PARAMETRIZING FULLY CONVOLUTIONAL NETS WITH A SINGLE HIGH-ORDER TENSOR

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent findings indicate that over-parametrization, while crucial to the success of deep learning, also introduces large amounts of redundancy. Tensor methods have the potential to parametrize over-complete representations in a compact manner by leveraging this redundancy. In this paper, we propose fully parametrizing Convolutional Neural Networks (CNNs) with a single, low-rank tensor. Previous works on network tensorization haved focused on parametrizing individual layers (convolutional or fully connected) only, and perform the tensorization layer-by-layer disjointly. In contrast, we propose to jointly capture the full structure of a CNN by parametrizing it with a *single*, high-order tensor, the modes of which represent each of the architectural design parameters of the CNN (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 by imposing a low-rank structure on that tensor. Further, our network is end-to-end trainable from scratch, which has been shown to be challenging in prior work. We study the case of networks with rich structure, namely Fully Convolutional CNNs, which we propose to parametrize them with a single $8-$dimensional 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

State-of-the-art performance is currently attained with Convolutional Neural Networks for a wide range of challenging tasks including recognition (Krizhevsky et al., 2012; Simonyan & Zisserman, 2014; He et al., 2016), detection (Ren et al., 2015), semantic segmentation (Long et al., 2015; He et al., 2017) and human pose estimation (Newell et al., 2016) outperforming prior works by a large margin. There is evidence that a key feature behind the success of these methods is over-parameterization which helps find good local minima (Du & Lee, 2018; Soltanolkotabi et al., 2018). However, at the same time, over-parameterization leads to a great amount of redundancy, and from a statistical perspective, it might hinder generalization because it excessively increases the number of network parameters. Furthermore, models with excessive number of parameters have increased storage and computation requirements rendering them problematic for deployment on devices with limited resources. This paper focuses on a novel way of leveraging the redundancy in the parameters of CNNs by tensorizating the whole network.

There is a significant amount of recent work on using tensors to reduce the redundancy and improve the efficiency of CNNs; mostly focusing on reparametrizing individual layers. For example, Tai et al. (2015); Kim et al. (2016) treat a convolutional layer as a 4D tensor and then computes a low-rank decomposition of the 4D convolution kernel tensor into a sum of a small number of low rank tensors. Similarly, Novikov et al. (2015) proposed tensorizing the fully-connected layers. The bulk of these methods focus on tensorizing individual layers only, and perform the tensorization layer-by-layer disjointly, usally by applying tensor decomposition to the pretrained weights and fine-tuning to compensate for performance loss. For example Tai et al. (2015) tensorize the second convolutional layer of AlexNet Krizhevsky et al. (2012).

Our paper primarily differs from prior work by the use of a single 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 CNN. For the case of Fully Convolutional Networks (FCNs) with an encoder-decoder structure considered herein (see 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 CNN with a single tensor, our approach allows to learn correlations between the different tensor dimensions and hence fully capturing the structure of the network. Moreover, this parametrization allows to regularize the whole network and drastically reduce 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 are the first to propose using a single high-order tensor for whole network tensorization, and applying it for capturing the rich structure of Fully Convolutional Networks. Our approach allows for a wide spectrum of network decompositions and compression rates which can be chosen and optimized for a particular application. Further, our network can be trained end-to-end – not only by fine-tuning, by also from scratch, which has been shown to be challenging in prior work.
- We show that, for a wide spectrum 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 computer vision task of human pose estimation. The experiments sheds light on several interesting properties of our method including the effect of varying the rank for each mode of the tensor, as well as the decomposition method used. Further, we validate some of our conclusions by conducting experiments for a different dense prediction task, namely facial part segmentation.

1.1 Related Work

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 (Kolda & Bader, 2009). Tensor decompositions have large reaching applications, including for learning a wide range of probabilistic latent-variable models (Anandkumar et al., 2014). Tensor methods have recently been applied to deep learning, for instance to provide a theoretical analysis of deep neural nets (Cohen et al., 2015). New layers were also proposed, leveraging tensor methods. Kossaifi et al. (2017) propose tensor contraction layers to reduce the dimensionality of activation tensors while preserving their multilinear structure. Tensor regression layers (Kossaifi et al., 2018a) express outputs through a low-rank multilinear mapping from a high-order activation tensor to an output tensor of arbitrary order.

A lot of the 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 Lebedev et al. (2014) 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 pretrained network are decomposed into a sum of rank–1 tensors using CP decomposition. The resulting factors are used to replace each existing convolutions with a series of 4 convolutional layers with smaller kernels. The network is then fine-tuned to restore the performance. Kim et al. (2016) propose a similar approach but use Tucker decomposition instead of CP to decompose the convolutional layers of a pretrained network, before fine-tuning to restore the performance. Specifically, Tucker decomposition is applied to each convolutional kernel of a pretrained 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. Astrid & Lee (2017) also reparametrize the layers of deep convolutional neural networks using CP decomposition, but optimize this using the tensor power method. The network is then iteratively fine-tuned to restore performance. Similarly, Tai et al. (2015) propose 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 1×1. Layers other than convolutional can be also parametrized. For instance, Novikov et al. (2015) use the Tensor-Train (TT) format to impose low-rank tensor structure on the weights of the fully-connected layers. Tensorization of generative adversarial networks (Cao & Zhao, 2017) and sequence models (Yu et al., 2017) have also been proposed.

Tensor decomposition has also been used for weight sharing between layers and multi-task learning. Chen et al. (2017) proposes the so-called *collective residual units* (CRU), which are obtained by stacking the convolutional kernels of several residual units sharing the same size along the last mode (corresponding to the number of output channels) to obtain a 4th order tensor f size (width $(= 3)$, height $(= 3)$, input channels, $L\times$ output channels). Generalized block term decomposition (shortly put, a sum of Tucker decompositions), is used to decompose the last two modes of the resulting tensors along these dimensions. These CRUs are stacked to form the final network. The sharing is done for 6 residual units only, and each of the CRUs is parametrized individually. Yang & Hospedales (2016) leverage 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.

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

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 (Toshev & Szegedy, 2014; Tompson et al., 2014; Pfister et al., 2015; Bulat & Tzimiropoulos, 2016; Newell et al., 2016; Wei et al., 2016). Arguably, one of the most widely used architectures for this task is the stacked Hourglass (HG) network proposed by Newell et al. (2016). 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. Newell et al. (2016) use a stack of eight of these networks to achieve state-of-the-art performance on the MPII dataset (Andriluka et al., 2014). 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.

2 Methodology

In this section we first introduce some mathematical background in subsection 2.1, before introducing our method in subsection 2.2.

2.1 Mathematical background

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

Mode-n unfolding of a tensor $\tilde{\mathcal{X}} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_N}$, 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 $j = \sum_{\substack{k=0,\\ k\neq n}}^{N} i_k \times \prod_{\substack{m=k+1,\\ m\neq n}}^{N} I_m$.

Tensor vectorization of a tensor $\tilde{\mathcal{X}} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_N}$, is a vector $\text{vec}(\tilde{\mathcal{X}})$ of size $(I_0 \times \cdots \times I_N)$ defined by the mapping from element $(i_0, i_1, \cdots, i_N)$ of $\tilde{\mathcal{X}}$ to element j of $\text{vec}(\tilde{\mathcal{X}})$, with $j = \sum_{k=0}^{N} i_k \times \prod_{m=k+1}^{N} I_m$.

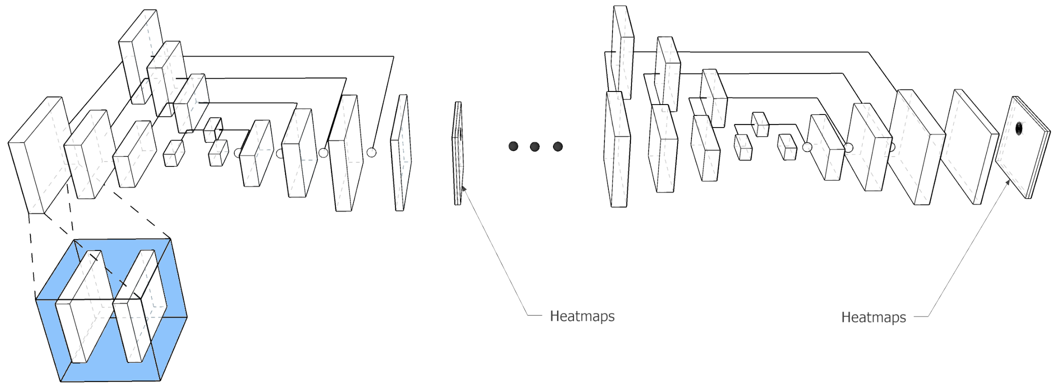

Figure 1: **Overall network architecture.** Each block in the fully convolutional network is a basic-block module (*blue insert*), as introduced by He et al. (2016), containing 2 convolutional layers with 3×3 kernels followed by BatchNorm and ReLU. For all of our experiments, unless explicitly stated otherwise, we used a stack of 4 encoder-decoder sub-networks.

Mode-n product: for a tensor $\tilde{\mathcal{X}} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_N}$ 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 $\tilde{\mathcal{X}}$ and the classical dot product as $\tilde{\mathcal{X}} \times_n \mathbf{M} = \mathbf{M}\tilde{\mathcal{X}}_{[n]} \in \mathbb{R}^{I_0 \times \cdots \times I_{n-1} \times R \times I_{n+1} \times \cdots \times I_N}$

Tensor diagrams: given the high order of the tensor manipulated, it is impractical to represent them graphically. We instead use tensor diagrams, where tensors are represented by circles (vertices), and their order by edges from originating from this circle. Tensor contraction over two modes is then represented by simply linking together the two edges corresponding to these two modes. Figure 2 shows the Tucker decomposition of the 8thorder weight tensor with tensor diagrams.

2.2 Fully-tensorized architecture

In this section, we describe how to tensorize the stacked hourglass architecture of Newell et al. (2016). The hourglass 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 blocks, the number of input and output features for each convolutional block and finally the spatial dimensions of each of the convolutional kernels.

To enable 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 He et al. (2016), 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.

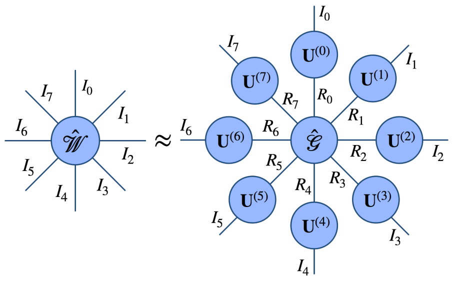

Figure 2: **Tensor diagram** of the Tucker form of the weight tensor $\tilde{\mathcal{W}}$.

From the network described above, we derive the proposed Tensorized-Network (T-Net) as follows: all weights are parametrized by a *single* 8th order tensor $\tilde{\mathcal{W}} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_7}$, 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 blocks ($I_3 = \#blocks$), 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.

Tucker form: Given the weight tensor $\tilde{\mathcal{W}} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_7}$, we can express it as a rank–$(R_0, \cdots, R_7)$ Tucker tensor, composed of a low rank core $\tilde{\mathcal{G}} \in \mathbb{R}^{R_0 \times R_1 \times \cdots \times R_7}$ along with projection factors $\left(\mathbf{U}^{(0)}, \cdots, \mathbf{U}^{(7)}\right)$, with $\mathbf{U}^{(k)} \in \mathbb{R}^{R_k, I_k}, k \in (0, \cdots, 7)$. This allows us to write the

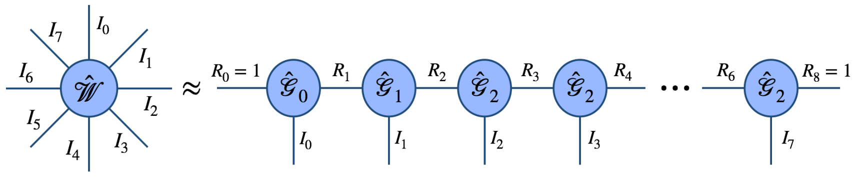

Figure 3: **Tensor diagram of the MPS/TTrain form of the weight tensor $\tilde{\mathcal{W}}$.** Note the *train*–like shape from which the method takes its name, as well as the boundary conditions ($R_0 = R_7 = 1$).

weight tensor in a decomposed form as: $\tilde{\mathcal{W}} = \tilde{\mathcal{G}} \times_0 \mathbf{U}^{(0)} \times_1 \mathbf{U}^{(2)} \times \cdots \times_7 \mathbf{U}^{(7)}$ See also Fig. 2 for a tensor diagram of the tucker form of the weight tensor.

In this work, we either initialize both the factors and the core randomly, when training from scratch, or apply Tucker decomposition to the full weight tensor for fine-tuning.

Matrix-Product-State (MPS) form also known as *tensor-train* (Oseledets, 2011), the Matrix-Product-State expresses a tensor as a series of third order tensors (the *cores*) and allows for especially large space-savings. In our case, given the weight tensor $\tilde{\mathcal{W}} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_7}$, we can decompose it into a rank $R_0, R_1, \cdots R_8$–MPS as a series of third order cores $\tilde{\mathcal{G}}_0 \in \mathbb{R}^{R_0, I_0, R_1}$, $\tilde{\mathcal{G}}_0 \in \mathbb{R}^{R_1, I_1, R_2}, \cdots, \tilde{\mathcal{G}}_0 \in \mathbb{R}^{R_7, I_7, R_8}$. The boundary conditions dictate $R_0 = R_8 = 1$. In term of individual elements, we can write, for $i_0 \in [0 \mathrel{..} I_0], i_1 \in [0 \mathrel{..} I_1], \cdots, i_7 \in [0 \mathrel{..} I_7]$:

$$\tilde{\mathcal{W}}(i_0, i_1, \cdots, i_7) = \underbrace{\tilde{\mathcal{G}}_0[i_0]}_{1 \times R_0} \times \underbrace{\tilde{\mathcal{G}}_1[i_1]}_{R_0 \times R_1} \times \cdots \times \underbrace{\tilde{\mathcal{G}}_7[i_7]}_{R_6 \times 1}$$

Parameter analysis: Considering a Tucker rank–$R_0, R_1, \cdots, R_7$ of the weight tensor parametrizing the whole network, the corresponding number of parameters is:

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

Compressing each of the N_{conv} convolutional layer separately (Kim et al., 2016), with a rank R_4 and R_5 for the number of input and output features, respectively, we obtain a number of parameters of:

$N_{\text{conv}} \times (R_4 \times R_5 \times I_6 \times I_7 + R_4 \times I_4 + R_5 I_5)$, with $N_{\text{conv}} = \prod_{k=0}^{4} I_k$.

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: convolutions expressed in CP or Tucker format, can be replaced by a series of convolutions with smaller kernels (Lebedev et al., 2014; Kim et al., 2016), thus allowing for large computational speedups. This also applies to our model as, given a weight tensor $\tilde{\mathcal{W}} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_7}$, we can write, using the Tucker form, $\tilde{\mathcal{W}} = \tilde{\mathcal{G}} \times_0 \mathbf{U}^{(0)} \times_1 \mathbf{U}^{(2)} \times \cdots \times_7 \mathbf{U}^{(7)}$. Each element $\tilde{\mathcal{W}}(i_0, i_1, i_2, i_3, :, :, :)$ corresponds to one of the convolutional kernels. Let's denote $\tilde{\mathcal{K}} = \tilde{\mathcal{W}}(i_0, i_1, i_2, i_3, :, :, :)$ one such kernel. By leaving the last two dimensions untouched ($\mathbf{U}^{(6)} = \mathbf{U}^{(7)} = \mathbf{Id}$) and re-arranging the terms, we can write: $\tilde{\mathcal{K}}(s, t, j, k) = \sum_{r_4=0}^{R_4} \sum_{r_5=0}^{R_5} \tilde{\mathcal{C}}(r_4, r_5, j, k) \mathbf{U}^{(4)}(s, r_3) \mathbf{U}^{(5)}(s, r_4)$ with $\tilde{\mathcal{C}} = \left(\tilde{\mathcal{G}} \times_0 \mathbf{U}^{(0)} \times_1 \mathbf{U}^{(2)} \times \cdots \times_3\right)(i_0, i_1, i_2, i_3, :, :, :, :) \in \mathbb{R}^{(I_4, I_5, I_6, I_7)}$. This gives us an effective way of approximating each convolution by three, smaller convolutions (Kim et al., 2016).

3 Experimental Setup

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

Human Pose Estimation. Following Tompson et al. (2014), throughout this work, we conduct experiments using the standard train and validation split of one of the most challenging single pose

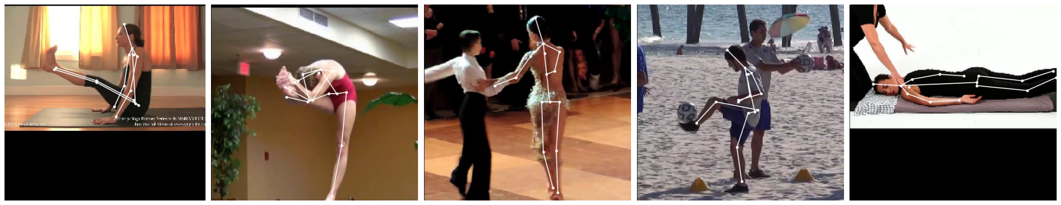

Figure 4: Qualitative results produced by our method on MPII.

human pose estimation datasets, namely MPII (Andriluka et al., 2014). 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 similarly to (Bulat & Tzimiropoulos, 2017): for training, we used the 300W training dataset (more than 3,000 images) and for testing the whole 300W competition test set (600 images) Sagonas et al. (2013).

Implementation details We used a stacked HG architecture with the following architectural parameters: $\#hg$ = 4, hg_{depth} = 4, hg_{subnet}=3, $\#blocks$ = 2, f_{in}=128, f_{out}=128, and h = w = 3. This resulted in a 8th-order tensor of size $4 \times 4 \times 3 \times 2 \times 128 \times 128 \times 3 \times 3$. The network is composed of a total of $15,830,976$ parameters ($1,675,200$ for the base convolutions and $14,555,776$ for the weight tensor parametrizing the convolutional blocks).

For the uncompressed baseline network, we reduce the number of its parameters by simple decreasing the number of channels in each residual block, varying 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 (Tieleman & Hinton, 2012). The learning rate was varied from from $2.5e-4$ to $1e-6$ using a Multi-Step fixed scheduler. During training, we randomly augment the data using: rotation($-25°$ to $25°$ for human pose and $-40°$ to $40°$ for face parsing), 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* (Paszke et al., 2017), and *TensorLy* (Kossaifi et al., 2018b) was used for all tensor operations.

Metrics For the human pose estimation experiments, we report accuracy using the PCKh (Andriluka et al., 2014). For facial part segmentation, we report segmentation accuracy using the mean accuracy and mIO metrics (Long et al., 2015). Finally, we measure the compression of the number of parameters 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.

4 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 Kim et al. (2016), which is the most closely related work to our method; ii) Outperforms the uncompressed, original network for low compression rates; iii) Offers consistent compression ratios across arbitrary dimensions. Finally, we further validate some of these results for the task of semantic facial part segmentation.

All results reported are 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 Kim et al. (2016) reaches sub-par performance, as also reported in their paper.

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 1 shows the accuracy *(PCKh)* as well as the compression ratio obtained by compressing

Tucker-rank								Accuracy (PCKh)	Compression ratio
#hg	**hg_{depth}**	**$\text{hg}_{\text{subnet}}$**	**#blocks**	**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 1: **Human pose estimation task**. Study of the redundancy of each of the modes of the parameter tensor. We compress one dimension at a time by reducing its corresponding rank in the Tucker-tensor. Reported accuracy is in terms of PCKh.

Method	Parameters	Compression ratio	Accuracy
Original	full, f_{in}=f_{out}=128	1x	86.99%
Original	f_{in}=f_{out}=112	1.32x	86.87%
Original	f_{in}=f_{out}=92	1.97x	85.92%
Original	f_{in}=f_{out}=64	3.99x	84.54%
(Kim et al., 2016)	Tucker–(128, 128, 2, 2)	1.42x	84.86%
(Kim et al., 2016)	Tucker–(96, 96, 3, 3)	1.32x	86.85%
(Kim et al., 2016)	Tucker–(64, 64, 3, 3)	2.33x	86.39%
(Kim et al., 2016)	Tucker–(32, 32, 3, 3)	4.73x	85.33%
(Kim et al., 2016)	Tucker–(16, 16, 3, 3)	6.91x	83.74%
T-Net [Ours]	Tucker–$(4, 3, 3, 2, 110, 110, 3, 3)$	1.66x	**87.45%**
T-Net [Ours]	Tucker–$(4, 4, 2, 2, 110, 110, 3, 3)$	1.82x	**87.36%**
T-Net [Ours]	Tucker–$(3, 3, 3, 2, 110, 110, 2, 2)$	3.67x	**87.09%**
T-Net [Ours]	Tucker–$(3, 2, 3, 2, 96, 96, 3, 3)$	3.38x	**86.73%**
T-Net [Ours]	Tucker–$(3, 3, 2, 2, 80, 80, 3, 3)$	4.20x	**86.27%**
T-Net [Ours]	Tucker–$(2, 2, 2, 2, 96, 96, 3, 3)$	5.23x	**86.00%**
T-Net [Ours]	MPS–$(1, 4, 4, 12, 24, 110, 9, 3, 1)$	7.39x	**85.52%**

Table 2: **Human pose estimation task**. Comparison between our T-Net and (a) the original network "compressed" by reducing the number of features and (b) the layer-wise decomposition method of Kim et al. (2016). Reported accuracy is in terms of PCKh.

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 blocks (*#blocks*) 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. These results are shown along with the performance of the original uncompressed network. We observe that by taking advantage of the redundancy at the network-level (as opposed to (Lebedev et al., 2014; Kim et al., 2016) which compress individual layers), the proposed approach is able to efficiently 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.

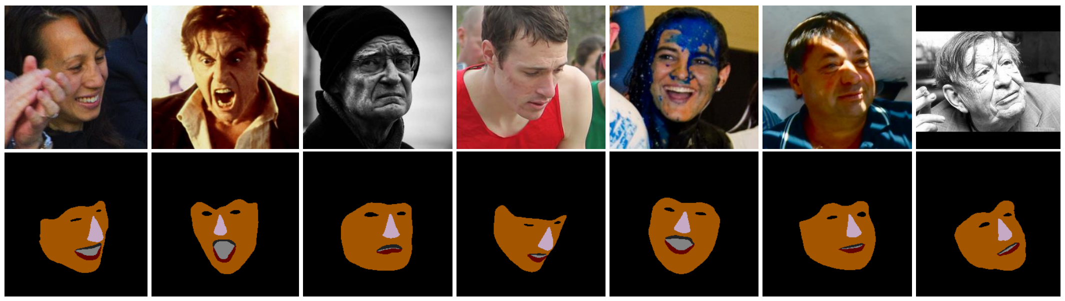
Figure 5: Qualitative results produced by our method on the facial part segmentation task.

Based on the insights gained from the previous experiment, we select promising configurations and compress over multiple dimensions simultaneously. The results can be seen in Table 2 along with (a) the performance of the original network "compressed" by just reducing the number of feature dimensions and (b) the layer-wise decomposition method of Kim et al. (2016). We firstly observe that by just reducing the number of feature dimensions in the original network, a significant drop in performance can be noticed. Secondly, our method consistently outperforms (Kim et al., 2016) across the whole spectrum of compression ratios. This can be seen by comparing the accuracy provided for any compression ratio for (Kim et al., 2016) with the accuracy of the closest but higher compression ratio for our method (for example, compare 2.33x for (Kim et al., 2016) with 3.67x for our method). It can be seen that our method always achieves higher accuracy even the compression ratio is also higher. In addition, contrary to (Kim et al., 2016) 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 while maintaining similar level of performance. 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. However 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.

Finally, we select two of our best performing models and retrain them on semantic facial part segmentation. Our method offers significant compression ratios (up to 6x) with virtually no loss in accuracy (see Table 3). These results further confirm that our method is task-independent.

Method	Parameters	Compression ratio	mIO	mAcc
Uncompressed	full, f_{in}=f_{out}=128	1x	76.02%	97.31%
Tucker	rank–$(3, 2, 3, 2, 96, 96, 3, 3)$	3.38x	76.01%	97.29%
Tucker	rank–$(2, 2, 2, 2, 64, 64, 3, 3)$	6.94x	75.57%	97.01%

Table 3: **Segmentation task**. Comparison between our T-Net and a network with the same architecture and the same number of features as the compressed one. Our approach is capable of retaining a high accuracy even at high compression rates (up to 7x).

5 Conclusions

We proposed a novel method to jointly capture the full structure of a new Fully Convolutional Network by parametrizing it with a *single*, high-order 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 to jointly regularize the whole network and drastically reduce the number of parameters by imposing a low-rank structure on that tensor. Further, our network can be trained in an end-to-end manner not only during fine-tuning but also from scratch which has been shown to be challenging in prior work. We showed that our approach can achieve superior performance with small compression rates, and attain high compression rates with negligible drop in accuracy, on both the challenging task of human pose estimation and face segmentation.

REFERENCES

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.

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

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

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

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

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

Yunpeng Chen, Xiaojie Jin, Bingyi Kang, Jiashi Feng, and Shuicheng Yan. Sharing residual units through collective tensor factorization in deep neural networks. 2017.

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

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

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

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

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.

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

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*, pp. 1940–1946. IEEE, 2017.

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

Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. Tensorly: Tensor learning in python. *CoRR*, abs/1610.09555, 2018b.

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

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.

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

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

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, pp. 442–450, 2015.

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

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.

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

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

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*, pp. 397–403, 2013.

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

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.

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

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.

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.

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

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

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

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