

# A Flexible Selection Scheme for Minimum-Effort Transfer Learning

Amélie Royer  
IST Austria

aroyer@ist.ac.at

Christoph H. Lampert  
IST Austria

chl@ist.ac.at

## Abstract

*Fine-tuning* is a popular way of exploiting knowledge contained in a pre-trained convolutional network for a new visual recognition task. However, the orthogonal setting of transferring knowledge from a pretrained network to a visually different yet semantically close source is rarely considered: This commonly happens with real-life data, which is not necessarily as clean as the training source (noise, geometric transformations, different modalities, etc.).

To tackle such scenarios, we introduce a new, generalized form of fine-tuning, called **flex-tuning**, in which any individual **unit** (e.g. layer) of a network can be tuned, and the most promising one is chosen automatically. In order to make the method appealing for practical use, we propose two **lightweight and faster selection procedures** that prove to be good approximations in practice. We study these selection criteria empirically across a variety of domain shifts and data scarcity scenarios, and show that fine-tuning individual units, despite its simplicity, yields very good results as an adaptation technique. As it turns out, in contrast to common practice, rather than the last fully-connected unit it is best to tune an intermediate or early one in many domain-shift scenarios, which is accurately detected by flex-tuning.

## 1. Introduction

Deep convolutional networks have substantially advanced the state of the art in many areas of computer vision. These networks are often interpreted as a feature extraction stage (typically convolutional layers), followed by a small classifier (fully connected layers), and have the ability to learn features from data directly instead of having them hard-coded, as was the case for previous shallow techniques. However, this comes with a cost, as it requires a lot more training data than methods relying on fixed ad-hoc feature extraction. Consequently, it is not surprising that the first successes of deep networks in image classification occurred as large annotated datasets were made available, e.g. MNIST [22] for digit recognition (60,000 training samples) or ImageNet [32] for object classification (1.2 million).

When only little available training data is available, however, training a deep feature extraction pipeline from scratch is not possible, as it often leads to severe overfitting. Instead, two main *transfer learning* strategies have emerged, exploiting the fact that deep convolutional networks pre-trained on large datasets are freely available these days [25, 38, 39]: *Either*, one isolates and “freezes” the feature extraction stage of the pre-trained model and then uses the available new data to train only the smaller, less prone to overfitting, classifier stage, *or alternatively*, one fully fine-tunes the model, *i.e.* initializes the network parameters from the pre-trained network, and then trains all layers using the new data, typically only for a few steps, to avoid overfitting. Choosing the best solution depends not only on the *amount of available samples*, though, but also on the *data characteristics*. For example, it has been observed that features learned on large and varied natural images datasets, e.g. ImageNet, transfer well to related domains such as aerial or even biomedical images [19]. However, for domains with very different low-level image statistics, e.g. sketches, fine-tuning all layers is preferable [3]. Moreover, fine-tuning only a few classification layers is often easier, hence when both options are viable, one might prefer this alternative.

In this work, we argue for a more systematic approach to exploiting pre-trained networks, in situations where the new input domain can *vary greatly in terms of visual appearance*, but its output space shares *similar semantics* with the one the model was pre-trained on. We introduce the idea of *flex-tuning*, a general-purpose transfer learning scheme that leverages the information of an available pre-trained model by fine-tuning a *targeted part of the model*, not necessarily the last layer or all layers, *but any individual layer or block of consecutive layers, selected in a data-dependent way*. In fact, the idea of focusing training resources on specific intermediate layers draws inspiration from an important transfer learning paradigm: It has been consistently observed across various networks and datasets in the literature that early convolutional layers capture elementary local properties of images such as edges or local textures, while middle layers rather represent configurations of several such elements, and the last feature layers extract information about

high-level concepts, such as object parts and their configurations [5, 27, 41]. Thus, in order to adapt, for instance, a network trained on clean natural images to work with noisy ones, we hypothesize it is easier to fine-tune early layers, while for adapting the same network to artistic paintings, focusing on a later layer would be more promising.

Our contribution is three-fold: *First*, we formally define *flex-tuning*, which is a strategy for, given a pre-trained network and a new training dataset, deciding in a data-dependent and automatic way which of the available layers to fine-tune, based on a selection criterion on a held-out validation dataset. *Second*, in order to make flex-tuning more appealing for practical use, we further introduce two variants based on a more efficient selection criterion, called *fast flex-tuning* and *even faster flex-tuning*, that avoid the need to train multiple fine-tuned models for the selection process. *Finally*, we design an extensive experimental setup that covers varied visual domain shifts, data scarcity scenarios and architectures. We show that flex-tuning almost always improves classification accuracy over standard fine-tuning techniques, particularly in settings where fine-tuning all layers is prone to overfitting, such as settings with small sample size and large networks. Furthermore, the (even) faster flex-tuning variants are generally on par with flex-tuning while providing a much lighter selection procedure.

## 2. Related work

Transferrability of pretrained convolutional networks across visual tasks has been often observed and extensively studied in the computer vision literature [1, 7, 8, 40, 42]. In fact, many state-of-the-art computer vision models are not trained from random initialization, but rely crucially on the re-use of weights from networks pre-trained on large classification tasks, such as ImageNet [32]. Popular examples include the YOLO object detector [31] or fully-convolutional networks for segmentation [24]. In the weakly supervised learning literature, pre-trained features are also used as a compact and semantically meaningful image representation, *e.g.* for image retrieval [2], style transfer [11, 16], colorization [21], or unsupervised part detection [35]. All of these approaches typically aim at transferring knowledge between two tasks that have different output structures but similar input domain appearances and distributions. Closest to our work is [40], which studies the outcome of fine-tuning from different levels of a pre-trained network for the standard transfer learning setting. In comparison, we analyze the effect of tuning a single unit of a pre-trained network, in particular for situations where source and target domains are visually dissimilar but semantically close.

In fact, our interest lies exactly in these orthogonal scenarios, *i.e.* where one has a similar output task, typically multi-class classification, but with –potentially significantly – different *source* and *target* input distributions. This setting

resembles, yet differs from, the problem of *domain adaptation* [33, 12, 10], where the goal is to construct a classifier for a, usually unlabelled, target task by exploiting one or more source tasks. In domain adaptation one typically assumes that samples from both source and target domain are available, while in the fine-tuning situation, one only has access to a pre-trained network, not the data distribution it was trained on: This aspect rules out adversarial training [17, 43], paired samples [14], or more generally, exploiting any concrete knowledge from the source distribution to improve predictions on the target domain.

In fact, with the growth of datasets and necessary compute resources, the ability to tune networks without access to the original training data is becoming more and more important: First, when dealing with very large source datasets, training jointly on the source and target domains (as many domain adaptation methods require) is computationally impractical. Second, source training data is sometimes non-public, especially in commercial settings. Third, specific applications require data privacy, preventing public data release, for instance for protecting individuals identities in face recognition models. As such, learning under privacy constraints has become a popular topic in recent years [28].

Recent work has also tackled the problem of domain adaptation by transferring from source to target directly at the pixel level, either via generative models [4] or by identifying simpler causal transformations [29]. Weight tuning methods are nonetheless simpler to use, as they directly act on feature representations, rather than learning a transformation that holds independently of the pre-trained network.

## 3. Flex-Tuning

Our first contribution in this work is to highlight that *simple and lightweight, but surprisingly effective, model adaptation is possible by fine-tuning the weights of only a single unit in a pretrained network, provided that the right unit is chosen*. Which is the right unit depends crucially, and in a non-trivial way, on the relation between source and target domains as well as on the amount of available data. We propose to identify the best unit automatically in a data-dependent manner using a procedure we call *flex-tuning*.

### 3.1. Transferring knowledge across domain shifts

First, we formally introduce the transfer learning scenario we are interested in: We are given a pre-trained convolutional network,  $N$ , mapping input space  $X$  to an output space  $Y$ , and whose weights were pre-trained on a training dataset from a *source domain*, that is however not available anymore. Our goal is to learn a network for a *target domain*, for which a new, and potentially small, annotated dataset,  $\mathcal{D} \stackrel{iid}{\sim} \mathcal{P}(X, Y)$ , is available. In contrast to the standard *transfer learning* application scenario, we consider practi-

cal settings where the target domain is *semantically* close but *visually* different from the source domain. Here, by *semantically* close, we mean that the output space of the target task is a subset of the source task. Extending the framework to different output structures, *e.g.* from a classification task to a detection task, would be possible by fine-tuning both the unit selected by flex-tuning and the last fully-connected layer. In this work, we focus on thoroughly analyzing and characterizing the influence of *single units* on transferring knowledge across visually different domains and leave the possibility of combining multiple units for future work.

Nonetheless, the setting we consider encompasses a variety of real-world scenarios, where the source and target domains do not overlap well. For example, we can consider a source network trained on natural images, with the target task of classifying monochrome sketches; or a source network trained on scenes under daylight, that should also operate at night, *etc.* Here we work with images as inputs, and discrete labels as outputs. However, the underlying principles apply equally to other input domains and tasks.

### 3.2. Flex-tuning

We consider pre-trained multi-layer convolutional architectures, that we decompose into smaller *units*, which we denote by  $N = N_L \circ \dots \circ N_1$ . In practice, a unit can simply be a single convolutional or fully-connected layer, or, for more complex architectures, a block of consecutive layers. Intuitively, we think of units  $N_1$  to  $N_{L-1}$  as the feature extraction part, while the last layer  $N_L$  is the performs the actual classification, however the method applies to arbitrary decompositions. Given such a decomposition, the goal of flex-tuning is to analyze the influence of tuning specific units, not only the last one, for transferring knowledge across domains with different visual appearances. [Algorithm 1](#) describes the steps of flex-tuning in pseudo-code: For each unit of the network, we construct a fine-tuned network  $N_{ft-\ell}$  by training the network on the available target data, allowing only the weights of the  $\ell$ -th unit to change, while keeping all the others frozen. We also create a network  $N_{ft-all}$ , for which all layers are fine-tuned. We train each network with an early stopping criterion, monitoring its performance on the validation set,  $\mathcal{D}_{val}$ . This prevents overfitting in a way that is data-dependent and adaptive to each training setting. In fact, different units might have very different numbers of weight parameters, and therefore will often need different numbers of epochs to converge. Finally, we choose the best model out of these  $L + 1$  networks by comparing their accuracy on the validation set and output it as the *flex-tuned* model,  $N_{flex}$ .

### 3.3. Practicality of the method

Technically, [Algorithm 1](#) performs an exhaustive search over the potential fine-tuned models. Therefore, the exist-

---

#### Algorithm 1 Flex-Tuning (*flex*)

---

**input** target training and validation sets,  $\mathcal{D}_{train}$  and  $\mathcal{D}_{val}$   
**input** pre-trained network with  $L$  units,  $N = N_L \circ \dots \circ N_1$

- 1: **for**  $\ell = 1, \dots, L$  **do**
- 2:  $N_{ft-\ell} \leftarrow$  fine-tune unit  $\ell$  of  $N$  on  $\mathcal{D}_{train}$  until accuracy on  $\mathcal{D}_{val}$  stops improving
- 3:  $a_{ft-\ell} \leftarrow$  accuracy of  $N_{ft-\ell}$  on  $\mathcal{D}_{val}$
- 4: **end for**
- 5:  $N_{ft-all} \leftarrow$  fine-tune all units of  $N$  on  $\mathcal{D}_{train}$  until accuracy on  $\mathcal{D}_{val}$  stops improving
- 6:  $a_{ft-all} \leftarrow$  accuracy of  $N_{ft-all}$  on  $\mathcal{D}_{val}$
- 7:  $N_{flex} \leftarrow N_{best}$  for  $best \leftarrow \arg \max_{X \in \{1, \dots, L, all\}} a_{ft-X}$

**output**  $N_{flex}$

---

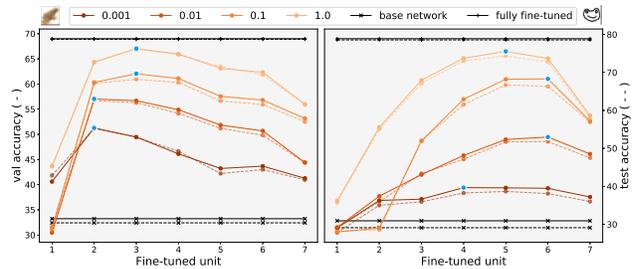


Figure 1. Validation (-) and test (- -) accuracies for fine-tuning a single unit of a pretrained CIFAR network to the Blurry CIFAR (left) and Quick, Draw! (right) datasets. Each line color represents a different subsampling ratio of the target training dataset, while blue markers indicate the unit picked based on validation accuracy.

ing theoretical results for model selection [34] apply, and we obtain that, in the limit, flex-tuning will indeed choose the best of the possible models. Moreover, the difference between flex-tuning’s accuracy estimated from the validation data and the expected accuracy on future data decreases with a rate of  $O(\sqrt{\frac{\log L}{|\mathcal{D}_{val}|}})$ . In [Figure 1](#), we illustrate flex-tuning’s practical use: We apply the proposed method on a small network (5 convolutional and 2 fully connected layers) pre-trained on CIFAR [20] and to be adapted to a subset of the “Quick, Draw!” dataset [6] and a blurred variant of CIFAR, for different sizes of the target training dataset. These preliminary results show that (i) it is often beneficial to fine-tune an intermediate layer rather than the last one and that (ii) well-performing units strongly depend on the dataset and in a non-trivial way, but can be efficiently pinpointed with a simple selection criterion such as flex-tuning.

For deep networks however, flex-tuning can be computationally costly: It requires training as many networks as there are units, plus another one in which all units are fine-tuned. Let us denote the average number of training epochs by  $E_{one}$  when fine-tuning a single unit, and by  $E_{all}$  when fine-tuning all. Also, let us denote the corresponding average computational cost of one such epoch as  $c_{one}$  and  $c_{all}$ ,

respectively. Then the total runtime complexity of flex-tuning is  $O(LE_{\text{one}}c_{\text{one}} + E_{\text{all}}c_{\text{all}})$ . Even when taking into account that typically  $E_{\text{all}} > E_{\text{one}}$  and  $c_{\text{all}} > c_{\text{one}}$ , for reasonably large networks the complexity is often dominated by the computational cost of fine-tuning the network once for each unit. Since ultimately only one of the models is chosen, these computations end up wasted. To address this issue, we introduce two improved selection criteria in the following section to efficiently approximate flex-tuning.

## 4. Efficient Selection Criteria

### 4.1. Fast flex-tuning

To overcome the aforementioned computational inefficiency of flex-tuning, we propose a different criterion, *fast flex-tuning*, for selecting the unit to be fine-tuned. It relies on the idea that a given unit’s influence can be approximated by a few feed-forward passes rather than a full training process. While it does not come with formal guarantees, we found it to work nearly as well as the exhaustive search in practice, while at the same time requiring only 2 networks to be trained instead of  $L + 1$ . Algorithm 2 describes fast flex-tuning in pseudo-code: The method starts by training one new model,  $N_{\text{ft-all}}$ , by fine-tuning all units of the pre-trained network on the training data available for the target domain. From this, we construct  $L$  new networks by *network surgery*. For any  $\ell = 1, \dots, L$ , we create a proxy network,  $N_{\text{prox-}\ell}$ , by copying all units from  $N$ , except the  $\ell$ -th one, which is copied from the fine-tuned network,  $N_{\text{ft-all}}$ . Clearly, the resulting hybrid networks are not functional models, as the  $\ell$ -th unit and the other units were not trained together. Nevertheless, the construction allows us to derive a measure which of the network units is the most promising candidate for fine-tuning, namely the one that leads to the *biggest improvement in accuracy (if any) when applied to the target domain*. Numerically, we compute the accuracy of each model  $N_{\text{prox-}\ell}$  on the validation dataset and identify the value for  $\ell$  with highest accuracy. We then create a viable model by fine-tuning the selected unit on the target dataset  $\mathcal{D}$ . Finally, we output either this model, or the one in which all layers were fine-tuned (which is available as we created it at the beginning of the procedure), depending on which achieved the higher validation accuracy. We report the validation accuracies of the  $N_{\text{prox-}\ell}$  models for our different experimental settings in the supplemental material.

In comparison to flex-tuning, fast flex-tuning only has to fine-tune two networks instead of  $L + 1$ . Its runtime complexity is hence  $O(E_{\text{one}}c_{\text{one}} + E_{\text{all}}c_{\text{all}})$ , thereby providing substantial computational savings for large networks.

### 4.2. Even faster flex-tuning

In some situations, training from scratch or fine-tuning the complete network is simply computationally too costly:

---

#### Algorithm 2 Fast Flex-Tuning (*fast-flex*)

---

**input** target training and validation sets,  $\mathcal{D}_{\text{train}}$  and  $\mathcal{D}_{\text{val}}$   
**input** pre-trained network with  $L$  units,  $N = N_L \circ \dots \circ N_1$

- 1:  $N_{\text{ft-all}} \leftarrow$  fine-tune all units of  $N$  on  $\mathcal{D}_{\text{train}}$  until accuracy on  $\mathcal{D}_{\text{val}}$  stops improving
- 2:  $a_{\text{ft-all}} \leftarrow$  accuracy of  $N_{\text{ft-all}}$  on  $\mathcal{D}_{\text{val}}$
- 3: **for**  $\ell = 1, \dots, L$  **do**
- 4:  $N_{\text{prox-}\ell} \leftarrow N_L \circ \dots \circ N_{\ell+1} \circ [N_{\text{ft-all}}]_{\ell} \circ N_{\ell-1} \circ \dots \circ N_1$
- 5:  $a_{\ell} \leftarrow$  accuracy of  $N_{\text{prox-}\ell}$  on  $\mathcal{D}_{\text{val}}$
- 6: **end for**
- 7:  $\text{best} \leftarrow \arg \max_{\ell} a_{\ell}$ ,  $\ell \in \{1, \dots, L\}$
- 8:  $N_{\text{best}} \leftarrow$  fine-tune unit  $\text{best}$  of  $N$  on  $\mathcal{D}_{\text{train}}$  until accuracy on  $\mathcal{D}_{\text{val}}$  stops improving
- 9:  $a_{\text{ft-best}} \leftarrow$  accuracy of  $N_{\text{best}}$  on  $\mathcal{D}_{\text{val}}$
- 10:  $N_{\text{flex}} \leftarrow$  **if**  $a_{\text{ft-best}} \geq a_{\text{ft-all}}$  **then**  $N_{\text{best}}$  **else**  $N_{\text{ft-all}}$

**output**  $N_{\text{flex}}$

---



---

#### Algorithm 3 Even Faster Flex-Tuning (*faster-flex*)

---

**input** target training and validation sets,  $\mathcal{D}_{\text{train}}$  and  $\mathcal{D}_{\text{val}}$   
**input** pre-trained network with  $L$  units,  $N = N_L \circ \dots \circ N_1$

- 1:  $N_{\text{ft-all}} \leftarrow$  fine-tune all units of  $N$  on  $\mathcal{D}_{\text{train}}$  for a single epoch
- 2: **for**  $\ell = 1, \dots, L$  **do**
- 3:  $N_{\text{prox-}\ell} \leftarrow N_L \circ \dots \circ N_{\ell+1} \circ [N_{\text{ft-all}}]_{\ell} \circ N_{\ell-1} \circ \dots \circ N_1$
- 4:  $a_{\ell} \leftarrow$  accuracy of  $N_{\text{prox-}\ell}$  on  $\mathcal{D}_{\text{val}}$
- 5: **end for**
- 6:  $\text{best} \leftarrow \arg \max_{\ell} a_{\ell}$ ,  $\ell \in \{1, \dots, L\}$
- 7:  $N_{\text{flex}} \leftarrow$  fine-tune unit  $\text{best}$  of  $N$  on  $\mathcal{D}_{\text{train}}$  until accuracy on  $\mathcal{D}_{\text{val}}$  stops improving

**output**  $N_{\text{flex}}$

---

Neither flex-tuning nor fast flex-tuning are applicable, as both require training a network by fine-tuning all units as the first step of their selection process. To overcome this, we propose an even faster variant, as described in Algorithm 3.

*Even faster flex-tuning* resembles fast flex-tuning in that it selects a unit to be fine-tuned based on the accuracies of different proxy models that are obtained by network surgery, each time preserving  $L - 1$  units from the pre-trained source network and replacing the remaining one with its fine-tuned counterpart. The difference lies in that the fine-tuned units are obtained from a network in which all units have been fine-tuned for just a *single epoch*. This results in a total computational runtime of  $O(E_{\text{one}}c_{\text{one}} + c_{\text{all}})$ . We consider this close to optimal for an adaptive technique, as at least the cost  $E_{\text{one}}c_{\text{one}}$  clearly cannot be avoided, if the goal is to produce a network in which at least one unit has been fine-tuned. The drawback of the acceleration is that the *even faster flex-tuning* algorithm does not have access to a reliable estimate of what performance a network with all units fully fine-tuned would have achieved. This is however not relevant here as, by assumption, the computational

method	computational cost
flex	$LE_{\text{one}}c_{\text{one}} + E_{\text{all}}c_{\text{all}}$
fast-flex	$E_{\text{one}}c_{\text{one}} + E_{\text{all}}c_{\text{all}}$
faster-flex	$E_{\text{one}}c_{\text{one}} + c_{\text{all}}$
ft-fc	$E_{\text{one}}c_{\text{one}}$
ft-all	$E_{\text{all}}c_{\text{all}}$

Table 1. Runtime complexities.  $L$  is the number of units in the network,  $E_{\text{one}}$  and  $c_{\text{one}}$  are the average number of epochs until early stopping for fine-tuning one unit, and the estimated cost of one such epoch.  $E_{\text{all}}$  and  $c_{\text{all}}$  are the analogous quantities when fine-tuning all network units. In general,  $E_{\text{all}} > E_{\text{one}}$  and  $c_{\text{all}} > c_{\text{one}}$ .

budget does not suffice for training such a model anyway.

In summary, even faster flex-tuning is a generalization of fine-tuning the last unit of the network, as is often done in practice, but instead the most promising unit is chosen by a brief selection process. Table 1 summarizes the runtime complexity of all proposed models, as well as the two main baselines we use in our experiments: `ft-fc`, which fine-tunes always the last unit (*i.e.* the fully-connected layer(s)), and `ft-all`, which fine-tunes always all layers.

## 5. Experiments

In this section, we introduce our experimental setup, covering a large number of domain shifts and data scarcity scenarios. We then describe fine-tuning baselines commonly used in the literature, and compare them to the proposed methods, flex-tuning (`flex`), fast flex-tuning (`fast-flex`) and even faster flex-tuning (`faster-flex`).

### 5.1. Experimental set-up

We build several domain shift scenarios, ranging from simple parametric transformations to severe visual appearance shifts. In order to explore the impact of data scarcity, we additionally consider several subsampled versions of each target dataset, ranging from a few images per class to hundreds of them. The different settings are thus mainly characterized by: (i) the depth of the base source network, (ii) the size of the target dataset we tune on, and (iii) the type of input domain shift: simple parametric transformations, *e.g.* manipulating color channels, complex (non-trivially invertible) parametric transformations, and general free-range transformations. We summarize our setup in Table 2.

**Medium-sized experiments.** We first consider a small 4 layers network (which we decompose in 4 one-layer units: 2 convolutional layers followed by 2 fully-connected ones) pretrained on a subset of MNIST training images. We use the remaining samples (except 5000 of them that we keep for validation) to build synthetic domain shifts such as *affine transformations* (randomized or fixed for all images), or *random occlusions*. Second, we build a 7 layers network (7 one-layer units: 5 convolutional and 2 fully connected ones) that we pre-train on half of the CIFAR training set [20]. As

target domains, we consider several synthetic transformations of the remaining samples, as well as a subset of the QuickDraw dataset [6]: We restrict ourselves to the object classes they have in common, *i.e.* all CIFAR classes except for “deer”. We also consider the converse setting, *i.e.* pre-training on QuickDraw and using as target domains CIFAR and synthetically generated blurry and noisy QuickDraw samples. Since both aforementioned architectures have two fully connected layers, we consider two baselines, `ft-fc` (1) and `ft-fc` (2), corresponding to fine-tuning only the last, or the last two fully-connected layers respectively.

**Large-scale setting.** Finally, we consider two large-scale settings using the Inception2 architecture [13, 39, 37]. We decompose the model so as to not separate layers belonging to the same Inception module, which results in 13 units, the last one being the single fully-connected classification layer of the architecture. We first experiment on synthetic transformations of natural images. For this setting, we use a network pretrained on ILSVRC2012-train. We then split ILSVRC2012-val in three parts. 25k images are used to create target datasets, 5k are kept for validation and the remaining 20k are used for testing. Second, we consider the more challenging setting of stylistic transformations using the PACS dataset [23], initially introduced for the task of domain generalization: We use art paintings, cartoons and sketches, as target domains, which we further split into train/val/test sets. In this setting, the target task is a subset of the source ILSVRC classification task (ignoring the “person” class in PACS as it does not have an equivalent).

**Baselines.** We first consider the two most common transfer learning schemes as baselines. Starting with a network initialized with the same weights and architectures as the source pre-trained network,  $N$ : (i) `ft-all` consists in fine-tuning *all* layers  $N_1, \dots, N_L$  on the training set  $\mathcal{D}$  from the target domain, and (ii) `ft-fc`, which corresponds to fine-tuning only the last *fully-connected* units of the network, while keeping earlier units frozen. We also consider using scaling and shifting operations as in [36] and refer to this baseline as `ft-ss`: It consists in fine-tuning the last classification layer as well as lightweight kernel-scaling and bias-shifting parameters at every layer. Thus `ft-ss` acts on all levels of the architecture, but requires few additional learning parameters, hoping to prevent overfitting problems.

**Training.** We measure performance as  $\text{top-1}$  classification accuracy, and  $\text{top-5}$  for ILSVRC-based domains. We use the same hyperparameters as were used during training of the base source network. As is common, for finetuning, we use a lower base learning rate:  $10^{-3}$  for the small convolutional networks, and  $10^{-4}$  for the Inception2 networks. We train all models using the Adam [18] optimizer. As mentioned previously, we also employ an early stopping criterion based on validation accuracy, regularly computed

Source	Target domains
 <p><b>MNIST (subset) [22]</b> 25k images 10 classes 4-layers top-1: 0.989</p>	 <b>Blurry</b> top-1: 0.748  <b>Occluded</b> top-1: 0.581  <b>MNIST-M [9]</b> top-1: 0.439  <b>Transform (random)</b> top-1: 0.322  <b>SVHN [26]</b> top-1: 0.211  <b>Transform (fixed)</b> top-1: 0.160 <b>ratios ~ 3, 30, 300 and 3k images per class</b>
 <p><b>CIFAR (subset) [20]</b> 18k images 9 classes 7-layers top-1: 0.738</p>	 <b>Noisy</b> top-1: 0.540  <b>Blurry</b> top-1: 0.324  <b>QuickDraw [6]</b> top-1: 0.291 <b>ratios ~ 2, 20, 200 and 2k images per class</b>
 <p><b>ILSVRC [32] ('12 train split)</b> 1M images 1000 classes Inception2 top-5: 0.918</p>	 <b>YUV</b> top-5: 0.841  <b>Fixed rotation</b> top-5: 0.743  <b>Fixed scaling (symmetric pad)</b> top-5: 0.519  <b>Fixed scaling (stretch pad)</b> top-5: 0.440  <b>HSV</b> top-5: 0.384 <b>ratios ~ 2, 12 and 25 images per class</b>  <b>Art</b> top-1: 0.532  <b>Cartoon</b> top-1: 0.346  <b>Sketch</b> top-1: 0.142 <b>ratios ~ 2, 20 and 200 images per class</b>

Table 2. Source domains and architectures (*left*) we consider, with the corresponding target domains (*right*) and the training dataset **subsampling ratios** we consider, as the average number of images per class: the last entry corresponds to the full dataset size.

during training (every 5-10 epochs). This also dampens the negative effect of overfitting in scenarios that are overly prone to it (*e.g.* `ft-all` with small sample size and a large network). Finally, in the very scarce data setting ( $\sim 1$  image per class) we report metrics averaged over 20 runs, to avoid a potential bias towards the sampled training images.

## 5.2. Main results

In Table 3 we compare the proposed method and baselines on the MNIST, CIFAR and ILSVRC-based settings, for one subsampling ratio of the target training set. Results for other ratios show similar trends and are available in the supplemental material. For the more challenging PACS scenario, which exhibits both a strong visual shift and slight semantic labels shift from the source task, we report complete results across all subsampling ratios.

We observe that `flex` outperforms fine-tuning baselines

in almost all settings. It very rarely loses to the `ft-fc` baseline, but is sometimes tied with `ft-all`, which is a subcase of `flex` and `fast-flex` through the selection criterion. More precisely, over all subsampling ratios and domain shifts we have in total 72 transfer scenarios. Out of these, the two overall best methods are `flex` and `fast-flex`, achieving best accuracy 60 and 41 times respectively. Compared to this, `ft-all` only reaches the best accuracy 26 times, mostly for large sample size and medium-sized networks. It consistently loses due to overfitting in other scenarios. More interestingly, in terms of absolute values, we observe that when `flex` strictly wins, *i.e.* when it reaches the best accuracy and not in a tie with `ft-all`, it typically does so by a higher margin than in the reverse scenario, *i.e.* when one of the baselines strictly wins. We detail our main observations in the rest of the section.

**Comparison to baselines.** In the medium network or large sample size settings, `flex`-tuning expectedly generally chooses to fine-tune all layers, *i.e.* `flex` recovers `ft-all`. However, as the dataset size to network depth ratio decreases, fine-tuning all layers becomes more prone overfitting. In that case, `flex` prefers to fine-tune a specific unit, which generally performs better than the `ft-fc` baseline. More generally, the behavior of `ft-fc` strongly correlates with the difficulty of the input domain shift: it performs best in settings where the source domain early layers generalize well to the target domain, *e.g.* in the noisy CIFAR setting where the small additive random noise does not impact activations significantly. When the domain shift is more pronounced however, `ft-fc` is often outperformed by `flex`, `fast-flex` and `faster-flex` which pick a more adequate unit to tune. This shows there is a benefit to having the method pick the best unit to fine-tune, rather than restricting transfer learning to the last fully-connected layers. These conclusions also hold for `ft-ss`, although it provides a much stronger baseline than `ft-fc` and is sometimes on-par or outperforms the faster flex-tuning variants. However, its performance seems to depend on the type of domain shift: For instance, `ft-ss` performs moderately well on the colorized-ILSVRC setting. We attribute it to the fact that this setting involves a recombination of the channels which is not well captured by affine transformations of the parameters. Finally, `flex` and its variants are easier to implement in practice as they do not introduce additional parameters nor require to know how layers actually operate.

**Selecting the best unit.** We observe that the most promising unit selected by `flex`-tuning is often an intermediate one and does not follow an obvious pattern, showing that different domain shifts affect layer representations at different depths of the architecture: This is illustrated in Figure 2. On the same figure, we see that fast flex-tuning and even faster flex-tuning are good approximations of flex-tuning as

ILSVRC 	flex			ft-		
	flex	fast	faster	fc	ss	all
<b>ratio: 2 images per class</b>						
Art (0.53) 	<b>0.669</b>	<b>0.703</b>	<b>0.655</b>	0.626	0.630	0.628
Cartoon (0.32) 	0.639	<b>0.683</b>	0.593	0.618	0.647	0.507
Sketch (0.14) 	<b>0.625</b>	<b>0.606</b>	0.414	0.554	0.581	0.337
<b>ratio: 20 images per class</b>						
Art (0.53) 	<b>0.870</b>	<b>0.851</b>	<b>0.861</b>	0.729	0.849	0.724
Cartoon (0.32) 	<b>0.912</b>	<b>0.893</b>	0.841	0.820	0.887	0.709
Sketch (0.14) 	<b>0.852</b>	0.638	0.638	0.766	0.801	0.542
<b>ratio: 200 images per class</b>						
Art (0.53) 	<b>0.906</b>	<b>0.906</b>	0.823	0.791	0.887	0.746
Cartoon (0.32) 	<b>0.958</b>	0.956	0.952	0.868	0.956	0.925
Sketch (0.14) 	<b>0.924</b>	<b>0.924</b>	0.890	0.767	0.916	0.875

Table 3. Break-down of results comparing our proposed flex, fast-flex and faster-flex, to fine-tuning baselines, ft-all and ft-fc. In each table, the first column lists each *source*→*target* domain shifts, with the base accuracy reached by the pretrained source network on the target test set. Bold entries indicate the score is better than that of *all* baselines (ft-). For space reason, we only report results for a specific subsampling ratio for settings other than PACS (roughly 30 images per class for MNIST, 20 for CIFAR, 12 for ILSVRC). Full results are in the supplemental material.

MNIST 	flex			ft-			
	flex	fast	faster	fc (1)	fc (2)	ss	all
Blurry (0.75) 	0.926	0.926	0.926	0.921	<b>0.928</b>	<b>0.928</b>	0.921
Occluded (0.58) 	<b>0.806</b>	<b>0.806</b>	0.801	0.785	0.801	0.792	<b>0.806</b>
MNIST-M (0.44) 	<b>0.683</b>	<b>0.683</b>	0.671	0.615	0.670	0.675	<b>0.683</b>
SVHN (0.21) 	<b>0.669</b>	<b>0.669</b>	0.572	0.451	0.595	0.657	<b>0.669</b>
Transf. (rnd) (0.32) 	<b>0.644</b>	<b>0.644</b>	<b>0.644</b>	0.573	0.638	0.624	0.625
Transf. (fix) (0.16) 	<b>0.908</b>	0.887	0.879	0.839	0.875	0.866	0.887

CIFAR 	flex			ft-			
	flex	fast	faster	fc (1)	fc (2)	ss	all
Blurry (0.32) 	<b>0.577</b>	<b>0.577</b>	0.512	0.444	0.501	0.569	<b>0.577</b>
Noisy (0.54) 	<b>0.624</b>	<b>0.624</b>	<b>0.624</b>	0.583	0.597	0.618	0.621
QuickDraw (0.29) 	0.518	0.517	0.517	0.475	<b>0.525</b>	0.495	0.501

QuickDraw 	flex			ft-			
	flex	fast	faster	fc (1)	fc (2)	ss	all
Blurry (0.19) 	0.642	0.631	0.560	0.426	0.468	<b>0.707</b>	0.631
Noisy (0.63) 	0.801	0.801	0.795	0.788	0.792	<b>0.805</b>	0.801
CIFAR (0.20) 	<b>0.424</b>	<b>0.424</b>	0.401	0.333	0.347	0.388	<b>0.424</b>

ILSVRC 	flex			ft-		
	flex	fast	faster	fc	ss	all
YUV (0.84) 	<b>0.893</b>	<b>0.893</b>	<b>0.893</b>	0.835	0.699	0.808
HSV (0.38) 	<b>0.856</b>	<b>0.856</b>	<b>0.856</b>	0.533	0.646	0.687
Scaling (stretch) (0.44) 	<b>0.724</b>	<b>0.696</b>	<b>0.696</b>	0.502	0.584	0.653
Scaling (sym.) (0.52) 	<b>0.770</b>	<b>0.757</b>	<b>0.757</b>	0.663	0.650	0.716
Rotation (0.74) 	<b>0.826</b>	<b>0.832</b>	<b>0.812</b>	0.667	0.652	0.771

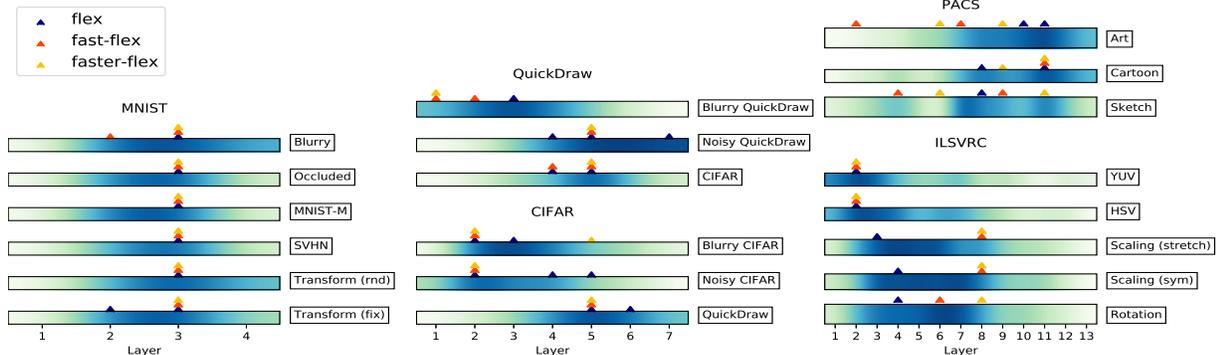


Figure 2. Individual units selected by flex, fast-flex and faster-flex, based on *validation* set accuracy. Triangles denote actual picks for flex, fast-flex (if ignoring the option to fine-tune all units) and faster-flex. The background values is obtained by summing the selection ranks of each unit across ratios, based on their *test performance*: in other words, the darker the color, the best performance fine-tuning this unit yields on the test set. We observe that flex-tuning’s selection criterion generally chooses the best performing unit. The two variants’ choices are more scattered, but overall positively correlate with flex-tuning’s decisions.

they often pick similar units. Similarly in Table 3 we observe that they both often outperform fine-tuning baselines, although still being somewhat subpar to flex. This shows that only a few gradient updates, as is done in even faster flex-tuning, are enough to pin-point relevant units.

**Effect of the domain shift on flex.** From Figure 2, we distinguish three input domain shifts categories: For local *pixel-level transformations*, such as noisy CIFAR, or YUV/HSV ILSVRC, flex-tuning tends to choose early units. This coincides with the fact that (i) early layers are most affected by local pixel-level changes, and (ii) such transformations are easy to correct in early layers: *e.g.* YUV is a linear transformation of RGB. For *geometric affine*

*transformations*, flex-tuning picks more central units of the architecture. In fact, such transformations do not change the global appearance of images and, moreover, most modern deep learning architecture are trained for invariance to small geometric manipulations (*e.g.* flip, rotations) via synthetic data augmentation, hence earlier layers are more easily transferable across these domain shifts. The free-transform scenarios are harder to generalize: First, we observe that natural images features transfer particularly well across various domains. As such, flex-tuning often picks later layer in the architecture for general transforms scenarios with natural images as their source domain, *e.g.* *photo*→*{art, cartoon, sketch}*. However, this does not seem to be

	YUV 	HSV 	Art 	Cartoon 	Sketch 
ft-fc	0.80	0.47	0.66	0.53	0.39
ft-all	0.75	0.52	0.80	0.83	0.85
flex	0.85	0.85	0.88	0.86	0.86

Table 4. mAP@10 retrieval results for the fine- and flex-tuned network embeddings, queried against the source domain embeddings

the case in the reverse scenario, *e.g.* *QuickDraw*  $\rightarrow$  *CIFAR* and *MNIST*  $\rightarrow$  *SVHN*, which indicates that features learned from the simple structure and particular distribution of binary sketches do not generalize as well to natural images. Second, in some complex settings such as PACS, it can be the case that *two non-consecutive units* are good fine-tuning candidates. This suggests that units sometimes interact in complex patterns and that considering combination of units rather than single ones is an interesting future direction.

### 5.3. Retrieval Experiments

A benefit of fine-tuning the last layer only is that it preserves a common feature representation across domains. However this property breaks in our setting: Images visually different from the training set fall out of the usual operation zone of the feature extractor. One can still learn a good classifier from these features [30], but the representations themselves are meaningless with respect to the initial source domain. On the other hand, tuning an intermediate unit instead could help to “mend” the representation. To evaluate this, we use a retrieval experiment: We extract features for the initial source validation domain through the source network, and for the target domain through the flex-tuned or finetuned network. For each target sample, we retrieve its top- $k$  nearest neighbors in the source domain and consider them correctly retrieved if they share the same semantic class, and evaluate the average precision (AP@ $k$ ). For space reasons, we only report AP@10, on the most challenging scenarios, in Table 4. Full results are reported in the supplemental material. The results follow our previous observations: For small networks (MNIST, CIFAR) the result of fine-tuning all layers is often better aligned with the initial representations. However for the larger architectures, tuning an intermediate unit better recover the initial source embedding space as shown in Table 4.

### 5.4. Towards pixel-level adaptation

An alternative to tuning a pre-trained network is to instead learn to map target samples back to the source domain while keeping the network’s weights untouched; This has the advantage of only depending on the domain shift and not on the architecture. Such *image-to-image mapping* modules have been studied for domain adaption, but typically require data from both source and target domains [4, 43].

Building on this idea, we introduce an *image-to-image transformation unit* as a pre-processing module before the feature extraction phase of the pre-trained source network.

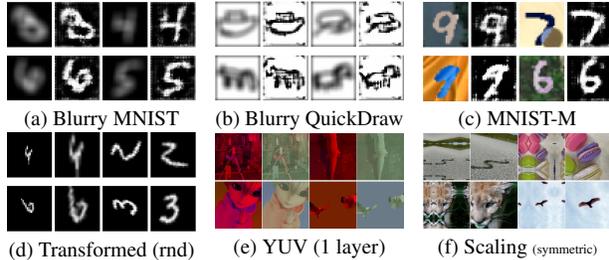


Figure 3. Example images generated by the pre-processing module. For each pair, the left image is the input from the target domain and the right one is the pre-processed output.

The resulting architecture is considered as a new model selection option for flex-tuning, where only the image-to-image unit’s weights are trained and the rest of the network is frozen. We implement this image-to-image unit as a small Pix2Pix network [14] except in a few scenarios where we leverage our prior knowledge of the domain shift: For example, color channel transformations occur pixel-wise, thus we build the preprocessing module for YUV and HSV ILSVRC with 1x1 convolutions. Similarly, for geometric transforms, we use a Spatial Transformer Network [15]. Figure 3 shows exemplary outputs of the learned image-to-image units. Quantitative results are in the supplemental material. The specialized pre-processing modules performs very well for simple parametric transformations, and results are also encouraging on simple domain shifts such as *blur*, *noise* and added *random background*. We believe this to be a consequence of the skip connections in the Pix2Pix architecture, which enforce local pixel constraints between the input and output. In all these successful cases, flex-tuning’s selection criterion also selects the image-to-image unit as the most promising unit to tune. For complex transformations, *e.g.* *photo*  $\rightarrow$  *sketch*, the pre-processing module performs poorly. Nevertheless, flex-tuning is able to notice this and falls back to one of the other units to adapt.

## 6. Conclusions

We introduce a new transfer learning method for neural networks, *flex-tuning*, that adapts a pre-trained network to a new domain by tuning just a single network unit (*e.g.* a layer or block layers). Our experiments on a variety of scenarios show that this is a surprisingly strong adaptation technique, as long as the right unit is chosen. Specifically, we study the case where output classes stay consistent but the input data characteristics change, potentially dramatically, *e.g.* from images to sketch drawings. We find that, contrary to common practice, it is then rarely the last fully-connected unit, but rather an intermediate or early unit, that leads to the best adaptation results, and *flex-tuning* reliably identifies it. We also introduce two accelerated variants that perform almost equally good but are significantly more computationally efficient in selecting the unit to be fine-tuned.

## References

- [1] H. Azizpour, A. S. Razavian, J. Sullivan, A. Maki, and S. Carlsson. Factors of transferability for a generic ConvNet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 2016. 2
- [2] A. Babenko and V. S. Lempitsky. Aggregating local deep features for image retrieval. In *International Conference on Computer Vision (ICCV)*, 2015. 2
- [3] P. Ballester and R. M. Araujo. On the performance of GoogLeNet and AlexNet applied to sketches. In *Conference on Artificial Intelligence (AAAI)*, 2016. 1
- [4] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2, 8
- [5] S. A. Cadena, M. A. Weis, L. A. Gatys, M. Bethge, and A. S. Ecker. Diverse feature visualizations reveal invariances in early layers of deep neural networks. In *European Conference on Computer Vision (ECCV)*, 2018. 2
- [6] S. Cheema, S. Gulwani, and J. LaViola. QuickDraw: Improving drawing experience for geometric diagrams. In *Conference on Human Factors in Computing Systems (SIGCHI)*, 2012. 3, 5, 6
- [7] B. Chu, V. Madhavan, O. Beijbom, J. Hoffman, and T. Darrell. Best practices for fine-tuning visual classifiers to new domains. In *ECCV Workshop TASK-CV: Transferring and Adapting Source Knowledge in Computer Vision*, 2016. 2
- [8] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. DeCAF: A deep convolutional activation feature for generic visual recognition. In *International Conference on Machine Learning (ICML)*, 2014. 2
- [9] Y. Ganin and V. Lempitsky. Unsupervised domain adaptation by backpropagation. In *International Conference on Machine Learning (ICML)*, 2015. 6
- [10] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. Domain-adversarial training of neural networks. *Journal of Machine Learning Research (JMLR)*, 2016. 2
- [11] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 2
- [12] R. Gopalan, R. Li, and R. Chellappa. Domain adaptation for object recognition: An unsupervised approach. In *International Conference on Computer Vision (ICCV)*, 2011. 2
- [13] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Journal of Machine Learning Research (JMLR)*, 2015. 5
- [14] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2, 8
- [15] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. In *Conference on Neural Information Processing Systems (NIPS)*, 2015. 8
- [16] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision (ECCV)*, 2016. 2
- [17] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim. Learning to discover cross-domain relations with generative adversarial networks. In *International Conference on Machine Learning (ICML)*, 2017. 2
- [18] D. P. Kingma and J. L. Ba. Adam: a method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. 5
- [19] S. Kornblith, J. Shlens, and Q. V. Le. Do better ImageNet models transfer better? In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 1
- [20] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. 3, 5, 6
- [21] G. Larsson, M. Maire, and G. Shakhnarovich. Learning representations for automatic colorization. In *European Conference on Computer Vision (ECCV)*, 2016. 2
- [22] Y. LeCun and C. Cortes. MNIST handwritten digit database. <http://yann.lecun.com/exdb/mnist/>. 1, 6
- [23] D. Li, Y. Yang, Y.-Z. Song, and T. M. Hospedales. Deeper, broader and artier domain generalization. In *International Conference on Computer Vision (ICCV)*, 2017. 5
- [24] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 2
- [25] Deep Learning Model Zoo. <https://modelzoo.co/>. 1
- [26] 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*, 2011. 6
- [27] C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2017. <https://distill.pub/2017/feature-visualization>. 2
- [28] N. Papernot, M. Abadi, Ú. Erlingsson, I. J. Goodfellow, and K. Talwar. Semi-supervised knowledge transfer for deep learning from private training data. In *International Conference on Learning Representations (ICLR)*, 2017. 2
- [29] G. Parascandolo, N. Kilbertus, M. Rojas-Carulla, and B. Schölkopf. Learning independent causal mechanisms. In *International Conference on Machine Learning (ICML)*, 2018. 2
- [30] A. Rahimi and B. Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Conference on Neural Information Processing Systems (NIPS)*, 2008. 8
- [31] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 2
- [32] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 2015. 1, 2, 6

- [33] K. Saenko, B. Kulis, M. Fritz, and T. Darrell. Adapting visual category models to new domains. In *European Conference on Computer Vision (ECCV)*, 2010. 2
- [34] S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014. 3
- [35] M. Simon and E. Rodner. Neural activation constellations: Unsupervised part model discovery with convolutional networks. In *International Conference on Computer Vision (ICCV)*, 2015. 2
- [36] Q. Sun, Y. Liu, T.-S. Chua, and B. Schiele. Meta-transfer learning for few-shot learning. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 5
- [37] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 5
- [38] TensorFlow Hub. <https://www.tensorflow.org/hub/>. 1
- [39] TensorNets. <https://github.com/taehoonlee/tensornets>. 1, 5
- [40] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Conference on Neural Information Processing Systems (NIPS)*, 2014. 2
- [41] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision (ECCV)*, 2014. 2
- [42] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2
- [43] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *International Conference on Computer Vision (ICCV)*, 2017. 2, 8