Performance Analysis of CNN Frameworks

: Performance Analysis of CNN Frameworks for GPUs
Martin Hwang

Microsoft
CNTK

Performance Analysis of CNN Frameworks

핵심 개념: Concept

동기: Motivation

방법론: Detail

실험결과: Experiment

핵심 개념: Concept

각 딥러닝 프레임워크의

GPU에서의 Convolution Layer 연산과

Data Parallelism을 비교 분석하자

theano

동기: Motivation

다양한 딥러닝 프레임워크

Caffe, CNTK, Torch, Tensorflow 등

여러가지 옵션 존재

여러 연산에 대한 cuDNN 옵션이 존재

각 프레임워크에 대한 분석

여러 옵션과 병렬화에 대해
프레임워크별 분석 필요

이전 분석 연구들보다 더 자세히 프레임워크별 성능 분석을 해보자

Detail

Convolution Operation

Parallelism

Frameworks

theano

TensorFlow

Microsoft CNTK

torch

Convolution Operation

- Definition of Convolution Operation

  - 인공 신경망에서 영상처리를 위해 사용되는 Convolution Operation의 정의

  - Convolution Operation을 수행하기 위한 방법론은 나이브한 방법부터 다양한 방법이 존재

$$(D * F)[h][w] = \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} D[h+r][w+s]F[r][s]$$

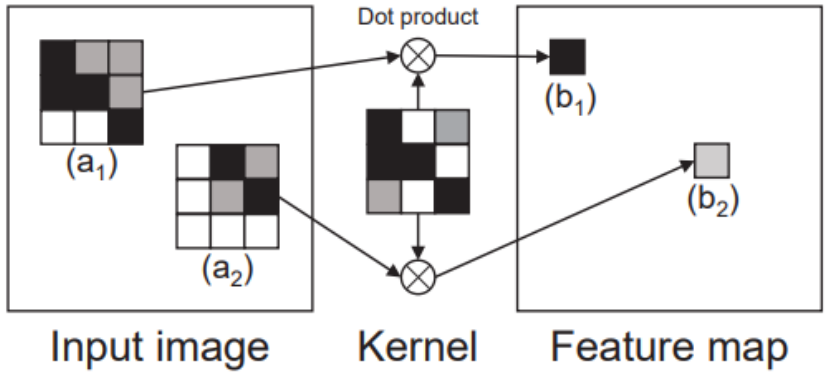


Fig. 1: 2D convolution.

Convolution Operation

- Direct Convolution

  - Convolution Operation을 수행할 때, 기본적으로 사용하는 연산 방법

  - 연산 복잡도는 $O(K \times CRS \times NWH)$
  K : 커널 수
  $R \times S$: 커널 크기
  C : 입력 채널 수
  $H \times W$: 입력 이미지 크기
  N : 이미지 수

  - 연산 복잡도는 Fully Connected Layer와 같음

  - BLAS를 활용하여 쉽게 병렬화 할 수 있음

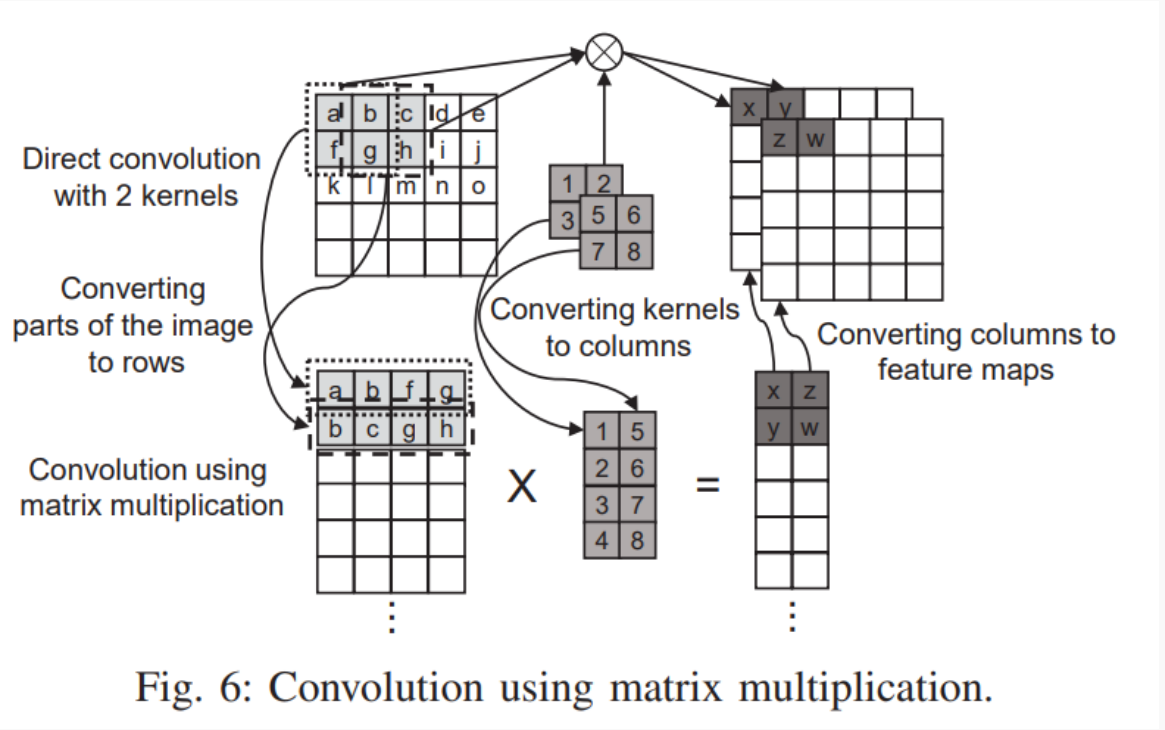


Fig. 6: Convolution using matrix multiplication.

Convolution Operation

- FFT Convolution

  - Convolution Operation은 FFT(Fast Fourier Transform) 을 활용하여 구현 가능

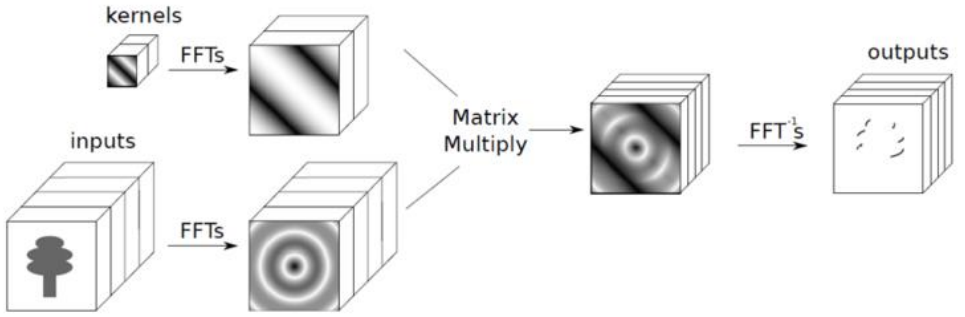
  - 연산 복잡도는 $O(K \times CWW \times logW)$

  - 커널 크기와는 의존성이 없는 것이 특징

  - Stride가 1인 경우만 적용이 가능

  - 필터의 차원을 입력 데이터의 차원에 맞춰야하므로 메모리를 더 사용하게 됨

$$f * g = \mathcal{F}^{-1}(\mathcal{F}(f) \cdot \mathcal{F}(g))|$$



- **Computation**
  - $n \times n$ *image*, $k \times k$ *kernel*
  - *conventional convolution*: $(n-k+1)^2k^2$ *operations*
  - $FFT-based\ method: 6n^2 + 6Cn^2 logn$ (C: *constant*) *operations*
    - 2 * FFT (Image, filter) + 1 IFFT (frequency -> spatial domain)

	Direct Convolution	Our Method
$\sum_f x_f * w_{f'f}$	$S \cdot f' \cdot f \cdot n'^2 \cdot k^2$	$2Cn^2 \log n[f' \cdot S + f \cdot S + f' \cdot f] + 4S \cdot f' \cdot f \cdot n^2$
$\frac{\partial L}{\partial y_{f'}} * w^T_{f'f}$	$S \cdot f' \cdot f \cdot n^2 \cdot k^2$	$2Cn'^2 \log n'[f' \cdot S + f \cdot S + f' \cdot f] + 4S \cdot f' \cdot f \cdot n'^2$
$\frac{\partial L}{\partial y_{f'}} * x_f$	$S \cdot f' \cdot f \cdot k^2 \cdot n'^2$	$2Cn \log n^2[f' \cdot S + f \cdot S + f' \cdot f] + 4S \cdot f' \cdot f \cdot n^2$

- **4배 이상의 메모리 사용** $4n(n+1)(S \cdot f + S \cdot f' + f \cdot f')$

https://arxiv.org/pdf/1312.5851.pdf

Convolution Operation

- Winograd Convolution

  - Strassen 알고리즘과 유사

  행렬 곱셈 알고리즘 복잡도 : $O(n^3)$
  Strassen 알고리즘 복잡도 : $O(n^{2.807})$
  Winograd 알고리즘 복잡도 : $O(n^{2.376})$

  - 커널 크기가 고정되면 곱셈 연산이 크게 줄어듬

  - 다른 커널 크기에는 다른 필터링 알고리즘 필요

  - cuDNN 5.1은 3x3, 5x5 필터 크기에 대해서만 Winograd Convolution 지원

$$F(2,3) = \begin{bmatrix} d_0 & d_1 & d_2 \\ d_1 & d_2 & d_3 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 - m_4 \end{bmatrix} \tag{5}$$

$$m_1 = (d_0 - d_2)g_0 \qquad m_2 = (d_1 + d_2)\frac{g_0 + g_1 + g_2}{2}$$
$$m_4 = (d_1 - d_3)g_2 \qquad m_3 = (d_2 - d_1)\frac{g_0 - g_1 + g_2}{2}$$

https://arxiv.org/pdf/1312.5851.pdf

Parallelism

- **Data Parallelism**

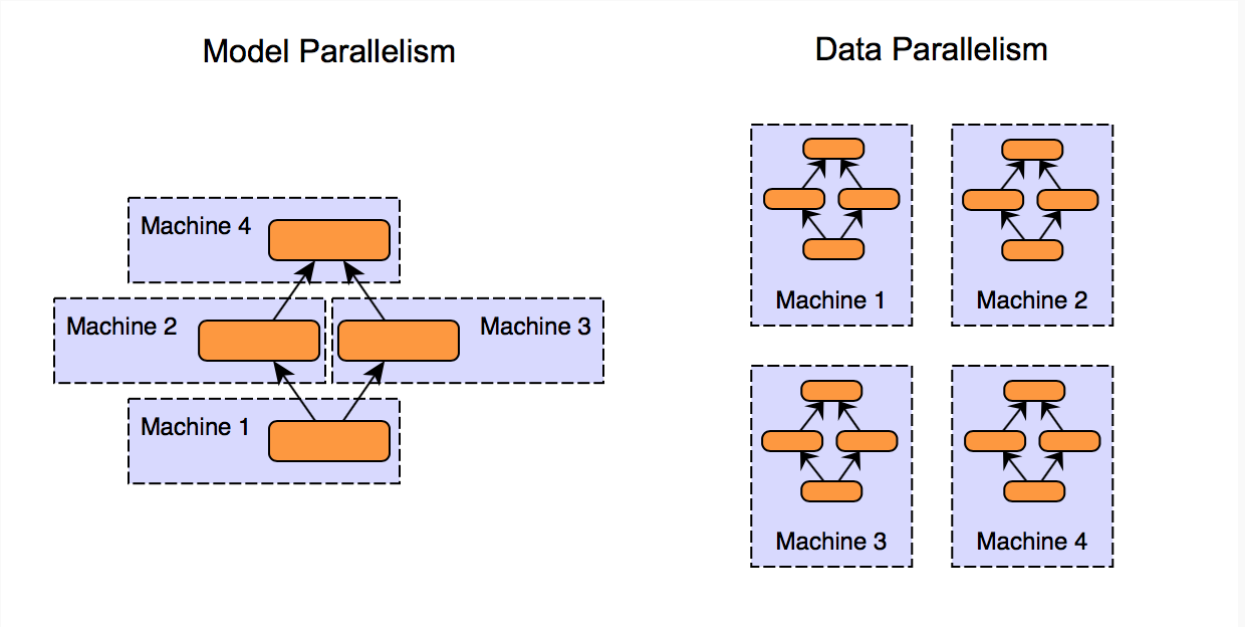
  - 같은 모델을 여러 머신에 넣어서 학습하는 방법

  - 많은 데이터를 한번에 학습하기 위해서 사용(Muti-GPU)

- **Model Parallelism**

  - 하나의 모델을 분리하여 여러 머신에서 학습(Distribution Computing)

  - 모델의 크기가 하나의 GPU에 메모리 크기보다 클 때 사용



https://www.cs.toronto.edu/~ranzato/publications/DistBeliefNIPS2012_withAppendix.pdf
https://xiandong79.github.io/Intro-Distributed-Deep-Learning

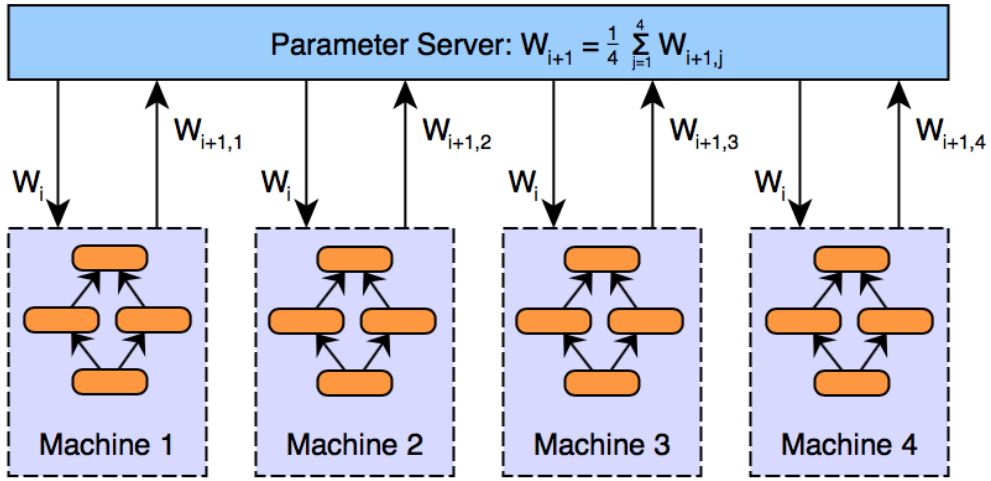
Parallelism

- Communication Cost in data Parallelism

  - 통신 비용은 신경망의 파라미터 수에 의존적

  - AlexNet의 경우 62M의 파라미터를 가지고 있으며 이는 250MB의 크기를 가짐

  - 이는 각 이터레이션마다 Parameter Server역활을 하는 머신에게 250MB를 전송해야한다는 것을 의미



https://www.cs.toronto.edu/~ranzato/publications/DistBeliefNIPS2012_withAppendix.pdf
https://xiandong79.github.io/Intro-Distributed-Deep-Learning

Frameworks

- Deep learning Frameworks

  - 본 논문에서는 CNTK, TensorFlow, Torch(Lua), Theano, Caffe를 대상으로 비교

  - 프레임워크의 비교는 Convolution Operation, Data Parallelism을 대상으로 함

TABLE I: Deep Learning Frameworks Used

Framework	Version	Multi-GPUs (parallelism)	Operating system	User interface	Libraries used
Caffe	1.0.0-rc3	Data	Linux	MATLAB, protobuf, Python	CUDA 7.5, cuDNN 5.0.5
CNTK	1.7.2	Data	Linux, Windows	BrainScript, C++, C#	CUDA 7.5, cuDNN 5.0.5
TensorFlow	0.10.0rc0	Data, Model	Linux	Python, C++	CUDA 7.5, cuDNN 5.0.5
Theano	0.8.2	–	Linux	Python	CUDA 7.5, Lasagne 0.2, cuDNN 5.0.5
Torch	7	Data, Model	Linux	LuaJIT	CUDA 7.5, cuda-convnet3, cuDNN 5.0.5, cuDNN 5.1.3, ccn2.torch

*All the frameworks spupport a single GPU.

Frameworks

: Convolution Algorithms

- **Caffe**

  - Convolution Algorithms을 명시적으로 선택할 수 있는 옵션이 없으며, cuDNN의 휴리스틱 알고리즘을 활용하여 암묵적으로 적합한 Convolution Algorithms을 선택함

  - 또한 적절한 메모리 관리 메커니즘이 없음. 이로 인해 cuDNN의 휴리스틱 알고리즘을 사용하더라도 GEMM 및 Winograd와 같이 더 작은 메모리 공간이 필요한 알고리즘을 부적절하게 선택해 성능 저하가 발생할 수 있음

- **TensorFlow**

  - Convolution Algorithms을 명시적으로 선택할 수 있는 옵션이 없음, 다만 Caffe와는 다르게 처음 시작 시, 가능한 알고리즘을 모두 실행해서 각 레이어에서 가장 빠른 실행시간을 갖는 알고리즘을 선택

  - 선택할 수 있는 알고리즘이 고정되어 있으며 cuDNN R5.1에서 제공하는 옵션이 포함되지 않음

  - AlexNet의 첫번째 Convolution Layer가 Stride가 1 이상이어서 FFT 사용 불가

- **Theano**

  - Convolution Algorithms을 명시적으로 선택할 수 있는 옵션 제공.

  - GEMM과 cuDNN으로 구현된 구현체를 선택할 수 있으며, Layer 전역에 설정할 수 있음

  - 알고리즘이 레이어의 조건과 일치되지 않으면 암묵적으로 GEMM구현체가 선택됨

  - 명시적인 GEMM 구현체는 암묵적인 GEMM 구현체보다 느림

  - AlexNet의 첫번째 Convolution Layer가 Stride가 1 이상이어서 FFT 사용 불가

  - Convolution Operation은 제일 빠른 알고리즘을 실행하나 Bias와 ReLU로 인해 Convolution Layer 연산이 느려짐

  - GPU 구현체가 런타임시 동적으로 컴파일 되어 다른 프레임워크에 비교했을 때, 눈에 띄게 느림

Frameworks

: Convolution Algorithms

- Torch

  - Convolution Algorithms을 명시적으로 선택할 수 있는 옵션 제공.

  - cuDNN 백엔드를 사용시 자동으로 최적의 Convolution Algorithms을 선정함. 이 경우 Torch의 Convolution Layer 연산이 다른 프레임워크 중에서 가장 빠름

- CNTK

  - Convolution Algorithms을 명시적으로 선택할 수 있는 옵션을 제공하지 않음

  - Theano나 Torch와 같이 cuDNN 함수가 가장 빠른 Convolution Algorithms을 찾음

  - Bias addition 구현체로 인해 다른 프레임워크 중에서 Convolution Algorithm이 제일 느림

TABLE IV: Other GPU Kernels

	Caffe	CNTK	TensorFlow	Theano	Torch
Bias addition	cuDNN 2.64 ms	CNTK 4.74 ms	TensorFlow 2.84 ms	Theano 7.88 ms	cuDNN 2.63 ms
ReLU activation	cuDNN 2.56 ms	CNTK 2.26 ms	Eigen 2.43 ms	Theano 4.59 ms	cuDNN 2.56 ms

Experiments

- Environments
  - cuDNN R 5.1 사용
  - Titian-X 4ea 사용

TABLE III: System Configuration

CPU	2 x Intel Xeon E5 2650@2.0GHz
GPU	4 x NVIDIA Titan X (GM200)
Main memory	128GB DDR3
GPU memory	4 x 12GB GDDR5
Operating system	CentOS 7.2.1511 (Linux 3.10.0-327)

Experiments

: Convolution Algorithms

• Effect of Data Layout

- Caffe, CNTK, Theano, Torch는 NCHW 4d tensor를 사용

- TensorFlow는 유일하게 NHWC 4d tensor를 사용

- NHWC 4d tensor는 channel-wise operation에서 더 좋은 성능을 내기 때문에 사용하는 것으로 추정

- cuDNN에서 제공하는 FFT Convolution은 NHWC 4d tensor를 지원하지 않음. 따라서 TensorFlow는 NHWC를 NCHW로 변경한 후에 NHWC로 다시 변경하는 코드가 존재

- 이는 컨버팅 오버헤드를 발생시키며 이를 제거 시 15%의 성능 향상이 있음

• Unnecessary Backpropagation

- 첫번째 Convolution Layer는 Backpropagation을 위한 연산을 할 필요가 없음

- Caffe, CNTK, Theano는 첫번째 Convolution Layer에서는 Backpropagation을 위한 연산을 수행하지 않음

- TensorFlow는 race condition을 방지하기 위해서 첫번째 Convolution Layer도 Backpropagation을 수행함

- 이로 인해 TensorFlow의 Convolution Algorithms의 연산 속도가 Torch보다 느림

Experiments

: Convolution Algorithms

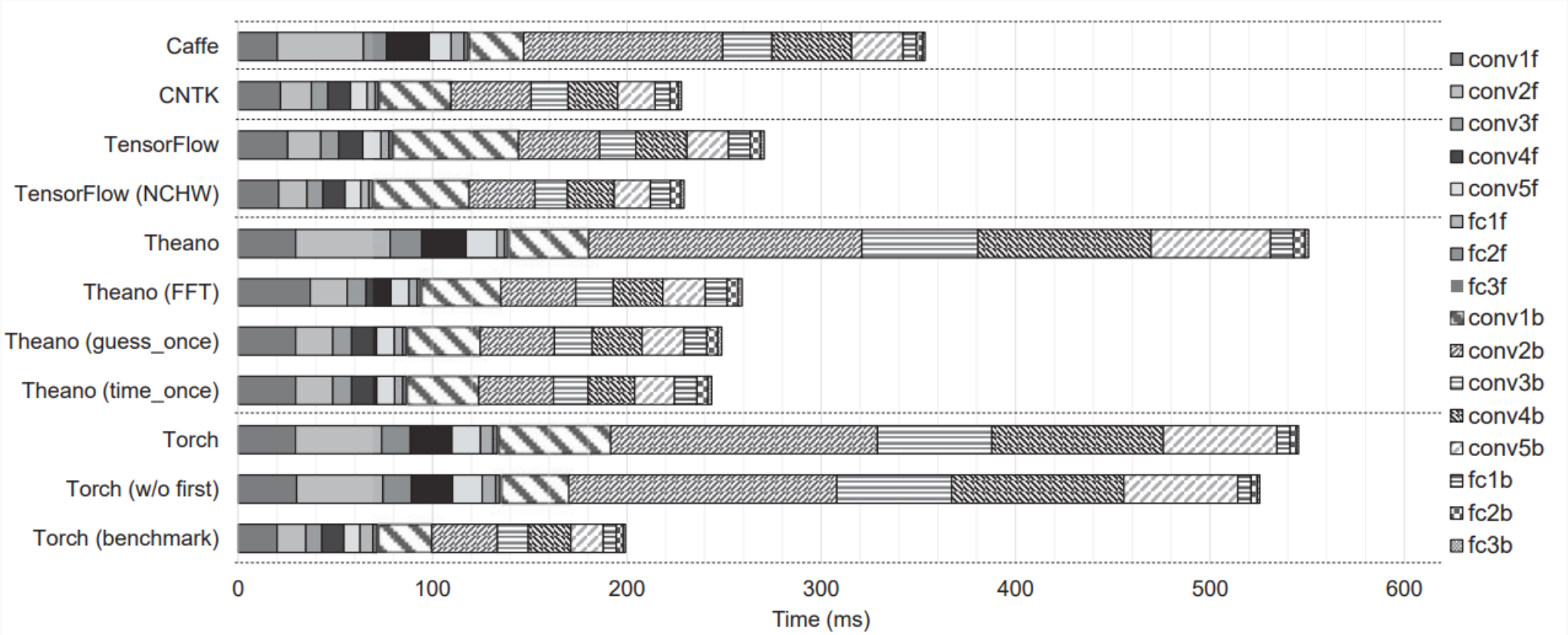


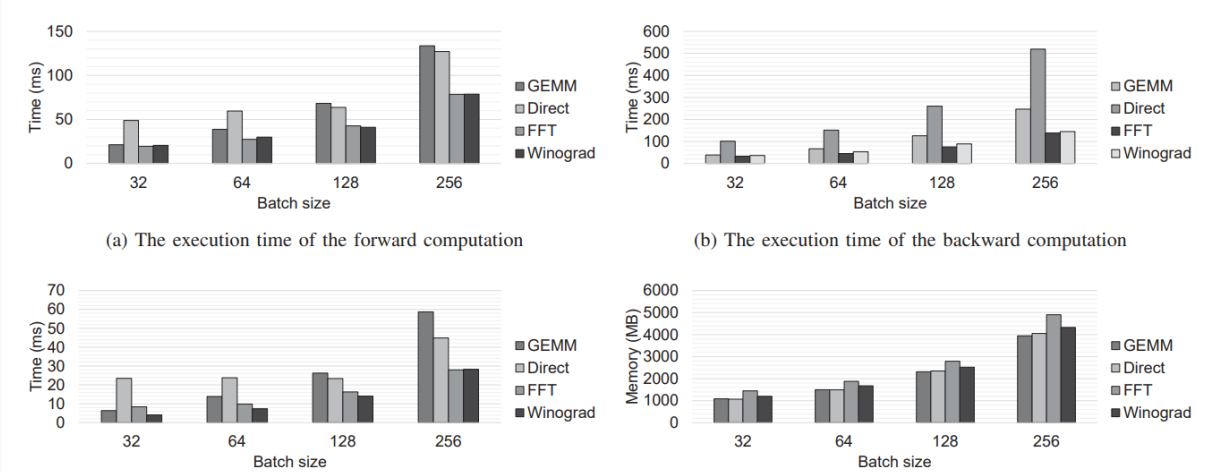
Fig. 7: The execution time of training AlexNet with a batch of input images for different deep learning frameworks.

Experiments

: Convolution Algorithms

- **Execution Time of Convolution Algorithms**
  - Winograd나 FFT가 Direct나 GEMM보다 성능이 좋음
  - 학습 시간 측면에서는 FFT가 제일 좋음
  - 작은 배치인 경우에는 Winograd가 FFT보다 좋음

- Torch7을 기반으로 Convolution 연산 속도를 비교(최신 버전의 cuDNN, CUDA 적용)
- -100 이터레이션의 평균으로 측정
- 파일 I/O 지연시간을 최소화하기 위해 랜덤으로 생성된 이미지를 사용
- FFT가 GPU 메모리 공간을 제일 많이 사용함



(a) The execution time of the forward computation

(b) The execution time of the backward computation

(c) The execution time of the forward computation from conv3 to conv5

(d) Peak GPU memory usage

Fig. 8: Execution time and memory usage between different convolution algorithms.

Experiments

: Convolution Algorithms

- **Layer-wise Analysis**

  - Batch 256에서 실험, NVIDIA nvprof profiler를 이용해서 통계값 기록

  - Direct Convolution의 경우 구현체의 스레드가 1024가 최대값으로 Titan X의 CUDA 3072개를 모두 사용하지 못해 성능 한계가 보임

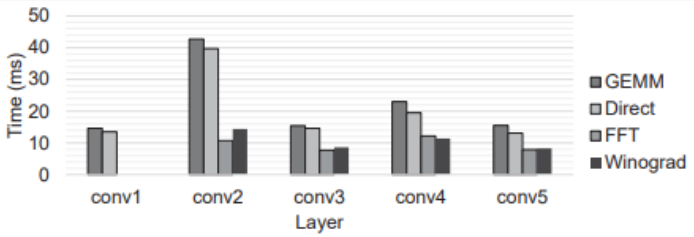
- **Floating-points operation counts**

  - NVIDIA nvprof profiler를 사용하여 카운팅

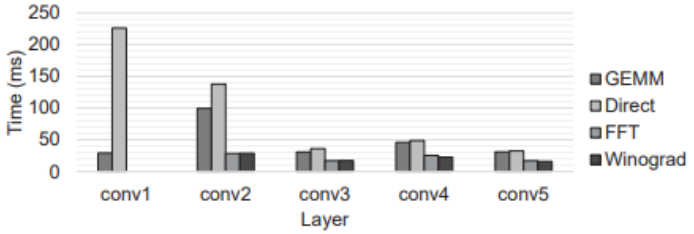
  - FFT 알고리즘이 연산복잡도와 필터 사이즈의 영향을 받지 않아 Floating-points 연산이 제일 적음

  - Winograd 또한 Floating-points 연산이 반 이상으로 줄어듬

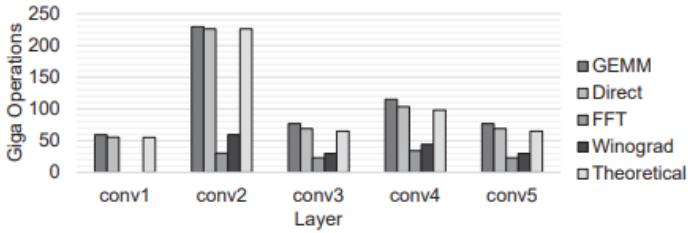
  - FFT, Winograd가 다른 알고리즘(Direct, GEMM)보다 Floating-points 연산이 적음



(a) The execution time of the forward computation



(b) The execution time of the backward computation



(c) FP operation count in the forward computation

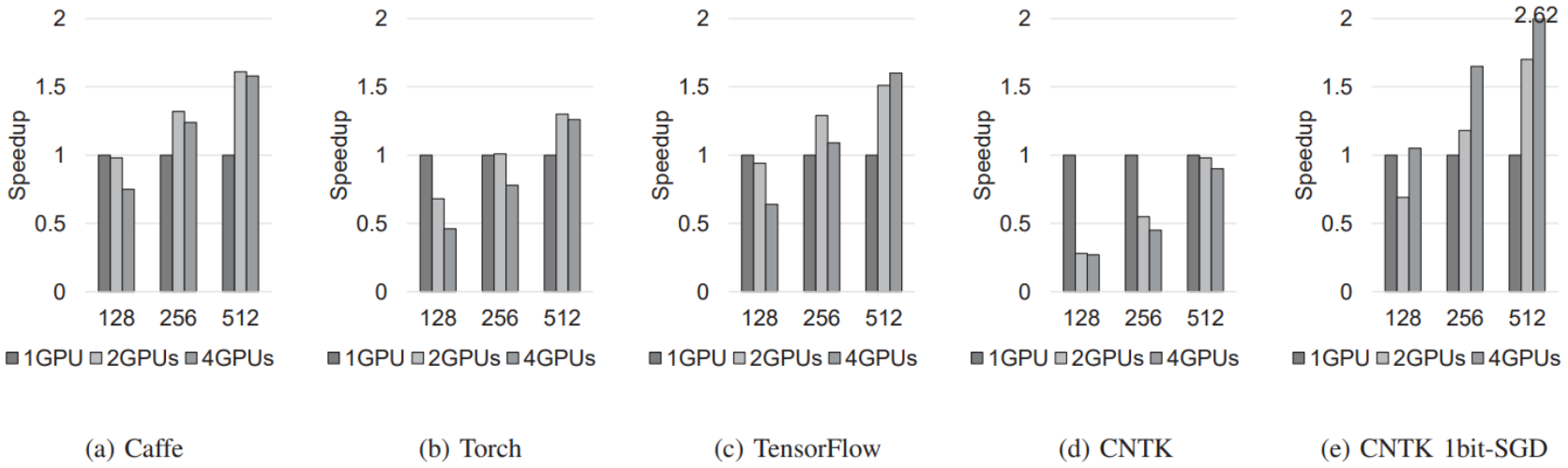
Fig. 9: Layerwise analysis of different convolution algorithms.

Experiments

: Parallelism

- **Multi-GPU**
  - Theano는 Multi-GPU를 지원하지 않음
  - Caffe, CNTK, TensorFlow, Torch는 Multi-GPU 지원

- **Data Parallelism**
  - 본 논문에서는 Data Parallelism으로만 성능 비교
  - 그레디언트 수집으로 인한 통신 오버헤드는 $O(N-1)$
    N: GPU 개수



(a) Caffe (b) Torch (c) TensorFlow (d) CNTK (e) CNTK 1bit-SGD

Fig. 10: Speedup (over a single GPU) of multi-GPU training of the AlexNet models.

Experiments

: Parallelism

- Single-GPU

  - Torch와 TensorFlow를 비교시 Signle GPU에서의 학습 성능은 TensorFlow가 더 좋음

  - 이는 전체 gradient를 수집하고 나서 전송하는 Torch와 layer단위로만 gradien를 수집하고 전송하는 TensorFlow의 gradient 수집 방식에 기인

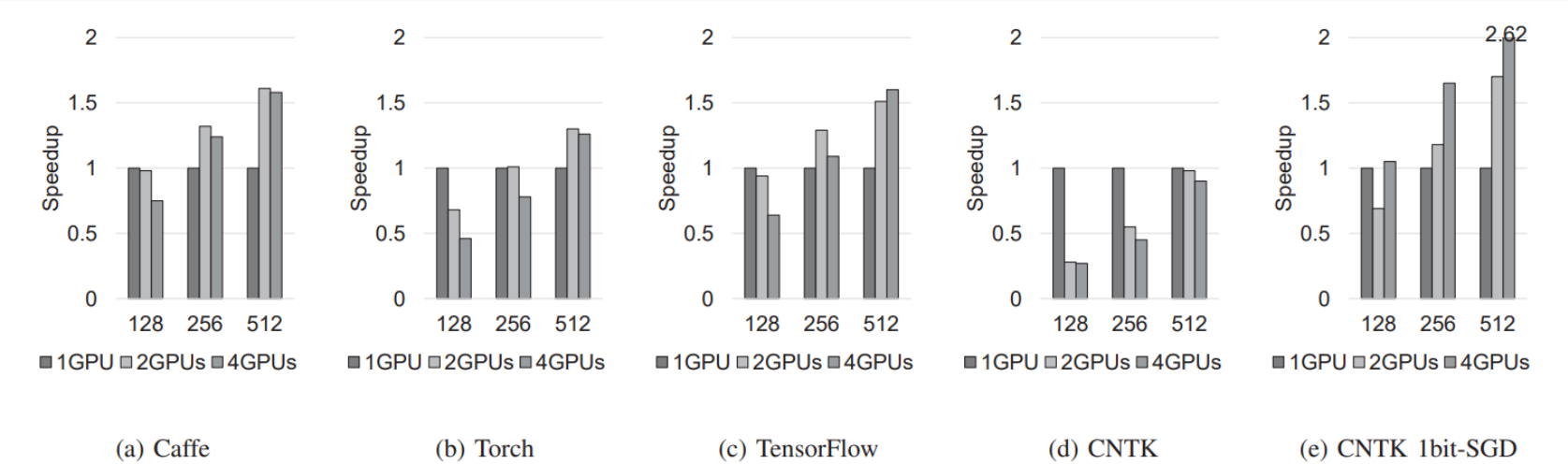


Fig. 10: Speedup (over a single GPU) of multi-GPU training of the AlexNet models.

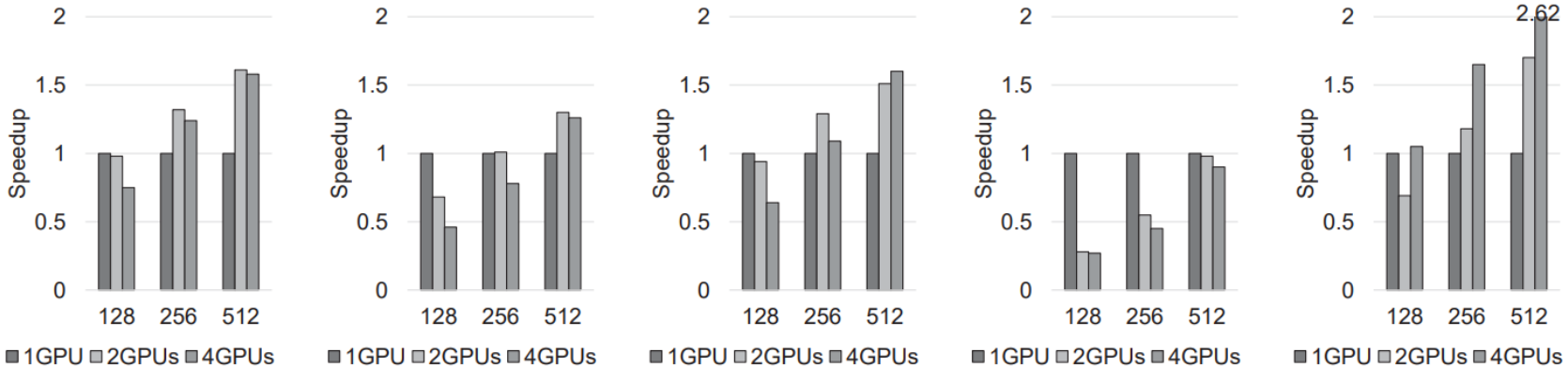
Experiments

: Parallelism

- Data Parallelism

  - 통신과 연산을 병렬로 처리한다면 전체 오버헤드는 $O(logN)$로 감소 가능

  - AlexNet 모델 크기인 250MB를 전송하는데 걸리는 시간은 25ms. Batch가 256인 경우 forward/backward의 연산시간 총합이 200ms 가 걸리는 것을 감안할 때, 굉장히 큰 시간 지연



(a) Caffe (b) Torch (c) TensorFlow (d) CNTK (e) CNTK 1bit-SGD

Fig. 10: Speedup (over a single GPU) of multi-GPU training of the AlexNet models.

Experiments

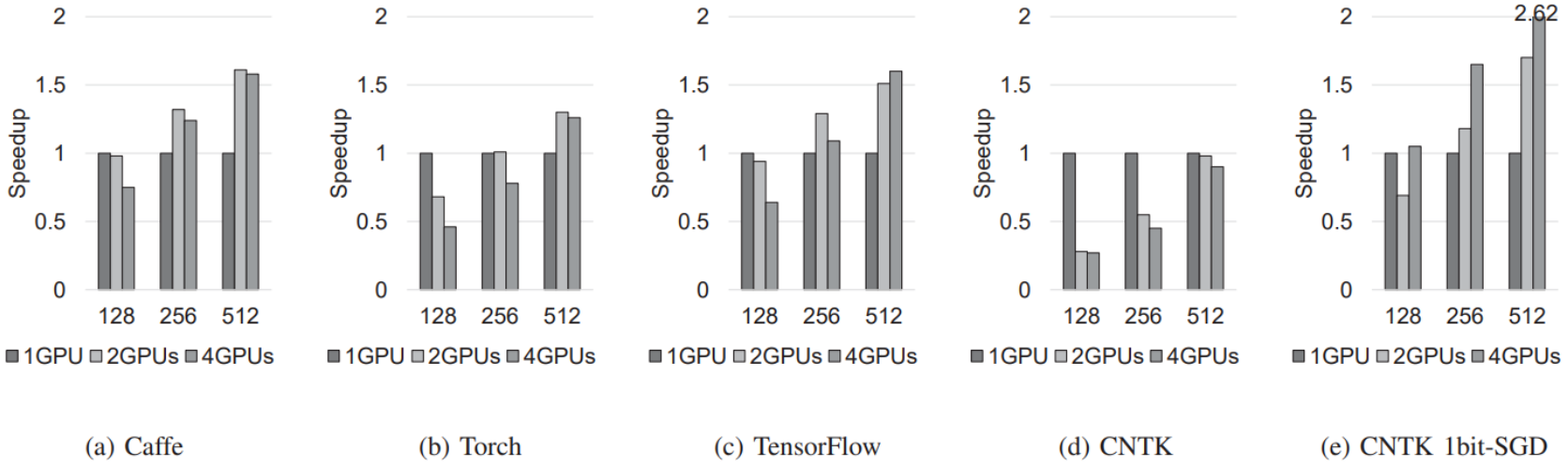
: Parallelism

- Data Parallelism

  - Batch가 클 수록 학습 속도가 빨라짐

  - Batch가 128인 경우 Caffe, Torch에서 2~4개의 GPU로 학습하는 것보다 단일 GPU가 학습 성능이 더 좋음

  - Batch가 클 경우 연산은 더 많이 하지만 통신은 적게함. 따라서 통신 오버헤드가 상대적으로 줄어듬



(a) Caffe (b) Torch (c) TensorFlow (d) CNTK (e) CNTK 1bit-SGD

Fig. 10: Speedup (over a single GPU) of multi-GPU training of the AlexNet models.

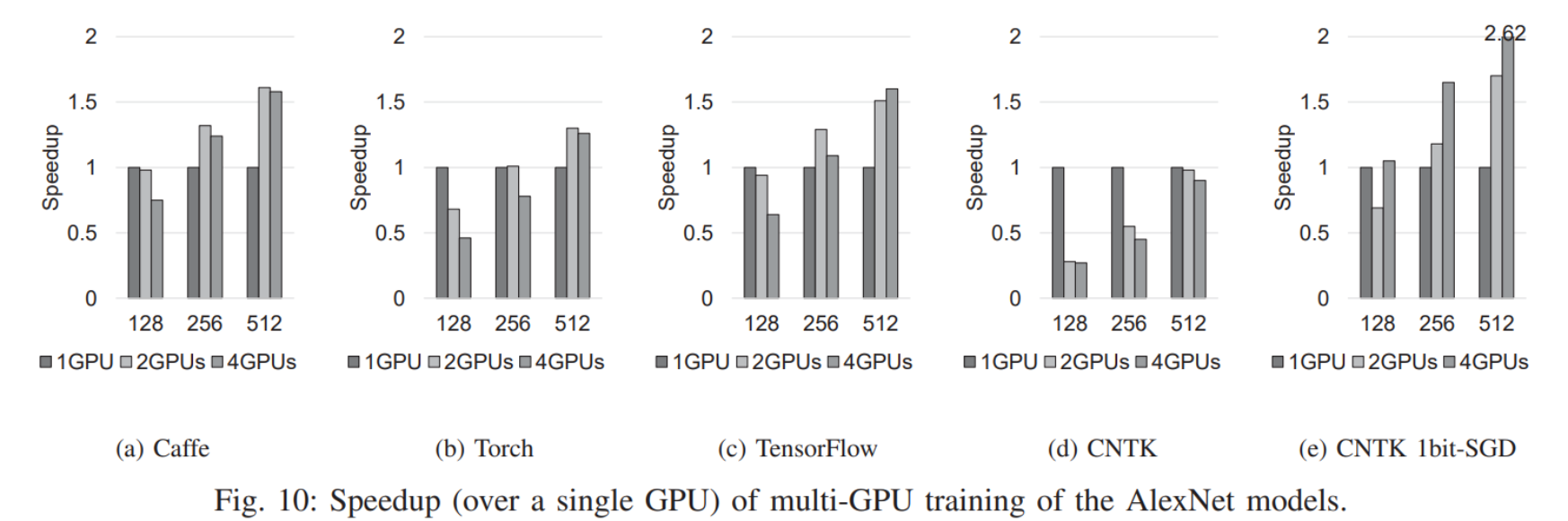
Experiments

: Parallelism

- Scalability

  - 모든 프레임워크는 Scalability 성능이 떨어짐

  - $O(logN)$이라는 복잡도로 인해서 GPU가 4개인 경우 학습 성능이 GPU2개와 비슷하거나 더 낮음.



(a) Caffe (b) Torch (c) TensorFlow (d) CNTK (e) CNTK 1bit-SGD

Fig. 10: Speedup (over a single GPU) of multi-GPU training of the AlexNet models.

Conclusion

- Convolution Operation은 FFT와 Winograd가 연산 효율이 좋다.

- Multi-GPU의 경우 네트워크 파라미터 크기에 따른 통신 오버헤드가 크다.
  통신 오버헤드를 줄여주는 것이 병렬화에 큰 도움이 된다.

- 프레임워크의 각 특징을 잘 파악한 다음에 자신의 환경에 알맞은 프레임워크를 잘 조절해서 쓰는 것이 중요하다.

Framework에서 제공하는 옵션을 잘 조절하면 프레임워크 성능을 코드 수정없이
2배 이상 증가시킬 수 있다.

감사합니다