Automatic Selection of Tensor Decomposition for Compressing Convolutional Neural Networks
A Case Study on VGG-type Networks

Chia-Chun Liang, Che-Rung Lee

National Tsing Hua University

May 15, 2021
Outline

1. Introduction
   - Tensor Decompositions for CNN

2. Mixed Tensor Decomposition Algorithm
   - Case Studies on VGG11
   - MTD Algorithm

3. Experiments

4. Concluding Remarks
Convolution Neural Networks (CNN) have revolutionized the field of computer vision and became ubiquitous through a range of applications.

Among all operations, the most time-consuming task is 2D convolution, taking 50% to 90% of the computation time for various CNN based models.

Numerous methods for reducing the time of convolution operations have been proposed.

- Fast convolution, octave convolution, split convolution, ...
- Pruning methods, channel pruning, structure preserving pruning, ...
- Low-rank approximations, SVD, tensor decomposition...

Comparing to other methods, tensor decomposition based methods usually can explore more latent relation among structures, and give a better compression ratio.
Mathematically, a tensor is a multilinear map between vector spaces. A tensor may be represented as a potentially multidimensional array. The scalar components of the tensor (or simply its components) are denoted by the indices giving their position in the array. The number of the dimensions of a given tensor is called as order.

We can create subarrays by fixing some of the given tensor’s indices. Fibers are created when fixing all but one index. Slices are created by fixing all but two indices.

A tensor decomposition is any scheme for expressing a tensor as a sequence of elementary operations acting on other, often simpler tensors.

Two commonly used tensor decomposition methods are:
- Canonical Polyadic Decomposition (CPD)
- Tucker Decomposition (TKD)
Tensors in Convolutional Neural Networks

- A convolution layer in CNNs maps a 3-order input tensor $X$ of size $C_{in} \times W_0 \times H_0$ into a 3-order output tensor $Y$ of size $C_{out} \times W_1 \times H_1$ using a 4-order kernel tensor $T$ of size $C_{out} \times C_{in} \times W_f \times H_f$ to represent the weights:
  - $C_{in}$ is the number of input channels;
  - $C_{out}$ is the number of output channels;
  - $W_0$ and $H_0$ are two spatial dimensions for the input data;
  - $W_1$ and $H_1$ are for the output data, and
  - $W_f$ and $H_f$ are for the kernel shape in a convolution layer.

- With these notations, a convolution operation can be expressed as

$$Y(x, y, k) = \sum_{i=1}^{W_f} \sum_{j=1}^{H_f} \sum_{h=1}^{C_{in}} T(k, j, i, h)X(i + x - x_c, j + y - y_c, h), \quad (1)$$

  - $x_c = (W_f - 1)/2$ and $y_c = (H_f - 1)/2$.
  - For simplicity, we can assume $W_f$ and $H_f$ are both odd numbers.
Canonical Polyadic Decomposition

- CPD expresses a tensor as a sum of outer products of rank-one tensors. For an order \( d \) tensor \( A \) of size \( n_1 \times \cdots \times n_d \), a CPD has the following form,

\[
A = \sum_{r=1}^{R} A_r, \tag{2}
\]

where the minimal possible \( R \) is called the \textit{canonical rank}.
- Each \( A_r \) is an outer product of rank-one tensors (ie. vectors) of dimension \( d \),

\[
A_r = A_1^r \otimes A_2^r \otimes \cdots A_d^r, \tag{3}
\]

where \( \otimes \) denotes the Kronecker product operation, and each \( A_i^r \) is a rank-one tensor for \( i = 1 \cdots d \).
- Element-wise, we can calculate the result by

\[
A(i_1, \ldots, i_d) = \sum_{r=1}^{R} A_1(i_1, r)A_2(i_2, r)\cdots A_d(i_d, r). \tag{4}
\]
The Tucker decomposition uses $n$-mode product of a tensor, which is a product of a tensor and a matrix.

- The $k$-mode product of an $d$-order tensor $A$ of size $n_1 \times \ldots, n_k, \ldots \times n_d$ and a matrix $U$ of size $j \times n_k$, is another $d$-order tensor, denoted by $A \times_k U$, of size $n_1 \times \ldots, j, \ldots \times n_d$.
- Each $k$-mode fiber in $A \times_k U$ is generated from the vector-matrix product $v^T U$, where $v$ is a $k$-mode fiber in $A$.

Tucker decomposition factorizes a tensor into a core tensor multiplied by a matrix along each mode,

$$A = G \times_1 A^{(1)} \times_2 A^{(2)} \ldots \times_d A^{(d)},$$

(5)

where $A^{(i)} \in \mathbb{R}^{N_i \times M_i}$ are the factor matrices.

- The upper script in $A^{(i)}$ denotes the matrix in the $i$th-mode.
- The tensor $G \in \mathbb{R}^{M_1 \times \ldots \times M_d}$ is called the core tensor, which has the same order as $A$. 

Chia-Chun Liang, Che-Rung Lee (National Tsing Hua University)  
Automatic Selection of Tensor Decomposition for Compressing Convolutional Neural Networks 
May 15, 2021 7 / 24
Mixed Tensor Decomposition (MTD)

- Given a CNN model $\mathcal{M} = (\{T_i, S_j\})$, where $\{T_i\}$ is a series of tensors formed from convolution layers, and $\{S_j\}$ is the set of other weight and parameters.
  - Let $|\mathcal{M}|$ be the size of model, and $f(\mathcal{M}, X, L)$ be the model accuracy of $\mathcal{M}$ for input data $X$ and their corresponding labels $L$.

- The goal is to minimize the size of a compressed model $\mathcal{M}' = (\{T'_i, S_j\})$, where $T'_i$ is obtained from $T_i$ by some transformations $D_i = \{TKD, CPD, Identity\}$.

$$\begin{align*}
\min_{D_i} |\mathcal{M}'| \\
\text{s.t. } f(\mathcal{M}, X, L) - f(\mathcal{M}', X, L) \leq \epsilon,
\end{align*}$$

where $\epsilon$ is a small number for the tolerance of accuracy drop, which means the model accuracy of $\mathcal{M}'$ should be maintained.
The Computational Challenges of MTD

- Each layer can be decomposed by one of three possible methods. If there are $N$ layers, the possible combination is $3^N$.

- For Tucker decomposition, the rank can be determined by the Variational Bayesian Matrix Factorization (VBMF) method. But for CPD, there is no good algorithm to determine a proper rank for decomposed network.

- The auto-tuning method can be used to solve this problem.
  - For each layer, decide which decomposition method should be used, and decide the proper rank automatically.
  - We first performed a case study on the VGG typed network, and based on the case study we proposed an automatic strategy.
VGG Network

- VGG network is invented by the Visual Geometry Group in Oxford University. This architecture is the runner up of ILSVR2014 in the classification task while the winner is GoogLeNet.
- VGG uses a lot of 3x3 convolution layers in the network, and partitions them into blocks using pooling.
- A typical VGG-16 network architecture. ¹

¹Figure is from http://deanhan.com/2018/07/26/vgg16/
Case Study on VGG11

- We benchmarked VGG11, which has eight convolution layers in five blocks, indexed by block ID and layer ID.
  - The used dataset is CIFAR-10.
  - All networks are implemented with *PyTorch*.
  - The tensor decomposition is performed using the library *Tensorly*.
  - The rank of TKD is decided by VBMF; the rank of CPD is decided by keeping its compression ratio (CR) similar to TKD’s.
  - The baseline is the uncompressed VGG11, whose accuracy is 88.9%.

- We performed the following studies.
  - Decompose each single layer of VGG11 using CPD or TKD.
  - Decompose all layers of VGG11 using CPD or TKD in the sequential order (forward) and the reverse order (backward).
  - Evaluate the accuracy and compression ratio (CR).
Decompose Single Layer

The compression ratio and model accuracy for VGG11.

<table>
<thead>
<tr>
<th>Conv. layer</th>
<th>Ipt chnl</th>
<th>Opt chnl</th>
<th>Tucker R</th>
<th>CR</th>
<th>Acc</th>
<th>CPD R</th>
<th>CR</th>
<th>Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1_1</td>
<td>3</td>
<td>64</td>
<td>(2, 12)</td>
<td>1.75</td>
<td>90.4%</td>
<td>9</td>
<td>1.59</td>
<td>91.1%</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>128</td>
<td>(32, 43)</td>
<td>3.70</td>
<td>90.9%</td>
<td>44</td>
<td>3.67</td>
<td>88.8%</td>
</tr>
<tr>
<td>2_1</td>
<td>128</td>
<td>256</td>
<td>(54, 59)</td>
<td>5.82</td>
<td>90.8%</td>
<td>65</td>
<td>5.86</td>
<td>89.4%</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>256</td>
<td>(61, 50)</td>
<td>10.56</td>
<td>90.4%</td>
<td>63</td>
<td>10.52</td>
<td>88.8%</td>
</tr>
<tr>
<td>3_1</td>
<td>256</td>
<td>512</td>
<td>(90, 103)</td>
<td>7.41</td>
<td>91.0%</td>
<td>112</td>
<td>7.31</td>
<td>88.5%</td>
</tr>
<tr>
<td>3_2</td>
<td>512</td>
<td>512</td>
<td>(123, 126)</td>
<td>8.84</td>
<td>91.1%</td>
<td>143</td>
<td>8.77</td>
<td>90.8%</td>
</tr>
<tr>
<td>4_1</td>
<td>512</td>
<td>512</td>
<td>(75, 75)</td>
<td>18.52</td>
<td>91.4%</td>
<td>84</td>
<td>18.38</td>
<td>91.3%</td>
</tr>
<tr>
<td>4_2</td>
<td>512</td>
<td>512</td>
<td>(61, 65)</td>
<td>23.55</td>
<td>91.3%</td>
<td>69</td>
<td>23.78</td>
<td>91.5%</td>
</tr>
</tbody>
</table>

**CR** = \[\frac{\text{No. parameters in the original layer}}{\text{No. parameters in the compressed layer}}\], \(R\) is the rank.
We also studied the influence of CPD’s rank to the model accuracy.
The results show that in the earlier layer, conv2_1, no matter what rank is, the accuracy cannot be fully restored.
On the other hand, for later layer, conv5_2, even a small rank can fully restore the model accuracy.
The results suggest the later layers have more redundancy, and using CPD with lower ranks does not hurt the model accuracy.
## Decompose Multiple Layers

<table>
<thead>
<tr>
<th></th>
<th>TKD</th>
<th>CPD</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rank</strong></td>
<td><strong>VBMF</strong></td>
<td><strong>max/3</strong></td>
</tr>
<tr>
<td>order</td>
<td>seq.</td>
<td>seq.</td>
</tr>
<tr>
<td>Acc</td>
<td>89%</td>
<td>78%</td>
</tr>
<tr>
<td>CR</td>
<td>11.84 ×</td>
<td>6.01 ×</td>
</tr>
</tbody>
</table>

### Accuracy (%)

#### Index of Convolutional layer

**Sequential order**

- Baseline
- CP, rank=max//3
- CP, rank=max//10
- CP, rank=max//20
- Tucker

**Reverse order**

- Baseline
- CP, rank=max//3
- CP, rank=max//10
- CP, rank=max//20
- Tucker

(a) Sequential order

(b) Reverse order
Lessons Learned from the Case Study

1. For a layer, if both TKD and CPD can keep similar model accuracy, using CPD for compression, because it can give better compression ratio.

2. If CPD cannot maintain the model accuracy, using TKD; if TKD also fails, using Identity (no compression).

3. Compressing the model in the reverse order. Although the results show that sequential order gives better model accuracy, the plots in the previous page indicate the accuracy of CPD remains high in the first few layers in the reverse order. On the other hand, in the sequential order, the model accuracy of CPD drops quickly from the beginning. Thus, if CPD is tried first, one should use the reverse order.
Mixed Tensor Decomposition Algorithm

Let $M$ be the model, and $f$ be the original model accuracy. The high level description of the algorithm

1. Decompose the network layer by layer in the reverse order.
2. For each layer $L_i$, perform TKD first. Assume the decomposed layer is $L_i^T$ and the model accuracy after TKD is $f_i^T$.
3. If $f - f_i^T > \epsilon$, this means even TKD cannot keep the desired accuracy, so we terminate the compression. This means the rest of the layers are un-compressed.
4. For the same layer $L_i$, perform CPD, and get the decomposed layer $L_i^C$ and model accuracy $f_i^C$.
5. If $f - f_i^C > \epsilon$ or $|L_i^C| > |L_i^T|$, this means CPD cannot achieve the desired accuracy or the model size compressed by CPD is larger than that of TKD. In that case, we use TKD for layer $L_i$.
6. Continue step 2-step 5 until all the layers are complete.
Rank Selection and Fine Tuning

- The rank of TKD is determined by VBMF. ²
- For CPD, numerical rank selection is an NP-hard problem. In addition, high numerical rank does not imply high model accuracy. We used the result of TKD and an empirical method: Iterative Two-Pass Decomposition. ³ to decide the rank of CPD.
- The fine-tuning process follows the suggestions given by Astrid and Lee ⁴, which gradually transforms a dense network into the decomposed one layer by layer.
- We found that the network is retrained after the decomposition of each layer, rather than the decomposition of the entire network, the accuracy drops would be negligible.

Experiments

- Our experiments run on a system with an Intel i7-4790 CPU and a NVIDIA GeForce GTX1080 Ti GPU.
- All networks are implemented with PyTorch.
- The tensor decomposition is performed using the library Tensorly.
- We used the widely-used VGG-11 and VGG-16, which have 8 and 13 convolution layers respectively.
- The used dataset is CIFAR-10, for which VGG11 can achieve 88.9% accuracy and VGG16 can reach 90.63% accuracy.
- We compared MTD with other four tensor decomposition methods: TKD, CPD, iterative two pass method, and MUSCO.\(^5\)

---

\(^5\)Gusak, J., K holyavchenko, M., Ponomarev, E., Markeeva, L., Blagoveschensky, P., Cichocki, A., Oseledets, I.V.: Automated Multi-stage Compression of Neural Networks. In: 2019 IEEE/CVF International Conference on Computer Vision Workshops
## Compared Results for VGG11

<table>
<thead>
<tr>
<th>VGG11</th>
<th>CR</th>
<th>CRFS</th>
<th>Accuracy</th>
<th>ITR (C)</th>
<th>ITR (G)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>-</td>
<td>-</td>
<td>88.90%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Tucker</td>
<td>11.82 ×</td>
<td>1.43 ×</td>
<td>89.41% (+0.07%)</td>
<td>2.24 ×</td>
<td>1.41 ×</td>
</tr>
<tr>
<td>CPD</td>
<td>11.76 ×</td>
<td>1.43 ×</td>
<td>84.64% (-4.26%)</td>
<td>1.75 ×</td>
<td>1.50 ×</td>
</tr>
<tr>
<td>Two pass</td>
<td>1.84 ×</td>
<td>1.38 ×</td>
<td>89.21% (+0.31%)</td>
<td>0.97 ×</td>
<td>1.34 ×</td>
</tr>
<tr>
<td>MUSCO</td>
<td>23.55 ×</td>
<td>1.46 ×</td>
<td>88.33% (-0.57%)</td>
<td>2.28 ×</td>
<td>1.54 ×</td>
</tr>
<tr>
<td>MTD</td>
<td>32.59 ×</td>
<td>1.47 ×</td>
<td>88.06% (-0.84%)</td>
<td>1.99 ×</td>
<td>1.56 ×</td>
</tr>
</tbody>
</table>

\[
CR = \frac{\text{No. parameters in all convolution layers}}{\text{No. parameters in all compressed convolution layers}}
\]

\[
CRFS = \frac{\text{File size for the original model}}{\text{File size for the compressed model}}
\]

\[
ITR = \frac{\text{Inference time of the original model}}{\text{Inference time of the compressed model}}
\]

(C) for CPU and (G) for GPU
## Compared Results for VGG16

<table>
<thead>
<tr>
<th>Method</th>
<th>CR</th>
<th>CRFS</th>
<th>Accuracy</th>
<th>ITR (C)</th>
<th>ITR (G)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>-</td>
<td>-</td>
<td>90.63%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Tucker</td>
<td>13.53 ×</td>
<td>1.68 ×</td>
<td>91.02% (+0.39%)</td>
<td>2.21 ×</td>
<td>1.53 ×</td>
</tr>
<tr>
<td>CPD</td>
<td>13.48 ×</td>
<td>1.68 ×</td>
<td>87.02% (-3.61%)</td>
<td>1.38 ×</td>
<td>1.50 ×</td>
</tr>
<tr>
<td>Two pass</td>
<td>2.02 ×</td>
<td>1.59 ×</td>
<td>89.57% (-1.06%)</td>
<td>1.69 ×</td>
<td>1.36 ×</td>
</tr>
<tr>
<td>MUSCO</td>
<td>14.05 ×</td>
<td>1.69 ×</td>
<td>87.07% (-3.56%)</td>
<td>2.05 ×</td>
<td>1.46 ×</td>
</tr>
<tr>
<td>MTD</td>
<td>37.25 ×</td>
<td>1.76 ×</td>
<td>89.80% (-0.83%)</td>
<td>2.19 ×</td>
<td>1.64 ×</td>
</tr>
</tbody>
</table>

**CR** = \[ \frac{\text{No. parameters in all convolution layers}}{\text{No. parameters in all compressed convolution layers}} \]

**CRFS** = \[ \frac{\text{File size for the original model}}{\text{File size for the compressed model}} \]

**ITR** = \[ \frac{\text{Inference time of the original model}}{\text{Inference time of the compressed model}} \]

(C) for CPU and (G) for GPU
## Selected Methods for VGG11 and VGG16

<table>
<thead>
<tr>
<th>layer.</th>
<th>VGG11</th>
<th></th>
<th>VGG16</th>
<th></th>
<th>method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Conv.</td>
<td>Ipt</td>
<td>Opt</td>
<td>Conv.</td>
<td>Ipt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv1_1</td>
<td>3</td>
<td>64</td>
<td>ID</td>
<td>conv1_1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>conv1_2</td>
<td>64</td>
<td>TKD</td>
<td></td>
<td>64</td>
</tr>
<tr>
<td>conv2_1</td>
<td>64</td>
<td>128</td>
<td>TKD</td>
<td>conv2_1</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>conv2_2</td>
<td>128</td>
<td>TKD</td>
<td></td>
<td>128</td>
</tr>
<tr>
<td>conv3_1</td>
<td>128</td>
<td>256</td>
<td>TKD</td>
<td>conv3_1</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>conv3_2</td>
<td>256</td>
<td>TKD</td>
<td></td>
<td>256</td>
</tr>
<tr>
<td>conv3_3</td>
<td>256</td>
<td>512</td>
<td>TKD</td>
<td>conv4_1</td>
<td>256</td>
</tr>
<tr>
<td>conv4_1</td>
<td>256</td>
<td>512</td>
<td>TKD</td>
<td>conv4_2</td>
<td>512</td>
</tr>
<tr>
<td>conv4_2</td>
<td>512</td>
<td>512</td>
<td>CPD</td>
<td>conv4_3</td>
<td>512</td>
</tr>
<tr>
<td>conv5_1</td>
<td>512</td>
<td>512</td>
<td>CPD</td>
<td>conv5_1</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td>conv5_2</td>
<td>512</td>
<td>CPD</td>
<td></td>
<td>512</td>
</tr>
<tr>
<td>conv5_2</td>
<td>512</td>
<td>512</td>
<td>CPD</td>
<td>conv5_3</td>
<td>512</td>
</tr>
</tbody>
</table>
Accuracy of VGG11 and VGG16

(a) VGG11

(b) VGG16
Concluding Remarks

- We used the auto-tuning technique to develop a mixed tensor decomposition algorithm (MTD) which selects the most suitable decomposition method for each layer.
  - We studied the properties of TKD and CPD applying to the compression of convolutional neural networks.
  - Based on the observation, we can design an automatic strategy to select the decomposition method for each layer.

- Experimental results show that MTD can achieve a great compression ratio without sacrificing the accuracy.
  - For VGG11, the compression ratio is $32 \times$, and for VGG16, it is $37 \times$.
  - The accuracy drops for both models are less than 1%.
  - Comparing to existing methods, the compression ratio of MTD is more than two times better than that of other methods.
Future Work

- The proposed method is orthogonal to many other methods, such as MUSCO, so that they can be applied together.
- The current work is only verified on VGG-based networks. We believe our method or framework can be extended for more complex network models, such as ResNet or Inception, or even for the models of other tasks other than the convolution operations.
- There are other types of tensor decompositions, such as tensor train factorization or block decomposition methods. How to integrate them into the model compression is still a challenge.
- There are many fundamental problems for tensor decompositions, such as rank selection and model retraining, which require more investigation to solve.