network can be defined as a parameter function $f_O = f(W, B, Af, Ns)$, in which W is weight matrix, B is bias matrix, Af is activation function, Ns is network structure. The output is $y = f_O(x)$. Take "weight" as example, the operator that mutate weight is defined by op_w. The mutated neural network is denoted by $f_m = op_w(f_O)$. A test sample t is called to kill the mutant f_m, if $f_O(t) \neq f_m(t)$. The mutation operators of neural networks regarded as feasible are listed in Table I.

Mutation operators are commonly classified three operation types: insert, delete, and change. "insert" is more complex than other ones in neural networks. We study "delete" and "change" and propose five mutation operators as follows.

- Delete Input Neuron (DIN) means that one neuron in input layer is deleted, so that its corresponding dimension of input vector receives only 0.
- Delete Hidden Neuron (DHN) means that one or more neurons in hidden layer are deleted, so that it has no effect on the subsequent neurons.
- Change Activation Function (CAF) means that one or more activation functions are changed, so that its affection on the subsequent neurons is changed.
- Change Bias Value (CBV) means that one or more bias values are changed, so that its affection on the subsequent neurons is changed.
- Change Weight Value (CWV) means that one or more weight values are changed, so that its affection on the subsequent neurons is changed.

TABLE I
Details of Mutation Operators on DNN software

Mutation Operator	Mutation Function	Description
DIN	$op_{din}(n)$	Delete Input Neuron
DHN	$op_{dhn}(layer, index)$	Delete Hidden Neuron
CAF	$op_{af}(type)$	Change Activation Function
CBV	$op_b(extend, ratio)$	Change Bias Value
CWV	$op_w(extend, ratio)$	Change Weight Value

V. Experiment

We choose the famous data set MNIST and it has 60000 train samples and 10000 test samples. On MNIST two distinct neural networks are trained, one only constituted by the simplest fully connected neural network (denoted as $model_A$) and one constituted unitedly by convolutional neural network (CNN) and fully connected neural network (denoted as $model_B$). There are two hidden layers in $model_A$, 128 neurons in the first layer and 64 neurons in the second. There are three convolution Layers in $model_B$ which contains 32, 64 and 128 filters in each layer with a convolution kernel size (3, 3). The accuracies of trained $model_A$ and $model_B$ are 98.03% and 98.77% respectively which are all higher than the baseline 98%. We run our python programs in Keras and its backend is tensorflow.

Two research questions are highlighted as follows.

- **RQ1**: How does mutation affect neural networks?
- **RQ2**: How does neuron depth affect mutation?

A. RQ1

In order to answer RQ1, we study experimentally the influence of each neuron mutation operator on the model in details and then discuss the results one by one.

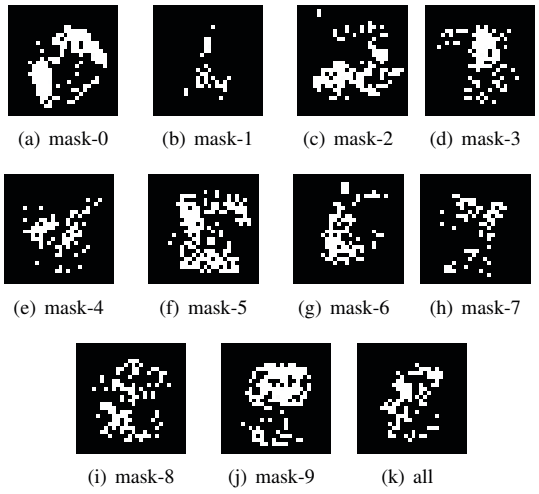
The results of deleting input neurons of $model_A$ are showed in the subfigure: all in Fig. 4. Because the input picture has 28*28 (784) dimensions, we deleting 784 input neurons by turns, and then print the results in the form of gray scale picture. Black indicates that deleting the input neuron has almost

no influence (the accuracy is greater than 98%) while white indicating lower than 98%. But shielding only one dimensional input has little influence on the model, and the accuracies of all 784 model mutants are always above 97%. This result is also in line with our intuition for the reason that strokes of numbers are usually in the middle of the picture and the pixel on the edge has little effect on the recognition results.

Further results by classification labels are shown in Fig. 4. Taking 0 as an example, the black part of this 28*28 dimensional picture indicates when deleting the input neuron on this dimension, this DIN operator does not affect identifying the number of 0 while the white part indicates the performance decreasing in identifying 0. We can see that different types of input (e.g. 0 or 4) are fluctuation on the DIN mutation, and different types of input actually represent different functions in the actual application.

We also evaluate the intersections of every two numbers in Fig. 4 with black as 0 and white as 1. The result in Table II shows 0 and 9 have the maximum common input neurons influncing the original model while 1 and 4 the minimum. The former phenomenon is in line with our intuition because 0 and 9 have the maximum common parts. Because the vertical strokes of 4 in the test samples are more tilted than that of 1, there are minimum overlaps in their strokes, which explains the experimental results.

From the experimental results above, we can see that mutation analysis of neural networks can have strong domain characteristics, which can even help researchers better understanding the works of neural network. In practical, we can mutate neural network based on its domain characteristics of application. For example, we can mutate the road sign recognition function of the automatic driving system by deleting input neurons critical for this function, so as to evaluate the completeness of the test samples.



(a) mask-0 (b) mask-1 (c) mask-2 (d) mask-3

(e) mask-4 (f) mask-5 (g) mask-6 (h) mask-7

(i) mask-8 (j) mask-9 (k) all

Fig. 4. Results of deleting input neurons (DIN) by labels

[1]There are 28*28 input neurons totally.

TABLE II
COMMON INPUT NEURONS [1] BY LABELS INFLUNCING THE ORIGINAL MODEL (DIN)

Common	0	1	2	3	4	5	6	7	8	9
0	140	11	50	37	26	50	34	28	37	63
1	–	33	14	16	5	12	12	6	10	11
2	–	–	125	16	23	38	38	11	29	36
3	–	–	–	99	15	28	16	29	28	54
4	–	–	–	–	77	33	20	16	17	28
5	–	–	–	–	–	156	37	25	42	60
6	–	–	–	–	–	–	92	12	38	30
7	–	–	–	–	–	–	–	65	15	35
8	–	–	–	–	–	–	–	–	104	44
9	–	–	–	–	–	–	–	–	–	142

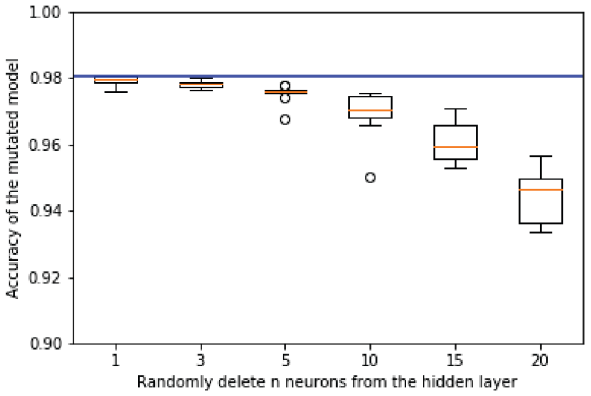
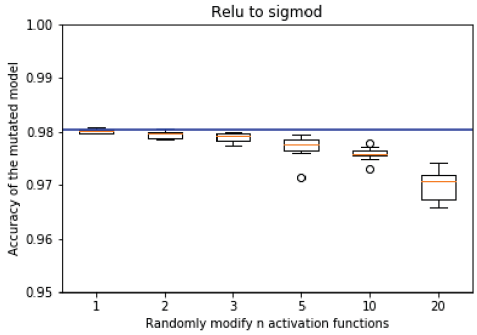


Fig. 5. Results of randomly deleting hidden neurons (DHN)

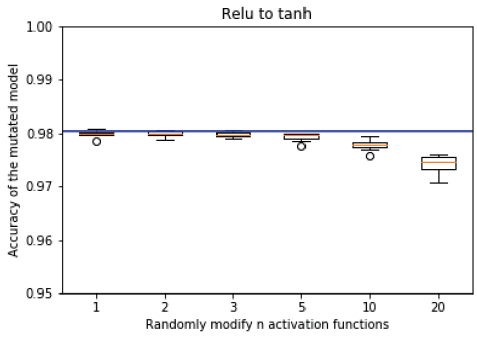
Fig. 5 shows the influence of deleting neurons on the original model. There are total 192 neurons in the hidden layer of $model_A$. We randomly delete n neurons to study the influence of DHN on the neural network. For each n, we ran our program 10 times to get ten sample accuracies of the mutants and draw a Box-plot. The mean accuracies are 0.97894, 0.97809, 0.97526, 0.96927, 0.96062 and 0.9371 respectively. The final result shows that when only 5 or less neurons are deleted, there is very little influence on the model. When 20 neurons are deleted, the performance of the model will be significantly reduced.

Fig. 6 shows the effect of changing activation functions of neurons on $model_A$. We randomly selected n neurons to change their activation functions, studying the corresponding influence on the origin model. For each n, we ran our program ten times to get ten sample accuracies of the mutants and draw a Box-plot. Compared to tanh, altering relu to sigmod can significantly affect the performance of the model. We think the reason is that differences between the tanh function and the relu function are less than sigmod function.

In the practical stage, we found that modifying only one parameter (weight or bias) value has little influence on the origin model. If all the parameter values are modified, it is obvious that the model will be greatly affected. So, we have done an empirical experiment to study the effect on origin model by different $extent$ and $ratio$. The range of $extent$ we



(a) relu to sigmod



(b) relu to tanh

Fig. 6. Results of randomly changing activation functions (CAF)

selecte is 0.01 to 10 while which of $ratio$ was 1% to 20%. But because we randomly select the parameter values to modify, so the result should not be fixed but be a statistical value. We run the program 10 times and obtain 10 sample accuracies of the mutated models whose mean value are listed in Table III IV V.

TABLE III
MEAN ACCURACY OF MUTATING BIASES OF $model_A$ (CBV)

Ratio \ Extent	0.01	0.1	0.5	1.5
1%	0.98029	0.98027	0.98031	0.98033
3%	0.98023	0.98020	0.98025	0.98034
5%	0.98016	0.98024	0.98030	0.98045
10%	0.98017	0.98023	0.98024	0.98041
20%	0.98012	0.98019	0.98031	0.98046
Ratio \ Extent	2	3	5	10
1%	0.98031	0.98028	0.98027	0.97994
3%	0.98033	0.98040	0.98030	0.97950
5%	0.98040	0.98029	0.97996	0.97856
10%	0.98051	0.98029	0.97949	0.97610
20%	0.98058	0.97983	0.97861	0.96922

With regard to bias, we show the experimental results in Table III. It can be seen obviously that the effect of changing bias values on the model is not significant. Only when the $extent$ reaches the magnitude 10, will all the mean accuracies be below 98%. In practice, testers can use CBV to generate

TABLE IV
MEAN ACCURACY OF MUTATING WEIGHTS OF $model_A$ (CWV)

Ratio \ Extent	0.01	0.1	0.5	1.5
1%	0.97907	0.97934	0.98010	0.98004
3%	0.97522	0.97665	0.97922	0.97914
5%	0.97299	0.97502	0.97847	0.97860
10%	0.95792	0.96561	0.97705	0.97703
20%	0.89806	0.94043	0.97189	0.97585
Ratio \ Extent	2	3	5	10
1%	0.97997	0.97562	0.93586	0.78210
3%	0.97668	0.96060	0.85078	0.40978
5%	0.97541	0.94749	0.82478	0.33574
10%	0.97242	0.92105	0.69123	0.30439
20%	0.96120	0.90294	0.66980	0.37409

tricky mutants which are hard to kill. Another interesting phenomenon is that at the $extent$ of 1.5 and 2, all the accuracy of mutants is higher than the origin model. In our knowledge, in theory, the random mutation of parameters may have a certain possibility of optimizing the original model, achieving the desired goal of training. Because the essence of training is also a process of constantly modifying parameters. In the future, we can consider mutation as a model optimization method to further study.

The results in Table IV show that when the extent is in the range [0.5, 1.5] even we modify 20% of all weights, the mean accuracies are higher than 97%. Whether weight (CWV) or bias (CBV), it follows a rule: the greater the magnitude of the modification, the greater the proportion of the parameters selected, the greater the model will be disturbed (see from the accuracies).

TABLE V
MEAN ACCURACY OF MUTATING SHARED WEIGHTS OF $model_B$ (CWV)

Ratio \ Extent	0.01	0.1	0.5	1.5
1%	0.98770	0.95922	0.98712	0.98728
3%	0.86529	0.95701	0.98692	0.98216
5%	0.83334	0.92182	0.98569	0.98634
10%	0.78773	0.83037	0.98708	0.98502
20%	0.59631	0.77958	0.94868	0.98360
Ratio \ Extent	2	3	5	10
1%	0.98770	0.98770	0.98770	0.98770
3%	0.97341	0.98124	0.97898	0.86648
5%	0.98599	0.87552	0.97665	0.74833
10%	0.95823	0.86828	0.90358	0.90178
20%	0.93456	0.84594	0.75011	0.55541

In addition to the common weight in the fully connected neural network, there is a special weight value in CNN, called shared weight. Essentially its convolution kernel parameters are connection weights, but they are shared. From Table V, we can see some interesting facts. First, when the $extent$ is 1.5, even changing 20% of the parameters, the mean accuracies are not lower than 98% (origin: 98.77%). Secondly, when only 1% of the shared weights are modified, all the $extent$ does not disturb the model except for 0.1. In practice, tester can chose

ratio as below 5% and *extent* as in range [0.5,2] to generate tricky mutants which are hard to kill.

B. RQ2

In order to answer RQ2, we have additionally trained 4 fully connected neural network models with 1, 3, 5 and 10 hidden layers respectively, and 64 neurons in each hidden layer. The accuracies of trained models are 96.92%, 97.24%, 97.12% and 97.21% respectively.

Fig. 7 shows the result of modifying weights of neurons on 4 different models. We run every mutation program 10 times and get 10 sample accuracies. Then we show the decrease amplitude of accuracy from the original accuracies in the figure. In general, if the modification extend and modification ratio are fixed, the deeper the neural network, the faster the accuracy will decrease. In other words, if we want to achieve the same mutation effect, the deeper the network is, the less ratio of weight values we need to modify. Although the proportion is decreasing, the fact that the number of weight values is multiplied with the depth growth must be taken into account. Therefore, as the number of layers increases, the number of weights actually modified is increased. We can conclude that the deeper the neural network, the greater number of weights needed to be modified to achieve the same effect. But the proportion of the weights needed is reduced. Therefore, when we mutate the deep model, we can not simply apply the conclusion of the shallow neural network.

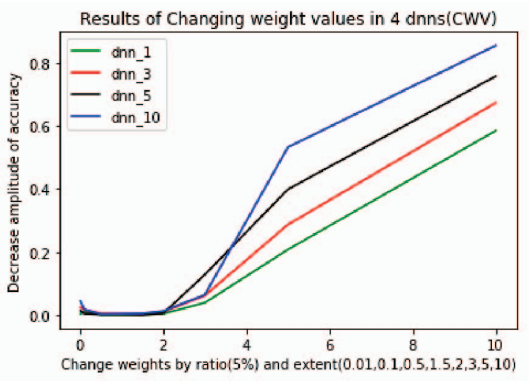


Fig. 7. Results of Changing weight values in 4 dnns(CWV)

Fig. 8 shows the result (similar to weight) of changing biases of neurons on 4 different models. It can be seen that in general, CBV has a obvious weaker influence on the model than CWV (see the decrease amplitude of accuracy), but still some interesting phenomena. For example, 10-layer neural network will be easier to disturb. Therefore, the conclusion that the bias mutation in the shallow network has little effect on the final result, can not be generalized in the deeper network. This once again reminds us that we should pay attention to the influence of the depth on mutation.

From the result in Fig. 9, we can't come to any obvious conclusion. Deeper neural network does not show obvious robustness on this kind of mutation and the single layer

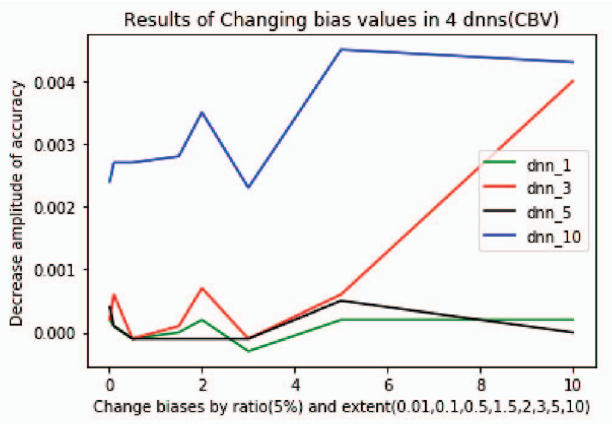


Fig. 8. Results of modifying biases in 4 dnns (CBV)

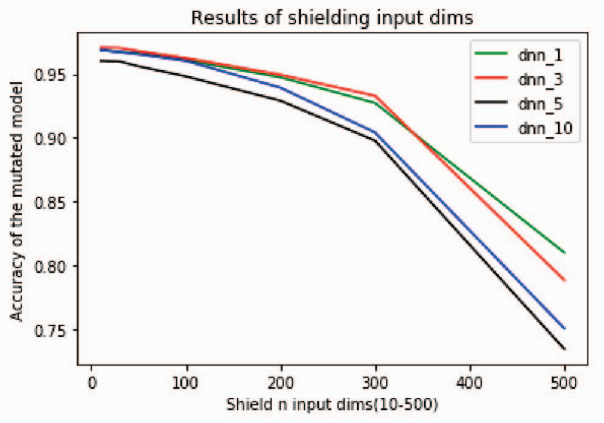


Fig. 9. Results of deleting input neurons in 4 dnns (DIN)

neural network is surprisingly more robust. We think under the mutation operator HIN, the first layer of neurons can be significantly affected in the recognition of input characteristics. So the deeper neurons do not bring better robustness under this condition. The more layers the neural network has, more vulnerable it is to this kind of mutation, which needs more attention and research later.

The influence of deleting hidden neurons on the 4 models is shown in Fig. 10. The percentage of deleted neurons was 0.0625, 0.125, 0.25, 0.3125 and 0.375 respectively (which can make sure the number is integer). It can be found that when the same proportion of neurons is deleted and the deeper the neural network is, the performance declines greater. This shows that to achieve the same mutation effect, the deeper neural network does not need to delete the same proportion of neurons as in the shallow network.

Unlike the infinite possibility of modifying parameters (weight and bias), deleting a neuron is a deterministic operation. We have studied the effect of deleting neurons by an experiment, mainly on how many test samples can kill generated mutants. In the experiment, we take turns to delete each neuron in the

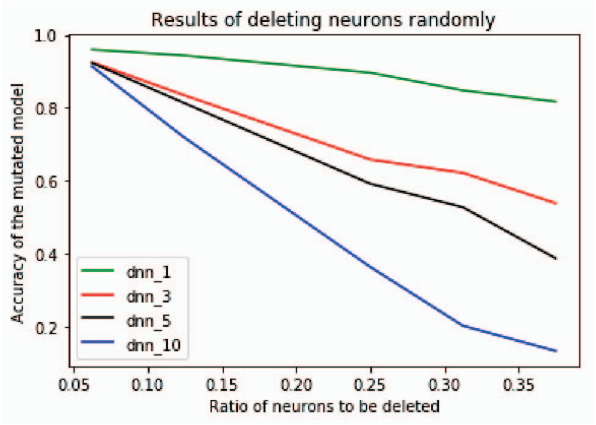


Fig. 10. Results of deleting hidden neurons in 4 dnns (DHN)

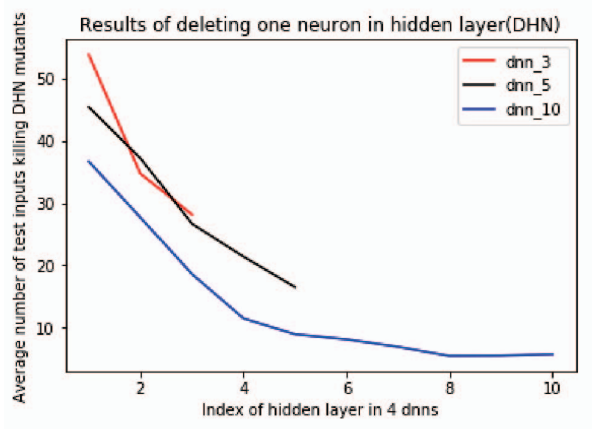
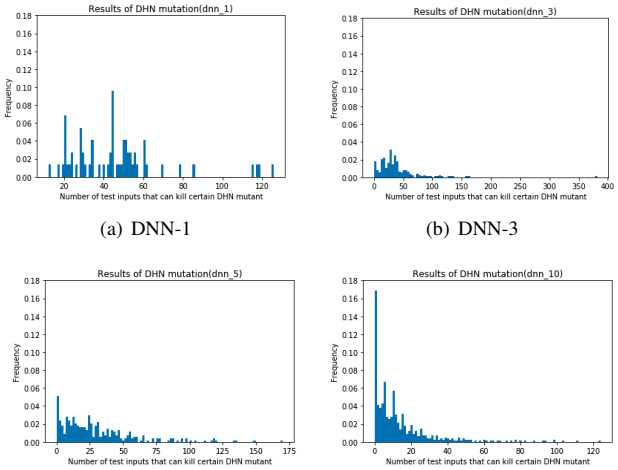


Fig. 11. Results of deleting one neuron in hidden layer (DHN)

input layer and hidden layer, that is, deleting each neuron produces a mutant. Then the numbers of test samples that can kill every mutant are evaluated, and note that the test samples here are not including those which are not passed on the original model.

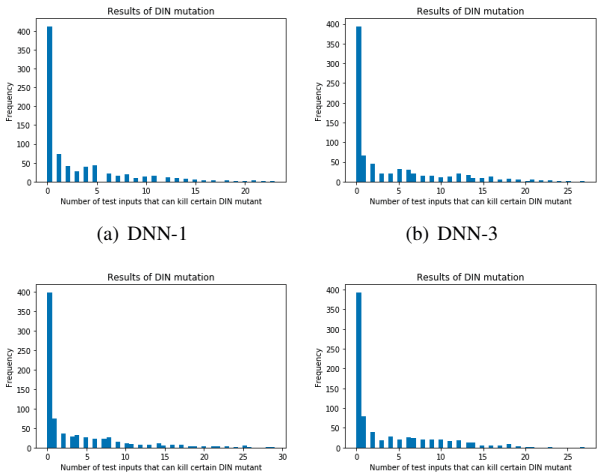
As we can see in Fig. 11, with the increase of the hidden layers, the mutants gained by deleting the neuron in deeper layer are more difficult to kill, and the corresponding mutants are more valuable. Several conclusions can be drawn from Fig. 12. When the hidden layer has only one layer, each mutant corresponding to each hidden neuron can be killed (the frequency of 0 is 0). But when the depth of the hidden layer reaches 3, a lot of mutants will not be killed (the frequency of 0 is not 0), that is, this deletion operator can not be detected by test samples. At the same time, we can see a trend from Fig. 12, deeper as the neural network is , less number of test samples can kill the mutant, which shows lower probability of detecting the DHN operators in the deep neural network. Fig. 13 shows the result of deleting the neuron in the input layer, and it can be seen that most of the mutants can not be killed (the frequency of 0 all reaches about 400). Deleting neurons in



(a) DNN-1 (b) DNN-3

(c) DNN-5 (d) DNN-10

Fig. 12. Frequency histogram about numbers of test samples killing DHN mutants



(a) DNN-1 (b) DNN-3

(c) DNN-5 (d) DNN-10

Fig. 13. Frequency histogram about numbers of test samples killing DIN mutants

the input layer is not clearly associated with the depth of neural network, which is also in line with our intuition, because the neural network is a black box for the input.

To sum up, for RQ2, we have done a lot of experiments. And the depth of neural network will have diverse influence on the mutation. Neural networks of different depth require different ways of mutation. Some mutation operators on shallow networks can not be directly used in deeper neural networks in the same way.

VI. Conclusion

In this paper, we firstly propose a novel method, namely MuNN, for mutation analysis of neural networks. Five mutation operators are designed to construct potential buggy deep learning models. Given a test set, we can calculate the mutation

score as test adequacy. The mutation results can also be used to guide test generation, test optimization and other aspects of neural network testing. In particular, the mutation results also show some potential applications in neural network understanding.

The experimental results show some new features of mutation analysis different from conventional software. Even in the application area of handwritten digit recognition, there are great difference between two different numbers. This inspires us to design some domain-dependent mutation operators rather than common mutation operators. The results also indicate that the neuron depth is a sensitive factor in mutation analysis. It is reasonable that the robustness of neural networks usually comes from the network depth. This also inspires us to design depth-dependent mutation operators for deep neural networks. In summary, mutation analysis of neural networks is a challenging and promising area in both machine learning and software engineering.

ACKNOWLEDGMENT

The work is supported in part by the National Key Research and Development Program of China (2016YFC0800805) and the National Natural Science Foundation of China (61690201).

REFERENCES

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770-778.

[2] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. 2016. Achieving Human Parity in Conversational Speech Recognition. *arXiv preprint arXiv: 1610.05256*.

[3] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. 2014. Droid-sec: Deep Learning in Android Malware Detection. *ACM conference on SIGCOMM*, pp. 371-372.

[4] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to End Learning for Self-driving Cars. *arXiv preprint arXiv: 1604.07316*.

[5] Greenspan H, Ginneken B V, Summers R M. Guest Editorial Deep Learning in Medical Imaging: Overview and Future Promise of An Exciting New Technique. *IEEE Transactions on Medical Imaging*, 2016, 35(5): 1153-1159.

[6] LÍőpez V, FernÍćndez A, GarcÍła S, et al. An Insight into Classification with Imbalanced Data: Empirical Results and Current Trends on Using Data Intrinsic Characteristics. *Information Sciences*, 2013, 250(11):113-141.

[7] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. *Symposium on Operating Systems Principles*, pp. 1-18.

[8] Jia Y, Harman M. An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions on Software Engineering*, 2011, 37(5): 649-678.

[9] R. Lipton, Fault Diagnosis of Computer Programs, *Student Report, Carnegie Mellon University*, 1971.

[10] R.A. DeMillo, R.J. Lipton, and F.G. Sayward, Hints on Test Data Selection: Help for the Practicing Programmer, *Computer*, vol. 11, no. 4, pp. 34-41, Apr. 1978.

[11] R.G. Hamlet, Testing Programs with the Aid of a Compiler, *IEEE Transactions on Software Engineering*, 1977, 4(3): 279-290.

[12] Changbin Ji, Zhenyu Chen, Baowen Xu, Ziyuan Wang, A New Mutation Analysis Method for Testing Java Exception Handling, *Computer Software and Applications Conference*, 2009: 556-561.

[13] Zhenyu Chen, Axel Hollmann, Positive and Negative Testing with Mutation-Driven Model Checking, *GI Jahrestagung*. 2012, 133: 187-192.

[14] R.A. DeMillo, Program Mutation: An Approach to Software Testing, *technical report, Georgia Inst. of Technology*, 1983.

[15] A.S. Gopal and T.A. Budd, Program Testing by Specification Mutation, *Technical Report 83-17, University of Arizona*, 1983.

[16] Changbin Ji, Zhenyu Chen, Baowen Xu, Zhihong Zhao, A Novel Method of Mutation Clustering Based on Domain Analysis, *Software Engineering and Knowledge Engineering*. 2009: 422-425.

[17] Zhiyi Zhang, Dongjiang You, Zhenyu Chen, Yuming Zhou, Baowen Xu, Mutation Selection: Some Could be Better than All, *International Workshop on Evidential Assessment of Software Technologies, in Conjunction with Enase 2011, Beijing, China, June. DBLP,* 2015: 10-17.

[18] Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 2014, 15(1): 1929-1958.

[19] Y. Guo, A. Yao, and Y. Chen, Dynamic Network Surgery for Efficient DNNs. *International Conference on Neural Information Processing Systems*, 2016.

[20] Han S, Mao H, Dally W J. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *Fiber*, 2015, 56(4): 3-7.

[21] Akhtar N, Mian A. Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey. *IEEE Access*, 2018.

[22] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. DeepTest: Automated Testing of Deep-neural-network-driven Autonomous Cars. *International Conference on Software Engineering*, 2018.

[23] Shang W, Almeida D, Almeida D, et al. Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units, *International Conference on International Conference on Machine Learning*. JMLR.org, 2016: 2217-2225.