UNIVERSITY OF CALIFORNIA SAN DIEGO

Dynamic Neural Networks for Resource Constrained Image Recognition

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

in

Electrical Engineering (Machine Learning & Data Science)

by

Yunsheng Li

Committee in charge:

Professor Nuno Vasconcelos, Chair
Professor Nikolay Atanasov
Professor David Kriegman
Professor Truong Nguyen
Professor Ravi Ramamoorthi

2021

The dissertation of Yunsheng Li is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2021

DEDICATION

Dedicated to my parents

EPIGRAPH

We may hope that machines will eventually compete with men in all purely intellectual fields.

—Alan Turing

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my sincerest gratitude to my family, advisor, committee members, mentors, collaborators and colleagues. Without their help, it would be impossible for me to complete the Ph.D. degree.

First, I would like to thank my Ph.D. advisor, Professor Nuno Vasconcelos. It was him to bring me into this interesting research field of computer vision. He gave me very detailed advice on research projects especially in the early years when I worked with him. His critical thinking, research philosophy and technical writing highly advanced my research ability as well as changing the way I think about the world. Besides, I would also want to thank Professor Nikolay Atanasov, Professor Truong Nguyen, Professor David Kriegman and Professor Ravi Ramamoorthi for being my committee members and the advice they gave to me when I pursued the Ph.D. degreee.

I worked as a research intern for multiple times in Microsoft, where I obtained unique and terrific research experiences. Particularly, I would like to thank Lu Yuan, the principle research manager from Microsoft, for offering my first intern job and giving me comprehensive help. The new perceptive of scientific research he showed has a profound influence on my research life. I would like to thank Yinpeng Chen, who worked as a mentor with me during my internship, for promoting my ability to think and training my presentation skills. I would also like to thank other co-workers, Xiyang Dai, Dongdong Chen, Mengchen Liu, Ying Jing, Pei Yu, Zicheng Liu, Ye Yue, who brainstormed with me.

I also want to thank all my colleagues from SVCL, Weixin Li, Mandar Dixit, Zhaowei Cai, Bo Liu, Pedro Morgado, Yingwei Li, Xiangyun Zhao, Xudong Wang, Pei Wang, Yi Li, Gautam Nain, Tz-Ying Wu, Chih-Hui Ho, Jiacheng Cheng, Brandon Leung, Yiran Xu, for both the insightful discussion that inspired me and enriching my life beyond research. Among them, I want to particularly thank Weixin Li and Mandar Dixit who helped me a lot when I started to work in SVCL and my officemate Bo Liu who worked closely with me throughout my Ph.D.

career. In addition, I would like to thank my co-authors, Pedro Morgado, Pei Wang, Tz-Ying Wu and Yiran Xu, of several papers I published in SVCL.

Finally, I would like to thank my families. I am very grateful to my parents who support me to go abroad to pursue my Ph.D. degree and encouraged me when I encountered difficulties. Besides, I want to thank my grandparents, who took care of me when I was a child. I would also like to thank my aunt. She helped me a lot when I made major decisions in my life.

Chapter 2 is, in full, based on the material as it appears in the publication of "Efficient Multi-Domain Learning by Covariance Normalization", Yunsheng Li and Nuno Vasconcelos, in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 2019. The dissertation author was the primary investigator and author of this material.

Chapter 3 is, in full, based on the material as it appears in the publication of "Revisiting Dynamic Convolution via Matrix Decomposition", Yunsheng Li, Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Ye Yue, Lu Yuan, Zicheng Liu, Mei Chen and Nuno Vasconcelos, in *The Ninth International Conference on Learning Representations* (ICLR), 2021. The dissertation author was the primary investigator and author of this material.

Chapter 4 is, in full, based on the material as it appears in the submission of "MicroNet: Improving Image Recognition with Extremely Low FLOPs", Yunsheng Li, Yinpeng Chen, Xiyang Dai, Dongdong Chen, Mengchen Liu, Lu Yuan, Zicheng Liu, Lei Zhang and Nuno Vasconcelos, in *Proceedings of IEEE International Conference on Computer Vision* (ICCV), 2021. The dissertation author was the primary investigator and author of this material.

Chapter 5 is, in full, based on the material as it appears in the publication of "Bidirectional Learning for Domain Adaptation of Semantic Segmentation", Yunsheng Li, Lu Yuan and Nuno Vasconcelos, in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 2019. The dissertation author was the primary investigator and author of this material.

Chapter 6 is, in full, based on the material as it appears in the publication of "Dynamic Transfer for Multi-Source Domain Adaptation", Yunsheng Li, Lu Yuan, Yinpeng Chen, Pei Wang

and Nuno Vasconcelos, in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 2021. The dissertation author was the primary investigator and author of this material.

VITA

B. S. in Electrical and Electronics Engineering, Harbin Engineering University, China

M. S. in Electrical Engineering (Communication Theory & Systems), UC San Diego

Ph. D. in Electrical Engineering (Machine Learning & Data Science), UC San Diego

PUBLICATIONS

Yunsheng Li, Yinpeng Chen, Xiyang Dai, Dongdong Chen, Mengchen Liu, Lu Yuan, Zicheng Liu, Lei Zhang, Nuno, Vasconcelos. "MicroNet: Towards Image Recognition with Extremely Low FLOPs", in *Proceedings of IEEE International Conference on Computer Vision* (ICCV), 2021.

Yunsheng Li, Lu Yuan, Yinpeng Chen, Pei Wang, Nuno Vasconcelos. "Dynamic Transfer for Multi-Source Domain Adaptation", in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 2021

Yunsheng Li, Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Ye Yu, Lu Yuan, Zicheng Liu, Mei Chen, Nuno Vasconcelos. "Revisiting Dynamic Convolution via Matrix Decomposition", in *The Ninth International Conference on Learning Representations* (ICLR), 2021

Pedro Morgado, **Yunsheng Li** and Nuno Vasconcelos. "Deep Hashing with Hash-Consistent large Margin Codewords", in *International Journal of Computer Vision* (IJCV), 2021

Yiran Xu, Xiaoyin Yang, Lihang Gong, Hsuan-Chu Lin, Tz-Ying Wu, **Yunsheng Li** and Nuno Vasconcelos. "Explainable Object-induced Action Decision for Autonomous Vehicles", in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 2020

Yunsheng Li, Lu Yuan and Nuno Vasconcelos. "Bidirectional Learning for Domain Adaptation of Semantic Segmentation", in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 2019

Yunsheng Li and Nuno Vasconcelos. "Efficient Multi-Domain Learning by Covariance Normalization", in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 2019

Mandar Dixit, **Yunsheng Li** and Nuno Vasconcelos. "Semantic Fisher Scores for Task Transfer: Using Objects to Classify Scenes", in *IEEE Transactions on Pattern Analysis and Machine Intelligence* (TPAMI), 2019

Yunsheng Li, Mandar Dixit, and Nuno Vasconcelos. "Deep Scene Image Classification with the MFAFVNet", in *Proceedings of IEEE International Conference on Computer Vision* (ICCV), 2017.

ABSTRACT OF THE DISSERTATION

Dynamic Neural Networks for Resource Constrained Image Recognition

by

Yunsheng Li

Doctor of Philosophy in Electrical Engineering (Machine Learning & Data Science)

University of California San Diego, 2021

Professor Nuno Vasconcelos, Chair

Deep learning has brought remarkable improvement for the performance of image recognition tasks. However, the resource limitation forms a big obstacle for the real application of deep learning. Two types of resource constraints- limited machine's computing power and lack of annotated data are considered in the thesis. Compared to leveraging a large static network formed by input independent blocks, we try to overcome the issues of resource limitation with a more effective architecture by proposing a series of dynamic neural networks with input dependent blocks.

For the tasks with constrained computational resources, we first consider the multi-domain learning problem, which requires a single framework to perform well on multiple datasets for

image classification. CovNorm is proposed to dynamically project a common feature to different feature spaces according to the dataset ID by consuming tiny numbers of extra parameters and computation. Then a large scale image recognition problem under different computational resources is explored. Dynamic Convolution Decomposition (DCD) is proposed for the machines with computing power from the order of 100 MFLOPs to 10 GFLOPs while MicroNet is designed to be applied on the machines with the computational cost far below 100 MFLOPs. Empowered by the dynamic architecture, both DCD and MicroNet achieve a significant improvement within their working scope.

For the issue of lacking annotated data, we work on the domain adaptation tasks, where the dataset is partially labeled and a domain gap exists between the labeled data (source domain) and the unlabeled data (target domain). We start by considering a relatively simple case with a single source and single target domain on semantic segmentation. A bidirectional learning (BDL) framework is designed and it reveals the synergy of several key factors, i.e., adversarial learning and self-training for domain adaptation. Based on the techniques given by BDL and the power of dynamic networks, a more complex problem- multi-source domain adaptation is investigated. Dynamic residual transfer (DRT) is presented and shows tremendous improvement for the adaptation performance compared to its static version. It confirms the effectiveness of dynamic networks for the image recognition problem when the amount of annotated data is limited.

Chapter 1

Introduction

1.1 Resource Constrained Image Recognition

Image recognition tasks, e.g., image classification, keypoint detection, object detection and segmentation segmentation have been widely studied and significantly boosted since the pioneer deep learning framework [4]. And the deep learning model is still being designed and trained to be applied on more complex tasks with the target of better performance. As a result, as long as the model is large enough, the task complexity can be scaled from recognizing thousands of images to billions of images or from one pure classification task to multi-task learning. Although the progress of deep learning is quite amazing, its success highly relies on two types of resources, i.e, the powerful machines and large amount of training (annotated) data. Hence, the generality of deep learning is impaired in the scenario where the resources are constrained. In the thesis, we attempt to deal with this issue by working on resource constrained vision tasks and try to maximize the performance of neural networks under a strict computational restriction or insufficient annotated data.

For the constraint of computational resources, it is a widely existing issue for vision problems. For instance, most of the mobile devices only contain CPUs or very weak GPUs that can only handle the models with millions of FLOPs. For a GPU cluster, although its computing power is significantly stronger than mobile devices, which allows it to run models that cost tens of millions of FLOPs or even billions of FLOPs, it still has a ceiling for the computing power, depending on the number of GPUs it contains. Thus how to maximize the performance given limited computational resources has become a popular research subject and people have developed diverse novel techniques from the view of network compression [5, 6] to neural architecture search [7, 8].

For the shortage of annotated data, there exist different research subjects, e.g., self-supervised learning, semi-supervised learning and domain adaptation, based on the data styles and the ratio of annotated data. In the thesis, we mainly focus on the domain adaptation, where the

dataset is partially labeled and the labeled data (served as source domain) has a clear discrepancy to the unlabeled data (served as target domain) that people care more about. The hypothesis of domain adaptation is annotating data is time consuming or has very high requirement on the labelers for some specific tasks, e.g., semantic segmentation or fine-grained classification. Besides, even though enough data is labeled for these tasks, the style of the data will change according to the time or the places it is collected, which makes the labeling work intractable. The aim of domain adaptation is to fully leverage the existing labeled data and the large scale of unlabeled data to train a model that can perform equally well on the unlabeled data as the labeled data. With this clear target, people have developed a series of effective techniques, e.g., adversarial learning [9, 10], self-training [11] and data mixup [12, 13].

1.2 Limitations of Existing Methods

The two resource restriction issues have been widely explored and eased by various methods. For the constraint of computational resources, as mentioned before, network compression and neural architecture search were proposed to overcome this issue from different aspects. The network compression mainly tries to reduce the computational cost via discarding some unimportant parts of the network via either matrix decomposition [14] or channel pruning [15] for the convolution layers. The neural architecture search (NAS) aims to seek the best aggregation of network blocks with different width and depth under certain computational budget and reinforcement learning [7] is a quite popular method used for NAS. For domain adaptation, people care more about the metric design, e.g., maximum mean discrepancy (MMD) [16] or adversarial learning [9], to force the network to project data from different domains to the shared feature space. Therefore, so long as the features from the source domain are discriminative, which is assured through supervision given by labels, the target features will also be discriminative.

For the techniques used to solve both issues, although most of them seem to be irrelevant,

they share to use a *static* model, where the model is data independent. The static model forms two types of weakness that are severely harmful for the image recognition tasks under limited resources. First, the static model is less efficient. For image recognition tasks, a network is required to map all data to the feature space that can maximize the inter-class distance while minimizing the intra-class distance. Hence, so as to achieve this goal, the network has to be very complex to form a sophisticated function and sometimes the complexity is beyond the computational ability of some devices. Second, the adaptiveness of static model is limited. In another word, the static model is not invariant to data styles. For instance, if the model is trained to classify a bunch of real images, its performance will drop in a cliff manner when it is deployed to cartoon images, even though all classses are shared. This weak adaptive capacity prevents the performance of domain adaptation to be further promoted, especially when the data has a large variation (large domain discrepancy).

1.3 Contributions of the Thesis

In the thesis, we present a series of *dynamic* networks to overcome the weakness brought by the aforementioned static models for various image recognition tasks under different types of resource constraints. The dynamic networks refer to the family of neural networks that contains input dependent parameters or architectures. Contrary to the static counterpart, the key advantages of dynamic networks lie on two aspects- *efficiency* and *adaptiveness*. In the following parts of the thesis, we show that these two advantages make the dynamic network a perfect tool for different resource constrained image recognition problems. Specifically, we first consider the problem of computational resource restriction for multi-domain learning and large scale image recognition tasks. We present a domain-wise dynamic framework and two sample-wise dynamic frameworks that are explained in the first three works respectively. All the dynamic frameworks show excellent computational efficiency under different circumstances. The rest two works are

mainly designed for the domain adaptation problem. They first reveal several critical techniques for domain adaptation and based on these techniques several dynamic modules are investigated. No matter simple or complex, all the dynamic modules show tremendous improvement for the adaptation performance and confirm the adaptiveness for dynamic models.

1.3.1 A Domain-Wise Dynamic Framework for Efficient Multi-Domain Learning

The problem of multi-domain learning of deep networks is considered. The multi-domain learning asks for a generic framework to do classification job on different datasets (domains). In order to avoid building a model per dataset, a series of adaptive layers are dynamically induced according to each target dataset and a novel procedure, denoted covariance normalization (CovNorm), is proposed to reduce its parameters. CovNorm is applied per convolutional layer to convert the domain-common feature to the domain-specific feature. It is a data driven method of fairly simple implementation, requiring two principal component analyzes (PCA) and fine-tuning of a mini-adaptation layer. Nevertheless, it is shown, both theoretically and experimentally, to have several advantages over previous approaches, such as batch normalization or geometric matrix approximations. Furthermore, CovNorm can be deployed both when target datasets are available sequentially or simultaneously. Experiments show that, in both cases, it has performance comparable to a fully fine-tuned network, costing as very few parameters and computation per target domain.

1.3.2 A Sample-Wise Dynamic Framework for Efficient Large-Scale Image Recognition

The domain-wise dynamic framework- CovNorm has been shown to be effective on multi-domain learning. It reflects that the domain-common features can be easily adapted

to different domains via tuning only the adaptive layers. We further explore the power of CovNorm by deploying it on the large-scale image recognition and parametrize it in a sample-wise manner with an input dependent branch. The new framework is denoted as dynamic convolution decomposition (DCD). Surprisingly, it not only achieves an excellent performance on large-scale image classification dataset- ImageNet on different backbones but also successfully addresses the two issues, i.e., model redundancy and hardness of optimization, existing in the previous dynamic modules built on convolutional layers. More important, we reveal the fundamental reason that leads to these drawbacks mathematically for the previous methods, which is the application of dynamic attention over channel groups in a high dimensional latent space. And the proposed DCD can perfectly overcome this problem by applying dynamic channel fusion in a low dimensional space.

1.3.3 Large-Scale Image Recognition with Extremely Low FLOPs

The previously introduced dynamic operator is built on existing backbones, i.e., ResNet and MobileNet V2. They are mainly leveraged by the devices with computational budget ranging from the order of 100 MFLOPs to 10 GFLOPs. In order to extend the application of deep learning to some extreme cases of computational resources, e.g., below 10 MFLOPs, we present MicroNet. We handle the issue of extremely low FLOPs based upon two design principles: (a) avoiding the reduction of network width by lowering the node connectivity, and (b) compensating for the reduction of network depth by introducing more complex non-linearity per layer. First, we propose Micro-Factorized convolution to factorize both pointwise and depthwise convolutions into low rank matrices for a good tradeoff between the number of channels and input/output connectivity. Second, we propose a new activation function, named Dynamic Shift-Max, to improve the non-linearity via maxing out multiple dynamic fusions between an input feature map and its circular channel shift. The fusions are dynamic as their parameters are adapted to the input. Building upon Micro-Factorized convolution and Dynamic Shift-Max, a family

of MicroNets achieve a significant performance gain over the state-of-the-art in the low FLOP regime for several recognition tasks, i.e., large scale image classification, object detection and keypoint detection.

1.3.4 A Bidirectional Learning Framework for Domain Adaptation on Semantic Segmentation

The annotated data is another critical type of resources for deep learning and domain adaptation is one of the key subjects to deal with the issue when the annotated data is lacking. Specifically, domain adaptation assumes the annotated data is restricted to some domains which are different to the domain of the unlabeled data that we care more about. This difference is denoted as domain discrepancy. In order to tackle the domain discrepancy, we propose a novel bidirectional learning framework. The framework is mainly designed for segmentation tasks and it contains two parts- image translation model and segmentation model. Using the bidirectional learning, the image translation model and the segmentation model can be learned alternatively and promote to each other. Furthermore, we propose a self-training algorithm to learn a better segmentation model and in return improve the image translation model. Experiments show that our method is superior to the state-of-the-art methods in domain adaptation of segmentation with a big margin. More critical, bidirecctional learning framework reveals that the synergy of image translation, segmentation model and self-training.

1.3.5 Multi-Source Domain Adaptation with Dynamic Transfer

Motivated by the findings given by the bidirectional learning framework, a more complex domain adaptation task- multi-source domain adaptation is considered. The multi-source domain adaptation task considers the source domain as an union of images with various styles (domains). Recent works focus on learning a domain-agnostic model, of which the parameters are static.

However, such a static model is difficult to handle conflicts across multiple domains, and suffers from a performance degradation in both source domains and target domain. Inspired by the success of dynamic networks for supervised cases, we present dynamic transfer to address domain conflicts. The key insight is that adapting model across domains is achieved via adapting model across samples. Thus, it breaks down source domain barriers and turns multi-source domains into a single-source domain. This also simplifies the alignment between source and target domains, as it only requires the target domain to be aligned with any part of the union of source domains. Furthermore, we find dynamic transfer can be simply modeled by aggregating residual matrices and a static convolution matrix. Experimental results show that, without using domain labels, our dynamic transfer outperforms the state-of-the-art method by a large margin on the large multi-source domain adaptation datasets– DomainNet.

1.4 Organization of the Thesis

The thesis is organized as follows. In Chapter 2, we introduce the domain-wise dynamic framework for multi-domain learning, which switches the efficient dynamic module (CovNorm) according to the dataset (domain) ID. In Chapter 3, we extend the dynamic module from the manner of domain-wise to sample-wise for the large scale image recognition problem and propose a new framework named dynamic convolution decompostion (DCD). Chapter 4 further explores the network design for image recognition problem under extremely low FLOPs and presents MicroNet that is formed by a novel Micro-Factorized convolution and a dynamic activation function, i.e., Dynamic Shift-Max. Chapter 5 discusses the domain adaptation problem on semantic segmentation with a bidirectional learning (BDL) framework that unifies several key techniques used by domain adaptation. Chapter 6 extends the domain adaptation task from single-source domain to multi-source domain and presents a novel dynamic module denoted as dynamic transfer. In the last, Chapter 7 concludes the thesis.

Chapter 2

Efficient Multi-Domain Learning

2.1 Introduction

Convolutional nerual networks (CNNs) have enabled transformational advances in classification, object detection and segmentation, among other tasks. However they have non-trivial complexity. State-of-the-art models contain millions of parameters and require implementation in expensive GPUs. This creates problems for applications with computational constraints, such as mobile devices or consumer electronics. Figure 2.1 illustrates the problem in the context of a smart home equipped with an ecology of devices such as a camera that monitors package delivery and theft, a fridge that keeps track of its content, a treadmill that adjusts fitness routines to the facial expression of the user, or a baby monitor that keeps track of the state of a baby. As devices are added to the ecology, the GPU server in the house must switch between a larger number of classification, detection, and segmentation tasks. Similar problems will be faced by mobile devices, robots, smart cars, etc.

Under the current deep learning paradigm, this task switching is difficult to perform. The predominant strategy is to use a different CNN to solve each task. Since only a few models can be cached in the GPU, and moving models in and out of cache adds too much overhead to enable real-time task switching, there is a need for very efficient parameter sharing across tasks. The individual networks should share most of their parameters, which would always reside on the GPU. A remaining small number of task specific parameters would be switched per task. This problem is known as *multi-domain learning* (MDL) and has been addressed with the architecture of Figure 2.1 [17, 18]. This consists of set of *fixed* layers (denoted as '**F**') shared by all tasks and a set of task specific *adaptation* layers (denoted as '**A**') fine-tunned to each task. If the **A** layers are much smaller than the **F** layers, many models can be cached simultaneously. Ideally, the **F** layers should be pre-trained, e.g. on ImageNet, and used by all tasks without additional training, enabling the use of special purpose chips to implement the majority of the computations. While **A** layers would still require a processing unit, the small amount of computation could enable the

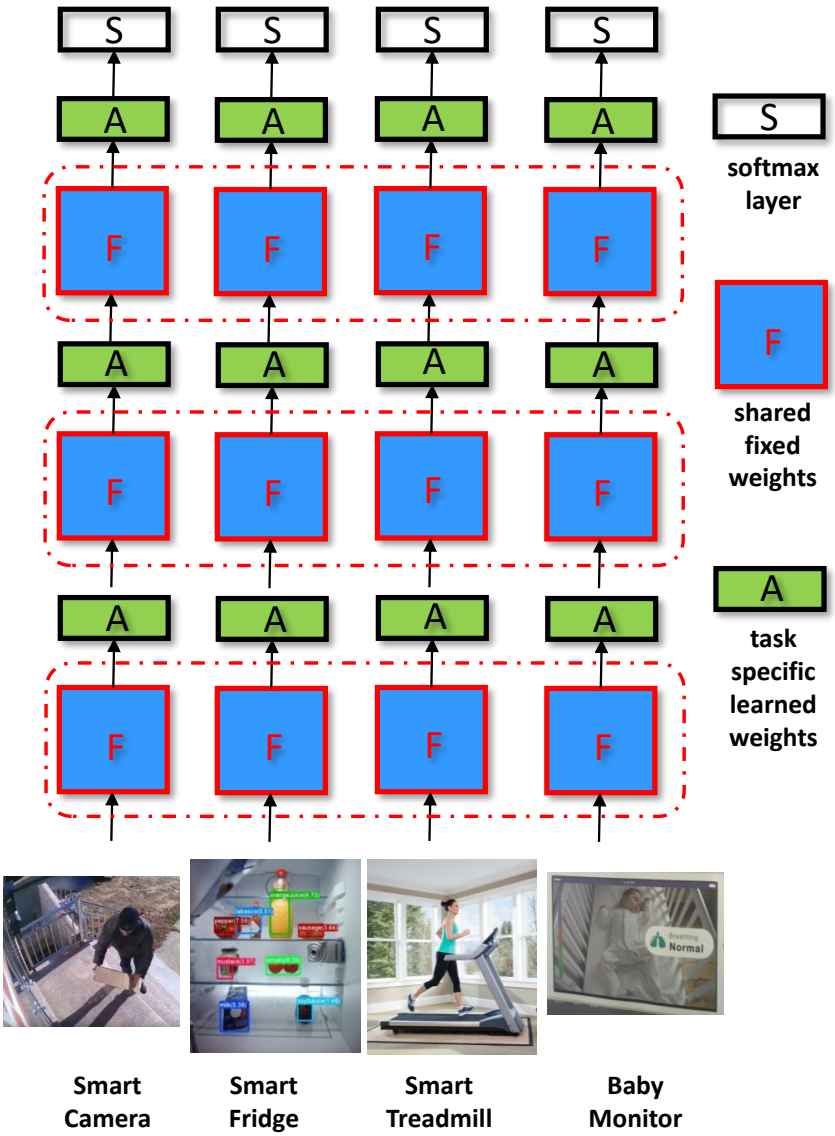


Figure 2.1: Multi-domain learning addresses the efficient solution of several tasks, defined on different domains. Each task is solved by a different network with shared and fixed layers **F**, which contain the majority of network parameters. These are complemented by small task-specific adaptation layers **A**.

use of a CPU, making it cost-effective to implement each network on the device itself.

In summary, MDL aims to maximize the performance of the network ecology while minimizing the ratio of task specific (**A**) to total parameters (both types **F** and **A**) per network. [17, 18] have shown that the architecture of Figure 2.1 can match the performance of fully

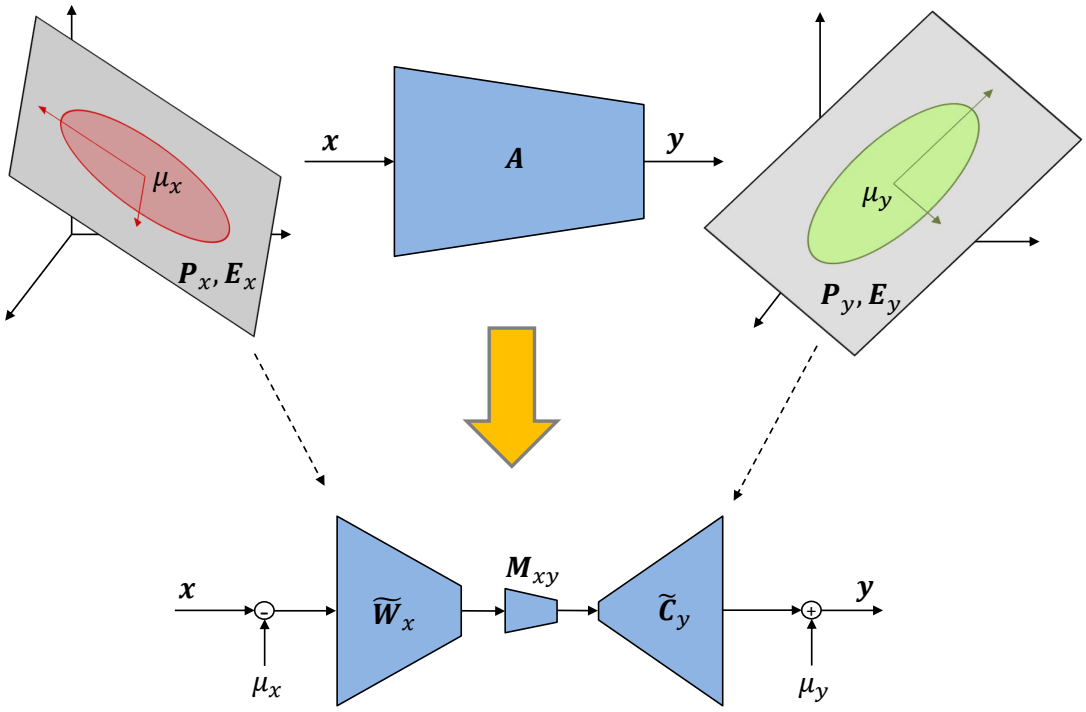


Figure 2.2: Covariance normalization. Each adaptation layer **A** is approximated by three transformations: $\tilde{\mathbf{W}}_x$, which implements a projection onto the PCA space of the input **x**, $\tilde{\mathbf{W}}_y$, which reconstructs the PCA space of the output **y**, and a mini-adaptation layer $\mathbf{M}_{xy}$.

fine-tuning each network in the ecology, even when **A** layers contain as few as 10% of the total parameters. In this chapter, we show that **A** layers can be substantially further shrunk, using a data-driven low-rank approximation. As illustrated in Figure 2.2, this is based on transformations that match the 2^{nd}-order statistics of the **A** layer inputs and outputs. Given principal component analyses (PCAs) of both input and output, the layer is approximated by a *recoloring transformation:* a projection into input PCA space, followed by a reconstruction into the output PCA space. By controlling the intermediate PCA dimensions, the method enables low-dimensional approximations of different input and output dimensions. To correct the mismatch (between PCA components) of two PCAs learned independently, a small *mini-adaptation* layer is introduced between the two PCA matrices, and fine-tunned on the target target.

Since the overall transformation generalizes *batch normalization*, the method is denoted *covariance normalization* (CovNorm). CovNorm is shown to outperform, with both theoretical

and experimental arguments, purely geometric methods for matrix approximation, such as the singular value decomposition (SVD) [19], fine-tuning of the original **A** layers [17, 18], or adaptation based on batch normalization [20]. It is also quite simple, requiring two PCAs and the fine-tuning of a very small mini-adaptation layer per **A** layer and task. Experimental results show that it can outperform full network fine-tuning while reducing **A** layers to as little as 0.53% of the total parameters. When all tasks can be learned together, **A** layers can be further reduced to 0.51% of the full model size. This is achieved by combining the individual PCAs into a global PCA model, of parameters shared by all tasks, and only fine-tunning mini-adaptation layers in a task specific manner.

2.2 Related Work

MDL is a transfer learning problem, namely the transfer of a model trained on a *source* learning problem to an ecology of *target* problems. This makes it related to different types of transfer learning problems, which differ mostly in terms of input, or *domain,* and range space, or *task*.

2.2.1 Task Transfer

Task transfer addresses the use of a model trained on a source task to the solution of a target task. The two tasks can be defined on the same or different domains. Task transfer is prevalent in deep learning, where a CNN pre-trained on a large source dataset, such as ImageNet, is usually fine-tunned [21] to a target task. While extremely effective and popular, full network fine-tunning changes most network parameters, frequently all. MDL addresses this problem by considering multiple target tasks and extensive parameter sharing between them.

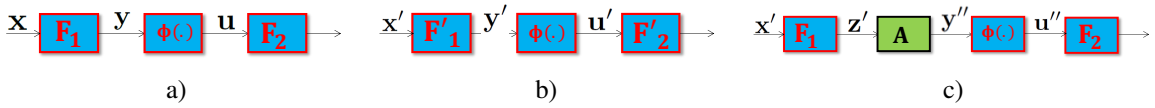


Figure 2.3: a) original network, b) after fine-tuning, and c) with adaptation layer **A**. In all cases, $\mathbf{W}_i$ is a weight layer and $\phi(.)$ a non-linearity.

2.2.2 Domain Adaptation

In domain adaptation, the source and target tasks are the same, and a model trained on a source domain is transfered to a target domain. Domain adaptation can be supervised, in which case labeled data is available for the target domain, or unsupervised, where it is not. Various strategies have been used to address these problems. Some methods seek the network parameters that minimize some function of the distance between feature distributions in the two domains [22, 23, 24]. Others introduce an adversarial loss that maximizes the confusion between the two domains [25, 9]. A few methods have also proposed to do the transfer at the image level, e.g. using GANs [26] to map source images into (labeled) target images, then used to learn a target classifier [27, 28, 29]. All these methods exploit the commonality of source and domain tasks to align source and target domains. This is unlike MDL, where source and target tasks are different. Nevertheless, some mechanisms proposed for domain adaptation can be used for MDL. For example, [30, 31] use a batch normalization layer to match the statistics of source and target data, in terms of means and standard deviation. This is similar to an early proposal for MDL [20]. We show that these mechanisms underperform covariance normalization.

2.2.3 Multi-Task Learning

Multi-task learning [32, 33] addresses the solution of multiple tasks by the same model. It assumes that all tasks have the same visual domain. Popular examples include classification and bounding box regression in object detection [34, 35], joint estimation of surface normals and depth [36] or segmentation [37], joint representation in terms of attributes and facial land-

marks [38, 39], among others. Multi-task learning is sometimes also used to solve auxiliary tasks that strengthen performance of a task of interest, e.g. by accounting for context [40], or representing objects in terms of classes and attributes [41, 37, 42, 43]. Recently, there have been attempts to learn models that solve many problems jointly [44, 45, 46].

Most multi-task learning approaches emphasize the learning of the interrelationships between tasks. This is frequently accomplished by using a single network, combining domain agnostic lower-level network layers with task specific network heads and loss functions [38, 36, 40, 41, 35, 45], or some more sophisticated forms of network branching [43]. The branching architecture is incompatible with MDL, where each task has its own input, different from those of all other tasks. Even when multi-task learning is addressed with multiple tower networks, the emphasis tends to be on inter-tower connections, e.g. through cross-stitching [37, 47]. In MDL, such connections are not feasible, because different networks can join the ecology of Figure 2.1 asynchronously, as devices are turned on and off.

2.2.4 Lifelong Learning

Lifelong learning aims to learn multiple tasks sequentially with a shared model. This can be done by adapting the parameters of a network or adapting the network architecture. Since training data is discarded upon its use, constraints are needed to force the model to remember what was previously learned. Methods that only change parameters either use the model output on previous tasks [48], previous parameters values [49], or previous network activations [50] to regularize the learning of the target task. They are very effective at parameter sharing, since a single model solves all tasks. However, this model is not optimal for any specific task, and can perform poorly on all tasks, depending on the mismatch between source and target domains [51]. We show that they can significantly underperform MDL with CovNorm. Methods that adapt the network architecture usually add a tower per new task [52, 53]. These methods have much larger complexity than MDL, since several towers can be needed to solve a single task [52], and there is

no sharing of fixed layers across tasks.

2.2.5 Multi-Domain Learning

This work builds on previous attempts at MDL, which have investigated different architectures for the adaptation layers of Figure 2.1. [20] used a BN layer [54] of parameters tunned per task. While performing well on simple datasets, this does not have enough degrees of freedom to support transfer of large CNNs across different domains. More powerful architectures were proposed by [18], who used a 1×1 convolutional layer and [17], who proposed a ResNet-style residual layer, known as a residual adaptation (RA) module. These methods were shown to perform surprisingly well in terms of recognition accuracy, equaling or surpassing the performance of full network fine-tunning, but can still require a substantial number of adaptation parameters, typically 10% of the network size. [19] addressed this problem by combining adapters of multiple tasks into a large matrix, which is approximated with an SVD. This is then fine-tuned on each target dataset. Compressing adaptation layers in this way was shown to reduce adaptive parameter counts to approximately half of [17]. However, all tasks have to be optimized simultaneously. We show that CovNorm enables a further ten-fold reduction in adaptation layer parameters, without this limitation, although some additional gains are possible with joint optimization.

2.3 MDL by Covariance Normalization

In this section, we introduce the CovNorm procedure for MDL with deep networks.

2.3.1 Multi-Domain Learning

Figure 2.3 a) motivates the use of **A** layers in MDL. The figure depicts two fixed weight layers, $\mathbf{F}_1$ and $\mathbf{F}_2$, and a non-linear layer $\phi(.)$ in between. Since the fixed layers are pre-trained on a *source* dataset $\mathcal{S}$, typically ImageNet, all weights are optimized for the source statistics. For

standard losses, such as cross entropy, this is a maximum likelihood (ML) procedure that matches $\mathbf{F}_1$ and $\mathbf{F}_2$ to the statistics of activations $\mathbf{x}, \mathbf{y}$ and $\mathbf{u}$ in $\mathcal{S}$. However, when the CNN is used on a different *target* domain, the statistics of these variables change and $\mathbf{F}_1, \mathbf{F}_2$ are no longer an ML solution. Hence, the network is sub-optimal and must be fine-tunned on a target dataset $\mathcal{T}$. This is denoted full network fine-tuning and converts the network into an ML solution for $\mathcal{T}$, with the outcome of Figure 2.3 b). In the target domain, the intermediate random variables become $\mathbf{x}'$, $\mathbf{y}'$, and $\mathbf{u}'$ and the weights are changed accordingly, into $\mathbf{F}'_1$ and $\mathbf{F}'_2$.

While very effective, this procedure has two drawbacks, which follow from updating all weights. First, it can be computationally expensive, since modern CNNs have large weight matrices. Second, because the weights $\mathbf{F}'_i$ are not optimal for $\mathcal{S}$, i.e., the CNN forgets the source task, there is a need to store and implement two CNNs to solve both tasks. This is expensive in terms of storage and computation and increases the complexity of managing the network ecology. A device that solves both tasks must store two CNNs and load them in and out of cache when it switches between the tasks. These problems are addressed by the MDL architecture of Figure 2.1, which is replicated in greater detail on Figure 2.3 c). It introduces an *adaptation layer* $\mathbf{A}$ and fine-tunes this layer only, leaving $\mathbf{F}_1$ and $\mathbf{F}_2$ unchanged. In this case, the statistics of the input are still those of $\mathbf{x}'$, but the distributions along the network are now those of $\mathbf{z}', \mathbf{y}''$, and $\mathbf{u}''$. Since $\mathbf{F}_1$ is fixed, nothing can be done about $\mathbf{z}'$. However, the fine-tuning of $\mathbf{A}$ encourages the statistics of $\mathbf{y}''$ to match those of $\mathbf{y}'$, i.e., $\mathbf{y}'' = \mathbf{y}'$ and thus $\mathbf{u}'' = \mathbf{u}'$. Even if $\mathbf{A}$ cannot match statistics exactly, the mismatch is reduced by repeating the procedure in subsequent layers, e.g. introducing a second $\mathbf{A}$ layer after $\mathbf{F}_2$, and optimizing adaptation matrices as a whole.

2.3.2 Adaptation Layer Size

Obviously, MDL has limited interest if $\mathbf{A}$ has size similar to $\mathbf{F}_1$. In this case, each domain has as many adaptation parameters as the original network, all networks have twice the size, task switching is complex, and training complexity is equivalent to full fine-tunning of the original

network. On the other hand, if $\mathbf{A}$ is much smaller than $\mathbf{F}_1$, MDL is computationally light and task-switching much more efficient. In summary, the goal is to introduce an adaptation layer $\mathbf{A}$ as *small as possible,* but still powerful enough to *match the statistics* of $\mathbf{y}'$ and $\mathbf{y}''$. A simple solution is to make $\mathbf{A}$ a batch normalization layer [54]. This was proposed in [20] but, as discussed below, is not effective. To overcome this problem, [18] proposed a linear transformation $\mathbf{A}$ and [17] adopted the residual structure of [55], i.e., an adaptation layer $\mathbf{T} = (\mathbf{I} + \mathbf{A})$. To maximize parameter savings, $\mathbf{A}$ was implemented with a 1×1 convolutional layer in both cases.

This can, however, still require a non-trivial number of parameters, especially in upper network layers. Let $\mathbf{F}_1$ convolve a bank of d filters of size $k \times k \times l$ with l feature maps. Then, $\mathbf{F}_1$ has size dk^2l, $\mathbf{y}$ is d dimensional, and $\mathbf{A}$ a $d \times d$ matrix. Since in upper network layers k is usually small and $d > l$, $\mathbf{A}$ can be only marginally smaller than $\mathbf{F}_1$. [19] exploited redundancies across tasks to address this problem, creating a matrix with the $\mathbf{A}$ layer parameters of multiple tasks and computing a low-rank approximation of this matrix with an SVD. The compression achieved with this approximation is limited, because the approximation is purely geometric, not taking into account the statistics of $\mathbf{z}'$ and $\mathbf{y}'$. In this chapter, we propose a more efficient solution, motivated by the interpretation of $\mathbf{A}$ as converting the statistics of $\mathbf{z}'$ into those of $\mathbf{y}'$. It is assumed that the fine-tuning of $\mathbf{A}$ produces an output variable $\mathbf{y}''$ whose statistics match those of $\mathbf{y}'$. This could leverage adaptation layers in other layers of the network, but that is not important for the discussion that follows. The only assumption is that $\mathbf{y}'' = \mathbf{y}'$. The goal is to replace $\mathbf{A}$ by a simpler matrix that maps $\mathbf{z}'$ into $\mathbf{y}'$. For simplicity, we drop the primes and notation of Figure 2.3 in what follows, considering the problem of matching statistics between input $\mathbf{x}$ and output $\mathbf{y}$ of a matrix $\mathbf{A}$.

2.3.3 Geometric Approximations

One possibility is to use a purely geometric solution [19]. Geometrically, the closest low rank approximation of a matrix $\mathbf{A}$ is given by the SVD, $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$. More precisely, the

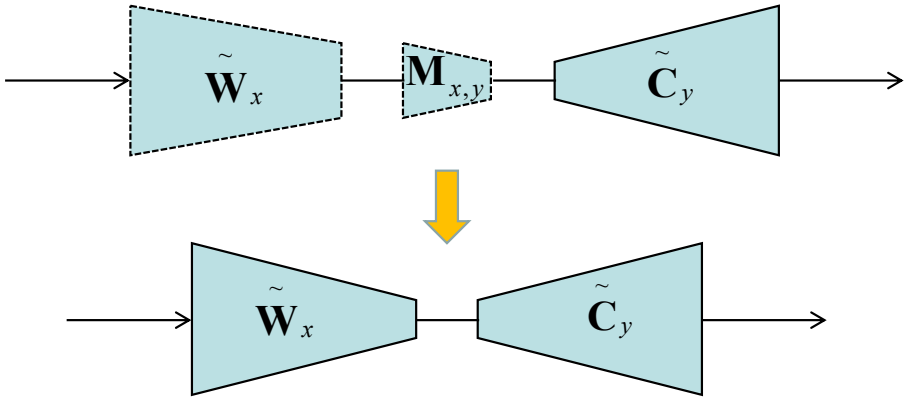


Figure 2.4: Top: CovNorm approximates adaptation layer $\mathbf{A}$ by a sequence of whitening $\tilde{\mathbf{W}}_x$, mini-adaptation $\mathbf{M}_{x,y}$, and coloring $\tilde{\mathbf{C}}_y$ operations. Bottom: after covnorm, the mini adaptation layer can be absorbed into $\tilde{\mathbf{W}}_x$ (shown in the figure) or $\tilde{\mathbf{C}}_y$.

minimum Frobenius norm approximation $\tilde{\mathbf{A}} = \arg\min_{\{\mathbf{B}|rank(\mathbf{B})=r\}} ||\mathbf{A}-\mathbf{B}||_F^2$, where $r < rank(\mathbf{A})$, is $\tilde{\mathbf{A}} = \mathbf{U}\tilde{\mathbf{S}}\mathbf{V}^T$ where $\tilde{\mathbf{S}}$ contains the r largest singular values of $\mathbf{A}$. This can be written as $\tilde{\mathbf{A}} = \mathbf{CW}$, where $\mathbf{C} = \mathbf{U}\sqrt{\tilde{\mathbf{S}}}$ and $\mathbf{W} = \sqrt{\tilde{\mathbf{S}}}\mathbf{V}^T$. If $A \in \mathbb{R}^{d\times d}$, these matrices have a total of $2rd$ parameters. An even simpler solution is to define $\mathbf{C} \in \mathbb{R}^{d\times r}$ and $\mathbf{W} \in \mathbb{R}^{r\times d}$, replace $\mathbf{A}$ by their product in Figure 2.3 c), and fine-tune the two matrices instead of $\mathbf{A}$. We denote this as the *fine-tunned approximation* (FTA). These approaches are limited by their purely geometric nature. Note that d is determined by the source model (output dimension of $\mathbf{F}_1$) and fixed. On the other hand, the dimension r should depend on the target dataset $\mathcal{T}$. Intuitively, if $\mathcal{T}$ is much smaller than $\mathcal{S}$, or if the target task is much simpler, it should be possible to use a smaller r than otherwise. There is also no reason to believe that a single r, or even a single ratio r/d, is suitable for all network layers. While r could be found by cross-validation, this becomes expensive when there are multiple adaptation layers throughout the CNN. We next introduce an alternative, data driven, procedure that bypasses these difficulties.

2.3.4 Covariance Matching

Assume that, as illustrated in Figure 2.2, $\mathbf{x}$ and $\mathbf{y}$ are Gaussian random variables of means μ_x, μ_y and covariances $\mathbf{\Sigma}_x$, $\mathbf{\Sigma}_y$, respectively, related by $\mathbf{y} = \mathbf{Ax}$. Let the covariances have

eigendecomposition

$$\boldsymbol{\Sigma}_x = \mathbf{P}_x \mathbf{E}_x \mathbf{P}_x^T \qquad \boldsymbol{\Sigma}_y = \mathbf{P}_y \mathbf{E}_y \mathbf{P}_y^T \tag{2.1}$$

where $\mathbf{P}_x, \mathbf{P}_y$ contain eigenvectors as columns and $\mathbf{E}_x, \mathbf{E}_y$ are diagonal eigenvalue matrices. We refer to the triplet $\mathcal{P}_x = (\mathbf{P}_x, \mathbf{E}_x, \boldsymbol{\mu}_x)$ as the PCA of $\mathbf{x}$. Then, it is well known that the statistics of $\mathbf{x}$ and $\mathbf{y}$ are related by

$$\boldsymbol{\mu}_y = \mathbf{A}\boldsymbol{\mu}_x \qquad \boldsymbol{\Sigma}_y = \mathbf{A}\boldsymbol{\Sigma}_x \mathbf{A}^T \tag{2.2}$$

and, combining (2.1) and (2.2), $\mathbf{P}_y \mathbf{E}_y \mathbf{P}_y^T = \mathbf{A}\mathbf{P}_x \mathbf{E}_x \mathbf{P}_x^T \mathbf{A}^T$. This holds when $\mathbf{P}_y \sqrt{\mathbf{E}_y} = \mathbf{A}\mathbf{P}_x \sqrt{\mathbf{E}_x}$ or, equivalently,

$$\begin{aligned} \mathbf{A} &= \mathbf{P}_y \sqrt{\mathbf{E}_y} \sqrt{\mathbf{E}_x^{-1}} \mathbf{P}_x^T. & (2.3) \\ &= \mathbf{C}_y \mathbf{W}_x & (2.4) \end{aligned}$$

where $\mathbf{W}_x = \sqrt{\mathbf{E}_x^{-1}} \mathbf{P}_x^T$ is the "whitening matrix" of $\mathbf{x}$ and $\mathbf{C}_y = \mathbf{P}_y \sqrt{\mathbf{E}_y}$ the "coloring matrix" of $\mathbf{y}$. It follows that (2.2) holds if $\mathbf{y} = \mathbf{A}\mathbf{x}$ is implemented with a sequence of two operations. First, $\mathbf{x}$ is mapped into a variable $\mathbf{w}$ of zero mean and identity covariance, by defining

$$\mathbf{w} = \mathbf{W}_x (\mathbf{x} - \boldsymbol{\mu}_x). \tag{2.5}$$

Second, $\mathbf{w}$ is mapped into $\mathbf{y}$ with

$$\mathbf{y} = \mathbf{C}_y \mathbf{w} + \boldsymbol{\mu}_y. \tag{2.6}$$

In summary, for Gaussian $\mathbf{x}$, the effect of $\mathbf{A}$ is simply the combination of a whitening of $\mathbf{x}$ followed by a colorization with the statistics of $\mathbf{y}$.

2.3.5 Covariance Normalization

The interpretation of the adaptation layer as a recoloring operation (whitening + coloring) sheds light on the number of parameters effectively needed for the adaptation, since the PCAs $\mathcal{P}_x, \mathcal{P}_y$ capture the *effective* dimensions of $\mathbf{x}$ and $\mathbf{y}$. Let k_x (k_y) be the number of eigenvalues significantly larger than zero in $\mathbf{E}_x$ ($\mathbf{E}_y$). Then, the whitening and coloring matrices can be approximated by

$$\tilde{\mathbf{W}}_x = \sqrt{\tilde{\mathbf{E}}_x^{-1}}\tilde{\mathbf{P}}_x^T \qquad \tilde{\mathbf{C}}_y = \tilde{\mathbf{P}}_y\sqrt{\tilde{\mathbf{E}}_y} \tag{2.7}$$

where $\tilde{\mathbf{E}}_x \in \mathbb{R}^{k_x \times k_x}$ ($\tilde{\mathbf{E}}_y \in \mathbb{R}^{k_y \times k_y}$) contains the non-zero eigenvalues of $\mathbf{\Sigma}_x$ ($\mathbf{\Sigma}_y$), and $\tilde{\mathbf{P}}_x \in \mathbb{R}^{d \times k_x}$ ($\tilde{\mathbf{P}}_y \in \mathbb{R}^{d \times k_y}$) the corresponding eigenvectors. Hence, $\mathbf{A}$ is well approximated by a pair of matrices ($\tilde{\mathbf{W}}_x$, $\tilde{\mathbf{C}}_y$) totaling $d(k_x + k_y)$ parameters.

On the other hand, the PCAs are only defined up to a permutation, which assigns an ordering to eigenvalues/eigenvectors. When the input and output PCAs are computed independently, the principal components may not be aligned. This can be fixed by introducing a permutation matrix between $\mathbf{C}_y$ and $\mathbf{W}_x$ in (2.4). The assumption that all distributions are Gaussian also only holds approximately in real networks. To account for all this, we augment the recoloring operation with a mini-adaptation layer $\mathbf{M}_{x,y}$ of size $k_x \times k_y$. This leads to the covariance normalization (CovNorm) transform

$$\tilde{\mathbf{y}} = \tilde{\mathbf{C}}_y\mathbf{M}_{x,y}\tilde{\mathbf{W}}_x(\mathbf{x} - \boldsymbol{\mu}_x) + \boldsymbol{\mu}_y, \tag{2.8}$$

where $\mathbf{M}_{x,y}$ is learned by fine-tuning on the target dataset $\mathcal{T}$. Beyond improving recognition performance, this has the advantage of further parameters savings. The direct implementation of (2.8) increases the parameter count to $d(k_x + k_y) + k_x k_y$. However, after fine-tuning, $\mathbf{M}_{x,y}$ can be absorbed into one of the two other matrices , as shown in Figure 2.4. When $k_x > k_y$, $\mathbf{M}_{x,y}\tilde{\mathbf{W}}_x$

has dimension $k_y \times d$ and replacing the two matrices by their product reduces the total parameter count to $2dk_y$. In this case, we say that $\mathbf{M}_{x,y}$ is absorbed into $\tilde{\mathbf{W}}_x$. Conversely, if $k_x < k_y$, $\mathbf{M}_{x,y}$ can be absorbed into $\tilde{\mathbf{C}}_y$. Hence, the total parameter count is $2d\min(k_x, k_y)$. CovNorm is summarized in Algorithm 1.

Algorithm 1: Covariance Normalization

Data: source $\mathcal{S}$ and target $\mathcal{T}$

1 Insert an adaptation layer $\mathbf{A}$ on a CNN trained on $\mathcal{S}$ and fine-tune $\mathbf{A}$ on $\mathcal{T}$.
2 Store the layer input and output PCAs $\mathcal{P}_x$, $\mathcal{P}_y$, select the k_x, k_y non-zero eigenvalues and corresponding eigenvectors from each PCA, and compute $\tilde{\mathbf{C}}_y, \tilde{\mathbf{W}}_x$ with (2.7).
3 add mini-adaptation layer $\mathbf{M}_{x,y}$ and replace $\mathbf{A}$ by (2.8). Note that, as usual, the constant $\tilde{\mathbf{C}}_y\mathbf{M}_{x,y}\tilde{\mathbf{W}}_x\mu_x + \mu_y$ can be implemented with a vector of biases.
4 fine-tune $\mathbf{M}_{x,y}$ with $\tilde{\mathbf{W}}_x$ and $\tilde{\mathbf{C}}_y$ on $\mathcal{T}$ and absorb $\mathbf{M}_{x,y}$ into the larger of $\tilde{\mathbf{W}}_x$ and $\tilde{\mathbf{C}}_y$.

2.3.6 The Importance of Covariance Normalization

The benefits of covariance matching can be seen by comparison to previously proposed MDL methods. Assume, first, that $\mathbf{x}$ and $\mathbf{y}$ consist of *independent* features. In this case, $\mathbf{P}_x, \mathbf{P}_y$ are identity matrices and (2.5)-(2.6) reduce to

$$y_i = \sqrt{e_{y,i}}\frac{x_i - \mu_{x,i}}{\sqrt{e_{x,i}}} + \mu_{y,i}, \tag{2.9}$$

which is the batch normalization equation. Hence, CovNorm is a generalized form of the latter. There are, however, important differences. First, there is no batch. The normalizing distribution $\mathbf{x}$ is now the distribution of the feature responses of layer $\mathbf{F}_1$ on the target dataset $\mathcal{T}$. Second, the goal is not to facilitate the learning of $\mathbf{F}_2$, but produce a feature vector $\mathbf{y}$ with statistics matched to $\mathbf{F}_2$. This turns out to make a significant difference. Since, in regular batch normalization, $\mathbf{F}_2$ is allowed to change, it can absorb any initial mismatch with the independence assumption. This is not the case for MDL, where $\mathbf{F}_2$ is *fixed.* Hence, (2.9) usually fails, significantly underperforming (2.5)-(2.6).

Next, consider the geometric solution. Since CovNorm reduces to the product of two tall matrices, e.g. $\mathbf{K} = \tilde{\mathbf{C}}_y \mathbf{M}_{x,y}$ and $\mathbf{L} = \tilde{\mathbf{W}}_x$ of size $d \times k_x$, it should be possible to replace it with the fine-tuned approximation based on two matrices of this size. Here, there are two difficulties. First, k_x is not known in the absence of the PCA decomposition. Second, in our experience, even when k_x is set to the value used by PCA, the fine-tuned approximation does not work. As shown in the experimental section, when the matrices are initialized with Gaussian weights, performance can decrease significantly. This is an interesting observation because $\mathbf{A}$ is itself initialized with Gaussian weights. It appears that a good initialization is more critical for the low-rank matrices.

Finally, CovNorm can be compared to the SVD, $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$. From (2.3), this holds whenever $\mathbf{V} = \mathbf{P}_x$, $\mathbf{S} = \sqrt{\mathbf{E}_y}\sqrt{\mathbf{E}_x^{-1}}$ and $\mathbf{U} = \mathbf{P}_y$. The problem is that the singular value matrix $\mathbf{S}$ conflates the variances of the input and output PCAs. The fact that $s_i = e_{y,i}/e_{x,i}$ has two important consequences. First, it is impossible to recover the dimensions k_x and k_y by inspection of the singular values. Second, the low-rank criteria of selecting the largest singular values is *not* equivalent to CovNorm. For example, the principal components of $\mathbf{x}$ with largest eigenvalues $e_{x,i}$ have the smallest singular values s_i. Hence, it is impossible to tell if singular vectors $\mathbf{v}_i$ of small singular values are the most important (PCA components of large variance for $\mathbf{x}$) or the least important (noise). Conversely, the largest singular values can simply signal the least important input dimensions. CovNorm eliminates this problem by explicitly selecting the important input and output dimensions.

2.3.7 Joint Training

[19] considered a variant of MDL where the different tasks of Figure 2.1 are all optimized simultaneously. This is the same as assuming that a joint dataset $\mathcal{T} = \cup_i \mathcal{T}_i$ is available. For CovNorm, the only difference with respect to the single dataset setting is that the PCAs $\mathcal{P}_x, \mathcal{P}_y$ are now those of the joint data $\mathcal{T}$. These can be derived from the PCAs $\mathcal{P}_{x,i}, \mathcal{P}_{y,i}$ of the individual

target datasets $\mathcal{T}_i$ with

$$\begin{aligned} \boldsymbol{\mu}_{\mathcal{T}} &= \frac{1}{N}\sum_i N_i \boldsymbol{\mu}_i \\ \boldsymbol{\Sigma}_{\mathcal{T}} &= \sum_i \frac{N_i}{N}(\mathbf{P}_i \mathbf{E}_i {\mathbf{P}_i}^T + \boldsymbol{\mu}_i {\boldsymbol{\mu}_i}^T)) - \boldsymbol{\mu}_{\mathcal{T}} \boldsymbol{\mu}_{\mathcal{T}}^T \end{aligned} \tag{2.10}$$

where N_i is the cardinality of $\mathcal{T}_i$. Hence, CovNorm can be implemented by finetuning $\mathbf{A}$ to each $\mathcal{T}_i$, storing the PCAs $\mathcal{P}_{x,i}, \mathcal{P}_{y,i}$, using (2.10) to reconstruct the covariance of $\mathcal{T}$, and computing the global PCA. When tasks are available sequentially, this can be done recursively, combining the PCA of all previous data with the PCA of the new data. In summary, CovNorm can be extended to any number of tasks, with constant storage requirements (a single PCA), and no loss of optimality. This makes it possible to define two CovNorm *modes*.

- *independent:* $\mathbf{A}$ layers of network i are adapted to target dataset $\mathcal{T}_i$. A PCA is computed for $\mathcal{T}_i$ and the mini-adaptation fine-tuned to $\mathcal{T}_i$. This requires $2d\min(k_x, k_y)$ task specific parameters (per layer) per dataset.

- *joint:* a global PCA is learned from $\mathcal{T}$ and $\tilde{\mathbf{C}}_y, \tilde{\mathbf{W}}_x$ shared across tasks. Only a mini-adaptation layer is fine-tuned per $\mathcal{T}_i$. This requires $\min(k_x, k_y)$ task-specific parameters (per layer) per dataset. All $\mathcal{T}_i$ must be available simultaneously.

The independent model is needed if, for example, the devices of Figure 2.1 are produced by different manufacturers.

2.4 Experiments

In this section, we present results for both the independent and joint CovNorm modes.

2.4.1 Experimental Set-up

Dataset: [17] proposed the decathlon dataset for evaluation of MDL. However, this is a collection of relatively small datasets. While sufficient to train small networks, we found it hard to use with larger CNNs. Instead, we used a collection of seven popular vision datasets. **SUN 397** [56] contains 397 classes of scene images and more than a million images. **MITIndoor** [57] is an indoor scene dataset with 67 classes and 80 samples per class. **FGVC-Aircraft Benchmark** [58] is a fine-grained classification dataset of $10,000$ images of 100 types of airplanes. **Flowers102** [59] is a fine-grained dataset with 102 flower categories and 40 to 258 images per class. **CIFAR100** [60] contains $60,000$ tiny images, from 100 classes. **Caltech256** [61] contains $30,607$ images of 256 object categories, with at least 80 samples per class. **SVHN** [62] is a digit recognition dataset with 10 classes and more than $70,000$ samples. In all cases, images are resized to 224×224 and the training and testing splits defined by the dataset are used, if available. Otherwise, 75% is used for training and 25% for testing.

Implementation: In all experiments, fixed **F** layers were extracted from a source **VGG16** [63] model trained on ImageNet. This has convolution layers of dimensions ranging from 64 to 4096. In a set of preliminary experiments, we compared the MDL performance of the architecture of FIgure 2.1 with these **F** layers and adaptation layers implemented with 1) a convolutional layer **A** of kernel size 1×1 [18], 2) the residual adapters $\mathbf{T} = \mathbf{B_2}(\mathbf{I} + \mathbf{AB_1})$ of [17], where $\mathbf{B_1}$ and $\mathbf{B_2}$ are batch normalization layers and **A** as in 1), and 3) the parallel adapters of [19]. Since residual adapters produced the best results, we adopted this structure in all our experiments. However, CovNorm can be used with any of the other structures, or any other matrix **A**. Note that $\mathbf{B}_1$ could be absorbed into **A** after fine-tuning but we have not done so, for consistency with [17].

In all experiments, fine-tuning used initial learning rate of 0.001, reduced by 10 when the loss stops decreasing. After fine-tuning the residual layer, features were extracted at the input and output of **A** and the PCAs $\mathcal{P}_x, \mathcal{P}_y$ computed and used in Algorithm 1. Principal components were selected by the explained variance criterion. Once the eigenvalues e_i were computed and sorted

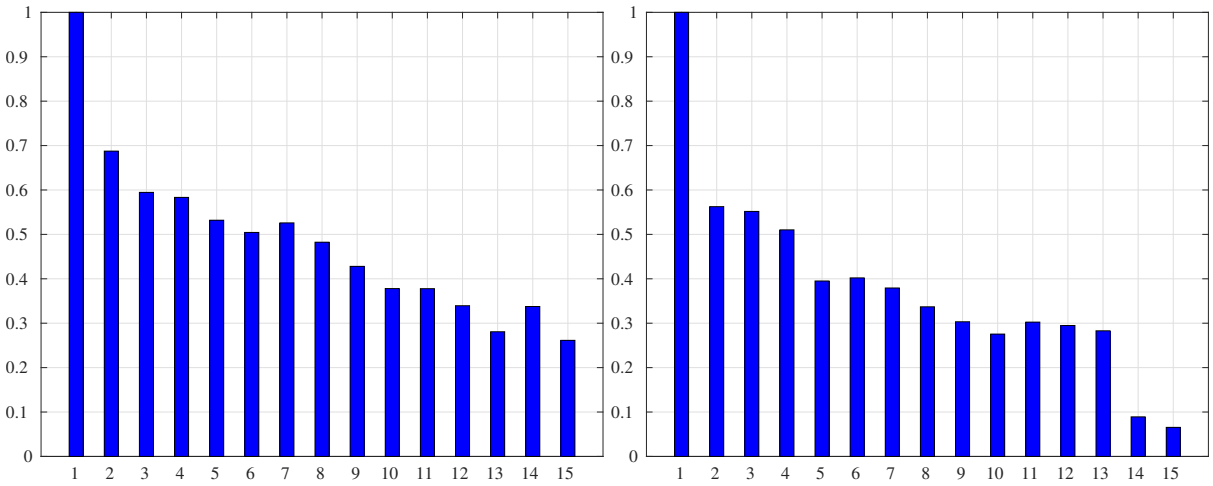


Figure 2.5: Ratio of effective dimensions (η) for different network layers. Left: MITIndoor. Right: CIFAR100.

by decreasing magnitude, i.e., $e_1 \geq e_2 \geq \ldots \geq e_d$, the variance explained by the first i eigenvalues is $r_i = \frac{\sum_{k=1}^{i} e_i}{\sum_{k=1}^{d} e_i}$. Given a threshold t, the smallest index i^* such that $r_{i^*} > t$ was determined, and only the i^* first eigenvalues/eigenvectors were kept. This set the dimensions k_x, k_y (depending on whether the procedure was used on $\mathcal{P}_x$ or $\mathcal{P}_y$). Unless otherwise noted, we used $t = 0.99$, i.e., 99% of the variance was retained.

2.4.2 Benefits of CovNorm

We start with some independent MDL experiments that provide insight on the benefits of CovNorm over previous MDL procedures. While we only report results for MITIndoor and CIFAR100, they are typical of all target datasets. Figure 2.5 shows the ratio $\eta = k_y/k_x$ of effective output to input dimensions, as a function of adaptation layer. It shows that the input of $\mathbf{A}$ typically contains more information than the output. Note that η is rarely one, is almost always less than 0.6, frequently smaller than 0.3, and smallest for the top network layers.

We next compared CovNorm to batch normalization (BN) [20], and geometric approximations based on the fine-tunned approximation (FTA) of Section 2.3.3. We also tested a mix of the geometric approaches (SVD+FTA), where $\mathbf{A}$ was first approximated by the SVD and the matrices

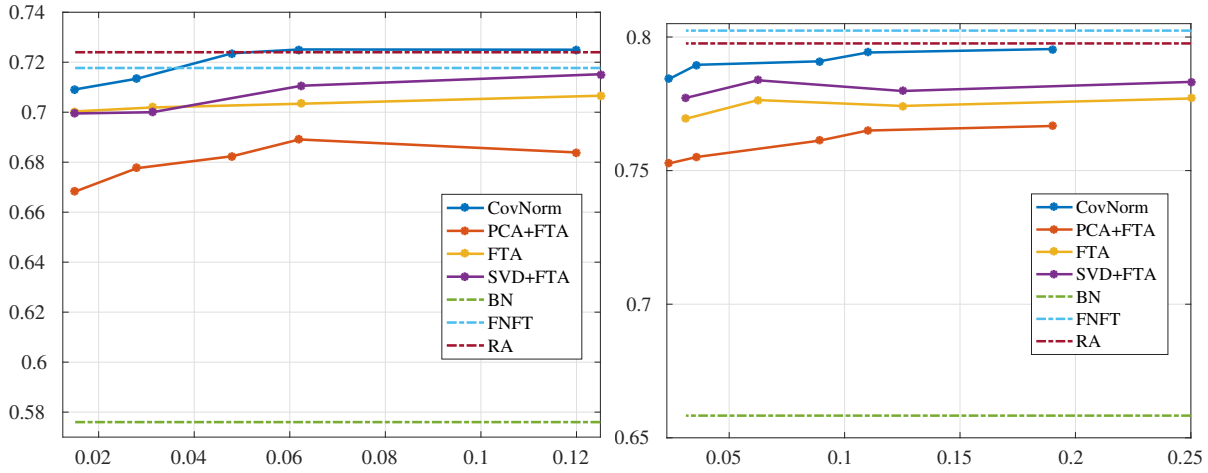


Figure 2.6: Accuracy vs. % of parameters used for adaptation. Left: MITIndoor. Right: CIFAR100.

C, **W** finetuned on $\mathcal{T}$, and a mix of PCA and FTA (PCA+FTA), where the mini-adaptation layer $\mathbf{M}_{x,y}$ of CovNorm was removed and $\tilde{\mathbf{C}}_y, \tilde{\mathbf{W}}_x$ fine-tuned on $\mathcal{T}$, to minimize the PCA alignment problem. All geometric approximations were implemented with low-rank parameter values $r = d/2^i$, where d is the dimension of $\mathbf{x}$ or $\mathbf{y}$ and $i \in \{2,\ldots,6\}$. For CovNorm, the explained variance threshold was varied in $[0.8, 0.995]$. Figure 2.6 shows recognition accuracy vs. the % of parameters. Here, 100% parameters corresponds the adaptation layers of [17]: a network with residual adapters whose matrix $\mathbf{A}$ is fine-tunned on $\mathcal{T}$. This is denoted RA and shown as an upper-bound. A second upper-bound is shown for full network fine-tuning (FNFT). This requires $10\times$ more parameters than RA. BN, which requires close to zero parameters, is shown as a lower bound.

Several observations are possible. First, all geometric approximations underperform CovNorm. For comparable sizes, the accuracy drop of the best geometric method (SVD+FTA) is as large as 2%. This is partly due to the use of a constant low rank r throughout the network. This cannot match the effective, data-dependent, dimensions, which vary across layers (see Figure 2.5). CovNorm eliminates this problem. We experimented with heuristics for choosing variable ranks but, as discussed below (Figure 2.7), could not achieve good performance. Among the geometric

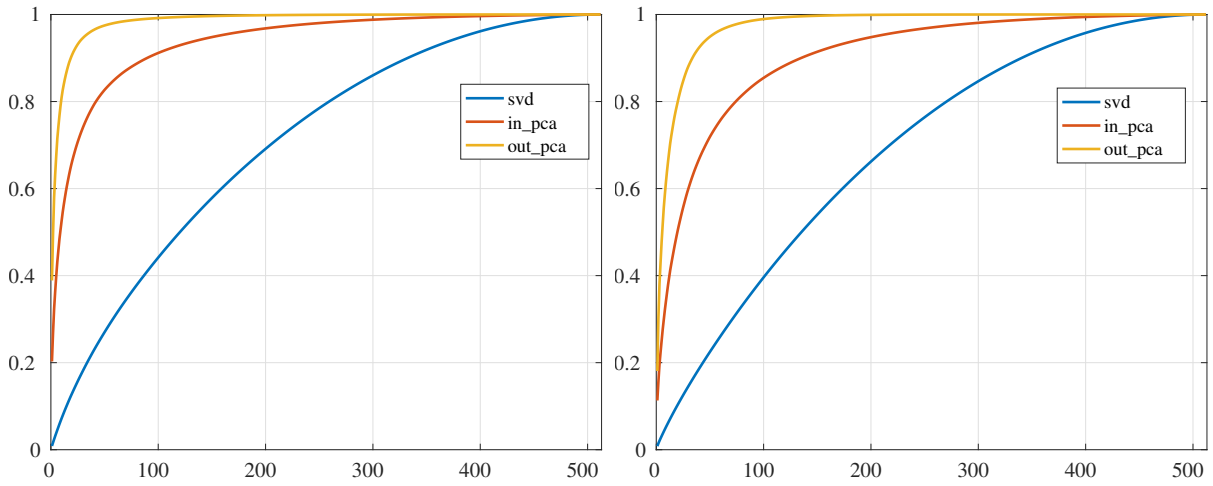


Figure 2.7: Variance explained by eigenvalues of a layer input and output, and similar plot for singular values. Left: MITIndoor. Right: CIFAR100.

approaches, SVD+FTA outperforms FTA, which has performance drops in most of datasets. It is interesting that, while **A** is fine-tuned with random initialization, the process is not effective for the low-rank matrices of FTA. In several datasets, FTA could not match SVD+FTA.

Even more surprising were the weaker results obtained when the random initialization was replaced by the two PCAs (PCA+FTA). Note the large difference between PCA+FTA and CovNorm (up to 4%), which differ by the mini-adaptation layer $\mathbf{M}_{x,y}$. This is explained by the alignment problem of Section 2.3.5. Interestingly, while mini-adaptation layers are critical to overcome this problem, they are as easy to fine-tune as **A**. In fact, the addition of these layers (CovNorm) often outperformed the full matrix **A** (RA). In some datasets, like MITIndoor, with 4.8% of the parameters, CovNorm matched the performance of RA, Finally, as previously reported by [17], FNFT frequently underperformed RA. This is likely due to overfitting.

2.4.3 CovNorm vs SVD

Figure 2.7 provides empirical evidence for the vastly different quality of the approximations produced by CovNorm and the SVD. The figure shows a plot of the variance explained by the eigenvalues of the input and output distributions of an adaptation layer **A** and the corre-

sponding plot for its singular values. Note how the PCA energy is packed into a much smaller number of coefficients than the singular value energy. This happens because PCA only accounts for the subspaces populated by data, restricting the low-rank approximation to these subspaces. Conversely, the geometric approximation must approximate the matrix behavior even outside of these subspaces. Note that the SVD is not only less efficient in identifying the important dimensions, but also makes it difficult to determine how many singular values to keep. This prevents the use of a layer-dependent number of singular values.

2.4.4 Comparison to Previous Methods

Table 2.1 summarizes the recognition accuracy and % of adaptation layer parameters vs. VGG model size (100% parameters), for various methods. All abbreviations are as above. Beyond MDL, we compare to learning without forgetting (LwF) [48] a lifelong method to learn a model that shares all parameters among datasets. The table is split into independent and joint MDL. For joint learning, CovNorm is implemented with (2.10) and compared to the SVD approach of [19].

Several observations can be made. First, CovNorm adapts the number of parameters to the task, according to its complexity and how different it is from the source (ImageNet). For the simplest datasets, such as the 10-digit class SVHN, adaptation can require as few as 0.13% task-specific parameters. Datasets that are more diverse but ImageNet-like, such as Caltech256, require around 0.46% parameters. Finally, larger adaptation layers are required by datasets that are either complex or quite different from ImageNet, e.g. scene (MITIndoor, SUN397) recognition tasks. Even here, adaptation requires less than 1% parameters. On average, CovNorm requires 0.53% additional parameters per dataset.

Second, for independent learning, all methods based on residual adapters significantly outperform BN and LwF. As shown by [17], RA outperforms FNFT. BN is uniformly weak, LwF performs very well on MITIndoor and Caltech256, but poorly on most other datasets. Third, CovNorm outperforms even RA, achieving higher recognition accuracy with 20$\times$ less parameters.

Table 2.1: Classification accuracy and % of adaptation parameters (with respect to VGG size) per target dataset.

	FGVC	MITIndoor	Flowers	Caltech256	SVHN	SUN397	CIFAR100	average
FNFT	85.73%	71.77%	95.67%	83.73%	96.41%	57.29%	80.45%	81.58%
	100%							100%
Independent learning								
BN [20]	43.6%	57.6%	83.07%	73.66%	91.1%	47.04%	64.8%	65.83%
	0%							0%
LwF[48]	66.25%	**73.43%**	89.12%	80.02%	44.13%	52.85%	72.94%	68.39%
	0%							0%
RA [17]	88.92%	72.4%	96.43%	84.17%	96.13%	57.38%	**79.55%**	82.16%
	10%							10%
SVD+FTA	89.07%	71.66%	95.67%	84.46%	96.04%	57.12%	78.28%	81.75%
	5%							5%
FTA	87.31%	70.26%	95.43%	83.82%	95.96%	56.43%	78.23%	81.06%
	5%							5%
CovNorm	**88.98%**	72.51%	**96.76%**	**84.75%**	**96.23%**	**57.97%**	79.42%	**82.37%**
	0.34%	0.62%	0.35%	0.46%	0.13%	0.71%	1.1%	0.53%
Joint learning								
SVD [19]	88.98%	71.7%	96.37%	83.63%	96%	56.58%	78.26%	81.65%
	5%							5%
CovNorm	**88.99%**	**73.0%**	**96.69%**	**84.77%**	**96.22%**	**58.2**	**79.22%**	**82.44%**
	0.51%							0.51%

It also outperforms SVD+FTA and FTA by $\approx 0.6\%$ and $\approx 1.3\%$, respectively, while reducing parameter sizes by a factor of ≈ 10. On a per-dataset basis, CovNorm outperforms RA on all datasets other than CIFAR100, and SVD+FTA and FTA on all of them. In all datasets, the parameter savings are significant. Fourth, for joint training, CovNorm is substantially superior to the SVD [19], with higher recognition rates in all datasets, gains of up to 1.62% (SUN397), and close to $10\times$ less parameters. Finally, comparing independent and joint CovNorm, the latter has slightly higher recognition for a slightly higher parameter count. Hence, the two approaches are roughly equivalent.

2.4.5 Results on Visual Decathlon

Final, we apply the CovNorm on the Decathlon challenge [17], composed of ten different datasets- ImageNet, Aircraft, Cifar100, Daimler Pedestrians, Describable Textures, German

Traffic Signs, VGG-Flowers, OmniGlot, SVHN and UCF101 Dynamic Images. The resolution for images from all the datasets is 72×72. Following [17], ResNet-26 [55] is adopted as backbone and is trained with a combination of training and validation set. The results are based on the test set and obtained online. Table 2.2 presents the result for each dataset and CovNorm is evaluated in terms of classification accuracy, parameter size and decatholon score **S**. It clearly shows that CovNorm achieves state-of-the-art performance from the view of accuracy and parameter efficiency.

2.5 Conclusion

In this chapter, we proposed CovNorm, which is an multi-domain learning technique of very simple implementation. When compared to previous methods, it dramatically reduces the number of adaptation parameters without loss of recognition performance. It was used to show that large CNNs can be "recycled" across problems as diverse as digit, object, scene, or fine-grained classes, with no loss, by simply tuning 0.5% of their parameters. Except the good performance on multi-domain learning task, its domain-wise dynamic architecture also sheds light on the power of dynamic network for complex vision tasks. It reveals that the representation capacity of the network can be significantly boosted even though the dynamic module is very tiny, which will be further discussed in the following chapters on large scale image recognition tasks.

Chapter 2 is, in full, based on the material as it appears in the publication of "Efficient Multi-Domain Learning by Covariance Normalization", Yunsheng Li and Nuno Vasconcelos, in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 2019. The dissertation author was the primary investigator and author of this material.

Table 2.2: Visual Decathlon results. The results are obtained on the test set of the Decathlon dataset. The proposed CovNorm achieves the best accuracy compared to other methods with a lighter framework.

	ImNet	Airc	C100	DPed	DTD	GTSR	Flwr	OGlt	SVHN	UCF	avg acc	S	#par
RA [17]	59.67%	61.87%	81.20%	93.88%	57.13%	97.57%	81.67%	89.62%	96.13%	**50.12%**	76.89%	2621	2
DAN [64]	57.74%	64.12%	80.07%	91.3%	56.54%	98.46%	**86.05%**	**89.67%**	96.77%	49.38%	77.01%	2851	2.17
Piggyback [65]	57.69%	65.29%	79.87%	96.99%	57.45%	97.27%	79.09%	87.63%	**97.24%**	47.48%	76.6%	2838	1.28
CovNorm	**60.37%**	**69.37%**	**81.34%**	**98.75%**	**59.95%**	**99.14%**	83.44%	87.69%	96.55%	48.92%	**78.55%**	**3713**	1.25

Chapter 3

Dynamic Convolution Decomposition for Efficient Large Scale Image Recognition

3.1 Introduction

CovNorm has been shown to be effective in terms of both accuracy and efficiency for multi-domain learning task, due to its architecture design and more critical its dynamic property. In this chapter, we will further explore this property by extending the task domain from multi-domain learning to large scale image recognition and change the dynamic module from the domain-wise manner to sample-wise manner.

Actually, the design of dynamic network has be studied recently and dynamic convolution [1, 2] is one of the most successful dynamic framework for the implementation of light-weight networks [66, 67]. Its ability to achieve significant performance gains with negligible computational cost has motivated its adoption for multiple vision tasks [68, 69, 70, 71]. The basic idea is to aggregate multiple convolution kernels dynamically, according to an input dependent attention mechanism, into a convolution weight matrix

$$\boldsymbol{W}(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k(\boldsymbol{x}) \boldsymbol{W}_k \quad \text{s.t.} \quad 0 \leq \pi_k(\boldsymbol{x}) \leq 1, \sum_{k=1}^{K} \pi_k(\boldsymbol{x}) = 1, \tag{3.1}$$

where K convolution kernels $\{\boldsymbol{W}_k\}$ are aggregated linearly with attention scores $\{\pi_k(\boldsymbol{x})\}$.

Dynamic convolution has two main limitations: (a) lack of compactness, due to the use of K kernels, and (b) a challenging joint optimization of attention scores $\{\pi_k(\boldsymbol{x})\}$ and static kernels $\{\boldsymbol{W}_k\}$. [1] proposed the use of a sigmoid layer to generate attention scores $\{\pi_k(\boldsymbol{x})\}$, leading to a significantly large space for the convolution kernel $\boldsymbol{W}(\boldsymbol{x})$ that makes the learning of attention scores $\{\pi_k(\boldsymbol{x})\}$ difficult. [2] replaced the sigmoid layer with a softmax function to compress the kernel space. However, small attention scores π_k output by the softmax make the corresponding kernels $\boldsymbol{W}_k$ difficult to learn, especially in early training epochs, slowing training convergence. To mitigate these limitations, these two methods require additional constraints. For instance, [2] uses a large temperature in the softmax function to encourage near-uniform attention.

In this chapter, inspired by CovNorm, we revisit the two limitations via matrix decomposi-

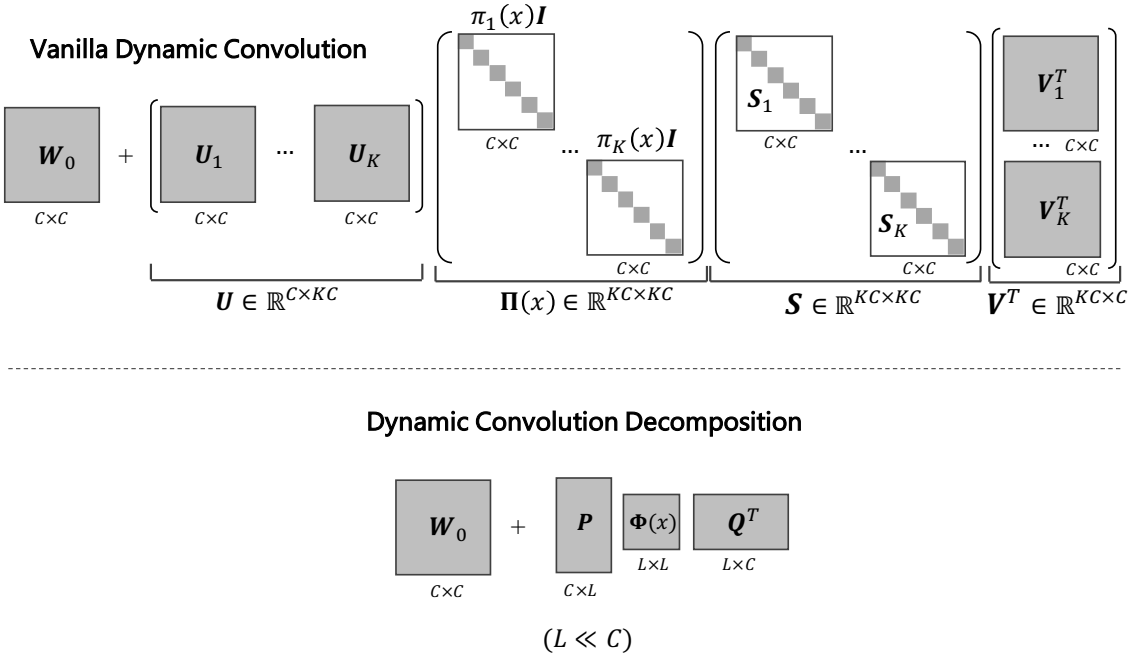


Figure 3.1: Dynamic convolution via matrix decomposition. **Top**: the vanilla dynamic convolution. It applies ***dynamic attention*** $\boldsymbol{\Pi}(\boldsymbol{x})$ in a ***high dimensional space***. **Bottom**: proposed dynamic convolution decomposition, which applies ***dynamic channel fusion*** $\boldsymbol{\Phi}(\boldsymbol{x})$ in a ***low dimensional space***.

tion. To expose the limitations, we reformulate dynamic convolution in terms of a set of residuals, re-defining the static kernels as

$$\boldsymbol{W}_k = \boldsymbol{W}_0 + \Delta\boldsymbol{W}_k, \quad k \in \{1,\ldots,K\} \tag{3.2}$$

where $\boldsymbol{W}_0 = \frac{1}{K}\sum_{k=1}^{K}\boldsymbol{W}_k$ is the average kernel and $\Delta\boldsymbol{W}_k = \boldsymbol{W}_k - \boldsymbol{W}_0$ a residual weight matrix. Further decomposing the latter with an SVD, $\Delta\boldsymbol{W}_k = \boldsymbol{U}_k\boldsymbol{S}_k\boldsymbol{V}_k^T$, leads to

$$\boldsymbol{W}(\boldsymbol{x}) = \sum_{k=1}^{K}\pi_k(\boldsymbol{x})\boldsymbol{W}_0 + \sum_{k=1}^{K}\pi_k(\boldsymbol{x})\boldsymbol{U}_k\boldsymbol{S}_k\boldsymbol{V}_k^T = \boldsymbol{W}_0 + \boldsymbol{U}\boldsymbol{\Pi}(\boldsymbol{x})\boldsymbol{S}\boldsymbol{V}^T, \tag{3.3}$$

where $\boldsymbol{U} = [\boldsymbol{U}_1,\ldots,\boldsymbol{U}_K]$, $\boldsymbol{S} = diag(\boldsymbol{S}_1,\ldots,\boldsymbol{S}_K)$, $\boldsymbol{V} = [\boldsymbol{V}_1,\ldots,\boldsymbol{V}_K]$, and $\boldsymbol{\Pi}(\boldsymbol{x})$ stacks attention scores diagonally as $\boldsymbol{\Pi}(\boldsymbol{x}) = diag(\pi_1(\boldsymbol{x})\boldsymbol{I},\ldots,\pi_K(\boldsymbol{x})\boldsymbol{I})$, where $\boldsymbol{I}$ is an identity matrix. This decomposition, illustrated in Figure 3.1-(Top), shows that the dynamic behavior of $\boldsymbol{W}(\boldsymbol{x})$ is implemented

by the dynamic residual $\boldsymbol{U}\boldsymbol{\Pi}(\boldsymbol{x})\boldsymbol{S}\boldsymbol{V}^T$, which projects the input $\boldsymbol{x}$ to a higher dimensional space $\boldsymbol{S}\boldsymbol{V}^T\boldsymbol{x}$ (from C to KC channels), applies dynamic attention $\boldsymbol{\Pi}(\boldsymbol{x})$ over channel groups, and reduces the dimension back to C channels, through multiplication by $\boldsymbol{U}$. This suggests that the limitations of vanilla dynamic convolution are due to the use of *attention over channel groups,* which induces a high dimensional latent space, leading to small attention values that may suppress the learning of the corresponding channels.

To address this issue, we propose a *dynamic convolution decomposition* (DCD), that replaces dynamic attention over channel groups with *dynamic channel fusion*. The latter is based on a full dynamic matrix $\boldsymbol{\Phi}(\boldsymbol{x})$, of which each element $\phi_{i,j}(\boldsymbol{x})$ is a function of input $\boldsymbol{x}$. As shown in Figure 3.1-(bottom), the dynamic residual is implemented as the product $\boldsymbol{P}\boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{Q}^T$ of $\boldsymbol{\Phi}(\boldsymbol{x})$ and two static matrices $\boldsymbol{P}, \boldsymbol{Q}$, such that $\boldsymbol{Q}$ compresses the input into a low dimensional latent space, $\boldsymbol{\Phi}(\boldsymbol{x})$ dynamically fuses the channels in this space, and $\boldsymbol{P}$ expands the number of channels to the output space. The key innovation is that dynamic channel fusion with $\boldsymbol{\Phi}(\boldsymbol{x})$ enables a significant dimensionality reduction of the latent space ($\boldsymbol{Q}^T\boldsymbol{x} \in \mathbb{R}^L$, $L \ll C$). Hence the number of parameters in $\boldsymbol{P}, \boldsymbol{Q}$ is significantly reduced, when compared to $\boldsymbol{U}, \boldsymbol{V}$ of Eq. 3.3, resulting in a more compact model. Dynamic channel fusion also mitigates the joint optimization challenge of vanilla dynamic convolution, as each column of $\boldsymbol{P}, \boldsymbol{Q}$ is associated with multiple dynamic coefficients of $\boldsymbol{\Phi}(\boldsymbol{x})$. Hence, a few dynamic coefficients of small value are not sufficient to suppress the learning of static matrices $\boldsymbol{P}, \boldsymbol{Q}$. Experimental results show that DCD both significantly reduces the number of parameters and achieves higher accuracy than vanilla dynamic convolution, without requiring the additional constraints of [1, 2].

3.2 Related Work

3.2.1 Efficient CNNs

MobileNet [66, 72, 73] decomposes $k \times k$ convolution into a depthwise and a pointwise convolution. ShuffleNet [67, 74] uses group convolution and channel shuffle to further simplify pointwise convolution. Further improvements of these architectures have been investigated recently. EfficientNet [75, 76] finds a proper relationship between input resolution and width/depth of the network. [77] mix up multiple kernel sizes in a single convolution. [78] trades massive multiplications for much cheaper additions. [79] applies a series of cheap linear transformations to generate ghost feature maps. [80] flips the structure of inverted residual blocks to alleviate information loss. [81] and [82] train one network that supports multiple sub-networks of different complexities.

3.2.2 Matrix Decomposition

[83] and [84] use Canonical Polyadic decomposition (CPD) of convolution kernels to speed up networks, while [14] investigates Tucker decompositions for the same purpose. More recently, [85] combines tensor decompositions with MobileNet to design efficient higher-order networks for video tasks, while [86] proposes a stable CPD to deal with degeneracies of tensor decompositions during network training. Unlike DCD, which decomposes a convolutional kernel dynamically by adapting the core matrix to the input, these works all rely on static decompositions.

3.2.3 Dynamic Neural Networks

Dynamic networks boost representation power by adapting parameters or activation functions to the input. [87] uses a secondary network to generate parameters for the main network. [88] reweights channels by squeezing global context. [89] adapts attention over kernels of

different sizes. Dynamic convolution [1, 2] aggregates multiple convolution kernels based on attention. [70] uses grouped fully connected layer to generate convolutional weights directly. [69] extends dynamic convolution from spatial agnostic to spatial specific. [68] proposes dynamic group convolution that adaptively selects input channels to form groups. [71] applies dynamic convolution to instance segmentation. [90] adapts slopes and intercepts of two linear functions in ReLU [91, 92].

3.3 Dynamic Convolution Decomposition

In this section, we introduce the *dynamic convolution decomposition* proposed to address the limitations of vanilla dynamic convolution. For conciseness, we assume a kernel $\boldsymbol{W}$ with the same number of input and output channels ($C_{in} = C_{out} = C$) and ignore bias terms. We focus on 1×1 convolution in this section and generalize the procedure to $k \times k$ convolution in the following section.

3.3.1 Revisiting Vanilla Dynamic Convolution

Vanilla dynamic convolution aggregates K convolution kennels $\{\boldsymbol{W}_k\}$ with attention scores $\{\pi_k(\boldsymbol{x})\}$ (see Eq. 3.1). It can be reformulated as adding a dynamic residual to a static kernel, and the dynamic residual can be further decomposed by SVD (see Eq. 3.3), as shown in Figure 3.1-(Top). This has two limitations. First, the model is not compact. Essentially, *it expands the number of channels by a factor of K and applies dynamic attention over K channel groups.* The dynamic residual $\boldsymbol{U}\boldsymbol{\Pi}(\boldsymbol{x})\boldsymbol{S}\boldsymbol{V}^T$ is a $C \times C$ matrix, of maximum rank C, but sums KC rank-1 matrices, since

$$\boldsymbol{W}(\boldsymbol{x}) = \boldsymbol{W}_0 + \boldsymbol{U}\boldsymbol{\Pi}(\boldsymbol{x})\boldsymbol{S}\boldsymbol{V}^T = \boldsymbol{W}_0 + \sum_{i=1}^{KC} \pi_{\lceil i/C \rceil}(\boldsymbol{x})\boldsymbol{u}_i s_{i,i} \boldsymbol{v}_i^T, \tag{3.4}$$

where $\boldsymbol{u}_i$ is the i^{th} column vector of matrix $\boldsymbol{U}$, $\boldsymbol{v}_i$ is the i^{th} column vector of matrix $\boldsymbol{V}$, $s_{i,i}$ is the i^{th} diagonal entry of matrix $\boldsymbol{S}$ and $\lceil\cdot\rceil$ is ceiling operator. The static basis vectors $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ are not shared across different rank-1 matrices ($\pi_{\lceil i/C\rceil}(\boldsymbol{x})\boldsymbol{u}_i s_{i,i}\boldsymbol{v}_i^T$). This results in model redundancy. Second, it is difficult to jointly optimize static matrices $\boldsymbol{U}$, $\boldsymbol{V}$ and dynamic attention $\boldsymbol{\Pi}(\boldsymbol{x})$. This is because a small attention score $\pi_{\lceil i/C\rceil}$ may suppress the learning of corresponding columns $\boldsymbol{u}_i$, $\boldsymbol{v}_i$ in $\boldsymbol{U}$ and $\boldsymbol{V}$, especially in early training epochs (as shown in [2]).

3.3.2 Dynamic Channel Fusion

We propose to address the limitations of the vanilla dynamic convolution with a dynamic channel fusion mechanism, implemented with a full matrix $\boldsymbol{\Phi}(\boldsymbol{x})$, where each element $\phi_{i,j}(\boldsymbol{x})$ is a function of input $\boldsymbol{x}$. $\boldsymbol{\Phi}(\boldsymbol{x})$ is a $L\times L$ matrix, dynamically fusing channels in the latent space $\mathbb{R}^L$. The key idea is to significantly reduce dimensionality in the latent space, $L \ll C$, to enable a more compact model. Dynamic convolution is implemented with dynamic channel fusion using

$$\boldsymbol{W}(\boldsymbol{x}) = \boldsymbol{W}_0 + \boldsymbol{P}\boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{Q}^T = \boldsymbol{W}_0 + \sum_{i=1}^{L}\sum_{j=1}^{L}\boldsymbol{p}_i\phi_{i,j}(\boldsymbol{x})\boldsymbol{q}_j^T, \tag{3.5}$$

where $\boldsymbol{Q}\in\mathbb{R}^{C\times L}$ compresses the input into a low dimensional space ($\boldsymbol{Q}^T\boldsymbol{x}\in\mathbb{R}^L$), the resulting L channels are fused dynamically by $\boldsymbol{\Phi}(\boldsymbol{x})\in\mathbb{R}^{L\times L}$ and expanded to the number of output channels by $\boldsymbol{P}\in\mathbb{R}^{C\times L}$. This is denoted as **dynamic convolution decomposition** (DCD). The dimension L of the latent space is constrained by $L^2 < C$. The default value of L in this chapter is empirically set to $\lfloor\frac{C}{2^{\lfloor\log_2\sqrt{C}\rfloor}}\rfloor$, which means dividing C by 2 repeatedly until it is less than $\sqrt{C}$.

With this new design, the number of static parameters is significantly reduced (i.e., LC parameters in $\boldsymbol{P}$ or $\boldsymbol{Q}$ *v.s.* KC^2 parameters in $\boldsymbol{U}$ or $\boldsymbol{V}$, $L<\sqrt{C}$), resulting in a more compact model. Mathematically, the dynamic residual $\boldsymbol{P}\boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{Q}^T$ sums L^2 rank-1 matrices $\boldsymbol{p}_i\phi_{i,j}(\boldsymbol{x})\boldsymbol{q}_j^T$, where $\boldsymbol{p}_i$ is the i^{th} column vector of $\boldsymbol{P}$, and $\boldsymbol{q}_j$ is the j^{th} column vector of $\boldsymbol{Q}$. The constraint $L^2<C$, guarantees that this number (L^2) is much smaller than the counterpart (KC) of vanilla

dynamic convolution (see Eq. 3.4). Nevertheless, due to the use of a full matrix, dynamic channel fusion $\boldsymbol{\Phi}(\boldsymbol{x})$ retains the representation power needed to achieve good classification performance.

DCD also mitigates the joint optimization difficulty. Since each column of $\boldsymbol{P}$ (or $\boldsymbol{Q}$) is associated with multiple dynamic coefficients (e.g. $\boldsymbol{p}_i$ is related to $\phi_{i,1},\ldots,\phi_{i,L}$), it is unlikely that the learning of $\boldsymbol{p}_i$ is suppressed by a few dynamic coefficients of small value.

In summary, DCD performs dynamic aggregation differently from vanilla dynamic convolution. Vanilla dynamic convolution uses a *shared dynamic attention* mechanism to aggregate *unshared static basis vectors* in a *high* dimensional latent space. In contrast, DCD uses an *unshared dynamic channel fusion* mechanism to aggregate *shared static basis vectors* in a *low* dimensional latent space.

3.3.3 More General Formulation

So far, we have focused on the dynamic residual and shown that dynamic channel fusion enables a compact implementation of dynamic convolution. We next discuss the static kernel $\boldsymbol{W}_0$. Originally, it is multiplied by a dynamic scalar $\sum_k \pi_k(\boldsymbol{x})$, which is canceled in Eq. 3.3 as attention scores sum to one. Relaxing the constraint $\sum_k \pi_k(\boldsymbol{x}) = 1$ results in the more general form

$$\boldsymbol{W}(\boldsymbol{x}) = \boldsymbol{\Lambda}(\boldsymbol{x})\boldsymbol{W}_0 + \boldsymbol{P}\boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{Q}^T, \tag{3.6}$$

where $\boldsymbol{\Lambda}(\boldsymbol{x})$ is a $C \times C$ diagonal matrix and $\lambda_{i,i}(\boldsymbol{x})$ a function of $\boldsymbol{x}$. In this way, $\boldsymbol{\Lambda}(\boldsymbol{x})$ implements channel-wise attention after the static kernel $\boldsymbol{W}_0$, generalizing Eq. 3.5 where $\boldsymbol{\Lambda}(\boldsymbol{x})$ is an identity matrix. Later, we will see that this generalization enables additional performance gains.

Relation to Squeeze-and-Excitation (SE) [88]: The dynamic channel-wise attention mechanism implemented by $\boldsymbol{\Lambda}(\boldsymbol{x})$ is related to but *different* from SE. It is parallel to a convolution and shares the input with the convolution. It can be thought of as either a *dynamic convolution kernel* $\boldsymbol{y} = (\boldsymbol{\Lambda}(\boldsymbol{x})\boldsymbol{W}_0)\boldsymbol{x}$ or an input-dependent attention mechanism applied to the *output feature map*

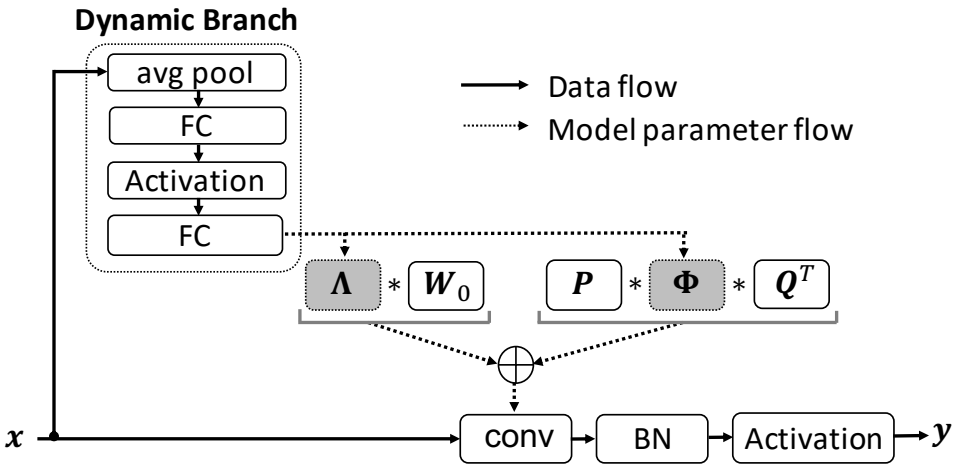


Figure 3.2: **Dynamic convolution decomposition layer**. The input $\boldsymbol{x}$ first goes through a dynamic branch to generate $\boldsymbol{\Lambda}(\boldsymbol{x})$ and $\boldsymbol{\Phi}(\boldsymbol{x})$, and then to generate the convolution matrix $\boldsymbol{W}(\boldsymbol{x})$ using Eq. 3.6.

of the convolution $\boldsymbol{y} = \boldsymbol{\Lambda}(\boldsymbol{x})(\boldsymbol{W}_0\boldsymbol{x})$. Thus, its computational complexity is $\min(O(C^2), O(HWC))$, where H and W are height and width of the feature map.

In contrast, SE is placed *after* a convolution and uses the output of the convolution as input. It can only apply channel attention on the *output feature map* of the convolution as $\boldsymbol{y} = \boldsymbol{\Lambda}(\boldsymbol{z})\boldsymbol{z}$, where $\boldsymbol{z} = \boldsymbol{W}_0\boldsymbol{x}$. Its computational complexity is $O(HWC)$. Clearly, SE requires more computation than dynamic channel-wise attention $\boldsymbol{\Lambda}(\boldsymbol{x})$ when the resolution of the feature map ($H \times W$) is high.

3.3.4 Dynamic Convolution Decomposition Layer

Implementation: Figure 3.2 shows the diagram of a dynamic convolution decomposition (DCD) layer. It uses a light-weight dynamic branch to generate coefficients for both dynamic channel-wise attention $\boldsymbol{\Lambda}(\boldsymbol{x})$ and dynamic channel fusion $\boldsymbol{\Phi}(\boldsymbol{x})$. Similar to Squeeze-and-Excitation [88], the dynamic branch first applies average pooling to the input $\boldsymbol{x}$. This is followed by two fully connected (FC) layers with an activation layer between them. The first FC layer reduces the number of channels by r and the second expands them into $C + L^2$ outputs (C for $\boldsymbol{\Lambda}$ and L^2 for $\boldsymbol{\Phi}$).

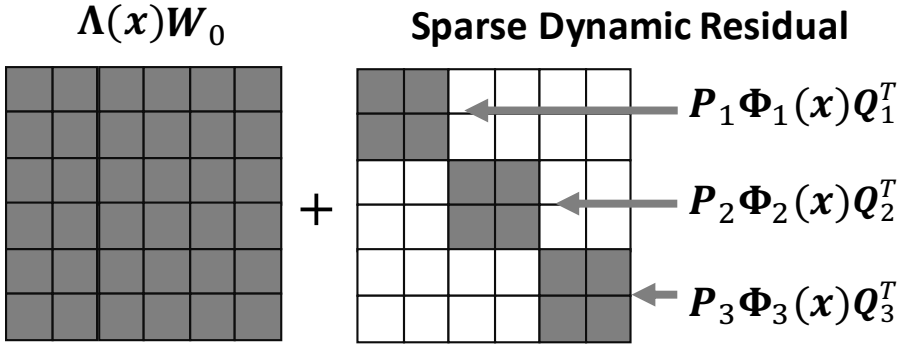


Figure 3.3: **Sparse dynamic residual**, which is represented as a diagonal block matrix. Each diagonal block is decomposed separately as $\boldsymbol{P}_b\boldsymbol{\Phi}_b\boldsymbol{Q}_b^T$. Note that the static kernel $\boldsymbol{W}_0$ is still a full size matrix.

Eq. 3.6 is finally used to generate convolutional weights $\tilde{\boldsymbol{W}}(\boldsymbol{x})$. Similarly to a static convolution, a DCD layer also includes a batch normalization and an activation (e.g. ReLU) layer.

Parameter Complexity: DCD has similar FLOPs to the vanilla dynamic convolution. Here, we focus on parameter complexity. Static convolution and vanilla dynamic convolution require C^2 and KC^2 parameters, respectively. DCD requires C^2, CL, and CL parameters for static matrices $\boldsymbol{W}_0$, $\boldsymbol{P}$ and $\boldsymbol{Q}$, respectively. An additional $(2C+L^2)\frac{C}{r}$ parameters are required by the dynamic branch to generate $\boldsymbol{\Lambda}(\boldsymbol{x})$ and $\boldsymbol{\Phi}(\boldsymbol{x})$, where r is the reduction rate of the first FC layer. The total complexity is $C^2+2CL+(2C+L^2)\frac{C}{r}$. Since L is constrained as $L^2<C$, the complexity upper bound is $(1+\frac{3}{r})C^2+2C\sqrt{C}$. When choosing $r=16$, the complexity is about $1\frac{3}{16}C^2$. This is much less than what is typical for vanilla dynamic convolution ($4C^2$ in [2] and $8C^2$ in [1]).

3.4 Extensions of Dynamic Convolution Decomposition

In this section, we extend the dynamic decomposition of 1×1 convolution (Eq. 3.6) in three ways: (a) sparse dynamic residual where $\boldsymbol{P}\boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{Q}^T$ is a diagonal block matrix, (b) $k\times k$ depthwise convolution, and (c) $k\times k$ convolution. Here, k refers to the kernel size.

3.4.1 DCD with Sparse Dynamic Residual

The dynamic residual $\boldsymbol{P}\boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{Q}^T$ can be further simplified into a block-diagonal matrix of blocks $\boldsymbol{P}_b\boldsymbol{\Phi}_b(\boldsymbol{x})\boldsymbol{Q}_b^T, b \in \{1,\ldots,B\}$, leading to

$$\boldsymbol{W}(\boldsymbol{x}) = \boldsymbol{\Lambda}(\boldsymbol{x})\boldsymbol{W}_0 + \bigoplus_{b=1}^{B} \boldsymbol{P}_b\boldsymbol{\Phi}_b(\boldsymbol{x})\boldsymbol{Q}_b^T, \tag{3.7}$$

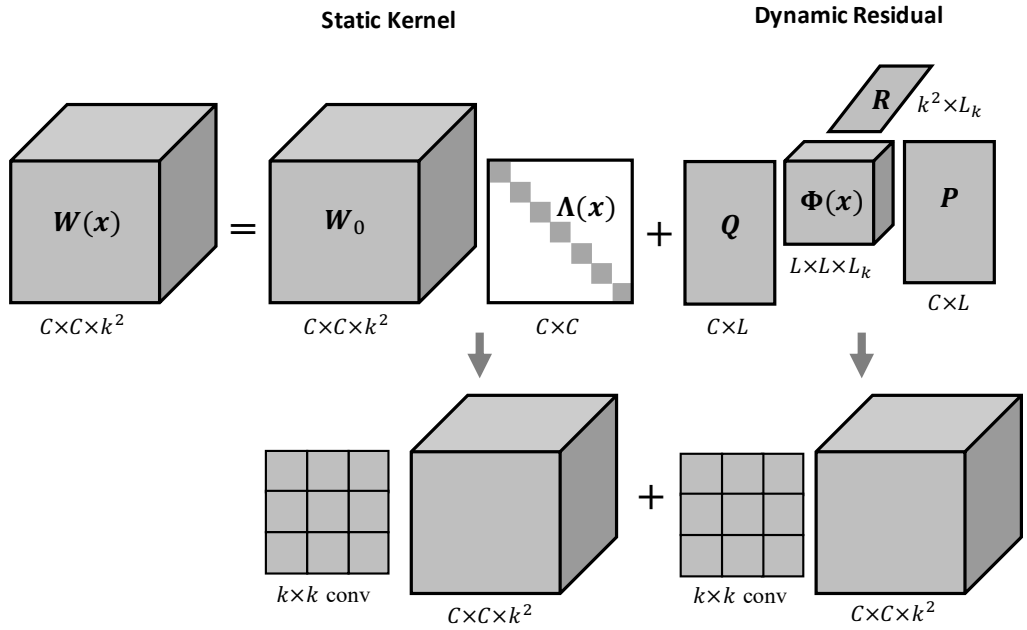
where $\bigoplus_{i=1}^{n} A_i = diag(A_1,\ldots,A_n)$. This form has Eq. 3.6 as a special case, where $B = 1$. Note that the static kernel $\boldsymbol{W}_0$ is still a full matrix and only the dynamic residual is sparse (see Figure 3.3). We will show later that keeping as few as $\frac{1}{8}$ of the entries of the dynamic residual non-zero ($B = 8$) has a minimal performance degradation, still significantly outperforming a static kernel.

3.4.2 DCD of $k \times k$ Depthwise Convolution

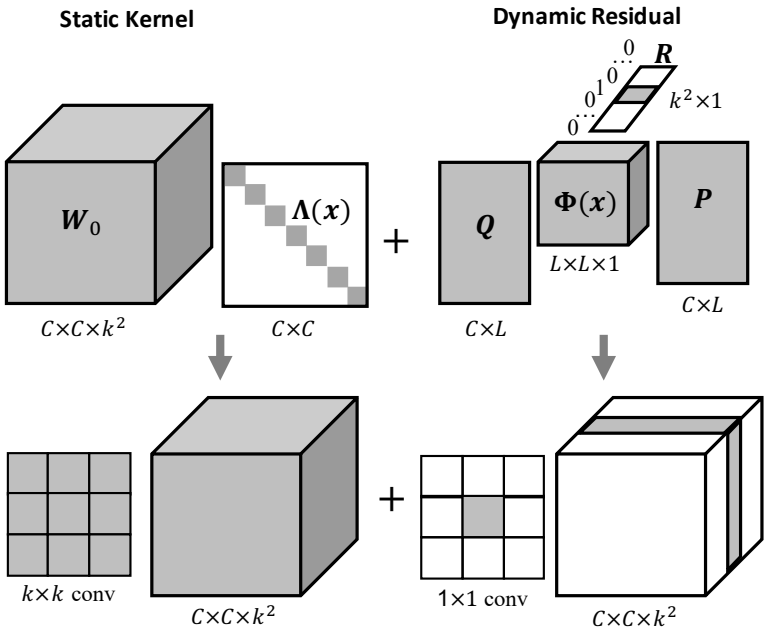
The weights of a $k \times k$ depthwise convolution kernel form a $C \times k^2$ matrix. DCD can be generalized to such matrices by replacing in Eq. 3.6 the matrix $\boldsymbol{Q}$ (which squeezes the number of channels) with a matrix $\boldsymbol{R}$ (which squeezes the number of kernel elements)

$$\boldsymbol{W}(\boldsymbol{x}) = \boldsymbol{\Lambda}(\boldsymbol{x})\boldsymbol{W}_0 + \boldsymbol{P}\boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{R}^T, \tag{3.8}$$

where $\boldsymbol{W}(\boldsymbol{x})$ and $\boldsymbol{W_0}$ are $C \times k^2$ matrices, $\boldsymbol{\Lambda(x)}$ is a diagonal $C \times C$ matrix that implements channel-wise attention, $\boldsymbol{R}$ is a $k^2 \times L_k$ matrix that reduces the number of kernel elements from k^2 to L_k, $\boldsymbol{\Phi(x)}$ is a $L_k \times L_k$ matrix that performs dynamic fusion along L_k latent kernel elements and $\boldsymbol{P}$ is a $C \times L_k$ weight matrix for depthwise convolution over L_k kernel elements. The default value of L_k is $\lfloor k^2/2 \rfloor$. Since depthwise convolution is channel separable, $\boldsymbol{\Phi(x)}$ does not fuse channels, fusing instead L_k latent kernel elements.



(a) $\Phi(x)$ fuses both channels and kernel elements.



(b) $\Phi(x)$ fuses channels alone.

Figure 3.4: The dynamic convolution decomposition for $k \times k$ convolution.

3.4.3 DCD of $k \times k$ Convolution

Joint fusion of channels and kernel elements: A $k \times k$ convolution kernel forms a $C \times C \times k^2$ tensor. DCD can be generalized to such tensors by extending Eq. 3.6 into a tensor form (see Figure 3.4)

$$\boldsymbol{W}(\boldsymbol{x}) = \boldsymbol{W_0} \times_2 \boldsymbol{\Lambda}(\boldsymbol{x}) + \Phi(\boldsymbol{x}) \times_1 \boldsymbol{Q} \times_2 \boldsymbol{P} \times_3 \boldsymbol{R}, \tag{3.9}$$

where $\times_n$ refers to n-mode multiplication [93], $\boldsymbol{W_0}$ is a $C \times C \times k^2$ tensor, $\boldsymbol{\Lambda}(\boldsymbol{x})$ is a diagonal $C \times C$ matrix that implements channel-wise attention, $\boldsymbol{Q}$ is a $C \times L$ matrix that reduces the number of input channels from C to L, $\boldsymbol{R}$ is a $k^2 \times L_k$ matrix that reduces the number of kernel elements from k^2 to L_k, $\boldsymbol{\Phi}(\boldsymbol{x})$ is a $L \times L \times L_k$ tensor that performs joint fusion of L channels over L_k latent kernel elements, and $\boldsymbol{P}$ is a $C \times L$ matrix that expands the number of channels from L to C. The numbers of latent channels L and latent kernel elements L_k are constrained by $L_k < k^2$ and $L^2 L_k \leq C$. Their default values are set empirically to $L_k = \lfloor k^2/2 \rfloor$, $L = \lfloor \frac{C/L_k}{2^{\lfloor log_2 \sqrt{C/L_k} \rfloor}} \rfloor$.

Channel fusion alone: We found that the fusion of channels $\Phi(\boldsymbol{x}) \times_1 \boldsymbol{Q}$ is more important than the fusion of kernel elements $\Phi(\boldsymbol{x}) \times_3 \boldsymbol{R}$. Therefore, we reduce L_k to 1 and increase L accordingly. $\boldsymbol{R}$ is simplified into a one-hot vector $[0, \ldots, 0, 1, 0, \ldots, 0]^T$, where the '1' is located at the center (assuming that k is an odd number). As illustrated in Figure 3.4-(b), the tensor of dynamic residual $\Phi(\boldsymbol{x}) \times_1 \boldsymbol{Q} \times_2 \boldsymbol{P} \times_3 \boldsymbol{R}$ only has one non-zero slice, which is equivalent to a 1×1 convolution. Therefore, the DCD of a $k \times k$ convolution is essentially adding a 1×1 dynamic residual to a static $k \times k$ kernel.

3.5 Experiments

In this section, we present the results of DCD on ImageNet classification [3]. ImageNet has 1,000 classes with 1,281,167 training and $50,000$ validation images. We also report ablation

studies on different components of the approach.

3.5.1 Experimental Set-up

All experiments are based on two network architectures: ResNet [55] and MobileNetV2 [72]. DCD is implemented on all convolutional layers of ResNet and all 1×1 convolutional layers of MobileNetV2. The reduction ratio r is set to 16 for ResNet and MobileNetV2 $\times 1.0$, and to 8 for smaller models (MobileNetV2 $\times 0.5$ and $\times 0.35$). All models are trained by SGD with momentum 0.9. The batch size is 256 and remaining training parameters are as follows.

ResNet: The learning rate starts at 0.1 and is divided by 10 every 30 epochs. The model is trained with 100 epochs. Dropout [94] 0.1 is used only for ResNet-50.

MobileNetV2: The initial learning rate is 0.05 and decays to 0 in 300 epochs, according to a cosine function. Weight decay of 2e-5 and a dropout rate of 0.1 are also used. For MobileNetV2 $\times 1.0$, Mixup [95] and label smoothing are further added to avoid overfitting.

3.5.2 Inspecting Different DCD Formulations

Table 3.1 summarizes the influence of different components (e.g. dynamic channel fusion $\boldsymbol{\Phi}(\boldsymbol{x})$, dynamic channel-wise attention $\boldsymbol{\Lambda}(\boldsymbol{x})$) of DCD on MobileNet V2 $\times 0.5$ and ResNet-18 performance. The table shows that both dynamic components, $\boldsymbol{\Lambda}(\boldsymbol{x})$ and $\boldsymbol{\Phi}(\boldsymbol{x})$ of Eq. 3.6. enhance accuracy substantially (+2.8% and +3.8% for MobileNetV2 $\times 0.5$, +1.1% and +2.4% for ResNet-18), when compared to the static baseline. Using dynamic channel fusion only ($\boldsymbol{W}_0 + \boldsymbol{P}\boldsymbol{\Phi}\boldsymbol{Q}^T$) has slightly more parameters, FLOPs, and accuracy than using dynamic channel-wise attention only ($\boldsymbol{\Lambda}\boldsymbol{W}_0$). The combination of the two mechanisms provides additional improvement.

Table 3.1: **Different formulations** of dynamic convolution decomposition on ImageNet classification.

Model	Params	MAdds	Top-1
$\boldsymbol{W}_0$ (static)	2.0M	97.0M	65.4
$\boldsymbol{\Lambda W}_0$	2.4M	97.4M	68.2
$\boldsymbol{W}_0 + \boldsymbol{P\Phi Q}^T$	2.7M	104.4M	69.2
$\boldsymbol{\Lambda W}_0 + \boldsymbol{P\Phi Q}^T$	2.9M	104.6M	**69.8**

(a) MobileNet V2 $\times 0.5$

Model	Params	MAdds	Top-1
$\boldsymbol{W}_0$ (static)	11.1M	1.81G	70.4
$\boldsymbol{\Lambda W}_0$	11.7M	1.81G	71.5
$\boldsymbol{W}_0 + \boldsymbol{P\Phi Q}^T$	13.6M	1.83G	72.8
$\boldsymbol{\Lambda W}_0 + \boldsymbol{P\Phi Q}^T$	14.0M	1.83G	**73.1**

(b) ResNet-18

Table 3.2: **Dimension of the latent space** L evaluated on ImageNet classification (MobileNetV2 $\times 0.5$ is used).

Model	L	Params	MAdds	Top-1
static	-	2.0M	97.0M	65.4
DCD	$\times 0.25$	2.4M	99.8M	68.7
	$\times 0.50$	2.5M	101.3M	69.0
	$\times 0.75$	2.6M	102.9M	69.6
	$\times 1.0$	2.9M	104.6M	**69.8**

3.5.3 Ablations

A number of ablations were performed on MobileNet V2 $\times 0.5$ to analyze DCD performance in terms of two questions.

1. **How** does the dimension (L) of the latent space affect performance?

2. **How** do three DCD variants perform?

The default configuration is the general form of DCD (Eq. 3.6) with a full size dynamic residual ($B = 1$) for all pointwise convolution layers. The default latent space dimension is $L = \lfloor \frac{C}{2^{\lfloor log_2 \sqrt{C} \rfloor}} \rfloor$.

Latent Space Dimension L: The dynamic channel fusion matrix $\boldsymbol{\Phi}(\boldsymbol{x})$ has size $L \times L$. Thus, L controls both the representation and the parameter complexity of DCD. We adjust it by applying different multipliers to the default value of L. Table 3.2 shows the results of MobileNetV2 $\times 0.5$ for four multiplier values ranging from $\times 1.0$ to $\times 0.25$. As L decreases, fewer parameters are required and the performance degrades slowly. Even with a very low dimensional

latent space ($L \times 0.25$), DCD still outperforms the static baseline by 3.3% top-1 accuracy.

Number of Diagonal Blocks B in the Dynamic Residual: Table 3.3-(a) shows classification results for four values of B. The dynamic residual is a full matrix when $B = 1$, while only $\frac{1}{8}$ of its entries are non-zero for $B = 8$. Accuracy degrades slowly as the dynamic residual becomes sparser (increasing B). The largest performance drop happens when B is changed from 1 to 2, as half of the weight matrix $\boldsymbol{W}(\boldsymbol{x})$ becomes static. However, performance is still significantly better than that of the static baseline. The fact that even the sparsest $B = 8$ outperforms the static baseline by 2.9% (from 65.4% to 68.3%) demonstrates the representation power of the dynamic residual. In all cases, dynamic channel-wise attention $\boldsymbol{\Lambda}(\boldsymbol{x})$ enables additional performance gains.

Table 3.3: **Extensions of dynamic convolution decompostion (DCD)** evaluated on ImageNet classification (MobileNetV2 $\times 0.5$ is used).

Network	B	Params	MAdds	Top-1
$\boldsymbol{W}_0$ (static)	-	2.0M	97.0M	65.4
$\boldsymbol{W}_0 + \boldsymbol{P\Phi Q}^T$	1	2.7M	104.4M	**69.2**
	2	2.6M	101.0M	68.5
	4	2.5M	99.1M	68.4
	8	2.5M	98.5M	68.3
$\boldsymbol{\Lambda W}_0 + \boldsymbol{P\Phi Q}^T$	1	2.9M	104.6M	**69.8**
	2	2.8M	101.3M	68.9
	4	2.7M	99.4M	68.8
	8	2.7M	98.8M	68.5

(a) Number of diagonal blocks B in the dynamic residual.

DW	PW	CLS	Params	MAdds	Top-1
			2.0M	97.0M	65.4
✓			2.4M	97.5M	68.3
	✓		2.9M	104.6M	69.8
		✓	2.2M	97.2M	66.6
✓		✓	2.6M	97.7M	69.0
✓	✓		3.3M	105.1M	69.6
	✓	✓	3.1M	104.8M	**70.2**
✓	✓	✓	3.5M	105.3M	70.0

(b) DCD at different layers. DW, PW, and CLS indicate depthwise convolution, pointwise convolution and classifier respectively.

DCD at Different Layers: Table 3.3-(b) shows the results of implementing DCD for three different types of layers (a) DW: depthwise convolution (Eq. 3.8), (b) PW: pointwise convolution (Eq. 3.6), and (c) CLS: fully connected classifier, which is a special case of pointwise convolution (the input resolution is 1×1). Using DCD in any type of layer improves on the performance of the static baseline (+2.9% for depthwise convolution, +4.4% for pointwise convolution, and +1.2% for classifier). Combining DCD for both pointwise convolution and classifier achieves the best performance (+4.8%). We notice a performance drop (from 70.2% to 70.0%) when using DCD in all three types of layers. We believe this is due to overfitting, as it has higher training

Table 3.4: Comparing DCD with the vanilla dynamic convolution CondConv [1] and DY-Conv [2]. ✶indicates the dynamic model with the fewest parameters (static model is not included). CondConv contains $K = 8$ kernels and DY-Conv contains $K = 4$ kernels.

Width	Model	Params	MAdds	Top-1
×1.0	static	3.5M	300.0M	72.0
	DY-Conv	11.1M	312.9M	**75.2**
	CondConv	27.5M	329.0M	74.6
	DCD (ours)	✶5.5M	326.0M	**75.2**
×0.5	static	2.0M	97.0M	65.4
	DY-Conv	4.0M	101.4M	69.9
	CondConv	15.5M	113.0M	68.4
	DCD (ours)	✶3.1M	104.8M	**70.2**
×0.35	static	1.7M	59.2M	60.3
	DY-Conv	2.8M	62.0M	65.9
	DCD (ours)	✶2.3M	63.1M	**66.6**

(a) **MobileNetV2**.

Depth	Model	Params	MAdds	Top-1
ResNet-50	static	23.5M	3.8G	76.2
	DCD (ours)	30.7M	3.9G	**77.9**
ResNet-18	static	11.1M	1.81G	70.4
	DY-Conv	42.7M	1.85G	72.7
	DCD (ours)	✶14.0M	1.83G	**73.1**
ResNet-10	static	5.2M	0.89G	63.5
	DY-Conv	18.6M	0.91G	67.7
	DCD (ours)	✶6.5M	0.90G	**68.8**

(b) **ResNet**.

accuracy.

Extension to 3×3 Convolution: We use ResNet-18, which stacks 16 layers of 3×3 convolution, to study the 3×3 extension of DCD (see Section 3.4.3). Compared to the static baseline (70.4% top-1 accuracy), DCD with *joint fusion of channels and kernel elements* (Eq. 3.9) improves top-1 accuracy (71.3%) by 0.9%. The top-1 accuracy is further improved by 1.8% (73.1%), when using DCD with *channel fusion alone*, which transforms the dynamic residual as a 1×1 convolution matrix (see Figure 3.4-(b)). This demonstrates that dynamic fusion is more effective across channels than across kernel elements.

Summary: Based on the ablations above, DCD should be implemented with both dynamic channel fusion $\mathbf{\Phi}$ and dynamic channel-wise attention $\mathbf{\Lambda}$, the default latent space dimension L, and a full size residual $B = 1$. DCD is recommended for pointwise convolution and classifier layers in MobileNetV2. For 3×3 convolutions in ResNet, DCD should be implemented with channel fusion alone. The model can be made more compact, for a slight performance drop, by (a) removing dynamic channel-wise attention $\mathbf{\Lambda}$, (b) reducing the latent space dimension L, (c) using a sparser dynamic residual (increasing B), and (d) implementing DCD in depthwise convolution alone.

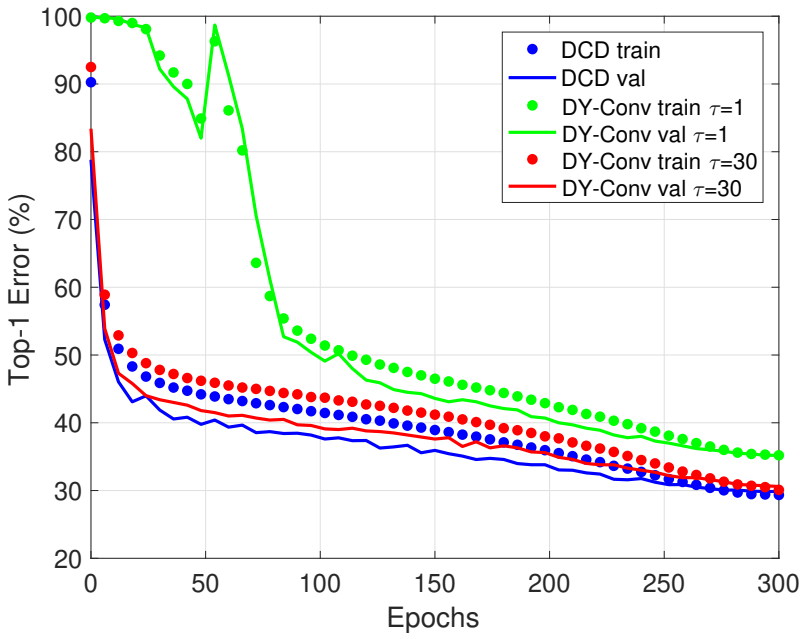


Figure 3.5: The comparison of training and validation error between DCD and DY-Conv on MobileNetV2 $\times 0.5$. τ is the temperature in softmax. Best viewed in color.

3.5.4 Main Results

DCD was compared to the vanilla dynamic convolution [1, 2] for MobileNetV2 and ResNet, using the settings recommended above, with the results of Table 3.4[1]. DCD significantly reduces the number of parameters while improving the performance of both network architectures. For MobileNetV2 $\times 1.0$, DCD only requires 50% of the parameters of [2] and 25% of the parameters of [1]. For ResNet-18, it only requires 33% of the parameters of [2], while achieving a 0.4% gain in top-1 accuracy. Although DCD requires slightly more MAdds than [2], the increment is negligible. These results demonstate that DCD is more compact and effective.

Figure 3.5 compares DCD to DY-Conv [2] in terms of training convergence. DY-Conv uses a large temperature in its softmax to alleviate the joint optimization difficulty and make training more efficient. Without any additional parameter tuning, DCD converges even faster than DY-Conv with a large temperature and achieves higher accuracy.

[1]The baseline results are from the original papers. Our implementation, under the setup used for DCD, has either similar or slightly lower results, e.g. for MobileNetV2$\times$1.0 the original paper reports 72.0%, while our implementation achieves 71.8%.

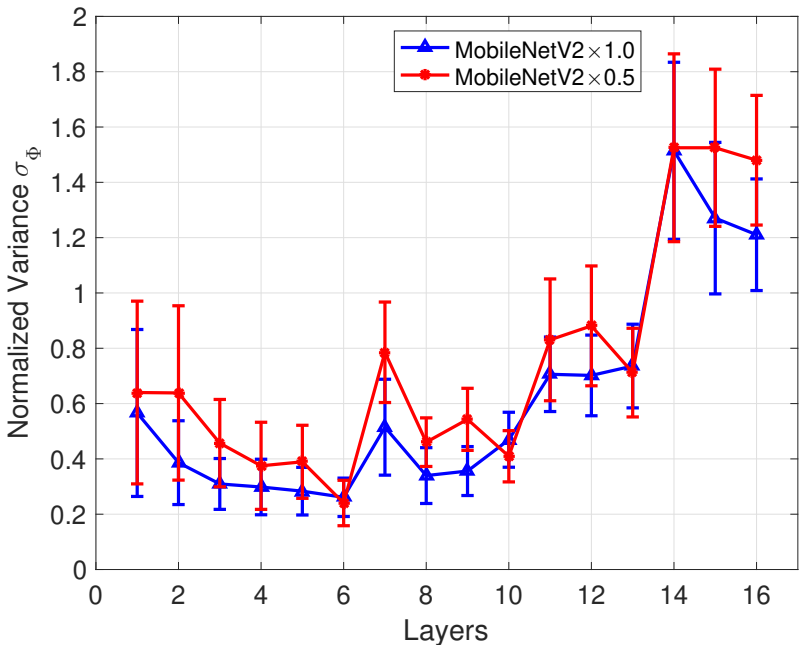


Figure 3.6: Normalized variance of dynamic coefficients σ_Φ across layers in MobileNetV2 $\times 0.5$ and $\times 1.0$.

3.5.5 Analysis of Dynamic Channel Fusion

To validate the *dynamic* property, $\boldsymbol{\Phi}(\boldsymbol{x})$ should have different values over different images. We measure this by averaging the variance of each entry $\sigma_\Phi = \sum_{i,j} \sigma_{i,j}/L^2$, where $\sigma_{i,j}$ is the variance of $\phi_{i,j}(\boldsymbol{x})$, over all validation images. To compare σ_Φ across layers, we normalize it by the variance of the corresponding input feature map. Figure 3.6 shows the normalized variance σ_Φ across layers in MobileNetV2. Clearly, the dynamic coefficients vary more in the higher layers. We believe this is because the higher layers encode more context information, providing more clues to adapt convolution weights.

3.5.6 Inference Time

We use a single-threaded core AMD EPYC CPU 7551P (2.0 GHz) to measure running time (in milliseconds) on MobileNetV2 $\times 0.5$ and $\times 1.0$. Running time is calculated by averaging the inference time of 5,000 images with batch size 1. Both static baseline and DCD are implemented

in PyTorch. Compared with the static baseline, DCD consumes about 8% more MAdds (97.0M vs 104.8M) and 14% more running time (91ms vs 104ms) for MobileNetV2 $\times 0.5$. For MobileNetV2 $\times 1.0$, DCD consumes 9% more MAdds (300.0M vs 326.0M) and 12% more running time (146ms vs 163ms). The overhead is higher in running time than MAdds. We believe this is because the optimizations of global average pooling and fully connected layers are not as efficient as convolution. This small penalty in inference time is justified by the DCD gains of 4.8% and 3.2% top-1 accuracy over MobileNetV2 $\times 0.5$ and $\times 1.0$ respectively.

3.6 Conclusion

In this chapter, we discussed the dynamic network used on large scale image recognition tasks by revisiting dynamic convolution from the view of matrix decomposition and demonstrated the limitations of dynamic attention over channel groups: it multiplies the number of parameters by K and increases the difficulty of joint optimization. We proposed a dynamic convolution decomposition to address these issues. This applies dynamic channel fusion to significantly reduce the dimensionality of the latent space, resulting in a more compact model that is easier to learn with often improved accuracy. We hope that our work provides a deeper understanding of the gains recently observed for dynamic convolution. And in the following chapter, we will further explore how the dynamic architecture can help design networks that requires extremely low computational cost.

Chapter 3 is, in full, based on the material as it appears in the publication of "Revisiting Dynamic Convolution via Matrix Decomposition", Yunsheng Li, Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Ye Yue, Lu Yuan, Zicheng Liu, Mei Chen and Nuno Vasconcelos, in *The Ninth International Conference on Learning Representations* (ICLR), 2021. The dissertation author was the primary investigator and author of this material.

Chapter 4

Large Scale Image Recognition with Extremely Low FLOPs

4.1 Introduction

In this chapter, we further work on the problem of large scale image recognition but with much less computational resources. We start by designing a novel efficient convolution operator. Then motivated by the success of dynamic network, we extend it from being applied on the convolution layers to the activation functions and demonstrate the power of dynamic network under the extremely strict limitation of computational resources.

Recent progress in efficient CNN architectures [96, 66, 72, 97, 67, 74, 75] successfully decreases the computational cost of ImageNet classification from 3.8G FLOPs (ResNet-50 [55]) by two orders of magnitude to about 40M FLOPs (e.g. MobileNet, ShuffleNet), with a reasonable performance drop. However, they suffer from a significant performance degradation when reducing computational cost further. For example, the top-1 accuracy of MobileNetV3 degrades substantially from 65.4% to 58.0% and 49.8% when the computational cost drops from 44M to 21M and 12M MAdds, respectively. In this chapter, *we aim at improving accuracy at the extremely low FLOP regime from 21M to 4M MAdds*, which marks the computational cost decrease of another order of magnitude (from 40M).

The problem of dealing with extremely low computational cost (4M–21M FLOPs) is very challenging, considering that 2.7M MAdds are consumed by a thin stem layer that contains a single 3×3 convolution with 3 input channels and 8 output channels over a 112×112 grid (stride=2). The remaining resources are too limited to design the convolution layers and 1,000 class classifier required for effective classification. As shown in Figure 4.1, a common strategy to reduce the width or depth of existing efficient CNNs (e.g. MobileNet [66, 72, 97] and ShuffleNet [67, 74]) results in a severe performance degradation. Note that we focus on new operator design while fixing the input resolution to 224×224 even for the budget of 4M FLOPs.

In this chapter, we handle the extremely low FLOPs from two perspectives: *node connectivity* and *non-linearity*, which are related to the network width and depth. First, we show

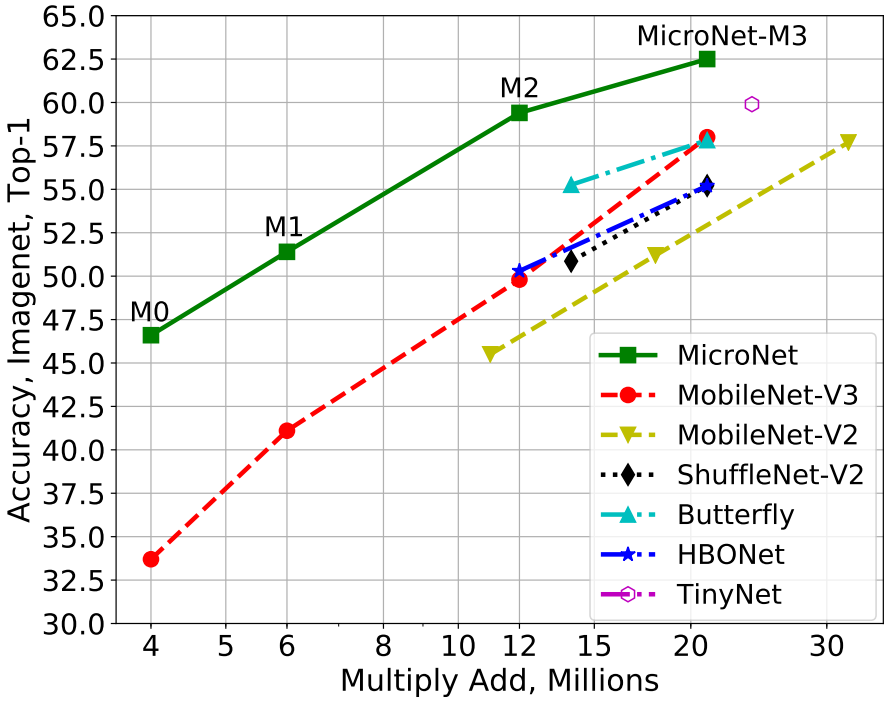


Figure 4.1: **Computational Cost (MAdds) vs. ImageNet Accuracy.** MicroNet significantly outperforms the state-of-the-art efficient networks at very low FLOPs (from 4M to 21M MAdds).

that lowering node connectivity to enlarge network width provides a good trade-off for a given computational budget. Second, we rely on improved layer non-linearities to compensate for reduced network depth, which determines the non-linearity of the network. These two factors motivate the design of more efficient convolution and activation functions.

Regarding convolutions, we propose a *Micro-Factorized convolution (MF-Conv)* to factorize a pointwise convolution into two group convolution layers, where the group number G adapts to the number of channels C as:

$$G = \sqrt{C/R},$$

where R is the channel reduction ratio in between. As analyzed in Section 4.2.1, this equation achieves a good trade-off between the number of channels and node connectivity for a given computational cost. Mathematically, the pointwise convolution matrix is approximated by a block

matrix ($G \times G$ blocks), whose blocks have rank-1. This guarantees minimal path redundancy (with only one path between any input-output pair) and maximum input coverage (per output channel), enabling more channels implementable by the network for a given computational budget.

With regards to non-linearities, we propose a new activation function, named *Dynamic Shift-Max (DY-Shift-Max)*, which non-linearly fuses channels with dynamic coefficients. In particular, the new activation forces the network to learn to fuse different circular channel shifts of the input feature maps, using coefficients that adapt to the input, and to select the best among these fusions. This is shown to enhance the representation power of the group factorization with little computational cost.

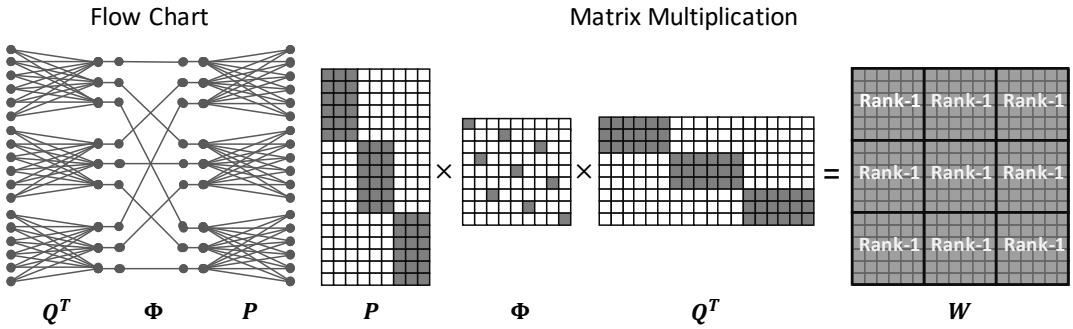
Based upon the two new operators (MF-Conv and DY-Shift-Max), we obtain a family of models, called *MicroNets*. Figure 4.1 summarizes the ImageNet performance, where MicroNets outperform the state-of-the-art by a large margin. In particular, our MicroNet models of 12M and 21M FLOPs outperform MobileNetV3 by 9.6% and 4.5% in terms of top-1 accuracy, respectively. For the extremely challenging regime of 6M FLOPs, MicroNet achieves 51.4% top-1 accuracy, outperforming by 1.6% over MobileNetV3, which is twice as complex (12M FLOPs).

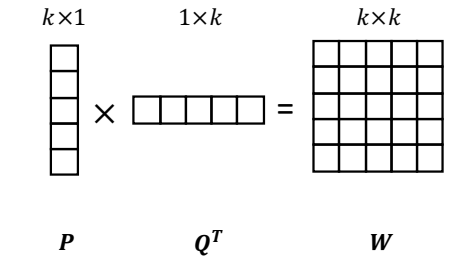
Even though MicroNet is manually designed for theoretical FLOPs, it outperforms MobileNetV3 (which is searched over inference latency) with fast inference on edge devices. Furthermore, our MicroNet surpasses MobileNetV3 on object detection and keypoint detection, but uses substantially less computational cost.

4.2 Micro-Factorized Convolution

The goal of Micro-Factorized convolution is to optimize the trade-off between the number of channels and node connectivity. Here, the *connectivity E* of a layer is defined as the number of paths per output node, where a path connects an input node and an output node.

4.2.1 Micro-Factorized Pointwise Convolution





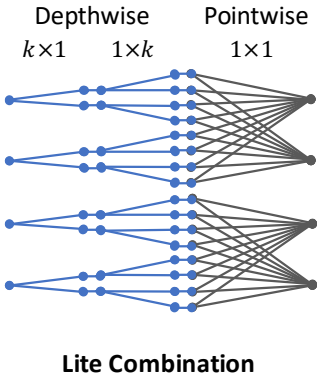


Figure 4.2: Micro-Factorized pointwise and depthwise convolutions. **Top:** factorizing a pointwise convolution into two group-adaptive convolutions. **Middle:** factorizing a $k \times k$ depthwise convolution into a $k \times 1$ and a $1 \times k$ depthwise convolutions. **Bottom:** lite combination of **Top** and **Middle**.

We propose the use of group-adaptive convolution to factorize a pointwise convolution. For conciseness, we assume the convolution kernel $\boldsymbol{W}$ has the same number of input and output channels ($C_{in} = C_{out} = C$) and ignore bias terms. The kernel matrix $\boldsymbol{W}$ is factorized into two group-adaptive convolutions, where the number of groups G depends on the number of channels C, according to

$$\boldsymbol{W} = \boldsymbol{P}\boldsymbol{\Phi}\boldsymbol{Q}^T, \tag{4.1}$$

where $\boldsymbol{W}$ is a $C \times C$ matrix, $\boldsymbol{Q}$ is a $C \times \frac{C}{R}$ matrix that compresses the number of channels by a factor of R, and $\boldsymbol{P}$ is a $C \times \frac{C}{R}$ matrix that expands the number of channels back to C. $\boldsymbol{P}$ and $\boldsymbol{Q}$ are diagonal block matrices with G blocks, each implementing the convolution of a group of channels. $\boldsymbol{\Phi}$ is a $\frac{C}{R} \times \frac{C}{R}$ permutation matrix, shuffling channels similarly to [67]. The computational complexity of the factorized layer is $O = \frac{2C^2}{RG}$. Figure 4.2-Top shows an example of the factorization, for $C = 18$, $R = 2$ and $G = 3$.

The $\frac{C}{R}$ channels of matrix $\boldsymbol{\Phi}$ are denoted *hidden channels*. The grouping structure limits the number of these channels that are affected by (affect) each input (output) of the layer. Specifically, each hidden channel connects to $\frac{C}{G}$ input channels and each output channel connects to $\frac{C}{RG}$ hidden channels. The number $E = \frac{C^2}{RG^2}$ of input-output connections per output channel denotes the *connectivity* E of the layer. When the computational budget $O = \frac{2C^2}{RG}$ and the compression factor R are fixed, the number of channels C and connectivity E change with G in opposite directions,

$$C = \sqrt{\frac{ORG}{2}}, \;\; E = \frac{O}{2G}. \tag{4.2}$$

This is illustrated in Figure 4.3. As the number of groups G increases, C increases but E decreases.

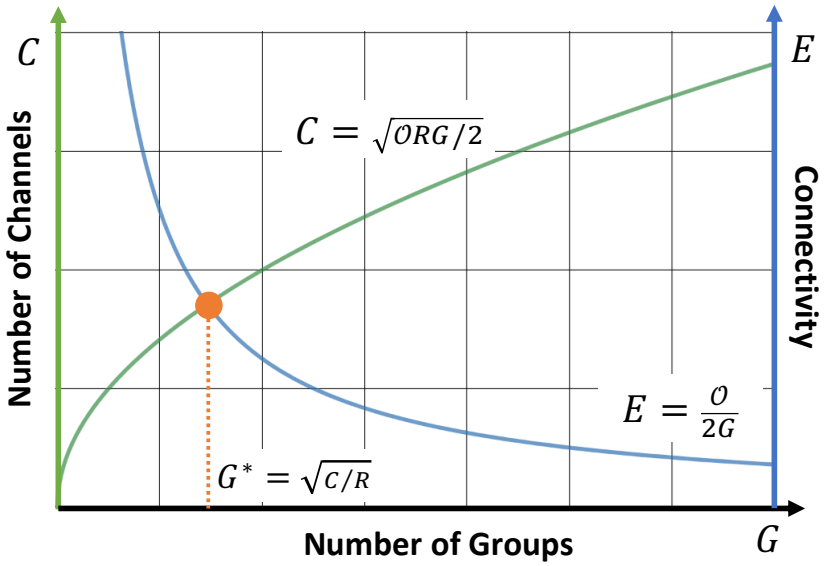


Figure 4.3: **Number of Channels** C **vs. Connectivity** E over number of groups G. We assume that the computational cost O and the reduction ratio R are fixed. Best viewed in color.

The two curves intersect ($C = E$) when

$$G = \sqrt{C/R}, \tag{4.3}$$

in which case each output channel connects to all input channels exactly once ($E = C$). This guarantees that no redundant paths exist between any input-output pair (minimum path redundancy) while guaranteeing the existence of a path between each pair (maximum input coverage). Eq. 4.3 is a defining property of micro-factorized pointwise convolution. It implies that the number of groups G is *not* fixed, but defined by the number of channels C and the compression factor R, according to a square root law that optimally balances the number of channels C and input/output connectivity. Mathematically, the resulting convolution matrix $\boldsymbol{W}$ is divided into $G \times G$ rank-1 blocks, as shown in Figure 4.2-Top.

4.2.2 Micro-Factorized Depthwise Convolution

Figure 4.2-Middle shows how micro-factorization can be applied to a $k \times k$ depthwise convolution. The convolution kernel is factorized into a $k \times 1$ and a $1 \times k$ kernel. This follows Eq. 4.1, with per channel $k \times k$ kernel matrix $\boldsymbol{W}$, $k \times 1$ vector $\boldsymbol{P}$, $1 \times k$ vector $\boldsymbol{Q^T}$ and $\boldsymbol{\Phi}$ a scalar of value 1. This low rank approximation reduces the computational complexity from $O(k^2C)$ to $O(kC)$.

4.2.3 Combining Micro-Factorized Pointwise and Depthwise Convolutions

Micro-Factorized pointwise and depthwise convolutions can be combined in two different ways: (a) regular combination, and (b) lite combination. The former simply concatenates the two convolutions. The lite combination, shown in Figure 4.2-Bottom, uses Micro-Factorized depthwise convolutions to expand the number of channels, by applying multiple spatial filters per channel. It then applies one group-adaptive convolution to fuse and squeeze the number of channels. Compared to its regular counterpart, it spends more resources on learning spatial filters (depthwise) by saving channel fusion (pointwise) computations, which is empirically validated to be more effective for implementation of lower network layers.

4.3 Dynamic Shift-Max

So far, we have discussed the design of efficient static networks, which do not change their weights according to the input. We now introduce dynamic Shift-Max (DY-Shift-Max), a new dynamic non-linearity that strengthens connections between the groups created by micro-factorization. This is complementary to Micro-Factorized pointwise convolution, which focuses on connections within a group.

Let $\boldsymbol{x} = \{x_i\}$ $(i = 1, \ldots, C)$ denote an input vector (or tensor) with C channels that are divided into G groups of $\frac{C}{G}$ channels each. The j-group circular shift (shifting $j\frac{C}{G}$ channels) of

$\boldsymbol{x}$ is the vector $\hat{\boldsymbol{x}}^j$ such that $\hat{x}_i^j = x_{(i+j\frac{C}{G}) \bmod C}$. Dynamic Shift-Max outputs the maximum of K fusions, each of which combines multiple (J) group shifts as:

$$y_i = \max_{1 \le k \le K} \{ \sum_{j=0}^{J-1} a_{i,j}^k(\boldsymbol{x}) x_{(i+j\frac{C}{G}) \bmod C} \}, \tag{4.4}$$

where $a_{i,j}^k(\boldsymbol{x})$ is a dynamic weight, i.e., a weight that depends on the input $\boldsymbol{x}$. It is implemented as a hyper-function (with CJK output dimension) that consists of a sequence of average pooling, two fully connected layers, and a sigmoid layer, as in Squeeze-and-Excitation [96].

In this way, DY-Shift-Max implements two forms of non-linearity: it (a) outputs the maximum of K fusions of J groups, and (b) weighs each fusion by a dynamic parameter $a_{i,j}^k(\boldsymbol{x})$. The first non-linearity is complementary to Micro-Factorized pointwise convolution, which focuses on connectivity within each group, strengthening the connections between groups. The second enables the network to tailor this strengthening to the input $\boldsymbol{x}$. The two operations increase the representation power of the network, compensating for the loss inherent to the reduced number of layers.

DY-Shift-Max synthesizes CJK weights $a_{i,j}^k(\boldsymbol{x})$ from input $\boldsymbol{x}$. Its computational complexity is a sum of (a) average pooling $O(HWC)$, (b) generation of the $a_{i,j}^k(\boldsymbol{x})$ weights $O(C^2JK)$, and (c) application of dynamic Shift-Max per channel and spatial location $O(HWCJK)$. This leads to a light-weight model when J and K are small. Empirically, a good trade-off between classification performance and complexity is achieved when $J = 2$ and $K = 2$.

4.4 MicroNet

Below we describe in detail the design of MicroNet, using Micro-Factorized convolution and dynamic Shift-Max.

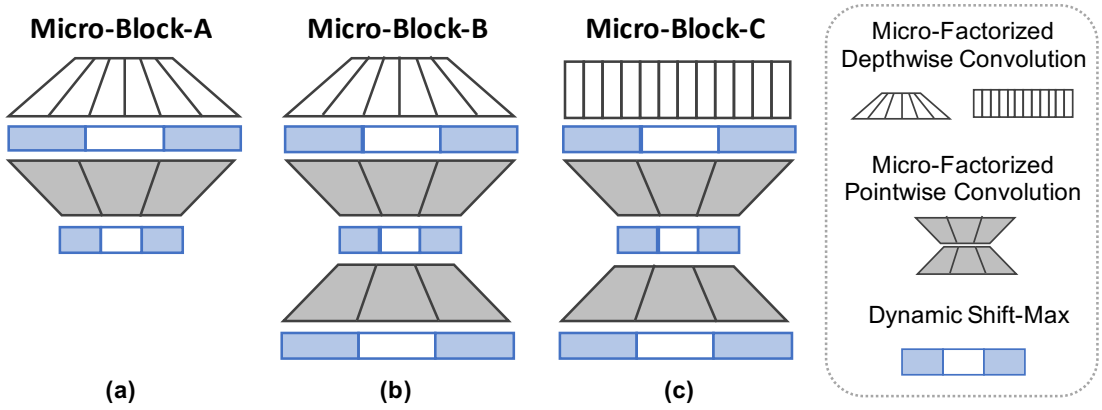


Figure 4.4: **Diagram of three Micro-Blocks. (a) Micro-Block-A**: the lite combination of Micro-Factorized pointwise (PW) and depthwise (DW) Conv. **(b) Micro-Block-B**: combination of Micro-Block-A and Micro-Block-C. **(c) Micro-Block-C**: the regular combination of Micro-Factorized PW and DW Conv.

4.4.1 Micro-Blocks

MicroNet models consist of three Micro-Blocks of Figure 4.4, which combine Micro-Factorized pointwise and depthwise convolutions in different ways. All of the Micro-Blocks use the dynamic Shift-Max activation function.

Micro-Block-A: The Micro-Block-A of Figure 4.4a, uses the lite combination of Micro-Factorized pointwise and depthwise convolutions of Figure 4.2-Right. It expands the number of channels with Micro-Factorized depthwise convolution, and compresses them with a group-adaptive convolution. It is best suited to implement lower network layers of higher resolution (e.g. 112×112 or 56×56).

Micro-Block-B: The Micro-Block-B of Figure 4.4b is used to connect Micro-Block-A and Micro-Block-C. Different from Micro-Block-A, it uses a full Micro-Factorized pointwise convolution, which includes two group-adaptive convolutions. Hence, it both compresses and expands the number of channels. All MicroNet models have a single Micro-Block-B (see Table 4.1).

Micro-Block-C: The Micro-Block-C of Figure 4.4c implements the regular combination

of Micro-Factorized depthwise and pointwise convolutions. It is best suited for the higher network layers (see Table 4.1) since it assigns more computation to channel fusion (pointwise) than the lite combination. The skip connection is used when the input and output have the same dimension.

Each micro-block has three hyper-parameters: kernel size k, number of output channels C, compression factor R of the bottleneck of Micro-Factorized pointwise convolution. Note that the number of groups in the two group-adaptive convolutions is determined by Eq. 4.3.

4.4.2 Architectures

All models are manually designed to optimize for FLOPs, which is a theoretical and device independent metric. We hope this can be leveraged by new hardware design and optimization for edge devices. We aware that FLOPs is *not* equivalent to inference latency at existing hardware and will show in experiment that MicroNet also improves accuracy and latency. We propose four models (M0, M1, M2, M3) of different computational cost (4M, 6M, 12M, 21M MAdds) based on the Micro-Blocks above. Table 4.1 presents their full specification. These networks follow the same pattern from low to high layers: stem layer $\rightarrow$ Micro-Block-A $\rightarrow$ Micro-Block-B $\rightarrow$ Micro-Block-C. All models are handcrafted, without network architecture search (NAS). The network hyper-parameters are selected based on simple rules: R is fixed (4 for M0, 6 for MicroNet-M1,M2,M3), C increases from low to high levels, depth increases from M0 to M3. For the deepest model (M3), we only use one dynamic Shift-Max layer per block after the depthwise convolution. The stem layer includes a 3×1 convolution and a 1×3 group convolution, and is followed by a ReLU. The second convolution expands the number of channels.

Table 4.1: **MicroNet Architectures**. "stem" refers to the stem layer. "Micro-A", "Micro-B", and "Micro-C" refers to three Micro-Blocks (see section 4.4.1 and Figure 4.4 for more details). k is the kernel size, C is the number of output channels, R is the channel reduction ratio in Micro-Factorized pointwise convolution.

	M0				M1				M2				M3			
Output	Block	k	C	$\frac{C}{R}$	Block	k	C	$\frac{C}{R}$	Block	k	C	$\frac{C}{R}$	Block	k	C	$\frac{C}{R}$
112×112	stem	3	4	2	stem	3	6	3	stem	3	8	4	stem	3	12	4
56×56	Micro-A	3	16	8	Micro-A	3	24	8	Micro-A	3	32	12	Micro-A	3	48	16
	Micro-A	3	32	12	Micro-A	3	32	16	Micro-A	3	48	16	Micro-A	3	64	24
28×28									Micro-B	3	144	24	Micro-B	3	144	24
	Micro-B	5	64	16	Micro-B	5	96	16	Micro-C	5	192	32	Micro-C	3	192	32
	Micro-C	5	128	32	Micro-C	5	192	32	Micro-C	5	192	32	Micro-C	5	192	32
14×14									Micro-C	5	384	64	Micro-C	5	384	64
													Micro-C	5	480	80
													Micro-C	5	480	80
	Micro-C	5	256	64	Micro-C	5	384	64	Micro-C	5	576	96	Micro-C	5	720	120
7×7	Micro-C	3	384	96	Micro-C	3	576	96	Micro-C	3	768	128	Micro-C	3	720	120
													Micro-C	3	864	144
1×1	avg pool → 2fc → softmax															
	4M MAdds, 1.0M Param				6M MAdds, 1.8M Param				12M MAdds, 2.4M Param				21M MAdds, 2.6M Param			

4.4.3 Relation to Prior Work

MicroNet has various connections to the recent deep learning literature. It is related to the popular MobileNet [66, 72, 97] and ShuffleNet [67, 74] models. It shares the inverted bottleneck structure with MobileNet and the use of group convolution with ShuffleNet. In contrast, MicroNet differs from these models in both its convolutions and activation functions. First, it factorizes pointwise convolutions into group-adaptive convolutions, with the number of groups $G = \sqrt{C/R}$ that is channel adaptive and guarantees minimum path redundancy. Second, it factorizes depthwise convolution. Third, it relies on a novel activation function, dynamic Shift-Max, to strengthen group connectivity in a non-linear and input dependent manner. Dynamic Shift-Max itself generalizes the recently proposed dynamic ReLU [90] (i.e., dynamic ReLU is a special case where $J = 1$ and each channel is activated alone).

4.5 Experiments on ImageNet Classification

We start by evaluating the four MicroNet models (M0–M3) on the task of ImageNet [3] classification. ImageNet has 1000 classes, including 1,281,167 images for training and 50,000 images for validation. In this and the following sections, the baseline MobileNetV3-Small in [97] is denoted as MobileNetV3, for conciseness.

4.5.1 Experimental Set-up

All models are trained using an SGD optimizer with 0.9 momentum. The image resolution is 224×224. Data augmentation of standard random cropping and flipping is used. We use a mini-batch size of 512, and a learning rate of 0.02. Each model is trained for 600 epochs with cosine learning rate decay. The weight decay is 3e-5 and dropout rate is 0.05 for smaller MicroNets (M0, M1, M2). For the largest model M3, the weight decay is 4e-5 and dropout rate is 0.1.

Table 4.2: **The path from MobileNet to MicroNet** evaluated on ImageNet classification. Under similar FLOPs, MobileNet is compared to three Micro-Factorized convolution options: depthwise (DW), pointwise (PW), lite combination at low levels (Lite) and dynamic Shift-Max with static coefficient $a^k_{i,j}$ in Eq. 4.4.

	Micro-Fac Conv			Shift-Max				
	DW	PW	Lite	static	dynamic	Param	MAdds	Top-1
Mobile						1.3M	10.6M	44.9
Micro	✓					1.7M	10.6M	46.4
	✓	✓				1.7M	10.6M	50.0
	✓	✓	✓			1.8M	10.5M	51.7
	✓	✓	✓	✓		1.9M	11.8M	54.4
	✓	✓	✓		✓	2.4M	12.4M	**58.5**

4.5.2 Ablation Studies

Several ablations were performed using MicroNet-M2. All models are trained for 300 epochs. The default hyper parameters of DY-Shift-Max were set as J=2, K=2.

From MobileNet to MicroNet: Table 4.2 shows the path from MobileNet to MicroNet. Both share the inverted bottleneck structure. Here, we modify MobileNetV2 (without SE [88]) such that it has complexity (10.6M MAdds) similar to the static Micro-Factorized convolution variants of row 2–4. The introduction of Micro-Factorized depthwise convolutions improves performance by 1.5%. Micro-Factorized pointwise convolutions adds another 3.6% and the lite combination at lower layers adds a final gain of 1.7%. Altogether the three factorizations boost the top-1 accuracy of the static network from 44.9% to 51.7%. The addition of static and dynamic Shift-Max further increases this gain by 2.7% and 6.8% respectively, for a small increase in computation. This demonstrates that both *Micro-Factorized Convolutions* and *Dynamic Shift-Max* are effective and complementary mechanisms for the implementation of networks with extremely low computational cost.

Number of Groups G: Micro-Factorized pointwise convolution includes two group-adaptive convolutions, with a number of groups equal to the integer closest to $G = \sqrt{C/R}$. Table 4.3a compares this to networks of similar structure and FLOPs (about 10.5M MAdds), but using a

Table 4.3: **Ablations of Micro-Factorized convolution** on ImageNet classification. ✶ indicates the default choice for the rest of the experiments.

G	Param	MAdds	Top-1
1	1.3M	10.6M	48.8
2	1.5M	10.5M	50.2
4	1.7M	10.6M	50.7
8	1.7M	10.6M	50.8
$G=\sqrt{C/R}$	1.8M	10.5M	**51.7**

(a) **Fixed group number** G**.**

$\lambda=\frac{G}{\sqrt{C/R}}$	Param	MAdds	Top-1
0.25	1.5M	10.5M	50.2
0.5	1.7M	10.6M	51.6
✶ 1.0	1.8M	10.5M	**51.7**
2.0	2.1M	10.5M	50.6
4.0	2.2M	10.7M	47.6

(b) **Adaptive group number** G**.**

Levels				
low	high	Param	MAdds	Top-1
		1.7M	10.6M	50.0
✶ ✓		1.8M	10.5M	**51.7**
✓	✓	2.0M	10.6M	51.2

(c) **Lite combination** at different levels

fixed group cardinality. Group-adaptive convolution achieves higher accuracy, demonstrating the importance of its optimal trade-off between input/output connectivity and the number of channels.

This is further confirmed by Table 4.3b, which compares different options for the adaptive number of groups. This is controlled by a multiplier λ such that $G=\lambda\sqrt{C/R}$. Larger λ corresponds to more channels but less input/output connectivity (see Figure 4.3). The optimal balance is achieved when λ is between 0.5 and 1. Top-1 accuracy drops when λ either increases (more channels but less connectivity) or decreases (fewer channels but more connectivity) from this optimal point. The value $\lambda=1$ is used in the remainder of the chapter. Note that all models in Table 4.3b have similar computational cost (about 10.5M MAdds).

Lite combination: Table 4.3c compares using the lite combination of Micro-Factorized pointwise and depthwise convolutions (Figure 4.2-Right) at different layers. The lite combination is more effective for lower layers. Compared to the regular combination, it saves computations from channel fusion (pointwise) to allow more spatial filters (depthwise).

Activation functions: Dynamic Shift-Max is compared to three previous activation

Table 4.4: **Dynamic Shift-Max vs. other activation functions** on ImageNet classification. MicroNet-M2 is used.

Activation	Param	MAdds	Top-1	Top-5
ReLU[91]	1.8M	10.5M	51.7	74.3
SE[88]+ReLU	2.1M	10.9M	54.4	76.8
Dynamic ReLU [90]	2.4M	11.8M	56.0	78.0
Dynamic Shift-Max	2.4M	12.4M	**58.5**	**80.1**

Table 4.5: **Dynamic Shift-Max at different layers** evaluated on ImageNet. MicroNet-M2 is used. A_1, A_2, A_3 indicate three activation layers sequentially in Micro-Block-B and Micro-Block-C (see Figure 4.4). Micro-Block-A only includes A_1 and A_2.

	A_1	A_2	A_3	Param	MAdds	Top-1	Top-5
ReLU	–	–	–	1.8M	10.5M	51.7	74.3
Dynamic Shift-Max	✓	–	–	2.1M	11.3M	55.9	77.9
	–	✓	–	2.0M	10.6M	53.3	76.0
	–	–	✓	2.1M	11.2M	54.8	77.2
	✓	✓	–	2.2M	11.5M	56.6	78.3
	✓	–	✓	2.3M	12.2M	57.9	79.6
	–	✓	✓	2.2M	11.4M	55.5	77.8
	✓	✓	✓	2.4M	12.4M	**58.5**	**80.1**

functions: ReLU [91], SE+ReLU [88], and dynamic ReLU [90]. Table 4.4 shows that dynamic Shift-Max outperforms all three by a clear margin (at least 2.5%). Note that dynamic ReLU is the special case of dynamic Shift-Max with $J = 1$ (see Eq. 4.4).

Location of DY-Shift-Max: Table 4.5 shows the top-1 accuracy when dynamic Shift-Max is implemented in different combinations of the three layers of the micro-blocks of Figure 4.4. When used in a single layer, dynamic Shift-Max should be placed after the depthwise convolution. This improves the top-1 accuracy over a network with ReLU activations by 4.2%. Adding a Dynamic Shift-Max activation at the Micro-Block output further improves performance by 2%. Finally, using three layers of Dynamic Shift-Max further increases the gain over the ReLU network to 6.8%.

Hyper-parameters in DY-Shift-Max: Table 4.6 shows the results of using different combinations of K and J in Eq. 4.4. We add a ReLU when $K = 1$ as only one element is left in the max operator. The baseline of the first row ($J = 1$, $K = 1$) is equivalent to SE+ReLU [88]. For

Table 4.6: **Ablations of two hyper parameters in dynamic Shift-Max** (J, K in Eq. 4.4) on ImageNet classification. ✶ indicates the default choice for the rest of the experiments.

J	K	Param	MAdds	Top-1	Top-5
1	1	2.1M	10.9M	54.4	76.8
2	1	2.2M	11.8M	55.9	78.2
✶ 2	2	2.4M	12.4M	58.5	80.1
2	3	2.6M	13.8M	58.1	79.7
1	2	2.2M	11.2M	55.5	77.6
✶ 2	2	2.4M	12.4M	58.5	80.1
3	2	2.6M	14.2M	59.0	**80.3**
3	3	2.8M	15.3M	**59.1**	**80.3**

fixed $J = 2$ (fusion of two groups), the best of two fusions ($K = 2$) is better than a single fusion ($K = 1$), but adding a third fusion does not help, since it only adds path redundancy. When K is fixed at $K = 2$ (best of two fusions), fusing more groups J is consistently better but requires more FLOPs. A good tradeoff is achieved with $J = 2$ and $K = 2$, enabling a gain of 4.1% over the baseline, for an additional 1.5M MAdds.

4.5.3 Comparison to Prior Networks

Table 4.7 compares MicroNet to the state-of-the-art models, which have complexity less than 24M FLOPs. As the prior works lack of reported results within 10M FLOPs budget, we extend the popular MobileNetV3 to 6M and 4M FLOPs as baseline, by using width multiplier 0.2 and 0.15 respectively. They share the same training setup with MicroNet.

To make comparison fair, two variations of M1–M3 (e.g. M3$^{\#}$ and M3) are used. The former (M3$^{\#}$) requires similar model size to but fewer FLOPs than the baseline (MobileNetV3 0.5×). The latter (M3) requires similar FLOPs but allows more parameters (up to 1M), best serving scenarios that FLOPs is more critical than memory. This is due to the difficulty to match both model size and FLOPs, except for the smallest model (M0). Note that M3$^{\#}$ has similar structure to M3, only shrinking the model size by reducing network width and parameters in dynamic Shift-Max.

Table 4.7: **ImageNet [3] classification results**. # stands for the MicroNet variation with similar model size to but fewer MAdds than the MobileNetV3-Small baseline. † indicates our implementation. "–": not available in the original paper. Note that input resolution 224×224 is used other than HBONet/TinyNet.

Model	#Param	MAdds	Top-1	Top-5
MobileNetV3 0.15×†	1.0M	4M	33.7	57.2
MicroNet-M0	1.0M	4M	**46.6**	**70.6**
MobileNetV3 0.2×†	1.2M	6M	41.1	65.2
MicroNet-M1#	1.2M	5M	49.4	72.9
MicroNet-M1	1.8M	6M	**51.4**	**74.5**
ShuffleNetV1 0.25× [67]	–	13M	47.3	–
MobileNetV3 0.35× [97]	1.4M	12M	49.8	–
HBONet (96×96) [100]	–	12M	50.3	73.8
MobileNetV3+BFT 0.5× [98]	–	15M	55.2	–
MicroNet-M2#	1.4M	11M	58.2	80.1
MicroNet-M2	2.4M	12M	**59.4**	**80.9**
HBONet (128×128) [100]	–	21M	55.2	78.0
ShuffleNetV2+BFT [98]	–	21M	57.8	–
MobileNetV3 0.5× [97]	1.6M	21M	58.0	–
TinyNet-E (106×106) [99]	2.0M	24M	59.9	81.8
MicroNet-M3#	1.6M	20M	61.3	82.9
MicroNet-M3	2.6M	21M	**62.5**	**83.1**

In all cases, MicroNet outperforms all prior networks by a clear margin. For instance, MicroNet-M1#, M2#, M3# outperform their MobileNetV3 counterpart by 8.3%, 8.4%, and 3.3%, respectively. Given another 1M budget on model size, MicroNet-M1, M2, M3 increase these gains by 2.0%, 1.2% and 1.2%, respectively. MicroNet-M0 outperforms MobileNetV3 0.15× by 12.9% (46.6% vs. 33.7%), demonstrating its better handle of cutting computational cost from 6M to 4M MAdds. In particular, the top-1 accuracy drops by 4.8% from MicroNet-M1 to M0, while the accuracy degrades by 7.4% from MobileNetV3 ×0.2 to ×0.15. When compared to recent MobileNet and ShuffleNet improvements, such as ButterflyTransforms [98] and TinyNet [99], MicroNet models have gains of more than 2.6% top-1 accuracy but use less FLOPs. This demonstrates the effectiveness of MicroNet at extremely low FLOPs.

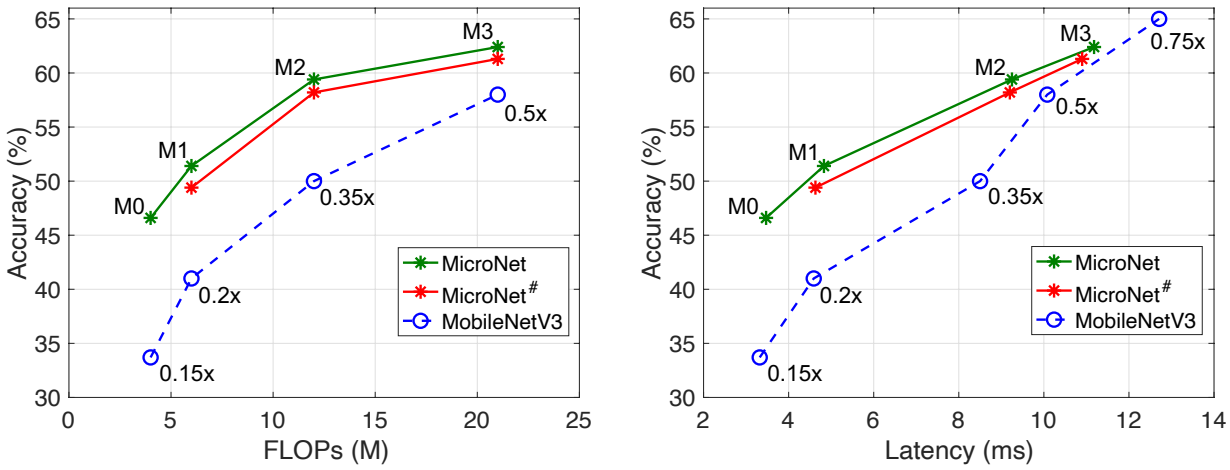


Figure 4.5: Evaluation on ImageNet classification. **Left**: *top-1 accuracy vs. FLOPs*. **Right**: *top-1 accuracy vs. latency*. MicroNet outperforms MobileNetV3, especially at extremely low computational cost (more than 5% gain on top-1 accuracy when FLOPs is less than 15M or latency is less than 9ms).

4.5.4 Inference Latency

We also measure the inference latency of MicroNet on an Intel(R) Xeon(R) CPU E5-2620 v4 (2.10GHz). Following the common settings in [72, 97], we test under single-threaded mode with batch size 1. The average inference latency of 5,000 images (with resolution 224×224) is reported. Figure 4.5-Right shows the comparison between MicroNet and MobileNetV3-Small. To achieve similar performance, MicroNet clearly consumes less runtime than MobileNetV3. For example, MicroNet with 55% accuracy has a latency less than 7ms, while MobileNetV3 requires about 9.5ms. The accuracy-latency curve is slightly degraded when using MicroNet with fewer parameters (M1$^{\#}$, M2$^{\#}$, M3$^{\#}$), but it still outperforms MicroNetV3. Although the largest MicroNet model (M3) only slightly outperforms MobileNetV3 for the same latency, MicroNet gains significantly more improvement over MobileNetV3 when the latency decreases. In particular, at a latency of 4ms, MicroNet improves over MobileNetV3 by 10%, demonstrating its strength at low computational cost.

4.5.5 Discussion

As shown in Figure 4.5, MicroNet clearly outperforms MobileNetV3 under the same FLOPs, but the gap shrinks under the same latency. This is due to two reasons. First, different from MobileNetV3 that is optimized for latency by search, MicroNet is manually designed based on theoretical FLOPs. Second, the implementation of group convolution and dynamic Shift-Max are not optimized (we use PyTorch for implementation). We observe that the latency of group convolution is not proportionally reduced as the number of groups increases, and dynamic Shift-Max is significantly slower than convolution with the same FLOPs.

We believe that the runtime performance of MicroNet can be further improved by using hardware-aware architecture search to find latency friendly combination of Micro-Factorized convolution and dynamic Shift-Max. MicroNet can also leverage the improvement of optimization in group convolution [101] and dynamic Shift-Max to speed up. We will investigate these in the future work.

4.6 Experiments on Object Detection

In this section, we evaluate the generalization ability of MicroNet on COCO object detection [102]. All models are trained on `train2017` and evaluated in mean Average Precision (mAP) on `val2017`. Following [79], MicroNet is used as a drop-in replacement for the backbone feature extractor in both the two-stage Faster R-CNN [103] with Feature Pyramid Networks (FPN) [104] and the one-stage RetinaNet [105]. All models are trained using SGD for 36 epochs ($3\times$) from ImageNet pretrained weights with the hyper-parameters and data augmentation suggested in [106].

The detection results are shown in Table 4.8, where the backbone FLOPs are calculated using image size 224×224 as common practice. With significantly lower backbone FLOPs (21M *vs* 56M), MicroNet-M3 achieves higher mAP than MobileNetV3-Small $\times 1.0$ both on Faster

Table 4.8: **COCO object detection results**. All models are trained on `train2017` for 36 epochs (3×) and tested on `val2017`. MAdds is computed on image size 224×224.

Backbone	DET Framework	MAdds	mAP
MobileNetV3 ×1.0	R-CNN	56M	25.9
MicroNet-M3		21M	**26.2**
MicroNet-M2		12M	22.7
MobileNetV3 ×1.0	RetinaNet	56M	24.0
MicroNet-M3		21M	**25.4**
MicroNet-M2		12M	22.6

R-CNN and RetinaNet frameworks, demonstrating its capability to transfer to detection task.

4.7 Experiments on Human Pose Estimation

We also evaluate MicroNet on COCO single person keypoint detection. All models are trained on `train2017` that includes 57K images and 150K person instances labeled with 17 keypoints, and evaluated on `val2017` that contains 5000 images, using the mean average precision (AP) over 10 object key point similarity (OKS) thresholds.

Similar to object detection, two MicroNet models (M2, M3) are considered. The models are modified for the keypoint detection task, by increasing the resolution (×2) of a select set of blocks (all blocks with stride of 32). Each model contains a head with three micro-blocks (one of stride 8 and two of stride 4) and a pointwise convolution that generates heatmaps for 17 keypoints. Bilinear upsampling is used to increase the head resolution, and the spatial attention mechanism of [90] is used. Both models are trained from scratch for 250 epochs using Adam optimizer [107]. The human detection boxes are cropped and resized to 256×192. The training and testing follow the setup of [108, 109].

Table 4.9 compares MicroNet-M3 and M2 with a strong efficient baseline, which only requires 726.9M MAdds and 2.1M parameters. The baseline applies MobileNetV3-Small ×1.0 as backbone and mobile blocks (inverted residual bottleneck blocks) in the head (see [2] for details). Our MicroNet-M3 only consumes 22% (163.2M/726.9M) of the FLOPs used by the baseline

Table 4.9: **COCO keypoint detection results**. All models are trained on `train2017` and tested on `val2017` with input resolution 256×192. The head structure in [2] is used for all models. MicroNet-M3 has similar model size, consumes significantly less MAdds, but achieves higher accuracy than MobileNetV3.

Backbone	Head	Param	MAdds	AP	$AP^{0.5}$	$AP^{0.75}$	AP^{M}	AP^{L}
MobileNetV3 ×1.0	Mobile-Blocks	2.1M	726.9M	57.1	83.8	63.7	55.0	62.2
MicroNet-M3	Micro-Blocks	2.2M	163.2M	**58.7**	**84.0**	**65.5**	**56.0**	**64.2**
MicroNet-M2	Micro-Blocks	1.8M	116.8M	54.9	82.0	60.3	53.2	59.6

but achieves higher performance, demonstrating its effectiveness for low-complexity keypoint detection. MicroNet-M2 provides a good handle for even lower complexity (116.8M FLOPs).

4.8 Conclusion

In this chapter, we presented MicroNet to handle extremely low computational cost on image recognition tasks. It is built on two proposed operators: Micro-Factorized convolution and Dynamic Shift-Max. The former balances between the number of channels and input/output connectivity via low rank approximations on both pointwise and depthwise convolutions. The latter fuses consecutive channel groups dynamically, enhancing both node connectivity and non-linearity to compensate the loss caused by the depth reduction. Experimental results show that a family of MicroNets achieve solid improvement for three tasks (image classification, object detection and human pose estimation) under extremely low FLOPs. We hope this work provides good baselines for efficient CNNs on multiple vision tasks.

Chapter 4 is, in full, based on the material as it appears in the submission of "MicroNet: Improving Image Recognition with Extremely Low FLOPs", Yunsheng Li, Yinpeng Chen, Xiyang Dai, Dongdong Chen, Mengchen Liu, Lu Yuan, Zicheng Liu, Lei Zhang, Nuno Vasconcelos, in *Proceedings of IEEE International Conference on Computer Vision* (ICCV), 2021. The dissertation author was the primary investigator and author of this material.

Chapter 5

Domain Adaptation on Semantic Segmentation

5.1 Introduction

In the previous chapters, we discussed the design of dynamic networks for different constraints of computational resources from efficient multi-domain learning to the design of extremely efficient networks. In this and the following chapters, we will move on to another type of resource constrained problem where the data annotation is lacking. Due to the large varieties of research subjects about dealing with the issue of lack of data annotation, in the thesis we mainly discuss one of the critical subjects- *domain adaptation* and show how the dynamic network can be applied in this scenario. As a start point, in this chapter, we will explore a series of key techniques that are widely used for domain adaptation on semantic segmentation.

Recent progress on image semantic segmentation [110] has been driven by deep neural networks trained on large datasets. Unfortunately, collecting and manually annotating large datasets with dense pixel-level labels has been extremely costly due to the amount of human effort requires. Recent advances in computer graphics make it possible to train CNNs on photo-realistic synthetic imagery with computer-generated annotations [111, 112]. Despite this, the domain mismatch between the real images (*target*) and the synthetic data (*source*) cripples the models' performance. Domain adaptation helps address this domain shift problem. Specifically, we focus on the hard case of the problem where no labels from the target domain are available. This class of techniques is commonly referred to as Unsupervised Domain Adaptation.

Traditional methods for domain adaptation involve minimizing some measure of distance between the source and the target distributions. Two commonly used measures are the first and second order moment [30], and learning the distance metrics using Adversarial approaches [9, 113]. Both approaches have had good success in the classification problems (e.g., MNIST [114], USPS [115] and SVHN [62]); however, as pointed out in [116],their performance improvement are quite limited on the semantic segmentation problem.

Recently, domain adaptation for semantic segmentation has made good progress by

separating it into two sequential steps. First, it translates images from the source domain to the target domain with an image-to-image translation model (e.g., CycleGAN [117]). And then it adds a discriminator on top of the features of the segmentation model to further decrease the domain gap [29, 118]. When the domain gap is reduced by the former step, the latter one is easy to learn and can further decreases the domain shift. Unfortunately, the segmentation model heavily relies on the quality of image-to-image translation. Once the image-to-image translation fails, nothing can be done to make it up in the following stages.

In this chapter, we propose a new *bidirectional learning* framework for domain adaptation of semantic segmentation. The system involves two separated modules: image-to-image translation model and segmentation adaptation model similar to [29, 118], but the learning process involves two directions (i.e., "translation-to-segmentation" and "segmentation-to-translation"). The whole system forms a closed-loop learning. Both models will be motivated to promote each other alternatively, causing the domain gap to be gradually reduced. Thus, how to allow one of both modules providing positive feedback to the other is the key to success.

On the forward direction (i.e., "translation-to-segmentation", similar to [29, 118]), we propose a *self-training* (ST) approach in training our segmentation adaptation model. Different from segmentation models trained on real data, the segmentation adaptation model is trained on both synthetic and real datasets, but the real data have no annotations. At every time, we may regard the predicted labels for real data with high confidence as the approximate to the ground truth labels, and then use them only to update the segmentation adaptation model by excluding predicted labels with low confidence. This process is referred as *self-training*, which aligns two domains better than one-trial learning that are widely used in existing approaches. Furthermore, better segmentation adaptation model would contribute to better translation model through our reverse direction learning.

On the reverse direction (i.e., "segmentation-to-translation"), our translation model would be iteratively improved by the segmentation adaptation model, which is different from [29, 118]

where the image-to-image translation is not updated once the model is trained. For the purpose, we propose a new *perceptual loss*, which forces the semantic consistency between every image pixel and its translated version, to build the bridge between translation model and segmentation adaptation model. With the constraint in the translation model, the gap in visual appearance (e.g., lighting, object textures), between the translated images and real datasets (*target*) can be further decreased. Thus, the segmentation model can be further improved accordingly through our forward direction learning.

From the above two directions, both the translation model and the segmentation adaptation model complement each other, which helps achieve state-of-the-art performance for adapting large-scale rendered image dataset SYNTHIA [112]/GTA5 [111], to real image dataset, Cityscapes [119], and outperform other methods by a large margin. Moreover, the proposed method is generic to different kinds of backbone networks.

In summary, our key contributions are:

1. We present a *bidirectional learning* system for semantic segmentation, which is a closed loop to learning the segmentation adaptation model and the image translation model alternatively.

2. We propose a *self-training* algorithm for the segmentation adaptation model, which incrementally align the source domain and the target domain at the feature level, based on the translated results.

3. We introduce a new *perceptual loss* to the image-to-image translation, which supervises the translation by the updated segmentation adaptation model.

5.2 Related Work

5.2.1 Domain Adaptation

When transferring knowledge from virtual images to real photos, it is often the case that there exists some discrepancy from the training to the test stage. Domain adaptation aims to rectify this mismatch and tune the models toward better generalization at testing [120]. The existing work on domain adaptation have largely focused on image classification [121]. A lot of work aims to learn domain-invariant representations through minimizing the domain distribution discrepancy. Maximum Mean Discrepancy (MMD) loss [122], computing the mean of representations, is a common distance metric between two domains. As the extension to MMD, some statistics of feature distributions such as mean and covariance [30, 31] are used to match two different domains. Unfortunately, when the distribution is not Gaussian, solely matching mean and covariance is not enough to align the two different domains well.

Adversarial learning [26] recently becomes popular, and another kind of domain adaptation methods. It reduces the domain shift by forcing the features from different domains to fool the discriminator. [9] would be the pioneer work, which introduces an adversarial loss on top of the high-level features of the two domains with the classification loss for the source dataset and achieves a better performance than the statistical matching methods. Expect for adversarial loss, some work proposed some extra loss functions to further decrease the domain shift, such as reweighted function for each class [123], and disentangled representations for separated matching [113]. All of these methods work on simple and small classification datasets (e.g., MNIST [114] and SVHN [62]), and may have quite limited performance in more challenging tasks, like segmentation.

5.2.2 Domain Adaptation for Semantic Segmentation

Recently, more domain adaptation techniques are proposed for semantic segmentation models, since an enormous amount of labor-intensive work is required to annotate so many images that are needed to train high-quality segmentation networks. A possible solution to alleviate the human efforts is to train networks on virtual data which is labeled automatically. For example, GTA5 [111] and SYHTHIA [112] are two popular synthetic datasets of city streets with overlapped categories, similar views to the real datasets (e.g., CITYSCAPE [119], CamVid [124]). Domain adaptation can be used to align the synthetic and the real datasets.

The first work to introduce domain adaptation for semantic segmentation is [10], which does the global and local alignments between two domains in the feature level. Curriculum domain adaptation [116] estimates the global distribution and the labels for the superpixel, and then learns a segmentation model for the finer pixel. In [125], multiple discriminators are used for different level features to reduce domain discrepancy. In [126], foreground and background classes are separately treated for decreasing the domain shift respectively. All these methods target to directly align features between two domains. Unfortunately, the visual (e.g., appearance, scale, etc.) domain gap between synthetic and real data usually makes it difficult for the network to learn transferable knowledge.

Motivated by the recent progress of unpaired image-to-image translation work (e.g., CycleGAN [117], UNIT [127], MUNIT [128]), the mapping from virtual to realistic data is regarded as the image synthesis problem. It can help reduce the domain discrepancy before training the segmentation models. Based on the translated results, Cycada [29] and DCAN [118] further align features between two domains in feature level. By separately reducing the domain shift in learning, these approaches obtained the state-of-the-art performance. However, the performance is limited by the quality of image-to-image translation. Once it fails, nothing can be done in the following step. To address this problem, we introduce a bidirectional learning framework where both translation and segmentation adaption models can promote each other in a

closed loop.

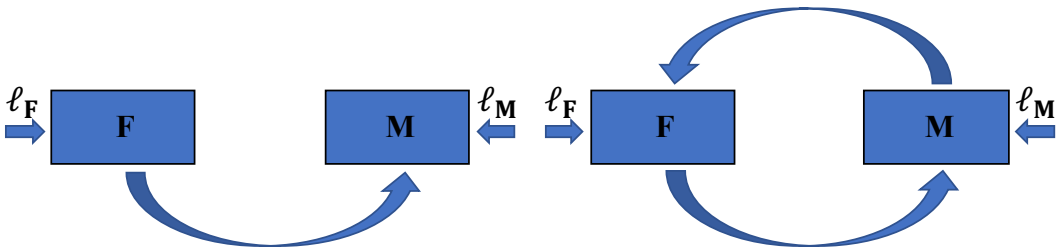
There are two most related work. In [129], the segmentation model is also used to improve the image translation, but not to adapt the source domain to the target domain since it is only trained on source data. [11] also proposed a self-training method for training the segmentation model iteratively. However, the segmentation model is only trained on source data and uses none of image translation techniques.

5.2.3 Bidirectional Learning

The kind of techniques were first proposed to solve the neural machine translation problem, such as [130, 131], which train a language translation model for both directions of a language pair. It improves the performance compared with the uni-direction learning and reduces the dependency on large amount of data. Bidirectional learning techniques were also extended to image generation problem [132], which trains a single network for both classification and image generation problem from both top-to-down and down-to-top directions. A more related work [133] proposed bidirectional image translation (i.e., source-to-target, and target-to-source), then trained two classifiers on both domains respectively and finally fuses the classification results. By contrast, our bidirectional learning refers to translation boosting the performance of segmentation and vise verse. The proposed method is used to deal with the semantic segmentation task.

5.3 Method

Given the source dataset $\mathcal{S}$ with segmentation labels $Y_{\mathcal{S}}$ (e.g., synthetic data generated by computer graphics) and the target dataset $\mathcal{T}$ with no labels (i.e., real data), we want to train a network for semantic segmentation, which is finally tested on the target dataset $\mathcal{T}$. Our goal is to make its performance to be as close as possible to the model trained on $\mathcal{T}$ with ground truth



(a): Sequential Learning (b):Bidirectional Learning

Figure 5.1: Sequential Learning vs Bidirectional Learning

labels $Y_{\mathcal{T}}$. The task is unsupervised domain adaptation for semantic segmentation. The task is not easy since the visual (e.g., lighting, scale, object textures, etc.) domain gap between $\mathcal{S}$ and $\mathcal{T}$ makes it difficult for the network to learn transferable knowledge at once.

To address this problem, the recent work [29] proposed two separated subnetworks. One is image-to-image translation subnetwork **F** which learn to translate an image from $\mathcal{S}$ to $\mathcal{T}$ in absence of paired examples. The another is segmentation adaptation subnetwork **M** that is trained on translated results $\mathbf{F}(\mathcal{S})$, which have the same labels $Y_{\mathcal{S}}$ to $\mathcal{S}$, and the target images $\mathcal{T}$ with no labels. Both subnetworks are learnt in a sequential way shown in Figure 5.1(a). Such a two-stage solution has two advantages: 1) **F** helps decrease the visual domain gap; 2) when domain gap is reduced, **M** is easy to learn, causing better performance. However, the solution has some limitations. Once **F** is learnt, it is fixed. There is no feedback from **M** to boost the performance of **F**. Besides, one-trial learning for **M** seems to just learn limited transferable knowledge.

In this section, we propose a new learning framework which can address the above two issues well. We inherit the way of separated subnetworks, but employ a *bidirectional learning* instead (in Section 5.3.1), which uses a closed-loop to iteratively update both **F** and **M**. Furthermore, we introduce a *self-training* to allow **M** being self-motivated in training (in Section 5.3.2). The network architecture and loss functions are presented in Section 5.3.3.

5.3.1 Bidirectional Learning

Our learning consists of two directions shown in Figure 5.1(b).

The forward direction (i.e., $\mathbf{F} \rightarrow \mathbf{M}$) is similar to the behavior of previous sequential learning [29]. We first train the image-to-image translation model $\mathbf{F}$ using images from $\mathcal{T}$ and $\mathcal{S}$. Then, we get the translated results $\mathcal{S}' = \mathbf{F}(\mathcal{S})$. Note that $\mathbf{F}$ won't change the labels of $\mathcal{S}'$, which are the same to $Y_{\mathcal{S}}$ (labels of $\mathcal{S}$). Next, we train the segmentation adaptation model $\mathbf{M}$ using $\mathcal{S}'$ with $Y_{\mathcal{S}}$ and $\mathcal{T}$. The loss function to learn $\mathbf{M}$ can be defined as:

$$\ell_{\mathbf{M}} = \lambda_{adv}\ell_{adv}(\mathbf{M}(\mathcal{S}'),\mathbf{M}(\mathcal{T})) + \ell_{seg}(\mathbf{M}(\mathcal{S}'),Y_{\mathcal{S}}), \tag{5.1}$$

where ℓ_{adv} is adversarial loss that enforces the distance between the feature representations of $\mathcal{S}'$ and the feature representations of $\mathcal{T}$ (obtained after $\mathcal{S}'$, $\mathcal{T}$ are fed into $\mathbf{M}$) as small as possible. ℓ_{seg} measures the loss of semantic segmentation. Since only $\mathcal{S}'$ have the labels, we solely measure the accuracy for the translated source images $\mathcal{S}'$.

The backward direction (i.e., $\mathbf{M} \rightarrow \mathbf{F}$) is newly added. The motivation is to promote $\mathbf{F}$ using updated $\mathbf{M}$. In [113, 128], a perceptual loss, which measures the distance of features obtained from a pre-trained network on object recognition, is used in the image translation network to improve the quality of translated result. Here, we use $\mathbf{M}$ to compute features for measuring the perceptual loss. By adding the other two losses: GAN loss and image reconstruction loss, the loss function for learning $\mathbf{F}$ can be defined as:

$$\begin{aligned}\ell_{\mathbf{F}} = {} & \lambda_{GAN}[\ell_{GAN}(\mathcal{S}',\mathcal{T}) + \ell_{GAN}(\mathcal{S},\mathcal{T}')] \\ & + \lambda_{recon}[\ell_{recon}(\mathcal{S},\mathbf{F}^{-1}(\mathcal{S}')) + \ell_{recon}(\mathcal{T},\mathbf{F}(\mathcal{T}')] \\ & + \ell_{per}(\mathbf{M}(\mathcal{S}),\mathbf{M}(\mathcal{S}')) + \ell_{per}(\mathbf{M}(\mathcal{T}),\mathbf{M}(\mathcal{T}'),\end{aligned} \tag{5.2}$$

where three losses are computed symmetrically, i.e., $\mathcal{S} \rightarrow \mathcal{T}$ and $\mathcal{T} \rightarrow \mathcal{S}$, to ensure the image-to-image translation consistent. The GAN loss ℓ_{GAN} enforces two distributions between $\mathcal{S}'$

and $\mathcal{T}$ similar to each other. $\mathcal{T}' = \mathbf{F}^{-1}(\mathcal{T})$, where $\mathbf{F}^{-1}$ is the reverse function of $\mathbf{F}$ that maps the image from $\mathcal{T}$ to $\mathcal{S}$. The loss ℓ_{recon} measures the reconstruction error when the image from $\mathcal{S}'$ is translated back to $\mathcal{S}$. ℓ_{per} is the perceptual loss that we propose to maintain the semantic consistency between $\mathcal{S}$ and $\mathcal{S}'$ or between $\mathcal{T}$ and $\mathcal{T}'$. That is, once we obtained an ideal segmentation adaptation model $\mathbf{M}$, whether $\mathcal{S}$ and $\mathcal{S}'$, or $\mathcal{T}$ and $\mathcal{T}'$ should have the same labels, even although there is the visual gap between $\mathcal{S}$ and $\mathcal{S}'$, or between $\mathcal{T}$ and $\mathcal{T}'$.

5.3.2 Self-Training

In the forward direction (i.e., $\mathbf{F} \rightarrow \mathbf{M}$), if the label is available for both the source domain $\mathcal{S}$ and the target domain $\mathcal{T}$, the fully supervised segmentation loss ℓ_{seg} is always the best choice to reduce the domain discrepancy. But in our case, the label for the target dataset is missing. As we known, self-training (ST) has been used in semi-supervised learning before, especially when the labels of dataset are insufficient or noisy. Here, we use ST to help promote the segmentation adaptation model $\mathbf{M}$.

Based on the prediction probability of $\mathcal{T}$, we can obtain some pseudo labels $\widehat{Y}_{\mathcal{T}}$ with high confidence. Once we have the pseudo labels, the corresponding pixels can be aligned directly with $\mathcal{S}$ according to the segmentation loss. Thus, we modify the overall loss function used to learn $\mathbf{M}$ (in Equation 5.1) as:

$$\begin{aligned} \ell_{\mathbf{M}} = & \lambda_{adv}\ell_{adv}(\mathbf{M}(\mathcal{S}'),\mathbf{M}(\mathcal{T})) \\ & + \ell_{seg}(\mathbf{M}(\mathcal{S}'),Y_{\mathcal{S}}) + \ell_{seg}(\mathbf{M}(\mathcal{T}_{st}),\widehat{Y}_{\mathcal{T}}), \end{aligned} \tag{5.3}$$

where $\mathcal{T}_{st} \subset \mathcal{T}$ is a subset of the target dataset in which the pixels have the pseudo labels $\widehat{Y}_{\mathcal{T}}$. It can be empty at the beginning. When a better segmentation adaptation model $\mathbf{M}$ is achieved, we can use $\mathbf{M}$ to predict more high-confident labels for $\mathcal{T}$, causing the size of $\mathcal{T}_{st}$ to grow. The recent work [11] also use ST for segmentation adaptation. By contrast, ST used in our work is combined

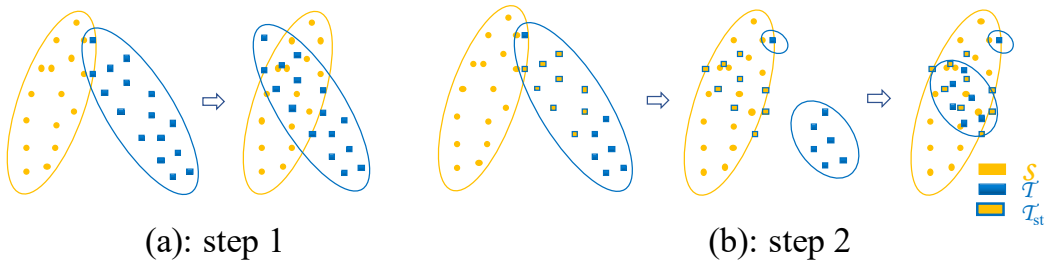


Figure 5.2: Self-training process. (a) Step 1: adversarial learning only. This process will leave some samples in the target domain unaligned with source domain. (b) Step 2: self-training with adversarial learning. Aided by adversarial learning, self-training achieves a better performance on sample alignments

with adversarial learning, which can work much better for the segmentation adaptation model.

We use the illustration (shown in Figure 5.2) to explain the principle of this process. When we learn the segmentation adaptation model for the first time, $\mathcal{T}_{st}$ is empty and the domain gap between $\mathcal{S}$ and $\mathcal{T}$ can be reduced with the loss shown in Equation 5.1. This process is shown in Figure 5.2 (a). Then we pick up the points in the target domain $\mathcal{T}$ that have been well aligned with $\mathcal{S}$ to construct the subset $\mathcal{T}_{st}$. In the second step, we can easily shift $\mathcal{T}_{st}$ to $\mathcal{S}$ and keep them being aligned with the help of the segmentation loss provided by the pseudo labels. This process is shown in the middle of Figure 5.2 (b). Therefore, the amount of data in $\mathcal{T}$ that needs to be aligned with $\mathcal{S}$ is decreased. We can continue to shift the remaining data to $\mathcal{S}$ same as step 1, as shown the right side of Figure 5.2 (b). It worth noting that ST helps adversarial learning process focus on the rest data that is not fully aligned at each step, since ℓ_{adv} can hardly change the data from $\mathcal{S}$ and $\mathcal{T}_{st}$ that has been aligned well.

5.3.3 Network and Loss Function

In this section, we introduce the network architecture (shown in Figure 5.3), details of loss functions and the training process (shown in Algorithm 2). The network is mainly composed with two components – the image translation model and segmentation adaptation model.

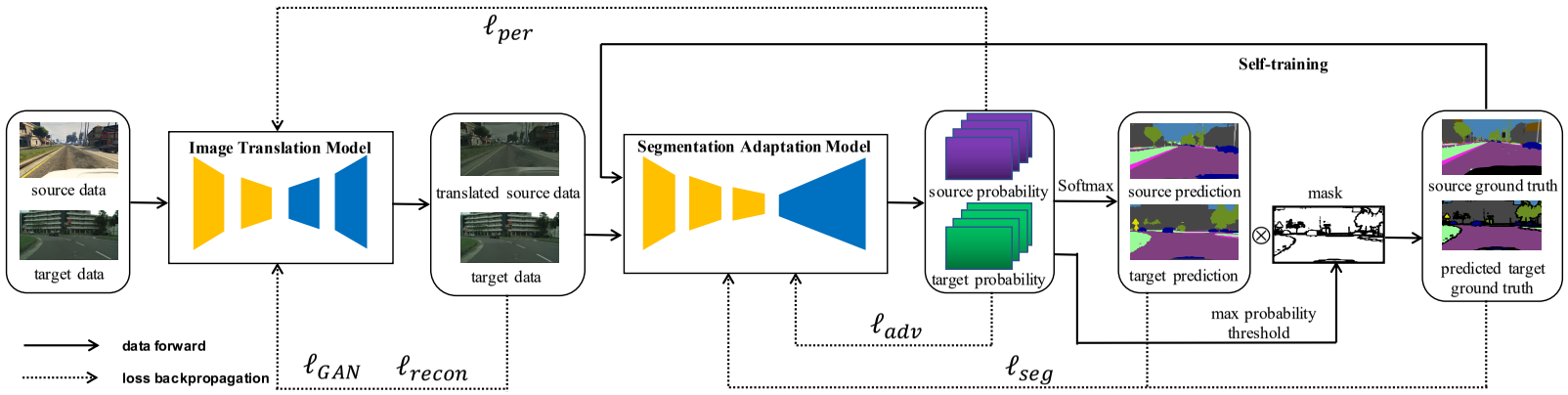


Figure 5.3: Network architecture and loss function. The framework is composed with an image translation model supervised with a GAN based loss ℓ_{GAN}, a reconstruction loss ℓ_{recon} and a perceptual loss ℓ_{per}, and an adaptive segmentation model learned by a adversarial loss ℓ_{adv} aided with self-training.

Algorithm 2: Training process of the BDL framework

Input: $(\mathcal{S}, Y_{\mathcal{S}})$, $(\mathcal{T}, \mathcal{T}_{st} = \emptyset)$, $\mathbf{M}^{(\mathbf{0})}$.
Output: $\mathbf{M}_N^{(K)}(\mathbf{F}^{(K)})$.

```
for k ← 1 to K do
    train F^(k) with Equation 5.2
    train M_0^(k) with Equation 5.1
    for i ← 1 to N do
        update T_st with M_{i-1}^(k)
        train M_i^(k) again with Equation 5.3
    end
end
```

While the translation model is learned, the loss ℓ_{GAN} and loss ℓ_{recon} (shown in Figure 5.3 and Equation 5.2) can be defined as:

$$\ell_{GAN}(\mathcal{S}', \mathcal{T}) = \mathbb{E}_{I_{\mathcal{T}} \sim \mathcal{T}}[D_{\mathbf{F}}(I_{\mathcal{T}})] + \mathbb{E}_{I_{\mathcal{S}} \sim \mathcal{S}}[1 - D_{\mathbf{F}}((I'_{\mathcal{S}}))], \tag{5.4}$$

$$\ell_{recon}(\mathcal{S}, \mathbf{F^{-1}}(\mathcal{S}')) = \mathbb{E}_{I_{\mathcal{S}} \sim \mathcal{S}}[||\mathbf{F^{-1}}((I'_{\mathcal{S}})) - I_{\mathcal{S}}||_1], \tag{5.5}$$

where $I_{\mathcal{S}}$ and $I_{\mathcal{T}}$ are the input images from source and target dataset. $I'_{\mathcal{S}}$ is the translated image given by $\mathbf{F}$. $D_{\mathbf{F}}$ is the discriminator added to reduce the difference between $I_{\mathcal{T}}$ and $I'_{\mathcal{S}}$. For the reconstruction loss, L_1 norm is used to keep the cycle consistency between $I_{\mathcal{S}}$ and $\mathbf{F}^{-1}(I'_{\mathcal{S}})$ when $\mathbf{F^{-1}}$ is the reverse function of $\mathbf{F}$. Here, we only show two losses for one direction, and $\ell_{GAN}(\mathcal{S}, \mathcal{T}'), \ell_{recon}(\mathcal{T}, \mathbf{F}(\mathcal{T}'))$ can be defined similarly.

As shown in Figure 5.3, the perceptual loss ℓ_{per} connects the translation model and segmentation adaptation model. When we learn the perceptual loss ℓ_{per} for the translation model, instead of only keeping the semantic consistency between $I_{\mathcal{S}}$ and its translated result $I'_{\mathcal{S}}$, we add another term weighted by λ_{per_recon}, to keep the semantic consistency between $I_{\mathcal{S}}$ and its corresponding reconstruction $\mathbf{F}^{-1}(I'_{\mathcal{S}})$. With the new term, the translation model can be more

stable especially for the reconstruction part. ℓ_{per} is defined as:

$$\begin{aligned}\ell_{per}(\mathbf{M}(\mathcal{S}),\mathbf{M}(\mathcal{S}')) = \lambda_{per}\mathbb{E}_{I_\mathcal{S}\sim\mathcal{S}}||\mathbf{M}(I_\mathcal{S}) - \mathbf{M}((I'_\mathcal{S}))||_1 + \\ \lambda_{per\_recon}\mathbb{E}_{I_\mathcal{S}\sim\mathcal{S}}[||\mathbf{M}(\mathbf{F}^{-1}((I'_\mathcal{S}))) - \mathbf{M}(I_\mathcal{S})||_1]\end{aligned} \tag{5.6}$$

Due to the symmetry, $\ell_{per}(\mathbf{M}(\mathcal{T}),\mathbf{M}(\mathcal{T}'))$ (shown in Equation 5.2) can be defined similarly.

When the segmentation adaptation model is trained, it requires the adversarial learning with the loss ℓ_{adv} and the self-training with the loss ℓ_{seg} (shown in Equation 5.3). For adversarial learning, we add a discriminator $D_\mathbf{M}$ to decrease the difference between the source and target probabilities shown in Figure 5.3. ℓ_{adv} can be defined as:

$$\begin{aligned}\ell_{adv}(\mathbf{M}(\mathcal{S}'),\mathbf{M}(\mathcal{T})) = \mathbb{E}_{I_\mathcal{T}\sim\mathcal{T}}[D_\mathbf{M}(\mathbf{M}(I_\mathcal{T}))] \\ + \mathbb{E}_{I_\mathcal{S}\sim\mathcal{S}}[1 - D_\mathbf{M}(\mathbf{M}(I'_\mathcal{S}))].\end{aligned} \tag{5.7}$$

The segmentation loss ℓ_{seg} uses the cross-entropy loss. For the source image $I_\mathcal{S}$, ℓ_{seg} can be defined as:

$$\ell_{seg}(\mathbf{M}(\mathcal{S}'),Y_\mathcal{S}) = -\frac{1}{HW}\sum_{H,W}\sum_{c=1}^{C}\mathbb{1}_{[c=y_\mathcal{S}^{hw}]}\log P_\mathcal{S}^{hwc}, \tag{5.8}$$

where $y_\mathcal{S}$ is the label map for $I_\mathcal{S}$, C is the number of classes, H and W are the height and width of the output probability map. $P_\mathcal{S}$ is the source probability of the segmentation adaptation model which can be defined as $P_\mathcal{S} = \mathbf{M}(I'_\mathcal{S})$. For the target image $I_\mathcal{T}$, we need to define how to choose the pseudo label map $\widehat{y}_\mathcal{T}$ for it. We choose to use a common method we call as "max probability threshold(MPT)" to filter the pixels with high prediction confidence in $I_\mathcal{T}$. Thus we can define $\widehat{y}_\mathcal{T}$ as $\widehat{y}_\mathcal{T} = \text{argmax}\,\mathbf{M}(I_\mathcal{T})$ and the mask map for $\widehat{y}_\mathcal{T}$ as $m_\mathcal{T} = \mathbb{1}_{[\text{argmax}\,\mathbf{M}(I_\mathcal{T})>\text{threshold}]}$. Thus the segmentation loss for $I_\mathcal{T}$ can be expressed as:

$$\ell_{seg}(\mathbf{M}(\mathcal{T}_{st}),\widehat{Y}_\mathcal{T}) = -\frac{1}{HW}\sum_{H,W}m_\mathcal{T}^{hw}\sum_{c=1}^{C}\mathbb{1}_{[c=y_\mathcal{T}^{hw}]}\log P_\mathcal{T}^{hwc}, \tag{5.9}$$

where $P_{\mathcal{T}}$ is the target output of $\mathbf{M}$.

We present the training processing in Algorithm 2. The training process consists of two loops. The outer loop is mainly to learn the translation model and the segmentation adaptation model through the forward direction and the backward direction. The inner loop is mainly used to implement the ST process. In the following section, we will introduce how to choose the number of iteration for learning $\mathbf{F}$, $\mathbf{M}$, and how to estimate the MPT for ST.

5.4 Ablation

To know the effectiveness of bidirectional learning and self-training for improving $\mathbf{M}$, we conduct some ablation studies. We use GTA5 [111] as the source dataset and Cityscapes [119] as the target dataset. The translation model is CycleGAN [117] and the segmentation adaptation model is DeepLab V2 [134] with the backbone ResNet101 [55]. All the following experiments use the same model, unless it is specified.

Here, we first provide the description of notations used in the following ablation study and tables. $\mathbf{M}^{(0)}$ is the initial model to start the bidirectional learning and is trained only with source data. $\mathbf{M}^{(1)}$ is trained with source and target data with adversarial learning. For $\mathbf{M}^{(0)}(\mathbf{F}^{(1)})$, a translation model $\mathbf{F}^{(1)}$ is used to translate the source data and then a segmentation model $\mathbf{M}^{(0)}$ is learned based on the translated source data. $\mathbf{M}_i^{(k)}(\mathbf{F}^{(k)})$ for $k = 1,2$ and $i = 0,1,2$ refers to the model of k-th iteration for the outer loop and i-th iteration for the inner loop in Algorithm 2.

5.4.1 Bidirectional Learning

We show the results obtained by the model trained in a bidirectional learning system without ST. In Table 5.1, $\mathbf{M}^{(0)}$ is our baseline model that gives the lowerbound for mIoU. We find a similar performance between the model $\mathbf{M}^{(1)}$ and $\mathbf{M}^{(0)}(\mathbf{F}^{(1)})$ both of which achieve more than 7% improvement compared to $\mathbf{M}^{(0)}$ and about 1.6% further improvement is given by $\mathbf{M}^{(1)}(\mathbf{F}^{(1)})$.

It means segmentation adaptation model and the translation model can work independently and when combined together which is basically one iteration of the bidirectional learning they can be complementary to each other. We further show that through continue training the bidirectional learning system, in which case $\mathbf{M}^{(1)}(\mathbf{F}^{(1)})$ is used to replace $\mathbf{M}^{(0)}$ for the backward direction, a better performance can be given by the new model $\mathbf{M}_0^{(2)}(\mathbf{F}^{(2)})$.

Table 5.1: Performance of bidirectinal learning

GTA5 $\rightarrow$ Cityscapes	
model	mIoU
$\mathbf{M}^{(0)}$	33.6
$\mathbf{M}^{(1)}$	40.9
$\mathbf{M}^{(0)}(\mathbf{F}^{(1)})$	41.1
$\mathbf{M}_0^{(1)}(\mathbf{F}^{(1)})$	42.7
$\mathbf{M}_0^{(2)}(\mathbf{F}^{(2)})$	43.3

5.4.2 Bidirectional Learning with Self-Training

In this section, we show how the ST can further improve the ability of segmentation adaption model and in return influence the bidirectional learning process. In Table 5.2, we show results given by two iterations($k = 1,2$) based on Algorithm 2. In Figure 5.4, we show the segmentation results and the corresponding mask map given by the max probability threshold (MPT) which is 0.9. In Figure 5.4, the white pixels are the ones with prediction confidence higher than MPT and the black pixels are the low confident pixels.

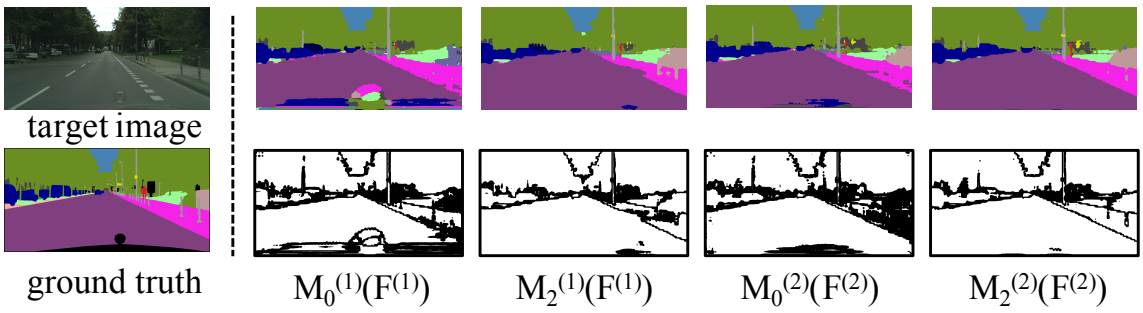


Figure 5.4: Visualization of segmentation results for each step in bidirectional learning

While $k = 1$, when model $\mathbf{M}_0^{(1)}(\mathbf{F}^{(1)})$ is updated to $\mathbf{M}_2^{(1)}(\mathbf{F}^{(1)})$ with self-training, the mIoU can be improved by 4.5%. We can find for each category when the IoU is below 50, a big improvement can be obtained from $\mathbf{M}_0^{(1)}(\mathbf{F}^{(1)})$ to $\mathbf{M}_2^{(1)}(\mathbf{F}^{(1)})$. It can prove our previous analysis in section 5.3.2 that with self-training the well aligned data from source and target domain can be kept and the rest data can be further aligned through the adversarial learning process.

While $k = 2$, we first replace $\mathbf{M}^{(0)}$ with $\mathbf{M}_2^{(1)}(\mathbf{F}^{(1)})$ to start the backward direction. Without self-training the mIoU is 44.3 which is a larger improvement (44.3% vs. 42.7%) compared to the results (43.3% vs. 42.7%) shown in Table 5.1. It can further prove our discussion in section 5.4.1 about the important role played by the segmentation adaptation model in the backward direction. Furthermore, we can find from Table 5.2, although in the beginning of the second iteration the mIoU drops from 47.2 to 44.3, while the self-training technique is induced, the mIoU can be promoted to 48.5 which outperforms the results in the first iteration. From the segmentation results shown in Figure 5.4, we can obverse that the segmentation maps given by $\mathbf{M}_2^{(1)}(\mathbf{F}^{(1)})$ and $\mathbf{M}_2^{(2)}(\mathbf{F}^{(2)})$ are better than those generated by $\mathbf{M}_0^{(1)}(\mathbf{F}^{(1)})$ and $\mathbf{M}_0^{(2)}(\mathbf{F}^{(2)})$ respectively which further confirms the effectiveness of self-training. By comparing $\mathbf{M}_2^{(2)}(\mathbf{F}^{(2)})$ to $\mathbf{M}_2^{(1)}(\mathbf{F}^{(1)})$, we can also find some improvements, e.g., the smoother segmentation prediction given by $\mathbf{M}_2^{(2)}(\mathbf{F}^{(2)})$ on the sidewalk and it demonstrates the benefits brought by the proposed bidirectional learning framework. Besides, as we improve the segmentation performance, the segmentation adaptation model can give more confident prediction which can be observed by the increasing white area in the mask map. It gives us the motivation to use the mask map to choose the threshold and number of iterations for the self-training process in Algorithm 2.

Table 5.2: Performance of bidirectional learning with self-training

		GTA5 → Cityscapes																			
		road	sidewalk	building	wall	fence	pole	t-light	t-sign	vegetation	terrain	sky	person	rider	car	truck	bus	train	motorbike	bicycle	mIoU
	$\mathbf{M}^{(0)}$	69.0	12.7	69.5	9.9	19.5	22.8	31.7	15.3	73.9	11.3	67.2	54.7	23.9	53.4	29.7	4.6	11.6	26.1	32.5	33.6
$k=1$	$\mathbf{M}_0^{(1)}(\mathbf{F}^{(1)})$	89.1	42.0	82.0	24.3	15.1	27.4	35.7	24.6	81.1	32.4	78.0	57.6	28.7	76.0	26.5	36.0	4.0	25.7	24.9	42.7
	$\mathbf{M}_1^{(1)}(\mathbf{F}^{(1)})$	91.2	47.8	84.0	34.8	28.9	31.7	37.7	36.0	84.0	40.4	76.6	57.9	25.3	80.4	31.2	41.7	2.8	27.2	32.4	46.8
	$\mathbf{M}_2^{(1)}(\mathbf{F}^{(1)})$	91.4	47.9	84.2	32.4	26.0	31.8	37.3	33.0	83.3	39.2	79.2	57.7	25.6	81.3	36.3	39.7	2.6	31.3	33.5	47.2
$k=2$	$\mathbf{M}_0^{(2)}(\mathbf{F}^{(2)})$	88.2	41.3	83.2	28.8	21.9	31.7	35.2	28.2	83.0	26.2	83.2	57.6	27.0	77.1	27.5	34.6	2.5	28.3	36.1	44.3
	$\mathbf{M}_1^{(2)}(\mathbf{F}^{(2)})$	91.2	46.1	83.9	31.6	20.6	29.9	36.4	31.9	85.0	39.7	84.7	57.5	29.6	83.1	38.8	46.9	2.5	27.5	38.2	47.6
	$\mathbf{M}_2^{(2)}(\mathbf{F}^{(2)})$	91.0	44.7	84.2	34.6	27.6	30.2	36.0	36.0	85.0	43.6	83.0	58.6	31.6	83.3	35.3	49.7	3.3	28.8	35.6	48.5

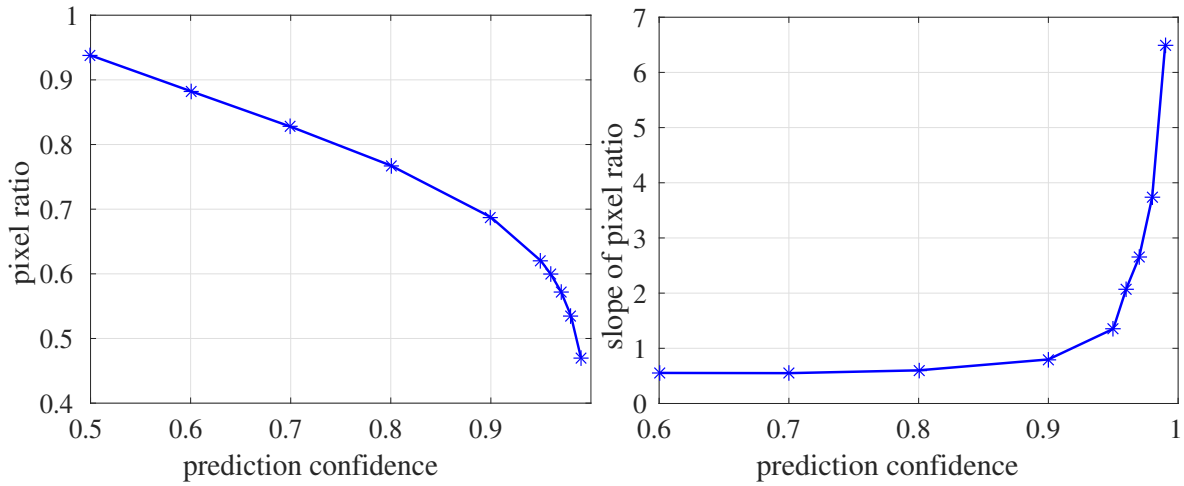


Figure 5.5: Relationship between pixel ratio and the prediction confidence

5.4.3 Hyper Parameter Learning

We will describe how to choose the threshold to filter out data with high confidence and the iteration number N in Algorithm 2.

When we choose the threshold, we have to balance between two folds. On one hand, we desire the predicted labels with high confidence as many as possible (presented as white areas in Figure 5.4). On the other hand, we want to avoid inducing too much noise caused by the incorrect prediction, namely, the threshold should be as high as possible. We present the relationship of the prediction confidence (maximum class probability of per pixel from $\mathbf{M}$) and the ratio between selected pixels and all pixels (i.e., percentage of all white areas shown in Figure 5.4) on the left side of Figure 5.5, then show the slope in the right side of Figure 5.5. We can find when the prediction confidence increases from 0.5 to 0.9, the ratio decreases almost linearly and the slope stays almost unchanged. But from 0.9 to 0.99, the ratio decreases much faster. Based on the observation, we choose the inflection point 0.9 as the threshold as the trade-off between the number and the quality of selected labels.

In order to further prove our choice, in Table 5.3-(a), we show segmentation results using different thresholds to the self-training of $\mathbf{M}_N^K$ when $K = 1$ and $N = 1$ in Algorithm 2. As another

Table 5.3: Influence of the threshold and number of iterations (N) on the self-training

GTA5 → Cityscapes		
model	threshold	mIoU
$\mathbf{M}_1^{(1)}(\mathbf{F}^{(1)})$	0.95	45.7
$\mathbf{M}_1^{(1)}(\mathbf{F}^{(1)})$	0.9	46.8
$\mathbf{M}_1^{(1)}(\mathbf{F}^{(1)})$	0.8	46.4
$\mathbf{M}_1^{(1)}(\mathbf{F}^{(1)})$	0.7	45.9
$\mathbf{M}_1^{(1)}(\mathbf{F}^{(1)})$	–	44.9

(a) Influence of threshold

GTA5 → Cityscapes		
model	pixel ratio	mIoU
$\mathbf{M}_0^{(1)}$	66%	40.9
$\mathbf{M}_0^{(1)}(\mathbf{F}^{(1)})$	69%	42.7
$\mathbf{M}_1^{(1)}(\mathbf{F}^{(1)})$	79%	46.8
$\mathbf{M}_2^{(1)}(\mathbf{F}^{(1)})$	81%	47.2
$\mathbf{M}_3^{(1)}(\mathbf{F}^{(1)})$	81%	47.1

(b) Influence of N

option, we also consider soft threshold instead of hard one, namely, every pixel being weighted by its maximum class probability. We show the result on the bottom row. All the results confirm our analysis. When the threshold is lower than 0.9, the uncorrected prediction becomes the key issue to influence the performance of ST. While we increase the threshold to 0.95, the ST process is more sensitive to the number of pixels that can be used. When we use soft threshold, the result is still worse. It is probably because an amount of labeling noise are involved and the bad impact cannot be well alleviated by assigning a lower weight to the noise label. Thus, 0.9 seems to be a good choice for the threshold in the following experiments.

For the iteration number N, we select a proper value according to the predicted labels as well. When N increases, the segmentation adaptation model becomes much stronger, causing more labels to be used for ST. Once the pixel ratio for ST stops increasing, it means that the learning for the segmentation adaptation model is converged and nearly no improved. We definitely increase the value of K to start another iteration. In Table 5.3-(b), we show some segmentation results with the theshold 0.9 as we increase the value of N. We can find the mIoU becomes better with the increasing of N. When $N = 2$ or 3, the mIoU almost stopped increasing, and the pixel ratio stay around the same. It may suggest that $N = 2$ is a good choice, and we use it in our work.

5.5 Experiments

In this section, we compare the results obtained between our method and the state-of-the-art methods.

5.5.1 Network Architecture

In our experiments, we choose to use DeepLab V2 [134] with ResNet101 [55] and FCN-8s [110] with VGG16 [63] as our segmentation model. They are initialized with the network pre-trained with ImageNet [4]. The discriminator we choose for segmentation adaptation model is similar to [135] which has 5 convolution layers with kernel 4×4 with channel numbers $\{64, 128, 256, 512, 1\}$ and stride of 2. For each convolutional layer except the last one, a leaky ReLU [136] parameterized by 0.2 is followed. For the image translation model, we follow the architecture of CycleGAN [117] with 9 blocks and add the segmentation adaptation model as the perceptual loss.

5.5.2 Implementation Details

When training CycleGAN [117], the image is randomly cropped to the size 452×452 and it is trained for 20 epochs. For the first 10 epochs, the learning rate is 0.0002 and decreases to 0 linearly after 10 epochs. We set $\lambda_{GAN} = 1$, $\lambda_{recon} = 10$ in Equation 5.3 and set $\lambda_{per} = 0.1$, $\lambda_{per\_recon} = 10$ for the perceptual loss. When training the segmentation adaptation model, images are resized with the long side to be $1,024$ and the ratio is kept. Different parameters are used for DeepLab V2 [134] and FCN-8s [110]. For DeepLab V2 with ResNet 101, we use SGD as the optimizer. The initial learning rate is 2.5×10^{-4} and decreased with 'poly' learning rate policy with power as 0.9. For FCN-8s with VGG16, we use Adam as the optimizer with momentum as 0.9 and 0.99. The initial learning rate is 1×10^{-5} and decreased with 'step' learning rate policy with step size as 5000 and $\gamma = 0.1$. For both DeepLab V2 and FCN-8s, we use the same discriminator that is trained with Adam optimizer with initial learning rate as 1×10^{-4}

for DeepLab V2 and 1×10^{-6} for FCN-8s. The momentum is set as 0.9 and 0.99. We set $\lambda_{adv} = 0.001$ for ResNet101 and 1×10^{-4} for FCN-8s in Equation 5.1.

5.5.3 Dataset

As we have mentioned before, two synthetic datasets – GTA5 [111] and SYNTHIA [112] are used as the source dataset and Cityscapes [119] is used as the target dataset. For GTA5 [111], it contains $24,966$ images with the resolution of 1914×1052 and we use the 19 common categories between GTA5 and Cityscapes dataset. For SYNTHIA [112], we use the SYNTHIA-RAND-CITYSCAPES set which contains $9,400$ images with the resolution 1280×760 and 16 common categories with Cityscapes [119]. For Cityscapes [119], it is splited into training set, validation set and testing set. The training set contains $2,975$ images with the resolution 2048×1024. We use the training set as the target dataset only. Since the ground truth labels for the testing set are missing, we have to use the validation set which contains 500 images as the testing set in our experiments.

5.5.4 Comparison with State-of-the-Art

We compare the results between our method and the state-of-the-art method with two different backbone networks: ResNet101 and VGG16 respectively. We perform the comparison on two tasks: "GTA5 to Cityscapes" and "SYNTHIA to Cityscapes". In Table 5.4, we present the adaptation result on the task "GTA5 to Cityscapes" with ResNet101 and VGG16. We can observe the role of backbone in all domain adaptation methods, namely ResNet101 achieves a much better result than VGG16. In [116, 125, 137], they mainly focus on feature-level alignment with different adversarial loss functions. But working only on the feature level is not enough, even though the best result [118] among them is still about 5% worse than our results. Cycada [29] (we run their codes with ResNet101) and DCAN [118] used the translation model followed by the

segmentation adaptation model to further reduce the visual domain gap, and both achieved very similar performance. Ours uses similar loss function compared to Cycada [29], but with a new proposed bidirectional learning method, 6% improvement can be achieved. CBST [11] proposed a self-training method, and further improved the performance with space prior information. For a fair comparison, we show the results that only use self-training. With VGG16, we can get 10.4% improvement. Therefore, we can find without bidirectional learning, the self-training method is not enough to achieve a good performance.

In Table 5.5, we present the adaptation result on the task "SYNTHIA to Cityscapes" for both ResNet101 and VGG16. The domain gap between SYNTHIA and Cityscapes is much larger than that of GTA5 and Cityscapes, and their categories are not fully overlapped. As the baseline results [125, 137] chosen for ResNet101 only use 13 categories, we also list results for the 13 categories for a fair comparison. We can find from Table 5.5, as the domain gap increases, the adaptation result for Cityscapes is much worse compared to the result in Table 5.4. For example, the category like 'road', 'sidewalk' and 'car' are more than 10% worse. And this problem will have a bad impact on the ST because of the lower prediction confidence. But we can still achieve at least 4% better than most of other results given by [116, 11, 118, 125].

5.5.5 Performance Gap to Upper-Bound.

We use the target dataset with ground truth labels to train a segmentation model, which shares the same backbone as we used, to get the upper-bound result. For "GTA5 to Cityscapes" with 19 categories, the upper bounds are 65.1 and 60.3 for ResNet101 and VGG16 respectively. For "SYNTHIA to Cityscapes" with 13 categories for ResNet101 and 16 categories for VGG16, the upper bounds are 71.7 and 59.5. For our method, although the performance gap is 16.6 at least, it has been reduced significantly compared to other methods. However, it means there is still big room to improve the performance.

Table 5.4: Comparison results from GTA5 to Cityscapes

GTA5 → Cityscapes																					
Oracle	Method	road	sidewalk	building	wall	fence	pole	t-light	t-sign	vegetation	terrain	sky	person	rider	car	truck	bus	train	motorbike	bicycle	mIoU
ResNet101[55] 65.1	Cycada[29]	86.7	35.6	80.1	19.8	17.5	**38.0**	**39.9**	**41.5**	82.7	27.9	73.6	**64.9**	19	65.0	12.0	28.6	4.5	31.1	**42.0**	42.7
	AdaptSegNet[125]	86.5	25.9	79.8	22.1	20.0	23.6	33.1	21.8	81.8	25.9	75.9	57.3	26.2	76.3	29.8	32.1	**7.2**	29.5	32.5	41.4
	DCAN[118]	85.0	30.8	81.3	25.8	21.2	22.2	25.4	26.6	83.4	36.7	76.2	58.9	24.9	80.7	29.5	42.9	2.50	26.9	11.6	41.7
	CLAN[137]	87.0	27.1	79.6	27.3	23.3	28.3	35.5	24.2	83.6	27.4	74.2	58.6	28.0	76.2	33.1	36.7	6.7	**31.9**	31.4	43.2
	Ours	**91.0**	**44.7**	**84.2**	**34.6**	**27.6**	30.2	36.0	36.0	**85.0**	**43.6**	**83.0**	58.6	**31.6**	**83.3**	**35.3**	**49.7**	3.3	28.8	35.6	**48.5**
VGG16[63] 60.3	Curriculum[116]	74.9	22.0	71.7	6.0	11.9	8.4	16.3	11.1	75.7	13.3	66.5	38.0	9.3	55.2	18.8	18.9	0.0	16.8	16.6	28.9
	CBST[11]	66.7	26.8	73.7	14.8	9.5	**28.3**	25.9	10.1	75.5	15.7	51.6	47.2	6.2	71.9	3.7	2.2	**5.4**	18.9	**32.4**	30.9
	Cycada[29]	85.2	37.2	76.5	21.8	15.0	23.8	22.9	**21.5**	80.5	31.3	60.7	50.5	9.0	76.9	17.1	28.2	4.5	9.8	0	35.4
	DCAN[118]	82.3	26.7	77.4	23.7	**20.5**	20.4	**30.3**	15.9	80.9	25.4	69.5	52.6	11.1	79.6	24.9	21.2	1.30	17.0	6.70	36.2
	CLAN[137]	88.0	30.6	79.2	23.4	**20.5**	26.1	23.0	14.8	81.6	34.5	72.0	45.8	7.9	80.5	**26.6**	**29.9**	0.0	10.7	0.0	36.6
	Ours	**89.2**	**40.9**	**81.2**	**29.1**	19.2	14.2	29.0	19.6	**83.7**	**35.9**	**80.7**	**54.7**	**23.3**	**82.7**	25.8	28.0	2.3	**25.7**	19.9	**41.3**

Table 5.5: Comparison results from SYNTHIA to Cityscapes

SYNTHIA → Cityscapes																			
Oracle	Method	road	sidewalk	building	wall	fence	pole	t-light	t-sign	vegetation	sky	person	rider	car	bus	motorbike	bicycle	mIoU
ResNet101[55] 71.7	AdaptSegNet[125]	79.2	37.2	78.8	-	-	-	9.9	10.5	78.2	80.5	53.5	19.6	67.0	29.5	21.6	31.3	45.9
	CLAN[137]	81.3	37.0	80.1	-	-	-	**16.1**	**13.7**	78.2	**81.5**	53.4	21.2	73.0	32.9	22.6	30.7	47.8
	Ours	**86.0**	**46.7**	**80.3**	-	-	-	14.1	11.6	**79.2**	81.3	**54.1**	**27.9**	**73.7**	**42.2**	**25.7**	**45.3**	**51.4**
VGG16[63] 59.5	FCN wild[10]	11.5	19.6	30.8	4.4	0.0	20.3	0.1	11.7	42.3	68.7	51.2	3.8	54.0	3.2	0.2	0.6	20.2
	Curriculum[116]	65.2	26.1	**74.9**	0.1	0.5	10.7	3.5	3.0	76.1	70.6	47.1	8.2	43.2	20.7	0.7	13.1	29.0
	CBST[11]	69.6	28.7	69.5	**12.1**	0.1	**25.4**	**11.9**	13.6	**82.0**	**81.9**	49.1	14.5	66.0	6.6	3.7	32.4	35.4
	DCAN[118]	79.9	**30.4**	70.8	1.6	**0.6**	22.3	6.7	23.0	76.9	73.9	41.9	16.7	61.7	11.5	**10.3**	38.6	35.4
	Ours	**72.0**	30.3	74.5	0.1	0.3	24.6	10.2	**25.2**	80.5	80.0	**54.7**	**23.2**	**72.7**	**24.0**	7.5	**44.9**	**39.0**

5.6 Conclusion

In this chapter, we proposed a bidirectional learning method with self-training for segmentation adaptation problem. Extensive experiments show that the adaptation performance can be promoted when the model is trained in a bidirectional manner. The BDL framework demonstrates that the image translation model, segmentation model with feature adversarial alignment and self-training are synergetic. We hope this work can shed lights on future research of domain adaptation and in the next chapter we will further show the power of adversarial learning and self-training on multi-source domain adaptation aided by dynamic networks.

Chapter 5 is, in full, based on the material as it appears in the publication of "Bidirectional Learning for Domain Adaptation of Semantic Segmentation", Yunsheng Li, Lu Yuan and Nuno Vasconcelos, in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 2019. The dissertation author was the primary investigator and author of this material.

Chapter 6

Dynamic Transfer for Multi-Source Domain Adaptation

6.1 Introduction

The bidirectional learning framework shows tremendous improvement on the domain adaptation task. Moreover, it reveals the adversarial learning and self-training can boost to each other. Based on this finding, in this chapter, we further explore a novel domain adaptation problem, i.e., multi-source domain adaptation and discuss the application of dynamic networks on this new research subject.

Multi-source domain adaptation addresses the adaptation from multiple source domains to a target domain. It is challenging because a clear domain discrepancy exists not only between source and target domains, but also among multiple source domains (see exemplar images in Figure 6.2). This suggests that successful adaptation requires significant *elasticity* of the model to adapt. A nature way to achieve this elasticity is to make model dynamic, i.e., the mapping implemented by the model should vary with the input sample.

This hypothesis has not been explored by existing work, e.g. [138, 139], which instead aims to learn a domain agnostic model $f_{\boldsymbol{\theta}_c}$, of *static* parameters $\boldsymbol{\theta}_c$, that works well for all source $\{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_N\}$ and target $\mathcal{T}$ domains. We refer to this approach as *static transfer.* As illustrated in Figure 6.1 (a), the model implements a fixed mapping across all domains. However, learning a domain agnostic model is difficult, since different domains can give rise to very different image distributions. When forcing a model to be domain agnostic, it essentially averages the domain conflict. Thus the performance drops on each source domain. This is validated by our preliminary study. As shown in Figure 6.2, compared to the optimal model per domain, the static transfer model consistently degrades in each source domain.

In this chapter, we propose *dynamic transfer* to address this issue. As shown in Figure 6.1(b), it contains a parameter predictor that changes the model parameters on a per-sample basis, i.e., implements mapping $f_{\boldsymbol{\theta}(\boldsymbol{x})}$. It has the advantage of not requiring the definition of domains or the collection of domain labels. In fact, it unifies the problems of single-source and

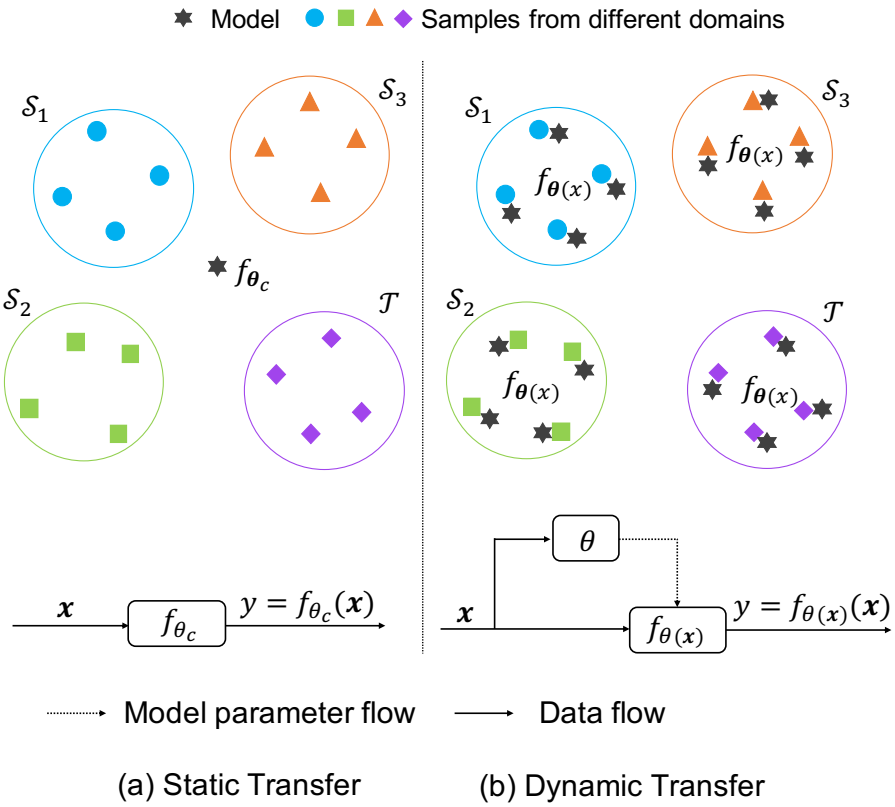


Figure 6.1: **Static Transfer vs. Dynamic Transfer**. (a) 'Static Transfer' implements domain adaptation via a static model $f_{\boldsymbol{\theta}_c}$ with fixed parameters $\boldsymbol{\theta}_c$. (b) 'Dynamic Transfer' ($f_{\boldsymbol{\theta}(\boldsymbol{x})}$) adapts the model parameters $\boldsymbol{\theta}(\boldsymbol{x})$ according to samples, which generates a different model per sample.

multi-source domain adaptation. By breaking down source domain barriers, it turns multiple source domain adaptation into a single-source domain problem. The only difference is the complexity of this domain.

The key insight is that adapting model according to domains is achieved statistically by adapting model per sample, since each domain is viewed as a distribution of image samples. The dynamic transfer learns how to adapt the model's parameters and fit to the union of source domains. Thus the alignment between source domains and target domain is significantly simplified, as it is no longer necessary to pull all source domains together with the target domain. In this case, as long as the target domain is aligned to any part of the source domains, the model can be easily adapted to the target samples.

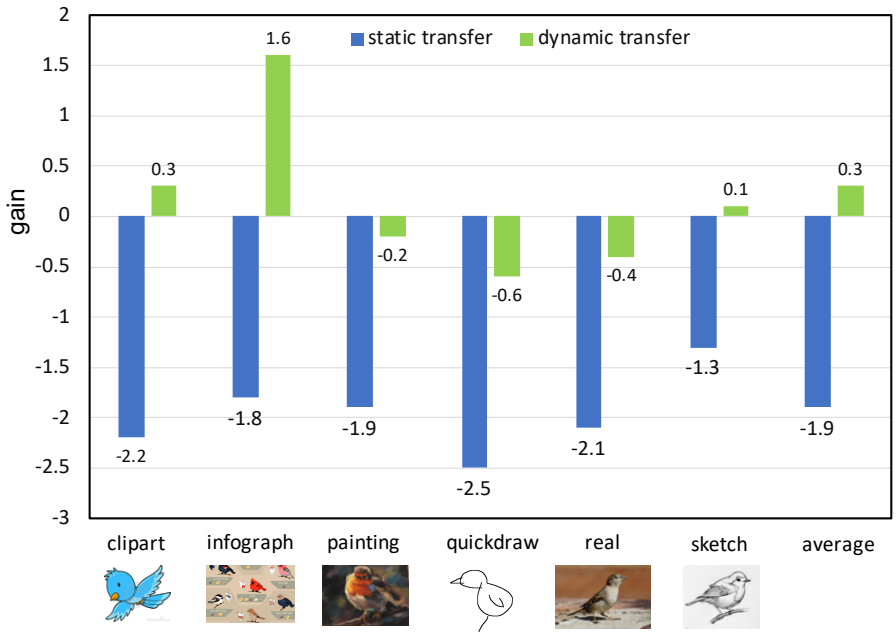


Figure 6.2: **Static Transfer vs. Dynamic Transfer on the performance degradation of source domains** compared to the oracle results. Both transfer models are tested across source domains.

When compared to the domain adaption literature, dynamic transfer introduces a significant paradigm shift. In the literature, most works assume the static network of Figure 6.1(a) and focus on loss functions. The goal is to define losses that somehow "pull all the domains" together into a shared latent representation. The problem is that the domains are usually very different at the network input. Hence, the force introduced by the loss at the output, to bring them together, is counter-balanced by an input force to keep them apart. This usually leads to a difficult optimization and compromises adaptation performance. The introduction of a dynamic network, as in Figure 6.1(b), enables a more elastic mapping. In this case, it is not necessary to pull all domains together. The model adaptation given by the dynamic transfer can be generalized to target domain easily when the target domain is shifted to the space formed by entire source domain. In this way, dynamic transfer shifts the focus of the domain adaptation problem from the design of good *loss functions* to the design of good *network architectures* for dynamic transfer.

An immediate difficulty is that the architecture of Figure 6.1(b) can be very hard to train,

since the parameter predictor cannot generate all parameters for a large model. The question is whether it is possible to perform the model adaptation by only modifying a small subset of parameters on a sample basis. In this chapter, we show that this is indeed possible by the addition of dynamic residuals to the convolution kernels of a static network. Since the residual blocks can be much smaller than the static ones, this has both very low additional computational cost (less than 0.1%) to aggregate dynamic residuals with static kernel and little tendency to overfit. However, it is shown to significantly enhance the domain adaptation performance. Experimental results show that the proposed *dynamic residual transfer* (DRT) can model domain variation in source domains (see Figure 6.2) and outperform its static counterpart (MCD [140] method) by a large margin (11.2% on DomainNet). Compared to state-of-the-art multi-source domain adaptation methods [139], it achieves a sizeable gain (3.9%) with a much simpler loss function and training algorithm.

6.2 Related Work

6.2.1 Domain Adaptation

Domain adaptation with purely single source domain approaches adapt a model from a source to a target domain. A common method is to minimize the distance between the two domains. While some methods [141, 24] minimize distance functions defined in terms of first and second order data statistics, others learn a latent space shared across domains by adversarial learning [9, 140, 142, 143]. Although these methods are effective for single-source domain distributions and relatively simple datasets (such as VisDA [144] or Office-31 [121]), they are not competitive for the multi-source domain adaptation problem, due to a more complex data distribution.

6.2.2 Multi-Source Domain Adaptation

Multi-source domain adaptation considers the domain adaptation problem when the source contains domains with a variety of styles. [145] pioneered this problem by adaptively picking the best among a set of hypothesis learned for different source domains. [146] derived an upper bound on the classification error achievable in the target domain, based on the $\mathcal{H}\Delta\mathcal{H}$ divergence. Several methods have been proposed after the introduction of deep learning. Some of these align domains pair-wise. [147] uses a discriminator to align each source domain with the target domain, while [138] matches moments across all pairs of source and target domains. These methods learn one classifier per domain and use their weighted combination to predict the class of target samples. [148] uses mutual learning techniques to align feature distributions among pairs of source and target domains. Other methods focus on the joint alignment of the feature distributions of all domains. [139] models interactions between domains with a knowledge graph. Target sample predictions are based on both their features and relationship to different domains. [149] proposes a meta-learning technique to search the best initial conditions for multi-source domain adaptation. [150] uses an auxiliary network to predict the transferability of each source sample and use it as a weight to learn a domain discriminator. All these works use a static transfer model. In this work, we propose that the model should instead be dynamic, i.e., a function that changes with samples, and show that this can significantly enhance multi-source domain adaptation.

6.2.3 Dynamic Networks

Dynamic Networks have architectures based on blocks [151, 152, 153, 154] or channels [155, 156, 157, 90] that change depending on the input sample. [151, 152] proposed an input dependent block path that decides whether a network block should be kept or dropped. [153, 154] widen the network by adding new parallel blocks and train an attention module to choose the best combination of features dynamically. [155, 156, 157, 90] rely on feature based attention

modules that reweigh features depending on the input example. [158] unified the two approaches by combining a paralleled dynamic block and a channel attention module. In this work, we propose a dynamic convolution residual branch, which adds an input-dependent residual matrix to a static kernel, to implement dynamic multi-source domain adaptation.

6.3 Method

In this section, we introduce dynamic transfer for multi-source domain adaptation, in which the model is adaptive to the domain implicitly, but adaptive to the input explicitly. It not only has better performance, but also turns multi-source domains into a single-source domain.

6.3.1 Multi-Source Domain Adaptation

Multi-source domain adaptation (MSDA) aims to transfer a model learned on a source data distribution drawn from several domains $\mathcal{S} = \{\mathcal{S}_1, ..., \mathcal{S}_N\}$ to a target domain $\mathcal{T}$. While the following ideas can be applied to various tasks, we consider a classification model $f_{\boldsymbol{\theta}}$, of parameters $\boldsymbol{\theta}$, which maps images $\boldsymbol{x} \in \mathcal{X}$ to class predictions $y \in \mathcal{Y} = \{1, \ldots, C\}$, where C is the number of classes and $\mathcal{X}$ is some image space. The goal is to adapt the parameters $\boldsymbol{\theta}$ of a model learned from a dataset $\mathcal{D}^{\mathcal{S}} = \{(\boldsymbol{x}_i^{\mathcal{S}}, \mathbf{y}_i)\}_{i=1}^{N_{\mathcal{S}}}$ of examples from the source distribution $\mathcal{S}$ ($\mathbf{y}_i$ is the one-hot encoding of the label of example $\boldsymbol{x}_i^{\mathcal{S}}$) to a dataset $\mathcal{D}^{T} = \{\boldsymbol{x}_i^{\mathcal{T}}\}_{i=1}^{N_{\mathcal{T}}}$ of unlabeled examples from the target distribution. Note that, in the most general formulation of the problem, the domain of origin of each source example, $(\boldsymbol{x}_i^{\mathcal{S}}, \mathbf{y}_i)$ is *unknown*. This is ignored by many approaches e.g. [138, 148], that assume a source dataset $\mathcal{D}^{\mathcal{S}} = \{(\boldsymbol{x}_i^{\mathcal{S}}, \mathbf{y}_i, z_i)\}_{i=1}^{N_{\mathcal{S}}}$ contains domain labels $z_i \in \{1, \ldots, N\}$ and aligning pairs of domains. We refer to this a domain supervised multi-source domain adaptation.

6.3.2 Static vs. Dynamic Transfer

The model $f_{\boldsymbol{\theta}}$ is denoted static or dynamic depending on whether the model parameters $\boldsymbol{\theta}$ vary with samples $\boldsymbol{x}$. Static models have constant parameters $\boldsymbol{\theta} = \boldsymbol{\theta}_c$, while dynamic models have parameters $\boldsymbol{\theta} = \boldsymbol{\theta}(\boldsymbol{x})$ that depend on $\boldsymbol{x}$. In the case of deep networks, this implies that layer transfer functions depend on the input $\boldsymbol{x}$. Figure 6.1 illustrates the static transfer and dynamic transfer model built for multi-source domain adaptation.

Static Transfer. Static transfer, shown on Figure 6.1(a), consists of learning of a single model $f_{\boldsymbol{\theta}_c}$ that is applied to *all* examples from source and target domains. The model might, for instance, map images into a latent space where all the distributions are aligned. Since the big variation among the input samples, this is a difficult problem and the model $f_{\boldsymbol{\theta}_c}$ usually has sub-optimal performance on all domains.

Dynamic Transfer. In this case the model parameters are a function of the input example $\boldsymbol{x}$ directly, i.e., the model has the form $f_{\boldsymbol{\theta}(\boldsymbol{x})}$ where $\boldsymbol{x} \in \mathcal{S}_1 \bigcup \cdots \bigcup \mathcal{S}_N \bigcup \mathcal{T}$. This is illustrated in Figure 6.1(b), where there exists a model per sample. Compared to the static transfer, dynamic transfer varies the model according to sample explicitly and chooses domains *implicitly,* relying on the distribution of samples $\boldsymbol{x}$. Dynamic transfer learns to adapt the parameters to fit the model to the distribution formed by the union of source domains. The target domain is not required to be aligned with any specific domains $\mathcal{S}_i$ and there are no rigid domain boundaries. The model parameters $\boldsymbol{\theta}(\boldsymbol{x})$ can be similar for examples from different domains and different for examples from the same domain.

The key insight is that adapting model per domain is achieved statistically by adapting model per sample, as each domain can be considered as a distribution of image samples. The dynamic transfer learns to adapt model parameters over samples in the union of all source domains. This simplifies the alignment between source and target domains, as it is not necessary to pull all source domains and target domain together. As long as the target domain is aligned with any part of the union of source domains, the model can be easily adapted to the target samples.

Dynamic transfer has two advantages over static transfer. First, it turns multi-source domains into a single-source domain, voiding the need for domain labels. Second, it simplifies learning, since domain labels can be arbitrary. In practice, any "domain" can contain a mixture of unlabeled sub-domains and some of these can be shared by multiple "domains". Due to this, explicit assignment of data to domains can be difficult, and models learned over single domain can loose access to shared sub-domain data.

6.3.3 Dynamic Residual Transfer

The main difficulty of dynamic transfer is the model $f_{\boldsymbol{\theta}(\boldsymbol{x})}$ can be difficult to learn. Given the large number of parameters of modern networks, it is impossible to simply predict all parameter values at inference time. The key is to restrict the model's dependence on input $\boldsymbol{x}$ to a *small number of parameters*. To guarantee this, we propose a model composed by a *static network* and *dynamic residual blocks*

$$f_{\boldsymbol{\theta}(\boldsymbol{x})} = f_0 + \Delta f_{\boldsymbol{\theta}(\boldsymbol{x})}, \tag{6.1}$$

where f_0 represents the static component and $\Delta f_{\boldsymbol{\theta}(\boldsymbol{x})}$ the dynamic residual that depends on the input sample $\boldsymbol{x}$. As usual, the residual is implemented by adding residual blocks to the various network layers. Since the static component f_0 is shared by all samples, static transfer is a special case of the proposed approach, where $\Delta f_{\boldsymbol{\theta}(\boldsymbol{x})} = 0$. This approach is denoted as *dynamic residual transfer* (DRT).

To implement DRT in convolution neural networks (CNNs), we represent a $k \times k$ convolution kernel as a $C_{out} \times C_{in}k^2$ weight matrix, where C_{in} and C_{out} are the number of input and output channels. We ignore bias terms in this discussion for the sake of brevity. DRT is implemented by applying Equation 6.1 to each convolution kernel in a CNN, i.e., defining the

network convolutions as

$$\boldsymbol{W}(\boldsymbol{x}) = \boldsymbol{W}_0 + \Delta\boldsymbol{W}(\boldsymbol{x}), \tag{6.2}$$

where $\boldsymbol{W}_0$ is a static convolution kernel matrix, and $\Delta\boldsymbol{W}(\boldsymbol{x})$ a dynamic residual matrix. We next discuss several possibilities for the latter.

Channel Attention: in this case, the residual only rescales the output channels of $\boldsymbol{W}_0$. This is implemented as

$$\Delta\boldsymbol{W}(\boldsymbol{x}) = \boldsymbol{\Lambda}(\boldsymbol{x})\boldsymbol{W}_0, \tag{6.3}$$

where $\boldsymbol{\Lambda}(\boldsymbol{x})$ is a diagonal $C_{out} \times C_{out}$ matrix, whose entries are functions of $\boldsymbol{x}$. This can be seen as a dynamic feature-based attention mechanism.

Subspace Routing: as shown in Figure 6.3, the dynamic residual is a linear combination of K static matrices $\boldsymbol{\Phi}_i$

$$\Delta\boldsymbol{W}(\boldsymbol{x}) = \sum_{i=1}^{K} \pi_i(\boldsymbol{x})\boldsymbol{\Phi}_i, \tag{6.4}$$

whose weights depend on $\boldsymbol{x}$. The matrices $\boldsymbol{\Phi}_i$ can be seen as a basis for CNN weight space, although they are not necessarily linearly independent. And the dynamic coefficients $\pi_i(\boldsymbol{x})$ can be seen as the projections of the residual matrix in the corresponding weight subspaces. By choosing these projections in an input dependent manner, the network chooses different feature subspaces to route different $\boldsymbol{x}$.

To reduce the number of parameters and computation, the matrices can be further simplified into 1×1 convolution kernels and applied to the narrowest layer of the bottleneck architecture in ResNet [55]. In this case, only C_{in} rows of $\boldsymbol{\Phi}_i$ are non-zero.

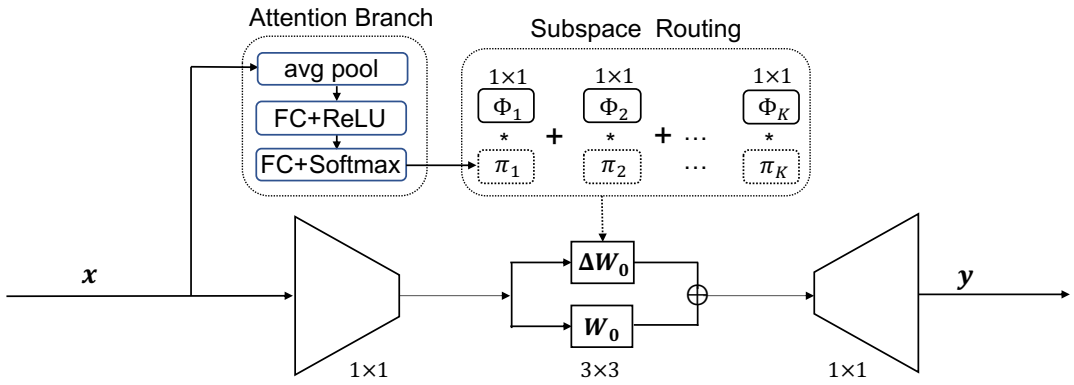


Figure 6.3: Subspace routing of DRT: dynamic coefficients are generated by a dynamic branch given the input $\boldsymbol{x}$. Each dynamic coefficient $\pi_i(\boldsymbol{x})$ is then multiplied by a matrix $\boldsymbol{\Phi}_i$, and the K matrices are aggregated as the residual kernel- $\boldsymbol{\Delta W}_0(\boldsymbol{x})$.

Combination: the two mechanisms are combined into

$$\Delta \boldsymbol{W}(\boldsymbol{x}) = \boldsymbol{\Lambda}(\boldsymbol{x})\boldsymbol{W}_0 + \sum_{i=1}^{K} \pi_i(\boldsymbol{x})\boldsymbol{\Phi}_i. \tag{6.5}$$

Similar to squeeze-and-excitation block [96], the dynamic coefficients $\boldsymbol{\Lambda}(\boldsymbol{x})$ and $\{\pi_i(\boldsymbol{x})\}$ are implemented by a light-weight attention branch that includes average pooling and two fully connected layers (See Figure 6.3). A sigmoid is used to normalize $\boldsymbol{\Lambda}(\boldsymbol{x})$ and a softmax to normalize $\{\pi_i(\boldsymbol{x})\}$. As explained by [154, 153], the extra FLOPs caused by dynamic coefficient generation and residual aggregation of dynamic transfer is negligible (less than 0.1% in our implementation) compared to the static model.

6.3.4 Learning

As usual for domain adaptation problems, the DRT network is learned with a combination of two losses,

$$\mathcal{L} = \mathcal{L}_{ce} + \lambda \mathcal{L}_d, \tag{6.6}$$

where λ is a hyperparameter that controls the trade-off between the loss components. The first loss

$$\mathcal{L}_{ce} = \frac{1}{N_{\mathcal{S}}} \sum_{i=1}^{N_{\mathcal{S}}} \mathbf{y}_i^T \log f_{\boldsymbol{\theta}(\boldsymbol{x}_i^{\mathcal{S}})}(\boldsymbol{x}_i^{\mathcal{S}}), \tag{6.7}$$

is the cross entropy loss over the source data $\mathcal{D}^{\mathcal{S}}$. The second is a domain alignment loss that encourages the minimization of the distance between source and target domains

$$\mathcal{L}_d = \mathcal{H}\left(f_{\boldsymbol{\theta}(\mathcal{D}^{\mathcal{S}})}(\mathcal{D}^{\mathcal{S}}), f_{\boldsymbol{\theta}(\mathcal{D}^{\mathcal{T}})}(\mathcal{D}^{\mathcal{T}})\right), \tag{6.8}$$

where $\mathcal{D}^{\mathcal{T}}$ is the target data and $\mathcal{H}$ a measure of discrepancy between feature distributions of the source and target domains. $\mathcal{H}$ can be any distance function previously proposed for domain adaptation, e.g. the MMD [22] or adversarial learning [9]. Note that the two losses above operate on the entire source dataset $\mathcal{D}^{\mathcal{S}}$, i.e., there is no need for domain labels and not even a difference between the single domain and multiple domains adaptation problems. For the domain alignment losses commonly used in multi-source domain adaptation, Equation 6.8 also does require the evaluation of pairwise distances between all source domains and target domain, which is not necessary in dynamic transfer.

6.4 Experiments

In this section, adaptation performance of DRT is evaluated on the tasks of multi-source domain adaptation.

6.4.1 Datasets and Experimental Settings

Following [138], we consider two datasets, Digit-five and DomainNet [138], which contain images from several domains but shared classes. Each domain is alternatively used as the

target domain and the remaining ones as the source domain. All experiments are repeated with 5 times and mean and variance are reported.

Digit-five. Digit-five contains digit images from 5 domains: MNIST [114] (mt), Synthetic [25] (sy), MNIST-M [25] (mm), SVHN [62] (sv) and USPS [25] (up). These domains contribute $25,000$ images for training and 9000 for validation, with the exception of USPS which uses 29752 and 1860, respectively. Since these datasets are relatively small, LeNet [114] is used as the backbone model. A dynamic residual is added on each convolutional layer. The model is trained from scratch with initial learning rate 0.002 and SGD optimizer. The learning rate is decayed by 0.1 every 100 epochs and decreased to $2e-5$ in 300 epochs.

DomainNet. DomainNet [138] is a dataset with 0.6 million images of 345 classes from 6 domains of different image styles: clipart (clp), infograph (inf), painting (pnt), quickdraw (qdr), real (rel) and sketch (skt). Results are obtained with ImageNet [3] pretrained ResNet-101 [55]. The dynamic residual is only added on the 3×3 kernel of each bottleneck block. The networks are trained with SGD for 15 epochs with initial learning rate of 0.001 and batch size as 64. The learning rate is decayed by 0.1 every 5 epochs.

6.4.2 Ablation Study

An ablation study was performed on DomainNet to evaluate the three key components of DRT: (a) the three implementations of the dynamic transfer, (b) the number of basis used for subspace routing (Equation 6.4), and (c) different alignment losses $\mathcal{L}_d$. The default model uses subspace routing with $K = 4$ and is trained with the MCD [140] loss.

DRT Implementations. Table 6.1 shows that all implementations of DRT have significantly better adaptation performance than the static model. The average gains are of 9.3% for channel attention, 10.8% for combined and 11.2% for subspace routing. The weaker performance of channel attention suggests that it is not enough to re-scale the features of the static model. Routing the input $\boldsymbol{x}$ through different subspaces appears to be more effective, although the differ-

Table 6.1: Comparison of **different implementations for dynamic residual transfer**: Channel Attention (Equation 6.3), Subspace Routing (Equation 6.4) and Combination (Equation 6.5).

Models	inf,pnt,qdr rel,skt → clp	clp,pnt,qdr rel,skt → inf	clp,inf,qdr rel,skt → pnt	clp,inf,pnt rel,skt → qdr	clp,inf,pnt qdr,skt → rel	clp,inf,pnt qdr,rel → skt	Avg
static	54.3±0.64	22.1±0.70	45.7±0.63	7.6±0.49	58.4±0.65	43.5±0.57	38.5±0.61
Channel Attention	67.8±0.46	30.9±0.85	57.1±0.36	6.9±1.12	66.7±0.42	57.4±0.33	47.8±0.59
Subspace Routing	**69.7**±0.24	31.0±0.56	**59.5**±0.43	9.9±1.03	**68.4**±0.28	**59.4**±0.21	**49.7**±0.46
Combination	69.1±0.35	**31.6**±0.61	58.2±0.25	**11.9**±0.96	67.8±0.36	58.8±0.44	49.6±0.50

ences are not staggering. While combining the two approaches has no additional overall benefit, the combination was beneficial for specific transfer problems. When 'infograph' and 'quickdraw' were used as target domains, the combination model outperformed subspace routing. Since these are the hardest transfer problems, this suggests that the enhanced dynamics of the combined implementation can be beneficial as the domain gap increases. It is because the enhanced dynamics make the model more elastic. Therefore, it is more likely to adapt models to target domain with larger gap. On the other hand, for the problems of smaller domain gap, stronger dynamics can cause the model to overfit to the source domain, as is the case for the remaining target domains. More experiments on datasets with more domains will likely be needed to resolve this question. In any case, subspace routing and the combination model have similar performance.

Number of Residual Basis. The impact of the number of basis K used in Equation 6.4 for subspace routing is ablated. For different values of $K \in \{2,4,6,8\}$, DRT achieves the adaptation performance by $\{48.8, 49.7, 49.5, 49.3\}$, all of which improve the adaptation performance of static transfer (38.5%) by a large margin (more than 10%). Best performance is achieved with $K = 4$, although the results are not highly sensitive to this parameter.

Alignment Loss Function. Three different domain alignment losses with different forms of $\mathcal{H}$ (see Equation 6.8) were compared: ADDA [9], MCD [140] and M^3SDA [138]. They are representative of previously proposed losses for reducing single-source domain shift at the domain level and class level, and multi-source domain shift, respectively.

Table 6.2 shows that dynamic residual transfer (DRT) outperforms static transfer for all

Table 6.2: **Static transfer vs. Dynamic transfer** evaluated on DomainNet with different domain alignment loss functions ('Src Only' refers to 'Source Only').

$\mathcal{L}_d$	inf,pnt,qdr rel,skt → clp	clp,pnt,qdr rel,skt → inf	clp,inf,qdr rel,skt → pnt	clp,inf,pnt rel,skt → qdr	clp,inf,pnt qdr,skt → rel	clp,inf,pnt qdr,rel → skt	Avg
Src Only	52.1±0.51	23.4±0.28	47.7±0.96	**13.0**±0.72	60.7±0.23	46.5±0.56	40.6±0.56
Src Only + **DRT**	**63.1**±0.62	**25.9**±0.84	**48.4**±1.02	6.4±0.98	**66.4**±0.54	**46.8**±0.44	**42.8**±0.74
ADDA [9]	47.5±0.76	11.4±0.67	36.7±0.53	**14.7**±0.50	49.1±0.82	33.5±0.49	32.2±0.63
ADDA+**DRT**	**63.6**±0.52	**27.6**±0.43	**52.3**±0.68	8.2±1.44	**67.9**±0.42	**49.6**±0.33	**44.9**±0.64
MCD [140]	54.3±0.64	22.1±0.70	45.7±0.63	7.6±0.49	58.4±0.65	43.5±0.57	38.5±0.61
MCD+**DRT**	**69.7**±0.24	**31.0**±0.56	**59.5**±0.43	**9.9**±1.03	**68.4**±0.28	**59.4**±0.21	**49.7**±0.46
M^3SDA-β [138]	58.6±0.53	26.0±0.89	52.3±0.55	6.3±0.58	62.7±0.51	49.5±0.76	42.6±0.64
M^3SDA-β+**DRT**	**67.4**±0.52	**31.3**±0.83	**56.5**±0.67	**13.6**±0.34	**66.9**±0.42	**56.8**±0.49	**48.8**±0.55

loss functions, by a large margin (12.7%, 11.2%, 6.2% respectively). Its improved performance is in part, due to the fact that DRT takes a much larger advantage of the domain alignment losses. It confirms our claim that DRT simplifies the domain alignment by unifying all source domains into a single domain. Thus the target samples are more likely to be aligned with the union of source domains and the same alignment loss will give more benefits to the dynamic model than the static one. However, the gains over the 'source only' case, where no alignment loss $\mathcal{L}_d$ is used in Equation 6.6, is only 2%. It means alignment loss is very critical for dynamic transfer. Without alignment loss, even though the model can adapt to the entire source domain very well, it can hardly adapt to target samples due to a large domain gap.

These conclusions also apply to the individual transfer problems, except when 'quickdraw' is the target domain. In this case, DRT is only effective with the M^3SDA-β [138] loss. It is because when 'quickdraw' is the target domain, the domain discrepancy is much larger and makes it harder for DRT to adapt model to this domain. Thus, M^3SDA-β [138] which proposed a more powerful alignment loss, can shift 'quickdraw' closer to source domains and works better with DRT. However, the strong alignment loss will cause 'over-alignment' for the domains e.g. 'clipart' that have much smaller gap. The 'over-alignment' reduces the adaptability of the dynamic model, which causes performance degradation compared to that given by simpler alignment losses e.g. MCD.

Table 6.3: Comparison between **dynamic residual transfer (DRT)** with the state-of-the-art models on Digit-five dataset. The source domains and target domain are shown at the top of each column.

Models	mm,up,sv sy → mt	mt,up,sv sy → mm	mt,mm,sv sy → up	mt,mm,up sy → sv	mt,mm,up sv → sy	Avg
Source Only	63.37±0.74	90.50±0.83	88.71±0.89	63.54±0.93	82.44±0.65	77.71±0.81
DANN [159]	71.30±0.56	97.60±0.75	92.33±0.85	63.48±0.79	85.34±0.84	82.01±0.76
ADDA [9]	71.57±0.52	97.89±0.84	92.83±0.74	75.48±0.48	86.45±0.62	84.84±0.64
MCD [140]	72.50±0.67	96.21±0.81	95.33±0.74	78.89±0.78	87.47±0.65	86.10±0.73
DCTN [147]	70.53±1.24	96.23±0.82	92.81±0.27	77.61±0.41	86.77±0.78	84.79±0.72
M^3SDA-β [138]	72.82±1.13	98.43±0.68	96.14±0.81	81.32±0.86	89.58±0.56	87.65±0.75
CMSS [150]	75.3±0.57	99.0±0.08	97.7±0.13	**88.4**±0.54	93.7±0.21	90.8±0.31
DRT	**81.03**±0.34	**99.31**±0.05	**98.40**±0.12	86.67±0.38	**93.89**±0.34	**91.86**±0.25

6.4.3 Comparisons to the State-of-the-Art

DRT was compared to the results in the literature for Digit Five and DomainNet dataset. In these experiments, DRT is implemented with subspace routing (4 basis), using the MCD loss [140], and $\lambda = 50$ in Equation 6.6.

Evaluation on Digit Five Dataset. Table 6.3 shows a comparison to 6 baselines on Digit Five. DRT outperforms all other methods, beating the state of the art (CMSS) by more than 1%, despite a much simpler implementation. Comparing performance in individual adaptation problems, DRT has the best performance on four of the five problems considered. The only exception occurs when SVHN is the target domain, where DRT achieves the second best performance of all methods. Beyond this, the smallest gains occur when MNIST is the target domain. This was expected, since MNIST is easier to transfer to and somewhat saturated. In general, the gains of DRT increase with domain discrepancy, reaching 5.7% for the hardest transfer problem (MNIST-M as target domain).

Evaluation on DomainNet Dataset. For DomainNet [138], a ResNet-101 [55] was used as backbone and DRT was compared to 11 baselines. Among these, ADDA [9], DANN [159] and MCD [140] were developed for traditional unsupervised domain adaptation (UDA), where a single-source domain is assumed. The remaining are multi-source domain adaptation methods

Table 6.4: Comparison between **dynamic residual transfer (DRT)** with the state-of-the-art models on DomainNet. ('DRT+ST' represents the combination between dynamic residual transfer and self-training for domain adaptation)

Models	inf,pnt,qdr rel,skt → clp	clp,pnt,qdr rel,skt → inf	clp,inf,qdr rel,skt → pnt	clp,inf,pnt rel,skt → qdr	clp,inf,pnt qdr,skt → rel	clp,inf,pnt qdr,rel → skt	Avg
Source Only	52.1±0.51	23.4±0.28	47.7±0.96	13.0±0.72	60.7±0.23	46.5±0.56	40.6±0.56
ADDA [9]	47.5±0.76	11.4±0.67	36.7±0.53	14.7±0.50	49.1±0.82	33.5±0.49	32.2±0.63
MCD [140]	54.3±0.64	22.1±0.70	45.7±0.63	7.6±0.49	58.4±0.65	43.5±0.57	38.5±0.61
DANN [159]	60.6±0.42	25.8±0.43	50.4±0.51	7.7±0.68	62.0±0.66	51.7±0.19	43.0±0.46
DCTN [147]	48.6±0.73	23.5±0.59	48.8±0.63	7.2±0.46	53.5±0.56	47.3±0.47	38.2±0.57
M^3SDA-β [138]	58.6±0.53	26.0±0.89	52.3±0.55	6.3±0.58	62.7±0.51	49.5±0.76	42.6±0.64
ML-MSDA [148]	61.4±0.79	26.2±0.41	51.9±0.20	**19.1**±0.31	57.0±1.04	50.3±0.67	44.3±0.24
Meta-MCD [149]	62.8±0.22	21.4±0.07	50.5±0.08	15.5±0.22	64.6±0.16	50.4±0.12	44.2±0.07
LtC-MSDA [139]	63.1±0.5	28.7±0.7	56.1±0.5	16.3±0.5	66.1±0.6	53.8±0.6	47.4±0.6
CMSS [150]	64.2±0.18	28.0±0.20	53.6±0.39	16.0±0.12	63.4±0.21	53.8±0.35	46.5±0.24
DRT	69.7±0.24	31.0±0.56	59.5±0.43	9.9±1.03	68.4±0.28	59.4±0.21	49.7±0.46
DRT+ST	**71.0**±0.21	**31.6**±0.44	**61.0**±0.32	12.3±0.38	**71.4**±0.23	**60.7**±0.31	**51.3**±0.32

that require domain labels.

Table 6.4 shows that DRT improves on the state-of-the-art method- CMSS by more than 3% (49.7% vs 46.5%). When DRT is combined with a naive self-training method (DRT+ST), it achieves gains of 3.9% over LtC-MSDA [139], a methods that generates pseudo-labels for the target samples (during self-training, pseudo-labels for target samples with confidence greater than 0.8 are used to train DRT again with source samples). Compared to the adaptation methods that use no domain labels (single-source), DRT improves the best average adaptation results (DANN) by 6.7% (49.7% vs. 43%).

Regarding individual adaptation problems, DRT achieves the best performance for all target domains other than 'quickdraw'. This can be explained by the large domain gap between 'quickdraw' and the other domains, and the fact that the MCD loss does not fare well in this problem. Better results would likely be possible by using the M^3SDA loss, as was the case in Table 6.2.

Table 6.5: **Single source domain adaptation performance on DomainNet**. Each column, shows the average/best classification accuracy for transfer from all source to the specified target domain.

Models	inf,pnt,qdr rel,skt → clp	clp,pnt,qdr rel,skt → inf	clp,inf,qdr rel,skt → pnt	clp,inf,pnt rel,skt → qdr	clp,inf,pnt qdr,skt → rel	clp,inf,pnt qdr,rel → skt	Avg
ADDA [9]	28.2/39.5	9.3/14.5	20.1/29.1	**8.4/14.9**	31.1/41.9	21.7/30.7	19.8/28.4
MCD [140]	31.4/42.6	13.1/19.6	24.9/42.6	2.2/3.8	35.7/50.5	23.9/33.8	21.9/32.2
DRT	**41.9/56.2**	**19.6/26.6**	**35.3/53.4**	8.0/12.2	**44.5/55.5**	**35.0/44.8**	**30.7/41.5**

6.4.4 Single-Source to Single-Target Adaptation

The performance of DRT on the traditional domain adaptation problem (single-source domain) was also evaluated on DomainNet [138]. In this case, for each target domain, adaptation was performed from each of the other five domains (sources). The average and best performance among these adaptations is shown in Table 6.5, for each target domain. DRT again significantly outperforms the previous domain adaptation methods. For example, when 'clipart' is the target domain, its average adaptation performance is 10.5% better than that of the MCD method. On average, over all pairs of source and target domains, it outperforms MCD by more than 8%. These results show that, for problems with hundreds of classes, dynamic residual transfer can lead to very large adaptation gains even in the traditional domain adaptation setting. This confirms the claim that even these problems tend to have many sub-domains. When this is the case, the ability of dynamic residual transfer to adapt the model on a per-example basis can be a significant asset.

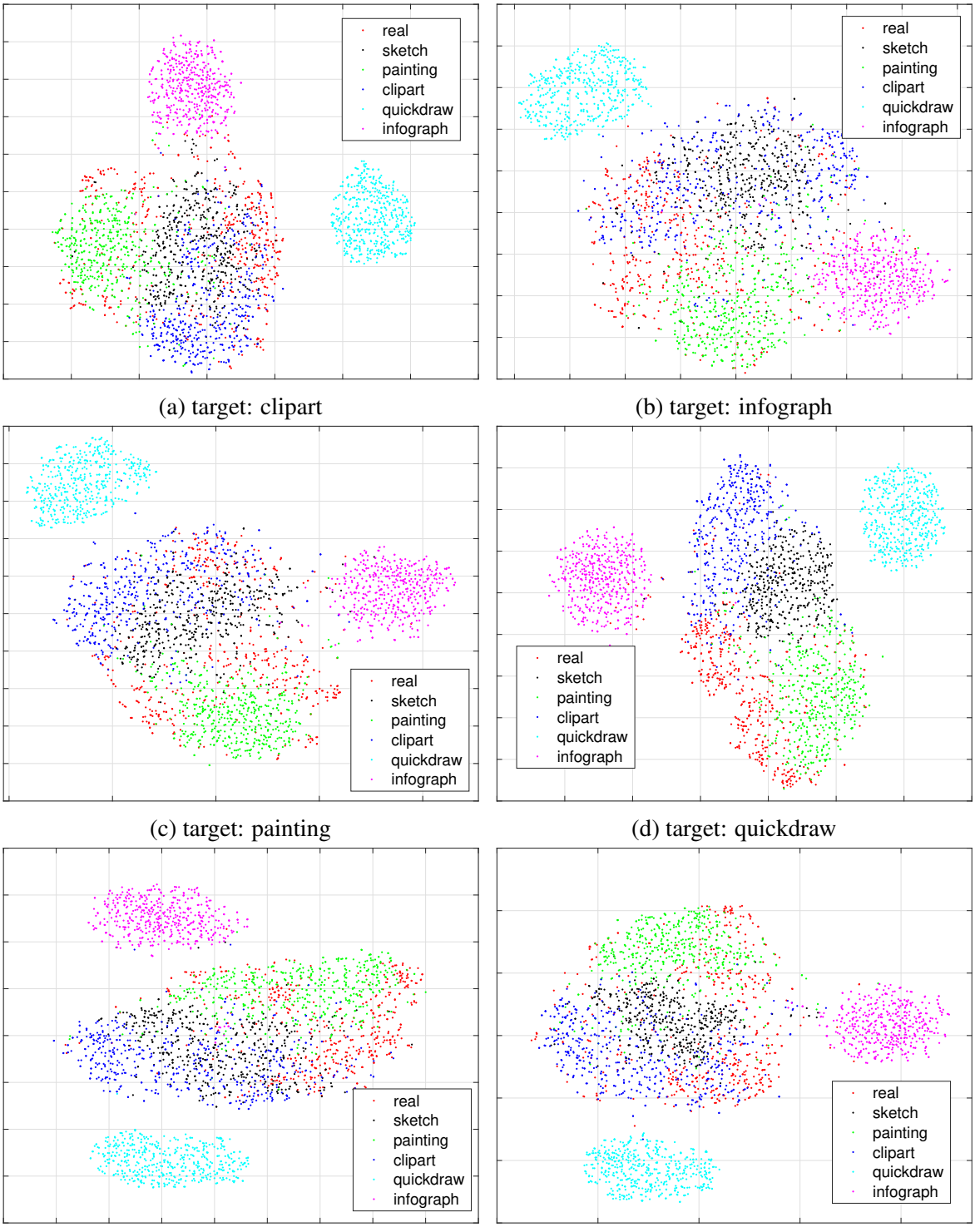
Finally, comparing the results of Tables 6.4 and 6.5 shows that DRT trained on multi-source domains performs 8.2% better (49.7% vs. 41.5%) than the average of the best single-source domain transfers. This improvement is about 2% better than that given by MCD (8.2% vs. 6.3%). This shows that considering a variety of source domains improves domain adaptation performance, especially when dynamic residual transfer is used. A main advantage of DRT is that it can be applied to all settings, since it does not require domain labels. Its universal nature makes it irrelevant if the problem is single-source or multi-source. It suffices to collect training data and DRT will automatically figure out how to adapt the network to all settings. There is no need to

even define "source domains."

6.4.5 Visualization

To obtain further insight about dynamic residual transfer (DRT), we visualized the dynamic coefficients of Equation 6.4 with t-SNE [160]. For each sample, we created a vector $\Pi = \{\pi_i^l(\boldsymbol{x})\}, i \in \{1,2,...K\}, l \in \{1,2,...L\}$ by concatenating the dynamic coefficients from the L network layers. The vectors Π from different target domains are visualized in Figure 6.4. A more detailed visualization is given in Figure 6.5, by splitting Π into Π_{low} and Π_{high}, which include the coefficients from lower and higher network layers. For brevity, Figure 6.5, only visualizes the model trained with 'clipart' and 'real' as target domain and the other domains show similar trend.

Figure 6.4 first shows that domain information is embedded into the dynamic coefficients $\{\pi_i^l(\boldsymbol{x})\}$. This can be observed by samples from same domains tend to group in identifiable clusters, which confirms our claim that adapting model across domains can be achieved through adapting model to samples. Secondly, the distance among clusters reflects domain shifts that explain how adaptation performance varies with target domain. For example, the fact that the dynamic coefficients of 'quickdraw' are always quite different from others, explains why adaptation performance is weaker when this is the target domain. Thus for 'quickdraw', either a more powerful alignment loss is needed to shift the samples close enough to the source domains to enable the dynamic model adapted to this domain or a more complex dynamic model e.g. combination of 'channel attention' and 'subspace routing' is required. Figure 6.5 further shows that the dynamic coefficients from lower layers (Figure 6.5(a)) form much more clear domain clusters than the coefficients of the higher layers (Figure 6.5(b)). This shows that the network features become more domain agnostic in the higher layers, confirming the effectiveness of DRT to reduce domain discrepancies.



(a) target: clipart

(b) target: infograph

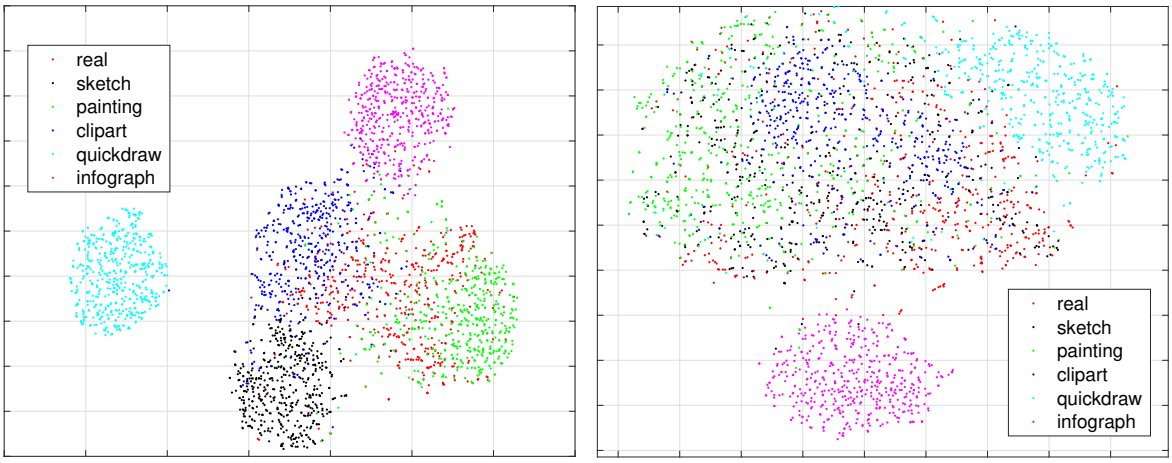
(c) target: painting

(d) target: quickdraw

(e) target: real

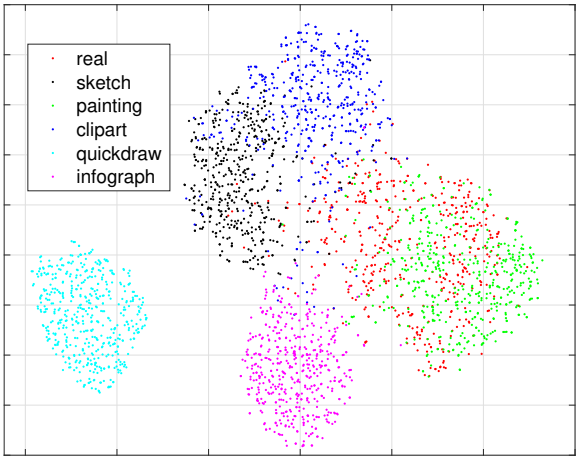
(f) target: sketch

Figure 6.4: The t-SNE **visualization of dynamic coefficients** $\Pi = \{\pi_i^l(\boldsymbol{x})\}$ when DRT is trained with target domain- 'clipart', 'infograph', 'painting', 'quickdraw', 'real' and 'sketch'. (Best view in color)

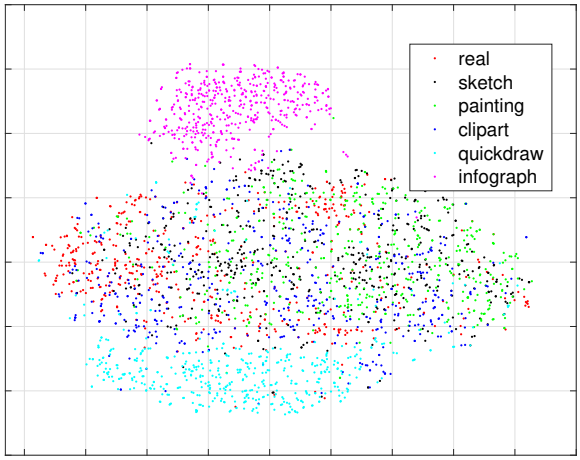


(a) target: clipart (Π_{low})

(b) target: clipart (Π_{high})



(c) target: real (Π_{low})



(d) target: real (Π_{high})

Figure 6.5: The t-SNE visualization of **first half** (Π_{low}) and **second half** (Π_{high}) dynamic coefficients when DRT is trained with target domain- 'clpart' and 'real'. (Best view in color)

6.5 Conclusion

In this chapter, we discussed a novel domain adaptation task, i.e., multi-source domain adaptation. We introduced dynamic transfer for multi-source domain adaptation, in which the model parameters are not static but adaptive to input samples. Dynamic transfer mitigates conflicts across domains and unifies multiple source domains into a single source domain, thereby simplifying the alignment between source and target domains. Experimental results show that

aided by adversarial training, dynamic transfer achieves a better adaptation performance compared to the state-of-the-art methods for multi-source domain adaptation and the performance can be further promoted with self-training. We hope this work can give a new understanding about the role played by dynamic networks on multi-source domain adaptation.

Chapter 6 is, in full, based on the material as it appears in the publication of "Dynamic Transfer for Multi-Source Domain Adaptation", Yunsheng Li, Lu Yuan, Yinpeng Chen, Pei Wang and Nuno Vasconcelos, in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 2021. The dissertation author was the primary investigator and author of this material.

Chapter 7

Discussion and Conclusion

In the thesis, we studied a series of computer vision tasks under two types of resource restriction, i.e., the limitation of computing power for machines and the lack of data annotation. In order to overcome these issues, we mainly focused on the design of efficient neural networks and the development of novel adaptive frameworks via exploring variants of the dynamic network based on its two key features, i.e., efficiency and adaptiveness.

First, we discussed the design of dynamic networks for efficient networks. We start by introducing a domain-wise dynamic adapter- CovNorm that is associated with each convolution layer to convert the convolution operation from static to dynamic. We found, by only replacing the dynamic adapter according to each dataset, the whole framework can be fitted to different datasets efficiently by consuming tiny amount of extra computation and parameters. Then we extended the proposed CovNorm to the large scale image recognition task. By changing CovNorm from the manner of domain-wise to sample-wise via inducing some sample-dependent parameters, a new method, i.e., dynamic convolution decomposition (DCD) was derived. It follows the architecture of CovNorm but making the mini-adaptation layer dynamic to samples. With DCD, samples no longer share the same feature extractor as well as the classifier but are able to choose the most appropriate network parameters. Therefore the models' representation capacity is significantly improved, especially for the ones under low FLOPs. Extensive experiments on large scale image recognition tasks demonstrated the effectiveness of the proposed DCD and further confirmed the power of dynamic blocks built on the convolution layers. In the last, we investigated the network design with extremely low FLOPs by reducing the availability of computational resources to a new regime of 4-20 MFLOPs and proposed a new framework, i.e., MicroNet. We first redesigned the convolution layer by proposing an efficient Micro-Factorized convolution. Compared to the Micro-Factorized convolution, the dynamic blocks, i.e., CovNorm and DCD that have been applied on convolution layers are no longer efficient. Therefore, we proposed a novel dynamic block- Dynamic Shift-Max, which is an activation function to improve the non-linearity of the network. By combining Micro-Factorized convolution and Dynamic Shift-Max, MicroNet shows

huge success from the view of accuracy, FLOPs and latency on various image recognition tasks.

Second, we studied the domain adaptation tasks and further discussed the role played by dynamic networks in this new scenario. We first worked on the domain adaptive semantic segmentation problem and proposed a novel framework, i.e., Bidirection Learning (BDL). BDL achieved a good performance on the target dataset even though no annotation is available and it revealed several key techniques, i.e., adversarial learning and self-training, that can be leveraged on the path to the success of domain adaptation. Then based on BDL, we moved forward to a novel multi-source domain adaptation problem, where the source domain is a mixture of several sub-domains. In order to align the domain discrepancy existing both among source domains and between source domains and the target domain, we modified DCD and proposed a novel dynamic module, i.e., Dynamic Residual Transfer (DRT). Experiments showed that DRT was a very generic method that can be added to diverse domain alignment frameworks and outperformed its corresponding baseline by a large margin.

In summary, the success of deep learning has enabled an excellent performance on extensive computer vision tasks and the performance is being promoted as long as enough computational resources and data annotation are assumed. However, this hypothesis is not always true due to the shortage of resources is a common issue and this resource shortage has formed a big challenge to prevent the real application of deep learning. In the thesis, we worked on this challenge and tried to overcome it via inducing several novel designs of dynamic networks from the forms of domain-wise to sample-wise and from acting as a convolution operator to an activation function. We demonstrated, through a series of experiments, the dynamic network can boost the neural network's representation capacity efficiently even though it is trained with limited annotated data. We hope our work can motivate people to further explore the power of the dynamic network and scale it to more vision tasks, e.g., 3D or temporal-spatial computer vision, or even beyond the field of computer vision, e.g., natural language processing.

Bibliography

[1] B. Yang, G. Bender, Q. V. Le, and J. Ngiam, "Condconv: Conditionally parameterized convolutions for efficient inference," in *NeurIPS*, 2019.

[2] Y. Chen, X. Dai, M. Liu, D. Chen, L. Yuan, and Z. Liu, "Dynamic convolution: Attention over convolution kernels," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255, IEEE, 2009.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[5] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE international conference on computer vision*, pp. 5058–5066, 2017.

[6] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *arXiv preprint arXiv:1810.05270*, 2018.

[7] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[8] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 19–34, 2018.

[9] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," in *Computer Vision and Pattern Recognition (CVPR)*, vol. 1, p. 4, 2017.

[10] J. Hoffman, D. Wang, F. Yu, and T. Darrell, "Fcns in the wild: Pixel-level adversarial and constraint-based adaptation," *arXiv preprint arXiv:1612.02649*, 2016.

[11] Y. Zou, Z. Yu, B. V. Kumar, and J. Wang, "Unsupervised domain adaptation for semantic segmentation via class-balanced self-training," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 289–305, 2018.

[12] M. Xu, J. Zhang, B. Ni, T. Li, C. Wang, Q. Tian, and W. Zhang, "Adversarial domain adaptation with domain mixup," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 6502–6509, 2020.

[13] X. Mao, Y. Ma, Z. Yang, Y. Chen, and Q. Li, "Virtual mixup training for unsupervised domain adaptation," *arXiv preprint arXiv:1905.04215*, 2019.

[14] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," *arXiv preprint arXiv:1511.06530*, 2015.

[15] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 1389–1397, 2017.

[16] M. Long, J. Wang, G. Ding, J. Sun, and P. S. Yu, "Transfer feature learning with joint distribution adaptation," in *Proceedings of the IEEE international conference on computer vision*, pp. 2200–2207, 2013.

[17] S.-A. Rebuffi, H. Bilen, and A. Vedaldi, "Learning multiple visual domains with residual adapters," in *Advances in Neural Information Processing Systems*, pp. 506–516, 2017.

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

[19] S.-A. Rebuffi, H. Bilen, and A. Vedaldi, "Efficient parametrization of multi-domain deep neural networks," *arXiv preprint arXiv:1803.10082*, 2018.

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

[21] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[22] M. Long, Y. Cao, J. Wang, and M. I. Jordan, "Learning transferable features with deep adaptation networks," *International Conference in Machine Learning*, 2015.

[23] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan, "Domain separation networks," in *Advances in Neural Information Processing Systems*, pp. 343–351, 2016.

[24] B. Sun and K. Saenko, "Deep coral: Correlation alignment for deep domain adaptation," in *European Conference on Computer Vision*, pp. 443–450, Springer, 2016.

[25] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation," *International Conference in Machine Learning*, 2014.

[26] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[27] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan, "Unsupervised pixel-level domain adaptation with generative adversarial networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, p. 7, 2017.

[28] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training.," in *CVPR*, vol. 2, p. 5, 2017.

[29] J. Hoffman, E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. A. Efros, and T. Darrell, "Cycada: Cycle-consistent adversarial domain adaptation," *arXiv preprint arXiv:1711.03213*, 2017.

[30] F. M. Carlucci, L. Porzi, B. Caputo, E. Ricci, and S. R. Bulò, "Autodial: Automatic domain alignment layers.," in *ICCV*, pp. 5077–5085, 2017.

[31] M. Mancini, L. Porzi, S. R. Bulò, B. Caputo, and E. Ricci, "Boosting domain adaptation by discovering latent domains," *arXiv preprint arXiv:1805.01386*, 2018.

[32] R. Caruana, "Multitask learning," in *Learning to learn*, pp. 95–133, Springer, 1998.

[33] Y. Zhang and Q. Yang, "A survey on multi-task learning," *arXiv preprint arXiv:1707.08114*, 2017.

[34] R. Girshick, "Fast r-cnn," *arXiv preprint arXiv:1504.08083*, 2015.

[35] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: towards real-time object detection with region proposal networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[36] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2650–2658, 2015.

[37] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, "Cross-stitch Networks for Multi-task Learning," in *CVPR*, 2016.

[38] Z. Zhang, P. Luo, C. C. Loy, and X. Tang, "Facial landmark detection by deep multi-task learning," in *European Conference on Computer Vision*, pp. 94–108, Springer, 2014.

[39] R. Ranjan, V. M. Patel, and R. Chellappa, "Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[40] G. Gkioxari, R. Girshick, and J. Malik, "Contextual action recognition with r* cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 1080–1088, 2015.

[41] J. Huang, R. S. Feris, Q. Chen, and S. Yan, "Cross-domain image retrieval with a dual attribute-aware ranking network," in *Proceedings of the IEEE international conference on computer vision*, pp. 1062–1070, 2015.

[42] P. Morgado and N. Vasconcelos, "Semantically consistent regularization for zero-shot recognition," in *CVPR*, vol. 9, p. 10, 2017.

[43] Y. Lu, A. Kumar, S. Zhai, Y. Cheng, T. Javidi, and R. S. Feris, "Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification.," in *CVPR*, vol. 1, p. 6, 2017.

[44] A. Kendall, Y. Gal, and R. Cipolla, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," *arXiv preprint arXiv:1705.07115*, vol. 3, 2017.

[45] I. Kokkinos, "Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory.," in *CVPR*, vol. 2, p. 8, 2017.

[46] A. R. Zamir, A. Sax, W. Shen, L. Guibas, J. Malik, and S. Savarese, "Taskonomy: Disentangling task transfer learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3712–3722, 2018.

[47] B. Jou and S.-F. Chang, "Deep cross residual learning for multitask visual recognition," in *Proceedings of the 2016 ACM on Multimedia Conference*, pp. 998–1007, ACM, 2016.

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

[49] S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang, "Overcoming catastrophic forgetting by incremental moment matching," in *Advances in Neural Information Processing Systems*, pp. 4655–4665, 2017.

[50] A. R. Triki, R. Aljundi, M. B. Blaschko, and T. Tuytelaars, "Encoder based lifelong learning," *IEEE Conference Computer Vision and Pattern Recognition*, 2017.

[51] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "icarl: Incremental classifier and representation learning," in *Proc. CVPR*, 2017.

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

[53] R. Aljundi, P. Chakravarty, and T. Tuytelaars, "Expert gate: Lifelong learning with a network of experts.," in *CVPR*, pp. 7120–7129, 2017.

[54] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[55] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[56] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba, "Sun database: Large-scale scene recognition from abbey to zoo," in *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pp. 3485–3492, IEEE, 2010.

[57] M. Valenti, B. Bethke, D. Dale, A. Frank, J. McGrew, S. Ahrens, J. P. How, and J. Vian, "The mit indoor multi-vehicle flight testbed," in *Robotics and Automation, 2007 IEEE International Conference on*, pp. 2758–2759, IEEE, 2007.

[58] S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi, "Fine-grained visual classification of aircraft," *arXiv preprint arXiv:1306.5151*, 2013.

[59] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *Computer Vision, Graphics & Image Processing, 2008. ICVGIP'08. Sixth Indian Conference on*, pp. 722–729, IEEE, 2008.

[60] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[61] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," 2007.

[62] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS workshop on deep learning and unsupervised feature learning*, vol. 2011, p. 5, 2011.

[63] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[64] A. Rosenfeld and J. K. Tsotsos, "Incremental learning through deep adaptation," *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[65] A. Mallya, D. Davis, and S. Lazebnik, "Piggyback: Adapting a single network to multiple tasks by learning to mask weights," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 67–82, 2018.

[66] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[67] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[68] Z. Su, L. Fang, W. xiong Kang, D. Hu, M. Pietikäinen, and L. Liu, "Dynamic group convolution for accelerating convolutional neural networks," in *ECCV*, August 2020.

[69] J.-R. Chen, X. Wang, Z. Guo, X. Zhang, and J. Sun, "Dynamic region-aware convolution," *ArXiv*, vol. abs/2003.12243, 2020.

[70] N. Ma, X. Zhang, J. Huang, and J. Sun, "Weightnet: Revisiting the design space of weight networks," vol. abs/2007.11823, 2020.

[71] Z. Tian, C. Shen, and H. Chen, "Conditional convolutions for instance segmentation," in *ECCV*, August 2020.

[72] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

[73] A. Howard, M. Sandler, G. Chu, L. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," *CoRR*, vol. abs/1905.02244, 2019.

[74] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *The European Conference on Computer Vision (ECCV)*, September 2018.

[75] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *ICML*, (Long Beach, California, USA), pp. 6105–6114, 09–15 Jun 2019.

[76] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[77] M. Tan and Q. V. Le, "Mixconv: Mixed depthwise convolutional kernels," in *30th British Machine Vision Conference 2019*, 2019.

[78] H. Chen, Y. Wang, C. Xu, B. Shi, C. Xu, Q. Tian, and C. Xu, "Addernet: Do we really need multiplications in deep learning?," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[79] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, "Ghostnet: More features from cheap operations," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[80] D. Zhou, Q.-B. Hou, Y. Chen, J. Feng, and S. Yan, "Rethinking bottleneck structure for efficient mobile network design," in *ECCV*, August 2020.

[81] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," in *International Conference on Learning Representations*, 2019.

[82] H. Cai, C. Gan, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," *ArXiv*, vol. abs/1908.09791, 2019.

[83] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," *arXiv preprint arXiv:1412.6553*, 2014.

[84] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in neural information processing systems*, pp. 1269–1277, 2014.

[85] J. Kossaifi, A. Toisoul, A. Bulat, Y. Panagakis, T. M. Hospedales, and M. Pantic, "Factorized higher-order cnns with an application to spatio-temporal emotion estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6060–6069, 2020.

[86] A.-H. Phan, K. Sobolev, K. Sozykin, D. Ermilov, J. Gusak, P. Tichavsky, V. Glukhov, I. Oseledets, and A. Cichocki, "Stable low-rank tensor decomposition for compression of convolutional neural network," *arXiv preprint arXiv:2008.05441*, 2020.

[87] D. Ha, A. M. Dai, and Q. V. Le, "Hypernetworks," *ICLR*, 2017.

[88] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[89] X. Li, W. Wang, X. Hu, and J. Yang, "Selective kernel networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[90] Y. Chen, X. Dai, M. Liu, D. Chen, L. Yuan, and Z. Liu, "Dynamic relu," *arXiv preprint arXiv:2003.10027*, vol. abs/2003.10027, 2020.

[91] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines.," in *ICML*, 2010.

[92] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?," in *The IEEE International Conference on Computer Vision (ICCV)*, 2009.

[93] L. D. Lathauwer, B. D. Moor, and J. Vandewalle, "A multilinear singular value decomposition," in *SIAM J. Matrix Anal. Appl*, 2000.

[94] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[95] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," in *International Conference on Learning Representations*, 2018.

[96] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *CoRR*, vol. abs/1602.07360, 2016.

[97] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[98] K. A. vahid, A. Prabhu, A. Farhadi, and M. Rastegari, "Butterfly transform: An efficient fft based neural architecture design," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[99] K. Han, Y. Wang, Q. Zhang, W. Zhang, C. XU, and T. Zhang, "Model rubikas cube: Twisting resolution, depth and width for tinynets," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 19353–19364, Curran Associates, Inc., 2020.

[100] D. Li, A. Zhou, and A. Yao, "Hbonet: Harmonious bottleneck on two orthogonal dimensions," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3316–3325, 2019.

[101] P. Gibson, J. Cano, J. Turner, E. J. Crowley, M. O'Boyle, and A. Storkey, "Optimizing grouped convolutions on edge devices," in *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 189–196, 2020.

[102] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, pp. 740–755, Springer, 2014.

[103] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *arXiv preprint arXiv:1506.01497*, 2015.

[104] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017.

[105] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

[106] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, "Detectron2." `https://github.com/facebookresearch/detectron2`, 2019.

[107] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[108] B. Xiao, H. Wu, and Y. Wei, "Simple baselines for human pose estimation and tracking," in *European conference on computer vision*, 04 2018.

[109] K. Sun, B. Xiao, D. Liu, and J. Wang, "Deep high-resolution representation learning for human pose estimation," in *CVPR*, 2019.

[110] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

[111] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *European Conference on Computer Vision*, pp. 102–118, Springer, 2016.

[112] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3234–3243, 2016.

[113] H. T. Vu and C.-C. Huang, "Domain adaptation meets disentangled representation learning and style transfer.," *CoRR*, 2017.

[114] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[115] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*, vol. 1. Springer series in statistics New York, NY, USA:, 2001.

[116] Y. Zhang, P. David, and B. Gong, "Curriculum domain adaptation for semantic segmentation of urban scenes," in *The IEEE International Conference on Computer Vision (ICCV)*, vol. 2, p. 6, 2017.

[117] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," *arXiv preprint*, 2017.

[118] Z. Wu, X. Han, Y.-L. Lin, M. G. Uzunbas, T. Goldstein, S. N. Lim, and L. S. Davis, "Dcan: Dual channel-wise alignment networks for unsupervised scene adaptation," *arXiv preprint arXiv:1804.05827*, 2018.

[119] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[120] V. M. Patel, R. Gopalan, R. Li, and R. Chellappa, "Visual domain adaptation: A survey of recent advances," *IEEE signal processing magazine*, vol. 32, no. 3, pp. 53–69, 2015.

[121] K. Saenko, B. Kulis, M. Fritz, and T. Darrell, "Adapting visual category models to new domains," in *European conference on computer vision*, pp. 213–226, Springer, 2010.

[122] B. Geng, D. Tao, and C. Xu, "Daml: Domain adaptation metric learning," *IEEE Transactions on Image Processing*, vol. 20, no. 10, pp. 2980–2989, 2011.

[123] Q. Chen, Y. Liu, Z. Wang, I. Wassell, and K. Chetty, "Re-weighted adversarial adaptation network for unsupervised domain adaptation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7976–7985, 2018.

[124] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla, "Segmentation and recognition using structure from motion point clouds," in *ECCV (1)*, pp. 44–57, 2008.

[125] Y.-H. Tsai, W.-C. Hung, S. Schulter, K. Sohn, M.-H. Yang, and M. Chandraker, "Learning to adapt structured output space for semantic segmentation," *arXiv preprint arXiv:1802.10349*, 2018.

[126] F. S. Saleh, M. S. Aliakbarian, M. Salzmann, L. Petersson, and J. M. Alvarez, "Effective use of synthetic data for urban scene semantic segmentation," in *European Conference on Computer Vision*, pp. 86–103, Springer, Cham, 2018.

[127] M.-Y. Liu, T. Breuel, and J. Kautz, "Unsupervised image-to-image translation networks," in *Advances in Neural Information Processing Systems*, pp. 700–708, 2017.

[128] X. Huang, M.-Y. Liu, S. Belongie, and J. Kautz, "Multimodal unsupervised image-to-image translation," *arXiv preprint arXiv:1804.04732*, 2018.

[129] A. Dundar, M.-Y. Liu, T.-C. Wang, J. Zedlewski, and J. Kautz, "Domain stylization: A strong, simple baseline for synthetic to real image domain adaptation," *arXiv preprint arXiv:1807.09384*, 2018.

[130] D. He, Y. Xia, T. Qin, L. Wang, N. Yu, T. Liu, and W.-Y. Ma, "Dual learning for machine translation," in *Advances in Neural Information Processing Systems*, pp. 820–828, 2016.

[131] X. Niu, M. Denkowski, and M. Carpuat, "Bi-directional neural machine translation with synthetic parallel data," *arXiv preprint arXiv:1805.11213*, 2018.

[132] S. Pontes-Filho and M. Liwicki, "Bidirectional learning for robust neural networks," *arXiv preprint arXiv:1805.08006*, 2018.

[133] P. Russo, F. M. Carlucci, T. Tommasi, and B. Caputo, "From source to target and back: symmetric bi-directional adaptive gan," *arXiv preprint arXiv:1705.08824*, vol. 3, 2017.

[134] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2018.

[135] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[136] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, p. 3, 2013.

[137] Y. Luo, L. Zheng, T. Guan, J. Yu, and Y. Yang, "Taking a closer look at domain shift: Category-level adversaries for semantics consistent domain adaptation," *arXiv preprint arXiv:1809.09478*, 2018.

[138] X. Peng, Q. Bai, X. Xia, Z. Huang, K. Saenko, and B. Wang, "Moment matching for multi-source domain adaptation," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1406–1415, 2019.

[139] H. Wang, M. Xu, B. Ni, and W. Zhang, "Learning to combine: Knowledge aggregation for multi-source domain adaptation," *arXiv preprint arXiv:2007.08801*, 2020.

[140] K. Saito, K. Watanabe, Y. Ushiku, and T. Harada, "Maximum classifier discrepancy for unsupervised domain adaptation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3723–3732, 2018.

[141] F. M. Cariucci, L. Porzi, B. Caputo, E. Ricci, and S. R. Bulo, "Autodial: Automatic domain alignment layers," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 5077–5085, IEEE, 2017.

[142] J. Hoffman, E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. Efros, and T. Darrell, "Cycada: Cycle-consistent adversarial domain adaptation," in *International conference on machine learning*, pp. 1989–1998, PMLR, 2018.

[143] Y. Li, L. Yuan, and N. Vasconcelos, "Bidirectional learning for domain adaptation of semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6936–6945, 2019.

[144] X. Peng, B. Usman, N. Kaushik, J. Hoffman, D. Wang, and K. Saenko, "Visda: The visual domain adaptation challenge," *arXiv preprint arXiv:1710.06924*, 2017.

[145] J. Yang, R. Yan, and A. G. Hauptmann, "Cross-domain video concept detection using adaptive svms," in *Proceedings of the 15th ACM international conference on Multimedia*, pp. 188–197, 2007.

[146] J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. Wortman, "Learning bounds for domain adaptation," in *Advances in neural information processing systems*, pp. 129–136, 2008.

[147] R. Xu, Z. Chen, W. Zuo, J. Yan, and L. Lin, "Deep cocktail network: Multi-source unsupervised domain adaptation with category shift," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3964–3973, 2018.

[148] Z. Li, Z. Zhao, Y. Guo, H. Shen, and J. Ye, "Mutual learning network for multi-source domain adaptation," *arXiv preprint arXiv:2003.12944*, 2020.

[149] D. Li and T. Hospedales, "Online meta-learning for multi-source and semi-supervised domain adaptation," *arXiv preprint arXiv:2004.04398*, 2020.

[150] L. Yang, Y. Balaji, S.-N. Lim, and A. Shrivastava, "Curriculum manager for source selection in multi-source domain adaptation," *arXiv preprint arXiv:2007.01261*, 2020.

[151] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime neural pruning," in *Advances in Neural Information Processing Systems*, pp. 2181–2191, 2017.

[152] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, and R. Feris, "Blockdrop: Dynamic inference paths in residual networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8817–8826, 2018.

[153] B. Yang, G. Bender, Q. V. Le, and J. Ngiam, "Condconv: Conditionally parameterized convolutions for efficient inference," in *Advances in Neural Information Processing Systems*, pp. 1307–1318, 2019.

[154] Y. Chen, X. Dai, M. Liu, D. Chen, L. Yuan, and Z. Liu, "Dynamic convolution: Attention over convolution kernels," *arXiv preprint arXiv:1912.03458*, 2019.

[155] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141, 2018.

[156] Y. Cao, J. Xu, S. Lin, F. Wei, and H. Hu, "Gcnet: Non-local networks meet squeeze-excitation networks and beyond," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 0–0, 2019.

[157] X. Wang, Z. Cai, D. Gao, and N. Vasconcelos, "Towards universal object detection by domain attention," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7289–7298, 2019.

[158] Y. Li, Y. Chen, X. Dai, M. Liu, D. Chen, Y. Yu, L. Yuan, Z. Liu, M. Chen, and N. Vasconcelos, "Revisiting dynamic convolution via matrix decomposition," *arXiv preprint arXiv:2103.08756*, 2021.

[159] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation," in *International conference on machine learning*, pp. 1180–1189, PMLR, 2015.

[160] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.