香港大學
The University of Hong Kong

Deformable Butterfly: A Highly Structured and Sparse Linear Transform

Rui Lin[1,*] **Jie Ran**[1,*] Kung Hung Chiu[2] Graziano Chesi[1] Ngai Wong[1,*]

[1] Dept. of Electrical and Electronic Engineering, The University of Hong Kong

[2] United Microelectronic Center (Hong Kong) Limited

Video Presentation For NeurIPS 2021

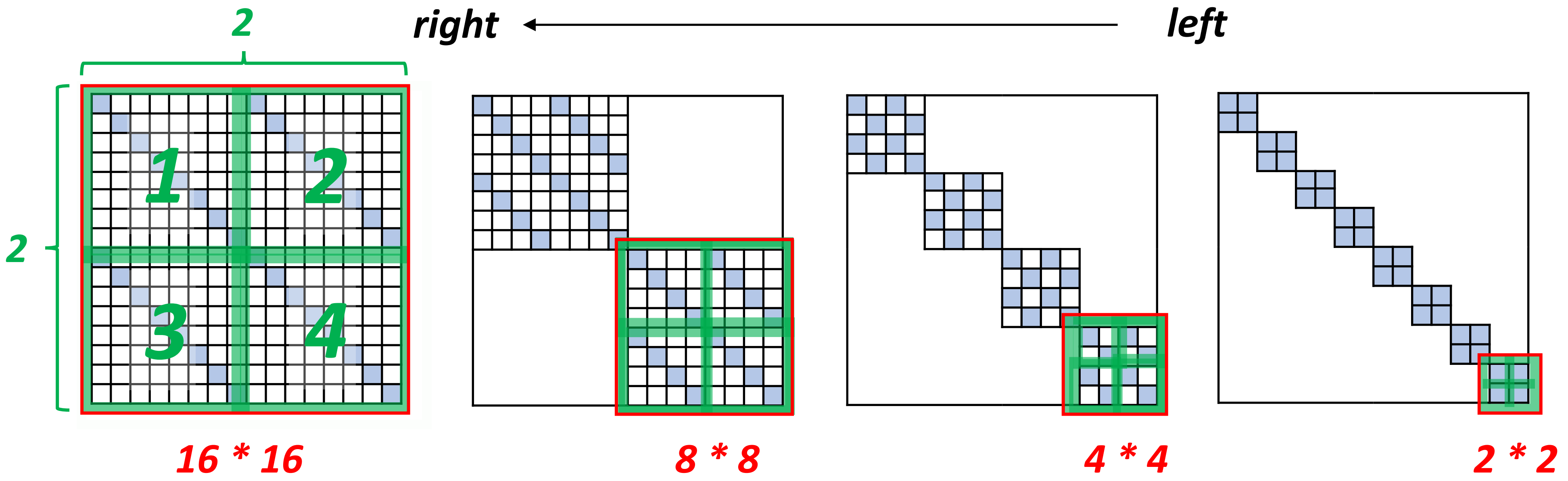
NEURAL INFORMATION PROCESSING SYSTEMS

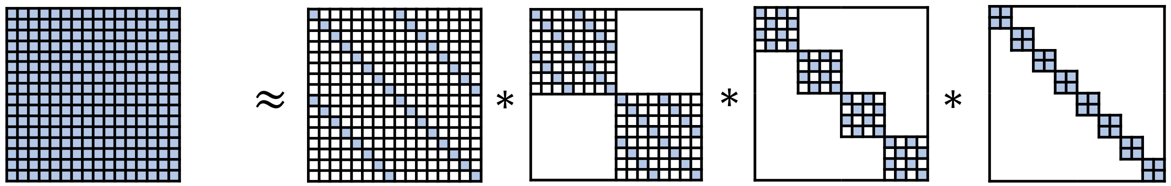
* RL, JR, and NW contributed equally to this work.

Source Code: https://github.com/RuiLin0212/DeBut

Content

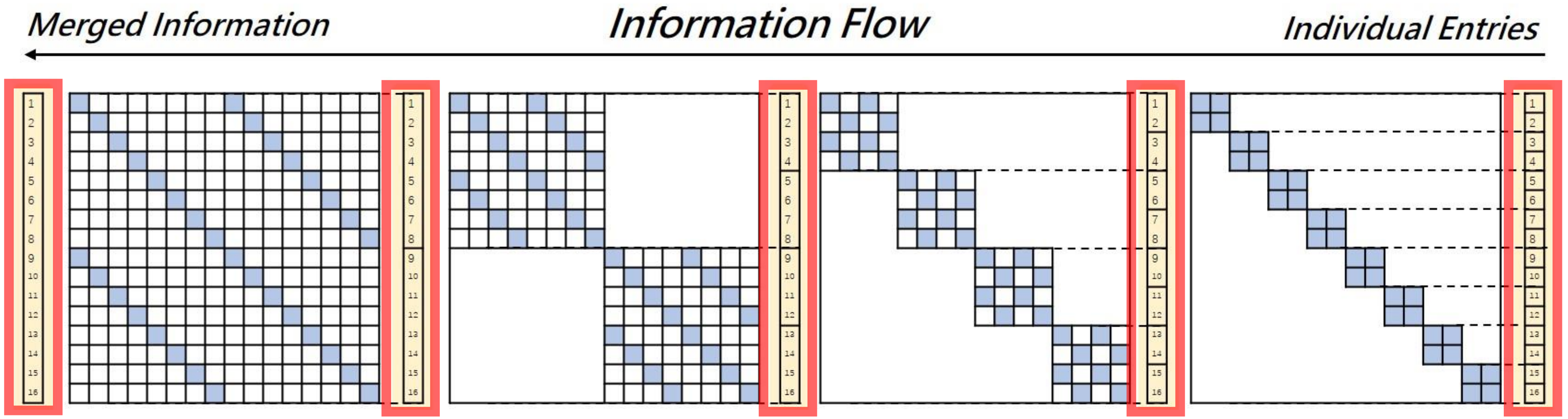
Butterfly Matrix



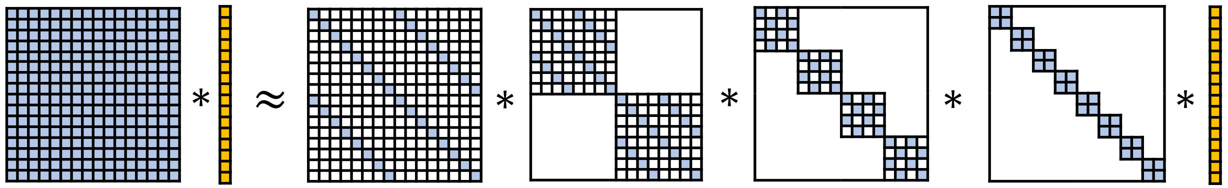


<u>Standard</u> Butterfly matrix has ***<u>Powers-of-Two limitations.</u>***

Butterfly Matrix

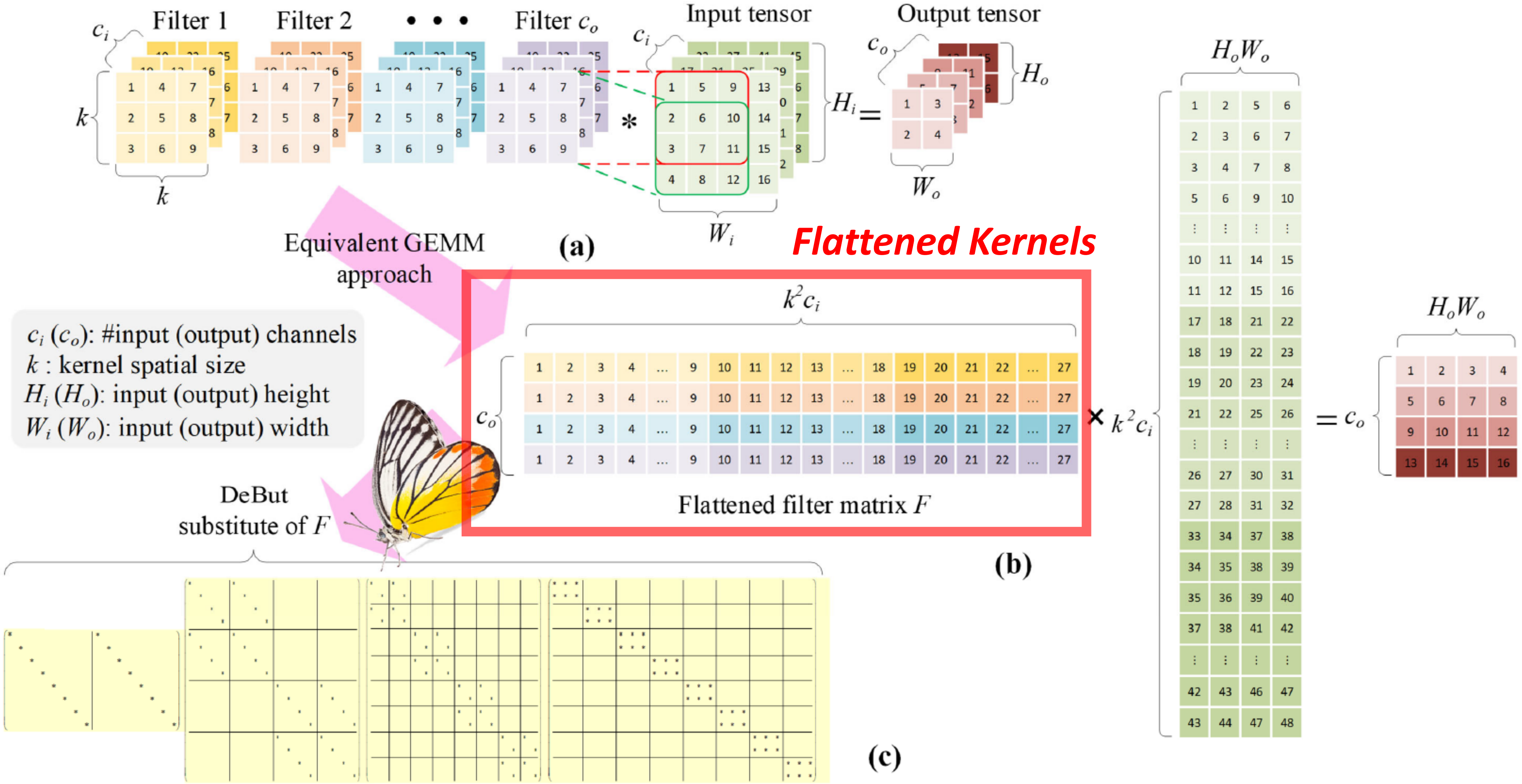


Fully-mixed | Eight in a group | Four in a group | Two in a group | Separate numbers

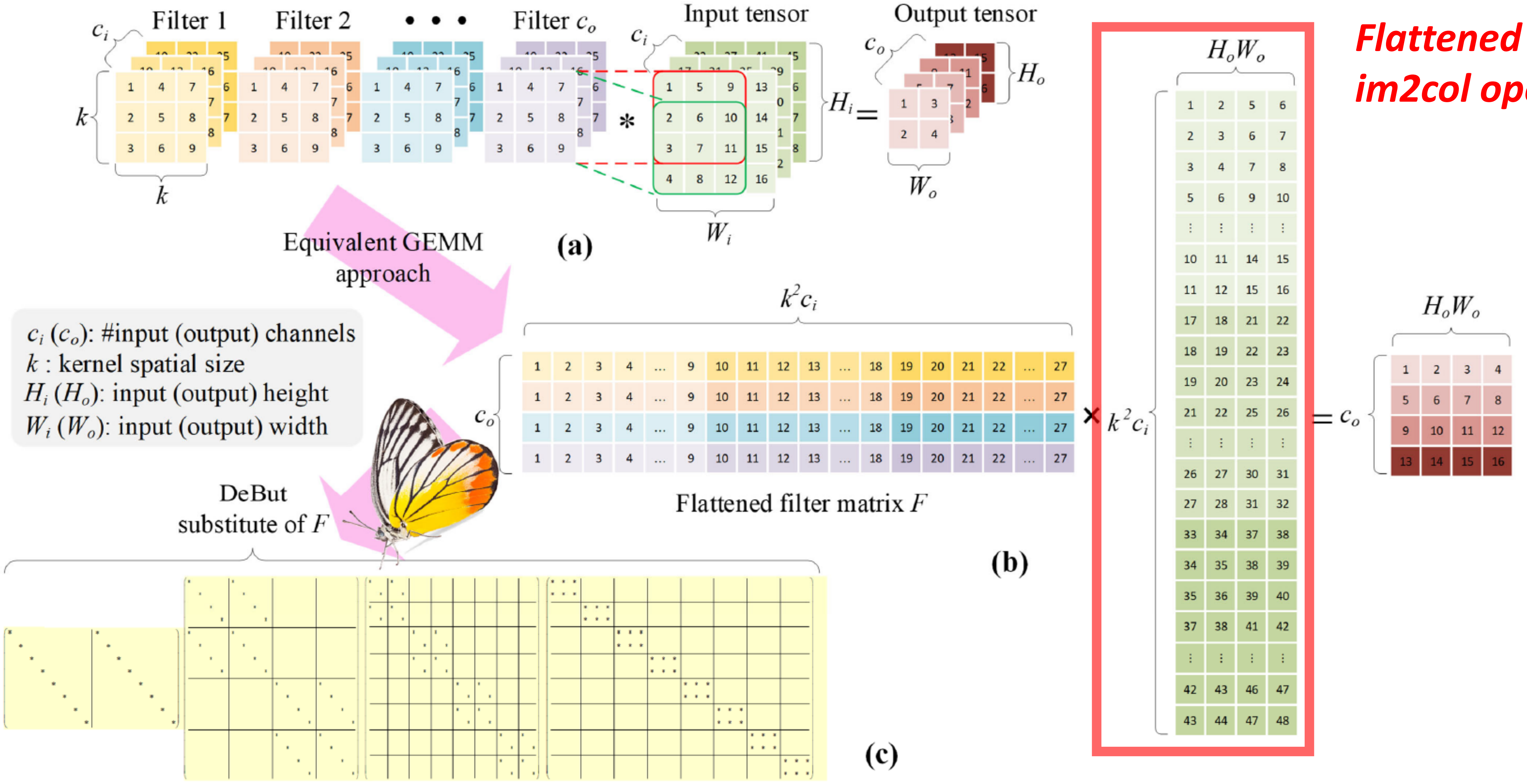


<u>From right to left</u>, the ***<u>information carried by the input is merged</u>***.

Convolution as a Matrix Product

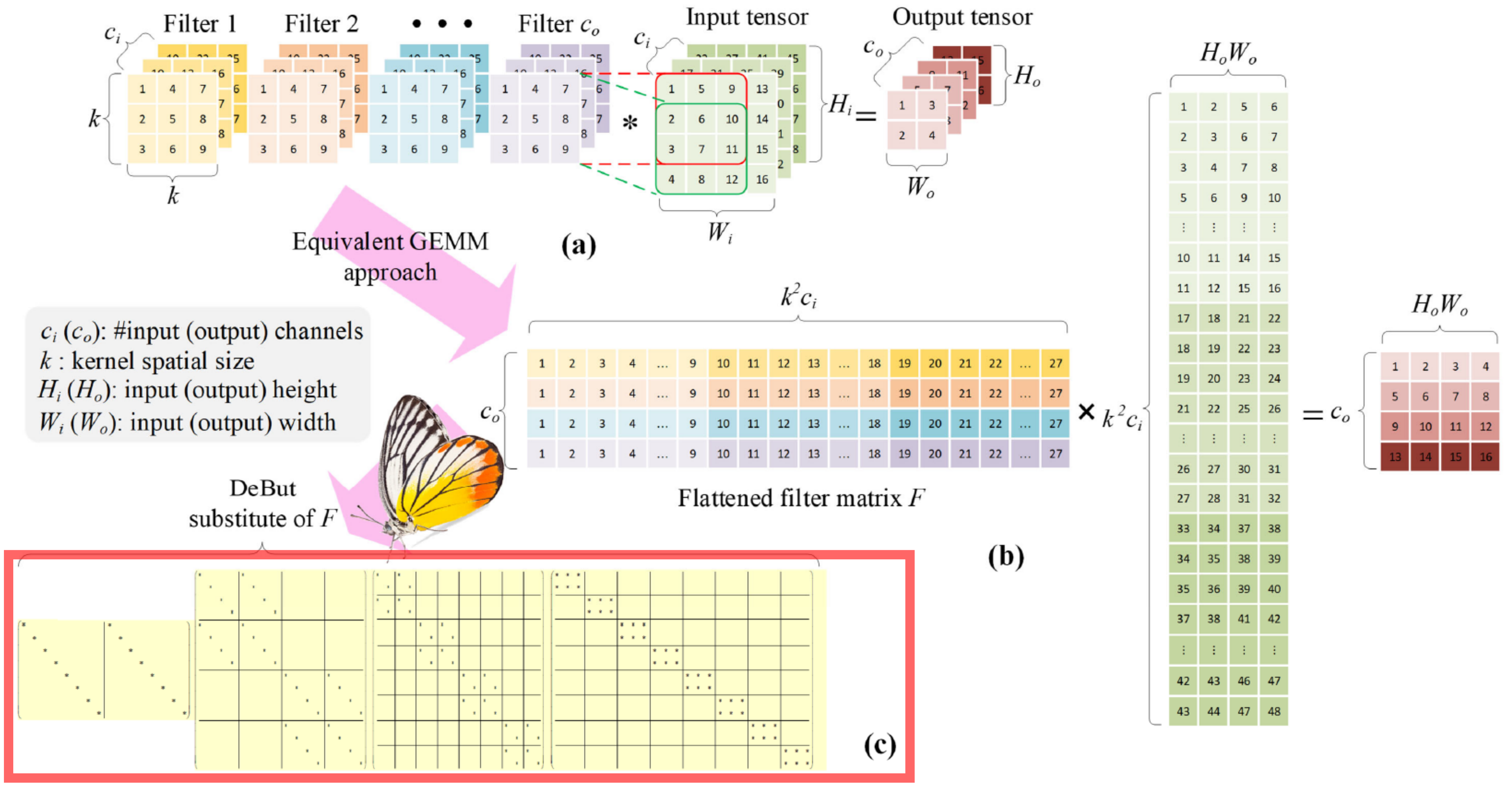


Convolution as a Matrix Product



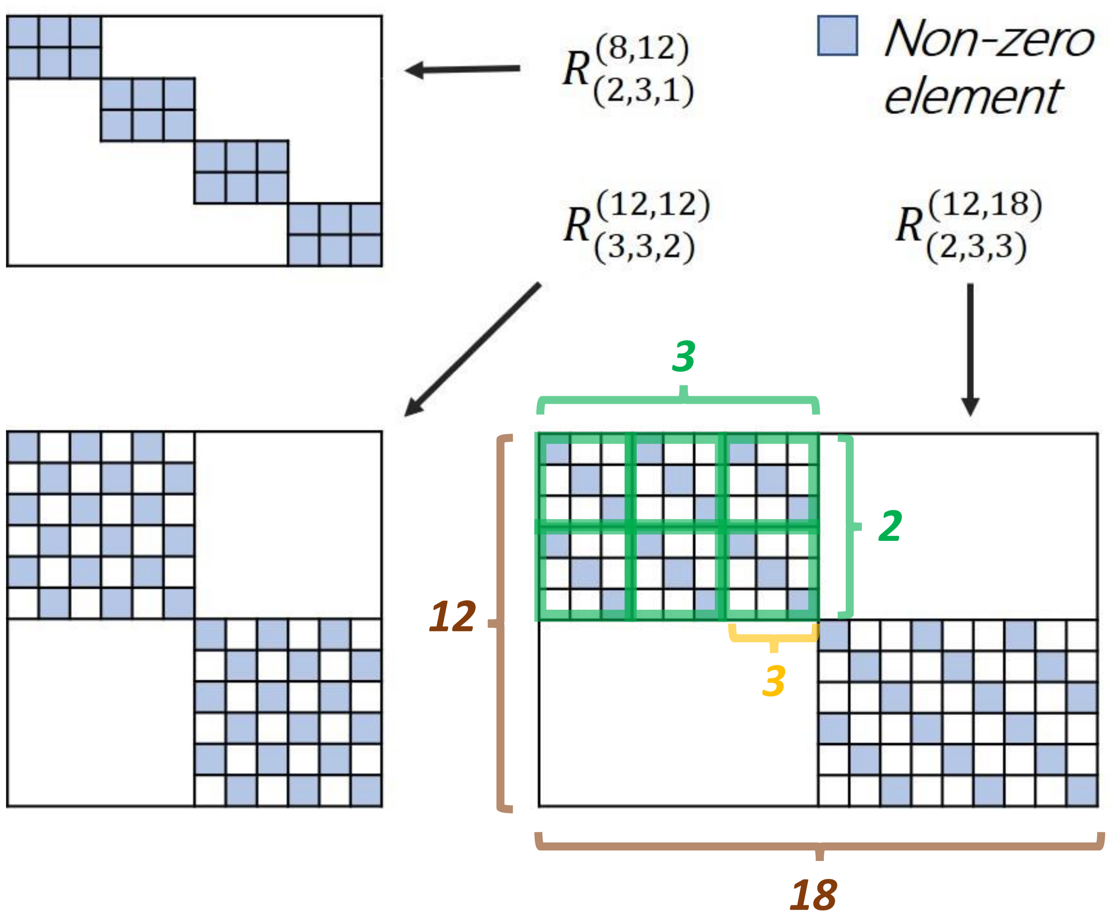
Flattened Input by im2col operation

Convolution as a Matrix Product



Linear Transform

DeBut Chains



The spatial size of the factor

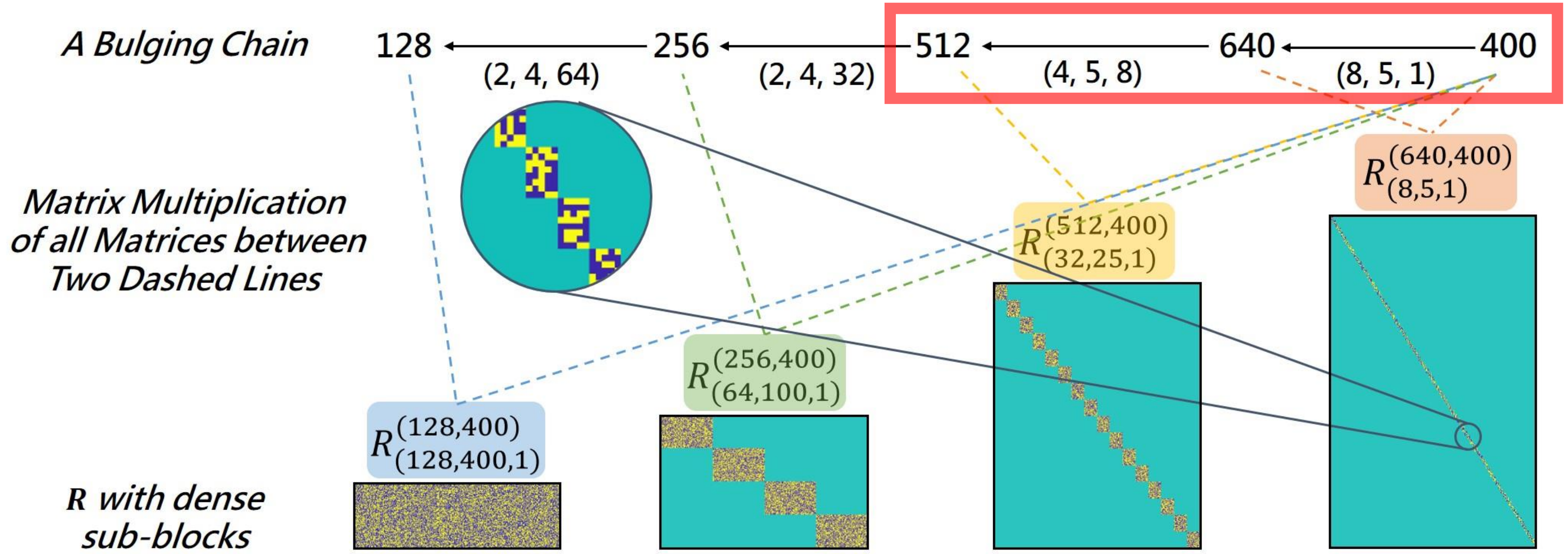
$$R^{\boxed{(p,q)}}_{\boxed{(r,s)},\boxed{t})} \in \mathbb{R}^{p\times q}$$

Number of diagonal matrices in the block

The size of the diagonal matrix

No more *Powers-of-Two limitations!*

DeBut Chains



A bulging DeBut chain

Another possible DeBut chain: $128 \xleftarrow[(2,2,64)]{} \boxed{128 \xleftarrow[(2,4,32)]{} 256 \xleftarrow[(32,50,1)]{} 400}$

A monotonic DeBut chain

DeBut Chains

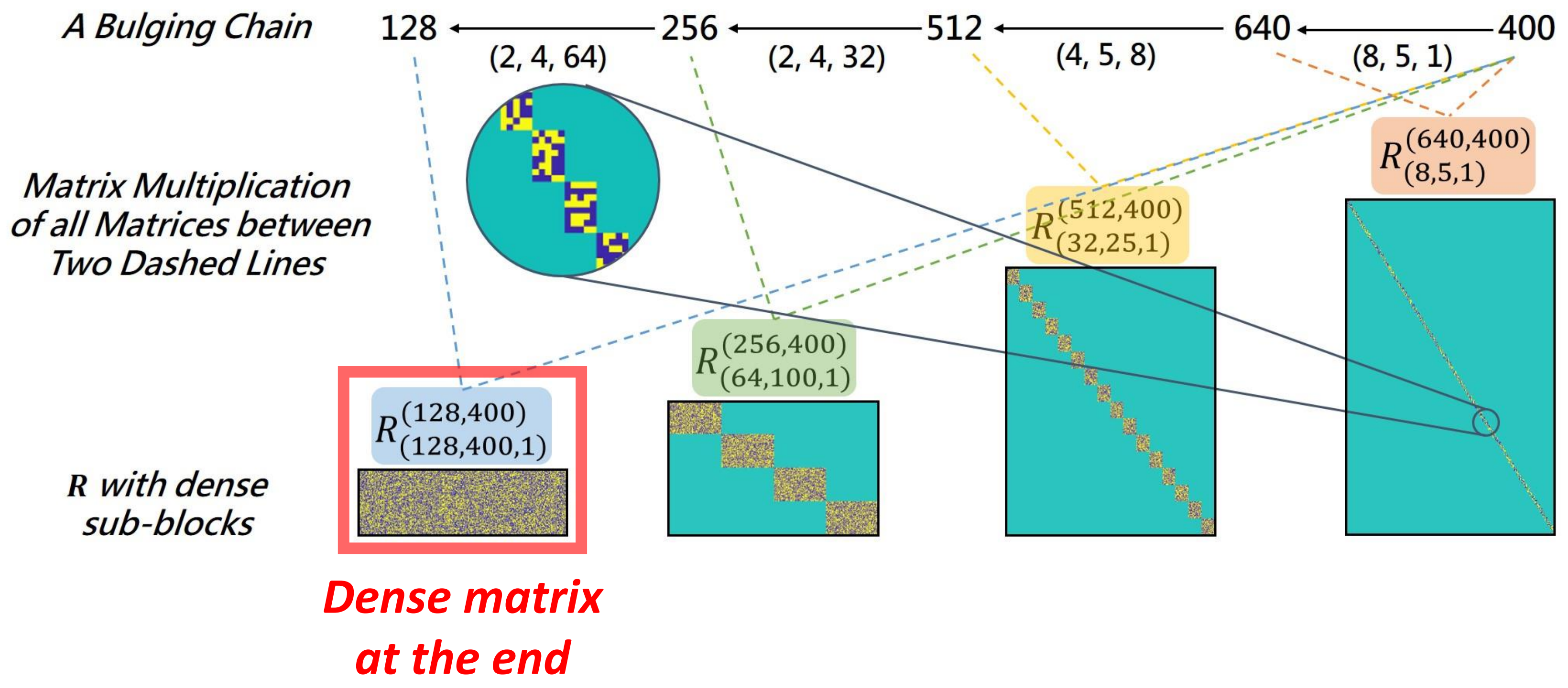
A bulging DeBut chain

$$128 \xleftarrow[(2,4,64)]{} 256 \xleftarrow[(2,4,32)]{} \boxed{512 \xleftarrow[(4,5,8)]{} 640 \xleftarrow[(8,5,1)]{} 400}$$

A monotonic DeBut chain

$$128 \xleftarrow[(2,2,64)]{} \boxed{128 \xleftarrow[(2,4,32)]{} 256 \xleftarrow[(32,50,1)]{} 400}$$

DeBut Chains



<u>No entry</u> *in the linear transform is* ***<u>forced zero</u>***.

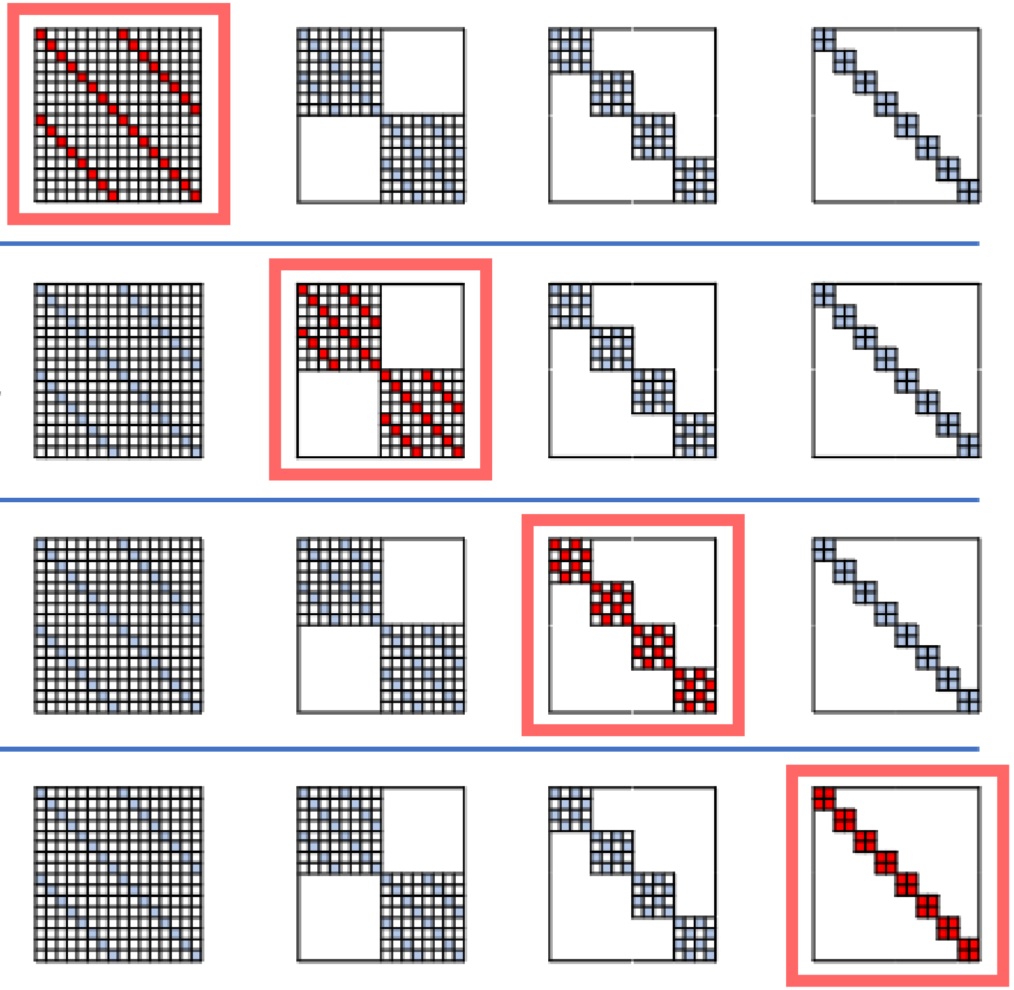
Alternating Least Squares (ALS)

Update The 1st Factor

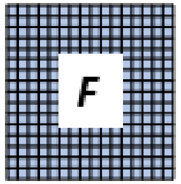
Update The 2nd Factor

Update The 3rd Factor

Update The 4th Factor



To Approximate $\approx$



$$error = \frac{|F - \hat{F}|_2}{|F|_2}$$

Each time only one factor is updated

LeNet Trained on MNIST (Baseline: 99.29%)

Layer	Monotonic/Bulging Chains	LC	MC	Params	Acc% (no ALS)	Acc% (w/ ALS)
FC1	$\xleftarrow[(1,2,32)]{} 256 \xleftarrow[(2,2,16)]{} 256 \xleftarrow[(16,25,1)]{} 400$ $128 \xleftarrow[(2,2,64)]{} 128 \xleftarrow[(2,2,32)]{} 128$	85.00%	70.78%	$17.96K$	98.89($\pm$0.08)	98.72($\pm$0.02)
CONV2	$16 \xleftarrow[(2,2,8)]{} 16 \xleftarrow[(2,6,4)]{} 48 \xleftarrow[(1,2,4)]{} 96 \xleftarrow[(4,3,1)]{} 72$	41.67%	1.04%	$60.84K$	99.02($\pm$0.06)	98.86($\pm$0.02)
CONV2	$16 \xleftarrow[(2,6,8)]{} 48 \xleftarrow[(1,2,8)]{} 96 \xleftarrow[(2,2,4)]{} 96 \xleftarrow[(4,3,1)]{} 72$	41.67%				
FC1	$\xleftarrow[(1,2,32)]{} 256 \xleftarrow[(2,2,16)]{} 256 \xleftarrow[(16,25,1)]{} 400$ $128 \xleftarrow[(2,2,64)]{} 128 \xleftarrow[(2,2,32)]{} 128$	85.00%	83.43%	$10.19K$	98.64($\pm$0.15)	97.27($\pm$0.02)
FC2	$\xleftarrow[(2,2,8)]{} 64 \xleftarrow[(2,2,4)]{} 64 \xleftarrow[(2,2,2)]{} 64 \xleftarrow[(2,4,1)]{} 128$ $64 \xleftarrow[(2,2,32)]{} 64 \xleftarrow[(2,2,16)]{} 64$	89.06%				

Reduce the #params significantly

DeBut is able to ***reduce the number of parameters*** further while delivering ***promising output accuracy***.

LeNet Trained on MNIST (Baseline: 99.29%)

Layer	Monotonic/Bulging Chains	LC	MC	Params	Acc% (no ALS)	Acc% (w/ ALS)
FC1	$\xleftarrow[(1,2,32)]{} 256 \xleftarrow[(2,2,16)]{} 256 \xleftarrow[(16,25,1)]{} 400$ $128 \xleftarrow[(2,2,64)]{} 128 \xleftarrow[(2,2,32)]{} 128$	85.00%	70.78%	$17.96K$	$98.89(\pm0.08)$	$98.72(\pm0.02)$
CONV2	$16 \xleftarrow[(2,2,8)]{} 16 \xleftarrow[(2,6,4)]{} 48 \xleftarrow[(1,2,4)]{} 96 \xleftarrow[(4,3,1)]{} 72$	41.67%	1.04%	$60.84K$	$99.02(\pm0.06)$	$98.86(\pm0.02)$
CONV2	$16 \xleftarrow[(2,6,8)]{} 48 \xleftarrow[(1,2,8)]{} 96 \xleftarrow[(2,2,4)]{} 96 \xleftarrow[(4,3,1)]{} 72$	41.67%				
FC1	$\xleftarrow[(1,2,32)]{} 256 \xleftarrow[(2,2,16)]{} 256 \xleftarrow[(16,25,1)]{} 400$ $128 \xleftarrow[(2,2,64)]{} 128 \xleftarrow[(2,2,32)]{} 128$	85.00%	83.43%	$10.19K$	$98.64(\pm0.15)$	$97.27(\pm0.02)$
FC2	$\xleftarrow[(2,2,8)]{} 64 \xleftarrow[(2,2,4)]{} 64 \xleftarrow[(2,2,2)]{} 64 \xleftarrow[(2,4,1)]{} 128$ $64 \xleftarrow[(2,2,32)]{} 64 \xleftarrow[(2,2,16)]{} 64$	89.06%				

w/o ALS has better performance in small examples

In the ***small example***, ALS initialization is ***not always*** beneficial for learning the latent information. However, its advantage will become obvious in ***larger examples***.

VGG Trained on CIFAR-10 (Baseline: 93.96%)

Layer	Chain(s)	LC	MC	Params	Acc% (w/ ALS)
CONV13	$4096 \xleftarrow{(2,2,16)} 4096 \xleftarrow{(2,2,8)} 4096 \xleftarrow{(8,9,1)} 4608$ $512 \xleftarrow{(2,4,256)} 1024 \xleftarrow{(2,4,128)} 2048 \xleftarrow{(2,4,64)} 4096 \xleftarrow{(2,2,32)}$	96.79%	15.23%	$12.71M$	$93.91(\pm0.08)$
CONV8	$2048 \xleftarrow{(2,2,32)} 2048 \xleftarrow{(2,2,16)} 2048 \xleftarrow{(2,2,8)} 2048 \xleftarrow{(8,9,1)} 2304$ $512 \xleftarrow{(2,2,256)} 512 \xleftarrow{(2,4,128)} 1024 \xleftarrow{(2,4,64)}$	96.79%	83.77%	$2.43M$	$93.72(\pm0.07)$
CONV9∼13	$4096 \xleftarrow{(2,2,16)} 4096 \xleftarrow{(2,2,8)} 4096 \xleftarrow{(8,9,1)} 4608$ $512 \xleftarrow{(2,4,256)} 1024 \xleftarrow{(2,4,128)} 2048 \xleftarrow{(2,4,64)} 4096 \xleftarrow{(2,2,32)}$	96.79%			

Only a slight 0.24% accuracy drop

Amazingly, this VGG-16-BN with DeBut layers achieves ***a remarkable MC of 83.77%*** with only ***a slight 0.24% accuracy drop***.

ResNet-50 Trained on ImageNet (Baseline: 76.01%)

Model	MC	Params	Top-1(%) with ALS	Top-5(%) with ALS
ResNet-50	$--$	$25.55M$	76.01	92.93
DeBut-bulging	47.56%	$13.40M$	74.52	92.18
DeBut-mono	48.74%	$13.10M$	74.34	92.31

47.56% Parameters fewer, 1.49% accuracy lower

For ***DeBut-bulging***, the number of parameters reduces by ***47.56%***, the top-1 accuracy is 74.52%, ***1.47% lower*** compared with the baseline 76.01%.

ResNet-50 Trained on ImageNet (Baseline: 76.01%)

Model	MC	Params	Top-1(%) with ALS	Top-5(%) with ALS
ResNet-50	$--$	$25.55M$	76.01	92.93
DeBut-bulging	47.56%	$13.40M$	74.52	92.18
DeBut-mono	48.74%	$13.10M$	74.34	92.31

0.3M Parameters fewer than Debut-bulging, the accuracy 74.34% is comparable

For ***DeBut-mono***, the compressed model has ***0.3M fewer parameters than the DeBut-bulging*** model, yet still achieving ***a comparable 74.34% top-1 accuracy***.

DeBut vs. Other Linear Transform Schemes

Layer	Method	MC	Params	Acc%	Training Time(s/epoch)	Inference Time(s)
CONV8∼13	Adaptive Fastfood	85.65%	$2.15M$	$93.60(\pm0.02)$	2100	148.27
	Butterfly	85.82%	$2.13M$	$93.34(\pm0.12)$	105	4.58
	DeBut	83.77%	$2.43M$	$93.72(\pm0.07)$	50	4.01

** VGG trained on CIFAR-10 using different linear transform schemes*

DeBut is the fastest algorithm

Fast Hadamard Transform in Adaptive Fastfood is **exceedingly time-consuming** and makes it ***difficult*** to train Adaptive FastFood on ***multiple layers in a large CNN***.

DeBut vs. Conventional Compression Schemes

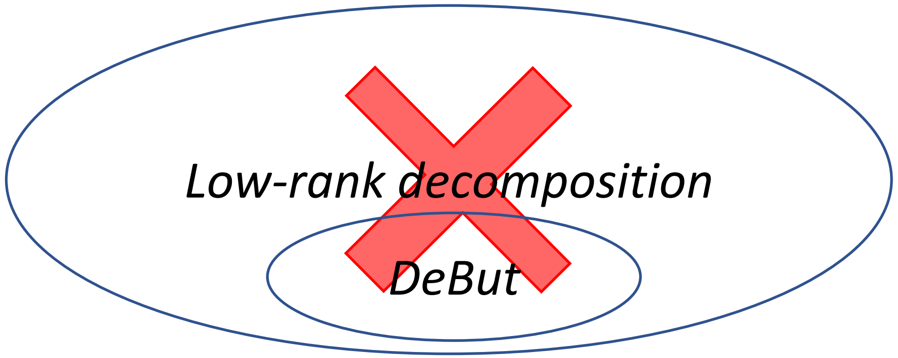
Redundant model → *Pruning / Quantization* → Compact model → *DeBut* → More compact model

Debut are orthogonal and complementary to Pruning & Quantization

Layer	Method	Params	Acc(%)
CONV8∼13	Baseline	$14.99M$	93.96
	Tucker-2	$3.21M$	93.36
	DeBut	$2.43M$	93.71

** VGG trained on CIFAR-10 using different compression methods*

DeBut achieves better performance with fewer parameters



Conclusion

1. DeBut Layers are ***truly practical*** .
2. DeBut is ***fundamentally different*** from standard butterfly.
3. DeBut layers ***further unify* and homogenize** FC and CONV layers.
4. Bulging and monotonic chains have different ***compression ability, performance and stability***.
5. The DeBut factor product chain provides an important implication for a ***pipelined DNN inference speedup***.

Thanks for Your Attention!

If you have any question, please contact us.