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

SpotTune: Transfer Learning through Adaptive Fine-tuning

Yunhui Guo[*1,2], Honghui Shi[1], Abhishek Kumar[†], Kristen Grauman[3], Tajana Rosing[2], Rogerio Feris[1]

[1]IBM Research & MIT-IBM Watson AI Lab, [2]University of California, San Diego, [3]The University of Texas at Austin

Abstract

Transfer learning, which allows a source task to affect the inductive bias of the target task, is widely used in computer vision. The typical way of conducting transfer learning with deep neural networks is to fine-tune a model pre-trained on the source task using data from the target task. In this paper, we propose an adaptive fine-tuning approach, called SpotTune, which finds the optimal fine-tuning strategy per instance *for the target data. In SpotTune, given an image from the target task, a policy network is used to make routing decisions on whether to pass the image through the fine-tuned layers or the pre-trained layers. We conduct extensive experiments to demonstrate the effectiveness of the proposed approach. Our method outperforms the traditional fine-tuning approach on 12 out of 14 standard datasets. We also compare SpotTune with other state-of-the-art fine-tuning strategies, showing superior performance. On the* Visual Decathlon *datasets, our method achieves the highest score across the board without bells and whistles.*

1. Introduction

Deep learning has shown remarkable success in many computer vision tasks, but current methods often rely on large amounts of labeled training data [22, 15, 16]. *Transfer learning*, where the goal is to transfer knowledge from a related *source task*, is commonly used to compensate for the lack of sufficient training data in the *target task* [35, 3]. *Fine-tuning* is arguably the most widely used approach for transfer learning when working with deep learning models. It starts with a pre-trained model on the source task and trains it further on the target task. For computer vision tasks, it is a common practice to work with ImageNet pre-trained models for fine-tuning [20]. Compared with training from scratch, fine-tuning a pre-trained convolutional neural network on a target dataset can significantly improve performance, while reducing the target labeled data requirements [14, 51, 44, 20].

[*]This work was done when Yunhui Guo was an intern at IBM Research.
[†]Abhishek Kumar is now with Google Brain. The work was done when he was at IBM Research.

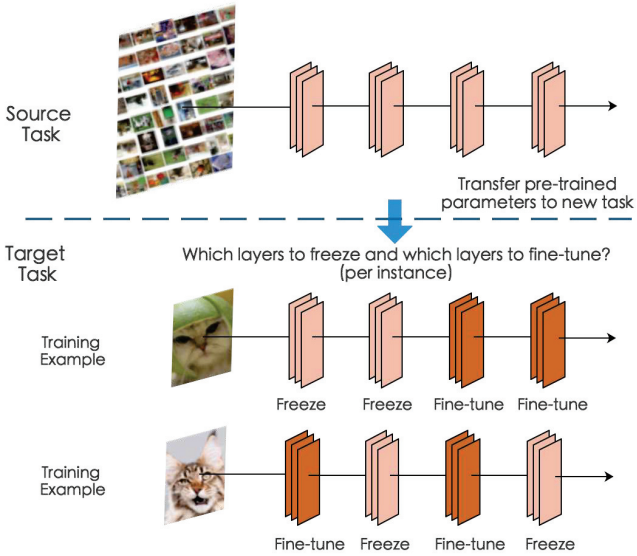

Figure 1: Given a deep neural network pre-trained on a source task, we address the question of *where to fine-tune* its parameters with examples of the target task. We propose a novel method that decides, per training example, which layers of the pre-trained model should have their parameters fixed, i.e., shared with the source task, and which layers should be fine-tuned to improve the accuracy of the model in the target domain.

There are several choices when it comes to realizing the idea of fine-tuning of deep networks in practice. A natural approach is to optimize *all* the parameters of the deep network using the target training data (after initializing them with the parameters of the pre-trained model). However, if the target dataset is small and the number of parameters is huge, fine-tuning the whole network may result in overfitting [51]. Alternatively, the last few layers of the deep network can be fine-tuned while freezing the parameters of the remaining initial layers to their pre-trained values [44, 1]. This is driven by a combination of limited training data in the target task and the empirical evidence that initial layers learn low-level features that can be directly shared across various computer vision tasks. However, the number of initial layers to freeze during fine-tuning still remains a man-

ual design choice which can be inefficient to optimize for, especially for networks with hundreds or thousands of layers. Further, it has been empirically observed that current successful multi-path deep architectures such as ResNets [15] behave like ensembles of shallow networks [47]. It is not clear if restricting the fine-tuning to the last contiguous layers is the best option, as the ensemble effect diminishes the assumption that early or middle layers should be shared with common low-level or mid-level features.

Current methods also employ a *global fine-tuning* strategy, *i.e.*, the same decision of which parameters to freeze vs. fine-tune is taken for all the examples in the target task. The assumption is that such a decision is optimal for the entire target data distribution, which may not be true, particularly in the case of insufficient target training data. For example, certain classes in the target task might have higher similarity with the source task, and routing these target examples through the source pre-trained parameters (during inference) might be a better choice in terms of accuracy. Ideally, we would like these decisions to be made individually for each layer (*i.e.*, whether to use pre-trained parameters or fine-tuned parameters for that layer), per input example, as illustrated in Figure 1.

In this paper, we propose *SpotTune*, an approach to learn a decision policy for input-dependent fine-tuning. The policy is sampled from a discrete distribution parameterized by the output of a lightweight neural network, which decides which layers of a pre-trained model should be fine-tuned or have their parameters frozen, on a per instance basis. As these decision functions are discrete and non-differentiable, we rely on a recent Gumbel Softmax sampling approach [30, 18] to train the policy network. At test time, the policy decides whether the features coming out of a layer go into the next layer with the source pre-trained parameters or the fine-tuned parameters.

We summarize our contributions as follows:

- We propose an input-dependent fine-tuning approach that automatically determines which layers to fine-tune per target instance. This is in contrast to current fine-tuning methods which are mostly ad-hoc in terms of determining *where to fine-tune* in a deep neural network (*e.g.*, fine-tuning last k layers).

- We also propose a global variant of our approach that constrains all the input examples to fine-tune the same set of k layers which can be distributed anywhere in the network. This variant results in fewer parameters in the final model as the corresponding set of pre-trained layers can be discarded.

- We conduct extensive empirical evaluation of the proposed approach, comparing it with several competitive baselines. The proposed approach outperforms standard fine-tuning on 12 out of 14 datasets. Moreover, we show the effectiveness of *SpotTune* compared to other state-of-the-art fine-tuning strategies. On the Visual Decathlon Challenge [37], which is a competitive benchmark for testing the performance of multi-domain learning algorithms with a total of 10 datasets, the proposed approach achieves the highest score compared with the state-of-the-art methods.

2. Related Work

Transfer Learning. There is a long history of transfer learning and domain adaptation methods in computer vision [8, 35]. Recently, transfer learning based on deep neural networks has received significant attention in the community [12, 6, 7, 24, 13]. Fine-tuning a pre-trained network model such as ImageNet on a new dataset is the most common strategy for knowledge transfer in the context of deep learning. Methods have been proposed to fine-tune all network parameters [14], only the parameters of the last few layers [28], or to just use the pre-trained model as a fixed feature extractor with a classifier such as SVM on top [42]. Kornblith et al. [20] studied several of these options to address the question of whether better ImageNet models transfer better. Yosinski et al. [51] conducted a study on the impact of transferability of features from the bottom, middle, or top of the network with early models, but it is not clear whether their conclusions hold for modern multi-path architectures such as Residual Networks [15] or DenseNets [16]. Yang et al. [50] have recently proposed to learn relational graphs as transferable representations, instead of unary features. Closely related to our work, Li et al. [25] investigated several regularization schemes that explicitly promote the similarity of the fine-tuned model with the original pre-trained model. Different from all these methods, our proposed approach automatically decides the optimal set of layers to fine-tune in a pre-trained model on a new task. In addition, we make this decision on a *per-instance* basis.

Feature Sharing Across Tasks. In the multi-task setting, knowing which tasks or parameters are shareable is a longstanding challenge [19, 23, 45, 29]. Early methods were designed for shallow classification models [52, 17, 36], while more recent approaches address the problem of "with whom" each task should share features using deep neural networks [29, 32]. Cross-stitching networks [33] and Progressive Networks [40] have been recently proposed to learn an optimal combination of shared and task-specific representations for joint multi-task optimization and life-long learning, respectively. These methods rely on per-layer inter-column adapters, which requires more memory and leads to more computational cost. In addition, they learn global feature adapters *per task*, whereas SpotTune adaptively routes computation *per input example*, which is important to boost accuracy.

Dynamic Routing. Our proposed approach is related to *conditional computation* methods [4, 27, 11], which aim to dynamically route information in neural networks with the goal of improving computational efficiency. Bengio et al. [2] used sparse activation policies to selectively execute neural network units on a per-example basis. Shazeer et al. [43] introduced a Sparsely-Gated Mixture-of-Experts layer, where a trainable gating network determines a sparse combination of sub-networks (experts) to use for each example. Wu, Nagarajan et al. proposed *BlockDrop* [49], a method that uses reinforcement learning to dynamically select which layers of a Residual Network to execute, exploiting the fact that ResNets are resilient to layer dropping [47]. Veit and Belongie [46] investigated the same idea using Gumbel Softmax [18] for on-the-fly selection of residual blocks. Our work also explores dynamic routing based on the *Gumbel trick*. However, unlike previous methods, our goal is to determine the parameters in a neural network that should be frozen or fine-tuned during learning to improve accuracy, instead of dropping layers to improve efficiency.

3. Proposed Approach

Given a pre-trained network model on a source task (e.g., ImageNet pre-trained model), and a set of training examples with associated labels in the target domain, our goal is to create an adaptive fine-tuning strategy that decides, per training example, which layers of the pre-trained model should be fine-tuned (adapted to the target task) and which layers should have their parameters frozen (shared with the source task) during training, in order to improve the accuracy of the model in the target domain. To this end, we first present an overview of our approach in Section 3.1. Then, we show how we learn our adaptive fine-tuning policy using Gumbel Softmax sampling in Section 3.2. Finally, in Section 3.3, we present a global policy variant of our proposed image-dependent fine-tuning method, which constrains all the images to follow a single fine-tuning policy.

3.1. SpotTune Overview

Although our approach could be applied to different deep neural network architectures, in the following we focus on a Residual Network model (ResNet) [15]. Recently, it has been shown that ResNets behave as ensembles of shallow classifiers and are resilient to residual block swapping [47]. This is a desirable property for our approach, as later we show that SpotTune dynamically swaps pre-trained and fine-tuned blocks to improve performance.

Consider the l-th residual block in a pre-trained ResNet model:

$$x_l = F_l(x_{l-1}) + x_{l-1}. \quad (1)$$

In order to decide whether or not to fine-tune a residual block during training, we *freeze* the original block F_l and create a new *trainable* block $\hat{F}_l$, which is initialized with the parameters of F_l. With the additional block $\hat{F}_l$, the output of the l-th residual block in SpotTune is computed as below:

$$x_l = I_l(x)\hat{F}_l(x_{l-1}) + (1 - I_l(x))F_l(x_{l-1}) + x_{l-1} \quad (2)$$

where $I_l(x)$ is a binary random variable that indicates whether the residual block should be frozen or fine-tuned, conditioned on the input image. During training, given an input image x, the *frozen* block F_l trained on the source task is left unchanged and the replicated block $\hat{F}_l$, which is initialized from F_l, can be optimized towards the target dataset. Hence, the given image x can either share the *frozen* block F_l, which allows the features computed on the source task to be reused, or fine-tune the block $\hat{F}_l$, which allows x to use the adapted features. $I_l(x)$ is sampled from a discrete distribution with two categories (freeze or fine-tune), which is parameterized by the output of a lightweight policy network. More specifically, if $I_l(x) = 0$, then the l-th frozen block is re-used. Otherwise, if $I_l(x) = 1$ the l-th residual block is fine-tuned by optimizing $\hat{F}_l$.

Figure 2 illustrates the architecture of our proposed *SpotTune* method, which allows each training image to have its own fine-tuning policy. During training, the policy network is jointly trained with the target classification task using Gumbel Softmax sampling, as we will describe next. At test time, an input image is first fed into a policy network, whose output is sampled to produce routing decisions on whether to pass the image through the fine-tuned or pre-trained residual blocks. The image is then routed through the corresponding residual blocks to produce the final classification prediction. Note that the effective number of executed residual blocks is the same as the original pre-trained model. The only additional computational cost is incurred by the policy network, which is designed to be lightweight (only a few residual blocks) in comparison to the original pre-trained model.

3.2. Training with the Gumbel Softmax Policy

SpotTune makes decisions as to whether or not to freeze or fine-tune each residual block per training example. However, the fact that the policy $I_l(x)$ is discrete makes the network non-differentiable and therefore difficult to be optimized with backpropagation. There are several ways that allow us to "back-propagate" through the discrete nodes [4]. In this paper, we use a recently proposed Gumbel Softmax sampling approach [30, 18] to circumvent this problem.

The Gumbel-Max trick [30] is a simple and effective way to draw samples from a categorical distribution parameterized by $\{\alpha_1, \alpha_2, ..., \alpha_z\}$, where α_i are scalars not confined to the simplex, and z is the number of categories. In our work, we consider two categories (freeze or fine-tune), so $z = 2$, and for each residual block, α_1 and α_2 are scalars corresponding to the output of a policy network.

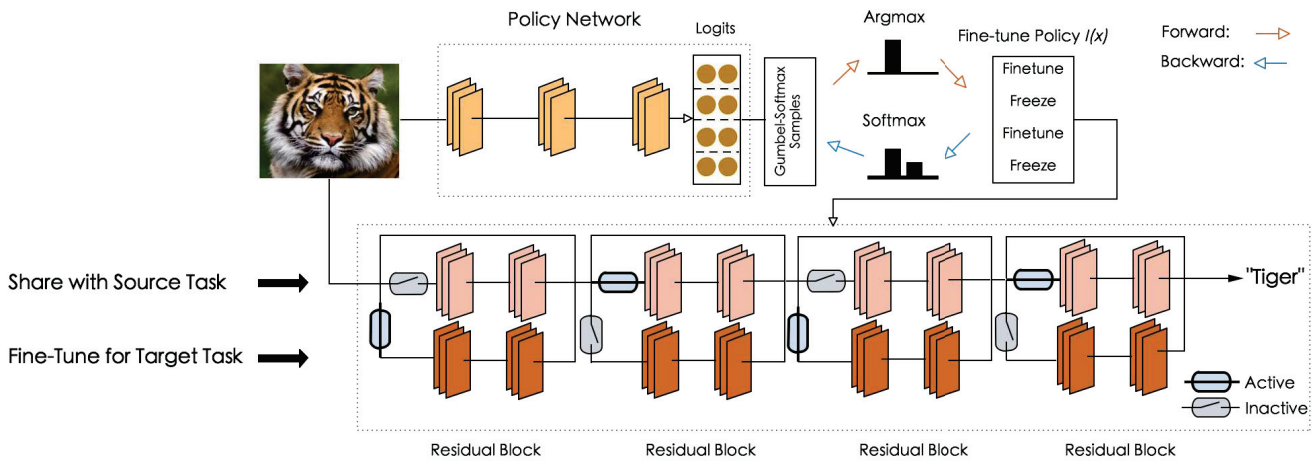

Figure 2: Illustration of our proposed approach. The policy network is trained to output routing decisions (fine-tune or freeze parameters) for each block in a ResNet pre-trained on the source dataset. During learning, the fine-tune vs. freeze decisions are generated based on a Gumbel Softmax distribution, which allows us to optimize the policy network using backpropagation. At test time, given an input image, for each residual block the computation is routed so that either the fine-tuned path or the frozen path is activated.

A random variable G is said to have a standard Gumbel distribution if $G = -\log(-\log(U))$ with U sampled from a uniform distribution, i.e. $U \sim Unif[0, 1]$. Based on the Gumbel-Max trick [30], we can draw samples from a discrete distribution parameterized by α_i in the following way: we first draw i.i.d samples $G_i, ..., G_z$ from $Gumbel(0, 1)$ and then generate the discrete sample as follows:

$$X = \arg\max_i [\log \alpha_i + G_i]. \tag{3}$$

The $\arg\max$ operation in Equation 3 is non-differentiable. However, we can use the Gumbel Softmax distribution [30, 18], which adopts softmax as a continuous relaxation to $\arg\max$. We represent X as a one-hot vector where the index of the non-zero entry of the vector is equal to X, and relax the one-hot encoding of X to a z-dimensional real-valued vector Y using softmax:

$$Y_i = \frac{\exp((\log \alpha_i + G_i)/\tau)}{\sum_{j=1}^{z} \exp((\log \alpha_j + G_j)/\tau)} \quad \text{for } i = 1, .., z \tag{4}$$

where τ is a temperature parameter, which controls the discreteness of the output vector Y. When τ becomes closer to 0, the samples from the Gumbel Softmax distribution become indistinguishable from the discrete distribution (i.e, almost the same as the one-hot vector).

Sampling our fine-tuning policy $I_l(x)$ from a Gumbel Softmax distribution parameterized by the output of a policy network allows us to backpropagate from the discrete freeze/fine-tune decision samples to the policy network, as the Gumbel Softmax distribution is smooth for $\tau > 0$ and therefore has well-defined gradients with respect to the parameters α_i. By using a standard classification loss l_c for the target task, the policy network is jointly trained with the pre-trained model to find the optimal fine-tuning strategy that maximizes the accuracy of the target task.

Similar to [49], we generate all freeze/fine-tune decisions for all residual blocks at once, instead of relying on features of intermediate layers of the pre-trained model to obtain the fine-tuning policy. More specifically, suppose there are L residual blocks in the pre-trained model. The output of the policy network is a two-dimensional matrix $\beta \in \mathbb{R}^{L \times 2}$. Each row of β represents the logits of a Gumbel-Softmax Distribution with two categories, i.e, $\beta_{l,0} = \log \alpha_1$ and $\beta_{l,1} = \log \alpha_2$. After obtaining β, we use the straight-through version of the Gumbel-Softmax estimator [18]. During the forward pass, we sample the fine-tuning policy $I_l(x)$ using Equation 3 for the l-th residual block. During the backward pass, we approximate the gradient of the discrete samples by computing the gradient of the continuous softmax relaxation in Equation 4. This process is illustrated in Figure 2.

3.3. Compact Global Policy Variant

In this section, we consider a simple extension of the image-specific fine-tuning policy, which constrains all the images to fine-tune the same k blocks that can be distributed anywhere in the ResNet. This variant reduces both the memory footprint and computational costs, as k can be set to a small number so most blocks are shared with the source task, and at test time the policy network is not needed.

Consider a pre-trained ResNet model with L residual blocks. For the l-th block, we can obtain the number of images that use the fine-tuned block and the pre-trained block

based on the image-specific policy. We compute the fraction of images in the target dataset that uses the fine-tuned block and denote it as $v_l \in [0, 1]$. In order to constrain our method to fine-tune k blocks, we introduce the following loss:

$$l_k = ((\sum_{l=1}^{L} v_l) - k)^2. \tag{5}$$

Moreover, in order to achieve a deterministic policy, we add another loss l_e:

$$l_e = \sum_{l=1}^{L} -v_l \log v_l. \tag{6}$$

The additional loss l_e pushes v_l to be exactly 0 or 1, so that a global policy can be obtained for all the images. The final loss is defined below:

$$l = l_c + \lambda_1 l_k + \lambda_2 l_e, \tag{7}$$

where l_c is the classification loss, λ_1 is the balance parameter for l_k, and λ_2 is the the balance parameter for l_e. The additional losses push the policy network to learn a global policy for all the images. As opposed to manually selecting k blocks to fine-tune, the global-k variant *learns* the k blocks that can achieve the best accuracy on the target dataset. We leave for future work the task of finding the optimal k, which could be achieved e.g., by using reinforcement learning with a reward proportional to accuracy and inversely proportional to the number of fine-tuned blocks.

4. Experiments

4.1. Experimental Setup

Datasets and metrics. We compare our SpotTune method with other fine-tuning and regularization techniques on 5 public datasets, including three fine-grained classification benchmarks: CUBS [48], Stanford Cars [21] and Flowers [34], and two datasets with a large domain mismatch from ImageNet: Sketches [10] and WikiArt [41]. The statistics of these datasets are listed in Table 1. Performance is measured by classification accuracy on the evaluation set.

We also report results on the datasets of the Visual Decathlon Challenge [37], which aims at evaluating visual recognition algorithms on images from multiple visual domains. There are a total of 10 datasets as part of this challenge: (1) ImageNet, (2) Aircraft, (3) CIFAR-100, (4) Describable textures, (5) Daimler pedestrian classification, (6) German traffic signs, (7) UCF-101 Dynamic Images, (8) SVHN, (9) Omniglot, and (10) Flowers. The images of the Visual Decathlon datasets are resized isotropically to have a shorter side of 72 pixels, in order to alleviate the computational burden for evaluation. Following [37], the performance is measured by a single scalar score

Dataset	Training	Evaluation	Classes
CUBS	5,994	5,794	200
Stanford Cars	8,144	8,041	196
Flowers	2,040	6,149	102
Sketch	16,000	4,000	250
WikiArt	42,129	10,628	195

Table 1: Datasets used to evaluate SpotTune against other fine-tuning baselines.

$S = \sum_{i=1}^{10} \alpha_i \text{max}\{0, E_i^{\text{max}} - E_i\}^2$, where E_i is the test error on domain D_i, and E_i^{max} is the error of a reasonable baseline algorithm. The coefficient α_i is $1000(E_i^{\text{max}})^{-2}$, so a perfect classifier receives score 1000. The maximum score achieved across 10 domains is 10000. Compared with average accuracy across all the 10 domains, the score S is a more reasonable measurement for comparing different algorithms, since it considers the difficulty of different domains, which is not captured by the average accuracy [37].

In total, our experiments comprise 14 datasets, as the Flowers dataset is listed in both sets described above. We note that for the experiments in Table 2, we use the full resolution of the images, while those are resized in the Visual Decathlon experiments to be consistent with other approaches.

Baselines. We compare SpotTune with the following fine-tuning and regularization techniques:

- **Standard Fine-tuning**: This baseline fine-tunes all the parameters of the pre-trained network on the target dataset [14, 51].
- **Feature Extractor**: We use the pre-trained network as a feature extractor [42, 9] and only add the classification layer for each newly added dataset.
- **Stochastic Fine-tuning**: We randomly sample 50% of the blocks of the pre-trained network to fine-tune.
- **Fine-tuning last-k** (k = 1, 2, 3): This baseline fine-tunes the last k residual blocks of the pre-trained network on the target dataset [28, 44, 1]. In our experiments, we consider fine-tuning the last one (k = 1), last two (k = 2) and the last three (k = 3) residual blocks.
- **Fine-tuning ResNet-101**: We fine-tune all the parameters of a pre-trained ResNet-101 model on the target dataset. SpotTune uses ResNet-50 instead (for the experiments in Table 2), so this baseline is more computationally expensive and can fine-tune twice as many residual blocks. We include it as the total number of parameters during training is similar to SpotTune, so it will verify any advantage is not merely due to our having 2x residual blocks available.

Model	CUBS	Stanford Cars	Flowers	WikiArt	Sketches
Feature Extractor	74.07%	70.81%	85.67%	61.60%	75.50%
Standard Fine-tuning	81.86%	89.74%	93.67%	75.60%	79.58%
Stochastic Fine-tuning	81.03%	88.94%	92.95%	73.06%	78.30%
Fine-tuning last-3	81.54%	88.21%	89.03%	72.68 %	77.72%
Fine-tuning last-2	80.34%	85.36%	91.81%	70.82%	78.37%
Fine-tuning last-1	78.68%	81.73%	89.99%	68.96%	77.20%
Random Policy	81.63 %	88.57%	93.44%	73.82%	78.30%
Fine-tuning ResNet-101	82.13%	90.32%	94.21%	**76.52%**	78.92%
L^2-SP	83.69%	91.08%	95.21%	75.38%	79.60%
Progressive Neural Nets	83.08 %	91.59%	95.55%	75.41%	79.71%
SpotTune (running fine-tuned blocks)	82.36%	92.04%	93.49%	67.27%	78.88%
SpotTune (Global-k)	83.48%	90.51%	**96.60%**	75.63%	80.02%
SpotTune	**84.03** %	**92.40%**	96.34%	75.77%	**80.20%**

Table 2: Results of *SpotTune* and baselines on CUBS, Stanford Cars, Flowers, WikiArt and Sketches.

- **Random Policy**: This baseline method adopts a random policy network that always finetunes the last three layers and randomly decides whether to fine-tune or not for each training sample for other layers.
- **L^2-SP** [25]: This is a recently proposed state-of-the-art regularization method for fine-tuning. The authors recommend using an L^2 penalty to allow the fine-tuned network to have an explicit inductive bias towards the pre-trained model, sharing similar motivation with our approach.
- **Progressive Neural Networks** [40]: This is a recent method which learns an optimal combination of shared and task-specific representations for life-long learning. Different form the original work, which uses a random weight initialization, we use an ImageNet pre-trained model as the frozen source network, since the former leads to much worse performance for classification.

Regarding the methods that have reported results on the Visual Decathlon datasets, the most related to our work are models trained from *Scratch*, *Standard Fine-tuning*, the *Feature Extractor* baseline as described above, and *Learning without Forgetting (LwF)* [26], which is a recently proposed technique that encourages the fine-tuned network to retain the performance on ImageNet or previous tasks, while learning consecutive tasks. Other methods include *Piggyback* [31], *Residual Adapters* and its variants [37, 38], *Deep Adaptation Networks (DAN)* [39], and *Batch Norm Adaptation (BN Adapt)* [5], which are explicitly designed to minimize the number of model parameters, while our method sits at the other end of the spectrum, with a focus on accuracy instead of parameter reduction. We also compare with training from scratch using Residual Adapters (*Scratch+*), as well as the high-capacity version of Residual Adapters described in [37], which have a similar number of parameters as SpotTune.

Pre-trained model. For comparing SpotTune with fine-tuning baselines in Table 2, we use ResNet-50 pre-trained on ImageNet, which starts with a convolutional layer followed by 16 residual blocks. The residual blocks contain three convolutional layers and are distributed into 4 segments (i.e, [3, 4, 6, 3]) with downsampling layers in between. We use the pre-trained model from Pytorch which has a classification accuracy of 75.15% on ImageNet. For the Visual Decathlon Challenge, we use a ResNet-26 as described in [38].

Policy network architecture. For the experiments with ResNet-50 (Table 2), we use a ResNet with 4 blocks for the policy network. The channel size of each block is 64, 128, 256, 512, respectively. For the Visual Decathlon Challenge with ResNet-26, the policy network consists of a ResNet with 3 blocks. The channel size of each block is 64, 128, 256, respectively.

Implementations details. We use SGD with momentum as the optimizer. For the Visual Decathlon Challenge, we freeze the first macro blocks (4 residual blocks) of the ResNet-26 and only apply the adaptive fine-tuning for the rest of the residual blocks. This choice reduces the number of parameters and has a regularization effect.

4.2. Results and Analysis

4.2.1 SpotTune vs. Fine-tuning Baselines

The results of SpotTune and the fine-tuning baselines are listed in Table 2. Clearly, SpotTune yields consistently better results than other methods. Using the pre-trained model on ImageNet as a *feature extractor* (with all parameters frozen) can reduce the number of parameters when the model is applied to a new dataset, but it leads to bad performance due to the domain shift. All the fine-tuning variants (*Standard Fine-tuning, Stochastic Fine-tuning, Fine-tuning last-k*) achieve higher accuracy than the *Feature Extractor* baseline, as expected. Note that the results of *Fine-tuning last-k* show that manually deciding the number of layers

to fine-tune may lead to worse results than *standard fine-tuning*. The *Fine-tuned ResNet-101* has higher capacity and thus performs better than the other fine-tuning variants. Although it has twice as many fine-tuned blocks and is significantly more computationally expensive than SpotTune, it still performs worse than our method in all datasets, except in WikiArt. We conjecture this is because WikiArt has more training examples than the other datasets. To test this hypothesis, we evaluated both models when 25% of the WikiArt training data is used. In this setting, SpotTune achieves 61.24% accuracy compared to 60.20% of the fine-tuned ResNet-101. This gap increases even more when 10% of the data is considered (49.59% vs. 47.05%).

By inducing the fine-tuned models to be close to the pre-trained model, L^2-SP achieves better results than other fine-tuning variants, but it is inferior to SpotTune in all datasets. However, we note that L^2-SP is complementary to SpotTune and can be combined with it to further improve results. Compared with *Progressive Neural Networks*, SpotTune is faster, requires less memory, and achieves more accuracy by adaptively routing computation per input example.

SpotTune is different from all the baselines in two aspects. On one hand, the fine-tuning policy in SpotTune is specialized for each instance in the target dataset. This implicitly takes the similarities between the images in the target dataset and the source dataset into account. On the other hand, sharing layers with the source task without parameter refinement reduces overfitting and promotes better re-use of features extracted from the source task. We also consider three variants of SpotTune in the experiments. The first one is *SpotTune (running fine-tuned blocks)* in which during testing all the images are routed through the fine-tuned blocks. With this setting, the accuracy drops on all the datasets. This suggests that certain images in the target data can benefit from reusing some of the layers of the pre-trained network. The second variant is *SpotTune (global-k)* in which we set k to 3 in the experiments. Generally, SpotTune (global-3) performs worse than SpotTune, but is around 3 times more compact and, interestingly, is better than *Fine-tuning last-3*. This suggests that it is beneficial to have an image-specific fine-tuning strategy, and manually selecting the last k layers is not as effective as choosing the optimal non-contiguous set of k layers for fine-tuning. The third variant is *Random Policy* where we always fine-tune the last three layers and use a random policy network for other layers. The results show that an optimized policy outperforms a random policy.

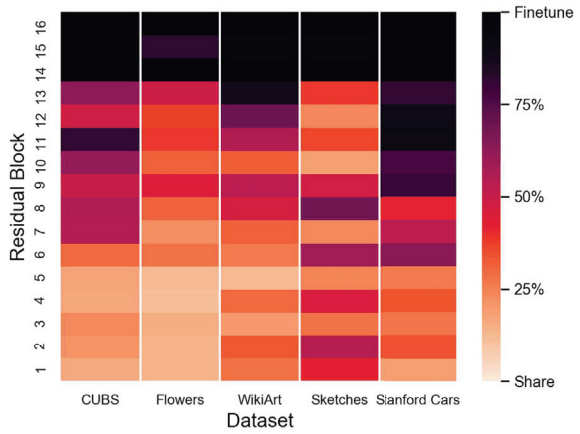

Figure 3: Visualization of policies on CUBS, Flowers, WikiArt, Sketches and Stanford Cars. Note that different datasets have very different policies. SpotTune automatically identifies the right fine-tuning policy for each dataset, for each training example.

4.2.2 Visualization of Policies

To better understand the fine-tuning policies learned by the policy network, we visualize them on CUBS, Flowers, WikiArt, Sketches, and Stanford Cars in Figure 3. The polices are learned on a ResNet-50 which has 16 blocks. The tone of red of a block indicates the number of images that were routed through the fine-tuned path of that block. For example, a block with a dark tone of red and a 75% level of fine-tuning (as shown in the scale depicted in the right of Figure 3) means 75% of the images in the test set use the fine-tuned block and the remaining 25% images share the pre-trained ImageNet block. The illustration shows that different datasets have very different fine-tuning policies. SpotTune allows us to automatically identify the right policy for each dataset, as well as for each training example, which would be infeasible through a manual approach.

4.2.3 Visualization of Block Usage

Besides the learned policies for each residual block, we are also interested in the number of fine-tuned blocks used by each dataset during testing. This can reveal the difference of the distribution of each target dataset and can also shed light on how the policy network works. In Figure 4, we show the distribution of the number of fine-tuned blocks used by each target dataset. During testing, for each dataset we categorize the test examples based on the number of fine-tuned blocks they use. For example, from Figure 4, we can see around 1000 images in the test set of the CUBS dataset use 7 fine-tuned blocks.

We have the following two observations based on the results. First, for a specific dataset, different images tend to use a different number of fine-tuned blocks. This again validates our hypothesis that it is more accurate to have an image-specific fine-tuning policy rather than a global fine-tuning policy for all images. Second, the distribution of fine-tuned blocks usage differs significantly across different target datasets. This demonstrates that based on the char-

	#par	ImNet	Airc.	C100	DPed	DTD	GTSR	Flwr	OGlt	SVHN	UCF	Score
Scratch	10x	59.87	57.10	75.73	91.20	37.77	96.55	56.30	88.74	96.63	43.27	1625
Scratch+ [37]	11x	59.67	59.59	76.08	92.45	39.63	96.90	56.66	88.74	96.78	44.17	1826
Feature Extractor	1x	59.67	23.31	63.11	80.33	55.53	68.18	73.69	58.79	43.54	26.80	544
Fine-tuning [38]	10x	60.32	61.87	82.12	92.82	55.53	99.42	81.41	89.12	96.55	51.20	3096
BN Adapt. [5]	1x	59.87	43.05	78.62	92.07	51.60	95.82	74.14	84.83	94.10	43.51	1353
LwF [26]	10x	59.87	61.15	82.23	92.34	58.83	97.57	83.05	88.08	96.10	50.04	2515
Series Res. adapt. [37]	2x	60.32	61.87	81.22	93.88	57.13	99.27	81.67	89.62	96.57	50.12	3159
Parallel Res. adapt. [38]	2x	60.32	64.21	81.92	94.73	58.83	99.38	84.68	89.21	96.54	50.94	3412
Res. adapt. (large) [37]	12x	67.00	67.69	84.69	94.28	59.41	97.43	84.86	89.92	96.59	52.39	3131
Res. adapt. decay [37]	2x	59.67	61.87	81.20	93.88	57.13	97.57	81.67	89.62	96.13	50.12	2621
Res. adapt. finetune all [37]	2x	59.23	63.73	81.31	93.30	57.02	97.47	83.43	89.82	96.17	50.28	2643
DAN [39]	2x	57.74	64.12	80.07	91.30	56.54	98.46	86.05	89.67	96.77	49.48	2851
PiggyBack [31]	1.28x	57.69	65.29	79.87	96.99	57.45	97.27	79.09	87.63	97.24	47.48	2838
SpotTune (Global-k)	4x	60.32	61.57	80.30	95.78	55.80	99.48	85.38	88.41	96.47	51.05	3401
SpotTune	11x	60.32	63.91	80.48	96.49	57.13	99.52	85.22	88.84	96.72	52.34	**3612**

Table 3: Results of SpotTune and baselines on the Visual Decathlon Challenge. The number of parameters is specified with respect to a ResNet-26 model as in [37].

acteristics of the target dataset, standard fine-tuning (which optimizes all the parameters of the pre-trained network towards the target task) may not be the ideal choice when conducting transfer learning with convolutional networks.

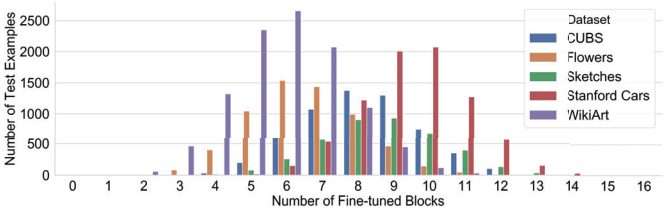

Figure 4: Distribution of the number of fine-tuned blocks used by the test examples. Different tasks and images require substantially different fine-tuning for best results, and this can be automatically inferred by SpotTune.

4.2.4 Visual Decathlon Challenge

We show the results of SpotTune and the baselines on the Visual Decathlon Challenge in Table 3. Among all the baselines, SpotTune achieves the highest Visual Decathlon score. Compared to standard fine-tuning, SpotTune has almost the same amount of parameters and improves the score by a large margin (3612 vs 3096). Considering the Visual Decathlon datasets, and the 5 datasets from our previous experiments, SpotTune shows superior performance on 12 out of 14 datasets over standard fine-tuning. Compared with other recently proposed methods on the Visual Decathlon Challenge [31, 39, 37, 38, 26], SpotTune sets the new state of the art for the challenge by only exploiting the transferability of the features extracted from ImageNet, without changing the network architecture. This is achieved without bells and whistles, i.e., we believe the results could be even further improved with more careful parameter tuning, and the use of other techniques such as data augmentation, including jittering images at test time and averaging their predictions. Compared to standard fine-tuning, our method uses 1.47x time in training (tested with 4 Titan Xp GPUs, batch size 96). At test time, the additional cost is negligible (0.013s vs 0.015s per image).

In SpotTune (Global-k), we fine-tune 3 blocks of the pre-trained model for each task which greatly reduces the number of parameters and still preserves a very competitive score. Although we focus on accuracy instead of parameter reduction in our work, we note that training our global-k variant with a multi-task loss on all 10 datasets, as well as model compression techniques, could further reduce the number of parameters in our method. We leave this research thread for future work.

5. Conclusion

We proposed an adaptive fine-tuning algorithm called SpotTune which specializes the fine-tuning strategy for each training example of the target dataset. We showed that our method outperforms the key most popular and widely used protocols for fine-tuning on a variety of public benchmarks. We also evaluated SpotTune on the Visual Decathlon challenge, achieving the new state of the art, as measured by the overall score across the 10 datasets.

Acknowledgements. We would like to thank Prof. Song Han for helpful discussions. This work is supported in part by IARPA via DOI/IBC contract number D17PC00341, by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, NSF CHASE-CI #1730158, DARPA Lifelong Learning Machines, IBM OCR, and an IBM Faculty Award. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DOI/IBC, or the U.S. Government.

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*, 38(9):1790–1802, 2016. 1, 5

[2] E. Bengio, P.-L. Bacon, J. Pineau, and D. Precup. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015. 3

[3] Y. Bengio. Deep learning of representations for unsupervised and transfer learning. In *ICML Workshop on Unsupervised and Transfer Learning*, 2012. 1

[4] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 3

[5] H. Bilen and A. Vedaldi. Universal representations: The missing link between faces, text, planktons, and cat breeds. *arXiv preprint arXiv:1701.07275*, 2017. 6, 8

[6] Q. Chen, J. Huang, R. Feris, L. M. Brown, J. Dong, and S. Yan. Deep domain adaptation for describing people based on fine-grained clothing attributes. In *CVPR*, 2015. 2

[7] S. Chopra, S. Balakrishnan, and R. Gopalan. Dlid: Deep learning for domain adaptation by interpolating between domains. In *ICML workshop on challenges in representation learning*, 2013. 2

[8] G. Csurka. Domain adaptation for visual applications: A comprehensive survey. In *Domain Adaptation in Computer Vision Applications*. Springer, 2017. 2

[9] 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 *ICML*, 2014. 5

[10] M. Eitz, J. Hays, and M. Alexa. How do humans sketch objects? *ACM Transactions on Graphics*, 31(4):44–1, 2012. 5

[11] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. P. Vetrov, and R. Salakhutdinov. Spatially adaptive computation time for residual networks. In *CVPR*, 2017. 3

[12] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016. 2

[13] W. Ge and Y. Yu. Borrowing treasures from the wealthy: Deep transfer learning through selective joint fine-tuning. In *CVPR*, 2017. 2

[14] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 1, 2, 5

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 2, 3

[16] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 1, 2

[17] L. Jacob, J.-p. Vert, and F. R. Bach. Clustered multi-task learning: A convex formulation. In *NeurIPS*, 2009. 2

[18] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016. 2, 3, 4

[19] Z. Kang, K. Grauman, and F. Sha. Learning with whom to share in multi-task feature learning. In *ICML*, 2011. 2

[20] S. Kornblith, J. Shlens, and Q. V. Le. Do better imagenet models transfer better? *arXiv preprint arXiv:1805.08974*, 2018. 1, 2

[21] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3d object representations for fine-grained categorization. In *ICCV Workshop on Workshop on 3D Representation and Recognition*, pages 554–561, 2013. 5

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012. 1

[23] A. Kumar and H. Daumé III. Learning task grouping and overlap in multi-task learning. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, pages 1723–1730, 2012. 2

[24] A. Kumar, P. Sattigeri, K. Wadhawan, L. Karlinsky, R. S. Feris, W. T. Freeman, and G. Wornell. Co-regularized alignment for unsupervised domain adaptation. In *NeurIPS*, 2018. 2

[25] X. Li, Y. Grandvalet, and F. Davoine. Explicit inductive bias for transfer learning with convolutional networks. In *ICML*, 2018. 2, 6

[26] Z. Li and D. Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. 6, 8

[27] L. Liu and J. Deng. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. In *AAAI*, 2018. 3

[28] M. Long, Y. Cao, J. Wang, and M. I. Jordan. Learning transferable features with deep adaptation networks. In *ICML*, 2015. 2, 5

[29] Y. Lu, A. Kumar, S. Zhai, Y. Cheng, T. Javidi, and R. S. Feris. Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. In *CVPR*, 2017. 2

[30] C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016. 2, 3, 4

[31] A. Mallya and S. Lazebnik. Piggyback: Adding multiple tasks to a single, fixed network by learning to mask. *arXiv preprint arXiv:1801.06519*, 2018. 6, 8

[32] E. Meyerson and R. Miikkulainen. Beyond shared hierarchies: Deep multitask learning through soft layer ordering. In *ICLR*, 2018. 2

[33] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert. Cross-stitch networks for multi-task learning. In *CVPR*, 2016. 2

[34] M.-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics & Image Processing*, 2008. 5

[35] S. J. Pan, Q. Yang, et al. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010. 1, 2

[36] A. Passos, P. Rai, J. Wainer, and H. Daume III. Flexible modeling of latent task structures in multitask learning. *arXiv preprint arXiv:1206.6486*, 2012. 2

[37] S.-A. Rebuffi, H. Bilen, and A. Vedaldi. Learning multiple visual domains with residual adapters. In *NeurIPS*, 2017. 2, 5, 6, 8

[38] S.-A. Rebuffi, H. Bilen, and A. Vedaldi. Efficient parametrization of multi-domain deep neural networks. In *CVPR*, 2018. 6, 8

[39] A. Rosenfeld and J. K. Tsotsos. Incremental learning through deep adaptation. *arXiv preprint arXiv:1705.04228*, 2017. 6, 8

[40] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. 2, 6

[41] B. Saleh and A. Elgammal. Large-scale classification of fine-art paintings: Learning the right metric on the right feature. *arXiv preprint arXiv:1505.00855*, 2015. 5

[42] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *CVPR DeepVision Workshop*, 2014. 2, 5

[43] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *ICLR*, 2017. 3

[44] N. Tajbakhsh, J. Y. Shin, S. R. Gurudu, R. T. Hurst, C. B. Kendall, M. B. Gotway, and J. Liang. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE transactions on medical imaging*, 35(5):1299–1312, 2016. 1, 5

[45] S. Thrun and J. OSullivan. Clustering learning tasks and the selective cross-task transfer of knowledge. In *Learning to learn*. Springer, 1998. 2

[46] A. Veit and S. Belongie. Convolutional networks with adaptive inference graphs. In *ECCV*, 2018. 3

[47] A. Veit, M. J. Wilber, and S. Belongie. Residual networks behave like ensembles of relatively shallow networks. In *NeurIPS*, 2016. 2, 3

[48] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. 2011. 5

[49] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, and R. Feris. Blockdrop: Dynamic inference paths in residual networks. In *CVPR*, 2018. 3, 4

[50] Z. Yang, B. Dhingra, K. He, W. W. Cohen, R. Salakhutdinov, Y. LeCun, et al. Glomo: Unsupervisedly learned relational graphs as transferable representations. *arXiv preprint arXiv:1806.05662*, 2018. 2

[51] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *NeurIPS*, 2014. 1, 2, 5

[52] J. Zhou, J. Chen, and J. Ye. Clustered multi-task learning via alternating structure optimization. In *NeurIPS*, 2011. 2