Learning without Forgetting

Zhizhong Li and Derek Hoiem

Abstract—When building a unified vision system or gradually adding new apabilities 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 with similar old and new task datasets for improved new task performance.

Index Terms—Convolutional neural networks, transfer learning, multi-task 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 suffering from Catastrophic Forgetting [1], [2] (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. This problem is similar in spirit to transfer, multitask, and lifelong learning.

We aim at developing a simple but effective strategy 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 [3] architecture), task-specific parameters for previously learned tasks θ_o (e.g., the output layer for ImageNet [4] 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 (Figs. 1, 2b-d) to learning θ_n while benefiting from previously learned θ_s, and they differ mostly on which parameters are unchanged:

Feature Extraction (e.g., [5]): θ_s and θ_o are 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., [6]): 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.

It is also possible to use a variation of fine-tuning where part of θ_s – e.g., the convolutional layers—are frozen to prevent overfitting, and only top fully connected layers are fine-tuned. This can be seen as a compromise between fine-tuning and feature extraction. In this work we call this method *Fine-tuning FC* where FC stands for fully connected.

Joint Training (e.g., [7]): All parameters θ_s, θ_o, θ_n are jointly optimized, for example by interleaving samples from each task. This method's performance may be seen as an *upper bound* of what our proposed method can achieve.

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 (if all tasks are performed on each sample), rather than sharing computation for shared parameters. Fine-tuning FC, as we show in our experiments, still degrades performance on the new task. 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.

Besides these commonly used approaches, methods [8], [9] have emerged that can continually add new prediction tasks by adapting shared parameters *without access to training data* for previously learned tasks. (See Section 2)

In this paper, we expand on our previous work [10], *Learning without Forgetting* (LwF). Using only examples

• *The authors are with the Department of Computer Science, University of Illinois, Urbana Champaign, IL 61801. E-mail: {zli115, dhoiem}@illinois.edu.*

Manuscript received 10 Feb. 2017; revised 30 Aug. 2017; accepted 7 Nov. 2017. Date of publication 14 Nov. 2017; date of current version 1 Nov. 2018. (Corresponding author: Zhizhong Li.)
Recommended for acceptance by H. Larochelle.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPAMI.2017.2773081

	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

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. Limitations of our method include requiring knowing discrete task ID for each sample, requiring all new task data in advance, and the performance influenced by task similarity. (Section 5.1).

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. Our method is similar to joint training, except that our method does not need the old task's images and labels. Clearly, if the network is preserved such that θ_o produces exactly the same outputs on all relevant images, the old task 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 performance on the new task. Our Learning without Forgetting approach has several advantages:

(1) Classification performance: 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. Our method also generally perform better in experiments than recent alternatives [8], [9].

(2) Computational efficiency: Training time is faster than joint training and only slightly slower than fine-tuning, and test time of one sample on all tasks is faster than if one uses multiple fine-tuned networks for different tasks.

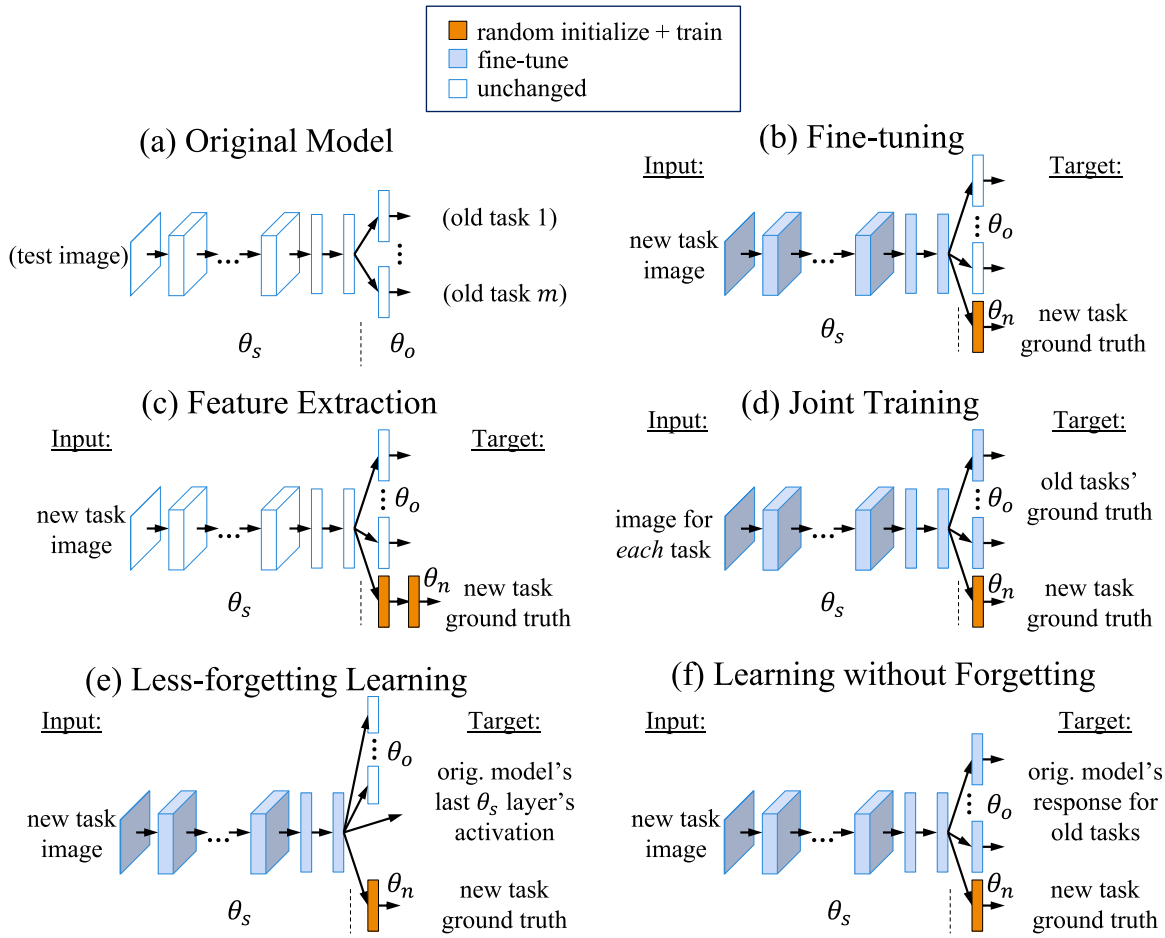


Fig. 2. Illustration for our method (f) and methods we compare to (b-e). Images and labels used in training are shown. Data for different tasks are used in alternation in joint training.

(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.

Compared to our previous work [10], we conduct more extensive experiments. We compare to additional methods – fine-tune FC, a commonly used baseline, and Less Forgetting Learning, a recently proposed method. We experiment on adjusting the balance between old-new task losses, providing a more thorough and intuitive comparison of related methods (Fig. 7). We switch from the obsolete Places2 to a newer Places365-standard dataset. We perform stricter, more careful hyperparameter selection process, which slightly changed our results. We also include more detailed explanation of our method. Finally, we perform an experiment on application to video object tracking in Appendix A, available in the online supplemental material.

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 [11] and fine-tuning [6]. 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.

2.1 Compared Methods

Feature Extraction [5], [12] (Fig. 2c) 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 [5]. Further studies [13] 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 [6] modifies the parameters of an existing CNN to train a new task. (Fig. 2b) As mentioned in Section 1, a small learning rate is often used, and sometimes part of the network is frozen to prevent overfitting. Using appropriate hyper-parameters for training, the resulting model often outperforms feature extraction [6], [13] or learning from a randomly initialized network [14], [15]. Fine-tuning makes θ_s 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 in most experiments.

Multitask learning (e.g., [7]; Fig. 2d) aims to improve 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 [16]. For neural networks, Caruana [7] 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.

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. [17] propose Deep Block-Modular Neural Networks for fully-connected neural networks, and Rusu et al. [18] propose Progressive Neural Networks for reinforcement learning. Parameters for the original network are untouched, and newly added nodes are fully connected to the layer beneath them. These methods has the downside of substantially expanding the number of parameters in the network, and can underperform [17] both fine-tuning and feature extraction if insufficient training data is available to learn the new parameters, since they require a substantial number of parameters to be trained from scratch. 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.

2.2 Topically Relevant Methods

Our work also relates to *methods that transfer knowledge* between networks. Hinton et al. [11] 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 Section 3) that encourages both large and small responses of the original and new network to be similar. Romero et al. [19] builds on this work to transfer to a deeper network by applying extra guidance on the middle layer. Chen et al. [20] 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). These are different from multitask learning in that tasks are not simultaneously optimized. Transfer Learning uses knowledge from one task to help another, as surveyed by Pan et al. [21]. The Deep Adaption Network by Long et al. [22] 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.

Methods that *integrate knowledge over time*, e.g., Lifelong Learning [24] and Never Ending Learning [25], 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. [26] 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.

2.3 Concurrently Developed Methods

Concurrent with our previous work [10], two methods have been proposed for continually add and integrate new tasks without using previous tasks' data.

A-LTM [8], developed independently, is nearly identical in method but has very different experiments and conclusions. The main differences of method are in the weight decay regularization used for training and the warm-up step that we use prior to full fine-tuning.

However, we use large datasets to train our initial network (e.g., ImageNet) and then extend to new tasks from smaller datasets (e.g., PASCAL VOC), while A-LTM uses small datasets for the old task and large datasets for the new task. The experiments in A-LTM [8] find much larger loss due to fine-tuning than we do, and the paper concludes that maintaining the data from the original task is necessary to maintain performance. Our experiments, in contrast, show that we can maintain good performance for the old task while performing as well or sometimes better than fine-tuning for the new task, without access to original task data. We believe the main difference is the choice of old-task new-task pairs and that we observe less of a drop in old-task performance from fine-tuning due to the choice (and in part to the warm-up step; see Table 2 b). We believe that our experiments, which start from a well-trained network and add tasks with less training data available, are better motivated from a practical perspective.

Less Forgetting Learning [9] is also a similar method, which preserves the old task performance by discouraging the shared representation to change (Fig. 2e). This method argues that the task-specific decision boundaries should not change, and the shared representation should not change. Therefore, LFL adds a L_2 loss that discourages the *output after* θ_s from changing for new task images, while θ_o remains as is. In comparison, our LwF method adds a loss that discourages the *old task output* to change for new task images, and jointly optimizes both the shared representation and all the final layers. We empirically show that our method outperforms Less Forgetting Learning on the new tasks.

Elastic Weight Consolidation (EWC) [27] can be seen as an advanced version of the L2 soft-constraint baseline in Section 4.2. Instead of adding a loss proportional to the L_2 distance between modified network weights and original ones, in EWC the squared distance of each parameter are weighted according to their importance to the old task. This weight is estimated using the diagonal precision matrix of an assumed gaussian distribution of the posterior of $P(\theta_s, \theta_o|D_A)$, which is again estimated using the Fisher matrix. Although the computation of the Fisher matrix requires the presence of the old task data D_A, fortunately this matrix can be computed before throwing away the old data, and be seen as part of the model. In this way the method would not require the old data to be present for new task training.

However, it is unclear if the weights need to be computed again (needing all previous old task data) when subsequent new tasks are added. Moreover, this method reduces new task performance in the MNIST→ permuted-MNIST experiment, compared to the fine-tuning baseline. This phenomenon lies in line with the observations of our L2 soft-constraint baseline. Also notable is that the paper focuses on situations where the input and output forms of all tasks are the same, and no task-specific final layers are needed, which is slightly different from our multiple task scenario.

Cross-stitch Network [28] works on multi-task learning that resembles an advanced version of our network expansion experiment. It introduces the cross-stitch module, which takes two same-structured inputs, jointly learning two same-structured network blocks, as well as two pairs of weights to average their activations to obtain two same-structured outputs. By replacing the original network modules with the cross-stitch module, the authors were able to outperform joint training, among other methods, in most experiments. However, this method still needs the old task data to be present to jointly optimize, and it increases the network size proportionally to the number of tasks.

WA-CNN (Growing a Brain [29]) is most similar to our network expansion alternative design in Section 4.2. The paper increases layer sizes or network depth to improve fine-tuning performance on the new task, using a normalization in the activation scale of the new nodes. Although the model focuses on improving new task performance, when the method freezes all old parameters, it can maintain the performance of the old task while still outperforming traditional fine-tuning on the new task. However, the performance comes at the price of an increased network size – the number of parameters increases faster than LwF when aiming for a higher performance than fine-tuning. (See discussion in Section 4.2) But if a greater network size is not considered to be an issue, the method can be applied in parallel to ours.

3 Learning without Forgetting

Given a CNN with shared parameters θ_s and task-specific parameters θ_o (Fig. 2a), our goal is to add task-specific parameters θ_n for a new task and to learn parameters that work well on old and new tasks, using images and labels

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*
$$\theta_s^*,\ \theta_o^*,\ \theta_n^* \leftarrow \underset{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n}{\text{argmin}} \left(\lambda_o \mathcal{L}_{old}(Y_o, \hat{Y}_o) + \mathcal{L}_{new}(Y_n, \hat{Y}_n) + \mathcal{R}(\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n)\right)$$

Fig. 3. Procedure for Learning without Forgetting.

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. 2f.

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 (Section 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 (warm-up step). Then, we jointly train all weights θ_s, θ_o, and θ_n until convergence (joint-optimize step). The warm-up step greatly enhances fine-tuning's old-task performance, but is not so crucial to either our method or the compared Less Forgetting Learning (see Table 2 b). We still adopt this technique in Learning without Forgetting (as well as most compared methods) for the slight enhancement and a fair comparison.

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 [3], [30] 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. [11] 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

$$\mathcal{L}_{old}(\mathbf{y}_o, \hat{\mathbf{y}}_o) = -H(\mathbf{y}'_o, \hat{\mathbf{y}}'_o) \tag{2}$$

$$= -\sum_{i=1}^{l} y_o'^{(i)} \log \hat{y}_o'^{(i)}, \tag{3}$$

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

$$y_o'^{(i)} = \frac{(y_o^{(i)})^{1/T}}{\sum_j (y_o^{(j)})^{1/T}}, \quad \hat{y}_o'^{(i)} = \frac{(\hat{y}_o^{(i)})^{1/T}}{\sum_j (\hat{y}_o^{(j)})^{1/T}}. \tag{4}$$

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. [11] 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 authors' recommendations. In our experiments, we find that most reasonable losses would lead to similar performance, and the use of knowledge distillation loss leads to a marginal boost (Figs. 7c, d). Therefore, it is important to constrain outputs for original tasks to be similar to the original network, but the similarity measure is not very crucial.

λ_o is a loss balance weight, set to 1 for most our experiments. Making λ larger will favor the old task performance over the new task's, so we can obtain a old-task-new-task performance line by changing λ_o. (Fig. 7)

Relationship to joint training. As mentioned before, the main difference between joint training and our method is the need for the old dataset. Joint training uses the old task's images and labels in training, while Learning without Forgetting no longer uses them, and instead uses the new task images X_n and the recorded responses Y_o as substitutes. This eliminates the need to require and store the old dataset, brings us the benefit of joint optimization of the shared θ_s, and also saves computation since the images X_n only has to pass through the shared layers once for both the new task and the old task. However, the distribution of images from these tasks may be very different, and this substitution may potentially decrease performance. Therefore, joint training's performance may be seen as an upper-bound for our method.

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, where m is the total number of tasks.

3.1 Implementation Details

We use MatConvNet [31] to train our networks using stochastic gradient descent with momentum of 0.9 and dropout enabled in the fully connected layers. The data normalization (mean subtraction) 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 with offset on a 5×5 grid, randomly mirroring the crop, and adding variance to the RGB values like in AlexNet [3]. This data augmentation is applied to feature extraction too.

When training networks, we follow the standard practices for fine-tuning existing networks. The selection of hyperparameters, mainly the number of epochs, warm-up peroid, and the learning rate schedule, are chosen using the new task performance on a held-out set, which is a 20 percent subset of the training set whenever (1) the dataset does not have a validation set, or (2) the validation set is used for testing. When testing on VOC test set, the official validation set is used for hyperparameter selection. We do not look at the old task performance during this selection.

For random initialization of θ_n, we use Xavier [32] initialization. We use a learning rate much smaller than when training the original network ($0.1 \sim 0.02$ times the original rate). The learning rates are selected to maximize new task performance with a reasonable number of epochs. 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.

We choose the number of epochs for both the warm-up step and the joint-optimize step based on validation on the held-out set. Since we look at only the new task performance during validation, our hyperparameters favor the new task more. The compared methods converge at similar speeds, so we used the same number of epochs for each method for fair comparison; however, the convergence speed heavily depend on the original network and the task pair, and we validate for the number of epoch separately for each scenario. We lower the learning rate once by $10\times$ at the epoch when the held out accuracy plateaus. Exception to all these are ImageNet→ Scene with feature extraction where we observe overfitting and have to shorten the training, and Places365→ MNIST with feature extraction where convergence is slower and we double the number of epochs. We perform stricter validation than in our previous work [10], and the number of epochs is generally longer for each scenario.

To make a fair comparison, the intermediate network trained using our method (after the warm-up step) is used as a starting point for joint training and Fine Tuning, since this may speed up training convergence. In other words, for each run of our experiment, we first freeze θ_s, θ_o and train θ_n, and use the resulting parameters to initialize our method, joint training and fine-tuning. Feature extraction is trained separately because does not share the same network structure as our method.

For the feature extraction baseline, instead of extracting features at the last hidden layer of the original network (at the top of θ_s), we freeze the shared parameters θ_s, disable the dropout layers, and add a two-layer network with 4096 nodes in the hidden layer on top of it. This has the same effect of training a 2-layer network on the extracted features, but with data augmentation.

For joint training, loss for one task's output nodes is applied to only its own training images. We interleave batches of different tasks for gradient descent. For implementation convenience, we consider the epoch length of all tasks to be one iteration of the new task dataset. Note that the new dataset is typically much smaller than the old one, therefore in one epoch of the new dataset, not all training data in the old task will be seen by the network. We simply shuffle both datasets at the end of each epoch, so that every old task training sample will potentially appear in subsequent epochs. Another strategy can be maintaining separate epoch iterator for each dataset and shuffle them at the end of their own epoch, but we believe that these two strategies would perform very similarly.

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 common approaches of *feature extraction*, *fine-tuning*, and *fine-tuning FC*, and also *Less Forgetting Learning* (LFL) [9]. These methods 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 old task 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* [4] and the *Places365-standard* [33] dataset. Note that our previous work used Places2, a taster challenge in ILSVRC 2015 [4] and an earlier version of Places365, but the dataset was deprecated after our publication. ImageNet has 1,000 object category classes and more than 1,000,000 training images. Places365 has 365 scene classes and $\sim 1,600,000$ 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* [34] ("VOC"), *Caltech-UCSD*

TABLE 1
Performance for the Single New Task Scenario

(a) Using AlexNet structure (validation performance for ImageNet/Places365/VOC)

	ImageNet→VOC		ImageNet→CUB		ImageNet→Scenes		ImageNet→MNIST		Places365→VOC		Places365→CUB		Places365→Scenes		Places365→MNIST	
	old	new	old	new	old	new	old	new	old	new	old	new	old	new	old	new
LwF (ours)	56.2	76.1	54.7	57.7	55.9	64.5	49.8	99.3	50.6	70.2	47.9	34.8	50.9	75.2	38.3	99.2
Fine-tuning	-0.9	-0.3	-3.8	-0.7	-2.0	-0.8	-2.8	0.0	-2.2	0.1	-4.6	1.0	-2.1	-1.7	-0.9	0.1
LFL	0.0	-0.4	-1.9	-2.6	-0.3	-0.9	-2.9	-0.6	0.2	-0.7	0.7	-1.7	-0.2	-0.5	-0.4	-0.1
Fine-tune fc	0.5	-0.7	0.2	-3.9	0.6	-2.1	7.0	-0.2	0.5	-1.3	1.8	-4.9	0.3	-1.1	13.0	-0.2
Feat. Extraction	0.8	-0.5	2.3	-5.2	1.2	-3.3	7.3	-0.8	1.1	-1.4	3.8	-12.3	0.8	-1.7	13.3	-1.1
Joint Training	0.7	-0.2	0.6	-1.1	0.5	-0.6	7.2	-0.0	0.7	-0.0	2.3	1.5	0.3	-0.3	13.4	-0.1

(b) Test set performance

	Places365→VOC	
	old	new
LwF (ours)	50.6	73.7
Fine-tuning	-2.1	0.1
Feat. Extraction	1.3	-2.3
Joint Training	0.9	-0.1

(c) Using VGG structure

	ImageNet→CUB		ImageNet→Scenes	
	old	new	old	new
LwF (ours)	60.6	72.5	66.8	74.9
Fine-tuning	-9.9	0.6	-4.1	-0.3
LFL	0.3	-2.8	-0.0	-2.1
Fine-tune fc	3.2	-6.7	1.4	-2.4
Feat. Extraction	8.2	-8.6	1.9	-5.1
Joint Training	8.0	2.5	4.1	1.5

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 in most scenarios, 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 slightly worse performance than joint training. An exception is the ImageNet-MNIST task where LwF does not perform well on the old task.*

Birds-200-2011 fine-grained classification [35] ("CUB"), and *MIT indoor scene classification* [36] ("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 Places365. 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 [37] 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 [3] network structure because it is fast to train and well-studied by the community [6], [13], [15]. We also verify that similar results hold using 16-layer VGGnet [30] on a smaller set of experiments. For both network structures, the final layer (`fc8`) is treated as task-specific, and the rest are shared (θ_s) unless otherwise specified. The original networks pre-trained on ImageNet and Places365-standard are obtained from public online sources.

TABLE 2
Performance of Our Method Versus Various Alternative Design Choices

(a) Changing the number of task-specific layers, using network expansion, or attempting to lower θ_s's learning rate when fine-tuning.

	ImageNet→CUB		ImageNet→Scenes		Places365→VOC	
	old	new	old	new	old	new
LwF at output layer (ours)	54.7	57.7	55.9	64.5	50.6	70.2
last hidden layer	54.7	56.2	55.7	65.0	50.7	70.6
2^{nd} last hidden (Fig. 6(a))	54.6	57.1	55.8	64.2	50.8	70.5
network expansion	57.0	54.0	57.0	62.5	51.7	67.1
network expansion + LwF	54.4	57.0	55.7	63.9	50.7	70.4
fine-tuning (10% θ_s learning rate)	52.2	54.9	54.8	62.7	49.3	69.5

(b) Performing LwF and fine-tuning with and without warmup. The warmup step is not crucial for LwF, but is essential for fine-tuning's old task performance.

	ImageNet→Scenes		Places365→VOC	
	old	new	old	new
LwF	55.9	64.5	50.6	70.2
fine-tuning	53.9	63.8	48.4	70.3
LFL	55.5	63.6	50.8	69.5
LwF (no warm-up)	55.2	64.9	50.4	70.0
fine-tuning (no warm-up)	49.8	63.9	42.3	70.0
LFL (no warm-up)	55.4	63.0	50.6	69.1

In most cases, these alternative choices do not provide consistent advantage or disadvantage compared to our method.

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 Places365, 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. Due to the randomness within CNN training, we run our experiments three times, and report the mean performance.

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. Note that the results have multiple sources of variance, including random initialization and training, pre-determined termination (performance can fluctuate by training 1 or 2 additional epochs), etc.

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. Tables 1 a, 1 b shows the performance of our method, and the relative performance of other methods compared to it using AlexNet. We also visualize the old-new performance comparison on two of the task pairs in Fig. 7. We make the following observations:

On the new task, our method consistently outperforms LFL, fine-tuning FC, and feature extraction, while outperforming fine-tuning on most task pairs except Places365→CUB, Places365→VOC, Places365→MNIST (similar performance), and ImageNet→MNIST (worse performance). 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 underperforms feature extraction, fine-tuning FC, and occasionally LFL. 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 an old task similar to the new one, the performance loss is greatly reduced.

Considering both tasks, Fig. 7 shows that if λ_o is adjusted, LwF can perform better than LFL and fine-tuning FC on the new task for the same old task performance on the first task pair, and perform similarly to LFL on the second. Indeed, fine-tuning FC gives a performance between fine-tuning and feature extraction. LwF provides freedom of changing the shared representation compared to LFL, which may have boosted the new task performance.

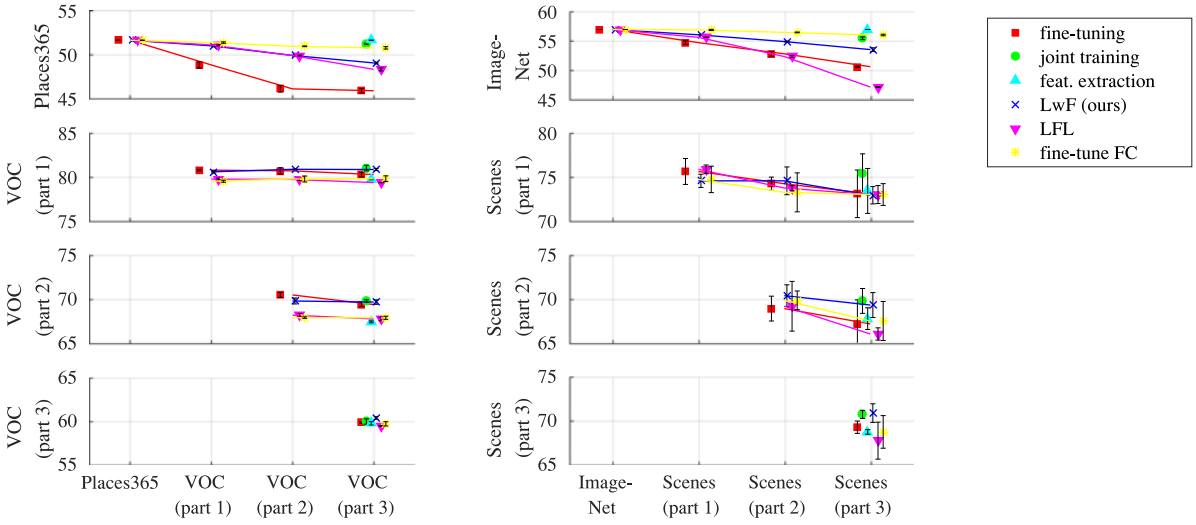
Our method usually performs similarly to joint training with AlexNet. Our method tends to slightly outperform joint training on the new task but underperform on the old task, which we attribute to a different distribution in the two task datasets. Overall, the methods perform similarly (except on the extreme *→MNIST cases), a positive result since our method does not require access to the old task training data and is faster to train. Note that sometimes both tasks' performance degrade with λ_o too large or too small. We suspect that making it too large essentially increases the old task learning rate, potentially making it suboptimal, and making it too small lessens the regularization.

Dissimilar new tasks degrade old task performance more. For example, CUB is very dissimilar task from Places365 [13], and adapting the network to CUB leads to a Places365 accuracy loss of 8.4% ($3.8\% + 4.6\%$) for fine-tuning, 3.8 percent for LwF, and 1.5% ($3.8\% - 2.3\%$) 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 Places365 images is very different. Even joint training leads to more accuracy loss on the old task because it cannot find a set of shared parameters that works well for both tasks. Our method does not outperform fine-tuning for Places365→CUB and, as expected, *→MNIST on the new task, since the hand-written characters provide poor indirect supervision for the old task. The old task accuracy drops substantially with fine-tuning and LwF, though more with fine-tuning.

Similar observations hold for both VGG and AlexNet structures, except that joint training outperforms consistently for VGG, and LwF performs worse than before on the old task. (Table 1) This indicates 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. Among these results, LFL diverges using stochastic gradient descent, so we tuned down the learning rate ($0.5\times$) and used $\lambda_i = 0.2$ instead.

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 Places365, 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. (See supplemental material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPAMI.2017.2773081) 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.

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. Fig. 4 shows the results on both dataset pairs. Our findings are usually consistent with the single new task experiment: *LwF outperforms fine-tuning, feature extraction, LFL, and fine-tuning FC for*



(a) Places365→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 in most scenarios. For Places2→VOC, our method performs comparably to joint training.

most newly added tasks. However, LwF performs similarly to joint training only on newly added tasks (except for Scenes part 1), and underperforms joint training on the old task after more tasks are added.

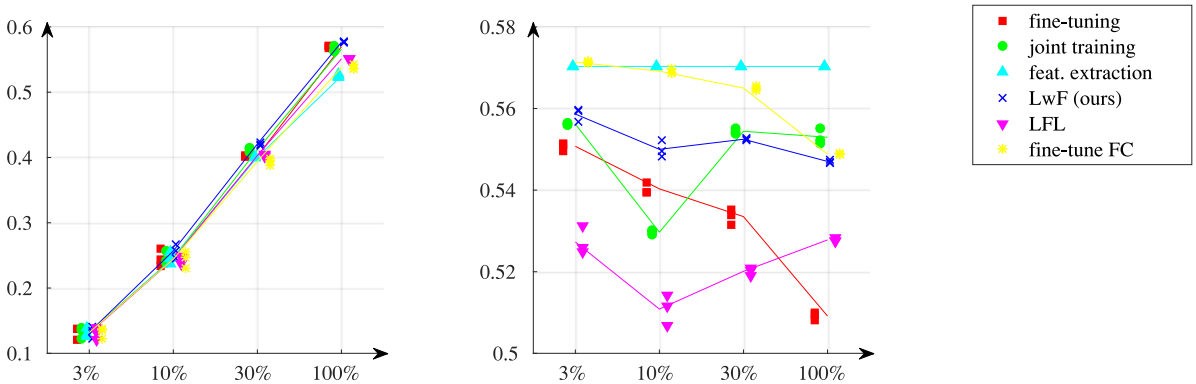
Influence of dataset size. We inspect whether the size of the new task dataset affects our performance relative to other methods. We perform this experiment on adding CUB to ImageNet AlexNet. We subsample the CUB dataset to 30, 10 and 3 percent when training the network, and report the result on the entire validation set. Note that for joint training, since we interleave batches from both datasets, only a small percentage of the ImageNet training set is iterated over for one CUB epoch. (See the end of Section 3.1) Our results are shown in Fig. 5. Results show that *the same observations hold. Our method outperforms fine-tuning on both tasks. Differences between methods tend to increase with more data used, although the correlation is not definitive.*

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 Fig. 6a) 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 [13]. 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.

Fig. 6b 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



(a) CUB accuracy (new) (b) ImageNet 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 compared methods on both the old task and the new task decrease with less data, but the observations remain the same. LwF outperforms fine-tuning despite the change in training set size.

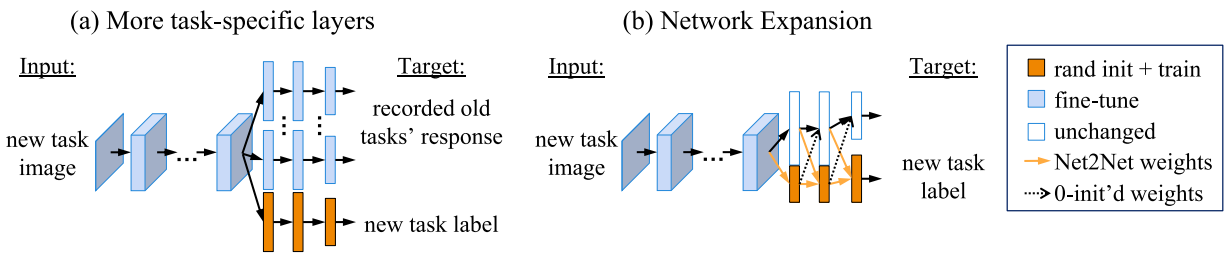


Fig. 6. Illustration for alternative network modification methods. In (a), more fully connected layers are task-specific, rather than shared. In (b), nodes for multiple old tasks (not shown) are connected in the same way. LwF can also be applied to Network Expansion by unfreezing all nodes and matching output responses on the old tasks.

initialized the same way Net2Net [20] 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"), similar to progressive network [18]; or train using Learning without Forgetting ("network expansion + LwF"). Note that both methods needs the network to scale quadratically with respect to the number of new tasks.

Table 2 a shows the comparison with our original method. *Network expansion by itself performs better than feature extraction, but not as well as LwF on the new task. Network Expansion + LwF performs similarly to LwF* with additional computational cost and complexity.

We note that Growing a Brain [29] offers a more thorough experiment of network expansion (w/o LwF). By freezing all old parameters, adding a novel normalization step, and experimenting on the layer for adding nodes and the number of nodes to add, their method WA-CNN is able to outperform fine-tuning, but still at the cost of an increased network size. For example, on the SUN-397 dataset [38], the network outperforms fine-tuning by 0.53 percent while maintaining the original old task performance by adding 1024 nodes to the 4096-node `fc7`; 0.88 percent if 2048 nodes are added. Adding the 2048 nodes and the new top layer increases the number of parameters by 21.1 percent, while LwF increases network size by 2.7 percent.

We also note that the WA-CNN variant that does not freeze parameters performs better on the new task. In a way, the variant that freezes old parameters suffer from the

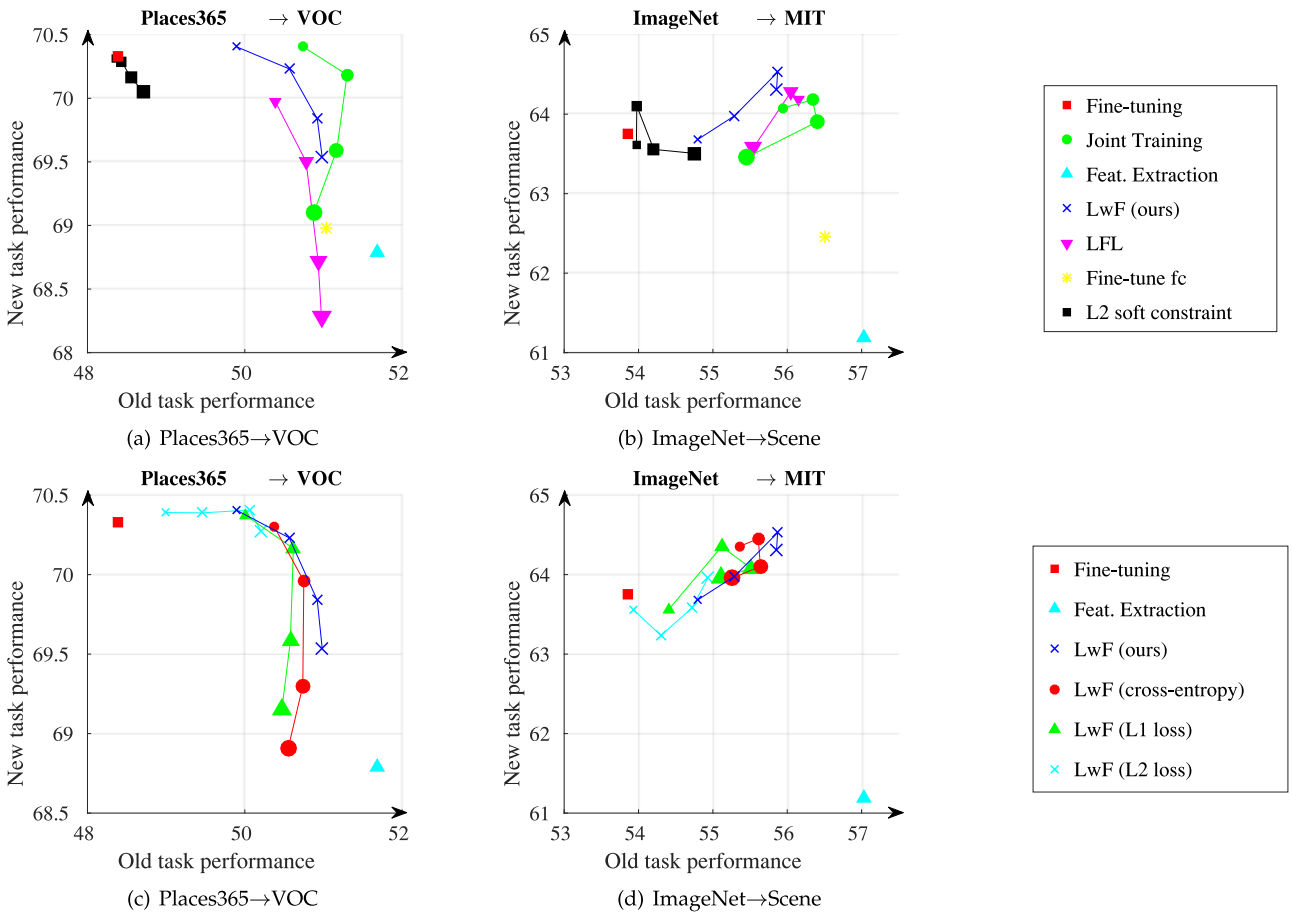
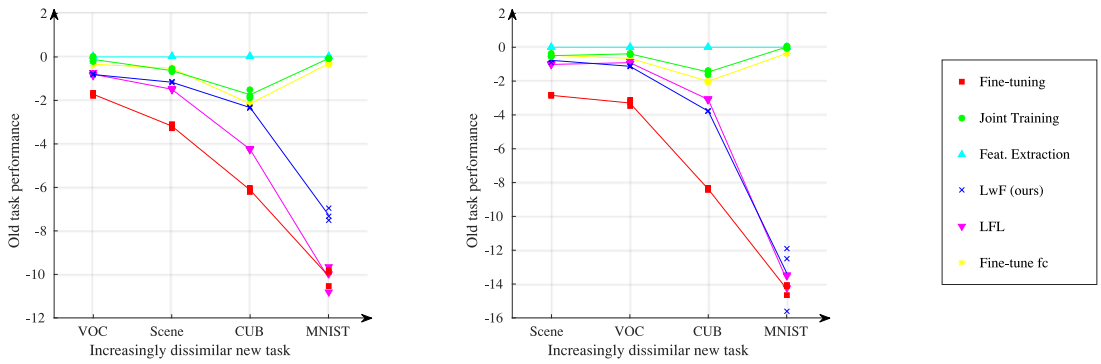


Fig. 7. Visualization of both new and old task performance for compared methods, some with different weights of losses. (a)(b): comparing methods; (c)(d): comparing losses. Larger symbols signifies larger λ_o i.e., heavier weight towards response-preserving loss.



(a) ImageNet as old task (b) Places365 as old task

Fig. 8. Influence of new-old task similarity on old task performance preservation, related to the original old task performance. As the tasks becomes further irrelevant, the old task preservation drops.

same drawback as feature extraction. Potentially the freezing variant can be improved by unfreezing and applying LwF, which can potentially increase new task performance while only slightly sacrificing the old task performance. However, this experiment is outside the scope of this paper.

Effect of lower learning rate of shared parameters. We investigate whether simply lowering the learning rate of the shared parameters θ_s would preserve the original task performance. The result is shown in Table 2 a. A reduced 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.*

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 $\frac{1}{2}\lambda_o\|w - w_0\|^2$ to the loss for fine-tuning, where w and w_0 are flattened vectors of all shared parameters θ_s and their original values. We change the coefficient λ_o and observe its effect on the performance. λ_o is set to 0.15, 0.5, 1.5, 2.5 for Places365→VOC, and 0.005, 0.015, 0.05, 0.15, 0.25 for ImageNet→Scene.

As shown in Fig. 7, *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 the same task pairs as before. Fig. 7 shows our results. *Results indicate our knowledge distillation loss slightly outperforms compared losses, although the advantage is not large.*

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. We show the effectiveness of our method on a number of classification tasks.

5.1 Limitations

First, it is worth pointing out that LwF operates on distinct tasks. Like many multitask learning methods, it cannot properly deal with domains that are continually changing on a spectrum (e.g., old task being classification from top-down view, and new task being classification from views of unknown angles); the tasks must be enumerated. In addition, LwF requires each sample to be accompanied by the information of which task it belongs to, and this information is needed for both training and testing.

Second, in contrast to methods such as Never Ending Learning [25], LwF requires all new task training data to be present before computing their old task responses. This would not be applicable if the data come in a stream and the model is required to be trained incrementally.

Third, the ability of LwF to incrementally learn new tasks is limited, as the performance of old tasks gradually drop. Fourth, the gap between LwF and joint training performance on both tasks are larger when experimented on VGG structure.

Finally, as observed in Section 4.1, the performance of LwF largely depends on how much the new task data resembles the old task's. For example, when the old task is ImageNet classification, we have new tasks ranging from PASCAL VOC multilabel classification (very similar), to MIT indoor scenes (somewhat similar, since scene classification can rely on the presence of certain object categories), to CUB (not similar, since CUB only have pictures of birds, mostly in nature scenes, lacking most ImageNet objects), to MNIST (no resemblance at all). As shown in Table 1a, and visualized in Fig. 8, the old task preservation compared to original performance (Feat. Extraction) is quite well for the former two, a little worse on the quite dissimilar CUB, and bad on the irrelevant MNIST. The same trend emerges with Places365 as the old task, where Places365→CUB old task preservation is perhaps less than satisfactory. We conjecture that LwF will be not very effective for task pairs that are more dissimilar than ImageNet and CUB.

5.2 Usage and Future Work

Our 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 for previous tasks. Second, if we care only about the performance for the new task, our method often 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 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.

As an additional use-case example, we investigate using LwF in the application of tracking in the Appendix, available in the online supplemental material. We build on MDNet [39], which views tracking as a template classification task. A classifier transferred from training videos is fine-tuned online to classify regions as the object or background. We propose to replace the fine-tuning step with Learning without Forgetting. We leave the details and implementation to the appendix, available in the online supplemental material. We observe some improvements by applying LwF, but the difference is not statistically significant.

We see several directions for future work. We have demonstrated the effectiveness of LwF for image classification and one experiment on tracking, but would like to further experiment on semantic segmentation, detection, and problems outside of computer 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 old task performance 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.

Acknowledgments

This work is supported in part by NSF Awards 14-46765 and 10-53768 and ONR MURI N000014-16-1-2007.

References

[1] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," *Psychology Learn. Motivation*, vol. 24, pp. 109–165, 1989.

[2] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," in *Proc. Int. Conf. Learn. Representations*, 2014.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Int. Conf. Adv. Neural Inform. Process. Syst.*, 2012, pp. 1097–1105.

[4] O. Russakovsky, et al., "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.

[5] J. Donahue, et al., "DeCAF: A deep convolutional activation feature for generic visual recognition," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. I-647–I-655.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 580–587.

[7] R. Caruana, "Multitask learning," *Mach. Learn.*, vol. 28, no. 1, pp. 41–75, 1997.

[8] T. Furlanello, J. Zhao, A. M. Saxe, L. Itti, and B. S. Tjan, "Active long term memory networks," *CoRR*, vol. abs/1606.02355, 2016, http://arxiv.org/abs/1606.02355.

[9] H. Jung, J. Ju, M. Jung, and J. Kim, "Less-forgetting learning in deep neural networks," *arXiv preprint arXiv:1607.00122*, 2016.

[10] Z. Li and D. Hoiem, "Learning without forgetting," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 614–629.

[11] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst. Workshop*, 2014, http://www.dlworkshop.org/accepted-papers

[12] A. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "CNN features off-the-shelf: An astounding baseline for recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, 2014, pp. 806–813.

[13] H. Azizpour, A. Razavian, J. Sullivan, A. Maki, and S. Carlsson, "Factors of transferability for a generic convnet representation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 9, pp. 1790–1802, Sep. 2016.

[14] P. Agrawal, R. Girshick, and J. Malik, "Analyzing the performance of multilayer neural networks for object recognition," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 329–344.

[15] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Proc. Int. Conf. Advances Neural Inform. Process. Syst.*, 2014, pp. 3320–3328.

[16] O. Chapelle, P. Shivaswamy, S. Vadrevu, K. Weinberger, Y. Zhang, and B. Tseng, "Boosted multi-task learning," *Mach. Learn.*, vol. 85, no. 1–2, pp. 149–173, 2011.

[17] A. V. Terekhov, G. Montone, and J. K. ORegan , "Knowledge transfer in deep block-modular neural networks," in *Biomimetic and Biohybrid Systems*. Berlin, Germany: Springer, 2015, pp. 268–279.

[18] A. A. Rusu, et al., "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[19] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "FitNets: Hints for thin deep nets," in *Proc. Int. Conf. Learn. Representations*, 2015, http://www.iclr.cc/doku.php?id=iclr2015:main

[20] T. Chen, I. Goodfellow, and J. Shlens, "Net2net: Accelerating learning via knowledge transfer," in *Proc. Int. Conf. Learn. Representations*, 2016, http://www.iclr.cc/doku.php?id=iclr2016:main

[21] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[22] M. Long Y. Cao, J. Wang, and M. Jordan, "Learning transferable features with deep adaptation networks," *Int. Conf. Mach. Learn.*, pp. 97–105, 2015.

[23] E. Tzeng, J. Hoffman, T. Darrell, and K. Saenko, "Simultaneous deep transfer across domains and tasks," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 4068–4076.

[24] S. Thrun, "Lifelong learning algorithms," in *Learning to Learn*. Berlin, Germany: Springer, 1998, pp. 181–209.

[25] T. Mitchell, et al., "Never-ending learning," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 2302–2310.

[26] E. Eaton and P. L. Ruvolo, "ELLA: An efficient lifelong learning algorithm," in *Proc. 30th Int. Conf. Mach. Learn.*, 2013, pp. 507–515.

[27] J. Kirkpatrick, et al., "Overcoming catastrophic forgetting in neural networks," *Proc. Nat. Acad. Sci.*, vol. 114, no. 13, pp. 3521–3526, 2017.

[28] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, "Cross-stitch networks for multi-task learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 3994–4003.

[29] Y. Wang, D. Ramanan, and M. Hebert, "Growing a brain: Fine-tuning by increasing model capacity," in *IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 3029–3038.

[30] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Representations*, 2015.

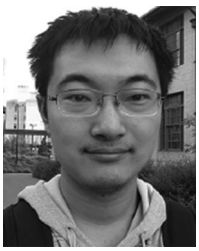
[31] A. Vedaldi and K. Lenc, "MatConvNet – convolutional neural networks for MATLAB," in *Proc. ACM Int. Conf. Multimedia*, 2015, pp. 689–692.

[32] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Aistats*, vol. 9, 2010, pp. 249–256.

[33] B. Zhou, A. Lapedriza, A. Khosla, and A. Oliva, A. Torralba, "Places: A 10 million image database for scene recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2017.

[34] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *Int. J. Comput. Vis.*, vol. 111, no. 1, pp. 98–136, Jan. 2015.

[35] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The Caltech-UCSD Birds-200–2011 Dataset," California Inst. of Technol., Pasadena, CA, USA, Tech. Rep. CNS-TR-2011-001, 2011.
[36] A. Quattoni and A. Torralba, "Recognizing indoor scenes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 413–420.
[37] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[38] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba, "Sun database: Large-scale scene recognition from abbey to zoo," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2010, pp. 3485–3492.
[39] H. Nam and B. Han, "Learning multi-domain convolutional neural networks for visual tracking," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 4293–4302.



Zhizhong Li received the BEng degree from the Department of Automation, Tsinghua University, completing his thesis with Changshui Zhang, and the MS degree from the Robotics Institute, Carnegie Mellon University, where he was supervised by Daniel Huber. He is working toward the second year Ph. D. degree in computer science with the University of Illinois, Urbana-Champaign, supervised by Derek Hoiem. His research interest include the computer vision, especially its intersection with machine learning. Most recently his research is focused on the application of transfer learning and deep learning in vision.

Derek Hoiem received the PhD degree in robotics from Carnegie Mellon University, and completed the postdoctoral fellowship degree from the Beckman Institute, in 2007 and 2008, respectively. He is an associate professor of computer science with the University of Illinois, Urbana-Champaign, since January 2009. His primary research goal is to model the physical and semantic structure of the world, so computers can better understand scenes from images. In particular, he researches algorithms to interpret physical space from images and to relate objects to their environment and to each other. Example applications include creating 3D models of scenes and objects from one image, photorealistic rendering of object models into images, robot navigation, and creating and matching as-built 3D models of construction scenes to planned models. He has published dozens of papers and several patents, and his work has been recognized with awards including an ACM Doctoral Dissertation Award honorable mention, CVPR best paper award, Intel Early Career Faculty award, Sloan Fellowship, and PAMI Significant Young Researcher award. He is also co-founder and CTO of Reconstruct, which visually documents construction sites, matching images to plans and analyzing productivity and risk for delay.