This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

Efficient Multi-Domain Learning by Covariance Normalization

Yunsheng Li Nuno Vasconcelos
University of California San Diego
La Jolla, CA 92093
yul554@ucsd.edu, nvasconcelos@ucsd.edu

Abstract

The problem of multi-domain learning of deep networks is considered. An adaptive layer is induced per target domain and a novel procedure, denoted covariance normalization (CovNorm), proposed to reduce its parameters. CovNorm is a data driven method of fairly simple implementation, requiring two principal component analyzes (PCA) and fine-tuning of a mini-adaptation layer. Nevertheless, it is shown, both theoretically and experimentally, to have several advantages over previous approaches, such as batch normalization or geometric matrix approximations. Furthermore, CovNorm can be deployed both when target datasets are available sequentially or simultaneously. Experiments show that, in both cases, it has performance comparable to a fully fine-tuned network, using as few as 0.13% *of the corresponding parameters per target domain.*

1. Introduction

Convolutional nerual networks (CNNs) have enabled transformational advances in classification, object detection and segmentation, among other tasks. However they have non-trivial complexity. State of the art models contain millions of parameters and require implementation in expensive GPUs. This creates problems for applications with computational constraints, such as mobile devices or consumer electronics. Figure 1 illustrates the problem in the context of a smart home equipped with an ecology of devices such as a camera that monitors package delivery and theft, a fridge that keeps track of its content, a treadmill that adjusts fitness routines to the facial expression of the user, or a baby monitor that keeps track of the state of a baby. As devices are added to the ecology, the GPU server in the house must switch between a larger number of classification, detection, and segmentation tasks. Similar problems will be faced by mobile devices, robots, smart cars, etc.

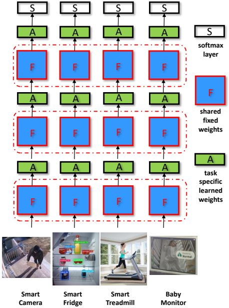

Figure 1: Multi-domain learning addresses the efficient solution of several tasks, defined on different domains. Each task is solved by a different network but all networks share a set of fixed layers **F**, which contain the majority of network parameters. These are complemented by small task-specific adaptation layers **A**.

Under the current deep learning paradigm, this task switching is difficult to perform. The predominant strategy is to use a different CNN to solve each task. Since only a few models can be cached in the GPU, and moving models in and out of cache adds too much overhead to enable real-time task switching, there is a need for very efficient parameter sharing across tasks. The individual networks should share most of their parameters, which would always reside on the GPU. A remaining small number of task specific parameters would be switched per task. This problem is known as *multi-domain learning* (MDL) and has been addressed with the architecture of Figure 1 [34, 38]. This consists of set of *fixed* layers (denoted as '**F**') shared by all tasks and a set of task specific *adaptation* layers (denoted as '**A**') fine-tunned to each task. If the **A** layers are much smaller than the **F** layers, many models can be cached simultaneously. Ideally, the **F** layers should be pre-trained, e.g. on ImageNet, and used by all tasks without additional training, enabling the use of special purpose chips to implement the majority of the computations. While **A** layers would still require a processing unit, the small amount of computation could enable the use of a CPU, making it cost-effective to implement each network on the device itself.

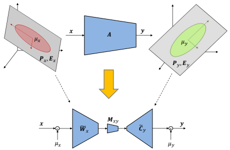

Figure 2: Covariance normalization. Each adaptation layer **A** is approximated by three transformations: $\tilde{\mathbf{W}}_x$, which implements a projection onto the PCA space of the input $\mathbf{x}$ (principal component matrix $\mathbf{P}_x$ and eigenvalue matrix $\mathbf{E}_x$), $\tilde{\mathbf{W}}_y$, which reconstructs the PCA space of the output $\mathbf{y}$ (matrices $\mathbf{P}_y$ and $\mathbf{E}_y$), and a mini-adaptation layer $\mathbf{M}_{xy}$.

In summary, MDL aims to maximize the performance of the network ecology while minimizing the ratio of task specific (**A**) to total parameters (both types **F** and **A**) per network. [34, 38] have shown that the architecture of Figure 1 can match the performance of fully fine-tuning each network in the ecology, even when **A** layers contain as few as 10% of the total parameters. In this work, we show that **A** layers can be substantially further shrunk, using a data-driven low-rank approximation. As illustrated in Figure 2, this is based on transformations that match the 2nd-order statistics of the **A** layer inputs and outputs. Given principal component analyzes (PCAs) of both input and output, the layer is approximated by a *recoloring transformation:* a projection into input PCA space, followed by a reconstruction into the output PCA space. By controlling the intermediate PCA dimensions, the method enables low-dimensional approximations of different input and output dimensions. To correct the mismatch (between PCA components) of two PCAs learned independently, a small *mini-adaptation* layer is introduced between the two PCA matrices, and fine-tunned on the target target.

Since the overall transformation generalizes *batch normalization*, the method is denoted *covariance normalization* (CovNorm). CovNorm is shown to outperform, with both theoretical and experimental arguments, purely geometric methods for matrix approximation, such as the singular value decomposition (SVD) [35], fine-tuning of the original **A** layers [34, 38], or adaptation based on batch nor-

malization [2]. It is also quite simple, requiring two PCAs and the finetuning of a very small mini-adaptation layer per $\mathbf{A}$ layer and task. Experimental results show that it can outperform full network fine-tuning while reducing $\mathbf{A}$ layers to as little as 0.53% of the total parameters. When all tasks can be learned together, $\mathbf{A}$ layers can be further reduced to 0.51% of the full model size. This is achieved by combining the individual PCAs into a global PCA model, of parameters shared by all tasks, and only fine-tunning mini-adaptation layers in a task specific manner.

2. Related work

MDL is a transfer learning problem, namely the transfer of a model trained on a *source* learning problem to an ecology of *target* problems. This makes it related to different types of transfer learning problems, which differ mostly in terms of input, or *domain,* and range space, or *task*.

Task transfer: Task transfer addresses the use of a model trained on a source task to the solution of a target task. The two tasks can be defined on the same or different domains. Task transfer is prevalent in deep learning, where a CNN pre-trained on a large source dataset, such as ImageNet, is usually fine-tunned [21] to a target task. While extremely effective and popular, full network fine-tunning changes most network parameters, frequently all. MDL addresses this problem by considering multiple target tasks and extensive parameter sharing between them.

Domain Adaptation: In domain adaptation, the source and target tasks are the same, and a model trained on a source domain is transfered to a target domain. Domain adaptation can be supervised, in which case labeled data is available for the target domain, or unsupervised, where it is not. Various strategies have been used to address these problems. Some methods seek the network parameters that minimize some function of the distance between feature distributions in the two domains [24, 4, 43]. Others introduce an adversarial loss that maximizes the confusion between the two domains [8, 45]. A few methods have also proposed to do the transfer at the image level, e.g. using GANs [11] to map source images into (labeled) target images, then used to learn a target classifier [3, 41, 14]. All these methods exploit the commonality of source and domain tasks to align source and target domains. This is unlike MDL, where source and target tasks are different. Nevertheless, some mechanisms proposed for domain adaptation can be used for MDL. For example, [5, 28] use a batch normalization layer to match the statistics of source and target data, in terms of means and standard deviation. This is similar to an early proposal for MDL [2]. We show that these mechanisms underperform covariance normalization.

Multitask learning: Multi-task learning [6, 49] addresses the solution of multiple tasks by the same model. It assumes that all tasks have the same visual domain. Popular examples include classification and bounding box regression in object detection [9, 37], joint estimation of surface normals and depth [7] or segmentation [29], joint representation in terms of attributes and facial landmarks [50, 33], among others. Multitask learning is sometimes also used to solve auxiliary tasks that strengthen performance of a task of interest, e.g. by accounting for context [10], or representing objects in terms of classes and attributes [15, 29, 30, 25]. Recently, there have been attempts to learn models that solve many problems jointly [18, 19, 48].

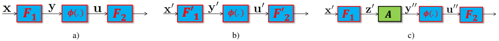

Figure 3: a) original network, b) after fine-tuning, and c) with adaptation layer $\mathbf{A}$. In all cases, $\mathbf{W}_i$ is a weight layer and $\phi(.)$ a non-linearity.

Most multitask learning approaches emphasize the learning of the interrelationships between tasks. This is frequently accomplished by using a single network, combining domain agnostic lower-level network layers with task specific network heads and loss functions [50, 7, 10, 15, 37, 19], or some more sophisticated forms of network branching [25]. The branching architecture is incompatible with MDL, where each task has its own input, different from those of all other tasks. Even when multi-task learning is addressed with multiple tower networks, the emphasis tends to be on inter-tower connections, e.g. through cross-stitching [29, 17]. In MDL, such connections are not feasible, because different networks can join the ecology of Figure 1 asynchronously, as devices are turned on and off.

Lifelong learning: Lifelong learning aims to learn multiple tasks sequentially with a shared model. This can be done by adapting the parameters of a network or adapting the network architecture. Since training data is discarded upon its use, constraints are needed to force the model to remember what was previously learned. Methods that only change parameters either use the model output on previous tasks [23], previous parameters values [22], or previous network activations [44] to regularize the learning of the target task. They are very effective at parameter sharing, since a single model solves all tasks. However, this model is not optimal for any specific task, and can perform poorly on all tasks, depending on the mismatch between source and target domains [36]. We show that they can significantly underperform MDL with CovNorm. Methods that adapt the network architecture usually add a tower per new task [40, 1]. These methods have much larger complexity than MDL, since several towers can be needed to solve a single task [40], and there is no sharing of fixed layers across tasks.

Multi-domain learning: This work builds on previous attempts at MDL, which have investigated different architectures for the adaptation layers of Figure 1. [2] used a BN layer [16] of parameters tunned per task. While performing well on simple datasets, this does not have enough degrees of freedom to support transfer of large CNNs across different domains. More powerful architectures were proposed by [38], who used a 1×1 convolutional layer and [34], who proposed a ResNet-style residual layer, known as a residual adaptation (RA) module. These methods were shown to perform surprisingly well in terms of recognition accuracy, equaling or surpassing the performance of full network fine tunning, but can still require a substantial number of adaptation parameters, typically 10% of the network size. [35] addressed this problem by combining adapters of multiple tasks into a large matrix, which is approximated with an SVD. This is then fine-tuned on each target dataset. Compressing adaptation layers in this way was shown to reduce adaptive parameter counts to approximately half of [34]. However, all tasks have to be optimized simultaneously. We show that CovNorm enables a further ten-fold reduction in adaptation layer parameters, without this limitation, although some additional gains are possible with joint optimization.

3. MDL by covariance normalization

In this section, we introduce the CovNorm procedure for MDL with deep networks.

3.1. Multi-domain learning

Figure 3 a) motivates the use of $\mathbf{A}$ layers in MDL. The figure depicts two fixed weight layers, $\mathbf{F}_1$ and $\mathbf{F}_2$, and a non-linear layer $\phi(.)$ in between. Since the fixed layers are pre-trained on a *source* dataset $\mathcal{S}$, typically ImageNet, all weights are optimized for the source statistics. For standard losses, such as cross entropy, this is a maximum likelihood (ML) procedure that matches $\mathbf{F}_1$ and $\mathbf{F}_2$ to the statistics of activations $\mathbf{x}, \mathbf{y}$ and $\mathbf{u}$ in $\mathcal{S}$. However, when the CNN is used on a different *target* domain, the statistics of these variables change and $\mathbf{F}_1, \mathbf{F}_2$ are no longer an ML solution. Hence, the network is sub-optimal and must be finetunned on a target dataset $\mathcal{T}$. This is denoted full network finetuning and converts the network into an ML solution for $\mathcal{T}$, with the outcome of Figure 3 b). In the target domain, the intermediate random variables become $\mathbf{x}'$, $\mathbf{y}'$, and $\mathbf{u}'$ and the weights are changed accordingly, into $\mathbf{F}'_1$ and $\mathbf{F}'_2$.

While very effective, this procedure has two drawbacks, which follow from updating all weights. First, it can be computationally expensive, since modern CNNs have large weight matrices. Second, because the weights $\mathbf{F}'_i$ are not optimal for $\mathcal{S}$, i.e. the CNN forgets the source task, there is a need to store and implement two CNNs to solve both tasks. This is expensive in terms of storage and computation and

increases the complexity of managing the network ecology. A device that solves both tasks must store two CNNs and load them in and out of cache when it switches between the tasks. These problems are addressed by the MDL architecture of Figure 1, which is replicated in greater detail on Figure 3 c). It introduces an *adaptation layer* $\mathbf{A}$ and fine-tunes this layer only, leaving $\mathbf{F}_1$ and $\mathbf{F}_2$ unchanged. In this case, the statistics of the input are still those of $\mathbf{x}'$, but the distributions along the network are now those of $\mathbf{z}', \mathbf{y}''$, and $\mathbf{u}''$. Since $\mathbf{F}_1$ is fixed, nothing can be done about $\mathbf{z}'$. However, the fine-tuning of $\mathbf{A}$ encourages the statistics of $\mathbf{y}''$ to match those of $\mathbf{y}'$, i.e. $\mathbf{y}'' = \mathbf{y}'$ and thus $\mathbf{u}'' = \mathbf{u}'$. Even if $\mathbf{A}$ cannot match statistics exactly, the mismatch is reduced by repeating the procedure in subsequent layers, e.g. introducing a second $\mathbf{A}$ layer after $\mathbf{F}_2$, and optimizing adaptation matrices as a whole.

3.2. Adaptation layer size

Obviously, MDL has limited interest if $\mathbf{A}$ has size similar to $\mathbf{F}_1$. In this case, each domain has as many adaptation parameters as the original network, all networks have twice the size, task switching is complex, and training complexity is equivalent to full fine tunning of the original network. On the other hand, if $\mathbf{A}$ is much smaller than $\mathbf{F}_1$, MDL is computationally light and task-switching much more efficient. In summary, the goal is to introduce an adaptation layer $\mathbf{A}$ as *small as possible,* but still powerful enough to *match the statistics* of $\mathbf{y}'$ and $\mathbf{y}''$. A simple solution is to make $\mathbf{A}$ a batch normalization layer [16]. This was proposed in [2] but, as discussed below, is not effective. To overcome this problem, [38] proposed a linear transformation $\mathbf{A}$ and [34] adopted the residual structure of [13], i.e. an adaptation layer $\mathbf{T} = (\mathbf{I} + \mathbf{A})$. To maximize parameter savings, $\mathbf{A}$ was implemented with a 1×1 convolutional layer in both cases.

This can, however, still require a non-trivial number of parameters, especially in upper network layers. Let $\mathbf{F}_1$ convolve a bank of d filters of size $k \times k \times l$ with l feature maps. Then, $\mathbf{F}_1$ has size dk^2l, $\mathbf{y}$ is d dimensional, and $\mathbf{A}$ a $d \times d$ matrix. Since in upper network layers k is usually small and $d > l$, $\mathbf{A}$ can be only marginally smaller than $\mathbf{F}_1$. [35] exploited redundancies across tasks to address this problem, creating a matrix with the $\mathbf{A}$ layer parameters of multiple tasks and computing a low-rank approximation of this matrix with an SVD. The compression achieved with this approximation is limited, because the approximation is purely geometric, not taking into account the statistics of $\mathbf{z}'$ and $\mathbf{y}'$. In this work, we propose a more efficient solution, motivated by the interpretation of $\mathbf{A}$ as converting the statistics of $\mathbf{z}'$ into those of $\mathbf{y}'$. It is assumed that the fine-tuning of $\mathbf{A}$ produces an output variable $\mathbf{y}''$ whose statistics match those of $\mathbf{y}'$. This could leverage adaptation layers in other layers of the network, but that is not important for the discussion that follows. The only assumption is that $\mathbf{y}'' = \mathbf{y}'$. The goal is to replace $\mathbf{A}$ by a simpler matrix that maps $\mathbf{z}'$ into $\mathbf{y}'$. For simplicity, we drop the primes and notation of Figure 3 in what follows, considering the problem of matching statistics between input $\mathbf{x}$ and output $\mathbf{y}$ of a matrix $\mathbf{A}$.

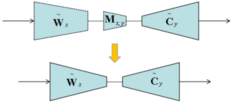

Figure 4: Top: covnorm approximates adaptation layer $\mathbf{A}$ by a sequence of whitening $\tilde{\mathbf{W}}_x$, mini-adaptation $\mathbf{M}_{x,y}$, and coloring $\tilde{\mathbf{C}}_y$ operations. Bottom: after covnorm, the mini adaptation layer can be absorbed into $\tilde{\mathbf{W}}_x$ (shown in the figure) or $\tilde{\mathbf{C}}_y$.

3.3. Geometric approximations

One possibility is to use a purely geometric solution [35]. Geometrically, the closest low rank approximation of a matrix $\mathbf{A}$ is given by the SVD, $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$. More precisely, the minimum Frobenius norm approximation $\tilde{\mathbf{A}} = \arg\min_{\{\mathbf{B}|rank(\mathbf{B})=r\}} ||\mathbf{A} - \mathbf{B}||_F^2$, where $r < rank(\mathbf{A})$, is $\tilde{\mathbf{A}} = \mathbf{U}\tilde{\mathbf{S}}\mathbf{V}^T$ where $\tilde{\mathbf{S}}$ contains the r largest singular values of $\mathbf{A}$. This can be written as $\tilde{\mathbf{A}} = \mathbf{C}\mathbf{W}$, where $\mathbf{C} = \mathbf{U}\sqrt{\tilde{\mathbf{S}}}$ and $\mathbf{W} = \sqrt{\tilde{\mathbf{S}}}\mathbf{V}^T$. If $A \in \mathbb{R}^{d \times d}$, these matrices have a total of $2rd$ parameters. An even simpler solution is to define $\mathbf{C} \in \mathbb{R}^{d \times r}$ and $\mathbf{W} \in \mathbb{R}^{r \times d}$, replace $\mathbf{A}$ by their product in Figure 3 c), and fine-tune the two matrices instead of $\mathbf{A}$. We denote this as the *fine-tunned approximation* (FTA). These approaches are limited by their purely geometric nature. Note that d is determined by the source model (output dimension of $\mathbf{F}_1$) and fixed. On the other hand, the dimension r should depend on the target dataset $\mathcal{T}$. Intuitively, if $\mathcal{T}$ is much smaller than $\mathcal{S}$, or if the target task is much simpler, it should be possible to use a smaller r than otherwise. There is also no reason to believe that a single r, or even a single ratio r/d, is suitable for all network layers. While r could be found by cross-validation, this becomes expensive when there are multiple adaptation layers throughout the CNN. We next introduce an alternative, data driven, procedure that bypasses these difficulties.

3.4. Covariance matching

Assume that, as illustrated in Figure 2, $\mathbf{x}$ and $\mathbf{y}$ are Gaussian random variables of means μ_x, μ_y and covariances $\mathbf{\Sigma}_x, \mathbf{\Sigma}_y$, respectively, related by $\mathbf{y} = \mathbf{A}\mathbf{x}$. Let the covariances have eigendecomposition

$$\mathbf{\Sigma}_x = \mathbf{P}_x\mathbf{E}_x\mathbf{P}_x^T \qquad \mathbf{\Sigma}_y = \mathbf{P}_y\mathbf{E}_y\mathbf{P}_y^T \tag{1}$$

where $\mathbf{P}_x, \mathbf{P}_y$ contain eigenvectors as columns and $\mathbf{E}_x, \mathbf{E}_y$ are diagonal eigenvalue matrices. We refer to the triplet $\mathcal{P}_x = (\mathbf{P}_x, \mathbf{E}_x, \mu_x)$ as the PCA of $\mathbf{x}$. Then, it is well known that the statistics of $\mathbf{x}$ and $\mathbf{y}$ are related by

$$\mu_y = \mathbf{A}\mu_x \qquad \mathbf{\Sigma}_y = \mathbf{A}\mathbf{\Sigma}_x\mathbf{A}^T \tag{2}$$

and, combining (1) and (2), $\mathbf{P}_y\mathbf{E}_y\mathbf{P}_y^T = \mathbf{A}\mathbf{P}_x\mathbf{E}_x\mathbf{P}_x^T\mathbf{A}^T$. This holds when $\mathbf{P}_y\sqrt{\mathbf{E}_y} = \mathbf{A}\mathbf{P}_x\sqrt{\mathbf{E}_x}$ or, equivalently,

$$\begin{aligned} \mathbf{A} &= \mathbf{P}_y\sqrt{\mathbf{E}_y}\sqrt{\mathbf{E}_x^{-1}}\mathbf{P}_x^T. & (3) \\ &= \mathbf{C}_y\mathbf{W}_x & (4) \end{aligned}$$

where $\mathbf{W}_x = \sqrt{\mathbf{E}_x^{-1}}\mathbf{P}_x^T$is the "whitening matrix" of $\mathbf{x}$ and $\mathbf{C}_y = \mathbf{P}_y\sqrt{\mathbf{E}_y}$the "coloring matrix" of $\mathbf{y}$. It follows that (2) holds if $\mathbf{y} = \mathbf{A}\mathbf{x}$ is implemented with a sequence of two operations. First, $\mathbf{x}$ is mapped into a variable $\mathbf{w}$ of zero mean and identity covariance, by defining

$$\mathbf{w} = \mathbf{W}_x(\mathbf{x} - \mu_x). \tag{5}$$

Second, $\mathbf{w}$ is mapped into $\mathbf{y}$ with

$$\mathbf{y} = \mathbf{C}_y\mathbf{w} + \mu_y. \tag{6}$$

In summary, for Gaussian $\mathbf{x}$, the effect of $\mathbf{A}$ is simply the combination of a whitening of $\mathbf{x}$ followed by a colorization with the statistics of $\mathbf{y}$.

3.5. Covariance normalization

The interpretation of the adaptation layer as a recoloring operation (whitening + coloring) sheds light on the number of parameters effectively needed for the adaptation, since the PCAs $\mathcal{P}_x, \mathcal{P}_y$ capture the *effective* dimensions of $\mathbf{x}$ and $\mathbf{y}$. Let k_x (k_y) be the number of eigenvalues significantly larger than zero in $\mathbf{E}_x$ ($\mathbf{E}_y$). Then, the whitening and coloring matrices can be approximated by

$$\tilde{\mathbf{W}}_x = \sqrt{\tilde{\mathbf{E}}_x^{-1}}\tilde{\mathbf{P}}_x^T \qquad \tilde{\mathbf{C}}_y = \tilde{\mathbf{P}}_y\sqrt{\tilde{\mathbf{E}}_y} \tag{7}$$

where $\tilde{\mathbf{E}}_x \in \mathbb{R}^{k_x \times k_x}$ ($\tilde{\mathbf{E}}_y \in \mathbb{R}^{k_y \times k_y}$) contains the non-zero eigenvalues of $\mathbf{\Sigma}_x$ ($\mathbf{\Sigma}_y$), and $\tilde{\mathbf{P}}_x \in \mathbb{R}^{d \times k_x}$ ($\tilde{\mathbf{P}}_y \in \mathbb{R}^{d \times k_y}$) the corresponding eigenvectors. Hence, $\mathbf{A}$ is well approximated by a pair of matrices ($\tilde{\mathbf{W}}_x$, $\tilde{\mathbf{C}}_y$) totaling $d(k_x + k_y)$ parameters.

On the other hand, the PCAs are only defined up to a permutation, which assigns an ordering to eigenvalues/eigenvectors. When the input and output PCAs are computed independently, the principal components may not be aligned. This can be fixed by introducing a permutation matrix between $\mathbf{C}_y$ and $\mathbf{W}_x$ in (4). The assumption that all distributions are Gaussian also only holds approximately in real networks. To account for all this, we augment the recoloring operation with a mini-adaptation layer $\mathbf{M}_{x,y}$ of size $k_x \times k_y$. This leads to the covariance normalization (CovNorm) transform

$$\tilde{\mathbf{y}} = \tilde{\mathbf{C}}_y\mathbf{M}_{x,y}\tilde{\mathbf{W}}_x(\mathbf{x} - \mu_x) + \mu_y, \tag{8}$$

Algorithm 1: Covariance Normalization

```
Data: source $\mathcal{S}$ and target $\mathcal{T}$
Insert an adaptation layer $\mathbf{A}$ on a CNN trained on $\mathcal{S}$
  and fine-tune $\mathbf{A}$ on $\mathcal{T}$.
Store the layer input and output PCAs $\mathcal{P}_x, \mathcal{P}_y$,
  select the $k_x, k_y$ non-zero eigenvalues and
  corresponding eigenvectors from each PCA, and
  compute $\tilde{\mathbf{C}}_y, \tilde{\mathbf{W}}_x$ with (7).
add mini-adaptation layer $\mathbf{M}_{x,y}$ and replace $\mathbf{A}$ by
  (8). Note that, as usual, the constant
  $\tilde{\mathbf{C}}_y\mathbf{M}_{x,y}\tilde{\mathbf{W}}_x\mu_x + \mu_y$ can be implemented with a
  vector of biases.
fine-tune $\mathbf{M}_{x,y}$ with $\tilde{\mathbf{W}}_x$ and $\tilde{\mathbf{C}}_y$ on $\mathcal{T}$ and absorb
  $\mathbf{M}_{x,y}$ into the larger of $\tilde{\mathbf{W}}_x$ and $\tilde{\mathbf{C}}_y$.
```

where $\mathbf{M}_{x,y}$ is learned by fine-tuning on the target dat
$\mathcal{T}$. Beyond improving recognition performance, this
the advantage of further parameters savings. The di
implementation of (8) increases the parameter coun
$d(k_x + k_y) + k_xk_y$. However, after fine-tuning, $\mathbf{M}_{x,y}$
be absorbed into one of the two other matrices , as show
Figure 4. When $k_x > k_y$, $\mathbf{M}_{x,y}\tilde{\mathbf{W}}_x$ has dimension k_y
and replacing the two matrices by their product reduces
total parameter count to $2dk_y$. In this case, we say
$\mathbf{M}_{x,y}$ is absorbed into $\tilde{\mathbf{W}}_x$. Conversely, if $k_x < k_y$, M
can be absorbed into $\tilde{\mathbf{C}}_y$. Hence, the total parameter co
is $2d\min(k_x, k_y)$. CovNorm is summarized in Algorith

3.6. The importance of covariance normalizatio

The benefits of covariance matching can be seen by c
parison to previously proposed MDL methods. Assu
first, that $\mathbf{x}$ and $\mathbf{y}$ consist of *independent* features. In
case, $\mathbf{P}_x, \mathbf{P}_y$ are identity matrices and (5)-(6) reduce to

$$y_i = \sqrt{e_{y,i}}\frac{x_i - \mu_{x,i}}{\sqrt{e_{x,i}}} + \mu_{y,i},$$

which is the batch normalization equation. Hence, C
Norm is a generalized form of the latter. There are, h
ever, important differences. First, there is no batch.
normalizing distribution $\mathbf{x}$ is now the distribution of the
ture responses of layer $\mathbf{F}_1$ on the target dataset $\mathcal{T}$. Seco
the goal is not to facilitate the learning of $\mathbf{F}_2$, but prod
a feature vector $\mathbf{y}$ with statistics matched to $\mathbf{F}_2$. This tu
out to make a significant difference. Since, in regular ba
normalization, $\mathbf{F}_2$ is allowed to change, it can absorb
initial mismatch with the independence assumption. Th
not the case for MDL, where $\mathbf{F}_2$ is *fixed.* Hence, (9) usu
fails, significantly underperforming (5)-(6).

Next, consider the geometric solution. Since CovN
reduces to the product of two tall matrices, e.g. $\mathbf{K}$
$\tilde{\mathbf{C}}_y\mathbf{M}_{x,y}$ and $\mathbf{L} = \tilde{\mathbf{W}}_x$ of size $d \times k_x$, it should be
sible to replace it with the fine-tuned approximation ba
on two matrices of this size. Here, there are two difficul

First, k_x is not known in the absence of the PCA decompositions. Second, in our experience, even when k_x is set to the value used by PCA, the fine-tuned approximation does not work. As shown in the experimental section, when the matrices are initialized with Gaussian weights, performance can decrease significantly. This is an interesting observation because $\mathbf{A}$ is itself initialized with Gaussian weights. It appears that a good initialization is more critical for the low-rank matrices.

Finally, CovNorm can be compared to the SVD, $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$. From (3), this holds whenever $\mathbf{V} = \mathbf{P}_x$, $\mathbf{S} = \sqrt{\mathbf{E}_y}\sqrt{\mathbf{E}_x^{-1}}$ and $\mathbf{U} = \mathbf{P}_y$. The problem is that the singular value matrix $\mathbf{S}$ conflates the variances of the input and output PCAs. The fact that $s_i = e_{y,i}/e_{x,i}$ has two important consequences. First, it is impossible to recover the dimensions k_x and k_y by inspection of the singular values. Second, the low-rank criteria of selecting the largest singular values is *not* equivalent to CovNorm. For example, the principal components of $\mathbf{x}$ with largest eigenvalues $e_{x,i}$ have the smallest singular values s_i. Hence, it is impossible to tell if singular vectors $\mathbf{v}_i$ of small singular values are the most important (PCA components of large variance for $\mathbf{x}$) or the least important (noise). Conversely, the largest singular values can simply signal the least important input dimensions. CovNorm eliminates this problem by explicitly selecting the important input and output dimensions.

3.7. Joint training

[35] considered a variant of MDL where the different tasks of Figure 1 are all optimized simultaneously. This is the same as assuming that a joint dataset $\mathcal{T} = \cup_i \mathcal{T}_i$ is available. For CovNorm, the only difference with respect to the single dataset setting is that the PCAs $\mathcal{P}_x, \mathcal{P}_y$ are now those of the joint data $\mathcal{T}$. These can be derived from the PCAs $\mathcal{P}_{x,i}, \mathcal{P}_{y,i}$ of the individual target datasets $\mathcal{T}_i$ with

$$\begin{aligned} \mu_{\mathcal{T}} &= \frac{1}{N}\sum_i N_i \mu_i \\ \mathbf{\Sigma}_{\mathcal{T}} &= \sum_i \frac{N_i}{N}(\mathbf{P}_i \mathbf{E}_i {\mathbf{P}_i}^T + \mu_i {\mu_i}^T)) - \mu_{\mathcal{T}} \mu_{\mathcal{T}}^T \end{aligned} \quad (10)$$

where N_i is the cardinality of $\mathcal{T}_i$. Hence, CovNorm can be implemented by finetuning $\mathbf{A}$ to each $\mathcal{T}_i$, storing the PCAs $\mathcal{P}_{x,i}, \mathcal{P}_{y,i}$, using (10) to reconstruct the covariance of $\mathcal{T}$, and computing the global PCA. When tasks are available sequentially, this can be done recursively, combining the PCA of all previous data with the PCA of the new data. In summary, CovNorm can be extended to any number of tasks, with constant storage requirements (a single PCA), and no loss of optimality. This makes it possible to define two CovNorm *modes*.

- *independent:* $\mathbf{A}$ layers of network i are adapted to target dataset $\mathcal{T}_i$. A PCA is computed for $\mathcal{T}_i$ and the mini-adaptation fine-tuned to $\mathcal{T}_i$. This requires $2d\min(k_x, k_y)$ task specific parameters (per layer) per dataset.
- *joint:* a global PCA is learned from $\mathcal{T}$ and $\tilde{\mathbf{C}}_y, \tilde{\mathbf{W}}_x$ shared across tasks. Only a mini-adaptation layer is fine-tuned per $\mathcal{T}_i$. This requires $\min(k_x, k_y)$ task-specific parameters (per layer) per dataset. All $\mathcal{T}_i$ must be available simultaneously.

The independent model is needed if, for example, the devices of Figure 1 are produced by different manufacturers.

4. Experiments

In this section, we present results for both the independent and joint CovNorm modes.

Dataset: [34] proposed the decathlon dataset for evaluation of MDL. However, this is a collection of relatively small datasets. While sufficient to train small networks, we found it hard to use with larger CNNs. Instead, we used a collection of seven popular vision datasets. **SUN 397** [47] contains 397 classes of scene images and more than a million images. **MITIndoor** [46] is an indoor scene dataset with 67 classes and 80 samples per class. **FGVC-Aircraft Benchmark** [26] is a fine-grained classification dataset of $10,000$ images of 100 types of airplanes. **Flowers102** [32] is a fine-grained dataset with 102 flower categories and 40 to 258 images per class. **CIFAR100** [20] contains $60,000$ tiny images, from 100 classes. **Caltech256** [12] contains $30,607$ images of 256 object categories, with at least 80 samples per class. **SVHN** [31] is a digit recognition dataset with 10 classes and more than $70,000$ samples. In all cases, images are resized to 224×224 and the training and testing splits defined by the dataset are used, if available. Otherwise, 75% is used for training and 25% for testing.

Implementation: In all experiments, fixed $\mathbf{F}$ layers were extracted from a source **VGG16** [42] model trained on ImageNet. This has convolution layers of dimensions ranging from 64 to 4096. In a set of preliminary experiments, we compared the MDL performance of the architecture of Figure 1 with these $\mathbf{F}$ layers and adaptation layers implemented with 1) a convolutional layer $\mathbf{A}$ of kernel size 1×1 [38], 2) the residual adapters $\mathbf{T} = \mathbf{B_2}(\mathbf{I} + \mathbf{A}\mathbf{B_1})$ of [34], where $\mathbf{B_1}$ and $\mathbf{B_2}$ are batch normalization layers and $\mathbf{A}$ as in 1), and 3) the parallel adapters of [35]. Since residual adapters produced the best results, we adopted this structure in all our experiments. However, CovNorm can be used with any of the other structures, or any other matrix $\mathbf{A}$. Note that $\mathbf{B}_1$ could be absorbed into $\mathbf{A}$ after fine-tuning but we have not done so, for consistency with [34].

In all experiments, fine-tuning used initial learning rate of 0.001, reduced by 10 when the loss stops decreasing. After fine-tuning the residual layer, features were extracted at the input and output of $\mathbf{A}$ and the PCAs $\mathcal{P}_x, \mathcal{P}_y$ computed and used in Algorithm 1. Principal components were selected by the explained variance criterion. Once the eigenvalues e_i were computed and sorted by decreasing magnitude, i.e. $e_1 \geq e_2 \geq \ldots \geq e_d$, the variance explained by the first i eigenvalues is $r_i = \frac{\sum_{k=1}^{i} e_i}{\sum_{k=1}^{d} e_i}$. Given a threshold t, the smallest index i^* such that $r_{i^*} > t$ was determined, and only the i^* first eigenvalues/eigenvectors were kept. This set the dimensions k_x, k_y (depending on whether the procedure was used on $\mathcal{P}_x$ or $\mathcal{P}_y$). Unless otherwise noted, we used $t = 0.99$, i.e. 99% of the variance was retained.

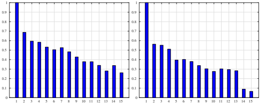

Figure 5: Ratio of effective dimensions (η) for different network layers. Left: MITIndoor. Right: CIFAR100.

Benfits of CovNorm: We start with some independent MDL experiments that provide insight on the benefits of CovNorm over previous MDL procedures. While we only report results for MITIndoor and CIFAR100, they are typical of all target datasets. Figure 5 shows the ratio $\eta = k_y/k_x$ of effective output to input dimensions, as a function of adaptation layer. It shows that the input of $\mathbf{A}$ typically contains more information than the output. Note that η is rarely one, is almost always less than 0.6, frequently smaller than 0.3, and smallest for the top network layers.

We next compared CovNorm to batch normalization (BN) [2], and geometric approximations based on the finetunned approximation (FTA) of Section 3.3. We also tested a mix of the geometric approaches (SVD+FTA), where $\mathbf{A}$ was first approximated by the SVD and the matrices $\mathbf{C}$, $\mathbf{W}$ finetuned on $\mathcal{T}$, and a mix of PCA and FTA (PCA+FTA), where the mini-adaptation layer $\mathbf{M}_{x,y}$ of CovNorm was removed and $\tilde{\mathbf{C}}_y, \tilde{\mathbf{W}}_x$ fine-tuned on $\mathcal{T}$, to minimize the PCA alignment problem. All geometric approximations were implemented with low-rank parameter values $r = d/2^i$, where d is the dimension of $\mathbf{x}$ or $\mathbf{y}$ and $i \in \{2, \ldots, 6\}$. For CovNorm, the explained variance threshold was varied in $[0.8, 0.995]$. Figure 6 shows recognition accuracies vs. the $\%$ of parameters. Here, 100% parameters corresponds the adaptation layers of [34]: a network with residual adapters whose matrix $\mathbf{A}$ is fine-tunned on $\mathcal{T}$. This is denoted RA and shown as an upper-bound. A second upper-bound is shown for full network fine tuning (FNFT). This requires $10\times$ more parameters than RA. BN, which requires close to zero parameters, is shown as a lower bound.

Several observations are possible. First, all geometric approximations underperform CovNorm. For comparable sizes, the accuracy drop of the best geometric method

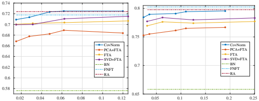

Figure 6: accuracy vs. % of parameters used for adaptation. Left: MITIndoor. Right: CIFAR100.

(SVD+FTA) is as large as 2%. This is partly due to the use of a constant low rank r throughout the network. This cannot match the effective, data-dependent, dimensions, which vary across layers (see Figure 5). CovNorm eliminates this problem. We experimented with heuristics for choosing variable ranks but, as discussed below (Figure 7), could not achieve good performance. Among the geometric approaches, SVD+FTA outperforms FTA, which has performance drops in most of datasets. It is interesting that, while $\mathbf{A}$ is fine-tuned with random initialization, the process is not effective for the low-rank matrices of FTA. In several datasets, FTA could not match SVD+FTA.

Even more surprising were the weaker results obtained when the random initialization was replaced by the two PCAs (PCA+FTA). Note the large difference between PCA+FTA and CovNorm (up to 4%), which differ by the mini-adaptation layer $\mathbf{M}_{x,y}$. This is explained by the alignment problem of Section 3.5. Interestingly, while mini-adaptation layers are critical to overcome this problem, they are as easy to fine-tune as $\mathbf{A}$. In fact, the addition of these layers (CovNorm) often outperformed the full matrix $\mathbf{A}$ (RA). In some datasets, like MITIndoor, with 4.8% of the parameters, CovNorm matched the performance of RA, Finally, as previously reported by [34], FNFT frequently underperformed RA. This is likely due to overfitting.

CovNorm vs SVD: Figure 7 provides empirical evidence for the vastly different quality of the approximations produced by CovNorm and the SVD. The figure shows a plot of the variance explained by the eigenvalues of the input and output distributions of an adaptation layer $\mathbf{A}$ and the corresponding plot for its singular values. Note how the PCA energy is packed into a much smaller number of coefficients than the singular value energy. This happens because PCA only accounts for the subspaces populated by data, restricting the low-rank approximation to these subspaces. Conversely, the geometric approximation must approximate the matrix behavior even outside of these subspaces. Note that the SVD is not only less efficient in identifying the important dimensions, but also makes it difficult to determine how many singular values to keep. This prevents the use of a layer-dependent number of singular values.

Comparison to previous methods: Table 1 summarizes the recognition accuracy and % of adaptation layer param-

Table 1: Classification accuracy and % of adaptation parameters (with respect to VGG size) per target dataset.

	FGVC	MITIndoor	Flowers	Caltech256	SVHN	SUN397	CIFAR100	average
FNFT	85.73% 100%	71.77%	95.67%	83.73%	96.41%	57.29%	80.45%	81.58% 100%
Independent learning								
BN [2]	43.6% 0%	57.6%	83.07%	73.66%	91.1%	47.04%	64.8%	65.83% 0%
LwF[23]	66.25% 0%	**73.43%**	89.12%	80.02%	44.13%	52.85%	72.94%	68.39% 0%
RA [34]	88.92% 10%	72.4%	96.43%	84.17%	96.13%	57.38%	**79.55%**	82.16% 10%
SVD+FTA	89.07% 5%	71.66%	95.67%	84.46%	96.04%	57.12%	78.28%	81.75% 5%
FTA	87.31% 5%	70.26%	95.43%	83.82%	95.96%	56.43%	78.23%	81.06% 5%
CovNorm	**88.98%** 0.34%	72.51% 0.62%	**96.76%** 0.35%	**84.75%** 0.46%	**96.23%** 0.13%	**57.97%** 0.71%	79.42% 1.1%	**82.37%** 0.53%
Joint learning								
SVD [35]	88.98% 5%	71.7%	96.37%	83.63%	96%	56.58%	78.26%	81.65% 5%
CovNorm	**88.99%** 0.51%	**73.0%**	**96.69%**	**84.77%**	**96.22%**	**58.2**	**79.22%**	**82.44%** 0.51%

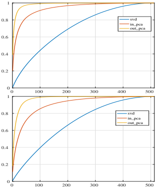

Figure 7: Variance explained by eigenvalues of a layer input and output, and similar plot for singular values. Left: MITIndoor. Right: CIFAR100.

	ImNet	Airc	C100	DPed	DTD	GTSR	Flwr	OGlt	SVHN	UCF	avg acc	S	#par
RA [34]	59.67%	61.87%	81.20%	93.88%	57.13%	97.57%	81.67%	89.62%	96.13%	**50.12%**	76.89%	2621	2
DAN [39]	57.74%	64.12%	80.07%	91.3%	56.54%	98.46%	**86.05%**	**89.67%**	96.77%	49.38%	77.01%	2851	2.17
Piggyback [27]	57.69%	65.29%	79.87%	96.99%	57.45%	97.27%	79.09%	87.63%	**97.24%**	47.48%	76.6%	2838	1.28
CovNorm	**60.37%**	**69.37%**	**81.34%**	**98.75%**	**59.95%**	**99.14%**	83.44%	87.69%	96.55%	48.92%	**78.55%**	**3713**	1.25

Table 2: Visual Decathlon results

eters vs. VGG model size (100% parameters), for various methods. All abbreviations are as above. Beyond MDL, we compare to learning without forgetting (LwF) [23] a lifelong method to learn a model that shares all parameters among datasets. The table is split into independent and joint MDL. For joint learning, CovNorm is implemented with (10) and compared to the SVD approach of [35].

Several observations can be made. First, CovNorm adapts the number of parameters to the task, according to its complexity and how different it is from the source (ImageNet). For the simplest datasets, such as the 10-digit class SVHN, adaptation can require as few as 0.13% task-specific parameters. Datasets that are more diverse but ImageNet-like, such as Caltech256, require around 0.46% parameters. Finally, larger adaptation layers are required by datasets that are either complex or quite different from ImageNet, e.g. scene (MITIndoor, SUN397) recognition tasks. Even here, adaptation requires less than 1% parameters. On average, CovNorm requires 0.53% additional parameters per dataset.

Second, for independent learning, all methods based on residual adapters significantly outperform BN and LwF. As shown by [34], RA outperforms FNFT. BN is uniformly weak, LwF performs very well on MITIndoor and Caltech256, but poorly on most other datasets. Third, CovNorm outperforms even RA, achieving higher recognition accuracy with 20× less parameters. It also outperforms SVD+FTA and FTA by $\approx 0.6\%$ and $\approx 1.3\%$, respectively, while reducing parameter sizes by a factor of ≈ 10. On a per-dataset basis, CovNorm outperforms RA on all datasets other than CIFAR100, and SVD+FTA and FTA on all of them. In all datasets, the parameter savings are significant. Fourth, for joint training, CovNorm is substantially superior to the SVD [35], with higher recognition rates in all datasets, gains of up to 1.62% (SUN397), and close to 10× less parameters. Finally, comparing independent and joint CovNorm, the latter has slightly higher recognition for a slightly higher parameter count. Hence, the two approaches are roughly equivalent.

Results on Visual Decathlon Table 2 presents results on the Decathlon challenge [34], composed of ten different datasets of small images (72×72). Models are trained with a combination of training and validation set and results obtained online. For fair comparison, we use the learning protocol of [34]. CovNorm achieves state of the art performance in terms of classification accuracy, parameter size, and decathlon score $\mathbf{S}$.

5. Conclusion

CovNorm is an MDL technique of very simple implementation. When compared to previous methods, it dramatically reduces the number of adaptation parameters without loss of recognition performance. It was used to show that large CNNs can be “recycled” across problems as diverse as digit, object, scene, or fine-grained classes, with no loss, by simply tuning 0.5% of their parameters.

6. Acknowledgment

This work was partially funded by NSF awards IIS-1546305 and IIS-1637941, a GRO grant from Samsung, and NVIDIA GPU donations.

References

[1] R. Aljundi, P. Chakravarty, and T. Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *CVPR*, pages 7120–7129, 2017.

[2] H. Bilen and A. Vedaldi. Universal representations: The missing link between faces, text, planktons, and cat breeds. *arXiv preprint arXiv:1701.07275*, 2017.

[3] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, page 7, 2017.

[4] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan. Domain separation networks. In *Advances in Neural Information Processing Systems*, pages 343–351, 2016.

[5] F. M. Carlucci, L. Porzi, B. Caputo, E. Ricci, and S. R. Bulò. Autodial: Automatic domain alignment layers. In *ICCV*, pages 5077–5085, 2017.

[6] R. Caruana. Multitask learning. In *Learning to learn*, pages 95–133. Springer, 1998.

[7] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2650–2658, 2015.

[8] Y. Ganin and V. Lempitsky. Unsupervised domain adaptation by backpropagation. *International Conference in Machine Learning*, 2014.

[9] R. Girshick. Fast r-cnn. *arXiv preprint arXiv:1504.08083*, 2015.

[10] G. Gkioxari, R. Girshick, and J. Malik. Contextual action recognition with r* cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1080–1088, 2015.

[11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[12] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. 2007.

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

[14] J. Hoffman, E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. A. Efros, and T. Darrell. Cycada: Cycle-consistent adversarial domain adaptation. *arXiv preprint arXiv:1711.03213*, 2017.

[15] J. Huang, R. S. Feris, Q. Chen, and S. Yan. Cross-domain image retrieval with a dual attribute-aware ranking network. In *Proceedings of the IEEE international conference on computer vision*, pages 1062–1070, 2015.

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[17] B. Jou and S.-F. Chang. Deep cross residual learning for multitask visual recognition. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 998–1007. ACM, 2016.

[18] A. Kendall, Y. Gal, and R. Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. *arXiv preprint arXiv:1705.07115*, 3, 2017.

[19] I. Kokkinos. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *CVPR*, volume 2, page 8, 2017.

[20] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[21] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[22] S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *Advances in Neural Information Processing Systems*, pages 4655–4665, 2017.

[23] Z. Li and D. Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[24] M. Long, Y. Cao, J. Wang, and M. I. Jordan. Learning transferable features with deep adaptation networks. *International Conference in Machine Learning*, 2015.

[25] Y. Lu, A. Kumar, S. Zhai, Y. Cheng, T. Javidi, and R. S. Feris. Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. In *CVPR*, volume 1, page 6, 2017.

[26] S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013.

[27] A. Mallya, D. Davis, and S. Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 67–82, 2018.

[28] M. Mancini, L. Porzi, S. R. Bulò, B. Caputo, and E. Ricci. Boosting domain adaptation by discovering latent domains. *arXiv preprint arXiv:1805.01386*, 2018.

[29] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert. Cross-stitch Networks for Multi-task Learning. In *CVPR*, 2016.

[30] P. Morgado and N. Vasconcelos. Semantically consistent regularization for zero-shot recognition. In *CVPR*, volume 9, page 10, 2017.

[31] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5, 2011.

[32] M.-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *Computer Vision, Graphics & Image Processing, 2008. ICVGIP'08. Sixth Indian Conference on*, pages 722–729. IEEE, 2008.

[33] R. Ranjan, V. M. Patel, and R. Chellappa. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[34] S.-A. Rebuffi, H. Bilen, and A. Vedaldi. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, pages 506–516, 2017.

[35] S.-A. Rebuffi, H. Bilen, and A. Vedaldi. Efficient parametrization of multi-domain deep neural networks. *arXiv preprint arXiv:1803.10082*, 2018.

[36] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert. icarl: Incremental classifier and representation learning. In *Proc. CVPR*, 2017.

[37] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1137–1149, 2017.

[38] A. Rosenfeld and J. K. Tsotsos. Incremental learning through deep adaptation. *arXiv preprint arXiv:1705.04228*, 2017.

[39] A. Rosenfeld and J. K. Tsotsos. Incremental learning through deep adaptation. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[40] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[41] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb. Learning from simulated and unsupervised images through adversarial training. In *CVPR*, volume 2, page 5, 2017.

[42] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[43] B. Sun and K. Saenko. Deep coral: Correlation alignment for deep domain adaptation. In *European Conference on Computer Vision*, pages 443–450. Springer, 2016.

[44] A. R. Triki, R. Aljundi, M. B. Blaschko, and T. Tuytelaars. Encoder based lifelong learning. *IEEE Conference Computer Vision and Pattern Recognition*, 2017.

[45] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell. Adversarial discriminative domain adaptation. In *Computer Vision and Pattern Recognition (CVPR)*, volume 1, page 4, 2017.

[46] M. Valenti, B. Bethke, D. Dale, A. Frank, J. McGrew, S. Ahrens, J. P. How, and J. Vian. The mit indoor multi-vehicle flight testbed. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 2758–2759. IEEE, 2007.

[47] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 3485–3492. IEEE, 2010.

[48] A. R. Zamir, A. Sax, W. Shen, L. Guibas, J. Malik, and S. Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3712–3722, 2018.

[49] Y. Zhang and Q. Yang. A survey on multi-task learning. *arXiv preprint arXiv:1707.08114*, 2017.

[50] Z. Zhang, P. Luo, C. C. Loy, and X. Tang. Facial landmark detection by deep multi-task learning. In *European Conference on Computer Vision*, pages 94–108. Springer, 2014.