Sharing Residual Units Through Collective Tensor Factorization To Improve Deep Neural Networks

Yunpeng Chen[1], **Xiaojie Jin**[1], **Bingyi Kang**[1], **Jiashi Feng**[1], **Shuicheng Yan**[2,1]
[1]National University of Singapore
[2]Qihoo 360 AI Institute
{chenyunpeng, xiaojie.jin, kang}@u.nus.edu, elefjia@nus.edu.sg,
yanshuicheng@360.cn

Abstract

The residual unit and its variations are wildly used in building very deep neural networks for alleviating optimization difficulty. In this work, we revisit the standard residual function as well as its several successful variants and propose a unified framework based on *tensor Block Term Decomposition* (BTD) to explain these apparently different residual functions from the tensor decomposition view. With the BTD framework, we further propose a novel basic network architecture, named the Collective Residual Unit (CRU). CRU further enhances parameter efficiency of deep residual neural networks by sharing core factors derived from *collective tensor factorization* over the involved residual units. It enables efficient knowledge sharing across multiple residual units, reduces the number of model parameters, lowers the risk of over-fitting, and provides better generalization ability. Extensive experimental results show that our proposed CRU network brings outstanding parameter efficiency—it achieves comparable classification performance with ResNet-200 while using a model size as small as ResNet-50 on the ImageNet-1k and Places365-Standard benchmark datasets.

1 Introduction

Deep residual networks (ResNets) [He *et al.*, 2016a] are built by stacking multiple *residual units* and have achieved remarkable success in many applications, *e.g.*, image segmentation [Wu *et al.*, 2016; Zhao *et al.*, 2016] and object localization [Ren *et al.*, 2015; Li *et al.*, 2016]. The effectiveness of ResNets is largely attributed to the internal identity mappings and residual functions. Although He *et al* [2016b] have explained the importance of identity mapping in alleviating optimization difficulty for training ResNets, understandings on the residual functions are still rare. Besides, performance gain by stacking a number of residual units usually does not well compensate the model size increase in practice. In this work, we aim to develop new and unified explanations on different variants of the residual function. Then, founded on the explanations and analysis, we propose a novel basic residual unit that further improves parameter efficiency as well as learning capacity over the vanilla residual units.

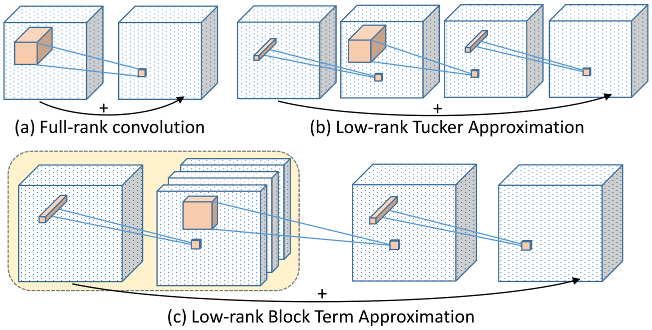

+
(a) Full-rank convolution
+
(b) Low-rank Tucker Approximation
+
(c) Low-rank Block Term Approximation

Figure 1: Residual units with different convolution kernel approximations. **(a)** The standard one w/o approximation: a full rank convolution layer. **(b)** Approximation of (a) by using the low-rank Tucker Decomposition. **(c)** An approximation of (a) by using low-rank Block Term Decomposition. The new proposed CRU shares the convolution kernel in the first two layers (highlighted in yellow) across different residual functions.

We develop the new residual unit by revisiting residual unit[1] architectures. In the seminal residual network paper [He *et al.*, 2016a], the residual function was designed as a 3-layer bottleneck architecture. The three layers consist of 1×1, 3×3 and 1×1 convolutional filters respectively, where the second complex 3×3 convolutional layer usually has less channels than its adjacent two simple 1×1 convolutional layers. Since then different variants of the residual unit have been developed. For instance, the Wide Residual Network (WRN) [Zagoruyko and Komodakis, 2016] increases the number of channels in the second 3×3 convolutional layer and finds it learns better feature; the ResNeXt [Xie *et al.*, 2016] divides the second 3×3 convolution layer into several groups to enhance the model's learning capacity. However, they are built upon different motivations and have different architectures, making them hard to analyze together.

In our work, we revisit these variants of residual functions proposed in [He *et al.*, 2016a; 2016b; Zagoruyko and Ko-

[1] A residual unit, *i.e.* $f(x)+x$, consists of a residual function, *i.e.* $f(x)$ and a residual term, *i.e.* x.

modakis, 2016; Xie *et al.*, 2016] which are highly modularized and widely used in practice. We find that all of these residual functions, although apparently different, can be unified by viewing them through the lens of tensor Block Term Decomposition [De Lathauwer, 2008] when we ignore the intermediate nonlinear activation. By applying such tensor decomposition, a high order tensor (*e.g.*, a collection of multiple convolutional kernels) can be decomposed into summation of multiple low-rank Tuckers [Tucker, 1966]. More importantly, by simply varying the rank of these Tuckers, one can obtain realizations of different residual functions mentioned above.

With the unified BTD framework and motivated by *collective matrix factorization*[2] [Singh and Gordon, 2008], we further propose a novel Collective Residual Unit (CRU), where a subset of the decomposed factors are shared across layers to enable efficient and general cross-layer knowledge sharing for different residual units, as illustrated in Figure 1 (c). With such a novel collective tensor factorization induced unit, the knowledge learned by one specific residual unit can be effectively reused by others through sharing core factors (highlighted in yellow in Figure 1 (c)), which further enhances the parameter efficiency and consequently improves the model generalization ability.

The main contributions of our work can be summarized as follows:

1) We provide new perspectives for explaining and understanding the popular residual networks and we unify existing variants of residual functions under a single framework for better understanding.

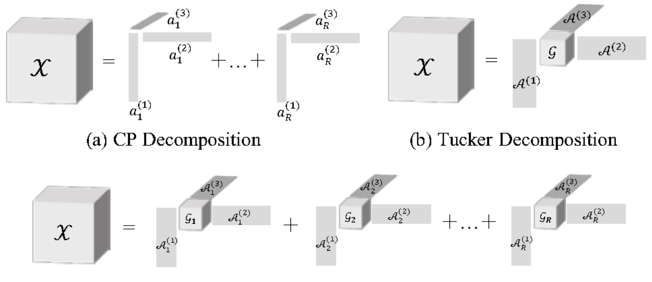
2) Founded on the analysis, we develop a new and principled knowledge sharing method for building residual units. Based on this method, we design a novel Collective Residual Unit (CRU) which brings higher parameter efficiency than existing residual units.

3) The CRU Network achieves higher parameter efficiency compared with the state-of-the-art models on two large-scale benchmark datasets with better performance.

2 Related Work

Tensor decomposition has been introduced to neural networks by many works [Oseledets, 2011; Rigamonti *et al.*, 2013; Lebedev *et al.*, 2014; Wang and Cheng, 2016] and the idea of sharing knowledge across different convolution layers has been proposed ever since the emergence of recurrent neural networks. In this section, we briefly review the related works in both areas and highlight the novelty of our work.

Mathematically, given a tensor, there are several different ways to factorize it. As can be seen in Figure 2, the CANDECOMP/PARAFAC (CP) Decomposition factorizes a tensor as a summation of several tensors with rank equal to one; the Tucker Decomposition factorizes a tensor as a core tensor with multiple 2d matrices. More recently, researchers have combined the CP Decomposition and Tucker Decomposition and proposed a more general decomposition method called *Block Term Decomposition* [De Lathauwer, 2008; Kolda and Bader, 2009], where a high-order tensor is approximated in a sum of several low-rank Tuckers. When the rank of each Tucker is equal to one, it degrades to CP Decomposition; when the number of Tuckers equals one, it degrades to Tucker Decomposition. In [Garipov *et al.*, 2016] and [Cohen *et al.*, 2016], the authors demonstrated that CNNs can be analyzed through tensor factorization, which inspires our work.



(a) CP Decomposition

(b) Tucker Decomposition

(c) Block Term Decomposition

Figure 2: Illustration on factorization of a third order tensor with different tensor decomposition methods.

One of the many important applications of tensor decomposition is to increase the parameter efficiency. [Lebedev *et al.*, 2014] proposed to compress convolutional layers of a trained model by using CP Decomposition. [Jaderberg *et al.*, 2014] proposed to approximate an initial convolutional kernel tensor by two low-rank components. Similarly, [Garipov *et al.*, 2016] proposed to decompose fully connected layers by using Tensor-Train Decomposition [Oseledets, 2011]. These methods either do not support end-to-end training or lack experiments to prove their effectiveness on very large datasets. Sharing knowledge across the neural network is another efficient way to reduce redundancy and increase parameter efficiency. Recurrent Neural Network (RNN) can be seen as a good example where weights are shared across time steps. The work [Liao and Poggio, 2016] generalized both RNN and ResNet architectures by sharing the entire residual unit throughout the CNNs. [Eigen *et al.*, 2013] proposed to repeat the convolution operation for several times to capture context information. However, there is still a gap in accuracy between the state-of-the-arts and the methods mentioned above, which indicates potential defects within these recurrent architectures and doubts about usefulness of sharing knowledge across layers for the image classification task.

Different from these existing works, we make the first attempt to introduce the Block Term Decomposition into deep learning and derive its corresponding convolutional architectures, which, in turn, forms an unrevealed perspective on various residual functions. Based on this finding, we further propose a novel architecture, Collective Residual Unit, to enhance the parameter efficiency of residual networks. This is achieved by sharing some of the decomposed factors across residual units, so that features learned from one unit can directly help the feature learning in other units. More importantly, the proposed method achieves state-of-the-art accuracy without requiring a pre-trained model and can be easily optimized in an end-to-end manner.

[2] In our work, following the naming conventions in tensor analysis, we interchangeably use *factorization* and *decomposition*.

3 Tensor Decomposition View on Residual Functions

In this section, we introduce the tensor Block Term Decomposition and analyze existing variants of the residual function [He *et al.*, 2016a; 2016b; Zagoruyko and Komodakis, 2016; Xie *et al.*, 2016] under this view, which lays the foundation for developing the CRU network.

3.1 Tensor Block Term Decomposition

A *tensor* is a multi-dimensional array and the *order* of a tensor is the number of its dimensions. Throughout the paper, we use calligraphic capital letters, *e.g.*, $\mathcal{A} \in \mathbb{R}^{d_1 \times d_2 \times \ldots \times d_N}$, to denote an N-th order tensor, and use $d_1, d_2, ..., d_N$ to denote the size of each mode.

Tensor Block Term Decomposition (BTD) was first proposed by De Lathauwer [2008], which factorizes a high order tensor into the sum of multiple low-rank Tuckers [Tucker, 1966]. More concretely, given an N-th order tensor $\mathcal{X} \in \mathbb{R}^{d_1 \times d_2 \times \ldots \times d_N}$, BTD factorizes it into the sum of R rank-$(d_1^*, d_2^*, ..., d_N^*)$ terms:

$$\begin{aligned} &\mathcal{X} = \sum_{r=1}^{R} \mathcal{G}_r \times_1 \mathcal{A}_r^{(1)} \times_2 \mathcal{A}_r^{(2)} \times_3 \ldots \times_N \mathcal{A}_r^{(N)} \ , \\ &\text{where} \begin{cases} \mathcal{G}_r \ \in \mathbb{R}^{d_1^* \times d_2^* \times \ldots \times d_N^*}, \\ \mathcal{A}_r^{(n)} \in \mathbb{R}^{d_n \times d_n^*}, \ n = \forall 1, ..., N. \end{cases} \end{aligned} \tag{1}$$

Here, $\mathcal{G}$ is known as the *core tensor* and $\times_n$ denotes the *mode-n* product [De Lathauwer, 2008]. Figure 2 (c) illustrates the block term decomposition on a third order tensor.

The block term decomposition in Eqn. (1) can be written in the following element-wise form:

$$\begin{aligned} \mathcal{X}(i_1, i_2, ..., i_N) = \sum_{r=1}^{R} \sum_{j_1, \ldots, j_N} & \mathcal{G}_r(j_1, j_2, ..., j_N) \\ & \mathcal{A}_r^{(1)}(i_1, j_1) \mathcal{A}_r^{(2)}(i_2, j_2) ... \mathcal{A}_r^{(N)}(i_N, j_N) \ , \end{aligned} \tag{2}$$

where $i_n \in \{1, ..., d_n\}$, $j_n \in \{1, ..., d_N^*\}$ and $n \in \{1, ..., N\}$. We will use this element-wise form in the following analysis.

3.2 From BTD to Convolutional Bottleneck Architectures

Convolutional kernels in one layer can be formulated as a 4th order tensor $\mathcal{X} \in \mathbb{R}^{d_1 \times d_2 \times d_3 \times d_4}$, where d_1, d_2 represent size of the filter kernel and d_3, d_4 denote the number of the input and output channels. Thus we can decompose such a tensor following the above introduced Block Term Decomposition. Usually the kernel size d_1 and d_2 are very small, *e.g.*, $d_1 = d_2 = 3$. Therefore, further decomposing $\mathcal{X}$ along these two modes is unnecessary. For this reason, we apply the Block Term Decomposition in Eqn. (1) to the other two modes:

$$\begin{aligned} &\mathcal{X} = \sum_{r=1}^{R} \mathcal{G}_r \times_3 \mathcal{A}_r^{(3)} \times_4 \mathcal{A}_r^{(4)} \ , \\ &\text{where} \begin{cases} \mathcal{G}_r \in \mathbb{R}^{d_1 \times d_2 \times d_3^* \times d_4^*}, \\ \mathcal{A}_r^{(3)} \in \mathbb{R}^{d_3 \times d_3^*}, \\ \mathcal{A}_r^{(4)} \in \mathbb{R}^{d_4 \times d_4^*}. \end{cases} \end{aligned} \tag{3}$$

Let $\mathcal{U} \in \mathbb{R}^{w \times h \times d_3}$ denote the input data to the convolutional filters $\mathcal{X}$. Then the convolutional operation along two dimensions will output the following tensor $\mathcal{V} \in \mathbb{R}^{w \times h \times d_4}$:

$$\begin{aligned} &\mathcal{V}(x, y, c) \\ &= \sum_{m=1}^{d_3} \sum_{i=x-\delta_1}^{x+\delta_1} \sum_{j=y-\delta_2}^{y+\delta_2} \mathcal{X}(i - x + \delta_1, j - y + \delta_2, m, c) \mathcal{U}(i, j, m), \end{aligned} \tag{4}$$

where δ_1 and δ_2 denote "half-width" of the kernel size on each dimension. By substituting $\mathcal{X}$ with its element-wise factorization form of Eqn. (2), the output can be rewritten as

$$\begin{aligned} &\mathcal{V}(x, y, c) \\ &= \sum_{m=1}^{d_3} \sum_{i=x-\delta_1}^{x+\delta_1} \sum_{j=y-\delta_2}^{y+\delta_2} \Bigg[\sum_{r=1}^{R} \sum_{q=1}^{d_4^*} \sum_{p=1}^{d_3^*} \mathcal{G}_r(i - x + \delta_1, j - y + \delta_2, p, q) \\ &\qquad \mathcal{A}_r^{(3)}(m, p) \mathcal{A}_r^{(4)}(c, q) \Bigg] \mathcal{U}(i, j, m) \\ &= \sum_{r=1}^{R} \sum_{q=1}^{d_4^*} \Bigg[\sum_{p=1}^{d_3^*} \sum_{i=x-\delta_1}^{x+\delta_1} \sum_{j=y-\delta_2}^{y+\delta_2} \mathcal{G}_r(i - x + \delta_1, j - y + \delta_2, p, q) \\ &\qquad \Big(\sum_{m=1}^{d_3} \mathcal{U}(i, j, m) \mathcal{A}_r^{(3)}(m, p) \Big) \Bigg] \mathcal{A}_r^{(4)}(c, q). \end{aligned} \tag{5}$$

By introducing $\mathcal{T}^{(1)} \in \mathbb{R}^{w \times h \times d_3^*}$ and $\mathcal{T}^{(2)} \in \mathbb{R}^{w \times h \times d_4^*}$ defined as below to denote intermediate results,

$$\mathcal{T}_r^{(1)}(i, j, p) \triangleq \sum_{m=1}^{d_3} \mathcal{U}(i, j, m) \mathcal{A}_r^{(3)}(m, p), \tag{6}$$

$$\begin{aligned} \mathcal{T}_r^{(2)}(x, y, q) \triangleq & \sum_{p=1}^{d_3^*} \sum_{i=x-\delta_1}^{x+\delta_1} \sum_{j=y-\delta_2}^{y+\delta_2} \mathcal{T}_r^{(1)}(i, j, p) \\ & \mathcal{G}_r(i - x + \delta_1, j - y + \delta_2, p, q), \end{aligned} \tag{7}$$

Eqn. (5) can be simplified as

$$\mathcal{V}(x, y, c) = \sum_{r=1}^{R} \sum_{q=1}^{d_4^*} \mathcal{T}_r^{(2)}(x, y, q) \mathcal{A}_r^{(4)}(c, q). \tag{8}$$

Eqn. (6) to Eqn. (8) show the corresponding architecture when adopting such a Block Term Decomposition on convolutional kernels $\mathcal{X}$. Figure 1 (c) shows an example of the corresponding convolutional architectures. Specifically, the input tensor is first convoluted by a 1×1 convolution kernel (Eqn. (6)) and then passed to a group convolutional layer [Krizhevsky *et al.*, 2012] which evenly separates the input tensor into R groups along the third mode and conducts convolution operation within each group separately (Eqn. (7)). After that, the results are mapped through 1×1 convolutional kernel and aggregated into the final output (Eqn. (8)).

Interestingly, under this unified framework shown in Eqn. (6) to Eqn. (8) (and essentially Eqn. (3)), we will show that such a three-layer architecture is not only closely related to the bottleneck architecture used in various residual functions but also can degenerate into certain residual functions by varying the architecture parameters (including $R, d_3, d_3^*.d_4, d_4^*$).

3.3 Unifying Residual Functions

In this section, we illustrate how Eqn. (6) to Eqn. (8) unify a broad variety of residual functions and provide deeper understanding on the recently proposed ResNeXt [Xie *et al.*, 2016] architecture. W.l.o.g. we ignore all activation functions for clearer explanation at first.

The conventional residual functions proposed in [He *et al.*, 2016a; 2016b; Zagoruyko and Komodakis, 2016] are special cases of the above unified framework when

$$\begin{cases} R &= 1, \\ d_3 &= d_4 = \text{number of input channels}, \\ d_3^* &= d_4^* = \text{width of the bottleneck}. \end{cases} \tag{9}$$

Figure 1 (b) demonstrates the case when R=1 that is exactly the form of vanilla residual function [He *et al.*, 2016a; 2016b] and is also used in the wide residual network [Zagoruyko and Komodakis, 2016]. More specifically, the tensor $\mathcal{A}^{(3)}$ in Eqn. (3) corresponds to the first 1×1 convolutional kernel tensor where the numbers of input and output channels are d_3 and d_3^* respectively. The tensor $\mathcal{A}^{(4)}$ corresponds to 1×1 convolutional kernels of the last layer and $\mathcal{G}$ corresponds to the 3×3 convolutional kernels of the second layer. The wide residual unit actually uses a higher-rank core tensor compared with the standard residual unit.

Moreover, the outer summation of Eqn. (8) can be seen as an aggregation of multiple transformations. In [Xie *et al.*, 2016], R is named as *cardinality* and it has been found that increasing *cardinality* can help increase the networks' learning capacity, but their work did not explain what *cardinality* really is. In our work, we find that R actually corresponds to the number of low-rank Tuckers in BTD, as shown in Eqn. (3). Increasing *cardinality* is equivalent to adding more low-rank Tuckers, *i.e.* $\mathcal{G}_r \times_3 \mathcal{A}_r^{(3)} \times_4 \mathcal{A}_r^{(4)}$. Here, we unify the ResNeXt [Xie *et al.*, 2016] function in BTD framework with the following parameter choice:

$$\begin{cases} R &= \text{cardinality}, \\ d_3 &= d_4 = \text{number of input channels}, \\ d_3^* &= d_4^* = \frac{\text{width of the bottleneck}}{R}, \end{cases} \tag{10}$$

where the *width of the bottleneck* refers to the number of channels for the second 3×3 group convolutional layer.

Most residual functions have Batch Normalization (BN) layers before (or after) the convolutional layer to avoid the covariance shift problem [Ioffe and Szegedy, 2015]. However, we note that adding the batch normalization layer does not change the results derived as above, since the BN operation is element-wisely linear and can be absorbed into the nearest convolutional kernel tensor. The activation functions that we ignored earlier can be seen as an intermediate layer for increasing the non-linearity of a unit.

4 Collective Residual Unit Networks

In this section, we introduce a novel residual unit, Collective Residual Unit (CRU), which is based on collective tensor factorization, to further increase the parameter efficiency of residual function by sharing some of the BTD decomposed factors across residual units using Collective Tensor Factorization.

4.1 Collective Tensor Factorization

A highly modularized deep residual network consists of residual units with the same architecture. Removing any one of the residual units will not result in obvious performance drop [Veit *et al.*, 2016]. This indicates that some critical features (*i.e.*, knowledge for decision) are learned by different residual units for multiple times and redundant knowledge exists across residual units. Motivated by this observation, we propose to reduce the redundancy by reusing (*i.e.*, sharing) knowledge across residual functions. To this end, we simultaneously factorize multiple convolutional operators by treating them as tensors and sharing core factors across them. We name this approach as *collective tensor factorization*.

In particular, for a residual network with L similar residual functions, we concatenate each corresponding convolutional kernel tensor $\mathcal{X}_l \in R^{d_1 \times d_2 \times d_3 \times d_4}$ along the 4th mode and form $\mathcal{X}^+ = [\mathcal{X}_1, \mathcal{X}_2, ..., \mathcal{X}_L]$, where $\mathcal{X}^+ \in R^{d_1 \times d_2 \times d_3 \times (L * d_4)}$. Then we factorize $\mathcal{X}^+$ via BTD with rank-$(\cdot, \cdot, d_3^*, d_4^*)$:

$$\mathcal{X}^+ = \sum_{r=1}^{R} \mathcal{G}_r \times_3 \mathcal{A}_r^{(3)} \times_4 \mathcal{A}_r^{(4)} , \\ \text{where, } \begin{cases} \mathcal{G}_r \in \mathbb{R}^{d_1 \times d_2 \times d_3^* \times d_4^*}, \\ \mathcal{A}_r^{(3)} \in \mathbb{R}^{d_3 \times d_3^*}, \\ \mathcal{A}_r^{(4)} \in \mathbb{R}^{(L * d_4) \times d_4^*}. \end{cases} \tag{11}$$

This is equivalent to decomposing each $\mathcal{X}_l$ with rank-$(\cdot, \cdot, d_3^*, d_4^*)$ separately and then sharing the first two factor terms.

By the above collective tensor factorization, parameters are shared across different residual units through the shared $\mathcal{G}_r$ and $\mathcal{A}_r^{(3)}$. The critical knowledge learned by one unit can thus be directly used by other units. Meanwhile during the propagation, gradients gathered from multiple units will be summed together to update these shared parameters, also leading to more efficient learning.

Figure 3 (left) shows the corresponding convolution structure, where the first 1×1 convolutional layer and the second 3×3 convolutional layer are shared across L different layers, and meanwhile each layer has its own individual $\mathcal{A}_r^{(4)}$ that is not shared. During the training stage, different from the unshared version, the gradient information from each kernel would aggregate before updating the shared factors, making the learning process more efficient. We name this new residual unit as *Collective Residual Unit (CRU)*. We build a new and modularized CRU network by stacking multiple CRUs.

4.2 Proposed CRU Network Architecture

Our proposed CRU network is built by stacking multiple architecturally similar Collective Residual Units. Our designing principle follows ResNeXt [Xie *et al.*, 2016] and the ablation experiments are conducted by replacing the residual units with our new proposed CRUs. We roughly keep the model size unchanged compared with the vanilla residual network [He *et al.*, 2016a] throughout our design.

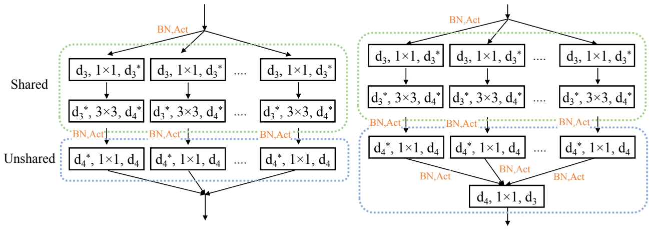


Figure 3: The proposed collective residual unit (CRU) architecture based on collective tensor factorization. **Left** is the standard form without considering the nonlinearity; **Right** is the improved form with better nonlinearity. “BN” refers to “Batch Normalization”; “Act” refers to “Activation”. The parameters of the first two layers (highlighted by the green bounding box) are shared *across different residual units*, while the other layer(s) are not shared. Best viewed in zoomed PDF.

CRU with Activation and Batch Normalization

The analysis above does not consider the batch normalization and activation layers. Here we consider adding them back to build a more conventional deep neural network.

When parameters are not shared across layers, batch normalization layers can be directly added. However, when parameters are shared across layers, adding them becomes complicated since the batch normalization [Ioffe and Szegedy, 2015] cannot be shared together with these parameters across layers by nature. Moreover, since the activation function is usually combined with the batch normalization, it would lose one “BN, Act” as shown in Figure 3 (left). To solve this problem, we append another 1×1 convolution layer after the second 1×1 convolutional layer as shown in Figure 3 (right).

Note that introducing this 1×1 convolution layer does not increase the model size compared with baseline models, benefiting from parameter sharing by CRUs. Also, because the 1×1 convolution operation has very low computational cost, it does not affect the computational efficiency much.

Overall Architecture

Table 1 shows the architecture details. In CRU, parameters are shared within every six layers and for those with less than six layers, we do not share parameters across units. The notation “$R\times d_3^*$d” is introduced to represent the network settings, where R is the number of low-rank Tuckers, which corresponds to the number of groups. $d_3^* = d_4^*$ controls the rank of each Tucker. See Eqn. (3). For CRU-Net, we use “@stage” to indicate the stage where we adopt CRU. For example, in Table 1, “32 × 4d @×28×14” denotes $R = 32$, $d_3^* = d_4^* = 4$, and CRU adopted at *conv3* and *conv4*.

5 Experiments

We evaluate our proposed model on two large-scale datasets, the ImageNet-1k dataset [Russakovsky *et al.*, 2015], and the Places365-Standard dataset [Zhou *et al.*, 2016]. For the ImageNet-1k, we report single crop validation error rate following [He *et al.*, 2016a]. For the Places365-Standard, we report 10 crops validation accuracy following [Zhou *et al.*, 2016]. Throughout the experiments, we use “(ours)” to denote the baseline reproduced by ourselves.

The networks are implemented by MXNet [Chen *et al.*, 2015], and trained on a cluster with 68 GPUs using SGD [Kingma and Ba, 2014] with a mini-batch size of 32 for each GPU and updated in a synchronized manner. The weight decay and the base learning rate are set to 0.0005 and 0.1 for all CNNs with 50 layers, while 0.0002 and $\sqrt{0.1}$ for all deeper CNNs. During testing, we report the standard 224×224 center crop from the raw input image with short length equal to 256, following the same setting as [He *et al.*, 2016a; 2016b; Xie *et al.*, 2016].

5.1 Results on ImageNet-1k

The ImageNet-1k dataset contains about 1.28 million high-resolution images of 1,000 categories. For this dataset, we first conduct ablation experiments to study the properties of the introduced block term decomposition method. Then we compare our proposed model with state-of-the-art models by building deeper and wider CRU networks.

First, we conduct a set of experiments to study the effect of the number of Tuckers, *i.e.* R, on the performance. We fix the model size roughly the same, *i.e.* about 97 MB, and vary the value of R as shown in Table 2. Since the overall number of parameters is fixed, more Tuckers means a lower representation ability for each Tucker (see Eqn. (3)). As can be seen in the last row of Table 2, when we set the number of R to its maximum value for each residual function, the error rate increases from 22.1% to 22.5%. This might be caused by the insufficient learning capacity of each Tucker. The best setting, as can be seen in the first three rows, is to double the rank of each Tucker when the input increases. Such an observation indicates that the number of groups, *i.e.*, *Cardinality* [Xie *et al.*, 2016], is not proportional to the performance.

Secondly, we compare our proposed network with other state-of-the-art residual networks. Table 3 summarizes different variants of the residual network under the same model size. The ResNet-200 gives the best published performance [He *et al.*, 2016b]. As shown in the table, under the setting of “32x4d” our proposed model achieves 21.9% top-1 error rate, better than 22.2% of ResNeXt. When we change the setting from “32x4d” to “136x1d”, the performance of ResNeXt becomes saturate. In contrast, our proposed CRU achieves 21.7% top-1 error rate. The performance is comparable with ResNet-200 [He *et al.*, 2016b] but the model requires much fewer parameters (98MB v.s. 247MB).

Finally, we increase the size of our proposed model to build a more complex CRU network. We compare its performance with the state-of-the-art residual networks. As can be seen from Table 4, our proposed model achieves 20.6% top-1 error rate compared with the 21.2% of ResNeXt. Even on extremely large model where severe over-fitting dominates the training process, the propose CRU-Net can still slightly boost the top-1 accuracy.

5.2 Results on Places365-Standard

We further evaluate our proposed network on one of the largest scene classification datasets, Places365-Standard [Zhou *et al.*, 2016]. It consists of 1.8 million images of 365 scene categories. Different from the object classification task where the image category can be told from

stage	output	ResNet-50	ResNeXt-50 (32×4d)	ResNeXt-50 (N×1d)	**CRU-Net-56 (32×4d @×14)**	**CRU-Net-116 (32×4d @×28×14)**
conv1	112x112	7×7, 64, stride 2	7×7, 64, stride 2	7×7, 64, stride 2	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56x56	3×3 max pool, stride 2	3×3 max pool, stride 2	3×3 max pool, stride 2	3×3 max pool, stride 2	3×3 max pool, stride 2
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64, R=1 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, R=32 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 136 \\ 3\times3, 136, R=136 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, R=32 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, R=32 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, R=1 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, R=32 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 272 \\ 3\times3, 272, R=272 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, R=32 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} \mathbf{1\times1, 352} \\ \mathbf{3\times3, 352, R=352} \\ 1\times1, 352 \\ 1\times1, 512 \end{bmatrix} \times 6$
conv4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, R=1 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, R=32 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 544 \\ 3\times3, 544, R=544 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} \mathbf{1\times1, 640} \\ \mathbf{3\times3, 640, R=640} \\ 1\times1, 640 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\left[\begin{bmatrix} \mathbf{1\times1, 704} \\ \mathbf{3\times3, 704, R=704} \\ 1\times1, 704 \\ 1\times1, 1024 \end{bmatrix} \times 6\right] \times 3$
conv5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, R=1 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, R=32 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 1088 \\ 3\times3, 1088, R=1088 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, R=32 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, R=32 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params		$\mathbf{25.5} \times 10^6$	$\mathbf{25.0} \times 10^6$	$\mathbf{24.9} \times 10^6$	$\mathbf{25.5} \times 10^6$	$\mathbf{43.7} \times 10^6$
FLOPs		$\mathbf{4.1} \times 10^9$	$\mathbf{4.2} \times 10^9$	$\mathbf{4.3} \times 10^9$	$\mathbf{4.9} \times 10^9$	$\mathbf{13.3} \times 10^9$

Table 1: Comparison of our proposed CRU Network (CRU-Net) and different residual networks. We compare our proposed CRU-Net with three baseline methods: vanilla ResNet [He *et al.*, 2016a], ResNeXt [Xie *et al.*, 2016], and ResNeXt with the highest R for each 3×3 convolutional layer in the residual unit. We also show the detailed setting of a deeper CRU-Net (CRU-Net-116) with a slightly less number of parameters than the vanilla ResNet-101 [He *et al.*, 2016a] (#params: 44.31×10^6).

Method	setting	model size	top-1 err.(%)
ResNeXt-50 (ours)	2 x 40d	98 MB	22.8
ResNeXt-50 (ours)	32 x 4d	96 MB	22.2
ResNeXt-50 (ours)	136 x 1d	97 MB	**22.1**
ResNeXt-50 (ours)	N x 1d	96 MB	**22.5**

Table 2: Single crop validation error rate of residual networks with different R on ImageNet-1k dataset.

Method	setting	model size	top-1 err.(%)
ResNet-50 [He *et al.*, 2016a]	1 x 64d	98 MB	23.9
ResNet-200 [He *et al.*, 2016b]	1 x 64d	247 MB	**21.7**
ResNeXt-50 [Xie *et al.*, 2016]	2 x 40d	98 MB	23.0
ResNeXt-50 [Xie *et al.*, 2016]	32 x 4d	96 MB	22.2
ResNeXt-50 (ours)	2 x 40d	98 MB	22.8
ResNeXt-50 (ours)	32 x 4d	96 MB	**22.2**
ResNeXt-50 (ours)	136 x 1d	97 MB	**22.1**
CRU-Net-56 @×14	32 x 4d	98 MB	**21.9**
CRU-Net-56 @×14	136 x 1d	98 MB	**21.7**

Table 3: Single crop validation error rate on ImageNet-1k dataset.

the most distinguishable parts, scene recognition requires a stronger reasoning ability and a larger receptive filed over the image for classification.

Table 5 shows the results of different models on Places365-Standard dataset. The results in the first four rows are provided by [Zhou *et al.*, 2016] and the last two rows are from our implementation. Our proposed method achieves the best classification accuracy compared with other methods. Compared with the vanilla ResNet-152, our proposed method improves the top-1 performance by absolute value of 1.2% with a significantly smaller model size (163MB v.s. 226MB), which again confirms the effectiveness of our proposed CRU-Net.

Method	setting	model size	top-1 err.(%)	top-5 err.(%)
ResNet-101 [He *et al.*, 2016b]	1 x 64d	170 MB	22.4	6.2
WRN [Zagoruyko and Komodakis, 2016]	1 x 128d	263 MB	21.9	5.8
ResNet-200 [He *et al.*, 2016b]	1 x 64d	247 MB	21.7	5.8
ResNeXt-101 [Xie *et al.*, 2016]	32 x 4d	170 MB	**21.2**	**5.6**
CRU-Net-116 @×28×14	32 x 4d	168 MB	**20.6**	**5.4**
ResNeXt-101, wider [Xie *et al.*, 2016]	64 x 4d	320 MB	20.4	5.3
ResNeXt-101, wider (ours)	64 x 4d	320 MB	20.4	5.3
CRU-Net-116, wider @×28×14	64 x 4d	318 MB	**20.3**	**5.3**

Table 4: Comparison with state-of-the-art residual networks. Single crop validation error rate on ImageNet-1k dataset.

Method	setting	model size	top-1 acc.(%)	top-5 acc.(%)
AlexNet	–	223MB	53.2	82.9
VGG-16	–	518MB	55.2	84.9
ResNet-152	1 × 64d	226MB	54.7	85.1
ResNeXt-101 (ours)	32 × 4d	165MB	56.2	86.3
CRU-Net-116 @×28×14	32 × 4d	163MB	**56.6**	**86.6**

Table 5: Single model, 10 crops validation accuracy on Places365-Standard. Results in first four rows are from [Zhou *et al.*, 2016].

6 Conclusion

In this work, we introduced the tensor Block Term Decomposition to analyze the popular bottleneck architecture from a new perspective. We revealed the relation of different residual functions with the tensor BTD framework and further proposed a novel network architecture called CRU based on the collective tensor factorization. The CRU enables efficient knowledge sharing across different residual units and thus further enhances parameter efficiency of the residual units. Employing CRUs provided the state-of-the-art performance on two large scale benchmark datasets, which confirms that CRU is effective at sharing knowledge throughout the convolutional neural network and increasing parameter efficiency as well as model performance.

References

[Chen *et al.*, 2015] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[Cohen *et al.*, 2016] Nadav Cohen, Or Sharir, and Amnon Shashua. Deep simnets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4782–4791, 2016.

[De Lathauwer, 2008] Lieven De Lathauwer. Decompositions of a higher-order tensor in block terms–part ii: Definitions and uniqueness. *SIAM Journal on Matrix Analysis and Applications*, 30(3):1033–34, 2008.

[Eigen *et al.*, 2013] David Eigen, Jason Rolfe, Rob Fergus, and Yann LeCun. Understanding deep architectures using a recursive convolutional network. *arXiv preprint arXiv:1312.1847*, 2013.

[Garipov *et al.*, 2016] Timur Garipov, Dmitry Podoprikhin, Alexander Novikov, and Dmitry Vetrov. Ultimate tensorization: compressing convolutional and fc layers alike. *arXiv preprint arXiv:1611.03214*, 2016.

[He *et al.*, 2016a] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[He *et al.*, 2016b] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

[Ioffe and Szegedy, 2015] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[Jaderberg *et al.*, 2014] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[Kingma and Ba, 2014] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Kolda and Bader, 2009] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[Lebedev *et al.*, 2014] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

[Li *et al.*, 2016] Yi Li, Kaiming He, Jian Sun, et al. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in Neural Information Processing Systems*, pages 379–387, 2016.

[Liao and Poggio, 2016] Qianli Liao and Tomaso Poggio. Bridging the gaps between residual learning, recurrent neural networks and visual cortex. *arXiv preprint arXiv:1604.03640*, 2016.

[Oseledets, 2011] Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[Ren *et al.*, 2015] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[Rigamonti *et al.*, 2013] Roberto Rigamonti, Amos Sironi, Vincent Lepetit, and Pascal Fua. Learning separable filters. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 2754–2761. IEEE, 2013.

[Russakovsky *et al.*, 2015] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[Singh and Gordon, 2008] Ajit P Singh and Geoffrey J Gordon. Relational learning via collective matrix factorization. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 650–658. ACM, 2008.

[Tucker, 1966] Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[Veit *et al.*, 2016] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in Neural Information Processing Systems*, pages 550–558, 2016.

[Wang and Cheng, 2016] Peisong Wang and Jian Cheng. Accelerating convolutional neural networks for mobile applications. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 541–545. ACM, 2016.

[Wang *et al.*, 2017] Qi Wang, Junyu Gao, and Yuan Yuan. A joint convolutional neural networks and context transfer for street scenes labeling. *IEEE Transactions on Intelligent Transportation Systems*, 2017.

[Wang *et al.*, 2018] Qi Wang, Jia Wan, and Yuan Yuan. Locality constraint distance metric learning for traffic congestion detection. *Pattern Recognition*, 75:272–281, 2018.

[Wu *et al.*, 2016] Zifeng Wu, Chunhua Shen, and Anton van den Hengel. Wider or deeper: Revisiting the resnet model for visual recognition. *arXiv preprint arXiv:1611.10080*, 2016.

[Xie *et al.*, 2016] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *arXiv preprint arXiv:1611.05431*, 2016.

[Zagoruyko and Komodakis, 2016] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[Zhao *et al.*, 2016] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. *arXiv preprint arXiv:1612.01105*, 2016.

[Zhou *et al.*, 2016] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Antonio Torralba, and Aude Oliva. Places: An image database for deep scene understanding. *arXiv preprint arXiv:1610.02055*, 2016.