Performance Analysis of CNN Frameworks for GPUs

Heehoon Kim†, Hyoungwook Nam†, Wookeun Jung, and Jaejin Lee
Department of Computer Science and Engineering
Seoul National University, Korea
http://aces.snu.ac.kr
†The two authors contributed equally to this work as the first authors

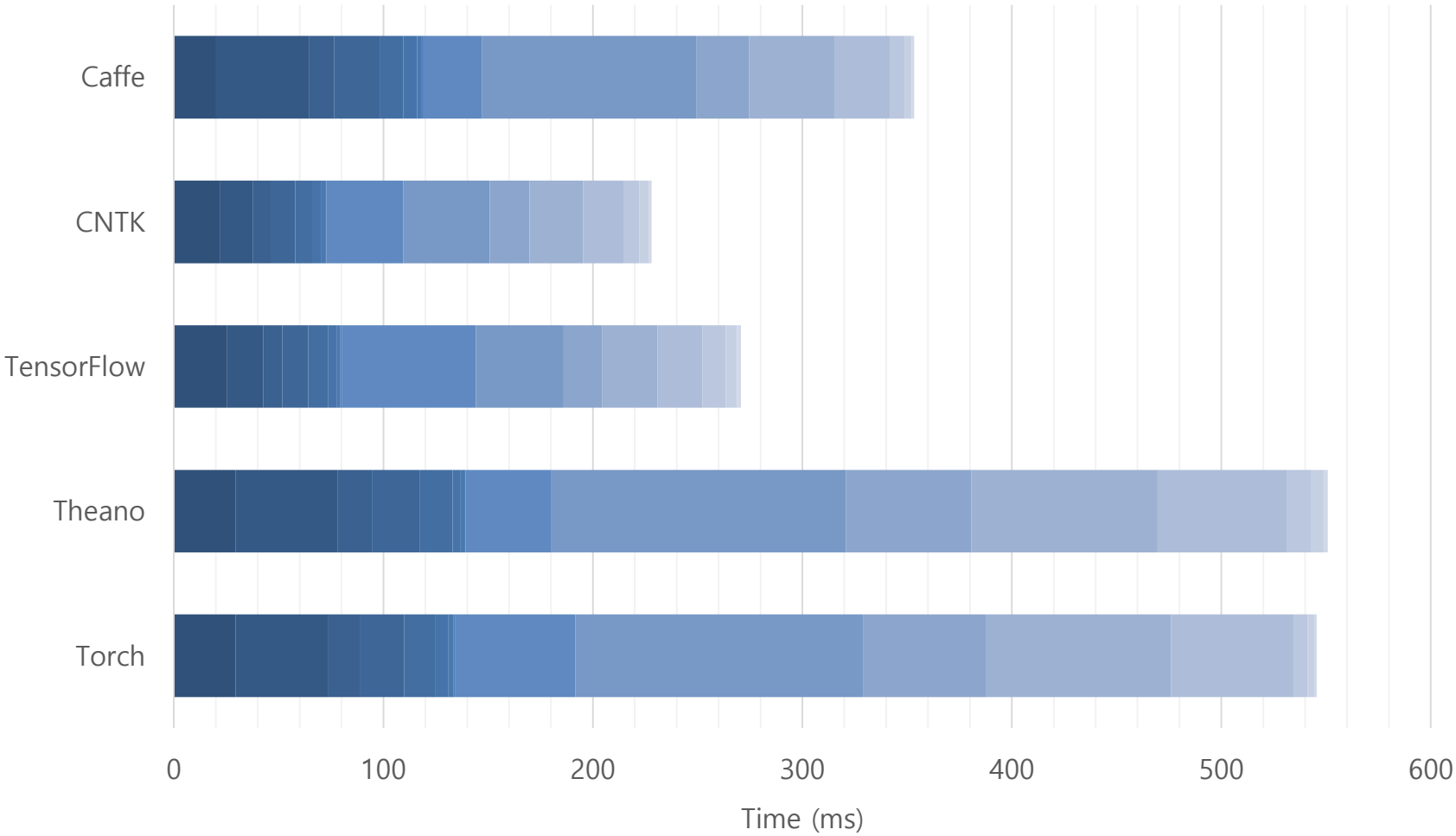
Convolutional Neural Network

Deep Learning Framework

GPU Library

Motivation

- Convolutional Neural Networks (CNN) have been successful in machine learning tasks such as visual recognition
- Previous studies reveal **performance differences** among deep learning frameworks
- However, those studies **do not identify reasons** for the differences


Caffe
CNTK
TensorFlow
Theano
Torch
0
100
200
300
400
500
600
Time (ms)

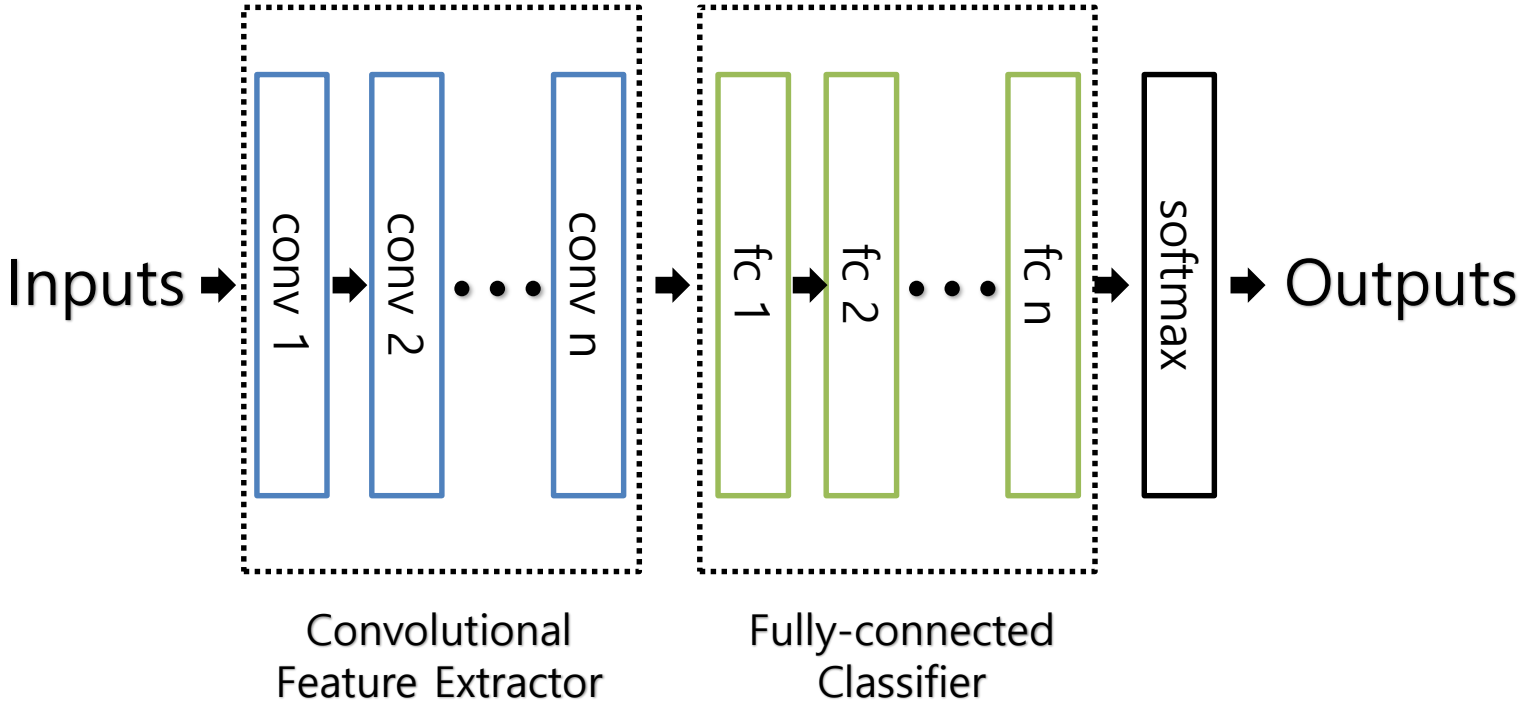
Goals

- Analyze differences in the performance characteristics of the **five deep learning frameworks** in a single GPU context
- Analyze scalability of the frameworks in the **multiple GPU context**
- Analyze performance characteristics of **different convolution algorithms** for each layer

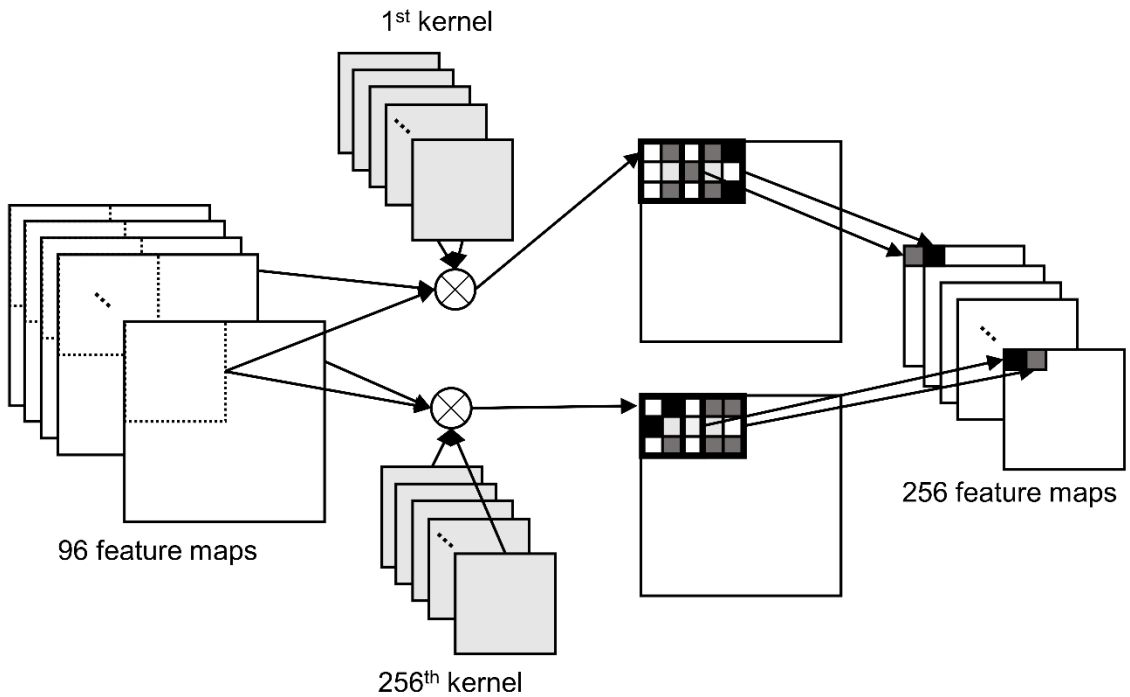
Outline

- **Convolutional Neural Network**
- Deep Learning Frameworks
- Framework Comparison
- Multi-GPU Comparison
- Layer-wise Analysis of Convolution Algorithms
- Conclusions

Convolutional Neural Network



Computational Complexity of Convolution



Conv2 layer

C = 96
(input channel)

[H,W] = [13, 13]
(input dimension)

[R,S] = [5, 5]
(kernel dimension)

K = 256
(output channel)

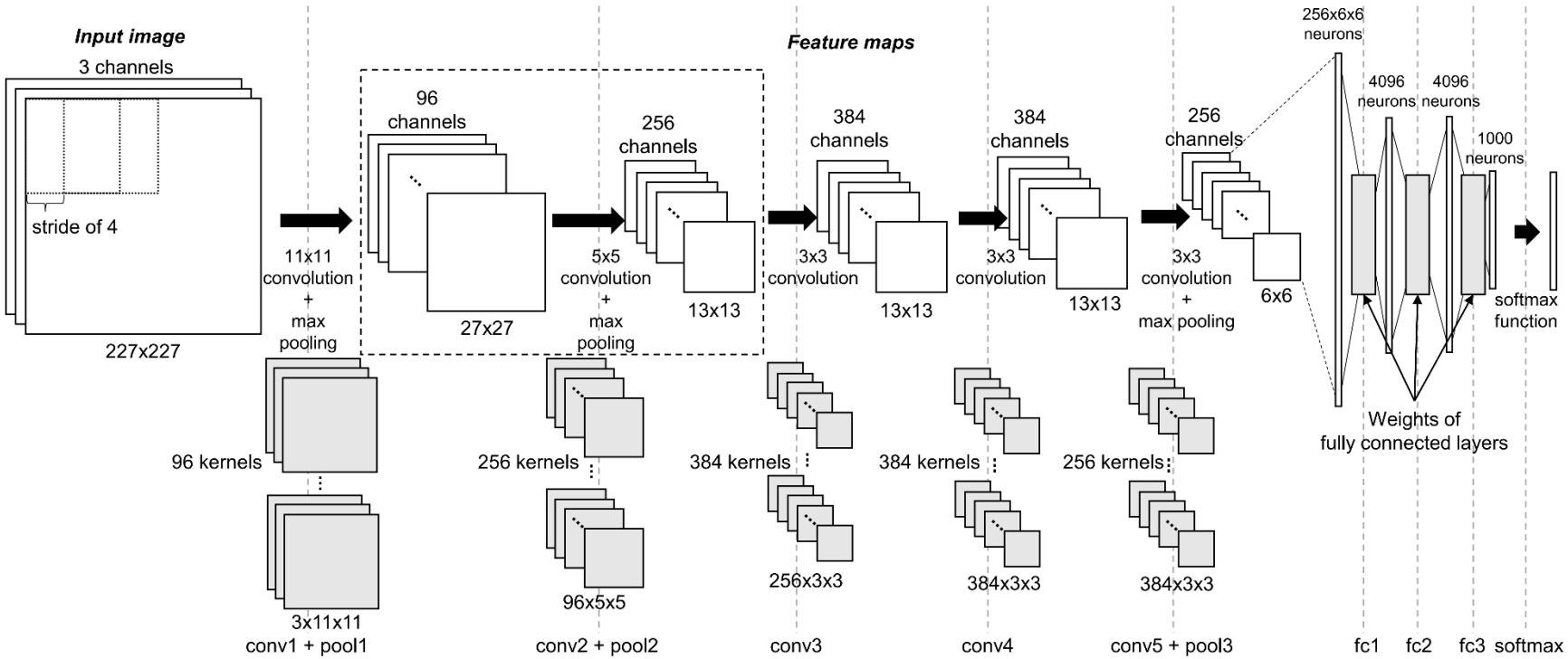
N = 256
(batch size)

- $C \times HW \times RS \times K \times N \times 2\ (multiply\ and\ add)$
- Ex) $96 \times 27 \times 27 \times 5 \times 5 \times 256 \times 256 \times 2\ =\ 229\ Gops$

Convolution Algorithms for GPU

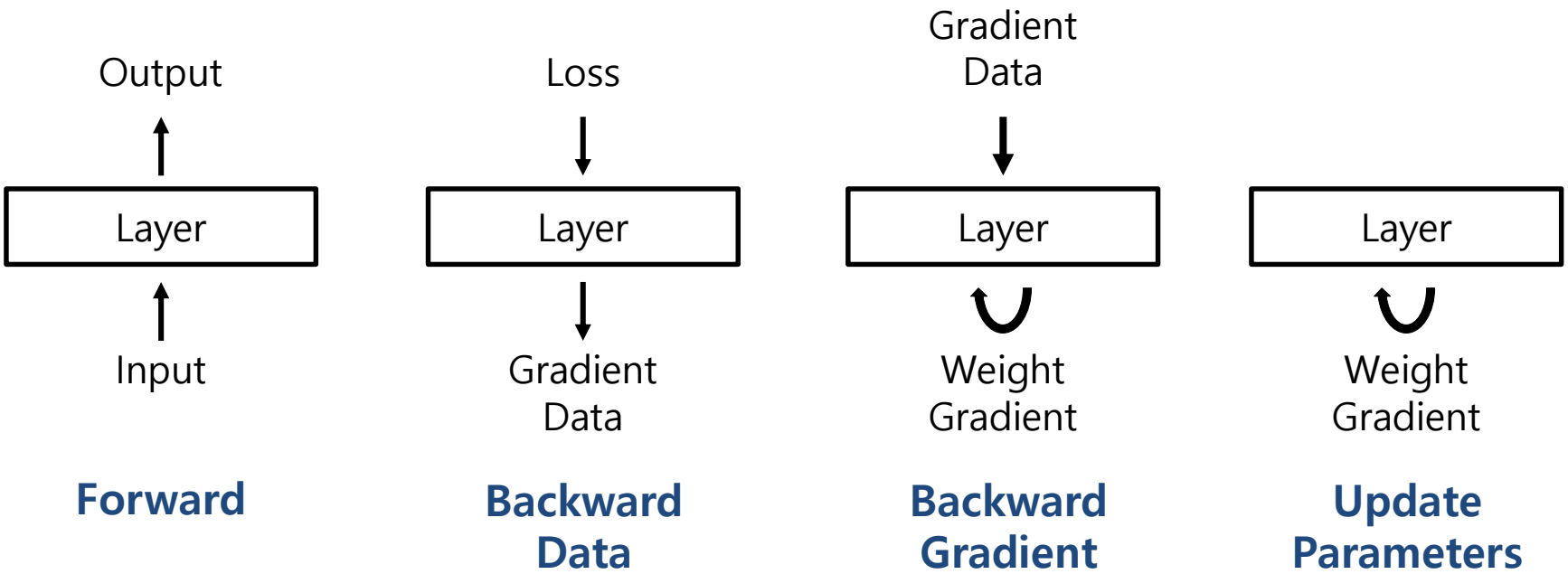
- **Direct Convolution**
  - Straightforward, but hard to optimize
- **GEMM Convolution**
  - Converts convolutions into matrix multiplications
  - Easier to optimize
- **FFT Convolution**
  - Reduced computational complexity
  - $O(KN)$ (Direct convolution) → $O(NlogN)$ (FFT convolution)
- **Winograd Convolution**
  - Reduces the complexity of convolution like Strassen's algorithm
  - Specific filtering algorithm is required for each kernel dimension

AlexNet Model



- Winner of ILSVRC 2012 (ImageNet Challenge)
- Commonly used CNN model for benchmarking
- Includes various kinds of layers
  - 3x3 convolution, 5x5 convolution, fully connected layers, etc.

Training a CNN



- **1 forward computation** and **2 backward computations**
- Forward and backward computations are **symmetric** and have the **same computational cost**

Outline

- Convolutional Neural Network
- **Deep Learning Frameworks**
- Framework Comparison
- Multi-GPU Comparison
- Layer-wise Analysis of Convolution Algorithms
- Conclusions

Five Deep Learning Frameworks

Framework	User Interface	Data Parallelism	Model Parallelism
Caffe	protobuf, C++, Python	Yes	Limited
CNTK	BrainScript, C++, C#	Yes	No
TensorFlow	Python, C++	Yes	Yes
Theano	Python	No	No
Torch	LuaJIT	Yes	Yes

- Popular frameworks chosen by GitHub stars
- All five frameworks use cuDNN as backend
- Theano only supports single GPU

cuDNN

- Deep Neural Network library with **NVIDIA CUDA**
- Provides DNN primitives
  - Convolution, pooling, normalization, activation, …
- State-of-the-art performance
- **All five frameworks** support use of cuDNN as a backend
- Unfortunately, not open-source (distributed in binaries)

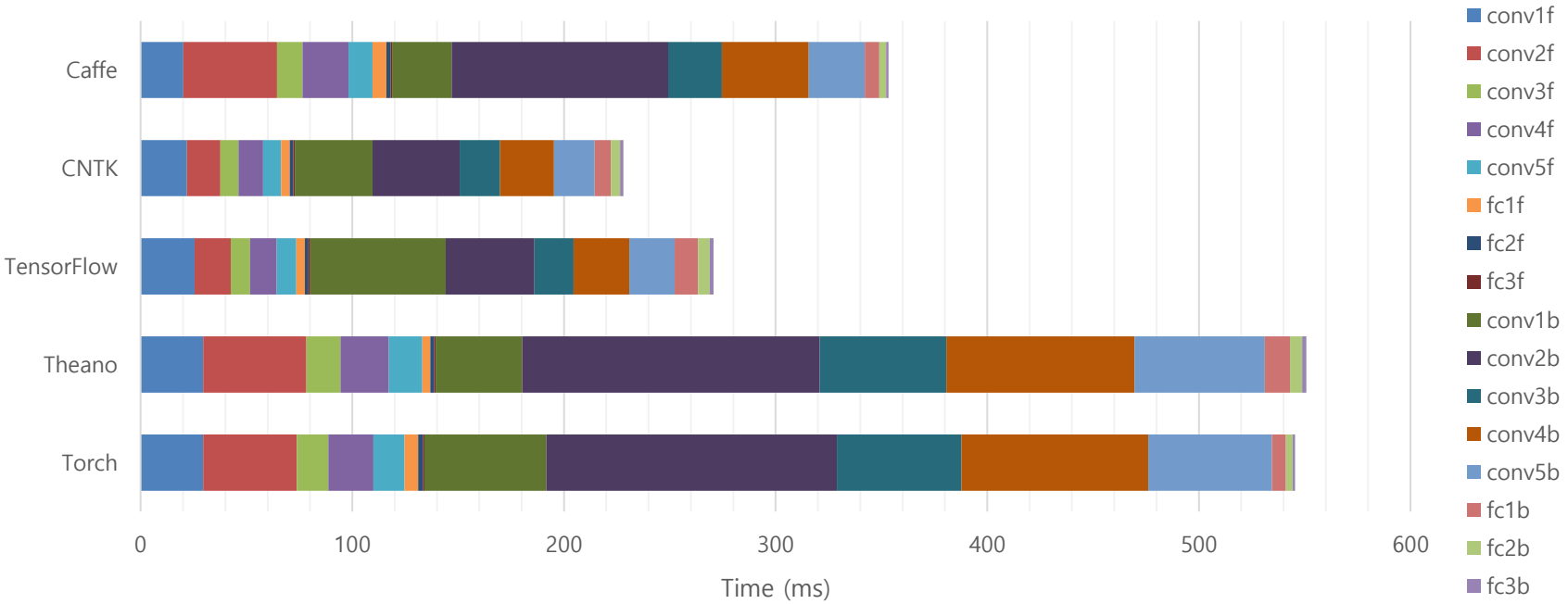
System Setup

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

Outline

- Convolutional Neural Network
- Deep Learning Frameworks
- **Framework Comparison**
- Multi-GPU Comparison
- Layer-wise Analysis of Convolution Algorithms
- Conclusions

Execution Time Comparison (default setting)



- **Convolution layers** take up **more than 70%** of training time
- f: forward computation, b: backward computation

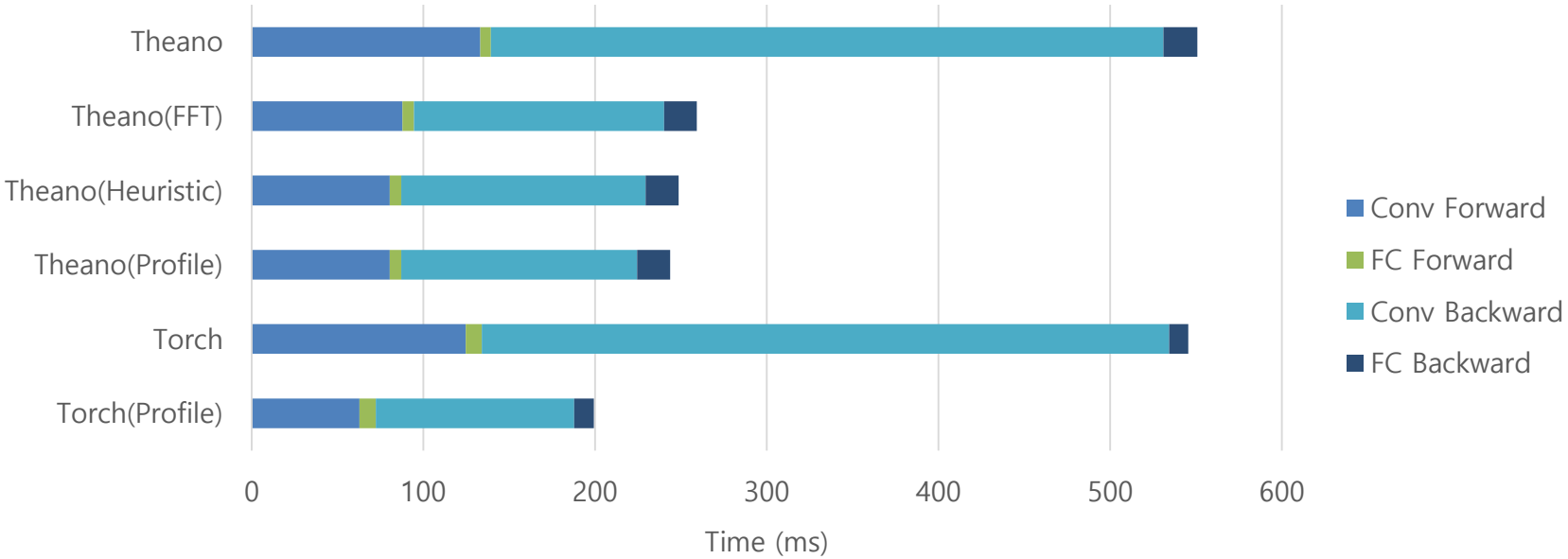
Options for Convolution Algorithms

Framework	User Selectable	Heuristic-based	Profile-based	Default
Caffe	No	Yes	No	Heuristic-based
CNTK	No	No	Yes	Profile-based
TensorFlow	No	No	No	Heuristic-based†
Theano	**Yes**	Yes	Yes	GEMM
Torch	**Yes**	Yes	Yes	GEMM

†TensorFlow uses its own heuristic algorithm

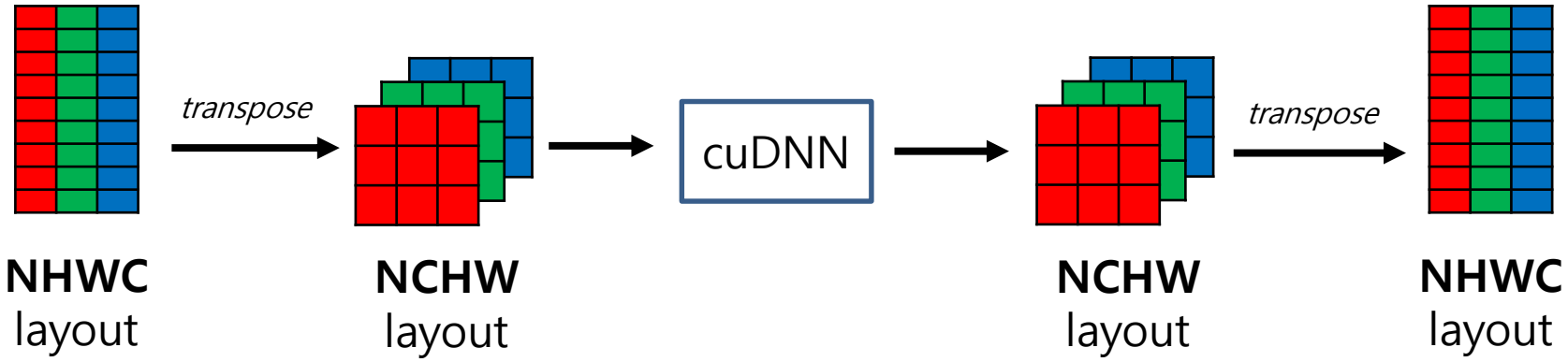
- cuDNN Get API is a **heuristic based** approach to choose an algorithm
- cuDNN Find API is a **profile-based** approach to choose an algorithm
- By default, Torch and Theano use GEMM convolution

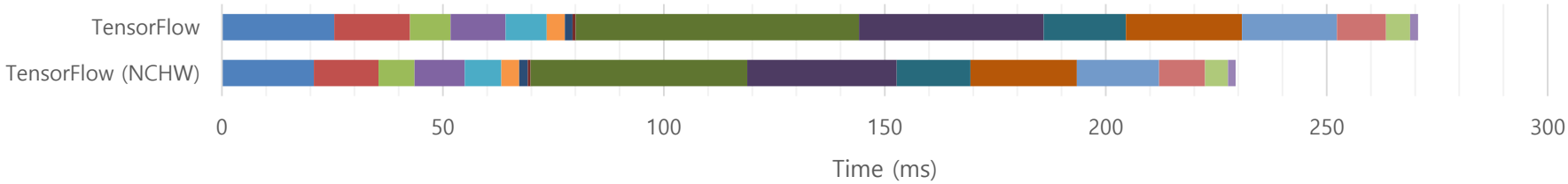
Options for Convolution Algorithms



- Up to 2x speedup by providing algorithm options

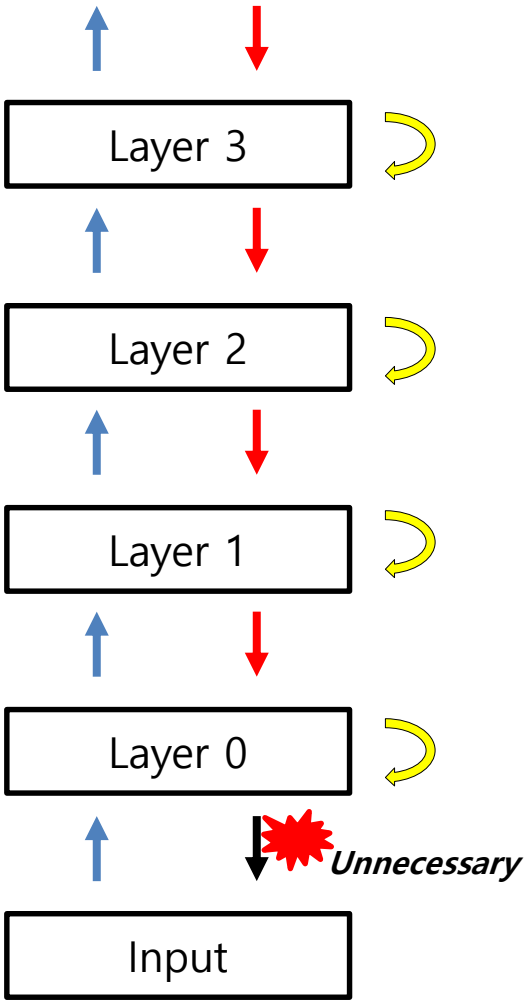
Data Layout





- For example, cuDNN's FFT convolution only supports NCHW
- If the user uses another layout, TensorFlow implicitly transposes
- Changing the layout leads to **15% speedup** in TensorFlow

Unnecessary Backpropagation



Forward

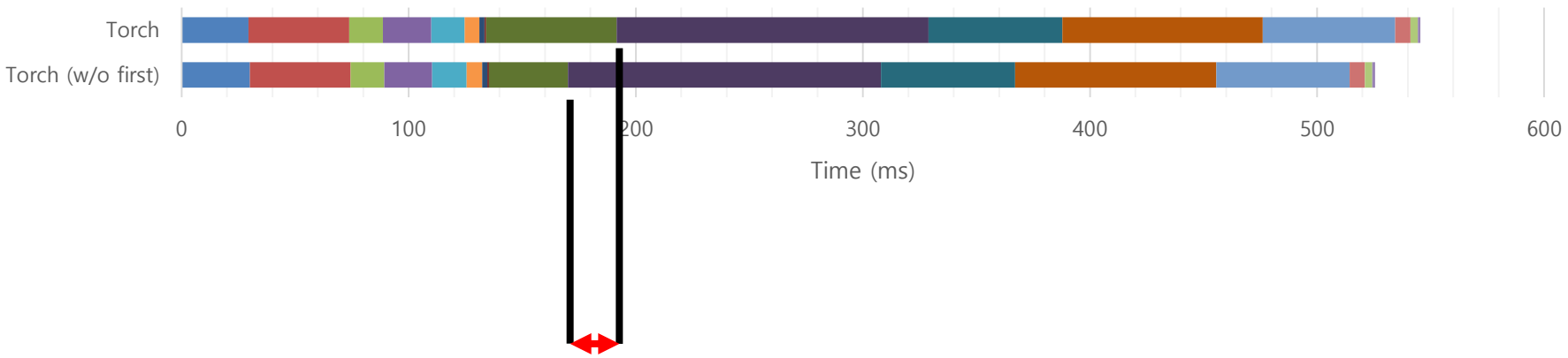
Backward Data

Backward Gradient

- 'Backward Data' is unnecessary in the **first layer**.

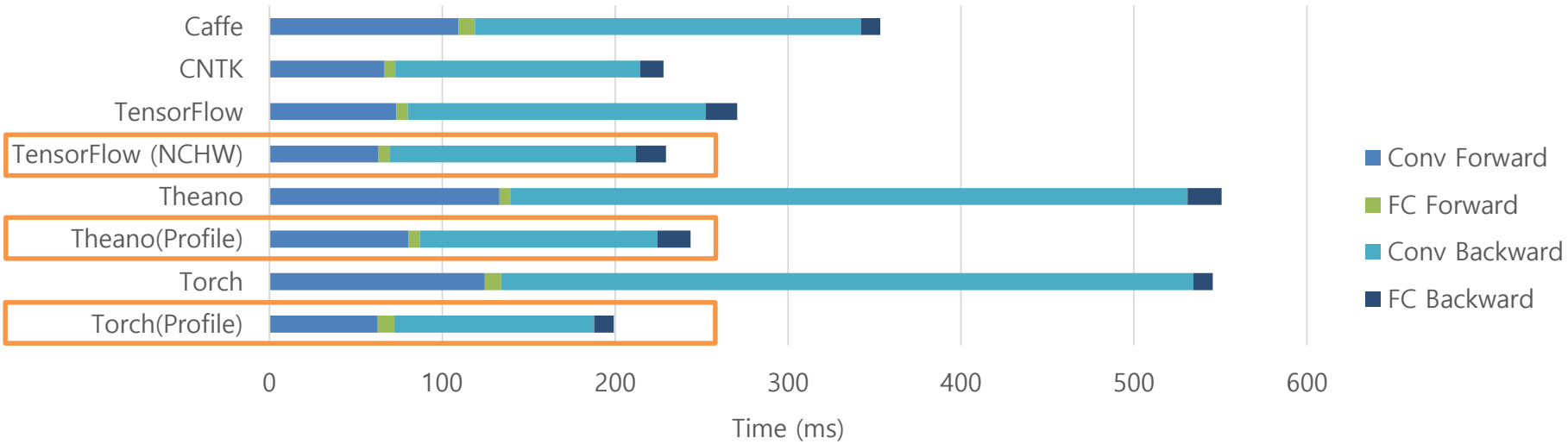
- **Caffe, CNTK, Theano**
  - Automatically omitted.
- **Torch**
  - User option (layer0.gradInput = nil)
- **TensorFlow**
  - No options to users

Unnecessary Backpropagation



- Speedup in the backward computation of the first layer

Optimized Results

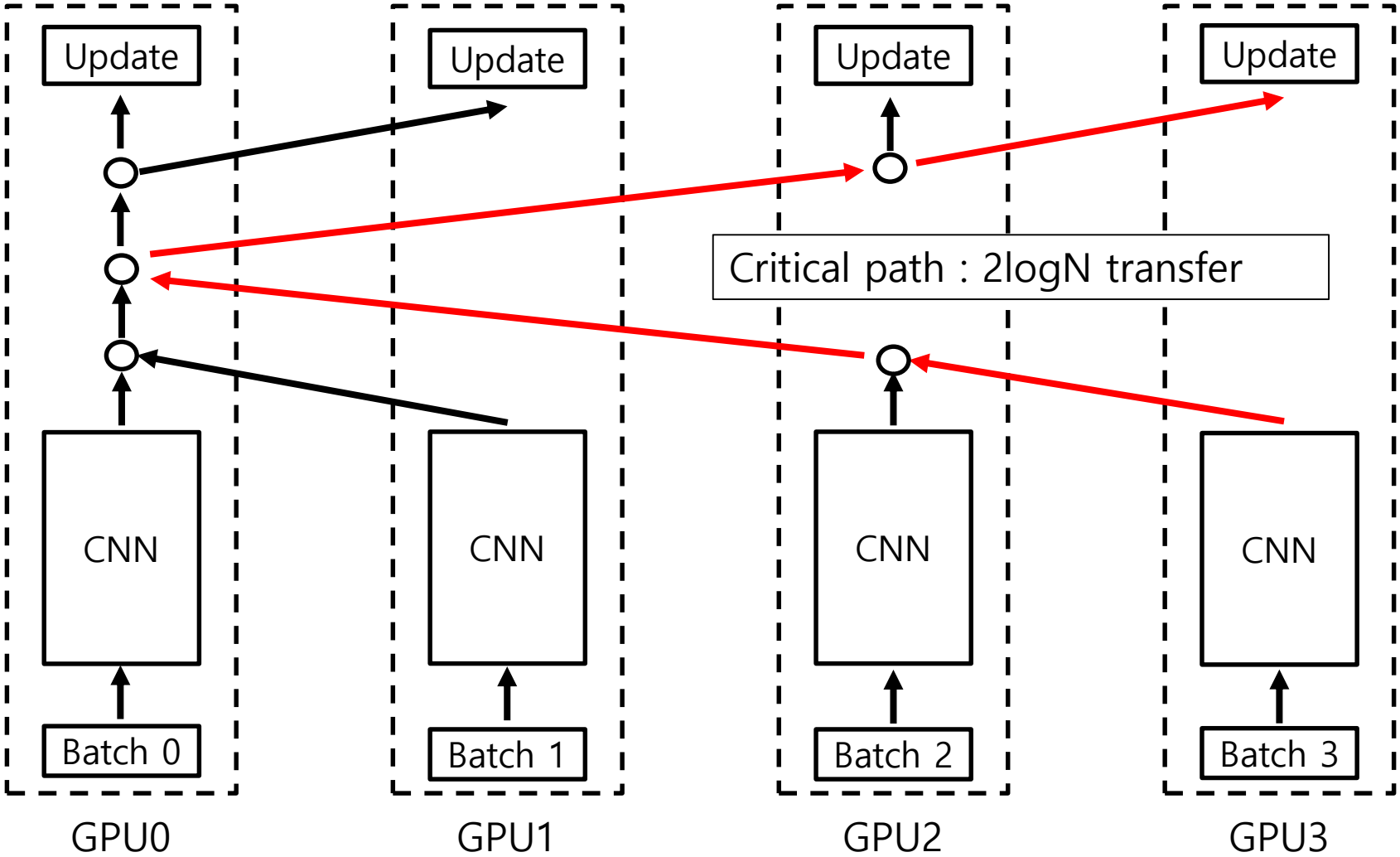


- Framework differences are not significant if carefully optimized
- Remaining differences come from other operations, such as bias addition and ReLU activation

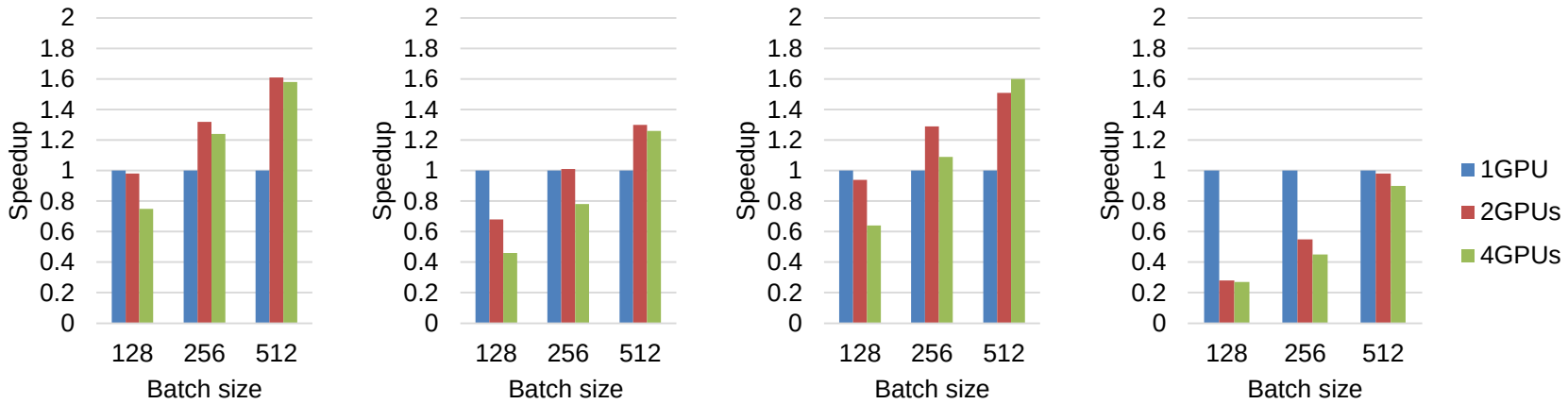
Outline

- Convolutional Neural Network
- Deep Learning Frameworks
- Framework Comparison
- **Multi-GPU Comparison**
- Layer-wise Analysis of Convolution Algorithms
- Conclusions

Data-parallel SGD



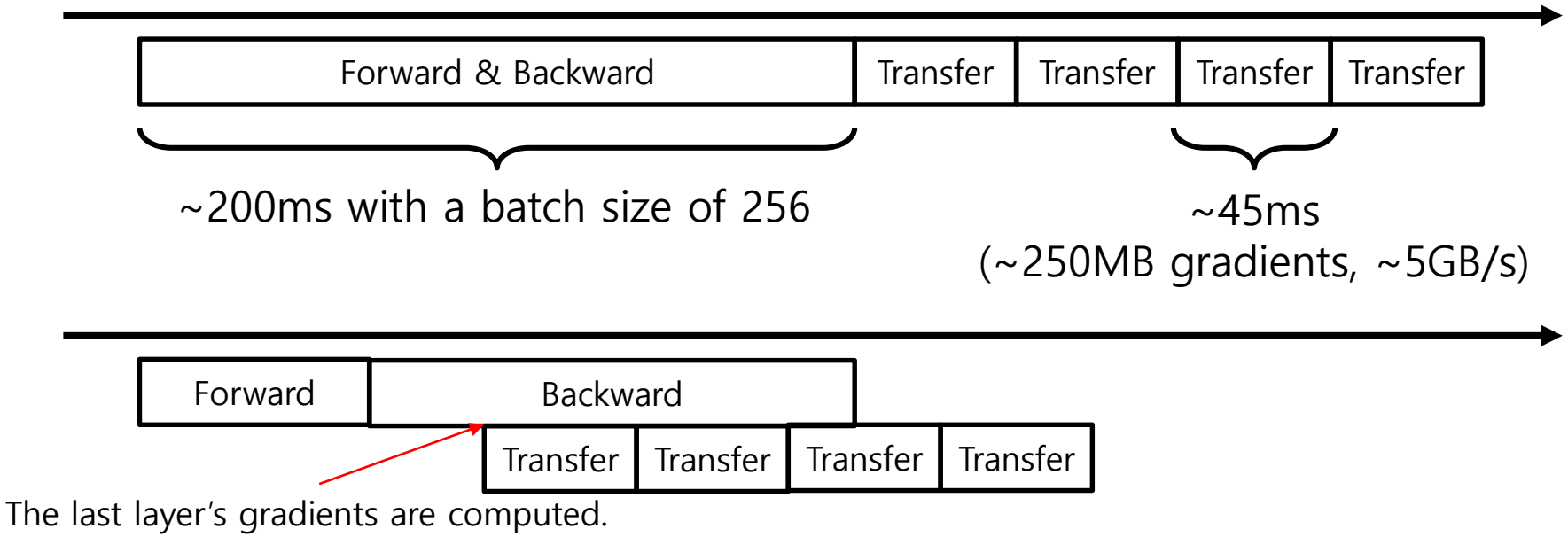
Multi-GPU Scalability



Caffe Torch TensorFlow CNTK

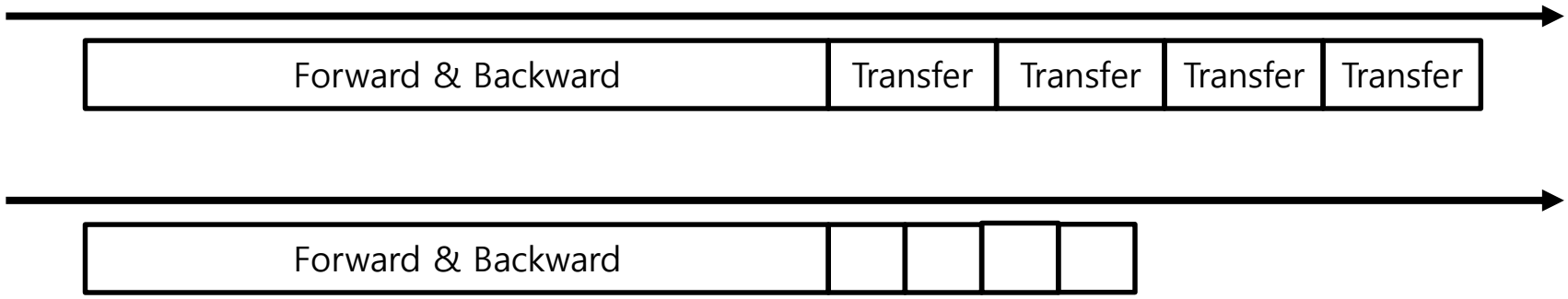
- With **small batches**, multi-GPU is **worse** than a single GPU
- Even with large batches, 4GPUs' speedup is only around **1.5x**

Communication-Compute Overlapping

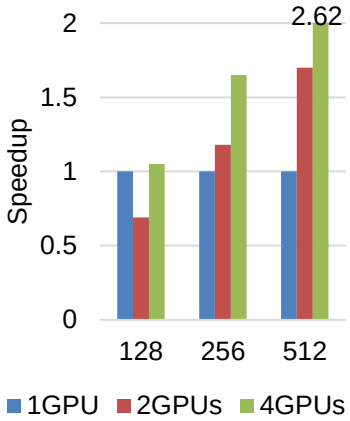


- Transfer overhead is not negligible
- Transfer as soon as gradients of each layer become available
- **TensorFlow** is partly doing this

Reducing Amount of Data Transfer



- **Quantization** methods
  - CNTK's 1bit-SGD (1/32 transfer)
- Avoid fully connected layers
  - 90% of parameters reside in fully-connected layers
  - Use 1x1 convolution layers instead of fully-connected layers (e.g. GoogLeNet)



CNTK 1bit-SGD

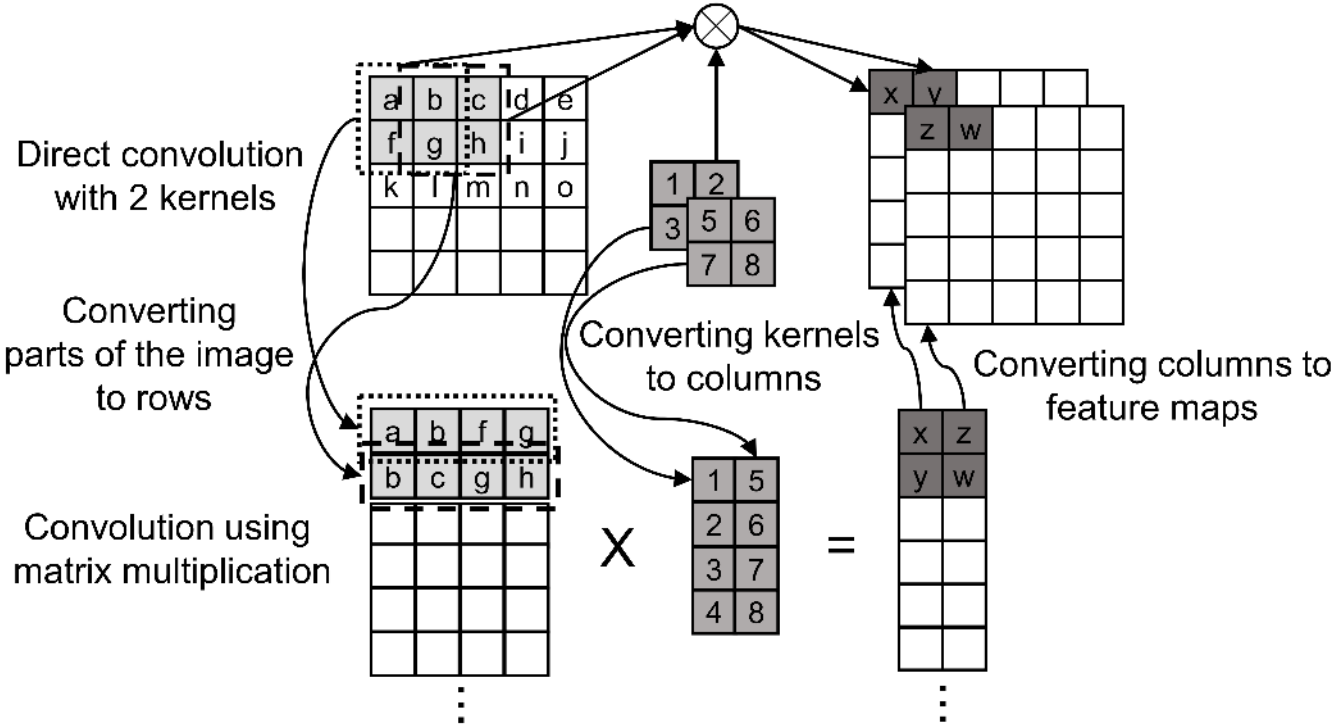
Outline

- Convolutional Neural Network
- Deep Learning Frameworks
- Framework Comparison
- Multi-GPU Comparison
- **Layer-wise Analysis of Convolution Algorithms**
- Conclusions

Direct Convolution Algorithm

- Straightforward convolution algorithm
- Not supported by cuDNN, thus we use **cuda-convnet3** for testing
- Easy to implement but hard to optimize
- cuda-convnet requires **CHWN** tensor layout instead of **NCHW**
- Computation time for forward and backward computations are **not symmetric**

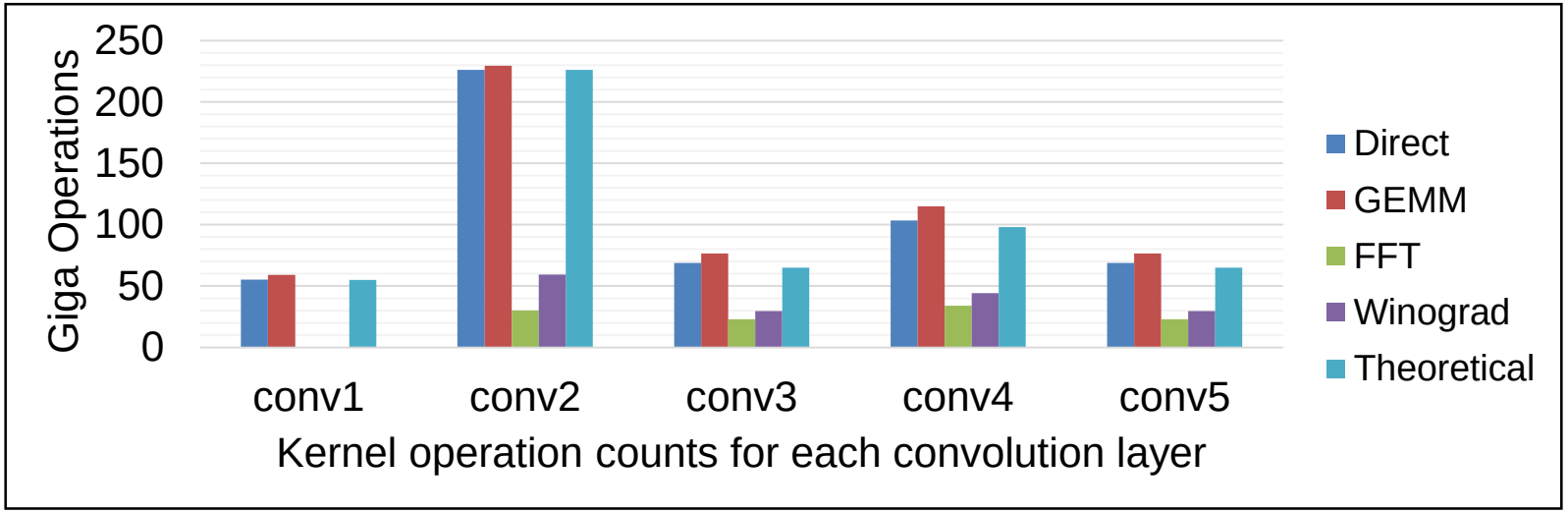
GEMM Convolution Algorithm



- Treat convolutions as **vector dot products** in matrix multiplication
- Forward and backward computations are **symmetric**
- Efficiently optimized, but tiling inserts unnecessary computations

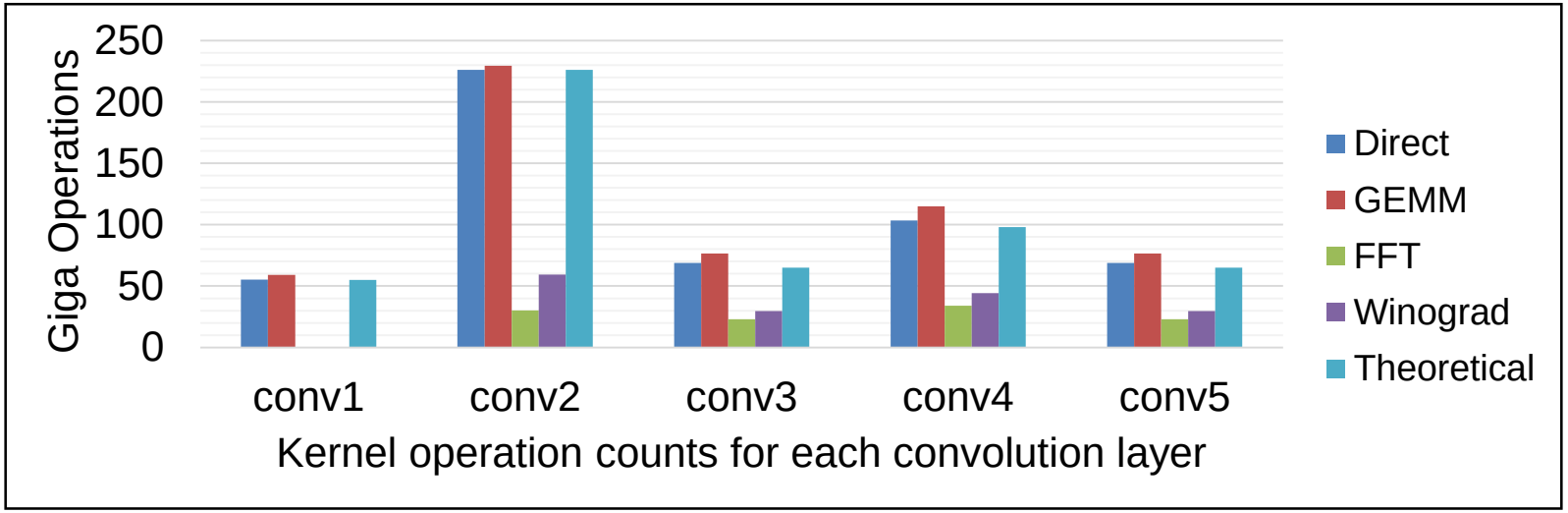
FFT Convolution Algorithm

- FFT → CGEMM → inverse FFT == Convolution
- In 2D convolution, computational complexity reduces from $O(HWRS)$ to $O(HW\log(HW))$
- Computational cost **does not depend on kernel dimension**
- cuDNN FFT convolution does not support **strides**

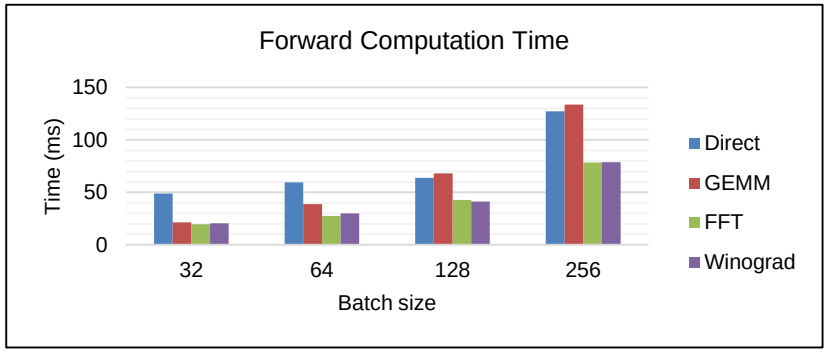


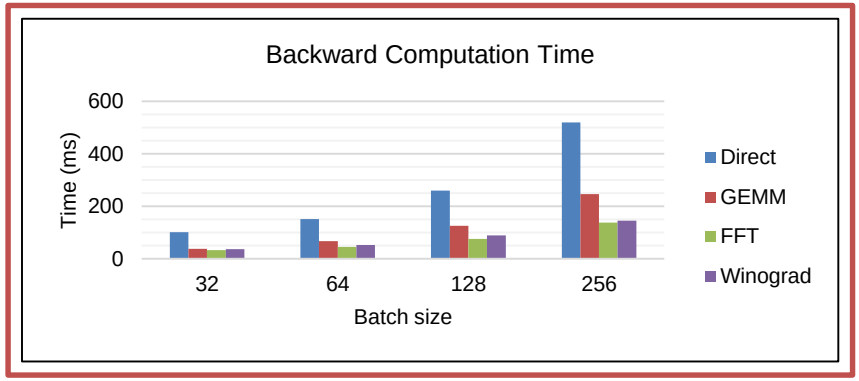
Winograd Convolution Algorithm

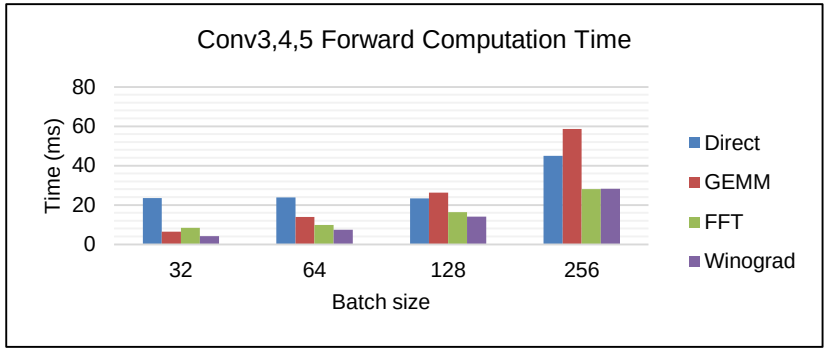
- Based on GEMM convolution method
- **Minimal filtering algorithm** for 3x3 kernel and 4x4 tiling reduces 144 multiplications into 36 (4x difference).
- Each kernel dimension requires own minimal filtering algorithm.
- cuDNN 5.1 supports Winograd algorithm for **3x3** and **5x5** convolutions with no strides

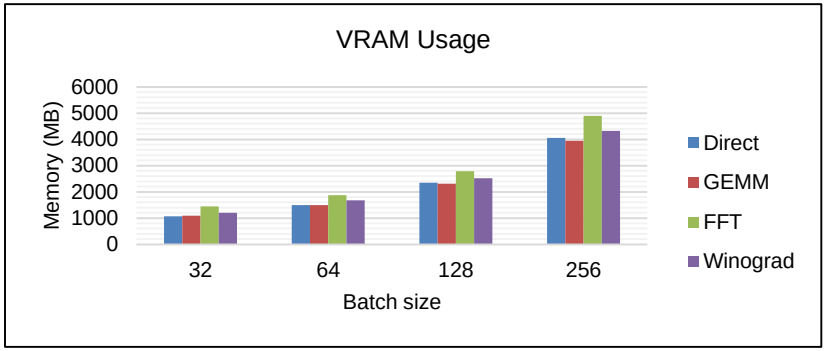


Computation Time Comparison



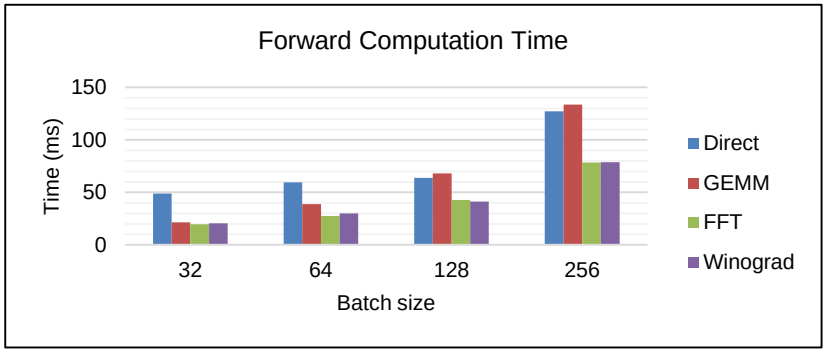


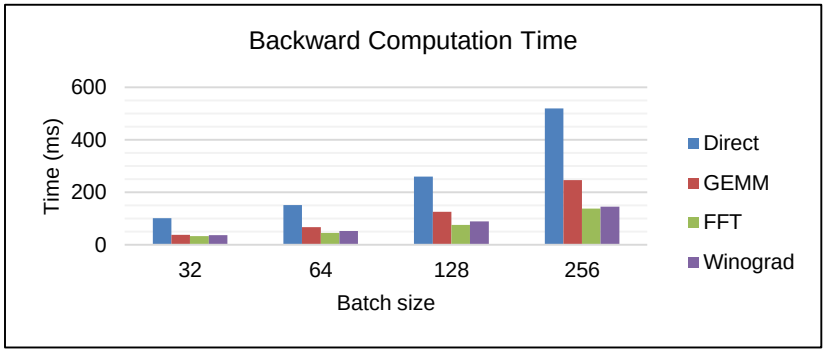


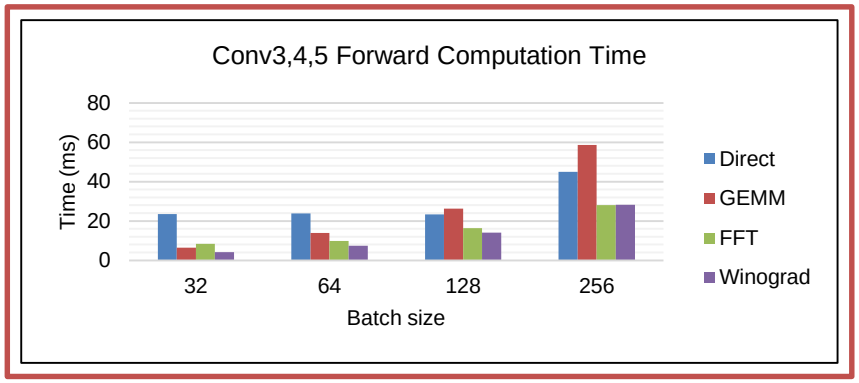


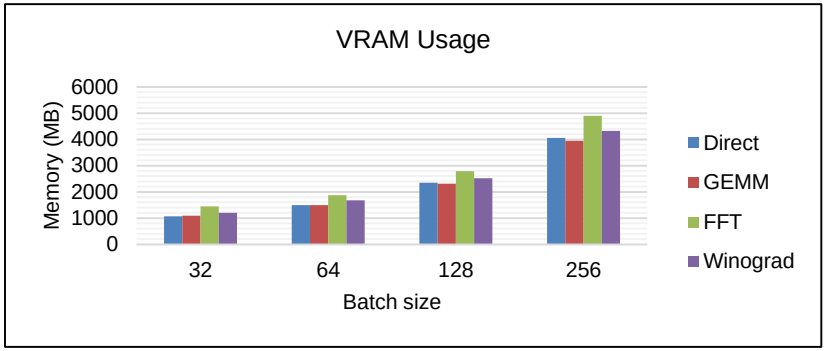
- **Direct** algorithm shows poor performance on **backward computations**
- **FFT** is the fastest algorithm for most of the time

Computation Time Comparison



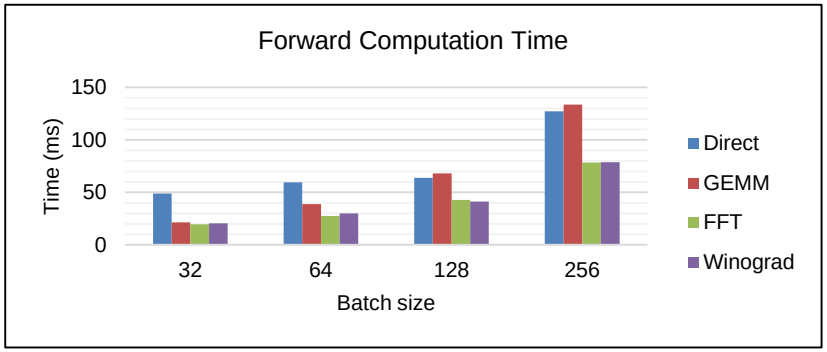


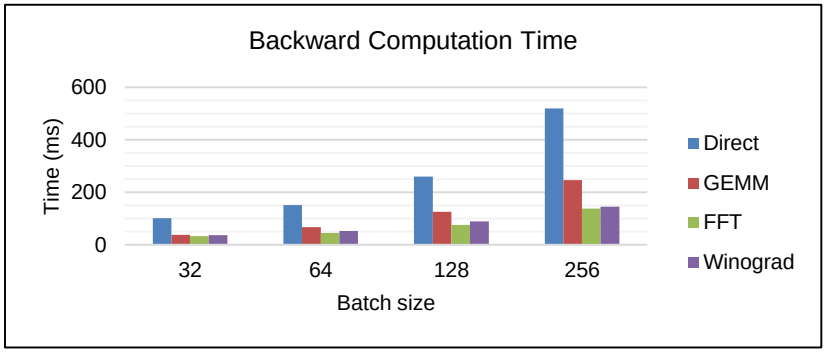


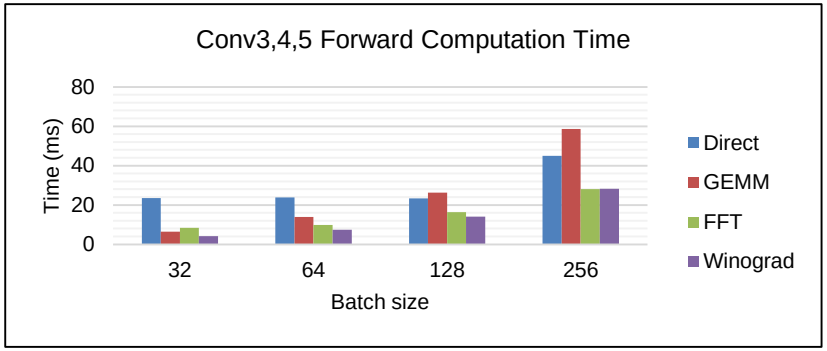


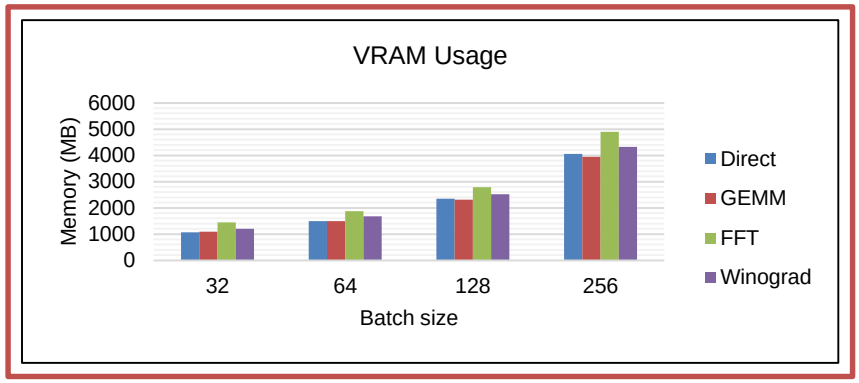
- **Direct** algorithm shows poor performance on **backward computations**
- **FFT** is the fastest algorithm for most of the time
- **Winograd** performs better in **smaller batches** and **3x3 convolutions**

Computation Time Comparison



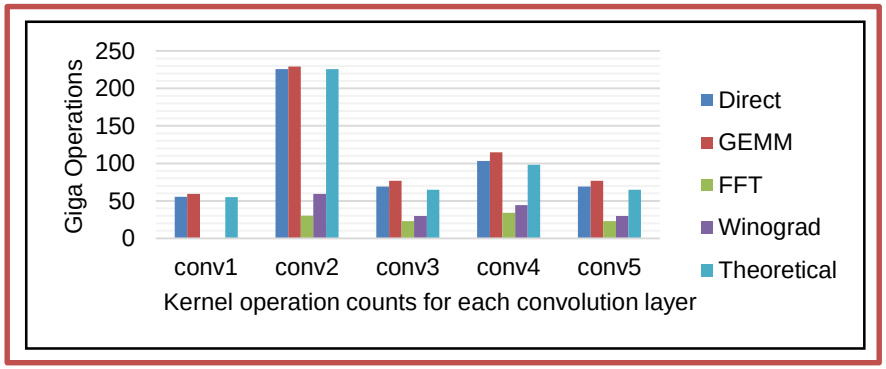


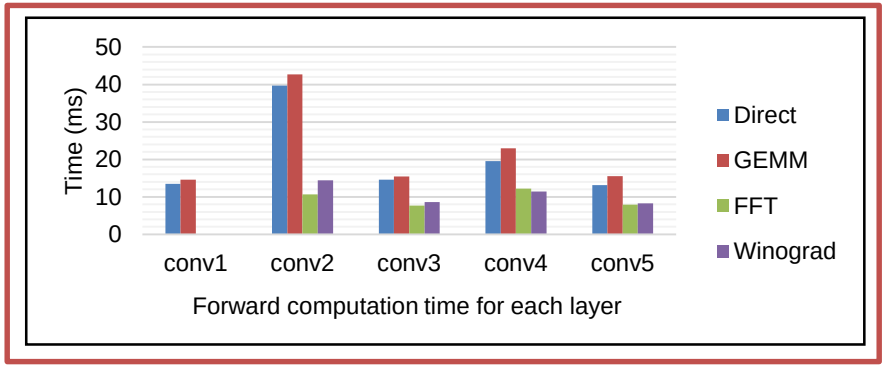


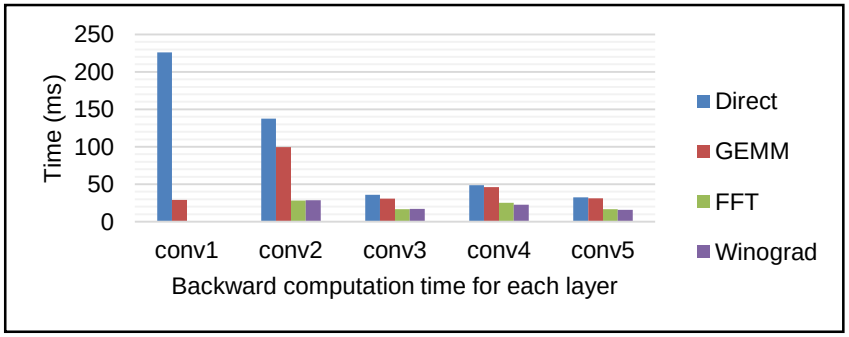


- **Direct** algorithm shows poor performance on **backward computation**
- **FFT** is the fastest algorithm for most of the time
- **Winograd** performs better in **smaller batches** and **3x3 convolutions**
- Memory usage differences are not significant

Layer-wise Analysis of Convolution Layers

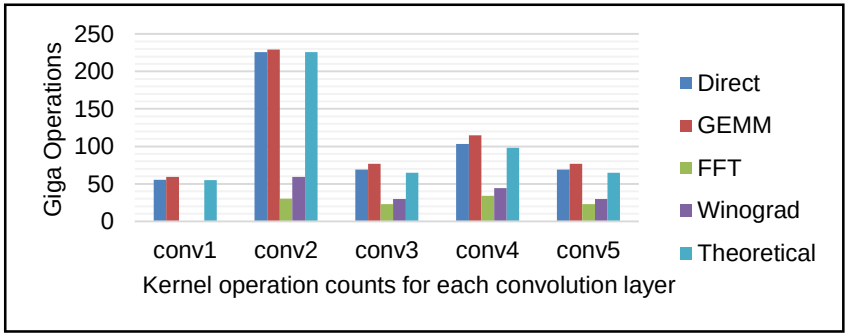


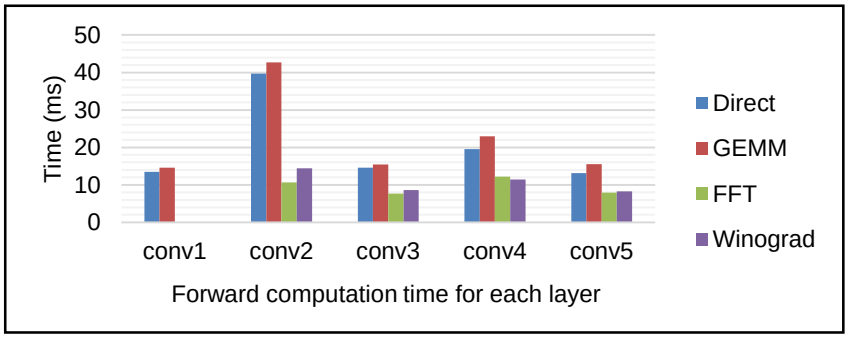


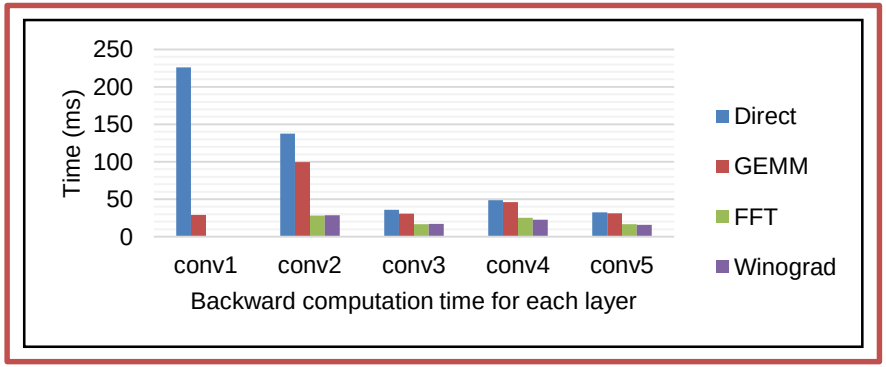


- **Operation count** is the primary factor for the execution time
- Conv2 layer requires the most computations
- Thus, **FFT** and **Winograd** are faster than **Direct** or **GEMM**

Layer-wise Analysis of Convolution Layers







- **Operation count** is the primary factor for execution time
- Conv2 layer requires the most computations
- Thus, **FFT** and **Winograd** are faster than **Direct** or **GEMM**
- Direct convolution is slow because its backward computation in the first layer is inefficient

Conclusions

- **Convolution layers** take up most of the computation time while training CNN models
- Performance difference of the frameworks are mainly due to **convolution algorithms**
- Choosing **optimal options** can **double the training speed** of the AlexNet model
- **Tensor layout** and **unnecessary backpropagation** might result in minor performance differences

Conclusions

- **FFT convolution** algorithm is the fastest in most of the time because of its **reduced computation complexity**
- **Winograd convolution** can be faster than FFT in **3x3** convolution layers with **small batch sizes**
- Data parallelism is **inefficient** in most frameworks because of the **communication cost**, but some techniques might improve the multi-GPU scalability