Deep Learning Optimization - Network Compression

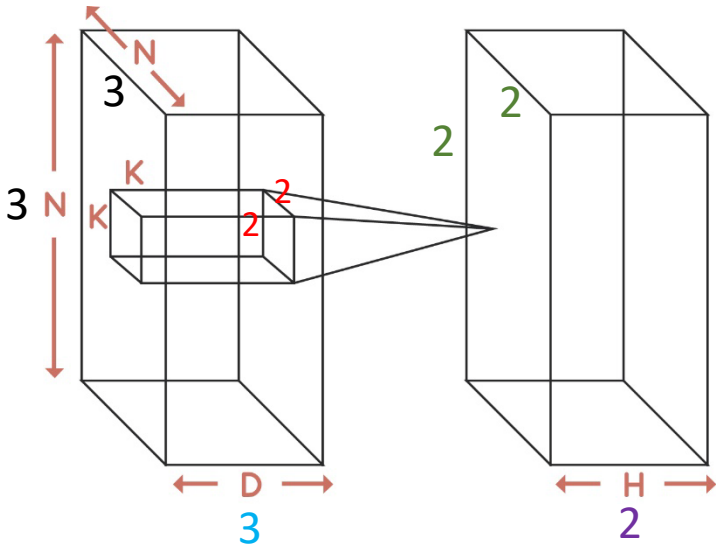
May 15, 2023

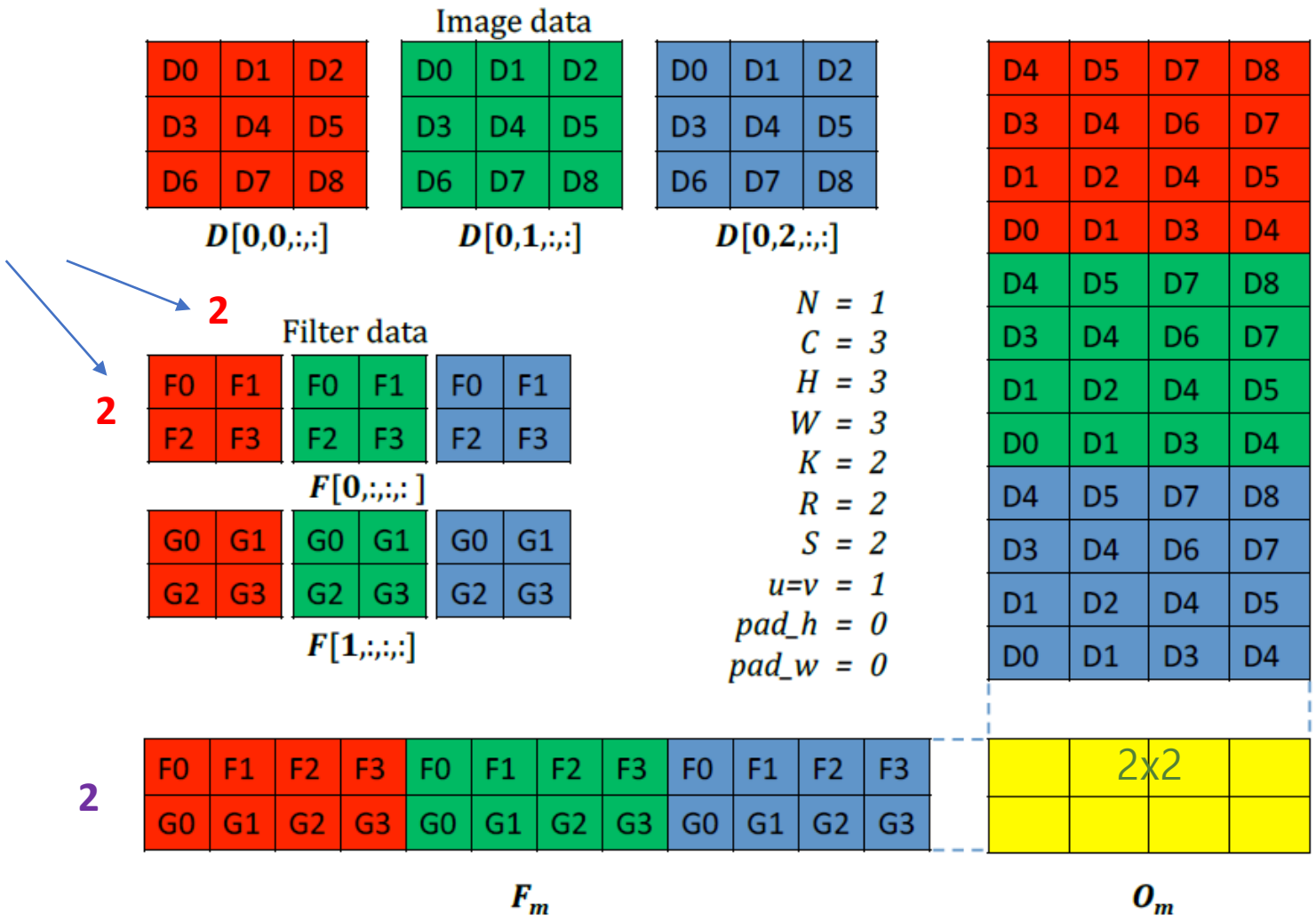
Eunhyeok Park

Compression of Deep Convolutional Neural Networks for Fast and Low Power Applications

Convolution with Matrix Multiplication (Convolution Lowering)

- Input: 3x3x3
- Output: 2x2x2
- Convolutional kernel: 3x2x2





Low-Rank Approximation

- Based on truncated singular value decomposition (SVD)
- Stronger methods
  - E.g., CP decomposition, Tucker decomposition

Singular Value Decomposition (SVD)

SVD is based on a theorem from linear algebra which says that a rectangular matrix A can be broken down into the product of three matrices - an orthogonal matrix U, a diagonal matrix S, and the transpose of an orthogonal matrix V. The theorem is usually presented something like this:

$$A_{mn} = U_{mm} S_{mn} V_{nn}^T$$

where $U^T U = I, V^T V = I$; the columns of U are orthonormal eigenvectors of AA^T, the columns of V are orthonormal eigenvectors of $A^T A$, and S is a diagonal matrix containing the square roots of eigenvalues from U or V in descending order.

[K. Baker, Singular Value Decomposition Tutorial]

SVD Examples

$$A = \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix} = U_{mm} S_{mn} V^T_{nn} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \sqrt{12} & 0 & 0 \\ 0 & \sqrt{10} & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ \frac{2}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & 0 \\ \frac{1}{\sqrt{30}} & \frac{2}{\sqrt{30}} & \frac{-5}{\sqrt{30}} \end{bmatrix}$$

$$AA^T = \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 3 & -1 \\ 1 & 3 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix}$$

$$\begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

eigenvectors are defined by the equation $A\vec{v} = \lambda\vec{v}$,

$$A^T A = \begin{bmatrix} 3 & -1 \\ 1 & 3 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix} = \begin{bmatrix} 10 & 0 & 2 \\ 0 & 10 & 4 \\ 2 & 4 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad \lambda = 10, \lambda = 12$$

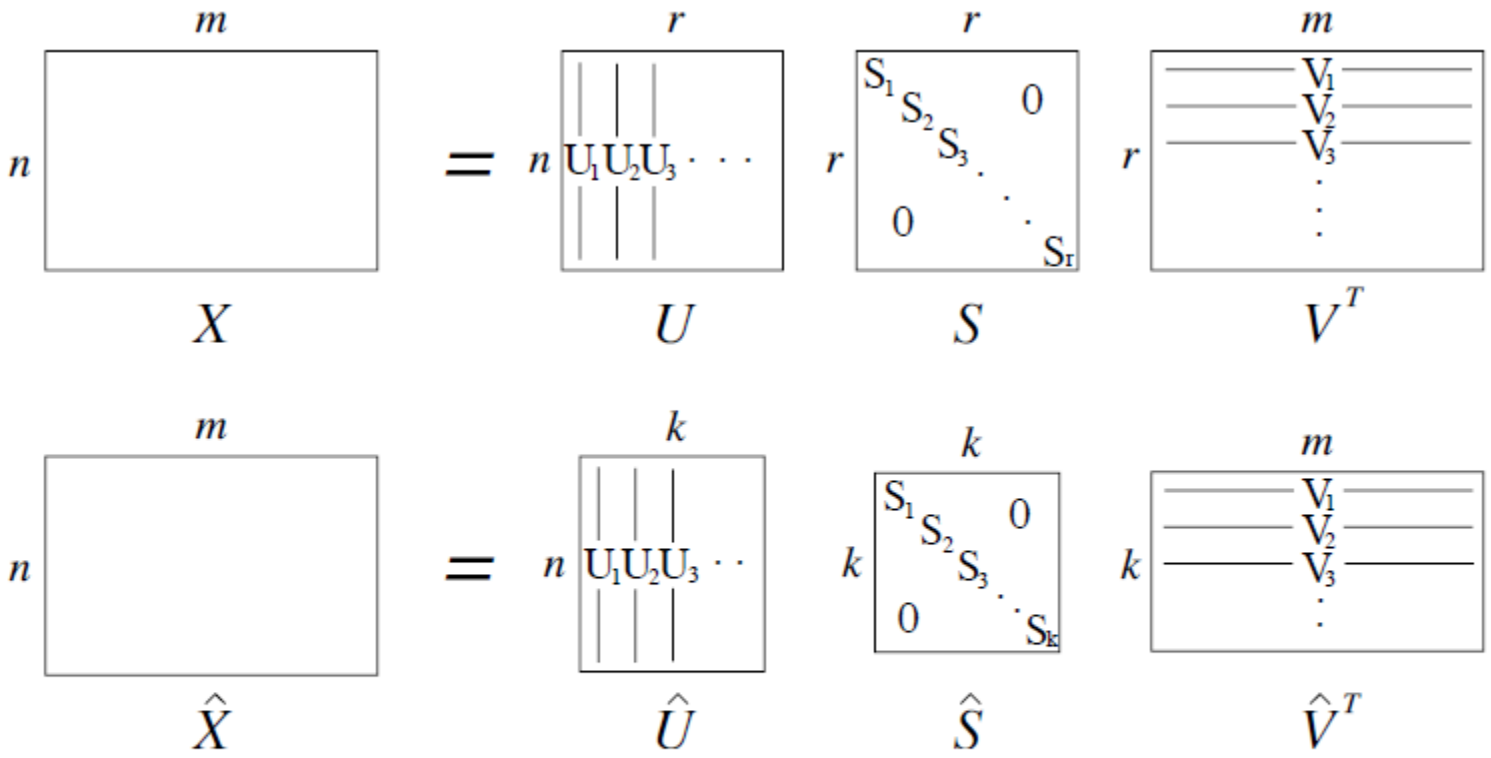
$$V^T = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ \frac{2}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & 0 \\ \frac{1}{\sqrt{30}} & \frac{2}{\sqrt{30}} & \frac{-5}{\sqrt{30}} \end{bmatrix}$$

After normalization,
we obtain an orthonormal matrix, *U*

$$U = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix}$$

[K. Baker, Singular Value Decomposition Tutorial]

A Low-Rank Approximation: Truncated SVD



$\hat{X}$ is the best rank k approximation to X, in terms of least squares.

[http://cs224d.stanford.edu/lectures/CS224d-Lecture2.pdf]

Example of Truncated SVD: $A = USV^T$

$$A = \begin{bmatrix} 2 & 0 & 8 & 6 & 0 \\ 1 & 6 & 0 & 1 & 7 \\ 5 & 0 & 7 & 4 & 0 \\ 7 & 0 & 8 & 5 & 0 \\ 0 & 10 & 0 & 0 & 7 \end{bmatrix}$$

$$AA^T = \begin{bmatrix} 2 & 0 & 8 & 6 & 0 \\ 1 & 6 & 0 & 1 & 7 \\ 5 & 0 & 7 & 4 & 0 \\ 7 & 0 & 8 & 5 & 0 \\ 0 & 10 & 0 & 0 & 7 \end{bmatrix} \begin{bmatrix} 2 & 1 & 5 & 7 & 0 \\ 0 & 6 & 0 & 0 & 10 \\ 8 & 0 & 7 & 8 & 0 \\ 6 & 1 & 4 & 5 & 0 \\ 0 & 7 & 0 & 0 & 7 \end{bmatrix} = \begin{bmatrix} 104 & 8 & 90 & 108 & 0 \\ 8 & 87 & 9 & 12 & 109 \\ 90 & 9 & 90 & 111 & 0 \\ 108 & 12 & 111 & 138 & 0 \\ 0 & 109 & 0 & 0 & 149 \end{bmatrix}$$

eigenvectors are defined by the equation $A\vec{v} = \lambda\vec{v}$,

Eigenvalues of AA^T are

$$\lambda = 321.07, \lambda = 230.17, \lambda = 12.70, \lambda = 3.94, \lambda = 0.12$$

$$U = \begin{bmatrix} -0.54 & 0.07 & 0.82 & -0.11 & 0.12 \\ -0.10 & -0.59 & -0.11 & -0.79 & -0.06 \\ -0.53 & 0.06 & -0.21 & 0.12 & -0.81 \\ -0.65 & 0.07 & -0.51 & 0.06 & 0.56 \\ -0.06 & -0.80 & 0.09 & 0.59 & 0.04 \end{bmatrix}$$

U consists of 5 eigenvectors

[K. Baker, Singular Value Decomposition Tutorial]

Example of Truncated SVD: $A = USV^T$

$$A^T A = \begin{bmatrix} 79 & 6 & 107 & 68 & 7 \\ 6 & 136 & 0 & 6 & 112 \\ 107 & 0 & 177 & 116 & 0 \\ 68 & 6 & 116 & 78 & 7 \\ 7 & 112 & 0 & 7 & 98 \end{bmatrix}$$

Calculating eigenvectors & normalization

$$V^T = \begin{bmatrix} -0.46 & 0.02 & -0.87 & -0.00 & 0.17 \\ -0.07 & -0.76 & 0.06 & 0.60 & 0.23 \\ -0.74 & 0.10 & 0.28 & 0.22 & -0.56 \\ -0.48 & 0.03 & 0.40 & -0.33 & 0.70 \\ -0.07 & -0.64 & -0.04 & -0.69 & -0.32 \end{bmatrix}$$

- Approximate A (with 5 ranks) with 3 ranks, i.e., 3 eigenvalues and vectors

[K. Baker, Singular Value Decomposition Tutorial]

Example of Truncated SVD: $A \sim USV^T$

$$U = \begin{bmatrix} -0.54 & 0.07 & 0.82 & -0.11 & 0.12 \\ -0.10 & -0.59 & -0.11 & -0.79 & -0.06 \\ -0.53 & 0.06 & -0.21 & 0.12 & -0.81 \\ -0.65 & 0.07 & -0.51 & 0.06 & 0.56 \\ -0.06 & -0.80 & 0.09 & 0.59 & 0.04 \end{bmatrix}$$

$$\lambda = 321.07, \lambda = 230.17, \lambda = 12.70, \lambda = 3.94, \lambda = 0.12$$

$$V^T = \begin{bmatrix} -0.46 & 0.02 & -0.87 & -0.00 & 0.17 \\ -0.07 & -0.76 & 0.06 & 0.60 & 0.23 \\ -0.74 & 0.10 & 0.28 & 0.22 & -0.56 \\ -0.48 & 0.03 & 0.40 & -0.33 & 0.70 \\ -0.07 & -0.64 & -0.04 & -0.69 & -0.32 \end{bmatrix}$$

Take 3 eigenvectors associated with the selected eigenvalues

Take 3 largest square roots of eigenvalues

Take 3 eigenvectors associated with the selected eigenvalues

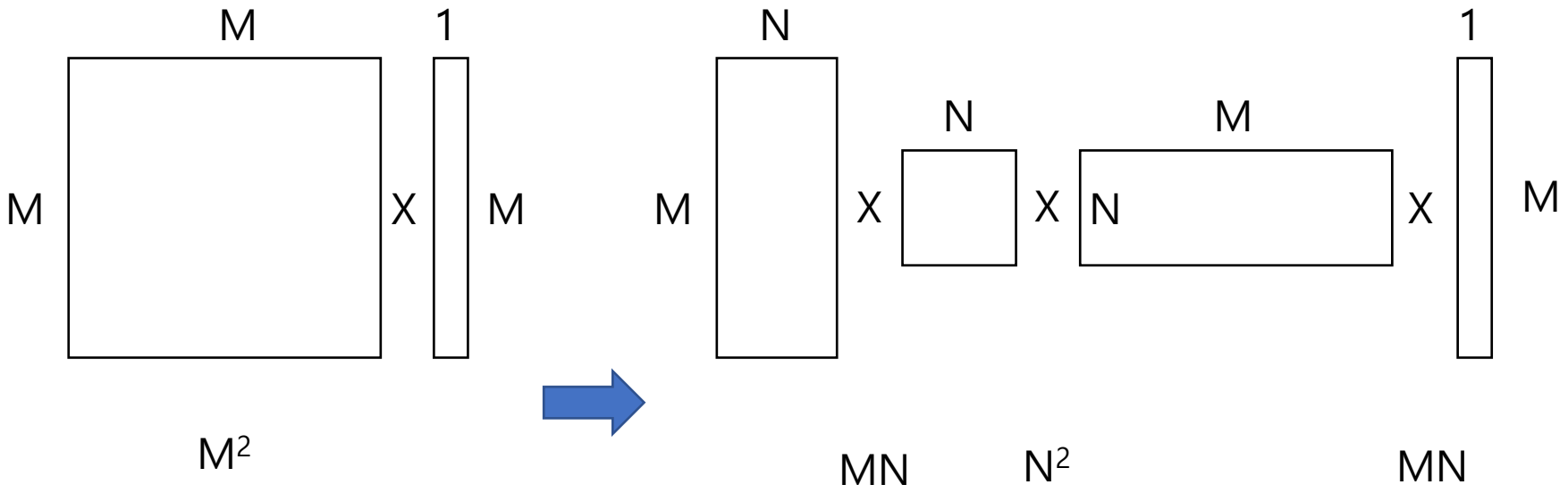
$$\hat{A} = \begin{bmatrix} -0.54 & 0.07 & 0.82 \\ -0.10 & -0.59 & -0.11 \\ -0.53 & 0.06 & -0.21 \\ -0.65 & 0.07 & -0.51 \\ -0.06 & -0.80 & 0.09 \end{bmatrix} \begin{bmatrix} 17.92 & 0 & 0 \\ 0 & 15.17 & 0 \\ 0 & 0 & 3.56 \end{bmatrix} \begin{bmatrix} -0.46 & 0.02 & -0.87 & -0.00 & 0.17 \\ -0.07 & -0.76 & 0.06 & 0.60 & 0.23 \\ -0.74 & 0.10 & 0.28 & 0.22 & -0.56 \end{bmatrix}$$

Error degrades accuracy. How to reclaim lost accuracy?

$$= \begin{bmatrix} 2.29 & -0.66 & 9.33 & 1.25 & -3.09 \\ 1.77 & 6.76 & 0.90 & -5.50 & -2.13 \\ 4.86 & -0.96 & 8.01 & 0.38 & -0.97 \\ 6.62 & -1.23 & 9.58 & 0.24 & -0.71 \\ 1.14 & 9.19 & 0.33 & -7.19 & -3.13 \end{bmatrix} \Leftrightarrow A = \begin{bmatrix} 2 & 0 & 8 & 6 & 0 \\ 1 & 6 & 0 & 1 & 7 \\ 5 & 0 & 7 & 4 & 0 \\ 7 & 0 & 8 & 5 & 0 \\ 0 & 10 & 0 & 0 & 7 \end{bmatrix}$$

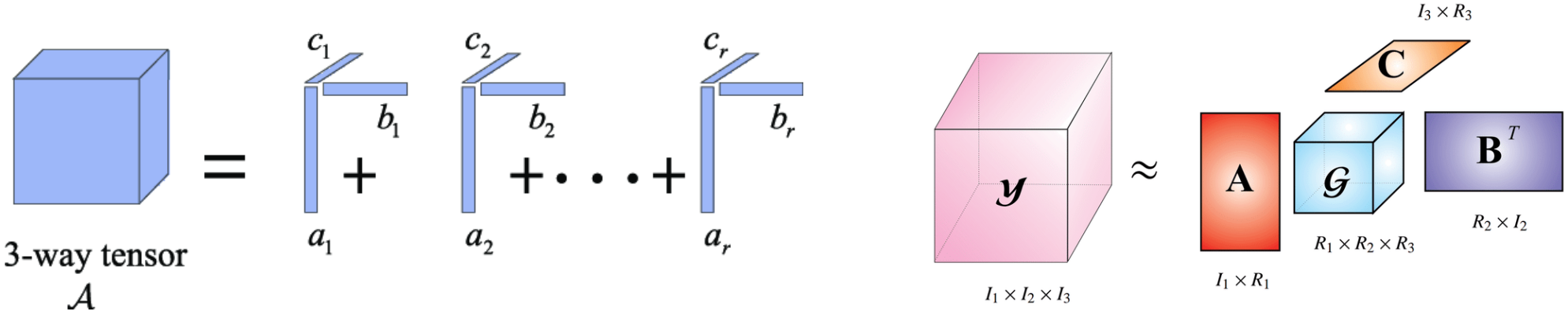
Truncated SVD can Reduce Computation

- MM x M → MN x NN x NM x M
- # multiplications: M^2 → $2MN + N^2$
  - M=100, N=20
  - 10000 → 4400 (56% reduction)



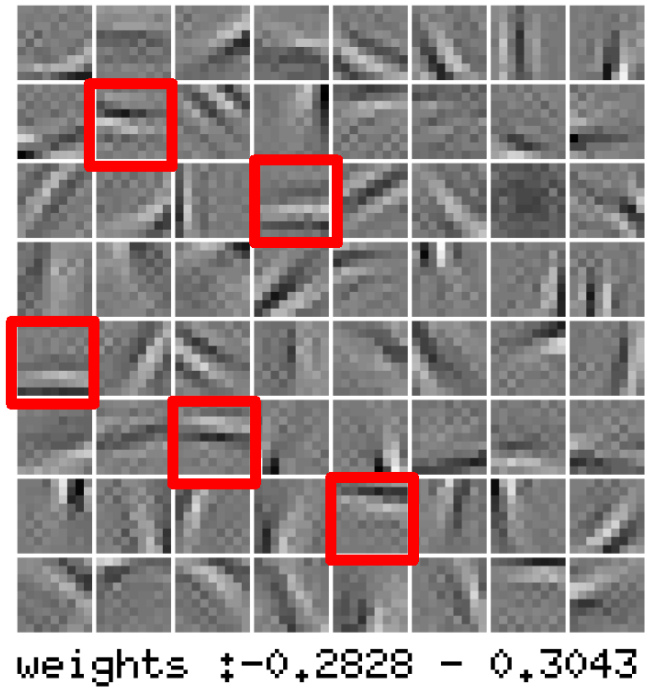
Low-Rank Approximation

- Based on truncated singular value decomposition (SVD)
- Stronger methods
  - E.g., CP decomposition, Tucker decomposition



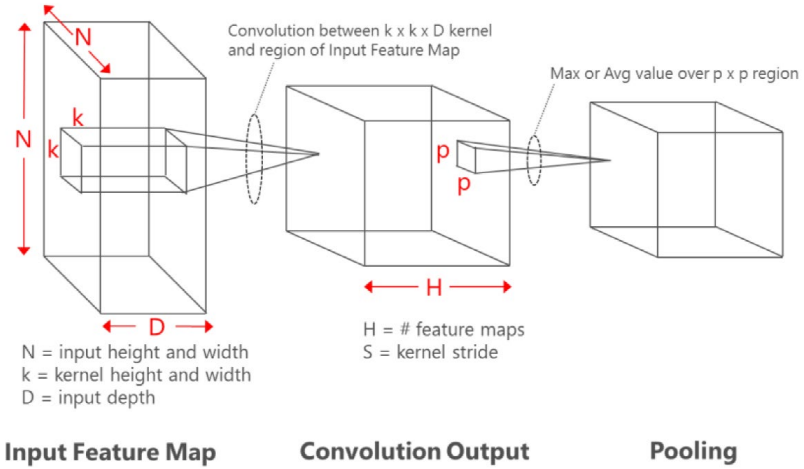
Observation: Redundancy Problem

- **Problem:**
  - With patch-level training, the learning algorithm must reconstruct the entire patch with a single feature vector
  - But when the filters are used convolutionally, neighboring feature vectors will be highly redundant



Yann LeCun

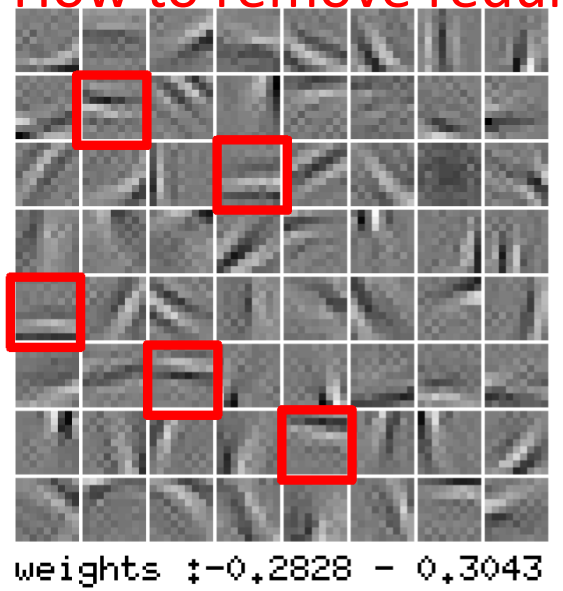
Patch-level training produces lots of filters that are shifted versions of each other.



Tucker Method to Resolve Redundancy Problem: Reducing # Feature Maps

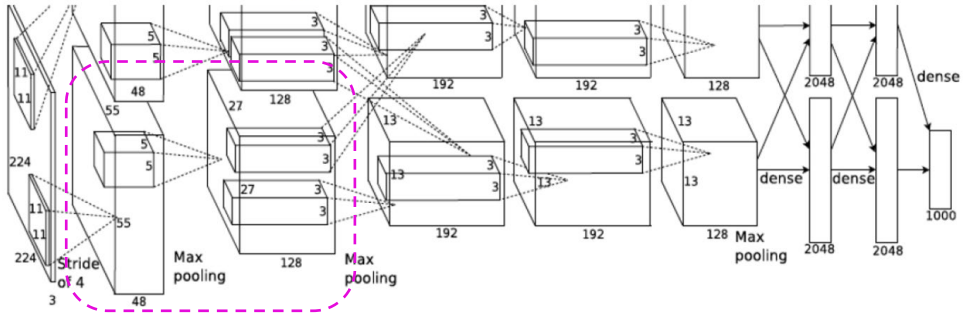
- **Problem:**
  - With patch-level training, the learning algorithm must reconstruct the entire patch with a single feature vector
  - But when the filters are used convolutionally, neighboring feature vectors will be highly redundant

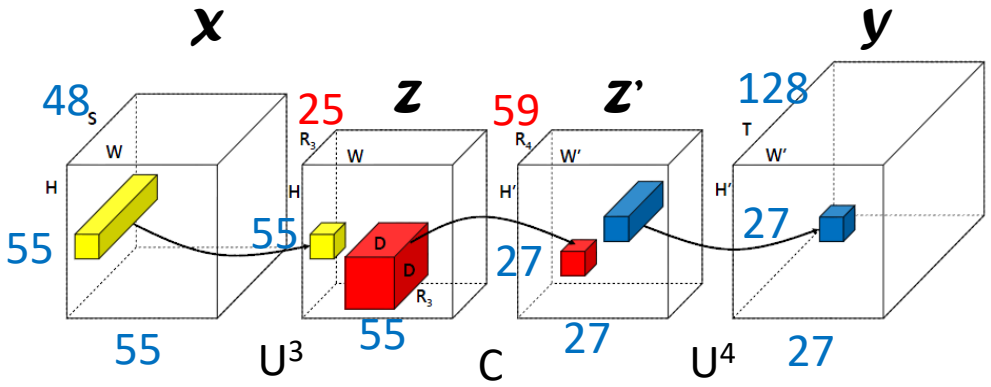
How to remove redundant feature maps?



Yann LeCun

Patch-level training produces lots of filters that are shifted versions of each other.

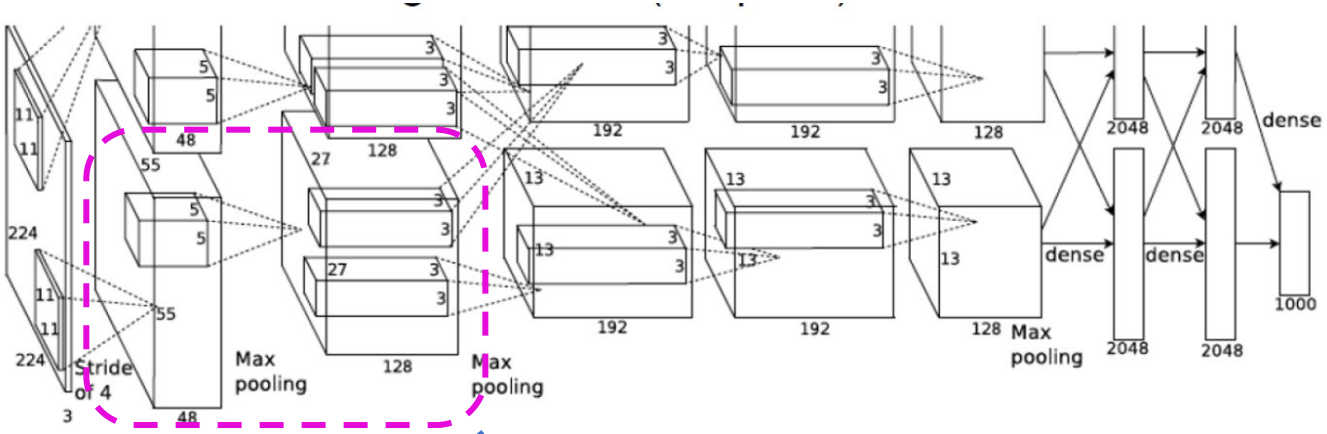


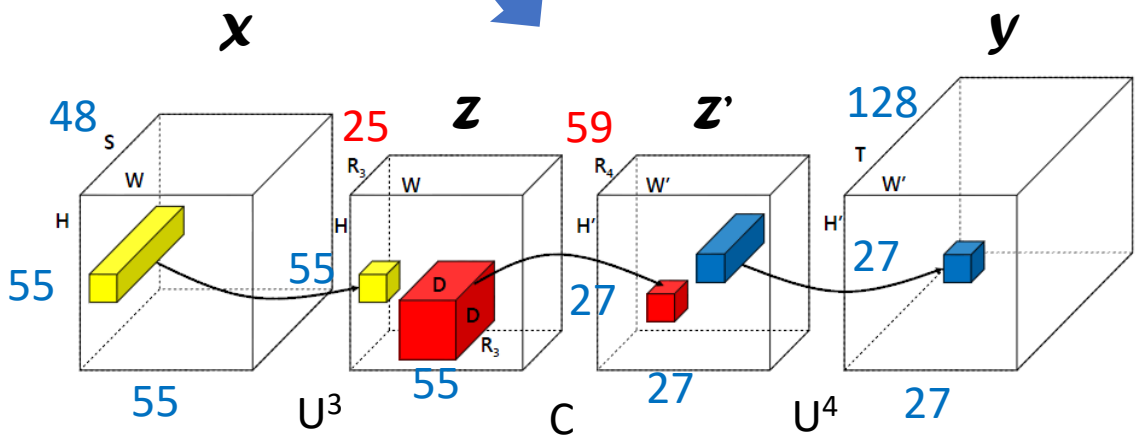


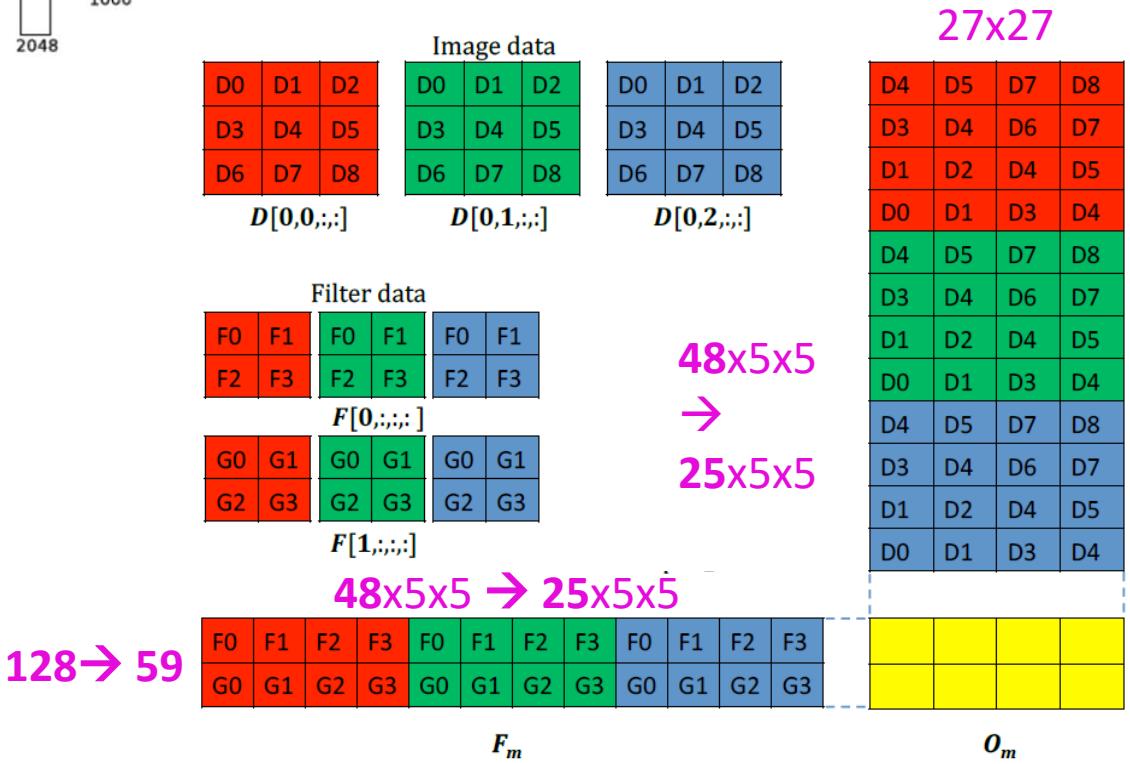
feature maps is reduced at input (48→25 in ***z***) and output (128→59 in ***z'***)

1x1 convolutions are used at both input and output to match with the original layers

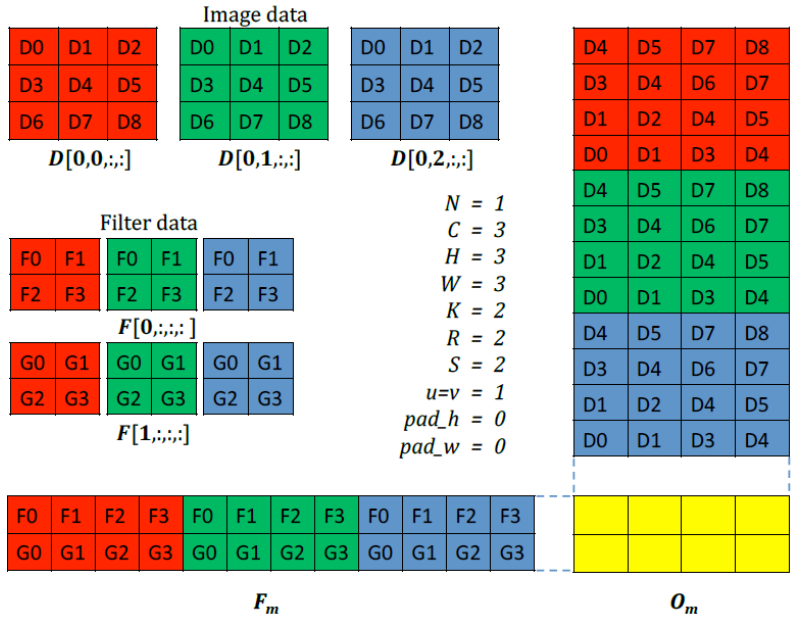
Effect of Low-rank Approximation

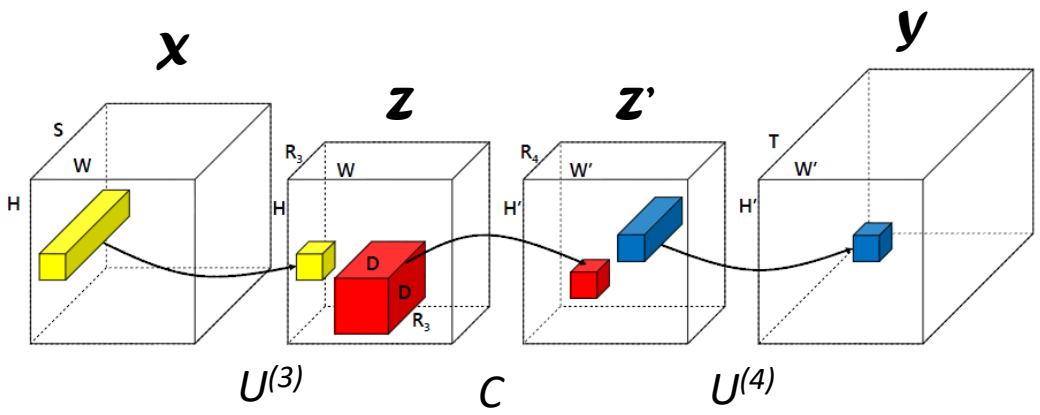






Tucker Decomposition





$$\mathcal{Y}_{h',w',t} = \sum_{i=1}^{D}\sum_{j=1}^{D}\sum_{s=1}^{S} \mathcal{K}_{i,j,s,t}\, \mathcal{X}_{h_i,w_j,s},$$

Tucker decomposition

$$\mathcal{K}_{i,j,s,t} = \sum_{r_3=1}^{R_3}\sum_{r_4=1}^{R_4} \mathcal{C}_{i,j,r_3,r_4}\, U^{(3)}_{s,r_3}\, U^{(4)}_{t,r_4},$$

$$\mathcal{Z}_{h,w,r_3} = \sum_{s=1}^{S} U^{(3)}_{s,r_3}\, \mathcal{X}_{h,w,s},$$

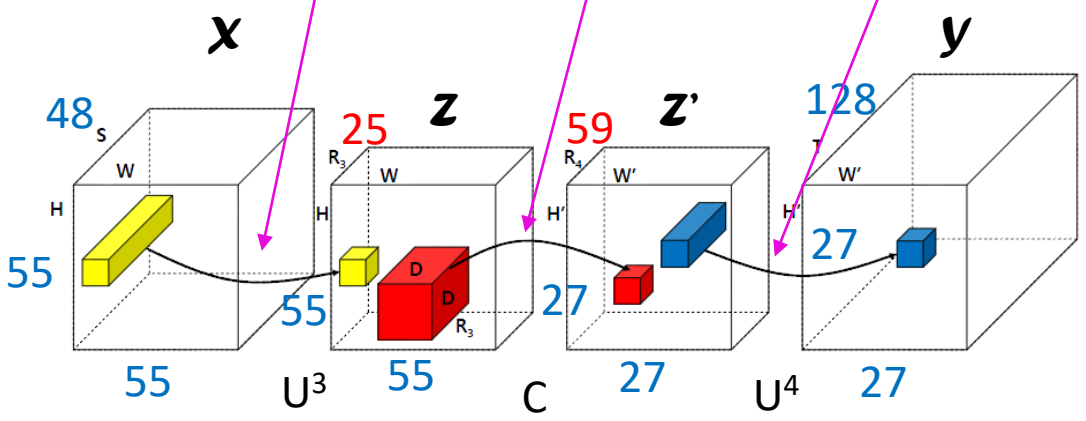
$$\mathcal{Z}'_{h',w',r_4} = \sum_{i=1}^{D}\sum_{j=1}^{D}\sum_{r_3=1}^{R_3} \mathcal{C}_{i,j,r_3,r_4}\, \mathcal{Z}_{h_i,w_j,r_3},$$

$$\mathcal{Y}_{h',w',t} = \sum_{r_4=1}^{R_4} U^{(4)}_{t,r_4}\, \mathcal{Z}'_{h',w',r_4},$$

Reduction in Computation

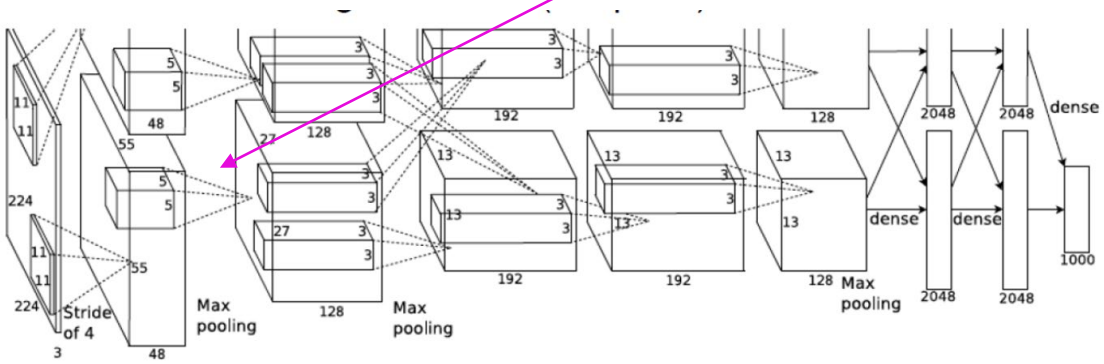
Layer	$S/R3$	T/R_4	Weights	FLOPs	S6
conv1	3	96	35K	105M	15.05 ms
conv1*		26	11K	36M(=29+7)	10.19m(=8.28+1.90)
(imp.)			(×2.92)	(×2.92)	(×1.48)
conv2	48 × 2	128 × 2	307K	224M	24.25 ms
conv2*	25 × 2	59 × 2	91K	67M(=2+54+11)	10.53ms(=0.80+7.43+2.30)
(imp.)			(×3.37)	(×3.37)	(×2.30)
conv3	256	384	885K	150M	18.60ms
conv3*	105	112	178K	30M(=5+18+7)	4.85ms(=1.00+2.72+1.13)
(imp.)			(×5.03)	(×5.03)	(×3.84)
conv4	192 × 2	192 × 2	664K	112M	15.17ms
conv4*	49 × 2	46 × 2	77K	13M(=3+7+3)	4.29 ms(=1.55+1.89+0.86)
(imp.)			(×7.10)	(×7.10)	(×3.53)
conv5	192 × 2	128 × 2	442K	75.0M	10.78ms
conv5*	40 × 2	34 × 2	49K	8.2M(=2.6+4.1+1.5)	3.44 ms(=1.15+1.61+0.68)
(imp.)			(×9.11)	(×9.11)	(×3.13)
fc6	256	4096	37.7M	37.7M	18.94ms
fc6*	210	584	6.9M	8.7M(=1.9+4.4+2.4)	5.07 ms(=0.85+3.12+1.11)
(imp.)			(×8.03)	(×4.86)	(×3.74)
fc7	4096	4096	16.8M	16.8M	7.75ms
fc7*		301	2.4M	2.4M(=1.2+1.2)	1.02 ms(=0.51+0.51)
(imp.)			(×6.80)	(×6.80)	(×7.61)
fc8	4096	1000	4.1M	4.1M	2.00ms
fc8*		195	1.0M	1.0M(=0.8+0.2)	0.66ms(=0.44+0.22)
(imp.)			(×4.12)	(×4.12)	(×3.01)

48x**25**x55x55+**25**x5x5x**59**x27x27+**59**x128x27x27



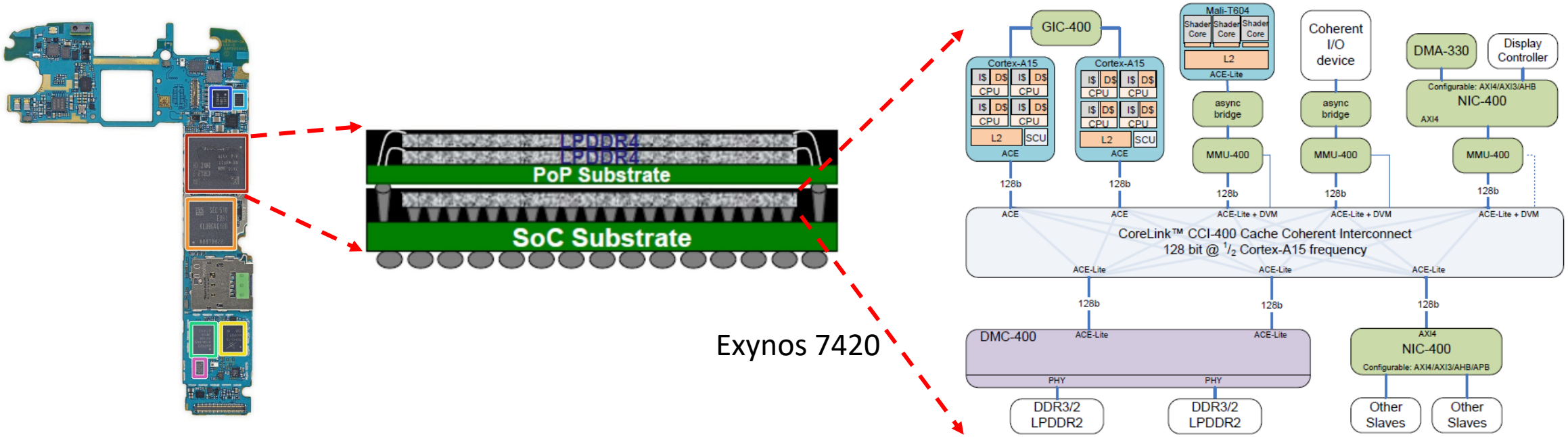
3.1X less computation!

Conv2: # multiplications = 48x5x5x128x27x27



Evaluating CNNs on Samsung Galaxy S6

- Exynos 7420 + LPDDR4
  - ARM Mali T760, 190Gflops for 8 cores, max 256 threads/core, 32KB L1 cache/core, 1MB shared L2 cache, 25.6GB/s LPDDR4 with four x16 channels
  - Comparison: nVidia Titan X provides 6.6TFlops for 24 cores, 336GB/s main memory, max 2k threads/core, 64KB L1 cache/core, 3MB shared L2 cache



Results

- Significant reductions in energy consumption and runtime
  - Energy: x4.26~x1.6
  - Runtime:x3.68~x1.42

Model	Top-5	Weights	FLOPs	S6		Titan X
AlexNet	80.03	61M	725M	117ms	245mJ	0.54ms
*AlexNet**	78.33	11M	272M	43ms	72mJ	0.30ms
(imp.)	(-1.70)	(×5.46)	(×2.67)	(×2.72)	(×3.41)	(×1.81)
VGG-S	84.60	103M	2640M	357ms	825mJ	1.86ms
*VGG-S**	84.05	14M	549M	97ms	193mJ	0.92ms
(imp.)	(-0.55)	(×7.40)	(×4.80)	(×3.68)	(×4.26)	(×2.01)
GoogLeNet	88.90	6.9M	1566M	273ms	473mJ	1.83ms
*GoogLeNet**	88.66	4.7M	760M	192ms	296mJ	1.48ms
(imp.)	(-0.24)	(×1.28)	(×2.06)	(×1.42)	(×1.60)	(×1.23)
VGG-16	89.90	138M	15484M	1926ms	4757mJ	10.67ms
*VGG-16**	89.40	127M	3139M	576ms	1346mJ	4.58ms
(imp.)	(-0.50)	(×1.09)	(×4.93)	(×3.34)	(×3.53)	(×2.33)

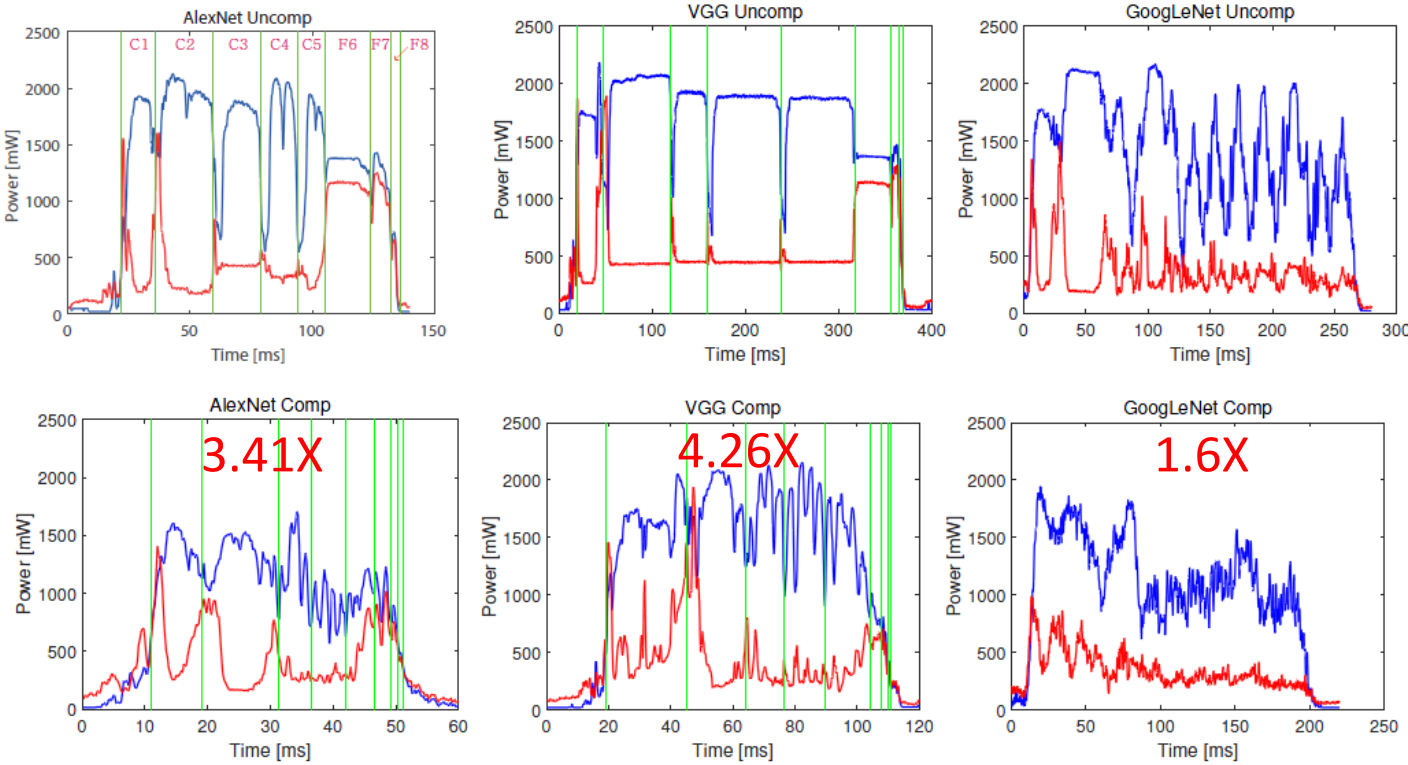


Figure 5: Power consumption over time for each model. (Blue: GPU, Red: main memory).

Example of Truncated SVD: $A \sim USV^T$

$$U = \begin{bmatrix} -0.54 & 0.07 & 0.82 & -0.11 & 0.12 \\ -0.10 & -0.59 & -0.11 & -0.79 & -0.06 \\ -0.53 & 0.06 & -0.21 & 0.12 & -0.81 \\ -0.65 & 0.07 & -0.51 & 0.06 & 0.56 \\ -0.06 & -0.80 & 0.09 & 0.59 & 0.04 \end{bmatrix}$$

$$\lambda = 321.07, \lambda = 230.17, \lambda = 12.70, \lambda = 3.94, \lambda = 0.12$$

$$V^T = \begin{bmatrix} -0.46 & 0.02 & -0.87 & -0.00 & 0.17 \\ -0.07 & -0.76 & 0.06 & 0.60 & 0.23 \\ -0.74 & 0.10 & 0.28 & 0.22 & -0.56 \\ -0.48 & 0.03 & 0.40 & -0.33 & 0.70 \\ -0.07 & -0.64 & -0.04 & -0.69 & -0.32 \end{bmatrix}$$

Take 3 eigenvectors associated with the selected eigenvalues

Take 3 largest square roots of eigenvalues

Take 3 eigenvectors associated with the selected eigenvalues

$$\hat{A} = \begin{bmatrix} -0.54 & 0.07 & 0.82 \\ -0.10 & -0.59 & -0.11 \\ -0.53 & 0.06 & -0.21 \\ -0.65 & 0.07 & -0.51 \\ -0.06 & -0.80 & 0.09 \end{bmatrix} \begin{bmatrix} 17.92 & 0 & 0 \\ 0 & 15.17 & 0 \\ 0 & 0 & 3.56 \end{bmatrix} \begin{bmatrix} -0.46 & 0.02 & -0.87 & -0.00 & 0.17 \\ -0.07 & -0.76 & 0.06 & 0.60 & 0.23 \\ -0.74 & 0.10 & 0.28 & 0.22 & -0.56 \end{bmatrix}$$

Error degrades accuracy. How to reclaim lost accuracy?

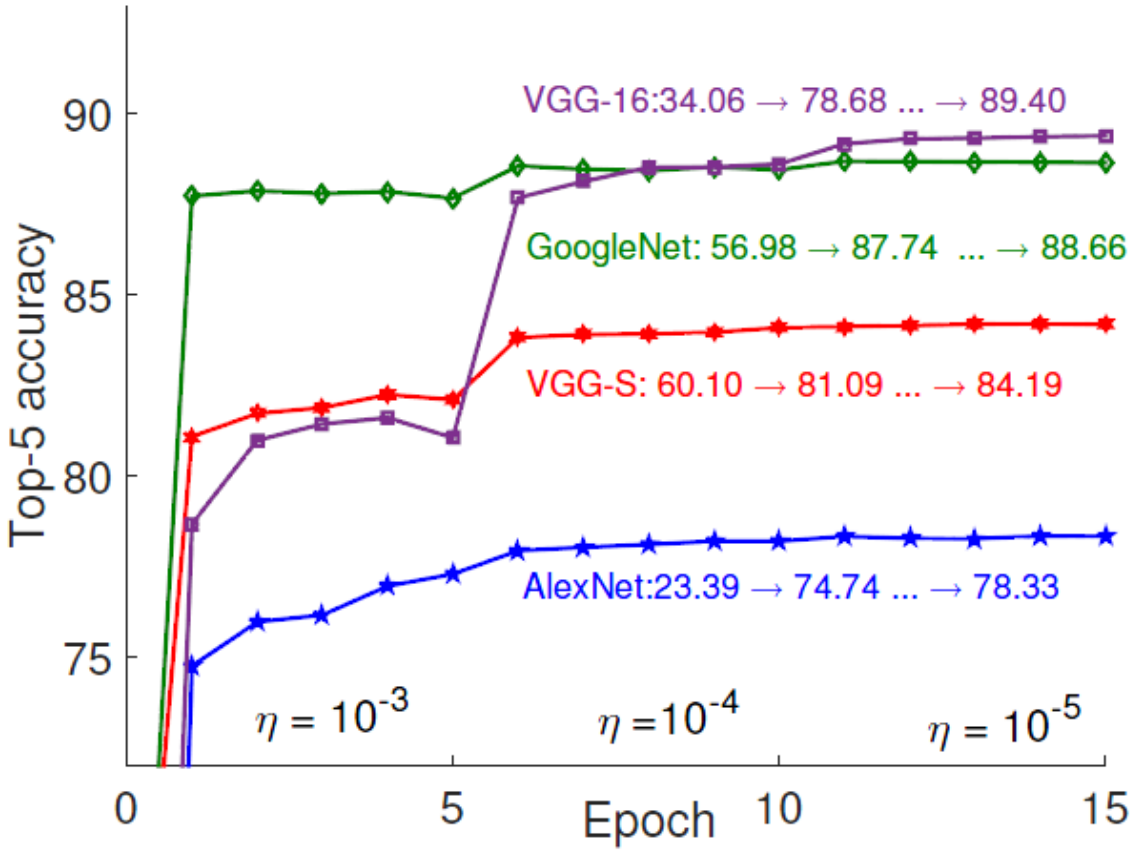
$$= \begin{bmatrix} 2.29 & -0.66 & 9.33 & 1.25 & -3.09 \\ 1.77 & 6.76 & 0.90 & -5.50 & -2.13 \\ 4.86 & -0.96 & 8.01 & 0.38 & -0.97 \\ 6.62 & -1.23 & 9.58 & 0.24 & -0.71 \\ 1.14 & 9.19 & 0.33 & -7.19 & -3.13 \end{bmatrix} \Leftrightarrow A = \begin{bmatrix} 2 & 0 & 8 & 6 & 0 \\ 1 & 6 & 0 & 1 & 7 \\ 5 & 0 & 7 & 4 & 0 \\ 7 & 0 & 8 & 5 & 0 \\ 0 & 10 & 0 & 0 & 7 \end{bmatrix}$$

Fine-tuning

- Low-rank approximation loses accuracy
- Fine-tuning recovers lost error
  - 1 epoch: 1 run of back propagation with the entire training set

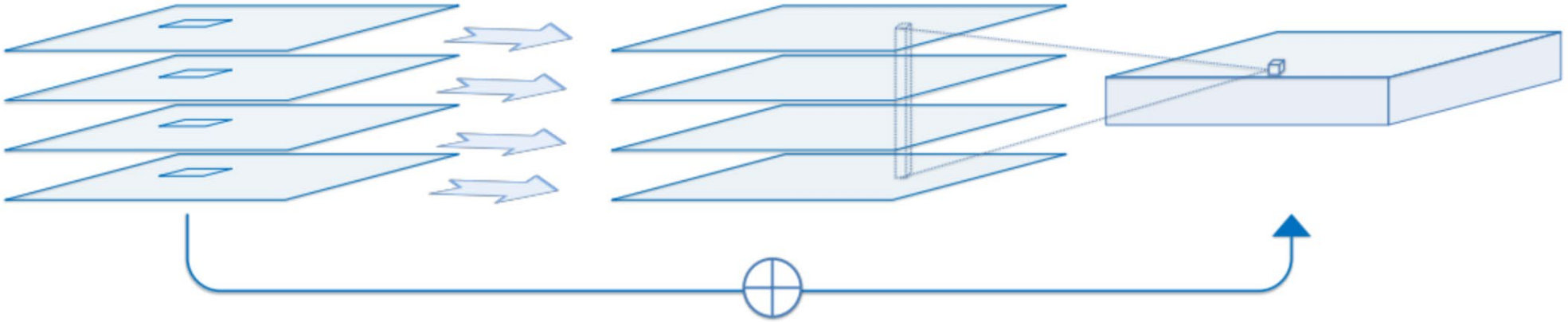
$$\hat{A} = \begin{bmatrix} -0.54 & 0.07 & 0.82 \\ -0.10 & -0.59 & -0.11 \\ -0.53 & 0.06 & -0.21 \\ -0.65 & 0.07 & -0.51 \\ -0.06 & -0.80 & 0.09 \end{bmatrix} \begin{bmatrix} 17.92 & 0 & 0 \\ 0 & 15.17 & 0 \\ 0 & 0 & 3.56 \end{bmatrix} \begin{bmatrix} -0.46 & 0.02 & -0.87 & -0.00 & 0.17 \\ -0.07 & -0.76 & 0.06 & 0.60 & 0.23 \\ -0.74 & 0.10 & 0.28 & 0.22 & -0.56 \end{bmatrix}$$

$$= \begin{bmatrix} 2.29 & -0.66 & 9.33 & 1.25 & -3.09 \\ 1.77 & 6.76 & 0.90 & -5.50 & -2.13 \\ 4.86 & -0.96 & 8.01 & 0.38 & -0.97 \\ 6.62 & -1.23 & 9.58 & 0.24 & -0.71 \\ 1.14 & 9.19 & 0.33 & -7.19 & -3.13 \end{bmatrix} \Leftrightarrow A = \begin{bmatrix} 2 & 0 & 8 & 6 & 0 \\ 1 & 6 & 0 & 1 & 7 \\ 5 & 0 & 7 & 4 & 0 \\ 7 & 0 & 8 & 5 & 0 \\ 0 & 10 & 0 & 0 & 7 \end{bmatrix}$$

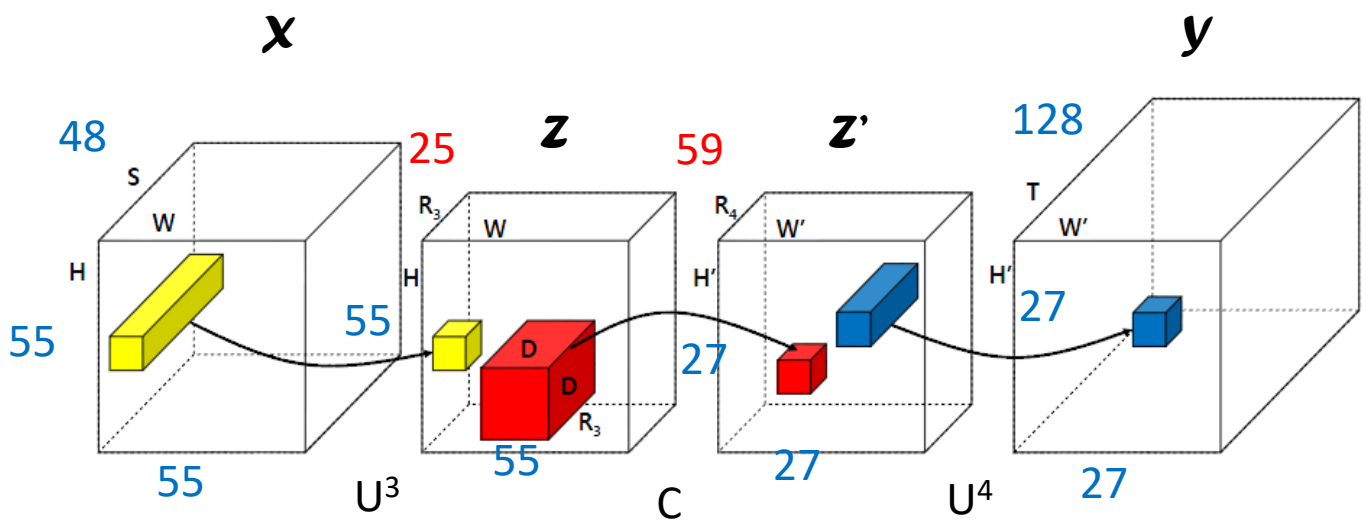


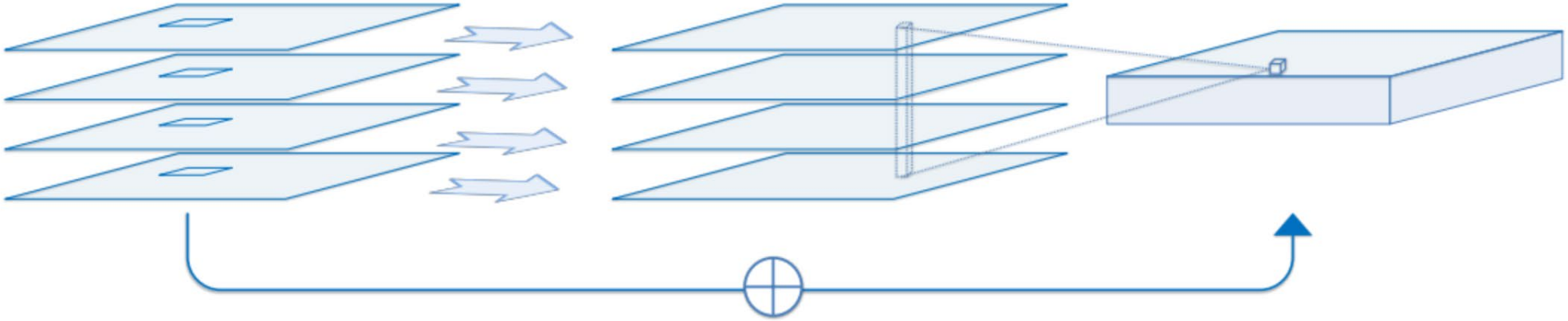
Google's MobileNet

- Depth separable convolution
  - 3D convolution → 2D convolution on each input feature map + 1x1 convolution to produce output



Comparison: LRA vs Depth Separable Conv.



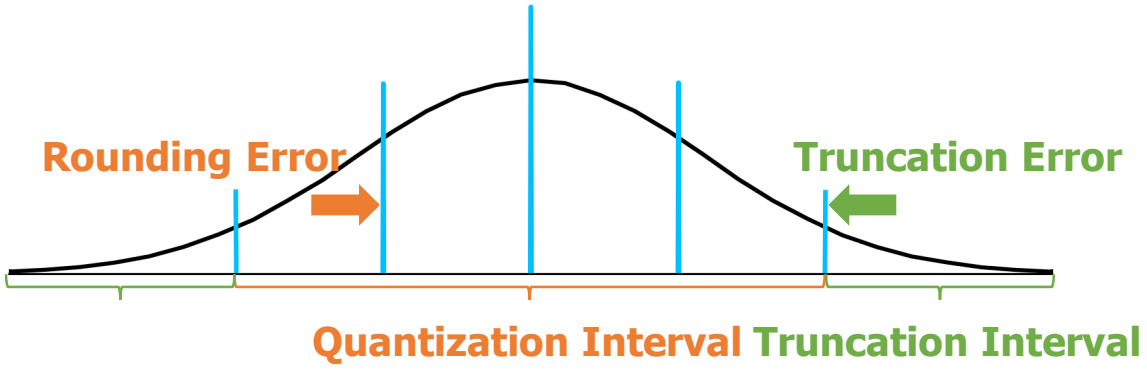


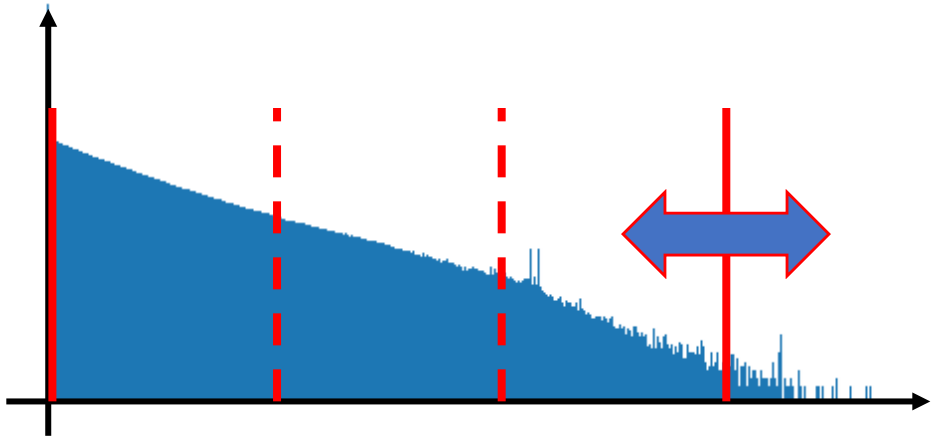
HAWQ-V2: Hessian Aware trace-Weighted Quantization of Neural Networks

Hyper-parameters of Quantization

- Hyperparameters of linear quantization for activation/weight
  - Quantization bit-width (quantization level)
  - Quantization boundary or clipping threshold

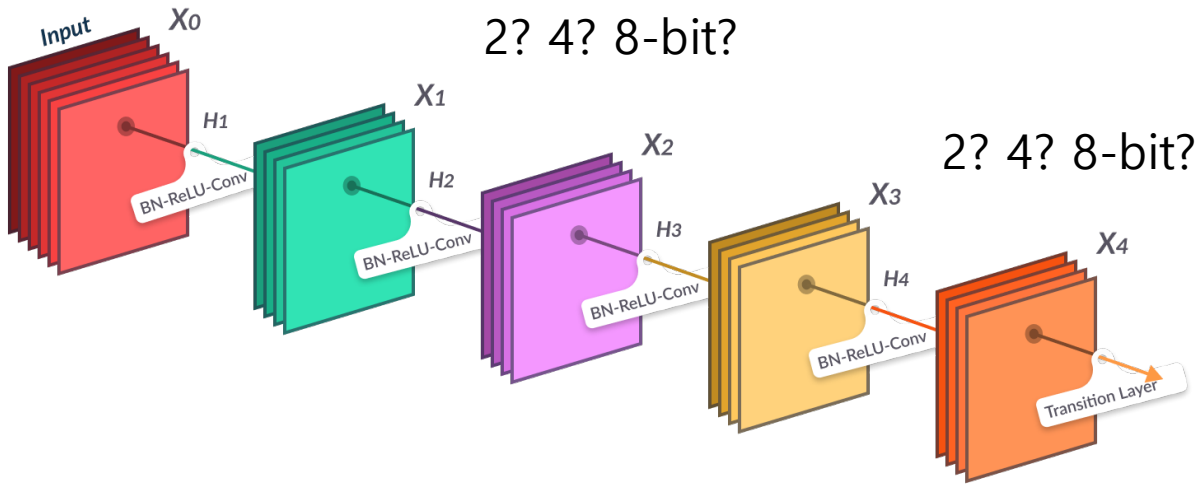
  - Conventionally use L2/L1 norm-based scaling or predefined value





Layer-wise Bit-width Assignment

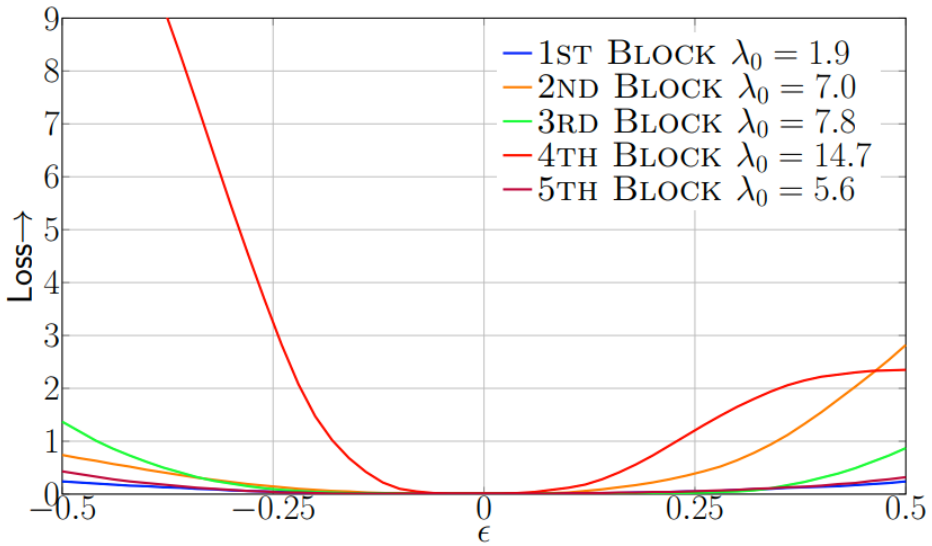
- Need to assign bit-width for each convolution/linear layer
  - Each layer has different quantization sensitivity
  - Search space is increased exponentially proportional to the number of layers
  - Difficulty is hard to measure
    - Post-training quantization is time-consuming
- We need to develop more efficient estimation metric!

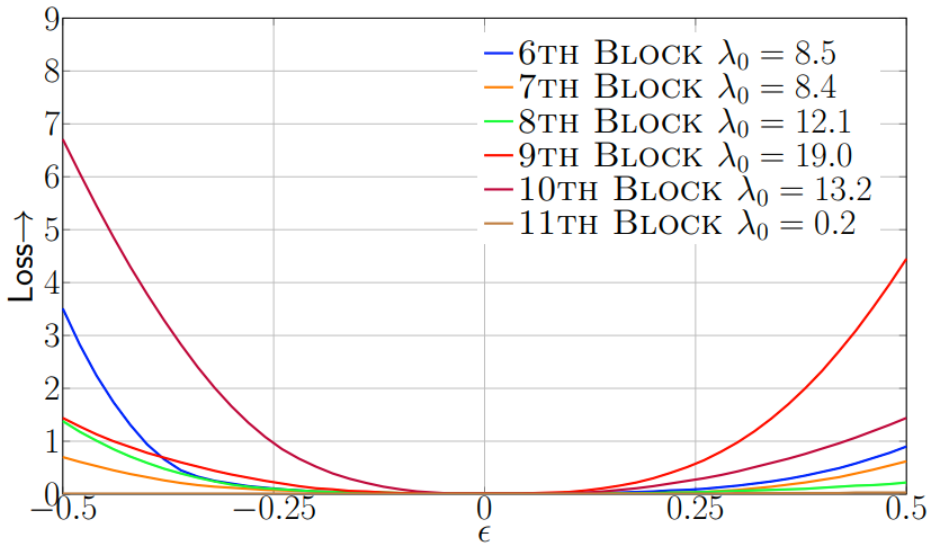


Hessian-Trace Aware Quantization

$$\mathbf{H}f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

- Basic Motivation
  - Assume that the network is fully trained
    - Weights of the network should be located near the optimal point
      - First derivative of the weights are close to zero
    - If then, the sensitivity of the weights can be measured through 2nd-order derivative
      - Assign bit-width considering the 2nd order derivative (Hessian-trace)





Formulation of Problem

Assumption 1 *Assume that:*

- *The model is twice differentiable and has converged to a local minima such that the first and second order optimality conditions are satisfied, i.e., the gradient is zero and the Hessian is positive semi-definite.*
- *If we denote the Hessian of the i^{th} layer as H_i, and its corresponding orthonormal eigenvectors as v_1^i, $v_2^i, ..., v_{n_i}^i$, then the quantization-aware fine-tuning perturbation, $\Delta W_i^* = \arg\min_{W_i^* + \Delta W_i^* \in Q(\cdot)} L(W_i^* + \Delta W_i^*)$, satisfies*

$$\Delta W_i^* = \alpha_{bit} v_1^i + \alpha_{bit} v_2^i + ... + \alpha_{bit} v_{n_i}^i. \tag{2.3}$$

Here, n_i is the dimension of W_i, W_i^* is the converging point of i^{th} layer, and $Q(\cdot)$ is the quantization function which maps floating point values to reduced precision values. Note that α_{bit} is a constant number based on the precision setting and quantization range.

Hessian Trace can be a Metric for Sensitivity

Lemma 1 *Suppose we quantize two layers (denoted by B_1 and B_2) with same amount of perturbation, namely $\|\Delta W_1^*\|_2^2 = \|\Delta W_2^*\|_2^2$. Then, under Assumption 1, we will have:*

$$\mathcal{L}(W_1^* + \Delta W_1^*, W_2^*, \cdots, W_L^*) \leq \mathcal{L}(W_1^*, W_2^* + \Delta W_2^*, W_3^*, \cdots, W_L^*), \quad (2.4)$$

if

$$\frac{1}{n_1} Tr(\nabla_{W_1}^2 \mathcal{L}(W_1^*)) \leq \frac{1}{n_2} Tr(\nabla_{W_2}^2 \mathcal{L}(W_2^*)). \quad (2.5)$$

- Proof
  - Based on the taylor's expansion, the loss value can be approximated as

$$\mathcal{L}(W_1^* + \Delta W_1^*) = \mathcal{L}(W_1^*) + g_1^T \Delta W_1^* + \frac{1}{2} \Delta W_1^{*T} H_1 \Delta W_1^* = \mathcal{L}(W_1^*) + \frac{1}{2} \Delta W_1^{*T} H_1 \Delta W_1^*.$$

Hessian Trace can be a Metric for Sensitivity

Lemma 1 *Suppose we quantize two layers (denoted by B_1 and B_2) with same amount of perturbation, namely $\|\Delta W_1^*\|_2^2 = \|\Delta W_2^*\|_2^2$. Then, under Assumption 1, we will have:*

$$\mathcal{L}(W_1^* + \Delta W_1^*, W_2^*, \cdots, W_L^*) \leq \mathcal{L}(W_1^*, W_2^* + \Delta W_2^*, W_3^*, \cdots, W_L^*), \quad (2.4)$$

[1]

if

$$\frac{1}{n_1} Tr(\nabla^2_{W_1} \mathcal{L}(W_1^*)) \leq \frac{1}{n_2} Tr(\nabla^2_{W_2} \mathcal{L}(W_2^*)). \quad (2.5)$$

- Proof
  - Because $\Delta W_i^* = \alpha_{bit} v_1^i + \alpha_{bit} v_2^i + \ldots + \alpha_{bit} v_{n_i}^i$, $\Delta {W_1^*}^T H_1 \Delta W_1^* = \sum_{i=1}^{n_1} \alpha_{bit,1}^2 {v_i^1}^T H_1 v_i^1 = \alpha_{bit,1}^2 \sum_{i=1}^{n_1} \lambda_i^1,$
  - Now $\mathcal{L}(W_1^*) = \mathcal{L}(W_2^*)$, and $\sqrt{n_1}\alpha_{bit,1} = \sqrt{n_2}\alpha_{bit,2}$ (because $\|\Delta W_1^*\|_2 = \|\Delta W_2^*\|_2,$)
  - Therefore,

$$\mathcal{L}(W_2^* + \Delta W_2^*) - \mathcal{L}(W_1^* + \Delta W_1^*) = \alpha_{bit,2}^2 n_2 \left(\frac{1}{n_2} \sum_{i=1}^{n_2} \lambda_i^2 - \frac{1}{n_1} \sum_{i=1}^{n_1} \lambda_i^1\right) \geq 0.$$

How to Measure 2nd-order Information

- Explicit computation for the hessian is expensive and requires large memory
- Instead, using a matrix-free method
  - Assume that $z \in R^d$ is a random vector sampled from gaussian distribution or Rademacher distribution

$$Tr(H) = Tr(HI) = Tr(H\,\mathbb{E}[zz^T]) = \mathbb{E}[Tr(Hzz^T)] = \mathbb{E}[z^T Hz],$$

- Based on this equation, Hutchinson algorithm can be used to estimate the hessian trace

$$Tr(H) \approx \frac{1}{m}\sum_{i=1}^{m} z_i^T H z_i = Tr_{Est}(H).$$

How to Calculate $E[z^T Hz]$

- From https://arxiv.org/pdf/1905.03696.pdf (HAWQ original)
- Denote g_i as the gradient of loss L with respect to the i^{th} block parameters
  - $g_i = \frac{\partial L}{\partial W_i}$
- For a random vector v,
  - $\frac{\partial (g_i^T v)}{\partial W_i} = \frac{\partial g_i^T}{\partial W_i} v + g_i^T \frac{\partial v}{\partial W_i} = \frac{\partial g_i^T}{\partial W_i} v = H_i v$
- Based on this gradient vector, we can get $E[v^T Hv]$

Pytorch's Autograd.grad

- Getting 2nd derivative
  - https://discuss.pytorch.org/t/second-order-derivatives-of-loss-function/71797/3

```
import torch
x = torch.tensor(1., requires_grad = True)
loss = 2*x**3 + 5*x**2 + 8
first_derivative = torch.autograd.grad(loss, x, create_graph=True)
# We now have dloss/dx
second_derivative = torch.autograd.grad(first_derivative, x)
# This computes d/dx(dloss/dx) = d2loss/dx2
```

Weight Precision Selection

- Main idea
  - Sort each candidate bit-precision setting based on the total second-order perturbation

$$\Omega = \sum_{i=1}^{L} \Omega_i = \sum_{i=1}^{L} \overline{Tr}(H_i) \cdot \|Q(W_i) - W_i\|_2^2,$$

  - Given a target model size, we sort the elements of B based on their Ω value, and choose the bit precision setting with minimal Ω.

Results

- Inception-v3 / ResNet-50 results

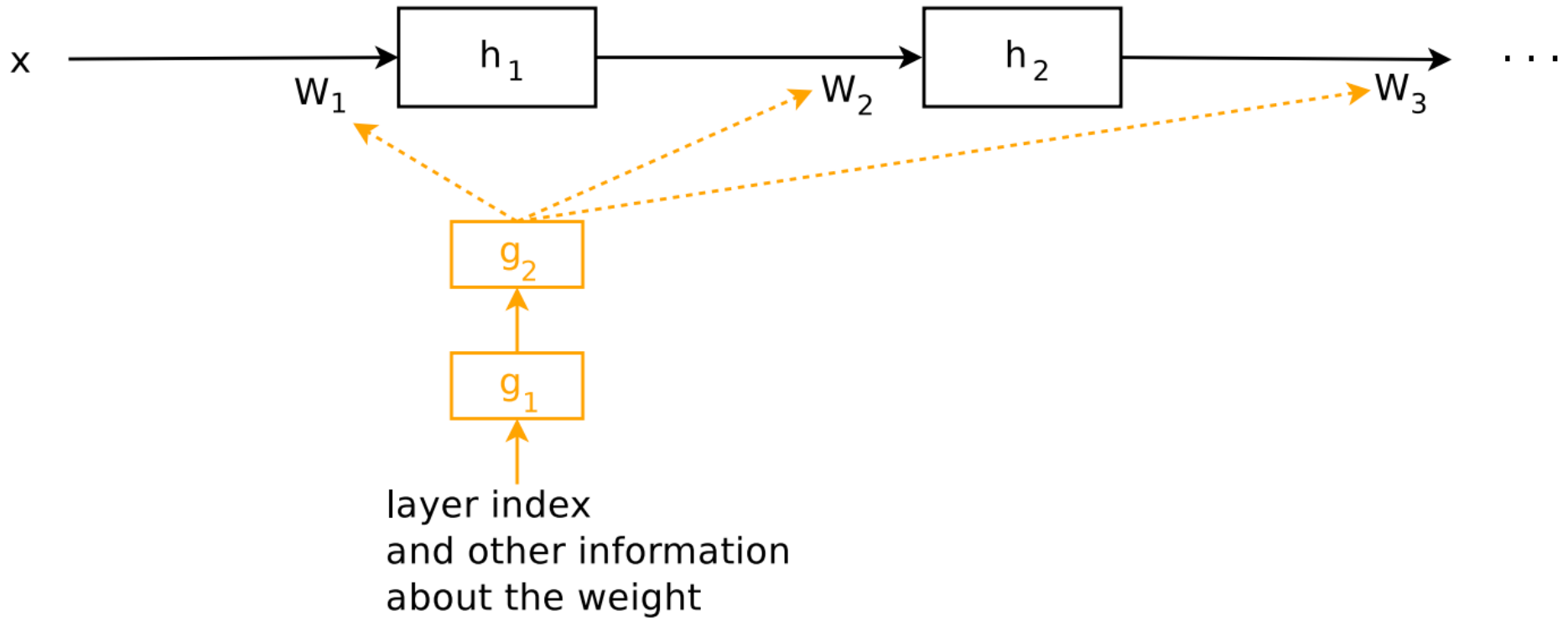
Method	w-bits	a-bits	Top-1	W-Comp	Size(MB)
Baseline	32	32	77.45	1.00×	91.2
Integer-Only [11]	8	8	75.40	4.00×	22.8
Integer-Only [11]	7	7	75.00	4.57×	20.0
RVQuant [18]	3 MP	3 MP	74.14	10.67×	8.55
Direct	2 MP	4 MP	69.76	15.88×	5.74
HAWQ [7]	2 MP	4 MP	75.52	12.04×	**7.57**
HAWQ-V2	2 MP	4 MP	**75.68**	12.04×	**7.57**

Method	w-bits	a-bits	Top-1	W-Comp	Size(MB)
Baseline	32	32	77.39	1.00×	97.8
Dorefa [30]	2	2	67.10	16.00×	6.11
Dorefa [30]	3	3	69.90	10.67×	9.17
PACT [5]	2	2	72.20	16.00×	6.11
PACT [5]	3	3	75.30	10.67×	9.17
LQ-Nets [28]	3	3	74.20	10.67×	9.17
Deep Comp. [9]	3	MP	75.10	10.41×	9.36
HAQ [22]	MP	MP	75.30	10.57×	9.22
HAWQ [7]	2 MP	4 MP	75.48	12.28×	7.96
HAWQ-V2	2 MP	4 MP	**75.76**	12.24×	**7.99**

MetaPruning: Meta Learning for Automatic Neural Network Channel Pruning

HyperNetwork

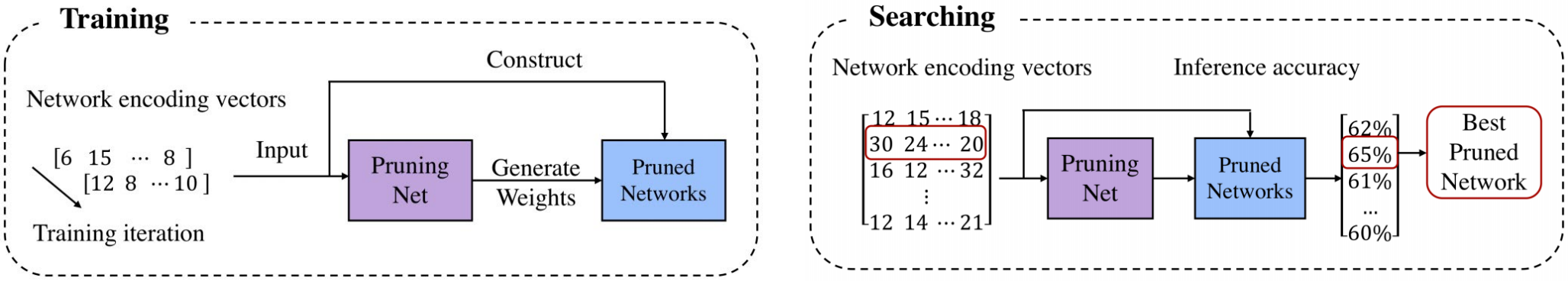
- We can design a network predicting weight for the different network



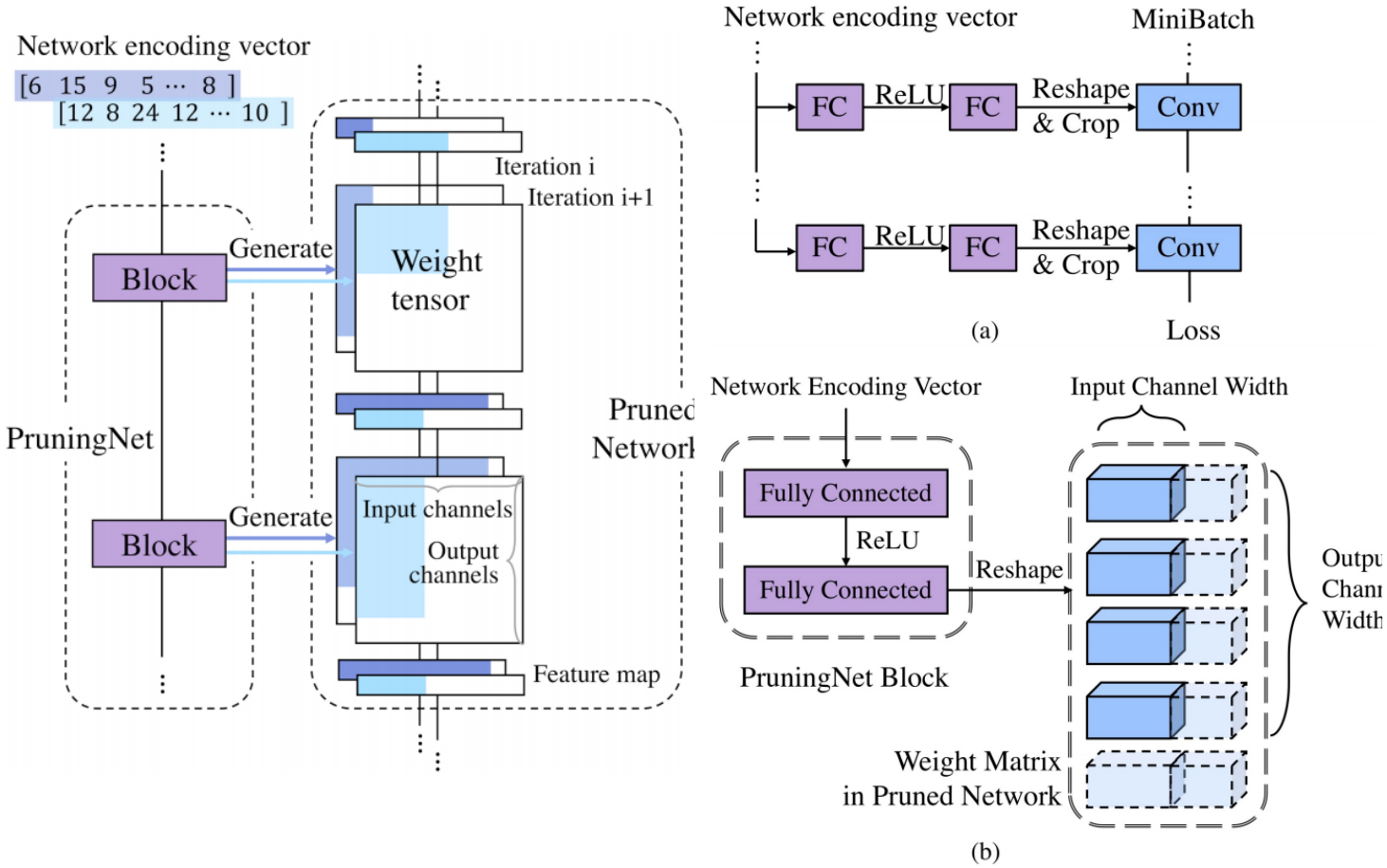
Motivation of MetaPruning

- In order to find out the optimal pruning strategy, we need to evaluate multiple **networks having different channel constraints**
- However, **the optimal weights should be different depending on the network** structure, it is practically impossible

- In MetaPruning, the idea is to train an meta network (PruningNet) that predicts the weight of the pruned network having divergent structural characteristics



2-step Search with Evolutionary Algorithm



Algorithm 1 Evolutionary Search Algorithm

Hyper Parameters: Population Size: $\mathcal{P}$, Number of Mutation: $\mathcal{M}$, Number of Crossover: $\mathcal{S}$, Max Number of Iterations: $\mathcal{N}$.

Input: PruningNet: $PruningNet$, Constraints: $\mathcal{C}$.

Output: Most accurate gene: $\mathcal{G}_{top}$.

$\mathcal{G}_0$ = Random($\mathcal{P}$), s.t. $\mathcal{C}$;
$\mathcal{G}_{topK} = \emptyset$;
for $i = 0 : \mathcal{N}$ **do**
  $\{\mathcal{G}_i, \text{accuracy}\}$ = Inference($PruningNet(\mathcal{G}_i)$);
  $\mathcal{G}_{topK}, accuracy_{topK}$ = TopK($\{\mathcal{G}_i, accuracy\}$);
  $\mathcal{G}_{mutation}$ = Mutation($\mathcal{G}_{topK}, \mathcal{M}$), s.t. $\mathcal{C}$;
  $\mathcal{G}_{crossover}$ = Crossover($\mathcal{G}_{topK}, \mathcal{S}$), s.t. $\mathcal{C}$;
  $\mathcal{G}_i = \mathcal{G}_{mutation} + \mathcal{G}_{crossover}$;
end for
$\mathcal{G}_{top1}, accuracy_{top1}$= Top1($\{\mathcal{G}_{\mathcal{N}}, accuracy\}$);
return $\mathcal{G}_{top1}$;

Results

Table 1. This table compares the top-1 accuracy of MetaPru method with the uniform baselines on MobileNet V1 [24].

Uniform Baselines			MetaPruning	
Ratio	Top1-Acc	FLOPs	Top1-Acc	FLOPs
1×	70.6%	569M	–	–
0.75×	68.4%	325M	**70.9%**	324M
0.5×	63.7%	149M	**66.1%**	149M
0.25×	50.6%	41M	**57.2%**	41M

Table 2. This table compares the top-1 accuracy of MetaPruning method with the uniform baselines on MobileNet V2 [46]. MobileNet V2 only reports the accuracy with 585M and 300M FLOPs, so we apply the uniform pruning method on MobileNet V2 to obtain the baseline accuracy for networks with other FLOPs.

Uniform Baselines		MetaPruning	
Top1-Acc	FLOPs	Top1-Acc	FLOPs
74.7%	585M	–	–
72.0%	313M	**72.7%**	291M
67.2%	140M	**68.2%**	140M
66.5%	127M	**67.3%**	124M
64.0%	106M	**65.0%**	105M
62.1%	87M	**63.8%**	84M
54.6%	43M	**58.3%**	43M

Table 4. This table compares the top-1 accuracy of MetaPruning method with other state-of-the-art AutoML-based methods.

Network	FLOPs	Top1-Acc
0.75x MobileNet V1 [24]	325M	68.4%
NetAdapt [52]	284M	69.1%
AMC [21]	285M	70.5%
MetaPruning	281M	**70.6%**
0.75x MobileNet V2 [46]	220M	69.8%
AMC [21]	220M	70.8%
MetaPruning	217M	**71.2%**

Results

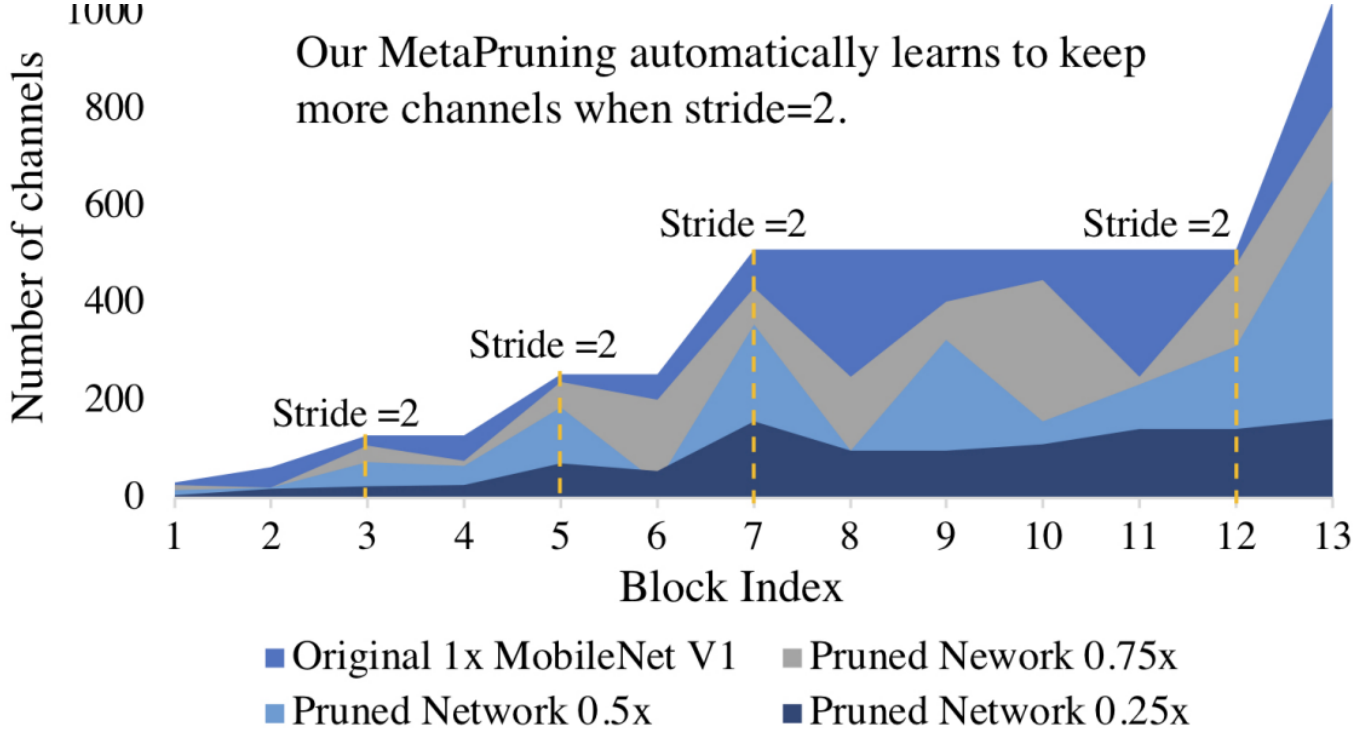


Figure 5. This figure presents the number of output channels of each block of the pruned MobileNet v1. Each block contains a 3x3 depth-wise convolution followed by a 1x1 point-wise convolution, except the first block is composed by a 3x3 convolution only.

Results

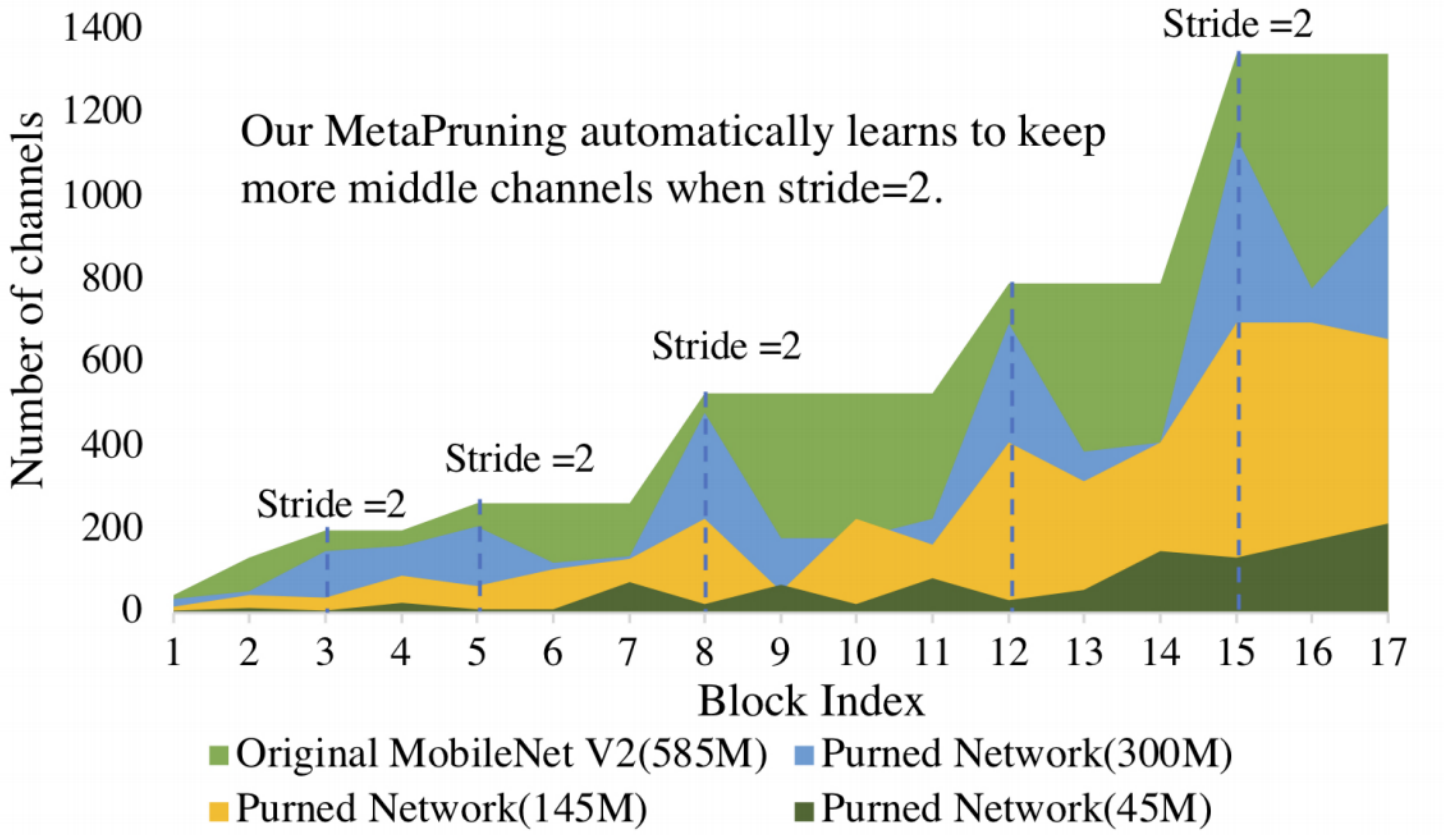


Figure 6. A MobileNet V2 block is constructed by concatenating a 1x1 point-wise convolution, a 3x3 depth-wise convolution and a 1x1 point-wise convolution. This figure illustrates the number of middle channels of each block.

Results

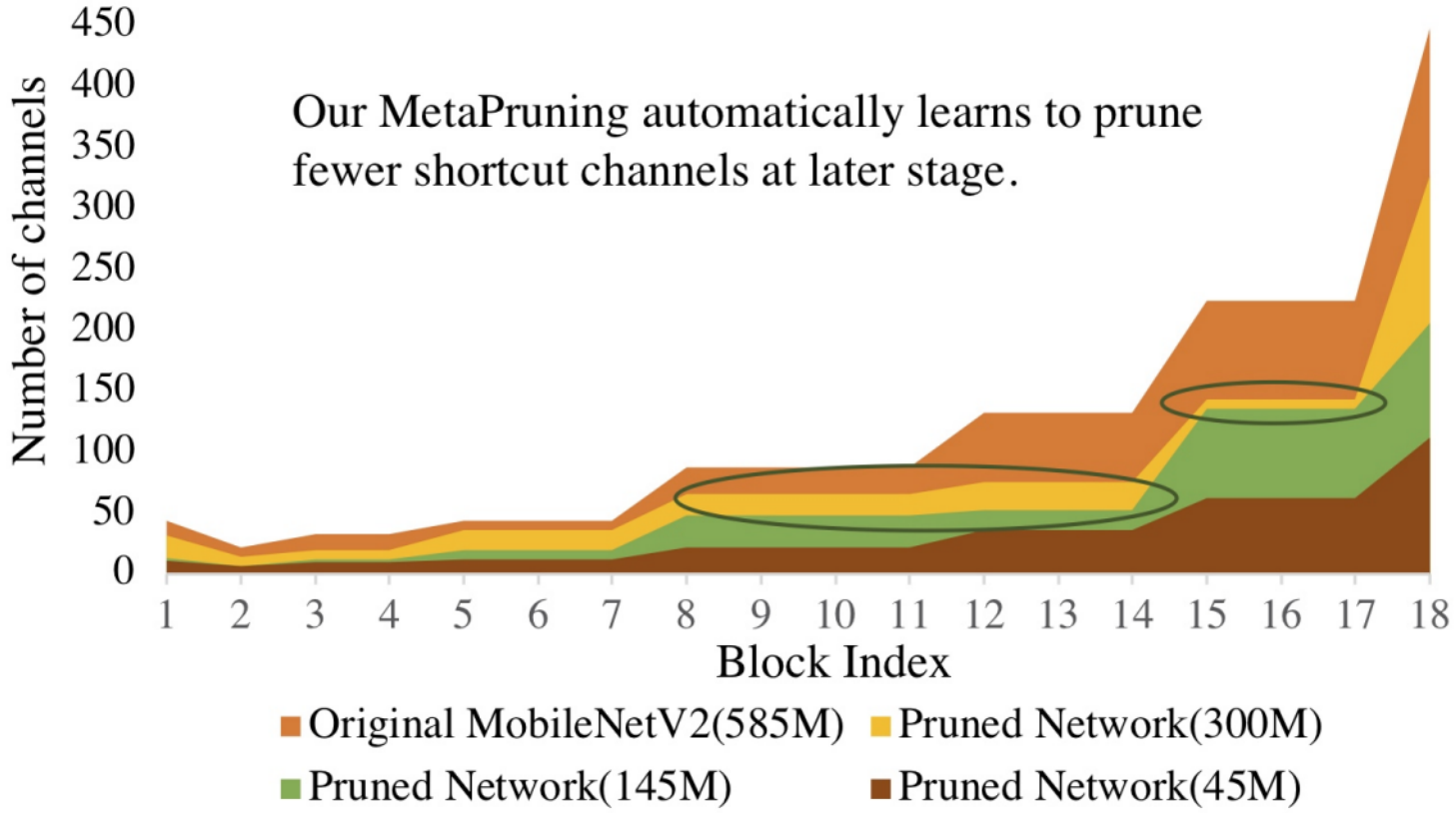


Figure 7. In MobileNet V2, each stage starts with a bottleneck block with differed input and output channels and followed by several repeated bottleneck blocks. Those bottleneck blocks with the same input and output channels are connected with a shortcut. MetaPruning prunes the channels in the shortcut jointly with the middle channels. This figure illustrates the number of shortcut channel in each stage after being pruned by the MetaPruning.