Technical Report on Machine Learning Quality Evaluation and Improvement

1st English Edition

February 4, 2022
(a revision of the Japanese version published on July 5, 2021)

Technical Report DigiARC-TR-2022-02
Digital Architecture Research Center
National Institute of Advanced Industrial Science and Technology (AIST)

Technical Report CPSEC-TR-2022003
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 two years (FY 2019~2020).

Table of Contents

1 Introduction

Machine Learning Quality Management Guideline has been developed to clearly explain the quality of various industrial products using statistical machine learning (Version 2 [1]). The second edition of the guideline focuses on the nine internal quality characteristics for machine learning systems (Stability of the trained model, Reliability of underlying software system, etc.), 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, the number in each box shows the chapter number explained in this report) that were researched and developed for the recent two years (FY 2019~2020). 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 ~ 8.

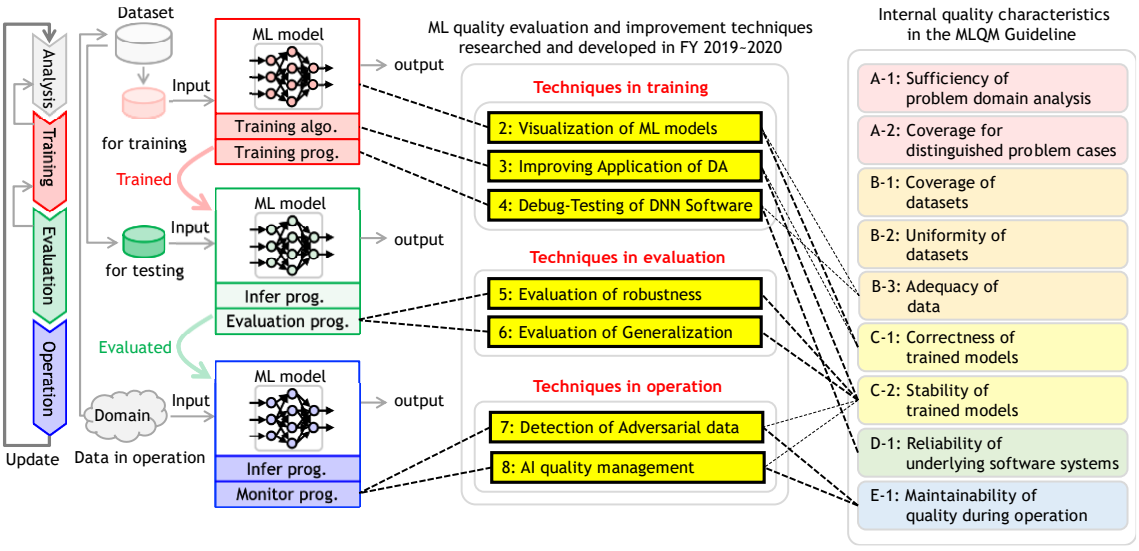

Figure 1.1 Machine learning quality evaluation and improvement techniques in this report

- Visualization of Machine Learning Models (in Chapter 2) :
  To visualize machine learning models from the two points of views, (1) comparison results between multiple models and (2) the sensitivity of workers (e.g., annotators of training data, designers of model structures) reflected in the models, we report the results of a survey on such recent visualizations and our prototype visualization tool for model comparison.

- Improving Application of Data Augmentation in Deep Learning (in Chapter 3):
  To improve the data-diversity obtained by data augmentation and increase accuracy and stability in deep learning, we devise 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 [2].

- Debug-Testing of DNN Software (in Chapter 4):
  To test the presence or absence of defects in training/learning programs, we derive a light-weight index from the internal index (neuron coverage) in trained DNN models, and then report the results of several experiments and discuss the usefulness of the derived index [3].

- Evaluation and Improvement of Robustness (in Chapter 5):
  To evaluate and improve robustness of machine learning 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.

- Estimation of Generalization Error Upper Bounds (in Chapter 6):
  To evaluate the generalization performance of machine learning models, we report on the results of a survey on methods for estimating the upper bound on the expected value of the error rate (i.e., generalization error) for all inputs, including unseen samples.

- Adversarial Example Detection (in Chapter 7):
  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.

- AI Quality Management in Operation (in Chapter 8):
  To maintain quality of machine learning models 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). The results include not only supervised methods, but also unsupervised/semi-supervised methods.

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: Yoshinao Isobe (AIST CPSEC)
- Chapter 6: Yoshinao Isobe (AIST CPSEC)
- Chapter 7: Yusei Nakashima (Techmatrix)
- Chapter 8: 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

This chapter first explains the results of a survey of recent machine learning model visualization methods, and analysis about workers using models. Then we report on an example of a prototype visualization tool for model comparison developed in 2020, and future implementation plans.

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 [4], 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. [5] 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 ([6]) or model structure ([7]). 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 [8], 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 [9]. 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.

Based on the above survey results, we emphasized "comparative visualization of multiple models" and "visualization of worker sensitivity" among model visualization methods and

proceeded to design a visualization tool that has both properties.

2.2 Developing prototype of model difference visualization tool

Based on the results of the survey described above, we proceeded with the study and trial production of a visualization tool specialized for the comparison of multiple models. 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 2.1 shows the overview of prototype visualization views, which visualize the results of MNIST training and output for two simple models.

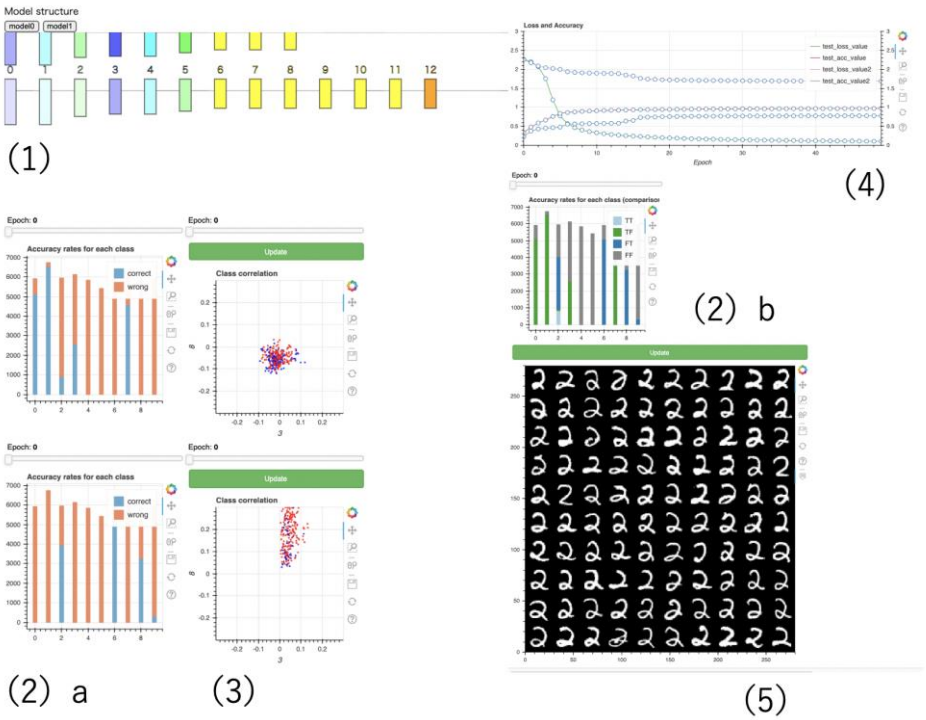

Figure 2.1 The whole image of the prototype visualization views.

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 2.2 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 2.2, 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 2.2, 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.

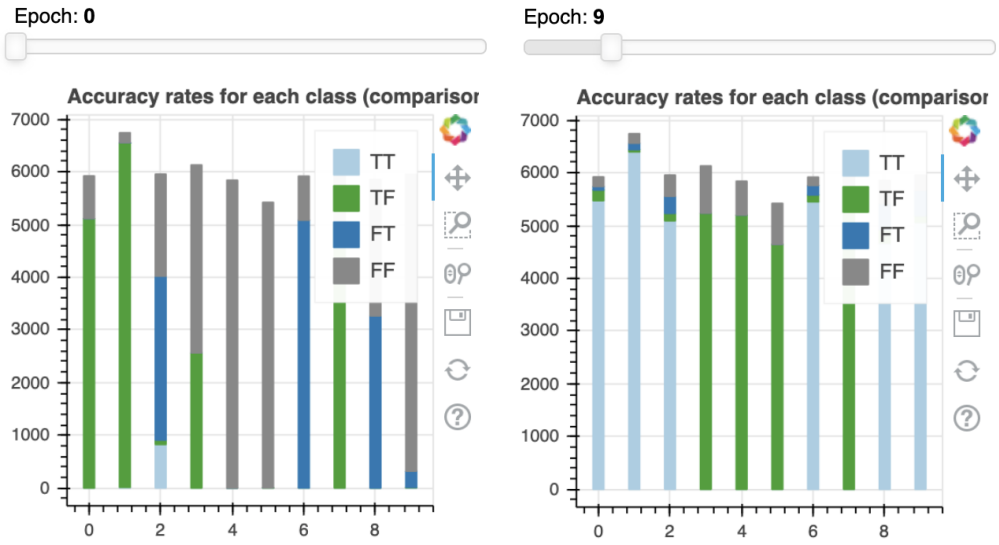

Figure 2.2 Example of comparing the output results of two models

2.3 Future work

In addition to the view shown in Figure 2.1, we will add a view for visualizing the sensibility of workers. Specifically, we are working on the implementation of (1) time-series visualization of the work content and accuracy changes in the training process, and (2) visualization of the degree of effect to the model by the annotator and model designer. For the time being, we will try to collect and visualize the information of workers based on logs of training and adjustment tasks for models, so that the input in interactive operations, which would increase the burden on users, will not be excessive. In addition, for each visualization view, including the two that we plan to add, we would like to enhance the coordination function and implement expressions that draw attention to important differences.

3 Improving Application of Data Augmentation in Deep Learning

In this study, we developed a new data augmentation method, which is widely used in both basic and applied situations where deep learning is used, and verified its impact on the quality of deep learning through experiments using benchmark datasets for multi-class classification.

3.1 Purpose

Data augmentation is a technique to increase the number of samples by adding transformations to those, and it is highly effective in deep learning, which has a tendency to reduce performance when the number of training samples is small. On the other hand, the effectiveness of data augmentation strongly depends on data to be used, so data augmentation methods and the parameters of each method must be selected appropriately. However, the theoretical analysis of data augmentation is difficult, and the appropriate way to use it has not yet been established, so it is often used empirically (or conventionally). This leads to inappropriate use, which impairs the quality of learning.

Therefore, in order to facilitate a move away from the empirical use of data augmentation, this study focuses on data diversity. Increasing diversity is an intrinsic goal of data augmentation, and investigating its impact is expected to contribute to a better understanding of data augmentation and its development. It has been demonstrated in the work of [10] that the increase in data diversity due to data augmentation has a significant effect on the improvement of generalization performance. In this study, we propose a new augmentation method to increase data diversity and verify its effectiveness. Recently, a method called RandAugment [11], which dynamically applies randomly selected operations from multiple data augmentation operations during training, has been attracting attention. While this method greatly improves diversity, it is difficult to use effectively because many hyperparameters need to be adjusted. In this study, we devised two methods with simple algorithms (FC-mixup in Section 3.2 and Latent DA in Section 3.3). Although the work in [10] also proposes a metric for evaluating diversity, verification using this metric is our future work, and we only investigate its impact on generalization performance in the experiments in this study.

In addition, data augmentation is sometimes applied to the input data prior to training, but in deep learning, data augmentation can be applied each time a sample is input to the model during training, so it is possible to devise various ways of applying it. The methods in Sections 3.2 and 3.3 make good use of this property and apply data augmentation dynamically to the data obtained in hidden layers of neural networks.

3.2 Feature combination mixup（FC-mixup）

Mixup [12] is a versatile data augmentation method that can be applied not only to image data but also to tabular data. It is a method that generates a new sample by linearly interpolating

two samples using the same ratio of linear interpolation for each of the input values and labels, as shown in the following equation.

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

Here, (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 rate sampled from the beta distribution. A modification of this method that can be used in hidden layers of neural networks is called Manifold mixup [13], but both methods have the disadvantage that they generate samples only in a local area of the data distribution, on a line segment between two points, and are inappropriate for datasets with distributions in which the properties of the points on the line segment vary nonlinearly.

Feature combination mixup (FC-mixup) proposed in this study is a method of mixing samples in a different way from the conventional mixup, and its overview is shown in Figure 3.1. Here, suppose that two samples A and B in the same batch output features Z_{A} and Z_B in a randomly selected layer. d is the total number of features in the layer, and FC-mixup randomly extracts and combines $d\lambda$ features from Z_{A} and $d(1-\lambda)$ features from Z_B to generate a new sample Z_{X}. Since a large number of combinations are possible for a single value of λ, different data can be generated depending on the random number, and thus a wide range of samples can be generated on the data distribution. FC-mixup is expressed as follows, so we mix Z_{A} and Z_B so that this equation is satisfied.

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

This technique of generating new data by combining the parts of two samples is also seen in Puzzle Mix [14], but it is limited to the input image. A similar technique is used in Adversarial mixup resynthesis [15], but it is limited to autoencoders, while FC-mixup is designed for more general use. In this study, we proposed Hybrid1 and Hybrid2, which are a fusion of FC-mixup and Manifold mixup [13], to further increase the diversity of the generated data. Hybrid1 is a method that uses either mixup with a probability of 0.5 for each batch, and Hybrid2 is a method that realizes both methods simultaneously.

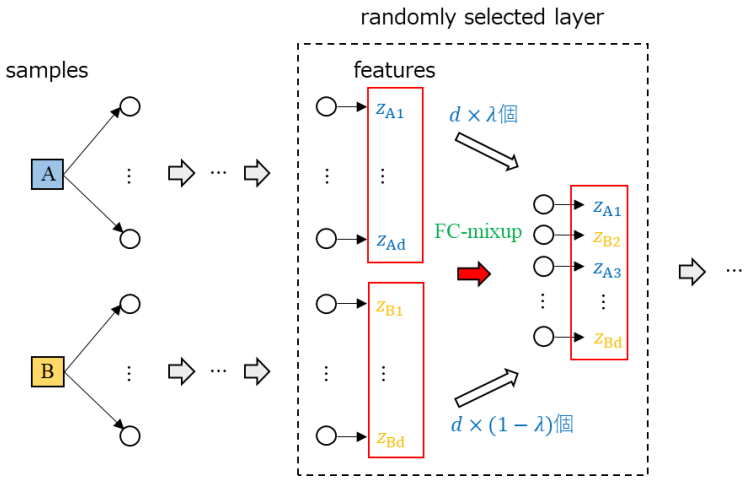

Figure 3.1 Overview of FC-mixup

In order to investigate the performance of FC-mixup, we used several multi-class classification datasets and compared the test accuracy between the conventional method (no data augmentation, mixup at the input layer [12], Manifold mixup [13]) and the proposed method (FC-mixup, Hybrid1, Hybrid2). The model includes WideResNet28-10, ResNet50, multilayer perceptron (MLP), and small size convolutional neural network (small CNN) were used as models. In addition to full datasets, experiments were also conducted on reduced data, which was randomly extracted from 1,000 samples. From the results in Table 3.1, we can see that the proposed method shows the highest accuracy in most cases. Although some datasets show lower accuracy than Manifold mixup (ImageNet, reduced CIFAR-10), using FC-mixup as one of available methods may lead to the improvement of the accuracy.

Table 3.1 Test accuracy for multi-class classification data (underline indicates the proposed method)

	CIFAR-10 WRN-28-10	CIFAR-100 WRN-28-10	Core SVHN WRN-28-10	Car Evaluation MLP	Epileptic Seizure MLP	Letter Recognition MLP
Default	96.65	81.15	96.73	92.42	39.18	89.98
Input	96.97	83.29	97.10	92.79	46.19	91.01
Manifold	97.16	83.90	97.36	92.98	45.17	91.23
FC	96.81	**84.15**	97.36	**93.87**	**46.71**	91.16
Hybrid1	**97.19**	83.85	97.49	93.42	45.44	91.18
Hybrid2	96.94	83.88	**97.66**	93.57	46.45	**91.42**
	Full SVHN **WRN-28-10**	**Tiny ImageNet** **ResNet50**	**ImageNet** **ResNet50**	**Reduced MNIST** **small CNN**	**Reduced CIFAR-10** **small CNN**	**Reduced SVHN** **small CNN**
Default	98.39	62.16	76.54	97.37	43.88	77.87
Input	98.59	65.94	77.14	97.47	44.45	73.51
Manifold	98.60	67.21	**77.26**	97.91	**46.93**	75.55
FC	98.36	66.76	76.81	97.92	45.64	**78.60**
Hybrid1	**98.61**	65.97	76.46	97.93	46.05	77.25
Hybrid2	98.59	**67.55**	77.06	**97.99**	45.59	77.57

3.3 Applying data augmentations to feature maps (Latent DA)

Data augmentation is usually applied to the input data, but in neural networks, it is possible to apply it to outputs in hidden layers. Manifold mixup [13] actually realizes this, but it is specialized for mixup [12] and cannot handle other operations. Since features are extracted hierarchically in CNNs so that they become more complex as they approach the output layer, the effect of data augmentation in various layers chosen randomly for each batch is different and produces a variety of samples. It is also easy to implement, because data augmentation can be applied to feature maps in the same way it is applied to the input image.

Examples of actual application of masking and translation to input images and feature maps are shown in Figure 3.2. Here, samples are input to the model under training, and the images immediately after applying data augmentation set to the same hyperparameters (mask position and displacement) in different layers are displayed in the top row with their sizes aligned. The feature maps of the sample at the last layer before fully connected layers are shown in the bottom row, and their images are different depending on the layer where data augmentation was applied. This result shows that applying data augmentation in various layers leads to an increase in the

diversity of the generated data, and that the effect of such a training is different from that when data augmentation is applied only to the input data.

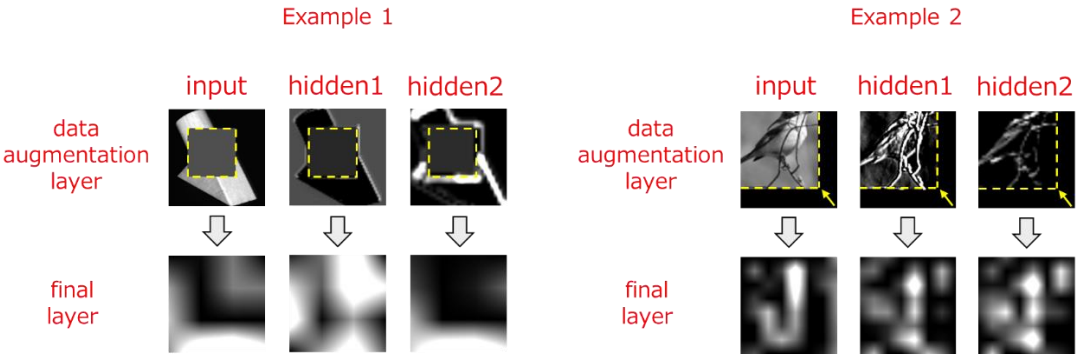

Figure 3.2 Examples of applying data augmentation to input images and feature maps obtained at hidden layers.

In order to compare the performance of data augmentation in the input layer with that in feature maps, we used various data augmentation operations and computed the test accuracy of the supervised trained models. Here, we trained WideResNet28-10 on CIFAR-10, Fashion-MNIST, and SVHN (without extra data) datasets for 200 epochs. The results are shown in Figure 3.3. 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 gives higher accuracy even in the case where the conventional method gives lower accuracy, such as the result using crop. These results show that diverse samples generated by applying data augmentation to random layers are effective in improving the performance.

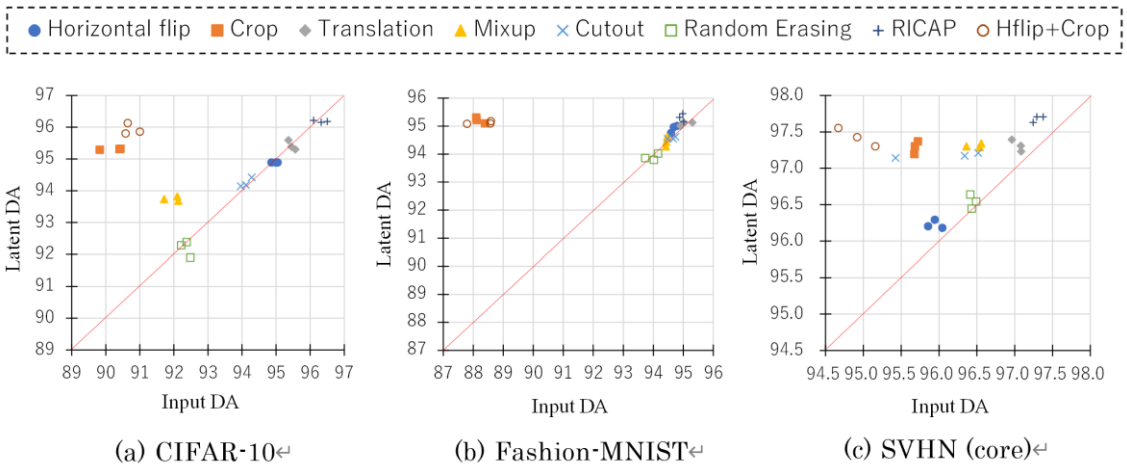

Figure 3.3 Comparison of test accuracy between Input DA and Latent DA

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 [16]. 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 [17], 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.

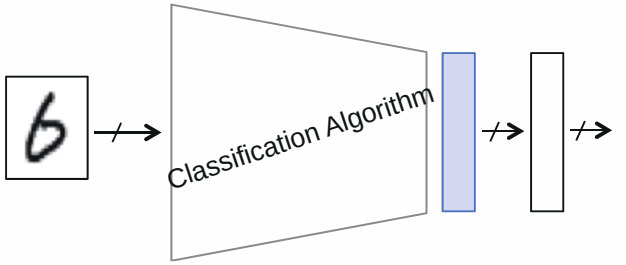

Figure 4.1 Trained DNN model.

In this chapter, NC is assumed to be used as an internal index [18] 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 [17].

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.

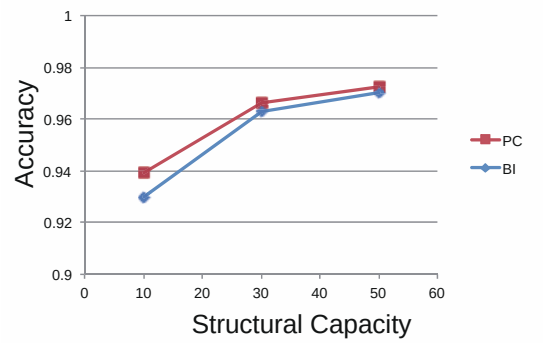

Figure 4.2 Learning models of different capacities.

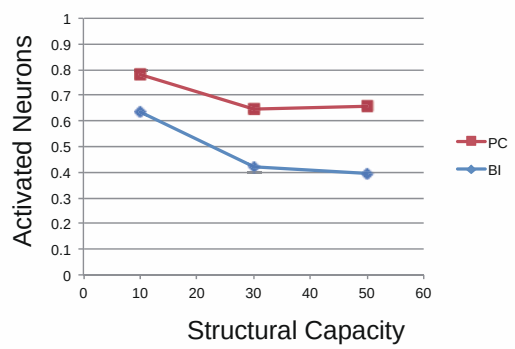

Figure 4.3 Relationship with internal indices

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.

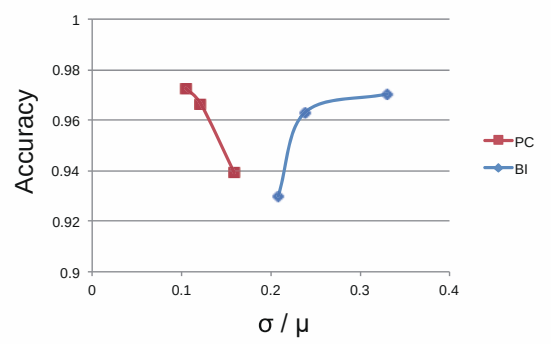

Figure 4.4 Derived index

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.

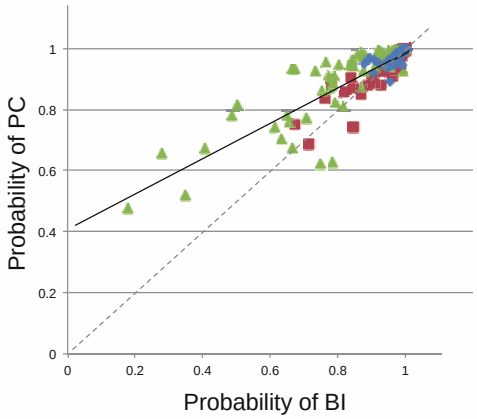

Figure 4.5 Classification certainty for corrupted data.

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).

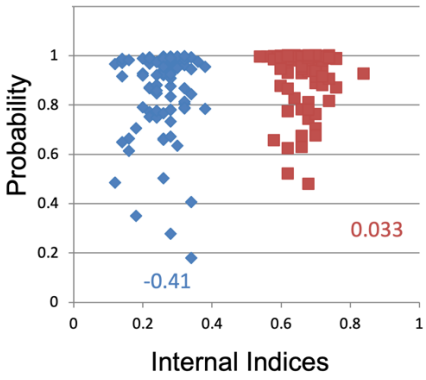

Figure 4.6 Differences in internal indices

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).

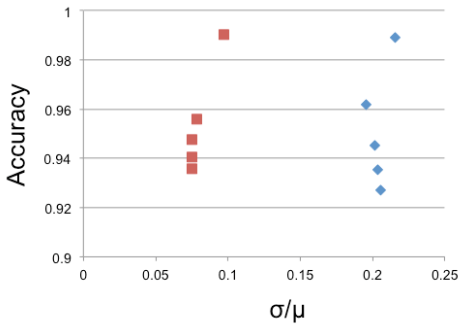

Figure 4.7 Differences in training datasets.

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 [17] 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 [19], 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 [20]. The classical data augmentation method can be

replaced by other approaches such as test input generation based on GAN [21]. Test generation method using GAN with a help of NC is reported in [22]. 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 [20][21] use the calculated steering angle as the oracle. There is also a research work [23] 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 [23] 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 [23], 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 [16][18]. 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. [24] 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 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.

5.1 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 [25], 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.

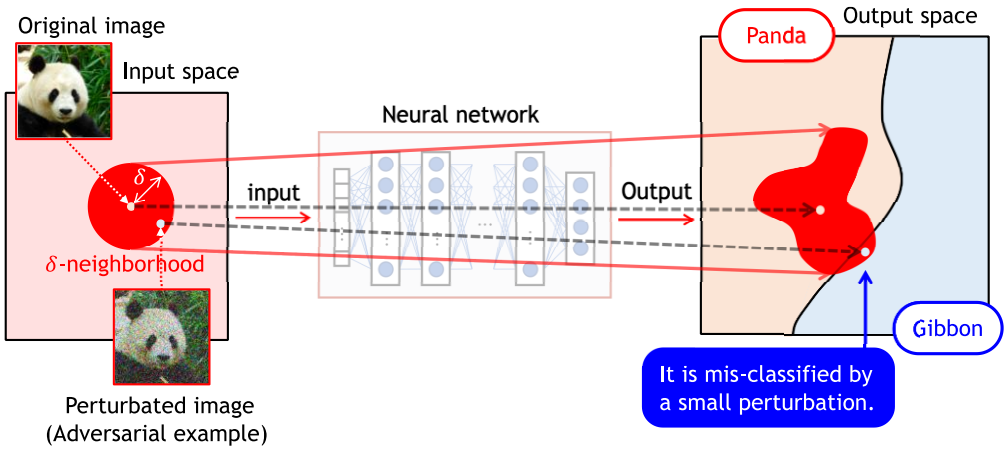

Figure 5.1 An adversarial example from an image of a panda, which is mis-classified into a gibbon

Adversarial examples are explained by Figure 5.1. The left side in Figure 5.1 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 5.1 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 5.2 is the maximum safe radius because all the input images inside the δ-neighborhood in Figure 5.2 are correctly classified.

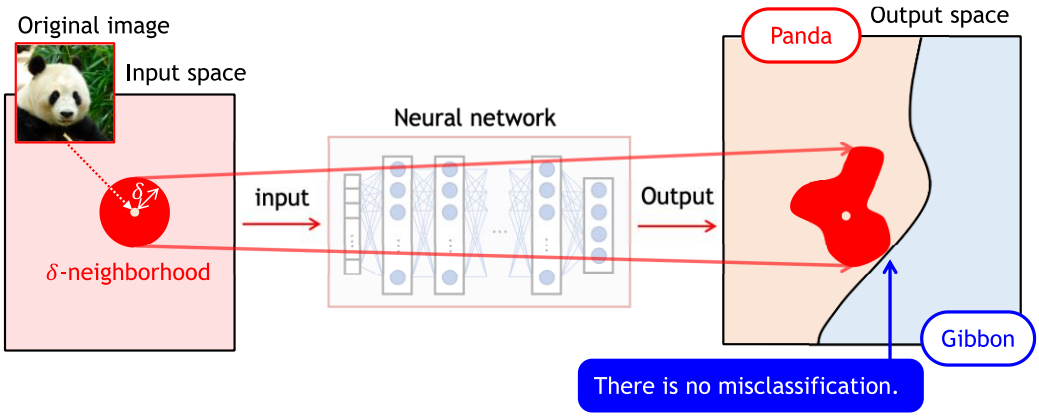

Figure 5.2 The maximum safe radius δ

5.2 A survey on methods for evaluation and improvement of robustness

Table 5.1 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 5.1 is categorized by the following perspectives:

Table 5.1 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) [26] ACAS-XU-DNN, 300 ReLU nodes 6 hidden layers, (Limitation: hundreds of nodes) Tjeng et al. 2019 [27] 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) [28] CIFAR, 6-layer, 12,288 ReLU units About 10,000 times faster than Reluplex Boopathy et al. 2019 (CNN-Cert)[29] 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 [33] 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) [30] CIFAR, 5-layer, CNN, ReLU almost same as CNN-Cert	Randomized smoothing after training Lecuyer at el. 2019 [34] ImageNet (299x299x3), Inception-v3 + auto-encoder Cohen at el. 2019 [35] ImageNet (299x299x3), ResNet-50 (50-layer) Tighter certification than Lecuyer [34]
Uncertified			Estimation of an upper bound (UB) of MSR Carlini and Wagner 2017 [31] ImageNet (299x299x3), Inception-v3 Estimation of an approximation of MSR Weng et al. 2018 (CLEVER) [32] ImageNet (299x299x3), ResNet-50 (50-layer)	Training by detecting near adversarial exes Madry et al. 2018 [36] CIFAR (32x32x3), 28-10 wide ResNet

- Columns in Table 5.1 (application):
  - Evaluation of robustness by estimating MSR
  - Improvement of robustness by increasing data samples with a specified MSR
- Row in Table 5.1 (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 5.1 are explained in the following Subsections 5.2.1~5.2.7.

5.2.1 Certified and rigorous evaluation of robustness

Katz et al. [26] 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. [27] 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.

5.2.2 Certified, approximative, and deterministic evaluation of robustness

Weng et al. [28] 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 5.3. 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.

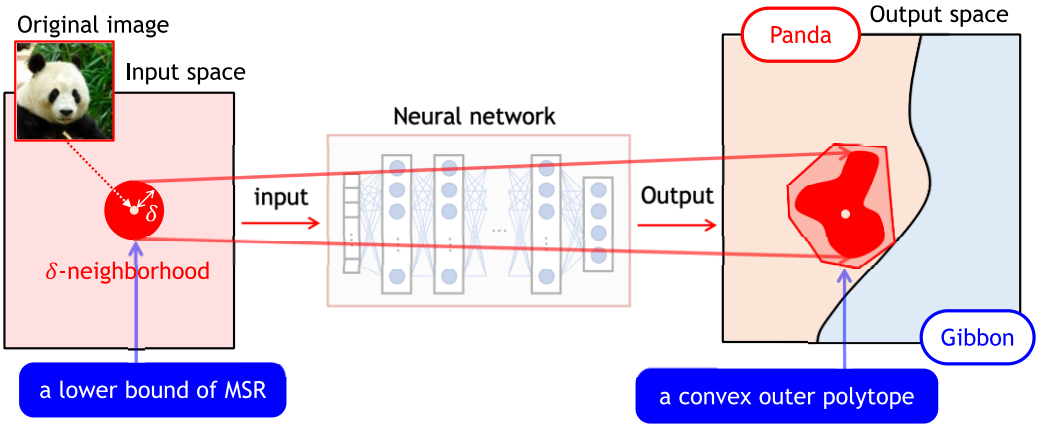

Figure 5.3 An approximation δ that is slightly smaller than the maximum safe radius (MSR)

Boopathy et al. proposed CNN-Cert, which is an improved version of Fast-Lin [29]. 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.

5.2.3 Certified, approximative, and probabilistic evaluation of robustness

Weng et al. [30] proposed a method, PROVEN, to approximate probabilistic maximum safety radii. As shown in Figure 5.4, 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.

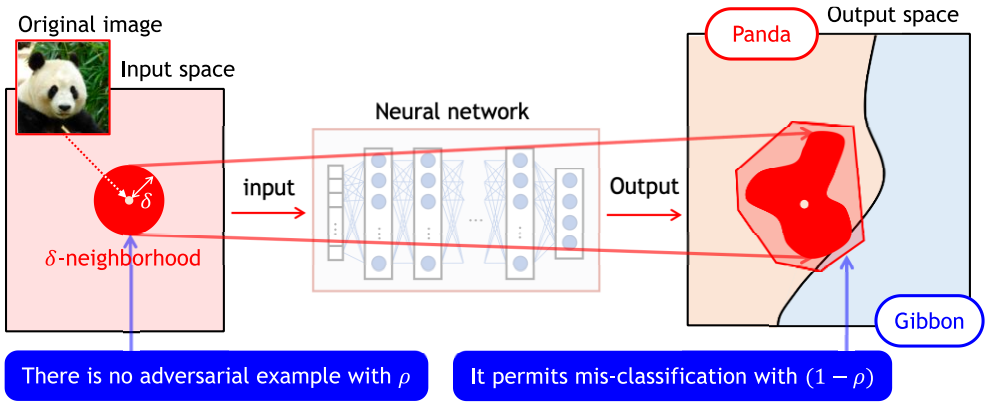

Figure 5.4 An approximation δ that is slightly smaller than the probabilistic MSR with ρ

5.2.4 Uncertified evaluation of robustness

Carlini and Wagner [31] 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. [32] 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 5.5, 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).

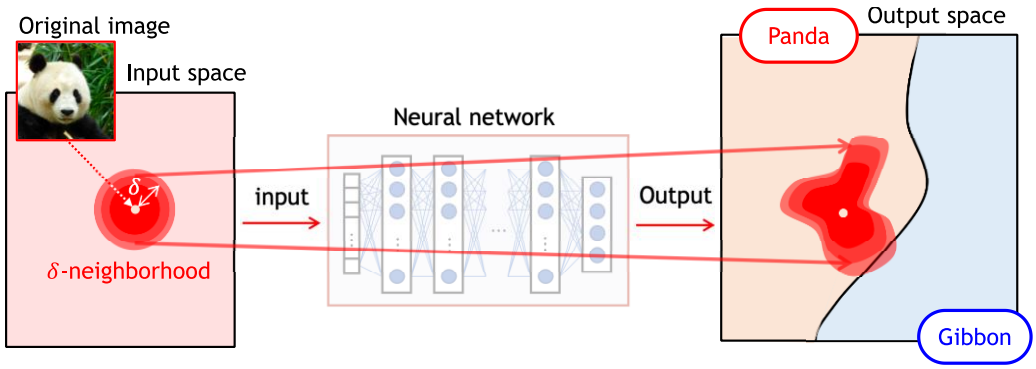

Figure 5.5 An approximation of the maximum safe radius (uncertified)

5.2.5 Certified, approximative, and deterministic improvement of robustness

Wong et al. [33] 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 5.6, 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. [33] 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).

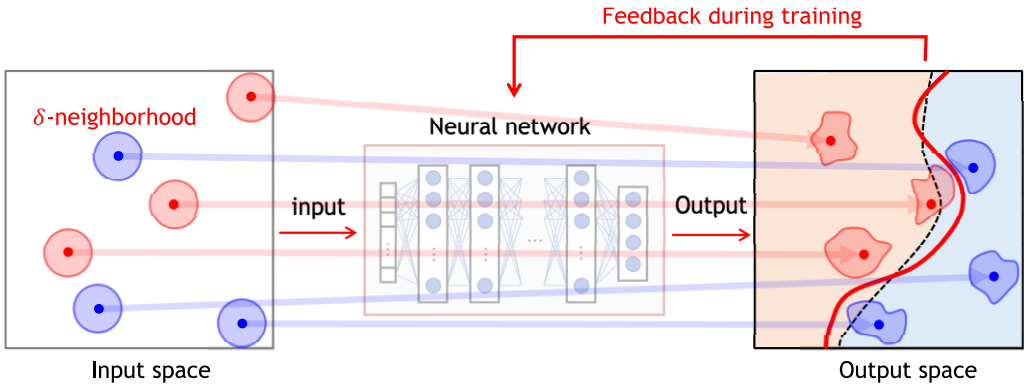

Figure 5.6 Robust-trining by input data with δ-neighberhood

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

Lecuyer et al. [34] 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 5.7, 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. [34] 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. [34] 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. [35] proposed a randomized smoothing based method that can estimate tighter certifiable approximative probabilistic maximum safe radii than one of Lecuyer et al. [34].

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.

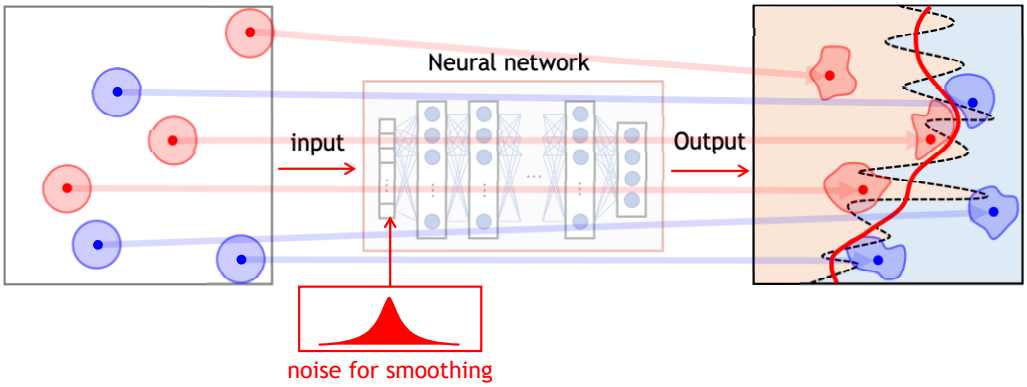

Figure 5.7 Improvement of robustness by randamized smoothing

5.2.7 **Uncertified improvement of robustness**

Madry et al. [36] 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. [33], 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.

5.3 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.

6 Estimation of Generalization Error Upper Bounds

In this chapter, we report the results of a survey on techniques for estimating the generalization error upper bound in feedforward neural networks, especially classifiers, for the purpose of guaranteeing the behavior of machine learned models for unseen input data.

6.1 Generalization error upper bounds

In this chapter, we focus on classifiers based on feedforward neural networks, as shown in Figure 6.1, where it is assumed that there exists a correct classification class $y \in \mathcal{Y}$ for each input data sample $x \in \mathcal{X}$, and sample data (x, y) are according to the distribution $\mathcal{D}$. A trained neural network is a function h from the input space $\mathcal{X}$ to the output space $\mathcal{Y}$. In this chapter, the function h is called a *hypothesis*. A neural network can represent various hypotheses by adjusting parameters (weight matrix, etc.), where the set of representable hypotheses is denoted by $\mathcal{H}$. Therefore, training is to select a hypothesis h from a set of hypotheses $\mathcal{H}$ that fits the training dataset by a training algorithm.

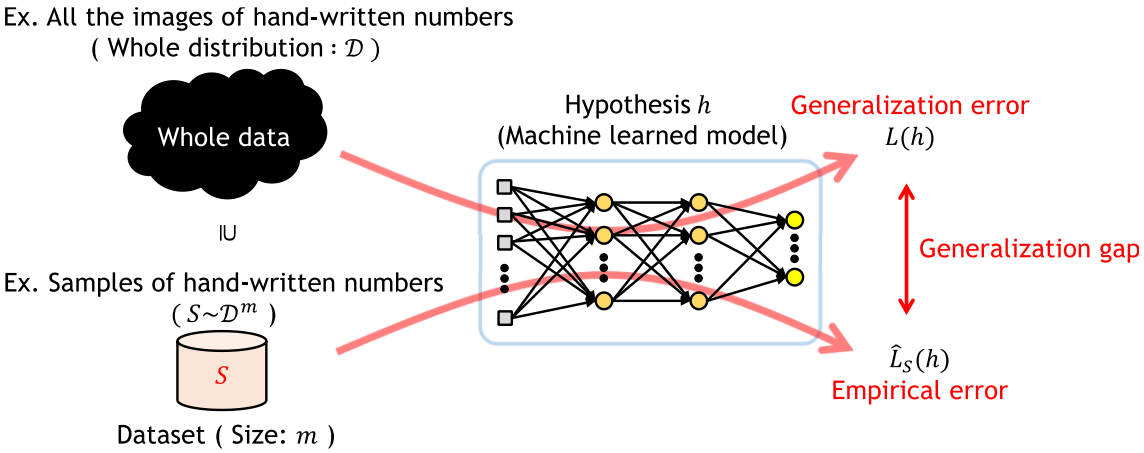

Figure 6.1 Generalization error and empirical error

Then, the generalization error $L(h)$ of the hypothesis h (a classifier) is the expected value of the error rate for all input data according to distribution $\mathcal{D}$, and is defined as follows.

$$L(h) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[l\left(h,(x,y)\right)\right]$$

where $l\left(h,(x,y)\right)$ is the loss function, which returns 0 if the inference $h(x)$ matches the correct output y for the input x and return 1 otherwise (i.e., 0-1 loss function).

$$l\left(h,(x,y)\right) = \begin{cases} 0 & \text{if } h(x) = y \\ 1 & \text{otherwise} \end{cases}$$

The empirical error $\hat{L}_S(h)$ of the hypothesis h is the error rate of the inferences $h(x)$ for m input data samples x in the dataset $S \sim \mathcal{D}^m$, which is according to distribution $\mathcal{D}$.

$$\hat{L}_S(h) = \frac{1}{m} \sum_{(x,y) \in S} l\big(h, (x, y)\big)$$

Especially, if S is the training dataset, then $\hat{L}_S(h)$ is also called a training error.

In general, it is difficult to exactly calculate the generalization error because there are numerous input data in the input space, but various estimation methods of generalization error upper bound, which guarantee that the generalization error is probabilistically less than the upper bound, have been already proposed. Valle-Pérez and Louis [37] presented a table (Table 1 in the paper [37]) of classified generalization error upper bound estimation methods according to their application conditions.

Table 6.1 A classification of generalization error upper bound estimation methods (Table 1 in [37])

<table>
<tr><th></th><th></th><th>Data independent</th><th>Data dependent</th></tr>
<tr><td rowspan="3">Algorithm independent</td><td>Uniform convergence</td><td>VC dimension-based bounds
By VC dimension that is a complexity measure of hypothesis set.</td><td>Rademacher complexity-based bounds
By Rademacher complexity of hypothesis set and dataset.</td></tr>
<tr><td rowspan="2">Non-uniform convergence</td><td colspan="2">Structural Risk Minimization-based bounds
By dividing a hypothesis set with non-uniform convergence into countable hypothesis subsets with uniform convergence and by applying a complexity measure to each subset.</td></tr>
<tr><td rowspan="2">Algorithm independent</td><td>Margin-based bounds
By the output margin and norm of weight matrix in a trained hypothesis

Sensitivity-based bounds
By the insensitivity to noise and norm of weight matrix in a trained hypothesis</td></tr>
<tr><td>Other</td><td>Compression-based bounds
By the dependency of a trained hypothesis to training dataset</td><td>Stability-based bounds
By the stability of the randomized training algorithm to training dataset

Marginal likelihood-based bounds
By the marginal likelihood of the prior distribution of hypotheses</td></tr>
</table>

Table 6.1 shows the estimation methods, which are classified in the same perspectives in Table 1 of Valle-Pérez and Louis [37]. The estimation methods in the upper part of Table 6.1 are basic and can be applied without any conditions on training algorithm, but the accuracy of the estimation is low (i.e., the difference between the generalization error and the upper bound is

very large) because it estimates the generalization error for the worst hypothesis in the hypothesis set. On the other hand, the estimation methods in the lower-right part of Table 6.1 impose some conditions on their application (e.g., the stochastic gradient descent method must be applied, and it must achieve zero training-error), but the accuracy of the estimated upper bound is higher.

6.2 Generalization error upper bound estimation methods

In this section, we briefly explain each generalization error upper bound estimation method in Table 6.1. For more details, refer to the paper [37] by Valle-Pérez and Louis.

6.2.1 Vapnik-Chervonenkis-based bounds

One of basic generalization error upper bound estimation methods is based on the VC (Vapnik-Chervonenkis) dimension [38]. The VC dimension $VC(\mathcal{H})$ is a complexity measure of the hypothesis set $\mathcal{H}$, which is the maximum number of data that can be divided by $\mathcal{H}$. For example, the VC dimension of the hypothesis set of a linear classifier with n-dimensional inputs is $n+1$. Although it is difficult to calculate the exact VC dimension of neural networks, their approximations and upper bounds can be estimated efficiently [39].

The generalization error upper bound can be estimated by the following inequality [37] using the VC dimension $VC(\mathcal{H})$ of the hypothesis set $\mathcal{H}$ prepared as candidate hypotheses. This inequality holds with probability $(1-\delta)$ for any training data set $S \sim \mathcal{D}^m$ (size: m) and any hypothesis $h \in \mathcal{H}$.

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

When the parameter δ is close to 0, the probability of this inequality is close to 100%, but the right side becomes infinitely large. Therefore δ must be set to a appropriate value. For example, if δ is set to 0.01, it can be guaranteed with probability 99% that the generalization error $L(h)$ is less than the value on the right side of the above inequation.

Most of the generalization error upper bound estimated based on VC-dimension exceed 100% because it considers the case that the worst hypothesis is selected from the hypothesis set $\mathcal{H}$ (determined by the structure of the neural network, etc.). Therefore, it is difficult to use it to evaluate the generalization performance of hypotheses (machine learned models).

It is noted that $VC(\mathcal{H})$ becomes infinite and cannot be given an upper bound if the hypothesis set $\mathcal{H}$ does not have the uniform convergence property, where the uniform convergence means that if sufficiently many training data samples are prepared then the difference between the generalization error and the training error can be sufficiently small for any hypothesis $h \in \mathcal{H}$. Since the hypothesis set does not always have the uniform convergence property in practical machine learning, the generalized error upper bound estimation method

for non-uniform convergence is explained in Subsection 6.2.3.

6.2.2 Rademacher complexity-based bounds

The Rademacher complexity [38] is the complexity $R(S,\mathcal{H})$ of the hypothesis set $\mathcal{H}$ for the dataset S. The complexity $R(S,\mathcal{H})$ is the expected value of the supremum of the difference between the two empirical errors $\left|L_{S_1}(h) - L_{S_2}(h)\right|$ for any hypothesis $h \in \mathcal{H}$, when the dataset S is randomly divided into two subsets S_1 and S_2.

The generalization error upper bound can be estimated by the following inequality [37] using the Rademacher complexity $R(S,\mathcal{H})$. This inequality holds with probability $(1-\delta)$.

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

where c is a constant. Since the training dataset S is considered, the Rademacher complexity bounds are lower than the VC dimension bounds. But since the hypothesis (the machine learned model after training) is not considered, most of the generalization error upper bound exceed 100%. It is difficult to use the Rademacher complexity bounds to evaluate the generalization performance of hypotheses by the same reason as the case of the VC dimension bounds.

6.2.3 Structural Risk Minimization-based bounds

The structural risk minimization (SRM) can be used for estimating the generalization error upper bound if a hypothesis set $\mathcal{H}$ can be countably divided into hypothesis subsets $\mathcal{H}_i$ with uniform convergence (i.e., $\mathcal{H} = \bigcup_{i\in\mathbb{N}} \mathcal{H}_i$), namely $\mathcal{H}$ has the non-uniform convergence property which is sufficiently practical.

In the structural risk minimization for a hypothesis set $\mathcal{H}$ with non-uniform convergence, the generalization error upper bound can be estimated by the following inequality [37] using each hypothesis subsets $\mathcal{H}_i$. This inequality holds with probability $(1-\delta)$ if the training error of the hypothesis h is zero (i.e., $\hat{L}_S(h) = 0$).

$$L(h) \leq \frac{1}{m}\left(\ln\frac{1}{P_i(h)} + C_i(S) + \ln\frac{1}{\delta}\right)$$

where $P_i(h)$ is the distribution function of the hypotheses h in the hypothesis subset $\mathcal{H}_i$, and $C_i(S)$ is the arbitrary complexity measure of $\mathcal{H}_i$ and the training dataset S. For example, since $\mathcal{H}_i$ has the uniform convergence property, $C_i(S)$ can be instantiated by the VC-dimension $VC(\mathcal{H}_i)$, the Rademacher complexity $R(S,\mathcal{H}_i)$, and so on.

Generalization error upper bounds for practical hypothesis set $\mathcal{H}$ with non-uniform convergence can be estimated in the structural risk minimization. However, most of the generalization error upper bounds exceed 100% by the VC dimension or the Rademacher complexity. It is still difficult to use it to evaluate the generalization performance of hypotheses.

6.2.4 **Margin based bounds**

The output margin is the difference between the largest output of the neuron and the second largest output of the neuron in the output layer. Even if two hypotheses have the same empirical error, the hypothesis with larger margins can be more robust for noise than the other one. The empirical error $\hat{L}_{S,\gamma}(h)$ considering such output margins is defined as follows.

$$\hat{L}_{S,\gamma}(h) = \frac{1}{m} \sum_{(x,y)\in S} l_\gamma\big(h,(x,y)\big),$$

where γ is a threshold of output margins and $l_\gamma\big(h,(x,y)\big)$ is the loss function with margin defined as follows.

$$l_\gamma\big(h,(x,y)\big) = \begin{cases} 0 & \text{if } h(x)[y] \le \gamma + \max\limits_{y' \ne y} h(x)[y'] \\ 1 & \text{otherwise} \end{cases}$$

where, $h(x)[y]$ is the output value of the neuron assigned to the class $y \in \mathcal{Y}$ in the output layer of the neural network (i.e., $h(x) = \mathrm{argmax}_{y\in\mathcal{Y}}(h(x)[y])$). In the empirical error $\hat{L}_{S,\gamma}(h)$ with margin, even if the output value of a neuron assigned to the correct class is larger than the maximum output value of the other neurons, if the difference (output margin) is less than the threshold γ, it is counted as a misclassification.

In the margin-based bounds, the generalization error upper bound can be estimated by the following inequality [40][41] using the margin-threshold γ and the weight matrix w, where h_w is the hypothesis derived from w. This inequality holds with probability $(1-\delta)$.

$$L(h_w) \le \hat{L}_{S,\gamma}(h_w) + \sqrt{\frac{\left(42\sum_{i=1}^{d}\left(2\sqrt{\omega_i} + \sqrt{2\ln(2d)}\right)\right)^2 \prod_{i=1}^{d}\|w_i\|_2^2 \times \sum_{i=1}^{d}\frac{\|w_i\|_F^2}{\|w_i\|_2^2} + \ln\left(\frac{m}{\delta}\right)}{\gamma^2 m}}$$

where d is the number of layers, w_i is the weight matrix to the i-th layer, and ω_i is the number of elements in w_i. Note that $\|w\|_2$ is the spectral norm of matrix w, and $\|w\|_F$ is the Frobenius norm of w.

The second term on the right-hand side of the inequality shows that the generalization gap (the difference between the generalization error and the empirical error) decreases as the margin threshold γ is increased. On the other hand, the first term $\hat{L}_{S,\gamma}(h_w)$ of the right-hand side increases with γ. They mean that the generalization errors of hypotheses that have larger output margins become smaller, which is consistent with the empirical results. In general, since the margin-based bounds also often exceed 100%, it is difficult to apply the bounds to absolute evaluation of generalization performance, but the threshold γ is useful as a generalization measure [40] for relative evaluation of generalization performance.

6.2.5 **Sensitivity-based bounds**

The sensitivity-based bound is a special case (for the Gaussian noise on weights) of the PAC-

Bayesian bound. In the PAC-Bayesian bounds, the amount of information that a hypothesis has obtained from the training dataset is estimated from the difference (called the KL-divergence) between the two (prior and posterior) probability distributions of parameters before and after training. Therefore, since the KL-divergence implies the dependency of the trained hypothesis to the training dataset.

In the sensitivity based bounds, when the Gaussian noise $\mathcal{N}(0,\sigma^2)$, whose mean is zero and standard deviation is σ, is added to each weight-parameter in w after training (ω: the number of parameters), where each parameter is randomly initialized by the Gaussian noise $\mathcal{N}(0,\sigma^2)$ before training, the upper bound of the expected value of the generalization error $L(h_{w+u})$ with noise u can be estimated by the following inequality [40][42] using the expected value of empirical (training) error $\hat{L}_S(h_{w+u})$ with noise u and the standard deviation σ of noise. This inequality holds with probability $(1-\delta)$.

$$\mathbb{E}_{u\sim\mathcal{N}(0,\sigma^2)^\omega}[L(h_{w+u})] \leq \mathbb{E}_{u\sim\mathcal{N}(0,\sigma^2)^\omega}[\hat{L}_S(h_{w+u})] + 4\sqrt{\frac{1}{m}\left(\frac{\|w\|_2^2}{2\sigma^2} + \ln\frac{2m}{\delta}\right)} \quad \cdots\cdots \quad (*_1)$$

where ω is the number of weigh-parameters.

The first term on the RHS (right-hand side) of this inequality can be approximated as the average of several measurements of the empirical error $\hat{L}_S(h_{w+u})$ under the Gaussian noise u, as shown in Figure 6.2.

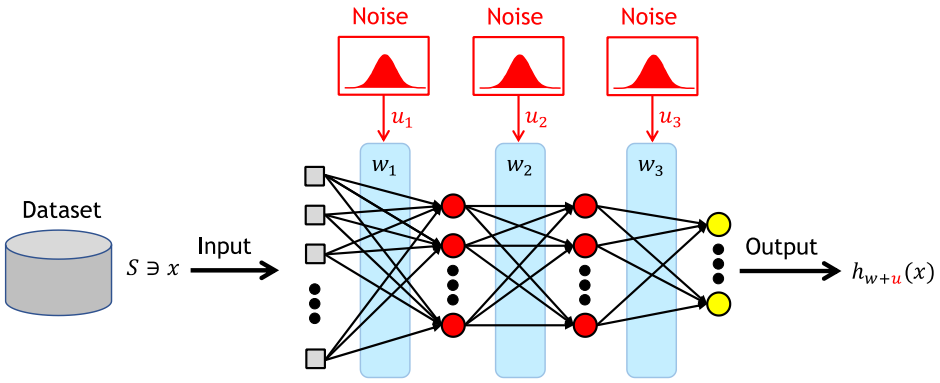

Figure 6.2 The measurement of the empirical error with the Gaussian noise

The second term on the right-hand side of the inequality shows that the generalization gap decreases as the standard deviation σ of the noise is increased. On the other hand, the first term on the RHS increases with the noise σ. They mean that the generalization errors of hypotheses that are more insensitive to noise become smaller, which is consistent with the empirical results.

The sensitivity-based bounds also often exceed 100%. Figure 6.3 shows an example of expected generalization error upper bounds (i.e., the RHS of the inequality $(*_1)$) estimated by the sensitivity-based method for a simple feedforward neural network, where $\delta = 0.1$. The neural network consists of fully connected three layers, where the input layer has 28×28 neurons, the hidden layer has 64 neurons with ReLU, and the output layer has 10 neurons with Softmax, and it was trained on the MNIST (brightness: $[0, 1]$ for each pixel) dataset, whose size is 54,000. As shown in Figure 6.3, the estimated upper bounds exceed 100%, for example, the

expected generalization error upper bound (i.e., the RHS of $(*_1)$) is 323% when $\sigma = 0.05$. Therefore, it is difficult to directly apply the bounds to absolute evaluation of generalization performance. However, similarly to the margin threshold γ explained in Subsection 6.2.4, the standard deviation σ which increases the expected training error with noise to a constant value e_T (e.g., if $e_T = 10\%$ then $\sigma = 0.05$ in Figure 6.3), is also useful as a generalization measure [40] for relative evaluation of generalization performance.

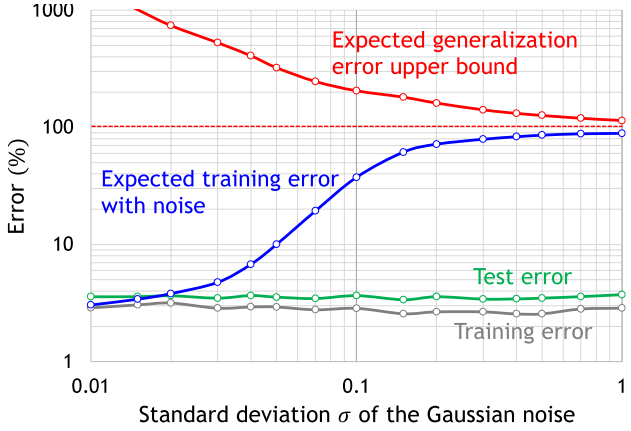

Figure 6.3 An example of expected generalization errors estimated by the sensitivity-based method

6.2.6 **Compression-based bounds**

In the compression-based bounds, for a hypothesis h trained by using a training dataset S, it is necessary to divide S into a dependent subset T_h and an independent subset V_h ($S = T_h \cup V_h$) such that the following two conditions are satisfied:

- The hypothesis h depends only on the dependent subset T_h
- The empirical error of h for the independent subset V_h (i.e., $\hat{L}_{V_h}(h) = 0$)

For the hypothesis h, if the training data set S (size: m) can be divided into a dependent subset T_h (size: k) and an independent subset V_h (size: $m - k$), then the generalization error upper bound can be estimated by the following inequality [40] using the size k of the subset T_h. This inequality holds with probability $(1 - \delta)$.

$$L(h) \leq \frac{8k}{m} \ln \frac{m}{\delta}$$

The right-hand side shows that the generalization error upper bound becomes small by reducing the dependence (k/m) of the hypothesis h on the dataset S. The methods (called compression schemes) for dividing a dataset into a dependent subset and an independent subset are necessary for the application of this inequality and have been being developed. For example, Brutzkus et al. [43] developed a compression scheme for two-layer neural networks trained with stochastic gradient descent method. More generic compression schemes are still in the research phase.

6.2.7 **Stability-based bounds**

In the stability-based bounds, the generalization error upper bounds are estimated based on the stability of stochastic algorithms. The stability of a stochastic algorithm $\mathcal{A}$ is defined by the supremum ϵ on the expected value of the incremental loss $l\left(\mathcal{A}_{(S-\{z\})\cup\{z'\}}, z\right) - l(\mathcal{A}_S, z)$ when any one data sample z in the training dataset S is replaced by another sample z', where $\mathcal{A}_S$ is the hypothesis obtained by training dataset S with algorithm $\mathcal{A}$. The small supremum ϵ means that the stochastic algorithm is less dependent on the training dataset S, in other words, it is stable.

Based on the stability of the algorithm $\mathcal{A}$ for the dataset S (size: m), for the hypothesis $\mathcal{A}_{S_k}$ where S_k is the subset of k samples ($k \leq m$) randomly selected from the dataset S, the upper bound of the difference between the generalization error and the empirical error can be estimated by the following inequality [44].

$$\mathbb{E}_{(S\sim\mathcal{D}^m,\ \mathcal{A})}\left[L\left(\mathcal{A}_{S_k}\right) - \hat{L}_S\left(\mathcal{A}_{S_k}\right)\right] \leq \mathcal{O}\left(\sqrt{c\left(L\left(h_{w_1}\right) - L^*\right)} \cdot \frac{\sqrt[4]{k}}{m} + c\sigma\frac{\sqrt{k}}{m}\right)$$

where c is a constant related to the step size in the stochastic algorithm, h_{w_1} is the initial hypothesis derived from the initial weight matrix w_1, L^* is the minimum error achievable in the hypothesis set $\mathcal{H}$, and σ is the standard deviation of the stochastic gradient during training. Although the generalization error $L\left(h_{w_1}\right)$ of the initial hypothesis h_{w_1} cannot be exactly calculated, it can be approximated by its empirical error $\hat{L}_S\left(h_{w_1}\right)$ because the initial weight matrix is generally random.

In the paper [44], upper bounds close to the generalization errors are estimated based on stability, but the stability-based bounds require some conditions such that the loss function must be smooth, and each data sample must be used once at most (one-pass). The research for weakening the conditions (e.g., non-smooth loss function and multi-pass) has been ongoing.

6.2.8 **Marginal likelihood-based bounds**

Similarly to the sensitivity-based bounds explained in Subsection 6.2.5, the marginal likelihood-based bound is also a special case of the PAC-Bayesian bound, but it uses the KL-divergence between prior- and posterior-distributions of hypotheses instead of weight matrix. The hypothesis $h_w \in \mathcal{H}$ is a function derived from a weight matrix $w \in \mathcal{W}$, and in general, several weight matrixes w, w' are mapped into the same hypothesis $h_w = h_{w'}$. Therefore, the KL-divergence between the prior- and posterior-distributions is less on the hypothesis set $\mathcal{H}$ than on the weight matrix set $\mathcal{W}$. It means that the marginal likelihood-based bounds are tighter to the generalization errors than the sensitivity-based bounds. However, the marginal likelihood-based bounds require the following two conditions.

- The training error of h for the training dataset S must be zero (i.e., $\hat{L}_S(h) = 0$).
- The stochastic training algorithm must almost uniformly sample the zero-error region.

If the two conditions are satisfied, then the following equation holds for any zero-error hypothesis $h \in \mathcal{H}_0(S) = \{h \in \mathcal{H} \mid L_S(h) = 0\}$,

$$Q(h) = \frac{P(h)}{M_0(S)}, \quad where \ M_0(S) = \sum_{h \in \mathcal{H}_0(S)} P(h)$$

where $P(h)$ and $Q(h)$ are the prior distributions (before training) and the posterior distributions (after training) of hypotheses, respectively, and $M_0(S)$ is the probability of selecting a hypothesis with zero training error for S in the prior distribution $P(h)$, which corresponds to the marginal likelihood of S. It has been shown that the stochastic gradient descent method (SGD) tends to satisfy the conditions [45].

When the above conditions are satisfied, the generalization error upper bound can be estimated by the following inequality [37] for the hypothesis h such that $Q(h) \geq 1 - \gamma$.

$$L(h) < 1 - \left(\frac{m}{M_0(S)\,\gamma\,\delta}\right)^{-\frac{1}{m-1}}$$

It has been shown [45] that simplicity bias is imposed when weight matrixes are mapped into hypotheses in neural networks. Consequently, even if weight matrixes are initialized with uniform random numbers, the distribution $P(h)$ of the hypotheses derived from the matrixes is not uniform because simpler hypotheses (with lower Kolmogorov complexity) tend to have a higher probability. In general, it is expected that real-world hypotheses (rather than, for example, artificial random functions) have a kind of regularity and/or structure. Therefore, the probability $P(h)$ of a hypothesis h for the real world will be high, which means that $M_0(S)$ and γ can be large, and then implies that the upper bounds can become small.

Valle-Pérez and Louis [37] proved some optimality of the marginal likelihood-based bounds and performed experiments on the estimation of the upper bound by the two-class classification problems but have not shown sufficiently comparisons with the other methods yet. The research on the marginal likelihood-based bounds (e.g., multi-class classification and computation of marginal likelihood) has been ongoing.

6.3 Conclusion

It is still difficult for absolute evaluation of the generalization performance of hypotheses to apply the methods for estimating the generalization error upper bounds introduced in this chapter because most of the estimation result exceed 100% (i.e., vacuous). Most research papers do not concern about the absolute values of bounds because they aim at theoretically explaining the empirical phenomenon (e.g., generalization performance can be obtained even in over-parameterized network) of deep neural network. Even though the estimation results of the generalization error upper bounds exceed 100%, it is still useful as a relative generalization measure for comparing the generalization performance of hypotheses, and such generalization measures have been theoretically and empirically evaluated [40][46]. On the other hand, some recent papers have been presented the methods for estimating the non-vacuous upper bound

(i.e., less than 100%) of generalization errors and the upper bounds have been becoming close to the generalization errors. For example, Zhou et al. [47] formalized a generalization bound for compressed trained neural network and provided a non-vacuous generalization guarantee for the realistic dataset ImageNet. And Garg et al. [48] proposed a method, named RATT, that leverages unlabeled data to estimate generalization bounds and their several experiments showed that the bounds are non-vacuous. We think that the generalization error upper bound can be one of measures of generalization performance in a few years.

7 Adversarial Example Detection

7.1 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

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 [49])
② Denoisers approaches (example [50])
③ Prediction inconsistency based approaches (example [51])
④ Neural Network Invariant Checking approaches (example [52])

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 [52], 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 a NIC according to the paper [52]. Therefore, this chapter mainly describes 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. Finally, the results of the follow-up experiments of each approach and the experiments by NIC are described.

7.2 Overview of adversarial example detection approaches

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

7.2.1 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) [49]. 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.

7.2.2 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 [50] is a method of detecting adversarial examples using detectors and reformers (trained automatic encoders and automatic decoders).

7.2.3 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 [51], 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.

7.2.4 Neural Network Invariant Checking approaches

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 [51], 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 value invariants (VIs) and provenance invariants (PIs) inside neural networks in deep neural network NICs [52]. The value invariant VI is the distribution of possible neuron values in each layer, and the history 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. Train 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 [52]. The outline and the implementation of the NIC system design are explained in detail in Sections 7.3 and 7.4, respectively.

7.3 NIC system design overview

The procedure for building and detecting the NIC detector (steps A to C: during training, D to E: during execution) is explained using Figure 7.1 [52]. This invariant VI, PI training uses only non-adversarial benign data.

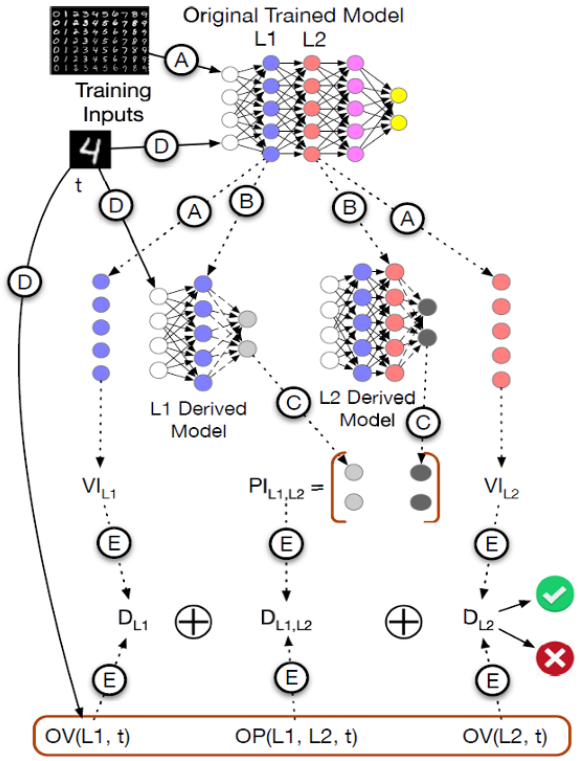

Figure 7.1 Outline of system design (Fig. 8 of thesis [52])

- **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 submodels 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 7.1)

- **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 7.1) 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 7.1) 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.

7.4 NIC system implementation

In order to detect adversarial examples based on NIC, a direct sum space (vector) is constructed from PI and VI, and an OSVM (One Class Support Vector Machine) for classifying this vector 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.

7.5 Computer experiment

In order to confirm the effect of adversarial example detection technology (NIC), the experiment of the paper [52] was retested in the following experimental environment.

- Hardware environment: AIST ABCI [53]
- Datasets: Two common image datasets, MNIST [54] and CIFAR-10 [55], 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) [56].
- 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 [57] 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 [49], MagNet [50], and feature squeezing [51] was used to evaluate each paper. A follow-up experiment was conducted. As a result, the detection rates reported in each paper could 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 7.4. Table 7.1 to Table 7.3 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 [52], but also for the unreported dataset LFW and attack method (FGSM L^∞).

Table 7.1 Adversarial example detection computational experiment results for MNIST dataset

Data Set	Attack	Invariant	Performance	Number of data	Performance reported in the paper [52]
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 7.2 Adversarial example detection computational experimental results for CIFAR-10 dataset

Data Set	Attack	Invariant	Performance	Number of data	Performance reported in the paper [52]
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 7.3 Adversarial example detection computational experiment results for LFW dataset

Data Set	Attack	Invariant	Performance	Number of data	Performance reported in the paper [52]
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 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 as a technique for AI quality management during operation.

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 [58]. Table 8.1 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 [59] and Lu et al. added recently published drift detection and adaptation methods in [60]. 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 [61]. 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 8.1 Comparison of survey papers on concept drift detection and adaptation

	Detection	**Adaptation**
Supervised	Gama et al.[59], Lu et al.[60]	
Unsupervised / Semi-supervised	Ishida et al.[61]	Okawa and Kobayashi [58] (Ours)

9 References

Chapter 1:

[1] National Institute of Advanced Industrial Science and Technology (AIST), Machine Learning Quality Management Guideline, 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/

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

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

Chapter 2:

[4] 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).

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

Chapter 3:

[10] 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.

[11] 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.

[12] Zhang, H., Cisse, M., Dauphin, Y. N., and Lopez-Paz, D., Mixup: Beyond Empirical Risk

Minimization. In International Conference on Learning Representations, 2018.

[13] 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.

[14] 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.

[15] 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:

[16] Nakajima, S., Quality Issues in Machine Learning from Software Engineering Viewpoints, Maruzen Publisher, 2020. (in Japanese)

[17] Pei, K., et al., DeepXplore: Automated Whitebox Testing of Deep Learning Systems, In Proc. 26th SOSP, 2017, pp.1-18.

[18] Nakajima, S., Distortion and Faults in Machine Learning Software, In Post-Proc. 9th SOFL+MSVL, 2020, pp.29-41.

[19] Ma, L., et al., DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems, In Proc. ASE, 2018, pp.120-131.

[20] Tian, Y., et al., DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars, In Proc. 40th ICSE, 2018, pp.303-314.

[21] Zhang, M., et al., DeepRoad: GAN-Based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems, In Proc. ASE, 2018, pp.132-142.

[22] Zhang, P, et al., CAGFuzz: Coverage-Guided Adversarial Generative Fussing Testing of Deep Learning Systems, arXiv:1911.07931, 2019.

[23] Harel-Canada, F., et al., Is Neuron Coverage a Meaningful Measure for Testing Deep Neural Networks? In ESEC/FSE, 2020, pp.851-862.

[24] Kim, J. et al., Guiding Deep Learning System Testing Using Surprise Adequacy, In Proc. 41st ICSE, 2019, pp.1039-1049.

Chapter 5:

[25] 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

[26] 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

[27] 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

[28] 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

[29] 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

[30] 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

[31] 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

[32] 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

[33] 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

[34] 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

[35] 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

[36] 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 6:

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

[38] 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

[39] Peter L. Bartlett, Nick Harvey, Christopher Liaw, and Abbas Mehrabian, Nearly-tight VC-dimension and Pseudodimension Bounds for Piecewise Linear Neural Networks, Journal of Machine Learning Research, Vol.20, No.63, pp.1－17, 2019. https://jmlr.org/papers/v20/17-612.html

[40] 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

[41] 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

[42] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nathan Srebro, Exploring Generalization in Deep Learning, NeurIPS 2017. arXiv:1706.08947. https://arxiv.org/abs/1706.08947

[43] Alon Brutzkus, Amir Globerson, Eran Malach, and Shai Shalev-Shwartz, SGD Learns Over-parameterized Networks that Provably Generalize on Linearly Separable Data, ICLR 2018. https://openreview.net/forum?id=rJ33wwxRb

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

[45] Guillermo Valle-Perez, Chico Q. Camargo, and Ard A. Louis, Deep learning generalizes because the parameter-function map is biased towards simple functions, ICLR 2019. arXiv:1805.08522. https://arxiv.org/abs/1805.08522

[46] 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

[47] 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

[48] 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

Chapter 7:

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

[50] 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.

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

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

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

[54] 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/

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

[56] 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.

[57] 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.

Chapter 8:

[58] 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.

[59] 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.

[60] 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.

[61] 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.