Learning without Forgetting

Zhizhong Li, Derek Hoiem

University of Illinois Urbana Champaign

Abstract. When building a unified vision system or gradually adding new capabilities to a system, the usual assumption is that training data for all tasks is always available. However, as the number of tasks grows, storing and retraining on such data becomes infeasible. A new problem arises where we add new capabilities to a Convolutional Neural Network (CNN), but the training data for its existing capabilities are unavailable. We propose our Learning without Forgetting method, which uses only new task data to train the network while preserving the original capabilities. Our method performs favorably compared to commonly used feature extraction and fine-tuning adaption techniques and performs similarly to multitask learning that uses original task data we assume unavailable. A more surprising observation is that Learning without Forgetting may be able to replace fine-tuning as standard practice for improved new task performance.

Keywords: Convolutional Neural Networks, Transfer Learning, Multitask Learning, Deep Learning, Visual Recognition

1 Introduction

Many practical vision applications require learning new visual capabilities while maintaining performance on existing ones. For example, a robot may be delivered to someone's house with a set of default object recognition capabilities, but new site-specific object models need to be added. Or for construction safety, a system can identify whether a worker is wearing a safety vest or hard hat, but a superintendent may wish to add the ability to detect improper footware. Ideally, the new tasks could be learned while sharing parameters from old ones, without degrading performance on old tasks or having access to the old training data. Legacy data may be unrecorded, proprietary, or simply too cumbersome to use in training a new task. Though similar in spirit to transfer, multitask, and lifelong learning, we are not aware of any work that provides a solution to the problem of continually adding new prediction tasks based on adapting shared parameters *without access to training data* for previously learned tasks.

In this paper, we demonstrate a simple but effective solution on a variety of image classification problems with Convolutional Neural Network (CNN) classifiers. In our setting, a CNN has a set of shared parameters θ_s (e.g., five convolutional layers and two fully connected layers for AlexNet [11] architecture), task-specific parameters for previously learned tasks θ_o (e.g., the output layer

Fig. 1. We wish to add new prediction tasks to an existing CNN vision system without requiring access to the training data for existing tasks. This table shows relative advantages of our method compared to commonly used methods.

	Fine Tuning	Duplicating and Fine Tuning	Feature Extraction	Joint Training	Learning without Forgetting
new task performance	good	good	X medium	best	✓best
original task performance	X bad	good	good	good	✓good
training efficiency	fast	fast	fast	X slow	✓fast
testing efficiency	fast	X slow	fast	fast	✓fast
storage requirement	medium	X large	medium	X large	✓medium
requires previous task data	no	no	no	X yes	✓no

for ImageNet [19] classification and corresponding weights), and randomly initialized task-specific parameters for new tasks θ_n (e.g., scene classifiers). It is useful to think of θ_o and θ_n as classifiers that operate on features parameterized by θ_s. Currently, there are three common approaches (Figures 1, 2) to learning θ_n while benefiting from previously learned θ_s:

Feature Extraction (e.g., [6]): θ_s and θ_o are fixed (unchanged), and the outputs of one or more layers are used as features for the new task in training θ_n.

Fine-tuning (e.g., [9]): θ_s and θ_n are optimized for the new task, while θ_o is fixed. A low learning rate is typically used to prevent large drift in θ_s. Potentially, the original network could be duplicated and fine-tuned for each new task to create a set of specialized networks.

Joint Training (e.g., [3]): All parameters θ_s, θ_o, θ_n are jointly optimized, for example by interleaving samples from each task.

Each of these strategies has a major drawback. Feature extraction typically underperforms on the new task because the shared parameters fail to represent some information that is discriminative for the new task. Fine-tuning degrades performance on previously learned tasks because the shared parameters change without new guidance for the original task-specific prediction parameters. Duplicating and fine-tuning for each task results in linearly increasing test time as new tasks are added, rather than sharing computation for shared parameters. Joint training becomes increasingly cumbersome in training as more tasks are learned and is not possible if the training data for previously learned tasks is unavailable.

We propose a new strategy that we call **Learning without Forgetting** (LwF). Using only examples for the new task, we optimize both for high accuracy for the new task and for preservation of responses on the existing tasks from the original network. Clearly, if the new network produces exactly the same outputs on all relevant images, its accuracy will be the same as the original network. In practice, the images for the new task may provide a poor sampling of the original task domain, but our experiments show that preserving outputs on these examples is still an effective strategy to preserve performance on the old task and also has an unexpected benefit of acting as a regularizer to improve
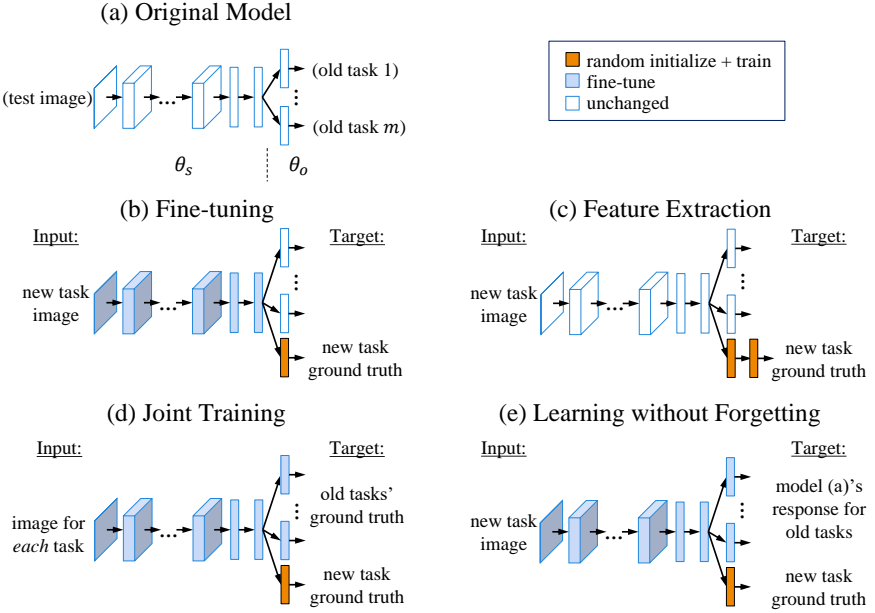
Fig. 2. Illustration for our method (e) and methods we compare to (b-d). Images and labels used in training are shown. Data for different tasks are used in alternation in joint training.

performance on the new task. Our Learning without Forgetting approach has several advantages:

(1) Classification performance: We show that Learning without Forgetting outperforms feature extraction and, more surprisingly, fine-tuning on the new task while greatly outperforming using fine-tuned parameters θ_s on the old task.
(2) Computational efficiency: Training time is faster than joint training and only slightly slower than fine-tuning, and test time is faster than if one uses multiple fine-tuned networks for different tasks.
(3) Simplicity in deployment: Once a task is learned, the training data does not need to be retained or reapplied to preserve performance in the adapting network.

2   Related work

Multi-task learning, transfer learning, and related methods have a long history. In brief, our Learning without Forgetting approach could be seen as a combination of Distillation Networks [10] and fine-tuning [9]. Fine-tuning initializes with parameters from an existing network trained on a related data-rich problem

and finds a new local minimum by optimizing parameters for a new task with a low learning rate. The idea of Distillation Networks is to learn parameters in a simpler network that produce the same outputs as a more complex ensemble of networks either on the original training set or a large unlabeled set of data. Our approach differs in that we solve for a set of parameters that works well on both old and new tasks using the same data to supervise learning of the new tasks and to provide unsupervised output guidance on the old tasks.

We now summarize existing methods for transfer and multitask learning and compare them to our approach.

Feature Extraction [6,17] uses a pre-trained deep CNN to compute features for an image. The extracted features are the activations of one layer (usually the last hidden layer) or multiple layers given the image. Classifiers trained on these features can achieve competitive results, sometimes outperforming human-engineered features [6]. Further studies [2] show how hyper-parameters, e.g. original network structure, should be selected for better performance. Feature extraction does not modify the original network and allows new tasks to benefit from complex features learned from previous tasks. However, these features are not specialized for the new task and can often be improved by fine-tuning.

Fine-tuning [9] modifies the parameters of an existing CNN to train a new task. The output layer is extended with randomly intialized weights for the new task and a small learning rate is used to tune parameters from their original values to minimize the loss on the new task. Using appropriate hyper-parameters for training, the resulting model often outperforms feature extraction [9,2] or learning from a randomly initialized network [1,26]. Fine-tuning adapts the shared parameters θ_s to make them more discriminative for the new task, and the low learning rate is an indirect mechanism to preserve some of the representational structure learned in the original tasks. Our method provides a more direct way to preserve representations that are important for the original task, improving both original and new task performance relative to fine-tuning.

Adding new nodes to each network layer is a way to preserve the original network parameters while learning new discriminative features. For example, Terekhov et al. [21] proposes Deep Block-Modular Neural Networks for fully-connected neural networks. Parameters for the original network are untouched, and newly added nodes are fully connected to the layer beneath them. This method has the downside of substantially expanding the number of parameters in the network, and can underperform both fine-tuning and feature extraction if insufficient training data is available to learn the new parameters. We experiment with expanding the fully connected layers of original network but find that the expansion does not provide an improvement on our original approach.

Our work also relates to **methods that transfer knowledge** between networks. Hinton et al. [10] propose Knowledge Distillation, where knowledge is transferred from a large network or a network assembly to a smaller network for efficient deployment. The smaller network is trained using a modified cross-entropy loss (further described in Sec. 3) that encourages both large and small responses of the original and new network to be similar. Romero et al. [18] builds

on this work to transfer to a deeper network by applying extra guidance on the middle layer. Chen et al. [5] proposes the Net2Net method that immediately generates a deeper, wider network that is functionally equivalent to an existing one. This technique can quickly initialize networks for faster hyper-parameter exploration. These methods aim to produce a differently structured network that approximates the original network, while we aim to find new parameters for the original network structure (θ_s, θ_o) that approximate the original outputs while tuning shared parameters θ_s for new tasks.

Feature extraction and fine-tuning are special cases of **Domain Adaptation** (when old and new tasks are the same) or **Transfer Learning** (different tasks). Transfer Learning uses knowledge from one task to help another, as surveyed by Pan et al. [15]. The Deep Adaption Network by Long et al. [13] matches the RKHS embedding of the deep representation of both source and target tasks to reduce domain bias. Another similar domain adaptation method is by Tzeng et al. [23], which encourages the shared deep representation to be indistinguishable across domains. This method also uses knowledge distillation, but to help train the *new* domain instead of preserving the old task. Domain adaptation and transfer learning require that at least unlabeled data is present for both task domains. In contrast, we are interested in the case when training data for the original tasks (i.e. source domains) are not available.

Multitask learning (e.g., [3]) differs from transfer learning in that it aims at improving all tasks simultaneously by combining the common knowledge from all tasks. Each task provides extra training data for the parameters that are shared or constrained, serving as a form of regularization for the other tasks [4]. For neural networks, Caruana [3] gives a detailed study of multi-task learning. Usually the bottom layers of the network are shared, while the top layers are task-specific. Multitask learning requires data from all tasks to be present, while our method requires only data for the new tasks.

Methods that **integrate knowledge over time**, e.g. Lifelong Learning [22] and Never Ending Learning [14], are also related. Lifelong learning focuses on flexibly adding new tasks while transferring knowledge between tasks. Never Ending Learning focuses on building diverse knowledge and experience (e.g. by reading the web every day). Though topically related to our work, these methods do not provide a way to preserve performance on existing tasks without the original training data. Ruvolo et al. [7] describe a method to efficiently add new tasks to a multitask system, co-training all tasks while using only new task data. However, the method assumes that weights for all classifiers and regression models can be linearly decomposed into a set of bases. In contrast with our method, the algorithm applies only to logistic or linear regression on engineered features, and these features cannot be made task-specific, e.g. by fine-tuning.

3   Learning without forgetting

Given a CNN with shared parameters θ_s and task-specific parameters θ_o (Fig. 2(a)), our goal is to add task-specific parameters θ_n for a new task and to learn param-

LEARNINGWITHOUTFORGETTING:
    Start with:
        θ_s: shared parameters
        θ_o: task specific parameters for each old task
        X_n, Y_n: training data and ground truth on the new task
    Initialize:
        $Y_o \leftarrow \text{CNN}(X_n, \theta_s, \theta_o)$ // *compute output of old tasks for new data*
        $\theta_n \leftarrow \text{RANDINIT}(|\theta_n|)$ // *randomly initialize new parameters*
    Train:
        Define $\hat{Y}_o \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_o)$ // *old task output*
        Define $\hat{Y}_n \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_n)$ // *new task output*
        $\theta_s^*, \; \theta_o^*, \; \theta_n^* \leftarrow \underset{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n}{\operatorname{argmin}} \left(\mathcal{L}_{old}(Y_o, \hat{Y}_o) + \mathcal{L}_{new}(Y_n, \hat{Y}_n) + \mathcal{R}(\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n) \right)$

Fig. 3. Procedure for Learning without Forgetting.

eters that work well on old and new tasks, using images and labels from only the new task (i.e., *without using data from existing tasks*). Our algorithm is outlined in Fig. 3, and the network structure illustrated in Fig. 2(e).

First, we record responses $\mathbf{y}_o$ on each new task image from the original network for outputs on the old tasks (defined by θ_s and θ_o). Our experiments involve classification, so the responses are the set of label probabilities for each training image. Nodes for each new class are added to the output layer, fully connected to the layer beneath, with randomly initialized weights θ_n. The number of new parameters is equal to the number of new classes times the number of nodes in the last shared layer, typically a very small percent of the total number of parameters. In our experiments (Sec. 4.2), we also compare alternate ways of modifying the network for the new task.

Next, we train the network to minimize loss for all tasks and regularization $\mathcal{R}$ using stochastic gradient descent. The regularization $\mathcal{R}$ corresponds to a simple weight decay of 0.0005. When training, we first freeze θ_s and θ_o and train θ_n to convergence. Then, we jointly train all weights until convergence.

For simplicity, we denote the loss functions, outputs, and ground truth for single examples. The total loss is averaged over all images in a batch in training. For new tasks, the loss encourages predictions $\hat{\mathbf{y}}_n$ to be consistent with the ground truth $\mathbf{y}_n$. The tasks in our experiments are multiclass classification, so we use the common [11,20] multinomial logistic loss:

$$\mathcal{L}_{new}(\mathbf{y}_n, \hat{\mathbf{y}}_n) = -\mathbf{y}_n \cdot \log \hat{\mathbf{y}}_n \tag{1}$$

where $\hat{\mathbf{y}}_n$ is the softmax output of the network and $\mathbf{y}_n$ is the one-hot ground truth label vector. If there are multiple new tasks, or if the task is multi-label classification where we make true/false predictions for each label, we take the sum of losses across the new tasks and the labels.

For each original task, we want the output probabilities for each image to be close to the recorded output from the original network. We use the Knowledge

Distillation loss, which was found by Hinton et al. [10] to work well for encouraging the outputs of one network to approximate the outputs of another. This is a modified cross-entropy loss that increases the weight for smaller probabilities:

$$\mathcal{L}_{old}(\mathbf{y}_o, \hat{\mathbf{y}}_o) = -H(\mathbf{y}'_o, \hat{\mathbf{y}}'_o) = -\sum_{i=1}^{l} y'^{(i)}_o \log \hat{y}'^{(i)}_o \tag{2}$$

where l is the number of labels and $y'^{(i)}_o$, $\hat{y}'^{(i)}_o$ are the modified versions of recorded and current probabilities $y^{(i)}_o$, $\hat{y}^{(i)}_o$:

$$y'^{(i)}_o = \frac{(y^{(i)}_o)^{1/T}}{\sum_j (y^{(j)}_o)^{1/T}}, \quad \hat{y}'^{(i)}_o = \frac{(\hat{y}^{(i)}_o)^{1/T}}{\sum_j (\hat{y}^{(j)}_o)^{1/T}}. \tag{3}$$

If there are multiple old tasks, or if an old task is multi-label classification, we take the sum of the loss for each old task and label. Hinton et al. [10] suggest that setting $T > 1$, which increases the weight of smaller logit values and encourages the network to better encode similarities among classes. We use $T = 2$ according to a grid search on a held out set, which aligns with the author's recommendations. In experiments, use of knowledge distillation loss leads to similar performance to other reasonable losses. Therefore, it is important to constrain outputs for original tasks to be similar to the original network, but the similarity measure is not crucial.

Implementation details. We use MatConvNet [24] to train our networks using stochastic gradient descent with momentum of 0.9 and dropout enabled in the fully connected layers. The data normalization of the original task is used for the new task. The resizing follows the implementation of the original network, which is 256×256 for AlexNet and 256 pixels in the shortest edge with aspect ratio preserved for VGG. We randomly jitter the training data by taking random fixed-size crops of the resized images and adding variance to the RGB values, as with AlexNet. This data augmentation is applied to feature extraction too.

When training networks, we follow the standard practices for fine-tuning existing networks. We use a learning rate much smaller than when training the original network ($0.1 \sim 0.02$ times the original rate), and lower it once by $10\times$ after the accuracy on a held out set plateaus. The learning rates are selected to maximize new task performance with a reasonable number of epochs. The compared methods converge at similar speeds, so we used the same number of epochs for each method (but not the same for different task pairs). For each scenario, the same learning rate are shared by all methods except feature extraction, which uses $5\times$ the learning rate due to its small number of parameters.

For the feature extraction baseline, we extract features as the last hidden layer of the original network and classify with a two-layer network with 4096 nodes in the hidden layer. For joint training, loss for one task's output nodes is only applied for its own training images. The same number of images are subsampled for every task in each epoch to balance their loss, and we interleave batches of different tasks for gradient descent.

Efficiency comparison. The most computationally expensive part of using the neural network is evaluating or back-propagating through the shared parameters θ_s, especially the convolutional layers. For training, feature extraction is the fastest because only the new task parameters are tuned. LwF is slightly slower than fine-tuning because it needs to back-propagate through θ_o for old tasks but needs to evaluate and back-propagate through θ_s only once. Joint training is the slowest, because different images are used for different tasks, and each task requires separate back-propagation through the shared parameters.

All methods take approximately the same amount of time to evaluate a test image. However, duplicating the network and fine-tuning for each task takes m times as long to evaluate as the other methods, where m is the total number of tasks.

4   Experiments

Our experiments are designed to evaluate whether Learning without Forgetting (LwF) is an effective method to learn a new task while preserving performance on old tasks. We compare to baselines of *feature extraction* and *fine-tuning*, which are common approaches to leverage an existing network for a new task without requiring training data for the original tasks. Feature extraction maintains the exact performance on the original task. We also compare to *joint training* (sometimes called multitask learning) as an upper-bound on possible performance, since joint training uses images and labels for original and new tasks, while LwF uses only images and labels for the new tasks.

We experiment on a variety of image classification problems with varying degrees of inter-task similarity. For the original ("old") task, we consider the ILSVRC 2012 subset of *ImageNet* [19] and the *Places2* [27] taster challenge in ILSVRC 2015 [19]. ImageNet has 1,000 object category classes and more than 1,000,000 training images. Places2 has 401 scene classes and more than 8,000,000 training images. We use these large datasets also because we assume we start from a well-trained network, which implies a large-scale dataset. For the new tasks, we consider PASCAL *VOC 2012 image classification* [8] ("VOC"), *Caltech-UCSD fine-grained bird classification* from the 2011 dataset [25] ("CUB"), and *MIT indoor scene classification* [16] ("Scenes"). These datasets have a moderate number of images for training: 5,717 for VOC; 5,994 for CUB; and 5,360 for Scenes. Among these, VOC is very similar to ImageNet, as subcategories of its labels can be found in ImageNet classes. MIT indoor scene dataset is in turn similar to Places2. CUB is dissimilar to both, since it includes only birds and requires capturing the fine details of the image to make a valid prediction. In one experiment, we use MNIST [12] as the new task expecting our method to underperform, since the hand-written characters are completely unrelated to ImageNet classes.

We mainly use the AlexNet [11] network structure because it is fast to train and well-studied by the community [26,2,9]. We also verify that similar results

Table 1. Performance for the single new task scenario. For all tables, the **difference** of methods' performance with LwF (our method) is reported to facilitate comparison. Mean Average Precision is reported for VOC and accuracy for all others. On the new task, LwF outperforms baselines, and performs comparably with joint training, which uses old task training data we consider unavailable for the other methods. On the old task, our method greatly outperforms fine-tuning and achieves similar performance to joint training. An exception is the MNIST task where LwF does not perform well.

(a) Using AlexNet structure (validation performance for ImageNet/Places2/VOC)

	ImageNet→VOC		ImageNet→CUB		ImageNet→Scenes		Places2→VOC		Places2→CUB		Places2→Scenes		ImageNet→MNIST	
	old	new	old	new	old	new	old	new	old	new	old	new	old	new
LwF (ours)	56.5	75.8	55.1	57.5	55.9	64.5	43.3	72.1	38.4	41.7	43.0	75.3	52.1	99.0
fine-tuning	-1.4	-0.3	-5.1	-1.5	-3.4	-1.0	-1.8	-0.1	-9.1	-0.8	-4.1	-0.8	-4.9	0.2
feat. extraction	0.5	-1.1	2.0	-5.3	1.2	-3.7	-0.2	-3.9	4.7	-19.4	0.2	-0.5	5.0	-0.8
joint training	0.2	0.0	0.5	-0.9	0.5	-0.6	-0.1	0.1	3.3	-0.2	0.2	0.1	4.7	0.2

(b) Test set performance

	Places2→VOC	
	old	new
LwF (ours)	41.1	75.2
fine-tuning	-1.9	-0.1
feat. extraction	0.1	-3.5
joint training	0.0	0.0

(c) Using VGGnet structure

	ImageNet→CUB		ImageNet→Scenes	
	old	new	old	new
LwF (ours)	65.6	72.3	68.1	74.7
fine-tuning	-11.0	-0.2	-5.6	-0.7
feat. extraction	3.1	-9.1	0.7	-5.1
joint training	2.5	2.3	2.0	0.8

hold using 16-layer VGGnet [20] on a smaller set of experiments. The original networks pre-trained on ImageNet and Places2 are obtained from public online sources. At suggestion of the authors of Places2, we fine-tuned the provided Places2 original network on the Places2 training set, due to its sensitivity to image rescaling methods, which slightly improved performance (44% top-1 validation accuracy with 10 jitters) compared to the reported 43%.

We report the center image crop mean average precision for VOC, and center image crop accuracy for all other tasks. We report the accuracy of the validation set of VOC, ImageNet and Places2, and on the test set of CUB and Scenes dataset. Since the test performance of the former three cannot be evaluated frequently, we only provide the performance on their test sets in one experiment.

Our experiments investigate adding a single new task to the network or adding multiple tasks one-by-one. We also examine effect of dataset size and network design. In ablation studies, we examine alternative response-preserving losses, the utility of expanding the network structure, and fine-tuning with a lower learning rate as a method to preserve original task performance.

4.1 Main experiments

Single new task scenario. First, we compare the results of learning one new task among different task pairs and different methods. Table 1(a), 1(b) shows the performance of our method, and the relative performance of other methods compared to it using AlexNet. We make the following observations:

On the new task, our method consistently outperforms fine-tuning and fea-ture extraction except for ImageNet→MNIST. The gain over fine-tuning was unexpected and indicates that preserving outputs on the old task is an effective regularizer. (See Section 5 for a brief discussion). This finding motivates replacing fine-tuning with LwF as the standard approach for adapting a network to a new task.

On the old task, our method performs better than fine-tuning but often under-performs feature extraction. By changing shared parameters θ_s, fine-tuning significantly degrades performance on the task for which the original network was trained. By jointly adapting θ_s and θ_o to generate similar outputs to the original network on the old task, the performance loss is greatly reduced.

Our method performs similarly to joint training. Our method tends to slightly outperform joint training on the new task but underperform on the old task, which we attribute to a different balance of the losses in the two methods. Overall, the methods perform similarly, a positive result since our method does not require access to the old task training data and is faster to train.

Dissimilar new tasks degrade old task performance more. For example, CUB is very dissimilar task from Places2 [2], and adapting the network to CUB leads to a Places2 accuracy loss of 13.8% (4.7% + 9.1%) for fine-tuning, 4.7% for LwF, and 1.4% (4.7% − 3.3%) for joint training. In these cases, learning the new task causes considerable drift in the shared parameters, which cannot fully be accounted for by LwF because the distribution of CUB and Places2 images is very different. Even joint training leads to more accuracy loss on Places2→CUB's old task because it cannot find a set of shared parameters that works well for both tasks. As expected, our method does not outperform fine-tuning for ImageNet-¿MNIST on the new task, since the hand-written characters provide poor indirect supervision for the old task, and the old task accuracy drops substantially with both methods, though more with fine-tuning.

Similar observations hold for both VGG and AlexNet structures, except that joint training outperforms consistently for VGG (Table 1(c)), indicating that these results are likely to hold for other network structures as well, though joint training may have a larger benefit on networks with more representational power.

Multiple new task scenario. Second, we compare different methods when we cumulatively add new tasks to the system, simulating a scenario in which new object or scene categories are gradually added to the prediction vocabulary. We experiment on gradually adding VOC task to AlexNet trained on Places2, and adding Scene task to AlexNet trained on ImageNet. These pairs have moderate difference between original task and new tasks. We split the new task classes into three parts according to their similarity – VOC into transport, animals and objects, and Scenes into large rooms, medium rooms and small rooms. The images in Scenes are split into these three subsets. Since VOC is a multilabel dataset, it is not possible to split the images into different categories, so the labels are split for each task and images are shared among all the tasks.
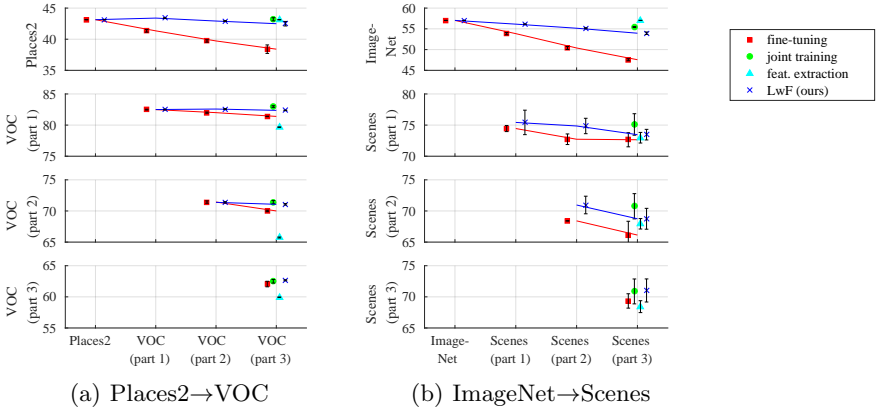
(a) Places2→VOC   (b) ImageNet→Scenes

Fig. 4. Performance of each task when gradually adding new tasks to a pre-trained network. Different tasks are shown in different sub-graphs. The x-axis labels indicate the new task added to the network each time. Error bars shows ± 2 standard deviations for 3 runs with different θ_n random initializations. Markers are jittered horizontally for visualization, but line plots are not jittered to facilitate comparison. For all tasks, our method degrades slower over time than fine-tuning and outperforms feature extraction. For Places2→VOC, our method performs comparably to joint training.

Each time a new task is added, the responses of all other tasks Y_o are re-computed, to emulate the situation where data for *all* original tasks are unavailable. Therefore, Y_o for older tasks changes each time. For feature extractor and joint training, cumulative training does not apply, so we only report their performance on the final stage where all tasks are added. Figure 4 shows the results on both dataset pairs. Our findings are usually consistent with the single new task scenario: *LwF outperforms fine-tuning on all tasks, outperforms feature extraction for new tasks, and except on the old tasks in ImageNet→Scenes, performs similarly overall to joint training.*

Influence of dataset size. We inspect whether the size of the new task dataset affects our performance relative to other methods. We perform this experiment on adding VOC to Places2 AlexNet. We subsample the VOC dataset to 30%, 10% and 3% when training the network, and report the result on the entire validation set. Note that for joint training, since each dataset has a different size, the same number of images are subsampled to train both tasks (resampled each epoch), which means a smaller number of Places2 images being used at one time. Our results are shown in Figure 5. Results show that *the same observations hold, except that our method suffers more than joint training on the old task as the number of examples is decreased. Differences between LwF and fine-tuning on the old task and between LwF and feature extraction on the new task increase with less data.*
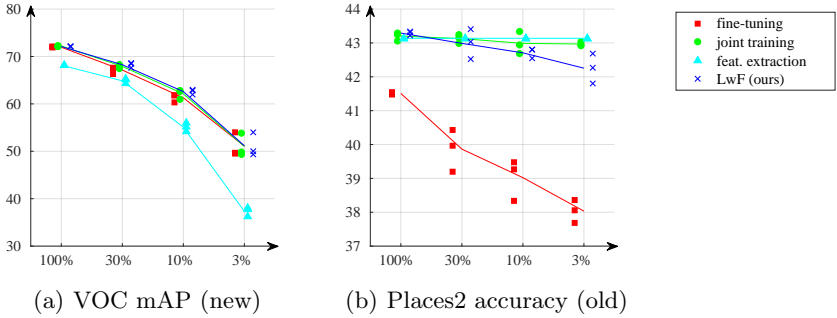
(a) VOC mAP (new)   (b) Places2 accuracy (old)

Fig. 5. Influence of subsampling new task training set on compared methods. The x-axis indicates diminishing training set size. Three runs of our experiments with different random θ_n initialization and dataset subsampling are shown. Scatter points are jittered horizontally for visualization, but line plots are not jittered to facilitate comparison. Differences between LwF and fine-tuning on the old task and between LwF and feature extraction on the new task increase with less data.
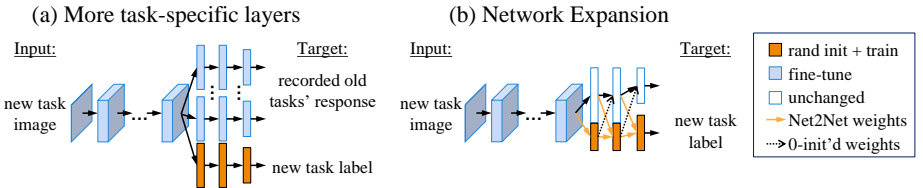

(a) More task-specific layers   (b) Network Expansion

Fig. 6. Illustration for alternative network modification methods. In (a), more fully connected layers are task-specific, rather than shared. In (b), nodes for multiple old tasks (not shown) are connected in the same way. Learning without Forgetting can also be applied to Network Expansion by unfreezing all nodes and matching output responses on the old tasks.

4.2   Design choices and alternatives

Choice of task-specific layers. It is possible to regard more layers as task-specific θ_o, θ_n (see Figure 6(a)) instead of regarding only the output nodes as task-specific. This may provide advantage for both tasks because later layers tend to be more task specific [2]. However, doing so requires more storage, as most parameters in AlexNet are in the first two fully connected layers. Table 2(a) shows the comparison on three task pairs. *Our results do not indicate any advantage to having additional task-specific layers.*

Network expansion. We explore another way of modifying the network structure, which we refer to as "network expansion", which adds nodes to some layers. This allows for extra new-task-specific information in the earlier layers while still using the original network's information.

Table 2. Performance of our method versus various alternative design choices. In most cases, these alternative choices do not provide consistent advantage or disadvantage compared to our method.

(a) Changing the number of task-specific layers

	Places2→VOC		ImageNet→Scenes		ImageNet→CUB	
starting from	old	new	old	new	old	new
output layer (ours)	43.3	72.1	55.9	64.5	55.1	57.5
last hidden layer	43.1	72.5	55.5	64.2	54.8	56.5
2nd last hidden (Fig. 6(a))	43.1	72.6	55.8	64.2	55.2	56.1

(b) Comparison of our method, network expansion, and L2 soft-constraint. Performance relative to Learning without Forgetting is reported. Fine-tuning and feature extraction are shown for comparison. Network expansion by itself underperforms on the new task. Using it with LwF yields worse to similar performance as LwF. L2 soft-constraint produces results between fine-tuning and feature extraction.

	ImageNet→VOC		ImageNet→CUB		ImageNet→Scenes		Places2→VOC		Places2→CUB		Places2→Scenes	
	old	new	old	new	old	new	old	new	old	new	old	new
LwF (ours)	56.5	75.8	55.1	57.5	55.9	64.5	43.3	72.1	38.4	41.7	43.0	75.3
fine-tuning	-1.4	-0.3	-5.1	-1.5	-3.4	-1.0	-1.8	-0.1	-9.1	-0.8	-4.1	-0.8
feat. extraction	0.5	-1.1	2.0	-5.3	1.2	-3.7	-0.2	-3.9	4.7	-19.4	0.2	-0.5
network expansion	0.5	-1.1	2.0	-3.6	1.2	-2.0	-0.2	-3.6	4.7	-6.9	0.2	0.0
network expansion + LwF	0.0	0.1	-0.3	-0.5	-0.2	-0.4	0.0	0.2	1.3	-0.1	-0.3	-0.2
L2 weight soft-constraint	-1.0	-0.4	-4.2	-2.0	-3.0	-1.0	-1.5	-0.5	-8.5	-3.3	-4.0	-0.4

(c) Changing the response-preserving loss. Only L_2 underperforms on the old task.

	Places2	VOC
LwF (L_1)	43.5	71.9
LwF (L_2)	42.2	72.3
LwF (cross-entropy)	43.6	71.8
LwF (know. distill.)	43.3	72.1

(d) Attempt to lower θ_s's learning rate when fine-tuning. The degradation of old task performance is not avoided, and the new task performance drops.

	ImageNet→VOC		Places2→VOC	
	old	new	old	new
fine-tuning	55.2	75.6	41.5	71.9
fine-tuning (10% θ_s learning rate)	55.9	75.5	41.5	71.4

Figure 6(b) illustrates this method. We add 1024 nodes to each layer of the top 3 layers. The weights from all nodes at previous layer to the new nodes at current layer are initialized the same way Net2Net [5] would expand a layer by copying nodes. Weights from new nodes at previous layer to the original nodes at current layer are initialized to zero. The top layer weights of the new nodes are randomly re-initialized. Then we either freeze the existing weights and fine-tune the new weights on the new task ("network expansion"), or train using Learning without Forgetting as before ("network expansion + LwF").

Table 2(b) shows the comparison with our original method. *Network expansion by itself performs better than feature extraction, but neither variant performs as well as LwF on new tasks.* We leave exploration of other possible versions of network expansion (e.g. number of top layers to expand, number of new nodes at each layer, parameter initialization method) as future work.

L2 soft-constrained weights. Perhaps an obvious alternative to LwF is to

keep the network parameters (instead of the response) close to the original. We compare with the baseline that adds $\frac{1}{2}\lambda_c\|w - w_0\|^2$ to the loss for fine-tuning, where w and w_0 are flattened vectors of all shared parameters θ_s and their original values. Coefficient λ is set to 0.5 for VOC and 0.05 for other new tasks.

As shown in Table 2(b), *our method outperforms this baseline, which produces a result between feature extraction (no parameter change) and fine-tuning (free parameter change).* We believe that by regularizing the output, our method maintains old task performance better than regularizing individual parameters, since many small parameter changes could cause big changes in the outputs.

Choice of response preserving loss. We compare the use of L_1, L_2, cross-entropy loss, and knowledge distillation loss with $T = 2$ for keeping $\mathbf{y}'_o, \hat{\mathbf{y}}'_o$ similar. We test on adding VOC to Places2 AlexNet. Table 2(c) shows our results. *Results indicate no clear overall advantage or disadvantage for any loss, though L_2 underperforms on the original task.*

Effect of lower learning rate of shared parameters. We investigate whether a simple solution of lowering the learning rate of the shared parameters θ_s would be enough for preserving the original task performance. Result of lowering θ_s's learning rate by $10\times$ is shown in Table 2(d). Lowering the learning rate does not prevent fine-tuning from significantly reducing original task performance, and it reduces new task performance. This shows that *simply reducing the learning rate of shared layers is insufficient for original task preservation.*

5   Discussion

We address the problem of adapting a vision system to a new task while preserving performance on original tasks, without access to training data for the original tasks. We propose the Learning without Forgetting method for convolutional neural networks, which can be seen as a hybrid of knowledge distillation and fine-tuning, learning parameters that are discriminative for the new task while preserving outputs for the original tasks on the training data.

This work has implications for two uses. First, if we want to expand the set of possible predictions on an existing network, our method performs similarly to joint training but is faster to train and does not require access to the training data used to learn previous tasks. Second, if we care only about the performance for the new task, our method consistently outperforms the current standard practice of fine-tuning. Fine-tuning approaches use a low learning rate in hopes that the parameters will settle in a "good" local minimum that is not too far from the original values. Preserving outputs on the old task is a more direct and interpretable way to to retain the important shared structures learned for the previous tasks.

We see several directions for future work. We have demonstrated the effectiveness of LwF for image classification problems but would like to further experiment on semantic segmentation, detection, and problems outside of com-

puter vision. Additionally, one could explore variants of the approach, such as maintaining a set of unlabeled images to serve as representative examples for previously learned tasks. Theoretically, it would be interesting to bound the performance on the original task based on preserving outputs for a sample drawn from a different distribution. More generally, there is a need for approaches that are suitable for online learning across different tasks, especially when classes have heavy tailed distributions.

Acknowledgement

References

1. Agrawal, P., Girshick, R., Malik, J.: Analyzing the performance of multilayer neural networks for object recognition. In: Proceedings of the European Conference on Computer Vision (ECCV) (2014)
2. Azizpour, H., Razavian, A., Sullivan, J., Maki, A., Carlsson, S.: Factors of transferability for a generic convnet representation. In: IEEE Transactions on Pattern Analysis & Machine Intelligence (2014)
3. Caruana, R.: Multitask learning. Machine learning 28(1), 41–75 (1997)
4. Chapelle, O., Shivaswamy, P., Vadrevu, S., Weinberger, K., Zhang, Y., Tseng, B.: Boosted multi-task learning. Machine learning 85(1-2), 149–173 (2011)
5. Chen, T., Goodfellow, I., Shlens, J.: Net2net: Accelerating learning via knowledge transfer. In: Proceedings of the International Conference on Learning Representations (ICLR). p. to appear (2016)
6. Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., Darrell, T.: Decaf: A deep convolutional activation feature for generic visual recognition. In: International Conference in Machine Learning (ICML) (2014)
7. Eaton, E., Ruvolo, P.L.: Ella: An efficient lifelong learning algorithm. In: Proceedings of the 30th International Conference on Machine Learning. pp. 507–515 (2013)
8. Everingham, M., Eslami, S.M.A., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The pascal visual object classes challenge: A retrospective. International Journal of Computer Vision 111(1), 98–136 (Jan 2015)
9. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2014)
10. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. In: NIPS Workshop (2014)
11. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
12. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE 86(11), 2278–2324 (1998)
13. Long, M., Wang, J.: Learning transferable features with deep adaptation networks. arXiv preprint arXiv:1502.02791 (2015)

14. Mitchell, T., Cohen, W., Hruschka, E., Talukdar, P., Betteridge, J., Carlson, A., Dalvi, B., Gardner, M., Kisiel, B., Krishnamurthy, J., Lao, N., Mazaitis, K., Mohamed, T., Nakashole, N., Platanios, E., Ritter, A., Samadi, M., Settles, B., Wang, R., Wijaya, D., Gupta, A., Chen, X., Saparov, A., Greaves, M., Welling, J.: Never-ending learning. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15) (2015)
15. Pan, S.J., Yang, Q.: A survey on transfer learning. Knowledge and Data Engineering, IEEE Transactions on 22(10), 1345–1359 (2010)
16. Quattoni, A., Torralba, A.: Recognizing indoor scenes. In: Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. pp. 413–420 (2009)
17. Razavian, A., Azizpour, H., Sullivan, J., Carlsson, S.: Cnn features off-the-shelf: an astounding baseline for recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. pp. 806–813 (2014)
18. Romero, A., Ballas, N., Kahou, S.E., Chassang, A., Gatta, C., Bengio, Y.: Fitnets: Hints for thin deep nets. In: Proceedings of the International Conference on Learning Representations (ICLR) (2015)
19. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) 115(3), 211–252 (2015)
20. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR abs/1409.1556 (2014)
21. Terekhov, A.V., Montone, G., ORegan, J.K.: Knowledge transfer in deep block-modular neural networks. In: Biomimetic and Biohybrid Systems. pp. 268–279. Springer (2015)
22. Thrun, S.: Lifelong learning algorithms. In: Learning to learn, pp. 181–209. Springer (1998)
23. Tzeng, E., Hoffman, J., Darrell, T., Saenko, K.: Simultaneous deep transfer across domains and tasks. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 4068–4076 (2015)
24. Vedaldi, A., Lenc, K.: Matconvnet – convolutional neural networks for matlab. In: Proceeding of the ACM Int. Conf. on Multimedia (2015)
25. Wah, C., Branson, S., Welinder, P., Perona, P., Belongie, S.: The Caltech-UCSD Birds-200-2011 Dataset. Tech. Rep. CNS-TR-2011-001, California Institute of Technology (2011)
26. Yosinski, J., Clune, J., Bengio, Y., Lipson, H.: How transferable are features in deep neural networks? In: Advances in Neural Information Processing Systems. pp. 3320–3328 (2014)
27. Zhou, B., Khosla, A., Lapedriza, A., Torralba, A., Oliva, A.: Places2: A large-scale database for scene understanding. arXiv preprint, to appear (2015)