Generating Biased Dataset for Metamorphic Testing of Machine Learning Programs

Shin Nakajima, Tsong Yueh Chen

Generating Biased Dataset for Metamorphic Testing of Machine Learning Programs

Shin NAKAJIMA[1] and Tsong Yueh CHEN[2]

[1] National Institute of Informatics, Tokyo, Japan
[2] Swinburne University of Technology, Melbourne, Australia

Abstract. Although both positive and negative testing are important for assuring quality of programs, generating a variety of test inputs for such testing purposes is difficult for machine learning software. This paper studies why it is difficult, and then proposes a new method of generating datasets that are test inputs to machine learning programs. The proposed idea is demonstrated with a case study of classifying handwritten numbers.

1 Introduction

Machine learning such as deep neural networks (DNN) [4] is applied to advanced services of extreme reliability such as auto-pilot cars. However, quality assurance of DNN-based programs is more difficult than conventional software systems, because the programs are categorized as *untestable* [12]. Thus, Metamorphic Testing (MT) framework [1] has been used as a standard practice for testing of machine learning programs [2].

Because machine learning programs work on collections of data (datasets) to derive useful information, generating a wide variety of datasets systematically is key to effective testing. Most existing works [7][9][15][16][17] make use of application characteristics and serve for specific test targets.

This paper proposes a systematic follow-up dataset generation method. The method is formulated as an optimization problem referring only to trained learning models without accessing to any application specific information. We conjecture that the method is applicable to a wide variety of neural network learning models [4][5]. The paper also discusses how the proposed method is related to *distortion degrees* in trained learning models, which may open a new research direction of the machine learning software quality.

2 Machine Learning Software

We explain a basic idea of machine learning problems with the case of neural network (NN) supervised learning method [5]. Assume that a learning model $\boldsymbol{y}(W;\boldsymbol{x})$ is given as a multi-dimensional non-linear function, differentiable with respect to both learning parameters W and input data $\boldsymbol{x}$. Learning aims at

obtaining a set of parameter values W^* by solving a numerical optimization problem. Below, a training dataset LS is expressed as $\{\langle \boldsymbol{x}^n, \boldsymbol{t}^n \rangle\}$.

$$W^* \;=\; \underset{W}{argmin}\; \mathcal{E}(W; LS), \qquad \mathcal{E}(W; LS) \;=\; \frac{1}{N}\sum_{n=1}^{N} \ell(\boldsymbol{y}(W;\; \boldsymbol{x}^n),\; \boldsymbol{t}^n)$$

The function $\ell(\_, \_)$ denotes distances to represent how much a calculated output $\boldsymbol{y}(W; \boldsymbol{x}^n)$ differs from an expected supervisor tag value $\boldsymbol{t}^n$. The machine learning problem is to search for learning parameter values to minimize the mean of ℓ under ρ_{em} of the empirical distribution of the training dataset LS.

Let $\mathcal{F}_L(X)$ be a function to calculate W^*, that is a program to solve the above optimization problem with its input dataset X. $\mathcal{F}_L(X)$ is a faithful implementation of a machine learning method [5]. Another function, $\mathcal{F}_I(\boldsymbol{x})$, calculates prediction or inference results of incoming data $\boldsymbol{x}$. Its functional behavior depends on W^* or $\boldsymbol{y}(W^*; \boldsymbol{x})$. Quality assurance of machine learning software involves two different programs [8]. Although this paper focuses on testing of $\mathcal{F}_L$, data generated with the proposed method can be input to $\mathcal{F}_I$ as well.

3 Dataset Diversity in Testing

Software testing is a systematic method to provide quality assurance of programs, and is sometimes viewed from two different perspectives. *Positive testing* focuses on ensuring that a program does what it is supposed to do for normal input satisfying the preconditions. *Negative testing* ensures that the program does not do what it is not supposed to do even when the input is unexpected or violates the preconditions.

We raise a question here how we consider preconditions for machine learning programs. $\mathcal{F}_L$ accepts a training dataset LS and calculates learning parameter values W^*. Alternatively, $\mathcal{F}_L$ is regarded to synthesize W^* from a piece of specification information that LS implicitly has. Although it is related to *specifications*, LS is just a collection of concrete data, and can not be a checkable condition.

A dataset X is a sample by statistics, and its elements are theoretically viewed to be generated *independently and identically distributed* according to a probabilistic distribution ρ; $X = \{d \mid d \sim_{i.i.d.} \rho\}$ (written as $X \sim_{i.i.d.} \rho$ in this paper). If X does not satisfy the relation $X \sim_{i.i.d.} \rho$, X is not a faithful representative of ρ. The situation is a sample selection bias [11] and consequently considered as a specification error [6].

In machine learning, we know only an empirical distribution ρ_{em} of LS, which is believed to be an approximation of a hypothetical distribution ρ. We assume that there are M ($M > 0$) LS_js, each of which is of a different empirical distribution ρ_j, but collectively satisfies a relationship, $(\cup_{j=1}^{M} LS_j) \sim_{i.i.d.} \rho$. Such a ρ defines a group of input, and hense can be interpreted as a precondition. Some of LS_js are free from sample selection bias ($LS_j \sim_{i.i.d.} \rho$); they form a group of datasets LSS_P. The others LS_k, each of which is still a sample of ρ, are biased

and written here as $LS_k \sim_{\neg i.i.d.} \rho$; they form LSS_N. We consider all datasets in LSS_P satisfying the precondition, and all datasets in LSS_N not satisfying it. Then, $LSS_P \cap LSS_N = \phi$. Furthermore, all of them serve the same machine learning task, because they are related to the ρ. The datasets in $LSS_P \cup LSS_N$ show different nature, and thus embed the *dataset diversity* [9].

Note that a randomly generated dataset does not have any relationship with ρ and thus is not appropriate even for negative testing.

4 Generating Distorted Dataset

We generate biased or distorted datasets algorithmically from LS of ρ_{em}. The method is based on the L-BFGS [13], explained below.

Given a dataset LS of $\{\langle \boldsymbol{x}^n, \boldsymbol{t}^n \rangle\}$, $W^* = \mathcal{F}_L(LS)$. An adversarial example is a solution of an optimization problem; $\boldsymbol{x}^A = \underset{\boldsymbol{x}}{argmin}\ A_\lambda(W^*; \boldsymbol{x}_S, \boldsymbol{t}_T, \boldsymbol{x})$, where

$$A_\lambda(W^*; \boldsymbol{x}_S, \boldsymbol{t}_T, \boldsymbol{x}) = \ell(\boldsymbol{y}(W^*; \boldsymbol{x}), \boldsymbol{t}_T) \ + \lambda \cdot \ell(\boldsymbol{x}, \boldsymbol{x}_S),$$

with a hyper-parameter λ. Such a data $\boldsymbol{x}^A$ is visually close to a seed $\boldsymbol{x}_S$ for human eyes, but actually adds faint noises so as to induce miss-inference such that $\mathcal{F}_I(\boldsymbol{x}^A) = \boldsymbol{t}_T$; we assume here $\langle \boldsymbol{x}_S, \boldsymbol{t}_T \rangle \notin LS$.

Our method uses this optimization problem, in which a seed $\boldsymbol{x}_S$ is $\boldsymbol{x}^n$ and its target label $\boldsymbol{t}_T$ is $\boldsymbol{t}^n$. Let $\boldsymbol{x}^{n*}$ be a solution of $\underset{\boldsymbol{x}}{argmin}\ A_\lambda(W^*; \boldsymbol{x}^n, \boldsymbol{t}^n, \boldsymbol{x})$ for $\langle \boldsymbol{x}^n, \boldsymbol{t}^n \rangle \in LS$. The method basically obtains a new data $\boldsymbol{x}^{n*}$ such that $\boldsymbol{x}^n$ is augmented with *noises*. Because the predicted label is not changed, $\boldsymbol{x}^{n*}$ is not adversarial, but may be *distorted* from the seed $\boldsymbol{x}^n$. When λ is very small, the distortion from $\boldsymbol{x}^n$ is large. If λ is appropriate, $\boldsymbol{x}^{n*}$ can be close to $\boldsymbol{x}^n$. By applying the method to all the elements in LS, a new dataset $\{\langle \boldsymbol{x}^{n*}, \ \boldsymbol{t}^n \rangle\}$ is obtained. Now, let $LS^{(K)}$ be $\{\langle \boldsymbol{x}^n, \boldsymbol{t}^n \rangle\}$. We define $T_\lambda(LS^{(K)})$ to generate a new dataset $LS^{(K+1)}$;

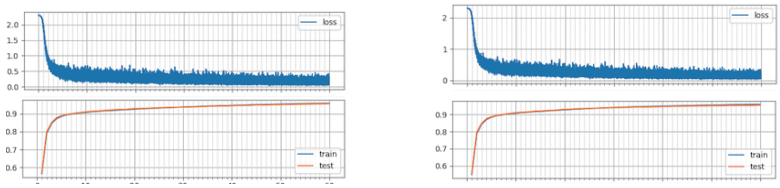
$$LS^{(K+1)} = T_\lambda(LS^{(K)}) \equiv \{\ \langle \underset{\boldsymbol{x}}{argmin}\ A_\lambda(\mathcal{F}_L(LS^{(K)}); \boldsymbol{x}^n, \boldsymbol{t}^n, \boldsymbol{x}),\ \boldsymbol{t}^n \rangle\ \}.$$

Because W^* used in A_λ is derived from a training dataset $LS^{(0)}$ of $\{\langle \boldsymbol{x}^n, \boldsymbol{t}^n \rangle\}$, the solution $\boldsymbol{x}^{n*}$ of the optimization problem can be $\boldsymbol{x}^n$ as a special case with appropriately chosen λ values. $LS^{(1)}$, a result of $T_\lambda(LS^{(0)})$, is not much different from $LS^{(0)}$ and may refer to the same machine learning task as $LS^{(0)}$ does. Contrarily, if we choose some small λ values or we apply the function T_λ repeatedly, the deviation of $\boldsymbol{x}^{n*}$ from $\boldsymbol{x}^n$ can be large, and $LS^{(K)}$ is mostly not *i.i.d.* We can obtain datasets in either LSS_P or LSS_N automatically.

5 A Case Study

5.1 Machine Learning Classifier for MNIST

MNIST dataset is a standard problem of classifying handwritten numbers. It consists of a training dataset LS of 60,000 vectors, and a testing dataset TS



(a) Probably Correct (ProgPC) (b) Bug-Injected (ProgBI)

Fig. 1. Loss and Accuracy: MNIST Dataset

of 10,000. Both LS and TS are randomly selected from a pool of vector data, and thus are considered to follow the same empirical distribution. The machine learning task is to classify an input vector data into one of ten categories from 0 to 9. An input data is presented as a sheet of 28×28 pixels, each representing gray scales. Pixel values comprise handwritten strokes, and a number appears as a specific pattern of these pixels.

In the experiments, the learning model is a classical neural network with a hidden layer and an output layer. Activation function for neurons in the hidden layer is $ReLU$; its output is linear to positive input values and a constant zero for negatives. The activation function of the output layer is *softmax*; it is a probability distribution over a variable with n values; $n = 10$ in this case.

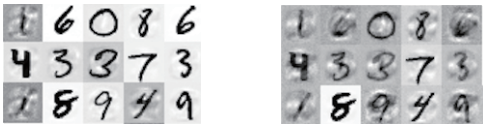
We prepared two learning programs $\mathcal{F}_L^{PC}$ and $\mathcal{F}_L^{BI}$. The former is a probably correct implementation of NN learning algorithms, and the latter is its bug-injected version. We conducted two experiments in parallel, one using $\mathcal{F}_L^{PC}$ and the other with $\mathcal{F}_L^{BI}$, and made comparisons. Their behavior in the learning process is shown in Figure 1. The graphs in Figure 1(b) are mostly similar to those in Figure 1(a). Note that the graphs in Figure 1(b) are the results of a bug-injected program. We cannot conclude that the learning program is free from faults just looking at loss and accuracy graphs [9].

5.2 Metamorphic Testing with Distorted Dataset

We will see how we conduct testing of MNIST machine learning programs with distorted datasets.

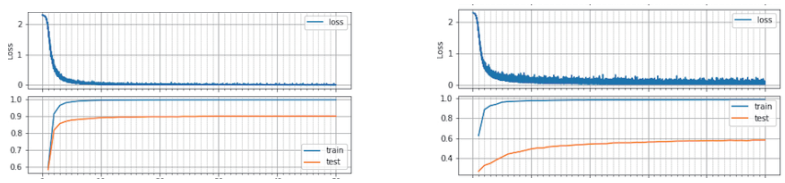
A Quick Review of Metamorphic Testing Metamorphic testing [1] is a property-based testing technique. Let $f(x)$ be a test target. For N input data $x^{(n)}$ ($N{\geq}2$ and $1{\leq}n{\leq}N$) and N execution results $f(x^{(n)})$, a $2N$ relation $\mathcal{R}(x^{(1)}, \ldots, x^{(N)}, f(x^{(1)}), \ldots, f(x^{(N)}))$ is a metamorphic relation (MR) [2]. If the MR is violated for a particular combination of inputs, then we consider that the program $f(x)$ may contain some faults.

MR, employed in various literature, takes a form simpler than the above. It consists of two sub-relations: an input sub-relation T which only involves source and follow-up inputs in the form of a mapping, and an output sub-relation which



(a) Probably Correct (ProgPC) (b) Bug-Injected (ProgBI)

Fig. 2. Distorted Data



(a) Probably Correct (ProgPC) (b) Bug-Injected (ProgBI)

Fig. 3. Loss and Accuracy: Distorted Dataset

involves only the outputs of the source and follow-up inputs. Given a source data x, the translation function T generates a follow-up data $T(x)$. An MR, $Rel_T(f(x), f(T(x)))$, is violated if f is not correct with respect to its functional specifications.

When test targets are machine learning programs, the function T must work not on a single discrete data, but on a dataset so as to build up the dataset diversity [9]. Finding appropriate Rel_T together with such a T is key in conducting the metamorphic testing.

Follow-up Dataset We generated distorted datasets; $LS^{(1)} = T_\lambda(LS^{(0)})$ where $LS^{(0)}$ is the original MNIST training dataset. The formula takes the form of the translation function T, which works on a dataset $LS^{(0)}$. Thus, T_λ in Section 4 can be considered as a function to generate a follow-up dataset.

Figure 2 shows a fragment of $LS^{(1)}_C$ where C is either PC or BI. All the data are not so clear as those of the original MNIST dataset $LS^{(0)}$ and thus are considered as distorted. Furthermore, for human eyes, Figure 2 (b) is unclear as compared with Figure 2 (a). It implies that $LS^{(1)}_{BI}$ is more distorted than $LS^{(1)}_{PC}$.

Metamorphic Relations In Figure 3, the two accuracy graphs for the MNIST testing dataset TS show different behavior. This indicates that the ProgBI case is less accurate than the ProgPC case; the accuracy of the ProgPC case (Figure 3(a)) is about 90%, while the ProcBI case (Figure 3(b)) is about 60%. The distorted follow-up data are thus useful in that the accuracy graphs show some discrepancy.

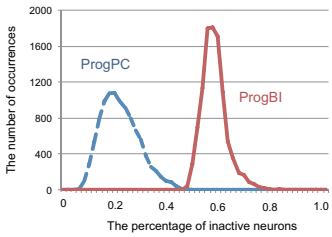


Fig. 4. Frequencies of Inactive Neurons

Comparing Figure 1 and Figure 3, we notice some differences in the accuracy graphs for TS. Note that Figure 1 are monitored results of $\mathcal{F}_L^C(LS^{(0)})$ where $LS^{(0)}$ is the original MNIST training dataset, and that Figure 3 are those of $\mathcal{F}_L^C(LS^{(1)})$ where $LS^{(1)} = T_\lambda(LS^{(0)})$. For the case of ProgBI, the accuracy graph for the MNIST testing dataset TS reaches nearly 100% (Figure 1(b)), while the accuracy in Figure 3(b) is about 60%. For the case of ProgPC, the accuracy in Figure 1(a) is nearly 100% as well. The graph in Figure 3(a) shows that the accuracy is about 90%, which is still not so bad.

Therefore, we define a metamorphic relation such that it compares the two resultant accuracies, one calculated with the learning model trained against $LS^{(0)}$ and another for the case of $LS^{(1)}$. If the difference is larger than a given threshold, we conclude that the test is failed. Particularly, for this case study, we may choose 0.2 as the threshold so that the MR is violated for ProgBI.

5.3 Degrees of Distortion

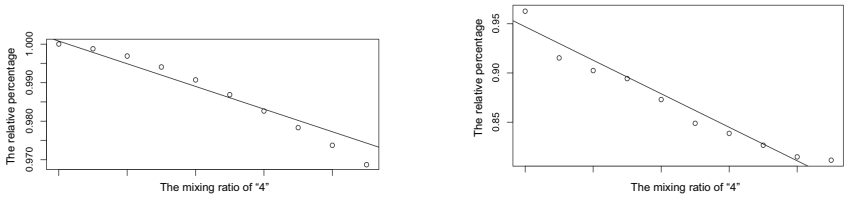
We think that a certain degree of distortion in trained learning models W^* must account for distorted datasets, since the generation method in Section 4 refers to trained learning models of $\mathcal{F}_L$. We now have a question what metrics are suitable for the degrees of distortion in the trained learning models. We will study below *neuron coverage* [10] whether it is appropriate.

Neuron Coverages and Distorion A neuron is said to be *activated* if its output signal *out* is larger than a given threshold when a set of input signals in_j is presented; $out = \sigma(\sum w_j \times in_j)$. The weight values w_js constitute the trained learning model W^*. *Activated Neurons* refer to a set of neurons to have values larger than a given threshold when a vector data $\boldsymbol{a}$ is input as in $\boldsymbol{y}(W^*; \boldsymbol{a})$. *Neuron coverage (NC)* is defined in terms of the size of the two sets; $NC = |Activated\ Neurons|/|Total\ Neurons|$.

We focus on the neurons in the hidden layer of our NN learning model. As its activation function is *ReLU*, we choose 0 as the threshold. Figure 4 is a graph to show the numbers of input vectors in the distorted dataset leading to $(1 - NC)$ neurons (i.e. inactive neurons).

(a) 6 mixed with 4 (b) 4 mixed with 6

Fig. 5. Mixing Two Numbers



(a) Softmax Values (b) Inactive Neurons

Fig. 6. Metrics for Mixed Numbers: 6 mixed with 4

As observed in Figure 4, the graph for the case of ProgPC shows that the ratio of inactive neurons is almost 20%; i.e. 80% of neurons in the hidden layer are activated for the classification results. However, the ProgBI graph shows that about 60% of them are inactive and do not contribute to the results. This difference in the ratios of inactive neurons implies that the trained learning model W^*_{BI} of ProgBI is more distorted than W^*_{PC} of ProgPC.

Mixing Data and Neuron Coverages We extend the proposed dataset generation method to synthesize a new vector data by mixing two data. B_λ mixes $\boldsymbol{x}_S$ with $\boldsymbol{x}_M$ to generate a new data $\boldsymbol{x}^*$ such that its tag is $\boldsymbol{t}_S$.

$$B_\lambda(W^*; \boldsymbol{x}_S, \boldsymbol{t}_S, \boldsymbol{x}_M, \boldsymbol{x}) = \ell(\boldsymbol{y}(W^*; \boldsymbol{x}), \boldsymbol{t}_S) \ + \lambda_S{\cdot}\ell(\boldsymbol{x}, \boldsymbol{x}_S) + \lambda_M{\cdot}\ell(\boldsymbol{x}, \boldsymbol{x}_M),$$

where $\lambda_S + \lambda_M = \lambda$ (a constant). Figure 5 shows two series of number data with a variety of λ_S and λ_M combinations. The leftmost image in Figure 5(a) is the case where $\lambda_{S=6} = \lambda$ and the rightmost one is that $\lambda_{S=6} = 0$. We see, in Figure 5(a), the images change their shapes of 6 to seemingly 4, and reach an adversarial example case; the rightmost one is visually seen as 4 but the inferred tag is 6.

Figure 6 shows two graphs that are calculated for the case of Figure 5(a). Figure 6 (a) is a graph to show the ratio of softmax values of the output signal of the anticipated supervisor tag 6. The graph values are normalized with respect to the softmax value of the leftmost one. As all the values are nearly 1.0, we can say that $\mathcal{F}_I$ shows a good prediction performance, although the softmax value at the rightmost is slightly smaller, about 3% less, than that the leftmost.

Figure 6 (b) is a graph to show the ratio of inactive neurons. The value at the rightmost one is about 15% less than that at the leftmost. In other word, about 15% more neurons are activated to infer in the adversarial example case.

This may imply that adversarial examples activate neurons that are *inactive* for normal data.

6 Related Work and Discussion

DeepRoad [16] adopts an approach with Generative Adversarial Networks (GAN) [3] to synthesize various weather conditions as driving scenes. GAN is formulated as a two-player zero-sum game. Given a dataset whose empirical distribution is ρ^{DS}, its Nash equilibrium, solved with Mixed Integer Linear Programing (MILP), results in a DNN-based generative model to emit new data to satisfy the relation $\boldsymbol{x} \sim_{i.i.d.} \rho^{DS}$. Thus, such new data preserve characteristics of the original machine learning problem. Consequently, we regard the GAN-based approach as a method to enlarge coverage of test scenes within what is anticipated at the training time.

Machine Teaching [18] is an inverse problem of machine learning, and is a methodology to obtain a dataset to optimally derive a *given* trained learning model. The method is formalized as a two-level optimization problem, which is generally difficult to solve. We regard the machine teaching as a method to generate unanticipated dataset. Obtained datasets can be used in negative testing.

Our method uses an optimization problem with one objective function for generating datasets that are not far from what is anticipated, but probably are biased to build up the dataset diversity.

Last, reconciling learning algorithms with biased training dataset has been one of the major concerns in machine learning research [11]. The view therein is that biased distribution is what to be avoided. Contrarily, this paper makes use of such biased datasets to conduct either positive or negative testing.

7 Concluding Remarks

The proposed dataset generation method employs the L-BFGS [13], but the other methods on adversarial input generation (e.g. [14]) can be adapted as well. Since our study only involves the MNIST dataset with a simple NN learning model, studies on various DNN learning models, such as CNN, are desirable. In addition, further studies are needed to see whether the neuron coverage [10] is effective as a metric for the distortion degrees.

Acknowledgment

The work is supported partially by JSPS KAKENHI Grant Number JP18H03224, and is partially based on results in a project commissioned by the NEDO.

References

1. T.Y. Chen, S.C. Chung, and S.M. Yiu : Metamorphic Testing - A New Approach for Generating Next Test Cases, HKUST-CS98-01, The Hong Kong University of Science and Technology, 1998.
2. T.Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T.H. Tse, and Z.Q. Zhou : Metamorphic Testing: A Review of Challenges and Opportunities, *ACM Computing Surveys* 51(1), Article No.4, pp. 1-27, 2018.
3. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, anY. d Bengio : Generative Adversarial Nets, In *Adv. NIPS 2014*, pp.2672-2680, 2014.
4. I. Goodfellow, Y. Bengio, and A. Courville : *Deep Learning*, The MIT Press 2016.
5. S. Haykin : *Neural Networks and Learning Machines (3ed.)*, Pearson India 2016.
6. J.J. Heckman: Selection Bias as a Specification Error, *Econometrica*, vol.47, no.1, pp.153-161, 1979.
7. S. Nakajima and H.N. Bui : Dataset Coverage for Testing Machine Learning Computer Programs, In *Proc. 23rd APSEC*, pp.297-304, 2016.
8. S. Nakajima : Quality Assurance of Machine Learning Software, In *Proc. GCCE 2018*, pp.601-604, 2018.
9. S. Nakajima : Dataset Diversity for Metamorphic Testing of Machine Learning Software, In *Proc. 8th SOFL+MSVL*, pp.21-38, 2018.
10. K. Pei, Y. Cao, J. Yang, and S. Jana : DeepXplore: Automated Whitebox Testing of Deep Learning Systems, In *Proc. 26th SOSP*, pp.1-18, 2017.
11. J. Quinonero-Candela, M. Sugiyama, A. Schwaighofer, and N.D. Lawrence (eds.) : *Dataset Shift in Machine Learning*, The MIT Press 2009.
12. S. Segura, D. Towey, Z.Q. Zhou and T.Y. Chen: Metamorphic Testing: Testing the Untestable, IEEE Software (in press).
13. C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus: Intriguing properties of neural networks, In *Proc. ICLR 2014*, 2014.
14. D. Warde-Farley and I. Goodfellow: Adversarial Perturbations of Deep Neural Networks, in *Perturbation, Optimization and Statistics*, pp.1-32, The MIT Press 2016.
15. X. Xie, J.W.K. Ho, C. Murphy, G. Kaiser, B. Xu, and T.Y. Chen : Testing and Validating Machine Learning Classifiers by Metamorphic Testing, *J. Syst. Softw.*, 84(4), pp.544-558, 2011.
16. M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid: DeepRoad: GAN-Based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems, In *Proc. 33rd ASE*, pp.132-142, 2018.
17. Z.Q. Zhou and L. Sun: Metamorphic Testing of Driverless Cars, *Comm. ACM*, 62(3), pp.61-67, 2019.
18. X. Zhu : Machine Teaching: An Inverse Problem to Machine Learning and an Approach Toward Optimal Education, In *Proc. 29th AAAI*, pp.4083-4087, 2015.