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 $D \times D$ convolution is decomposed to $D \times 1$, $1 \times D$, and $1 \times 1$ 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)

- 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 $H \times W \times S$
- output (target) tensor $Y$ - size $H' \times W' \times S$

\[
Y_{h',w',t} = \sum_{i=1}^{D} \sum_{j=1}^{D} \sum_{s=1}^{S} K_{i,j,s,t} X_{h_t,w_t,s},
\]

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

- $K = 4$-way kernel tensor of size $D \times D \times S \times T$
- $\Delta = \text{stride}$
- $P = \text{zero-padding size}$
Tucker Decomposition

- K = The rank-$(R_1; R_2; R_3; R_4)$ Tucker decomposition of 4-way kernel tensor

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

→ C' = core tensor of size $R_1 \times R_2 \times R_3 \times R_4$

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

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

\[
K_{i,j,s,t} = \sum_{r_3=1}^{R_3} \sum_{r_4=1}^{R_4} C_{i,j,s,t} U^{(3)}_{s,r_3} U^{(4)}_{t,r_4}
\]

→ C = a core tensor of size $D \times D \times R_3 \times R_4$
After substituting, performing rearrangements and grouping summands:

\[ Z_{h,w,r_3} = \sum_{s=1}^{S} U_{s,r_3}^{(3)} X_{h,w,s} \]  

\[ Z'_{h',w',r_4} = \sum_{i=1}^{D} \sum_{j=1}^{D} \sum_{r_3=1}^{R_3} C_{i,j,r_3,r_4} Z_{h_4,w_j,r_3} \]  

\[ Y_{h',w',t} = \sum_{r_4=1}^{R_4} U_{t,r_4}^{(4)} Z'_{h',w',r_4} \]

⇒ Z and Z’ are intermediate tensors of sizes H x W x R₃ and H’ x W’ x R₄
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

\[
M = \frac{D^2ST}{SR_3 + D^2R_3R_4 + TR_4} \quad \text{and} \quad E = \frac{D^2STH'W'}{SR_3HW + D^2R_3R_4H'W' + TR_4H'W'},
\]

\[\Rightarrow \quad M = \text{Compression ratio} \quad \Rightarrow \quad E = \text{Speed-up ratio}\]

- 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

Convolutional Layer 1

Convolutional Layer 2

Convolutional Layer 3

Convolutional Layer 4

Convolutional Layer 5

R1 - Rank of mode 1 matricization
R2 - Rank of mode 2 matricization
R3 - Rank of mode 3 matricization
R4 - Rank of mode 4 matricization
What is Reconstruction Error?

- The distance between original data point and its 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

<table>
<thead>
<tr>
<th>Model</th>
<th>Top-5</th>
<th>Weights</th>
<th>FLOPs</th>
<th>S6</th>
<th>Titan X</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>80.03</td>
<td>61M</td>
<td>725M</td>
<td>117ms</td>
<td>245mJ</td>
</tr>
<tr>
<td>AlexNet*</td>
<td>78.33</td>
<td>11M</td>
<td>272M</td>
<td>43ms</td>
<td>72mJ</td>
</tr>
<tr>
<td>(imp.)</td>
<td>(-1.70)</td>
<td>(×5.46)</td>
<td>(×2.67)</td>
<td>(×2.72)</td>
<td>(×3.41)</td>
</tr>
<tr>
<td>VGG-S</td>
<td>84.60</td>
<td>103M</td>
<td>2640M</td>
<td>357ms</td>
<td>825mJ</td>
</tr>
<tr>
<td>VGG-S*</td>
<td>84.05</td>
<td>14M</td>
<td>549M</td>
<td>97ms</td>
<td>193mJ</td>
</tr>
<tr>
<td>(imp.)</td>
<td>(-0.55)</td>
<td>(×7.40)</td>
<td>(×4.80)</td>
<td>(×3.68)</td>
<td>(×4.26)</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>88.90</td>
<td>6.9M</td>
<td>1566M</td>
<td>273ms</td>
<td>473mJ</td>
</tr>
<tr>
<td>GoogLeNet*</td>
<td>88.66</td>
<td>4.7M</td>
<td>760M</td>
<td>192ms</td>
<td>296mJ</td>
</tr>
<tr>
<td>(imp.)</td>
<td>(-0.24)</td>
<td>(×1.28)</td>
<td>(×2.06)</td>
<td>(×1.42)</td>
<td>(×1.60)</td>
</tr>
<tr>
<td>VGG-16</td>
<td>89.90</td>
<td>138M</td>
<td>15484M</td>
<td>1926ms</td>
<td>4757mJ</td>
</tr>
<tr>
<td>VGG-16*</td>
<td>89.40</td>
<td>127M</td>
<td>3139M</td>
<td>576ms</td>
<td>1346mJ</td>
</tr>
<tr>
<td>(imp.)</td>
<td>(-0.50)</td>
<td>(×1.09)</td>
<td>(×4.93)</td>
<td>(×3.34)</td>
<td>(×3.53)</td>
</tr>
</tbody>
</table>

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, (R3, R4): Tucker-2 rank.)

<table>
<thead>
<tr>
<th>Layer</th>
<th>S/R3</th>
<th>T/R4</th>
<th>Weights</th>
<th>FLOPs</th>
<th>S6</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv1</td>
<td>3</td>
<td>96</td>
<td>35K</td>
<td>105M</td>
<td>15.05 ms</td>
</tr>
<tr>
<td>(imp.)</td>
<td></td>
<td>26</td>
<td>11K</td>
<td>36M(=29+7)</td>
<td>10.19m(=8.28+1.90)</td>
</tr>
<tr>
<td>conv2</td>
<td>48 x 2</td>
<td>128 x 2</td>
<td>307K</td>
<td>224M</td>
<td>24.25 ms</td>
</tr>
<tr>
<td>(imp.)</td>
<td>25 x 2</td>
<td>59 x 2</td>
<td>91K</td>
<td>67M(=2+54+11)</td>
<td>10.53ms(=0.80+7.43+2.30)</td>
</tr>
<tr>
<td>conv3</td>
<td>256</td>
<td>384</td>
<td>885K</td>
<td>150M</td>
<td>18.60ms</td>
</tr>
<tr>
<td>(imp.)</td>
<td>105</td>
<td>112</td>
<td>178K</td>
<td>30M(=5+18+7)</td>
<td>4.85ms(=1.00+2.72+1.13)</td>
</tr>
<tr>
<td>conv4</td>
<td>192 x 2</td>
<td>192 x 2</td>
<td>664K</td>
<td>112M</td>
<td>15.17 ms</td>
</tr>
<tr>
<td>(imp.)</td>
<td>49 x 2</td>
<td>46 x 2</td>
<td>77K</td>
<td>13M(=3+7+3)</td>
<td>4.29ms(=1.55+1.89+0.86)</td>
</tr>
<tr>
<td>conv5</td>
<td>192 x 2</td>
<td>128 x 2</td>
<td>442K</td>
<td>75.0M</td>
<td>10.78ms</td>
</tr>
<tr>
<td>(imp.)</td>
<td>40 x 2</td>
<td>34 x 2</td>
<td>49K</td>
<td>8.2M(=2.6+4.1+1.5)</td>
<td>3.44 ms(=1.15+1.61+0.68)</td>
</tr>
<tr>
<td>fc6</td>
<td>256</td>
<td>4096</td>
<td>37.7M</td>
<td>37.7M</td>
<td>18.94ms</td>
</tr>
<tr>
<td>(imp.)</td>
<td>210</td>
<td>584</td>
<td>6.9M</td>
<td>8.7M(=1.9+4.4+2.4)</td>
<td>5.07ms(=0.85+3.12+1.11)</td>
</tr>
<tr>
<td>fc7</td>
<td>4096</td>
<td>4096</td>
<td>16.8M</td>
<td>16.8M</td>
<td>7.75ms</td>
</tr>
<tr>
<td>(imp.)</td>
<td>301</td>
<td>301</td>
<td>2.4M</td>
<td>2.4M(=1.2+1.2)</td>
<td>1.02 ms(=0.51+0.51)</td>
</tr>
<tr>
<td>fc8</td>
<td>4096</td>
<td>1000</td>
<td>4.1M</td>
<td>4.1M</td>
<td>2.00ms</td>
</tr>
<tr>
<td>(imp.)</td>
<td>195</td>
<td>195</td>
<td>1.0M</td>
<td>1.0M(=0.8+0.2)</td>
<td>0.66ms(=0.44+0.22)</td>
</tr>
</tbody>
</table>
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).
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