
This is the author version of the work deposited here under a Creative Commons license: https://creativecommons.org/licenses/by/4.0/

Copyright © 2023 IEEE

This is the author version of the work. There may be differences between this version and the published version. You are advised to consult the published version if you wish to cite from it: https://doi.org/10.1109/ASE56229.2023.00147

https://eprints.gla.ac.uk/303713/

Deposited on 28 July 2023

Enlighten – Research publications by members of the University of Glasgow http://eprints.gla.ac.uk
Fault Localization for Buggy Deep Learning Framework Conversions in Image Recognition

Abstract—When deploying Deep Neural Networks (DNNs), developers often convert models from one deep learning framework to another (e.g., TensorFlow to PyTorch). However, this process is error-prone and can impact target model accuracy. To identify the extent of such impact, we perform and briefly present a differential analysis against three DNNs used for image recognition (MobileNetV2, ResNet101, and InceptionV3), converted across four well-known deep learning frameworks (PyTorch, Keras, TensorFlow (TF), and TFLite), which revealed numerous model crashes and output label discrepancies of up to 100%. To mitigate such errors, we present a novel approach towards fault localization and repair of buggy deep learning framework conversions, focusing on pre-trained image recognition models. Our technique consists of four primary stages of analysis: 1) conversion tools, 2) model parameters, 3) model hyperparameters, and 4) graph representation. In addition, we propose a number of strategies towards fault repair of the faults detected.

We implement our technique on top of Apache TVM deep learning compiler, and we test it by conducting a preliminary fault localization analysis for the conversion of InceptionV3, from TF to TFLite. Our approach detected that the tf2onnx tool used in the conversion process introduced precision errors to model weights for convolutional layers in particular, which negatively affected the model accuracy. We then repaired the target model by replacing the affected weights with those from source model.

I. INTRODUCTION

A deep learning (DL) model, trained to work with a given framework (such as PyTorch [1], TensorFlow [2]), may be converted to a different framework (such as Keras, TensorFlow) for deployment, and sometimes to its lightweight counterpart (such as TensorFlow Lite, PyTorch Mobile) to support inferences on target environments that may be more resource constrained such as mobile or IoT devices. Conversion of DNN models between DL frameworks is facilitated by automated conversion processes using tools like tf2onnx [3], onnx2keras [4], onnx2torch [5] and MMDnn [6]. The conversion process can, however, contain certain faults [7]–[9] that can make the translated models undeployable [10], [11].

In order to mitigate this problem, we propose an approach for fault localization and repair of faults introduced by the DL framework conversion process. We focus on DL framework conversion used in deployment of pre-trained image recognition models, utilized for image classification tasks. Our approach detects faults introduced in model parameters, hyperparameters, but also in the model graph, due to faulty transformations applied in the model conversion process. In particular, our proposed approach performs analysis and comparison against source and target model parameters and hyperparameters, as well as comparison of layer activations for inputs resulting in output label discrepancies against the source and the target model. Additionally, we explore potential discrepancies introduced by graph transformations between the source and the target model. Subsequently, we propose a set of strategies for the mitigation of conversion faults such as the replacement of model parameters of the target model with those from source, and applying graph transformations that eliminate the error from the converted model. Finally, we present a preliminary evaluation on the conversion process of InceptionV3, converted from TensorFlow to TFLite. Our technique was able to detect precision errors in weights related to convolutional layers introduced by tf2onnx tool, with value deviations of up to 0.01 between Source and Target, which, although negligible, they affected the model performance.

We make the following contributions: 1) A novel method to systematically localize faults in DL framework conversion process, and 2) Repair strategies for conversion faults.

II. RELATED WORK

A number of studies have been conducted in relation to the faults introduced in the deployment process of DNNs. In particular, a comprehensive study [10] has been conducted based on 3023 Stack Overflow posts, building a taxonomy or faults and highlighting the difficulty of the DNN deployment procedure. Another study [11] explores the effect of DNN faults on mobile devices by identifying 304 faults from GitHub and StackOverflow. In terms of fault localization, DeepCover [12] attempts to apply a statistical fault localization approach, focusing on the extraction of heatmap explanations from DNN inputs. DeepFault [13] focuses on a suspiciousness-oriented spectrum analysis algorithm in order to detect parts of the DNN that can be responsible for faults, while it also proposes a method for adversarial input generation. DeepLocalize [14] attempts to detect faults in DNNs by converting it to an imperative representation and then performing dynamic analysis on top of its execution traces.

In relation to DL framework fault localization, CRADLE [15] tries to detect faults introduced by DL frameworks by performing model execution graph analysis. LEMON [16] is based on the metrics used by CRADLE for its analysis to apply mutation testing.
Although all of the above works attempt to overcome fault localization challenges for DNNs, none of them considers model conversions as a factor of fault introduction in DNNs and, therefore, no previous work explores this domain. However, several tools exist to ease the DL model conversion process, namely, Mmdnn [17], tf2onnx [3], onnx2keras [4], onnx2torch [5], tensorflow-onnx [18] along with the native APIs for DL framework conversion from PyTorch [1] and TFLite [2] to ONNX [19]. All these projects are extensively used, as they contain more than 100 stars on their GitHub repositories. In addition, recent study by Openja et al. [20] highlights the challenges of the conversion process, while a preliminary study by Louloudakis et al. [7] explores the robustness of DNNs against different computational environment aspects, including DL framework conversions. However, the impact of DL framework conversions on DNN model correctness is not explored in-depth in the literature.

To the best of our knowledge, this paper is the first attempt focusing on the error proneness, fault localization and repair of DL framework conversions for DNN models. We focus on image recognition models as a a starting point, but our work is applicable to DNNs used in other domains.

### III. Motivation

To observe the potential impact of model conversions, we conducted an initial evaluation using three image recognition DNN models – MobileNetV2 [21], ResNet101 [22] and InceptionV3 [23]. For each model, we used pre-trained versions from official repositories of four different DL frameworks – TensorFlow [2], TFLite [2], Keras [24] and PyTorch [1]. We refer to the pre-trained model of each DL framework as the Source model. As a result, we have 4 Source versions for each of our 3 models. We then convert each Source model to use a different DL framework; we refer to the converted model as Target. To implement the conversion, we use tools that convert the Source model to the ONNX [19] format, a popular model representation format that is designed to bridge the gap between frameworks. Some DL frameworks, such as PyTorch, have native tools for this conversion; whereas for others, such as TensorFlow, we leverage popular third-party conversion tools like tf2onnx [3] and tensorflow-onnx [18]. We then convert from ONNX to Target using a number of widely used libraries, such as onnx2keras [4], and onnx2torch [5].

Following the conversion process, we performed pairwise comparison between Source and Target model inferences using the ILSVRC 2017 object detection dataset [25]. We compare output labels of Target against the Source for each image to check if any errors were introduced by model conversion. The proportion of output label dissimilarities between Source, Target pairs across all images in the dataset is shown in Figure 1. As can be seen from the empty grey boxes, the conversion tool crashes in 11 out of the 36 conversions across the three DNN models, indicating that the conversion process failed. This happened due to compatibility issues between the conversion tool and a given model architecture, or the Source or Target DL framework. Additionally, we observe further 10 cases where the conversion succeeded without crashing, but the Target model gave entirely different labels from the Source model. We also observe a 66% discrepancy in the output labels when converting the ONNXMobileNetV2 model from PyTorch to Keras. The conversion of PyTorch models to TF gives varying results across models, with MobileNetV2 having no dissimilarity, ResNet101 having a considerable amount of dissimilarity (45%), and InceptionV3 having 100% dissimilarity. This points to weaknesses in the conversion tool with certain model architectures. Finally, for conversions between TF to TFLite across all models, we see relatively small discrepancies, 0 – 10%, demonstrating a more reliable conversion. However, even small amounts of discrepancies can cause safety concerns when these models are used in safety critical applications.

From the results in Figure 1, it is clear that the conversion process is error-prone and there is a need for a technique to localize and fix faults introduced by DL frameworks converters. We discuss our approach for fault localization and repair in the next section.

### IV. Methodology

The stages in our proposed approach for fault localization and repair are shown in Figure 2. It starts by converting a given model from Source to Target DL framework, inference over both models over an input dataset, comparison of output labels to identify parts of the dataset that led to different outputs, and finally the fault localization and repair process. We describe the fault localization and repair steps below.

#### A. Fault Localization & Repair

We first start by examining the tools involved in the model conversion to identify if the fault is introduced during conver-
sion from Source to ONNX intermediate representation or from ONNX to Target. We then complement this analysis by examining differences in three key DNN model architecture aspects: 1) Hyperparameters – such as pooling and batch size, 2) Parameters – such as layer weights and biases, and 3) Graph architecture representing model layers (such as convolutions and activation functions) and their structure. We describe these steps in the Subsections below.

1) Tools Inference Analysis: Following the model conversion process, when discrepancies are observed between the Source and the Target model, it is important to identify which part of the process was responsible. The conversion process typically uses more than one tool – e.g., use one tool to convert to ONNX format from Source, and another tool to convert from ONNX to Target. We explore this over the sample of images producing different inference results between Source and Target while also considering the intermediate ONNX representation. In particular, we consider a subset of dataset inputs that presented different outputs between Source and Target models. In addition, we perform inference using the ONNX intermediate representation of the conversion process, and compare the outputs against the Source and the Target. If the conversion process involves multiple steps, we repeat the process for all intermediate ONNX representation of the steps. The aim of this additional step, is to determine the part of the conversion process (the Source-to-ONNX, or the ONNX-to-Target converter that introduced the discrepancies.

2) Parameters Analysis: A correct model conversion should result in a target model having the same parameters, and produce the same output with the source model. However, if for some reason the parameters are altered (e.g., due to a precision error in the conversion process), this could potentially affect the model correctness.

In order to detect this fault, we take the Source and Target model variants, and extract their parameters from the model graph metadata. We then compare the model parameters across layers of the same type (e.g., convolutions, bias additions, etc.), by computing mean(abs(Psource - Ptarget)), where Psource/Ptarget are the parameters of the source and converted target models, respectively. The layer type selection is based on layers playing a key role in the DNN architecture (e.g., Convolutions), and therefore they are expected to remain unchanged in the conversion process. The value of this computation is expected to be zero when the model parameters are unaffected, and any other value indicates that there is a difference across the parameters in a specific layer, which is a potential cause for bugs.

3) Hyperparameter Analysis: Much like parameters, incorrectly converted hyperparameters are another potential source of error that we examine. For example, we would expect for a convolutional layer, padding, strides, dilation, and other configurations to remain unchanged. However, if we find a difference, this could indicate a potential source of error and is marked for further evaluation in our fault localization approach.

4) Layer Analysis: To detect faults that occur on a specific layer, we propose a layer-based dynamic analysis approach. Using a subset of the images that cause output label discrepancies between Source and Target models, we perform inference and compare per-layer activations between the models. For each input, we compute the mean of differences found across activations for each layer. We then further examine the layers affected sequentially, starting from the first layer and moving forward. We focus on errors on graph representation concerning that layer, as well as on implementation details. In particular, we examine if a layer or its graph neighbours are implemented in a different manner or are using different functions between the Source and Target model.

5) Fault Repair: Once a difference is detected, we attempt one of the following options based on the location of the difference for fault repair.

(a) For differences in model hyperparameters, weights and biases, the respective values from source can be replaced to target model. Since the conversion process should preserve those values, then the replacement in the target model should resolve the observed differences.

(b) For differences detected in layer activations, there are a number of measures that can be applied. First, a set of mappings can be applied in order to apply in-place replacement of parts of the graph that should behave similarly, but differences in implementation (such as the selection of a different layer type, or the addition of extra redundant layers) could cause differences in layer outputs. For example, we observed cases (e.g., MobileNetV2, PyTorch to Keras conversion) where the Flatten layer was replaced by a Reshape layer by the converter tools. We will instruct a layer replacement to the target based on the layer of the source model, while adjusting tensor inputs and outputs to preserve model validity. In addition, if there are extra nodes added close to the layer affected, they could be modified and removed as an attempt to eliminate errors. For instance, we observed the addition of some Padding layers to the target model for a number of conversions (e.g., MobileNetV2, TF to PyTorch conversion). A potential fix is to simply remove this node. Our current approach has limitations towards the cases where whole sub-graphs in the target model have completely different structure than the source. A replacement in this scenario is non-straightforward. We will consider this case in future work.

Once a fix is applied, inference is performed with the target model against the inputs causing discrepancies, and the behaviour is monitored. If an improved result is detected for some or all of the images under test, then the fix is considered as successful.

B. Implementation Details

Our methodology is implemented using Apache TVM [26]. We use the tool in order to build and perform inference for the Source and Target models, while we extract the graph parameters, graph structure provided by each model for weights, biases and hyperparameters, utilizing the model static parameters and graph description metadata generated
across the build process. We also use ONNXRuntime [27] in order to perform intermediate representation inference. In addition, we utilize the TVM Debugger to extract layer activations upon inference, as well as set specific inputs and extracting targeted outputs from hidden layers. TVM Debugger was also used in order to achieve model repair strategies, such as replacing weights, biases and hyperparameters. For the graph modification part, we utilized the ONNX [19] API in combination with ONNXModifier [28] in order to apply graph modifications. We also used the Netron [29] for DNN graph observation purposes.

C. Preliminary Evaluation

As an initial case study, we consider library conversion of InceptionV3 using TensorFlow (TF) framework as Source and converting it to TFLite as Target. The conversion involved two tools, tf2onnx and the native API of TFLite for conversion from ONNX. We observed label differences between Source and Target models for 4% of the input images (240 out of 5500 images). We were interested in this particular case study as the conversion error is not too obvious but still present in a small number of images. We believe this case study is non-trivial for fault localization and repair owing to this small difference.

For fault localization, we start by performing an analysis of the conversion tools on the images showing label differences. As seen in Table I for three sample images, label difference occurs when converting TF model to ONNX while the subsequent step of ONNX to TFLite does not present label difference. As a result, we find that the tf2onnx tool is the problematic part of the conversion process. We perform further investigation into this tool in the next steps.

<table>
<thead>
<tr>
<th>Image ID</th>
<th>TF</th>
<th>ONNX</th>
<th>TFLite (TF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image 1</td>
<td>drum</td>
<td>drum</td>
<td>drum</td>
</tr>
<tr>
<td>Image 2</td>
<td>wallet</td>
<td>purse</td>
<td>purse</td>
</tr>
<tr>
<td>Image 3</td>
<td>wallaby</td>
<td>lt. greyhound</td>
<td>lt. greyhound</td>
</tr>
</tbody>
</table>

We then proceed with parameters and layers analysis between Source and Target to further examine the effects and the potential reasons of the problem to the converted Target model.

We perform parameters and layer activations comparison for an image presenting no discrepancies and two images presenting minor and major label discrepancies, respectively. We present the results in Figure 3, where the Parameters line, indicates the mean of differences per-layer (x-axis) in parameters, for two types of layers - convolutions and bias additions. The remaining lines depict the differences in activations (mean of tensor values comparison) for each layer, between Source and Target. Image 1 presented no discrepancies, Image 2 presented a small deviation, and Image 3, a large deviation. We can observe that layer 2 started presenting discrepancies, but also that layers 170 onwards followed a spike for the image posing great discrepancies. This indicates that 1) The model was affected by a fault early in the process, and 2) layers 170 onwards were cumulatively affected by a larger extent from the error. We examined if the cause was errors introduced in the model weights during the tf2onnx conversion step. In particular, we performed a manual source and target model parameters inspection using Netron [29], which confirmed the fault localization finding – as we observed precision errors in the generated ONNX graph from Source.

In order to fix the error, we replaced the model weights of Target model with those from Source, and we performed inference against the subset of images presenting discrepancies between the models. The inference outputs of updated Target was then identical with the original Source, and therefore the issue was resolved.

V. Conclusions and Future Work

We have presented a novel fault localization technique for errors encountered during DL framework conversion in image recognition models. The fault localization approach focuses on key model architecture elements, such as parameters, hyperparameters and graph architecture. We also propose strategies to repair the detected errors. To support our methodology, we examined the performance degradation on InceptionV3 when converted from TF to TFLite, which resulted in discrepancies for a small fraction of input images. We used our approach to localize the conversion bug and fix it. As future work, we aim to evaluate our methodology against all conversion tools in Figure 1 and other image recognition models. We will also apply our approach to other DL tasks such as object detection. Finally, we plan to expand our fault repair strategies to address conversion errors that cause significant changes in Source and Target model graphs.
REFERENCES


