Assessing Robustness of Image Recognition Models to Changes in the Computational Environment*

Nikolaos Louloudakis
University of Edinburgh
n.louloudakis@ed.ac.uk

Perry Gibson
University of Glasgow
p.gibson.2@research.gla.ac.uk

José Cano
University of Glasgow
jose.canoreyes@glasgow.ac.uk

Ajitha Rajan
University of Edinburgh
arajan@ed.ac.uk

Abstract

Image recognition tasks typically use deep learning and require enormous processing power, thus relying on hardware accelerators like GPUs and TPUs for fast, timely processing. Failure in real-time image recognition tasks can occur due to incorrect mapping on hardware accelerators, which may lead to timing uncertainty and incorrect behavior. In addition, the increasing demand for optimal performance has led to progress towards the optimization of different neural network operations, such as operator fusion. Owing to the increased use of image recognition tasks in safety-critical applications like autonomous driving and medical imaging, it is imperative to assess the performance and impact of such optimizations, and explore their effectiveness.

In this paper we conduct robustness analysis of four popular image recognition models with the ImageNet dataset, assessing the impact of the compiler optimizations applied, utilizing different Deep Learning frameworks and executing on hardware devices of varying capabilities. Our results indicate output label discrepancies of up to 37% across deep learning framework conversions, and up to 81.8% unexpected performance degradation upon application of compiler optimizations.

1 Introduction

Much of the existing literature for assessing robustness and safety of image recognition has focused on testing the Deep Neural Network (DNN) structure and addressing bias in the training dataset through adversarial testing and data augmentation [35, 28, 12]. Existing techniques failed to consider safety violations caused by interactions of the DNN with the underlying computational environment: both software and hardware. This can include the Deep Learning (DL) frameworks (e.g., TensorFlow, PyTorch, etc), compiler optimizations for device code generation (e.g., operator fusion, loop unrolling, etc), and the hardware platforms they run on (e.g., CPUs, GPUs, etc).

In this paper, we conduct an empirical study to evaluate the robustness of image recognition models in the presence of changes in specific aspects of the computational environment. We consider the following two parameters in the computational environment: (1) Conversions between DL frameworks, i.e. transforming a model defined in one DL framework to the model format of another framework; (2) Compiler optimizations, considering different levels of optimizations when generating device code. We utilize a range of GPU accelerator devices for that purpose. We assess the robustness of four widely used image recognition models with respect to the output label classification and inference time when changing each of the two environment parameters. In total, we observe up to 37% discrepancies across DL framework conversions, and up to 81.8% unexpected performance degradation when utilizing compiler optimizations.

*This work was supported by Royal Society Industry Fellowship, “AutoTest: Testing Autonomous Vehicle Perception Safety on Hardware Accelerators".

ML Safety Workshop, 36th Conference on Neural Information Processing Systems (NeurIPS 2022).

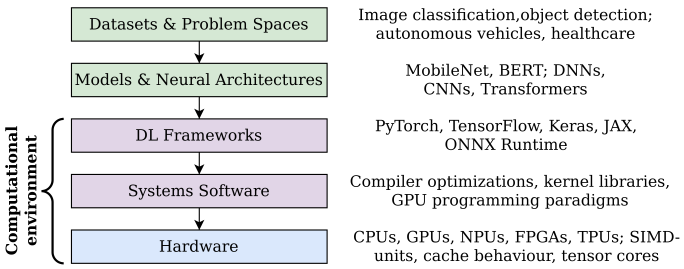


Figure 1: Deep learning systems stack showing layers in the computational environment.

2 Related Work

Existing work on DNN reliability has primarily focused on the robustness of the dataset and model architecture layers (first two layers) of the deep learning systems stack (Figure 1), but also in the effect of using different DL frameworks (third layer) for training and executing DNNs. Works like DeepHunter [33], DeepTest [28], DeepRoad [35], and DeepBillboard [37] focus on adversarial testing, while DeepXPlore [20] applies whitebox testing. For a more comprehensive overview of adversarial examples for images, we refer the readers to a survey [26]. In addition, there have been explorations of fault detection in the level of DL frameworks, such as CRADLE [21] and LEMON [31]. Several works have also explored benchmarking DL Frameworks with respect to model accuracy and training time [25, 17, 18, 32].

We consider *conversions between DL frameworks* as a parameter which has not been adequately explored in existing work. We believe this is relevant, as the rise of DL frameworks such as TFLite and TensorRT have shown that the developer community is looking at deployment specific DL frameworks, which may be different from more training focused frameworks such as TensorFlow and PyTorch. Thus, studying the impact of conversion between frameworks reflects a real risk that application developers may face.

In addition, an existing study [24] analyzes already discovered bugs, introduced by different deep learning compilers and uses them as the base to propose new mutation operators, supporting that optimization code logic is accounted for 9% of the bugs introduced by compilers. We also examine the effect of changing compiler optimizations within a specific deep learning compiler (Apache TVM [5]) on model performance. However, we do not focus on bugs already detected; our aim is to comprehensively explore the effects of these optimizations under realistic conditions, by executing models under different optimization settings utilizing a challenging dataset.

3 Experiments

We consider four different DNN (image recognition) models: MobileNetV2 [23], ResNet101V2 [13] [14], DenseNet121 [15], and InceptionV3 [27]. We use models pre-trained on ImageNet [11], using official model definitions and weights sourced from 4 different Deep Learning framework repositories : *Keras* [9], *PyTorch* [19], *TensorFlow* [3], and *TFLite* [3]. Each model is evaluated in the Apache TVM compiler framework (v0.8.0), imported via ONNX [2] (a format utilized for machine learning model representation) with all possible combinations of values for each of the following environment parameters.

DL Framework Conversions: We consider TensorFlow and PyTorch as sources for our models and we convert each of these models to TFLite.
Compiler Optimizations: We explored the impact of different levels of TVM graph-level compiler optimizations: basic, default, and extended variants. **Basic** (`o0`) applies only "inference simplification", which generates simplified expressions with the same semantic equivalence as the original DNN. **Default** (`o2`) applies all optimizations of `o0`, as well as operator fusion for operations such as ReLU activation functions, constant and scale axis folding. **Extended** optimization (`o4`) applies all optimizations from Default, as well as additional ones such as eliminating common subexpressions, applying canonicalization of operations, combining parallel convolutions, dense matrix and batch matrix multiplication operations, and enabling "fast math" (which allows the compiler to break strict IEEE standard compliance for float operations if it could improve performance). We examine all models sourced from the 4 different DL frameworks .
Devices: We used four different hardware devices, featuring high-end to low-end GPU accelerators: an Intel-based server featuring an Nvidia Tesla K40c (GK11BGL) GPU (*Server*), a Nvidia AGX Xavier featuring an Nvidia Volta GPU (*Xavier*), a Laptop featuring an Intel(R) GEN9 HD Graphics

NEO (Local) and a mobile-class Hikey 970 board featuring an Arm Mali-G72 GPU (*Hikey*). For the Xavier device we use TVM to generate `CUDA` code, while for the rest we utilize `OpenCL` code.

In total, we evaluate 96 model variants, as a combination of 4 models, 2 DL framework conversions, 4 devices, and 3 optimization levels. We discuss the challenges faced in compiling and executing certain configurations in Section 3.3.

3.1 Dataset

We use the ImageNet object detection test dataset [22] in our experiments, consisting of 5500 RGB images of 224×224 pixels, performing classification of 1000 possible labels and measure inference time on each image.

3.2 Robustness Measurements

Our experiments are aimed at evaluating: (1) Robustness of Model Output, by recording the top-ranked output label for every combination of environment parameters and performing pairwise comparisons; and (2) Robustness of Model Execution Time, by measuring average inference times across executions in our dataset and comparing across different configurations. We perform pairwise comparison of configurations for each image in the dataset.

3.3 Execution Issues

All environment parameter combinations could not be executed with all models due to the following incompatibility issues. First, TF and TFLite versions of the DenseNet121 model resulted in incorrect output labels for most images. We believe the used model has been deprecated, as it no longer appears in the list of pre-trained models within the TensorFlow repository[2]. Second, for ResNet101 sourced from PyTorch, we selected the V1 version the model instead of V2 as the V2 version was not provided in the official PyTorch repository. The version difference may have a larger effect on model inference time when we compare across DL frameworks. Third, Regarding MobileNetV2, we experienced problems when executing it on the `Xavier` device, as we received a `CUDA_ERROR_INVALID_PTX` error. We do not consider this device configuration for MobileNetV2 in our experiments.

4 Results

4.1 Robustness of Output Label Prediction

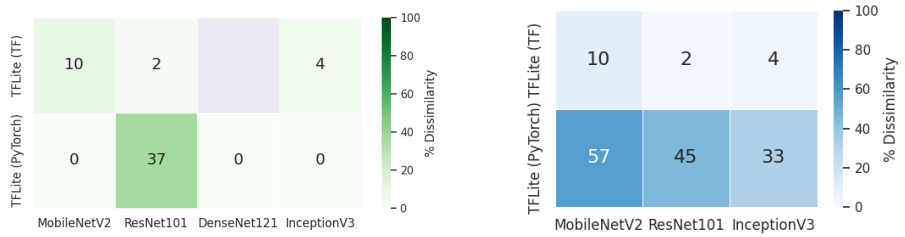
Impact of DL Framework Conversions We explore the impact of framework conversion on the model output by converting models from PyTorch and TF to TFLite, and evaluating the converted TFLite models in TVM. For TF-to-TFLite, we were able to convert the model directly. However, for PyTorch-to-TFLite, we had to first convert to ONNX, followed by a conversion to TensorFlow (TF) and finally to TFLite. We compare output the labels given by the native and converted models for each image to check if any errors were introduced by model conversion. We also compare labels of TF-to-TFLite and PyTorch-to-TFLite against the native TFLite model provided by the official TensorFlow repository.

The results are presented in Figure 2. We observe 37% discrepancy in the output labels when converting the ResNet101 model from PyTorch to TFLite (Figure 2a bottom row). In addition, we observe significant dissimilarities when comparing the converted models with the native TFLite model from TensorFlow repository, with PyTorch to TFLite version of MobileNetV2 exhibiting the largest discrepancy (57% as seen in Figure 2b). Each DL framework implements crucial operations and functionality (such as convolution and batch normalization) differently from the other frameworks and this can cause dissimilarities. In addition, intermediate formats used in conversions (such as ONNX) can give rise to small differences owing to a change in graph representation and supported operations in these formats. There are also considerable differences in the implementation of parallel operations due to the utilization of hardware acceleration libraries (such as CUDA and OpenCL).

Varying Compiler Optimizations We conducted experiments across all framework and device combinations described in section 3. On each experiment setup, we kept a framework and a device constant, while we varied the optimization level within the TVM framework between *Basic*, *Default*, and *Extended*.

We found that varying compiler optimization levels causes no discrepancies in output labels for all four models. The lack of discrepancies/sensitivity is notable, since the *Extended* (`-o4`) level enables

[2] `https://github.com/tensorflow/models/tree/master/research/slim#pre-trained-models`



(a) Comparison with base DL framework model. (b) Comparison with official TFLite model.

Figure 2: Pairwise comparison of output label dissimilarities

unsafe math optimizations that allow code violating IEEE `float` conventions to be generated. The conclusion is that these potential unsafe perturbations were small enough that all four models were resilient to them. It is however worth considering robustness checks with respect to optimization levels in safety-critical domains, in case unsafe optimizations result in undesirable model outputs.

4.2 Robustness of Model Inference Time

We conducted a pairwise comparison of model inference times for each image in the dataset across two configurations. We varied frameworks through conversion and compiler optimizations in the configurations.

Varying DL Framework Conversions Between TF and the converted TF-to-TFLite models, we observed up to 15% difference in inference times using the `InceptionV3` model, *Extended* optimization, *Hikey* device, with the TFLite model being faster. Similarly, we observed up to 10% difference between PyTorch and the PyTorch-to-TFLite converted model (`MobileNetV2` model, *Extended* optimization, *Hikey* device). We also observed a 96% discrepancy for ResNet101, however we use v1.5 instead of v2 used in TensorFlow or other frameworks utilized, as ResNet101V2 is not provided by the official PyTorch distribution.

Varying Compiler Optimizations We typically observed a speedup in inference time with increasing optimization levels, with a maximum speedup of 114%. However, there were instances where increased optimization led to a slowdown in inference time. For instance, on `MobileNetV2` with `Keras` DL framework, *Extended* optimization was 81.8% slower than *Basic*. Considering that `Extended` optimization applies a variety of strategies, such as constant and scale axis folding, canonicalization of operations, elimination of subexpressions and combination of parallel operations related to tensor manipulations, we hypothesize that one or more of these strategies fails to operate as intended, therefore hindering the performance. Figure 3 shows percentage difference in inference times for *Basic* versus *Extended* optimization on different devices and models with the `PyTorch` DL framework. For each device, we find times generally improved with increased optimization, in the range of $3.8 - 8.4\%$ for `Server`, and $17 - 54\%$ for `Local`. Increased optimizations on `Hikey`, however, had a 81% slowdown. The `Xavier` device also had a 36% slowdown when increasing optimization level from `Basic` to `Extended` on `InceptionV3` model.

For low to mid range devices, `Xavier` and `Hikey`, that experienced a slowdown with increased optimization, we believe the limited GPU memory poses a problem for the optimizations with parallel operations in the `Extended` optimization setting, leading to additional wait times, context switches and GPU data transfer time, that result in a slowdown. We will investigate each optimization pass, cache behavior, data transfer times between CPU–GPU and processor idle times in the future to confirm our hypothesis.

5 Conclusion

We explored the impact of DL framework conversions (e.g. PyTorch to TFLite and TF to TFLite) and varying compiler optimization levels on output label classification and inference times on four common image recognition models. We found that DL framework conversion can result in significant discrepancies in output labels (up to 37%) and also affects inference times (up to 15%). Changing compiler optimization levels does not affect classification labels but impacts inference times. Higher optimization levels typically imply faster inference times but there are outliers with some devices and models. Based on the significant discrepancies observed in our experiments, we believe it would be beneficial to benchmark the performance of DNN models for variations in these computational environment parameters that are currently not rigorously evaluated.

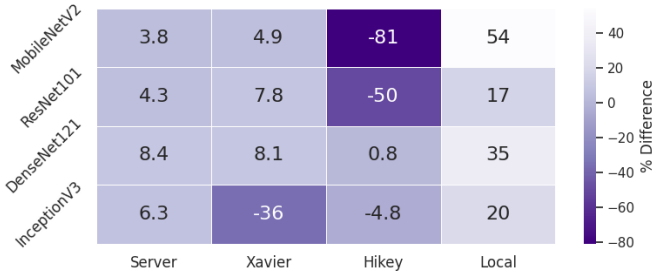


Figure 3: Execution time differences (%) for between Basic and Extended optimization across devices, utilizing PyTorch DL framework.

6 Appendix

6.1 Background

Figure 1 gives an overview of the typical layers in the deep learning systems stack [29]. Much of the existing work has focused on testing and robustness with respect to the top two layers, *Datasets* and *Models*, while *DL Frameworks* have also been explored to an extent. In this paper, we consider robustness with respect to the bottom three layers that make up the computational environment, required for executing a given DL model, that includes the DL framework (focusing on DL framework conversions), related systems software, and the underlying hardware.

6.2 Deep Learning Frameworks

Deep Learning Frameworks, shown as the third layer in Figure 1, provide utilities such as model declaration, training and inference to machine learning engineers. For our study, we use four DL frameworks that are widely used in the community: Keras, PyTorch, TensorFlow (TF), and TensorFlow Lite (also known as TFLite). We use these frameworks as sources for the image recognition models, as each has its own native definition for the models, and we consider a subset of them for our conversion experiments, as well as all of them for our optimization measurements. We briefly describe each of the four frameworks below.

Keras [9] is a high-level DL framework, providing APIs for effective deep learning usage. Keras acts as an interface for TensorFlow, and we aim to observe potential overheads and bug introductions from the extra layer of complexity.

PyTorch [19] is an open source machine learning framework based on the Torch library and developed by Meta AI team. It supports hardware acceleration for tensor computing operations.

TensorFlow (TF) [3] is an open-source DL framework, developed by Google, and widely used for training and inference of DNNs.

TensorFlow Lite (TFLite) [3] is a lightweight version of TensorFlow and part of the original TensorFlow library, providing framework focused only on the inference of neural networks on mobile and lightweight devices.

6.3 Systems Software: Apache TVM

Apache TVM [5] is an end-to-end machine learning compiler framework for CPUs, GPUs, and accelerators. It generates optimized code for specific DNN models and hardware backends, allows us to import DNN models from a range of DL frameworks, and provides profiling utilities such as per-layer inference times. A simplified representation of Apache TVM can be seen in Figure 4. TVM's support of several DL frameworks, optimization settings, and hardware accelerators made it a suitable choice explore varying different environment parameters in our experiments. TVM provides direct importers for models from most popular DL frameworks, which load said models as a TVM computation graph.

The first level of optimization available in TVM is graph-level optimizations, which is the focus of this study. These optimizations impact the full model and include operator fusion (e.g., batch normalization, activation functions), elimination of common subexpressions, and potentially unsafe optimizations such as fast math.

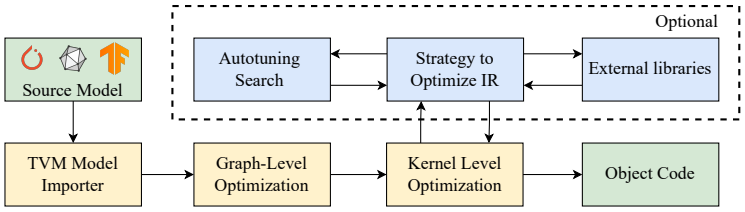


Figure 4: Overview of DNN compilation in Apache TVM.

DNN Model / Framework	PyTorch	Keras	TF	TFLite
ResNet101	81.9	76.4	77.0	77.0
InceptionV3	77.3	77.9	78.0	78.0
MobileNetV2	72.2	71.3	71.9	71.9
DenseNet121	74.4	75.0	N/A	N/A

Table 1: Accuracy of native models on the ImageNet dataset.

TVM also supports optimizations for a given operation type (e.g., convolutional layers, matrix-multiplications) such as loop tiling, loop re-ordering, unrolling, vectorization, auto-tuning [6], and auto-scheduling [36], among others.

TVM also supports third-party libraries such as cuDNN [7] and the Arm Compute Library [1].

6.4 The Perception AI Models

A common benchmark for Perception AI models is the ImageNet image object detection test dataset [22], performing classification for one of a possible 1000 class labels to RGB images of size 224×224 pixels. For solving Perception AI problems, such as classification and semantic segmentation, convolutional neural networks (CNNs) are commonly used, which are DNNs characterized by convolutional layers. Transformer-based architectures [30] have begun to provide competitive results in the past two years [10, 34], however are still maturing. Thus for our evaluation we explore four widely used CNN models: MobileNetV2 [23], ResNet101V2 [13], DenseNet121 [15], and InceptionV3 [27]. These models are widely known and extensively used for classification and semantic segmentation operations, as well as being the "backbone network" for other tasks such as object detection [8].

All four models have native definitions within the DL frameworks under study.

6.5 Model Selection

We selected the models aforementioned as they are widely used in the deep learning community [16]. The accuracy of the native version of each model is shown in Table 1. We verify that each model in correctly imported into TVM by comparing the output labels with the source framework.

7 Threats To Validity

There are five threats to validity in our experiments. First, we only evaluate robustness using four image recognition models that are widely used. Results are model dependent as seen in our experiments and will likely vary on other models. Second, we use the ImageNet [22] image object detection test dataset for our experiments, which believe adequately stresses configurations. Other datasets may yield different robustness results on the models considered. Third, model pre-processing is crucial for model performance [4]. We use the recommended pre-processing for each model and DL framework from the official repositories extracted. Fourth, we utilize TVM compiler framework and importing models into it, which can be an error-prone process. To ensure that errors are not introduced, for each model we validated the output labels for an indicative number of random image samples from their source framework against the output label given by the model after importing into TVM. The final threat is in inference time measurement. To ensure that time deviations are taken into account, we repeat inferences for each image 10 times and use the average inference time across 10 runs in a small-scale test dataset, verifying that no deviations happen on scaling.

References

[1] Compute Library. Arm Software, Aug. 2022.

[2] Open neural network exchange. https://onnx.ai/, 2022.

[3] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[4] J. Camacho-Collados and M. T. Pilehvar. On the role of text preprocessing in neural network architectures: An evaluation study on text categorization and sentiment analysis. *CoRR*, abs/1707.01780, 2017.

[5] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy. TVM: An automated end-to-end optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 578–594, Oct. 2018.

[6] T. Chen, L. Zheng, E. Yan, Z. Jiang, T. Moreau, L. Ceze, C. Guestrin, and A. Krishnamurthy. Learning to Optimize Tensor Programs. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 3393–3404. Curran Associates, Inc., 2018.

[7] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. Cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[8] Y.-C. Chiu, C.-Y. Tsai, M.-D. Ruan, G.-Y. Shen, and T.-T. Lee. Mobilenet-SSDv2: An Improved Object Detection Model for Embedded Systems. In *2020 International Conference on System Science and Engineering (ICSSE)*, pages 1–5, Aug. 2020.

[9] F. Chollet et al. Keras. `https://keras.io`, 2015.

[10] Z. Dai, H. Liu, Q. V. Le, and M. Tan. CoAtNet: Marrying Convolution and Attention for All Data Sizes. In *Advances in Neural Information Processing Systems*, volume 34, pages 3965–3977. Curran Associates, Inc., 2021.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[12] J. Guo, Y. Zhao, H. Song, and Y. Jiang. Coverage Guided Differential Adversarial Testing of Deep Learning Systems. *IEEE Transactions on Network Science and Engineering*, 8(2):933–942, Apr. 2021.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016.

[15] G. Huang, Z. Liu, and K. Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[16] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi. A survey of the recent architectures of deep convolutional neural networks. *CoRR*, abs/1901.06032, 2019.

[17] L. Liu, Y. Wu, W. Wei, W. Cao, S. Sahin, and Q. Zhang. Benchmarking deep learning frameworks: Design considerations, metrics and beyond. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1258–1269, 2018.

[18] N. Mahmoud, Y. Essam, R. Elshawi, and S. Sakr. Dlbench: An experimental evaluation of deep learning frameworks. In *2019 IEEE International Congress on Big Data (BigDataCongress)*, pages 149–156, 2019.

[19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019.

[20] K. Pei, Y. Cao, J. Yang, and S. Jana. Deepxplore: Automated whitebox testing of deep learning systems. *CoRR*, abs/1705.06640, 2017.

[21] H. V. Pham, T. Lutellier, W. Qi, and L. Tan. Cradle: Cross-backend validation to detect and localize bugs in deep learning libraries. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1027–1038, 2019.

[22] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[23] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.

[24] Q. Shen, H. Ma, J. Chen, Y. Tian, S.-C. Cheung, and X. Chen. A comprehensive study of deep learning compiler bugs. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2021, page 968–980, New York, NY, USA, 2021. Association for Computing Machinery.

[25] S. Shi, Q. Wang, P. Xu, and X. Chu. Benchmarking state-of-the-art deep learning software tools. *CoRR*, abs/1608.07249, 2016.

[26] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

[27] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[28] Y. Tian, K. Pei, S. Jana, and B. Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*, pages 303–314, 2018.

[29] J. Turner, J. Cano, V. Radu, E. J. Crowley, M. O'Boyle, and A. Storkey. Characterising Across-Stack Optimisations for Deep Convolutional Neural Networks. In *IISWC*, pages 101–110, 2018.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is All You Need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[31] Z. Wang, M. Yan, J. Chen, S. Liu, and D. Zhang. Deep learning library testing via effective model generation. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, page 788–799, New York, NY, USA, 2020. Association for Computing Machinery.

[32] Y. Wu, L. Liu, C. Pu, W. Cao, S. Sahin, W. Wei, and Q. Zhang. A comparative measurement study of deep learning as a service framework. *IEEE Transactions on Services Computing*, 15(1):551–566, 2022.

[33] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See. Deephunter: A coverage-guided fuzz testing framework for deep neural networks. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2019, page 146–157, New York, NY, USA, 2019. Association for Computing Machinery.

[34] X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer. Scaling Vision Transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12104–12113, 2022.

[35] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid. Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 132–142. IEEE, 2018.

[36] L. Zheng, C. Jia, M. Sun, Z. Wu, C. H. Yu, A. Haj-Ali, Y. Wang, J. Yang, D. Zhuo, K. Sen, J. E. Gonzalez, and I. Stoica. Ansor: Generating High-Performance Tensor Programs for Deep Learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 863–879, 2020.

[37] H. Zhou, W. Li, Z. Kong, J. Guo, Y. Zhang, B. Yu, L. Zhang, and C. Liu. Deepbillboard: Systematic physical-world testing of autonomous driving systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, page 347–358, New York, NY, USA, 2020. Association for Computing Machinery.