

---

# EINCONV: EXPLORING UNEXPLORED TENSOR DECOMPOSITIONS FOR CONVOLUTIONAL NEURAL NETWORKS

---

A PREPRINT

**Kohei Hayashi**  
Preferred Networks  
hayasick@preferred.jp

**Taiki Yamaguchi\***  
The University of Tokyo  
yamaguchi@hep-th.phys.s.u-tokyo.ac.jp

**Yohei Sugawara**  
Preferred Networks  
suga@preferred.jp

**Shin-ichi Maeda**  
Preferred Networks  
ichi@preferred.jp

## ABSTRACT

Tensor decomposition methods are one of the primary approaches for model compression and fast inference of convolutional neural networks (CNNs). However, despite their potential diversity, only a few typical decompositions such as CP decomposition have been applied in practice; more importantly, no extensive comparisons have been performed between available methods. This raises the simple question of how many decompositions are available, and which of these is the best. In this paper, we first characterize a decomposition class specific to CNNs by adopting graphical notation, which is considerably flexible. When combining with the nonlinear activations, the class includes renowned CNN modules such as depthwise separable convolution and bottleneck layer. In the experiments, we compare the tradeoff between prediction accuracy and time/space complexities by enumerating all the possible decompositions. Also, we demonstrate, using a neural architecture search, that we can find nonlinear decompositions that outperform existing decompositions.

## 1 Introduction

Convolutional neural networks (CNNs) process multiple discrete convolution operations through so-called convolutional layers, which are typically useful to temporal/spatial data such as images [Goodfellow et al., 2016]. Despite their high performance, their high usage of memory and CPU/GPU is a bottleneck when deploying them on edge devices such as mobile phones [Howard et al., 2017].

To reduce costs, one straightforward approaches is to introduce a low-dimensional linear structure into the convolutional layers [Smith et al., 1997, Rigamonti et al., 2013, Tai et al., 2015, Kim et al., 2015, Denton et al., 2014, Lebedev et al., 2014, Wang et al., 2018]; this typically takes the form of tensor decomposition. Tensor decomposition represents the convolution filter using a fewer number of parameters in a sum-product form, which saves both memory space and the calculation cost for forwarding paths.

The manner in which the cost is reduced depends heavily on the structure of the tensor decomposition. For example, if a target tensor is of 2 ways, i.e., a matrix, meaningful decomposition is uniquely determined as  $\mathbf{X} = \mathbf{UV}$ , because other decompositions such as  $\mathbf{X} = \mathbf{ABC}$  are reduced to that form with a fewer number of parameters. However, for higher-order tensors, there are numerous variations for decomposition, of which only a few have been actively studied in the tensor decomposition research community (e.g., see [Kolda and Bader, 2009]). The existing studies applied such multi-purpose decompositions to CNNs. However, these decompositions are not necessarily optimal for CNNs owing to the tradeoff between the prediction accuracy and time/space complexity. Because the best tradeoffs associate with multiple factors such as application domains, tasks, entire architectures of CNNs, and hardware limitations, new options are inevitable for optimization.

---

\*This work was completed during an internship at Preferred Networks.

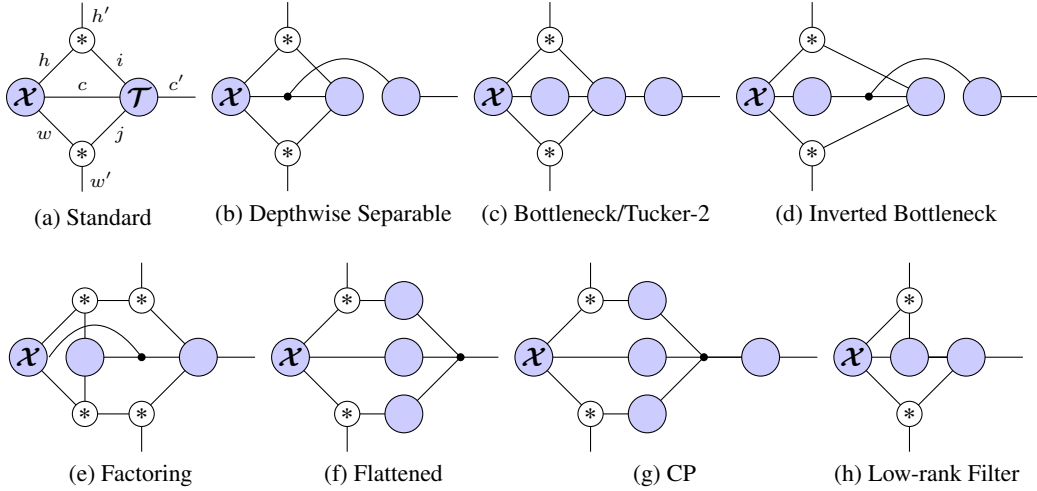


Figure 1: Visualizing linear structures in various convolutional layers, where  $\mathcal{X}$  is input and  $\mathcal{T}$  is a convolution kernel. The connected edges only on one side going up, down, and right respectively represent the spatial height, spatial width, and output channels. We will further explain this in Section 3.

In this paper, we study a hidden realm of tensor decompositions to identify the most resource-efficient convolutional layers. We first characterize a decomposition class specific to CNNs by adopting a hypergraphical notation based on tensor networks [Penrose, 1971], which is considerably flexible and, when combined with the nonlinear activations, the class includes modern light-weight CNN layers such as bottleneck layers used in ResNet [He et al., 2015], depthwise separable convolution used in Mobilenet V1 [Howard et al., 2017], inverted bottleneck layers used in Mobilenet V2 [Sandler et al., 2018], and more, as shown in Figure 1. The notation straightforwardly handles 3D or more higher-order convolutions. In the experiments, we compare the tradeoffs by enumerating all possible decompositions for 2D and 3D image data sets. Also, we evaluate the nonlinear extension by combining neural architecture search with the LeNet and ResNet architectures. The code implemented in Chainer [Tokui et al., 2019] is available at <https://github.com/pfnet-research/einconv>.

**Notation** For a positive integer  $n$ , we denote  $[n] = \{1, \dots, n\}$ . We denote scalars, vectors, matrices, and tensors respectively in lower, bold lower, bold upper, and bold script upper case, as  $a$ ,  $\mathbf{a}$ ,  $\mathbf{A}$ , and  $\mathcal{A}$ .

## 2 Preliminaries

**Convolution in Neural Networks** Assuming we have a 2D image of height  $H \in \mathbb{N}$ , width  $W \in \mathbb{N}$ , and number of channels  $C \in \mathbb{N}$ , where the channel is a feature that each image has for each pixel, such as RGB, the image can be considered a 3-way tensor  $\mathcal{X} \in \mathbb{R}^{H \times W \times C}$ . Usually, the convolution operation changes the size and the number of channels. We assume that the size of the convolution filter is odd, and that  $I, J \in \{1, 3, 5, \dots\}$  is its height and width,  $P \in \mathbb{N}$  is the padding size, and  $S \in \mathbb{N}$  is the stride. Then, the spatial size of the output is determined by height  $H' = (H + 2P - I)/S + 1$  and width  $W' = (W + 2P - J)/S + 1$ . When we set the number of output channels as  $C' \in \mathbb{N}$ , the convolution layer yields an output  $\mathcal{Z} \in \mathbb{R}^{H' \times W' \times C'}$  in which each element is given as

$$z_{h'w'c'} = \sum_{i \in [I]} \sum_{j \in [J]} \sum_{c \in [C]} t_{ijc} x_{h'_i w'_j c}, \quad (1)$$

where  $\mathcal{T} \in \mathbb{R}^{I \times J \times C \times C'}$  is a weight called an  $I \times J$  kernel, and  $h'_i = (h' - 1)S + i - P$  and  $w'_j = (w' - 1)S + j - P$  are spatial indices used for convolution. For simplicity, we omit the bias parameter. There are  $IJC'C'$  parameters and the time complexity of (1) is  $O(IJCH'W'C')$ .

Although (1) is the standard, there are several variations to reduce the computational complexity. When the spatial dimensions are both 1 ( $I = J = 1$ ), it is called  $1 \times 1$  convolution [Lin et al., 2013, Szegedy et al., 2015], which applies linear transformation to only the channels and does not affect the spatial directions. Depthwise convolution [Chollet, 2016] is possibly the opposite of  $1 \times 1$  convolution, which works as though the input and output channels are one

dimension, i.e.,

$$z_{h'w'c'} = \sum_{i \in [I]} \sum_{j \in [J]} t_{ijc'} x_{h'_i} w'_{j'} c'. \quad (2)$$

**Tensor Decomposition in Convolution** To reduce the computational complexity, Kim et al. [2015] applied Tucker-2 decomposition [Tucker, 1966] to the kernel  $\mathcal{T}$ , which replaces the original kernel by  $\mathcal{T}^{\text{T}2}$  where each element is given as

$$t_{ijc'}^{\text{T}2} = \sum_{\alpha \in [A]} \sum_{\beta \in [B]} g_{ij\alpha\beta} u_{c\alpha} v_{c'\beta} \quad (3)$$

where  $\mathcal{G} \in \mathbb{R}^{I \times J \times A \times B}$ ,  $\mathbf{U} \in \mathbb{R}^{C \times A}$ ,  $\mathbf{V} \in \mathbb{R}^{C' \times B}$  are new parameters and  $A, B \in \mathbb{N}$  are rank-like hyperparameters. Note that the convolution with the Tucker-2 kernel  $\mathcal{T}^{\text{T}2}$  is equivalent to three consecutive convolutions:  $1 \times 1$  convolution with kernel  $\mathbf{U}$ ,  $I \times J$  convolution with kernel  $\mathcal{G}$ , and  $1 \times 1$  convolution with kernel  $\mathbf{V}$ . In this view,  $A$  and  $B$  can be seen as intermediate channels during the three convolutions. Hence, when  $A, B$  are smaller than  $C, C'$ , a cost reduction is expected, because the heavy  $I \times J$  convolution is now taken with the  $A, B$  channel pair instead of  $C, C'$ . When compared with the original convolution, the reduction ratio of the number of parameters and the inference cost are both at least  $AB/CC'$ .

Similarly, several authors [Denton et al., 2014, Lebedev et al., 2014] have employed CP decomposition [Hitchcock, 1927], which reparameterizes the kernel as

$$t_{ijc'}^{\text{CP}} = \sum_{\gamma \in [\Gamma]} \tilde{u}_{i\gamma} \tilde{v}_{j\gamma} \tilde{w}_{c\gamma} \tilde{s}_{c'\gamma}, \quad (4)$$

where  $\tilde{\mathbf{U}}, \tilde{\mathbf{V}}, \tilde{\mathbf{W}}, \tilde{\mathbf{S}}$  are new parameters and  $\Gamma \in \mathbb{N}$  is a hyperparameter.

### 3 The Einconv Layer

We have seen that both the convolution operation (1), (2) and the decompositions of the kernel (3), (4) are given as the sum-product of multiple tensors with many indices. Although the indices may appear cluttered, they play important roles. In particular, they can be divided into two classes: ones that are connected to the output shape  $(h', w', c')$  and ones that are used for summation  $(i, j, c, \alpha, \beta, \gamma)$ . Convolution and its decomposition are actually specified by how those indices interact, and how they are distributed into tensor variables. For example, Tucker-2 decomposition separates the spatial, input channel, and output channel information as  $\mathcal{G}, \mathbf{U}, \mathbf{V}$  through their indices  $(i, j), c', c$ , respectively. Moreover, they are joined by two-step connections: the connection between the input channel and spatial information via  $\alpha$ , and the connection between the output channel and spatial information via  $\beta$ . Here, we can consider the indices used for summation to be paths that deliver input information to the output.

This viewpoint brings us the notion that a hypergraph captures the index interaction in a clean manner. The basic idea is that we distinguish tensors only by the indices they own and we consider them as vertices. The vertices are connected if they share some indices to be summed. For example, consider the decomposition of a kernel  $\mathcal{T}$ . Let *outer indices*  $\mathcal{O} = \{i, j, c, c'\}$  be the indices of the shape of  $\mathcal{T}$ , *inner indices*  $\mathcal{I} = (r_1, r_2, \dots)$  be the indices used for summation, and *inner dimensions*  $\mathcal{R} = (R_1, R_2, \dots) \in \mathbb{R}^{|\mathcal{I}|}$  be the dimensions of  $\mathcal{I}$ . Assume that  $M \in \mathbb{N}$  tensors are involved in the decomposition where each tensor is denoted by a set of indices, and let  $\mathcal{V} = \{v_1, \dots, v_M \mid v_m \in 2^{\mathcal{O} \cup \mathcal{I}}\}$  denote the set of the tensors, where  $2^{\mathcal{A}}$  denotes the power set of a set  $\mathcal{A}$ . Here we identify each tensor by its indices, i.e.,  $\mathcal{U} = (u_{abc})_{a \in [A], b \in [B], c \in [C]}$  is equivalent to  $\{a, b, c\}$ . Given  $\mathcal{V}$ , each inner index  $r \in \mathcal{I}$  defines a hyperedge  $e_r = \{v \mid r \in v \text{ for } v \in \mathcal{V}\}$ . Let  $\mathcal{E} = \{e_n \mid n \in \mathcal{O} \cup \mathcal{I}\}$  denote the set of hyperedges. For example, suppose  $\mathcal{I} = \{\alpha, \beta\}$  and  $\mathcal{V} = \{\{i, j, \alpha, \beta\}, \{c, \alpha\}, \{c', \beta\}\}$ ; then, the undirected weighted hypergraph  $(\mathcal{V}, \mathcal{E}, \mathcal{R})$  is equivalent to Tucker-2 decomposition (3).

This idea is also applicable to the convolution operation by the introduction of dummy tensors that absorb the index patterns used in convolution. Recall that in (1) the special index  $h'_i$  represents which vertical elements of the kernel and the input image are coupled in the convolution. Let  $\mathcal{P} \in \{0, 1\}^{H \times H' \times I}$  be a binary tensor where each element is defined as  $p_{hh'i} = 1$  if  $h = h'_i$  and 0 otherwise. Similarly, let  $\mathcal{Q} \in \{0, 1\}^{W \times W' \times J}$  be the horizontal counterpart of  $\mathcal{P}$ . Also, let us modify the index sets as  $\mathcal{O} = \{h', w', c'\}$  and  $\mathcal{I} = (h, w, i, j, c)$ , and the dimensions  $\mathcal{R} = (H, W, I, J, C)$ . Then, vertices  $\mathcal{V} = \{\{h, w, c\}, \{i, j, c, c'\}, \{h, h', i\}, \{w, w', j\}\}$  and hyperedges  $\mathcal{E}$  that are automatically defined by  $\mathcal{V}$  exactly represents the convolution operation (1), where we ensure that the tensor of  $\{h, h', i\}$  is fixed by  $\mathcal{P}$  and the tensor of  $\{w, w', j\}$  is fixed by  $\mathcal{Q}$ .

The above mathematical explanation may sound too winding, but visualization will help greatly. Let us introduce several building blocks for the visualization. A circle indicates a tensor and a line connected to a circle indicates an index associated with the tensor. When an edge is connected on only one side, it appears in the resulting tensor as an outer index; it is otherwise used for summation as an inner index, namely,

$$\begin{array}{c} \text{---} \\ | \\ \textcircled{A} \\ | \\ \text{---} \end{array} \begin{array}{c} \text{---} \\ | \\ \textcircled{B} \\ | \\ \text{---} \end{array} = \begin{array}{c} \text{---} \\ | \\ \textcircled{C} \\ | \\ \text{---} \end{array} \iff \sum_j a_{ij} b_{jk} = c_{ik}. \quad (5)$$

The summation and the elimination of inner indices is called *contraction*. A hyperedge that is connected by more than three vertices is depicted with a black dot:

$$\begin{array}{c} \text{---} \\ | \\ \textcircled{A} \\ | \\ \text{---} \end{array} \begin{array}{c} \text{---} \\ | \\ \textcircled{B} \\ | \\ \text{---} \end{array} \begin{array}{c} \text{---} \\ | \\ \textcircled{C} \\ | \\ \text{---} \end{array} \iff \sum_j a_{ij} b_{jk} c_{jk}. \quad (6)$$

Finally, a node with symbol “\*” indicates the dummy tensors  $\mathcal{P}$  or  $\mathcal{Q}$  that implicitly indicate that vertical or horizontal convolution is involved:

$$\begin{array}{c} \text{---} \\ | \\ \textcircled{A} \\ | \\ \text{---} \end{array} \begin{array}{c} \text{---} \\ | \\ * \\ | \\ \text{---} \end{array} \begin{array}{c} \text{---} \\ | \\ \textcircled{B} \\ | \\ \text{---} \end{array} \iff \sum_{h,i} p_{hh'} a_h b_i \quad (7)$$

Note that the above diagram descriptions are essentially equivalent to what the Einstein notation represents. Analogous to this and NumPy’s `einsum` function [Wiebe, 2011], we term a hypergraphically-representable convolution layer an *Einconv* layer.

### 3.1 Examples

In Figure 1, we depict several examples of the hypergraphical notation. When we put aside nonlinear activation, we find that many existing CNN modules are described as Einconv layers.

**Separable and Low-rank Filters** Although the size of a kernel is usually square, i.e.,  $I = J$ , we often take the convolution separately along with the vertical and horizontal directions. In this case, the convolution operation is equivalent to the application of two filters of sizes  $(I, 1)$  and  $(1, J)$ . This can be considered as the rank-1 approximation of the  $I \times J$  convolution. A separable filter [Smith et al., 1997] is a technique to speed up convolution when the filter is exactly of rank one. Rigamonti et al. [2013] extended this idea by approximating filters by low-rank matrices for single input channel and Tai et al. [2015] further extended it for multiple input channels (Figure 1h).

**Factored Convolution** In case of a large filter size, a common technique called factoring convolution is used to replace the large filter with multiple small-sized convolutions [Szegedy et al., 2016]. For example, two consecutive  $3 \times 3$  convolutions are equivalent to one  $5 \times 5$  convolution in which the first  $3 \times 3$  filter is enlarged by the second  $3 \times 3$  filter. Interestingly, the factorization of convolution is exactly represented as Einconv by adding two additional dummy tensors.

**Bottleneck Layers** In ResNet [He et al., 2015], the bottleneck module is used as a building block, where input channels are reduced before convolution and then expanded after convolution. Finally, the original input is added, which is referred to as skip connection. Figure 1c shows the module without skip connection. According to the diagram, we see that the linear structure of the bottleneck is equivalent to Tucker-2 decomposition.

**Depthwise Separable Convolution** Mobilenet V1 [Howard et al., 2017] is a seminal light-weight architecture. It employs depthwise separable convolution [Sifre and Mallat, 2014, Chollet, 2016] as a building block, which is a combination of depthwise convolution and  $1 \times 1$  convolution (Figure 1b) to work with limited computational resources.

**Inverted Bottleneck Layers** Mobilenet V2 [Sandler et al., 2018], the second generation of Mobilenet, employs a building block called the inverted bottleneck module (Figure 1d). It is similar to the bottleneck module, but there are two differences. First, the number of intermediate channels is smaller than both the numbers of input and output channels in the bottleneck module. In contrast, this relationship is reversed in the inverted bottleneck, as the intermediate channels are “ballooned”. In addition, there are two intermediate channels in the bottleneck module, whereas the inverted bottleneck module consists of only one.

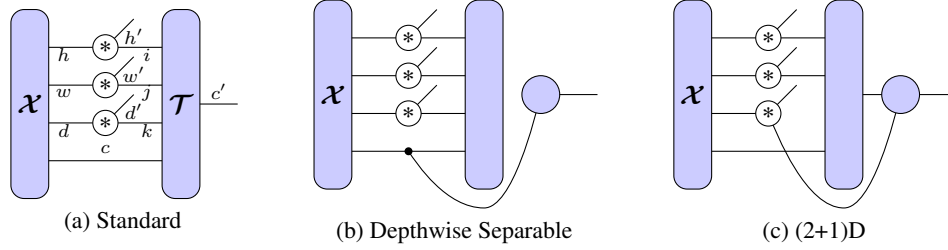


Figure 2: Graphical visualizations of 3D convolutions.

### 3.2 Higher Order Convolution

We have, thus far, considered 2D convolution, but have yet determined what happens we deal with 3D data. Einconv can handle 3D or higher-order convolution. For example, consider a 3D convolution and let  $d, d'$  be the input/output indices for depth and  $k$  be the index of filter depth. Then, by adding  $d'$  to  $\mathcal{O}$  and  $d, k$  to  $\mathcal{I}$ , we can construct a hypergraph for 3D convolution (Figure 2), from the standard to a light-weight convolution such as depthwise separable convolution [Köpüklü et al., 2019] and (2+1)D convolution [Tran et al., 2018], which factorizes a full 3D convolution into 2D and 1D convolutions.

### 3.3 Reduction and Enumeration

Although the hypergraphical notation is powerful, we need to be careful about its redundancy. For example, consider a hypergraph  $(\mathcal{V}, \mathcal{E})$  where an inner index  $a \in \mathcal{I}$  is only used by the  $m$ -th tensor, i.e.  $a \in v_m$  and  $a \notin v_n$  for  $n \neq m$ . Then any tensors represented by  $(\mathcal{V}, \mathcal{E})$  with any inner dimensions are also represented by removing  $a$  from every element of  $\mathcal{V}$  and the  $a$ -th hyperedge from  $\mathcal{E}$ . Similarly, any self loops do not increase the representability [Ye and Lim, 2018]. In terms of representability of the Einconv layer, there is no reason to choose redundant hypergraphs.<sup>2</sup> We therefore consider to remove them efficiently. Note that, although we only consider the 2D convolution case here for simplicity, the results are straightforwardly extensible to higher-order cases.

Let  $\setminus$  denote the set difference operator and  $\ominus$  denote the element-wise set difference operator, which is used to remove an index from all the vertices, e.g.,  $\mathcal{V} \ominus a = \{v_1 \setminus a, \dots, v_M \setminus a\}$  for index  $a \in \mathcal{O} \cup \mathcal{I}$ . For convenience, we define a map  $\theta : \mathcal{O} \cup \mathcal{I} \rightarrow \mathbb{N}$  that returns the dimension of index  $a \in \mathcal{O} \cup \mathcal{I}$ , e.g.  $\theta(i) = I$ . To discuss the representability, we introduce the notation for the space of the Einconv layers.

**Definition 1.** Given vertices  $\mathcal{V} = \{v_1, \dots, v_M\}$  and inner dimensions  $\mathcal{R}$ , let  $\mathcal{F}_{\mathcal{V}} : \mathbf{U}_1, \dots, \mathbf{U}_M \mapsto \mathcal{Z} \in \mathbb{R}^{I \times J \times C \times C'}$  be the function that calculates the contraction of  $M$  tensors  $\mathbf{U}_1, \dots, \mathbf{U}_M$  along with  $\mathcal{V}$ . In addition, let  $\mathbb{T}_{\mathcal{V}}(\mathcal{R}) \subseteq \mathbb{R}^{I \times J \times C \times C'}$  be the space that  $\mathcal{F}_{\mathcal{V}}$  covers, i.e.,  $\mathbb{T}_{\mathcal{V}}(\mathcal{R}) = \{\mathcal{F}_{\mathcal{V}}(\mathbf{U}_1, \dots, \mathbf{U}_M) \mid \mathbf{U}_m \in \mathbb{R}^{\times_{a \in v_m} \theta(a)} \text{ for } m \in [M]\}$ .

Next, we show several sufficient conditions of redundant hypergraphs.

**Proposition 1** (Ye and Lim 2018, Proposition 3.5). Given inner dimensions  $\mathcal{R} \in \mathbb{R}^{|\mathcal{I}|}$ , if  $R_a = 1$ ,  $\mathbb{T}_{\mathcal{V}}(\mathcal{R})$  is equivalent to  $\mathbb{T}_{\mathcal{V} \ominus a}(\dots, R_{a-1}, R_{a+1}, \dots)$ .

**Proposition 2.** If  $v_m \subseteq v_n$  for  $m, n \in [M]$ ,  $\mathbb{T}_{\mathcal{V}}(\mathcal{R})$  is equivalent to  $\mathbb{T}_{\mathcal{V} \setminus v_m}(\mathcal{R})$ .

**Proposition 3.** If  $e_a = e_b$  for  $a, b \in \mathcal{I}$ ,  $\mathbb{T}_{\mathcal{V}}(\mathcal{R})$  is equivalent to  $\mathbb{T}_{\mathcal{V} \ominus a}(\tilde{\mathcal{R}})$  where  $\tilde{\mathcal{R}} = (\dots, R_{a-1}, R_{a+1}, \dots, R_{b-1}, R_a R_b, R_{b+1}, \dots)$ .

**Proposition 4.** Assume the convolution is size-invariant, i.e.,  $H = H'$  and  $W = W'$ . Then, given filter height and width  $I, J \in \{1, 3, 5, \dots\}$ , the number of possible combinations that eventually achieve  $I \times J$  convolution is  $\pi(\frac{I-1}{2})\pi(\frac{J-1}{2})$ , where  $\pi : \mathbb{N} \rightarrow \mathbb{N}$  is the partition function of integers, see [Sloane, 2019] for examples.

Proposition 1 says that, if the inner dimension of an inner index is one, we can eliminate it from the hypergraph. Proposition 2 shows that, if the indices of a vertex is the subset of the indices of another vertex (e.g.  $v_1 = \{a, c\}$  and  $v_2 = \{a, b, c\}$ ), we can remove the former vertex. Proposition 3 means that the ‘‘double’’ hyperedge is reduced to a single hyperedge by increasing its dimension as it meets the product of theirs. Proposition 4 tells us the possible choices of filter size. We defer the proofs to Supplementary material. By combining the above propositions, we can conclude the following theorem.

<sup>2</sup>It might be possible that some redundant Einconv layer outperforms equivalent nonredundant ones, because the parametrization influences optimization. However, we focus on the representability here.

**Theorem 1.** *If the number of inner indices and the filter size is finite, the set of nonredundant hypergraphs representing convolution (1) is finite.*

To enumerate the nonredundant hypergraphs, we first use the condition of Proposition 2. Because of the vertex-subset constraint in Proposition 2, valid vertex sets must be the subset of the power set of all the indices  $\mathcal{O} \cup \mathcal{I}$ , and its size is at most  $2^{2^{|\mathcal{O} \cup \mathcal{I}|}}$ . After enumerating the vertex set satisfying the vertex-subset constraint, we eliminate some of them using the other propositions.<sup>3</sup> We used this algorithm in the experiment (Section 6).

## 4 Nonlinear Extension

Tensor decomposition involves multiple linear operations, and each vertex can be seen as a linear layer. For example, consider a linear map  $\mathbf{W} : \mathbb{R}^C \rightarrow \mathbb{R}^{C'}$ . If  $\mathbf{W}$  is written by a product of three matrices as  $\mathbf{W} = \mathbf{ABC}$ , we can consider the linear map to be a composition of three linear layers:  $\mathbf{W}(\mathbf{x}) = (\mathbf{A} \circ \mathbf{B} \circ \mathbf{C})(\mathbf{x})$  for a vector input  $\mathbf{x} \in \mathbb{R}^C$ . This leads to the assumption that, in addition to reducing the computational complexity, tensor decomposition with many vertices also contributes to an increase in representability. However, because the rank of  $\mathbf{W}$  is determined by the minimum rank of either  $\mathbf{A}$ ,  $\mathbf{B}$ ,  $\mathbf{C}$ , and the representability of a matrix is solely controlled by the rank, adding linear layers does not improve the representability as a function. This would be happen in Einconv layers.

To avoid this problem, a simple solution is to add nonlinear functions between linear layers. Although it is easy for implementation, enumeration is no longer possible, because the equivalence relation becomes non-trivial — with nonlinearity, there exists an infinite number of candidates. Of course we cannot enumerate an infinite number of candidates, and we need an efficient search algorithm. This type of study, which searches the best structure of neural networks, is called neural architecture search [Zoph and Le, 2016]. Many search algorithms have been proposed based on genetic algorithms (GAs) [Real et al., 2018], reinforcement learning [Zoph and Le, 2016] and others [Zoph et al., 2018, Pham et al., 2018]. In this study, we employ GA because hypergraphs are discretely structured and GA is highly compatible with them. As we need to solve multiobjective optimization (e.g. number of parameters v.s. prediction accuracy), we use the nondominated sorting genetic algorithm II (NSGA2) [Deb et al., 2002], which is one of the most popular multiobjective GAs. In Section 6.2 we will demonstrate that, using GA, we can find better Einconv layers than enumeration.

## 5 Related Work

The graphical notation of linear tensor operations, termed tensor networks, has been developed by the quantum many-body physics community (see tutorial by Bridgeman and Chubb [2017]). Our notation is basically a subset of the tensor network notation, with the exception that ours allows hyperedges. The hyperedges are convenient for representing several convolutions, such as depthwise convolution (see Figure 1b; the rightmost vertex indicates depthwise convolution). The reduction of redundant tensor networks was recently studied by Ye and Lim [2018], and we extended the idea to adopt convolution (Section 3.3).

There are several studies that combine deep neural networks and tensor networks. Stoudenmire and Schwab [2016] studied shallow fully-connected neural networks, where the weight is decomposed by the so-called tensor train decomposition [Oseledets, 2011]. Novikov et al. [2015] used a similar idea for deep feed-forward networks, which was also extended to recurrent neural networks [He et al., 2017, Yang et al., 2017]. For CNNs, Cohen and Shashua [2016] addressed a CNN architecture that can be viewed as a huge tensor decomposition. In contrast to our case that reformulates a single convolutional layer, they interpreted the entire forward process including the pooling operation as a tensor decomposition. Another difference is that they focused on a specific decomposition called hierarchical Tucker decomposition [Hackbusch and Kühn, 2009]; we do not impose any restrictions on decomposition forms.

## 6 Experiments

We examine the performance tradeoffs of Einconv layers in image classification tasks. We measured FLOPs of the entire forwarding path as time complexity and the total number of parameters as space complexity. All the experiments were conducted on NVIDIA P100 and V100 GPUs. The details of training recipes are described in Supplementary material.

<sup>3</sup>For more details, see the real code: [https://github.com/pfnet-research/einconv/blob/master/enumerate\\_graph.py](https://github.com/pfnet-research/einconv/blob/master/enumerate_graph.py)

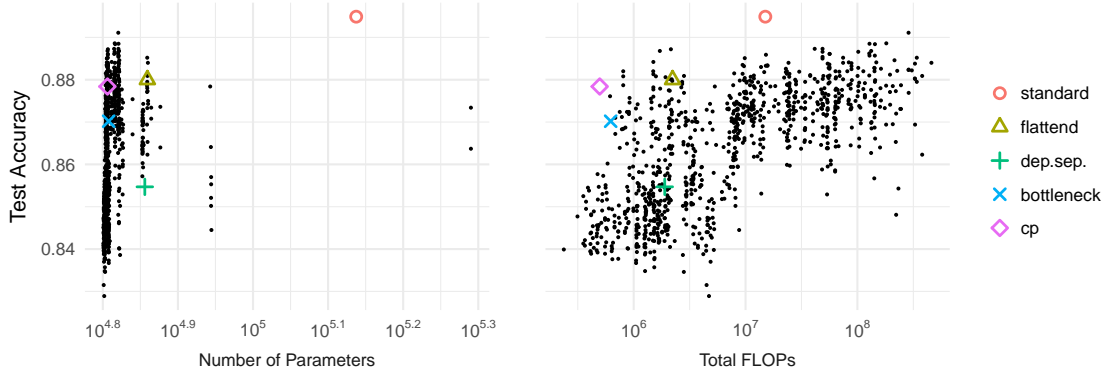


Figure 3: Enumeration of 2D Einconv for LeNet-5 trained with Fashion-MNIST. Black dots indicate unnamed tensor decompositions found by the enumeration.

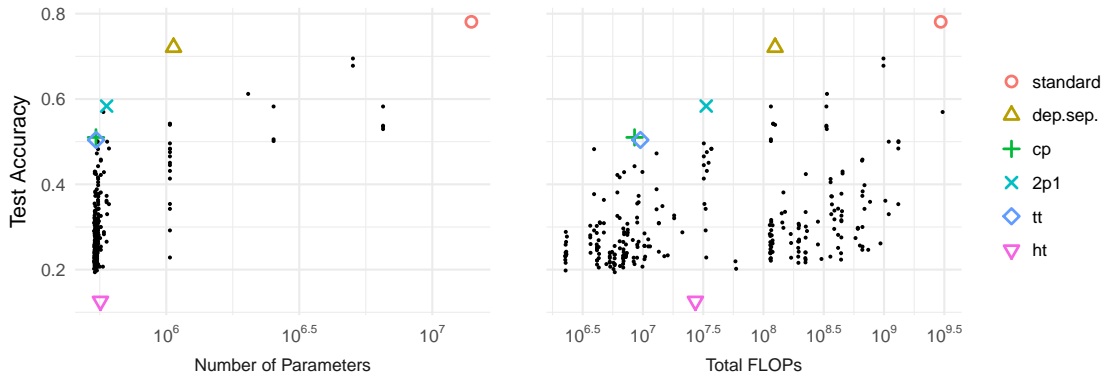


Figure 4: Enumeration of 3D Einconv for C3D-like networks trained with 3D MNIST, where 2p1, tt, and ht correspond to (2+1)D convolution [Tran et al., 2018], tensor train decomposition [Oseledets, 2011], and hierarchical Tucker decomposition [Hackbusch and Kühn, 2009], respectively.

## 6.1 Enumeration

First, we investigated the basic classes of Einconv for 2D and 3D convolutions. For 2D convolution with a filter size of  $3 \times 3$ , we enumerated nonredundant hypergraphs having at most two inner indices, which were 901 instances in total, where the inner dimensions were all fixed to 2. In addition to these, we compared baseline Einconv layers that include nonlinear activations and/or more inner indices. We used the Fashion-MNIST dataset [Xiao et al., 2017] to train the LeNet-5 network [LeCun et al., 1998]. The result (Figure 3) shows that, in terms of FLOPs, two baselines (standard and CP) achieve the Pareto optimality but other nameless Einconv layers fill the gap between those two.

Similarly, for  $3 \times 3 \times 3$  filter, we enumerated 3D Einconv having at most one inner index, which were 492 instances in total. We used the 3D MNIST dataset [de la Iglesia Castro, 2016] with the architecture inspired from C3D [Tran et al., 2014]. The results (Figure 4) show that, in contrast to 2D case, the baselines dominated Pareto frontier. This could be because we did not enumerate the case with two inner indices due to its enormous size.<sup>4</sup>

## 6.2 GA Search with Non-linear Activation

Next, we evaluate the full potential of Einconv by combining it with a neural architecture search. In contrast to the previous experiments, we searched Einconv layers from a larger space, i.e., allowing nonlinear activations (ReLU), factoring-like multiple convolutions, and changing the inner dimensions. We employed two architectures: LeNet-5

<sup>4</sup>For 3D convolution, the number of tensor decompositions having two inner indices is more than ten thousand. Training all of them requires 0.1 million CPU/GPU days, which was infeasible in our computational resources.

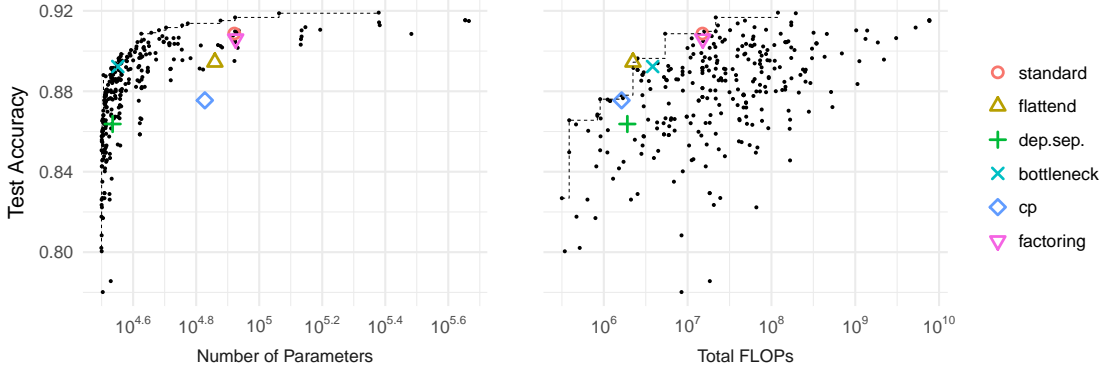


Figure 5: GA search of 2D Einconv for LeNet-5 trained with Fashion-MNIST. Black dots indicate unnamed tensor decompositions found by the GA search.

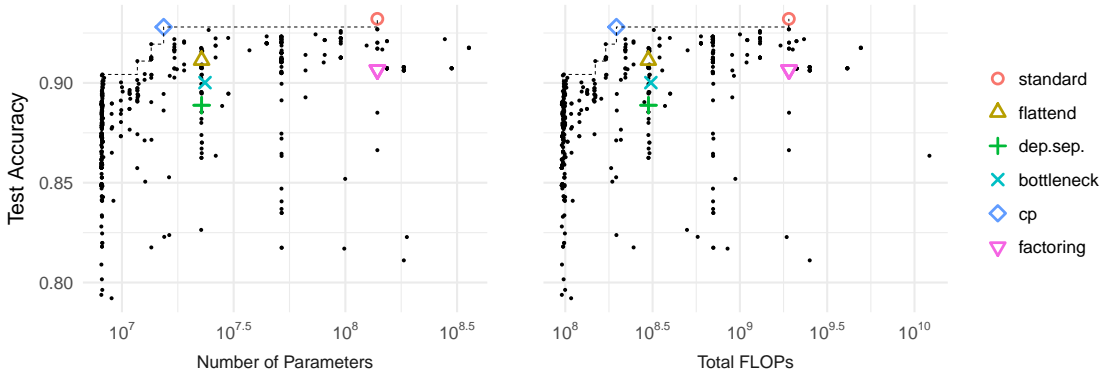


Figure 6: GA search of 2D Einconv for ResNet-50 trained with CIFAR-10.

and ResNet-50. We trained LeNet-5 with the Fashion-MNIST dataset and trained the ResNet-50 with the CIFAR-10 dataset. Note that, for ResNet-50, we could not train a significant number of Einconv instances because of the out of GPU memory. For a GA search, we followed the strategy of AmoebaNet [Real et al., 2018]. Namely, we did not use crossover operations; siblings were produced only by mutation. We prepare five mutation operations for changing the number of vertices/hyperedges and two mutation operations for changing the order of contraction.<sup>5</sup> We set test accuracy and the number of parameters as multiobjectives to be optimized by NSGA2.

The results of LeNet-5 (Figure 5) show the tradeoff between the multiobjectives, which we see an ideal curve of the Pareto frontier where nameless Einconv layers outperform the baselines. Also, we observe the best accuracy achieved by Einconv was  $\sim 0.92$ , which was better than the standard convolution ( $\sim 0.91$ ). Although the results of ResNet-50 (Figure 6) show a relatively rugged Pareto frontier, it achieves better tradeoffs except the standard and CP convolutions.

## 7 Conclusion and Discussion

In this paper, we studied hypergraphical structures in CNNs. We found that a variety of CNN layers are described hypergraphically, and there exist enormous number of variants that we have never encountered. In the experiments, we show that the Einconv layer, the proposed generalized CNN layer, helped to find better solutions.

One of the striking observations from the experiments is that some of the existing decompositions, such as CP decomposition consistently achieved good tradeoff. This empirical result is somehow unexpected because there is no theoretical reason that the existing decompositions outperform the unnamed ones. Developing a theory that can explain

<sup>5</sup>See <https://github.com/pfnet-research/einconv/blob/master/mutation.py> for implementation details.



this phenomenon or at least characterizing necessary conditions (e.g. symmetricity of decomposition) to achieve good tradeoff is a promising future work (it seems hard, though).

A current major limitation is the computational cost for searching. For example, the GA search for ResNet-50 in Section 6.2 took 829 CPU/GPU days. This is mainly because of the long training periods (approximately 10 CPU/GPU hours for each training), but it is also because GA may be not leveraging the information on hypergraphs well. Although we incorporated some prior knowledge for hypergraphs such as the proximity regarding edge removing and vertex adding through mutation operations, simultaneous optimization of hypergraph structures and neural networks using sparse methods such as LASSO or Bayesian sparse models may be more promising.

## Acknowledgments

The authors thank our colleagues, especially Tommi Kerola, Mitsuru Kusumoto, Kazuki Matoya, Shotaro Sano, Gentaro Watanabe, and Toshihiko Yanase for helpful discussion and Takuya Akiba for implementing the prototype of an enumeration algorithm. We also thank Jacob Bridgeman for sharing the nice TikZ style for drawing tensor network pictures.

## References

- J. C. Bridgeman and C. T. Chubb. Hand-waving and interpretive dance: an introductory course on tensor networks. *Journal of Physics A: Mathematical and Theoretical*, 50(22):223001, 2017.
- F. Chollet. Xception: Deep learning with depthwise separable convolutions, corr abs/1610.02357. URL <http://arxiv.org/abs/1610.02357>, 2016.
- N. Cohen and A. Shashua. Convolutional rectifier networks as generalized tensor decompositions. In *International Conference on Machine Learning*, pages 955–963, 2016.
- D. de la Iglesia Castro. 3d mnist dataset. <https://www.kaggle.com/daavoo/3d-mnist>, 2016.
- K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.
- E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pages 1269–1277, 2014.
- I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.
- W. Hackbusch and S. Kühn. A new scheme for the tensor representation. *Journal of Fourier analysis and applications*, 15(5):706–722, 2009.
- K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. eprint. *arXiv preprint arXiv:0706.1234*, 2015.
- Z. He, S. Gao, L. Xiao, D. Liu, H. He, and D. Barber. Wider and deeper, cheaper and faster: Tensorized lstms for sequence learning. In *Advances in neural information processing systems*, pages 1–11, 2017.
- F. L. Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189, 1927.
- A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.
- Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.
- T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.
- O. Köpüklü, N. Kose, A. Gunduz, and G. Rigoll. Resource efficient 3d convolutional neural networks. *arXiv preprint arXiv:1904.02422*, 2019.
- V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

- Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.
- A. Novikov, D. Podoprikin, A. Osokin, and D. P. Vetrov. Tensorizing neural networks. In *Advances in neural information processing systems*, pages 442–450, 2015.
- I. V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.
- R. Penrose. Applications of negative dimensional tensors. *Combinatorial mathematics and its applications*, 1:221–244, 1971.
- H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.
- E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018.
- R. Rigamonti, A. Sironi, V. Lepetit, and P. Fua. Learning separable filters. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2754–2761, 2013.
- M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.
- L. Sifre and S. Mallat. Rigid-motion scattering for image classification. *PhD thesis, Ph. D. thesis*, 1:3, 2014.
- N. J. A. Sloane. The On-Line Encyclopedia of Integer Sequences. A000041, 2019.
- S. W. Smith et al. *The scientist and engineer’s guide to digital signal processing*. California Technical Pub. San Diego, 1997.
- E. Stoudenmire and D. J. Schwab. Supervised learning with tensor networks. In *Advances in Neural Information Processing Systems*, pages 4799–4807, 2016.
- C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.
- C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.
- C. Tai, T. Xiao, Y. Zhang, X. Wang, et al. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.
- S. Tokui, R. Okuta, T. Akiba, Y. Niitani, T. Ogawa, S. Saito, S. Suzuki, K. Uenishi, B. Vogel, and H. Yamazaki Vincent. Chainer: A deep learning framework for accelerating the research cycle. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2002–2011. ACM, 2019.
- D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. “learning spatiotemporal features with 3d convolutional networks.”. *arXiv preprint arXiv:1412.0767*, 1177, 2014.
- D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri. A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 6450–6459, 2018.
- L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.
- W. Wang, Y. Sun, B. Eriksson, W. Wang, and V. Aggarwal. Wide compression: Tensor ring nets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9329–9338, 2018.
- M. Wiebe. Numpy-discussion: einsum. <https://mail.python.org/pipermail/numpy-discussion/2011-January/054586.html>, 2011.
- H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Y. Yang, D. Krompass, and V. Tresp. Tensor-train recurrent neural networks for video classification. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3891–3900. JMLR. org, 2017.

K. Ye and L.-H. Lim. Tensor network ranks. *arXiv preprint arXiv:1801.02662*, 2018.

B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

## Appendix

### A Proofs in Section 3.3

*Proof of Proposition 2.* Let  $\odot$  denote the element-wise multiplication with broadcasting<sup>6</sup> and  $m < n$ . Then the contraction of  $\mathbf{U}_1, \dots, \mathbf{U}_M$  along with  $\mathcal{V}$  is written as  $\mathcal{F}_{\mathcal{V}}(\mathbf{U}_1, \dots) = \mathcal{F}_{\mathcal{V} \setminus v_m}(\mathbf{U}_1, \dots, \mathbf{U}_{m-1}, \mathbf{U}_{m+1}, \dots, \mathbf{U}_{n-1}, \mathbf{U}_m \odot \mathbf{U}_n, \dots)$ . Since  $\mathbf{U}_m \odot \mathbf{U}_n \in \mathbb{R}^{\times_{a \in v_n} \theta(a)}$  for any  $\mathbf{U}_m$  and  $\mathbf{U}_n$ ,  $\mathcal{F}_{\mathcal{V}}(\dots)$  is reduced to  $\mathcal{F}_{\mathcal{V} \setminus v_m}(\dots)$ .  $\square$

*Proof of Proposition 3.* Let  $\text{Col}(\mathbf{U}_m, a, b)$  denote the collapsing operator that reshapes tensor  $\mathbf{U}_m$  by concatenating its indices  $\{a, b\}$  if  $\{a, b\} \subseteq v_m$ , and creates a new index  $b'$  where its inner dimension is  $R_{b'} = R_a R_b$ . Since  $\mathcal{F}_{\mathcal{V}}(\mathbf{U}_1, \mathbf{U}_2, \dots) = \mathcal{F}_{\mathcal{V} \otimes a}(\text{Col}(\mathbf{U}_1, a, b), \text{Col}(\mathbf{U}_2, a, b), \dots)$  and using the same technique used in the proof of Proposition 2, we can conclude the statement.  $\square$

*Proof of Proposition 4.* Suppose we have  $M$  size-invariant convolutions of size  $(I_1, J_1), \dots, (I_M, J_M)$ . By simple calculation, we see that the final convolution size  $(I, J)$  is determined by  $I = 1 + \sum_{m \in [M]} (I_m - 1)$  and  $J = 1 + \sum_{m \in [M]} (J_m - 1)$ . Therefore possible choices are to change  $\{I_m, J_m \mid m \in [M]\}$  with varying  $M \geq \min(\frac{I-1}{2}, \frac{J-1}{2})$ . This problem is reduced to the partition of integers, which is calculated by  $\pi$ .  $\square$

*Proof of Theorem 1.* For simplicity, consider 2D convolution with  $3 \times 3$  filter ( $I = J = 3$ ). Suppose we have  $L \in \mathbb{N}$  inner indices  $\mathcal{A} = \{c, r_1, \dots, r_{L-1}\}$ . According to Proposition 4, the vertical index  $i$  and the horizontal index  $j$  have to be used only once, and their usage is divided into two patterns: (i) they are used in the same vertex, or (ii) they are separated in different vertices. First we consider case (i). Assume  $v_1$  contains  $\{i, j\}$  and the subset of  $\mathcal{A}$ , which contains  $2^L$  patterns, and let  $\mathcal{A}_1 = v_1 \setminus \{i, j\} \subseteq 2^{\mathcal{A}}$  be the selected subset. Similarly, consider  $v_2$ . To avoid the redundancy described in Proposition 2,  $v_2$  must contain indices that are not contained in  $v_1$ , which means that the choices for  $v_2$  are in  $2^{\mathcal{A}} \setminus 2^{\mathcal{A}_1}$ . As we continue the process for  $v_3, v_4, \dots$ , we see that the number of patterns monotonically decreases. In addition, the maximum length of the vertices is at most  $L + 1$ , which is achieved when  $\mathcal{A}_1 = \{\emptyset\}$  and each of  $v_2, \dots, v_{L+1}$  has a single inner index, and the number of nonredundant hypergraphs is finite. Case (ii) is analyzed in the same way, except the maximum length of the vertices is  $L + 2$ . For the case of large filter sizes, we can discuss a similar method using Proposition 4 that ensures the combination patterns of factoring convolution is also finite.  $\square$

## B Training Recipes

### B.1 Enumeration

**2D** The architecture is Einconv(64)–MaxPooling–Einconv(128)–MaxPooling–FC(10)–Softmax, where Einconv( $k$ ) denotes an Einconv layer with  $k$  output channels and FC( $k$ ) denotes a fully-connected layer with  $k$  output units. Maxpooling is performed by a factor of 2 for each spatial dimension. We trained for 50 epochs using Adam optimizer of the batch size 16 with learning rate 2E-4 and weight decay of rate 1E-6.

**3D** The architecture is Einconv(64)–ReLU–Einconv(128)–ReLU–MaxPooling–Einconv(256)–ReLU–Einconv(256)–ReLU–MaxPooling–Einconv(512)–ReLU–Einconv(512)–ReLU–GAP–FC(512)–FC(512)–FC(10)–Softmax, where GAP denotes global average pooling. We applied dropout with rate 50% to fully-connected layers except the last layer. Other settings were the same as the 2D case.

<sup>6</sup><https://docs.scipy.org/doc/numpy-1.13.0/user/basics.broadcasting.html>

## B.2 GA Search

**LeNet-5** The architecture is Einconv(32)–MaxPooling–Einconv(32)–MaxPooling–FC(10)–Softmax. We trained for at most 250 epochs using Adam optimizer of the batch size 128 with learning rate  $2E-4$  and weight decay of rate  $5E-4$ .

**ResNet-50** The architecture is that we replace all the bottleneck layers in ResNet-50 that do not rescale the spatial size by Einconv layer. We trained for at most 300 epochs using momentum SGD of the batch size 32 and learning rate 0.05 that was halved for every 25 epochs and weight decay of rate  $5E-4$ . Also, we used standard data augmentation methods of random rotation, color lightning, color flip, random expansion, and random cropping.