Reviewing Inference Performance of State-of-the-Art Deep Learning Frameworks

Berk Ulker[1], Sander Stuijk[1], Henk Corporaal[1], Rob Wijnhoven[2]
{b.ulker,s.stuijk,h.corporaal}@tue.nl,rob.wijnhoven@vinotion.nl
[1]Eindhoven University of Technology, [2]ViNotion
Eindhoven, The Netherlands

ABSTRACT

Deep learning models have replaced conventional methods for machine learning tasks. Efficient inference on edge devices with limited resources is key for broader deployment. In this work, we focus on the tool selection challenge for inference deployment. We present an extensive evaluation of the inference performance of deep learning software tools using state-of-the-art CNN architectures for multiple hardware platforms. We benchmark these hardware-software pairs for a broad range of network architectures, inference batch sizes, and floating-point precision, focusing on latency and throughput. Our results reveal interesting combinations for optimal tool selection, resulting in different optima when considering minimum latency and maximum throughput.

ACM Reference Format:
Berk Ulker[1], Sander Stuijk[1], Henk Corporaal[1], Rob Wijnhoven[2]. 2020. Reviewing Inference Performance of State-of-the-Art Deep Learning Frameworks. In *23rd International Workshop on Software and Compilers for Embedded Systems (SCOPES '20), May 25–26, 2020, Sankt Goar, Germany.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3378678.3391882

1 INTRODUCTION

Deep learning (DL) is applied successfully in several domains, such as computer vision [18], speech [15], text and natural language processing. While the capabilities and accuracy of these methods keep improving, the networks are becoming computationally complex with significant resource requirements. Due to resource limitations and stringent performance requirements on inference, deployment of Deep Neural Network (DNN)-based solutions on edge devices poses a challenging design problem. The selection of appropriate tools is becoming a more critical step in the design process, as the variety of such tools and target platforms continues to increase. In this work, we focus on the tool selection problem for the inference of DNNs. Deployment of DL-based solutions on edge devices requires specific hardware and software package combinations, in addition to algorithmic design. Based on different technical requirements and budget considerations, several different approaches are adopted. FPGA- and ASIC-based platforms have been used in industrial application deployments, which enable configurations of hardware and software specifically designed for an application. Although superior resource efficiency can be achieved through such solutions [28], custom implementation effort and hardware-specific knowledge [19] are often required. Resulting cost increase and limited reuse potential for implementation limit the feasibility of this approach. The most widely-adopted approach is the use of parallel processors, particularly GPUs [7], with software frameworks designed to provide an implementation of DNN functionality on these platforms. These tools provide implementations of various network architectures by their software packages combined with third-party primitive function libraries such as cuDNN [8], cuBLAS [4], MKL [3] and Eigen [12]. By providing abstraction and flexibility, software frameworks rapidly become popular as they help accelerating design, prototyping and testing tasks in industry and research domains.

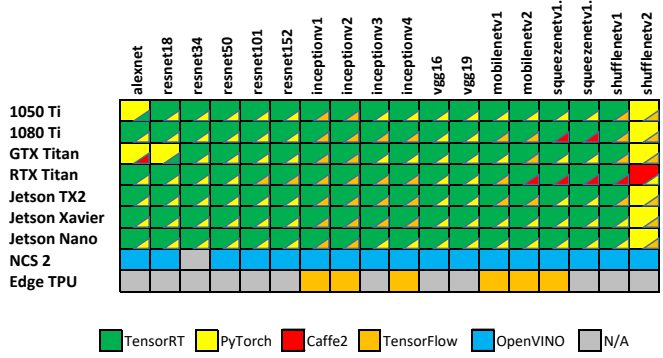


Figure 1: Software tools delivering minimum execution time for each platform-network configuration. Runner-up is shown on the lower right triangle in each cell, if available.

Available DL tools target various stages of deployment and have different sets of capabilities. Although a set of common libraries implementing primitives is shared, frameworks have diverged with differences in implementation, representation of network architectures and the level of hardware support. Combined with limitations of inference environments and application requirements, divergence in tools makes tool selection a key challenge. In our work, we aim to address this challenge by providing a comparative analysis of the inference performance of DL frameworks on different platforms with various network models, batch sizes and floating-point precision levels. We measure latency and throughput under variations over the benchmark dimensions to survey tool performance. Our discussions are extended through application scenarios with

This work is funded by the NWO Perspectief program ZERO

SCOPES '20, May 25–26, 2020, Sankt Goar, Germany

ACM ISBN 978-1-4503-7131-5/20/05…$15.00
https://doi.org/10.1145/3378678.3391882

	TensorFlow	PyTorch	Caffe2	TensorRT	OpenVINO
GTX 1050 Ti	✓	✓	✓	✓	
GTX 1080 Ti	✓	✓	✓	✓	
GTX Titan	✓	✓	✓	✓	
RTX Titan	✓	✓	✓	✓	
Jetson TX2	✓	✓		✓	
Jetson Nano	✓	✓		✓	
Jetson Xavier	✓	✓		✓	
Intel NCS2					✓
Edge TPU	✓				
Library Version	1.14.0	1.2.0	1.2.0	5.1.6.1	2.0.1
cuDNN	7.5.0.66	7.5.0.66	7.5.0.66	7.5.0.66	-
CUDA	10.0	10.0	10.0	10.0	-

Table 1: Hardware-software tool pairs used for evaluation.

Frameworks	TensorFlow [5], PyTorch [22], Caffe2, TensorRT, OpenVINO
Hardware	1050Ti, 1080Ti, GTX Titan, RTX Titan, Jetson TX2, Jetson Nano, Jetson Xavier, Neural Compute Stick 2, Edge TPU (dev. board)
Models	AlexNet, VGG-16/19, ResNet-18,34,50,101,152, Inception-v1,v2,v3,v4, MobileNet-v1,v2, ShuffleNet-v1,v2, Squeezenet-v1.0,v1.1
FP Precision	Single, Half
Batch Size	1-512

Table 2: Experimental grid.

minimum execution time and maximum throughput requirements, which reflect the most typical deployment requirements.

Our contribution in this paper is twofold. First, we provide a broad benchmark with most commonly used CNN architectures, popular and active DL software tools, hardware covering several edge devices and server type GPUs. Second, an in-depth analysis of benchmark results for the most popular software frameworks allows to quantify and compare relative inference performance. To the best of our knowledge, this work provides the most extensive and up to date comparison of the inference performance of DL software tools, running on a broad range of recent target platforms.

The rest of the paper is structured as follows. Section 2 provides an overview of existing works. In Section 3, our approach and motivation in the design of experiments are explained. Results of our benchmark, key cross-sections and discussions are provided in Sections 4 and 5. Finally, the paper is concluded in Section 6.

2 RELATED WORK

Increasing popularity of DL-based methods triggered development of various software tools to facilitate their implementation. Several works have tried to classify and evaluate the capabilities of DL software frameworks. However, comparing all such tools on common ground is challenging, as many of them are developed for specific and different use cases, target platforms and user bases.

Druzhkov [10], Erickson [11] and Pandey [21] present surveys of software frameworks by exploring software package properties and available DL functionality. These works provide insights on tool behavior and target platforms, but they lack performance evaluations based on any experiments or measurements. Hanhirova *et al.* [13] study latency and throughput characteristics for mobile devices. Although they study multiple inference run-time characteristics, their focus is on performance evaluation of different network architectures concentrated on TensorRT and TensorFlow [5]. With limited selection of software frameworks, this work does not address the selection problem completely.

Bahrampour *et al.* [6] provides a comparative study of DL software frameworks including community involvement and framework properties. Their work compares forward and backward runtimes of different networks on single- and multi-CPU and GPU platforms with different levels of primitive library support. Shi *et al.* [24] compare frameworks for training and inference performance on different hardware and analyse GPU utilization and relative performance against CPU execution. pCAMP by Zhang *et al.* [30] evaluates inference performance on edge devices. Power consumption, memory footprint and latency of several frameworks are studied with the Alexnet and Squeezenet models on different edge devices. The impact of initialization is also discussed. However, our analysis of inference includes experiments on additional hardware, uses different floating-point precision levels and batch sizes to analyze throughput latency characteristics.

In summary, existing works do not present a comprehensive and up to date analysis on inference performance of state-of-the-art DL software frameworks. We aim to resolve this gap, by comparing benchmark statistics of DL frameworks on an experimental grid constructed by different frameworks, platforms, network models, floating-point precision levels and batch sizes listed in Table 2.

3 BENCHMARK APPROACH

We designed our experiments to explore software tool behaviour in different dimensions. The most common approach in literature is to measure processing time of inference as a performance metric. Execution time alone does not reflect application requirements where processing is done on multiple objects or images in a single time frame. For such applications, throughput is key given that latency is within requirements. We therefore extended processing time measurements to larger batch sizes. Thus, we use forward processing latency and throughput as metrics to evaluate inference performance for each software tool.

Our benchmark includes a set of popular software frameworks, evaluated on hardware ranging from embedded accelerators, to high-end graphics cards (see Table 1). For each configuration, framework and support library versions are aligned to present a fair comparison. Framework and GPU accelerator library versions are shown in Table 1. CuDNN and CUDA library versions are aligned on the latest versions supported on the Jetson SOM platform.

Being the core of DL-based computer vision applications, CNN models are employed for many tasks, including classification. These models are also the backbone of more complex applications such as object detection, semantic segmentation, and flow estimation, which require additional layers implemented on top of the backbone network architecture. However, the computational cost of backbone networks is significantly higher compared to these additional layers. Thus, for our evaluation, we used backbone network architectures to represent computation loads.

The following widely utilized CNN network architectures are included in the benchmark: AlexNet [18], VGG [25], ResNet [14], Inception [26], MobileNet [16] [23], ShuffleNet [29] [20] and Squeezenet [17]. The models we use in our evaluation are model zoo implementations or third party implementations closest to the original work. We did not benchmark TensorRT with ShuffleNet-v2, PyTorch with Inception v1 and Inception v2, due to lack of these implementations.

The use of half-precision floating-point numbers is widely supported by software frameworks to reduce computational load. We performed our evaluation with both single and half-precision floating-point numbers. Note that the availability and performance of half-precision floating-point numbers are dependent on both the target platform and DL framework. Because we focus on exploring framework inference performance, we ignored algorithmic classification accuracy. This enabled us to use randomly initialized network parameters and avoid restrictions on our test space to models with pre-trained weights. Random images from the ImageNet dataset [9] were used as inputs for our experiments.

To explore the throughput-latency trade-off, we included different input batch sizes in our experiments. The limiting factor of the batch size is the available memory. Therefore, the maximum batch size can be different for each hardware/framework/floating-point-precision/network model combination. For our experiments, we measured inference latency for batch sizes ranging from 1 to 512. For TensorRT, the inference engine is generated independently for each batch size as different types of kernels may be used. For PyTorch, TensorFlow, and Caffe2, such decision is made at run-time, without hardware probing. For OpenVINO (Intel NCS2), the inference engine is also generated for each configuration, as the number of parallel workers may change based on the input batch size.

In our experiments, we measured the end-to-end execution time of the forward inference cycle in all combinations of our experimental grid. This timing does not include any data retrieval, initialization and pre-processing of input. Data transfers from host to accelerator are included in our measurements. In order to conduct an accurate benchmark, we used warm-up rounds and we ran our tests under the same power management modes. The complete experimental grid is shown in Table 2. It must be noted that we did not manage to evaluate certain configurations, which are discussed in the following sections with underlying reasons.

While measurement of forward execution is straightforward, getting layer-wise metrics of inference requires built-in and external profiler tools. Extent, output, and reliability of these tools show significant differences. Caffe2 profiler has support for per-layer measurements. TensorFlow's profiler Tfprof also has similar per layer profiling capabilities with a breakdown of layer execution times into CPU and accelerator. PyTorch provides similar timings, but results are not directly comparable. GPU operations in PyTorch are asynchronously executed: layer execution times result from both operator implementation efficiency and execution schedule. Kernel execution times are separately measured for GPU/CPU time for operations in each layer [2]. As a result of these differences, we do not present results based on the output of the profilers, such as per operation timings. We used the profiler tools only to identify and analyze run-time issues and measurements that stand out.

4 RESULTS

We present benchmark results in four subsections: inference execution times, inference throughput, throughput latency trade-off and half-precision floating-point performance. The total set of benchmark measurements and the source code for experiments are provided in our repository[1]. For additional cross-sections of our experiments, refer to Figures 8-17 in [27].

4.1 Inference execution times

Minimum forward execution time is often targeted for time-critical applications, where having minimal latency has more priority than having higher throughput. For such deployments, the batch size is often set to minimum, and lower floating-point precision is used. Figure 1 shows the software tools that achieve minimum execution time for each hardware and network architecture.

Our results in Figure 1 show that TensorRT delivers minimum average execution time for the network models that can be translated into TensorRT engines. This result is in line with our expectations, as TensorRT is a specialized proprietary inference tool for NVIDIA hardware with hardware probing and optimization capabilities beyond open-source DL tools. We failed to successfully translate or parse any official Shufflenet V2 model into a TensorRT engine.

When ignoring the closed-source TensorRT results, the clear runner-up is PyTorch for most networks and TensorFlow for the Inception architectures. Open source tools share more third party functionality and libraries, and the performance gap between them is smaller.

Figure 2 show inference execution times for each framework and hardware-software configuration, using unity batch size. We show results only for RTX Titan and Jetson Xavier as representations for server and edge hardware platforms, respectively. The performance gap between TensorRT and other tools are noticeable. PyTorch, TensorFlow and Caffe2 have comparable execution times, while on average PyTorch is the fastest. Caffe2 and PyTorch show similar scaling characteristics with network size, which can be explained by sharing codebase. TensorFlow has the highest overall execution time, but it shows a better scaling with increasing network size as is visible in the deeper ResNet and Inception variants.

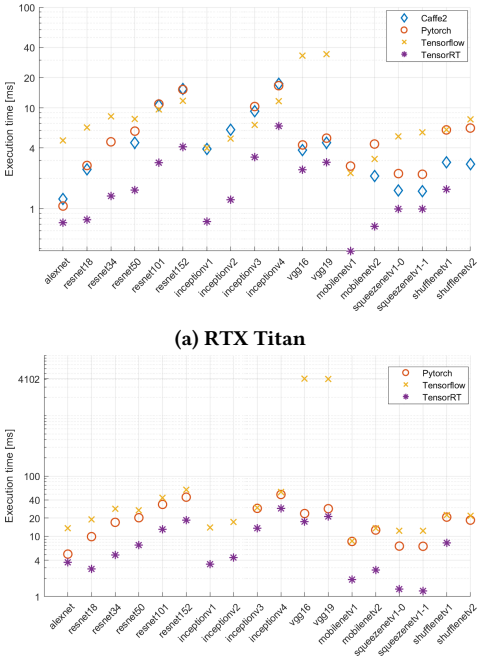
Caffe2 has performance issues in networks with group convolutions and depth-wise convolutions. Figure 2 shows that MobileNet-v2 and ShuffleNet variants have significantly higher execution times in Caffe2, with the only exception when being run on the RTX Titan.

TensorFlow execution times of Alexnet, VGG-16, and VGG-19 are significantly higher than other software frameworks, and also stand out when compared to other network timings. This can be explained by different implementations of fully connected layers of these network models. These models use the convolution operator, instead of using kernels designed for the fully connected operator. This affects the execution performance of Alexnet, VGG-16, and VGG-19 negatively for TensorFlow implementations.

4.2 Inference throughput

Inference throughput is key for applications which involve multiple inference operations in a single time frame. Applications without strict latency limitations can benefit from increased throughput, by increasing the inference batch size, at the cost of an increased latency. An overview of the best tools for each network and hardware combination is shown in Figure 3. Comparing these results to the latency evaluation in Figure 1, TensorRT again obtains overall best results. Runners up are PyTorch and TensorFlow, where TensorFlow now obtains second-best results on several ResNet variants.

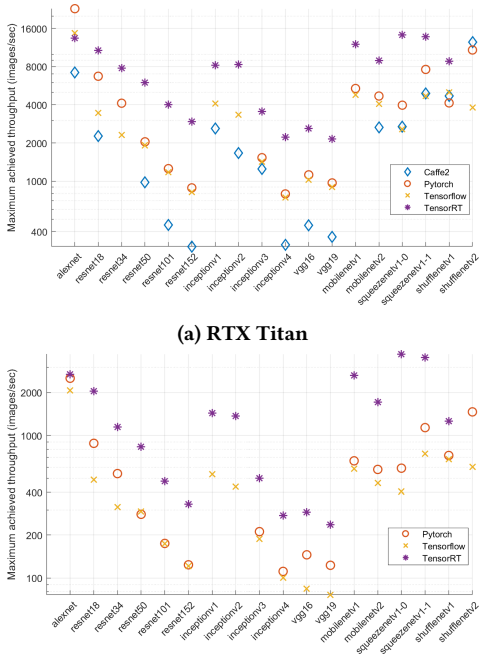
Figure 4 shows maximum achieved throughput on different hardware, by varying batch size and precision. In maximum throughput calculation, all available batch sizes and both half and single-precision floating-point levels are considered. This means that the



(a) RTX Titan

(b) Jetson Xavier

Figure 2: Inference execution times (batch size one, single-precision floating-point).



(a) RTX Titan

(b) Jetson Xavier

Figure 4: Maximum achieved throughput.

maximum throughput operating point of a framework-network-hardware combination can have a different batch size and floating-point precision, as compared to other combinations. This point is at a similar batch size for every network-framework pair with small variations only, and the point of optimum performance is often reached before all available memory is allocated. In general, TensorRT attains the highest throughput. As in the previous latency evaluation, the performance gap increases on hardware with more processing units.

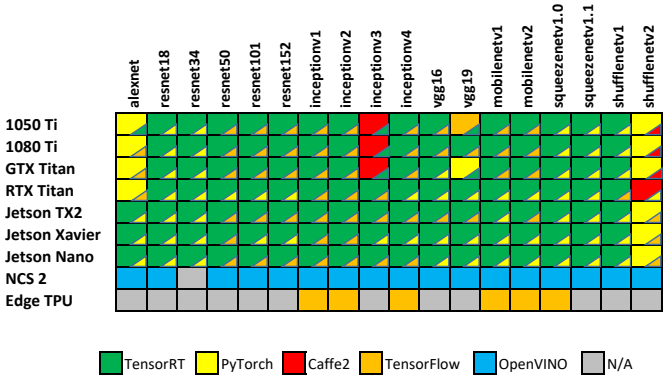


Figure 3: Software tools delivering maximum throughput for each platform-network configuration. Runner-up is shown on the lower right triangle in each cell, if available.

Caffe2 networks are executed only in single-precision floating-point. Figure 4 shows the effect of this limitation. Caffe2 has the lowest overall throughput, with an increasing gap for deeper networks. TensorFlow reduces the performance gap observed in the previous section, i.e. it benefits more from increased batch size as compared to the other frameworks.

4.3 Throughput-Latency Trade off

We now compare throughput to the corresponding latency. We used batch sizes of powers of two, ranging from 1 to 512. Figure 5 shows a subset of the resulting throughput/latency curves, measured on both server and embedded hardware. Increasing the batch size increases throughput significantly until performance converges at a certain batch size, typically between 16 and 32. A further increase provides a limited gain in throughput with increased latency. This occurs when further parallelization inside the GPU is not possible.

4.4 Half-precision floating-point performance

Half-precision floating-point computation is widely supported by DL software frameworks. If a limited reduction in accuracy can be accepted, then precision can often be brought to single-precision floating-point levels. Although all software frameworks included in this benchmark have half-precision floating-point support, we encountered several issues, especially with Caffe2. Because support for conversion of a model defined with single- to half-precision floating-point is not present in Caffe2, these measurements lack in our results.

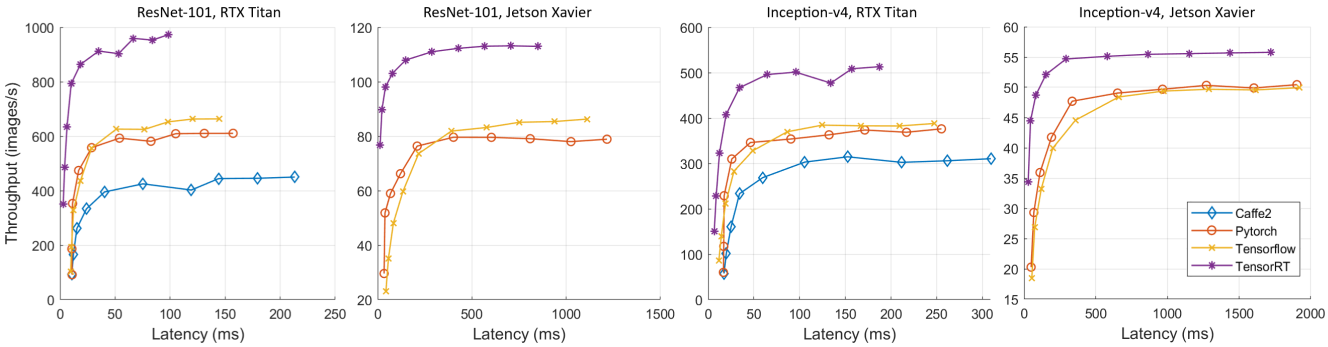
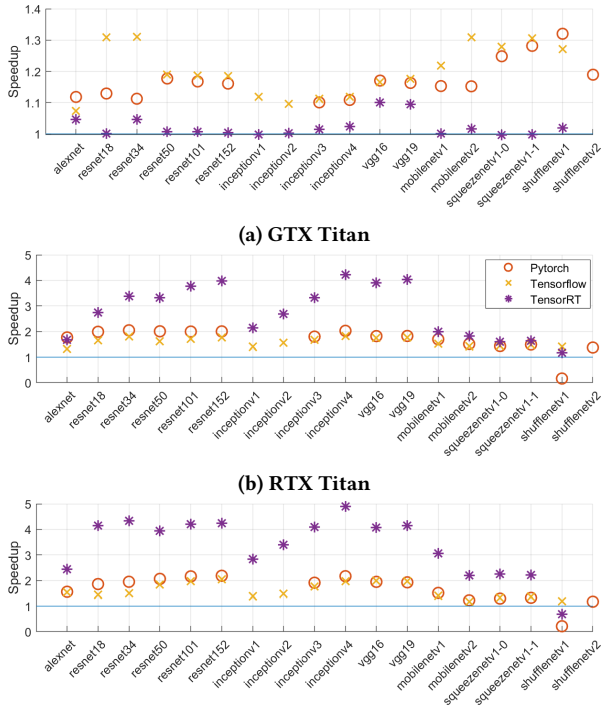


Figure 5: Throughput vs. latency curves for ResNet-101 and Inception-v4 networks, obtained by increasing batch size.



(a) GTX Titan

(b) RTX Titan

(c) Jetson Xavier

Figure 6: Speedup by switching to half-precision floating-point precision.

The performance gain from half-precision floating-point is dependent on both hardware and software tool support. Among the platforms used in the benchmark, hardware support for half-precision floating-point computation is available in RTX Titan, Jetson Nano, Jetson TX2, and Jetson Xavier. Figure 6 shows the speedup achieved by switching from single- to half-precision floating-point for each applicable software framework. Results show an overall increase in performance, with some notable exceptions. For GPUs without half-precision support, performance is almost similar in TensorRT on GPUs (a unity speedup). This is a result of not using type constraints, which allows TensorRT to select single-precision kernels. When strict type constraints are applied in TensorRT and only half-precision kernels are executed, negative performance influences can be observed for GTX Titan and 1080Ti. For PyTorch and TensorFlow, execution in half-precision floating-point is not enforced, yielding speedups slightly larger than unity in most cases. Layers with efficient half-precision implementation are executed in half-precision, all other combinations use single-precision implementations. Required data type conversion introduces overhead for conversion between single- and half-precision values. However, with hardware support for half-precision, much larger speedup factors are achieved by TensorRT. To investigate the unexpected performance decrease for MobileNet v1, MobileNet v2, and ShuffleNet V1, the internal profiler of TensorRT is used. We found that the reason are grouped convolutions which are used when the model is executed in half floating-point precision, which apparently cannot be mapped efficiently by TensorRT.

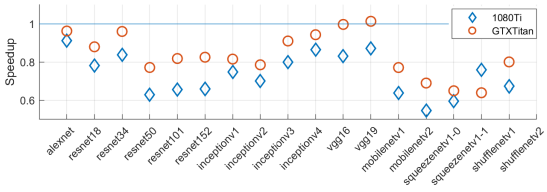


Figure 7: Speedup of TensorRT inference with strict type constraints set for half-precision, for hardware without half-precision computation capability.

5 DISCUSSION

Due to different requirements and priorities, the use of a single performance metric for all deployment scenarios is not feasible.

Overall, TensorRT delivers the highest throughput and smallest latency, although there are cases where it is either not applicable, or outperformed by others. Also, capabilities and flexibility of TensorRT are limited compared to others, as it can only perform inference on a limited set of platforms. Due to the lack of training functionality, deployment has a dependence on additional training and translation tools. We experienced issues in the translation of models and therefore used pre-trained networks in ONNX and Caffe format, which can be directly parsed into TensorRT.

Our results show that PyTorch can deliver less latency and more throughput on average, as compared to the Caffe2 and TensorFlow. While Caffe2 and PyTorch share functionality and part of their source code, overall Caffe2 performance is inferior compared to PyTorch. Among open-source frameworks, TensorFlow shows the best scaling with increased computational load. This effect is visible in Figure 2 and 4, where an increase in batch size and network complexity closes the gap with other frameworks.

NCS2 and Edge TPU can only be used with the respective frameworks and limited configurations, therefore providing an extensive comparison of frameworks on such hardware is not possible. However, we decided to include these results for completeness of absolute timing comparison of hardware-software pairs. Detailed results are presented in [27] and [1].

In light of our results and observations, it is not possible to conclude that one of the evaluated software frameworks is superior in all aspects. Latency and throughput characteristics provided in Figure 4 and 5 show which framework has better performance for certain operation points and network architectures. However, this comparative data does not only depend on tool selection. Application requirements may require specific functionality, which may be supported in various levels with different frameworks. Furthermore, dependency on additional tools and ease of use also need to be considered in the tool selection process. According to our experience, this decision requires a clearly defined scenario, where available hardware and resources, required capabilities, range of expected throughput and latency are included.

Throughput and execution time characteristics given in Figure 4 and 5 show that based on latency/throughput metrics, TensorRT delivers best performance. However, some use case-specific application requirements can only be provided with more complete, extendable and flexible tools. The open-source software frameworks PyTorch and TensorFlow can address those requirements better than TensorRT. Furthermore, when the latency limit is relaxed and larger batch sizes are allowed, the gap between framework performances decreases. For larger batch sizes, the throughput delivered by TensorFlow is higher than for PyTorch for many cases. Network composition can be a deciding factor since there are several edge cases where framework performance stands out negatively due to inefficient implementation of certain types of operators or network architectures. Depending on application specifications, PyTorch, TensorFlow or TensorRT can be the tool of choice. We do not recommend the use of Caffe2 due to inefficiencies in certain operator implementations, less flexible model structure, and lower level of active support while having similar performance as PyTorch.

6 CONCLUSIONS

This paper presents an extensive comparative inference performance evaluation of a set of deep learning software frameworks on a broad range of target hardware platforms. We focus on the local computation of convolutional neural network inference on edge devices. Based on our benchmark results, we discuss framework performance in terms of latency and throughput characteristics. We supplemented our interpretation with an investigation of shortcomings and standing-out behavior. Our discussions include tool selection for common scenarios in deep learning inference deployment for computer vision tasks. Results show that TensorRT has the best overall performance for inference on NVIDIA platforms. PyTorch and TensorFlow are runners-up, depending on the batch size, network depth and floating-point precision. For smaller batch sizes and more compact networks, PyTorch delivers higher throughput, while for most cases, TensorFlow delivers more throughput when batch sizes increase. We conclude that none of the software frameworks emerge as the best choice in all scenarios. However, an analysis of the requirements and limitations of the targeted deployment application can narrow the operating region, where our results enable a more precise tool selection.

REFERENCES

[1] [n.d.]. https://git.ics.ele.tue.nl/Public0/inference-benchmark
[2] [n.d.]. Pytorch autograd. https://pytorch.org/docs/stable/autograd.html. Accessed: 2019.
[3] 2009. *Intel Math Kernel Library. Reference Manual.* Intel Corporation, Santa Clara, USA. ISBN 630813-054US.
[4] 2018. cuBLAS. https://developer.nvidia.com/cublas
[5] Martín Abadi et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/
[6] Soheil Bahrampour et al. 2015. Comparative study of deep learning software frameworks. *arXiv:1511.06435* (2015).
[7] Tal Ben-Nun and Torsten Hoefler. 2018. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *arXiv:1802.09941* (2018).
[8] Chetlur et al. 2014. cudnn: Efficient primitives for deep learning. *arXiv:1410.0759* (2014).
[9] Deng et al. 2009. Imagenet: A large-scale hierarchical image database. In *CVPR*.
[10] PN Druzhkov and VD Kustikova. 2016. A survey of deep learning methods and software tools for image classification and object detection. *Pattern Recognition and Image Analysis* 26, 1 (2016), 9–15.
[11] Erickson et al. 2017. Toolkits and libraries for deep learning. *Journal of digital imaging* 30, 4 (2017), 400–405.
[12] Gaël Guennebaud et al. 2010. Eigen v3. http://eigen.tuxfamily.org.
[13] Hanhirova et al. 2018. Latency and throughput characterization of convolutional neural networks for mobile computer vision. In *MMSys*. ACM.
[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*.
[15] Hinton et al. 2012. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine* 29 (2012).
[16] Howard et al. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861* (2017).
[17] Iandola et al. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv:1602.07360* (2016).
[18] Alex Krizhevsky et al. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*.
[19] Griffin Lacey, Graham W Taylor, and Shawki Areibi. 2016. Deep learning on fpgas: Past, present, and future. *arXiv:1602.04283* (2016).
[20] Ma et al. 2018. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*.
[21] Hari Mohan Pandey and David Windridge. 2019. A comprehensive classification of deep learning libraries. In *ICICT*. Springer.
[22] Paszke et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035.
[23] Sandler et al. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*.
[24] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. 2016. Benchmarking state-of-the-art deep learning software tools. In *CCBD*. IEEE.
[25] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556* (2014).
[26] Szegedy et al. 2015. Going deeper with convolutions. In *CVPR*.
[27] B. Ulker, S. Stuijk, H. Corporaal, and R. Wijnhoven. 2020. *Reviewing Inference Performance of State-of-the-Art Deep Learning Frameworks.* Technical Report. TU Eindhoven. http://www.es.ele.tue.nl/esreports/esr-2020-02.pdf
[28] Wang et al. 2017. DLAU: A scalable deep learning accelerator unit on FPGA. *TCAD* 36, 3 (2017), 513–517.
[29] Zhang et al. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*.
[30] Xingzhou Zhang et al. 2018. pcamp: Performance comparison of machine learning packages on the edges. In {*USENIX*} *Workshop on HotEdge 18*.