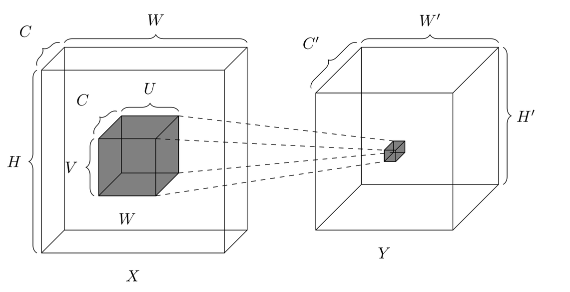
μ-cuDNN: Accelerating Deep Learning Frameworks with Micro-batches

Yosuke Oyama[1*], Tal Ben-Nun[2], Torsten Hoefler[2], Satoshi Matsuoka[3 1]
[1]Tokyo Institute of Technology [2]ETH Zurich [3]RIKEN Center for Computational Science *oyama.y.aa@m.titech.ac.jp

Background

Convolutional Neural Network (CNN)

- **Convolution** is one of the critical operations in Convolutional Neural Networks (CNNs)
  - Most of the computational time of CNN is spent on performing convolution
  - Most CNNs use two-dimensional convolution, which is composed of seven-nested loops



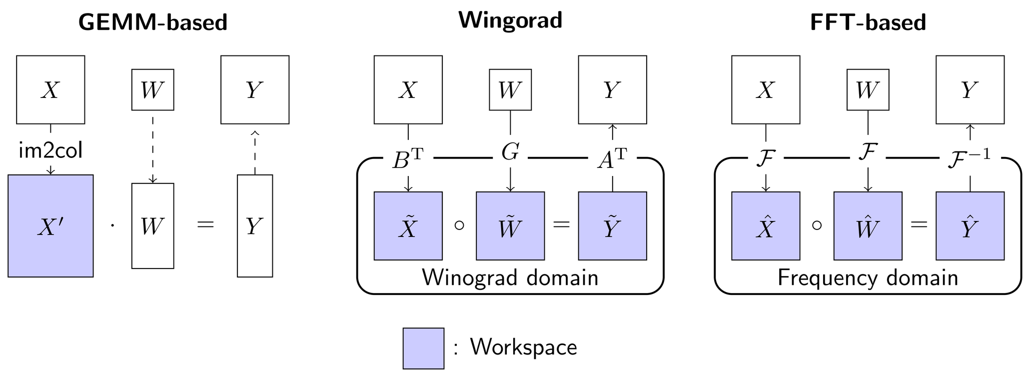
Two-dimensional convolution

Pseudo-code of two-dimensional convolution

for($n = 0$; $n < N$; n++) // Mini-batch loop
 for($k = 0$; $k < K$; k++) // Output channel loop
  for($h = 0$; $h < H$; h++) // Height loop
   for($w = 0$; $w < W$; w++) // Width loop
    for($c = 0$; $c < C$; c++) // Input channel loop
     for($v = 0$; $v < V$; v++) // Kernel width loop
      for($u = 0$; $u < U$; u++) // Kernel height loop
       $\mathbf{Y}[n,k,h,w] \mathrel{+}= \mathbf{W}[k,c,v,u] \times \mathbf{X}[n,c,h+v,w+u]$;

cuDNN

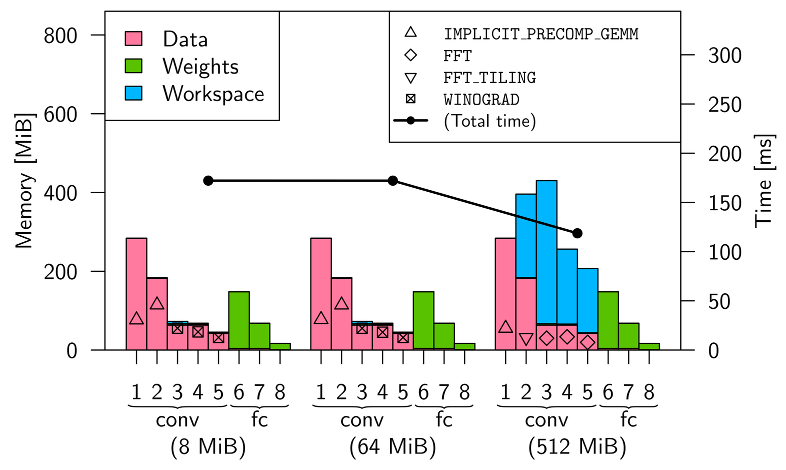
- **NVIDIA cuDNN** library [2] provides state-of-the-art deep learning primitives for GPUs
- cuDNN provides several equivalent convolution algorithms
  - GEMM-based convolution
  - FFT-based convolution
  - Winograd's algorithm



Convolution algorithms supported by cuDNN

cuDNN's Workspace Problem

- **Problem**: cuDNN may require **a workspace as large as the network itself (magnitude of GiB)** to use efficient convolution algorithms



Memory consumption (bars) and computation time (line/points) of AlexNet on P100-SXM2 with different workspace sizes

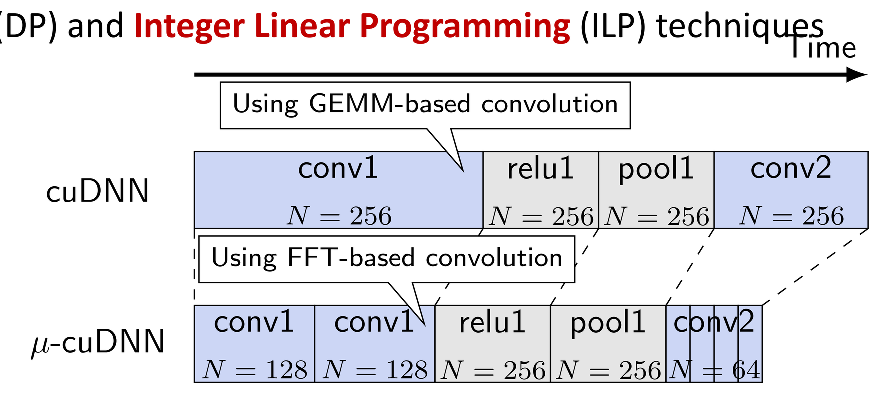
Problem Statement

- How can we use cuDNN's faster convolution algorithms with a small memory?
- How can we apply the approach to existing DL frameworks?

Proposal: μ-cuDNN

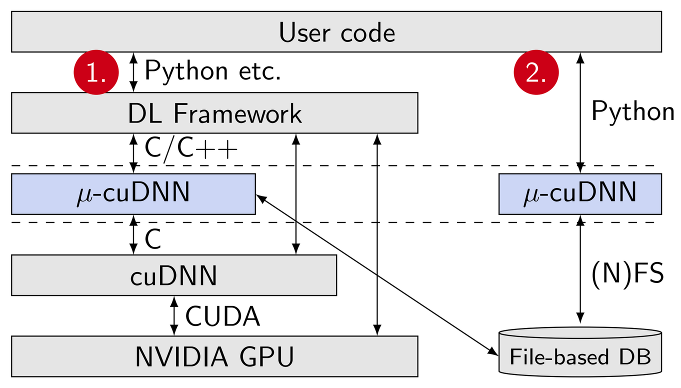
Overview

- **μ-cuDNN is a C++ thin wrapper library for cuDNN** [1]
  - It divides a mini-batch into "micro-batches" by applying loop splitting
  - The micro-batch division is optimized by using **Dynamic Programming** (DP) and **Integer Linear Programming** (ILP) techniques



The conceptual execution timeline of μ-cuDNN (*N* represents the mini-batch size)

- μ-cuDNN can be called by
  1. a DL framework as low-level performance tuning library
     - by overloading some of cuDNN's functions with a μ-cuDNN handle type
  2. its dedicated Python frontend for high-level performance analysis



μ-cuDNN software stack

μ-cuDNN is available online at https://github.com/spcl/ucudnn

Workspace Policies: WR and WD

- μ-cuDNN employs one of two workspace utilization policies:
  - **Workspace Reuse (WR)**: Each layer reuses a private workspace
  - **Workspace Division (WD)**: Each layer uses a part of a unified workspace

μ-cuDNN's workspace utilization policies

	WR	WD
Maximum total WS size	$\mathcal{O}$(# of layer) ☹	constant ☺
Optimizer	DP	DP+ILP
WS owner	DL framework	μ-cuDNN

Workspace Reuse (WR)

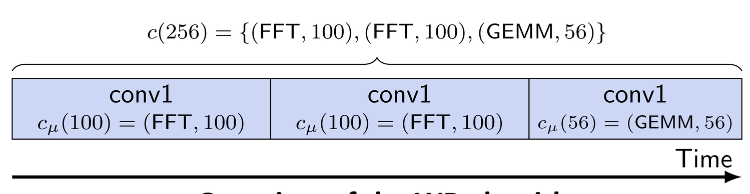
- Given a mini-batch size B and the fastest execution time $T_\mu(b)$ ($b = 1,2,\ldots,B$), compute $T(B)$ where

$$T(b) = \min \left\{ \begin{array}{l} T_\mu(b), \\ \min_{b'=1,2,\ldots,b-1} T(b') + T(b-b') \end{array} \right\}$$

- **Solution**: Solve a Dynamic Programming problem:

DP-based solution of WR policy

for $b = 1$ to B **do**
 $\hat{b_\mu} \leftarrow \operatorname*{argmin}_{b_\mu = 1,2,\ldots,b} \{T_\mu(b_\mu) + T(b - b_\mu)\}$
 $T(b) \leftarrow T_\mu(\hat{b_\mu}) + T(b - \hat{b_\mu})$
 $c(b) \leftarrow \{c_\mu(\hat{b_\mu})\} + c(b - \hat{b_\mu})$
end for
return $c(B)$ // **Configuration**; a list of (algorithm ID, batch size)



Overview of the WR algorithm

Workspace Division (WD)

- Given
  - M: A total workspace limit
  - $\mathcal{K}$: A set of convolution kernels
  - C_k: A set of configurations of a kernel k
  - $T_k(c)$: The fastest execution time of a kernel k and using a configuration c,
- Compute

$$\operatorname*{argmin}_{f:\mathcal{K}\to C_k} \sum_{k\in\mathcal{K}} f(k) T_k(c)$$

$$\text{s.t.} \quad \sum_{k\in\mathcal{K}} f(k) M_k(c) \le M$$

- **Solution**: Solve an Integer Linear Programming problem:
  - We remove all of the Pareto-suboptimal configurations in advance

$$\min \quad T = \sum_{k\in\mathcal{K}} \sum_{c\in C_k} T_k(c) x_{k,c}$$

$$\text{s.t.} \quad \sum_{k\in\mathcal{K}} \sum_{c\in C_k} M_k(c) x_{k,c} \le M$$

$$\sum_{c\in C_k} x_{k,c} = 1 \ (\forall k \in \mathcal{K})$$

$$x_{k,c} \in \{0,1\} \ (\forall k \in \mathcal{K}, \forall c \in C_k)$$

Workspace granularities and datatypes

- μ-cuDNN users one of three micro-batch size granularities
  - `undivided` is equivalent to cuDNN
- μ-cuDNN increases the number of algorithms by exploiting higher computation precision (`PSEUDO_HALF`) than specified (`TRUE_HALF`) without decreasing the accuracy

Micro-batch size granularities

	Micro-batch size set
`all`	$\{1,2,3,\cdots,B\}$
`powerOfTwo`	$\{2^0,2^1,2^2,\cdots,B\}$
`undivided`	$\{B\}$

cuDNN's convolution datatypes

Configuration	Data Type	Compute Type	FFT
`TRUE_HALF`	`half`	`half`	
`PSEUDO_HALF`	`half`	`float`	✓
`FLOAT`	`float`	`float`	✓

Evaluation

Evaluation Environment

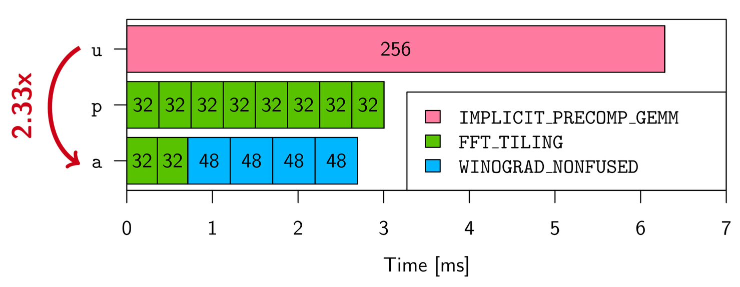
- **GPUs**: NVIDIA Tesla K80, P100-SXM2, V100-SXM2, K20Xm, and 750Ti
- **cuDNN**: 7.1 (or 6.0 for Caffe and TensorFlow)
- **Frameworks**: Caffe 1.0, TensorFlow 1.4.1
- **LP solver**: GNU Linear Programming Kit (GLPK) 4.63

GPU specification

	Generation	TFlop/s FP32	TFlop/s FP16	Memory [GiB]	Tensor cores	Host
K80	Kepler	8.73	-	24	-	TSUBAME-KFC/DL
P100-SXM2	Pascal	10.6	21.2	16	-	TSUBAME 3.0
V100-SXM2	Volta	15.7	125	16	✓	NVIDIA DGX-1
GTX 750Ti	Maxwell	1.31	-	2	-	TSUBAME-KFC/DL
K20Xm	Kepler	3.95	-	6	-	TSUBAME-KFC/DL

Single Convolutional Layer

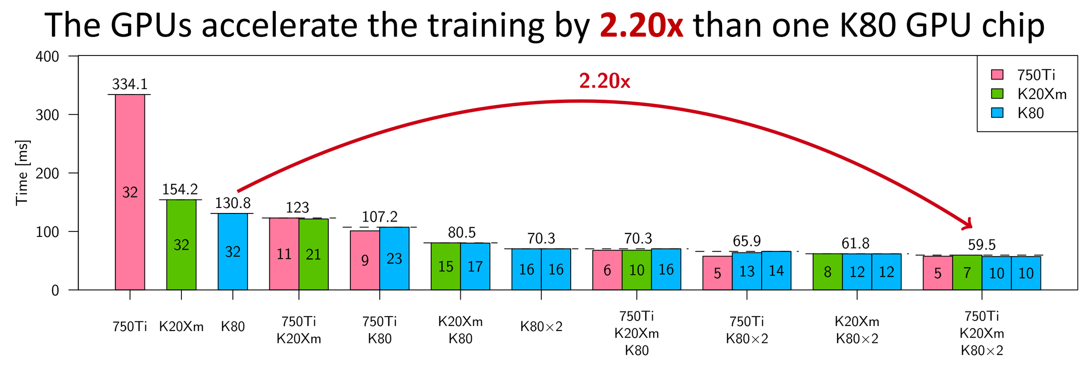
- μ-cuDNN achieves **2.33x** speedup on AlexNet's "conv2" layer by utilizing both FFT-based convolution and Winograd's algorithm
  - GEMM-based convolution requires only a workspace of 4.3 KiB but relatively slow
  - FFT-based convolution is faster than GEMM, but it requires a workspace of 213 MiB with a mini-batch size of 256



Time (bars) and micro-batch sizes (labels in bars) of forward convolution of AlexNet's "conv2" layer on P100-SXM2. We use a workspace size of 64 MiB

Case Study: Heterogeneous cluster optimization

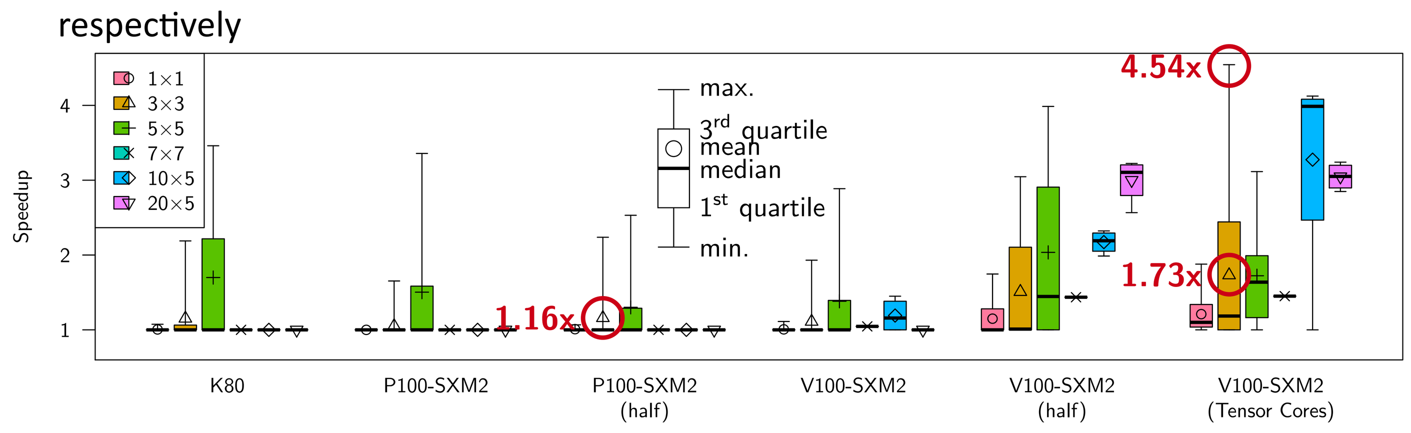
- We estimate time to perform a data-parallel training pass on a heterogeneous GPU cluster (K80 + 750Ti + K20Xm) using μ-cuDNN's Python frontend
  - The GPUs accelerate the training by **2.20x** than one K80 GPU chip



Predicted time to perform a data-parallel training pass of ResNet-18 on a heterogeneous cluster

DeepBench

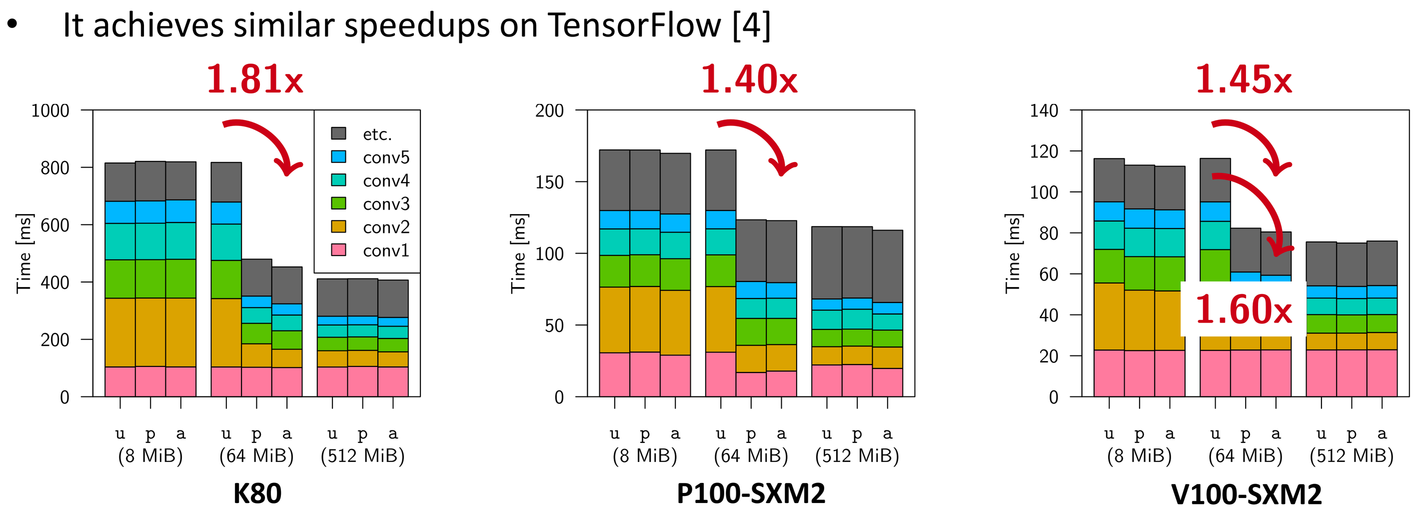
- DeepBench is a set of frequently used layer configurations in DL
  - which contains 94 convolutional layers
- μ-cuDNN achieves up to **4.54x** speedup (1.60x on average) on V100-SXM2 using Tensor Cores
  - μ-cuDNN exploits PSEUDO_HALF in 69% of the layers
- μ-cuDNN achieves **1.16x**, **1.73x** average speedups for 3 × 3 kernels on P100 and V100 respectively



Relative speedups of DeepBench's forward convolution layers against cuDNN

Workspace Reuse (WR)

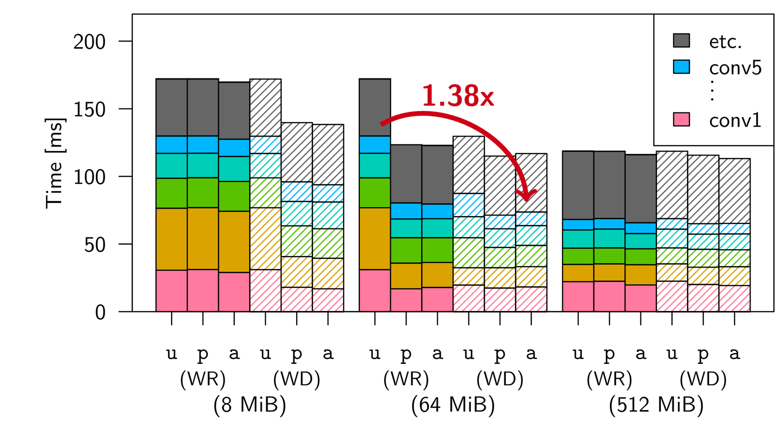
- μ-cuDNN on the Caffe framework [3] achieves **1.45x** speedup (1.60x w.r.t. convolutions alone) on V100-SXM2
  - It achieves fewer speedups with a small workspace (8 MiB) or a considerable workspace (512 MiB), due to lack of effectiveness of micro-batching
- It achieves similar speedups on TensorFlow [4]



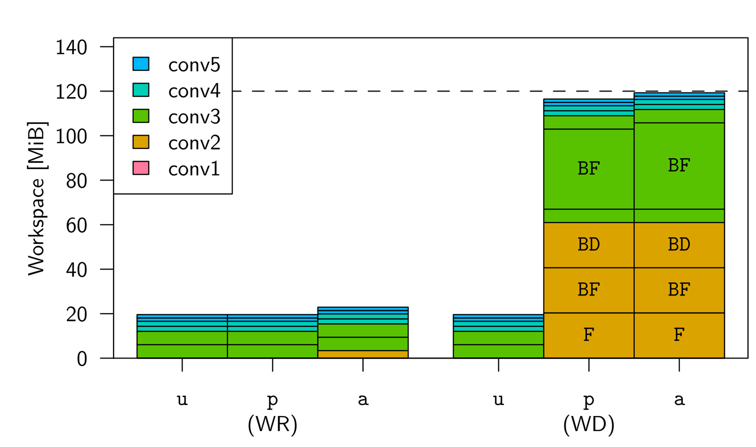
Time to perform forward- and backward-passes of AlexNet on three different GPUs using different workspace sizes (8, 64, 512 MiB)

Workspace Division (WD)

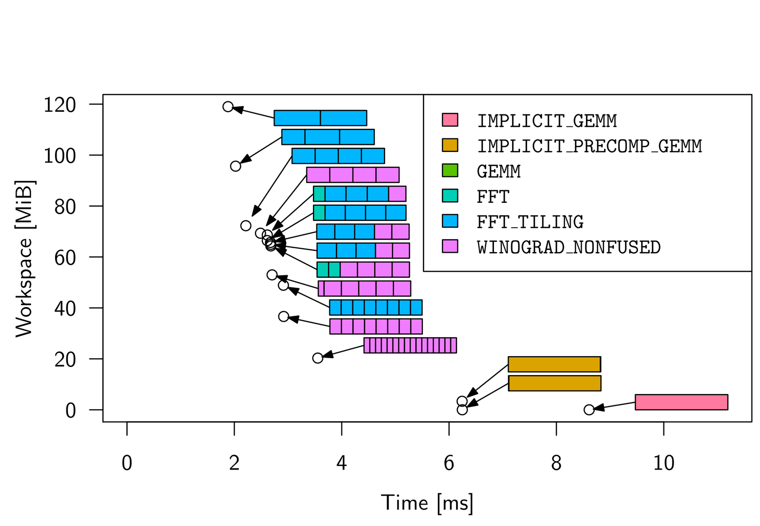
- μ-cuDNN on Caffe achieves **1.38x** and **1.14x** speedups for convolutional layers of AlexNet and ResNet-50 on P100-SXM2
- Time to solve the ILP problem was negligible (5.46 ms for ResNet-50)



Time to perform forward- and backward-passes of AlexNet on three different GPUs using different workspace sizes (8, 64, 512 MiB)



Assigned workspace sizes for AlexNet on P100-SXM2



Assigned workspace sizes for AlexNet on P100-SXM2

References

[1] Y. Oyama, T. Ben-Nun, T. Hoefler, S. Matsuoka, "Accelerating Deep Learning Frameworks with Micro-batches," in proceedings of IEEE Cluster 2018, Sep 2018.
[2] NVIDIA, NVIDIA "cuDNN," https://developer.nvidia.com/cudnn.
[3] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," arXiv preprint arXiv:1408.5093, 2014.
[4] 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," https://www.tensorflow.org/, Nov 2015.