JAISCR, 2023, Vol. 13, No. 4, pp. 215 – 228
10.2478/jaiscr-2023-0016

TOWARDS ENSURING SOFTWARE INTEROPERABILITY BETWEEN DEEP LEARNING FRAMEWORKS

Youn Kyu Lee[1], Seong Hee Park[1], Min Young Lim[1], Soo-Hyun Lee[1], Jongwook Jeong[2,*]

[1]*Department of Computer Engineering, Hongik University, 94 Wausan-ro, Mapo-gu, Seoul (04066), Republic of Korea*

[2]*Department of Computer Science and Artificial Intelligence, Jeonbuk National University, 567 Baekje-daero, Deokjin-gu, Jeonju-si (54896), Republic of Korea*

[*]*E-mail: jwjeong55@jbnu.ac.kr*

Submitted: 26th June 2023; Accepted: 11th September 2023

Abstract

With the widespread of systems incorporating multiple deep learning models, ensuring interoperability between target models has become essential. However, due to the unreliable performance of existing model conversion solutions, it is still challenging to ensure interoperability between the models developed on different deep learning frameworks. In this paper, we propose a systematic method for verifying interoperability between pre- and post-conversion deep learning models based on the validation and verification approach. Our proposed method ensures interoperability by conducting a series of systematic verifications from multiple perspectives. The case study confirmed that our method successfully discovered the interoperability issues that have been reported in deep learning model conversions.

Keywords: deep learning, interoperability, validation&verification, deep learning frameworks, model conversion

1 Introduction

With the increasing use of artificial intelligence (AI) software, a number of deep learning (DL) frameworks for AI software development have been published [1, 2], such as TensorFlow [3], Keras [4], and PyTorch [5]. Since each DL framework offers different characteristics in terms of grammar, underlying mechanisms, and performance, engineers can select a particular framework depending on their purposes, development constraints, deployment environments, and preferences [1]. However, recently, as the need for incorporating multiple DL models (e.g., federated learning, ensemble learning, DL model compression, and DL model fusion increases), it has become important to ensure interoperability between DL models developed on different DL frameworks [6, 7, 8]. For example, to improve a target system's inference performance, it might be required to ensemble two different models, one developed on TensorFlow and the other developed on PyTorch. Another example can be the requirement to convert a PyTorch model into C language in order to deploy it on embedded systems.

Ensuring interoperability between DL models requires a reliable conversion process that allows a model developed on a particular DL framework to be deployed on another framework without any failure. To date, a variety of tools, such as Open Neural Network Exchange (ONNX) [9], Neural

Network Exchange Format (NNEF) [10], Model Management deep neural network (MMdnn) [11], Neural Network Tools (nn_tools) [12], and pytorch2keras [13], have been proposed to facilitate automated conversion of DL models.

However, due to their unreliable conversion performance, it is challenging to guarantee interoperability between pre- and post-conversion DL models [11, 14, 15]. Several cases of incorrect conversions that led to deployment errors and performance inconsistencies have been reported [14, 15]. Nevertheless, systematic verification methods for ensuring interoperability between DL models have not yet been developed. For example, sample testing, a general method for assessing the inference performance of DL models, is insufficient to verify interoperability between target models. Because the results may vary depending on the test dataset or the corner cases that can be discovered from other perspectives can be overlooked. Hence, it is required to develop a robust and systematic method for verifying the interoperability between DL models.

In this paper, we propose a new method for verifying interoperability between DL models from multiple perspectives based on the validation and verification (V&V) approach. V&V approach, which conducts both architecture-level and code-level verification, is being used for minimizing software erosion that can be caused by changes in target software [16, 17]. Our proposed method adapts the V&V approach to minimize software erosion that occurs during DL model conversion. Specifically, our method involves architectural validation of DL model, which verifies whether a target model's architecture is maintained after conversion. Our method also includes verification based on performance analyses, which confirms whether a target model's accuracy, inference values, and time were maintained after conversion. Our approach enables the identification of conversion issues that cannot be discovered by single-perspective analysis. For example, in the case when the architectural configuration of a target model was converted correctly but its weight values were not maintained, our multi-perspective method can detect interoperability issues that a single-perspective analysis (e.g., architecture-level analysis) cannot discover. To evaluate the effectiveness and applicability of our proposed method, we conducted case studies including representative open-source conversion tools, such as ONNX, NNEF, MMdnn, and pytorch2keras. The results confirmed that our proposed method was able to successfully validate and verify interoperability between pre- and post-conversion DL models.

The main contributions of this paper are as follows: (1) Proposal of a new method for ensuring interoperability between DL models; (2) Providing an effective process compatible with and adaptable to existing DL frameworks and conversion tools; (3) Design of a systematic process integrating multi-perspective validation and verification mechanisms; and (4) Evaluation on real-world cases including popular models, frameworks, and conversion tools. This paper is organized as follows. Section 2 presents related work. Section 3 describes our approach, and Section 4 illustrates our case studies and their results. Threats to validity are discussed in Section 5, and Section 6 provides the conclusion of this study and future work.

2 Related Work

2.1 Deep Learning Model Conversion

To use DL models in various environments, a number of tools have been developed to facilitate the conversion between different DL frameworks [9, 10, 11, 12, 13]. ONNX and NNEF, representative open-source DL tools, enable a target DL model to be run on a target framework by defining specification rules such as intermediate representation.

ONNX, developed by Microsoft and Facebook, provides high-fidelity conversion between DL frameworks. As shown in Figure 1, ONNX supports a variety of DL frameworks, including PyTorch, Caffe2 [18], Keras, and MXNet [19]. NNEF, developed by Khronos Group, enables a DL model to be deployed on different types of inference engines. By encapsulating a model's structure, operations, and parameters, NNEF supports reliable conversions between DL frameworks, including PyTorch, Caffe [20], and TensorFlow. MMdnn, an open-source tool developed by Microsoft, supports conversion between DL frameworks such as TensorFlow, PyTorch, Keras, and Caffe. nn_tools, a Python-based open-source tool, provides conversion between DL models as well as analysis of model layer information such as input, output, and weight size. However, it only supports one-way conversion from PyTorch to Caffe. Moreover, various open-source conversion tools have been developed, such as pytorch2caffe [21] and pytorch2keras. However, it has been reported that interoperability issues, such as inference performance degradation or model architecture modification, occur when converting DL models with existing tools [22, 23].

2.2 Validation and Verification

Validation and verification (V&V) is a representative process for developing a reliable software. [17] proposed a V&V process specifically designed for model-based software engineering. Since typical model-based software engineering presents automatically generated code (AGC) based on a defined model, their V&V process includes verification for AGC. Although a converted DL model can also be classified as an AGC, since their process has mainly focused on generic software, its direct application to DL model conversion is infeasible. Specifically, their process provides a set of tasks to verify the interoperability between a defined model and its AGC, which is not directly applicable to verification between pre- and post-conversion DL models.

For systems using neural networks, [24] proposed a V&V process for statistically estimating the reliability of pre-trained neural networks. Their proposed process verifies whether the target neural network can be deployed in mission-critical systems, such as flight control systems. However, their process was unsuitable for verifying conversion between DL models, as it focused primarily on development lifecycles specialized for systems using pre-trained neural networks. Thus, their process is not directly applicable to DL model interoperability verification. To ensure interoperability between DL models, a specialized and systematic V&V process is required.

3 Proposed Method

In this section, we present our proposed method for ensuring interoperability between DL models. Our proposed method facilitates minimization of conversion errors and inconsistencies, which may occur during cross-framework conversions of DL models. As shown in Figure 1, a DL model (*M1*) implemented in a particular framework (e.g., Keras) can be converted to a DL model (*M2*) implemented in the other framework (e.g., PyTorch) using conversion tools such as ONNX. Despite being implemented in different frameworks, the two models (i.e., *M1* and *M2*) should ensure interoperability by providing the same inference operations and performance. With pre- and post-conversion DL models, our method verifies interoperability between them via four main steps from different perspectives. An overview of our method is illustrated in Figure 2, and details of each step are described in the following subsections.

3.1 Step1: Architectural Analysis

In this step, by analyzing the architectural characteristics of target models (i.e., pre- and post-conversion DL models), their interoperability is verified. Specifically, the following conditions are evaluated in this step: (*Condition#1*) whether each layer of a pre-conversion model has been converted to a layer that performs the same operation after conversion; (*Condition#2*) whether the connections between layers in a pre-conversion model have been preserved; and (*Condition#3*) whether each layer's hyperparameters in a pre-conversion model are preserved. For example, during a conversion, a *"MaxPool2d"* layer in a PyTorch model can be converted to a *"MaxPooling2D"* layer in a Keras model, while some connections between layers have been removed. Thus, architectural analysis is essential to confirm interoperability as these architectural changes may be neglected in sample testing. Architectural analysis is divided into two sub-steps, `Step1-A`, which validates *Conditions#1* and *#2*, and `Step1-B`, which validates *Condition#3*. Note that, since each sub-step is independent and complementary to each other, even if `Step1-A` has not been passed, `Step1-B` should be validated for complete validation.

- **Step1-A.** Architectural changes can be distinctly identified through visualization-based analysis. In visualization-based analysis, pre- and post-conversion DL models are compared at the architectural-level by visualizing their layer

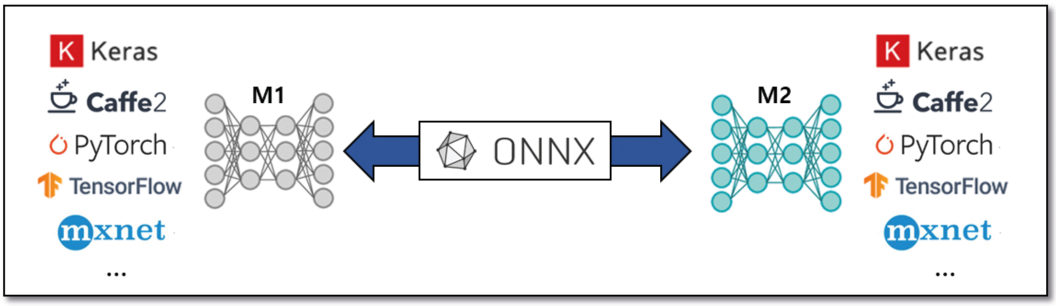

Figure 1. Interoperability between DL frameworks supported by ONNX

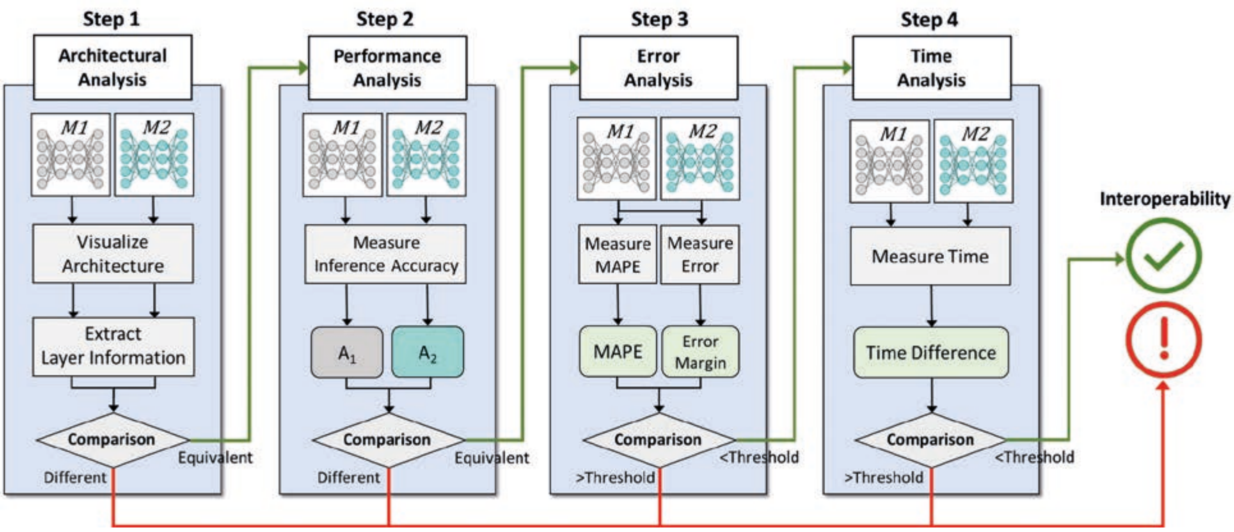

Figure 2. An overview of our proposed method

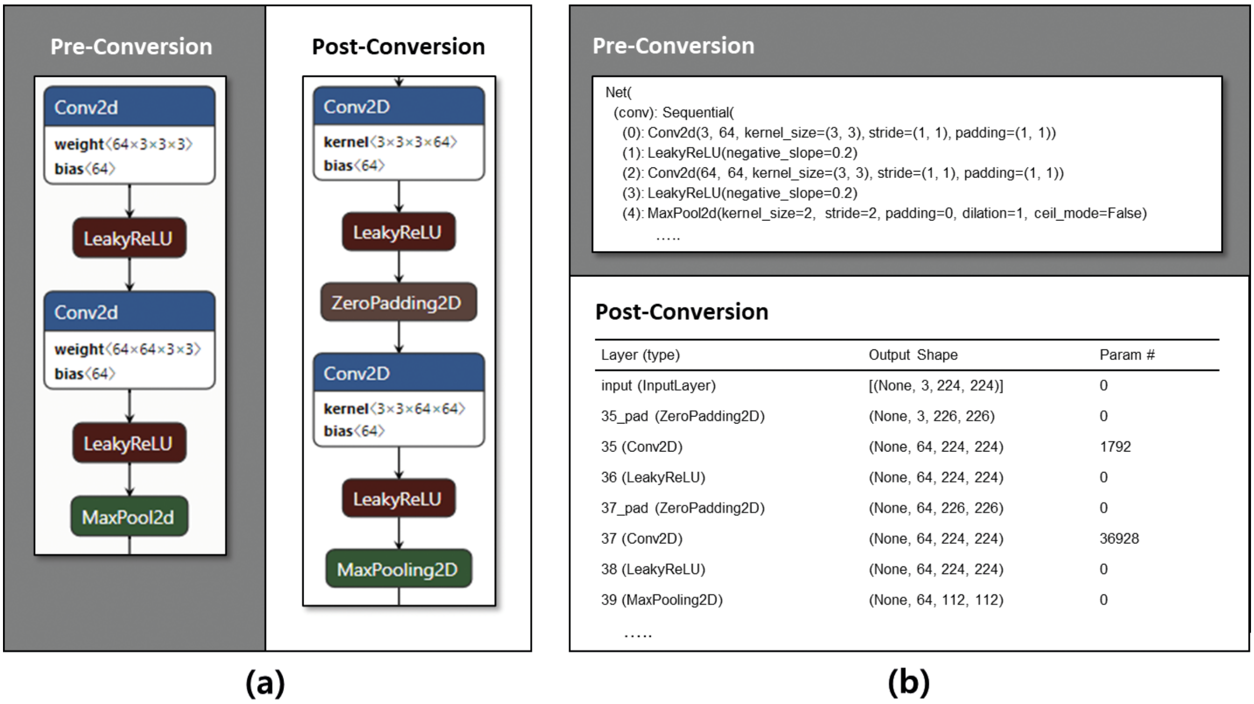

Figure 3. An example of architectural analysis: (a) `Step1-A`: visualization-based analysis and (b) `Step1-B`: detailed inspection

types, compositions, and connections between layers. Figure 3(a) shows an example visualization by Netron [25], a visualization tool for DL models. The example visualizes the architecture of pre-conversion DL model implemented in PyTorch (left) and that of its post-conversion DL model implemented in Keras (right), respectively. Comparing architectural characteristics using the aforementioned conditions reveals that a *"ZeroPadding2D"* layer has been added to the post-conversion model, which violates *Condition#1*. As mentioned earlier, after visualization-based analysis, a detailed inspection on each layer is required in order to thoroughly examine the changes in architectural variables.

– **Step1-B.** Detailed inspection verifies whether any change has occurred in architectural variables of each layer (e.g., hyperparameters, dimensions, and bias values). Continuing from the example of Figure 3(a), although a *"ZeroPadding2D"* layer has been added to the post-conversion model, this can be considered a parameter option. Padding can be applied, especially in Keras, by incorporating a padding layer in the model, which actually serves the same operation as setting a parameter option for padding in other DL frameworks. In this example, we can confirm that interoperability between the two models has been ensured. If it is confirmed that a post-conversion model preserved the architectural characteristics of a pre-conversion model, then we can proceed to the next step to perform the following verifications. Otherwise, it is confirmed that their interoperability has not been achieved.

3.2 Step2: Performance Analysis

In this step, by analyzing the inference performance of the target models, their interoperability is verified. Specifically, it is verified whether the inference performance of the model on a test dataset has been maintained in terms of accuracy. By measuring the numerical differences in accuracy of two models, it can be determined whether the inference performance has been preserved or not. For example, in general, an accuracy is calculated as follows: (*the number of true positives*+*the number of true negatives*)/(*the total number of inferences*) [26]. For the same dataset, if the pre-conversion model's accuracy is 98.75% and the post-conversion model's accuracy is 98.11%, then the two models are considered to provide different inference performances, confirming that interoperability has not been achieved. If it is confirmed that a post-conversion model preserved the inference performance, then we can proceed to the next step to perform the following verifications. Otherwise, it is confirmed that their interoperability has not been achieved.

3.3 Step3: Error Analysis

In this step, by analyzing the inference error between the target models, their interoperability is verified. Even if pre- and post-conversion DL models had produced different inference values, their accuracies could be measured as equal according to a decision threshold. A performance analysis of a classification model identifies how accurately it predicts the class based on the decision threshold rather than an inference value for each test data. Hence, inference values are examined in this step for a detailed analysis on the inference operations. Error analysis can be conducted in two different ways, mean absolute percentage error (MAPE) analysis [27, 28] and error margin analysis [29].

- **Step3-A.** This step verifies the degree of inference error by measuring MAPE. In our method, MAPE represents the difference ratio of numerical values calculated from pre- and post-conversion DL models. This step verifies the degree of inference error by measuring MAPE, which is a widely used metric for estimating errors as a single numerical value [30, 31]. The formula for calculating MAPE is as follows:

$$\mathrm{MAPE} = \frac{100\%}{n} \sum_{t=1}^{n} \left| \frac{M1_t - M2_t}{M1_t} \right| \tag{1}$$

  where, for each data t among n total test data, $M1_t$ is an inference value of pre-conversion model (=$M1$), $M2_t$ is an inference value of post-conversion model (=$M2$).

  After calculating MAPE, it is compared with a threshold, which can be empirically determined. If MAPE is greater than the threshold, we can consider that the two models offer significantly different inference values, confirming that interoperability has not been achieved. Note that MAPE provides a numerical rationale of the changes in a target model's inference operations, but in some cases, it provides undesirable results. For example, in the case of "divide by zero" where $M1_t$ is 0, since calculating MAPE is infeasible, we need to proceed to the next step to perform further verification.

- **Step3-B.** This step verifies the range of inference errors by measuring error margins. In our method, error margin represents the range of differences in numerical values calculated from pre- and post-conversion DL models. If errors are observed within an acceptable margin, which can be empirically determined, we can consider that the two models offer interoperable inference values.

Algorithm 1: Error margin analysis

```
Input: Testdata
Let [0, p] be an acceptable margin
Let M1_x be an inference value of
  pre-conversion model for data x
Let M2_x be an inference value of
  post-conversion model for data x
foreach t ∈ Testdata do
    error_t = |M1_t − M2_t|
    if error_t > p then
        return fail
    end
end
return pass
```

Algorithm 1 iterates over each test data t in a test dataset (=$Testdata$) and considers the error margin between pre- and post-conversion models. Specifically, for each t, an error margin (=$error_t$) is estimated by calculating a difference between an inference value of pre-conversion model (=$M1_t$) and that of post-conversion model (=$M2_t$). Algorithm 1 then identifies if $error_t$ is within an acceptable margin (=p). If so, it stops its processing while returning a fail. When it finishes iterating without failing, it returns a pass. For example, if the acceptable margin of error is defined as [0, 1e-05] and an error occurs at a margin of [0, 1e-03], we can consider that the error has occurred at a wider margin, confirming that interoperability has not been achieved. If an error occurs only within [0, 1e-06], we can consider that the error has occurred at an acceptable margin, confirming that interoperability has been achieved. If it is confirmed that the post-conversion model preserved the inference operations, then we can proceed to the next step to perform the following verifications. Otherwise, it is confirmed that their interoperability has not been achieved.

3.4 Step4: Time Analysis

In this step, by analyzing the inference time of the target models, their interoperability is verified. Even if pre- and post-conversion DL models had produced the same inference values, the inference time may not have been preserved. Inference time is a key metric of a DL model's performance. When inference time noticeably increases after conversion, it is typically considered that interoperability has not been achieved. The formula for calculating time difference (=$TimeDiff$) is as follows:

$$TimeDiff = \frac{1}{n}\sum_{t=1}^{n}(T2_t - T1_t) \qquad (2)$$

where, for each data t among n total test data, $T1_t$ is an inference time of pre-conversion model (=$M1$), $T2_t$ is an inference time of post-conversion model (=$M2$).

After calculating *TimeDiff*, it is compared with a threshold, which can be empirically determined. If *TimeDiff* is greater than the threshold, we can consider that the two models require significantly different inference times, confirming that interoperability has not been achieved. If it is confirmed that the post-conversion model preserved the inference time, then we can finalize the verification process.

By passing all the validation and verification steps, the target DL models are confirmed to have achieved interoperability in terms of architecture, inference performance, operations, and time.

4 Case Study and Results

We conducted case studies to evaluate the effectiveness and applicability of our proposed method. We designed four different cases each of which represents a model conversion between different DL frameworks. Each case demonstrates how our method can be utilized to verify interoperability between pre- and post-conversion DL models. Our case studies mainly focused on the conversion scenarios of one model to various frameworks. To ensure representativeness, we selected a representative DL model and DL frameworks.

As a target model, we selected VGG16 [32], a popular DL model architecture for image classification. To train and test the model, we used two different datasets: (1) MNIST [33], a dataset consisting of handwritten images from 0 to 9; and (2) ImageNet [34], a large image dataset designed for visual object recognition. Moreover, we selected three representative DL frameworks, PyTorch, Keras, and TensorFlow. As DL conversion tools, we employed ONNX, NNEF, MMdnn, and pytorch2keras. As shown in Table 1, we designed four different cases of DL conversion: *Case#1*–a PyTorch model to a Keras model using ONNX, *Case#2*–a PyTorch model to a Keras model using pytorch2keras, *Case#3*–a PyTorch model to a TensorFlow model using NNEF, and *Case#4*–a Keras model to a PyTorch model using MMdnn.

Note that, for *Case#3*, we first converted a PyTorch model to ONNX format, then its result to NNEF format, and finally its results to a TensorFlow model because NNEF does not support PyTorch.

With the selected datasets (*Case#1* to *#3*: MNIST and *Case#4*: ImageNet), we trained four VGG16 models using each source DL framework (i.e., PyTorch and Keras), and then converted each model to each destination DL framework (i.e., Keras, TensorFlow, and PyTorch) using different DL conversion tools (i.e., ONNX, NNEF, MMdnn, and pytorch2keras). Note that, in *Case#4*, since MMdnn is basically compatible with the ImageNet dataset, we used it instead of MNIST. For each conversion (i.e., pre- and post-conversion DL models), we verified their interoperability using our proposed method. All experiments were conducted using Jupyter Notebook with the following settings: Python 3.7 and 3.8, ONNX 1.8.1 and 1.9.0, TensorFlow 1.15.2, Keras 2.2.4, PyTorch 1.5.1, and one GPU (NVIDIA Tesla T4).

Case#1: To perform architectural analysis, we first visualized the architecture of target DL models using Netron `(Step1-A)`. The visualized architecture, as shown in Figure 3(a), illustrated that the pre- and post-conversion models have the same layer types, connections, and configurations, except

Table 1. DL conversion cases in our case studies

Case	Source Framework (Pre-Conversion)	Destination Framework (Post-Conversion)	DL Conversion tool	Dataset
Case#1	PyTorch	Keras	ONNX	MNIST
Case#2	PyTorch	Keras	pytorch2keras	MNIST
Case#3	PyTorch	TensorFlow	NNEF	MNIST
Case#4	Keras	PyTorch	MMdnn	ImageNet

Table 2. The experimental results of case studies

	Step1 (Architectural Analysis)			**Step2 (Performance Analysis)**			**Step3 (Error Analysis*)**			**Step4 (Time Analysis**)**	
	Viz. Analysis	Detailed Inspection	**Interop.**	Accuracy (Pre-conv.)	Accuracy (Post-conv.)	**Interop.**	MAPE	Error Margin	**Interop.**	Time Difference	**Interop.**
Case#1	Pass	Pass	Pass	99.63%	99.63%	Pass	2e-04	[0,1e-06]	Pass	-0.045 sec.	Pass
Case#2	Pass	Pass	Pass	99.41%	99.41%	Pass	1e-04	[0,1e-06]	Pass	-0.277 sec.	Pass
Case#3	Pass	Fail	Fail	99.42%	Error	Fail	Error	Error	Fail	Error	Fail
Case#4	Fail	Fail	Fail	0.40%	0.18%	Fail	7e-01	[0,1e-02]	Fail	+0.303 sec.	Fail

* MAPE threshold: 0.1, Acceptable margin: [0, 1e-05]
** Time difference threshold: $<$+0.2sec.

for *"ZeroPadding2D"* layer. As mentioned in Section 3.1, since *"ZeroPadding2D"* layer has been added due to different characteristics between the DL frameworks, the model architecture is considered to have been maintained in this case. Furthermore, as shown in Figure 3(b), the detailed inspection demonstrated that no changes had occurred in the variables of each layer (`Step1-B`). After passing `Step1`, we compared the accuracy of the models using MNIST test dataset (`Step2`). The results confirmed that the accuracies of the pre- and post-conversion models were identical (=99.63%), as shown in Table 2.

The threshold for MAPE was set at 0.1 and the acceptable error margin was set at [0, 1e-05], both determined empirically. The time difference threshold was set at 0.2 seconds because this is the point at which users begin to notice a slowdown in mobile application responsiveness [35]. The results of case studies are summarized in Table 2.

After passing `Step2`, we measured MAPE of target models (`Step3-A`). The MAPE value (=0.00024) was smaller than the threshold (=0.1), confirming that interoperability had been achieved. In `Step3-B`, we measured the inference error of target models. As shown in Figure 4, there were 37 errors in [0, 1e-06], and none in [0, 1e-05], which was acceptable (acceptable error margin=[0, 1e-05]). Finally, in `Step4`, we calculated the time difference between the target models. The time difference value (-0.045 seconds) was smaller than the threshold ($<$0.2 seconds). Consequently, by passing the verifications at all steps, *Case#1*'s conversion was confirmed to have ensured interoperability.

Case#2: We first verified that the target models' layer types, connections, and configurations were preserved by visually inspecting their architectures (`Step1-A`). The detailed inspection also verified that no changes had occurred in the variables of each layer (`Step1-B`). We then compared the accuracy of the models using MNIST test dataset (`Step2`). The results confirmed that the accuracies of the pre- and post-conversion models were identical (=99.41%), as shown in Table 2. After passing `Step2`, we measured MAPE of target models (`Step3-A`). The MAPE value (=0.00011) was smaller than the threshold (=0.1), confirming

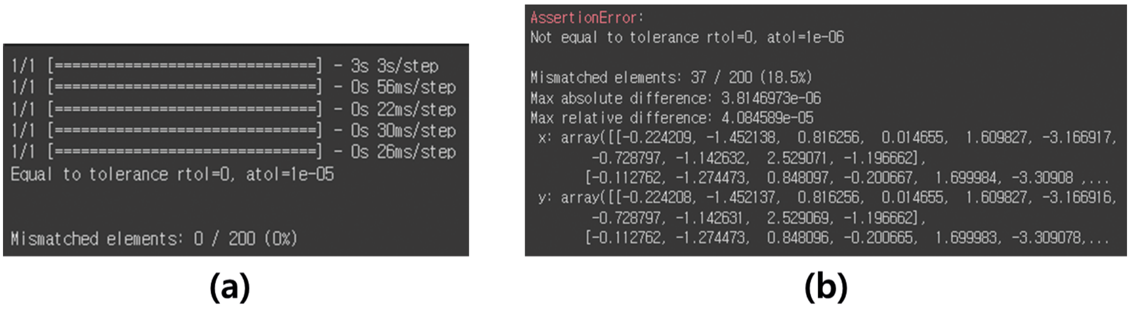

Figure 4. Result of Case#1's error margin analysis: (a) Inference errors in [0, 1e-05] and (b) Inference errors in [0, 1e-06] `(Step3)`

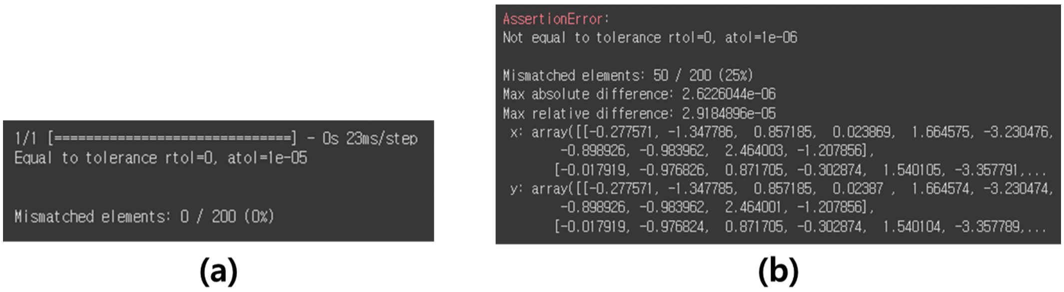

Figure 5. Result of *Case#2*'s error margin analysis: (a) Inference errors in [0, 1e-05] and (b) Inference errors in [0, 1e-06] `(Step3)`

that interoperability had been achieved. Finally, in `Step3-B`, we measured the inference error of target models. As shown in Figure 5, there were 50 errors in [0, 1e-06], and none in [0, 1e-05], which was acceptable (acceptable error margin=[0, 1e-05]). Finally, in `Step4`, we calculated the time difference between the target models. The time difference value (-0.277 seconds) was smaller than the threshold (<0.2 seconds). Consequently, by passing the verifications at all steps, *Case#2*'s conversion was confirmed to have ensured interoperability.

Case#3: To perform architectural analysis, we first visualized the architecture of target DL models `(Step1-A)`. The visualized architecture, as shown in Figure 6, illustrated that the post-conversion model contains additional layer types and connections while maintaining the order of layers from the pre-conversion model: *Conv2d-LeakyReLU-Conv2d-LeakyReLU-MaxPool2d*. Although we subsequently performed a detailed inspection `(Step1-B)`, a conversion error in the post-conversion model file precluded its proper execution. Due to the same error, this case could not pass the subsequent verifications either `(Step2 to Step4)`. Consequently, by failing the verifications at all steps, *Case#3*'s conversion was confirmed to not have ensured interoperability.

Case#4: To perform architectural analysis, we first visualized the architecture of target models `(Step1-A)`. As shown in Figure 7, while the number of layers was maintained, all connections between layers were removed in the post-conversion model's architecture. Moreover, in the post-conversion model, all layers other than *Conv2D* layers (i.e., *ReLU* and *MaxPooling2D*) have been removed. The result confirms that *Case#4* did not achieve interoperability. To confirm the validity of our proposed process, we additionally proceeded with the remaining steps. In `Step2`, we observed the difference in accuracy between target models (pre-conversion: 0.40% and post-conversion: 0.18%), indicating that inference performance was not preserved. In `Step3-A`, the measured MAPE was 0.74, which exceeded the threshold (=0.1). In `Step3-B`, we confirmed that two errors occurred in the range of [0, 1e-02], which was unacceptable. In `Step4`, we calculated the time difference between the target models. The time difference value (0.303

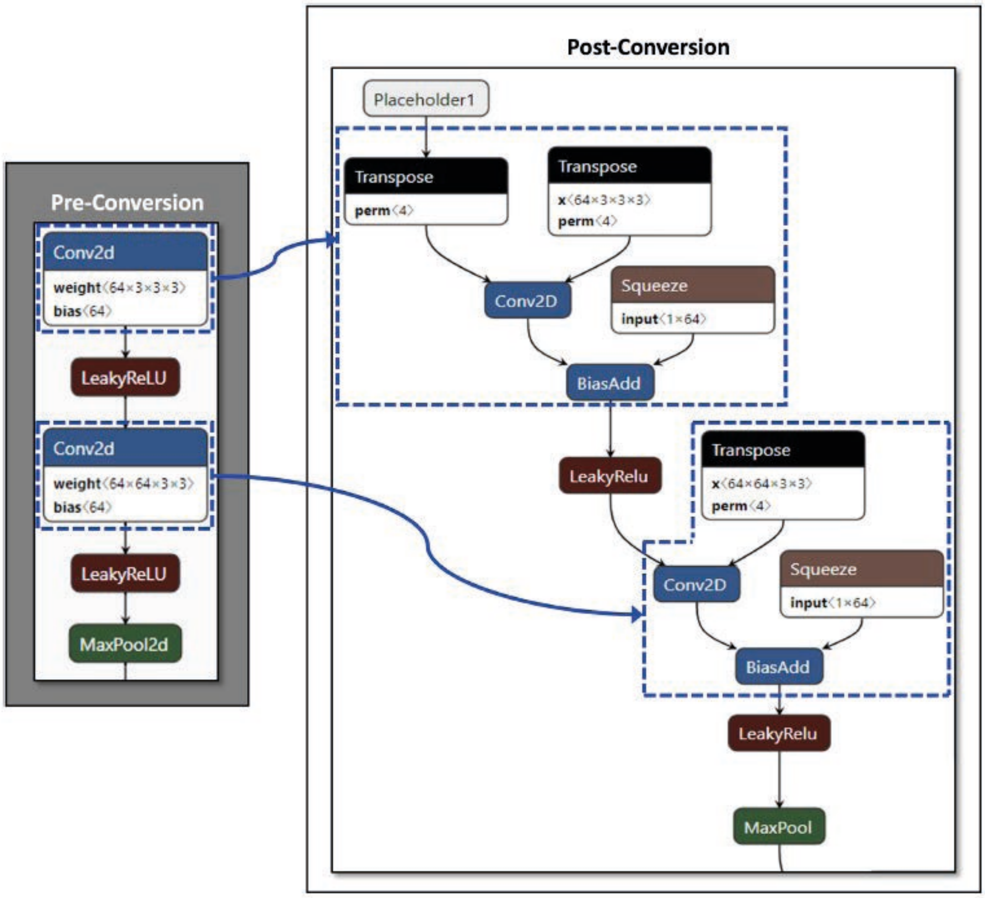

Figure 6. Excerpt from Case#3's architectural analysis

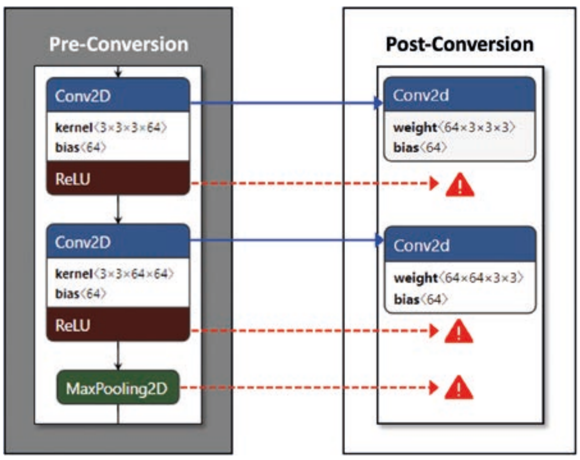

Figure 7. Excerpt from *Case#4*'s architectural analysis

seconds) was greater than the threshold (<0.2 seconds). Consequently, by failing the verifications at all steps, *Case#4*'s conversion was confirmed to not have ensured interoperability.

Through four different cases, we checked whether our proposed process was valid in verifying interoperability between DL models. Both in *Case#1* and *Case#2*, we confirmed that interoperability was achieved between the target DL models because no issues were identified at any step of our method. On the other hand, in both *Case#3* and *Case#4*, we confirmed that interoperability was not achieved between the target DL models because interoperability issues were identified at every step of our method. For all cases, our proposed method successfully discovered the interoperability issues that have been reported in DL model conversion [23, 22] (e.g., inference performance degradation or model architecture changes) via its multi-perspective validation and verification.

5 Threats to validity

In our case studies, to address any resulting bias, we carefully selected DL models, DL frameworks, datasets, and DL conversion tools by focusing on their popularity and generality. As a target DL model architecture, VGG16 was selected because it has a simplified architecture with only 16 elemental layers, which enables a precise comparison between the pre- and post-conversion DL models. As target DL frameworks, out of the top 10 most popular DL frameworks in 2021 [36, 37], the top three were chosen: PyTorch, TensorFlow, and Keras. Moreover, to enable reproducible analysis, MNIST, and ImageNet, representative image classification datasets, were selected as target datasets because they required minimal preprocessing for training and testing. For target DL conversion tools, among the open-source tools published on GitHub, we selected four tools based on the rankings (for representativeness) and the number of commits and contributors (for stability): ONNX, NNEF, MMdnn, and pytorch2keras. In addition, we plan to conduct additional experiments by securing more conversion cases.

Especially in `Step1`, due to the qualitative aspect of architectural analysis, its results may be different depending on an engineer's expertise or the complexity of model architecture. For example, a non-expert engineer is more likely to make errors in examining complex model architectures. To minimize this, in our case studies, two expert engineers (with more than three years of experience in developing DL software) additionally performed a peer-review on our analysis results.

6 Conclusion

DL model conversion has enabled engineers to deploy DL models in various DL frameworks, but interoperability between the models may not be guaranteed in many cases due to their unreliable conversions. In this paper, we proposed a new method for ensuring interoperability between DL models based on the V&V approach. Our method validates and verifies the interoperability between target DL models by conducting systematic analyses from multiple perspectives: architectural analysis, performance analysis, error analysis, and time analysis. We conducted case studies on four different scenarios of DL model conversion, and confirmed that our method successfully discovered the interoperability issues that have been reported. As the versions, types, and environments of DL frameworks diversify, reliable DL model conversion becomes increasingly important. By enabling systematic validation and verification of DL conversion, our proposed solution can significantly contribute to ensuring interoperability between DL models.

As a future work, we plan to develop an automated method for architectural analysis. We will develop a mechanism that automatically compares the layers and parameters in target DL models and quantitatively measures the differences. Moreover, by examining additional conversion cases, we plan to develop a mechanism that automatically adapts DL models to be deployed in different environments.

Acknowledgement

This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2022-00165648); in part by BrainLink program funded by the Ministry of Science and ICT through the National Research Foundation of Korea (RS-2023-

00237308); and in part by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2021-0-00766, Development of Integrated Development Framework that supports Automatic Neural Network Generation and Deployment optimized for Runtime Environment).

References

[1] G. Nguyen, S. Dlugolinsky, M. Bobák, V. Tran, Á. López García, I. Heredia, P. Malík, and L. Hluchý, Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey, Artificial Intelligence Review, 52, 2019, pages 77-124.

[2] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M.P. Reyes, M.L. Shyu, S.C. Chen, S.S. and Iyengar, A survey on deep learning: Algorithms, techniques, and applications, ACM Computing Surveys (CSUR), 51(5), 2018, pages 1-36.

[3] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, and S. Ghemawat, Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2016, arXiv:1603.04467.

[4] F. Chollet, Keras: The python deep learning library, 2015, https://github.com/fchollet/keras.

[5] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, and A. Desmaison, Pytorch: An imperative style, high-performance deep learning library, Advances in neural information processing systems, 32, 2019.

[6] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, A survey of model compression and acceleration for deep neural networks, 2017, arXiv:1710.09282.

[7] T.G. Dietterich, Ensemble learning, In: The handbook of brain theory and neural networks, The MIT Press, 2002, pages 110-125.

[8] J. Gao, P. Li, Z. Chen, and J. Zhang, A survey on deep learning for multimodal data fusion, Neural Computation, 32(5), 2020, pages 829-864.

[9] Facebook and Microsoft, ONNX: Open Neural Network Exchange, 2017, https://github.com/onnx/onnx.

[10] The Khronos Group, Neural Network Exchange Format (NNEF), 2016, https://www.khronos.org/nnef.

[11] Y. Liu, C. Chen, R. Zhang, T. Qin, X. Ji, H. Lin, and M. Yang, Enhancing the interoperability between deep learning frameworks by model conversion, In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2020, pages 1320-1330.

[12] Hahnyuan, Neural network tools: Converter and analyzer, 2017, https://github.com/hahnyuan/nn_tools.

[13] Gmalivenko, pytorch2keras: Pytorch to keras model convertor, 2019, https://github.com/gmalivenko/pytorch2keras.

[14] Z. Chen, Y. Cao, Y. Liu, H. Wang, T. Xie, and X. Liu, A comprehensive study on challenges in deploying deep learning based software, In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2020, pages 750-762.

[15] M. Openja, A. Nikanjam, A.H. Yahmed, F. Khomh, and Z.M.J. Jiang, An empirical study of challenges in converting deep learning models, In 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2022, pages 13-23.

[16] H. Pham, Software reliability, Springer Science& Business Media, 2000.

[17] J. Schumann, P. Gupta, and S. Nelson, On verification & validation of neural network based controllers, EANN'03, 2003.

[18] Facebook, CAFFE2, 2017, https://caffe2.ai/.

[19] Apache Software Foundation, A flexible and efficient library for Deep Learning, 2017, https://mxnet.apache.org/versions/1.9.1/.

[20] Y. Jia, E. Shelhamer, F. Donahue, S. Karayev, K. Long, R. Girshick, S. Guadarrama, and T. Darrell, Caffe: Convolutional architecture for fast feature embedding, In Proceedings of the 22nd ACM international conference on Multimedia, 2014, pages 675-678.

[21] Woodsgao, pytorch2caffe, 2010, https://github.com/woodsgao/pytorch2caffe.

[22] Darshan, Torch to ONNX conversion going wrong, 2021, https://discuss.pytorch.org/t/torch-to-onnx-conversion-going-wrong/121596.

[23] ys.yusaito, Inference result is different between Pytorch and ONNX model, 2022, https://discuss.pytorch.org/t/inference-result-is-different-between-pytorch-and-onnx-model/147228/1.

[24] J. Schumann, and K. Goseva-Popstojanova, Verification and validation approaches for model-based software engineering, In 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2019, pages 514-518.

[25] L. Roeder, Netron, 2020, https://netron.app/.

[26] T. Fawcett, An introduction to ROC analysis. Pattern recognition letters, 27(8), 2006, pages 861-874.

[27] Y. Jiang, Prediction of monthly mean daily diffuse solar radiation using artificial neural networks and comparison with other empirical models, Energy policy, 36(10), 2008, pages 3833-3837.

[28] S. Zeiml, U. Seiler, K. Altendorfer, and T. Felberbauer, Simulation evaluation of automated forecast error correction based on mean percentage error, In 2020 Winter Simulation Conference (WSC), 2020, pages 1572-1583.

[29] M. Nejadgholi, and J. Yang, A study of oracle approximations in testing deep learning libraries, In 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2019, pages 785-796.

[30] M.V. Shcherbakov, A. Brebels, N.L. Shcherbakova, A.P. Tyukov, T.A. Janovsky, and V.A.E. Kamaev, A survey of forecast error measures, World applied sciences journal, 24(24), 2013, pages 171-176.

[31] H. Li, W. Ma, Y. Lian, and X. Wang, Estimating daily global solar radiation by day of year in China, Applied Energy, 87(10), 2010, pages 3011-3017.

[32] K. Simonyan, and A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014, arXiv:1409.1556.

[33] L. Deng, The mnist database of handwritten digit images for machine learning research [best of the web], IEEE signal processing magazine, 29(6), 2012, pages 141-142.

[34] A. Krizhevsky, I. Sutskever, and G.E. Hinton, Imagenet classification with deep convolutional neural networks, Communications of the ACM, 60(6), 2017, pages 84-90.

[35] Android Google, Keeping your app responsive, 2017, https://developer.android.com/training/articles/perf-anr.html.

[36] G. Kechit, Top 10 deep learning frameworks in 2022 you can't ignore, 2022, https://www.upgrad.com/blog/top-deep-learning-frameworks.

[37] O.G. Yalçın, Top 5 Deep Learning Frameworks to Watch in 2021 and Why TensorFlow, 2021, https://towardsdatascience.com/top-5-deep-learning-frameworks-to-watch-in-2021-and-why-tensorflow-98d8d6667351.

Youn Kyu Lee is currently an Assistant Professor in the Department of Computer Engineering at Hongik University, Seoul, Korea. He received the B.S. and M.S. degrees in computer science and engineering from Korea University, Seoul, Korea, in 2010 and 2012, respectively; and the Ph.D. degree in computer science from the University of Southern California (USC), Los Angeles, CA, USA, in 2017. Before joining Hongik University, he was with Samsung Advanced Institute of Technology (Suwon, Korea, 2018-2020), and Seoul Women's University (Seoul, Korea, 2020-2021). He was a recipient of Viterbi Graduate Fellowship with his Ph.D. admission from USC (2012), IEEE/ACM International Conference on Automated Software Engineering (ASE) Best Tool Paper Award (2018) and IEEE ICTC Best Paper Award (2022).
https://orcid.org/0000-0002-4569-2640

Seong Hee Park is currently pursuing the M.S. degree in Computer Engineering at Hongik University, Seoul, Republic of Korea. She received the B.S. degree in information security at Seoul Women's University, Seoul, Republic of Korea. Her research focuses include deep learning algorithms and their applications to information security.
https://orcid.org/0000-0002-6088-0658

Min Young Lim is currently pursuing the M.S. degree in Computer Engineering at Hongik University, Seoul, Republic of Korea. She received the B.S. degree in information security at Seoul Women's University, Seoul, Republic of Korea. Her research focuses include deep learning algorithms and their applications to AI security. She was a recipient of IEEE ICTC Best Paper Award (2022).
https://orcid.org/0000-0001-8016-5945

Soo-Hyun Lee is currently pursuing the M.S. degree in Computer Engineering at Hongik University, Seoul, Republic of Korea. She received the B.S. degree in information security at Seoul Women's University, Seoul, Republic of Korea. Her research focuses include deep learning algorithms and information security.
https://orcid.org/0000-0001-6696-0387

Jongwook Jeong is currently an Assistant Professor at the Department of Computer Science and Artificial Intelligence in Jeonbuk National University, Jeonju, Korea. He received his Ph.D. in Computer Engineering from Korea University, Seoul, Korea. His current research interests include software usability, software testing and user requirements.
https://orcid.org/0000-0002-3307-0940