The Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI-21)

Towards Reusable Network Components by Learning Compatible Representations

Michael Gygli, Jasper Uijlings, Vittorio Ferrari
Google Research
gyglim@google.com, jrru@google.com, vittoferrari@google.com

Abstract

This paper proposes to make a first step towards compatible and hence reusable network components. Rather than training networks for different tasks independently, we adapt the training process to produce network components that are compatible across tasks. In particular, we split a network into two components, a features extractor and a target task head, and propose various approaches to accomplish compatibility between them. We systematically analyse these approaches on the task of image classification on standard datasets. We demonstrate that we can produce components which are directly compatible without any fine-tuning or compromising accuracy on the original tasks. Afterwards, we demonstrate the use of compatible components on three applications: Unsupervised domain adaptation, transferring classifiers across feature extractors with different architectures, and increasing the computational efficiency of transfer learning.

1 Introduction

In computer vision we often train a different neural network for each task, where reuse of previously learnt knowledge typically remains limited to pre-training on ImageNet (ILSVRC-12) (Russakovsky et al. 2015). However, human knowledge is composable and reusable (Tenenbaum et al. 2011). Therefore it seems prudent to give neural networks these properties too. Similar to what humans do, computer vision methods should reuse and transfer from previously acquired knowledge in the form of previously trained models (Zamir et al. 2018; Ngiam et al. 2018; Dwivedi and Roig 2019; Achille et al. 2019; Kontogianni et al. 2020). For example, when a model can recognize cars in daylight, this knowledge should help recognizing cars by night through domain adaptation (Ben-David et al. 2010; Shu et al. 2018). In addition, when a model expands its knowledge, *e.g.* through more training examples or by learning a new concept, these improvements should be easily transferable to related tasks.

We believe that a general way to achieve network reusability is to build a large library of *compatible components* which are specialized for different tasks. For example, some would extract features from RGB images, depth images, or optical flow fields. Other components could use these features to classify animals, localize cars, segment roads, or estimate human body poses. The compatibility of the components would make it easy to mix and match them into a highly performing model for the task at hand. Besides domain adaptation and transfer learning, this would also enable training a single classifier which can be deployed on various devices, each with its own hardware-specific backbone network. We make a first step in the direction of reusable components by devising a training procedure to make the feature representations learnt on different tasks become compatible, without any post-hoc fine-tuning. On the long term, we envisage a future where the practice of building computer vision models will mature into a state similar to the car manufacturing or building construction industries: with a large pool of high-quality and functionally well-defined compatible parts that a designer can conveniently recombine into more complex models tailored to new tasks. The compatibility of components saves the designer the effort to make them work together in a new combination, so they are free to focus on designing ever more complex models.

Our quest for reusable components is related to the question of how similar the representations of independently trained networks are, when they are trained on similar data (Kornblith et al. 2019; Lenc and Vedaldi 2019; Li et al. 2016b; Lu et al. 2018; Mehrer, Kriegeskorte, and Kietzmann 2018; Morcos, Raghu, and Bengio 2018; Wang et al. 2018). Instead of such a post-hoc analysis, we make a first step towards training neural network that are *directly compatible*, rather than only similar (*e.g.* in terms of feature correlation (Li et al. 2016b; Morcos, Raghu, and Bengio 2018)). For the purpose of this paper, we define components by splitting a neural network into two parts: a feature extractor and a target task head. We say two networks are *compatible* if we can recombine the feature extractor of one network with the task head of the other while still producing good predictions, directly without any fine-tuning after recombination (Fig. 1). When network components become perfectly compatible, they can be interchanged at no loss of accuracy. It is important to note that compatibility does not require learning *identical* mappings, as in feature distillation (Romero et al. 2015). As an extreme case, we could not distill the features from a network for street view images to a network for underwater images. Instead, by only requiring compatibility as defined above, each network can learn a feature extractor that

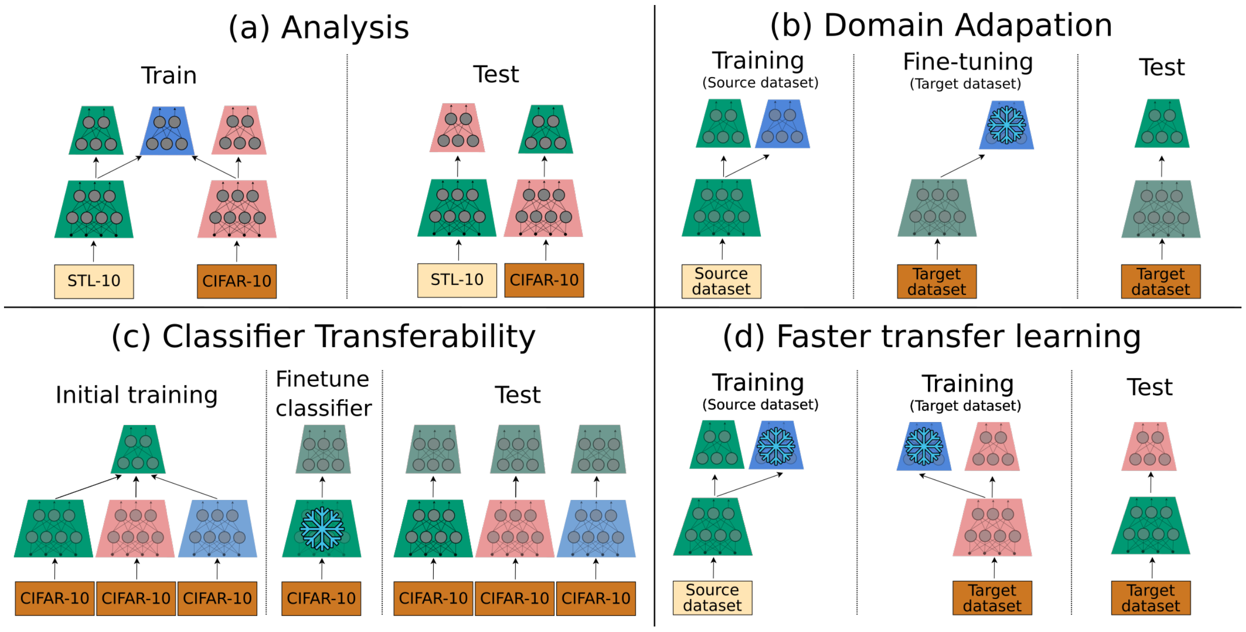


Figure 1: Experimental setups for the analysis and the applications. Our method enables recombining network components, which benefits domain adaptation, classifier transferability and efficient transfer learning.

is appropriate for its task.

Concretely, we introduce three ways to alter the training procedure of neural networks to encourage compatibility (Sec. 3): using a shared self-supervised auxiliary head which predicts rotation (Gidaris, Singh, and Komodakis 2018) (Sec 3.1), using a shared auxiliary head which discriminates common classes (Sec. 3.2), and starting training from identical initial weights (Sec. 3.3). We systematically analyse how well our methods make components compatible for the case of two image classification networks, one for CIFAR-10 and the other for STL-10 (Sec. 4). We also demonstrate that compatibility comes at no loss of accuracy on the original tasks. In Sec. 5 we apply our method to three diverse applications: unsupervised domain adaptation (Sec. 5.1), transferring pre-trained classifiers across networks with different architectures (Sec. 5.2) and increasing the computational efficiency of transfer learning (Sec. 5.3). These applications involve demonstrating compatibility between networks with different architectures, making several networks compatible at the same time, and testing on more complex datasets like CIFAR-100 and ILSVRC-12.

2 Related Work

In this section we discuss the relation of our work with existing methods. We provide a structured positioning in Tab.1.

Representational similarity analysis. Several works investigate whether neural networks learn different projections of the same high-level representations, when trained independently and potentially on different datasets (Li et al. 2016b; Lu et al. 2018; Mehrer, Kriegeskorte, and Kietzmann 2018; Morcos, Raghu, and Bengio 2018; Wang et al. 2018; Lenc and Vedaldi 2019; Kornblith et al. 2019; Shuai Tang 2020). Closest to our work, (Lenc and Vedaldi 2019) analyze representational similarity via the performance of networks after recombining their components. As they start from independently trained networks, they require adding a stitching layer and training the recombined network with a supervised loss for several epochs. We alter training, instead, so that the components are directly compatible and can be recombined, without the need for post-hoc optimization. This leads to features that are more similar than those of independently trained networks, as we show using (Li et al. 2016b).

Distillation & per-example feature alignment. Methods for feature alignment aim at training two or more networks so that they map a data sample to the same feature representation. Inspired by knowledge distillation (Buciluǎ, Caruana, and Niculescu-Mizil 2006; Hinton, Vinyals, and Dean 2015), feature distillation trains a network to approximate the feature activations of another network (Romero et al. 2015). Instead, multi-modal embeddings methods learn to map multiple views of the same example to a common representation (Frome et al. 2013; Socher et al. 2013; Karpathy and Fei-Fei 2015; Gupta, Hoffman, and Malik 2016; Wang, van de Weijer, and Herranz 2018). Examples include methods for image captioning, which train on image+caption pairs (Karpathy and Fei-Fei 2015), or mix-and-match networks (Wang, van de Weijer, and Herranz 2018), which map different modalities to the same representation, *e.g.* depth+RGB pairs. In contrast, our notion of compatibility does not require both networks to be trained from the same data, nor to see paired views of the same data. It is thus more generally applicable.

Multi-Task Learning (MTL). The goal of MTL is producing *a single* network which can solve multiple tasks. This

	Representational similarity	**Distillation**	**Multi-task learning**	**Continual learning**	**Unsupervised domain adaptation**	**Compatibility**
Analysis / application	Analysis	Application	Application	Application	Application	Both
Paired data	Yes	Yes	No	No	No	No
Identical output space	No	No	No	No	Yes	No
Identical architecture	No	No	Yes	Yes	Yes	No
Learns from / adapts existing model	Not applicable	Yes	No	Yes	Yes	No
Feature space alignment method	Post-hoc, Per-example	Per-example	Shared computational path	Various	Distribtion matching	Compatibility *w.r.t.* a head

Table 1: Positioning of compatibility *w.r.t.*related work.

is typically achieved with large architectures and partially sharing computational paths across tasks, *e.g.* (Misra et al. 2016; Kaiser et al. 2017; Maninis, Radosavovic, and Kokkinos 2019). In contrast, we train *different* networks whose components are compatible and our method regularizes the feature representation space only, without imposing architectural constraints. Furthermore, MTL requires access to all datasets of all tasks at the same time for training, which becomes burdensome in computation and engineering as the number of task grows, and might be infeasible due to licensing or privacy concerns. Instead, our incremental version (Sec. 3.4) does not require simultaneous access to all datasets, which we demonstrate experimentally in Sec. 4, 5.1 & 5.3.

Continual learning (CL). CL is typically formulated as the online version of MTL (Ruvolo and Eaton 2013; Rebuffi et al. 2017; Farquhar and Gal 2018; Chen and Liu 2018; Parisi et al. 2019). Thereby, the core challenge is to preserve compatibility between existing classification heads and a feature extractor that gets updated over time. This is often addressed by penalizing changing important weights (Kirkpatrick et al. 2017; Zenke, Poole, and Ganguli 2017; Aljundi et al. 2018), or changing predictions of the model on previous tasks (Li and Hoiem 2017; Shmelkov, Schmid, and Alahari 2017; Michieli and Zanuttigh 2019). Instead of just *preserving* compatibility between initially identical networks, we propose a method that *produces* compatible networks even if they have different architectures, are trained on different datasets, or start from different initializations (Sec. 4 & 5).

Unsupervised domain adaptation (UDA). The goal of UDA is to produce a model which works on the target domain, given a labeled source domain but only unlabeled data form the target domain. There are two dominant ways to approach this (Wang and Deng 2018; Zhang et al. 2018): (i) train one model that works on both the source and the target domain, *e.g.* (Tzeng et al. 2014; Ganin and Lempitsky 2014; Zhang et al. 2018; Kumar et al. 2018), who make features domain invariant through a domain discriminator; or (ii) train a model on the source domain and then adapt it to the target domain, *e.g.* (Saito, Ushiku, and Harada 2017; Zhang et al. 2018), who rely on pseudo labels. The latter is more general since it does not require access to the source and target dataset at the same time, and also works for non-conservative domain adaptation where a single classifier cannot perform well in both domains (Ben-David et al. 2010; Shu et al. 2018). In Sec. 5.3 we adopt this approach and show that the self-supervised version of our method improves UDA.

Transfer Learning (TL). The goal of TL is to improve results on a target task by reusing knowledge derived from a related source task. The current standard is to simply reuse the feature extractor of a model trained on ILSVRC-12 (Donahue et al. 2013; Sharif Razavian et al. 2014; Ren et al. 2015; He et al. 2017). However, what is the best source task depends on the target task (Zamir et al. 2018; Ngiam et al. 2018; Dwivedi and Roig 2019; Achille et al. 2019; Yan, Acuna, and Fidler 2020). In (Zamir et al. 2018), they proposed a computational framework to find good source tasks. But this is expensive: finding good source+target task combinations consumed 50'000 GPU hours to train 3000 transfer functions. Recently, (Dwivedi and Roig 2019; Achille et al. 2019) proposed methods to predict what tasks to transfer from. This allows to only transfer, fine-tune, and test the most promising feature extractors, thus saving computation. In Sec. 5.3 we demonstrate that we can reduce the amount of fine-tuning necessary to achieve good performance on the target task, which increases efficiency further.

3 Method

We consider neural networks formed by the combination of two components: A *feature extractor* $f(\cdot)$ and a *target task head* $h(\cdot)$, parameterized by $\mathbf{\Phi}$ and $\mathbf{\Theta}$, respectively. In standard supervised learning, one trains a neural network on task t by minimizing a task loss $\ell_t(h(f(\mathbf{x}_i; \mathbf{\Phi}_t); \mathbf{\Theta}_t), \mathbf{y}_i)$ over all examples $\mathbf{x}_i$ with label $\mathbf{y}_i$ in dataset $\mathcal{D}_t$. We denote a standard network trained on task t, using the feature extractor and target head of task t, as $n_{tt}(\mathbf{x}_i)$.

When independently training two networks on tasks a and b by minimizing their respective losses $\ell_a(\cdot)$ and $\ell_b(\cdot)$, the resulting networks are incompatible: Recombining their components into a new network $n_{ab}(\mathbf{x}_i) = h(f(\mathbf{x}_i; \mathbf{\Phi}_a); \mathbf{\Theta}_b)$ or $n_{ba}(\cdot)$ produces random or systematically wrong predictions (Sec. 4). This happens because the two feature extractors generally learn features responding to different image patterns, with different scaling of activation values, and even equivalent feature channels will appear in arbitrary orders (Lenc and Vedaldi 2019; Kornblith et al. 2019).

Compatibility. Our goal is to achieve compatibility between networks, directly after training. We define compatibility based on the performance of the recombined networks $n_{ab}(\cdot)$ and $n_{ba}(\cdot)$. When these network performs at chance level, we say that the components of $n_{aa}(\cdot)$ and $n_{bb}(\cdot)$ are *incompatible*. Instead, they are *compatible* when $n_{ab}(\cdot)$ and $n_{ba}(\cdot)$ directly output predictions that are significantly better than chance, without any fine-tuning after recombination. Generally, the recombined networks will not exceed the performance of the vanilla networks $n_{aa}(\cdot)$ and $n_{bb}(\cdot)$, trained and tested on their own task without recombining any component. Thus, we define this performance as the practical upper bound. When the recombined networks reach this upper bound, they are *perfectly compatible*, which allows to use their components interchangeably.

To achieve compatibility, we introduce some degree of dependency between the training processes of $n_{aa}(\cdot)$ and $n_{bb}(\cdot)$. Specifically, we encourage compatibility between the features produced by their extractors $f(\mathbf{x}_i; \mathbf{\Phi}_a)$ and $f(\mathbf{x}_i; \mathbf{\Phi}_b)$. As many different parameterizations of a neural network produce comparable performance (Choromanska et al. 2015; Lu et al. 2018), we hypothesize that we can make networks more compatible without decreasing the performance on their original task (confirmed in our experiments in Sec. 4).

Next, we introduce three different methods that encourage compatibility. For clarity of exposition, we describe the case for two networks, but our methods works with any number of networks (Sec. 5.2 & 5.3). Similarly, while we denote the model components with $f(\cdot)$ and $h(\cdot)$ for simplicity, our method also handles the case where networks n_{aa} and n_{bb} have a different architecture (Sec. 5.2).

3.1 Compatibility Through Self-Supervision (RP)

We propose to make components compatible via a generally applicable auxiliary task, based on a self-supervised objective. Self-supervision relies on supervised learning techniques, but the labels are created from the unlabelled input data itself. We adopt the approach of previous methods like (Noroozi and Favaro 2016; Doersch and Zisserman 2017; Gidaris, Singh, and Komodakis 2018). First, we transform an image $\mathbf{x}$ with $g(\mathbf{x}, \mathbf{s})$, a function which applies a transformation $\mathbf{s}$. Then, the task of the network is to predict what transformation was applied (its label).

To achieve compatibility, this auxiliary task has its own head s, but operates on the features produced by the extractors of the respective target tasks (Fig. 1a). Specifically, its prediction function is $s(f(\mathbf{x}; \mathbf{\Phi}_t); \mathbf{\Theta}_s)$, where $\mathbf{\Theta}_s$ are the parameters of the auxiliary task head. During training, we minimize the target task losses and the auxiliary task loss for both tasks:

$$\sum_{t\in\{a,b\}} \sum_{(\mathbf{x}_i,\mathbf{y}_i)\in\mathcal{D}_t} \Big[\ell_t\left(h\left(f\left(\mathbf{x}_i;\mathbf{\Phi}_t\right);\mathbf{\Theta}_t\right),\mathbf{y}_i\right) + \frac{1}{|\mathcal{S}|}\sum_{\mathbf{s}\in\mathcal{S}}\ell_s\left(h\left(f\left(g\left(\mathbf{x}_i,\mathbf{s}\right);\mathbf{\Phi}_t\right);\mathbf{\Theta}_s\right),\mathbf{s}\right)\Big] \tag{1}$$

where $\mathcal{S}$ is set of possible transformations that are applied, $\mathbf{\Theta}_s$ are the parameters of the auxiliary task head, and ℓ_s its associated loss. While there are target task parameters $\mathbf{\Phi}_t$ and $\mathbf{\Theta}_t$ specific to each task, we tie the auxiliary task parameters $\mathbf{\Theta}_s$ across tasks. This forces the feature extractors $f(\mathbf{x}_i; \mathbf{\Phi}_t)$ of each task t to produce features that are compatible with the same auxiliary task head. As we show in Sec. 4, this leads to feature extractors that are compatible more generally, allowing to recombine the feature extractor of one with the target task head of the other.

Choice of self-supervision task. Throughout this work we use rotation prediction (Gidaris, Singh, and Komodakis 2018). The input image is transformed by rotating it with an angle $\mathcal{S} = \{0°, 90°, 180°, 270°\}$ and the task is to classify which rotation angle was applied. For simplicity we refer to this method as *compatibility through rotation prediction* (RP), but any other self-supervised objective can be used here. We discuss considerations for choosing a suitable self-supervised task in the supplementary material.

Trade-offs. This compatibility method is very general. It only requires the shared self-supervised task to be both meaningful and non-trivial (Sun et al. 2019; Tschannen et al. 2019). While such a task can be defined on almost any dataset, the quality of the induced compatibility depends on how much the target task and the auxiliary task rely on the same features. In theory, a weakly related or orthogonal self-supervised auxiliary task could negatively affect the performance of the network on the target task. In practice though, it often improves performance (Zhai et al. 2019; Hénaff et al. 2019). Similarly, in our experiments we only observe positive effects on performance when adding rotation prediction.

3.2 Compatibility Through Discriminating Common Classes (DCC)

When tasks a, b have common classes, we can directly use these to achieve compatibility, rather than resorting to a self-supervised loss. Hence, we propose an auxiliary task head c, which discriminates among these common classes. Specifically, we minimize the following loss:

$$\sum_{t\in\{a,b\}} \sum_{(\mathbf{x}_i,\mathbf{y}_i)\in\mathcal{D}_t} \Big[\ell_t\left(h\left(f\left(\mathbf{x}_i;\mathbf{\Phi}_t\right);\mathbf{\Theta}_t\right),\mathbf{y}_i\right) + \ell_c\left(h\left(f\left(\mathbf{x}_i;\mathbf{\Phi}_t\right);\mathbf{\Theta}_c\right),\mathbf{y}_i\right)\cdot\mathbf{1}\left[\mathbf{y}_i\in\mathcal{C}\right]\Big] \tag{2}$$

where ℓ_c is the auxiliary task loss. It is computed only over examples in the set of common classes $\mathcal{C}$ ($\mathbf{1}$ is an indicator function returning 1 if its argument is true and 0 otherwise).

Trade-offs. While this method is expected to achieve high compatibility, it requires the target tasks to have common classes. Depending on the scenario, the target tasks might actually have few or even no common classes.

3.3 Compatibility Through Identical Initial Weights (IIW)

(Zhang, Bengio, and Singer 2019) demonstrated that for many layers in a trained network, resetting the weights of that layer to their initial values leads to a limited loss in accuracy. This suggests that the initialization defines a set of random projections which strongly shape the trained feature space. Hence, we propose to encourage compatibility simply by starting the loss minimization of both tasks from identical initial weights (IIW). For this method, we initialize using either identical *random* weights or identical *pre-trained* weights (Sec. 4).

Trade-offs. This method only works when both tasks have identical network architectures. Moreover, it only acts at the start of training, where it makes networks identical and thus perfectly compatible.

3.4 Training Schemes

For RP and DCC we consider two training schemes: *joint training* and *incremental training*.

In *joint training*, we minimize (1) (or (2)) by alternating between tasks a and b, each time minimizing the loss over a single minibatch. This resembles multi-task training (Doersch and Zisserman 2017; Maninis, Radosavovic, and Kokkinos 2019), but here each task has its own network, rather than having a single network with shared computation. By training jointly, both target tasks a, b influence the auxiliary task head parameters and use that head to solve the auxiliary task.

In *incremental training*, we first train the network n_{aa} by minimizing (1) (or (2)) over task a only. This also learns the parameters of the auxiliary task head. Later, we train the network n_{bb} on task b, but use the auxiliary task head with its parameters frozen. This encourages compatibility between n_{aa} and n_{bb}, without requiring both of them to be trained at the same time.

4 Analysis of Compatibility

We now analyse the induced compatibility of our methods introduced in Sec. 3: discriminating common classes (DCC), rotation prediction (RP), and initializing networks with identical initial weights (IIW).

Experimental setup. Fig. 1a illustrates our basic experimental setup when using a single auxiliary task (DCC or RP). As network architecture we use ResNet-56 (He et al. 2016), which consists of 3 stages with 9 ResNet blocks of two layer each. We split this into a feature extractor and target task head directly after the second stage (results for other splits are in the supp. material). We train one network on the CIFAR-10 (Krizhevsky 2009) train set and one on the STL-10 (Coates, Ng, and Lee 2011) train set. These datasets have 9 classes in common. For simplicity we mostly train networks jointly in this analysis. We also briefly explore incremental training (Sec. 3.4) which we use extensively in Sec. 5.

An important detail is that our network components use Batch Normalization (BN) (Ioffe and Szegedy 2015). At training time, BN normalizes the features in each batch to have zero mean and unit variance. At test time, features are normalized using aggregated statistics over the train set. However, DCC and RP encourage compatibility in the training regime of single-batch statistics, which may vary wildly per task. This makes the aggregated training statistics unreliable for any recombination of components. Therefore we use batch statistics at test time in all experiments (see supp. material for more discussion and alternatives)

As metric we define *recombination accuracy*: We recombine the CIFAR-10 feature extractor with the STL-10 classification head and measure accuracy on CIFAR-10 test, on the 9 common classes. We measure accuracy immediately after recombination, without any fine-tuning. We do the analogue for STL-10 and report the average over the two test sets.

Evaluation of methods to encourage compatibility. Fig. 2 shows recombination accuracy for our different compatibility measures. While the independently trained networks perform at chance level (10.6%), our proposed methods achieve good levels of compatibility: DCC works best (65.8%), followed by RP (50.1%). We note that RP is more generally applicable, since it does not not require any common classes. We investigate recombination accuracy as a function of the number of common classes in the supp. material.

Interestingly, there is even some compatibility between networks just by starting from identical initial weights and then separately minimizing the task loss on the two different datasets (IIW: 25.1%). The experiments also show that all three methods are complementary: using all methods together reaches 76.7% recombination accuracy.

As upper bound, we use the classical setting of training and testing a network on each task separately. For fairness, we give each network its own rotation prediction head which we found to improve results by 2.0%, but which does not encourage compatibility. This results in an upper bound of 85.6% accuracy. Given that the tasks are different, we consider IIW+RP+DCC (at 76.7%) to come rather close to this upper bound.

Importantly, we achieve compatibility without compromising accuracy on the original tasks. Using IIW+RP+DCC on the original networks and measuring accuracy on their own target tasks without recombination reaches 86.3%. This is slightly higher than our upper bound, likely because of beneficial regularization.

Starting from pre-trained models. It is common to start from a model pre-trained for ILSVRC-12 classification (Donahue et al. 2013; Ren et al. 2015; He et al. 2017). Therefore we repeat the above experiments but starting from models pre-trained for self-supervised rotation prediction on ILSVRC-12 (details in the supp. material).

For our IIW experiments we initialize both networks using the same pre-trained weights. When not using IIW, we initialize the two networks with *different* pre-trained weights.

Initializing networks using the same pre-trained weights (IIW) strongly encourages compatibility and already leads to a recombination accuracy of 74.3%. The strongest compatibility is achieved by combining IIW with DCC (82.7%).

Experiments not using IIW exhibit a counter-intuitive effect. For RP we reported 50.1% recombination accuracy when initializing the two networks using different *random* weights (Fig. 2). Now, when initializing using different *pre-trained* weights, recombination accuracy drops to 19.7%. Similarly, for DCC recombination accuracy drops from 65.8% to 52.3%. This suggests it is *harder* to make networks compatible after they are already independently (pre-)trained. Compatibility should thus be encouraged from the beginning of the training process.

Joint *vs.* incremental training. While so far we trained the two networks jointly, some practical applications require making a network compatible with an existing one (Sec. 5).

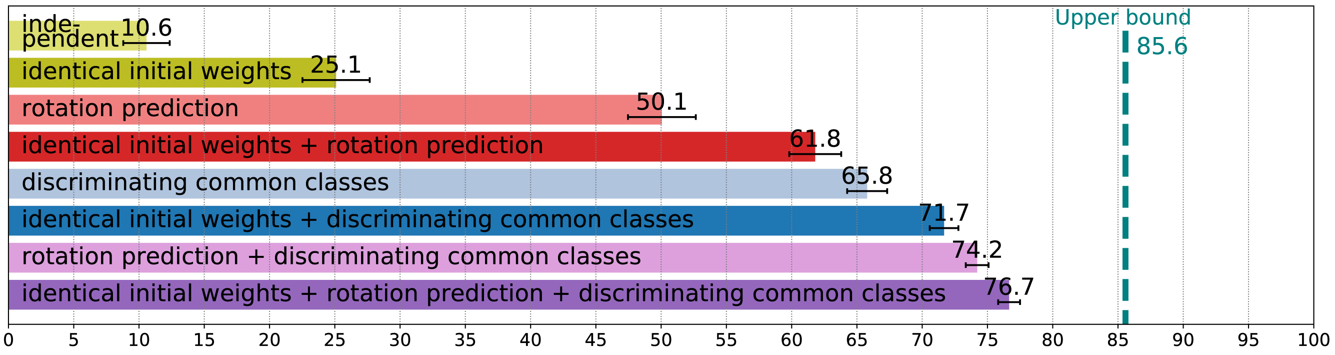


Figure 2: Recombination accuracy for different methods. We report recombination accuracy with standard deviations over 10 runs (horizontal line segments). Results for each dataset seperately are given in the supp. material.

Therefore we analyze here what happens when we train networks *incrementally* (Sec. 3.4). We first train a network on CIFAR-10 with auxiliary task heads DCC and RP. We then freeze the DCC and RP heads. Finally, we train a new network on STL-10 starting from identical initial weights (IIW) and also using the frozen DCC and RP heads. We repeat the analogue starting from STL-10.

Compared to joint training in Fig. 2, results decrease moderately from 76.7% to 72.7%. This demonstrates that we can make new networks compatible with existing ones.

Reaching the compatibility upper bound. While we achieved high compatibility in our experiments involving CIFAR-10 and STL-10 (Fig. 2), we did not reach the upper bound and hence our components are not *perfectly compatible*. However, a classification head optimized for CIFAR-10 is not expected to yield top accuracy on STL-10 (and vice versa). To remove the task mismatch, we repeat all experiments using CIFAR-10 for both tasks. In this setting, several combinations of methods reach the upper bound: IIW+RP, IIW+DCC, and IIW+RP+DCC (see supp. material). This shows that our methods are in principle strong enough to achieve perfect compatibility, when the data allows it.

Feature cross-correlation. Various analysis papers measure how similar, under some transformation, the features produced by two networks are. Their goal is to understand whether the networks activate on similar image patterns and hence learn similar representations (discussed in Sec. 2). Following (Li et al. 2016b), we repeat this analysis for two independently trained networks. We feed all images from the full CIFAR-10 and STL-10 test sets to both feature extractors and measure cross-correlation under an optimal permutation, as determined post-hoc (denoted as 'permuted independent'). For our methods of compatibility, instead, we measure cross-correlation of the features produced by two networks directly after training, *i.e. without* permutation, as these methods aim to directly make features compatible. Results are shown in Fig. 3.

We observe that all our compatibility methods directly yields reasonably correlated features (≥ 0.15). While correlation for the post-hoc aligned features is 0.34 (permuted independent), we measure higher correlation for RP+ random IIW(0.38) and very high correlation when starting from pre-trained IIW: ($0.69 - 0.76$). Hence, using pre-trained weights leads to strongly correlated features. However, we also find that RP leads to more strongly correlated features (0.38) than DCC (0.19), even though DCC leads to a higher recombination accuracy (Fig. 2). Similarly, when using pre-trained weights, IIW yields the same correlation as IIW+DCC, yet the latter yields 8.5% higher recombination accuracy (Fig. 2). We therefore conclude that a higher cross-correlation does not necessarily translate to more compatible features as measured by recombination accuracy. Hence trying to learn identical feature mappings (as in feature distillation, *e.g.* (Romero et al. 2015)) is not required for compatibility.

5 Applications

5.1 Unsupervised Domain Adaptation

Application. We transfer knowledge from a source domain with labeled data to a target domain with unlabeled data.

Experimental setup (Fig. 1b). We first train a model on the source training set, where the model consists of a feature extractor, a classification head and an auxiliary rotation prediction head RP (initialized by training for rotation prediction on ILSVRC-12 (Russakovsky et al. 2015)). We then want to adapt the feature extractor of this source model to the target domain while preserving compatibility with the original classification head. We do this by freezing the RP head while fine-tuning the feature extractor on the unlabeled target training set. For this we minimize the self-supervised RP loss for 1000 steps. Finally, we recombine this updated feature extractor with the source domain classification head to predict classes on the target domain. We report average class accuracy on the target test set. We evaluate adapting between CIFAR-10 and STL-10, as is common in this area (Ghifary et al. 2016; Shu et al. 2018; Sun et al. 2019). We use here a larger WRN-28 (Zagoruyko and Komodakis 2016) architecture as in (Sun et al. 2019) (see supp. material).

Results (Tab. 2). We compare our method to previous approaches and two baselines based on our source model. One baseline uses the model as is. The other updates BN statistics at test time, which performs significantly better. This

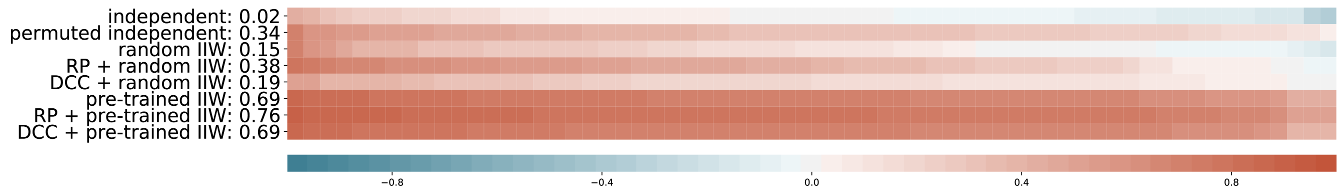


Figure 3: Cross-Correlation between features produced by two feature extractors. The blocks visualise the per-channel cross-correlations, sorted by magnitude. The number reports the cross-correlation averaged over all channels.

Method	Source: / Target:	CIFAR-10 / STL-10	STL-10 / CIFAR-10	Avg.
VADA (Shu et al. 2018)		78.3%	71.4%	74.9%
VADA+Co-DA^{bn} (Kumar et al. 2018)		81.3%	76.3%	78.8%
DTA (Lee et al. 2019)		82.6%	72.8%	77.7%
Joint w/ rotation (Sun et al. 2019)		81.2%	65.6%	73.4%
Joint multi-objective (Sun et al. 2019)		82.1%	74.0%	78.1%
Our source model		77.2%	52.9%	65.1%
Our source model w/ test BN		82.0%	71.3%	76.7%
Ours w/ adaptation through RP		82.6%	73.1%	77.9%

Table 2: Accuracy for unsupervised domain adaptation.

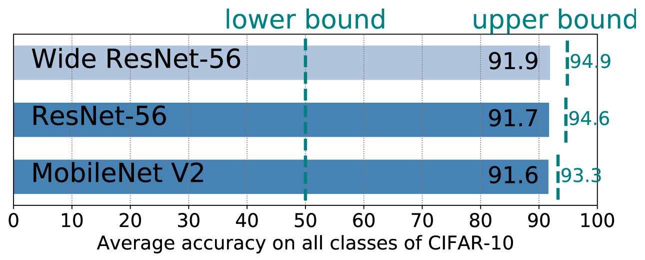


Figure 4: Accuracy when transferring a classification head to compatible feature extractors.

confirms their importance, as discussed in Sec. 4 and observed by (Li et al. 2016a). Our method improves performance further and matches the state-of-the-art on adapting from CIFAR-10 to STL-10 (Lee et al. 2019) (82.6%). The methods (Kumar et al. 2018; Sun et al. 2019) perform best for adapting from STL-10 to CIFAR-10. On average over both adaptation directions, our method is competitive (78.8% for (Kumar et al. 2018) *vs.* 77.9% for us).

Importantly, our method is simpler and faster than competing methods. The state-of-the-art (Kumar et al. 2018) combines multiple models, includes a domain discriminator (Ganin and Lempitsky 2014; Ganin et al. 2016), employs a custom network architecture (Shu et al. 2018), and trains for 80000 steps on the joint source and target training sets. Instead, we use a single ResNet model and fine-tune only for 1000 steps on the target domain, which makes our method computationally faster. Finally, (Sun et al. 2019) gets significant gains by combining multiple self-supervised objectives, which we could potentially include as well.

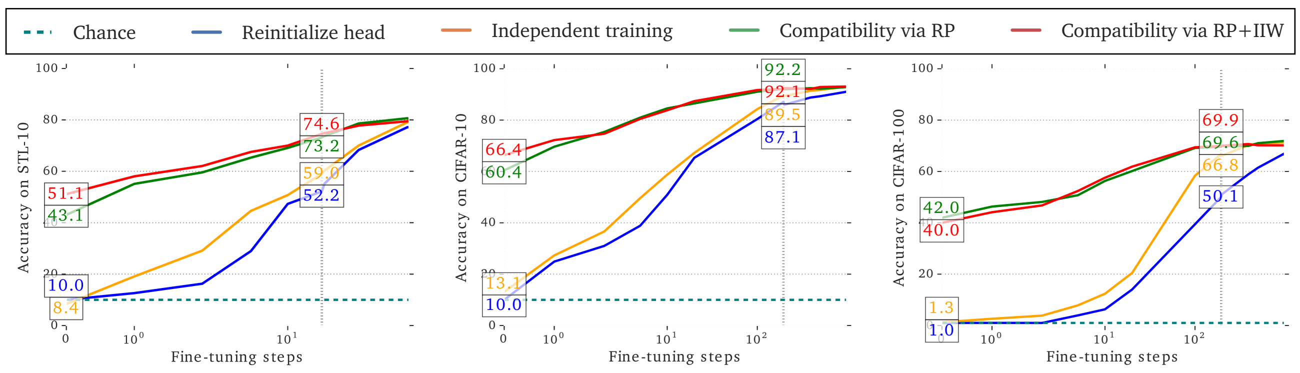
5.2 Compatibility Across Feature Extractors With Different Architectures

Application. We want to achieve compatibility between feature extractors having different architectures, thus enabling transferring task heads across them. As a practical application we consider a single classification task which runs on many devices, each with a hardware-tailored network architecture (e.g. a powerful server, a standard desktop, a mobile phone). Normally, every time the set of classes to be recognized changes, all networks need to be retrained. Instead, if their feature extractors are compatible, only one extractor and its corresponding classification head need to be retrained. We can then transfer that classification head to all other models. This greatly facilitates deployment of the updated classifier to all client devices, especially if different people are responsible for maintaining the different models.

Experimental setup (Fig. 1c). We consider three feature extractor architectures: ResNet-56 (He et al. 2016), Wide ResNet-56 (Zagoruyko and Komodakis 2016), MobileNet V2 (Sandler et al. 2018). We combine these with a DCC head based on MobileNet V2. In this application, DCC not only encourages compatibility but also directly solves the target task (as there is just one task). We split MobileNet V2 into components after the 11-th inverted ResNet block (out of 17). To fit all extractors to a single DCC head, we add to each extractor a 1x1 convolution layer with 64 output channels. Differences in spatial resolution are resolved by the average pooling in the penultimate layer of the MobileNet V2 DCC.

At first, we assume that we only have data for the first 5 classes of CIFAR-10. We use these to jointly train the three feature extractors with the DCC head. At this point, each ‘feature extractor plus DCC’ network addresses the target task for a particular device. Next, suppose we obtain labeled data for 5 new classes (resulting in the full CIFAR-10 training set). Instead of re-training everything, we only want to update the DCC head. To do so, we first extend the classification layer of the DCC head to handle 10 classes. Then, we choose the trained Wide ResNet-56 as the *reference feature extractor*. We freeze it, attach the DCC head, and fine-tune this combination on the CIFAR-10 training set. Finally, we attach the updated DCC to each individual extractor and evaluate on the CIFAR-10 test set. Note that in this process we updated none of the feature extractors after the initial training phase (updating it is investigated in the supp. material).

Results (Fig. 4). As an upper bound we train the individual networks on CIFAR-10 (also with a rotation prediction head). As an optimistic lower bound we consider perfectly discrim-



(a) STL-10 (b) CIFAR-10 (c) CIFAR-100

Figure 5: Average class accuracy of recombined components as a function of fine-tuning steps (log scale, up to 5 epochs). We overlay accuracy values directly after recombination and after 1 epoch. Through training with RP, components are compatible, enabling direct recombination.

inating the first five classes, leading to 50% accuracy. As Fig. 4 shows, recombining either ResNet-56 or MobileNet V2 with the updated DCC head lead to excellent accuracy of 91.7% and 91.6% respectively. While the upper bounds are even higher at 94.6% and 93.3%, our method requires much less computation and greatly facilitates deployment. Part of the gap to the upper bound can be attributed to changes in architectures: we added 1x1 convolutions and use mixed architectures with a simple MobileNet head. If we redo the upper bound using these changed architectures, we get accuracies between 92.7% and 92.8%. This suggests that optimizing architectures would lead to even better results.

5.3 Faster Transfer Learning

Application. In transfer learning, the goal is to improve results on a target task by reusing knowledge derived from a related source task. In the deep learning era, the standard approach is to reuse the feature extractor of a model trained on the source training set. This source feature extractor and a randomly initialized task-specific head are combined into a new model, which is then fine-tuned on the target training set. When there are many possible source tasks, this process is computationally expensive, *e.g.* (Zamir et al. 2018) reports consuming 50'000 GPU hours. Instead, we propose to train an initial target task head and *reuse* it when exploring different source tasks to transfer from (Fig. 1d). For this, we recombine the source feature extractor and the initial target task head into a new model. When these components are compatible, the benefits of transferring from a potential source can be evaluated and capitalized on with no or little fine-tuning on the target training set.

Experimental setup (Fig. 1d). We study transferring from a model trained on ILSVRC-12 as the source task. For this, we simply replace the feature extractor of the target task model with the source one, while keeping the target task head. We make these components compatible by training with rotation prediction (RP) and an incremental training scheme (Sec. 3.4). In this scheme, we first need to set the weights of RP ($\mathbf{\Theta}_s$), which we obtain by training a model on CIFAR-100 (Krizhevsky 2009). Then, the source and target models are trained with this frozen rotation prediction head, forcing their feature extractor to produce features that work with that same rotation prediction head.

We compare our method against re-initializing the target task head or recombining independently trained components. For these baselines, we also use rotation prediction as an auxiliary task for fair comparison, but initialize the weights of its head randomly for each network. We evaluate transferring a feature extractor trained on ILSVRC-12 (Russakovsky et al. 2015) to different target tasks, here CIFAR-10 (Krizhevsky 2009), STL-10 (Coates, Ng, and Lee 2011), or CIFAR-100 (Krizhevsky 2009). We measure transfer efficiency as the accuracy directly after recombination, and after a few epochs of fine-tuning on the target task.

Results (Fig 5). Our method achieves strong results in terms of accuracy on the target task, even without any fine-tuning. Here, the networks are trained separately and only made compatible via RP and optionally IIW. Nonetheless, our method achieves 40.0%-66.4% recombination accuracy, despite the differences in the datasets and their class vocabularies. Instead, the baselines yield random performance before fine-tuning, as expected. After 1 epoch of fine-tuning our methods are still significantly better than the baselines. They converge only after fine-tuning for several epochs (Fig 5).

In summary, our method reduces the need for fine-tuning when transferring components. As this is a core part of existing transfer learning methods (Zamir et al. 2018; Dwivedi and Roig 2019; Achille et al. 2019; Yan, Acuna, and Fidler 2020), our method can help speed these up.

6 Conclusion

We have demonstrated that we can train networks to produce compatible features, without compromising accuracy on the original tasks. We can do this through joint training, or by making new networks compatible with existing ones, through iterative training. By addressing three different applications, we demonstrated that our approach is widely applicable.

References

Achille, A.; Lam, M.; Tewari, R.; Ravichandran, A.; Maji, S.; Fowlkes, C. C.; Soatto, S.; and Perona, P. 2019. Task2Vec: Task embedding for meta-learning. In *ICCV*.

Aljundi, R.; Babiloni, F.; Elhoseiny, M.; Rohrbach, M.; and Tuytelaars, T. 2018. Memory aware synapses: Learning what (not) to forget. In *ECCV*.

Ben-David, S.; Blitzer, J.; Crammer, K.; Kulesza, A.; Pereira, F.; and Vaughan, J. W. 2010. A theory of learning from different domains. *Machine learning* .

Buciluǎ, C.; Caruana, R.; and Niculescu-Mizil, A. 2006. Model compression. In *ACM SIGKDD*.

Chen, Z.; and Liu, B. 2018. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* .

Choromanska, A.; Henaff, M.; Mathieu, M.; Arous, G. B.; and LeCun, Y. 2015. The loss surfaces of multilayer networks. In *Artificial intelligence and statistics*.

Coates, A.; Ng, A.; and Lee, H. 2011. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS*.

Doersch, C.; and Zisserman, A. 2017. Multi-task self-supervised visual learning. In *ICCV*.

Donahue, J.; Jia, Y.; Vinyals, O.; Hoffman, J.; Zhang, N.; Tzeng, E.; and Darrell, T. 2013. Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv preprint arXiv:1310.1531* .

Dwivedi, K.; and Roig, G. 2019. Representation similarity analysis for efficient task taxonomy & transfer learning. In *CVPR*.

Farquhar, S.; and Gal, Y. 2018. Towards robust evaluations of continual learning. *arXiv* .

Frome, A.; Corrado, G. S.; Shlens, J.; Bengio, S.; Dean, J.; Mikolov, T.; et al. 2013. DeViSE: A deep visual-semantic embedding model. In *NeurIPS*.

Ganin, Y.; and Lempitsky, V. 2014. Unsupervised domain adaptation by backpropagation. In *ICML*.

Ganin, Y.; Ustinova, E.; Ajakan, H.; Germain, P.; Larochelle, H.; Laviolette, F.; Marchand, M.; and Lempitsky, V. 2016. Domain-adversarial training of neural networks. *JMLR* .

Ghifary, M.; Kleijn, W. B.; Zhang, M.; Balduzzi, D.; and Li, W. 2016. Deep reconstruction-classification networks for unsupervised domain adaptation. In *ECCV*.

Gidaris, S.; Singh, P.; and Komodakis, N. 2018. Unsupervised representation learning by predicting image rotations. In *ICLR*.

Gupta, S.; Hoffman, J.; and Malik, J. 2016. Cross modal distillation for supervision transfer. In *CVPR*.

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

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

Hénaff, O. J.; Razavi, A.; Doersch, C.; Eslami, S.; and Oord, A. v. d. 2019. Data-efficient image recognition with contrastive predictive coding. *arXiv preprint arXiv:1905.09272* .

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* .

Ioffe, S.; and Szegedy, C. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *ICML*.

Kaiser, L.; Gomez, A. N.; Shazeer, N.; Vaswani, A.; Parmar, N.; Jones, L.; and Uszkoreit, J. 2017. One model to learn them all. *arXiv preprint arXiv:1706.05137* .

Karpathy, A.; and Fei-Fei, L. 2015. Deep visual-semantic alignments for generating image descriptions. In *CVPR*.

Kirkpatrick, J.; Pascanu, R.; Rabinowitz, N.; Veness, J.; Desjardins, G.; Rusu, A. A.; Milan, K.; Quan, J.; Ramalho, T.; Grabska-Barwinska, A.; et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proc. Nat. Acad. Sci. USA* .

Kontogianni, T.; Gygli, M.; Uijlings, J.; and Ferrari, V. 2020. Continuous adaptation for interactive object segmentation by learning from corrections. In *ECCV*.

Kornblith, S.; Norouzi, M.; Lee, H.; and Hinton, G. 2019. Similarity of neural network representations revisited. In *ICML*.

Krizhevsky, A. 2009. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto.

Kumar, A.; Sattigeri, P.; Wadhawan, K.; Karlinsky, L.; Feris, R.; Freeman, B.; and Wornell, G. 2018. Co-regularized alignment for unsupervised domain adaptation. In *NeurIPS*.

Lee, S.; Kim, D.; Kim, N.; and Jeong, S.-G. 2019. Drop to adapt: Learning discriminative features for unsupervised domain adaptation. In *ICCV*.

Lenc, K.; and Vedaldi, A. 2019. Understanding Image Representations by Measuring Their Equivariance and Equivalence. *IJCV* .

Li, Y.; Wang, N.; Shi, J.; Liu, J.; and Hou, X. 2016a. Revisiting batch normalization for practical domain adaptation. *arXiv preprint arXiv:1603.04779* .

Li, Y.; Yosinski, J.; Clune, J.; Lipson, H.; and Hopcroft, J. E. 2016b. Convergent learning: Do different neural networks learn the same representations? In *ICLR*.

Li, Z.; and Hoiem, D. 2017. Learning without forgetting. *IEEE Trans. on PAMI* .

Lu, Q.; Chen, P.-H.; Pillow, J. W.; Ramadge, P. J.; Norman, K. A.; and Hasson, U. 2018. Shared representational geometry across neural networks. In *NeurIPS*.

Maninis, K.-K.; Radosavovic, I.; and Kokkinos, I. 2019. Attentive single-tasking of multiple tasks. In *CVPR*.

Mehrer, J.; Kriegeskorte, N.; and Kietzmann, T. 2018. Beware of the beginnings: intermediate and higherlevel representations in deep neural networks are strongly affected by weight initialization. In *Conference on Cognitive Computational Neuroscience*.

Michieli, U.; and Zanuttigh, P. 2019. Incremental Learning Techniques for Semantic Segmentation. In *ICCV Workshop*.

Misra, I.; Shrivastava, A.; Gupta, A.; and Hebert, M. 2016. Cross-stitch networks for multi-task learning. In *CVPR*.

Morcos, A.; Raghu, M.; and Bengio, S. 2018. Insights on representational similarity in neural networks with canonical correlation. In *NeurIPS*.

Ngiam, J.; Peng, D.; Vasudevan, V.; Kornblith, S.; Le, Q. V.; and Pang, R. 2018. Domain adaptive transfer learning with specialist models. *arXiv preprint arXiv:1811.07056* .

Noroozi, M.; and Favaro, P. 2016. Unsupervised learning of visual representations by solving jigsaw puzzles. In *ECCV*.

Parisi, G. I.; Kemker, R.; Part, J. L.; Kanan, C.; and Wermter, S. 2019. Continual lifelong learning with neural networks: A review. *Neural Networks* .

Rebuffi, S.; Kolesnikov, A.; Sperl, G.; and Lampert, C. 2017. iCaRL: Incremental Classifier and Representation Learning. 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 *NeurIPS*.

Romero, A.; Ballas, N.; Kahou, S. E.; Chassang, A.; Gatta, C.; and Bengio, Y. 2015. Fitnets: Hints for thin deep nets. In *ICLR*.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; Berg, A.; and Fei-Fei, L. 2015. ImageNet Large Scale Visual Recognition Challenge. *IJCV* .

Ruvolo, P.; and Eaton, E. 2013. ELLA: An efficient lifelong learning algorithm. In *ICML*.

Saito, K.; Ushiku, Y.; and Harada, T. 2017. Asymmetric tri-training for unsupervised domain adaptation. In *ICML*.

Sandler, M.; Howard, A. G.; Zhu, M.; Zhmoginov, A.; and Chen, L. 2018. MobileNetV2: Inverted Residuals and Linear Bottleneck. In *CVPR*.

Sharif Razavian, A.; Azizpour, H.; Sullivan, J.; and Carlsson, S. 2014. CNN features off-the-shelf: an astounding baseline for recognition. In *CVPR workshop*.

Shmelkov, K.; Schmid, C.; and Alahari, K. 2017. Incremental learning of object detectors without catastrophic forgetting. In *ICCV*.

Shu, R.; Bui, H. H.; Narui, H.; and Ermon, S. 2018. A DIRT-T Approach to Unsupervised Domain Adaptation. In *ICLR*.

Shuai Tang, Wesley J. Maddox, C. D. T. D. A. D. 2020. Similarity of Neural Networks with Gradients. *arXiv preprint arXiv:2003.11498* .

Socher, R.; Ganjoo, M.; Manning, C. D.; and Ng, A. 2013. Zero-shot learning through cross-modal transfer. In *NeurIPS*.

Sun, Y.; Tzeng, E.; Darrell, T.; and Efros, A. A. 2019. Unsupervised Domain Adaptation through Self-Supervision. *arXiv preprint arXiv:1909.11825* .

Tenenbaum, J. B.; Kemp, C.; Griffiths, T. L.; and Goodman, N. D. 2011. How to grow a mind: Statistics, structure, and abstraction. *science* .

Tschannen, M.; Djolonga, J.; Ritter, M.; Mahendran, A.; Houlsby, N.; Gelly, S.; and Lucic, M. 2019. Self-Supervised Learning of Video-Induced Visual Invariances. *arXiv preprint arXiv:1912.02783* .

Tzeng, E.; Hoffman, J.; Zhang, N.; Saenko, K.; and Darrell, T. 2014. Deep domain confusion: Maximizing for domain invariance. *arXiv preprint arXiv:1412.3474* .

Wang, L.; Hu, L.; Gu, J.; Hu, Z.; Wu, Y.; He, K.; and Hopcroft, J. 2018. Towards understanding learning representations: To what extent do different neural networks learn the same representation. In *NeurIPS*.

Wang, M.; and Deng, W. 2018. Deep visual domain adaptation: A survey. *Neurocomputing* .

Wang, Y.; van de Weijer, J.; and Herranz, L. 2018. Mix and match networks: encoder-decoder alignment for zero-pair image translation. In *CVPR*.

Yan, X.; Acuna, D.; and Fidler, S. 2020. Neural Data Server: A Large-Scale Search Engine for Transfer Learning Data. *arXiv preprint arXiv:2001.02799* .

Zagoruyko, S.; and Komodakis, N. 2016. Wide residual networks. In *BMVC*.

Zamir, A. R.; Sax, A.; Shen, W.; Guibas, L. J.; Malik, J.; and Savarese, S. 2018. Taskonomy: Disentangling task transfer learning. In *CVPR*.

Zenke, F.; Poole, B.; and Ganguli, S. 2017. Continual learning through synaptic intelligence. In *ICML*.

Zhai, X.; Oliver, A.; Kolesnikov, A.; and Beyer, L. 2019. S4L: Self-Supervised Semi-Supervised Learning. In *ICCV*.

Zhang, C.; Bengio, S.; and Singer, Y. 2019. Are all layers created equal? In *ICML Workshop Deep Phenomena*.

Zhang, W.; Ouyang, W.; Li, W.; and Xu, D. 2018. Collaborative and adversarial network for unsupervised domain adaptation. In *CVPR*.