Compressing Convolutional Neural Network
for High Performance and Low Power Consumption

Yongdeok Kim*, Eunhyeok Park, Sungjoo Yoo, Taelim Choi*, Lu Yang*
and Dongjun Shin*

Software Lab, Samsung Electronics*

Computing Memory Architecture Lab., CSE, SNU

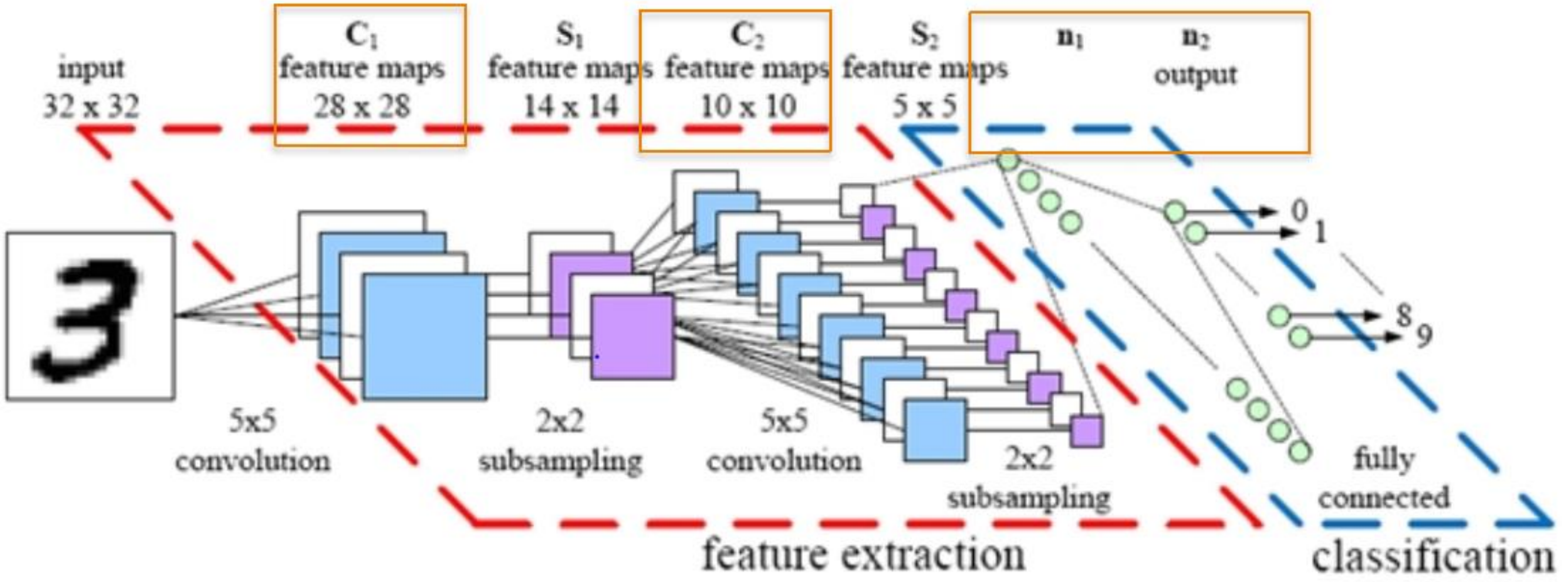
Published in ICLR 2016

Agenda

- Introduction
  - Convolutional neural network
  - Matrix multiplication-based implementation of convolution
  - Profiling CNNs on mobile GPU
- Redundancy in CNNs
  - Problem and existing methods to remove redundancy
- Proposed CNN compression method
  - Low-rank approximation based on Tucker and variational Bayesian matrix factorization
  - Evaluations on Titan X and Galaxy S6
- Summary

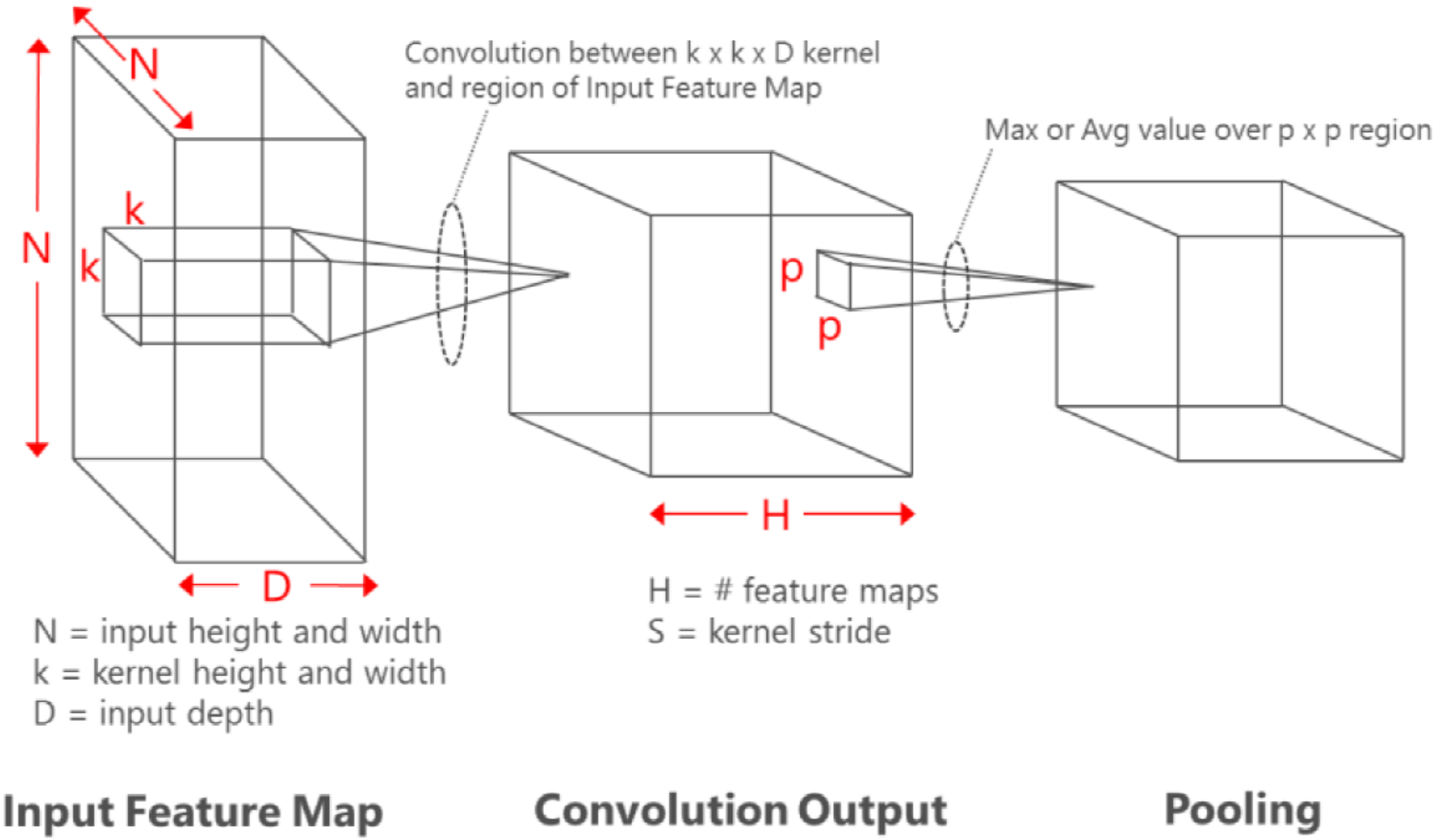
Convolutional Neural Network (CNN)

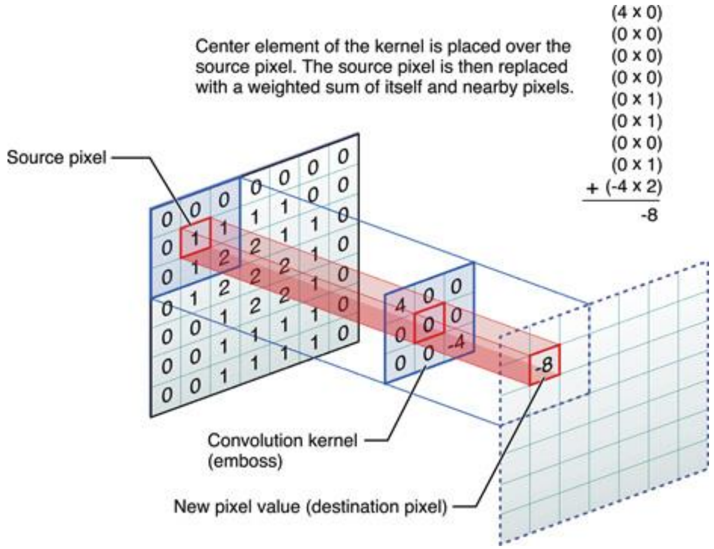
- LeNet (1989)



- CNN consists of convolution layer and subsampling (max-pooling) layer

Convolution and Pooling



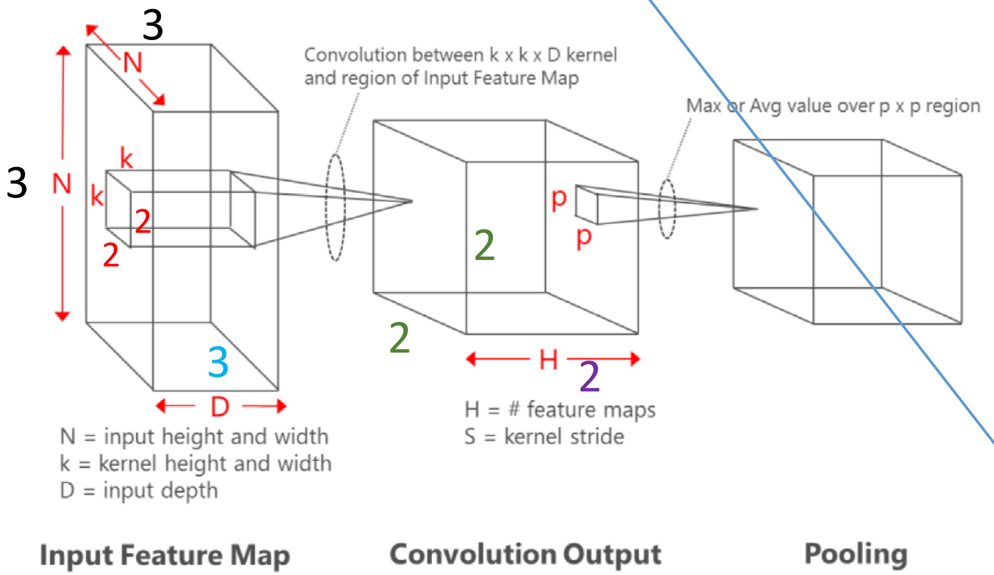


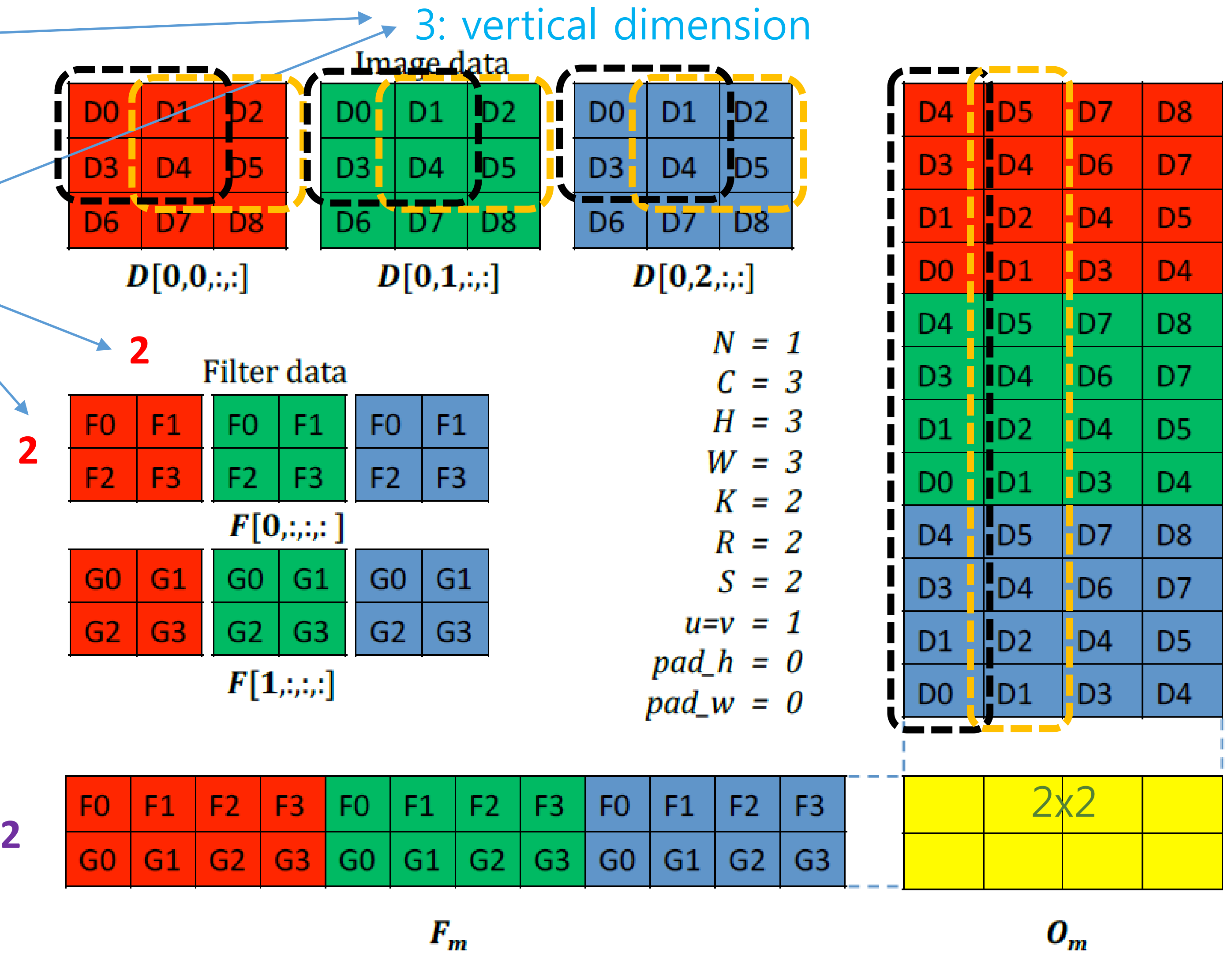
In convolution, # parameters = kxkxDxH

[Chetlur 2014]

Convolution with Matrix Multiplication (called Convolution Lowering)

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



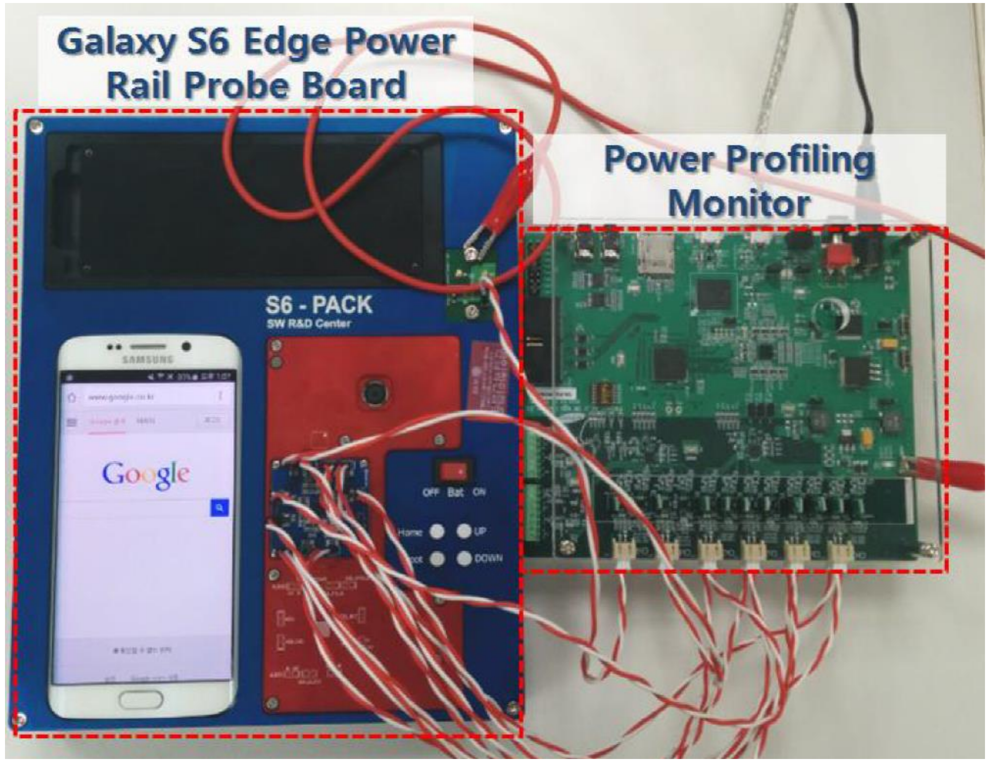


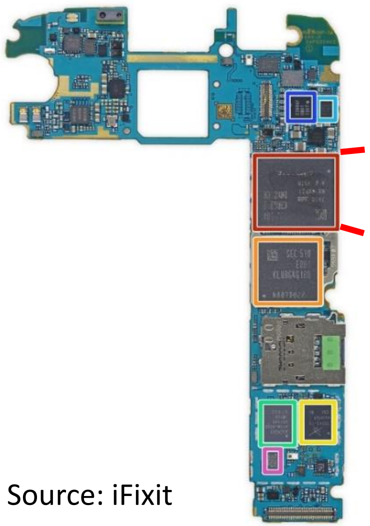
Agenda

- Introduction
  - Convolutional neural network
  - Matrix multiplication-based implementation of convolution
  - Profiling CNNs on mobile GPU
- Redundancy in CNNs
  - Problem and existing methods to remove redundancy
- Proposed CNN compression method
  - Low-rank approximation based on Tucker and variational Bayesian matrix factorization
  - Evaluations on Titan X and Galaxy S6
- Summary

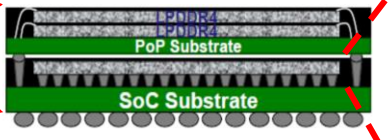
Measurement System

- Running CNNs (in OpenCL) on Galaxy S6 Edge
  - Exynos 7420 (Mali T760) + LPDDR4 DRAM
- 6 power probes to
  - CPU, GPU (T760), two DRAM dies, …

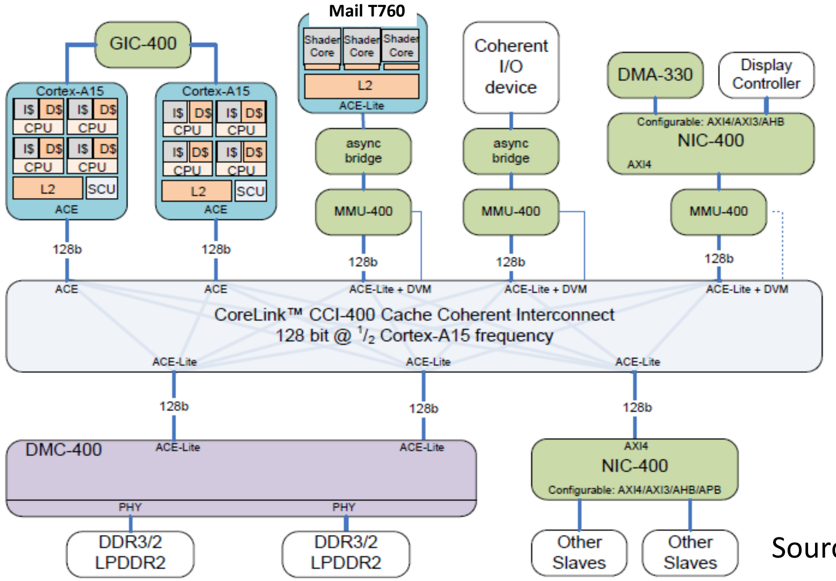




Source: iFixit



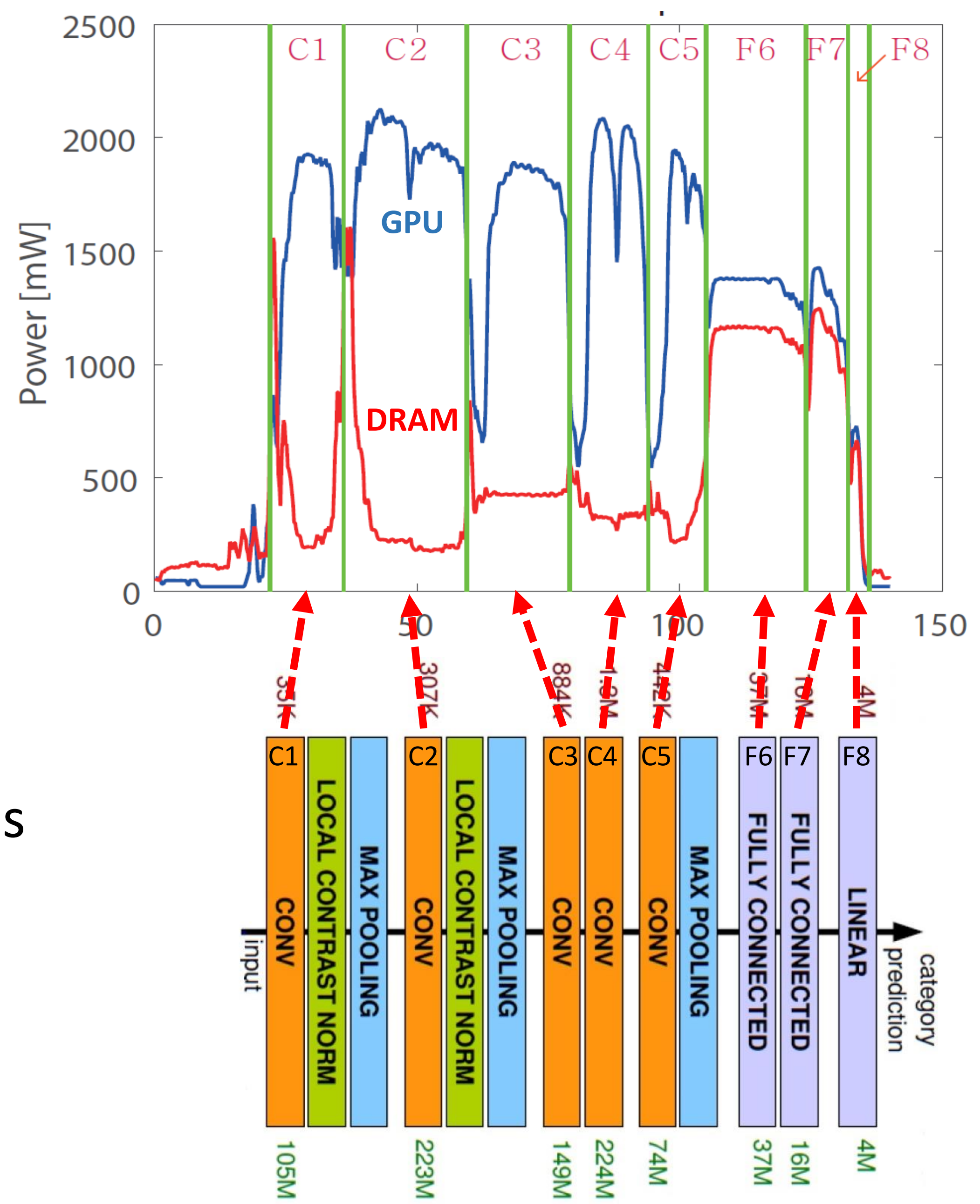
Exynos 7420



Source: ARM, Ltd.

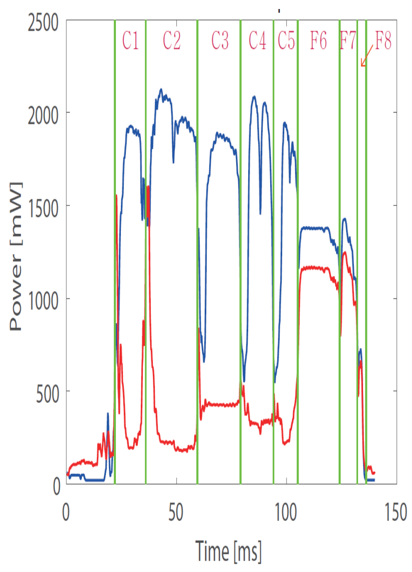
AlexNet: Power Consumption

- Total 245mJ/image, 117ms
  - GPU power > DRAM power
- Convolutional layers dominate total energy consumption and runtime
- At fully connected layers, GPU power drops while DRAM power increases
  - Due to a large number of memory accesses for weights and less data reuse, i.e., low core utilization (=long total idle time)



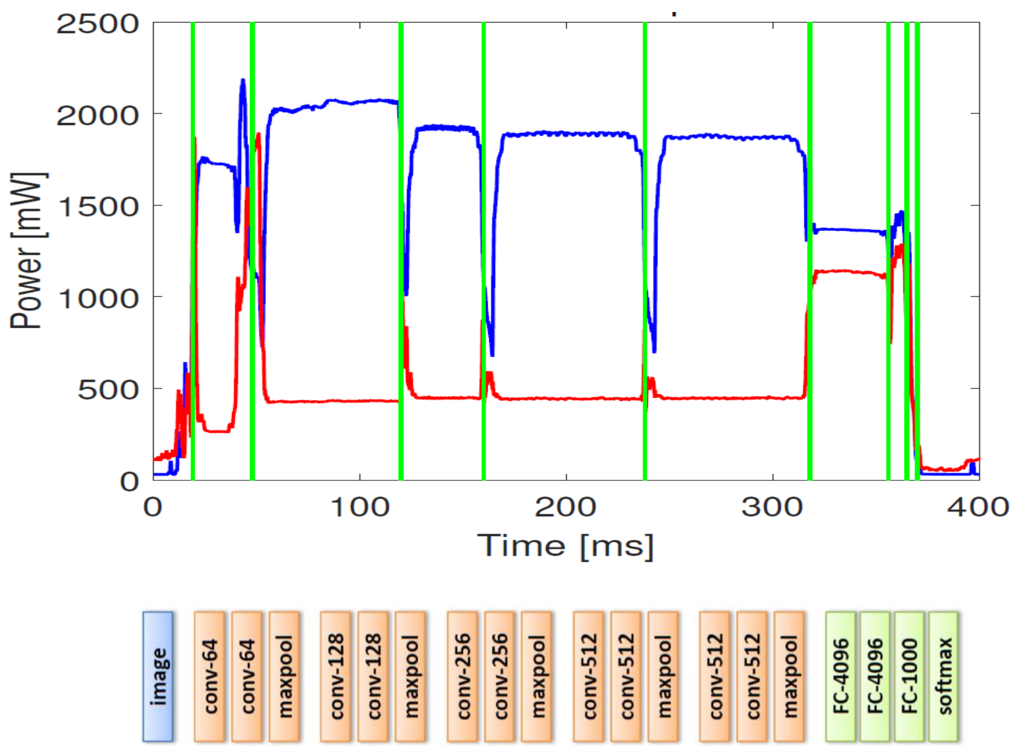
VGG_S: Power Consumption

- Total 825mJ/image, 357ms
- Convolutional layers dominate total energy consumption and runtime
- At convolutional layers, DRAM consumes larger power than in AlexNet due to a large number of weights
- At fully connected layers, similar trend as in AlexNet
  - GPU power ~ DRAM power



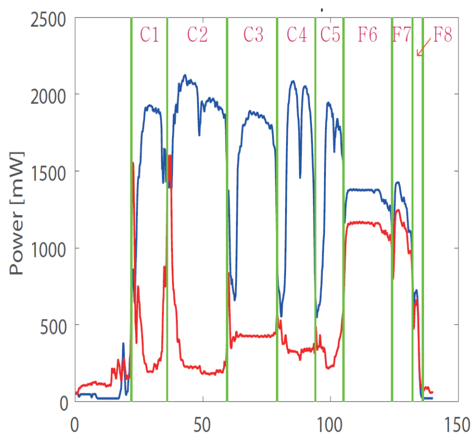
AlexNet

VGG_S



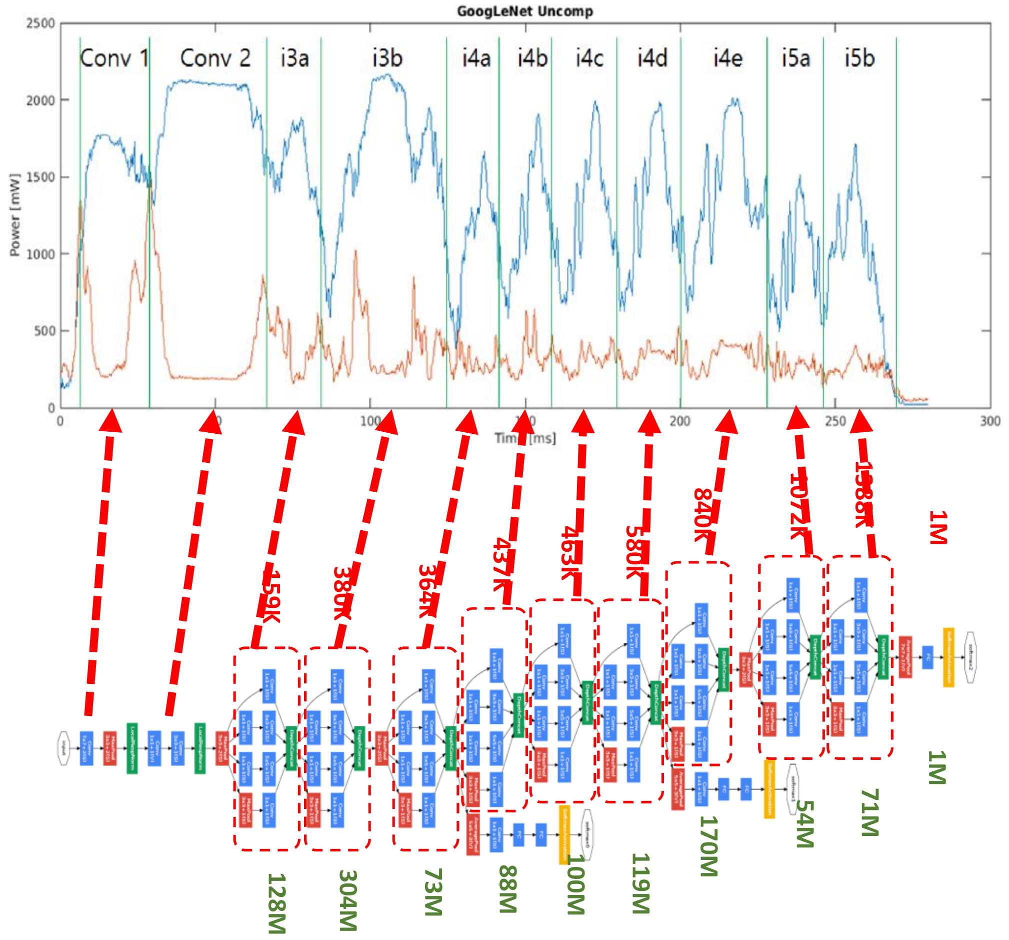
GoogLeNet: Power Consumption

- Total 473mJ/image, 273ms
- 1st and 2nd convolutional layers consume ~20% of total energy and runtime
- Inception modules
  - Relatively low power consumption in both GPU and DRAM
  - Power consumption fluctuates due to many convolution sub-layers in inception modules
- Fully connected layer (1M parameters) consumes a very little amount of power in GPU and DRAM

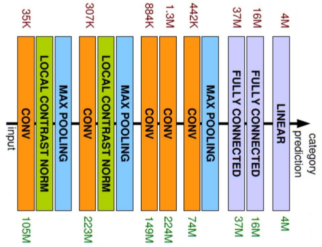


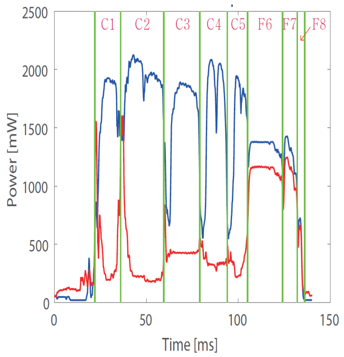
AlexNet

GoogLeNet

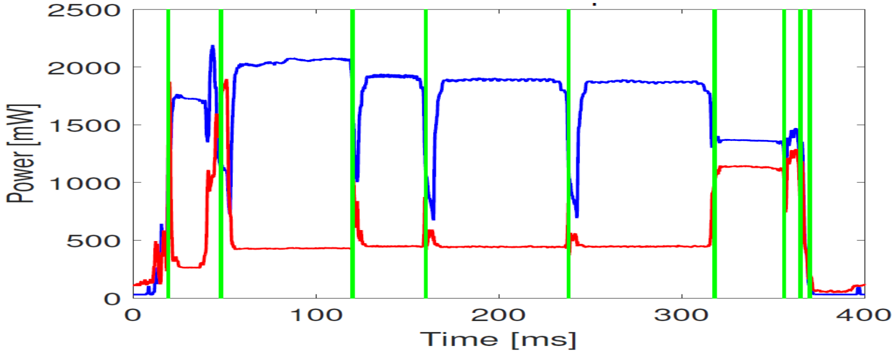


AlexNet, VGG_S vs GoogLeNet: Top-5, Runtime and Power

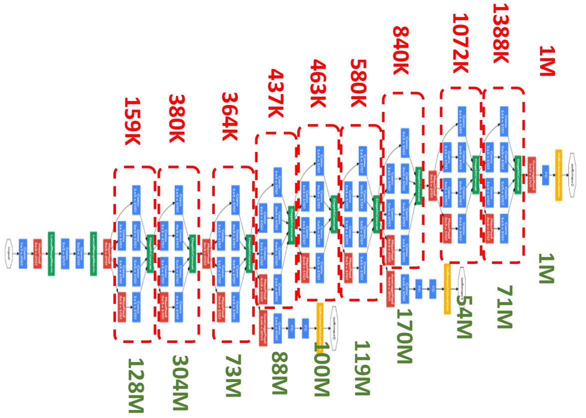


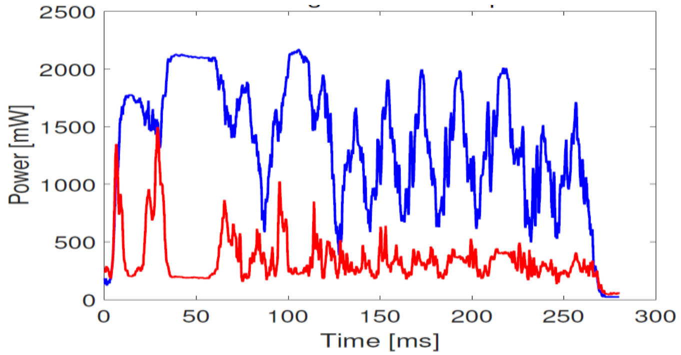


80.0%
117ms
245mJ



84.6%
357ms
825mJ





88.9%
273ms
473mJ

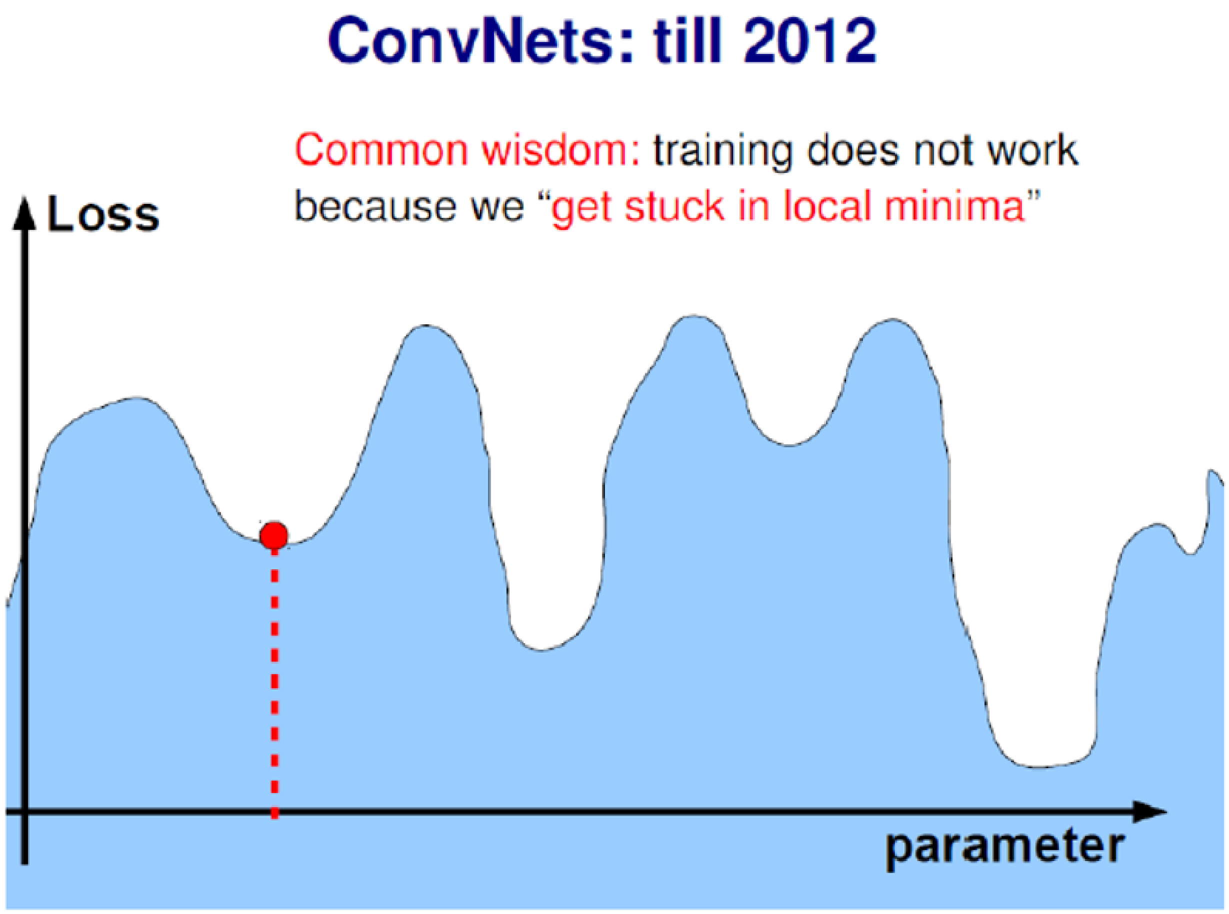
Agenda

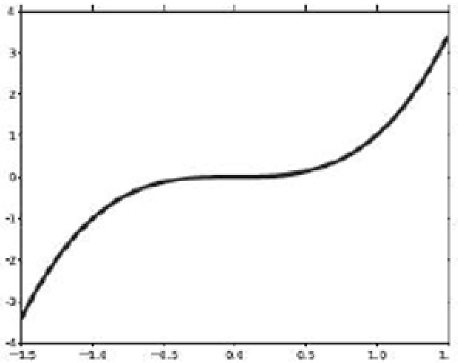
- Introduction
  - Convolutional neural network
  - Matrix multiplication-based implementation of convolution
  - Profiling CNNs on mobile GPU
- Redundancy in CNNs
  - Problem and existing methods to remove redundancy
- Proposed CNN compression method
  - Low-rank approximation based on Tucker and variational Bayesian matrix factorization
  - Evaluations on Titan X and Galaxy S6
- Summary

Typical CNN Design Steps

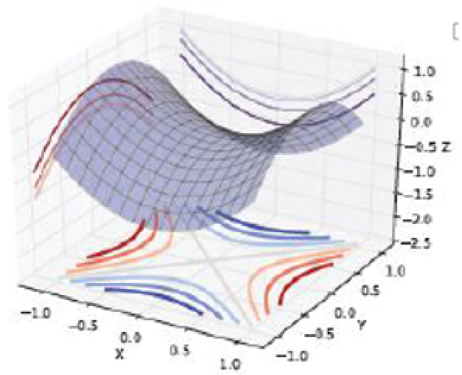
- Step 1: Train a large CNN
  - Training based on back propagation
  - The CNN under training will be over-parameterized, i.e., an over-design
  - It is to facilitate fast convergence to good local minima

Local Minima are mostly Saddle Points in High-Dimensional Space





(a)

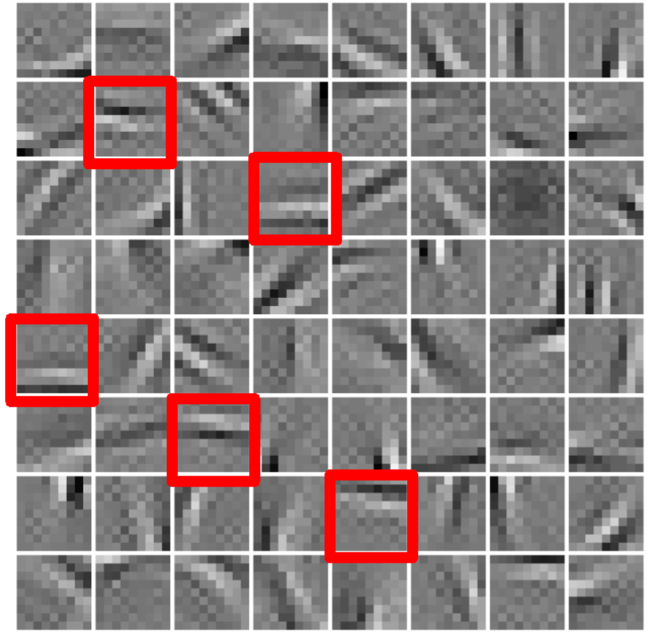


(b)

Slide: Ranzato

Redundancy in CNN

- **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



weights :-0.2828 - 0.3043

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

How to remove redundant feature maps?

Typical CNN Design Steps

- Step 1: Train a large CNN
  - Training based on back propagation
  - The CNN under training will be over-parameterized, i.e., an over-design
  - It is to facilitate fast convergence to good local minima

- Step 2: Compress the trained CNN
  - CNN compression aims at removing redundancy in the trained CNN
  - It is especially important for mobile and embedded devices which have very limited computing resource

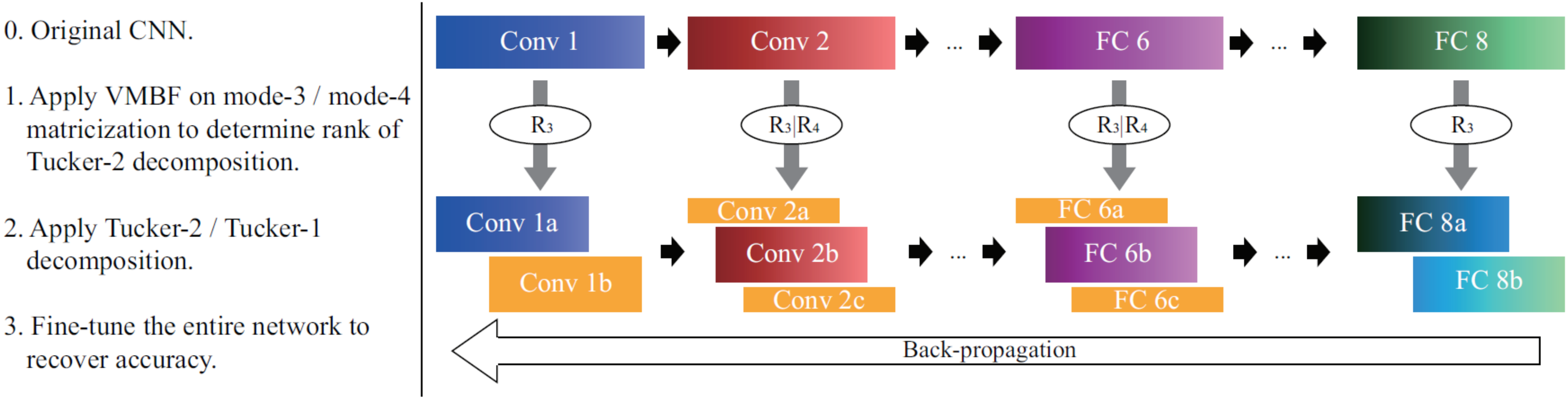
Existing Methods to Remove Redundancy

- Bit width optimization
  - 32-bit → 16-bit [NVIDIA Pascal]
  - BinaryConnect [Bengio, NIPS15], XNOR-Net [Rastegari, 2016]
- Pruning
  - Remove unimportant connections and neurons [Han, NIPS15]
  - Non-uniform quantization and weight compression [Han, ICLR16 submission]
- Low-rank approximation
  - Bi-clustering and truncated SVD [Denton, NIPS14]
  - CP decomposition [Lebedev, ICLR15]
  - Asymmetric 3D decomposition and truncated SVD [Zhang, arXiv:1505.06798]
  - **Tucker and variational Bayesian matrix factorization: ours**

Agenda

- Introduction
  - Convolutional neural network
  - Matrix multiplication-based implementation of convolution
  - Profiling CNNs on mobile GPU
- Redundancy in CNNs
  - Problem and existing methods to remove redundancy
- Proposed CNN compression method
  - Low-rank approximation based on Tucker and variational Bayesian matrix factorization
  - Evaluations on Titan X and Galaxy S6
- Summary

Overall Flow



- More details
  - Y. Kim, et al., "Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications," arXiv:1511.06530v1

[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

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

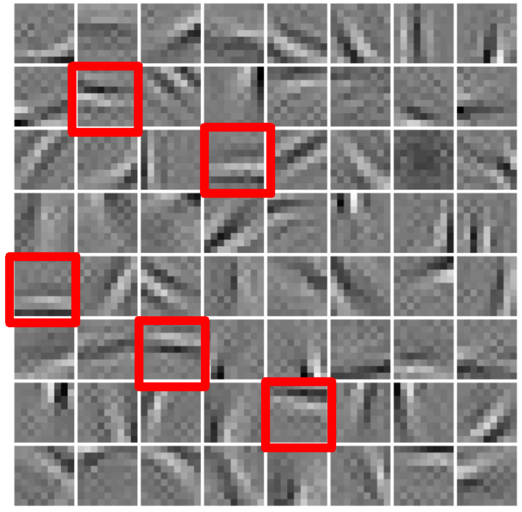
Error degrades accuracy. How to address this lost accuracy?

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

Tucker Method to Resolve Redundancy Problem: Reducing # Feature Maps

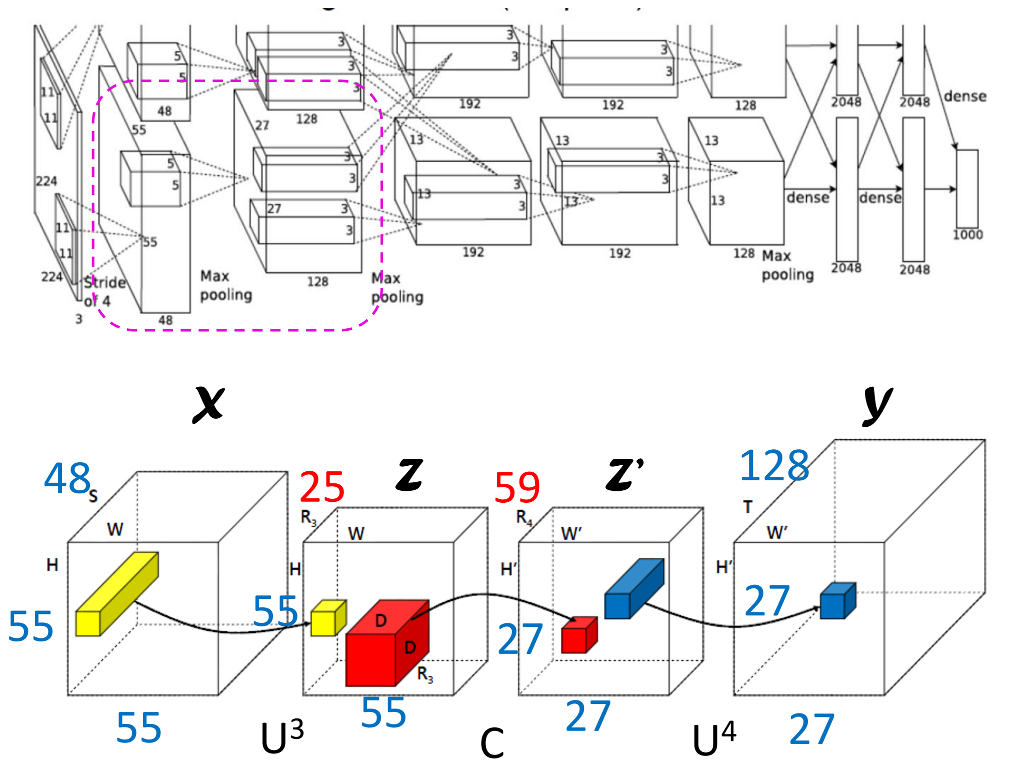
- **Problem:**
  - With patch-level training, the learning algorithm must reconstruct the entire patch with a single feature vector
  - But when the filters are used convolutionally, neighboring feature vectors will be highly redundant

How to remove redundant feature maps?



Yann LeCun

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

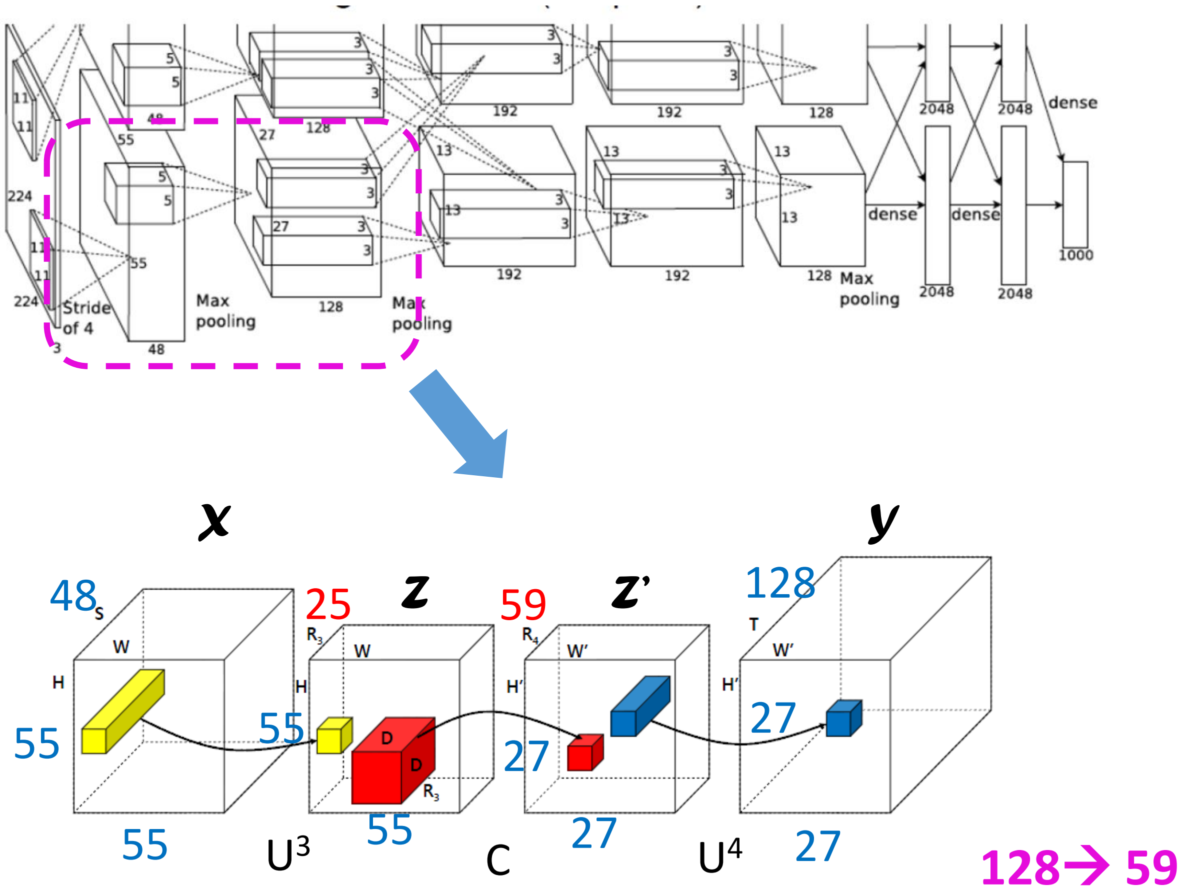


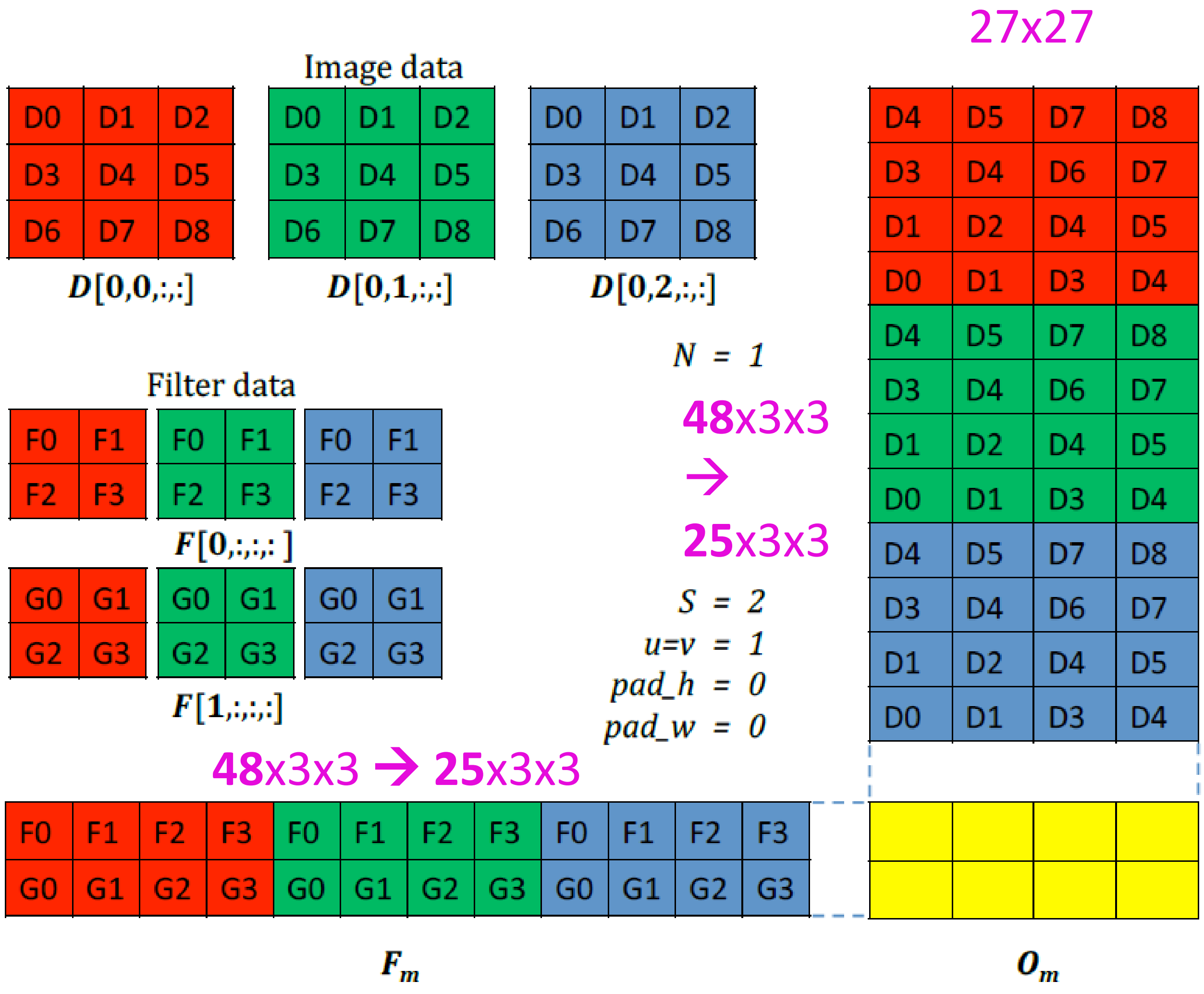
feature maps is reduced at input (48→25 in ***z***) and output (128→59 in ***z'***)

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

[Chetlur 2014]

Matrix Sizes are Reduced in Convolution

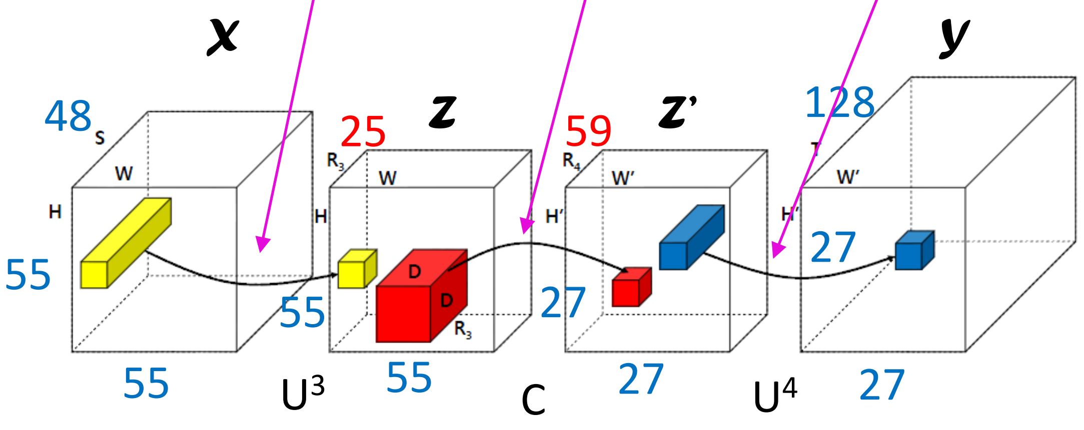




Reduction in Computation: AlexNet Case

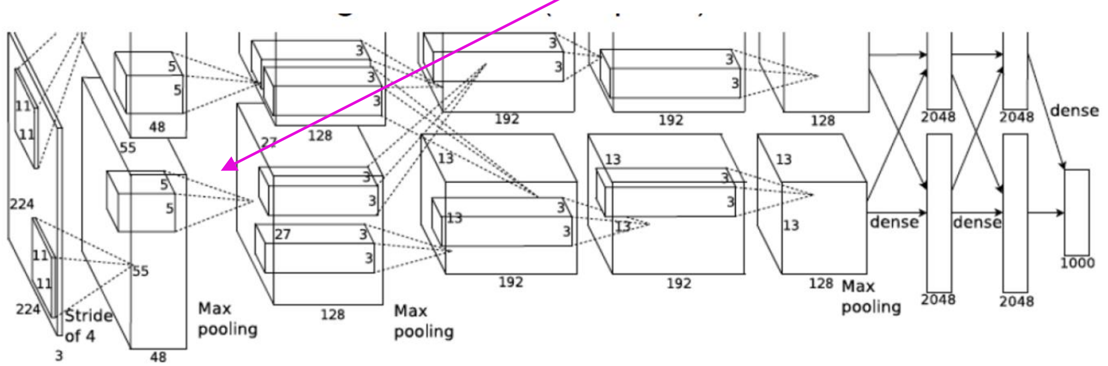
Layer	$S/R3$	T/R_4	Weights	FLOPs	S6
conv1	3	96	35K	105M	15.05 ms
conv1*		26	11K	36M(=29+7)	10.19m(=8.28+1.90)
(imp.)			(×2.92)	(×2.92)	(×1.48)
conv2	48 × 2	128 × 2	307K	224M	24.25 ms
conv2*	25 × 2	59 × 2	91K	67M(=2+54+11)	10.53ms(=0.80+7.43+2.30)
(imp.)			(×3.37)	(×3.37)	(×2.30)
conv3	256	384	885K	150M	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+**25x5x5**x**59**x27x27+**59**x128x27x27



3.1X less computation!

Conv2: # multiplications = 48x5x5x128x27x27



[K. Baker, Singular Value Decomposition Tutorial]

Error in Truncated SVD

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

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

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

Take 3 largest square roots of eigenvalues

Take 3 eigenvectors associated with the selected 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?

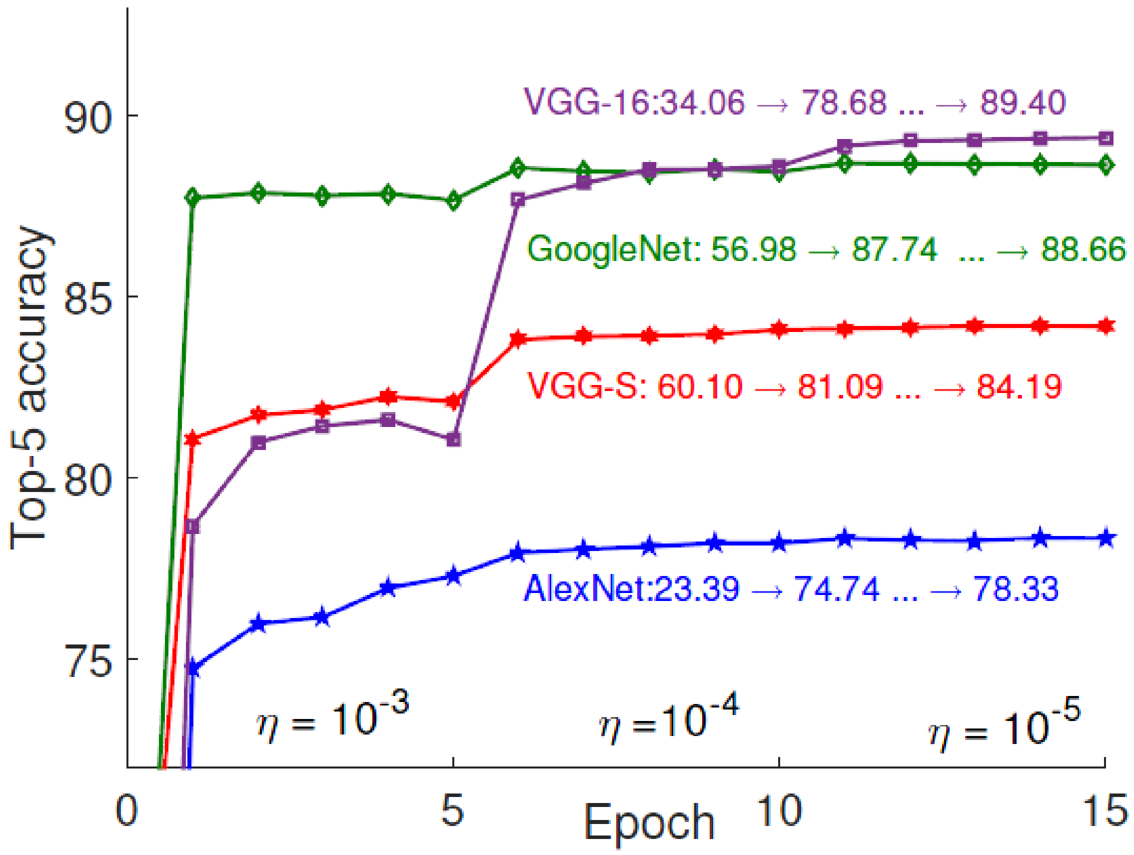
$$= \begin{bmatrix} 2.29 & -0.66 & 9.33 & 1.25 & -3.09 \\ 1.77 & 6.76 & 0.90 & -5.50 & -2.13 \\ 4.86 & -0.96 & 8.01 & 0.38 & -0.97 \\ 6.62 & -1.23 & 9.58 & 0.24 & -0.71 \\ 1.14 & 9.19 & 0.33 & -7.19 & -3.13 \end{bmatrix} \Leftrightarrow A = \begin{bmatrix} 2 & 0 & 8 & 6 & 0 \\ 1 & 6 & 0 & 1 & 7 \\ 5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

Fine-tuning

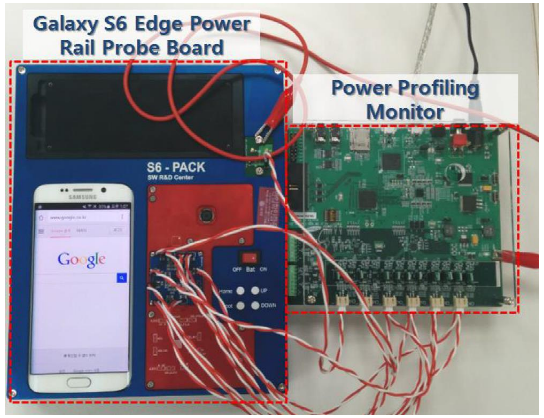
- Low-rank approximation loses accuracy
- Fine-tuning recovers lost error
  - 1 epoch: 1 run of back propagation with the entire training set

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

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



[Kim, 2015]



Results on Titan X and Galaxy S6

- Significant reductions in energy consumption and runtime
  - Energy: 4.26X~1.6X
  - Runtime: 3.68X~1.42X
  - Comparable to Zhang et al.'s

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)

Galaxy S6 results

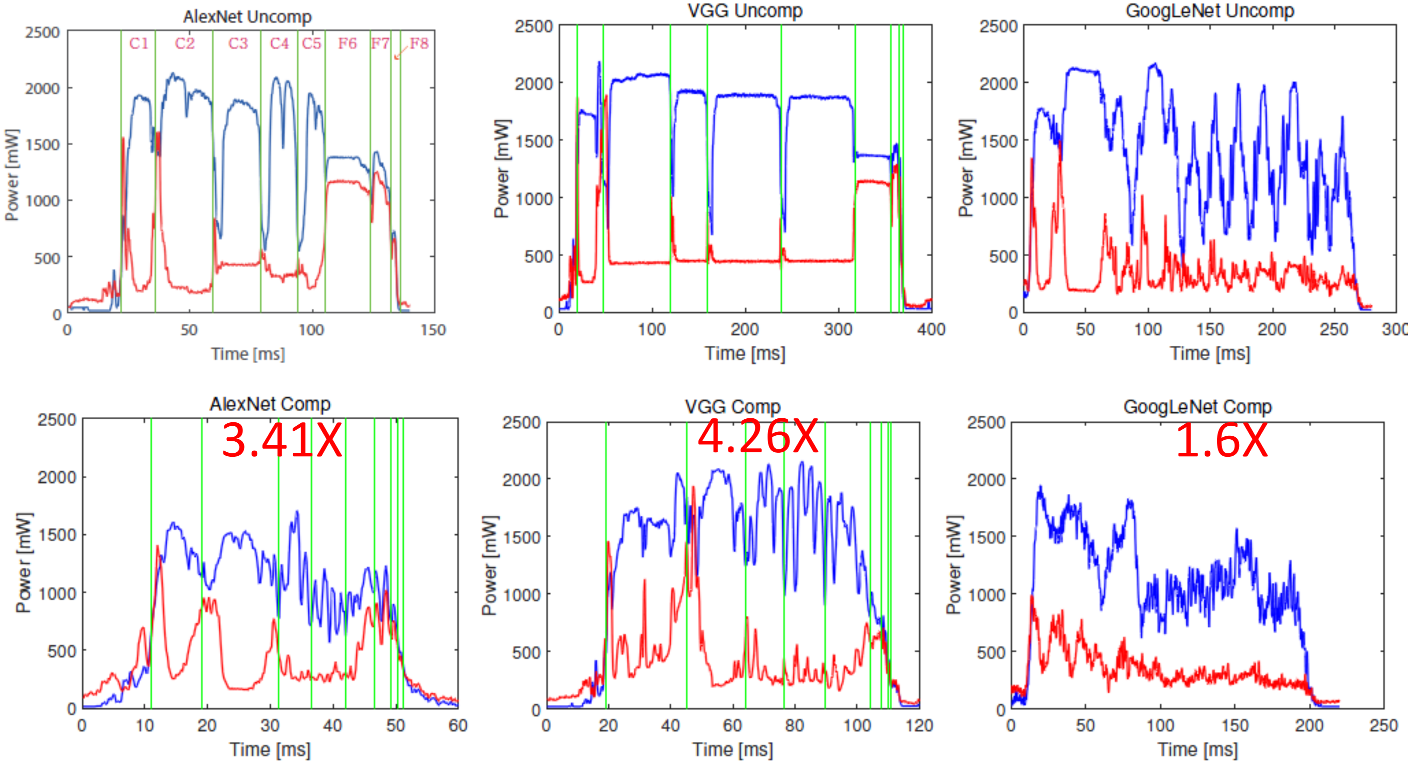


Figure 5: Power consumption over time for each model. (Blue: GPU, Red: main memory).

Summary

- CNNs are often over-parameterized
  - For fast convergence to good local minima during training
  - Redundancy needs to be removed for *test-time* performance and power consumption
- Low-rank approximation
  - Is a promising solution to remove redundancy, *statically*
  - Reduces matrix sizes in CNN computation thereby offering less computation and smaller model size
    - Rank selection: variational Bayesian matrix factorization
    - Low-rank approximation: Tucker method, e.g., reducing # feature maps
  - Can be applied together with other optimizations, e.g., hardware accelerator, bit-width optimization, FFT, cascading, etc.
- Next steps
  - *Dynamic* solutions to remove redundancy