When they go high we go low

Maksim Levental*
mlevental@uchicago.edu
University of Chicago

Elena Orlova
eorlova@uchicago.edu
University of Chicago

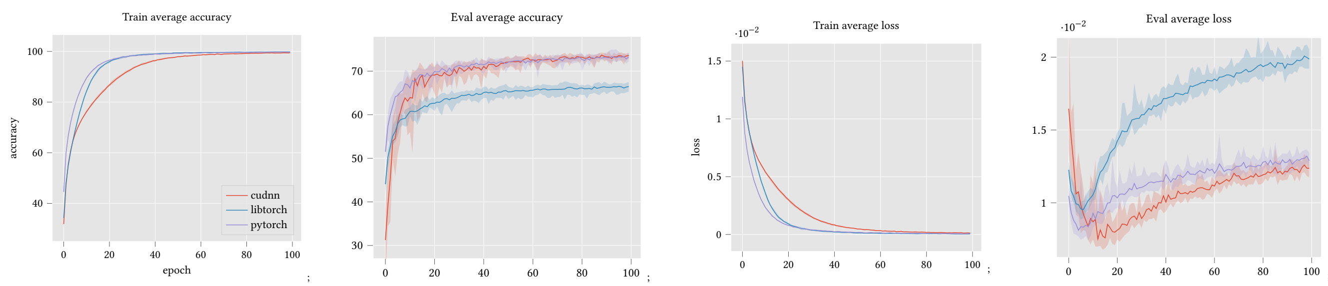


Figure 1: Comparison of PyTorch, LibTorch, and cuDNN implementations on CIFAR10.

ABSTRACT

High level abstractions for implementing, training, and testing Deep Learning (DL) models abound. Such frameworks function primarily by abstracting away the implementation details of arbitrary neural architectures, thereby enabling researchers and engineers to focus on design. In principle, such frameworks could be "zero-cost abstractions"; in practice, they incur enormous translation and indirection overheads. We study at which points exactly in the engineering life-cycle of a DL model are the highest costs paid and whether they can be mitigated. We train, test, and evaluate a representative DL model implemented using PyTorch, LibTorch, TorchScript, and cuDNN on representative datasets.

ACM Reference Format:
Maksim Levental and Elena Orlova. 2020. When they go high we go low. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In recent years, there has been a proliferation of Deep Learning (DL) frameworks[1, 3, 5, 7]. Many of these frameworks are implemented as domain-specific languages embedded in "high-level" languages[1] such as Python, Java, and C#. By virtue of this design decision, the frameworks implicitly trade-off ease-of-use for performance; quoting the architects of PyTorch:

> *To be useful, PyTorch needs to deliver compelling performance, although not at the expense of simplicity and ease of use. Trading 10% of speed for a significantly simpler to use model is acceptable; 100% is not.*

In general, this is a reasonable trade-off, especially during the early phases of the DL engineering/research process (i.e. during the hypothesis generation and experimentation phases). Invariable though, taking for granted a research direction bears fruit, one needs to put into production the DL model. It is at that phase of the process that every percentage point of performance becomes critical. Alternatively, there are many areas of DL research where the community strives to incrementally improve performance[2, 4, 6]. In such areas, where the ultimate goal is to build the most performant model, it's natural to wonder whether *any* trade-off that sacrifices performance is worthwhile.

2 BACKGROUND

2.1 GPUs

2.2 Graph compilers

2.2.1 Static.

2.2.2 Dynamic.

3 METHODOLOGY

We implement ResNet-50 at four levels of abstraction: PyTorch, TorchScript, LibTorch, and cuDNN. The reason for staying within the same ecosystem (PyTorch)is, in theory, we keep as many of the pieces of functionality orthogonal to our concerns as possible. We'll see that that reasoning doesn't quite bear out (see6).

*Both authors contributed equally to this research.
[1]For the purposes of this article, we take "high-level" to mean memory managed and agnostic with respect to hardware specifics.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

stage	output	ResNet-50
conv1	112×112	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$
conv4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$
conv5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	global average pool 1000-d fc, softmax
# params.		**25.5**$\times10^6$

3.1 Implementations

3.2 Profiling

4 RESULTS

4.1 Training and evaluation

4.2 Memory and utilization

5 DISCUSSION

6 FUTURE WORK

7 SPECULATION

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. arXiv:cs.DC/1603.04467

[2] Abdelrahman Abdelhamed, Mahmoud Afifi, Radu Timofte, Michael S. Brown, Yue Cao, Zhilu Zhang, Wangmeng Zuo, Xiaoling Zhang, Jiye Liu, Wendong Chen, Changyuan Wen, Meng Liu, Shuailin Lv, Yunchao Zhang, Zhihong Pan, Baopu Li, Teng Xi, Yanwen Fan, Xiyu Yu, Gang Zhang, Jingtuo Liu, Junyu Han, Errui Ding, Songhyun Yu, Bumjun Park, Jechang Jeong, Shuai Liu, Ziyao Zong, Nan Nan, Chenghua Li, Zengli Yang, Long Bao, Shuangquan Wang, Dongwoon Bai, Jungwon Lee, Youngjung Kim, Kyeongha Rho, Changyeop Shin, Sungho Kim, Pengliang Tang, Yiyun Zhao, Yuqian Zhou, Yuchen Fan, Thomas Huang, Zhihao Li, Nisarg A. Shah, Wei Liu, Qiong Yan, Yuzhi Zhao, Marcin Możejko, Tomasz Latkowski, Lukasz Treszczotko, Michał Szafraniuk, Krzysztof Trojanowski, Yanhong Wu, Pablo Navarrete Michelini, Fengshuo Hu, Yunhua Lu, Sujin Kim, Wonjin Kim, Jaayeon Lee, Jang-Hwan Choi, Magauiya Zhussip, Azamat Khassenov, Jong Hyun Kim, Hwechul Cho, Priya Kansal, Sabari Nathan, Zhangyu Ye, Xiwen Lu, Yaqi Wu, Jiangxin Yang, Yanlong Cao, Siliang Tang, Yanpeng Cao, Matteo Maggioni, Ioannis Marras, Thomas Tanay, Gregory Slabaugh, Youliang Yan, Myungjoo Kang, Han-Soo Choi, Kyungmin Song, Shusong Xu, Xiaomu Lu, Tingniao Wang, Chunxia Lei, Bin Liu, Rajat Gupta, and Vineet Kumar. 2020. NTIRE 2020 Challenge on Real Image Denoising: Dataset, Methods and Results. arXiv:cs.CV/2005.04117

[3] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. arXiv:cs.DC/1512.01274

[4] David Hall, Feras Dayoub, John Skinner, Haoyang Zhang, Dimity Miller, Peter Corke, Gustavo Carneiro, Anelia Angelova, and Niko Sünderhauf. 2020. Probabilistic Object Detection: Definition and Evaluation. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*.

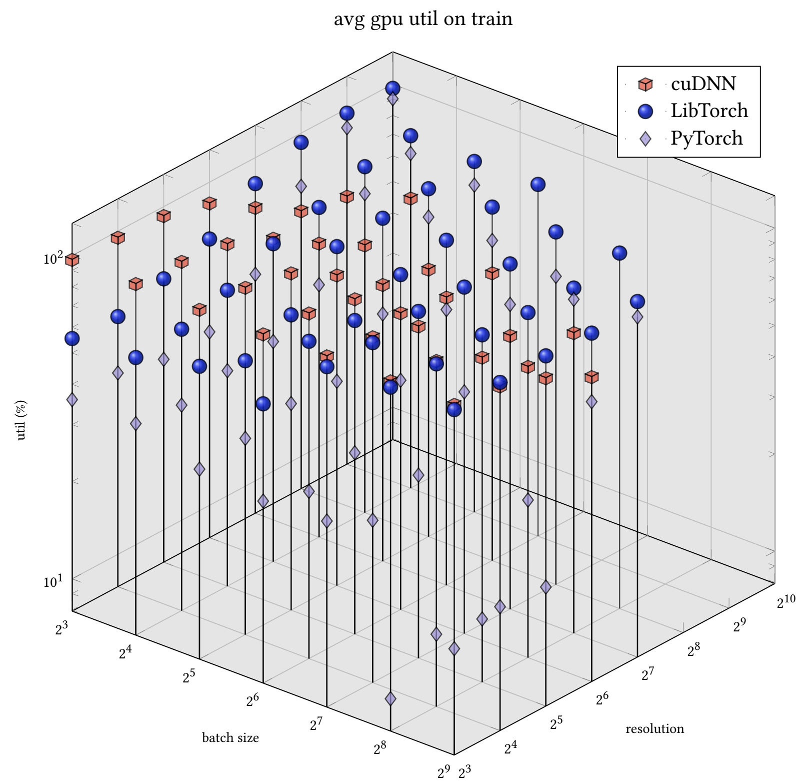
[5] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. arXiv:cs.LG/1912.01703

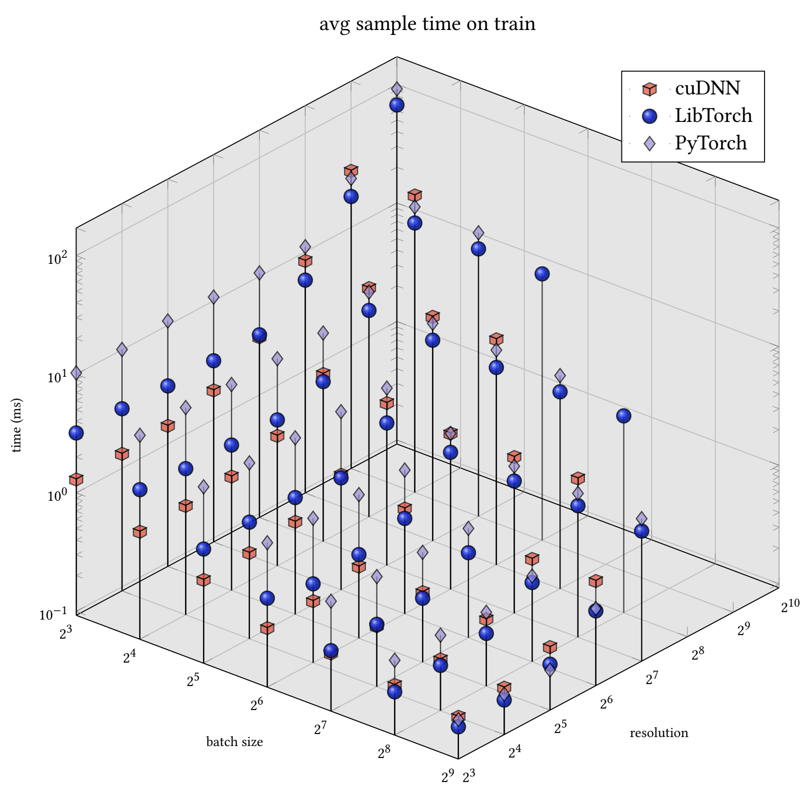
[6] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

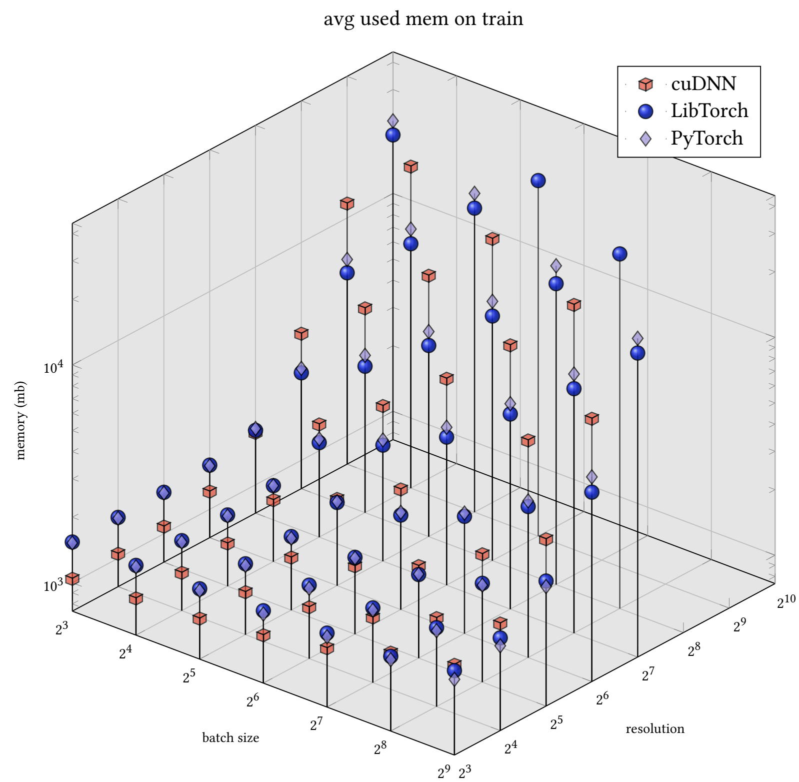
[7] Frank Seide and Amit Agarwal. 2016. CNTK: Microsoft's Open-Source Deep-Learning Toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. Association for Computing Machinery, New York, NY, USA, 2135. https://doi.org/10.1145/2939672.2945397

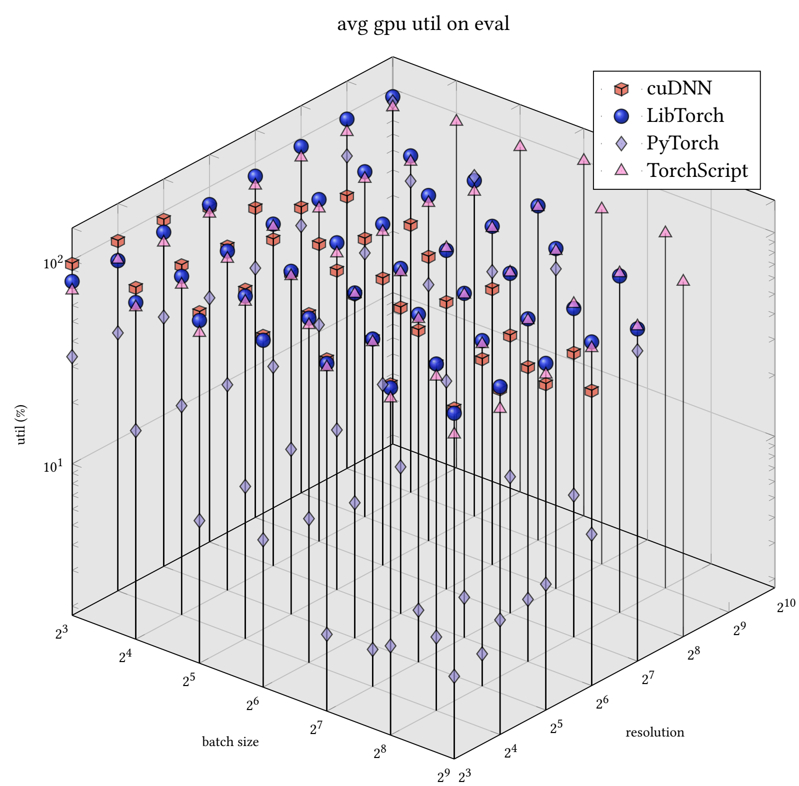
Appendices

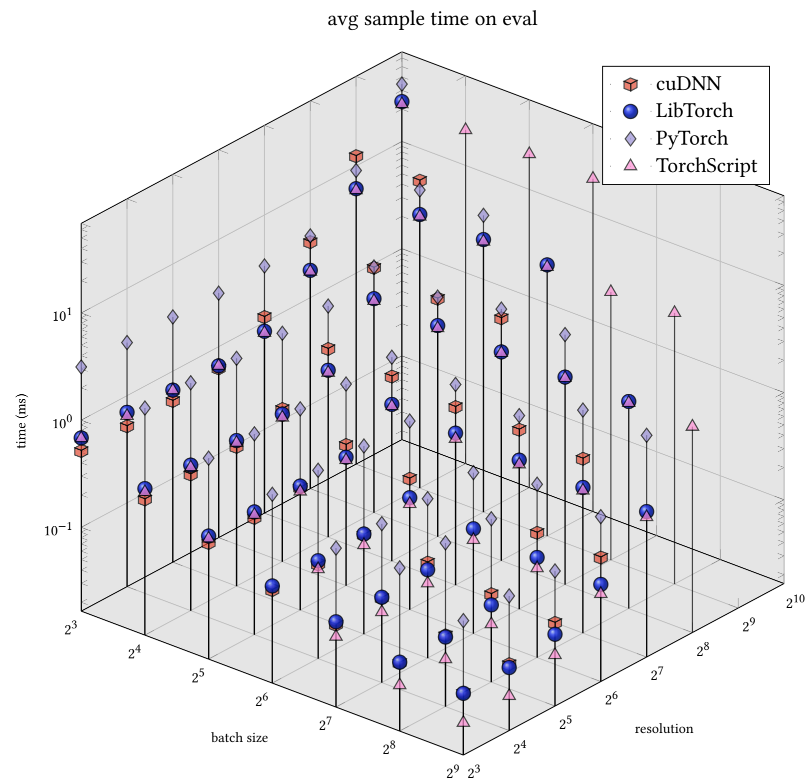
A APPENDIX

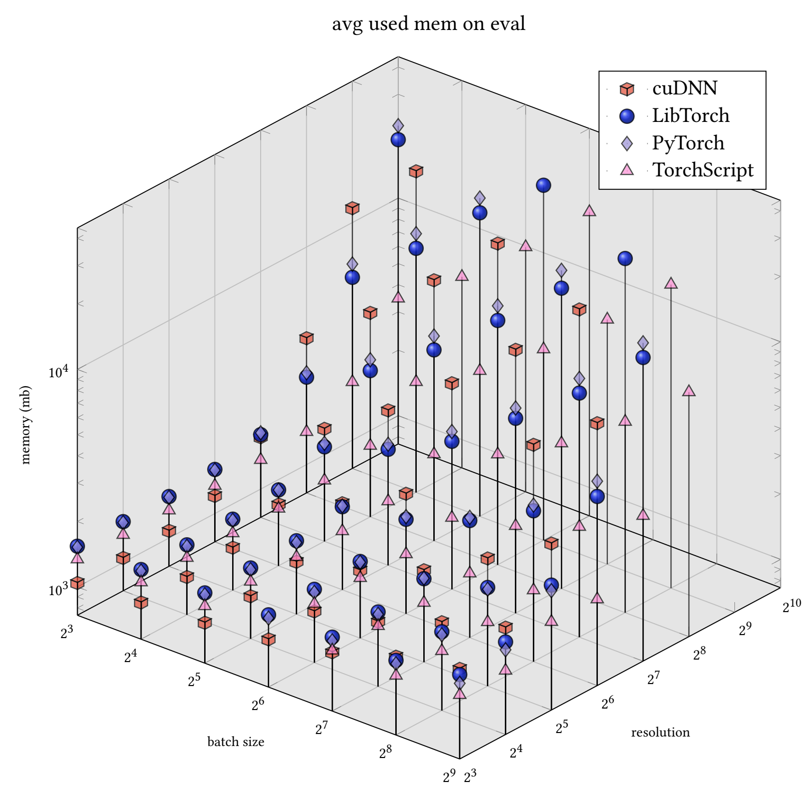

avg gpu util on train
cuDNN
LibTorch
PyTorch
util (%)
10^2
10^1
batch size
2^3
2^4
2^5
2^6
2^7
2^8
2^9
resolution
2^3
2^4
2^5
2^6
2^7
2^8
2^9
2^{10}


avg sample time on train
cuDNN
LibTorch
PyTorch
time (ms)
10^2
10^1
10^0
10^{-1}
batch size
2^3
2^4
2^5
2^6
2^7
2^8
2^9
resolution
2^3
2^4
2^5
2^6
2^7
2^8
2^9
2^{10}


avg used mem on train
cuDNN
LibTorch
PyTorch
memory (mb)
batch size
resolution


avg gpu util on eval
cuDNN
LibTorch
PyTorch
TorchScript
util (%)
batch size
resolution


avg sample time on eval
cuDNN
LibTorch
PyTorch
TorchScript
time (ms)
10^{1}
10^{0}
10^{-1}
batch size
2^3
2^4
2^5
2^6
2^7
2^8
2^9
resolution
2^3
2^4
2^5
2^6
2^7
2^8
2^9
2^{10}


avg used mem on eval
cuDNN
LibTorch
PyTorch
TorchScript
memory (mb)
10^4
10^3
batch size
2^3
2^4
2^5
2^6
2^7
2^8
2^9
resolution
2^3
2^4
2^5
2^6
2^7
2^8
2^9
2^{10}