T-Net: Parametrizing Fully Convolutional Nets with a Single High-Order Tensor

Jean Kossaifi* Adrian Bulat* Georgios Tzimiropoulos Maja Pantic

Samsung AI Center, Cambridge
United Kingdom

{j.kossaifi, adrian.bulat, georgios.t, maja.pantic}@samsung.com

Abstract

Recent findings indicate that over-parametrization, while crucial for successfully training deep neural networks, also introduces large amounts of redundancy. Tensor methods have the potential to efficiently parametrize over-complete representations by leveraging this redundancy. In this paper, we propose to fully parametrize Convolutional Neural Networks (CNNs) with a single high-order, low-rank tensor. Previous works on network tensorization have focused on parametrizing individual layers (convolutional or fully connected) only, and perform the tensorization layer-by-layer separately. In contrast, we propose to jointly capture the full structure of a neural network by parametrizing it with a single *high-order tensor, the modes of which represent each of the architectural design parameters of the network (e.g. number of convolutional blocks, depth, number of stacks, input features, etc). This parametrization allows to regularize the whole network and drastically reduce the number of parameters. Our model is end-to-end trainable and the low-rank structure imposed on the weight tensor acts as an implicit regularization. We study the case of networks with rich structure, namely Fully Convolutional Networks (FCNs), which we propose to parametrize with a single 8^{th}–order tensor. We show that our approach can achieve superior performance with small compression rates, and attain high compression rates with negligible drop in accuracy for the challenging task of human pose estimation.*

1. Introduction

For a wide range of challenging tasks, including recognition [22, 34, 13], detection [32], semantic segmentation [25, 12] and human pose estimation [26], the state-of-the-art is currently attained with Convolutional Neural Networks (CNNs). There is evidence that a key feature behind the success of these methods is over-parametrization, which helps find good local minima [11, 35]. However, at the same time, over-parametrization leads to a great amount of redundancy, and from a statistical perspective, it might hinder generalization because it excessively increases the number of parameters. Furthermore, models with an excessive number of parameters have increased storage and computation requirements, rendering them problematic for deployment on devices with limited computational resources. This paper focuses on a novel way of leveraging the redundancy in the parameters of CNNs by jointly parametrizing the whole network using tensor methods.

There is a significant amount of recent work on using tensors to reduce the redundancy and improve the efficiency of CNNs, mostly focusing on re-parametrizing individual layers. For example, [36, 17] treat a convolutional layer as a 4D tensor and then compute a decomposition of the 4D tensor into a sum of a small number of low-rank tensors. Similarly, [27] proposes tensorizing the fully-connected layers. The bulk of these methods focus on tensorizing individual layers only, and perform the tensorization layer-by-layer disjointly, usually by applying a tensor decomposition to the pre-trained weights and then fine-tuning to compensate for the performance loss. For example, [36] tensorizes the second convolutional layer of AlexNet [22].

Our paper primarily departs from prior work by using a single high-order tensor to parametrize the whole CNN as opposed to using different tensors to parametrize the individual layers. In particular, we propose to parametrize the network with a single high-order tensor, each dimension of which represents a different architectural design parameter of the network. For the case of Fully Convolutional Networks (FCNs) with an encoder-decoder structure considered herein (see also Fig. 1), each dimension of the $8-$dimensional tensor represents a different architectural design parameter of the network such as the number of (stacked) FCNs used, the depth of each network, the number of input and output features for each convolutional block and the spatial dimensions of each of the convolutional kernels. By modelling the whole FCN with a single tensor, our approach allows for learning correlations between the

*Equal contribution.

different tensor dimensions and hence to fully capture the structure of the network. Moreover, this parametrization implicitly regularizes the whole network and drastically reduces the number of parameters by imposing a low-rank structure on that tensor. Owing to these properties, our framework is much more general and flexible compared to prior work offering increased accuracy and high compression rates. In summary, the **contributions** of this work are:

- We propose using a single high-order tensor for whole network tensorization and applying it for capturing the rich structure of Fully Convolutional Networks. Our end-to-end trainable approach allows for a wide spectrum of network decompositions and compression rates which can be chosen and optimized for a particular application.
- We show that for a large range of compression rates (both high and low), our method preserves high accuracy. Compared to prior work based on tensorizing individual convolutional layers, our method consistently achieves higher accuracy, especially for the case of high compression rates. In addition, we show that, for lower compression rates, our method outperforms the original uncompressed network.
- We illustrate the favorable properties of our method by performing a large number of experiments and ablation studies for the challenging task of human pose estimation. The experiments shed light on several interesting aspects of our method including the effect of varying the rank for each mode of the tensor, as well as the decomposition method used. We further validate our conclusions on a different dense prediction task, namely semantic facial part segmentation.

2. Related Work

In this section, we review related work, both for tensor methods and human pose estimation.

Tensor methods offer a natural extension of traditional algebraic methods to higher orders. For instance, Tucker decomposition can be seen as a generalization of PCA to higher dimensions [18]. Tensor decompositions have wide-reaching applications, including learning a wide range of probabilistic latent-variable models [1]. Tensor methods have been recently applied to deep learning, for instance, to provide a theoretical analysis of deep neural nets [9]. New layers were also proposed, leveraging tensor methods. [19] proposes tensor contraction layers to reduce the dimensionality of activation tensors while preserving their multi-linear structure. Tensor regression layers [20] express outputs through a low-rank multi-linear mapping from a high-order activation tensor to an output tensor of arbitrary order.

A lot of existing work has been dedicated to leveraging tensor decompositions in order to re-parametrizing existing layers, either to speed up computation or to reduce the number of parameters. Separable convolutions, for instance, can be obtained from existing ones by applying CP decomposition to their kernel. The authors in [23] propose such parametrization of individual convolutional layers using CP decomposition with the goal of speeding them up. Specifically, each of the 4D tensors parametrizing the convolutional layers of a pre-trained network are decomposed into a sum of rank–1 tensors using CP decomposition. The resulting factors are used to replace each existing convolution with a series of 4 convolutional layers with smaller kernels. The network is then fine-tuned to restore performance. [17] proposes a similar approach but uses Tucker decomposition instead of CP to decompose the convolutional layers of a pre-trained network, before fine-tuning to restore the performance. Specifically, Tucker decomposition is applied to each convolutional kernel of a pre-trained network, on two of the modes (input and output channel modes). The resulting network is fine-tuned to compensate for the drop in performance induced by the compression.

In [3], the layers of deep convolutional neural networks are also re-parametrized using CP decomposition, optimized using the tensor power method. The network is then iteratively fine-tuned to restore performance. Similarly, [36] proposes to use tensor decomposition to remove redundancy in convolutional layers and express these as the composition of two convolutional layers with less parameters. Each 2D filter is approximated by a sum of rank–1 matrices. Thanks to this restricted setting, a closed-form solution can be readily obtained with SVD. This is done for each convolutional layer with a kernel of size larger than 1. While all these focus of convolutional layers, other types of layers can also be parametrized. For instance, [27] uses the Tensor-Train (TT) format [28] to impose a low-rank tensor structure on the weights of the fully-connected layers. Tensorization of generative adversarial networks [6] and sequence models [45] have been also proposed.

The work of [7] proposes a new residual block, the so-called *collective residual unit* (CRU), which is obtained by applying a generalized block term decomposition to the last two modes of a 4^{th}–order tensor obtained by stacking the convolutional kernels of several residual units. Similarly to existing works, each of the CRUs is parametrized individually. [44] leverages tensor decomposition for multi-task learning to allow for weight sharing between the fully-connected and convolutional layers of two or more deep neural networks.

Overall, to the best of our knowledge, our work is the first to propose an end-to-end trainable architecture, fully parametrized by a *single* high order low-rank tensor.

Other methods for network decomposition. There are also other methods, besides tensor-based ones, for reducing the redundancy and number of parameters in neural networks. A popular approach is quantization which is concerned with quantizing the weights and/or the features of a network [41, 43, 46, 37, 10, 31]. Quantization methods should be considered orthogonal to tensor methods as one could apply them to the output of tensor decompositions, too. Similarly, complementary to our work should be considered methods for improving the efficiency of neural networks using weight pruning [24, 14].

More related to our work are hand-crafted decomposition methods such as MobileNet [15] and Xception [8] which decompose 3×3 convolutions using efficient depth-wise and point-wise convolutions. We compare our methods with MobileNet, the method of choice for improving the efficiency of CNNs, and show that our approach outperforms it by large margin.

Human pose estimation. CNN–based methods have recently produced results of remarkable accuracy for the task of human pose estimation, outperforming traditional methods by large margin [40, 39, 30, 4, 26, 42]. Arguably, one of the most widely used architectures for this task is the stacked HourGlass (HG) network proposed by [26]. An HG is an encoder-decoder network with skip connections between the encoder and the decoder, suitable for making predictions at a pixel level in a fully convolutional manner. [26] uses a stack of 8 of these networks to achieve state-of-the-art performance on the MPII dataset [2]. The architecture is shown in Fig. 1. In this work, we choose tensorizing the HG network primarily because of its rich structure which makes it suitable to model it with a high-order tensor. We note that the aim of this work is not to produce state-of-the-art results for the task of human pose estimation but to show the benefits of modelling a state-of-the-art architecture with a single high-order tensor.

3. Mathematical background

In this section we first introduce some mathematical background regarding the notation and tensor methods used in this paper.

Notation. We denote vectors (1^{st}–order tensors) as $\mathbf{v}$, matrices (2^{nd}–order tensors) as $\mathbf{M}$, and tensors of order 3 or greater as $\mathcal{X}$. We denote element $(i_0, i_1, \cdots, i_N)$ of a tensor as $\mathcal{X}_{i_0,i_1,\cdots,i_N}$ or $\mathcal{X}(i_0, i_1, \cdots, i_N)$. A colon is used to denote all elements of a mode, e.g. the mode–1 fibers of $\mathcal{X}$ are denoted as $\mathcal{X}(:, i_2, i_3, \cdots, i_N)$. Finally, for any $i, j \in \mathbb{N}$, $[i \,..\, j]$ denotes the set of integers $\{i, i+1, \cdots, j-1, j\}$.

Mode–n unfolding of a tensor $\mathcal{X} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_N}$, is a matrix $\mathbf{X}_{[n]} \in \mathbb{R}^{I_n, I_M}$, with $M = \prod_{\substack{k=0, \\ k \neq n}}^{N} I_k$, defined by the mapping from element $(i_0, i_1, \cdots, i_N)$ to (i_n, j), with $j = \sum_{\substack{k=0, \\ k \neq n}}^{N} i_k \times \prod_{\substack{m=k+1, \\ m \neq n}}^{N} I_m$.

Mode-n product. For a tensor $\mathcal{X} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_N}$ and a matrix $\mathbf{M} \in \mathbb{R}^{R \times I_n}$, the n-mode product of a tensor is a tensor of size $(I_0 \times \cdots \times I_{n-1} \times R \times I_{n+1} \times \cdots \times I_N)$ and can be expressed using the unfolding of $\mathcal{X}$ and the classical dot product as $\mathcal{X} \times_n \mathbf{M} = \mathbf{M}\mathcal{X}_{[n]} \in \mathbb{R}^{I_0 \times \cdots \times I_{n-1} \times R \times I_{n+1} \times \cdots \times I_N}$

Tensor diagrams. While 2^{nd}–order tensors can easily be depicted as rectangles and 3^{rd}–order tensors as cubes, it is impractical to represent high-order tensors in such way. We instead use tensor diagrams, which are undirected graphs where the vertices represent tensors. The *degree* of each vertex (i.e. the number of edges originating from this circle) specifies the order of the corresponding tensor. Tensor contraction over two modes is then represented by simply linking together the two edges corresponding to these two modes. Fig. 2 depicts the Tucker decomposition (i.e. contraction of a core tensor with factor matrices along each mode) of an 8^{th}–order tensor with tensor diagrams.

4. T-Net: Fully-tensorized FCN architecture

In this section, we introduce our fully-tensorized method by first introducing the architecture before detailing the structure of the parametrization weights.

4.1. FCN tensorization

In this section, we describe how to tensorize the stacked HourGlass (HG) architecture of [26]. The HG has a number of design parameters namely the number of (stacked) HGs, the depth of each HG, the three signal pathways of each HG (skip, downsample and upsample), the number of convolutional layers in each residual block (i.e. the depth of each block), the number of input and output features of each block and finally, the spatial dimensions of each of the convolutional kernels.

To facilitate the tensorization of the whole network, we used a modified HG architecture in which we replaced all the residual modules with the basic block introduced by [13], maintaining the same number of input and output channels throughout the network. We made the encoder and the decoder symmetric, with 4 residual modules each. Figure 1 illustrates the modified HG architecture. We note that from an accuracy perspective, this modification performs (almost) the same as the original HG proposed in [26].

From the network described above, we derive the high-order tensor for the proposed Tensorized-Network (T-Net)

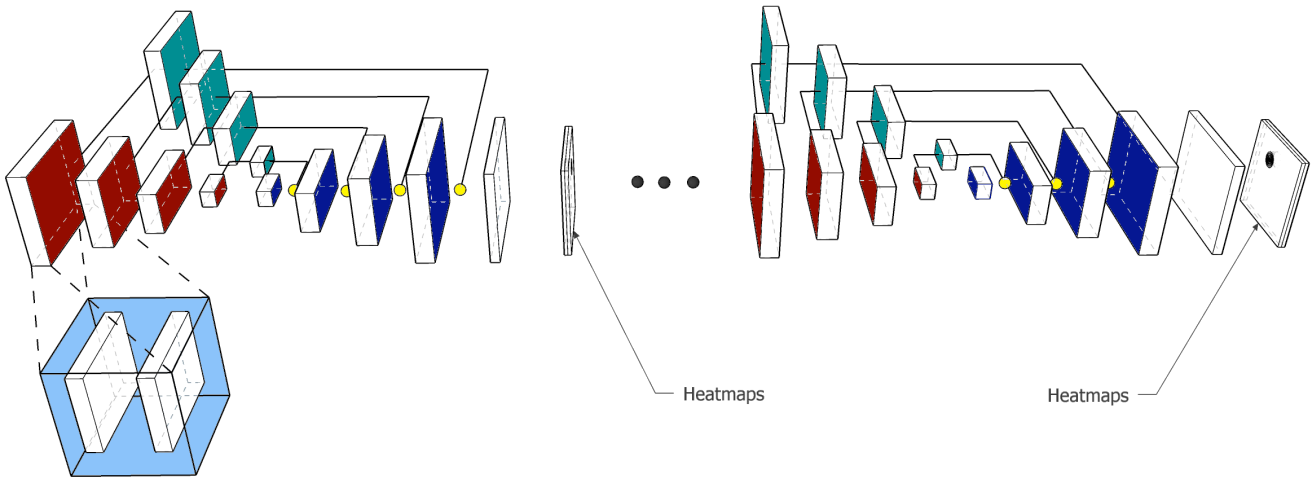

Figure 1: **Overall network architecture.** Each block in the fully convolutional network is a basic-block module [13] (*blue insert*), containing b_{depth} (by default 2) convolutional layers with 3×3 kernels followed by BatchNorm and ReLU. For all experiments, unless explicitly stated otherwise, we used a stack of 4 sub-networks with 3 pathways each: downsampling/encoder (red blocks), upsampling/decoder (dark blue) and skip connection (cyan). Yellow dots are element-wise sums.

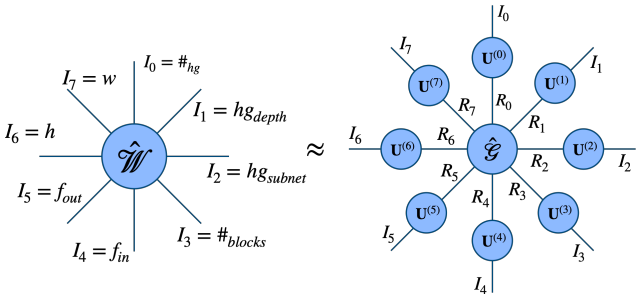

Figure 2: **Tensor diagram** of the Tucker form of the weight tensor $\mathcal{W}$ parametrizing our model.

as follows: all weights of the network are parametrized by a *single* 8th–order tensor $\mathcal{W} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_7}$, the modes of which correspond to the number of HGs ($I_0 = \#hg$), the depth of each HG ($I_1 = hg_{depth}$), the three signal pathways ($I_2 = hg_{subnet}$), the number of convolutional layers per block ($I_3 = b_{depth}$), the number of input features ($I_4 = f_{in}$), the number of output features ($I_5 = f_{out}$), and finally the height ($I_6 = h$) and width ($I_7 = w$) of each of the convolutional kernels.

4.2. T-Net variants

Based on the previous parametrization of the network, we can add various low-rank constraints on the weight tensor, leading to variants of our method.

Tucker T-Net. The Tucker form of our model expresses the constructed 8th–order tensor $\mathcal{W}$ as a rank–$(R_0, \cdots, R_7)$ Tucker tensor, composed of a low rank core $\mathcal{G} \in \mathbb{R}^{R_0 \times R_1 \times \cdots \times R_7}$ along with projection factors $\left(\mathbf{U}^{(0)}, \cdots, \mathbf{U}^{(7)}\right)$, with $\mathbf{U}^{(k)} \in \mathbb{R}^{R_k, I_k}, k \in [0\,..\,7]$. This allows us to write the network's weight tensor in a decomposed form as:

$$\begin{aligned} \mathcal{W} &= \mathcal{G} \times_0 \mathbf{U}^{(0)} \times_1 \mathbf{U}^{(2)} \times \cdots \times_7 \mathbf{U}^{(7)} \\ &= [\![\mathcal{G}; \mathbf{U}^{(0)}, \cdots, \mathbf{U}^{(7)}]\!] \end{aligned} \tag{1}$$

See also Fig. 2 for a tensor diagram of the Tucker form of the weight tensor. Note that the CP decomposition is the special case of the Tucker decomposition, where the core is super-diagonal.

MPS T-Net. The Matrix-Product-State (MPS) form (also known as *tensor-train* [28]) of our model expresses the constructed 8th–order weight tensor $\mathcal{W}$ as a series of third-order tensors (the *cores*) and allows for especially large space-savings. In our case, given $\mathcal{W} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_7}$, we can decompose it into a rank $(R_0, R_1, \cdots R_8)$–MPS as a series of third-order cores $\mathcal{G}_0 \in \mathbb{R}^{R_0, I_0, R_1}, \mathcal{G}_1 \in \mathbb{R}^{R_1, I_1, R_2}, \ldots, \mathcal{G}_7 \in \mathbb{R}^{R_7, I_7, R_8}$. The boundary conditions dictate $R_0 = R_8 = 1$. In terms of individual elements, we can then write, for any $i_0 \in [0\,..\,I_0], i_1 \in [0\,..\,I_1], \cdots, i_7 \in [0\,..\,I_7]$:

$$\mathcal{W}(i_0, i_1, \cdots, i_7) = \underbrace{\mathcal{G}_0[i_0]}_{1 \times R_1} \times \underbrace{\mathcal{G}_1[i_1]}_{R_1 \times R_2} \times \cdots \times \underbrace{\mathcal{G}_7[i_7]}_{R_7 \times 1}$$

4.3. Parameter analysis

This section compares the number of parameters of our model which parameterizes the whole weight tensor with

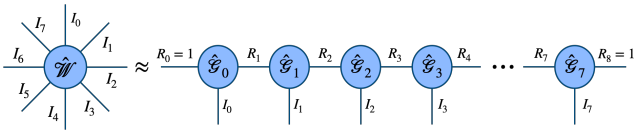

Figure 3: **Tensor diagram of the MPS/TTrain form of the weight tensor $\mathcal{W}$**. Note the *train*–like shape from which the method takes its name, as well as the boundary conditions ($R_0 = R_8 = 1$).

a single high-order tensor with methods based on layer-wise decomposition (e.g. [17, 23]). Considering a Tucker rank–$R_0, R_1, \cdots, R_7$ of the weight tensor parametrizing the whole network, the resulting number of parameters is:

$$N_{T-Net} = \prod_{k=0}^{7} R_k + \sum_{k=0}^{7} R_k \times I_k. \quad (2)$$

Compressing each of the N_{conv} convolutional layer separately [17], with a rank R_4 and R_5 for the number of input and output features, respectively, and writing $N_{\text{conv}} = \prod_{k=0}^{4} I_k$, we obtain the total number of parameters:

$$N_{\text{conv}} \times (R_4 \times R_5 \times I_6 \times I_7 + R_4 \times I_4 + R_5 I_5). \quad (3)$$

In comparison, our model, with the same ranks R_4 and R_5 imposed on the number of features, would only have $N_{\text{conv}} \times (R_4 \times R_5 \times 3 \times 3) + R_4 \times I_4 + R_5 I_5$ parameters. In other words, our model has $\left(\prod_{k=0}^{4} I_k - 1\right)(R_4 \times I_4 + R_5 I_5)$ parameters less than a corresponding layer-wise decomposition.

Speeding up the convolutions. When parametrized using a CP or Tucker decomposition, a convolutional layer can be efficiently replaced by a series of convolutions with smaller kernels [23, 17], thus allowing for large computational speedups. This efficient re-parametrization also applies to our model. To see this, given the weight tensor $\mathcal{W} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_7}$ of our Tucker T-Net, we have

$$\mathcal{W} = \mathcal{G} \times_0 \mathbf{U}^{(0)} \times_1 \mathbf{U}^{(1)} \times \cdots \times_7 \mathbf{U}^{(7)}.$$

For any $i_0, i_1, i_2, i_3 \in (I_0, I_1, I_2, I_3)$, let us denote $\tilde{\mathcal{K}} = \mathcal{W}(i_0, i_1, i_2, i_3, :, :, :, :)$, corresponding to one of the convolutional kernels of the T-Net. By re-arranging the terms, and considering the partially contracted core, we can write:

$$\tilde{\mathcal{K}}(s,t,j,k) = \sum_{r_4=0}^{R_4} \sum_{r_5=0}^{R_5} \mathcal{C}(r_4, r_5, j, k) \mathbf{U}^{(4)}(s, r_4) \mathbf{U}^{(5)}(t, r_5)$$

with $\mathcal{C} = \mathcal{C}_{i_0,i_1,i_2,i_3,:,:,:,:} \in \mathbb{R}^{(I_4, I_5, I_6, I_7)}$ and

$$\mathcal{C} = \left(\mathcal{G} \times_0 \mathbf{U}^{(0)} \times \cdots \times_3 \mathbf{U}^{(3)} \times_6 \mathbf{U}^{(6)} \times_7 \mathbf{U}^{(7)}\right).$$

Baseline	**Tucker** 1.37×	**Tucker** 2.77×	**Tucker** 4.17×
3.79 ms.	4.36 ms.	2.72 ms.	**2.45 ms.**

Table 1: **Timing of baseline conv. vs. naive Tucker.** Speed-up for a 3×3 convolution preserving the number of channels and input tensor of size ($128 \times 64 \times 64$), with a batch-size 64 . We vary the Tucker-rank and report times.

This gives us an effective way of approximating each convolution by three smaller convolutions [17]. While getting the full speedup would require the writing of specialized CUDA kernels, some timings results with a naive implementation using PyTorch are shown in Table 1, for a single convolutional layer with a kernel tensor of size $128 \times 128 \times 3 \times 3$ compressed using Tucker decomposition.

5. Experimental Setup

The bulk of our experiments were conducted for the task of human pose estimation. We also validated some of our conclusions by conducting experiments for a different dense prediction task, namely facial part segmentation.

Human pose estimation. Following [39], we conducted experiments using the standard train and validation splits of one of the most challenging single pose human pose estimation datasets, namely MPII [2]. The dataset contains 22,000 images for training and another 3,000 for validation.

Semantic facial part segmentation. We constructed the facial part segmentation dataset as in [5]: for training, we used the 300W training dataset (more than 3,000 images) and for testing the whole 300W competition test set (600 images) [33].

Implementation details We used a stacked HG architecture with the following architectural parameters: *#hg* = 4, hg_{depth} = 4, hg_{subnet}=3, b_{depth} = 2, f_{in}=128, f_{out}=128, and $h = w = 3$. This resulted in a 8^{th}–order tensor of size $4 \times 4 \times 3 \times 2 \times 128 \times 128 \times 3 \times 3$.

For the uncompressed baseline network, we reduced the number of its parameters by simply decreasing the number of channels in each residual block, varying it from 128 to 64. By doing so, (as opposed to reducing the number of stacks), we maintain all the architectural advantages offered by the stacked HG architecture and ensure a fair comparison with the proposed tensorized network.

Training. All models were trained for 110 epochs using RMSprop [38]. The learning rate was varied from $2.5e-4$ to $1e-6$ using a Multi-Step fixed scheduler. During training, we randomly augmented the data using: rotation ($-25°$

Tucker-rank								Accuracy (PCKh)	Compression ratio
#hg	hg_{depth}	hg_{subnet}	b_{depth}	f_{in}	f_{out}	h	w		
Original								86.99%	1.0x
3	4	3	2	128	128	3	3	87.42%	1.28x
2	4	3	2	128	128	3	3	86.95%	1.82x
1	4	3	2	128	128	3	3	86.05%	3.03x
4	**3**	3	2	128	128	3	3	87.71%	1.28x
4	**2**	3	2	128	128	3	3	87.59%	1.82x
4	**1**	3	2	128	128	3	3	86.89%	3.03x
4	4	**2**	2	128	128	3	3	87.53%	1.43x
4	4	**1**	2	128	128	3	3	86.19%	2.50x
4	4	3	**1**	128	128	3	3	82.59%	1.82x
4	4	3	2	**96**	**96**	3	3	87.43%	1.64x
4	4	3	2	**64**	**64**	3	3	86.13%	3.03x
4	4	3	2	**32**	**32**	3	3	83.10%	6.25x
4	4	3	2	128	128	**2**	**2**	87.30%	1.98x

Table 2: **Human pose estimation task.** Study of the redundancy of each of the modes of the 8^{th}–order weight tensor. We compress one dimension at a time by reducing its corresponding rank in the Tucker tensor. Reported accuracy is in terms of PCKh.

Figure 4: **Qualitative results produced by our method on MPII.**

to 25° for human pose and $-40°$ to 40° for face part segmentation), scale distortion (0.75 to 1.25), horizontal flipping and color jittering.

All experiments were run on a single NVIDIA TITAN V GPU. All networks were implemented using *PyTorch* [29]. *TensorLy* [21] was used for all tensor operations.

Performance measures. For the human pose estimation experiments, we report accuracy in terms of PCKh [2]. For facial part segmentation, we report segmentation accuracy using the mean accuracy and mIOU metrics [25]. Finally, we measure the parameter savings using the compression ratio $= \frac{\text{uncompressed}}{\text{compressed}}$, defined as the total number of parameters of the uncompressed network divided by the number of parameters of the compressed network.

6. Results

This section offers an in-depth analysis of the performance and accuracy of the proposed T-Net. Our main results are that the proposed approach: i) outperforms the layer-wise decomposition of [17] and [23], which are the most closely related works to our method; ii) outperforms

Method	Parameters	Compression ratio	Accuracy
Uncompressed Baseline	full, f_{in}=f_{out}=128	1x	87%
Trimmed Baseline	f_{in}=f_{out}=112	1.3x	86.9%
Trimmed Baseline	f_{in}=f_{out}=92	2x	85.9%
Trimmed Baseline	f_{in}=f_{out}=64	4x	84.5%
Trimmed Baseline	hg_depth=3	1.3x	86.79%
Trimmed Baseline	hg_depth=2	1.8x	86.82%
Trimmed Baseline	hg_depth=1	3.0x	85.30%
MobileNet-[16]	f_{in}=f_{out}=194	3.6x	84.3%
MobileNet-[16]	f_{in}=f_{out}=160	5.4x	82.7%
[17]	rank–(128, 128, 2, 2)	1.4x	84.9%
[17]	rank–(96, 96, 3, 3)	1.3x	86.8%
[17]	rank–(64, 64, 3, 3)	2.3x	86.4%
[17]	rank–(32, 32, 3, 3)	4.7x	85.3%
[17]	rank–(16, 16, 3, 3)	6.9x	83.7%
Tucker T-Net [Ours]	rank–$(4, 3, 3, 2, 110, 110, 3, 3)$	1.7x	**87.5%**
Tucker T-Net [Ours]	rank–$(4, 4, 2, 2, 110, 110, 3, 3)$	1.8x	**87.4%**
Tucker T-Net [Ours]	rank–$(3, 3, 3, 2, 110, 110, 2, 2)$	3.7x	**87.1%**
Tucker T-Net [Ours]	rank–$(3, 2, 3, 2, 96, 96, 3, 3)$	3.4x	**86.7%**
Tucker T-Net [Ours]	rank–$(3, 3, 2, 2, 80, 80, 3, 3)$	4.2x	**86.3%**
Tucker T-Net [Ours]	rank–$(2, 2, 2, 2, 96, 96, 3, 3)$	5.2x	**86.0%**
MPS T-Net [Ours]	rank–$(1, 4, 4, 12, 24, 110, 9, 3, 1)$	7.4x	**85.5%**

Table 3: **Human pose estimation task**. Comparison between T-Net and various baselines and state-of-the-art methods. Accuracy is reported in terms of PCKh. For the tensor decomposition-based methods, we report the rank, and for the others, the number of channels in the convolutional layers.

the uncompressed, original network for low compression rates; iii) achieves consistent compression ratios across arbitrary dimensions and iv) outperforms MobileNet [16] by large margin. Finally, we further validate some of these results for the task of semantic facial part segmentation.

All results reported were obtained by fine-tuning our networks in an end-to-end manner from a pre-trained uncompressed original network. We were able to reach the same level of accuracy when training from scratch, though this required training for more iterations. In contrast, we found that when trained from scratch, the layer-wise method of [17] reaches sub-par performance, as also reported in their paper.

6.1. Redundancy of the weight tensor

In order to better understand the compressibility of each mode of the weight tensor, we first investigate the redundancy of each of the modes of the tensor by compressing *only one* of the modes at a time. Table 2 shows the accuracy (PCKh) as well as the compression ratio obtained by compressing one of the modes, corresponding respectively to the number of HGs (*#hg*), the depth of each HG (hg_{depth}), the three pathways of each HG (hg_{subnet}), the number of convolutional layers per blocks (b_{depth}) and, finally, the number of input features (f_{in}), output features (f_{out}), height (h) and the width (w) of each of the convolutional kernels. The results are shown along with the performance of the original uncompressed network. We observe that by taking advantage of the redundancy at network-level (as opposed to [23, 17] which compress individual layers), the proposed approach is able to effectively compress across arbitrary dimensions for large compression ratios while maintaining similar, or even in some cases higher, accuracy than that of the original uncompressed network.

6.2. Performance of the T-Net

Based on the insights gained from the previous experiment, we selected promising configurations and compressed over multiple dimensions simultaneously. We then compared these configurations with baseline and state-of-the-art methods. The results can be seen in Table 3.

Compression vs. trimming. The obvious comparison is between T-Net and the original baseline network, "compressed" by trimming it, reducing the number of parameters

Method	Parameters	Compression ratio	mIOU	mAcc
Uncompressed baseline	full, f_{in}=f_{out}=128	1x	76.02%	97.31%
T-Net [Ours]	Tucker–$(3, 2, 3, 2, 96, 96, 3, 3)$	3.38x	**76.01%**	**97.29%**
T-Net [Ours]	Tucker–$(2, 2, 2, 2, 64, 64, 3, 3)$	6.94x	**75.57%**	**97.01%**

Table 4: **Facial part segmentation task**. Comparison between T-Net and a network with the same architecture and number of features as the compressed one. Our approach is able to retain a high accuracy even at high compression rates (up to 7x).

Figure 5: **Qualitative results produced by our method on the facial part segmentation task.**

to match the compression ratio achieved by T-Net.

Comparison with efficient architectures. A natural question is whether T-Net performs favourably when compared to architectures designed for efficiency. To answer this, we performed a comparison with MobileNet [16], for which we adjusted the number of channels of the convolutional layers in order to vary the number of parameters and obtain comparable compression ratios.

Comparison with the state-of-the-art. We also compared with the layer-wise decomposition method of [17].

We firstly observe that by just reducing the number of channels in the original network, a significant drop in performance can be noticed. Secondly, our method consistently outperforms [17] across the whole spectrum of compression ratios. This can be seen by comparing the accuracy provided for any compression ratio for [17] with the accuracy of the closest but higher compression ratio for our method (for example, compare 2.33x for [17] with 3.67x for our method). Our method always achieves higher accuracy even though the compression ratio is also higher. In addition, unlike to [17] which does not seem to work well when the size of the convolutional kernel is compressed from 3×3 to 2×2, our method is able to compress that dimension too while maintaining similar level of accuracy. Finally, our method outperforms MobileNet [16] by a large margin.

In the same table, we also report the performance of a variant of our method, using an MPS decomposition on the weights rather than a Tucker one. This result shows that our method works effectively with other decomposition methods as well. Nevertheless, we focused mainly on Tucker as it is the most flexible compression method, allowing us to control the rank of each mode of the weight tensor.

Results on face segmentation. Finally, we selected two of our best performing models and retrained them for the task of semantic facial part segmentation. Our method offers significant compression ratios (up to 7x) with virtually no loss in accuracy (see Table 4). These results further confirm that our method is task-independent.

7. Conclusions

We proposed an end-to-end trainable method to jointly capture the full structure of a fully-convolutional neural network, by parametrizing it with a single, high-order low-rank tensor. The modes of this tensor represent each of the architectural design parameters of the network (e.g. number of convolutional blocks, depth, number of stacks, input features, etc). This parametrization allows for a joint regularization of the whole network. The number of parameters can be drastically reduced by imposing a low-rank structure on the parameter tensor. We show that our approach can achieve superior performance with low compression rates, and attain high compression rates with negligible drop in accuracy, on both the challenging task of human pose estimation and semantic face segmentation.

References

[1] Animashree Anandkumar, Rong Ge, Daniel Hsu, Sham M. Kakade, and Matus Telgarsky. Tensor decompositions for learning latent variable models. *JMLR*, 15(1):2773–2832, jan 2014.

[2] Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2d human pose estimation: New benchmark and state of the art analysis. In *CVPR*, 2014.

[3] Marcella Astrid and Seung-Ik Lee. Cp-decomposition with tensor power method for convolutional neural networks compression. *CoRR*, abs/1701.07148, 2017.

[4] Adrian Bulat and Georgios Tzimiropoulos. Human pose estimation via convolutional part heatmap regression. In *ECCV*, 2016.

[5] Adrian Bulat and Georgios Tzimiropoulos. Binarized convolutional landmark localizers for human pose estimation and face alignment with limited resources. In *ICCV*, 2017.

[6] Xingwei Cao and Qibin Zhao. Tensorizing generative adversarial nets. *CoRR*, abs/1710.10772, 2017.

[7] Yunpeng Chen, Xiaojie Jin, Bingyi Kang, Jiashi Feng, and Shuicheng Yan. Sharing residual units through collective tensor factorization to improve deep neural networks. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 635–641. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[8] François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint*, pages 1610–02357, 2017.

[9] Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. *CoRR*, abs/1509.05009, 2015.

[10] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NIPS*, 2015.

[11] Simon S Du and Jason D Lee. On the power of over-parametrization in neural networks with quadratic activation. In *ICML*, 2018.

[12] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[14] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017.

[15] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[16] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[17] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *CoRR*, 05 2016.

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

[19] Jean Kossaifi, Aran Khanna, Zachary Lipton, Tommaso Furlanello, and Anima Anandkumar. Tensor contraction layers for parsimonious deep nets. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*, pages 1940–1946. IEEE, 2017.

[20] Jean Kossaifi, Zachary C. Lipton, Aran Khanna, Tommaso Furlanello, and Anima Anandkumar. Tensor regression networks. *CoRR*, abs/1707.08308, 2018.

[21] Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. Tensorly: Tensor learning in python. *Journal of Machine Learning Research*, 20(26):1–6, 2019.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[23] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan V. Oseledets, and Victor S. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *CoRR*, abs/1412.6553, 2014.

[24] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *ICLR*, 2017.

[25] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[26] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *ECCV*, 2016.

[27] Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry Vetrov. Tensorizing neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, NIPS'15, pages 442–450, 2015.

[28] I. V. Oseledets. Tensor-train decomposition. *SIAM J. Sci. Comput.*, 33(5):2295–2317, Sept. 2011.

[29] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[30] Tomas Pfister, James Charles, and Andrew Zisserman. Flowing convnets for human pose estimation in videos. In *ICCV*, 2015.

[31] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

[32] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[33] Christos Sagonas, Georgios Tzimiropoulos, Stefanos Zafeiriou, and Maja Pantic. 300 faces in-the-wild challenge: The first facial landmark localization challenge. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 397–403, 2013.

[34] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv*, 2014.

[35] Mahdi Soltanolkotabi, Adel Javanmard, and Jason D Lee. Theoretical insights into the optimization landscape of over-parameterized shallow neural networks. *IEEE Transactions on Information Theory*, 2018.

[36] Cheng Tai, Tong Xiao, Xiaogang Wang, and Weinan E. Convolutional neural networks with low-rank regularization. *CoRR*, abs/1511.06067, 2015.

[37] Zhiqiang Tang, Xi Peng, Shijie Geng, Lingfei Wu, Shaoting Zhang, and Dimitris Metaxas. Quantized densely connected u-nets for efficient landmark localization. In *ECCV*, 2018.

[38] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2), 2012.

[39] Jonathan J Tompson, Arjun Jain, Yann LeCun, and Christoph Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. In *NIPS*, 2014.

[40] Alexander Toshev and Christian Szegedy. Deeppose: Human pose estimation via deep neural networks. In *CVPR*, 2014.

[41] Frederick Tung and Greg Mori. Clip-q: Deep network compression learning by in-parallel pruning-quantization. In *CVPR*, 2018.

[42] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines. In *CVPR*, 2016.

[43] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *CVPR*, 2016.

[44] Yongxin Yang and Timothy M. Hospedales. Deep multi-task representation learning: A tensor factorisation approach. *CoRR*, abs/1605.06391, 2016.

[45] Rose Yu, Stephan Zheng, Anima Anandkumar, and Yisong Yue. Long-term forecasting using tensor-train rnns. *CoRR*, abs/1711.00073, 2017.

[46] Aojun Zhou, Anbang Yao, Kuan Wang, and Yurong Chen. Explicit loss-error-aware quantization for low-bit deep neural networks. In *CVPR*, 2018.