Parameter-efficient Transfer and Multitask Learning

Pramod Kaushik Mudrarkarta[*]
The University of Chicago
`pramodkm@uchicago.edu`

Mark Sandler, Andrey Zhmoginov, Andrew Howard
Google Inc.
`{sandler,azhmogin,howarda}@google.com`

Abstract

We introduce a novel method that enables parameter-efficient transfer and multitask learning. The basic approach is to allow a model patch - a small set of parameters - to specialize to each task instead of fine-tuning the last layer or the entire network. We show that learning a set of scales and biases converts an SSD detection model into a 1000-class classification model while reusing 98% of parameters. Similarly, we show that re-learning existing low-parameter layers (such as depth-wise convolutions) also improves accuracy significantly. Our method allows both simultaneous (multitask) as well as sequential (transfer) learning wherein we adapt pretrained networks to solve new problems. For multitask learning, despite using much fewer parameters than logits-only fine-tuning, we match single-task-based performance.

1 Introduction

Deep neural networks have revolutionized machine intelligence, and are ubiquitous in computer vision [15, 26, 18]. Nowadays, computation is being increasingly shifted to consumer devices; delivering faster response, and better security and privacy guarantees [11, 9]. The growing space of deep learning applications demands an ability to quickly build and customize models on device. While model sizes have dropped dramatically from >50M parameters [15, 26] to <5M [25, 9, 30, 19], accuracies have been improving. However, delivering and updating hundreds of models on the device is still a significant expense in terms of bandwidth, energy and storage costs.

Although designing even smaller models may be possible, we explore a different angle: re-purposing models to new tasks by training only a few parameters while incurring minimal loss in accuracy w.r.t. a model trained from scratch. While there is ample existing work on compressing models and learning as few weights as possible [24, 25, 9] to solve a single task, we do not know of any prior work on minimizing the combined model size when solving many tasks together.

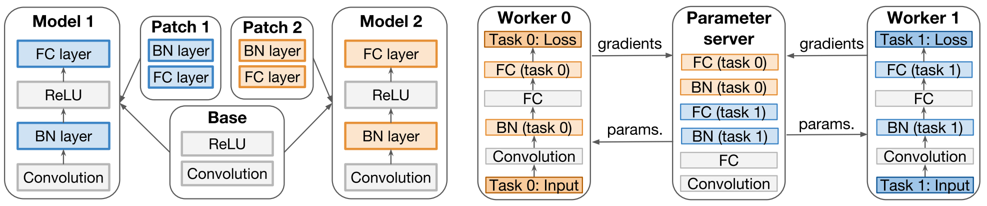

Figure 1: Left: An example illustrating the idea of a model patch. Right: An example of multi-task learning. FN: fully connected layer, BN: batch normalization layer, and Convolution: convolution layer.

Our contribution is a learning paradigm based on the idea of a **model patch**: a small set of per-channel transformations dispersed throughout the network amounting to only a tiny increase in model size.

[*]Work done while at Google.

For an example, see Figure 1 (left). Each task carries its own patch, which, along with a shared set of parameters constitutes the model for that task. We consider two scenarios:

Transfer learning: Here, we wish to adapt a pretrained model to new tasks. A different output space in the new task necessitates re-learning the last layer. We apply a model patch and train the patched parameters, optionally also the last layer. The rest of the parameters are left unchanged. When the last layer is not trained, it is fixed to its random initial value.

Multitask learning: We aim to simultaneously, but independently, train multiple networks that share most weights. Although all weights are updated, each task trains a patched model. By training all parameters, this setting offers more adaptability to tasks while not compromising on model size. In implementation, we use the distributed TensorFlow paradigm: a central parameter server receives gradient updates from workers, updates the weights and sends updated weights back to workers. We allow subsets of workers to train different tasks; workers thus may have different computational graphs, and task-specific input pipelines and loss functions. An illustration is in Figure 1.

Next, we describe the types of patches that we use.

Scale-and-bias patch: This patch applies per-channel scale and bias to every layer. In practice, this can be absorbed into normalization layers [10]. A batch-normalized activation tensor, $BN(\mathbf{X}) = \gamma\frac{\mathbf{X}-\mu(\mathbf{X})}{\sigma(\mathbf{X})} + \beta$ where $\mu(\mathbf{X}), \sigma(\mathbf{X})$ are mean and standard deviation computed per minibatch, and γ, β are learned. The patch corresponds to all the $\gamma, \beta, \mu, \sigma$, which is a small number of parameters. For instance, in both MobilenetV2 [25](3.5M params) and InceptionV3 [28](25M params) on ImageNet [5], it amounts to less than 40K parameters. While we utilize batch normalization in this paper, we note that using explicit biases and scales would have similar results. In Appendix A, we formally analyze the expressivity of training only the bias terms in a neural network.

Depthwise-convolution patch: This patch re-learns spatial convolution filters. Depth-wise separable convolutions were introduced in deep neural networks as way to reduce number of parameters without losing much accuracy [9, 3]. They are essentially standard convolutions decomposed into a depthwise convolution layer that applies one convolutional filter per input channel, and a pointwise layer that linearly combines the depthwise convolutional layers' output across channels. We find that the set of depthwise convolution layers can be used as a model patch. They are also lightweight - for instance, they account for less than 3% of MobilenetV2's parameters for ImageNet.

2 Related work

A widely used technique for transfer learning among practitioners is based on re-training only the last (few) layer(s) [29, 6]. It has been repeatedly shown that this approach often works best for similar tasks (for example, see [6]). Another approach is full fine-tuning [4] where a pretrained model is used as a warm start for the training process. While this often leads to significantly improved accuracy over last-layer fine-tuning, it requires creating full model for each new task, and may lead to overfitting under limited data. In this work, we are primarily interested in producing highly accurate models while reusing most weights of the original model which also addreseses the overfitting issue.

We see significant boost in performance when we fine-tune model patches along with last layer (Section 3). This result is in contrast with [8], where they show that the last layer does not matter when training full networks. Mapping out the conditions of when a linear classifier can be replaced with a random embedding is an important open question.

[16] show that re-computing batch normalization statistics for different domains helps improve accuracy. In [24] it is suggested that learning batch normalization layers in an otherwise randomly initialized network is sufficient to build non-trivial models. Re-computing batch normalization statistics is also frequently used for model quantization to prevent the model activation space from drifting [13]. In the present work, we significantly broaden and unify the scope of the idea and scale up the approach by performing transfer and multi-task learning across completely different tasks, providing a powerful tool for many practical applications.

Our work has interesting connections to meta-learning [22, 7, 2]. For instance, when training data is large, one can allow each task to carry a small model patch in the Reptile algorithm of [22] in order to increase expressivity at low cost.

3 Experiments

We evaluate the performance of our method using the image recognition networks MobilenetV2 [25], InceptionV3 [28] and Single-Shot Multibox Detector (SSD) [18], and a variety of datasets (details in Appendix C). We use both scale-and-bias (S/B) and depthwise-convolution (DW) patches. Both MobilenetV2 and InceptionV3 have batch normalization - we use those parameters as the S/B patch. MobilenetV2 has depthwise-convolutions from which we construct the DW patch. We compare with two scenarios: (1) only fine-tuning the last layer, and (2) fine-tuning the entire network. Details of training process and hyperparameters are described in Appendix D.

3.1 Transfer learning

We take MobileNetV2 and InceptionV3 pretrained on ImageNet (Top1 accuracies 71.8% and 76.6% respective), and fine-tune various model patches for other datasets. Results on InceptionV3 are shown in Table 1. We see that fine-tuning only the scale-and-bias patch (using a fixed, random last layer) results in accuracies oftentimes comparable to fine-tuning only the last layer, while using fewer parameters. Compared to full fine-tuning [4], we use orders of magnitude fewer parameters while achieving non-trivial performance.

Table 1: Transfer-learning on Inception V3, against full-network fine-tuning.

Fine-tuned params.	Flowers		Cars		Aircraft	
	Acc.	#params	Acc.	#params	Acc.	#params
Last layer	84.5	208K	55	402K	45.9	205K
S/B + last layer	90.4	244K	81	437K	70.7	241K
S/B only (random last)	79.5	**36K**	33	**36K**	52.3	**36K**
All (ours)	93.3	25M	92.3	25M	87.3	25M
All [4]	96.3	25M	91.3	25M	82.6	25M

Next, we take an object detection (SSD) model [18] pretrained on COCO images [17] and repurpose it for image classification on ImageNet. The SSD model uses MobilenetV2 (minus the last layer) as an input featurizer. We extract it, append a linear layer and then fine-tune. Results are in Table 2. Again, we see the effectiveness of training the model patch along with the last layer - a 2% increase in the parameters translates to 19.4% increase in accuracy.

Table 2: Learning Imagenet from SSD feature extractor (left) and random filters (right)

Fine-tuned params.	#params	COCO→Imagenet, Top1	Random→ Imagenet, Top1
Last layer	1.31M	29.2%	0%
S/B + last layer	1.35M	47.8%	20%
S/B only	34K	6.4%	2.3%
All params	3.5M	71.8%	71.8%

Typically, a small learning rate is used when fine-tuning the entire network, as large rates lead to overfitting. We observed the opposite behavior when fine-tuning small model patches: accuracy grows with learning rate. In practice, fine-tuning a patch that includes the last layer is more stable w.r.t. learning rate than full fine-tuning or fine-tuning only the S/B patch. An overview of results on MobilenetV2 with different learning rates and model patches (S/B, DW) is shown in Figure 2. The effectiveness of small model patches over fine-tuning only the last layer is again clear. Combining model patches and fine-tuning results in a synergistic effect. In Appendix B, we show additional experiments comparing the importance of learning custom bias/scale with simply updating batch-norm statistics (as suggested by [16]).

3.2 Multitask learning

We simultaneously train MobilenetV2 [25] on two large datasets: ImageNet and Places365. The two corresponding models share all parameters except those in the scale-and-bias patch and the last layer. Details of implementation are in Appendix D. Results are in Table 3.

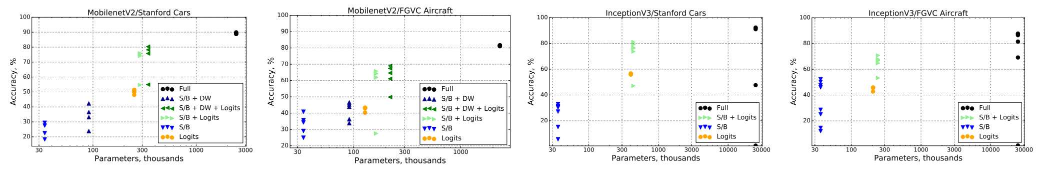

Figure 2: Performance of different fine-tuning approaches for different datasets for Mobilenet V2 and Inception. The same color points correspond to runs with different initial learning rates, starting from 0.0045 to 0.45 with factor 3. Note x-axis is exponential. Logit-only fine tuning (red circles) both require more parameters and are often worse than scale-and-bias patches. Best viewed in color.

Table 3: Multi-task learning with MobilenetV2 on ImageNet and Places-365.

Task	S/B patch + last layer	Last layer	Independently trained
Imagenet	70.2%	64.4%	71.8%
Places365	**54.3**%	51.4%	54.2%
# total parameters	3.97M	3.93M	6.05M

We see that multi-task training accuracy using a small model patch is comparable to single-task accuracy. Also, the benefit of using a model patch over not using it is evident. The transfer learning accuracy on Places-365 using an ImageNet-pretrained MobilenetV2 model is less than 47.6% even with full fine-tuning. This suggests that one may prefer multi-task learning over transfer learning when large datasets are involved.

Domain adaptation: In this experiment, each task is the classification of ImageNet images at a different resolution. Such a model collection allows operating a system at different speeds where models are chosen at inference time based on power and latency requirements. We only have the scale-and-bias patch private to each task; the last layer weights are shared. Results are in Table 4.

Table 4: Multi-task accuracies of 5 MobilenetV2 models acting at different resolutions on ImageNet.

Image resolution	S/B patch	All shared	Independently trained
96 x 96	60.3%	52.6%	60.3%
128 x 128	**66.3**%	62.4%	65.3%
160 x 160	**69.5**%	67.2%	68.8%
192 x 192	71%	69.4%	70.7%
224 x 224	71.8%	70.6%	71.8%
# total parameters	**3.7M**	3.5M	17.7M

One application of domain adaptation with model patches is cost-efficient model cascades. Following [27], we use multitask learning to train multiple MobilenetV2 models on images of different resolutions, and then build a cascaded model with the same accuracy as the best model but at 15.2% less average cost. Note that this requires storing only 5% more parameters than a single model.

4 Conclusions, Open questions and Future work

We introduced a new way of performing transfer and multi-task learning where we patch a small fraction of model parameters, that leads to high accuracy on very different tasks compared to traditional methods. This enables practitioners to build a large number of models with small incremental cost per model. While we see that model patches can adapt to a fixed, random last layer (also noted in [8]), we see a significant accuracy boost when we allow the last layer also to be trained. It is important to close this gap in our understanding of when the linear classifier is important for the final performance. From practical perspective, cross-domain multi-task learning (such as segmentation and classification) is a promising direction to pursue. Finally our approach provides for an interesting extension to the federated learning approach proposed in [11], where individual devices ship their gradient updates to the central server. In this extension we envision user devices keeping their local private patch to maintain personalized model while sending common updates to the server.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Fei Chen, Zhenhua Dong, Zhenguo Li, and Xiuqiang He. Federated meta-learning for recommendation. arXiv preprint arXiv:1802.07876, 2018.

[3] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017.

[4] Y. Cui, Y. Song, C. Sun, A. Howard, and S. Belongie. Large Scale Fine-Grained Categorization and Domain-Specific Transfer Learning. ArXiv e-prints, June 2018.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In CVPR09, 2009.

[6] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In Eric P. Xing and Tony Jebara, editors, Proceedings of the 31st International Conference on Machine Learning, volume 32 of Proceedings of Machine Learning Research, pages 647–655, Bejing, China, 22–24 Jun 2014. PMLR.

[7] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. arXiv preprint arXiv:1703.03400, 2017.

[8] Elad Hoffer, Itay Hubara, and Daniel Soudry. Fix your classifier: the marginal value of training the last weight layer. CoRR, abs/1801.04540, 2018.

[9] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. CoRR, abs/1704.04861, 2017.

[10] Sergey Ioffe and Christian Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. In Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, pages 448–456. JMLR.org, July 2015.

[11] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. In NIPS Workshop on Private Multi-Party Machine Learning, 2016.

[12] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In Proceedings of the IEEE International Conference on Computer Vision Workshops, pages 554–561, 2013.

[13] R. Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. ArXiv e-prints, June 2018.

[14] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States., pages 1106–1114, 2012.

[16] Yanghao Li, Naiyan Wang, Jianping Shi, Jiaying Liu, and Xiaodi Hou. Revisiting batch normalization for practical domain adaptation. CoRR, abs/1603.04779, 2016.

[17] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In European conference on computer vision, pages 740–755. Springer, 2014.

[18] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In European conference on computer vision, pages 21–37. Springer, 2016.

[19] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. arXiv preprint arXiv:1807.11164, 2018.

[20] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. arXiv preprint arXiv:1306.5151, 2013.

[21] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 27, pages 2924–2932. Curran Associates, Inc., 2014.

[22] Alex Nichol and John Schulman. Reptile: a scalable metalearning algorithm. arXiv preprint arXiv:1803.02999, 2018.

[23] M-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing, Dec 2008.

[24] Amir Rosenfeld and John K. Tsotsos. Intriguing properties of randomly weighted networks: Generalizing while learning next to nothing. CoRR, abs/1802.00844, 2018.

[25] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. arXiv preprint arXiv:1801.04381, 2018.

[26] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556, 2014.

[27] Matthew Streeter. Approximation algorithms for cascading prediction models. In ICML, 2018.

[28] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 2818–2826, 2016.

[29] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, NIPS'14, pages 3320–3328, Cambridge, MA, USA, 2014. MIT Press.

[30] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. CoRR, abs/1707.01083, 2017.

[31] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. IEEE transactions on pattern analysis and machine intelligence, 2017.

A Analysis

Experiments (Section 3) show that model-patch based fine-tuning, especially with the scale-and-bias patch, is comparable and sometimes better than last-layer-based fine-tuning, despite utilizing a significantly smaller set of parameters. At a high level, our intuition is based on the observation that individual channels of hidden layers of neural network form an embedding space, rather than correspond to high-level features. Therefore, even simple transformations to the space could result in significant changes in the target classification of the network.

In this section, we attempt to gain some insight into this phenomenon by taking a closer look at the properties of the last layer and studying low-dimensional models.

A deep neural network performing classification can be understood as two parts:

1. a network base corresponding to a function $F : \mathbb{R}^d \to \mathbb{R}^n$ mapping d-dimensional input space $\mathbb{X}$ into an n-dimensional embedding space $\mathbb{G}$, and
2. a linear transformation $s : \mathbb{R}^n \to \mathbb{R}^k$ mapping embeddings to logits with each output component corresponding to an individual class.

An input $\boldsymbol{x} \in \mathbb{X}$ producing the output $\boldsymbol{o} := s(F(\boldsymbol{x})) \in \mathbb{R}^k$ is assigned class c iff $\forall i \neq c, o_i < o_c$.

We compare fine-tuning model patches with fine-tuning only the final layer s. Fine-tuning only the last layer has a severe limitation caused by the fact that linear transformations preserve convexity.

It is easy to see that, regardless of the details of s, the mapping from embeddings to logits is such that if both $\boldsymbol{\xi}^a, \boldsymbol{\xi}^b \in \mathbb{G}$ are assigned label c, the same label is assigned to every $\boldsymbol{\xi}^\tau := \tau\boldsymbol{\xi}^b + (1-\tau)\boldsymbol{\xi}^a$ for $0 \leq \tau \leq 1$. Indeed, $[s(\boldsymbol{\xi}^\tau)]_c = \tau o_c^b + (1-\tau)o_c^a > \tau o_i^b + (1-\tau)o_i^a = [s(\boldsymbol{\xi}^\tau)]_i$ for any $i \neq c$ and $0 \leq \tau \leq 1$, where $\boldsymbol{o}^a := s(\boldsymbol{\xi}^a)$ and $\boldsymbol{o}^b := s(\boldsymbol{\xi}^b)$. Thus, if the model assigns inputs $\{\boldsymbol{x}_i | i = 1, \dots, n_c\}$ some class c, then the same class will also be assigned to any point in the preimage of the convex hull of $\{F(\boldsymbol{x}_i) | i = 1, \dots, n_c\}$.

This property of the linear transformation s limits one's capability to tune the model given a new input space manifold. For instance, if the input space is “folded” by F and the neighborhoods of very different areas of the input space $\mathbb{X}$ are mapped to roughly the same neighborhood of the embedding space, the final layer cannot disentangle them while operating on the embedding space alone (should some new task require differentiating between such “folded” regions).

We illustrate the difference in expressivity between model-patch-based fine-tuning and last-layer-based fine-tuning in the cases of 1D (below) and 2D (Section A.2) inputs and outputs. Despite the simplicity, our analysis provides useful insights into how by simply adjusting biases and scales of a neural network, one can change which regions of the input space are folded and ultimately the learned classification function.

In what follows, we will work with a construct introduced by [21] that demonstrates how neural networks can “fold” the input space $\mathbb{X}$ a number of times that grows exponentially with the neural network depth[2]. We consider a simple neural network with one-dimensional inputs and outputs and demonstrate that a single bias can be sufficient to alter the number of “folds”, the topology of the $\mathbb{X} \to \mathbb{G}$ mapping. More specifically, we illustrate how the number of connected components in the preimage of a one-dimensional segment $[\boldsymbol{\xi}^a, \boldsymbol{\xi}^b]$ can vary depending on a value of a single bias variable.

A.1 1D case

As in [21], consider the following function:

$$q(x; b) \equiv 2\,\mathrm{ReLU}\left([1, -1, \dots, (-1)^{p-1}] \cdot \boldsymbol{v}^T(x; \boldsymbol{b}) + b_p\right),$$

where

$$\boldsymbol{v}(x; \boldsymbol{b}) \equiv [\max(0, x + b_0), \max(0, 2x - 1 + b_1), \dots, \max(0, 2x - (p-1) + b_{p-1})],$$

[2]In other words, $F^{-1}(\boldsymbol{\xi})$ for some $\boldsymbol{\xi}$ contains an exponentially large number of disconnected components.

p is an even number, and $\boldsymbol{b} = (b_0, \dots, b_p)$ is a $(p+1)$–dimensional vector of tunable parameters characterizing q. Function $q(x; \boldsymbol{b})$ can be represented as a two-layer neural network with ReLU activations.

Set $p = 2$. Then, this network has 2 hidden units and a single output value, and is capable of "folding" the input space twice. Defining F to be a composition of k such functions

$$F(x; \boldsymbol{b}^{(1)}, \dots, \boldsymbol{b}^{(k)}) \equiv q(q(\dots q(q(x; \boldsymbol{b}^{(1)}); \boldsymbol{b}^{(2)}); \dots; \boldsymbol{b}^{(k-1)}); \boldsymbol{b}^{(k)}), \tag{1}$$

we construct a neural network with $2k$ layers that can fold input domain $\mathbb{R}$ up to 2^k times. By plotting $F(x)$ for $k = 2$ and different values of $b_0^{(1)}$ while fixing all other biases to be zero, it is easy to observe that the preimage of a segment $[0.2, 0.4]$ transitions through several stages (see figure 3), in which it: (a) first contains 4 disconnected components for $b_0^{(1)} > -0.05$, (b) then 3 for $b_0^{(1)} \in (-0.1, -0.05]$, (c) 2 for $b_0^{(1)} \in (-0.4, -0.1]$, (d) becomes a simply connected segment for $b_0^{(1)} \in [-0.45, -0.4]$ and (e) finally becomes empty when $b_0^{(1)} < -0.45$. This result can also be extended to $k > 2$, where, by tuning $b_0^{(1)}$, the number of "folds" can vary from 2^k to 0.

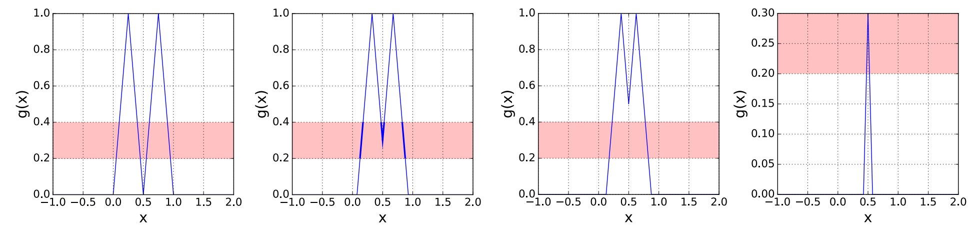

Figure 3: Function plots $F(x; \boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)})$ for a 4-layer network given by equation 1 with $k = 2$ and all biases except $b_0^{(1)}$ set to zero. From left to right: $b_0^{(1)} = 0$, $b_0^{(1)} = -0.075$, $b_0^{(1)} = -0.125$ and $b_0^{(1)} = -0.425$. The preimage of a segment $[0.2, 0.4]$ (shown as shaded region) contains 4, 3, 2 and 1 connected components respectively.

A.2 2D Case

Here we show an example of a simple network that "folds" input space in the process of training and associates identical embeddings to different points of the input space. As a result, fine-tuning the final linear layer is shown to be insufficient to perform transfer learning to a new dataset. We also show that the same network can learn alternative embedding that avoids input space folding and permits transfer learning.

Consider a deep neural network mapping a 2D input into 2D logits via a set of 5 ReLU hidden layers: $\underline{\text{2D input}} \to \underline{\text{8D state}} \to \underline{\text{16D state}} \to \underline{\text{16D state}} \to \underline{\text{8D state}} \to \underline{m\text{-D embedding}}$ (no ReLU) $\to \underline{\text{2D logits}}$ (no ReLU). Since the embedding dimension is typically smaller than the input space dimension, but larger than the number of categories, we first choose the embedding dimension m to be 2. This network is trained (applying sigmoid to the logits and using cross entropy loss function) to map (x, y) pairs to two classes according to the groundtruth dependence depicted in figure 4(a). Learned function is shown in figure 4(c). The model is then fine-tuned to approximate categories shown in figure 4(b). Fine-tuning all variables, the model can perfectly fit this new data as shown in figure 4(d).

Once the set of trainable parameters is restricted, model fine-tuning becomes less efficient. Figures 4(A) through 4(E) show output values obtained after fine-tuning different sets of parameters. In particular, it appears that training the last layer alone (see figure 4(E; top)) is insufficient to adjust to new training data, while training biases and scales allows to approximate new class assignment (see figure 4(C; top)). Notice that a combination of all three types of trainable parameters (biases, scales and logits) frequently results in the best function approximation even if the initial state is chosen to be random (see figure 4(A)-(E); bottom row).

Interestingly, poor performance of logit fine-tuning seen in figure 4(E) extends to higher embedding dimensions as well. Plots similar to those in figure 4, but generated for the model with the embedding dimension m of 4 are shown in figure 5. In this case, we can see that the final layer fine-tuning is

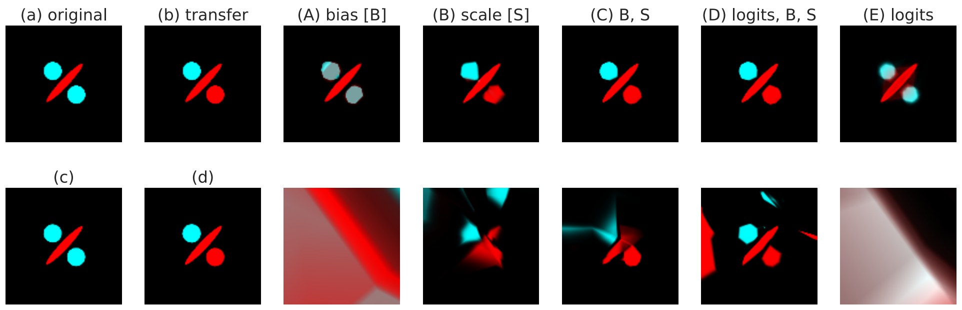

Figure 4: The neural network is first trained to approximate class assignment shown in (a) (with the corresponding learned outputs in (c)), network parameters are then fine-tuned to match new classes shown in (b). If all network parameters are trained, it is possible (d) to get a good approximation of the new class assignment. Outputs obtained by fine-tuning only a subset of parameters are shown in columns (A) through (E): functions fine-tuned from a pretrained state (c) are shown at the top row; functions trained from a random state (the same for all figures) are shown at the bottom. Each figure shows training with respect to a different parameter set: (A) biases; (B) scales; (C) biases and scales; (D) logits, biases and scales; (E) just logits.

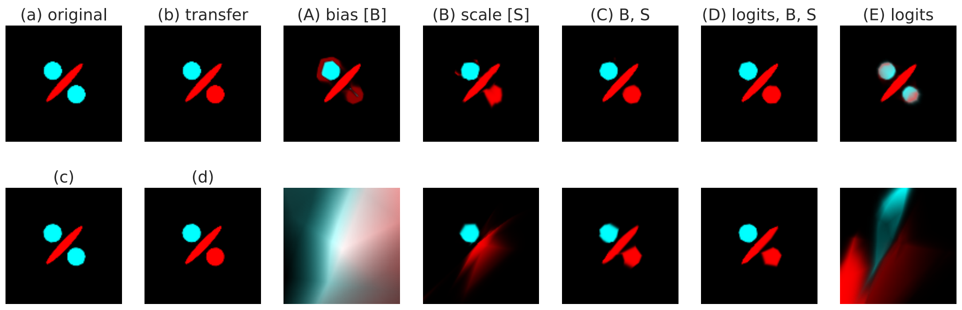

Figure 5: Plots similar to those shown in figure 4, but obtained for the embedding dimension of 4.

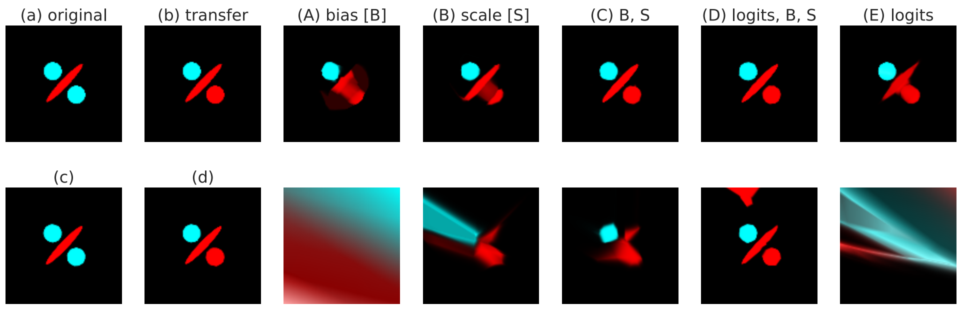

Figure 6: Plots similar to those shown in figure 4, but obtained for the embedding dimension of 8.

again insufficient to achieve successful transfer learning. As the embedding dimension goes higher, last layer fine-tuning eventually reaches acceptable results (see figure 6 showing results for $m = 8$).

The explanation behind poor logit fine-tuning results can be seen by plotting the embedding space of the original model with $m = 2$ (see figure 7(a)). Both circular regions are assigned the same embedding and the final layer is incapable of disentangling them. But it turns out that the same network could have learned a different embedding that would make last layer fine-tuning much more efficient. We show this by training the network on the classes shown in figure 7(b). This class assignment breaks the symmetry and the new learned embedding shown in figure 7(c) can now be used to adjust to new class assignments shown in figure 7(d), (e) and (f) by fine-tuning the final layer alone.

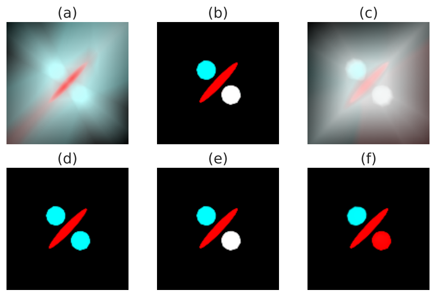

Figure 7: Original model trained to match class assignment shown in figure 4(a) results in the embedding shown in (a) that “folds” both circular regions together. After training the same model on different classes (b), the new embedding (c) allows one to fine-tune the last layer alone to obtain outputs shown in (d), (e) and (f).

B Additional experiments

B.1 Adjusting batch-normalization statistics

The results of [16] suggested that adjusting Batch Normalization statistics helps with domain adaption. Interestingly we found that it significantly worsens results for transfer learning, unless bias and scales are allowed to learn. We find that fine-tuning on last layer with batch-norm statistics readjusted to keep activation space at mean 0/variance 1, makes the network to significantly under-perform compared to fine-tuning with frozen statistics. Even though adding *learned* bias/scales significantly outperforms logit-only based fine-tuning. We summarize our experiments in Table 5

Table 5: The effect of batch-norm statistics on logit-based fine-tuning for MobileNetV2

Method	Flowers	Aircraft	Stanford Cars	Cifar100
Last layer (logits)	80.2	43.3	51.4	45.0
Same as above + batchnorm statistics	79.8	38.3	43.6	57.2
Same as above + scales and biases	**86.9**	**65.6**	**75.9**	**74.9**

B.2 Accuracy vs. learning rate

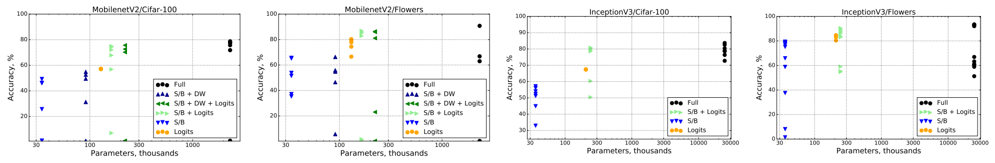

Figure 8: Performance of different fine-tuning approaches for Mobilenet V2 and Inception V3 for Cifar100 and Flowers. Best viewed in color.

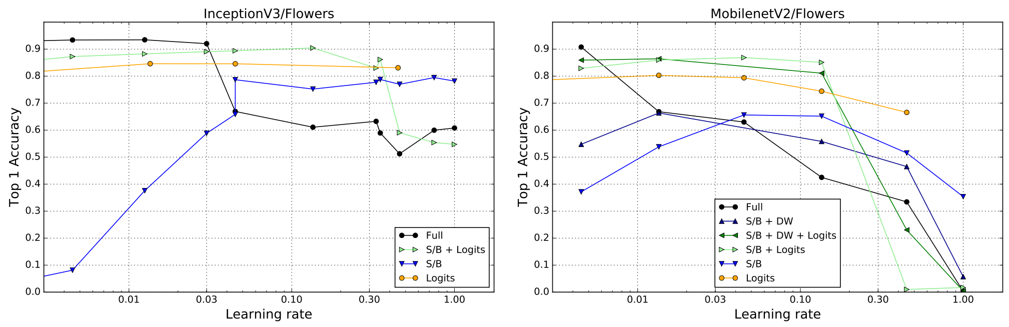

Figure 9: Final accuracy as a function of learning rate. Note how full fine-tuning requires learning rate to be small, while bias/scale tuning requires learning rate to be large enough.

C Details of datasets used

We use a variety of datasets: ImageNet [5], CIFAR-10/100 [14], Cars [12], Aircraft [20], Flowers-102 [23] and Places-365 [31]. Details of the datasets are shown in Table 6.

Table 6: Datasets used in experiments (Section 3)

Name	CIFAR-100	Flowers-102	Cars	Aircraft	Places-365	ImageNet
#images	60,000	8,189	16,185	10,200	1.8 million	1.3 million
#classes	100	102	196	102	365	1000

D Details of implementation

We use TensorFlow [1], and NVIDIA P100 and V100 GPUs for our experiments. Unless otherwise specified we use 224×224 images for MobilenetV2 and 299×299 for InceptionV3. As a special-case, for Places-365 dataset, we use 256×256 images. We use RMSProp optimizer with a learning rate of 0.045 and decay factor 0.98 per 2.5 epochs.

In multi-task learning, in order to inhibit one task from dominating the learning of the weights, we ensure that the learning rates for different tasks are comparable at any given point in time. This is achieved by setting hyperparameters such that the ratio of dataset size and the number of epochs per learning rate decay step is the same for all tasks. We assign the same number of workers for each task in the distributed learning environment. The results are shown in Table 3.