Technical Report on Machine Learning Quality Evaluation and Improvement

2nd English Edition

January 20, 2023
(a revision of the Japanese version published on August 2, 2022)

Technical Report DigiARC-TR-2023-02
Digital Architecture Research Center
National Institute of Advanced Industrial Science and Technology (AIST)

Technical Report CPSEC-TR-2023002
Cyber Physical Security Research Center
National Institute of Advanced Industrial Science and Technology (AIST)

Technical Report
Artificial Intelligence Research Center
National Institute of Advanced Industrial Science and Technology (AIST)

Foreword

In the project "Research and Development on the Quality Assessment Reference and Testbed of Machine-Learning /artificial intelligence systems" (JPNP20006) commissioned by the New Energy and Industrial Technology Development Organization (NEDO), we are developing Machine Learning Quality Management Guidelines [1] to explain the quality of machine learning. While developing the guidelines, we have also been researching and developing techniques for evaluating and improving the quality of machine learning. Although this research and development is still ongoing, since we have obtained technical knowledge on the quality evaluation described in the Machine Learning Quality Management Guidelines, we report on the progress of this research and development for the recent three years (FY 2019~2021).

Table of Contents

1 Introduction

Machine Learning Quality Management Guideline has been developed to clearly explain the quality of various industrial products including statistical machine learning (3rd Edition [1]). The third edition of the guideline focuses on the nine internal quality characteristics (e.g., Stability of the trained model, Reliability of underlying software system, etc.) for machine learning systems, but techniques for evaluating and improving these internal quality characteristics have not been sufficiently established yet. This document reports the current results on survey, research, and development of techniques for evaluating and improving the internal quality characteristics, which are being conducted for supporting the development of the guideline.

1.1 Overview of this research and development

Figure 1.1 shows the relationship between the machine learning quality evaluation and improvement techniques (the center yellow boxes in Figure 1.1, where the number in each box shows the chapter number explained in this report) that were researched and developed for the recent three years (FY 2019~2021). The relations to the phases of the machine learning model lifecycle and the nine internal quality characteristics are also shown. The techniques are briefly introduced here, and the details are explained in Chapters 2 ~ 9.

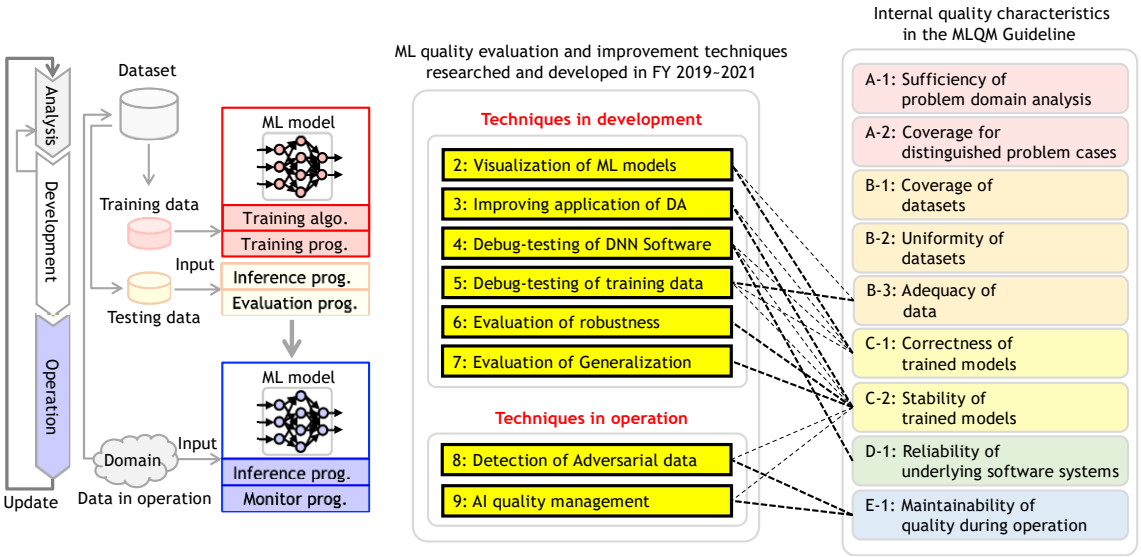


Figure 1.1 Machine learning quality evaluation and improvement techniques in this report

– Visualization of Machine Learning Models (in Chapter 2) :
To support the quality evaluation work of machine learning models, we attempted to visualize the difference and comparison results between multiple models and the sensitivity of the workers (annotators and model designers) reflected in each model. We proceeded with the implementation of a tool to visualize the work procedures of the workers involved in creating the models and their influence on the models with

multiple views[2][3].

- Improved Quality through Better Application of Data Augmentation (in Chapter 3):
  To improve the data-diversity obtained by data augmentation and increase accuracy and stability in deep learning, we devised new two data augmentation methods with simple algorithms (FC-mixup and Latent DA) and report the results of their impact on generalization performance in experiments [4]. In addition, for the Latent DA method, we have been developing AdaLASE, for dynamically selecting appropriate layers for the data augmentation.

- Debug-Testing of DNN Software (in Chapter 4):
  The failures of DNN (Deep Neural Network) models can be considered from two viewpoints of causes. One of them is the direct cause during inference (by prediction and inference programs) and the other one is the root cause during training (by training and learning programs, training models, and training data). We proposed an indicator and an analysis method for evaluating the presence of bugs in training programs by the internal information (e.g., neuron coverage) of DNN models, and then confirmed that the indicator is useful by experiments [5].

- Debugging and Testing of Training Data (in Chapter 5):
  For the case that failures in DNN (Deep Neural Network) models are caused by training data bias, we researched methods for detecting such bias from two quality viewpoints: model accuracy and model robustness. We proposed a method to evaluate the bias by the internal states (e.g., neuron coverage) in the DNN models and confirmed that the method is useful for debugging the training data by experiments.

- Evaluation and Improvement of Robustness (in Chapter 6):
  To evaluate and improve robustness of machine learned models, we report on the results of a survey on methods to measure the maximum safe radius (the maximum value of noise that can be guaranteed not to cause misclassifications) as a measure of robustness for input noise including adversarial examples, and methods to increase the safe radius.

- Generalization Bounds of Machine-Learned Models (in Chapter 7):
  To evaluate the generalization performance of machine-learned models, we report on the results of a survey on theorems for generalization bounds, that are the upper bounds on the expected values of the error rates (i.e., generalization errors) for all inputs, including unseen data-samples. Then, we confirmed that (perturbated) generalization bounds based on testing errors (i.e., test-set bounds) can be used for evaluating the generalization performance, by experiments.

- Adversarial Example Detection (in Chapter 8):
  To establish a practical method for detecting adversarial examples, we report on the results of a survey on the state-of-the-art adversarial example detection methods and classifies them into four main categories, and then present the results of follow-up experiments on representative methods. Consequently, we confirmed that NIC method shows the highest detection rate. Then, we constructed the NIC framework for detecting adversarial examples based on the NIC method and evaluated it by the Kullback-Leibler divergence for explaining the reason why the NIC method is effective.

- AI Quality Management in Operation (in Chapter 9):
  To maintain quality of machine learning models even for unseen data and/or changing trends during operation, we report on the results of a survey on detection and adaptation methods for changes in input-data distribution over time (e.g., concept drift), and also a survey on the latest unsupervised domain adaptation methods (e.g., label-shift). The surveys include not only supervised methods but also unsupervised/ semi-supervised methods that are promising approaches from the viewpoints of operational costs and practical adaptability.

1.2 Author list

The authors of each chapter are as follows:

- Chapter 1: Yoshinao Isobe (AIST CPSEC)
- Chapter 2: Yuri Miyagi (AIST AIRC)
- Chapter 3: Tomoumi Takase (AIST AIRC)
- Chapter 4: Shin Nakajima (NII)
- Chapter 5: Shin Nakajima (NII)
- Chapter 6: Yoshinao Isobe (AIST CPSEC)
- Chapter 7: Yoshinao Isobe (AIST CPSEC)
- Chapter 8: Yusei Nakashima and Keiichi Nishida (Techmatrix)
- Chapter 9: Yoshihiro Okawa and Kenichi Kobayashi (Fujitsu)

1.3 Acknowledgements

This report is based on results obtained from the project "Research and Development on the Quality Assessment Reference and Testbed of Machine-Learning /artificial intelligence systems" (JPNP20006), commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

2 Visualization of Machine Learning Models

Information visualization is becoming a popular method to support the analysis of the structure and behavior of machine learning models, which are known as black boxes. We have started research on a new method for visualizing machine learning models with the following two objectives:

- Visualization of differences and comparison results between multiple models
  - Implementation of visualization based on expressions that are easy for humans to interpret and understand
- Visualization of the sensitivity of workers (annotators of training data, designers of model structures) reflected in the model
  - Proposal of new factors that can be used for quality assessment

In this chapter, we first describe the results of a survey of recent machine learning model visualization techniques. Then, we introduce the results of a prototype visualization tool for observing model and worker information, developed in 2020-2021, and our future implementation policy.

2.1 Survey on methods to support using machine learning

The basic purpose of visualization methods for machine learning is to improve the interpretability of models, and this is closely related to XAI (Explainable AI), which has attracted attention in recent years. There are no definitive definitions or evaluation methods for XAI, however, many papers about the classification of XAI are published, and we can devise visualization objectives and methods along these lines. In [6], the approaches to increase interpretability are classified into four categories:

(1) Total explanation (Approximation of a complex model structure by a simple model)
(2) Partial explanation (Explaining the rationale for decisions about model output results)
(3) Design of explainable models (Creation of readable models at the design stage)
(4) Explanation of the deep learning model (e.g., Highlighting the parts of the image data that the model recognizes)

Especially (2) and (4) have much room for contribution by visualization. These machine learning visualization methods are continuously being studied, and the number of survey papers is increasing due to the diversity of applications and target cases. For example, Hohman et al. [7] described and classified deep learning visualization methods according to the 5W1H elements. It also presents several overall directions and issues in the field of deep learning visualization. Especially "improving interactions for model evaluation" and "improving interpretability through active human involvement in models" are closely related to our research, which aims to develop visualization methods for quality evaluation.

As research on machine learning visualization progresses and becomes popular in the real world, there is a growing tendency for complex analysis to be performed in a single visualization view. In the past, visualization methods basically focused on detailed analyses of single models specialized in either data ([8]) or model structure ([9]). However, in recent years, research has been conducted on combined visualization methods for data and model structure, as well as methods that aim to compare multiple models. The number of elements that make up a machine learning model is enormous, and it takes a lot of time and effort to create visualization results for the number of models and compare them side by side. Besides, the differences in structure and accuracy between the models to be compared are often small and features of the models may be overlooked. Therefore, there is a high need for a visualization method that uses expressions that emphasize the differences so that the differences can be found efficiently within a limited screen. (For example, in [10], the pipeline from data input to output, hyperparameter values, etc. for more than 10 models can be compared on a single screen.)

So far, we have introduced trends and examples of visualization methods regarding the properties and accuracy of the models themselves. In parallel with this, we have also investigated how the workers (annotators of training data, designers of model structures) involved in model creation interact with the models. In fields such as image recognition, models with accuracy beyond human recognition capabilities have been developed, but there is a persistent suggestion, regardless of the field, that active human intervention is desirable to improve the accuracy of models. There are many papers that discuss the following items regarding the relationship between AI and humans and effective intervention methods in the modeling process:

- Introduction of operations (adjustment and evaluation) to improve the accuracy of the model in the learning process
- Designing an interface that is easy to use and can maintain the motivation of the operator
- Collaboration with related fields such as cognitive science and psychology

As an example, Amershi et al. examined the psychological state of workers who were assigned feedback to evaluate and improve several models [11]. The authors found that the workers preferred to be able to directly tell the correct processing steps to models. They also said that workers get more motivated to give more active feedback when they find their actions are improving the accuracy of the model. Although there seem to be few examples of visualization of such information about the workers themselves and the impact of each worker on the model, it can be adopted as a ground for quality assurance as follows:

- Show that their knowledge is sufficiently reflected in the model's behavior when domain or machine learning experts participated in the creation of the model.
- Indicate which workers' behavior is strongly reflected in the model and use this as a clue to identify elements (training data, parameters, etc.) that should be adjusted.

2.2 Visualization of model structure and worker information

Based on the above research results, we place particular importance on "comparative visualization of multiple models" and "visualization of worker's sensitivity" among machine learning model visualization methods. We proceed to design a visualization tool with both properties. Figure 2.1 shows an overview of the proposed method.

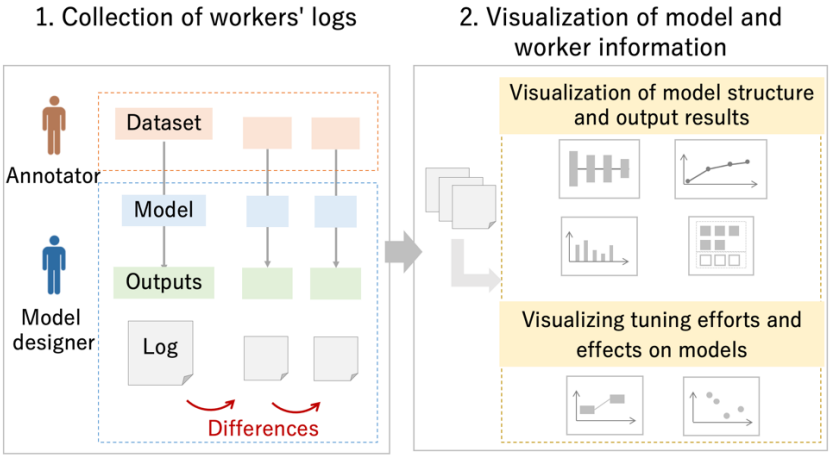


Figure 2.1 Overview of machine learning models and worker information visualization methods

2.2.1 Logging of differences between models

First, we collect logs of the structure of the model to be visualized (the adjustment process and test results). In the current implementation, we suppose image classification as a case study and obtain the model designer's parameter adjustment process and test results as text files using Cometl.ml, a machine learning experiment management tool. For the annotators, we do not directly collect work logs, but indirectly evaluate their work based on how the model designers selected data and applied preprocessing.

From these logs, we calculate differences between models (the amount of change from the model used immediately before). Differences between models are classified into three categories: training data, model structure, and optimization algorithm, and are calculated for each. The difference in training data is calculated by adding up the data used, the number of classes, and the difference in parameters used for preprocessing. The difference in model structure is obtained by creating pairs of layers that comprise the two models and summing the dissimilarities (differences in layer types and parameters) of each pair. For the difference in optimization algorithms, a constant is assigned if the algorithm types are different. If they are the same, the difference is calculated from the difference in parameters. After obtaining the three types of differences, we obtain the overall change in the model by summing these values.

2.2.2 Developing prototype of model difference visualization tool

In 2020, we implemented views on basic information such as model structure and output results, and in 2021, added views to visualize the progress of model adjustments and testing by workers.

Figure 2.2 shows the overview of prototype visualization views, which visualize the results of MNIST for two simple models developed in 2020. Assuming that the main users of this tool would be model designers and considering the possibility that users who were not familiar with visualization would be included, we combined basic visualization methods (line graphs, bar graphs, etc.) and implemented them with the policy of actively linking them (e.g., highlighting related parts).

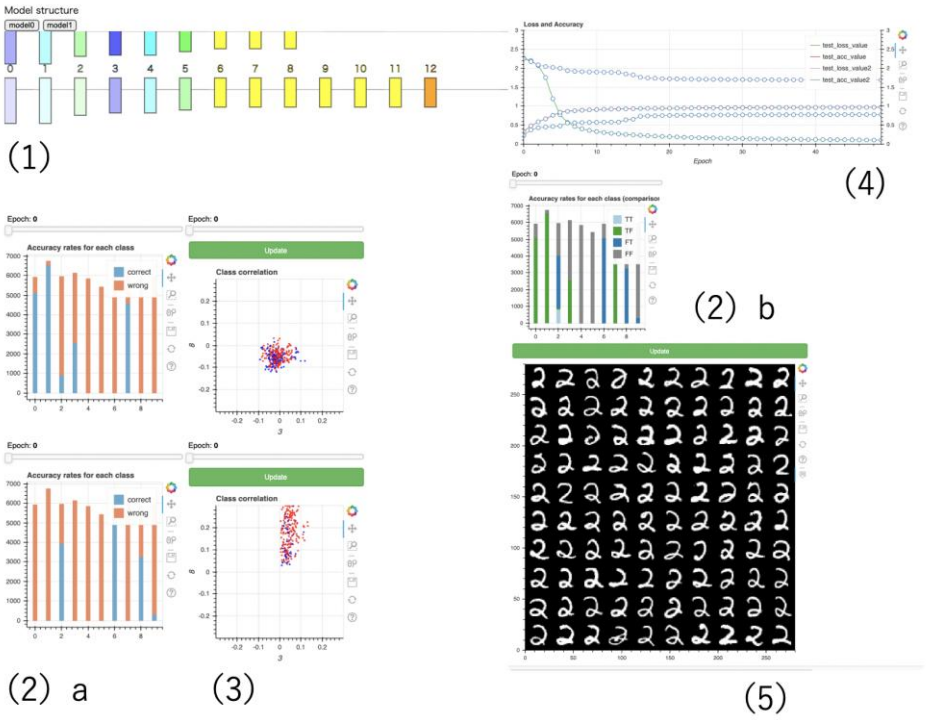


Figure 2.2 Visualization views on model structure and output results

We created the tool on JupyterLab, mainly using the machine learning library PyTorch and the visualization library Bokeh, so that we could compare the features of the two models:

(1) Network of each model structure
(2) Bar graph of output results for each class
    ① Visualization for each model
    ② Visualization of the difference between two models
(3) Scatter plot of output result correlation between two selected classes for each model
(4) Line graph of accuracy
(5) Thumbnail list of data classified with particularly high (low) confidence

Figure 2.3 shows an example of the results of classifying the output to MNIST for two models. The horizontal axis represents the class from 0 to 9, and the vertical axis represents the amount of data. The color-coding of each bar represents the combination of correct (T) and incorrect (F) answers for the two models, for example, where TF (FT) means that only model 1 (2) correctly classified. Immediately after the start of learning (Figure 2.3, left), model 1 had a high percentage of correct answers in classes 0, 1, and 7, and model 2 had a high percentage of correct answers in classes 2, 6, and 8, indicating that each model had different strengths. At the advanced stage of learning (Figure 2.3, right), both models had high percentages of correct answers in many classes. Besides, model 1 has a high percentage of correct answers, including classes 3, 4, 5, and 7, which model 2 is not good at, indicating that model 1 is more advanced in learning than model 2 at this stage.

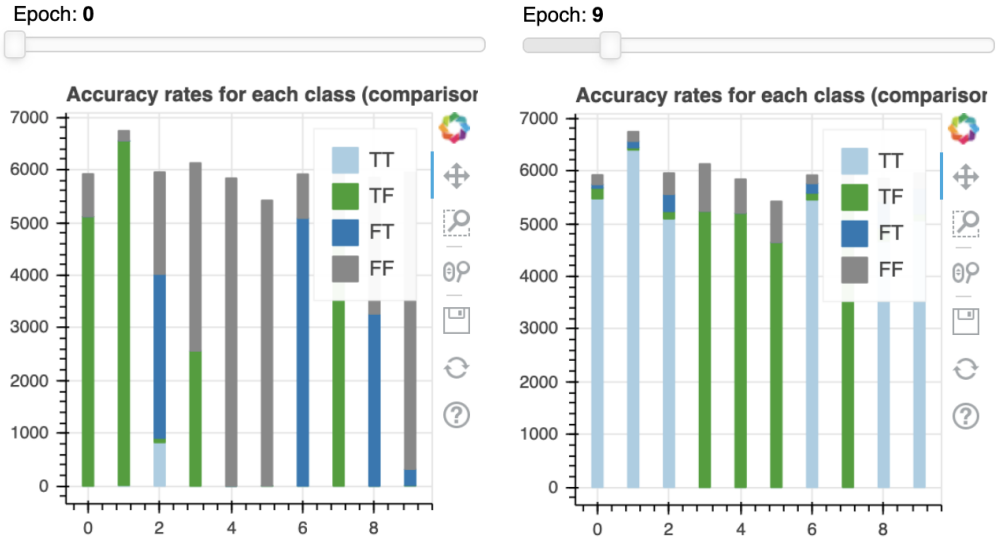


Figure 2.3 Examples of comparing the output results of two models

Next, we introduce the visualization results generated by the time-series visualization function for model test results, created in 2021. Figure 2.4 visualizes the amount of change in accuracy and model structure when multiple image classification models are tested in sequence. The horizontal axis represents the order of the tests, and the vertical axis means top-1 for each model. The boxes colored gray to yellow correspond to one model. The tool visualizes three small icons inside each box, except for the box for the first model used. The colors of these icons and boxes represent the amount of change from the model used immediately before, with higher saturation meaning greater change. Specifically, the top icon (red) represents the training data, the middle icon (blue) represents the model structure, and the bottom icon (green) represents the amount of change related to the optimization algorithm. The boxes (yellow) reflect the total amount of these changes. The center coordinates of each box are connected by edges to clearly indicate the change in accuracy.

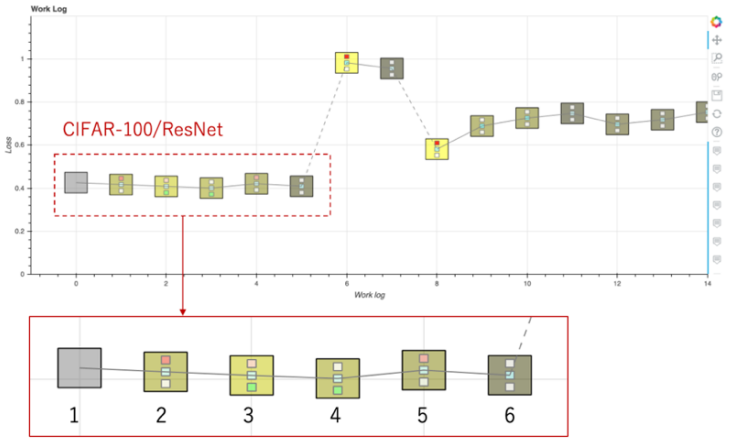


Figure 2.4 An example of visualization of a worker's model adjustment history

Table 2.1 Parameters of the model used for work history visualization

Index	l	m	p	a	d
1	0.1005	0.9	0	0.4	18
2	0.1005	0.9	0.45	0.4	18
3	0.06	0.5495	0.409	0.55	18
4	0.06	0.919	0.409	0.55	18
5	0.0335	0.919	0.2955	0.287	18
6	0.0335	0.919	0.2955	0.287	34

The area circled in red in Figure 2.4 shows the visualization results of the process of adjusting the ResNet model trained on CIFAR-100. Referring to the hyperparameter tuning scenario conducted in [12], the model was trained and tested 6 times and logged while changing the parameters as shown in Table 2.1. l is the learning rate, m is the momentum value. p and a are the erasing probability and max erasing area when random erasing was applied to the training data. d is the depth of the ResNet model used.

The numbers in the red boxes in Figure 2.4 correspond to the Indexes in Table 2.1; in 2 and 5, of the three icons, the top icon representing changes in the training data is highlighted in orange. This reflects the fact that the values of p and a, parameters related to the training data, were changed significantly when going from model 1 to 2 and from 4 to 5. Similarly, in 3 and 4, the bottom icons are highlighted in yellow-green, indicating changes related to the optimization algorithm. Specifically, it reflects the change in l and m. The color of the box also indicates that the third model had the largest amount of change from the model used immediately prior. Compared to model 2, four parameters ($l, m, p,$ and a) have been changed. On the other hand, the overall change in model accuracy was small, indicating that the impact of this parameter adjustment procedure was limited.

The box to the right of the red area plots the progress of the test after changing the training data to MNIST and ImageNet. The two boxes near the center of Figure 2.4 are plotted in bright yellow, which coincides with the timing of the change in the data set and model used. Thus, we can observe the long-term working history of multiple cases, in addition to showing the detailed adjustment process of a particular model.

2.3 Future work

In future work, we would like to work on extending the visualization function with the following policy. First, we will aim to visualize the results of long-term evaluation of models and work contents and recommendations for improvement for a single or a small number of workers. Then, we will compare the work patterns of a large number of workers and visualize the similarity and classification results among models or workers from an overhead perspective. By observing these visualization results, we would like to be able to estimate the skill level of workers and classify their work characteristics (work patterns).

3 Improved Quality through Better Application of Data Augmentation

This chapter describes the results of developing a new method for applying data augmentation in neural network learning and evaluating its effect to learning quality through experiments.

3.1 Research purpose

Data augmentation is a technique to increase the number of samples by adding deformations to the data, and it is highly effective in deep learning, which has a tendency of performance degradation when the number of training samples is small. On the other hand, the effectiveness of data augmentation strongly depends on the data used, so the selection of data augmentation methods and the parameters of each method must be set appropriately. However, theoretical analysis of data augmentation is difficult, and general ways to use it have not yet been established. This leads to unintentional and inappropriate use, which in turn compromises the quality of learning. In fact, there are many cases that training performance is degraded by setting inappropriate values for the amount of deformation of each data augmentation method, such as mask size or rotation angle, or where the user is puzzled as to what data augmentation method to select for the actual data to be used.

Therefore, to move away from the empirical use of data augmentation, this study focused on data diversity. Increasing diversity is the essential goal of data augmentation, and it has been demonstrated in the work of [13] that increasing diversity has a significant impact on improving generalization performance. Recently, a technique called RandAugment [14], which dynamically applies randomly selected operations from multiple data augmentation operations during training, has attracted much attention, but while it greatly improves diversity, effectively using it is not easy because many parameters need to be adjusted. In this study, we proposed the following two new methods for applying data augmentation related to data diversity, and improved the algorithms and evaluated their performance.

- We apply data augmentation at various layers of the neural network, including hidden layers, and perform automatic optimization of the applied layer (Section 3.2).
- We improve the Mixup method [15], a promising data augmentation method, and propose a new way to mix samples (Section 3.3).

3.2 Improved application layer for data augmentation

3.2.1 Data augmentation at hidden layers

Generally, data augmentation is considered to be applied to input data, but in neural networks, it is also possible to apply data augmentation to hidden layers. There are several previous studies on this subject, but most of them are not versatile methods, such as Manifold

mixup [16], which limits the method to mixup [15], or other methods that require specific networks and datasets. In this study, we considered applying various data augmentation methods used for image data, such as affine transformation and mask processing, in the hidden layers. Since features are extracted hierarchically in CNNs, data augmentation can be applied in various layers randomly selected for each minibatch to generate a wide variety of samples. As with application to input images, data augmentation can be applied to the feature maps obtained at the intermediate layers, making implementation easy.

An example of actual application of mask processing and translation to an input image and feature map is shown in Figure 3.1. Here, a sample is input to the model in training, and the images are shown in the upper row, aligned in size, immediately after data augmentation was applied at different layers with the same parameters (mask position and translation amount). The feature maps in the final layer of the sample are shown in the lower row. They are different images depending on the layer where the data augmentation was applied. This result shows that data augmentation at various layers leads to an increase in the diversity of the generated data and results in learning different from when data augmentation is applied only to the input data.

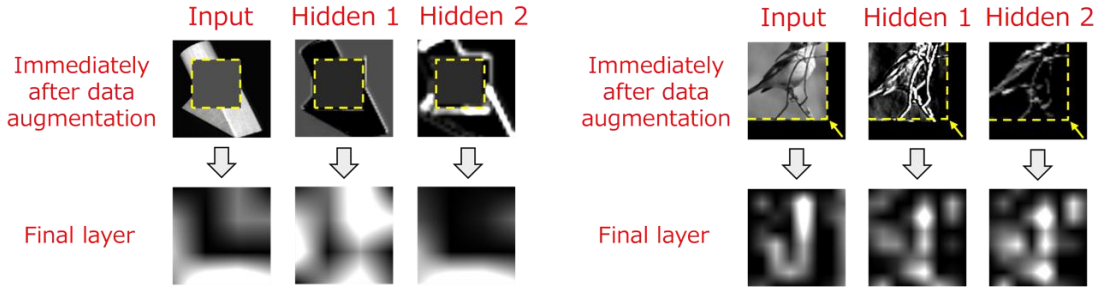


Figure 3.1 Example of applying data augmentation to input images and feature maps obtained at hidden layers

To compare the performance of data augmentations in the input layer and that in feature maps, we used various data augmentations and obtained test accuracies for models trained with supervision. Here, WideResNet28-10 was trained for 200 epochs using the CIFAR-10, Fashion-MNIST, and SVHN (without extra data) datasets. The results are shown in Figure 3.2. In each figure, the horizontal axis represents the accuracy [%] of the conventional method (Input DA) and the vertical axis represents the accuracy of the proposed method (Latent DA). As can be seen from these results, the proposed method tends to show higher accuracy than the conventional method, and the proposed method presented higher accuracy even in cases where the conventional method presented lower accuracy, such as the results using Crop. These results indicate that the diverse samples generated by the application of data augmentation to random layers are effective in improving performance.

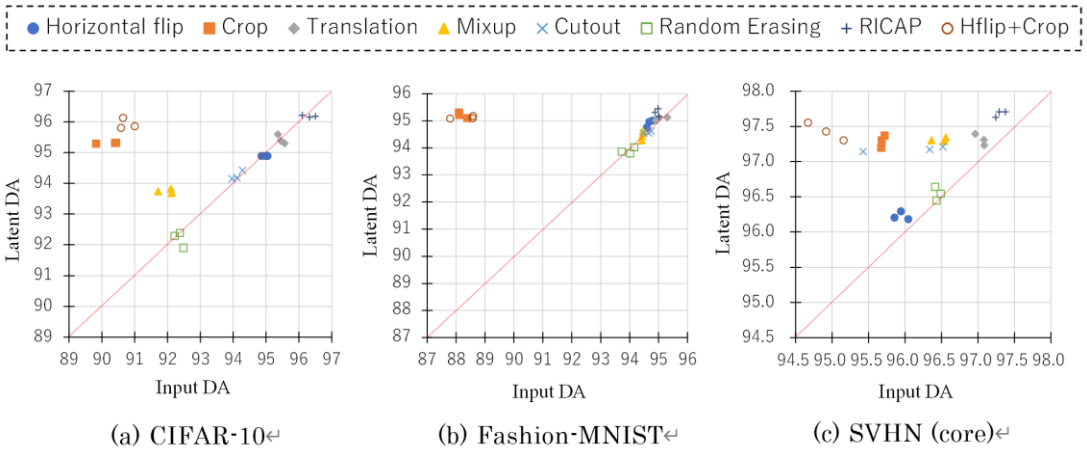


Figure 3.2 Comparison of test accuracy between input DA and latent DA

3.2.2 **Selecting appropriate layers for data augmentation**

Although previous studies have shown that data augmentation at hidden layers is effective, the question arises as to which layer is optimal for data augmentation. Although it is possible to find the optimal layer by repeatedly training with different layers of data augmentation and comparing the values of validation accuracy, it is an inefficient and impractical method because it increases the overall training time. Therefore, in this study, we worked on developing a method to dynamically discover the optimal layer for data augmentation in a single training session.

The approach is to prepare a parameter called the acceptance rate for each layer, update the acceptance rate during training, and apply data augmentation in the layer selected probabilistically according to the acceptance rate. The updating of the acceptance rate is done using the gradient descent method as shown below.

$$q_l \leftarrow q_l - \eta \frac{\partial L_{val}}{\partial q_l},$$

where q_l is the acceptance rate of layer l, L_{val} is the value of the error when the validation data is input, and η is the step width of the update. In practice, the values of the validation data should not be included in the algorithm, so the update is performed by creating pseudo-validation data with the training data with data augmentation. In the initial state of training, all acceptance rates are set to equal values so that the sum is 1, and the acceptance rate is updated for each minibatch. This optimization is expected to improve the generalization performance by increasing the acceptance rate of layers suitable for data augmentation and decreasing the acceptance rate of layers unsuitable for data augmentation.

We named this method Adaptive Layer Selection (AdaLASE) and compared it to conventional methods. Using CIFAR-10 as the data and ResNet18 as the model, we compared test accuracies for no data augmentation, data augmentation on input, data augmentation at random layers, and AdaLASE. Figure 3.3 (a) and (b) show the results using Cutout and Mixup, respectively. The mean and standard deviation of the accuracy for five different initial values are shown for each method. These results show that AdaLASE can perform as well as or better than conventional methods.

Future plans include a detailed analysis of how layers are selected and whether AdaLASE is working properly by looking at the change in acceptance rate during training.

(a) Cutout (b) Mixup

Figure 3.3 Comparison of test accuracy between AdaLASE and conventional methods

3.3 Proposal for a new mixing method by improving Mixup

In actual training, data augmentation often involves the simultaneous use of multiple methods, such as cropping, rotating, and flipping. Therefore, we focus on the compatibility between methods when multiple methods are used in this way, and in particular, we consider discussing the compatibility from the viewpoint of data diversity. As a first step in this approach, we propose a new method that is a variant of an existing method and use it simultaneously with the original method to increase the diversity of the data generated and improve performance. The method for formulating the diversity is described in the work of [13]. In this study, we first compare only the accuracy and verify whether the proposed method improves the performance.

Here, we have improved Mixup [15], one of the data augmentation methods. This method generates a new sample by linear interpolation of two samples, and takes the same ratio of linear interpolation for each of the input values and labels, as shown in the following equation.

$$\begin{cases} \tilde{x} = \lambda x_i + (1-\lambda)x_j \\ \tilde{y} = \lambda y_i + (1-\lambda)y_j \end{cases},$$

where (x_i, y_i) and (x_j, y_j) represent the input values for the i-th and j-th samples, and λ is the mixing ratio sampled from the beta distribution. Mixup was chosen as the subject of this study because of its versatility and because it can be used for many numerical data, including not only images but also time series data, and therefore, the impact of improving the Mixup method would be significant.

An improved version of mixup so that it can also be performed in a hidden layer of a neural network is called manifold mixup [16], but both mixups generate samples only in a localized region of the data distribution, on a line segment between two points, and are inappropriate for data sets with distributions in which the properties of the points on that line segment vary nonlinearly.

The Feature Combination Mixup (FC-mixup) proposed in this study is a method of mixing samples in a different way from conventional mixups, and is outlined in Figure 3.4. Suppose that two samples A and B in the same minibatch output the features Z_{A} and Z_{B} in a randomly

selected layer. d is the total number of features in that layer, FC-mixup randomly extracts and combines $d\lambda$ features from Z_{A} and $d(1-\lambda)$ from Z_{B} and generates a new sample Z_{X}. Since the number of possible combinations is large for a single value of λ, different data can be generated depending on the random number, and thus samples can be generated over a wide range of the data distribution. FC-mixup is expressed as follows, so Z_{A} and Z_{B} are mixed so that this equation is satisfied.

$$|Z_{\mathrm{A}} \cap Z_{\mathrm{X}}| = \mathrm{d}\lambda$$

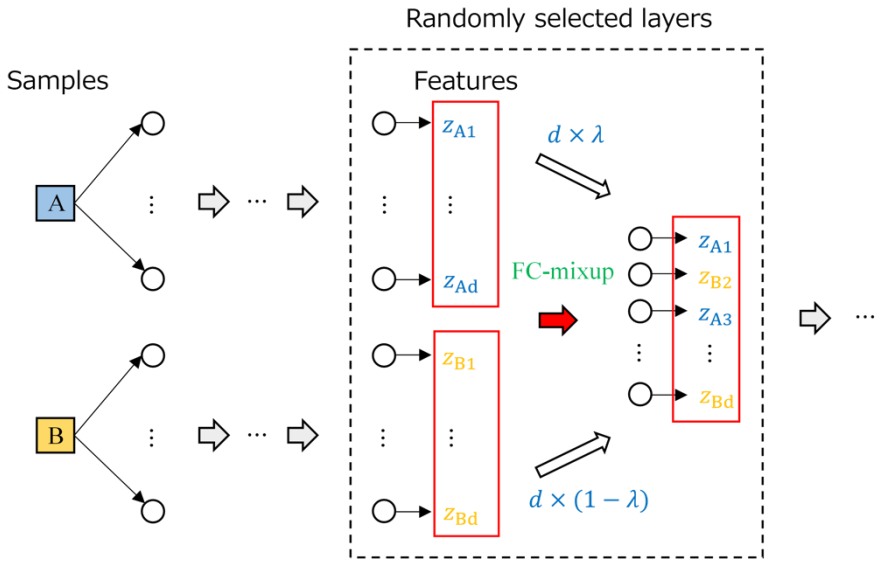


Figure 3.4 Overview of FC-mixup

This technique of generating new data by combining the parts of two data sets is also found in Puzzle Mix [17], but the target is limited to the input image. A similar technique is used in Adversarial mixup resynthesis [18], but it is limited to use in autoencoders, while FC-mixup is designed for more general use. To increase the diversity of the generated data, a method that simultaneously uses FC-mixup and Manifold mixup [16] is referred to here as the Hybrid method.

In our experiments, we used several multi-class classification datasets to compare the classification accuracy of the test data between the conventional method (no data augmentation, mixup at the input layer [15], Manifold Mixup [16]) and the proposed method (FC-mixup, Hybrid method). MNIST, CIFAR-10, CIFAR-100, SVHN, and TinyImageNet were used for the data. Models used were a multilayer perceptron (MLP) with one intermediate layer, a small convolutional neural network (CNN), ResNet18, and ResNet50. In addition to the full-size data, experiments were conducted on reduced data with 1,000 randomly selected samples. Means and standard deviations in five trials with different initial values were obtained and compared.

The results in Table 3.1 show that in most cases the proposed method gives the highest accuracy. Overall, FC-mixup tended to give better performance than the Hybrid method. It can be said that the results of the present study are promising results, indicating that using the FC-

mixup and Hybrid methods is likely to improve the quality. Detailed analysis of the compatibility between data augmentation methods focusing on diversity will be the subject of future work.

Table 3.1 Comparison of test accuracy on multi-class classification data

	MNIST MLP	CIFAR-10 SMALL CNN	CIFAR-10 RESNET18
DEFAULT	98.44 ±0.010	86.76 ±0.34	87.87 ±0.27
INPUT	98.41 ±0.066	87.03 ±0.30	88.47 ±0.35
MANIFOLD	98.55 ±0.044	87.22 ±0.32	88.50 ±0.37
FC	98.70 ±0.050	87.49 ±0.28	88.76 ±0.19
HYBRID	98.62 ±0.020	87.40 ±0.33	88.49 ±0.16
	SVHN RESNET18	CIFAR-100 RESNET50	TINYIMAGENET RESNET50
DEFAULT	93.79 ±1.92	54.97 ±1.58	65.38 ±0.26
INPUT	95.79 ±0.12	60.37 ±1.61	68.27 ±0.60
MANIFOLD	95.73 ±0.08	61.94 ±1.98	68.22 ±0.83
FC	95.71 ±0.12	63.45 ±1.82	68.04 ±0.49
HYBRID	95.68 ±0.16	62.48 ±3.40	69.04 ±0.49
	MNIST (1000) SMALL CNN	CIFAR-10 (1000) SMALL CNN	SVHN (1000) SMALL CNN
DEFAULT	96.24 ±0.12	56.59 ±0.72	68.34 ±1.06
INPUT	96.15 ±0.21	58.44 ±0.86	69.49 ±1.10
MANIFOLD	96.30 ±0.19	56.78 ±0.81	68.51 ±1.36
FC	96.86 ±0.10	59.73 ±0.90	73.82 ±0.59
HYBRID	96.63 ±0.19	58.22 ±0.66	70.47 ±0.64

4 Debug-Testing of DNN Software

In the initial development stage of Deep Neural Network software (DNN software), we ensure that the required functions and prediction performance are achieved through iterative trial-and-error processes, in which three viewpoints (elaborating and refining requirements, preparing datasets for training, and selecting appropriate machine learning models) are considered. This trial-and-error process corresponds to debugging in conventional program development. In the case of DNN software, the debugging activities involve generating datasets for debug-testing, monitoring the training and learning status, and identifying and removing root causes that hinder the fulfilment of requirements. In the following, we will report on a debug-testing method investigated in FY2020, discuss the experimental results obtained, and summarize our future plans.

4.1 Direct cause of failure

A standard method of supervised DNN learning involves two types of programs: training (or learning), and prediction (or inference). When training data is given and a learning task to achieve is made clear, a learning model for the target DNN software is selected, and some design decisions on the method used in the training and learning process is fixed. If we use available open-source machine learning frameworks, we may set up several parameters of the framework. The next step is to construct training dataset. Then, we run the training/learning program (possibly provided by the machine learning framework) with the training model and training dataset as input, and derive a trained DNN model as a computation result. More precisely, the training/learning program searches for a set of weight parameter values that define the trained DNN model uniquely. This trained DNN model defines behavior of the prediction/inference program.

From a user's point of view, a prediction/inference program is the entity to use. In the case of a classification learning task, for example, the program calculates certainty levels of probabilities of classification results for an input data. By examining the output results, we can determine whether the DNN software works as intended. When the program does not produce results as expected, we localize possible fault locations and remove them. In other words, we conduct debugging.

A failure may be occurred due to a flaw somewhere in the information used in the execution process of the training/learning program, either in the training dataset, the training model, the learning mechanism, or their combinations. However, direct causes of failure in prediction/inference results are attributed to the trained DNN model or set of obtained weight parameter values. While a root cause of failure is somewhere and often not known, the failure is attributed to a defect in the weight parameter values or the trained DNN model. Thus, from users' point of view, a certain distortion of the trained DNN model seems a direct cause of the failure [19]. A method to measure such distortion degrees is needed regardless of the root causes.

In this chapter, we investigate whether we can detect faults in DNN software with an internal metric to measure such distortion degrees of trained DNN models. The weight parameter values in the DNN models are the output of the training/learning program, but there is no direct way to check its validity, because those expected weight parameter values cannot be known in advance. If such expected parameter values were known, training/learning could be skipped. We can just use those known values, as embedded in a trained DNN model, to implement a prediction/ inference program.

4.2 Internal indices

This section first introduces the notion of neuron coverage (NC). We consider a learning model as a network of neurons. Given a threshold, neurons whose output values exceed the threshold are said to be activated. When the number of neurons constituting the learning model is N and the number of activated neurons is A, the neuron coverage is defined as the ratio of active neurons is (NC = A/N). In [20], NC is assumed to be criteria for test coverages of trained DNN models; the research work investigates how the choice of input data for evaluation affects NC values.

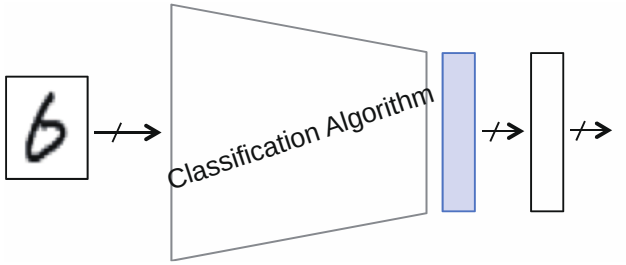


Figure 4.1 Trained DNN model.

In this chapter, NC is assumed to be used as an internal index [21] to represent distortion degrees by appropriately choosing the target neurons to be considered. Figure 4.1 shows a schematic diagram of the trained DNN model. NCs are defined for the neurons in the final stage of the middle layer (or the penultimate layer as shaded gray), but not for all the neurons in the trained DNN model as in [20].

In general, in machine learning techniques, this penultimate layer is often considered to hold meaningful information. For example, in the case of an image classification task, the early stages of the model is responsible for the correlation analysis (analysis of patterns of pixel values), which plays a specific role in algorithms such as image recognition, and their calculation results are summarized in the penultimate layer. In this chapter, we assume that direct causes of defects are manifested in this internal layer. Furthermore, various statistical indices can be derived based on NC values of this layer. We will investigate, through experiments, what derived index is appropriate depending on test objectives to be investigated.

4.3 Experiments: method and results

We present the results of several experiments and discuss the usefulness of the internal or derived indices mentioned in the previous section. First, we show the results of comparative experiments when a training/learning program (or a learning framework) has faults in it. In the following, BI is the training/learning program which is a bug-injected version of a probably correct program PC.

Figure 4.2 depicts the accuracy (the percentage of reconstructed correct answers) for a test dataset. In the experiments, a classical fully-connected network is chosen as the learning model, and different number of neurons are placed in the middle layer, which implies that each model is of different structural capacity. When we have a sufficient number of neurons (50 on the horizontal axis), there is no significant difference in the accuracy between PC and BI. Thus, it is difficult to distinguish between the PC and BI solely by examining their accuracy values, and thus the presence or absence of a defect cannot be identified. In addition to this finding (Figure 4.2), the results of an experiment to systematically investigate the situation further (Figure 4.3) are presented below.

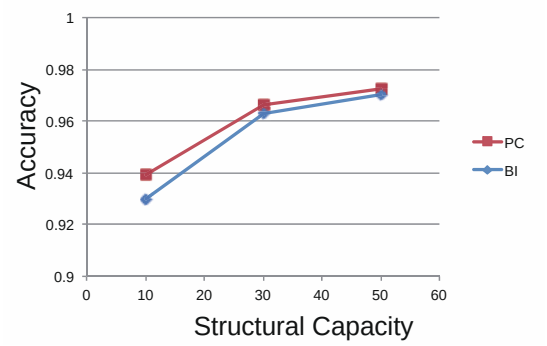


Figure 4.2 Learning models of different capacities.

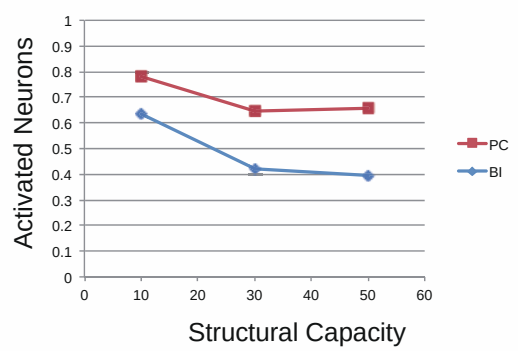


Figure 4.3 Relationship with internal indices

Figure 4.3 plots values of the internal index (activated neurons or neuron coverage) on the vertical axis. Their absolute values, for example, of 10 for BI and 30 for PC are both around 0.7, making it impossible to distinguish between BI and PC if we do not take into account the structural capacity. The indices are not usable to examine the activated states of neurons.

Therefore, we will study if there is an appropriate indicator to be derived from the internal index of NC. As a set of data (a sample), in the test dataset, leads to a collection of neuron coverages, we can obtain some statistics from the sample such as the mean μ and variance σ^2, and calculate σ / μ. Figure 4.4 shows the case where this derived index σ / μ is used on the horizontal axis. From the values on the vertical axis, we can find out which leaning model has which value by referring to Figure 4.3.

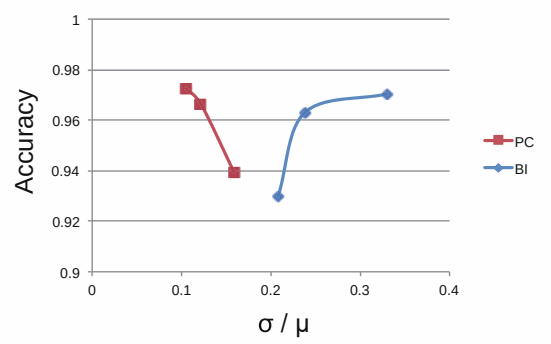


Figure 4.4 Derived index

Figure 4.4 shows that we can distinguish between the PC and BI. Although the internal index cannot distinguish between the PC and BI with different capacities (Figure 4.3), a derived index of σ / μ can discriminate between the PC and BI. We can see that the neuron coverage basically contains a piece of useful information.

Next, Figure 4.5 is a scatter plot of classification probability using corrupted data for the evaluation; the horizontal axis refers to the classification output by the BI and the vertical one by the PC.

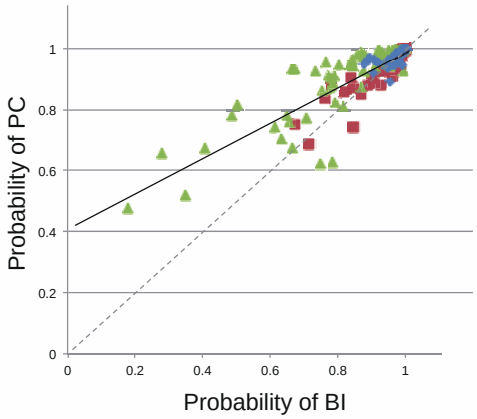


Figure 4.5 Classification certainty for corrupted data.

In Figure 4.5, a △ represents an output value for corrupted data, which is supposed to be distributed on the dotted line passing through the origin, if we assume that the PC and BI output the same value for the same data. In fact, it can be seen that □ selected from the test dataset

(without any corruption) mostly arranged on the dotted line. On the other hand, corrupted data (△) are distributed along the solid line, indicating that the PC is a better classification certainty than the BI. It implies that the BI, containing bugs in it, is less robust, although the accuracy remains the same as that of PC (Figure 4.2).

The following experiment confirms that differences in robustness can be detected by using an internal index (Figure 4.6).

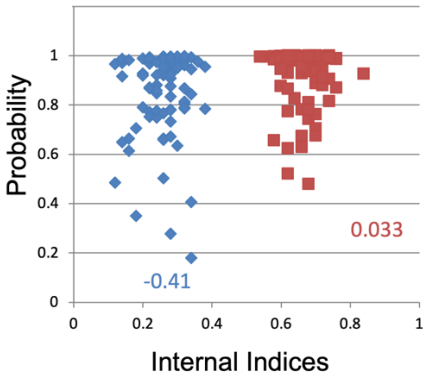


Figure 4.6 Differences in internal indices

The corrupted data described above were input, and the internal index for each input was plotted on the horizontal axis. The □ distributed in a group on the right side shows the results of PC, and the ◇ distributed in a group on the left side shows the results of BI. The scatter plot shows that (1) the value of the internal index of PC is large, and (2) the correlation between the internal index and prediction probability (certainty of classification) is negligible (0.033). Next, we calculate σ / μ, which is 0.0876 for PC and 0.2183 for BI. Figure 4.6 shows results that corrupted data affect the robustness, and that the value of σ / μ is considered to have correlations with the robustness.

Next, we conducted experiments to investigate how distorted training data affect the trained DNN model. We plotted the accuracy for a test dataset common to all the cases. Thus, differences in the vertical axis indicate a certain difference (distortion degree) in the training dataset used for obtaining the trained DNN model (Figure 4.7).

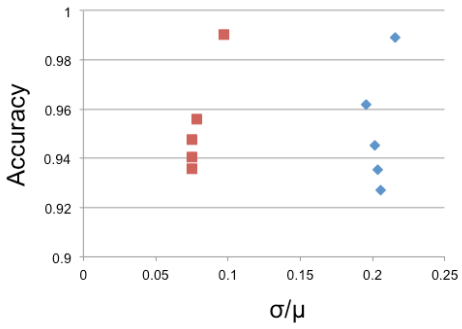


Figure 4.7 Differences in training datasets.

Figure 4.7 shows the two independent series for the PC (□) and BI (◇). From top to bottom in a series of each measured points (from better to worse accuracy), a training dataset with a larger distortion is used. Because the test dataset is common, the data shift of the test data is relatively larger as the distortion degrees in the training data is larger. Furthermore, the accuracy decreases as the shift becomes large. Figure 4.7 also shows that the value of the horizontal axis (σ / μ) is clearly different between the PC (□) and BI (◇). It can be confirmed that the accuracy and the robustness suggested by the σ / μ values are two independent perspectives.

From the above (Figure 4.7), the distortion in training dataset can be examined by the method based on the accuracy. As is done in practice, the method based on the accuracy is useful when checking the training dataset quality. On the other hand, if there is a possibility that other factors such as faults in a training/learning program are involved (multiple defects are assumed), it is desirable to examine the values of the internal and derived indices (σ / μ) at the same time.

4.4 Related work

Neuron coverage (NC) is a simple quantitative measure introduced in DeepXplore [20] as a test coverage metric. In conventional software testing, test coverage is defined in terms of the basic block of program codes, which is the statements executed by a given test input data. A program is represented as a Control Flow Graph (CFG) whose nodes refer to executable statements. In the simplest case, the criterion is whether or not a node in the CFG is contained in an execution path induced by an input test, i.e., whether or not the statements are executed. As a DNN model is represented as a network, a kind of graphs, metrics similar to those for CFG can be introduced. The neuron coverage concerns whether neurons located at nodes are activated (output values of these neurons exceed a specified threshold), which is comparable to the C0 criterion defined on the CFG. DeepXplore assumes that high NC values refer to the situations where high percentage of neurons are exercised by input data, and discusses how to generate new test input data to increase the NC values.

Neuron coverage would be a straightforward idea analogous to the conventional test coverage criteria. Later, satisfying the criteria, to achieve 100% in terms of NC, is found empirically not difficult. New metrics are proposed to take into account correlations among multiple neurons or those in different layers [22], which may be comparable to more elaborated coverage metrics, such as C1 or the others, in conventional software testing.

The original NC is simple and easy to use as a metric to guide or control automated test generation processes. Usually, a classical data augmentation method picks up a seed data, from which a series of new data is to be generated by pre-defined data transformation algorithms. New test data are successively generated until the accumulated NC values is saturated. When reached the situation where no increase in the NC is seen, the generation method switches a seed data to new one and continue the process [23]. The classical data augmentation method can be

replaced by other approaches such as test input generation based on GAN [24]. Test generation method using GAN with a help of NC is reported in [25]. Although it is a simple metric, NC is now considered as a practical criterion to control the automated test generation process (coverage-guided test generations).

Some of early works on testing pre-trained DNN models adapt application-specific properties as software test oracles; the DNN models for regression tasks in the auto-pilot car application [23][24] use the calculated steering angle as the oracle. There is also a research work [26] to investigate whether test inputs to increase the NC values are useful for detecting faults. The usefulness of NC is dependent on what are considered failures. The work [26] also indicates that the correlation between NC and external indices such as the accuracy is weak. In this chapter, based on this observation that the correlation between the two is weak, an internal index based on the NC is used for the test, which is not contradictory to the discussion in [26], but rather in the same direction. Note that the test coverage is a criterion for terminating testing, while detecting faults depends on whether the test input data executes corner cases. These two notions, the test coverage and corner cases, refer to different aspects. In fact, it has been reported that the enhancement of coverage does not necessarily leads to the improvement of the efficiency of fault detection in conventional software testing. The same findings would be applicable to cases of DNN testing.

In this chapter, we use the NC value as a simple test index, from which a sort of distortion degrees in trained DNN model is derived [19][21]. Our approach is based on a view that faults in DNN models appear as inappropriate NC values, whereas existing works use NC as a criterion for the test coverage. In our experiments, we were able to examine the reliability of the training and learning programs and the robustness of the trained DNN models. These are two primary concerns in debug-testing.

4.5 Conclusion

In this chapter, we used an internal index based on the neuron coverage (NC) defined on the penultimate layer for representing a sort of distortion degrees in trained DNN model. The NC is a scalar and easy to measure, and thus can be used as a light-weight test index. It, however, discards the information about the individual activated neurons, and thus lacks useful information. In fact, Kim et al. [27] proposes a method to estimate the distribution of activated neuron and to discuss the usefulness of input data for testing. Distribution on such neuron values may be considered to have rich information. In future, we will study how to debug training dataset by making use of such distribution information.

5 Debugging and Testing of Training Data

5.1 Three Problem Settings

In early stages of software development, in which programs are constructed to employ Deep Neural Networks (DNNs) [28], debugging and testing is performed to ensure that the core DNN components behave as expected. This is the process of feeding appropriate data to the DNN components and checking whether the predicted output is exactly what is expected. If the output is faulty in some ways, the DNN component under test contains a defect. The purpose of debugging is to identify and remove such unknown defects.

Defects in DNN components are the direct cause, but not the root cause, of failures. In the standard method for building DNN components [29], three distinctive components are basically involved: (a) the machine learning infrastructure, (b) the training model (a template of the DNN model), and (c) the training data. The root cause is one of them or their certain combinations leading to the failure that the DNN component exhibits. The problem setting of the inspection differs depending on where the root cause is assumed [30].

The basis of DNN component construction is to make use of a training dataset consisting of a huge number of training data and derive the information inherent in those data by means of statistical methods so as to obtain a DNN model (a nonlinear function) inductively. In a naive way, we may examine the DNN model to identify root causes. However, since the DNN model is a nonlinear function to exhibit some functional behavior, the software testing method using indirect test oracles is often employed; we feed evaluation data to the DNN model and check whether output results are valid or not [31].

In the case (a) above, the core of the machine learning infrastructure is a numerical program that solves an optimization problem, and the metamorphic testing method is known to be useful [32]. In the case (b), the learning model is not obviously flawed. It is to find an optimal or sub-optimal learning model for the target machine learning task, which has been, in a sense, one of the main challenges of the DNN technology [28]. In this chapter, we discuss the case (c), i.e., debugging and testing methods of training data.

5.2 Debugging Problems of Training Data

Debugging and testing of training data is to revise (add or delete) the training data so as to obtain a DNN model that exhibits the intended functional behavior. This view is based on the observation that the bias of the training data affects much the trained DNN model. In the following, we specifically consider the debugging problem of training data for supervised machine learning classification tasks.

5.2.1 **Model Accuracy and Model Robustness**

In the supervised task of classifying input data into C categories, a datapoint z is a tuple ($z = \langle x, y \rangle$) consisting of two types information, a multidimensional vector x and its correct answer tag (or simply a label) y (see Figure 5.1). The DNN model, derived from a given training dataset S ($S = \{z^{(k)} \mid k = 1, \ldots, N\}$), is inspected against input evaluation data x. Its output is a C dimensional classification probability vector P_x corresponding to the data x. If $P_x[j]$ (the j-th component of P_x), the component with the largest value j, is equal to y ($y = \mathrm{argmax}_{(j \in [1,C])} P_x[j]$), then the DNN model is considered to return a correct answer. In this case, the multidimensional vector P_x, in particular, the probability of the j-th component $P_x[j]$, is one of the good indicators of the model accuracy for the data x. For a collection of evaluation data E ($E = \{\langle x^{(\ell)}, y^{(\ell)} \rangle \mid \ell = 1, \ldots, M\}$), Accuracy is the number of correct answers (percentage of correct answers) for the collection. In addition, the variability of the probabilities of the classification categories (sometimes referred to as Gini Impurity) is an indicator of the model accuracy as well.

The accuracy for the training dataset S and the one for the other dataset E, different dataset from S, are compared. While the accuracy for S is good, the accuracy is sometimes worse for E. This phenomenon is known as overfitting to the training dataset. Usually, both S and E are constructed from one large data pool D, and are considered as different samples following the same data distribution; E in this case is sometimes called a testing dataset as compared with the training dataset of S. When there is no overfitting where the accuracies are not much different each other, the DNN model is considered to exhibit good generalization performance.

In the training data debugging problem, the evaluation data E may be selected from a dataset other than D. For example, in positive testing, where the goal is to confirm that the system exhibits the expected behavior, as in the evaluation of generalization performance, we can choose E from D, in which E is different from S. However, to test the behavior in exceptional situations, we may choose a dataset F for the evaluation that is not included in D. Model accuracy, measured with the percentage of correct answers, is not a good indicator for F. The evaluation criterion is model robustness, which expresses how the prediction probability is decreased depending on how much a data in F is deviate from data in D or S.

In the development in practice, if the expected prediction performance is not achieved for a given D, new data is collected and the training data itself is revised. Then, the DNN components are derived using the new training dataset, namely in an iterative manner. Moreover, during testing, we evaluate the model accuracy and model robustness in view of both positive and exceptional testing.

5.2.2 **Memorization of Training Data**

Overfitting or overlearning significantly affects the prediction performance (the model

accuracy and model robustness) of DNN models. Therefore, basic machine learning methods have been studied extensively to mitigate those problems; the study includes regularization or dropout [33]. In spite that such methods are adopted, the expected prediction performance cannot be obtained if the training dataset is inadequately biased. The debugging problem of training data is to improve the prediction performance of DNN models by revising the training dataset. Simply, it is to eliminate the inappropriate bias. However, it is difficult to evaluate the degree of bias as well as the appropriateness or inappropriateness of the bias.

One traditional approach to evaluate the bias of the training data (sample) is to examine statistical characteristics of the sample. For example, given that $S = \{\langle x^{(k)}, y^{(k)} \rangle \mid k = 1, \ldots, N\}$, let $S^C = \{\langle x, c \rangle \mid \langle x, c \rangle \in S \text{ and } c = 1, \ldots, C\}$ where c is a correct answer tag. If the sizes of S^C are equal in size, then we may say that there is no bias among S^C from the viewpoint of the correct answer tag. However, each S^C follows some data distribution ρ^C and we don't know whether S^C is sampled faithfully in regard to ρ^C. To check this, we need to know ρ^C, however, the data x is multidimensional, and such a multidimensional data distribution is not easy to estimate.

Alternatively, the prediction performance of DNN models is investigated by testing results with input evaluation data. DNN models derived from the same training data may exhibit different prediction performance, depending on the method of the machine learning. In other words, it is not enough to examine the statistical characteristics of the training data for the purpose of debugging the training data, but it is also necessary to consider how the bias of the training data is reflected in the trained DNN model.

The relationship between DNN models and training data bias can be discussed in terms of the DNN models remembering the labels of the training data. Now, when the training data S contains a datapoint $\langle a, t \rangle$ ($\langle a, t \rangle \in S$), we can construct S' so that the $\langle a, t \rangle$ is removed from the training data S ($S' = S \setminus \{\langle a, t \rangle\}$). Let each DNN model obtained by training with either S or S' be M or M' respectively. Then, the result, P_a for M or P'_a for M', is obtained for the common input data a. If the classification result t for $P_a[t]$ is very likely and $P'_a[t]$ is less likely, then M is said to *memorize* the datapoint $\langle a, t \rangle$ used as one the training data. From this definition, we can see that the DNN model memorizes the training data in the overfitting situation, where $P_a[t]$ is apparently more likely than $P'_a[t]$.

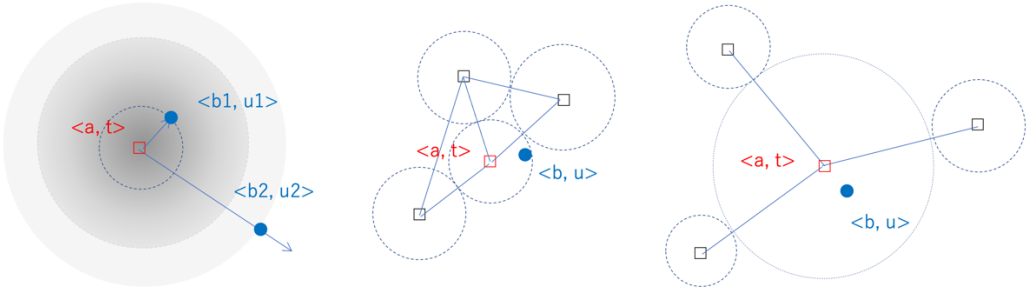
For DNN models, it is known that the Membership Inference is possible. The problem is to find out if a datapoint $\langle x, y \rangle$ ($\langle x, y \rangle \in D$) was included in the training dataset ($\langle x, y \rangle \in S$) just from the information obtained by feeding data to the trained model, $M^S(x)$. Black box methods make use of the classification probability vector P_x [34], or white-box methods use the information of the loss function $\ell(Y(W; x), y)$ calculated in the process of executing $M^S(x)$ [35], where W is the training parameter or weight and $Y(W; x)$ is the internal representation of the prediction for the input x.

Intuitively, Membership Inference method is based on the observation that the distribution of P_x or $\ell(Y(W; x), y)$ is different depending on whether the datapoint $\langle x, y \rangle$ is included in the training dataset S or not. Furthermore, these differences in the distributions are somehow attributed to the memorization of training data including overfitting cases [35]. Thus, the

approach to mitigate the threats of Membership Inference is to remove those data, that are memorized easily, from the training dataset, in addition to employing a machine learning method that avoids overfitting [36].

We now examine the situation involved with the memorization of training data. Consider a classification problem as in Figure 5.1; we assume $a \neq b$ whereas $t = u$. Figure 5.1 (a) illustrates a situation where the prediction probability of $\langle b, u \rangle$, a training data moved away from $\langle a, t \rangle$, decreases as the distance between them becomes large. Figure 5.1 (b) shows that removing that datapoint $\langle a, t \rangle$ from S does not significantly affect the prediction probability of the data $\langle b, u \rangle$ when the training data are dense in S. In other words, the removed training data is not memorized in that it does not significantly affect the prediction results. Figure 5.1 (c) represents a situation where the training data are sparse. Contrary to Figure 5.1 (b), it represents that the influence becomes large and is firmly remembered. Such outlier data significantly affects the predictive classification performance of the DNN model.

Finally, we consider the Membership Inference viewed from the training data debugging problem. In the situation where training data are memorized, the distribution of either P_x or $\ell(Y(W; x), y)$ is very different depending on whether the datapoint is included in the training dataset S or not. The Membership Inference method makes use of the fact that the predictive performance for z', far from training z datapoints, is poor. In other words, we can think of the Membership Inference as a test of model robustness; the phenomenon of training data memorization is related to model robustness.



(a) Predicted probability in the neighborhood (b) Dense region (c) Sparse region

Figure 5.1 Training data placement and prediction certainty.

Here, we refer to the schematic situation in Figure 5.1. Removing the dense data shown in Figure 5.1 (b) would have little impact on the model accuracy. On the other hand, removing the data in a sparse region as shown in Figure 5.1 (c) would improve model robustness, but would reduce model accuracy in the neighborhood because there would no longer be data to support their predictive classification results. Alternatively, adding new data in the neighborhood without removing this datapoint will make the region dense and improve the local model accuracy. Therefore, detecting outliers in the training data set S is important for debugging dataset.

Figure 5.1 schematically illustrates that the predictive classification performance of the input

data is affected by the location relationship with the training data. However, it does not say how the location relationship is defined, i.e., from what aspects of the data, the location relationship is defined. Conversely, now the question is how the location relation should be defined when discussing the difference in prediction classification performance; the outlier detection problem will become clear when such criteria are precisely defined.

5.3 Outliers and Neuron Coverage

We consider outlier detection methods for the purpose of training data debugging.

5.3.1 Outliers in Training Data

The debugging problem of training data is to find out outliers in the training dataset and to decide how to deal with the outliers according to the purpose of the DNN model under development. How we handle the outliers is related to the requirements specification of the DNN model. Thus, the general discussion of training data debugging may be limited within establishing a technique for outlier detection.

In general, outliers are data that have different characteristics from the data that make up the majority, and whether or not they are outliers is defined based on the data distribution (statistical data model) that the collection of target data exhibits [37]. For example, if the probability density function of the data distribution is known, then we can check whether the data are outlier or not based on the likelihood of the data.

In a naive way, we consider whether it is an outlier or not based on the empirical distribution of the training data. However, the training data is a multi-dimensional vector, and it is difficult to know the empirical distribution in a compact form. For example, it is difficult to apply methods such as Kernel Density Estimation, and as a result, the outlier detection method based on likelihood is not practical. Alternatively, analysis methods similar to Combination Testing, which is known in the field of software testing, may be applied. By selecting components (features) that are considered having a large impact on the empirical distribution and focusing on such representative dimensions, we may conduct analysis as an approximate of the case on the whole empirical distribution. While practically applicable, outliers are rare by definition, and the effectiveness of this approximate method is questionable.

For a slight change of perspective, the robustness radius of the standard method of analyzing model robustness [38] is considered. For two datapoints $\langle x, y \rangle$ and $\langle x', y' \rangle$ and the predictive classification results for each of the outputs $P_x[y]$ and $P_{x'}[y']$, let the robust radius δ be the tolerance level ε of the difference between the outputs; for a given ε, the robust radius is the maximum difference of input data that satisfies δ ($|P_x[y] - P_{x'}[y']| \leq \varepsilon$ when $|x - x'|_p \leq \delta$). Here, we define the difference of input data in terms of L_p-norm. In a naive way, for a given ε for given input data, we consider that the model robustness is good if the robustness radius δ is large. However, the calculated radius δ_p is dependent on the choice of the norm L_p. While

the definition of model robustness by the robust radius is strict, the analysis in the space of input data requires further discussion or interpretation of whether the norm used is appropriate or not, which complicates the problem.

We consider now how the training data $\langle a, t\rangle$ affect the prediction results of the other data $\langle b2, u2\rangle$ that are classified to different classification categories ($t \neq u2$) (see Figure 5.1 (a)). The two datapoints have different classification categories and can be assumed to be far apart in the input data space. We assume that the training dataset S contains $\langle a, t\rangle$ and let S' be the one to be removed $\langle a, t\rangle$ from S. Further, let the DNN models obtained from S and S' be M and M' respectively, and let the predictive classification results for the data $\langle b2, u2\rangle$ be P_{b2} and P'_{b2}. With the method of Influence Functions, which analyzes how S and S' affect the error function, we are able to know that there exists $\langle b2, u2\rangle$ such that the values of $P_{b2}[u2]$ and $P'_{b2}[u2]$ are different [39]. It shows that the presence or absence of the training datapoint $\langle a, t\rangle$ affects the classification probability of $\langle b2, u2\rangle$. Therefore, it is difficult to obtain the desired information by analyzing the differences in the input data space ($a \neq b2$) .

From the above, we can see that it is difficult to systematically detect the desired outliers by analyzing a collection of training data in the input data space. The reason for this is that model accuracy and model robustness are affected not only by the training data but also by various factors involved in the training process, such as the machine learning method. However, we do not claim that the analysis in the input data space is completely ineffective. Such an analysis would give us a vague idea of the empirical distribution of the training data.

In this chapter, we think that even if the features of the input data space are related to model accuracy and model robustness, they are not appropriate as a systematic training data debugging method. We will study systematic methods for detecting outliers in training data.

5.3.2 **Active Neurons**

Neuron coverage is defined as the ratio of active neurons to the number of target neurons considered [40]. $M^S(x)$ denotes the situation where the input signal (of x) propagates through the DNN model and activates each neuron. When the output of a particular neuron exceeds a given threshold, we call it active, an active neuron.

Neuron coverage was initially proposed as a coverage criterion for coverage-driven test data generation [40]. The active neurons for the input data x provide a useful information in that they influence the output results. On the other hand, the neurons not involved in the predictive inference process, are considered to be inactive. The input data that produce inactive neurons do not effectively test all the neurons, and then new input test data are needed so that they further activate the inactive neurons. When a set of input data makes all the neurons active, the set of test data are considered to reach 100% of the coverage.

After the original proposal in [40], there have been several research works to study the practical usefulness of the neuron coverage as a test coverage criterion [41][42][43]. In particular, it has been recognized that 100% of the neuron coverage is not difficult to achieve and thus is weak as a test coverage criterion, which is similar to the case of the C0 criterion in

conventional software testing methods.

On the other hand, we may consider that the training was appropriate in the first place, producing inactive neurons not involved in the predictive inference process. In this case, we can add new input data to the training data and conduct re-training [40]. This suggests the idea of using the neuron coverage as a criterion for evaluating the quality of the model M^S. The following is a discussion from the viewpoint of the neuron coverage as a model quality evaluation criterion [44].

In DNN models M^S for classification tasks, the upstream layers near the input perform encoding E' (Encoding), and is followed by classifying C'. Classifying is done after the encoding ($M^S = C' \circ E'$); $M^S(x) = (C' \circ E')(x) = C'\big(E'(x)\big)$. When the output is a classification probability vector, we place softmax functions in the final layer (logits) of the output; $M^S = \mathrm{SOFTMAX} \circ C \circ E'$. Next, we may place a layer of Fully Connected Network (FCN) between E' and C; $M^S = \mathrm{SOFTMAX} \circ C \circ \mathrm{FCN} \circ E$.

In FCN, a neuron in a layer considered is connected to all the neurons in the next layer, thus the output is swap-invariant, which means that the output is preserved when the neurons are exchanged within the same layer. Therefore, the neuron coverage may be useful to summarize the neuron activity in FCN layers. On the other hand, when the constituent neurons play a specific functional role, such as in SOFTMAX or CNN, it is questionable whether the neuron coverage, which considers all neurons equally, provides useful information. In fact, two different definitions are studied for CNNs, and depending on which one is adopted, the value of neuron coverage is different [45]. In this chapter, we consider neuron coverage for the FCN layer.

A series of experiments are conducted [45] in which training data are systematically generated by means of a classical data augmentation method and the effects on neuron coverage are investigated. The results showed that the difference in training data had influenced the neuron coverages at the E' layer, while only a small effect was made on the C layers. In addition, although the testing data are changed, very small differences are observed on the last layer in C (Penultimate Layer of the whole model). It implies that the differences in the training data are reflected in the FCN layer where $E' = \mathrm{FCN} \circ E$ as introduced early.

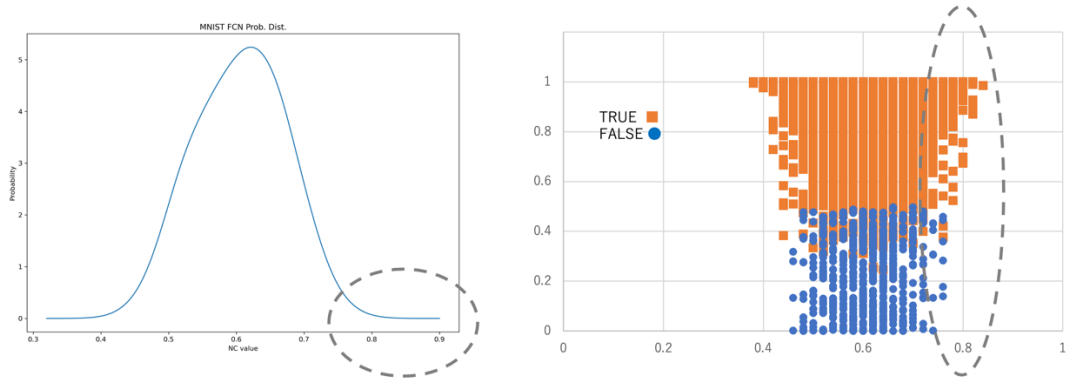
In addition, in previous experiments [32][44][46] in which we have measured the neuron coverage on the FCN located as the last layer of C, we observed little correlation between the classification prediction probability and neuronal coverage. Therefore, the neuron coverage may be considered to represent an aspect independent of the information contributing to the model accuracy. If it is found to be correlated with the model robustness, we can expect that the neuronal coverage on a particular layer is useful as a method to detect outliers for our purposes.

5.4 Experiments and Discussions

In our experiments, we used the machine learning model such that $M^S = \mathrm{SOFTMAX} \circ C \circ \mathrm{FCN} \circ E$, and the MNIST dataset. In particular, the training dataset S was entered as the evaluation input, and the neuron coverage at the FCN layer was measured, whose results are

shown in Figure 5.2.

Figure 5.2 (a) shows the frequency distribution of the neuron coverage for the input data (the result of KDE). Figure 5.2 (b) is a scatter plot of the neuron coverage of the input data on the horizontal axis and the prediction probability of the same input data plotted on the vertical axis. The red dots represent the data with correct predictions and the blue dots represent the incorrect one. The correct prediction rate of the training data and the test data were both 99%.



(a) Neuron coverage vs data frequency (b) Neuron coverage vs prediction probability

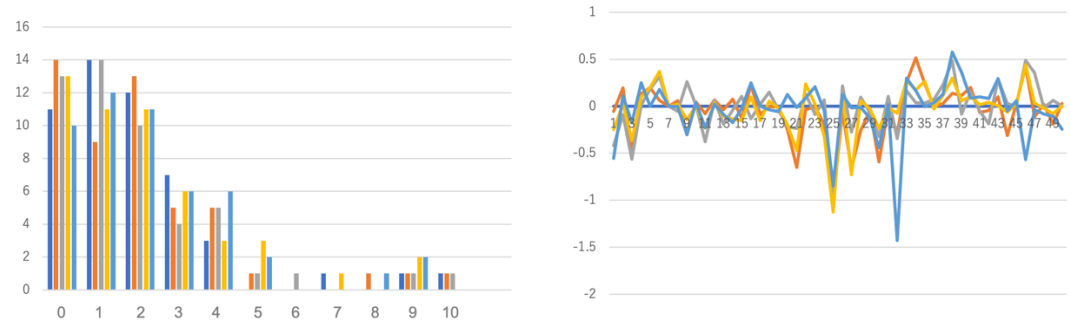
Figure 5.2 Analysis results of training data

Figure 5.2 (a) shows that the neuron coverage in the FCN layer is distributed between 0.38 and 0.84, and the median and mean values are 0.60. In Figure 5.2 (b), we confirmed again that there is almost no correlation between the neuronal coverage and the predicted classification probability. While there is a large difference in neuron coverage (between 0.38 and 0.84) , it can be considered that the magnitude of the contribution to prediction classification differs regardless of whether the answer is correct or incorrect. In other words, the neuron coverage takes a large value even when the contribution to being incorrect may be large.

In Figure 5.2, the area circled by the ellipse, for example, represents the training data with larger-than-average neuron coverage, resulting in a significant impact on the output, leading to the observed classification probabilities. Therefore, it can be considered to be a faithful representation of the characteristics (classification probabilities) of the target data. Suppose that we chose training data with smaller-than-average neuron coverage. According to Figure 5.2 (b), the predicted classification probabilities of the outputs are scattered, which is similar to the case for the elliptical regions in Figure 5.2 (b), while it is not certain that the small neuron coverage had an appropriate impact on the output. In other words, those training data may be considered not to play significant roles to obtain M^S. In this chapter, we consider the training data that has small neuron coverage, regardless of the predicted classification probability, to be an outlier.

We now sort the training data based on the neuron coverage and create training datasets $S^{(K)}$ of the same size. Then, we use $S^{(K)}$ to derive a trained training model $M^{(K)}$, and evaluate $M^{(K)}$ by means of a common dataset for the evaluation. In this experiment, the entire

sorted training data was studied, and no adjustments were made at the level of individual data.



(a) Median output value vs. test data frequency (b) Neuron vs. magnitude of displacement

Figure 5.3 Neuron activity vector

Figure 5.3 shows the results of the five trained models $M^{(K)}$ (K = 1, ..., 5), in the form of the neuron activity vectors of the FCN layer, where the test dataset was used for the evaluation. The neuron activity vector represents an internal feature, which is a multidimensional vector that consists of the output values of the constituent neurons.

The training datasets $S^{(K)}$ (K = 1, ..., 5) are those obtained by random selection, right-side editing (replacing the training data enclosed in the ellipse in the Figure 5.2), left-side editing, both-sides editing, random selection, and data augmentation after random selection. These correspond to the bar graph from left to right in Figure 5.3 (a). The accuracies are 97.14%, 96.89%, 96.90%, 96.94%, and 95.81%; there is no significant difference except for the $M^{(5)}$. The fact that there is no difference from $M^{(1)}$ to $M^{(4)}$ is consistent with the way that the training data sorting method (see Figure 5.2 (b)). In addition, because data augmentation is applied to obtain $S^{(5)}$ and thus its distribution characteristics are somewhat different from those of the test dataset used.

Figure 5.3 (a) shows bar graphs, and each bar represents the number of test data for which the median of the vector components is within the range of values on the horizontal axis. It represents that most of the test data have a small median value and thus a small contribution to the predictive classification result. In addition, the data distribution in the $S^{(K)}$ differs, where $S^{(1)}$ can be considered to follow the distribution of the original training dataset D because it is randomly selected. In Figure 5.3 (b), we consider this $S^{(1)}$ as a reference, and for each component of the neuron activity vector, we show the difference between the case of $S^{(1)}$ and the others. As the difference is large, the effect by the deviation from the $S^{(1)}$ distribution (and thus the original D) is large. In other words, it suggests that the effect of training data editing is large.

From the above, it can be said that the training data could be re-arranged and crafted, based on the neuron coverage, in such a way that the effect on the model accuracy is minute while the discrepancy of the neuron activity vector is apparent (Figure 5.3 (b)). For example, $S^{(3)}$ is the result of left-side editing, in which the training data with small neuron coverage were removed. Qualitatively, Figure 5.3 implies that $S^{(3)}$ is considered to have contributed to the removal of

outliers. In order to discuss further quantitatively, it is necessary to establish an empirical test method standardized, a kind of test-time augmentation method, for the model robustness.

5.5 Conclusion

In FY2021, we studied debugging and testing methods of training data for the case where bias in training data is the root cause of defects. Here, defects are judged from two quality characteristics, the model accuracy and model robustness. In general, it is necessary to debug training data considering that these two characteristics have a trade-off relationship. In the study, the main points of the training data debugging method are studied in view of the notions obtained for the membership inference, which has been discussed in the context of privacy quality characteristics, and attributed to the problem of outlier detection. However, how to define outliers is non-trivial. We proposed a method to estimate outliers in training data by computing the neuron coverage from the activated states inside the model and extracting the outliers from the bias in the distribution of the neuron coverage values. Experiments showed that the proposed method might provide a piece of information to aid debugging of training data.

In the future, we will establish a method to detect outliers for the purpose of training data debugging through refining the experiment methodology presented here and conducting experiments systematically. Furthermore, we will study methods to conduct debugging from the viewpoints of both model accuracy and model robustness.

6 Evaluation and Improvement of Robustness

In this chapter, *robustness* means the ability that a machine-learned model keeps correct output even when noise is added to input (including adversarial examples). For example, it evaluates how much noise can be added to the model without changing the correct results. One of the measures of its robustness is the maximum safe radius (MSR). In this chapter, we explain adversarial example and the maximum safe radius in a classifier based on a feedforward neural network, and then report the results of a survey on techniques for estimating and increasing the maximum safe radius.

6.1 Robustness measure (maximum safe radius)

It is well known that machine-learned models on inference programs mis-classify input data even when very small perturbations are added. Such perturbated data are called adversarial examples [47], and adversarial examples have been actively researched in recent years. The set $Adv_\delta(x)$ of all adversarial examples contained in the δ-neighborhood (inside the sphere of radius $\delta \in \mathbb{R}$, where $\mathbb{R}$ is the set of real numbers) of the input data sample $x \in \mathbb{R}^n$ is defined as follows:

$$Adv_\delta(x) = \{x' \mid \| x - x' \| \leq \delta \ \wedge \ f(x) \neq f(x')\},$$

where $f(x)$ is a function representing the machine-learned model that takes the input sample x and return the classification, and $\|x - x'\|$ is the distance between two data samples x and x'. The p-norm is often used to define the distance.

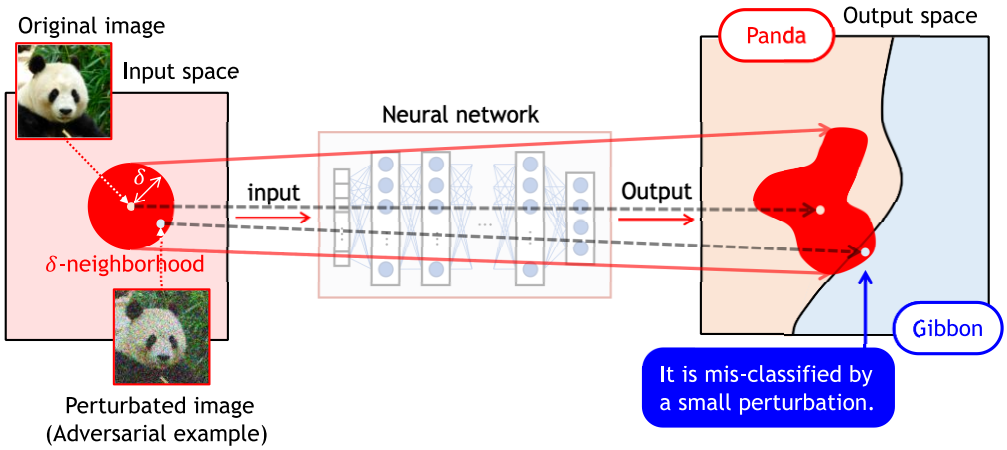


Figure 6.1 An adversarial example from an image of a panda, which is mis-classified into a gibbon

Adversarial examples are explained by Figure 6.1. The left side in Figure 6.1 shows the input space to the neural network and the right side shows the output space from the neural network. The center of the red sphere in the input space represents an original input image of a panda, and the inside of the sphere, whose radius is δ, (i.e., δ-neighborhood of the original image)

represents the set of perturbated images obtained from the original image by adding noises whose sizes are less than δ. The set of outputs from the neural network for all the input images in the δ-neighborhood corresponds to the red region in the output space on the right. Here, a part (lower-right) of the red region in the output side is beyond the decision boundary and is mapped into the region of gibbons. It means misclassification, and the input images mapped to the lower-right part are adversarial examples.

If there is no adversarial example in the δ-neighborhood of the input data x (i.e., inside the sphere whose radius is δ and center is x), then δ is said to be the *safe radius* of x. Then, the maximum safe radius of x, denoted by $MSR(x)$, is defined as follows:

$$MSR(x) = \max\{\delta \mid Adv_\delta(x) = \emptyset\}.$$

When the maximum safe radius of x is large, it is difficult to generate adversarial examples. Therefore, the maximum safe radius can be used as a measure of the robustness to input perturbations, including adversarial examples, of machine-learned models.

The radius δ in Figure 6.1 is not a safe radius because some perturbated input images inside the δ-neighborhood are misclassified into gibbons. On the other hand, δ in the following Figure 6.2 is the maximum safe radius because all the input images inside the δ-neighborhood in Figure 6.2 are correctly classified.

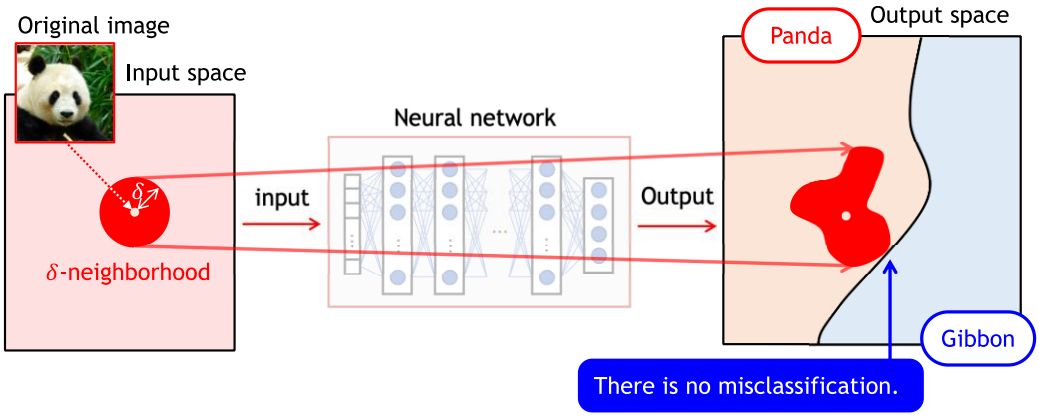


Figure 6.2 The maximum safe radius δ

6.2 A survey on methods for evaluation and improvement of robustness

Table 6.1 shows recent research papers on methods for evaluation and improvement of robustness, where each small box in the table represents a research paper with reference and the information on neural networks used in the experiments for evaluating the methods proposed in the paper. The information is useful for comparing applicable scales of the methods. Table 6.1 is categorized by the following perspectives:

Table 6.1 Methods for evaluation and improvement of robustness (MSR: Maximum Safe Radius)

			Evaluation of robustness	**Improvement of robustness**
Certified	Rigorous		Rigorous estimation of MSR Katz et al. 2017 (Reluplex) [48] ACAS-XU-DNN, 300 ReLU nodes 6 hidden layers, (Limitation: hundreds of nodes) Tjeng et al. 2019 [49] CIFAR-10, ResNet, 9-CNN, 2-layer, 107,496 ReLU units, 100~1,000 times faster than Reluplex	
	Approximative	Deterministic	Estimation of a lower bound (LB) of MSR Weng et al. 2018 (Fast-Lin) [50] CIFAR, 6-layer, 12,288 ReLU units About 10,000 times faster than Reluplex Boopathy et al. 2019 (CNN-Cert)[51] CIFAR-10 (32x32x3), 5-layer, 10 filters, 29,360 hidden nodes, Faster than Fast-Lin	Training by detecting all the adversarial exes Wong and Kolter 2018 [55] SVHN (32x32x3), 2-conv, 32-ch, 100, 10 hidden units, ReLU, (Non-applicable to ImageNet)
		Probabilistic	Estimation of a probabilistic LB of MSR Weng et al. 2019 (PROVEN) [52] CIFAR, 5-layer, CNN, ReLU almost same as CNN-Cert	Randomized smoothing after training Lecuyer at el. 2019 [56] ImageNet (299x299x3), Inception-v3 + auto-encoder Cohen at el. 2019 [57] ImageNet (299x299x3), ResNet-50 (50-layer) Tighter certification than Lecuyer [56]
Uncertified			Estimation of an upper bound (UB) of MSR Carlini and Wagner 2017 [53] ImageNet (299x299x3), Inception-v3 Estimation of an approximation of MSR Weng et al. 2018 (CLEVER) [54] ImageNet (299x299x3), ResNet-50 (50-layer)	Training by detecting near adversarial exes Madry et al. 2018 [58] CIFAR (32x32x3), 28-10 wide ResNet

- Columns in Table 6.1 (application):
  - Evaluation of robustness by estimating MSR
  - Improvement of robustness by increasing data samples with a specified MSR
- Row in Table 6.1 (certification and strictness):
  - Certification of no existence of adversarial examples in δ-neighborhood
    - Rigorous estimation of MSR
    - Approximative estimation of MSR
      - Deterministic (no adversarial example exist)
      - Probabilistic (the probability of no adversarial example is ρ%)
  - No certification of no existence of adversarial examples in δ-neighborhood

The methods in Table 6.1 are explained in the following Subsections 6.2.1~6.2.7.

6.2.1 Certified and rigorous evaluation of robustness

Katz et al. [48] proposed a method, Reluplex, to verify that a machine-learned model satisfies given properties. A demonstration tool that implements the method Reluplex has also been released. Properties are constraints on input-output relations of machine-learned models, and Reluplex can exhaustively and rigorously (soundly and completely) verify that there is no adversarial example in the δ-neighborhood of the input data sample. Therefore, the maximum safe radius (MSR) can be estimated by checking the existence of adversarial examples by changing the radius δ with binary search. Reluplex is an extended Simplex method (one of solvers for linear programming problems) with rules for the ReLU function and it is implemented by a satisfiability-checking tool (SMT-Solver) with a module for the theory of real numbers. Reluplex is a powerful tool to verify properties in addition to robustness, but the computational cost is expensive and the number of neurons it can verify is a few hundred ReLUs at most.

Tjeng et al. [49] proposed an efficient method for estimating maximum safe radii. Then, they implemented the method on a mixed integer linear programming (MILP) solver and demonstrated that the tool can exactly estimate the maximum safe radii of a neural network with 100,000 ReLU-type neurons. Although it is still difficult to apply the rigorous solver-based tools to practical large-scale machine-learned models, the scalability is being improved.

6.2.2 Certified, approximative, and deterministic evaluation of robustness

Weng et al. [50] proposed a method, Fast-Lin, to approximate the maximum safe radii of ReLU-type neural network. Fast-Lin linearly approximates the output region with a polytope and estimates an approximation δ that is slightly smaller than the maximum safe radius, as shown in Figure 6.3. It is guaranteed that there is no adversarial example inside the δ-neighborhood because the approximation δ does not exceed the maximum safe radius (i.e. sound). It means δ is a safe radius and is a lower bound of the maximum safe radius ($\delta \leq MSR(x)$). It was reported that Fast-Lin is 10,000 times faster than the rigorous method Reluplex by approximative convex

outer polytopes.

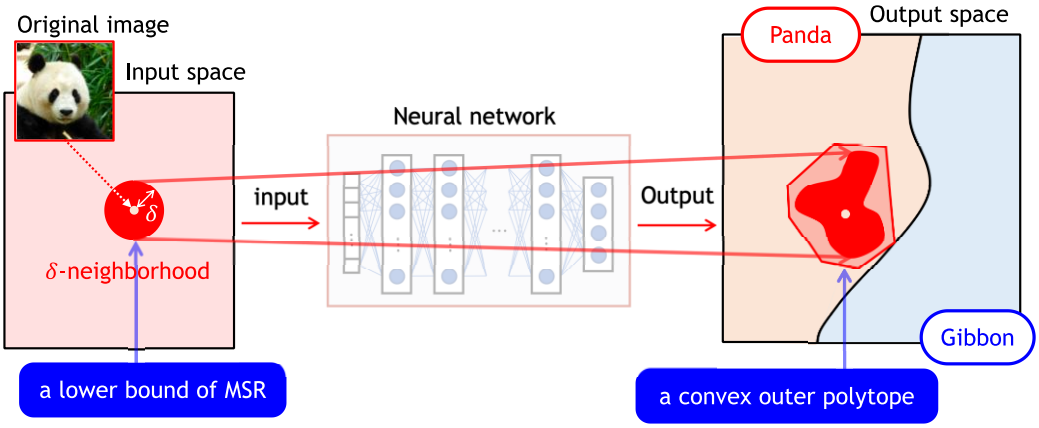


Figure 6.3 An approximation δ that is slightly smaller than the maximum safe radius (MSR)

Boopathy et al. proposed CNN-Cert, which is an improved version of Fast-Lin [51]. CNN-Cert also supports convolutional networks including not only the activation function ReLU but also sigmoid, tanh, and arctan, and it improves approximation accuracy and is faster than Fast-Lin.

6.2.3 Certified, approximative, and probabilistic evaluation of robustness

Weng et al. [52] proposed a method, PROVEN, to approximate probabilistic maximum safety radii. As shown in Figure 6.4, the probabilistic maximum safe radius δ with a probability ρ means that there is no adversarial example inside the δ-neighborhood with a probability ρ. In other words, it permits the existence of adversarial examples with the probability $(1-\rho)$. PROVEN has been developed based on CNN-Cert, and the computational complexity has not significantly increased from CNN-Cert.

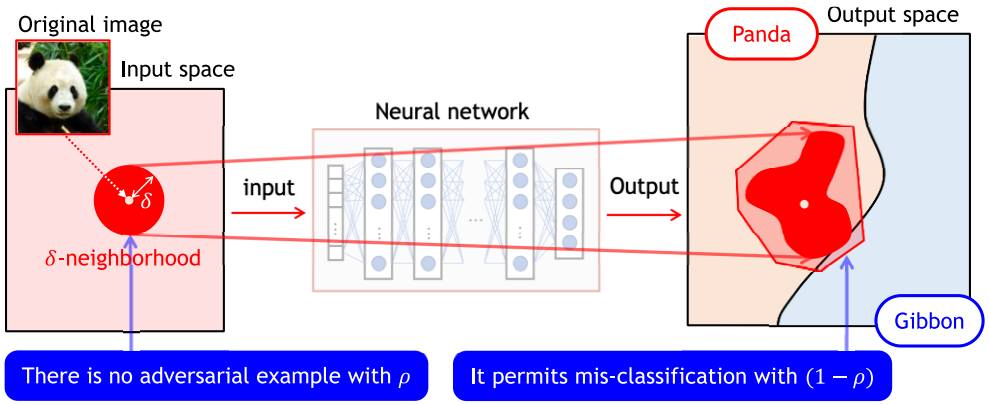


Figure 6.4 An approximation δ that is slightly smaller than the probabilistic MSR with ρ

6.2.4 Uncertified evaluation of robustness

Carlini and Wagner [53] proposed a method to detect the (almost) closest adversarial example to the input data sample x and estimate the distance δ as an approximative maximum safety radius by using an existing optimization tool (Adam). However, it is not guaranteed that the distance δ estimated by the method is the shortest distance to the adversarial example, and there is a possibility that there are adversarial examples closer than the distance. In other words, it is an upper bound of the maximum safe radius ($MSR(x) \leq \delta$). Although it is not guaranteed that the distance δ estimated by the method is a safe radius, it is often used for evaluation in recent papers on robustness as a measure of the maximum safe radius.

Weng et al. [54] proposed the method CLEVER to estimate an approximate maximum safe radius as an evaluation measure of robustness independent of attack methods. It was reported that the method could be applied to relatively large neural networks and the image recognition model Inception-v3 was evaluated in about 10 seconds. The method estimates an approximative maximum safe radius based on the maximum effect in output caused by small changes in input, where the maximum effect is approximated by the extreme value theory. As shown in Figure 6.5, the estimated value δ can be larger than the maximum safe radius, and thus there is a possibility that adversarial examples exist inside the δ-neighborhood (i.e., it is not guaranteed that δ is the safe radius).

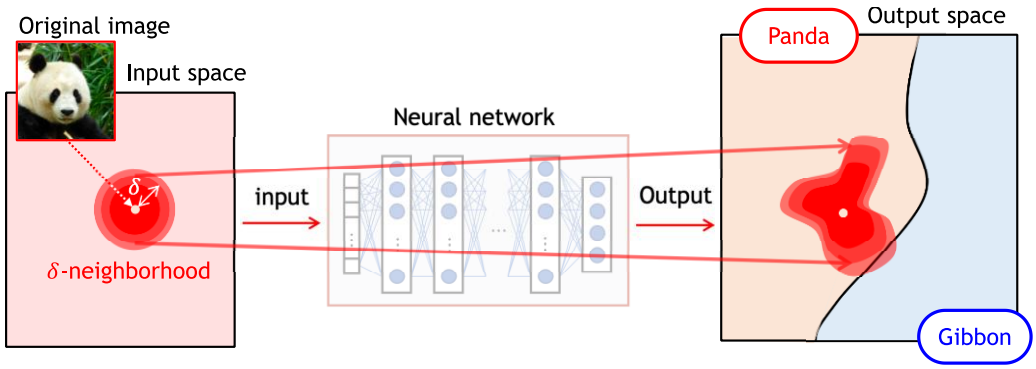


Figure 6.5 An approximation of the maximum safe radius (uncertified)

6.2.5 Certified, approximative, and deterministic improvement of robustness

Wong et al. [55] proposed a method (robust training) to train such that the maximum safe radius of each data in the training dataset to be a specified value δ. Although this method does not guarantee that the maximum safe radius δ is obtained for every training data sample after training, it also gives a method to estimate an approximative value (a safe radius) of the maximum safe radius for each input data sample. In the robust training, neural networks try to learn such that they correctly make inferences for not only training data samples but also the δ-neighborhood of every sample.

A sketch of the robust training is shown in Figure 6.6, where the black dotted line in the output space represents the decision boundary learned by a normal training, and the red solid line represents the decision boundary learned by the robust training. The six training data samples in the input space are correctly classified by both the boundaries, but some data in the δ-neighborhood of each sample are misclassified by the dotted boundary (normal training). On the other hand, data in the δ-neighborhood of each sample are also learned in the robust training as shown in the red boundary. The robust training can guarantee some safe radii, but it is difficult to apply the training to practical large scale neural networks due to the low scalability. Wong et al. [55] reports that the robust training was successfully applied to the datasets of images, MNIST (28×28) and SVHN (32×32) but was not applicable to ImageNet (256×256).

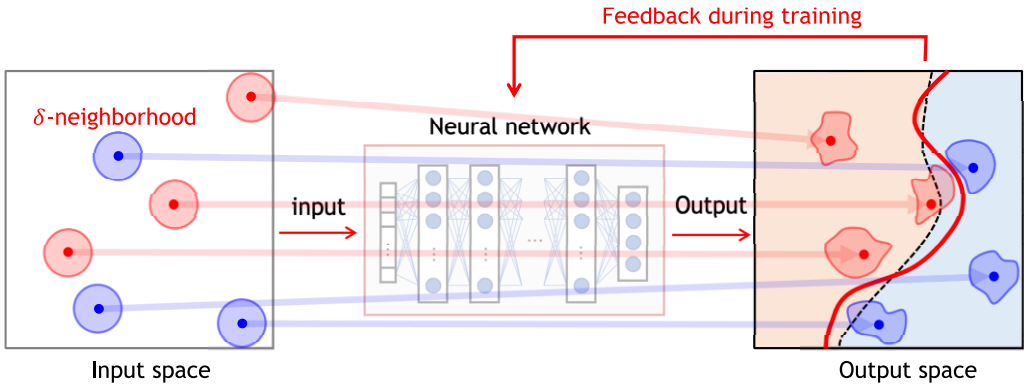


Figure 6.6 Robust-trining by input data with δ-neighberhood

6.2.6 **Certified, approximative, and probabilistic improvement of robustness**

Lecuyer et al. [56] proposed a method to estimate maximum safe radii that can be probabilistically guaranteed by randomized smoothing. In the randomized smoothing, the inference for the same input is repeated in a neural network where a noise layer is added after training, and the final output is the average of the outputs obtained by the repeated inferences.

A sketch of the randomized smoothing is shown in Figure 6.7, where the black dotted line in the output space represents the decision boundary without randomized smoothing, and the red solid line represents the decision boundary with randomized smoothing. The randomized smoothing of Lecuyer et al. [56] improves robustness by smoothing decision boundaries with certification of safe radii and has been successfully applied to guarantee the robustness of machine learned models for large-scale input data such as ImageNet ($299 \times 299 \times 3$). When the variance of the added noise is increased, the guaranteed safe radius also increases, but on the other hand, the correctness (e.g., accuracy) decreases. Lecuyer et al. [56] applied the technique of differential privacy, where the output for two similar inputs is made statistically indistinguishable, to clarify the relations between certifiable approximative probabilistic maximum safe radii, the standard deviation of noise, the number of inferences, and so on.

Cohen et al. [57] proposed a randomized smoothing based method that can estimate tighter certifiable approximative probabilistic maximum safe radii than one of Lecuyer et al. [56].

Although randomized smoothing needs repeated inferences (tens or hundreds of times experimentally) for an input, it can probabilistically guarantee robustness even for large-scale networks.

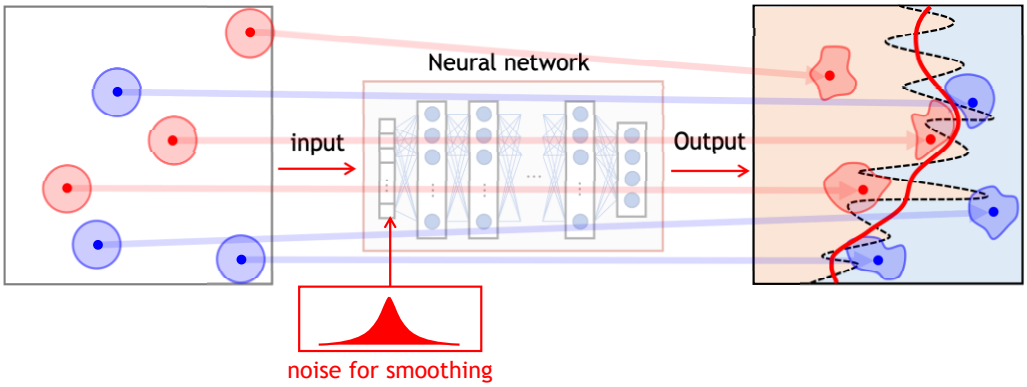


Figure 6.7 Improvement of robustness by randamized smoothing

6.2.7 **Uncertified improvement of robustness**

Madry et al. [58] proposed a method (adversarial training) to train such that maximum safe radius of each data in the training dataset to be a specified value δ. In the adversarial training, samples to be potentially adversarial examples in δ-neighborhood are detected during training and are also used as training data. Compared to the robust training of Wong et al. [55], the adversarial training cannot guarantee robustness, but it is more applicable to larger networks. In addition, compared to randomized smoothing, the adversarial training does not require repeated inferences.

6.3 Conclusion

In general, improvement of robustness tends to decrease accuracy, and currently accuracy is often more important. However, if robustness is not considered, accuracy may rapidly decrease even by small input perturbations. Therefore, robustness is important in critical systems. The methods related to the maximum safe radius, which is a measure of robustness, explained in this chapter have been proposed recently, and environments for applying such methods have not been established well yet. Since such methods have been experimentally applied also to practical machine learned models, we think that the maximum safe radius can be one of measures of robustness in a few years.

7 Generalization Bounds of Machine-Learned Models

In this chapter, we briefly introduce well-known theorems on generalization bounds, that are the expected values of the error rates of machine-learned models for *all* input data, and we show some computational results obtained by applying the theorems. We are aiming at guaranteeing the behavior of machine-learned models even for *unseen* input data.

7.1 Generalization bounds

In this chapter, we model *machine-learned neural networks* as shown in Figure 7.1, with the *input-output relations*. In particular, we focus on feed-forward neural networks trained as *classifiers* by supervised deep learning and denote the input-output relation as a *function* $y = f_w(x)$, where x and y are an input and the correct output, respectively, and w is the weights (training parameters) on connections between neurons in the neural network. In the field of *statistical learning theory*, the function f_w is often called a *hypothesis*, but it is called a *machine-learned model* in this chapter, as in the other chapters. Since a neural network can express multiple machine-learned models f_w by adjusting weights $w \in \mathcal{W}$, the set of expressible machine-learned models f_w in the network is denoted by $\mathcal{F}$. Hence, *Machine learning* means to select a machine-learned model f_w from $\mathcal{F}$ by a training algorithm such that f_w fits the training dataset. The model f_w can be denoted by f when the parameter w is not important.

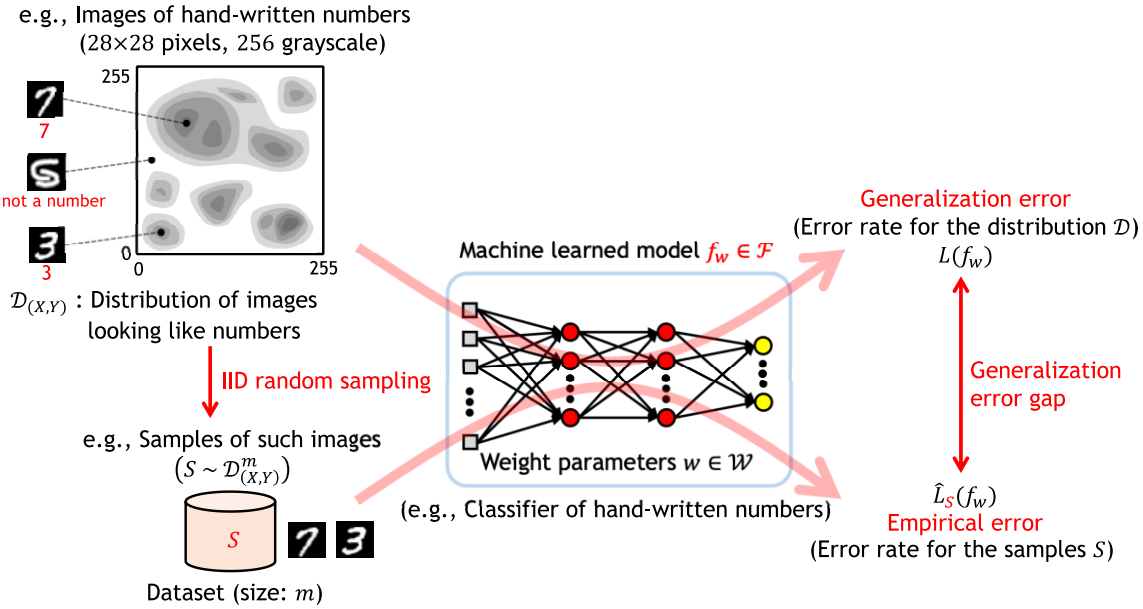


Figure 7.1 Generalization error and empirical error

The *generalization error* $L(f_w)$ of the machine-learned model f_w (a classifier) is the expected value of the error rate of f_w for every pair (x, y), of an input x and the correct output y, randomly selected according to the distribution $\mathcal{D}$, and is defined as follows:

$$L(f_w) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[\mathbb{I}(y \neq f_w(x))],$$

where $\mathbb{I}(b)$ is the following indicator function, that returns 0 if b is true and returns 1 otherwise, and therefore $\mathbb{I}(y \neq f_w(x))$ is the 0-1 loss function:

$$\mathbb{I}(b) \ = \text{ if } (b = \text{true}) \text{ then 1 else 0.}$$

For example, at the top left in Figure 7.1, the input space, that is drawn in 2 dimensions for simplicity but exactly has 784 dimensions, means "the distribution $\mathcal{D}$ of images ($28 \times 28 = 784$ pixels with 256 grayscale) looking like numbers." In this case, the generalization error is the expected value of the error rate of f_w for every image looking like numbers. Here, note that it is not the error rate for all the (256^{784}) images in the input space.

The *empirical error* $\hat{L}_S(f_w)$ of the machine-learned model f_w is the error rate of f_w for m input data samples in the dataset $S\sim\mathcal{D}^m$ selected according to distribution $\mathcal{D}$:

$$\hat{L}_S(f_w) = \frac{1}{m}\sum_{(x,y)\in S}[\mathbb{I}(y \neq f_w(x))].$$

Especially, if S is the training dataset, then $\hat{L}_S(f_w)$ is also called the *training error*, and if T is the testing dataset, then $\hat{L}_T(f_w)$ is also called the *testing error*.

7.2 The theory of generalization bounds

Even though it is almost impossible to exactly compute generalization errors because there are innumerable data in input-spaces, various theorems on *generalization bounds*, that are (upper) bounds of generalization errors, have been proposed for guaranteeing that "the generalization error of a machine-learned model is less than a generalization bound with probability at least p%," where p is the confidence of the bound. In this section, we briefly classify the theorems on the generalization bounds in Subsection 7.2.1, and then we explain the theorems in Subsections 7.2.2~7.2.5.

7.2.1 A classification of generalization bounds

In this subsection, according to the classification of the generalization bounds as shown in Figure 7.2, we briefly explain the feasibility for applying generalization bounds to the evaluation of generalization performance of machine-learned models. In Figure 7.2, it seems difficult to apply the VC bounds or the Rademacher bounds for evaluating the generalization performance of (trained) machine-learned models because they give upper bounds of the *worst* model in the set $\mathcal{F}$ of expressible machine-learned models in a given neural network. The PAC-Bayes bounds are effective for comparing the generalization performance of two or more machine-learned models [59][60] even though the absolute values of the bounds often are *vacuous* (close to or more than 100%). The Chernoff bounds can give upper bounds very close to the generalization errors by providing a large amount of testing data separately from the training dataset.

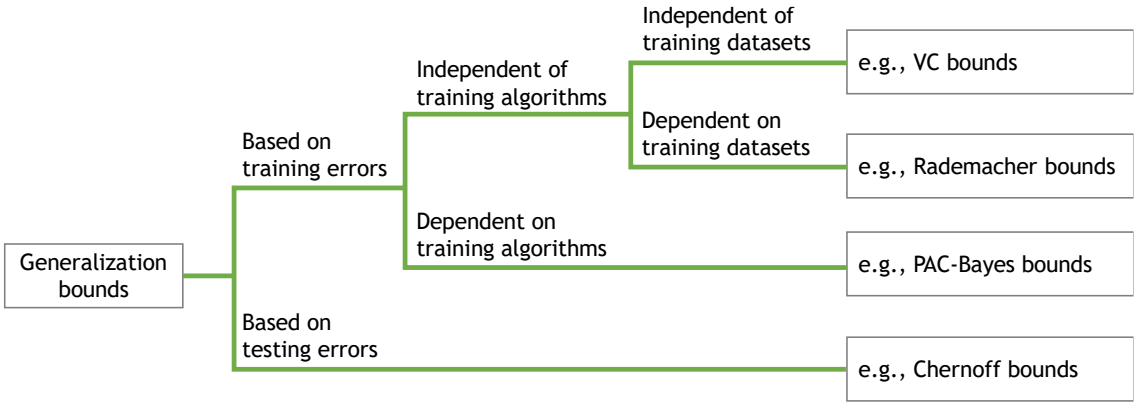


Figure 7.2 A Classification of generalization bounds and well-known examples of bounds

In most cases, generalization bounds based on training errors (i.e., training datasets) is larger than generalization bounds based on testing errors (i.e., testing datasets). This tendency is also seen not only for the PAC-Bayes bounds but also for the others such as bounds based on output margins [61], bounds based on the stability of the training algorithm [62], and so on.

The advantage of generalization bounds based on training errors is that they can be applied to the study of training (algorithms) for reducing generalization errors. The other advantage is that they can be computed only by training datasets without additional datasets such as testing datasets. For example, when the number of data samples is very small (e.g., a few dozen), it was reported that the PAC-Bayes bounds, where all samples were used for training, were able to be lower than the Chernoff bounds, where the samples were separated for training and testing [63].

Recently, several methods have been proposed for computing *non-vacuous* generalization bounds (less than 100%) even based on training errors. For example, such methods use distributions of machine-learned models (i.e., input-output functions instead of weights) in the PAC-Bayes bounds [64], or random labelled data in training [65], or model compression [66]. Furthermore, methods have also been proposed to optimize the distribution (the mean and the standard deviation) of each weight to reduce generalization errors using the PAC-Bayes bounds as objective functions [67][68]. In the near future, it is expected that generalization bounds based on training errors will also be effective as a measure of the generalization performance of machine-learned models, but currently, we consider that it is more realistic to adopt generalization bounds based on testing errors.

7.2.2 **VC bounds**

The VC dimension $VC(\mathcal{F})$ is a complexity measure of the set $\mathcal{F}$ of expressible machine-learned models and it means the maximum number of data that can be divided by $\mathcal{F}$ [69]. Then, by using the VC dimension $VC(\mathcal{F})$, the theorem, *VC bounds* [59], guarantees that the following inequality holds with probability $(1-\delta)$ at least, for any training data set $S \sim \mathcal{D}^m$ (size: m) and for any machine-learned model $f \in \mathcal{F}$:

$$L(f) \leq \hat{L}_S(f) + 144\sqrt{\frac{VC(\mathcal{F})}{m}} + \sqrt{\frac{\ln\frac{1}{\delta}}{m}},$$

where $\delta \in (0,1)$ represents the *uncertainty* of the inequality.

Most of computation results of VC-bounds exceed 100% because the bounds consider the worst case such that any model is selected from the set $\mathcal{F}$ for any training dataset when the set $\mathcal{F}$ of expressible machine-learned models (i.e., architecture of a neural network) is given, in other words, the bounds are independent of training dataset and training algorithm. Therefore, it is not appropriate to use it to evaluate the generalization performance of trained machine-learned models.

7.2.3 **Rademacher bounds**

The Rademacher complexity is the complexity $R(S,\mathcal{H})$ of the set $\mathcal{F}$ of expressible machine learned models for the dataset S [69]. Then, by using the complexity $R(S,\mathcal{H})$, the theorem, *Rademacher bounds* [59], guarantees that the following inequality holds with probability $(1-\delta)$ at least, for any training data set $S\sim\mathcal{D}^m$ (size: m) and for any machine-learned model $f \in \mathcal{F}$:

$$L(f) \leq \hat{L}_S(f) + 2R(S,\mathcal{F}) + 4c\sqrt{\frac{2\ln\frac{4}{\delta}}{m}},$$

where c is a constant.

The Rademacher bounds are lower than the VC bounds because the training dataset S is considered, but the most of bounds also exceed 100% because the bounds are still independent of training algorithm. Therefore, it is not appropriate to use the Rademacher bounds to evaluate the generalization performance of trained machine-learned models by the same reason as the VC bounds.

7.2.4 **PAC-Bayes bounds**

PAC (*Probably Approximately Correct*) represents that a machine-learned model f_w trained by a training dataset is an *approximation* of the *correct* model and the generalization error gap is less than a threshold with a *probability*. *PAC-Bayes* considers the expected value $\mathbb{E}_{w\sim Q}[L(f_w)]$ of the generalization error of the *probabilistic* machine-learned model f_w whose weights w are randomly selected according to the probability-distribution Q instead of fixed values.

Although several theorems on the PAC-Bayes bounds have been proved, two well-known theorems (Catoni bounds and Maurer bounds) are introduced in this subsection. The theorem, *Catoni bounds* [70], guarantees that the following inequality holds with probability $(1-\delta)$ at least, for any posterior distribution Q and for any $\beta, \delta > 0$, when a training dataset $S\sim\mathcal{D}^m$ (size: m) and a prior distribution P whose weights independent of S are given:

$$\mathbb{E}_{w\sim Q}[L(f_w)] \leq \frac{1}{1-\exp(-\beta)}\left(1-\exp\left(-\beta\mathbb{E}_{w\sim Q}[\hat{L}_S(f_w)]-\frac{1}{m}\left(\mathrm{KL}(Q \parallel P)+\ln\frac{1}{\delta}\right)\right)\right).$$

The theorem, Maurer bound [71], guarantees that the following inequality holds with probability $(1-\delta)$ at least, if $m \geq 8$:

$$\mathbb{E}_{w\sim Q}[L(f_w)] \leq kl^{-1}\left(\mathbb{E}_{w\sim Q}[\hat{L}_S(f_w)],\ \frac{1}{m}\left(KL(Q \parallel P)+\ln\left(\frac{2\sqrt{m}}{\delta}\right)\right)\right),$$

where $KL(Q \parallel P)$ is the *KL-divergence* (*Kullback-Leibler divergence*) that shows the difference between the two distributions Q and P, and it is defined as follows:

$$KL(Q \parallel P)=\int_{\mathcal{W}} Q(w)\ln\left(\frac{Q(w)}{P(w)}\right)dw,$$

and $kl^{-1}(q,b)$ is the *binary KL-inversion* defined by

$$kl^{-1}(q,b)=\sup\{\, p\in[x,1] : kl(q \parallel p)\leq b \,\},$$

where $kl(q \parallel p)$ is the binary KL-divergence (the KL-divergence of the Bernoulli distributions of q and p) defined as follows:

$$kl(q \parallel p)=\sum_{k\in\{0,1\}} q^k(1-q)^{1-k}\ln\left(\frac{q^k(1-q)^{1-k}}{p^k(1-p)^{1-k}}\right)=q\ln\left(\frac{q}{p}\right)+(1-q)\ln\left(\frac{1-q}{1-p}\right).$$

As shown in the Catoni bounds and the Maurer bounds, the PAC-Bayes bounds contain the posterior distribution Q of the trained machine-learned model (i.e., they depend on the training algorithms). Compared with the VC bounds and the Rademacher bounds, the PAC-Bayes bounds can give generalization bounds closer to the generalization errors, but the computation results of the PAC-Bayes bounds are often close to 100% (i.e., vacuous).

7.2.5 **Chernoff bounds**

There are generalization bounds based on testing errors for the dataset $T\sim\mathcal{D}^m$ (size: m) that is not used in training (i.e., held-out dataset prepared for evaluation). For example, the theorem, *Chernoff bounds* [72], guarantees that the following inequality holds with probability $(1-\delta)$ at least:

$$L(f_w)\leq kl^{-1}\left(\hat{L}_T(f_w),\ \frac{1}{m}\ln\left(\frac{1}{\delta}\right)\right),$$

where $kl^{-1}(q,b)$ is the binary KL-inversion explained in Subsection 7.2.4. Here, note that the testing error $\hat{L}_T(f_w)$ is used for expressing the upper bounds. The *Chernoff bounds* can give tight generalization bounds (i.e., close to the generalization errors) when sufficient testing dataset are prepared.

7.3 Computational examples of generalization bounds

In this section, we report some computational examples of generalization bounds based on the theorems introduced in Section 7.2. At first, in Subsection 7.3.1, it is explained how to compute generalization bounds. Next, in Subsection 7.3.2, the computation results are shown.

7.3.1 Computation of generalization bounds

As introduced in Subsection 7.2.5, the Chernoff bounds are useful for evaluating the generalization performance of trained machine-learned models because they can give tight generalization bounds, thus close to the generalization errors, and therefore the bounds are meaningful as absolute values. On the other hand, it has also been reported [59][60] that the generalization bounds, called *perturbation bounds*, of machine-learned models whose weights are perturbated, for example, by the Gaussian noise as shown in Figure 7.3, are useful as *generalization measures* for relatively comparing the generalization performance of machine-learned models. In this section, the standard deviation σ_i of the Gaussian noise $\mathcal{N}(0, \sigma_i^2)$ added to each weight w_i is decided to be proportional to the magnitude of w_i such that $\sigma_i = r|w_i|$, where r is a positive constant, called *SD-rate*. Such addition of noise is called *magnitude-aware perturbation* [59].

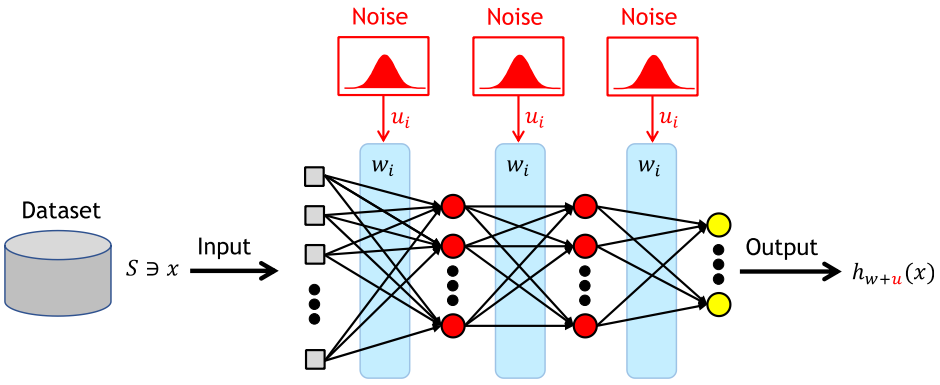


Figure 7.3 A machine-learned model whose weights are perturbated by the Gaussian noise

The theorems on the PAC-Bayes bounds can be applied for computing such generalization bounds of machine-learned models with probability distribution of perturbated weights. In this section, we use the posterior distribution Q equal to the prior distribution P trained by a training dataset S in the PAC-Bayes bounds (i.e., $Q = P$). As explained in Subsection 7.2.4, the posterior distribution Q can depend on the dataset used for computing generalization bounds, while the prior distribution P cannot depend on it. Hence, we compute the generalization bound by using a testing dataset T, separated from S, in this case of $Q = P$. The advantage of $Q = P$ is that the KL-divergence $KL(Q \parallel P)$ is zero. In general, one of the reasons why the PAC-Bayes bounds are close to 100% is that the KL-divergences are large. In this section, we apply the PAC-Bayes bounds in the case of $Q = P$ to the computation of generalization bounds based

on testing errors (i.e., by using testing datasets).

In order to apply the PAC-Bayes bounds, it is necessary to compute the expected value $\mathbb{E}_{u\sim\mathcal{N}(0,r^2)^\omega}\left[\hat{L}_T(f_{w\oplus u})\right]$ of the testing error of the machine learned model f_w with noise u, where ω is the total number of weights (training parameters), r is the rate of noise to weight magnitude (i.e., SD-rate), and $w\oplus u$ represents that the noise u is added to weights w proportionally to the magnitude of each weight element (i.e., for each element $i \in \{1, \dots, \omega\}$, $(w\oplus u)_i = w_i + u_i|w_i|$). However, since it is difficult to exactly compute the expected value, we compute the upper bound of the expected value from the average of testing errors computed with noise in n times. The upper bound $\bar{L}_{T,U,\delta'}(f_w)$ can be defined as follows, for the set $U = \{u_i \mid i \in \{1, \dots, n\}\}$ of randomly sampled n Gaussian noises $u_i \sim \mathcal{N}(0, r^2)^\omega$:

$$\bar{L}_{T,U,\delta'}(f_w) = kl^{-1}\left(\frac{1}{|U|}\sum_{u\in U}\hat{L}_T(f_{w\oplus u}),\ \ \frac{1}{|U|}\ln\frac{2}{\delta'}\right),$$

where δ' is the uncertainty for using the average of testing errors instead of the expected value. Indeed, the following inequality holds with probability $(1-\delta')$ at least (e.g. see Section 6 in [68]):

$$\mathbb{E}_{u\sim\mathcal{N}(0,r^2)^\omega}\left[\hat{L}_T(f_{w\oplus u})\right] \le \bar{L}_{T,U,\delta'}(f_w).$$

Consequently, in this section, we use the following three expressions for computing generalization bounds using a testing dataset T (size: m),

(1) Chernoff bounds: $kl^{-1}\left(\hat{L}_T(f_w),\ \ \frac{1}{m}\ln\left(\frac{1}{\delta}\right)\right)$,

(2) Catoni bounds: $\frac{1}{1-\exp(-\beta)}\left(1-\exp\left(-\beta\,\bar{L}_{T,U,\delta'}(f_w) - \frac{1}{m}\ln\left(\frac{1}{\delta-\delta'}\right)\right)\right)$,

(3) Maurer bounds: $kl^{-1}\left(\bar{L}_{T,U,\delta'}(f_w),\ \ \frac{1}{m}\ln\left(\frac{2\sqrt{m}}{\delta-\delta'}\right)\right)$,

where $KL(Q \parallel P)$ in the Catoni bounds and the Maurer bounds has disappeared because it is zero if $Q = P$. Note that the denominators in the logarithms in the expressions (2) and (3) are replaced by $(\delta-\delta')$ from δ because a part of the uncertainty δ of generalization bounds must be used as the uncertainty δ' of expected values. In this section, the uncertainty δ' is obtained as the solution of the following equation (e.g., by the Newton method) for approximately minimize the expression (3):

$$\left(2^{1-\frac{m}{n}}\sqrt{m}\right)\delta'^{\frac{m}{n}} + \delta' - \delta = 0,$$

where m is the size of the testing dataset and n is the number of testing errors repeatedly computed with noise added. Similarly, the parameter β is given for approximately minimizing the expression (2) as follows:

$$\beta = \sqrt{\frac{2\ln\left(\frac{1}{\delta}\right)}{m\,\bar{L}_{T,U,\delta'}(f_w)\left(1-\bar{L}_{T,U,\delta'}(f_w)\right)}}.$$

The binary KL-inversion $kl^{-1}(q, b)$ in the expressions (1) and (3) can be approximately computed, for example, by the Newton method (e.g., see Appendix C in [67]).

7.3.2 **Computational results of generalization bounds**

In this subsection, we report the computational results of the generalization bounds by the expressions (1), (2), and (3) described in Subsection 8.3.1, for the following two types of neural networks MLP and CNN trained on the dataset MNIST (pixels: 28×28, grayscale: [0,1], training size: 27,000, and testing size: 10,000) of handwritten digit images.

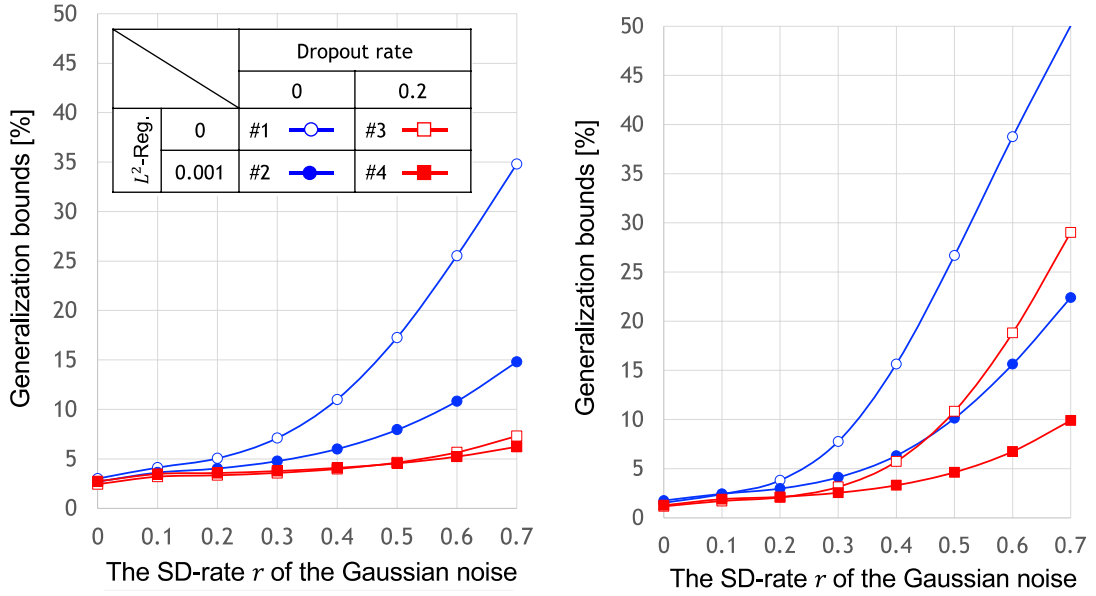
- MLP (Multi-Layer Perceptron):
  - The total number of the training parameters: 118,282
  - The layers: 3 fully connected layers
- CNN (Convolutional Neural Network):
  - The total number of the training parameters: 121,930
  - The layers: 2 convolutional layers, 2 pooling layers, and 2 fully connected layers

For comparing generalization performance, 4 machine-learned models for each neural network MLP/CNN were trained in the parameters shown in Table 7.1

Table 7.1 The training parameters in the 4 machine-learned models

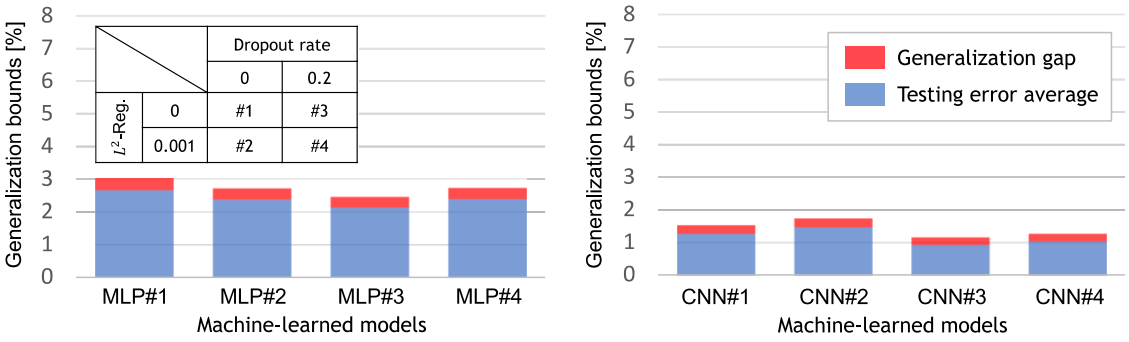
Model ID	Dropout rate	L^2-Regularization
#1	0	0
#2	0	0.001
#3	0.2	0
#4	0.2	0.001

Figure 7.4 shows the computational results of the generalization bounds for the machine-learned models of MLP and CNN (4 models for each), for the SD-rates $r = 0, \dots, 0.7$, where the Chernoff bounds are used for the case $r = 0$ (no noise) and the Catoni bounds are used for the cases $r = 0.1, \dots, 0.7$. The computational results by the Maurer bounds are omitted because they were always almost 1% higher than the Catoni bounds. The confidence $(1 - \delta)$, i.e., the probability that the generalization errors are less than the generalization bounds, is 90% in Figure 7.4. The upper bounds of expected values of testing errors with noise added are computed from the averages of 5,000 testing errors with noise added for 10,000 test data samples (i.e., $m = 10{,}000$ and $n = 5{,}000$). For the cases of $r = 0$ and 0.3 in Figure 7.4, the generalization gaps (the difference between the generalization bounds and the testing errors) and the expectation gaps (the difference between the expected values of testing errors and the averages) are shown in Figure 7.5 and Figure 7.6, respectively.



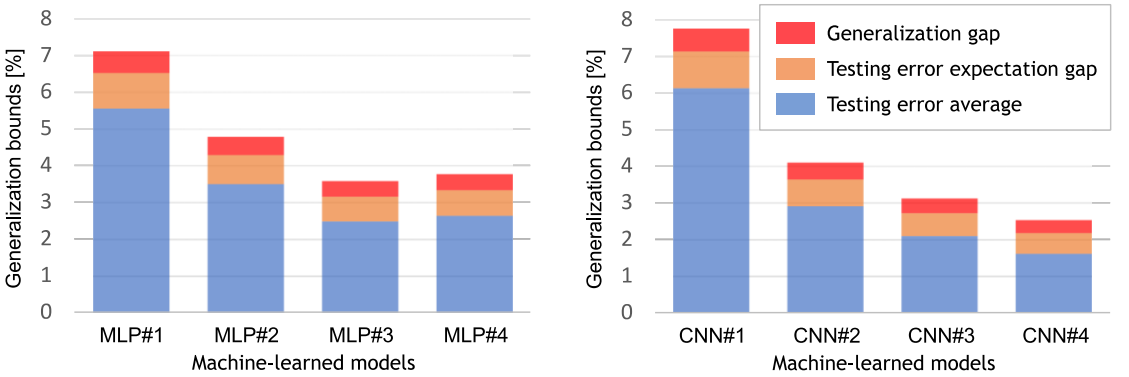
(a) The generalization bounds of MLP#1~4 (b) The generalization bounds of CNN#1~4

Figure 7.4 The computational results by the Chernoff $(r = 0)$ and the Catoni bounds $(r > 0)$



(a) The generalization bounds of MLP#1~4 (b) The generalization bounds of CNN#1~4

Figure 7.5 The generalization gaps in the Chernoff $(r = 0)$



(a) The generalization bounds of MLP#1~4 (b) The generalization bounds of CNN#1~4

Figure 7.6 The generalization gaps and the expectation gaps in the Catoni bounds $(r = 0.3)$

As shown in Figure 7.5, the generalization bounds by the Chernoff bounds are very close to the testing errors and the values of the bounds are meaningful on their own (as absolute values). On the other hand, as a relative evaluation between machine-learned models, the generalization bounds in Figure 7.5 do not show any difference from the (normal) testing errors by data samples. For example, the generalization bound of CNN#2 with L^2-regularization is larger than the bound of CNN#1 as shown in Figure 7.5 (b). It seems that the generalization bounds are not enough to evaluate generalization performance. Then, in order to evaluate the generalization performance at different viewpoints from data samples, it is useful to add noise to weights in neural networks. Figure 7.4 and Figure 7.6 clearly show that the L^2-regularization and the dropout can suppress increase of the generalization bounds when noise increases. Although such suppression-effect is empirically well-known, it is an advantage of the generalization bounds that they can quantitatively evaluate such effect and can probabilistically *guarantee* the upper bounds of the generalization errors according to the statistical learning theory.

In the rest of this subsection, as an example of generalization bounds based on training errors, we show a computational result of MLP#4 (i.e., with the dropout and the regularization). In the training of MLP#4, each initial weight w_{0i} was randomly selected according to the normal distribution $P_i = \mathcal{N}\left(0, \sigma_{0i}^2\right)$, i.e., the mean is 0 and the standard deviation is σ_{0i}. For each weight w_i, the KL-divergence $KL(Q_i \parallel P_i)$ between the prior distribution P_i and the posterior distribution $Q_i = \mathcal{N}(w_i, (r|w_i|)^2)$ is equal to or larger than $(1/2r^2)$. Here, the equality holds if $\sigma_{0i} = r|w_i|$, but we cannot assume the equality because σ_{0i} cannot depend on the training dataset (i.e., on w_i). Then, the minimum of the total KL-divergence of a neural network is $(\omega/2r^2)$, where ω is the total number of weights. For example, the minimum of the KL-divergence of MLP#4 ($\omega = 118{,}282$) in the case of $r = 0.5$ is $236{,}564$. Then, the generalization bound of the MLP#4 ($r = 0.5$) computed by the Catoni bounds is 99.99% with probability 90% at least, although the average of $5{,}000$ training errors was 2.17%. The several techniques for reducing the generalization bounds based on training errors have been proposed, but it is thought to be currently practical to use testing datasets (i.e., the generalization bounds based on testing errors).

7.4 Towards the evaluation of "the stability of trained models"

The stability of trained models is one of the nine internal quality characteristics described in Machine Learning Quality Management Guideline [1] and it represents that machine-learned components reasonably behave even for unseen input data. The theorems and the computations on the generalization bounds explained in this chapter make it possible to theoretically guarantee the probabilistic upper bounds of the generalization errors (i.e., expected values of error rates for all data including unseen data). Therefore, the generalization bounds will be one of useful methods for evaluating "the stability of trained models."

In Subsection 7.3.2, it has been shown that the Chernoff bounds based on testing errors can give meaningful values close to generalization errors, In addition, it has been also shown that

the PAC-Bayes bounds (e.g., the Catoni bounds) can gives meaningful *perturbated* generalization bounds based on testing errors (not on training errors) for *probabilistic* machine learned models by adding noise to the weights. Such perturbation bounds make it possible to quantitatively evaluate the performance at different viewpoints from normal testing by data samples.

As an additional investigation, we note the relation between perturbated generalization bounds and the randomized smoothing introduced in Subsection 6.2.6. In the randomized smoothing, certifiable approximative probabilistic maximum safe radii can be estimated by adding noise to input data. We are still investigating the possibility that the perturbated generalization bounds can theoretically guarantee robustness in a similar way.

8 Adversarial Example Detection

8.1 Research summary

With the goal of practically establishing a method for determining whether a given input image is an adversarial example, we focus on the following points regarding attacks and detection methods that generate adversarial examples. We are conducting a survey of typical technologies.

- Supporting adversarial example detection program code and confirmation by computational experiment
- Reproduction of experimental results of adversarial example detection method papers
- Implementation of the framework for detecting adversarial examples

Adversarial example detection stands for detecting adversarial examples from given inputs, and existing state-of-the-art adversarial example detection methods can be divided into four main categories.

① Metric based approaches (example [73])
② Denoisers approaches (example [74])
③ Prediction inconsistency based approaches (example [75])
④ Neural Network Invariant Checking (NIC) approaches (example [76])

In this chapter, we report the results of additional test experiments to compare and evaluate adversarial example detection methods based on each of these approaches ① to ④. As reported in the paper [76], it was confirmed that the approach of ④ (NIC: Neural Network Invariant Checking) shows the highest detection rate among ① to ④. In this follow-up experiment, the published implementation code was used for ① to ③, but the implementation code was not published for ④, so a computer experiment was conducted by implementing the NIC according to the paper [76]. Therefore, this chapter mainly describes the NIC ④.

After explaining the outline of the four approaches, the method of detecting adversarial examples by the NIC is explained, and the implementation method is described. Then, the results of the follow-up experiments of each approach and the experiments by the NIC are described. Finally, we report the implementation of the NIC framework and the effectiveness evaluation.

8.2 Overview of adversarial example detection approaches

In this section, the four state-of-the-art approaches to adversarial example detection are overviewed.

8.2.1 Metric based approaches

A method of performing statistical measurements of inputs (and outputs of each neuron) to detect adversarial examples, Ma et al. recently proposed the use of a measurement called Local Intrinsic Dimensionality (LID) [73]. This method estimates the LID value that evaluates the space-filling capacity of the area surrounding the sample by calculating the distance distribution of the sample and the number of neighbors in each layer, and the adversarial example tends to have a large LID value. It uses certain properties to detect adversarial examples. LID is superior to traditional kernel density estimation (KD) and Bayesian uncertainty (BU) for detecting adversarial examples and is currently the state-of-the-art technology for this type of detector.

8.2.2 Denoisers approaches

It is a method of detecting adversarial examples by removing noise in a preprocessing step for each input. In this method, the training model or noise remover (encoder and decoder) is trained to filter the image so that the key components in the training model can be highlighted. This filter can be used to remove noise added by an attacker to generate adversarial examples and correct misclassification. MagNet [74] is a method of detecting adversarial examples using detectors and reformers (trained automatic encoders and automatic decoders).

8.2.3 Prediction inconsistency based approach

A method of detecting adversarial examples by measuring the discrepancy between the original neural network and the neural network enhanced by human perceptible attributes. Feature Squeezing [75], the state-of-the-art detection technique of this method, can achieve very high detection rates against a variety of attacks. Feature squeezing focuses on detecting gradient-based attacks, focusing on the ability of attackers to generate adversarial examples through the unnecessarily large input feature space of deep neural networks DNN. The procedure for detecting adversarial examples by feature squeezing is shown below.

1. Apply squeezing technology (a technology that reduces the color depth of an image and smooths the image) to the original input image to generate multiple squeezed images.
2. Input the original input image and multiple squeeze images into the deep neural network, and measure the distance between the inference result (prediction vector) of the input image and the inference result of each squeeze image.
3. When one of the differences (distances) between the original input image and the squeeze image exceeds the threshold value, the original input image is detected as an adversarial example.

8.2.4 Neural Network Invariant Checking (NIC) approaches

The NIC (Neural Network Invariant Checking) method focuses on value invariants (VIs) and provenance invariants (PIs) inside deep neural networks [76]. The value invariant VI is the distribution of possible neuron values in each layer, and the provenance invariant PI is the possible neuron value pattern of two consecutive layers (summary of correlation between features across two layers). If an input violates these invariants, the input is detected as an adversarial example. The NIC [76] method trains these invariant VIs and PIs with benign input data and model them as a one-class classification (OCC) problem that detects adversarial examples. A higher detection rate has been reported than the methods based on (1) to (3) explained above. The outline and the implementation of the NIC system design are explained in detail in Sections 8.3 and 8.4, respectively.

8.3 NIC system design overview

The procedure for building the NIC detector (steps A to C: during training, D to E: during execution) is explained by using Figure 8.1 [76]. This invariant VI, PI training uses only non-adversarial benign data.

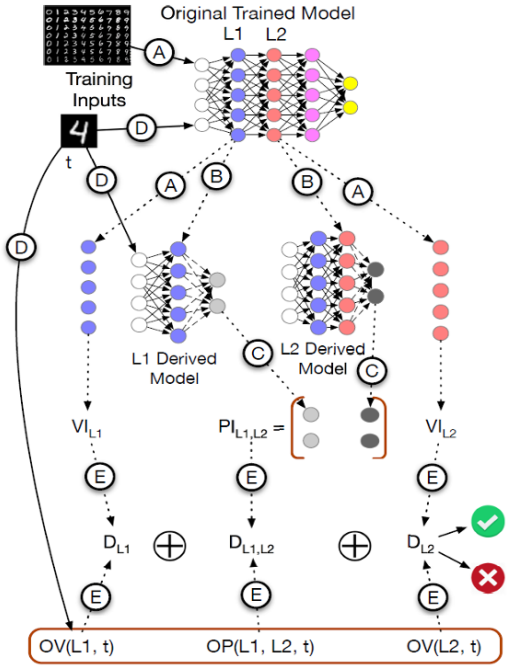


Figure 8.1 Outline of system design (Fig. 8 of thesis [76])

- **Step A**: Collect the output value of each neuron at each layer of each training data input.

- **Step B**: For each layer k (e.g., L1, L2), extract the sub-models from the input layer to the k layer and add a new softmax layer with the same output label as the original model. Then create a derived model (DerivedModel in Figure 8.1)

- **Step C**: Enter each benign training data for all derived models and collect the final output of these models (i.e., the output probability values of the individual classes). For each set of consecutive layers, we train using the distribution of the classification results of this derivative model. This trained distribution is the PI for these two layers.

- **Step D**: Input each test data t (for example, the image of "4" in Figure 8.1) to all derivative models in addition to the original model, and observe the activation value of each layer of the original model. Collect the value OV (for example, OV(L1, t) in Figure 8.1) and the classification result (set) of the derivative model of consecutive layers. From this classification result, the observed source OP (for example, OP(L1, L2, t), etc.) is obtained.

- **Step E**: Calculate the probability D that the OV and OP fit the corresponding VI and PI distributions. The possibility that the input t is adversarial is predicted at the same time by aggregating all these D values.

8.4 NIC system implementation

In order to detect adversarial examples based on NIC, a direct sum space (vector) is constructed from PI and VI, and for classifying this vector, an OSVM (One Class Support Vector Machine) is constructed. When the input to the layer l of the trained DNN (Deep Neural Network) model (hereinafter referred to as M) is x_l, the output f_l of the layer l is given by the following equation:

$$f_l = \sigma(x_l \cdot w_l^T + b_l),$$

where σ is the activation function of the layer l, w_l^T is the weight matrix, and b_l is the bias. At this time, the direct sum spaces classified by VI, PI, and OSVM are obtained as follows.

- VI calculation: The VI of each layer l of model M is determined by solving the following optimization problem.

$$VI_l = \min\left[\sum_{x \in X_b} J(f_l \circ f_{l-1} \circ \cdots \circ f_1(x) \cdots w^T - 1)\right]$$

  Here, J is the error evaluation function, and X_b is the batch used to create M. Also, $\circ$ is a monoid, in this case a vectorized version of f_k.

- PI calculation: $PI_{l,l+1}(x)$ is based on the classification output of the derived models of the layers l and $l+1$. The probability that x is benign (non-adversarial) is estimated by solving the following optimization problem.

$$PI_{l,l+1}(x) = \min\left[\sum_{x \in X_b} J\big(concat\big(D_l(x), D_{l+1}(x)\big) \cdots w^T - 1\big)\right]$$

Here, a derivative model D_l of the layer l is defined as follows, with the softmax layer added after the layer l.

$$D_l = \text{softmax} \circ f_l \circ f_{l-1} \circ \cdots \circ f_1$$

- Direct sum space of PI and VI: From the VI and PI obtained by the above optimization, the following direct sum space (vector) is created for each batch of training data of model M.

  $$VI_1 \oplus PI_{1,2} \oplus VI_2 \oplus PI_{2,3} \cdots VI_B \oplus PI_{B-1,B} \oplus VI_B$$

  This vector is $L \times 3$ dimensions (L is the number of layers of M), which is the vector space (direct sum space) of the number B. The NIC performs OSVM on this space.

8.5 Computer experiment

In order to confirm the effect of adversarial example detection technology (NIC), the experiment of the paper [76] was retested in the following experimental environment.

- Hardware environment: AIST ABCI [77]
- Datasets: Two common image datasets, MNIST [78] and CIFAR-10 [79], were used for image classification experiments. MNIST is a grayscale image dataset used for handwritten digit recognition, and CIFAR-10 is a color image dataset used for object recognition. For NIC, we also conducted an experiment on LFW (face image) [80].
- Attacks: Non-targeted attacks (FGSM L^2,L^∞), targeted attacks JSMA, and gradient-based attacks (CW L^2) were used to generate adversarial examples. The Cleverhans library [81] was used to implement FGSM and JSMA

First, in order to evaluate the adversarial example detection method based on each of the approaches ① to ③, the published implementation code of LID [73], MagNet [74], and feature squeezing [75] was used to evaluate each paper. Then, follow-up experiments were conducted. As the result, the detection rates reported in each paper were able to be confirmed, and among these three, feature squeezing showed the highest detection rate.

Next, in order to evaluate the adversarial example detection method based on the approach ④, an experiment was conducted using the NIC code implemented in Section 8.4. Table 8.1 to Table 8.3 show the results of adversarial example detection and computational experiments on the MNIST, CIFAR-10, and LFW datasets, respectively. Here, the correct answer rate is the rate at which adversarial examples are input to the classifier (OSVM) described in Section 7.4 and are determined to be adversarial examples. The CNN model used in the experiment is LeNet5, and the OSVM Kernel is RBF (MNIST: γ = 0.1 to 0.27, CIFAR-10: γ = 0.11 to 0.2, LFW: γ = 0.005 to 0.90). In the results of this experiment, high detection performance was confirmed not only for the dataset and attack method reported in the paper [76], but also for the unreported dataset LFW and attack method (FGSM L^∞).

Table 8.1 Adversarial example detection computational experiment results for MNIST dataset

Data Set	Attack	Invariant	Performance	Number of data	Performance reported in the paper [76]
MNIST	FGSM L^2	VI	97%	2800	100%
		PI	98%		84%
		NIC	97%		100%
	FGSM L^∞	VI	98%	2800	—
		PI	98%		—
		NIC	98%		—
	JSMA	VI	100%	280	83%
		PI	100%		100%
		NIC	100%		100%
	CW2	VI	100%	280	95%
		PI	100%		96%
		NIC	100%		100%
	Trojan	VI	100%	3200	100%
		PI	100%		100%
		NIC	100%		100%

Table 8.2 Adversarial example detection computational experimental results for CIFAR-10 dataset

Data Set	Attack	Invariant	Performance	Number of data	Performance reported in the paper [76]
CIFAR-10	FGSM L^2	VI	99%	6400	100%
		PI	99%		52%
		NIC	99%		100%
	FGSM L^∞	VI	100%	6400	—
		PI	100%		—
		NIC	100%		—
	JSMA	VI	97%	320	62%
		PI	95%		100%
		NIC	96%		100%
	CW2	VI	98%	320	88%
		PI	95%		89%
		NIC	96%		100%
	Trojan	VI	100%	3200	100%
		PI	100%		100%
		NIC	100%		100%

Table 8.3 Adversarial example detection computational experiment results for LFW dataset

Data Set	Attack	Invariant	Performance	Number of data	Performance reported in the paper [76]
LFW	FGSM L^2	VI	98%	28222	—
		PI	98%		—
		NIC	98%		—
	FGSM L^∞	VI	100%	2822	—
		PI	100%		—
		NIC	100%		—
	JSMA	VI	100%	280	—
		PI	100%		—
		NIC	100%		—
	CW2	VI	100%	840	—
		PI	100%		—
		NIC	100%		—
	Trojan	VI	100%	3200	—
		PI	100%		—
		NIC	100%		—

8.6 Implementation of the NIC framework

We have implemented a simplified NIC method based on Sections 8.3 and 8.4 in order to conduct the computer experiments for confirming the effectiveness of NIC in Section 8.5. In the simplified implementation, we have found some implementation issues in the original paper [76]. In this section, while clarifying the issues, we reconsider the algorithm in order to construct the *NIC framework* for high detection rates of adversarial examples on the testbed, that is used for creating an environment (attack, defense and detection) to benchmark vulnerability to adversarial examples.

8.6.1 **Overview of the NIC framework**

The NIC framework consists of five parts: taking output from each layer; calculating VI and PI for normal data; calculating VI, PI and NIC for adversarial examples; evaluating OSVM and displaying results. The use case of the NIC framework is shown in Figure 8.2. In addition, the process steps for detecting adversarial examples are shown in Figure 8.3.

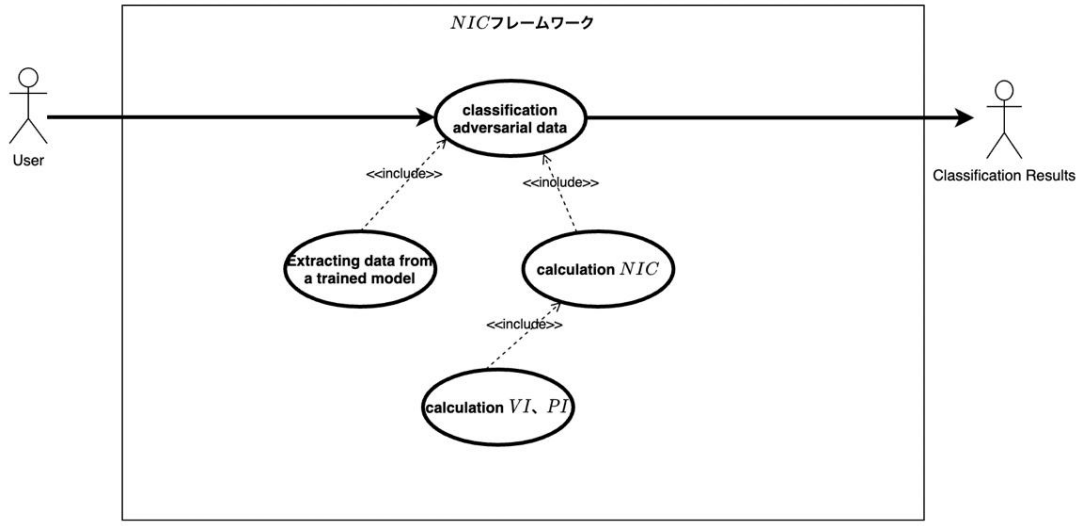


Figure 8.2 NIC framework use cases

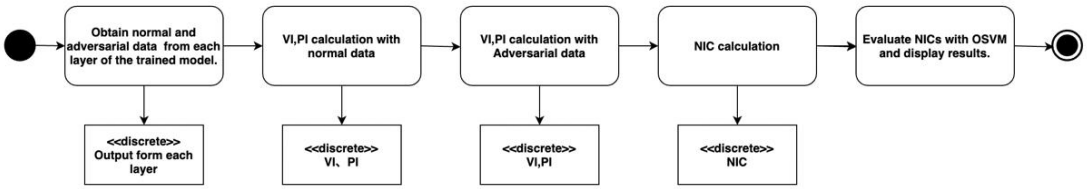


Figure 8.3 Processing procedures for adversarial example detection by the NIC framework

As shown in Figure 8.3, the overall processing procedure for adversarial example detection by the NIC framework consists of five parts. The function of each part (input, processing and output) is shown in Table 8.4.

8.6.2 **Output of OSVM evaluation results**

The NIC framework has been implemented using scikit-learn, that is a Python machine learning library. For example, the scikit-learn's OneClassSVM class is used for implementing the final part of the OSVM as shown in Figure 8.3 as follows.

```
class sklearn.svm.OneClassSVM(array, kernel='rbf', gamma='auto', nu=0.3)
```

Here, the meaning of each argument is as follows.

- array: parameters trained by normal data and used for detecting adversarial examples in NIC.
- kernel: the RBF kernel is used as the algorithm for One Class SVM.
- gamma: the gamma parameter of the RBF kernel is set to 'auto'.
- nu: the upper limit for the percentage of training error and the lower limit for the percentage of support vector are set to 0.3 in this case.

Table 8.4 Functions of the parts comprising the adversarial example detection process procedure

Output extraction from each layer	
input	Normal data (images)
	Adversarial examples (image).
	Trained models, trained on normal data (models trained on normal data)
processing	Obtain the output of each layer of the trained model for normal and adversarial examples and save it in 'numpy in numpy' format.
output (e.g. of dynamo)	Output data from each layer

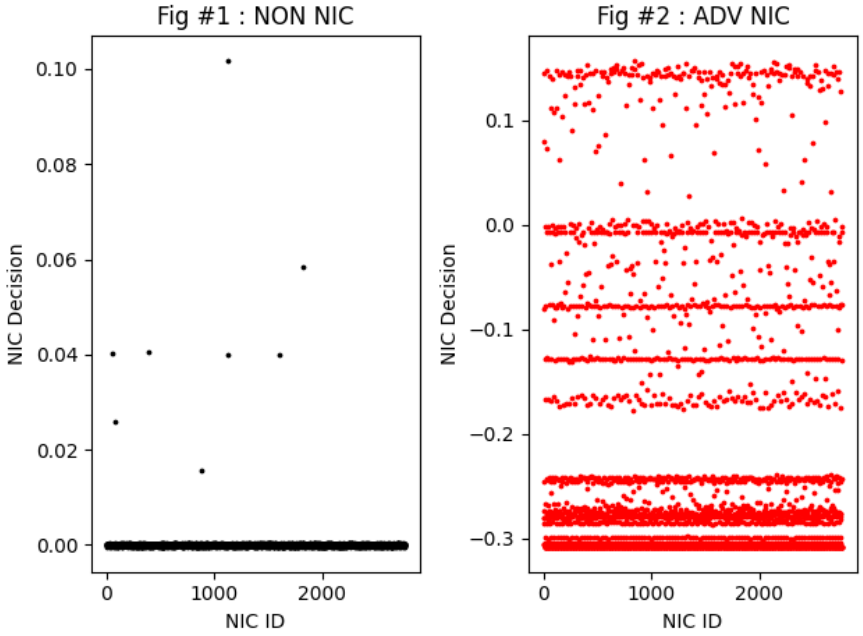
VI and PI calculations for normal data	
input	Output data from each layer of normal data
processing	Calculate VI, PI from the output data of each layer of normal data.
output (e.g. of dynamo)	VI, PI

VI and PI calculations for adversarial examples	
input	Output data from each layer of adversarial examples
	Created at the time of calculation to PI with normal data Derived model of PI
processing	Compute VI, PI from the output of each layer of adversarial examples.
output (e.g. of dynamo)	VI, PI

Calculation of NIC	
input	VI of normal data, PI
	VI of adversarial examples, PI
processing	NIC of normal data is created from VI and PI of normal data and NIC of adversarial examples is calculated from VI and PI of adversarial examples, respectively.
output (e.g. of dynamo)	NIC for normal data, NIC for adversarial examples

Evaluation and display of results in OSVM.	
input	NIC of normal data
	NIC for adversarial examples.
processing	Train OSVM on normal data to create a model, and use this trained model to judge adversarial examples; OSVM uses sk-learn's one class svm API. The judgement results are then displayed.
output (e.g. of dynamo)	Assessment Results

Figure 8.4 shows output values from each layer when one normal data and its adversarial examples are input to the NIC framework, where the horizontal axis is the ID of the model derived to calculate the NIC at each layer (Note. There are multiple outputs from each layer for one image, for example, a convolution layer in CNN), and the vertical axis represents the signed distance of each NIC to the One Class SVM classification hyperplane of the NIC of the normal data, that is the closeness to the normal data in this case. The black dots in Figure 8.4 (a) represent the output relative to the normal data, the red dots in Figure 8.4 (b) are the outputs for adversarial examples. In this calculation, the adversarial examples in Figure 8.4 (b) were generated by using the FGSM L^{∞} attack method.



(a) Normal input data (b) Adversarial input data

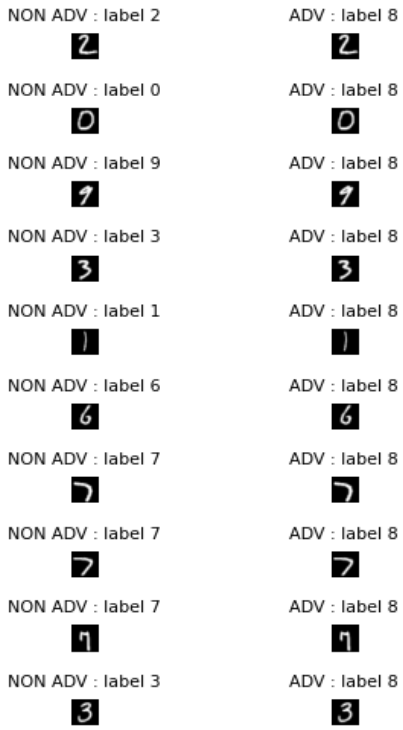
Figure 8.4 Comparison of NIC framework outputs

After training One Class SVM by normal data, One Class SVM function $f(x)$ can be used for detecting adversarial examples such that if $f(x) \geq 0$ then the input x is normal otherwise it is adversarial. Most of the output for normal data are close to zero as shown in Figure 8.4 (a), while approximately 94% of the outputs for adversarial examples are explicitly less than zero as shown in Figure 8.4 (b). This difference of the output between Figure 8.4 (a) and (b) explains that NIC can effectively detect adversarial examples.

8.6.3 **Generation of adversarial examples**

As shown in Figure 8.3, NIC framework does not include the program for generating adversarial examples. We recommend for using CleverHans [82] if adversarial examples are necessary. Figure 8.5 shows some examples in the normal (original) images of handwritten numbers (MNIST) and the adversarial examples generated from the normal images by attack

method FGSM L^2 with the misclassified labels inferred for the adversarial images. As shown in the inference results (label 8) in Figure 8.5 (b), all generated adversarial examples are misclassified as 8.



(a) Original MNIST data (b) Generated adversarial examples

Figure 8.5 Example of adversarial example generation from MNIST (handwritten numbers) images and its decision results

8.6.4 **Reducing calculation costs for VI, PI and VIC**

The calculation method for VI, PI and NIC in the original paper [76] has been explained in Section 8.4, but if the calculation method is used, then the dimension of each data (vector) becomes very large, due to the problem so-called 'dimension demon'. Therefore, we have tried to reduce the dimension as much as possible. In the following section, we explain how each calculation is simplified.

- **Calculation of VI**: in the NIC framework, let $X_B = 1$ for clarifying the correspondence between the input data (both normal and adversarial data) and the VI, PI and NIC (i.e., for the accuracy of the verification). In addition, as all input data are normalized and calculated, the following simplified formula is used:

$$VI_l = f_l \circ f_{l-1} \circ \cdots \circ f_2 \circ f_1.$$

- **Calculation of PI**: as in the case VI above, let $X_B = 1$. Then, the following simplified

formula is used:

$$PL_{l,l-1} = concat(D_l, D_{l-1}) \circ \cdots \circ concat(D_2, D_1).$$

– **NIC calculations**: for dimensionality suppression, X_B is set as follows:

$X_B =$ (The number of layers from which output are obtained)

8.7 Evaluation of the effectiveness of NIC with the Kullback-Leibler divergence

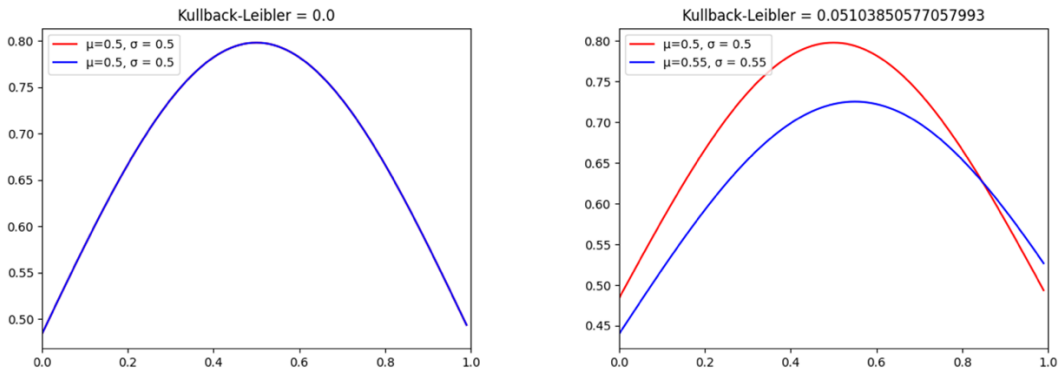
This section reports the results of the evaluation of the effectiveness of the NIC by calculating the degree of divergence between the images of normal and adversarial examples and the NIC by using the Kullback-Leibler divergence.

8.7.1 Kullback-Leibler divergence

The Kullback-Leibler divergence, denoted by $KL(P \parallel Q)$, is a measure of the degree of divergence between two probability distributions P (the probability density functions p) and Q (the probability density function q). The Kullback-Leibler divergence is defined by the following equation.

$$KL(P \parallel Q) = \int p(x) \log \frac{p(x)}{q(x)}$$

The Kullback-Leibler divergence is 0 when the two distributions are the same, and it increases as the divergence increases (the convergence is not guaranteed due to the presence of log). Figure 8.6 shows a simple calculation example of the Kullback-Leibler divergence. In Figure 8.6 (a), both of the distributions P and Q are the same normal distribution whose mean and variance are 0.5 and 0.5, respectively, and then the $KL(P \parallel Q)$ is 0. In Figure 8.6 (b), the means of P and Q are 0.5 and 0.55, and the variance of them are 0.5 and 0.55, respectively, and then the $KL(P \parallel Q)$ is 0.053.



(a) In the case of the same distributions (b) In the case of the different distributions

Figure 8.6 Example of Kullback-Leibler divergence calculation

8.7.2 Kullback-Leibler divergence estimation

The Kullback-Leibler divergence assumes that the probability distributions to be compared are fixed, but in practice, both normal and adversarial data are simply sets of images and the distributions are unknown. Fortunately, a method for approximating the Kullback-Leibler divergence between sets with unknown probability distributions [83] is known. The outline of the approximation method calculates the Kullback-Leibler divergence as a solution of an optimization problem on the following linear polynomial of $r_\theta(x)$ as the constraint for minimizing the density ratio $r(x) = p(x)/q(x)$:

$$r_\theta(x) = \sum_{j=1}^{b} \theta_j \psi_j(x) = \boldsymbol{\theta}^T \boldsymbol{\psi}(x),$$

where $\psi_j(x)$ is the RBF kernel and is defined by

$$\psi_j(x) = \exp\left(-\frac{\|x - x'\|^2}{2h^2}\right),$$

where h is a determinable constant and is the bandwidth.

Then, the Kullback-Leibler divergence can be approximately calculated by the linear polynomial $r_\theta(x)$ obtained as the solution of the optimization problem as follows [83]:

$$KL(P \parallel Q) \sim \frac{1}{n}\sum_{i=1}^{n} \log r(\boldsymbol{x}_i)$$

8.7.3 Effectiveness evaluation of NIC

In Section 8.5, we have shown that the NIC method can effectively detect adversarial examples as anomaly data by experiments. In this section, we show the degree of divergence between normal data and adversarial examples by comparing the Kullback-Leibler divergence of them for explaining the reason why NIC is effective.

At first, Figure 8.7 shows the computational results of the Kullback-Leibler of normal data and adversarial examples (generated by the attack method FGSM L^2) for 50 image data samples, as shown in Figure 8.5. The approximate value of the Kullback-Leibler divergence for the FGSM in Figure 8.7 is 0.46. Here, note that the average value of the multiple Kullback-Leibler divergence is shown in Figure 8.7 because there are multiple values of NIC for one image as explained in Figure 8.4.

Next, Figure 8.8 shows the computational results of the Kullback-Leibler of NIC of the normal data and the adversarial examples used in Figure 8.7. The approximate value of the Kullback-Leibler divergence in Figure 8.8 is 4.47. Therefore, the Kullback-Leibler divergence in Figure 8.8 is about 10 times larger than one in Figure 8.7. We conjecture that the results mean that the perturbations added to normal data can be extracted as more explicit difference by the NIC method.

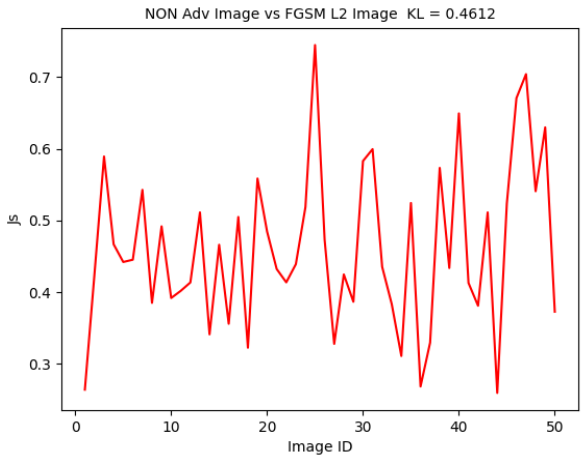


Figure 8.7 The Kullback-Leibler divergence for normal and adversarial examples

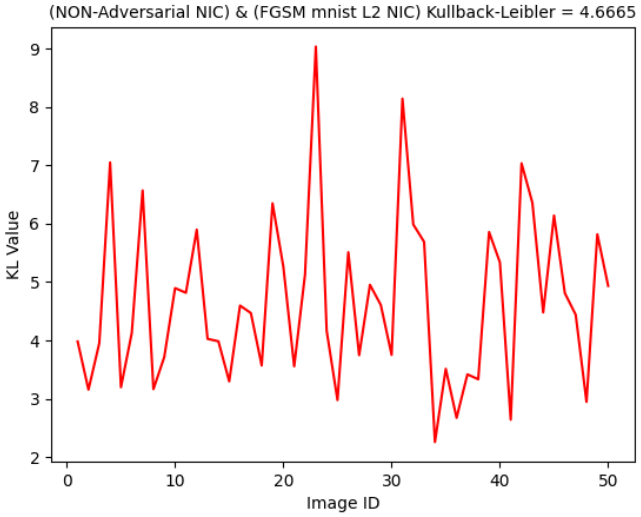


Figure 8.8 The Kullback-Leibler divergence of NIC for normal and adversarial examples

9 AI Quality Management in Operation

In this chapter, we report on the results of a survey on the latest technologies for detecting changes in data distribution over time, called concept drift, and adapting machine learning models to the changed distribution for AI quality management during operation. In addition, we also introduce the results of a survey on the latest unsupervised domain adaptation technologies published at recent international major conferences on machine learning and computer vision for further development of the AI quality management technologies.

Concept drift is one of the main causes of performance degradation of machine learning models running in AI systems during operation. In order to maintain quality that is satisfied at the beginning of the operation of the system throughout the operation period, it is necessary to continuously monitor whether drift occurs or not. In addition, if necessary, we retrain the machine learning models in the system with the latest data to adapt them to the distribution of data changed after the drift occurs. As the use of machine learning technologies has been expanded in recent years, AI systems operating with such technologies will require processing a large amount of data without their true labels (ground truths) in a short period of time, including types of data that have not been handled in the past.

In the fiscal year 2019-2020, we conducted a survey on the latest technologies for detecting and adapting to the concept drift to maintain the performance of machine learning models during operation. As a result of this survey, we found that most of the methods developed so far are supervised methods that use true labels of data additionally acquired during operation for the detection and adaptation. However, such true labels are not always available or are often costly even if they are available. In order to expand the applicability of the detection and adaptation methods and reduce their operational costs, we found that an "unsupervised method" that does not use the true labels or a "semi-supervised method" that uses only a limited number of the true labels is promising. We summarized the results of the surveys organized and discussed from this perspective.

For details on the survey on detection methods, see Section 7.8 of the Machine Learning Quality Management Guidelines [1]. In addition, adaptation methods are summarized in our survey result [84]. Table 9.1 shows the comparison of our survey with the other existing surveys on concept drift detection and adaptation methods. Gama et al. summarized their survey result in [85] and Lu et al. added recently published drift detection and adaptation methods in [86]. Those survey papers mainly focus on introducing "supervised" methods that use true labels of operational data for drift detection and adaptation. On the other hand, Ishida et al. introduced "unsupervised" concept drift detection methods that do not use true labels of data for drift detection in [87]. In comparison with those existing survey results, we introduced "unsupervised" and "semi-supervised" concept drift adaptation methods that do not use or use only a limited number of true labels as mentioned above. Furthermore, we introduced those drift adaptation methods based on the characteristic of each method. In detail, we listed ten remarkable unsupervised/semi-supervised drift adaptation methods and classified them

according to: i) types of drift that can be dealt with effectively, ii) processes where true labels of data are required during operation and the percentage of the labeled data used in verifications shown in the papers, and iii) machine learning models or clustering methods used in each method. Finally, we closed our survey by discussing further development of unsupervised and semi-supervised concept drift adaptation methods using knowledge obtained from relevant unsupervised domain adaptation techniques.

Table 9.1 Comparison of survey papers on concept drift detection and adaptation

	Detection	**Adaptation**
Supervised	Gama et al.[85], Lu et al.[86]	
Unsupervised / Semi-supervised	Ishida et al.[87]	Okawa and Kobayashi [84], [88] (Ours)

In the future operation of AI systems, there is a growing need for new adaptation techniques that do not use the original training data (i.e., source data) to adapt machine learning models from the viewpoint of data privacy and portability in addition to that can deal with changes other than those in the distribution of input data. In particular, adaptation techniques that do not depend on such training data (source data) are called "source-free domain adaptation techniques" or "test-time adaptation techniques (if they adapt online)". These source-free and test-time adaptation technologies have been attracting more attention because they can reduce costs not only on management and transmission of source data for adaptation but also on security for data storage.

In FY2021, following the above-mentioned surveys, we conducted a survey on the latest research trends in unsupervised adaptation techniques to data changes presented at major international conferences in the fields of machine learning and computer vision held in 2019-2021, focusing on unsupervised concept drift adaptation techniques and unsupervised domain adaptation techniques. The result of this survey is summarized in [88]. In detail, we listed and introduced 15 remarkable concept drift detection and unsupervised domain adaptation methods and classified them according to: i) kinds of adaptation problems, ii) kinds of data and labels used in detection and adaptation, iii) availability for adaptation to label shift, and iv) kinds of validation tasks. According to the results of this survey, it is shown that there has been development of the source-free adaptation and test-time adaptation techniques mentioned above and adaptation techniques that are able to adapt to changes other than the distribution of input data, such as label shifts. Furthermore, some techniques have been validated not only for image classification problems, but also for semantic segmentation and object detection problems. These research trends in unsupervised adaptation techniques are expected to solve new problems in AI operations, such as maintaining data privacy, and to be used in various situations in future AI operations.

10 References

Chapter 1:

[1] National Institute of Advanced Industrial Science and Technology (AIST), Machine Learning Quality Management Guideline (3rd English Edition), Digital Architecture Research Center, Cyber Physical Security Research Center, Artificial Intelligence Research Center, Technical Report DigiARC-TR-2023-01/ CPSEC-TR-2023001 , 2023 https://www.digiarc.aist.go.jp/publication/aiqm/

[2] Yuri Miyagi, Masaki Onishi, Machine Learning Model Comparison Visualization Focusing on Worker Information, The 24th Meeting on Image Recognition and Understanding 2021, I31-22, 2021.

[3] Yuri Miyagi, Masaki Onishi, Comparative Visualization Method Focusing on Workers for Evaluation of Machine Learning Models, The 49th Symposium on Visualization, OS12, 2021.

[4] Tomoumi Takase, Dynamic batch size tuning based on stopping criterion for neural network training, Neurocomputing, Volume 429, pp.1-11, 2021.

[5] Shin Nakajima, Software Testing with Statistical Partial Oracles, 10th SOFL+MSVL, 2021.

Chapter 2:

[6] Satoshi Hara, My Bookmark : Interpretability in Machine Learning, Journal of Japanese Society for Artificial Intelligence, vol. 33, no. 3, pp. 366-369, 2018 (in Japanese).

[7] Fred Hohman, Minsuk Kahng, Robert Pienta, Duen Horng Chau, Visual Analytics in Deep Learning: An Interrogative Survey for the Next Frontiers, IEEE Transactions on Visualization and Computer Graphics, vol. 25, no. 8, pp. 2674-2693, 2018.

[8] Bilal Alsallakh, Amin Jourabloo, Mao Ye, Xiaoming Liu, Liu Ren, Do Convolutional Neural Networks Learn Class Hierarchy?, IEEE Transactions on Visualization and Computer Graphics, vol. 24, no. 1, pp. 152-162, 2018.

[9] Mengchen Liu, Jiaxin Shi, Kelei Cao, Jun Zhu, Shixia Liu, Analyzing the Training Processes of Deep Generative Models, IEEE Transactions on Visualization and Computer Graphics, vol.24, no.1, pp.77-87, 2018.

[10] Jorge Piazentin Ono, Sonia Castelo, Roque Lopez, Enrico Bertini, Juliana Freire, Claudio Silva, PipelineProfiler: A Visual Analytics Tool for the Exploration of AutoML Pipelines, IEEE Transactions on Visualization and Computer Graphics, vol.27, no.2, pp.390-400, 2021.

[11] Saleema Amershi, Maya Cakmak, W. Bradley Knox, Todd Kulesza, Power to the People: The Role of Humans in Interactive Machine Learning. AI Magazine, vol.35, no.4, pp.105-120, 2014.

[12] Heungseok Park, Jinwoong Kim, Minkyu Kim, Ji-Hoon Kim, Jaegul Choo, Jung-Woo Ha and Nako Sung, VISUALHYPERTUNER: VISUAL ANALYTICS FOR USER-DRIVEN HYPERPARAMTER TUNING OF DEEP NEURAL NETWORKS, 2019.

Chapter 3:

[13] Gontijo-Lopes, R., Smullin, S. J., Cubuk, E. D., and Dyer, E., Affinity and Diversity: Quantifying Mechanisms of Data Augmentation. arXiv preprint arXiv:2002.08973, 2020.
[14] Cubuk, E. D., Zoph, B., Shlens, J., and Le, Q., RandAugment: Practical Automated Data Augmentation with a Reduced Search Space. In Neural Information Processing Systems, 33, 2020.
[15] Zhang, H., Cisse, M., Dauphin, Y. N., and Lopez-Paz, D., Mixup: Beyond Empirical Risk Minimization. In International Conference on Learning Representations, 2018.
[16] Verma, V., Lamb, A., Beckham, C., Najafi, A., Mitliagkas, I., Lopez-Paz, D., and Bengio, Y., Manifold Mixup: Better Representations by Interpolating Hidden States. In International Conference on Machine Learning, pp. 6438–6447, PMLR, 2019.
[17] Kim, J-H., Choo, W., and Song, H. O., Puzzle mix: Exploiting saliency and local statistics for optimal mixup. In International Conference on Machine Learning, 2020.
[18] Beckham, C., Honari, S., Verma, V., Lamb, A., Ghadiri, F., Hjelm, R. D., Bengio, Y., and Pal, C. On adversarial mixup resynthesis. In Neural Information Processing Systems, 2019.

Chapter 4:

[19] Nakajima, S., Quality Issues in Machine Learning from Software Engineering Viewpoints, Maruzen Publisher, 2020. (in Japanese)
[20] Pei, K., et al., DeepXplore: Automated Whitebox Testing of Deep Learning Systems, In Proc. 26th SOSP, 2017, pp.1-18.
[21] Nakajima, S., Distortion and Faults in Machine Learning Software, In Post-Proc. 9th SOFL+MSVL, 2020, pp.29-41.
[22] Ma, L., et al., DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems, In Proc. ASE, 2018, pp.120-131.
[23] Tian, Y., et al., DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars, In Proc. 40th ICSE, 2018, pp.303-314.
[24] Zhang, M., et al., DeepRoad: GAN-Based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems, In Proc. ASE, 2018, pp.132-142.
[25] Zhang, P, et al., CAGFuzz: Coverage-Guided Adversarial Generative Fussing Testing of Deep Learning Systems, arXiv:1911.07931, 2019.
[26] Harel-Canada, F., et al., Is Neuron Coverage a Meaningful Measure for Testing Deep Neural Networks? In ESEC/FSE, 2020, pp.851-862.
[27] Kim, J. et al., Guiding Deep Learning System Testing Using Surprise Adequacy, In Proc. 41st ICSE, 2019, pp.1039-1049.

Chapter 5:

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning*, The MIT Press 2016.

[29] Simon Haykin, *Neural Networks and Learning Machines (3ed.)*, Pearson India 2016.

[30] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Anath Grama, MODE: Automated Neural Network Model Debugging via State Differential Analysis and Input Selection, In Proc. 26th ESE/FSE, pp.175-186, 2018.

[31] Shin Nakajima, Software Testing with Statistical Partial Oracles – Applications to Neural Network Software, In Proc. 10th SOFL+MSVL, pp.275-192, 2021.

[32] Shin Nakajima and Tsong Yueh Chen, Generating Biased Dataset for Metamorphic Testing of Machine Learning Programs, In Proc. 31st ICTSS, pp.56-64, 2019.

[33] Gregor Montavon, Genevieve B. Orr, and Klaus-Robert Muller (eds.), *Neural Networks: Tricks of the Trade (2ed.)*, Springer 2012.

[34] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov, Membership Inference Attacks Against Machine Learning Models, arXiv:1610.05820v2, 2017.

[35] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha, Privacy Risk in Machine Learning: Analyzing the Connection to Overfitting, arXiv:1709.01604v5, 2018.

[36] Yunhui Long, Vincent Bindschaedler, Lei Wang, Diyue Bu, Xiaofeng Wang, Haixu Tang, Carl A. Gunter, and Kai Chen, Understanding Membership Inferences on Well-Generalized Learning Models, arXiv:1802.04489, 2018.

[37] Charu C. Aggarwal, *Outlier Analysis (2ed.)*, Springer 2017.

[38] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer, Replux: An Efficeint SMT Solver for Verifying Deep Neural Networks, In Proc. 29th CAV, pp.97-117, 2017.

[39] Pang Wei Koh and Percy Liang, Understanding Black-box Predictions via Influence Functions, arXiv:1703.04730v3, 2020.

[40] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana, DeepXplore: Automated Whitebox Testing of Deep Learning Systems, In Proc. 26th SOSP, pp.1-18, 2017.

[41] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems, In Proc. 33rd ASE, pp.120-131, 2018.

[42] Yizhen Dong, Peixin Zhang, Jingyi Wang, Shuang Liu, Jun Sun, Jianye Hao, Xinyu Wang, Li Wang, Jin Song Dong, and Dai Ting. There is Limited Correlation between Coverage and Robustness for Deep Neural Networks. arXiv:1911.05904, 2019.

[43] Fabrice Harel-Canada, Lingxiao Wang, Muhammad Ali Gulzar, and Miryung Kim, In Proc. 28th ESEC/FSE, pp.851-862, 2020.

[44] Shin Nakajima, Distortion and Faults in Machine Learning Software, In Proc. 9th SOFL+MSVL, pp.29-41, 2019.

[45] Stephanie Abrecht, Maram Akila, Sujan Sai Gannamaneni, Konrad Groh, Christian Heinzemann, Sebastian Houben, and Matthjas Woehrle, Revisiting Neuron Coverage and Its Application to Test Generation, In Proc. SAFECOMP 2020 Workshop, pp.289-301, 2020.

[46] National Institute of Advanced Industrial Science and Technology (AIST), Machine Learning Quality Management Guideline (1st English Edition), Chapter 4, Digital Architecture Research Center, Cyber Physical Security Research Center, Artificial Intelligence Research

Center, Technical Report DigiARC-TR-2022-01/ CPSEC-TR-2022002.
https://www.digiarc.aist.go.jp/publication/aiqm/

Chapter 6:

[47] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus, Intriguing properties of neural networks, The International Conference on Learning Representations (ICLR 2014), pp.1-10, 2014. https://arxiv.org/abs/1312.6199

[48] Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer, Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks, International Conference on Computer-Aided Verification (CAV), 2017. https://arxiv.org/abs/1702.01135

[49] Vincent Tjeng, Kai Xiao, and Russ Tedrake, Evaluating robustness of neural networks with mixed integer programming, International Conference on Learning Representations (ICLR), 2019. https://arxiv.org/abs/1711.07356

[50] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Duane Boning, Inderjit S. Dhillon, and Luca Daniel, Towards Fast Computation of Certified Robustness for ReLU Networks, International Conference on Machine Learning, PMLR 80, pp.5276-5285, 2018. https://arxiv.org/abs/1804.09699

[51] Akhilan Boopathy, Tsui-Wei Weng, Pin-Yu Chen, Sijia Liu, and Luca Daniel, CNN-Cert: An Efficient Framework for Certifying Robustness of Convolutional Neural Networks, The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI 2019), pp.3240-3247, 2019. https://arxiv.org/abs/1811.12395

[52] Tsui-Wei Weng, Pin-Yu Chen, Lam Nguyen, Mark Squillante, Akhilan Boopathy, Ivan Oseledets, and Luca Daniel, PROVEN: Verifying Robustness of Neural Networks with a Probabilistic Approach, International Conference on Machine Learning (ICML 2019), PMLR vol. 97, pp.6727-6736, 2019. http://proceedings.mlr.press/v97/weng19a.html

[53] Nicholas Carlini and David Wagner, Towards Evaluating the Robustness of Neural Networks, IEEE Symposium on Security and Privacy (SP), pp.39-57, 2017. https://arxiv.org/abs/1608.04644

[54] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel, Evaluating the Robustness of Neural Networks: An Extreme Value Theory Approach, International Conference on Learning Representations (ICLR 2018), 2018. https://arxiv.org/abs/1801.10578

[55] Eric Wong and J. Zico Kolter, Provable defenses against adversarial examples via the convex outer adversarial polytope, International Conference on Machine Learning (ICML 2018), PMLR vol. 80, pp.5283-5292, 2018. https://arxiv.org/abs/1711.00851

[56] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana, Certified Robustness to Adversarial Examples with Differential Privacy, The IEEE Symposium on Security and Privacy (SP), 2019. https://arxiv.org/abs/1802.03471

[57] Jeremy M Cohen, Elan Rosenfeld, and J. Zico Kolter, Certified Adversarial Robustness via

Randomized Smoothing, The 36th International Conference on Machine Learning (ICML 2019), 2019. https://arxiv.org/abs/1902.02918

[58] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu, Towards Deep Learning Models Resistant to Adversarial Attacks, The Sixth International Conference on Learning Representations (ICLR 2018), 2018. https://arxiv.org/abs/1706.06083

Chapter 7:

[59] Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio, Fantastic Generalization Measures and Where to Find Them, International Conference on Learning Representations (ICLR 2020). https://arxiv.org/abs/1912.02178

[60] Gintare Karolina Dziugaite, Alexandre Drouin, Brady Neal, Nitarshan Rajkumar, Ethan Caballero, Linbo Wang, Ioannis Mitliagkas, and Daniel M. Roy, In search of robust measures of generalization, NeurIPS 2020. arXiv:2010.11924. https://arxiv.org/abs/2010.11924

[61] Konstantinos Pitas, Mike Davies, and Pierre Vandergheynst, PAC-Bayesian Margin Bounds for Convolutional Neural Networks, arXiv:1801.00171, 2018. https://arxiv.org/abs/1801.00171

[62] Ilja Kuzborskij and Christoph H. Lampert, Data-Dependent Stability of Stochastic Gradient Descent, 2017. arXiv:1703.01678. https://arxiv.org/abs/1703.01678

[63] Andrew Y. K. Foong, Wessel P. Bruinsma, David R. Burt, and Richard E. Turner, How Tight Can PAC-Bayes be in the Small Data Regime? Neural Information Processing Systems (NeurIPS), 2021. https://arxiv.org/abs/2106.03542

[64] Guillermo Valle-Pérez and Ard A. Louis, Generalization bounds for deep learning, arXiv:2012.04115v2, 2020. https://arxiv.org/abs/2012.04115

[65] Saurabh Garg, Sivaraman Balakrishnan, J. Zico Kolter, and Zachary C. Lipton, RATT: Leveraging Unlabeled Data to Guarantee Generalization, ICML 2021. arXiv:2105.00303. https://arxiv.org/abs/2105.00303

[66] Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P. Adams, and Peter Orbanz, Non-vacuous Generalization Bounds at the ImageNet Scale: a PAC-Bayesian Compression Approach, ICLR 2019. https://arxiv.org/abs/1804.05862

[67] Gintare Karolina Dziugaite and Daniel M. Roy, Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters than Training Data, Thirty-Third Conference on Uncertainty in Artificial Intelligence (UAI), 2017. arXiv:1703.11008. https://arxiv.org/abs/1703.11008

[68] María Pérez-Ortiz, Omar Rivasplata, John Shawe-Taylor, and Csaba Szepesvári, Tighter risk certificates for neural networks, Journal of Machine Learning Research, 2021. arXiv:2007.12911. https://arxiv.org/abs/2007.12911

[69] Shai Shalev-Shwartz and Shai Ben-David, Understanding Machine Learning: From Theory to Algorithms, Cambridge University Press, 2014. https://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/index.html

[70] Oliver Catoni, PAC-Bayesian Supervised Classification: The Thermodynamics of Statistical Learning, Institute of Mathematical Statistics, Lecture Notes-Monograph Series, vol. 56, 2007. https://www.jstor.org/stable/i20461497

[71] Andreas Maurer, A Note on the PAC Bayesian Theorem, arXiv:cs/0411099, 2004. https://arxiv.org/abs/cs/0411099

[72] John Langford, Tutorial on Practical Prediction Theory for Classification, JMLR, vol.6, No.10, pp.273–306, 2005. https://jmlr.org/papers/v6/langford05a.html

Chapter 8:

[73] X. Ma, Characterizing adversarial subspaces using Local Intrinsic Dimensionality, 2018.

[74] D. Meng , Magnet: a two-pronged defense against adversarial examples, in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017.

[75] W. Xu, Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks, in Proceedings of the 2018 Network and Distributed Systems Security Symposium (NDSS), 2018.

[76] Shiqing Ma, NIC: Detecting Adversarial Samples with Neural Network Invariant Checking, Network and Distributed Systems Security Symposium (NDSS), NDSS 2019.

[77] National Institute of Advanced Industrial Science and Technology (AIST), AI Bridging Cloud Infrastructure, https://abci.ai/ja/

[78] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE, vol. 86, no. 11, pp.2278–2324, 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[79] A. Krizhevsky and G. Hinton, Learning multiple layers of features from tiny images, 2009.

[80] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, Labeled faces in the wild: A database for studying face recognition in unconstrained environments, University of Massachusetts, Amherst, Tech. Rep. 07-49, October 2007.

[81] Nicolas Papernot, Ian Goodfellow, Ryan Sheatsley, Reuben Feinman, and Patrick McDaniel. cleverhans v1.0.0: an adversarial machine learning library. arXiv preprint arXiv:1610.00768, 2016.

[82] CleverHans, https://github.com/cleverhans-lab/cleverhans

[83] Masashi Sugiyama, Taiji Suzuki, and Takafumi Kanamori, Density Ratio Estimation in Machine Learning, Cambridge University Press, 2012.

Chapter 9:

[84] Yoshihiro Okawa and Kenichi Kobayashi, A Survey on Concept Drift Adaptation Technologies for Unlabeled Data in Operation, *Proceedings of the 35th Annual Conference of the Japanese Society for Artificial Intelligence*, pp.1-4, 2021 (in Japanese), https://doi.org/10.11517/pjsai.JSAI2021.0_2G4GS2f03.

[85] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia,

A survey on concept drift adaptation, *ACM Computer Surveys*, vol. 46, no. 4, pp.1-37, 2014.

[86] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, João Gama, and Guangquan ZhangJ, Learning under Concept Drift: A Review, in *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346-2363, 2019.

[87] Tsutomu Ishida, Hiroaki Kingetsu, Yasuto Yokota, Yoshihiro Okawa, Kenichi Kobayashi, and Katsuhito Nakazawa, Evaluation of Concept Drift Detection Methods for Unlabeled Data in Operation, *Proceedings of the 34th Annual Conference of the Japanese Society for Artificial Intelligence*, pp.1-4, 2020 (in Japanese), https://doi.org/10.11517/pjsai.JSAI2020.0_4Rin105.

[88] Yoshihiro Okawa and Kenichi Kobayashi, Recent Research Trends in Unsupervised Adaptation Techniques for Data Changes, *Proceedings of the 36th Annual Conference of the Japanese Society for Artificial Intelligence*, pp.1-4, 2022 (in Japanese), https://doi.org/10.11517/pjsai.JSAI2022.0_3Yin240.