Deformable Butterfly: A Highly Structured and Sparse Linear Transform

Rui Lin[1,*], Jie Ran[1,*], King Hung Chiu[2], Graziano Chesi[1], Ngai Wong[1,*]
[1] Dept. of EEE, The University of Hong Kong * Equal Contribution
[2] United Microelectronics Centre (Hong Kong) Limited, HKSTP, Hong Kong
Email: {linrui, jieran, chesi, nwong}@eee.hku.hk khchiu@umechk.com

1. Butterfly Matrix

1.1 Standard Butterfly Matrix Format

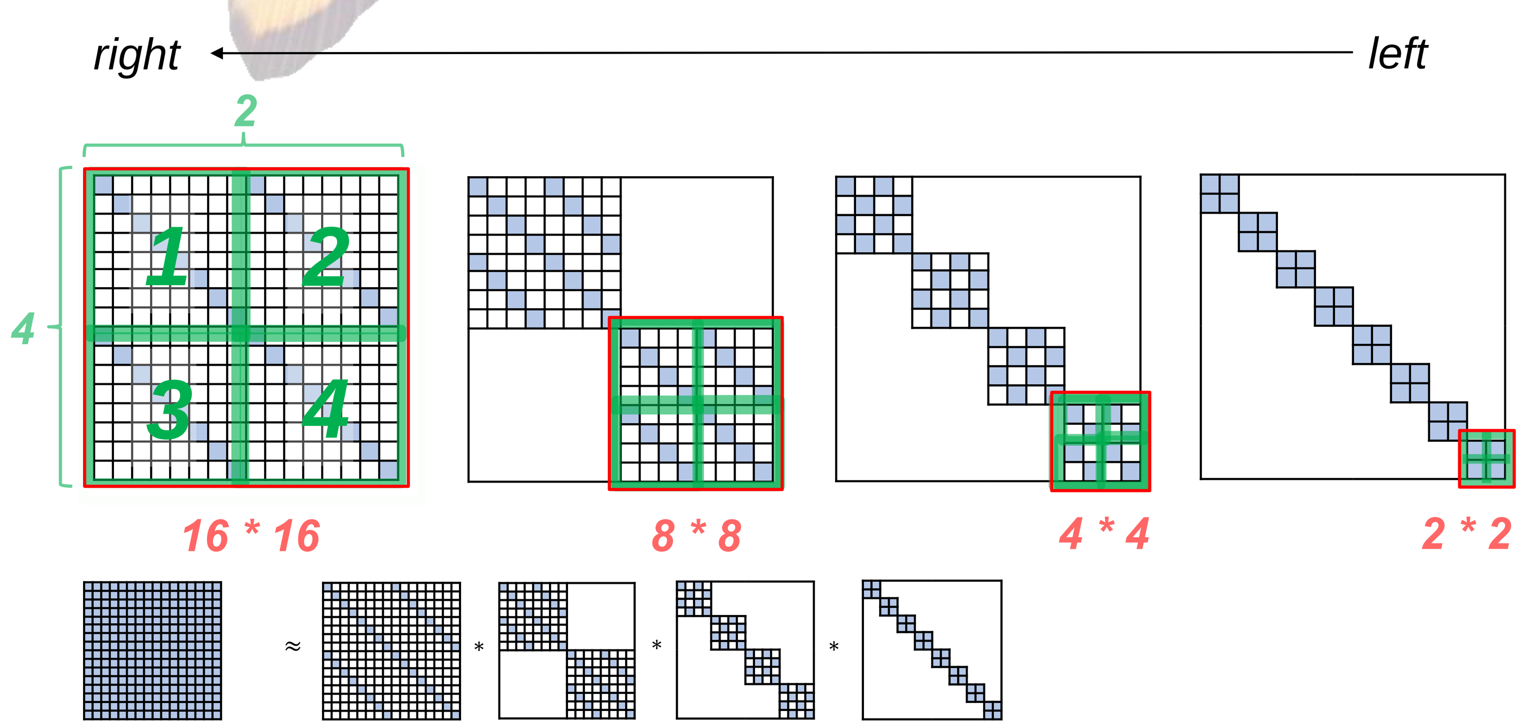

Figure 1. (Upper) 16 × 16 Butterfly factor matrices, the blue squares represent the nonzero elements. (Lower) Taking the 16 × 16 matrix on the left side as an example, the sparse Butterfly matrices are used to approximate the given dense matrix, which can reduce the number of parameters significantly.

- *From **right to left**, the size of the **blocks becomes larger.***
- Standard Butterfly matrices can be regarded as ***block-diagonal matrices.*** The blocks are formed by four diagonal matrices, organized in the 2 × 2 way.
- It is evident that the standard Butterfly matrix has the ***powers-of-two limitations.***

1.2 Information Flow Viewpoint

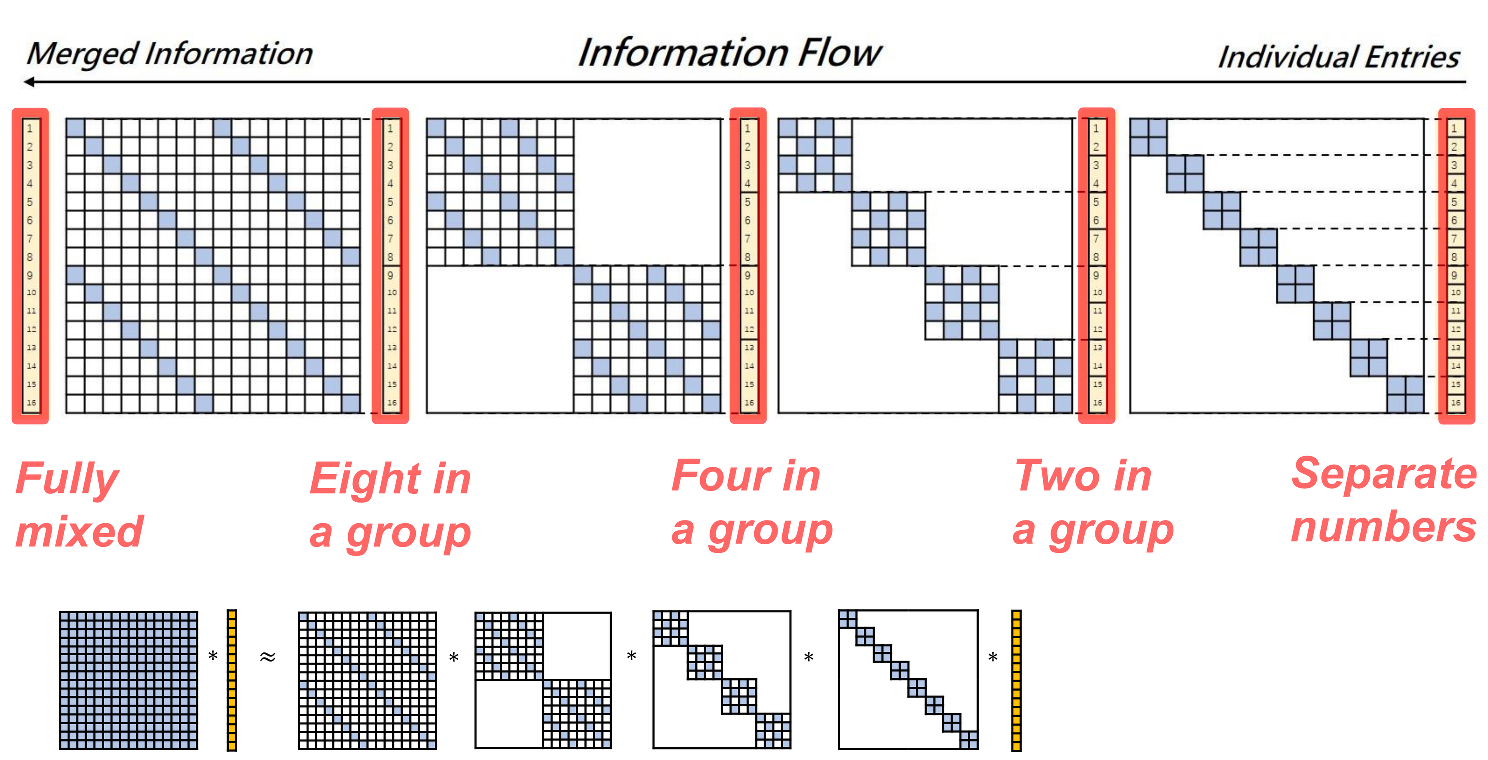

Figure 2. (Upper) 16 × 16 Butterfly factor matrices and the hierarchical information flow from right to left, where the blue squares stand for nonzeros and the numbers in the vectors denote the positional indices. The dashed lines, which connect the DeBut factor and the vector, mark the entries that will be mixed up in the vector and the corresponding sub-block in the DeBut factor that works as the mixer. The partitions in the vector denote the merging of information. (Lower) A vector is used to multiply with the matrix and the four butterfly factors.

- ***Each block in a Butterfly factor matrix works as a mixer***, which mixes the information carried by the vector.
- From right to left, ***the information carried by the input is merged***.

2. Deformable Butterfly (DeBut)

2.1 Convolution as a Matrix Product

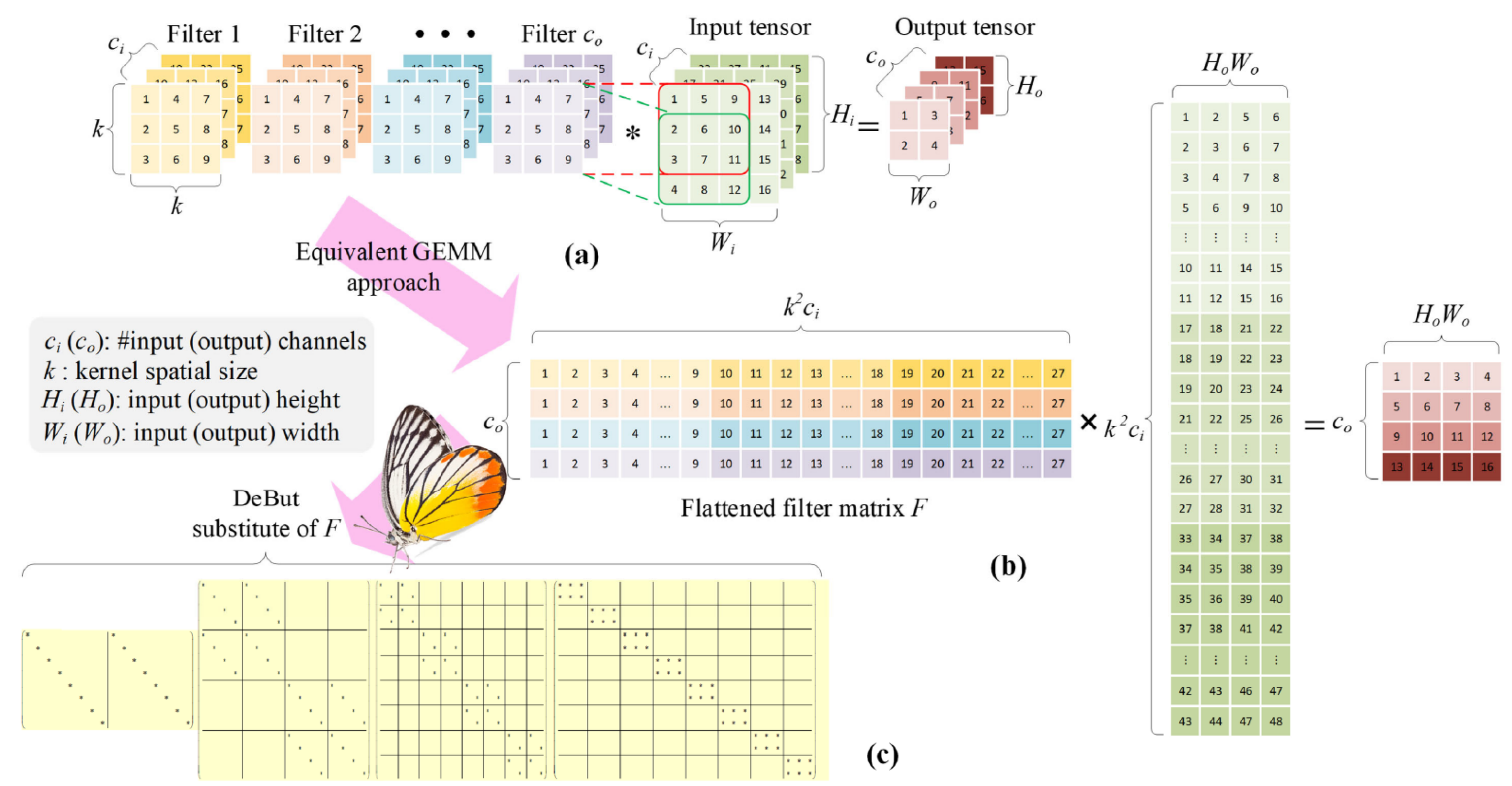

Figure 3. CNN convolution in its (a) conceptual, illustrative form; (b) equivalent matrix-matrix implementation by a flattened kernel matrix; (c) DeBut replacement of the kernel matrix.

2.2 Notation

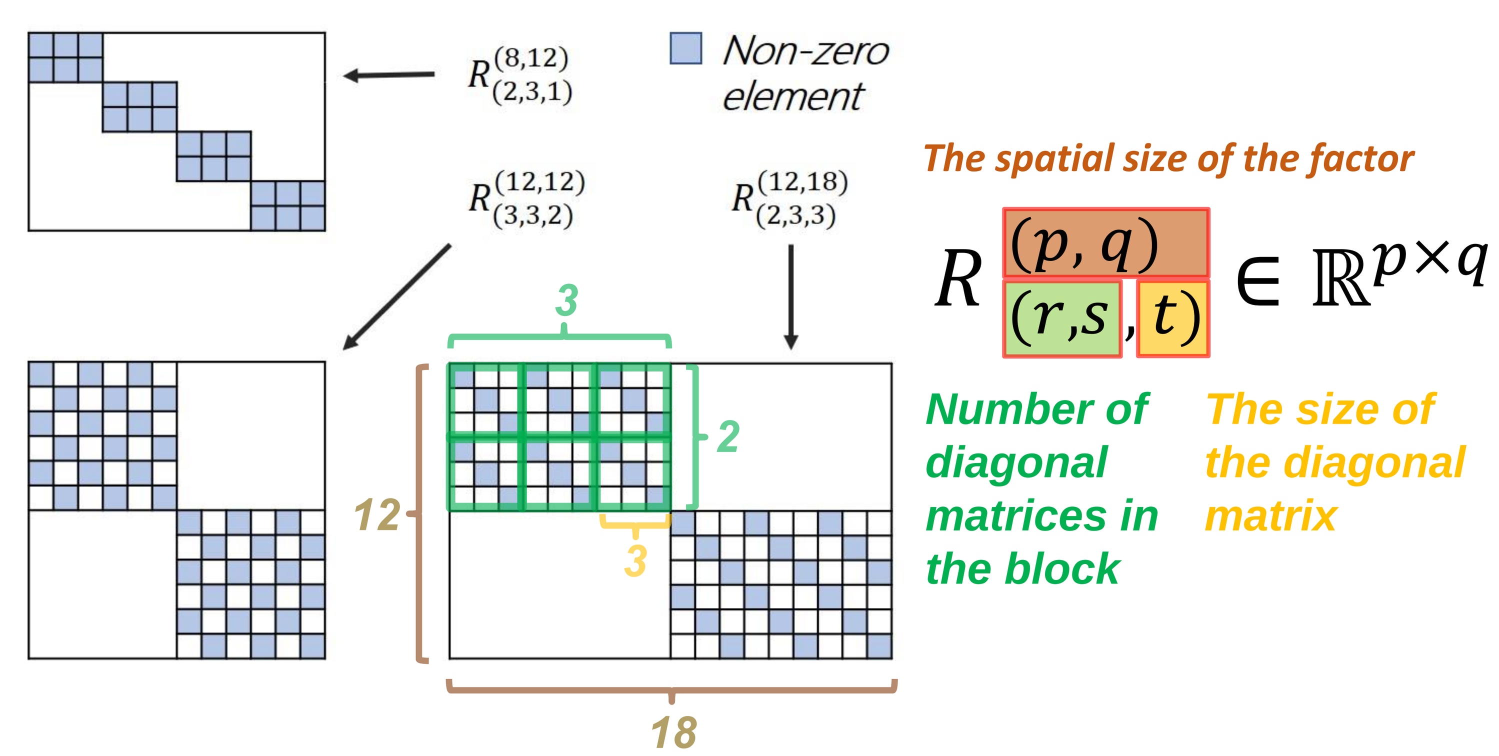

Figure 4. Notation and Example of the DeBut factors.

2.3 DeBut Chains

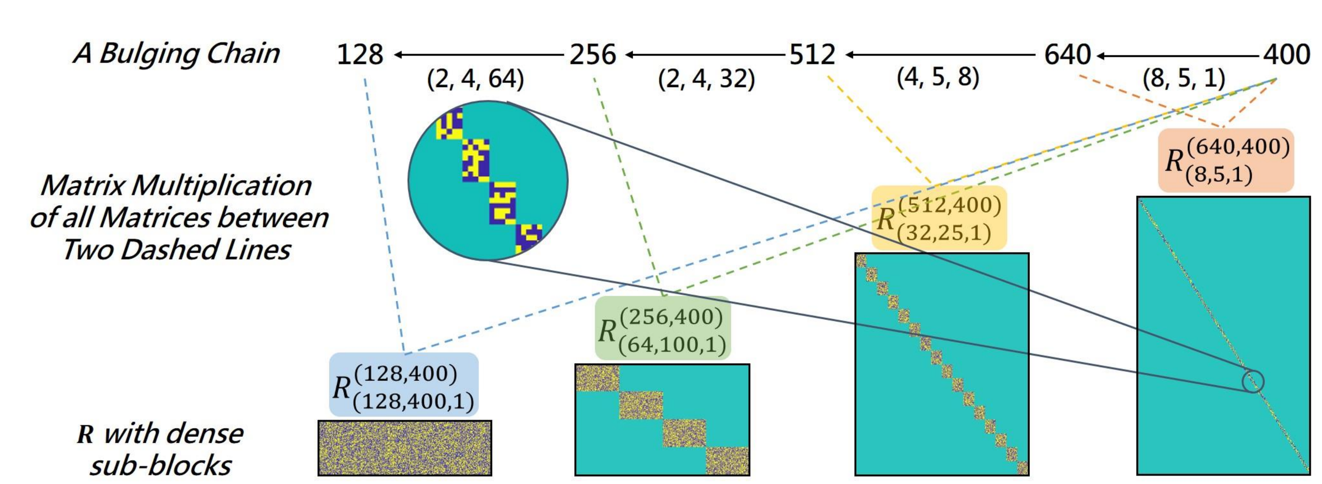

Figure 5. A bulging DeBut chain (not drawn to scale) and its densification process from right to left. The yellow, blue, and teal in the plots denote +1, -1, and 0, respectively.

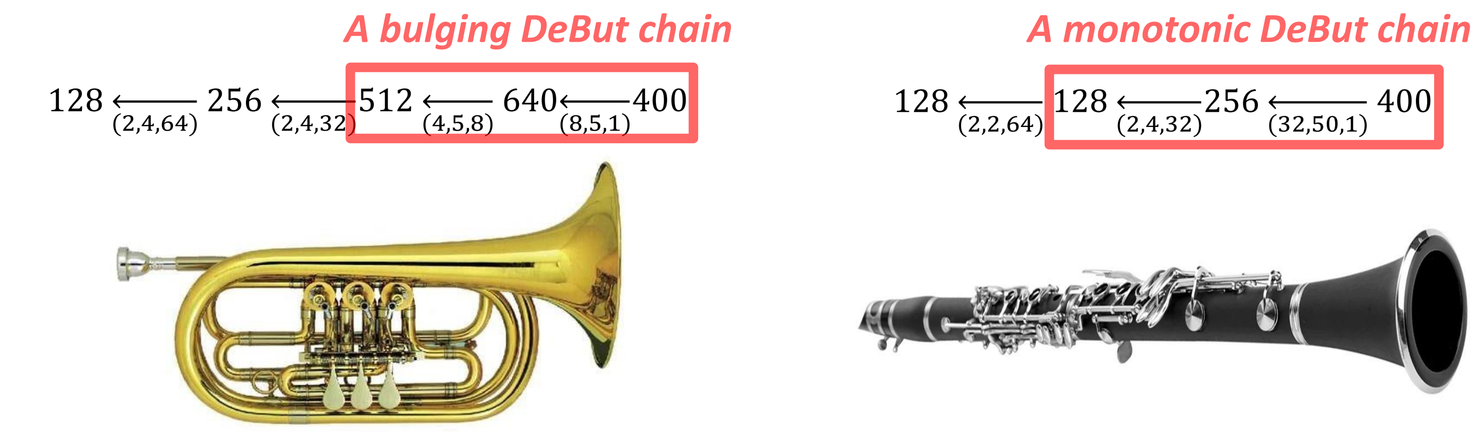

Figure 6. (Left) An example of a bulging DeBut chain, which is like a bass trumpet bulging in the middle. (Right) An example of a monotonic DeBut chain, which is like a clarinet having a monotonous shape.

2.4 Alternating Least Squares (ALS)

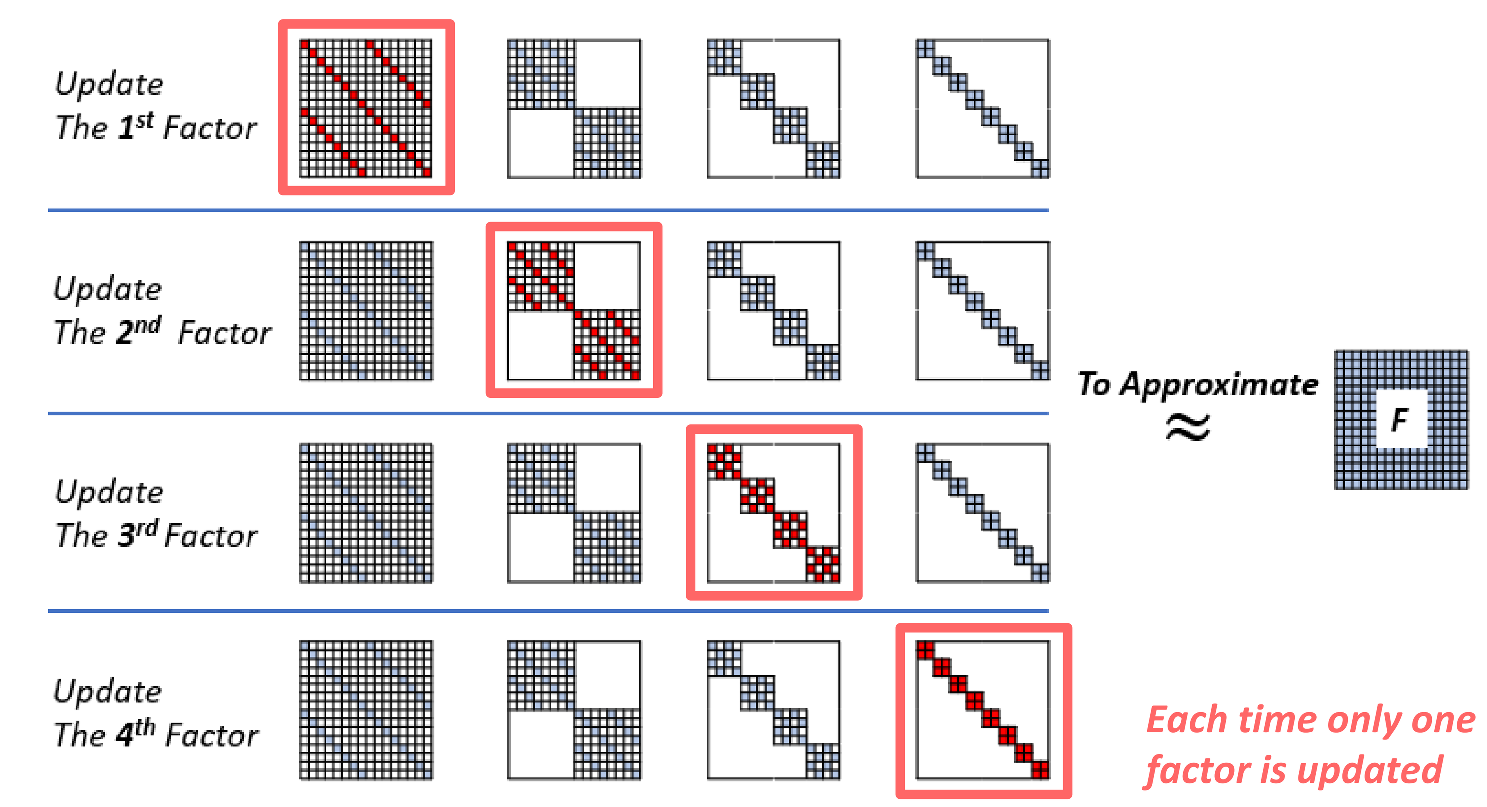

Figure 6. Alternating Least Squares (ALS) is employed to initialize the DeBut factors when substituting a selected layer in a pretrained neural network.

3. Selected Experimental Results

Method	MC	Params	Acc%	Training Time(s/epoch)	Inference Time(s)
Adaptive Fastfood	85.65%	$2.15M$	93.60(±0.02)	2100	148.27
Butterfly	85.82%	$2.13M$	93.34(±0.12)	105	4.58
DeBut	83.77%	$2.43M$	93.72(±0.07)	50	4.01

Table 1. Comparison results for VGG-16-BN on CIFAR-10. It showcases the effectiveness of DeBut versus the other two methods. It can be seen that DeBut achieves the highest prediction accuracy with only 0.3M more parameters. We also note that Adaptive Fastfood takes around 2100s for each training epoch, making its training prohibitively slow even for CIFAR-10.

Main References

Dao, T., et al. (2019, May). “Learning fast algorithms for linear transforms using butterfly factorizations”. In International conference on machine learning (pp. 1517-1527). PMLR.

Dao, T et al. (2020). “Kaleidoscope: An efficient, learnable representation for all structured linear maps”. arXiv preprint arXiv:2012.14966.

Le, Q. et al. (2013, June). “Fastfood-approximating kernel expansions in loglinear time”. In Proceedings of the international conference on machine learning (Vol. 85).

Yang, Z. et al. (2015). “Deep fried convnets”. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1476-1483).