Incremental Multi-domain Learning with Network Latent Tensor Factorization

Adrian Bulat[*,1] **Jean Kossaifi**[*,1,2] **Georgios Tzimiropoulos,**[1] **Maja Pantic**[1,2]
[1] Samsung AI Center Cambridge [2] Imperial College London
adrian@adrianbulat.com, jean.kossaifi@gmail.com, georgios.t@samsung.com, maja.pantic@samsung.com

Abstract

The prominence of deep learning, large amount of annotated data and increasingly powerful hardware made it possible to reach remarkable performance for supervised classification tasks, in many cases saturating the training sets. However the resulting models are specialized to a single very specific task and domain. Adapting the learned classification to new domains is a hard problem due to at least three reasons: (1) the new domains and the tasks might be drastically different; (2) there might be very limited amount of annotated data on the new domain and (3) full training of a new model for each new task is prohibitive in terms of computation and memory, due to the sheer number of parameters of deep CNNs. In this paper, we present a method to learn new-domains and tasks incrementally, building on prior knowledge from already learned tasks and without catastrophic forgetting. We do so by jointly parametrizing weights across layers using low-rank Tucker structure. The core is task agnostic while a set of task specific factors are learnt on each new domain. We show that leveraging tensor structure enables better performance than simply using matrix operations. Joint tensor modelling also naturally leverages correlations across different layers. Compared with previous methods which have focused on adapting each layer separately, our approach results in more compact representations for each new task/domain. We apply the proposed method to the 10 datasets of the Visual Decathlon Challenge and show that our method offers on average about $7.5\times$ reduction in number of parameters and competitive performance in terms of both classification accuracy and Decathlon score.

1 Introduction

It is now commonly accepted that supervised learning with deep neural networks can provide satisfactory solutions for a wide range of problems as long as i) the aim is to focus on a single task only, and ii) there is a sufficient availability of labelled training data and computational resources. This is the setting under which Convolutional Neural Networks (CNNs) have been employed in order to provide state-of-the-art solutions for a wide range of Computer Vision problems such as recognition (Krizhevsky, Sutskever, and Hinton 2012; Simonyan and Zisserman 2014; He et al. 2016), detection (Ren et al. 2015) and semantic segmentation (Long, Shelhamer, and Darrell 2015; He et al. 2017) to name a few.

[*]Denotes equal contribution

However, visual perception is not just concerned with being able to learn a single task at a time, assuming an abundance of labelled data, memory and computing capacity. A more desirable property is to be able to learn a set of tasks, possibly over multiple different domains, under limited memory and finite computing power. This setting is a very general one and many instances of it have been studied in Computer Vision and Machine Learning under various names. The main difference comes from whether we vary the *task* to be performed (classification or regression), or the *domain*, which broadly speaking refers to the distribution of the data or the labels for the considered task.

Herein, we are mostly concerned with the problem of multi-domain incremental learning. A key aspect of this setting is that the new task should be learned without harming the classification accuracy and representational power of the original model. This is called learning without catastrophic forgetting (French 1999; Li et al. 2017). Another important aspect is to keep newly introduced memory requirements low: a newly learned model should use as much as possible existing knowledge learned from already learned tasks, i.e. from a practical perspective, it should re-use or adapt the weights of an already trained (on a different task) network.

The aforementioned setting has only recently attracted the attention of the neural network community. Notably, Rebuffi, Bilen, and Vedaldi (2017) introduced the Visual Decathlon Challenge which is concerned with incrementally converting an Imagenet classification model to new ones for another 9 different domain/tasks. To our knowledge there are only a few methods that have been proposed recently in order to solve it (Rebuffi, Bilen, and Vedaldi 2017; 2018; Rosenfeld and Tsotsos 2017; Mallya, Davis, and Lazebnik 2018). These works all have in common that incremental learning is achieved with layer-specific adapting modules (which are simply called adapters) applied to each CNN layer separately. Although the adapters have only a small number of parameters, because they are layer specific, the total number of parameters introduced by the adaptation process scales linearly with the the number of layers, and in

practice an adaptation network requires about 10% extra parameters (see also (Rebuffi, Bilen, and Vedaldi 2018)). Our main contribution is to propose a tensor method for multi-domain incremental learning that requires significantly less number of new parameters for each new task.
In summary, **our contributions** are:

- We propose to leverage joint parametrization of neural networks for multi-domain learning without catastrophic forgetting. Our method differs from previously proposed layer-wise adaptation methods (and their straightforward layer-wise extensions) by grouping all identically structured blocks of a CNN within a single high-order tensor.
- We perform a thorough evaluation of our model on the 10 datasets of the visual decathlon challenge and show that our method offers on average about $7.5\times$ reduction in model parameters compared with training a new network from scratch.
- The joint parametrization of the tensor with a low-rank tensor naturally leverages correlations across different layers. This results in learning more compact representations for each new task/domain.
- We show that this tensor approach outperforms methods based on matrix algebra that discard the multi-linear structure in the data.

Intuitively, our method first learns, on the source domain, a task agnostic core tensor. This represents a shared, domain-agnostic, latent subspace. For each new domains, this core is specialized by learning a set of task specific factors defining the multi-linear mapping from the shared subspace to the parameter space of each of the domains.

2 Closely Related Work

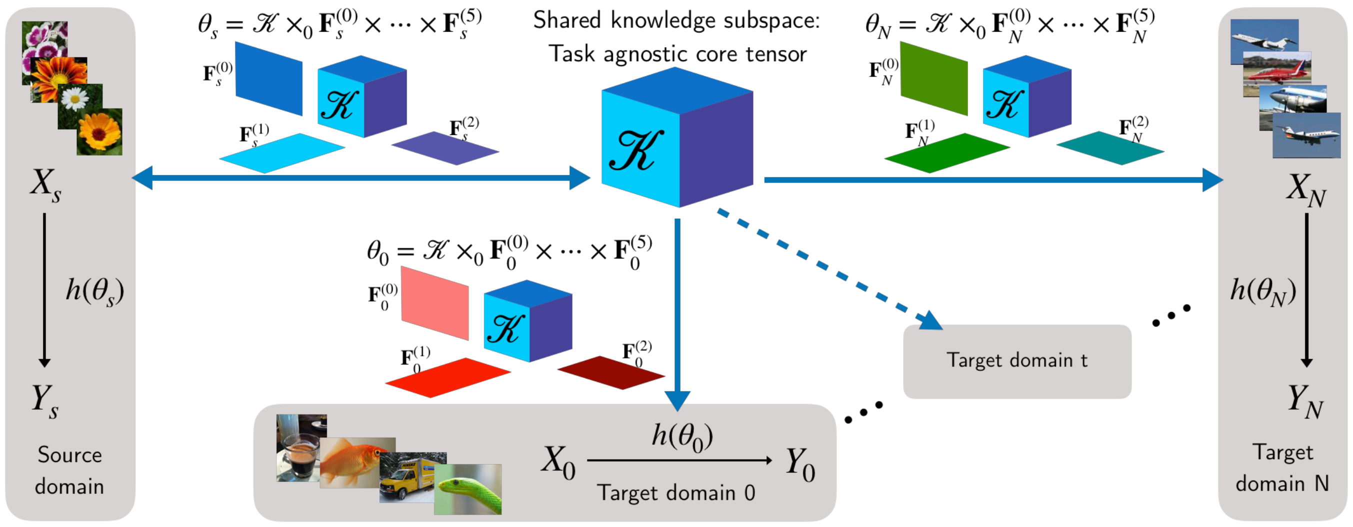
In this section, we review the related work on incremental multi-domain learning and tensor methods.

Incremental Multi-Domain Learning In the context of incremental learning, Rosenfeld and Tsotsos (2017) and Rebuffi, Bilen, and Vedaldi (2017) introduce the concept of layer adapters. Theses convert each layer[1] of a pre-trained CNN (typically on Imagenet) to adapt to a new classification task, for which new training data becomes available. Because the layers of the pre-trained CNN remain fixed, such approaches avoid the problem of catastrophic forgetting (French 1999; Li et al. 2017) so that performance on the original task is preserved. The method of (Rosenfeld and Tsotsos 2017) achieves this by computing new weights for each layer as a linear combination of old weights where the combination is learned in an end-to-end manner for all layers via back-propagation on the new task. The work in (Rebuffi, Bilen, and Vedaldi 2017) achieves the same goal by introducing small residual blocks composed of batch-norm followed by 1×1 convolutional layers after each 3×3 convolution of the original pre-trained network. Similarly, the newly introduced parameters are learned via back-propagation. The same work introduced the Visual Decathlon Challenge which is concerned with incrementally adapting an Imagenet classification model to 9 new and completely different domains and tasks. More recently, (Rebuffi, Bilen, and Vedaldi 2018) extends (Rebuffi, Bilen, and Vedaldi 2017) by making the adapters to work in parallel with the 3×3 convolutional layers. Although the adapters have only a small number of parameters each, they are layer specific, and hence the total number of parameters introduced by the adaptation process grows linearly with the the number of layers. In practice, an adaptation network requires about 10% extra parameters (see also (Rebuffi, Bilen, and Vedaldi 2018)). In (Mallya, Davis, and Lazebnik 2018) the authors propose to learn to adapt to a new task by learning how to mask individual weights of a pre-trained network. Following the same general idea, in (Morgado and Vasconcelos 2019), the authors introduce the so-called NetTailor method. Given an existing pretrained network with generic layers the methods learns how to combine them using a series of small task specific layers. (Guo et al. 2019a) makes uses of depthwise separable convolutions in order to build more efficient multi-task networks. In (Mancini et al. 2018), the authors propose to learn a task specific binary mask that selects a different set of activations depending of the target task.

Our method significantly differs from these works in that it models groups of identically structured blocks within a CNN with a single high-order tensor. This results in a much more compact representations for each new task/domain, with a latent subspace shared between domains. Only a set of factors, representing a very small fraction of this subspace, needs to be learnt for each new task or domain.

Tensor methods Herein, we focus on methods which have been used to re-parametrize existing deep neural networks. For a review of tensor methods, the reader is referred to existing surveys (Kolda and Bader 2009; Sidiropoulos et al. 2016; Papalexakis, Faloutsos, and Sidiropoulos 2016). In deep learning, tensor methods are typically used to speed up computation or to reduce the number of parameters. Convolutional kernels in particular can be decomposed and reformulated more efficiently using CP Lebedev et al.; Astrid and Lee (2015; 2017) or Tucker Kim et al. (2016) decomposition. Yang and Hospedales (2017) proposed a tensor factorization approach to multi-task learning. The weights of several networks (one per task) are parametrized, at each layer, with a low-rank tensor which allows to learn the sharing. The method of (Yunpeng et al. 2017) proposed a method to share parameters within a ResNeXt (Xie et al. 2017) block, by applying a Generalized Block Decomposition to a 4-th order tensor. As exemplified here, tensor algebraic operations have the potential to improve deep models. However, a straightforward extension of existing multi-domain adaptation methods to use tensor methods (e.g. (Rosenfeld and Tsotsos 2017)) can result in an adaptation model with a large number of parameters. To improve this, following (Kossaifi et al. 2019a), we propose to model groups of identically structured blocks within a CNN with a single high-order Tucker tensor, with a task agnostic core and task specific factors. We describe our method in details in the next section.

[1]The last layer typically requires retraining because the number of classes will in general be different.



$\theta_s = \mathcal{K} \times_0 \mathbf{F}_s^{(0)} \times \cdots \times \mathbf{F}_s^{(5)}$
$\mathbf{F}_s^{(0)}$
$\mathbf{F}_s^{(1)}$
$\mathbf{F}_s^{(2)}$
Shared knowledge subspace:
Task agnostic core tensor
$\theta_N = \mathcal{K} \times_0 \mathbf{F}_N^{(0)} \times \cdots \times \mathbf{F}_N^{(5)}$
$\mathbf{F}_N^{(0)}$
$\mathbf{F}_N^{(1)}$
$\mathbf{F}_N^{(2)}$
X_s
$h(\theta_s)$
Y_s
Source domain
$\theta_0 = \mathcal{K} \times_0 \mathbf{F}_0^{(0)} \times \cdots \times \mathbf{F}_0^{(5)}$
$\mathbf{F}_0^{(0)}$
$\mathbf{F}_0^{(1)}$
$\mathbf{F}_0^{(2)}$
Target domain t
X_0
$h(\theta_0)$
Y_0
Target domain 0
X_N
$h(\theta_N)$
Y_N
Target domain N

Figure 1: **Overview of our method** First, a task agnostic core $\mathcal{K}$ is learned jointly with the domain specific factors $\mathbf{F}_s^{(0)}, \cdots, \mathbf{F}_s^{(5)}$ on the source domain/task (left). For a new target domain/task, the same core is specialized for the new task by training a new set of factors $\mathbf{F}_t^{(0)}, \cdots, \mathbf{F}_t^{(5)}$ (bottom), similarly for any new task $k \in [1 \,..\, N]$ (right). Intuitively, the core represents a task agnostic subspace, while the task specific factors define the multi-linear mapping from that subspace to the parameter space of each of the domains. Note that here, we represent the 6th order tensors in 3D for clarity.

3 Method

In this section, we introduce our method (depicted in Figure 1) for incremental multi-domain learning, starting by the notation used (Section. 3.1). By considering a source domain X^s and output space Y^s, we aim to learn a function h (here, a ResNet based architecture) parametrized by a tensor θ^s, $h(\theta^s)\colon X^s \rightarrow Y^s$. The model and its tensor parametrization are introduced in detail in Section 3.2. The main idea is to then learn a task agnostic latent manifold $\mathcal{K}$ on the source domain. The parameter tensor θ^s is obtained from $\mathcal{K}$ with task specific factors $\mathbf{F}_s^{(0)}, \cdots, \mathbf{F}_s^{(N)}$. Given a new target task, we then adapt h and learn a new parameter tensor θ^t by specialising $\mathcal{K}$ with a new set of task specific factors $(\mathbf{F}_t^{(0)}, \cdots, \mathbf{F}_t^{(N)})$. This learning process is detailed in Section 3.3. In practice, most of the parameters are shared in $\mathcal{K}$, while the factors only contain a fraction of the parameters, which leads to large parameters savings. We offer an in-depth analysis of these space savings in Section 3.4.

3.1 Notation

In this paper, we denote **vectors** (1st order tensors) as $\mathbf{v}$, **matrices** (2nd order tensors) as $\mathbf{M}$ and **tensors**, which generalize the concept of matrices for orders (number of dimensions) higher than 2, as $\mathcal{X}$. **Id** is the identity matrix. **Tensor contraction** with a matrix, also called n–mode product, is defined, for a tensor $\mathcal{X} \in \mathbb{R}^{D_0 \times D_1 \times \cdots \times D_N}$ and a matrix $\mathbf{M} \in \mathbb{R}^{R \times D_n}$, as the tensor $\mathcal{T} = \mathcal{X} \times_n \mathbf{M} \in \mathbb{R}^{D_0 \times \cdots \times D_{n-1} \times R \times D_{n+1} \times \cdots \times D_N}$, with:

$$\mathcal{T}_{i_0, i_1, \cdots, i_N} = \sum_{k=0}^{D_n} \mathbf{M}_{i_n, k} \mathcal{X}_{i_0, \cdots, i_{n-1}, k, i_{n+1}, \cdots, i_N}.$$

3.2 Latent Network Parametrization

We propose to group all the parameters of a neural network into a set of high-order hyper-parameter tensors (Kossaifi et al. 2019a). We do so by collecting all the weights of the neural network into 3 parameter tensors $\theta^{(0)}$ and $\theta^{(2)}$, all of order 6. While the proposed method is not architecture specific, to allow for a fair comparison in terms of overall representation power, we follow (Rebuffi, Bilen, and Vedaldi 2017; 2018; Rosenfeld and Tsotsos 2017) and use a modified ResNet-26 (He et al. 2016). The network consists of 3 macro-modules, each consisting of 4 basic residual blocks (He et al. 2016). Each of these blocks contain two convolutional layers with 3×3 filters. Following (Rebuffi, Bilen, and Vedaldi 2017), the macro-modules output 64, 128, and 256 channels respectively. Throughout the network the resolution is dropped multiple times. First, at the beginning of each macro-module using a convolutional layer with a stride of 2. A final drop in resolution is done at the end of the network, before the classification layer, using an adaptive average pooling layer that reduces the spatial dimensions to resolution of 1×1 px.

In order to facilitate the proposed grouped tensorization process, we moved the feature projection layer (a convolutional layer with 1×1 filters), required each time the number of features changes between blocks, outside of the macro-modules (i.e. we place a convolutional layer with a 1×1 kernel before the 2nd and 3rd macro-modules).

We closely align our tensor re-parametrization to the network structure by grouping together all the convolutional layers within the same macro-module. For each macro-module $b \in \{0, 1, 2\}$, we construct a 6th -order tensor collecting the weights in that group:

$$\theta^{(b)} \in \mathbb{R}^{D_0 \times D_1 \times \cdots \times D_5} \tag{1}$$

where $\mathcal{W}^b$ is the tensor for the b^{th} macro-module. The 6 dimensions of the tensor are obtained as follows: $D_0 \times D_1 \times D_2 \times D_3$ corresponds to the shape of the weights of a particular convolution layer and represents the number of output channels, number of input channels, kernel width and kernel height respectively. The $D_4{}^{\text{th}}$ mode corresponds to the number of basic blocks per residual module (2 in this case) and, D_5 corresponds to the number of residual blocks present in each macro-module (4 for the specific architecture used).

Our model should be compared with previous methods for incremental multi-domain adaptation like (Rosenfeld and Tsotsos 2017) (the method of (Rebuffi, Bilen, and Vedaldi 2017) can be expressed in a similar way) which learn a linear transformation per layer. In particular, (Rosenfeld and Tsotsos 2017) learns a 2D adaptation matrix $\mathbf{F} \in \mathbb{R}^{D_0 \times (D_1 \times D_2 \times D_3)}$ per convolutional layer. Moreover, prior work on tensors (e.g. (Kim et al. 2016)) has focused on standard layer-wise modelling with a 4^{th}-order convolutional filter, of shape $D_0 \times D_1 \times D_2 \times D_3$. In contrast, our model has two additional dimensions incorporating intra-architecture correlations and can accommodate an arbitrary number of dimensions depending on the architecture used. Our approach is in particular not architecture specific and can be used for various network architectures.

3.3 Multi-Domain Tensorized Learning

We now consider a scenario with T tasks, from potentially very different domains. The traditional approach would consist in learning as many models, one for each task. In our framework, this would be equivalent to learning one parameter tensor $\theta_d^{(b)}$ independently for each task d and macro-module b. Since the reasoning is the same for each of the 3 macro-modules, for clarity and without loss of generality, we omit the b in the remaining of the paper. We propose that all the parameters are obtained from a shared latent subspace, modelled by a task agnostic tensor $\mathcal{K}$. The (multi-linear) mapping between this task agnostic core and the parameter tensor is then given by a set of task specific factors $(\mathbf{F}_s^{(0)}, \cdots \mathbf{F}_s^{(5)})$ that specialize the task agnostic subspace for the source domain s. Specifically, we write, for the source domain s:

$$\theta_s = \mathcal{K} \times_0 \mathbf{F}_s^{(0)} \times_1 \mathbf{F}_s^{(1)} \times \cdots \times_5 \mathbf{F}_s^{(5)}, \tag{2}$$

where $\mathcal{K} \in \mathbb{R}^{D_0 \cdots \times D_5}$ is a **task-agnostic** full rank core **shared** between all domains and $(\mathbf{F}_s^{(0)}, \mathbf{F}_s^{(1)}, \cdots, \mathbf{F}_s^{(5)})$ a set of **task specific** projection factors (for domain s). We assume here that the task used to train the shared core is a general one with many classes and large amount of training data (here, Imagenet classification). Moreover, a key observation to make at this point is that the number of parameters for the factors is orders of magnitudes smaller than the number of parameters of the core.

For each new target domain t, we form a new parameter tensor θ_t obtained from the *same* latent subspace $\mathcal{K}$. This is done by learning a new set of factors $(\mathbf{F_t}^{(0)}, \cdots \mathbf{F_t}^{(5)})$ to specialize $\mathcal{K}$ for the new task:

$$\theta_t = \mathcal{K} \times_0 \mathbf{F_t}^{(0)} \times_1 \mathbf{F_t}^{(1)} \times \cdots \times_5 \mathbf{F_t}^{(5)} \tag{3}$$

Note that the new factors represent only *a small fraction* of the total number of parameters, the majority of which are contained within the shared latent subspace. By expressing the new weight tensor θ_t as a function of the factors $\mathbf{F_t}$, one can learn them on the new task given that labelled data are available in an end-to-end manner via back-propagation. This allows to efficiently adapt the domain agnostic subspace to the new domains while retaining the performance on the original task, and training only a small number of additional parameters. Fig. 1 shows a graphical representation of our method, where the weight tensors have been simplified to 3D for clarity.

Auxiliary loss function: To prevent degenerate solutions and facilitate learning, we additionally explore orthogonality constraints on the task specific factors. This type of constraints have been shown to encourage regularization, improving the overall convergence stability and final accuracy (Brock et al. 2016; Bansal, Chen, and Wang 2018). In addition, by adding such constraint, we aim to enforce the factors of the decomposition to be full-column rank, which would ensure that the core of the decomposition preserves essential properties of the full weight tensor such as the Kruskal rank (Jiang, Yang, and Zhang 2017). In practice, rather than a hard constraint, we add a loss to the objective function:

$$\mathcal{L} = \lambda \sum_{k=0}^{5} \| \left(\mathbf{F_k}^{(k)}\right)^{\top} \mathbf{F_k}^{(k)} - \mathbf{Id} \|_F^2. \tag{4}$$

The regularization parameter λ was validated on a small validation set.

3.4 Complexity Analysis

In terms of unique, task specific parameters learned, our grouping strategy is significantly more efficient than a layer-wise parametrization. For a given group of convolutional layers, in this work defined by the macro-module structure present in a ResNet architecture, we can express the total number of parameters for a Layer-wise Tucker case (this is not proposed in this work but mentioned here for comparison purposes) as follows: $N_{\text{layerwise}} = (D_4 \times D_5) \times (\sum_{k=0}^{3} D_n R_k)$.

In particular, in the case of a full rank decomposition, by denoting $L = D_4 \times D_5$ the number of convolutional layers, we get:

$$N_{\text{layerwise}} = \underbrace{(D_4 \times D_5)}_{L} \times (D_0^2 + D_1^2 + D_2^2 + D_3^2), \tag{5}$$

where L is the number of re-parametrized layers in a given group.

For the linear case (Rosenfeld and Tsotsos 2017), we have that $D_0 = D_1 = D_c$, and the number of parameters simplifies to: $N_{\text{linear}} = \underbrace{(D_4 \times D_5)}_{L} \times D_c^2$ As opposed to this, for our proposed method, by grouping the parameters together into a single high-order tensor, the total number of parameters is: $N_{\text{T-Net}} = \sum_{k=0}^{5} D_n R_k$. For the full-rank case ($D_n = R_k$), this simplies to:

$$N_{\text{T-Net}} = D_0^2 + D_1^2 + D_2^2 + D_3^2 + \underbrace{D_4^2 + D_5^2}_{(\frac{L}{D_5})^2 + (\frac{L}{D_4})^2} \tag{6}$$

Note that here, $D_4 = 2$ and $D_5 = 4$ so $D_4^2 + D_5^2 \leq \frac{L^2}{4}$.

Because in practice $(D_0^2 + D_1^2 + D_2^2 + D_3^2) \gg L^2$, by using the proposed method, we achieve $\frac{N_{\text{layerwise}}}{N_{\text{T-Net}}} \approx L$ times less task-specific parameters.

Substituting the variables from Eq. (5) and Eq. (6) with the numerical values specific to the architecture used in this work we obtain in total: $N_{\text{layerwise}} = 1,376,688$ parameters. By contrast, using the same setting for our proposed method, we get $N_{\text{T-Net}} = 172,068$, thus verifying $\frac{N_{\text{layerwise}}}{N_{\text{T-Net}}} \approx 8 = L$.

Making the same assumptions as for the linear case, given that we use square convolutional kernels (i.e. $D_2 = D_3 = D_n$), and $D_c \gg D_n$, Eq. (6) becomes: $N_{\text{T-Net}} \leq 2D_c^2 + \frac{L^2}{4}$, resulting in $\approx \frac{L}{2}$ less parameters than in the linear case ($\frac{L}{2} = 4$ for the model used).

Conclusion: Our proposed approach uses L **times less parameters per group** than the layers-wise Tucker decomposition and $\frac{L}{2}$ **times less parameters than the layer-wise linear decomposition**. For the ResNet-26 architecture used in this work $L = 8$.

4 Experimental Setting

In this section, we detail the experimental setting, metrics used and implementation details.

Datasets: We evaluate our method on the 10 different datasets from very different visual domains that compose the Decathlon challenge (Rebuffi, Bilen, and Vedaldi 2017). Note that this dataset where modified in (Rebuffi, Bilen, and Vedaldi 2017), mainly by resizing and cropping them to the same resolution (72×72px). This challenge assesses explicitly methods designed to solve incremental multi-domain learning without catastrophic forgetting. **Imagenet** (Russakovsky et al. 2015) contains 1.2 millions images distributed across 1000 classes. Following (Rebuffi, Bilen, and Vedaldi 2017; 2018; Rosenfeld and Tsotsos 2017), this was used as the source domain to train the shared low-rank manifold for our model as detailed in Eq. (2). The **FGVC-Aircraft Benchmark** (Airc.) (Maji et al. 2013) contains 10,000 aircraft images across 100 different classes; **CIFAR100** (C100) (Krizhevsky and Hinton 2009) is composed of 60000 small images in 100 classes; **Daimler Mono Pedestrian Classification Benchmark** (DPed) (Munder and Gavrila 2006) is a dataset for pedestrian detection (binary classification) composed of 50,000 images; **Describable Texture Dataset** (DTD) (Cimpoi et al. 2014) contains 5640 images, for 47 texture categories; the **German Traffic Sign Recognition** (GTSR) Benchmark (Stallkamp et al. 2012) is a dataset of $50,000$ images of 43 traffic sign categories; **Flowers102** (Flwr) (Nilsback and Zisserman 2008) contains 102 flower categories with between 40 and 258 images per class; **Omniglot** (OGlt) (Lake, Salakhutdinov, and Tenenbaum 2015) is a dataset of 32000 images representing 1623 handwritten characters from 50 different alphabets; the **Street View House Numbers** (SVHN) (Netzer et al. 2011) is a digit recognition dataset containing 70000 images in 10 classes. Finally, **UCF101** (UCF) (Soomro, Zamir, and Shah 2012) is an action recognition dataset composed of 13,320 images representing 101 action classes.

Metrics: We follow the evaluation protocol of the Decathlon Challenge and report results in terms of mean accuracy and decathlon score S, computed as follows:

$$S = \sum_{t=1}^{N} \beta_t \max\{0, E_t^{reference} - E_t\}^{\lambda_t}, \tag{7}$$

where $E_t^{reference}$ is considered to be the upper limit allowed for a given task t in order to receive points, λ_t is an exponent that controls the reward proportionality, and β_t a scalar that enforces the limit of 1000 points per task. $E^{reference} = 2E^{baseline}$ where $E_{baseline}$ is the strong baseline from (Rebuffi, Bilen, and Vedaldi 2017).

One key limitation of this metric is that it doesn't take in consideration the model capacity or the methods compression abilities. As such, following (Mancini et al. 2018; Berriel et al. 2019) we also report an efficiency based scored (*EScore*) that is simply computed by dividing the decathlon score by the relative number of parameters required across all tasks with respect to the original ResNet-26 network from (Rebuffi, Bilen, and Vedaldi 2017).

Implementation details: We first train our adapted ResNet-26 model on ImageNet for 90 epochs using SGD with momentum (0.9), using a learning rate of 0.1 that is decreased in steps by $10\times$ every 30 epochs. To avoid overfitting, we use a weight decay equal to 10^{-5}. During training, we follow the best practices and randomly apply scale jittering, random cropping and flipping. We initialize our weights from a normal distribution $\mathcal{N}(0, 0.002)$, before decomposing them using Tucker decomposition (Section 3). Finally, we train the obtained core and factors (via backpropagation) by reconstructing the weights on the fly.

For the remaining 9 domains, we load the task-independent core and the factors trained on imagenet, freeze the core weights and only fine-tune the factors, batch-norm layers and the two 1×1 projection layers, all of which account for $\approx 3.5\%$ of the total number of parameters in total. The linear layer at the end of the network is trained from scratch for each task and was initialized from a uniform distribution. Depending on the size of the dataset, we adjust the weight decay to avoid overfitting (10^{-5} for larger datasets) and up to 0.005 for the smaller ones (e.g. Flowers102).

We used PyTorch (Paszke et al. 2017) to implement and train the models and TensorLy (Kossaifi et al. 2019b) for all tensor operations.

Model	#param	Dataset										Mean	EScore	Score
		ImNet	Airc.	C100	DPed	DTD	GTSR	Flwr	OGlt	SVHN	UCF			
Model \Number of images	-	1.3M	7K	50K	30K	4K	40K	2K	26K	70K	9K	-	-	-
Rebuffi et al. 2017	2×	59.23	63.73	81.31	93.30	57.02	97.47	83.43	89.82	96.17	50.28	77.17	1322	2643
Rosenfeld et al. 2017	2×	57.74	64.11	80.07	91.29	56.54	98.46	86.05	89.67	96.77	49.38	77.01	1425	2851
Mallya et al. 2018	1.28×	57.69	65.29	79.87	96.99	57.45	97.27	79.09	87.63	97.24	47.48	76.60	2217	2838
S. Adap. (Rebuffi et al. 2017)	2×	60.32	61.87	81.22	93.88	57.13	99.27	81.67	89.62	96.57	50.12	77.17	1580	3159
P. Adap. (Rebuffi et al. 2017)	2×	60.32	64.21	81.91	94.73	58.83	99.38	84.68	89.21	96.54	50.94	78.07	1706	3412
P. SVD (Rebuffi et al. 2017)	1.5×	60.32	66.04	81.86	94.23	57.82	99.24	85.74	89.25	96.62	52.50	78.36	2265	3398
NetTailor (Morgado et al. 2019)	1.5×	61.42	75.07	81.84	94.68	61.28	99.52	88.53	90.09	96.44	49.54	79.64	2496	3744
SpotTune (Guo et al. 2019b)	11×	60.32	63.91	80.48	94.49	57.13	99.52	85.22	88.84	96.72	52.34	77.89	328	3612
Depthwise (Guo et al. 2019a)*	1×	63.99	61.06	81.20	97.00	55.48	99.27	85.67	89.12	96.16	49.33	77.82	3507	3507
Bin. mask (Mancini et al. 2018)	1.29×	60.8	52.8	82.0	96.2	58.7	99.2	88.2	89.2	96.8	48.6	77.2	-	-
BA^2 (Berriel et al. 2019)	1.03×	56.9	49.9	78.1	95.5	55.1	99.4	86.1	88.7	96.9	50.2	75.7	3105	3199
Ours	1.35×	61.48	67.36	80.84	93.22	59.10	99.64	88.99	88.91	96.95	47.90	78.43	2656	3585

Table 1: **Comparison to the state-of-the-art:** Top-1 classification accuracy (%) and overall decathlon scores on all 10 dataset from the Visual Decathlon challenge. Our method is generic, applicable to any network architecture. It offers a good balance between accuracy and number of task-specific parameters used.

5 Results

First, we assess the performance of the proposed approach (Section 5.1) and compare it with the state-of-the-art on the challenging Visual Decathlon (Rebuffi, Bilen, and Vedaldi 2017). In Section 5.2, we study of the method, including the importance of source dataset, the influence of the amount of data on the overall performance and the effect of the constraints imposed on the core and factors of the model.

5.1 Comparison with State-of-the-Art

Herein, we compare against the current state-of-the-art methods on multi-domain transfer learning (Rebuffi, Bilen, and Vedaldi 2017; 2018; Rosenfeld and Tsotsos 2017; Mallya, Davis, and Lazebnik 2018) on the decathlon dataset. We train our core subspace on ImageNet and incrementally adapt to all 9 other domains. We report, for all methods, the relative increase in number of parameters (per domain), the top-1 accuracy on each of the 10 domain, as well as the average accuracy and overall challenge score, Table 1.

Overall, our approach offers competitive results in terms of both accuracy and efficiency (i.e. number of introduced task-specific parameters), offering a good balance between the two. When compared within the same class of methods, that directly apply a form of matrix decomposition (Rebuffi, Bilen, and Vedaldi 2018; Rosenfeld and Tsotsos 2017) our approach outperforms all of them, including the joint compression method of (Rebuffi, Bilen, and Vedaldi 2018) (denoted as "Parallel SVD") that takes advantage of the data redundancy in-between tasks. We could go further and impose a weight sharing within the prediction layers by removing the flattening and fully-connected layers altogether, replace them with a tensor regression layer (TRL) (Kossaifi et al. 2018). Our approach can then be readily applied to the low-rank Tucker tensor of the TRL. Furthermore, our method could be combined with binary masking approaches (Berriel et al. 2019; Mallya, Davis, and Lazebnik 2018) that reduce the number of additional parameters by compacting the binary values using bit packing.

5.2 Inter-class Transfer Learning

Most of the recent work on multi-domain incremental learning attempts to transfer the knowledge from a model, pre-trained on a large scale dataset such as ImageNet to another easier dataset and/or task. In this work, we go on step further and explore the efficiency of our transfer learning approach when such source dataset or computational resources are not available, by starting from a model pre-trained on a much smaller dataset. Table 2 shows the results for a network pre-trained of CIFAR100. Notice that on some datasets (i.e. GTSRB, OGlt) such model can match or marginally surpass the performance of its Imagenet counterpart. On the other hand, on some of the more challenging datasets (i.e. DTD, aircraft) there is still a large gap. This suggest that the features learned by Cifar-trained model are less generic and diverse. This is due to both the low quantity of available samples and the easiness/overfitting on the original dataset. A potential solution for this may be to enforce a diversity loss, and fine-tune the core jointly on all tasks. However we leave the exploration of this area for future works.

5.3 Varying the Amount of Training Data

An interesting aspect of incremental multi-domain learning not addressed thus far is the case where only a limited amount of labelled data available for the new domain or tasks, and how this affects performance. Although not all 9 remaining tasks of the Decathlon assume abundance of training data, in this section, we systematically assess the sensitivity to the amount of training data. Specifically, we vary the amount of training data for 4 tasks, namely DPed, DTD, GTSRB, UFC. Fig. 2 shows the classification accuracy on these datasets as function of the amount of training data. In the same figure, we also report the performance of a network for which both the cores and the factors are fine-tuned on these datasets, also trained with the same amount of data. In general, especially on the smaller dataasets, we observe that our method is at least as good as the fine-tuned network which should be considered as a very strong base-

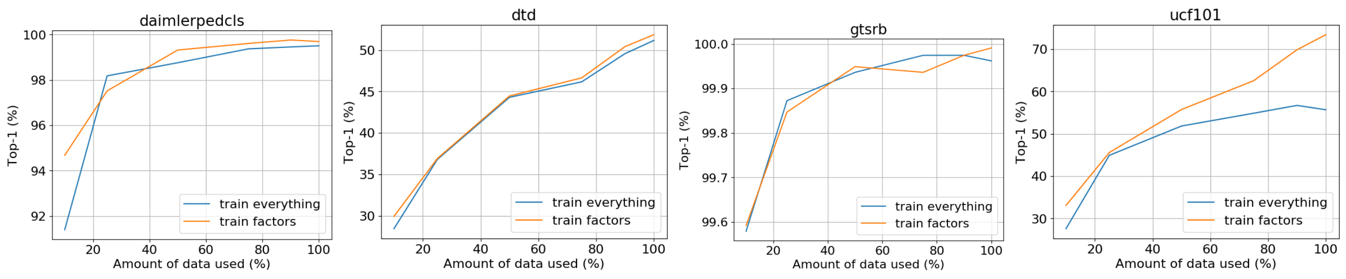


Figure 2: Top-1 classification accuracy (%) on DPed, DTD, GTSRB, UFC as function of the amount of training data. Our method is compared with the performance of a network for which both the cores and the factors are fine-tuned on these datasets, also trained with the same amount of data.

Model	Pretrained on	Airc.	C100	DPed	DTD	GTSR	Flwr	OGlt	SVHN	UCF
Ours	ImageNet	55.6	80.7	99.67	52.2	99.96	83.8	88.18	95.66	78.6
	Cifar100	41.7	74.5	99.82	37.55	99.98	70.9	88.35	95.43	72.1

Table 2: Mean Top-1 accuracy (%) on the unseen validation set, reported for two settings: (a) A model trained on Imagenet and adapted for the rest of the datasets, (same as the one used for the decathlon setting) (first row) and (b) a more challenging scenario where we train a model on Cifar100 and adapt it for the other datasets (second row). Notice that our method produced satisfactory results even for setting (b), marginally outperforming the Imagenet model on some datasets. This clearly illustrates the representational power of learned model and the generalization capabilities of the proposed method.

line, requiring as many parameters as the original Imagenet-trained model. This validates the robustness of our model for the case of training with limited amount of training data.

5.4 Rank Regularization

It is well-known that low-rank structure act as regularization mechanisms (Tai et al. 2015). By jointly modelling the parameters of our model as a high order tensor, our model allows such constraint. Limiting the multi-linear rank of these tensors effectively regularizes the whole network, thus

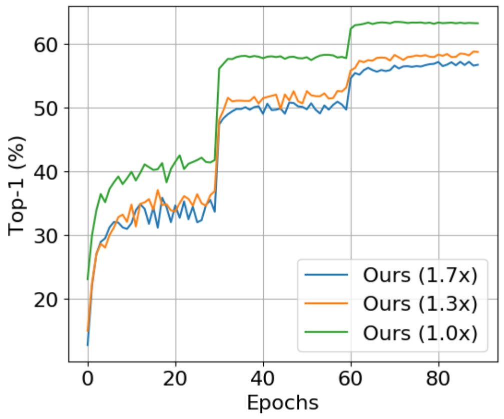


Figure 3: **Effect of the rank regularization** on ImageNet when training from scratch, for ranks achieving compression ratios of 1.0 (full-rank), 1.3$\times$ (reducing the rank with one over the number of blocks dimension) and 1.7$\times$ (decreasing the rank of #input and #output channels).

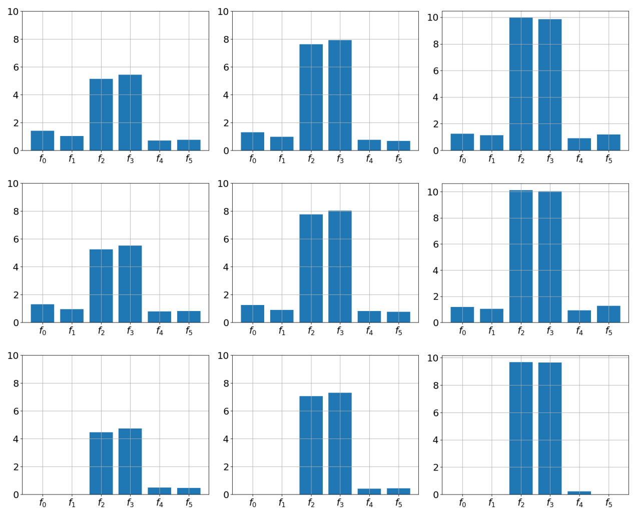


Figure 4: **Orthogonality loss on the factors** on the DTD dataset. We show the orthogonality loss $\mathcal{L}$ defined in Eq. (4), for $\lambda = 10^{-3}$ (first row), $\lambda = 10^{-2}$ (second row) and $\lambda = 10^{-1}$ (third row). Unsurprisingly, the orthogonality constraint is mostly violated for small values of λ. Interestingly, we observe that the factors are almost all orthogonal, except for the dimensions D_2 and D_3 of the weight tensor, which correspond respectively to the number of input and output channels, confirming that these require the most adaptation from the task agnostic subspace.

preventing over-fitting. This also allows for more efficient representations, by leveraging the redundancy in the multi-

Dataset	$\lambda = 0.1$	$\lambda = 0.01$	$\lambda = 0.001$
DTD	52.2%	51.3%	51.0%
vgg-flowers	80.9%	83.8%	82.2%

Table 3: Effect of enforcing weight orthogonality constraints for various values of $\lambda = \{0.1, 0.01, 0.001\}$ on two datasets: DTD and vgg-flowers. Results reported in terms of Top-1 classification accuracy (%) on the validation set of the these datasets.

linear structure of the network, allowing for large compression ratios, without decrease in performance.

In this section we investigate this possibility by studying the effect of such constraint. To this end, we attempted to train our Imagenet model by imposing a low-rank constraint on the weight tensor. However, as can be seen in Fig. 3, this leads to a significant drop in performance on the base task of Imagenet; hence we did not pursue the possibility of rank regularization further. We attribute this effect to the very small number of parameters in our ResNet model.

5.5 Effect of orthogonality regularization

To prevent degenerate solutions and facilitate learning, we added orthogonality constraints on the task specific factors. This type of constraints have been shown to encourage regularization, improving the overall convergence stability and final accuracy (Brock et al. 2016; Bansal, Chen, and Wang 2018).In addition, by adding such constraints, we aim to enforce the factors of the decomposition to be full-column rank, which would ensure that the core of the decomposition preserves essential properties of the full weight tensor such as the Kruskal rank (Jiang, Yang, and Zhang). This orthogonality constraint was enforced using a regularization term, rather than via a hard constraint. See Table 3 for results on two small datasets, namely DTD and vgg-flowers.

6 Conclusions

We presented a method for incremental multi-domain learning using a latent tensor factorization of the network. By modelling groups of identically structured blocks within a CNN as a high-order tensor, we are able to express the parameter space of a deep neural network as a (multi-linear) function of a task-agnostic subspace. This task-agnostic core is then specialized by learning a set of small, task-specific factors for each new domain. While previous methods which have focused on adapting each layer separately, we show that our proposed joint modelling naturally leverages correlations across different filters and layers, resulting in a more compact representation for each new task/domain. We evaluate the proposed method on the 10 datasets of the Visual Decathlon Challenge and show that our method offers on average about $7.5\times$ reduction in model parameters offering competitive results, both in terms of classification accuracy and Decathlon points.

References

Astrid, M., and Lee, S. 2017. Cp-decomposition with tensor power method for convolutional neural networks compression. *arxiv*.

Bansal, N.; Chen, X.; and Wang, Z. 2018. Can we gain more from orthogonality regularizations in training deep cnns? *arXiv*.

Berriel, R.; Lathuilière, S.; Nabi, M.; Klein, T.; Oliveira-Santos, T.; Sebe, N.; and Ricci, E. 2019. Budget-aware adapters for multi-domain learning. *arXiv*.

Brock, A.; Lim, T.; Ritchie, J. M.; and Weston, N. 2016. Neural photo editing with introspective adversarial networks. *arXiv*.

Cimpoi, M.; Maji, S.; Kokkinos, I.; Mohamed, S.; and Vedaldi, A. 2014. Describing textures in the wild. In *CVPR*.

French, R. M. 1999. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences* 3(4):128–135.

Guo, Y.; Li, Y.; Feris, R.; Wang, L.; and Rosing, T. 2019a. Depthwise convolution is all you need for learning multiple visual domains. In *AAAI*.

Guo, Y.; Shi, H.; Kumar, A.; Grauman, K.; Rosing, T.; and Feris, R. 2019b. Spottune: transfer learning through adaptive fine-tuning. In *CVPR*, 4805–4814.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*.

He, K.; Gkioxari, G.; Dollár, P.; and Girshick, R. 2017. Mask r-cnn. In *ICCV*.

Jiang, B.; Yang, F.; and Zhang, S. Tensor and its tucker core: The invariance relationships. *Numerical Linear Algebra with Applications* 24(3):e2086.

Jiang, B.; Yang, F.; and Zhang, S. 2017. Tensor and its tucker core: the invariance relationships. *Numerical Linear Algebra with Applications* 24(3):e2086.

Kim, Y.; Park, E.; Yoo, S.; Choi, T.; Yang, L.; and Shin, D. 2016. Compression of deep convolutional neural networks for fast and low power mobile applications. *ICLR*.

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

Kossaifi, J.; Lipton, Z. C.; Khanna, A.; Furlanello, T.; and Anandkumar, A. 2018. Tensor regression networks. *arxiv*.

Kossaifi, J.; Bulat, A.; Tzimiropoulos, G.; and Pantic, M. 2019a. T-net: Parametrizing fully convolutional nets with a single high-order tensor. In *CVPR*.

Kossaifi, J.; Panagakis, Y.; Anandkumar, A.; and Pantic, M. 2019b. Tensorly: Tensor learning in python. *JMLR* 20(26):1–6.

Krizhevsky, A., and Hinton, G. 2009. Learning multiple layers of features from tiny images. Technical report, Citeseer.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*.

Lake, B. M.; Salakhutdinov, R.; and Tenenbaum, J. B. 2015. Human-level concept learning through probabilistic program induction. *Science* 350(6266):1332–1338.

Lebedev, V.; Ganin, Y.; Rakhuba, M.; Oseledets, I. V.; and Lempitsky, V. S. 2015. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *ICLR*.

Li, D.; Yang, Y.; Song, Y.-Z.; and Hospedales, T. M. 2017. Learning to generalize: Meta-learning for domain generalization. *arXiv*.

Long, J.; Shelhamer, E.; and Darrell, T. 2015. Fully convolutional networks for semantic segmentation. In *CVPR*.

Maji, S.; Rahtu, E.; Kannala, J.; Blaschko, M.; and Vedaldi, A. 2013. Fine-grained visual classification of aircraft. *arXiv*.

Mallya, A.; Davis, D.; and Lazebnik, S. 2018. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *ECCV*.

Mancini, M.; Ricci, E.; Caputo, B.; and Rota Bulò, S. 2018. Adding new tasks to a single network with weight transformations using binary masks. In *ECCV*, 0–0.

Morgado, P., and Vasconcelos, N. 2019. Nettailor: Tuning the architecture, not just the weights. In *CVPR*, 3044–3054.

Munder, S., and Gavrila, D. M. 2006. An experimental study on pedestrian classification. *IEEE TPAMI* 28(11):1863–1868.

Netzer, Y.; Wang, T.; Coates, A.; Bissacco, A.; Wu, B.; and Ng, A. Y. 2011. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshops*, volume 2011.

Nilsback, M.-E., and Zisserman, A. 2008. Automated flower classification over a large number of classes. In *ICVGIP*.

Papalexakis, E. E.; Faloutsos, C.; and Sidiropoulos, N. D. 2016. Tensors for data mining and data fusion: Models, applications, and scalable algorithms. *ACM Trans. Intell. Syst. Technol.* 8.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in pytorch. In *NIPS-W*.

Rebuffi, S.-A.; Bilen, H.; and Vedaldi, A. 2017. Learning multiple visual domains with residual adapters. In *NIPS*.

Rebuffi, S.-A.; Bilen, H.; and Vedaldi, A. 2018. Efficient parametrization of multi-domain deep neural networks. In *CVPR*.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*.

Rosenfeld, A., and Tsotsos, J. K. 2017. Incremental learning through deep adaptation. *arXiv*.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. 2015. Imagenet large scale visual recognition challenge. *IJCV* 115(3).

Sidiropoulos, N. D.; De Lathauwer, L.; Fu, X.; Huang, K.; Papalexakis, E. E.; and Faloutsos, C. 2016. tensor decomposition for signal processing and machine learning. *arXiv*.

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

Soomro, K.; Zamir, A. R.; and Shah, M. 2012. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv*.

Stallkamp, J.; Schlipsing, M.; Salmen, J.; and Igel, C. 2012. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks* 32.

Tai, C.; Xiao, T.; Wang, X.; and E, W. 2015. Convolutional neural networks with low-rank regularization. *arxiv*.

Xie, S.; Girshick, R.; Dollár, P.; Tu, Z.; and He, K. 2017. Aggregated residual transformations for deep neural networks. In *CVPR*.

Yang, Y., and Hospedales, T. 2017. Deep multi-task representation learning: A tensor factorisation approach. *ICLR*.

Yunpeng, C.; Xiaojie, J.; Bingyi, K.; Jiashi, F.; and Shuicheng, Y. 2017. Sharing residual units through collective tensor factorization in deep neural networks. *arXiv*.