ConvReLU++: Reference-based Lossless Acceleration of Conv-ReLU Operations on Mobile CPU

Mobisys'23

Rui Kong*
kongrui@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Yuanchun Li†
Institute for AI Industry Research (AIR), Tsinghua University
Beijing, China

Yizhen Yuan
Institute for AI Industry Research (AIR), Tsinghua University
Beijing, China

Linghe Kong†
Shanghai Jiao Tong University
Shanghai, China

Introduction

- Deploying CNNs to edge devices is common
- The execution of typical CNNs requires **a lot of computing power and energy**
- Various CNN inference acceleration approaches
    - model compression
    - domain specific processors
    - system optimization

Introduction

- Deploying CNNs to edge devices is common
- The execution of typical CNNs requires a lot of computing power and energy
- Various CNN inference acceleration approaches
    - model compression
    - domain specific processors
    - system optimization

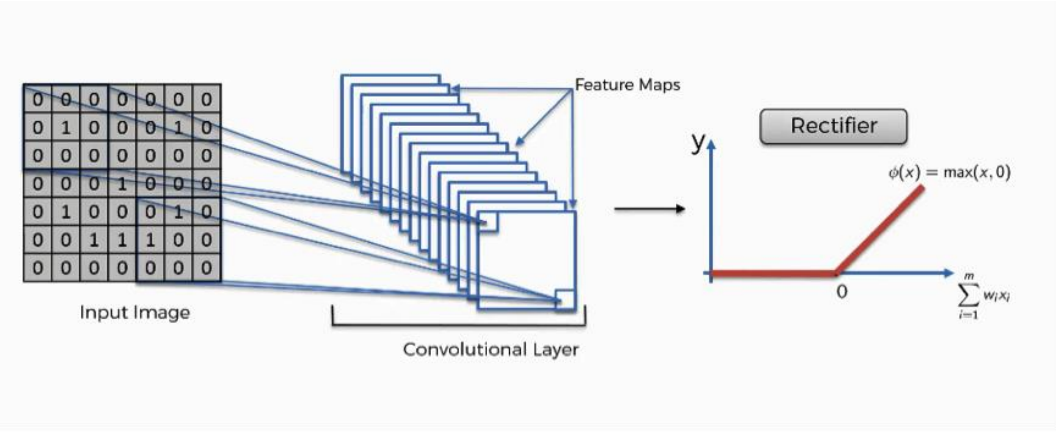
This paper's goal: Saving computation and latency from ReLU

Introduction

Saving computation and latency from ReLU

The output of Conv-ReLU may contain a large portion of **zeros**

The computation cost to obtain the precise negative values in the Convolution operation **may be wasted**



Input

-249	-91	-37
250	-134	101
27	61	-153

ReLU

0	0	0
250	0	101
27	61	0

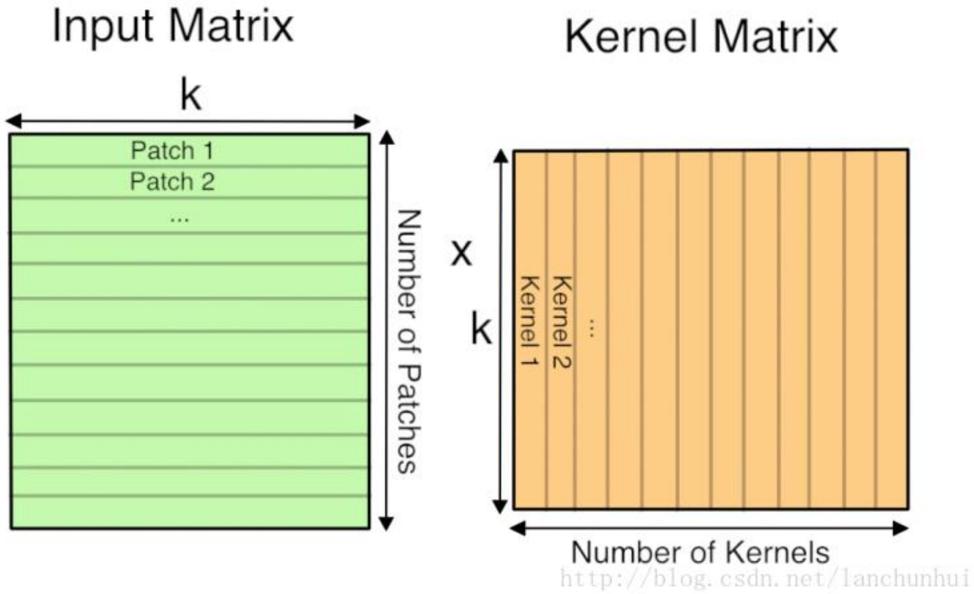
Introduction

- **Saving computation and latency from ReLU**
- **Insight: judging whether the output of a vector multiplication operation is negative can be faster than actually executing it.**
- The vector multiplication operations before a ReLU activation **can be skipped** if their output values are foreseen negative

key point : Identifying negative output operations with low overhead and high success rate

Background and Challenges

The whole computation in a Conv-ReLU structure can be seen as a batch of long-vector multiplications.



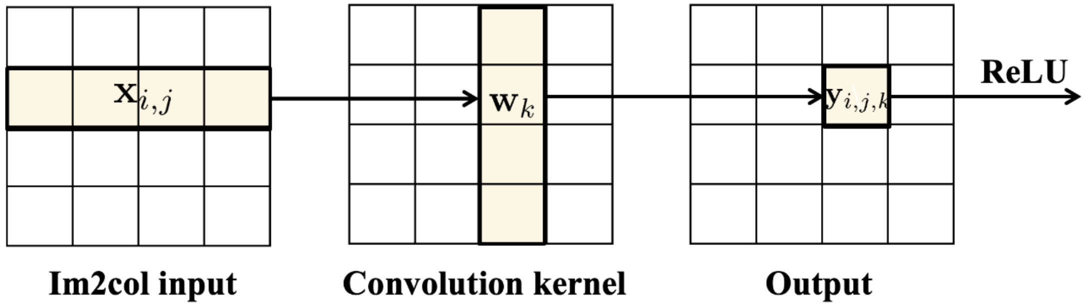


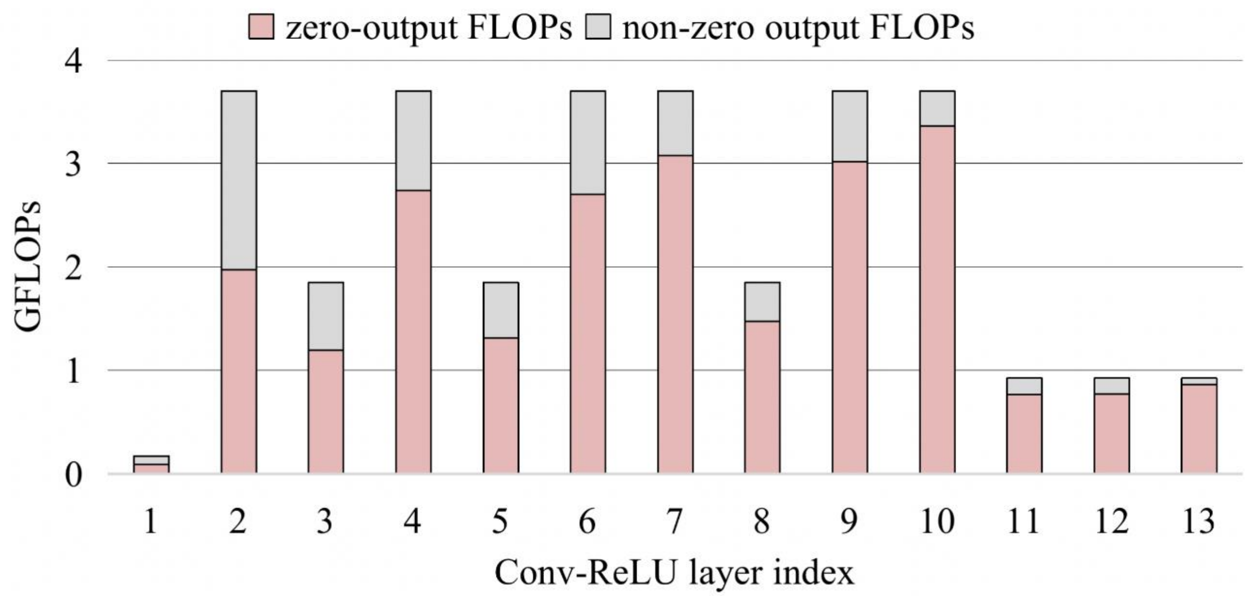
Figure 6: A convolution operation is executed as a matrix multiplication between the unfolded input matrix x and unfolded convolution kernel matrix W.

Background and Challenges

Data patterns of Conv-ReLU

1. high sparsity in the output

- The sparsity ratio and the portion of computation related to the sparse output are **high (53.29% ~ 93.43%)**
- **There is a great potential for acceleration**



Background and Challenges

Data patterns of Conv-ReLU

2. high similarity between input patches

Similarity between input patches of Conv-ReLU is common

- **(a)in the input image**
- **(b) across images in a batch**
- **(c) in the feature map**

(a) (b) (c)

Background and Challenges

Target: identify the negative-output vector multiplications in Conv-ReLU without calculating them

suppose we have a function ϕthat calculates the upper-bound

$$\overline{\mathbf{y}_{i,j,k}} = \phi(\mathbf{x}_{i,j}, \mathbf{w}_k) = upperbound(\mathbf{x}_{i,j} \cdot \mathbf{w}_k),$$

$$\mathbf{y}_{i,j,k} = \begin{cases} 0, & \text{if } \overline{\mathbf{y}_{i,j,k}} \leq 0 \\ \mathrm{ReLU}\left(\mathbf{x}_{i,j} \cdot \mathbf{w}_k\right), & \text{otherwise.} \end{cases}$$

The computation can be reduced if the upper-bound calculation function $\phi$is more lightweight than the vector multiplication and the portion of zeros is high

Background and Challenges

Challenges

1. How to select the references to reduce the computation?

2. How to effectively detect and skip unnecessary computations based on the selected references?

ConvReLU++ Design

Main idea: skip unnecessary long-vector multiplications based on the similarity between input patches

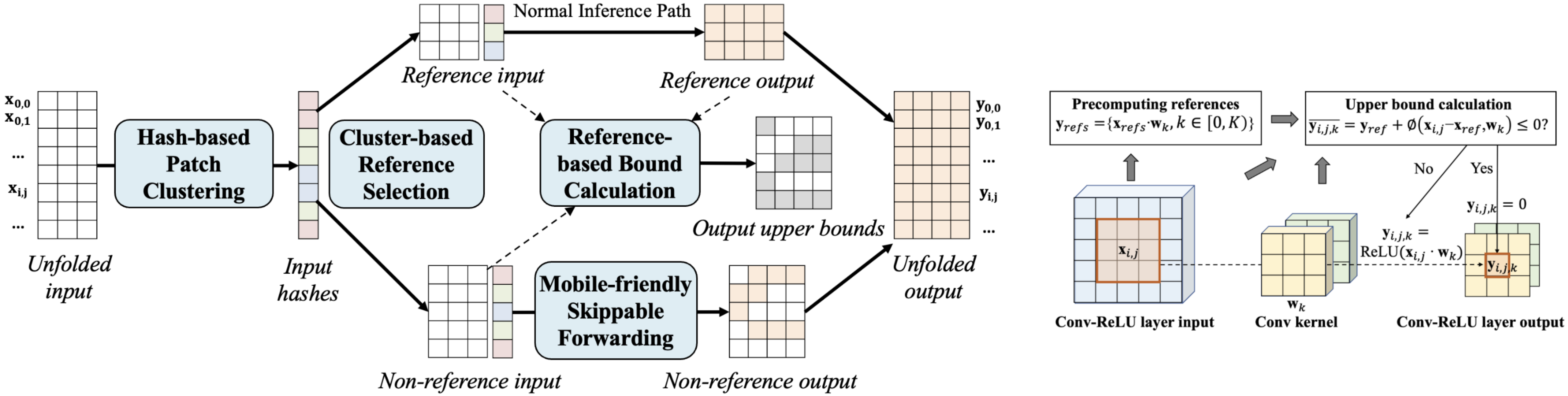


Figure 5: The overall procedure of the Conv-ReLU operation in ConvReLU++.

Hash-based patch Clustering: identify reference input patches

A tight upper-bound calculation method: identify unnecessary vector multiplications

ConvReLU++ Design

Hash-based patch Clustering: identify reference input patches

- the references should be representative of all input patches
- the selection must be efficient
- the number of selected references should be controllable

method: clustering

- k-means is too time consuming
- hashing is better
- a hash function that map the input patch to a hash id
- use a lightweight Conv layer as the hash function

$$patch\_hash(\mathbf{x}_{i,j}) = w^{hash} \cdot \mathbf{x}_{i,j},$$

select the first input patch in each cluster as the inference

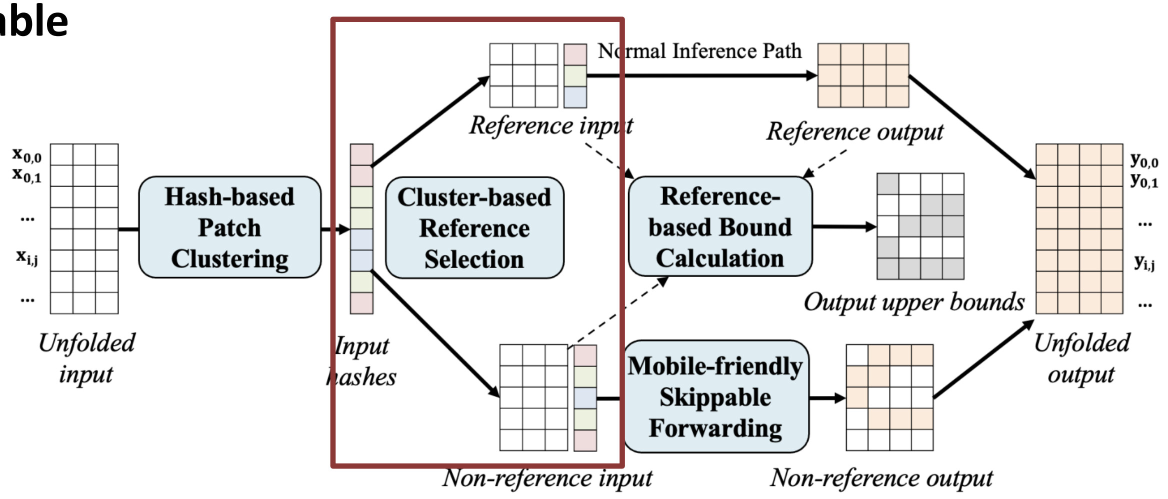


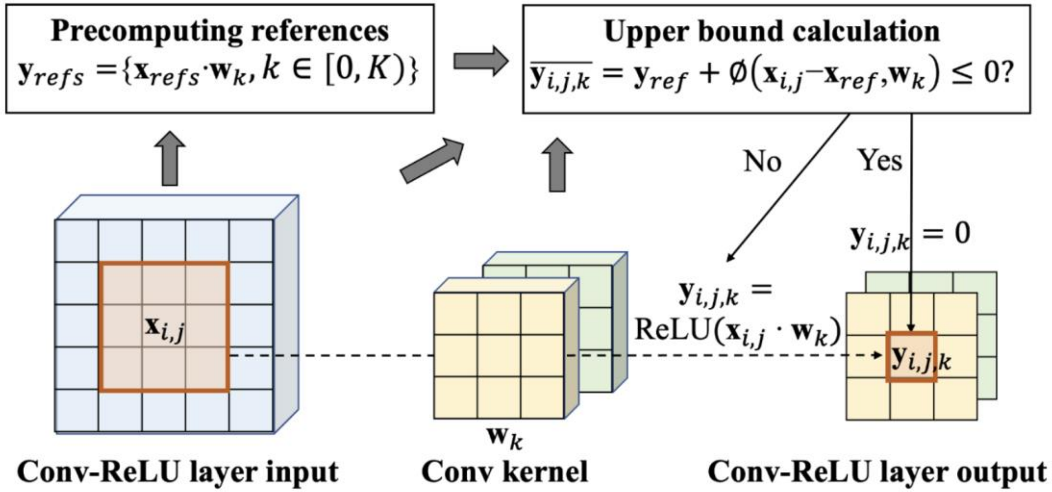
Figure 5: The overall procedure of the Conv-ReLU operation in ConvReLU++.

ConvReLU++ Design

A tight upper-bound calculation method: identify unnecessary vector multiplications

- **Goal: predict whether the dot product is negative**

$$\mathbf{y}_{i,j,k} = \mathbf{x}_{i,j} \cdot \mathbf{w}_k \le \mathbf{y}_{i,j,k}^{ref} + \phi(\mathbf{x}_{i,j} - \mathbf{x}_{i,j}^{ref}, \mathbf{w}_k),$$



- **precompute the Conv outputs for all reference input patches**
- **the key is how to design the function ϕ which is used to calculate the upper-bound**

ConvReLU++ Design

A tight upper-bound calculation method: identify unnecessary vector multiplications

$$\mathbf{y}_{i,j,k} = \mathbf{x}_{i,j} \cdot \mathbf{w}_k \le \mathbf{y}^{ref}_{i,j,k} + \phi(\mathbf{x}_{i,j} - \mathbf{x}^{ref}_{i,j}, \mathbf{w}_k),$$

$$I_{\text{same}} = \{i \mid \delta[i] \times \mathbf{w}_k[i] > 0\},$$

$$I_{\text{diff}} = \{i \mid \delta[i] \times \mathbf{w}_k[i] \le 0\}.$$

$$\begin{aligned}\delta \cdot \mathbf{w}_k \le &||\delta[I^c_{\text{diff-sub}}]|| \times ||\mathbf{w}_k[I^c_{\text{diff-sub}}]|| \\ &+ \delta[I_{\text{diff-sub}}] \cdot \mathbf{w}_k[I_{\text{diff-sub}}].\end{aligned}$$

$$\begin{aligned}\delta \cdot \mathbf{w}_k &\le ||\delta|| \times ||\mathbf{w}_k[I^c_{\text{diff-sub}}]|| + \delta[I_{\text{diff-sub}}] \cdot \mathbf{w}_k[I_{\text{diff-sub}}] \\ &= \phi(\delta, \mathbf{w}_k).\end{aligned}$$

$$\begin{aligned}\delta \cdot \mathbf{w}_k =&\delta[I_{\text{same}}] \cdot \mathbf{w}_k[I_{\text{same}}] + \delta[I_{\text{diff}}] \cdot \mathbf{w}_k[I_{\text{diff}}] \\ \le&||\delta[I_{\text{same}}]|| \times ||\mathbf{w}_k[I_{\text{same}}]|| + \delta[I_{\text{diff}}] \cdot \mathbf{w}_k[I_{\text{diff}}].\end{aligned}$$

Evaluation

Flops reduction

Table 1: Models and tasks used in our experiments and the average FLOPs reduction ratio achieved by our approach. The # Layers column is the number of Conv-ReLU layers compared with the total number of layers (excluding non-parametric layers).

ID	Task	Model	# Layers	Dataset	Original GFLOPs	Our GFLOPs
1	Classification	VanillaCNN	2/2	MNIST-ROT [25]	0.02	0.01 (-43.77%)
2	Classification	ResNet50 [18]	33/50	ImageNet [12]	8.24	8.02 (-2.62%)
3	Classification	VGG16 [32]	13/16	ImageNet [12]	15.50	14.87 (-4.08%)
4	Classification	SqueezeNet [21]	12/18	ImageNet [12]	0.35	0.33 (-5.28%)
5	Classification	ResNet50 [18]	33/50	Industrial Images [45]	24.22	21.40 (-11.81%)
6	Classification	VGG16 [32]	13/16	Industrial Images [45]	90.64	68.06 (-24.91%)
7	Detection	FasterRCNN [39]	8/17	COCO [29]	23.50	21.81 (-7.21%)
8	Detection	FasterRCNN [39]	8/17	TSRD [58]	23.50	20.64 (-12.16%)
9	Detection	MobileNet-SSD [20]	47/60	COCO [29]	1.23	1.16 (-5.44%)
10	Detection	MobileNet-SSD [20]	47/60	TSRD [58]	1.23	1.10 (-10.82%)

Evaluation

Flops reduction

Table 2: Relative FLOPs of our method & lossy baselines on MNIST-ROT dataset with Vanilla CNN model. r is a hyper-parameter of SparseNN. * means that SeerNet needs to run a whole 4-bit quantized model.

Method	Test Accuracy	Relative FLOPs
SparseNN with r=1	14.82%	17.23%
SparseNN with r=2	42.35%	35.75%
SparseNN with r=4	72.88%	68.31%
SparseNN with r=9	93.34%	144.09%
SeerNet	90.12%	27.44% + 100.00% (4-bit)*
Ours	93.34%	56.23%
Vanilla CNN	93.34%	100.00%

Evaluation

Breakdown analysis of the FLOPs reduction

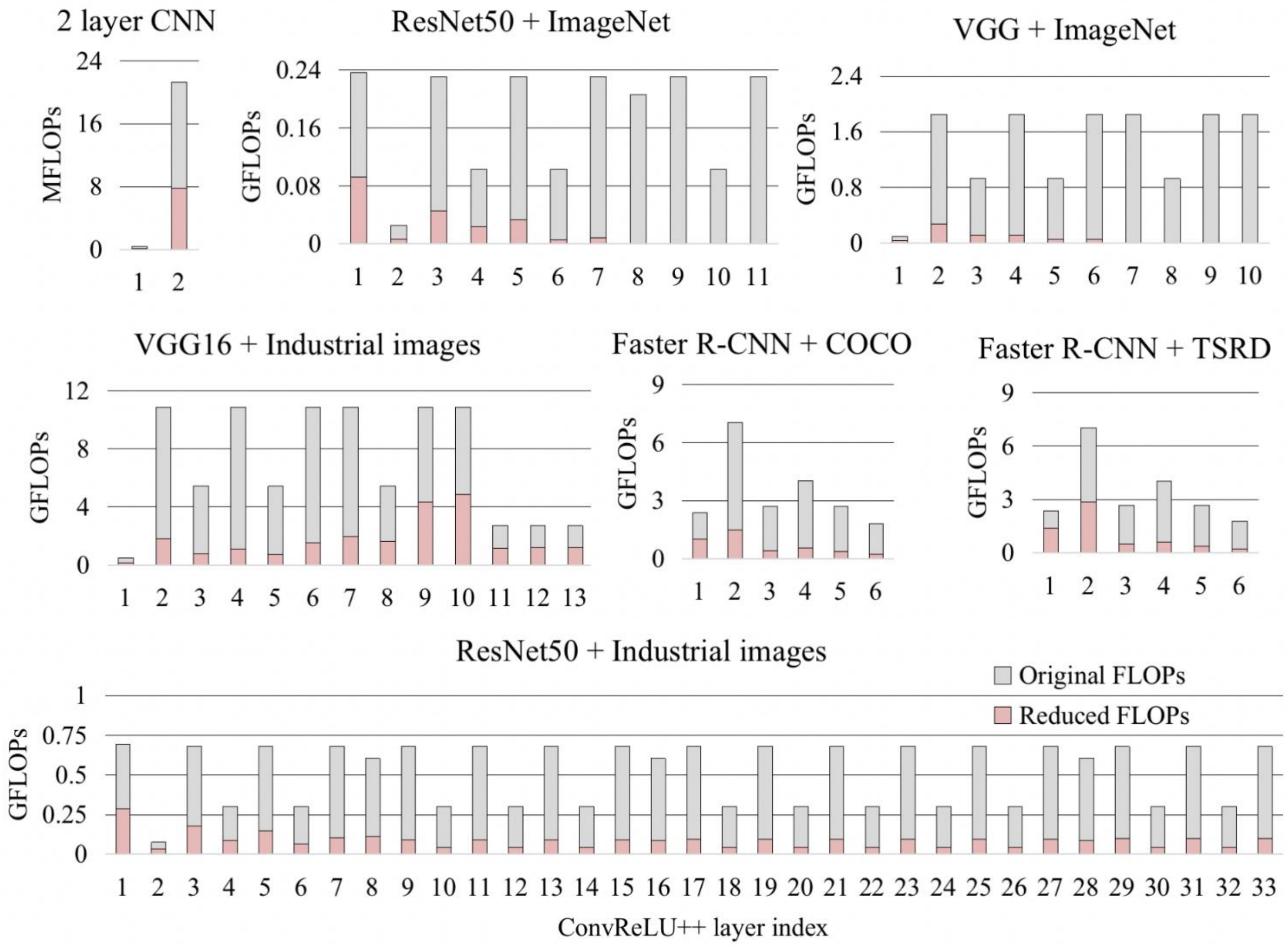


Figure 7: Layer-wise FLOPs reduction achieved by ConvReLU++ on different models and tasks.

Evaluation

Latency reduction

Table 4: Latency reduction and memory overhead of ConvReLU++ on real edge devices.

Device	Model + Dataset	Original Lat.	Our Lat.	Reduced FLOPs	Original Mem.	Our Mem.
Smartphone	ResNet50+Industrial Images	4.86s	4.47s (-8.02%)	-11.81%	148.40M	152.30M (+2.6%)
Smartphone	MobileNet-SSD+TSRD	1.08s	1.01s (-6.44%)	-10.82%	83.10M	84.20M (+1.32%)
Smartphone	SqueezeNet+ImageNet	0.24s	0.23s (-2.90%)	-5.28%	43.50M	44.30M (+1.84%)
Arduino	MobileNet+PnPLO	1.46s	1.33s (-8.91%)	-9.54%	198.23K	200.61K (+1.20%)
Arduino	MobileNet+COCO	1.46s	1.36s (-7.16%)	-8.37%	198.23K	200.61K (+1.20%)

Conclusion

- introduce an operator-level acceleration method for Conv-ReLU structures
- the method is lossless and can be applied to general vision tasks
- design a novel hash-based clustering method for input reference selection and a lightweight upper-bound calculation method for redundant vector multiplication detection

Limitations

- the proposed acceleration method has different effects on different data
- the acceleration effect is more evident for shallower convolution layers
- the method also does not support other activation functions like sigmoid

Thank You

Presented by Ye Wan

2023-06-08