Neural Networks 105 (2018) 328–339

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Improving efficiency in convolutional neural networks with multilinear filters

Dat Thanh Tran [a,*], Alexandros Iosifidis [b], Moncef Gabbouj [a]

[a] *Laboratory of Signal Processing, Tampere University of Technology, Finland*

[b] *Department of Engineering, Electrical and Computer Engineering, Aarhus University, Denmark*

ARTICLE INFO

Article history:
Received 23 October 2017
Received in revised form 8 May 2018
Accepted 25 May 2018
Available online 7 June 2018

Keywords:
Convolutional neural networks
Multilinear projection
Network compression

ABSTRACT

The excellent performance of deep neural networks has enabled us to solve several automatization problems, opening an era of autonomous devices. However, current deep net architectures are heavy with millions of parameters and require billions of floating point operations. Several works have been developed to compress a pre-trained deep network to reduce memory footprint and, possibly, computation. Instead of compressing a pre-trained network, in this work, we propose a generic neural network layer structure employing multilinear projection as the primary feature extractor. The proposed architecture requires several times less memory as compared to the traditional Convolutional Neural Networks (CNN), while inherits the similar design principles of a CNN. In addition, the proposed architecture is equipped with two computation schemes that enable computation reduction or scalability. Experimental results show the effectiveness of our compact projection that outperforms traditional CNN, while requiring far fewer parameters.

1. Introduction

In recent years, deep neural network architectures have excelled in several application domains, ranging from machine vision (Girshick, Donahue, Darrell, & Malik, 2014; Redmon, Divvala, Girshick, & Farhadi, 2016; Waris, Iosifidis, & Gabbouj, 2017), natural language processing (Graves, Mohamed, & Hinton, 2013; Hinton et al., 2012) to biomedical (An, Kuang, Guo, Zhao, & He, 2014; Zabihi, Rad, Kiranyaz, Gabbouj, & Katsaggelos, 2016) and financial data analysis (Tsantekidis et al., 2017a, b). Of those important developments, Convolutional Neural Network (CNN) has evolved as a main workhorse in solving computer vision tasks nowadays. The architecture was originally developed in the 1990s for handwritten character recognition using only two convolutional layers (LeCun, Bottou, Bengio, & Haffner, 1998). Over the years, with the development of Graphical Processing Units (GPUs) and efficient implementation of convolution operation, the depth of CNNs has been increased to tackle more complicated problems. Nowadays, prominent architectures such as Residual Network (ResNet) (He, Zhang, Ren, & Sun, 2016) or Google Inception (Szegedy et al., 2015) with hundreds of layers have become saturated. Researchers started to wonder whether millions of parameters are essential to achieve such performance. In order to extend the benefit of such deep nets to embedded devices with limited computation power and memory, recent works have focused on reducing the memory footprint and computation of a pre-trained network, i.e. they apply network compression in the post-training stage. In fact, recent works have shown that traditional network architectures such as Alexnet, VGG or Inception are highly redundant structures (Chen, Wilson, Tyree, Weinberger, & Chen, 2015; Denton, Zaremba, Bruna, LeCun, & Fergus, 2014; Gong, Liu, Yang, & Bourdev, 2014; Guo, Yao, & Chen, 2016; Han, Mao, & Dally, 2015; Ioannou, Robertson, Shotton, Cipolla, & Criminisi, 2015; Jaderberg, Vedaldi, & Zisserman, 2014; Lin, Talathi, & Annapureddy, 2016; Tai, Xiao, Zhang, Wang, & Weinan, 2015; Wen, Wu, Wang, Chen, & Li, 2016). For example, in Han et al. (2015) a simple heuristic based on magnitude of the weights was employed to eliminate the connections in a pre-trained network, which achieved considerable amount of compression without hurting the performance much. Additionally, representing network parameters with low bitwidth numbers, like in Gysel, Motamedi, and Ghiasi (2016), Hubara, Courbariaux, Soudry, El-Yaniv, and Bengio (2016) and Zhou, Wang, Wen, He, and Zou (2017), has shown that the performance of a 32-bit network can be closely retained with only 4-bit representations. It should be noted that the two approaches are complementary to each other. In fact, a compression pipeline called "Deep Compression" (Han et al., 2015) which consists of three compression procedures, i.e. weight pruning, weight quantization and Huffman-based weight encoding, achieved excellent compression performance on AlexNet and VGG-16 architectures.

Along pruning and quantization, low-rank approximation of both convolutional layers and fully connected layers was also

[*] Corresponding author.
E-mail address: dat.tranthanh@tut.fi (D.T. Tran).

https://doi.org/10.1016/j.neunet.2018.05.017
0893-6080/© 2018 Elsevier Ltd. All rights reserved.

employed to achieve computational speed up (Denil, Shakibi, Dinh, de Freitas, et al., 2013; Lin, Ji, Guo, Li, et al., 2016; Novikov, Podoprikhin, Osokin, & Vetrov, 2015). Viewed as high-order tensors, convolutional layers were decomposed using traditional tensor decomposition methods, such as CP decomposition (Denton et al., 2014; Jaderberg et al., 2014; Lebedev, Ganin, Rakhuba, Oseledets, & Lempitsky, 2014) or Tucker decomposition (Kim et al., 2015), and the convolution operation is approximated by applying consecutive 1D convolutions.

Overall, efforts to remove redundancy in already trained neural networks have shown promising results by determining networks with a much simpler structure. The results naturally pose the following question: *why should we compress an already trained network and not seek for a compact network representation that can be trained from scratch?*. Subsequently, one could of course exploit the above mentioned compression techniques to further decrease the cost. Under this perspective, the works in Ioannou et al. (2015) and Tai et al. (2015) utilizing a low-rank approximation approach were among the first to report simplified network structures.

The success of Convolutional Neural Networks can be attributed to four important design principles: sparse connectivity, parameter sharing, pooling and multilayer structure. Sparse connectivity (in convolutional layers) only allows local interaction between input neurons and output neurons. This design principle comes from the fact that in many natural data modalities such as images and videos local/neighboring values are often highly correlated. These groups of local values usually contain certain distinctive patterns, e.g. edges and color blobs, in images. Parameter sharing mechanism in CNNs enables the underlying model to learn location invariant cues. In other words, by sliding the filters over the input, the patterns can be detected regardless of the location. Pooling and multilayer structure design of deep neural networks in general and CNN in particular, captures the compositional hierarchies embedded within many natural signals. For example, in facial images, lower level cues such as edges, color and texture patterns form discriminative higher level cues of facial parts, like nose, eyes or lips. Similar compositional structure can be seen in speech or text, which are composed of phonemes, syllables, words and sentences. Although the particular structure of a deep network has evolved over time, the above important design principles remain unchanged. At the core of any convolution layer, each filter with $d \times d \times C$ elements operates as a micro feature extractor that performs linear projection of each data patch/volume from a feature space of d^2C dimensions to a real value. In order to enhance the discrimination power of this micro feature extractor, the authors of Lin, Chen, and Yan (2013) proposed to replace the GLM model by a general nonlinear function approximator, particularly the multilayer perceptron (MLP). The resulting architecture was dubbed Network in Network (NiN) since it consists of micro networks that perform the feature extractor functionality instead of simple linear projection.

In this paper, instead of seeking a more complex feature extractor, we propose to replace the linear projection of the traditional CNN by multilinear projection in the pursuit of simplicity. There has been a great effort to extend traditional linear methods to multilinear ones in an attempt to directly learn from the natural representation of the data as high order tensors (Li & Schonfeld, 2014; Lu, Plataniotis, & Venetsanopoulos, 2008; Tao, Li, Wu, & Maybank, 2007; Thanh, Kanniainen, Gabbouj, & Iosifidis, 2017; Yan et al., 2005; Zhou, Li, & Zhu, 2013). The beauty of multilinear techniques lies in the property that the input tensor is projected simultaneously in each tensor mode, allowing only certain connections between the input dimensions and output dimensions, hence greatly reducing the number of parameters. Previous works on multilinear discriminant learning and multilinear regression (Guo, Kotsia, & Patras, 2012; Li & Schonfeld, 2014; Tao et al., 2007; Thanh et al., 2017; Yan et al., 2005; Zhou et al., 2013) have shown competitive results of multilinear-based techniques. The proposed architecture still inherits the four fundamental design properties of a traditional deep network while utilizing multilinear projection as a generic feature extractor. The complexity of each feature extractor can be easily controlled through the "rank" hyper-parameter. Besides a fast computation scheme when the network is compact, we also propose an alternative computation method that allows efficient computation when complexity increases.

The contribution of our paper can be summarized as follows:

- We propose a generic feature extractor that performs multilinear mapping to replace the conventional linear filters in CNNs. The complexity of each individual feature extractor can be easily controlled via its rank, which is a hyperparameter of the method. By having the ability to adjust individual filter's complexity, the complexity of the entire network can be adjusted without the need of increasing the number of filters in a layer, i.e. the width of the layer. Since the proposed mapping is differentiable, the entire network can be easily trained end-to-end by using any gradient descent-based training process.
- We provide the analysis of computation and memory requirements of the proposed structure. In addition, based on the properties of the proposed mapping, we propose two efficient computation strategies leading to two different complexity settings.
- The theoretical analysis of the proposed approach is supported by experimental results in real-world classification problems, in comparison with CNN and the low-rank scheme in Tai et al. (2015).

The remainder of the paper is organized as follows: In Section 2, we provide an overview of the related works focusing on designing compact network structures. Section 3 gives the necessary notations and definitions before presenting the proposed structure and its analysis. In Section 4, we provide details of our experiment procedures, results and quantitative analysis. Section 5 concludes our work and discusses possible future extensions.

2. Related work

Research focusing on the design of a less redundant network architecture has gained much more attention recently. One of the prominent design pattern is the *bottleneck* unit which was first introduced in the ResNet architecture (He et al., 2016). The *bottleneck* pattern is formed by two 1 × 1 convolution layers with some 3 × 3 convolution layers in between. The first 1 × 1 convolution layer is used to reduce the number of input feature maps while the latter is used to restore the number of output feature maps. Several works such as Szegedy, Ioffe, Vanhoucke, and Alemi (2017), Wang, Liu, and Foroosh (0000) and Zhang, Zhou, Lin, and Sun (2017) have incorporated the *bottleneck* units into their network structure to reduce computation and memory consumed. By utilizing a dense pattern of residual connections, DenseNet architecture (Huang, Liu, Weinberger, & van der Maaten, 2017) has been shown to outperform the original ResNet architectures with fewer parameters. Recently, MobileNet architecture (Howard et al., 2017) was proposed which replaced normal convolution operation by depthwise separable convolution layers. Constituted by depthwise convolution and pointwise convolution, the depthwise separable convolution layer performs the filtering and combining steps independently. The resulting structure is many times more efficient in terms of memory and computation. It should be noted that bottleneck design or depthwise separable convolution layer is a design on a macro level of the network structure in which the arrangements of layers are investigated to reduce computation.

On a micro level, the works in Ioannou et al. (2015) and Tai et al. (2015) assumed a low rank structure of convolutional kernels in order to derive a compact network structure. In fact, low rank assumption has been incorporated into several designs prior to deep neural networks, such as dictionary learning, wavelet transform of high dimensional data. The first incorporation of low rank assumption in neural network compression was proposed in Denton et al. (2014), Jaderberg et al. (2014) and Lebedev et al. (2014). In Lebedev et al. (2014), CP decomposition was proposed to decompose the entire 4D convolutional layer into four 1D convolutions. Although the effective depth of the network remains the same, replacing one convolution operation by four can potentially lead to difficulty in training the network from scratch. With a carefully designed initialization scheme, the work of Ioannou et al. (2015) was able to train a mixture of low-rank filters from scratch with competitive performances. Improving on the idea of Lebedev et al. (2014), a different low-rank structure that allows both approximating an already trained network and training from scratch was proposed in Tai et al. (2015). Specifically, let us denote a convolution layer of N kernels by $\mathcal{W} \in \mathbb{R}^{d \times d \times C \times N}$, where C and d are the number of input feature maps and spatial size of the kernel, respectively. Tai et al. (2015) proposed to approximate $\mathcal{W}$ using a vertical kernel $\mathcal{V} \in \mathbb{R}^{d \times 1 \times C \times K}$ and a horizontal kernel $\mathcal{H} \in \mathbb{R}^{1 \times d \times K \times N}$. The approximation is in the following form:

$$\tilde{\mathcal{W}}_n^c \simeq \sum_{k=1}^{K} \mathcal{V}_k^c (\mathcal{H}_n^k)^T, \tag{1}$$

where the superscript and subscript denote the index of the channel and the kernel respectively. K is a hyper-parameter controlling the rank of the matrix approximation. Here $\mathcal{W}_n^c$ is just the 2D kernel weight of the nth filter applied to the cth channel of the input feature map; $\mathcal{V}_k^c$ and $\mathcal{H}_n^k$ are just d-dimensional vectors.

As can be seen from (1), the authors simplify a convolutional layer by two types of parameter sharing. The first is the sharing of right singular vectors ($\mathcal{H}_n^k$) across all C input channels within the nth filter while the second enforces the sharing of left singular vectors ($\mathcal{V}_k^c$) across all N filters. The work in Tai et al. (2015) is closely related to ours since we avoid designing a particular initialization scheme by including a Batch Normalization step (Ioffe & Szegedy, 2015). The resulting structure was easily trained from scratch with different network configurations. However, different from Tai et al. (2015), our work does not enforce parameter sharing across filters within one convolution layer to achieve compactness. Moreover, using the proposed multilinear projection, each individual filter in the same layer could possess different complexity by having different rank hyper-parameter values. On the contrary, the rank hyper-parameter is shared across filters within the same layer in Tai et al. (2015). It is worth noting that the proposed mapping is complementary to recent developments in architectural level such as ResNet or DenseNet to further reduce the number of parameters required.

3. Proposed method

We start this section by introducing some notations and definitions related to our work. We denote scalar values by either lower-case or upper-case characters ($x, y, X, Y \ldots$), vectors by lower-case bold-face characters ($\mathbf{x}, \mathbf{y}, \ldots$), matrices by upper-case bold-face characters ($\mathbf{A}, \mathbf{B}, \ldots$) and tensors by calligraphic capital characters ($\mathcal{X}, \mathcal{Y}, \ldots$). A tensor is a multilinear matrix with K modes, and is defined as $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_K}$, where I_k denotes the dimension in mode-k. The entry in the i_kth index in mode-k for $k = 1, \ldots, N$ is denoted as $\mathcal{X}_{i_1, i_2, \ldots, i_K}$.

3.1. Multilinear algebra concepts

Definition 1 (*Mode-k Fiber and Mode-k Unfolding*)**.** The mode-k fiber of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_K}$ is a vector of I_k-dimensional, given by fixing every index but i_k. The mode-k unfolding of $\mathcal{X}$, also known as mode-k matricization, transforms the tensor $\mathcal{X}$ to matrix $\mathbf{X}_{(k)}$, which is formed by arranging the mode-k fibers as columns. The shape of $\mathbf{X}_{(k)}$ is $\mathbb{R}^{I_k \times I_{\bar{k}}}$ with $I_{\bar{k}} = \prod_{i=1, i \neq k}^{K} I_i$.

Definition 2 (*Mode-k Product*)**.** The mode-k product between a tensor $\mathcal{X} = [x_{i_1}, \ldots, x_{i_K}] \in \mathbb{R}^{I_1 \times \cdots I_K}$ and a matrix $\mathbf{W} \in \mathbb{R}^{J_k \times I_k}$ is another tensor of size $I_1 \times \cdots \times J_k \times \cdots \times I_K$ and denoted by $\mathcal{X} \times_k \mathbf{W}$. The element of $\mathcal{X} \times_k \mathbf{W}$ is defined as $[\mathcal{X} \times_k \mathbf{W}]_{i_1, \ldots, i_{k-1}, j_k, i_{k+1}, \ldots, i_K} = \sum_{i_k=1}^{I_K} [\mathcal{X}]_{i_1, \ldots, i_{k-1}, i_k, \ldots, i_K} [\mathbf{W}]_{j_k, i_k}$.

For convenience, we denote $\mathcal{X} \times_1 \mathbf{W}_1 \times \cdots \times_K \mathbf{W}_K$ by $\mathcal{X} \prod_{k=1}^{K} \times_k \mathbf{W}_k$.

One of the nice properties of mode-k product is that the result of the projection does not depend on the order of projection, i.e.

$$(\mathcal{X} \times_{k_1} \mathbf{A}) \times_{k_2} \mathbf{B} = (\mathcal{X} \times_{k_2} \mathbf{B}) \times_{k_1} \mathbf{A}. \tag{2}$$

The above property allows efficient computation of the projection by selecting the order of computation.

3.2. Multilinear filter as generic feature extractor

Let $\mathcal{X}_{i,j} \in \mathbb{R}^{d \times d \times C}$ and $\mathcal{W} \in \mathbb{R}^{d \times d \times C}$ denote the input patch centered at spatial location (i, j) and the convolution filter respectively. At the core of a classic CNN, each convolution filter operates as a feature extractor sliding through the input tensor to generate a higher level representation. Specifically, the filter performs the following linear mapping:

$$\mathcal{Y}_{i,j} = f(\mathcal{X}_{i,j}; \mathcal{W}; b) = \langle \mathcal{X}_{i,j}, \mathcal{W} \rangle + b, \tag{3}$$

where $\mathcal{Y}_{i,j}$ and b denote the response at (i, j) and the intercept term respectively. $\langle \cdot, \cdot \rangle$ denotes the dot-product between two tensors. After the above linear projection, a nonlinearity is applied to $\mathcal{Y}_{i,j}$ using the layer's activation function.

We propose to replace the above linear projection by the following multilinear mapping:

$$\mathcal{Y}_{i,j} = \tilde{f}(\mathcal{X}_{i,j}; \tilde{W}; b) = \sum_{r=1}^{R} \mathcal{X}_{i,j} \prod_{k=1}^{3} \times_k \mathbf{w}_k(r)^T + b, \tag{4}$$

where R is the rank hyper-parameter of the projection and $\mathbf{w}_k(r)$ denotes the projection along mode-k. In our case, $\mathbf{w}_1(r) \in \mathbb{R}^d$, $\mathbf{w}_2(r) \in \mathbb{R}^d$, $\mathbf{w}_3(r) \in \mathbb{R}^C$, $\forall r = 1, \ldots, R$.

It should be noted that Eq. (4) represents the mapping of an individual filter, hence, different filters within one layer could have different ranks. Since the mapping in Eq. (4) operates on similar input patch and yields a scalar response as a linear mapping does in CNNs, the proposed multilinear mapping acts as a generic feature extractor and can be incorporated into any design of the CNN topology, such as AlexNet (Krizhevsky, Sutskever, & Hinton, 2012), VGG (Simonyan & Zisserman, 2014), Inception (Szegedy et al., 2017) or ResNet (He et al., 2016). In addition, since the mapping in Eq. (4) is differentiable with respect to each individual weight vector $\mathbf{w}_k(r)$, the resulting network architecture can be trained in an end-to-end fashion by back propagation algorithm. We hereby denote the layer employing our proposed multilinear mapping as MLconv.

Recently, mode-k multiplication has been introduced as a tensor contraction layer in Kossaifi, Khanna, Lipton, Furlanello, and Anandkumar (2017) to project the entire input layer as a high-order tensor to another tensor. This is fundamentally different

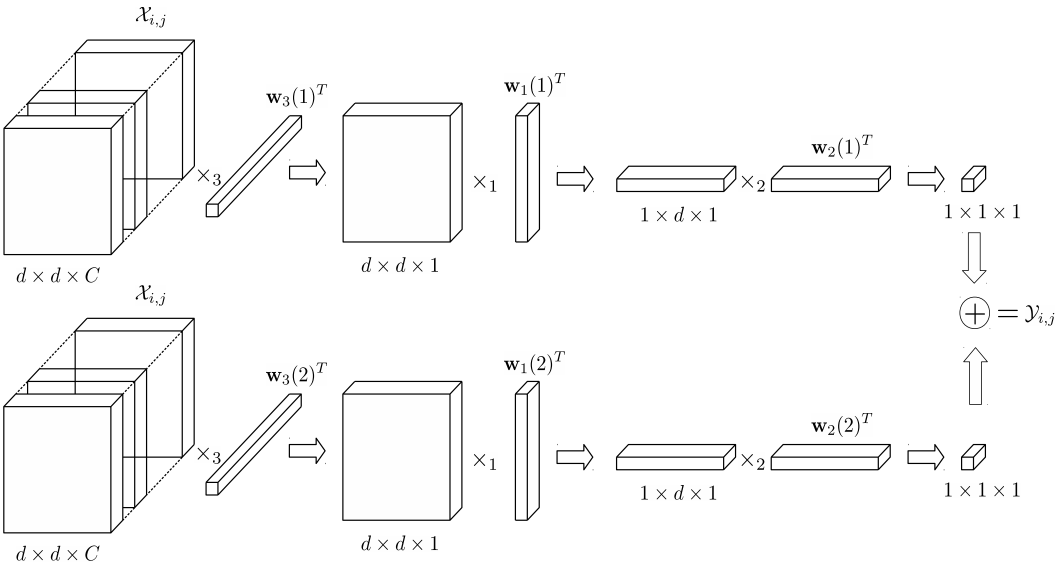

Fig. 1. Illustration of the proposed multilinear mapping according to Eq. (6) with $R = 2$ in sequence: mode-3, mode-1 and mode-2. Here the bias term is omitted.

from our approach since the tensor contraction layer is a global mapping which does not incorporate sparse connectivity and parameter sharing principles. In general, mode-k multiplication can be applied to an input patch/volume to output another tensor instead of a scalar as in our proposal. We restrict the multilinear projection in the form of Eq. (4) to avoid the increase in the output dimension which leads to computation overhead in the next layer. Moreover, tensor unfolding operation required to perform the multilinear projection that transforms a tensor to another tensor will potentially increase the computation. On the contrary, our proposed mapping is a special case of the general multilinear mapping using mode-k product in which the output tensor degenerates to a scalar. This special case allows efficient computation of the projection, as shown in the next section.

3.3. Memory and computation complexity

For simplicity, in the following section, we consider only the case when all filters in one network have the hyper-parameter R. One the most obvious advantages of the mapping in Eq. (4) is that it requires far fewer parameters to estimate the model, compared to the linear mapping in a CNN. In a CNN utilizing the mapping in Eq. (3), a layer with N kernels requires the storage of d^2CN parameters. On the other hand, a similar layer configuration with N mappings utilizing the projection in Eq. (4) requires only $R(2d + C)N$ parameters. The gain ratio is:

$$\frac{d^2C}{R(2d + C)}. \tag{5}$$

As compared to a similar CNN topology, the memory reduction utilizing the mapping in Eq. (4) varies for different layers. The case where $C \gg d$ (which is the usual case) leads to a gain ratio approximately equal to d^2/R. In our experiments, we have seen that with $d = 3$ and $R = 2$ in all layers, memory reduction is approximately 4×, while having competitive performance compared to a CNN with similar network topology.

Let us denote by $\mathcal{X}_l \in \mathbb{R}^{X \times Y \times C}$ and $\mathcal{W}_l \in \mathbb{R}^{d \times d \times C \times N}$ the input and N kernels of the lth convolutional layer having C input feature maps and N output feature maps. In addition, we assume zero-padding and sliding window with stride of 1. By using linear projection as in case of CNN, the computational complexity of this layer is $O(d^2XYCN)$. Before evaluating the computational cost of a layer using the proposed method, it should be noted that the projection in Eq. (4) can be efficiently computed by applying three consecutive convolution operations. Details of the convolution operations depend on the order of three modes. Therefore, although the result of the mapping in Eq. (4) is independent of the order of mode-k projection, the computational cost actually depends on the order of projections. For $C \gg d$, it is computationally more efficient to first perform the projection in mode-3 in order to reduce the number of input feature maps for subsequent mode-1 and mode-2 projection:

$$\mathcal{Y}_{i,j} = \sum_{r=1}^{R} \mathcal{X}_{i,j} \times_3 \mathbf{w}_3(r)^T \times_1 \mathbf{w}_1(r)^T \times_2 \mathbf{w}_2(r)^T + b. \tag{6}$$

The response $\mathcal{Y}_{i,j}$ in Eq. (6) is the summation of R independent projections with each projection corresponding to the following three consecutive steps, as illustrated in Fig. 1:

- Projection of $\mathcal{X}_{i,j}$ along the third mode which is the linear combination of C input feature maps. The result is a tensor $\mathcal{X}_{i,j}^{(3)}$ of size $d \times d \times 1$.
- Projection of $\mathcal{X}_{i,j}^{(3)}$ along the first mode which is the linear combination of d rows. The result is a tensor $\mathcal{X}_{i,j}^{(1)}$ of size $1 \times d \times 1$.
- Projection of $\mathcal{X}_{i,j}^{(1)}$ along the second mode which is the linear combination of d elements.

With the aforementioned configuration of the lth layer, the computational complexity of the lth MLconv layer utilizing our multilinear mapping is as follows:

- Mode-3 projection that corresponds to applying NR convolutions to the input $\mathcal{X}_l$ with kernels of size $1 \times 1 \times C$ elements, having computational complexity of $O(XYCNR)$. The output of the projection along the third mode is a tensor of size $X \times Y \times NR$.
- Mode-1 projection is equivalent to applying convolution with one $d \times 1 \times NR$ separable convolution kernel, having complexity of $O(dXYNR)$. This results in a tensor of size $X \times Y \times NR$.
- Mode-2, similar to mode-1 projection, can be computed by applying convolution with one $1 \times d \times NR$ separable convolution kernel, requiring $O(dXYNR)$ computation. This results in a tensor of size $X \times Y \times NR$. By summing over R ranks, we arrive at the output of layer l of size $X \times Y \times N$.

The total complexity of layer l using our proposed mapping is thus $O(XYNR(C+2d))$. Compared to linear mapping, our method achieves computational gain of:

$$\frac{d^2C}{R(C+2d)}. \tag{7}$$

From Eqs. (5) and (7), we can conclude that the proposed feature extractor achieves approximately d^2/R savings in both computation and memory when $C \gg d$.

3.4. Initialization with pre-trained CNN

The proposed mapping in Eq. (4) can be viewed as a constrained form of convolution kernel as follows:

$$\mathcal{Y}_{i,j} = \langle \mathcal{X}_{i,j}, \tilde{\mathcal{W}} \rangle + b, \tag{8}$$

where $\tilde{\mathcal{W}}$ is expressed in Kruskal form $\tilde{\mathcal{W}} = \sum_{r=1}^{R} \mathbf{w}_1(r) \circ \mathbf{w}_2(r) \circ \mathbf{w}_3(r)$ as the outer-product of the corresponding projection in three modes. By calculating $\mathcal{Y}_{i,j}$ using mode-k product definition as in Eq. (4) and using dot-product as in Eq. (8), the equivalence of Eqs. (4) and (8) can be found (Kolda & Bader, 2009).

Consequently, a convolution layer can be converted to an MLconv layer by decomposing each 3D convolution filter into Kruskal form using any CP decomposition method (Kolda & Bader, 2009). It should be noted here that, since there is no closed-form solution of the CP decomposition, such a conversion corresponds to an approximation step. Under this perspective, a pre-trained CNN can be used to initialize our network structure to speed up the training process. However, as we will show in the experimental section, random initialization of multilinear filters can lead to better performance.

In addition to an initialization scheme, Eq. (8) also complements our proposed mapping with an efficient computation strategy when R is large. The computation cost discussed in the previous subsection depends linearly with parameter R. When R is large, it is more efficient to compute the mapping according to Eq. (8) by first calculating $\tilde{\mathcal{W}}$ and then convolving the input with $\tilde{\mathcal{W}}$. The computational complexity of the first step is $O(d^2CRN)$ while for the convolution step is $O(d^2XYCN)$, resulting to an overall complexity of $O(d^2CRN + d^2XYCN)$ for the entire layer. The ratio between normal convolution layer and MLconv layer using this computation strategy is:

$$\frac{XY}{R+XY}. \tag{9}$$

It is clear that XY is usually much larger than R, therefore, the increase in computation as compared to normal convolution is marginal. Following this calculation strategy, a rank 6 network is marginally slower than a rank 1 network or a CNN. This will be demonstrated in our experiment section. In conclusion, the computation method discussed in this subsection allows the scalability of our proposed mapping when R is large while previous subsection proposes an efficient computation scheme that allows computation savings when R is small. Overall, we can conclude that the computation of the proposed layer structure is efficient while, as will be shown in the experimental evaluation, changing the rank of the adopted tensor definitions can increase performance.

4. Experiments

In this section, we provide experimental results to support the theoretical analysis in Section 3. The experimental protocol and datasets are described first, followed by the discussion of the experimental results.

4.1. Datasets

4.1.1. CIFAR-10 and CIFAR-100

CIFAR dataset (Krizhevsky & Hinton, 2009) is an object classification dataset which consists of 50k color images for training and 10k for testing with the resolution 32 × 32 pixels. CIFAR-10 refers to the 10-class classification problem of the dataset in which each class has 5000 images for training and 1000 images for testing while CIFAR-100 refers to a more fine-grained classification of the images into 100 classes.

4.1.2. SVHN

SVHN (Netzer et al., 2011) is a well-known dataset for handwritten digit recognition problem which consists of more than 600k images of house numbers extracted from natural scenes with varying number of samples from each class. This dataset poses a much harder character recognition problem as compared to the MNIST dataset (LeCun et al., 1998). We used 32 × 32 cropped images provided by the database from which each individual image might contain some distracting digits on the sides.

4.2. Tiny-ImageNet

Tiny-ImageNet dataset[1] is an object classification dataset with images coming from the ImageNet dataset (Deng et al., 2009). The dataset contains real-world images belonging to 200 classes with 100k images for training and 50k for validation. All of the images are down-sampled to the same resolution, 64 × 64 pixels.

4.3. Network topology

Traditional CNN topology consists of two modules: feature extractor module and classifier module. Several convolution and pooling layers stacked on top of each other act as feature extractor while one or two fully-connected layers act as the classifier. In order to evaluate the effectiveness of the proposed multilinear filter, we constructed the network architectures with only feature extractor layers, i.e. convolution layer or MLconv layer together with pooling layer while skipping fully-connected layer. As the name suggests, fully-connected layer has dense connections, accounting for large number of parameters in the network while being prone to overfitting. Moreover, a powerful and effective feature extractor module is expected to produce a highly discriminative latent space in which the classification task is made simple. Such fully-convolutional networks have attracted much attention lately due to their compactness and excellent performance in image-related problems like semantic segmentation, object localization and classification (Lin et al., 2013; Redmon et al., 2016; Ronneberger, Fischer, & Brox, 2015)

In our experiments, we have tested two different network topologies: a 9-layer convolution topology similar to Springenberg, Dosovitskiy, Brox, and Riedmiller (2014) on CIFAR and SVHN datasets and an 18-layer convolution topology with residual connections that resembles ResNet18 (He, Zhang, Ren, & Sun, 2015) on Tiny-ImageNet dataset. Details of the network configurations can be found in Appendix A.

Based on the configuration of the network topologies, we compare the performance between standard linear convolution kernel (CNN), our proposed multilinear kernel (MLconv) and the low-rank (LR) structure proposed in Tai et al. (2015).

[1] https://tiny-imagenet.herokuapp.com/.

Table 1
CIFAR-10 classification error (%).

	CNN	MLconv1	LR26	MLconv2	LR53	MLconv4	LR106
Scratch	7.47	8.54	9.14	7.68	8.31	7.34	8.00
Pretrained	–	8.17	8.64	7.76	7.49	7.38	7.10
# Parameters	≈1.38M	≈0.20M	≈0.20M	≈0.35M	≈0.35M	≈0.65M	≈0.64M

Table 2
CIFAR-100 classification error (%).

	CNN	MLconv1	LR26	MLconv2	LR53	MLconv4	LR106
Scratch	29.60	31.32	35.79	29.10	31.45	28.27	30.11
Pretrained	–	31.88	33.98	29.86	30.00	28.45	28.40
# Parameters	≈1.39M	≈0.21M	≈0.21M	≈0.37M	≈0.37M	≈0.67M	≈0.67M

4.4. Experimental settings

All networks were trained using both SGD optimizer (Rumelhart, Hinton, Williams, et al., 1988) as well as Adam (Kingma & Ba, 2014). While the proposed structure tends to arrive at better minima with Adam, this is not the case for the other two methods.

Regarding data augmentation, for CIFAR dataset, random horizontally flipped samples were added as well as random translation of the images by maximum 5 pixels were performed during the training process; for SVHN dataset, only random translation of maximum 5 pixels was performed; for Tiny-ImageNet, random crops of sizes between 56 and 64 pixels were taken during training and 10 random crops were taken during evaluation. For all datasets, no further preprocessing step was applied. For regularization, both weight decay and max-norm (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014) are individually and together exploited in our experiments. Max-norm regularizer was introduced in Srivastava et al. (2014) where it was used together with Dropout.

Details of the optimizer settings, learning rate schedules and regularization parameters are given in Appendix B.

For MLconv and LR structures, we experimented with several values for the rank parameter, namely R for the proposed mapping and K in Eq. (1) from Tai et al. (2015). In all of our experiments, we made no attempt to optimize R and K for each of individual filter and layer in order to get the maximal compact structure, since such an approach is impractical in real cases. We instead used the same rank value throughout all layers. The experiments are, hence, different from Tai et al. (2015) where the authors reported performance for different values of K at each layer without discussing the rank selection method. The experiments were conducted with $R = 1, 2, 4$ and the corresponding structures are denoted as MLconv1, MLconv2, MLconv4. The values of K are selected so that the number of parameters in an LR network is similar to the number of parameters of its MLconv counterpart with given R. The corresponding LR structures are denoted as LR26, LR53 and LR106 for 9-layer configuration and LR35, LR70, LR140 for 18-layer configuration, where the number denotes the value of K.

All of three competing structures training from scratch were initialized with random initialization scheme proposed in He et al. (2015). For CIFAR dataset, we additionally trained MLconv and LR structure with weights initialized from an optimal pre-trained CNN. The aforementioned protocols were also applied for this configuration. The weights of MLconv were initialized with CP decomposition using canonical alternating least square method (Kolda & Bader, 2009), while for the LR structure we followed the calculation proposed in Tai et al. (2015).

4.5. Experimental results

After obtaining the optimal hyper-parameter values, each network was trained for five times and the median value is reported.

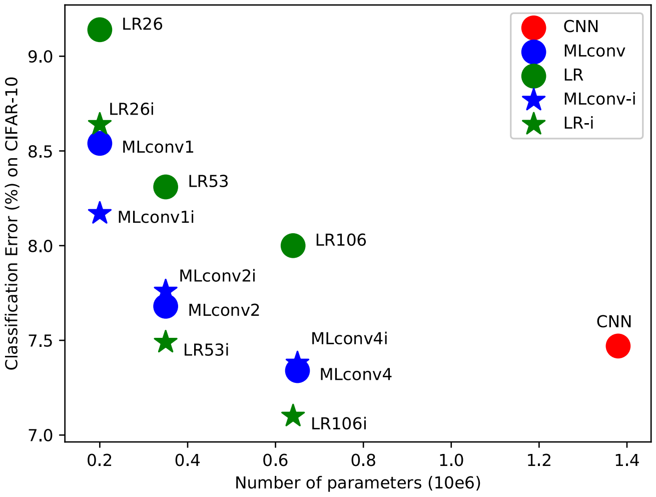

Fig. 2. Model size versus classification error on CIFAR-10 for different structures. MLconv and LR network initialized with CNN marked with "i" at last.

4.5.1. CIFAR and SVHN on 9-layer configuration

The second row of Tables 1 and 2 shows the classification errors of all competing methods trained from scratch on CIFAR-10 and CIFAR-100, respectively, while the third row shows the performance when initialized with a pre-trained CNN. The last row reports the model size of each network. As can be seen from both Tables 1 and 2, using the proposed multi-linear filters leads to a 2× reduction in memory, while outperforming the standard convolution filters in both coarse and fine-grained classification in CIFAR datasets. More interestingly, in CIFAR-100, a rank 4 multilinear filter network attains an improvement over 1%. In both CIFAR-10 and CIFAR-100, constraining $R = 2$ gains 4× memory reduction while keeping the performance relatively closed to the baseline CNN with less than 0.5% increment in classification error. Further limiting R to 1 maximizes the parameter reduction to nearly 7× with the cost of 1.07% and 1.72% increase in error rate for CIFAR-10 and CIFAR-100, respectively. A graphical illustration of the compromise between number of network's parameters and classification error on CIFAR-10 and CIFAR-100 is illustrated in Figs. 2 and 3, respectively. Figs. 4 and 5 show the training and validation accuracies of the proposed MLconv and CNN networks on CIFAR datasets. As can be observed, the MLconv and CNN networks lead to similar performance curves, for both training and validation sets, while all networks having similar speed of convergence. Moreover, It can be seen in Fig. 4 that CNN tends to overfit on the training data.

The classification error of each competing network trained from scratch on SVHN dataset is shown in Table 3. Using our proposed MLconv layers, we achieved 4× reduction in model size while slightly outperforming CNN. At the most compact configuration of

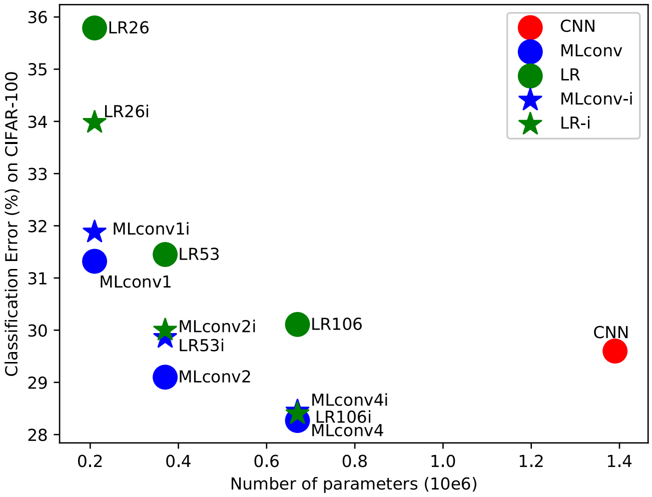

Fig. 3. Model size versus classification error on CIFAR-100 for different structures. MLconv and LR network initialized with CNN marked with "i" at last.

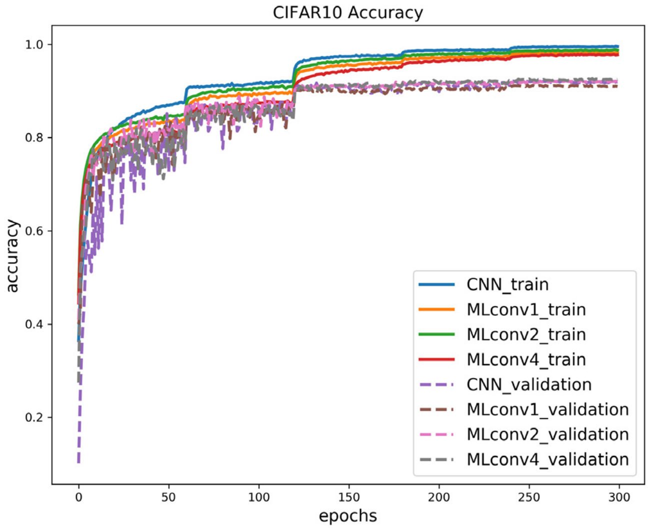

Fig. 4. Training and validation curves on CIFAR10.

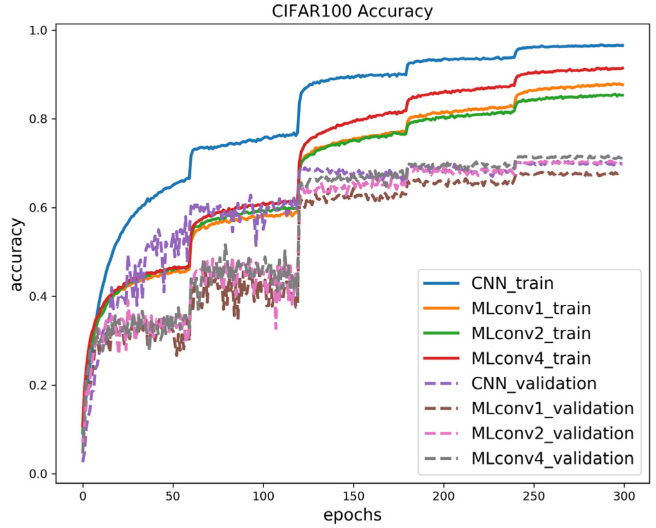

Fig. 5. Training and validation curves on CIFAR100.

Table 3
SVHN classification error.

	Error (%)	#Parameters
CNN	1.80	≈1.38M
MLconv1	1.92	≈0.20M
LR26	1.96	≈0.20M
MLconv2	1.76	≈0.35M
LR53	1.85	≈0.35M
MLconv4	1.75	≈0.65M
LR106	1.78	≈0.64M

MLconv structure, i.e. MLconv1, we only observed a small increment of 0.12% in classification error as compared to CNN baseline. While the gaps between all competing methods in SVHN dataset are relatively small, MLconv consistently outperforms LR structures in all configurations. This also shows that with an easier problem, we only need 200k parameters to achieve similar accuracy with traditional CNN instead of 1.38 million parameters.

Comparing the proposed multi-linear filter with the low rank structure LR, all configurations of MLconv network significantly outperform their LR counterparts. Specifically, in the most compact configuration, MLconv1 is better than LR26 by 0.6% and 4.47% on CIFAR-10 and CIFAR-100, respectively. The margin shrinks as the complexity increases but the proposed structure consistently outperforms LR when training the network from scratch. Similar comparison results can be observed on SVHN dataset. As opposed to the experimental results reported in Tai et al. (2015), we observed inferior results of the LR structure compared to standard CNN when training from scratch. The difference might be attributed to two main reasons: we incorporated batch normalization into the baseline CNN which could potentially improve the performance of the baseline CNN; our baseline configuration has no fully-connected layer to solely benchmark the efficiency of different filter structures as a feature extractor.

One interesting phenomenon was observed when we initialized MLconv and LR with a pre-trained CNN. For the LR structure, most configurations enjoy substantial improvement by initializing the network with weights decomposed from a pre-trained CNN on CIFAR dataset. The contrary happens for our proposed MLconv structure, since most configurations observe a degradation in performance. This can be explained by the fact that LR structure was designed to approximate each individual 2D convolution filter at every input feature map and the resulting structure comes with a closed-form solution for the approximation. With good initialization from a CNN, the network easily arrived at a good minimum while training a low-rank setting from scratch might have difficulty at achieving a good local minimum. Although the proposed mapping can be viewed as a form of convolution filter, the mapping in Eq. (4) embeds a multi-linear structure, hence possessing certain degree of difference. Initializing the proposed mapping by applying CP decomposition, which has no closed-form solution, may lead the network to a suboptimal state.

4.5.2. Tiny-ImageNet on 18-layer topology

Table 4 shows the classification errors on Tiny-ImageNet dataset. The evaluated topology has 18 convolution layers, many of which have far more filters than the 9-layer topology used to classify CIFAR and SVHN dataset. As a result, even with a rank 1 network, we achieved a lower classification error on the validation set with a 4% margin compared to CNN and with the cost of only 14% of parameters of the CNN, i.e. 7× less. As the complexity of the filters in our MLconv networks increases, we observe similar improvements in the classification errors, particularly 41.92% and 40.76% for rank 2 and rank 4 configuration respectively. Compared

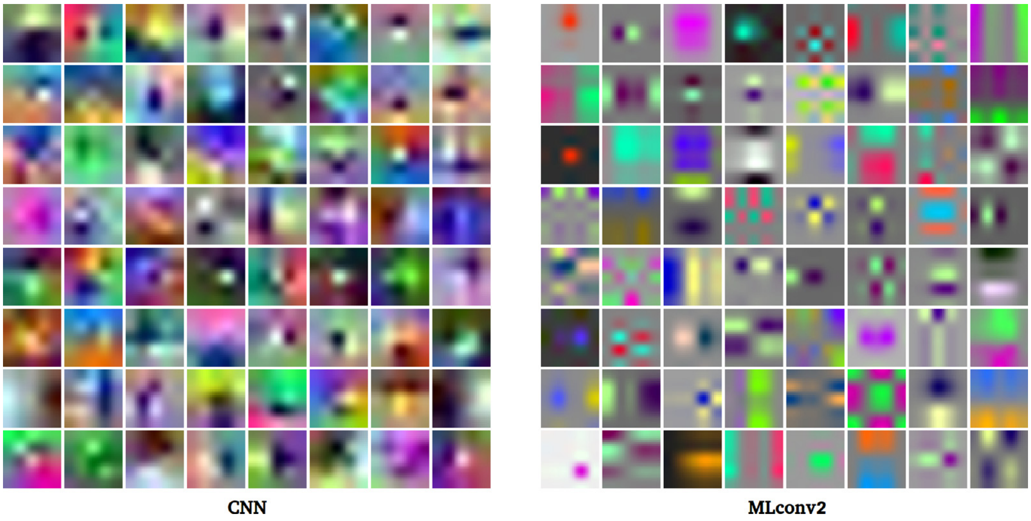

Fig. 6. Filters in the first convolution layer of 18-layer topology. Left side: CNN, Right side: MLconv2.

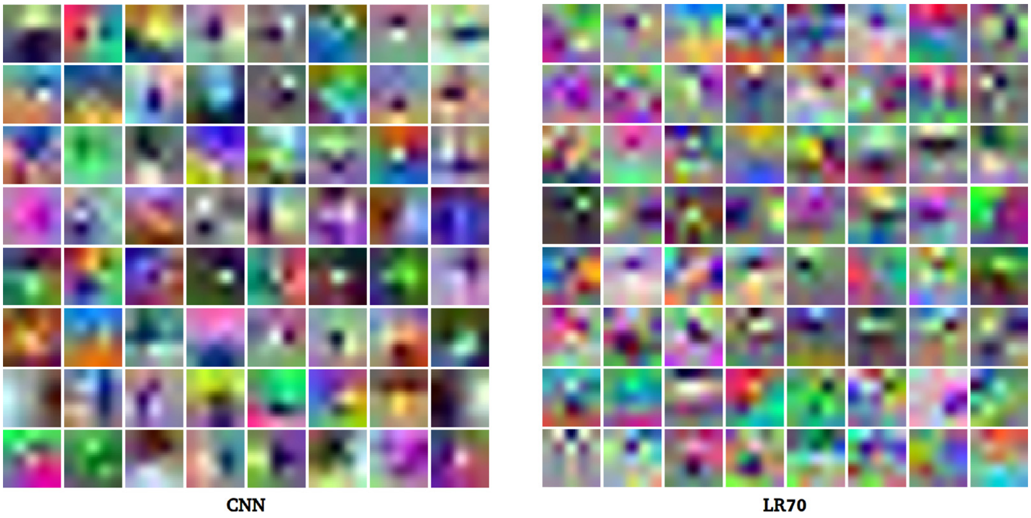

Fig. 7. Filters in the first convolution layer of 18-layer topology. Left side: CNN, Right side: LR70.

Table 4
Tiny-ImageNet classification errors.

	Error (%)	#Parameters
CNN	47.83	≈5.26M
MLconv1	43.44	≈0.755M
LR26	49.44	≈0.755M
MLconv2	41.92	≈1.34M
LR53	48.45	≈1.34M
MLconv4	40.76	≈2.50M
LR106	47.81	≈2.50M

with LR networks, our MLconv networks establish a consistent margin of about 7% lower in classification errors in all 3 configurations of complexity.

Figs. 6 and 7 visualize 64 filters in the first convolution layer of the 18-layer topology coming from CNN versus MLconv2 and LR70 configuration respectively. Images of the filters from other configurations such as MLconv1, MLconv4, LR35, LR140 are given in Appendix C. While the filtering patterns of the baseline CNN and LR network are very much similar, filters in our MLconv networks have very distinctive patterns. Since the proposed mapping is a multilinear mapping, the MLconv filters model the linear dependencies along the horizontal, vertical and the depth dimension, producing a variety of patterns on the right side of Fig. 6 such as the low-pass filters represented by a single blob (top left and bottom left filter) or the high-pass filters in the horizontal axis (position (8,8), (5,3), (8,4), . . .) or the high-pass filters in the vertical axis (position (7,8), (5,8)) and high-pass filters in both directions with grid-like patterns. Although the proposed mapping can be seen as a form of low-rank approximation of the CNN filter as discussed in Section 3.4, Fig. 6 shows that the proposed filters clearly have distinctive filtering patterns as compared to CNN or LR. The rich filtering patterns appeared in our MLconv networks might explain why MLconv networks largely outperform other competing structures.

Experiments on Tiny-ImageNet with a large topology indicate the advantage of the proposed mapping: while increasing the hierarchical representation of a topology by increasing the number of layers and filters, MLconv networks scale favorably in terms of capacity, i.e. the number of parameters. Even with 18 layers with more than a hundred of filters each, a rank 1 MLconv configuration only requires 800k parameters.

Table 5
Forward computation time on 9-layer topology (normalized with respect to CNN).

	MLconv1	LR26	MLconv2	LR53	MLconv4	LR106	MLconv1*	MLconv2*	MLconv4*
Theory	8.14×	6.48×	4.11×	3.20×	1.60×	1.38×	0.997×	0.993×	0.987×
Implementation	1.77×	1.79×	1.33×	1.54×	1.03×	1.23×	0.990×	0.980×	0.960×

Table 6
Forward computation time on 18-layer topology (normalized with respect to CNN).

	MLconv1	LR35	MLconv2	LR70	MLconv4	LR140	MLconv1*	MLconv2*	MLconv4*
Theory	6.18×	3.14×	3.58×	1.58×	1.94×	0.79×	0.998×	0.995×	0.990×
Implementation	2.07×	1.75×	1.44×	1.37×	0.97×	0.96×	0.961×	0.925×	0.917×

4.5.3. *Computational complexity*

Tables 5 and 6 report the average forward propagation time of a single sample measured on CPU for all three network structures on 9-layer topology and 18-layer topology respectively. The second row reports the theoretical gain measured by the number of multiply-accumulate operations while the third row reports the actual computation time measured on the same machine with the same setup. Both rows show the normalized numbers with respect to their convolution counterparts. For the proposed MLconv structure, we report the computation cost of both calculation strategies discussed in Section 3. We refer to the first calculation strategy using the separable convolution as Scheme 1, while the latter one using Kruskal form and normal convolution as Scheme 2. Results from Scheme 2 are denoted with the asterisk (*). All the networks are implemented using Keras library (Chollet, 2015) with Tensorflow (Abadi et al., 2015) backend.

Here we should note that the actual time taken by a network is not necessarily proportional to the number of multiply-accumulate operations since some of the operations can be performed in a concurrent manner while others have to be in a sequential manner. This can be observed from the statistic of the LR140 configuration in 18-layer topology. Although the number of multiply-accumulate of the corresponding CNN network is almost 20% less than the LR140 configuration, the actual computation time of the two networks is similar. This could be explained by the fact that the LR structure decomposes a standard convolution layer into two smaller convolution layers that require fewer memory. Since each convolution layer in the LR structure requires fewer parameters, all parameters and the input can potentially fit into CPU cache, allowing more efficient computation as compared to the CNN, especially in case of a CNN with large layers.

Another point worth noting is that the number of multiply-accumulate operations still reflects the total amount of computation required by a structure, which is proportional to the energy consumption of the structure, playing an important role in mobiles or embedded systems. In this aspect, it is clear that the proposed MLconv structures require fewer computations as compared to the LR structures, being more energy-efficient for power-hungry devices.

While the actual speed-up of the MLconv and the LR method are on the same order, our proposed MLconv networks have the potential to improve on the actual computation time since our current implementation relies on an unoptimized depthwise convolution operator that performs a for loop over the input channels. In fact, at the time of writing, implementation of depthwise convolution operation is still missing in most libraries since there exists no dedicated CUDA kernel for this operation.

Since the computation Scheme 2 relies on the standard convolution operation similar to CNN, the only additional computation step is the calculation of the Kruskal form as presented in Section 3.4. Thus, the actual computation time taken by Scheme 2 is just slightly longer than the CNN counterparts for all ranks. This shows the scalability of the proposed projection with high complexities, i.e. high ranks.

Table 7
9-layer topology.

Input layer
3 × 3 × 96 - BN - LReLU
3 × 3 × 96 - BN - LReLU
3 × 3 × 96 - BN - LReLU
2 × 2 MaxPooling
3 × 3 × 192 - BN - LReLU
3 × 3 × 192 - BN - LReLU
3 × 3 × 192 - BN - LReLU
2 × 2 MaxPooling
3 × 3 × 192 - BN - LReLU
1 × 1 × 192 - BN - LReLU
1 × 1 × *#class* LReLU
Global average over spatial dimension
Softmax activation

Table 8
18-layer topology with residual connections.

Input layer
5 × 5 × 64 - BN - ReLU
3 × 3 × 64 - Residual block
3 × 3 × 64 - Residual block
2 × 2 MaxPooling
3 × 3 × 128 - Residual block
3 × 3 × 128 - Residual block
2 × 2 MaxPooling
3 × 3 × 256 - Residual block
3 × 3 × 256 - Residual block
2 × 2 MaxPooling
3 × 3 × 256 - Residual block
3 × 3 × 256 - Residual block
1 × 1 × 200 LReLU
Global average over spatial dimension
Softmax activation

5. Conclusions

In this paper, we proposed a multilinear mapping to replace the conventional convolution filter in Convolutional Neural Networks. The resulting structure's complexity can be flexibly controlled by adjusting the number of projections in each mode through a hyper-parameter R. The proposed mapping comes with two computation schemes which either allow memory and computation reduction when R is small, or the scalability when R is large. Numerical results showed that with far fewer parameters, architectures employing our mapping could outperform standard CNNs. These are promising results and open future research directions focusing on optimizing parameter R on individual convolution layers to achieve the most compact structure and performance.

Acknowledgement

The work has been partially supported by Academy of Finland Project 289364.

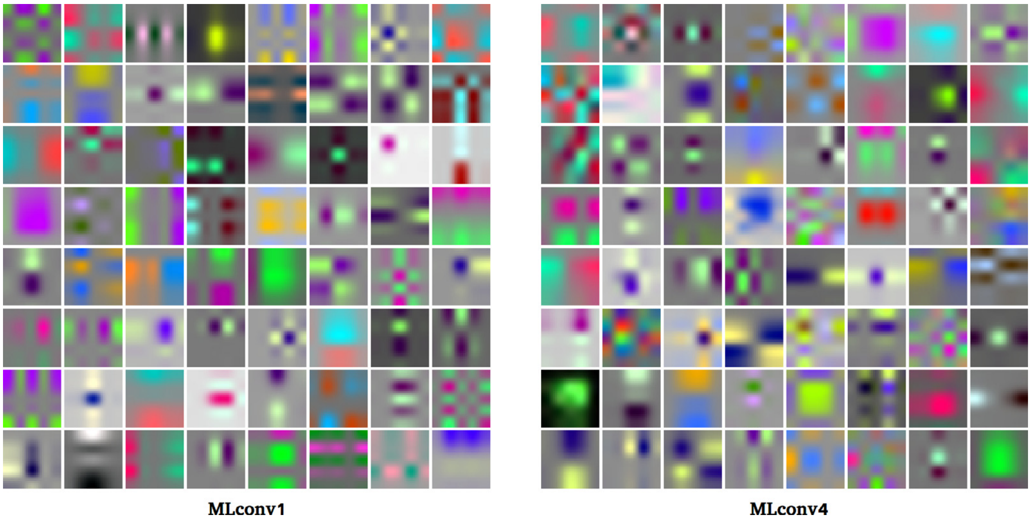

Fig. A.8. Filters in the first convolution layer of 18-layer topology. Left side: MLconv1, Right side: MLconv4.

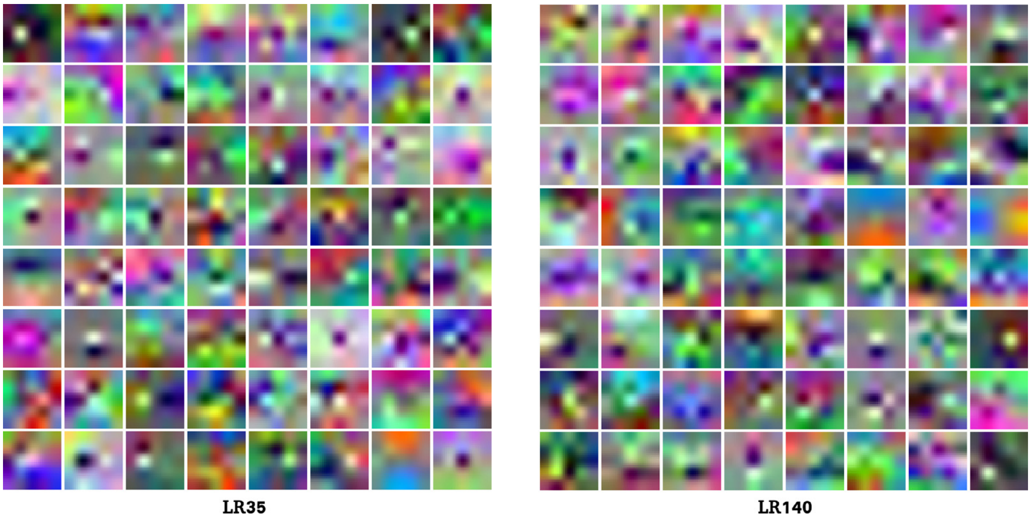

Fig. A.9. Filters in the first convolution layer of 18-layer topology. Left side: LR35, Right side: LR140.

Appendix A. Network topology

The configuration of the two topologies adopted in our experiment benchmark is shown in Tables 7 and 8 where 3 × 3 × N denotes N kernels with 3 × 3 spatial dimension, BN denotes Batch Normalization (Ioffe & Szegedy, 2015) and LReLU, ReLU denote Leaky Rectified Linear Unit (Maas, Hannun, & Ng, 2013) with $\alpha = 0.2$ and Rectified Linear Unit respectively. While the 9-layer topology is similar to the one proposed in Springenberg et al. (2014), the 18-layer topology resembles ResNet-18 in He et al. (2015). There are, however, some key differences. Firstly, we choose to retain a proper pooling layer instead of performing convolution with a stride of 2 in 9-layer topology as proposed in Springenberg et al. (2014). Secondly, Batch Normalization was applied after every convolution layer except the last one where the output goes through softmax to produce the class probability. In addition, LReLU activation was used in 9-layer topology while in 18-layer topology we used ReLU activation unit. As the experiment results in Section 4.5 show that our proposed mapping acts as a generic feature extractor that works well with both type of activations.

Appendix B. Training configuration

For SGD optimizer, the momentum was fixed to 0.9. In CIFAR and SVHN dataset, we adopted two sets of learning rate schedule $SC_1 = \{0.01, 0.005, 0.001, 0.0005, 0.0001\}$ and $SC_2 = \{0.01, 0.001, 0.0001\}$. Each schedule has initial learning rate $\gamma \in 0.01, 0.001$ and decreases to the next value after E epochs where E was cross-validated from the set $\{40, 50, 60, 100, 120\}$. We trained each network with maximum of 300 and 100 epochs for CIFAR, SVHN. The batch size was fixed to 200 samples for all competing networks. In Tiny-ImageNet experiments, the learning rate schedule of LR and CNN networks is 0.01, 0.001, 0.0001 which changes at epoch 60, 120. For MLconv structure, the learning rate schedule is 0.001, 0.0001, 0.00001 which changes at epoch 80 and 110. All networks were trained for maximum 140 epochs on Tiny-ImageNet dataset.

During the training process, the l_2 norm of each individual filter is constrained to lie inside the ball of a given radius r which was cross-validated from the set $\{0.8, 1.0, 2.0, 4.0, 6.0, 8.0\}$. The weight decay hyper-parameter λ was searched from the set $\{0.001, 0.0005, 0.0001\}$. In addition, Dropout with $p_i = 0.2$ was applied to the input and Dropout with $p = p_o$ was applied to the output of all pooling layers in 9-layer topology with the optimal p_o obtained from the set $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. No dropout was applied to the residual topology similar to He et al. (2015).

Due to the differences between the three competing structures, we observed that while the baseline CNN and LR networks work well with weight decay, applying weight decay to the proposed network structure tends to drive all the weight values close to zeros when λ is large, or the regularization effect is marginal when using a small value for λ, leading to the exhaustive search of suitable hyper-parameter λ. On the other hand, max-norm regularization works well with our method without being too sensitive to the performance.

Appendix C. Filters visualization

The filters in the first layer of MLconv1 and MLconv4 networks are shown in Fig. A.8 and the corresponding filters of LR35 and LR140 networks are shown in Fig. A.9. Generally, the two approaches show different filter patterns but the patterns generated by the same method are similar when changing the complexity, i.e. the ranks.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., & Citro, C. et al., (2015). TensorFlow: Large-scale machine learning on heterogeneous systems, Software available from tensorflow.org. https://www.tensorflow.org/.

An, X., Kuang, D., Guo, X., Zhao, Y., & He, L. (2014). A deep learning method for classification of eeg data based on motor imagery. In *International conference on intelligent computing* (pp. 203–210). Springer.

Chen, W., Wilson, J. T., Tyree, S., Weinberger, K. Q., & Chen, Y. (2015). Compressing convolutional neural networks, arXiv preprint arXiv:1506.04449.

Chollet, F. (2015). keras, https://github.com/fchollet/keras.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *IEEE conference on computer vision and pattern recognition, 2009* (pp. 248–255). IEEE.

Denil, M., Shakibi, B., Dinh, L., de Freitas, N., et al. (2013). Predicting parameters in deep learning. In *Advances in neural information processing systems* (pp. 2148–2156).

Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., & Fergus, R. (2014). Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems* (pp. 1269–1277).

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 580–587).

Gong, Y., Liu, L., Yang, M., & Bourdev, L. (2014). Compressing deep convolutional networks using vector quantization, arXiv preprint arXiv:1412.6115.

Graves, A., Mohamed, A.-r., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing* (pp. 6645–6649). IEEE.

Guo, W., Kotsia, I., & Patras, I. (2012). Tensor learning for regression. *IEEE Transactions on Image Processing*, *21*(2), 816–827.

Guo, Y., Yao, A., & Chen, Y. (2016). Dynamic network surgery for efficient dnns. In *Advances in neural information processing systems* (pp. 1379–1387).

Gysel, P., Motamedi, M., & Ghiasi, S. (2016). Hardware-oriented approximation of convolutional neural networks, arXiv preprint arXiv:1604.03168.

Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, arXiv preprint arXiv:1510.00149.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision* (pp. 1026–1034).

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770–778).

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, *29*(6), 82–97.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., & Weyand, T. et al., (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications, arXiv preprint arXiv:1704.04861.

Huang, G., Liu, Z., Weinberger, K. Q., & van der Maaten, L. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition, Vol. 1* (p. 3).

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., & Bengio, Y. (2016). Quantized neural networks: Training neural networks with low precision weights and activations, arXiv preprint arXiv:1609.07061.

Ioannou, Y., Robertson, D., Shotton, J., Cipolla, R., & Criminisi, A. (2015). Training cnns with low-rank filters for efficient image classification, arXiv preprint arXiv:1511.06744.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448–456).

Jaderberg, M., Vedaldi, A., & Zisserman, A. (2014). Speeding up convolutional neural networks with low rank expansions, arXiv preprint arXiv:1405.3866.

Kim, Y.-D., Park, E., Yoo, S., Choi, T., Yang, L., & Shin, D. (2015). Compression of deep convolutional neural networks for fast and low power mobile applications, arXiv preprint arXiv:1511.06530.

Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.

Kolda, T. G., & Bader, B. W. (2009). Tensor decompositions and applications. *SIAM Review*, *51*(3), 455–500.

Kossaifi, J., Khanna, A., Lipton, Z. C., Furlanello, T., & Anandkumar, A. (2017). Tensor contraction layers for parsimonious deep nets, arXiv preprint arXiv:1706.00439.

Krizhevsky, A., & Hinton, G. Learning multiple layers of features from tiny images.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097–1105).

Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I., & Lempitsky, V. (2014). Speeding-up convolutional neural networks using fine-tuned cp-decomposition, arXiv preprint arXiv:1412.6553.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Li, Q., & Schonfeld, D. (2014). Multilinear discriminant analysis for higher-order tensor data classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *36*(12), 2524–2537.

Lin, M., Chen, Q., & Yan, S. (2013). Network in network, arXiv preprint arXiv:1312.4400.

Lin, S., Ji, R., Guo, X., & Li, X. et al., 2016. Towards convolutional neural networks compression via global error reconstruction. In *IJCAI* (pp. 1753–1759).

Lin, D., Talathi, S., & Annapureddy, S. (2016). Fixed point quantization of deep convolutional networks. In *International conference on machine learning* (pp. 2849–2858).

Lu, H., Plataniotis, K. N., & Venetsanopoulos, A. N. (2008). MPCA: Multilinear principal component analysis of tensor objects. *IEEE Transactions on Neural Networks*, *19*(1), 18–39.

Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML, Vol. 30.*.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., & Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning, Vol. 2011* (p. 5).

Novikov, A., Podoprikhin, D., Osokin, A., & Vetrov, D. P. (2015). Tensorizing neural networks. In *Advances in neural information processing systems* (pp. 442–450).

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779–788).

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International conference on medical image computing and computer-assisted intervention* (pp. 234–241). Springer.

Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1988). Learning representations by back-propagating errors. *Cognitive Modeling*, *5*(3), 1.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556.

Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2014). Striving for simplicity: The all convolutional net, arXiv preprint arXiv:1412.6806.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting.. *Journal of machine learning research*, *15*(1), 1929–1958.

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI* (pp. 4278–4284).

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., & Anguelov, D. et al., (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1–9).

Tai, C., Xiao, T., Zhang, Y., Wang, X., & Weinan, E. (2015). Convolutional neural networks with low-rank regularization, arXiv preprint arXiv:1511.06067.

Tao, D., Li, X., Wu, X., & Maybank, S. J. (2007). General tensor discriminant analysis and gabor features for gait recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *29*(10).

Thanh, D. T., Kanniainen, J., Gabbouj, M., & Iosifidis, A. (2017). Tensor representation in high-frequency financial data for price change prediction, arXiv preprint arXiv:1709.01268.

Tsantekidis, A., Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M., & Iosifidis, A. (2017a). Forecasting stock prices from the limit order book using convolutional neural networks. In *2017 IEEE 19th conference on business informatics, Vol. 1* (pp. 7–12). IEEE.

Tsantekidis, A., Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M., & Iosifidis, A. (2017b). Using deep learning to detect price change indications in financial markets. In *European signal processing conference*, Kos, Greece.

Wang, M., Liu, B., & Foroosh, H. Design of efficient convolutional layers using single intra-channel convolution, topological subdivisioning and spatial bottleneck structure.

Waris, M. A., Iosifidis, A., & Gabbouj, M. (2017). Cnn-based edge filtering for object proposals. *Neurocomputing*.

Wen, W., Wu, C., Wang, Y., Chen, Y., & Li, H. (2016). Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems* (pp. 2074–2082).

Yan, S., Xu, D., Yang, Q., Zhang, L., Tang, X., & Zhang, H.-J. (2005). Discriminant analysis with tensor representation. In *IEEE computer society conference on computer vision and pattern recognition, 2005, Vol. 1* (pp. 526–532). IEEE.

Zabihi, M., Rad, A. B., Kiranyaz, S., Gabbouj, M., & Katsaggelos, A. K. (2016). Heart sound anomaly and quality detection using ensemble of neural networks without segmentation. In *Computing in cardiology conference (CinC), 2016* (pp. 613–616). IEEE.

Zhang, X., Zhou, X., Lin, M., & Sun, J. (2017). Shufflenet: An extremely efficient convolutional neural network for mobile devices, arXiv preprint arXiv:1707.01083.

Zhou, H., Li, L., & Zhu, H. (2013). Tensor regression with applications in neuroimaging data analysis. *Journal of the American Statistical Association*, *108*(502) 540–552.

Zhou, S.-C., Wang, Y.-Z., Wen, H., He, Q.-Y., & Zou, Y.-H. (2017). Balanced quantization: An effective and efficient approach to quantized neural networks. *Journal of Computer Science and Technology*, *32*(4), 667–682.