ClickTrain: Efficient and Accurate End-to-End Deep Learning Training via Fine-Grained Architecture-Preserving Pruning

Chengming Zhang	Washington State University
Geng Yuan	Northeastern University
Wei Niu	College of William and Mary
Jiannan Tian	Washington State University
Sian Jin	Washington State University
Donglin Zhuang	University of Sydney
Zhe Jiang	University of Alabama
Yanzhi Wang	Northeastern University
Bin Ren	College of William and Mary
Shuaiwen Leon Song	University of Sydney
Dingwen Tao	Washington State University

June 14, 2021 ICS'21 Worldwide online event

Outline

- Introduction
  - prune neural networks during training
  - Fined-Grained Pattern-Based Pruning
  - Contribution
- Background
  - Patterns
  - Impact of patterns
- Designs
  - Modeling framework
  - Algorithm-level design
  - System-level design
- Experimental Evaluation
  - Model Accuracy and Ratio Evaluation
  - Single-GPU Performance Evaluation
  - Multi-GPU Performance Evaluation

Outline

- Introduction
  - prune neural networks during training
  - Fined-Grained Pattern-Based Pruning
  - Contribution
- Background
  - Patterns
  - Impact of patterns
- Designs
  - Modeling framework
  - Algorithm-level design
  - System-level design
- Experimental Evaluation
  - Model Accuracy and Ratio Evaluation
  - Single-GPU Performance Evaluation
  - Multi-GPU Performance Evaluation

1 Introduction

➢ **What is neural network pruning?**

- Pruning is to reduce the number of DNN weights.
- Pruning reduces the computation complexity.

➢ **Why prune neural networks during training?**

- Ever-increasing scale and complexity of the networks with large-scale training datasets, leading to challenges to the cost of DNN training.
- Backward phase can consume more than 70% of the overall training FLOPs.

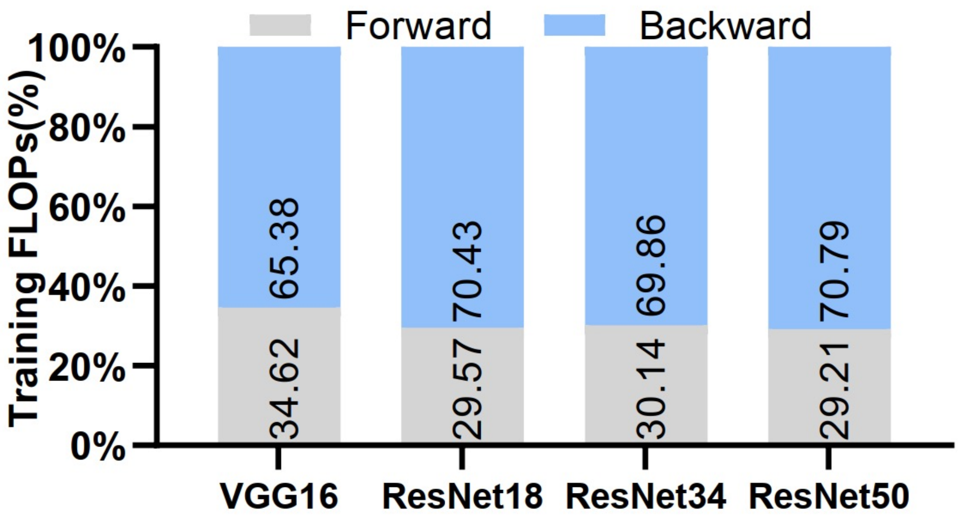

Figure 1: Percentage of FLOPs in forward and backward.

1 Introduction

➢ Typical Training Procedure

- Training a model to high accuracy.
- Pruning the well-trained model.
- Fine-tuning the pruned model.

➢ Non-Structured Pruning and Structured Pruning

- The non-structured pruning: heuristically prune the redundant weights on arbitrary locations.
- Structured Pruning: prune the entire filters, channels to maintain the structural regularity.

➢ Fined-Grained Pattern-Based Pruning

- As shown in Fig 2, fined-grained pattern-based Pruning: intermediate sparsity type between non-structured pruning and structured pruning.

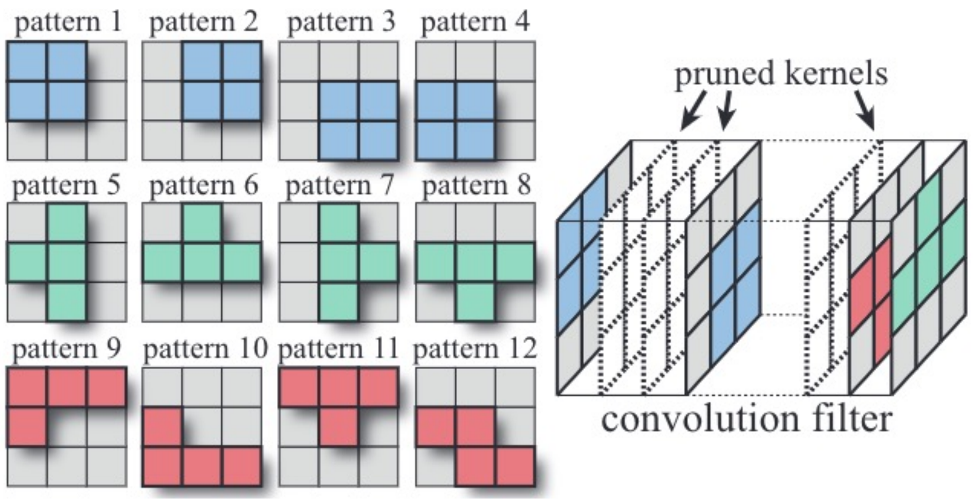

Fig 2. Fined grained pattern-based pruning. Gray parts are pruned.

1 Introduction

➢ **What we did?**

- Incorporate a weight importance estimation approach to select the desired patterns from a generated candidate pattern pool.
- Propose methods to gradually generate the candidate patterns.
- Propose a modified group-lasso regularization.
- Propose multiple system-level optimizations including fast sparse matrix format conversion, pattern-accelerated sparse convolution, pattern-based communication optimization, and compiler-assisted optimized code generation.
- Use pruning during training (PDT)-based method to significantly reduce the end-to-end time.
- Maintain the network architecture for high accuracy.

Outline

- Introduction
  - prune neural networks during training
  - Fined-Grained Pattern-Based Pruning
  - Contribution
- Background
  - Patterns
  - Impact of patterns
- Designs
  - Modeling framework
  - Algorithm-level design
  - System-level design
- Experimental Evaluation
  - Model Accuracy and Ratio Evaluation
  - Single-GPU Performance Evaluation
  - Multi-GPU Performance Evaluation

2 Background

➢ What is pattern?

- As shown in Figure 3, weights with higher absolute values form some specific shapes (named pattern).
- Repeatedly appears in the model.

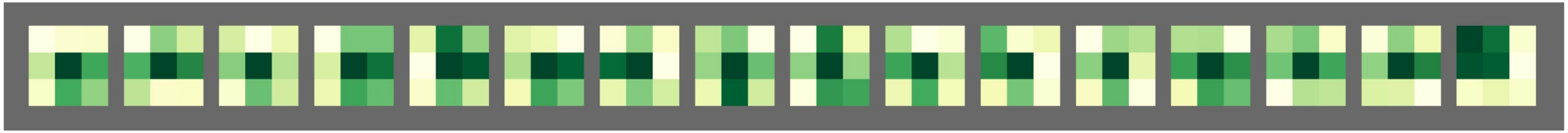

Fig 3. Heat map of convolutional layer of a VGG-16 [1].

➢ What is the impact of pattern on performance?

- Transform patterns to Gaussian filter

$$\underbrace{\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdots \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \cdots \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \cdots \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}}_{n \text{ interpolations}} = \begin{bmatrix} p & 2p & p \\ 2p & 4p & 2p \\ p & 2p & p \end{bmatrix}^n = \left[p \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \right]^n$$

Fig 4. Gaussian filter [1].

2 Background

- Transform patterns to Laplacian of Gaussian filter

$$\underbrace{\begin{bmatrix}0 & 1 & 0\\ 1 & 1 & 1\\ 0 & 0 & 0\end{bmatrix} \cdots \begin{bmatrix}0 & 1 & 0\\ 1 & 1 & 0\\ 0 & 1 & 0\end{bmatrix} \cdots \begin{bmatrix}0 & 0 & 0\\ 1 & 1 & 1\\ 0 & 1 & 0\end{bmatrix} \cdots \begin{bmatrix}0 & 1 & 0\\ 0 & 1 & 1\\ 0 & 1 & 0\end{bmatrix}}_{n \text{ interpolations}} = \begin{bmatrix}0 & p & 0\\ p & 1 & p\\ 0 & p & 0\end{bmatrix}^n = \left[p \begin{bmatrix}0 & 1 & 0\\ 1 & 1/p & 1\\ 0 & 1 & 0\end{bmatrix} \right]^n$$

Fig 5. Laplacian of Gaussian filter[1].

[1] Xiaolong Ma, et al. 2020. An Image Enhancing Pattern-based Sparsity for Real-time Inference on Mobile Devices. arXiv preprint arXiv:2001.07710 (2020).

Outline

- Introduction
  - prune neural networks during training
  - Fined-Grained Pattern-Based Pruning
  - Contribution
- Background
  - Patterns
  - Impact of patterns
- Designs
  - Modeling framework
  - Algorithm-level design
  - System-level design
- Experimental Evaluation
  - Model Accuracy and Ratio Evaluation
  - Single-GPU Performance Evaluation
  - Multi-GPU Performance Evaluation

3 Framework

- As shown is fig 6. Stage 1, 2, 3 and 4 are algorithm-level design, which focus on high compress ratio and high accuracy.
- Stage 5 is system-level supports, which focus on improve computation efficiency.

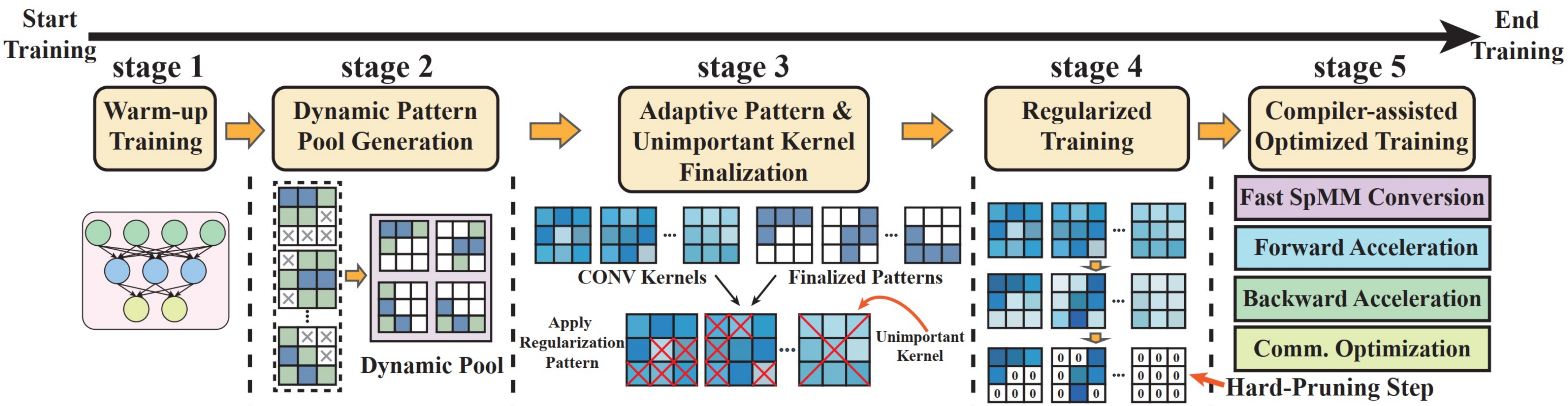

Fig 6. Overview of framework.

- Introduced Hyper-parameters:

Basic training hyperparameters (learning rate, etc.), compression ratio, regularized training epochs, regularization penalty coefficient, pre-training/warm-up epochs, and hard pruning epochs.

4 Algorithm-level design

➢ **Stage 1: Train network for N epochs.**

- N is hyper-parameter.

➢ **Stage 2: Dynamic pattern pool generate.**

- Generic pattern pool and dynamic pattern pool.
- First select one weight position.
- Select the second weight position.
- Create a candidate pattern pool.
- Calculate important score for each pattern and finalize patterns.

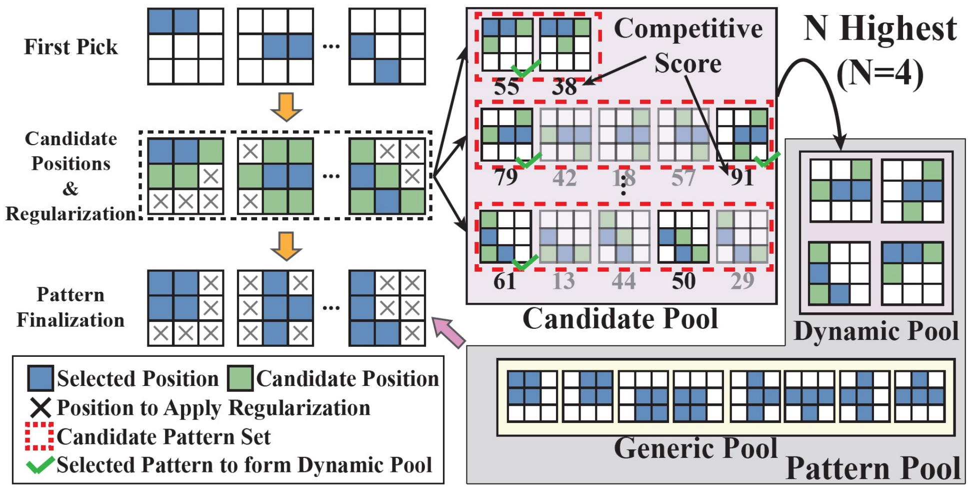

Fig 7. Generate pattern pool.

➢ **Stage 3: Adaptive choose pattern for each kernel.**

- Calculate important score for each pattern using importance formula.

$$t_{:,:} = G^{(\ell)}_{f_\ell,c_\ell,:,:} \odot W^{(\ell)}_{f_\ell,c_\ell,:,:} \odot p_i, \quad I_{p_i} = \sum_{h_\ell}^{H_\ell} \sum_{s_\ell}^{S_\ell} (t_{h_\ell \times s_\ell})^2,$$

➢ **Stage 4: Penalize unimportant weights using modified group lasso.**

$$Z^{(\ell)} = W^{(\ell)} \odot \left(\neg P^{(\ell)}\right), \quad U^{(\ell)} = W^{(\ell)} \odot \left(\neg I^{(\ell)}\right)$$

$$E(W,D) = E(W,D) + \lambda_P \sum_{l=1}^{L} \left(\sum_{f_\ell=1}^{F_\ell} \sum_{k_\ell=1}^{K_\ell} \left\| Z^{(\ell)}_{f_\ell,k_\ell,:,:} \right\|_g \right)$$

5 System-level design

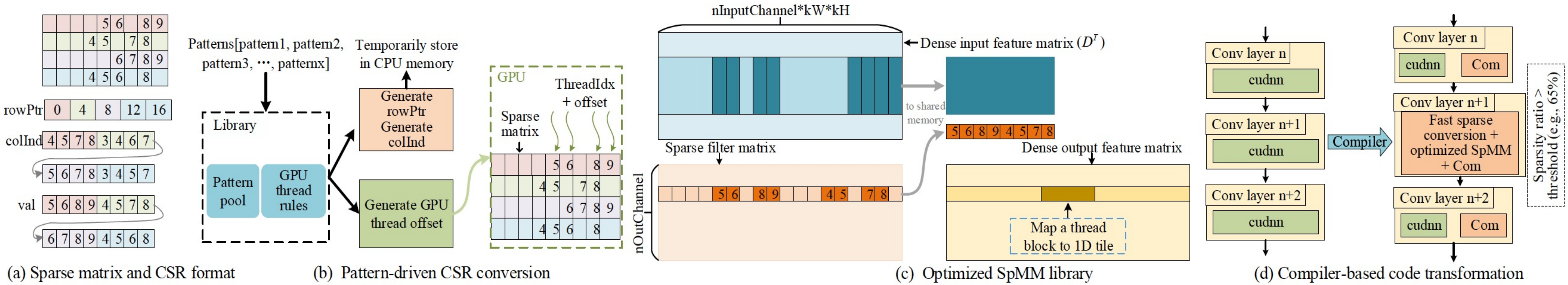

Fig 8. system level design: compiler-assisted pattern-accelerated sparse matrix-matrix multiplication for sparse convolution.

- Modern GPUs are more suitable for Matrix-Matrix multiplication.
- SpMM requires first converting dense input matrix to a sparse format such as Compressed Sparse Row (CSR).
- Pattern sparsity facilitate the fast conversion.
- Limit all the filters in the same layer to have the same number of un-pruned (non-zero) weights.
- 1D tiling strategy and map each thread block to a 1D row tile of the output matrix.

Outline

- Introduction
  - prune neural networks during training
  - Fined-Grained Pattern-Based Pruning
  - Contribution
- Background
  - Patterns
  - Impact of patterns
- Designs
  - Modeling framework
  - Algorithm-level design
  - System-level design
- Experimental Evaluation
  - Model Accuracy and Ratio Evaluation
  - Single-GPU Performance Evaluation
  - Multi-GPU Performance Evaluation

6 Results

- **Setup and Dataset**
  - Neural networks:
    ResNet18/32/50/101 and VGG11/13/16
  - Dataset:
    CIFAR10/100 [6] and ImageNet-2012
  - Experiment platform
    Pytorch
    Frontera supercomputer at TACC
    CUDA 10.1 and its default profiler

6 Results

		PDT Method	Base. Acc.	Valid. Acc. Δ	Comp. Ratio	Train./Inf. FLOPs	Hard Pr. Epoch
CIFAR10	ResNet32	PRT	93.6%	−2.1%	2.2×	53% / 66%	N/A
		CLK	93.6%	0±0.05%	8.6×	41.3% / 85.1%	98
		CLK	93.6%	**0±0.07%**	**10.7×**	**43.0% / 85.7%**	95
	ResNet50	PRT	94.2%	−1.1%	2.3×	50% / 70%	N/A
		CLK	94.1%	0±0.04%	8.5×	37.5% / 74.3%	95
		CLK	94.1%	**−0.2±0.05%**	**10.8×**	**41.2% / 77.6%**	90
	VGG11	PRT	92.1%	−0.7%	8.1×	57% / 65%	N/A
		CLK	92.1%	−0.1±0.04%	8.7×	41.2% / 81.5%	96
		CLK	92.1%	**−0.3±0.06%**	**11.5×**	**43.9% / 85.3%**	94
	VGG13	PRT	93.9%	−0.6%	8.0×	56% / 63%	N/A
		CLK	93.8%	0±0.08%	8.6×	41.3% / 81.3%	95
		CLK	93.8%	**−0.2±0.04%**	**10.9×**	**42.5% / 84.9%**	96
CIFAR100	ResNet32	PRT	71.0%	−1.4%	2.1×	32% / 46%	N/A
		CLK	71.0%	0±0.05%	8.3×	41.7% / 82.9%	95
		CLK	71.0%	**−0.2±0.05%**	**10.4×**	**45.2% / 85.6%**	90
	ResNet50	PRT	73.1%	−0.7%	1.9×	53% / 69%	N/A
		CLK	73.1%	0±0.04%	8.2×	36.7% / 73.6%	96
		CLK	73.1%	**−0.2±0.07%**	**9.7×**	**38.9% / 77.3%**	95
	VGG11	PRT	70.6%	−1.3%	3.0×	47% / 57%	N/A
		CLK	70.6%	0±0.1%	6.7×	40.1% / 78.6%	95
		CLK	70.6%	**−0.2±0.06%**	**8.4×**	**43.1% / 82.0%**	92
	VGG13	PRT	74.1%	−1.4%	2.9×	42% / 52%	N/A
		CLK	74.1%	−0.1±0.05%	7.4×	40.5% / 79.7%	95
		CLK	74.1%	**−0.2±0.08%**	**9.2×**	**41.7% / 83.3%**	96
ImageNet	ResNet50	PRT	76.2%	−1.9%	1.6×	40% / 53%	N/A
		CLK	76.2%	**−0.6±0.07%**	**4.3×**	**36.9% / 66%**	40

Fig 11. Comparison between ClickTrain (CLK) and PDT-based method PruneTrain (PRT). FLOPs are the saved FLOPs.

	PAT Method	Base. Acc.	Valid. Acc. Δ	Comp. Ratio	Total Epochs
ResNet-18	TAS [12]	70.6%	−1.5%	1.5×	120
	DCP [74]	69.6%	−5.5%	3.3×	well train + 60
	CLK	69.6%	**−0.9%**	**4.1×**	**90**
ResNet-50	GBN [65]	75.8%	−0.6%	2.2×	well train + 60
	GAL [29]	76.4%	−7.1%	2.5×	well train + 30
	CLK	76.2%	**−0.6%**	**4.3×**	**90**
ResNet-101	RSNLIA [63]	75.27%	−2.10%	1.9×	well train + tune
	CLK	76.4%	**−1.2%**	**4.2×**	**90**
VGG-16	NeST [8]	71.6%	−2.3%	6.5×	N/A
	CLK	73.1%	**−0.8%**	**6.6×**	**90**

Fig 12. Comparison between ClickTrain and PAT-based methods on ImageNet. Well-train costs about 90 epochs.

6 Results

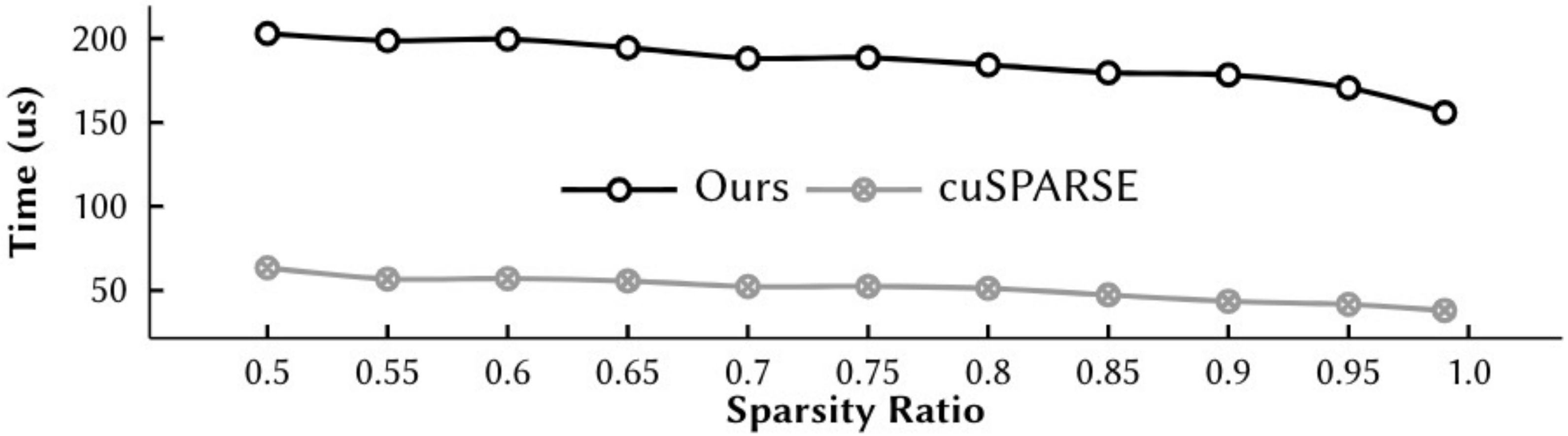

Fig 9. CSR format conversion time.

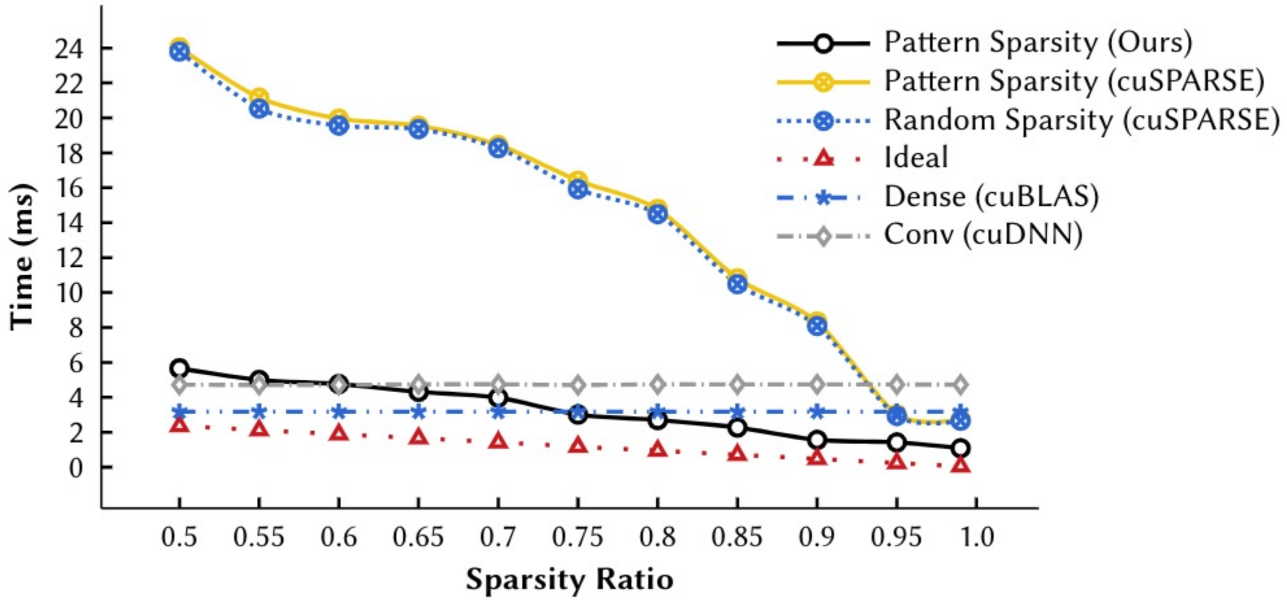

Fig 10. Convolution time with different methods.

6 Results

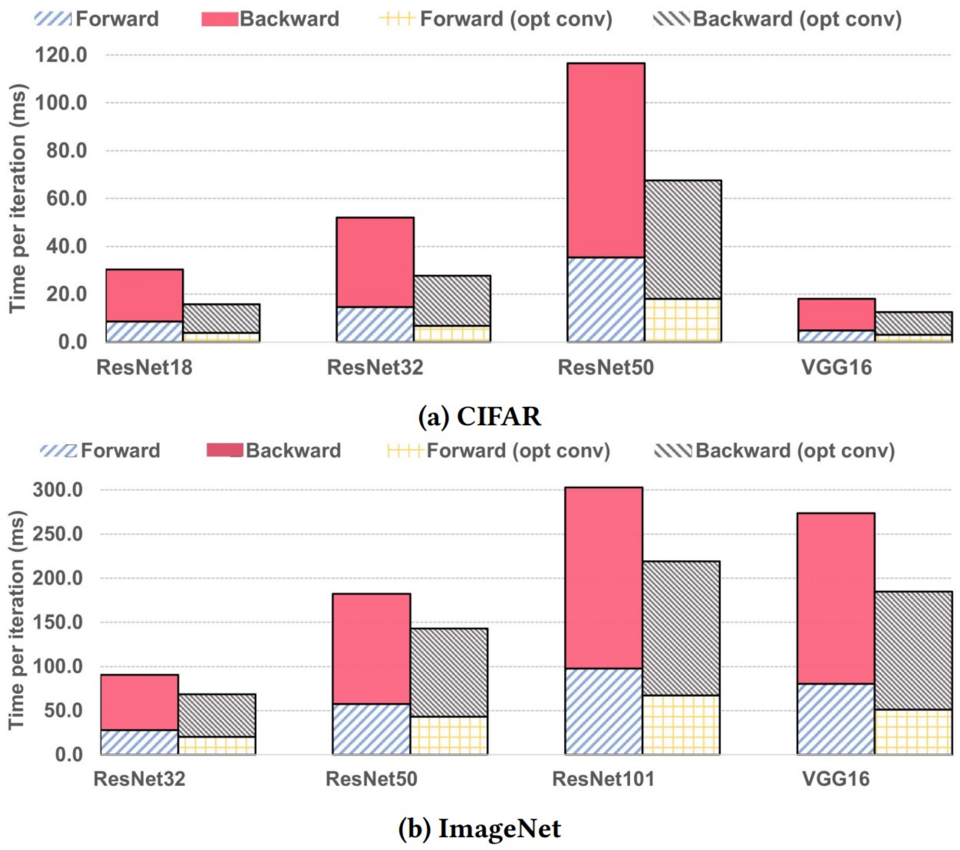

Fig 13. Average forward and backward time per iteration.

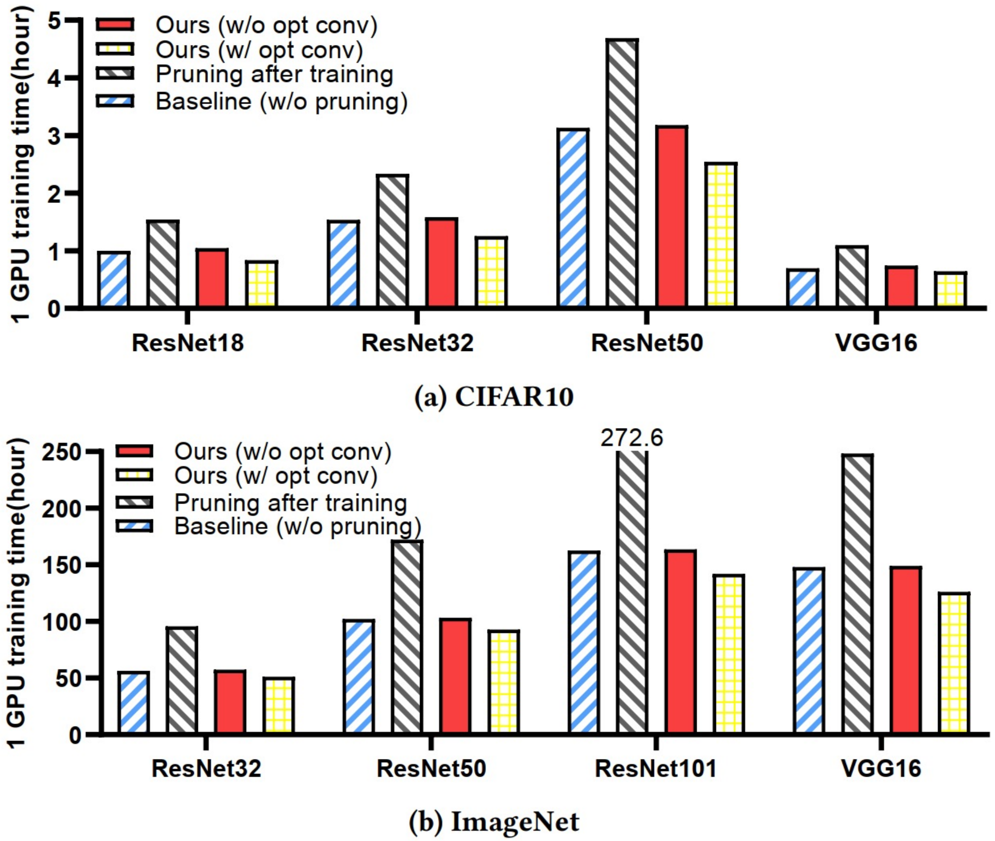

Fig 14. Total training time on CIFAR and ImageNet (single-GPU).

6 Results

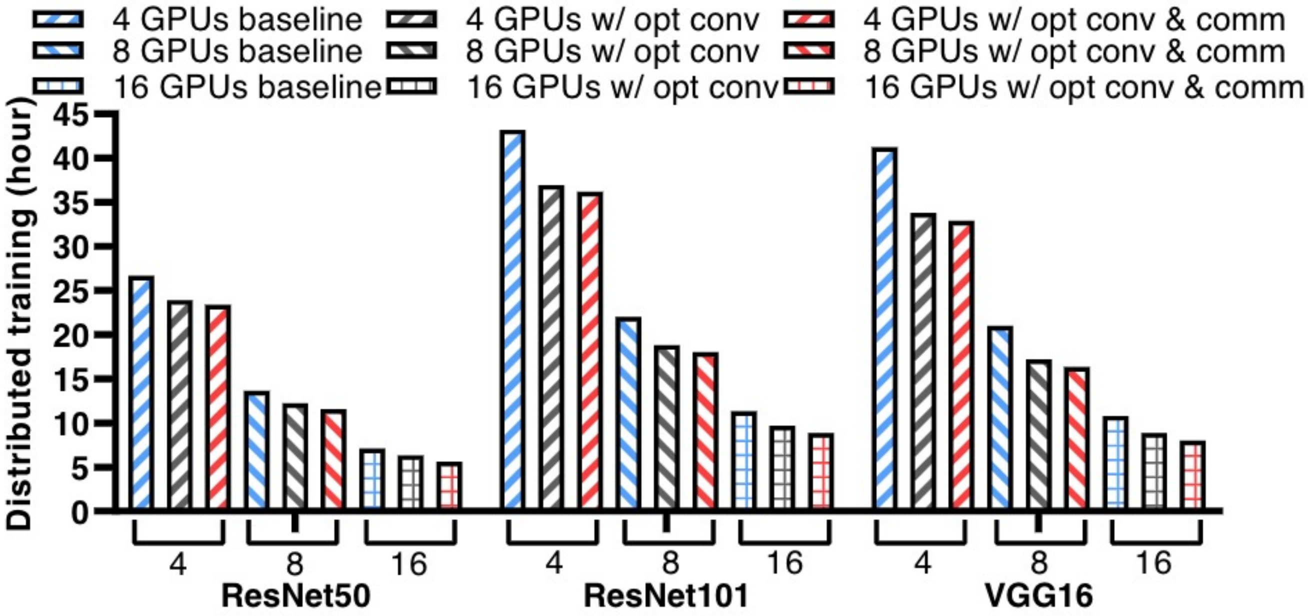

Fig 15. Total time of distributed training.

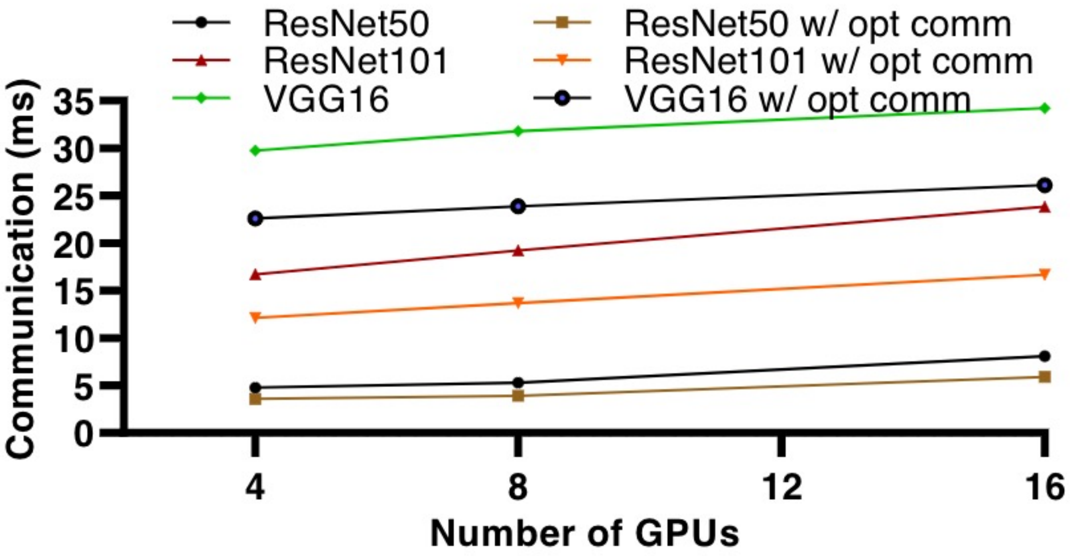

Fig 16. Total time of communication time.

6 Results

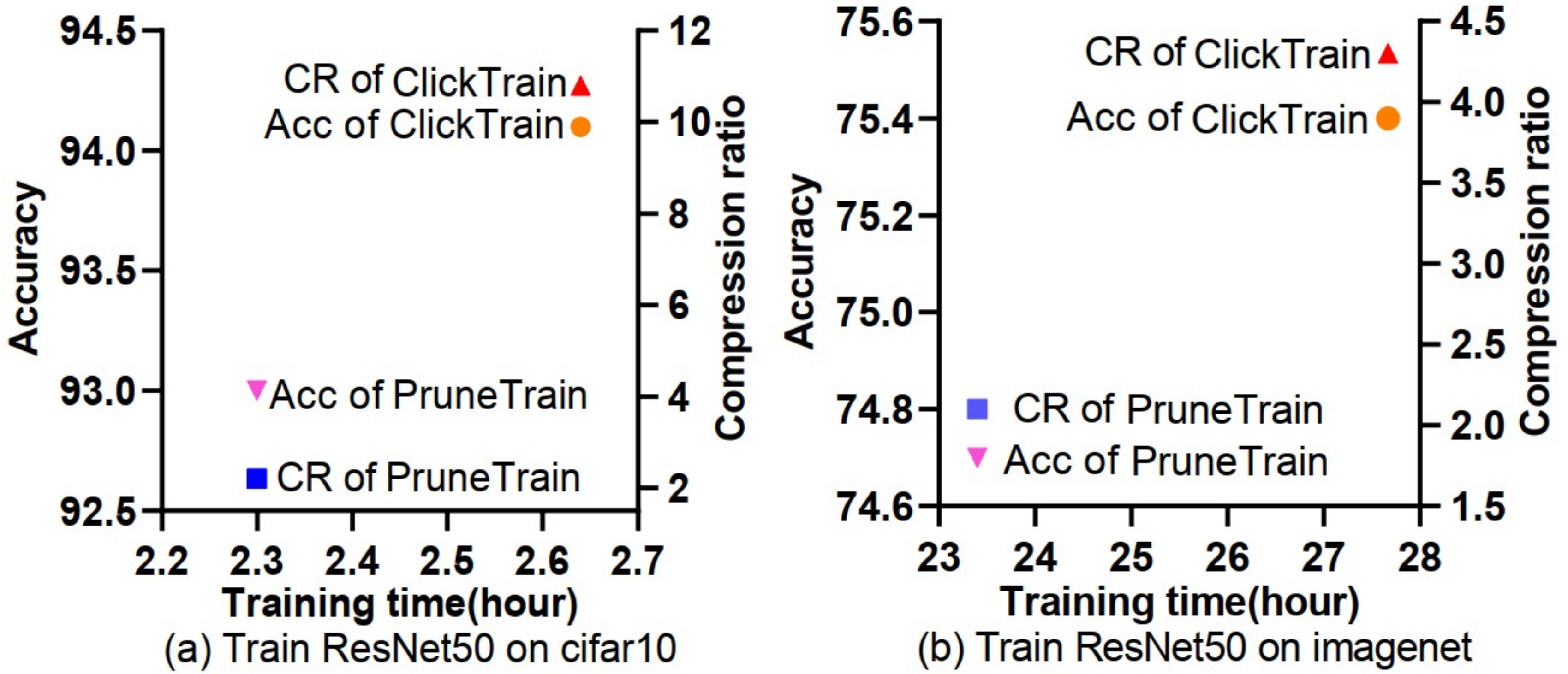

Fig 17. Comparison of PruneTrain and ClickTrain.

7 Conclusion & Future Work

➢ Conclusion

- Implement both algorithm-level and system-level optimizations with four stages.
  i) accurate weight importance estimation to select the pattern,
  ii) dynamic pattern generation and finalization,
  iii) regularized training for fine-tuning with an enhanced group-lasso,
  iv) compiler-assisted optimized training.
- Reduce the cost of PAT-based method by up to 2.3× with comparable accuracy and compression ratio.
- Improve the pruned accuracy by up to 1.8% and the compression ratio by up to 4.9× on the tested CNNs and datasets.
- We plan to extend ClickTrain to more types of DNNs in the future

Thank you!

Any questions are welcome!

Contact
Dingwen Tao: dingwen.tao@wsu.edu
Chengming Zhang: chengming.zhang@wsu.edu