UNIVERSITY OF CALIFORNIA SAN DIEGO

Efficient Learning across Multiple Domains with Deep Neural Networks

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Yunhui Guo

2020

DEDICATION

To my family.

EPIGRAPH

What I cannot create, I do not understand.

— Richard P. Feynman

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First of all, I would like to thank my Ph.D. advisor Professor Tajana Simunic Rosing for her insightful guidance and continuous support during my doctoral studies. Without her help and support, this thesis could not have been accomplished. I've learned a lot from her throughout the years. I could not have imagined having a better advisor for my Ph.D. study.

I would like to thank my thesis committee members, Professor Kamalika Chaudhuri, Professor Tara Javidi, Professor Arun Kumar and Professor Julian McAuley. I sincerely acknowledge Professor Kamalika Chaudhuri, Professor Tara Javidi and Professor Arun Kumar for their guidance and inspiring advice during our short time of working together.

I would also like to thank all the friends and colleagues here at SeeLab and UCSD. A partial list includes Mohsen Imani, Yeseong Kim, Anthony Thomas, Minxuan Zhou, Michael Ostertag, Sahand Salamat, Rishikanth Chandrasekaran, Xiaofan Yu, Jaeyoung Kang. I would like to thank you for making my PhD study at UCSD a pleasant period of time to pass by. I would also like to thank my manager Dr. Rogerio Feris during my internship at Thomas J. Watson Research Center for his guidance and support.

Special thanks go to my parents and my grandparents for their continuous support throughout my study abroad.

Chapters 3 contains material from "SpotTune: Dynamic Transfer Learning via Adaptive Fine-tuning", by Yunhui Guo, Honghui Shi, Abhishek Kumar,Tajana Rosing, Kristen Grauman, Rogerio Feris, which appears in *IEEE Conference on Computer Vi-*

sion and Pattern Recognition (CVPR), 2019. The dissertation author was the primary investigator and author of this paper.

Chapters 4 contains material from "AdaFilter: Adaptive Filter Fine-tuning for Deep Transfer Learning", by Yunhui Guo, Yandong Li, Liqiang Wang, Tajana Rosing, which appears in *The 34th AAAI Conference on Artificial Intelligence (AAAI)*, 2020. The dissertation author was the primary investigator and author of this paper.

Chapters 5 contains material from "A Broader Study of Cross-Domain Few-Shot Learning", by Yunhui Guo, Noel C. F. Codella, Leonid Karlinsky, James V. Codella, John R. Smith, Kate Saenko, Tajana Rosing, Rogerio Feris, which appears in *The 16th European Conference on Computer Vision (ECCV)*, 2020. The dissertation author was the primary investigator and author of this paper.

Chapters 6 contains material from "Depthwise Convolution is All You Need for Learning Multiple Visual Domains", by Yunhui Guo, Yandong Li, Liqiang Wang, Tajana Rosing, which appears in *The 33rd AAAI Conference on Artificial Intelligence (AAAI)*, 2019. The dissertation author was the primary investigator and author of this paper.

Chapters 7 contains material from "Active Sampling for Distributed Training in Heterogeneous Sensor Networks", by Yunhui Guo, Xiaofan Yu, Kamalika Chaudhuri, Tajana Rosing, which appears in *The 16th International Conference on Mobility, Sensing and Networking (MSN)*, 2020. The dissertation author was the primary investigator and author of this paper.

Chapters 8 contains material from "Improved Schemes for Episodic Memory based Lifelong Learning Algorithm", by Yunhui Guo, Mingrui Liu, Tianbao Yang, and Tajana

Rosing, which appears in The *34th Conference on Neural Information Processing Systems (NeurIPS)*, 2020. The dissertation author was the primary investigator and author of this paper.

VITA

2014	B. S. in Communication Engineering, University of Electronic Science and Technology of China, Chengdu, China
2017	M. S. in Computer Science, Zhejiang University, Hangzhou, China
2020	Ph. D. in Computer Science, University of California, San Diego, USA

PUBLICATIONS

Yunhui Guo, Mohsen Imani, Jaeyoung Kang, Sahand Salamat, Justin Morris, Baris Aksanli, Yeseong Kim, Tajana Rosing , "HyperRec: Efficient Recommender Systems with Hyperdimensional Computing", The 26th Asia and South Pacific Design Automation Conference (ASP-DAC), 2020.

Yunhui Guo, Noel C. F. Codella, Leonid Karlinsky, James V. Codella, John R. Smith, Kate Saenko, Tajana Rosing, Rogerio Feris, "A Broader Study of Cross-Domain Few-Shot Learning", The 16th European Conference on Computer Vision (ECCV), 2020.

Yunhui Guo, Mingrui Liu, Tianbao Yang, and Tajana Rosing, "Improved Schemes for Episodic Memory based Lifelong Learning Algorithm", The 34th Conference on Neural Information Processing Systems (NeurIPS), 2020

Yunhui Guo, Xiaofan Yu, Kamalika Chaudhuri, Tajana Rosing, "Active Sampling for Distributed Training in Heterogeneous Sensor Networks", The 16th International Conference on Mobility, Sensing and Networking (MSN), 2020.

Minxuan Zhou, Guoyang Chen, Weifeng Zhang, Yunhui Guo, Mohsen Imani, and Tajana Rosing. "Falcon: Boosting DNN Acceleration in Computing Memories via Data Layout Optimizations", Submitted to MICRO 2020

Yunhui Guo, Yandong Li, Liqiang Wang, Tajana Rosing, "AdaFilter: Adaptive Filter Fine-tuning for Deep Transfer Learning", *The 34th AAAI Conference on Artificial Intelligence (AAAI)*, 2020

Yunhui Guo, Honghui Shi, Abhishek Kumar,Tajana Rosing, Kristen Grauman, Rogerio Feris, "SpotTune: Dynamic Transfer Learning via Adaptive Fine-tuning", *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019

Yunhui Guo, Yandong Li, Liqiang Wang, Tajana Rosing, "Depthwise Convolution is All You Need for Learning Multiple Visual Domains", *The 33rd AAAI Conference on Artificial Intelligence (AAAI)*, 2019

Rishikanth Chandrasekaran, Yunhui Guo, Anthony Thomas, Massimiliano Menarini, Michael H Ostertag, Yeseong Kim, Tajana Rosing, “Efficient Sparse Processing in Smart Home Applications”, *Proceedings of the 1st Workshop on Machine Learning on Edge in Sensor Systems*, 2019

Anthony Thomas, Yunhui Guo, Yeseong Kim, Baris Aksanli, Arun Kumar, Tajana Rosing, “Hierarchy-Aware Machine Learning Inference for Networked Sensing Applications”, *16th IEEE International Conference on Networking, Sensing and Control*, 2019

Yunhui Guo, Congfu Xu, Hanzhang Song, Xin Wang, “Understanding Users’ Budget Constraints for Recommendation with Hierarchical Poisson Factorization”, *In Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI)*, 2017

Hanzhang Song, Yunhui Guo, Congfu Xu , “ Incorporating Collaborative Ranking Algorithm with Weighted Recursive Autoencoder for Item Recommendation ”, *The AAAI-17 Workshop on Crowdsourcing, Deep Learning, and Artificial Intelligence Agents*, 2017

Xin Wang, CongFu Xu, Yunhui Guo, Hui Qian, “ Constrained Preference Embedding for Item Recommendation ”, *In Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI)*, 2016

Yunhui Guo, Xin Wang, Congfu Xu, “LBMF: Log-Bilinear Matrix Factorization for Recommender Systems”, *In Advances in Knowledge Discovery and Data Mining - 20th Pacific-Asia Conference (PAKDD)*, 2016

Congfu Xu, Xin Wang, Yunhui Guo, “Collaborative Expert Recommendation for Community-Based Question Answering”, *In Machine Learning and Knowledge Discovery in Databases - European Conference (ECML/PKDD)*, 2016

Yunhui Guo, Xin Wang, Congfu Xu, “CroRank: Cross Domain Personalized Transfer Ranking for Collaborative Filtering”, *In International Conference on Data Mining Workshop (ICDMW)*, 2015

Xin Wang, Yunhui Guo, Congfu Xu, “Recommendation Algorithms for Optimizing Hit Rate, User Satisfaction and Website Revenue”, *In Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*, 2015

ABSTRACT OF THE DISSERTATION

Efficient Learning across Multiple Domains with Deep Neural Networks

by

Yunhui Guo

Doctor of Philosophy in Computer Science

University of California San Diego, 2020

Professor Tajana Simunic Rosing, Chair

Learning with data from multiple domains is a longstanding topic in machine learning research. In recent years, deep neural networks (DNN) have shown remarkable performance on different machine learning tasks. However, how to efficiently utilize deep neural networks for learning with multiple domains is largely unexploited. A model aware of the relationships between different domains can be trained to work on new domains with fewer resources and achieve better performance. However, to identify and leverage the transferable structure is challenging.

In this dissertation, we propose novel methods which allow efficient learning across multiple domains in several different scenarios. First, we address the problem of learning across two image domains with deep neural networks. We propose two adaptive methods which allow different images to fine-tune and reuse different residual blocks and convolutional filters of the pre-trained model. Experimental results show that the proposed *SpotTune* outperforms the standard fine-tuning on 12 out of 14 datasets. Second, we consider the case that the target domain only has few examples per category which is referred to as the cross-domain few-shot problem. We establish a new benchmark for cross-domain few-shot learning and propose a multi-model selection algorithm which achieves an average improvement of 2% compared with the state-of-the-art approach on the proposed benchmark. Third, we consider learning with multiple domains simultaneously. We propose a multi-domain learning method based on depthwise separable convolution which achieves the highest score on the Visual Decathlon Challenge and reduces the number of parameters by 50% compared with the state-of-the-art approach. We further propose an efficient multi-domain learning method for distributed training in sensor networks. The proposed method can reduce the communication cost by up to 53% and energy consumption by up to 67% without accuracy degradation compared with conventional approaches. Finally, we address the problem of learning with multiple domains sequentially. We propose an algorithm called *mixed stochastic gradient descent* (MEGA) which allows the model to maintain the performance on old domains while being trained on a new domain. MEGA achieves an average accuracy of 91.21±0.10% on **Permuted MNIST**, which is 2% better than the previous state-of-the-art model. On **Split CIFAR**, the proposed MEGA achieves an average accuracy of 66.12±1.93%, which is about 5% better than the state-of-the-art method.

Chapter 1

Introduction

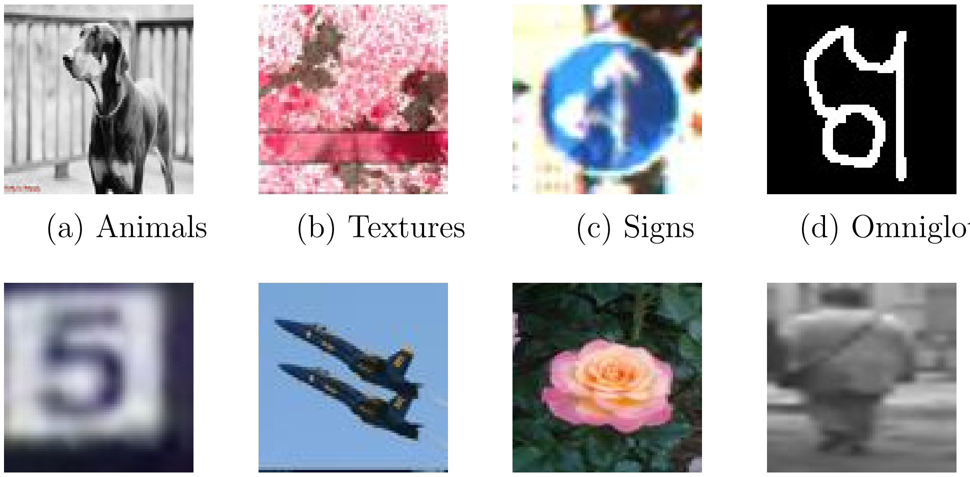
Deep learning has shown remarkable success in many computer vision tasks, but current methods often rely on the assumption that the training data and the test data are from the same domain [KSH12, HZRS16, HLvdMW17]. However, in practical applications, we often need to train models on a source domain which is different from the test domain. There are several reasons for this practice. First, collecting sufficient amounts of data for certain classes may be impossible in practice: for example, in dermatology, there are a multitude of instances of rare diseases, or diseases that become rare for particular types of skin [RHDC19, AS18, KOC$^+$]. Second, although the source domain and the target domain are different, by leveraging the knowledge in the source domain, it is possible to boost the performance or accelerate the learning on the target domain [PY$^+$10]. We introduce several different scenarios when it comes to realize the idea of learning across multiple domains with deep neural networks.

Learning across two domains

When there are two domains, *transfer learning*, where the goal is to transfer knowledge from a related *source domain*, is commonly used to compensate for the lack of sufficient training data in the *target domain* [PY$^+$10, Ben12]. Transfer learning allows learning across domains to overcome the domain shift between the training set and the test set. *Fine-tuning* is arguably the most widely used approach for transfer learning when working with deep learning models. It starts with a pre-trained model on the source task and trains it further on the target task. For computer vision tasks, it is a common practice to work with ImageNet pre-trained models for fine-tuning [KSL18]. Compared with training from scratch, fine-tuning a pre-trained convolutional neural network on a target dataset can significantly improve performance, while reducing the target labeled data requirements [GDDM14, YCBL14, TSG$^+$16, KSL18].

Few-shot cross-domain learning

Another typical case in learning across two domains is that the target domain only has few examples per category. Training deep neural networks for visual recognition typically requires a large amount of labelled examples [KSH12]. The generalization ability of deep neural networks relies heavily on the size and variations of the dataset used for training. However, collecting sufficient amounts of data for certain classes may be impossible in practice. In domains such as satellite imagery, there are instances of rare categories such as airplane wreckage, which poses severe challenges for deep neural networks to learn useful features for discrimination.



(a) Animals (b) Textures (c) Signs (d) Omniglot

(e) Digits (f) Aircraft (g) Flowers (h) Pedestrian

Figure 1.1: Image examples from different domains.

In contrast to deep neural networks, humans recognize new categories from a few examples in certain circumstances, such as when categories exhibit predictable variations across examples and have reasonable contrast from background in the image [LST15, LSGT11]. However, even humans have trouble recognizing new categories that vary too greatly between examples or differ from prior experience, such as for diagnosis in dermatology, radiology, or other fields [RHDC19]. Because there are many applications where learning must work from few examples, and both machines and humans have difficulty learning in these circumstances, finding new methods to tackle the problem remains a challenging but desirable goal.

Learning across multiple domains simultaneously

The existing deep neural network models are powerful but mostly designed for dealing with images from a specific visual domain (*e.g.* digits, animals, or flowers). This limits the applications of current approaches, as each time the network needs to be retrained when new domains arrive. In sharp contrast to such deep neural network models, humans can easily generalize to new domains based on the acquired knowledge [CG15, HTYN$^+$15]. Previous works [BV17, RBV18a] show that images from different domains may have a universal structure that can be captured via a common parameterization. A natural question then arises:

Can we build a single neural network that can deal with images across different domains simultaneously?

The question motivates the field called multi-domain learning, where we target designing a common feature extractor that can capture the universal structure in different domains and reducing the overhead of adding new tasks to the model. With multi-domain learning, the visual models are vested with the ability to work well on different domains with minimal or no domain-specific parameters. Ideally, a single model should achieve competitive performance across a variety of image domains as shown in Figure 1.1. This is especially important if we want to deploy the trained deep neural networks on small devices which have limited resources. If we train separate models for different domains, we need to store a large number of models which are beyond the capability of the small devices.

Learning across multiple domains sequentially

A significant step towards artificial general intelligence (AGI) is to enable the learning agent to acquire the ability of remembering past experiences while being trained on a continuum of tasks. Current deep neural networks are capable of achieving remarkable performance on a single task [GBCB16]. However when the network is retrained on a new task, its performance drops drastically on previously trained tasks, a phenomenon which is referred to as *catastrophic forgetting* [Rat90, Rob95, Fre99, KPR+17]. In stark contrast, human cognitive system is capable of acquiring new knowledge without damaging previously learned experiences. It is thus of great importance to develop algorithms to allow deep neural networks to achieve continual learning capability (i.e., avoiding catastrophic forgetting) across multiple domains.

The problem of catastrophic forgetting motivates the field called lifelong learning [KPR+17, PKP+19, TM95]. A central dilemma in lifelong learning is how to achieve a balance between the performance on old tasks and the new task [KPR+17, Rob95]. During the process of learning the new task, the originally learned knowledge will typically be disrupted, which leads to catastrophic forgetting. On the other hand, a learning algorithm biasing towards old tasks will interfere with the learning of the new task. Several lines of methods are proposed recently to address this issue. Examples include regularization based methods [KPR+17, ZPG17], knowledge transfer based methods [RRD+16], episodic memory based methods [LP+17, CRRE18, RCA+18]. Especially, episodic memory based methods such as GEM [LP+17] and A-GEM [CRRE18] have shown remarkable performance. In episodic memory based methods, a small episodic memory is used for storing examples from old tasks to guide the optimization of the current task.

Our contributions

In this dissertation, we consider learning with multiple domains with deep neural networks in several different scenarios. We first consider the setting that there are one source domain and one target domain. We pre-train the model on the source domain and propose adaptive fine-tuning methods for transferring knowledge to the target domain. Next, we consider a more challenging case that the target domain only has few examples. We establish a new benchmark and propose the *Incremental Multi-model Selection* method to enable the target domain to reuse knowledge from multiple source domains.

We then consider learning with multiple domains simultaneously. We propose *SharingNet* which can classify images from multiple different image domains. We generalize this idea to sensor networks involving different types of sensor data and propose active learning methods which allows efficient distributed training. Finally, we tackle the situation of learning with multiple domains sequentially and propose the *Mixed Stochastic Gradient* algorithm to allow the model to learn new domains while maintaining performance on old domains.

Learning across two domains

Adaptive transfer learning: In Chapter 4, we propose two adaptive fine-tuning methods, *SpotTune* and *AdaFilter*, to learn a decision policy for input-dependent transfer learning. As shown in Figure 1.2, we propose novel methods that decides, per training example, which layers or filters of the pre-trained model should have their parameters fixed, i.e., shared with the source task, and which layers or filters should be fine-tuned to improve

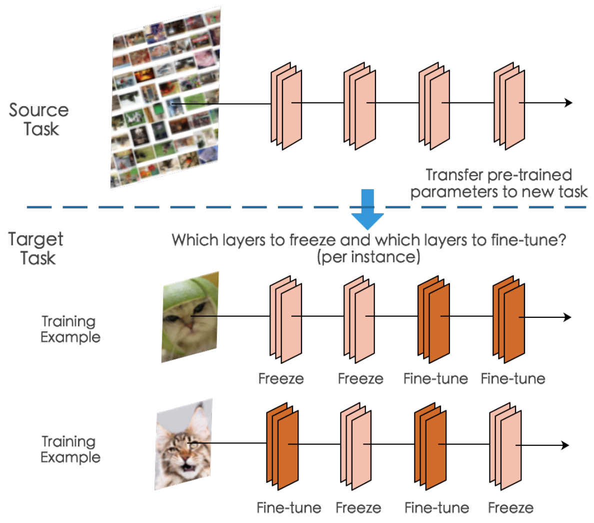


Figure 1.2: Given a deep neural network pre-trained on a source task, we address the question of *where to fine-tune* its parameters with examples of the target task.

the accuracy of the model in the target domain. The policy is sampled from a discrete distribution parameterized by the output of a lightweight neural network, which decides which layers of a pre-trained model should be fine-tuned or have their parameters frozen, on a per instance basis. As these decision functions are discrete and non-differentiable, In *SpotTune* we rely on a recent Gumbel Softmax sampling approach [MMT16, JGP16] to train the policy network. At test time, the policy decides whether the features coming out of a layer go into the next layer with the source pre-trained parameters or the fine-tuned parameters. In *AdaFilter*, we employ the *Straight Through Estimator* to allow differentiation through the discrete nodes.

We conduct extensive empirical evaluation of the proposed approaches, comparing it with several competitive baselines. The proposed SpotTune outperforms standard fine-tuning on 12 out of 14 datasets. Moreover, we show the effectiveness of *SpotTune* compared to other state-of-the-art fine-tuning strategies. On the Visual Decathlon Challenge [RBV17], which is a competitive benchmark for testing the performance of transfer learning algorithms with a total of 10 datasets, the proposed approach achieves the highest score compared with the state-of-the-art methods. Similar experimental results also demonstrate the competitive performance of the proposed AdaFilter approach.

Cross-domain few-shot learning: In Chapter 5, we further consider the case that the target domain only has few examples per class. We refer to this setting as cross-domain few-shot learning. We establish a new benchmark for cross-domain few-shot learning (CD-FSL), consisting of images from a diversity of domains with varying similarity to ImageNet, and lacking data for meta-learning. We extensively evaluate the performance of current meta-learning methods and variants of fine-tuning. The results show the following observations for CD-FSL: 1) meta-learning underperforms compared to fine-tuning, 2) accuracy gain with additional data is increased for fine-tuning versus meta-learning, 3) no individual fine-tuning method dominates performance versus the others across the benchmark, and 4) a general positive correlation between accuracy and dataset similarity to ImageNet exists. Finally, we propose *Incremental Multi-model Selection*, a method which integrates multiple pretrained models for cross-domain few-shot learning, and demonstrates best average performance on the new benchmark.

Learning across multiple domains

SharingNet: In Chapter 6, we propose a multi-domain learning approach based on depthwise separable convolution which can classify images from different domains simultaneously. Depthwise separable convolution has been proved to be a powerful variation of standard convolution for many applications, such as image classification [Cho17], natural language processing [KGC17] and embedded vision applications [HZC+17]. To the best of our knowledge, this is the first work that explores depthwise separable convolution for multi-domain learning. The proposed multi-domain learning model is compact and easily extensible. To promote transfer learning between different domains we further introduce a softmax gating mechanism. We evaluate our method on Visual Decathlon Challenge [RBV17]. Our method beats the state-of-the-art models with only 50% of the parameters.

Efficient multi-domain learning in sensor networks: In Chapter 7, we extend the idea of learning across multiple domains to the case of sensor networks. Training ML models in heterogeneous mobile networks incurs a large communication cost due to the necessity to deliver the local data to a central server. Inspired by active learning, which is traditionally used to reduce the labeling cost for training ML models, we propose an active sampling method to reduce the communication cost of learning in heterogeneous mobile networks. Instead of sending all the local data, the proposed active sampling method identifies and sends only informative data from each device to the central server. Extensive experiments on four real datasets, both with numerical simulation and on a networked mobile system, show that the proposed method can reduce the communication cost by up to 53% and energy consumption by up to 67% without accuracy degradation compared with the conventional approaches.

MEGA In Chapter 8, we propose a lifelong learning algorithm called MixEd stochastic GrAdient (MEGA) which can learn on multiple domains sequentially. We first present a unified view of episodic memory based lifelong learning methods including GEM

[LP$^+$17] and A-GEM [CRRE18] from an optimization's perspective. Specifically, we cast the problem of avoiding catastrophic forgetting in lifelong learning as an optimization problem with composite objective. We approximately solve the optimization problem using one-step stochastic gradient descent with the standard gradient replaced by the proposed mixed stochastic gradient. The mixed stochastic gradient is derived from the gradients computed on the data of the current task and an episodic memory which stores a small subset of observed examples from old tasks [LP$^+$17, CRRE18, RCA$^+$18]. We show that both GEM [LP$^+$17] and A-GEM [CRRE18] are degenerate cases of MEGA which consistently put the same emphasis on the current task, regardless of how the loss changes over time. In contrast, based on our derivation, the direction of the proposed mixed stochastic gradient balances the old tasks and the new task in an adaptive manner by considering the performance of the model on old tasks and on the new task. Therefore, the proposed MEGA algorithm allows deep neural networks to learn new tasks while avoiding catastrophic forgetting. Extensive experiments show the performance improvement of the proposed approach compared with existing lifelong learning algorithms.

Chapter 2

Preliminaries

2.1 Convolutional Neural Networks

Since the success on the ImageNet dataset [KSH12], deep neural networks has drawn a huge amount of attention from academia, industry and media. Subsequent works show that deep neural network models can achieve the state-of-the-art results on many real-world tasks, such as computer vision [HZRS16], natural language processing [YHPC17] and speech recognition [HDY+12]. Convolutional neural network (CNN) is a specialized type of deep neural network that has been successfully applied in many areas, especially in visual imagery [LBBH98].

Convolutional neural networks typically consist of multiple convolutional layers and pooling layers. In convolutional layers, convolution operation is employed to extract the features from the input. In each convolutional layer we have a set of filters. During the forward pass, we slide each filter across the input and compute dot products between the filter and the local receptive field. The output of the convolutional layer is called activation map that gives the response of each filter. Given an image I and a $m \times n$ filter F, an element $a_{i,j}$ in the activation map can be computed as,

$$a_{i,j} = \sum_{a=1}^{m} \sum_{b=1}^{n} I_{i+a-1,j+b-1} \times F_{a,b} \tag{2.1}$$

Convolutional layers have several properties which make them suitable for building deep neural networks: sparse interactions, parameter sharing and equivariant representations [GBCB16]. Sparse interactions refer to the fact that the convolutional kernels only apply on a small region of the input, thus the number of parameters can be greatly reduced. Since the same convolutional filter is applied in different region the input, the parameters are essentially shared in the convolution operation. The fact of parameter sharing further leads to equivariant representations, which indicates that if the input changes, the output changes in the same manner. A simple convolutional neural network is shown in Figure 2.1.

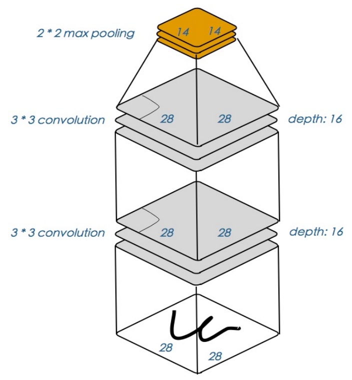


Figure 2.1: A simple illustration of convolutional neural networks.

Convolutional neural networks extract the features from the input in a hierarchical manner. As shown in Figure 2.2, the initial layers of convolutional neural networks are used for extracting lower features such as edges and textures. The middle layers are used for extracting features such as corners and contours. The top layers are used for actually detecting the actual objects in the images. Thus a natural way to leverage convolutional neural networks for transfer learning is to pre-train the network on a source domain and adjust the trained weights on the target domain, which is also referred to as fine-tuning. A widely used variant of fine-tuning is to share the initial layers of the pre-trained model on the source domain with the target domain for extracting low-level features.

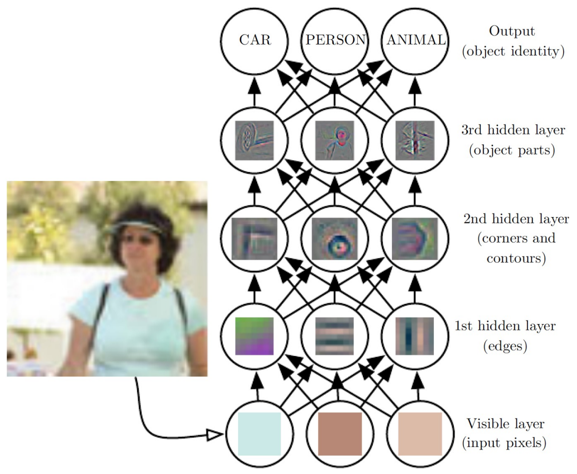

CAR
PERSON
ANIMAL
Output (object identity)
3rd hidden layer (object parts)
2nd hidden layer (corners and contours)
1st hidden layer (edges)
Visible layer (input pixels)

Figure 2.2: An illustration of using convolutional neural networks for feature extraction [GBCB16].

2.2 Learning Scenarios

In this dissertation, we focus on learning across multiple domains with deep convolutional neural networks. Especially, we consider learning across with two image domains, learning with multiple image domains , learning on a continuum of domains and cross-domain few-shot learning.

Learning across two domains

We define a *domain* as a joint distribution P over input space $\mathcal{X}$ and label space $\mathcal{Y}$. The marginal distribution of $\mathcal{X}$ is denoted as $P_{\mathcal{X}}$. We use the pair (x, y) to denote a sample x and the corresponding label y from the joint distribution P. For a deep neural network f_θ: $\mathcal{X} \rightarrow \mathcal{Y}$ with parameter θ, given a loss function ℓ, typically the cross-entropy loss for classification, the expected error of the network is defined as,

$$\epsilon(f_\theta) = E_{(x,y)\sim P}[\ell(f_\theta(x), y)] \tag{2.2}$$

Since the joint distribution is directly accessible, instead we minimize the following empirical error,

$$\tilde{\epsilon}(f_\theta) = \frac{1}{|D_{tr}|} \sum_{i=1}^{|D_{tr}|} \ell(f_\theta(x), y) \tag{2.3}$$

where D_{tr} is a training dataset. The empirical error is the average error over the samples in the training set D_{tr}.

Suppose we have a source domain $(\mathcal{X}_s, \mathcal{Y}_s)$ and a target domain $(\mathcal{X}_t, \mathcal{Y}_t)$ with joint distribution P_s and P_t respectively, and specially $P_{\mathcal{X}_s} \neq P_{\mathcal{X}_t}$. In transfer learning, the model is first trained on the source domain, the trained model is then adjusted on the target domain. Models trained on ImageNet [DDS$^+$09] are shown to be transferred across different vision-based tasks, such as objection detection [HGDG17] and semantic segmentation [GDDM14]. Empirical results also show that higher ImageNet accuracy leads to higher overall object detection accuracy [HRS+17], which demonstrates the benefits of transfer learning.

Learning with multiple domains simultaneously

Consider a set of image domains $\{D_1, D_2, ..., D_T\}$, each domain D_i consists of a triplet $\{X_i, Y_i, P_i\}$. $X_i \in \mathbb{R}^{C_i \times H_i \times W_i}$ is the input image space, $Y_i \in \{1, 2, ..., L_i\}$ is the output label space and P_i is the joint probability of (X_i, Y_i). In multi-domain learning, our goal is to design neural network architectures that can work well on all the domains simultaneously. Let $\mathcal{E}_{(D_i)}$ be the domain-specific parameters for domain D_i and $\mathcal{C}$ be the parameters that are shared across all the domains. For $x \in X_i$, the output of the network can be calculated as,

$$\hat{y} = (\mathcal{E}_{(D_i)} \circ \mathcal{C})(x) \tag{2.4}$$

where $\circ$ is the operation that connects the domain-specific parameters and sharable parameters which depends on the specific neural network architecture. The average error of the neural network across all the domains can be calculated as,

$$R = \frac{1}{T} \sum_{i=1}^{T} \mathbb{E}[\ell(y, (\mathcal{E}_{(D_i)} \circ \mathcal{C})(x)] \tag{2.5}$$

There are multiple goals in multi-domain learning. First, one natural goal is to minimize the average risk across different domains. Instead of training separate model for each domain, it has been shown that by sharing a part of parameters across multiple domains, we can achieve higher average test accuracy. One reason for the performance boost is that by sharing part of the parameters, the issue of overfitting can be alleviated. Moreover, by training the sharable parameters on some large datasets, we can extract transferable features can be used for smaller datasets. Second goal is to minimize the size of the domain-specific part $\mathcal{E}_{(D_i)}$. By reducing the domain-specific part, the model size can be made as small as possible to enable it to be deployed on resource limited devices.

Learning with multiple domains sequentially

Learning with multiple domains sequentially, also called lifelong learning (LLL) [RRD+16, KPR+17, LP+17, CRRE18] in this dissertation, considers the problem of learning a new domain without degrading performance on old domains, i.e., to avoid *catastrophic forgetting* [Fre99, KPR+17]. Suppose there are T tasks which are characterized by T datasets: $\{D_1, D_2, .., D_T\}$. Each dataset D_t consists of a list of triplets (x_i, y_i, t), where y_i is the label of i-th example x_i, and t is a task descriptor that indicates which task the example coming from. Similar to supervised learning, each dataset D_t is split into a training set D_t^{tr} and a test set D_t^{te}.

In the learning protocol introduced in [CRRE18], the tasks are separated into D^{CV} = $\{D_1$, D_2, ..., $D_{T^{CV}}\}$ and D^{EV} = $\{D_{T^{CV}+1}$, $D_{T^{CV}+2}$, ..., $D_T\}$. D^{CV} is used for cross-validation to search for hyperparameters. D^{EV} is used for actual training and evaluation. As pointed out in [CRRE18], some regularization-based lifelong learning algorithms, e.g., Elastic Weight Consolidation [KPR+17], are sensitive to the choice of the regularization parameters. Introducing D^{CV} can help find the best regularization parameter without exposing the actual training and evaluation data. While searching for the hyperparameters, we can have multiple passes over the examples in D^{CV}, the training is performed on D^{EV} with only a *single* pass over the examples [LP^{+}17, CRRE18].

In lifelong learning, a given model f_θ is trained sequentially on a series of tasks $\{D_{T^{CV}+1},\ D_{T^{CV}+2},\ ...,\ D_T\}$. When the model f_θ is trained on task D_t, the goal is to predict the labels of the examples in D_t^{te} by minimizing the empirical loss $\ell_t(w)$ on D_t^{tr} in an online fashion without suffering accuracy drop on $\{D_{T^{CV}+1}^{te},\ D_{T^{CV}+2}^{te},\ ...,\ D_t^{te}\}$.

Cross-domain few-shot learning

Cross-domain few-shot learning is closely related to the standard few-shot learning [SSZ17]. In cross-domain few-shot learning, we have a source domain $(\mathcal{X}_s, \mathcal{Y}_s)$ and a target domain $(\mathcal{X}_t, \mathcal{Y}_t)$ with joint distribution P_s and P_t respectively, $P_{\mathcal{X}_s} \neq P_{\mathcal{X}_t}$, and $\mathcal{Y}_s$ is disjoint from $\mathcal{Y}_t$. The base classes data are sampled from the source domain and the novel classes data are sampled from the target domain. During the training or meta-training stage, the model f_θ is trained (or meta-trained) on the base classes data. During testing (or meta-testing) stage, the model is presented with a *support set* $S = \{x_i, y_i\}_{i=1}^{K\times N}$ consisting of N examples from K novel classes. This configuration is referred to as "K-way N-shot" few-shot learning, as the support set has K novel classes and each novel class has N training examples. After the model is adapted to the support set, a *query set* from novel classes is used to evaluate the model performance.

Chapter 3

Learning with Two Domains: Adaptive Block Fine-tuning

3.1 Introduction

The standard approach of transfer learning with deep neural networks is to slightly adjust the pre-trained model on the target domain, also called fine-tuning. There are several choices when it comes to realizing the idea of fine-tuning of deep neural networks in practice. A natural approach is to optimize *all* the parameters of the deep neural network using the target training data (after initializing them with the parameters of the pre-trained model). However, if the target dataset is small and the number of parameters is huge, fine-tuning the whole network may result in overfitting [YCBL14]. Alternatively, the last few layers of the deep network can be fine-tuned while freezing the parameters of the remaining initial layers to their pre-trained values [TSG$^+$16, ARS$^+$16]. This is driven by a combination of limited training data in the target task and the empirical evidence that initial layers learn low-level features that can be directly shared across various computer vision tasks. However, the number of initial layers to freeze during fine-tuning still remains a manual design choice which can be inefficient to optimize for, especially for networks with hundreds or thousands of layers. Further, it has been empirically observed that current successful multi-path deep architectures such as ResNets [HZRS16] behave like ensembles of shallow networks [VWB16]. It is not clear if restricting the fine-tuning to the last contiguous layers is the best option, as the ensemble effect diminishes the assumption that early or middle layers should be shared with common low-level or mid-level features.

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

In this dissertation, we propose two adaptive fine-tuning methods, *SpotTune* and *AdaFilter*, to automatically decide which residual block and which filters should be fine-tuned. In this chapter, we propose *SpotTune*, an approach to learn a decision policy for input-dependent fine-tuning. The policy is sampled from a discrete distribution parameterized by the output of a lightweight neural network, which decides which layers of a pre-trained model should be fine-tuned or have their parameters frozen, on a per instance basis. As these decision functions are discrete and non-differentiable, we rely on a recent Gumbel Softmax sampling approach [MMT16, JGP16] to train the policy network. At test time, the policy decides whether the features coming out of a layer go into the next layer with the source pre-trained parameters or the fine-tuned parameters.

We summarize our contributions of SpotTune as follows:

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

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

- We conduct extensive empirical evaluation of the proposed approach, comparing it with several competitive baselines. The proposed approach outperforms standard fine-tuning on 12 out of 14 datasets. Moreover, we show the effectiveness of *SpotTune* compared to other state-of-the-art fine-tuning strategies. On the Visual Decathlon Challenge [RBV17], which is a competitive benchmark for testing the performance of multi-domain learning algorithms with a total of 10 datasets, the proposed approach achieves the highest score compared with the state-of-the-art methods.

3.2 Related Work

Transfer Learning There is a long history of transfer learning and domain adaptation methods in computer vision [Csu17, PY^{+}10]. Early approaches have concentrated on *shallow classifiers*, using techniques such as instance re-weighting [WQXY07, DPS06], model adaptation [DTXM09], and feature g2012geodesic. Recently, transfer learning based on deep neural networks has received significant attention in the community [GUA^{+}16, CHF^{+}15, CBG13, KSW^{+}18, GY17]. Fine-tuning a pre-trained network model such as ImageNet on a new dataset is the most common strategy for knowledge transfer in the context of deep learning. Methods have been proposed to fine-tune all network parameters [GDDM14], only the parameters of the last few layers [LCWJ15], or to just use the pre-trained model as a fixed feature extractor with a classifier such as SVM on top [SRASC14]. Kornblith et al. [KSL18] studied several of these options to address the question of whether better ImageNet models transfer better. Yosinski et al. [YCBL14] conducted a study on the impact of transferability of features from the bottom, middle, or top of the network with early models, but it is not clear whether their conclusions hold for modern multi-path architectures such as Residual Networks [HZRS16] or DenseNets [HLvdMW17]. Yang et al. [YDH^{+}18] have recently proposed to learn relational graphs as transferable representations, instead of unary features. In a more recent work, Li et al. [LGD18] investigated several regularization schemes that explicitly promote the similarity of the fine-tuned model with the original pre-trained model. In contrast, our proposed SpotTune and AdaFilter adaptively fine-tune different layers and filters of the pre-trained model which do not require the manual decision process.

Dynamic Routing The proposed SpotTune and AdaFilter are related to *conditional computation* methods [BLC13, LD18, FCZ$^+$17], which aim to dynamically route information in neural networks with the goal of improving computational efficiency. Bengio et al. [BBPP15] used sparse activation policies to selectively execute neural network units on a per-example basis. Shazeer et al. [SMM$^+$17] introduced a Sparsely-Gated Mixture-of-Experts layer, where a trainable gating network determines a sparse combination of sub-networks (experts) to use for each example. Wu, Nagarajan et al. proposed *BlockDrop* [WNK$^+$18], a method that uses reinforcement learning to dynamically select which layers of a Residual Network to execute, exploiting the fact that ResNets are resilient to layer dropping [VWB16]. Veit and Belongie [VB18] investigated the same idea using Gumbel Softmax [JGP16] for on-the-fly selection of residual blocks. In SpotTune and AdaFilter, the dynamic routing based on the *Gumbel trick* and *Straight Through Estimator*. Unlike previous methods, our goal is to determine the parameters in a neural network that should be frozen or fine-tuned during learning to improve accuracy, instead of dropping layers to improve efficiency.

Multi-task Learning Multi-task learning [BV16, DZ17, Kok17, WHG17] aims at extracting different features from a single input to simultaneously perform classification, object recognition, edge detection, etc. Various applications can be benefited from a multi-task learning approach since the training signals can be reused among related tasks

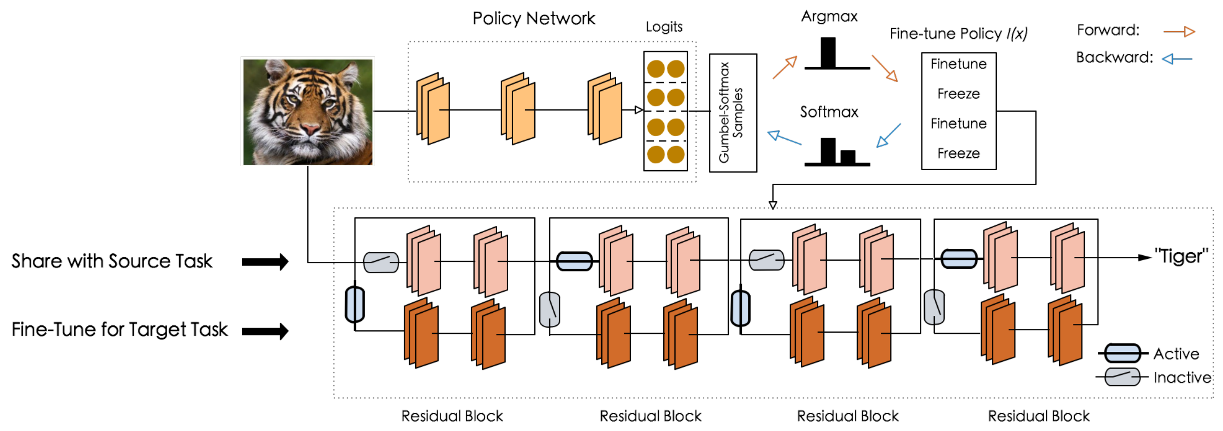


Figure 3.1: Illustration of the *SpotTune* method.

[Car97, ZSS+18]. In the multi-task setting, knowing which tasks or parameters are shareable is a longstanding challenge [KGS11, KDI12, TO98, LKZ+17]. Early methods were designed for shallow classification models [ZCY11, JVB09, PRWDI12], while more recent approaches address the problem of "with whom" each task should share features using deep neural networks [LKZ+17, MM18]. Cross-stitching networks [MSGH16] and Progressive Networks [RRD+16] have been recently proposed to learn an optimal combination of shared and task-specific representations for joint multi-task optimization and life-long learning, respectively. These methods rely on per-layer inter-column adapters, which requires more memory and leads to more computational cost.

3.3 SpotTune

Given a pre-trained network model on a source task (e.g., ImageNet pre-trained model), and a set of training examples with associated labels in the target domain, our goal is to create an adaptive fine-tuning strategy that decides, per training example, which layers of the pre-trained model should be fine-tuned (adapted to the target task) and which layers should have their parameters frozen (shared with the source task) during training, in order to improve the accuracy of the model in the target domain. To this end, we first present an overview of our approach in Section 3.3.1. Then, we show how we learn our adaptive fine-tuning policy using Gumbel Softmax sampling in Section 3.3.2. Finally, in Section 3.3.3, we present a global policy variant of our proposed image-dependent fine-tuning method, which constrains all the images to follow a single fine-tuning policy.

3.3.1 SpotTune Overview

Although our approach could be applied to different deep neural network architectures, in the following we focus on a Residual Network model (ResNet) [HZRS16]. Recently, it has been shown that ResNets behave as ensembles of shallow classifiers and are resilient to residual block swapping [VWB16]. This is a desirable property for our approach, as later we show that SpotTune dynamically swaps pre-trained and fine-tuned blocks to improve performance.

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

$$x_l = F_l(x_{l-1}) + x_{l-1}. \tag{3.1}$$

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

$$x_l = I_l(x)\hat{F}_l(x_{l-1}) + (1 - I_l(x))F_l(x_{l-1}) + x_{l-1} \tag{3.2}$$

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

Figure 3.1 illustrates the architecture of our proposed *SpotTune* method, which allows each training image to have its own fine-tuning policy. During training, the policy network is jointly trained with the target classification task using Gumbel Softmax sampling, as we will describe next. At test time, an input image is first fed into a policy network, whose output is sampled to produce routing decisions on whether to pass the image through the fine-tuned or pre-trained residual blocks. The image is then routed through the corresponding residual blocks to produce the final classification prediction. Note that the effective number of executed residual blocks is the same as the original pre-trained model. The only additional computational cost is incurred by the policy network, which is designed to be lightweight (only a few residual blocks) in comparison to the original pre-trained model.

3.3.2 Training with the Gumbel Softmax Policy

SpotTune makes decisions as to whether or not to freeze or fine-tune each residual block per training example. However, the fact that the policy $I_l(x)$ is discrete makes the network non-differentiable and therefore difficult to be optimized with backpropagation. There are several ways that allow us to "back-propagate" through the discrete nodes [BLC13]. In this work, we use a recently proposed Gumbel Softmax sampling approach [MMT16, JGP16] to circumvent this problem.

The Gumbel-Max trick [MMT16] is a simple and effective way to draw samples from a categorical distribution parameterized by $\{\alpha_1, \alpha_2, ..., \alpha_z\}$, where α_i are scalars not confined to the simplex, and z is the number of categories. In our work, we consider two categories (freeze or fine-tune), so $z = 2$, and for each residual block, α_1 and α_2 are scalars corresponding to the output of a policy network.

A random variable G is said to have a standard Gumbel distribution if $G = -\log(-\log(U))$ with U sampled from a uniform distribution, i.e. $U \sim Unif[0, 1]$. Based on the Gumbel-Max trick [MMT16], we can draw samples from a discrete distribution parameterized by α_i in the following way: we first draw i.i.d samples $G_i, ..., G_z$ from

Gumbel$(0, 1)$ and then generate the discrete sample as follows:

$$X = \arg\max_i [\log \alpha_i + G_i]. \tag{3.3}$$

The arg max operation in Equation 3.3 is non-differentiable. However, we can use the Gumbel Softmax distribution [MMT16, JGP16], which adopts softmax as a continuous relaxation to arg max. We represent X as a one-hot vector where the index of the non-zero entry of the vector is equal to X, and relax the one-hot encoding of X to a z-dimensional real-valued vector Y using softmax:

$$Y_i = \frac{\exp((\log \alpha_i + G_i)/\tau)}{\sum_{j=1}^{z} \exp((\log \alpha_j + G_j)/\tau)} \quad \text{for } i = 1, .., z \tag{3.4}$$

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

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

Similar to [WNK+18], we generate all freeze/fine-tune decisions for all residual blocks at once, instead of relying on features of intermediate layers of the pre-trained model to obtain the fine-tuning policy. More specifically, suppose there are L residual blocks in the pre-trained model. The output of the policy network is a two-dimensional matrix $\beta \in \mathbb{R}^{L \times 2}$. Each row of β represents the logits of a Gumbel-Softmax Distribution with two categories, i.e, $\beta_{l,0} = \log \alpha_1$ and $\beta_{l,1} = \log \alpha_2$. After obtaining β, we use the straight-through version of the Gumbel-Softmax estimator [JGP16]. During the forward pass, we sample the fine-tuning policy $I_l(x)$ using Equation 3.3 for the l-th residual block. During the backward pass, we approximate the gradient of the discrete samples by computing the gradient of the continuous softmax relaxation in Equation 3.4. This process is illustrated in Figure 3.1.

3.3.3 Compact Global Policy Variant

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

Consider a pre-trained ResNet model with L residual blocks. For the l-th block, we can obtain the number of images that use the fine-tuned block and the pre-trained block based on the image-specific policy. We compute the fraction of images in the target dataset that uses the fine-tuned block and denote it as $v_l \in [0, 1]$. In order to constrain our method to fine-tune k blocks, we introduce the following loss:

$$l_k = ((\sum_{l=1}^{L} v_l) - k)^2. \tag{3.5}$$

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

$$l_e = \sum_{l=1}^{L} -v_l \log v_l. \tag{3.6}$$

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

$$l = l_c + \lambda_1 l_k + \lambda_2 l_e, \tag{3.7}$$

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

3.4 Experimental Setup

Datasets and metrics. We compare our SpotTune method with other fine-tuning and regularization techniques on 5 public datasets, including three fine-grained classification benchmarks: CUBS [WBW$^+$11], Stanford Cars [KSDFF13] and Flowers [NZ08], and two datasets with a large domain mismatch from ImageNet: Sketches [EHA12] and WikiArt [SE15]. The statistics of these datasets are listed in Table 3.1. Performance is measured by classification accuracy on the evaluation set.

We also report results on the datasets of the Visual Decathlon Challenge [RBV17], which aims at evaluating visual recognition algorithms on images from multiple visual domains. There are a total of 10 datasets as part of this challenge: (1) ImageNet, (2) Aircraft, (3) CIFAR-100, (4) Describable textures, (5) Daimler pedestrian classification, (6) German traffic signs, (7) UCF-101 Dynamic Images, (8) SVHN, (9) Omniglot, and (10) Flowers. The images of the Visual Decathlon datasets are resized isotropically to have a shorter side of 72 pixels, in order to alleviate the computational burden for evaluation. Following [RBV17], the performance is measured by a single scalar score $S = \sum_{i=1}^{10} \alpha_i \max\{0, E_i^{\max} - E_i\}^2$, where E_i is the test error on domain D_i, and $E_i^{\max}$ is the error of a reasonable baseline algorithm. The coefficient α_i is $1000(E_i^{\max})^{-2}$, so a perfect classifier receives score 1000. The maximum score achieved across 10 domains is 10000. Compared with average accuracy across all the 10 domains, the score S is a more reasonable measurement for comparing different algorithms, since it considers the difficulty of different domains, which is not captured by the average accuracy [RBV17].

Table 3.1: Datasets used to evaluate SpotTune against other fine-tuning baselines.

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

In total, our experiments comprise 14 datasets, as the Flowers dataset is listed in both sets described above. We note that for the experiments in Table 6.2, we use the full resolution of the images, while those are resized in the Visual Decathlon experiments to be consistent with other approaches.

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

- **Standard Fine-tuning**: This baseline fine-tunes all the parameters of the pre-trained network on the target dataset [GDDM14, YCBL14].
- **Feature Extractor**: We use the pre-trained network as a feature extractor [SRASC14, DJV+14] and only add the classification layer for each newly added dataset.
- **Stochastic Fine-tuning**: We randomly sample 50% of the blocks of the pre-trained network to fine-tune.
- **Fine-tuning last-k** (k = 1, 2, 3): This baseline fine-tunes the last k residual blocks of the pre-trained network on the target dataset [LCWJ15, TSG+16, ARS+16]. In our experiments, we consider fine-tuning the last one ($k = 1$), last two ($k = 2$) and the last three ($k = 3$) residual blocks.

- **Fine-tuning ResNet-101**: We fine-tune all the parameters of a pre-trained ResNet-101 model on the target dataset. SpotTune uses ResNet-50 instead (for the experiments in Table 6.2), so this baseline is more computationally expensive and can fine-tune twice as many residual blocks. We include it as the total number of parameters during training is similar to SpotTune, so it will verify any advantage is not merely due to our having 2x residual blocks available.
- **Random Policy**: This baseline method adopts a random policy network that always finetunes the last three layers and randomly decides whether to fine-tune or not for each training sample for other layers.
- **L^2-SP** [LGD18]: This is a recently proposed state-of-the-art regularization method for fine-tuning. The authors recommend using an L^2 penalty to allow the fine-tuned network to have an explicit inductive bias towards the pre-trained model, sharing similar motivation with our approach.
- **Progressive Neural Networks** [RRD+16]: This is a recent method which learns an optimal combination of shared and task-specific representations for lifelong learning. Different form the original work, which uses a random weight initialization, we use an ImageNet pre-trained model as the frozen source network, since the former leads to much worse performance for classification.

Regarding the methods that have reported results on the Visual Decathlon datasets, the most related to our work are models trained from *Scratch*, *Standard Fine-tuning*, the *Feature Extractor* baseline as described above, and *Learning without Forgetting (LwF)*

Table 3.2: Results of *SpotTune* and baselines on CUBS, Stanford Cars, Flowers, WikiArt and Sketches.

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

[LH17], which is a recently proposed technique that encourages the fine-tuned network to retain the performance on ImageNet or previous tasks, while learning consecutive tasks. Other methods include *Piggyback* [ML18], *Residual Adapters* and its variants [RBV17, RBV18b], *Deep Adaptation Networks (DAN)* [RT17], and *Batch Norm Adaptation (BN Adapt)* [BV17], which are explicitly designed to minimize the number of model parameters, while our method sits at the other end of the spectrum, with a focus on accuracy instead of parameter reduction. We also compare with training from scratch using Residual Adapters (*Scratch+*), as well as the high-capacity version of Residual Adapters described in [RBV17], which have a similar number of parameters as SpotTune.

Pre-trained model. For comparing SpotTune with fine-tuning baselines in Table 6.2, we use ResNet-50 pre-trained on ImageNet, which starts with a convolutional layer followed by 16 residual blocks. The residual blocks contain three convolutional layers and are distributed into 4 segments (i.e, [3, 4, 6, 3]) with downsampling layers in between. We use the pre-trained model from Pytorch which has a classification accuracy of 75.15% on ImageNet. For the Visual Decathlon Challenge, we use a ResNet-26 as described in [RBV18b].

Policy network architecture. For the experiments with ResNet-50 (Table 6.2), we use a ResNet with 4 blocks for the policy network. The channel size of each block is 64, 128, 256, 512, respectively. For the Visual Decathlon Challenge with ResNet-26, the policy network consists of a ResNet with 3 blocks. The channel size of each block is 64, 128, 256, respectively.

Implementations details. Our implementation is based on Pytorch. All models are trained on 2 NVIDIA V100 GPUs. For comparing SpotTune with fine-tuning baselines, we use SGD with momentum as the optimizer. The momentum rate is set to be 0.9, the initial learning rate is 1e-2 and the batch size is 32. The initial learning rate of the policy network is 1e-4. We train the network with a total of 40 epochs and the learning rate decays twice at 15th and 30th epochs with a factor of 10. For the Visual Decathlon Challenge, we also use SGD with momentum as the optimizer. The momentum rate is 0.9 and the initial learning rate is 0.1. The batch size is 128. The initial learning rate of the policy network is 1e-2. We train the network with a total of 110 epochs and the learning rate decays three times at 40th, 60th and 80th epochs with a factor of 10. We freeze the first macro blocks (4 residual blocks) of the ResNet-26 and only apply the adaptive fine-tuning for the rest of the residual blocks. This choice reduces the number of parameters and has a regularization effect. The temperature of the Gumbel-Softmax distribution is set to 5 for all the experiments.

3.5 Results and Analysis

3.5.1 SpotTune vs. Fine-tuning Baselines

The results of SpotTune and the fine-tuning baselines are listed in Table 6.2. Clearly, SpotTune yields consistently better results than other methods. Using the pre-trained model on ImageNet as a *feature extractor* (with all parameters frozen) can reduce the number of parameters when the model is applied to a new dataset, but it leads to bad performance due to the domain shift. All the fine-tuning variants (*Standard Fine-tuning, Stochastic Fine-tuning, Fine-tuning last-k*) achieve higher accuracy than the *Feature Extractor* baseline, as expected. Note that the results of *Fine-tuning last-k* show that manually deciding the number of layers to fine-tune may lead to worse results than *standard fine-tuning*. The *Fine-tuned ResNet-101* has higher capacity and thus performs better than the other fine-tuning variants. Although it has twice as many fine-tuned blocks and is significantly more computationally expensive than SpotTune, it still performs worse than our method in all datasets, except in WikiArt. We conjecture this is because WikiArt has more training examples than the other datasets. To test this hypothesis, we evaluated both models when 25% of the WikiArt training data is used. In this setting, SpotTune achieves 61.24% accuracy compared to 60.20% of the fine-tuned ResNet-101. This gap increases even more when 10% of the data is considered (49.59% vs. 47.05%).

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

SpotTune is different from all the baselines in two aspects. On one hand, the fine-tuning policy in SpotTune is specialized for each instance in the target dataset. This implicitly takes the similarities between the images in the target dataset and the source dataset into account. On the other hand, sharing layers with the source task without parameter refinement reduces overfitting and promotes better re-use of features extracted from the source task. We also consider three variants of SpotTune in the experiments. The first one is *SpotTune (running fine-tuned blocks)* in which during testing all the images are routed through the fine-tuned blocks. With this setting, the accuracy drops on all the datasets. This suggests that certain images in the target data can benefit from reusing

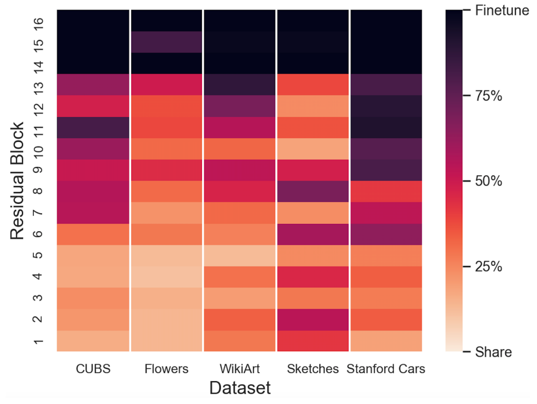


Figure 3.2: Visualization of policies on CUBS, Flowers, WikiArt, Sketches and Stanford Cars.

some of the layers of the pre-trained network. The second variant is *SpotTune (global-k)* in which we set k to 3 in the experiments. Generally, SpotTune (global-3) performs worse than SpotTune, but is around 3 times more compact and, interestingly, is better than *Fine-tuning last-3*. This suggests that it is beneficial to have an image-specific fine-tuning strategy, and manually selecting the last k layers is not as effective as choosing the optimal non-contiguous set of k layers for fine-tuning. The third variant is *Random Policy* where we always fine-tune the last three layers and use a random policy network for other layers. The results show that an optimized policy outperforms a random policy.

3.5.2 Visualization of Policies

To better understand the fine-tuning policies learned by the policy network, we visualize them on CUBS, Flowers, WikiArt, Sketches, and Stanford Cars in Figure 3.2. The polices are learned on a ResNet-50 which has 16 blocks. The tone of red of a block indicates the number of images that were routed through the fine-tuned path of that block. For example, a block with a dark tone of red and a 75% level of fine-tuning (as shown in the scale depicted in the right of Figure 3.2) means 75% of the images in the test set use the fine-tuned block and the remaining 25% images share the pre-trained ImageNet block. The illustration shows that different datasets have very different fine-tuning policies. SpotTune allows us to automatically identify the right policy for each dataset, as well as for each training example, which would be infeasible through a manual approach.

3.5.3 Visualization of Block Usage

Besides the learned policies for each residual block, we are also interested in the number of fine-tuned blocks used by each dataset during testing. This can reveal the difference of the distribution of each target dataset and can also shed light on how the policy network works. In Figure 3.3, we show the distribution of the number of fine-tuned blocks used by each target dataset. During testing, for each dataset we categorize the test examples based on the number of fine-tuned blocks they use. For example, from Figure 3.3, we can see around 1000 images in the test set of the CUBS dataset use 7 fine-tuned blocks.

We have the following two observations based on the results. First, for a specific dataset, different images tend to use a different number of fine-tuned blocks. This again validates our hypothesis that it is more accurate to have an image-specific fine-tuning policy rather than a global fine-tuning policy for all images. Second, the distribution of fine-tuned blocks usage differs significantly across different target datasets. This demonstrates that based on the characteristics of the target dataset, standard fine-tuning (which optimizes all the parameters of the pre-trained network towards the target task) may not be the ideal choice when conducting transfer learning with convolutional networks.

3.5.4 Visual Decathlon Challenge

We show the results of SpotTune and the baselines on the Visual Decathlon Challenge in Table 3.3. Among all the baselines, SpotTune achieves the highest Visual De-

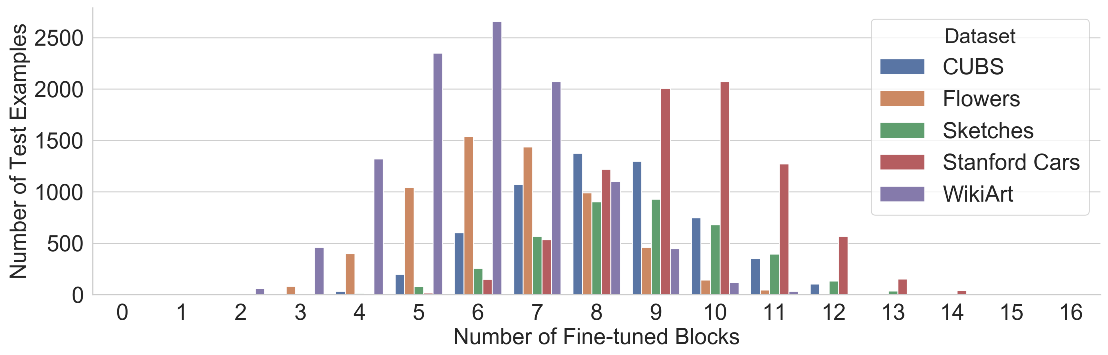


Figure 3.3: Distribution of the number of fine-tuned blocks used by the test examples. Different tasks and images require substantially different fine-tuning for best results, and this can be automatically inferred by SpotTune.

cathlon score. Compared to standard fine-tuning, SpotTune has almost the same amount of parameters and improves the score by a large margin (3612 vs 3096). Considering the Visual Decathlon datasets, and the 5 datasets from our previous experiments, Spot-Tune shows superior performance on 12 out of 14 datasets over standard fine-tuning. Compared with other recently proposed methods on the Visual Decathlon Challenge [ML18, RT17, RBV17, RBV18b, LH17], SpotTune sets the new state of the art for the challenge by only exploiting the transferability of the features extracted from ImageNet, without changing the network architecture. This is achieved without bells and whistles, i.e., we believe the results could be even further improved with more careful parameter tuning, and the use of other techniques such as data augmentation, including jittering images at test time and averaging their predictions. Compared to standard fine-tuning, our method uses 1.47x time in training (tested with 4 Titan Xp GPUs, batch size 96). At test time, the additional cost is negligible (0.013s vs 0.015s per image).

In SpotTune (Global-k), we fine-tune 3 blocks of the pre-trained model for each task which greatly reduces the number of parameters and still preserves a very competitive

Table 3.3: Results of SpotTune and baselines on the Visual Decathlon Challenge. The number of parameters is specified with respect to a ResNet-26 model as in [RBV17].

	#par	ImNet	Airc.	C100	DPed	DTD	GTSR	Flwr	OGlt	SVHN	UCF	Score
Scratch	10x	59.87	57.10	75.73	91.20	37.77	96.55	56.30	88.74	96.63	43.27	1625
Scratch+ [RBV17]	11x	59.67	59.59	76.08	92.45	39.63	96.90	56.66	88.74	96.78	44.17	1826
Feature Extractor	1x	59.67	23.31	63.11	80.33	55.53	68.18	73.69	58.79	43.54	26.80	544
Fine-tuning [RBV18b]	10x	60.32	61.87	82.12	92.82	55.53	99.42	81.41	89.12	96.55	51.20	3096
BN Adapt. [BV17]	1x	59.87	43.05	78.62	92.07	51.60	95.82	74.14	84.83	94.10	43.51	1353
LwF [LH17]	10x	59.87	61.15	82.23	92.34	58.83	97.57	83.05	88.08	96.10	50.04	2515
Series Res. adapt. [RBV17]	2x	60.32	61.87	81.22	93.88	57.13	99.27	81.67	89.62	96.57	50.12	3159
Parallel Res. adapt. [RBV18b]	2x	60.32	64.21	81.92	94.73	58.83	99.38	84.68	89.21	96.54	50.94	3412
Res. adapt. (large) [RBV17]	12x	67.00	67.69	84.69	94.28	59.41	97.43	84.86	89.92	96.59	52.39	3131
Res. adapt. decay [RBV17]	2x	59.67	61.87	81.20	93.88	57.13	97.57	81.67	89.62	96.13	50.12	2621
Res. adapt. finetune all [RBV17]	2x	59.23	63.73	81.31	93.30	57.02	97.47	83.43	89.82	96.17	50.28	2643
DAN [RT17]	2x	57.74	64.12	80.07	91.30	56.54	98.46	86.05	89.67	96.77	49.48	2851
PiggyBack [ML18]	1.28x	57.69	65.29	79.87	96.99	57.45	97.27	79.09	87.63	97.24	47.48	2838
SpotTune (Global-k)	4x	60.32	61.57	80.30	95.78	55.80	99.48	85.38	88.41	96.47	51.05	3401
SpotTune	11x	60.32	63.91	80.48	96.49	57.13	99.52	85.22	88.84	96.72	52.34	**3612**

score. Although we focus on accuracy instead of parameter reduction in our work, we note that training our global-k variant with a multi-task loss on all 10 datasets, as well as model compression techniques, could further reduce the number of parameters in our method. We leave this research thread for future work.

3.6 Conclusion

In this chapter, we proposed an adaptive fine-tuning algorithm called SpotTune which specializes the fine-tuning strategy for each training example of the target dataset. We showed that our method outperforms the key most popular and widely used protocols for fine-tuning on a variety of public benchmarks. We also evaluated SpotTune on the Visual Decathlon challenge, achieving the new state of the art, as measured by the overall score across the 10 datasets. In next chapter, we present the propose *Adafilter* which targets at reusing or fine-tuning different convolutional filters of the pre-trained model.

This chapter contains material from “SpotTune: Dynamic Transfer Learning via Adaptive Fine-tuning”, by Yunhui Guo, Honghui Shi, Abhishek Kumar,Tajana Rosing, Kristen Grauman, Rogerio Feris, which appears in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. The dissertation author was the primary investigator and author of this paper.

Chapter 4

Learning with Two Domains: Adaptive Filter Fine-tuning

4.1 Introduction

In last chapter, we present *SpotTune* which is a transfer learning method that can automatically decide which residual blocks to fine-tune and freeze. In this chapter, we further propose a deep transfer learning model, called *AdaFilter*, which automatically selects reusable filters from the pre-trained model on a per-example basis. Convolutional filters are basic feature extractors in a convolutional neural network. The convolutional filters of the pretrained model can detect the presence of specific features or patterns that are presented in the source domain data. Thus by transferring or fine-tuning the specific pre-trained convolutional filters based on the target domain data, we can have a better reuse of the pre-trained features by exploiting the domain similarity.

In AdaFilter, our goal is to allow different examples in the target dataset fine-tune or reuse different convolutional filters in the pre-trained model. This is achieved by using a recurrent neural network (RNN) gate [GMH13], which is conditioned on the activations of the previous layer, to layerwisely decide which filter should be reused and which filter should be further fine-tuned for each example in the target dataset. The adaptive filter fine-tuning scheme implicitly considers the similarity between the source domain and the target domain. Moreover, AdaFilter mitigates the overfitting issue by reducing the number of trainable parameters for each example in the target dataset via reusing pre-trained filters. We experiment on 7 publicly available image classification datasets. The results show that the proposed AdaFilter outperforms fine-tuning on all the datasets and achieves much faster convergence speed.

The proposed AdaFilter is complementary to the previous works in Section 3.2 of the last chapter. For example, the works that are based on fine-tuning can utilize AdaFilter to further improve the results. In AdaFilter, we allow different examples in the target dataset to fine-tune different convolutional filters in the pre-trained model. The examples in the target dataset which are similar to the source task can reuse more pre-trained filters to achieve better knowledge transfer.

The contributions of AdaFilter can be summarized as follows,

- We propose AdaFilter, a deep transfer learning algorithm which aims to improve the performance of the widely used fine-tuning method. We propose *filter selection*, *layer-wise recurrent gated network* and *gated batch normalization* techniques to allow

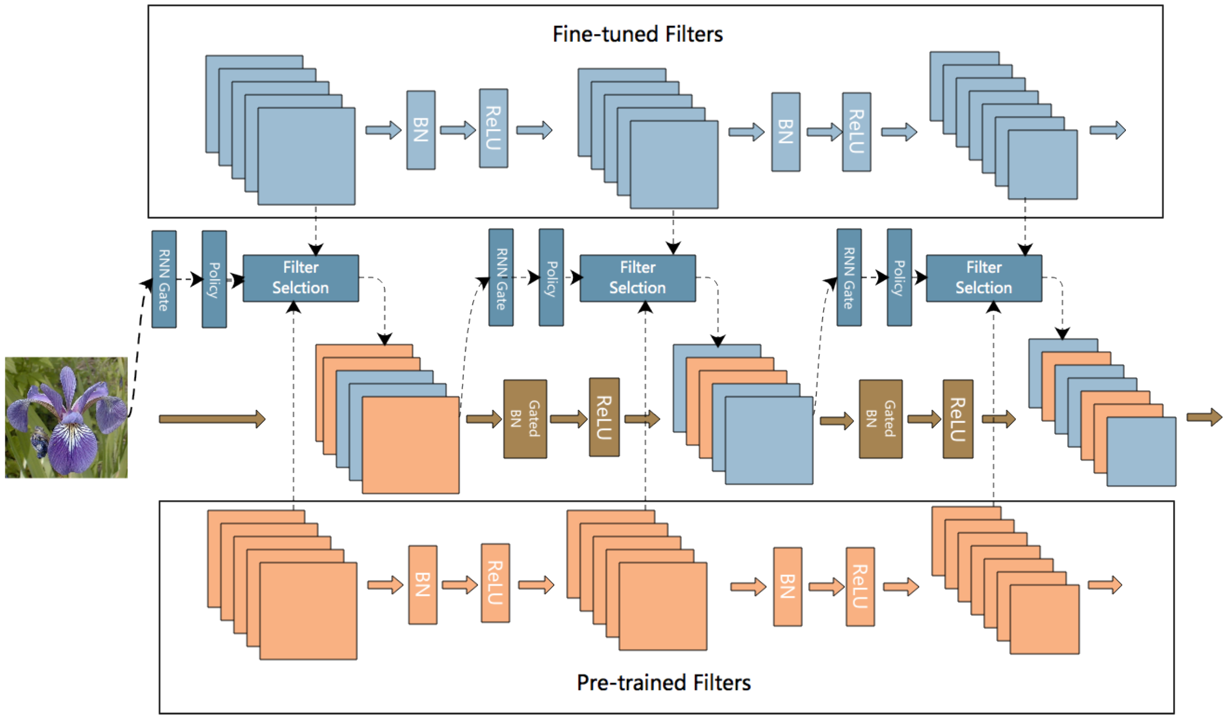


Figure 4.1: The overview of AdaFilter for deep transfer learning.

different images in the target dataset to fine-tune different convolutional filters in the pre-trained model.

- We experiment with 7 publicly available datasets and the results show that the proposed method can reduce the average classification error by 2.54% compared with the standard fine-tuning.

- We also show that AdaFilter can consistently perform better than the standard fine-tuning during the training process due to more efficient *knowledge transfer*.

4.2 AdaFilter

In this dissertation, we propose a deep transfer learning method which finds the convolutional filters in a pre-trained model that are reusable for each example in the target dataset. Figure 4.1 shows the overview of the proposed approach. We first use a *filter selection* method to achieve per-example fine-tuning scheme. We therefore leverage a *recurrent gated network* which is conditioned on the activations of the previous layer to layerwisely decide the fine-tuning policy for each example. Finally, we propose *gated batch normalization* to consider the different statistics of the output channels produced by the pre-trained layer and the fine-tuned layer.

4.2.1 Filter Selection

In convolutional neural network (CNN), convolutional filters are used for detecting the presence of specific features or patterns in the original images. The filters in the initial layers of CNN are used for detecting low level features such as edges or textures while the filters at the end of the network are used for detecting shapes or objects [ZF14]. When convolutional neural networks are used for transfer learning, the pre-trained filters can be reused to detect similar patterns on the images in the target dataset. For those images in the target dataset that are similar to the images in the source dataset, the pre-trained filters should not be fine-tuned to prevent from being destructed.

The proposed *filter selection* method allows different images in the target dataset to fine-tune different pre-trained convolutional filters. Consider the i-th layer in a convolutional neural network with input feature map $x_i \in \mathbb{R}^{n_i \times w_i \times h_i}$, where n_i the number of input channels, w_i is the width of the feature map and h_i is the height of the feature map. Given x_i, the convolutional filters in the layer i produce an output $x_{i+1} \in \mathbb{R}^{n_{i+1} \times w_{i+1} \times h_{i+1}}$. This is achieved by applying n_{i+1} convolutional filter $\mathcal{F} \in \mathbb{R}^{n_i \times k \times k}$ on the input feature map. Each filter $\mathcal{F} \in \mathbb{R}^{n_i \times k \times k}$ is applied on x_i to generate one channel of the output. All the n_{i+1} filters in the i-th convolutional layer can be stacked together as a $4D$ tensor.

We denote the $4D$ convolutional filters in the i-th layer as F_i. Given $x_i \in \mathbb{R}^{n_i \times w_i \times h_i}$, $F_i(x_i)$ is the output $x_{i+1} \in \mathbb{R}^{n_{i+1} \times w_{i+1} \times h_{i+1}}$. To allow different images to fine-tune different filters, we initialize a new $4D$ convolutional filter S_i from F_i and *freeze* S_i during training. We use a binary vector $G_i(x_i) \in \{0,1\}^{n_{i+1}}$, called the fine-tuning policy, which is conditioned on the input feature map x_i to decide which filters should be reused and which filters should be fine-tuned. With $G_i(x_i)$, the output of the layer i can be calculated as,

$$x_{i+1} = G_i(x_i) \circ F_i(x_i) + (1 - G_i(x_i)) \circ S_i(x_i) \tag{4.1}$$

where $\circ$ is the Hadamard product. Each element of $G_i(x_i) \in \{0,1\}^{n_{i+1}}$ is multiplied with the corresponding channel of $F_i(x_i) \in \mathbb{R}^{n_{i+1} \times w_{i+1} \times h_{i+1}}$ and $S_i(x_i) \in \mathbb{R}^{n_{i+1} \times w_{i+1} \times h_{i+1}}$. Essentially, the fine-tuning policy $G_i(x_i)$ selects each channel of x_{i+1} either from the output produced by the pre-trained layer S_i (if the corresponding element is 0) or the fine-tuned layer F_i (if the corresponding element is 1). Since $G_i(x_i)$ is conditioned on x_i, different examples in the target dataset can fine-tune different pre-trained convolutional filters in each layer.

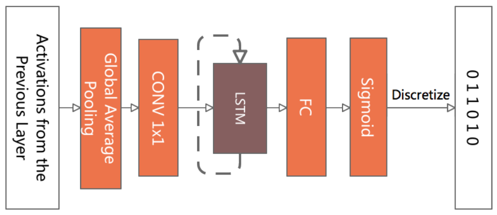


Figure 4.2: The proposed recurrent gated network.

4.2.2 Layerwise Recurrent Gated Network

There are many possible choices to generate the fine-tuning policy $G_i(x_i)$. We adopt a recurrent gated network to both consider the dependencies between different layers and the model size. Figure 4.2 illustrates the proposed recurrent gated network which takes activations from the previous layer as input and discretizes the output of sigmoid function as the fine-tuning policy.

Recurrent neural network (RNN) [GMH13] is a powerful tool for modelling sequential data. The hidden states of the recurrent neural network can remember the correlations between different timestamps. In order to apply the RNN gate, we need to map the input feature map x_i into a low-dimensional space. We first apply a global average pooling on the input feature map x_i and then use a 1×1 convolution which translates the $3D$ input feature map into a one-dimensional embedding vector. The embedding vector is used as the input of the RNN gate to generate the layer-dependent fine-tune policy $G_i(x_i)$. We translate the output of the RNN gate using a linear layer followed by a sigmoid function. To obtain the binary fine-tuned policy $G_i(x_i)$, we use a hard threshold function to discretize the output of the sigmoid function.

The discreteness of $G_i(x_i)$ makes it hard to optimize the network using gradient-based algorithm. To mitigate this problem, we use the *straight-through estimator* which is a widely used technique for training binarized neural network in the field of neural network quantization [Guo18]. In the *straight-through estimator*, during the forward pass we discretize the sigmoid function using a threshold and during backward we compute the gradients with respect to the input of the sigmoid function,

$$\begin{aligned} &\textbf{Forward:} \quad x_b = \begin{cases} 1, & \text{sigmoid}(x) \geq 0.5, \\ 0, & \text{otherwise} \end{cases} \\ &\textbf{Backward:} \quad \frac{\partial E}{\partial x} = \frac{\partial E}{\partial x_b} \end{aligned} \tag{4.2}$$

where E is the loss function. The adoption of the *straight-through estimator* allows us to back-propagate through the discrete output and directly use gradient-based algorithms to end-to-end optimize the network. In the experimental section, we use Long Short-Term Memory (LSTM) [HS97] which has shown to be useful for different sequential tasks. We also compare the proposed recurrent gated network with a CNN-based gated network. The experimental results show that we can achieve higher classification accuracy by explicitly modelling the cross-layer correlations.

4.2.3 Gated Batch Normalization

Batch normalization (BN) [IS15] layer is designed to alleviate the issue of internal covariate shifting of training deep neural networks. In BN layer, we first standardize each feature in a mini-batch, then scale and shift the standardized feature. Let $X_{p,n}$ denote a mini-batch of data, where p is the batch size and n is the feature dimension. BN layer normalizes a feature dimension x_j as below,

$$\hat{x}_j = \frac{x_j - E[X_{.j}]}{\sqrt{Var[X_{.j}]}} \tag{4.3}$$

$$y_j = \gamma_j \hat{x}_j + \beta_j \tag{4.4}$$

The scale parameter γ_j and shift parameter β_j are trained jointly with the network parameters. In convolutional neural networks, we use BN layer after each convolutional layer and apply batch normalization over each channel [IS15]. Each channel has its own scale and shift parameters.

In the standard BN layer, we compute the mean and variance of a particular channel across all the examples in the mini-batch. In AdaFilter, some examples in the mini-batch use the channel produced by the pre-trained filter while the others use the channel produced by the fine-tuned filter. The statistics of the channels produced by the pre-trained filters and the fine-tuned filters are different due to *domain shift*. To consider this fact, we maintain two BN layers, called Gated Batch normalization, which normalize the channels produced by the pre-trained filters and the fine-tuned filters separately. To achieve this, we apply the fine-tuning policy learned by the RNN gate on the output of the BN layers to select the normalized channels,

$$x_{i+1} = G_i(x_i) \circ BN_1(x_{i+1}) + (1 - G_i(x_i)) \circ BN_2(x_{i+1}) \tag{4.5}$$

where BN_1 and BN_2 denote the BN layer for the pre-trained filters and the fine-tuned filters separately and $\circ$ is the Hadamard product. In this way, we can deal with the case when the target dataset and the source dataset have very different domain distribution by adjusting the corresponding shift and scale parameters.

4.2.4 Discussion

The design of the proposed AdaFilter mitigates two issues brought by the standard fine-tuning. Since the fine-tuning policy is conditioned on the activations of the previous layer, different images can fine-tune different pre-trained filters. The images in the target dataset which are similar to the source dataset can reuse more filters from the pre-trained model to allow better *knowledge transfer*. On the other hand, while the number of parameters compared with standard fine-tuning increase by a factor of 2.2x with AdaFilter, the trainable parameters for a particular image are much fewer than the standard fine-tuning due to the reuse of the pre-trained filters. This alleviates the issue of *overfitting* which is critical if the target dataset is much smaller than the source dataset.

All the proposed modules are differentiable which allows us to use gradient-based algorithm to end-to-end optimize the network. During test time, the effective number of filters for a particular test example is equal to a standard fine-tuned model, thus AdaFilter has similar test time compared with the standard fine-tuning.

Table 4.1: Datasets used to evaluate AdaFilter against other fine-tuning baselines.

Dataset	**Training**	**Evaluation**	**Classes**
Stanford Dogs	12000	8580	120
UCF-101	7629	1908	101
Aircraft	3334	3333	100
Caltech 256 - 30	7680	5120	256
Caltech 256 - 60	15360	5120	256
MIT Indoors	5360	1340	67
Omniglot	19476	6492	1623

Table 4.2: The results of AdaFilter and all the baselines.

Method	Stanford-Dogs	UCF-101	Aircraft	Caltech256-30	Caltech256-60	MIT Indoors	Omniglot
Standard Fine-tuning	77.47%	73.10%	52.59%	78.09%	82.25%	76.42%	87.06%
Fine-tuning half	79.61%	76.43%	53.61%	78.86%	82.55%	76.94%	87.29%
Random Policy	81.84%	75.15%	54.15%	79.90%	83.35%	76.71%	85.78%
L2-SP	79.69%	74.33%	**56.52%**	79.33%	82.89%	76.41%	86.92%
AdaFilter	**82.44%**	**76.99%**	55.41%	**80.62%**	**84.31%**	**77.53%**	**87.46%**

4.3 Experimental Settings

4.3.1 Datasets

We compare the proposed AdaFilter method with other fine-tuning and regularization methods on 7 public image classification datasets coming from different domains: Stanford dogs [KJYFF11], Aircraft [MRK$^+$13], MIT Indoors [QT09], UCF-101 [BFG$^+$16], Omniglot [LST15], Caltech 256 - 30 and Caltech 256 - 60 [GHP07]. For Caltech 256 - x (x = 30 or 60), there are x training examples for each class. The statistics of the datasets are listed in Table 4.1. Performance is measured by classification accuracy on the evaluation set.

Baselines

We consider the following fine-tuning variants and regularization techniques for fine-tuning in the experiments,

- Standard Fine-tuning: this is the standard fine-tuning method which fine-tunes all the parameters of the pre-trained model.
- Fine-tuning half: only fine-tune second half of the layers of the pre-trained model and freeze the first half of layers.
- Random Policy: use AdaFilter with a random fine-tuning policy. This shows the effectiveness of fine-tuning policy learned by the recurrent gated network.
- L2-SP [LGD18]: this is a recently proposed regularization method for fine-tuning which explicitly adds regularization terms in the loss function to encourage the fine-tuned model to be similar to the pre-trained model.

4.3.2 Pretrained Model

To compare AdaFilter with each baseline. We use ResNet-50 which is pre-trained on ImageNet. The ResNet-50 starts with a convolutional layer followed by 16 blocks with residual connection. Each block contains three convolutional layers and are distributed into 4 macro blocks (i.e, [3, 4, 6, 3]) with downsampling layers in between. The ResNet-50 ends with an average pooling layer followed by a fully connected layer. For a fair

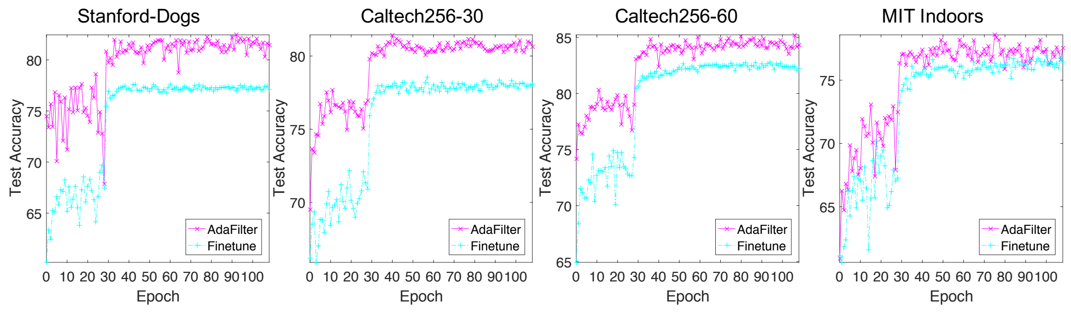


Figure 4.3: The test accuracy curve of AdaFilter and the standard fine-tuning on Stanford-Dogs, Caltech256-30, Caltech256-60 and MIT Indoors.

comparison with each baseline, we use the pre-trained model from Pytorch which has a classification accuracy of 75.15% on ImageNet.

4.3.3 Implementation Details

Our implementation is based on Pytorch. All methods are trained on 2 NVIDIA Titan Xp GPUs. We use SGD with momentum as the optimizer. The initial learning rate is 0.01 for the classification network and the initial learning rate for the recurrent gated network is 0.1. The momentum rate is 0.9 for both classification network and recurrent gated network. The batch size is 64. We train the network with a total of 110 epochs. The learning rate decays three times at the 30th, 60th and 90th epoch respectively.

4.4 Results and Analysis

4.4.1 AdaFilter vs Baselines

We show the results of AdaFilter and all the baselines in Table 4.2. AdaFilter achieves the best results on 6 out of 7 datasets. It outperforms the standard fine-tuning on all the datasets. Compared with the standard fine-tuning, AdaFilter can reduce the classification error by up to 5%. This validates our claim that by exploiting the idea of per-example filter fine-tuning, we can greatly boost the performance of the standard fine-tuning method by mitigating its drawbacks. While fine-tuning half of the layers generally performs better than the standard fine-tuning, it still performs worse than AdaFilter since it still applies the same fine-tuning policy for all the images which ignores the similarity between the target task and the source task.

Compared with Random policy and L2-SP, AdaFilter obtains higher accuracy by learning optimal fine-tuning policy for each image in the target dataset via the recurrent gated network. The results reveal that by carefully choosing different fine-tuning for different images in the target dataset, we can achieve better transfer learning results. With AdaFilter, we can automatically specialize the fine-tuning policy for each test example which cannot be done manually due to the huge search space.

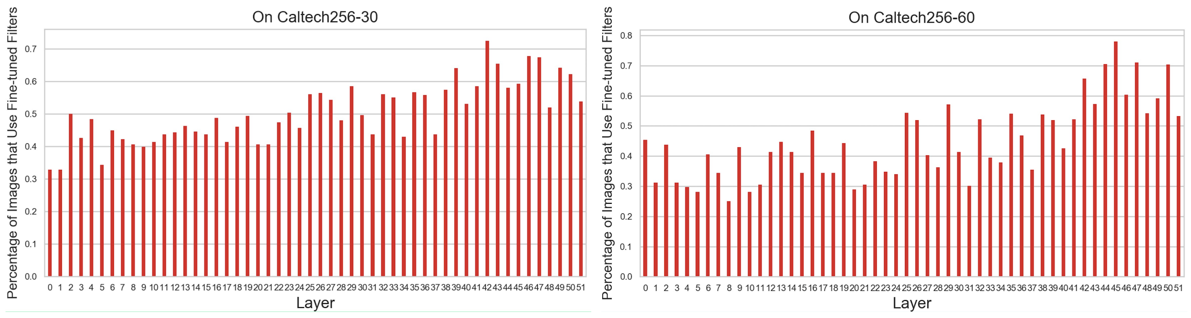


Figure 4.4: The visualization of fine-tuning policies on Caltech256-30 and Caltech256-60.

4.4.2 Test Accuracy Curve

We show the test accuracy curve on four benchmark datasets in Figure 4.3. We can clearly see that the proposed AdaFilter consistently achieves higher accuracy than the standard fine-tune method across all the datasets. For example, after training for one epoch, AdaFilter reaches a test accuracy of 71.96% on the Stanford Dogs dataset while the standard fine-tuning method only achieves 54.69%. Similar behavior is also observed on other datasets. The fact that AdaFilter can reach the same accuracy level as standard fine-tuning with much fewer epochs is of great practical importance since it can reduce the training time on new tasks.

4.4.3 Visualization of Policies

In this section, we show the fine-tuning policies learned by the recurrent gated network on Caltech256-30 and Caltech256-60 in Figure 4.4. The x-axis denotes the layers in the ResNet-50. The y-axis denotes the percentage of images in the evaluation set that use the fine-tuned filters in the corresponding layer. As we can see, there is a strong tendency for images to use the pre-trained filters in the initial layers while fine-tuning more filters at the higher layers of the network. This is intuitive since the filters in the initial layers can be reused on the target dataset to extract visual patterns (e.g., edges and corners). The higher layers are mostly task-specific which need to be adjusted further for the target task [LEN08]. We also note that the policy distribution is varied across different datasets, this suggests that for different datasets it is preferable to design different fine-tuning strategies.

Table 4.3: Comparison of Gated BN and the standard BN

Dataset	Stanford-Dogs	Aircraft	Omniglot	UCF-101	MIT Indoors	Caltech-30	Caltech-60
Gated BN	82.44%	55.41%	87.46%	76.99%	77.53%	80.06%	84.31%
Standard BN	82.02%	54.33%	87.27%	76.02%	77.01%	79.84%	83.84%

4.4.4 Ablation Study

Gated BN vs Standard BN

In this section, an ablation study is performed to demonstrate the effectiveness of the proposed *gated batch normalization*. We compare gated batch normalization (Gated BN) against the standard batch normalization (Standard BN). In Gated BN, we normalize the channels produced by the pre-trained filters and fine-tuned filters separately as in Equation 4.5. In standard batch normalization, we use one batch normalization layer to normalize each channel across a mini-batch,

$$x_{i+1} = BN(x_{i+1}) \tag{4.6}$$

Table 4.3 shows the results of the Gated BN and the standard BN on all the datasets. Clearly, Gated BN can achieve higher accuracy by normalizing the channels produced by the pre-trained filters and fine-tuned filters separately. This suggests that although we can reuse the pre-trained filters on the target dataset, it is still important to consider the difference of the domain distributions between the target task and the source task.

Table 4.4: Comparison of recurrent gated network and a CNN-based policy network. The "RNN-based" means the recurrent gated network.

Dataset	Stanford-Dogs	Aircraft	Omniglot	UCF-101	MIT Indoors	Caltech-30	Caltech-60
RNN-based	82.44%	55.41%	87.46%	76.99%	77.53%	80.06%	84.31%
CNN-based	83.05%	54.63%	87.04%	76.33%	77.46%	80.25%	83.41%

Recurrent Gated Network vs CNN-based Gated Network

In this section, we perform an ablation study to show the effectiveness of the proposed recurrent gated network. We compare the recurrent gated network against a CNN-based policy network. The CNN-based policy network is based on ResNet-18 which receives images as input and predicts the fine-tuning policy for all the filters at once. In the CNN-based model, the input image is directly used as the input for the CNN. The output of the CNN is a list of fully connected layers (one for each output feature map in the original backbone network) followed by sigmoid activation function. We show the results of the recurrent gated network and CNN-based policy network in Table 4.4. Recurrent gated network performs better than the CNN-based policy network on most of the datasets by explicitly considering the dependency between layers. More importantly, predicting the policy layerwisely and reusing the hidden states of the recurrent gated network can greatly reduce the number of parameters. The lightweight design of the recurrent gated network is also faster to train than the CNN-based alternative.

4.5 Conclusion

In this chapter, we propose a deep transfer learning method, called AdaFilter, which adaptively fine-tunes the convolutional filters in a pre-trained model. With the proposed *filter selection*, *recurrent gated network* and *gated batch normalization* techniques, AdaFilter allows different images in the target dataset to fine-tune different pre-trained filters to enable better knowledge transfer. We validate our methods on seven publicly available datasets and show that AdaFilter outperforms the standard fine-tuning on all the datasets. The proposed method can also be extended to lifelong learning by modelling the tasks sequentially. In next chapter, we consider a different scenario where the target domain only has few examples per category which is much more challenging.

This chapter contains material from "AdaFilter: Adaptive Filter Fine-tuning for Deep Transfer Learning", by Yunhui Guo, Yandong Li, Liqiang Wang, Tajana Rosing, which appears in *The 34th AAAI Conference on Artificial Intelligence (AAAI)*, 2020. The dissertation author was the primary investigator and author of this paper.

Chapter 5

Cross-domain Few-shot Learning

5.1 Introduction

In last chapter we consider learning across two domains where the target domain may have a large number of examples. However, in real applications, it is often costly to acquire labels for some categories. The problem of learning how to categorize classes with very few training examples is referred to as “few-shot learning”, and has been the topic of a large body of recent work [LFP06, RL16, VBL$^+$16, FAL17, SSZ17, CLK$^+$19, SYZ$^+$18]. Few-shot learning is typically composed of the following two stages: meta-learning and meta-testing. In the meta-learning stage, there is an abundance of base category classes on which a system can be trained to learn well under conditions of few-examples within that particular domain. In the meta-testing stage, a set of novel classes consisting of very few examples per class is used to adapt and evaluate the trained model. However, recent work [CLK$^+$19] points out that meta-learning based few-shot learning algorithms underperform compared to traditional "*pre-training and fine-tuning*" when there exists a large *domain shift* between base classes and novel classes. This is a major issue, as the *domain shift* problem occurs commonly in practice: by the nature of the problem, it is hard to collect data from the same domain for many few-shot classification tasks. This scenario is referred to as *cross-domain few-shot learning*, to distinguish it from the conventional few-shot learning setting.

Although benchmarks for conventional few-shot learning are well established, the lack of standard cross-domain few-shot learning benchmarks hinder the development of new few-shot learning algorithms. Therefore, to fill this gap, we propose the cross-domain few-shot learning (CD-FSL) benchmark (Fig. 5.1). The proposed benchmark covers a variety of image domains with varying levels of similarity with ImageNet, including agriculture (most similar), satellite (less similar), dermatology (even less similar), and radiological images (least similar). The performance of existing state-of-art meta-learning methods is then evaluated on the proposed benchmark, where results reveal that these meta-learning methods perform significantly worse than the standard "*pre-training and fine-tuning*" approach. Subsequently, variants of single model fine-tuning techniques are also evaluated, where results demonstrate that no individual method dominates performance across the benchmark, and the relative performance gain with increasing number of shots is greater with transfer methods than compared to meta-learning. In addition, performance across methods also positively correlates with dataset similarity to ImageNet. Further experiments transferring knowledge from multiple pretrained models, all from disjoint domains than the evaluation benchmark, demonstrates best performance.

In summary, the contributions of this section are itemized as follows:

- We establish a new benchmark for cross-domain few-shot learning, consisting of images from a diversity of domains with varying similarity to ImageNet, and lacking data for meta-learning.

- We extensively evaluate the performance of current meta-learning methods and variants of fine-tuning. The results show the following observations for CD-FSL: 1) meta-learning underperforms compared to fine-tuning, 2) accuracy gain with additional data is increased for fine-tuning versus meta-learning, 3) no individual fine-tuning method dominates performance versus the others across the benchmark, and 4) a general positive correlation between accuracy and dataset similarity to ImageNet exists.

- We propose *Incremental Multi-model Selection*, a method which integrates multiple pretrained models for cross-domain few-shot learning, and demonstrates best average performance on the new benchmark.

5.2 Related Work

Few-shot learning [LST15, VBL^{+}16, LSGT11] is an increasingly important topic in machine learning. Current few-shot leaning algorithms can be roughly classified into two threads. One thread includes meta-learning methods which aim to learn models that can be quickly adapted using a few examples [VBL$^+$16, FAL17, SSZ17, SYZ$^+$18, LMRS19]. MatchingNet [VBL$^+$16] learns an embedding that can map an unlabelled example to its label using a small number of labelled examples, while MAML [FAL17] aims at learning good initialization parameters that can be quickly adapted to a new task. In ProtoNet [SSZ17], the goal is to learn a metric space in which classification can be conducted by calculating distances to prototype representations of each class. RelationNet [SYZ$^+$18] targets learning a deep distance metric to compare a small number of images. More recently, MetaOpt [LMRS19] learns feature embeddings that can generalize well under a linear classification rule for novel categories.

Another line of few-shot learning algorithms is based on the idea of reusing features learned from the base classes for the novel classes, i.e., *transfer learning* [PY09]. Transfer learning with deep neural networks is conducted mainly by fine-tuning, which adjusts a pretrained model from a source task to a target task. Yosinski et al. [YCBL14] conducted extensive experiments to investigate the transfer utility of pretrained deep neural networks. In [KSL18], the authors investigated whether higher performing ImageNet models transfer better to new tasks. Ge et al. [GY17] proposed a selective joint fine-tuning method for improving the performance of models with a limited amount training data. In [GSK$^+$19], the authors proposed an adaptive fine-tuning scheme to decide which layers of the pretrained network should be fine-tuned.

Few-shot Learning Benchmarks. Current few-shot learning research assumes base classes and novel classes are from the same domain. The common benchmarks for evaluation are *miniImageNet* [VBL$^+$16], *CUB* [WBW$^+$11], *Omniglot* [LSGT11], *CIFAR-*

FS [BHTV18] and *tieredImageNet* [RTR$^+$18]. In [TZD$^+$19], the authors proposed *Meta-Dataset*, which is a new benchmark for training and evaluating few-shot learning algorithms. However, the included datasets are limited to natural images similar to previous benchmarks. The evaluation also follows the standard setting, that is, both the base classes and novel classes are from the same domain. Arguably, the current few-shot learning benchmarks do not reflect the reality of few-shot learning applications where meta-learning data in domain is commonly not available.

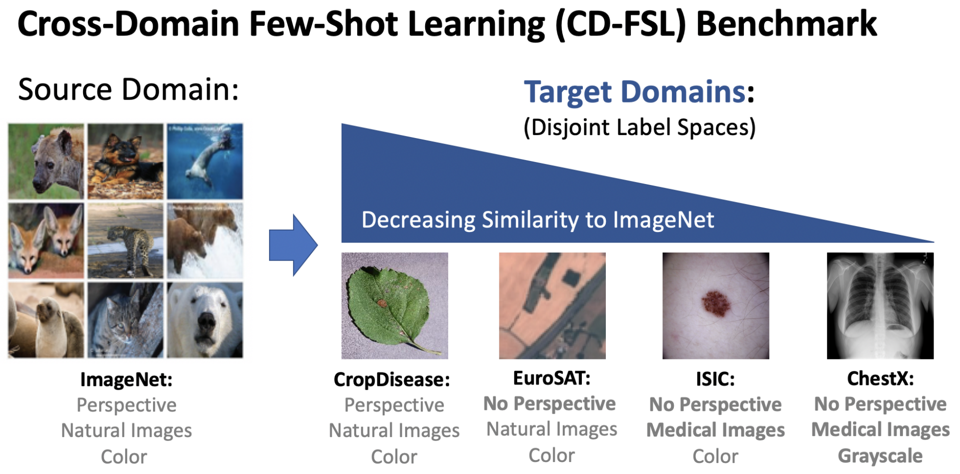

Cross-Domain Few-Shot Learning (CD-FSL) Benchmark
Source Domain:
Target Domains:
(Disjoint Label Spaces)
Decreasing Similarity to ImageNet
ImageNet: Perspective Natural Images Color
CropDisease: Perspective Natural Images Color
EuroSAT: No Perspective Natural Images Color
ISIC: No Perspective Medical Images Color
ChestX: No Perspective Medical Images Grayscale

Figure 5.1: The cross-domain few-shot learning (CD-DSL) benchmark. ImageNet is used for source training, and domains of varying dissimilarity from ImageNet are used for target evaluation.

5.3 Proposed Benchmark

In this section, we introduce the proposed cross-domain few-shot learning benchmark. The proposed benchmark includes data from the *CropDiseases* [MHS16], *EuroSAT* [HBDB19], *ISIC2018* [TRK18, CRT$^+$19], and *ChestX* [WPL$^+$17] datasets, which covers plant disease images, satellite images, dermoscopic images of skin lesions, and X-ray images, respectively. The selected datasets reflect real-world use cases for few-shot learning since collecting enough examples from above domains is often difficult, expensive, or in some cases not possible. In addition, they demonstrate the following spectrum of readily quantifiable domain shifts from ImageNet [DDS+09]: 1) *CropDiseases* images are most similar as they include perspective color images of natural elements, but are more specialized than anything available in ImageNet, 2) *EuroSAT* images are less similar as they have lost perspective distortion, but are still color images of natural scenes, 3) *ISIC2018* images are even less similar as they have lost perspective distortion and no longer represent natural scenes, and 4) *ChestX* images are the most dissimilar as they have lost perspective distortion, all color, and do not represent natural scenes. Example images from ImageNet and the proposed benchmark datasets are shown in Figure 5.1.

In practice, having a few-shot learning model trained on a source domain such as ImageNet [DDS+09] that can generalize to domains such as these, is highly desirable, as it enables effective learning for rare categories in new domains, which has previously not been studied in detail.

5.4 Methods for Few-Shot Learning

In this section, we describe the existing prevailing few-shot learning algorithms that will be evaluated on our proposed benchmark. We categorize existing few-shot learning algorithms into meta-learning based methods and transfer learning based methods. For

Table 5.1: The results of meta-learning methods on the proposed benchmark.

Methods	ChestX			ISIC		
	5-way 5-shot	5-way 20-shot	5-way 50-shot	5-way 5-shot	5-way 20-shot	5-way 50-shot
MatchingNet	22.40% ± 0.85%	23.61% ± 0.86%	22.12% ± 0.88%	36.74% ± 0.53%	45.72% ± 0.53%	54.58% ± 0.65%
MAML	23.48% ± 0.96%	27.53% ± 0.43%	-	40.13% ± 0.58%	52.36% ± 0.57%	-
ProtoNet	24.05% ± 1.01%	28.21% ± 1.15%	29.32% ± 1.12%	39.57% ± 0.57%	49.50% ± 0.55%	51.99% ± 0.52%
RelationNet	22.96% ± 0.88%	26.63% ± 0.92%	28.45% ± 1.20%	39.41% ± 0.58%	41.77% ± 0.49%	49.32% ± 0.51%
MetaOpt	22.53% ± 0.91%	25.53% ± 1.02%	29.35% ± 0.99%	36.28% ± 0.50%	49.42% ± 0.60%	54.80% ± 0.54%

Methods	EuroSAT			CropDiseases		
	5-way 5-shot	5-way 20-shot	5-way 50-shot	5-way 5-shot	5-way 20-shot	5-way 50-shot
MatchingNet	64.45% ± 0.63%	77.10% ± 0.57%	54.44% ± 0.67%	66.39% ± 0.78%	76.38% ± 0.67%	58.53% ± 0.73%
MAML	71.70% ± 0.72%	81.95% ± 0.55%	-	78.05% ± 0.68%	89.75% ± 0.42%	-
ProtoNet	73.29% ± 0.71%	82.27% ± 0.57%	80.48% ± 0.57%	79.72% ± 0.67%	88.15% ± 0.51%	90.81% ± 0.43%
RelationNet	61.31% ± 0.72%	74.43% ± 0.66%	74.91% ± 0.58%	68.99% ± 0.75%	80.45% ± 0.64%	85.08% ± 0.53%
MetaOpt	64.44% ± 0.73%	79.19% ± 0.62%	83.62% ± 0.58%	68.41% ± 0.73%	82.89% ± 0.54%	91.76% ± 0.38%

transfer learning methods, we also consider the effect of different type of classifiers. Finally, we show that transferring from multiple models pretrained on different datasets from the same source domain can generally boost performance.

5.4.1 Meta-learning Based Methods

Meta-learning [FAL17, RL16], or learning to learn, aims at learning task-agnostic knowledge in order to efficiently learn on new tasks. Each task $\mathcal{T}_i$ is assumed to be drawn from a fixed distribution, $\mathcal{T}_i \sim P(\mathcal{T})$. Specially, in few-shot learning, each task $\mathcal{T}_i$ is a small dataset $D_i := \{x_j, y_j\}_{j=1}^{K \times N}$. $P_s(\mathcal{T})$ and $P_t(\mathcal{T})$ are used to denote the task distribution of the source (base) classes data and target (novel) classes data respectively. During the meta-training stage, the model is trained on T tasks $\{\mathcal{T}_i\}_{i=1}^{T}$ which are sampled independently from $P_s(\mathcal{T})$. During the meta-testing stage, the model is expected to be quickly adapted to a new task $T_j \sim P_t(\mathcal{T})$.

Meta-learning methods differ in their way of learning the parameter of the initial model f_θ on the base classes data. In MatchingNet [VBL+16], the goal is to learn a model f_θ that can map an unlabelled example $\hat{x}$ to its label $\hat{y}$ using a small labelled set $D_i := \{x_j, y_j\}_{j=1}^{K\times N}$ as $\hat{y} = \sum_{j=1}^{K\times N} a_\theta(\hat{x}, x_j) y_j$, where a_θ is an attention kernel which leverages f_θ to compute the distance between the unlabelled example $\hat{x}$ and the labelled example x_j, and y_j is the one-hot representation of the label. In contrast, MAML [FAL17] aims at learning an initial parameter θ that can be quickly adapted to a new task. This is achieved by updating the model parameter via a two-stage optimization process. ProtoNet [SSZ17] represents each class k with the mean vector of embedded support examples as $c_k = \frac{1}{N}\sum_{j=1}^{N} f_\theta(x_j)$. Classification is then conducted by calculating distance of the example to the prototype representations of each class. In RelationNet [SYZ$^+$18] the metric of the nearest neighbor classifier is meta-learned using a Siamese Networks trained for optimal comparison between query and support samples. More recently, MetaOpt [LMRS19] employs convex base learners and aims at learning feature embeddings that generalize well under a linear classification rule for novel categories. All the existing meta-learning methods implicitly assume that $P_s(\mathcal{T}) = P_t(\mathcal{T})$ so the task-agnostic knowledge learned in the meta-training stage can be leveraged for fast learning on novel classes. However, in cross-domain few-shot learning $P_s(\mathcal{T}) \neq P_t(\mathcal{T})$ which poses severe challenges for current meta-learning methods.

5.4.2 Transfer Learning Based Methods

An alternative way to tackle the problem of few-shot learning is based on transfer learning, where an initial model f_θ is trained on the base classes data in a standard supervised learning way and reused on the novel classes. There are several options to realize

Table 5.2: The results of different variants of single model fine-tuning on the proposed benchmark.

Methods	**ChestX**			**ISIC**		
	5-way 5-shot	5-way 20-shot	5-way 50-shot	5-way 5-shot	5-way 20-shot	5-way 50-shot
Random	21.80% ± 1.03%	25.69% ± 0.95%	26.19% ± 0.94%	37.91% ± 1.39%	47.24% ± 1.50%	50.85% ± 1.37%
Fixed	25.35% ± 0.96%	30.83% ± 1.05%	36.04% ± 0.46%	43.56% ± 0.60%	52.78% ± 0.58%	57.34% ± 0.56%
Fine-tuning	25.97% ± 0.41%	31.32% ± 0.45%	35.49% ± 0.45%	48.11% ± 0.64%	59.31% ± 0.48%	66.48% ± 0.56%
Ft Last-1	25.96% ± 0.46%	31.63% ± 0.49%	37.03% ± 0.50%	47.20% ± 0.45%	59.95% ± 0.45%	65.04% ± 0.47%
Ft Last-2	26.79% ± 0.59%	30.95% ± 0.61%	36.24% ± 0.62%	47.64% ± 0.44%	59.87% ± 0.35%	66.07% ± 0.45%
Ft Last-3	25.17% ± 0.56%	30.92% ± 0.89%	37.27% ± 0.64%	48.05% ± 0.55%	60.20% ± 0.33%	66.21% ± 0.52%
Transductive Ft	26.09% ± 0.96%	31.01% ± 0.59%	36.79% ± 0.53%	49.68% ± 0.36%	61.09% ± 0.44%	67.20% ± 0.59%

Methods	**EuroSAT**			**CropDiseases**		
	5-way 5-shot	5-way 20-shot	5-way 50-shot	5-way 5-shot	5-way 20-shot	5-way 50-shot
Random	58.00% ± 2.01%	68.93% ± 1.47%	71.65% ± 1.47%	69.68% ± 1.72%	83.41% ± 1.25%	86.56% ± 1.42%
Fixed	75.69% ± 0.66%	84.13% ± 0.52%	86.62% ± 0.47%	87.48% ± 0.58%	94.45% ± 0.36%	96.62% ± 0.25%
Fine-tuning	79.08% ± 0.61%	87.64% ± 0.47%	90.89% ± 0.36%	89.25% ± 0.51%	95.51% ± 0.31%	97.68% ± 0.21%
Ft Last-1	80.45% ± 0.54%	87.92% ± 0.44%	91.41% ± 0.46%	88.72% ± 0.53%	95.76% ± 0.65%	97.87% ± 0.48%
Ft Last-2	79.57% ± 0.51%	87.67% ± 0.46%	90.93% ± 0.45%	88.07% ± 0.56%	95.68% ± 0.76%	97.64% ± 0.59%
Ft Last-3	78.04% ± 0.77%	87.52% ± 0.53%	90.83% ± 0.42%	89.11% ± 0.47%	95.31% ± 0.85%	97.45% ± 0.46%
Transductive Ft	81.76% ± 0.48%	87.97% ± 0.42%	92.00% ± 0.56%	90.64% ± 0.54%	95.91% ± 0.72%	97.48% ± 0.56%

the idea of transfer learning for few-shot learning. Previous works on transfer learning for few-shot learning [SSZ17, CLK$^+$19] simply freeze the pretrained model and use it as a fixed feature extractor. While it was pointed out that fine-tuning the pretrained model on the novel classes data would lead to overfitting in the limited-data regime [SSZ17, SYZ$^+$18], our results show that the conclusion does not hold in cross-domain few-shot learning.

Single Model Methods

In this section, we extensively evaluate the following commonly variants of single model fine-tuning:

- *Fixed feature extractor (Fixed)*: simply leverage the pretrained model as a fixed feature extractor.
- *Fine-tuning*: *Fine-tuning* adjusts the pretrained parameters on the new task with standard supervised learning.

Table 5.3: The results of varying the classifier for fine-tuning on the proposed benchmark.

Methods	**ChestX**			**ISIC**		
	5-way 5-shot	5-way 20-shot	5-way 50-shot	5-way 5-shot	5-way 20-shot	5-way 50-shot
Linear	25.97% ± 0.41%	31.32% ± 0.45%	35.49% ± 0.45%	48.11% ± 0.64%	59.31% ± 0.48%	66.48% ± 0.56%
Mean-centroid	26.31% ± 0.42%	30.41% ± 0.46%	34.68% ± 0.46%	47.16% ± 0.54%	56.40% ± 0.53%	61.57% ± 0.66%
Cosine-similarity	26.95% ± 0.44%	32.07% ± 0.55%	34.76% ± 0.55%	48.01% ± 0.49%	58.13% ± 0.48%	62.03% ± 0.52%

Methods	**EuroSAT**			**CropDiseases**		
	5-way 5-shot	5-way 20-shot	5-way 50-shot	5-way 5-shot	5-way 20-shot	5-way 50-shot
Linear	79.08% ± 0.61%	87.64% ± 0.47%	91.34% ± 0.37%	89.25% ± 0.51%	95.51% ± 0.31%	97.68% ± 0.21%
Mean-centroid	82.21% ± 0.49%	87.62% ± 0.34%	88.24% ± 0.29%	87.61% ± 0.47%	93.87% ± 0.68%	94.77% ± 0.34%
Cosine-similarity	81.37% ± 1.54%	86.83% ± 0.43%	88.83% ± 0.38%	89.15% ± 0.51%	93.96% ± 0.46%	94.27% ± 0.41%

- *Fine-tuning last-k (Ft last-k)*: only the last k layers of the pretrained model are optimized for the new task. In the paper, we consider Fine-tuning last-1, Fine-tuning last-2, Fine-tuning last-3.

- *Transductive fine-tuning (Transductive Ft)*: in transductive fine-tuning, the statistics of the query images are used via batch normalization [NAS18].

In addition, we compare these single model transfer learning techniques against a baseline of an embedding formed by a randomly initialized network to contrast against a fixed feature vector that has no pre-training. All the variants of single model fine-tuning are based on linear classifier but differ in their approach to fine-tune the single model feature extractor.

Another line of work for few-shot learning uses a broader variety of classifiers for transfer learning. For example, recent works show that mean-centroid classifier and cosine-similarity based classifier are more effective than linear classifier for few-shot learning [MVPC13, CLK$^+$19]. Therefore we study these two variations as well.

Mean-centroid classifier. The mean-centroid classifier is inspired from ProtoNet [SSZ17]. Given the pretrained model f_θ and a support set $S = \{x_i, y_i\}_{i=1}^{K \times N}$, where K is the number of novel classes and N is the number of images per class. The class prototypes are computed in the same way as in ProtoNet. Then the likelihood of an unlabelled example $\hat{x}$ belongs to class k is computed as,

$$p(y = k|\hat{x}) = \frac{\exp(-d(f_\theta, c_k))}{\sum_{l=1}^{K} \exp(-d(f_\theta, c_l))} \tag{5.1}$$

where $d()$ is a distance function. In the experiments, we use negative cosine similarity. Other distance functions such as Euclidean distance can also be used. Different from ProtoNet, f_θ is pretrained on the base classes data in a standard supervised learning way.

Cosine-similarity based classifier. In cosine-similarity based classifier, instead of directly computing the class prototypes using the pretrained model, each class k is represented as a d-dimension weight vector $\mathbf{w}_k$ which is initialized randomly. For each unlabeled example $\hat{x}_i$, the cosine similarity to each weight vector is computed as $c_{i,k} = \frac{f_\theta(\hat{x}_i)^T \mathbf{w}_k}{\|f_\theta(\hat{x}_i)\| \|\mathbf{w}_k\|}$. The predictive probability of the example $\hat{x}_i$ belongs to class k is computed by normalizing the cosine similarity with a softmax function. Intuitively, the weight vector $\mathbf{w}_k$ can be thought as the prototype of class k.

Transfer from Multiple Pretrained Models

Traditional transfer learning methods for few-shot learning only consider using one pretrained model for feature extraction. In this section, we propose a novel method that

Table 5.4: The results of using all embeddings, and the proposed *Incremental Multi-model Selection* (IMS-f) based on fine-tuned pretrained models on the proposed benchmark.

Methods	ChestX			ISIC		
	5-way 5-shot	5-way 20-shot	5-way 50-shot	5-way 5-shot	5-way 20-shot	5-way 50-shot
All embeddings	26.74% ± 0.42%	32.77% ± 0.47%	38.07% ± 0.50%	46.86% ± 0.60%	58.57% ± 0.59%	66.04% ± 0.56%
IMS-f	25.50% ± 0.45%	31.49% ± 0.47%	36.40% ± 0.50%	45.84% ± 0.62%	61.50% ± 0.58%	68.64% ± 0.53%

Methods	EuroSAT			CropDiseases		
	5-way 5-shot	5-way 20-shot	5-way 50-shot	5-way 5-shot	5-way 20-shot	5-way 50-shot
All embeddings	81.29% ± 0.62%	89.90% ± 0.41%	92.76% ± 0.34%	90.82% ± 0.48%	96.64% ± 0.25%	98.14% ± 0.18%
IMS-f	83.56% ± 0.59%	91.22% ± 0.38%	93.85% ± 0.30%	90.66% ± 0.48%	97.18% ± 0.24%	98.43% ± 0.16%

Algorithm 1 *Incremental Multi-model Selection.* S is the support set. Assume there is a library of C pretrained models $\{M_c\}_{c=1}^{C}$. Each model has L layers and l is used to denote one particular layer. Let $CW(S, I)$ be a function which returns the average cross-validation error given a dataset S and a set of layers I which are used to generate feature vector.

```
First stage:
$I_1 = \{\}$
for $c = 1 \rightarrow C$ do
    $min\_loss$ = -1
    $best\_l = None$
    for $l = 1 \rightarrow L$ do
        if $CW(S, \{l\}) < min\_loss$ then
            $best\_l = l$
            $min\_loss = CW(S, \{l\})$
        end if
    end for
    $I_l = I_l \bigcup best\_l$
end for
Second stage:
$I = \{\}$
$min\_loss$ = -1
for each $l$ in $I_1$ do
    if $CW(S, I \bigcup l) < min\_loss$ then
        $min\_loss = CW(S, I \bigcup l)$
        $I = I \bigcup l$
    end if
end for
Concatenate the feature vectors generated by the layers in $I$ and train a linear classifier.
```

utilizes multiple models pretrained on different source datasets from similar domains as ImageNet, where all source datasets are still disjoint from the target datasets, for cross-domain few-shot learning. The intuition is that by using models pretrained on different datasets, we can obtain more diverse and richer visual features. Unlike previous works on ensemble methods for few-shot learning [DSM19] that train diverse models on the same source dataset, the proposed method requires no change to how models are trained and is an off-the-shelf solution to leverage existing pretrained models for cross-domain few-shot learning, without requiring access to the source datasets.

Assume we have a library of C pretrained models $\{M_c\}_{c=1}^{C}$ which are trained on various datasets in a standard way. We denote the layers of all pretrained models as a set F. Given a support set $S = \{x_i, y_i\}_{i=1}^{K\times N}$ where $(x_i, y_i) \sim P_t$, our goal is to find a subset I of the layers to generate a feature vector for each example in order to achieve the lowest test error. Mathematically,

$$\operatorname*{arg\,min}_{I\subseteq F} \ {}_{(x,y)\sim P_t}\ell(f_s(T(\{l(x) : l \in I\}), y) \tag{5.2}$$

where ℓ is a loss function, $T()$ is a function which combines a set of feature vectors, l is one particular layer in the set I and f_s is a linear classifier. Practically, for feature vectors l coming from inner layers which are three-dimensional, we convert them to one-dimensional vectors by using Global Average Pooling. Since Eq. 5.2 is intractable generally, we instead adopt a two-stage greedy selection method, called *Incremental Multi-model Selection*, to iteratively find the best subset of layers for a given support S.

In the first stage, for each pretrained model, we a train linear classifier on the feature vector generated by each layer individually and select the corresponding layer which achieves the lowest average error using five-fold cross-validation on the support set S. Essentially, the goal of the first stage is to find the most effective layer of each pretrained model given the task in order to reduce the search space and mitigate risk of overfitting. For convenience, we denote the layers selected in the first selection stage as set I_1. In the second stage, we greedily add the layers in I_1 into the set I following a similar cross-validation procedure. First, we add the layer in I_1 into I which achieves the lowest cross-validation error. Then we iterate over I_1, and add each remaining layer into I if the cross-validation error is reduced when the new layer is added. Finally, we concatenate the feature vector generated by each layer in set I and train the final linear classifier. Please see Algorithm 1 further details.

5.5 Evaluation Setup

For meta-learning methods, we meta-train all meta-learning methods on the base classes of miniImageNet [VBL+16] and meta-test the trained models on each dataset of the proposed benchmark. For transfer learning methods, we train the pretrained model on base classes of miniImageNet. For transferring from multiple pretrained models, we use a maximum of five pretrained models, trained on *miniImagenet*, *CIFAR100* [K+09], *DTD* [CMK+14], *CUB* [WBM+10], *Caltech256* [GHP07], respectively. On all experiments we consider 5-way 5-shot, 5-way 20-shot, 5-way 50-shot. For all cases, the test (query) set has 15 images per class. All experiments are performed with ResNet-10 [HZRS16] for fair comparison. For each evaluation, we use the same 600 randomly sampled few-shot episodes (for consistency), and report the average accuracy and 95% confidence interval.

During the training (meta-training) stage, models used for transfer learning and meta-learning models are both trained for 400 epochs with Adam optimizer. The learning rate is set to 0.001. During testing (meta-testing), both transfer learning methods and those meta-learning methods that require adaptation on the support set of the test episodes (MAML, RelationNet, etc.) use SGD with momentum. The learning rate is 0.01 and the momentum rate is 0.9. All variants of fine-tuning methods are trained for 100 epochs. In the training or meta-training stage, we apply standard data augmentation including random crop, random flip, and color jitter.

5.6 Experimental Results

Results are discussed according to method categories as described in Section 5.4. First, results from state-of-art meta-learning methods are presented in Section 5.6.1. Next, transfer learning is evaluated and analyzed in Section 5.6.2, including single model transfer in Section 5.6.2, followed by multi-model transfer in Section 5.6.2. Finally, a succinct best-in-category comparison is presented in Section 5.6.3.

5.6.1 Meta-learning Based Results

Table 5.1 show the results on the proposed benchmark of meta-learning, for each dataset, method, and shot level in the benchmark. Across all datasets and shot levels, the average accuracies (and 95% confidence internals) are 50.21% (0.70) for MatchingNet, 38.75% (0.41) for MAML, 59.78% (0.70) for ProtoNet, 54.48% (0.71) for RelationNet, and 57.35% (0.68) for MetaOpt. The performance of MAML was impacted by its inability to scale to larger shot levels due to memory overflow.

What is immediately apparent from Table 5.1, is that performance in general strongly positively correlates to the dataset's similarity to ImageNet, confirming that the benchmark's intentional design allows us to investigate few-shot learning in a spectrum of cross-domain difficulties.

5.6.2 Transfer Learning Based Results

Single model results

Table 5.2 show the results on the proposed benchmark of various single model transfer learning methods. Across all datasets and shot levels, the average accuracies (and 95% confidence internals) are 53.99% (1.38) for random embedding, 64.24 (0.59) for fixed feature embedding, 67.23% (0.46) for fine-tuning, 67.41% (0.49) for fine-tuning the last 1 layer, 67.26% (0.53) for fine-tuning the last 2 layers, 67.17% (0.58) for fine-tuning the last 3 layers, and 68.14% (0.56) for transductive fine-tuning. From these results, several observations can be made. The first observation is that, although meta-learning methods have been previously shown to achieve higher performance than transfer learning in the standard few-shot learning setting [VBL+16, CLK+19], in the cross-domain few-shot learning setting this situation is reversed: meta-learning methods significantly underperform simple fine-tuning methods. In fact, *MatchingNet performs worse than a randomly generated fixed embedding*. A possible explanation is that meta-learning methods are fitting the task distribution on the base class data, improving performance in that circumstance, but hindering ability to generalize to another task distribution. The second observation is that, by leveraging the statistics of the test data, transductive fine-tuning achieves higher results than the standard fine-tuning. This suggests that reliable estimates of statistics are difficult to measure with only a few examples. The third observation is that the accuracy of most methods on the benchmark continues to be dependent on how similar the dataset is to ImageNet: *CropDiseases* commands the highest performance on average, while *EuroSAT* follows in 2^{nd} place, *ISIC* in 3^{rd}, and *ChestX* in 4^{th}. This further supports the motivation behind benchmark design in targeting applications with increasing visual domain dissimilarity to ImageNet.

Table 5.3 shows results from varying the classifier. While mean-centriod classifier and cosine-similarity classifier are shown to be more efficient than simple linear classifier in the conventional few-shot learning setting, our results show that mean-centroid and cosine-similarity classifier only have a marginal advantage on *ChestX* and *EuroSAT* over linear classifier in the 5-shot case (Table 5.3). As the shot increases, linear classifier begins to dominate mean-centroid and cosine-similarity classifier. One plausible reason is that both mean-centroid and cosine-similarity classifier conduct classification based on unimodal class prototypes, when the number of examples increases, unimodal distribution becomes less suitable to represent them, and multi-modal distribution is required.

We further analyze how layers are changed during transfer. We use θ to denote the original pretrained parameters and $\hat{\theta}$ to denote the parameters after fine-tuning. Figure 5.2 shows the relative parameter change of the ResNet10 miniImageNet pretrained model as $\frac{|\theta-\hat{\theta}|}{|\theta|}$, averaged over all parameters per layer, and 100 runs. Several interesting observations can be made from these results. First, across all the datasets and all the shots, the first layer of the pretrained model changes most. This indicates that if the target domain is different from the source domain, the lower layers of the pretrained models still need to be adjusted. Second, while the datasets are drastically different, we observe that some layers are consistently more *transferable* than other layers. One plausible explanation for this phenomenon is the heterogeneous characteristic of layers in overparameterized deep neural networks [ZBS19].

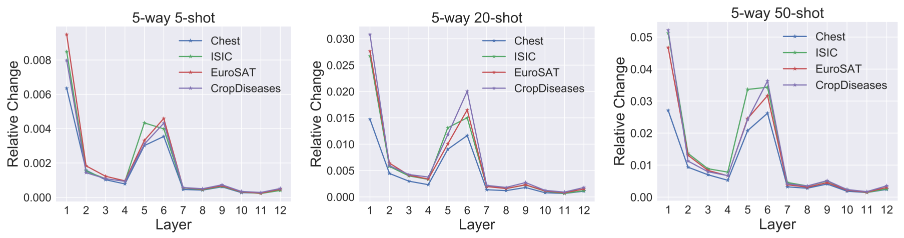


Figure 5.2: Relative change of pretrained network layers for single model transfer.

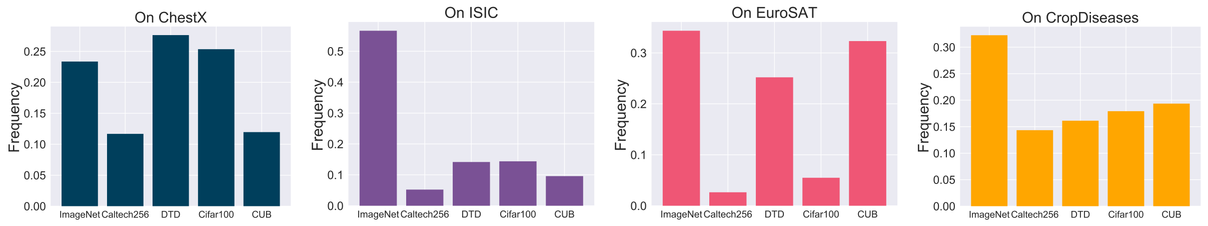


Figure 5.3: Histograms showing frequency of source model selection for each dataset in the benchmark.

Table 5.5: Number of models' effect on test accuracy.

# of models	2	3	4	5
ChestX	34.35%	36.29%	37.64%	37.89%
ISIC	59.4%	62.49%	65.07%	64.77%
EuroSAT	91.71%	93.49%	92.67%	93.00%
CropDiseases	98.43%	98.09%	98.05%	98.60%

Transfer from Multiple Pretrained Models

The results of the proposed *Incremental Muiti-model Selection* are shown in Table 5.4. *IMS-f* fine-tunes each pretrained model before applying the model selection. We include a baseline called *all embeddings* which concatenates the feature vectors generated by all the layers from the fine-tuned models. Across all datasets and shot levels, the average accuracies (and 95% confidence internals) are 68.22% (0.45) for *all embeddings*, and 68.69% (0.44) for *IMS-f*. The results show that *IMS-f* generally improves upon *all embeddings* which indicates the importance of selecting relevant pretrained models to the target dataset. Model complexity also tends to decrease by over 20% compared to *all embeddings* on average. We can also observe that it is beneficial to use multiple pretrained models than using just one model. Compared with standard finetuning with a linear classifier, the average improvement of *IMS-f* across all the shots on *ChestX* is 0.20%, on *ISIC* is 0.69%, on *EuroSAT* is 3.52% and on *CropDiseases* is 1.27%.

In further analysis, we study the effect of the number of pretrained models for the proposed multi-model selection method. We consider libraries consisting of two, three, four, and all five pretrained models. The pretrained models are added into the library in the order of *ImageNet*, *CIFAR100*, *DTD*, *CUB*, *Caltech256.* For each dataset, the experiment is conducted on 5-way 50-shot with 600 episodes. The results are shown in Table 5.5. As more pretrained models are added into the library, we can observe that the test accuracy on *ChestX* and *ISIC* gradually improves which can be attributed to the diverse features provided by different pretrained models. However, on *EuroSAT* and

CropDiseases, only a marginal improvement can be observed. One possible reason is that the features from *ImageNet* already captures the characteristics of the datasets and more pretrained models does not provide additional information.

Finally, we visualize for each dataset which pretrained models are selected in the proposed incremental multi-model selection. The experiments are conducted on 5-way 50-shot with all five pretrained models. For each dataset, we repeat the experiments for 600 episodes and calculate the frequency of each model being selected. The results are shown in Figure 5.3. We observe the distribution of the frequency differs significantly across datasets. This demonstrates that target datasets can benefit from features from different pretrained models.

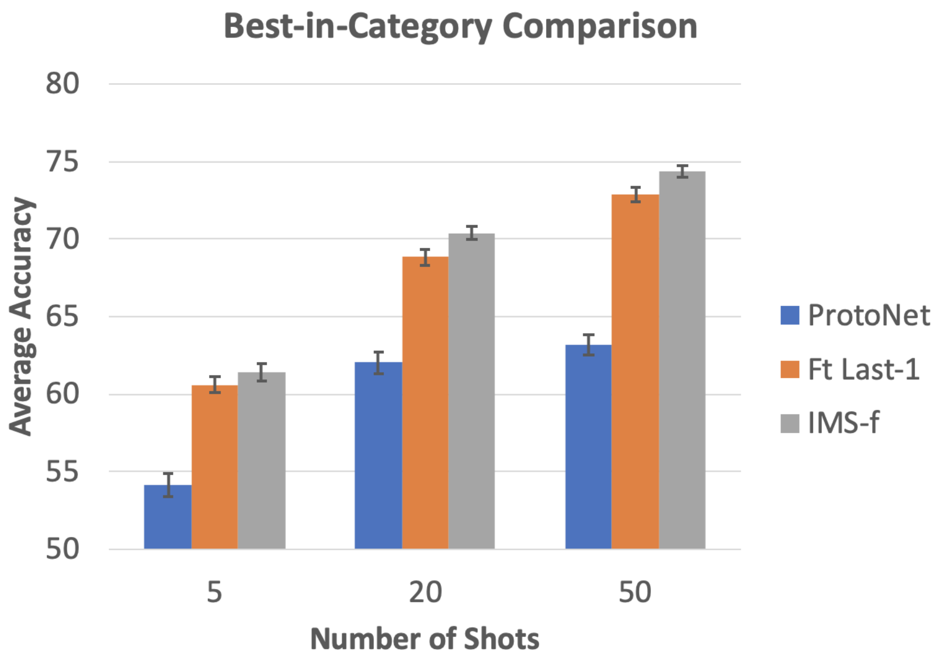


Figure 5.4: Top performing meta-learning, single model, and multi-model transfer learning.

5.6.3 Best-in-category Comparison

Figure 5.4 summarizes the comparison across best-in-category algorithms, according to the average accuracy across all datasets and shot levels in the benchmark. The degradation in performance suffered by meta-learning approaches is significantly greater than the gain between single model and multi-model learning strategies, emphasizing the risk of employing meta-learning strategies for few-shot learning when the application domain may sustain any degree of drift. In addition, the relative performance gain with increasing number of shots is greater with transfer methods compared to meta-learning: 5.8% and 5.7% from 20-shot to 50-shot for single and multi-model transfer learning, respectively, versus 1.8% for meta-learning. 5-shot to 20-shot was similar for all methods: 14.5%, 13.6%, 14.6%, for meta-learning, single model, and multi-model, respectively.

5.7 Conclusion

In this chapter, we formally introduce the problem of cross-domain few-shot learning (CD-FSL) and establish the new CD-FSL benchmark, which covers several target domains with varying similarity to the ImageNet source domain. We extensively analyze and evaluate existing meta-learning methods and variants of transfer learning. The results show that meta-learning approaches significantly underperform in comparison to fine-tuning methods. In addition, the relative performance gain with increasing data is greater with transfer methods compared to meta-learning. Finally, we propose a multi-model selection method to leverage multiple pretrained models from multiple source datasets with similar domains as ImageNet, and demonstrate that this method yields higher average performance than any single model fine-tuning approach. In conclusion, due to its spectrum of diversity and coverage, the proposed benchmark serves as a challenging platform to guide research on cross-domain few-shot learning. In next chapter, we consider learning multiple domains simultaneously with deep neural networks. In particular, we develop a compact representation of deep neural networks to classify images from multiple domain by exploring the universal structures in different domains.

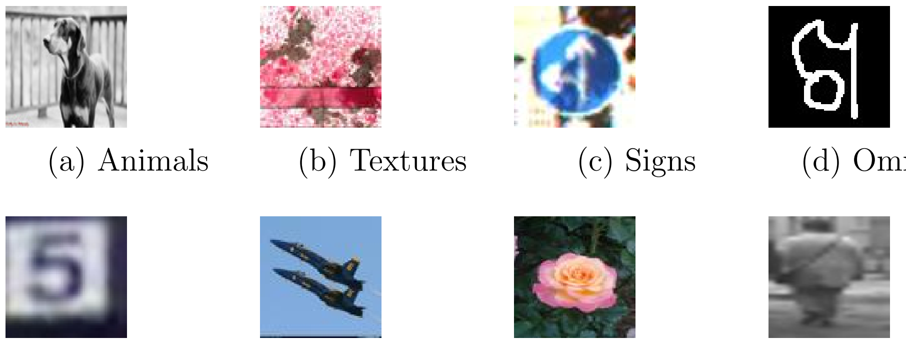
This chapter contains material from “A Broader Study of Cross-Domain Few-Shot Learning”, by Yunhui Guo, Noel C. F. Codella, Leonid Karlinsky, James V. Codella, John R. Smith, Kate Saenko, Tajana Rosing, Rogerio Feris, which appears in *The 16th European Conference on Computer Vision (ECCV)*, 2020. The dissertation author was the primary investigator and author of this paper.

Chapter 6

Learning across Multiple Domains Simultaneously

6.1 Introduction

Previous chapters consider problems of learning across two domains with deep neural networks. However, real-world applications may involve data from multiple domains. There are two challenges in learning with multiple domains simultaneously with deep neural networks. The first one is to identify a common structure among different domains. As shown in Fig 6.1, images from different domains are visually different, it is challenging to design a single feature extractor for all domains. Another challenge is to add new tasks to the model without introducing additional parameters. Existing neural network based multi-domain learning approaches [BV17, RBV17, RBV18a, RT17] mostly focus on the architecture design while ignoring the structural regularity hidden in different domains



(a) Animals (b) Textures (c) Signs (d) Omniglot

(e) Digits (f) Aircraft (g) Flowers (h) Pedestrian

Figure 6.1: Image examples from different domains.

that leads to sub-optimal solutions.

In this dissertation, we propose a multi-domain learning approach based on depthwise separable convolution. Depthwise separable convolution has been proved to be a powerful variation of standard convolution for many applications, such as image classification [Cho17], natural language processing [KGC17] and embedded vision applications [HZC$^+$17]. To the best of our knowledge, this is the first work that explores depthwise separable convolution for multi-domain learning. The proposed multi-domain learning model is compact and easily extensible. To promote transfer learning between different domains we further introduce a softmax gating mechanism. We evaluate our method on Visual Decathlon Challenge [RBV17], a benchmark for testing multi-domain learning models. Our method can beat the state-of-the-art models with only 50% of the parameters.

Summary and contributions: The contributions are summarized below:

- We propose a novel multi-domain learning approach by exploiting the structure regularity hidden in different domains. The proposed approach greatly reduces the number of parameters and can be easily adapted to work on new domains.

- The proposed approach is based on the assumption that images in different domains share cross-channel correlations while having domain-specific spatial correlations. We validate the assumption by analyzing the visual concepts captured by depthwise separable convolution using *network dissection* [BZK+17].
- Our approach outperforms the state-of-the-art results on Visual Decathlon Challenge with only half of the parameters.

6.2 Related Work

Different from multi-task learning, multi-domain learning aims at creating a single neural network to perform image classification tasks in a variety of domains. [BV17] showed that a single neural network can learn simultaneously several different visual domains by using an instance normalization layer. [RBV17, RBV18a] proposed *Residual Adapters* which is a universal parametric families of neural networks that contain specialized problem-specific models. [RT17] proposed a method called Deep Adaptation Networks (DAN) that constrains newly learned filters for new domains to be linear combinations of existing ones. Multi-domain learning can promote the application of deep learning based vision models since it reduces engineers' effort to train new models for images from different domains. Our proposed multi-domain learning algorithm called *SharingNet* based on the depthwise separable convolution which reduces the number of parameters of *Residual Adapters* by 50%.

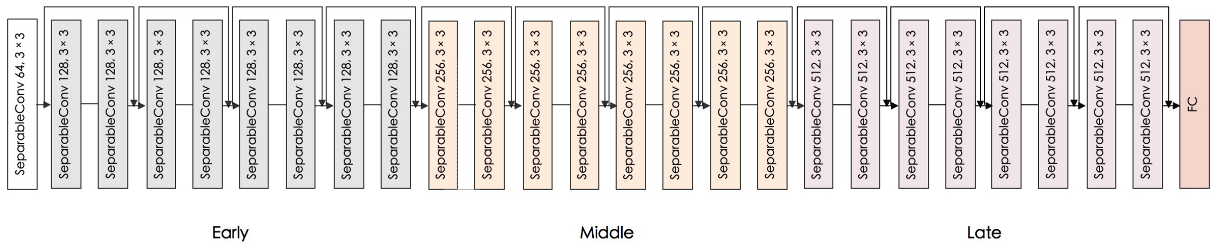


Figure 6.2: ResNet-26 with depthwise separable convolution.

6.3 Background

Depthwise
Pointwise

Figure 6.3: Standard convolution and depthwise separable convolution.

6.3.1 Depthwise Separable Convolution

Our proposed approach is based on depthwise separable convolution that factorizes a standard 3×3 convolution into a 3×3 depthwise convolution and a 1×1 pointwise convolution. While standard convolution performs the channel-wise and spatial-wise computation in one step, depthwise separable convolution splits the computation into two steps: depthwise convolution applies a single convolutional filter per each input channel and pointwise convolution is used to create a linear combination of the output of the depthwise convolution. The comparison of standard convolution and depthwise separable convolution is shown in Fig. 6.3.

Consider applying a standard convolutional filter K of size $W \times W \times M \times N$ on an input feature map F of size $D_f \times D_f \times M$ and produces an output feature map O is of size $D_f \times D_f \times N$,

$$O_{k,l,n} = \sum_{i,j,m} K_{i,j,m,n} \cdot F_{k+i-1,l+j-1,m} \tag{6.1}$$

In depthwise separable convolution, we factorize above computation into two steps. The first step applies a 3×3 depthwise convolution $\hat{K}$ to each input channel,

$$\hat{O}_{k,l,m} = \sum_{i,j} \hat{K}_{i,j,m} \cdot F_{k+i-1,l+j-1,m} \tag{6.2}$$

The second step applies 1×1 pointwise convolution $\tilde{K}$ to combine the output of depthwise convolution,

$$O_{k,l,n} = \sum_{m} \tilde{K}_{m,n} \cdot \hat{O}_{k-1,l-1,m} \tag{6.3}$$

Depthwise convolution and pointwise convolution have different roles in generating new features: the former is used for capturing spatial correlations while the latter is used for capturing channel-wise correlations.

Most the previous works [Cho17, HZC$^+$17, SHZ$^+$18] focus on the computational aspect of depthwise separable convolution since it requires less parameters than standard convolution and is more computationally effective. In [Cho17], the authors proposed the

"Inception hypothesis" stating that mapping cross-channel correlations and spatial correlations separately is more efficient than mapping them at once. In this chapter, we provide further evidence to support this hypothesis in the setting of multi-domain learning. We validate the assumption that images from different domains share cross-channel correlations but have domain-specific spatial correlations. Based on this idea, we develop a highly efficient multi-domain learning method. We further analyze the visual concepts captured by depthwise convolution and pointwise convolution based on *network dissection* [BZK+17]. The visualization results show that while having less parameters depthwise convolution captures more concepts than pointwise convolution.

6.4 Proposed Approach

6.4.1 Network Architecture

For the experiments, we use the same ResNet-26 architecture as in [RBV18a]. This allows us to fairly compare the performance of the proposed approach with previous ones. This original architecture has three macro residual blocks, each outputting 64, 128, 256 feature channels. Each macro block consists of 4 residual blocks. Each residual block has two convolutional layers consisting of 3×3 convolutional filters. The network ends with a global average pooling layer and a softmax layer for classification.

Different from [RBV18a], we replace each standard convolution in the ResNet-26 with depthwise separable convolution and increase the channel size. The modified network architecture is shown in Fig. 6.2. This choice leads to a more compact model while still

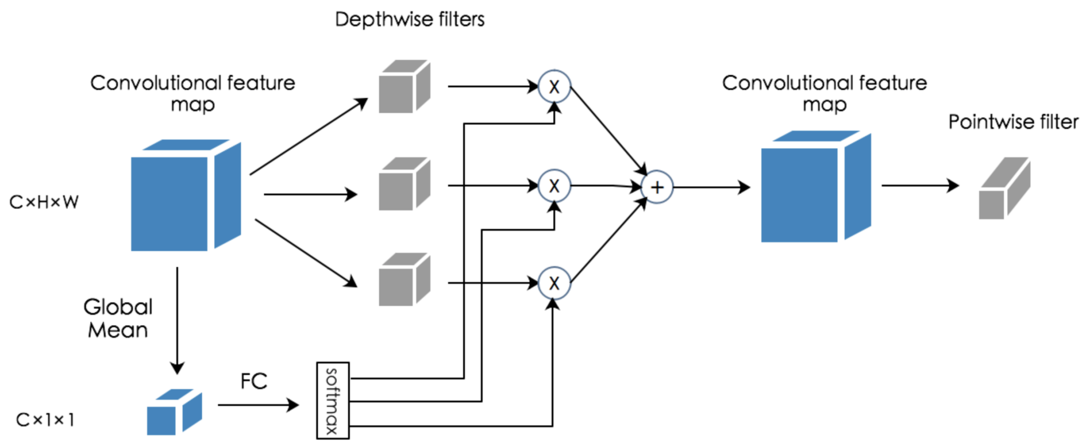


Figure 6.4: The proposed transfer learning approach for sharing spatial correlations.

maintaining enough network capacity. The original ResNet-26 has over 6M parameters while our modified architecture has only half the amount of parameters. In the experiments we found that the reduction of parameters does no harm to the performance of the model. The use of depthwise separable convolution allows us to model cross-channel correlations and spatial correlations separately. The idea behind our multi-domain learning method is to leverage the different roles of cross-channel correlations and spatial correlations in generating image features by sharing the pointwise convolution across different domains.

6.4.2 Learning Multiple Domains

For multi-domain learning, it is essential to have a set of universally sharable parameters that can generalize to unseen domains. To get a good starting set of parameters, we first train the modified ResNet-26 on ImageNet. After we obtain a well-initialized network, each time when a new domain arrives, we add a new output layer and finetune the depth-wise convolutional filters. The pointwise convolutional filters are shared accross different domains. Since the statistics of the images from different domains are different, we also allow domain-specific batch normalization parameters. During inference, we stack the trained depthwise convolutional filters for all domains as a 4D tensor and the output of domain d can be calculated as,

$$\hat{O}_{k,l,m,d} = \sum_{i,j} \hat{K}_{i,j,m,d} \cdot F_{k+i-1,l+j-1,m,d} \tag{6.4}$$

The adoption of depthwise separable convolution provides a natural separation for modeling cross-channel correlations and spatial correlations. Experimental evidence [Cho17] suggests the decouple of cross-channel correlations and spatial correlations would result in more useful features. We take one step further to develop a multi-domain domain method based on the assumption that different domains share cross-channel correlations but have domain-specific spatial correlations. Our method is based on two observations: model efficiency and interpretability of hidden units in a deep neural network.

Model efficiency Table 6.1 shows the comparison of standard 3×3 convolution, 3×3 depthwise convolution (Dwise) and 1×1 pointwise convolution (Pwise). Clearly, standard convolution has far more parameters than both depthwise convolution ($\times c_2$) and pointwise convolution ($\times 9$). Typically, pointwise convolution has more parameters than depthwise convolution. In the architecture shown in Fig 6.2, pointwise convolution accounts for 80% of the parameters in the convolutional layers. The choice of sharing pointwise convolution and adding depthwise convolution induces minimal additional parameters when dealing

Table 6.1: Comparison of standard 3×3 convolution, 3×3 depthwise convolution (Dwise) and 1×1 pointwise convolution (Pwise).

Input	Operator	Output	Parameters
$c_1 \times h \times w$	3×3 Conv2d	$c_2 \times h \times w$	$3 \times 3 \times c_1 \times c_2$
$c_1 \times h \times w$	3×3 Dwise	$c_1 \times h \times w$	$3 \times 3 \times c_1$
$c_1 \times h \times w$	1×1 Pwise	$c_2 \times h \times w$	$1 \times 1 \times c_1 \times c_2$

with new domains. In the experiments we found that only by adding depthwise convolution leads to a network with limited number of free parameters which cannot handle some large datasets. To increase the network capacity, we allow the last convolutional layer to be specific for each domain. Based on this modification, each new domain averagely introduces 0.3M additional parameters which is 10% of the modified ResNet-26.

Interpretability While depthwise convolution typical has less paramaters, by using the technique of *network dissection* [BZK$^+$17], we found it captures more visual concepts than pointwise convolution. Meanwhile, the results in the same convolutional layer show that depthwise convolution captures higher level concepts such as wheel and grass while pointwise convolution can only detect dots or honeycombed. This observation suggests that pointwise convolution can be generally shared between different image domains since it is typically used for dealing with lower level features.

6.4.3 Soft Sharing of Trained Depthwise Filters

The multi-domain learning extension of depthwise separable convolution allows us to share the channel-wise correlations among different domains. However, it still limits the level of transfer learning as there is no sharing between spatial correlations. To break the limit, we introduce a novel transfer learning approach in the multi-domain setting to allow the sharing of depthwise convolution. For each domain D_j, consider a network with L depthwise separable convolutional layers $S_1, ..., S_L$, the input to the pointwise convolution in layer l is defined as,

$$\hat{O}^l = \sum_{i=1}^{T} s_i^l \hat{O}_i^l, \quad \text{with} \sum_{i=1}^{T} s_i^l = 1 \tag{6.5}$$

where $\hat{O}_i^l$ is the output of the depthwise convolution of domain i in the layer l if we use images in domain D_j as input. s_i^l denotes a learned scale for the depthwise convolution of domain i in the layer l. Figure 6.4 shows an example of proposed transfer learning approach. The scales $s_1, ..., s_T$ are the output of a softmax gate. The input to the softmax gate is the convolutional feature map $X_{l-1} \in \mathbb{R}^{C \times H \times W}$ produced by the previous layer. Similar to [VB17], we only consider global channel-wise features. In particular, we perform global average pooling to compute channel-wise means,

$$M_c = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} X_{c,i,j} \tag{6.6}$$

The output is a 3-dimensional tensor of size $C \times 1 \times 1$. To achieve a lightweight design, we adopt a simple feedforward network consisting of two linear layers with ReLU activations to apply a nonlinear transformation on the channel-wise means and feed the output to the softmax gate. All the convolutional filters are freezed during transfer learning. The scales $s_1, ..., s_T$ and the parameters of the feedforward networks are learnt jointly via backpropagation.

It is widely believed that early layers in a convolutional neural network are used for detecting lower level features such as textures while later layers are used for detecting parts or objects. Based on this observation, we partition the network into three regions (early, middle, late) as shown in Figure 6.2 and consider different placement of the softmax gate which allows us to compare a variety of sharing strategies.

6.5 Experiment

6.5.1 Datasets and evaluation metrics

We evaluate our approach on Visual Domain Decathlon Challenge [RBV17]. It is a challenge to test the ability of visual recognition algorithms to cope with images from different visual domains. There are a total of 10 datasets: (1) **ImageNet** (2) **CIFAR-100** (3) **Aircraft** (4) **Daimler pedestrian classification** (5) **Describable textures** (6) **German traffic signs** (7) **Omniglot** (8) **SVHN** (9) **UCF101 Dynamic Images** (10) **VGG-Flowers**. The detailed statistics of the datasets can be found at http://www.robots.ox.ac.uk/ vgg/decathlon/.

The performance is measured in terms of a single scalar score S computed as follows,

$$S = \sum_{i=1}^{10} \alpha_i \max\{0, E_i^{\max} - E_i\}^{\gamma_i} \tag{6.7}$$

where,

$$E_i = \frac{1}{D_i^{\text{test}}} \sum_{(x,y)\in D_i^{\text{test}}} 1\{y \neq (\mathcal{E}_{(D_i)} \circ \mathcal{C})(x)\} \tag{6.8}$$

E_i is the average test error of domain D_i. $E_i^{\max}$ is the error of a reasonable baseline algorithm. The exponent γ_i is set to be 2 for all domains. The coefficient α_i is $1000(E_i^{\max})^{-\gamma_i}$ then a perfect classifier receives 1000. The maximum score achieved across 10 domains is 10000.

6.5.2 Baselines

We consider the following baselines in the experiments,

1. **Individual Network**: The simplest baseline we consider is Individual Network. We finetune the pretrained modified ResNet-26 on each domain which leads to 10 models altogether. This approach results in the largest model size since there is no sharing between different domains.

2. **Classifier Only**: We freeze the feature extractor part of the pretrained modified ResNet-26 on ImageNet and train domain-specific classifier layer for each domain.

3. **Depthwise Sharing**: Rather than sharing pointwise convolution, we consider an alternative approach of multi-domain extension of depthwise separable convolution which shares the depthwise convolution between different domains.

4. **Residual Adapters**: Residual Adapters [RBV17, RBV18a] are the state-of-the-art approaches for multi-domain learning which include Serial Residual Adapter [RBV17]

Table 6.2: Top-1 classification accuracy and the Visual Decathlon Challenge score (S) of the proposed approach and baselines. #par is the number of parameters w.r.t. the proposed approach.

Model	#par	ImNet	Airc.	C100	DPed	DTD	GTSR	Flwr	OGlt	SVHN	UCF	mean	S
# images		1.3m	7k	50k	30k	4k	40k	2k	26k	70k	9k		
Serial Res. Adapt.	2×	59.67	61.87	81.20	93.88	57.13	97.57	81.67	89.62	96.13	50.12	76.89	2621
Parallel Res. Adapt.	2×	60.32	64.21	81.91	94.73	58.83	99.38	84.68	89.21	96.54	50.94	78.07	3412
DAN	2.17×	57.74	64.12	80.07	91.30	56.64	98.46	86.05	89.67	96.77	49.38	77.01	2851
Piggyback	1.28×	57.69	65.29	79.87	96.99	57.45	97.27	79.09	87.63	97.24	47.48	76.60	2838
Individual Network	5×	63.99	65.71	78.26	88.29	52.19	98.76	83.17	90.04	96.84	48.35	76.56	2756
Classifier Only	0.6×	63.99	51.04	75.32	94.49	54.21	98.48	84.47	86.66	95.14	43.75	74.76	2446
Proposed Approach	1×	63.99	61.06	81.20	97.00	55.48	99.27	85.67	89.12	96.16	49.33	77.82	3507

Table 6.3: Top-1 classification accuracy and the Visual Decathlon Challenge score (S) of different soft sharing strategies.

Model	ImNet	Airc.	C100	DPed	DTD	GTSR	Flwr	OGlt	SVHN	UCF	mean	S
# images	1.3m	7k	50k	30k	4k	40k	2k	26k	70k	9k		
early	63.99	58.69	81.01	95.44	55.75	98.75	84.90	88.80	96.18	48.86	77.23	3102
middle	63.99	59.11	80.93	95.33	54.74	98.71	85.42	88.93	96.09	48.91	77.21	3086
late	63.99	58.81	80.93	96.63	54.74	98.91	84.79	89.35	96.30	49.01	77.88	3303

and Parallel Residual Adapter [RBV18a].

5. **Deep Adaptation Networks (DAN)**: In [RT17] the authors propose Deep Adaptation Networks (DAN) that constrains newly learned filters for new domains to be linear combinations of existing ones via *controller modules*.

6. **PiggyBack**: In [ML18] the authors present PiggyBack for adding multiple tasks to a single network by learning domain-specific binary masks. The main idea is derived from network quantization [CHS+16, Guo18] and pruning.

6.5.3 Implementation details

All networks were implemented using Pytorch and trained on 2 NVIDIA V100 GPUs. For the base network trained on ImageNet we use SGD with momentum as the optimizer. We set the momentum rate to be 0.9, the initial learning rate to be 0.1 and use a batch size of 256. We train the network with a total of 120 epochs and the learning rate decays twice at 80th and 100th epoch with a factor of 10. To prevent overfitting, we use a weight decay (L2 regularization) rate of 0.0001.

For the multi-domain extension of depthwise separable convolution, we keep the same optimization settings as training the base network. We train the network with a total of 100 epochs and the learning rate decays twice at 60th and 80th epoch by a factor of 10. We apply weight decay (L2 regularization) to prevent overfitting. Since the size of the datasets are highly unbalanced, we use different weight decay parameters for different domains. Similar to [RBV18a], higher weight decay parameters are used for smaller datasets. In particular, 0.002 for DTD, 0.0005 for Aircraft, CIFAR100, Daimler pedestrain, Omniglot and UCF101, and 0.0003 for GTSTB, SVHN and VGG-Flowers.

For the Soft Sharing of Trained Depthwise Filters part, we train the network with a total of 10 epochs and the learning rate decays once at the 5th epoch with a factor of 10. Other settings are kept the same as training multi-domain models.

6.6 Results and Analysis

6.6.1 Quantitative Results

The results of the proposed approach and the baselines on Visual Decathlon Challenge are shown in Table 6.2. Our approach achieves the highest score among all the methods while requiring the least amount of parameters. In particular, the proposed approach improves the current state-of-the-art approaches by 100 points with only 50% of the parameters. The ResNet-26 with depthwise separable convolution surpasses the performance of the original ResNet-26 by a large margin on ImageNet (63.99 vs 60.32). On other smaller datasets, our approach still achieves better or comparable performance to the baselines. The improvement can be attributed to the sharing of pointwise convolution that has a regularization effect and allows the training signals in ImageNet to be reused when training new domains.

Compared with other variations of the modified ResNet-26, our approach still achieves the highest score. Our approach obtains a remarkable improvement (3507 vs 2756) with only 20% of the parameters compared with Individual Network. One reason for the improvement is that the proposed approach is more robust to overfitting, especially for some small datasets. While only training domain-specific classifier layers leads to the smallest model, the score is about 1000 points lower than the proposed approach. Compared with Depthwise Sharing, the assumption of sharing pointwise convolution leads to a more compact and efficient model (3507 vs 3234). This validates our assumption that it is preferable to share pointwise convolution rather than depthwise convolution in the

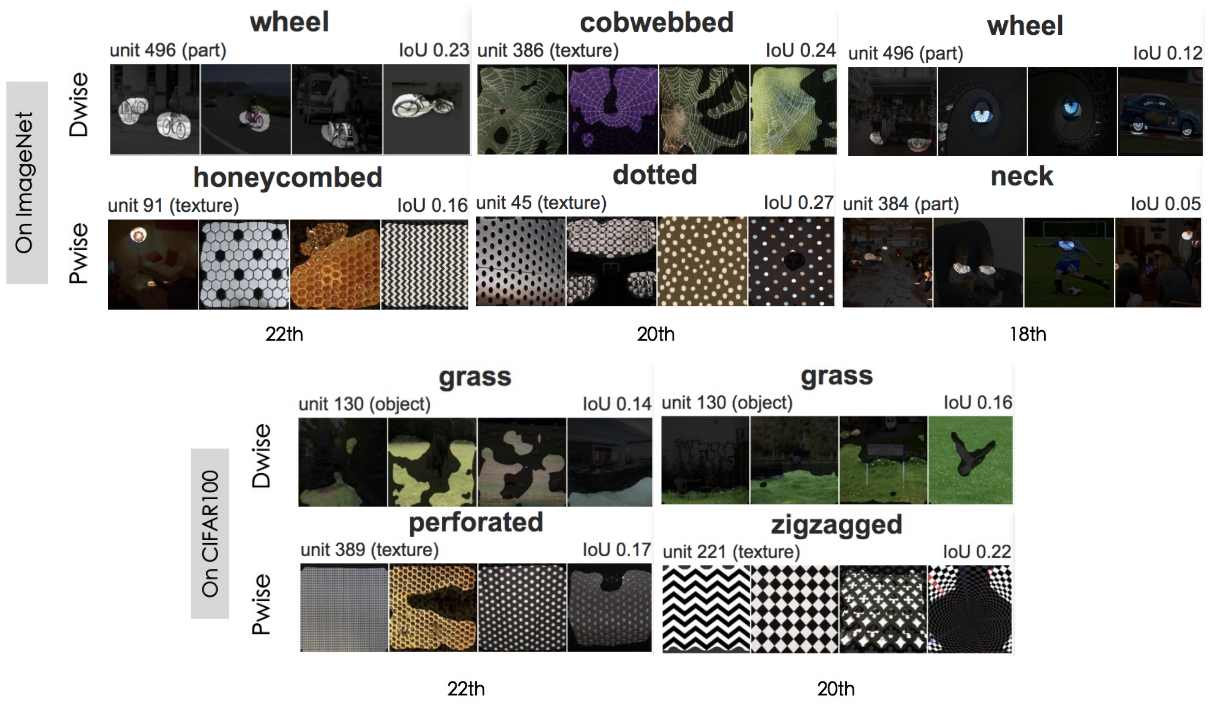
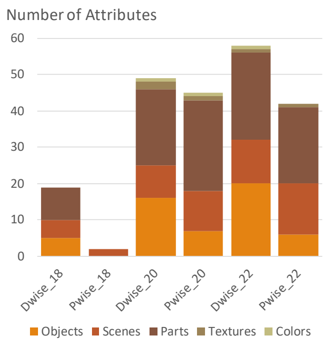


Figure 6.5: A comparison of visual concepts identified by network dissection in ResNet-26 with depthwise separable convolution trained on ImageNet and CIFAR100.

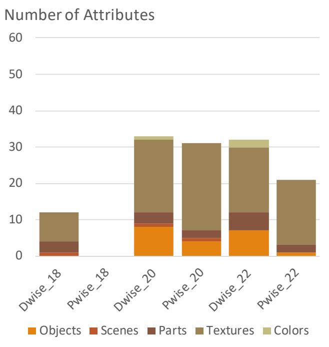
setting of mutli-domain learning. We provide more qualitative results in the next section to support this claim.

6.6.2 Qualitative Results

This section presents our visualization results of deptwise convolution and pointwise convolution based on *network dissection* [BZK+17]. *Network dissection* is a general framework for quantifying the interpretability of deep neural networks by evaluating the alignment between individual hidden units and a set of semantic concepts. The accuracy of unit k in detecting concept c is denoted as $IoU_{k,c}$. If the value of $IoU_{k,c}$ exceeds a threshold then we consider the unit k as a detector for the concept c. The details of calculating



(a) On ImageNet



(b) On CIFAR100

Figure 6.6: Number of attributes captured by the hidden units of depthwise convolution and pointwise convolution in the 18th, 20th and 22th convolutional layer.

$IoU_{k,c}$ is omited due to space limitation.

In the experiments, we use the individual networks trained on ImageNet and CIFAR100 as examples. We visualize the hidden units in the 18th, 20th, 22th convolutional layers. Fig 6.5 shows the interpretability of units of the depthwise convolution and pointwise convolution in the corresponding layer. The highest-IoU matches among hidden units of each layer are shown. We observe that the hidden units in depthwise convolution detect higher level concepts than the units in pointwise convolution. The units in the depthwise convolution can capture part or object while the units in pointwise convolution can only detect textures. Moreover, Fig 6.6 shows the number of attributes captured by the units in depth convolution and pointwise convolution. The results demonstrate that depthwise convolution consistently detects more attributes than pointwise convolution. These observations imply that pointwise convolution are mostly used for capturing low level features which can be generally shared across different domains.

6.6.3 Soft Sharing of Trained Depthwise Filters

Table 6.3 shows the results of sharing of trained depthwise filters. Regardless of the different placements of the softmax gate, the base approach without sharing still achieves the highest score on Visual Decathlon Challenge. One possible reason is that the datasets are from very different domains, sharing information between them may not generally improve the performance. However, for some specific datasets, we still observe some improvement. In particular, by sharing early layers we can obtain a slightly higher accuracy on DTD and SVHN. Another observation is that sharing later layers leads to a higher score than other alternatives. This implies that although images in different domain may not share similar low level features, they can still be benefited from each other by transfering information in later layers.

6.7 Conclusion

In this chapter, we present a multi-domain learning approach based on depthwise separable convolution. The proposed approach is based on the assumption that images from different domains share the same channel-wise correlation but have domain-specific spatial-wise correlation. We evaluate our approach on Visual Decathlon Challenge and achieve the highest score among the current approaches. We further visualize the concepts detected by the hidden units in depthwise convolution and pointwise convolution. The results reveal that depthwise convolution captures more attributes and higher level concepts than pointwise convolution. In next chapter, we extend the idea of learning with multiple domains to applications in distributed mobile networks.

This chapter contains material from “Depthwise Convolution is All You Need for Learning Multiple Visual Domains”, by Yunhui Guo, Yandong Li, Liqiang Wang, Tajana Rosing, which appears in *The 33rd AAAI Conference on Artificial Intelligence (AAAI)*, 2019. The dissertation author was the primary investigator and author of this paper.

Chapter 7

Energy Efficient Learning with Multiple Domains in Heterogeneous Mobile Networks

7.1 Introduction

In last chapter we consider learning multiple image domains, in this chapter we focus on learning in heterogeneous mobile networks which consist of data from different domains collected from the mobile devices. In mobile networks, devices such as mobile phones, tablets and mobile sensors are generating a huge amount of data each day, enabling everything from remotely monitoring heart rate to tracking the location of smartphone [AIM10, GBT$^+$04]. The mobile devices can monitor changes of user behavior and collect a large amount of data that can be used for downstream tasks such as human activity and image recognition [AIM10, RN04, MMR$^+$16]. The richness of data generated by mobile devices can potentially provide more information about the environment around us, but only if we are able to develop efficient processing systems to analyse the data.

Machine learning (ML) algorithms have become a core component for building data analytic systems [ABC$^+$16, ZCF$^+$]. Most ML algorithms are server-based and designed for handling centralized data, that is, all the training examples are generated in one place [MMR$^+$16]. However, mobile networks are distributed in nature. Each device only gathers a subset of the data and works collaboratively with a central server. Thus, learning in mobile networks requires extensive data communication which is challenging for training conventional ML algorithms. Mobile edge computing [PSBD19, MMR$^+$16, PKP06] has been proposed recently as an efficient way to learn ML models in mobile networks. The core idea of mobile edge computing is to push the computation from the central server to the mobile devices which can reduce the communication cost, energy consumption and protect users' privacy of learning in mobile networks [MMR$^+$16, PSBD19].

Combining information from multiple heterogeneous mobile devices is a new research direction for mobile edge computing [YYS18, TGK$^+$19, AVHN19]. One characteristic of learning with heterogeneous mobile devices is that the *feature vectors*, rather than the *examples*, are distributed across devices [YYS18, TGK$^+$19]. For example, for human activity recognition, accelerometer is used to measure acceleration and gyroscope is used to detect the orientation of the device. The data from these two devices form a single feature vector that can be used to accurately predict user's movements. In [AVHN19], the authors introduced *multi-mobile computing* which integrates multiple heterogeneous mobile devices to build more complex mobile applications.

Most of the works on mobile edge computing, such as *federated learning* [MMR$^+$16, KMY$^+$16] and *distributed gradient algorithms* [ZWD12, WTS$^+$18], mainly focus on learning in homogeneous mobile networks where the devices have different subsets of the dataset that share common feature space. In contrast, learning in heterogeneous mobile network has the following key properties which make it a more challenging task and has only been addressed by some recent works [TMK17, VGSR18, TGK$^+$19].

- **Distributed features**: Conventional ML algorithms need access to the full feature vectors to make predictions. However, in heterogeneous mobile network the feature vectors are distributed across devices. To gather together the distributed features in the central server incurs a large communication cost. This also introduces data from multiple domains which is challenging for conventional ML algorithms. Excessive data communication also dominates the energy consumption of the mobile devices [Mad03, WCH$^+$19] and leads to network congestion.

- **Asymmetrical network bandwidth**: Many telecom companies provide Internet plans with much faster download speeds than upload speeds [BCL10]. For example, AT&T top download and upload speeds can have as much as 5x difference. On the other hand, the devices typically have high sampling rates (100 Hz or higher) and deliver continuous data to the central server. The limited upload speed poses a challenge to communicate the local data which increases the network latency and power usage.

- **Online update**: The user behavior might change over time. It is thus important to have an online update mechanism to adjust the trained ML model in the central server with incoming data with minimal communication cost.

While learning in heterogeneous mobile networks is generally difficult, in this chapter we leverage two key facts and propose an active sampling algorithm for training in heterogeneous mobile networks to reduce the communication cost and energy consumption. First, we observe that not all the data are equally important. For example, the data that identify the transition from one activity to another are more informative. On each round of the online update, instead of sending all the data to the central server, the proposed active sampling method sends only the most *informative* data. Our key insight is that these informative data can be chosen by using ideas from the active learning literature [Set09]. Prior research on active learning mainly focuses on reducing the cost of acquiring labels for specific ML applications [RC10, BGNK18, KGUD07, LG13]. In this work, we show that active learning can be easily applied for reducing communication cost and energy consumption for learning in heterogeneous mobile networks with data from multiple domains. We propose active sampling, which is inspired by *uncertainty sampling* in active learning, to evaluate the *informativeness* of the data. By only sending the most informative data, we can greatly reduce the communication cost in harsh network conditions and still allow the model to be adaptive.

With the proposed active sampling method, the central server model cannot be updated since it only receives a subset of the full feature vectors. To solve this problem, we leverage the fact that the data from different devices within a region are highly correlated. Thus the missing data can be accurately recovered with the available data from other devices. More concretely, we model the missing data as a function of the received data and labels from other devices. This allows the central server model to be updated only based on a small set of informative data. Finally, we propose a lightweight load balancing algorithm to enable each device to communicate roughly the same number of measurements to prolong the lifetime of the overall system.

To summarize, our work makes three main contributions,

- To our knowledge this is the first attempt to systematically show that the idea of active learning can be used to greatly reduce the communication cost and energy consumption for training in heterogeneous mobile networks with data from multiple domains.
- We propose active sampling methods for communication and energy efficient training in heterogeneous mobile networks.
- We validate the proposed approaches on four real-world datasets by both numerical simulation and practical deployment. The results show that we can achieve a reduction in communication cost (in bits) by up to 53% and in energy consumption by up to 67% without accuracy loss compared with the conventional methods.

7.2 Related Work

A fundamental problem in learning in distributed mobile networks is how to make the right tradeoff between communication and computation [JC04, GBT$^+$04, MMR$^+$16, WTS$^+$18]. Based on the characteristics of the devices, previous work can be classified into two categories: learning with a homogeneous set of devices [GBT$^+$04, WTS$^+$18, MMR$^+$16] and learning with a heterogeneous set of devices [CTS14, YYS18, BT89]. Conventional distributed gradient descent algorithms [ZWD12, WTS$^+$18, DCM$^+$12] and federated learning [MMR$^+$16] fall under the first category where the examples are distributed across devices. The main communication cost in this setting comes from sending the model parameters between devices and the central server [KMY$^+$16]. The methods for reducing communication cost in homogeneous mobile networks can be classified into two threads. One is model compression and quantization, which tries to reduce the communication cost by compressing or quantizing the model parameters [KMY$^+$16, LHM$^+$17, Guo18]. Another is from an algorithmic perspective, which focuses on developing communication efficient optimization algorithms [SSZ14, JST$^+$14].

In contrast, in heterogeneous mobile networks the feature vectors, rather than the examples, are distributed across different devices [BT89, CTS14, YYS18, TGK$^+$19]. The distributed features further increase the communication cost between devices and the central server [BT89, CTS14]. In [YYS18], the authors assume the devices can communicate with each other which is unrealistic in real-world mobile applications. In [TMK17, TGK$^+$19, VGSR18], the authors proposed to distribute the model in the central server onto the devices and conduct computation locally to reduce the dimension of the data. However, these works mainly focus on inference rather than training in heterogeneous mobile networks.

Active learning has been widely applied in machine learning applications, ranging from object categorization [KGUD07] to text classification [TK01]. However, there are only a few works on exploiting active learning in distributed mobile networks. In [SNR06], the authors propose a mobile path planning scheme based on active learning, which leverages previous samples to guide the motion of the mobiles for further sampling. In [WNC06, WMN04], the authors use active learning to achieve faster rate of convergence of field estimation using wireless sensor networks.

In contrast to previous research, we address the problem of communication and energy efficient training in heterogeneous mobile networks with data from multiple domains. With the proposed active sampling methods, we only send informative data from the devices to the central server which can greatly reduce the communication cost and energy consumption. The idea of the proposed algorithm is general and can be easily integrated with the existing applications in distributed mobile networks.

7.3 Background

In this section, we formulate the problem of learning in heterogeneous mobile networks. Given a heterogeneous mobile network which consists of one central server C and K mobile devices. The central server and the mobile devices are connected via wireless network. We consider an online learning setting where the examples arrive sequentially $\{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), ..., (\mathbf{x}_T, \mathbf{y}_T)\}$, where $\mathbf{x}_t \in \mathbb{R}^m$ is an m-dimensional feature vector and $\mathbf{y}_t$ is the one-hot representation of the ground-truth label. The maximum class index is denoted as I. Consider learning a neural network $f_\theta(\mathbf{x})$ with parameter θ for classification in the central server. The prediction of the model can be computed as,

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z}) = \frac{\exp(\mathbf{z})}{\sum_{i=1}^{I} \exp(\mathbf{z}_i)} \tag{7.1}$$

where $\mathbf{z} = f_\theta(\mathbf{x})$. Denote the cross-entropy loss function as ℓ, the loss of the example $\mathbf{x}$ can be computed as,

$$\ell(\hat{\mathbf{y}}; \mathbf{y}) = -\frac{1}{I} \sum_{i=1}^{I} y_i \log \hat{y}_i \tag{7.2}$$

where $\hat{y}_i$ and y_i is the i-th component of $\hat{\mathbf{y}}$ and $\mathbf{y}$, respectively. In heterogeneous mobile network, each device k contains a partial feature vector $\mathbf{x}^k \in \mathbb{R}^{m_1}$. The data from all K devices can be concatenated to form a single feature vector $\mathbf{x}$, i.e., $m = \sum_{k=1}^{K} m_1$. Typically, $\mathbf{x}^k$ itself does not contain enough information to learn a predictive model locally, which indicates that all the mobile devices need to send the local data to the central server [TMK17, TGK$^+$19, VGSR18]. The communication process thus incurs prohibitively large communication and energy costs [TMK17, TGK$^+$19].

Several methods such as hierarchical learning [TGK$^+$19] and DDNN [TMK17] are proposed recently to address the question that how to enable efficient learning of machine learning models, especially neural networks, in heterogeneous mobile networks. The core idea of these approaches is to split or partition the original model f_θ into multiple submodels. Suppose the central server C has a model f_{θ_c} and each device k has a local model f_{θ_k}. The model on device k computes some statistics $f_{\theta_k}(\mathbf{x}^k)$ and send the result to the central server. The central server aggregates the results from all the devices using an aggregate function, such as concatenation. The output of the aggregate function is used as input for f_{θ_c} for classification. The reason that these approaches can reduce communication and energy cost is that the dimension of $f_k(\mathbf{x}^k)$ is much smaller than $\mathbf{x}^k$, it is thus more communication efficient to send $f_k(\mathbf{x}^k)$ rather than the original partial feature vector $\mathbf{x}^k$ to the central server.

The previous works such as [TMK17, TGK$^+$19] mainly focus on inference in heterogeneous mobile works and cannot support training. To train neural networks in heterogeneous mobile works, the commonly used approach is to send the original local data to the central server which incurs a large communication cost [TGK$^+$19], this approach is referred to as *passive learning* since the devices communicate with the central server in each online update round. In this work, we propose an active sampling method which allows the devices to *actively* decide which data to communicate to the central server in each online update round. Different from [TMK17, TGK$^+$19] which focus on inference in heterogeneous mobile works, the proposed method selectively transfers data from the devices to the central server to reduce the communication and energy cost for training in heterogeneous mobile works.

7.3.1 Computation Model for Mobile Devices

For modelling the mobile devices, we adopt a computation model which is also used in [BB96, TBZH19] for validating federated learning. The number of CPU cycles of device k to process one floating number is denoted as C_k. The CPU-cycle frequency of the device k is f_k. The CPU energy consumption of the device k for processing one floating number can be computed as [BB96],

$$E_p = \frac{\alpha_k}{2} C_k f_k^2, \tag{7.3}$$

where $\frac{\alpha_k}{2}$ is the effective capacitance coefficient of the device k.

7.3.2 Communication Model for Mobile Devices

We adopt a time-sharing multi-access protocol for the mobile devices similar to [TBZH19]. The transmission rate of the equipment k can be computed as,

$$r_k = B \ln(1 + \frac{h_k p_k}{N_0}) \tag{7.4}$$

where B is the bandwidth, N_0 is the background noise, p_k is the transmission power, and h_k is the channel gain of peer-to-peer link between the device k and the central server. Assume the fraction of communication time allocated to the device k is γ_k and the data size is s_k. Then the transmission rate of each device k can be computed as,

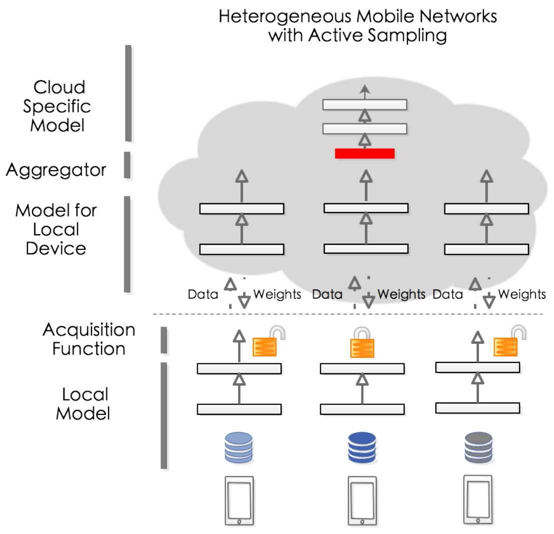


Figure 7.1: The overview of the proposed active sampling approach for training in heterogeneous mobile networks.

$$r_k = \frac{s_k}{\gamma_k} \tag{7.5}$$

To transmit the data the energy consumption is,

$$E_c = \gamma_k p_k \tag{7.6}$$

where the power function is,

$$p_k = \frac{N_0}{h_k}(e^{\frac{s_k/\gamma_k}{B}} - 1) \tag{7.7}$$

7.4 Active Sampling in Heterogeneous Mobile Networks

7.4.1 Background of Active Learning

Label acquisition for unlabeled data is expensive since it requires the participation of domain experts. One important topic in machine learning research is how to train an accurate predictive model based on as few labeled examples as possible. Active learning [Set09, Das11, KSF17], which reduces the labeling cost by querying the labels of the most informative examples, has attracted a lot of attention recently. Assume we have a dataset $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)\}$. The set of the examples is denoted as $D_\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$. In active learning, we aim to learn a probabilistic model $p(y|\mathbf{x}; \theta)$ using as few labeled examples as possible. To achieve this, we start with a subset $L \in D$ of the dataset and a large pool of unlabeled example $U_\mathbf{x} = D_\mathbf{x} \setminus L_\mathbf{x}$. We first train a model M_0 based on L. In each iteration of active learning, we pick an example $\mathbf{x} \in U_\mathbf{x}$ which can most improve the generalization ability of the current model. Then $\mathbf{x}$ is added to L and its label is given by an oracle at a cost. The oracle is typically a human annotator. We retrain the model again on L with the additional example $\mathbf{x}$. The above process is repeated until a predefined accuracy is met.

One way to evaluate the informativeness of the example is called *uncertainty sampling* [Set09]. In uncertainty sampling, an active learner queries the label of the example about which it is most uncertain how to label based on an acquisition function $U(\mathbf{x}; \theta)$.

Commonly used acquisition functions include:

- Entropy:

$$\mathbf{x}^*_{ENT} = \underset{\mathbf{x}\in U_{\mathbf{x}}}{\arg\max} - \sum_i p(y_i|\mathbf{x};\theta)\log p(y_i|\mathbf{x};\theta) \tag{7.8}$$

  where y_i ranges over all the possible labelings.

- Least confident:

$$\mathbf{x}^*_{LC} = 1 - \underset{\mathbf{x}\in U_{\mathbf{x}}}{\arg\min}\, p(y^*|\mathbf{x};\theta) \tag{7.9}$$

  where $y^* = \arg\max_{\mathbf{x}\in U_{\mathbf{x}}} p(y^*|\mathbf{x};\theta)$ is the most likely labeling.

- Confidence margin:

$$\mathbf{x}^*_{CM} = 1 - \underset{\mathbf{x}\in U_{\mathbf{x}}}{\arg\min}[p(y^*|\mathbf{x};\theta) - p(y^{**}|\mathbf{x};\theta)] \tag{7.10}$$

  where y^{**} is the second most likely labeling.

7.4.2 Proposed Approach

In the context of online learning, it is necessary to update the initial model in the central server to address the dynamics of the environment. However, naively sending all the local data to the central server incurs a large communication cost. In this work, we propose a communication and energy efficient active sampling algorithm for training in heterogeneous mobile networks. Suppose there are K devices, in the central server for each device k we construct a model f_{θ_k}. The output of all the models are concatenated via an aggregate function and are fed as input for a cloud specific model f_{θ_c}. The weights of all models in cloud is denoted as θ. On each device k, we construct a local model $\hat{f}_{\theta_k}$ with the same architecture as f_{θ_k}. On each round, the proposed algorithm identifies informative samples based on $\hat{f}_{\theta_k}$. This means only a subset of the devices need to communicate with the central server. On the other hand, in order to be updated, the central server models needs data from all the devices. For those devices which do not send data, we *synthesize* their data in the central server based on the available data from other devices. After the models are updated in the central server, the weights of model f_{θ_k} is sent to device k for synchronization. Finally, in order to prevent one device from dominating the data communication, we propose a lightweight load balancing mechanism to promote each device to communicate roughly the same which can increase the lifetime of the overall system. The overview of the proposed architecture is shown in Figure 7.1.

Design of Acquisition function

We assume that there is an initial model M_0 in the central server, we aim to update M_0 based on the data from all the devices. At each timestamp t, device k collects data $\mathbf{x}_t^k$. The data from all the devices are denoted as $\mathbf{x}_t$. The collection of data without $\mathbf{x}_t^k$ is denoted as $\mathbf{x}_t \setminus \mathbf{x}_t^k$. Our goal is to update the central server model with as few as communication rounds as possible. To achieve this, we propose active sampling to measure the informativeness of $\mathbf{x}_t^k$ based on the local model on each device. Instead of sending all the local measurements to the central server, we only consider the informative ones. We propose two acquisition functions, local uncertainty and delayed global uncertainty, for applying active sampling in heterogeneous mobile network.

Local Uncertainty. In local uncertainty (LU), we assume the devices are independent with each other when evaluating the uncertainty locally, that is, the acquisition function $U(\mathbf{x};\theta)$ only depends on $\mathbf{x}_k$ and θ_k. On each round t, the central server sends the weight θ_k to the corresponding device k. Device k updates the local model and evaluates the informativeness of $\mathbf{x}_t^k$ via an uncertainty function $U(\mathbf{x}_t^k;\theta_k)$ which can be the entropy function, least confident function or confidence margin function. Device k sends $\mathbf{x}_t^k$ to the central server only if the local uncertainty $U(\mathbf{x}_t^k;\theta_k)$ is above a given threshold γ. The independence assumption leads to a great reduction in communication cost – the overhead is that on each round the central server needs to send θ_k to each device k which is tolerable since the dimension of θ_k is usually small.

Delayed Global Uncertainty. In delayed global uncertainty (DGU), at each timestamp t, the central server sends the weights of all the models θ and $\mathbf{x}_{t-1} \setminus \mathbf{x}_{t-1,k}$ to device k. Together with $\mathbf{x}_t^k$, we form a new feature vector, denoted by $[\mathbf{x}_t^k, \mathbf{x}_{t-1} \setminus \mathbf{x}_{t-1}^k]$, which is $\mathbf{x}_{t-1}$ with $\mathbf{x}_{t-1}^k$ replaced by $\mathbf{x}_t^k$. In DGU, device k evaluates the informativeness of $\mathbf{x}_{t,k}$ via the function $U([\mathbf{x}_t^k, \mathbf{x}_{t-1} \setminus \mathbf{x}_{t-1}^k];\theta)$. Compared with the true global uncertainty $U(\mathbf{x}_t;\theta)$, it is called delayed global uncertainty since all the measurements have a lag of one timestamp except $\mathbf{x}_t^k$. The delayed global uncertainty can be seem as a better approximation to the true global uncertainty, however it increases the downlink communication cost.

The proposed active sampling approaches trade downlink communication cost for uplink communication cost. Although on each round the central server needs to communicate with the devices, the devices send data to the central server only if the data is informative. Examples of informative data include readings that indicate the device is misbehaving or the user activity is changing – both cases are only a small portion of the overall data stream. Due to the fact that the download speed is typically 5x faster than the upload speed, we can leverage the extra download bandwidth to reduce the upload congestion and increase the battery life of the devices.

Analysis

We provide analysis of DGU based on generalized linear models and the *smoothness* of the measurements.

Definition 7.4.1 *The data of device k are smooth if there exists a constant ρ_k and,*

$$\|\boldsymbol{x}_t^k - \boldsymbol{x}_{t-1}^k\|_2 \leq \rho_k \tag{7.11}$$

Definition 7.4.2 *A function $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$ is said to be have Lipschitz constant C if for all $\mathbf{x} \in \mathbb{R}^m$ and $\mathbf{y} \in \mathbb{R}^m$, it satisfies,*

$$\|f(\mathbf{x}) - f(\mathbf{y})\|_2 \leq C\|\mathbf{x} - \mathbf{y}\|_2 \tag{7.12}$$

It can be shown that the softmax function used in the multinomial logistic regression is Lipschitz with constant 1.

Lemma 7.4.1 *Suppose function $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$ and $g : \mathbb{R}^l \rightarrow \mathbb{R}^m$ are Lipschitz function with Lipschitz constant C_1 and C_2 respectively. Then the function $h : f \circ g : \mathbb{R}^l \rightarrow \mathbb{R}^n$ is Lipschitz with constant $C_3 \leq C_1C_2$*

Theorem 7.4.2 *In a distributed heterogeneous mobile network with K sensors, assume the data of each device satisfy the smoothness requirements. We aim to update a generalized linear model $M(\mathbf{x};\theta)$ online with delayed global uncertainty. Assume $M(\mathbf{x};\theta)$ is Lipschitz with constant C_1 and given an uncertainty function which is Lipschitz with constant C_2, at timestamp t the delayed global uncertainty of device k and true global uncertainty is related as,*

$$|U((\mathbf{x}_t^k, \mathbf{x}_{t-1} \setminus \mathbf{x}_{t-1}^k);\theta) - U(\mathbf{x}_t;\theta)| \leq C_1C_2 \sum_{i=1, i\neq k}^{K} ||\theta_i||_2\rho_i \tag{7.13}$$

The proof of theorem 7.4.2 follows from the smoothness of the data the Lipschitz property of the models.

Synthesized measurements

With active sampling, only a subset of the devices send data to the central server. However, the central server needs data from all the devices to update the model. We leverage the fact that the data of nearby devices are closely related. For the missing data, we predict their values using the available data from other devices.

On each round t, if the central server did not receive the data from device k, we use a model $S(\mathbf{x}_t^k|\mathbf{x}_t^j, y_t; \beta)$ with parameters β, typically a neural network, to predict the missing data of device k with the measurement received from device j. Note that $\mathbf{x}_t^k$ is not only modeled as a function of $\mathbf{x}_t^j$, but is also conditioned on the label y_t of the data. The label information provides extra supervision for training the synthesized model S to better recover the missing data. The choice of device j depends on the actual deployment and characteristics of the datasets. For example, we can choose the device which has the highest sampling rate, since it collects most fine-grained measurements. With the synthesized measurements, we can update the central server model by only relying on a small subset of informative local data.

Load balance

Excessive data communication is the main cause of the energy consumption in mobile and sensor networks [Mad03, GBT$^+$04]. It is thus critical to prevent one device from dominating the data communication which can shorten the lifetime of the overall system. To achieve this, we propose a lightweight load balancing mechanism to promote each device to communicate the same number of bits during the online update. The proposed load balancing mechanism adjusts the uncertainty of the data collected from a particular device based on the number of times this device has communicated before.

Suppose there are K devices in the mobile network. In the central server we maintain a vector $M \in \mathbb{R}^K$ which is initialized to be zeros. On each round t as the central server receives data from device k, $M[k]$ increases its stored value by 1. We normalize L via a softmax function to obtain M_{norm}. $M_{norm}[k]$ can be regarded as the proportion of

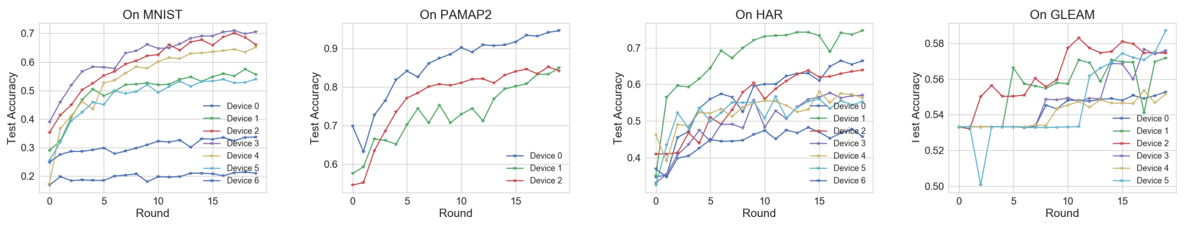


Figure 7.2: Results of local prediction on the four datasets during the online update.

data sent from device k. Then the central server sends $L_{norm}[k]$ to device k. With local uncertainty the informativeness of $\mathbf{x}_t^k$ is calculated as $(\frac{1/K}{M_{norm}[k]})^{\lambda_k} U(\mathbf{x}_t^k; \theta_k)$, where λ_k is a hyperparameter which can be used to adjust the strength of the load balancing. Intuitively, if $M_{norm}[k]$ is greater than $1/K$, which means that device k communicates more bits than the average, we scale down the informativeness of $\mathbf{x}_t^k$. With the proposed load balancing mechanism, the communication cost is distributed evenly across all the devices.

7.5 Analysis and Experiments

In the section, we conduct several experiments based on numerical simulation and practical deployment to address the following research questions that are aimed at understanding how the proposed active sampling works for distributed training in heterogeneous mobile networks,

- Do we need measurement from all the devices to learn an accurate predictive model in the central server?
- How does different active sampling methods perform for distributed training in het-

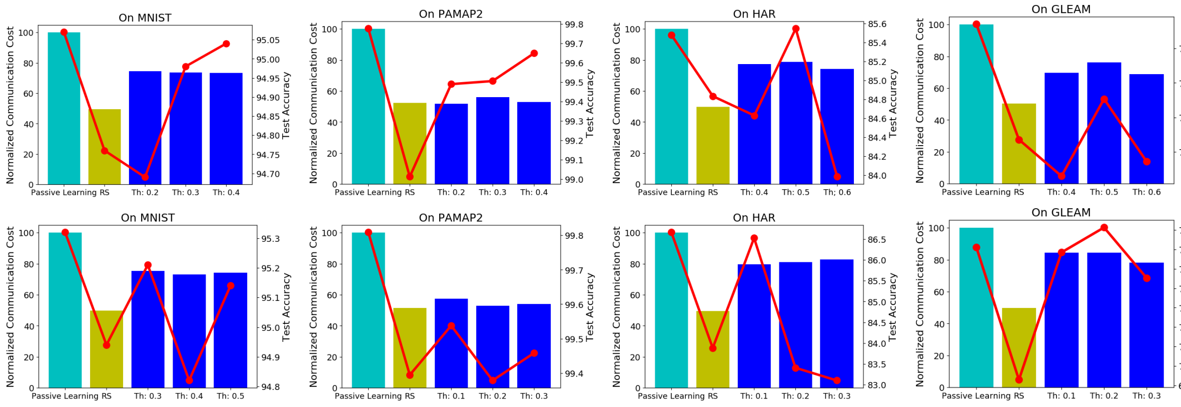


Figure 7.3: Normalized communication cost and final test accuracy with local uncertainty.

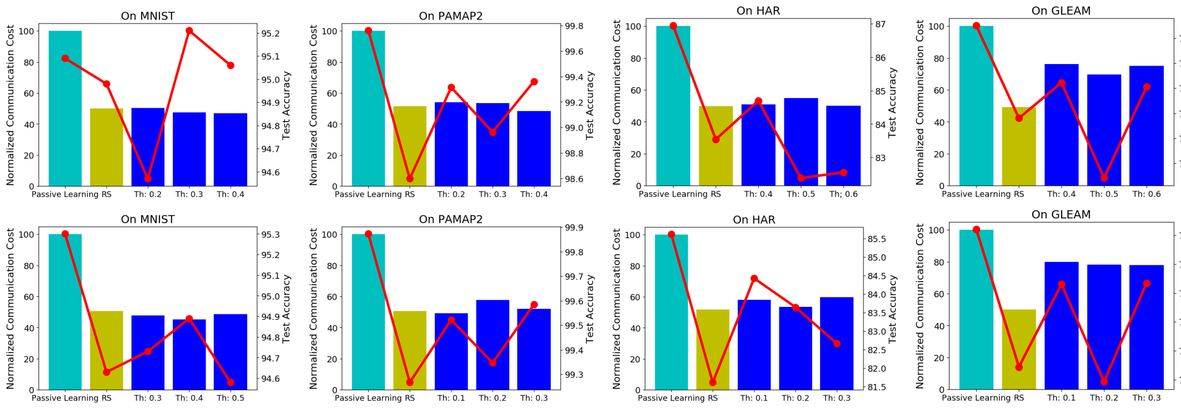


Figure 7.4: Normalized communication cost and final test accuracy of delayed global uncertainty.

erogeneous mobile networks?

- How does the uncertainty evolve during the online update?
- How does the proposed load balance algorithm affect the communication cost of individual devices?

7.5.1 Datasets

We consider the following four real-world applications in heterogeneous mobile networks. The detailed statistics of the datasets are listed in table 7.1. All the datasets are publicly available.

- **MNIST** [LBBH98]: In mobile networks, we may have multiple devices which record different angles or parts of an object. To identify the object in the image, we need to combine recordings from all the devices to form a single feature vector. We use the MNIST dataset, which is the canonical dataset used for digit classification, as a proof of concept. The MNIST dataset consists of hand-written digits from 0 to 9. Each image has 784 dimensions. We assume there are 7 devices, and each records 122 dimensions of the original image. The task is to identify the digit in the image in the test dataset by utilizing the recordings from all the 7 devices.

- **PAMAP2** [RS12]: This is a physical activity recognition dataset which contains 18 different physical activities performed by 9 subjects. There are a total of 4 sensors, 3 inertial measurement units and a heart rate monitor. We consider the 5 main activities {lying, sitting, standing, walking and running} and use the data from subject 1 in the experiments.

- **HAR** [AGO$^+$]: This is a recently introduced human activity recognition dataset. It contains data collecting from 17 mobile sensors. There are a total of 30 subject performing 6 activities {walking, walking_upstairs, walking_downstairs, sitting, standing, laying}. For consistency, we use the first 7 sensors in the experiments. The data partition is the same as in [AGO+].

- **Google Glass** (GLEAM) [RMHK15]: The GLEAM dataset consists of two hours of high resolution sensor data collected using Google Glass. There are 6 sensors and 38 subjects. The subjects conduct 6 activities {eating, talking, drinking, walking, going climbing stairs, computer_work} in a controlled environment. We use the data from the first subject as a case study. 40% of the examples are used as the training set and the rest is used as the test set.

7.5.2 Evaluation Protocol

In the experiments, 10% of the training examples are used to train the initial model M_0 in the central server and the rest is used for online update. We run the online update for a total of 1000 rounds. One each round, we use the proposed active sampling methods to decide whether or not to send the corresponding data. The central server model is updated every 50 rounds, also called a batch. The total number of batches is 200. We consider the following three metrics for comparing different methods,

- Communication cost: the total number of bits communicated by all the devices. It is the sum of the bits communicated by each individual device during the online update.

- Test accuracy: the accuracy of model on the test dataset after the online update.

- Energy consumption: the energy consumption of the all the devices during the online update. The energy consumption consists of the energy consumption for both computation and communication.

Table 7.1: Datasets statistics. TR: Number of training samples, TE: Number of test samples, C: Number of classes, S: Number of devices, D: Dimension of the feature vector.

Dataset	TR	TE	C	S	D
MNIST [LBBH98]	60,000	10,000	10	7	784
PAMAP2 [RS12]	4,182	6,275	5	3	52
HAR [AGO+]	7,352	2,947	6	17	561
GLEAM [RMHK15]	8,265	12,399	6	6	18

7.5.3 Baselines

We consider the following baselines in the experiments,

- Local Prediction (LP): we only utilize the data on each device to predict the labels of the test dataset. The total communication cost is minimal in this case, however the accuracy suffers since we cannot use the information from all the devices.

- Passive Learning (PL): In PL, each device sends all the data to the central server which incurs a large communication cost.

- Random Sampling (RS): We consider sending $\gamma\%$ ($\gamma = 50$) of the local data collected by each device to the central server. On each round, we sample from a Bernoulli($\gamma/100$) and decide whether or not to send the measurement based on the sampling output.

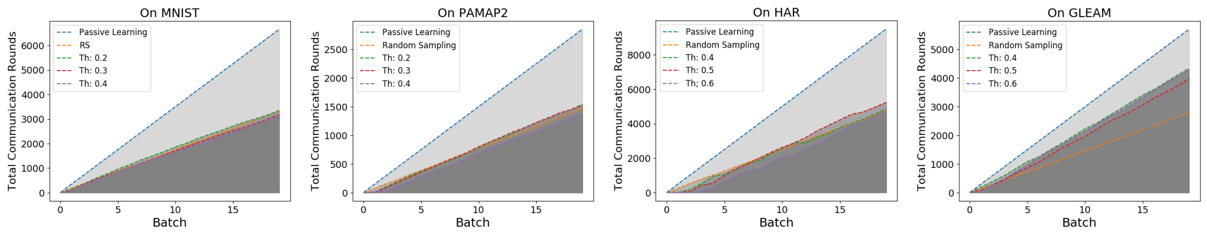


Figure 7.5: Total communication rounds as a function of batch. We use local uncertainty and least confident function with different thresholds (Th).

7.5.4 Implementation details

The neural network for each device consists of three layers with one layer for computing the local probability. The cloud specific model consists of two layers. We adopt Adam as the optimizer. The learning rate is set to be 0.01 and decays exponentially with a factor of 0.9. We train the initial models for a total of 10 epochs. No regularization is used in the experiments. All the models are implemented in Tensorflow [ABC$^+$16].

We train the model $S(\mathbf{x}_k|\mathbf{x}_j, y; \beta)$ for synthesizing measurements on the training dataset using mean squared error. For simplicity, we use the measurements from device 1 to predict the measurements for all other devices. During the online update, if the data of device 1 is missing, we impute the missing value with the historical average.

7.5.5 Numerical Results

Do we need measurement from all sensors to learn an accurate predictive model in the central server? In Fig 7.2, we show the results of local predictions on the four benchmarks. We make two observations based on the results. First, if we only train models locally using the data from one device, the models suffer great loss in accuracy. This is due to the fact that each device only captures a subset of the full feature vector which cannot approximate the underlying distribution between the input and the label. This also indicates the necessity to communicate the local data to a central server and train models based on data from all the devices. Second, the accuracy consistently improves with more and more incoming data. This implies the importance to adjust the trained model with newly collected measurements.

How does different active sampling methods work for distributed training in heterogeneous mobile networks? In Figure 7.3 - Figure 7.4, we show the results of the proposed active sampling methods on the four datasets with different uncertainty functions. For the bar plots, the total communication round is normalized against the passive learning. The normalized communication cost and test accuracy is calculated after the online update. The bars show the communication cost of different methods. Blue bars show the results of the proposed methods with different thresholds (Th). The red line is the test accuracy. First row: delayed global uncertainty with least confident function. Second row: delayed global uncertainty with confidence margin function. Yellow bar: random sampling (RS). The results were averaged over 5 runs.

Active Sampling vs. Passive Learning As shown in Figure 7.3 and Figure 7.4, we note that, as expected, compared with passive learning the proposed active learning methods can achieve the same test accuracy with much less communication cost. This is because in each round the proposed active sampling methods only send informative data about

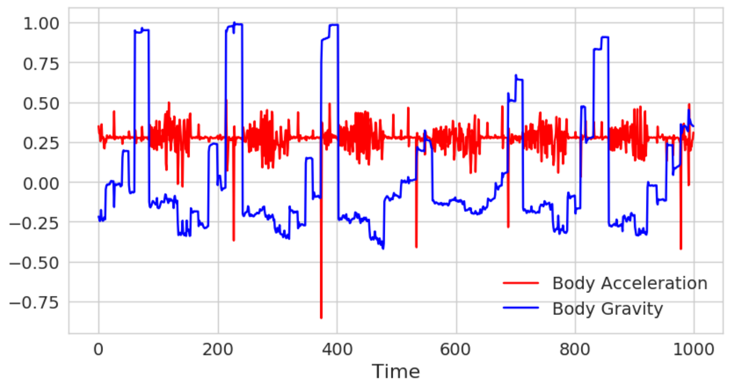


Figure 7.6: A comparison of sensor measurements recording body acceleration and chest acceleration shows a high correlation.

which the current central server model are most uncertain how to label. Thus the central server model can be better adapted to the incoming user data with the proposed approach. Thus we can greatly reduce the communication cost with the proposed approaches without accuracy loss for training in heterogeneous mobile networks.

Active Sampling vs. Random Sampling The proposed active sampling methods demonstrates a clear improvement over random sampling on the benchmarks in terms of test accuracy. Unlike the proposed active sampling method, naive random sampling fails to capture the important data and cannot identify the change points of user activities.

Acquisition Function Interestingly, we also observe that different acquisition functions yield similar performance on different datasets. This implies that the proposed active sampling methods are robust to the choices of acquisition functions, though the thresholds need to be adjusted to balance the trade-off between communication cost and test accuracy. The insensitivity of proposed method to the choice of acquisition functions is of great

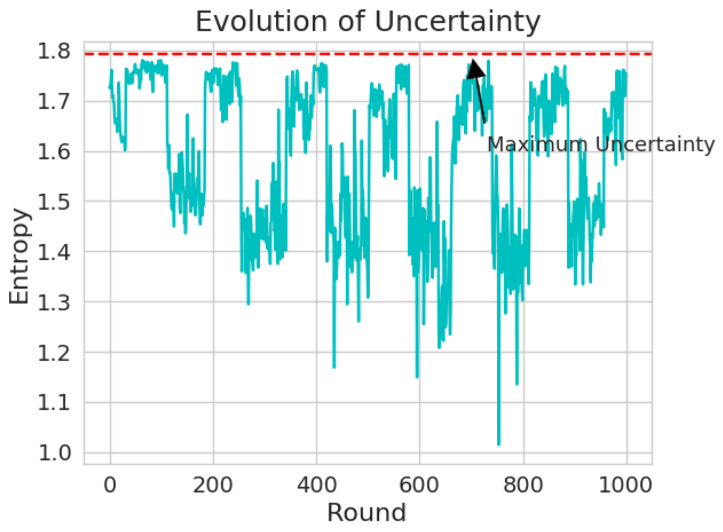


Figure 7.7: The evolution of uncertainty during the online updates shows that only a small portion of the measurements are informative.

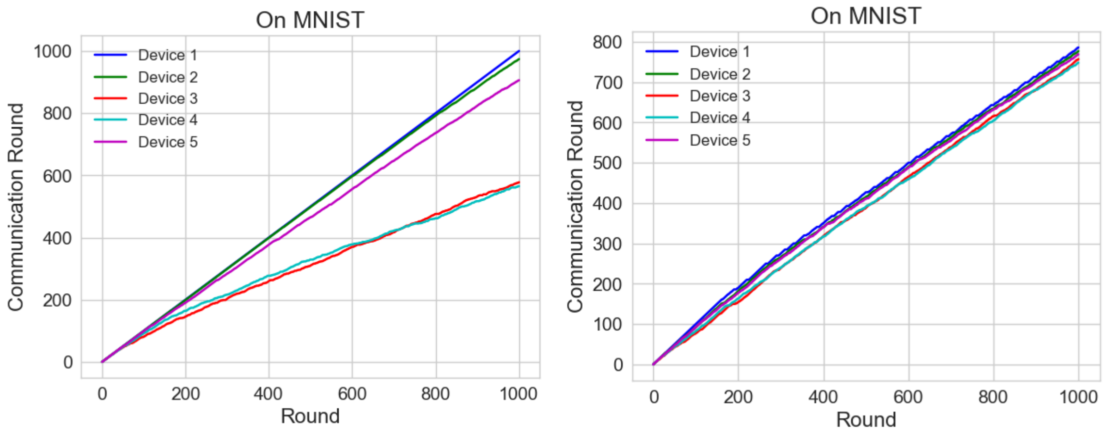


Figure 7.8: Left: communication round of the selected sensors during the online update without load balancing. Right: the results of applying the proposed load balancing mechanism.

importance for practical applications since it reduces the time for searching the optimal configurations. We omit the results of using entropy function due to the limitation of space.

LU vs. DGU Finally when we compare local uncertainty (LU) with delayed global uncertainty (DGU), we note that at the end of the online update, LU typically sends more data to the central server than DGU which leads to a higher test accuracy. We conjecture that this is because in DUG the data from other devices decrease the uncertainty of the newly collected data which results in a lower upload communication cost and test accuracy compared with LU. These results suggest that, we can select the methods based on the actual network condition and the test accuracy requirement. If the download bandwidth is high enough to communicate the sensor data from the central server to the local sensors, DGU allows us to trade for the excessive download bandwidth for a lower upload communication cost. Otherwise, LU can achieve a better test accuracy with a slightly increase in upload communication cost.

Total communication rounds vs. Batches In Figure 7.5 we show how the total communication rounds change with respect to the online batch on different datasets with local uncertainty and least confident function. It is clearly that the communication cost of the passive learning grows much faster than the proposed methods. With the proposed active sampling methods, the communication cost stops increasing when the local models are confident about the predictions.

How does the uncertainty evolve during the online update? The previous results show that only by sending the informative data on each device, we can still train an accurate predictive model in the central server. In this section we investigate how uncertainty evolves during the online update which allows us to gain fundamental insights on the proposed method. Figure 7.7 shows an example of how the uncertainty evolves on the ankle accelerometer sensor of the *PAMAP2* dataset using LU with entropy function. We first note that the uncertainty of different data varies over a large range and only a small portion of the measurements are highly informative. We also note the uncertainty has a cyclic pattern which can be attributed to the transition of user activities. Our proposed method can easily capture such transition and leverage the redundancy in the data to reduce the communication cost. We further show the relation of two sensors in Figure 7.6. We can see that the data are highly correlated and it is thus not necessary to communicate all the local data to the central server.

How does the proposed load balancing algorithm affect the communication cost of individual sensors? During the online update process, we observe that there is an obvious imbalance of the data communication between different devices: some devices communicate much more devices than other devices. To solve this problem, the proposed load balance algorithm distributes the communication cost across different devices by adjusting the informativeness of the data based on the received data in the central server.

We show the results of applying the proposed load balancing algorithm in Figure 7.8. We note that with the proposed algorithm, all the sensors deliver roughly the same number of measurements at the end of the online update. This reduces the risk of depleting the battery life of one or more sensors too quickly. Meanwhile, the proposed load balance algorithm has no additional overhead and can be easily integrated within the active sampling framework.

7.5.6 Energy Saving Based on the Device Models

In this section, we analyze the energy saving of the proposed active sampling compared with passive learning approach based on the device models. Suppose in each online update round, device k needs to process n samples and each sample is represented with 32bits. For the passive learning approach, the energy needed to transfer n samples can be computed as,

$$E_p = \frac{32n}{r_k} p_k \tag{7.14}$$

For the active learning approach, suppose only $100w\%$ of the samples are being sent and the number of the weight parameters is s, the total energy for computation and communication can be derived as,

$$E_a = \frac{\alpha_k s}{2} C_k f_k^2 + \frac{32nw}{r_k} p_k \tag{7.15}$$

The energy saving of the active sampling approach can be computed as,

$$E_s = (1-w)\frac{32n}{r_k} p_k - \frac{\alpha_k s}{2} C_k f_k^2 \tag{7.16}$$

In summary, the energy saving of the proposed approach can be easily derived from the CPU specification and the properties of the communication channel.

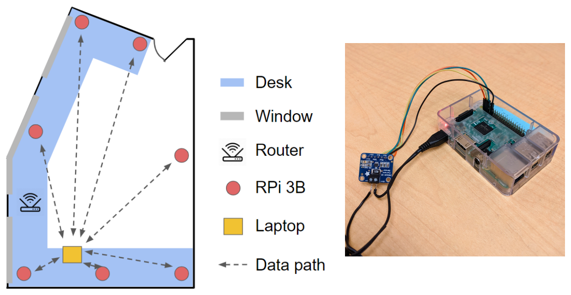


Figure 7.9: Left: deployment topology of the prototype network. Right: configuration of each RPi 3B, which has a current sensor INA219 attached to it.

7.6 Practical Deployment

7.6.1 Setup

To demonstrate the benefits the proposed active sampling method for real-world applications, we deploy a prototype network in our lab and measure the energy and communication saving of the proposed active learning approach compared with the passive learning approach. We consider a common topology setting in various Internet-of-Things (IoT) or mobile applications such as Smart Home, Smart Building, etc. As shown in Figure 7.9, the prototype network locates in our 6m$\times$10m lab, consisting of seven Raspberry Pi 3B (RPi 3B) devices and one conventional laptop as a central server. All devices are connected to a router and form a local heterogeneous mobile network. The number of used local devices is more than twice of the setup used in [WTS$^+$18] for validating federated learning. The RPis communicate with the central server using the MQTT protocol which is commonly used in IoT and mobile applications. The network bandwidth is controlled by the *wondershaper* tool [won]. We experiment with communication bandwidths of 200, 700 and 5000 kbps, which correspond to typical bandwidths of constrained Bluetooth,

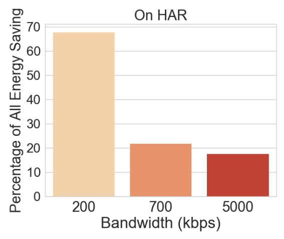
Bluetooth, and WiFi, respectively. For energy measurement, we attach a high side current sensor INA219 [ina] to each RPi 3B, as depicted in Figure 7.9. INA219 reports high side voltage and DC current draw with 1% precision.

The experiments are conducted based on two applications, image classification and human activity recognition with the MNIST dataset and the HAR dataset, respectively. For image classification, we evenly distribute the feature vector on each device. For human activity recognition, each device have the measurements of one particular sensor. The neural network architecture is consistent with the setup described in Section 7.5.4. We repeat the online update round for ten epochs and compare the active sampling approach with the passive learning approach in terms of energy consumption. For the active sampling approach, we use LU for simplicity as it has been shown that LU can achieve similar accuracy as DGU in Section 7.5.5. Entropy is used as the acquisition function for calculating the informativeness of the samples.

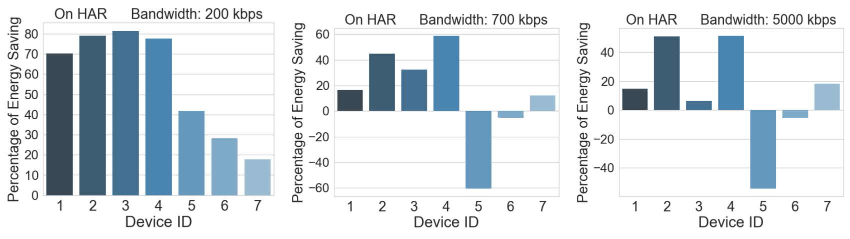
7.6.2 Results and Discussion

We compare the proposed active sampling approach and the passive learning approach in terms of the energy saving of all the devices and each individual device during the online updates.

All energy saving The results of energy saving of all the devices are shown in Figure 7.10a and 7.11a. It is clear that the proposed active sampling approach can greatly reduce the energy consumption of the system. For human activity recognition, the energy saving is 67.68% , 21.55%, and 17.34% with a bandwidth of 200 kbps, 700 kbps and 5000 kbps,

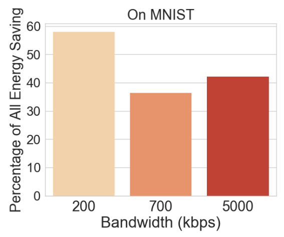


(a)

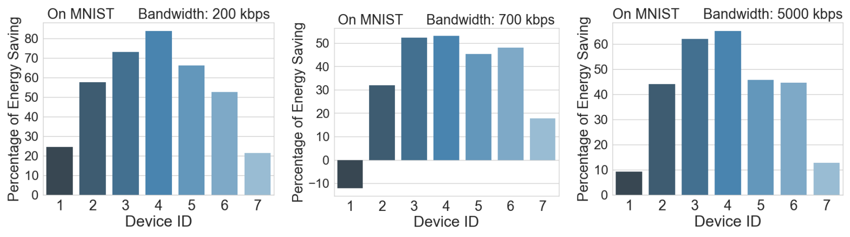


(b)

Figure 7.10: Energy saving for human activity recognition. Left: The energy saving of all devices. Right: the energy saving of each individual device.



(a)



(b)

Figure 7.11: Energy saving for image classification. Left: The energy saving of all devices. Right: the energy saving of each individual device.

respectively. For image classification, the energy saving is 58.01%, 36.37%, and 42.04% with a bandwidth of 200 kbps, 700 kbps and 5000 kbps, respectively. Notably the proposed active sampling approach achieves a large energy consumption reduction than the passive learning approach under a low bandwidth (200 kbps). This is particularly important for real-world mobile applications due to the network bandwidth is often limited. The proposed active sampling approach thus enables energy efficient machine learning on mobile devices or IoT devices.

Energy saving of each device We further show the energy saving of each individual device in Figure 7.10b and Figure 7.11b. It can be seen that due to the heterogeneity of the devices (since they have different features), different devices have different patterns

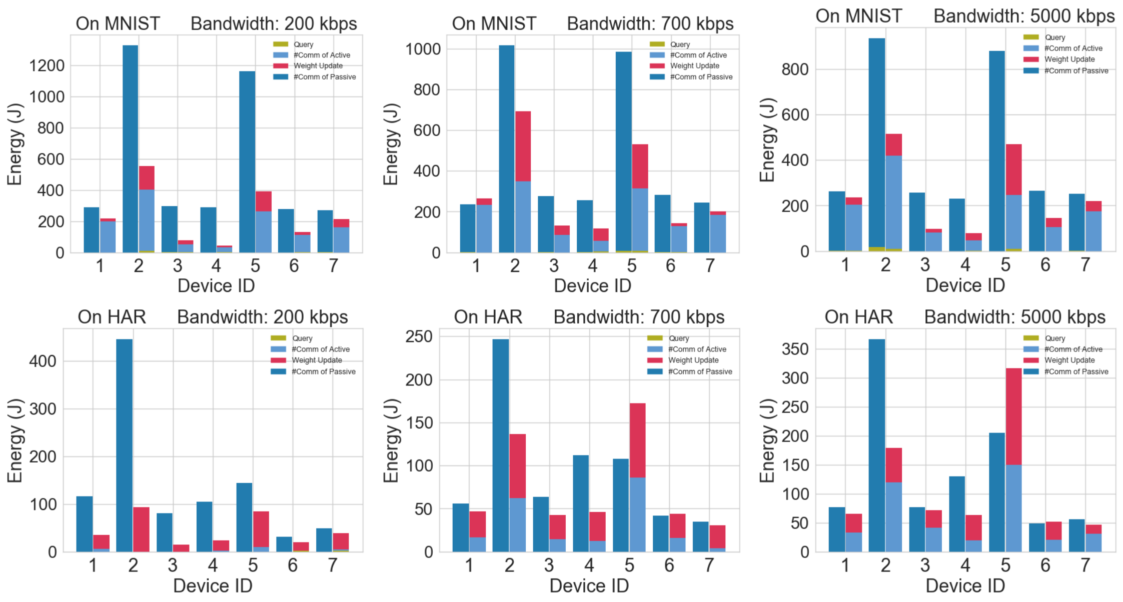


Figure 7.12: Energy consumption breakdown of the passive learning approach and the proposed active sampling approach.

of energy saving. Some devices even have a higher energy consumption with the active sampling approach due to the additional computation. This indicates that not all the devices are equally important for the task. With the proposed active sampling approach, we can leverage this fact to reduce the data communication of the unimportant devices to save the energy of the whole system.

Energy consumption breakdown To obtain a better understanding that why the proposed approach can reduce the energy consumption of the devices, we show the energy consumption breakdown of the proposed active sampling approach and the passive learning approach in Figure 7.12 under different network conditions. For the passive learning approach, the energy consumption consists of sample query and data communication. For the active sampling approach, the energy consumption consists of sample query, local weight update and data communication. We can see that for both approaches, the energy consumed by sample query is negligible. The total energy consumption of local weight update and data communication of the active sampling approach is usually less than the energy consumption of the passive learning approach. Essentially, the proposed active sampling approach trades computation for communication to reduce the energy consumption of the whole system.

7.7 Conclusion

In this chapter, we propose an active sampling method for communication efficient training in heterogeneous mobile networks involving data from multiple domains. Instead of sending all the local data to the central server, the proposed method identifies and sends only informative informative measurements. Experimental results show that the proposed active sampling methods can reduce the communication cost by up to 53% and energy consumption by up to 67% without accuracy degradation compared with the conventional approaches. In next chapter we consider the case that the model is trained on multiple domains sequentially which is also referred to as lifelong learning. The goal of learning with multiple domains sequentially is to allow the model to acquire the ability of maintaining the performance on old domains while learning a new domain.

This chapter contains material from “Active Sampling for Distributed Training in Heterogeneous Sensor Networks”, by Yunhui Guo, Xiaofan Yu, Kamalika Chaudhuri, Tajana Rosing, which appears in *The 16th International Conference on Mobility, Sensing and Networking (MSN)*, 2020. The dissertation author was the primary investigator and author of this paper.

Chapter 8

Learning across Multiple Domains Sequentially

8.1 Introduction

Different from the last chapter, in this chapter we consider learning across multiple domains sequentially which is also called lifelong learning [TM95, KPR$^+$17, PKP$^+$19]. A central dilemma in lifelong learning is how to achieve a balance between the performance on old tasks and the new task [KPR$^+$17, Rob95, LKJ$^+$17, SLKK17]. During the process of learning the new task, the originally learned knowledge will typically be disrupted, which leads to catastrophic forgetting. On the other hand, a learning algorithm biasing towards old tasks will interfere with the learning of the new task. Several lines of methods are proposed recently to address this issue. Examples include regularization based methods [KPR$^+$17, ZPG17, CDAT18], knowledge transfer based methods [RRD$^+$16] and episodic memory based methods [LP+17, CRRE18, RCA+18]. Especially, episodic memory based methods such as *Gradient Episodic Memory* (GEM) [LP+17] and *Averaged Gradient Episodic Memory* (A-GEM) [CRRE18] have shown remarkable performance. In episodic memory based methods, a small episodic memory is used for storing examples from old tasks to guide the optimization of the current task.

In this chapter, we present the first unified view of episodic memory based lifelong learning methods, including GEM [LP+17] and A-GEM [CRRE18], from an optimization's perspective. Specifically, we cast the problem of avoiding catastrophic forgetting as an optimization problem with composite objective. We approximately solve the optimization problem using one-step stochastic gradient descent with the standard gradient replaced by the proposed *Mixed Stochastic Gradient* (MEGA). We propose two different schemes, called MEGA-I and MEGA-II, which can be used in different scenarios. We show that both GEM [LP+17] and A-GEM [CRRE18] are degenerate cases of MEGA-I and MEGA-II which consistently put the same emphasis on the current task, regardless of how the loss changes over time. In contrast, based on our derivation, the direction of the gradients in the proposed MEGA-I and MEGA-II balance old tasks and the new task in an adaptive manner by considering the performance of the model in the learning process.

Our contributions are as follows. (1) We present the first unified view of current episodic memory based lifelong learning methods including GEM [LP+17] and A-GEM [CRRE18]. (2) From the presented unified view, we propose two different schemes, called MEGA-I and MEGA-II, for lifelong learning problems. (3) We extensively evaluate the proposed schemes on several lifelong learning benchmarks, and the results show that the proposed MEGA-I and MEGA-II significantly advance the state-of-the-art performance. We show that the proposed MEGA-I and MEGA-II achieve comparable performance in the existing setting for lifelong learning [CRRE18]. In particular, MEGA-II achieves an average accuracy of 91.21±0.10% on **Permuted MNIST**, which is 2% better than the previous state-of-the-art model. On **Split CIFAR**, our proposed MEGA-II achieves an average accuracy of 66.12±1.93%, which is about 5% better than the state-of-the-art method. (4) Finally, we show that the proposed MEGA-II outperforms MEGA-I when the number of examples per task is limited. We also analyze the reason for the effectiveness of MEGA-II over MEGA-I in this case.

8.2 Related Work

Several lifelong learning methods [DLAM+19, KMA+18] and evaluation protocols [FG18, HKCK18] are proposed recently. We categorize the methods into different types based on the methodology,

Regularization based approaches: EWC [KPR+17] adopted Fisher information matrix to prevent important weights for old tasks from changing drastically. In PI [ZPG17], the authors introduced *intelligent synapses* and endowed each individual synapse with a local measure of "importance" to avoid old memories from being overwritten. RWALK [CDAT18] utilized a KL-divergence based regularization for preserving knowledge of old tasks. While in MAS [ABE+18] the importance measure for each parameter of the network was computed based on how sensitive the predicted output function is to a change in this parameter. [AKT19] extended MAS for task-free continual learning. In [RBB18], an approximation of the Hessian was employed to approximate the posterior after every task. Uncertainties measures were also used to avoid catastrophic forgetting [EEDR19]. [FG19] proposed methods based on *approximate Bayesian* which recursively approximate the posterior of the given data.

Knowledge transfer based methods: In PROG-NN [RRD+16], a new "column" with lateral connections to previous hidden layers was added for each new task. In [LLSL19], the authors proposed a method to leverage unlabeled data in the wild to avoid catastrophic forgetting using knowledge distillation. [ZCCY19] proposed orthogonal weights modification (OWM) to enable networks to continually learn different mapping rules in a context-dependent way.

Episodic memory based approaches: In episodic memory based lifelong learning methods, a small reference memory is used for storing information from old tasks. GEM [LP+17] and A-GEM [CRRE18] rotated the current gradient when the angle between the current gradient and the gradient computed on the reference memory is obtuse. MER [RCA+18] is a recently proposed lifelong learning algorithm which employed a meta-learning training strategy. In [ALGB19], a line of methods are proposed to select important samples to store in the memory in order to reduce memory size. Instead of storing samples, in [FAML19] the authors proposed Orthogonal Gradient Descent (OGD) which projects the gradients on the new task onto a subspace in which the projected gradient will not affect the model's output on old tasks. [HJ18] proposed *conceptor aided backprop* which is a variant of the back-propagation algorithm for avoiding catastrophic forgetting.

Our proposed schemes aim to improve episodic memory based approaches and are most related to [CRRE18]. Different from [CRRE18], the proposed schemes explicitly consider the performance of the model on old tasks and the new task in the process of rotating the current gradient. Our proposed schemes are also related to several multi-task learning methods [SK18, KGC18, CBLR17]. In [SK18, KGC18], the authors aimed at achieving a good balance between different tasks by learning to weigh the loss on each task . In contrast, our schemes directly leverage loss information in the context of lifelong learning for overcoming catastrophic forgetting. Compared with [CBLR17], instead of using the gradient norm information, our schemes and [LP+17, CRRE18] focus on rotating the direction of the current gradient.

8.3 Background

8.3.1 Gradient Episodic Memory (GEM)

Gradient Episodic Memory (GEM) [LP+17] addresses the lifelong learning problem by utilizing a small episodic memory M_k for storing a subset of the examples from task k. The episodic memory is populated by choosing examples uniformly at random for each task. While training on task t, the loss on the episodic memory M_k can be computed as $\ell(w_t; M_k) = \frac{1}{|M_k|}\sum_{(x_i,y_i)\in M_k} \ell(f(x_i; w_t), y_i)$, where w_t is the weight of model during the training on task t. GEM ensures that each update on the t-th task will not increase the loss on the episodic memory, that is,

$$\text{minimize}_w \ell(w; D_t^{tr}) \quad \text{s.t.} \quad \ell(w; M_k) \le \ell(w_{t-1}; M_k) \quad \forall k < t \tag{8.1}$$

To inspect the increase of loss on the episodic memory, GEM computes the gradient g on the current task and the reference gradient g_k on the episodic memory M_k. When the angle between g and g_k is obtuse, GEM projects the current gradient g to have a right or acute angle with g_k,

$$\text{minimize}_{g_{\text{true}}} \frac{1}{2} \|g - g_{\text{true}}\|_2^2 \quad \text{s.t.} \quad g_{\text{true}}^\top g_{\text{k}} \ge 0 \quad \forall k < t \tag{8.2}$$

GEM solves above optimization problem via quadratic programming in the dual space with $v \in \mathbb{R}^{(t-1)\times 1}$:

$$\text{minimize}_v v^\top G^\top G v + g^\top G v \quad \text{s.t.} \quad v \ge 0 \tag{8.3}$$

where $G = -(g_1, ..., g_{t-1}) \in \mathbb{R}^{d\times(t-1)}$, $g \in \mathbb{R}^{d\times 1}$, and d is the number of parameters in the neural network. After obtaining the solution v^*, the gradient used for updating the model can be computed as $g_{true} = Gv^* + g$.

A-GEM [CRRE18] improves the efficiency of GEM by preventing the average episodic memory loss from increasing. In A-GEM, G is replaced by $-g_{ref}$ which is the gradient computed on a random subset of the examples from all old tasks. And v^* is replaced with a single scalar which can be computed in closed form as $\frac{g^\top g_{\text{ref}}}{g_{\text{ref}}^\top g_{\text{ref}}}$.

8.3.2 Evaluation Metrics

Average Accuracy and Forgetting Measure [CDAT18] are common used metrics for evaluating performance of lifelong learning algorithms. In [CRRE18], the authors introduce another metric, called Learning Curve Area (LCA), to assess the learning speed of different lifelong learning algorithms. In this chapter, we further introduce a new evaluation metric, called Long-term Remembering (LTR), to characterize the ability of lifelong learning algorithms for *remembering* the performance of tasks trained in the far past.

Suppose there are M_k mini-batches in the training set of task D_k. Similar to [CRRE18], we define $a_{k,i,j}$ as the accuracy on the test set of task D_j after the model is trained on the i-th mini-batch of task D_k. Generally, suppose the model $f(x; \mathbf{w})$ is trained on a sequence of T tasks $\{D_1, D_2, ..., D_T\}$. Average Accuracy and Forgetting Measure after the model is trained on the task D_k are defined as

$$A_k = \frac{1}{k}\sum_{j=1}^{k} a_{k,M_k,j} \quad F_k = \frac{1}{k-1}\sum_{j=1}^{k-1} f_j^k \tag{8.4}$$

where $f_j^k = \max_{l\in\{1,2,..,k-1\}} a_{l,M_l,j} - a_{k,M_k,j}$. Clearly, A_T is the average test accuracy and F_T assesses the degree of accuracy drop on old tasks after the model is trained on all the T tasks. Learning Curve Area (LCA) [CRRE18] at β is defined as,

$$\text{LCA}_\beta = \frac{1}{\beta+1}\sum_{b=0}^{\beta} Z_b \tag{8.5}$$

where $Z_b = \frac{1}{T}\sum_{k=1}^{T} a_{k,b,k}$. Intuitively, LCA measures the learning speed of different lifelong learning algorithms. A higher value of LCA indicates that the model learns quickly. We refer the readers to [CRRE18] for more details about LCA.

8.4 A Unified View of Episodic Memory Based Lifelong Learning

In this section, we provide a unified view for better understanding several episodic memory lifelong learning approach, including GEM [LP$^+$17] and A-GEM [CRRE18]. GEM [LP$^+$17] and A-GEM [CRRE18] address the lifelong learning problem by utilizing a small episodic memory M_k for storing a subset of the examples from task k. The episodic memory is populated by choosing examples uniformly at random for each task. While training on task t, the loss on the episodic memory M_k can be computed as $\ell_{\text{ref}}(w_t; M_k) = \frac{1}{|M_k|} \sum_{(x_i,y_i)\in M_k} \ell(f(x_i; w_t), y_i)$, where w_t is the weight of model during the training on task t.

In GEM and A-GEM, the lifelong learning model is trained via mini-batch stochastic gradient descent. We use w_k^t to denote the weight when the model is being trained on the k-th mini-batch of task t. To establish the tradeoff between the performance on old tasks and the t-th task, we consider the following optimization problem with composite objective in each update step:

$$\min_w \alpha_1(w_k^t)\ell_t(w) + \alpha_2(w_k^t)\ell_{\text{ref}}(w) := \mathbb{E}_{\xi,\zeta}\left[\alpha_1(w_k^t)\ell_t(w;\xi) + \alpha_2(w_k^t)\ell_{\text{ref}}(w;\zeta)\right], \tag{8.6}$$

where $w \in \mathbb{R}^d$ is the parameter of the model, ξ, ζ are random variables with finite support, $\ell_t(w)$ is the expected training loss of the t-th task, $\ell_{\text{ref}}(w)$ is the expected loss calculated on the data stored in the episodic memory, $\alpha_1(w), \alpha_2(w) : \mathbb{R}^d \mapsto \mathbb{R}_+$ are real-valued mappings which control the relative importance of $\ell_t(w)$ and $\ell_{\text{ref}}(w)$ in each mini-batch. Mathematically, we consider using the following update:

$$w_{k+1}^t = \arg\min_w \alpha_1(w_k^t) \cdot \ell_t(w; \xi) + \alpha_2(w_k^t) \cdot \ell_{\text{ref}}(w; \zeta). \tag{8.7}$$

The idea of GEM and A-GEM is to employ first-order methods (e.g., stochastic gradient descent) to approximately solve the optimization problem (8.7), where one-step stochastic gradient descent is performed with the initial point to be w_k^t:

$$w_{k+1}^t \leftarrow w_k^t - \eta \left(\alpha_1(w_k^t) \nabla \ell_t(w_k^t; \xi_k^t) + \alpha_2(w_k^t) \nabla \ell_{\text{ref}}(w_k^t; \zeta_k^t) \right), \tag{8.8}$$

where η is the learning rate, ξ_k^t and ζ_k^t are random variables with finite support, $\nabla \ell_t(w_k^t; \xi_k^t)$ and $\nabla \ell_{\text{ref}}(w_k^t; \zeta_k^t)$ are unbiased estimators of $\nabla \ell_t(w_k^t)$ and $\nabla \ell_{\text{ref}}(w_k^t)$ respectively. The quantity $\alpha_1(w_k^t) \nabla \ell_t(w_k^t; \xi_k^t) + \alpha_2(w_k^t) \nabla \ell_{\text{ref}}(w_k^t; \zeta_k^t)$ is referred to as the *mixed stochastic gradient*.

In A-GEM, $\nabla \ell_{\text{ref}}(w_k^t; \xi_k^t)$ is the reference gradient computed based on a random subset from the episodic memory M of all past tasks, where $M = \cup_{k<t} M_k$. And $\alpha_1(w_k^t)$ and $\alpha_2(w_k^t)$ can be written as,

$$\alpha_1(w_k^t) = 1, \alpha_2(w_k^t) = \mathbf{I}_{\langle \nabla \ell_{\text{ref}}(w_k^t; \zeta_k^t), \nabla \ell_t(w_k^t; \xi_k^t) \rangle \leq 0} \times \left(- \frac{\nabla \ell_t(w_k^t; \xi_k^t)^\top \nabla \ell_{\text{ref}}(w_k^t; \zeta_k^t)}{\nabla \ell_{\text{ref}}(w_k^t; \zeta_k^t)^\top \nabla \ell_{\text{ref}}(w_k^t; \zeta_k^t)} \right), \tag{8.9}$$

where $\mathbf{I}_u$ is the indicator function, which is 1 if u holds and otherwise 0.

In GEM, there are $t-1$ reference gradients based on the previous $t-1$ tasks respectively. In this case, $\nabla \ell_{\text{ref}}(w_k^t; \zeta_k^t) = [g_1, \ldots, g_{t-1}] \in \mathbb{R}^{d \times (t-1)}$ and $\alpha_2(w_k^t) \in \mathbb{R}^{t-1}$, where $g_1, \ldots, g_{t-1}$ are reference gradients based on $M_1, \ldots, M_{t-1}$ respectively. In GEM,

$$\alpha_1(w_k^t) = 1, \alpha_2(w_k^t) = v_*, \tag{8.10}$$

where v_* is the optimal solution for the quadratic programming problem (8.3.

As we can see from the formulation (8.9) and (8.10), both A-GEM and GEM set $\alpha_1(w) = 1$ in the whole training process. It means that both A-GEM and GEM always put the same emphasis on the current task, regardless of how the loss changes over time. During the lifelong learning process, the current loss and the reference loss are changing dynamically in each mini-batchs, and consistently choosing $\alpha_1(w) = 1$ may not capture a good balance between current loss and the reference loss.

8.5 Mixed Stochastic Gradient

In this section, we introduce Mixed Stochastic Gradient (MEGA) to address the limitations of GEM and A-GEM. We adopt the way of A-GEM for computing the reference loss due to the better performance of A-GEM over GEM. Instead of consistently putting the same emphasis on the current task, the proposed schemes allow adaptive balancing between current task and old tasks. Specifically, MEGA-I and MEGA-II utilize the loss information

Algorithm 2 MEGA, the proposed algorithm for lifelong learning. T is the number of tasks. n_t is the number of mini-batches of task t. M is the episodic memory. ξ_k^t is the k-th mini-batch of task t and y_k^t is the corresponding label. ζ_k^t is a random mini-batch from the episodic memory. w_k^t stands for the parameter after k-th mini-batch during the training of t-th task. $\ell_t(w_k^t; \xi_k^t)$ is the training loss calculated on ξ_k^t. $\ell_{\text{ref}}(w_k^t; \zeta_k^t)$ is the reference loss calculated on ζ_k^t. α_1 and α_2 are defined in Equation 8.7.

```
1: $M \leftarrow \{\}$
2: for $t \leftarrow 1$ to $T$ do
3:   for $k \leftarrow 1$ to $n_t$ do
4:     if $M \neq \{\}$ then
5:       $\zeta_k^t \leftarrow \textsc{sample}(M)$
6:       Option I (MEGA-I): choose $\alpha_1$ and $\alpha_2$ based on Equation 8.11.
7:       Option II (MEGA-II): choose $\alpha_1$ and $\alpha_2$ based on Equation 8.15.
8:     else
9:       Set $\alpha_1 = 1$ and $\alpha_2 = 0$.
10:    end if
11:    Update $w_k^t$ using Eq. 8.8.
12:    $M \leftarrow M \bigcup (\xi_k^t, y_k^t)$
13:    Discard the samples added initially if $M$ is full.
14:  end for
15: end for
```

during training which is ignored by GEM and A-GEM. In Section 8.5.1, we propose MEGA-I which utilizes loss information to balance the reference gradient and the current gradient. In Section 8.5.2, we propose MEGA-II which considers the cosine similarities between the update direction with the current gradient and the reference gradient.

8.5.1 MEGA-I

We introduce MEGA-I which is an adaptive loss-based approach to balance the current task and old tasks by only leveraging loss information. We introduce a pre-defined sensitivity parameter ϵ similar to [CDK+06]. In the update of (8.8), we set

$$\begin{cases} \alpha_1(w) = 1, \alpha_2(w) = \ell_{\text{ref}}(w;\zeta)/\ell_t(w;\xi) & \text{if } \ell_t(w;\xi) > \epsilon \\ \alpha_1(w) = 0, \alpha_2(w) = 1 & \text{if } \ell_t(w;\xi) \leq \epsilon, \end{cases} \tag{8.11}$$

Intuitively, if $\ell_t(w;\xi)$ is small, then the model performs well on the current task and MEGA-I focuses on improving performance on the data stored in the episodic memory. To this end, MEGA-I chooses $\alpha_1(w) = 0$, $\alpha_2(w) = 1$. Otherwise, when the current loss is larger than ϵ, MEGA-I keeps the balance of the two terms of mixed stochastic gradient according to $\ell_t(w;\xi)$ and $\ell_{\text{ref}}(w;\zeta)$. Intuitively, if $\ell_t(w;\xi)$ is relatively larger than $\ell_{\text{ref}}(w;\zeta)$, then MEGA-I put less emphasis on the reference gradient and vice versa.

Compared with GEM and A-GEM update rule in (8.10) and (8.9), MEGA-I makes an improvement to handle the case of overfitting on the current task (i.e., $\ell_t(w;\xi) \leq \epsilon$), and to dynamically change the relative importance of the current and reference gradient according to the losses on the current task and previous tasks.

8.5.2 MEGA-II

The magnitude of MEGA-I's mixed stochastic gradient depends on the magnitude of the current gradient and the reference gradient, as well as the losses on the current task and the episodic memory. Inspired by A-GEM, MEGA-II's mixed gradient is obtained from a rotation of the current gradient whose magnitude only depends on the current gradient.

The key idea of the MEGA-II is to first appropriately rotate the stochastic gradient calculated on the current task (i.e., $\nabla \ell_t(w_k^t; \xi_k^t)$) by an angle θ_k^t, and then use the rotated vector as the mixed stochastic gradient to conduct the update (8.8) in each mini-batch. For simplicity, we omit the subscript k and superscript t later on unless specified.

We use $\mathbf{g}_{\text{mix}}$ to denote the desired mixed stochastic gradient which has the same magnitude as $\nabla \ell_t(w; \xi)$. Specifically, we look for the mixed stochastic gradient $\mathbf{g}_{\text{mix}}$ which direction aligns well with both $\nabla \ell_t(w; \xi)$ and $\nabla \ell_{\text{ref}}(w; \zeta)$. Similar to MEGA-I, we use the loss-balancing scheme and desire to maximize

$$\ell_t(w; \xi) \cdot \frac{\langle \mathbf{g}_{\text{mix}}, \nabla \ell_t(w; \xi) \rangle}{\|\mathbf{g}_{\text{mix}}\|_2 \cdot \|\nabla \ell_t(w; \xi)\|_2} + \ell_{\text{ref}}(w; \zeta) \cdot \frac{\langle \mathbf{g}_{\text{mix}}, \nabla \ell_{\text{ref}}(w; \zeta) \rangle}{\|\mathbf{g}_{\text{mix}}\|_2 \cdot \|\nabla \ell_{\text{ref}}(w; \zeta)\|_2}, \tag{8.12}$$

which is equivalent to find an angle θ such that

$$\theta = \arg \max_{\beta \in [0, \pi]} \ell_t(w; \xi) \cos(\beta) + \ell_{\text{ref}}(w; \zeta) \cos(\tilde{\theta} - \beta). \tag{8.13}$$

where $\tilde{\theta} \in [0, \pi]$ is the angle between $\nabla \ell_t(w; \xi)$ and $\nabla \ell_{\text{ref}}(w; \zeta)$, and $\beta \in [0, \pi]$ is the angle between $\mathbf{g}_{\text{mix}}$ and $\nabla \ell_t(w; \xi)$. The closed form of θ is $\theta = \frac{\pi}{2} - \alpha$, where $\alpha = \arctan\left(\frac{k + \cos \tilde{\theta}}{\sin \tilde{\theta}}\right)$ and $k = \ell_t(w; \xi) / \ell_{\text{ref}}(w; \zeta)$ if $\ell_{\text{ref}}(w; \zeta) \neq 0$ otherwise $k = +\infty$. Here we give some discussions of several special cases of Eq. (8.13).

- When $\ell_{\text{ref}}(w; \zeta) = 0$, then $\theta = 0$, and in this case $\alpha_1(w) = 1$, $\alpha_2(w) = 0$ in (8.8), the mixed stochastic gradient reduces to $\nabla \ell_t(w; \xi)$. In the lifelong learning setting, $\ell_{\text{ref}}(w; \zeta) = 0$ implies that there is almost no catastrophic forgetting, and hence we can update the model parameters exclusively for the current task by moving in the direction of $\nabla \ell_t(w; \xi)$.

- When $\ell_t(w; \xi) = 0$, then $\theta = \tilde{\theta}$, and $\alpha_1(w) = 0$, $\alpha_2(w) = \|\nabla \ell_t(w; \xi)\|_2 / \|\nabla \ell_{\text{ref}}(w; \zeta)\|_2$, provided that $\|\nabla \ell_{\text{ref}}(w; \zeta)\|_2 \neq 0$ (define 0/0=0). In this case, the direction of the mixed stochastic gradient is the same as the stochastic gradient calculated on the data in the episodic memory (i.e., $\ell_{\text{ref}}(w; \zeta)$). In the lifelong learning setting, this update can help improve the performance on old tasks, i.e., avoid catastrophic forgetting.

After we find the desired angle θ, we can calculate $\alpha_1(w)$ and $\alpha_2(w)$ in Eq. (8.8) as below.

For notation simplicity, we use $\mathbf{g}$, $\hat{\mathbf{g}}$, a, b to replace $\nabla \ell_t(w; \xi)$, $\nabla \ell_{\text{ref}}(w; \zeta)$, $\alpha_1(w)$, $\alpha_2(w)$ respectively. If $\mathbf{g} = \hat{\mathbf{g}}$, then $a = 1$, $b = 0$. Otherwise, the goal is to solve

$$\begin{aligned} a\mathbf{g}^\top \mathbf{g} + b\mathbf{g}^\top \hat{\mathbf{g}} &= \|\mathbf{g}\|_2^2 \cos\theta \\ a\mathbf{g}^\top \hat{\mathbf{g}} + b\|\hat{\mathbf{g}}\|_2^2 &= \|\mathbf{g}\|\|\hat{\mathbf{g}}\| \cos(\tilde{\theta} - \theta) \end{aligned} \tag{8.14}$$

The solution of (8.14) is

$$\begin{aligned} a &= \frac{1}{\|\mathbf{g}\|_2^2\|\hat{\mathbf{g}}\|_2^2 - \mathbf{g}^\top \hat{\mathbf{g}}} \left[\|\hat{\mathbf{g}}\|_2^2\|\mathbf{g}\|_2^2 \cos\theta - (\mathbf{g}^\top \hat{\mathbf{g}})\|\mathbf{g}\|_2\|\hat{\mathbf{g}}\|_2 \cos(\tilde{\theta} - \theta)\right] \\ b &= \frac{1}{\|\mathbf{g}\|_2^2\|\hat{\mathbf{g}}\|_2^2 - \mathbf{g}^\top \hat{\mathbf{g}}} \left[-(\mathbf{g}^\top \hat{\mathbf{g}})\|\mathbf{g}\|_2^2 \cos\theta + \|\mathbf{g}\|_2^3\|\hat{\mathbf{g}}\|_2 \cos(\tilde{\theta} - \theta)\right] \end{aligned} \tag{8.15}$$

It is worth noting that different from GEM and A-GEM which always set $\alpha_1(w) = 1$, the proposed MEGA-I and MEGA-II adaptively adjust α_1 and α_2 based on performance of the model on the current task and the episodic memory. Please see Algorithm 2 for the summary of the algorithm.

8.6 Experiments

8.6.1 Datasets

In the experiments, we consider the following four conventional lifelong learning benchmarks,

- **Permuted MNIST** [KPR$^+$17]: this is a variant of standard MNIST dataset [LCB98] of handwritten digits with 20 tasks. Each task has a fixed random permutation of the input pixels which is applied to all the images of that task.

- **Split CIFAR** [ZPG17]: this dataset consists of 20 disjoint subsets of CIFAR-100 dataset [K$^+$09], where each subset is formed by randomly sampling 5 classes without replacement from the original 100 classes.

- **Split CUB** [CRRE18]: the CUB dataset [WBW$^+$11] is split into 20 disjoint subsets by randomly sampling 10 classes without replacement from the original 200 classes.

- **Split AWA** [CRRE18]: this dataset consists of 20 subsets of the AWA dataset [LNH09]. Each subset is constructed by sampling 5 classes with replacement from a total of 50 classes. Note that the same class can appear in different subsets. As in [CRRE18], in order to guarantee that each training example only appears once in the learning process, based on the occurrences in different subsets the training data of each class is split into disjoint sets.

We also include **Many Permutations** which is a variant of **Permuted MNIST** to introduce more non-stationality into the learning process. In **Many Permutations**, there are a total 100 tasks with 200 examples per task. The way to generate the tasks is the same as in **Permuted MNIST**, that is, a fixed random permutation of input pixels is applied to all the examples for a particular task.

8.6.2 Network Architectures

To be consistent with the previous works [LP$^+$17, CRRE18], for **Permuted MNIST** we adopt a standard fully-connected network with two hidden layers. Each layer has 256 units with ReLU activation. For **Split CIFAR** we use a reduced ResNet18. For **Split CUB** and **Split AWA**, we use a standard ResNet18 [HZRS16].

8.6.3 Baselines and Experimental Settings

We compare the proposed MEGA with several state-of-the-art lifelong learning methods,

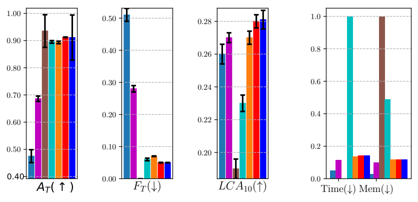
- VAN: in VAN, a single network is trained continuously on a sequence of tasks in a standard supervised learning manner.
- MULTI-TASK: in MULTI-TASK, a single network is trained on the shuffled data from all the tasks with a single pass.
- Episodic memory based approach: GEM [LP$^+$17] and A-GEM [CRRE18] are episodic memory based approaches which modify the current gradient when its angle between

the gradient computed on the episodic memory is obtuse. MER [RCA$^+$18] is another recently proposed episodic memory based approach which maintains an experience replay style memory with reservoir sampling and employs a meta-learning style training strategy.

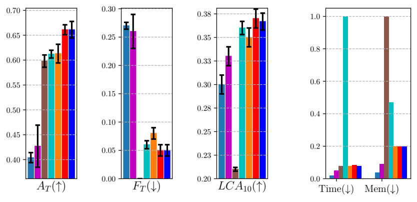
- Regularization-based approaches: EWC [KPR$^+$17], PI [ZPG17], RWALK [CDAT18] and MAS [ABE$^+$18] are regularization-based approaches which prevent the important weights of the old tasks from changing too much.

- Knowledge transfer based approach: in PROG-NN [RRD$^+$16], a new "column" with lateral connections with previous hidden layers is added for each new task. This allows knowledge transfer between old tasks and the new task.

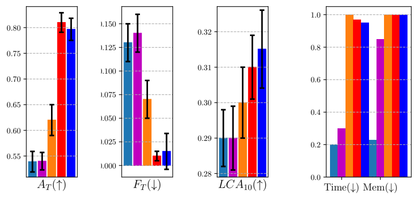
To be consistent with [CRRE18], for episodic memory based approaches, the episodic memory size for each task is 250, 65, 50, and 100, and the batch size for computing the gradients on the episodic memory (if needed) is 256, 1300, 128 and 128 for MNIST, CIFAR, CUB and AWA, respectively. To fill the episodic memory, the examples are chosen uniformly at random for each task as in [CRRE18]. For each dataset, 17 tasks are used for training and 3 tasks are used for hyperparameter search. For the baselines, we use the best hyperparameters found by [CRRE18]. For the detailed hyperparameters, please see Appendix G of [CRRE18]. For MER [RCA$^+$18], we reuse the best hyperparameters found in [RCA$^+$18]. In MEGA-I, the ϵ is chosen from $\{10^{-5:1:-1}\}$ via the 3 validation tasks. For MEGA-II, we reuse the hyperparameters from A-GEM [CRRE18]. All the experiments are done on 8 NVIDIA TITAN RTX GPUs.



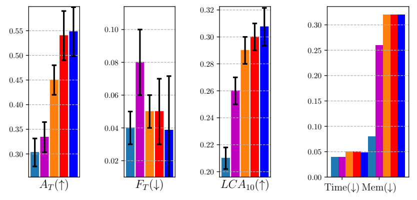
(a) Permuted MNIST



(b) Split CIFAR



(c) Split CUB



(d) Split AWA

Figure 8.1: Performance of lifelong learning models across different measures on **Permuted MNIST**, **Split CIFAR**, **Split CUB** and **Split AWA**.

8.7 Results

8.7.1 MEGA VS Baselines

In Fig. 8.1 we show the results across different measures on all the benchmark datasets. We have the following observations. First, MEGA-I and MEGA-II outperform all baselines across the benchmarks, except that PROG-NN achieves a slightly higher accuracy on **Permuted MNIST**. As we can see from the memory comparison, PROG-NN is very memory inefficient since it allocates a new network for each task, thus the number of parameters grows super-linearly with the number of tasks. This becomes problematic when large networks are being used. For example, PROG-NN runs out of memory on **Split CUB** and **Split AWA** which prevents it from scaling up to real-life problems. On other datasets, MEGA-I and MEGA-II consistently perform better than all the baselines.

From Fig. 8.2 we can see that on **Split CUB**, MEGA-I and MEGA-II even surpass the multi-task baseline which is previously believed as an upper bound performance of lifelong learning algorithms [CRRE18]. Second, MEGA-I and MEGA-II achieve the lowest Forgetting Measure across all the datasets which indicates their ability to overcome catastrophic forgetting. Third, MEGA-I and MEGA-II also obtain a high LCA across all the datasets which shows that MEGA-I and MEGA-II also learn quickly. The evolution of LCA in the first ten mini-batches across all the datasets is shown in Fig. 8.3. Last, we can observe that MEGA-I and MEGA-II achieve similar results in Fig. 8.1. For detailed results, please refer to Table 8.1 and Table 8.2.

In Fig. 8.2 we show the evolution of average accuracy during the lifelong learning process. As more tasks are added, while the average accuracy of the baselines generally drops due to catastrophic forgetting, MEGA-I and MEGA-II can maintain and even improve its performance. In the next section, we will show that MEGA-II outperforms MEGA-I when the number of examples is limited per task.

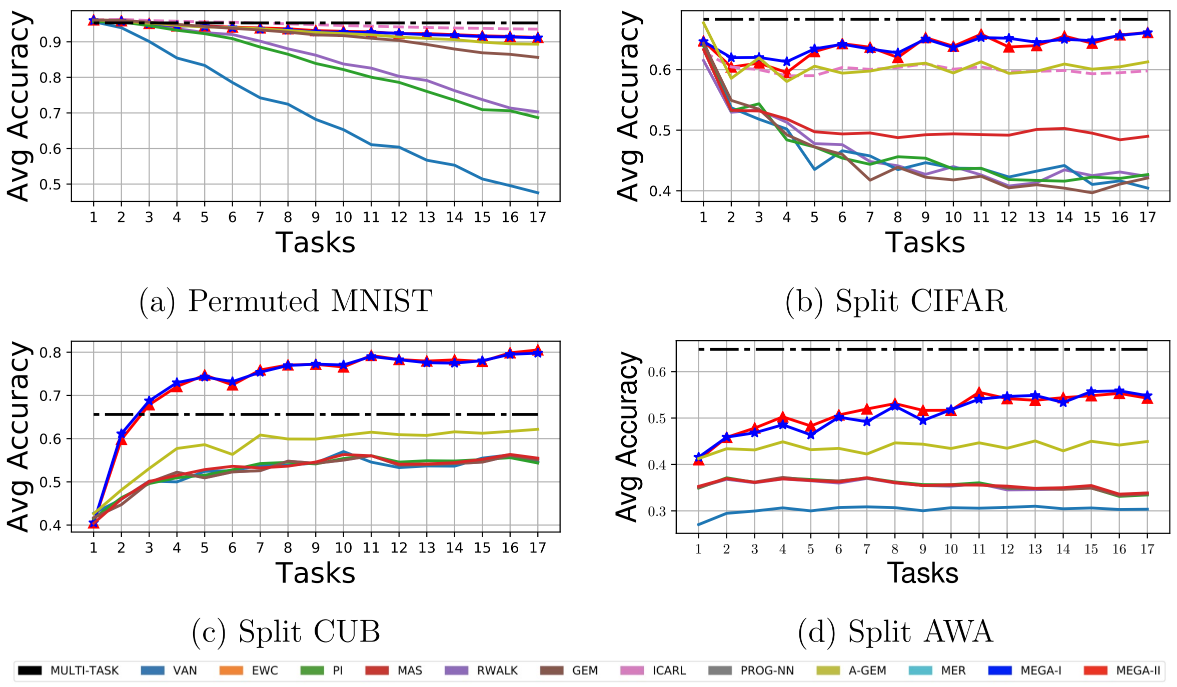
8.7.2 MEGA-II Outperforms Other Baselines and MEGA-I When the Number of Examples is Limited

Inspired by few-shot learning [SSZ17, VBL$^+$16, FAL17, GCK$^+$19], in this section we consider a more challenging setting for lifelong learning where each task only has a limited number of examples.

We construct 20 tasks with X number of examples per task, where $X = 200$, 400

Table 8.1: The results of Average Accuracy (A_T), Forgetting Measure (F_T) and LCA of different methods on **Permuted MNIST** and **Split CIFAR**. The results are averaged across 5 runs with different random seeds.

Methods	Permuted MNIST			Split CIFAR		
	$A_T(\%)$	F_T	LCA_{10}	$A_T(\%)$	F_T	LCA_{10}
VAN	47.55±2.37	0.52±0.026	0.259±0.005	40.44±1.02	0.27±0.006	0.309±0.011
EWC	68.68±0.98	0.28±0.010	0.276±0.002	42.67±4.24	0.26±0.039	0.336±0.010
MAS	70.30±1.67	0.26±0.018	0.298±0.006	42.35±3.52	0.26±0.030	0.332±0.010
RWALK	85.60±0.71	0.08±0.007	**0.319**±0.003	42.11±3.69	0.27±0.032	0.334±0.012
MER	-	-	-	37.27±1.68	0.03±0.030	0.051±0.101
PROG-NN	**93.55**±0.06	**0.0**±0.000	0.198±0.006	59.79±1.23	**0.0**±0.000	0.208±0.002
GEM	89.50±0.48	0.06±0.004	0.230±0.005	61.20±0.78	0.06±0.007	0.360±0.007
A-GEM	89.32±0.46	0.07±0.004	0.277±0.008	61.28±1.88	0.09±0.018	0.350±0.013
MEGA-I	91.10±0.08	0.05±0.001	0.281± 0.005	66.10±1.67	0.05±0.014	0.366±0.009
MEGA-II	91.21±0.10	0.05±0.001	0.283±0.004	**66.12**±1.94	0.06±0.015	**0.375**±0.012



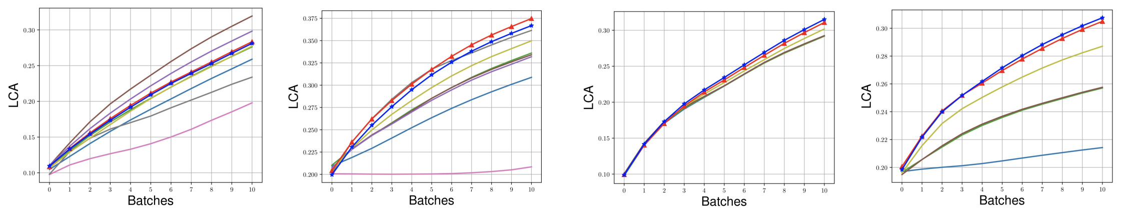
(a) Permuted MNIST (b) Split CIFAR (c) Split CUB (d) Split AWA

Figure 8.2: Evolution of average accuracy during the lifelong learning process.

and 600. The way to generate the tasks is the same as in **Permuted MNIST**, that is, a fixed random permutation of input pixels is applied to all the examples for a particular task. The running time is measured on one NVIDIA TITAN RTX GPU. The results

Table 8.2: The results of Average Accuracy (A_T), Forgetting Measure (F_T) and LCA of different methods on **Split CUB** and **Split AWA**. The results are averaged across 10 runs with different random seeds.

Methods	**Split CUB**			**Split AWA**		
	$A_T(\%)$	F_T	LCA_{10}	$A_T(\%)$	F_T	LCA_{10}
VAN	53.89±2.00	0.13±0.020	0.292±0.008	30.35±2.81	**0.04**±0.013	0.214±0.008
EWC	53.56±1.67	0.14±0.024	0.292±0.009	33.43±3.07	0.08±0.021	0.257±0.011
MAS	54.12±1.72	0.13±0.013	0.293±0.008	33.83±2.99	0.08±0.022	0.257±0.011
RWALK	54.11±1.71	0.13±0.013	0.293±0.009	33.63±2.64	0.08±0.023	0.258±0.011
PI	55.04±3.05	0.12±0.026	0.292±0.010	33.86±2.77	0.08±0.022	0.259±0.011
A-GEM	61.82±3.72	0.08±0.021	0.302±0.011	44.95±2.97	0.05±0.014	0.287±0.012
MEGA-I	79.67±2.15	0.01±0.019	**0.315**±0.011	**54.82**±4.97	0.04±0.034	**0.307**±0.014
MEGA-II	**80.58**±1.94	**0.01**±0.017	0.311±0.010	54.28±4.84	0.05±0.040	0.305±0.015



(a) Perm* MNIST (b) Split CIFAR (c) Split CUB (d) Split AWA

Figure 8.3: LCA of first ten mini-batches on different datasets, where "Perm*" stands for "Permutation".

🟦 VAN 🟪 EWC 🟧 A-GEM 🟦 MEGA-1 🟥 MEGA-2

of average accuracy are shown in Fig. 8.4(a). We can see that MEGA-II outperforms all the baseline and MEGA-I when the number of examples is limited. In Fig. 8.4(b), we show the execution time for each method, the proposed MEGA-I and MEGA-II are computational efficient compared with other methods. Compared with MER [RCA+18] which achieves similar results to MEGA-I and MEGA-II, MEGA-I and MEGA-II is much more time efficient since it does not rely on the meta-learning procedure.

We analyze the reason why MEGA-II outperforms MEGA-I when the number of examples is small. In this case, it is difficult to learn well on the current task, so the

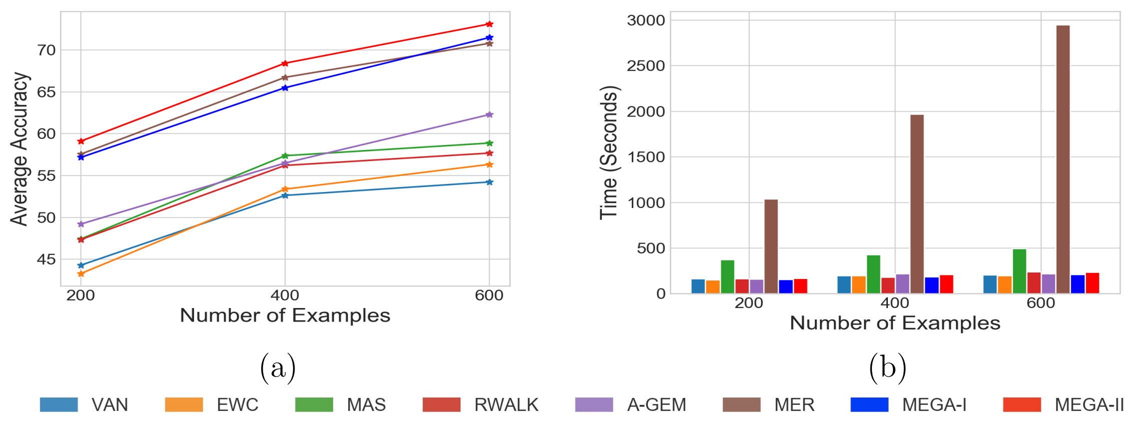


Figure 8.4: The average accuracy and execution time when the number of examples is limited.

magnitude of the current loss and the current gradient's norm are both large. MEGA-I directly balances the reference gradient and the current gradient, and the mixed stochastic gradient is dominated by the current gradient and it suffers from catastrophic forgetting. In contrast, MEGA-II balances the cosine similarity between gradients. Even if the norm of the current gradient is large, MEGA-II still allows adequate rotation of the direction of the current gradient to be closer to that of the reference gradient to alleviate catastrophic forgetting. We validate our claims by detailed analysis, which can be found in Section 8.7.3.

8.7.3 Detailed Analysis of MEGA-I and MEGA-II

In this section, we present a detailed analysis on the reason that why the MEGA-II outperforms MEGA-I significantly when the number of examples is limited. Define $k_1 = \frac{\ell_t}{\ell_{\text{ref}}}$, $k_2 = \frac{\|\nabla \ell_t(w;\xi)\|}{\|\nabla \ell_{\text{ref}}(w;\zeta)\|}$. We denote the angles between the mixed gradient $\mathbf{g}_{\text{mix}}$ and the current gradient $\nabla \ell_t(w;\xi)$ calculated by MEGA-I and MEGA-II by θ_1 and θ_2 respectively.

From the revious derivation, we know that

$$\cos\theta_2 = \frac{k_1 + \cos\tilde{\theta}}{\sqrt{k_1^2 + 2k_1\cos\tilde{\theta} + 1}}. \tag{8.16}$$

Now we derive the closed form of $\cos\theta_1$. For simplicity, we only consider the case where $\ell_t(w;\xi) \geq \epsilon$. By formula (8.11), we know that $\mathbf{g}_{\text{mix}} = \nabla\ell_t(w;\xi) + \frac{\ell_{\text{ref}}}{\ell_t}\nabla\ell_{\text{ref}}(w;\zeta)$. Define $\mathbf{g}_t = \ell_t(w;\xi)$, $\mathbf{g}_{\text{ref}} = \nabla\ell_{\text{ref}}(w;\zeta)$. By some algebra, we can show that

$$\mathbf{g}_{\text{mix}} = \frac{\ell_t}{\ell_{\text{ref}}}\|\mathbf{g}_{\text{ref}}\|\left(k_1k_2\frac{\mathbf{g}_t}{\|\mathbf{g}_t\|} + \frac{\mathbf{g}_{\text{ref}}}{\|\mathbf{g}_{\text{ref}}\|}\right).$$

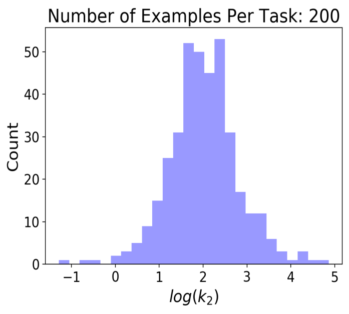
Hence, we have

$$\cos\theta_1 = \frac{\mathbf{g}_{\text{mix}}^\top\mathbf{g}_t}{\|\mathbf{g}_{\text{mix}}\|\|\mathbf{g}_t\|} = \frac{k_1k_2 + \cos\tilde{\theta}}{\sqrt{k_1^2k_2^2 + 2k_1k_2\cos\tilde{\theta} + 1}}. \tag{8.17}$$

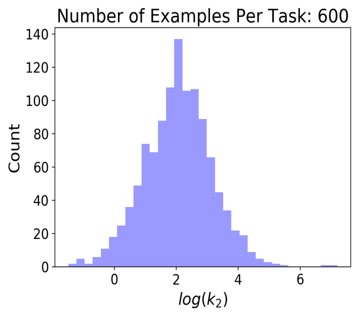
Comparing (8.17) and (8.16), and noting that the function $f(k) = \frac{k+\cos\tilde{\theta}}{\sqrt{k^2+2k\cos\tilde{\theta}+1}}$ is a monotonically increasing function with respect to k for $k \geq 0$, we know that if $k_1k_2 \geq k_1$, i.e., $k_2 \geq 1$, then $\cos\theta_1 \geq \cos\theta_2$, which means $\theta_1 \leq \theta_2$.

When the number of training examples is small, we empirically show that it is more common that $k_2 > 1$. This explains why MEGA-I's update direction is dominated by the current gradient's direction while MEGA-II still allows adequate rotation. This property helps MEGA-II obtain better performance than MEGA-I when the number of examples is small.

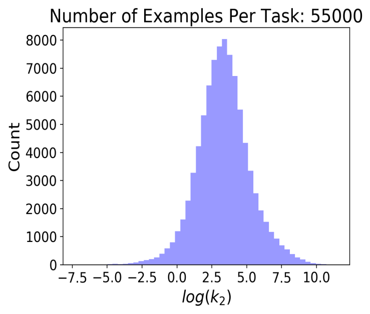
We construct 20 tasks with X number of examples per task, where X = 200, 600 and 55000. The way to generate the tasks is the same as in **Permuted MNIST**, that is,



(a) 200 Examples



(b) 600 Examples



(c) 55000 Examples

Figure 8.5: Count versus $\log(k_2)$, where $k_2 = \frac{\|\mathbf{g}_t\|}{\|\mathbf{g}_{\text{ref}}\|}$. $k_2 \geq 1$ holds for a larger proportion of all cases when the number of examples is smaller.

a fixed random permutation of input pixels is applied to all the examples for a particular task. During the learning process, we record the norm of the gradient on the current task and the norm of the gradient on the episodic memory in each mini-batch.

In Figure 8.5, we use histogram to show the distribution of $\log(k_2)$ of MEGA-I. As we can see, when the number of examples per task is smaller, k_2 tends to be greater than 1 for a larger proportion. In particular, when the number of examples per task is 55000, 3.61% of all k_2 are less than 1 and when the number of examples per task is 600, 3.15% of all k_2 are less than 1. Notably, when the number of examples per task is 200, only 1.05% of all k_2 are less than 1. As explained in the last paragraph, if $k_2 > 1$, then $\theta_1 \leq \theta_2$, which means MEGA-II allows a more significant rotation of the current gradient. So MEGA-II can offer better performance than MEGA-I, especially when the number of examples is small.

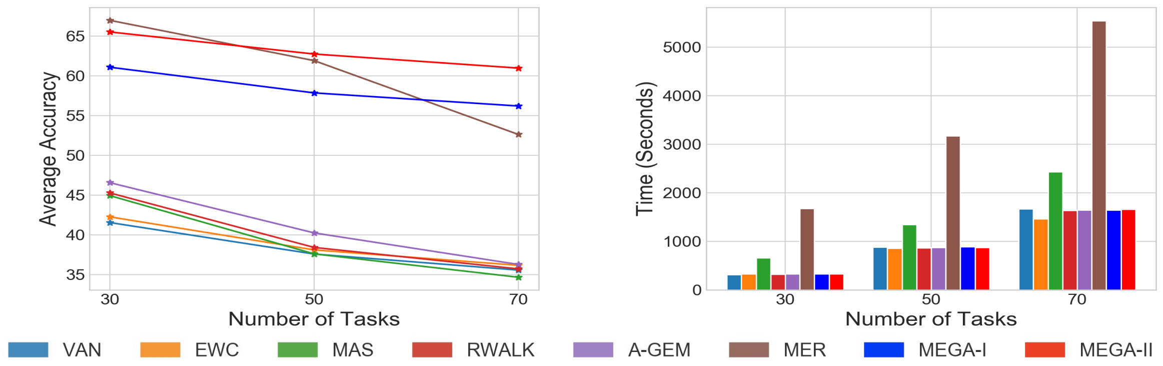


Figure 8.6: The average accuracy and execution time when the number of tasks is large.

8.7.4 MEGA-2 Outperforms Other Baseline and MEGA-1 When the Number of Tasks is Large

In this section, we increase the number of tasks 30, 50 and 70. Each task have 200 examples and is constructed in a similar way to the **Permuted MNIST**. In Fig. 8.6, we show the average accuracy and execution time for all the methods and all the cases. We can see that the proposed MEGA-II outperforms all the baselines, except in the cases of 30 tasks. From the execution time comparison in Fig. 8.6(b), we can see that MEGA-II is much more efficient than MER [RCA+18]. Note that MEGA-II also significantly outperforms MEGA-I in this case.

8.7.5 Ablation Studies

In this section, we include detailed ablation studies to analyze the reason why the proposed schemes can improve current episodic memory based lifelong learning methods. For MEGA-I, we consider the setting that both $\alpha_1(w) = 1$ and $\alpha_2(w) = 1$ during the training process. This ablation study is to show the effectiveness of the adaptive loss balancing scheme. For MEGA-II, we consider the setting that $\ell_t = \ell_{\text{ref}}$ in Eq. 8.12 to verify the effectiveness of the proposed gradient rotation scheme over A-GEM. The experimental settings are the same as Section 8.6.3. The results are shown in Table 8.3.

Table 8.3: Comparison of MEGA-I, MEGA-I ($\alpha_1(w) = 1, \alpha_2(w) = 1$), MEGA-II, MEGA-II ($\ell_t = \ell_{\text{ref}}$) and A-GEM.

Method	**Permuted MNIST**	**Split CIFAR**	**Split CUB**	**Split AWA**
	A_T (%)	A_T(%)	A_T(%)	A_T(%)
MEGA-I	91.10 ± 0.08	66.10 ± 1.67	79.67 ± 2.15	**54.82 ± 4.97**
MEGA-I ($\alpha_1(w) = 1, \alpha_2(w) = 1$)	90.66 ± 0.09	64.65 ± 1.98	79.44 ± 2.98	53.60 ± 5.21
MEGA-II	**91.21 ± 0.10**	**66.12 ± 1.93**	**80.58 ± 1.94**	54.28 ± 4.84
MEGA-II ($\ell_t = \ell_{\text{ref}}$)	91.15 ± 0.12	58.04 ± 1.89	68.60 ± 1.98	47.95 ± 4.54
A-GEM	89.32 ± 0.46	61.28 ± 1.88	61.82 ± 3.72	44.95 ± 2.97

In Table 8.3, we observe that MEGA-I achieves higher average accuracy than MEGA-I ($\alpha_1(w) = 1, \alpha_2(w) = 1$) by considering an adaptive loss balancing scheme. We also see that except on **Split CIFAR**, MEGA-II ($\ell_t = \ell_{\text{ref}}$) outperforms A-GEM on all the datasets. This demonstrates the benefits of the proposed approach for rotating the current gradient. By considering the loss information as in MEGA-II, we further improve the results on all the datasets. This shows that both of the two components (the rotation of the current gradient and loss balancing) contribute to the improvements of the proposed schemes.

8.8 Conclusion

In this chapter, we present a algorithm called MEGA for learning multiple domains with deep neural networks which achieves the state-of-the-art performance across all the benchmark datasets. In MEGA, we cast the lifelong learning problem as an optimization problem with composite objective and solve it with the proposed mixed stochastic gradient. Extensive experimental results show that the proposed MEGA achieves superior results across all the considered metrics and establishes the new state-of-the-art on all the datasets. The proposed method can learn multiple domains sequentially while maintaining the performance on old domains.

This chapter contains material from "Improved Schemes for Episodic Memory based Lifelong Learning Algorithm", by Yunhui Guo, Mingrui Liu, Tianbao Yang, and Tajana Rosing, which appears in The *34th Conference on Neural Information Processing Systems (NeurIPS)*, 2020. The dissertation author was the primary investigator and author of this paper.

Chapter 9

Summary and Future Work

9.1 Summary of the Dissertation

In this dissertation, we focus on learning across multiple domains using deep neural networks. We first consider learning across two image domains as in the standard transfer learning setting. Then we assume the target domain only has few examples per category, also called cross-domain few-shot learning. Next, we consider problems beyond two domains. One typical case is to learn multiple domains simultaneously to reduce the resource requirements for storing separate models for different domains or to boost the classification accuracy. Another one is to learn multiple domains sequentially while enabling the model to maintain performance on old domains. Learning across multiple domains has several benefits. First, the features extracted on the source domain can be transferred on the target domain to accelerate the learning process. Second, by exploiting the relation between different domains, we can train a single model that can be used for classifying images from different domains. Third, if the target domain only has few examples per category, by transferring the features from the source domain, we can still achieve high test accuracy on the target domain.

To address the issues of current methods for learning across multiple domains with deep neural networks, we propose new approaches in different scenarios. For learning across two images domains, we propose two adaptive fine-tuning methods called *SpotTune* and *AdaFilter* to automatically decide which residual block and convolutional filter should be fine-tuned and transferred. For cross-domain few-shot learning, we propose a new benchmark which covers images ranging from medical images to satellite images. We point out the issues of existing methods for few-shot learning and propose a new method called *incremental mult-model selection* which achieves the highest average accuracy on the benchmark. For learning across multiple domains, we first propose a multi-domain learning method called *SharingNet* which can classify images from multiple domains simultaneously. The proposed method leads to a small model size which allows the model to be deployed on memory-limited devices. Next, we target at a practical application in heterogeneous sensor networks. We propose a communication efficient distributed training framework for multiple domains based on active learning. Finally, for learning with multiple domains sequentially, we propose *Mixed Stochastic Gradient* which achieves the state-of-the-art performance across several datasets. The considered cases in this dissertation cover a wide range of real-world applications of learning with multiple domains and the proposed approaches advance the state-of-the-art in all considered scenarios.

In conclusion, in this dissertation we demonstrate the importance and effectiveness of learning across multi-domain with deep neural networks. We hope that our work will inspire researchers to investigate more in this direction.

9.2 Future Work

There are multiple directions for future work. One key aspect that is not considered in this dissertation is learning multiple domains with unlabelled data. It is thus natural to extend the cross-domain few-shot learning from standard supervised learning setting to unsupervised learning, semi-supervised learning and self-supervised learning. The idea is to explore the relationship between different domains with unlabelled data. Although it is hard to collect labels for some categories, to obtain unlabelled data is easier. We can utilize unlabelled data for few-shot learning classification to further improve the performance of the model. By allowing unlabelled data, we can develop more practical few-shot learning algorithms to address problems in real-world applications. Another key challenge is to fully utilize the resources of mobile devices. One direction is to apply model compression and quantization [Guo18] to multi-domain learning. One main purpose of multi-domain learning is to reduce the model size for mobile application deployment. By leveraging the idea of neural network quantization, we can further reduce the model size and enable faster training and inference speed. Different from the standard model quantization, in multi-domain learning during the quantization process we need to balance the performance of the model across different domains, which needs better ways to extract and understand the relationship between different domains. This poses challenges for current quantization methods and multi-domain learning methods.

Bibliography

[ABC$^+$16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[ABE$^+$18] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 139–154, 2018.

[AGO$^+$] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones.

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[AKT19] Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars. Task-free continual learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11254–11263, 2019.

[ALGB19] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. In *Advances in Neural Information Processing Systems*, pages 11816–11825, 2019.

[ARS$^+$16] Hossein Azizpour, Ali Sharif Razavian, Josephine Sullivan, Atsuto Maki, and Stefan Carlsson. Factors of transferability for a generic convnet representation. *IEEE transactions on pattern analysis and machine intelligence*, 38(9):1790–1802, 2016.

[AS18] Adewole S. Adamson and Avery Smith. Machine learning and health care disparities in dermatology. *JAMA Dermatology*, 154(11):1247–1248, November 2018.

[AVHN19] Naser AlDuaij, Alexander Van't Hof, and Jason Nieh. Heterogeneous multi-mobile computing. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, pages 494–507. ACM, 2019.

[BB96] Thomas D Burd and Robert W Brodersen. Processor design for portable systems. *Journal of VLSI signal processing systems for signal, image and video technology*, 13(2-3):203–221, 1996.

[BBPP15] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015.

[BCL10] Steven Bauer, David D Clark, and William Lehr. Understanding broadband speed measurements. Tprc, 2010.

[Ben12] Yoshua Bengio. Deep learning of representations for unsupervised and transfer learning. In *ICML Workshop on Unsupervised and Transfer Learning*, 2012.

[BFG$^+$16] Hakan Bilen, Basura Fernando, Efstratios Gavves, Andrea Vedaldi, and Stephen Gould. Dynamic image networks for action recognition. In *CVPR*, 2016.

[BGNK18] William H Beluch, Tim Genewein, Andreas Nürnberger, and Jan M Köhler. The power of ensembles for active learning in image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9368–9377, 2018.

[BHTV18] Luca Bertinetto, Joao F Henriques, Philip HS Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. *arXiv preprint arXiv:1805.08136*, 2018.

[BLC13] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[BT89] Dimitri P Bertsekas and John N Tsitsiklis. *Parallel and distributed computation: numerical methods*, volume 23. Prentice hall Englewood Cliffs, NJ, 1989.

[BV16] Hakan Bilen and Andrea Vedaldi. Integrated perception with recurrent multi-task neural networks. In *Advances in neural information processing systems*, pages 235–243, 2016.

[BV17] Hakan Bilen and Andrea Vedaldi. Universal representations: The missing link between faces, text, planktons, and cat breeds. *arXiv preprint arXiv:1701.07275*, 2017.

[BZK$^+$17] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. *arXiv preprint arXiv:1704.05796*, 2017.

[Car97] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[CBG13] S. Chopra, S. Balakrishnan, and R. Gopalan. Dlid: Deep learning for domain adaptation by interpolating between domains. In *ICML workshop on challenges in representation learning*, 2013.

[CBLR17] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. *arXiv preprint arXiv:1711.02257*, 2017.

[CDAT18] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 532–547, 2018.

[CDK$^+$06] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7(Mar):551–585, 2006.

[CG15] Joseph Cichon and Wen-Biao Gan. Branch-specific dendritic ca2+ spikes cause persistent synaptic plasticity. *Nature*, 520(7546):180–185, 2015.

[CHF$^+$15] Qiang Chen, Junshi Huang, Rogerio Feris, Lisa M Brown, Jian Dong, and Shuicheng Yan. Deep domain adaptation for describing people based on fine-grained clothing attributes. In *CVPR*, 2015.

[Cho17] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807. IEEE, 2017.

[CHS$^+$16] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[CLK$^+$19] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. In *International Conference on Learning Representations*, 2019.

[CMK+14] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3606–3613, 2014.

[CRRE18] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

[CRT+19] Noel Codella, Veronica Rotemberg, Philipp Tschandl, M Emre Celebi, Stephen Dusza, David Gutman, Brian Helba, Aadi Kalloo, Konstantinos Liopyris, Michael Marchetti, et al. Skin lesion analysis toward melanoma detection 2018: A challenge hosted by the international skin imaging collaboration (isic). *arXiv preprint arXiv:1902.03368*, 2019.

[Csu17] G. Csurka. Domain adaptation for visual applications: A comprehensive survey. In *Domain Adaptation in Computer Vision Applications*. Springer, 2017.

[CTS14] Jianshu Chen, Zaid J Towfic, and Ali H Sayed. Dictionary learning over distributed models. *IEEE Transactions on Signal Processing*, 63(4):1001–1016, 2014.

[Das11] Sanjoy Dasgupta. Two faces of active learning. *Theoretical computer science*, 412(19):1767–1781, 2011.

[DCM+12] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[DJV+14] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, 2014.

[DLAM+19] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. Continual learning: A comparative study on how to defy forgetting in classification tasks. *arXiv preprint arXiv:1909.08383*, 2019.

[DPS06] Miroslav Dudík, Steven J Phillips, and Robert E Schapire. Correcting sample selection bias in maximum entropy density estimation. In *NeurIPS*, 2006.

[DSM19] Nikita Dvornik, Cordelia Schmid, and Julien Mairal. Diversity with cooperation: Ensemble methods for few-shot classification. *arXiv preprint arXiv:1903.11341*, 2019.

[DTXM09] Lixin Duan, Ivor W Tsang, Dong Xu, and Stephen J Maybank. Domain transfer SVM for video concept detection. In *CVPR*, 2009.

[DZ17] Carl Doersch and Andrew Zisserman. Multi-task self-supervised visual learning. In *The IEEE International Conference on Computer Vision (ICCV)*, 2017.

[EEDR19] Sayna Ebrahimi, Mohamed Elhoseiny, Trevor Darrell, and Marcus Rohrbach. Uncertainty-guided continual learning with bayesian neural networks. *arXiv preprint arXiv:1906.02425*, 2019.

[EHA12] Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? *ACM Transactions on Graphics*, 31(4):44–1, 2012.

[FAL17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[FAML19] Mehrdad Farajtabar, Navid Azizan, Alex Mott, and Ang Li. Orthogonal gradient descent for continual learning. *arXiv preprint arXiv:1910.07104*, 2019.

[FCZ$^+$17] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry P Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *CVPR*, 2017.

[FG18] Sebastian Farquhar and Yarin Gal. Towards robust evaluations of continual learning. *arXiv preprint arXiv:1805.09733*, 2018.

[FG19] Sebastian Farquhar and Yarin Gal. A unifying bayesian view of continual learning. *arXiv preprint arXiv:1902.06494*, 2019.

[Fre99] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[GBCB16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

[GBT$^+$04] Carlos Guestrin, Peter Bodik, Romain Thibaux, Mark Paskin, and Samuel Madden. Distributed regression: an efficient framework for modeling sensor network data. In *Proceedings of the 3rd international symposium on Information processing in sensor networks*, pages 1–10. ACM, 2004.

[GCK$^+$19] Yunhui Guo, Noel CF Codella, Leonid Karlinsky, John R Smith, Tajana Rosing, and Rogerio Feris. A new benchmark for evaluation of cross-domain few-shot learning. *arXiv preprint arXiv:1912.07200*, 2019.

[GDDM14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[GHP07] Gregory Griffin, Alex Holub, and Pietro Perona. Caltech-256 object category dataset. 2007.

[GMH13] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[GSK$^+$19] Yunhui Guo, Honghui Shi, Abhishek Kumar, Kristen Grauman, Tajana Rosing, and Rogerio Feris. Spottune: transfer learning through adaptive fine-tuning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4805–4814, 2019.

[GUA$^+$16] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

[Guo18] Yunhui Guo. A survey on methods and theories of quantized neural networks. *arXiv preprint arXiv:1808.04752*, 2018.

[GY17] Weifeng Ge and Yizhou Yu. Borrowing treasures from the wealthy: Deep transfer learning through selective joint fine-tuning. In *CVPR*, 2017.

[HBDB19] Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(7):2217–2226, 2019.

[HDY$^+$12] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[HGDG17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[HJ18] Xu He and Herbert Jaeger. Overcoming catastrophic interference using conceptor-aided backpropagation. 2018.

[HKCK18] Tyler L Hayes, Ronald Kemker, Nathan D Cahill, and Christopher Kanan. New metrics and experimental paradigms for continual learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2031–2034, 2018.

[HLvdMW17] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

[HRS$^+$17] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[HTYN$^+$15] Akiko Hayashi-Takagi, Sho Yagishita, Mayumi Nakamura, Fukutoshi Shirai, Yi I Wu, Amanda L Loshbaugh, Brian Kuhlman, Klaus M Hahn, and Haruo Kasai. Labelling and optical erasure of synaptic memory traces in the motor cortex. *Nature*, 525(7569):333, 2015.

[HZC$^+$17] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[ina] INA219 High Side DC Current Sensor Breakout. https://www.adafruit.com/product/904. [Online].

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456, 2015.

[JC04] Ankur Jain and Edward Y Chang. Adaptive sampling for sensor networks. In *Proceeedings of the 1st international workshop on Data management for sensor networks: in conjunction with VLDB 2004*, pages 10–16. ACM, 2004.

[JGP16] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[JST$^+$14] Martin Jaggi, Virginia Smith, Martin Takác, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I Jordan. Communication-efficient distributed dual coordinate ascent. In *Advances in neural information processing systems*, pages 3068–3076, 2014.

[JVB09] Laurent Jacob, Jean-philippe Vert, and Francis R Bach. Clustered multi-task learning: A convex formulation. In *NeurIPS*, 2009.

[K$^+$09] Alex Krizhevsky et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[KDI12] Abhishek Kumar and Hal Daumé III. Learning task grouping and overlap in multi-task learning. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, pages 1723–1730, 2012.

[KGC17] Lukasz Kaiser, Aidan N Gomez, and Francois Chollet. Depthwise separable convolutions for neural machine translation. *arXiv preprint arXiv:1706.03059*, 2017.

[KGC18] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7482–7491, 2018.

[KGS11] Zhuoliang Kang, Kristen Grauman, and Fei Sha. Learning with whom to share in multi-task feature learning. In *ICML*, 2011.

[KGUD07] Ashish Kapoor, Kristen Grauman, Raquel Urtasun, and Trevor Darrell. Active learning with gaussian processes for object categorization. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE, 2007.

[KJYFF11] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. Novel dataset for fine-grained image categorization. In *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO, June 2011.

[KMA$^+$18] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler L Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[KMY$^+$16] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[KOC$^+$] Newton M. Kinyanjui, Timothy Odonga, Celia Cintas, Noel C. F. Codella, Rameswar Panda, Prasanna Sattigeri, and Kush R. Varshney. Estimating skin tone and effects on classification performance in dermatology datasets. *NeurIPS Fair ML for Health Workshop 2019*.

[Kok17] Iasonas Kokkinos. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5454–5463. IEEE, 2017.

[KPR$^+$17] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[KSDFF13] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *ICCV Workshop on Workshop on 3D Representation and Recognition*, pages 554–561, 2013.

[KSF17] Ksenia Konyushkova, Raphael Sznitman, and Pascal Fua. Learning active learning from data. In *Advances in Neural Information Processing Systems*, pages 4225–4235, 2017.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[KSL18] Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better imagenet models transfer better? *arXiv preprint arXiv:1805.08974*, 2018.

[KSW$^+$18] A. Kumar, P. Sattigeri, K. Wadhawan, L. Karlinsky, R. S. Feris, W. T. Freeman, and G. Wornell. Co-regularized alignment for unsupervised domain adaptation. In *NeurIPS*, 2018.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LCB98] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database. *URL http://yann. lecun. com/exdb/mnist*, 1998.

[LCWJ15] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I Jordan. Learning transferable features with deep adaptation networks. In *ICML*, 2015.

[LD18] Lanlan Liu and Jia Deng. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. In *AAAI*, 2018.

[LEN08] Honglak Lee, Chaitanya Ekanadham, and Andrew Y Ng. Sparse deep belief net model for visual area v2. In *Advances in neural information processing systems*, pages 873–880, 2008.

[LFP06] Fei-Fei Li, Rob Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence*, 28(4):594–611, 2006.

[LG13] Xin Li and Yuhong Guo. Adaptive active learning for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 859–866, 2013.

[LGD18] Xuhong Li, Yves Grandvalet, and Franck Davoine. Explicit inductive bias for transfer learning with convolutional networks. In *ICML*, 2018.

[LH17] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[LHM$^+$17] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

[LKJ$^+$17] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *Advances in neural information processing systems*, pages 4652–4662, 2017.

[LKZ$^+$17] Yongxi Lu, Abhishek Kumar, Shuangfei Zhai, Yu Cheng, Tara Javidi, and Rogério Schmidt Feris. Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. In *CVPR*, 2017.

[LLSL19] Kibok Lee, Kimin Lee, Jinwoo Shin, and Honglak Lee. Overcoming catastrophic forgetting with unlabeled data in the wild. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 312–321, 2019.

[LMRS19] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10657–10665, 2019.

[LNH09] Christoph H Lampert, Hannes Nickisch, and Stefan Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 951–958. IEEE, 2009.

[LP$^+$17] David Lopez-Paz et al. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pages 6467–6476, 2017.

[LSGT11] Brenden Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua Tenenbaum. One shot learning of simple visual concepts. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 33, 2011.

[LST15] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[Mad03] Samuel Ross Madden. *The design and evaluation of a query processing architecture for sensor networks.* PhD thesis, University of California, Berkeley, 2003.

[MHS16] Sharada P Mohanty, David P Hughes, and Marcel Salathé. Using deep learning for image-based plant disease detection. *Frontiers in plant science*, 7:1419, 2016.

[ML18] Arun Mallya and Svetlana Lazebnik. Piggyback: Adding multiple tasks to a single, fixed network by learning to mask. *arXiv preprint arXiv:1801.06519*, 2018.

[MM18] Elliot Meyerson and Risto Miikkulainen. Beyond shared hierarchies: Deep multitask learning through soft layer ordering. In *ICLR*, 2018.

[MMR$^+$16] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

[MMT16] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[MRK$^+$13] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013.

[MSGH16] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *CVPR*, 2016.

[MVPC13] Thomas Mensink, Jakob Verbeek, Florent Perronnin, and Gabriela Csurka. Distance-based image classification: Generalizing to new classes at near-zero cost. *IEEE transactions on pattern analysis and machine intelligence*, 35(11):2624–2637, 2013.

[NAS18] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[NZ08] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics & Image Processing*, 2008.

[PKP06] Joel B Predd, Sanjeev B Kulkarni, and H Vincent Poor. Distributed learning in wireless sensor networks. *IEEE Signal Processing Magazine*, 23(4):56–69, 2006.

[PKP$^+$19] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 2019.

[PRWDI12] Alexandre Passos, Piyush Rai, Jacques Wainer, and Hal Daume III. Flexible modeling of latent task structures in multitask learning. *arXiv preprint arXiv:1206.6486*, 2012.

[PSBD19] Jihong Park, Sumudu Samarakoon, Mehdi Bennis, and Mérouane Debbah. Wireless network intelligence at the edge. *Proceedings of the IEEE*, 107(11):2204–2239, 2019.

[PY09] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[PY$^+$10] Sinno Jialin Pan, Qiang Yang, et al. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[QT09] Ariadna Quattoni and Antonio Torralba. Recognizing indoor scenes. In *CVPR*, pages 413–420. IEEE, 2009.

[Rat90] Roger Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285, 1990.

[RBB18] Hippolyt Ritter, Aleksandar Botev, and David Barber. Online structured laplace approximations for overcoming catastrophic forgetting. In *Advances in Neural Information Processing Systems*, pages 3738–3748, 2018.

[RBV17] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *NIPS*, 2017.

[RBV18a] S.-A. Rebuffi, H. Bilen, and A. Vedaldi. Efficient parametrization of multi-domain deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[RBV18b] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Efficient parametrization of multi-domain deep neural networks. In *CVPR*, 2018.

[RC10] Parisa Rashidi and Diane J Cook. Mining sensor streams for discovering human activity patterns over time. In *2010 IEEE International Conference on Data Mining*, pages 431–440. IEEE, 2010.

[RCA$^+$18] Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. *arXiv preprint arXiv:1810.11910*, 2018.

[RHDC19] Veronica Rotemberg, Allan Halpern, Stephen W. Dusza, and Noel C. F. Codella. The role of public challenges and data sets towards algorithm development, trust, and use in clinical practice. *Seminars in Cutaneous Medicine and Surgery*, 38(1):E38–E42, March 2019.

[RL16] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

[RMHK15] Shah Atiqur Rahman, Christopher Merck, Yuxiao Huang, and Samantha Kleinberg. Unintrusive eating recognition using google glass. In *2015 9th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth)*, pages 108–111. IEEE, 2015.

[RN04] Michael Rabbat and Robert Nowak. Distributed optimization in sensor networks. In *Proceedings of the 3rd international symposium on Information processing in sensor networks*, pages 20–27. ACM, 2004.

[Rob95] Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146, 1995.

[RRD$^+$16] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[RS12] Attila Reiss and Didier Stricker. Introducing a new benchmarked dataset for activity monitoring. In *Wearable Computers (ISWC), 2012 16th International Symposium on*, pages 108–109. IEEE, 2012.

[RT17] Amir Rosenfeld and John K Tsotsos. Incremental learning through deep adaptation. *arXiv preprint arXiv:1705.04228*, 2017.

[RTR$^+$18] Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B Tenenbaum, Hugo Larochelle, and Richard S Zemel. Meta-learning for semi-supervised few-shot classification. *arXiv preprint arXiv:1803.00676*, 2018.

[SE15] Babak Saleh and Ahmed Elgammal. Large-scale classification of fine-art paintings: Learning the right metric on the right feature. *arXiv preprint arXiv:1505.00855*, 2015.

[Set09] Burr Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

[SHZ$^+$18] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *arXiv preprint arXiv:1801.04381*, 2018.

[SK18] Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. In *Advances in Neural Information Processing Systems*, pages 527–538, 2018.

[SLKK17] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pages 2990–2999, 2017.

[SMM$^+$17] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *ICLR*, 2017.

[SNR06] Aarti Singh, Robert Nowak, and Parmesh Ramanathan. Active learning for adaptive mobile sensing networks. In *Proceedings of the 5th international conference on Information processing in sensor networks*, pages 60–68. ACM, 2006.

[SRASC14] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *CVPR DeepVision Workshop*, 2014.

[SSZ14] Ohad Shamir, Nati Srebro, and Tong Zhang. Communication-efficient distributed optimization using an approximate newton-type method. In *International conference on machine learning*, pages 1000–1008, 2014.

[SSZ17] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, pages 4077–4087, 2017.

[SYZ$^+$18] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1199–1208, 2018.

[TBZH19] Nguyen H Tran, Wei Bao, Albert Zomaya, and Choong Seon Hong. Federated learning over wireless networks: Optimization model design and analysis. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 1387–1395. IEEE, 2019.

[TGK+19] Anthony Thomas, Yunhui Guo, Yeseong Kim, Baris Aksanli, Arun Kumar, and Tajana S Rosing. Hierarchical and distributed machine learning inference beyond the edge. In *2019 IEEE 16th International Conference on Networking, Sensing and Control (ICNSC)*, pages 18–23. IEEE, 2019.

[TK01] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal of machine learning research*, 2(Nov):45–66, 2001.

[TM95] Sebastian Thrun and Tom M Mitchell. Lifelong robot learning. *Robotics and autonomous systems*, 15(1-2):25–46, 1995.

[TMK17] Surat Teerapittayanon, Bradley McDanel, and HT Kung. Distributed deep neural networks over the cloud, the edge and end devices. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 328–339. IEEE, 2017.

[TO98] Sebastian Thrun and Joseph O'Sullivan. Clustering learning tasks and the selective cross-task transfer of knowledge. In *Learning to learn*. Springer, 1998.

[TRK18] Philipp Tschandl, Cliff Rosendahl, and Harald Kittler. The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Scientific data*, 5:180161, 2018.

[TSG+16] Nima Tajbakhsh, Jae Y Shin, Suryakanth R Gurudu, R Todd Hurst, Christopher B Kendall, Michael B Gotway, and Jianming Liang. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE transactions on medical imaging*, 35(5):1299–1312, 2016.

[TZD+19] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, and Hugo Larochelle. Meta-dataset: A dataset of datasets for learning to learn from few examples. *arXiv preprint arXiv:1903.03096*, 2019.

[VB17] Andreas Veit and Serge Belongie. Convolutional networks with adaptive computation graphs. *arXiv preprint arXiv:1711.11503*, 2017.

[VB18] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *ECCV*, 2018.

[VBL$^+$16] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

[VGSR18] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564*, 2018.

[VWB16] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *NeurIPS*, 2016.

[WBM$^+$10] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

[WBW$^+$11] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[WCH$^+$19] Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, and Lisha Hu. Deep learning for sensor-based activity recognition: A survey. *Pattern Recognition Letters*, 119:3–11, 2019.

[WHG17] Xiaolong Wang, Kaiming He, and Abhinav Gupta. Transitive invariance for selfsupervised visual representation learning. In *Proc. of Int'l Conf. on Computer Vision (ICCV)*, 2017.

[WMN04] Rebecca Willett, Aline Martin, and Robert Nowak. Backcasting: adaptive sampling for sensor networks. In *Proceedings of the 3rd international symposium on Information processing in sensor networks*, pages 124–133. ACM, 2004.

[WNC06] Rebecca Willett, Robert Nowak, and Rui M Castro. Faster rates in regression via active learning. In *Advances in Neural Information Processing Systems*, pages 179–186, 2006.

[WNK$^+$18] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *CVPR*, 2018.

[won] Wondershaper. https://github.com/magnific0/wondershaper. [Online].

[WPL$^+$17] Xiaosong Wang, Yifan Peng, Le Lu, Zhiyong Lu, Mohammadhadi Bagheri, and Ronald M Summers. Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2097–2106, 2017.

[WQXY07] W.Dai, Q.Yang, G. Xue, and Y.Yu. Boosting for transfer learning. In *ICML*, 2007.

[WTS$^+$18] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. When edge meets learning: Adaptive control for resource-constrained distributed machine learning. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 63–71. IEEE, 2018.

[YCBL14] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *NeurIPS*, 2014.

[YDH$^+$18] Zhilin Yang, Bhuwan Dhingra, Kaiming He, William W Cohen, Ruslan Salakhutdinov, Yann LeCun, et al. Glomo: Unsupervisedly learned relational graphs as transferable representations. *arXiv preprint arXiv:1806.05662*, 2018.

[YHPC17] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *arXiv preprint arXiv:1708.02709*, 2017.

[YYS18] Bicheng Ying, Kun Yuan, and Ali H Sayed. Supervised learning under distributed features. *IEEE Transactions on Signal Processing*, 67(4):977–992, 2018.

[ZBS19] Chiyuan Zhang, Samy Bengio, and Yoram Singer. Are all layers created equal? *arXiv preprint arXiv:1902.01996*, 2019.

[ZCCY19] Guanxiong Zeng, Yang Chen, Bo Cui, and Shan Yu. Continual learning of context-dependent processing in neural networks. *Nature Machine Intelligence*, 1(8):364–372, 2019.

[ZCF$^+$] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets.

[ZCY11] Jiayu Zhou, Jianhui Chen, and Jieping Ye. Clustered multi-task learning via alternating structure optimization. In *NeurIPS*, 2011.

[ZF14] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[ZPG17] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3987–3995. JMLR. org, 2017.

[ZSS$^+$18] Amir R Zamir, Alexander Sax, William Shen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3712–3722, 2018.

[ZWD12] Yuchen Zhang, Martin J Wainwright, and John C Duchi. Communication-efficient algorithms for statistical optimization. In *Advances in Neural Information Processing Systems*, pages 1502–1510, 2012.