# COMPRESSION OF DEEP CONVOLUTIONAL NEURAL NETWORKS FOR FAST AND LOW POWER MOBILE APPLICATIONS


Group 09

E/15/065 - DE SILVA K.G.P.M. - e15065@eng.pdn.ac.lk

E/15/119 - HASANIKA D.L.D. - e15119@eng.pdn.ac.lk

E/15/202 - LIYANAGE D.P. - e15202@eng.pdn.ac.lk

E/15/208 - MADHUSHANEE G.G.R.D. - rosh.madhu@eng.pdn.ac.lk

# Background

Authors : Yong-Deok Kim
Eunhyeok Park
Sungjoo Yoo
Taelim Choi
Lu Yang
Dongjun Shin

Published : a conference paper at ICLR 2016

# Introduction

# Mobile applications of CNNs

- Mobile devices use CPU and GPU, running deeper CNNs for complex tasks
  Ex: **ImageNet classification**

- Issues  - **Mobile devices have strict constraints in computing power, battery, and memory capacity**

- Improve test-time performance - **Compressions on convolution layers** without noticeable impact on accuracy

# Whole network compression

- Existing methods effective in reducing the computation cost of a single convolutional layer
- Aims at **compressing the entire network**
- Reduce the computational cost
- Nontrivial to compress whole and very deep CNNs for complex tasks such as ImageNet classification
- Methods used Earlier - Asymmetric (3d) decomposition (Zhang et al. (2015b)
- **This paper presents simple, powerful whole network compression**

# Contribution

- **One-shot whole network compression** consists of 3 steps
  - Rank selection
  - Low-rank tensor decomposition
  - Fine-tuning
- Can be easily implemented using publicly available tools
- Evaluate various compressed CNNs on both Titan X and smartphone
  - AlexNet
  - VGG-S
  - GoogLeNet
  - VGG-16
- Significant reduction in model size, runtime, and energy consumption are obtained, at the cost of small loss in accuracy
- Analyse power consumption over time and observe behaviours of $1 \times 1$ convolution

# Related Work

# 1. CNN Compression

- **Singular value decomposition(SVD)** (Denton et al., 2014)
  - The weight matrix of a fully-connected layer can be compressed by applying truncated SVD without significant drop in the prediction accuracy

- **Vector quantization** (Gong et al., 2014), **Hashing techniques** (Chen et al., 2015), **Circulant projection** (Cheng et al., 2015), **Tensor train decomposition** (Novikov et al., 2015)
  - Better compression capability than SVD

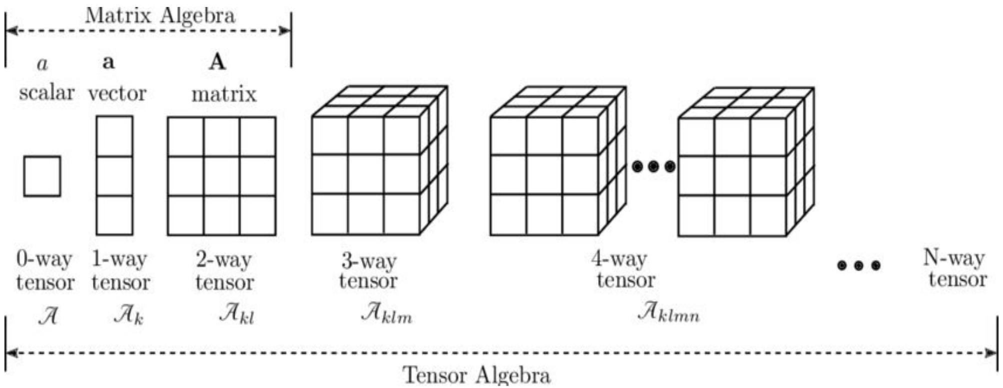
- **Low-rank decomposition of convolutional kernel tensor** (Denton et al., 2014; Jaderberg et al.,2014; Lebedev et al., 2015)
  - Speed up the convolutional layers
  - Compress only single or a few layers

- **Asymmetric (3d) decomposition** (Zhang et al. (2015b))
    - To accelerate the entire convolutional layers, the original D × D convolution is decomposed to D × 1, 1 × D, and 1 × 1 convolution
    - Present a rank selection method based on PCA accumulated energy
    - Present an optimization method which minimizes the reconstruction error of non-linear responses

- **Pruning approach** (Han et al., 2015b;a)
    - Reduce the total amount of parameters and operations in the entire network

- **Implementation level approaches**
    - **FFT method** was used to speed-up convolution (Mathieu et al., 2013)
    - In (Vanhoucke et al., 2011), CPU code optimizations to speed-up the execution of CNN

# 2. Tensor Decomposition




- Tensor - multiway array of data
  Example: Vector - 1 way tensor
  Matrix - 2 way tensor

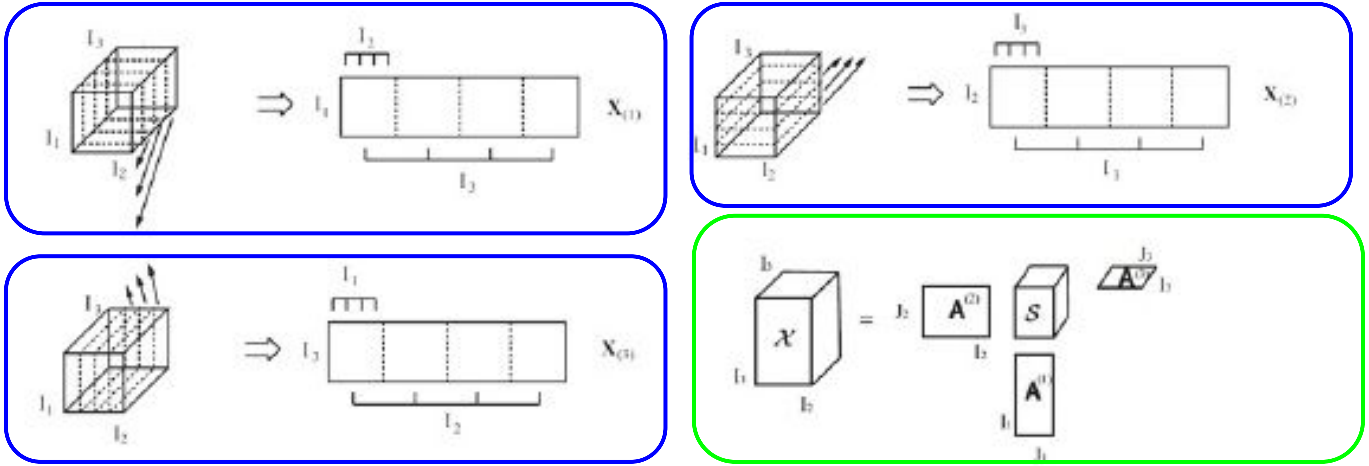
- Two of the most popular tensor decomposition models
  1. CANDECOMP/PARAFAC model (Carroll & Chang, 1970; Harshman & Lundy, 1994; Shashua & Hazan, 2005)
  2. Tucker model (Tucker, 1966; De Lathauwer et al., 2000; Kim & Choi, 2007)

- In the paper - Tucker model for whole network compression

- Tucker-2 decomposition (GLRAM)
  - from the second convolutional layer to the first fully connected layer

- Tucker-1 decomposition
  - Other layers
  - Equivalent to SVD

- Tucker decomposition
  - A higher order extension of the singular value decomposition (SVD) of matrix
  - Perspective: computing the orthonormal spaces associated with the different modes of a tensor
  - Analyzes mode-n matricizations of the original tensor
  - Merges them with core tensor



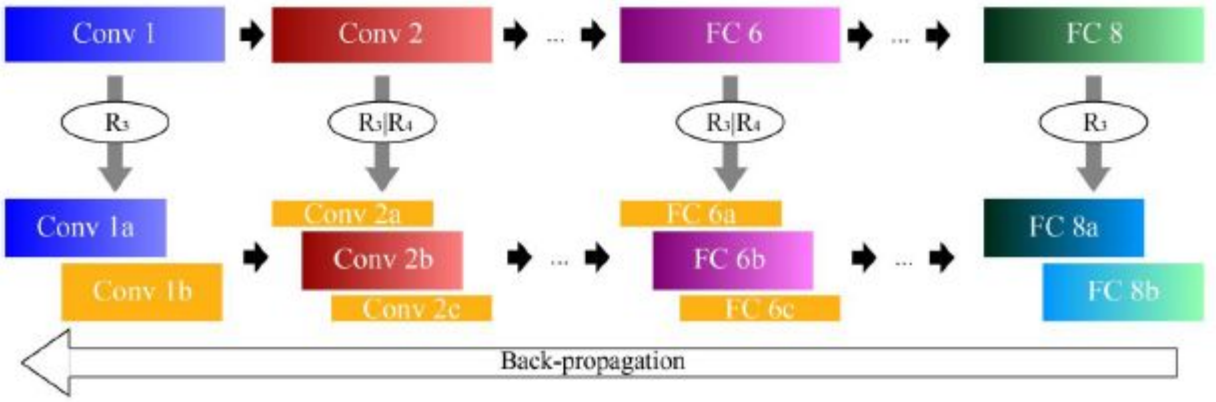
# Difference of this paper compared to above related works

- Tucker decomposition is adopted to compress the entire convolutional and fully-connected layers
- The kernel tensor reconstruction error is minimized instead of non-linear response
- A global analytic solution of VBMF (Nakajima et al., 2012) is applied to determine the rank of each layer
- A single run of fine-tuning is performed to account for the accumulation of errors.

# Proposed Method

# One-shot whole network compression scheme

- Three steps
  1) Rank Selection
     - Analyze principal subspace of mode-3 and mode-4 matricization of each layer's kernel tensor with global analytic variational Bayesian matrix factorization
  2) Tucker decomposition
  3) Fine-tuning
     - Standard back-propagation



# Tucker Decomposition on Kernel Tensor

Convolution kernel tensor

- input (source) tensor X - size H x W x S
- output (target) tensor Y - size H’ x W’ x S

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

$$h_i = (h'-1)\Delta + i - P \text{ and } w_j = (w'-1)\Delta + j - P$$

➔ K = 4-way kernel tensor of size D x D x S x T
➔ Δ = stride
➔ P = zero-padding size

## Tucker Decomposition

- K = The rank-(R1;R2;R3;R4) Tucker decomposition of 4-way kernel tensor

$$\mathcal{K}_{i,j,s,t} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \sum_{r_3=1}^{R_3} \sum_{r_4=1}^{R_4} \mathcal{C}'_{r_1,r_2,r_3,r_4} U^{(1)}_{i,r_1} U^{(2)}_{j,r_2} U^{(3)}_{s,r_3} U^{(4)}_{t,r_4}, \tag{2}$$

➔ C’ = core tensor of size $R_1$ x $R_2$ x $R_3$ x $R_4$

➔ $U^{(1)}$, $U^{(2)}$, $U^{(3)}$, $U^{(4)}$ = factor matrices - sizes D x $R_1$, D x $R_2$, S x $R_3$, and T x $R_4$

- Under Tucker-2 decomposition, the kernel tensor is decomposed to:

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

➔ C = a core tensor of size D x D x $R_3$ x $R_4$

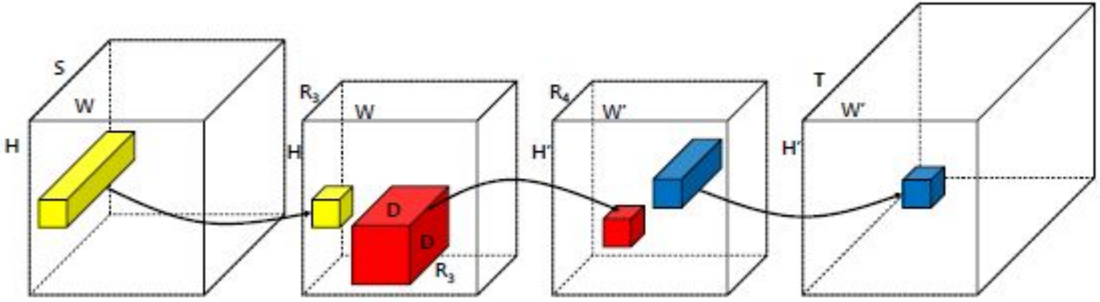
- After substituting, performing rearrangements and grouping summands:

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

$$\mathcal{Z}'_{h',w',r_4} = \sum_{i=1}^{D}\sum_{j=1}^{D}\sum_{r_3=1}^{R_3} \mathcal{C}_{i,j,r_3,r_4}\, \mathcal{Z}_{h_i,w_j,r_3}, \tag{4}$$

$$\mathcal{Y}_{h',w',t} = \sum_{r_4=1}^{R_4} U^{(4)}_{t,r_4}\, \mathcal{Z}'_{h',w',r_4}, \tag{5}$$

  ➔ Z and Z' are intermediate tensors of sizes H x W x $R_3$ and H' x W' x $R_4$




Tucker-2 decompositions
for speeding-up a convolution

- ➔ Each transparent box = 3-way tensor X, Z, Z’, Y
- ➔ Two frontal sides = spatial dimensions
- ➔ Arrows = linear mappings - illustrate how scalar values on the right are computed
- ➔ Yellow tube, red-box, and blue tube = 1 x 1, D x D, and 1 x 1 convolution in (3), (4), and (5)

# 1 x 1 convolution

- Computing Z from X in (3) and Y from Z’ in (5)
- perform pixel-wise linear re-combination of input maps
- Introduced in network-in-network
- Extensively used in inception module of GoogLeNet

## Complexity analysis

$$M = \frac{D^2 ST}{SR_3 + D^2 R_3 R_4 + TR_4} \quad \text{and} \quad E = \frac{D^2 STH'W'}{SR_3 HW + D^2 R_3 R_4 H'W' + TR_4 H'W'},$$

- ➔ M = Compression ratio
- ➔ E = Speed-up ratio

- Bounded by ST=$R_3R_4$

## Tucker vs CP

- CP decomposition
  - Applied to approximate the convolution layers of CNNs for ImageNet which consist of 8 layers
  - Cannot be applied to the entire layers
  - Instability issue of low-rank CP decomposition
- Kernel tensor approximation with Tucker decomposition
  - Can be successfully applied to the entire layers of AlexNet, VGG-S, GoogLeNet, and VGG-16

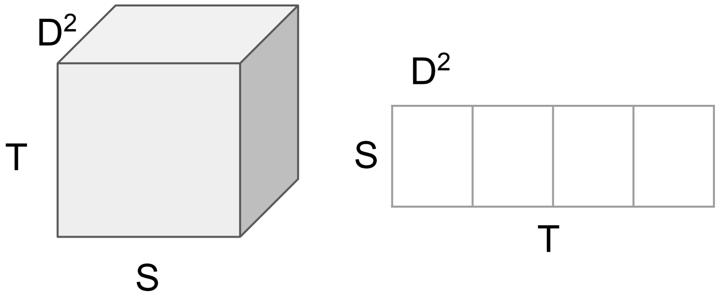
# Rank of a CNN

- **Key parameter that determines the complexity of each layer**
- **Directly related to,**
  - Memory usage
  - Runtime
  - Energy consumption
  - Accuracy

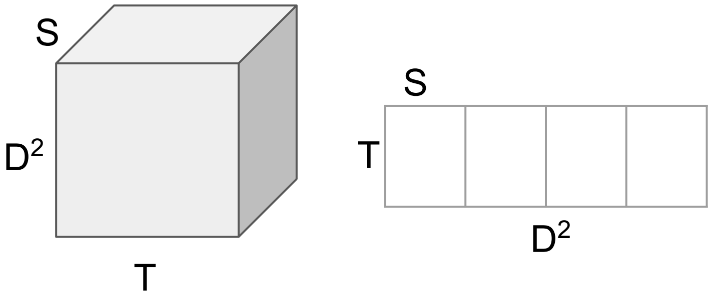
# Rank Selection with Global Analytic VBMF

- **VBMF - Variational Bayesian Matrix Factorization**
  - Available as a MATLAB function
  - Find the rank of matrix instead of tensor
  - Therefore tensors converted to matrices - process is called **matricization**
- **VBMF applied on,**
  - Mode 3 matricization - size is $S \times TD^2$
  - Mode 4 matricization - size is $T \times D^2S$
- **VBMF determined rank R3 and R4**

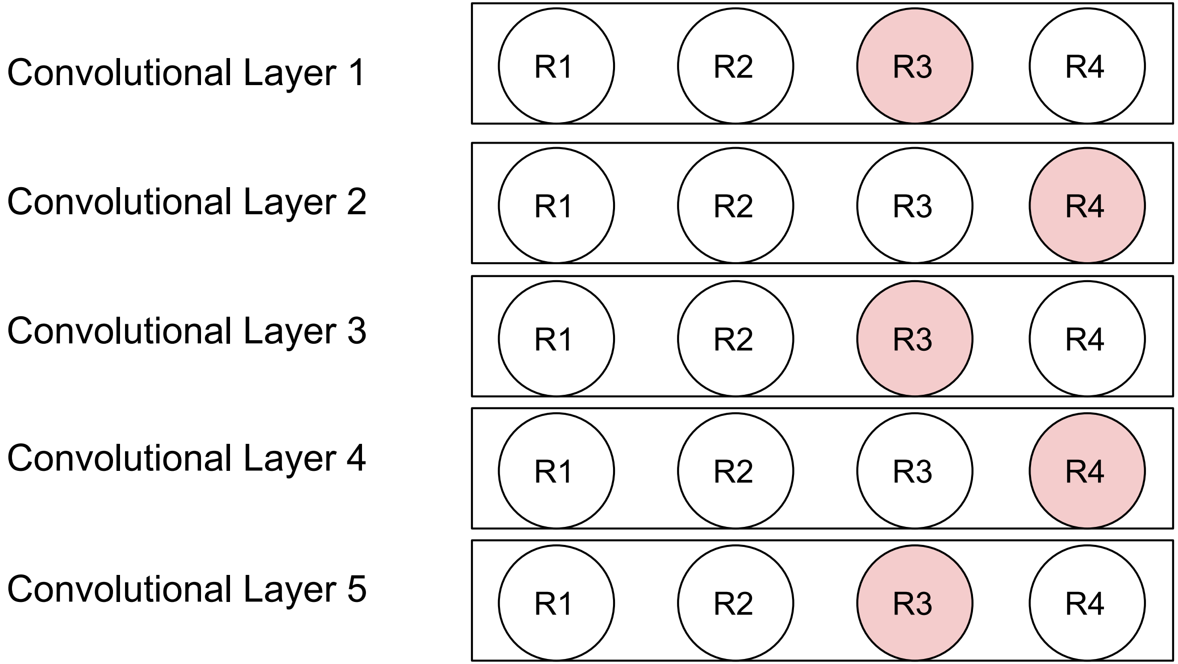
# Mode 3 matricization




# Mode 4 matricization



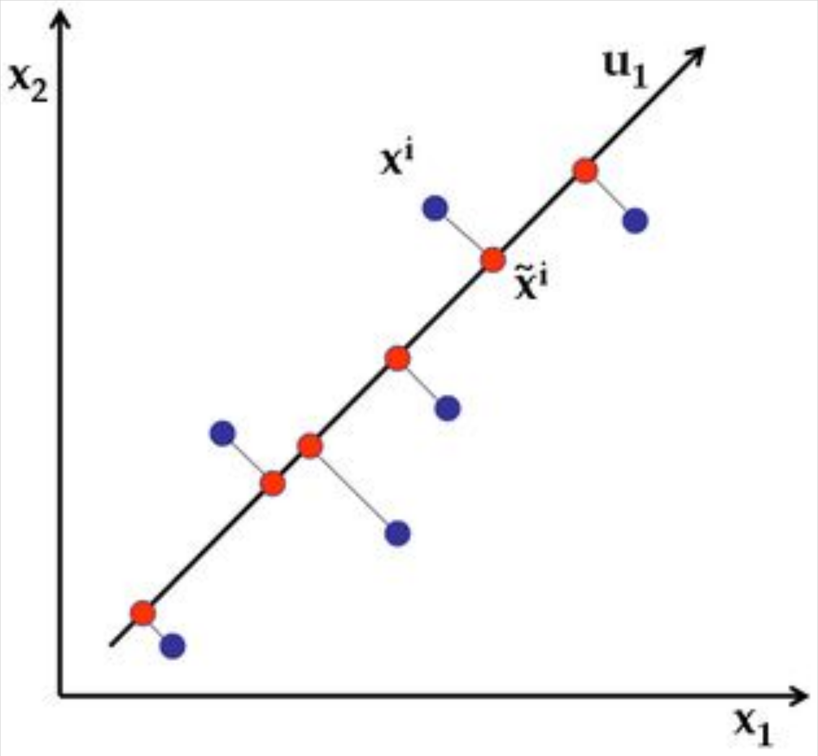
# Example of rank selection using VBMF on a CNN




**R1** - Rank of mode 1 matricization
**R2** - Rank of mode 2 matricization
**R3** - Rank of mode 3 matricization
**R4** - Rank of mode 4 matricization

E/15/202 - Dulanjali

# What is Reconstruction Error ?



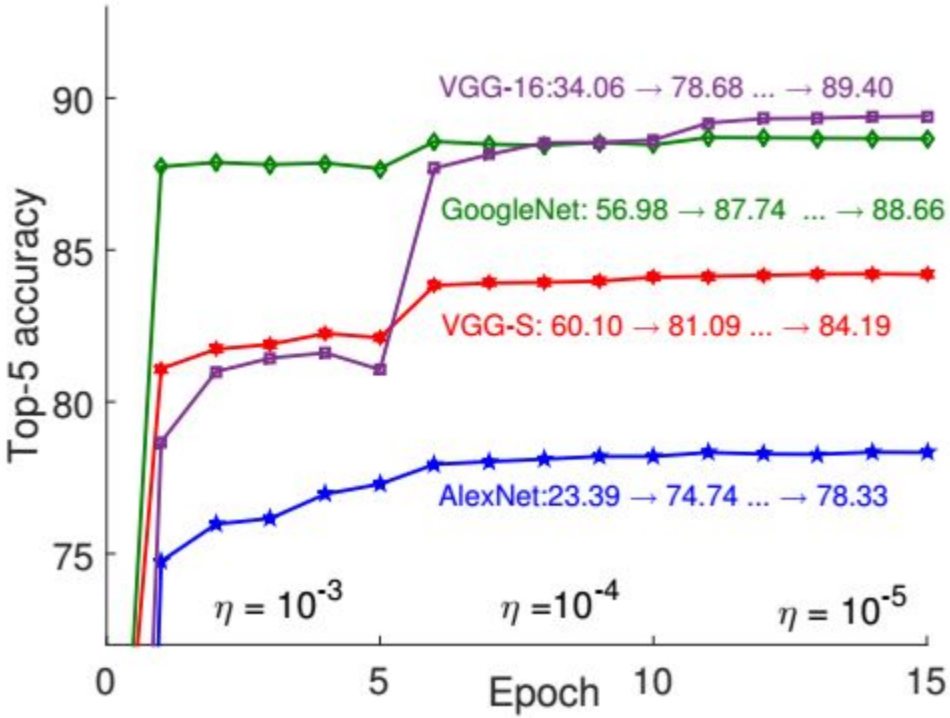

- The distance between original data point and it’s projection onto a lower dimensional subspace
- Red points - original data points
- Blue points - projected points

# Fine Tuning

- **Reconstruction error of linear kernel tensors were minimized**
  - Therefore accuracy dropped
  - For example AlexNet dropped more than 50%
- **Method of fine tuning ?**
  - Standard back propagation
- **Accuracy was recovered by using fine-tuning with ImageNet training dataset**
  - 1 epoch - recover accuracy quickly
  - More than 10 epochs - recover original accuracy

# Accuracy of compressed CNNs in fine-tuning

- Base learning $\eta = 10^{-3}$



# Experiments

# 1. Overall Results for ImageNet 2012 dataset

- Original Vs. Compressed CNN
- *** compression**
- Tested on
  - Smartphones
    - S6: Samsung Galaxy S6
  - Nvidia Titan X
- **FLOPs** - Floating point operations per second
- **Weights** - weights between input and hidden layer in NN

| 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) |

Accuracy

Runtime

Energy

# 2. Layerwise Analysis

Each row has two results

- Original uncompressed CNN
- Compressed CNN

Table 2: Layerwise analysis on *AlexNet*. Note that conv2, conv4, and conv5 layer have 2-group structure. ($S$: input channel dimension, $T$: output channel dimension, $(R_3, R_4)$: Tucker-2 rank).

| 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.) | | | $(\times 2.92)$ | $(\times 2.92)$ | $(\times 1.48)$ |
| conv2 | $48 \times 2$ | $128 \times 2$ | 307K | 224M | 24.25 ms |
| conv2* | $25 \times 2$ | $59 \times 2$ | 91K | 67M(=2+54+11) | 10.53ms(=0.80+7.43+2.30) |
| (imp.) | | | $(\times 3.37)$ | $(\times 3.37)$ | $(\times 2.30)$ |
| conv3 | 256 | 384 | 885K | 150M | 18.60ms |
| conv3* | 105 | 112 | 178K | 30M(=5+18+7) | 4.85ms(=1.00+2.72+1.13) |
| (imp.) | | | $(\times 5.03)$ | $(\times 5.03)$ | $(\times 3.84)$ |
| conv4 | $192 \times 2$ | $192 \times 2$ | 664K | 112M | 15.17ms |
| conv4* | $49 \times 2$ | $46 \times 2$ | 77K | 13M(=3+7+3) | 4.29 ms(=1.55+1.89+0.86) |
| (imp.) | | | $(\times 7.10)$ | $(\times 7.10)$ | $(\times 3.53)$ |
| conv5 | $192 \times 2$ | $128 \times 2$ | 442K | 75.0M | 10.78ms |
| conv5* | $40 \times 2$ | $34 \times 2$ | 49K | 8.2M(=2.6+4.1+1.5) | 3.44 ms(=1.15+1.61+0.68) |
| (imp.) | | | $(\times 9.11)$ | $(\times 9.11)$ | $(\times 3.13)$ |
| 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.) | | | $(\times 8.03)$ | $(\times 4.86)$ | $(\times 3.74)$ |
| 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.) | | | $(\times 6.80)$ | $(\times 6.80)$ | $(\times 7.61)$ |
| 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.) | | | $(\times 4.12)$ | $(\times 4.12)$ | $(\times 3.01)$ |

# Observations

- The smartphone tends to give larger performance gain than the Titan X
    - Mobile phone GPUs lacks in thread-level parallelism.
        - 24 times less number of threads than Titan X
    - Reduces the amount of weights by reducing cache conflicts and memory latency.

- Mobile phones shows larger performance in FC layers than Conv layers
    - Reduced cache conflicts enabled by network compression
    - The weights at the fully-connected layers are utilized only once (DoA)
    - DoA data are more harmful than Conv kernel weights

# 3. Energy Consumption Analysis

Compression reduces

- Power consumption
- Runtime

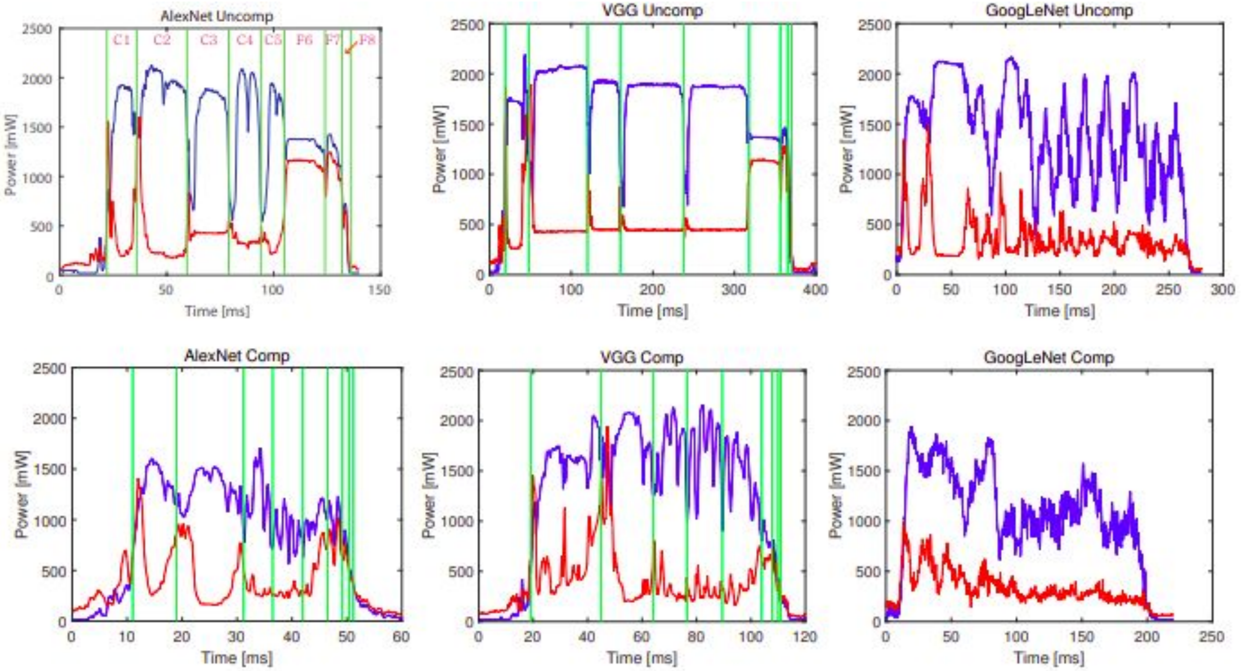



Figure 5: Power consumption over time for each model. (Blue: GPU, Red: main memory).

# Cond...

- The reduction in energy consumption is larger than that in runtime
- Power consumption of compressed CNN is smaller than uncompressed CNN
  - Due to the extensive usage of 1 × 1 convolutions in the compressed CNN
- For executing convolutions they applied optimization techniques such as Caffeinated convolution
- In cache efficiency, 1 × 1 convolutions are inferior to the other convolutions (3×3, 5×5 etc)
  - 1 × 1 convolutions tend to incur more cache misses
- However, 1 × 1 convolutions have negative impacts on cache efficiency and GPU core utilization

In the uncompressed networks,

- AlexNet and VGG-S - the power consumption of GPU core tends to be stable
- GoogLeNet - the power consumption tends to fluctuate.
- In fully connected layers incur significant amount of power consumption in main memory

In the compressed networks,

- The power consumption of GPU core  tends to change more frequently
- Reduces the amount of weights at fully connected layers

# Discussion

- One-shot rank selection
  - Very promising results
  - Not fully investigated yet whether the selected rank is really optimal/not
  - Future work: Investigate optimality of the proposed scheme
- 1 x 1 convolution
  - Key operation in compressed model and in inception module of GoogLeNet
  - Lacks in cache efficiency
  - Future work: investigating to make best use of 1x1 convolutions
- Whole network compression
  - Large design space and associated long design time
  - Propose:  a one-shot compression scheme (applies a single general low-rank approximation method and a global rank selection method)
- Oneshot compression
  - Fast design, easy implementation with publicly available tools
- Effectiveness evaluation - smartphone and Titan X
  - improvements in runtime (energy consumption) on the smartphone for 4 CNNs(AlexNet, VGG-S, GoogLeNet, and VGG- 16)

# THANK YOU !

# Q & A